ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αλγόριθμοι αναζήτησης σε βάσεις βιολογικών δεδομένων και υλοποίηση σε Java

ΓΑΒΡΑΣ ΠΕΤΡΟΣ

Εξεταστική επιτροπή: **Μ. Κουμπαράκης (Επιβλέπων)**
Δ. Πνευματικάτος
Β. Σαμολαδάς

ΧΑΝΙΑ, ΙΟΥΛΙΟΣ 2004

Ευχαριστίες

Στους καθηγητές κυρίους Μ. Κουμπαράκη, Δ. Πνευματικάτο, Β. Σαμολαδά
για τη συνεργασία τους σε αυτή την εργασία

Στη Γεωργία Αδαμοπούλου
για την πολύτιμη βοήθεια σε κάθε στάδιο της εκπόνησης αυτής της εργασίας

ΠΕΡΙΛΗΨΗ

Η αλματώδης ανάπτυξη της μοριακής βιολογίας έχει οδηγήσει στη δημιουργία ενός τεράστιου όγκου πληροφοριών σχετικά με το αυτό το επιστημονικό αντικείμενο. Έτσι λοιπόν γεννιέται το πρόβλημα της αποδοτικής αποθήκευσης αυτών των δεδομένων, αλλά και της αποδοτικής ανάκτησης και επεξεργασίας τους, έτσι ώστε να χρησιμοποιηθούν ως βάση για νέες γνώσεις. Σε αυτά τα προβλήματα καλείται να δώσει λύση ο κλάδος της υπολογιστικής βιολογίας, ο οποίος συνδυάζει τις έννοιες της μοριακής βιολογίας με αυτές της επιστήμης υπολογιστών.

Ένα πρόβλημα της υπολογιστικής βιολογίας που παρουσιάζει ιδιαίτερο ενδιαφέρον είναι η αναζήτηση σε βάσεις δεδομένων στις οποίες αποθηκεύονται πληροφορίες σχετικά με βιολογικές ακολουθίες, για ακολουθίες που παρουσιάζουν ομοιότητες. Με αυτό τον τρόπο οι μοριακοί βιολόγοι μπορούν να εντοπίζουν συγγένειες μεταξύ ακολουθιών, να τις ταξινομούν κατά οικογένειες και να προβλέπουν τις ιδιότητες και τη λειτουργία τους με βάση τις συγγενικές ακολουθίες.

Για το πρόβλημα της αναζήτησης στις βάσεις βιολογικών δεδομένων, έχουν προταθεί αλγόριθμοι οι οποίοι συγκρίνουν ακολουθίες μεταξύ τους, χρησιμοποιώντας κάποιο μέτρο ομοιότητας. Οι αλγόριθμοι αυτοί παρουσιάζουν σημαντικές διαφορές μεταξύ τους, κυρίως ως προς τη βασική φιλοσοφία τους, γεγονός όμως που επηρεάζει και την απόδοσή τους. Για την ανάπτυξη λοιπόν ενός συστήματος το οποίο θα έχει επαρκή απόδοση (κυρίως ως προς την ταχύτητα) θα πρέπει να εξεταστεί με πολύ προσοχή η επιλογή του κατάλληλου αλγόριθμου. Σε αυτή την εργασία εξετάζουμε αυτούς ακριβώς τους αλγόριθμους και την απόδοσή τους, προχωρώντας στη συνέχεια στην υλοποίηση εκείνου που κρίνεται αποτελεσματικότερος και ταχύτερος.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

Ευρετήριο Σχημάτων

Ευρετήριο Πινάκων

ΕΙΣΑΓΩΓΗ

Ι. Η ανάπτυξη της υπολογιστικής μοριακής βιολογίας

Ένας από τους επιστημονικούς τομείς που τα τελευταία χρόνια έχει παρουσιάσει αλματώδη ανάπτυξη είναι η μοριακή βιολογία. Η ανάπτυξη αυτή έχει οδηγήσει σε αύξηση των δυνατοτήτων των βιολόγων να μελετούν και να επεξεργάζονται βιομοριακές δομές. Χαρακτηριστικά παραδείγματα αποτελούν η έρευνα επάνω στο γενετικό υλικό των ζωντανών οργανισμών, όπως αυτό αποθηκεύεται στο DNA, η οποία οδηγεί και στην ανάπτυξη νέων κλάδων της βιολογίας όπως η γενετική μηχανική και άλλες. Αν αναλογιστεί επιπλέον κανείς ότι η δομή του DNA αποκαλύφθηκε για πρώτη φορά μόλις το 1953, τότε η ανάπτυξη της μοριακής βιολογίας φαντάζει ακόμα πιο εκπληκτική.

Η ταχύτατη ανάπτυξη του τομέα της μοριακής βιολογίας σε συνδυασμό με τις διευρυμένες δυνατότητες που προσφέρει στους ερευνητές η σύγχρονη τεχνολογία, έχουν οδηγήσει στην δημιουργία μεγάλων ποσοτήτων πληροφορίας για τις βιομοριακές δομές και λειτουργίες. Οι ποσότητες αυτών των πληροφοριών συνεχίζουν (και θα συνεχίσουν) να αυξάνονται και μάλιστα με επιταχυνόμενο ρυθμό. Όλες αυτές οι πληροφορίες όμως γενούν την ανάγκη για αποδοτική αποθήκευση και επεξεργασία τους, καθώς οι ίδιες αυτές πληροφορίες αποτελούν πηγή για περαιτέρω επιστημονική έρευνα. Για την αντιμετώπιση αυτής της ανάγκης αναπτύχθηκε ένας τομέας που συνδυάζει της έννοιες της μοριακής βιολογίας με της έννοιες των μαθηματικών και της επιστήμης υπολογιστών. Ο τομέας αυτός ονομάζεται υπολογιστική μοριακή βιολογία.

Αν και η υπολογιστική μοριακή βιολογία υπήρχε για αρκετά χρόνια στο περιθώριο της βιολογίας, η ουσιαστική ανάπτυξή της και η αξιοποίηση των δυνατοτήτων που προσφέρει ξεκίνησε με την ανάπτυξη και της μοριακής βιολογίας και κυρίως με την αύξηση των πληροφοριών που προέρχονταν από την έρευνα. Η υπολογιστική μοριακή βιολογία καλείται να δώσει λύσεις σε προβλήματα της μοριακής βιολογίας χρησιμοποιώντας τις αρχές και τις τεχνικές της επιστήμης υπολογιστών. Τα προβλήματα αυτά ξεκινούν από την ανάγκη αποθήκευσης των διαθέσιμων πληροφοριών της μοριακής βιολογίας, και εκτείνονται μέχρι την επεξεργασία των πληροφοριών που προκύπτουν άμεσα από την έρευνα.

Την ανάπτυξη αυτή ήρθε να ενισχύσει και η αλματώδης πρόοδος στην τεχνολογία των υπολογιστών που συντελείται τις τελευταίες δεκαετίες. Η βελτίωση των χαρακτηριστικών τους, τόσο σε υπολογιστική ισχύ όσο και σε αποθηκευτική ικανότητα έχει βοηθήσει στην επίλυση πολλών προβλημάτων. Έτσι, με την αύξηση των διαθέσιμων αποθηκευτικών μέσων είναι δυνατή η αποθήκευση των μεγάλων ποσοτήτων πληροφοριών που υπάρχουν διαθέσιμες για τις βιομοριακές δομές όπως το DNA και οι πρωτεΐνες. Παράλληλα, η αύξηση της υπολογιστικής ισχύος επιτρέπει τη χρήση πολύπλοκων αλγορίθμων για την επεξεργασία των διαθέσιμων δεδομένων. Παρ' όλα αυτά, η ανάγκη για την ανάπτυξη αποδοτικών αλγορίθμων παραμένει, καθώς ο όγκος των δεδομένων που καλούνται να επεξεργαστούν είναι μεγάλος.

Χαρακτηριστικό παράδειγμα προβλήματος της μοριακής βιολογίας που απαιτεί το σχεδιασμό αποδοτικού αλγορίθμου είναι το πρόβλημα της σύγκρισης βιολογικών ακολουθιών. Έχοντας δηλαδή δύο ή περισσότερες ακολουθίες που αντιστοιχούν σε βιολογικές μοριακές δομές, να βρούμε πόσο όμοιες είναι μεταξύ τους. Αν και το

πρόβλημα δεν είναι ιδιαίτερα απαιτητικό αν μελετηθεί μεμονωμένο, η πρακτική εφαρμογή του αποτελεί πρόκληση. Αυτό συμβαίνει γιατί στην πράξη το πρόβλημα θα πρέπει να επιλυθεί πάρα πολλές φορές, στον μικρότερο πάντοτε δυνατό χρόνο. Θα πρέπει δηλαδή είτε να συγκριθούν πολλά (ίσως και αρκετές χιλιάδες) ζεύγη ακολουθιών είτε να συγκριθούν πολλές (και πάλι ίσως χιλιάδες) ακολουθίες μεταξύ τους.

ΙΙ. Η αναζήτηση σε βάσεις βιολογικών δεδομένων

Όπως είδαμε και στο πρώτο μέρος της εισαγωγής, ένα από τα προβλήματα που καλείται να αντιμετωπίσει η υπολογιστική μοριακή βιολογία είναι αυτό της αποθήκευσης των πληροφοριών που παράγονται. Ένα θέμα της μοριακής βιολογίας σχετικά με τη μελέτη και επεξεργασία μορίων DNA και πρωτεϊνών, είναι η αποθήκευση των πληροφοριών που προκύπτουν για τη δομή και τη λειτουργία των μορίων αυτών. Για την αντιμετώπιση της ανάγκης αυτής δημιουργήθηκαν πολλές βάσεις δεδομένων, οι οποίες αυξάνονται σε μέγεθος, αριθμό και πληροφορίες παράλληλα με την ανάπτυξη της μοριακής βιολογίας.

Εκτός όμως από την αποδοτική αποθήκευση των πληροφοριών, είναι απαραίτητη και μια αποδοτική μέθοδος ανάκτησής τους, έτσι ώστε να μπορούν να αποτελέσουν αντικείμενο επεξεργασίας και βάση για μελλοντική έρευνα. Συνδυάζοντας την ανάγκη για αποδοτική ανάκτηση πληροφορίας με το πρόβλημα της σύγκρισης που αναφέραμε παραπάνω, προκύπτει το πρόβλημα της αναζήτησης σε βάσεις βιολογικών δεδομένων. Το πρόβλημα αυτό συνδυάζει τους και τους δύο παράγοντες που κάνουν το πρόβλημα σύγκρισης απαιτητικό. Από τη μία, η βάση βιολογικών δεδομένων μπορεί να περιέχει εκατοντάδες χιλιάδες ακολουθίες, με τις οποίες πρέπει να συγκριθεί η ακολουθία που εξετάζουμε. Από την άλλη, συναντάται συχνά στην πράξη η ανάγκη για επανάληψη της διαδικασίας αναζήτησης για πολλές ακολουθίες.

Έτσι λοιπόν γίνεται αντιληπτό ότι το πρόβλημα της αναζήτησης σε βάσεις βιολογικών δεδομένων παρουσιάζει ιδιαίτερο ενδιαφέρον, καθώς είναι ιδιαίτερα απαιτητικό ως προς τον αλγόριθμο που θα χρησιμοποιηθεί για τη λύση του. Σε αυτή την εργασία επιλέξαμε να ασχοληθούμε με αυτό ακριβώς το πρόβλημα, εξετάζοντας αλγόριθμους που έχουν προταθεί για την επίλυσή του και προχωρώντας στην υλοποίηση του αποδοτικότερου.

ΙΙΙ. Δομή της εργασίας

Η δομή της εργασίας αυτής ακολουθεί το παρακάτω πλάνο:

Στο Κεφάλαιο 1, *Βασικές έννοιες της μοριακής βιολογίας*, παρουσιάζονται συνοπτικά ορισμένες έννοιες και δίνεται έμφαση σε εκείνα τα στοιχεία και τα χαρακτηριστικά τους που θα μας απασχολήσουν και στη συνέχεια.

Στο Κεφάλαιο 2, *Υπολογιστική βιολογία και βάσεις βιολογικών δεδομένων*, περιγράφονται αναλυτικά οι απαραίτητες έννοιες της υπολογιστικής βιολογίας που σχετίζονται με το πρόβλημα που μελετάμε, καθώς επίσης και ορισμένα στοιχεία για τις βάσεις βιολογικών δεδομένων με τις οποίες θα ασχοληθούμε.

Στο Κεφάλαιο 3, *Αλγόριθμοι σύγκρισης βιολογικών ακολουθιών*, παρουσιάζονται αναλυτικά αλγόριθμοι που έχουν προταθεί για την επίλυση του προβλήματος της σύγκρισης ακολουθιών.

Στο Κεφάλαιο 4, *Ο αλγόριθμος BLAST*, περιγράφουμε αναλυτικά τη δομή και τη λειτουργία του αλγόριθμου που επιλέξαμε να υλοποιήσουμε, κρίνοντας ότι είναι ο αποδοτικότερος.

Στο Κεφάλαιο 5, *Υλοποίηση*, περιγράφουμε αναλυτικά το σύστημα το οποίο σχεδιάσαμε για την υλοποίηση του αλγόριθμου BLAST.

Στο Κεφάλαιο 6, *Εφαρμογή: το σύστημα JavaBLAST*, αναφερόμαστε στην εφαρμογή που σχεδιάσαμε και χρησιμοποιεί την υλοποίηση για να εκτελέσει μια αναζήτηση σε μια βάση βιολογικών δεδομένων.

Τέλος, στο Κεφάλαιο 7, *Συμπεράσματα*, συνοψίζουμε τα συμπεράσματά μας για το πρόβλημα της αναζήτησης και τους αλγόριθμους επίλυσής του και παρουσιάζουμε τις προτάσεις για μελλοντική εργασία επάνω στο θέμα.

ΚΕΦΑΛΑΙΟ 1

Βασικές έννοιες της μοριακής βιολογίας

Στο κεφάλαιο αυτό παρουσιάζουμε κάποιες από της βασικές έννοιες και αρχές της μοριακής βιολογίας. Η παρουσίαση αυτή κατέχει το ρόλο βοηθήματος προς τους αναγνώστες που δεν έχουν τις κατάλληλες γνώσεις σε θέματα βιολογίας, έτσι ώστε να αποκτήσουν το απαραίτητο, για την κατανόηση αυτής της εργασίας, γνωστικό υπόβαθρο. Οι έννοιες παρουσιάζονται αρκετά συνοπτικά και δίνεται μεγαλύτερη έμφαση στα στοιχεία εκείνα που θα μας απασχολήσουν και στη συνέχεια. Έτσι επικεντρώνουμε το ενδιαφέρον μας κυρίως στη δομή των πρωτεϊνών και των νουκλεϊκών οξέων DNA και RNA, καθώς επίσης και στο μηχανισμό «μετάφρασης» του DNA.

1.1. Πρωτεΐνες

1.1.1. Η σημασία των πρωτεϊνών

Πρόκειται για τις ουσίες που αποτελούν το μεγαλύτερο μέρος του σώματος κάθε ζωντανού οργανισμού, μετά βέβαια από το νερό. Τα είδη τους διαφέρουν σημαντικά και σε κάθε ζωντανό οργανισμό υπάρχουν πολλές χιλιάδες διαφορετικά πρωτεϊνικά μόρια, που αναλαμβάνουν τις διάφορες ζωτικές λειτουργίες των κυττάρων.
Ενδεικτικά, οι πρωτεΐνες χωρίζονται ανάλογα με τη λειτουργία που επιτελούν στα παρακάτω είδη [5]:

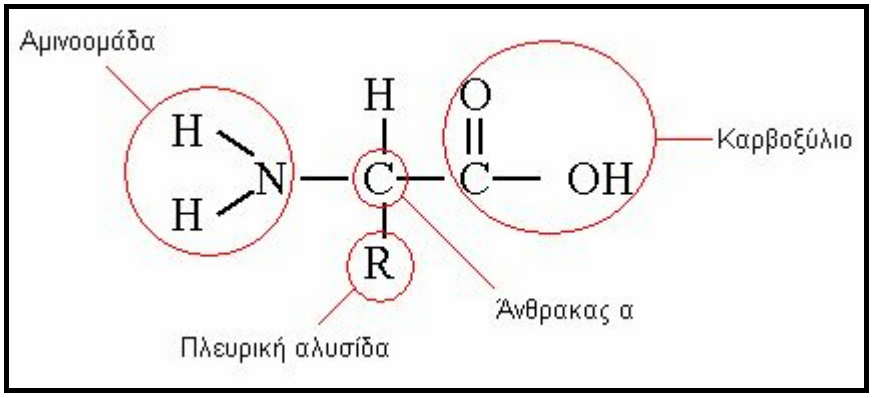
- **Ένζυμα.** Λειτουργούν ως επιταχυντές των χημικών αντιδράσεων στα κύτταρα (βιοκαταλύτες). Είναι γνωστά χιλιάδες διαφορετικά είδη ενζύμων.
- **Ορμόνες.** Μεταφέρουν πληροφορίες στον οργανισμό και παίζουν σημαντικό ρόλο στη ρύθμιση πολλών λειτουργιών του οργανισμού.
- **Ινώδεις πρωτεΐνες.** Παρουσιάζουν μεγάλη ικανότητα αντίστασης ως δομικά και σκελετικά συστατικά. Αποτελούν τους θεμελιώδεις σχηματισμούς των μυϊκών κυττάρων, των συνδετικών ιστών, του δέρματος, των μαλλιών, των νυχιών.
- **Συσταλτικές πρωτεΐνες.** Καθιστούν δυνατές τις κινήσεις καθώς οι πρωτεϊνικές ίνες συστέλλονται. Είναι υπεύθυνες για τις κινήσεις των κυττάρων και αποτελούν τα σημαντικότερα δομικά τμήματα των μυϊκών κυττάρων.
- **Μεταφορικές πρωτεΐνες.** Αναλαμβάνουν τη μεταφορά διαφόρων ουσιών μέσα στον οργανισμό (π.χ. η μεταφορά του οξυγόνου από την αιμοσφαιρίνη).

- **Αποταμιευτικές πρωτεΐνες.** Σχηματίζουν πρωτεϊνικό απόθεμα (π.χ. σε φυτικούς βολβούς, αυγά).
- **Προστατευτικές πρωτεΐνες.** Αναγνωρίζουν τις ξένες πρωτεΐνες (π.χ. ιοί, βακτήρια) και προκαλούν την καταστροφή τους.

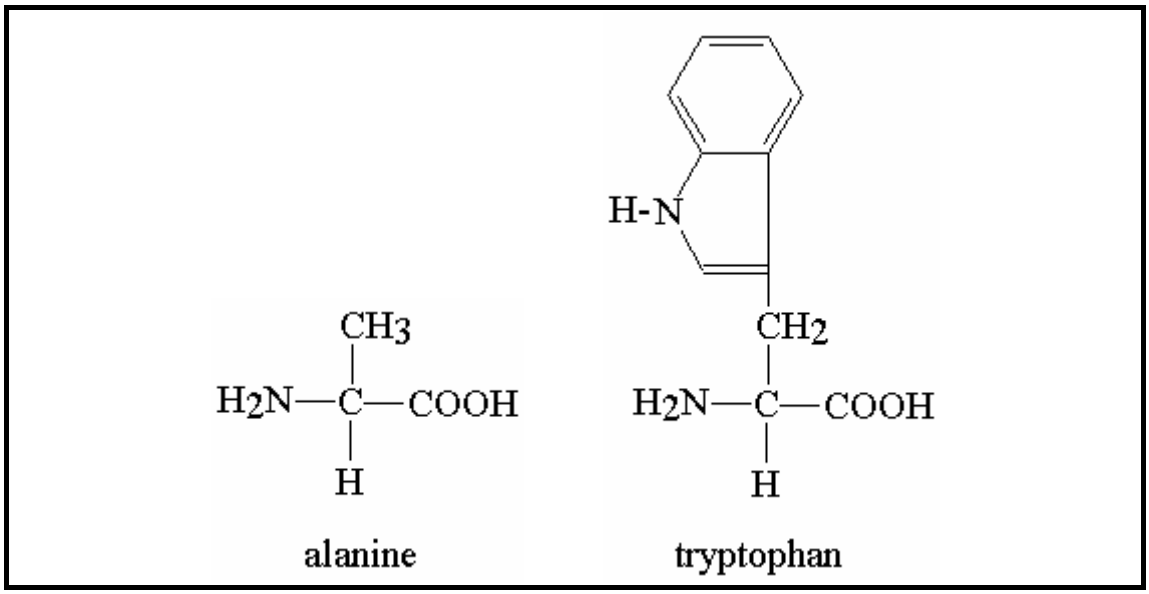
Η εξέχουσα σημασία των πρωτεϊνών φαίνεται και από το γεγονός ότι στο γενετικό υλικό κάθε οργανισμού με το DNA (που περιγράφεται αναλυτικά στην Ενότητα 1.2) αποθηκεύονται πληροφορίες που έχουν σχέση αποκλειστικά με τη δομή των πρωτεϊνών. Όλα τα άλλα συστατικά και οι αντιδράσεις στα κύτταρα είναι διατεταγμένα μετά από τις πρωτεΐνες και εξαρτώνται από αυτές.

1.1.2. Η δομή των πρωτεϊνών

Κάθε πρωτεΐνη είναι μια αλυσίδα από απλούστερες χημικές ενώσεις που ονομάζονται αμινοξέα. Αν και είναι γνωστά από τη χημεία εκατοντάδες αμινοξέα, οι πρωτεΐνες όλων των ζωντανών οργανισμών αποτελούνται μόνο από 20 διαφορετικά αμινοξέα. Τα αμινοξέα έχουν ως δομική βάση ένα άτομο άνθρακα, το οποίο ονομάζεται άνθρακας άλφα, ή C_α. Στο άτομο αυτό ενώνονται ένα άτομο υδρογόνου, μια αμινοομάδα (NH_2), ένα καρβοξύλιο (COOH), και μια πλευρική αλυσίδα (R) [5]. Η δομή αυτή των αμινοξέων φαίνεται στο Σχήμα 1.1. Η πλευρική αλυσίδα είναι αυτή που καθορίζει και διαχωρίζει τα αμινοξέα μεταξύ τους. Παραδείγματα αμινοξέων έχουμε στο Σχήμα 1.2.



Σχήμα 1.1: Ο γενικός τύπος των αμινοξέων.



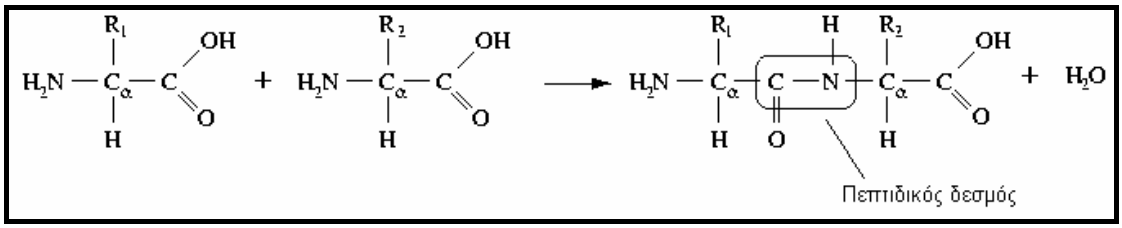
Σχήμα 1.2: Παραδείγματα αμινοξέων.

Για τα 20 αμινοξέα που σχηματίζουν της πρωτεΐνες χρησιμοποιούνται συχνά για ευκολία συντμήσεις τριών γραμμάτων ή και σύμβολα. Τα ονόματα των αμινοξέων και οι συντμήσεις αυτές φαίνονται στον Πίνακα 1.1 [3].

Πίνακας 1.1: Τα αμινοξέα που σχηματίζουν τις πρωτεΐνες, συντμήσεις και σύμβολα.

Ονομασία	Σύντμηση	Σύμβολο
Αλανίνη	Ala	A
Κυστεΐνη	Cys	C
Ασπαρτικό οξύ	Asp	D
Γλουταμικό οξύ	Glu	E
Φενυλαλανίνη	Phe	F
Γλυκίνη	Gly	G
Ιστιδίνη	His	H
Ισολευκίνη	Ile	I
Λυσίνη	Lys	K
Λευκίνη	Leu	L
Μεθιονίνη	Met	M
Ασπαραγγίνη	Asn	N
Προλίνη	Pro	P
Γλουταμίνη	Gln	Q
Αργινίνη	Arg	R
Σερίνη	Ser	S
Θρεονίνη	Thr	T
Βαλίνη	Val	V
Τρυπτοφάνη	Trp	W
Τυροσίνη	Tyr	Y

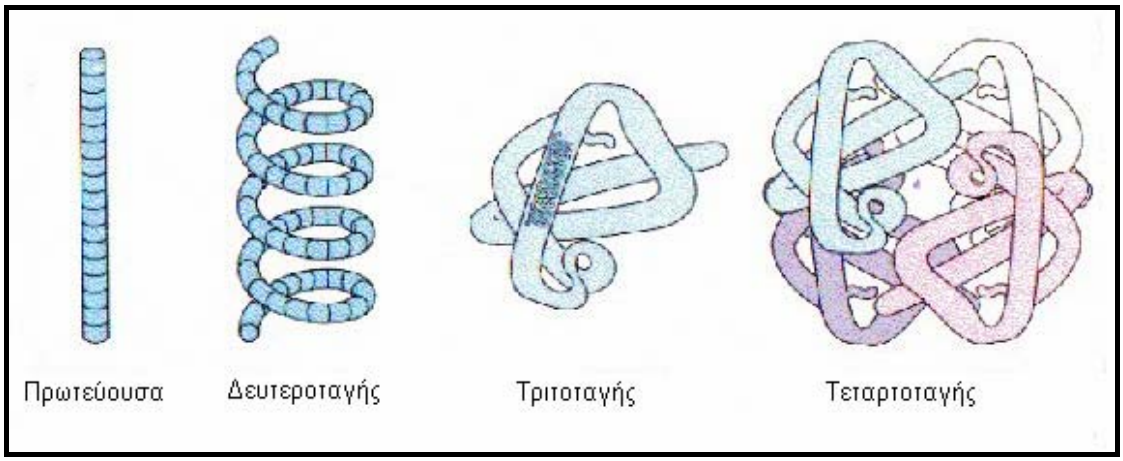
Σε κάθε πρωτεΐνη τα αμινοξέα ενώνονται μεταξύ τους με πεπτιδικούς δεσμούς. Γι' αυτό το λόγο οι πρωτεΐνες ονομάζονται και πολυπεπτιδικές αλυσίδες [5]. Στον πεπτιδικό δεσμό το καρβοξύλιο ενός αμινοξέος ενώνεται με την αμινοομάδα ενός άλλου. Ο πεπτιδικός δεσμός φαίνεται στο σχήμα 1.3. Με αυτό τον τρόπο συνδέονται πολλά αμινοξέα σε σειρά σχηματίζοντας την αλυσίδα της πρωτεΐνης. Ο αριθμός των αμινοξέων που συμμετέχουν στο σχηματισμό της πρωτεΐνης ποικίλει. Υπάρχουν πρωτεΐνες με 100 ως και 5000 περίπου αμινοξέα, αλλά ένας τυπικός αριθμός είναι περίπου 300.



Σχήμα 1.3: Πεπτιδικός δεσμός.

Στην αλυσίδα που σχηματίζεται με τη δημιουργία συνεχόμενων πεπτιδικών δεσμών, στο ένα άκρο έχουμε μια αμινοομάδα και στο άλλο καρβοξύλιο. Με αυτό τον τρόπο μπορούμε να διακρίνουμε τα δύο άκρα και επομένως να θεωρήσουμε ότι το μόριο είναι προσανατολισμένο. Ορίζουμε, λοιπόν, ότι το μόριο μιας πρωτεΐνης αρχίζει από την αμινοομάδα (άκρο N) και τελειώνει στο καρβοξύλιο (άκρο C) [3].

Η απεικόνιση μιας πρωτεΐνης απλά σαν μια ακολουθία των αμινοξέων που την αποτελούν ονομάζεται πρωτεύουσα δομή. Στην πραγματικότητα τα μόρια των πρωτεϊνών έχουν τρισδιάστατη μορφή και για την απεικόνισή τους χρησιμοποιούνται οι δευτεροταγείς, τριτοταγείς και τεταρτοταγείς δομές. Στη δευτεροταγή δομή λαμβάνονται υπ' όψιν μόνο οι αλληλεπιδράσεις μεταξύ των ατόμων C_α με αποτέλεσμα τη δημιουργία ελικοειδών δομών. Στην τριτοταγή δομή έχουμε έναν περαιτέρω προσανατολισμό στο χώρο του μορίου, με τη δημιουργία δομών στο εσωτερικό του. Αυτό συνήθως οδηγεί σε μια σφαιρική δομή, η οποία είναι πάντοτε η ίδια για μια συγκεκριμένη ακολουθία αμινοξέων. Στην τεταρτοταγή δομή έχουμε ομαδοποίηση πρωτεϊνών όπως συμβαίνει στους ζωντανούς οργανισμούς, όπου διαφορετικές πρωτεΐνες «συνεργάζονται» κατά κάποιο τρόπο για να επιτελέσουν μια συγκεκριμένη λειτουργία [5]. Παραδείγματα των δομών αυτών φαίνονται στο Σχήμα 1.4.



Σχήμα 1.4: Πρωτεύουσα, δευτεροταγής, τριτοταγής και τεταρτοταγής δομή μιας πρωτεΐνης.

Εκείνο που καθορίζει τη λειτουργία μιας πρωτεΐνης είναι η τρισδιάστατη δομή της. Αυτό γίνεται καθώς με τη διάταξη του μορίου στις τρεις διαστάσεις δημιουργούνται κατάλληλες περιοχές στις οποίες μπορούν να συνδεθούν άλλα μόρια. Το σχήμα της πρωτεΐνης είναι αυτό που καθορίζει με ποια μόρια μπορεί να ενωθεί. Έτσι για παράδειγμα οι πρωτεΐνες μπορούν να συνδέονται με δύο μόρια για να προκαλέσουν και να επιταχύνουν την μεταξύ τους αντίδραση ή και να συνδέονται με άλλες ίδιες ή διαφορετικές πρωτεΐνες για να επιτελέσουν πιο πολύπλοκες λειτουργίες.
Η σύνθεση των πρωτεϊνών σε έναν ζωντανό οργανισμό λαμβάνει χώρα στις κυτταρικές δομές που ονομάζονται ριβοσώματα. Εκεί τα αμινοξέα που αποτελούν την πρωτείνη συνδέονται με τη σειρά, σύμφωνα με τις πληροφορίες που αποθηκεύονται στον γενετικό κώδικα του οργανισμού. Το πως αποκωδικοποιούνται αυτές οι πληροφορίες θα παρουσιαστεί πιο αναλυτικά στην Ενότητα 1.3 που περιγράφει το ριβονουκλεϊκό οξύ (RNA).

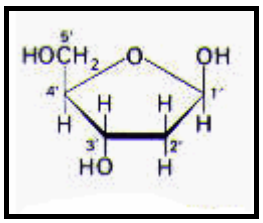
1.2. Δεοξυριβονουκλεϊκό οξύ (DNA)

1.2.1. Η σημασία του DNA

Το DNA (για λόγους ευκολίας θα χρησιμοποιούμε τη διεθνή σύντμηση) αποτελεί το κυριότερο δομικό τμήμα του γενετικού υλικού όλων των ζωντανών οργανισμών. Σε αυτό αποθηκεύονται οι πληροφορίες για τη σύνθεση όλων των πρωτεϊνών ενός ζωντανού οργανισμού. Επομένως όλες οι πληροφορίες για τη δομή και τις λειτουργίες του οργανισμού είναι αποθηκευμένες στο DNA. Ένα άλλο σημαντικό χαρακτηριστικό του είναι ότι αυτές οι πληροφορίες μεταδίδονται κατά τις κυτταρικές διαιρέσεις αναλλοίωτες, έτσι ώστε να μπορούν να παραχθούν όσα αντίγραφα του κυττάρου χρειάζονται.

1.2.2. Η δομή του DNA

Όπως και οι πρωτεΐνες έτσι και ένα μόριο DNA είναι μια αλυσίδα από απλούστερα μόρια. Για την ακρίβεια το DNA είναι μια διπλή αλυσίδα, αλλά για ευκολία στην κατανόηση εξετάζουμε χωριστά τη δομή κάθε αλυσίδας που ονομάζεται νημάτιο. Η βάση αυτής της αλυσίδας είναι μια επαναλαμβανόμενη δομή που σχηματίζεται από ένα σάκχαρο, την 2'-δεοξυριβόζη και μία φωσφορική ρίζα. Το σάκχαρο έχει 5 άτομα άνθρακα τα οποία και αριθμούνται, σύμφωνα με το Σχήμα 1.5. Ο δεσμός που δημιουργεί τη βάση της αλυσίδας σχηματίζεται μεταξύ του άνθρακα 3' του σακχάρου, της φωσφορικής ρίζας και του άνθρακα 5' του επόμενου σακχάρου. Επομένως μπορούμε πάλι να θεωρήσουμε ότι ένα μόριο DNA έχει προσανατολισμό και ορίζουμε ότι αρχίζει από το άκρο 5' και καταλήγει στο άκρο 3' [3].

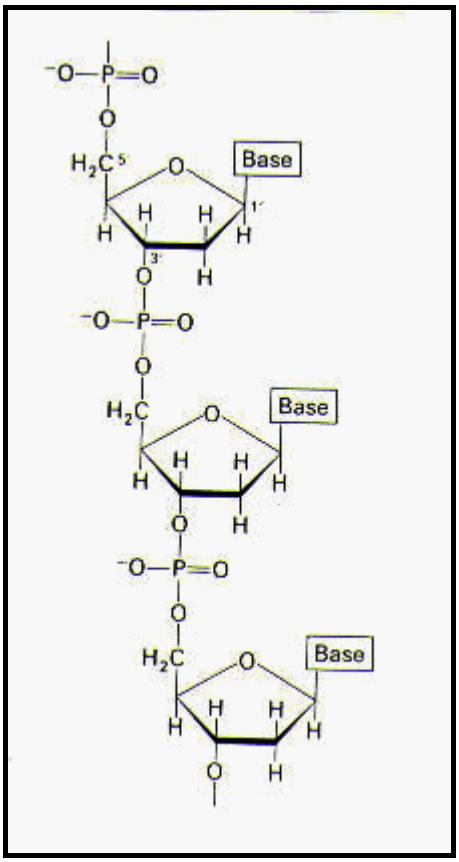


Σχήμα 1.5: Δομή της 2'-δεοξυριβόζης με αρίθμηση των ατόμων άνθρακα.

Σε κάθε άτομο άνθρακα 1' συνδέεται ένα μόριο βάσης. Στο DNA υπάρχουν τέσσερα είδη βάσεων: αδενίνη, γουανίνη, θυμίνη και κυττοσίνη (Πίνακας 1.2). Η βασική δομική μονάδα του DNA που αποτελείται από το σάκχαρο, τη φωσφορική ρίζα και μία βάση ονομάζεται νουκλεοτίδιο. Στο Σχήμα 1.6 έχουμε μια σχηματική απεικόνιση ενός νηματίου DNA [6].

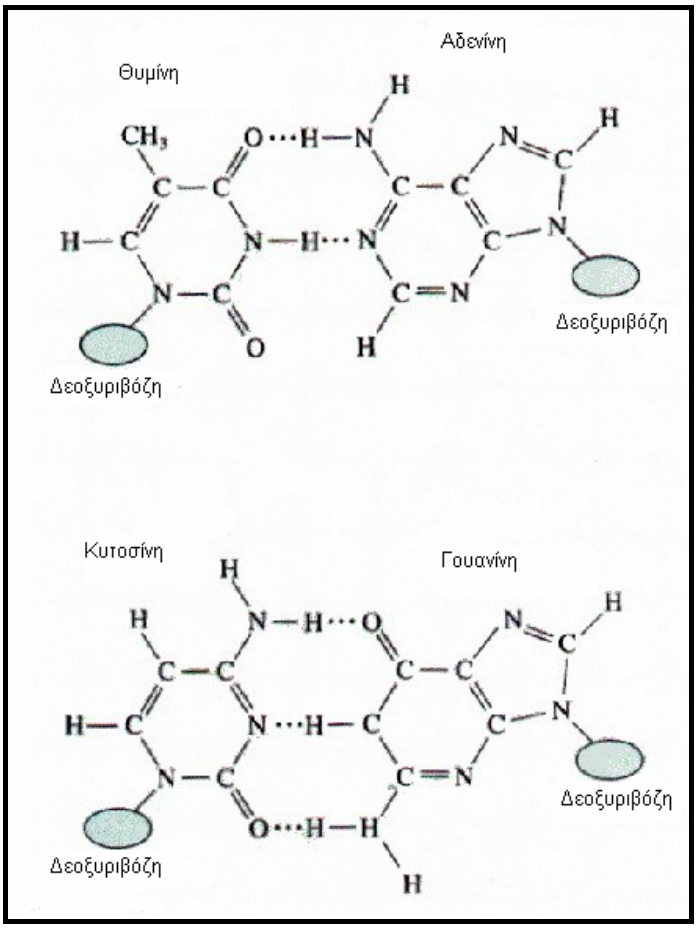
Πίνακας 1.2: Τα είδη βάσεων του DNA και τα σύμβολά τους

Βάση	Σύμβολο
Αδενίνη	A
Γουανίνη	G
Θυμίνη	T
Κυττοσίνη	C

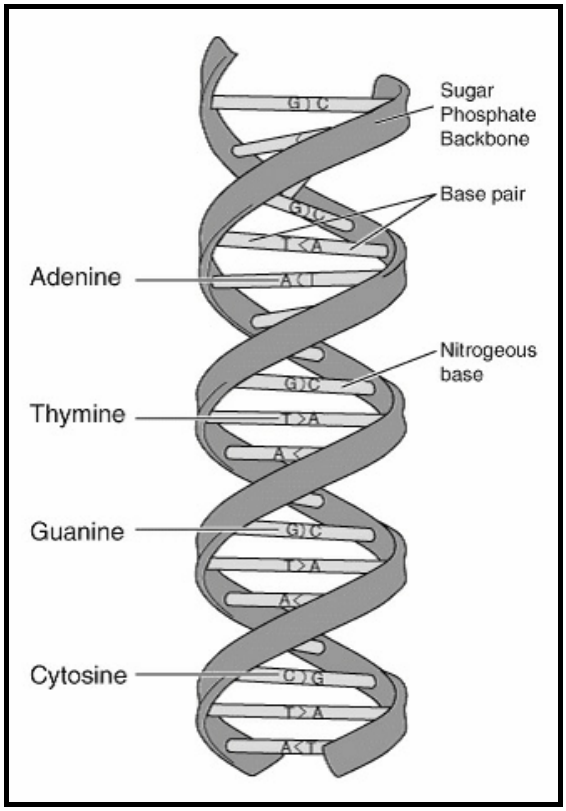


Σχήμα 1.6: Σχηματική απεικόνιση νηματίου DNA.

Όπως αναφέραμε και στην αρχή της παραγράφου, κάθε μόριο DNA αποτελείται από δύο νημάτια, τα οποία συνδέονται μεταξύ τους. Η σύνδεση γίνεται με τη δημιουργία δεσμών μεταξύ των αμινοβάσεων των δυο νηματίων. Ο τρόπος που συνδέονται οι βάσεις είναι μοναδικός. Η αδενίνη συνδέεται πάντα με θυμίνη και η γουανίνη με κυτοσίνη, όπως φαίνεται στο Σχήμα 1.7. Εξαιτίας αυτής της μοναδικότητας στη σύνδεσή τους, οι βάσεις λέγονται και συμπληρωματικές. Έτσι έχοντας μια ακολουθία ορίζουμε ως συμπληρωματική της αυτή που σε κάθε θέση της έχει την συμπληρωματική βάση της αρχικής.
Η μορφή που έχει ένα μόριο DNA στο χώρο είναι ελικοειδής και φαίνεται στο Σχήμα 1.8.



Σχήμα 1.7: Δομή και συνδέσεις μεταξύ των βάσεων.



Σχήμα 1.8: Η ελικοειδής μορφή ενός μορίου DNA.

1.3. Ριβονουκλεϊκό οξύ (RNA)

Τα μόρια RNA παρουσιάζουν πολλές ομοιότητες με τα μόρια DNA. Οι βασικές τους διαφορές εντοπίζονται στα εξής [3]:

- Το σάκχαρο που χρησιμοποιείται στο σχηματισμό μορίων RNA είναι η ριβόζη αντί της 2'-δεοξυριβόζης.
- Στα μόρια RNA δεν υπάρχει θυμίνη (T) αλλά ουρακίλη (U). Όπως και η θυμίνη, η ουρακίλη συνδέεται μόνο με αδενίνη.
- Τα μόρια RNA δε σχηματίζουν τη διπλή έλικα που σχηματίζουν τα μόρια DNA. Αντίθετα, έχουν μόνο ένα νημάτιο και η τρισδιάστατη μορφή τους είναι πιο πολύπλοκη.

