ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗ ΒΙΟΪΑΤΡΙΚΗ

Υπολογιστικές μεθοδολογίες για την εύρεση πρωτεϊνικών περιοχών

ΕΠΙΒΛΕΠΩΝ

ΠΑΝΤΕΛΕΗΜΩΝ ΜΠΑΓΚΟΣ, **ΚΑΘΗΓΗΤΗΣ**

ΕΥΑΓΓΕΛΙΑ ΚΟΥΡΤΖΕΛΛΗ

(ΑΜ: 01921)

ΛΑΜΙΑ 2025

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗ ΒΙΟΪΑΤΡΙΚΗ

Υπολογιστικές μεθοδολογίες για την εύρεση πρωτεϊνικών περιοχών

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΕΥΑΓΓΕΛΙΑ ΚΟΥΡΤΖΕΛΛΗ

ΕΠΙΒΛΕΠΩΝ

ΠΑΝΤΕΛΕΗΜΩΝ ΜΠΑΓΚΟΣ, **ΚΑΘΗΓΗΤΗΣ**

ΛΑΜΙΑ, 2025

Υπολογιστικές μεθοδολογίες για την εύρεση πρωτεϊνικών περιοχών

ΤΡΙΜΕΛΗΣ ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ:

Παντελεήμων Μπάγκος, Καθηγητής, Τμήμα Πληροφορικής με Εφαρμογές στη Βιοϊατρική
Γεωργία Μπράλιου, Αναπληρώτρια Καθηγήτρια, Τμήμα Πληροφορικής με Εφαρμογές στη Βιοϊατρική
Παναγιώτα Κοντού, Επίκουρος Καθηγήτρια, Τμήμα Μαθηματικών

Με ατομική μου ευθύνη και γνωρίζοντας τις κυρώσεις [(1)], που προβλέπονται από της διατάξεις της παρ. 6 του άρθρου 22 του Ν. 1599/1986, δηλώνω ότι:

1. *Δεν παραθέτω κομμάτια βιβλίων ή άρθρων ή εργασιών άλλων αυτολεξεί **χωρίς να τα περικλείω σε εισαγωγικά** και χωρίς να αναφέρω το συγγραφέα, τη χρονολογία, τη σελίδα. Η αυτολεξεί παράθεση χωρίς εισαγωγικά χωρίς αναφορά στην πηγή, είναι λογοκλοπή. Πέραν της αυτολεξεί παράθεσης, λογοκλοπή θεωρείται και η παράφραση εδαφίων από έργα άλλων, συμπεριλαμβανομένων και έργων συμφοιτητών μου, καθώς και η παράθεση στοιχείων που άλλοι συνέλεξαν ή επεξεργάσθηκαν, χωρίς αναφορά στην πηγή. Αναφέρω πάντοτε με πληρότητα την πηγή κάτω από τον πίνακα ή σχέδιο, όπως στα παραθέματα.*

2. *Δέχομαι ότι η αυτολεξεί **παράθεση χωρίς εισαγωγικά**, ακόμα κι αν συνοδεύεται από αναφορά στην πηγή σε κάποιο άλλο σημείο του κειμένου ή στο τέλος του, είναι αντιγραφή. Η αναφορά στην πηγή στο τέλος π.χ. μιας παραγράφου ή μιας σελίδας, δεν δικαιολογεί συρραφή εδαφίων έργου άλλου συγγραφέα, έστω και παραφρασμένων, και παρουσίασή τους ως δική μου εργασία.*

3. *Δέχομαι ότι υπάρχει επίσης περιορισμός στο μέγεθος και στη συχνότητα των παραθεμάτων που μπορώ να εντάξω στην εργασία μου εντός εισαγωγικών. Κάθε μεγάλο παράθεμα (π.χ. σε πίνακα ή πλαίσιο, κλπ), προϋποθέτει ειδικές ρυθμίσεις, και όταν δημοσιεύεται προϋποθέτει την άδεια του συγγραφέα ή του εκδότη. Το ίδιο και οι πίνακες και τα σχέδια*

4. *Δέχομαι όλες τις συνέπειες σε περίπτωση λογοκλοπής ή αντιγραφής.*

Ημερομηνία: ……/…../20……

Ο – Η Δηλ.

(Υπογραφή)

(1) «Όποιος εν γνώσει του δηλώνει ψευδή γεγονότα ή αρνείται ή αποκρύπτει τα αληθινά με έγγραφη υπεύθυνη δήλωση του άρθρου 8 παρ. 4 Ν. 1599/1986 τιμωρείται με φυλάκιση τουλάχιστον τριών μηνών. Εάν ο υπαίτιος αυτών των πράξεων σκόπευε να προσπορίσει στον εαυτόν του ή σε άλλον περιουσιακό όφελος βλάπτοντας τρίτον ή σκόπευε να βλάψει άλλον, τιμωρείται με κάθειρξη μέχρι 10 ετών.

Ευχαριστίες

Με τη ολοκλήρωση της πτυχιακής μου εργασίας θα ήθελα να εκφράσω τις εγκάρδιες ευχαριστίες μου σε όσους συνέβαλαν στην υλοποίησή της.

Αρχικά, θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου, κ. Παντελεήμονα Μπάγκο, για την πολύτιμη καθοδήγηση, τις γνώσεις και τις ιδέες που μου προσέφερε. Η καθοδήγηση του και η ερευνητική του προσέγγιση με ενθάρρυναν να εμβαθύνω περισσότερο στο αντικείμενο και μου καλλιέργησαν πραγματικό ενδιαφέρον στο κομμάτι της έρευνας. Επίσης, ευχαριστίες εκφράζονται στα υπόλοιπα δύο μέλη της επιτροπής, την κ. Γεωργία Μπράλιου και την κ. Παναγιώτα Κοντού για την υποστήριξή τους.

Ευχαριστώ και τον κ. Γεώργιο Μανιό για τη συνεχή βοήθεια και τις καίριες συμβουλές του κατά τη διάρκεια της εργασίας.

Τέλος, νιώθω βαθιά ευγνωμοσύνη προς την οικογένειά μου και τους φίλους μου, για την αδιάκοπη συμπαράστασή τους και την παρουσία τους σε κάθε βήμα αυτής της διαδρομής.

Περίληψη

Η μετρική φύση των πρωτεϊνών υπαγορεύει ότι συγκεκριμένες περιοχές, γνωστές ως επικράτειες (domains), αποτελούν τα θεμελιώδη δομικά στοιχεία όλων των πρωτεϊνικών αλληλουχιών. Οι περιοχές αυτές συχνά εμφανίζονται με διαφορετικούς συνδυασμούς, συμβάλλοντας στη λειτουργική ποικιλομορφία των πρωτεϊνών. Για τον εντοπισμό τους, έχουν αναπτυχθεί υπολογιστικές μεθοδολογίες όπως τα profile Hidden Markov Models (HMMs), που αξιοποιούνται από εργαλεία όπως το HMMER και βάσεις δεδομένων όπως η PFAM.

Η παρούσα πτυχιακή εργασία επικεντρώνεται στην ανάπτυξη μιας μεθοδολογίας για την ανακάλυψη νέων ή άγνωστων περιοχών σε πρωτεΐνες, συγκεκριμένα των β-βαρελιών (beta barrels) με έμφαση στην εύρεση συνδυασμών Pfam δομικών πεδίων που συνυπάρχουν με διαμεμβρανικές δομές τύπου β-βαρελιού (β-barrels). Χρησιμοποιούνται δεδομένα από τις βάσεις PFAM και OMPdb, καθώς και το εργαλείο hmmscan του λογισμικού HMMER, προκειμένου να εντοπιστούν πιθανές συσχετίσεις μεταξύ απλών Pfam domains και β-βαρελιών από τη βάση OMPdb. Μέσω αυτής της ανάλυσης, επιχειρείται να ανακαλυφθούν νέες β-βαρελοειδείς πρωτεΐνες και να σκιαγραφηθούν πιθανά λειτουργικά μοτίβα, εξελικτικές σχέσεις ή ακόμα και νέες οικογένειες β-βαρελιών.

Λέξεις - Κλειδιά: Προφίλ Hidden Markov Model (pHMM), γονιδίωμα, Πρωτεϊνικά domains, β-βαρέλια, HMMER, Pfam, OMPdb, Συνύπαρξη domains, Μεμβρανικές Πρωτεΐνες

Abstract

The structural nature of proteins dictates that specific regions, known as domains, are the fundamental building blocks of all protein sequences. These domains often occur in different combinations, contributing to the functional diversity of proteins. To identify them, computational methodologies such as profile Hidden Markov Models (HMMs) have been developed, exploited by tools such as HMMER and databases such as PFAM.

This thesis focuses on the development of a methodology for the discovery of new or unknown regions in proteins, namely beta barrels, with a focus on finding combinations of Pfam structural domains that coexist with transmembrane-like beta barrel structures (β-barrels). Data from the PFAM and OMPdb databases, as well as the HMMER hmmscan tool, are used to identify possible associations between simple domains and β-barrels. Through this analysis, an attempt is made to discover novel β-barrel proteins and to outline possible functional patterns or evolutionary relationships and even discover a new family of β-barrels domains.

Key – Words: Profile Hidden Markov Model (pHMM), Genome, Protein domains, Beta-barrel, HMMER, Pfam, OMPdb, Domain co-occurrence, Membrane protein

Περιεχόμενα

1. **Εισαγωγή**

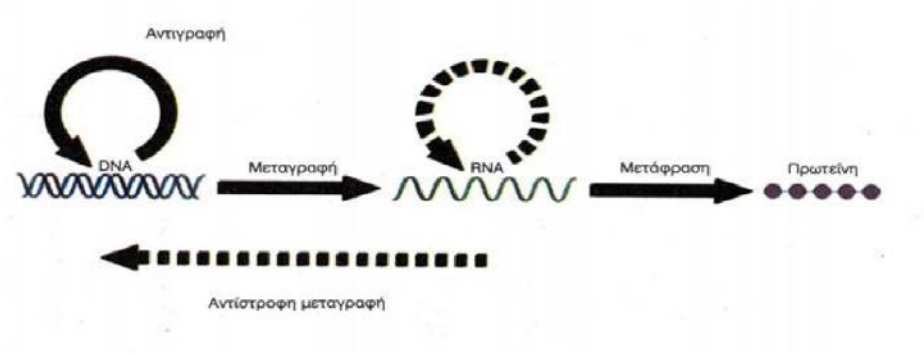
1.1 Πρωτεΐνες

Οι πρωτεΐνες είναι πρωταρχικής σημασίας βιολογικά μακρομόρια με σημαντικό ρόλο στις περισσότερες χημικές διεργασίες που είναι απαραίτητες για τη ζωή. Ο όρος πρωτεΐνη προέρχεται από την ελληνική λέξη «πρώτειος», που σημαίνει «κατέχοντας την πρώτη θέση» και αυτό αποδεικνύει τη σημασία τους. Αποτελούνται από αλυσίδες αμινοξέων οι οποίες συνδέονται μεταξύ τους με πεπτιδικούς δεσμούς. Υπάρχουν 20 διαφορετικά αμινοξέα που συναντώνται στις πρωτεΐνες .Η σύσταση και η αλληλουχία της πρωτεΐνης καθορίζει την τελική τρισδιάστατη δομή ,η οποία και καθορίζει τη βιολογική λειτουργία κάθε πρωτεΐνης. Κάθε αμινοξύ σε μία πρωτεΐνη διαθέτει μία χαρακτηριστική πλευρική ομάδα (R), η οποία σχετίζεται με τις ιδιότητες και την αλληλεπίδραση της πρωτεΐνης με άλλες ενώσεις ή μόρια.

Η δομή μίας πρωτεΐνης κατηγοριοποιείται σε 4 επίπεδα, στην πρωτογενή δομή (η αλληλουχία των αμινοξέων) , στη δευτερογενή δομή (χωρική διάταξη της κύριας πεπτιδικής αλυσίδας) , στη τριτογενή δομή (η διάταξη των πλευρικών αλυσίδων και των άλλων γειτονικών τμημάτων της κύριας αλυσίδας, χωρίς να λαμβάνονται υπόψη οι γειτονικές πεπτιδικές αλυσίδες) και στη τεταρτοταγή δομή (που είναι η διάταξη πανομοιότυπων ή διαφορετικών υπομονάδων μιας μεγάλης πρωτεΐνης, όπου κάθε υπομονάδα είναι μία ξεχωριστή πεπτιδική αλυσίδα).

Οι πρωτεΐνες έχουν κύριο ρόλο στις περισσότερες κυτταρικές διεργασίες, όπως η κατάλυση χημικών αντιδράσεων (ένζυμα), η κυτταρική σήμανση, η ανοσολογική απόκριση, η μεταφορά μορίων, καθώς και η δομική υποστήριξη κυτταρικών δομών.

Η γενετική πληροφορία ρέει από το DNA στο RNA πριν μεταφραστεί σε πρωτεΐνη. Το DNA μεταγράφεται σε RNA με τη βοήθεια της RNA πολυμεράσης, η οποία «διαβάζει» την αλληλουχία του DNA και συνθέτει mRNA. Στη συνέχεια, η μετάφραση μετατρέπει το mRNA σε πρωτεΐνη: το mRNA προσκολλάται στα ριβοσώματα, τα οποία καθορίζουν την αλληλουχία αμινοξέων. Η μετάφραση γίνεται από το 5' προς το 3' άκρο. Το αμινοτελικό άκρο σηματοδοτεί την έναρξη της πρωτεϊνοσύνθεσης και η διαδικασία συνεχίζεται έως το καρβοξυτελικό άκρο. Ο γενετικός κώδικας χρησιμοποιεί τρία νουκλεοτίδια (κωδικόνιο) για τον καθορισμό συγκεκριμένων αμινοξέων.



Εικόνα 1 . 1 Το κεντρικό δόγμα της μοριακής βιολογίας.

1.2.1 **Πρωτεϊνικές Περιοχές**

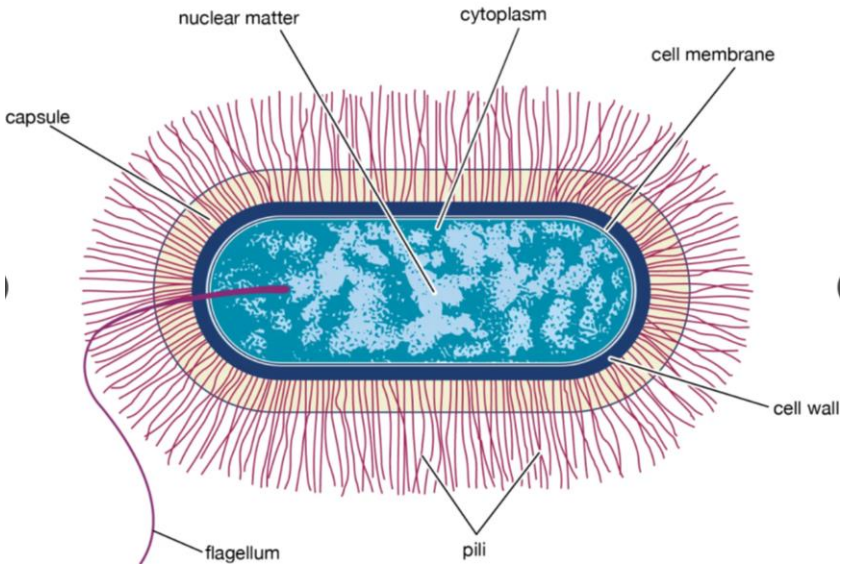
Οι πρωτεϊνικές επικράτειες ή αλλιώς δομικά domains είναι αυτοτελή, λειτουργικά και δομικά διακριτά τμήματα μιας πρωτεΐνης. Πρόκειται για τμήματα της πρωτεϊνικής αλυσίδας που έχουν τη δυνατότητα να αναδιπλώνονται ανεξάρτητα σε μια σταθερή τρισδιάστατη δομή και συνήθως συνδέονται με κάποια συγκεκριμένη βιολογική λειτουργία. Η παρουσία και ο συνδυασμός τέτοιων περιοχών καθορίζει σε μεγάλο βαθμό την ποικιλομορφία των πρωτεϊνών και την εξειδίκευσή τους σε διαφορετικά κυτταρικά καθήκοντα. Ένα δομικό domain συνήθως αποτελείται από περίπου 100 έως 150 αμινοξέα και μπορεί να παρατηρηθεί σε πολλές διαφορετικές πρωτεΐνες, ακόμη και σε μη ομόλογες μεταξύ τους, γεγονός που αποδεικνύει ότι οι πρωτεϊνικές περιοχές λειτουργούν ως «δομικά τούβλα» της εξέλιξης. Μέσω μηχανισμών όπως η συνένωση γονιδίων, το εναλλακτικό μάτισμα και ο ανασυνδυασμός εξωνίων, τα domains μπορούν να επαναχρησιμοποιηθούν, να συνδυαστούν με νέους τρόπους ή να εξελιχθούν σε διαφορετικές μορφές, επιτρέποντας τη δημιουργία νέων πρωτεϊνικών λειτουργιών.

Η λειτουργία ενός domain μπορεί να διατηρηθεί σταθερή κατά την εξέλιξη, αλλά μπορεί επίσης να μεταβληθεί μέσω μεταλλάξεων. Ακόμα και μικρές αλλαγές στη δομή ενός domain μπορούν να έχουν σημαντικές επιπτώσεις στη λειτουργία του, οδηγώντας είτε σε εξειδίκευση είτε σε απώλεια αρχικής λειτουργίας. Κάποια domains εμφανίζονται συχνά σε πολλαπλές πρωτεΐνες (promiscuous domains), ενώ άλλα περιορίζονται σε πολύ συγκεκριμένα πρωτεϊνικά συμφραζόμενα.

Συνολικά, οι πρωτεϊνικές περιοχές αντιπροσωπεύουν βασικές μονάδες δομής και λειτουργίας, μέσω των οποίων οι πρωτεΐνες αποκτούν την απαραίτητη ποικιλομορφία για την εκτέλεση των πολλών και διαφορετικών ρόλων τους στους ζωντανούς οργανισμούς.

1.2.2 **Βακτήρια**

Τα βακτήρια είναι μονοκύτταροι, μικροσκοπικοί οργανισμοί που βρίσκονται σχεδόν σε κάθε περιβάλλον στη Γη, συμπεριλαμβανομένων των φρεατίων βαθιάς θάλασσας, πολύ κάτω από την επιφάνεια της Γης και μέσα στο πεπτικό σύστημα του ανθρώπου. Ανήκουν σε μια ομάδα μονοκύτταρων μορφών ζωής που ονομάζονται προκαρυώτες, επειδή στερούνται πυρήνα που συνδέεται με μεμβράνη και άλλων εσωτερικών δομών. Οι προκαρυώτες είναι οι κυρίαρχοι ζωντανοί οργανισμοί στη Γη και έχουν προσαρμοστεί σε όλα σχεδόν τα διαθέσιμα οικολογικά ενδιαιτήματα [1].Τα μεμονωμένα βακτήρια μπορούν να λάβουν ένα από τρία βασικά σχήματα: Σφαιρικό (κόκκος), ραβδόμορφο (βάκιλος), καμπύλο (vibrio, spirillum ή spirochete). Τα βακτηριακά κύτταρα έχουν συνήθως μήκος από 0,5 έως 5,0 μικρόμετρα (μm).



Εικόνα 1.2 Σχηματική απεικόνιση της δομής ενός τυπικού βακτηριακού κυττάρου τύπου βάκιλλου[3].

Τα βακτήρια αναπαράγονται κυρίως με ασεξουαλική αναπαραγωγή, δηλαδή χωρίς ανταλλαγή γενετικού υλικού με άλλο οργανισμό. Ο βασικός και συχνότερος μηχανισμός είναι η δυαδική σχάση, μια διαδικασία κατά την οποία το κύτταρο αναπτύσσεται και διαιρείται σε δύο πανομοιότυπα θυγατρικά κύτταρα.

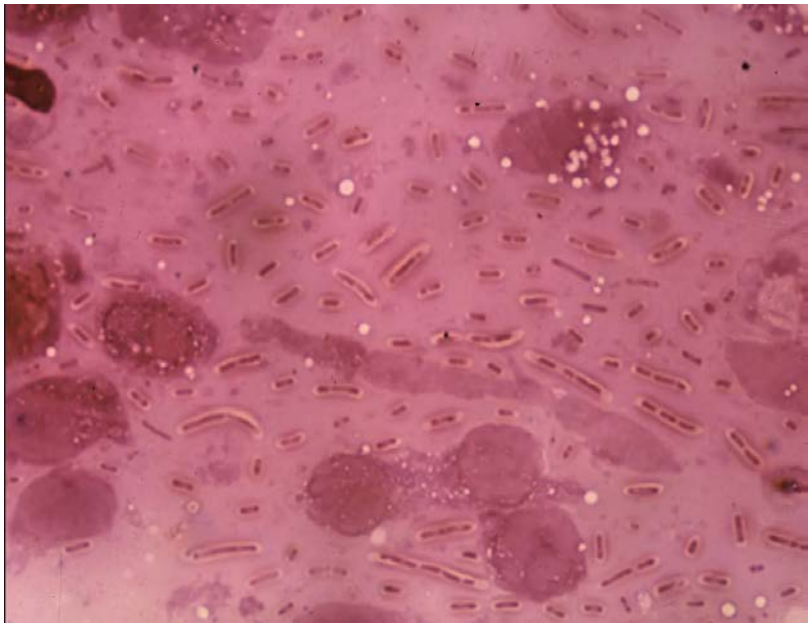
Τα βακτήρια ταξινομούνται με βάση διάφορα χαρακτηριστικά, συμπεριλαμβανομένου του σχήματος των κυττάρων, των πολυκυτταρικών συσσωματωμάτων, της κινητικότητας, του σχηματισμού σπορίων και της αντίδρασης στη χρώση Gram. Οι βιοχημικές δοκιμές, όπως η χρησιμοποίηση θρεπτικών ουσιών, τα μεταβολικά προϊόντα, η απόκριση σε συγκεκριμένες χημικές ουσίες και η παρουσία χαρακτηριστικών ενζύμων, είναι επίσης σημαντικές για την ταυτοποίηση βακτηριακών γενών και ειδών [2].

1.2.3 Gram Negative Bacteria

Τα αρνητικά κατά Gram βακτήρια είναι μια ομάδα βακτηρίων με ποικίλες μορφές που χαρακτηρίζεται από ένα λεπτό κυτταρικό τοίχωμα πεπτιδογλυκάνης που βρίσκεται μεταξύ της κυτταροπλασματικής μεμβράνης και μιας εξωτερικής μεμβράνης που περιέχει λιποπολυσακχαρίτη, η οποία περιβάλλεται από μια κάψουλα. Ονομάζονται "αρνητικά κατά Gram" εξαιτίας της αντίδρασής τους στη χρώση κατά Gram, μια μικροβιολογική τεχνική χρώσης που χρησιμοποιείται για την ταυτοποίηση και τον χαρακτηρισμό των βακτηρίων. Στη διαδικασία χρώσης κατά Gram, τα βακτήρια χρωματίζονται αρχικά με κρυσταλλικό ιώδες, το οποίο χρωματίζει όλα τα κύτταρα μωβ. Στη συνέχεια χρησιμοποιείται διάλυμα ιωδίου και ένα οργανικό διαλύτη, όπως το αλκοόλ ή η ακετόνη. Τα Gram-αρνητικά βακτήρια χάνουν το μωβ χρώμα λόγω του λεπτού στρώματος πεπτιδογλυκάνης και στη συνέχεια χρωματίζονται με σαφρανίνη, αποκτώντας ροζ ή κόκκινο χρώμα , γεγονός που διαφέρει από τα θετικά κατά Gram βακτήρια, τα οποία χρωματίζονται μοβ λόγω των παχιών κυτταρικών τοιχωμάτων τους [4].

Το κυτταρικό τοίχωμα των Gram-αρνητικών βακτηρίων είναι σύνθετο και έχει εξωτερική μεμβράνη από φωσφολιπίδια, λιποπολυσακχαρίτες, και ένα λεπτό στρώμα πεπτιδογλυκάνης. Οι λιποπολυσακχαρίτες, που είναι γνωστοί και ως ενδοτοξίνες, προκαλούν πυρετό και σοκ σε ζώα και ανθρώπους. Η εξωτερική μεμβράνη τους ,τους προσφέρει προστασία λειτουργεί από επιβλαβείς χημικές ουσίες που αυξάνουν την ανθεκτικότητα των βακτηρίων σε αντίξοες συνθήκες.

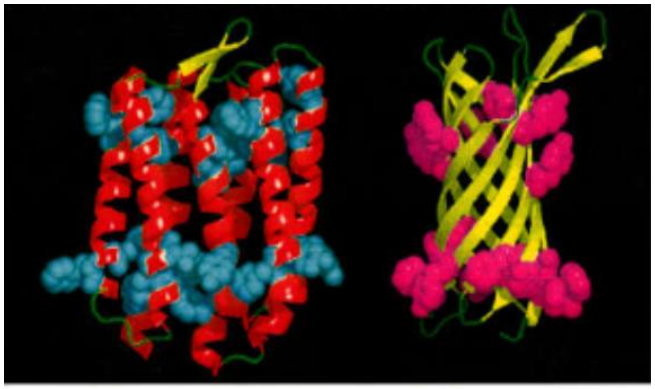
Παραδείγματα αρνητικών κατά Gram βακτηρίων είναι τα Enterobacter, Escherichia, Haemophilus, Klebsiella, Pseudomonas, Salmonella, Shigella και Yersinia. Πολλά από αυτά μπορούν να προκαλέσουν σοβαρές ασθένειες στον άνθρωπο, όπως μηνιγγίτιδα, πνευμονία, σηψαιμία και λοιμώξεις του ουροποιητικού συστήματος. Οι λοιμώξεις που αφορούν αρνητικά κατά Gram βακτήρια περιπλέκονται από την ανθεκτικότητα στα αντιβιοτικά, επειδή πολλοί αρνητικοί κατά Gram οργανισμοί είναι φυσικά ανθεκτικοί στα ευρέως χρησιμοποιούμενα αντιβιοτικά και μπορούν να αποκτήσουν γρήγορα και να μεταδώσουν την ανθεκτικότητα σε άλλα βακτηριακά κύτταρα μετά από έκθεση σε αντιβιοτικά [5].



Εικόνα 1.3 Gram-αρνητικοί βάκιλλοι, Klebsiella pneumoniae, απομονωμένοι από απόστημα πνεύμονα σε ασθενή με πνευμονία [6].

1.2.4 **β-μεμβρανικές πρωτεΐνες**

Οι μεμβρανικές πρωτεΐνες χωρίζονται σε δύο κατηγορίες ανάλογα με τη δευτερογενή δομή που παίρνουν τα τμήματα τους που διαπερνούν τη λιπιδική διπλοστοιβάδα. Έτσι, οι α-μεμβρανικές πρωτεΐνες έχουν τμήματα που σχηματίζουν δομή α-έλικας και οι β-μεμβρανικές πρωτεΐνες δομές που σχηματίζουν δομές β-έλικας (Εικόνα 1.4). Οι α-ελικοειδείς μεμβρανικές πρωτεΐνες παρατηρούνται σχεδόν σε όλες τις κυτταρικές μεμβράνες ενώ οι β-ελικοειδείς έχουν παρατηρηθεί μόνο στη εξωτερική μεμβράνη των gram negative βακτηρίων [7].



Εικόνα 1.4 Οι δύο κατηγορίες διαμεμβρανικών πρωτεϊνών. Αριστερά βλέπουμε α-μεμβρανική πρωτεΐνη ενώ δεξιά β-μεμβρανική πρωτεΐνη [8].

Τα gram negative bacteria έχουν τόσο εσωτερική (IM) όσο και εξωτερική μεμβράνη (OM), οι οποίες διαδραματίζουν σημαντικό ρόλο στη μεταφορά θρεπτικών, τη σηματοδότηση, την αποβολή αποβλήτων και την προστασία. Η εσωτερική μεμβράνη αποτελείται από διαμεμβρανικές πρωτεΐνες με α-ελικοειδή δομή, ενώ αντίθετα η εξωτερική μεμβράνη περιέχει σχεδόν αποκλειστικά διαμεμβρανικές πρωτεΐνες με β-βαρελιακή δομή [9], που αποτελούνται από 8–26 αντιπαράλληλες β-έλικες που σχηματίζουν κυλινδρικές δομές και αγκυροβολούν τη πρωτεΐνη στην OM [11].

Οι β-μεμβρανικές πρωτεΐνες αποτελούν μια σημαντική κατηγορία πρωτεϊνών που βρίσκονται κυρίως στην εξωτερική μεμβράνη των Gram-αρνητικών βακτηρίων, των μιτοχονδρίων και των χλωροπλαστών. Σε αντίθεση με τις περισσότερες διαμεμβρανικές πρωτεΐνες που δομούνται με α-έλικες, οι β-μεμβρανικές πρωτεΐνες αποτελούνται από β-έλικες (β-strands), οι οποίες οργανώνονται σε κυλινδρικές δομές, σχηματίζοντας βαρέλια [10]. Λόγω του υδρόφοβου περιβάλλοντος της μεμβράνης, οι εξωτερικές επιφάνειες αυτών των β-έλικων είναι υδρόφοβες, ενώ το εσωτερικό τους συνήθως είναι υδρόφιλο, επιτρέποντας τη δημιουργία διαύλων μέσω των οποίων μπορούν να περάσουν υδατοδιαλυτές ουσίες[8].Η δομή αυτή διευκολύνει τη μεταφορά μορίων μέσα από τη μεμβράνη.

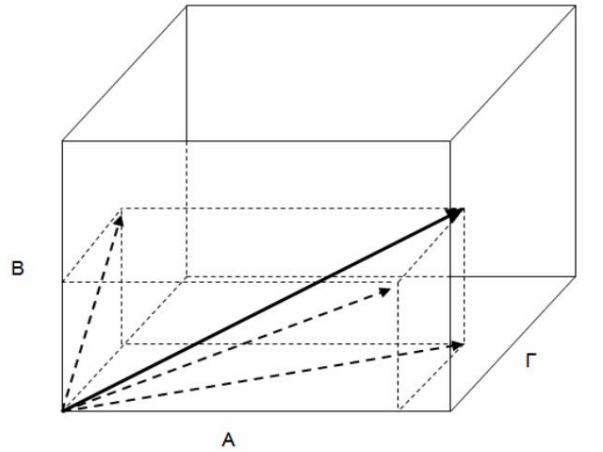
Στα Gram-negative βακτηρία οι β-μεμβρανικές πρωτεΐνες (β-barrel proteins) ενσωματώνονται στην εξωτερική μεμβράνη μέσω μιας εξειδικευμένης διαδικασίας. Αρχικά με μια σηματοδοτική αλληλουχία συντίθενται στο κυτταρόπλασμα ,όπου και στη συνέχεια θα μεταφερθούν μέσω του συστήματος Sec στον περιπλασματικό χώρο. Εκεί, ειδικές συνοδοί πρωτεΐνες, οι Skp και SurA αποτρέπουν την πρόωρη αναδίπλωση και συσσωμάτωση. Τελικά, η ενσωμάτωσή τους γίνεται στην εξωτερική μεμβράνη πραγματοποιείται με τη βοήθεια από το σύμπλοκο Bam, το οποίο αποτελείται από την πρωτεΐνη BamA και επιπλέον λιποπρωτεϊνικές υπομονάδες (BamB–E) [11]. Η διαδικασία αυτή είναι εξελικτικά συντηρημένη και μοιάζει με την ενσωμάτωση παρόμοιων πρωτεϊνών στα μιτοχόνδρια των ευκαρυωτικών κυττάρων.

Οι πρωτεΐνες τύπου β-βαρελιού (β-barrel) επιτελούν ποικίλες λειτουργίες· ανάμεσά τους περιλαμβάνονται πρωτεΐνες που σχηματίζουν πόρους και επιτρέπουν τη διέλευση μεταβολιτών μέσω της μεμβράνης με παθητική διάχυση, ενεργοί μεταφορείς σιδηροφόρων, ένζυμα, δομικές πρωτεΐνες και πρωτεΐνες που διαμεσολαβούν στη μεταφορά ή την ενσωμάτωση πρωτεϊνών διαμέσου ή μέσα σε μεμβράνες. Δεν έχουν όλες οι β-μεμβρανικές πρωτεΐνες ρόλο διαύλου αλλά έχουν εξελιχθεί για διαφορετικές λειτουργίες. Για παράδειγμα, η OmpA συμβάλλει στη σταθεροποίηση της εξωτερικής μεμβράνης, ενώ η OmpX συμμετέχει στην αγκύρωση ή την προσκόλληση κυττάρων [11].

1.3.1 **Πολλαπλή Στοίχιση Ακολουθιών**

Η πολλαπλή στοίχιση είναι μία πολύ σημαντική μέθοδος στην βιοπληροφορική που μας επιτρέπει στοιχίζοντας περισσότερες από τρεις βιολογικές αλληλουχίες (κυρίως πρωτεϊνικές), να ανιχνεύσουμε αν υπάρχουν κάποιες συντηρημένες περιοχές και με αυτόν τον τρόπο δίνεται η δυνατότητα για ανάλυση της δομής και άρα της λειτουργίας τους. Χρησιμοποιούμε τη πολλαπλή στοίχιση κυρίως για τρεις σκοπούς. Πρώτον, μέσω της στοίχισης συγγενικών ακολουθιών μπορούμε να εντοπίσουμε διατηρημένες περιοχές και στη συνέχεια να προσδιορίσουμε τα λειτουργικά και δομικά σημαντικά στοιχεία, όπως τα ενεργά κέντρα ενζύμων ή χαρακτηριστικά δευτεροταγούς δομής. Το αποτέλεσμα μπορεί να αποτυπωθεί με τη μορφή προφίλ ή μοντέλων (όπως Hidden Markov Models), τα οποία χρησιμοποιούνται στη δευτερογενή αναζήτηση παρόμοιων ακολουθιών σε βάσεις δεδομένων (π.χ. PROSITE, PFAM). Αυτό διευκολύνει τη διαδικασία του χαρακτηρισμού ολόκληρων οικογενειών πρωτεϊνών. Επίσης, με τη πολλαπλή στοίχιση είμαστε σε θέση να ανακαλύψουμε συντηρημένα μοτίβα ,δηλαδή μοτίβα που έχουν έναν κοινό πρόγονο και άρα να ανακαλύψουμε εξελικτικές σχέσεις μεταξύ των ακολουθιών. Οι ομοιότητες και διαφορές τους χρησιμοποιούνται για την κατασκευή φυλογενετικών δέντρων, τα οποία απεικονίζουν τη διαδικασία απόκλισης από έναν κοινό πρόγονο. Τέλος η πολλαπλή στοίχιση μπορεί να συμβάλει στη βελτίωση της ακρίβειας και της αποτελεσματικότητας των μεθόδων πρόγνωσης δομής πρωτεϊνών και στην καλύτερη γνώση των χαρακτηριστικών των πρωτεϊνών [12].

Στη πολλαπλή στοίχιση επεκτείνονται οι τεχνικές του δυναμικού προγραμματισμού σε r-διαστάσεις. Συγκεκριμένα, κάθε διαδοχική ευθυγράμμιση αντιπροσωπεύεται σε έναν πολυδιάστατο πίνακα, και η βέλτιστη στοίχιση προκύπτει ως η βέλτιστη διαδρομή εντός αυτού του χώρου. Το σκορ της στοίχισης είναι το άθροισμα των επιμέρους στηλών της ευθυγράμμισης και λαμβάνει υπόψη τη συμβατότητα χαρακτήρων και τα κενά μέσω κατάλληλων συναρτήσεων κόστους [12].



Εικόνα 1.5 Απεικόνιση του πίνακα δυναμικού προγραμματισμού για την πολλαπλή στοίχιση τριών ακολουθιών [10].

Ο κύβος αναπαριστά το τρισδιάστατο πλέγμα στοίχισης τριών ακολουθιών (Α, Β, Γ), με τους άξονες x, y, z να αντιστοιχούν σε κάθε μία από αυτές. Κάθε σημείο του πλέγματος έχει συντεταγμένες (i, j, k), που υποδεικνύουν συγκεκριμένες θέσεις στις αντίστοιχες ακολουθίες.Οι γραμμές και τα βέλη δείχνουν δυνατές 'διαδρομές' μέσα στο πλέγμα, δηλαδή τρόπους με τους οποίους μπορούμε να προχωρήσουμε από τη μία θέση στην άλλη, με σκοπό να στοιχίσουμε χαρακτήρες ή να εισάγουμε κενά. Η παχιά γραμμή είναι η βέλτιστη διαδρομή (στοίχιση δηλαδή) που έχει υπολογιστεί, ενώ οι διακεκομμένες γραμμές δείχνουν άλλες εναλλακτικές διαδρομές.

1.3.2 **Προοδευτική πολλαπλή στοίχιση**

Η προοδευτική πολλαπλή στοίχιση (MSA) είναι μία από τις πιο ευρέως χρησιμοποιούμενες ευριστικές τεχνικές για την στοίχιση πολλών βιολογικών ακολουθιών (όπως πρωτεϊνών ή DNA). Η μέθοδος βασίζεται στην σταδιακή προσθήκη ακολουθιών σε μια υπάρχουσα στοίχιση, ξεκινώντας από τα πιο όμοια ζεύγη και επεκτείνοντάς την προοδευτικά με βάση την ομοιότητά τους. Η βασική ιδέα αναπτύχθηκε από τους Feng και Doolittle (1987) και περιλαμβάνει τα εξής βήματα:

- Υπολογισμός όλων των δυνατών κατά ζεύγη στοιχίσεων μεταξύ των ακολουθιών.
- Δημιουργία πίνακα αποστάσεων και οδηγού δέντρου (guide tree) με βάση την ομοιότητα των ακολουθιών.
- Σταδιακή στοίχιση των ακολουθιών σύμφωνα με τη δομή του οδηγού δέντρου.

Η προσέγγιση αυτή είναι ευέλικτη και μπορεί να ενσωματώνει διαφορετικές στρατηγικές σε κάθε στάδιο. Για παράδειγμα, η κατά ζεύγη στοίχιση μπορεί να γίνει είτε με ευριστικές μεθόδους όπως BLAST ή FASTA, είτε με δυναμικό προγραμματισμό. Οι αποστάσεις μπορεί να υπολογίζονται είτε ως ποσοστά ομοιότητας είτε με πιο σύνθετους υπολογισμούς [12]. Ένα πολύ χρήσιμο εργαλείο που χρησιμοποιεί προοδευτική πολλαπλή στοίχιση είναι το CLUSTAL, και ιδιαίτερα οι εκδόσεις CLUSTALW και CLUSTALX.

1.3.3 **Κρυφά Μαρκοβιανά μοντέλα** (HMMs) **και** Profile Hidden Markov Models (pHMMs)

Τα HMM (Hidden Markov Models) είναι πιθανοθεωρητικά μοντέλα ακολουθιών, τα οποία χρησιμοποιούνται για την αντιστοίχιση κάθε μονάδας μιας ακολουθίας (όπως λέξεις, γράμματα ή αμινοξέα) σε ετικέτες ή κατηγορίες. Τα HMMs υπολογίζουν μια κατανομή πιθανοτήτων πάνω σε πιθανές ακολουθίες ετικετών και επιλέγουν τη βέλτιστη δυνατή, σύμφωνα με το μοντέλο. Τα Κρυφά Μαρκοβιανά Μοντέλα εισάγουν την έννοια των «κρυφών καταστάσεων». Στα HMMs, το σύστημα μεταβαίνει μεταξύ καταστάσεων με πιθανότητες μετάβασης, όπως σε μια απλή Μαρκοβιανή διαδικασία, ωστόσο οι ίδιες οι καταστάσεις δεν είναι άμεσα παρατηρήσιμες. Αντίθετα, κάθε κρυφή κατάσταση "εκπέμπει" ένα παρατηρούμενο σύμβολο με ορισμένη πιθανότητα [13].

