ΠΑΝΕΠΙΣΤΗΜΙΟ ΣΤΕΡΕΑΣ ΕΛΛΑΔΑΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗΝ ΒΙΟΙΑΤΡΙΚΗ

ΑΝΑΛΥΣΗ ΠΡΩΤΕΪΝΗΣ ΣΕ ΠΛΕΓΜΑ

ΕΥΦΡΟΣΥΝΗ ΔΟΥΤΣΗ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ
ΕΠΙΒΛΕΠΩΝ
ΕΥΡΙΠΙΔΗΣ ΜΑΡΚΟΥ
ΛΕΚΤΟΡΑΣ

ΛΑΜΙΑ 2012

Ευχαριστίες

Με το τέλος των προπτυχιακών μου σπουδών και της παρούσας διετούς έρευνας, θα ήταν φοβερή αμέλεια εκ μέρους μου αν δεν ευχαριστούσα όλους τους ανθρώπους που με στήριξαν στην προσπάθειά μου αυτή.

Θα ξεκινήσω με τον επιβλέποντα καθηγητή, τον λέκτορα κ. Ευριπίδη Μάρκου, που τον ευχαριστώ πρώτα από όλα για την εμπιστοσύνη του, την ανεξάντλητη υπομονή του και κυρίως για την πολύτιμη βοήθεια και καθοδήγησή του. Θέλω επίσης να ευχαριστήσω τα υπόλοιπα δύο μέλη της εξεταστικής επιτροπής, τους επίκουρους καθηγητές κ. Παντελεήμωνα Μπάγκο και κ. Βασίλειο Πλαγιανάκο για τις εύστοχες επισημάνσεις τους κατά τη διάρκεια της εκπόνησης αυτής της εργασίας. Δεν θα μπορούσα επίσης να μην αναφερθώ στην κα. Μαρία Αδάμ, επίκουρη καθηγήτρια, την οποία ευχαριστώ για το αμείωτο ενδιαφέρον της αλλά και τις σημαντικές υποδείξεις της.

Τέλος, ευχαριστώ την οικογένεια μου που ήταν στο πλάι μου ανιδιοτελώς όλα αυτά τα χρόνια και στην οποία αφιερώνω την εργασία αυτή.

Περίληψη

Ένα πολύ γνωστό πρόβλημα της βιολογίας είναι ο υπολογισμός (πρόβλεψη) της τρισδιάστατης δομής μίας πρωτεΐνης, όταν είναι γνωστή μόνο η ακολουθία των αμινοξέων που την αποτελούν. Το πρόβλημα αυτό (γνωστό ως πρόβλημα Αναδίπλωσης μιας Πρωτεΐνης - Protein Folding Problem) φαίνεται να είναι πολύ δύσκολο να επιλυθεί ακόμα και σε απλοποιημένα μοντέλα. Επιπλέον, το πρόβλημα παραμένει δύσκολο ακόμα και όταν αναζητούμε την βέλτιστη αναδίπλωση μίας πρωτεΐνης σε δισδιάστατα πλέγματα (lattices).

Στην εργασία αυτή, μελετήσαμε το παρακάτω σχετικό πρόβλημα: Δεδομένης της τρισδιάστατης αναδίπλωσης μίας πρωτεΐνης (είναι γνωστές οι συντεταγμένες των αμινοξέων που την αποτελούν), ψάχνουμε να βρούμε την πλησιέστερη αναδίπλωση της αλυσίδας των αμινοξέων της πρωτεΐνης πάνω σε lattice. Το πρόβλημα αυτό είναι γνωστό με την ονομασία "Ανάλυση Πρωτεΐνης σε Πλέγμα" (Protein Chain Fitting Lattice (PCLF)). Η λύση στο πρόβλημα αυτό ουσιαστικά θα καθορίσει ποιο πλέγμα πρέπει να προτιμήσουμε στο Protein Folding Problem. Πρόσφατα αποδείχθηκε ότι το PCLF πρόβλημα είναι NP-complete για συγκεκριμένα κυβικά lattices. Για το PCLF πρόβλημα έχουν προταθεί πολλοί εκθετικού χρόνου αλλά και προσεγγιστικοί αλγόριθμοι, οι οποίοι όμως δεν δίνουν καμία εγγύηση για την ποιότητα των λύσεων που επιστρέφουν. Στην εργασία αυτή σχεδιάσαμε και υλοποιήσαμε έναν αλγόριθμο που παίρνει ως είσοδο την τρισδιάστατη δομή μίας πρωτεΐνης μαζί με ένα lattice και επιστρέφει τον ορισμό ενός προβλήματος Ακέραιου Προγραμματισμού του οποίου η λύση θα δώσει μία βέλτιστη αναδίπλωση της πρωτεΐνης πάνω στο δεδομένο lattice. Στη συνέχεια λύσαμε το πρόβλημα αυτό χρησιμοποιώντας το πρόγραμμα ILOG CPLEX, που λύνει προβλήματα Ακέραιου Προγραμματισμού και του οποίου τα πνευματικά δικαιώματα ανήκουν στην εταιρία IBM. Στο τέλος, εφαρμόσαμε τις παραπάνω τεχνικές σε συγκεκριμένες ομάδες πρωτεϊνών (οι οποίες παρουσιάζουν κοινά μακροσκοπικά χαρακτηριστικά) και σε διαφορετικά lattices. Τα πρώτα αποτελέσματα έδειξαν ότι πρωτεΐνες που ανήκουν στις ίδιες ομάδες τείνουν να αναδιπλώνονται το ίδιο καλά σε ένα συγκεκριμένο lattice.

Ο αλγόριθμός μας μετά από χρόνο $O(n^4)$, δημιουργεί το μετασχηματισμένο PCLF πρόβλημα Ακέραιου Προγραμματισμού, το οποίο αποτελείται από $O(n^4)$ μεταβλητές και περιορισμούς. Παρόλο που καταβάλαμε προσπάθεια να μειώσουμε το μέγεθος του (μετασχηματισμένου) προβλήματος Ακέραιου Προγραμματισμού, η οποιαδήποτε περαιτέρω μείωση του μεγέθους θα ήταν κάτι πραγματικά επιθυμητό, αφού θα μας έδινε τη δυνατότητα να μελετήσουμε γρήγορα μεγαλύτερου μήκους αλυσίδες αμινοξέων.

Τέλος, η τεχνική που ακολουθήσαμε μπορεί να επεκταθεί πολύ εύκολα έτσι ώστε να πάρουμε αποτέλεσματα για βέλτιστες αναδιπλώσεις χρησιμοποιώντας διαφορετικά κριτήρια από αυτά που χρησιμοποιήσαμε εδώ.

Λέξεις - Κλειδιά: Πρόβλημα Αναδίπλωσης μιας Πρωτεΐνης, πρωτεΐνη, πλέγμα, Ανάλυση πρωτεΐνης σε Πλέγμα, NP-complete, Ακέραιος Προγραμματισμός.

Abstract

An important problem in biology is to computationally determine (predict) the structure of a protein given its sequence of amino acids. This problem (known as Protein Folding) appears to be extremely hard even when very simplified models are considered. Moreover the problem remains hard even if we are interested to find an optimal folding in a 2D lattice.

We consider here the following closely related problem: given a 3D folding of a protein chain (coordinates of its atoms), find the closest lattice representation of this folding. This problem is known as Protein Chain Lattice Fitting (PCLF) problem. A solution to this problem would indicate which lattice to test in the original Protein Folding problem. It has been recently shown that the PCLF problem is NP-complete for specific cubic lattices. A number of exponential-time and approximation algorithms (but without any guarantee of the quality of returned solutions) have been proposed for this problem. In this work we first design and implement an algorithm which takes as input the 3D structure of a protein along with a lattice and returns the definition of an Integer Programming problem whose solution would give an optimal folding of the protein to the given lattice. We next solve this problem using the ILOG CPLEX software which is a software developed by IBM and solves Integer Programming problems. Finally we apply our techniques to specific protein data sets with similar macroscopic properties together with a number of different lattices. Our first experiments show that proteins which belong to the same group tend to fold optimally on the same lattice.

Our algorithm outputs after $O(n^4)$ time an Integer Programming problem consisted of $O(n^4)$ variables and constraints. Although we made an effort to decrease the size of the (transformed) input of the Integer Programming problem, a further decreasing of the size would be desirable as it would enable us to quickly test longer protein chains.

Finally, our techniques can be easily extended to give us results with respect to closest lattice representation under different criteria.

Keywords: Protein Folding Problem, protein, lattice, Protein Chain Lattice Fitting Problem, NP-complete, Integer Programming.

Περιεχόμενα

Κεφάλαιο 1

ΕΙΣΑΓΩΓΗ

1.1 ΕΙΣΑΓΩΓΗ ΣΤΟ ΠΡΟΒΛΗΜΑ PCLF

Ένα πολύ σημαντικό πρόβλημα στη βιολογία είναι η πρόβλεψη της τρισδιάστατης μορφής της πρωτεΐνης όταν είναι γνωστή η ακολουθία των αμινοξέων που την αποτελούν. Με άλλα λόγια, πόσο γρήγορα μπορούμε να υπολογίσουμε το σχήμα που παίρνει η πρωτεΐνη στο χώρο αν ξέρουμε την ακολουθία των αμινοξέων που την αποτελούν. Το παραπάνω πρόβλημα είναι γνωστό στην επιστημονική κοινότητα με διάφορες ονομασίες από τις οποίες η πιο διαδεδομένη είναι "Πρόβλημα Αναδίπλωσης της Πρωτεΐνης" (Protein Folding Problem).

Η λύση του προβλήματος αυτού είναι εξαιρετικά δύσκολη ακόμα και αν περιοριστούμε σε απλά μοντέλα, όπως το υδροφοβικό μοντέλο (HP-Model) που προτάθηκε το 1990 από τον Dill. Στο μοντέλο αυτό, η απεικόνιση της πρωτεΐνης γίνεται πάνω σε δισδιάστατο πλέγμα (lattice) με μοναδικό κριτήριο τις υδροφοβικές αλληλεπιδράσεις μεταξύ των αμινοξέων που γειτονεύουν στο πλέγμα. Για αυτό το απλοποιημένο μοντέλο το πρόβλημα αποδείχθηκε ότι ανήκει στην κατηγορία εκείνων των προβλημάτων που ονομάζονται NP-πλήρη (NP-complete). Ειδικότερα αποδείχθηκε ότι δεδομένης της ακολουθίας αμινοξέων μιας πρωτεΐνης και ενός δισδιάστατου ή τρισδιάστατου πλέγματος (lattice), το πρόβλημα της εύρεσης κάποιας αναδίπλωσης της πρωτεΐνης πάνω στο πλέγμα έτσι ώστε να μεγιστοποιείται το πλήθος των μη γειτονικών-διαδοχικών υδρόφοβων αμινοξέων, είναι NP-πλήρες. Η κατηγοριοποίηση του προβλήματος στα NP-πλήρη σημαίνει πως αν υπάρχει πολυωνυμικού χρόνου αλγόριθμος που υπολογίζει τη βέλτιστη απεικόνιση της πρωτεΐνης σε ένα πλέγμα, τότε $P = NP$. Επιπρόσθετα έχει αποδειχθεί ότι το πρόβλημα είναι APX-hard, δηλαδή αν υπάρχει πολυωνυμικού χρόνου αλγόριθμος που πετυχαίνει μία λύση "πολύ κοντά" στη βέλτιστη τότε $P = NP$. Ακόμα όμως και αν βρίσκαμε τη βέλτιστη αναδίπλωση μιας πρωτεΐνης σε ένα πλέγμα αυτή μπορεί να απείχε σημαντικά από την αναδίπλωσή της στο χώρο, λόγω των περιορισμών που

θέτει το πλέγμα.

Συνεπώς, ένα σχετικό πρόβλημα με το οποίο ασχολούνται πολλοί ερευνητές είναι η εύρεση ενός "καλού" πλέγματος στο οποίο η αναπαράσταση του διπλώματος μιας πρωτεΐνης μπορεί να είναι κοντά στην πραγματική τρισδιάστατη δομή της. Για να αξιολογήσουμε πόσο "καλό" είναι ένα πλέγμα για μία ομάδα πρωτεϊνών συνήθως ακολουθούμε την παρακάτω διαδικασία:

- Αρχικά διαλέγουμε ένα σύνολο πρωτεΐνων, για τις οποίες είναι γνωστές οι συντεταγμένες των αμινοξέων που τις αποτελούν και ένα σύνολο τρισδιάστατων πλεγμάτων (lattices) πάνω στα οποία θα τις απεικονίσουμε.

- Κάθε μία πρωτεΐνη την τοποθετούμε σε όλα τα πλέγματα ψάχνοντας να βρούμε σε ποιο απ' όλα τα πλέγματα το σχήμα της πρωτεΐνης είναι πιο κοντά στο σχήμα που έχει στο χώρο. Η αξιολόγηση της απεικόνισης γίνεται συνήθως με βάση δύο κριτήρια, τη μέση τετραγωνική απόκλιση συντεταγμένων (coordinate root mean squared deviation-CRMS) ή τη μέση τετραγωνική απόκλιση αποστάσεων (distance mean squared deviation-DRMS). Η καλύτερη απεικόνιση είναι αυτή που δίνει την ελάχιστη τιμή στο CRMS ή στο DRMS.

- Το lattice που δίνει κατά μέσο όρο το ελάχιστο CRMS ή DRMS για το σύνολο των πρωτεϊνών αυτών επιλέγεται ως το βέλτιστο.

Δεδομένων μιας πρωτεΐνης στο χώρο και ενός lattice, το κρίσιμο βήμα στην παραπάνω διαδικασία είναι η εύρεση του σχήματος της πρωτεΐνης στο lattice που ελαχιστοποιεί το κριτήριο που επιλέγεται (π.χ. CRMS, DRMS).

Το παραπάνω πρόβλημα αναφέρεται στη βιβλιογραφία με τις ονομασίες: "Protein Chain Lattice Fitting Problem (PCLF)" (την οποία θα χρησιμοποιούμε από εδώ και στο εξής), "the discretization of a protein backbone", "modeling protein structures on lattices", "lattice approximation of 3D structure of a chain molecule", "discrete state model fitting to X-ray structures" κ.λ.π.

Πρόσφατα αποδείχτηκε ότι το πρόβλημα "Protein Chain Lattice Fitting Problem (PCLF)" ανήκει στα NP-πλήρη προβλήματα για απόκλιση CRMS και τρισδάστατα πλέγματα με πλευρά 3.8Å. Για το λόγο αυτό, οι περισσότεροι αλγόριθμοι, οι οποίοι αναζητούν τη βέλτιστη λύση και έχουν προταθεί μέχρι τώρα, είναι εκθετικοί ή ευριστικοί. Ακόμα, έχουν χρησιμοποιηθεί και προσεγγιστικοί αλγόριθμοι, οι οποίοι επιστρέφουν γρήγορα κάποια λύση χωρίς όμως να δίνουν καμία εγγύηση για το πόσο καλή είναι ως προς τη βέλτιστη (με άλλα λόγια δεν συνοδεύονται από κάποια απόδειξη για το μέγιστο σφάλμα της προσεγγιστικής λύσης).

Στην συνέχεια θα αναλύσουμε κάποιες βασικές έννοιες που βοηθάνε στην κατανόηση του προβλήματος αλλά και των εργαλείων που χρησιμοποιήσαμε στην παρούσα πτυχιακή για την επίλυσή του. Οι έννοιες αυτές είναι:

- Η δομή των πρωτεϊνών
- Το πλέγμα (lattice)
- Τι είναι πολυπλοκότητα ενός προβλήματος
- Τι σημαίνει όταν ένα πρόβλημα ανήκει στα NP-πλήρη προβλήματα
- Τι είναι γραμμικός προγραμματισμός και ποια είναι η δομή που απαιτείται να έχει ένα πρόβλημα γραμμικού προγραμματισμού
- Τι είναι το CRMS

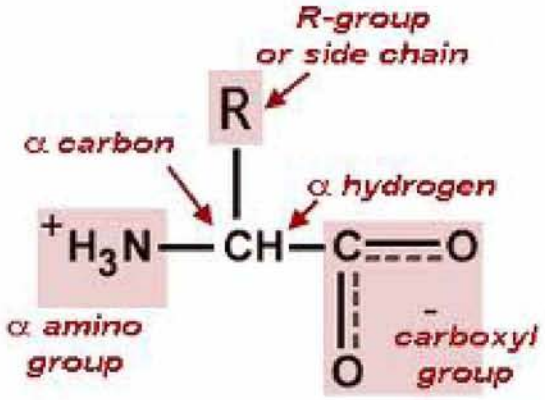
1.2 ΕΙΣΑΓΩΓΗ ΣΤΙΣ ΠΡΩΤΕΪΝΕΣ

1.2.1 Σύντομη Περιγραφή της Σύνθεσης των Πρωτεϊνών

Όπως αναφέραμε στο εισαγωγικό σημείωμα, το "Protein Chain Lattice Fitting Problem (PCLF)" πραγματεύεται την αναζήτηση της βέλτιστης τρισδιάστατης απεικόνισης μίας πρωτεΐνης σε ένα πλέγμα αν γνωρίζουμε τις συντεταγμένες (στο χώρο) της ακολουθίας των αμινοξέων από τα οποία αποτελείται. Τι ακριβώς είναι τα αμινοξέα; Ποιος είναι ο δομικός τους ρόλος στην πρωτεΐνη; Πώς αναδιπλώνεται στο χώρο μία πρωτεΐνη σύμφωνα με τους νόμους της φύσης; Όλα αυτά είναι ερωτήματα που θα απαντήσουμε στη συνέχεια κάνοντας μία μικρή εισαγωγή αρχικά στη χημεία των μορίων.

Μονομερές μόριο, στη χημεία, ονομάζουμε ένα μικρό μόριο που ενώνεται με τη βοήθεια κάποιου χημικού δεσμού με ένα άλλο μονομερές μόριο με σκοπό να δημιουργήσουν ένα πολυμερές μόριο. Πιο αναλυτικά, ο χημικός αυτός δεσμός δημιουργείται κατά τη διάρκεια μιας χημικής διαδικασίας που λέγεται "αντίδραση συμπύκνωσης". Κατά τη διαδικασία της ένωσης δύο μονομερών, το ένα μονομερές χάνει ένα άτομο υδρογόνου (-H) και το άλλο μία υδροξυλομάδα (-OH) δηλαδή, αφαιρείται ένα μόριο νερού ($-H_20$). Το αποτέλεσμα τέτοιων επαναλαμβανόμενων δομών είναι η δημιουργία πολυμερών μακρομορίων. *Πολυμερή* μακρομόρια στη βιολογία θεωρούμε τα νουκλεϊκά οξέα, τους πολυσακχαρίτες, τα λιπίδια και τις πρωτεΐνες με τις οποίες θα ασχοληθούμε στην παρούσα πτυχιακή εργασία.

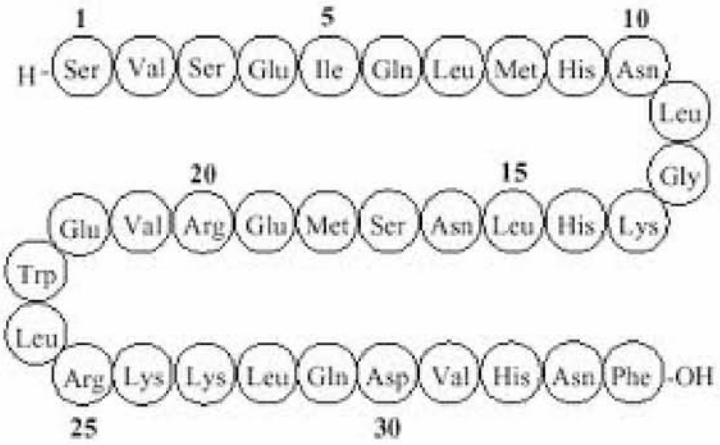
Τα μονομερή των πρωτεϊνών είναι τα **αμινοξέα** (π.χ. [1]). Η σύνθεση μιας συγκεκριμένης πρωτεΐνης απαιτεί μία συγκεκριμένη ακολουθία αμινοξέων. Υπάρχουν 20 αμινοξέα που συμμετέχουν στην πρωτεϊνοσύνθεση και διαφέρουν δομικά μεταξύ τους. Πιο συγκεκριμένα, όπως περιγράφει και το Σχήμα 1.1, κάθε αμινοξύ διακρίνεται από δύο τμήματα ένα "σταθερό" και ένα "μεταβλητό", τα οποία ενώνονται σε ένα άτομο a-άνθρακα (Ca, Βλέπε στο Σχήμα 1.1: α-carbon). Το σταθερό τμήμα αποτελείται από ένα άτομο υδρογόνου (-H) (Βλέπε στο Σχήμα 1.1: α-hydrogen), μία αμινομάδα (-NH2) (Βλέπε στο Σχήμα 1.1: α-amino group) και μία όξινη καρβοξυλομάδα (-RCOOH) (Βλέπε στο Σχήμα 1.1: carboxyl group). Το "μεταβλητό" αποτελείται από την "πλευρική ομάδα R" (Βλέπε στο Σχήμα 1.1: R-group or side chain). Τα αμινοξέα διακρίνονται μεταξύ τους από τις διαφορετικές πλευρικές ομάδες R που διαθέτουν.