Τα μόρια RNA εκτελούν διάφορες λειτουργίες (σε αντίθεση με τα μόρια DNA που μόνο κωδικοποιούν τις γενετικές πληροφορίες). Έτσι ανάλογα με τη λειτουργία τους χωρίζονται σε τρεις τύπους [5]:

- **rRNA**: συμμετέχει στη δομή των ριβοσωμάτων.
- **tRNA**: μεταφέρουν αμινοξέα.
- **mRNA**: αντιγράφουν τα μόρια του DNA.

1.4. Γονίδια και γενετικός κώδικας

Κάθε κύτταρο ενός ζωντανού οργανισμού έχει μόνο λίγα αλλά πολύ μεγάλα μόρια DNA, που ονομάζονται χρωμοσώματα. Στα χρωμοσώματα αποθηκεύονται οι γενετικές πληροφορίες του οργανισμού, με την κωδικοποίηση των οδηγιών για τη σύνθεση των πρωτεϊνών. Οι οδηγίες αυτές κωδικοποιούνται από μεγάλα συνεχή τμήματα των χρωμοσωμάτων. Κάποια άλλα τμήματα όμως δεν κωδικοποιούν πληροφορίες. Ένα άλλο σημαντικό στοιχείο είναι ότι για τη σύνθεση κάποιας πρωτεΐνης αντιστοιχεί συνήθως ένα και μοναδικό τμήμα χρωμοσώματος, το οποίο ονομάζεται γονίδιο. Στα κύτταρα των ζωντανών οργανισμών υπάρχουν οι κατάλληλοι μηχανισμοί για την αναγνώριση της αρχής και του τέλους ενός γονιδίου.
Όπως είδαμε, μια πρωτεΐνη αποτελείται από μια αλυσίδα αμινοξέων. Συνεπώς για να προσδιοριστεί μια πρωτεΐνη πρέπει να προσδιορίσουμε τα αμινοξέα που περιλαμβάνει. Αυτή ακριβώς τη λειτουργία επιτελεί το DNA σε ένα γονίδιο, χρησιμοποιώντας τριάδες νουκλεοτιδίων για τον καθορισμό κάθε αμινοξέος. Οι αντιστοιχίσεις ανάμεσα σε κάθε τέτοια πιθανή τριάδα και τα αμινοξέα αποτελούν το γενετικό κώδικα.
Χρησιμοποιώντας τα τέσσερα διαφορετικά νουκλεοτίδια, έχουμε 64 διαφορετικούς συνδυασμούς, ενώ υπάρχουν 20 διαφορετικά αμινοξέα. Κατά συνέπεια, υπάρχουν διαφορετικές τριάδες που αντιστοιχούν στο ίδιο αμινοξύ. Υπάρχουν επίσης τρεις συνδυασμοί νουκλεοτιδίων που σηματοδοτούν το τέλος ενός γονιδίου. Ακόμα, η τριάδα ATG εκτός από το αμινοξύ μεθιονίνη σηματοδοτεί και την αρχή του γονιδίου. Στον Πίνακα 1.3 που παρουσιάζεται

ο γενετικός κώδικας, οι συνδυασμοί αυτοί σημειώνονται σαν τέλος και αρχή αντίστοιχα [6].

Πίνακας 1.3: Γενετικός κώδικας.

Πρώτη βάση	Δεύτερη βάση				Τρίτη βάση
	G	A	C	U	
G	Gly	Glu	Ala	Val	G
	Gly	Glu	Ala	Val	A
	Gly	Asp	Ala	Val	C
	Gly	Asp	Ala	Val	U
A	Arg	Lys	Thr	Met/ Αρχή	G
	Arg	Lys	Thr	Ile	A
	Ser	Asn	Thr	Ile	C
	Ser	Asn	Thr	Ile	U
C	Arg	Gln	Pro	Leu	G
	Arg	Gln	Pro	Leu	A
	Arg	His	Pro	Leu	C
	Arg	His	Pro	Leu	U
U	Trp	Τέλος	Ser	Leu	G
	Τέλος	Τέλος	Ser	Leu	A
	Cys	Tyr	Ser	Phe	C
	Cys	Tyr	Ser	Phe	U

Σημειώνουμε εδώ ότι στον γενετικό κώδικα χρησιμοποιούμε την ουρακίλη αντί της θυμίνης. Αυτό γίνεται επειδή για τη σύνθεση πρωτεϊνών αντιγράφεται το αντίστοιχο τμήμα DNA με τη δημιουργία ενός μορίου mRNA. Αυτό το μόριο χρησιμοποιείται για τη σύνθεση της πρωτεΐνης στα ριβοσώματα.
Ένα θέμα που πρέπει να σημειώσουμε για την αποκωδικοποίηση του DNA είναι ότι η ίδια ακολουθία μπορεί να αποκωδικοποιηθεί με διαφορετικούς τρόπους, ανάλογα με τον τρόπο που ομαδοποιούμε τα νουκλεοτίδια [4]. Ας θεωρήσουμε για παράδειγμα την ακολουθία:

TAATCGAATGGGC

Αν ξεκινήσουμε από την πρώτη θέση, έχουμε τις τριάδες TAA, TCG, AAT, GGG, αφήνοντας έξω το τελευταίο C. Μια άλλη ομαδοποίηση αφήνοντας έξω το πρώτο T θα έδινε AAT, CGA, ATG, GGC, ενώ αφήνοντας έξω τα αρχικά σύμβολα TA και τα τελικά GC έχουμε ATC, GAA, TGG. Πρόκειται για τις τρεις ομαδοποιήσεις που προκύπτουν αν ξεκινήσουμε την αποκωδικοποίηση από το $1^{ο}$, $2^{ο}$ και $3^{ο}$ σύμβολο αντίστοιχα. Αν σε αυτές προσθέσουμε και τις τρεις ομαδοποιήσεις που αντιστοιχούν στην συμπληρωματική ακολουθία, τότε έχουμε συνολικά έξι διαφορετικές αποκωδικοποιήσεις για κάθε ακολουθία DNA.

1.5. Ανακεφαλαίωση

Στο κεφάλαιο αυτό παρουσιάσαμε μερικές βασικές έννοιες της μοριακής βιολογίας. Τα σημεία στα οποία εστιάσαμε περισσότερο την περιγραφή ήταν τα μόρια πρωτεϊνών και DNA, ιδιαίτερα σε ότι αφορά τη δομή τους. Ο λόγος για τον οποίο έγινε αυτό είναι ότι αυτή η δομή είναι που επιτρέπει την αναπαράσταση των μορίων αυτών ως μία σειρά συμβόλων, στην οποία κάθε σύμβολο αντιστοιχεί σε μια μικρότερη δομική μονάδα των μορίων αυτών (αμινοξύ για πρωτεΐνες ή βάση για DNA). Και επειδή, όπως είδαμε, μπορούμε να θεωρήσουμε ότι το κάθε μόριο έχει προσανατολισμό, η σειρά αυτή των συμβόλων είναι μοναδική για κάθε μόριο. Το γεγονός αυτό είναι πολύ σημαντικό για τα θέματα της υπολογιστικής βιολογίας, με τα οποία θα ασχοληθούμε στο Κεφάλαιο 2.

ΚΕΦΑΛΑΙΟ 2

Υπολογιστική βιολογία και βάσεις βιολογικών δεδομένων

Σκοπός αυτού του κεφαλαίου είναι η περιγραφή του προβλήματος με το οποίο ασχοληθήκαμε σε αυτή την εργασία και η κατανόηση των παραμέτρων του. Για μια πλήρη παρουσίαση του προβλήματος, ξεκινάμε με μια εισαγωγή στις απαραίτητες έννοιες της υπολογιστικής βιολογίας που θα μας απασχολήσουν. Για αυτή την εισαγωγή ακολουθήσαμε τη δομή του αντίστοιχου μέρους της εργασίας της Γεωργίας Αδαμοπούλου "Continuous Queries In Biological Sequence Databases". Στη συνέχεια γίνεται αναφορά στις βάσεις δεδομένων που περιέχουν βιολογικές ακολουθίες, τόσο ως προς το περιεχόμενο όσο και ως προς τη διαθεσιμότητά τους. Τέλος, ανακεφαλαιώνουμε αναφέροντας τις παραμέτρους εκείνες που επηρεάζουν τις επιλογές μας για τη λύση.

2.1. Βασικές έννοιες της υπολογιστικής βιολογίας.

2.1.1. Αλφάβητο

Ορισμός 2.1: *Ένα πεπερασμένο σύνολο από σύμβολα ονομάζεται αλφάβητο.*

Για το πρόβλημα που εξετάζουμε, χρειαζόμαστε δύο διαφορετικά αλφάβητα, ένα για κάθε είδος μορίων (πρωτεΐνες ή DNA). Κάθε ένα από αυτά τα αλφάβητα αποτελείται κατά βάση από τα σύμβολα για τα αμινοξέα και τις βάσεις αντίστοιχα (Πίνακες 1.1 και 1.2). Έτσι έχουμε τα δύο βασικά αλφάβητα:

$$\Sigma_{DNA}=\{ A, C, G, T \}$$

και

$$\Sigma_{PROTEIN}=\{ A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y \}$$

Εκτός από τα βασικά σύμβολα, σε κάθε αλφάβητο προσθέτουμε και ένα άλλο σύνολο από σύμβολα τα οποία ονομάζουμε αμφίσημα (ambiguous symbols). Τα σύμβολα αυτά χρησιμοποιούνται στην περίπτωση που είτε είναι άγνωστο είτε δεν έχει σημασία ποιο σύμβολο βρίσκεται σε κάποια θέση. Έτσι κάθε αμφίσημο σύμβολο μπορεί να ερμηνευτεί ως κάποιο από δύο ή περισσότερα διαφορετικά βασικά σύμβολα.
Η περίπτωση που έχουμε ένα άγνωστο σύμβολο προκύπτει συνήθως από την αδυναμία πειραματικής προσδιορισμού του αμινοξέος ή της βάσης που βρίσκεται στο σημείο της ακολουθίας. Για την περίπτωση αυτή χρησιμοποιούμε ένα σύμβολο το οποίο μπορεί να ερμηνευτεί ως οποιοδήποτε

βασικό σύμβολο. Υπάρχει όμως και η πιθανότητα να μην έχει προσδιοριστεί ακριβώς το αμινοξύ ή η βάση σε μια θέση, αλλά να έχει προσδιοριστεί κάποια γενικότερη οικογένεια στην οποία ανήκει. Τα αμινοξέα ασπαραγγίνη N και ασπαρτικό οξύ D, όπως και τα γλουταμικό οξύ E και γλουταμίνη Q είναι συγγενικά και υπάρχει η πιθανότητα να μην γίνει ακριβής προσδιορισμός τους σε μια πρωτεΐνη. Για κάθε ένα από τα παραπάνω ζεύγη εισάγουμε στο αλφάβητο ένα νέο αμφίσημο σύμβολο. Επιπλέον, στις ακολουθίες DNA υπάρχουν περιπτώσεις στις οποίες δεν έχει σημασία το αμινοξύ που βρίσκεται σε κάποια θέση. Αυτό οφείλεται στο γεγονός ότι διαφορετικές τριάδες βάσεων αποκωδικοποιούνται με βάση τον γενετικό κώδικα (Πίνακας 1.3) στο ίδιο αμινοξύ (για παράδειγμα οι τριάδες GGA, GGC, GGG, GGT αντιστοιχούν στο αμινοξύ γλυκίνη, επομένως η βάση στην $3^{η}$ θέση δεν επηρεάζει την αποκωδικοποίηση). Επομένως εισάγουμε στο αλφάβητο αμφίσημα σύμβολα που αντιπροσωπεύουν τους πιθανούς συνδυασμούς των βάσεων ανά δύο και ανά τρεις. Τα αμφίσημα σύμβολα για πρωτεΐνες και ακολουθίες DNA και τα σύμβολα στα οποία μπορεί να αντιστοιχούν φαίνονται στους Πίνακες 2.1 και 2.2 αντίστοιχα.

Πίνακας 2.1: Αμφίσημα σύμβολα για ακολουθίες πρωτεϊνών και αντιστοιχίες.

Σύμβολο	Αντιστοιχίες
B	D, N
Z	E, Q
X	Οποιοδήποτε σύμβολο

Πίνακας 2.2. Αμφίσημα σύμβολα για ακολουθίες DNA και αντιστοιχίες.

Σύμβολο	Αντιστοιχίες
S	G, C
W	A, T
R	A, G
Y	T, C
K	T, G
M	A, C
B	T, G, C
V	A, G, C
H	A, T, C
D	A, T, G
N	Οποιοδήποτε σύμβολο

Τα τελικά αλφάβητα για DNA και πρωτεΐνες λοιπόν είναι:

$$\Sigma_{DNA}=\{ A, C, G, T, S, W, R, Y, K, M, B, V, H, D, N \}$$

και

$$\Sigma_{PROTEIN}=\{ A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y, B, Z, X \}$$

2.1.2. Ακολουθία (Sequence)

Ορισμός 2.2: *Κάθε διατεταγμένη σειρά συμβόλων που ανήκουν σε ένα αλφάβητο ονομάζεται ακολουθία* [3].

Η ακολουθία αποτελεί βασική έννοια για την υπολογιστική βιολογία, καθώς χρησιμοποιείται για την αναπαράσταση των μορίων DNA και πρωτεϊνών. Όπως είδαμε στο Κεφάλαιο 1, η δομή των μορίων αυτών επιτρέπει την αναπαράστασή τους σαν μια σειρά από αμινοξέα (πρωτεΐνες) ή βάσεις (DNA). Έτσι λοιπόν, μια ακολουθία DNA είναι μία σειρά από σύμβολα του αλφαβήτου Σ_{DNA}, ενώ μια ακολουθία πρωτεΐνης είναι μια σειρά από σύμβολα του αλφαβήτου $\Sigma_{PROTEIN}$.
Το μήκος μιας ακολουθίας δίνεται από τον συνολικό αριθμό συμβόλων που την αποτελούν. Το μήκος μιας ακολουθίας *s* συμβολίζεται με $|s|$. Αν μία ακολουθία έχει μήκος ίσο με μηδέν, τότε λέγεται κενή ακολουθία και συμβολίζεται με *e*. Το σύμβολο στη θέση *i* μιας ακολουθίας συμβολίζεται με $s[i]$.

Ορισμός 2.2: *Μια υποακολουθία* t *μιας ακολουθίας* s *ορίζεται ως ένα υποσύνολο των συμβόλων της* s*, για τα οποία διατηρείται η σχετική σειρά εμφάνισης* [2].

Τα σύμβολα αυτά δεν είναι απαραίτητο να εμφανίζονται συνεχόμενα στην ακολουθία *s*. Αν η υποακολουθία *t* προκύπτει από συνεχόμενα σύμβολα της υποακολουθίας *s* τότε ονομάζεται συνεχής.

Παράδειγμα 2.1

Η συμβολοσειρά

GATTACA

αποτελεί μια ακολουθία μήκους 7, με s[1]=*G,* s[2]=*A... που μπορεί να είναι είτε ακολουθία DNA είτε πρωτεΐνη. Η συμβολοσειρά*

GTTC

αποτελεί μια (μη συνεχή) υποακολουθία με μήκος 4, ενώ η

ATTAC

αποτελεί μια συνεχή υποακολουθία μήκους 5. Η συμβολοσειρά

EATHANHAWKE

αποτελεί μια ακολουθία πρωτεΐνης μήκους 11. Αντίθετα, η συμβολοσειρά

δεν αποτελεί έγκυρη ακολουθία ούτε DNA ούτε πρωτεΐνης, αφού ο χαρακτήρας U δεν ανήκει σε κανένα από τα δύο αλφάβητα.

2.1.3. Αντιστοίχιση (Alignment)

Ορισμός 2.3: *Έχοντας δύο ακολουθίες* s_1 *και* s_2*. Μια αντιστοίχιση (alignment) μεταξύ των ακολουθιών* s_1 *και* s_2 *είναι η εισαγωγή κενών σε τυχαία σημεία των ακολουθιών έτσι ώστε να αποκτήσουν το ίδιο μήκος και η τοποθέτησή τους με τέτοιο τρόπο ώστε σε κάθε σύμβολο στην* s_1 *να αντιστοιχίζεται με ένα σύμβολο ή κενό στην* s_2 *ενώ κάθε κενό στην* s_1 *αντιστοιχίζεται αποκλειστικά με ένα σύμβολο στην* s_2 [3].

Για τα κενά που εισάγουμε ώστε οι ακολουθίες να αποκτήσουν το ίδιο μήκος, δεν υπάρχει κάποιος περιορισμός. Μπορούν να εισάγονται σε οποιοδήποτε σημείο των ακολουθιών: στην αρχή, ανάμεσα στα σύμβολα, στο τέλος. Η βασική διάκριση των αντιστοιχίσεων περιλαμβάνει δύο είδη:

- **Ολική αντιστοίχιση:** όταν η αντιστοίχιση αφορά σε όλο το μήκος των ακολουθιών και χρησιμοποιούνται όλα τα σύμβολα και των δύο.
- **Τοπική αντιστοίχιση:** όταν αφορά συνεχείς υποακολουθίες των δύο ακολουθιών, όχι απαραίτητα ίσου μήκους, επομένως χρησιμοποιούμε μόνο ένα μέρος των συμβόλων τους. Και σε αυτή την περίπτωση εισάγουμε κενά έτσι ώστε οι δύο υποακολουθίες να αποκτήσουν ίδιο μήκος.

Παράδειγμα 2.2

Έστω οι ακολουθίες DNA s_1=AGTCCATT, s_2=GTACCT. *Οι παρακάτω αποτελούν ολικές αντιστοιχίσεις μεταξύ τους:*

```
AGTCCATT
-GTACCT-

A-GTC-CATT
GT-A-C—C-T

--AGTCCATT
GTACCT----

AGT-CCA-TT
--GTACCT--
```

όπου ο χαρακτήρας '-' συμβολίζει τα κενά.

2.1.4. Βαθμολόγηση (Score) και Ομοιότητα (Similarity)

Όπως είδαμε στην Ενότητα 2.1.3 μεταξύ δύο ακολουθιών μπορούν να υπάρξουν πολλές αντιστοιχίσεις. αυτό που μας ενδιαφέρει λοιπόν είναι να έχουμε ένα μέτρο αξιολόγησης των αντιστοιχίσεων, για να επιλέγουμε αυτές που παρουσιάζουν το μεγαλύτερο ενδιαφέρον. Γι' αυτό το λόγο, εισάγουμε την έννοια της βαθμολόγησης, αρχικά για ζεύγη συμβόλων και στη συνέχεια για αντιστοιχίσεις.

Ορισμός 2.4: *Ας θεωρήσουμε το αλφάβητο* Σ' *που προκύπτει από το αλφάβητο Σ, για μια ακολουθία, προσθέτοντας το σύμβολο '-' για το κενό. Τότε η βαθμολόγηση (score) για κάθε ζεύγος συμβόλων* x, y *που ανήκει στο* Σ' *συμβολίζεται με* S(x, y) *και είναι μία συνάρτηση που αντιστοιχεί στο ζεύγος* x, y *μια τιμή που υποδηλώνει την αξία της αντιστοίχισης μεταξύ των δύο χαρακτήρων* [2].

Οι τιμές της συνάρτησης βαθμολόγησης είναι προκαθορισμένες και χρησιμοποιούνται για την βαθμολόγηση των αντιστοιχίσεων ακολουθιών. Η συνάρτηση βαθμολόγησης δίνει συνήθως αρνητικές τιμές στην αντιστοίχιση διαφορετικών συμβόλων και στην αντιστοίχιση συμβόλου με κενό (γι' αυτό και η αξία αυτών των αντιστοιχίσεων ονομάζεται και κόστος), ενώ δίνει θετικές όταν τα σύμβολα που αντιστοιχίζονται είναι όμοια.
Ας εξετάσουμε τώρα την περίπτωση της αντιστοίχισης ακολουθιών.

Ορισμός 2.5: *Έστω μια αντιστοίχιση* a *μεταξύ δύο ακολουθιών* s_1, s_2. *Αν* s_1', s_2' *είναι οι ακολουθίες που προκύπτουν μετά την εισαγωγή κενών και* n *το κοινό τους μήκος στην αντιστοίχιση* a, *τότε η βαθμολόγηση της αντιστοίχισης (alignment score) ορίζεται ως:*

$$\sum_{i=1}^{n} S(s_1'(i), s_2'(i)) \text{ [2]}$$

Η βαθμολόγηση της αντιστοίχισης δηλαδή προκύπτει από το άθροισμα των αντιστοιχίσεων των συμβόλων των ακολουθιών μετά την εισαγωγή των κενών.

Ορισμός 2.6: *Έστω μία αντιστοίχιση* a *μεταξύ δύο ακολουθιών* s_1, s_2, *η οποία δίνει βαθμολόγηση υψηλότερη από κάθε άλλη δυνατή αντιστοίχιση, με μία δεδομένη συνάρτηση βαθμολόγησης* S *για το αλφάβητο* Σ' *των ακολουθιών. Η μέγιστη αυτή βαθμολόγηση για την αντιστοίχιση* a *ονομάζεται ομοιότητα (similarity) και συμβολίζεται με* $\text{sim}(s_1,s_2)$. [2]

Αν και η τιμή της ομοιότητας είναι μοναδική για δύο ακολουθίες, πολλές φορές υπάρχουν περισσότερες από μία αντιστοιχίσεις που μας δίνουν αυτή τη μέγιστη βαθμολόγηση. Κάθε τέτοια αντιστοίχιση ονομάζεται βέλτιστη αντιστοίχιση (optimal alignment).

Παράδειγμα 2.3

Έστω η συνάρτηση βαθμολόγησης:

$$S(a,b) = \begin{cases} 1, \text{αν a} = \text{b} \\ -2, \text{αν a} = \text{'–'ή b} = \text{'–'} \\ -1, \text{αν a} \neq \text{'–'}, \text{b} \neq \text{'–'}, \text{a} \neq \text{b} \end{cases}$$

τότε οι αντιστοιχίσεις του Παραδείγματος 2.2 έχουν τις παρακάτω βαθμολογήσεις:

(-2)+1+1+(-1)+1+(-1)+1+(-2)= -2
(-1)+(-2)+(-2)+(-1)+(-2)+(-2)+(-2)+(-1)+(-2)+1= -14
(-2)+(-2)+1+(-1)+(-1)+(-1)+(-2)+(-2)+(-2)+(-2)= -14
(-2)+(-2)+(-1)+(-2)+(-1)+1+(-1)+(-2)+(-2)+(-2)= -13

Επειδή έχουμε να κάνουμε με μικρές ακολουθίες μπορούμε να διαπιστώσουμε σχετικά εύκολα ότι η πρώτη αντιστοίχιση είναι και βέλτιστη (δεν υπάρχει αντιστοίχιση με υψηλότερη βαθμολόγηση). Επομένως η ομοιότητα των δύο ακολουθιών είναι:

sim(AGTCCATT, GTACCT) = -2.

2.1.5. Πίνακες βαθμολόγησης (Scoring matrices)

Όπως είδαμε στην Ενότητα 2.1.4, για την εύρεση της βαθμολόγησης μιας αντιστοίχισης, είναι απαραίτητο να έχουμε κάποια μέθοδο που να αντιστοιχίζει σε κάθε δυνατό ζεύγος συμβόλων μια τιμή. Στα μέχρι τώρα παραδείγματα χρησιμοποιήσαμε μια συνάρτηση, η οποία μας έδινε το πολύ τρεις διαφορετικές τιμές: μία για τη συμφωνία συμβόλων (match), μία για την ασυμφωνία (mismatch) και μία για την αντιστοίχιση συμβόλου με κενό. Στην πράξη όμως, ιδιαίτερα όταν ασχολούμαστε με ακολουθίες πρωτεϊνών, μας ενδιαφέρει η συνάρτηση αυτή να μας δίνει μεγαλύτερη «ποικιλία» τιμών. Αυτό είναι απαραίτητο, καθώς παρατηρούνται πολύ συχνά μεταβολές στα μόρια των πρωτεϊνών και DNA. Οι μεταβολές αυτές συνίστανται στην προσθήκη αμινοξέων ή βάσεων, συνεπώς και συμβόλων σε μια ακολουθία, στην αντικατάσταση, που παρατηρείται κυρίως μεταξύ αμινοξέων και βάσεων που έχουν συγγενική δομή ή λειτουργία και στην αφαίρεση, κατά την οποία το αμινοξύ ή η βάση που βρίσκεται σε μία θέση «εξαφανίζεται» και γειτονικά του αμινοξέα ή βάσεις συνδέονται μεταξύ τους, κλείνοντας το κενό. Οι μεταβολές αυτές δε συμβαίνουν με την ίδια συχνότητα για κάθε αμινοξύ ή βάση. Αντίθετα έχει παρατηρηθεί ότι ειδικά στις αντικαταστάσεις, για κάποια από τα αμινοξέα υπάρχει μεγαλύτερη συχνότητα αντικατάστασής τους, τόσο από οποιοδήποτε αμινοξύ όσο και από κάποια συγκεκριμένα (συνήθως συγγενικά).

Συνεπώς, όταν μας ενδιαφέρει να βρούμε ακολουθίες με μεγάλη ομοιότητα, θα πρέπει να λάβουμε υπ' όψιν ότι δεν έχουν ούτε όλες οι συμφωνίες συμβόλων την ίδια αξία, ούτε όλες οι ασυμφωνίες το ίδιο κόστος. Θα πρέπει λοιπόν να χρησιμοποιήσουμε μια μέθοδο που σε κάθε δυνατό ζεύγος συμβόλων θα αντιστοιχίζει μια τιμή από ένα ευρύτερο πεδίο. Για το λόγο αυτό, χρησιμοποιούμε τους πίνακες βαθμολόγησης.

Ορισμός 2.7: *Ένας πίνακας βαθμολόγησης (scoring matrix)* M *για ένα αλφάβητο* Σ *είναι ένας πίνακας* n×n*, όπου n το πλήθος των συμβόλων του αλφάβητου, στον οποίο κάθε στοιχείο* M[i, j] *μας δίνει τη βαθμολόγηση της αντιστοίχισης των συμβόλων* i, j *του αλφάβητου.*

Παράδειγμα 2.4

Η συνάρτηση βαθμολόγησης που χρησιμοποιήσαμε στο Παράδειγμα 2.3 μας δίνει τον παρακάτω πίνακα:

	A	**C**	**G**	**T**	**-**
A	1	-1	-1	-1	-2
C	-1	1	-1	-1	-2
G	-1	-1	1	-1	-2
T	-1	-1	-1	1	-2
-	-2	-2	-2	-2	-2

Οι πίνακες βαθμολόγησης που χρησιμοποιούμε στην πράξη για πρωτεΐνες κατασκευάζονται θεωρητικά, λαμβάνοντας υπ' όψιν τις παραμέτρους που αναφέραμε πριν. Έτσι, ανάλογα με τον τρόπο που κατασκευάζονται διακρίνουμε δύο είδη: τους πίνακες PAM και τους πίνακες BLOSUM.

2.1.5.1. Πίνακες PAM (Point Accepted Mutation)

Οι πίνακες PAM [8] είναι πίνακες βαθμολόγησης που κωδικοποιούν τις αναμενόμενες εξελικτικές μεταβολές σε επίπεδο αμινοξέων. Κάθε πίνακας PAM είναι κατασκευασμένος για να χρησιμοποιείται στη σύγκριση ακολουθιών που διαφέρουν κατά μία συγκεκριμένη τιμή μονάδων PAM. Δύο ακολουθίες s_1, s_2 διαφέρουν κατά μία μονάδα PAM όταν μετά από μια σειρά αντικαταστάσεων αμινοξέων (και όχι προσθηκών ή αφαιρέσεων) η ακολουθία s_1 μετατράπηκε στην s_2, με μέσο όρο μία αντικατάσταση κάθε εκατό αμινοξέα. Έτσι, για παράδειγμα, ο πίνακας PAM 120 είναι ιδανικός για τη σύγκριση ακολουθιών που διαφέρουν κατά 120 μονάδες PAM. Για κάθε ζεύγος αμινοξέων-συμβόλων A_i, A_j το στοιχείο $[i, j]$ ενός πίνακα PAM n αντιπροσωπεύει την πιθανότητα να αντικαταστήσει το A_i το A_j σε δύο ακολουθίες που διαφέρουν κατά n μονάδες PAM.
Η κατασκευή ενός πίνακα PAM n γίνεται θεωρητικά χρησιμοποιώντας πολλά ζεύγη ακολουθιών που είναι γνωστό ότι διαφέρουν κατά n μονάδες PAM [2]. Αφού αντιστοιχιστεί κάθε ζεύγος ακολουθιών

υπολογίζουμε για κάθε ζεύγος συμβόλων A_i, A_j την συχνότητα αντιστοίχισης των συμβόλων, την οποία συμβολίζουμε με $f(i, j)$. Αν $f(i)$, $f(j)$ είναι οι συχνότητες εμφάνισης των αμινοξέων σε όλες της ακολουθίες τότε το στοιχείο $[i, j]$ του πίνακα PAM n δίνεται από τη σχέση:

$$\log \frac{f(i,j)}{f(i)f(j)}$$

Οι τιμές αυτές πολλαπλασιάζονται συνήθως επί 10 και στρογγυλοποιούνται σε ακέραιες. Στο Σχήμα 2.1 φαίνεται ένα παράδειγμα πίνακα PAM, ο πίνακας PAM 120. [11]

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	B	Z	X	*
A	3	-3	-1	0	-3	-1	0	1	-3	-1	-3	-2	-2	-4	1	1	1	-7	-4	0	0	-1	-1	-8
R	-3	6	-1	-3	-4	1	-3	-4	1	-2	-4	2	-1	-5	-1	-1	-2	1	-5	-3	-2	-1	-2	-8
N	-1	-1	4	2	-5	0	1	0	2	-2	-4	1	-3	-4	-2	1	0	-4	-2	-3	3	0	-1	-8
D	0	-3	2	5	-7	1	3	0	0	-3	-5	-1	-4	-7	-3	0	-1	-8	-5	-3	4	3	-2	-8
C	-3	-4	-5	-7	9	-7	-7	-4	-4	-3	-7	-7	-6	-6	-4	0	-3	-8	-1	-3	-6	-7	-4	-8
Q	-1	1	0	1	-7	6	2	-3	3	-3	-2	0	-1	-6	0	-2	-2	-6	-5	-3	0	4	-1	-8
E	0	-3	1	3	-7	2	5	-1	-1	-3	-4	-1	-3	-7	-2	-1	-2	-8	-5	-3	3	4	-1	-8
G	1	-4	0	0	-4	-3	-1	5	-4	-4	-5	-3	-4	-5	-2	1	-1	-8	-6	-2	0	-2	-2	-8
H	-3	1	2	0	-4	3	-1	-4	7	-4	-3	-2	-4	-3	-1	-2	-3	-3	-1	-3	1	1	-2	-8
I	-1	-2	-2	-3	-3	-3	-3	-4	-4	6	1	-3	1	0	-3	-2	0	-6	-2	3	-3	-3	-1	-8
L	-3	-4	-4	-5	-7	-2	-4	-5	-3	1	5	-4	3	0	-3	-4	-3	-3	-2	1	-4	-3	-2	-8
K	-2	2	1	-1	-7	0	-1	-3	-2	-3	-4	5	0	-7	-2	-1	-1	-5	-5	-4	0	-1	-2	-8
M	-2	-1	-3	-4	-6	-1	-3	-4	-4	1	3	0	8	-1	-3	-2	-1	-6	-4	1	-4	-2	-2	-8
F	-4	-5	-4	-7	-6	-6	-7	-5	-3	0	0	-7	-1	8	-5	-3	-4	-1	4	-3	-5	-6	-3	-8
P	1	-1	-2	-3	-4	0	-2	-2	-1	-3	-3	-2	-3	-5	6	1	-1	-7	-6	-2	-2	-1	-2	-8
S	1	-1	1	0	0	-2	-1	1	-2	-2	-4	-1	-2	-3	1	3	2	-2	-3	-2	0	-1	-1	-8
T	1	-2	0	-1	-3	-2	-2	-1	-3	0	-3	-1	-1	-4	-1	2	4	-6	-3	0	0	-2	-1	-8
W	-7	1	-4	-8	-8	-6	-8	-8	-3	-6	-3	-5	-6	-1	-7	-2	-6	12	-2	-8	-6	-7	-5	-8
Y	-4	-5	-2	-5	-1	-5	-5	-6	-1	-2	-2	-5	-4	4	-6	-3	-3	-2	8	-3	-3	-5	-3	-8
V	0	-3	-3	-3	-3	-3	-3	-2	-3	3	1	-4	1	-3	-2	-2	0	-8	-3	5	-3	-3	-1	-8
B	0	-2	3	4	-6	0	3	0	1	-3	-4	0	-4	-5	-2	0	0	-6	-3	-3	4	2	-1	-8
Z	-1	-1	0	3	-7	4	4	-2	1	-3	-3	-1	-2	-6	-1	-1	-2	-7	-5	-3	2	4	-1	-8
X	-1	-2	-1	-2	-4	-1	-1	-2	-2	-1	-2	-2	-2	-3	-2	-1	-1	-5	-3	-1	-1	-1	-2	-8
*	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	1

Σχήμα 2.1: Ο πίνακας PAM 120.

2.1.5.2. Πίνακες BLOSUM (BLOcks SUbstitution Matrix)

Οι πίνακες BLOSUM [9] κατασκευάζονται χρησιμοποιώντας τα στοιχεία της βάσης δεδομένων BLOCKS. Η βάση αυτή περιέχει τα τμήματα ακολουθιών που διατηρούνται αμετάβλητα σε οικογένειες συγγενών πρωτεϊνών. Περιέχει περίπου 3000 ομάδες από σύντομες ακολουθίες από 800 περίπου ομάδες πρωτεϊνών.
Για την κατασκευή ενός πίνακα BLOSUM ορίζουμε σαν P το σύνολο όλων των ζευγών θέσεων στις ομάδες που περιέχονται στη βάση δεδομένων BLOCKS, έτσι ώστε οι δύο θέσεις κάθε ζεύγους στο P να βρίσκονται στην ίδια στήλη της ίδιας ομάδας. Αν για παράδειγμα κάθε ομάδα έχει n στήλες και κάθε στήλη ακριβώς k γραμμές, τότε το

σύνολο P περιλαμβάνει $n \times \binom{k}{2}$ ζεύγη θέσεων. Αν $n(i, j)$ είναι ο αριθμός των ζευγών θέσεων για τις οποίες στη μία θέση βρίσκεται το σύμβολο A_i και στην άλλη το A_j και $f(i), f(j)$ οι συχνότητες εμφάνισης των συμβόλων αυτών στην BLOCKS, τότε η βάση για την τιμή του στοιχείου $[i, j]$ του πίνακα BLOSUM δίνεται από τη σχέση:

$$\log_2 \frac{n(i, j)}{|P| f(i) f(j)}$$

Η τιμή αυτή πολλαπλασιάζεται με 2 και στρογγυλοποιείται στον πλησιέστερο ακέραιο.
Ο πίνακας BLOSUM x υπολογίζεται αν για κάθε ζεύγος ακολουθιών, μέσα σε μια ομάδα, που είναι περισσότερο από x % όμοιες αφαιρέσουμε τη μία. Το αποτέλεσμα είναι ότι σε κάθε ομάδα οποιοδήποτε ζεύγος ακολουθιών είναι λιγότερο από x % όμοιο [2]. Με αυτό τον τρόπο περιορίζεται η επιρροή των ακολουθιών που έχουν μεταξύ τους υψηλή ομοιότητα. Ο λόγος για τον οποίο γίνεται αυτό, είναι ότι με αυτό τον τρόπο εντοπίζονται καλύτερα ομοιότητες μεταξύ τμημάτων ακολουθιών από ένα σύνολο ακολουθιών που διαφέρουν σημαντικά. Στο Σχήμα 2.2 φαίνεται ένα παράδειγμα πίνακα BLOSUM, ο πίνακας BLOSUM 62 [12].

	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	0	-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	-2	-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	-2	-4
S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4	1	-3	-2	-2	0	0	0	-4
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5	-2	-2	0	-1	-1	0	-4
W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11	2	-3	-4	-3	-2	-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	0	-1	-1	-1	-2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-2	0	0	-2	-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

Σχήμα 2.2: Ο πίνακας BLOSUM 62.

2.2. Βάσεις βιολογικών δεδομένων

Η κύρια κατηγοριοποίηση των βάσεων βιολογικών δεδομένων γίνεται με βάση τον τύπο των ακολουθιών που περιέχουν, ακολουθίες DNA ή πρωτεϊνών. Τέτοιες βάσεις δεδομένων υπάρχουν δεκάδες (για καθένα από τα είδη ακολουθιών) και διαφέρουν σημαντικά τόσο ως προς τη δομή και τις πληροφορίες που αποθηκεύονται, όσο και ως προς το μέγεθος, ανάλογα με την εξειδίκευσή τους. Οι βάσεις αυτές είναι συνήθως διαθέσιμες στο διαδίκτυο και περιέχουν συνήθως αρκετές χιλιάδες ακολουθίες. Στον Πίνακα 2.3 παρουσιάζουμε μερικές από αυτές [4], καθώς και τις ιστοσελίδες στις οποίες υπάρχουν περισσότερες πληροφορίες για αυτές.

Οι περισσότερες (και σίγουρα οι κυριότερες) από αυτές είναι διαθέσιμες σε μορφή αρχείου κειμένου. Σε αυτό το αρχείο χρησιμοποιούνται ετικέτες (labels) για να επισημάνουν την αρχή κάθε εγγραφής, τα διάφορα πεδία της και το τέλος της. Ανάλογα με τη βάση δεδομένων τα πεδία μπορεί να διαφέρουν σημαντικά. Τα βασικά πεδία είναι ένας αναγνωριστικός κωδικός για κάθε ακολουθία-εγγραφή, η ίδια η ακολουθία σαν συμβολοσειρά, πληροφορίες για την πρώτη δημοσίευσή της, για τον οργανισμό από τον οποίο προέρχεται, χημικές ιδιότητες και άλλα. Η SwissProt για παράδειγμα, που περιέχει ακολουθίες πρωτεϊνών, για κάθε εγγραφή έχει αποθηκευμένες και πληροφορίες για την τρισδιάστατη μορφή του μορίου της πρωτεΐνης (Σχήμα 1.4, τριτοταγής μορφή). Στο Σχήμα 2.3 φαίνονται παραδείγματα πεδίων από μια εγγραφή της βάσης SwissProt [13].

Εκτός από την βασική μορφή στην οποία βρίσκεται κάθε βάση, υπάρχει συνήθως διαθέσιμη και μια απλουστευμένη μορφή. Η μορφή αυτή, που ονομάζεται μορφή FASTA από το όνομα ενός αλγορίθμου αναζήτησης (θα παρουσιαστεί αναλυτικά στο Κεφάλαιο 3), περιλαμβάνει μόνο έναν αναγνωριστικό κωδικό, το όνομα της ακολουθίας και την ίδια την ακολουθία σαν συμβολοσειρά. Στην περίπτωση λοιπόν που μας ενδιαφέρει μόνο η ακολουθία και όχι οι υπόλοιπες πληροφορίες που αποθηκεύονται στη βάση δεδομένων, μπορούμε να χρησιμοποιούμε την μορφή FASTA της βάσης. Παράδειγμα της μορφής FASTA για μια βάση με ακολουθίες πρωτεϊνών φαίνεται στο Σχήμα 2.4.

Ένα σημαντικό ζήτημα που αφορά στις βάσεις βιολογικών δεδομένων είναι η αλματώδης αύξηση του μεγέθους τους. Υπολογίζεται ότι το μέγεθος των βάσεων που αποθηκεύονται ακολουθίες DNA αυξάνεται κατά 75% περίπου κάθε χρόνο [10], ενώ και για τις βάσεις πρωτεϊνών το μέγεθος αυξάνεται με εκθετικό ρυθμό. Η εξέλιξη του μεγέθους για δύο από αυτές, της GenBank (ακολουθίες DNA) και της PDB (ακολουθίες πρωτεϊνών) φαίνεται στα Σχήματα 2.5 και 2.6 αντίστοιχα. Για να είναι δυνατή η παρακολούθηση των προσθηκών και μεταβολών σε αυτές, υπάρχει συνεργασία μεταξύ των οργανισμών που τις διαχειρίζονται και αμοιβαίες διασυνδέσεις, έτσι ώστε οι ενημερώσεις σε κάποια από αυτές να προκαλεί ενημερώσεις και στις υπόλοιπες. Επίσης, για την ευκολότερη παρακολούθηση των ενημερώσεων, για τις κυριότερες βάσεις υπάρχουν διαθέσιμες ξεχωριστά οι πιο πρόσφατες ενημερώσεις (συνήθως αυτές του τελευταίου μήνα).

Πίνακας 2.3: Βάσεις βιολογικών δεδομένων.

