Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

ΕΝΤΟΠΙΣΜΟΣ K-Type RNA Pseudoknot με τεχνικές συντακτικής αναγνώρισης προτύπων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΤΟΥ

ΓΕΩΡΓΙΟΥ ΧΑΝΗ

Επιβλέπων : Παναγιώτης Τσανάκας, Καθηγητής Ε.Μ.Π.

Αθήνα, Ιανουάριος 2025

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

ΕΝΤΟΠΙΣΜΟΣ K-Type RNA Pseudoknot με τεχνικές συντακτικής αναγνώρισης προτύπων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΤΟΥ

ΓΕΩΡΓΙΟΥ ΧΑΝΗ

Επιβλέπων : Παναγιώτης Τσανάκας, Καθηγητής Ε.Μ.Π

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή στις 3/1/2025

Π. Τσανάκας	Δ. Κουτσουρής	Γ. Ματσόπουλος
Καθηγητής Ε.Μ.Π	Ομότιμος Καθηγήτης Ε.Μ.Π	Καθηγητής Ε.Μ.Π

Αθήνα, Ιανουάριος 2025

.....
ΧΑΝΗΣ ΓΕΩΡΓΙΟΣ

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

Περίληψη

Για πολλά χρόνια το RNA θεωρείτο ότι έπαιζε απλά έναν δευτερεύοντα ρόλο στους ζωντανούς οργανισμούς παρέχοντας την υπηρεσία της μεταφοράς πληροφοριών. Για τους περισσότερους επιστήμονες ήταν απλά το «όχημα» που μετασχηματίζει και μεταφέρει την γενετική πληροφορία στο ριβόσωμα. Αντιθέτως σήμερα είναι πια ευρέως αποδεκτό και γνωστό ότι το RNA είναι πολύ πιο πλούσιο και περίπλοκο στην δομή του από ότι το DNA. Επιπρόσθετα, παρότι συναντάται σαν μονόκλωνο μόριο, είναι δυνατόν να εμφανίζει δίκλωνες ελικοειδείς δομές και να διπλώνει σχηματίζοντας ποικίλες τριτοταγείς δομές. Αυτές είναι οι δομές που θα μελετήθούν την παρούσα εργασία και συγκεκριμένα δομές που περιέχουν άτυπα ζεύγη βάσεων που οι διαμορφώσεις τους στη δομή προσομοιάζουν με κόμπους. Ουσιαστικά γίνεται λόγος για το ζευγάρωμα βάσεων μεταξύ μη παρακείμενων αλληλουχιών με αποτέλεσμα τον σχηματισμό αυτών των πολύπλοκων δομών. Αυτές οι δομές ονομάζονται από την επιστήμη της Βιολογίας ως pseudoknots.

Η διερεύνηση της δευτεροταγούς δομής του RNA και συγκεκριμένα η πρόβλεψη της θέσης των pseudoknots τύπου K σε μια δοσμένη ακολουθία RNA είναι το ζήτημα με το οποίο ασχολείται η παρούσα διπλωματική εργασία. Στην εργασία θα παρουσιαστεί σε βάθος η διαδικασία και η ανάλυση των βημάτων με τα οποία φτάνουμε στο επιθυμητό αποτέλεσμα. Χρησιμοποιούνται κριτήρια όπως η ελάχιστη ελεύθερη ενέργεια αναδίπλωσης του RNA και η μεγιστοποίηση του αριθμού των ζευγών μεταξύ των βάσεων. Στη συνέχεια μέσω της ανάλυσης μας ξεχωρίζεται η βέλτιστη λύση καθώς σε κάθε δοσμένη ακολουθία παρατηρούνται πάνω από μία προτεινόμενες λύσεις. Τα παραπάνω γίνονται με δύο εργαλεία: με την χρήση μιας γραμματικής χωρίς συμφραζόμενα για την συντακτική αναγνώριση του pseudoknot και με έναν άπλητο αλγόριθμο δυναμικού προγραμματισμού για την εύρεση της τελικής λύσης. Ο σύνδεσμος για το σύνολο του κώδικα (project) της εργασίας παρατίθεται στην βιβλιογραφία [1].

Λέξεις κλειδιά

Ριβονουκλεϊκό οξύ, συντακτική αναγνώριση προτύπων, ψευδοκόμβος τύπου K, γραμματικές χωρίς συμφραζόμενα

Abstract

For many years RNA was thought to play only a secondary role in living organisms, that of providing the service of transferring information. For most scientists it was just the tool that transforms and transfers genetic information to the ribosomes. On the contrary, it is now widely accepted knowledge that RNA is much richer and more complex in its structure than the DNA itself. Additionally, although it occurs as a single-stranded molecule, it is possible to display double-stranded helical structures and also form a variety of tertiary structures. These are the structures that we will study in the present work, specifically the ones that contain atypical base pairs whose configurations in the structure resemble knots. Essentially we are talking about base pairing between non-adjacent sequences resulting in the formation of these complex structures. These structures are called pseudoknots by the science of Biology.

The investigation of the secondary structure of RNA and specifically the prediction of the location of K-type pseudoknots in a given RNA sequence is the issue with which this thesis deals. The paper will present in depth the process and the analysis of the steps by which we reach the desired result. Criteria such as the minimum free energy of RNA folding and the maximization of the number of base pairs are used. Then through our analysis we distinguish the optimal solution as in each given sequence we can have more than one proposed solution. The above is done with two tools: using a context-free grammar for the syntactic recognition of the pseudoknot and a greedy dynamic programming algorithm for finding the final solution. The link for all the code of the work is listed in the bibliography [1].

Keywords

RNA, syntactic pattern recognition, K-type pseudoknot, context-free grammars

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω το διδακτικό προσωπικό και τον καθηγητή που μου εμπιστεύτηκε αυτό το θέμα της διπλωματικής εργασίας και με βοήθησε να το φέρω εις πέρας. Επιπλέον θα ήθελα να ευχαριστήσω τους γονείς μου, τον αδερφό μου και την κοπέλα μου για την αμέριστη συμπαράσταση τους όλα αυτά τα χρόνια.

Περιεχόμενα

Κατάλογος Σχημάτων

1. Εισαγωγή

Το RNA (Ribonucleic Acid) αποτελεί ένα βασικό μόριο που εμπλέκεται σε πολλαπλές βιολογικές διαδικασίες, όπως η μετάφραση της γενετικής πληροφορίας από το DNA στην πρωτεϊνοσύνθεση. Συνιστά μονοκλωνικό μόριο που αποτελείται από νουκλεοτίδια, τα οποία περιλαμβάνουν τη ριβόζη, μια φωσφορική ομάδα και μία από τις τέσσερις αζωτούχες βάσεις (αδενίνη, ουρακίλη, γουανίνη, και κυτοσίνη). Η δευτεροταγής δομή του RNA διαμορφώνεται μέσω του ζευγαρώματος των βάσεων, όπως οι ζεύξεις A-U και G-C, δημιουργώντας έτσι αναδιπλώσεις και δομές. Αυτή η δομή είναι κρίσιμη για την κατανόηση της λειτουργίας του RNA, καθώς καθορίζει το πώς το RNA αλληλεπιδρά με άλλα μόρια και επιτελεί τις βιολογικές του λειτουργίες.

Το πρόβλημα της πρόβλεψης της δευτεροταγούς δομής του RNA είναι ένα σημαντικό και δύσκολο θέμα στη βιοπληροφορική. Η ακριβής πρόβλεψη του τρόπου με τον οποίο τα νουκλεοτίδια ζευγαρώνουν για να δημιουργήσουν τη δευτεροταγή δομή είναι κρίσιμη για την κατανόηση της λειτουργικότητας του RNA. Καθώς η δομή του RNA επηρεάζει τη βιολογική του δραστηριότητα, η πρόβλεψή της μπορεί να συμβάλει στην κατανόηση ασθενειών και στη σχεδίαση νέων φαρμάκων. Ωστόσο, η δομή δεν καθορίζεται απευθείας από την αλληλουχία του RNA, και οι υπάρχουσες πειραματικές μέθοδοι είναι συχνά δαπανηρές και χρονοβόρες. Για να αντιμετωπιστεί αυτό το πρόβλημα, έχουν αναπτυχθεί διάφορες μεθοδολογίες πρόβλεψης, οι οποίες περιλαμβάνουν αλγορίθμους δυναμικού προγραμματισμού, στατιστικά μοντέλα, και μεθόδους μηχανικής μάθησης.

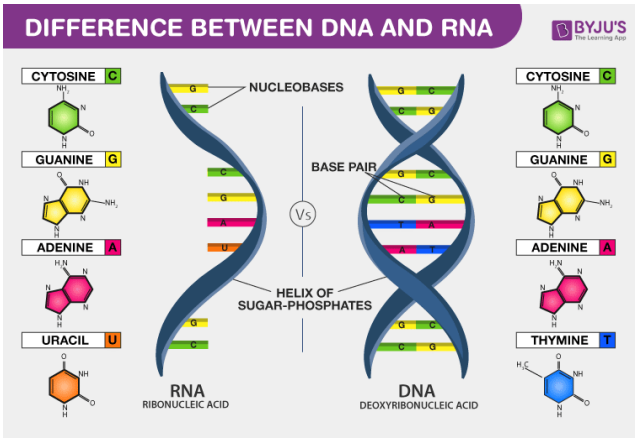
Στην παρούσα διπλωματική, θα ασχοληθούμε με την πρόβλεψη του ψευδοκόμβου τύπου K σε μια ακολουθία RNA. Βασιζόμενοι στη δημοσίευση knotify, στην οποία το πρόβλημα έχει επιλυθεί για τύπου H ψευδοκόμβους, θα προβούμε στις κατάλληλες τροποποιήσεις ώστε να φτάσουμε σε αποτέλεσμα. Η εργασία χωρίζεται σε 7 κεφάλαια. Καθένα από τα κεφάλαια αναλύει μια διαφορετική πτυχή της μελέτης και επίλυσης του προβλήματος της πρόβλεψης της δευτεροταγούς δομής του RNA. Στο κεφάλαιο 2, παρουσιάζεται η θεωρία γύρω από το RNA, εστιάζοντας στις κατηγορίες του RNA, όπως το mRNA, tRNA και rRNA, καθώς και στις διαφορετικές κατηγορίες δευτεροταγών δομών του, όπως οι βρόχοι και οι ψευδόκομβοι. Στόχος αυτού του κεφαλαίου είναι να παράσχει το απαραίτητο θεωρητικό υπόβαθρο για την κατανόηση της δομής και λειτουργίας του RNA. Στο κεφάλαιο 3, αναπτύσσονται οι θεωρητικές έννοιες που σχετίζονται με το πρόβλημα της συντακτικής αναγνώρισης προτύπων. Αναλύονται έννοιες όπως οι γραμματικές, τα συντακτικά δέντρα και ο αλγόριθμος YAEP, τα οποία χρησιμοποιούνται για τη μοντελοποίηση της δευτεροταγούς δομής του RNA. Στη συνέχεια, το κεφάλαιο 4 εστιάζει στην επίλυση του προβλήματος της αναγνώρισης του ψευδοκόμβου τύπου K, με χρήση συντακτικής αναγνώρισης προτύπων. Περιλαμβάνει τη δημιουργία μιας γραμματικής, την ανάλυσή της, την παραγωγή συντακτικού δέντρου και τη διάσχισή του για την αναγνώριση των ψευδοκόμβων. Επιπλέον, γίνεται χρήση ενός άπληστου

αλγορίθμου δυναμικού προγραμματισμού για τη υλοποίηση της διαδικασίας και με εναλλακτικό τρόπο. Στο κεφάλαιο 5, επιδεικνύεται η λειτουργία του αλγορίθμου μας μέσω εικόνων και παραδειγμάτων, προσφέροντας μια πιο οπτική κατανόηση της λύσης. Επιπρόσθετα το κεφάλαιο 6 περιλαμβάνει τη σύγκριση της μεθοδολογίας μας με άλλες γνωστές μεθόδους, αναλύοντας τα πλεονεκτήματα και τις αδυναμίες της. Τέλος, στο κεφάλαιο 7, συνοψίζονται τα συμπεράσματα της έρευνας και προτείνονται μελλοντικές επεκτάσεις της παρούσας μεθόδου, με στόχο τη βελτίωση της ακρίβειας και της απόδοσης στην πρόβλεψη της δευτεροταγούς δομής του RNA.

2. Θεωρία γύρω από το RNA

2.1 Τι είναι το RNA

Το RNA είναι ένα μακρομόριο το οποίο παίζει κρίσιμο ρόλο στη μετάδοση γενετικών πληροφοριών εντός των κυττάρων, λειτουργώντας παράλληλα με το DNA. Το RNA έχει αρκετές ομοιότητες με το DNA στη βασική του δομή, αποτελούμενο από νουκλεοτίδια (ριβόζη), ένα φωσφορικό οξύ, και τις εξής βάσεις αζώτου: αδενίνη (A), γουανίνη (G), κυτοσίνη (C) και ουρακίλη (U) στην περίπτωση του RNA (αντικαθιστώντας την θυμίνη που βρίσκεται στο DNA). Αυτά τα νουκλεοτίδια σχηματίζουν μονόκλωνα RNA μέσω φωσφοδιεστερικών δεσμών.
Η κύρια λειτουργία του RNA είναι να μεταφράσει γενετικές πληροφορίες από το DNA σε πρωτεΐνες, μια διαδικασία γνωστή ως σύνθεση πρωτεΐνης ή και μετάφραση. Αυτό συμβαίνει από τρία βασικά είδη RNA: το αγγελιοφόρο RNA (mRNA), το μεταφορικό RNA (tRNA) και το ριβοσωμικό RNA (rRNA). Το mRNA μεταφέρει γενετικές πληροφορίες από τον κυτταρικό πυρήνα στα ριβοσώματα, όπου πραγματοποιείται η πρωτεϊνική σύνθεση. Το tRNA μεταφέρει αμινοξέα στα ριβοσώματα με βάση τις οδηγίες του mRNA, διευκολύνοντας τη συναρμολόγηση των πρωτεϊνών. Τέλος, το rRNA καταλύει την πρωτεϊνοσύνθεση στο κύτταρο
[2].
Σε σύγκριση με το DNA, που χρησιμεύει ως “αποθήκη” γενετικών πληροφοριών, το RNA λειτουργεί ως ο μεσολαβητής και αγγελιοφόρος που μεταφράζει το γενετικό κώδικα σε λειτουργικές πρωτεΐνες. Το DNA συνήθως υπάρχει ως διπλή αλυσίδα (δίκλωνο), ενώ το RNA είναι μονόκλωνο. Επιπλέον, η παρουσία της ουρακίλης στο RNA το διαφοροποιεί από το DNA το οποίο και περιέχει θυμίνη.
Η κατανόηση των ρόλων και των δομών του RNA και του DNA είναι θεμελιώδης για την αποκωδικοποίηση της πολυπλοκότητας της μοριακής βιολογίας.



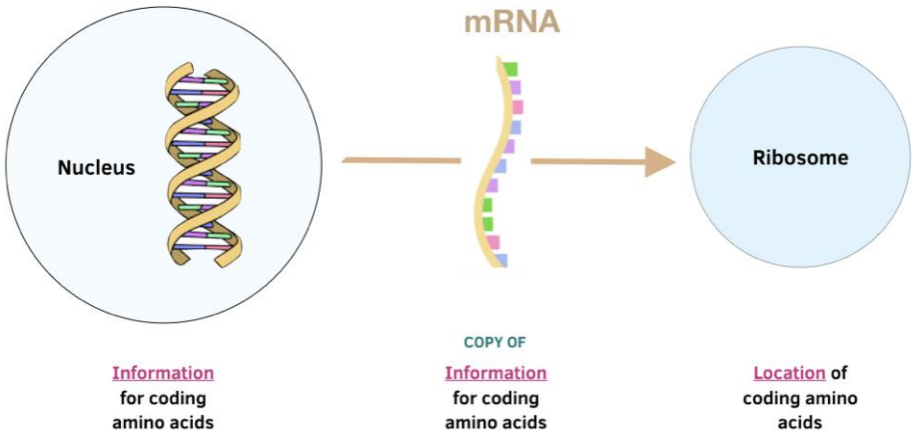
Σχήμα 2.1: *DNA και RNA [3]*

2.2 Οι κατηγορίες του RNA

Με βάση την θεμελιώδη κατανόηση του RNA που παρουσιάστηκε στο προηγούμενο κεφάλαιο, σε αυτό το κεφάλαιο παρέχεται μια εις βάθος διερεύνηση και παρουσίαση των διαφορετικών μορφών RNA. Κάθε τύπος RNA συμβάλλει με μοναδικό τρόπο στην ενορχήστρωση της γονιδιακής έκφρασης και έτσι η παρουσίαση τους ενισχύει την κατανόηση των μοριακών μηχανισμών που διέπουν τις κυτταρικές λειτουργίες.

2.2.1 Αγγελιοφόρο RNA(mRNA)

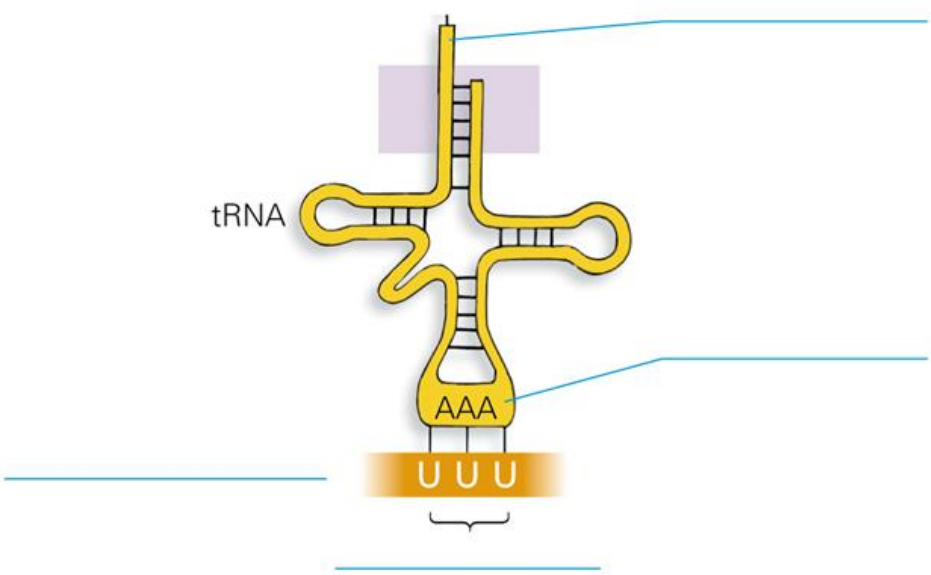
Το αγγελιοφόρο RNA (mRNA) παίζει το ρόλο ενός ενδιάμεσου κόμβου στο πολύπλοκη διαδικασία που ξεκινάει από το DNA και καταλήγει στη σύνθεση των λειτουργικών πρωτεϊνών. Το mRNA, που συντίθεται εντός των ορίων του κυτταρικού πυρήνα, ενθυλακώνει τις περίπλοκες γενετικές οδηγίες που είναι ενσωματωμένες στο DNA. Ο βασικός του ρόλος έγκειται στη μεταφορά αυτού του γενετικού κώδικα στα ριβοσώματα, τα οποία βρίσκονται στο κυτταρόπλασμα, όπου λαμβάνει χώρα η πρωτεϊνική σύνθεση. Κάθε μόριο mRNA λειτουργεί ως ένα μοναδικό αντίγραφο, που φέρει τις πληροφορίες για μια συγκεκριμένη πρωτεΐνη. Αυτή η δυναμική διαδικασία μεταγραφής και μετάφρασης είναι θεμελιώδης για την κυτταρική λειτουργία, ενορχηστρώνοντας τη σύνθεση πρωτεϊνών απαραίτητων για τις μυριάδες διεργασίες στους ζωντανούς οργανισμούς.