Ένα HMM ορίζεται από:

- Ένα σύνολο κρυφών καταστάσεων.
- Ένα μοντέλο μετάβασης μεταξύ των καταστάσεων (πίνακας μεταβάσεων).
- Ένα μοντέλο εκπομπής, που καθορίζει την πιθανότητα παρατήρησης κάθε συμβόλου από κάθε

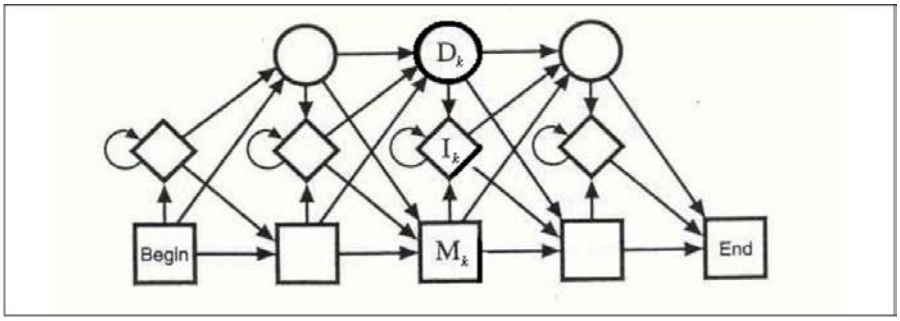
κατάσταση.

- Και έναν αρχικό καταμερισμό πιθανοτήτων για την κατάσταση έναρξης.

Το κύριο πρόβλημα που λύνουν τα HMMs είναι η εξαγωγή πληροφορίας από παρατηρήσιμα δεδομένα (όπως DNA, RNA ή ακολουθίες πρωτεϊνών), με σκοπό την ανακατασκευή της πιο πιθανής ακολουθίας κρυφών καταστάσεων. Αυτή η προσέγγιση είναι θεμελιώδης για πολλές εφαρμογές στη βιοπληροφορική, όπως η αναγνώριση γονιδίων, η ανάλυση γονιδιωματικών περιοχών, στοίχιση ακολουθιών και η πρόβλεψη δομικών και λειτουργικών περιοχών σε πρωτεΐνες.

Τα Profile Hidden Markov Models (pHMMs) είναι μια εξειδικευμένη κατηγορία των Hidden Markov Models που μοντελοποιούν πολλαπλές στοιχίσεις αλληλουχιών με πιθανοθεωρητικό τρόπο (Eddy, 1998). Στα Profile Hidden Markov Models κάθε κατάσταση αντιστοιχεί σε συγκεκριμένη θέση (στήλη) της πολλαπλής στοίχισης, με position-specific παραμέτρους. Έτσι, η κατεύθυνση των μεταβάσεων είναι μονόδρομη (left-to-right), σε αντίθεση με τα κλασικά κυκλικά HMMs που το μοντέλο έχει την δυνατότητα να επισκεφθεί μία κατάσταση περισσότερες από μία φορές. Επιπλέον, τα pHMMs δεν μοντελοποιούν μόνο τις πιθανότητες εμφάνισης συμβόλων σε κάθε θέση, αλλά και τις πιθανότητες εισαγωγής κενών και απαλοιφών με πιθανοθεωρητικό τρόπο. Με αυτόν τον τρόπο λύνεται το πρόβλημα της κλασικής στοίχισης, δηλαδή το πρόβλημα επιλογής της ποινής για τα κενά.

Μια ακόμη βασική διαφορά των pHMM σε σχέση με τα κλασικά HMM είναι η ύπαρξη σιωπηρών καταστάσεων. Αυτές οι καταστάσεις δεν παράγουν σύμβολα και χρησιμεύουν για να επιτρέπουν άμεσες μεταβάσεις ανάμεσα σε απομακρυσμένες θέσεις του μοντέλου, χωρίς να απαιτούνται ενδιάμεσα βήματα. Έτσι, μειώνεται η πολυπλοκότητα και ο αριθμός των παραμέτρων που χρειάζεται να υπολογιστούν.



Εικόνα 1.6 Σχηματική αναπαράσταση ενός τυπικού profile Hidden Markov Model [13].

Οι καταστάσεις που συναντώνται σε ένα pHMM διακρίνονται σε (πλην των καταστάσεων εκκίνησης και τερματισμού):

- Καταστάσεις Ταύτισης (Match states – Mk), που αντιστοιχούν σε καλά στοιχισμένες θέσεις,

- Καταστάσεις Εισαγωγής (Insertion states – Ik), που αντιστοιχούν σε επιπλέον χαρακτήρες,
- Καταστάσεις Απαλοιφής (Deletion states – Dk), που δεν παράγουν σύμβολα και μοντελοποιούν τα κενά.

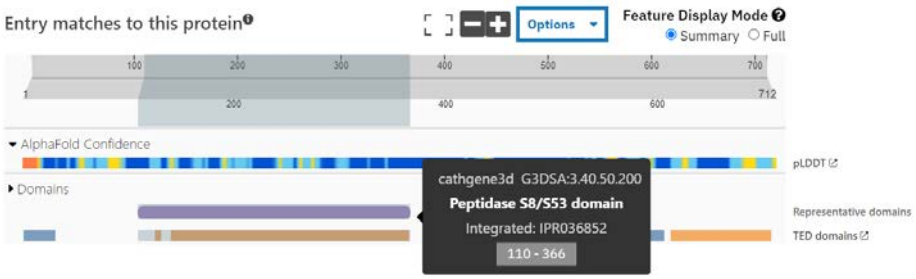
Η ακολουθία καταστάσεων είναι κρυφή, ενώ η παρατηρούμενη αλληλουχία παράγεται από αυτές μέσω πιθανοτήτων γέννησης συμβόλων. Οι καταστάσεις ταύτισης και εισαγωγής είναι κανονικές (emitting) καταστάσεις, ενώ οι απαλοιφές είναι σιωπηρές [13].

1.3.4 PFAM

Η βάση δεδομένων Pfam είναι μια ευρέως γνωστή συλλογή από οικογένειες πρωτεϊνών, καθεμία από τις οποίες αντιπροσωπεύεται από στοιχίσεις πολλαπλών αλληλουχιών και κρυφά Μαρκοβιανά μοντέλα (HMMs). Χρησιμοποιείται για την ανάλυση νέων γονδιωμάτων, μεταγονδιωμάτων, ανίχνευση δομικών ή λειτουργικών τομέων σε άγνωστες πρωτεΐνες και για πειράματα σε συγκεκριμένες πρωτεΐνες και συστήματα. Η PFAM κατατάσσει σε ομάδες συγγενικές καταχωρήσεις, γνωστές ως clans (ομάδες). Ένα clan είναι μια συλλογή από καταχωρήσεις του Pfam που σχετίζονται μεταξύ τους βάσει ομοιότητας στην αλληλουχία, στη δομή ή στο προφίλ-HMM. Τα δεδομένα που εμφανίζονται στη βάση δεδομένων προέρχονται από τις πρωτεϊνικές αναφορές (reference proteomes) του UniProtKB. Το Pfam είναι προσβάσιμο μέσω της ιστοσελίδας https://pfam.xfam.org. Σύμφωνα με την έκδοση Pfam 33.1, που κυκλοφόρησε τον Μάιο του 2020, η Pfam περιλαμβάνει 18.259 οικογένειες πρωτεϊνών, 635 clans και 77,0% των πρωτεϊνικών αλληλουχιών στη UniProtKB έχουν τουλάχιστον μία αντιστοιχία με καταχώρηση της Pfam (sequence coverage), 53,2% των υπολειμμάτων (residues) των αλληλουχιών βρίσκονται εντός κάποιας καταχώρησης της Pfam (residue coverage) [14].

Κάθε οικογένεια της Pfam έχει μια αρχική στοίχιση (seed alignment) που περιλαμβάνει ένα αντιπροσωπευτικό σύνολο αλληλουχιών για την καταχώρηση. Ένα προφίλ κρυφού Μαρκοβιανού μοντέλου (HMM) δημιουργείται αυτόματα από αυτή την αρχική στοίχιση και γίνεται αναζήτηση σε μια βάση δεδομένων ακολουθιών που ονομάζεται pfamseq, χρησιμοποιώντας το λογισμικό HMMER (http://hmmer.org/). Όλες οι περιοχές αλληλουχιών που πληρούν το καθορισμένο όριο (gathering threshold) για την οικογένεια στοιχίζονται στο προφίλ HMM για να δημιουργηθεί η πλήρης στοίχιση. Κατά τη ένταξης μιας νέας εγγραφής στο Pfam, γίνεται επαναλαμβανόμενη αναζήτηση στη βάση pfamseq με σκοπό τον εντοπισμό μακρινών εξελικτικά συγγενών πρωτεϊνών.

Οι εγγραφές διαμορφώνονται με τέτοιο τρόπο ώστε να μην επικαλύπτονται μεταξύ τους, πράγμα που σημαίνει πως ένα συγκεκριμένο τμήμα μιας αλληλουχίας δεν πρέπει να συσχετίζεται με περισσότερες από μία οικογένειες πρωτεϊνών. Αυτός ο κανόνας είναι ένας έλεγχος ποιότητας, που συμβάλλει στη μείωση των ψευδώς θετικών αντιστοιχιών. Από την έκδοση 28.0 του Pfam και μετά, ο κανόνας αυτός έχει μερικώς χαλαρώσει, επιτρέποντας περιορισμένες επικαλύψεις, ώστε να μειωθεί ο χρόνος που απαιτείται για την επίλυση αυτών των συγκρούσεων σε κάθε ενημέρωση της βάσης [14].



Εικόνα 1.7 Στην ιστοσελίδα της InterPro, φαίνεται ότι η βάση Pfam εντόπισε στην πρωτεΐνη C7NEA8 το λειτουργικό domain Peptidase S8/S53 (IPR036852) στις θέσεις 110–366.

1.3.5 Ompdb

Η OMPdb (Outer Membrane Protein database) είναι μία άλλη βάση δεδομένων εξειδικευμένη και ιδιαίτερα εμπεριστατωμένη που περιλαμβάνει πληροφορίες για β-βαρελοειδείς διαμεμβρανικές πρωτεΐνες της εξωτερικής μεμβράνης Gram-αρνητικών βακτηρίων. Η δυσκολία λήψης κρυστάλλων κατάλληλων για μελέτες υψηλής ανάλυσης των εξωτερικών μεμβρανικών πρωτεϊνών έχει οδηγήσει στην ελλιπή μελέτη τους στη βάση δεδομένων Protein Data Bank (PDB), όπου αποτελούν από τις λιγότερες οικογένειες πρωτεϊνών με γνωστή τρισδιάστατη δομή. Ο ανεπαρκής σχολιασμός και ταξινόμηση που παρατηρείται στις δημόσιες βάσεις δεδομένων, καθώς και η αυξανόμενη βιολογική σημασία των β-βαρελιών επιβάλλει την ανάγκη για περαιτέρω μελέτες και ακριβή συλλογή δεδομένων σχετικά με εκείνες [15].

Η βάση δεδομένων περιλαμβάνει πρωτεΐνες, οι οποίες είναι ταξινομημένες σε 154 οικογένειες με βάση δομικά και λειτουργικά κριτήρια. Η βάση είναι ελεύθερα προσβάσιμη για ακαδημαϊκούς χρήστες στη διεύθυνση: http://bioinformatics.biol.uoa.gr/OMPdb και αναμένεται να αποτελέσει σημαντικό εργαλείο για τη γονιδιωματική ανάλυση, τη συγκριτική γονιδιωματική και την ανάπτυξη αλγορίθμων πρόβλεψης για τις β-βαρελοειδείς διαμεμβρανικές πρωτεΐνες [15].

Η OMPdb ακολουθεί τη τεχνική σχολιασμού (annotation) της βάσης δεδομένων Pfam, ωστόσο περιλαμβάνει καινοτόμες οικογένειες πρωτεϊνών που δεν είχαν προηγουμένως καταχωρηθεί ή χαρακτηριστεί σε άλλες δημόσιες βάσεις. Η κάθε οικογένεια β βαρελιών εκπροσωπείται από ένα μοναδικό profile Hidden Markov Model (pHMM), το οποίο περιγράφει τον δομικό και λειτουργικό ρόλο της αντίστοιχης περιοχής. Επιπλέον, παρέχονται διασυνδέσεις με άλλες βάσεις δεδομένων, βιβλιογραφικές παραπομπές και λειτουργικές σημειώσεις, όπως η παρουσία διαμεμβρανικών τμημάτων ή σημάτων στόχευσης. Η βάση δεδομένων, η συλλογή από προφίλ Κρυφών Μαρκοβιανών Μοντέλων (Hidden Markov Models - HMMs), όταν χρησιμοποιείται σε συνδυασμό με τον αλγόριθμο PRED-TMBB2, μπορεί να λειτουργήσει ως ένα ισχυρό εργαλείο για τη διάκριση και την ταξινόμηση νέων β-βαρελοειδών πρωτεϊνών, καθώς και για αναλύσεις ολόκληρων πρωτεωμάτων [15].

Μέσω της ιστοσελίδας της OMPdb, ο χρήστης μπορεί να πλοηγηθεί στα δεδομένα, να πραγματοποιήσουν προηγμένες αναζητήσεις, καθώς και να εκτελέσουν ευθυγραμμίσεις τύπου BLAST και αναζητήσεις δομικών περιοχών με χρήση HMM.

2. Υλικά και Μέθοδοι

2.1 Συλλογή δεδομένων Pfam και OMP domains

Σχεδιάστηκε και υλοποιήθηκε αυτοματοποιημένη ροή επεξεργασίας και ανάλυσης πρωτεϊνικών αλληλουχιών από το σύνολο των gram negative βακτηρίων με σκοπό την ανίχνευση δομικών περιοχών (domains) από τις βάσεις δεδομένων Pfam και Ompdb μέσω του εργαλείου βασισμένου σε προφίλ κρυφών Μαρκοβιανών Μοντέλων (HMMs), hmmscan. Το εργαλείο hmmscan είναι ένα από τα βασικά εργαλεία του HMMER . Το HMMER είναι ένα λογισμικό για την αναζήτηση ομοιοτήτων σε πρωτεϊνικές αλληλουχίες, χρησιμοποιώντας κρυφά μοντέλα Markov (Hidden Markov Models - HMMs).

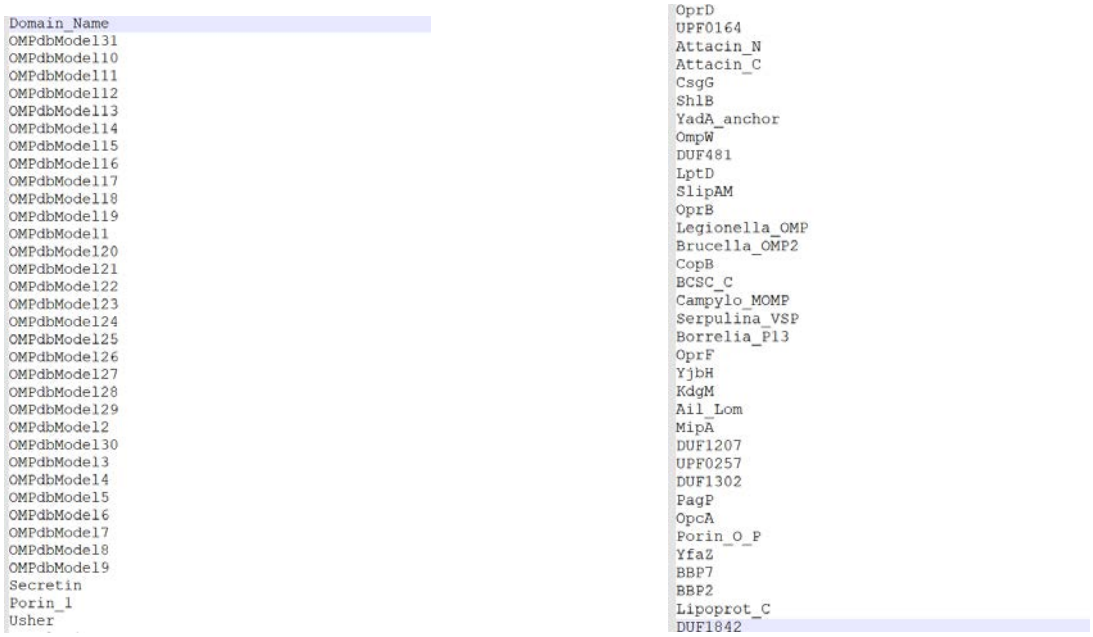
- hmmscan: Πρόγραμμα με το οποίο πραγματοποιούνται αναζητήσεις μίας ή περισσότερων αλληλουχιών έναντι μιας βάσης δεδομένων από μοντέλα HMM.

Για κάθε αλληλουχία που περιέχεται στο αρχείο seqfile, η εντολή χρησιμοποιεί αυτήν την αλληλουχία ως query και τη συγκρίνει με τη βάση δεδομένων προφίλ HMMs (hmmdb), εμφανίζοντας μια κατάταξη των προφίλ με τις πιο σημαντικές αντιστοιχίσεις (matches) για την κάθε αλληλουχία.

- Το seqfile μπορεί να περιέχει περισσότερες από μία αλληλουχίες, και η αναζήτηση θα πραγματοποιηθεί ξεχωριστά για κάθε μία, με στόχο να βρεθούν τα καταλληλότερα HMM προφίλ στη βάση.
- Η βάση δεδομένων hmmdb πρέπει πρώτα να υποβληθεί σε επεξεργασία με την εντολή hmmpress πριν μπορέσει να χρησιμοποιηθεί για αναζήτηση με το hmmscan. Η εντολή hmmpress δημιουργεί τέσσερα δυαδικά αρχεία με καταλήξεις .h3f, .h3i, .h3m, .h3p.[16].

Αρχικά, ξεκινήσαμε συλλέγοντας σε έναν φάκελο, το σύνολο των 5004 γονιδιωμάτων που έχουν βρεθεί στα Gram-negative βακτήρια. Για κάθε οργανισμό που ανήκει στη κατηγορία των gram negative βακτηρίων υπήρχε ένα αρχείο σε μορφή FASTA με όλες τις πρωτεΐνες που συναντώνται στον κάθε οργανισμό. Πάνω σε αυτά τα αρχεία, στην συνέχεια τρέξαμε το εργαλείο HMMscan, ένα εργαλείο που σε συνδυασμό με μια βάση δεδομένων με μοντέλα Hidden Markov, μας βοηθάει να αντλήσουμε πληροφορίες για τα δομικά domains των πρωτεϊνών. Για την αναγνώριση των domains χρησιμοποιήσαμε δύο βάσεις δεδομένων, την PFAM και την OMPdb. Ωστόσο, επειδή η PFAM περιλαμβάνει ήδη κάποια β-βαρέλια τα οποία συναντώνται και στην OMPdb, για να αποφύγουμε επανάληψη εύρεσης των ίδιων OMP domains στην ανάλυση, αφαιρέσαμε όλα τα β-βαρέλια που περιέχονται στην PFAM. Η διαδικασία έγινε ως εξής: Από το αρχείο OMPdb_2022_11_17.hmm (είναι μια βάση δεδομένων με μοντέλα Hidden Markov (HMMs) που χρησιμοποιούνται για την αναγνώριση και χαρακτηρισμό για OMP domains), εξήγαμε όλους τους τίτλους των OMP domains σε μία λίστα.

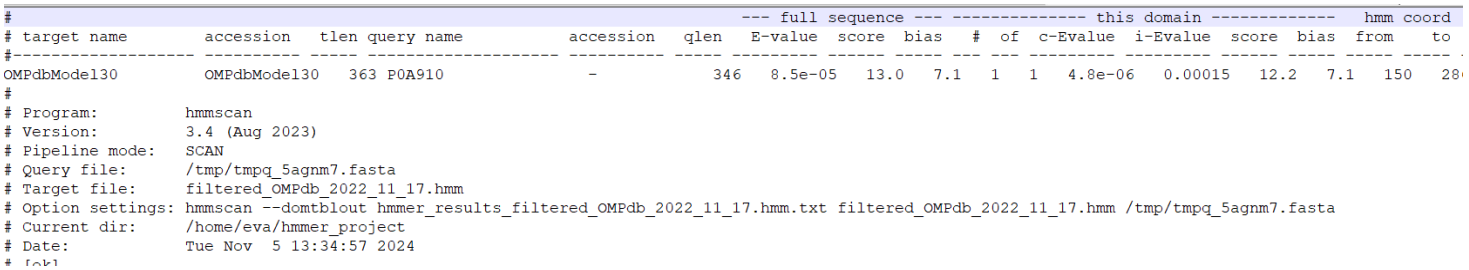
Στη συνέχεια, χρησιμοποιήσαμε αυτή τη λίστα για να φιλτράρουμε το αρχείο Pfam.hmm, αφαιρώντας τις καταχωρήσεις που αντιστοιχούσαν στα domains με τους συγκεκριμένους τίτλους. Έτσι, διασφαλίστηκε ότι όλα τα β-βαρέλια domains θα εντοπίζονται μόνο από μία από τις δύο βάσεις και συγκεκριμένα από την Ompdb.



```
Domain_Name
OMPdbModel31
OMPdbModel10
OMPdbModel11
OMPdbModel12
OMPdbModel13
OMPdbModel14
OMPdbModel15
OMPdbModel16
OMPdbModel17
OMPdbModel18
OMPdbModel19
OMPdbModel1
OMPdbModel20
OMPdbModel21
OMPdbModel22
OMPdbModel23
OMPdbModel24
OMPdbModel25
OMPdbModel26
OMPdbModel27
OMPdbModel28
OMPdbModel29
OMPdbModel2
OMPdbModel30
OMPdbModel3
OMPdbModel4
OMPdbModel5
OMPdbModel6
OMPdbModel7
OMPdbModel8
OMPdbModel9
Secretin
Porin_1
Usher
OprD
UPF0164
Attacin_N
Attacin_C
CsgG
ShlB
YadA_anchor
OmpW
DUF481
LptD
SlipAM
OprB
Legionella_OMP
Brucella_OMP2
CopB
BCSC_C
Campylo_MOMP
Serpulina_VSP
Borrelia_P13
OprF
YjbH
KdgM
Ail_Lom
MipA
DUF1207
UPF0257
DUF1302
PagP
OpcA
Porin_O_P
YfaZ
BBP7
BBP2
Lipoprot_C
DUF1842
```

Εικόνα 2.1 Λίστα με τα 154 omp domains της OMPdb_2022_11_17.hmm.

Αφού ολοκληρώθηκε το φιλτράρισμα της Pfam.hmm, τρέξαμε το HMMscan σε όλα τα FASTA αρχεία, ξεχωριστά για καθεμία από τις δύο βάσεις.



```
#                                                                            --- full sequence ---  -------------- this domain -------------   hmm coord
# target name        accession   tlen query name           accession   qlen   E-value  score  bias   #  of  c-Evalue  i-Evalue  score  bias  from    to
#------------------- ---------- ----- -------------------- ---------- ----- --------- ------ ----- --- --- --------- --------- ------ ----- ----- -----
OMPdbModel30         OMPdbModel30   363 P0A910               -            346   8.5e-05   13.0   7.1   1   1   4.8e-06   0.00015   12.2   7.1   150   28
#
# Program:         hmmscan
# Version:         3.4 (Aug 2023)
# Pipeline mode:   SCAN
# Query file:      /tmp/tmpq_5agnm7.fasta
# Target file:     filtered_OMPdb_2022_11_17.hmm
# Option settings: hmmscan --domtblout hmmer_results_filtered_OMPdb_2022_11_17.hmm.txt filtered_OMPdb_2022_11_17.hmm /tmp/tmpq_5agnm7.fasta
# Current dir:     /home/eva/hmmer_project
# Date:            Tue Nov  5 13:34:57 2024
# [ok]
```

Εικόνα 2.2 Αποτέλεσμα της HMMscan με βάση OMPdb_2022_11_17.hmm για τη πρωτεΐνη P0A910.

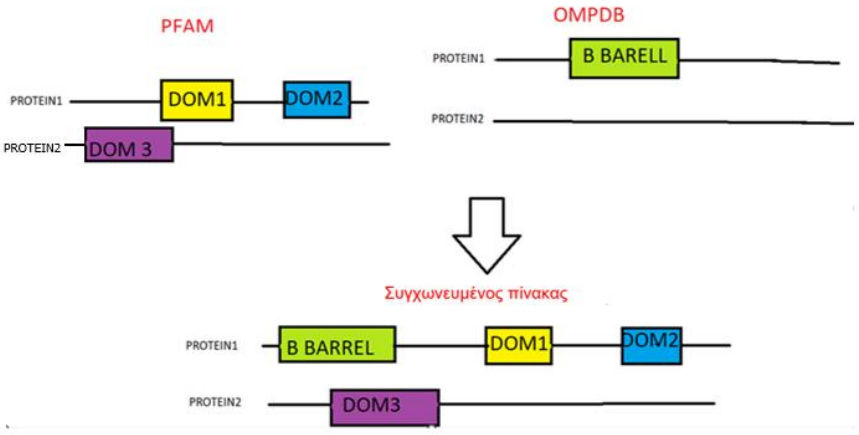
Για παράδειγμα, το εργαλείο HMMscan εντόπισε ότι η πρωτεΐνη με κωδικό P0A910 παρουσιάζει σημαντική ομοιότητα με το μοντέλο OMPdbModel30, που ανήκει σε μία οικογένεια εξωτερικών μεμβρανικών πρωτεϊνών με E-value=8.5e-05 στην περιοχή 150 – 280.

Από τα αποτελέσματα κρατήσαμε μόνο τις περιπτώσεις όπου το alignment score ξεπερνούσε την τιμή του TC (trusted cutoff) για το εκάστοτε domain. Ο όρος score αναφέρεται στο alignment score που

προκύπτει από την αντιστοίχιση της πρωτεΐνης με ένα προφίλ HMM και φανερώνει πόσο καλά ευθυγραμμίζεται η ακολουθία με το μοντέλο του domain. Από την άλλη, το TC (trusted cutoff) είναι μια τιμή που ορίζεται μέσα στο αρχείο HMM για κάθε domain και καθορίζει το ελάχιστο score που θεωρείται στατιστικά σημαντικό, ώστε να αναγνωριστεί αυτό το domain σε μία πρωτεΐνη. Αν το alignment score είναι μεγαλύτερο από το TC, θεωρείται αξιόπιστο ταίριασμα. Για κάθε έγκυρο domain καταγράψαμε τον κωδικό της πρωτεΐνης (protein_id), καθώς και τις θέσεις έναρξης και λήξης του domain (domain_start και domain_end) και τα e-value αυτών.

Στη συνέχεια, τα αποτελέσματα από το HMMscan για κάθε βάση δεδομένων οργανώθηκαν σε δύο ξεχωριστούς πίνακες (dataframes). Ένας πίνακας περιείχε τα domains που προέκυψαν από την PFAM και ένας δεύτερος τα domains από την OMPdb. Κάθε πίνακας περιλάμβανε τις πληροφορίες: την ταυτότητα της πρωτεΐνης (protein_id), το συνολικό μήκος της πρωτεΐνης, τη λίστα των domains που βρέθηκαν, τις θέσεις των domains (έναρξη – λήξη), τα μήκη των domains και τα e-value αυτών.

Ακολούθως, οι δύο πίνακες συγχωνεύτηκαν σε έναν ενιαίο πίνακα (combined table), ευθυγραμμίζοντας τις πληροφορίες με βάση το protein_id, δηλαδή ενώνοντας τα δεδομένα που αφορούν την ίδια πρωτεΐνη.



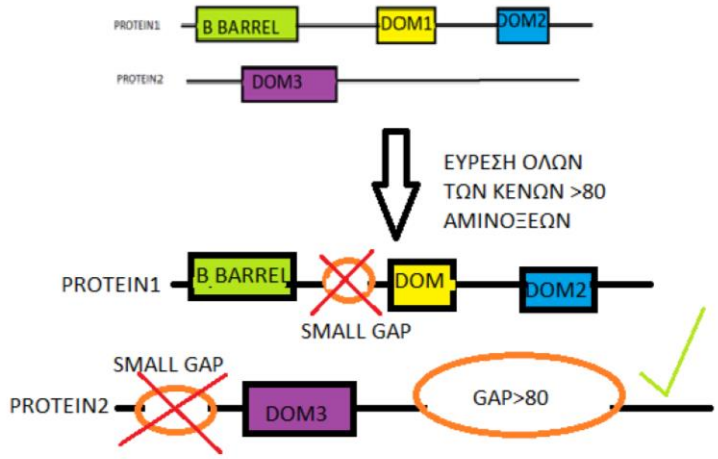
Εικόνα 2.3 Απεικόνιση της συγχώνευσης αποτελεσμάτων από τις βάσεις δεδομένων Pfam και OMPdb.

Έτσι δημιουργήθηκε ένας ολοκληρωμένος πίνακας, με τα domains ανά πρωτεΐνη, προερχόμενα και από τις δύο βάσεις δεδομένων.

Protein ID	Length	PFAM Domains	OMPdb Domains	PFAM Domain Lengths	OMPdb Domain Lengths	PFAM E-values	OMPdb E-values
P12345	402	PF00005,PF01389	OMP34,OMP12	120,80	110,95	1e45, 2e10	5e-20, 3e-15

Πινακας 2.1 Μορφή συγχωνευμένου πίνακα

Στη συνέχεια υπολογίσαμε τις κενές περιοχές ανάμεσα στα domains των πρωτεϊνών, κρατώντας μόνο όσες ήταν μεγαλύτερες από 80 αμινοξέα. Επιλέξαμε αυτό το μήκος διότι απώτερος σκοπός μας είναι να βρούμε αν στις κενές αυτές περιοχές υπάρχει κάποιο μη εντοπισμένο β-βαρέλι και αυτό συνήθως δεν εμφανίζεται με μήκος μικρότερο των 80 αμινοξέων.

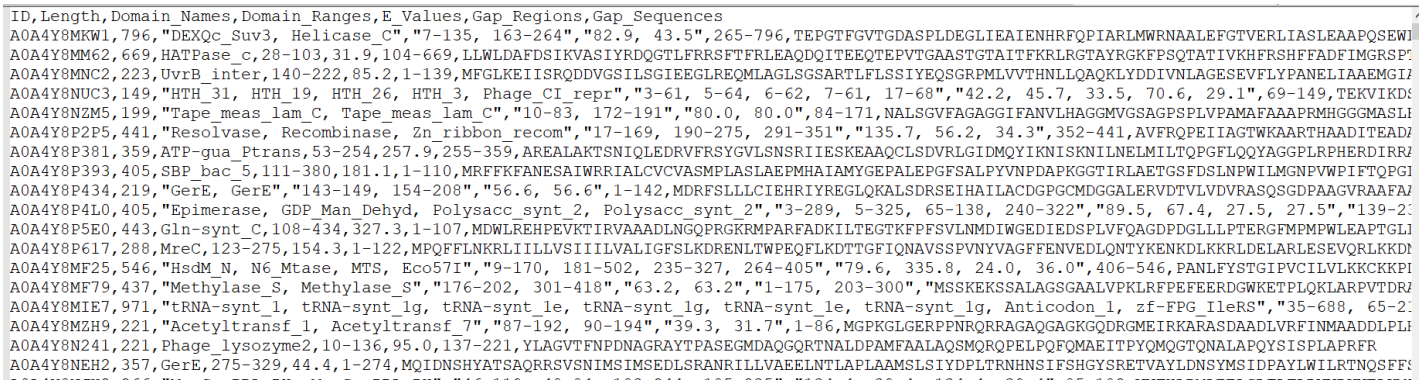


Εικόνα 2.4 Απεικόνιση της διαδικασίας εντοπισμού κενών περιοχών >80 αμινοξέων ανάμεσα σε όλα τα domains.

Άρα ο τελικός διαμορφωμένος πίνακας έχει την εξής παρακάτω μορφή:

Protein ID	Length	PFAM Domains	OMPdb Domains	PFAM Domain Lengths	OMPdb Domain Lengths	PFAM domains E-values	OMPdb domainsE-values	Gap Regions	Gap Sequenes
P1235	402	PF00005 , PF01389	OMP001, OMP002	120, 80	110, 95	1e-45, 2e-10	5e-20, 3e-15	85-170, 189-340	ALAK...TYR NQ, ADKV...RLM N

Πινακας 2.2 Μορφή τελικού συγχωνευμένου πίνακα

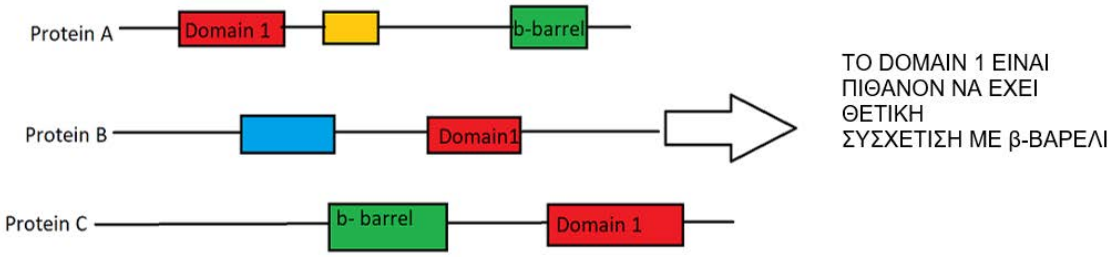


```
ID,Length,Domain_Names,Domain_Ranges,E_Values,Gap_Regions,Gap_Sequences
A0A4Y8MKW1,796,"DEXQc_Suv3, Helicase_C","7-135, 163-264","82.9, 43.5",265-796,TEPGTFGVTGDASPLDEGLIEAIENHRFQPIARLMWRNAALEFGTVERLIASLEAAPQSEWI
A0A4Y8MM62,669,HATPase_c,28-103,31.9,104-669,LLWLDAFDSIKVASIYRDQGTLFRRSFTFRLEAQDQITEEQTEPVTGAASTGTAITFKRLRGTAYRGKFPSQTATIVKHFRSHFFADFIMGRSP
A0A4Y8MNC2,223,UvrB_inter,140-222,85.2,1-139,MFGLKEIISRQDDVGSILSGIEEGLREQMLAGLSGSARTLFLSSIYEQSGRPMLVVTHNLLQAQKLYDDIVNLAGESEVFLYPANELIAAEMGIA
A0A4Y8NUC3,149,"HTH_31, HTH_19, HTH_26, HTH_3, Phage_CI_repr","3-61, 5-64, 6-62, 7-61, 17-68","42.2, 45.7, 33.5, 70.6, 29.1",69-149,TEKVIKDS
A0A4Y8NZM5,199,"Tape_meas_lam_C, Tape_meas_lam_C","10-83, 172-191","80.0, 80.0",84-171,NALSGVFAGAGGIFANVLHAGGMVGSAGPSPLVPAMAFAAAPRMHGGGMASLI
A0A4Y8P2P5,441,"Resolvase, Recombinase, Zn_ribbon_recom","17-169, 190-275, 291-351","135.7, 56.2, 34.3",352-441,AVFRQPEIIAGTWKAARTHAADITEADA
A0A4Y8P381,359,ATP-gua_Ptrans,53-254,257.9,255-359,AREALAKTSNIQLEDRVFRSYGVLSNSRIIESKEAAQCLSDVRLGIDMQYIKNISKNILNELMILTQPGFLQQYAGGPLRPHERDIRRA
A0A4Y8P393,405,SBP_bac_5,111-380,181.1,1-110,MRFFKFANESAIWRRIALCVCVASMPLASLAEPMHAIAMYGEPALEPGFSALPYVNPDAPKGGTIRLAETGSFDSLNPWILMGNPVWPIFTQPGI
A0A4Y8P434,219,"GerE, GerE","143-149, 154-208","56.6, 56.6",1-142,MDRFSLLLCIEHRIYREGLQKALSDRSEIHAILACDGPGCMDGGALERVDTVLVDVRASQSGDPAAGVRAAFAA
A0A4Y8P4L0,405,"Epimerase, GDP_Man_Dehyd, Polysacc_synt_2, Polysacc_synt_2","3-289, 5-325, 65-138, 240-322","89.5, 67.4, 27.5, 27.5","139-2
A0A4Y8P5E0,443,Gln-synt_C,108-434,327.3,1-107,MDWLREHPEVKTIRVAAADLNGQPRGKRMPARFADKILTEGTKFPFSVLNMDIWGEDIEDSPLVFQAGDPDGLLLPTERGFMPMPWLEAPTGLI
A0A4Y8P617,288,MreC,123-275,154.3,1-122,MPQFFLNKRLIILLVSIIILVALIGFSLKDRENLTWPEQFLKDTTGFIQNAVSSPVNYVAGFFENVEDLQNTYKENKDLKKRLDELARLESEVQRLKKDN
A0A4Y8MF25,546,"HsdM_N, N6_Mtase, MTS, Eco57I","9-170, 181-502, 235-327, 264-405","79.6, 335.8, 24.0, 36.0",406-546,PANLFYSTGIPVCILVLKKCKKPI
A0A4Y8MF79,437,"Methylase_S, Methylase_S","176-202, 301-418","63.2, 63.2","1-175, 203-300","MSSKEKSSALAGSGAALVPKLRFPEFEERDGWKETPLQKLARPVTDRA
A0A4Y8MIE7,971,"tRNA-synt_1, tRNA-synt_1g, tRNA-synt_1e, tRNA-synt_1g, tRNA-synt_1e, tRNA-synt_1g, Anticodon_1, zf-FPG_IleRS","35-688, 65-2
A0A4Y8MZH9,221,"Acetyltransf_1, Acetyltransf_7","87-192, 90-194","39.3, 31.7",1-86,MGPKGLGERPPNRQRRAGAQGAGKGQDRGMEIRKARASDAADLVRFINMAADDLPLH
A0A4Y8N241,221,Phage_lysozyme2,10-136,95.0,137-221,YLAGVTFNPDNAGRAYTPASEGMDAQGQRTNALDPAMFAALAQSMQRQPELPQFQMAEITPYQMQGTQNALAPQYSISPLAPRFR
A0A4Y8NEH2,357,GerE,275-329,44.4,1-274,MQIDNSHYATSAQRRSVSNIMSIMSEDLSRANRILLVAEELNTLAPLAAMSLSIYDPLTRNHNSIFSHGYSRETVAYLDNSYMSIDPAYLWILRTNQSFFS
```

Εικόνα 2.5 Αρχείο με τον συγχωνευμένο πίνακα

Επειδή το HMMscan είναι υπολογιστικά απαιτητικό και χρονοβόρο, χωρίσαμε τα FASTA των Gram-αρνητικών βακτηρίων σε 4 μέρη. Αυτό μας έδωσε 4 επιμέρους πίνακες με αποτελέσματα για το κάθε μέρος. Αυτός ο διαχωρισμός μάς επέτρεψε να τρέξουμε το HMMscan παράλληλα, μειώνοντας το συνολικό χρόνο εκτέλεσης.

Στόχος είναι να μελετήσουμε πώς συσχετίζεται κάθε Pfam domain με τα OMPdb domains, για κάθε ένα από αυτά τα 4 datasets ξεχωριστά. Η συσχέτιση αυτή εξετάζεται ξεχωριστά για κάθε ένα από τα τέσσερα datasets, ώστε να διατηρηθεί η ακρίβεια και η ανεξαρτησία της πληροφορίας ανά μέρος. Σκοπός αυτού του βήματος είναι να εντοπίσουμε ποια Pfam domains φαίνονται να συνυπάρχουν συστηματικά με B-barrel domains από την OMPdb. Με αυτή τη πληροφορία, μπορούμε να εντοπίσουμε πρωτεΐνες που περιέχουν τα συγκεκριμένα Pfam domains και ταυτόχρονα παρουσιάζουν μεγάλες κενές περιοχές που μπορεί να είναι μη εντοπισμένα β-βαρέλια. Αυτό είναι σημαντικό διότι, άμα αποδείξουμε ότι ένα Pfam domain εμφανίζεται συχνά μαζί με OMPdb βαρέλια, και στη πρωτεΐνη δεν υπάρχει OMPdb domain, μπορεί η κενή περιοχή να περιέχει ένα μη ανιχνευμένο B-barrel. Άρα, οι πρωτεΐνες που περιέχουν αυτά τα Pfam domains και έχουν μεγάλες κενές περιοχές, αποτελούν υποψήφιες για περαιτέρω διερεύνηση.