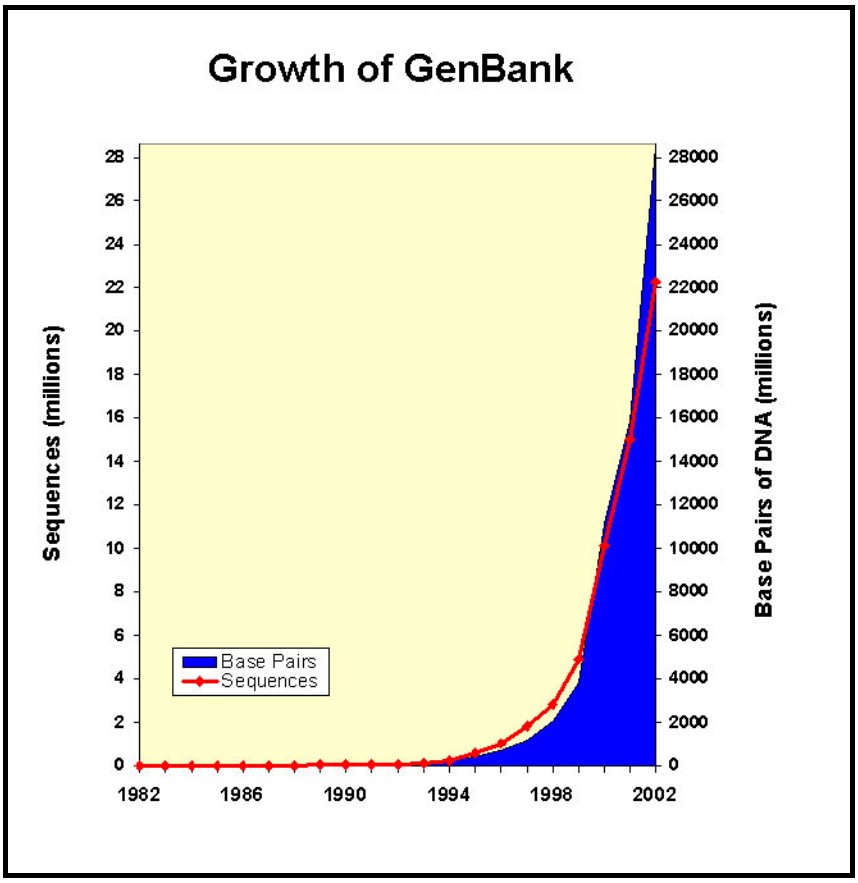
Ονομασία	Τύπος	Ιστοσελίδα
GenBank	DNA	http://www.ncbi.nlm.nih.gov
EMBL European Molecular Biology Library	DNA	http://www.ebi.ac.uk/embl
DDBJ DNA Data Bank of Japan	DNA	http://www.ddbj.nig.ac.jp
GSDB Genome Sequence DataBase	DNA	http://scop.wehi.edu.au/gsdb/
SwissProt	Πρωτεΐνες	http://www.ebi.ac.uk/swissprot/
PIR Protein Information Resource	Πρωτεΐνες	http://pir.georgetown.edu

```
ID   GRAA_HUMAN     STANDARD;      PRT;   262 AA.
AC   P12544;
DT   01-OCT-1989 (Rel. 12, Created)
DT   01-OCT-1989 (Rel. 12, Last sequence update)
DT   15-MAR-2004 (Rel. 43, Last annotation update)
DE   Granzyme A precursor (EC 3.4.21.78) (Cytotoxic T-lymphocyte proteinase
DE   1) (Hanukkah factor) (H factor) (HF) (Granzyme 1) (CTL tryptase)
DE   (Fragmentin 1).
GN   GZMA OR CTLA3 OR HFSP.
OS   Homo sapiens (Human).
OC   Eukaryota; Metazoa; Chordata; Craniata; Vertebrata; Euteleostomi;
OC   Mammalia; Eutheria; Primates; Catarrhini; Hominidae; Homo.
OX   NCBI_TaxID=9606;
RN   [1]
RP   SEQUENCE FROM N.A.
RC   TISSUE=T-cell;
RX   MEDLINE=88125000; PubMed=3257574;
RA   Gershenfeld H.K., Hershberger R.J., Shows T.B., Weissman I.L.;
RT   "Cloning and chromosomal assignment of a human cDNA encoding a T
RT   cell- and natural killer cell-specific trypsin-like serine
RT   protease.";
RL   Proc. Natl. Acad. Sci. U.S.A. 85:1184-1188(1988).
…
SQ   SEQUENCE   262 AA;  28968 MW;  DA87363A0D92BAF4 CRC64;
     MRNSYRFLAS SLSVVVSLLL IPEDVCEKII GGNEVTPHSR PYMVLLSLDR KTICAGALIA
     KDWVLTAAHC NLNKRSQVIL GAHSITREEP TKQIMLVKKE FPYPCYDPAT REGDLKLLQL
     TEKAKINKYV TILHLPKKGD DVKPGTMCQV AGWGRTHNSA SWSDTLREVN ITIIDRKVCN
     DRNHYNFNPV IGMNMVCAGS LRGGRDSCNG DSGSPLLCEG VFRGVTSFGL ENKCGDPRGP
     GVYILLSKKH LNWIIMTIKG AV
//
```

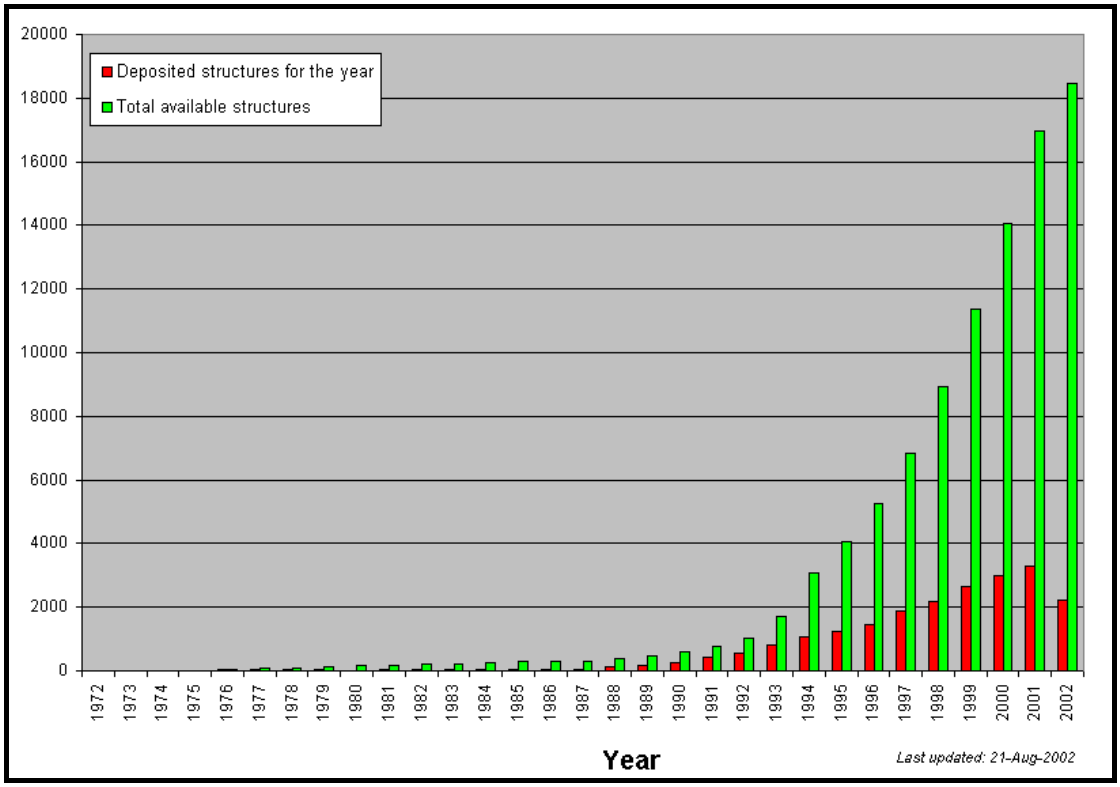
Σχήμα 2.3: Παράδειγμα πεδίων εγγραφής της SwissProt. Διακρίνονται οι αναγνωριστικοί κωδικοί στην αρχή και η ακολουθία σε μορφή συμβολοσειράς στο τέλος.

```
>gi|2501594|sp|Q57997|Y577_METJA PROTEIN MJ0577
MSVMYKKILYPTDFSETAEIALKHVKAFKTLKAEEVILLHVIDEREIKKRDIFSLLLGVAGLNKSVEEFE
NELKNKLTEEAKNKMENIKKELEDVGFKVKDIIVVGIPHEEIVKIAEDEGVDIIIMGSHGKTNLKEILLG
SVTENVIKKSNKPVLVVKRKNS
```

Σχήμα 2.4: Παράδειγμα της μορφής FASTA μιας εγγραφής βάσης δεδομένων.



Σχήμα 2.5: Η ανάπτυξη της βάσης δεδομένων GenBank.



Σχήμα 2.6: Η ανάπτυξη της βάσης δεδομένων PDB.

2.3. Το πρόβλημα της αναζήτησης σε βάσεις βιολογικών δεδομένων

2.3.1. Η σημασία της σύγκρισης ακολουθιών στην υπολογιστική βιολογία

Με τον όρο της σύγκρισης ακολουθιών εννοούμε την εύρεση των τμημάτων των ακολουθιών στα οποία εμφανίζονται ομοιότητες και εκείνα στα οποία εμφανίζονται διαφορές. Από βιολογικής πλευράς η σύγκριση ακολουθιών έχει μεγάλη σημασία, καθώς οι ακολουθίες που έχουν δομικές ομοιότητες είναι πολύ πιθανό ότι θα αντιστοιχούν σε μόρια πρωτεϊνών ή DNA που θα έχουν και λειτουργικές ομοιότητες. Έτσι η εύρεση ομοιοτήτων μεταξύ των ακολουθιών μπορεί να δώσει στους μοριακούς βιολόγους πληροφορίες σχετικά με τις ιδιότητες και τη λειτουργία άλλων ακολουθιών. Από την πλευρά της υπολογιστικής βιολογίας, το πρόβλημα της σύγκρισης ακολουθιών αποτελεί θεμελιώδη διεργασία, καθώς αποτελεί τη βάση για πιο προχωρημένη και πολύπλοκη επεξεργασία των ακολουθιών.
Αν και η βασική ιδέα της σύγκρισης ακολουθιών φαίνεται απλή, υπάρχει ένας σημαντικός αριθμός διακριτών προβλημάτων που σχετίζονται με αυτήν, το καθένα από τα οποία συνήθως έχει διαφορετικές απαιτήσεις για δομές δεδομένων και αλγορίθμους.
Μερικά παραδείγματα προβλημάτων σχετικών με τη σύγκριση ακολουθιών είναι τα παρακάτω [3]:

i. Έχουμε δύο ακολουθίες για το ίδιο αλφάβητο, με ίδιο περίπου μήκος, της τάξης των δεκάδων χιλιάδων συμβόλων η καθεμία. Γνωρίζοντας ότι οι ακολουθίες είναι σχεδόν ίσες, με λίγες μεμονωμένες διαφορές, εισαγωγές, διαγραφές, αντικαταστάσεις που εμφανίζονται με μικρή συχνότητα (για παράδειγμα μία κάθε εκατό σύμβολα), το ζητούμενο είναι να βρεθούν τα σημεία στα οποία βρίσκονται οι διαφορές αυτές.
ii. Έχουμε δύο ακολουθίες για το ίδιο αλφάβητο, με μήκος μερικές εκατοντάδες σύμβολα η καθεμία. Το ζητούμενο είναι αν υπάρχει κάποιο πρόθεμα (prefix) της μίας που να εμφανίζει ομοιότητες με κάποια κατάληξη (suffix) της άλλης, και αν ναι, ποια είναι αυτά.
iii. Έχουμε το ίδιο πρόβλημα με το (ii), αλλά σε αυτή την περίπτωση έχουμε μερικές εκατοντάδες ακολουθίες, και πρέπει να συγκρίνουμε καθεμιά με όλες τις υπόλοιπες. Γνωρίζουμε επιπλέον, ότι τα ζεύγη ακολουθιών που προκύπτουν, στην πλειοψηφία τους δεν εμφανίζουν την απαιτούμενη ομοιότητα.
iv. Έχουμε δύο ακολουθίες για το ίδιο αλφάβητο, με μήκος μερικές εκατοντάδες σύμβολα η καθεμία. Θέλουμε να βρούμε αν υπάρχει μια συνεχής υποακολουθία της μίας που να παρουσιάζει ομοιότητα με μια αντίστοιχη της άλλης.
v. Έχουμε το ίδιο πρόβλημα με το (iv), αλλά αυτή τη φορά έχουμε μία ακολουθία που πρέπει να συγκριθεί με χιλιάδες άλλες.

Το πρόβλημα της αναζήτησης σε βάσεις δεδομένων για ακολουθίες με υψηλή ομοιότητα, είναι το πρόβλημα (v) από την παραπάνω λίστα και αυτό που θα μας απασχολήσει στη συνέχεια αυτής της εργασίας.

2.3.2. Ολική και τοπική αντιστοίχιση

Εκτός από τις αλλαγές που μπορεί να συμβούν σε τμήματα των ακολουθιών όπως τις είδαμε στην Ενότητα 2.1.5, υπάρχουν και τμήματα των ακολουθιών που επαναλαμβάνονται όχι μόνο σε ακολουθίες του ίδιου ζωντανού οργανισμού (για παράδειγμα σε διαφορετικά γονίδια ή συγγενείς πρωτεΐνες) αλλά και μεταξύ ακολουθιών που προέρχονται από διαφορετικούς οργανισμούς. Έτσι λοιπόν έχουμε συχνά ακολουθίες που συνολικά διαφέρουν σημαντικά, όμως σε κάποια επιμέρους τμήματα παρουσιάζουν υψηλή ομοιότητα. Σε αυτές τις περιπτώσεις, η ολική αντιστοίχιση των ακολουθιών δε θα μπορούσε να εντοπίσει αυτές τις ομοιότητες.
Για παράδειγμα, έστω ότι συγκρίνουμε μεγάλες ακολουθίες DNA, για τις οποίες δεν γνωρίζουμε τι σχέση υπάρχει μεταξύ τους. Επειδή αν υπάρχουν ομοιότητες αυτές θα βρίσκονται σε περιορισμένα τμήματα, η ολική αντιστοίχιση δε θα μπορέσει να τις εντοπίσει και το αποτέλεσμα θα είναι μια χαμηλή ολική ομοιότητα. Αντίθετα, η τοπική αντιστοίχιση είναι πιο κατάλληλη για τον εντοπισμό αυτών των ομοιοτήτων.
Στην περίπτωση λοιπόν που έχουμε ακολουθίες με άγνωστη μεταξύ τους ομοιότητα (για παράδειγμα πρωτεΐνες από διαφορετικές οικογένειες) οι τοπικές αντιστοιχίσεις δίνουν καλύτερα αποτελέσματα, σε ότι αφορά των εντοπισμό των ομοιοτήτων [2]. Στην περίπτωση που μας απασχολεί, η πλειοψηφία των ζευγών ακολουθιών περιλαμβάνουν ακολουθίες που δεν έχουν κάποια σχέση μεταξύ τους. Επομένως είναι προτιμότερη η χρήση τοπικών αντιστοιχίσεων.

2.3.3. Αποδοτικότητα και αποτελέσματα της αναζήτησης

Είδαμε στην Ενότητα 2.2 ότι οι βάσεις βιολογικών δεδομένων αποτελούνται κατά κανόνα από αρκετές χιλιάδες ακολουθίες. Το ιδιαίτερα μεγάλο μέγεθός τους μπορεί να επηρεάσει τόσο την αποδοτικότητα ενός αλγόριθμου αναζήτησης όσο και τα αποτελέσματά του.
Εύκολα γίνεται αντιληπτό ότι για να είναι αποδοτικός ένας αλγόριθμος αναζήτησης πρέπει να είναι γρήγορος. Και η ταχύτητά του επιτυγχάνεται με την χαμηλή χρονική πολυπλοκότητα ως προς το μέγεθος της βάσης δεδομένων. Εκείνο που δεν γίνεται εύκολα αντιληπτό είναι το πως το μέγεθος της βάσης δεδομένων επηρεάζει τα αποτελέσματα ενός αλγόριθμου. Αν και η πιθανότητα δύο τυχαίες ακολουθίες να παρουσιάζουν μεγάλη ολική ομοιότητα είναι πολύ μικρή, όταν έχουμε σε μία βάση δεδομένων πολλές χιλιάδες ακολουθίες και χρησιμοποιούμε τοπικές αντιστοιχίσεις, η πιθανότητα να προκύψουν τυχαίες ομοιότητες είναι σημαντική. Ένας αλγόριθμος αναζήτησης λοιπόν θα πρέπει να παρέχει και έναν μηχανισμό αξιολόγησης των αποτελεσμάτων, που θα απορρίπτει αυτά τα αποτελέσματα. Ο μηχανισμός αυτός θα πρέπει να λαμβάνει υπ' όψιν παραμέτρους όπως τα μεγέθη των ακολουθιών και της βάσης δεδομένων, τη μέθοδο βαθμολόγησης που χρησιμοποιήθηκε, έτσι ώστε να αποφασίζει αν μια τιμή ομοιότητας μεταξύ ακολουθιών μπορεί να προέκυψε τυχαία.

2.4. Ανακεφαλαίωση

Παρουσιάσαμε στο κεφάλαιο αυτό μερικές από τις έννοιες και τις αρχές της υπολογιστικής βιολογίας, τις οποίες θα χρησιμοποιήσουμε στη συνέχεια αυτής της εργασίας. Παρουσιάσαμε επίσης και μια περιγραφή του προβλήματος της αναζήτησης σε βάσεις βιολογικών δεδομένων, που αποτελεί και το πρόβλημα που θα μελετήσουμε στη συνέχεια. Ιδιαίτερη αναφορά έγινε στις παραμέτρους εκείνες που θα καθορίσουν τις επιλογές μας τόσο ως προς το θεωρητικό μέρος της επιλογής αλγορίθμου όσο και στις λεπτομέρειες της υλοποίησης.

ΚΕΦΑΛΑΙΟ 3

Αλγόριθμοι σύγκρισης βιολογικών ακολουθιών

Στο κεφάλαιο αυτό παρουσιάζουμε αναλυτικά τους αλγόριθμους που χρησιμοποιούνται για τη σύγκριση βιολογικών ακολουθιών. Ξεκινώντας από τους αλγόριθμους δυναμικού προγραμματισμού, περιγράφουμε τους αλγόριθμους ολικής και μερικής αντιστοίχισης και συνεχίζουμε με τον αλγόριθμο FASTA, που χρησιμοποιεί ευριστικές μεθόδους. Τέλος αναφέρουμε και τον αλγόριθμο BLAST, τον οποίο παρουσιάζουμε αναλυτικά στο Κεφάλαιο 4.

3.1. Αλγόριθμος ολικής αντιστοίχισης δυναμικού προγραμματισμού

Σε αυτή την ενότητα παρουσιάζουμε τον αλγόριθμο δυναμικού προγραμματισμού που χρησιμοποιείται για την ολική σύγκριση δύο ακολουθιών [3]. Η τεχνική του δυναμικού προγραμματισμού συνίσταται στην επίλυση ενός προβλήματος, χρησιμοποιώντας τις λύσεις για μικρότερες περιπτώσεις του ίδιου προβλήματος τις οποίες έχουμε υπολογίσει και αποθηκεύσει νωρίτερα.
Το πρώτο βήμα είναι να υπολογίσουμε την ομοιότητα μεταξύ των ακολουθιών. Στη συνέχεια, χρησιμοποιούμε τη λύση του πρώτου βήματος για να κατασκευάσουμε μια βέλτιστη αντιστοίχιση για την οποία έχουμε και την τιμή της ομοιότητας που υπολογίσαμε.

3.1.1. Ολική ομοιότητα

Η τεχνική του δυναμικού προγραμματισμού εφαρμόζεται με την εύρεση της ομοιότητας για όλα τα προθέματα των δύο ακολουθιών, ξεκινώντας από τα μικρότερα προθέματα και χρησιμοποιώντας τις τιμές που έχουμε ήδη υπολογίσει για να υπολογίσουμε την ομοιότητα για μεγαλύτερα προθέματα.
Για την αποθήκευση των τιμών που υπολογίζουμε χρησιμοποιούμε έναν πίνακα. Αν έχουμε δύο ακολουθίες s_1 και s_2 με μήκη n_1 και n_2 αντίστοιχα, τότε έχουμε n_1+1 πιθανά προθέματα για την ακολουθία s_1, συμπεριλαμβανομένης και της κενής συμβολοσειράς και n_2+1 για την s_2. Για την αποθήκευση επομένως των αποτελεσμάτων μπορούμε να χρησιμοποιήσουμε έναν πίνακα $(n_1+1)\times(n_2+1)$, όπου το στοιχείο $[i, j]$ του πίνακα μας δίνει την ομοιότητα μεταξύ $s_1[1..i]$ και $s_2[1..j]$.
Ξεκινάμε αρχικοποιώντας την πρώτη γραμμή και στήλη του πίνακα με πολλαπλάσια του κόστους κενού. Αυτό γιατί η μόνη δυνατή αντιστοίχιση αν η μια ακολουθία είναι κενή, είναι να προσθέσουμε τόσα κενά όσα και το μήκος της άλλης ακολουθίας. Αν το μήκος της μη κενής ακολουθίας είναι n τότε η αντιστοίχιση αυτή έχει βαθμολόγηση $g\times n$, όπου g το κόστος αντιστοίχισης του κενού που χρησιμοποιούμε.

Για κάθε άλλο στοιχείο $[i, j]$ του πίνακα, μπορούμε να υπολογίσουμε την τιμή ελέγχοντας μόνο τρεις από τις προηγούμενα υπολογισμένες τιμές: $[i\text{-}1, j]$, $[i\text{-}1, j\text{-}1]$, $[i, j\text{-}1]$. Αυτό γίνεται γιατί υπάρχουν μόνο τρεις τρόποι να αντιστοιχίσουμε τα $s_1[1..i]$, $s_2[i..j]$ και καθένας από αυτούς χρησιμοποιεί μια από αυτές τις τιμές. Συγκεκριμένα, οι επιλογές που έχουμε είναι οι εξής:

- Αντιστοίχιση των $s_1[1..i\text{-}1]$, $s_2[1..j\text{-}1]$ και του $s_1[i]$ με το $s_2[j]$.
- Αντιστοίχιση των $s_1[1..i]$, $s_2[1..j\text{-}1]$ και του $s_2[j]$ με κενό.
- Αντιστοίχιση των $s_1[1..i\text{-}1]$, $s_2[1..j]$ και του $s_1[i]$ με κενό.

Για να υπολογίσουμε λοιπόν την ομοιότητα χρησιμοποιούμε τις τιμές που έχουμε αποθηκεύσει για μικρότερα προθέματα.. Αυτό είναι εφικτό αν προσέξουμε τη σειρά με την οποία υπολογίζουμε τις τιμές του πίνακα. Εύκολα παρατηρούμε ότι για να έχουμε κάθε φορά διαθέσιμες τις τρεις τιμές που χρειαζόμαστε, αρκεί να υπολογίζονται τα στοιχεία του πίνακα είτε για κάθε γραμμή με τη σειρά και από αριστερά προς τα δεξιά σε κάθε γραμμή, είτε για κάθε στήλη με τη σειρά και από πάνω προς τα κάτω σε κάθε στήλη. Σε αυτή την περίπτωση κάθε στοιχείο $A[i, j]$ του πίνακα υπολογίζεται ως εξής:

$$A[i, j] = \max\begin{cases} A[i, j-1] + g \\ A[i-1, j-1] + S(i, j) \\ A[i-1, j] + g \end{cases}$$

όπου $S(i, j)$ είναι η βαθμολόγηση μεταξύ των συμβόλων $s_1[i]$, $s_2[j]$ και g το κόστος αντιστοίχισης του κενού.
Τέλος, για κάθε στοιχείο του πίνακα σχεδιάζουμε ένα βέλος που μας δείχνει από ποιο προηγούμενο στοιχείο υπολογίσαμε την τιμή του. Αν και σε αυτό το σημείο, η χρήση αυτών των βελών δε φαίνεται να έχει κάποια ιδιαίτερη χρησιμότητα, πέραν της κατανόησης της επιλογής του στοιχείου που χρησιμοποιήθηκε, είναι σημαντική και για την κατανόηση του αλγόριθμου βέλτιστης αντιστοίχισης, που θα περιγράψουμε στην Ενότητα 3.1.2.
Μετά την ολοκλήρωση του αλγόριθμου, η τιμή της ολικής ομοιότητας είναι το στοιχείο $[n_1+1, n_2+1]$ του πίνακα, καθώς είναι αυτό που αντιστοιχεί στη σύγκριση των προθεμάτων που είναι ίσα με το μήκος των ακολουθιών.
Στο Σχήμα 3.1 φαίνεται ο αλγόριθμος που περιγράψαμε σε ψευδοκώδικα [3]. Το αποτέλεσμα εξαρτάται από την παράμετρο g που καθορίζει το κόστος αντιστοίχισης του κενού και τη συνάρτηση S που δίνει την βαθμολόγηση για το ζεύγος συμβόλων.

Παράδειγμα 3.1

Έστω οι ακολουθίες s_1=*AGC,* s_2=*AAAC. Χρησιμοποιούμε g = -2 και τη συνάρτηση βαθμολόγησης* S(a, b) = *1, αν* a = b *και* S(a, b) = *-1, αν* a≠b. *Ο πίνακας που προκύπτει εφαρμόζοντας τον παραπάνω αλγόριθμο φαίνεται στο Σχήμα 3.2, όπου για ευκολία έχουμε σημειώσει και τις ακολουθίες, έτσι ώστε να γίνεται ευκολότερα αντιληπτό για ποια προθέματα υπολογίζονται οι τιμές.*

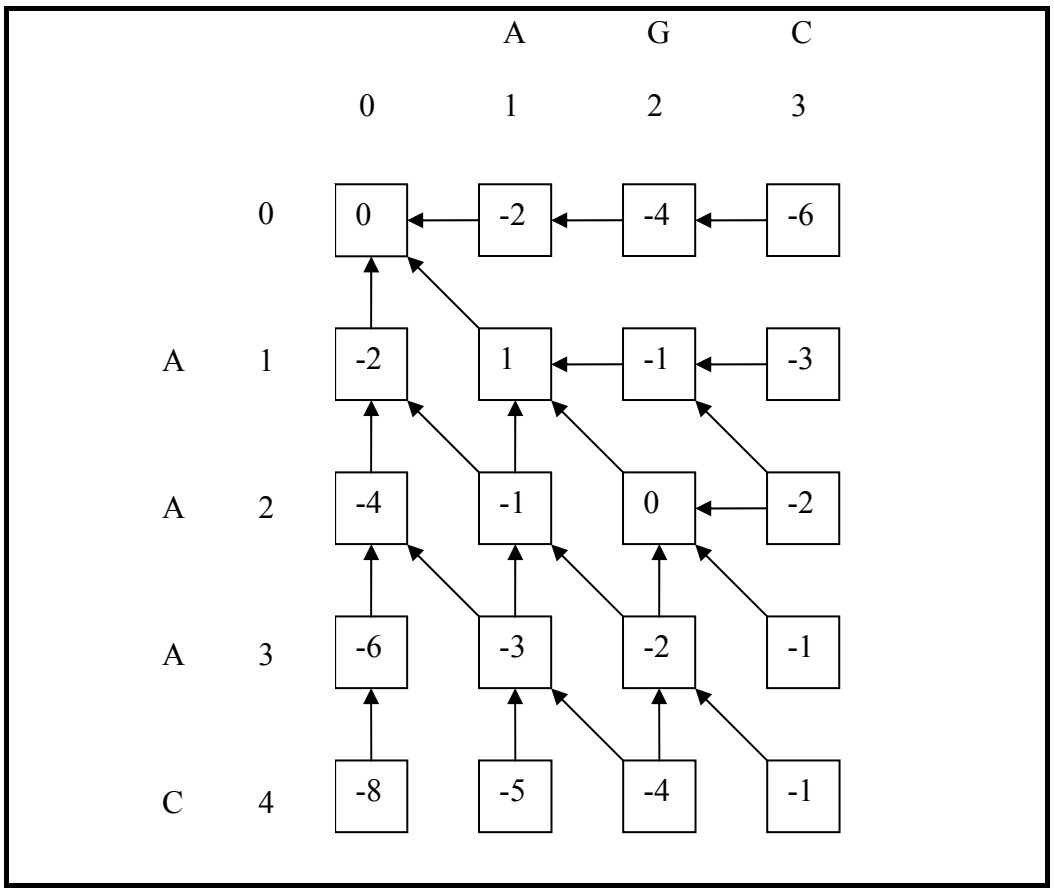
Σύμφωνα με τον πίνακα λοιπόν, η ομοιότητα μεταξύ των δύο ακολουθιών είναι ίση με

$$sim(s_1, s_2) = a[4, 3] = -1.$$

Algorithm *Similarity*
input: sequences s_1, s_2
output: similarity between s_1,s_2

$n_1 \leftarrow |s_1|$
$n_2 \leftarrow |s_2|$
for $i \leftarrow 0$ **to** n_1 **do**
  $a[i, 0] \leftarrow i \times g$
for $j \leftarrow 0$ **to** n_2 **do**
  $a[0, j] \leftarrow j \times g$
for $i \leftarrow 1$ **to** n_1 **do**
  for $j \leftarrow 1$ **to** n_2 **do**
    $a[i, j] \leftarrow \max (a[i\text{-}1, j] + g, a[i\text{-}1, j\text{-}1] + S(i, j), a[i, j\text{-}1] + g)$
return $a[n_1, n_2]$

Σχήμα 3.1: Αλγόριθμος δυναμικού προγραμματισμού για την εύρεση της ομοιότητας μεταξύ δύο ακολουθιών.



Σχήμα 3.2: Πίνακας για τον υπολογισμό της ομοιότητας για το Παράδειγμα 3.1.

3.1.2. Βέλτιστες αντιστοιχίσεις

Έχοντας κατασκευάσει τον πίνακα που χρησιμοποιεί ο αλγόριθμος για τον υπολογισμό της ομοιότητας, μπορούμε πολύ εύκολα να κατασκευάσουμε την βέλτιστη αντιστοίχιση για τις δύο ακολουθίες. Το μόνο που χρειάζεται να κάνουμε είναι να ξεκινήσουμε από το στοιχείο $[n_1+1, n_2+1]$ του πίνακα και να ακολουθήσουμε τα βέλη μέχρι να φτάσουμε στο στοιχείο [1, 1]. Κάθε βέλος που χρησιμοποιούμε σε αυτή τη διαδικασία μας δίνει και μία στήλη την αντιστοίχιση. Ο τρόπος που καθορίζεται η στήλη αυτή εξαρτάται από τον τύπο του βέλους. Έτσι λοιπόν για κάθε στοιχείο $[i, j]$ του πίνακα, έχουμε τρεις επιλογές για τον τύπο του βέλους που ερμηνεύονται σαν τρεις διαφορετικές αντιστοιχίσεις για τα στοιχεία $s_1[i]$, $s_2[j]$:

- Οριζόντιο βέλος: το σύμβολο $s_2[j]$ αντιστοιχίζεται με κενό στην ακολουθία s_1.
- Διαγώνιο βέλος: το σύμβολο $s_1[i]$ αντιστοιχίζεται με το σύμβολο $s_2[j]$.
- Κάθετο βέλος: το σύμβολο $s_1[i]$ αντιστοιχίζεται με κενό στην ακολουθία s_2.

Για την εύρεση της βέλτιστης αντιστοίχισης χρησιμοποιούμε έναν αναδρομικό αλγόριθμο, στον οποίο ο έλεγχος των βελών μπορεί να αντικατασταθεί με έναν απλό έλεγχο μεταξύ των στοιχείων του πίνακα που μας ενδιαφέρουν. Στο Σχήμα 3.3 παρουσιάζουμε τον ψευδοκώδικα για τον αλγόριθμο αυτό. Ο αλγόριθμος αυτός χρησιμοποιεί τον πίνακα που κατασκευάζει ο αλγόριθμος υπολογισμού της ομοιότητας και δίνει σαν αποτέλεσμα δύο ακολουθίες συμβόλων, που μπορεί να περιέχουν κενά, στις οποίες αποθηκεύεται η αντιστοίχιση μεταξύ των ακολουθιών, καθώς και το μήκος της αντιστοίχισης.

```
Algorithm Align
    input: indices i, j, array a given by algorithm Similarity
    output: alignment in align_s1, align_s2 and length in len
    if i = 0 and j = 0 then
        len ← 0
    else if i > 0 and a[i, j] = a[i-1, j] + g then
        Align(i-1, j, len)
        len ← len+1
        align_s1[len] ← s1[i]
        align_s2[len] ← -
    else if i > 0 and a[i, j] = a[i-1, j-1] + S(i, j) then
        Align(i-1, j-1, len)
        len ← len + 1
        align_s1[len] ← s1[i]
        align_s2[len] ← s2[i]
    else//has to be j > 0 and a[i, j] = a[i, j-1] + g
        Align(i, j-1, len)
        len ← len + 1
        align_s1[len] ← -
        align_s2[len] ← s2[i]
```

Σχήμα 3.3: Αναδρομικός αλγόριθμος κατασκευής βέλτιστης αντιστοίχισης.

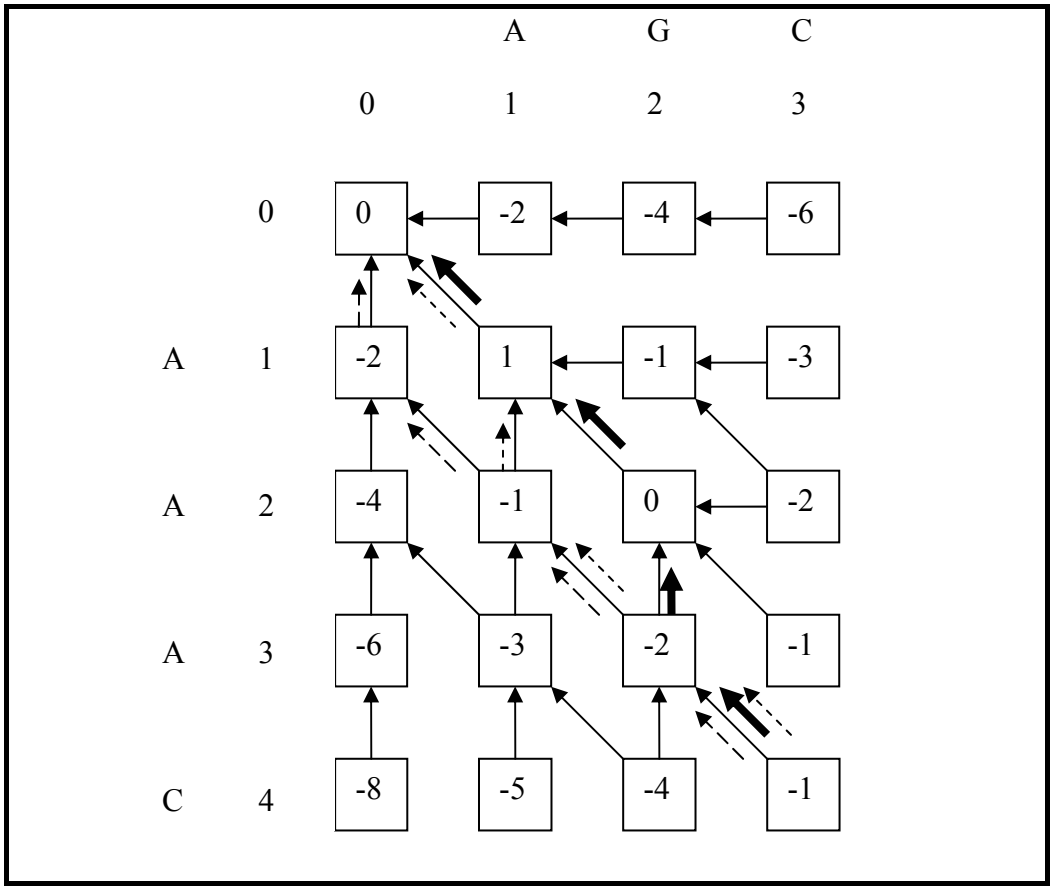
Κατά τη σύγκριση δύο ακολουθιών, υπάρχει η πιθανότητα να έχουμε περισσότερες από μία βέλτιστες αντιστοιχίσεις. Αυτό μπορεί να γίνει αντιληπτό, παρατηρώντας τον πίνακα, όταν έχουμε περισσότερες από μία διαδρομές από το στοιχείο $[n_1+1, n_2+1]$ προς το στοιχείο [1, 1]. Ο αλγόριθμος σε αυτή την περίπτωση επιστέφει μόνο μία αντιστοίχιση. Για να γίνει αυτό, χρησιμοποιείται μια σειρά προτίμησης για τα είδη των βελών. Συγκεκριμένα, ο αλγόριθμος του Σχήματος 3.3 ακολουθεί τη σειρά: κάθετο βέλος, διαγώνιο, οριζόντιο. Η σειρά προτίμησης μπορεί να μεταβληθεί εύκολα, αλλάζοντας τη σειρά των συνθηκών ελέγχου στον αλγόριθμο.

Παράδειγμα 3.2

Στο Σχήμα 3.4 σημειώνουμε τρεις διαφορετικές βέλτιστες αντιστοιχίσεις για τις ακολουθίες του Παραδείγματος 3.1. Οι αντιστοιχίσεις αυτές είναι:

AAAC	AAAC	AAAC
AG-C	-AGC	A-GC

Η πρώτη αντιστοίχιση είναι και αυτή που επιλέγεται από τον αλγόριθμο.



Σχήμα 3.4: Βέλτιστες αντιστοιχίσεις για το Παράδειγμα 3.2. Με έντονα βέλη η αντιστοίχιση που επιλέγεται από τον αλγόριθμο.

3.2. Αλγόριθμος τοπικής αντιστοίχισης δυναμικού προγραμματισμού

Όπως είδαμε και στην Ενότητα 2.1.3, μια τοπική αντιστοίχιση μεταξύ δύο ακολουθιών s_1, s_2 είναι μια αντιστοίχιση μεταξύ δύο υποακολουθιών των s_1 και s_2, η οποίες πριν την αντιστοίχιση είναι συνεχείς και χωρίς κενά. Σε αυτή την ενότητα παρουσιάζουμε έναν αλγόριθμο δυναμικού προγραμματισμού για την εύρεση της τοπικής αντιστοίχισης με την υψηλότερη βαθμολόγηση για δύο ακολουθίες.
Ο αλγόριθμος αυτός αποτελεί παραλλαγή του βασικού αλγορίθμου ολικής αντιστοίχισης που παρουσιάσαμε στην Ενότητα 3.1. Η βασική δομή που χρησιμοποιούμε είναι πάλι ένας πίνακας $(n_1+1)\times(n_2+1)$, στην περίπτωση αυτή όμως κάθε στοιχείο $[i, j]$ αντιπροσωπεύει την βαθμολόγηση μεταξύ μιας κατάληξης του $s_1[1..i]$ και μιας κατάληξης του $s_2[1..j]$. Τα στοιχεία της πρώτης γραμμής και της πρώτης στήλης του πίνακα αρχικοποιούνται στην τιμή μηδέν.
Για κάθε στοιχείο $[i, j]$ του πίνακα υπάρχει η αντιστοίχιση των κενών καταλήξεων των $s_1[1..i]$ και $s_2[1..j]$, η οποία έχει βαθμολόγηση μηδέν. Επομένως όλα τα στοιχεία του πίνακα θα έχουν τιμές μεγαλύτερες ή ίσες του μηδενός.
Μετά από την αρχικοποίηση, τα στοιχεία του πίνακα υπολογίζονται με τον ίδιο τρόπο, όπου η τιμή του στοιχείου $[i, j]$ εξαρτάται από τις τιμές τριών άλλων στοιχείων που έχουμε προηγούμενα υπολογίσει.