Σχήμα 1.1: Δομή Αμινοξέος

Τα αμινοξέα συνδέονται μεταξύ τους με *πεπτιδικούς δεσμούς* . Οι πεπτιδικοί δεσμοί είναι ισχυροί ομοιοπολικοί δεσμοί και σχηματίζονται σύμφωνα με την αντίδραση συμπύκνωσης που αναφέραμε παραπάνω. Με άλλα λόγια, κατά τη σύνδεση δύο αμινοξέων αφαιρείται ένα μόριο νερού (H_20) που παράγεται όταν ενώνεται η καρβοξυλομάδα του ενός αμινοξέος με την αμινομάδα του άλλου. Δύο αμινοξέα που ενώνονται μεταξύ τους με πεπτιδικό δεσμό λέγονται *διαδοχικά*, διαφορετικά λέγονται *μη-διαδοχικά*. Όταν ενωθούν δύο αμινοξέα σχηματίζεται ένα *διπεπτίδιο*, όταν ενωθούν τρία σχηματίζεται ένα *τριπεπτίδιο* και όταν ενωθούν πολλά, ένα *πολυπεπτίδιο* (εναλλακτικά ένα πολυπεπτίδιο ονομάζεται και πολυπεπτιδική αλυσίδα, πολυπεπτιδικός σκελετός ή ακολουθία αμινοξέων). Μία τέτοια ακολουθία απεικονίζεται στο Σχήμα 1.2.

Όπως φαίνεται και στο Σχήμα 1.3, η δομή των πρωτεϊνών εξετάζεται σε διάφορα επίπεδα που είναι τα παρακάτω: (π.χ. [9] [1])



Σχήμα 1.2: Ακολουθία Αμινοξέων

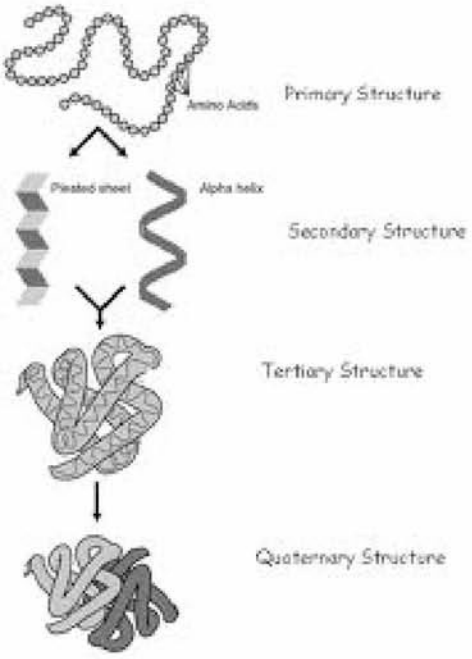
- Το πρώτο επίπεδο ονομάζεται *"πρωτοταγής δομή"* και περιλαμβάνει τη διαδοχή των αμινοξέων στην πολυπεπτιδική αλυσίδα

- Το δεύτερο επίπεδο ονομάζεται *"δευτεροταγής δομή"* και περιλαμβάνει τις σχέσεις των διαδοχικών αμινοξέων στο χώρο

- Το τρίτο επίπεδο ονομάζεται *"τριτοταγής δομή"* και περιλαμβάνει την αναδίπλωση της πολυπεπτιδικής αλυσίδας

- Και τέλος το τέταρτο επίπεδο ονομάζεται *"τεταρτοταγής δομή"* που περιλαμβάνει το επίπεδο των σχέσεων δύο ή περισσότερων πολυπεπτιδικών αλυσίδων.

Ένας πολυπεπτιδικός σκελετός μπορεί να αναδιπλωθεί με διάφορους τρόπους. Στην αναδίπλωση αυτή σημαντικό ρόλο παίζουν διάφορα είδη ασθενών μη ομοιοπολικών δεσμών, που σχηματίζονται τόσο μεταξύ μη διαδοχικών ατόμων του πεπτιδικού σκελετού όσο και μεταξύ των πλευρικών αλυσίδων των αμινοξέων. Οι ασθενείς αυτοί δεσμοί μπορεί να είναι δεσμοί υδρογόνου, ιοντικοί δεσμοί, έλξεις van der Waals και οι υδροφοβικές αλληλεπιδράσεις. Λόγω του ότι οι μη ομοιοπολικοί δεσμοί είναι ασθενείς σε σχέση με τους ομοιοπολικούς δεσμούς χρειάζονται πολλοί μη ομοιοπολικοί δεσμοί για να κρατηθούν δύο περιοχές μιας αναδιπλωμένης πολυπεπτιδικής αλυσίδας ενωμένες. Όσο περισσότεροι μη ομοιοπολικοί δεσμοί σχηματίζονται κατά την αναδίπλωση μίας πρωτεΐνης τόσο μεγαλύτερη σταθερότητα αποκτά το μόριό της. Μία πρωτεΐνη αναφέρεται σαν σταθερή όταν η *ελεύθερη ενέργεια* του μορίου είναι ελάχιστη (π.χ. [1] [9].

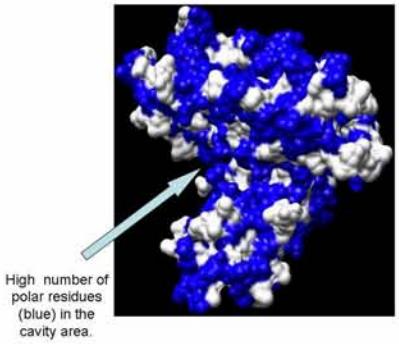


Σχήμα 1.3: Στάδια Αναδίπλωσης των Πρωτεϊνών

1.2.2 Αναδίπλωση Πρωτεΐνης

Όπως αναφέρθηκε ήδη στο Υποκεφάλαιο (1.2.1), η πρωτεϊνική αναδίπλωση καθορίζεται από την αλληλουχία των αμινοξέων. Από ένα σύνολο ασθενών δυνάμεων που παίζουν ρόλο στην αναδίπλωση αυτή, οι υδροφοβικές αλληλεπιδράσεις είναι αυτές που συναντάμε συχνότερα, περίπου στο 80% των πρωτεϊνικών αναδιπλώσεων. Στις αλληλεπιδράσεις αυτές, τα υδρόφοβα αμινοξέα ή αν θέλουμε να είμαστε πιο ακριβείς τα αμινοξέα που αποτελούνται από μη πολικές (υδρόφοβες) πλευρικές ομάδες, όταν βρεθούν σε υδάτινο περιβάλλον συνωθούνται ώστε να αποφευχθεί η διάσπασή τους από τα μόρια του νερού. Συνεπώς, ένας σημαντικός παράγοντας που κατευθύνει το δίπλωμα ή πτύχωση κάθε πρωτεΐνης είναι η κατανομή των *πολικών (υδρόφιλων)* και των *μη πολικών (υδρόφοβων)* αμινοξέων της.

Η τελική διαμόρφωση που υιοθετεί μία πολυπεπτιδική αλυσίδα καθορίζεται από ενεργειακές παραμέτρους και γενικά είναι εκείνη, στην οποία η ελεύθερη ενέργεια ελαχιστοποιείται. Κατά την αναδίπλωση μίας τέτοιας δομής, οι πολικές (υδρόφιλες) αλυσίδες των αμινοξέων έχουν την τάση να συγκεντρώνονται στην εξωτερική επιφάνεια του μορίου της πρωτεΐνης, όπου έχουν τη δυνατότητα να αλληλεπιδρούν με το νερό. Αντίθετα, οι μη πολικές (υδρόφοβες) πλευρικές αλυσίδες κρύβονται στο εσωτερικό και σχηματίζουν έναν υδρόφοβο "πυρήνα" σφιχτά συσσωρευμένων ατόμων, που αποφεύγουν το νερό [9] [1].



Σχήμα 1.4: Απεικόνιση πρωτεΐνης με βάση τα πολικά και μη-πολικά αμινοξέα της

1.3 ΟΡΙΣΜΟΣ ΤΟΥ LATTICE

Όπως αναφέρθηκε και στο εισαγωγικό σημείωμα, στο PCLF πρόβλημα σκοπός μας είναι να τοποθετήσουμε μία πρωτεΐνη σε ένα πλέγμα (lattice). Τί ακριβώς όμως είναι τα lattices; Τα lattices είναι γραφήματα τα οποία έχουν συγκεκριμένα χαρακτηριστικά. Στο σημείο αυτό θα κάνουμε μία μικρή ανάλυση του τί είναι γράφημα, χρησιμοποιώντας στοιχεία από τη θεωρία γραφημάτων και στη συνέχεια θα επανέλθουμε στα lattices και τα ιδιαίτερα χαρακτηριστικά τους.

1.3.1 ΣΤΟΙΧΕΙΑ ΑΠΟ ΤΗ ΘΕΩΡΙΑ ΓΡΑΦΗΜΑΤΩΝ

Ένα **Γράφημα G** (graph) (π.χ. [16]) ορίζεται από:

1. το σύνολο κορυφών $V(G) = \{v_1, v_2, \ldots, v_n\}$ και
2. το σύνολο ακμών $E(G) = \{e_1, e_2, \ldots, e_m\}$, όπου κάθε ακμή είναι ένα ζεύγος δύο κορυφών. Μία ακμή που ξεκινάει και τελειώνει στην ίδια κορυφή λέγεται *βρόγχος*. Δύο κορυφές που ανήκουν στην ίδια ακμή λέγονται *γειτονικές*.

Ένα **Κατευθυνόμενο Γράφημα** (directed graph) (π.χ. [16]) ορίζεται όπως το γράφημα με τη διαφορά ότι οι ακμές του είναι διατεταγμένα ζεύγη κορυφών. Αν η ακμή $e = (v, w)$, τότε λέμε πως η e είναι μία (κατευθυνόμενη) ακμή από το v στο w. *Βαθμός* ($deg(v)$) μίας κορυφής είναι το πλήθος των ακμών οι οποίες προσπίπτουν στην κορυφή.

Έστω τώρα ένα γράφημα G και v, w κορυφές του G.

Περίπατος (walk) (π.χ. [16]) από την κορυφή v στην κορυφή w είναι μία πεπερασμένη ακολουθία γειτονικών κορυφών και ακμών του G, όπως παρακάτω:

$$v_0 e_1 v_1 e_2 \ldots v_{n-1} e_n v_n, \tag{1.1}$$

όπου τα v_i αναπαριστούν κορυφές, τα e_i ακμές, $v_0 = v, v_n = w$, και για κάθε $i = 1, 2 \ldots, n$ $e_i = v_{i-1}, v_i$.

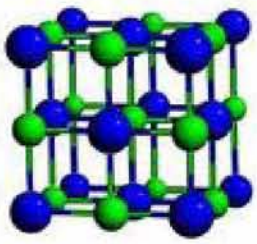
Ένα **Μονοπάτι** (path) (π.χ. [16]) από την κορυφή v στην κορυφή w είναι ένας περίπατος ο οποίος περνά το πολύ μία φορά από κάθε κορυφή. Άρα, ένα μονοπάτι από την κορυφή v στην κορυφή w είναι ένας περίπατος της μορφής

$$v = v_0 e_1 v_1 e_2 \ldots v_{n-1} e_n v_n = w, \tag{1.2}$$

όπου όλες οι v_i είναι διακριτές (δηλαδή, $v_i \neq v_k$ για οποιοδήποτε $i \neq k$).

1.3.2 ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΤΟΥ LATTICE

Στην παρούσα πτυχιακή εργασία χρησιμοποιούμε μία ειδική κατηγορία γραφημάτων, τα πλέγματα (lattices). Τα πλέγματα μπορεί να είναι δισδιάστατα (2Dimensional lattices (2D)) ή τρισδιάστατα (3Dimensional lattices (3D)). Ένα τρισδιάστατο lattice φαίνεται στο Σχήμα 1.5.



Σχήμα 1.5: 3D Lattice

Σε ένα δισδιάστατο lattice n*m κάθε κορυφή ορίζεται από ένα ζεύγος συντεταγμένων (v_x, v_y), όπου $1 \leq x \leq n$, $1 \leq y \leq m$. Αναλόγως γίνεται η γενίκευση στις τρεις διαστάσεις. Στην μέχρι τώρα μελέτη του PCLF (π.χ. [8]) προβλήματος χρησιμοποιήθηκαν κυρίως 3D lattices με τα εξής χαρακτηριστικά:

- Το μήκος των ακμών τους είναι ίσο με 3.8Å
- Η ελάχιστη απόσταση μεταξύ δύο κορυφών είναι ίση με 3.8Å
- Σχηματίζουν γωνίες 90°ή 120°
- Τέλος, τις περισσότερες φορές οι ακμές που προσπίπτουν σε κάποια κορυφή εμφανίζουν δομή που επαναλαμβάνεται.

1.4 ΓΡΑΜΜΙΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

Το PCLF πρόβλημα μπορεί να διατυπωθεί χρησιμοποιώντας Γραμμικό Προγραμματισμό. Πριν δώσουμε όμως μία τέτοια περιγραφή του προβλήματος, για λόγους πληρότητας της εργασίας και ευκολίας του αναγνώστη, θα κάνουμε σε αυτή την ενότητα μία σύντομη εισαγωγή στο Γραμμικό Προγραμματισμό.

Σε ένα πρόβλημα Γραμμικού Προγραμματισμού σκοπός είναι η μεγιστοποίηση ή ελαχιστοποίηση μίας γραμμικής συνάρτησης με βάση κάποιους γραμμικούς περιορισμούς. Οι περιορισμοί αυτοί μπορεί να είναι ισότητες ή ανισότητες. Στη συνέχεια ακολουθεί ένα παράδειγμα περιγραφής ενός προβλήματος με Γραμμικό Προγραμματισμό. [15].

Παράδειγμα 1 *Να βρεθούν οι τιμές των x_1, x_2 που ελαχιστοποιούν τη συνάρτηση $x_1 + x_2$ ενώ ισχύουν οι περιορισμοί:*

$$
\begin{aligned}
& x_1 \geq 0 \\
& x_2 \geq 0 \\
& x_1 + 2x_2 \geq 4 \\
& 4x_1 + 2x_2 \geq 12 \\
& -x_1 + x_2 \geq 1
\end{aligned}
$$

Στο παραπάνω πρόβλημα υπάρχουν δύο μεταβλητές και πέντε γραμμικοί περιορισμοί για αυτές. Οι δύο πρώτοι περιορισμοί $x_1 \geq 0$ και $x_2 \geq 0$ ανήκουν σε μία ξεχωριστή κατηγορία περιορισμών που συχνά συναντάμε σε προβλήματα Γραμμικού Προγραμματισμού και λέγονται "*μη αρνητικοί περιορισμοί*". Οι υπόλοιποι τρεις περιορισμοί λέγονται βασικοί περιορισμοί. Η συνάρτηση που ζητάμε να ελαχιστοποιηθεί λέγεται "*αντικειμενική συνάρτηση*" (objective function). Στο παράδειγμά μας η αντικειμενική συνάρτηση είναι το άθροισμα $x_1 + x_2$.

Η θεωρία του Γραμμικού Προγραμματισμού μέχρι τις αρχές της δεκαετίας του 70, εξελίχθηκε ως μέθοδος βελτιστοποίησης μίας μόνο αντικειμενικής συνάρτησης. Αργότερα μελετήθηκαν προβλήματα στα οποία στόχος είναι η ταυτόχρονη βελτιστοποίηση δύο ή και περισσότερων συναρτήσεων.

1.4.1 Ιστορική αναδρομή Γραμμικού Προγραμματισμού

Η ιστορία του Γραμμικού Προγραμματισμού ξεκινάει το 1939, όταν ο Leonid Kantorovich ανέπτυξε πρώτος το παλαιότερο πρόβλημα Γραμμικού Προγραμματισμού κατά τη διάρκεια του δεύτερου παγκοσμίου πολέμου. Στη συνέχεια ακολούθησαν ο George Dantzig, (π.χ.

[5] [12]) ο οποίος το 1947 εισήγαγε, μία μέθοδο επίλυσης προβλημάτων Γραμμικού Προγραμματισμού, την μέθοδο Simplex, που σήμερα είναι ευρέως διαδεδομένη. Η μέθοδος αυτή αποτελεί μία αλγεβρική επαναληπτική διαδικασία η οποία επιλύει ακριβώς, κάθε πρόβλημα γραμμικού προγραμματισμού σε ένα πεπερασμένο πλήθος βημάτων. Η μέθοδος αυτή είναι τόσο διαδεδομένη γιατί μπορεί να χειριστεί πολλές εξισώσεις ταυτόχρονα και γιατί στην πράξη είναι γρήγορη. Παρόλα αυτά, η μέθοδος Simplex στη χειρότερη περίπτωση απαιτεί εκθετικό αριθμό βημάτων ως προς το πλήθος των μεταβλητών.

Την ίδια χρονιά ο John von Neumann ανέπτυξε τη θεωρία της δυαδικότητας (Duality) (π.χ. [5] [12]). Η μέθοδος αυτή κρατήθηκε μυστική για χρόνια μέχρι τη δημοσίευσή της το 1947 όταν εφαρμόστηκε στη Θεωρία Παιγνίων (Game Theory).

Ο πρώτος αλγόριθμος πολυωνυμικού χρόνου για την επίλυση προβλημάτων Γραμμικού Προγραμματισμού, δόθηκε το 1979 από τον Leonid Khachiyan (π.χ. [5] [12]). Τελικά, το 1984 ο Narendra Karmarkar (π.χ. [5] [12]) εισήγαγε μία νέα μέθοδο επίλυσης προβλημάτων Γραμμικού Προγραμματισμού, τη Μέθοδο Εσωτερικών Σημείων (Interior-Point Method). Η μέθοδος του Karmarkar ήταν ένας νέος πολυωνυμικού χρόνου αλγόριθμος, ο οποίος ήταν και πρακτικά πιο γρήγορος από τον Simplex. Σε πολλές όμως πρακτικές εφαρμογές ο αλγόριθμος Simplex έδινε πιο γρήγορα αποτελέσματα από τον αλγόριθμο του Khachiyan.

1.4.2 Δομή προβλήματος στο Γραμμικό Προγραμματισμό

Για να είναι ένα πρόβλημα επιλύσιμο με Γραμμικό Προγραμματισμό πρέπει να έχει τα εξής χαρακτηριστικά:

1. Μια αντικειμενική συνάρτηση που ζητείται να ελαχιστοποιηθεί ή να μεγιστοποιηθεί

2. Οι μεταβλητές πρέπει να είναι μεγαλύτερες ή ίσες του μηδενός

3. Και τέλος, κάθε περιορισμός του προβλήματος πρέπει να είναι γραμμική ανίσωση της μορφής ($\leq$) της οποίας το δεύτερο μέλος είναι μία μη αρνητική σταθερά.

Σε περίπτωση που οι μεταβλητές επιτρέπεται να παίρνουν μόνο ακέραιες τιμές τότε το πρόβλημα ονομάζεται πρόβλημα *Ακέραιου Γραμμικού Προγραμματισμού*.

Το γενικό πρόβλημα Ακέραιου Προγραμματισμού έχει αποδειχθεί ότι είναι NP-complete (Βλέπε επόμενο κεφάλαιο).

1.5 ΧΡΟΝΙΚΗ ΠΟΛΥΠΛΟΚΟΤΗΤΑ/ NP-ΠΛΗΡΗ ΠΡΟΒΛΗΜΑΤΑ

Στο εισαγωγικό σημείωμα αναφέρεται πως το πρόβλημα "Protein Chain Lattice Fitting Problem (PCLF)" αποδείχτηκε πως ανήκει στα NP-complete προβλήματα πράγμα που σημαίνει πως δεν υπάρχει αλγόριθμος πολυωνυμικού χρόνου που να μπορεί να το επιλύσει εκτός αν P=NP. Στο Υποκεφάλαιο αυτό αρχικά θα ορίσουμε κάποιες βασικές έννοιες σχετικά με τη χρονική πολυπλοκότητα των προβλημάτων. Επίσης, θα αναλύσουμε τι είναι η κλάση των NP-complete προβλήματων. Τέλος, θα εξηγήσουμε τις επιπτώσεις που έχει στην εξέλιξη της έρευνας σχετικά με το πρόβλημα "Protein Chain Lattice Fitting Problem" το γεγονός ότι αποδείχτηκε πως ανήκει στην κλάση των NP-complete προβλημάτων.

Έστω P ένα πρόβλημα και I(x) ένα στιγμιότυπο του προβλήματος του P με είσοδο x. Έστω $f_a(I)$ ο αριθμός των πράξεων που κάνει ο αλγόριθμος a ώστε να λύσει το στιγμιότυπο I∈P.

Όρισμος 1 *Ορίζουμε* Χρονική Πολυπλοκότητα (T) αλγόριθμου *[16] [14] :*

$$T_a(n) = max_{I(x) \in P:|x|=n} f_a(I(x)) \tag{1.3}$$

Όπως εκφράζει η παραπάνω σχέση, η χρονική πολυπλοκότητα T ενός αλγόριθμου a είναι ο μέγιστος χρόνος που απαιτείται ώστε να λυθεί ένα στιγμιότυπο I του προβλήματος P.

Όρισμος 2 *Ορίζουμε* Χρονική Πολυπλοκότητα (C) προβλήματος *[16] [14] :*

$$C_p(n) = min_{a\ solves\ P} T_a(n) \tag{1.4}$$

Σύμφωνα με την παραπάνω σχέση η χρονική πολυπλοκότητα του προβλήματος C ορίζεται ως η χρονική πολυπλοκότητα T_a του πιο γρήγορου αλγόριθμου a που επιλύει το P.

Τα προβλήματα κατατάσσονται σε διάφορες κλάσεις πολυπλοκότητας ανάλογα με τους αλγόριθμους που τα επιλύουν. Μερικές από αυτές τις κλάσεις αναλύουμε στη συνέχεια.