Εικόνα 2.5 Παράδειγμα πιθανής συσχέτισης του Domain 1 με τη δομή β-barrel.

Για να μελετήσουμε τη συσχέτιση των domains, δημιουργούμε για κάθε Pfam domain έναν 2x2 στατιστικό πίνακα (contingency table), ο οποίος περιλαμβάνει τις τέσσερις πιθανές περιπτώσεις συνύπαρξης μεταξύ του εκάστοτε Pfam domain και οποιουδήποτε domain της OMPdb.

	OMPdb domain **υπάρχει**	OMPdb domain **Δεν υπάρχει**
Pfam **υπάρχει**	Pfam_and_Barrel	Only_Domain
Pfam Δεν υπάρχει	Only_Barrel	No_Domain_No_Barrel

Πινακας 2.3 Μορφή contingency table συσχέτισης ενός pfam domain με ompdb domains

Περιγραφή των τεσσάρων περιπτώσεων:

- Pfam_and_Barrel

  Ο αριθμός των πρωτεϊνών που περιέχουν το συγκεκριμένο Pfam domain και τουλάχιστον ένα OMPdb domain.
- Only_Domain

  Ο αριθμός των πρωτεϊνών που περιέχουν μόνο το συγκεκριμένο Pfam domain, χωρίς κανένα OMPdb domain.
- Only_Barrel

  Ο αριθμός των πρωτεϊνών που δεν περιέχουν το συγκεκριμένο Pfam domain, αλλά περιέχουν τουλάχιστον ένα OMPdb domain.
- No_Domain_No_Barrel

  Ο αριθμός των πρωτεϊνών που δεν περιέχουν ούτε το συγκεκριμένο Pfam domain, ούτε κάποιο OMPdb domain.

FG-GAP_3	B-barrel(+)	B-barrel(-)
PFAM domain (+)	13	5308
PFAM domain (-)	105327	3846104

Πίνακας 2.4 2x2 contingency table συσχέτισης του pfam domain FG-GAP_3 με ompdb domain.

Συνδυάζουμε τα τέσσερα αρχεία coappearanceX.csv που περιέχουν πληροφορίες για τη συνύπαρξη Pfam domains και β-barrels σε ένα τελικό αρχείο (coappearance_final.csv), συγκεντρώνοντας όλους τους μετρητές.

```
Pfam_Domain,Pfam_and_Barrel,No_Domain_No_Barrel,Only_Barrel,Only_Domain
DUF2511,0,3851406,105340,6
FG-GAP_3,13,3846104,105327,5308
DUF3431,0,3851410,105340,2
Penicillinase_R,0,3851351,105340,61
DNA_meth_N,0,3851409,105340,3
PIN_8,0,3851319,105340,93
CTNNBL,0,1051066,27253,1
BBE,0,3850679,105340,733
DUF493,0,3851405,105340,7
GlutR_N,0,3850410,105340,1002
SBP_bac_10,0,3850612,105340,800
MDD_C,0,3851367,105340,45
EKR,0,3850400,105340,1012
EVA_Class_A,0,1051066,27253,1
Fer4_8,0,3837772,105340,13640
SapC,0,3851391,105340,21
HRQ1_WHD,0,3851207,105340,205
DsrC,0,3851400,105340,12
DCD,0,3850776,105340,636
Glyco_transf_22,0,3851230,105340,182
CBM_26,0,3851409,105340,3
DUF3783,0,3851408,105340,4
PoNi_N,0,3851390,105340,22
DUF2982,0,3851406,105340,6
YlbD_coat,0,1526376,40402,0
Ntox21,0,3851401,105340,11
BcrAD_BadFG,0,3850824,105340,588
DUF977,0,3851378,105340,34
56B_RHH,0,3851412,105340,0
DUF1919,0,3851406,105340,6
SnoaL_2,0,3850721,105340,691
```

Εικόνα 2.6 Τελικό αρχείο συνύπαρξης β-βαρελιών με Pfam domains

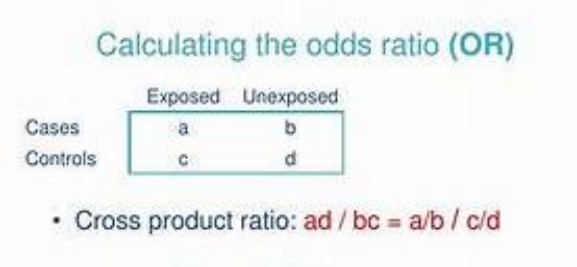
2.2 **Στατιστικά Τεστ για την εύρεση συσχέτισης β-βαρελιών με** Pfam domains

Αφού έχουμε πλέον δημιουργήσει το τελικό αρχείο , το οποίο περιέχει για κάθε Pfam domain τον αριθμό των περιπτώσεων όπου:

- συνυπάρχει με OMPdb domain (Pfam_and_Barrel),
- υπάρχει μόνο του (Only_Domain),
- υπάρχει μόνο OMPdb domain (Only_Barrel),
- δεν υπάρχει ούτε Pfam ούτε OMPdb domain (No_Domain_No_Barrel),

προχωράμε σε στατιστική ανάλυση για να εξετάσουμε ποιες συσχετίσεις είναι στατιστικά σημαντικές .
Για κάθε Pfam domain, εφαρμόζουμε τρεις διαφορετικούς στατιστικούς ελέγχους πάνω στον 2x2 πίνακα συσχετίσεων, με στόχο να αξιολογήσουμε αν υπάρχει σημαντική συσχέτιση μεταξύ του συγκεκριμένου Pfam και της παρουσίας B-barrel από την OMPdb.

Odds Ratio **(Λόγος Πιθανοτήτων)**



Το Odds Ratio είναι ένας δείκτης που μετράει πόσο πιθανότερο είναι ένα Pfam domain να συνυπάρχει με OMPdb domain σε σχέση με το να μην συνυπάρχει. Τιμές:

- > 1: θετική συσχέτιση (το Pfam εμφανίζεται συχνότερα με β-barrel),
- = 1: καμία συσχέτιση,
- < 1: αρνητική συσχέτιση (το Pfam εμφανίζεται λιγότερο συχνά όταν υπάρχει B-barrel).

Στην ανάλυσή μας, κρατάμε μόνο τα Pfam domains με Odds Ratio > 1, γιατί μας ενδιαφέρουν αυτά που συσχετίζονται θετικά με τα β-barrel. Επιπλέον, προσδιορίζεται και το p-value, που δείχνει αν το Odds Ratio είναι στατιστικά σημαντικό ($p < 0.05$).

Fisher's Exact Test

Ο έλεγχος Fisher's Exact εξετάζει αν υπάρχει σημαντική μη τυχαία συσχέτιση μεταξύ των δύο μεταβλητών (Pfam, OMPdb). Είναι ιδιαίτερα κατάλληλος όταν οι τιμές του πίνακα είναι μικρές .

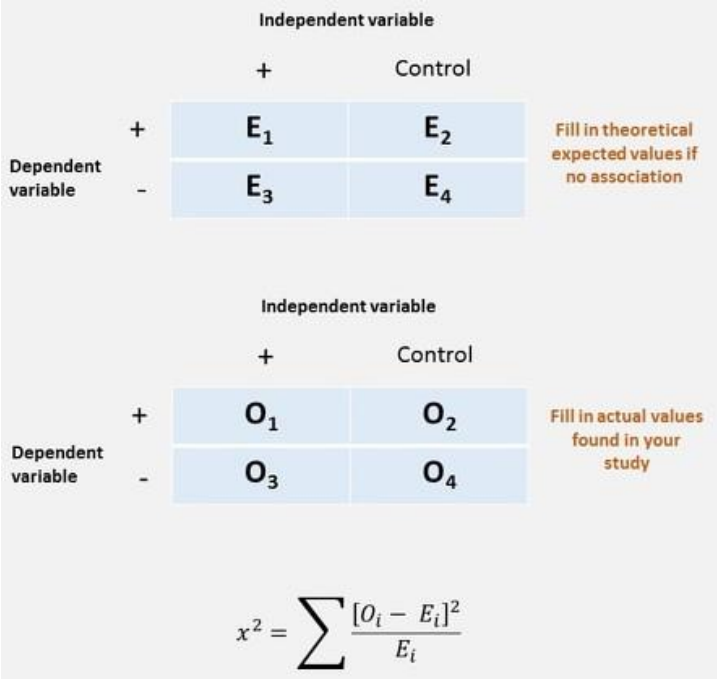
Το αποτέλεσμα είναι ένα p-value. Αν p < 0.05, τότε η συσχέτιση θεωρείται στατιστικά σημαντική.

	case	control	
Allele 1	a	b	a+b
Allele 2	c	d	c+d
	a+c	b+d	n=a+b+c+d

$$P= \left(\frac{a+b}{a}\right)\left(\frac{c+d}{c}\right)\Big/\left(\frac{n}{a+c}\right)=\frac{(a+b)!(c+d)!(a+c)!(b+d)!}{n!a!b!c!d!}$$

Pearson's Chi-Square Test (X^2)

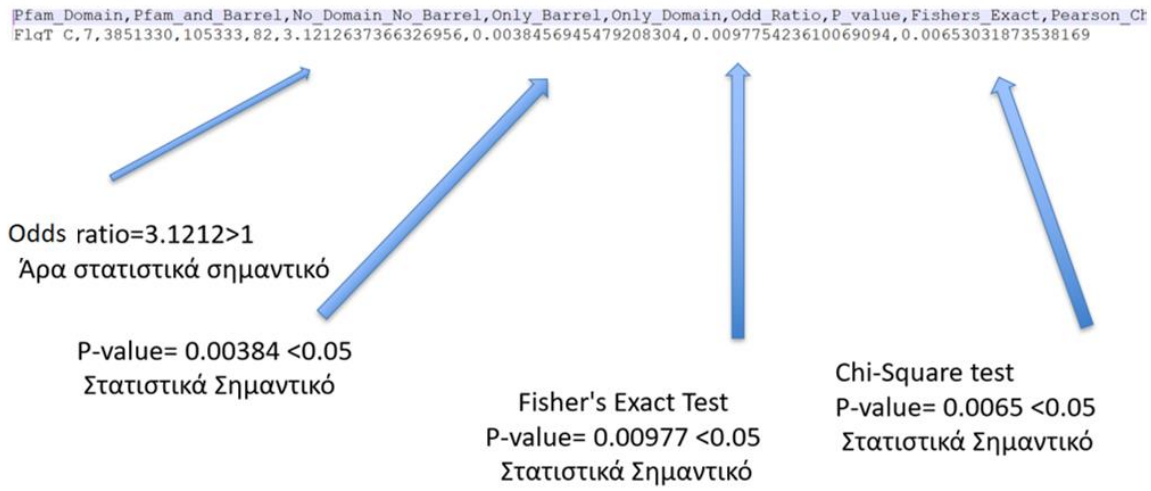
Ο έλεγχος Chi-Square μετράει τη διαφορά μεταξύ των παρατηρούμενων και των αναμενόμενων τιμών στο 2x2 πίνακα. Επίσης δίνει p-value που δείχνει τη σημαντικότητα της διαφοράς.
Αν το p-value είναι < 0.05, τότε απορρίπτουμε την υπόθεση ανεξαρτησίας – δηλαδή, υπάρχει συσχέτιση.



Μετά την εφαρμογή των στατιστικών ελέγχων, δημιουργείται ένα αρχείο που περιέχει μόνο τις στατιστικά σημαντικές εξαρτήσεις με Odds Ratio > 1 και p-value< 0.05, που δείχνουν θετική συσχέτιση με β-barrel.
Αυτά τα φίλτρα μας επιτρέπουν να εστιάσουμε στα Pfam domains που έχουν αποδεδειγμένα και επαναλαμβανόμενα θετική συσχέτιση με β-barrels, και άρα αξίζει να εξετάσουμε περαιτέρω τις πρωτεΐνες που τα περιέχουν.

```
Pfam_Domain,Pfam_and_Barrel,No_Domain_No_Barrel,Only_Barrel,Only_Domain,Odd_Ratio,P_value,Fishers_Exact,Pearson_Chi_Test
FlgT_C,7,3851330,105333,82,3.121263736632696,0.0038456945479208,0.009775423610069,0.0065303187353816
DUF5648,13,3851235,105327,177,2.6855323686861885,0.0005868208027339,0.0018964507145398,0.0007971344943181
PATR,902,3850403,104438,1009,32.95816410990452,0.0,0.0,0.0
Hemopexin,7,3851371,105333,41,6.242593929171897,7.532454249670195e-06,0.0002675863632174,2.8366751091048937e-06
KfrB,1,3851410,105339,2,18.281026020752048,0.0176624220435983,0.0777610995804616,0.131860760316536
PapC_N,497,3851408,104843,4,4564.324218116612,0.0,0.0,0.0
Fib_alpha,2,3851405,105338,7,10.44638619084688,0.0034304943449807,0.0225289779027025,0.0090578215620705
NOGI_N,1,3851411,105339,1,36.562061534664274,0.0109315619348353,0.052536923023906,0.0497167163828178
POTRA_1,450,3850248,104890,1164,14.191037640807798,0.0,0.0,0.0
T2SSN,11,3851391,105329,21,19.153274840872747,2.220446049250313e-15,3.658881513449514e-10,3.144732222199559e-26
Tropomyosin,4,3851407,105336,5,29.25045188729399,4.842948075900466e-07,5.684653208976075e-05,1.4661810762384898e-11
Cep192_D1,1,2681611,72591,3,12.313790047434717,0.0296799428231921,0.1013318472537178,0.218117850959042
FlaC_arch,1,3851409,105339,3,12.18734751611464,0.0303570240289405,0.1023137425859295,0.2216167112685618
FAS_AT_central,1,2220863,60000,2,18.507191666666667,0.0171883608963181,0.0768608919347103,0.1287560890668417
POTRA_2,1943,3851120,103397,292,247.8386657306106,0.0,0.0,0.0
AMIN,1127,3850216,104213,1196,34.8141609381378,0.0,0.0,0.0
VacA,1,2206304,59442,1,37.11692069580431,0.0106027911171817,0.0517826826021621,0.0477190605687597
MACPF_D3,6,3376101,92185,2,109.869317134023 96,8.644443827421355e-09,9.43001758346326e-09,3.209329640731094e-31
CAP_N-CM,1,3851409,105339,3,12.18734751611464,0.0303570240289405,0.1023137425859295,0.2216167112685618
Phage_T7_tail,4,3851360,105336,52,2.812509128298602,0.0462738232778567,0.0619578587094106,0.0953529319825009
Sipho_tail,1,3851408,105339,4,9.140508263795937,0.0478041148429113,0.1262127299164472,0.3080869654833107
TraK,13,3851371,105327,41,11.594049148215998,1.376676550535194e-14,1.3432448828957236e-09,8.632502631042854e-21
HphA_N,6,3851406,105334,6,36.56374959652154,4.5566461714940937e-10,2.864606508544281e-07,1.5427746778267827e-20
FlgO,16,3851286,105324,126,4.643311975202143,7.252906941701553e-09,1.4413725581754314e-06,9.993240812842792e-10
PEGA,527,3849887,104813,1525,12.693272461978736,0.0,0.0,0.0
LpoB,175,3851336,105165,76,84.32660956341789,0.0,1.1409462979829152e-211,0.0
SdrD_B,118,3849921,105222,1491,2.895673751696471,0.0,4.668190400185308e-22,6.185356037221224e-31
STN,4854,3850458,100486,954,194.96570704113856,0.0,0.0,0.0
PT,4,3851387,105336,25,5.850059998481052,0.0010374686869569,0.0070081829083749,0.001650676461282
CFSR,4,3851406,105336,6,24.37537024379129,7.52096074085884e-07,9.273563662120192e-05,2.1189595922134918e-10
Tcp10_C,3,3851398,105337,14,7.834849781449741,0.001213612758206,0.0096994161736033,0.0020375325633396
Secretin_N,3377,3851183,101963,229,556.98965512235,0.0,0.0,0.0
TPR_MLP1_2,2,3851409,105338,3,24.374926427310186,0.0004681573107785,0.0067177932564468,0.0001462561188177
```

Εικόνα 2.7 Το αρχείο με τα Pfam domains που έχουν θετική συσχέτιση με β-barrels.

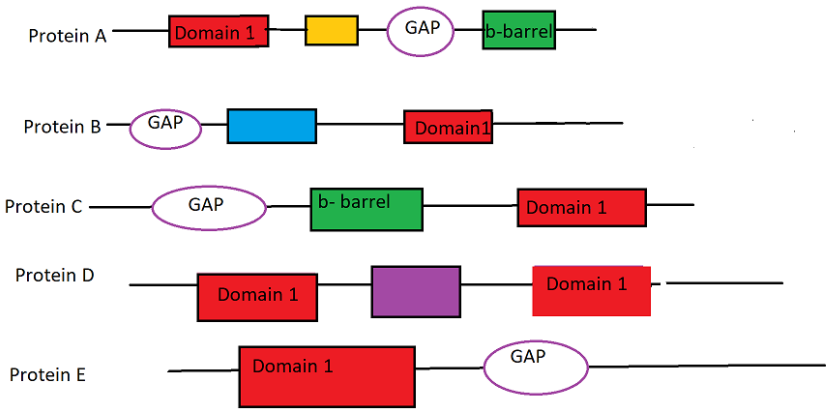


Εικόνα 2.8 Τα αποτελέσματα των στατιστικών τεστ για τη θετική συσχέτιση του domain FlgT_C με β-barrel domains.

Το domain FlgT_C έχει καλή θετική συσχέτιση με την παρουσία β-barrel domains (OR ~3.1 και P-value=0.003) και η στατιστική σημαντικότητα είναι ξεκάθαρη. Αξίζει περαιτέρω διερεύνηση για πιθανές νέες barrel περιοχές.

2.3 **Εύρεση Πρωτεϊνών με Κενές Περιοχές**, **Pfam Domains και χωρίς β**-barrel

Ο στόχος αυτού του βήματος είναι ο εντοπισμός όλων των πρωτεϊνών που περιέχουν κάποιο Pfam Domain που έχει συσχέτιση με β-βαρέλι ,δεν περιέχουν OMPdb domains και επιπλέον έχουν καταγεγραμμένες κενές περιοχές (gap regions) με μήκος μεγαλύτερο των 80 αμινοξέων. Αξιοποιούμε αυτή την πληροφορία για να αναζητήσουμε περιοχές που είναι τυχόν β-βαρέλια τα οποία θα φανερωθούν από την συνύπαρξη τους με τα συσχετιζόμενα μαζί τους Pfam domains. Αυτό το επιτυγχάνουμε, εξάγοντας από το αρχείο το οποίο περιέχει τα Pfam domains που βρέθηκαν να συσχετίζονται θετικά με την παρουσία β-barrel, μόνο τις πρωτεΐνες που περιέχουν τουλάχιστον ένα από τα σημαντικά Pfam domains και δεν περιέχουν κανένα β-barrel domain.

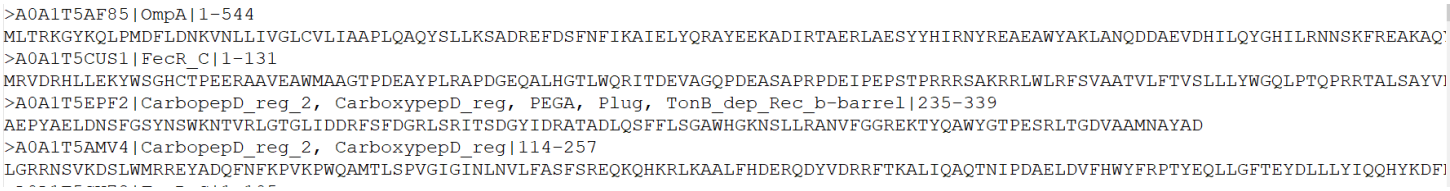


Εικόνα 2.9 Οι πρωτεΐνες B και E, οι οποίες περιέχουν Pfam domains που σχετίζονται με β-βαρέλια, δεν φέρουν γνωστά β-barrel domains και διαθέτουν gap regions μήκους άνω των 80 αμινοξέων, αποτελούν υποψήφιες για περαιτέρω ανάλυση.

Ο διαχωρισμός βασίζεται στην στήλη Domain_Names του συνδυαστικού πίνακα που είχε προκύψει από την εκτέλεση του εργαλείου hmmscan, στα γονιδιώματα των gram negative βακτηρίων, ο οποίος περιλαμβάνει τον κωδικό της κάθε πρωτεΐνης (protein_id), καθώς και τις θέσεις έναρξης και λήξης του domain (domain_start και domain_end) που παρατηρήθηκαν και τα e-value αυτών. Εφαρμόζεται φίλτρο ώστε να παραμείνουν μόνο οι πρωτεΐνες που πληρούν τα παραπάνω κριτήρια. Για κάθε πρωτεΐνη που επιλέχθηκε, εντοπίζονται οι gap περιοχές και οι αντίστοιχες ακολουθίες. Κάθε ακολουθία καταγράφεται σε μορφή FASTA με επικεφαλίδα που περιλαμβάνει:

- Το ID της πρωτεΐνης
- Το όνομα του Pfam domain
- Την τοπολογική θέση του κενού (gap)

Οι αλληλουχίες αποθηκεύονται και το τελικό αποτέλεσμα είναι ένα οργανωμένο αρχείο FASTA (grouped_filtered_results.fasta), στο οποίο καταγράφεται μία εγγραφή για κάθε μοναδικό συνδυασμό (πρωτεΐνη, gap).



Εικόνα 2.10 Το grouped_filtered_results.fasta

2.4 **Πρόβλεψη β-barrel σε κενές περιοχές με το εργαλείο** PRED-TMBB2

Εκτελούμε το εργαλείο PRED-TMBB2, το οποίο πραγματοποιεί πρόγνωση για το αν μια περιοχή ακολουθίας (π.χ. gap) ενδέχεται να σχηματίζει β-barrel δομή. Το εργαλείο εφαρμόζεται στις gap περιοχές των πρωτεϊνών που εντοπίστηκαν στο προηγούμενο στάδιο (οι οποίες φέρουν συσχετιζόμενες Pfam domains αλλά όχι γνωστό β-barrel domain). Το Pred-TMBB2 είναι ένα υπολογιστικό εργαλείο που έχει αναπτυχθεί για την πρόβλεψη της τοπολογίας των β-barrel εξωτερικών μεμβρανικών πρωτεϊνών. Για τις β-barrel πρωτεΐνες, που βρίσκονται κυρίως στις εξωτερικές μεμβράνες Gram-αρνητικών βακτηρίων, μιτοχονδρίων και χλωροπλαστών, οι διαθέσιμες μέθοδοι είναι περιορισμένες και συνήθως δεν έχουν ικανοποιητική ακρίβεια ή ελεύθερη πρόσβαση. Το PRED-TMBB2 είναι ενσωματωμένο μέσα στο πρόγραμμα JUCHMME (https://github.com/pbagos/juchmme) και είναι προσβάσιμο μέσω της διαδικτυακής πλατφόρμας http://195.251.108.230/PRED-TMBB2/ [17]. Η λειτουργία του εργαλείου βασίζεται σε Κρυμμένα Μοντέλα Markov (Hidden Markov Models - HMMs), τα οποία εκπαιδεύτηκαν με βάση την κριτήρια Συνθηκών Μέγιστης Πιθανοφάνειας (Conditional Maximum Likelihood). Το μοντέλο έχει αναπροσαρμοστεί ώστε να συμπεριλαμβάνει 16 εξωτερικές μεμβρανικές πρωτεΐνες γνωστής τρισδιάστατης δομής σε ατομικό επίπεδο, εξασφαλίζοντας υψηλότερη αξιοπιστία και ακρίβεια στις προβλέψεις) [17] .

```
eva@DESKTOP-NB5CHVO:~/hmmer_project$ java -Xmx4g -cp ./bin hmm.Juchmme -a tables/A_TMBB2_TRAINED -e tables/E_TMBB2_TR
AINED -c conf/conf.tmbb -m models/tmbb2.mdel  -f input/grouped_filtered_results.fasta > predtmbb_results.txt & tail -
f predtmbb_results.txt
```

Η παραπάνω εντολή χρησιμοποιείται για την εκτέλεση του εργαλείου PRED-TMBB2 μέσω γραμμής εντολών και την αυτόματη αποθήκευση των αποτελεσμάτων σε αρχείο, ώστε να γίνει επεξεργασία στη συνέχεια. Αναλυτικά εξηγούνται τα στοιχεία της εντολής:

1. java -Xmx4g

   Εκτελεί το εργαλείο μέσω της Java Virtual Machine, δίνοντας μέγιστο όριο μνήμης 4 GB για την εκτέλεση.

2. -cp ./bin

   Ορίζει το classpath, δηλαδή τον φάκελο ./bin όπου βρίσκεται το αρχείο .class του Pred-TMBB

(hmm.Juchmme) που περιέχει τον βασικό κώδικα της εφαρμογής.

3. hmm.Juchmme

   Είναι η Java κλάση που εκτελεί τον πυρήνα του Pred-TMBB, υλοποιώντας το μοντέλο HMM και τις μεθόδους αποκωδικοποίησης.

4. -a tables/A_TMBB2_TRAINED

   Το αρχείο αυτό περιέχει τις πίνακες μεταβάσεων (transition tables) του εκπαιδευμένου HMM μοντέλου για την κατηγορία TMBB2 (β-barrel transmembrane proteins).

5. -e tables/E_TMBB2_TRAINED

   Αυτό το αρχείο περιέχει τους πίνακες εκπομπής (emission tables), δηλαδή τις πιθανότητες εμφάνισης κάθε αμινοξέος σε κάθε κατάσταση του HMM.

6. -c conf/conf.tmbb

   Αρχείο ρυθμίσεων που περιέχει παραμέτρους αποκωδικοποίησης, κατώφλια, τύπο decoding (π.χ. Viterbi),κ.λπ.

7. -m models/tmbb2.mdel

   Το εκπαιδευμένο μοντέλο HMM για β-barrel πρωτεΐνες, περιλαμβάνει τη συνολική δομή του μοντέλου και τη σύνδεσή του με τις παραπάνω παραμέτρους.

8. -f input/grouped_filtered_results.fasta

   Το αρχείο εισόδου σε μορφή FASTA, που περιέχει τις πρωτεϊνικές ακολουθίες στις οποίες θα εφαρμοστεί η πρόβλεψη τοπολογίας.

9. predtmbb_results.txt

   Η έξοδος της εντολής (αποτελέσματα πρόβλεψης) ανακατευθύνεται στο αρχείο predtmbb_results.txt, ώστε να αποθηκευτούν εκεί τα δεδομένα.

10. &

    Εκτελεί την εντολή στο παρασκήνιο (background), επιτρέποντας στον χρήστη να συνεχίσει να χρησιμοποιεί το τερματικό.

11. tail -f predtmbb_results.txt

    Παρακολουθεί σε πραγματικό χρόνο την παραγωγή γραμμών στο αρχείο εξόδου, ώστε να μπορεί ο χρήστης να βλέπει την πρόοδο της πρόβλεψης όσο τρέχει το πρόγραμμα.

```
ID: >C6BZQ2|TPR_19|1-110
SQ: MDTFWNRLPRELAEKFLAGGVGRTHLTSIGYKCIELSKNDPANSELFIKLAFETLISAWCHDPLNLQSAQLQQNLMGQEGPNSPMGKMLSRIVEESRKDDLGAQEKRLRN
CC:     len = 110.0     logodds = -24.006308431467517   maxProb = 0.8144134098274586    (-logprob/lng) = 3.1282764099978193
VS: -1.6848525440412345
VR: 0.6462088927771572
VP: IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
LS: 348.62839515467
LR: 0.8331927167913261
LP: IIIIIIIIIIIIIMMMMMMMMOOOOOOOOOOOOOOOOOOOOOOOMMMMMMMMMMMMMIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
ID: >C6C0S9|TPR_19|1-90
SQ: MSQEMKVLGVYSQNIKTRIGAGTTSQKSDHKTYHYVIRVGYSKYHVQPLNEQDIPSGMITVISEKDFIAQFSPELYYYQRKTLPALKTLQ
CC:     len = 90.0      logodds = -21.638058162149775   maxProb = 0.9931143124709957    (-logprob/lng) = 3.1834561960295265
VS: -0.24304207675095313
VR: 0.9817362798892435
VP: IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIMMMMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOOMMMMMMMMMMMIIIIIIIIIIIII
LS: 290.99243475220857
LR: 0.9817362798892435
LP: IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIMMMMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOOMMMMMMMMMMMIIIIIIIIIIIII
ID: >C6C1P9|TPR_19|189-317
SQ: MSGTEKLNLLGDFTYRNFIFGTRNLTLRSITFKPGQSDSDFDSKSSLHVIAAMMEENTDLRFHIVAYVAGNAELAKARAKGVRDYILSGHPDFAPSRMPLSWFGSAEEVERSGKTFKLDESIQFITVIN
CC:     len = 129.0     logodds = -34.842156444190834   maxProb = 0.9918554872077314    (-logprob/lng) = 3.159201708408543
VS: -1.6248931960136588
VR: 0.9185717476745938
VP: IIIIIMMMMMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOMMMMMMMIIIIIMMMMMMMMMOOOOOOOOOOOOOOOOOMMMMMMMMMIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
LS: 412.31189752564853
LR: 0.9185717476745938
LP: IIIIIMMMMMMMMMOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOMMMMMMMIIIIIMMMMMMMMMOOOOOOOOOOOOOOOOOMMMMMMMMMIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
```

Εικόνα 2.11 Αποτέλεσματα του εργαλείου PRED-TMBB2

Το εργαλείο παρέχει παρακάτω χαρακτηριστικά όπως:

- Το logodds score από το HMM μοντέλο,
- Την κανονικοποιημένη log-πιθανότητα (logprob),
- Την αξιοπιστία της πρόβλεψης (VR – reliability),
- Τον αριθμό μεμβρανικών τμημάτων (transmembrane segments),
- Και το συνολικό μήκος της πρωτεΐνης ή της gap περιοχής (length).

Ωστόσο, η εκτίμηση του κατά πόσο μία περιοχή μπορεί να χαρακτηριστεί β-barrel δεν βασίζεται αποκλειστικά στην έξοδο του PRED-TMBB2. Για την τελική απόφαση, εφαρμόζεται ένας τύπος λογιστικής παλινδρόμησης (logistic regression), που συνδυάζει τα παραπάνω χαρακτηριστικά σε έναν γραμμικό τύπο. Το αποτέλεσμα της εξίσωσης εισάγεται στη λογιστική συνάρτηση, η οποία επιστρέφει μία πιθανότητα p.

Υπολογισμός score (γραμμικός συνδυασμός χαρακτηριστικών):

score = (

0.187989 × logodds

- 9.133431 × logprob

- 12.07592 × reliability

+ 1.270188 × transmembrane

- 71.9362 × (transmembrane / length)

+ 34.00917

)

Υπολογισμός πιθανότητας με λογιστική συνάρτηση:

$$p = \frac{1}{1 + e^{-\text{score}}}$$

Αν η τιμή της πιθανότητας είναι p > 0.25, τότε η περιοχή χαρακτηρίζεται ως πιθανή β-barrel δομή εξωτερικής μεμβράνης.

```
ID,p,beta barrel?
C6BSI9|AMIN|1-157,0.0000,NO
C6BZA5|ChAPs, TPR_19|90-191,0.0000,NO
C6BYA3|TPR_19|1-93,0.0000,NO
C6BZQ2|TPR_19|1-110,0.0035,NO
C6C0S9|TPR_19|1-90,0.0000,NO
C6C1P9|TPR_19|189-317,0.0001,NO
C6BZ42|TPR_19|1-84,0.0064,NO
C6BZ42|TPR_19| 242-326,0.0001,NO
C6BZP7|Peptidase_S8|1-274,0.0004,NO
C6BZP7|Peptidase_S8| 624-1100,0.0006,NO
C6BTU5|TPR_19|94-203,0.0000,NO
C6C1V1|TPR_19|1-120,0.0025,NO
C6C1V1|TPR_19| 242-477,0.0000,NO
C6BXJ8|Patatin|265-404,0.0000,NO
C6C1R0|OmpA|1-294,0.0000,NO
C6C1Y3|AMIN|1-136,0.0000,NO
C6BW93|TPR_19|1-95,0.0000,NO
C6C0P2|TPR_19|1-98,0.0000,NO
C6C1V4|TPR_19|247-362,0.0009,NO
C6BS53|TPR_19|1-150,0.0000,NO
C6BXW9|OmpA|1-253,0.0000,NO
C6BWM1|PEGA|72-334,0.0046,NO
C6C036|TPR_19|178-375,0.0000,NO
C6BUR4|OmpA|44-130,0.0000,NO
C6C0T0|TPR_19|1-94,0.0000,NO
C6BWY4|PilO|338-515,0.0000,NO
C6BZA4|TPR_19|94-179,0.0001,NO
C6BZA4|TPR_19| 323-614,1.0000,YES
```

Εικόνα 2.12 Το αρχείο με τα εκτυπωμένα scores κάθε αλληλουχίας

Άμα κάποια κενή περιοχή βρεθεί με p>0,25 και άρα παρουσιάζει δομή β-βαρελιού, τοτέ γράφουμε στο τέλος της γραμμής 'YES' αλλιώς 'NO'.

2.5 **Ομαδοποίηση Πρωτεϊνικών Αλληλουχιών με CD**-HIT

Μετά την εκτέλεση του εργαλείου πρόγνωσης Pred-TMBB2, καταγράφηκαν όλες οι gap περιοχές που εμφανίζουν χαρακτηριστικά β-barrel δομών μεμβράνης. Παράλληλα, με στόχο την ταξινόμηση και τη καλύτερη κατανόηση της ομολογίας μεταξύ των gap περιοχών, πραγματοποιήθηκε ομαδοποίηση (clustering) των ακολουθιών που περιέχονται στο αρχείο grouped_filtered_results.fasta. Στόχος ήταν να ομαδοποιήσουμε τις κενές περιοχές αυτές και έπειτα να προσθέσουμε τα αποτελέσματα από το PRED-TMBB2. Εάν μέσα σε ένα cluster παρατηρούνται πολλαπλές β-barrel δομές, τότε υπάρχει πιθανότητα να εντοπιστεί κοινό μοτίβο

αμινοξέων.Ένα τέτοιο μοτίβο μπορεί να είναι χαρακτηριστικό για τις β-barrel πρωτεΐνες, και να χρησιμοποιηθεί για να αποκαλύψει τυχόν νέες οικογένειες συγγενών πρωτεϊνών β-βαρελιών .

Το εργαλείο CD-HIT (Cluster Database at High Identity with Tolerance) είναι ένα εργαλείο ταχείας ομαδοποίησης βιολογικών αλληλουχιών, το οποίο εντοπίζει και συγκεντρώνει παρόμοιες αλληλουχίες σε clusters, με βάση προκαθορισμένο ποσοστό ομοιότητας. Η ταχύτητα και η ικανότητά του να διαχειρίζεται τεράστιες βάσεις, όπως η NCBI-nr, το έχουν καταστήσει απαραίτητο εργαλείο σε πολλούς τομείς της βιοπληροφορικής [18-20]. Το CD-HIT είναι προσβάσιμο μέσω της διαδικτυακής ιστοσελίδας https://www.bioinformatics.org/cd-hit/. Λαμβάνει ως είσοδο ένα αρχείο σε μορφή FASTA με αλληλουχίες, εντοπίζει και αφαιρεί πλεονάζουσες (υψηλά όμοιες) αλληλουχίες, κρατώντας μία "αντιπροσωπευτική" από κάθε ομάδα, δημιουργεί clusters, δηλαδή ομάδες παρόμοιων αλληλουχιών, βάσει ενός καθορισμένου ποσοστού ταύτισης (identity threshold). Εφαρμόζει έναν αλγόριθμο "longest sequence first": ξεκινά από τη μεγαλύτερη αλληλουχία και αφαιρεί όσες είναι παρόμοιες (πάνω από το όριο) με αυτή. Χρησιμοποιεί ταχείες ευριστικές μεθόδους για να εντοπίσει περιοχές υψηλής ομοιότητας, αποφεύγοντας ακριβούς αλλά χρονοβόρους ευθυγραμμίσεις (alignments) [18-20].

Τρέξαμε το πρόγραμμα με την παρακάτω εντολή:

```
cd-hit -i grouped_filtered_results.fasta -o clustered_30.fasta -c 0.40 -n 2
```

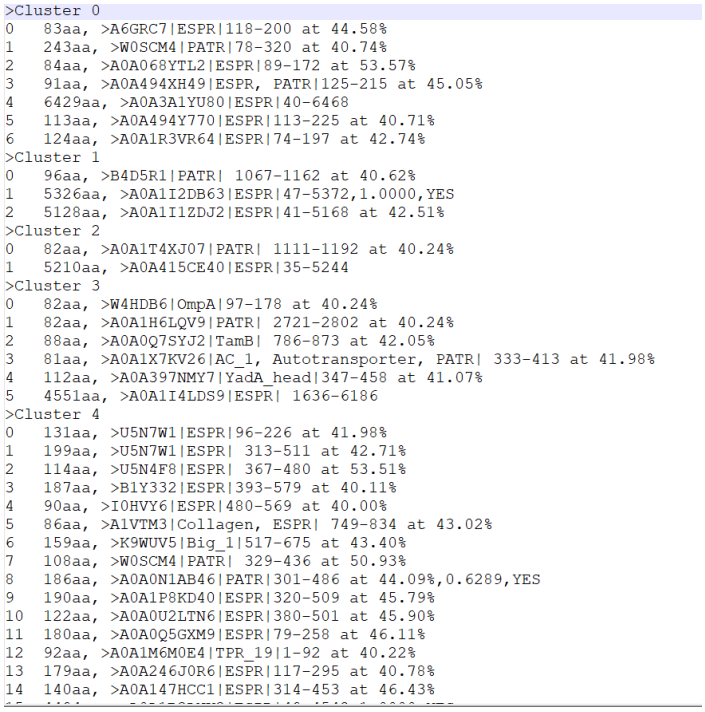
- Όπου :

  -i grouped_filtered_results.fasta: αρχείο εισόδου με τις πρωτεϊνικές αλληλουχίες.

  -o clustered_40.fasta: αρχείο εξόδου με τις αλληλουχίες-αντιπροσώπους κάθε cluster.

  -c 0.40: κατώφλι ομοιότητας 40%. Αλληλουχίες με ≥40% ταύτιση συγκεντρώνονται στο ίδιο cluster.

  -n 2: μέγεθος του "word length" (μικρή τιμή για χαμηλά thresholds όπως 40%).



Εικόνα 2.13 Το αρχείο clustered_40_numbered_with_headers

2.6 **Αντιστοίχιση προβλέψεων β-βαρελιών στις ομάδες** (clusters)

Αφού πραγματοποιήθηκε η ομαδοποίηση των πρωτεϊνικών gap περιοχών σε ομάδες (clusters) με χρήση του εργαλείου CD-HIT, και προσδιορίστηκαν με το εργαλείο PRED-TMBB2 οι περιοχές που εμφανίζουν υψηλή πιθανότητα να αποτελούν β-barrel δομές εξωτερικής μεμβράνης, το επόμενο στάδιο περιλάμβανε τον εμπλουτισμό των clusters με πληροφορία σχετικά με την παρουσία ή μη τέτοιων δομών. Αν μια ομάδα περιλαμβάνει πολλές αλληλουχίες που είναι β-βαρέλια, τότε ίσως αποτελούν μέλη μιας άγνωστης οικογένειας β-βαρελιών.

Η πληροφορία αυτή αντλήθηκε από το αρχείο predtmbb_scores.txt, το οποίο περιέχει την έξοδο του PRED-TMBB2 μετά την εφαρμογή του μοντέλου λογιστικής παλινδρόμησης. Από το αρχείο αυτό επιλέχθηκαν οι γραμμές όπου η τελική πρόβλεψη ήταν "YES" (δηλαδή όταν η πιθανότητα $p > 0.25$), υποδεικνύοντας ότι η ακολουθία παρουσιάζει χαρακτηριστικά β-barrel δομής. Οι σχετικοί τίτλοι των αλληλουχιών, μαζί με τα αντίστοιχα scores, αποθηκεύτηκαν σε λεξικό για γρήγορη αναφορά.