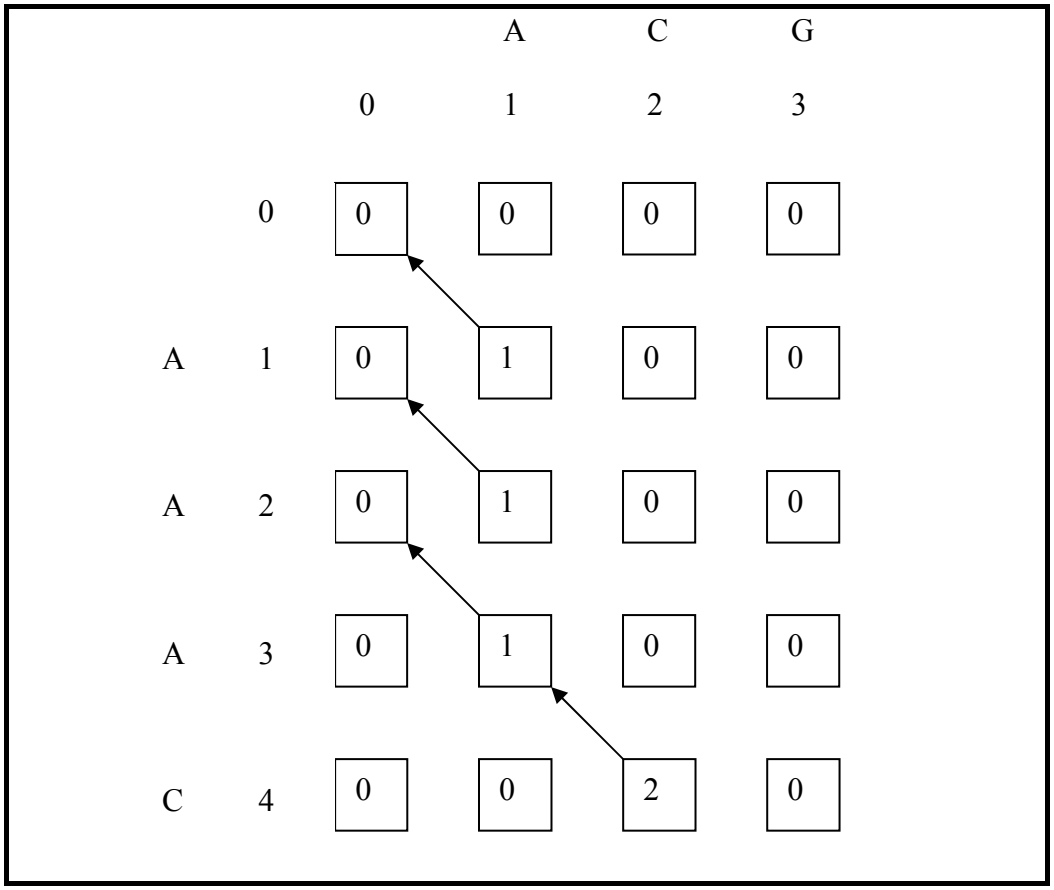
$$A[i,j] = \max\begin{cases} A[i,j-1]+g \\ A[i-1,j-1]+S(i,j) \\ A[i-1,j]+g \\ 0 \end{cases}$$

Στην περίπτωση αυτή έχουμε και μια τέταρτη επιλογή, αυτή της κενής αντιστοίχισης. Όταν χρησιμοποιούμε κάποιο από τα τρία στοιχεία του πίνακα που έχουμε υπολογίσει, σχεδιάζουμε το κατάλληλο βέλος, ενώ όταν χρησιμοποιούμε την κενή αντιστοίχιση δεν σχεδιάζουμε κανένα βέλος.
Στο τέλος, χρειάζεται απλά να βρούμε την μέγιστη τιμή του πίνακα, η οποία αποτελεί και την βαθμολόγηση της βέλτιστης τοπικής αντιστοίχισης. κάθε στοιχείο που περιέχει αυτή την τιμή μπορεί να χρησιμοποιηθεί για να βρεθεί μια βέλτιστη τοπική αντιστοίχιση. Για να κατασκευάσουμε μια τέτοια αντιστοίχιση ξεκινάμε, όπως και στην ολική αντιστοίχιση, από το στοιχείο αυτό και ακολουθούμε τα βέλη, μέχρι να φτάσουμε σε στοιχείο που δεν έχει βέλος για να ακολουθήσουμε ή (ισοδύναμα) έχει τιμή μηδέν.

Παράδειγμα 3.3

Έστω οι ακολουθίες s_1=*ACG,* s_2=*AAAC,* g = *-2 και η συνάρτηση βαθμολόγησης* S(a, b) = *1, αν* a = b *και* S(a, b) = *-1, αν* a≠b. *Κατασκευάζουμε τον πίνακα που φαίνεται στο Σχήμα 3.5. Η μέγιστη τιμή που βρίσκουμε στον πίνακα είναι το στοιχείο* A*[4, 2] = 2. Από το στοιχείο αυτό ξεκινάμε για τον υπολογισμό της βέλτιστης τοπικής αντιστοίχισης, η οποία είναι για τις υποακολουθίες* s_1'=*AC,* s_2'=*AC*

των αρχικών ακολουθιών. Η μεταξύ τους αντιστοίχιση είναι προφανής.



Σχήμα 3.5: Πίνακας για την εύρεση της βέλτιστης τοπικής αντιστοίχισης για το Παράδειγμα 3.3.

3.3. Χρονική πολυπλοκότητα των αλγορίθμων δυναμικού προγραμματισμού

Όπως είδαμε στην Ενότητα 2.3.3, η υπολογιστική πολυπλοκότητα ενός αλγορίθμου έχει πολύ μεγάλη σημασία. Οι αλγόριθμοι δυναμικού προγραμματισμού για ολική και τοπική αντιστοίχιση αποτελούνται ουσιαστικά από τα ίδια δύο μέρη.

Στο πρώτο μέρος κατασκευάζεται ένας πίνακας με διαστάσεις $(n_1+1)\times(n_2+1)$, όπου n_1, n_2 τα μήκη των ακολουθιών. Για κάθε στοιχείο του πίνακα εκτελείται σταθερός αριθμός από λειτουργίες, επομένως το πρώτο μέρος του αλγόριθμου εκτελείται σε χρόνο $O(n_1 \times n_2)$ [3].

Στο δεύτερο μέρος κατασκευάζεται η αντιστοίχιση μεταξύ των ακολουθιών. Αυτή η λειτουργία εκτελείται σε χρόνο $O(L)$, όπου L το μήκος της αντιστοίχισης. Στην περίπτωση της ολικής αντιστοίχισης, είναι

$$L \leq n_1 + n_2,$$

ενώ το ίδιο όριο ισχύει και για τις τοπικές αντιστοιχίσεις, εφόσον οι υποακολουθίες που αντιστοιχίζονται είναι σίγουρα μικρότερες από τις αρχικές ακολουθίες. Επομένως το δεύτερο μέρος εκτελείται σε χρόνο $O(n_1+n_2)$ [3].
Ας εξετάσουμε τώρα την περίπτωση της αναζήτησης σε βάσεις βιολογικών δεδομένων. Ο αλγόριθμος (είτε ολικής είτε τοπικής αντιστοίχισης) θα πρέπει να εκτελεστεί για κάθε ζεύγος ακολουθιών, που αποτελείται από την ακολουθία που εξετάζουμε και μία ακολουθία από τη βάση. Αν η ακολουθία που εξετάζουμε έχει μήκος m και οι ακολουθίες της βάσης δεδομένων $n_1,n_2,\ldots,n_n$, ο συνολικός χρόνος εκτέλεσης για το πρώτο μέρος θα είναι:

$$O(m\times n_1+m\times n_2+\ldots+m\times n_n)=$$
$$O(m\times(n_1+n_2+\ldots+n_n))=$$
$$O(m\times N),$$

όπου N ο συνολικός αριθμός συμβόλων στη βάση δεδομένων.
Για το δεύτερο μέρος, ο χρόνος εκτέλεσης είναι:

$$O(m+n_1+m+n_2+\ldots+m+n_n)=$$
$$\mathrm{O}(n\times m+N),$$

όπου n ο συνολικός αριθμός ακολουθιών στη βάση δεδομένων.
Βλέπουμε λοιπόν ότι η πολυπλοκότητα του πρώτου μέρους, άρα και όλου του αλγόριθμου εφόσον είναι μεγαλύτερη από αυτή του δεύτερου μέρους, είναι πολλαπλάσια του μεγέθους της βάσης δεδομένων, κατά ένα παράγοντα ίσο με το μήκος της ακολουθίας που εξετάζουμε. Με δεδομένο το μεγάλο μέγεθος των βάσεων βιολογικών δεδομένων και το αρκετά μεγάλο μήκος των βιολογικών ακολουθιών (μερικές εκατοντάδες σύμβολα), συμπεραίνουμε ότι οι αλγόριθμοι δυναμικού προγραμματισμού δεν είναι αποδοτικοί για αναζήτηση σε βάσεις βιολογικών δεδομένων.

3.4. Ο αλγόριθμος FASTA

Όπως είδαμε στην Ενότητα 3.3, οι αλγόριθμοι δυναμικού προγραμματισμού έχουν μεγάλη χρονική πολυπλοκότητα για να χρησιμοποιηθούν στην πράξη για αναζήτηση σε βάσεις δεδομένων. Αυτή η πρακτική αδυναμία έδωσε ώθηση στην ανάπτυξη νέων αλγορίθμων, οι οποίοι θα έπρεπε να είναι πιο αποτελεσματικοί για το πρόβλημα που εξετάζουμε.
Ένας από αυτούς τους αλγόριθμους είναι ο FASTA (FAST-All) [14]. Πρόκειται για έναν ευριστικό αλγόριθμο, που χρησιμοποιείται για τον υπολογισμό κατά προσέγγιση τοπικών αντιστοιχίσεων και ομοιοτήτων μεταξύ δύο ακολουθιών. Η διαδικασία του επαναλαμβάνεται για κάθε ακολουθία που βρίσκεται στη βάση δεδομένων. Ο αλγόριθμος αυτός δημιουργήθηκε από τους D. Lipman και W. Pearson.
Σε ότι αφορά τη δομή και τη λειτουργία του αλγόριθμου, ξεκινάει με την επιλογή μιας παραμέτρου που ονομάζεται *ktup*. Στο πρώτο βήμα του αλγόριθμου εντοπίζονται τα ζεύγη (i, j), τέτοια ώστε μια συνεχής υποακολουθία μήκους *ktup* στη μία ακολουθία που αρχίζει στο σημείο i να

ταυτίζεται με μία αντίστοιχη που αρχίζει στο σημείο *j* της άλλης ακολουθίας. Οι προτεινόμενες τιμές για την παράμετρο *ktup* είναι 6 για ακολουθίες DNA και 2 για πρωτεΐνες. Για αυτές τις μικρές τιμές της *ktup* τα παραπάνω ζεύγη μπορούν να βρεθούν χρησιμοποιώντας μια συνάρτηση κατακερματισμού (hash function) για της υποακολουθίες μήκους ktup της ακολουθίας εισόδου ή και των ακολουθιών της βάσης δεδομένων. Για τις ακολουθίες της βάσης αυτό μπορεί να γίνει πριν από τη δημοσίευση της βάσης, αρκεί να υπάρχει διαθέσιμος ο κατάλληλος χώρος για τον πίνακα κατακερματισμού. Σε αυτή την περίπτωση, ο αλγόριθμος ψάχνει κάθε υποακολουθία μήκους *ktup* της ακολουθίας εισόδου σε αυτό τον πίνακα. Αν δεν υπάρχει διαθέσιμος χώρος για τον πίνακα κατακερματισμού της βάσης, τότε ακολουθείτε η αντίστροφη διαδικασία: οι υποακολουθίες της ακολουθίας εισόδου τοποθετούνται στον πίνακα κατακερματισμού και ο αλγόριθμος ψάχνει σε αυτό τον πίνακα για τις υποακολουθίες που προκύπτουν από τη βάση. Ο αλγόριθμος FASTA στην πράξη χρησιμοποιεί τη δεύτερη μέθοδο.

Κάθε ζεύγος (*i*, *j*) που εντοπίζετε στο πρώτο βήμα αντιστοιχεί σε ένα διάστημα μήκους ktup στη διαγώνιο (*i-j*) του πίνακα δυναμικού προγραμματισμού. Η αρίθμηση γίνεται θεωρώντας μηδενική την κύρια διαγώνιο, θετικές όσες βρίσκονται πάνω από αυτή και αρνητικές όσες βρίσκονται κάτω από την κύρια. Στη συνέχεια ο αλγόριθμος FASTA εντοπίζει σειρές από ταυτιζόμενα ζεύγη, που βρίσκονται στην ίδια διαγώνιο, και επιλέγει τις δέκα καλύτερες. Η αξιολόγηση αυτών των σειρών γίνεται δίνοντας σε κάθε ταυτιζόμενο ζεύγος μία θετική βαθμολόγηση και σε κάθε ενδιάμεσο κενό μία αρνητική, η οποία μειώνεται όσο μεγαλώνει η απόσταση μεταξύ των ταυτιζόμενων ζευγών. Η βαθμολόγηση κάθε σειράς προκύπτει από το άθροισμα των βαθμολογήσεων για τα ζεύγη και τα κενά και ο αλγόριθμος επιλέγει τις σειρές με τις δέκα μεγαλύτερες βαθμολογήσεις.

Κάθε μία από τις δέκα σειρές που εντοπίζονται με τον παραπάνω τρόπο, αντιπροσωπεύει μια αντιστοίχιση μεταξύ συνεχών υποακολουθιών. Επειδή οι αντιστοιχίσεις αυτές προέρχονται από μία διαγώνιο του πίνακα δυναμικού προγραμματισμού, αντιπροσωπεύουν αντιστοιχίσεις χωρίς κενά. Στη συνέχεια, οι δέκα αυτές αντιστοιχίσεις ελέγχονται για την ύπαρξη ενός ζεύγους υποακολουθιών που δίνει μια αντιστοίχιση με μέγιστη βαθμολόγηση, χρησιμοποιώντας αυτή τη φορά για την περίπτωση των πρωτεϊνών, έναν από τους πίνακες βαθμολόγησης (Ενότητα 2.1.5). Η βαθμολόγηση της καλύτερης αντιστοίχισης, που εντοπίζεται σε αυτό το βήμα, ονομάζεται *init1*.

Στο τρίτο βήμα ο αλγόριθμος FASTA επιχειρεί να συνδυάσει όσες από αυτές τις αντιστοιχίσεις έχουν βαθμολόγηση πάνω από μία τιμή αποκοπής, σε μία μεγαλύτερη αντιστοίχηση υψηλής βαθμολόγησης, χρησιμοποιώντας κενά. Για να εξηγήσουμε καλύτερα το πως γίνεται αυτό, ας θεωρήσουμε έναν κατευθυνόμενο γράφο με βάρη στους κόμβους. Κάθε κόμβος αντιπροσωπεύει μία από τις δέκα αντιστοιχίσεις του δεύτερου μέρους με βαθμολόγηση πάνω από την τιμή αποκοπής και το βάρος κάθε κόμβου είναι ίσο με τη βαθμολόγηση αυτή. Έστω *u* μία από τις αντιστοιχίσεις αυτές, που αρχίζει από το ζεύγος θέσεων (*i*, *j*) και τελειώνει στο (*i+d*, *j+d*) και *v* μία δεύτερη αντιστοίχιση που αρχίζει στα σημεία (*i'*, *j'*). Τότε στον γράφο σχεδιάζουμε μια ακμή από τον κόμβο *u* στον *v* αν και μόνο αν είναι *i'>i+d*. Διαισθητικά, αυτό σημαίνει ότι η αντιστοίχιση *u* ξεκινάει από χαμηλότερη σειρά του πίνακα από αυτή που τελειώνει η *v*. Στην ακμή αυτή αντιστοιχούμε ένα βάρος το οποίο αντιπροσωπεύει το κόστος των κενών που πρέπει να εισάγουμε

μεταξύ των δύο αντιστοιχίσεων. Αυτό το κόστος θα είναι μεγάλο αν οι αντιστοιχίσεις βρίσκονται σε διαγωνίους που απέχουν πολύ ή αν βρίσκονται στην ίδια διαγώνιο και η μεταξύ τους απόσταση είναι μεγάλη. Εναλλακτικά όμως, μπορούμε να χρησιμοποιήσουμε και σταθερά βάρη για τις ακμές. Στη συνέχεια ο αλγόριθμος εντοπίζει στον γράφο αυτό τη διαδρομή μέγιστου βάρους. Αυτή η διαδρομή αντιπροσωπεύει μια τοπική αντιστοίχιση για τις δύο ακολουθίες, η οποία μπορεί να διαφέρει από την βέλτιστη που θα επέλεγε ο αντίστοιχος αλγόριθμος δυναμικού προγραμματισμού, θα έχει όμως καλή βαθμολόγηση σε σύγκριση με αυτή. Το αποτέλεσμα του τρίτου βήματος ονομάζεται *initn*.

Τέλος, στο τέταρτο βήμα ο αλγόριθμος FASTA υπολογίζει μία εναλλακτική βαθμολόγηση. Επιστρέφοντας στην αντιστοίχιση *init1* (το αποτέλεσμα από το δεύτερο βήμα) επιλέγει μία ζώνη διαγωνίων γύρω από τη διαγώνιο που περιέχει την *init1*. Για πρωτεΐνες και *ktup*=1 η ζώνη αυτή αποτελείται από 32 διαγωνίους, ενώ για *ktup*=2 από 16. Στη συνέχεια χρησιμοποιεί τον αλγόριθμο δυναμικού προγραμματισμού για να υπολογίσει την ιδανική τοπική αντιστοίχιση, μεταξύ των ακολουθιών, περιορισμένο στη ζώνη αυτή. Το αποτέλεσμα αυτού του βήματος είναι μια αντιστοίχιση, της οποίας τη βαθμολόγηση ονομάζουμε *opt*.

Με την εφαρμογή του αλγόριθμου σε διαδοχικές ακολουθίες της βάσης, ο αλγόριθμος συγκεντρώνει στατιστικά στοιχεία για τις παραμέτρους *init1*, *initn*, *opt*, τις οποίες και χρησιμοποιεί στη συνέχεια για την αξιολόγηση της στατιστικής σημασίας των αποτελεσμάτων.

3.5. Ο αλγόριθμος BLAST

Σε αυτή την ενότητα θα αναφέρουμε περιληπτικά κάποιες γενικές πληροφορίες για τον αλγόριθμο BLAST. Για την αναλυτική περιγραφή του θα αφιερώσουμε ολόκληρο το Κεφάλαιο 4, καθώς είναι ο αλγόριθμος που επιλέξαμε να υλοποιήσουμε.

Ο αλγόριθμος BLAST (Basic Local Alignment Search Tool) [1] είναι όπως και ο FASTA ένας ευριστικός αλγόριθμος που υπολογίζει τοπικές αντιστοιχίσεις χωρίς κενά μεταξύ ακολουθιών, που προσεγγίζουν τις βέλτιστες. Δημιουργήθηκε ουσιαστικά από το συνδυασμό τριών προσπαθειών. Η πρώτη ήταν αυτή των D. Lipman, W.Gish και άλλων για τη βελτίωση της ταχύτητας του αλγόριθμου FASTA, εισάγοντας πιο αυστηρούς κανόνες για των εντοπισμό (λιγότερων και καλύτερων) αντιστοιχίσεων. Η δεύτερη ήταν η εισαγωγή της ιδέας της γειτονιάς υποακολουθιών και των αυτομάτων για την εύρεση των σημείων έναρξης ταυτιζόμενων υποακολουθιών από τον E. Myers. Η τρίτη ήταν η εργασία των S. Karlin, A. Dembo και S. Altschul [7], που έδωσε τα στατιστικά αποτελέσματα που χρησιμοποιεί ο BLAST για την αξιολόγηση της στατιστικής σημασίας των αποτελεσμάτων.

Σχεδόν αμέσως μετά τη δημοσίευσή του το 1990, ο αλγόριθμος BLAST έγινε ο επικρατέστερος αλγόριθμος για αναζήτηση σε βάσεις βιολογικών δεδομένων, εκτοπίζοντας από τη θέση αυτή τον FASTA. Οι βασικοί λόγοι είναι η ταχύτητά του, η δυνατότητα να παρουσιάζει μια ευρεία σειρά αποτελεσμάτων και το γεγονός ότι κάθε αποτέλεσμα συνοδεύεται από μια

εκτίμηση της στατιστικής σημασίας. Στην αρχική του έκδοση ο αλγόριθμος BLAST αναφερόταν ότι είναι μία τάξη μεγέθους ταχύτερος από τον FASTA [1]. Παρόλο που ο αλγόριθμος FASTA έχει εξελιχθεί σημαντικά μετά την παρουσίαση του BLAST, η παράλληλη εξέλιξη του BLAST του επιτρέπει να παραμένει και σήμερα ταχύτερος και επικρατέστερος στο πρόβλημα της αναζήτησης σε βάσεις βιολογικών δεδομένων.

3.6. Ανακεφαλαίωση

Είδαμε σε αυτό το κεφάλαιο κάποιους από τους αλγόριθμους που μπορούν να χρησιμοποιηθούν για το πρόβλημα που εξετάζουμε. Οι αλγόριθμοι αυτοί ανήκουν σε δύο κατηγορίες: αλγόριθμους δυναμικού προγραμματισμού και ευριστικούς. Οι αλγόριθμοι δυναμικού προγραμματισμού εγγυώνται την εύρεση της βέλτιστης λύσης, θυσιάζοντας όμως την ταχύτητα εκτέλεσης. Αντίθετα, οι ευριστικοί αλγόριθμοι θυσιάζουν μέρος της ευαισθησίας και της ακρίβειάς τους προς όφελος της ταχύτητας. Από τους αλγόριθμους που έχουμε αναφέρει εδώ, ο αλγόριθμος BLAST πετυχαίνει την βέλτιστη ισορροπία μεταξύ ταχύτητας και ακρίβειας. Αυτός ήταν και ο λόγος που οδήγησε στην επιλογή του για την υλοποίηση.

ΚΕΦΑΛΑΙΟ 4

Ο αλγόριθμος BLAST

Το κεφάλαιο αυτό είναι αφιερωμένο στο σύνολό του στον αλγόριθμο που επιλέξαμε να υλοποιήσουμε. Ο αλγόριθμος αυτός είναι ο BLAST, ο οποίος όπως αναφέραμε στην Ενότητα 3.5 είναι ο επικρατέστερος αλγόριθμος για αναζήτηση σε βάσεις βιολογικών δεδομένων. Ξεκινάμε με μια αναλυτική περιγραφή της δομής του αλγόριθμου και των εργασιών που εκτελούνται σε κάθε αλγοριθμικό βήμα, παρουσιάζοντας και κατάλληλα παραδείγματα όπου είναι απαραίτητο. Στη συνέχεια, περιγράφουμε το μηχανισμό της αξιολόγησης των αποτελεσμάτων, που αποτελεί και ένα από τα χαρακτηριστικά του BLAST που τον κάνουν τόσο δημοφιλή.

4.1. Η δομή του αλγόριθμου BLAST

4.1.1. Γενικά

Ο αλγόριθμος BLAST επιχειρεί να προσεγγίσει τις βέλτιστες τοπικές αντιστοιχίσεις μεταξύ δύο ακολουθιών, υπολογίζοντας βαθμολογήσεις μεταξύ συνεχών υποακολουθιών [1]. Τα τμήματα αυτά των ακολουθιών σχηματίζουν ζεύγη, τα οποία ανάλογα με τη βαθμολόγηση και τις ιδιότητές τους έχουν διαφορετικές ονομασίες.

Ορισμός 4.1: *Μέγιστο Ζεύγος Τμημάτων (Maximal Segment Pair- MSP) είναι το ζεύγος των συνεχών υποακολουθιών ίσου μήκους που δίνουν τη μέγιστη βαθμολόγηση από όλα τα πιθανά ζεύγη υποακολουθιών που προέρχονται από τις ακολουθίες που εξετάζουμε* [1].

Η βαθμολόγηση αυτού του ζεύγους τμημάτων μας δίνει και μια εκτίμηση της τοπικής ομοιότητας των ακολουθιών. Η βασική λειτουργία του BLAST είναι ο εντοπισμός αυτού του ζεύγους για δύο ακολουθίες.
Επειδή εκείνο που συχνά ενδιαφέρει τους βιολόγους είναι ο εντοπισμός όλων των ζευγών υποακολουθιών που εμφανίζουν υψηλή ομοιότητα, ορίζουμε τα τοπικά μέγιστα ζεύγη τμημάτων.

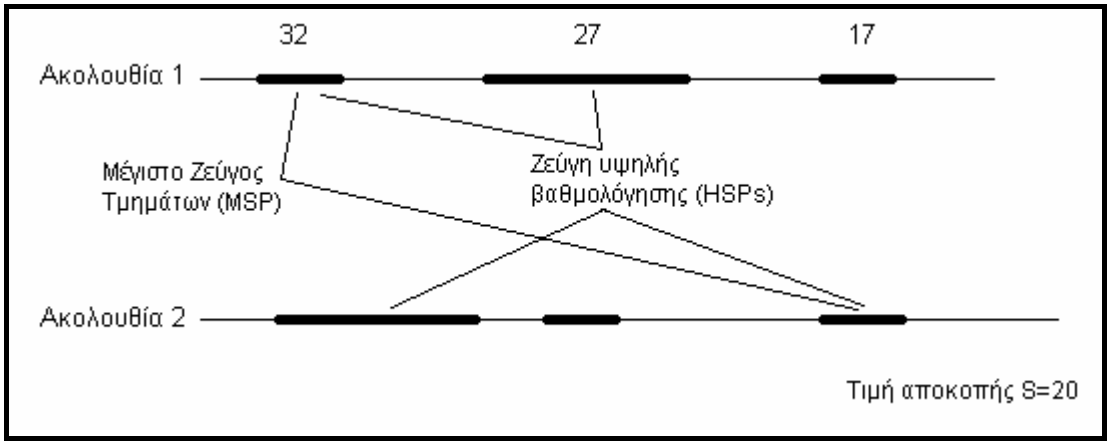
Ορισμός 4.2: *Ένα ζεύγος τμημάτων του οποίου η βαθμολόγηση δεν μπορεί να βελτιωθεί είτε επεκτείνοντας είτε περιορίζοντας τα όριά του ονομάζεται τοπικά μέγιστο ζεύγος τμημάτων (locally maximal segment pair).* [1]

Ορισμός 4.3: *Κάθε τοπικά μέγιστο ζεύγος τμημάτων του οποίου η βαθμολόγηση είναι μεγαλύτερη από μία τιμή αποκοπής ονομάζεται ζεύγος τμημάτων υψηλής βαθμολόγησης (High-scoring Segment Pair- HSP).* [1]

Η κατηγοριοποίηση αυτή των ζευγών υποακολουθιών φαίνεται στο Σχήμα 4.1. Ο αλγόριθμος BLAST εντοπίζει και αναφέρει όλα τα ζεύγη υψηλής βαθμολόγησης, για μία τιμή αποκοπής που επιλέγεται με βάση ένα στατιστικό μοντέλο που λαμβάνει υπ' όψιν τα χαρακτηριστικά των ακολουθιών και του πίνακα βαθμολόγησης που χρησιμοποιούμε. Για κάθε ζεύγος υψηλής βαθμολόγησης, ο αλγόριθμος υπολογίζει και μια εκτίμηση της στατιστικής σημασίας της αντιστοίχισης των τμημάτων αυτών. Τα ζεύγη για τα οποία η στατιστική σημασία ικανοποιεί ένα κριτήριο που επιλέγεται από το χρήστη αναφέρονται σαν αποτελέσματα του BLAST.
Τα βασικά βήματα του αλγόριθμου BLAST είναι τρία [1]:

- Κατασκευή λίστας λέξεων, χρησιμοποιώντας αποκλειστικά την ακολουθία που εξετάζουμε.
- Σάρωση της βάσης δεδομένων για την εύρεση επιτυχιών (hits), τμημάτων δηλαδή ακολουθιών που ταυτίζονται με λέξεις στη λίστα.
- Επέκταση των επιτυχιών για των εντοπισμό ζευγών υψηλής βαθμολόγησης.

Ανάλογα με τον τύπο των ακολουθιών που χρησιμοποιούμε, έχουμε μικρές διαφοροποιήσεις στην διαδικασία του πρώτου βήματος (όπως περιγράφεται στην Ενότητα 4.4.2). Τα άλλα δύο βήματα είναι όμοια, ανεξάρτητα του τύπου των ακολουθιών.



Σχήμα 4.1: Παραδείγματα Μέγιστων Ζευγών, ζευγών υψηλής βαθμολόγησης, τοπικά μέγιστων ζευγών. Όλα τα τμήματα που σημειώνονται αποτελούν τοπικά μέγιστα ζεύγη και οι αριθμοί φανερώνουν τις βαθμολογήσεις τους.

4.1.2. Τα βήματα του αλγόριθμου

Όπως είδαμε στην Ενότητα 4.1.1 ο αλγόριθμος BLAST αποτελείται από τρία βήματα. Εδώ θα δούμε αναλυτικά τις ενέργειες που εκτελούνται σε κάθε βήμα και για τους δύο τύπους ακολουθιών.

4.1.2.1. Κατασκευή λίστας λέξεων

Στο πρώτο βήμα του αλγόριθμου κατασκευάζουμε μία λίστα με όλες τις ακολουθίες που έχουν ένα σταθερό μήκος w, οι οποίες δίνουν υψηλή βαθμολόγηση όταν αντιστοιχιστούν με κάποια συνεχή υποακολουθία της ακολουθίας που εξετάζουμε. Οι ακολουθίες αυτές ονομάζονται και λέξεις (words), από όπου προκύπτει και το όνομα λίστα λέξεων (word list). Η δημιουργία αυτής της λίστας διαφέρει ανάλογα με τον τύπο της ακολουθίας που εξετάζουμε.

Ακολουθίες DNA

Στην περίπτωση των ακολουθιών DNA, στη λίστα αυτή τοποθετούμε όλες τις ακολουθίες που έχουν μήκος ίσο με w, οι οποίες ταυτίζονται με κάποια υποακολουθία της ακολουθίας που εξετάζουμε. Για μία ακολουθία μήκους L έχουμε L-w+1 τέτοιες υποακολουθίες. Επειδή υπάρχει η πιθανότητα κάποιες από αυτές τις υποακολουθίες να είναι όμοιες, ο παραπάνω αριθμός αποτελεί το άνω όριο του μεγέθους της λίστας λέξεων. Στο Σχήμα 4.2 παρουσιάζουμε τον ψευδοκώδικα για την κατασκευή της λίστας λέξεων για ακολουθίες DNA.

Παράδειγμα 4.1

Έστω η ακολουθία Q=*AGTCGATCGA, με* L=*10. Χρησιμοποιώντας* w=*6 έχουμε τις παρακάτω* L-w+*1= 5 λέξεις οι οποίες τοποθετούνται στη λίστα λέξεων:*

AGTCGA
GTCGAT
TCGATC
CGATCG
GATCGA

```
Algorithm DNAWordList
    input: sequence Q, word length w
    output: the word list for Q with word length w

    for index ← 0 to Q-w+1 do
        word = subsequence Q[index, index+1]
        add word to word list
    return word list
```

Σχήμα 4.2: Αλγόριθμος για την κατασκευή λίστας λέξεων για ακολουθίες DNA.

Ακολουθίες πρωτεϊνών

Σε αυτή την περίπτωση, η κατασκευή είναι πιο πολύπλοκη. Στην περίπτωση αυτή, χρησιμοποιούμε μια βαθμολόγηση κατωφλίου *T* και ένα πίνακα βαθμολόγησης. Στη συνέχεια, τοποθετούμε στη λίστα λέξεων όλες τις ακολουθίες που, με τον δεδομένο πίνακα βαθμολόγησης, δίνουν αντιστοιχίσεις με βαθμολόγηση μεγαλύτερη της τιμής κατωφλίου για κάποια υποακολουθία της ακολουθίας που εξετάζουμε. Το σύνολο αυτών των ακολουθιών ονομάζονται γειτονιά λέξεων (word neighborhood) [1]. Κατά αυτό τον τρόπο, ακόμα και λέξεις που ταυτίζονται με υποακολουθίες μπορεί να μην συμπεριληφθούν στη λίστα λέξεων, καθώς η διαλογή γίνεται με βάση το κριτήριο της βαθμολόγησης. Συμπεριλαμβάνοντας αυτές τις ακολουθίες, δίνουμε μια αυξημένη ευαισθησία στον αλγόριθμο.
Για μία ακολουθία *Q* μήκους *L*, βαθμολόγηση κατωφλίου *T* και μήκος λέξεων *w*, έχουμε *L*-*w*+1 συνεχείς υποακολουθίες. Για κάθε μία από αυτές υπολογίζουμε μια λίστα ακολουθιών που δίνουν βαθμολόγηση μεγαλύτερη του *T* και συνενώνουμε αυτές τις λίστες. Μια λύση είναι η κατασκευή όλων των δυνατών ακολουθιών με μήκος w για το αλφάβητο των ακολουθιών πρωτεϊνών, η σύγκριση με την τρέχουσα υποακολουθία και η επιλογή αυτών που δίνουν βαθμολογήσεις μεγαλύτερες του *T*. Επειδή αυτή η προσέγγιση είναι αργή (αν *N* είναι το μήκος του αλφαβήτου, υπάρχουν N^w δυνατές ακολουθίες), στην πράξη χρησιμοποιείται μια διαδικασία που σταματά τους υπολογισμούς βαθμολογήσεων όταν αυτές γίνουν μικρότερες από την τιμή *T*. Αυτό πετυχαίνεται με κατευθυνόμενες αντικαταστάσεις συμβόλων. Για κάθε θέση της υποακολουθίας αντικαθιστούμε το σύμβολο, ξεκινώντας από αυτό που δίνει τη μεγαλύτερη βαθμολόγηση και ακολουθώντας φθίνουσα σειρά. Όταν η βαθμολόγηση για την υποακολουθία γίνει μικρότερη της τιμής *T* σταματάμε τις αντικαταστάσεις, καθώς όλα τα υπόλοιπα σύμβολα θα δίνουν χαμηλότερες βαθμολογήσεις. Με αυτό τον τρόπο κατασκευάζουμε μία λίστα με τα διαθέσιμα σύμβολα για κάθε θέση. Στο Σχήμα 4.3 παρουσιάζουμε τον αλγόριθμο για την κατασκευή αυτής της λίστας.
Αφού βρούμε τα διαθέσιμα σύμβολα για κάθε θέση με τον παραπάνω τρόπο, χρησιμοποιούμε μια αναδρομική συνάρτηση για την κατασκευή της λίστας λέξεων. Με τη συνάρτηση αυτή, δημιουργούμε τις αντικαταστάσεις για όλους τους συνδυασμούς θέσεων. Επειδή η λίστα συμβόλων είναι ταξινομημένη (από το σύμβολο με τη μεγαλύτερη βαθμολόγηση προς το σύμβολο με τη μικρότερη) μπορούμε πάλι να σταματήσουμε τις αντικαταστάσεις όταν η βαθμολόγηση γίνει μικρότερη από την τιμή *T*. Στο Σχήμα 4.4 φαίνεται ο αλγόριθμος που χρησιμοποιεί την λίστα συμβόλων για την κατασκευή της λίστας λέξεων. Ο αλγόριθμος καλείται για πρώτη φορά με παραμέτρους μια κενή συμβολοσειρά, τη λίστα λέξεων που υπολογίζεται από τον αλγόριθμο του Σχήματος 4.3, τη διαφορά της βαθμολόγησης κατωφλίου από τη μέγιστη βαθμολόγηση και τον πίνακα βαθμολόγησης που χρησιμοποιούμε.

Για κάθε λέξη αποθηκεύουμε το σημείο έναρξής της στην ακολουθία που εξετάζουμε, για να χρησιμοποιηθεί στα επόμενα βήματα του αλγορίθμου.

```
Algorithm SymbolList
    input: a w-length subsequence q of Q, scoring matrix M, threshold T
    output: the symbol list for q

    MaxScore ← 0
    for i ← 0 to w do
        x ← max score from row q[i] of M
        MaxScore ← MaxScore + x
    for i ← 0 to w do
        j ← 1
        S ← symbol with j-th greatest score from row q[i] of M
        while M[ q[i], S] ≥ T  - MaxScore do
            add symbol S to row i of symbol list
            j ← j + 1
            S ← symbol with j-th greatest score from row q[i] of M
    return symbol list
```

Σχήμα 4.3: Αλγόριθμος για την κατασκευή της λίστας συμβόλων.

```
Algorithm ProteinWordList
    input: a sequence temp, a symbol list for a w-length subsequence q of Q, a
    distance D, scoring matrix M
    output: the word list for q

    position ← length of temp
    if number of symbols in row position of symbol list = 0
        return null
    for i ← 0 to number of symbols in row position of symbol list
        if M[q[position], symbol list[position, i] ≥
                        max score of row q[ position] of M - D
            temp ← concatenate temp, symbol list[position, i]
            D ← max score of row q[ position] of M
                        - M[q[position], symbol list[position, i]
            if |temp| = w
                add temp to word list
            else
                ProteinWordList( temp, symbol list, D, M)
```

Σχήμα 4.4: Αναδρομικός αλγόριθμος για την κατασκευή της λίστας λέξεων από τη λίστα συμβόλων.

Παράδειγμα 4.2

Έστω ότι έχουμε μία ακολουθία πρωτεΐνης, για την οποία χρησιμοποιούμε w*=4. Ας υποθέσουμε ότι στην ακολουθία αυτή υπάρχει η υποακολουθία* q*=ACDE. Για αυτή την υποακολουθία, θα υπολογίσουμε τη λίστα λέξεων, χρησιμοποιώντας* T*=20 και πίνακα βαθμολόγησης τον PAM 120 (Σχήμα 2.1).*
Αρχικά υπολογίζουμε τον πίνακα διαθέσιμων συμβόλων. Η μέγιστη βαθμολόγηση για την ακολουθία αυτή είναι

$$MaxScore = 22$$

Εφαρμόζοντας τον αλγόριθμο του Σχήματος 4.3, έχουμε τα παρακάτω αποτελέσματα:

Θέση	**Διαθέσιμα σύμβολα**
1	A, G, P, S, T
2	C
3	D, E
4	E, D

Στη λίστα συμβόλων τοποθετούνται και τα σύμβολα της αρχικής ακολουθίας, για να ελεγχθεί η περίπτωση να μην ικανοποιείται το κριτήριο της τιμής κατωφλίου από την αρχική ακολουθία.
Εφαρμόζοντας τον αλγόριθμο του Σχήματος 4.4, έχουμε τα παρακάτω αποτελέσματα:

Λέξη	**D**	**Λέξη**	**D**	**Λέξη**	**D**	**Λέξη**	**D**
A	2	A**C**	2	AC**D**	2	ACD**E**	2
						ACD**D**	0
				AC**E**	0	ACE**E**	0
						ACE**D**	-2 stop
G	0	G**C**	0	GC**D**	0	GCD**E**	0
						GCD**D**	-2 stop
				GC**E**	-2 stop	-	-
						-	-
P	0	P**C**	0	PC**D**	0	PCD**E**	0
						PCD**D**	-2 stop
				PC**E**	-2 stop	-	-
						-	-
S	0	S**C**	0	SC**D**	0	SCD**E**	0
						SCD**D**	-2 stop
				SC**E**	-2 stop	-	-
						-	-
T	0	T**C**	0	TC**D**	0	TCD**E**	**D**
						TCD**D**	-2 stop
				TC**E**	-2 stop	-	0
						-	0

Στον παραπάνω πίνακα το σύμβολο που προστίθεται σε κάθε βήμα σημειώνεται με έντονο χαρακτήρα. Τα υπόλοιπα σύμβολα προέρχονται από το προηγούμενο βήμα μέσω της μεταβλητής temp. *Για ευκολία, σημειώνουμε και τις ακολουθίες που δίνουν αρνητική τιμή για την παράμετρο* D, *αν και στην πραγματικότητα η ακολουθίες αυτές δε δημιουργούνται ποτέ, καθώς γνωρίζουμε ότι η αντικατάσταση με το τρέχον σύμβολο οδηγεί σε μη αποδεκτή βαθμολόγηση. Δίπλα στις (αρνητικές) τιμές της παραμέτρου* D *σημειώνουμε τη λέξη stop, καθώς εκεί σταματούν οι αντικαταστάσεις. Τα στοιχεία του πίνακα που εμφανίζεται το σύμβολο – δεν υπολογίζονται, αφού πρόκειται για ακολουθίες που σίγουρα έχουν τιμή χαμηλότερη του*

4.1.2.2. Σάρωση της βάσης δεδομένων

Το δεύτερο βήμα, που είναι και το πιο απλό, δεν παρουσιάζει διαφοροποίηση για τους δύο τύπους ακολουθιών. Αφού κατασκευάσουμε τη λίστα λέξεων για την ακολουθία που εξετάζουμε, παίρνουμε με τη σειρά τις ακολουθίες της βάσης δεδομένων και τις εξετάζουμε για επιτυχίες. Σαν επιτυχία ορίζουμε ένα τμήμα μιας ακολουθίας που ταυτίζεται με μια από τις λέξεις της λίστας. Η επιτυχία εντοπίζεται αναζητώντας στη λίστα λέξεων μια λέξη που μόλις διαβάσαμε από την βάση δεδομένων. Η αναζήτηση αυτή μπορεί να γίνει με δύο βασικούς τρόπους: με τη χρήση συνάρτησης κατακερματισμού και με τη χρήση ενός πεπερασμένου ντετερμινιστικού αυτόματου.