1.5.1 ΚΛΑΣΗ P

Στην κλάση P (polynomial) [16] [14] ανήκουν τα προβλήματα που επιλύονται από αλγόριθμους των οποίων η χρονική πολυπλοκότητα είναι μία πολυωνυμική συνάρτηση του μεγέθους της εισόδου.

1.5.2 ΚΛΑΣΗ NP

Στην κλάση NP (non-deterministic polynomial) [16] [14] ανήκουν τα προβλήματα που επιλύονται από μη-ντετερμινιστικούς αλγόριθμους πολυωνυμικού χρόνου. Η λύση ενός προβλήματος που ανήκει στην κλάση NP μπορεί να επαληθευτεί σε πολυωνυμικό χρόνο. Επίσης ένα πρόβλημα που ανήκει στο NP μπορεί να επιλυθεί από ντετερμινιστικό αλγόριθμο εκθετικής πολυπλοκότητας. Το ερώτημα όμως αν όλα τα προβλήματα που ανήκουν στο NP επιδέχονται αλγόριθμους πολυωνυμικού χρόνου παραμένει ανοικτό για περισσότερα από 40 χρόνια και οι πιο πολλοί επιστήμονες πιστεύουν ότι η απάντηση είναι αρνητική.

1.5.3 ΠΡΟΒΛΗΜΑΤΑ NP-ΠΛΗΡΗ (NP-complete)

Μία σημαντική εξέλιξη στο ερώτημα αν "$P = NP$" πραγματοποιήθηκε στις αρχές της δεκαετίας του 1970 με εργασίες των Stephen Cook και Leonid Levin. Οι ερευνητές αυτοί, ανακάλυψαν ορισμένα προβλήματα που ανήκουν στην κλάση NP, τα οποία έχουν την εξής ιδιαιτερότητα: η πολυπλοκότητα καθενός από αυτά συνδέεται με την πολυπλοκότητα ολόκληρης της κλάσης. Συγκεκριμένα, εάν οποιοδήποτε από αυτά τα προβλήματα μπορεί να επιλυθεί με κάποιο αλγόριθμο πολυωνυμικού χρόνου, τότε το ίδιο ισχύει και για οποιοδήποτε άλλο πρόβλημα της κλάσης NP. [14]

Τα προβλήματα αυτά ονομάζονται NP-πλήρη (NP-complete). Για την επίλυση αυτών των προβλημάτων προτάθηκαν *προσεγγιστικοί αλγόριθμοι* οι οποίοι επιτυγχάνουν την εύρεση λύσης σε πολυωνυμικό χρόνο. Η λύση αυτή δεν είναι βέλτιση, έχει όμως την εγγύηση ότι δεν απέχει πολύ από την βέλτιστη.

Ένα άλλο είδος αλγόριθμου που προτάθηκε για την επίλυση των παραπάνω προβλημάτων είναι οι *ευριστικοί αλγόριθμοι*. Σε αυτή την κατηγορία ανήκουν αλγόριθμοι που συνήθως είτε επιστρέφουν τη βέλτιστη λύση σε εκθετικό χρόνο είτε επιστρέφουν μία προσεγγιστική λύση χωρίς όμως καμμία εγγύηση ποιότητας.

1.6 CRMS και DRMS

Όπως αναφέρθηκε στο εισαγωγικό σημείωμα, δύο πολύ συνηθισμένα κριτήρια που χρησιμοποιούμε και τα οποία δίνουν μία καλή ένδειξη για το αν η απεικόνιση μίας πρωτεΐνης πάνω σε ένα lattice είναι καλή ή όχι είναι το CRMS και το DRMS. Τι ακριβώς είναι όμως αυτά τα δύο μεγέθη και πώς ορίζονται; Στα ερωτήματα αυτά θα απαντήσουμε στη συνέχεια του υποκεφαλαίου αυτού.

1.6.1 CRMS

Το CRMS (Coordinates Root Mean Squared deviation) (π.χ. [3] [8]) , δηλαδή η ρίζα του μέσου όρου των τετραγωνικών αποκλίσεων των συντεταγμένων, μετράει την απόκλιση μεταξύ της απεικόνισης μιας πρωτεΐνης σε ένα lattice και της τρισδιάστατης πρωτεϊνικής δομής. Το CRMS υπολογίζεται ως εξής:

$$\text{CRMS} = \sqrt{\frac{\sum_{p \in P} d^2(p, f(p))}{n}} \tag{1.5}$$

όπου:
p είναι η θέση του αμινοξέος στην τρισδιάστατη δομή,
$f(p)$ είναι η θέση του αμινοξέος p στο lattice,
$d(p, f(p))$ είναι η Ευκλείδια απόσταση μεταξύ του p και του f(p),
n το πλήθος των αμονοξέων.

DRMS

Το DRMS (Distance Root Mean Squared deviation) (π.χ. [3] [8]) , δηλαδή η ρίζα του μέσου όρου των τετραγωνικών αποκλίσεων των αποστάσεων μετράει πόσο καλά μπορεί να παρουσιαστεί ολόκληρη η δομή στο lattice. Το DRMS υπολογίζεται ως εξής:

$$\text{DRMS} = \sqrt{\frac{\sum_{i-1}^{n-1} \sum_{j=i+1}^{n} (d(p_i, p_j) - d(q_i, q_j))^2}{\frac{n(n-1)}{2}}} \tag{1.6}$$

όπου:
p_i είναι οι θέσεις των αμινοξέων στην τρισδιάστατη δομή,
q_i είναι οι θέσεις του αμινοξέος p_i στο lattice,
n το πλήθος των αμονοξέων.

Κεφάλαιο 2

ΠΕΡΙΓΡΑΦΗ ΤΟΥ PCLF ΠΡΟΒΛΗΜΑΤΟΣ

Στο κεφάλαιο αυτό θα αναφερθούμε αφενός στη σχετική προηγούμενη έρευνα που έχει γίνει για το PCLF πρόβλημα και αφετέρου στον πλήρη ορισμό του PCLF προβλήματος.

2.1 ΣΧΕΤΙΚΗ ΠΡΟΗΓΟΥΜΕΝΗ ΕΡΕΥΝΑ ΓΙΑ ΤΟ PCLF

Στο κεφάλαιο αυτό θα αναφερθούμε σε προηγούμενη έρευνα που σχετίζεται με το PCLF και τα αποτελέσματα των ερευνών αυτών. Επίσης, θα κάνουμε μία εισαγωγή στον τρόπο με τον οποίο προσεγγίσαμε το πρόβλημα στην παρούσα πτυχιακή εργασία. Το PCLF πρόβλημα, κατατάσσεται σε μία περιοχή της υπολογιστικής βιολογίας της οποίας ένα από τα πιο διάσημα σχετικά προβλήματα είναι το Protein Folding Problem.

Στο Protein Folding Problem σκοπός είναι να βρεθεί η τρισδιάστατη αναδίπλωση μίας πρωτεΐνης με μόνο δεδομένο την ακολουθία των αμινοξέων της. Σε ένα σχετικό πρόβλημα (το Inverse Protein Folding Problem) (π.χ. [7]) δίνεται μία πρωτεΐνη και σκοπός είναι να βρεθεί η ακολουθία των αμινοξέων της.

Μία πρώτη απόπειρα να λυθεί το Protein Folding Problem έγινε το 1990 από τον Dill [4]. Ο Dill πρότεινε ένα μοντέλο που βασιζότανε στις υδροφοβικές αλληλεπιδράσεις που καθορίζουν την αναδίπλωση των πρωτεϊνικών αλυσίδων. Για το λόγο αυτό το μοντέλο του ονομάστηκε υδροφοβικό - πολικό μοντέλο (Hydrophobic - Polar (HP) Model).

Στο μοντέλο αυτό, τα 20 αμινοξέα που συμμετέχουν στην πρωτεϊνοσύνθεση, παρουσιάζονται ως υδρόφοβα (H) ή πολικά (P) μονομερή (π.χ. [3] [8]). Το χαρακτηριστικό της υδροφοβικότητας ή μη των μορίων, εξαρτάται από την αμοιβαία έλξη ή άπωση μεταξύ αυτών

και του νερού. Αναφορικά με το μόριο της πρωτεΐνης, η ένωση δύο υδρόφοβων αμινοξέων προκαλεί τη μείωση της ελεύθερης ενέργειας του μορίου. Πιο συγκεκριμένα, όσο μεγαλύτερος είναι ο αριθμός των μη-διαδοχικών υδρόφοβων αμινοξέων που ενώνονται, τόσο πιο σταθερό είναι το μόριο. Το μοντέλο του Dill εφαρμόστηκε σε δισδιάστατα τετραγωνικά lattices (2D), όπου διαδοχικά αμινοξέα τοποθετήθηκαν σε διαδοχικές κορυφές των lattices. Το κριτήριο της τοποθέτησης ήταν η μεγιστοποίηση των μη-γειτονικών υδρόφοβων αμινοξέων. Ο σκοπός της τοποθέτησης ήταν η τελική απεικόνιση στο lattice να προβλέπει καλά τη φυσική αναδίπλωση της αλυσίδας των αμινοξέων και άρα την τρισδιάστατη μορφή της πρωτεΐνης.

Στη συνέχεια ο Gupta [7] σχεδίασε θεωρητικές δισδιάστατες πρωτεΐνες και χρησιμοποιώντας το HP-Model, τοποθετησε τις αλυσίδες αμινοξέων πάνω σε δισδιάστατα lattices (2D lattices). Ενδιαφέρουσα πρόκληση της έρευνας την εποχή εκείνη αποτελούσε η επέκταση για τρισδιάστατα lattices (3D lattices) που έχουν πρακτικό βιολογικό ενδιαφέρον (π.χ. [3] [8]).

Για την επίλυση του PCLF προβλήματος έχουν γίνει πολλές μελέτες, οι σπουδαιότερες από τις οποίες παρουσιάζονται παρακάτω:

Στην έρευνα του Covell [2], χρησιμοποιήθηκε ένα σύνολο από κορυφές ενός lattice, κάθε μία από τις οποίες αντιστοιχούσε σε ένα αμινοξύ της πρωτεΐνης. Στην έρευνα αυτή, μελετήθηκαν όλες οι πιθανές αναδιπλώσεις της πρωτεΐνης πάνω στις κορυφές αυτές του lattice. Κάθε μία από αυτές απαριθμήθηκε με βάση το σύνολο των μη γειτονικών αμινοξέων που ερχόταν σε επαφή κατά την τοποθέτηση. Η αναδίπλωση που είχε τον μεγαλύτερο αριθμό μη γειτονικών αμινοξέων θεωρήθηκε η βέλτιστη, καθώς η δομή είχε τη μεγαλύτερη σταθερότητα. Σημειώνουμε πως στη μελέτη του Covell ανάμεσα στις λύσεις, υπήρχε πάντα η βέλτιση αναδίπλωση. (π.χ. [3] [8]).

Ο Godzik [6] πρότεινε μία νέα μέθοδο. Στην μέθοδο αυτή ξεκινάμε από το πρώτο αμινοξύ της πρωτεΐνης, το οποίο τοποθετείται στην κατάλληλη κορυφή του lattice, έτσι ώστε το CRMS μίας μικρής περιοχής της ολικής απεικόνισης της πρωτεΐνης πάνω στο lattice να ελαχιστοποιείται. Η κεντρική ιδέα της μεθόδου αυτής ήταν ότι η ολική βέλτιστη λύση μπορεί να προκύψει από μικρότερα τοπικά βέλτιστα (π.χ. [3] [8]).

Στην έρευνα των Park και Levitt [13] μελετήθηκε η σχέση μεταξύ της πολυπλοκότητας (Complexity) και της ακρίβειας (Accuracy) της αναδίπλωσης ενός πολυπεπτιδικού σκελετού πάνω σε ένα lattice. Αρχικά, πρότειναν έναν αλγόριθμο που έδινε μία αναδίπλωση της πρωτεΐνης πάνω σε lattice, πολύ κοντά στην φυσική τρισδιάστατη δομή της πρωτεΐνης. Βασίστηκαν στη μελέτη του Godzik και δημιούργησαν μία καθολική καλή τοποθέτηση κρατώντας έναν μικρό αριθμό από τοπικά βέλτιστα (500 συνολικά). Συγκεκριμένα διάλεγαν καλές τοπικές τοποθετήσεις ώστε να εξασφαλίσουν την ελάχιστη CRMS απόκλιση από κάθε τμήμα της

πρωτεϊνικής αλυσίδας (π.χ. [3] [8]). Τελικά, με βάση τον αλγόριθμο αυτό απέδειξαν ότι η σχέση μεταξύ της πολυπλοκότητας και της ακρίβειας είναι $(Accuracy) * (Complexity)^{-1/2}$.

Επεκτείνοντας τη μελέτη των Park και Levitt, ο Mead (π.χ. [3] [8]) πρότεινε έναν προσεγγιστικό αλγόριθμο για να υπολογίσει την πλησιέστερη απεικόνιση μίας γνωστής πρωτεϊνικής δομής πάνω σε ένα lattice. Παρ'ολα αυτά οι προσεγγιστικοί αλγόριθμοι δεν εγκυώνται ότι η προσεγγιστική λύση που βρίσκουν, ελαχιστοποιεί ταυτόχρονα και την CRMS (ή την DRMS) απόκλιση.

2.2 ΟΡΙΣΜΟΣ ΤΟΥ (PCLF) ΠΡΟΒΛΗΜΑΤΟΣ

Το πρόβλημα της αναδίπλωσης μιας πρωτεΐνης σε πλέγμα (Protein Chain Lattice Fitting Problem (PCLF)) πραγματεύεται την τοποθέτηση μιας πρωτεΐνης πάνω σε ένα πλέγμα. Πιο συγκεκριμένα, το πρόβλημα έχει ως εξής:

Έστω μία πρωτεΐνη A για την οποία είναι γνωστά:

1. Το σχήμα της πρωτεΐνης στο χώρο (οι συντεταγμένες κάθε αμινοξέος είναι γνωστές).
2. Η αλληλουχία των αμινοξέων που την αποτελούν.

Έστω και ένα τρισδιάστατο lattice L για το οποίο είναι γνωστό το μήκος των ακμών του.

Γνωρίζοντας τη δομή της πρωτεΐνης στο χώρο, προσπαθούμε να τοποθετήσουμε τα αμινοξέα ένα-ένα πάνω στις κορυφές του τρισδιάστατου lattice με σκοπό η τελική απεικόνιση της τοποθέτησης αυτής να είναι όσο το δυνατόν πιο κοντά στη τρισδιάστατη δομή της πρωτεΐνης.

Για να περιγράψουμε καλύτερα το πρόβλημα, ορίζουμε ένα σύνολο κορυφών V_P που αντιπροσωπεύουν τους a-άνθρακες των αμινοξέων της πρωτεΐνης

$$p_i \in V_P$$

όπου $1 \leq i \leq n$ και n το πλήθος των αμινοξέων

και ένα δεύτερο σύνολο κορυφών που αναπαριστά το σύνολο των κορυφών-κόμβων του lattice

$$q_j \in V_L$$

όπου $1 \leq j \leq m$ και m το πλήθος των κορυφών του. lattice.

Η τοποθέτηση των αμινοξέων στο lattice θα γίνει με τέτοιο τρόπο ώστε να ικανοποιούνται ορισμένοι περιορισμοί. Πριν αναλύσουμε τους περιορισμούς αυτούς πρέπει να αναφέρουμε πως για τη μαθηματική περιγραφή των περιορισμών αυτών χρησιμοποιήθηκαν βοηθητικές

μεταβλητές της μορφής $X_{i,j}$ (όπου τα i αντιπροσωπεύουν τα αμινοξέα και τα j τις κορυφές του lattice). Όλες οι μεταβλητές $X_{i,j}$ θα είναι δυαδικές μεταβλητές (binary variables) πράγμα που σημαίνει πως μπορούν να πάρουν τις τιμές 1 ή 0.

Αυτό μαθηματικά εκφράζεται ως εξής:

$$X_{i,j} = \begin{cases} 1, & if \quad \varphi(p_i) = q_j \\ 0, & otherwise \end{cases} \tag{2.1}$$

Όπως είναι προφανές από την παραπάνω σχέση, η μεταβλητή $X_{i,j}$ θα πάρει την τιμή 1 μόνο αν ισχύει η σχέση $\varphi(p_i) = q_j$. Η σχέση $\varphi(p_i) = q_j$ ουσιαστικά δηλώνει πως το αμινοξύ p_i αντιστοιχίζεται σε μία κορυφή q_j του lattice. Σε οποιαδήποτε άλλη περίπτωση, η μεταβλητή $X_{i,j}$ παίρνει την τιμή 0.

2.2.1 ΠΕΡΙΟΡΙΣΜΟΙ

Οι περιορισμοί του προβλήματος που θα αναλύσουμε στη συνέχεια αναπαριστούν τους κανόνες που διέπουν τη σωστή τοποθέτηση των αμινοξέων πάνω στο lattice.

Περιορισμός 1 *Κάθε αμινοξύ p_i τοποθετείται σε μία μοναδική κορυφή q_j του lattice*

$$\forall p_i \in V_P : \sum_{q_j \in V_L} X_{i,j} = 1 \tag{2.2}$$

Δηλαδή για τυχαίο p_i υπάρχει $X_{i,k} = 1$ και $X_{i,j} = 0$, $\forall j \neq k$.

Περιορισμός 2 *Σε κάθε κορυφή του lattice μπορεί να τοποθετηθεί το πολύ ένα αμινοξύ.*

$$\forall q_j \in V_L : \sum_{p_i \in V_P} X_{i,j} \leq 1 \tag{2.3}$$

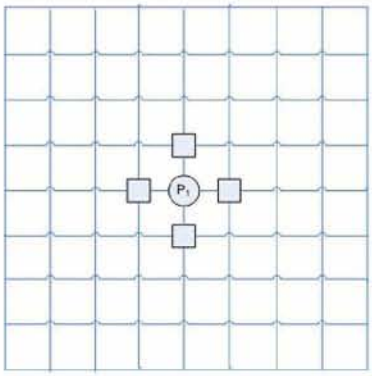
Περιορισμός 3 *Δύο διαδοχικά αμινοξέα τοποθετούνται σε γειτονικές κορυφές του lattice*

$$\forall q_j \in V_L, \forall p_i \in V_P : X_{i,j} + \sum_{q_k \in V_L - N(q_j)} X_{i+1,k} \leq 1 \tag{2.4}$$

Ο παραπάνω τύπος εκφράζει τα εξής:

Σε ένα τρισδιάστατο lattice αν μία κορυφή του lattice q_j καταληφθεί από ένα αμινοξύ p_i (δηλαδή $X_{i,j} = 1$), τότε το επόμενο στην αλυσίδα αμινοξύ p_{i+1} δεν μπορεί να τοποθετηθεί σε καμία άλλη κορυφή του lattice εκτός του συνόλου $N(q_j)$ στο οποίο ανήκουν οι 6 γειτονικές κορυφές της q_j.

Πιο αναλυτικά, δύο αμινοξέα που ενώνονται μεταξύ τους με πεπτιδικό δεσμό και άρα είναι γειτονικά στην αλυσίδα (Υποκεφάλαιο 1.2.1) θα τοποθετηθούν υποχρεωτικά σε γειτονικές κορυφές του lattice. Το Σχήμα 2.1 δείχνει τις γειτονικές κορυφές ενός σημείου (συμβολίζεται με κύκλο) σε ένα δισδιάστατο lattice, οι οποίες συμβολίζονται με τους τέσσερις (4) κύβους. Στις τρεις διαστάσεις, οι γειτονικές κορυφές ενός σημείου είναι έξι (6).



Σχήμα 2.1: Γειτονικές κορυφές του σημείου p_1 σε ένα δισδιάστατο lattice

2.2.2 ΑΝΤΙΚΕΙΜΕΝΙΚΗ ΣΥΝΑΡΤΗΣΗ

Στο PCLF η αντικειμενική συνάρτηση είναι η Ευκλείδια απόσταση CRMS (Υποκεφάλαιο 1.6.1 - Σχέση 1.5) μεταξύ των αμινοξέων της πρωτεΐνης και των εικόνων τους στο lattice, το τετράγωνο της οποίας θέλουμε να ελαχιστοποιήσουμε.

$$Minimize \sum_{p_i \in V_P, q_j \in V_L} c_{i,j} * X_{i,j}, \tag{2.5}$$

όπου

$$c_{i,j} = d^2(p_i, q_j) \tag{2.6}$$

Με βάση τους παραπάνω περιορισμούς πρέπει να γίνει η τοποθέτηση των αμινοξέων στις κορυφές του lattice. Στην παρούσα έρευνα, αρχικά τροποποιούμε τις μαθηματικές εξισώσεις

των περιορισμών αυτών, και σχεδιάζουμε και υλοποιούμε ένα πρόγραμμα σε C που παράγει τον ορισμό του PCLF προβλήματος στη μορφή προβλήματος Γραμμικού Προγραμματισμού με παραμέτρους μία πρωτεΐνη και ένα lattice.

Στο επόμενο κεφάλαιο περιγράφουμε βασικά χαρακτηριστικά του προγράμματος που δημιουργήσαμε και παραθέτουμε σχεδιαγράμματα για να κάνουμε τη δομή και τη λειτουργία του προγράμματος πιο κατανοητή.