Στη συνέχεια, χρησιμοποιήθηκε το αρχείο clustered_40_numbered_with_headers.clstr, το οποίο περιέχει την έξοδο από το clustering. Για κάθε αλληλουχία εντός ενός cluster, πραγματοποιήθηκε αντιστοίχιση με την

πρόβλεψη του PRED-TMBB2, βάσει του τίτλου της αλληλουχίας (π.χ. >A0A7X1G036|CarboxypepD_reg, SdrD_B|1323-1672). Έτσι, για κάθε cluster προσδιορίστηκε ο συνολικός ο αριθμός των αλληλουχιών που αναγνωρίστηκαν ως β-barrel και το ποσοστό εμφάνισης β-barrel δομών εντός του cluster.
Η πληροφορία αυτή ενσωματώθηκε στον τίτλο κάθε cluster, με τη μορφή: "Cluster 2: 6/10 (60%) WITH B BARRELS", επιτρέποντας άμεσα την εκτίμηση της συνάφειας ενός cluster με τις β-barrel πρωτεΐνες.

Για σκοπούς ποσοτικής και συγκριτικής ανάλυσης, τα αποτελέσματα διαχωρίστηκαν σε τρεις διαφορετικές κατηγορίες, δημιουργώντας ισάριθμα αρχεία εξόδου:

- clusters_over_50.csv: Clusters με ποσοστό β-barrel >50% και τουλάχιστον 5 αλληλουχίες.
- clusters_over_60.csv: Clusters με ποσοστό >60%.
- clusters_over_70.csv: Clusters με ποσοστό >70%.

Κάθε αρχείο περιλαμβάνει πληροφορίες όπως: αριθμός cluster, αριθμός β-barrel αλληλουχιών, συνολικός αριθμός αλληλουχιών, και ποσοστό β-barrel.

```
>Cluster 233 35/49 (71.4%) WITH B BARRELS
0   1180aa, >A3WD51|CarboxypepD_reg| 551-1730,1.0000,YES
1   931aa, >A0A074N198|CarboxypepD_reg|552-1482 at 71.75%,1.0000,YES
2   208aa, >A0A074N198|CarboxypepD_reg| 1507-1714 at 73.08%,0.7472,YES
3   967aa, >F1ZBI3|SdrD_B| 556-1522 at 45.92%,1.0000,YES
4   155aa, >F1ZBI3|SdrD_B| 1559-1713 at 52.90%
5   687aa, >A5P6N4|CarboxypepD_reg| 530-1216 at 63.03%,1.0000,YES
6   439aa, >A5P6N4|CarboxypepD_reg| 1227-1665 at 64.24%,1.0000,YES
7   1148aa, >N9VXT2|SdrD_B| 562-1709 at 44.25%,1.0000,YES
8   519aa, >A0A074M8P1|CarboxypepD_reg| 566-1084 at 77.65%
9   634aa, >A0A074M8P1|CarboxypepD_reg| 1118-1751 at 77.60%,1.0000,YES
10  1151aa, >Q2NAG5|CarboxypepD_reg| 534-1684 at 57.17%,1.0000,YES
11  1142aa, >Q1GTI0|CarboxypepD_reg| 561-1702 at 45.18%,1.0000,YES
12  175aa, >A0A0G9MXW5|CarboxypepD_reg| 582-756 at 47.43%
13  930aa, >A0A0G9MXW5|CarboxypepD_reg| 770-1699 at 63.23%,1.0000,YES
14  710aa, >A0A109LS21|CarboxypepD_reg| 549-1258 at 73.80%,1.0000,YES
15  424aa, >A0A109LS21|CarboxypepD_reg| 1301-1724 at 74.76%,1.0000,YES
16  745aa, >A0A1C7DAK3|CarboxypepD_reg| 552-1296 at 59.33%,1.0000,YES
17  360aa, >A0A1C7DAK3|CarboxypepD_reg| 1334-1693 at 60.83%,1.0000,YES
18  878aa, >A0A0N9UAM8|CarboxypepD_reg| 769-1646 at 47.27%,1.0000,YES
19  561aa, >A0A102D5G3|CarboxypepD_reg|552-1112 at 48.13%
20  277aa, >A0A102D5G3|CarboxypepD_reg| 1445-1721 at 48.74%,0.9018,YES
21  587aa, >A0A0Q7YQC1|CarboxypepD_reg| 550-1136 at 44.63%,0.8389,YES
22  515aa, >A0A0Q7YQC1|CarboxypepD_reg| 1185-1699 at 45.24%,1.0000,YES
23  1143aa, >A0A246K6H7|CarboxypepD_reg| 555-1697 at 44.88%,1.0000,YES
24  926aa, >A0A246JRQ7|CarboxypepD_reg, SdrD_B| 628-1553 at 46.54%
25  1132aa, >A0A220W508|CarboxypepD_reg| 583-1714 at 44.17%,1.0000,YES
26  1138aa, >A0A1B62DK3|CarboxypepD_reg| 531-1668 at 44.02%,1.0000,YES
27  1148aa, >A0A192D7N7|CarboxypepD_reg| 551-1698 at 68.47%,1.0000,YES
28  1167aa, >A0A1L3JEZ6|SdrD_B| 563-1729 at 41.22%,1.0000,YES
29  1152aa, >A0A3D9FCN3|CarboxypepD_reg| 566-1717 at 46.01%,1.0000,YES
30  980aa, >A0A270BMI6|CarboxypepD_reg, SdrD_B| 719-1698 at 48.06%
31  963aa, >A0A396RJQ1|CarboxypepD_reg| 475-1437 at 46.11%,1.0000,YES
32  141aa, >A0A396RJQ1|CarboxypepD_reg| 1459-1599 at 40.43%
33  1125aa, >A0A2A4B388|CarboxypepD_reg| 534-1658 at 48.18%,1.0000,YES
34  1085aa, >A0A369Q5K9|CarboxypepD_reg| 646-1730 at 63.96%,1.0000,YES
35  1134aa, >A0A553WL64|CarboxypepD_reg, SdrD_B| 553-1686 at 43.92%
```

Εικόνα 2.14 Το αρχείο με τα clusters,εμπλουτισμένα με την πληροφορία από το PRED-TMBB2.

Ενσωματώνουμε στα clusters τις αλληλουχίες των κενών περιοχών, που το εργαλείο CD-HIT δεν είχε προσθέσει, κατά την ομαδοποίηση τους. Αυτό το πραγματοποιήσαμε, αντιστοιχίζοντας το όνομα των αλληλουχιών του cluster αρχείου με το όνομα των αλληλουχιών του αρχείου εισόδου στο CD-HIT που περιλάμβανε τις την αλληλουχία αυτών.

Εικόνα 2.15 Το αρχείο με τα clusters, εμπλουτισμένα με την πληροφορία από το PRED-TMBB2 και τις αλληλουχίες των κενών περιοχών.

2.7 **Πολλαπλή Στοίχιση με** Clustal Omega

Στα τρία αρχεία με τις ομαδοποιημένες αλληλουχίες με ποσοστό (β-βαρέλια/συνολικές πρωτεϊνικές αλληλουχίες) μεγαλύτερο από 50%,60% και 70% αντίστοιχα, κάνουμε πολλαπλή στοίχιση, με σκοπό να εντοπίσουμε τυχόν συντηρημένα μοτίβα ή περιοχές (conserved regions) που εμφανίζονται σε όλες ή τις περισσότερες πρωτεΐνες του cluster.

Η πολλαπλή στοίχιση αλληλουχιών αποτελεί θεμελιώδες εργαλείο της βιοπληροφορικής και χρησιμοποιείται ευρέως για τη μελέτη ομοιότητας και εξελικτικών σχέσεων μεταξύ πρωτεϊνών. Στην περίπτωση μας η εφαρμογή του Clustal Omega γίνεται για να ενισχύσουμε την υπόθεση λειτουργικής ή δομικής συγγένειας μεταξύ των πρωτεϊνών και την επιβεβαίωση ταξινόμησης σε ίδια νέα δομική οικογένεια πρωτεϊνικών περιοχών. Η πρόσβαση στο Clustal Omega γίνεται στην ιστοσελίδα http://www.clustal.org/omega/.
Το ClustalΩ (Omega) είναι το πιο πρόσφατο εργαλείο της οικογένειας Clustal για MSA. Χρησιμοποιεί προοδευτική ευθυγράμμιση και κατασκευή δέντρου (guide tree) για να δημιουργήσει στοίχιση. Είναι ιδανικό για μεγάλα datasets και υψηλή ακρίβεια στη στοίχιση [21-23].

Αρχικά φτιάχνουμε ένα ξεχωριστό fasta αρχείο για κάθε cluster στο οποίο θα τρέξουμε το Clustal Omega. Στη συνέχεια γίνεται αυτόματη εκτέλεση του Clustal Omega μέσω subprocess για κάθε αρχείο FASTA που

δημιουργήθηκε. Το αποτέλεσμα για κάθε cluster είναι ένα αρχείο στοίχισης .aln, το οποίο περιλαμβάνει την ευθυγράμμιση όλων των αλληλουχιών του cluster.

```
>E1JXD6|Plug|166-664
LAAVDMTPKRMDHEGFSSRVTAQYGSYNTSQETAETGGKIGRFDYYAAQSYRYSSGSRAMSDGMLTSYYGRLGYGLTDNWNAHATVIRTDNYAMDPGPEGQPEERDGKYVTRDWHMVATLAN
>W0JAA3|Plug|163-675
AFTAVNITPRQKRTEGLAADINVAVGSDRTRVEKMSAGGKTGPFDFYAGQSYRRADGDRTNSGGELQDYFLRLGYALNENWSLSFFGDRTDNHAEDPGAENAPYHMGDYQTDDWLSVLTLAN
>Q72FW3|Plug|161-649
AFSAIDVIPKRMTEDGYRTRLFTSYGSYNTFTQGVEHGGKQGGVDYYAGQSVRLSDGHREHSDGQIENYFGRLGYDLYENWNVSWFGNVTRNVARDPGSSLSTGNDGRFYTDDVLNVITLAN
>A0A426CFI2|Plug|195-668
KSGPLDFYAGQSYRRSDGDRPNSGGELQDYFLRLGYALNENWSLSFFGDRTNNYAEDPGAENSPYHMGDYKTDDILSVVTLGNRYQRAEGWIKAYWNSGHAAWYDQGMRDTGAPVAPGDTDN
>A0A2Z6B331|Plug|468-649
GISHTFQDLFKIDLTYFHDDGQDRYVFTLPPFPPTWSNTEEFEIEGFEASLSVTPTPDLSLFTGLTIQNSTPGDMPYVPETTVSGGFNWQFLEDFNLSMDCEYVSDMYALKQVRKAGTSNTE
>A0A5K8AAB5|Plug|165-468
GVVNIVPKRKTIEGYTTKAQVAFGSYDTLVAKAEHGGKISGWDYYIGGGHRESDGHRDNSEGELDNAYGRLGRQLNENWDASLFAMWNDNYADDPGAEGADPDARE GRYETRAFLSTLTLSH
>A0A846QDZ5|Plug|338-504
GIDQQWWDGNIRNTKDDGTESSVLVPEFSLCMPYASANYLIGDEKGWHAIPSVGARFYSHNKFDDTTAPHAGIVAGYGTTEVHASIAKGVVYPGLEVTMVPPVSSAITKWDELDPEEVWHTE
>A0A846QDZ5|Plug|530-642
VDPDNMPYAPDTTVKAGVNWHFAELWTLSADCEYVDDMYALSQSRKTTVANTEKVDSHFLLNARLARAFELPSWSAKGEVFCALENITDTDYEYRPDYPMPGINGMVGARLTF
```

Εικόνα 2.16 Fasta file of 4854 cluster

Κάθε κλήση γίνεται με παραμέτρους όπως:

- --force για αντικατάσταση υπαρχόντων αρχείων
- --outfmt=clu για μορφή Clustal
- --wrap=80 για αναγνωσιμότητα (σπάσιμο κάθε 80 χαρακτήρες)

Στο τέλος έχουμε τρία αρχεία clustal_alignments_50, clustal_alignments_60 και clustal_alignments_70 αντίστοιχα, με τα αποτελέσματα της πολλαπλής στοίχισης .

```
File  Edit  Format  View  Help
CLUSTAL O(1.2.4) multiple sequence alignment


E1JXD6|Plug|166-664          -LAAVDMTPKRMDHEGFSSRVTAQYGSYNTSQETAETGGKIGRFDYYAAQSYRYSSGSRAMSDGMLTSYYGRLGYGLTDN
W0JAA3|Plug|163-675          AFTAVNITPRQKRTEGLAADINVAVGSDRTRVEKMSAGGKTGPFDFYAGQSYRRADGDRTNSGGELQDYFLRLGYALNEN
Q72FW3|Plug|161-649          AFSAIDVIPKRMTEDGYRTRLFTSYGSYNTFTQGVEHGGKQGGVDYYAGQSVRLSDGHREHSDGQIENYFGRLGYDLYEN
A0A426CFI2|Plug|195-668      ------------------------------------KSGPLDFYAGQSYRRSDGDRPNSGGELQDYFLRLGYALNEN
A0A2Z6B331|Plug|468-649      ------------------------------------------------------------------------------
A0A5K8AAB5|Plug|165-468      --GVVNIVPKRKTIEGYTTKAQVAFGSYDTLVAKAEHGGKISGWDYYIGGGHRESDGHRDNSEGELDNAYGRLGRQLNEN
A0A846QDZ5|Plug|338-504      ------------------------------------------------------------------------------
A0A846QDZ5|Plug|530-642      ------------------------------------------------------------------------------


E1JXD6|Plug|166-664          WNAHATVIRTDNYAMDPGPEGQ-PEERDGKYVTRDWHMVATLANTYDCAEGTLKGYWNRGEGNWFNQ----------SGK
W0JAA3|Plug|163-675          WSLSFFGDRTDNHAEDPGAENA-PY-HMGDYQTDDWLSVLTLANRYRRAEGWIKAYWNSGHAAWYDQGMRDTGAPVAPGD
Q72FW3|Plug|161-649          WNVSWFGNVTRNVARDPGSSLS-TG-NDGRFYTDDVLNVITLANDYGDADGHIKVYWNNGYANWAGQ----------KGN
A0A426CFI2|Plug|195-668      WSLSFFGDRTNNYAEDPGAENS-PY-HMGDYKTDDILSVVTLGNRYQRAEGWIKAYWNSGHAAWYDQGMRDTGAPVAPGD
A0A2Z6B331|Plug|468-649      ------------------------------------------------------------------------------
A0A5K8AAB5|Plug|165-468      WDASLFAMWNDNYADDPGAEGADPDAREGRYETRAFLSTLTLSHDYERFSGEIKGYRSAGEGDWLDQPTDT------EGV
A0A846QDZ5|Plug|338-504      ------------------------------------------------------------------------------
A0A846QDZ5|Plug|530-642      ------------------------------------------------------------------------------


E1JXD6|Plug|166-664          DNDNTNDSDLYGVRAKEKLRLWQGGEIAFGADMDWISGQAEFTTDPELTINSLTDWSFTPGGVTHFDRMTWSIFSPFLGV
W0JAA3|Plug|163-675          TDNTIMDWDLYGIRAGEKITAWEGGEVLAGLDVDVMSGRGEFERYN----------PALTTRFGREDFRIVSPHAAV
Q72FW3|Plug|161-649          ADTTISDWDLYGLRAREAFRAWEGGEVTAGFDYDVMSGKALFENDN----------G-TSSEFEREDFVLFSPYAAV
A0A426CFI2|Plug|195-668      TDNTIMDWDLYGVRAAEKFAAWEGGEVLAGLDVDVMSGKGRFERYN----------PALNTRFGREDFRIVSPYAAV
A0A2Z6B331|Plug|468-649      ------------------------------------------------------------------------------
A0A5K8AAB5|Plug|165-468      REDLYNDFLYYGIKAKEKVNLGHGTELLAGLDWDYTEGDYTQEYS----------DGSSDAWDGDNFDILSPYAAI
A0A846QDZ5|Plug|338-504      ----------------------------GIDQQWWDGNIRNTKD----------DGTESSVLVPEFSLCMPYASA
A0A846QDZ5|Plug|530-642      ------------------------------------------------------------------------------


E1JXD6|Plug|166-664          SQMIGSKEGFYATPSAGLRYYGHSQFQSEWSPQAGLLLGYKDLELHGTYSRGVNYPGLNVAVFSQNVISSL------GQN
W0JAA3|Plug|163-675          SHLFGTRDGYYAQPSAGVRFYDHNIFGSEVAPHAGVVAGYKATEAHAGYARGVSYPGLNVAAFSQAVLPPLYNDPAKRNA
Q72FW3|Plug|161-649          SHEFGSRDDFYAIPSAGMRFYTHNEFGDQVAPHTGLVFGYRDTELHAGYARGVNYPGLNVAVFSENVIPPIMGGPF-RDD
A0A426CFI2|Plug|195-668      NHLFGARDGYYAQPSAGIRFYDHSIFGSEVAPHAGVVAGYKDTETHAGYARGVSYPGLNVAAFSQAVLPPLYNDPAKREA
A0A2Z6B331|Plug|468-649      ------------------------------------------------------------------------------
A0A5K8AAB5|Plug|165-468      SWRWGDENGLQVIPSAGIRYYDHSLFDSQWAPHAGLVAAYGPAEVHVGYSRGVLYPGLEVEVMSEAVLPAL------GES
A0A846QDZ5|Plug|338-504      NYLIGDEKGWHAIPSVGARFYSHNKFDDTTAPHAGIVAGYGTTEVHASIAKGVVYPGLEVTMVP-----PVS---SAITK
A0A846QDZ5|Plug|530-642      ------------------------------------------------------------------------------
```

Εικόνα 2.17 Alignment result file of 4854 cluster

2.8 **Συμπληρωματική Ανάλυση στα** Clusters **με υψηλό κατώφλι για εύρεση** Domains **μέσω** HMMscan **και της Βάσης** OMPdb

Πραγματοποιείται ένας συμπληρωματικός έλεγχος των ομάδων (clusters) με τα υψηλότερα ποσοστά β-βαρελιών προς συνολικές κενές περιοχές, με στόχο την αξιολόγηση της πιθανότητας οι υποψήφιες β-barrel περιοχές από κάθε ομάδα να ανιχνεύονται από το εργαλείο hmmscan με την βάση omdb.hmm έστω και με υψηλό κατώφλι E-value. Όλες οι αλληλουχίες από κάθε cluster υποβάλλονται εκ νέου σε ανάλυση με το εργαλείο hmmscan, αυτή τη φορά με υψηλό κατώφλι E-value: 0.1, που επιτρέπει την ανίχνευση ακόμα και ασθενών σηματοδοτήσεων και χρήση της βάσης OMPdb, μέσω του HMM αρχείου ompdb.hmm, το οποίο περιλαμβάνει μοντέλα για τα ήδη γνωστά και καταγεγραμμένα domains β-barrel πρωτεϊνών εξωτερικής μεμβράνης.

Η επιλογή ενός αυξημένου E-value αποσκοπεί στη διεύρυνση του εύρους ευαισθησίας της ανίχνευσης, ώστε να διερευνηθεί κατά πόσο οι gap περιοχές φέρουν οριακή ή ασθενή ομοιότητα με ήδη γνωστά OMP (Outer Membrane Protein) domains. Η ανίχνευση ακόμη και χαμηλής εμπιστοσύνης ομοιοτήτων από το εργαλείο hmmscan με χρήση του ompdb.hmm αποτελεί ένδειξη ότι τα προφίλ του OMPdb φέρουν πληροφορία σχετική με τις υπό διερεύνηση περιοχές, έστω και χωρίς υψηλή ακρίβεια, δηλαδή ότι τα αναγνωρισμένα β-βαρέλια φαίνονται να μοιάζουν σε κάποιο βαθμό με τις κενές περιοχές που εικάζουμε ως β-βαρέλια.

Στο εργαλείο hmmscan εισάγουμε όλες τις αλληλουχίες από το αρχείο με τα clusters και άμα εντοπιστεί κάποιο από τα 154 β-βαρέλια γράφεται αυτό και το E-value του δίπλα στην κάθε αλληλουχία.

```
1   522aa, >N1MLS8|Plug| 305-826 at 54.98%,1.0000,YES  OMPdbModel29 E=5.3e-32
PVDANGDGQLSQQEFSAIGDFNPFGSFGCGGGSVFANASNNDACYATNYPWLGLTNVGTTKLSRRKVLTGKNDLLVNKAKTAYFDMIFNADSGLEIKNQFFYDGIDNDNENSYGFSQFHKS
2   529aa, >E0TCY1|Plug| 274-802 at 45.37%,1.0000,YES  OMPdbModel29 E=2.7e-32
DQNGDGELGRREFDAAGVFNPFGNFVCDPSDVASFGVGPEGMTDACLMNSLPGLALTDVGETTLDRRNVLTGEDDFLENKTLTFYFDSIYENDDGFELISQLFFEGYENENENDYGFSQFH
3   107aa, >A0A0N1ALR1|Plug| 298-404 at 48.60%
```

Εικόνα 2.18 Στη κενή περιοχή N1MLS8|Plug| 305-826 εντοπίστηκε γνωστό domain (OMPdbModel29) ,το οποίο έχει και πολύ ισχυρή αντιστοίχιση (E=5.3e-32).

```
/DMTPKRMDHEGFSSRVTAQYGSYNTSQETAETGGKIGRFDYYAAQSYRYSSGSRAMSDGMLTSYYGRLGYGLTDNWNAHATVIRTDNYAMDPGPEGQPEERDGKYVTRDWHMVATLAN
 513aa, >W0JAA3|Plug|163-675,1.0000,YES  DUF2490 E=0.0066 | DUF2490 E=0.042 | OMP_b-brl_3 E=7.3e-05 | Porin_2 E=0.00018
AVNITPRQKRTEGLAADINVAVGSDRTRVEKMSAGGKTGPFDFYAGQSYRRADGDRTNSGGELQDYFLRLGYALNENWSLSFFGDRTDNHAEDPGAENAPYHMGDYQTDDWLSVLTLAN
```

Εικόνα 2.19 Η αλληλουχία παρουσιάζει πολλαπλές ενδείξεις domain όπως την DUF2490 που είναι αδύναμη αλλά επαναλαμβανόμενη με (E=0.0062 και E=0.042), το OMP_b-brl_3 με πολύ ισχυρή συσχέτιση (E=7.3e-05) και το Porin_2 με (E=0.00018).

Ωστόσο, αξίζει να τονιστεί ότι παρόλο που τα παραπάνω E-values είναι σχετικά μικρά και υποδηλώνουν κάποια βαθμό ομοιότητας, δεν ξεπερνούν το προκαθορισμένο όριο TC (Trusted Cutoff) που έχει τεθεί για κάθε HMM μοντέλο. Το TC βασίζεται στο score και χρησιμεύει ως αυστηρό κατώφλι αξιοπιστίας για να διαχωρίζει τις υψηλής εμπιστοσύνης ανιχνεύσεις από τις πιο αμφίβολες. Συνεπώς, ακόμα και αν το E-value

είναι χαμηλό, εφόσον το αντίστοιχο score δεν υπερβαίνει το TC, η συγκεκριμένη ανίχνευση δεν μπορεί να θεωρηθεί αξιόπιστη ένδειξη παρουσίας του domain.

2.9 **Αναζήτηση Ύπαρξης Δομών στις Κενές Περιοχές για κάθε** Cluster **με AlphaFold**

Το AlphaFold DB είναι μια ανοιχτή βάση δεδομένων που κάνει πρόβλεψη της τρισδιάστατης δομής για περισσότερες από 200 εκατομμύρια πρωτεΐνες, αποτελώντας ισχυρό εργαλείο για τη την καλύτερη γνώση της φυσιολογίας των πρωτεϊνών, τη φαρμακευτική ανάπτυξη και τη μοριακή βιολογία. Αναπτύχθηκε από τη Google DeepMind, σε συνεργασία με το EMBL-EBI. Ουσιαστικά το AlphaFold είναι ένα τεχνητό νευρωνικό δίκτυο που προβλέπει τη 3D δομή μιας πρωτεΐνης με βάση την αλληλουχία αμινοξέων της. Η πρόσβαση μπορεί να γίνει από την ιστοσελίδα: https://alphafold.com/.

Η ανάπτυξη του AlphaFold σηματοδότησε σημαντική πρόοδο στο μακροχρόνιο πρόβλημα της πρόβλεψης της δομής των πρωτεϊνών (protein folding problem), και για πρώτη φορά επιτεύχθηκε ακρίβεια συγκρίσιμη με εκείνη πειραματικών τεχνικών, ακόμη και για πρωτεΐνες που δεν είχαν γνωστά ομόλογα. Η προσέγγιση του AlphaFold ενσωματώνει εξελικτικές πληροφορίες από πολλαπλές ευθυγραμμίσεις αλληλουχιών (MSAs) καθώς και φυσικοχημικά στοιχεία, ξεπερνώντας έτσι περιορισμούς προηγούμενων ομολογιακών ή φυσικών μεθόδων.

Η βάση δεδομένων AlphaFold έχει σημαντικές προοπτικές εφαρμογής:

- Στην ανακάλυψη και σχεδίαση φαρμάκων, μέσω της πρόβλεψης στόχων-πρωτεϊνών και της αλληλεπίδρασης φαρμακομόριων με τις δομές τους.
- Στη μελέτη σπάνιων ή κακώς χαρακτηρισμένων πρωτεϊνών, όπου δεν υπάρχουν διαθέσιμες πειραματικές δομές.
- Στην κατανόηση της λειτουργίας των πρωτεϊνών σε διάφορα βιολογικά συστήματα [24-25].

Η παρουσία προβλεπόμενης τρισδιάστατης δομής σε μία αλληλουχία μέσω του εργαλείου AlphaFold ,είναι βασικός δείκτης για την δομή καθώς και την λειτουργική σημασία της πρωτεϊνικής περιοχής . Επιθυμητό αποτέλεσμα θα ήταν στις ομάδες μας που έχουν πολλά πιθανά β-βαρέλια (από PRED-TMBB2), οι αλληλουχίες να τείνουν να υιοθετούν σταθερές, επαναλαμβανόμενες δομές, που συχνά διατηρούνται εξελικτικά και να μην είναι τυχαίες ή μη λειτουργικές περιοχές, οι οποίες εμφανίζουν ασαφείς ή ασταθείς διαμορφώσεις .Μέσω της χρήσης του AlphaFold, αναζητήθηκε η ύπαρξη τέτοιων σταθερών δομών στις κενές περιοχές των

αλληλουχιών κάθε cluster. Ιδιαίτερα σημαντικό είναι όταν οι προβλέψεις δείχνουν δομές τύπου β-βαρελιού, οι οποίες αποτελούν χαρακτηριστικό των εξωτερικών μεμβρανικών πρωτεϊνών.

Από κάθε cluster εισάγουμε στο εργαλείο AlphaFold τα ονόματα από όλες τις πρωτεΐνες που παρατηρούνται σε αυτό και κρατάμε την πληροφορία αν εμφανίζουν δομή ή όχι .

```
>Cluster 159 8/10 (80.0%) WITH B BARRELS
0   1179aa, >J3CU79|OmpA| 202-1380 at 41.14%,1.0000,YES
RVELRVWFQPAVPAAAPVAMPAPLAPCTPQAVGQSNVPFRITIDGEPINTAEIPNEADGQRCADVALERADIQVRYDSLAAVPLMNAWATPGTAVKGEAVEFRAWSNYIPWIKKAELRLFRPGQKTQEQPFEILPIGWSGASRWNVPANTA
1   1175aa, >A6T1W6|OmpA| 202-1376 at 42.04%,1.0000,YES
RVALQVWFDVAPPVVALPKPAPCSPQAVSQIDAPFRITIDGEPFHGNEANEADGQRCTDVALERADIQVRYDSLAIAPTMNIWATPNAAVKGQPVEFRAWSNYIPWLKKAELRLFRAGQKTQEKPIEVLPVSWNGVTNWTAPANTTDDQVF
2   1296aa, >J3D481|OmpA| 225-1520,1.0000,YES
RVEIRVWYTEAPAMVTREKLVENNACVANPQAVDALPFSITVDGQPVDVDKQAQEADRQRCVDVGLEKADIQIKFDPLNVAPALNVWALPSTAVRSKPVVFGTYTNYAWWIKRAELRVFAKGQNAQETPLAVVPFTAGDAVQWQAPDNAPS
3   1184aa, >Q1GYN0|OmpA| 219-1402 at 41.39%,1.0000,YES
RVIIRVWFDEQEIIPPTPVTTVPPVRSACEASSVARAGAPFRISIDGEPVSLDEGPLEADRQRCADVALEKADIQVRYDNLSTLPQLNVWNTTRGVVRGEEAEFRAWANYLPWIKKAEIRLFKPGQQPQEIPLAVLPVNWNGVTKWQAKVD
4   428aa, >M7NVU2|OmpA| 210-637 at 42.06%
RVEVEIWYDEIISNSEFRCTAGQVTSEAVAPFRISVDGQPMGEHAAGTADQQRCVDVALAEAHLQVRYDNLSAQPRLNVASAPRMAVPGQTHYFRGYSNYMSWVESGEVRIYGKPSRWSQAELLETVALDPLLNGEWVPTDNLPSQLYYQL
5   1173aa, >A0A1A9T5I8|OmpA| 210-1382 at 40.84%,1.0000,YES
RVEIRVWFDQPILAAPLPTPIRSACETVSTAHSNAPFRISVDGEPLDLNNTPLEADRQRCTDVALEKADIQVRFDNLSTLPQMNAWVTPNGIVRGEKATFRAWANYLPWIKKAEIRLFKPGQQVQETPIEILPVNWNGVTTWQTPLTGPDQ
6   1177aa, >A0A1U9JRX4|OmpA| 200-1376 at 41.97%,1.0000,YES
RVDLRLWQQAPEPVFAPVAANPPAACAPLAAQPDLPFRISVDGEALTLGEPVNEADRQRCTDVALAQADIQIKYDDLAATPSLNVWTARDLALRGQPVQFLGYSNYVHWIHRAEVRVFKRGERPDAPPLAVTPLNWAEASEWTVPAEGQEE
7   314aa, >A0A418X0I5|OmpA| 221-534 at 74.52%,0.9195,YES
RVEIRVWFSEAPAQVTVTREKVIDNNACVTNPSAAELPFSISIDGQPVQLDAKAQEADFQRCVDVALEKADIQIKFDPLNVSPALNVWWVPSTAVRGKQTYFGTYTNYAWWLRRAELRVFAKGQNAQETPFAVVPLAPGAALQWQVPDDAF
8   1179aa, >A0A323UWK0|OmpA| 211-1389 at 41.56%,1.0000,YES
RVEIRAWYDVTPLVALDSPLVVADPQRCASLGEAEALPFRVTVDGEVLYPDVGVNEADRQRCADVALERADIQVKYDDLAVRPALNVWVEEDVGVRGSVVRFRGYSNYIAWIRKAEVRIFRTGELPELTPLATLPLTWEASTEWKIPADDG
9   442aa, >A0A1H2ZDD1|OmpA| 211-652 at 41.18%
DWALNRRVEVEIWYDEAVVVTPAKAEQCAPGNVPLQNVLPFMISVDGAPLNNGEAGSADAQRCVDVELAKDLLQVRFDNLSTQPRLNVVSGARVARIGEVQQFSGYSNYLHWIKRAEVRILGRKRLFGAPELLQTLPLDPELKADWTPTAG
>Cluster 177 6/8 (75.0%) WITH B BARRELS
0   786aa, >G8NS47|CarboxypepD_reg|116-901 at 71.63%,0.8344,YES
AVGADSTTVEVTASNADLQTMSAGTGTTVDPALVDSLPTIGREVSTFSTLQPGVTPGGNVAGTTTDQATFMLDGGNNTFDMDGTSSTYTTSFAASTTGGPGGQPQAGTMPMPQDSIEEFKVSTTGQTADFNNSSGSQTQAVTKRGRDKWTG
1   443aa, >G8NS47|CarboxypepD_reg| 938-1380 at 69.07%,0.7665,YES
QRQINRKMLVEVGYIGRIVHHEFNQLNPNSVPYMMTLGGQSFESAYMAIETAFGCTTSASLCAKSPTPSGIAPQPFFENALGGPNSPFCAKHASCTAALVANATYASDFRQQKVFSLWQGLDNNTFGAAASATNPQFYFARSLMGTPTSNA
2   540aa, >A0A1H5VV10|CarbopepD_reg_2, CarboxypepD_reg, EMC7_beta-sandw, SpaA|132-671 at 68.89%
NADLQTINATTGQTVEQAMVNSLPAIGRDVATFVTMQPGVSPAGYVAGTAQDQASISLDGGQNSADMDGTQGVYTSNNLGSTTGGFLGAGASGVMPMPQDSIEEFKVSTSGQTADFNNSTGSNSQVVTKRGHDTIHGTVYEYYLDNTFNAN
3   691aa, >A0A1H5VV10|CarbopepD_reg_2, CarboxypepD_reg, EMC7_beta-sandw, SpaA| 691-1381 at 61.36%
PAATPIGAHTTIPYYNIYGTDTWKVTPTLTLNYGISYAIEMPPHEANGNQVMWTDPSGNEQSTDKWLENRYTAAAQGTVYNPYFAFALLKNVDGGGRKYAYNPYYASVSPRISFAWNPKFTNEGMAKIFGSGATVIRGGYGRLYARINGDL
4   1232aa, >A0A7L5AB98|CarboxypepD_reg|119-1350 at 52.35%,0.9717,YES
AIGSESTTIEVQASNADLQTLNSTVGESVDPVMVDSLPAIGRDVSSFASFQPGVTPGGSVAGTVSDQAVITLDGGSNSSDMDGNMQQYTGSFGNSTTGGFLGAGSSGVMPMPQDSVEEFRVSTSGQTADFNNSSGSQSQIVTKRGRDRWTG
5   1251aa, >A0A7D7XCN7|CarboxypepD_reg|131-1381,0.9999,YES
VGAENTTVEVTASNADLQTMNASTGTTVDPSMVQALPTIGREVATFTTMQPGVTPGGNVAGTTTDQASFTLDGGNNSNDMDGTLTTYTTSYATSTTGGFLGAAPQGTMPMPQDSIEEFKVATTGQTADFNNSSGSQTQAVTKRGRDRWTGT
6   543aa, >A0A7D8BE67|CarboxypepD_reg|118-660 at 69.24%,0.8013,YES
ALGSTNTTVDVQASNADLQTMNASTGTTVDPALVDSLPAIGRDVATFMEMQPGVAPGGQTAGTTADQTTFTLDGGANTSDMDGTAIGYTSGNTNSTTGGALGAGPAGVVPMPQDSIEEFKVSTTGQTADFAASSGSQSQVVTKRGHDQWHG
7   688aa, >A0A7D8BE67|CarboxypepD_reg| 678-1365 at 61.05%,0.8787,YES
```

Εικόνα 2.20 Το αρχείο με τα Clusters

Τα αποτελέσματα του Alpha Fold εισάγονται στο αρχείο με τα clusters και συγκεκριμένα όταν έχει βρεθεί δομή για μία αλληλουχία γράφεται δίπλα σε αυτή 'ALPHA' ενώ άμα δεν έχει βρεθεί 'NOT ALPHA'.

```
>Cluster 159 8/10 (80.0%) WITH B BARRELS 4
0   1179aa, >J3CU79|OmpA| 202-1380 at 41.14%,1.0000,YES NOT ALPHA
RVELRVWFQPAVPAAAPVAMPAPLAPCTPQAVGQSNVPFRITIDGEPINTAEIPNEADGQRCADVALERADIQVRYDSLAAVPLMNAWATPGTAVKGEAVEFRAWSNYIPWIKKAELRLFRPGQKTQEQPFEILPI(
1   1175aa, >A6T1W6|OmpA| 202-1376 at 42.04%,1.0000,YES  OMP_b-brl E=0.00038    NOT ALPHA
RVALQVWFDVAPPVVALPKPAPCSPQAVSQIDAPFRITIDGEPFHGNEANEADGQRCTDVALERADIQVRYDSLAIAPTMNIWATPNAAVKGQPVEFRAWSNYIPWLKKAELRLFRAGQKTQEKPIEVLPVSWNGVT
2   1296aa, >J3D481|OmpA| 225-1520,1.0000,YES  OMP_b-brl E=7.3e-05 | OMP_b-brl E=0.014 | YadA_anchor E=0.0097   NOT ALPHA
RVEIRVWYTEAPAMVTREKLVENNACVANPQAVDALPFSITVDGQPVDVDKQAQEADRQRCVDVGLEKADIQIKFDPLNVAPALNVWALPSTAVRSKPVVFGTYTNYAWWIKRAELRVFAKGQNAQETPLAVVPFT
3   1184aa, >Q1GYN0|OmpA| 219-1402 at 41.39%,1.0000,YES NOT ALPHA
RVIIRVWFDEQEIIPPTPVTTVPPVRSACEASSVARAGAPFRISIDGEPVSLDEGPLEADRQRCADVALEKADIQVRYDNLSTLPQLNVWNTTRGVVRGEEAEFRAWANYLPWIKKAEIRLFKPGQQPQEIPLAVLI
4   428aa, >M7NVU2|OmpA| 210-637 at 42.06%  NOT ALPHA
RVEVEIWYDEIISNSEFRCTAGQVTSEAVAPFRISVDGQPMGEHAAGTADQQRCVDVALAEAHLQVRYDNLSAQPRLNVASAPRMAVPGQTHYFRGYSNYMSWVESGEVRIYGKPSRWSQAELLETVALDPLLNGEW
5   1173aa, >A0A1A9T5I8|OmpA| 210-1382 at 40.84%,1.0000,YES NOT ALPHA
RVEIRVWFDQPILAAPLPTPIRSACETVSTAHSNAPFRISVDGEPLDLNNTPLEADRQRCTDVALEKADIQVRFDNLSTLPQMNAWVTPNGIVRGEKATFRAWANYLPWIKKAEIRLFKPGQQVQETPIEILPVNWN
6   1177aa, >A0A1U9JRX4|OmpA| 200-1376 at 41.97%,1.0000,YES  OMP_b-brl E=6.3e-05    NOT ALPHA
RVDLRLWQQAPEPVFAPVAANPPAACAPLAAQPDLPFRISVDGEALTLGEPVNEADRQRCTDVALAQADIQIKYDDLAATPSLNVWTARDLALRGQPVQFLGYSNYVHWIHRAEVRVFKRGERPDAPPLAVTPLNWA
7   314aa, >A0A418X0I5|OmpA| 221-534 at 74.52%,0.9195,YES   ALPHA
RVEIRVWFSEAPAQVTVTREKVIDNNACVTNPSAAELPFSISIDGQPVQLDAKAQEADFQRCVDVALEKADIQIKFDPLNVSPALNVWWVPSTAVRGKQTYFGTYTNYAWWLRRAELRVFAKGQNAQETPFAVVPLA
8   1179aa, >A0A323UWK0|OmpA| 211-1389 at 41.56%,1.0000,YES  OMP_b-brl E=0.0028 | CopB E=0.00057    NOT ALPHA
RVEIRAWYDVTPLVALDSPLVVADPQRCASLGEAEALPFRVTVDGEVLYPDVGVNEADRQRCADVALERADIQVKYDDLAVRPALNVWVEEDVGVRGSVVRFRGYSNYIAWIRKAEVRIFRTGELPELTPLATLPLT
9   442aa, >A0A1H2ZDD1|OmpA| 211-652 at 41.18%  NOT ALPHA
DWALNRRVEVEIWYDEAVVVTPAKAEQCAPGNVPLQNVLPFMISVDGAPLNNGEAGSADAQRCVDVELAKDLLQVRFDNLSTQPRLNVVSGARVARIGEVQQFSGYSNYLHWIKRAEVRILGRKRLFGAPELLQTLI
>Cluster 177 6/8 (75.0%) WITH B BARRELS 0
0   786aa, >G8NS47|CarboxypepD_reg|116-901 at 71.63%,0.8344,YES NOT ALPHA
AVGADSTTVEVTASNADLQTMSAGTGTTVDPALVDSLPTIGREVSTFSTLQPGVTPGGNVAGTTTDQATFMLDGGNNTFDMDGTSSTYTTSFAASTTGGPGGQPQAGTMPMPQDSIEEFKVSTTGQTADFNNSSGS(
1   443aa, >G8NS47|CarboxypepD_reg| 938-1380 at 69.07%,0.7665,YES   NOT ALPHA
QRQINRKMLVEVGYIGRIVHHEFNQLNPNSVPYMMTLGGQSFESAYMAIETAFGCTTSASLCAKSPTPSGIAPQPFFENALGGPNSPFCAKHASCTAALVANATYASDFRQQKVFSLWQGLDNNTFGAAASATNPQI
2   540aa, >A0A1H5VV10|CarbopepD_reg_2, CarboxypepD_reg, EMC7_beta-sandw, SpaA|132-671 at 68.89%    NOT ALPHA
NADLQTINATTGQTVEQAMVNSLPAIGRDVATFVTMQPGVSPAGYVAGTAQDQASISLDGGQNSADMDGTQGVYTSNNLGSTTGGFLGAGASGVMPMPQDSIEEFKVSTSGQTADFNNSTGSNSQVVTKRGHDTIH(
3   691aa, >A0A1H5VV10|CarbopepD_reg_2, CarboxypepD_reg, EMC7_beta-sandw, SpaA| 691-1381 at 61.36%  NOT ALPHA
PAATPIGAHTTIPYYNIYGTDTWKVTPTLTLNYGISYAIEMPPHEANGNQVMWTDPSGNEQSTDKWLENRYTAAAQGTVYNPYFAFALLKNVDGGGRKYAYNPYYASVSPRISFAWNPKFTNEGMAKIFGSGATVII
4   1232aa, >A0A7L5AB98|CarboxypepD_reg|119-1350 at 52.35%,0.9717,YES   NOT ALPHA
AIGSESTTIEVQASNADLQTLNSTVGESVDPVMVDSLPAIGRDVSSFASFQPGVTPGGSVAGTVSDQAVITLDGGSNSSDMDGNMQQYTGSFGNSTTGGFLGAGSSGVMPMPQDSVEEFRVSTSGQTADFNNSSGS(
5   1251aa, >A0A7D7XCN7|CarboxypepD_reg|131-1381,0.9999,YES NOT ALPHA
VGAENTTVEVTASNADLQTMNASTGTTVDPSMVQALPTIGREVATFTTMQPGVTPGGNVAGTTTDQASFTLDGGNNSNDMDGTLTTYTTSYATSTTGGFLGAAPQGTMPMPQDSIEEFKVATTGQTADFNNSSGSQT
```

Εικόνα 2.21 Το τελικό αρχείο με τα Clusters.