Σχήμα 2.2: *Η λειτουργία του mRNA [4]*

2.2.2 Μεταφορικό RNA(tRNA)

Το μεταφορικό RNA (tRNA) αναδύεται επίσης ως κρίσιμο παράγοντας στην περίπλοκη συμφωνία της πρωτεϊνοσύνθεσης. Πέρα από τη φαινομενικά απλοϊκή δομή του, το tRNA αναλαμβάνει έναν κεντρικό ρόλο ως μοριακός προσαρμογέας, συνδέοντας τη γλώσσα των νουκλεϊκών οξέων με αυτή των αμινοξέων. Φορτωμένο με συγκεκριμένα αμινοξέα, το tRNA γεφυρώνει το χάσμα μεταξύ του γενετικού κώδικα που εγγράφεται στο mRNA και της πραγματικής σύνθεσης των πρωτεϊνών. Η μοναδική του δομή, που διαθέτει σχήμα τριφυλλιού, εξασφαλίζει ακρίβεια στο ζευγάρωμα αμινοξέων με τα αντίστοιχα κωδικόνια. Αυτή η ακρίβεια είναι υψίστης σημασίας για την πιστότητα της μετάφρασης, αποτρέποντας οποιαδήποτε λάθη κατά τη διάρκεια της συναρμολόγησης της πρωτεΐνης. Ένας τύπος tRNA μπορεί να αντιστοιχεί σε πολλά κωδικόνια αλλά μόνο σε ένα αμινοξύ.



Σχήμα 2.3: *Η δομή του tRNA [5]*

2.2.3 Ριβοσωμικό RNA(rRNA)

Το ριβοσωμικό RNA (rRNA) βρίσκεται στο επίκεντρο της περίπλοκης διαδικασίας της πρωτεϊνικής σύνθεσης ως αναπόσπαστο συστατικό των ριβοσωμάτων, του κυτταρικού μηχανισμού που ενορχηστρώνει αυτή τη διαδικασία. Εμφανίζεται τόσο σε μικρές όσο και σε μεγάλες υπομονάδες και ο ρόλος του είναι να καταλύει το σχηματισμό πεπτιδικών δεσμών μεταξύ αμινοξέων. Αυτή η καταλυτική ικανότητα είναι απαραίτητη για την επιμήκυνση και την ωρίμανση των πολυπεπτιδικών αλυσίδων κατά τη μετάφραση. Το ριβόσωμα, με το αντίστοιχο rRNA του ως οδηγό, εξασφαλίζει την ακριβή και διαδοχική συναρμολόγηση των αμινοξέων,

με αποκορύφωμα τη σύνθεση λειτουργικών πρωτεϊνών. Πέρα από την καταλυτική του λειτουργία, το rRNA συμβάλλει επίσης και στη δομική σταθερότητα των ριβοσωμάτων. Τέλος, αξίζει να σημειωθεί ότι αποτελούν το 80% των συνολικών RNA σε ένα τυπικό ευκαρυωτικό κύτταρο.

2.2.4 Μικρό πυρηνικό RNA (snRNA)

Τα μικρά πυρηνικά RNA (snRNA) είναι μικρά μόρια RNA που βρίσκονται στον πυρήνα των κυττάρων των ευκαρυωτικών οργανισμών. Παίζουν κρίσιμο ρόλο στην επεξεργασία του RNA, ειδικότερα στο splicing των προ-mRNA μεταγραφών κατά τη διαδικασία έκφρασης του γονιδίου. Τα snRNA συνδυάζονται με πρωτεΐνες για να δημιουργήσουν ριβονουκλεοπρωτεΐνες (snRNPs). Τα σωματίδια αυτά καταλύουν την ¨ωρίμανση" του mRNA.

2.2.5 Μικρο-RNA (miRNA)

Τα μικρο-RNA (miRNAs) είναι μικρά μόρια RNA, που συνήθως αποτελούνται από περίπου 22 νουκλεοτίδια, τα οποία βρίσκονται σε ευκαρυωτικά κύτταρα. Παίζουν κρίσιμο ρόλο στη γονιδιακή ρύθμιση μετά την μεταγραφή με την προσκόλληση τους σε συμπληρωματικές αλληλουχίες του mRNA. Αυτή η δέσμευση συνήθως οδηγεί είτε σε αποικοδόμηση του mRNA είτε σε μεταφραστική καταστολή, ελέγχοντας έτσι την έκφραση του γονιδίου. Τα miRNA εμπλέκονται σε διάφορες βιολογικές διεργασίες, συμπεριλαμβανομένης της ανάπτυξης και της διαφοροποίησης των κυττάρων. Συμβάλλουν σημαντικά στη ρύθμιση των δικτύων γονιδιακής έκφρασης και για αυτό η απορρύθμιση της έκφρασης τους έχει συνδεθεί με πολλές ασθένειες, συμπεριλαμβανομένου του καρκίνου, των καρδιαγγειακών διαταραχών και των νευρολογικών παθήσεων.

2.2.6 Μικρό παρεμβατικό RNA (siRNA)

Τα μικρά παρεμβατικά RNA (siRNA) είναι δίκλωνα μόρια RNA, τυπικά μήκους περίπου 20-25 ζευγών βάσεων, που παίζουν κρίσιμο ρόλο στην παρεμβολή του RNA. Η παρεμβολή του RNA είναι ένας μηχανισμός ρύθμισης της γονιδιακής έκφρασης κατά την οποία το ίδιο το RNA καταστέλλει την έκφραση συμπληρωματικών προς αυτό γονιδίων.

2.2.7 Μακρύ μη κωδικοποιητικό RNA (incRNA)

Τα μακρά μη κωδικοποιητικά RNA (incRNA) είναι μόρια μεγαλύτερα από 200 νουκλεοτίδια τα οποία ποτέ τους δεν μεταφράζονται σε πρωτεΐνες. Κάποια από αυτά κωδικοποιούν για μικρά πεπτίδια αλλά τα περισσότερα δεν μεταφράζονται καθόλου. Αλληλεπιδρούν με πρωτεΐνες, με άλλα μόρια RNA και DNA και συνήθως ελέγχονται από τα miRNA.

2.2.8 Piwi αλληλεπιδρών RNA (piRNA)

Τα piwi αλληλεπιδρώντα RNA (piRNA) είναι μια κατηγορία μικρών μη κωδικοποιητικών μορίων RNA, τυπικά μήκους 24 έως 31 νουκλεοτιδίων. Αλληλεπιδρούν αποκλειστικά με τις πρωτεΐνες Piwi και βρίσκονται κυρίως σε ζωικά κύτταρα. Παίζουν έναν ουσιαστικό ρόλο στη σίγαση των τρανσποζονίων, στη διατήρηση της σταθερότητας του γονιδιώματος και στην ανάπτυξη των γεννητικών κυττάρων.

2.2.9 Διαγονιδιακό μη-κωδικοποιητικό RNA (lincRNA)

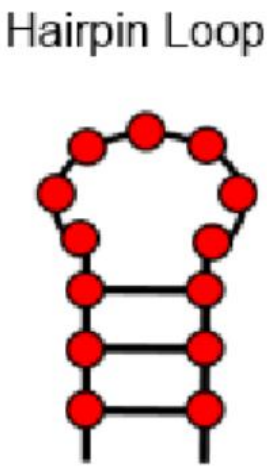
Τα διαγονιδιακά μη-κωδικοποιητικά μόρια RNA (lincRNA) αποτελούν μια υποκατηγορία των lncRNAS και βρίσκονται μεταξύ γονιδίων που κωδικοποιούν πρωτεΐνες. Τυπικά, είναι μεγαλύτερα από 200 νουκλεοτίδια και παίζουν ποικίλους ρυθμιστικούς ρόλους στην έκφραση των γονιδίων, της αναδιαμόρφωση της χρωματίνης και άλλες κυτταρικές διεργασίες.

2.3 Κατηγορίες δομών εντός του RNA

Οι πολύπλοκες δομές που παρουσιάζονται εντός των μορίων RNA διαδραματίζουν θεμελιώδη ρόλο στις διαφορετικές λειτουργίες αυτών. Σε αυτό το κεφάλαιο, εξετάζονται οι διαφορετικές αυτές δομές του RNA και αναδεικνύεται η σημασία τους στις βιολογικές διεργασίες και τις μοριακές αλληλεπιδράσεις. Τα μόρια RNA εμφανίζουν μοναδική ικανότητα στο να υιοθετούν σύνθετες δομές, οι οποίες προκύπτουν από συγκεκριμένες αλληλεπιδράσεις μεταξύ των βάσεών τους και τριτογενών στοιχείων. Η κατανόηση αυτών των δομών αποτελεί ζωτικό παράγοντα για την αποκωδικοποίηση των μηχανισμών που ρυθμίζουν διαδικασίες, όπως ο έλεγχος της έκφρασης γονιδίων και η μοριακή αναγνώριση.

2.3.1 Hairpin Loop

Η δομή hairpin loop δημιουργείται όταν ένα κομμάτι ενός μονόκλωνου RNA διπλώνεται και σχηματίζει μια διπλή έλικα μένα ένα άλλο τμήμα του ίδιο νήματος. Μπορούν επίσης να δημιουργηθούν σε μόρια DNA αλλά παρατηρούνται συνήθως στο mRNA. Συμβάλουν στην σταθερότητα της δομής του mRNA[6].



Σχήμα 2.4: *Hairpin Loop [7]*

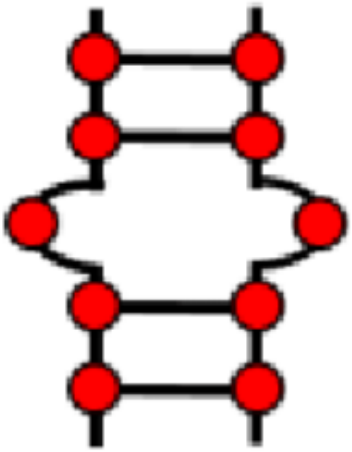
2.3.2 Kissing Hairpin

Η δομή Kissing Hairpin ουσιαστικά πρόκειται για τη σύγκλιση δύο βάσεων από hairpin loops. Αυτή η αλληλεπίδραση δημιουργεί μια σταθερή διμερή δομή, συχνά κρίσιμη για διαδικασίες που διενεργούνται μέσω RNA, όπως η συσκευασία του γονιδίου ενός ιού, η διμεροποίηση του RNA και γενικότερα την ρύθμιση των γονιδίων. Το "Kissing Hairpin" επιτρέπει συγκεκριμένες αλληλεπιδράσεις RNA-RNA, επιτρέποντας τον σχηματισμό υψηλότερης τάξης σύνθετων RNA που εμπλέκονται σε ποικίλες βιολογικές λειτουργίες.

Σχήμα 2.5: *Kissing hairpins [8]*

2.3.3 Internal Loops

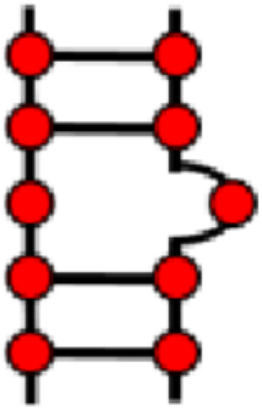
Τα Internal Loops είναι δομικά μοτίβα εντός των μορίων RNA, τα οποία χαρακτηρίζονται από την μη σύζευξη νουκλεοτιδίων εντός της διπλής αλυσίδας. Μπορούν να ποικίλουν σε μέγεθος και σύνθεση αλληλουχιών, επηρεάζοντας τόσο την σταθερότητα όσο και την δυναμική των δευτεροταγών δομών RNA. Διευκολύνουν επίσης την μοριακή αναγνώριση καθώς και την αλληλεπίδραση του μορίου με άλλα βιομόρια, όπως πρωτείνες ή μικρομόρια.



Σχήμα 2.6: *Internal Loop [7]*

2.3.4 Bulges

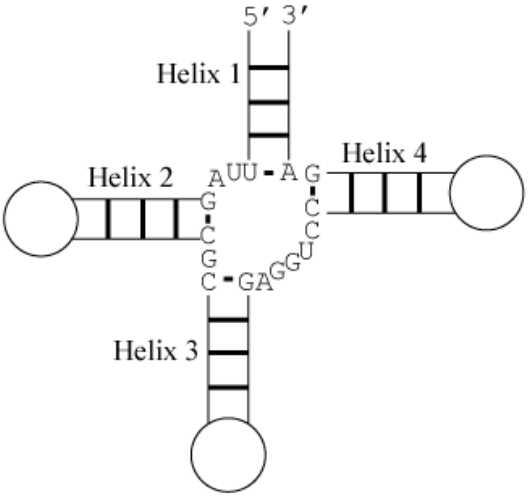
Τα bulges χαρακτηρίζονται από μη συζευγμένα νουκλεοτίδια τα οποία διακόπτουν τη συνέχεια της διπλής αλυσίδας. Ποικίλουν σε μέγεθος καθώς και στη σύνθεση της αλληλουχίας και συχνά λειτουργούν ως χώροι αναγνώρισης για πρωτεΐνες που συνδέονται με το RNA, διευκολύνοντας έτσι συγκεκριμένες αλληλεπιδράσεις RNA-πρωτεΐνης ή RNA-μικρομορίου.



Σχήμα 2.7: *Bulge στη δεξιά πλευρά της αλυσίδας [7]*

2.3.5 Multibranch Loops

Τα Multibranch Loops χαρακτηρίζονται από πολλαπλά μη συζευγμένα νουκλεοτίδια τα οποία ενώνονται σε μια κεντρική περιοχή του stem. Είναι αρκετά περίπλοκα loops τα οποία παίζουν μείζονα ρόλο στη σταθερότητα και στη λειτουργία του RNA. Σχηματικά τα ζευγάρια των βάσεων δημιουργούνε έναν κύκλο.

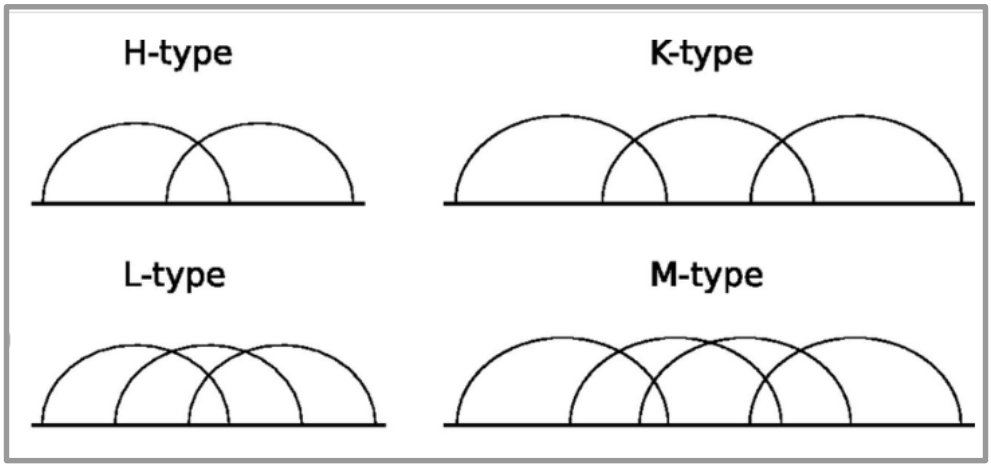


Σχήμα 2.8: *Multibranch loop [9]*

2.3.6 Pseudoknots

Τα pseudoknots ή ψευδοκόμβοι είναι δομικά στοιχεία που βρίσκονται σχεδόν σε όλες τις κατηγορίες του RNA. Αναγνωρίστηκαν πρώτη φορά στα γονιδιώματα των φυτών, όμως πλέον έχουν καθιερωθεί ως ένα ευρέως διαδεδομένο μοτίβο με ποικίλες λειτουργίες σε διάφορες βιολογικές διεργασίες. Δομικά, πρόκειται για μια δευτεροταγή δομή που περιέχει τουλάχιστον δύο stem-loops, εκ των οποίων το μισό του ενός παρεμβάλλεται ανάμεσα στα δύο μισά του άλλου. Η λειτουργία της δομής συχνά σχετίζεται με την αλληλεπίδραση με τα ριβοσώματα, την αναπαραγωγή ιών, την μετάφραση και τη ρύθμιση της έκφρασης των γονιδίων [11]. Συγκεκριμένα, στους ιούς οι ψευδοκόμβοι συχνά λειτουργούν ως ρυθμιστικά στοιχεία ελέγχοντας την έναρξη της μετάφρασης. Στο ριβοσωμικό RNA, συμβάλλουν στην διατήρηση της ριβοσωμικής δομής και στην ιδανική εκτέλεση της πρωτεινοσύνθεσης. Τέλος λειτουργούν ως χώροι αναγνώρισης για πρωτεΐνες που συνδέονται με το RNA επηρεάζοντας και με αυτόν τον τρόπο την μοριακή διαδικασία. Υπάρχουν τέσσερις βασικές κατηγορίες ψευδοκόμβων:

1. Ψευδοκόμβος τύπου H
2. Ψευδοκόμβος τύπου K
3. Ψευδοκόμβος τύπου L
4. Ψευδοκόμβος τύπου M



Σχήμα 2.9: *Τα 4 είδη ψευδοκόμβων* Brierley, I., & Dos Ramos, F. J. (2006). Programmed ribosomal frameshifting in HIV-1 and the SARS-CoV. *Nature Reviews Microbiology, 4*(11), 766–774. https://www.nature.com/articles/nrmicro1704 [12]

Στη φύση συναντάται πιο συχνά ο τύπος Η. Στην παρούσα διπλωματική θα επιλυθεί το προβλημα της αναγνώρισης του ψευδοκόμβου τύπου Κ στηριζόμενοι στην υλοποίηση της δημοσίευσης knotify [13] στην οποία επιλύθηκε το ίδιο πρόβλημα όσον αφορά τον τύπο Η.

3. Θεωρητικές έννοιες

3.1 Βιβλιογραφία σχετική με το υπάρχον πρόβλημα

Η πρόβλεψη της δευτεροταγούς δομής του RNA είναι ένα δομικό πρόβλημα στην επιστήμη της υπολογιστικής Βιολογίας και για αυτό έχουν αναπτυχθεί ποικίλοι μέθοδοι για την επίλυση του. Έχει αποδειχτεί ότι το πρόβλημα αυτό ανήκει στην κατηγορία των NP-complete προβλημάτων όταν η δομή RNA εμπεριέχει και ψευδοκόμβους.

Η πιο διαδεδομένη μορφή αντιμετώπισης και επίλυσης του συγκεκριμένου ζητηματος είναι ο δυναμικός προγραμματισμός. Χαρακτηριστικό παράδειγμα αποτελεί ο αλγόριθμος των Eddy και Rivas [14], ο οποίος επεξεργάζοντας θερμοδυναμικές παραμέτρους υπολογίζει την δομή RNA με την ελάχιστη ενέργεια. Αλγόριθμοι όπως του Nusssinov [15] και του Zucker [16] είναι επίσης δυναμικά παραδείγματα. Ο αλγόριθμος του Nussinov προβλέπει την δευτεροταγή δομή του RNA υπολογίζοντας τα μέγιστα δυνατά ζευγάρια βάσεων, ενώ ο αλγόριθμος του Zucker εισάγει θερμοδυναμικές παραμέτρες για να ελαχιστοποιήσει την ελάχιστη ελεύθερη ενέργεια. Και οι δύο αυτοί αλγόριθμοι έχουν πολυωνυμική χρονική πολυπλοκότητα, καθιστώντας τους επαρκείς για το πρόβλημα της πρόβλεψης.