Κεφάλαιο 3

ΕΠΙΛΥΣΗ ΤΟΥ PCLF ΠΡΟΒΛΗΜΑΤΟΣ ΜΕ ΣΧΕΔΙΑΓΡΑΜΜΑ

Στο κεφάλαιο αυτό, γίνεται μια πρώτη περιγραφή του προγράμματος που δημιουργήσαμε σε γλώσσα προγραμματισμού C, χρησιμοποιώντας ένα σχεδιάγραμμα (Σχήμα 3.7). Το σχεδιάγραμμα αυτό αποτελείται από 12 στάδια. Κάθε ένα από τα στάδια, περιγράφει μία βασική διεργασία που συντελεί στην επίλυση του προβλήματος. Στη συνέχεια, θα αναλύσουμε βήμα προς βήμα το σχεδιάγραμμα αυτό πρώτα όμως είναι απαραίτητο να πούμε δύο λόγια για τη διαδικασία επίλυσης του προβλήματος που επιλέξαμε να υλοποιήσουμε.

Τα δεδομένα που χρειαζόμαστε για κάθε μία πρωτεΐνη που μελετάμε, τα παίρνουμε από την Protein Data Bank (PDB) . Η PDB είναι μία βάση δεδομένων που περιέχει πληροφορίες σχετικά με διάφορες τρισδιάστατες μορφές μεγάλων βιολογικών μακρομορίων, όπως είναι οι πρωτεΐνες και τα νουκλεϊκά οξέα. Οι πληροφορίες αυτές βρίσκονται σε μορφή αρχείων. Ένα τέτοιο αρχείο έχει διάφορες πληροφορίες σχετικές με μία πρωτεΐνη. Από αυτές μας ενδιαφέρει να κρατήσουμε μόνο τις συντεταγμένες των αμινοξέων της πρωτεΐνης. Η εξόριξη αυτών των δεδομένων γίνεται από το πρόγραμμα που δημιουργήσαμε. Με την ολοκλήρωση της εκτέλεσης του προγράμματος δημιουργείται ένα νέο αρχείο στο οποίο καταγράφονται τα παρακάτω δεδομένα, που ορίζουν το πρόβλημα στη μορφή γραμμικού προγραμματισμού:

- Η αντικειμενική συνάρτηση του PCLF προβλήματος που επιθυμούμε να ελαχιστοποιηθεί (Σχέση 2.5)
- Οι περιορισμοί που διέπουν το πρόβλημα (Σχέσεις 2.2, 2.3 και 2.4) και τέλος,
- Ο τύπος των μεταβλητών που χρησιμοποιούμε για την επίλυση του προβλήματος. Στην

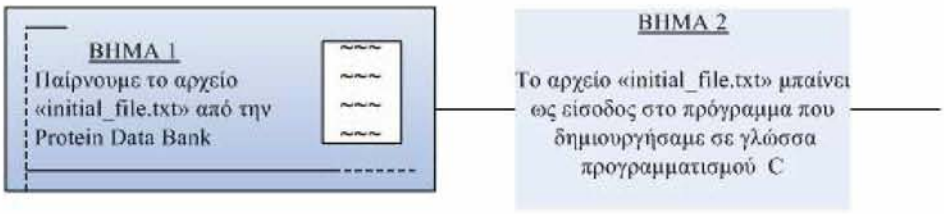
προκειμένη περίπτωση οι μεταβλητές είναι δυαδικές (binary), δηλαδή παίρνουν τιμές 0 ή 1.

Κατόπιν το αρχείο αυτό χρησιμοποιείται ως είσοδος σε ένα εμπορικό πρόγραμμα που ονομάζεται ILOG CPLEX. Το πρόγραμμα CPLEX είναι μία βιβλιοθήκη αλγορίθμων για την επίλυση προβλημάτων Γραμμικού Προγραμματισμού. Δίνοντας στο CPLEX τα δεδομένα, μας επιστρέφει μία βέλτιστη λύση. Περισσότερες λεπτομέρειες για τη λειτουργία του CPLEX θα εξηγήσουμε σε επόμενο κεφάλαιο.

Ας ξεκινήσουμε τώρα την ανάλυση του σχεδιαγράμματος με τις εισόδους που παίρνει το πρόγραμμα.

3.1 ΤΟ ΚΥΡΙΩΣ ΠΡΟΓΡΑΜΜΑ

Όπως βλέπουμε στο Σχήμα 3.1, το πρόγραμμα δέχεται ως είσοδο ένα αρχείο με όνομα "initial_file" από την PDB.

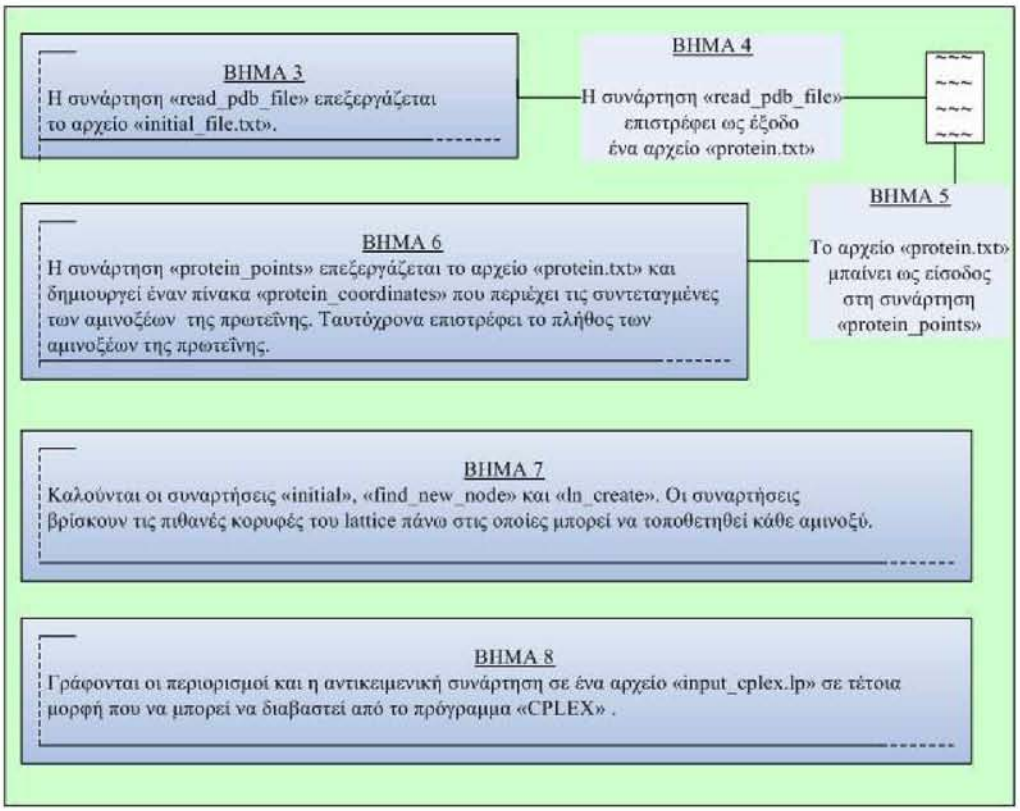


Σχήμα 3.1: Αρχείο Εισόδου

Ας συνεχίσουμε τώρα την ανάλυση του σχεδιαγράμματος, εστιάζοντας στις λειτουργίες του προγράμματος (Βλέπε Σχήμα 3.2). Το Σχήμα 3.2, απεικονίζει τις συναρτήσεις από τις οποίες αποτελείται το πρόγραμμά μας και περιγράφει τις βασικές λειτουργίες που εκτελεί κάθε μία από αυτές. Στη συνέχεια θα αναλύσουμε δύο από τις πιο σημαντικές συναρτήσεις, τη "read_pdb_file" και τη "protein_points" (Βλέπε Σχήμα 3.3).

Αρχικά εκτελείται η συνάρτηση "read_pdb_file", η οποία διαβάζει το αρχείο της PDB. Από το αρχείο αυτό συγκρατεί μόνο τις γραμμές που περιέχουν τις συντεταγμένες των αμινοξέων της πρωτεΐνης και τις αντιγράφει σε ένα δεύτερο αρχείο που ονομάζουμε "protein" και το οποίο είναι η έξοδος της συνάρτησης (Βλέπε Σχήμα 3.3, Βήματα 3 και 4).

Το αρχείο "protein" μπαίνει ως είσοδος στη συνάρτηση "protein_points". Η συνάρτηση αυτή διαβάζει μία-μία τις γραμμές του αρχείου. Το πλήθος των γραμμών είναι όσο και το πλήθος των αμινοξέων της πρωτεΐνης που διαβάστηκαν από τη συνάρτηση "read_pdb_file" (έστω p_p). Κάθε μία από τις γραμμές αυτές περιέχει τρείς αριθμούς, που αντιστοιχούν

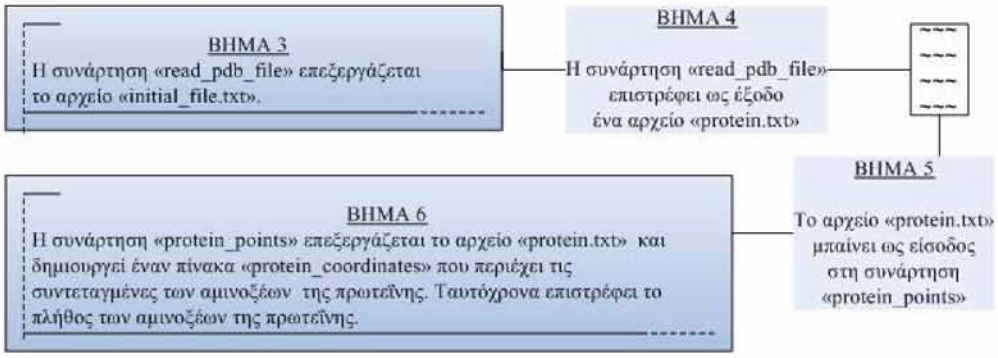


Σχήμα 3.2: Πρόγραμμα C-C++

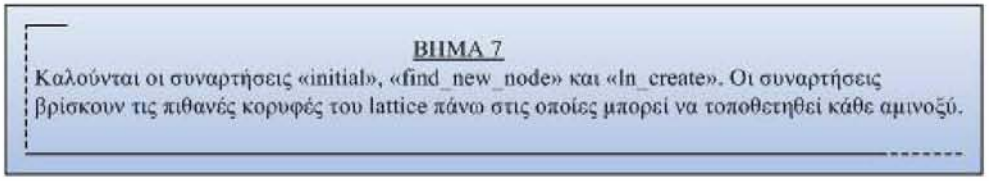
στις τρεις συντεταγμένες που έχει κάθε αμινοξύ στο χώρο. Η συνάρτηση "protein_points" δημιουργεί έναν πίνακα που ονομάζουμε "protein_coordinates", μεγέθους p_p. Σε κάθε μία από τις γραμμές του πίνακα, η συνάρτηση αντιγράφει τις συντεταγμένες διαφορετικού κάθε φορά αμινοξέος. Τέλος, η συνάρτηση "protein_points" επιστρέφει εκτός από τον πίνακα "protein_coordinates" και το πλήθος p_p των αμινοξέων (Βλέπε Σχήμα 3.3, Βήματα 5 και 6)

Όπως παρατηρούμε στο Σχήμα 3.4 στο Βήμα 7 γίνεται μία σειριακή εκτέλεση τριών συναρτήσεων, της "initial", της "find_new_node" και της "ln_create". Οι συναρτήσεις αυτές εκτελούνται λαμβάνοντας υπόψην τον Περιορισμό 3 του PCLF προβλήματος που εξηγήσαμε στο Κεφάλαιο 2.2 και βρίσκουν τις πιθανές κορυφές του lattice πάνω στις οποίες μπορεί να τοποθετηθεί κάθε ένα από τα αμινοξέα της πρωτεΐνης.

Τέλος, αφού έχουν βρεθεί οι πιθανές κορυφές τοποθέτησης για κάθε ένα αμινοξύ, το πρόγραμμα καταγράφει σε ένα αρχείο "input_cplex.lp" την αντικειμενική συνάρτηση, τους Περιορισμούς 1,2 και 3 που διέπουν το PCLF πρόβλημα και τον τύπο των μεταβλητών που χρησιμοποιήσαμε για την περιγραφή του προβλήματος (Σχήμα 3.5).



Σχήμα 3.3: Συναρτήσεις "read_pdb_file" και "protein_points"



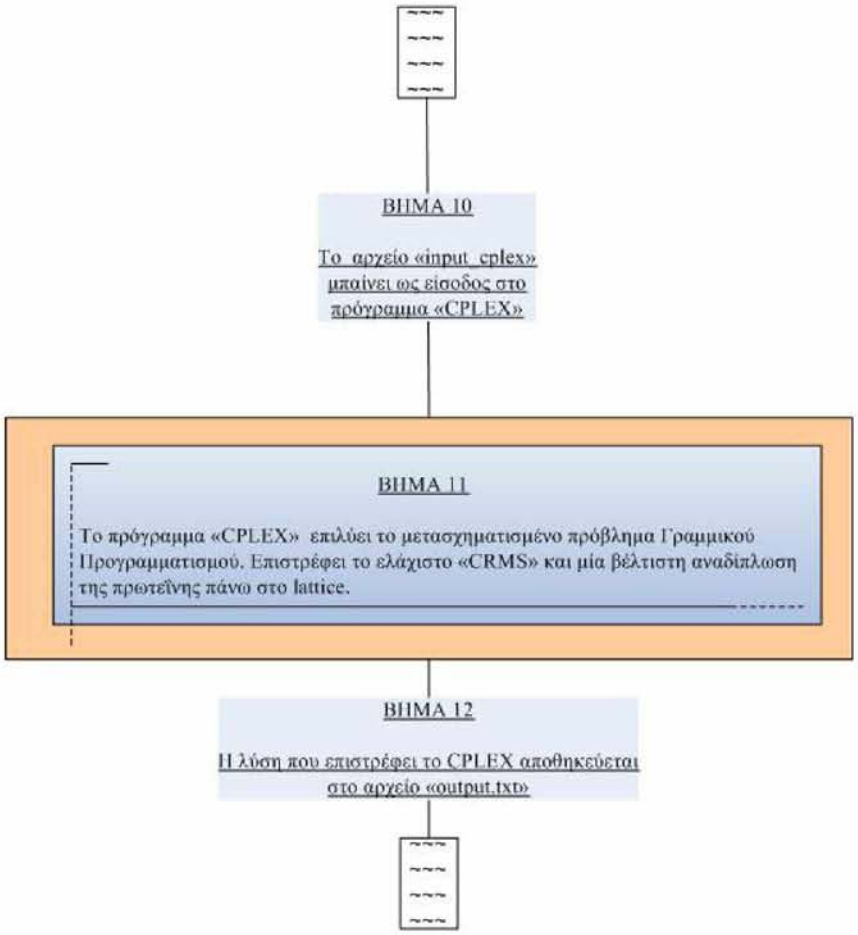
Σχήμα 3.4: Συναρτήσεις "initial", "find_new_node" και "ln_create"

3.2 ΠΡΟΓΡΑΜΜΑ ILOG CPLEX

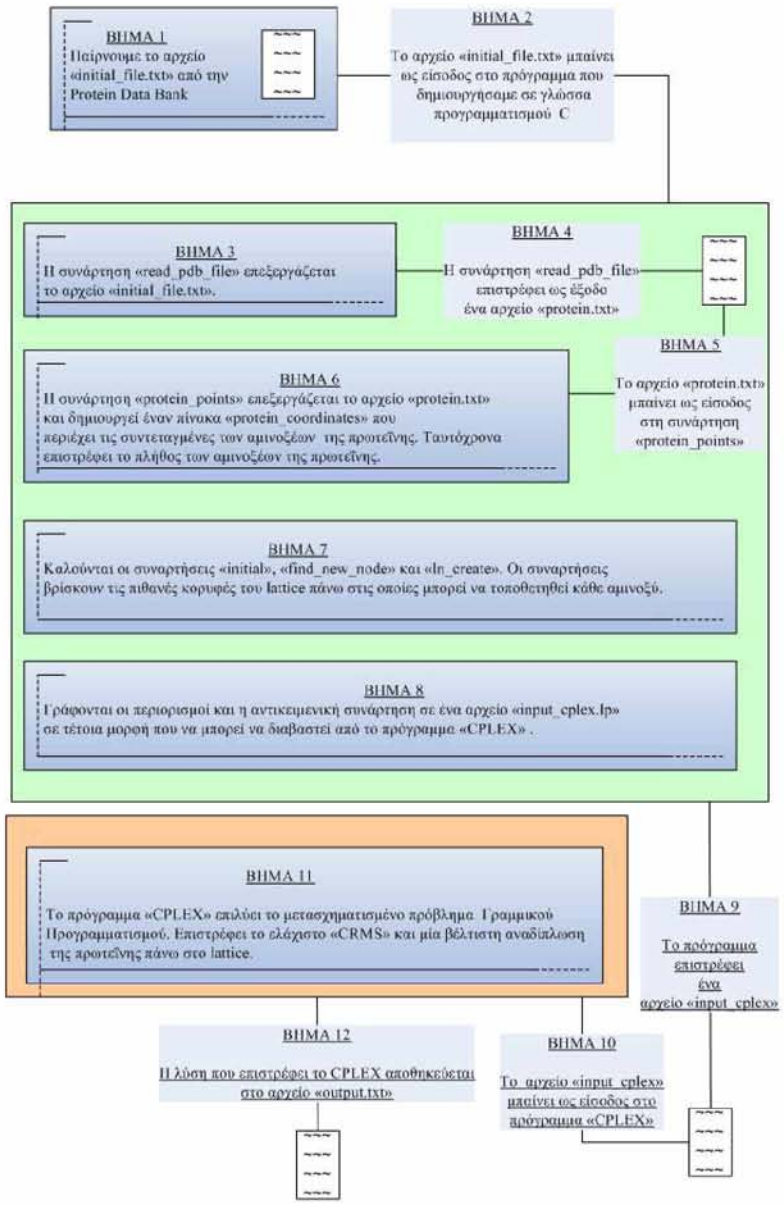
Όπως φαίνεται στο Σχήμα 3.6 το αρχείο "input_cplex" μπαίνει ως είσοδος στο πρόγραμμα ILOG CPLEX. Το αρχείο αυτό περιέχει περιέχει τον πλήρη ορισμό του προβλήματος (αντικειμενική συνάρτηση, περιορισμοί και τύπος μεταβλητών). Το CPLEX επιλύει το μετασχηματισμένο πρόβλημα εμφανίζοντας στο χρήστη την τιμή της αντικειμενικής συνάρτησης, το χρόνο που χρειάστηκε το CPLEX για να βρει τη λύση, καθώς επίσης και τη λύση με αναλυτική ανάθεση τιμών σε κάθε μία από τις μεταβλητές του προβλήματος. Το CPLEX υπολογίζει μία βέλτιστη αναδίπλωση της πρωτεΐνης στο lattice ελαχιστοποιώντας το CRMS (Βήμα 11). Η λύση αποθηκεύεται στο αρχείο "output.txt" (Σχήμα 3.6 Βήμα 12).

Το ολοκληρωμένο πλάνο του υπολογισμού μίας βέλτιστης αναδίπλωσης μίας πρωτεΐνης πάνω σε ένα τρισδιάστατο lattice απεικονίζεται στο Σχήμα 3.7, το οποίο περιέχει όλες τις διεργασίες που περιγράψαμε παραπάνω.

Σχήμα 3.5: Δημιουργία του αρχείο "input_cplex.lp"



Σχήμα 3.6: Πρόγραμμα ILOG CPLEX



Σχήμα 3.7: Συνολικό πλάνο υπολογισμού μίας βέλτιστης αναδίπλωσης μίας δεδομένης πρωτεΐνης πάνω σε ένα δεδομένο lattice.

Κεφάλαιο 4

ΑΝΑΛΥΣΗ ΠΡΟΓΡΑΜΜΑΤΟΣ

Στο κεφάλαιο αυτό θα αναλύσουμε σε τεχνικό επίπεδο το πρόγραμμα που δημιουργήσαμε. Θα αναλύσουμε τη λειτουργία του, αιτιολογώντας τα κριτήρια με βάση τα οποία επιλέξαμε να το φτιάξουμε κατά αυτόν τον τρόπο. Στο σημείο αυτό, θα ήταν χρήσιμο να επαναλάβουμε το σκοπό του PCLF προβλήματος. Στο PCLF πρόβλημα δίνεται η τρισδιάστατη αναδίπλωση μίας πρωτεΐνης και ένα lattice, ενώ ζητείται είναι μία αναδίπλωση της πρωτεΐνης πάνω στο lattice, τέτοια ώστε οι δύο αναδιπλώσεις να "απέχουν" όσο το δυνατόν λιγότερο. Το κριτήριο που χρησιμοποιούμε για να αξιολογήσουμε πόσο καλή είναι η λύση, είναι η CRMS απόκλιση. Όσο πιο μικρή (κοντά στο 0) είναι η CRMS απόκλιση, τόσο καλύτερη είναι η απεικόνιση.

4.1 ΣΥΛΛΟΓΗ ΔΕΔΟΜΕΝΩΝ ΑΠΟ ΤΗΝ PDB

Αρχικά, πρέπει να συλλέξουμε βασικές πληροφορίες σχετικά με την πρωτεΐνη και πιο συγκεκριμένα:

- Την ακριβή αλληλουχία των αμινοξέων της στην πρωτοταγή δομή της και
- Τις συντεταγμένες (p_x, p_y, p_z) του κεντρικού άνθρακα (a-άνθρακα) όλων των αμινοξέων της

Πληροφορίες σχετικά με πρωτεΐνες, των οποίων η δομή έχει αποκωδικοποιηθεί από βιολόγους, είναι αποθηκευμένες σε βάσεις δεδομένων μία από τις οποίες είναι η Protein Data Bank (PDB) σε μορφή αρχείων κειμένου. Σε ένα τέτοιο αρχείο περιέχονται διάφορες πληροφορίες όπως το όνομα της πρωτεΐνης, σε ποια ομάδα ανήκει, τα ονόματα των ερευνητών που την μελέτησαν, την αλληλουχία των αμινοξέων της, τη γεωμετρική και στερεοχημική δομή της, συντεταγμένες όλων των στοιχείων των αμινοξέων της κ.α. Από τις πληροφορίες αυτές, μας

ενδιαφέρει να διαβάσουμε εκείνες τις γραμμές του αρχείου που περιέχουν τις συντεταγμένες των αμινοξέων και πιο συγκεκριμένα μόνο των a-ανθράκων.