Θα πρέπει να επισημάνουμε ότι μπορεί η AlphaFold να μας προσφέρει πρόβλεψη για την τρισδιάστατη δομή μίας πρωτεΐνης, όπως για παράδειγμα ενός β-βαρελιού, ωστόσο δεν εγγυάται την ταξινόμησή του ως διακριτό domain. Σε τέτοιες περιπτώσεις, εξετάσαμε εάν η συγκεκριμένη δομή αναγνωρίζεται ως domain από αυτόματα συστήματα, όπως το TED.

Το TED (The Encyclopedia of Domains) είναι ένα υπολογιστικό σύστημα σχεδιασμένο για τη συστηματική ανίχνευση και κατηγοριοποίηση δομικών domains σε προβλεπόμενες πρωτεϊνικές δομές της AFDB. Σε αντίθεση με τις παραδοσιακές προσεγγίσεις που βασίζονται αποκλειστικά σε αλληλουχιακές ομοιότητες (όπως Pfam και Gene3D), το TED αξιοποιεί τεχνικές βαθιάς μάθησης και εξελιγμένες δομικές συγκρίσεις. Μέσω αυτών των εργαλείων, είναι ικανό να ανιχνεύσει μακρινές εξελικτικές σχέσεις και να αναγνωρίσει νέες ή άγνωστες δομικές αρχιτεκτονικές. Το TED έχει καταφέρει να εντοπίσει σχεδόν 365 εκατομμύρια domains σε περισσότερα από 1 εκατομμύριο ταξινομικά είδη, με πάνω από 100 εκατομμύρια εξ αυτών να μην είχαν ανιχνευθεί με βάση αποκλειστικά την αλληλουχία. Επιπλέον, έχει προσδιορίσει πάνω από 10.000 νέες δομικές αλληλεπιδράσεις μεταξύ υπεροικογενειών, αναδεικνύοντας μια πολύ μεγαλύτερη ποικιλία στις γεωμετρίες συσκευασίας domains από ό,τι καταγραφόταν προηγουμένως σε πειραματικές δομές. Η συμβολή του TED είναι ουσιώδης για την επανακατάταξη και καλύτερη αξιοποίηση του τεράστιου δομικού χώρου της AFDB [26]. Σε εφαρμογές όπως η ανάλυση βαρελοειδών περιοχών, το TED παρέχει μια αντικειμενική επιβεβαίωση για το αν η δομή αναγνωρίζεται πράγματι ως αυτόνομο domain, ή όχι.

3. **Αποτελέσματα**

Στην παρούσα εργασία, εντοπίστηκαν και ομαδοποιήθηκαν σε clusters 174.962 κενές περιοχές των πρωτεϊνικών αλληλουχιών, μήκους άνω των 80 αμινοξέων, που παρουσιάζουν σημαντική ομοιότητα μεταξύ τους. Οι πρωτεϊνικές περιοχές είχαν επιλεχτεί με το κριτήριο ότι στις πρωτεΐνες που τις περιλαμβάνουν σε κάθε μία από αυτές εντοπίστηκε τουλάχιστον ένα Pfam domain, το οποίο έχει αποδειχθεί ότι σχετίζεται με κάποιο από τα 154 OMP domains του OMPdb και κανένα OMP domain. Η συσχέτιση των Pfam domains με τα β-βαρέλια επιβεβαιώθηκε με τη χρήση τριών ανεξάρτητων στατιστικών ελέγχων: Odds Ratio, Fisher's exact test και Chi-squared test. Η εφαρμογή και των τριών τεστ είχε στόχο να αυξήσει την αξιοπιστία των αποτελεσμάτων. Αξίζει να σημειωθεί ότι τα τρία τεστ παρήγαγαν συνεπή αποτελέσματα ως προς τη στατιστική σημαντικότητα, ενισχύοντας την εγκυρότητα των αποτελεσμάτων.

Παρακάτω παρουσιάζεται η λίστα με όλα τα 159 Pfam domains που συσχετίζονται θετικά με OMP domains.

Pfam_Domain	Odds_Ratio	P_value
FlgT_C	3.121	0.00385
DUF5648	2.686	0.00059
PATR	32.958	<1e-308
Hemopexin	6.243	7.5e-06
KfrB	18.281	0.01766
PapC_N	4564.324	<1e-308
Fib_alpha	10.446	0.00343
NOG1_N	36.562	0.01093
POTRA_1	14.191	<1e-308
T2SSN	19.153	2.2e-15
Tropomyosin	29.250	4.8e-07
Cep192_D1	12.314	0.02968
FlaC_arch	12.187	0.03036
FAS_AT_central	18.507	0.01719
POTRA_2	247.839	<1e-308
AMIN	34.814	<1e-308
VacA	37.117	0.01060
MACPF_D3	109.869	8.6e-09
CAP_N-CM	12.187	0.03036
Phage_T7_tail	2.813	0.04627

Sipho_tail	9.141	0.04780
TraK	11.594	1.4e-14
HphA_N	36.564	4.6e-10
FlgO	4.643	7.3e-09
PEGA	12.693	<1e-308
LpoB	84.327	<1e-308
SdrD_B	2.896	<1e-308
STN	194.966	<1e-308
PT	5.850	0.00104
CFSR	24.375	7.5e-07
Tcp10_C	7.835	0.00121
Secretin_N	556.990	<1e-308
TPR_MLP1_2	24.375	0.00047
Hia_Tpr_ring_dom	66.502	5.3e-14
Patatin	2.289	<1e-308
Autotransporter	190.437	<1e-308
TonB_dep_Rec_b-barrel	928.918	<1e-308
ACCA_BT	18.311	0.01760
Snapin_Pallidin	12.187	0.03036
LptD_N	50.389	<1e-308
LPP	9.141	0.04780
Beta-prism_lec	36.622	0.00032
Alanine_zipper	10.156	4.5e-06
LbR_Ice_bind	12.188	0.00001
Fas1-AflB-like_hel	12.187	0.03036
SLH	18.919	<1e-308
Invasin_D4	4.366	0.00008
FecR_C	1.458	9.2e-10
CBM9_1	28.129	<1e-308
CooC_C	4.632	3.1e-10
Secretin_N_2	2163.962	<1e-308
Peptidase_S6	25.442	<1e-308

UPF0184	18.281	0.01766
Leukocidin	36.566	<1e-308
Invasin_D3	13.970	<1e-308
HphA_C	82.271	2.2e-13
DUF1935	12.187	0.03036
DUF1416	2.348	0.02859
ChlamPMP_M	22.889	4.0e-08
Ribosomal_S7	12.187	0.03036
CarboxypepD_reg	87.148	<1e-308
Pectate_lyase_5	9.141	0.00061
PgaA_barrel	379.322	<1e-308
WYL_2	9.141	0.04780
TSP_3	2.676	<1e-308
DUF874	18.281	0.01766
DUF6221	19.284	0.01568
TM1506	12.187	0.03036
POTRA	102.451	<1e-308
fn3_3	9.656	<1e-308
SpaA	9.657	<1e-308
DUF3943	1.828	0.03377
Big_12	3.447	3.6e-14
PilO	3.047	0.03229
FctA	14.625	0.00134
DUF6850	7.312	0.01021
AC_1	102.168	<1e-308
Cadherin-like	3.047	0.00003
PapC_C	26690.943	<1e-308
DUF807	12.372	0.02937
He_PIG	1.868	6.3e-10
ETX_MTX2	20.895	<1e-308
DUF3438	47.012	2.2e-14
Trp_ring	66.488	<1e-308
POTRA_TamA_1	244.392	<1e-308

CarbopepD_reg_2	96.788	<1e-308
OstA_2	1.684	0.00258
T2SS-T3SS_pil_N	705.907	<1e-308
Sarcoglycan_2	4.875	0.03535
DUF6584	5.970	2.8e-06
TolB_N	11.622	<1e-308
Peptidase_S8	1.229	0.00102
DUF599	12.187	0.03036
AIDA	32.844	<1e-308
DUF3869	7.698	1.6e-07
InvE_AD	33.954	<1e-308
Spore_III_AB	36.562	0.01093
Cohesin	18.077	<1e-308
DUF2730	7.312	0.01021
TPR_19	1.125	0.00107
POTRA_3	5089.989	<1e-308
CCDC90-like	36.562	0.01093
MACPF_1	73.129	2.4e-12
UspA1_rep	37.117	0.01060
EMC7_beta-sandw	14.930	<1e-308
BON	7.591	<1e-308
Chlam_PMP	2.460	2.7e-06
MACPF_D2	73.245	7.1e-07
DUF1093	9.141	0.04780
YadA_stalk	87.357	<1e-308
Cornifin	36.562	0.01093
Omp85	77.591	<1e-308
ARib	9.215	0.04700
SprA_N	26579.735	<1e-308
Filamin	11.846	<1e-308
OmpA	1.355	8.5e-14
NifT	18.281	0.01766
Caps_syn_GfcC_N	5.223	0.02875

MG4	6.023	<1e-308
Lipase_GDSL	1.946	<1e-308
TamB	14.964	<1e-308
DUF2158	9.141	0.04780
DUF348	12.187	0.03036
Coatomer_E	2.099	0.00024
Collagen	2.419	5.0e-11
DUF2937	18.281	0.01766
DinI	36.562	0.01093
type_II_gspD_N0	3297.127	<1e-308
DUF5372	18.281	0.01766
SSSPR-51	36.562	0.01093
ESPR	20.376	<1e-308
Nudix_N	36.562	0.00032
LptC	5.116	<1e-308
ZirS_C	12.187	0.00220
Pertactin	122.846	<1e-308
DUF4520	37.015	0.00030
Ice_binding	3.580	5.3e-06
DUF3447	18.507	0.01719
Plug	302.979	<1e-308
YadA_head	41.590	<1e-308
DUF4573	5.625	0.02297
Big_1	5.836	<1e-308
SF-assemblin	10.446	0.00343
YtkA	2.216	0.02887
TcfC	3.722	2.4e-07
BIg21	42.661	<1e-308
Mac-1	4.388	0.01551
DUF6139	18.281	0.01766
DUF3107	18.471	0.01726
T3S_SPI-1_N0	118.884	<1e-308
Caps_syn_GfcC_C	7.373	<1e-308

FPP	35.844	0.01138
Utp8_C	35.804	0.01140
HAT	35.804	0.01140
YGR210-like_G4	35.722	0.01146
GBP_repeat	35.722	0.01146
UDP-g_GGTase	35.964	0.01130
Peptidase_C23	17.945	0.01840
TRPM_tetra	35.964	0.01130

Πίνακας 3.1 Λίστα όλα τα Pfam Domains που παρουσιάζουν θετική συσχέτιση με β-βαρέλια.

Επιπλέον, στο τελικό αρχείο clusters_with_b_barrels.txt ενσωματώθηκε και η έξοδος του αλγορίθμου PRED-TMBB2 (εντόπισε 19.191 πιθανά β-βαρέλια από τις συνολικές κενές περιοχές που μελετούσαμε), με στόχο την πρόβλεψη της πιθανότητας οι αντίστοιχες αλληλουχίες να σχηματίζουν β-βαρέλια. Από το σύνολο των clusters (Το cd-hit είχε ομαδοποιήσει τις αλληλουχίες σε 77.104 clusters), επιλέχθηκαν μόνο εκείνα στα οποία το ποσοστό των προβλεπόμενων β-βαρελιών υπερβαίνει το 70%, καθώς αυτά παρουσιάζουν το μεγαλύτερο ενδιαφέρον για περαιτέρω μελέτη ως πιθανοί εκπρόσωποι νέων οικογενειών β-βαρελιών.

Επιπλέον, χρησιμοποιήθηκαν τα αποτελέσματα από το εργαλείο HMMER (hmmscan) με κατώφλι E-value < 0.1, σε συνδυασμό με τα OMPdb HMMs, για την αναγνώριση ασθενών ενδείξεων παρουσίας OMP σχετιζόμενων domains στις κενές περιοχές. Τέλος, ενσωματώθηκαν προβλέψεις από την AlphaFold, ώστε να αξιολογηθεί η δευτεροταγής και τριτοταγής δομή των αλληλουχιών, ενισχύοντας τη βιολογική σημασία των προτεινόμενων περιοχών.

Όλα τα παραπάνω δεδομένα συνοψίζονται σε πίνακα, ο οποίος περιλαμβάνει τα 525 συνολικά clusters, με τις ακόλουθες πληροφορίες: αριθμός συνολικών αλληλουχιών, αριθμός προβλεπόμενων β-βαρελιών, παρουσία/απουσία ALPHA-fold δομής, τα αναγνωρισμένα domains με υψηλό E-value και τη ελάχιστη E-value από αυτά.

Clusters Summary

Cluster	Total	β-barrels	Alpha	NotAlpha	Domains	Min E-value
Cluster 159	10	8	1	9	CopB, OMP_b-brl, YadA_anchor	6.3e-05

Cluster 177	8	6	0	8	-	None
Cluster 182	13	12	1	12	BBP2, DUF481, OMP_b-brl, Toluene_X, YadA_anchor	1.9e-06
Cluster 184	2	2	0	2	-	None
Cluster 198	8	8	0	8	DUF481, OMP_b-brl, OMP_b-brl_2, YadA_anchor	6.9e-06
Cluster 229	4	3	3	1	-	None
Cluster 232	5	5	0	5	BBP2, DUF2490, OMP_b-brl, Porin_7, YadA_anchor	4.4e-06
Cluster 233	49	35	0	49	DUF2490, MtrB_PioB, OMP_b-brl	0.00034
Cluster 234	4	3	1	3	DUF2490	0.0032
Cluster 236	4	3	1	3	OMP_b-brl	0.00092
Cluster 260	5	4	4	1	OMPdbModel29	0.0014
Cluster 266	12	9	12	0	OMPdbModel29	7.7e-06
Cluster 272	5	5	5	0	OMPdbModel29	0.0025
Cluster 284	7	5	5	2	OMPdbModel29, OprD	6e-08
Cluster 287	4	3	3	1	-	None
Cluster 302	5	5	5	0	-	None
Cluster 326	2	2	1	1	DUF5916	0.00092
Cluster 333	6	5	0	6	BBP2, DUF2490, OMP_b-brl_2, OmpA_membrane, Porin_4	2.8e-05
Cluster 376	2	2	2	0	-	None
Cluster 378	4	3	4	0	-	None
Cluster 389	2	2	2	0	-	None
Cluster 390	2	2	2	0	-	None
Cluster 399	5	5	5	0	-	None

Cluster 404	5	4	5	0	-	None
Cluster 408	5	5	5	0	-	None
Cluster 419	4	3	4	0	-	None
Cluster 437	4	3	4	0	-	None
Cluster 442	3	3	3	0	-	None
Cluster 494	4	3	4	0	OMPdbModel29, YadA_anchor	3.5e-08
Cluster 506	2	2	0	2	-	None
Cluster 534	2	2	2	0	OMPdbModel29	1.8e-06
Cluster 541	2	2	2	0	-	None
Cluster 543	2	2	0	2	-	None
Cluster 548	4	3	4	0	OMPdbModel29	0.00017
Cluster 562	5	5	5	0	OMPdbModel29	3.5e-13
Cluster 563	7	5	7	0	DUF2490, OMPdbModel29	3.9e-09
Cluster 586	2	2	2	0	-	None
Cluster 587	11	11	11	0	DUF2490, OMPdbModel29	1.6e-08
Cluster 600	27	19	27	0	-	None
Cluster 607	14	12	14	0	OMPdbModel29	7.3e-07
Cluster 621	3	3	3	0	OMPdbModel29	4e-08
Cluster 647	4	4	4	0	OMPdbModel29	4.3e-06
Cluster 664	3	3	3	0	OMPdbModel29	2.1e-08
Cluster 673	5	4	5	0	-	None
Cluster 682	8	7	8	0	OMPdbModel29	1.9e-07
Cluster 685	2	2	2	0	-	None
Cluster 687	3	3	3	0	-	None
Cluster 700	4	3	4	0	OMPdbModel29	1.4e-08
Cluster 715	5	4	5	0	OMPdbModel29	1.3e-05
Cluster 803	2	2	2	0	OMPdbModel29	8.1e-07
Cluster 826	3	3	3	0	-	None
Cluster 851	3	3	3	0	-	None
Cluster 890	4	3	4	0	-	None

Cluster 930	2	2	0	2	-	None
Cluster 937	4	4	4	0	-	None
Cluster 976	3	3	3	0	-	None
Cluster 1146	2	2	2	0	-	None
Cluster 1171	3	3	3	0	-	None
Cluster 1182	59	43	59	0	OMPdbModel29	2.6e-06
Cluster 1267	3	3	3	0	OMP_b-brl_3	0.00024
Cluster 1310	34	32	34	0	-	None
Cluster 1366	5	5	3	2	OMPdbModel30	2.3e-18
Cluster 1428	9	8	9	0	OMPdbModel29	4.4e-06
Cluster 1475	2	2	2	0	OMPdbModel29	8.2e-05
Cluster 1482	2	2	2	0	OMPdbModel29	0.00047
Cluster 1497	4	4	4	0	-	None
Cluster 1501	2	2	2	0	-	None
Cluster 1528	11	8	11	0	OMPdbModel29	1.2e-23
Cluster 1530	4	3	4	0	OMPdbModel29	6.4e-13
Cluster 1534	4	3	4	0	-	None
Cluster 1536	3	3	3	0	OMPdbModel29	4.4e-26
Cluster 1537	9	7	9	0	OMPdbModel29	1.1e-17
Cluster 1541	2	2	2	0	-	None
Cluster 1549	4	3	4	0	OMP_b-brl_3, OMPdbModel29	3e-24
Cluster 1552	10	9	10	0	OMP_b-brl_3, OMPdbModel29	6e-27
Cluster 1557	2	2	2	0	-	None
Cluster 1559	3	3	3	0	-	None
Cluster 1581	2	2	2	0	-	None
Cluster 1585	20	17	20	0	DUF3575, OMP_b-brl_3, OMPdbModel29, Porin_2	1e-29
Cluster 1586	5	4	5	0	DUF5777	0.00065
Cluster 1588	4	3	4	0	OMP_b-brl	0.012

Cluster 1596	3	3	3	0	OMPdbModel29	7.1e-21
Cluster 1604	21	20	21	0	DUF481, OMPdbModel29, Porin_2	2.8e-27
Cluster 1611	3	3	3	0	OMPdbModel29	4.5e-12
Cluster 1618	2	2	0	2	-	None
Cluster 1622	2	2	2	0	-	None
Cluster 1626	2	2	2	0	OMP_b-brl_3, OMPdbModel29, Phenol_MetA_deg	4.1e-07
Cluster 1632	3	3	3	0	-	None
Cluster 1639	5	4	5	0	-	None
Cluster 1650	2	2	2	0	-	None
Cluster 1657	4	3	4	0	DUF6089, OMP_b-brl, OMPdbModel30	2.6e-07
Cluster 1682	23	18	23	0	OMP_b-brl_3	8.3e-08
Cluster 1714	5	4	5	0	OMPdbModel29	2.2e-06
Cluster 1715	4	4	4	0	-	None
Cluster 1732	2	2	0	2	-	None
Cluster 1793	14	10	14	0	OMPdbModel29	1.6e-20
Cluster 1813	3	3	3	0	OMPdbModel29	0.00039
Cluster 1817	9	8	9	0	OMP_b-brl, OMP_b-brl_3	1.3e-07
Cluster 1818	3	3	3	0	-	None
Cluster 1832	2	2	2	0	-	None
Cluster 1841	5	5	5	0	-	None
Cluster 1905	2	2	0	2	-	None
Cluster 1911	4	3	4	0	-	None
Cluster 1916	2	2	2	0	-	None
Cluster 1922	5	5	5	0	-	None
Cluster 1923	2	2	2	0	OMPdbModel29	5.7e-05
Cluster 1965	5	4	5	0	-	None
Cluster 1970	2	2	2	0	-	None

Cluster 1973	5	4	5	0	-	None
Cluster 2001	17	13	17	0	OMPdbModel29	7.7e-31
Cluster 2032	2	2	2	0	-	None
Cluster 2062	3	3	3	0	-	None
Cluster 2065	2	2	2	0	OMP_b-brl, OMPdbModel29	5.7e-14
Cluster 2081	4	4	4	0	-	None
Cluster 2099	2	2	2	0	-	None
Cluster 2120	5	4	5	0	OMPdbModel29	3.8e-20
Cluster 2159	2	2	2	0	-	None
Cluster 2196	28	21	28	0	OMP_b-brl, OMP_b-brl_3, OMPdbModel29	3.5e-22
Cluster 2203	2	2	2	0	-	None
Cluster 2311	7	5	7	0	OMPdbModel29	7e-10
Cluster 2329	5	4	5	0	OMPdbModel29	6.3e-07
Cluster 2345	2	2	2	0	-	None
Cluster 2350	4	3	4	0	-	None
Cluster 2355	3	3	3	0	-	None
Cluster 2362	4	3	4	0	-	None
Cluster 2365	4	3	4	0	-	None
Cluster 2373	6	5	6	0	-	None
Cluster 2406	7	6	7	0	-	None
Cluster 2460	3	3	3	0	-	None
Cluster 2491	2	2	2	0	-	None
Cluster 2492	3	3	3	0	-	None
Cluster 2493	3	3	3	0	-	None
Cluster 2526	5	4	5	0	OMP_b-brl	0.028
Cluster 2528	4	3	4	0	OMP_b-brl_3	5.3e-06
Cluster 2531	2	2	2	0	-	None
Cluster 2553	2	2	2	0	-	None
Cluster 2597	5	4	5	0	-	None
Cluster 2633	2	2	2	0	-	None
Cluster 2643	2	2	2	0	-	None

Cluster 2649	3	3	3	0	-	None
Cluster 2650	4	3	4	0	DUF3570, OMP_b-brl_3, OMPdbModel29	2.2e-14
Cluster 2692	4	3	4	0	-	None
Cluster 2695	2	2	2	0	OMPdbModel29	9.8e-33
Cluster 2712	3	3	3	0	-	None
Cluster 2738	2	2	2	0	-	None
Cluster 2741	3	3	3	0	-	None
Cluster 2774	3	3	0	3	-	None
Cluster 2800	5	4	5	0	-	None
Cluster 2808	4	4	4	0	OMPdbModel29	4.7e-11
Cluster 2839	3	3	3	0	-	None
Cluster 2883	4	3	4	0	-	None
Cluster 2909	8	7	8	0	OMPdbModel29	1.2e-37
Cluster 2916	2	2	2	0	-	None
Cluster 2948	2	2	2	0	-	None
Cluster 2959	2	2	2	0	OMP_b-brl_3	1.1e-07
Cluster 2962	3	3	3	0	LptD	4.4e-05
Cluster 2987	7	6	7	0	OMP_b-brl	0.0072
Cluster 2992	2	2	2	0	OMP_b-brl	0.019
Cluster 3064	3	3	3	0	OMP_b-brl, OMP_b-brl_2	0.00045
Cluster 3065	2	2	2	0	-	None
Cluster 3093	4	3	4	0	OMPdbModel29	0.019
Cluster 3097	2	2	0	2	DUF2490, KdgM	0.00033
Cluster 3102	14	10	14	0	OMPdbModel29	1.1e-26
Cluster 3139	2	2	2	0	-	None
Cluster 3190	7	7	7	0	OMP_b-brl_3	0.0018
Cluster 3191	27	20	27	0	DUF2490, OMP_b-brl_3, OMPdbModel29	1e-09
Cluster 3272	2	2	2	0	OMP_b-brl_3, OMPdbModel29	6e-12
Cluster 3384	4	3	4	0	OMP_b-brl	0.0015

Cluster 3430	3	3	3	0	OMPdbModel29	9.4e-10
Cluster 3444	2	2	2	0	-	None
Cluster 3466	3	3	3	0	-	None
Cluster 3513	4	4	4	0	-	None
Cluster 3546	60	49	60	0	OMP_b-brl_3, OMPdbModel29	2.5e-11
Cluster 3566	5	4	5	0	-	None
Cluster 3607	5	4	5	0	Omp85_2	3.2e-05
Cluster 3650	4	4	4	0	-	None
Cluster 3669	3	3	3	0	-	None
Cluster 3673	8	7	8	0	OMP_b-brl_3, OMPdbModel29	1.6e-25
Cluster 3675	4	3	4	0	-	None
Cluster 3698	3	3	3	0	TraF_2	0.00059
Cluster 3733	4	4	4	0	-	None
Cluster 3743	2	2	2	0	DUF2490, Porin_2	0.00027
Cluster 3764	2	2	2	0	-	None
Cluster 3765	2	2	2	0	-	None
Cluster 3795	8	6	8	0	OMP_b-brl_3	0.00042
Cluster 3841	2	2	2	0	OMP_b-brl_3	1.6e-05
Cluster 3859	2	2	2	0	-	None
Cluster 3867	4	3	4	0	OMP_b-brl	0.0057
Cluster 3868	11	8	11	0	OMP_b-brl_3, OMPdbModel29, OMPdbModel30	1.5e-05
Cluster 3870	2	2	2	0	OMP_b-brl	0.003
Cluster 3878	3	3	3	0	OMP_b-brl_3, OMPdbModel29	4.5e-12
Cluster 3946	9	7	9	0	OMP_b-brl_3, OMPdbModel29	1.1e-12
Cluster 3991	2	2	2	0	OMPdbModel29	0.039
Cluster 3996	2	2	1	1	OMPdbModel30	1.9e-21
Cluster 4013	3	3	3	0	OMP_b-brl_3,	2.8e-18

					OMPdbModel29	
Cluster 4028	4	3	4	0	-	None
Cluster 4041	7	5	7	0	OMPdbModel29	1.8e-31
Cluster 4077	2	2	2	0	-	None
Cluster 4086	2	2	2	0	OMP_b-brl_3	1.3e-07
Cluster 4100	5	5	5	0	OMPdbModel29	1.1e-18
Cluster 4116	2	2	2	0	-	None
Cluster 4143	8	6	8	0	-	None
Cluster 4154	4	3	4	0	OMPdbModel29	8.7e-16
Cluster 4171	3	3	3	0	-	None
Cluster 4174	4	4	4	0	DUF2490	0.0091
Cluster 4185	9	7	9	0	-	None
Cluster 4187	2	2	0	2	-	None
Cluster 4189	2	2	2	0	-	None
Cluster 4194	2	2	2	0	-	None
Cluster 4200	8	7	8	0	OMP_b-brl_3, OMPdbModel29	4.7e-27
Cluster 4218	2	2	2	0	-	None
Cluster 4225	4	4	4	0	-	None
Cluster 4237	5	4	5	0	OMPdbModel29	2.7e-39
Cluster 4238	3	3	3	0	OMPdbModel29	1.7e-22
Cluster 4243	15	12	15	0	Caps_assemb_Wzi, OMPdbModel29	5.3e-35
Cluster 4246	2	2	2	0	-	None
Cluster 4251	2	2	2	0	-	None
Cluster 4261	2	2	2	0	OMP_b-brl_3, OMPdbModel29	2.7e-24
Cluster 4263	3	3	3	0	-	None
Cluster 4270	2	2	2	0	-	None
Cluster 4278	4	4	4	0	-	None
Cluster 4310	2	2	2	0	-	None
Cluster 4322	3	3	3	0	OMPdbModel29	5e-24
Cluster 4338	3	3	3	0	OMPdbModel29	1.3e-05

Cluster 4343	2	2	2	0	-	None
Cluster 4353	3	3	3	0	-	None
Cluster 4400	5	5	5	0	-	None
Cluster 4477	2	2	2	0	OMP_b-brl_3, OMPdbModel29	5.2e-31
Cluster 4485	14	12	14	0	OMP_b-brl, OMPdbModel29, OprF	1.2e-23
Cluster 4496	4	3	4	0	-	None
Cluster 4505	2	2	2	0	DUF2490, LamB	0.00058
Cluster 4515	2	2	2	0	DUF2490, Porin_4	2.1e-05
Cluster 4522	7	6	7	0	-	None
Cluster 4542	4	4	4	0	BBP2_2, OMP_b-brl_3, OMPdbModel29, OMPdbModel3	1.3e-24
Cluster 4581	2	2	2	0	OMPdbModel30	4.1e-05
Cluster 4599	2	2	0	2	-	None
Cluster 4618	2	2	2	0	-	None
Cluster 4633	4	4	4	0	OMPdbModel29	1.7e-09
Cluster 4682	2	2	2	0	OMPdbModel29	6.4e-32
Cluster 4700	4	4	4	0	-	None
Cluster 4706	2	2	0	2	-	None
Cluster 4719	2	2	2	0	PagL	0.0014
Cluster 4720	2	2	2	0	-	None
Cluster 4724	4	3	0	4	-	None
Cluster 4737	2	2	2	0	Porin_4	0.00036
Cluster 4834	2	2	2	0	OMPdbModel29	2e-34
Cluster 4854	8	6	8	0	DUF2490, OMP_b-brl_3, OMPdbModel29, Porin_2	4.2e-22
Cluster 4882	4	3	4	0	OMP_b-brl_3, OMPdbModel29	5e-12

Cluster 4900	3	3	3	0	OMP_b-brl	0.041
Cluster 4917	4	4	4	0	-	None
Cluster 4968	3	3	3	0	-	None
Cluster 5001	12	11	12	0	DUF481, OMPdbModel29	2.5e-31
Cluster 5078	2	2	2	0	-	None
Cluster 5118	2	2	2	0	-	None
Cluster 5121	2	2	0	2	-	None
Cluster 5170	2	2	2	0	OMPdbModel29	9.7e-20
Cluster 5179	2	2	2	0	-	None
Cluster 5183	10	10	10	0	MipA, OMP_b-brl_3, OMPdbModel29	1.4e-32
Cluster 5198	2	2	2	0	-	None
Cluster 5219	4	4	0	4	OMPdbModel30	1.7e-07
Cluster 5255	2	2	2	0	-	None
Cluster 5286	2	2	0	2	-	None
Cluster 5311	3	3	3	0	-	None
Cluster 5343	5	5	5	0	-	None
Cluster 5348	2	2	2	0	-	None
Cluster 5387	2	2	1	1	-	None
Cluster 5404	2	2	2	0	-	None
Cluster 5446	7	5	7	0	OMP_b-brl_3	9.5e-05
Cluster 5532	4	4	4	0	-	None
Cluster 5606	2	2	0	2	-	None
Cluster 5635	2	2	2	0	YadA_anchor	0.0011
Cluster 5639	6	6	6	0	OMP_b-brl_3	0.00012
Cluster 5671	2	2	2	0	OMPdbModel9	0.0071
Cluster 5672	2	2	2	0	OMPdbModel30	0.084
Cluster 5684	2	2	2	0	-	None
Cluster 5764	2	2	2	0	Porin_4	1.5e-05
Cluster 5829	2	2	2	0	OMP_b-brl_3	4.9e-06
Cluster 5905	8	8	8	0	-	None
Cluster 5939	2	2	2	0	-	None

Cluster 5993	2	2	2	0	-	None
Cluster 6022	2	2	2	0	-	None
Cluster 6056	2	2	0	2	OMPdbModel9	0.0022
Cluster 6083	2	2	2	0	-	None
Cluster 6170	4	3	4	0	ShlB	0.00039
Cluster 6210	2	2	2	0	-	None
Cluster 6253	63	56	63	0	OMPdbModel29, PagL	9.8e-05
Cluster 6372	2	2	2	0	-	None
Cluster 6421	8	8	8	0	DUF2490, OMP_b-brl, Porin_4, Porin_O_P	3.2e-05
Cluster 6490	2	2	2	0	OMP_b-brl_3	2.8e-05
Cluster 6566	4	4	4	0	YadA_anchor	0.035
Cluster 6708	7	5	7	0	-	None
Cluster 6731	2	2	2	0	-	None
Cluster 6740	14	10	14	0	OMP_b-brl, OMP_b-brl_2, OMPdbModel11	8e-05
Cluster 6741	2	2	2	0	OMPdbModel29	2e-25
Cluster 6765	2	2	2	0	-	None
Cluster 6858	2	2	2	0	BBP2, OMP_b-brl	0.00037
Cluster 6895	2	2	2	0	-	None
Cluster 6921	2	2	2	0	DUF2490, OMP_b-brl, Porin_4	0.00016
Cluster 6947	2	2	2	0	DUF2490, OMP_b-brl	8.4e-05
Cluster 6976	9	8	9	0	OMP_b-brl	0.00015
Cluster 7003	5	4	5	0	-	None
Cluster 7066	7	6	7	0	BBP2, DUF3187	5.6e-05
Cluster 7126	2	2	2	0	ShlB	1e-07
Cluster 7143	2	2	2	0	-	None
Cluster 7202	2	2	2	0	-	None
Cluster 7265	2	2	2	0	-	None
Cluster 7276	2	2	0	2	-	None
Cluster 7326	10	8	10	0	Phenol_MetA_deg	2.3e-06
Cluster 7346	2	2	2	0	-	None

Cluster 7349	11	9	11	0	DUF3187, OMP_b-brl, Phenol_MetA_deg, Porin_4	1.2e-06
Cluster 7373	2	2	2	0	OMP_b-brl, OprB	2.3e-06
Cluster 7416	13	12	13	0	OMP_b-brl, YfaZ	0.00011
Cluster 7449	3	3	3	0	OMPdbModel28	5.5e-13
Cluster 7486	4	3	4	0	-	None
Cluster 7540	5	5	5	0	-	None
Cluster 7554	5	4	5	0	-	None
Cluster 7589	2	2	2	0	-	None
Cluster 7609	15	15	15	0	DUF2490, OMP_b-brl, OprB	0.0002
Cluster 7683	7	5	7	0	-	None
Cluster 7727	2	2	2	0	-	None
Cluster 7846	4	4	1	3	DUF481, OMP_b-brl	1.6e-07
Cluster 8032	3	3	3	0	-	None
Cluster 8040	24	22	24	0	OMP_b-brl, OMPdbModel11, PagL	1e-06
Cluster 8050	2	2	2	0	-	None
Cluster 8062	2	2	2	0	-	None
Cluster 8114	4	4	4	0	-	None
Cluster 8147	15	11	15	0	DUF3575, DUF6089, MipA, OMP_b-brl, OMP_b-brl_2, OMPdbModel	8.5e-06
Cluster 8198	2	2	2	0	-	None
Cluster 8201	3	3	3	0	-	None
Cluster 8202	5	4	5	0	Porin_5	7.4e-07
Cluster 8264	2	2	2	0	DUF6089, OMP_b-brl	6.1e-05
Cluster 8273	5	4	5	0	-	None
Cluster 8285	2	2	2	0	-	None
Cluster 8323	4	3	4	0	Campylo_MOMP, Porin_5	6.9e-07

Cluster 8344	2	2	1	1	-	None
Cluster 8386	3	3	3	0	Porin_4	0.00063
Cluster 8397	2	2	1	1	-	None
Cluster 8487	14	11	14	0	DUF481, OMPdbModel28, ShlB	3.2e-07
Cluster 8497	7	5	7	0	Caps_assemb_Wzi, DUF2490, Porin_5, Porin_O_P	2.7e-06
Cluster 8540	4	4	4	0	OMP_b-brl, OMP_b-brl_2	2.2e-07
Cluster 8600	20	20	20	0	OMP_b-brl, OMP_b-brl_2, OMPdbModel11, OMPdbModel30, OmpA_m	2.2e-06
Cluster 8602	4	4	4	0	-	None
Cluster 8628	2	2	2	0	-	None
Cluster 8643	2	2	2	0	OMPdbModel29	5.5e-07
Cluster 8754	8	6	8	0	-	None
Cluster 8812	5	4	5	0	BBP2_2, DUF481	3.3e-06
Cluster 8842	3	3	3	0	-	None
Cluster 8941	5	4	5	0	OMP_b-brl, PagL	1.6e-06
Cluster 8958	2	2	2	0	OMPdbModel30	0.016
Cluster 8978	5	4	5	0	Caps_assemb_Wzi	0.0015
Cluster 8990	6	6	6	0	OMPdbModel29	0.00022
Cluster 8999	62	53	62	0	DUF6089, OMP_b-brl, OMP_b-brl_2, OMPdbModel11, OMPdbModel3	2.6e-09
Cluster 9000	3	3	3	0	Porin_O_P	0.00022
Cluster 9016	2	2	2	0	-	None
Cluster 9052	12	11	12	0	DUF2490, OMP_b-brl, Porin_4, Porin_5	3.4e-06
Cluster 9063	59	42	59	0	DUF1302, DUF2490,	7.4e-09