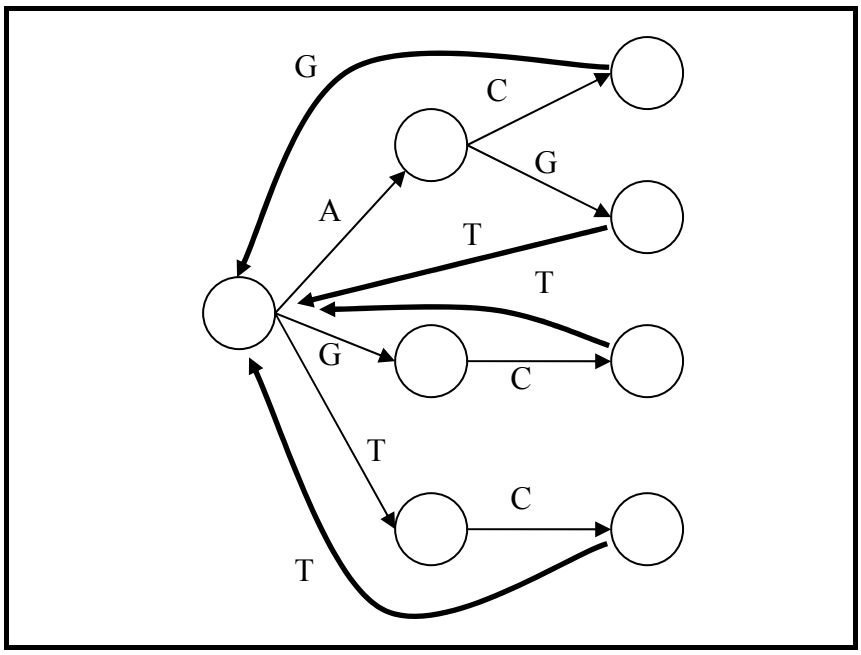
Με τη συνάρτηση κατακερματισμού, χρησιμοποιούμε μια συνάρτηση που σε κάθε δυνατή ακολουθία μήκους w για το αλφάβητο της ακολουθίας που εξετάζουμε, αντιστοιχεί μια τιμή που λειτουργεί ως δείκτης σε έναν πίνακα. Στον πίνακα αυτό τοποθετούνται οι λέξεις τις λίστας, χρησιμοποιώντας τη συνάρτηση κατακερματισμού. Κατόπιν, χρησιμοποιώντας την ίδια συνάρτηση, ελέγχουμε αν η λέξη που διαβάσαμε από τη βάση δεδομένων αντιστοιχεί σε μια εγγραφή του πίνακα που περιέχει μια λέξη της λίστας.

Η δεύτερη επιλογή είναι το ντετερμινιστικό αυτόματο, για το οποίο προτιμούμε το μοντέλο των μεταβάσεων αντί των καταστάσεων αποδοχής για μεγαλύτερη ταχύτητα. Το αυτόματο κατασκευάζεται έτσι ώστε για τον τελευταίο χαρακτήρα κάθε λέξης της λίστας να χρησιμοποιείται μία μετάβαση αποδοχής. Η σάρωση της βάσης δεδομένων γίνεται ακολουθώντας την κατάλληλη μετάβαση από την τρέχουσα κατάσταση προς μια επόμενη, για κάθε χαρακτήρα της ακολουθίας που διαβάζουμε από τη βάση. Όταν η μετάβαση που χρησιμοποιείται είναι μετάβαση αποδοχής τότε έχουμε επιτυχία. Από πειραματικές μετρήσεις, βρέθηκε ότι η μέθοδος με το ντετερμινιστικό αυτόματο είναι ταχύτερη αλλά και έχει μικρότερες απαιτήσεις σε μνήμη για αποθήκευση των δομών του, γι' αυτό και είναι αυτή που χρησιμοποιείται στην πράξη [1].

Για κάθε επιτυχία αποθηκεύουμε τα σημεία έναρξής της στις δύο ακολουθίες, έτσι ώστε να προχωρήσουμε στην επέκτασή της στο τρίτο βήμα του αλγόριθμου.

Παράδειγμα 4.3

Έστω ότι για μία ακολουθία DNA δημιουργούμε τη λίστα λέξεων για w=*3 και σε αυτή τη λίστα υπάρχουν οι λέξεις AGT, ACG, GCT, TCT. Τότε ένα αυτόματο για την αποδοχή τους θα έχει τη μορφή του Σχήματος 4.5. Στο σχήμα αυτό παραλείπονται οι υπόλοιπες μεταβάσεις για λόγους ευκολίας της απεικόνισης. Όσες μεταβάσεις παραλείπονται αντιστοιχούν σε απλές μεταβάσεις από μια κατάσταση προς την αρχική. Τα έντονα βέλη αντιστοιχούν στις μεταβάσεις αποδοχής.*



Σχήμα 4.5: Αυτόματο για το Παράδειγμα 4.3.

4.1.2.3. Επέκταση των επιτυχιών

Κάθε επιτυχία που εντοπίζεται στο δεύτερο βήμα επεκτείνεται σε ένα τοπικά μέγιστο ζεύγος τμημάτων. Η επέκταση γίνεται και προς τις δύο κατευθύνσεις στις ακολουθίες και σταματάει για κάποια κατεύθυνση όταν φτάσουμε στο άκρο της μιας ακολουθίας προς την κατεύθυνση αυτή, ή όταν η βαθμολόγηση του ζεύγους γίνει μικρότερη κατά μία διαφορά *X* από τη μέγιστη τιμή την οποία υπολογίσαμε για κάποιο προηγούμενο βήμα της επέκτασης. Στην δεύτερη περίπτωση, μπορούμε να κρατήσουμε μόνο το τμήμα εκείνο της εκτεταμένης επιτυχίας στο οποίο η βαθμολόγηση αυξάνεται, απορρίπτοντας ουσιαστικά τα τμήματα που προκαλούν την πτώση της βαθμολόγησης.

Στη συνέχεια συγκρίνουμε την βαθμολόγηση της εκτεταμένης επιτυχίας με μία προκαθορισμένη τιμή μιας τιμής αποκοπής S_2. Αν είναι μεγαλύτερη τότε η εκτεταμένη επιτυχία αποτελεί και ζεύγος τμημάτων υψηλής βαθμολόγησης. Με βάση την βαθμολόγηση του ζεύγους τμημάτων υπολογίζεται η στατιστική σημασία του και αν αυτή ικανοποιεί μια τιμή *E* που επιλέγεται από το χρήστη, τότε το ζεύγος παρουσιάζεται στα αποτελέσματα του αλγορίθμου.
Σε μια προσπάθεια να επιταχυνθεί η διαδικασία επέκτασης των επιτυχιών, εισάγουμε έναν μηχανισμό που δεν ξεκινά την επέκταση επιτυχιών, οι οποίες έχουν καλυφθεί από προηγούμενες επεκτάσεις. Για το λόγο αυτό εισάγουμε την έννοια της διαγωνίου για της επιτυχίες.

Ορισμός 4.4: *Η διαγώνιος για μια επιτυχία που έχει σημεία έναρξης τα σύμβολα στις θέσεις* i, j *δύο ακολουθιών* s_1, s_2 *είναι ίση με* i-j.

Μπορούμε βέβαια και να ορίσουμε αντίθετα την διαγώνιο, δηλαδή ως *j-i*, ερκεί να ακολουθούμε την ίδια σύμβαση για όλες τις επιτυχίες. Σύμφωνα με αυτό τον ορισμό, υπάρχουν και αρνητικές διαγώνιοι. Όλα τα ζεύγη υποακολουθιών για τα οποία η διαφορά *i-j* είναι ίδια, ανήκουν στην ίδια διαγώνιο.

Παράδειγμα 4.4

Έστω οι λέξεις ATC, GTA, οι οποίες δίνουν τις παρακάτω επιτυχίες σε δύο ακολουθίες:

*120: ...AA**GTA**AGGACC**ATC**GA...*

*45: ...GA**ATC**CCA**GTA**CG...*

Για τις επιτυχίες αυτές έχουμε:

Λέξη	**Σημείο έναρξης στην ακολουθία 1**	**Σημείο έναρξης στην ακολουθία 2**	**Διαγώνιος**
ATC	122	47	75
GTA	53	131	-78

Για κάθε διαγώνιο αποθηκεύουμε το τελευταίο σημείο στο οποίο τελειώνει μια προηγούμενη εκτεταμένη επιτυχία για αυτή τη διαγώνιο. Έτσι όταν προκύψει μια νέα επιτυχία για αυτή τη διαγώνιο, συγκρίνουμε το σημείο έναρξής της με το σημείο το οποίο έχουμε αποθηκευμένο για τη διαγώνιο. Αν είναι μικρότερο τότε η επιτυχία αυτή έχει καλυφθεί από μια προηγούμενη επέκταση και δεν ξεκινά η διαδικασία επέκτασής της. Αν είναι μεγαλύτερο, γίνεται η επέκταση της επιτυχίας και το σημείο που τελειώνει αποθηκεύεται για την διαγώνιο στην οποία ανήκει η επιτυχία.

Για κάθε ζεύγος υψηλής βαθμολόγησης που εντοπίζεται στο τρίτο βήμα, αποθηκεύουμε κάποιες βασικές πληροφορίες, όπως τα σημεία έναρξης στις δύο ακολουθίες, το μήκος και η βαθμολόγηση.

4.2. Υπολογισμός της στατιστικής σημασίας των αποτελεσμάτων

Για την αξιολόγηση της στατιστικής σημασίας χρησιμοποιούμε τα συμπεράσματα που προέκυψαν από τη θεωρητική ανάλυση της κατανομής των βαθμολογήσεων μέγιστων ζευγών τμημάτων από τυχαίες ακολουθίες [7]. Συνοπτικά, το μοντέλο που χρησιμοποιούμε βασίζεται σε δύο παραμέτρους K και λ, που υπολογίζονται με βάση τις συχνότητες εμφάνισης των συμβόλων τόσο για την ακολουθία που εξετάζουμε όσο και τις συχνότητες που προκύπτουν από παρατηρήσεις σε πραγματικές ακολουθίες του κάθε είδους. Η παράμετρος K αντιπροσωπεύει την επιρροή του μεγέθους του χώρου αναζήτησης (στην περίπτωσή μας, της βάσης δεδομένων) στα αποτελέσματα. Η παράμετρος λ αντιπροσωπεύει την επιρροή του συστήματος βαθμολόγησης που χρησιμοποιούμε. Η πιθανότητα να βρεθεί ένα ζεύγος τμημάτων με βαθμολόγηση ίση ή μεγαλύτερη από μια τιμή S, κατά τη σύγκριση δύο ακολουθιών με μήκη n και m είναι [1]:

$$P = 1 - e^{-E}$$

όπου

$$E = Kmne^{-\lambda S} \quad (4.1)$$

Στη σχέση αυτή, το E αντιπροσωπεύει τον αριθμό των αναμενόμενων μέγιστων ζευγών τμημάτων με βαθμολόγηση ίση με S. Στην πράξη αυτή είναι και η τιμή που χρησιμοποιούμε για την αξιολόγηση της στατιστικής σημασίας ενός ζεύγους τμημάτων, καθώς είναι πιο εύχρηστη αλλά και πιο κατανοητή (είναι προτιμότερο να λέμε ότι «αναμένουμε να έχουμε 10 ζεύγη με βαθμολόγηση 100» παρά «η πιθανότητα να έχουμε ένα ζεύγος με βαθμολόγηση 100 είναι 0.148×10^{-5}...»).
Η πιθανότητα να βρεθούν x ή περισσότερα διακριτά ζεύγη τμημάτων με βαθμολόγηση S το καθένα είναι

$$P = 1 - e^{-E} \sum_{i=0}^{x} \frac{E^i}{i!} \quad (4.2)$$

Με βάση την σχέση (4.2), δύο ακολουθίες που έχουν αρκετές περιοχές με ομοιότητες μπορεί να θεωρηθούν σαν συνολικά συγγενείς, ακόμα κι αν κανένα από τα ζεύγη τμημάτων δεν είναι στατιστικά σημαντικό όταν εξετάζεται απομονωμένο.
Χρησιμοποιώντας τη σχέση (4.1), μπορούμε να υπολογίσουμε την ελάχιστη τιμή της βαθμολόγησης που πρέπει να έχει ένα μέγιστο ζεύγος τμημάτων για να θεωρηθεί στατιστικά σημαντικό, όταν δίνεται ο αριθμός των αναμενόμενων ζευγών τμημάτων E. Αντίστροφα, όταν δίνεται η ελάχιστη βαθμολόγηση S που πρέπει να έχει ένα ζεύγος τμημάτων, μπορούμε να

υπολογίσουμε τον αναμενόμενο αριθμό ζευγών τμημάτων που έχουν αυτή τη βαθμολόγηση. Για κάθε εκτεταμένη επιτυχία, έχουμε αποθηκευμένη τη βαθμολόγησή της καθώς και τις ακολουθίες από τις οποίες προκύπτει. Επομένως μπορούμε να υπολογίσουμε τον αναμενόμενο αριθμό ζευγών E που έχουν αυτή τη βαθμολόγηση κατά τη σύγκριση των δύο ακολουθιών. Εκείνο που μας ενδιαφέρει όμως είναι ο αριθμός ζευγών E_{DB} που αναμένεται να προκύψουν κατά την αναζήτηση στη βάση. Για την αναγωγή της τιμής E στην E_{DB}, χρησιμοποιούμε τον τύπο [16]:

$$E_{DB} = (1 - e^{-E}) \frac{D}{d} \quad (4.3)$$

όπου D το μέγεθος της βάσης δεδομένων μετρούμενο σε σύμβολα και d το μήκος της ακολουθίας που διαβάζουμε από τη βάση. Αυτή είναι και η τιμή που συγκρίνεται με το κριτήριο της στατιστικής σημασίας E για τα ζεύγη υψηλής βαθμολόγησης.

4.3. Παράμετροι του αλγόριθμου BLAST

Σε αυτή την ενότητα θα παρουσιάσουμε τις παραμέτρους του αλγόριθμου BLAST, τον τρόπο που γίνεται η επιλογή τους, καθώς και τον τρόπο που αυτές επηρεάζουν την απόδοση και τα αποτελέσματα του αλγόριθμου. Για ορισμένες από αυτές θα παρουσιαστούν και κάποιες συνήθεις ή προτεινόμενες τιμές.

- ***w*:** Μήκος λέξεων (word length). Αποτελεί το μήκος των συνεχών υποακολουθιών που αποθηκεύονται στη λίστα λέξεων. Μεγάλο μήκος λέξεων ισοδυναμεί με λιγότερες επιτυχίες, επομένως ταχύτερη εκτέλεση του τρίτου βήματος αλλά και μειωμένη ευαισθησία. Ταυτόχρονα όμως, αυξάνεται το μέγεθος της λίστας λέξεων για ακολουθίες πρωτεϊνών, επομένως επιβαρύνεται το πρώτο βήμα του αλγόριθμου. Συνηθισμένες τιμές για την παράμετρο w για πρωτεΐνες είναι από 2 ως 5, με προτεινόμενες τις 3 ή 4, ενώ για DNA η τιμή 11. Η επιλογή γίνεται από τον χρήστη.
- ***T*:** Βαθμολόγηση κατωφλίου για τη γειτονιά λέξεων (neighborhood threshold). Χρησιμοποιείται μόνο στις ακολουθίες πρωτεϊνών και αποτελεί την ελάχιστη τιμή της βαθμολόγησης που πρέπει να πετυχαίνει μια λέξη για να αποθηκευτεί στη λίστα λέξεων. Μια υψηλή τιμή οδηγεί σε μικρό αριθμό λέξεων στη λίστα, επομένως ταχύτερη κατασκευή. Προκαλεί όμως και μειωμένη ευαισθησία αφού υπάρχουν λιγότερες λέξεις που δημιουργούν επιτυχίες. Η τιμές για την παράμετρο αυτή εξαρτώνται σημαντικά από την επιλογή της w. Στον Πίνακα 4.1 παρουσιάζεται μια σύγκριση της ευαισθησίας του αλγόριθμου για διάφορες τιμές των w και T [1]. Από αυτές προκύπτει ότι τα ζεύγη τιμών w=3, T=14, w=4, T=16 και w=5, T=18 μας δίνουν ικανοποιητικές και συγκρίσιμες ευαισθησίες. Η επιλογή γίνεται από τον χρήστη.

- ***E*:** Κριτήριο στατιστικής σημασίας. Αντιπροσωπεύει τον επιθυμητό αναμενόμενο αριθμό ζευγών τμημάτων. Ένα ζεύγος τμημάτων υψηλής βαθμολόγησης, για να παρουσιαστεί ως αποτέλεσμα, θα πρέπει να το ικανοποιεί, να έχει δηλαδή βαθμολόγηση που από τη σχέση (4.1) δίνει μικρότερο *E*. Η τιμή του *E* επηρεάζει τον αριθμό των ζευγών τμημάτων που ο αλγόριθμος αναφέρει ως αποτελέσματα: μια χαμηλή τιμή θα προκαλέσει την απόρριψη περισσότερων ζευγών τμημάτων, άρα ο αλγόριθμος θα επιστρέψει λιγότερα αλλά στατιστικά σημαντικότερα αποτελέσματα. Επιλέγεται από τον χρήστη.
- ***S*:** Τιμή κατωφλίου για τη βαθμολόγηση ζευγών τμημάτων. Είναι η βαθμολόγηση που πρέπει να έχει ένα ζεύγος τμημάτων για να θεωρηθεί σημαντικό, με βάση την παράμετρο *E*. Υπολογίζεται για δεδομένη τιμή της *E* από τη σχέση (4.1).
- ***E_2*:** Κριτήριο στατιστικής σημασίας για τα ζεύγη υψηλής βαθμολόγησης. Είναι ο αριθμός ζευγών υψηλής βαθμολόγησης που αναμένεται να βρεθούν κατά τη σύγκριση δύο ακολουθιών ίσου και σταθερού μήκους: 1000 σύμβολα για ακολουθίες DNA και 300 σύμβολα για πρωτεΐνες. Έχει σταθερή τιμή, συνήθως 0.15 [16].
- ***S_2*:** Τιμή αποκοπής. Η ελάχιστη τιμή που πρέπει να έχει ένα τοπικά μέγιστο ζεύγος τμημάτων για να θεωρηθεί ζεύγος υψηλής βαθμολόγησης. Υπολογίζεται από την E_2, χρησιμοποιώντας τη σχέση (4.1) και τα μήκη που αναφέραμε προηγουμένως.
- ***X*:** Απόσταση από τη μέγιστη βαθμολόγηση στην οποία σταματά η επέκταση. Μειώνοντας την τιμή της παραμέτρου *X*, έχουμε μείωση της πιθανότητας να μην εντοπιστεί ένα ζεύγος τμημάτων υψηλής βαθμολόγησης, με ταυτόχρονη όμως αύξηση του χρόνου εκτέλεσης. Υπολογίζεται με βάση την παράμετρο λ για τον πίνακα βαθμολόγησης:

$$X = \left\lceil 10 \cdot \frac{\ln 2}{\lambda} \right\rceil$$

Πίνακας 4.1: Πιθανότητες επιτυχίας και ποσοστό των μέγιστων ζευγών τμημάτων που δεν εντοπίζονται από τον BLAST, για διάφορες τιμές των παραμέτρων *w* και *T*.

			Implied % of MSPs missed by BLAST when S equals						
w	T	Probability of a hit x10^5	45	50	55	60	65	70	75
3	11	253	1	1	0	0	0	0	0
	12	147	4	3	2	1	1	0	0
	13	83	11	8	6	4	3	2	2
	14	48	20	16	12	10	8	6	5
	15	26	33	28	23	20	17	14	12
	16	14	46	41	36	32	29	26	23
	17	7	59	55	51	47	43	40	37
	18	4	70	67	63	60	57	54	51
4	13	127	2	1	1	0	0	0	0
	14	78	5	3	2	1	1	0	0
	15	47	10	7	5	4	3	2	1
	16	28	18	14	11	8	4	5	4
	17	16	28	23	19	16	13	11	9
	18	9	40	35	30	26	22	19	17
	19	5	51	46	41	37	33	30	27
	20	3	62	57	53	49	45	41	38
5	15	64	3	2	1	1	0	0	0
	16	40	6	4	3	2	1	1	0
	17	25	12	9	6	4	3	2	2
	18	15	20	15	12	9	7	5	4
	19	9	29	23	19	15	13	10	8
	20	5	38	32	28	23	20	17	14
	21	3	48	42	37	32	29	25	22
	22	2	57	52	47	42	38	35	31
Expected number of MSPs with score at least S:			50	9	2	0.3	0.06	0.01	0.002

4.4. Εμφάνιση των αποτελεσμάτων

Είδαμε στην Ενότητα 4.1.2.3 ότι τα αποτελέσματα του BLAST αποτελούνται από όλα τα ζεύγη τμημάτων υψηλής βαθμολόγησης που ικανοποιούν το κριτήριο της στατιστικής σημασίας *E*. Για αυτά τα αποτελέσματα υπάρχουν κάποιες συγκεκριμένες επιλογές για την εμφάνιση, τις οποίες θα αναφέρουμε στην ενότητα αυτή.

4.4.1. Πίνακας επιτυχιών (Hit table)

Στην περίπτωση αυτή, εμφανίζουμε έναν πίνακα στον οποίο περιλαμβάνονται πληροφορίες για κάθε ζεύγος τμημάτων που προέκυψε από μια επιτυχία. Οι πληροφορίες αυτές περιλαμβάνουν πληροφορίες για την ακολουθία της βάσης για την οποία προέκυψε η επιτυχία, τα σημεία στα οποία αρχίζει στις δύο ακολουθίες, το μήκος και τη βαθμολόγηση του ζεύγους, τη στατιστική σημασία και άλλα. Παράδειγμα αυτής της μορφής εξόδου φαίνεται στο Σχήμα 4.6 [17].

```
# Fields: Subject id, % identity, length, q.start, s.start, e-value, bit
score
gi|2501594|sp|Q57997|Y577_METJA  100.00      162  1   1    5.1e-65   244.6
gi|2501593|sp|Q57951|Y531_METJA  37.82       156  4   25   9.0e-14   74.71
gi|1177001|sp|P42297|YXIE_BACSU  30.32       155  4   1    6.2e-11   65.08
gi|2501590|sp|P73475|YC30_SYNY3  37.14       70   89  218  3.6e-09   59.31
gi|2501596|sp|Q50777|YB54_METTM  35.14       148  4   1    1.9e-07   53.53
gi|2501591|sp|P74148|YD88_SYNY3  26.42       159  5   3    8.4e-07   51.22
gi|2507517|sp|P39177|USPG_ECOLI  28.03       157  4   1    3.2e-06   49.29
gi|3334425|sp|O27222|YB54_METTH  32.45       151  1   1    4.4e-06   48.91
gi|38372565|sp|Q83M07|USPG_SHIFL 28.03       157  4   1    4.8e-06   48.91
gi|38372601|sp|Q8XBT3|USPG_ECO57 28.03       157  4   1    5.3e-06   48.52
```

Σχήμα 4.6: Παράδειγμα πίνακα επιτυχιών για ακολουθία πρωτεΐνης (επιλογή).

4.4.2. Εμφάνιση κατά ζεύγη (Pairwise)

Η εμφάνιση κατά ζεύγη χρησιμοποιείται όταν θέλουμε να έχουμε μια απεικόνιση της αντιστοίχισης στην οποία αντιστοιχεί το ζεύγος τμημάτων. Στην περίπτωση αυτή, δημιουργούμε την αντιστοίχιση αυτών των τμημάτων ενώ αναφέρονται και ορισμένες πληροφορίες για το ζεύγος τμημάτων. Στο Σχήμα 4.7 βλέπουμε ένα παράδειγμα αυτής της απεικόνισης [17].

```
gi|2501590|sp|P73475|YC30_SYNY3   Hypothetical protein slr1230     Length = 287

 Score = 59.3 bits (142), Expect = 4e-09
 Identities = 26/70 (37%), Positives = 48/70 (68%)

Query: 89  KKELEDVGFKVKDIIVVGIPHEEIVKIAEDEGVDIIIMGSHGKTNLKEILLGSVTENVIK 148
           +K LE  GFK++  ++VG   E IV+  ED  +D+++MG+HG + ++ +++GS T  V++
Sbjct: 218 EKVLEKAGFKLEVELLVGHAEEAIVRYQEDNAIDLLLMGAHGHSRIRHLVIGSTTAQVLR 277

Query: 149 KSNKPVLVVK 158
           K++ PVL  +
Sbjct: 278 KTSIPVLTFR 287
```

Σχήμα 4.7: Παράδειγμα εμφάνισης κατά ζεύγη για ακολουθίες πρωτεϊνών.

4.4.3. Αντιστοίχιση στην ακολουθία εισόδου (Query-anchored)

Σε αυτή την περίπτωση, εμφανίζουμε όλη την ακολουθία που εξετάζουμε. Σε αυτή, αντιστοιχίζουμε τα τμήματα από τις ακολουθίες της βάσης, τα οποία συμμετέχουν στα ζεύγη υψηλής βαθμολόγησης, έτσι ώστε να έχουμε μια συνολική άποψη για τα τμήματα της ακολουθίας για τα οποία εντοπίστηκαν ομοιότητες. Παράδειγμα αυτής της απεικόνισης έχουμε στο Σχήμα 4.8 [17].

```
1      MSVMYKKILYPTDFSETAEIALKHVKAFKTLKAEEVILLHVIDEREIKKRDIFSLLLGVA      60
25        LYKKIVIPTDGSDVSLEAAKHAINIAKEFDAEVYAIYVVD---------VSPFVGLP      72
1         MFNKMLVAIDGSDMSAKALDAAVHLAKEQQAELSILHVGREAVVTTSSL-------T      50
1         MYRKILVPT-MGEYMDELIEHTLDLLHGREAEVICLYVVDT------------SVP      43
3          YGKILVALDRSELAKEVLQQAIALGQKESSQLMVFYCIDSQDLS---IYPSFYGEA      57


61     GLNKSVEEFENELKNKLTEEAKNKMENIKKELEDVGFKVKDIIVVGIPHEEIVKIAEDEG      120
61     GLNKSVEEFENELKNKLTEEAKNKMENIKKELEDVGFKVKDIIVVGIPHEEIVKIAEDEG      120
73     A--EGSWELISEL---LKEEGQEALKKVKKMAEEWGVKIHTEMLEGVPANEIVEFAEKKK      127
51     GIVYVPEHFIDEIRNEVKKEGLKILENAKEKAAEKGVQAETIYANGEPAHEILNHAKEKG      110
218                               EKVLEKAGFKLEVELLVGHAEEAIVRYQEDNA      249


121    VDIIIMGSHGKTNLKEILLGSVTENVIKKSNKPVLVVKRKNS                 162
121    VDIIIMGSHGKTNLKEILLGSVTENVIKKSNKPVLVVKRKNS                 162
128    ADLIVMGTTGKTGLERILLGSVAERVIKNAHCPVLVVKK                    166
111    VSLIVVGSRGISGLKEMMLGSVSHKVSQLSTCPVLIVR                     148
250    IDLLLMGAHGHSRIRHLVIGSTTAQVLRKTSIPVLTFR                     287
```

Σχήμα 4.8: Παράδειγμα αντιστοίχισης στην ακολουθία εισόδου. Η ακολουθία εισόδου σημειώνεται με έντονους χαρακτήρες.

4.5. Ανακεφαλαίωση

Στο κεφάλαιο αυτό παρουσιάσαμε τη δομή και τη λειτουργία του αλγόριθμου BLAST. Ξεκινώντας από μια γενική αναφορά της δομής του, προχωρήσαμε σε αναλυτική περιγραφή των τριών βημάτων του αλγόριθμου, δίνοντας περισσότερη έμφαση στις επιλογές που πραγματοποιήσαμε για την υλοποίηση που θα παρουσιάσουμε στο Κεφάλαιο 5. Έτσι περιγράψαμε αναλυτικά την κατασκευή της λίστας λέξεων, ιδιαίτερα για πρωτεΐνες, καθώς αποτελεί ίσως το σημαντικότερο μέρος του αλγόριθμου, επηρεάζοντας και την ταχύτητα και την ακρίβεια. Καθοδηγούμενοι από τις επιλογές για την υλοποίηση, δώσαμε περισσότερη έμφαση στη σάρωση της βάσης με τη χρήση ντετερμινιστικού αυτόματου. Στη συνέχεια παρουσιάσαμε τις παραμέτρους του αλγόριθμο BLAST με μια σύντομη ανάλυση του τρόπου που επηρεάζουν την ταχύτητα και την ακρίβειά του και παρουσιάσαμε το μοντέλο αξιολόγησης της στατιστικής σημασίας των αποτελεσμάτων του BLAST. Τέλος, αναφερθήκαμε σε μερικούς από τους τρόπους που μπορούν να παρουσιαστούν τα αποτελέσματα του αλγόριθμου.

ΚΕΦΑΛΑΙΟ 5

Υλοποίηση

Σε αυτό το κεφάλαιο παρουσιάζουμε αναλυτικά και αιτιολογούμε όλες τις επιλογές που κάναμε κατά την υλοποίηση του αλγόριθμου. Δίνουμε επίσης αναλυτική περιγραφή για όλες τις δομές που χρησιμοποιούμε για την αποθήκευση και επεξεργασία των δεδομένων. Περιγράφουμε επίσης τον τρόπο με τον οποίο εκτελούνται οι λειτουργίες για τα τρία μέρη του αλγόριθμου, όπως τα παρουσιάσαμε στο Κεφάλαιο 4.

5.1. Γενικά

Ξεκινώντας την περιγραφή του συστήματος που υλοποιήσαμε, θα πρέπει να αναφερθούμε στην επιλογή της γλώσσας προγραμματισμού. Η επιλογή που ακολουθήσαμε ήταν αυτή της χρήσης της Java. Ο βασικότερος λόγος για τον οποίο έγινε αυτό, είναι η δυνατότητα που προσφέρει για μεταφορά ενός προγράμματος μεταξύ διαφορετικών υπολογιστικών συστημάτων [18]. Στη συνέχεια η επιλογή αυτή ενισχύθηκε από το γεγονός ότι η χρήση μιας γλώσσας αντικειμενοστραφούς προγραμματισμού θα βοηθούσε στην υλοποίηση, καθώς πολλές από τις δομές που χρησιμοποιούμε και περιγράφουμε στη συνέχεια ταιριάζουν στις ιδιότητες του αντικειμένου. Ακόμα, για τη Java υπάρχουν διαθέσιμες κλάσεις που υλοποιούν δομές, οι οποίες ήταν ιδιαίτερα χρήσιμες στην υλοποίησή μας (για παράδειγμα οι πίνακες μεταβλητού μήκους ArrayList, συνδεδεμένες λίστες LinkedList και άλλα). Οι παραπάνω λόγοι ήταν αυτοί που μας οδήγησαν στην επιλογή της Java ως γλώσσας προγραμματισμού για την υλοποίησή μας.

Στη συνέχεια, υλοποιήσαμε ένα σύστημα το οποίο χρησιμοποιεί τον αλγόριθμο BLAST για να σαρώσει μία βάση βιολογικών δεδομένων και να βρίσκει ακολουθίες που εμφανίζουν ομοιότητα με την ακολουθία εισόδου. Ως ακολουθία εισόδου μπορεί να οριστεί είτε μια ακολουθία DNA είτε πρωτεΐνη, ενώ για τη σάρωση της βάσης δεδομένων χρησιμοποιείται ένα πεπερασμένο ντετερμινιστικό αυτόματο. Ανάλογα με τον τύπο της ακολουθίας, χρησιμοποιείται η κατάλληλη διαδικασία κατασκευής του αυτόματου, σύμφωνα με την διαδικασία κατασκευής της λίστας λέξεων του πρώτου βήματος του αλγορίθμου που περιγράψαμε στην Ενότητα 4.1.2.1. Στη συνέχεια, γίνεται η σάρωση της βάσης δεδομένων. Καθώς οι κυριότερες βάσεις βιολογικών δεδομένων είναι διαθέσιμες στο διαδίκτυο, το σύστημα που υλοποιήσαμε έχει τη δυνατότητα να εκτελεί τη διαδικασία της σάρωσης είτε για ένα τοπικό αντίγραφο της βάσης στον υπολογιστή που εκτελείται, είτε για ένα απομακρυσμένο αντίγραφο μέσω του (δια)δικτύου. Έχουν υλοποιηθεί τέλος οι κατάλληλες δομές για την εμφάνιση των αποτελεσμάτων του αλγόριθμου.

5.2. Υλοποίηση των δομών του αλγόριθμου

Σε αυτή την ενότητα παρουσιάζουμε τον τρόπο με τον οποίο υλοποιήσαμε της απαραίτητες δομές για τον αλγόριθμο BLAST. Κάθε τέτοια δομή υλοποιήθηκε ως μία ξεχωριστή κλάση, που αντιπροσωπεύει στη Java την έννοια του αντικειμένου. Η παρουσίαση θα γίνει ξεκινώντας από της μικρότερες βοηθητικές δομές και στη συνέχεια θα προχωρήσουμε προς τις κύριες, έτσι ώστε να είναι πιο κατανοητή η περιγραφή τους. Για κάθε τέτοια δομή, εκτός από μια γενική περιγραφή, θα παρουσιάσουμε τα δεδομένα τα οποία αποθηκεύει, τις μεθόδους κατασκευής (constructors) όπως επίσης και τις μεθόδους επεξεργασίας των δεδομένων που προσφέρουν.

5.2.1. Η κλάση CharIntRecord

Αυτή η κλάση αποτελεί μια βασική (και γενική) βοηθητική δομή. Μπορούμε να την δούμε απλά σαν μια δομή που αποθηκεύει μία μεταβλητή τύπου χαρακτήρα και μία τύπου ακεραίου. Η κατασκευή της είναι μία απλή ανάθεση τιμών στις δύο αυτές μεταβλητές. Οι μόνες μέθοδοι «επεξεργασίας» των δεδομένων αυτών είναι δύο μέθοδοι που επιστρέφουν τις τιμές των αντίστοιχων μεταβλητών.

5.2.2. Η κλάση ScoringMatrix

Η κλάση αυτή αποτελεί την υλοποίηση του πίνακα βαθμολόγησης που χρησιμοποιούμε (Ενότητα 2.1.5). Στη δομή αυτή αποθηκεύεται ο πίνακας βαθμολόγησης που θα χρησιμοποιήσουμε στον αλγόριθμο, με μία ειδική μορφή και πληροφορίες για αυτόν όπως η ονομασία του (για παράδειγμα PAM 120), η μέγιστη και η ελάχιστη βαθμολόγηση και η μέση (ή αναμενόμενη) βαθμολόγηση.
Πριν προχωρήσουμε στην περιγραφή της κατασκευής του πίνακα βαθμολόγησης, θα πρέπει να αναλύσουμε πως προκύπτει η ειδική μορφή στην οποία αποθηκεύονται οι βαθμολογήσεις. Είδαμε στην Ενότητα 4.1.2.1 ότι για να επιταχύνουμε την κατασκευή της λίστας λέξεων για πρωτεΐνες χρησιμοποιούμε κατευθυνόμενες αντικαταστάσεις συμβόλων. Για να το πετύχουμε αυτό πρέπει να εντοπίζουμε κάθε φορά το σύμβολο με την υψηλότερη βαθμολόγηση και στη συνέχεια να ακολουθούμε φθίνουσα πορεία. Επειδή αυτή η διαδικασία θα επιβαρύνει το χρόνο εκτέλεσης επιλέγουμε να χρησιμοποιήσουμε μια ταξινομημένη μορφή του πίνακα βαθμολόγησης, έτσι ώστε να εντοπίζουμε αμέσως τη σειρά των κατευθυνόμενων αντικαταστάσεων. Αυτή όμως η μορφή του πίνακα διαφέρει από την (απλή) μορφή των Σχημάτων 2.1 και 2.2, καθώς η σειρά των αντιστοιχίσεων αλλάζει για κάθε σύμβολο. Επομένως θα πρέπει να χρησιμοποιήσουμε μια δομή, η οποία, εκτός από τις (ταξινομημένες πλέον) βαθμολογήσεις, θα αποθηκεύει και την αντιστοίχιση για την οποία προκύπτει αυτή η βαθμολόγηση. Για το σκοπό αυτό, χρησιμοποιούμε για κάθε στοιχείο του πίνακα μία δομή τύπου *CharIntRecord*, στην οποία αποθηκεύεται μια βαθμολόγηση και το σύμβολο για το οποίο προκύπτει. Επειδή για κάθε σύμβολο η μέγιστη βαθμολόγηση προκύπτει κατά την αντιστοίχισή του με

όμοιο σύμβολο, εξετάζοντας το πρώτο (και μέγιστο ως προς τη βαθμολόγηση) στοιχείο κάθε σειράς του πίνακα μπορούμε να βρούμε σε ποιο σύμβολο αναφέρεται η συγκεκριμένη σειρά. Ακόμα, σε αυτή τη μορφή του πίνακα, τα στοιχεία που στην κανονική μορφή βρίσκονται στη διαγώνιό του και είναι τα μέγιστα, βρίσκονται στην πρώτη στήλη.

Παράδειγμα 5.1

Από τον πίνακα PAM 120 του Σχήματος 2.1 για κάποιο σύμβολο θα έχουμε:

Στοιχείο	**Τιμή**	**Στοιχείο**	**Τιμή**	**Στοιχείο**	**Τιμή**	**Στοιχείο**	**Τιμή**
1	A, 3	7	E, 0	13	Z, -1	19	H, -3
2	G, 1	8	V, 0	14	X, -1	20	L, -3
3	P, 1	9	B, 0	15	K, -2	21	F, -4
4	S, 1	10	N, -1	16	M,-2	22	Y, -4
5	T, 1	11	Q, -1	17	R, -3	23	W, -7
6	D, 0	12	I, -1	18	C, -3	24	*, -8

Το σύμβολο '' αντιπροσωπεύει το κενό, στην υλοποίησή μας όμως δεν επιτρέπονται κενά στις αντιστοιχίσεις, επομένως παραλείπεται. Να σημειώσουμε επίσης ότι χρησιμοποιείται το διευρυμένο αλφάβητο για κάθε τύπο ακολουθίας, το οποίο περιλαμβάνει και τα αμφίσημα σύμβολα.*
Ο παραπάνω πίνακας αντιπροσωπεύει μία σειρά του πίνακα βαθμολόγησης. Εξετάζοντας το στοιχείο 1 του πίνακα, διαπιστώνουμε ότι η σειρά αυτή αναφέρεται στο σύμβολο A. Πρόκειται δηλαδή για τις βαθμολογήσεις των αντιστοιχίσεων του συμβόλου A με όλα τα υπόλοιπα σύμβολα.