Παράδειγμα 2 *Ο Πίνακας 4.1 αναφέρεται στο αμινοξύ σερίνη (Ser). Στις γραμμές του πίνακα αυτού, καταγράφονται οι συντεταγμένες των χημικών στοιχείων που αποτελούν το αμινοξύ "σερίνη" με τη μορφή που απεικονίζονται και σε ένα αρχείο της PDB.*

ATOM	1	N	SER	A	6	−2.300	21.658	−13.857	1.00	37.08	N
ATOM	2	CA	SER	A	6	−1.777	21.184	−12.541	1.00	35.91	C
ATOM	3	C	SER	A	6	−2.000	22.265	−11.485	1.00	32.46	C
ATOM	4	O	SER	A	6	−2.091	23.455	−11.807	1.00	30.67	O
ATOM	5	CB	SER	A	6	−0.274	20.864	−12.654	1.00	37.80	C
ATOM	6	OG	SER	A	6	0.296	20.468	−11.405	1.00	36.96	O

Πίνακας 4.1: Συντεταγμένες των στοιχείων της σερίνης Ser στην PDB

Από αυτές τις παραπάνω γραμμές μας ενδιαφέρει να συγκρατήσουμε μόνο τη δεύτερη γραμμή (Πίνακας 4.2) που αναφέρεται στον a-άνθρακα (CA) του αμινοξέος Ser.

ATOM	2	CA	SER	A	6	−1.777	21.184	−12.541	1.00	35.91	C

Πίνακας 4.2: Συντεταγμένες του a-άνθρακα της σερίνης Ser

Παρακάτω θα δούμε πως ακριβώς γίνεται η διαδικασία αυτή από το πρόγραμμα που δημιουργήσαμε. Πρίν ξεκινήσουμε να περιγράφουμε οποιαδήποτε διαδικασία όμως είναι χρήσιμο να πούμε λίγα λόγια για τα χαρακτηριστικά του προγράμματος.

Το πρόγραμμα είναι γραμμένο στη γλώσσα C. Ο βασικός του στόχος είναι να δημιουργήσει ένα αρχείο, στο οποίο θα ορίζεται ακριβώς το PCLF πρόβλημα για μία δεδομένη πρωτεΐνη και ένα δεδομένο lattice στη μορφή όμως ενός προβλήματος Γραμμικού Προγραμματισμού, έτσι ώστε το αρχείο αυτό να μπει ως είσοδος στο CPLEX (Βλέπε Σχήμα 3.7). Ας δούμε όμως αναλυτικά όλα τα βήματα που συντελούν στη σωστή δημιουργία του αρχείου αυτού.

4.2 ΑΝΑΛΥΤΙΚΗ ΠΕΡΙΓΡΑΦΗ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ

4.2.1 Η ΣΥΝΑΡΤΗΣΗ "read_pdb_file"

Αρχικά το πρόγραμμα καλεί τη συνάρτηση "read_pdb_file". Τα βασικά βήματα του αλγόριθμου που εκτελεί η "read_pdb_file" είναι τα παρακάτω:

1. Η συνάρτηση "read_pdb_file" παίρνει ως είσοδο ένα αρχείο από την PDB.
2. Διαβάζει μία-μία τις γραμμές του αρχείου.
3. Όσες γραμμές από αυτές έχουν τα τρία πρώτα πεδία ίδια με αυτά του Πίνακα 4.2, τις αντιγράφει σε ένα νέο αρχείο που ονομάζει protein.txt.

Πιο αναλυτικά, η συνάρτηση *"read_pdb_file"* παίρνει ως είσοδο το αρχείο initial_file.txt, το οποίο περιέχει όλες τις πληροφορίες που χρειαζόμαστε για την πρωτεΐνη (Βλέπε Υποκεφάλαιο 4.1), και δημιουργεί ένα νέο αρχείο με όνομα protein.txt στο οποίο είναι αποθηκευμένες μόνο οι συντεταγμένες των κεντρικών a-ανθράκων κάθε αμινοξέος.

Η συνάρτηση αυτή ψάχνει να βρει σε όλες τις γραμμές του αρχείου όσες είναι παρόμοιες με αυτήν του Πίνακα 4.2. Μόλις συναντήσει στο αρχείο την πρώτη γραμμή που ξεκινάει με τη λέξη "ATOM", προχωράει στην ανάγνωση αυτής της γραμμής. Αν διαβάσει, στο τρίτο πεδίο της γραμμής, τους χαρακτήρες "CA" (πράγμα που σημαίνει πως ακολουθούν οι συντεταγμένες του a-άνθρακα), δημιουργεί ένα νέο αρχείο κειμένου που ονομάζουμε protein.txt και αντιγράφει στην πρώτη γραμμή αυτού του αρχείου τις συντεταγμένες που διάβασε.

Στη συνέχεια, η συνάρτηση προχωράει την αναζήτησή της στις επόμενες γραμμές του initial_file.txt και όσο η παραπάνω διαδικασία αναζήτησης των "ATOM" (στο πρώτο πεδίο) και "CA" (στο τρίτο πεδίο) γίνεται επιτυχώς, οι συντεταγμένες των υπόλοιπων a-ανθράκων αντιγράφονται σε διαφορετικές γραμμές του αρχείου protein.txt. Αν όμως ο χαρακτήρας που διαβάζει στο τρίτο πεδίο είναι μόνο ο χαρακτήρας "C" ή είναι της μορφής "CX", όπου X οποιοσδήποτε λατινικός χαρακτήρας εκτός του A, συνεχίζει την αναζήτηση ("ATOM" ->"CA") στις επόμενες γραμμές του αρχείου *initial_file.txt*.

Τελικά, το αρχείο protein.txt αποτελείται από ένα πλήθος γραμμών όσο και το πλήθος των αμινοξέων που διάβασε προηγουμένως η συνάρτηση "read_pdb_file". Κάθε μία από αυτές τις γραμμές έχει τρείς αριθμούς που αντιπροσωπεύουν τις τρείς συντεταγμένες καθενός αμινοξέος που είναι καταγεγραμμένες στο αρχείο initial_file.txt.

Το αρχείο αυτό που δημιουργήθηκε, το επεξεργάζεται μία άλλη συνάρτηση η "protein_points". Τη διαδικασία της επεξεργασίας αυτής περιγράφουμε στη συνέχεια.

4.2.2 Η ΣΥΝΑΡΤΗΣΗ "protein_points"

Τα βασικά βήματα του αλγόριθμου που εκτελεί η "protein_points" είναι τα παρακάτω:

1. Η συνάρτηση "protein_points" διαβάζει από κάθε γραμμή του αρχείου protein.txt τρεις αριθμούς (Οι αριθμοί αυτοί διαβάζονται ως χαρακτήρες).

2. Μετατρέπει τους χαρακτήρες σε δεκαδικούς αριθμούς.

3. Αποθηκεύει τους αριθμούς σε έναν πίνακα protein_coordinates.

4. Επιστρέφει το πλήθος των γραμμών του αρχείου protein.txt που αντιστοιχεί στο πλήθος των αμινοξέων της πρωτεΐνης.

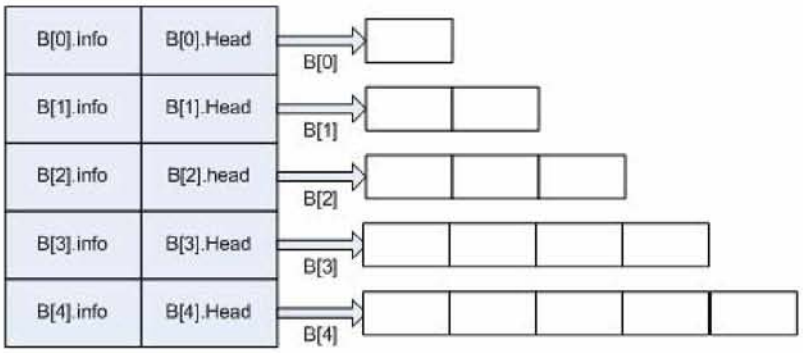
Πιο αναλυτικά, η συνάρτηση "protein_points" δέχεται ως όρισμα το αρχείο protein.txt. Επιστρέφει έναν δισδιάστατο πίνακα που ονομάζουμε protein_coordinates[p_p][3] και το πλήθος των αμινοξέων (p_p), των οποίων οι συντεταγμένες είναι καταγεγραμμένες στο αρχείο protein.txt. Το μέγεθος της πρώτης διάστασης του πίνακα είναι όσο το πλήθος p_p των αμινοξέων της πρωτεΐνης. Το μέγεθος της δεύτερης διάστασης του πίνακα ισούται με τον αριθμό τρία (3). Κάθε μία γραμμή του πίνακα αναφέρεται σε ένα διαφορετικό αμινοξύ, ενώ οι τρεις στήλες αντιπροσωπεύουν τις τρεις συντεταγμένες (p_x, p_y, p_z) του κεντρικού a-άνθρακα αυτού του αμινοξέος.

4.2.3 Η ΣΥΝΑΡΤΗΣΗ "initial"

Επόμενο βήμα του προγράμματος είναι η κλήση της συνάρτησης "initial". Η συνάρτηση αυτή παίρνει ως όρισμα έναν μονοδιάστατο πίνακα B με μέγεθος που ισούται με το πλήθος των αμινοξέων της πρωτεΐνης B[p_p].

ΠΙΝΑΚΑΣ B

Ο πίνακας B είναι μία δομή που παίζει καθοριστικό ρόλο στο πρόγραμμά μας. Όπως ήδη έχουμε αναφέρει σε προηγούμενα κεφάλαια, το PCLF πρόβλημα διέπεται από κάποιους περιορισμούς (Βλέπε Σχέσεις 2.2, 2.3 και 2.4). Οι περιορισμοί αυτοί καθορίζουν ποιες θα είναι οι υποψήφιες κορυφές του lattice πάνω στις οποίες μπορεί να τοποθετηθεί κάθε αμινοξύ της πρωτεΐνης. Οι κορυφές αυτές αποθηκεύονται στον πίνακα B.



Σχήμα 4.1: Δομή του πίνακα B[ ] για π.χ. 5 αμινοξέων

Πιο αναλυτικά, κάθε στοιχείο του πίνακα B αντιστοιχεί σε ένα αμινοξύ και αποτελείται από δύο πεδία (Βλέπε Σχήμα 4.1). Στο πρώτο πεδίο (B.info) αποθηκεύεται ένας ακέραιος αριθμός, που αντιπροσωπεύει το πλήθος των πιθανών κορυφών του lattice πάνω στις οποίες μπορεί να τοποθετηθεί ένα αμινοξύ. Το δεύτερο πεδίο (B.Head) είναι ένας δείκτης που δείχνει σε μία λίστα που αποτελείται από κόμβους το πλήθος των οποίων είναι B.info. Περισσότερες πληροφορίες σχετικά με τη δομή και το περιεχόμενο μίας τέτοιας λίστας θα πούμε στο Υποκεφάλαιο 4.2.8.

Επιστρέφουμε στη συνάρτηση "initial" η οποία αρχικοποιεί τον πίνακα B όπως φαίνεται στο Σχήμα 4.2.

Σχήμα 4.2: Αρχικοποίηση του πίνακα B από την "initial"

4.2.4 Η ΣΥΝΑΡΤΗΣΗ "ln_create"

Η συνάρτηση "ln_create" είναι υπεύθυνη για τη δημιουργία της λίστας υποψήφιων κόμβων

του lattice στους οποίους θα μπορούσε να τοποθετηθεί ένα αμινοξύ. Τα βασικά βήματα του αλγόριθμου που εκτελεί η "ln_create" είναι τα παρακάτω:

1. Η "ln_create" παίρνει ένα-ένα τα αμινοξέα με τη σειρά που εμφανίζονται στην αλυσίδα της πρωτεΐνης.

2. Δημιουργεί για κάθε ένα από αυτά μία λίστα υποψήφιων κορυφών του lattice στις οποίες κορυφές θα μπορούσε να τοποθετηθεί το δεδομένο αμινοξύ που μελετάει κάθε φορά. Για την εύρεση της λίστας των κορυφών ενός αμινοξέος εξετάζει τη λίστα των κορυφών του προηγούμενου στη σειρά αμινοξέος. Από την παραπάνω διαδικασία εξαιρείται το πρώτο αμινοξύ το οποίο τοποθετείται "αυτόματα" σε μία μόνο κορυφή του lattice (άρα η λίστα που του αντιστοιχεί έχει μόνο έναν κόμβο) όπως θα αναλύσουμε στη συνέχεια.

Συγκεκριμένα για το πρώτο αμινοξύ, υπάρχει μία μόνο κορυφή στο lattice όπου μπορεί να τοποθετηθεί. Η κορυφή αυτή προκαθορίστηκε απο εμάς μετά από μελέτη διαφόρων περιπτώσεων. Η συνάρτηση "ln_create" επιδρά πάνω στον πίνακα B έτσι ώστε μετά την κλήση της για το i-οστό αμινοξύ της πρωτεΐνης ο δείκτης B.Head να δείχνει στην κορυφή μίας λίστας που περιέχει όλες τις κορυφές του lattice στις οποίες θα μπορούσε να τοποθετηθεί το συγκεκριμένο αμινοξύ ενώ η μεταβλητή B.info ισούται με το πλήθος των κορυφών αυτών. Στο Υποκεφάλαιο 4.2.5 που ακολουθεί περιγράφουμε με βάση ποια κριτήρια αποφασίσαμε να τοποθετήσουμε το πρώτο αμινοξύ σε μία συγκεκριμένη κορυφή του lattice και άρα κατ'επέκταση ποια είναι η θέση του lattice στο χώρο σε σχέση με τη θέση της πρωτεΐνης.

4.2.5 ΚΡΙΤΗΡΙΑ ΤΟΠΟΘΕΤΗΣΗΣ ΤΟΥ LATTICE

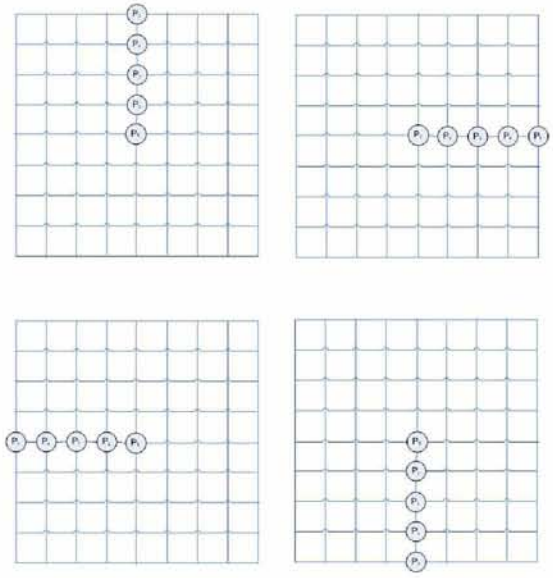
Δύο ερωτήματα που μας προβλημάτισαν σχετικά με την τοποθέτηση της πρωτεΐνης πάνω στο lattice ήταν τα παρακάτω:

1. Σε ποιο σημείο του lattice πρέπει να τοποθετηθεί το πρώτο αμινοξύ.

2. Ποιο θα πρέπει να είναι το μέγεθος του lattice.

Μετά από προσεκτική μελέτη αποφασίσαμε πως τόσο η θέση του lattice στο χώρο όσο και το μέγεθός του είναι άρρηκτα συνδεδεμένα με τη θέση της πρωτεΐνης στο χώρο και με το πλήθος (p_p) των αμινοξέων της αντίστοιχα. Η επιλογή της θέσης του lattice στο χώρο ως προς τη θέση της πρωτεΐνης επιλέγεται με τον παρακάτω τρόπο:

Παίρνουμε μία κορυφή K του lattice στην οποία δίνουμε συντεταγμένες τις συντεταγμένες του πρώτου αμινοξέος (P_1) της πρωτεΐνης. Στην κορυφή K θα τοποθετηθεί το πρώτο αμινοξύ.



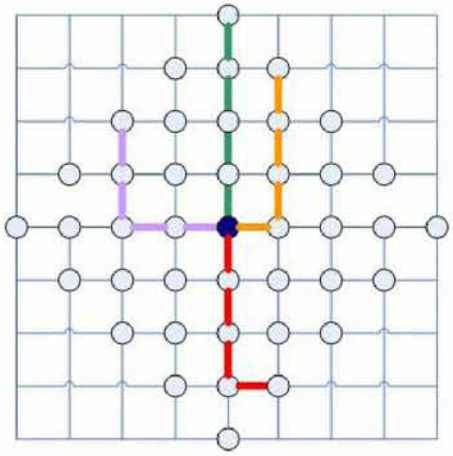
Σχήμα 4.3: Αλυσίδα αμινοξέων πλήθους p_p=5 που τοποθετείται σε δισδιάστατο lattice χωρίς να αναδιπλωθεί.

Επομένως απαιτούμενο πλήθος κορυφών ενός τρισδιάστατου lattice πρέπει να είναι $((2 * p\_p) - 1)^3$ (αντίστοιχα $((2 * p\_p) - 1)^2$ για δισδιάστατο lattice) έτσι ώστε να υπάρχουν αρκετές κορυφές για να αναπαραστήσουν οποιαδήποτε αναδίπλωση μίας πρωτεΐνης (ακόμα και τη χειρότερη περίπτωση, κατά την οποία η πρωτεΐνη δεν αναδιπλώνεται αλλά τοποθετείται γραμμικά πάνω στο lattice, Βλέπε Σχήμα 4.3).

Υπενθυμίζουμε πως γειτονικές κορυφές της κορυφής Κ είναι όσες ενώνονται με αυτήν μέσω μίας ακμής. Έτσι στο επίπεδο το πλήθος των κορυφών αυτών είναι ίσο με τέσσερα (4) ενώ στο χώρο ίσο με έξι (6). Με δεδομένο το Σχήμα 4.4 και όσων είπαμε παραπάνω σχετικά με τη γειτνίαση των κορυφών του lattice, εύκολα καταλαβαίνει κανείς το λόγο που αρκετές κορυφές δεν μπορεί να είναι υποψήφιες για την τοποθέτηση κάποιου αμινοξέος της πρωτεΐνης.

Για την ακρίβεια, μία κορυφή του lattice (για λόγους απλούστευσης έστω ότι το lattice είναι δισδιάστατο) με "συντεταγμένες θέσης" (l_x, l_y) (οι "συντεταγμένες θέσης" από εδώ και στο εξής στο συγκεκριμένο κείμενο θα περιγράφουν τη σχετική θέση μίας κορυφής του lattice ως προς την κορυφή (0,0) για δισδιάστατα lattices και αντίστοιχα ως προς την κορυφή (0,0,0) για τρισδιάστατα lattices) μπορεί να είναι υποψήφια θέση για κάποιο αμινοξύ αν και μόνο αν ισχύει η παρακάτω σχέση:

$$|p_x| + |p_y| \leq p_p - 1$$



Σχήμα 4.4: Υποψήφιες κορυφές του lattice για την τοποθέτηση των αμινοξέων

καθώς οποιοδήποτε μονοπάτι μήκους p_p που ξεκινάει από την κορυφή (0,0) δεν μπορεί να εκτίνεται σε καμία κορυφή για την οποία ισχύει:

$$|p_x| + |p_y| > p_p - 1$$

Έτσι λοιπόν οι υποψήφιες θέσεις συνολικά στο lattice είναι:

$$sol = 4 \sum_{i=1}^{p\_p-2} i + 4(p\_p - 1) + 1 =$$

$$\frac{4(p\_p - 2)(p\_p - 1)}{2} + 4(p\_p - 1) + 1 =$$

$$2p\_p(p\_p - 1) + 1$$

Με δεδομένο ότι ο συνολικός αριθμός των κορυφών του δισδιάστατου lattice είναι $tot = ((2 * p\_p) - 1)^2$, ισχύει:

$$2 * sol = tot + 1 \implies sol = \frac{tot + 1}{2}$$

Όπως φαίνεται παραπάνω ο συνολικός αριθμός των υποψήφιων θέσεων (sol) στο lattice είναι περίπου ο μισός του συνολικού πλήθους (tot) των κορυφών του lattice.

Το πρόγραμμά μας βρίσκει μόνο αυτές τις υποψήφιες θέσεις κάνοντας "οικονομία" στο συνολικό αριθμό μεταβλητών και εξισώσεων του τελικού προβλήματος Γραμμικού Προγραμματισμού. Υπενθυμίζουμε στο σημείο αυτό ότι το τελικό πρόβλημα είναι πρόβλημα ακέραιου Γραμμικού Προγραμματισμού για το οποίο είναι γνωστοί μόνο εκθετικοί αλγόριθμοι. Συνεπώς, η παραπάνω οικονομία έχει ως αποτέλεσμα την μείωση του εκθέτη στην υπολογιστική

πολυπλοκότητα του αλγόριθμου επίλυσης (CPLEX) και άρα κάνει εφικτή τη μελέτη μεγαλύτερων αλυσίδων. Παρ'όλα αυτά, το σύστημα που διαθέταμε δεν μας επέτρεψε από άποψη μνήμης να επεξεργαστούμε αλυσίδες με περισσότερα από 20 αμινοξέα.