					DUF6089, OMP_b-brl, OMP_b-brl_2, OMPdbMo	
Cluster 9166	18	17	18	0	Caps_assemb_Wzi, DUF2490, DUF3078, OMP_b-brl, OMPdbModel20	1.5e-07
Cluster 9186	2	2	2	0	Porin_5	7.6e-06
Cluster 9208	2	2	2	0	Porin_5	3.8e-06
Cluster 9245	17	14	17	0	Caps_assemb_Wzi, DUF2490, DUF3373, DUF481, LptD, Porin_4,	2.4e-09
Cluster 9357	3	3	0	3	-	None
Cluster 9391	4	3	4	0	OMP_b-brl	2.9e-06
Cluster 9471	13	11	13	0	OMP_b-brl, OMPdbModel18, Porin_4, Porin_5	3.7e-08
Cluster 9529	4	4	4	0	-	None
Cluster 9656	6	6	6	0	DUF2490, OMP_b-brl, Porin_5, Porin_O_P	8.2e-07
Cluster 9659	5	5	5	0	-	None
Cluster 9675	6	6	6	0	DUF3373, Porin_5	0.00011
Cluster 9759	4	4	4	0	OMPdbModel28, ShlB	2e-06
Cluster 9790	73	61	73	0	-	None
Cluster 9808	4	4	4	0	DUF2490	0.019
Cluster 9847	2	2	2	0	MtrB_PioB, OMP_b-brl_3, OMPdbModel29	4.1e-18
Cluster 9854	2	2	2	0	-	None
Cluster 9864	2	2	2	0	-	None
Cluster 9937	2	2	0	2	-	None
Cluster 9966	10	9	10	0	Campylo_MOMP, DUF2490, DUF3373, Porin_4, Porin_5	8.4e-06

Cluster 9977	2	2	2	0	ShlB	1.6e-05
Cluster 10008	2	2	0	2	-	None
Cluster 10056	4	4	0	4	DUF5723, OMPdbModel30	4.5e-06
Cluster 10082	4	3	4	0	-	None
Cluster 10102	2	2	2	0	-	None
Cluster 10109	3	3	3	0	OMP_b-brl, OMP_b-brl_2, OMPdbModel11	4.4e-09
Cluster 10166	4	4	4	0	OMP_b-brl	1.7e-07
Cluster 10210	2	2	0	2	-	None
Cluster 10249	38	32	38	0	BBP2_2, DUF481, DUF5777, OMP_b-brl, OMP_b-brl_2, OMPdbMode	2.6e-08
Cluster 10338	3	3	3	0	ShlB	1.1e-05
Cluster 10374	2	2	2	0	-	None
Cluster 10390	2	2	2	0	-	None
Cluster 10490	45	32	45	0	DUF3034, OMP_b-brl, OMPdbModel27, OMPdbModel28, Omp85_2, S	2.9e-07
Cluster 10491	9	9	9	0	OMP_b-brl, Porin_4, Porin_5	3.1e-07
Cluster 10504	3	3	3	0	-	None
Cluster 10529	2	2	2	0	OMP_b-brl, SlipAM	2.7e-05
Cluster 10544	4	3	4	0	OMP_b-brl, OMP_b-brl_2	5.5e-08
Cluster 10552	2	2	0	2	OMPdbModel30	1.6e-34
Cluster 10562	2	2	2	0	BBP2	3.5e-05
Cluster 10820	4	4	0	4	OMPdbModel1	0.00047
Cluster 10874	18	15	18	0	DUF481, OMP_b-brl, Porin_5	3e-06
Cluster 10919	2	2	2	0	DUF481, SlipAM	1.4e-05
Cluster 10935	2	2	0	2	-	None

Cluster 11007	2	2	2	0	SlipAM	2.6e-05
Cluster 11026	15	14	15	0	DUF5777, OMP_b-brl, OMP_b-brl_2	4.9e-09
Cluster 11043	2	2	2	0	-	None
Cluster 11056	4	4	3	1	-	None
Cluster 11059	3	3	3	0	-	None
Cluster 11103	2	2	2	0	SlipAM	2.9e-05
Cluster 11121	8	6	8	0	OMP_b-brl	3.5e-09
Cluster 11140	9	8	9	0	LptD, OMPdbModel30	7.3e-05
Cluster 11158	2	2	2	0	-	None
Cluster 11177	3	3	3	0	CBP_BcsS, Serpulina_VSP	7.5e-05
Cluster 11199	2	2	2	0	BCSC_C	2.7e-05
Cluster 11352	2	2	2	0	-	None
Cluster 11450	5	4	5	0	Channel_Tsx, DUF6089, DUF6268, OMP_b-brl, OMPdbModel11, Pa	2.1e-08
Cluster 11544	3	3	3	0	OMPdbModel29	1.4e-20
Cluster 11565	2	2	2	0	BBP2_2, DUF2860, DUF481	1e-05
Cluster 11668	2	2	2	0	-	None
Cluster 11721	19	14	19	0	OMP_b-brl, OMPdbModel27, OMPdbModel28, ShlB, Toluene_X	3.2e-08
Cluster 11775	3	3	3	0	-	None
Cluster 11807	2	2	2	0	SlipAM	3.3e-13
Cluster 11829	4	3	4	0	-	None
Cluster 11841	2	2	2	0	-	None
Cluster 11948	7	5	7	0	OMP_b-brl, SlipAM	0.0017
Cluster 11955	2	2	2	0	-	None
Cluster 11959	4	4	4	0	-	None

Cluster 11961	2	2	2	0	-	None
Cluster 11973	2	2	2	0	-	None
Cluster 12088	13	11	13	0	-	None
Cluster 12123	6	5	6	0	BBP2_2, DUF2860, OMP_b-brl, Opacity, SlipAM	5.6e-08
Cluster 12133	6	6	6	0	Porin_4	8.3e-05
Cluster 12136	2	2	2	0	-	None
Cluster 12164	2	2	2	0	OMP_b-brl, OMP_b-brl_2, OMPdbModel11, OmpA_membrane	4e-07
Cluster 12258	5	5	5	0	-	None
Cluster 12375	2	2	2	0	BCSC_C, Porin_4	5e-07
Cluster 12403	2	2	1	1	BBP2_2, DUF2490, DUF2860, DUF481	3.8e-06
Cluster 12499	2	2	2	0	BBP2_2, DUF2860, SlipAM	9.7e-06
Cluster 12532	2	2	2	0	ShlB	1.1e-06
Cluster 12633	2	2	2	0	DUF2860	1.6e-05
Cluster 12701	3	3	3	0	OMPdbModel29	0.015
Cluster 12702	2	2	2	0	-	None
Cluster 12729	2	2	2	0	-	None
Cluster 12756	2	2	2	0	-	None
Cluster 12922	4	4	3	1	-	None
Cluster 12930	2	2	2	0	OMPdbModel30	5.6e-12
Cluster 12948	2	2	2	0	OMP_b-brl	4.5e-08
Cluster 12957	2	2	2	0	-	None
Cluster 13046	2	2	2	0	-	None
Cluster 13067	2	2	2	0	OMP_b-brl	0.0042
Cluster 13102	43	36	43	0	DUF3187, DUF4421, OMP_b-brl, OMP_b-brl_2, OMPdbModel11, OM	3.8e-15

Cluster 13121	2	2	2	0	-	None
Cluster 13208	2	2	2	0	OMPdbModel29	1.6e-20
Cluster 13265	7	6	6	1	-	None
Cluster 13307	3	3	3	0	-	None
Cluster 13313	2	2	2	0	BCSC_C, LamB, Porin_4	7.4e-06
Cluster 13381	3	3	3	0	OEP	0.0016
Cluster 13467	2	2	2	0	-	None
Cluster 13555	2	2	2	0	Porin_5	0.0006
Cluster 13595	2	2	0	2	MtrB_PioB, OMP_b-brl, YfaZ	7.7e-06
Cluster 13600	7	6	7	0	-	None
Cluster 13676	15	14	15	0	-	None
Cluster 13704	21	20	21	0	OEP, OMP_b-brl, OMPdbModel30, PagL, Surface_Ag_2, YadA_anc	6.4e-08
Cluster 13724	2	2	2	0	BBP2_2, DUF2860	9.5e-07
Cluster 13830	21	19	21	0	OMP_b-brl, OMPdbModel30, Opacity, Surface_Ag_2	3.1e-06
Cluster 13873	2	2	0	2	-	None
Cluster 13876	21	19	21	0	DUF481, OMP_b-brl, OMPdbModel30, OprF, PagL, Phenol_MetA_d	5.2e-08
Cluster 13989	4	3	4	0	-	None
Cluster 14019	2	2	2	0	Surface_Ag_2	3.3e-06
Cluster 14037	4	3	4	0	DUF481, OMP_b-brl, OMP_b-brl_2, OMPdbModel11, PagL	1.7e-08
Cluster 14088	2	2	1	1	-	None
Cluster 14181	4	3	4	0	-	None

Cluster 14329	4	3	4	0	OMP_b-brl, PagL	1.8e-08
Cluster 14336	3	3	3	0	-	None
Cluster 14356	4	4	4	0	-	None
Cluster 14365	7	5	7	0	OMP_b-brl, OMPdbModel11	1.6e-15
Cluster 14369	8	7	8	0	OMP_b-brl, OMPdbModel11, OmpA_membrane	2.7e-17
Cluster 14615	3	3	3	0	Porin_5	0.00024
Cluster 14681	2	2	2	0	-	None
Cluster 14773	2	2	0	2	-	None
Cluster 14799	2	2	2	0	-	None
Cluster 14948	3	3	3	0	OMP_b-brl, OMPdbModel11, Opacity, PagL	2.1e-16
Cluster 15012	4	3	4	0	-	None
Cluster 15716	2	2	2	0	DUF2860, OMP_b-brl, SlipAM	2.8e-06
Cluster 15858	2	2	2	0	-	None
Cluster 15888	5	4	5	0	OMPdbModel30, OmpA_membrane	7e-06
Cluster 16311	3	3	3	0	-	None
Cluster 16339	2	2	2	0	-	None
Cluster 16539	2	2	2	0	-	None
Cluster 16642	2	2	2	0	-	None
Cluster 16737	2	2	2	0	DUF481, OMPdbModel30	4.8e-05
Cluster 17068	3	3	3	0	OMP_b-brl, OMP_b-brl_2, Phenol_MetA_deg	1.6e-07
Cluster 17277	4	4	4	0	Attacin_C, OMP_b-brl	1.1e-11
Cluster 17666	2	2	1	1	-	None
Cluster 17803	4	4	4	0	OMP_b-brl	0.024
Cluster 17899	4	4	4	0	DUF6089, OMP_b-brl,	6.2e-08

					OprF, SlipAM	
Cluster 17950	2	2	2	0	BBP2_2	2.8e-05
Cluster 17960	2	2	2	0	OMP_b-brl, OMPdbModel11, OMPdbModel16	4.3e-20
Cluster 18164	3	3	3	0	OMP_b-brl, OMPdbModel30, OprF, PagL	9.7e-08
Cluster 18354	7	6	7	0	Campylo_MOMP, MipA, OMP_b-brl, OprF, YaiO	1.5e-07
Cluster 18707	3	3	3	0	-	None
Cluster 18892	2	2	0	2	OMPdbModel30	3.5e-33
Cluster 19028	2	2	2	0	OMP_b-brl, OMPdbModel29	4.6e-05
Cluster 19064	2	2	2	0	BBP2, OMP_b-brl, Porin_4	0.00083
Cluster 19165	3	3	3	0	OMP_b-brl, OMPdbModel30, Porin_4	2.8e-08
Cluster 19303	2	2	2	0	-	None
Cluster 19409	5	5	5	0	OMPdbModel29, YaiO	8e-08
Cluster 19706	2	2	2	0	OMP_b-brl	3.2e-06
Cluster 19759	4	3	4	0	DUF481, OMP_b-brl, OMPdbModel30, OmpA_membrane, OprF	5.2e-09
Cluster 20015	4	3	4	0	OMP_b-brl, OprB, Porin_4, YaiO	5.1e-07
Cluster 20128	2	2	2	0	DUF481, OMP_b-brl, OmpA_membrane, OprF	4.5e-09
Cluster 20268	2	2	2	0	-	None
Cluster 20774	2	2	0	2	-	None

Cluster 21086	3	3	3	0	OMP_b-brl, OMPdbModel11, OMPdbModel30, Phenol_MetA_deg	5.1e-11
Cluster 21249	2	2	2	0	OMP_b-brl, OprF	3.6e-07
Cluster 21410	2	2	2	0	-	None
Cluster 22085	2	2	2	0	-	None
Cluster 22109	2	2	2	0	OMP_b-brl, OMP_b-brl_2, OMPdbModel11, OmpW, OprF, PagL	1.9e-09
Cluster 22429	2	2	0	2	-	None
Cluster 23403	2	2	1	1	TraF_2	1.6e-05
Cluster 23429	2	2	0	2	-	None
Cluster 24753	2	2	0	2	-	None
Cluster 25087	2	2	2	0	-	None
Cluster 27449	2	2	2	0	-	None
Cluster 28192	2	2	0	2	-	None
Cluster 28415	2	2	2	0	-	None
Cluster 28466	2	2	2	0	-	None
Cluster 30593	2	2	0	2	-	None
Cluster 32444	22	16	22	0	OMP_b-brl, OmpW	1.7e-06
Cluster 53509	2	2	0	2	-	None

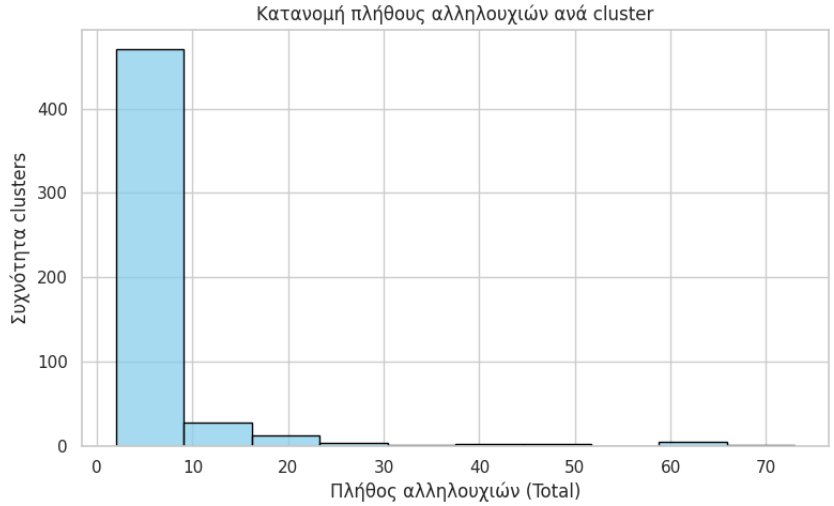
Πίνακας 3.2 Αναλυτικά χαρακτηριστικά των clusters (>70% β-βαρέλια) με βάση PRED-TMBB2, HMMER (E-value < 0.1) και προβλέψεις δομής από AlphaFold.

Από το σύνολο των 525 πρωτεϊνικών clusters που εντοπίστηκαν, στις 295 (56,2%) δεν εντοπίστηκε καμία ασθενή συσχέτιση με β-βαρέλι. Αντίθετα, στα 266 clusters (43,8%) ανιχνεύθηκε τουλάχιστον μία συσχέτιση με γνωστή δομή β-βαρέλι. Επιπλέον, οι 482 από τις 525 ομάδες είχαν έστω μία κενή περιοχή με δομή βαρελιού στην AlphaFold.

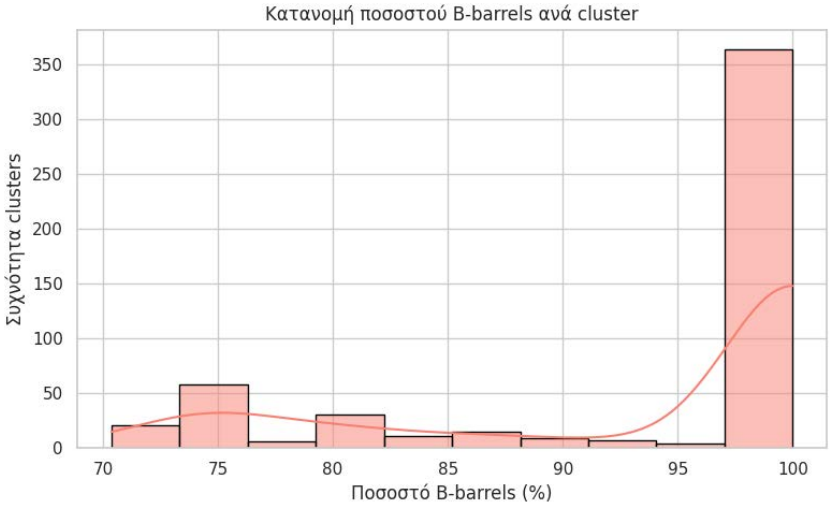
Παρακάτω παρουσιάζονται τρία ιστογράμματα συχνοτήτων για διαφορετικά αποτελέσματα των χαρακτηριστικών των ομάδων.

1. Το πρώτο γράφημα απεικονίζει τον αριθμό των clusters σε σχέση με το πλήθος των αλληλουχιών που περιέχουν. Παρατηρούμε ότι τα περισσότερα clusters περιλαμβάνουν λιγότερες από 10 αλληλουχίες.

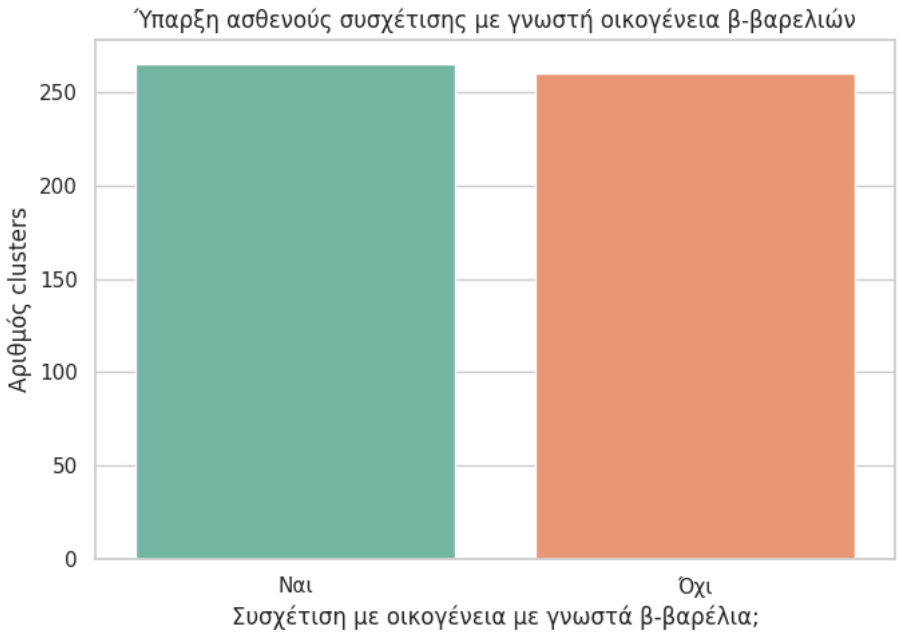
2. Το δεύτερο γράφημα απεικονίζει το ποσοστό των β-barrel δομών σε σχέση με το σύνολο των αλληλουχιών για κάθε cluster. Παρατηρούμε ότι η μεγάλη πλεοψηφία των clusters έχει ποσοστό β-βαρέλια προς σύνολο ακουλουθιών 100%, γεγονός που οφείλεται εν μέρει στην ομοιότητα των αλληλουχιών των β-βαρελιών, η οποία οδηγεί στην πρόβλεψη ότι πρόκειται για β-βαρέλια από το εργαλείο PRED-TMBB2. Αλλά κυρίως το υψηλό αυτό ποσοστό εξηγείται και από το γεγονός ότι τα περισσότερα clusters αποτελούνται από πολύ μικρό αριθμό πρωτεϊνικών αλληλουχιών (συνήθως δύο), γεγονός που αυξάνει την πιθανότητα όλες οι αλληλουχίες του cluster να είναι β-βαρέλια.
3. Το τρίτο γράφημα καταγράφει πόσα clusters εμφανίζουν ή δεν εμφανίζουν ομοιότητα με γνωστές οικογένειες β-barrel domains.



Εικόνα 3.1 Ιστόγραμμα συχνοτήτων για κατανομή των clusters ως προς το πλήθος των αλληλουχιών τους.



Εικόνα 3.2 Ιστόγραμμα συχνοτήτων για κατανομή του ποσοστού πιθανών β-barrel ανά cluster.



Εικόνα 3.3 Ιστόγραμμα συχνοτήτων για το πόσα clusters εμφανίζουν ασθενή συσχέτιση με γνωστές β-barrel δομές.

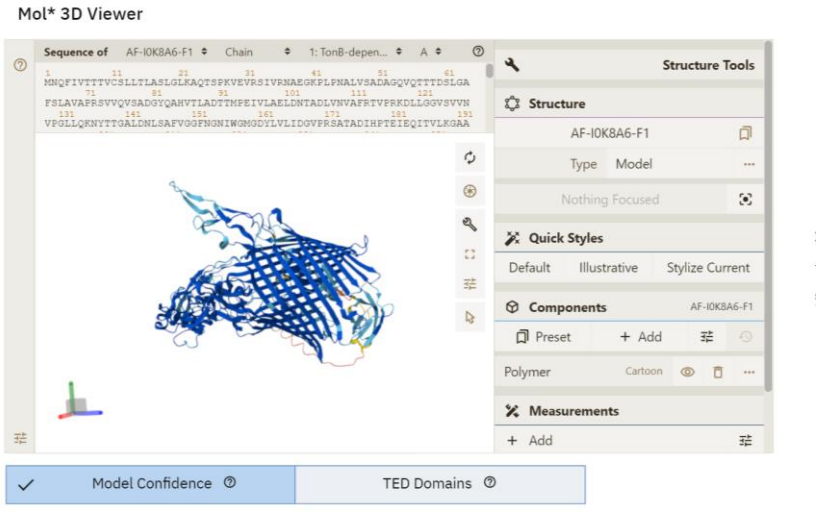
4. **Συζήτηση**

Από τις 525 ομάδες που αναδείχθηκαν με τα παραπάνω φίλτρα, ορισμένες παρουσιάζουν ιδιαίτερο ενδιαφέρον, καθώς είναι πιθανό να αποτελούν νέες οικογένειες β-βαρελιών. Ιδιαίτερη έμφαση δίνεται στα clusters που περιλαμβάνουν μεγάλο αριθμό συνολικών αλληλουχιών, εκ των οποίων σημαντικό ποσοστό έχει χαρακτηριστεί ως β-βαρέλια. Αυτό διότι σε πολυπληθή clusters αυξάνεται η πιθανότητα παρουσίας συντηρημένων μοτίβων. Αυτά τα clusters αποτελούν τους καλύτερους υποψηφίους για περαιτέρω διερεύνηση.

Σε περιπτώσεις όπου παρατηρούνται έστω και ασθενείς ενδείξεις παρουσίας γνωστών OMP domains μέσω των HMMER αποτελεσμάτων (με E-value < 0.1), ενισχύεται η πιθανότητα ότι οι ομάδες αυτές μοιάζουν με ήδη χαρακτηρισμένες οικογένειες εξωτερικών μεμβρανικών πρωτεϊνών.

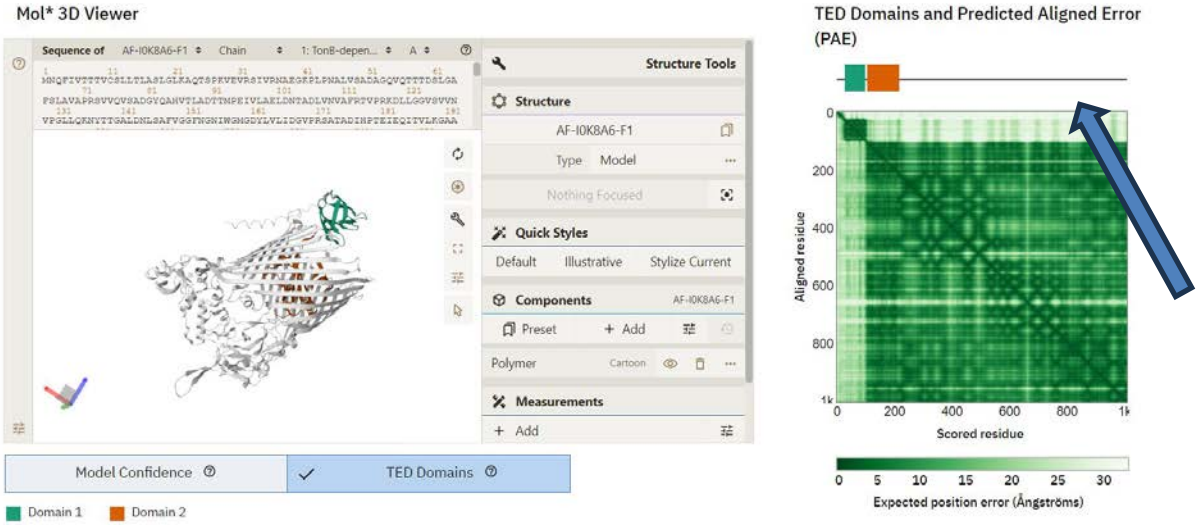
Επιπλέον, όταν διαπιστώνεται μέσω AlphaFold η παρουσία δομής που σχηματίζει β-βαρέλι ή γενικότερα δομή βαρελιού, η πιθανότητα ότι το cluster αντιπροσωπεύει μια πραγματική νέα OMP οικογένεια αυξάνεται σημαντικά. Έτσι, με τον συνδυασμό των κριτηρίων: ποσοστού β-βαρελιών, παρουσίας ασθενών OMP χαρακτηριστικών και δομικής προβλεψιμότητας, προσδιορίζονται οι πιο υποσχόμενες ομάδες για περαιτέρω μελέτη.

Για παράδειγμα, το Cluster 1182, το οποίο περιλαμβάνει 59 συνολικές αλληλουχίες, εκ των οποίων οι 43 έχουν χαρακτηριστεί ως β-βαρέλια βάσει του PRED-TMBB2. Επιπλέον, όλες οι αλληλουχίες που εντοπίσαμε ότι είναι και β-βαρέλια του συγκεκριμένου cluster διαθέτουν προβλεπόμενη βαρελοειδή δομή μέσω AlphaFold, γεγονός που ενισχύει την υπόθεση ύπαρξης διατηρημένης δευτεροταγούς και τριτοταγούς δομής σε επίπεδο οικογένειας.

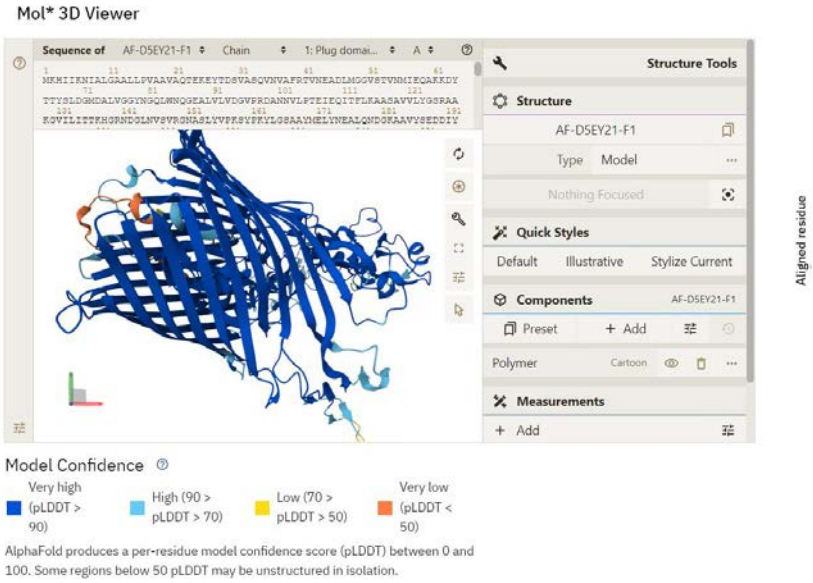


Εικόνα 4.1 Προβλεπόμενη βαρελοειδή δομή μίας αλληλουχίας (πρωτεΐνη I0K8A6) του cluster 1182 μέσω AlphaFold.

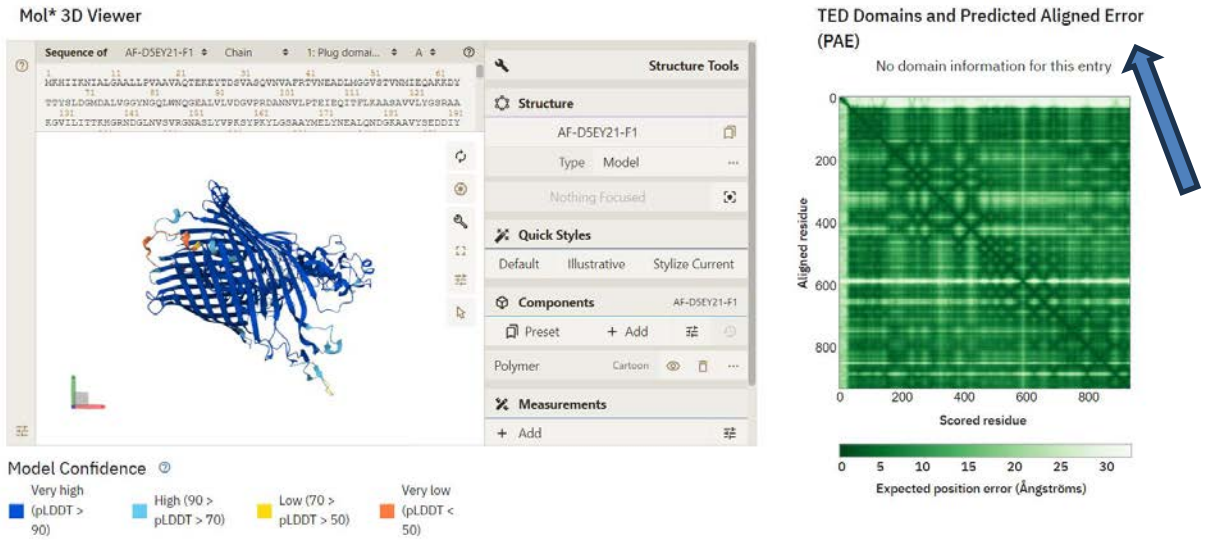
Παρά το γεγονός ότι η προβλεπόμενη τρισδιάστατη δομή μέσω AlphaFold καταδεικνύει με σαφήνεια τη μορφολογία β-βαρελιού για όλες τις αλληλουχίες του cluster, παρατηρούμε ότι το TED δεν αναγνωρίζει κάποιο διακριτό domain σε αυτή την περιοχή. Αυτό μπορεί να οφείλεται είτε στο ότι η περιοχή δεν πληροί τα κριτήρια αυτονομίας που θέτει το TED, είτε στο ότι πρόκειται για μη χαρακτηρισμένη πτύχωση, η οποία δεν έχει ακόμη ενσωματωθεί στη βάση δεδομένων.



Εικόνα 4.2 Παρουσίαση των domains που ανιχνεύθηκαν από το TED στη προβλεπόμενη πρωτεϊνική δομή της πρωτεΐνης I0K8A6 μέσω AlphaFold. Παρατηρείται ότι, παρά τη σαφή διαμόρφωση δομής τύπου β-βαρελιού στην απεικόνιση, το TED δεν την ταξινομεί ως ανεξάρτητο domain.



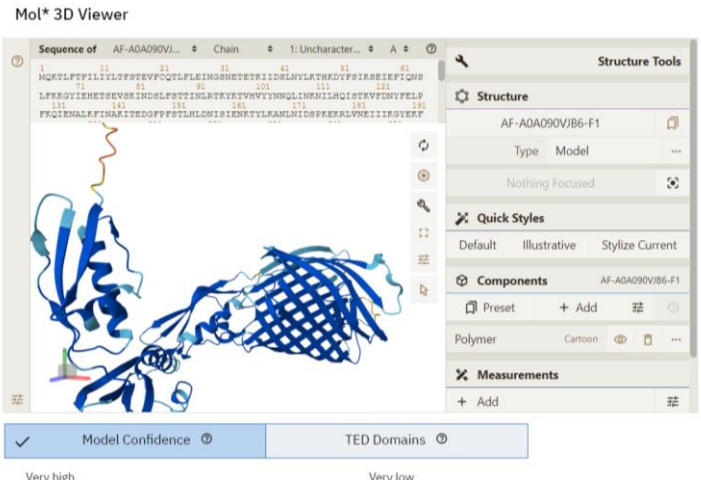
Εικόνα 4.3 Προβλεπόμενη βαρελοειδή δομή μίας αλληλουχίας (πρωτεΐνη D5EY21) του cluster 1182 μέσω AlphaFold.



Εικόνα 4.4 Το TED στη προβλεπόμενη πρωτεϊνική δομή της πρωτεΐνης D5EY21 μέσω AlphaFold, δεν ανιχνεύει κανένα doman. Παρατηρείται ότι, παρά τη σαφή διαμόρφωση δομής τύπου β-βαρελιού στην απεικόνιση, το TED δεν την ταξινομεί ως ανεξάρτητο domain.

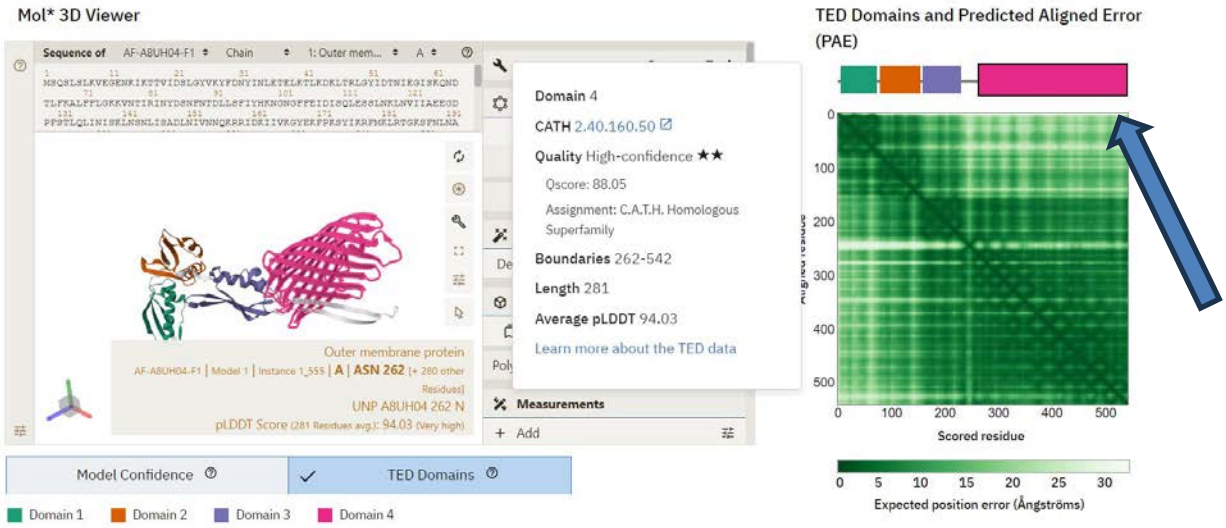
Το υψηλό ποσοστό (73%) β-βαρελιών, σε συνδυασμό με την πρόβλεψη δομών βαρελιών από AlphaFold, συνιστά ισχυρή ένδειξη ότι το Cluster 1182 θα μπορούσε να είναι μια έως τώρα μη χαρακτηρισμένη οικογένεια εξωτερικών μεμβρανικών πρωτεϊνών τύπου β-βαρελιού.

Ένα ακόμη αξιόλογο παράδειγμα αποτελεί το Cluster 10490, το οποίο περιλαμβάνει 45 συνολικές αλληλουχίες, από τις οποίες οι 32 έχουν χαρακτηριστεί ως β-βαρέλια, δηλαδή ποσοστό 71,1. Και σε αυτή τη περίπτωση οι δομές των αλληλουχιών του Cluster 10490, όταν αναζητηθούν στο AlphaFold ,δίνουν βαρελοειδείς δομές.
Το cluster αυτό παρουσιάζει επίσης 25 περιπτώσεις αλληλουχιών με εντοπισμένα OMPdb domains, τα οποία όμως εμφανίζονται με ασθενή E-values (< 0.1), σύμφωνα με το εργαλείο hmmscan. Αυτό υποδηλώνει πιθανή ομοιότητα του cluster με ήδη γνωστές οικογένειες β-βαρελιών στο OMPdb, παρά την απουσία ισχυρών χαρακτηριστικών ταύτισης.

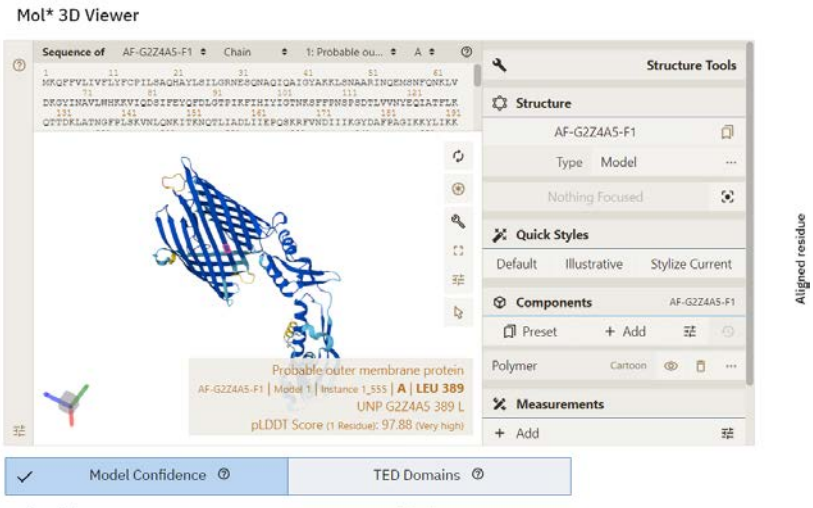


Εικόνα 4.5 Προβλεπόμενη βαρελοειδή δομή μίας αλληλουχίας (πρωτεΐνη A8UH04) του cluster 10490 μέσω AlphaFold.

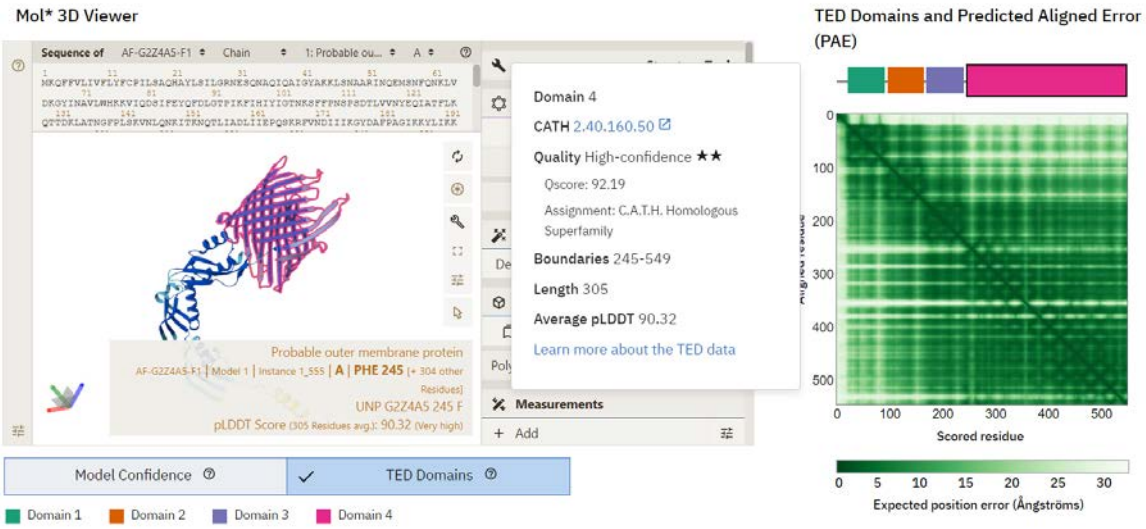
Στη συγκεκριμένη περίπτωση, το TED αναγνωρίζει τη βαρελοειδή περιοχή ως διακριτό domain, ενισχύοντας έτσι την υπόθεση ότι πρόκειται για συντηρημένο δομικό και λειτουργικό μοτίβο που παρουσιάζει ομοιότητες με κάποια οικογένεια β-βαρελιών.



Εικόνα 4.6 Παρουσίαση των domains που ανιχνεύθηκαν από το TED στη προβλεπόμενη πρωτεϊνική δομή της πρωτεΐνης A8UH04 μέσω AlphaFold. Το TED σε αυτή την περίπτωση έχει αναγνωρίσει το προβλεπόμενο βαρέλι ως ανεξάρτητο domain.