Εφόσον αναφερθήκε ο παράγοντας της ελάχιστης ενέργειας (Minimum Free Energy-MFE) αξίζει να σημειωθεί ότι αποτελεί μέθοδο που αποκλειστικά επιχειρεί να προβλέψει την δευτεροταγή δομή. Υπαρκτό παράδειγμα αποτελεί το εργαλείο ViennaRNA [17] το οποίο χρησιμοποιεί μεταβλητές που αφορούν την ενέργεια των μορίων για να υπολογίσει την σταθερότητα των δομών RNA. Οι μέθοδοι MFE συχνά μας τροφοδοτούν με ακριβή αποτελέσματα στην πρόβλεψη δομών RNA και λόγω απλότητας και ακρίβειας παραμένουν δημοφιλείς. Επιπρόσθετα, στην παρούσα διπλωματική χρησιμοποείται η προαναφερθείσα μέθοδο.

Εναλλακτική μέθοδο πρόβλεψης στη σύγχρονη εποχή αποτελούν και οι στοχαστικοί αλγόριθμοι, οι οποίοι παράγουν ένα σύνολο πιθανών δομών με βάση τις θερμοδυναμικές αρχές. Οι μέθοδοι Monte Carlo [18] και οι στοχαστικές μέθοδοι με γραμματικές χωρίς συμφραζόμενα είναι οι δύο πιο διευρυμένες υποκατηγορίες. Αυτές οι μέθοδοι προσφέρουν μια πιθανοτική σκοπιά στην αναδίπλωση του RNA, επιτρέποντας την εξερεύνηση για εναλλακτικές δομές. Παρότι οι στοχαστικοί αλγόριθμοι μπορούν να φωτογραφίσουν κάποιες διαφορές δομών, χρειάζονται σημαντικούς υπολογιστικούς πόρους για να παράξουν παρόμοια αποτελέσματα για μεγάλες αλληλουχίες RNA. Παράδειγμα τέτοιου αλγορίθμου αποτελεί ο αλγόριθμος Pfold [19] ο οποίος παρουσιάζει ιδιαίτερη ταχύτητα και μεγάλη ανεκτικότητα σε σφάλματα. Μια άλλη έκδοση της Pfold αποτελεί και ο αλγόριθμος PPfold [20] ο

οποίος χρησιμοπποιεί πολλαπλά νήματα παρουσιάζοντας έτσι καλύτερο χρόνο εκτέλεσης. Τέλος, σε αυτή την κατηγορία αλγορίθμων βρίσκεται και η μέθοδος RNADecoder [21] , η οποία λαμβάνει ακολουθίες RNA μαζί με την γραμματική και προβλέπει τις πιθανότητες ύπαρξης ζευγαριού βάσεων.

Επιπρόσθετα, τεχνικές μηχανικής μάθησης όπως είναι τα νευρωνικά δίκτυα και τα vector machines έχουν κερδίσει έδαφος στο πρόβλημα της πρόβλεψης της δευτεροταγούς δομής του RNA. Οι συγκεκριμένες μέθοδοι μαθαίνουν μοτίβα, από δεδομένα με τα οποία εκπαιδεύονται, και με βάση αυτά κάνουν την πρόβλεψη. Τέτοια μέθοδος είναι η DMfold [22] η οποία χρησιμοποιεί μονάδες πρόβλεψης και διόρθωσης για να καταλήξει στην σωστή έξοδο. Αρχικά η μία μονάδα προβλέπει την ακολουθία τους RNA σε κλασική αναπαράσταση dot-bracket και στη συνέχει η μονάδα διόρθωσης με την αρχή μεγιστοποίησης των ζευγαριών βάσεων καταλήγει στο σωστό αποτέλεσμα. Παραδείγματα τέτοιων αλγορίθμων αποτελούν επίσης το SPOT-RNA [23] και το DeepRNA [24]. Ο πρώτος αλγόριθμος συνδυάζει πληροφορίες της αλληλουχίες καθώς και θερμοδυναμικά χαρακτηρηστικά για να προβεί σε αποτέλεσμα ενώ ο δεύτερος χρησιμοποιεί νευρωνικά δίκτυα και βαθιά μάθηση, ενσωματώνοντας πληροφορίες από την γονιδιακή αλλουχία για την πρόβλεψη της δευτεροταγούς δομής του RNA. Τέτοιες μέθοδοι μπορούν να προσφέρουν ιδιαίτερα υψηλή ακρίβεια και να είναι αποτελεσματικές όταν εκπαιδεύονται σε πολλά και διαφορετικά μεταξύ τους σύνολα δεδομένος. Η ποιότητα τους ποικίλλει ανάλογα με την ποιότητα και την αντιπροσωπευτικότητα των δεδομένων εκπαίδευσης.

Τέλος, πρέπει να αναφερθεί ο αλγόριθμος του Knotify+, κομμάτι του οποίου αποτελεί και η παρούσα διπλωματική. Αποτελεί μια προηγμένη πλατφόρμα που εστιάζει στην επίλυση του προβλήματος της πρόβλεψης της δευτεροταγούς δομής, εστιάζοντας κυρίως στην ανίχνευση ψευδοκόμβων. Χρησιμοποιεί μια προσέγγιση βασισμένη σε γραμματικές τύπου context-free για την ακριβή ανάλυση των μοτίβων που σχετίζονται με τους ψευδοκόμβους. Μια σημαντική βελτίωση του Knotify+ είναι επίσης η ικανότητα του να ανιχνεύει και άλλες δομές όπως τα bulges και οι εσωτερικοί βρόγχοι, που περιβάλλουν τα κύρια στελέχη των ψευδοκόμβων, προσφέροντας μια πιο ολοκληρωμένη κατανόηση των δομών RNA. Με τη χερήση προηγμένων τεχνικών όπως ο παραλλαγμένος αναλυτής του Early (YAEP), το Knotify+ αποτελεί ένα ισχυρό εργαλείο για την έρευνα και την εφαρμογή στην βιοπληροφορική.

3.2 Θεωρητικές Έννοιες

Αυτό το κεφάλαιο παρέχει μια ολοκληρωμένη εξέταση βασικών θεωρητικών εννοιών και δομών οι οποίες είναι θεμελιώδεις για την συντακτική ανάλυση και

αναγνώριση προτύπων στην επιστήμη των υπολογιστών. Αρχικά θα παρουσιαστεί η έννοια της συντακτικής αναγνώρισης προτύπων, θέτοντας έτσι το υπόβαθρο για μια διερεύνηση των γραμματικών χωρίς συμφραζόμενα, οι οποίες είναι ζωτικής σημασίας για τον καθορισμό των κανόνων στις γλώσσες προγραμματισμού. Στη συνέχεια, θα παρουσιαστούν τα αφηρημένα συντακτικά δέντρα και οι parsers που αξιοποιούν τις παραπάνω γραμματικές για την ανάλυση και την επικύρωση δομημένων inputs. Τέλος, γίνεται μία ανάλυση του αλγορίθμου CYK καθώς και του αλγορίθμου Early, ώστε να καταλήξει στον αλγόριθμο YAEP, ο οποίος χρησιμοποιείται και για την δική μας επίλυση του προβλήματος.

3.2.1 Συντακτική αναγνώριση προτύπων

Η συντακτική αναγνώριση προτύπων είναι μια σειρά μεθόδων με τις οποίες αποδίδεται κάποια τιμή ή κάποιο διακριτό στοιχείο σε εισαγόμενα δεδομένα [25]. Αυτό συνήθως επιτυγχάνεται με την ανάπτυξη κάποιου αλγορίθμου με την οποία τα δεδομένα ταξινομούνται σε ομάδες και υποκατηγορίες με βάσει τα κριτήρια τα οποία θα τεθούν. Είναι δυνατόν να χρησιμοποιηθεί αντί για κάποια στατιστική ταξινόμηση αντικειμένων εάν και εφόσον υπάρχει ξεκάθαρο μοτίβο στα χαρακτηριστικά των υπό επεξεργασία αντικειμένων. Στη συντακτική αναγνώριση προτύπων, η γλώσσα αποτελείται από μια σειρά συντακτικών κανόνων που κατασκευάζουν μια συμβολοσειρά. Ένα τέτοιο σύνολο μπορεί να χαρακτηριστεί ως υποσύνολο μιας γραμματικής της γλώσσας, το οποία με τη σειρά του παράγει συγκεκριμένα υποσύνολα. Υπάρχουν τέσσερις κατηγορίες τέτοιων γραμματικών και στο υπάρχον πρόβλημα θα χρησιμοποιηθεί η γραμματική χωρίς συμφραζόμενα.

3.2.2 Γραμματικές χωρίς συμφραζόμενα

Στις γραμματικές χωρίς συμφραζόμενα(Context free grammars) το αριστερό μέρος κάθε κανόνα παραγωγής περιέχει μόνο ένα μη τερματικό σύμβολο, είναι δηλαδή της μορφής A->a [26]. Το a μπορεί να είναι μια συμβολοσειρά και από τερματικά και από μη τερματικά σύμβολα. Μια τέτοια γραμματική G ορίζεται από την πλειάδα G=(V,Σ,R,S). Το V είναι ένα πεπερασμένο σύνολο όπου κάθε στοιχείο που ανήκει σε αυτό ονομάζεται μη τερματικός χαρακτήρας ή μεταβλητή. Κάθε τέτοια μεταβλητή ορίζει μια υπογλώσσα της γλώσσας η οποία παράγεται από την γραμματική G. Το Σ είναι ένα σύνολο τερματικών συμβόλων, τα οποία αποτελούν ουσιαστικά το περιεχόμενα της παραχθείσας τελικής πρότασης. Είναι τελικά το “αλφάβητο” της οριζόμενης από την G γλώσσας. Το R είναι μια πεπερασμένη σχέση στο σύνολο V x (V U Σ)* ,όπου ο αστερίσκος αναπαριστά το άστρο Kleene, και είναι το σύνολο των κανόνων παραγωγής της γραμματικής. Τέλος, το S είναι το αρχικό σύμβολο της γραμματικής και εμπεριέχεται στο σύνολο των μη τερματικών χαρακτήρων. Τυπικά, αναφέρεται ότι οι κεφαλαίοι λατινικοί χαρακτήρες εκφράζουν τα μη τερματικά σύμβολα και οι αντίστοιχοι μικροί τα τερματικά. Χρησιμοποιούνται

επίσης ελληνικοί χαρακτήρες για την αναπαράσταση και των δύο κατηγοριών συμβόλων.

3.2.3 Αφηρημένα συντακτικά δέντρα

Τα αφηρημένα συντακτικά δέντρα (ASTs) είναι δομές δεδομένων που χρησιμοποιούνται για την αναπαράσταση μια δομής ενός προγράμματος ή κάποιου κώδικα [27]. Αποτελούν μια αναπαράσταση στη μορφή δέντρου της δομής, όπου οι κόμβοι του δέντρου αντιπροσωπεύουν δομές όπως δηλώσεις ή εκδράσεις και οι σύνδεσμοι μεταξύ των κόμβων δείχνουν την ιεραρχική σχέση μεταξύ αυτών. Είναι θεμελιώδη για την υλοποίηση μεταγλωτιστών και ερμηνευτών, καθώς διευκολύνουν την ανάλυση και τη μεταφραστική διαδικασία από τον αρχικό κώδικα σε εκτελέσιμη μορφή. Είναι μια ιδανική μέθοδος για τη δημιουργία περιβαλλόντων που υποστηρίζουν οπτικοποιήσεις του κώδικα, παρέχοντας στον αναγνώστη μια ολοκληρωμένη άποψη για την δομή του. Τέλος, αξίζει να αναφερθεί ότι βοηθούν στην αναγνώριση προβλημάτων απόδοσης και ασφάλειας του κώδικα και προσφέρονται ως μια αξιόπιστη μέθοδος για την αλλαγή του κώδικα χωρίς αλλοίωση της λειτουργικότητας του.

3.2.4 CFG Parsers

Οι αναλυτές (parsers) των γραμματικών χωρίς συμφραζόμενα είναι εργαλεία λογισμικού τα οποία αναλύουν την συντακτική δομή του εισαγόμενου κειμένου με βάση την δοθείσα γραμματική. Οι συγκεκριμένοι parsers αναγνωρίζουν και κατασκευάζουν αναλυτικά δέντρα από ακολουθίες συμβόλων και είναι ζωτικής σημασίας για τη μεταγλώττιση διαδικασιών, όπου ερμηνεύουν τον πηγαίο κώδικα για να κατανοήσουν τις δομές του προγράμματος. Οι δύο από τους σημαντικότερους αλγορίθμους που πραγματοποιούν την προαναφερθείσα διαδικασία είναι ο αλγόριθμος CYK (Cocke, Younger and Kasami) και ο αλγόριθμος του Early τους οι οποίοι παρουσιάζονται παρακάτω.

3.2.5 Αλγόριθμος CYK

Ο αλγόριθμος CYK είναι ένας αλγόριθμος parser για γραμματικές χωρίς συμφραζόμενα ο οποίος δημοσιεύτηκε το 1961 Wikipedia contributors. (n.d.). *Abstract syntax tree*. Wikipedia. https://en.wikipedia.org/wiki/Abstract_syntax_tree

[28]. Χρησιμοποιεί την μέθοδο του δυναμικού προγραμματισμού για να αποφανθεί αν μία συμβολοσειρά μπορεί να παραχθεί από μία δοσμένη CFG. Η γραμματική πρέπει να δίνεται στον parser σε μορφή CNF (Chomsky Normal Form) για να μπορέσει να δουλέψει. Η μετατροπή οποιασδήποτε CFG στην παραπάνω μορφή

είναι κάτι εφικτό οπότε αυτό δεν αποτελεί πρόβλημα. Η απόδοση του συγκεκριμένου αλγορίθμου είναι επίσης πάρα πολύ ικανοποιητική καθώς παρουσιάζει πολυπλοκότητα χειρότερης περίπτωσης $O(n^3 * G)$. Το n αναπαριστά το μήκος της συμβολοσειράς και το G το μέγεθος της γραμματικής. Στη συνέχεια παρατίθεται ο ψευδοκώδικας του αλγορίθμου CYK:

```
let the input be a string I consisting of n characters: a1 ... an.
let the grammar contain r nonterminal symbols R1 ... Rr, with start symbol R1.
let P[n,n,r] be an array of booleans. Initialize all elements of P to false.
let back[n,n,r] be an array of lists of backpointing triples.
Initialize all elements of back to the empty list.
for each s = 1 to n
   for each unit production Rv → as
      set P[1,s,v] = true
for each l = 2 to n -- Length of span
   for each s = 1 to n-l+1 -- Start of span
      for each p = 1 to l-1 -- Partition of span
         for each production Ra  → Rb Rc
            if P[p,s,b] and P[l-p,s+p,c] then
               set P[l,s,a] = true,
               append <p,b,c> to back[l,s,a]
if P[n,1,1] is true then
   I is member of language
   return back -- by retracing the steps through back, one can easily construct
   all possible parse trees of the string
else

   return "not a member of language
```

Ο αλγόριθμος αναλύει κάθε δυνατό υποσύνολο της εισερχόμενης συμβολοσειράς και καθορίζει το στοιχείο P[l,s,v] ως αληθές εάν η υποσειρά με μήκος l που αρχίζει από την θέση s μπορεί να γεννηθεί από το μη τερματικό σύμβολο Rv.

Η ανάλυση εκκινεί από υποσειρές με μήκος 1 και συνεχίζει με τα μεγαλύτερα μήκη. Εάν το μήκος είναι 2 ή μεγαλύτερο εξετάζει κάθε δυνατή διαίρεση της αλληλουχίας σε δύο τμήματα και ελέγχει αν υπάρχει κάποιος κανόνας της μορφής A->BC όπου το B αναπαριστά το πρώτο μέρος και το C το δεύτερο. Όταν και εάν αυτό το κριτήριο ικανοποιηθεί τότε η διαδικασία ολοκληρώνεται, καταγράφεται ότι το A ταιριάζει με την συμβολοσειρά και συμπερασματικά η συμβολοσειρά μπορεί να παραχθεί από το αρχικό σύμβολο της γραμματικής.

3.2.6 Αλγόριθμος του Early

Ο αλγόριθμος του Early είναι ένας αναλυτής που χρησιμοποιεί δυναμικό προγραμματισμό με σκοπό την συντακτική αναγνώριση [29]. Αναλύει συμβολοσειρές οι οποίες ανήκουν σε μια γραμματική χωρίς συμφραζόμενα με μεγάλη απόδοση, αλλά είναι επίσης αποτελεσματικός και για την ανάλυση δομικά πιο πολύπλοκων γλωσσών λόγω της δυνατότητας του να χειρίζεται ασαφείς γραμματικές. Εισήχθη για πρώτη φορά στην διατριβή του εφευρέτη του, Jay Early, το 1968. Η χρονική πολυπλοκότητα του αλγορίθμου είναι συνήθως $O(n^3)$ για τη γενική περίπτωση, αλλά βελτιώνεται σε $O(n^2)$ για διφορούμενες γραμματικές και μπορεί να φτάσει το O(n) για γραμματικές που έχουν ντετερμινιστικές ιδιότητες. Ο ψευδοκώδικας του αλγορίθμου Early παρουσιάζεται παρακάτω:

```
DECLARE ARRAY S;
function INIT(words)
    S ← CREATE_ARRAY(LENGTH(words) + 1)
    for k ← from 0 to LENGTH(words) do
        S[k] ← EMPTY_ORDERED_SET
function EARLEY_PARSE(words, grammar)
    INIT(words)
    ADD_TO_SET((γ → •S, 0), S[0])
    for k ← from 0 to LENGTH(words) do
        for each state in S[k] do  // S[k] can expand during this loop
            if not FINISHED(state) then
                if NEXT_ELEMENT_OF(state) is a nonterminal then
                    PREDICTOR(state, k, grammar)        // non_terminal
                else do
                    SCANNER(state, k, words)            // terminal
            else do
                COMPLETER(state, k)
        end
```

```
    end
    return chart
procedure PREDICTOR((A → α•Bβ, j), k, grammar)
    for each (B → γ) in GRAMMAR_RULES_FOR(B, grammar) do
        ADD_TO_SET((B → •γ, k), S[k])
    end
procedure SCANNER((A → α•aβ, j), k, words)
    if j < LENGTH(words) and a ⊂ PARTS_OF_SPEECH(words[k])
then
        ADD_TO_SET((A → αa•β, j), S[k+1])
    end
procedure COMPLETER((B → γ•, x), k)
    for each (A → α•Bβ, j) in S[x] do
        ADD_TO_SET((A → αB•β, j), S[k])
    end
```

Στην παραπάνω περιγραφή του ψευδοκώδικα το σύμβολα α,β και γ αντιπροσωπεύουν τερματικά και μη τερματικά σύμβολα, τα X και Y μη τερματικά σύμβολα ενώ το a ένα τερματικό σύμβολο. Αποτελεί εναν top-down αλγόριθμο δυναμικού προγραμματισμού και για αυτό για έναν κανόνα X->αβ, η δήλωση X->α•β υποδεικνύει ότι το σύμβολο α έχει ήδη αναλυθεί ενώ το β θα αναλυθεί στην συνέχεια. Κατα τη διάρκεια της εκτέλεσης του αλγορίθμου για κάθε χαρακτήρα της εισερχόμενης συμβολοσειράς δημιουργείται ένα σύνολο πλειάδων της μορφής (X->α•β, i) όπου:

1) Οι κανόνες παραγωγής είναι X->αβ
2) Η θέση στην οποία βρισκόμαστε στον κανόνα είναι η τελεία
3) Το i υποδηλώνει τη θέση έναρξης του ταιριάσματος του κανόνα

Το σύνολο της κατάστασης στη θέση της εισόδου k συμβολίζεται με S(k). Ο parser ξεκινά από το S(0) και εκτελεί επαναληπτικά τις παρακάτω τρεις λειτουργίες:

1) Prediction: Για κάθε κατάσταση στο S(k) που βρίσκεται σε μορφή (X->α•Yβ, j), προσθέτει στο σύνολο όλους τους κανόνες που έχουν το Y ως αριστερό στοιχείο.
2) Scanning: Αν το επόμενο σύμβολο είναι τερματικό, για κάθε κανόνα σε μορφή (X->α•αβ, j) μετακινεί την τελεία κατά μία θέση δεξιά και προσθέτει τον παραγόμενο κανόνα στο S(k+1).