4.2.6 Η ΣΥΝΑΡΤΗΣΗ "find_new_node"

Στη διαδικασία εύρεσης των υποψήφιων κορυφών του lattice για το i-οστό αμινοξύ (όπου $i > 1$) παίζουν σημαντικό ρόλο δύο συναρτήσεις που εκτελούνται σειριακά, η συνάρτηση "find_new_node" και η συνάρτηση "ln_create". Η σειριακή εκτέλεση αποτελείται από τα παρακάτω βήματα:

1. Η "find_new_node" ελέγχει τη λίστα των υποψήφιων κορυφών που έχει δημιουργηθεί για το i-1 αμινοξύ.

2. Για κάθε μία από αυτές τις κορυφές η "find_new_node" βρίσκει και αποθηκεύει σε έναν βοηθητικό πίνακα C τις έξι (6) πιθανές γειτονικές κορυφές για το i-οστό αμινοξύ (Βλέπε Πίνακα 4.3).

3. Με βάση τον βοηθητικό πίνακα C, η "ln_create" δημιουργεί τη λίστα με τις πιθανές κορυφές του i-οστού αμινοξέος απαλοίφοντας εκείνες που είχε ήδη τοποθετήσει στη λίστα, συμπεριλαμβανομένης και της κορυφής K (αν αυτή προκύψει).

4. Στη συνέχεια η "ln_create" επεμβαίνει στην i-οστή γραμμή του πίνακα B στην οποία ενώνει τη λίστα που δημιούργησε για το i-οστό αμινοξύ.

4.2.7 Η ΣΥΝΑΡΤΗΣΗ "main"

Οι πραγματικές συντεταγμένες των κορυφών του lattice μπορούν να βρεθούν εύκολα σαν συνάρτηση της κορυφής K της οποίας οι συντεταγμένες έχουν οριστικοποιηθεί και είναι ίσες με τις συντεταγμένες του πρώτου αμινοξέος. Τη διαδικασία αυτή εκτελεί η συνάρτηση "main". Στο πρόγραμμά μας οι πραγματικές συντεταγμένες αποθηκεύονται σε έναν τρισδιάστατο πίνακα LAT_ARR. Ορίζουμε τις συντεταγμένες θέσης της κορυφή K (p_p-1, p_p-1, p_p-1).

ΠΙΝΑΚΑΣ LAT_ARR

Ο πίνακας LAT_ARR είναι τρισδιάστατος και το μέγεθος (l_l) κάθε διάστασής του είναι $l\_l = (2 * p\_p) - 1$. Ο πίνακας αυτός είναι μία δομή που αποτελείται από τέσσερα πεδία. Στα τρία πρώτα πεδία $(LAT\_ARR.x, LAT\_ARR.y, LAT\_ARR.z)$ αποθηκεύεται ένας δεκαδικός αριθμός, ενώ στο τέταρτο πεδίο $(LAT\_ARR.var)$ αποθηκεύεται ένας ακέραιος.

Επιστρέφοντας στη "main", η συνάρτηση αποθηκεύει στον πίνακα LAT_ARR τις πραγματικές συντεταγμένες των κορυφών με τον παρακάτω τρόπο:

$$LAT_ARR[i][j][k].x = LAT_ARR[p\_p-1][p\_p-1][p\_p-1] + (p\_p-1-i) + L_x$$

$$LAT_ARR[i][j][k].y = LAT_ARR[p\_p-1][p\_p-1][p\_p-1] + (p\_p-1-i) + L_y$$

$$LAT_ARR[i][j][k].z = LAT_ARR[p\_p-1][p\_p-1][p\_p-1] + (p\_p-1-i) + L_z$$

Στη συνέχεια εφόσον είναι γνωστές οι υποψήφιες κορυφές (q_x^i, q_y^i, q_z^i) στις οποίες μπορεί να τοποθετηθεί το αμινοξύ p_i δημιουργούνται μεταβλητές $(p_i, (x, y, z))$ για το p_i, τόσες όσες και οι υποψήφιες κορυφές για το p_i στο lattice. Επόμενο βήμα της "main είναι η δημιουργία της Ευκλείδιας απόστασης μεταξύ του p_i και κάθε τέτοιας κορυφής (x, y, z), έτσι με αυτόν τον τρόπο δημιουργείται η αντικειμενική συνάρτηση του προβλήματος. Οι υπόλοιποι περιορισμοί μπορούν να υπολογιστούν εύκολα (Βλέπε Σχέσεις 2.2, 2.3 και 2.4).

4.2.8 ΤΕΧΝΙΚΗ ΑΝΑΛΥΣΗ ΤΩΝ ΣΥΝΑΡΤΗΣΕΩΝ "ln_create" ΚΑΙ "find_new_node" - ΠΡΩΤΕΙΝΗ 5 ΑΜΙΝΟΞΕΩΝ

Στο Υποκεφάλαιο αυτό περιγράφουμε σε τεχνικό επίπεδο την εκτέλεση των συναρτήσεων "ln_create" και "find_new_node" με βάση το πρόγραμμα που δημιουργήσαμε σε γλώσσα C.

ΠΡΩΤΟ ΑΜΙΝΟΞΥ

Στην περίπτωση του πρώτου αμινοξέος, το πρόγραμμα καλεί τη συνάρτηση "ln_create" που παίρνει ως όρισμα την πρώτη γραμμή του πίνακα (B[0]), έναν βοηθητικό πίνακα C και το πλήθος των αμινοξέων p_p. Τα πεδία της γραμμής αυτής του πίνακα B έχουν τις τιμές B.info=0 και B.Head=NULL (Βλέπε Υποκεφάλαιο 4.2.3). Επιπλέον ο πίνακας C είναι μηδενικός, δηλαδή όλα τα στοιχεία του έχουν την τιμή μηδέν (0).

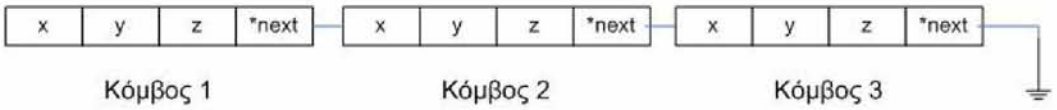
1. Η "ln_create", χρησιμοποιεί μία βοηθητική μεταβλητή που ονομάζεται empty. Με αυτή τη μεταβλητή ελέγχει αν ο πίνακας C είναι μηδενικός. Αν είναι μηδενικός, τότε empty=0, διαφορετικά empty=1.

2. Στη συνέχεια, η "ln_create" ελέγχει αν αληθεύει η σχέση:
   $((B->Head == NULL)\&\&(empty == 0))$,
   δηλαδή αν ο δείκτης B.Head δεν δείχνει σε λίστα και ταυτόχρονα αν ο πίνακας C είναι μηδενικός. Στην περίπτωση του πρώτου αμινοξέος η παραπάνω σχέση ισχύει. Η "ln_create" "καταλαβαίνει" ότι πρόκειται για το πρώτο αμινοξύ της πρωτεΐνης και χρησιμοποιεί μία νέα βοηθητική μεταβλητή που ονομάζεται middle. Στη μεταβλητή αυτή δίνει την τιμή middle=p_p-1. Οι συντεταγμένες θέσης (l_z, l_y, l_z) της κεντρικής κορυφής K του lattice παίρνουν την τιμή middle $(l_x = l_y = l_z = middle)$.

3. Η συνάρτηση "ln_create" δεσμεύει μνήμη με τη βοήθεια της malloc και δημιουργεί μία λίστα.

   Η ΔΟΜΗ ΤΗΣ ΛΙΣΤΑΣ

   Η λίστα αυτή αποτελείται από κόμβους. Κάθε κόμβος είναι μία δομή που αποτελείται από τέσσερα πεδία. Στα τρία πρώτα πεδία (x, y, z) αποθηκεύονται ακέραιες μεταβλητές που αντιπροσωπεύουν τις συντεταγμένες θέσης μίας κορυφής του lattice, ενώ το τέταρτο πεδίο $(next)$ είναι ένας δείκτης. Ο δείκτης κάθε κόμβου δείχνει στον επόμενο κόμβο της λίστας (εξαιρείται ο δείκτης του τελευταίου κόμβου της λίστας, Βλέπε Σχήμα 4.5).



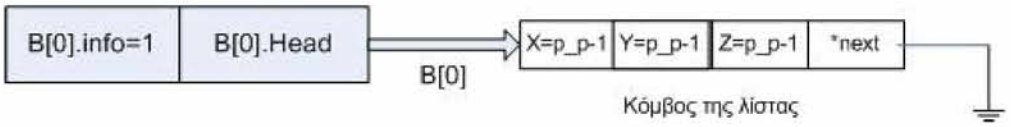
Σχήμα 4.5: Κόμβοι της λίστας που δημιουργεί η Συνάρτηση "ln_create"

   Στον πρώτο κόμβο της λίστας η "ln_create" αποθηκεύει τις συντεταγμένες θέσης της κορυφής K. Συνδέει τον κόμβο αυτό που δημιούργησε με τη γραμμή του πίνακα B[0] κάνοντας τον δείκτη B.Head του πίνακα να δείχνει στον κόμβο.

4. Ταυτόχρονα, αυξάνει στον πίνακα B την τιμή της μεταβλητής B.info κατά μία μονάδα, εφόσον πρόσθεσε έναν κόμβο στη λίστα.

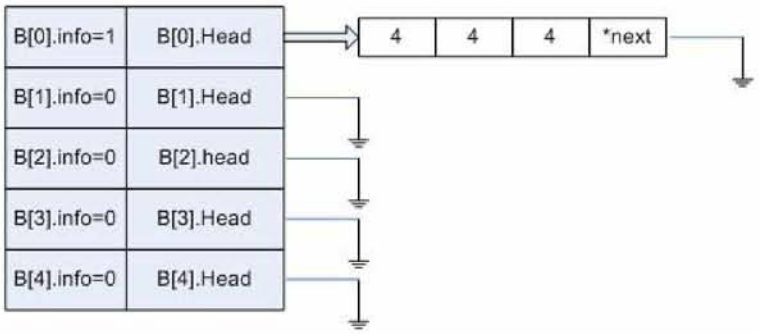
Απαριθμώντας τις κορυφές ενός τρισδιάστατου lattice από την κορυφή (0,0,0) έως την κορυφή $(2p\_p-2, 2p\_p-2, 2p\_p-2)$, μετά την κλήση της συνάρτησης "ln_create" για το πρώτο αμινοξύ (P_1), ο δείκτης B[0].Head θα δείχνει σε μία λίστα που θα αποτελείται από έναν κόμβο, ενώ ταυτόχρονα η μεταβλητή B[0].info θα ισούται με τον αριθμό ένα (1) (Βλέπε Σχήμα 4.6). Όπως θα φανεί αργότερα, οι πραγματικές συντεταγμένες της

μεσαίας αυτής κορυφής στην οποία τοποθετείται το πρώτο αμινοξύ συμπίπτουν με τις συντεταγμένες του πρώτου αμινοξέος.



Σχήμα 4.6: Λίστα που δημιουργεί η Συνάρτηση "ln_create" για το πρώτο αμινοξύ

Για παράδειγμα, σε μία πρωτεΐνη που αποτελείται από 5 αμινοξέα ($p\_p = 5$), το πρώτο αμινοξύ σύμφωνα με όσα αναφέραμε παραπάνω θα τοποθετηθεί στην κορυφή του lattice με συντεταγμένες θέσης (4,4,4) (Βλέπε Σχήμα 4.7).



Σχήμα 4.7: Π.χ. 5 αμινοξέω - Στιγμιότυπο του Πίνακα B μετά την κλήση της "ln_create" για το πρώτο αμινοξύ

ΔΕΥΤΕΡΟ ΑΜΙΝΟΞΥ

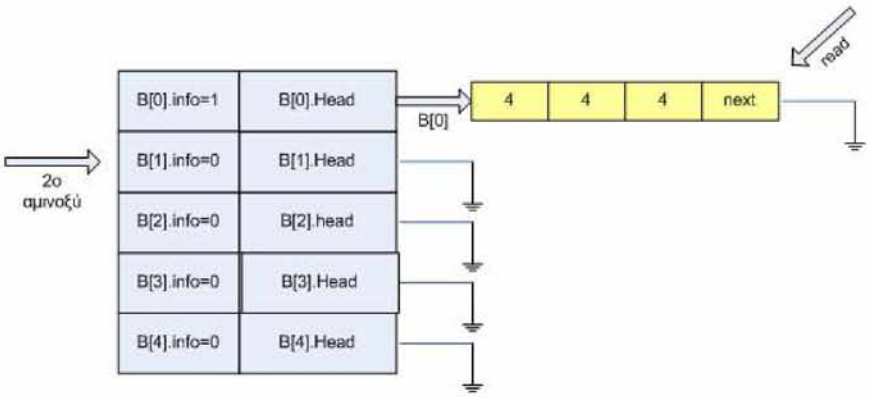
Μετά την τοποθέτηση του πρώτου αμινοξέος, σειρά έχει η εύρεση των πιθανών κορυφών τοποθέτησης του δεύτερου αμινοξέος. Το πρόγραμμα καλεί πρώτα τη συνάρτηση "find_new_node". Η "find_new_node" παίρνει ως ορίσματα μία βοηθητική δομή read, έναν βοηθητικό πίνακα C και τη λίστα του πρώτου αμινοξέος. Η δομή read ισούται με το δείκτη B[0].Head, άρα δείχνει στον πρώτο κόμβο της λίστας που αντιστοιχεί στο πρώτο αμινοξύ, και ο πίνακας C είναι μηδενικός.

Η ΔΟΜΗ read

Η βοηθητική δομή read έχει την ίδια ακριβώς σύσταση με τη δομή ενός κόμβου της λίστας (Βλέπε Υποκεφάλαιο 4.2.8). Όταν το πρόγραμμα ξεκινάει τη μελέτη του i-οστού αμινοξέος, η δομή read ισούται με το δείκτη B[i-1].Head. Για παράδειγμα, κατά τη μελέτη των πιθανών κορυφών του lattice που μπορεί να τοποθετηθεί το δεύτερο αμινοξύ, η δομή read ισούται με το δείκτη B[0].Head (Βλέπε Σχήμα 4.8).

ΠΙΝΑΚΑΣ C

Ο βοηθητικός πίνακας C είναι ένας δισδιάστατος πίνακας. Το μέγεθος της πρώτης διάστασης είναι πάντα ίσο με 6 γιατί κάθε κορυφή ενός τρισδιάστατου lattice έχει έξι γειτονικές κορυφές. Το μέγεθος της δεύτερης διάστασης του πίνακα είναι πάντα ίσο με 3, όσο και το πλήθος των συντεταγμένων θέσης μίας κορυφής (l_x, l_y, l_z).



Σχήμα 4.8: Στιγμιότυπο του πίνακα B πριν την κλήση της συνάρτησης "find_new_node" για το δεύτερο αμινοξύ

1. Η "find_new_node" ελέγχει αν η μεταβλητή B.info=1. Αν αυτό ισχύει τότε η "find_new_node" "καταλαβαίνει" πως το προηγούμενο αμινοξύ ήταν το πρώτο. (Υπενθυμίζουμε πως το πρώτο αμινοξύ είναι το μοναδικό που θα τοποθετηθεί αυστηρά σε μία κορυφή πάνω στο lattice και άρα το μοναδικό για το οποίο ισχύει B[0].info=1).
2. Χρησιμοποιεί τη δομή read για να διαβάσει τις συντεταγμένες θέσης της κορυφής K που τοποθετήθηκε το πρώτο αμινοξύ (Βλέπε Σχήμα 4.8)
3. Με βάση τις συντεταγμένες θέσης της κορυφής K $(4, 4, 4,)$ βρίσκει και αποθηκεύει στον πίνακα C τις συντεταγμένες θέσης των 6 γειτονικών κορυφών της K σύμφωνα με τον Περιορισμό 3. Οι συντεταγμένες αυτές απεικονίζονται στον Πίνακα 4.3 και 4.4.

1^o αμινοξύ	2^o αμινοξύ
$(p\_p-1,\ p\_p-1,\ p\_p-1)$	$(p\_p-2,\ p\_p-1,\ p\_p-1)$
	$(p\_p,\ p\_p-1,\ p\_p-1)$
	$(p\_p-1,\ p\_p-2,\ p\_p-1)$
	$(p\_p-1,\ p\_p,\ p\_p-1)$
	$(p\_p-1,\ p\_p-1,\ p\_p-2)$
	$(p\_p-1,\ p\_p-1,\ p\_p)$

Πίνακας 4.3: Πιθανές κορυφές του lattice για το δεύτερο αμινοξύ

1^o αμινοξύ	2^o αμινοξύ
(4, 4, 4)	(3, 4, 4)
	(5, 4, 4)
	(4, 3, 4)
	(4, 5, 4)
	(4, 4, 3)
	(4, 4, 5)

Πίνακας 4.4: Πιθανές κορυφές του lattice για το δεύτερο αμινοξύ (αριθμητικό παράδειγμα)

Στη συνέχεια εκτελείται η "ln_create" που παίρνει ως όρισμα τη δεύτερη γραμμή του πίνακα (B[1]) και τον παραπάνω πίνακα C.

1. Η "ln_create" με τη βοήθεια της μεταβλητής empty ελέγχει αν ο πίνακας C είναι μηδενικός ή όχι. Στην περίπτωση του δεύτερου αμινοξέος η "find_new_node" έχει δώσει τιμές στον πίνακα οπότε δεν είναι μηδενικός. Για το λόγο αυτό empty=1.

2. Στη συνέχεια, η "ln_create" ελέγχει αν αληθεύει η σχέση:

   $((B->Head == NULL)\&\&(empty == 0))$

   Στην περίπτωση του δεύτερου αμινοξέος όμως empty=1 άρα η παραπάνω σχέση δεν ισχύει. Η "ln_create" συνεχίζει τον έλεγχο στην επόμενη σχέση:

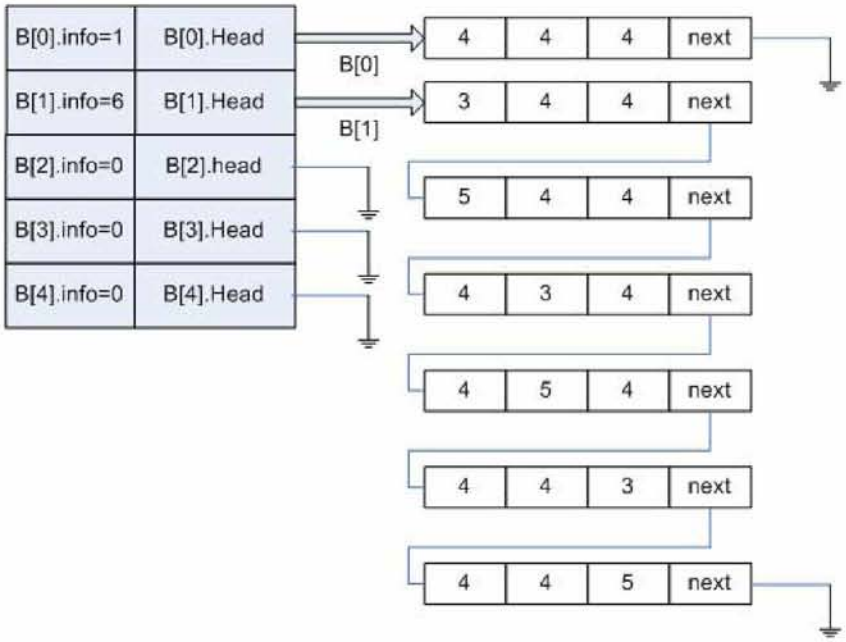
   $((B->Head == NULL)\&\&(empty == 1))$,

   η οποία ισχύει για το συγκεκριμένο αμινοξύ. Η "ln_create" "καταλαβαίνει" ότι πρόκειται να δημιουργήσει μία λίστα με περισσότερους από έναν κόμβους. Σε

κάθε νεό κόμβο που δημιουργεί καταγράφει τα στοιχεία διαφορετικής κάθε φορά γραμμής του πίνακα C, αρκεί αυτά να μην έχουν καταγραφεί ξανά σε προηγούμενο κόμβο.

3. Συνδέει τη λίστα στο δείκτη B[1] και αυξάνει τη μεταβλητή B[1].info τόσες μονάδες όσοι είναι και οι κόμβοι της λίστας.