Εικόνα 4.7 Πιθανή βαρελοειδή δομή μίας αλληλουχίας(πρωτεΐνη G2Z4A5) του cluster 10490 μέσω AlphaFold.

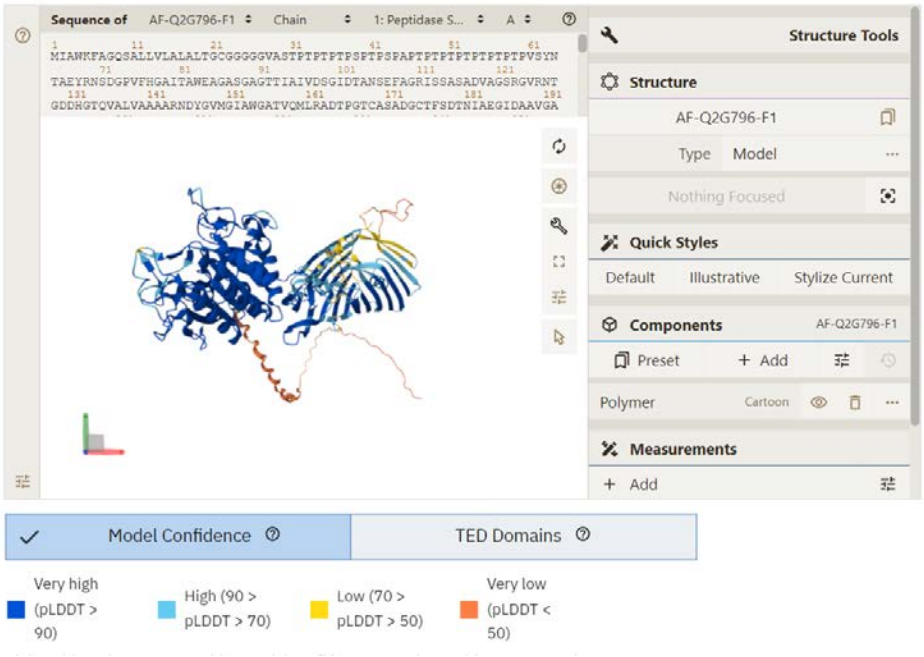


Εικόνα 4.8 Παρουσίαση των domains που ανιχνεύθηκαν από το TED στη προβλεπόμενη πρωτεϊνική δομή της πρωτεΐνης G2Z4A5 μέσω AlphaFold. Το TED σε αυτή την περίπτωση έχει αναγνωρίσει το προβλεπόμενο βαρέλι ως ανεξάρτητο domain.

Η ταυτόχρονη παρουσία μεγάλου αριθμού β-βαρελιών και η ένδειξη συσχέτισης (έστω και ασθενής) με χαρακτηρισμένες οικογένειες καθιστούν το Cluster 10490 έναν ακόμα ισχυρό υποψήφιο για νέα οικογένεια εξωτερικών μεμβρανικών πρωτεϊνών.

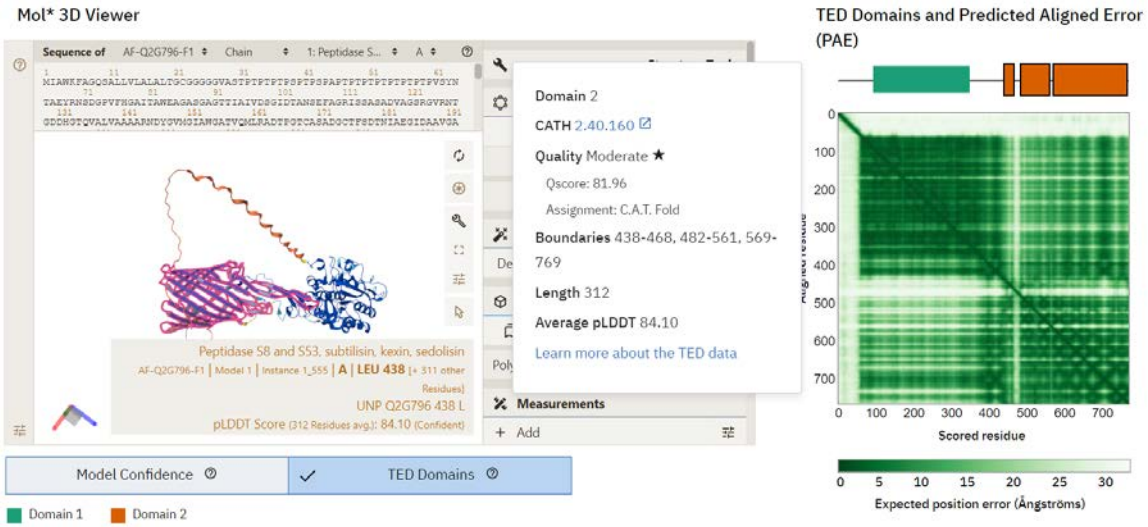
Τέλος, το Cluster 6253 περιλαμβάνει 63 συνολικές αλληλουχίες, εκ των οποίων οι 56 έχουν χαρακτηριστεί ως β-βαρέλια, ποσοστό που φτάνει το 88,9% που είναι πολύ υψηλό.Παρότι στο συγκεκριμένο cluster δεν ανιχνεύτηκε ομοιότητα με ήδη γνωστά OMPdb domains μέσω του εργαλείου hmmscan, η υψηλή του περιεκτικότητα σε β-βαρέλια σε συνδυασμό με τις προβλέψεις του AlphaFold, οι οποίες δίνουν ως αποτέλεσμα δομές τύπου βαρελιού, καθιστούν το cluster ιδιαίτερα υποσχόμενο για περαιτέρω μελέτη.

Η απουσία ομοιοτήτων ενδεχομένως υποδηλώνει ότι το Cluster 6253 μπορεί να αντιπροσωπεύει μια εντελώς νέα οικογένεια β-βαρελιών, η οποία δεν έχει ακόμη καταγραφεί σε υπάρχουσες βάσεις δεδομένων όπως το OMPdb.

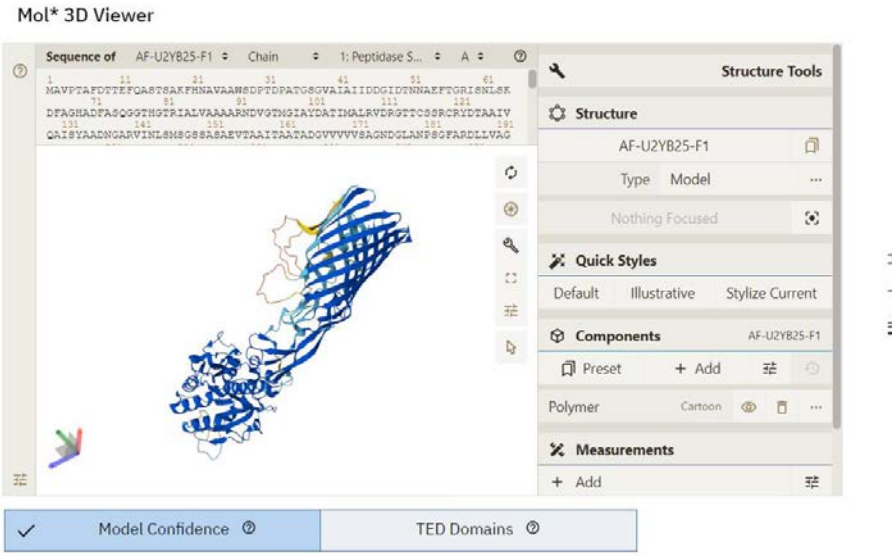


Εικόνα 4.9 Προβλεπόμενη βαρελοειδή δομή μίας αλληλουχίας (πρωτεΐνη Q2G796) του cluster 6253 μέσω AlphaFold.

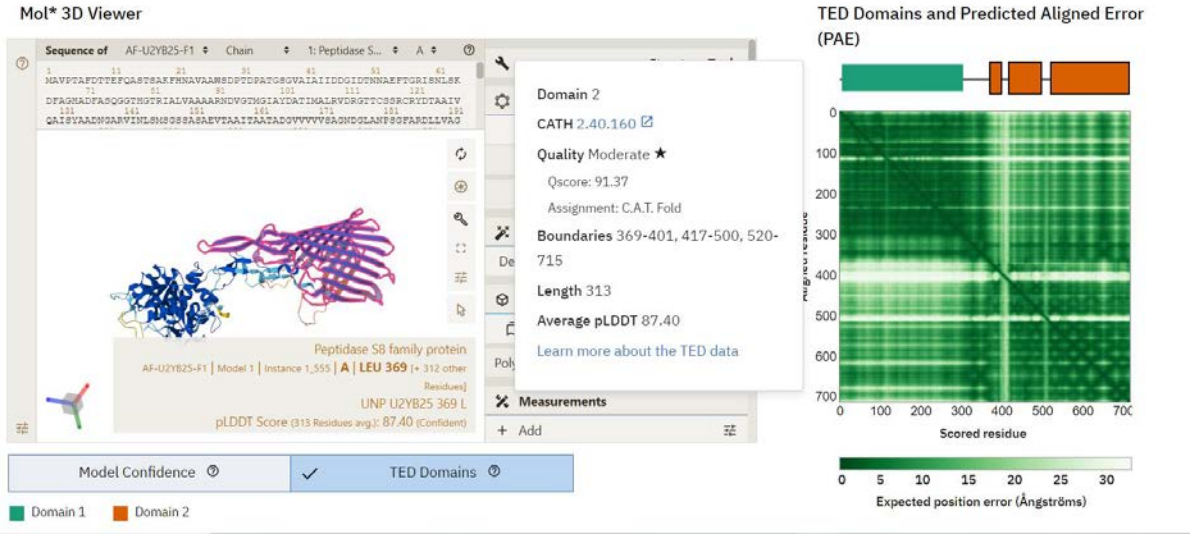
Και σε αυτή τη περίπτωση, το TED αναγνωρίζει τη βαρελοειδή περιοχή ως διακριτό domain, ενισχύοντας έτσι την υπόθεση ότι πρόκειται για συντηρημένο δομικό και λειτουργικό μοτίβο που παρουσιάζει ομοιότητες με κάποια οικογένεια β-βαρελιών.



Εικόνα 4.10 Παρουσίαση των domains που ανιχνεύθηκαν από το TED στη προβλεπόμενη πρωτεϊνική δομή της πρωτεΐνης Q2G796 μέσω AlphaFold. Το TED σε αυτή την περίπτωση έχει αναγνωρίσει το προβλεπόμενο βαρέλι ως ανεξάρτητο domain.



Εικόνα 4.11 Προβλεπόμενη βαρελοειδή δομή μίας αλληλουχίας (πρωτεΐνη U2YB25) του cluster 6253 μέσω AlphaFold.



Εικόνα 4.12 Παρουσίαση των domains που ανιχνεύθηκαν από το TED στη προβλεπόμενη πρωτεϊνική δομή της πρωτεΐνης U2YB25 μέσω AlphaFold. Το TED σε αυτή την περίπτωση έχει αναγνωρίσει το προβλεπόμενο βαρέλι ως ανεξάρτητο domain.

Η μεθοδολογία που εφαρμόστηκε, η οποία βασίστηκε στον συνδυασμό δεδομένων από δύο διαφορετικές βάσεις (OMPdb και Pfam) καθώς και στη χρήση εργαλείων πρόβλεψης δομής όπως το AlphaFold, PRED-TMBB2 αποδείχθηκε αποτελεσματική στην ταυτοποίηση νέων υποψήφιων οικογενειών β-βαρελοειδών πρωτεϊνών. Εντοπίστηκαν clusters που είτε παρουσιάζουν κάποια ομοιότητα με ήδη καταγεγραμμένες β-βαρελοειδείς δομές της Ompdb, όπως για παράδειγμα το Cluster 1182, είτε δεν παρουσιάζουν καμία ομοιότητα, όπως το Cluster 6253, το οποίο ωστόσο χαρακτηρίζεται από πολύ υψηλό ποσοστό προβλεπόμενων β-βαρελιών. Το γεγονός αυτό υποδεικνύει ότι η προσέγγιση μπορεί να αξιοποιηθεί

επιτυχώς για την ανακάλυψη νέων οικογενειών β-μεμβρανικών πρωτεϊνών. Ένα μειονέκτημα της μεθόδου είναι ότι δεν ανιχνεύει πρωτεΐνες που αποτελούνται από ένα μόνο βαρελοειδές domain (single-domain β-barrels). Η ίδια μεθοδολογία θα μπορούσε να επεκταθεί και σε άλλες κατηγορίες βιολογικών μορίων, όπως για παράδειγμα τα καρβοξυτελικά πεπτίδια βακτηρίων, με στόχο την ανακάλυψη νέων πρωτεϊνικών οικογενειών από καρβοξυτελικά (C-terminal) πεπτίδια.

Βιβλιογραφικές Αναφορές

1. Encyclopaedia Britannica. (2025, May 24). Bacteria: Cell, evolution, & classification. Encyclopaedia Britannica. https://www.britannica.com/science/bacteria

2. Encyclopaedia Britannica. (2025, May 24). Bacteria - Diversity of structure of bacteria. Encyclopaedia Britannica. https://www.britannica.com/science/bacteria/Diversity-of-structure-of-bacteria

3. Encyclopædia Britannica, Inc. (n.d.). Bacillus-type bacterial cell [Image]. Encyclopædia Britannica. Retrieved May 24, 2025, from https://www.britannica.com/science/bacteria#/media/1/48203/

4. Encyclopædia Britannica. (n.d.). Gram stain: Introduction. Retrieved May 12, 2025, from https://www.britannica.com/science/Gram-stain

5. Encyclopaedia Britannica. (2025, April 13). Gram-negative bacterium. In Britannica.com. https://www.britannica.com/science/Gram-negative-bacterium

6. Rakosy, A. W. (n.d.). Klebsiella pneumoniae in pneumonia [Image]. Encyclopædia Britannica, Inc. Retrieved May 24, 2025, from https://www.britannica.com/science/bacteria#/media/1/48203/100173

7. Μπάγκος, Π. (2005). Prediction of structure and function of membrane proteins [Διδακτορική διατριβή, Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών]. Εθνικό Αρχείο Διδακτορικών Διατριβών. http://hdl.handle.net/10442/hedi/23252

8. Λιτού, Ζ. (2009). Prediction of structure and classification of membrane proteins [Διδακτορική διατριβή, Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών]. Εθνικό Αρχείο Διδακτορικών Διατριβών. https://doi.org/10.12681/eadd/21488

9. Tamm, L. K., Arora, A., & Kleinschmidt, J. H. (2001). Structure and assembly of β-barrel membrane proteins. Journal of Biological Chemistry, 276(35), 32399–32402. https://doi.org/10.1074/jbc.R100021200

10. Walther, D. M., Rapaport, D., & Tommassen, J. (2009). Biogenesis of β-barrel membrane proteins in bacteria and eukaryotes: Evolutionary conservation and divergence. Cellular and Molecular Life Sciences, 66(17), 2789–2804. https://doi.org/10.1007/s00018-009-0029-z.1-5.

11. Wimley, W. C. (2003). The versatile β-barrel membrane protein. Current Opinion in Structural Biology, 13(4), 404–411. https://doi.org/10.1016/S0959-440X(03)00099-X

12. Μπάγκος, Π. Γ. (2015). Βιοπληροφορική (1η έκδοση). Ελληνικά Ακαδημαϊκά Ηλεκτρονικά Συγγράμματα και Βοηθήματα, σ. 161–167,300-305.

13. Ταμπόσης, Ι. (2021). Hidden Markov models (HMMs) in bioinformatics [Διπλωματική εργασία, Πανεπιστήμιο Θεσσαλίας]. Εθνικό Αρχείο Διδακτορικών Διατριβών σ.28-31. http://hdl.handle.net/10442/hedi/49086

14. Mistry, J., Chuguransky, S., Williams, L., Qureshi, M., Salazar, G. A., Sonnhammer, E. L. L., Tosatto, S. C. E., Paladin, L., Raj, S., Richardson, L. J., Finn, R. D., & Bateman, A. (2021). Pfam: The protein families database in 2021. Nucleic Acids Research, 49(D1), D412–D419. https://doi.org/10.1093/nar/gkaa913

15. Tsirigos, K. D., Bagos, P. G., & Hamodrakas, S. J. (2011). OMPdb: a database of β-barrel outer membrane proteins from Gram-negative bacteria. Nucleic Acids Research, 39(Database issue), D325–D331. https://doi.org/10.1093/nar/gkq1019

16. Eddy, S. R. (2011). Accelerated profile HMM searches. PLoS Computational Biology, 7(10), e1002195. https://doi.org/10.1371/journal.pcbi.1002195

17. Tsirigos, K. D., Elofsson, A., & Bagos, P. G. (2016). PRED-TMBB2: Improved topology prediction and detection of beta-barrel outer membrane proteins. Bioinformatics, 32(17), i665–i671. https://doi.org/10.1093/bioinformatics/btw444

18. Li, W., Jaroszewski, L., & Godzik, A. (2001). Clustering of highly homologous sequences to reduce the size of large protein databases. Bioinformatics, 17(3), 282–283. https://doi.org/10.1093/bioinformatics/17.3.282

19. Li, W., Jaroszewski, L., & Godzik, A. (2002). Tolerating some redundancy significantly speeds up clustering of large protein databases. Bioinformatics, 18(1), 77–82. https://doi.org/10.1093/bioinformatics/18.1.77

20. Li, W., & Godzik, A. (2006). Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. Bioinformatics, 22(13), 1658–1659. https://doi.org/10.1093/bioinformatics/btl158

21. Sievers, F., Wilm, A., Dineen, D. G., Gibson, T. J., Karplus, K., Li, W., Lopez, R., McWilliam, H., Remmert, M., Söding, J., Thompson, J. D., & Higgins, D. G. (2011). Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega. Molecular Systems Biology, 7(1), 539. https://doi.org/10.1038/msb.2011.75

22. Sievers, F., & Higgins, D. G. (2018). Clustal Omega for making accurate alignments of many protein sequences. Protein Science, 27(1), 135–145. https://doi.org/10.1002/pro.3290

23. Sievers, F., Barton, G. J., & Higgins, D. G. (2020). Multiple Sequence Alignment. In A. D. Baxevanis, G. D. Bader, & D. S. Wishart (Eds.), Bioinformatics (pp. 227–250). Wiley.

24. Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., ... & Hassabis, D. (2021). Highly accurate protein structure prediction with AlphaFold. *Nature*, *596*(7873), 583–589. https://doi.org/10.1038/s41586-021-03819-2

25. Varadi, M., Anyango, S., Deshpande, M., Nair, S., Natassia, C., Yordanova, G., ... & Velankar, S. (2024). AlphaFold Protein Structure Database in 2024: providing structure coverage for over 214 million protein sequences. *Nucleic Acids Research*, *52*(D1), D745–D755. https://doi.org/10.1093/nar/gkad1061

26. Lau, A. M., Bordin, N., Kandathil, S. M., Sillitoe, I., Waman, V. P., Wells, J., Orengo, C. A., & Jones, D. T. (2024). Exploring structural diversity across the protein universe with The Encyclopedia of Domains. Science, 386(6721). https://doi.org/10.1126/science.adq4

Παράρτημα

Πρόγραμμα για την αφαίρεση όλων των OMP DOMAINS **από τη βάση** PFAM.hmm

```
import pandas as pd

# extract all ompdb domains from the PFAM HMM file.

def find_pfdomains(file):
#it reads the OMPdb HMM file and extracts all Pfam accessions lines( lines that start with 'ACC   PF') and
#return a list of all Pfam

    domains = []  #list to store Pfam domains accesions

    # open the file and read all lines
    with open(file, 'r') as f:
        lines = f.readlines()

    # for  each line
    for line in lines:
        if line.startswith("ACC   PF"):  # check if line contains a Pfam accession
            domains.append(line.strip())  #store the line

    return domains  # Return a list of Pfam accessions


# remove all  domains that were found in ompdb list from the main Pfam-A file and create a
#filtered Pfam-A file
def filter_pfam(pfam_file, output, domains):

    # read all lines from the Pfam-A.hmm file
    with open(pfam_file, 'r') as f:
        lines = f.readlines()

    filtered_lines = []
    delete = False

    for line in lines:

        # each HMM entry starts with this line — reset the delete flag
        if line.startswith("HMMER3/f"):
            delete = False
            filtered_lines.append(line)
            continue
```

```
        # if the line matches any domain in the OMPdb domain list, we skip the entire entry
        if any(domain in line for domain in domains):
            delete = True

            # remove the two previous lines because they are part of the same entry
            filtered_lines.pop()
            filtered_lines.pop()
            continue

        if not delete:
            filtered_lines.append(line)

    with open(output, 'w') as f:
        f.writelines(filtered_lines)


# extract all Pfam domains from the OMPdb file
ompdb_pf_domains = find_pfdomains('OMPdb_2022_11_17.hmm')

# filter the Pfam A HMM file and remove all domains found in the OMPdb list
filter_pfam('Pfam-A.hmm', 'filtered_PFAM.hmm', ompdb_pf_domains)
```

Πρόγραμμα για τη δημιουργία πίνακα με τα εντοπισμένα από την hmmscan, Pfam **και** Ompdb domains (**Χωρίσαμε τα γονιδιώματα από τα** gram negative bacteria **σε 4 μέρη. Το πρόγραμμα αυτό το τρέξαμε ξεχωριστά για κάθε μέρος.**

```
import subprocess
import pandas as pd
import os
import tempfile
from scipy.stats.contingency import odds_ratio
from scipy.stats import fisher_exact, chi2_contingency
from math import log, sqrt
from scipy.stats import norm


#  create a temp file
def create_temp_fasta(df):

    temp_fasta = tempfile.NamedTemporaryFile(delete=False, suffix='.fasta')

    with open(temp_fasta.name, 'w') as f:
```

```
        for _, row in df.iterrows():


            if row['ID'] and row['Sequence']:
                f.write(f">{row['ID']}\n{row['Sequence']}\n")

    return temp_fasta



# create a dataframe with ID of each protein, sequence, and sequence length

def parse_fasta(filename):

    with open(filename, 'r') as file:

        entries = []
        sequence = ''
        uniprot_id = ''

        for line in file:

            line = line.strip()
            if line.startswith('>'):

                if uniprot_id:
                    entries.append([uniprot_id, sequence, len(sequence)])
                parts = line.split('|')
                if len(parts) > 1:
                    uniprot_id = parts[1]
                else:
                    uniprot_id = line[1:]
                sequence = ''
            else:
                sequence += line

        if uniprot_id:
            entries.append([uniprot_id, sequence, len(sequence)])
    return pd.DataFrame(entries, columns=['ID', 'Sequence', 'Length'])



# run HMMERscan on a  FASTA file which contains plenty amount of fasta
```

```
def run_hmmer(fasta_file, db_path, threads=50):

    result_file = f"hmmer_results_{os.path.basename(db_path)}.txt"

    try:
        result = subprocess.run(
            ["hmmscan", "--domtblout", result_file, "--cpu", str(threads), db_path, fasta_file],
            capture_output=True, text=True
        )
        if result.returncode != 0:
            print(f"HMMER execution failed for {db_path}. Moving to the next sequence.")
            print("STDOUT:", result.stdout)
            print("STDERR:", result.stderr)
            return None
        return result_file

    except Exception as e:
        print(f"Error running HMMER for {db_path}: {e}")
        return None


#  extract the  domain names and their TC values into a dictionary

def extract_tc_from_hmm(hmm_file):

    domain_tc_dict = {}

    with open(hmm_file, 'r') as hmm_f:

        current_domain = None

        for line in hmm_f:

            if line.startswith("NAME"):

                current_domain = line.split()[1]

            elif line.startswith("TC") and current_domain:

                    tc_value = float(line.split()[1])  # extract the TC value
                    domain_tc_dict[current_domain] = tc_value
                    current_domain = None  # Reset to none


    return domain_tc_dict
```

```
#  parse HMMER results for Pfam
def parse_hmmer_results(result_file, tc_pfam):

    proteins = []

    with open(result_file, 'r') as f:

        for line in f:
            if line.startswith('#') or not line.strip():
                continue

            parts = line.split()

            if len(parts) >= 19:  # if enough fields in the line

                try:

                    score = float(parts[7])  # extract the score  in place 7
                    domain_name = parts[0]  # extract the domain name

                    # Check if the domain exists in the TC dictionary
                    if domain_name in tc_pfam:
                        tc_value = tc_pfam[domain_name]  # get the TC value for the  domain

                        # append  only if its score > its TC value
                        if score > tc_value:

                            protein_id = parts[3]
                            domain_start = int(parts[17])
                            domain_end = int(parts[18])
                            proteins.append((protein_id, domain_name, domain_start, domain_end, score))
                except ValueError:
                    print(f"Warning: Error parsing line: {line}")

    return proteins



    # function to parse HMMER results for OMPdb
def parse_hmmer_results_ompdb(result_file, tc_ompdb):
    proteins = []
```

```
    with open(result_file, 'r') as f:
        for line in f:
            if line.startswith('#') or not line.strip():
                continue
            parts = line.split()

            if len(parts) >= 22:  # if there are enough parts in the line
                try:
                    score = float(parts[7])  # extract the score
                    domain_name = parts[0]  # extract the domain name

                    # if the domain exists in the TC dictionary
                    if domain_name in tc_ompdb:
                        tc_value = tc_ompdb[domain_name]  # Get the TC

                        # append the domain only if its score >its TC
                        if score > tc_value:
                            protein_id = parts[3]
                            domain_start = int(parts[17])
                            domain_end = int(parts[18])
                            proteins.append((protein_id, domain_name, domain_start, domain_end, score))
                except ValueError:
                    print(f"Warning: Error parsing line: {line}")

    return proteins



# function to add the found  domains to the DataFrame for pfam
def add_domains_to_df(df, result_file):

    df.loc[:, 'Domain_Names'] = ''
    df.loc[:, 'Domain_Ranges'] = ''
    df.loc[:, 'E_Values'] = ''

    domains = parse_hmmer_results(result_file, tc_pfam)

    if domains:

        for index, row in df.iterrows():
            domain_names = []
            domain_ranges = []
            e_values = []
```

```
            for d in domains:

                if d[0] == row['ID']:

                    domain_names.append(d[1])
                    domain_ranges.append(f"{d[2]}-{d[3]}")
                    e_values.append(str(d[4]))


            df.at[index, 'Domain_Names'] = ', '.join(domain_names)
            df.at[index, 'Domain_Ranges'] = ', '.join(domain_ranges)
            df.at[index, 'E_Values'] = ', '.join(e_values)


    # function to add beta-barrel domains to the DataFrame for OMPdb
    def add_beta_barrel_to_df(df, result_file):
        df.loc[:, 'Domain_Names'] = ''
        df.loc[:, 'Domain_Ranges'] = ''
        df.loc[:, 'E_Values'] = ''

        domains = parse_hmmer_results_ompdb(result_file, tc_ompdb)

        for index, row in df.iterrows():
            domain_names = []
            domain_ranges = []
            e_values = []

            for d in domains:
                if d[0] == row['ID']:
                    domain_names.append(d[1])
                    domain_ranges.append(f"{d[2]}-{d[3]}")
                    e_values.append(str(d[4]))

            # updates here to execute once per row
            if domain_names:
                df.at[index, 'Domain_Names'] = ', '.join(domain_names)
                df.at[index, 'Domain_Ranges'] = ', '.join(domain_ranges)
                df.at[index, 'E_Values'] = ', '.join(e_values)


    #function to find gap regions larger than 80 amino acids between domains to pfam
    def find_gap_regions(df):

        df.loc[:, 'Gap_Regions'] = ''
        df.loc[:, 'Gap_Sequences'] = ''
```

```
    for index, row in df.iterrows():
        if pd.notna(row['Domain_Ranges']) and row['Domain_Ranges'].strip():
            try:
                domain_ranges = [(int(x.split('-')[0]), int(x.split('-')[1])) for x in row['Domain_Ranges'].split(', ')
if '-' in x]

                domain_ranges.sort(key=lambda x: x[0])
                gaps = []
                gap_sequences = []
                last_end = 0

                for start, end in domain_ranges:

                    if start - last_end > 80:

                        gap_seq = row['Sequence'][last_end:start-1]
                        gaps.append(f"{last_end+1}-{start-1}")
                        gap_sequences.append(gap_seq)
                    last_end = end

                sequence_length = row['Length']

                if sequence_length - last_end > 80:

                    gap_seq = row['Sequence'][last_end:sequence_length]
                    gaps.append(f"{last_end+1}-{sequence_length}")
                    gap_sequences.append(gap_seq)


                df.at[index, 'Gap_Regions'] = ', '.join(gaps)
                df.at[index, 'Gap_Sequences'] = ', '.join(gap_sequences)



            except Exception as e:
                print(f"Error processing row {index} with Domain_Ranges={row['Domain_Ranges']}: {e}")



#  function filters proteins to keep only those with gap regions larger than 80 amin
def filter_proteins_with_large_gaps(pfam_df, min_gap_size=80):


    filtered_rows = []
```

```
    for index, row in pfam_df.iterrows():
        gap_str = row['Gap_Regions']


        if gap_str:

            for gap in gap_str.split(', '):
                try:
                    start, end = map(int, gap.split('-'))
                    if end - start + 1 > min_gap_size:
                        filtered_rows.append(row)
                        break
                except ValueError:
                    continue

    # create a DataFrame from the filtered rows and return
    return pd.DataFrame(filtered_rows)




#  process all FASTA files in a directory in batches
def process_fasta_directory(directory_path, pfam_db_path, ompdb_db_path, batch_size=100):

    filtered_pfam_df = pd.DataFrame()
    all_ompdb_df = pd.DataFrame()
    genome_counter = 0

    for filename in os.listdir(directory_path):

        if filename.endswith(".fasta"):

            genome_counter += 1  # counter

            print(f"Starting processing genome {genome_counter}")

            file_path = os.path.join(directory_path, filename)
            df = parse_fasta(file_path)

            # Process in batches of size 100
            # dataframe/100 and if %>0 then 1 more
            num_batches = len(df) // batch_size + (1 if len(df) % batch_size != 0 else 0)
```

```
        for batch_num in range(num_batches):

            start = batch_num * batch_size
            end = min((batch_num + 1) * batch_size, len(df))
            batch_df = df.iloc[start:end]

            # Create temporary FASTA file
            temp_fasta = create_temp_fasta(batch_df)

            # Process with Pfam
            pfam_result_file = run_hmmer(temp_fasta.name, pfam_db_path)

            if pfam_result_file:

                add_domains_to_df(batch_df, pfam_result_file)
                find_gap_regions(batch_df)  # Process gap regions
                # Concatenate the batch directly into the filtered_pfam_df
                filtered_pfam_df = pd.concat([filtered_pfam_df, batch_df], ignore_index=True)

            # Process separately with OMPdb
            ompdb_result_file = run_hmmer(temp_fasta.name, ompdb_db_path)

            if ompdb_result_file:
                add_beta_barrel_to_df(batch_df, ompdb_result_file)


                batch_df = batch_df.drop(columns=['Gap_Regions', 'Gap_Sequences'], errors='ignore')

                all_ompdb_df = pd.concat([all_ompdb_df, batch_df], ignore_index=True)

            os.remove(temp_fasta.name)

    return filtered_pfam_df, all_ompdb_df


#  calculate the frequencies of  all domains  seperately in filtered_pfam
def calculate_domain_frequencies(pfam_df):

    domain_frequencies = {}

    for _, row in pfam_df.iterrows():

        domains = row['Domain_Names'].split(', ') if row['Domain_Names'] else []
```

```
        for domain in domains:
            if domain in domain_frequencies:
                domain_frequencies[domain] += 1
            else:
                domain_frequencies[domain] = 1

    domain_frequencies_list = list(domain_frequencies.items())
    frequency_df = pd.DataFrame(domain_frequencies_list, columns=['Domain', 'Total_Pfam_Count'])

    return frequency_df

def combine_pfam_ompdb(pfam_df, ompdb_df):

    combined_rows = []

    for _, pfam_row in pfam_df.iterrows():
        protein_id = pfam_row['ID']
        same_ompdb = ompdb_df[ompdb_df['ID'] == protein_id]

        combined_info = []


        pfam_domains = pfam_row['Domain_Names'].split(', ')
        pfam_ranges = pfam_row['Domain_Ranges'].split(', ')
        pfam_e_values = pfam_row['E_Values'].split(', ')

        for domain, drange, e_value in zip(pfam_domains, pfam_ranges, pfam_e_values):

            if drange.strip():

                combined_info.append((domain, drange, e_value))


        if not same_ompdb.empty:

            ompdb_domains = same_ompdb['Domain_Names'].iloc[0].split(', ')
            ompdb_ranges = same_ompdb['Domain_Ranges'].iloc[0].split(', ')
            ompdb_e_values = same_ompdb['E_Values'].iloc[0].split(', ')


            for domain, drange, e_value in zip(ompdb_domains, ompdb_ranges, ompdb_e_values):
                if drange.strip():

                    combined_info.append((domain, drange, e_value))
```

```
        combined_info.sort(key=lambda x: int(x[1].split('-')[0]))


        combined_domains = ', '.join([info[0] for info in combined_info])
        combined_ranges = ', '.join([info[1] for info in combined_info])
        combined_e_values = ', '.join([info[2] for info in combined_info])


        combined_row = pfam_row.copy()
        combined_row['Domain_Names'] = combined_domains
        combined_row['Domain_Ranges'] = combined_ranges
        combined_row['E_Values'] = combined_e_values

        combined_rows.append(combined_row)

    return pd.DataFrame(combined_rows)



def find_omp_domains(file_path):

    domain_names = []

    with open(file_path, 'r') as file:

        for line in file:
            if line.startswith('NAME'):

                domain_name = line.split()[1]
                domain_names.append(domain_name)


    omp_domains_df = pd.DataFrame(domain_names, columns=['Domain_Name'])
    # omp_domains_df.to_csv('omp_domains.csv', index=False)

    return omp_domains_df



# Calculate domain coappearance
def calculate_domain_coappearance(frequency_df, combined_df, ompdb_domains):
```

```
coappearance = []

for _, freq_row in frequency_df.iterrows():
    domain_name = freq_row['Domain']


    pfam_and_barrel = 0
    no_domain_no_barrel = 0
    only_barrel = 0
    only_domain = 0

    for _, row in combined_df.iterrows():

        domains = row['Domain_Names'].split(', ')
        ompdb_present = any(d in domains for d in ompdb_domains)

        if domain_name in domains and ompdb_present:
            pfam_and_barrel += 1

        elif domain_name not in domains and not ompdb_present:
            no_domain_no_barrel += 1

        elif ompdb_present:
            only_barrel += 1

        else:
            only_domain += 1



    #put all information in dataframe
    coappearance.append({
        'Pfam_Domain': domain_name,
        'Pfam_and_Barrel': pfam_and_barrel,
        'no_domain_no_barrel': no_domain_no_barrel,
        'Only_Barrel': only_barrel,
        'Only_Domain': only_domain

    })


coappearance_df = pd.DataFrame(coappearance)


coappearance_df.to_csv(coappearance_file, index=False)
```

```
    print(f"Coappearance results saved to {coappearance_file}")



    return coappearance_df


pfam_db_path = 'filtered_PFAM.hmm'
ompdb_db_path = 'OMPdb_2022_11_17.hmm'
combined_output_file = 'combined_output.csv'
filtered_combined_output_fileiltered_combined_output= 'f.csv'
pfam_output_file = 'pfam_output.csv'
ompdb_output_file = 'ompdb_output.csv'
matching_protein_file = 'matching_proteins.csv'
output_fasta_file = 'matching_proteins.fasta'
domain_combinations_file = 'domain_combinations.csv'
filtered_pfam_output_file = 'filtered_pfam_output.csv'
beta_barrel_output_file = 'beta_barrel_cooccurrences.csv'
output_file = 'domain_frequencies.csv'
pfam_tc_file_path = ''pfam_tc_values.csv''
ompdb_tc_file_path = ''ompdb_tc_values.csv''
coappearance_file= 'coappearance_results.csv'

# file that contains whole FASTA files of all reference genomes
directory_path = 'fasta_files'


#extract the TC values from the HMM files and put them in 2 dictionaries

#For Pfam
tc_pfam = extract_tc_from_hmm(pfam_db_path)

#For Ompdb
tc_ompdb = extract_tc_from_hmm(ompdb_db_path)



# process the FASTA files to get Pfam and OMPdb DataFrames
filtered_pfam_df, all_ompdb_df = process_fasta_directory(directory_path, pfam_db_path,
ompdb_db_path)
```

```
# save the filtered Pfam DataFrame to a CSV file
#filtered_pfam_df.to_csv(filtered_pfam_output_file, index=False)
#print(f"Filtered Pfam results saved to {filtered_pfam_output_file}")


# save the OMPdb DataFrame to a CSV file
#all_ompdb_df.to_csv(ompdb_output_file, index=False)
#print(all_ompdb_df.head())
#print(f"OMPdb results saved to {ompdb_output_file}")

# save the filtered Pfam DataFrame to a CSV file without the sequence column
filtered_pfam_df_no_seq = filtered_pfam_df.drop(columns=['Sequence'], errors='ignore')
filtered_pfam_df_no_seq.to_csv(filtered_pfam_output_file, index=False)
print(f"Filtered Pfam results saved to {filtered_pfam_output_file}")

# save the OMPdb DataFrame to a CSV file without the Sequence column
all_ompdb_df_no_seq = all_ompdb_df.drop(columns=['Sequence'], errors='ignore')
all_ompdb_df_no_seq.to_csv(ompdb_output_file, index=False)
print(all_ompdb_df_no_seq.head())
print(f"OMPdb results saved to {ompdb_output_file}")


# get all Pfam domain names and save it in a cvs. Calculate how many times each Pfam domain  appears
#in filtered_Pfam df
frequency_df = calculate_domain_frequencies(filtered_pfam_df)
#frequency_df.to_csv(output_file, index=False)
print(f"Domain frequencies saved to {output_file}")

# combine the data from Pfam and OMPdb
combined_df = combine_pfam_ompdb(filtered_pfam_df, all_ompdb_df)


#save the combined DataFrame before processing gap regions to a CSV
#cbined_before_gaps_file = "combined_before_gaps.csv"
#ombined_df.to_csv(combined_before_gaps_file, index=False)
#rint(f"Combined DataFrame before gaps  to {combined_before_gaps_file}")


# find gap regions in the combined DataFrame
find_gap_regions(combined_df)


# filter the combined DataFrame to retain proteins with large gaps
filtered_combined_df = filter_proteins_with_large_gaps(combined_df)
```

```
# drop the Sequence column from the filtered DataFrame
filtered_combined_df_no_seq = filtered_combined_df.drop(columns=['Sequence'])

# save the filtered DataFrame without the Sequence column to a CSV file
filtered_combined_df_no_seq.to_csv(filtered_combined_output_file, index=False)


print("Filtered and combined DataFrame:")
print(filtered_combined_df_no_seq.head())



combined_df_no_seq = combined_df.drop(columns=['Sequence'], errors='ignore')

combined_df_no_seq.to_csv(combined_output_file, index=False)
print(f"Combined results saved to {combined_output_file}")


# get all OMPdb domain names and save it in a cvs
Ompdp_om_domains = find_omp_domains('OMPdb_2022_11_17.hmm')
ompdb_domains = Ompdp_om_domains['Domain_Name'].tolist()  # Convert the DataFrame to a list of domain names

# calculate the coappearance of Pfam domains with OMPdb domains
coappearance_df = calculate_domain_coappearance(frequency_df, combined_df, ompdb_domains)
```