Η κατασκευή του πίνακα γίνεται διαβάζοντας από το κατάλληλο αρχείο, με βάση τον τύπο του, τα δεδομένα που πρέπει να αποθηκευτούν σε αυτόν. Για της ανάγκες του αλγόριθμου υλοποιήθηκαν αντιπροσωπευτικά οι πίνακες PAM 60, PAM 120, PAM 250, BLOSUM 35, BLOSUM 62, BLOSUM 90 και ο πίνακας για τις βαθμολογήσεις των συμβόλων για ακολουθίες DNA. Κάθε πίνακας αποθηκεύεται σε ένα αρχείο με τη μορφή που φαίνεται στο Σχήμα 5.1, στο οποίο φαίνεται ο πίνακας PAM 120. Σε αυτή τη μορφή, το *j* ζεύγος που αποτελείται από ένα σύμβολο και έναν αριθμό που ακολουθεί και βρίσκεται στην *i* σειρά του αρχείου, ισοδυναμεί με το στοιχείο [*i*, *j*] του πίνακα βαθμολόγησης.

```
A 3 G 1 P 1 S 1 T 1 D 0 E 0 V 0 B 0 N -1 Q -1 I -1 X -1 Z -1 K -2 M -2 R -3 C -3 H -3 L -3 F -4 Y -4 W -7
C 9 S 0 Y -1 A -3 I -3 T -3 V -3 R -4 G -4 H -4 P -4 X -4 N -5 M -6 F -6 B -6 D -7 Q -7 E -7 L -7 K -7 Z -7 W -8
D 5 B 4 E 3 Z 3 N 2 Q 1 A 0 G 0 H 0 S 0 K -1 T -1 X -2 R -3 I -3 P -3 V -3 M -4 L -5 Y -5 C -7 F -7 W -8
E 5 Z 4 D 3 B 3 Q 2 N 1 A 0 G -1 H -1 K -1 S -1 X -1 P -2 T -2 R -3 I -3 M -3 V -3 L -4 Y -5 C -7 F -7 W -8
F 8 Y 4 I 0 L 0 M -1 W -1 H -3 S -3 V -3 X -3 A -4 N -4 T -4 R -5 G -5 P -5 B -5 C -6 Q -6 Z -6 D -7 E -7 K -7
G 5 A 1 S 1 N 0 D 0 B 0 E -1 T -1 P -2 V -2 X -2 Z -2 Q -3 K -3 R -4 C -4 H -4 I -4 M -4 L -5 F -5 Y -6 W -8
H 7 Q 3 N 2 R 1 B 1 Z 1 D 0 E -1 P -1 Y -1 K -2 S -2 X -2 A -3 L -3 F -3 T -3 W -3 V -3 C -4 G -4 I -4 M -4
I 6 V 3 L 1 M 1 F 0 T 0 A -1 X -1 R -2 N -2 S -2 Y -2 D -3 C -3 Q -3 E -3 K -3 P -3 B -3 Z -3 G -4 H -4 W -6
K 5 R 2 N 1 Q 0 M 0 B 0 D -1 E -1 S -1 T -1 A -2 H -2 P -2 X -2 Z -2 G -3 I -3 L -4 V -4 W -5 Y -5 C -7 F -7
L 5 M 3 I 1 V 1 F 0 Q -2 Y -2 X -2 A -3 H -3 P -3 T -3 W -3 Z -3 R -4 N -4 E -4 K -4 S -4 B -4 D -5 G -5 C -7
M 8 L 3 I 1 V 1 K 0 R -1 Q -1 F -1 T -1 A -2 S -2 X -2 Z -2 N -3 E -3 P -3 D -4 G -4 H -4 Y -4 B -4 C -6 W -6
N 4 B 3 D 2 H 2 E 1 K 1 S 1 Q 0 G 0 T 0 Z 0 A -1 R -1 X -1 I -2 P -2 Y -2 M -3 V -3 L -4 F -4 W -4 C -5
P 6 A 1 S 1 Q 0 R -1 H -1 T -1 Z -1 N -2 E -2 G -2 K -2 V -2 X -2 B -2 D -3 I -3 L -3 M -3 C -4 F -5 Y -6 W -7
Q 6 Z 4 H 3 E 2 R 1 D 1 N 0 K 0 P 0 B 0 A -1 M -1 X -1 L -2 S -2 T -2 G -3 I -3 V -3 Y -5 F -6 W -6 C -7
R 6 K 2 Q 1 H 1 W 1 N -1 M -1 P -1 S -1 Z -1 I -2 T -2 X -2 B -2 A -3 D -3 E -3 V -3 C -4 G -4 L -4 F -5 Y -5
S 3 T 2 A 1 N 1 G 1 P 1 D 0 C 0 B 0 R -1 E -1 K -1 X -1 Z -1 Q -2 H -2 I -2 M -2 W -2 V -2 F -3 Y -3 L -4
T 4 S 2 A 1 N 0 I 0 V 0 B 0 D -1 G -1 K -1 M -1 P -1 X -1 R -2 Q -2 E -2 Z -2 C -3 H -3 L -3 Y -3 F -4 W -6
V 5 I 3 L 1 M 1 A 0 T 0 X -1 G -2 P -2 S -2 R -3 N -3 D -3 C -3 Q -3 E -3 H -3 F -3 Y -3 B -3 Z -3 K -4 W -8
W 12 R 1 F -1 S -2 Y -2 H -3 L -3 N -4 K -5 X -5 Q -6 I -6 M -6 T -6 B -6 A -7 P -7 Z -7 D -8 C -8 E -8 G -8 V -8
Y 8 F 4 C -1 H -1 N -2 I -2 L -2 W -2 S -3 T -3 V -3 X -3 B -3 A -4 M -4 R -5 D -5 Q -5 E -5 K -5 Z -5 G -6 P -6
X -2 A -1 N -1 D -1 Q -1 E -1 I -1 S -1 T -1 V -1 B -1 R -2 G -2 H -2 L -2 K -2 M -2 P -2 Z -2 F -3 Y -3 C -4 W -5
B 4 D 4 N 3 E 3 Z 2 H 1 A 0 Q 0 G 0 K 0 S 0 T 0 X -1 R -2 P -2 I -3 Y -3 V -3 L -4 M -4 F -5 C -6 W -6
Z 4 Q 4 E 4 D 3 B 2 H 1 N 0 A -1 R -1 K -1 P -1 S -1 X -1 G -2 M -2 T -2 I -3 L -3 V -3 Y -5 F -6 C -7 W -7
```

Σχήμα 5.1: Η μορφή που ο πίνακας PAM 120 αποθηκεύεται σε αρχείο.

Αφού αποθηκευτούν οι τιμές για τα στοιχεία του πίνακα, υπολογίζονται και αποθηκεύονται οι τιμές για τη μέγιστη, ελάχιστη και μέση βαθμολόγηση.
Η κλάση *ScoringMatrix* διαθέτει ορισμένες μεθόδους που επιστρέφουν τιμές είτε των στοιχείων του πίνακα είτε των άλλων παραμέτρων του. Για την επιστροφή της βαθμολόγησης υπάρχει μία μέθοδος που επιστρέφει την βαθμολόγηση μεταξύ δύο συμβόλων *x*, *y*, που δίνονται ως παράμετροι, όπως επίσης και μία μέθοδος που επιστρέφει το στοιχείο *i* της γραμμής του συμβόλου *x*, όπου τα *i*, *x* δίνονται ως παράμετροι. Υπάρχουν επίσης μέθοδοι που επιστρέφουν τις τιμές της μέγιστης , μέσης, ελάχιστης βαθμολόγησης, καθώς και το όνομα το πίνακα (για παράδειγμα BLOSUM 62).

5.2.3. Η κλάση Transition

Όπως αναφέραμε και στην Ενότητα 4.1.2.2 η χρήση ενός πεπερασμένου μη ντετερμινιστικού αυτόματου για τη σάρωση της βάσης δεδομένων αποτελεί την ταχύτερη λύση [1]. Γι' αυτό το λόγο επιλέξαμε κι εμείς την μέθοδο αυτή για την υλοποίησή μας. Η κλάση *Transition* υλοποιεί μια απλή μετάβαση για το αυτόματο που χρησιμοποιούμε. Για κάθε μετάβαση αποθηκεύεται ο χαρακτήρας για τον οποίο χρησιμοποιείται και η κατάσταση του αυτόματου στην οποία οδηγεί. Η κατασκευή μιας μετάβασης γίνεται με απλή ανάθεση τιμών στις παραμέτρους της. Η κλάση *Transition* έχει μεθόδους που επιστρέφουν το χαρακτήρα για τον οποίο χρησιμοποιείται, την τελική κατάσταση, καθώς και μια μέθοδο που ενημερώνει την τιμή της τελικής κατάστασης.

5.2.4. Η κλάση AcceptingTransition

Η κλάση αυτή υλοποιεί τις μεταβάσεις αποδοχής του αυτόματου. Δημιουργείται από την κλάση *Transition*, χρησιμοποιώντας τον μηχανισμό της κληρονομικότητας που διαθέτει η Java για τις κλάσεις της [18]. Έτσι, κληρονομεί από την *Transition* τις παραμέτρους του χαρακτήρα και της κατάστασης που οδηγεί. Επίσης σε κάθε κατάσταση αποδοχής αποθηκεύονται και τα σημεία έναρξης των λέξεων οι οποίες οδηγούν σε αυτή την κατάσταση,

σε έναν πίνακα μεταβλητού μεγέθους (κλάση *ArrayList* της Java [18]). Κατά την κατασκευή μιας κατάστασης αποδοχής, εκτός από την ανάθεση τιμών στις παραμέτρους που κληρονομεί, προστίθεται ένα σημείο έναρξης που δίνεται ως παράμετρος στη δομή που αποθηκεύονται στην κατάσταση αποδοχής. Η κλάση *AcceptingTransition* έχει μεθόδους που επιστρέφουν τις τιμές των δεδομένων που αποθηκεύει. Υπάρχουν επίσης διαθέσιμες μία μέθοδος για την προσθήκη ενός σημείου έναρξης στη δομή που αποθηκεύονται όπως επίσης και μία μέθοδος που επιστρέφει τη δομή αυτή.

5.2.5. Η κλάση State

Όπως φανερώνει και το όνομά της, η κλάση *State* υλοποιεί μια κατάσταση του αυτόματου. Αν και το αυτόματο θα μπορούσε να υλοποιηθεί περιγράφοντας αποκλειστικά τις μεταβάσεις, η υλοποίηση με τη χρήση και χωριστών δομών για τις καταστάσεις κάνει την κατασκευή του πιο κατανοητή και τη χρήση του πιο εύκολη. Για κάθε κατάσταση του αυτόματου αποθηκεύεται η ακολουθία συμβόλων που οδηγεί σε αυτή την κατάσταση και ένας πίνακας μεταβάσεων, μία για κάθε σύμβολο του αλφαβήτου που χρησιμοποιούμε, οι οποίες υλοποιούνται σαν αντικείμενα της κλάσης *Transition*. Η κατασκευή μιας κατάστασης περιλαμβάνει την ανάθεση τιμής στην παράμετρο της ακολουθίας που αποθηκεύεται σε αυτή, καθώς και τη δημιουργία ενός πίνακα μεταβάσεων. Κάθε μετάβαση αρχικοποιείται έτσι ώστε η τελική κατάσταση να είναι η αρχική κατάσταση του αυτόματου.
Οι μέθοδοι που είναι διαθέσιμες για την κλάση *State* περιλαμβάνουν τη μέθοδο που επιστρέφει την ακολουθία που αποθηκεύεται σε αυτή, μία μέθοδο που επιστρέφει την επόμενη κατάσταση με βάση το σύμβολο *x* που δίνεται ως παράμετρος, μία μέθοδο που επιστρέφει την μετάβαση που χρησιμοποιείται για το σύμβολο αυτό, καθώς και δύο μεθόδους για την επεξεργασία των μεταβάσεων. Η μία μέθοδος παίρνει ως παράμετρο ένα σύμβολο *x* και μια κατάσταση *S* και ενημερώνει την μετάβαση από την τρέχουσα κατάσταση για το σύμβολο *x* έτσι ώστε να οδηγεί στην νέα κατάσταση *S*. Η μέθοδος αυτή χρησιμοποιεί τη μέθοδο της κλάσης *Transition* που ενημερώνει την τιμή της τελικής κατάστασης. Η δεύτερη μέθοδος παίρνει ως παράμετρο ένα σύμβολο *x*, ένα σημείο έναρξης *i*, και την αρχική κατάσταση του αυτόματου S_0. Κατόπιν ελέγχει αν η μετάβαση για το σύμβολο *x* είναι μετάβαση αποδοχής. Αν είναι, τότε καλεί τη μέθοδό της για την προσθήκη του σημείου έναρξης *i*. Αν δεν είναι, τότε δημιουργεί μια μετάβαση αποδοχής για το σύμβολο *x* προς την αρχική κατάσταση S_0, αποθηκεύοντας σε αυτή το σημείο έναρξης *i*. Στη συνέχεια αντικαθιστά την παλιά μετάβαση για το σύμβολο *x* με τη νέα.

5.2.5. Η κλάση HSP

Η κλάση *HSP* χρησιμοποιείται για την αποθήκευση όλων των απαραίτητων πληροφοριών για ένα ζεύγος τμημάτων υψηλής βαθμολόγησης που εντοπίζεται στο τρίτο βήμα του αλγόριθμου BLAST. Οι πληροφορίες αυτές περιλαμβάνουν τα σημεία έναρξης στις ακολουθίες εισόδου και βάσης δεδομένων, το μήκος του ζεύγους, τη βαθμολόγησή του και την παράμετρο *E*. Κατά την κατασκευή ενός αντικειμένου της κλάσης *HSP*, δίνονται ως παράμετροι τα σημεία έναρξης στις δύο ακολουθίες, ενώ οι τιμές του μήκους

και βαθμολόγησης αρχικοποιούνται στην τιμή μηδέν. Η κλάση *HSP* έχει διαθέσιμες μεθόδους που επιστρέφουν τις τιμές για τις παραμέτρους που προαναφέραμε. Η σημαντικότερη όμως μέθοδος για την κλάση *HSP* είναι αυτή που κάνει την επέκτασή του όπως την παρουσιάσαμε στην Ενότητα 4.1.2.3.

Η αρχικοποίηση ενός αντικειμένου της κλάσης *HSP* ισοδυναμεί με τη δημιουργία ενός ζεύγους τμημάτων για μια επιτυχία που εντοπίστηκε κατά τη σάρωση της βάσης δεδομένων. Στη συνέχεια, κάθε τέτοια επιτυχία επεκτείνεται έτσι ώστε να υπολογιστεί αν είναι μέρος ενός ζεύγους τμημάτων υψηλής βαθμολόγησης. Για να γίνει η επέκταση ακολουθούμε την ίδια διαδικασία και προς τις δύο κατευθύνσεις.

Έστω ότι έχουμε δύο ακολουθίες q, s, για τις οποίες ένα ζεύγος τμημάτων έχει επεκταθεί μέχρι τις θέσεις i, j αντίστοιχα. Από τον πίνακα βαθμολόγησης βρίσκουμε τη βαθμολόγηση για τα σύμβολα $q[i+1]$, $s[j+1]$. Αν αυτή είναι θετική, τότε επεκτείνουμε τα όρια του ζεύγους στη θέση $i+1$, $j+1$ και επαναλαμβάνουμε τη διαδικασία. Αν είναι αρνητική, τότε δεν επεκτείνουμε το ζεύγος. Ξεκινώντας τώρα από τις θέσεις $i+1$, $j+1$, υπολογίζουμε τη βαθμολόγηση της αντιστοίχισης από τη θέση αυτή και πάνω. Αν δηλαδή εξετάσουμε τις επόμενες k θέσεις, θα πρέπει να υπολογίσουμε τη βαθμολόγηση της αντιστοίχισης $q[i+1..i+k]$, $s[j+1..j+k]$. Σε κάθε βήμα ελέγχουμε την τιμή αυτής της βαθμολόγησης.

Αν γίνει μικρότερη από την τιμή $-X$ (Ενότητα 4.3) τότε σταματάμε την επέκταση. Τα όρια του ζεύγους σε αυτή την περίπτωση είναι στις θέσεις i, j, οι οποίες είναι οι τελευταίες θέσεις για τις οποίες είχαμε θετική αντιστοίχιση. Επομένως μέχρι εκείνες τις θέσεις η βαθμολόγηση του ζεύγους αυξανόταν.

Αν η τιμή αυτής της βαθμολόγησης για κάποια τιμή του k γίνει θετική, τότε εκτείνουμε το ζεύγος τμημάτων μέχρι τη θέση $i+k$, $j+k$, ενημερώνουμε τη βαθμολόγηση και επαναλαμβάνουμε τη διαδικασία. Στη θέση αυτή, η βαθμολόγηση του ζεύγους είναι μεγαλύτερη από τη θέση i, j. Επομένως, με τον τρόπο αυτό, η επέκταση σταματάει όταν η διαφορά της τρέχουσας βαθμολόγησης από τη μέγιστη είναι μεγαλύτερη από την παράμετρο X, αλλά στο ζεύγος τμημάτων δεν περιλαμβάνονται τα σύμβολα για τα οποία παρουσιάστηκε αυτή η διαφορά. Τα όρια δηλαδή της εκτεταμένης επιτυχίας είναι τέτοια ώστε να έχει το ζεύγος των τμημάτων τη μέγιστη βαθμολόγηση. Στο Παράδειγμα 5.2 έχουμε μια εφαρμογή της παραπάνω διαδικασίας για την επέκταση μιας επιτυχίας για δύο ακολουθίες.

Παράδειγμα 5.2

*Ας υποθέσουμε ότι έχουμε δύο ακολουθίες πρωτεϊνών για τις οποίες έχουμε επιτυχία στα σημεία (*i, j*) όπως φαίνεται στο Σχήμα 5.2. Χρησιμοποιούμε τον πίνακα βαθμολόγησης PAM 120 και* X*=20 και εφαρμόζουμε την παραπάνω διαδικασία.*

```
i …MIKLSENQPWVCWHGI…

j …MIKLDDCFPWRNCTRN…
```

Σχήμα 5.2: Τμήματα ακολουθιών για το παράδειγμα 5.2.

Τα σύμβολα στις θέσεις i, i+*1,* i+*2,* i+*3 ταυτίζονται με αυτά στις* j, j+*1,* j+*2,* j+*3, επομένως οι βαθμολογήσεις τους είναι θετικές. Το ζεύγος λοιπόν επεκτείνεται απ' ευθείας μέχρι τις θέσεις* i+*3,* j+*3 και η βαθμολόγησή του είναι* S_3=*8+6+5+5=24.*

Τα σύμβολα στις θέσεις i+*4,* j+*4 δεν ταυτίζονται όμως από τον πίνακα βαθμολόγησης είναι* M*(S,D)=0. Η βαθμολόγηση* M*(E,D)=+3 για τα σύμβολα στις θέσεις* i+*5,* j+*5 είναι επίσης θετική, επομένως το ζεύγος επεκτείνεται απ' ευθείας ως τις θέσεις* i+*5,* j+*5 με βαθμολόγηση* S_5=S_3+*0+3=27.*

Η βαθμολόγηση M*(N,C)=-5 για τις θέσεις* i+*6,* j+*6 είναι αρνητική επομένως το ζεύγος δεν επεκτείνεται απ' ευθείας. Αντίθετα, αρχίζουμε να υπολογίζουμε τη βαθμολόγηση από αυτό το σημείο και έπειτα. Η βαθμολόγηση αυτή είναι* D=*-5 και επειδή είναι* X = *20 άρα* D>-X *συνεχίζουμε τον έλεγχο για την επέκταση.*

Για τις θέσεις i+*7,* j+*7 έχουμε* M*(Q,F)=-6, επομένως έχουμε* D=*-11 >-*X*, οπότε συνεχίζουμε τον έλεγχο.*

Για τις θέσεις i+*8,* j+*8 έχουμε* M*(P,P)=6, επομένως* D=*-5. Είναι* D<*0 επομένως παρόλο που τα σύμβολα ταυτίζονται δεν επεκτείνουμε το ζεύγος.*

Για τις θέσεις i+*9,* j+*9 έχουμε* M*(W,W)=12, επομένως* D=*7. Επειδή η τιμή της απόστασης* D *είναι θετική επεκτείνουμε το ζεύγος μέχρι τις θέσεις αυτές. Η νέα βαθμολόγηση θα είναι* S_9=S_5+D=*34.*

Για τις θέσεις i+*10,* j+*10 έχουμε* M*(V,R)=-3, επομένως δεν επεκτείνουμε και* D=*-3.*

Για τις θέσεις i+*11,* j+*11,* M*(C,N)=-5, επομένως* D=*-8.*

Για τις θέσεις i+*12,* j+*12,* M*(W,C)=-8,* D=*-16.*

Για τις θέσεις i+13, j+13, M(H,T)=-3, D=-19.

Για τις θέσεις i+*14,* j+*14,* M*(G,R)=-4,* D=*-23*<-X*, επομένως σταματάμε τον έλεγχο για επέκταση. Τα όρια της εκτεταμένης επιτυχίας βρίσκονται στις θέσεις* i+*9,* j+*9 και η βαθμολόγησή της είναι* S=*34.*

Στο Σχήμα 5.3 παρουσιάζουμε τον αλγόριθμο για την επέκταση των επιτυχιών προς τη μία κατεύθυνση, που περιγράψαμε παραπάνω σε μορφή ψευδοκώδικα.

```
Algorithm Extension
    input: sequences q, s, starting points qsp, ssp, scoring matrix M
    output: length of HSP, score of HSP

    i ← 0
    length ← 0
    score ← 0
    D ← 0
    while qsp+i ≤ |q| and ssp+i ≤ |s|
        D ← D + M[q[qsp+i], s[ssp+i]]
        if D ≥ 0
            length = i
            score = score +D
        else if D < -X
            return length, score
        i ← i +1
    return length, score
```

Σχήμα 5.3: Αλγόριθμος για την επέκταση των επιτυχιών.

5.2.6. Η κλάση Sequence

Η κλάση αυτή χρησιμοποιείται για να αποθηκεύουμε μια ακολουθία που διαβάζουμε από τη βάση. Στην κλάση *Sequence* αποθηκεύονται απλά το όνομα και τον διακριτικό κωδικό της ακολουθίας όπως υπάρχουν στη μορφή FASTA της βάσης δεδομένων (Ενότητα 2.2.) και την ακολουθία με τη μορφή συμβολοσειράς. Η κατασκευή γίνεται με απλή ανάθεση τιμών στα δύο αυτά πεδία. Υπάρχουν ακόμα διαθέσιμες μέθοδοι που επιστρέφουν τα δεδομένα αυτά, όπως επίσης και μέθοδοι που επιστρέφουν το μήκος, το χαρακτήρα σε κάποια θέση που δίνεται ως παράμετρος καθώς και μια μέθοδος που επιστρέφει μια υποακολουθία της σε μορφή συμβολοσειράς.

5.2.7. Η κλάση MatchingSequence

Η κλάση *MatchingSequene* προκύπτει από την κλάση *Sequence* χρησιμοποιώντας και πάλι το μηχανισμό κληρονομικότητας των κλάσεων. Χρησιμοποιείται για την υλοποίηση μιας ακολουθίας που παρουσιάζει ομοιότητα με την ακολουθία που εξετάζουμε. Για κάθε τέτοια ακολουθία θα πρέπει να αποθηκεύσουμε τα ζεύγη τμημάτων που παρουσιάζουν ομοιότητα με την ακολουθία εισόδου. Εκτός από τα πεδία που κληρονομεί από την κλάση *Sequence*, στην κλάση αυτή αποθηκεύουμε και μια δομή που περιλαμβάνει τα ζεύγη τμημάτων υψηλής βαθμολόγησης. Επίσης αποθηκεύουμε τη συνολική βαθμολόγηση και το συνολικό μήκος για αυτά τα τμήματα. Η κλάση Sequence εκτός από τις μεθόδους που κληρονομεί από την κλάση Sequence, διαθέτει μια μέθοδο που επιστρέφει την τιμή της συνολικής βαθμολόγησης καθώς και μια μέθοδο που εμφανίζει τα ζεύγη υψηλής βαθμολόγησης κατά ζεύγη (Ενότητα 4.4.2).

5.2.8. Η κλάση Diagonal

Χρησιμοποιούμε την κλάση *Diagonal* ως μια δομή για να αποθηκεύουμε τις απαραίτητες πληροφορίες για τις διαγωνίους των επιτυχιών (Ενότητα 4.1.2.3). Για κάθε διαγώνιο αποθηκεύουμε την τιμή της (όπως προκύπτει από τη διαφορά των σημείων έναρξης) και το πιο «προχωρημένο» τελικό σημείο ενός ζεύγους τμημάτων για αυτή τη βαθμολόγηση. Η κατασκευή ενός αντικειμένου κλάσης *Diagonal* γίνεται με ανάθεση τιμών στα πεδία αυτά. Η κλάση διαθέτει μεθόδους που επιστρέφουν τις τιμές των πεδίων αυτών, καθώς και μία μέθοδο που ενημερώνει την τιμή του τελικού σημείου.

5.2.9. Η κλάση DFA

Η κλάση *DFA* αποτελεί την κύρια δομή του αλγόριθμου. Με την κλάση αυτή υλοποιούμε το αυτόματο που χρησιμοποιείται για τη σάρωση της βάσης δεδομένων [1]. Στη δομή και τη λειτουργία του χρησιμοποιούνται όλες οι προηγούμενες δομές που περιγράψαμε.
Τα δεδομένα που αποθηκεύουμε στη δομή του αυτόματου περιλαμβάνουν τις πληροφορίες για τη δομή του, τις τιμές που χρησιμοποιούμε για τις παραμέτρους του αλγόριθμου BLAST και τις πληροφορίες για τα αποτελέσματα. Πιο αναλυτικά, οι πληροφορίες που αποθηκεύουμε στην κλάση *DFA* περιλαμβάνουν:

- Το σύνολο των καταστάσεων του αυτόματου. Κάθε κατάσταση υλοποιείται σαν αντικείμενο της κλάσης *State*.
- Την αρχική κατάσταση. Υλοποιείται ως αντικείμενο της κλάσης *State*. Η αρχική κατάσταση αποθηκεύεται χωριστά κυρίως για λόγους καλύτερης κατανόησης και ευκολίας.
- Τον τύπο του αυτόματου, αν δηλαδή χρησιμοποιείται για ακολουθίες DNA ή πρωτεϊνών.
- Το αλφάβητο που χρησιμοποιείται για τις ακολουθίες, ανάλογα με τον τύπο τους. Όπως αναφέραμε στην Ενότητα 5.2.2, χρησιμοποιούμε τα αλφάβητα που περιλαμβάνουν τα αμφίσημα σύμβολα.
- Τον πίνακα βαθμολόγησης που χρησιμοποιούμε και υλοποιείται σαν αντικείμενο της κλάσης *ScoringMatrix*.
- Την ακολουθία *Q* που εξετάζουμε σε μορφή συμβολοσειράς.
- Το μήκος λέξεων *w* που χρησιμοποιούμε.
- Την βαθμολόγηση κατωφλίου *T*, που χρησιμοποιείται μόνο για πρωτεΐνες.
- Τις παραμέτρους *K*, *λ*, που υπολογίζονται με βάση την ακολουθία εισόδου και τον πίνακα βαθμολόγησης και χρησιμοποιούνται για την στατιστική αξιολόγηση των αποτελεσμάτων.
- Την παράμετρο *E* που χρησιμοποιούμε ως κριτήριο στατιστικής σημασίας.
- Τις παραμέτρους E_2, S_2 που χρησιμοποιούνται για τον εντοπισμό των ζευγών υψηλής βαθμολόγησης. Η παράμετρος E_2 είναι σταθερή και ίση με 0.15, ενώ η παράμετρος S_2 υπολογίζεται με βάση τη σχέση (4.1) με n=m=1000 για ακολουθίες DNA και n=m=300 για πρωτεΐνες.

- Την παράμετρο *X* για την οποία σταματάμε την επέκταση μιας επιτυχίας.
- Το μέγεθος της βάσης δεδομένων, μετρούμενο σε αριθμό συμβόλων, το οποίο υπολογίζεται κατά τη σάρωσή της.
- Τη λίστα των ακολουθιών της βάσης που παρουσιάζουν ομοιότητες με την ακολουθία που εξετάζουμε. Κάθε μία από αυτές υλοποιείται σαν αντικείμενο της κλάσης *MatchingSequence*.

Η κατασκευή του αυτόματου, όπως και η κατασκευή της λίστας λέξεων, διαφέρει για τους δύο τύπους ακολουθιών που εξετάζουμε. Έτσι λοιπόν έχουμε διαθέσιμες δύο μεθόδους κατασκευής για το αυτόματο ανάλογα με τον τύπο του. Και στις δύο περιπτώσεις, η κατασκευή απλοποιείται αν κατασκευάσουμε πρώτα τη λίστα λέξεων, χρησιμοποιώντας τους αλγόριθμους των Σχημάτων 4.2, 4.3 και 4.4 Κάτι τέτοιο όμως θα απαιτούσε την αποθήκευση της λίστας λέξεων σε μια ξεχωριστή δομή, συνεπώς αυξημένες απαιτήσεις σε μνήμη. Γι' αυτό το λόγο, προσαρμόζουμε τους αλγόριθμους αυτούς έτσι ώστε αντί των λέξεων να κατασκευάζονται απ' ευθείας οι απαιτούμενες καταστάσεις και μεταβάσεις του αυτόματου.

Ακολουθίες DNA

Για ακολουθίες DNA παίρνουμε μια συνεχή υποακολουθία μήκους *w*. Ξεκινώντας από την αρχική κατάσταση του αυτόματου και το πρώτο σύμβολο της υποακολουθίας, ελέγχουμε αν υπάρχει μετάβαση για αυτό το σύμβολο προς μια άλλη κατάσταση εκτός της αρχικής. Αν υπάρχει, τότε ακολουθούμε αυτή τη μετάβαση και εφαρμόζουμε την ίδια διαδικασία για αυτή τη νέα κατάσταση και το επόμενο σύμβολο της υποακολουθίας. Αν δεν υπάρχει μετάβαση προς άλλη κατάσταση εκτός της αρχικής, δημιουργούμε ένα νέο αντικείμενο της κλάσης State και το αποθηκεύουμε στη λίστα καταστάσεων του αυτόματου. Στη συνέχεια, ενημερώνουμε την μετάβαση από την κατάσταση που βρισκόμαστε και για το σύμβολο που εξετάζουμε, ώστε να οδηγεί στη νέα κατάσταση που δημιουργήσαμε, χρησιμοποιώντας την μέθοδο της κλάσης *State* για την ενημέρωση των μεταβάσεων. Έπειτα χρησιμοποιούμε την ενημερωμένη μετάβαση και επαναλαμβάνουμε τη διαδικασία. Όταν φτάσουμε στο τελευταίο σύμβολο της υποακολουθίας, χρησιμοποιούμε τη μέθοδο της κλάσης *State* που μεταβάλλει την κατάλληλη μετάβαση σε μετάβαση αποδοχής, ή, αν είναι ήδη μετάβαση αποδοχής, προσθέτει το σημείο έναρξης στη δομή της κλάσης *AcceptingTransition* που αποθηκεύονται. Αυτή η διαδικασία σε ψευδοκώδικα φαίνεται στο Σχήμα 5.4.

```
Algorithm DFAforDNA
    input: sequence q, word length w
    output: a DFA that accepts all w-length words of q

    create state S0
    for i ← 1 to |q|-w+1 do
        current state ← S0
        for j ← 1 to w do
            if j ≠ w
                find next state from current state for symbol q[i+j]
                if next state = S0
                    create state new state
                    set transition from current state for symbol q[i+j] to new state
                    current state ← new state
                else
                    current state ← next state
            else
                find transition from current state for symbol q[i+j]
                if transition is accepting
                    add i to starting points of transition
                else
                    make transition accepting with starting point i
```

Σχήμα 5.4: Ο αλγόριθμος για την κατασκευή του αυτόματου για ακολουθίες DNA.

Ακολουθίες πρωτεϊνών

Για την περίπτωση του αυτόματου για ακολουθίες πρωτεϊνών, όπως και με τη λίστα λέξεων, η διαδικασία είναι πιο πολύπλοκη. Και πάλι, η κατασκευή του αυτόματου γίνεται με μια αναδρομική μέθοδο, η οποία βασίζεται στον αλγόριθμο του Σχήματος 4.4.

Για κάθε συνεχή υποακολουθία μήκους *w* της ακολουθίας που εξετάζουμε, υπολογίζουμε τη μέγιστη βαθμολόγηση *MaxScore*. Κατόπιν, για κάθε θέση υπολογίζουμε τον αριθμό των διαθέσιμων συμβόλων, με βάση τη διαφορά *D=MaxScore-T*. Αν η τιμή αυτή είναι αρνητική, τότε η λέξη που διαβάζουμε από την ακολουθία εισόδου δεν ικανοποιεί την βαθμολόγηση κατωφλίου *T*, επομένως δε συνεχίζουμε τη διαδικασία, αφού καμία λέξη δε θα δίνει μεγαλύτερη βαθμολόγηση. Αν η τιμή της παραμέτρου *D* είναι θετική τότε προχωρούμε στον υπολογισμό των διαθέσιμων συμβόλων. Επειδή ο πίνακας βαθμολόγησης είναι ταξινομημένος, γνωρίζοντας αυτό τον αριθμό μπορούμε να βρούμε τα διαθέσιμα σύμβολα: σε κάθε περίπτωση θα είναι τα *n* πρώτα σύμβολα της σειράς του πίνακα για το αρχικό σύμβολο. Στη συνέχεια καλούμε μια αναδρομική συνάρτηση για την κατασκευή του αυτόματου. Ως παράμετροι στη συνάρτηση, δίνονται μια συμβολοσειρά *temp*, μία κατάσταση του αυτόματου, μία απόσταση *d*, η οποία αντιπροσωπεύει την μέγιστη διαθέσιμη διαφορά της βαθμολόγησης από τη μέγιστη βαθμολόγηση και το σημείο έναρξης της λέξης που εξετάζουμε. Στην πρώτη κλήση της

συνάρτησης χρησιμοποιούμε ως ορίσματα μία κενή συμβολοσειρά, την αρχική κατάσταση, και την απόσταση *D=MaxScore-T*.
Η συνάρτηση αυτή ελέγχει τα διαθέσιμα σύμβολα για τη θέση που εξετάζουμε. Αν για κάποιο σύμβολο *S* από αυτά η διαφορά της βαθμολόγησής τους από τη μέγιστη βαθμολόγηση για το αρχικό σύμβολο της θέσης είναι μικρότερη από τη διαφορά *d*, δημιουργείται μία νέα συμβολοσειρά με την προσθήκη του συμβόλου *S* στο τέλος της συμβολοσειράς temp. Αν το μήκος της νέας συμβολοσειράς είναι μικρότερο του *w*, εξετάζουμε αν υπάρχει μετάβαση από την κατάσταση που έχουμε σαν όρισμα προς μία άλλη, εκτός της αρχικής. Αν υπάρχει, τότε καλούμε αναδρομικά τη συνάρτηση, με ορίσματα τη νέα συμβολοσειρά, την νέα κατάσταση του αυτόματου και τη νέα διαθέσιμη απόσταση, που υπολογίζεται αν από την απόσταση *d* αφαιρέσουμε τη διαφορά της βαθμολόγησης του συμβόλου *S* με το αρχικό σύμβολο, από τη μέγιστη βαθμολόγηση για το αρχικό σύμβολο. Αν το μήκος της νέας συμβολοσειράς είναι ίσο με *w*, τότε μετατρέπουμε την μετάβαση από την τρέχουσα κατάσταση για το σύμβολο *S* σε κατάσταση αποδοχής, χρησιμοποιώντας την κατάλληλη μέθοδο της κλάσης *State*. Ο αλγόριθμος για την κατασκευή του αυτόματου για ακολουθίες πρωτεϊνών φαίνεται στο Σχήμα 5.5.

Σε αυτό το σημείο, θα πρέπει να σημειώσουμε ότι οι παραπάνω αλγόριθμοι οδηγούν στην κατασκευή ενός αυτόματου που περιέχει όλες τις απαραίτητες καταστάσεις και μεταβάσεις αποδοχής για να δέχεται τις λέξεις που υπάρχουν στη λίστα λέξεων μιας ακολουθίας. Η κατασκευή αυτή όμως είναι τέτοια ώστε, κατά τη σάρωση της βάσης δεδομένων, για κάθε λέξη μιας ακολουθίας που διαβάζουμε απαιτείται επανεκκίνηση του αυτόματου. Επειδή για κάθε λέξη χρησιμοποιούνται ακριβώς w μεταβάσεις και στη βάση δεδομένων υπάρχουν (περίπου) L-w+1 λέξεις, με L το μέγεθος της βάσης μετρούμενο σε σύμβολα, εύκολα βγάζουμε το συμπέρασμα ότι για τη σάρωση της βάσης απαιτείται χρόνος $O(wL)$. Εφαρμόζοντας μια διαδικασία αναπροσαρμογής συγκεκριμένων μεταβάσεων που θα περιγράψουμε στη συνέχεια, μπορούμε να πετύχουμε την αποδοχή όλων των λέξεων χωρίς να χρειάζεται επανεκκίνηση του αυτόματου, χρησιμοποιώντας μία μετάβαση για κάθε σύμβολο. Έτσι πετυχαίνουμε χρόνο σάρωσης της βάσης δεδομένων της τάξης του $O(L)$.

```
Algorithm DFAforProtein
    input: sequence q, word length w, threshold T, scoring matrix M
    output: a DFA that accepts all w-length words that score above T with a w-length
    word of q

    for SP ← 0 to |q|-w+1 do
        MaxScore=0
        for j ← 0 to w do
            MaxScore ← MaxScore + M[q[SP+j], q[SP+j]]
        D ← MaxScore – T
        for j ← 0 to w do
            k ← 1
            while M[q[SP+j], q[SP+j]] - M[q[SP+j],k] ≤ D
                k ← k+1
            available symbols for position j ← k
        DFAConstructor (e, S0, D, SP)

Function DFAConstructor
    input: a string temp, a state S, a distance d, a starting point SP
    output: a set of states and transitions of the DFA for the word starting at SP

    i ← 1
    position ← |temp| +1
    score ← M[q[SP+position], i-th symbol from available symbols]
    while  i ≤ available symbols for position
            and M[q[SP+position], q[SP+position]]- score ≤ d
        new string ← concatenate temp, i-th symbol from available symbols
        if |new string| = w
            find transition from S  for i-th symbol  from available symbols
            make transition accepting with starting point SP
        else
            find next state from current state for i-th symbol  from available symbols
            if next state = S0
                create state new state
                set transition from S  for i-th symbol  from available symbols
                                                    to new state
                next state ← new state
            DFAConstructor (new string, next state, d - M[q[SP+position],
                                q[SP+position]]+ score, SP)
        i ← i+1
```

Σχήμα 5.5: Αναδρομικός αλγόριθμος για την κατασκευή αυτόματου για ακολουθίες πρωτεϊνών.

Η διαδικασία για την αναπροσαρμογή των μεταβάσεων είναι όμοια και για τα δύο είδη ακολουθιών. Ο χρόνος εκτέλεσής της είναι ανάλογος του μεγέθους του αυτόματου, συνεπώς η επιβάρυνση στο συνολικό χρόνο εκτέλεσης είναι μικρότερη από αυτή που θα προκαλούσε η σάρωση της βάσης δεδομένων με επανεκκίνηση του αυτόματου για κάθε λέξη. Για να αποφύγουμε τις

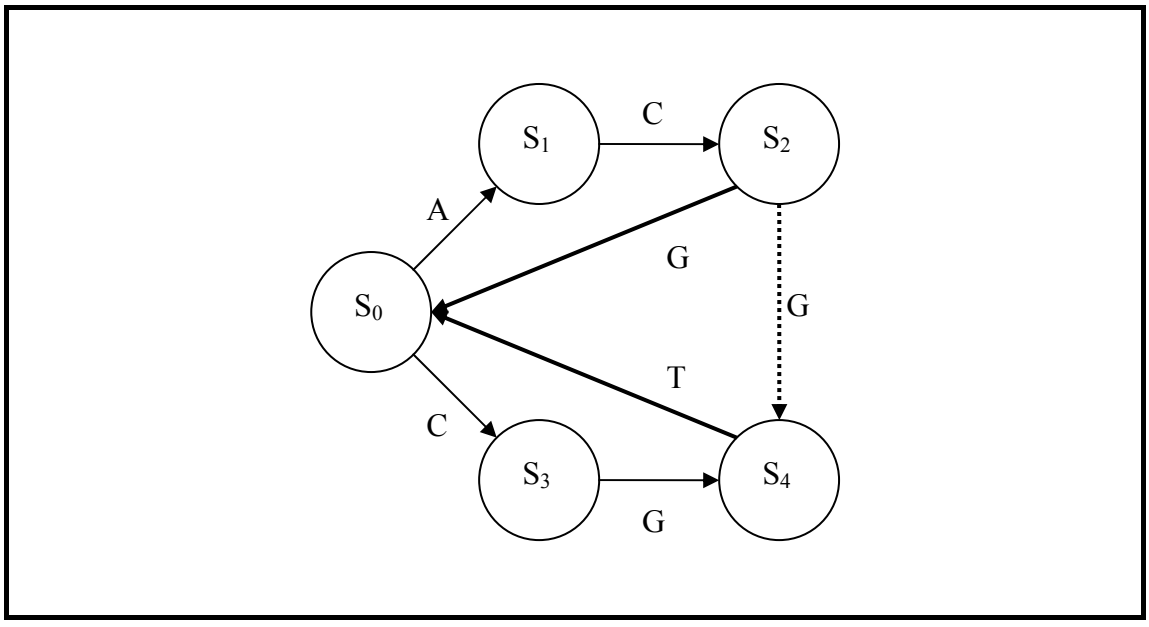
επανεκκινήσεις θα πρέπει να αναπροσαρμόσουμε όσες από τις μεταβάσεις οδηγούν στην αρχική κατάσταση μπορούμε, έτσι ώστε να έχουμε αποδοχή όλων των δυνατών λέξεων με μία απλή σάρωση της ακολουθίας που διαβάζουμε από τη βάση δεδομένων.

Παράδειγμα 5.3

Έστω ότι για μια ακολουθία DNA έχουμε τις λέξεις ACG, CGT. Τότε ο αλγόριθμος του Σχήματος 5.4 μας δίνει το αυτόματο του Σχήματος 5.6. Έστω τώρα ότι κατά τη σάρωση της βάσης προκύπτει ένα τμήμα ακολουθίας:

...ACGT...

Εύκολα μπορούμε να διαπιστώσουμε ότι αν από την κατάσταση 2 διαβάζοντας το σύμβολο G οδηγούμαστε στην κατάσταση 4 αντί της αρχικής, τότε διαβάζοντας το σύμβολο T θα είχαμε απ' ευθείας αποδοχή της λέξης CGT, χωρίς να απαιτείται επανεκκίνηση του αυτόματου. Εκείνο που πρέπει να κάνουμε λοιπόν είναι να μεταβάλουμε την μετάβαση (S_2,G,S_0) *σε* (S_2,G,S_4).



Σχήμα 5.6: Το αυτόματο για το Παράδειγμα 5.3. Το διακεκομμένο βέλος αντιστοιχεί στη νέα μετάβαση που πρέπει να κατασκευάσουμε.