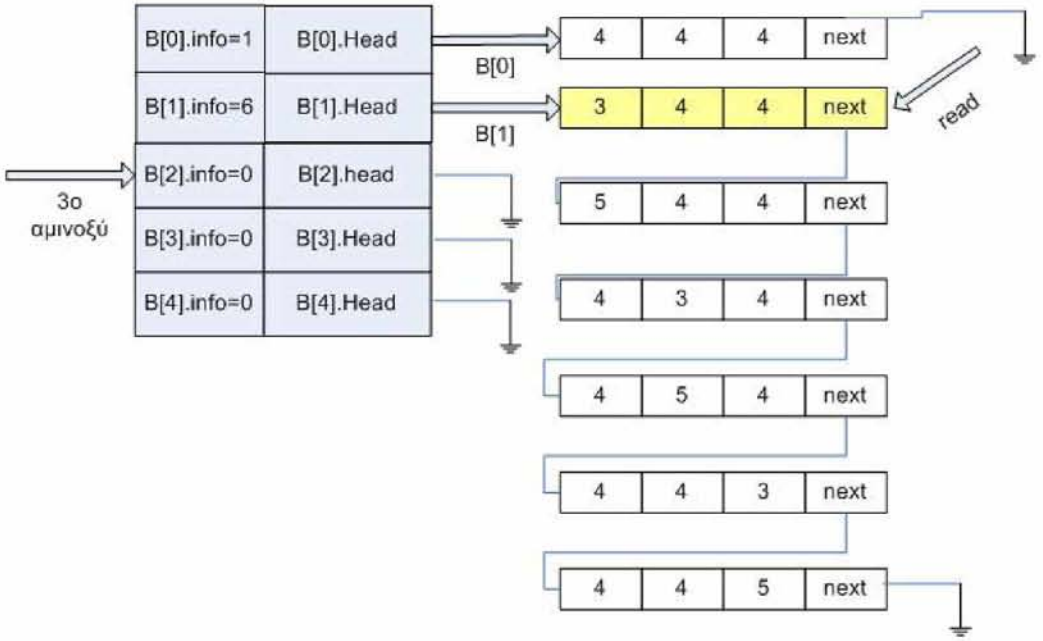
   Το Σχήμα 4.9 δείχνει ποια θα είναι η δομή του πίνακα B μετά την κλήση των συναρτήσεων "find_new_node" και "ln_create" για το δεύτερο αμινοξύ μίας πρωτεΐνης που αποτελείται συνολικά από 5 αμινοξέα.



Σχήμα 4.9: Π.χ. 5 αμινοξέων - Στιγμιότυπο του Πίνακα B μετά την κλήση των συναρτήσεων "find_new_node" και "ln_create" για το δεύτερο αμινοξύ

ΤΡΙΤΟ ΑΜΙΝΟΞΥ

Η διαδικασία εύρεσης των πιθανών κορυφών τοποθέτησης του τρίτου κατά σειρά αμινοξέος της πρωτεΐνης που θα περιγράψουμε στη συνέχεια, είναι ακριβώς ίδια και για τα υπόλοιπα αμινοξέα της πρωτεΐνης. Το πρόγραμμα καλεί πρώτα να εκτελεστεί η συνάρτηση "find_new_node". Η "find_new_node" δέχεται ως όρισμα τη δομή read που δείχνει στον πρώτο κόμβο της λίστας που αντιστοιχεί στο δεύτερο αμινοξύ (B[1].Head, Βλέπε Σχήμα 4.10) καθώς επίσης και τον βοηθητικό πίνακα C.



Σχήμα 4.10: Δεδομένα που επεξεργάζεται η δομή read κατά την πρώτη κλήση της συνάρτησης "find_new_node" για το τρίτο αμινοξύ

1. Η "find_new_node" αρχικοποιεί τον πίνακα C ώστε να μπορέσει να αποθηκεύσει στη συνέχεια νέες συντεταγμένες θέσης. Η διαδικασία αυτή γίνεται καθαρά για τυπικούς λόγους και με σκοπό να αποφύγουμε διάφορα λάθη κατά την εύρεση των συντεταγμένων θέσης.

2. Στη συνέχεια η "find_new_node" ελέγχει αν η μεταβλητή B.info έχει την τιμή 1.
   $B.info == 1$
   Η σχέση αυτή ισχύει μόνο για το πρώτο αμινοξύ. Άρα η "find_new_node" απορρίπτει την περίπτωση αυτή. Στη συνέχεια η συνάρτηση ελέγχει αν αληθεύει η σχέση:
   $((B->info! = 1)\&\&(read! = NULL))$

2^o αμινοξύ	3^o αμινοξύ
(3, 4, 4)	(2, 4, 4)
	(4, 4, 4)
	(3, 3, 4)
	(3, 5, 4)
	(3, 4, 3)
	(3, 4, 5)

Πίνακας 4.5: Πιθανές κορυφές του lattice για το τρίτο αμινοξύ κατά την πρώτη κλήση της συνάρτησης "find_new_node"

Δηλαδή, αν η μεταβλητή B.info δεν έχει την τιμή 1 και ταυτόχρονα αν η δομή read δείχνει σε κάποια λίστα, σχέση που ισχύει.

3. Η "find_new_node" με τη βοήθεια της δομής read διαβάζει τον πρώτο κόμβο της λίστας που αντιστοιχεί στο δεύτερο αμινοξύ. Με βάση την κορυφή στην οποία ανήκουν οι συντεταγμένες θέσης που είναι αποθηκευμένες στον κόμβο αυτόν, βρίσκει τις συντεταγμένες θέσης των γειτονικών της κορυφών πάνω στο lattice. Ο Πίνακας 4.5 δείχνει ποιες ακριβώς θα είναι αυτές οι συντεταγμένες θέσης του τρίτου αμινοξέος.

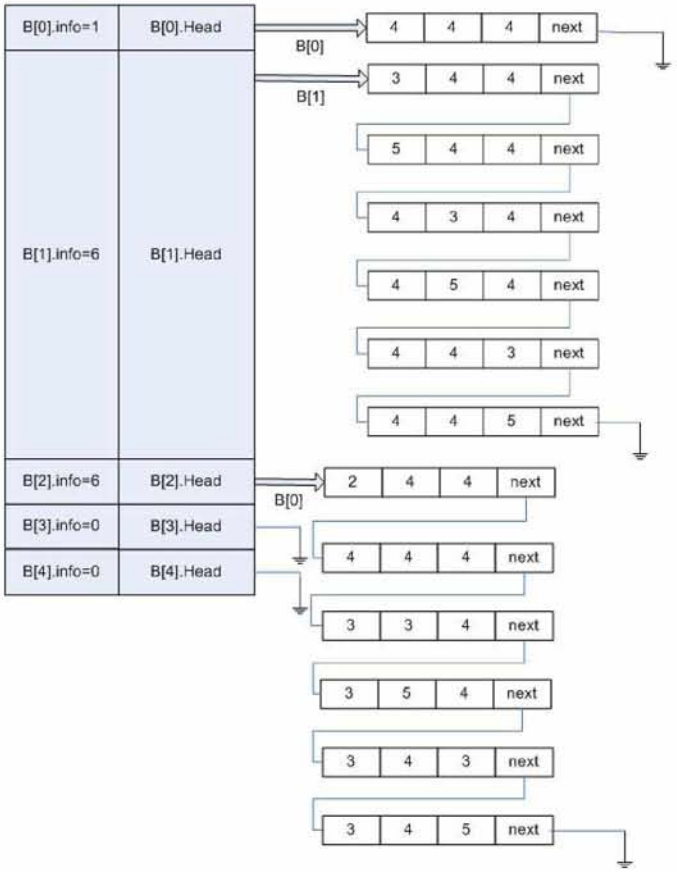
Στη συνέχεια εκτελείται η "ln_create" που παίρνει ως όρισμα την τρίτη γραμμή του πίνακα (B[1]) και τον παραπάνω πίνακα C.

1. Η "ln_create" με τη βοήθεια της μεταβλητής empty ελέγχει αν ο πίνακας C είναι μηδενικός ή όχι. Στην περίπτωση αυτή η "find_new_node" έχει δώσει τιμές στον πίνακα οπότε δεν είναι μηδενικός. Για το λόγο αυτό empty=1.

2. Στη συνέχεια, η "ln_create" ελέγχει αν αληθεύει η σχέση:
   $((B->Head == NULL)\&\&(empty == 0))$
   Στην προκειμένη περίπτωση empty=1 άρα η παραπάνω σχέση δεν ισχύει. Η "ln_create" συνεχίζει τον έλεγχο στην επόμενη σχέση
   $((B->Head == NULL)\&\&(empty == 1))$,
   η οποία ισχύει. Η "ln_create" δημιουργεί μία λίστα και σε κάθε νέο κόμβο που δημιουργεί καταγράφει τα στοιχεία διαφορετικής κάθε φορά γραμμής του πίνακα C, αρκεί αυτά να μην έχουν καταγραφεί ξανά σε προηγούμενο κόμβο.

3. Συνδέει τη λίστα στο δείκτη B[1] και αυξάνει τη μεταβλητή B[1].info τόσες μονάδες όσοι είναι και οι κόμβοι της λίστας.

Το Σχήμα 4.11 περιγράφει ποια θα είναι η δομή του πίνακα B μετά την πρώτη κλήση των συναρτήσεων "find_new_node" και "ln_create" για το τρίτο αμινοξύ.



Σχήμα 4.11: Στιγμιότυπο του πίνακα B μετά την πρώτη κλήση των συναρτήσεων "find_new_node" και "ln_create" για το τρίτο αμινοξύ

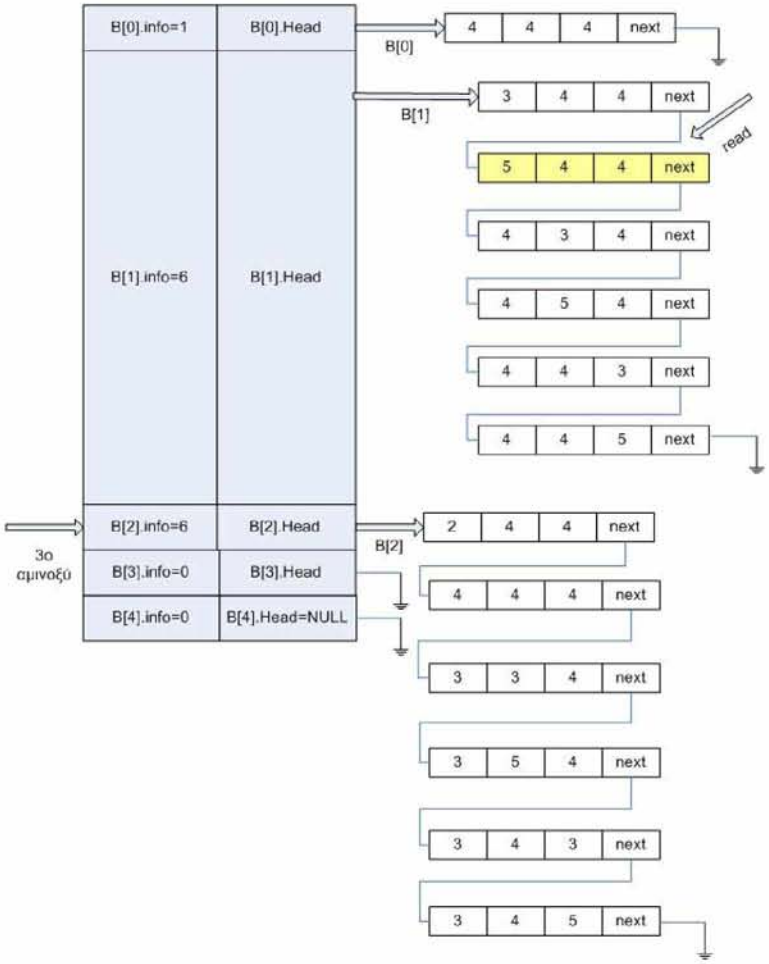
Στη συνέχεια το πρόγραμμα καλεί και πάλι τη συνάρτηση "find_new_node".

Η "find_new_node" χρησιμοποιεί τη δομή read η οποία διαβάζει τον επόμενο κόμβο της λίστας που αντιστοιχεί στο δεύτερο αμινοξύ. Η διαδικασία που ακολουθεί είναι ίδια με αυτή που μόλις περιγράψαμε για την πρώτη κλήση των συναρτήσεων για το τρίτο αμινοξύ (Βλέπε Σχήματα 4.12,4.13, 4.14, 4.15 και 4.16).

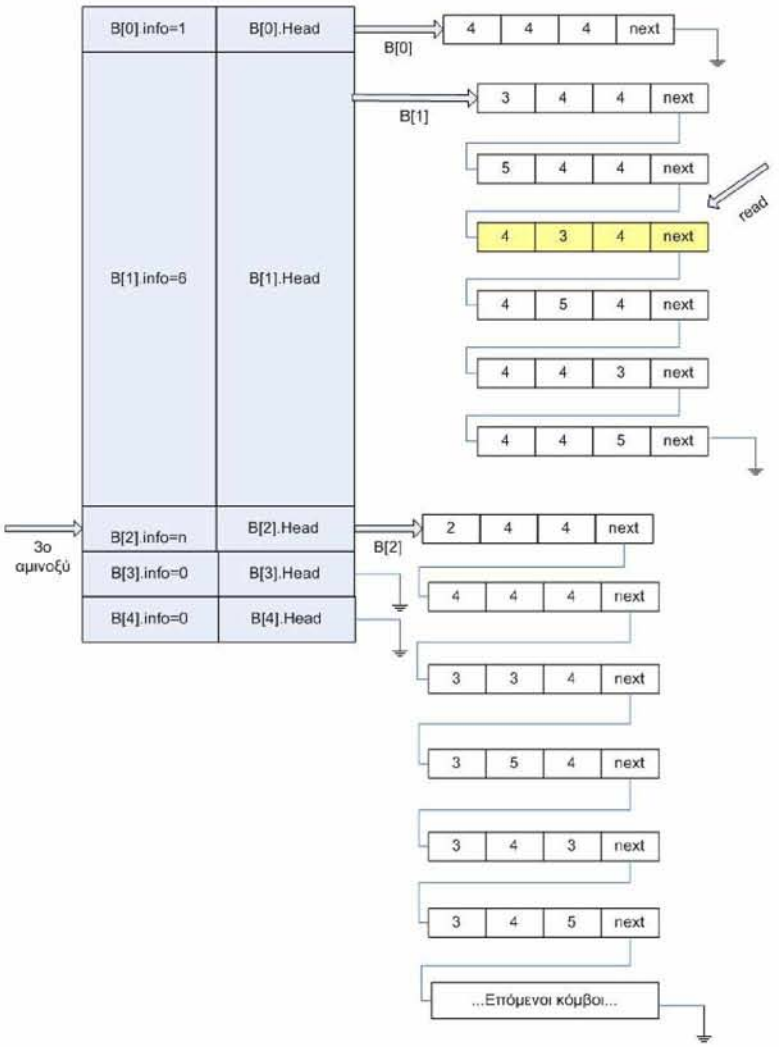
Τέλος, μόλις η δομή read διαβάσει τον τελευταίο κόμβο της λίστας του δεύτερου αμινοξέος (Βλέπε Σχήμα 4.16) και εκτελεστούν οι συναρτήσεις "find_new_node" και "ln_create", το πρόγραμμα συνεχίζει την αναζήτηση των πιθανών κορυφών για τα επό-

μενα αμινοξέα ακολουθώντας την ίδια ακριβώς διαδικασία με αυτή που πραγματοποιήσε για το τρίτο αμινοξύ (Βλέπε Σχήμα 4.17).

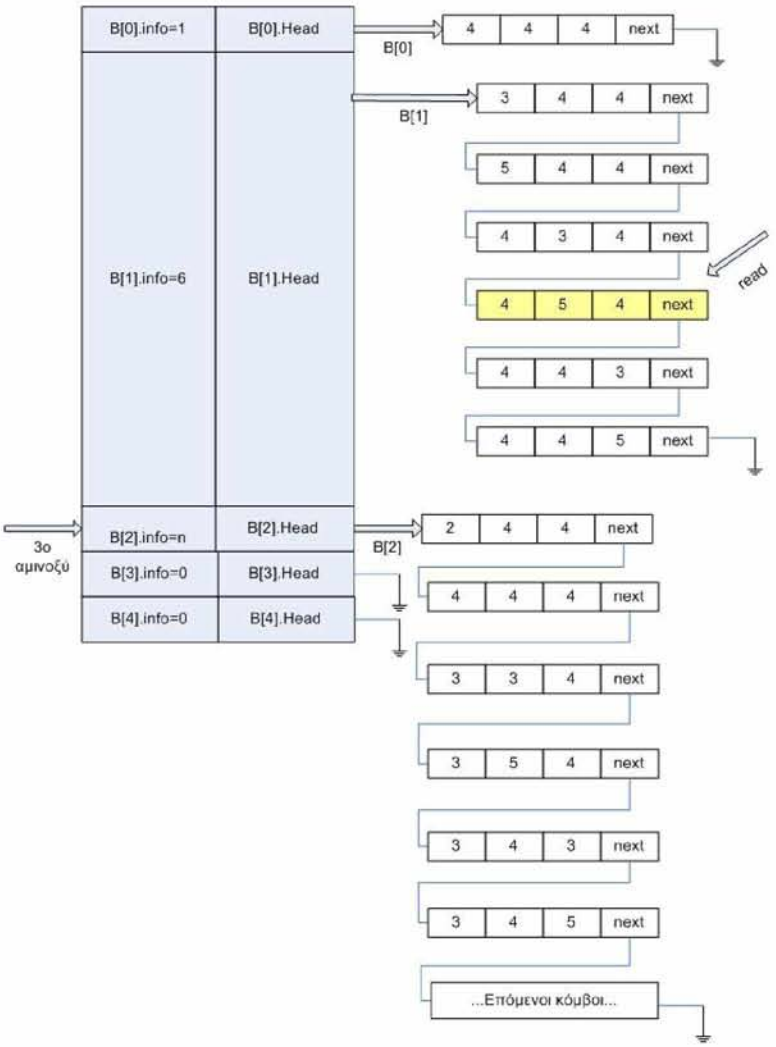
Οι πίνακες 4.6 και 4.7 απεικονίζουν τις συντεταγμένες θέσης για όλες τις πιθανές κορυφές του lattice που μπορεί να τοποθετηθούν τα τρία πρώτα αμινοξέα. Πιο αναλυτικά, κάθε στήλη αντιστοιχεί στη λίστα που φτιάξαμε για κάθε ένα αμινοξύ. Η δομή των πινάκων είναι τέτοια που μας επιτρέπει να δούμε παραδείγματος χάρην πως προκύπτει η κορυφή με συντεταγμένες θέσης (2,4,4) του τρίτου αμινοξέος σε σχέση με τις συντεταγμένες των προηγούμενων δύο αμινοξέων. Οι γραμμές με τις συντεταγμένες στις οποίες σημειώνουμε πως "δεν τοποθετούνται" αντιστοιχούν σε συντεταγμένες που έχουν ήδη τοποθετηθεί στη λίστα και απορρίπτονται από την "ln_create".



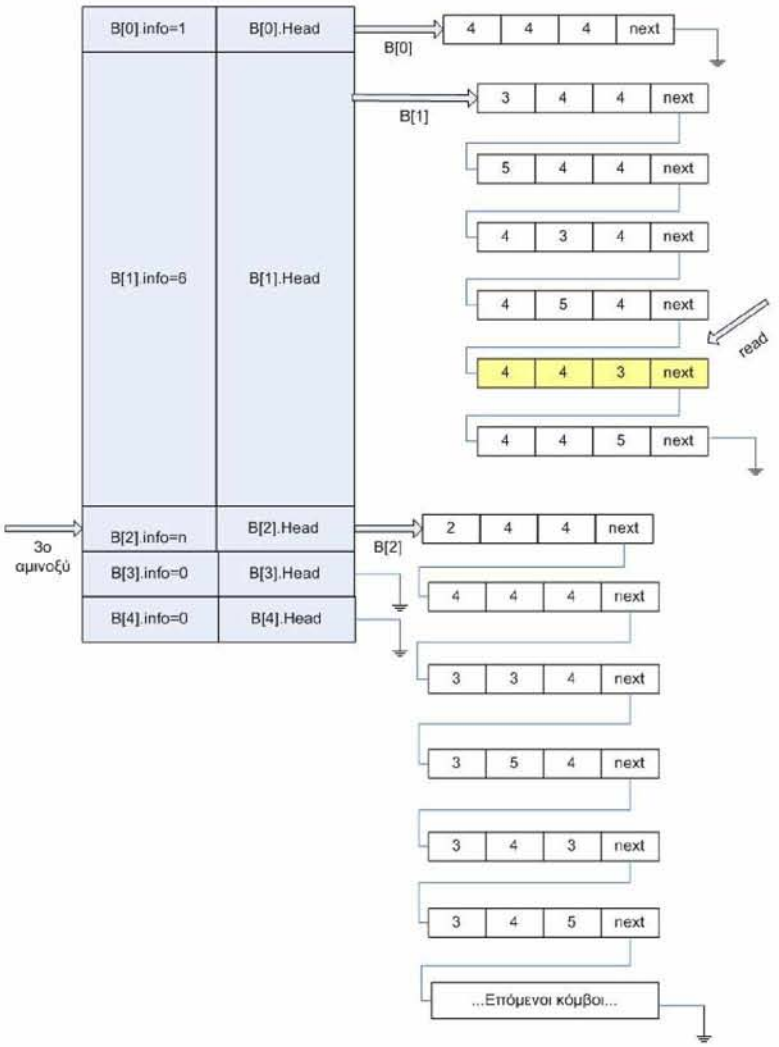
Σχήμα 4.12: Δεδομένα που επεξεργάζεται η δομή read κατά την δεύτερη κλήση της συνάρτησης "find_new_node" για το τρίτο αμινοξύ