Πρόγραμμα για τη δημιουργία πίνακα 2x2 **συνύπαρξης** Pfam domain **με** Ompdb domains **για καθένα από τα 4 μέρη.**

```
import pandas as pd
from tqdm import tqdm


pfam_domains_file = 'pfam_domains_list_part4.csv'  #file with all Pfam domains that we found in the #previous code

ompdb_domains_file = 'omp_domains.csv'  #file with all OMPdb domains that we found in the previous #code

filtered_combined_output_file = 'filtered_combined_output_part4.csv'  #combined dataset
coappearance_file = 'coappearance4.csv'  #file for results

#  Pfam domains (set)
```

```
with open(pfam_domains_file, 'r') as file:
    pfam_domains_set = {line.strip() for line in file if line.strip()}

#  OMPdb domains  (set)
with open(ompdb_domains_file, 'r') as file:
    ompdb_domains_set = {line.strip() for line in file if line.strip()}

# load combined dataset  in DataFrame
combined_df = pd.read_csv(filtered_combined_output_file)

# function to find coappearance των domains
#calculates co-appearance between specific Pfam domains and OMPdb domains
 #   For each Pfam domain in the pfam_domains_set, it counts how many proteins:
#    1. Contain both the Pfam domain and an OMPdb domain
 #    2. Contain neither the Pfam domain nor an OMPdb domain
  #   3. Contain only an OMPdb domain (but not the current Pfam domain)
    #4. Contain only the Pfam domain (but not any OMPdb domain)

def calculate_domain_coappearance(pfam_domains_set, combined_df, ompdb_domains_set):

    pfam_and_barrel_count = {domain: 0 for domain in pfam_domains_set}
    no_domain_no_barrel_count = {domain: 0 for domain in pfam_domains_set}
    only_barrel_count = {domain: 0 for domain in pfam_domains_set}
    only_domain_count = {domain: 0 for domain in pfam_domains_set}



    for _, row in tqdm(combined_df.iterrows(), total=combined_df.shape[0], desc="Processing"):
        domains = set(row['Domain_Names'].split(', ')) #split the domains when you find ‘,’

        ompdb_present = any(d in ompdb_domains_set for d in domains)
        for domain_name in pfam_domains_set:
            if domain_name in domains and ompdb_present:
                pfam_and_barrel_count[domain_name] += 1
            elif domain_name not in domains and not ompdb_present:
                no_domain_no_barrel_count[domain_name] += 1
            elif ompdb_present:
                only_barrel_count[domain_name] += 1
            elif domain_name in domains:
                only_domain_count[domain_name] += 1

    # create list with results
    coappearance = []
    for domain_name in pfam_domains_set:
        coappearance.append({
            'Pfam_Domain': domain_name,
```

```
            'Pfam_and_Barrel': pfam_and_barrel_count[domain_name],
            'No_Domain_No_Barrel': no_domain_no_barrel_count[domain_name],
            'Only_Barrel': only_barrel_count[domain_name],
            'Only_Domain': only_domain_count[domain_name]
        })

    # save df in CSV
    coappearance_df = pd.DataFrame(coappearance)
    coappearance_df.to_csv(coappearance_file, index=False)
    print(f"Results on {coappearance_file}")
    return coappearance_df

coappearance_df = calculate_domain_coappearance(pfam_domains_set, combined_df, ompdb_domains_set)
```

Πρόγραμμα για Ένωση των τεσσάρων coappearance_tables **σε ένα ενιαίο**

```
import pandas as pd

# load all 4 coappearance files
coappearance_files = ['coappearance1.csv', 'coappearance2.csv', 'coappearance3.csv', 'coappearance4.csv']
final_coappearance_file = 'coappearance_final.csv' #will be the final coappearance_file

# Create an empty DataFrame that will hold the final merged result
merged_df = pd.DataFrame()

#go through each co-appearance file
for file in coappearance_files:

    # put the  current file into a DataFrame
    df = pd.read_csv(file)

    # if it's the first file, put it to merged_df
    if merged_df.empty:
        merged_df = df
    else:
        # merge current DataFrame with the existing merged one, using 'Pfam_Domain' as key
        # 'outer' ensures that no domain is lost even if it's not present in both files
        merged_df = pd.merge(merged_df, df, on='Pfam_Domain', how='outer', suffixes=('', '_new'))

        # for each relevant column, add the values from both merged and new columns
```

```
        for col in ['Pfam_and_Barrel', 'No_Domain_No_Barrel', 'Only_Barrel', 'Only_Domain']:
            if col in df.columns:
                # sum the values from original and new column versions
                merged_df[col] = merged_df[[col, f'{col}_new']].sum(axis=1, min_count=1)
                # Remove the temporary "_new" column after summing
                merged_df.drop(columns=[f'{col}_new'], inplace=True)

            else:
                # if the column didn't exist in the current file, just fill it with 0s
                merged_df[col] = merged_df[col].fillna(0)

# replace any remaining NaN values with 0
merged_df.fillna(0, inplace=True)

# convert count columns to integers (they may be floats due to NaNs or summing)
merged_df[['Pfam_and_Barrel', 'No_Domain_No_Barrel', 'Only_Barrel', 'Only_Domain']] = merged_df[
    ['Pfam_and_Barrel', 'No_Domain_No_Barrel', 'Only_Barrel', 'Only_Domain']
].astype(int)

# Save the final results to a csv file
merged_df.to_csv(final_coappearance_file, index=False)

print(f'Results saved to {final_coappearance_file}')
```

Πρόγραμμα για Εκτέλεση στατιστικών τεστ, με σκοπό την απομόνωσή των στατιστικά σημαντικών (odds_ratio>1, p-value<0.05) **συνυπάρξεων των** Pfam domains **με β-βαρέλια**.

```
import pandas as pd
import numpy as np
from scipy.stats.contingency import odds_ratio
from scipy.stats import fisher_exact, chi2_contingency
from math import log, sqrt
from scipy.stats import norm

# files
coappearance_file = 'man.csv'
significant_file = 'significant_results.csv'
pfam_and_barrel_file = 'minn.csv'

# check if all values are > 0 for a valid 2x2 contingency table
def is_valid_contingency_table(pfam_and_barrel, only_domain, only_barrel, no_domain_no_barrel):
    return all(val > 0 for val in [pfam_and_barrel, only_domain, only_barrel, no_domain_no_barrel])
```

```
# Odds Ratio calculation with 2x2 contingency table
def find_odd_ratio(pfam_and_barrel, only_domain, only_barrel, no_domain_no_barrel):
    if not is_valid_contingency_table(pfam_and_barrel, only_domain, only_barrel, no_domain_no_barrel):
        return np.nan, np.nan  # Return NaNs

    contingency_table = [[pfam_and_barrel, only_domain], [only_barrel, no_domain_no_barrel]]
    res = odds_ratio(contingency_table, kind='sample')  # Calculate odds ratio

    try:
        #its p-value using normal approximation
        log_odds_ratio = log(res.statistic)
        variance = (1 / pfam_and_barrel) + (1 / only_domain) + (1 / only_barrel) + (1 /
no_domain_no_barrel)
        p_value = 2 * (1 - norm.cdf(abs(log_odds_ratio / sqrt(variance))))
    except:
        return np.nan, np.nan  # Return NaNs

    return res.statistic, p_value

# Fisher's Exact Test 2-sided
def find_Fishers_Exact(pfam_and_barrel, only_domain, only_barrel, no_domain_no_barrel):
    return fisher_exact([[pfam_and_barrel, only_domain], [only_barrel, no_domain_no_barrel]],
alternative='two-sided')[1] \
        if is_valid_contingency_table(pfam_and_barrel, only_domain, only_barrel, no_domain_no_barrel) else
np.nan

# Pearson's Chi-Square Test with Yates' correction
def find_Pearson_Chi_Test(pfam_and_barrel, only_domain, only_barrel, no_domain_no_barrel):
    return chi2_contingency([[pfam_and_barrel, only_domain], [only_barrel, no_domain_no_barrel]],
correction=True)[1] \
        if is_valid_contingency_table(pfam_and_barrel, only_domain, only_barrel, no_domain_no_barrel) else
np.nan

# load co-appearance data
df = pd.read_csv(coappearance_file)

required_columns = ['Pfam_and_Barrel', 'Only_Domain', 'Only_Barrel', 'No_Domain_No_Barrel']
for col in required_columns:
    if col not in df.columns:
        raise ValueError(f"Column {col} was not found in file {coappearance_file}")
    df[col] = df[col].fillna(0).astype(int)  # ensure numeric integer values

# put each row to calculate stats
results = []
for _, row in df.iterrows():
```

```
    pfam_and_barrel = row['Pfam_and_Barrel']
    only_domain = row['Only_Domain']
    only_barrel = row['Only_Barrel']
    no_domain_no_barrel = row['No_Domain_No_Barrel']

    # compute all three statistical measures
    odds_ratio_val, p_value = find_odd_ratio(pfam_and_barrel, only_domain, only_barrel,
no_domain_no_barrel)
    fishers_exact = find_Fishers_Exact(pfam_and_barrel, only_domain, only_barrel, no_domain_no_barrel)
    pearson_chi_test = find_Pearson_Chi_Test(pfam_and_barrel, only_domain, only_barrel,
no_domain_no_barrel)

    # Store results
    results.append({
        'Pfam_Domain': row['Pfam_Domain'],
        'Pfam_and_Barrel': pfam_and_barrel,
        'No_Domain_No_Barrel': no_domain_no_barrel,
        'Only_Barrel': only_barrel,
        'Only_Domain': only_domain,
        'Odd_Ratio': odds_ratio_val,
        'P_value': p_value,
        'Fishers_Exact': fishers_exact,
        'Pearson_Chi_Test': pearson_chi_test
    })

# convert list of results to df
results_df = pd.DataFrame(results)

# save results to file
results_df.to_csv(coappearance_file, index=False)
print(f"Results saved to {coappearance_file}")

# save domains with Odds Ratio > 1 and p-value < 0.05
significant_df = results_df[(results_df['Odd_Ratio'] > 1) & (results_df['P_value'] < 0.05)]
significant_df.to_csv(significant_file, index=False)
print(f"Significant results (Odd_Ratio > 1 and P_value < 0.05) saved to {significant_file}")

# save domains that actually co-occurred with barrel domains at least once
pfam_and_barrel_df = results_df[results_df['Pfam_and_Barrel'] >= 1]
pfam_and_barrel_df.to_csv(pfam_and_barrel_file, index=False)
print(f"Results with Pfam_and_Barrel >= 1 saved to {pfam_and_barrel_file}")
```

Πρόγραμμα για Εύρεση των πρωτεϊνών που έχουν Pfam domain,**τουλάχιστον** 1 **κενή περιοχή >80 αμινοξέων και όχι β-βαρέλι**

```
import pandas as pd
from collections import defaultdict

# load the Pfam and OMP domain data from CSV files
pfam_df = pd.read_csv("significant_results.csv")  # a list with Pfam domains
omp_df = pd.read_csv("omp_domains.csv")  # a list with OMP domains

# Create sets  for pfam and ompdb domains for fast check
pfam_domains = set(pfam_df["Pfam_Domain"].dropna())
omp_domains = set(omp_df["Domain_Name"].dropna())

#keep only proteins that contain at least one Pfam domain that coappears with ompdb barrels, don't
#contain any OMP domain

def filter_proteins(row):


    domains = set(row["Domain_Names"].split(", "))  # get all domains in this protein
    return any(d in pfam_domains for d in domains) and not any(d in omp_domains for d in domains)

#the 4 dataframes
file_names = [
    "filtered_combined_output_part1.csv",
    "filtered_combined_output_part2.csv",
    "filtered_combined_output_part3.csv",
    "filtered_combined_output_part4.csv"
]

# list that will hold all sequences for the final fasta file
fasta_content = []

# dictionary to group sequences that have the same protein ID and gap region
grouped_fasta = defaultdict(lambda: {"domains": set(), "sequences": {}})

#processing of each one of 4 dataframes file

for file in file_names:
    print(f"Processing : {file}")

    # load the file
    filtered_df = pd.read_csv(file)

    # function to keep only interesting proteins
    #for each row  in the df, pass the row to the function filter_proteins(row) and check if it returns true or
```

```
#false
    filtered_proteins = filtered_df[filtered_df.apply(filter_proteins, axis=1)]

    # loop for each filtered protein entry
    for _, row in filtered_proteins.iterrows():
        protein_id = row["ID"]
        domain_names = row["Domain_Names"].split(", ")
        gap_regions = row["Gap_Regions"]
        gap_sequences = row["Gap_Sequences"]

        # skip proteins that don't have any gap regions or sequences
        if pd.isna(gap_regions) or pd.isna(gap_sequences):
            continue

        # split multiple gap regions and their sequences
        gaps = gap_regions.split(",")
        sequences = gap_sequences.split(", ")

        # loop over each gap and its sequence
        for gap, sequence in zip(gaps, sequences):

            for domain in domain_names:

                # if the domain is  Pfam but not OMP)
                if domain in pfam_domains:

                    # save the sequence in fasta format
                    fasta_content.append(f">{protein_id}|{domain}|{gap}\n{sequence}")

                    # group sequences by( protein ID and gap region)
                    grouped_fasta[(protein_id, gap)]["domains"].add(domain)
                    grouped_fasta[(protein_id, gap)]["sequences"][sequence] = True

# save individual domains sequences in fasta file

with open("final_filtered_results.fasta", "w") as fasta_file:
    fasta_file.write("\n".join(fasta_content))

# save grouped sequences (same protein ID and gap and MERGED domains) in another fasta(We will use
#this file

grouped_fasta_content = []
for (protein_id, gap), data in grouped_fasta.items():

    domain_list = ", ".join(sorted(data["domains"]))  # combine domain names
```

```
    sequence = list(data["sequences"].keys())[0]  # get the actual sequence
    grouped_fasta_content.append(f">{protein_id}|{domain_list}|{gap}\n{sequence}")

with open("grouped_filtered_results.fasta", "w") as grouped_fasta_file:
    grouped_fasta_file.write("\n".join(grouped_fasta_content))
```

Πρόγραμμα για Επεξεργασία αποτελεσμάτων από το εργαλείο PRED-TMBB2 **με λογιστική παλινδρόμηση, για να βρούμε ποιες πρωτεΐνες αποτελούν β-βαρέλια**

```
import re
from math import exp

# to calculate a score to define if a( protein gap) is b barrel or not

def calc_score(logodds, logprob, reliability, transmembrane, length):
    # calculate the score
    transtolng = transmembrane / length if length else 0

    # return score using a linear combination of parameters
    return (
        0.187989 * logodds
        - 9.133431 * logprob
        - 12.07592 * reliability
        + 1.270188 * transmembrane
        - 71.9362 * transtolng
        + 34.00917
    )

# files
input_file = "predtmbb_results.txt"
output_file = "predtmbb_scores.txt"

# open input file for reading and output file for writing
with open(input_file) as infile, open(output_file, "w") as outfile:
    # write the header line to the output file
    outfile.write("ID,p,beta barrel?\n")

    #for each line in the input file
   for line in infile:
        line = line.strip()

        # if the line contains the protein ID

        if line.startswith("ID:"):
```

```
            # extract the protein ID using regex (after '>')
            protein_id = re.search(r'>(.+)', line).group(1)

        # if the line contains sequence length, log-odds, and log-probability
        elif "len =" in line:
            # extract the length of the protein
            length = float(re.search(r'len = ([\d.]+)', line).group(1))

            # extract log-odds score
            logodds = float(re.search(r'logodds = ([-\d.]+)', line).group(1))

            # extract log-probability score (already divided by length)
            logprob = float(re.search(r'\(-logprob/lng\) = ([-\d.]+)', line).group(1))

        # if the line contains reliability value (VR)
        elif line.startswith("VR:"):
            # get reliability score from the line
            reliability = float(line.split()[1])

        #if the line contains LP sequence
        elif line.startswith("LP:"):

            # LP sequence is a string where 'M' indicates membrane regions
            lp_sequence = line.split()[1]

            #count the number of contiguous 'M' regions as transmembrane count
            transmembrane = len(re.findall(r'M+', lp_sequence))

            # calculate final score using the defined function
            score = calc_score(logodds, logprob, reliability, transmembrane, length)

            #convert score to probability with the sigmoid function
            p = 1 / (1 + exp(-score))

            # annotate as beta-barrel if p > 0.25
            result = "YES" if p > 0.25 else "NO"

            outfile.write(f"{protein_id},{p:.5f},{result}\n")

print("results on:", output_file)
```

Πρόγραμμα για Αντικατάσταση του ονόματος των κενών περιοχών με αριθμούς , με στόχο την καλύτερη αναγνώριση των περιοχών αυτών στα αποτελέσματα των clusters **μετά το** CD-HIT**(Το**

CD-HIT **δεν συγκρατεί όλους τους χαρακτήρες του ονόματος των κενών περιοχών π.χ.** >J3D481|OmpA| 225-1520)

```
from pathlib import Path

#replace the header name (for example >J3D481|OmpA| 225-1520) with a number. We will later replace
#again the number with the real header.
def number_fasta_headers(input_file, output_file):
    counter = 1
    output_lines = []

    with open(input_file) as f:
        for line in f:
            if line.startswith(">"):
                new_line = f">{counter} {line[1:]}"  # put the number after '>'
                counter += 1
                output_lines.append(new_line)
            else:
                output_lines.append(line)

    Path(output_file).write_text("".join(output_lines))
    print(f" {counter-1} headers")

number_fasta_headers("grouped_filtered_results.fasta", "grouped_filtered_results_numbered.fasta")
```

Πρόγραμμα για Αντικατάσταση των αριθμών από τις πραγματικές αλληλουχίες στα clusters

```
# read grouped_filtered_numbered.fasta and create a dictionary to raplace numbers with the real headers
id_to_header = {}

with open("grouped_filtered_results_numbered.fasta") as fasta_file:
    for line in fasta_file:
        if line.startswith(">"):
            parts = line[1:].split(maxsplit=1)  # remove '>' and split into number and rest of header
            if len(parts) >= 2:
                number = parts[0].strip()       # get the number header
                header = parts[1].strip()       # get the actual header
                id_to_header[number] = header   # store mapping

print(f"{len(id_to_header)} number-to-header matches.")

# read the cluster file and replace the numbers with full headers
output_lines = []

with open("clustered_40_numbered.fasta.clstr") as clstr_file:
```

```
    for line in clstr_file:
        if ">" in line and "..." in line:
            parts = line.strip().split(">")
            before = parts[0]              # keep part before '>'
            after = parts[1]               # part after '>'

            number = after.split("...")[0]   # extract the number before '...'
            header = id_to_header.get(number)

            if header:
                if "at" in after:
                    # if 'at' exists, keep it after the new header
                    after_at = after.split("at", 1)[1].strip()
                    new_line = f"{before}>{header} at {after_at}\n"
                else:
                    # just replace with the new header
                    new_line = f"{before}>{header}\n"
                output_lines.append(new_line)
            else:
                print(f"no header found for number: {number}")
                output_lines.append(line)  # keep same the line if no match found
        else:
            output_lines.append(line)      # if it doesn't match the pattern don't change it

# write updated lines to a new output file
with open("clustered_40_numbered_with_headers.clstr", "w") as f:
    f.writelines(output_lines)
print("saved as clustered_40_numbered_with_headers.clstr")
```

Πρόγραμμα για Εισαγωγή των β-βαρελίων από το αρχείο pred_scores.txt**(δηλαδή από τα αποτελέσματα του** PRED-TMBB2**) στα** Clusters

```
from pathlib import Path

def load_pred_info(pred_file):
    # load only the lines that end with YES into a dictionary (b-barrels)
    pred_info = {}
    with open(pred_file) as f:
        for line in f:
            parts = line.strip().split(maxsplit=1)  # split the line into number and the rest
            if len(parts) == 2:
                number, rest = parts
                if rest.strip().endswith("YES"):  # only keep if prediction is YES
                    pred_info[number] = rest
    return pred_info
```

```
def annotate_clusters(cluster_file, pred_info):
    # add annotation to cluster lines if there is a match in pred_info
    output_lines = []
    match_count = 0

    with open(cluster_file) as f:
        for line in f:
            new_line = line  # default is the original line
            if ">" in line and "..." in line:  # check for lines that contain sequence identifiers
                try:
                    raw = line.strip().split(">")[1]  # get the part after '>'
                    cluster_number = raw.split("...")[0]  # extract the number before '...'

                    if cluster_number in pred_info:
                        annotation = pred_info[cluster_number]
                        new_line = line.rstrip() + f" [PRED] {annotation}\n"  # add prediction info
                        match_count += 1
                        print(f"match (YES): {cluster_number} --> {annotation}")
                except Exception as e:
                    print(f"error in line: {line.strip()} - {e}")
            output_lines.append(new_line)
    return output_lines, match_count

def main():
    pred_file = "predtmbb_scores_numbered.txt"
    cluster_file = "clustered_40_numbered_with_headers.fasta.clstr"
    output_file = "clustered_40_numbered_annotated.clstr"

    # load YES predictions
    print("loading YES predictions...")
    pred_info = load_pred_info(pred_file)

    # annotate clusters based on predictions
    print("annotating clusters...")
    annotated_lines, match_count = annotate_clusters(cluster_file, pred_info)

    # save the annotated output
    print(f"saving to {output_file}...")
    Path(output_file).write_text("".join(annotated_lines))

    print(f"\n{match_count} YES matches found!")

if __name__ == "__main__":
    main()
```

Πρόγραμμα για Υπολογισμό ποσοστού β βαρέλιων / συνολικές αλληλουχίες σε κάθε cluster **και απομόνωση μόνο αυτών με ποσοστό >50% ,>60% και >70%**

```
import re

# get all (YES) b-barrel proteins from prediction file
def get_yes_predictions(file_path):
    yes_data = {}
    with open(file_path, 'r') as file:
        for line in file:
            if line.strip().endswith("YES"):
                parts = line.strip().split(',')
                header = parts[0].strip()
                score = parts[1].strip()
                yes_data[header] = f"{score},YES"
    return yes_data

# for each cluster file and annotate sequences if they are YES b-barrel predictions
def put_b_barrels_on_clusters(cluster_path, yes_data, output_path):
    total_matches = 0
    result = []
    current_cluster = None
    yes_count = 0
    total_count = 0
    lines = []
    clusters_with_many_barrels = {}

    with open(cluster_path, 'r') as file:
        for line in file:
            if line.startswith(">Cluster"):
                # handle previous cluster before begin with the next
                if current_cluster is not None:
                    percent = round((yes_count / total_count) * 100, 1) if total_count > 0 else 0 #calculate perc
                    result.append(f">Cluster {current_cluster} {yes_count}/{total_count} ({percent}%) WITH B
BARRELS\n")
                    result.extend(lines)
                    clusters_with_many_barrels[current_cluster] = yes_count
                # start new cluster
                current_cluster = line.strip().split()[1]



                yes_count = 0
                total_count = 0
```

```
            lines = []
        else:
            total_count += 1
            modified = line
            if '>' in line:

                match = re.search(r'>\s*([\w|,\s-]+?)(?:\s+at|\s*$)', line)
                if match:
                    header = match.group(1).strip()
                    if header in yes_data:
                        modified = line.rstrip() + f",{yes_data[header]}\n"
                        yes_count += 1
                        total_matches += 1
            lines.append(modified)

    # for the last cluster
    if current_cluster is not None:
        percent = round((yes_count / total_count) * 100, 1) if total_count > 0 else 0
        result.append(f">Cluster {current_cluster} {yes_count}/{total_count} ({percent}%) WITH B
BARRELS\n")
        result.extend(lines)
        clusters_with_many_barrels[current_cluster] = yes_count

with open(output_path, 'w') as out:
    out.writelines(result)

# extract clusters with high  percentange (50 %, 60%, 70 %)of b-barrel and save them to 3 CSV files
def export_filtered_cluster_headers(output_path):
    import csv

    over_50 = []
    over_60 = []
    over_70 = []

    with open(output_path, 'r') as file:
        for line in file:

            if line.startswith(">Cluster"):
                match = re.search(r'>Cluster\s+(\d+)\s+(\d+)/(\d+)\s+\((\d+\.?\d*)%\)', line)
                if match:
                    cluster_id = match.group(1)
                    yes = int(match.group(2))
                    total = int(match.group(3))
                    percent = float(match.group(4))
```

```
                if total >= 5:

                    row = [cluster_id, yes, total, percent]
                    if percent > 50:
                        over_50.append(row)
                    if percent > 60:
                        over_60.append(row)
                    if percent > 70:
                        over_70.append(row)

    with open("clusters_over_50.csv", 'w', newline='') as f50:
        writer = csv.writer(f50)
        writer.writerow(["Cluster Number", "B barrels", "Total of sequences", "Percent"])
        writer.writerows(over_50)

    with open("clusters_over_60.csv", 'w', newline='') as f60:
        writer = csv.writer(f60)
        writer.writerow(["Cluster Number", "B barrels", "Total of sequences", "Percent"])
        writer.writerows(over_60)

    with open("clusters_over_70.csv", 'w', newline='') as f70:
        writer = csv.writer(f70)
        writer.writerow(["Cluster Number", "B barrels", "Total of sequences", "Percent"])
        writer.writerows(over_70)

    print("clusters saved: clusters_over_50.csv, clusters_over_60.csv, clusters_over_70.csv")

# parse a fasta file and return a dictionary (header: sequence)
def parse_fasta(fasta_path):
    sequences = {}
    with open(fasta_path, 'r') as file:
        header = None
        seq_lines = []
        for line in file:
            line = line.strip()
            if line.startswith('>'):
                if header:
                    sequences[header] = ''.join(seq_lines)
                header = line[1:].strip()
                seq_lines = []
            else:
                seq_lines.append(line)
        if header:
            sequences[header] = ''.join(seq_lines)
    return sequences
```

```
# enrich a cluster file with real sequences from the fasta file
def enrich_cluster_file(cluster_file_path, fasta_sequences, output_path):
    output_lines = []
    with open(cluster_file_path, 'r') as f:
        for line in f:
            output_lines.append(line)
            if '>' in line:
                match = line.strip().split('>', 1)[1].split(',')[0].split(' at')[0].strip()
                if match in fasta_sequences:
                    output_lines.append(f"{fasta_sequences[match]}\n")
    with open(output_path, 'w') as out:
        out.writelines(output_lines)
    print(f"done: {output_path}")

def main():
    pred_file = 'predtmbb_scores.txt'
    cluster_file = 'clustered_40_numbered_with_headers.clstr'
    output_file = 'clustered_with_pred_matches.txt'

    #  load YES predictions (b-barrels)
    yes_predictions = get_yes_predictions(pred_file)

    #  annotate clusters with YES predictions
    put_b_barrels_on_clusters(cluster_file, yes_predictions, output_file)

    # extract high-percentage clusters to CSV
    export_filtered_cluster_headers(output_file)

    # add sequences into the clusters

    from pathlib import Path
    fasta_file = 'grouped_filtered_results.fasta'
    cluster_files = [
        'clusters_over_50.csv',
        'clusters_over_60.csv',
        'clusters_over_70.csv'
    ]
    fasta_data = parse_fasta(fasta_file)

    for cluster_path in cluster_files:
        base = Path(cluster_path).stem
        output_path = f"{base}_with_sequences.txt"
        enrich_cluster_file(cluster_path, fasta_data, output_path)
```

```
if __name__ == "__main__":

    main()
```

Πρόγραμμα για την εισαγωγή της αλληλουχίας των πρωτεϊνών των clusters **στο αρχείο με τα αποτελέσματα του** CD-HIT

```
import re
from pathlib import Path

# parse a fasta file and store each sequence under a key extracted that  it is itsheader
def parse_fasta(fasta_path):
    sequences = {}
    with open(fasta_path, 'r') as file:
        header = None
        seq_lines = []
        for line in file:
            line = line.strip()
            if line.startswith('>'):
                if header:
                    sequences[header] = ''.join(seq_lines)
                raw_header = line[1:]
                header_key = extract_fasta_key(raw_header)
                header = header_key
                seq_lines = []
            else:
                seq_lines.append(line)
        if header:
            sequences[header] = ''.join(seq_lines)
    return sequences


# extract a unique key from fasta header (e.g. A0A7X1G036|1323-1672)
def extract_fasta_key(header_line):
    # ([^|]+): anything before the first |.*?: anything after([0-9]+-[0-9]+): gap like 1323-1668
"A0A7X1G036|CarboxypepD_reg, SdrD_B| 1323-1672"
    match = re.match(r'([^|]+)\|.*?\|\s*([0-9]+-[0-9]+)', header_line)
    if match:
        return f"{match.group(1)}|{match.group(2)}"
    return header_line.strip()

# extract the same kind of key from a line in the cluster file
def extract_cluster_key(line):
    # line: ">A0A7X1G036|CarboxypepD_reg, SdrD_B| 1323-1672 at 49.43%"
    match = re.search(r'>([^|]+)\|.*?\|\s*([0-9]+-[0-9]+)', line)
```

```
    if match:
        return f"{match.group(1)}|{match.group(2)}"
    return None

# add sequences from fasta into cluster file if keys match
def enrich_cluster_file(cluster_file_path, fasta_sequences, output_path):
    output_lines = []
    not_found = []
    with open(cluster_file_path, 'r') as f:
        for line in f:
            output_lines.append(line)
            if '>' in line and not line.startswith('>Cluster'):
                key = extract_cluster_key(line)
                sequence = fasta_sequences.get(key)
                if sequence:
                    output_lines.append(f"{sequence}\n")
                else:
                    not_found.append(key)
    with open(output_path, 'w') as out:
        out.writelines(output_lines)

    print(f"done: {output_path}")
    if not_found:
        print("not found:")
        for nf in not_found:
            print(f"  - {nf}")

fasta_file = 'grouped_filtered_results.fasta'
cluster_files = [
    'clusters_over_50.txt',
    'clusters_over_60.txt',
    'clusters_over_70.txt'
]

# load sequences from fasta
fasta_data = parse_fasta(fasta_file)

# add sequences to each filtered cluster file
for cluster_path in cluster_files:
    base = Path(cluster_path).stem
    output_path = f"{base}_with_sequences.txt"
    enrich_cluster_file(cluster_path, fasta_data, output_path)
```

Πρόγραμμα για δημιουργία ενιαίου fasta file **αρχείου με τις αλληλουχίες που περιέχει κάθε** cluster **και εκτέλεση** clustalo **για κάθε ένα από αυτά**

```
import os
import re
import subprocess

def extract_clusters_to_fasta(input_file, output_dir):

    # creates fasta files for each cluster with the gap region sequences of the cluster

    os.makedirs(output_dir, exist_ok=True)
    current_cluster = None
    current_entries = []
    next_line_is_sequence = False

    with open(input_file, "r") as f:
        lines = f.readlines()

    for line in lines:
        line = line.strip()

        if line.startswith(">Cluster"):

            # if new cluster, save the previous one

            if current_cluster and current_entries:

                out_path = os.path.join(output_dir, f"{current_cluster}.fasta")
                with open(out_path, "w") as out_f:
                    out_f.write("\n".join(current_entries) + "\n")

            # extract the cluster number and set it as the current cluster

            match = re.search(r"Cluster (\d+)", line)
            if match:

                current_cluster = f"Cluster_{match.group(1)}"
                current_entries = []

        elif re.match(r"^\d+\s+\d+aa, >", line):

            # if find a line of a sequence, extract the header
            match = re.search(r">([^,\s]+)\|([^\|]+)\|\s*([\d\-]+)", line)
            if match:
                acc, label, region = match.groups()
                header = f"{acc}|{label}|{region.replace(' ', '')}"
```

```
                current_entries.append(f">{header}")
                next_line_is_sequence = True

            elif next_line_is_sequence and re.match(r"^[A-Z]+$", line):

                #  next line contains the actual sequence add it to the list
                current_entries.append(line)
                next_line_is_sequence = False

        # save the last cluster
        if current_cluster and current_entries:

            out_path = os.path.join(output_dir, f"{current_cluster}.fasta")
            with open(out_path, "w") as out_f:
                out_f.write("\n".join(current_entries) + "\n")

        print(f"Fasta saved on  '{output_dir}'.")

    def run_clustal_omega(fasta_dir, output_dir):

        # run Clustal Omega on each FASTA file
        os.makedirs(output_dir, exist_ok=True)

        for fasta_file in os.listdir(fasta_dir):

            if fasta_file.endswith(".fasta"):

                input_path = os.path.join(fasta_dir, fasta_file)
                output_file = os.path.splitext(fasta_file)[0] + ".aln"
                output_path = os.path.join(output_dir, output_file)

                cmd = [
                    "clustalo",
                    "-i", input_path,
                    "-o", output_path,
                    "--force",      # overwrites existing  file if needed
                    "--outfmt=clu", # use Clustal format for the alignment
                    "--wrap=80"     # wrap sequences every 80 characters for readability
                ]

                print(f"Running Clustal Omega for: {fasta_file}")
                subprocess.run(cmd, check=True)

        print(f"Alignment files saved in '{output_dir}'.\n")
```

```
def process_clusters_for_threshold(threshold):

    # for the 3 thresholds files, do:
    # generate FASTA files and perform multiple sequence alignment

    input_file = f"clusters_over_{threshold}_with_sequences.txt"

    fasta_dir = f"clustal_{threshold}"
    aln_dir = f"clustal_alignments_{threshold}"

    print(f"\nProcessing threshold {threshold}")
    extract_clusters_to_fasta(input_file, fasta_dir)
    run_clustal_omega(fasta_dir, aln_dir)

if __name__ == "__main__":

    thresholds = [50, 60, 70]
    for t in thresholds:
        process_clusters_for_threshold(t)
```

Πρόγραμμα για εκτέλεση hmmscan **με** e-value=0.**1 με την βάση** OMPdb_2022_11_17.hmm

```
import subprocess
import tempfile
import os
import re

# Input cluster file
input_file = 'clusters_over_70_with_sequences.txt'

# Output file where we will write results
output_file = 'annotated_clusters_output.txt'

# HMM file from OMPdb
hmm_db = 'OMPdb_2022_11_17.hmm'

# Run hmmscan for a single sequence creating  a temporary FASTA file
def scan_sequence(name, seq):
    #  temporary FASTA file for the sequence
    with tempfile.NamedTemporaryFile(delete=False, suffix=".fasta", mode="w") as tmp_fasta:
        tmp_fasta.write(f">{name}\n{seq}\n")
        fasta_path = tmp_fasta.name
```

```
    # Output file for domtblout result from hmmscan
    out_file = fasta_path + ".out"

 # Run hmmscan with domtblout output format
    subprocess.run(
        ["hmmscan", "--domtblout", out_file, "--cpu", "1", hmm_db, fasta_path]
    )
    hits = []
    with open(out_file) as f:
        for line in f:
            # skip empty lines or empty lines
            if line.startswith(">") or not line.strip():
                continue
            parts = line.strip().split()
            if len(parts) >= 13:
                try:
                    name = parts[0]
                    evalue = float(parts[12]) #e value is on the 12
                    # only keep hits with E-value < 0.1
                    if evalue < 0.1:
                        hits.append((name, evalue))
                except:
                    continue

    # clean temporary files
    os.remove(fasta_path)
    os.remove(out_file)

    return hits  # Return list of hits (domain name, evalue)

# Counters
total = 0
hits = 0

# Open input an output files
with open(input_file) as f, open(output_file, 'w') as out:
    lines = f.readlines()
    i = 0
    current_cluster_line = ""
    domain_yes_count = 0  # how many sequences in this cluster had a domain
```

```
cluster_seq_count = 0
cluster_lines = []

while i < len(lines):
    line = lines[i].strip()

    # Check if line is a new cluster header (e.g., >Cluster 123)
    if line.startswith(">Cluster"):

        # if this is not the first cluster, write the previous cluster to output
        if current_cluster_line:
            updated_line = current_cluster_line.strip() + f"  {domain_yes_count}\n"
            out.write(updated_line)
            out.writelines(cluster_lines)

        # start new cluster
        current_cluster_line = lines[i]
        cluster_lines = []
        domain_yes_count = 0
        cluster_seq_count = 0
        i += 1
        continue

    # check if line starts with for example(0 123aa, >...)
    if re.match(r"^\d+\s+\d+aa,", line):
        header = line
        seq = lines[i + 1].strip()  # sequence is the next line
        total += 1
        cluster_seq_count += 1

        found = scan_sequence(header, seq)  # run hmmscan
        if found:
            hits += 1
            if "YES" in header.upper():
                domain_yes_count += 1
            # Append name and e-value to header line
            header += "  " + " | ".join(f"{d[0]} E={d[1]:.4g}" for d in found)

        # save updated header and sequence to output cluster buffer
        cluster_lines.append(header + "\n")
        cluster_lines.append(seq + "\n")
```

```
            i += 2  # move to next sequence
        else:
            cluster_lines.append(lines[i])
            i += 1

    # write  last cluster
    if current_cluster_line:
        updated_line = current_cluster_line.strip() + f"  {domain_yes_count}\n"
        out.write(updated_line)
        out.writelines(cluster_lines)


print(f"Saved to: {output_file})")
```

ΠΡΟΓΡΑΜΜΑ ΓΙΑ ΕΥΡΕΣΗ ΔΟΜΗ**Σ ΣΤΟ** ALPHAFOLD

```
import requests

# extract the protein accesions IDs from our cluster FASTA file
def extract_accession_ids(input_file):
    ids = set()  # Store unique IDs only, avoid the duplicate
    with open(input_file, "r", encoding="utf-8") as file:
        for line in file:
            line = line.strip()
            if line.startswith(">Cluster") or not line:
                continue  # Skip cluster headers and empty lines
            if ">" in line and "|" in line:
                try:
                    part = line.split(">")[1]
                    accession = part.split("|")[0]  # Get ID before the first |
                    ids.add(accession)
                except IndexError:
                    continue  # Skip if something is wrong in the line
    return list(ids)

# Check if there is a result in AlphaFold for each ID
def check_alphafold(ids, output_file="alphafold_results.txt"):
    base_url = "https://alphafold.ebi.ac.uk/api/prediction/"
    headers = {
        "User-Agent": "Mozilla/5.0"  # Makes request like it  comes from a browser
    }
```

```
    found = []      # IDs with AlphaFold structure
    not_found = []  # IDs without

    for acc in ids:
        url = base_url + acc
        print(f"\n {acc} in {url}")  # We are checking this id

        try:
            response = requests.get(url, headers=headers, timeout=10)  # wait 10 sec until timeout
            if response.status_code == 200:
                data = response.json()  # Convert response to Python dict
                if data:
                    print(f"found: {acc}")  # It found structure
                    found.append(acc)
                else:
                    print(f"not found: {acc}")  # No structure
                    not_found.append(acc)
            elif response.status_code == 404:
                print(f"not found: {acc}")  # Not in database
                not_found.append(acc)

    # save results in file
    with open(output_file, "w", encoding="utf-8") as f:
        f.write("Found\n")
        for acc in found:
            f.write(f"{acc}\n")
        f.write("\nNot found\n")
        for acc in not_found:
            f.write(f"{acc}\n")
    print(f"Results on: {output_file}")

# Add ALPHA/NOT ALPHA result to the original file
def annotate_with_alphafold(input_file, found_ids_file, output_file):
    with open(found_ids_file, "r", encoding="utf-8") as f:
        lines = f.readlines()

    found_ids = set()
    in_found = False

    for line in lines:
```

```
            line = line.strip()
            if line.lower() == "found":
                in_found = True
                continue
            if line.lower() == "not found":
                break
            if in_found and line:
                found_ids.add(line)

        with open(input_file, "r", encoding="utf-8") as infile, \
             open(output_file, "w", encoding="utf-8") as outfile:

            for line in infile:
                line_stripped = line.strip()
                if line_stripped.startswith(">Cluster") or not line_stripped:
                    outfile.write(line)  # Keep same cluster headers and empty lines
                    continue
                if ">" in line and "|" in line:
                    try:
                        part = line.split(">")[1]
                        accession = part.split("|")[0]
                        tag = "ALPHA" if accession in found_ids else "NOT ALPHA"
                        outfile.write(line.strip() + f"\t{tag}\n")
                    except IndexError:
                        outfile.write(line)  # In case of error
                else:
                    outfile.write(line)  # Unexpected format continue

    #  extract IDs from the cluster file
    input_file = "clusters_over_70_with_sequences.txt"
    accession_list = extract_accession_ids(input_file)

    #check which ones have AlphaFold structures
    check_alphafold(accession_list, output_file="alphafold_results.txt")

    # annotate the cluster file with the results of Alpha Fold
    annotate_with_alphafold(
        input_file="annotated_clusters_output",  # the input  file
        found_ids_file="alphafold_results.txt",  # list with results
        output_file="annotated_with_alphafold.txt"  #output file
    )
```