3) Completion: Για κάθε κατάσταση στο S(k) σε μορφή (Y->γ•, j), εντοπίζει όλες τις καταστάσεις (X->α•Yβ, i) στο S(j) και μετακινεί την τελεία κατά μία θέση δεξιά, προσθέτοντας τον κανόνα στο S(k).

Ο αλγόριθμος Early επαναλαμβάνει αυτές τις λειτουργίες μέχρι να μην υπάρχει περαιτέρω προσθήκη δυνατή στα σύνολα καταστάσεων. Η συμβολοσειρά είναι αποδεκτή από τον αλγόριθμο εάν υπάρχει κατάσταση (X->γ•, 0) στο S(n), όπου (X->γ) αποτελεί τον πρώτο κανόνα της γραμματικής και n το μήκος της συμβολοσειράς. Αν δεν υπάρχει τέτοια κατάσταση, η συμβολοσειρά απορρίπτεται. Πρέπει να σημειωθεί ότι ο αλγόριθμος δεν προσθέτει διπλότυπα στα σύνολα των καταστάσεων.

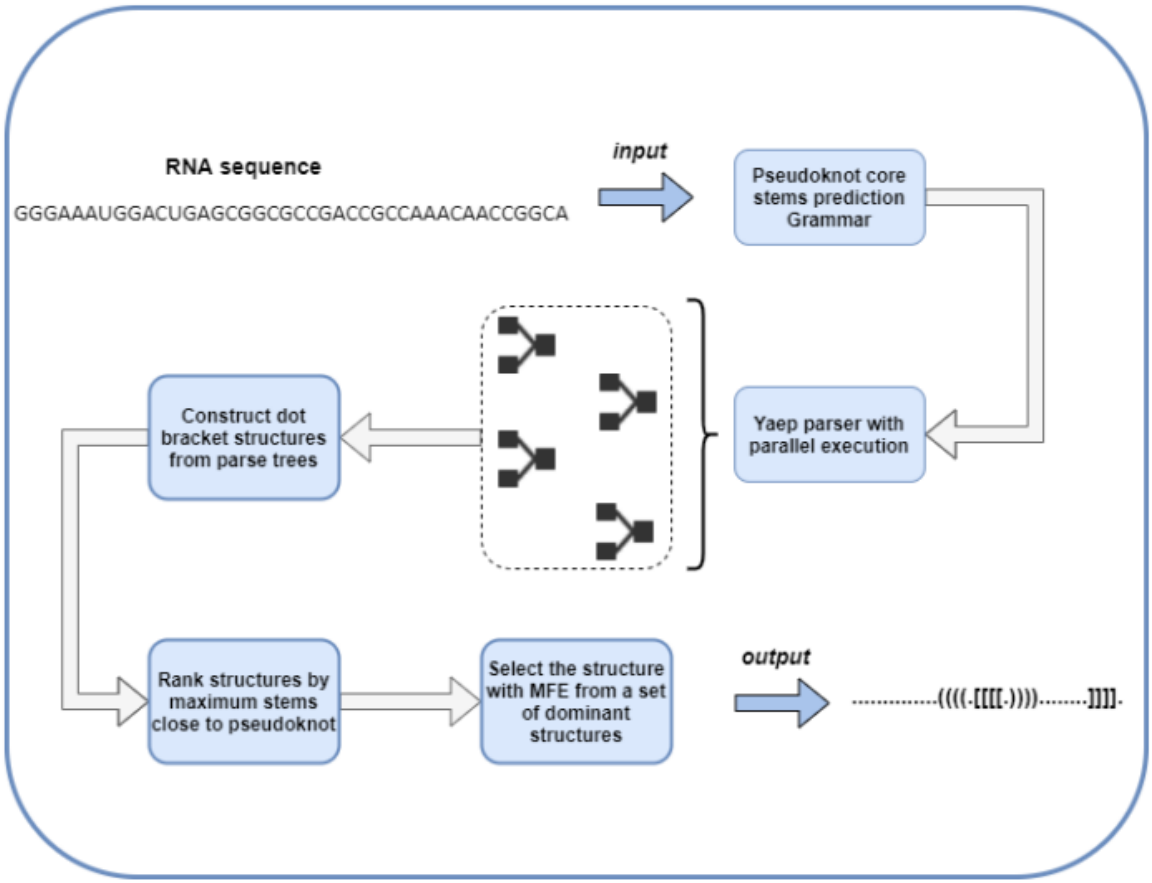
3.2.7 YAEP

Ο YAEP [30] ή αλλιώς Yet Another Early Parser, αποτελεί μια από τις ταχύτερες και πιο αποδοτικές εκδοχές του αλγορίθμου Early. Κύρια προτερήματα του yaep είναι η υψηλή ταχύτητα και οι χαμηλές απαιτήσεις μνήμης, κάτι που τον καθιστά ιδανικό για χρήση σε μεταγλωττιστές και επεξεργαστές γλωσσών. Δεν περιορίζεται μόνο στην ανάλυση απλών συντακτικών δομών αλλά μπορεί επίσης να διαχειριστεί εισόδους από διφορούμενες γραμματικές, παράγοντας πολλαπλά αφηρημένα συντακτικά δέντρα. Σε αυτόν τον αλγόριθμο θα βασιστεί η συγκεκριμένη διπλωματική για την επίλυση του προβλήματος του εντοπισμού του ψευδοκόμβου K.

4. Επίλυση του προβλήματος για τον ψευδοκόμβο τύπου K

4.1 Εισαγωγή στην μεθοδολογία επίλυσης

Η προσέγγιση μας για την αντιμετώπιση του ζητήματος σχετικά με τον ψευδοκόμβο τύπου K αποτελείται από τρία κύρια στάδια. Κατά το πρώτο στάδιο, διενεργείται η ανάλυση της ακολουθίας RNA, όπου και αναγνωρίζονται όλες οι εμφανίσεις του ψευδοκόμβου K. Για την εκτέλεση αυτής της ανάλυσης, έχουν αναπτυχθεί δύο διακριτοί αλγορίθμοι, οι οποίοι διαχειρίζονται την ακολουθία RNA ως μια συμβολοσειρά με τους χαρακτήρες “A”,”G”,”U” και ”C” οι οποίοι αντιστοιχίζονται στην αδενίνη, την γουανίνη, την ουρακίλη και την κυτοσίνη. Ο πρώτος αλγόριθμος που εφαρμόζεται βασίζεται στην μεθοδολογία συντακτικής αναγνώρισης προτύπων, η οποία αναφέρθηκε στο προηγούμενο κεφάλαιο. Αρχικά ορίζεται μια γραμματική χωρίς συμφραζόμενα η οποία περιλαμβάνει τους κανόνες για την παραγωγή ενός ψευδοκόμβου τύπου K. Χρησιμοποιώντας αυτή την γραμματική και έναν συντακτικό αναλυτή(parser), παράγονται όλα τα δέντρα που αναπαριστούν ψευδοκόμβους τύπου K στην δοθείσα RNA ακολουθία. Αυτά τα δέντρα στη συνέχεια εξετάζονται μέσω ειδικά σχεδιασμένης συνάρτησης για τον καθορισμό κρίσιμων μεγεθών για κάθε ψευδοκόμβο, βοηθώντας να καθοριστούν οι θέσεις των βάσεων για τους δεσμούς των ψευδοκόμβων. Ο δεύτερος αλγόριθμος που εφαρμόζεται είναι ένας άπληστος αλγόριθμος δυναμικού προγραμματισμού που καταλήγει στον υπολογιστικό προσδιορισμό των ίδιων μεγεθών, βασιζόμενες σε διαδοχικές προσπελάσεις της ακολουθίας. Ειδικά για τους ψευδοκόμβους τύπου H, χρησιμοποιήθηκαν και οι δύο αλγόριθμοι, ενώ για τις δομές των hairpins χρησιμοποιήθηκε μόνο η συντακτική αναγνώριση προτύπων. Στη συνέχεια. υπολογίζονται οι δεσμοί που δημιουργούνται μεταξύ βάσεων γύρω από τους κύριους δεσμούς στην ακολουθία. Προχωρώντας, η αναπαράσταση της ακολουθίας RNA σε μορφή dot-bracket επιτρέπει τη διάκριση των βάσεων που σχηματίζουν δεσμούς από τις υπόλοιπες βάσεις. Αυτό επιτυγχάνεται με τη χρήση αγκύλων για τις ζευγαρωμένες βάσεις και τελείες για αυτές που δεν έχουν ζεύγος. Το τελικό στάδιο της διαδικασίας στοχεύει στον εντοπισμό της βέλτιστης δομής με κριτήριο την ελαχιστοποίηση της ελεύθερης ενέργειας αναδίπλωσης και τη μεγιστοποίηση του αριθμού των ζευγαριών μεταξύ βάσεων. Έτσι εν τέλει καθορίζεται η τελική δομή που θα παρουσιαστεί ως έξοδος της εφαρμογής. Παρακάτω φαίνονται σχηματικά όλα τα βήματα της διαδικασίας όπως παρουσιάστηκαν για πρώτη φορά στην επίλυση του προβλήματος για ψευδοκόμβους τύπου H:



Σχήμα 4.1: *Μεθοδολογία επίλυσης του προβλήματος [26]*

4.2 Συντακτική αναγνώριση προτύπων

4.2.1 Η δημιουργία της γραμματικής για την αναγνώριση των ψευδοκόμβων

Το πρώτο βήμα της διαδικασίας συντακτικής αναγνώρισης προτύπων ήταν να δημιουργηθεί την απαραίτητη γραμματική χωρίς συμφραζόμενα. Για την δημιουργία αυτής που χρησιμοποιείται στην αναγνώριση της συγκεκριμένης δομής του RNA βασιστήκαμε στα διακριτά χαρακτηριστικά του ζητούμενο ψευδοκόμβου τύπου K. Σύμφωνα με το παρακάτω διάγραμμα, ο συγκεκριμένος ψευδοκόμβος αποτελείται από έξι βασικές βάσεις που σχηματίζουν τρία ζευγάρια. Το κάθε ένα από τα τρία ζευγάρια σχηματίζεται από δύο βάσεις από αδενίνη, γουανίνη, ουρακίλη και κυτοσίνη. Συγκεκριμένα για τον τύπο K, η πρώτη βάση συζευγνύεται με την τρίτη, η δεύτερη με την πέμπτη και η τέταρτη με την έκτη. Σε αυτό το σημείο να αναφερθεί ότι ο συνολικός αριθμός κανόνων της γραμματικής για την αναγνώριση της ζητούμενης δομής ειναι 6^3=216 . Αυτό συμβαίνει διότι στον τύπο K υπάρχουν τρία κύρια

ζευγάρια και μεταξύ των τεσσάρων νουκλεοτιδίων μπορούν να σχηματιστούν 6 συγκεκριμένοι τύποι ζευγαριών:

1. αδενίνη-ουρακίλη
2. ουρακίλη-αδενίνη
3. κυτοσίνη-γουανίνη
4. γουανίνη-κυτοσίνη
5. γουανίνη-ουρακίλη
6. ουρακίλη-γουανίνη

Στη συνέχεια ξεκίνηση η παραγωγή της γραμματικής όπως παρατίθεται παρακάτω. Αρχικά συμπεριλαμβάνονται όλοι οι κανόνες για τον ψευδοκόμβο τύπου K. Ακολουθούν μερικοί από τους κανόνες:

```
S :  'a' L 'a' L 'u' D 'a' L 'u' L 'u' # R01 (1 3 5 7 9)
  | 'a' L 'a' L 'u' D 'u' L 'u' L 'a' # R02 (1 3 5 7 9)
  | 'a' L 'a' L 'u' D 'c' L 'u' L 'g' # R03 (1 3 5 7 9)
  | 'a' L 'a' L 'u' D 'g' L 'u' L 'c' # R04 (1 3 5 7 9)
…
  | 'g' L 'g' L 'c' D 'a' L 'c' L 'u' # R61 (1 3 5 7 9)
  | 'g' L 'g' L 'c' D 'u' L 'c' L 'a' # R62 (1 3 5 7 9)
  | 'g' L 'g' L 'c' D 'c' L 'c' L 'g' # R63 (1 3 5 7 9)
  | 'g' L 'g' L 'c' D 'g' L 'c' L 'c' # R64 (1 3 5 7 9)
```

Οι κανόνες της γραμματικής όπως παρατηρείται είναι της μορφής S->aLaLuDaLuLu, όπου οι μικροί λατινικοί χαρακτήρες αντιστοιχούν στις βάσεις που δημιουργούν τα κύρια ζευγάρια της δομής. Εν προκειμένω η αδενίνη εκφράζεται με τον χαρακτήρα “a” στην πρώτη θέση του κανόνα δημιουργεί ζευγάρι με την ουρακίλη, η οποία αναπαρίσταται με τον χαρακτήρα “u” στην τρίτη θέση του κανόνα. Αντίστοιχα συμβαίνει με όλα τα πιθανά ζευγάρια και σημειώνεται επίσης ότι η η γουανίνη αναπαρίσταται από το “g” και η κυτοσίνη από το “c”. Οι κεφαλαίοι χαρακτήρες στο δεξί μέρος του κανόνα εκφράζουν τα μήκη των ακολουθιών που υπάρχουν ανάμεσα στις κύριες βάσεις. Ο χαρακτήρας S αντιπροσωπεύει το σύμβολο εκκίνησης της γραμματικής, ενώ οι χαρακτήρες L εκφράζουν τα κενά ανάμεσα στις κύριες βάσεις, εκτός από τα κενά μεταξύ της τρίτης και της τέταρτης βάσης. Όπως φαίνεται στο σχήμα του ψευδοκόμβου τύπου K, κάθε τέτοιο κενό μεταξύ των βάσεων πρέπει να έχει μήκος τουλάχιστον ένα, οδηγώντας στον ακόλουθο κανόνα για το χαρακτήρα L. Με βάση το παραπάνω προκύπτει και ο κανόνας για τον χαρακτήρα L που συμπεριλαμβάνεται επίσης στην γραμματική:

```
L : 'a' L # L1 (1)
  | 'u' L # L2 (1)
  | 'c' L # L3 (1)
  | 'g' L # L4 (1)
```

```
 | 'a'   # 0
 | 'u'   # 0
 | 'c'   # 0
 | 'g'   # 0
 ;
```

Το μήκος της ακολουθίας μεταξύ των δύο ενδιάμεσων βάσεων της δομής, το οποίο αναπαρίσταται από το χαρακτήρα D στη γραμματική, υπόκειται σε ένα ανώτατο όριο που καθορίζεται από τη μεταβλητή max_dd_size. Αυτή η μεταβλητή μπορεί να οριστεί από τον χρήστη κατά την εκτέλεση του προγράμματος, ενώ η προεπιλεγμένη της τιμή, εάν δεν οριστεί από τον χρήστη, είναι 2. Ο κανόνας για το σύμβολο D στη γραμματική περιγράφεται παρακάτω.

```
D : {% for x in range(max_dd_size) %}D{{ x }} {% endfor %} # M1
({% for x in range(max_dd_size) %}{{ x }} {% endfor %})
{% for x in range(max_dd_size) %}
D{{ x }} : E # N{{ x }} (0)
{% endfor %}
```

Το σύμβολο E, που αναφέρεται στον παραπάνω κανόνα, αντιπροσωπεύει τη βάση που μπορεί να υπάρχει σε κάθε μία από τις θέσεις που καθορίζονται από το ανώτατο όριο του μήκους της ακολουθίας (τη μεταβλητή max_dd_size) στον κανόνα για το χαρακτήρα D. Επειδή δεν υπάρχει περιορισμός σχετικά με το ποια βάση μπορεί να εμφανιστεί σε κάθε θέση, ο κανόνας για το σύμβολο E διαμορφώνεται όπως φαίνεται παρακάτω:

```
E : 'a' # 0
  | 'u' # 0
  | 'c' # 0
  | 'g' # 0
  | ;
```

Τέλος, έχει αναπτυχθεί μια συνάρτηση που παράγει την τελική γραμματική, την οποία χρησιμοποιεί ο συντακτικός αναλυτής για να δημιουργήσει όλα τα πιθανά δέντρα που περιέχουν έναν ψευδοκόμβο τύπου L. Αυτή η συνάρτηση δέχεται δύο παραμέτρους που καθορίζουν τη μορφή της γραμματικής: την max_dd_size, η οποία, όπως αναφέρθηκε προηγουμένως, θέτει το ανώτατο όριο για το μήκος της ακολουθίας μεταξύ των δύο ενδιάμεσων κύριων βάσεων του ψευδοκόμβου και την δυαδική μεταβλητή allow_ug, η οποία, ανάλογα με την τιμή της (True ή False), καθορίζει αν θα συμπεριληφθούν οι ψευδοκόμβοι με δεσμούς γουανίνης και ουρακίλης (u-g) στην τελική γραμματική. Ακολουθεί η συνάρτηση παραγωγής της γραμματικής:

```
def generate_grammar(allow_ug: bool, max_dd_size: int) -> str:
    """
    Generate grammar for pseudoknot detection, based on parameters.
    """
    return (
        jinja2.Environment()
        .from_string(TEMPLATE)
        .render(
            allow_ug=allow_ug,
            max_dd_size=max_dd_size,
        )
    )
```

4.2.2 Ανάλυση της γραμματικής και παραγωγή του συντακτικού δέντρου

Η διαδικασία αυτή υλοποιήθηκε μέσω της συνάρτησης parse[25], η οποία λαμβάνει ως είσοδο τη γραμματική και επιστρέφει έναν κωδικό σφάλματος. Ο κωδικός αυτός υποδεικνύει αν υπήρξε κάποιο πρόβλημα κατά την εκτέλεση. Αν ο κωδικός είναι μηδενικός, αυτό σημαίνει ότι η εκτέλεση ήταν επιτυχής. Η συνάρτηση επιστρέφει τη ρίζα του δέντρου που περιέχει όλα τα πιθανά υποδέντρα που αναπαριστούν τους ψευδοκόμβους τύπου K στην RNA ακολουθία. Επιπλέον, η συνάρτηση δέχεται ένα σημείο εισόδου, το οποίο καθορίζει την αρχή της υποακολουθίας της συμβολοσειράς που θα υποβληθεί σε συντακτική ανάλυση. Ακολουθεί ο κώδικας:

```
struct yaep_tree_node *parse (const char *description) {
struct grammar *g;
struct yaep_tree_node *root;
int ambiguous_p;
if ((g = yaep_create_grammar ())==NULL) {
fprintf (stderr , ” yaep_create_grammar : No memory\n ”) ;
exit (1) ;
}
yaep _set _debug _level (g , 0);
yaep _set _one _parse _ flag (g , 0);
yaep _set _ cost _ flag (g , 0);
yaep _set _lookahead _level (g , 2) ;
if ( yaep_parse_grammar (g , TRUE , description) != 0) {
fprintf (stderr , ”%s\n ” , yaep_error_message (g)) ;
```

```
exit (1) ;
}
int parsed = yaep_parse (g , read_token_ func , syntax _ error_func ,
parse_alloc_func , NULL , &root , &ambiguous_p);
if (parsed) {
printf (” this should not be accepted \n ”) ;
fprintf (stderr , ” yaep_parse : %s\n ” , yaep_error_message (g));
}
return root ;
}
```