Πιο αναλυτικά, η διαδικασία που ακολουθούμε έχει ως εξής: Ελέγχουμε όλες τις καταστάσεις για μεταβάσεις που οδηγούν στην αρχική κατάσταση. Για κάθε τέτοια μετάβαση, ελέγχουμε αν μπορεί να μεταβληθεί, ώστε να οδηγεί σε κάποια άλλη, με βάση την συμβολοσειρά που οδηγεί στην τρέχουσα κατάσταση και το σύμβολο για το οποίο χρησιμοποιείται η μετάβαση. Πιο συγκεκριμένα, κατασκευάζουμε μία νέα συμβολοσειρά που προέρχεται από

τη συμβολοσειρά που οδηγεί στην τρέχουσα κατάσταση αν προσθέσουμε στο τέλος της το σύμβολο για το οποίο έχουμε μετάβαση προς την αρχική κατάσταση. Στη συνέχεια, εξετάζουμε αν υπάρχει κατάσταση στην οποία να οδηγούμαστε από την αρχική, για κάποια γνήσια κατάληξη της συμβολοσειράς αυτής (proper suffix, κατάληξη με μήκος *len*<|*s*|, όπου *s* η συμβολοσειρά). Για τον έλεγχο αυτό, ξεκινάμε για κάθε κατάληξη (αρχίζοντας από τη μεγαλύτερη) από την αρχική κατάσταση και ακολουθούμε τις υπάρχουσες μεταβάσεις για κάθε σύμβολο. Για την κατάσταση που καταλήγουμε ελέγχουμε αν η συμβολοσειρά που οδηγεί σε αυτή είναι ίδια με αυτή που χρησιμοποιήσαμε. Αν είναι, τότε αλλάζουμε την μετάβαση έτσι ώστε να οδηγεί στην κατάσταση που μόλις εντοπίσαμε. Αν δεν είναι ίδια, δοκιμάζουμε την επόμενη (μικρότερη) κατάληξη με τον ίδιο τρόπο. Στην περίπτωση που δεν υπάρχει κατάσταση για καμία κατάληξη, η μετάβαση δεν αλλάζει και παραμένει προς την αρχική κατάσταση. Να σημειώσουμε εδώ, ότι κατά τη διαδικασία αυτή, δεν έχει σημασία αν πρόκειται για απλή μετάβαση ή μετάβαση αποδοχής. Ο αλγόριθμος για αυτή τη διαδικασία φαίνεται στο Σχήμα 5.7.

```
Algorithm FinalizeDFA
    input: a DFA that accepts all words for a sequence q, an alphabet
    output: a DFA that scans a sequence s in O(|s|) time

    for each state in DFA do
        for each symbol in alphabet do
            find next state from state for symbol
            if next state = S0
                new string ← concatenate string for state, symbol
                for i ← 2 to |new string|
                    new string ← substring new string[i..|new string|]
                    current state ← S0
                    for j ← 1 to |new string|
                        find new state from current state for new string[j]
                        current state ← new state
                    if string for current state = new string
                        set transition from state for symbol to current state
                        break inmost for-loop
```

Σχήμα 5.7: Αλγόριθμος για την προσαρμογή των μεταβάσεων.

Παράδειγμα 5.4

Έστω το αυτόματο του Σχήματος 5.8.α) το οποίο δέχεται τη λέξη ATT μιας ακολουθίας DNA και σε κάθε κατάσταση σημειώνουμε τη συμβολοσειρά που οδηγεί σε αυτή. Για ευκολία στην απεικόνιση, δεν εμφανίζεται ξεχωριστά η μετάβαση αποδοχής. Για το αυτόματο αυτό έχουμε τον ακόλουθο πίνακα μεταβάσεων:

Κατάσταση	Μετάβαση	Σύμβολο	Νέα κατάσταση
e	1	A	A
	2	C	e
	3	G	e
	4	T	e
A	**5**	A	e
	6	C	e
	7	G	e
	8	T	AT
AT	**9**	A	e
	10	C	e
	11	G	e
	12	T	e

Στον πίνακα αυτό, αντί για αρίθμηση των καταστάσεων χρησιμοποιούμε τις συμβολοσειρές που οδηγούν σε αυτές, ενώ οι μεταβάσεις που πρέπει να ελέγξουμε φαίνονται με έντονους χαρακτήρες.

Κατάσταση e:

- *Μετάβαση 2: Δημιουργούμε τη συμβολοσειρά* e*C=C. Η συμβολοσειρά αυτή δεν έχει γνήσιες καταλήξεις επομένως δεν μπορούμε να αλλάξουμε την μετάβαση.*
- *Μετάβαση 3: Δημιουργούμε τη συμβολοσειρά* e*G=G. Όμοια με παραπάνω.*
- *Μετάβαση 4: Δημιουργούμε τη συμβολοσειρά* e*T=T. Όμοια με παραπάνω.*

Κατάσταση A:

- *Μετάβαση 5: Δημιουργούμε τη συμβολοσειρά AA. Η μόνη γνήσια κατάληξη είναι A. Ξεκινώντας από την αρχική κατάσταση, ακολουθώντας τη μετάβαση για A (1), οδηγούμαστε στην κατάσταση A, όπου προφανώς A=A. Επομένως αλλάζουμε την μετάβαση 5 ώστε η νέα κατάσταση να είναι η A.*
- *Μετάβαση 6: Δημιουργούμε τη συμβολοσειρά AC. Η μόνη γνήσια κατάληξη είναι η C. Ξεκινώντας από την αρχική κατάσταση, ακολουθώντας την μετάβαση για C (2), οδηγούμαστε στην κατάσταση* e*, για την οποία ισχύει C≠*e*, επομένως δεν αλλάζουμε την μετάβαση*
- *Μετάβαση 7: Δημιουργούμε τη συμβολοσειρά AG. Όμοια με την μετάβαση 6.*

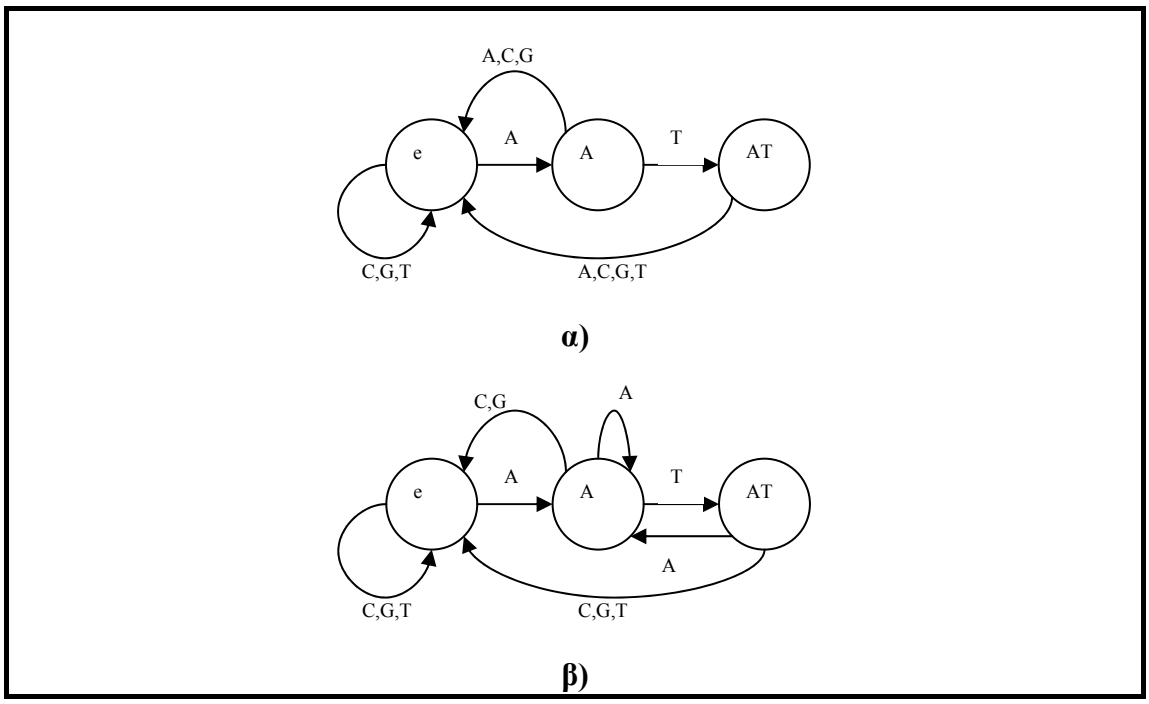
Κατάσταση AT:

- *Μετάβαση 9: Δημιουργούμε τη συμβολοσειρά ATA. Για την κατάληξη TA ξεκινάμε από την αρχική κατάσταση και ακολουθούμε τη μετάβαση για T (4) που οδηγεί στην κατάσταση* e *και στη συνέχεια τη μετάβαση για A (1) που οδηγεί στην κατάσταση A. Επειδή TA≠A, δεν αλλάζουμε την μετάβαση και δοκιμάζουμε την επόμενη κατάληξη. Η επόμενη κατάληξη είναι*

Α, για την οποία ξεκινώντας από την αρχική κατάσταση και ακολουθώντας τη μετάβαση για Α (1) οδηγούμαστε στην κατάσταση Α, όπου Α=Α. Επομένως αλλάζουμε την μετάβαση 9 ώστε η νέα κατάσταση να είναι η Α.

- *Μετάβαση 10: Δημιουργούμε τη συμβολοσειρά ATC. Ακολουθώντας την ίδια διαδικασία, βλέπουμε ότι δεν οδηγούμαστε για κάποια κατάληξη σε έγκυρη κατάσταση, επομένως δεν αλλάζουμε την μετάβαση.*
- *Μετάβαση 11: Δημιουργούμε τη συμβολοσειρά ATG. Όμοια με την μετάβαση 10.*
- *Μετάβαση 12: Δημιουργούμε τη συμβολοσειρά ATT. Όμοια με τη μετάβαση 10.*

Το αυτόματο μετά τις αλλαγές των καταστάσεων φαίνεται στο Σχήμα 5.8.β).



Σχήμα 5.8: Το αυτόματο του παραδείγματος 5.4. α) πριν τις αλλαγές μεταβάσεων β) μετά τις αλλαγές.

Μετά και το τέλος του δεύτερου βήματος κατασκευής που περιγράψαμε παραπάνω, η δομή του αυτόματου είναι ολοκληρωμένη. Στη συνέχεια λοιπόν ορίζονται οι τιμές για τις υπόλοιπες τιμές των παραμέτρων που αποθηκεύονται στην κλάση DFA. Έτσι υπολογίζουμε τις τιμές των K, λ [7] και αναθέτουμε τιμές για τις υπόλοιπες, με βάση τις επιλογές του χρήστη.
Οι μέθοδοι που είναι διαθέσιμες στην κλάση DFA είναι αρκετά περιορισμένες, καθώς το μεγαλύτερο μέρος της λειτουργικότητας έχει μεταφερθεί στις βοηθητικές κλάσεις. Υπάρχει μία μέθοδος που ελέγχει μία κατά πόσο μία ακολουθία αποτελείται από έγκυρα σύμβολα, από σύμβολα δηλαδή που ανήκουν στο αλφάβητο του τύπου της. Για την ακολουθία εισόδου, ο έλεγχος γίνεται πριν την κατασκευή του αυτόματου και αν η

ακολουθία δεν είναι έγκυρη η διαδικασία τερματίζεται. Ελέγχονται επίσης και οι ακολουθίες που διαβάζονται από τη βάση, αν όμως βρεθεί κάποια που δεν είναι έγκυρη απλά αγνοείται. Η σημαντικότερη όμως μέθοδος για την κλάση *DFA* είναι αυτή που εκτελεί το δεύτερο βήμα του αλγόριθμου BLAST για μία ακολουθία που δίνεται ως παράμετρος. Στο βήμα αυτό (Ενότητα 4.1.2.2) χρησιμοποιείται το αυτόματο για την εύρεση των επιτυχιών. Με την κατασκευή του αυτόματου που περιγράψαμε, για τη σάρωση της ακολουθίας εισόδου διαβάζονται τα σύμβολα με τη σειρά και ακολουθούνται οι κατάλληλες μεταβάσεις. Όταν η μετάβαση που χρησιμοποιούμε είναι μετάβαση αποδοχής, έχουμε επιτυχία. Μόλις εντοπιστεί μία επιτυχία, υπολογίζεται η τιμή της διαγωνίου της. Κατόπιν, ελέγχεται μία λίστα στην οποία αποθηκεύονται οι διαγώνιοι ως αντικείμενα της κλάσης *Diagonal* (Ενότητα 5.2.8), έτσι ώστε να βρούμε αν έχει εντοπιστεί επιτυχία για αυτή τη διαγώνιο και αν ναι, ποιο είναι το τελικό σημείο για τη διαγώνιο αυτή. Αν δεν υπάρχουν επιτυχίες για τη διαγώνιο ή αν υπάρχουν αλλά το τελικό σημείο είναι μικρότερο του σημείου έναρξης της νέας επιτυχίας, δημιουργούμε ένα αντικείμενο της κλάσης *HSP* (Ενότητα 5.2.5) για το οποίο στη συνέχεια καλούμε τη μέθοδο επέκτασης. Αν μετά την επέκταση η βαθμολόγηση του ζεύγους τμημάτων είναι μεγαλύτερη της παραμέτρου S_2, το αποθηκεύουμε σε μια λίστα. Στο τέλος της σάρωσης της ακολουθίας, αν η δομή που αποθηκεύει τα ζεύγη τμημάτων υψηλής βαθμολόγησης δεν είναι κενή, αποθηκεύουμε στην λίστα των ακολουθιών που παρουσιάζουν ομοιότητες ένα αντικείμενο της κλάσης *MatchingSequence*, χρησιμοποιώντας την τρέχουσα ακολουθία και τη λίστα των ζευγών τμημάτων. Η λίστα των ακολουθιών που παρουσιάζουν ομοιότητες ταξινομείται με βάση το άθροισμα των βαθμολογήσεων για τα ζεύγη τμημάτων που προέκυψαν από κάθε ακολουθία. Παράλληλα, στη μέθοδο αυτή υπολογίζεται το συνολικό μέγεθος της βάσης δεδομένων, αθροίζοντας τα μήκη των ακολουθιών που σαρώνονται.

Εκτός από τη μέθοδο σάρωσης μιας ακολουθίας, η κλάση *DFA* έχει διαθέσιμη και μία μέθοδο που παρουσιάζει τα αποτελέσματα. Η μέθοδος αυτή εμφανίζει κατά ζεύγη (Ενότητα 4.4.2) τα ζεύγη τμημάτων υψηλής βαθμολόγησης ανά ακολουθία. Για κάθε ζεύγος τμημάτων υπολογίζεται η τιμή E_{DB} από τη σχέση (4.3) και παρουσιάζονται μόνο εκείνα για τα οποία είναι $E_{DB} \leq E$.

Τέλος, υπάρχει διαθέσιμη και μία μέθοδος που εμφανίζει της παραμέτρους του αυτόματου. Η μέθοδος αυτή μπορεί να συνδυαστεί με τη μέθοδο εμφάνισης των αποτελεσμάτων και ένα παράδειγμα εμφάνισης της εξόδου φαίνεται στο Σχήμα 5.9.

```
DFA for BLASTp using the 250-residue sequence:
MPSIDINCDMGESFGPWVMGQDTELMPHITAANIACGFHAGDPDVMLATVRAAIAADVAIGAHPGLPDLQGFGRRVMAVTPDEVYALTVYQVGALQAI
ARSQGGRLHHVKTHGALYTLTARDPALADAVARAVADVDPALPIYVANAAIAQATRERGLRAVYEVYADRSYQDDGTLTPRSQPHAMIEDVDQAIAQV
KRMVKEGVVRALSGKDVPITADTLCIHGDQPGAALFARRIRAALEAEGIEIRTV
Word length: 4
Threshold: 17
Scoring matrix used: PAM120
Number of states: 6487
Expected number of HSPs used: 1.0

Sequence name:gi|33516973|sp|O30444|Y150_BORPE Hypothetical UPF0271 protein BP0150
Total score: 800
Identities:  89  (62.67605633802817%),  Positives:  38  (26.760563380281692%),  Mismatches:  15
(10.56338028169014%) Score: 479 Expect: 0.0
    2    PSIDINCDMGESFGPWVMGQDTELMPHITAANIACGFHAGDPDVMLATVR
         P+ID+NCDMGES+G+W MG+D ++++ +T+ANIACGFH+GDP++M  TV
    3    PAIDLNCDMGESYGAWRMGNDEAVLQFVTSANIACGFHGGDPSTMRQTVA

   52    AAIAADVAIGAHPGLPDLQGFGRRVMAVTPDEVYALTVYQVGALQAIARS
         AA+A +VA+GAHP+LPDL GFGRR+M +TP+E+Y+L+VYQVGAL ++A S
   53    AALAHGVALGAHPSLPDLAGFGRRAMQITPQEAYDLVVYQVGALAGVAAS

  102    QGGRLHHVKTHGALYTLTARDPALADAVARAVADVDPALPIY
         QG+RLHHVK+HGALY+++A+D+ALA A+ +AV DVD++L +Y
  103    QGARLHHVKAHGALYNMAAKDAALARAICQAVRDVDSDLVLY

Identities:  65  (60.74766355140187%),  Positives:  27  (25.233644859813083%),  Mismatches:  15
(14.018691588785046%) Score: 321 Expect: 0.0
  144    VANAAIAQATRERGLRAVYEVYADRSYQDDGTLTPRSQPHAMIEDVDQAI
         +A++A+ +A+R+ GLRA+ EV+ADR+YQ+DG LTPRSQP+AMI D+DQAI
  146    LAGSALIDAARAIGLRAAQEVFADRTYQADGQLTPRSQPDAMITDLDQAI

  194    AQVKRMVKEGVVRALSGKDVPITADTLCIHGDQPGAALFARRIRAALEAE
         AQV  MV++G VR+ +G+ V++ ADTLCIHGDQP+A +FAR IR ALE +
  196    AQVLGMVRDGSVRTPDGQTVALQADTLCIHGDQPDALVFARGIRLALERD

  244    GIEIRTV
         GI+I+++
  246    GIAIQAA

------------------------------------------------------------
Sequence name:gi|33517100|sp|Q9I626|Y492_PSEAE Hypothetical UPF0271 protein PA0492
Total score: 573
Identities: 12 (50.0%), Positives: 10 (41.666666666666664%), Mismatches: 2 (8.333333333333334%)
Score: 84 Expect: 0.0014138875222402816
    1    MPSIDINCDMGESFGPWVMGQDTE
         M  +D+N+DMGE+FGPW++G++++
    1    MTEVDLNSDMGEGFGPWTIGDGVD

Identities: 62 (50.0%), Positives: 27 (21.774193548387096%), Mismatches: 35 (28.225806451612904%)
Score: 257 Expect: 0.0
   20    GQDTELMPHITAANIACGFHAGDPDVMLATVRAAIAADVAIGAHPGLPDL
         G D+++MP I++ANIA GFHAGDP+ M  TV  A + +VAIGAHPG+ DL
   22    GVDAAIMPLISSANIATGFHAGDPSSMRRTVEMAAEHGVAIGAHPGFRDL

   70    QGFGRRVMAVTPDEVYALTVYQVGALQAIARSQGGRLHHVKTHGALYTLT
          GFGRR +A+ + E+    +YQ+GAL+++AR QG  L+HVK HGALY
   72    VGFGRRHIAAPAIELVNDMLYQLGALREFARLQGLSLQHVKPHGALYMHL

  120    ARDPALADAVARAVADVDPALPIY
         ARD + A   + ++  ++P+L +Y
  122    ARDEVAARLFVETLQRLEPELLLY

Identities:  26  (41.935483870967744%),  Positives:  19  (30.64516129032258%),  Mismatches:  17
(27.419354838709676%) Score: 112 Expect: 1.1839713384009797E-7
  188    DVDQAIAQVKRMVKEGVVRALSGKDVPITADTLCIHGDQPGAALFARRIR
         D +Q+ ++V R  +EG VR++ G D+ I+ D++CIH+D PGA  ++  +R
  189    DPQQVADKVLRACREGKVRTVEGEDLDIAFDSVCIHSDTPGALELVASTR

  238    AALEAEGIEIRT
         A LE++GI I++
  239    ARLEGAGIRIKA
```

Σχήμα 5.9: Παράδειγμα εμφάνισης των αποτελεσμάτων για ακολουθία πρωτεΐνης (επιλογή).

5.3. Βάσεις βιολογικών δεδομένων

5.3.1. Μορφή και διαθεσιμότητα

Όπως αναφέραμε και στην Ενότητα 2.2 υπάρχει ένας μεγάλος αριθμός βάσεων δεδομένων. Οι κυριότερες από αυτές είναι δημόσια διαθέσιμες μέσω του διαδικτύου στην τοποθεσία ftp του NCBI. Θα χρησιμοποιήσουμε λοιπόν αυτή την τοποθεσία, καθώς εκεί βρίσκονται συγκεντρωμένα αντίγραφα όλων των βάσεων που χρησιμοποιούμε.
Οι βάσεις δεδομένων βρίσκονται στην πλήρη τους μορφή στην τοποθεσία:

ftp://ftp.ncbi.nlm.nih.gov/blast/db/

Λόγω του μεγάλου τους μεγέθους, οι βάσεις δεδομένων είναι σε συμπιεσμένη μορφή και κάποιες από αυτές είναι χωρισμένες σε τμήματα.
Υπάρχουν όμως διαθέσιμα και αντίγραφα των βάσεων στη μορφή FASTA. Η τοποθεσία που βρίσκονται τα αντίγραφα αυτά είναι:

ftp://ftp.ncbi.nlm.nih.gov/blast/db/FASTA/

Και πάλι το μέγεθος των βάσεων δεδομένων είναι μεγάλο (αν και αρκετά μικρότερο από αυτό για την πλήρη μορφή τους), γι' αυτό είναι σε συμπιεσμένη μορφή.
Όπως είδαμε και στην Ενότητα 5.2.6. η πληροφορίες που αποθηκεύουμε για κάθε ακολουθία περιορίζονται στο όνομά της και την ίδια την ακολουθία σε μορφή συμβολοσειράς. Αυτές είναι ακριβώς οι πληροφορίες που περιλαμβάνει και η μορφή FASTA μιας βάσης δεδομένων. Έτσι λοιπόν μπορούμε να χρησιμοποιήσουμε τη μορφή FASTA των βάσεων δεδομένων που μας ενδιαφέρουν. Με αυτό τον τρόπο επιταχύνεται η σάρωση της επιλεγμένης βάσης δεδομένων, ειδικά στην περίπτωση που η σάρωση γίνεται μέσω δικτύου για ένα απομακρυσμένο αντίγραφο.

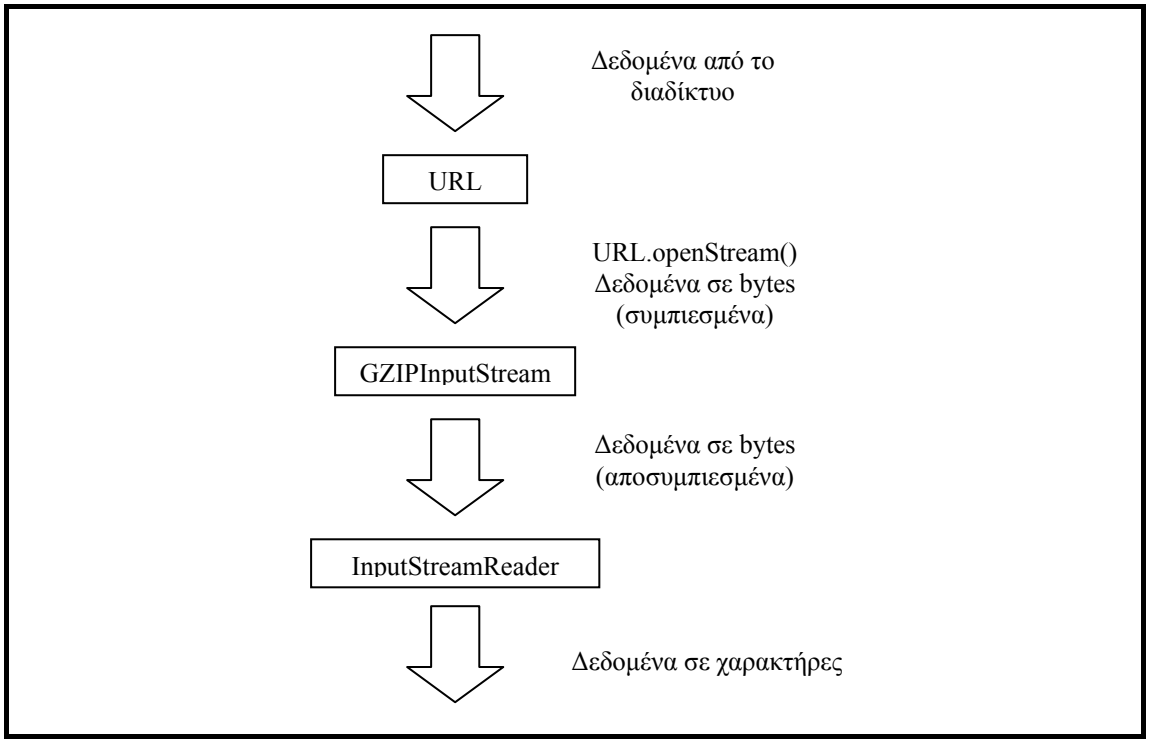
5.3.2. Σάρωση της βάσης δεδομένων: Η κλάση DBReader

Για τη σάρωση της βάσης δεδομένων, διακρίνουμε δύο περιπτώσεις:

- Σάρωση τοπικού αντίγραφου
- Σάρωση μέσω (δια)δικτύου

Και στις δύο περιπτώσεις θέτουμε ως προϋπόθεση να βρίσκεται η βάση δεδομένων αποθηκευμένη σε μορφή FASTA.
Στην πρώτη περίπτωση, η διαδικασία είναι απλή: η βάση δεδομένων είναι αποθηκευμένη σε ένα αρχείο απ' όπου μπορούμε εύκολα να διαβάσουμε τις πληροφορίες που μας ενδιαφέρουν. Για την ανάγνωση από το αρχείο χρησιμοποιούμε την κλάση *FileReader* της Java [18]. Στη συνέχεια χρησιμοποιείται η κλάση *BufferedReader*, έτσι ώστε να μορφοποιηθούν τα δεδομένα κατά γραμμή. Κάθε γραμμή που αντιπροσωπεύει αναγνωριστικό κωδικό και όνομα ακολουθίας στη μορφή FASTA ξεκινάει με τον χαρακτήρα '>', επομένως μπορούμε να βρούμε εύκολα τα ονόματα των ακολουθιών για

την κλάση *Sequence*. Μετά από κάθε σειρά που αντιπροσωπεύει όνομα ακολουθούν μία ή περισσότερες (συνήθως) σειρές που αντιπροσωπεύουν την ακολουθία. Αυτές οι σειρές συνενώνονται για να σχηματίσουν την ακολουθία σαν μια ενιαία συμβολοσειρά. Όταν βρεθεί ξανά σειρά που αντιπροσωπεύει όνομα έχουμε νέα ακολουθία. Για κάθε ακολουθία που εντοπίζεται με αυτό τον τρόπο χρησιμοποιούμε τη μέθοδο σάρωσης της κλάσης *DFA*.
Στη δεύτερη περίπτωση θέτουμε ως πρόσθετη προϋπόθεση το αρχείο που περιέχει τα δεδομένα να και είναι συμπιεσμένο σε μορφή GZip (είναι η μορφή στην οποία βρίσκονται οι βάσεις δεδομένων στις τοποθεσίες που αναφέραμε στην Ενότητα 5.3.1), έτσι ώστε να περιορίσουμε τον όγκο των δεδομένων που μετακινούνται μέσω δικτύου. Σε αυτή την περίπτωση, χρησιμοποιούμε μια σειρά από έτοιμες κλάσεις και ροές της Java έτσι ώστε να έχουμε πρόσβαση στα δεδομένα μέσω του (δια)δικτύου, να γίνει η αποσυμπίεση σε πραγματικό χρόνο και να πάρουμε τις ακολουθίες. Η διαδικασία αυτή γίνεται χρησιμοποιώντας της κλάσεις *URL*, *GZIPInputStream*, *InputStreamReader* [18], με τέτοιο τρόπο ώστε η κάθε ροή να δίνεται ως είσοδος στην επόμενη. Στο Σχήμα 5.10 παρουσιάζεται η σειρά με την οποία χρησιμοποιούνται οι ροές αυτές. Στη συνέχεια ακολουθείται η ίδια διαδικασία με την περίπτωση του τοπικού αντίγραφου για τον εντοπισμό των ακολουθιών.



Σχήμα 5.10: Η σειρά που χρησιμοποιούνται οι ροές για ανάγνωση από απομακρυσμένο αντίγραφο της βάσης δεδομένων.

Για τη δημιουργία της κατάλληλης ροής για τα δεδομένα, ανάλογα με την περίπτωση που εξετάζουμε (τοπικό ή απομακρυσμένο αντίγραφο), δημιουργούμε μια κλάση η οποία ονομάζεται *DBReader*. Η κλάση αυτή διαθέτει μόνο μία στατική μέθοδο που μας επιστρέφει την κατάλληλη ροή,

όπως την περιγράψαμε παραπάνω, ανάλογα με τον τύπο του αντίγραφου που επιθυμούμε να χρησιμοποιήσουμε.

5.4. Ανακεφαλαίωση

Στο κεφάλαιο αυτό παρουσιάσαμε αναλυτικά όλες τις δομές που υλοποιήσαμε για το σύστημα που σχεδιάσαμε. Για κάθε μία από αυτές της δομές, περιγράψαμε τις παραμέτρους που χρησιμοποιεί, τη δομή και τη λειτουργία της. Εκείνο που απομένει πλέον είναι να περιγράψουμε τον τρόπο με τον οποίο οι δομές αυτές χρησιμοποιούνται στην πράξη, από ένα ολοκληρωμένο σύστημα. Το θέμα αυτό θα μας απασχολήσει στο Κεφάλαιο 6, στο οποίο παρουσιάζουμε ένα ολοκληρωμένο σύστημα αλληλεπίδρασης με το χρήστη που σχεδιάσαμε, που χρησιμοποιεί τις δομές αυτές για την αναζήτηση σε μιας βάσης βιολογικών δεδομένων με τον αλγόριθμο BLAST.

ΚΕΦΑΛΑΙΟ 6

Εφαρμογή: Το σύστημα JavaBLAST

Στο κεφάλαιο αυτό περιγράφουμε αναλυτικά την εφαρμογή που σχεδιάσαμε με βάση την υλοποίηση που περιγράψαμε στο Κεφάλαιο 5. Η εφαρμογή αυτή χρησιμοποιεί τις δομές της υλοποίησης για την αναζήτηση ακολουθιών με ομοιότητες ως προς μία ακολουθία εισόδου. Για να γίνει αυτό απαιτείται η επιλογή από το χρήστη των κατάλληλων παραμέτρων. Στη συνέχεια θα περιγράψουμε αναλυτικά το πως γίνεται η επιλογή αυτή.

6.1. Γενικά

Το σύστημα JavaBLAST αποτελεί μία γραφική εφαρμογή σε Java, η οποία δέχεται σαν είσοδο μία ακολουθία και τις απαραίτητες παραμέτρους για τον αλγόριθμο BLAST και δίνει ως έξοδο τα αποτελέσματα της σάρωσης μιας επιλεγμένης βάσης βιολογικών δεδομένων. Η επιλογή της Java ως γλώσσας προγραμματισμού ενισχύθηκε εδώ από την ευκολία που παρέχει στο σχεδιασμό γραφικών εφαρμογών. Τα βασικά μέρη της εφαρμογής είναι τρία:

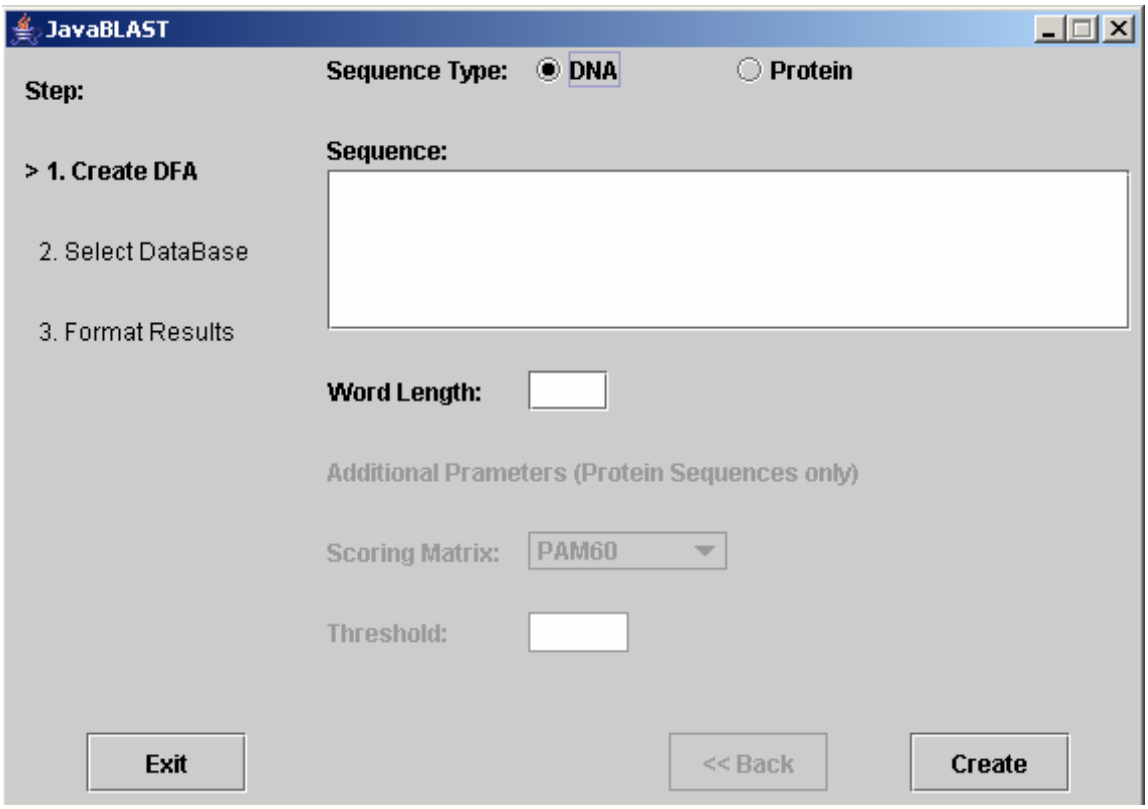
- Στο πρώτο μέρος γίνεται η εισαγωγή της ακολουθίας και των παραμέτρων που θα χρησιμοποιηθούν για το αυτόματο (Ενότητα 5.2.9) και δημιουργείται το αυτόματο.
- Στο δεύτερο μέρος επιλέγεται η βάση βιολογικών δεδομένων που θα χρησιμοποιήσουμε, καθώς επίσης και αν πρόκειται για τοπικό αντίγραφο ή αν η σάρωση θα γίνει για απομακρυσμένο αντίγραφο, χρησιμοποιώντας την τοποθεσία που αναφέραμε στην Ενότητα 5.3.1 και γίνεται η σάρωση.
- Στο τρίτο μέρος ορίζεται το κριτήριο στατιστικής σημασίας E και εμφανίζονται τα αποτελέσματα που το ικανοποιούν στην οθόνη, ενώ παράλληλα αποθηκεύονται και σε ένα αρχείο που επιλέγει ο χρήστης. Επίσης, υπάρχει η δυνατότητα να περιοριστεί ο αριθμός των αποτελεσμάτων που εμφανίζονται, δίνοντας ένα όριο (για παράδειγμα οι δέκα ακολουθίες με τις μεγαλύτερες βαθμολογήσεις).

6.2. Περιγραφή του συστήματος JavaBLAST

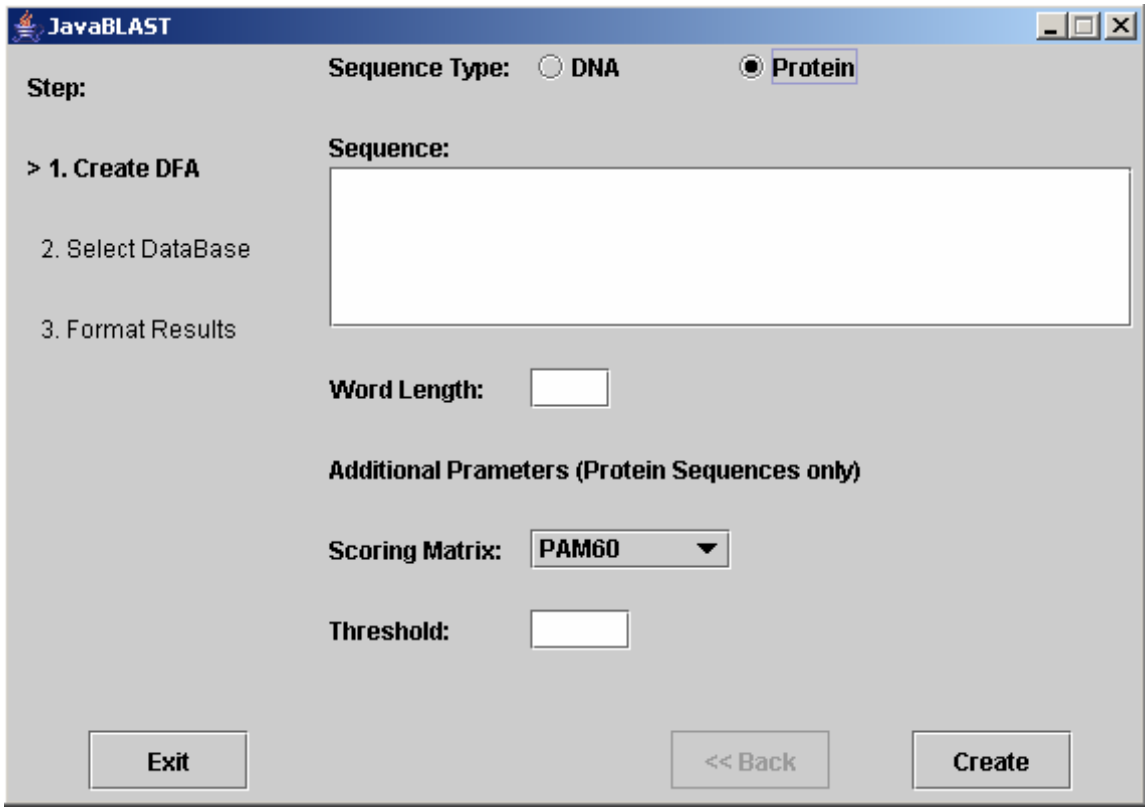
Στην ενότητα αυτή θα παρουσιάσουμε αναλυτικά τα τρία μέρη της εφαρμογής JavaBLAST, περιγράφοντας αναλυτικά το γραφικό περιβάλλον και τις ενέργειες που πρέπει να κάνει ο χρήστης. Για κάθε μέρος επίσης θα περιγράψουμε και τα σφάλματα που μπορεί να προκύψουν από λανθασμένες επιλογές του χρήστη στην εισαγωγή των δεδομένων.

6.2.1. Δημιουργία αυτόματου

Το πρώτο μέρος της εφαρμογής JavaBLAST φαίνεται στο Σχήμα 6.1. Στο αριστερό μέρος φαίνεται μία λίστα με τα τρία μέρη της εφαρμογής. Η λίστα αυτή υπάρχει σε κάθε μέρος της και κάθε φορά το τρέχον μέρος σημειώνεται με έντονους χαρακτήρες και το χαρακτήρα '>' πριν από την ονομασία του.
Στο επάνω μέρος φαίνονται τα πλήκτρα επιλογής του τύπου της ακολουθίας (*Sequence Type*), απ' όπου ο χρήστης επιλέγει τον επιθυμητό τύπο. Ακολουθεί το πεδίο εισαγωγής της ακολουθίας με τη μορφή συμβολοσειράς (*Sequence*). Η ακολουθία θα πρέπει να αποτελείται από σύμβολα μόνο του κατάλληλου αλφάβητου (Ενότητα 2.1.1). Στη συνέχεια, έχουμε το πεδίο για την εισαγωγή του μήκους λέξεων w (*Word Length*). Μπορεί να εισαχθεί από τον χρήστη οποιοσδήποτε ακέραιος αριθμός μεγαλύτερος του μηδενός. Ακολουθούν οι πρόσθετες παράμετροι για τις πρωτεΐνες: ο ορισμός του πίνακα βαθμολόγησης (*Scoring Matrix*) όπου ο χρήστης καλείται να επιλέξει από τους πίνακες βαθμολόγησης που έχουν υλοποιηθεί (Ενότητα 5.2.2) και το πεδίο για την βαθμολόγηση κατωφλίου T (*Threshold*), η οποία μπορεί να είναι οποιοσδήποτε ακέραιος. Οι επιλογές αυτές ενεργοποιούνται μόνο όταν στον τύπο της ακολουθίας είναι επιλεγμένος αυτός των πρωτεϊνών, όπως φαίνεται στο Σχήμα 6.2.



Σχήμα 6.1: Το πρώτο μέρος της εφαρμογής JavaBLAST.