Κατά την εκτέλεση, η πρώτη συνάρτηση που καλείται είναι η yaep_create_grammar, η οποία δημιουργεί έναν αναλυτή με απροσδιόριστη γραμματική. Εάν δεν υπάρχει αρκετή μνήμη για την ανάλυση, η συνάρτηση επιστρέφει NULL. Η συνάρτηση yaep_set_debug_level δέχεται έναν ακέραιο που καθορίζει τον βαθμό εμφάνισης πληροφοριών στην έξοδο σχετικά με το δέντρο ανάλυσης, τη γραμματική και άλλα συναφή δεδομένα. Αν το όρισμα είναι μηδέν, τότε δεν εμφανίζονται καθόλου πληροφορίες στην έξοδο. Η συνάρτηση yaep_set_one_parse_flag καθορίζει αν θα δημιουργηθεί ένα μόνο δέντρο μετάφρασης της γραμματικής ή πολλαπλά συντακτικά δέντρα για αμφίδρομες γραμματικές. Στην περίπτωση μη αμφίδρομης γραμματικής, η τιμή της παραμέτρου δεν επηρεάζει το αποτέλεσμα. Η συνάρτηση yaep_set_cost_flag δέχεται μια ακέραια παράμετρο. Εάν η παράμετρος αυτή είναι μηδέν, η συνάρτηση παράγει μόνο ένα δέντρο μετάφρασης με το χαμηλότερο δυνατό κόστος, εφόσον η γραμματική είναι αμφίδρομη. Διαφορετικά, παράγονται πολλαπλά δέντρα. Για μη αμφίδρομες γραμματικές η τιμή της παραμέτρου δεν επηρεάζει το αποτέλεσμα. Επιπλέον, η συνάρτηση yaep_set_lookahead_level καθορίζει το επίπεδο χρήσης των lookaheads κατά την ανάλυση της γραμματικής μέσω μιας ακέραιας παραμέτρου. Εάν η τιμή της παραμέτρου είναι 0, τότε δεν χρησιμοποιούνται καθόλου lookaheads. Με τιμή 1, τα lookaheads χρησιμοποιούνται ανεξάρτητα από το σημείο εισόδου που ορίζεται στην αρχική συνάρτηση. Τέλος, η τιμή 2 υποδεικνύει ότι τα lookaheads θα χρησιμοποιηθούν δυναμικά, λαμβάνοντας υπόψη το σημείο εισόδου. Αυτές οι ρυθμίσεις επιτρέπουν την προσαρμογή της ανάλυσης στις συγκεκριμένες ανάγκες και περιορισμούς της εφαρμογής, βελτιστοποιώντας τόσο την απόδοση όσο και την ακρίβεια της διαδικασίας. Η συνάρτηση yaep_parse_grammar συντονίζει τον αναλυτή με τη γραμματική που δημιουργήθηκε, ενώ η συνάρτηση yaep_parse πραγματοποιεί την ανάλυση της γραμματικής και την παραγωγή των δέντρων, επιστρέφοντας παράλληλα έναν κωδικό σφάλματος. Το πρώτο όρισμα που λαμβάνει η yaep_parse είναι η γραμματική, η οποία αποθηκεύεται στη μεταβλητή g. Το δεύτερο όρισμα, η συνάρτηση read_token_func, παρέχει στην κύρια συνάρτηση το σημείο εισόδου της συμβολοσειράς μαζί με ένα χαρακτηριστικό του. Εάν η συνάρτηση read_token_func επιστρέψει αρνητική τιμή, αυτό σημαίνει ότι έχουν διαβαστεί όλα τα σημεία εισόδου, οπότε η ανάλυση ολοκληρώνεται. Αυτή η διαδικασία διασφαλίζει ότι η γραμματική

έχει αναλυθεί πλήρως και ότι όλα τα συντακτικά δέντρα που αντιστοιχούν στην είσοδο έχουν παραχθεί σωστά. Το τρίτο όρισμα, η συνάρτηση syntax_error_func, καλείται όταν αναγνωρίζεται κάποιο συντακτικό λάθος κατά τη διαδικασία ανάλυσης. Η συνάρτηση parse_alloc_func είναι υπεύθυνη για τη δέσμευση της απαραίτητης μνήμης για την ανάλυση και δέχεται έναν ακέραιο που καθορίζει το μέγεθος της μνήμης που θα δεσμευτεί. Το επόμενο όρισμα είναι η συνάρτηση parse_free, η οποία χρησιμοποιείται για την απελευθέρωση της μνήμης που δεσμεύτηκε από την parse_alloc_func. Πρέπει να σημειωθεί ότι η συγκεκριμένη συνάρτηση αρχικοποιείται με NULL. Επιπλέον, η μεταβλητή ambiguous_p δείχνει αν η γραμματική εισόδου είναι ambiguous ή όχι. Τέλος, η μεταβλητή root είναι αυτή που επιστρέφεται από την αρχική συνάρτηση και αντιπροσωπεύει τη ρίζα του συντακτικού δέντρου που παράγεται από τη διαδικασία ανάλυσης. Αυτού του τύπου η ρύθμιση διασφαλίζει ότι όλα τα απαραίτητα στοιχεία για την ανάλυση και τη διαχείριση της μνήμης είναι σωστά καθορισμένα, επιτρέποντας την ομαλή εκτέλεση της διαδικασίας ανάλυσης και παραγωγής του συντακτικού δέντρου.

4.2.3 Διάσχιση των δέντρων για την αναγνώριση των ζητούμενων ψευδοκόμβων

Στη συνέχεια ακολούθησε η διαδικασία της διάσχισης του δέντρου που δημιουργήθηκε στο προηγούμενο βήμα, ώστε να εντοπιστούν όλοι οι ζητούμενοι ψευδοκόμβοι. Για την επίτευξη αυτού του στόχου, αρχικά δημιουργήθηκε μια δομή (struct) η οποία αναπαριστά μια συνδεδεμένη λίστα. Αυτή η λίστα περιλαμβάνει όλους τους ψευδοκόμβους που προκύπτουν από τη διάσχιση του δέντρου. Κάθε ψευδοκόμβος στη λίστα περιέχει κάποια βασικά χαρακτηριστικά και ειδικά για τον τύπο K, η δομή περιέχει τις εξής τέσσερις μεταβλητές:

- Τη μεταβλητή left_left_loop_size, η οποία αναπαριστά το μήκος της ακολουθίας RNA ανάμεσα στη πρώτη και στη δεύτερη κύρια βάση
- Τη μεταβλητή left_right_loop_size, η οποία αναπαριστά το μήκος της ακολουθίας RNA ανάμεσα στη δεύτερη και στην τρίτη κύρια βάση
- Τη μεταβλητή dd_size, η οποία αναπαριστά το μήκος της ακολουθίας ανάμεσα στις δύο ενδιάμεσες βάσεις (τρίτη και τέταρτη)
- Τη μεταβλητή right_left_loop, η οποία αναπαριστά το μήκος της ακολουθίας ανάμεσα στην τέταρτη και την πέμπτη κύρια βάση.
- Τη μεταβλητή next, η οποία είναι και αυτή ίδιου τύπου με την αρχική δομή και αναπαριστά τον επόμενο ψευδοκόμβο στη λίστα

Ακολουθεί το struct:

```
struct pseudoknot {
int left_left_loop_size;
int left_right_loop_size;
int right_left_loop_size;
int dd_size;
```

```
struct pseudoknot *next ;
};
```

ΕΙΔΙΚΗ ΜΝΕΙΑ: ΕΙΔΗ ΚΟΜΒΩΝ ΣΤΑ ΔΕΝΤΡΑ YAEP

Οι κατηγορίες των κόμβων σε αυτά τα δέντρα περιέχονται στο enumeration yaep_tree_node_type [25], το οποίο αναπαριστά όλους τους πιθανούς κόμβους του αφηρημένου συντακτικού δέντρου που προκύπτει από την ανάλυση της γραμματικής. Παρακάτω παρατίθονται οι κατηγορίες αυτές, οι ιδιαιτερότητες της κάθε μίας(εσωτερικές μεταβλητές) καθώς και σε παρένθεση η ονομασία της αντίστοιχης μεταβλητή:

- YAEP_NIL(nil): Αναπαριστά έναν κόμβο με κενή μετάφραση.
- YAEP_ERROR(error): Αναπαριστά έναν κόμβο που περιέχει μετάφραση λάθους.
- YAEP_TERM(term): Αναπαριστά έναν τερματικό κόμβο. Περιέχει την ακέραια μεταβλητή “code”, η οποία δείχνει τον κωδικό του τερματικού κόμβου, καθώς και τη μεταβλητή “attr” τύπου *void που είναι αναφορά σε ένα χαρακτηριστικό του συγκεκριμένου κόμβου.
- YAEP_ANODE(anode): Αναπαριστά έναν αφηρημένο κόμβο. Έχει τη μεταβλητή “name” τύπου const char*, που εκφράζει το όνομα του κόμβου όπως δίνεται στη μετάφραση. Περιέχει επίσης τη μεταβλητή “children” τύπου yaep_tree_node, η οποία αναφέρεται στους κόμβους που είναι μεταφράσεις των συμβόλων του κανόνα της γραμματικής, ουσιαστικά τα παιδιά του κόμβου. Επιπλέον, υπάρχει η ακέραια μεταβλητή “cost”, η οποία ανάλογα με τη ρύθμιση ενός flag, εκφράζει είτε το συνολικό κόστος του κόμβου μαζί με το κόστος των παιδιών του, είτε μόνο το κόστος του συγκεκριμένου αφηρημένου κόμβου.
- YAEP_ALT(alt): Αυτός ο τύπος κόμβου υποδεικνύει ότι υπάρχουν πολλαπλές πιθανές μεταφράσεις για τον συγκεκριμένο κόμβο. Εμπεριέχει τη μεταβλητή “node” τύπου yaep_tree_node, η οποία αναφέρεται στην πρώτη εναλλακτική μετάφραση του κόμβου. Τέλος περιλαμβάνει την μεταβλητή “next”, επίσης ίδια τύπου, η οποία αναφέρεται στην επόμενη εναλλακτική μετάφραση.

Επεξηγηματικό παράδειγμα:

Για τον κανόνα της γραμματικής μας`
το όνομα του αφηρημένου κόμβου είναι το “R01” και η λίστα (1 3 5 7 9) μας δείχνει τις θέσεις των παιδιών του κόμβου. Επομένως για να διασχίσθεί το δέντρο, έχει υλοποιήσει μια συνάρτηση με την οποία ελέγχεται το είδος του κάθε κόμβου που συναντάμε. Εάν ο κόμβος είναι τύπου YAEP_NIL, YAEP_ERROR ή YAEP_TERM, η συνάρτηση επιστρέφει NULL. Αυτό γίνεται διότι δεν μπορεί να πραγματοποιηθεί

κάποια διάσχιση σε αυτούς τους κόμβους σύμφωνα με τη γραμματική. Στην περίπτωση που ο κόμβος είναι τύπου YAEP_ALT, η ίδια συνάρτηση εφαρμόζεται αναδρομικά τόσο για όλες τις μεταφράσεις του κόμβου, αν υπάρχουν πάνω από μία, και οι ψευδοκόμβοι που προκύπτουν από αυτές τις κλήσεις προστίθενται στην λίστα. Αν ο κόμβος ανήκει στην κατηγορία YAEP_ANODE, ελέγχεται ο πρώτος χαρακτήρας του ονόματός του και χωρίζονται δύο περιπτώσεις. Αν ο πρώτος χαρακτήρας του ονόματος του κόμβου είναι 'R', τότε το όνομα είναι έγκυρο σύμφωνα με τη γραμματική και επομένως υπολογίζονται όλα τα μεγέθη που περιέχονται στο struct της δομής του κόμβου. Εάν ο πρώτος χαρακτήρας δεν είναι 'R', τότε, σύμφωνα με τη γραμματική που έχει υλοποιηθεί, επιστρέφεται NULL, διότι όλοι οι κανόνες της γραμματικής ξεκινούν με το χαρακτήρα 'R'.

Αναλυτικά, η διαδικασία ξεκινά με την κλήση της συνάρτησης traverse_parse_tree_for_loop για το πρώτο παιδί σύμφωνα με τον κανόνα, δηλαδή το χαρακτήρα στη θέση 1. Όπως αναφέρθηκε προηγουμένως, η λίστα (1, 3, 5, 7, 9) υποδεικνύει ότι ο πρώτος χαρακτήρας είναι ο L, οπότε υπολογίζεται το μήκος της ακολουθίας χαρακτήρων μεταξύ της πρώτης και της δεύτερης κύριας βάσης. Στη συνέχεια, η ίδια συνάρτηση καλείται για το δεύτερο παιδί του κόμβου, δηλαδή το χαρακτήρα L στη θέση 3, για να υπολογιστεί το μήκος της ακολουθίας μεταξύ της δεύτερης και της τρίτης κύριας βάσης. Μετά, καλείται η συνάρτηση traverse_parse_tree_for_dd για το χαρακτήρα D στη θέση 5, ώστε να βρεθεί το μήκος της ακολουθίας μεταξύ των δύο ενδιάμεσων βάσεων για τον τύπο K, δηλαδή την τρίτη και την τέταρτη βάση. Τέλος, καλείται ξανά η συνάρτηση traverse_parse_tree_for_loop για το τέταρτο παιδί του κανόνα, δηλαδή το χαρακτήρα L στη θέση 7, προκειμένου να υπολογιστεί το μήκος της ακολουθίας μεταξύ της τέταρτης και της πέμπτης κύριας βάσης. Ακολουθεί η συνάρτηση traverse_parse_tree:

```
struct pseudoknot *traverse_parse_tree(struct yaep_tree_node *node) {
   int left_left_loop_size, left_right_loop_size, right_left_loop_size;
   struct size *dd_sizes;
   struct pseudoknot *pknots = NULL;
   struct pseudoknot *pseudoknots = NULL;

   if (!node) {
      return pseudoknots;
   }

   switch (node->type) {
      case YAEP_ERROR:
         return NULL;
      case YAEP_NIL:
         return NULL;
      case YAEP_TERM:
```

```
        return NULL;
    case YAEP_ANODE:
        if ((node->val.anode.name)[0] != 'R') {
            return NULL;
        }
```

```
left_left_loop_size =traverse_parse_tree_for_loop(node-
>val.anode.children[0]);
left_right_loop_size =traverse_parse_tree_for_loop(node-
>val.anode.children[1]);
dd_sizes = traverse_parse_tree_for_dd(node->val.anode.children[2]);
right_left_loop_size =traverse_parse_tree_for_loop(node-
>val.anode.children[3]);
while (dd_sizes != NULL) {
pseudoknots = append_pseudoknot_if_not_exists
(pseudoknots, create_pseudoknot(left_left_loop_size,
left_right_loop_size,   right_left_loop_size, dd_sizes->value));
dd_sizes = dd_sizes->next;
}
```

```
        return pseudoknots;
        case YAEP_ALT:
        pknots = traverse_parse_tree(node->val.alt.node);
        pseudoknots = NULL;
        if (node->val.alt.next != NULL) {
            pseudoknots = traverse_parse_tree(node->val.alt.next);
        }
        pseudoknots = concatenate_punique(pseudoknots, pknots);
        return pseudoknots;
                default:
        printf("I don't care!\n");
        return pseudoknots;
                }
    }
```

Μελετώντας την κάθε συνάρτηση ξεχωριστά, αρχικά η συνάρτηση create_pseudoknot είναι υπεύθυνη για τη δημιουργία ενός νέου ψευδοκόμβου, χρησιμοποιώντας τις τιμές που έχουν υπολογιστεί προηγουμένως ως χαρακτηριστικά του. Ακολούθως, η συνάρτηση append_pseudoknot_if_not_exists συγκρίνει αυτά τα χαρακτηριστικά με εκείνα των ψευδοκόμβων που βρίσκονται ήδη στην τελική λίστα. Εάν βρεθεί κάποιος ψευδοκόμβος με ακριβώς τα ίδια χαρακτηριστικά, η λίστα παραμένει αμετάβλητη. Αν δεν εντοπιστεί ταύτιση, ο νέος ψευδοκόμβος προστίθεται στην αρχή της λίστας. Η συνάρτηση concatenate_punique ενώνει σε μια λίστα όλους τους ψευδοκόμβους που προκύπτουν από τις κλήσεις της

συνάρτησης. Έτσι, δημιουργείται μια λίστα που εμπεριέχει όλες τις μεταφράσεις κόμβων τύπου YAEP_ALT.

Για να υπολογιστούν όλα τα μήκη των ακολουθιών πίσω από τους χαρακτήρες L χρησιμοποιήθηκε μια άλλη συνάρτηση, η traverse_parse_tree_for_loop, η οποία έκανε κατά βάθος διάσχιση των υποδέντρων. Πιο αναλυτικά, αν ο κόμβος που εξετάζεται είναι τύπου YAEP_ANODE, τότε η συνάρτηση καλείται αναδρομικά για το πρώτο παιδί. Για παράδειγμα, αν ο κανόνας περιέχει δύο χαρακτήρες ως παιδιά, το πρώτο παιδί βρίσκεται στη θέση 1 και ο επόμενος χαρακτήρας είναι ο L, οπότε η διαδικασία κατά βάθος διάσχισης συνεχίζεται. Εάν ο κόμβος έχει μόνο ένα παιδί, τότε η συνάρτηση καλείται για αυτό το παιδί, το οποίο είναι τερματικός χαρακτήρας. Σε αυτήν την περίπτωση, η συνάρτηση επιστρέφει την τιμή 1. Ακολυθεί η συνάρτηση traverse_parse_tree_for_loop:

```
int traverse_parse_tree_for_loop(struct yaep_tree_node *node) {
   switch (node->type) {
      case YAEP_ANODE:
        return traverse_parse_tree_for_loop(node->val.anode.children[0])   + 1;
       break;
      case YAEP_TERM:
         return 1;
         break;
      default:
         return -1;
         break;
   }
}
```

Έπειτα τον έλεγχο για το είδος του κόμβου και τη δημιουργία λίστας με όλες τις τιμές που είναι δυνατόν να έχει το μήκος της ακολουθίας D αναλαμβάνει η συνάρτηση traverse_parse_tree_for_dd. Αν ο κόμβος ανήκει στον τύπο YAEP_ALT, η συνάρτηση καλείται αναδρομικά για κάθε μία από τις δύο μεταφράσεις του κόμβου. Στην περίπτωση που ο κόμβος είναι τύπου YAEP_ANODE, η συνάρτηση καλείται αναδρομικά για κάθε παιδί του κόμβου, εφόσον ο πρώτος χαρακτήρας του ονόματος του κανόνα είναι 'M'. Αντίθετα, εάν ο πρώτος χαρακτήρας είναι 'N', η συνάρτηση καλείται για το ένα και μοναδικό παιδί του κόμβου, το οποίο αντιπροσωπεύεται από τον κανόνα για το χαρακτήρα 'E'. Όταν ο κόμβος είναι τύπου YAEP_TERM, η συνάρτηση δημιουργεί ένα αντικείμενο τύπου size και του αναθέτει την τιμή 1, υποδεικνύοντας απλά ότι υπάρχει .Τέλος, για κόμβο τύπου YAEP_NIL, αποδίδεται η τιμή 0, δηλώνοντας τη μη ύπαρξη κόμβου σε εκείνη τη θέση. Ακολουθεί η συνάρτηση:

```
struct size *traverse_parse_tree_for_dd(struct yaep_tree_node *node) {
```

```
    switch (node->type) {
        case YAEP_ANODE:
            if ((node->val.anode.name)[0] == 'M') {
           struct size **childSizes = malloc(s_max_dd_size * sizeof(struct size *));
                    for (int i = 0; i < s_max_dd_size; i++) {
           childSizes[i] =traverse_parse_tree_for_dd(node->val.anode.children[i]);
                }
                return multiCartesianProduct(childSizes, s_max_dd_size);
            } else if ((node->val.anode.name)[0] == 'N') {
                return traverse_parse_tree_for_dd(node->val.anode.children[0]);
            }
            break;
        case YAEP_TERM:
            return create_size(1);
            break;
        case YAEP_NIL:
            return create_size(0);
            break;
        case YAEP_ALT: {
         struct size *anode_dds = traverse_parse_tree_for_dd(node->val.alt.node);
         struct size *alt_dds = NULL;
            if (node->val.alt.next != NULL) {
                alt_dds = traverse_parse_tree_for_dd(node->val.alt.next);
            }
            struct size *new_list = concatenate_sunique(anode_dds, alt_dds);
            return new_list;
            break;
        }
        default:
            printf("I think something went really wrong with node type\n");
    }
    return NULL;
}
```

Η συνάρτηση concatenate_sunique ενώνει σε μια λίστα όλους τους αριθμούς που προέκυψαν από τις κλήσεις, τις συνάρτησεις σε περίπτωση τύπου YAEP_ALT. Έτσι έχουμε όλες τις πιθανές τιμές του μήκους της ακολουθίας.

Η παραπάνω διαδικασία ανάλυσης της συμβολοσειράς και ανίχνευσης όλων των ψευδοκόμβων πραγματοποιείται μέσω μιας διπλής επανάληψης, στην οποία καθορίζεται τόσο το σημείο εκκίνησης κάθε ψευδοκόμβου όσο και το συνολικό του μέγεθος. Συγκεκριμένα, οι μεταβλητές `left` και `right`, οι οποίες υπάρχουν μέσα στη διπλή επανάληψη, καθορίζουν τα τμήματα της εισόδου όπου ο parser θα αναζητήσει τους ψευδοκόμβους, εάν και εφόσον υπάρχουν. Η γνώση των δύο άκρων του

ψευδοκόμβου μέσω των παραπάνω μεταβλητών μας απαλλάσσει από την ανάγκη υπολογισμού του μήκους της ακολουθίας μεταξύ των δύο τελευταίων κύριων βάσεων για τον τύπο L. Για τον λόγο αυτό, στο struct του ψευδοκόμβου που αναφέρθηκε νωρίτερα, δεν περιλαμβάνεται μεταβλητή που να εκφράζει αυτή την απόσταση. Για κάθε ψευδοκόμβο που εντοπίζεται, ελέγχεται η μεταβλητή `dd_size`, η οποία αντιπροσωπεύει το μήκος της ακολουθίας μεταξύ των δύο ενδιάμεσων κύριων βάσεων. Εάν η τιμή της είναι κάτω από ένα συγκεκριμένο όριο, το οποίο καθορίζεται από τη μεταβλητή `min_dd_size`, τότε ο ψευδοκόμβος απορρίπτεται. Η μεταβλητή `min_dd_size` μπορεί να οριστεί από τον χρήστη κατά την εκτέλεση του προγράμματος, ενώ η προεπιλεγμένη τιμή της αν δεν οριστεί είναι 0. Οι ψευδοκόμβοι που πληρούν τα κριτήρια επιστρέφονται σε μια συνάρτηση υλοποιημένη σε Python, η οποία εκτελεί όλες τις κλήσεις των συναρτήσεων σε C για την ανάλυση της RNA ακολουθίας. Ακολουθεί ο κώδικας που υλοποιεί όλα τα προαναφερθέντα:

```
for (int right = len - 1; right >= min_window_size - 1; right--) {
  for (int left = right - min_window_size + 1;
       left > right - max_window_size && left >= 0; left--) {
    s_ntok = left;
    struct yaep_tree_node *root = parse(s_definition);
    struct pseudoknot *ps = traverse_parse_tree(root);

    for (struct pseudoknot *i = ps; i != NULL; i = i->next) {
      if (i->dd_size < s_min_dd_size) {
        continue;
      }
      cb(left, right - left + 1, i->left_left_loop_size,
         i->left_right_loop_size, i->right_left_loop_size, i->dd_size);
    }
  }
  s_input[right] = '\0';
 }
}
```

Πρέπει να σημειωθεί επίσης ότι μέσα από τη συνάρτηση cb (callback) επιστρέφονται τα χαρακτηριστικά του κάθε ψευδοκόμβου στον κώδικα της python ώστε μετά να επεξεργαστούν σε άλλο κομμάτι και να προκύψει η dot bracket μορφή. Η εντολή της συνάρτησης σε c από τη συνάρτηση σε python είναι η εξής:

```
for (i, j, left_left_loop_size, left_right_loop_size, right_left_loop_size, dd_size)
in parser(sequence):
```

Η μεταβλητή parser μπορεί να πάρει δύο πιθανές τιμές από τον χρήστη, yaep και bruteforce. Με την πρώτη επιλογή θα εκτελεστεί ο αλγόριθμος συντακτικής αναγνώρισης που περιγράφτηκε παραπάνω και με την δεύτερη ο άπληστος αλγόριθμος δυναμικού προγραμματισμού που θα περιγραφεί στο επόμενο κεφάλαιο.

4.3 Άπληστος Αλγόριθμος Δυναμικού Προγραμματισμού

Στον συγκεκριμένο αλγόριθμο, αρχικά έχουν δημιουργηθεί δύο δομές για τους δεσμούς που εντοπίζονται μεταξύ των βάσεων στην ακολουθία του RNA. Η πρώτη δομή, που ονομάζεται `stem`, περιέχει δύο ακέραιες μεταβλητές, τις `left` και `right`, οι οποίες αναπαριστούν τις θέσεις των δύο βάσεων που σχηματίζουν κάθε ζευγάρι στην ακολουθία του RNA. Η δεύτερη δομή ονομάζεται `core_stem` και περιέχει έναν πίνακα τύπου `stem` με τρεις θέσεις. Σε αυτόν τον πίνακα αποθηκεύονται οι θέσεις των βάσεων για καθένα από τους τρεις κύριους δεσμούς που αντιπροσωπεύουν τον ψευδοκόμβο τύπου K. Για κάθε χαρακτήρα της συμβολοσειράς που αντιπροσωπεύει μία βάση, ελέγχεται αν υπάρχει βάση σε θέση μεγαλύτερη από τη δική του με την οποία μπορεί να σχηματίσει ζευγάρι. Επιπλέον, έχει αρχικοποιηθεί ένας πίνακας με μέγεθος ίσο με το τετράγωνο του μήκους της ακολουθίας, για την αποθήκευση όλων των ζευγαριών που εντοπίζονται, καθώς και μία ακέραια μεταβλητή, η `n_stems`, η οποία αυξάνεται κάθε φορά που εντοπίζεται ένα ζευγάρι βάσεων. Αυτή η προσέγγιση διασφαλίζει ότι όλες οι απαραίτητες πληροφορίες σχετικά με τους δεσμούς των βάσεων καταγράφονται και αποθηκεύονται με ακρίβεια, επιτρέποντας την περαιτέρω ανάλυση και επεξεργασία των δεδομένων. Παρακάτω παρατίθεται ο κώδικας για αυτή τη διαδικασία.

```
for (int i = 0; i < n; i++) {
   switch (sequence[i]) {
   case 'a':
     for (int j = i + 1; j < n; j++)
       if (sequence[j] == 'u') {
         cs_position->left = i;
         cs_position->right = j;
         cs_position++;
         n_stems++;
       }
     break;
   case 'u':
     for (int j = i + 1; j < n; j++)
       if ((sequence[j] == 'a') || ((sequence[j] == 'g') && s_allow_ug)) {
         cs_position->left = i;
         cs_position->right = j;
```

```
        cs_position++;
        n_stems++;
       }
     break;
   case 'c':
     for (int j = i + 1; j < n; j++)
       if (sequence[j] == 'g') {
         cs_position->left = i;
         cs_position->right = j;
         cs_position++;
         n_stems++;
       }
     break;
   case 'g':
     for (int j = i + 1; j < n; j++)
       if ((sequence[j] == 'c') || ((sequence[j] == 'u') && s_allow_ug)) {
         cs_position->left = i;
         cs_position->right = j;
         cs_position++;
         n_stems++;
       }
     break;
   default:
     printf("Invalid character\n");
   }
 }
```

Όπως φαίνεται στη συνάρτηση, κάθε φορά που εντοπίζεται ένα ζευγάρι βάσεων, αποθηκεύεται στη μεταβλητή left η θέση της βάσης που βρίσκεται πιο αριστερά στο ζευγάρι και στη μεταβλητή right η θέση της βάσης που βρίσκεται πιο δεξιά. Παράλληλα, προχωράμε στην επόμενη θέση του πίνακα αυξάνοντας τη μεταβλητή cs_position, ενώ αυξάνεται κατά 1 και η μεταβλητή n_stems, η οποία διατηρεί τον συνολικό αριθμό ζευγαριών στη δομή. Έπειτα ελέγχονται ανά τριάδες όλα τα ζευγάρια μεταξύ βάσεων για να διαπιστωθεί αν οι θέσεις των βάσεων είναι τέτοιες ώστε να σχηματίζουν τη δομή του ψευδοκόμβου τύπου Κ. Τα παραπάνω υλοποιούνται στον ακόλουθο κώδικα:

```
 for (int i = 0; i < n_stems; i++) {

   cs_position = all_stems + i;

   for (int j = i + 1; j < n_stems; j++) {
```

```
    cs_position2 = all_stems + j;
    for (int k = j + 1; k < n_stems; k++) {
      cs_position3 = all_stems + k;
      if ((cs_position->left < cs_position2->left) &&
          (cs_position2->left < cs_position->right) &&
          (cs_position3->left < cs_position2->right) &&
          (cs_position->right < cs_position3->left &&
          (cs_position2->right < cs_position3->right)) {
        ccs_position->cstem[0].left = cs_position->left;
        ccs_position->cstem[0].right = cs_position->right;
        ccs_position->cstem[1].left = cs_position2->left;
        ccs_position->cstem[1].right = cs_position2->right;
        ccs_position->cstem[2].left = cs_position3->left;
        ccs_position->cstem[2].right = cs_position3->right;
        ccs_position++;
        n_cstems++;
      }
    }
  }
}
```

Οι μεταβλητές cs_position, cs_position1 και cs_position2 χρησιμοποιούνται για να επιτρέψουν την πρόσβαση στη δομή όπου είναι αποθηκευμένα τα ζευγάρια από την προηγούμενη φάση. Γίνεται τριπλή πρόσβαση μέσω μιας τριπλής επανάληψης `for`. Μετά, χρησιμοποιώντας την εντολή `if`, ελέγχεται αν οι θέσεις των βάσεων στα τρία ζευγάρια είναι τέτοιες ώστε να σχηματίζεται ένας ψευδοκόμβος τύπου K. Ο έλεγχος που αφορά την if έχει τα εξής κριτήρια:

- Η θέση της πρώτης βάσης του πρώτου ζευγαριού είναι μικρότερη από την αντίστοιχη της πρώτης βάσης του δεύτερου ζευγαριού.
- Η θέση της πρώτης βάσης του δεύτερου ζευγαριού είναι μικρότερη από την αντίστοιχη της δεύτερης βάσης του πρώτου ζευγαριού.
- Η θέση της πρώτης βάσης του τρίτου ζευγαριού είναι μικρότερη από την αντίστοιχη της δεύτερης βάσης του δεύτερου ζευγαριού.
- Η θέση της δεύτερης βάσης του πρώτου ζευγαριού είναι μικρότερη από την αντίστοιχη της πρώτης βάσης του τρίτου ζευγαριού.
- Η θέση της δεύτερης βάσης του δευτέρου ζευγαριού είναι μικρότερη από την αντίστοιχη της δεύτερης βάσης του τρίτου ζευγαριού.

Εάν η if ικανοποιείται από όλες τις παραπάνω συνθήκες τότε καταχωρείται η τριάδα των ζευγαριών στη δομή τύπου core_stem, στην οποία θα καταχωρηθούν και όλες οι τριάδες ψευδοκόμβων τύπου K. Έπειτα προχωράμε στην επόμενη θέση του πίνακα για να αποθηκευτεί η επόμενη τριάδα. Όταν εντοπιστούν όλες οι πιθανές τριάδες βάσεων, υπολογίζεται τα χαρακτηριστικά του κάθε ψευδοκόμβου ως συνάρτηση των θέσεων των παραπάνω βάσεων. Γίνονται περαιτέρω έλεγχοι για τα μεγέθη αυτά και αν ικανοποιούνται οι επιθυμητές συνθήκες ο ψευδοκόμβος επιστρέφεται στην python. Ο κώδικας που υλοποιεί τα παραπάνω είναι ο εξής:

```
for (int i = 0; i < n_cstems; i++) {

  int left = (ccs_position->cstem[0]).left;

  int size = (ccs_position->cstem[2]).right - ccs_position->cstem[0].left + 1;

  int left_left_loop_size =

     ccs_position->cstem[1].left - ccs_position->cstem[0].left - 1;

  int left_right_loop_size =

     ccs_position->cstem[0].right - ccs_position->cstem[1].left - 1;

  int right_left_loop_size =

     ccs_position->cstem[1].right - ccs_position->cstem[2].left - 1;

  int dd_size =

     ccs_position->cstem[2].left - ccs_position->cstem[0].right - 1;

  if (size < min_window_size || size > max_window_size ||

     dd_size < s_min_dd_size || dd_size > s_max_dd_size ||

     left_left_loop_size == 0 || left_right_loop_size == 0 ||
```

```
      right_left_loop_size == 0 ||
      (ccs_position->cstem[1].right == ccs_position->cstem[2].right - 1)) {
    ccs_position++;
    continue;
  }

  cb(left, size, left_left_loop_size, left_right_loop_size,
     right_left_loop_size, dd_size);
  ccs_position++;
}
```

Στον παραπάνω κώδικα η μεταβλητή left εκφράζει τη θέση της αρχής του ψευδοκόμβου και λαμβάνει ως τιμή τη θέση της αριστερής βάσης του πρώτου ζευγαριού. Η μεταβλητή size δείχνει το συνολικό μέγεθος του ψευδοκόμβου και λαμβάνει ως τιμή τη διαφορά των δύο ακριανών βάσεων. Η left_left_loop_size ορίζεται ως η διαφορά των θέσεων της αριστερής βάσης του δευτέρου ζευγαριού και της αριστερής βάσης του πρώτου ζευγαριού, μειωμένη κατά 1. Η μεταβλητή left_right_loop_size ορίζεται ως η διαφορά των θέσεων της δεξιάς βάσης του πρώτου ζευγαριού και της αριστερής βάσης του δευτέρου ζευγαριού, μειωμένη κατά 1. Η μεταβλητή right_left_loop_size υπολογίζεται ως η διαφορά των θέσεων μεταξύ της δεξιάς βάσης του δεύτερου ζευγαριού και της αριστερής βάσης του τρίτου ζευγαριού, μειωμένη επίσης κατά 1. Τέλος η μεταβλητή dd_size λαμβάνει ως τιμή τη διαφορά των θέσεων της πρώτης βάσης του τρίτου ζευγαριού και της δεξιάς βάσης του πρώτου ζευγαριού μειωμένη και αυτή κατά 1. Στην τελευταία if γίνεται ο έλεγχος για κάποιες αναγκαίες συνθήκες που πρέπει να ικανοποιεί ο ψευδοκόμβος όπως όλες οι μεταβλητές να έχουν τιμή άνω του μηδενός, οι δύο τελευταίες βάσεις να έχουν διαφορά θέσης μεγαλύτερη από ένα ώστε να μην συμπίπτουν, η dd_size να είναι μεταξύ των ορίων min_dd_size και max_dd_size που έχει τεθεί και αντίστοιχα η μεταβλητή size μεταξύ των ορίων min_window_size και max_window_size.

4.4 Εύρεση ζευγαριών μεταξύ των βάσεων γύρω από τη δομή του ψευδοκόμβου τύπου K

Ακολουθεί η συνάρτηση μέσω της οποίας εντοπίστηκαν όλα τα ζευγάρια βάσεων γύρω από τη δομή του ψευδοκόμβου τύπου K:

```
void pairalign(char *sequence, int i, int j, int left_left_loop_size, int left_right_loop_size, int right_left_loop_size,int dd_size,
            void (*cb)(char *, int, int, int)) {

  int L = i;
  int M = i + left_left_loop_size + 1;
  int R = i +left_left_loop_size +left_right_loop_size + dd_size +3;
  int l = i + left_left_loop_size + left_right_loop_size + 2;
  int m= l +right_left_loop_size  + dd_size +2;
  int r = i + j - 1;

  char *dot_bracket = strdup(sequence);
  int left_loop_stems = 0, middle_loop_stems=0,right_loop_stems = 0;

  // initialize dot bracket
  int len = strlen(sequence);
  memset(dot_bracket, '.', len);
  dot_bracket[L] = '(';
  dot_bracket[l] = ')';
  dot_bracket[M] = '[';
  dot_bracket[m] = ']';
  dot_bracket[R] = '{';
  dot_bracket[r] = '}';

  // left loop stems
  left_loop_stems = 0;
  for (int a = L - 1, b = l + 1; a >= 0 && b <= R - 1; a--, b++) {
    if (!IS_PAIR(sequence[a], sequence[b])) {
      break;
    }
    dot_bracket[a] = '(';
    dot_bracket[b] = ')';
    left_loop_stems++;
  }

  middle_loop_stems=0;
  for (int a = M - 1, b = m + 1; a >= L + 1 && b <= r - 1; a--, b++) {
    if (!IS_PAIR(sequence[a], sequence[b])) {
      break;
    }
```

```
    dot_bracket[a] = '[';
    dot_bracket[b] = ']';
    middle_loop_stems++;
  }

  // right loop stems
  right_loop_stems = 0;
  for (int a = R - 1, b = r + 1; a >= l + 1 && b <= len - 1; a--, b++) {
    if (!IS_PAIR(sequence[a], sequence[b])) {
      break;
    }
    dot_bracket[a] = '{';
    dot_bracket[b] = '}';
    right_loop_stems++;
  }

  cb(dot_bracket, left_loop_stems, middle_loop_stems, right_loop_stems);

  free(dot_bracket);
}
```

Σε αυτό το σημείο θα περιγραφεί η διαδικασία την οποία υλοποιεί ο παραπάνω κώδικας. Αρχικά, αναφέρεται ότι τα ορίσματα της συνάρτησης είναι οι τιμές που υπολογίστηκαν στο προηγούμενο βήμα. Εντός της συνάρτησης οι μεταβλητές L, M, R αναπαριστούν τις θέσεις της πρώτης, της δεύτερης και της τρίτης κύριας βάσης ενώ οι l, m, r της τέταρτης, της πέμπτης και της έκτης. Η συνολική ακολουθία θα απεικονιστεί με τελίτσες και στη συνέχεια τοποθετούνται αγκύλες, παρενθέσεις και άγκιστρα στις θέσεις των κύριων βάσεων του ψευδοκόμβου, ώστε να φτάσουμε στην επιθυμητή αναπαράσταση με την μορφή dot-bracket. Οι βάσεις του πρώτου ζευγαριού αναπαριστώνται με τα σύμβολα “(“ και “)”, οι βάσεις του δευτέρου με τα σύμβολα “[“ και “]” και τέλος οι βάσεις του τρίτου ζευγαριού με τα “{“ και “}”. Αυτό γίνεται διότι καθώς αυτά τα σύμβολα είναι συμμετρικά αντίθετα μεταξύ τους σαν ζεύγη ο χρήστης να μπορεί οπτικά να κατανοήσει που βρίσκονται οι βάσεις του ψευδοκόμβου στην έξοδο του προγράμματος.