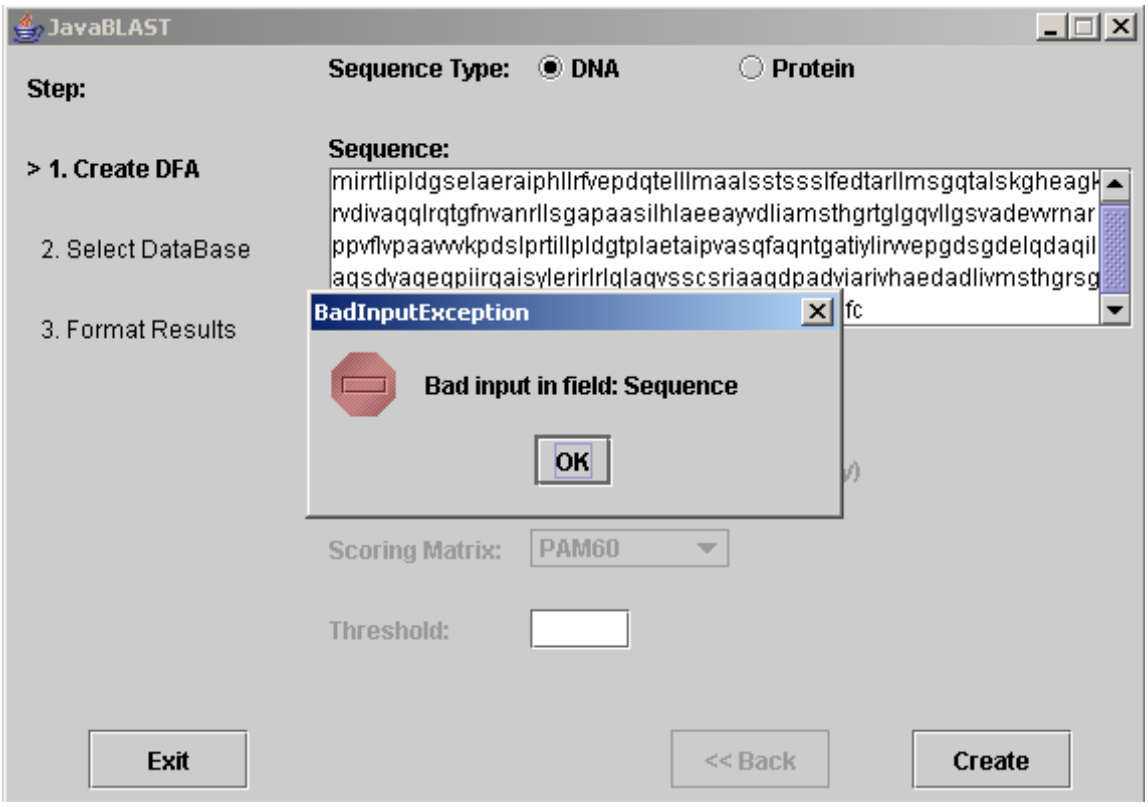


Σχήμα 6.2: Το πρώτο μέρος της εφαρμογής με ενεργοποιημένες τις πρόσθετες παραμέτρους.

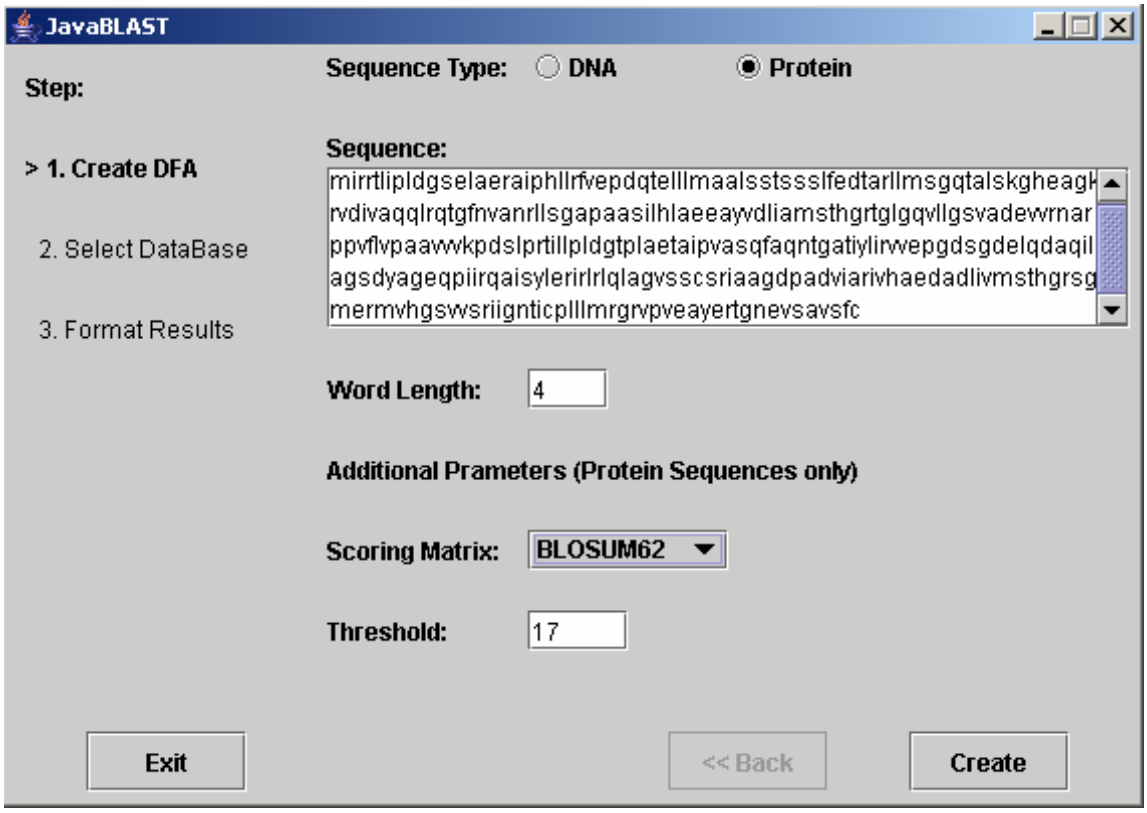
Στο κάτω μέρος διακρίνονται τα πλήκτρα εξόδου από την εφαρμογή (*Exit*), επιστροφής στο προηγούμενο μέρος *(<< Back*) το οποίο για το πρώτο μέρος είναι ανενεργό και αυτό της δημιουργίας του αυτόματου (*Create*).
Με το πάτημα του πλήκτρου δημιουργίας του αυτόματου, αν τα δεδομένα που έχει εισάγει ο χρήστης είναι έγκυρα, δημιουργείται το αυτόματο. Άκυρα δεδομένα θεωρούνται τα παρακάτω:

- Κενή ακολουθία ή ακολουθία που περιλαμβάνει χαρακτήρες που δεν ανήκουν στο αντίστοιχο αλφάβητο.
- Κενό, δεκαδικό, αρνητικό ή μηδενικό μήκος λέξεων.
- Κενή ή δεκαδική βαθμολόγηση κατωφλίου.

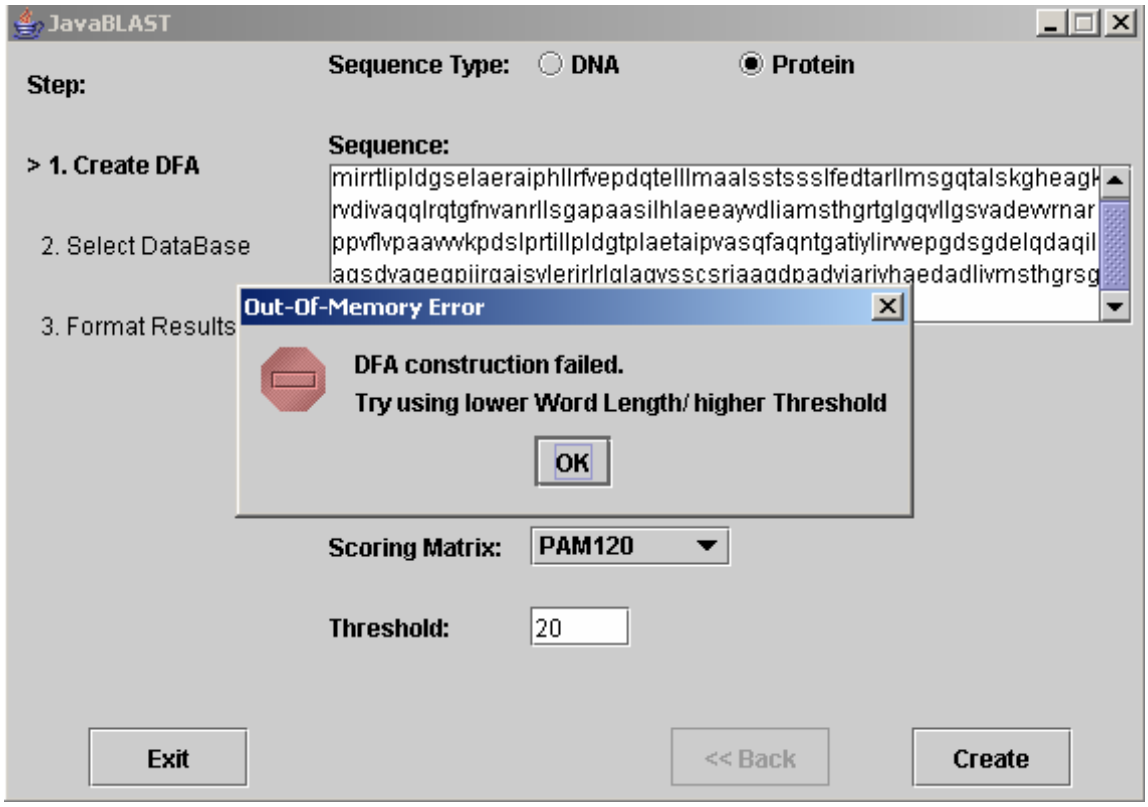
Σε κάθε περίπτωση εισαγωγής άκυρων δεδομένων εμφανίζεται ένα μήνυμα σφάλματος όπως φαίνεται στο Σχήμα 6.3. Στο Σχήμα 6.4 παρουσιάζεται ένα παράδειγμα έγκυρης εισαγωγής δεδομένων, για μια ακολουθία πρωτεΐνης.
Μια άλλη περίπτωση σφάλματος που μπορεί να προκύψει κατά την κατασκευή του αυτόματου είναι η εξάντληση της διαθέσιμης μνήμης. Αυτό μπορεί να συμβεί είτε αν ο χρήστης εισάγει πολύ μεγάλη τιμή για το μήκος λέξεων, είτε αν εισάγει πολύ χαμηλή τιμή για την βαθμολόγηση κατωφλίου. Στην περίπτωση αυτή εμφανίζεται ένα μήνυμα που προτρέπει τον χρήστη σε αλλαγή αυτών των παραμέτρων, όπως αυτό του Σχήματος 6.5.



Σχήμα 6.3: Παράδειγμα μηνύματος σφάλματος. Η ακολουθία που εισήχθηκε δεν είναι έγκυρη ακολουθία DNA.



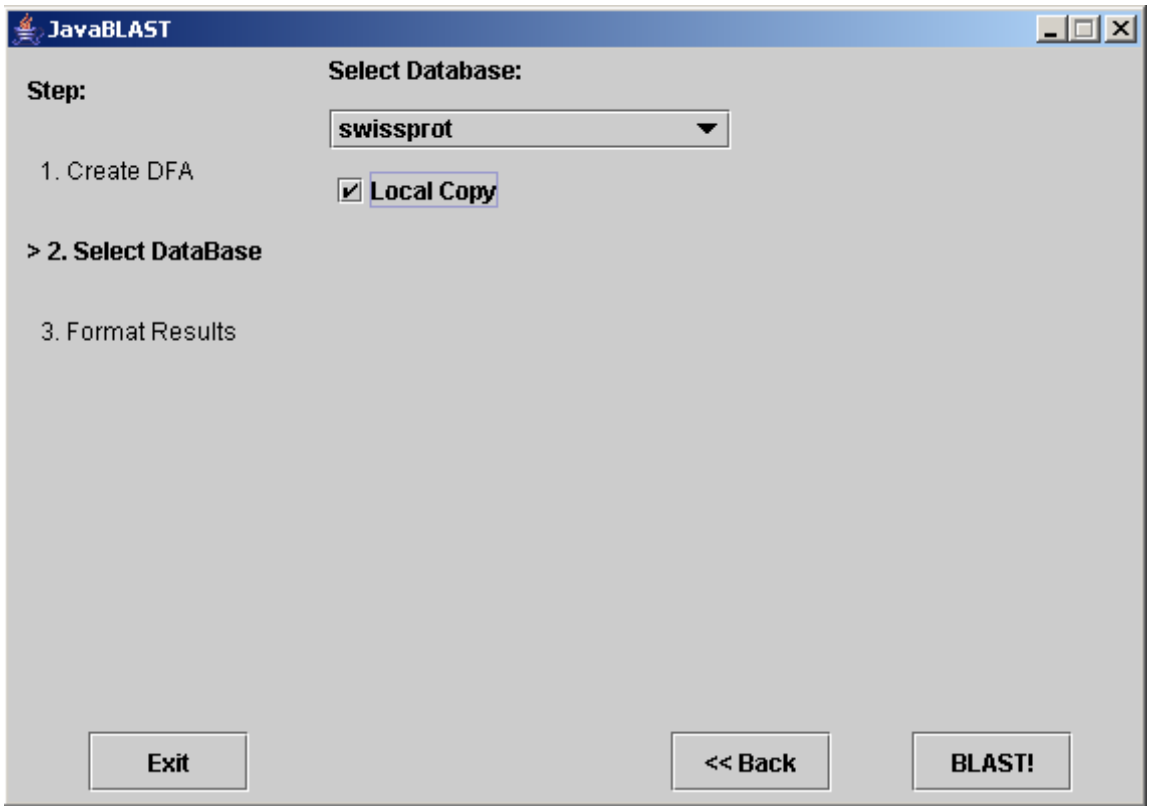
Σχήμα 6.4: Παράδειγμα έγκυρης συμπλήρωσης των πεδίων του πρώτου μέρους.



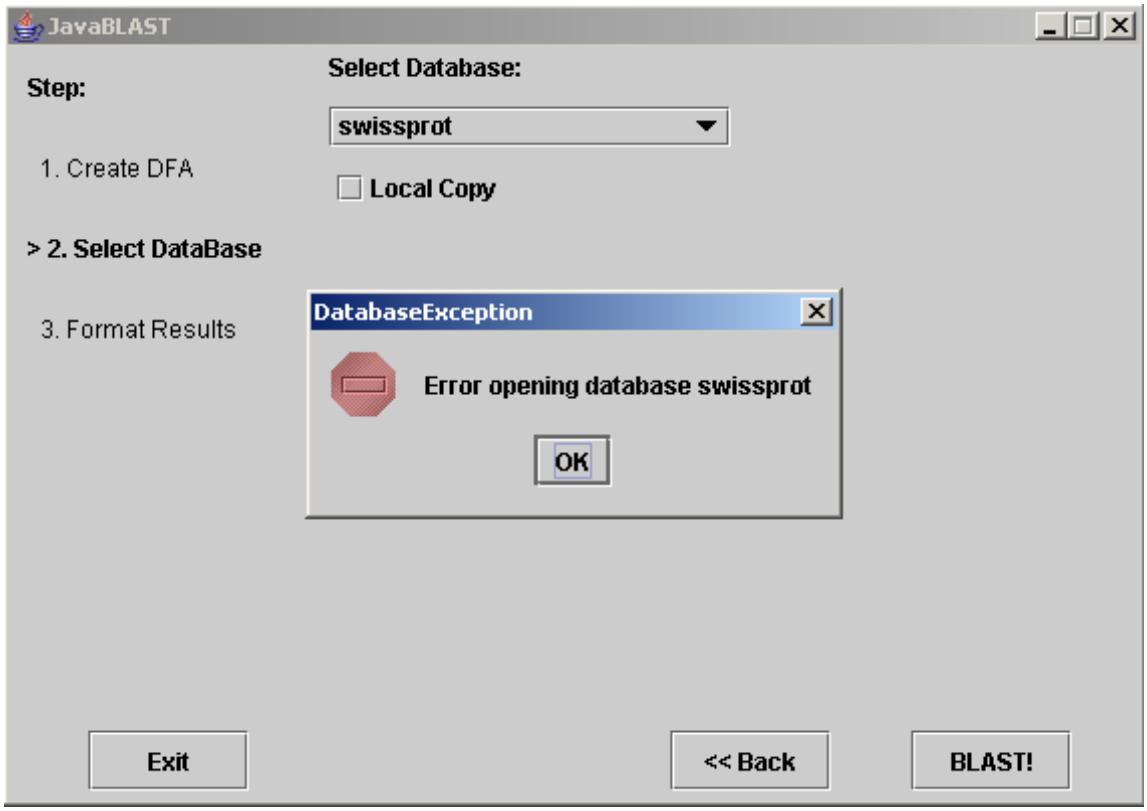
Σχήμα 6.5: Μήνυμα σφάλματος σε περίπτωση εξάντλησης της διαθέσιμης μνήμης.

6.2.2. Επιλογή βάσης δεδομένων

Αφού κατασκευαστεί το αυτόματο, προχωράμε στο δεύτερο μέρος της εφαρμογής που φαίνεται στο Σχήμα 6.6 (για ακολουθία πρωτεΐνης, ενώ ανάλογο είναι και για ακολουθία DNA). Στο μέρος αυτό ο χρήστης επιλέγει την βάση βιολογικών δεδομένων για την οποία επιθυμεί να εκτελεστεί ο αλγόριθμος BLAST. Επιλέγει επίσης αν επιθυμεί να εκτελεστεί ο αλγόριθμος για τοπικό αντίγραφο ή για αντίγραφο που βρίσκεται στην τοποθεσία ftp του NCBI (Ενότητα 5.3.1). Στο κάτω μέρος διακρίνεται το πλήκτρο επιστροφής στο πρώτο βήμα *(<< Back*) το οποίο είναι πλέον ενεργό, καθώς και το πλήκτρο *BLAST!* το οποίο ξεκινά την εκτέλεση του αλγόριθμου. Στο μέρος αυτό εμφανίζονται μηνύματα σφάλματος στην περίπτωση που αποτύχει το άνοιγμα του αρχείου στο οποίο αποθηκεύεται η βάση δεδομένων ή αν προκύψει σφάλμα κατά την ανάγνωση από αυτό (Σχήμα 6.7). Τα σφάλματα στο άνοιγμα του αρχείου μπορεί να προέρχονται είτε από την αδυναμία εντοπισμού του αρχείου είτε, στην περίπτωση της σάρωσης μέσω δικτύου, από την αδυναμία σύνδεσης με τον εξυπηρετητή. Τα σφάλματα κατά την ανάγνωση μπορεί να οφείλονται σε σφάλματα στο αρχείο, ή, στην περίπτωση του απομακρυσμένου αντίγραφου, σε διακοπή της σύνδεσης με τον εξυπηρετητή.



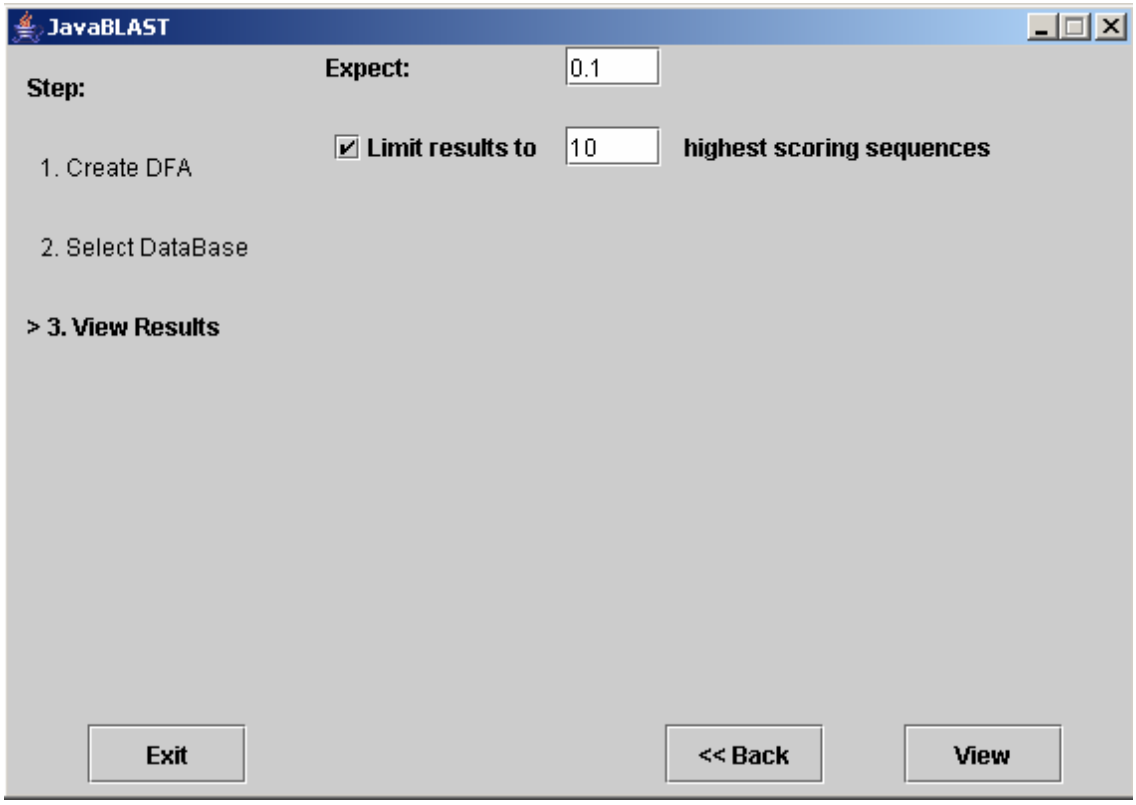
Σχήμα 6.6: Το δεύτερο μέρος της εφαρμογής (ακολουθία πρωτεΐνης).



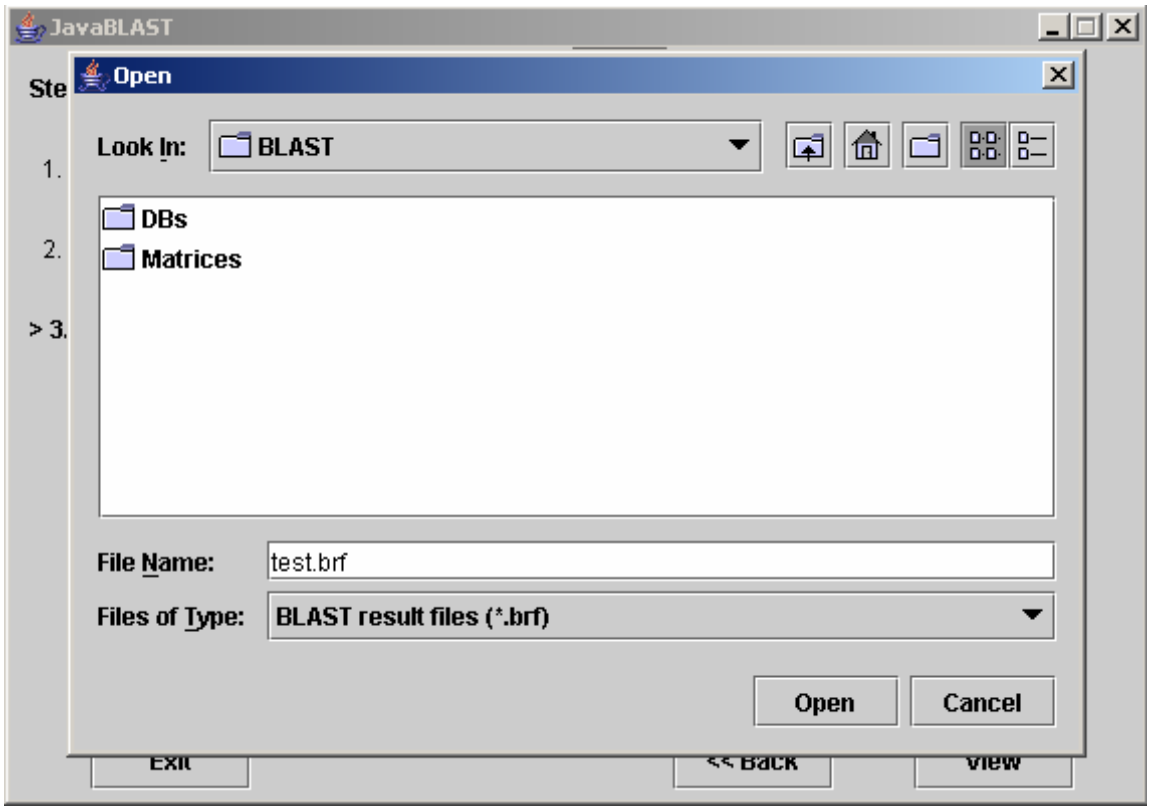
Σχήμα 6.7: Μήνυμα σφάλματος κατά το άνοιγμα του αρχείου.

6.2.3. Εμφάνιση των αποτελεσμάτων

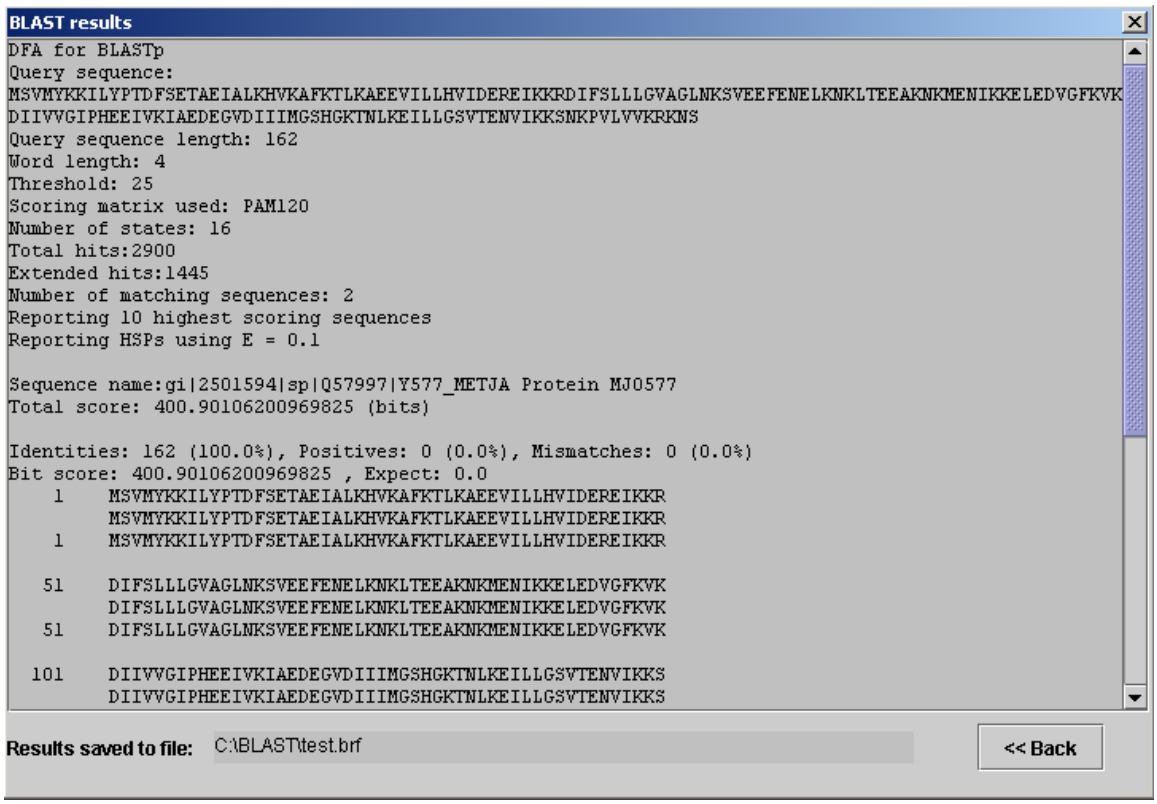
Στο τρίτο μέρος της εφαρμογής και αφού έχει ολοκληρωθεί η σάρωση της βάσης δεδομένων, ο χρήστης καλείται να εισάγει τις παραμέτρους για την εμφάνιση των αποτελεσμάτων. Το τρίτο μέρος της εφαρμογής φαίνεται στο Σχήμα 6.8. Ο χρήστης επιλέγει την τιμή για το κριτήριο στατιστικής σημασίας και τον αριθμό των ακολουθιών που επιθυμεί να παρουσιαστούν ως αποτελέσματα. Το κριτήριο στατιστικής σημασίας *E* είναι ένας πραγματικός αριθμός αυστηρά θετικός. Έτσι αν η τιμή που εισάγεται είναι μικρότερη ή ίση του μηδενός εμφανίζεται ένα μήνυμα σφάλματος παρόμοιο με αυτά του πρώτου μέρους (Ενότητα 6.2.1, Σχήμα 6.3). Στη συνέχεια, ο χρήστης μπορεί να επιλέξει να περιορίσει τα αποτελέσματα που παρουσιάζονται, δίνοντας τον επιθυμητό αριθμό ακολουθιών που θα εμφανιστούν. Ο αριθμός αυτός θα πρέπει να είναι ακέραιος και μεγαλύτερος του μηδενός. Σε αντίθετη περίπτωση εμφανίζεται μήνυμα σφάλματος. Αν ο αριθμός που εισάγεται είναι μεγαλύτερος από τον αριθμό των ακολουθιών που εντοπίστηκαν από τον αλγόριθμο BLAST, τότε το όριο αυτό αγνοείται και εμφανίζονται όλες οι ακολουθίες. Στο κάτω μέρος διακρίνεται το πλήκτρο επιστροφής στο δεύτερο μέρος (<< *Back*), όπως και το πλήκτρο εμφάνισης των αποτελεσμάτων *View*.
Στα αποτελέσματα εμφανίζονται στην αρχή πληροφορίες για το αυτόματο, χρησιμοποιώντας την κατάλληλη μέθοδο της κλάσης *DFA*. Στη συνέχεια χρησιμοποιείται η μέθοδος εμφάνισης των αποτελεσμάτων της ίδιας κλάσης για να εμφανιστούν τα αποτελέσματα (Ενότητα 5.2.9). Με το πάτημα του πλήκτρου *View*, εμφανίζεται ένα νέο παράθυρο και ο χρήστης καλείται να επιλέξει το όνομα του αρχείου στο οποίο θα αποθηκευτούν τα αποτελέσματα (Σχήμα 6.9). Στη συνέχεια, εμφανίζονται σε ένα νέο παράθυρο τα αποτελέσματα με τη μορφή που αναφέραμε στην Ενότητα 5.2.9. Με την ίδια ακριβώς μορφή αποθηκεύονται τα αποτελέσματα και στο αρχείο. Στο παράθυρο αυτό, εμφανίζεται στο κάτω μέρος το όνομα του αρχείου στο οποίο αποθηκεύτηκαν τα αποτελέσματα, ενώ υπάρχει και ένα πλήκτρο επιστροφής στην εισαγωγή παραμέτρων για την εμφάνιση των αποτελεσμάτων (Σχήμα 6.10).



Σχήμα 6.8: Το τρίτο μέρος της εφαρμογής.



Σχήμα 6.9: Επιλογή αρχείου αποθήκευσης των αποτελεσμάτων.



Σχήμα 6.10: Εμφάνιση των αποτελεσμάτων του αλγόριθμου BLAST.

ΚΕΦΑΛΑΙΟ 7

Συμπεράσματα

Στο κεφάλαιο αυτό συνοψίζουμε τα συμπεράσματά μας επάνω στο θέμα που μελετήσαμε σε αυτή την εργασία. Στη συνέχεια, προτείνουμε κάποια σημεία επάνω στα οποία μπορεί να γίνει κάποια μελλοντική εργασία για την επέκταση των δυνατοτήτων του συστήματος.

7.1. Ανακεφαλαίωση

Σε αυτή την εργασία μελετήσαμε το πρόβλημα της αναζήτησης σε βάσεις βιολογικών δεδομένων για ακολουθίες που παρουσιάζουν μεγάλη ομοιότητα με κάποια εξεταζόμενη ακολουθία. Το πρόβλημα της σύγκρισης μιας ακολουθίας με πολλές άλλες για την εύρεση ομοιοτήτων αποτελεί ένα από τα προβλήματα της μοριακής βιολογίας που καλείται να επιλύσει η υπολογιστική βιολογία. Για το πρόβλημα λοιπόν αυτό, περιγράψαμε τους κυριότερους αλγόριθμους που έχουν προταθεί για τη σύγκριση ακολουθιών και επιλέξαμε να υλοποιήσουμε τον αλγόριθμο BLAST, ο οποίος είναι ο πιο αποδοτικός όταν εξετάζουμε την περίπτωση της αναζήτησης σε μια βάση δεδομένων. Υλοποιήσαμε στη συνέχεια ένα σύστημα σε Java, το οποίο περιλαμβάνει όλες τις απαραίτητες δομές και λειτουργίες για την εκτέλεση του αλγόριθμου, τόσο για βάσεις και ακολουθίες DNA όσο και για πρωτεΐνες. Τέλος με βάση το σύστημα αυτό σχεδιάσαμε μια εφαρμογή, στην οποία προσφέρεται η δυνατότητα στους χρήστες να εισάγουν μία ακολουθία, τις τιμές που επιθυμούν για τις παραμέτρους του αλγόριθμου, να επιλέξουν μία βάση δεδομένων, να εκτελέσουν τον αλγόριθμο και να παρατηρήσουν τα αποτελέσματα. Η χρησιμότητα ενός τέτοιου εργαλείου γίνεται περισσότερο εμφανής αν αναλογιστεί κανείς ότι μόνο το αντίστοιχο σύστημα του NCBI εξυπηρετεί περισσότερες από 10000 αιτήσεις καθημερινά.

7.2. Μελλοντική εργασία

Στην ενότητα αυτή θα παρουσιάσουμε κάποιες προτάσεις για μελλοντική εργασία επάνω στο σύστημα που υλοποιήσαμε, την οποία κατηγοριοποιούμε σε τρεις τομείς.

- Ο πρώτος τομέας είναι η βελτίωση του αλγορίθμου BLAST. Νεότερες εκδόσεις του BLAST που έχουν προταθεί τα τελευταία χρόνια, περιλαμβάνουν πρόσθετα χαρακτηριστικά που ο βασικός αλγόριθμος δε διαθέτει. Ένα από αυτά είναι η χρήση κενών στις αντιστοιχίσεις

που αναφέρονται ως αποτελέσματα. Με αυτό τον τρόπο, υπάρχει η δυνατότητα για ακόμα μεγαλύτερη βελτίωση των αποτελεσμάτων, με τη συνένωση επιτυχιών που ανήκουν στην ίδια ακολουθία.

- Ο δεύτερος τομέας είναι η επέκταση του συστήματος που σχεδιάσαμε. Μία βελτίωση που μπορεί εύκολα να γίνει είναι η προσθήκη επιλογών για εμφάνιση των αποτελεσμάτων με κάποια από της άλλες μορφές που αναφέραμε στην Ενότητα 4.4. Στην περίπτωση αυτή θα καλείται ο χρήστης να επιλέξει τη μορφοποίηση που επιθυμεί για τα αποτελέσματα. Μία άλλη επέκταση που μπορεί να γίνει είναι η υλοποίηση ενός υποσυστήματος που θα «μεταφράζει» τις ακολουθίες DNA σε πρωτεΐνες, σύμφωνα με τον γενετικό κώδικα που παρουσιάσαμε στον Πίνακα 1.3. Με αυτόν τον τρόπο, μία ακολουθία πρωτεΐνης θα μπορεί να συγκριθεί με ακολουθίες DNA, έτσι ώστε να βρεθούν οι πιθανές ακολουθίες DNA που μπορεί να ελέγχουν το σχηματισμό της πρωτεΐνης. Ένα τέτοιο εργαλείο έχει ιδιαίτερη χρησιμότητα για τους βιολόγους. Θα μπορούσε επίσης να γίνει μια προσαρμογή του συστήματος έτσι ώστε να μπορεί να σαρώνει τη βάση δεδομένων για πολλές ακολουθίες ταυτόχρονα. Στην περίπτωση αυτή, ο χρήστης θα δίνει ένα σύνολο από ακολουθίες τις οποίες επιθυμεί να συγκριθούν, ενώ θα έχει και τη δυνατότητα να ορίζει χωριστές παραμέτρους για κάθε μία ξεχωριστά. Μια άλλη χρήσιμη προσθήκη θα ήταν η ανάπτυξη ενός υποσυστήματος το οποίο, για κάθε ακολουθία που εμφανίζεται στα αποτελέσματα, θα ανακτά από τη βάση και θα εμφανίζει την πλήρη εγγραφή της ακολουθίας αυτής.

- Ο τρίτος τομέας είναι η χρήση του συστήματος για τη δημιουργία μεγαλύτερων και πιο πολύπλοκων συστημάτων. Για παράδειγμα, μπορεί να χρησιμοποιηθεί σαν βάση για τη δημιουργία ενός συστήματος ειδοποίησης που θα σαρώνει της βάσεις δεδομένων, ενημερώνοντας το χρήστη (ή τους χρήστες) όταν εισάγονται σε αυτές ακολουθίες που παρουσιάζουν ομοιότητα με αυτή (ή αυτές) που έχουν εισάγει. Επειδή όπως είδαμε στην Ενότητα 2.2 για τις κυριότερες βάσεις δεδομένων υπάρχουν διαθέσιμες ως χωριστές βάσεις οι πιο πρόσφατες προσθήκες και μεταβολές, ένα τέτοιο σύστημα δε θα απαιτείται να σαρώνει ολόκληρη τη βάση αλλά μόνο το μέρος με τις πρόσφατες ενημερώσεις.

7.3. Επίλογος

Ο βασικός σκοπός αυτής της εργασίας ήταν η παρουσίαση του προβλήματος της αναζήτησης σε βάσεις βιολογικών δεδομένων και η υλοποίηση του αποδοτικότερου αλγόριθμου για την επίλυσή του. Αυτό που πετύχαμε ήταν η ανάπτυξη ενός πλήρους συστήματος, με το οποίο ο χρήστης μπορεί να εκτελέσει την αναζήτηση για μια επιλεγμένη ακολουθία σε μια επιλεγμένη βάση, χρησιμοποιώντας τον αλγόριθμο BLAST. Ελπίζουμε ότι αυτή η εργασία θα αποτελέσει μια καλή αρχή και ένα χρήσιμο βοήθημα για μελλοντική εργασία επάνω στο ίδιο πρόβλημα.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, David Lipman: Basic Local Alignment Search Tool, *J. Mol. Biol.* (1990) 251**,** 403-410.

[2] Dan Gusfield: *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*, Cambridge University Press, 1997.

[3] João Setubal, João Meidanis: *Introduction to Computational Molecular Biology*, PWS Publishing Company, 1997.

[4] Cynthia Gibas, Per Jambeck: *Developing Bioinformatics Computer Skills*, O' Reilly, 2001.

[5] Claus-Dieter Paul: *Biologie, für Schule und Beruf*, Verlag Europa Lehrmittel Nourney, 1993. Για την ελληνική έκδοση, μετάφραση Κάρολος Βούλγαρης, *Βιολογία: Τεχνολογία και Περιβάλλον*, Ευρωπαϊκές Τεχνικές Εκδόσεις, 1997.

[6] K. Sivarama Sastry, G. Padmanaban, C. Subramanyam: *Textbook of Molecular Biology*, Macmillan India Limited, 1994.

[7] Samuel Karlin, Stephen Altschul: Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes, *Proc. Natl. Acad. Sci. USA*, vol. 87, pp 2264-2268, March 1990, Evolution.

[8] R. Schwarz, M. Dayhoff: Matrices for detecting distant relationships, *Atlas of Protein Sequences*, p. 353-358, Natl. Biomed. Res. Found., 1979.

[9] S. Henikoff, J. G. Henikoff: Amino acid substitution matrices from protein blocks, *Proc. Natl. Acad. Sci.*, 89:10, 915-919, 1992.

[10] N. Williams: Europe opens institute to deal with gene data deluge, *Science*, 269:630, 1995.

[11] Τοποθεσία ftp του National Center for Biotechnology Information (NCBI): ftp://ftp.ncbi.nih.gov/blast/matrices/PAM250

[12] Τοποθεσία ftp του NCBI: ftp://ftp.ncbi.nih.gov/blast/matrices/BLOSUM62

[13] SwissProt Knowledgebase User Manual: http://au.expasy.org/sprot/userman.html

[14] D. J. Lipman, W. R. Pearson: Rapid and Sensitive Protein Similarity Searches, *Science*, 227:1435-1441, 1985.

[15] E. W. Myers: A Sublinear Algorithm for Approximate Keyword Searching, *Algorithmica*, 12:345-374, 1994.

[16] SunOS 5.5, *BLAST Manual Pages*, 1995.

[17] NCBI BLAST: http://www.ncbi.nlm.nih.gov/BLAST/

[18] C. S. Horstmann, G. Cornell: *Core Java 2: Volume I – Fundamentals*, Sun Microsystems Press, 2003.