Στη συνέχεια, βλέπουμε πως υπολογίζονται τα ζευγάρια μεταξύ των βάσεων και χειρίζονται από τις μεταβλητές: left_loop_stems, για τα ζευγάρια γύρω από το κύριο ζευγάρι, middle_loop_stems, για τα ζευγάρια γύρω από το δεύτερο ζευγάρι και right_loop_stemas για τα ζευγάρια γύρω από το τρίτο. Η γενική περιγραφή της διαδικασίας είναι ότι στην πρώτη επανάληψη της for γίνεται ο έλεγχος για το εάν η τελευταία βάση της πρώτης περιοχής και η πρώτη της δεύτερης έχουν τη δυνατότητα να σχηματίσουν ζευγάρι. Εάν το ζευγάρι σχηματίζεται, προχωράμε στην επόμενη θέση και στις δύο περιοχές για τον έλεγχο του επόμενου ζευγαριού ειδάλλως διακόπτεται η διαδικασία μέσω του break. Για τον εντοπισμό ζευγαριών γύρω από το

δεύτερο κύριο ζευγάρι εφαρμόζεται η ίδια διαδικασία όπως πριν αλλά με περιοχή αναζήτησης εκείνη ανάμεσα στην πρώτη και την δεύτερη κύρια βάση και εκείνη ανάμεσα στην πέμπτη και την έκτη. Τέλος η περιοχή αναζήτησης για το τρίτο σύνολο ζευγαριών είναι η περιοχή ανάμεσα στην τρίτη και στην τέταρτη κύρια βάση και η περιοχή δεξιά από την πέμπτη. Εν τέλει εφόσον έχει ολοκληρωθεί η αναπαράσταση dot-bracket επιστρέφεται στην αρχική συνάρτηση python μαζί με τις μεταβλητές που υποδεικνύουν τον αριθμό των ζευγαριών που δημιουργούνται στις προαναφερθείσες περιοχές.

4.5 Απαλοιφή ζευγαριών αδενίνης ουρακίλης

Σε αυτό το κεφάλαιο αναφέρεται το τελευταίο στάδιο της αναπαράστασης που υλοποιήθηκε και αφορά την αφαίρεση των ζευγαριών αδενίνης-ουρακίλης στα άκρα της ακολουθίας, αν το επιλέγει ο χρήστης. Η μεταβλητή μέσω της οποίας ορίζει ο χρήστης εάν επιθυμεί την απαλοιφή, είναι μια δυαδική μεταβλητή και παρακάτω παρατίθεται η υλοποίηση της συνάρτησης αφαίρεσης των ζευγαριών:

```
void skip_final_au(char *sequence, char *dot_bracket, int left_loop_stems,
                   int right_loop_stems, void (*cb)(char *, int, int)) {
  int L = -1, l = -1, R = -1, r = -1;

  char *bracket = strdup(dot_bracket);

  // find indices of last loop stems
  for (int i = 0; i < strlen(sequence); i++) {
    if (bracket[i] == '(' && L == -1) {
      L = i;
    } else if (bracket[i] == '{' && R == -1) {
      R = i;
    } else if (bracket[i] == ')') {
      l = i;
    } else if (bracket[i] == '}') {
      r = i;
    }
  }

  // left loop stem ends with an AU pair
  int left_is_au = left_loop_stems > 0 && IS_AU(sequence[L], sequence[l]);
  int right_is_au = right_loop_stems > 0 && IS_AU(sequence[R], sequence[r]);

  if (left_is_au) {
```

```
    bracket[L] = bracket[l] = '.';
    cb(bracket, left_loop_stems - 1, right_loop_stems);
    bracket[L] = '(';
    bracket[l] = ')';
  }
  if (right_is_au) {
    bracket[R] = bracket[r] = '.';
    cb(bracket, left_loop_stems, right_loop_stems - 1);
    bracket[R] = '{';
    bracket[r] = '}';
  }
  if (left_is_au && right_is_au) {
    bracket[L] = bracket[l] = bracket[R] = bracket[r] = '.';
    cb(bracket, left_loop_stems - 1, right_loop_stems - 1);
  }

  free(bracket);
}
```

Αρχικά η συνάρτηση εντοπίζει τις θέσεις των βάσεων που βρίσκονται στις άκρες της δομής και τις βάσεις με τις οποίες αυτές κάνουν ζευγάρια. Αποθηκεύεται στην μεταβλητή L η θέση της πιο αριστερής βάσης και στην μεταβλητή l εκείνη με την οποία κάνει ζευγάρι. Αντίστοιχα αποθηκεύεται η πιο δεξιά στην μεταβλητή r και στην R το ζευγάρι της. Η συνάρτηση left_is_au αφού ελέγξει αν το σύνολο των ζευγαριών στο οποίο εντοπίστηκε το αριστερό ζευγάρι είναι άνω του μηδενός (δηλαδή εάν υπάρχουν) και ένα, αποτελεί ζευγάρι μεταξύ αδενίνης και ουρακίλης. Με την μεταβλητή right_is_au ελέγχεται αντίστοιχα το ζευγάρι που περιέχει την πιο δεξιά βάση. Εάν κάποια από τις δύο προαναφερθείσες μεταβλητές είναι true, οι αγκύλες των συγκεκριμένων θέσεων στην αναπαράσταση dot-bracket αντικαθίστανται με τελίτσες. Επίσης μειώνεται η μεταβλητή που υποδεικνύει το πλήθος των ζευγαριών και επιστρέφεται η νέα dot-bracket αναπαράσταση μαζί τις μεταβλητές για τα δύο σύνολα ζευγαριών. Αν και οι δύο είναι true τότε αντικαθίστανται και τα 4 ειδικά σύμβολα με τελίτσες, μειώνονται κατά 1 οι μεταβλητές που μετράει το πλήθος των ζευγαριών και επιστρέφονται στο αρχικό πρόγραμμα.

Όλοι οι ψευδοκόμβοι και οι αναπαραστάσεις τους σε dot-bracket προστίθενται στην λίστα με όλες τις πιθανές περιπτώσεις δομής ψευδοκόμβου τύπου K στην εισαγόμενη ακολουθία. Εάν δεν έχει εντοπιστεί ψευδοκόμβος αυτού του τύπου από την όλη διαδικασία η τελική έξοδος του προγράμματος είναι μια dot bracket αναπαράσταση που αποτελείται μόνο από τελίτσες.

4.6 Επιλογή της βέλτιστης δομής

Το κριτήριο της επιλογής του βέλτιστου ψευδοκόμβου από το σύνολο της λίστας που έχει επιστραφεί από τον κώδικα των προηγούμενων βημάτων είναι το μικρότερο ποσό ελεύθερης ενέργειας αναδίπλωσης και ο μεγαλύτερος αριθμός ζευγαριών γύρω από τα κύρια ζευγάρια του ψευδοκόμβου. Αρχικά, αθροίζονται για κάθε ψευδοκόμβο της λίστας τα ζευγάρια βάσεων γύρω από τα κύρια ζευγάρια και κρατάμε μόνο εκείνους των οποίων το παραπάνω άθροισμα είναι πάνω από ένα όριο. Το όριο αυτό ορίζεται ως η διαφορά της μέγιστης τιμής του “left_loop__stems + middle_loop_stems + right_loop_stems” με την μεταβλητή max_stem_allow_smaller, η οποία ορίζεται από τον χρήστη.

Εφόσον έχει εκτελεστεί το παραπάνω βήμα και έχουν μείνει λιγότεροι ψευδοκόμβοι στη λίστα προχωράμε στο κριτήριο της ελεύθερης ενέργειας αναδίπλωσης. Αυτή η ενέργεια υπολογίζεται είτε με το πακέτο pkenergy είτε με το πακέτο Vienna RNA, τα οποία είναι και τα δύο υλοποιημένα σε C++. Στην πρώτη περίπτωση η ενέργεια υπολογίζεται από τον αριθμό των ζευγαριών των κύριων βάσεων και τον αριθμό των αζευγάρωτων βάσεων ενώ στην δεύτερη υπολογίζεται από την ακολουθία του RNA και την τελική dot bracket αναπαράστασή της. Αφού υπολογιστεί η ενέργεια για κάθε μία από τις δομές, αυτές ταξινομούνται σε αύξουσα σειρά με βάση την ενέργεια, σε φθίνουσα σειρά με βάση τον συνολικό αριθμό δεσμών και τέλος σε αύξουσα σειρά με βάση την τιμή του μήκους μεταξύ των δύο ενδιάμεσων κύριων βάσεων. Η πρώτη δομή της ταξινόμησης επιλέγεται και εμφανίζεται στην έξοδο του προγράμματος. Στην έξοδο της εκτέλεσης εμφανίζεται η συμβολοσειρά που αποτελεί την ακολουθία RNA που έχει δωθεί σαν input μαζί με την dot bracket αναπαράσταση, την ελεύθερη ενέργεια αναδίπλωσης και τον χρόνο εκτέλεσης. Ακολουθεί ο κωδικας που υλοποιεί την διαδικασία ταξινόμησης των ψευδοκόμβων με βάση όλα τα παραπάνω:

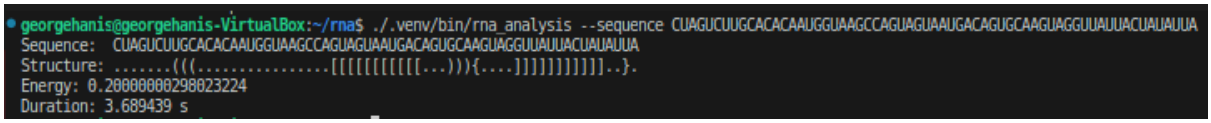
```
def apply_free_energy_and_stems_criterion(
    data: pd.DataFrame,
    sequence: str,
    max_stem_allow_smaller: int,
    energy: BaseEnergy,
):
    data["stems"] = data["left_loop_stems"] + data["right_loop_stems"] + data["middle_loop_stems"]
    data = data[
        data["stems"] >= data["stems"].max() - max_stem_allow_smaller
    ].reset_index()

    data["energy"] = data["dot_bracket"].apply(lambda r: energy.eval(sequence, r))
    data.sort_values(
        ["energy", "stems", "dd"],
```

```
        ascending=(True, False, True),
        inplace=True,
    )
    return data
```

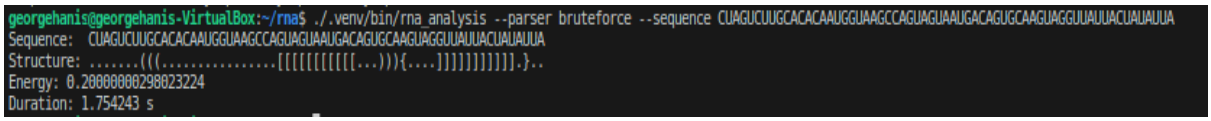
5. Επίδειξη λειτουργίας

5.1 Επιλογή αλγορίθμου

Όπως αναφέρεται σε προηγούμενο κεφάλαιο ο χρήστης έχει την δυνατότητα να εκτελέσει το πρόγραμμα με χρήση του αλγορίθμου συντακτικής αναγνώρισης και με χρήση του άπληστου αλγορίθμου δυναμικού προγραμματισμού. Παρακάτω παρατίθενται εικόνες που δείχνουν και τις δύο εκτελέσεις:



Σχήμα 5.1: *Εκτέλεση με χρήση του αλγορίθμου συντακτικής αναγνώρισης προτύπων*

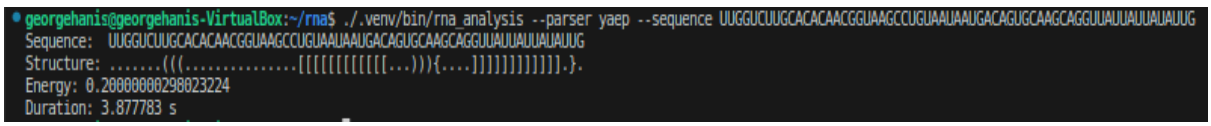


Σχήμα 5.2: *Εκτέλεση με χρήση του άπληστου αλγορίθμου δυναμικού προγραμματισμού*

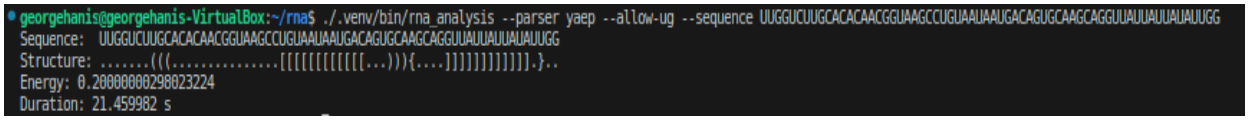
Βλέπουμε ότι η έξοδος του προγράμματος είναι η ίδια εκτός από τον χρόνο εκτέλεσης στον οποίο ο άπληστος αλγόριθμος υπερτερεί.

5.2 Έλεγχος για δεσμό γουανίνης-ουρακίλης

Παρακάτω φαίνεται η εκτέλεση του προγράμματος για μια ακολουθία RNA, με και χωρίς το όρισμα allow-ug, το οποίο επιτρέπει τον εντοπισμό ψευδοκόμβων που περιέχουν ζεύγη γουανίνης-ουρακίλης. Το πρόγραμμα εκτελείται με τον αλγόριθμο συντακτικής αναγνώρισης προτύπων.



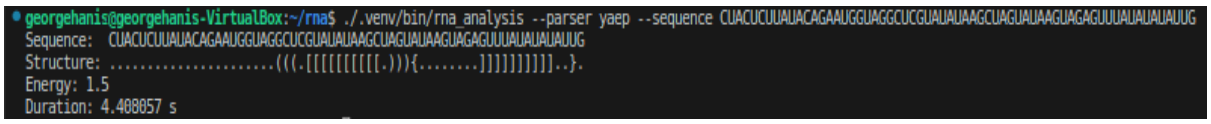
Σχήμα 5.3: *Εκτέλεση με χρήση του αλγορίθμου συντακτικής αναγνώρισης προτύπων*



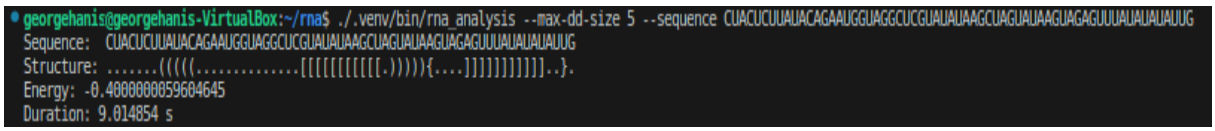
Σχήμα 5.4: *Εκτέλεση με χρήση του αλγορίθμου συντακτικής αναγνώρισης προτύπων με χρήση του ορίσματος allow-ug*

5.3 Εκτέλεση του προγράμματος με άνω και κάτω όριο στο μήκος ανάμεσα στις δυο ενδιάμεσες βάσεις

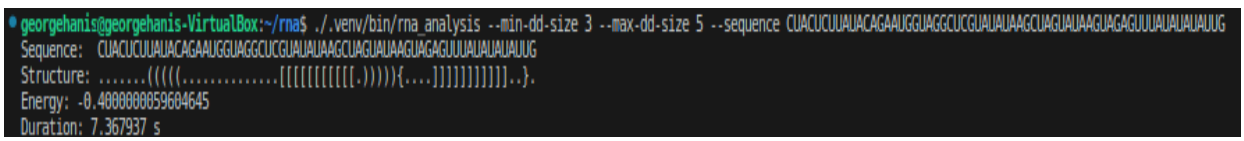
Παρακάτω παρατίθενται εικόνες εκτέλεσης του προγράμματος με τον αλγόριθμο συντακτικής αναγνώρισης προτύπων, έχοντας ορίσει τις μεταβλητές των ορίων για το μήκος της ακολουθίας ανάμεσα στις δύο ενδιάμεσες βάσεις. Βλέπουμε την εκτέλεση αφού ορίστηκε διαφορετικό max_dd_size και έπειτα αφού ορίστηκε η μεταβλητή min_dd_size. Στην πρώτη εικόνα φαίνεται η εκτέλεση χωρίς να έχει τεθεί όριο στην τιμή.



Σχήμα 5.5: *Εκτέλεση με χρήση του αλγορίθμου συντακτικής αναγνώρισης προτύπων*



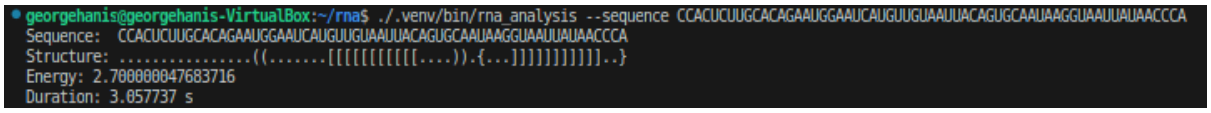
Σχήμα 5.6: *Εκτέλεση με χρήση του αλγορίθμου συντακτικής αναγνώρισης προτύπων με ορισμό άνω ορίου στο μήκος της ακολουθίας μεταξύ των δύο ενδιάμεσων βάσεων*



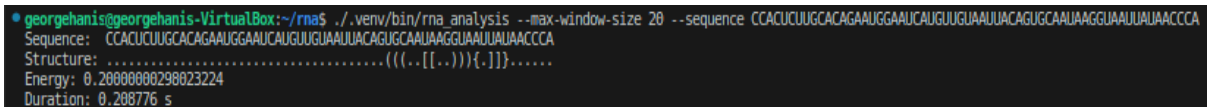
Σχήμα 5.7: *Εκτέλεση με χρήση του αλγορίθμου συντακτικής αναγνώρισης προτύπων με ορισμό κάτω ορίου στο μήκος της ακολουθίας μεταξύ των δύο ενδιάμεσων βάσεων*

5.4 Εκτέλεση του προγράμματος με άνω και κάτω όριο για το συνολικό μήκος του ψευδοκόμβου

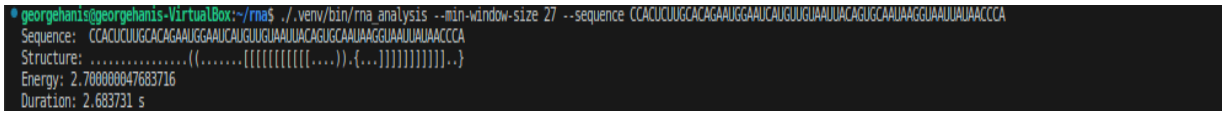
Παρακάτω παρατίθενται εικόνες εκτέλεσης του προγράμματος με τον αλγόριθμο συντακτικής αναγνώρισης προτύπων, έχοντας ορίσει όρια για το μήκος του ψευδοκόμβου. Βλέπουμε την εκτέλεση αφού ορίστηκε διαφορετικό max_window_size και έπειτα αφού ορίστηκε τη μεταβλητή min_window_size. Στην πρώτη εικόνα φαίνεται η εκτέλεση χωρίς να έχει τεθεί όριο στην τιμή.



Σχήμα 5.8: *Εκτέλεση με χρήση του αλγορίθμου συντακτικής αναγνώρισης προτύπων*



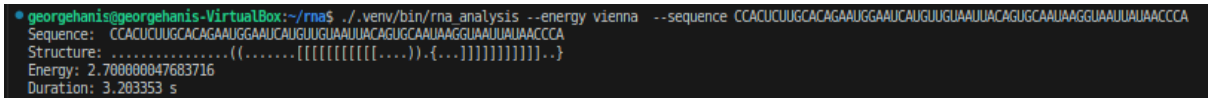
Σχήμα 5.9: *Εκτέλεση με χρήση του αλγορίθμου συντακτικής αναγνώρισης προτύπων με ορισμό άνω ορίου στο συνολικό μήκος του ψευδοκόμβου*



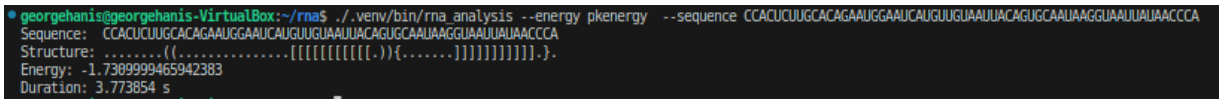
Σχήμα 5.10: *Εκτέλεση με χρήση του αλγορίθμου συντακτικής αναγνώρισης προτύπων με ορισμό κάτω ορίου στο συνολικό μήκος του ψευδοκόμβου*

5.5 Τροποποίηση του πακέτου μέσω του οποίου θα υπολογιστεί η ενέργεια της δομής

Στις επόμενες δύο εικόνες παρουσιάζεται η εκτέλεση του προγράμματος χρησιμοποιώντας τον αλγόριθμο συντακτικής αναγνώρισης προτύπων, ορίζοντας ως τρόπο υπολογισμού της ενέργειας το πακέτο Vienna RNA στην πρώτη εκτέλεση και το πακέτο pkenergy στην δεύτερη αντίστοιχα.



Σχήμα 5.11: *Εκτέλεση με χρήση του αλγορίθμου συντακτικής αναγνώρισης προτύπων με χρήση του πακέτου ViennaRNA*



Σχήμα 5.12: *Εκτέλεση με χρήση του αλγορίθμου συντακτικής αναγνώρισης προτύπων με χρήση του πακέτου pkenergy*

6. Αξιολόγηση της επίδοσης του συστήματος

6.1 Σύγκριση εξόδων διαφόρων συστημάτων για μια συγκεκριμένη ακολουθία

Για την αξιολόγηση της απόδοσης του συστήματός μας, εισάχθηκε σε αυτό μια ακολουθία RNA που αντιστοιχεί σε έναν γνωστό ψευδοκόμβο τύπου K [32]. Στην εικόνα που ακολουθεί, παρουσιάζεται η πραγματική αναπαράσταση της ακολουθίας RNA σε μορφή dot_bracket, καθώς και οι αναπαραστάσεις dot_bracket που προέκυψαν τόσο από το δικό μας σύστημα όσο και από δύο άλλα γνωστά συστήματα, τα IPknot και Knotty.

Platform *RNA/DotBracket*

	AAGGGCGUCGUCGCCCCGAGUCGUAGCAGUUGACUACUGUUAUGU
Ground truth	..((((…[[[)))).….{{{{{{{{{.]]].}}}}}}}}}.
Knotify	..((.....[[[[[[[.....)){.]]]]]]]}
IPknot	..(((((….))))).……(((((((…..)))))))..
Knotty	..(((((….))))).((((([[[[[[[[.))))]]]]]]]]..

Σχήμα 6.1: *Πρόβλεψη dot-bracket αναπαράστασης για την δοσμένη ακολουθία*

Παρατηρούμε ότι το δικό μας σύστημα προέβλεψε επιτυχημένα κάποιους βασικούς δεσμούς για τον ψευδοκόμβο τύπου K, ενώ τα άλλα δύο συστήματα προέβλεψαν ψευδοκόμβους τύπου H στην καλύτερη περίπτωση, ή και καθόλου ψευδοκόμβους.

Εκτός από την έξοδο dot bracket των συστημάτων χρησιμοποιήθηκαν και άλλες μετρικές για να γίνει η σύγκριση του συστήματος μας με τα άλλα δύο. Οι μεταβλητές που χρησμοποιήθηκαν είναι οι εξής:

1) True positives (TP): Τα ζευγάρια βάσεων που προβλέφθηκε ότι υπήρχαν στη δομή του RNA και πράγματι υπήρχαν
2) False positives (FP): Τα ζευγάρια βάσεων που προβλέφθηκε ότι υπήρχαν στη δομή ενώ στην πραγματικότητα δεν υπήρχαν
3) True negatives (TN): Τα ζευγάρια βα΄σεων που προβλέφθηκε ότι δεν υπήρχαν στη δομή του RNA και δεν υπήρχαν στην πραγματικότητα
4) False negatives (FN): Τα ζευφάρια βάσεων που προβλέφθηκε ότι δεν υπήρχαν στη δομή του RNA ενώ στην πραγματικότητα υπήρχαν.

Οι 4 μετρικές λοιπόν με βάση τις οποίες έγινε η σύγκριση των συστημάτων είναι οι εξής:

1) Positive Predictive Value (PPV): ισούται με το πηλίκο $\frac{TP}{(TP+FP)}$
2) Recall: ισούται με το πηλίκο $\frac{TP}{(TP+FN)}$
3) F1-score: ισούται με $\frac{(2*PPV*Recall)}{(PPV+Recall)}$ (αρμονικός μέσος των PPV και Recall)
4) Mathews Correlation Coefficient (MCC): ισούται με το κλάσμα που έχει ως αριθμητή τη διαφορά του γινομένου TP με το TN με το γινόμενο του FP με το FN και ως παρανομάστή την ποσότητα:
   X= $\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}$

Platform	TP	TN	FP	FN	Precision	Recall	F1-score	MCC
knotify	17	11	3	14	0.85	0.5484	0.667	0.31
IPKnot	24	12	1	6	0.96	0.8	0.875	0.674
Knotty	28	7	5	3	0.8485	0.9032	0.875	0.515

Σχήμα 6.2: *Οι επιδόσεις σε θεμελιώδεις μετρικές ανά πλατφόρμα*

Παρακάτω παρουσιάζονται αποτελέσματα για τέσσερις επιπλέον ακολουθίες ώστε να βγούνε ευρύτερα συμπεράσματα. Παρατηρείται ότι το Knotify έχει σταθερά υψηλή την τιμή Precision.

	GCACCGGCUAACUCCGUGCCAGCAGCCGCGGUAAUACGGAGGGUGC
Ground truth	(((((((:::[[[[[[[::::::)))::::::]]]]]]]))))).
Knotify	.....(((...[[[[[.)))**{**.....**]**]]]]..**}**..
IPknot	(((((((...[[[[[[)))**.....]**]]]]]))))).
Knotty	(((((.....(((((((((.((....)))))...))))))))))).

Σχήμα 6.3: *Πρόβλεψη dot-bracket αναπαράστασης για την δοσμένη ακολουθία*

Platform	TP	TN	FP	FN	Precision	Recall	F1-score	MCC
knotify	14	14	3	14	0.8235	0.5	0.622	0.3234
IPKnot	25	15	3	4	0.8929	0.8621	0.877	0.689
Knotty	23	8	10	7	0.6969	0.7667	0.7304	0.2205

Σχήμα 6.4: *Οι επιδόσεις σε θεμελιώδεις μετρικές ανά πλατφόρμα*

	GCCGGGCGCGGUGGCGCGUGCCUGUAGUCCCAGCUACUCGGGAGGCU
Ground truth	(((:(((((((:[[[[:))))))):::::((((:]]]]:::))))))):
Knotify	(((([[[[[[...)))){.]]]]]]..................}...
IPknot	((((((((((.....)))))))[[[[[(((.]]]]]..))).))).
Knotty	(((([[[[[[[..))).]]]]]]]....((((([[[[....)))))]]]]

Σχήμα 6.5: *Πρόβλεψη dot-bracket αναπαράστασης για την δοσμένη ακολουθία*

Platform	TP	TN	FP	FN	Precision	Recall	F1-score	MCC
knotify	14	10	6	16	0.7	0.4667	0.56	0.1728
IPKnot	28	6	7	5	0.8	0.8485	0.8235	0.6024
Knotty	25	9	9	2	0.7353	0.9259	0.8182	0.6628

Σχήμα 6.6: *Οι επιδόσεις σε θεμελιώδεις μετρικές ανά πλατφόρμα*

	ACCCUACUGUGCUAACCGAACCAGAUAACGGUACAGUAGGGGUAAAUUCU CCGCAUUCGGUGCGGAAAAAAAA
Ground truth	:((((((((((((:[[[[[[:::::::::))))))))))))):::::((:(((((:]]]]]])))))):))::::
Knotify	..([[[[[[[[[[.....){{.........]]]]]]]]]..............}}..............
IPknot	.((((((((((((..............))))))))))))........(((((((...))))))).....
Knotty	.((((((((((((..[[[[[.........))))))))))))........((((((]]]]])))))).....

Σχήμα 6.7: *Πρόβλεψη dot-bracket αναπαράστασης για την δοσμένη ακολουθία*

Platform	TP	TN	FP	FN	Precision	Recall	F1-score	MCC
knotify	23	23	1	26	0.9583	0.4694	0.6296	0.5317
IPKnot	36	22	2	13	0.9474	0.7347	0.8276	0.7481
Knotty	45	22	1	5	0.9783	0.9	0.9375	0.8996

Σχήμα 6.8: *Οι επιδόσεις σε θεμελιώδεις μετρικές ανά πλατφόρμα*

	UUUAAACGAGUCCGGGGCUCUAGUGCCGCUCGACUAGAGCCCUGUAACA GUUAUGGACCACGAGCAGUCCAUGUA
Ground truth	::::::::::::((((((((((((:::[[[[[))))))))))))::::::::((((((::]]]]]:)))))))::
Knotify	.......(((..[[[[[[[[[[[.....)))){.]]]]]]]]]].....}.....
IPknot	.....((.....((((((((((((........))))))))))))).....)).((((((........))))))..
Knotty	....(((.....((((((((((((...[[[[[))))))))))))).....)))((((((..]]]]].))))))..

Σχήμα 6.9: *Πρόβλεψη dot-bracket αναπαράστασης για την δοσμένη ακολουθία*

Platform	TP	TN	FP	FN	Precision	Recall	F1-score	MCC
knotify	24	24	6	21	0.8	0.5333	0.64	0.4134
IPKnot	36	24	4	11	0.9	0.7659	0.8286	0.7368
Knotty	46	22	6	1	0.8846	0.9787	0.9296	0.8794

Σχήμα 6.10: *Οι επιδόσεις σε θεμελιώδεις μετρικές ανά πλατφόρμα*

Η πλατφόρμα Knotify καταγράφει αξιοσημείωτη επίδοση στον τομέα της ακρίβειας σωστής πρόβλεψης (Precision) με αθροιστική τιμή 0.8288, διατηρώντας έναν ιδιαίτερα ικανοποιητικό ισοζύγιο σωστών προβλέψεων σε σχέση με τις λανθασμένες (FP). Παρά την ελαφρώς χαμηλότερη επίδοση στην ανάκληση (Recall) συγκριτικά με τις άλλες πλατφόρμες, αυτό μπορεί να αποδοθεί στην ισορροπία που επιδιώκει η Knotify. Παρακάτω παρουσιάζονται αθροιστικά οι μετρικές για όλες τις ακολουθίες που ελέγχθηκαν στα προηγούμενα παραδείγματα:

Platform	TP	TN	FP	FN	Precision	Recall	F1-score	MCC
knotify	92	82	19	91	0.8288	0.5027	0.6271	0.4655
IPKnot	149	79	17	39	0.8975	0.7927	0.8417	0.7243
Knotty	167	68	31	18	0.8436	0.9028	0.8723	0.7347

Σχήμα 6.10: *Οι αθροιστικές επιδόσεις σε θεμελιώδεις μετρικές ανά πλατφόρμα*

7. Συμπεράσματα και μελλοντικές επεκτάσεις

Η μελέτη και ανάλυση του RNA είναι εξαιρετικά σημαντική, όπως αποδεικνύεται τόσο από την πρόσφατη εμφάνιση του Covid-19, όσο και από την ανάγκη κατανόησης διάφορων φυσικών φαινομένων. Στο πλαίσιο αυτής της έρευνας, παρουσιάστηκε μια ακριβής και αποδοτική μεθοδολογία για την πρόβλεψη μιας σπάνιας μορφής ψευδοκόμβου, του ψευδοκόμβου τύπου Κ. Η μέθοδος αυτή βασίστηκε στον αλγόριθμο του Earley, παράγοντας όλα τα πιθανά συντακτικά δέντρα, τα οποία αντιπροσώπευαν πιθανές θέσεις της δομής στην RNA ακολουθία. Επιπλέον, αναπτύχθηκε ένας άπληστος αλγόριθμος, ο οποίος συνέβαλε σημαντικά στη βελτίωση της απόδοσης του συστήματος πρόβλεψης. Για την επιλογή της βέλτιστης δομής, χρησιμοποιήθηκαν κριτήρια όπως η μεγιστοποίηση των ζευγαριών βάσεων και η ελαχιστοποίηση της ελεύθερης ενέργειας αναδίπλωσης της δομής. Όσον αφορά μελλοντικές επεκτάσεις, η ίδια διαδικασία πρόβλεψης θα μπορούσε να εφαρμοστεί και για άλλες δομές, όπως ο ψευδοκόμβος τύπου Μ.

Βιβλιογραφία

[1] Hanis, G. (n.d.). *RNA*. GitHub. https://github.com/georgehanis/rna

[2] Wikipedia contributors. (n.d.). *RNA*. Wikipedia. https://el.wikipedia.org/wiki/RNA

[3] BYJU'S. (n.d.). *Difference between DNA and RNA*. https://byjus.com/biology/difference-between-dna-and-rna/

[4] Study Mind. (n.d.). *Messenger RNA*. https://studymind.co.uk/notes/messenger-rna/

[5] Pearson. (n.d.). *Which statement regarding the structure and function of tRNA is false?* https://www.pearson.com/channels/biology/asset/5f839600/which-statement-regarding-the-structure-and-function-of-trna-is-false

[6] Nature Education. (n.d.). *Hairpin loop (mRNA)*. https://www.nature.com/scitable/definition/hairpin-loop-mrna-314/#:~:text=A%20hairpin%20loop%20is%20an,loop%20or%20a%20U%2Dshape

[7] Sundaram, S. (2013). Different types of RNA secondary structures [Figure]. ResearchGate. https://www.researchgate.net/figure/Different-types-of-RNA-Secondary-Structures-1-First-structure-represents-a-stem-with_fig1_261030242

[8] Alvarez, D. E., & Gamarnik, A. V. (2018). Known viral RNA structures from stem loops to complex tRNA-like structures [Figure]. ResearchGate. https://www.researchgate.net/figure/Known-viral-RNA-structures-from-stem-loops-to-complex-tRNA-like-structures-A-The_fig1_322244039

[9] Turner, D. H. (2004). NNDB example: RNA secondary structure. University of Rochester. https://rna.urmc.rochester.edu/NNDB/turner04/mb-example.html

[10] Wikipedia contributors. (n.d.). *Pseudoknot*. Wikipedia. https://en.wikipedia.org/wiki/Pseudoknot

[11] Brierley, I., & Dos Ramos, F. J. (2006). Programmed ribosomal frameshifting in HIV-1 and the SARS-CoV. *Nature Reviews Microbiology, 4*(11), 766–774. https://www.nature.com/articles/nrmicro1704

[12] Ren, J., & Rastegari, B. (2015). Top: Four basic types of pseudoknots considered in gfold program. Bottom: The conflict graph representation [Figure].

ResearchGate. https://www.researchgate.net/figure/Top-Four-basic-types-of-pseudoknots-considered-in-gfold-program-Bottom-The-conflict_fig5_282434729

[13] Deng, H., Gao, Y., & Yin, Z. (2021). RNA secondary structure prediction: Progress and perspective. *Computational and Mathematical Methods in Medicine, 5*(1), 14. https://www.mdpi.com/2409-9279/5/1/14

[14] Staple, D. W., & Butcher, S. E. (1999). Pseudoknots: RNA structures with diverse functions. *Trends in Biochemical Sciences, 24*(9), 240–245. https://pubmed.ncbi.nlm.nih.gov/9925784/

[15] Pleij, C. W., Rietveld, K., & Bosch, L. (1985). A new principle of RNA folding based on pseudoknotting. *Nucleic Acids Research, 9*(1), 171–191. https://pubmed.ncbi.nlm.nih.gov/6161375/

[16] Theimer, C. A., & Giedroc, D. P. (2000). Equilibrium unfolding pathway of an H-type pseudoknot RNA. *Journal of Molecular Biology, 297*(5), 1205–1220. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC326673/

[17] Ren, J., & Rastegari, B. (2011). A dynamic programming algorithm for RNA pseudoknot prediction based on multiple context-free grammars. *Algorithms for Molecular Biology, 6*, 26. https://almob.biomedcentral.com/articles/10.1186/1748-7188-6-26

[18] Kaplan, E. L., & Meier, P. (1958). Nonparametric estimation from incomplete observations. *Journal of the American Statistical Association, 53*(282), 457–481. https://doi.org/10.2307/2280232

[19] Hansen, T. M., Reihani, S. N. S., Oddershede, L. B., & Sørensen, M. A. (2007). Correlation between mechanical strength of messenger RNA pseudoknots and ribosomal frameshifting. *Proceedings of the National Academy of Sciences of the United States of America, 104*(14), 5830–5835. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC169020/

[20] Ren, J., & Rastegari, B. (2011). A dynamic programming algorithm for pseudoknotted RNA secondary structure prediction. *BMC Bioinformatics, 12*, 103. https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-12-103

[21] Staple, D. W., & Butcher, S. E. (2005). Pseudoknots: RNA structures with diverse functions. *Nucleic Acids Research, 33*(8), 2455–2467. https://pubmed.ncbi.nlm.nih.gov/15448187/

[22] Danaee, P., Rouches, M., Wiley, M., Liu, B., Engelhardt, J., & Wang, X. (2018). Improving the accuracy of RNA secondary structure prediction using deep learning. *Nucleic Acids Research, 46*(16), 8105–8113. https://pubmed.ncbi.nlm.nih.gov/30886627/

[23] Sparks Lab. (n.d.). *SPOT-RNA: RNA secondary structure prediction*. https://sparks-lab.org/server/spot-rna/

[24] Bharadwaj, K. (n.d.). *DeepRNA: Deep learning models for RNA secondary structure prediction*. GitHub. https://github.com/kangkanbharadwaj/DeepRNA

[25] Wikipedia contributors. (n.d.). *Syntactic pattern recognition*. Wikipedia. https://en.wikipedia.org/wiki/Syntactic_pattern_recognition

[26] Wikipedia contributors. (n.d.). *Context-free grammar*. Wikipedia. https://en.wikipedia.org/wiki/Context-free_grammar

[27] Wikipedia contributors. (n.d.). *Abstract syntax tree*. Wikipedia. https://en.wikipedia.org/wiki/Abstract_syntax_tree

[28] Wikipedia contributors. (n.d.). *CYK algorithm*. Wikipedia. https://en.wikipedia.org/wiki/CYK_algorithm

[29] Wikipedia contributors. (n.d.). *Earley parser*. Wikipedia. https://en.wikipedia.org/wiki/Earley_parser

[30] Makarov, V. (n.d.). *YAEP: Yet Another Earley Parser*. GitHub. https://github.com/vnmakarov/yaep/blob/master/doc/yaep.pdf

[31] Christos Andrikos, Evangelos Makris, Angelos Kolaitis, Georgios Rassias, Christos Pavlatos και Panayiotis Tsanakas. Knotify: An Efficient Parallel Platform for RNA Pseudoknot Prediction Using Syntactic Pattern Recognition. Methods and Protocols, 5(1), 2022.

[32] Van Batenburg, E. (2000, June 16). *PseudoBase entry: PKB178 - Ni_VS pseudoknot*. PseudoBase. Retrieved November 12, 2024, from https://ekevanbatenburg.nl/PKBASE/PKB00178.HTML