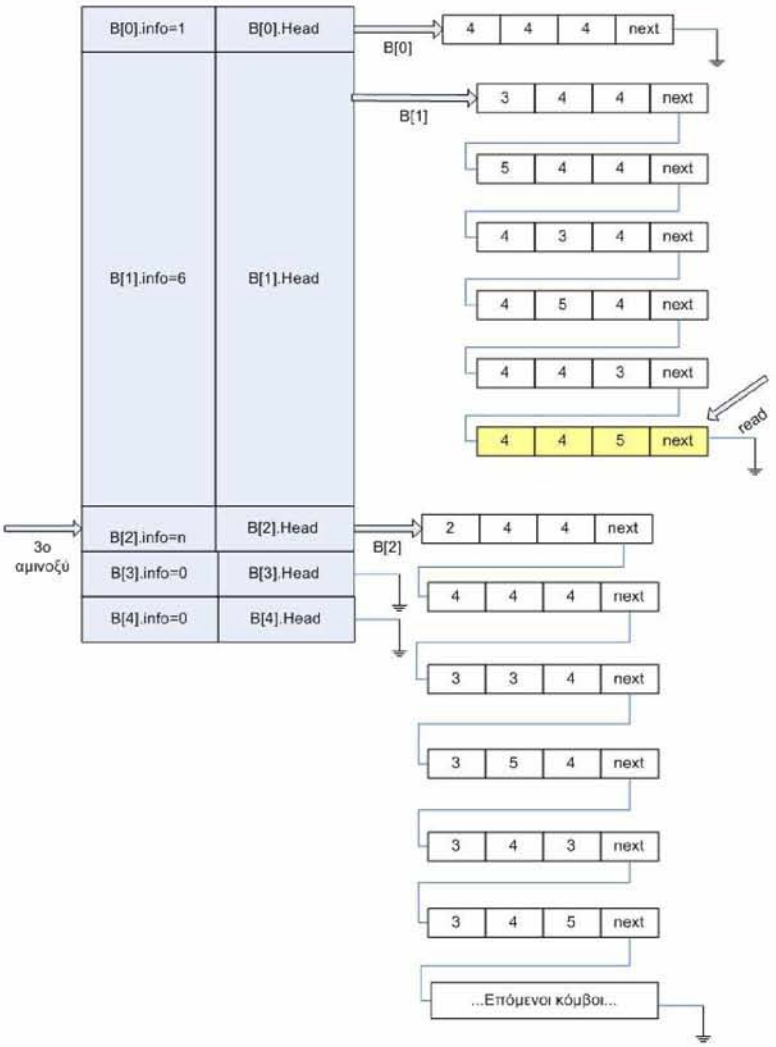
Σχήμα 4.13: Δεδομένα που επεξεργάζεται η δομή read κατά την τρίτη κλήση της συνάρτησης "find_new_node" για το τρίτο αμινοξύ



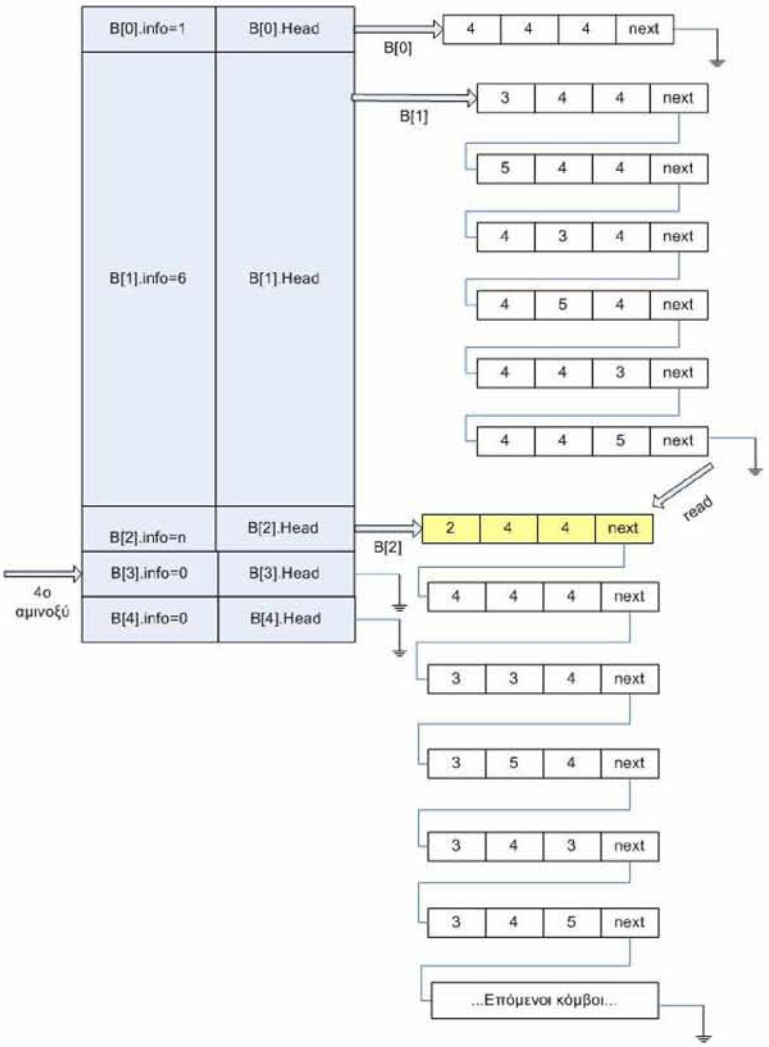
Σχήμα 4.14: Δεδομένα που επεξεργάζεται η δομή read κατά την τέταρτη κλήση της συνάρτησης "find_new_node" για το τρίτο αμινοξύ



Σχήμα 4.15: Δεδομένα που επεξεργάζεται η δομή read κατά την πέμπτη κλήση της συνάρτησης "find_new_node" για το τρίτο αμινοξύ



Σχήμα 4.16: Δεδομένα που επεξεργάζεται η δομή read κατά την έκτη κλήση της συνάρτησης "find_new_node" για το τρίτο αμινοξύ



Σχήμα 4.17: Στιγμιότυπο του πίνακα B πριν από την επεξεργασία του τέταρτου αμινοξέος

1^o αμινοξύ	2^o αμινοξύ	3^o αμινοξύ
(4, 4, 4)	(3, 4, 4)	(2, 4, 4)
		(4, 4, 4)
		(3, 3, 4)
		(3, 5, 4)
		(3, 4, 3)
		(3, 4, 5)
	(5, 4, 4)	(4, 4, 4) δεν τοποθετείται
		(6, 4, 4)
		(5, 3, 4)
		(5, 5, 4)
		(5, 4, 3)
		(5, 4, 5)
	(4, 3, 4)	(3, 3, 4) δεν τοποθετείται
		(5, 3, 4) δεν τοποθετείται
		(4, 2, 4)
		(4, 4, 4) δεν τοποθετείται
		(4, 3, 3)
		(4, 3, 5)
	(4, 5, 4)	(3, 5, 4) δεν τοποθετείται
		(5, 5, 4) δεν τοποθετείται
		(4, 4, 4) δεν τοποθετείται
		(4, 6, 4)
		(4, 5, 3)
		(4, 5, 5)

Πίνακας 4.6: Τοποθέτηση 3 αμινοξέων - Συντεταγμένες θέσης των πιθανών κορυφών πάνω στο lattice 1-2

1^o αμινοξύ	2^o αμινοξύ	3^o αμινοξύ
(4, 4, 4)	(4, 4, 3)	(3, 4, 3) δεν τοποθετείται
		(5, 4, 3) δεν τοποθετείται
		(4, 3, 3) δεν τοποθετείται
		(4, 5, 3) δεν τοποθετείται
		(4, 4, 2)
		(4, 4, 4) δεν τοποθετείται
	(4, 4, 5)	(3, 4, 5) δεν τοποθετείται
		(5, 4, 5) δεν τοποθετείται
		(4, 3, 5) δεν τοποθετείται
		(4, 5, 5) δεν τοποθετείται
		(4, 4, 4) δεν τοποθετείται
		(4, 4, 6)

Πίνακας 4.7: Τοποθέτηση 3 αμινοξέων - Συντεταγμένες θέσης των πιθανών κορυφών πάνω στο lattice 2-2

4.2.9 ΤΕΧΝΙΚΗ ΑΝΑΛΥΣΗ ΤΗΣ ΣΥΝΑΡΤΗΣΗΣ "main" - ΠΡΩΤΕΙΝΗ 5 ΑΜΙΝΟΞΕΩΝ

Η "main" είναι η βασική συνάρτηση του προγράμματος που διαχειρίζεται όλες τις υπόλοιπες συναρτήσεις του προγράμματος. Τα βασικά βήματα του αλγόριθμου που εκτελεί η "main" είναι τα παρακάτω:

1. Αρχικοποιεί τον πίνακα LAT_ARR.

2. Τοποθετεί την κεντρική κορυφή K με συντεταγμένες θέσης (middle, middle, middle) του lattice (Βλέπε Υποκεφάλαιο 4.2.8) πάνω στο πρώτο αμινοξύ την πρωτεΐνης. Για παράδειγμα ο Πίνακας 4.8 περιγράφει τις συντεταγμένες 5 αμινοξέων.

	x	y	z
p_1	15	2	-16
p_2	15	3	-16
p_3	15	3	-17
p_4	15	3	-18
p_5	16	3	-18

Πίνακας 4.8: Π.χ. Συντεταγμένες των 5 αμινοξεών

Οι συντεταγμένες (15,2,-16) του πρώτου αμινοξέος γίνονται συντεταγμένες (l_x, l_y, l_z) της κεντρικής κορυφής K του lattice (η οποία έχει συντεταγμένες θέσης [4][4][4]):

$$LAT\_ARR[4][4][4].x = 15$$

$$LAT\_ARR[4][4][4].y = 2$$

$$LAT\_ARR[4][4][4].z = -16$$

3. Δίνει συντεταγμένες σε όλες τις κορυφές του lattice. Έστω ότι στη συνέχεια η "main" ψάχνει τις πραγματικές συντεταγμένες (l_x, l_y, l_z) της κορυφής [3][4][4].

   (α') Υπολογίζει τη διαφορά των συντεταγμένων θέσης της κεντρικής κορυφής K [4][4][4] από τις συντεταγμένες θέσης της κορυφής [3][4][4].

(β′) Αποθηκεύει τις διαφορές που υπολογίσαμε σε τρεις βοηθητικές μεταβλητές found.x, found.y και found.z.

(γ′) Πολλαπλασιάζει κάθε μία μεταβλητή found με το μήκος της ακμής του αντίστοιχου άξονα του lattice, δηλαδή τη μεταβλητή found.x με το μήκος Lx κ.ο.κ.

(δ′) Τέλος, προσθέτει στο παραπάνω γινόμενο τις πραγματικές συντεταγμένες της κεντρικής κορυφής K. Πιο αναλυτικά η διαδικασία έχει ως εξής:

Υπολογισμός της συντεταγμένης x:

$$found.x = [3] - [4] = -1$$

$$found.x * Lx = (-1) * 2 = -2$$

$$LAT\_ARR[3][4][5].x = LAT\_ARR[4][4][4].x + found.x * Lx = 15 + (-2) = 13$$

Υπολογισμός της συντεταγμένης y:

$$found.y = [4] - [4] = 0$$

$$found.y * Ly = (0 * 1) = 0$$

$$LAT\_ARR[3][4][5].y = LAT\_ARR[4][4][4].y + found.y * Ly = 2 + 0 = 2$$

Υπολογισμός της συντεταγμένης z:

$$found.z = [5] - [4] = 1$$

$$found.z * Lz = (1 * 1) = 1$$

$$LAT\_ARR[3][4][5].z = LAT\_ARR[4][4][4].z + found.z * Lz = (-16) + 1 = -15$$

4. Ανοίγει ένα αρχείο με όνομα input_cplex.lp. Η πρώτη εντολή που γράφει η "main" στο αρχείο αυτό είναι "Minimize obj:".

   Με την εντολή αυτή ο αλγόριθμος του CPLEX καταλαβαίνει πως ακολουθεί η αντικειμενική συνάρτηση του προβλήματος την οποία πρέπει να ελαχιστοποιήσει.

(α′) Υπολογίζει και γράφει την αντικειμενική συνάρτηση στο αρχείο input_cplex.lp (Βλέπε Υποκεφάλαιο 1.6.1, Σχέση 1.5). Έστω p_x, p_y, p_z οι συντεταγμένες του πρώτου αμινοξέος και έστω l_x, l_y, l_z οι συντεταγμένες της κορυφής Κ του lattice όπου τοποθετείται. Η Ευκλείδια απόσταση υπολογίζεται ως εξής:

$$(p_x - l_z)^2 + (p_y - l_y)^2 + (p_z - l_z)^2$$

Κάθε Ευκλείδια απόσταση που υπολογίζει η "main" και καταγράφει στο αρχείο input_cplex.lp συνοδεύεται και από μία μεταβλητή τύπου $X(p_i, (x, y, z))$.

(β′) Γράφει τον Περιορισμό 1 του προβλήματος (Βλέπε Σχέση 2.2). Συγκεκριμένα δίνει πρώτα την εντολή "Subject to:" με την οποία ο αλγόριθμος του CPLEX καταλαβαίνει ότι ακολουθούν οι Περιορισμοί του προβλήματος. Για παράδειγμα έστω πρωτεΐνη 5 αμινοξέων, τότε η "main" θα γράψει στο αρχείο για τον Περιορισμό 1 και τα δύο πρώτα αμινοξέα τα παρακάτω:

$1^ο$ αμινοξύ:

$$X(1, (4, 4, 4)) = 1$$

$2^ο$ αμινοξύ:

$$X(2, (3, 4, 4)) + X(2, (5, 4, 4)) + X(2, (4, 3, 4)) +$$
$$X(2, (4, 5, 4)) + X(2, (4, 4, 3)) + X(2, (4, 4, 5)) = 1$$

(γ′) Χαρακτηρίζει κάθε μία μεταβλητή που χρησιμοποιούμε στο πρόβλημα ως binary δίνοντας την εντολή "Binaries:" με την οποία ο αλγόριθμος του CPLEX καταλαβαίνει πως οι μεταβλητές τύπου $X(p_i, (x, y, z))$ που ακολουθούν είναι δυαδικές (δηλαδή μπορούν να πάρουν δύο τιμές: 0 ή 1).

(δ′) Κλείνει το αρχείο input_cplex.lp δίνοντας την εντολή "quit"

5. Ανοίγει ένα δεύτερο αρχείο με όνομα input_cplex1.lp.

(α′) Γράφει τον Περιορισμό 2 του προβλήματος (Βλέπε Σχέση 2.3). Στον περιορισμό αυτόν ψάχνει να βρει ποια αμινοξέα έχουν στις λίστες τους ίδιους κόμβους. Έστω ότι βρίσκει έναν ίδιο κόμβο στη λίστα του πρώτου και τρίτου αμινοξέος, αυτό θα γραφεί στο αρχείο ως εξής:

$$X(1,(middle,middle,middle))+$$
$$X(3,(middle,middle,middle)) <= 1$$

Αντίθετα για όσους κόμβους βρίσκονται στη λίστα μόνο ενός αμινοξέος η "main" γράφει στο αρχείο π.χ:

$$X(2,(middle-1,middle,middle)) <= 1$$

(β′) Κλείνει το αρχείο input_cplex1.pl δίνοντας την εντολή "quit"

6. Ανοίγει ένα τρίτο αρχείο με όνομα input_cplex2.lp.

(α′) Γράφει τον Περιορισμό 3 του προβλήματος. Για παράδειγμα έστω μία πιθανή κορυφή τοποθέτησης [3][4][4] του δεύτερου αμινοξέος. Σε περίπτωση που το δεύτερο αμινοξύ τοποθετηθεί στην κορυφή [3][4][4] (X(2,(3,4,4))=1), τότε το τρίτο αμινοξύ πρέπει υποχρεωτικά να τοποθετηθεί σε μία από τις γειτονικές κορυφές της. Ενώ αντίθετα αν το δεύτερο αμινοξύ δεν τοποθετηθεί στη κορυφή [3][4][4] (δηλαδή η μεταβλητή X(2(3,4,4))=0), τότε το τρίτο αμινοξύ ίσως να τοποθετηθεί σε κάποια γειτονική της κορυφή αλλά ίσως και όχι. Τα παραπάνω γράφονται ως εξής στο αρχείο:

$$X(3,(2,4,4)) + X(3,(4,4,4))+$$
$$X(3,(3,3,4)) + X(3,(3,5,4))+$$
$$X(3,(3,4,3)) + X(3,(3,4,5)) >= X(2,(3,4,4))$$

Που ισοδυναμεί με τη Σχέση:

$$X(3,(2,4,4)) + X(3,(4,4,4))+$$
$$X(3,(3,3,4)) + X(3,(3,5,4))+$$
$$X(3,(3,4,3)) + X(3,(3,4,5)) - X(2,(3,4,4)) >= 0$$

(β′) Κλείνει το αρχείο input_cplex2.lp δίνοντας την εντολή "quit"

7. Η έξοδος της συνάρτησης (ισοδυναμεί με την έξοδο του προγράμματος στο χρήστη) είναι τα τρία αρχεία input_cplex.l, input_cplex1.lp και input_cplex2.lp.

Η "main" δημιουργεί τρία διαφορετικά αρχεία αντί για ένα, κυρίως για λόγους μνήμης του υπολογιστή, καθώς χρειάζονται πολλά GB μνήμης ώστε να δημιουργηθεί ένα μόνο αρχείο. Τα αρχεία αυτά περιγράφουν τον πλήρη ορισμό του PCLF προβλήματος και μπορούν πλέον να μπούνε ως είσοδο στο πρόγραμμα ILOG CPLEX.

Το πλήρες πρόγραμμα που μόλις περιγράψαμε, το οποίο επεξεργάζεται για παράδειγμα τα 5 πρώτα αμινοξέα της πρωτεΐνης $1bw8$ και τα τοποθετεί πάνω στο lattice(3.8-3.8-3.8) βρίσκεται στο Παράρτημα Α′. Στο Παράρτημα Β′ παραθέτουμε τα αρχεία input_cplex.l, input_cplex1.lp και input_cplex2.lp που δημιουργεί το πρόγραμμα για τα 5 αμινοξέα, και τα οποία περιέχουν τον πλήρη ορισμό του PCLF προβλήματος. Επίσης στο Παράρτημα Β′ παραθέτουμε και το αρχείο output1bw8.txt στο οποίο το πρόγραμμα CPLEX καταγράφει τη λύση του PCLF προβλήματος.

Κεφάλαιο 5

ΠΕΙΡΑΜΑΤΙΚΑ ΑΠΟΤΕΛΕΣΜΑΤΑ

Σε αυτό το κεφάλαιο πρόκειται να κάνουμε μία μικρή περιγραφή του τρόπου λειτουργίας του προγράμματος CPLEX και στη συνέχεια θα αναλύσουμε τα πειραματικά αποτελέσματα που προέκυψαν κατά την επίλυση του PCLF προβλήματος από τον αλγόριθμο του CPLEX.

5.1 ΠΑΡΟΥΣΙΑΣΗ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ ILOG CPLEX

Τα πνευματικά δικαιώματα του προγράμματος ILOG CPLEX ανήκουν στην εταιρία IBM, την οποία ευχαριστούμε για τη δωρεάν παραχώρηση της πλήρους έκδοσης του προγράμματος για τις ανάγκες της πτυχιακής αυτής εργασίας [17] [18]. Το πρόγραμμα ILOG CPLEX είναι μία βιβλιοθήκη αλγόριθμων που επιλύουν προβλήματα γραμμικού προγραμματισμού. Το πρόγραμμα λειτουργεί με εντολές που δίνονται από το χρήστη. Το βασικό μενού εντολών του CPLEX περιγράφεται στον Πίνακα 5.1.

Ο λόγος που επιλέξαμε το πρόγραμμα ILOG CPLEX είναι αφενός γιατί είναι ένα πρόγραμμα που δεν έχει περιορισμούς σχετικά με το πλήθος των μεταβλητών και των εξισώσεων, και αφετέρου γιατί δίνει, σε σύγκριση με άλλα προγράμματα, γρήγορα λύση σε ένα πρόβλημα.

Το πρόγραμμα αυτό δέχεται ως είσοδο τα αρχεία input_cplex.lp, input_cplex1.lp και input_cplex2.lp στα οποία καταγράψαμε τον πλήρη ορισμό του PCLF προβλήματος για μία δεδομένη πρωτεΐνη και ένα δεδομένο lattice. Ο αλγόριθμος του CPLEX λύνει το PCLF πρόβλημα και επιστρέφει στο χρήστη το χρόνο που χρειάστηκε για την επίλυσή του και την τιμή της αντικειμενικής συνάρτησης. Η αντικειμενική συνάρτηση αντιστοιχεί στην Ευκλείδια απόσταση. Για το λόγο αυτό χρειάζεται ένας επιπλέον υπολογισμός από το χρήστη για την

add	*add constraints to the problem*
baropt	*solve using barrier algorithm*
change	*change the problem*
conflict	*refine a conflict for an infeasible problem*
display	*display problem, solution, or parameter setting*
enter	*enter a new problem*
feasopt	*find relaxation to an infeasible problem*
help	*provide information on CPLEX command*
mipopt	*solve a mixed integer program*
optimize	*solve the problem*
populate	*get additional solutions for a mixed integer program*
primopt	*solve using the primal method*
quit	*leave CPLEX*
read	*read problem or advanced start information from a file*
set	*set parameters*
tranopt	*solve using the dual method*
tune	*try a variety of parameter setting*
write	*write problem or solution information to a file*
xecute	*execute a command from the operation system*

Πίναχας 5.1: Εντολές CPLEX

τιμή του CRMS (Βλέπε Σχέση 2.5). Φυσικά μπορούμε να δούμε και τις τιμές όλων των μεταβλητών του προβλήματος. Κατά τη διαδικασία επίλυσης του προβλήματος ο χρόνος που απαιτεί ο αλγόριθμος ποικίλει (5-15 λεπτά). Ο απαιτούμενος χρόνος επίλυσης διαφέρει ανάλογα με τη δεδομένη πρωτεΐνη και το δεδομένο lattice.

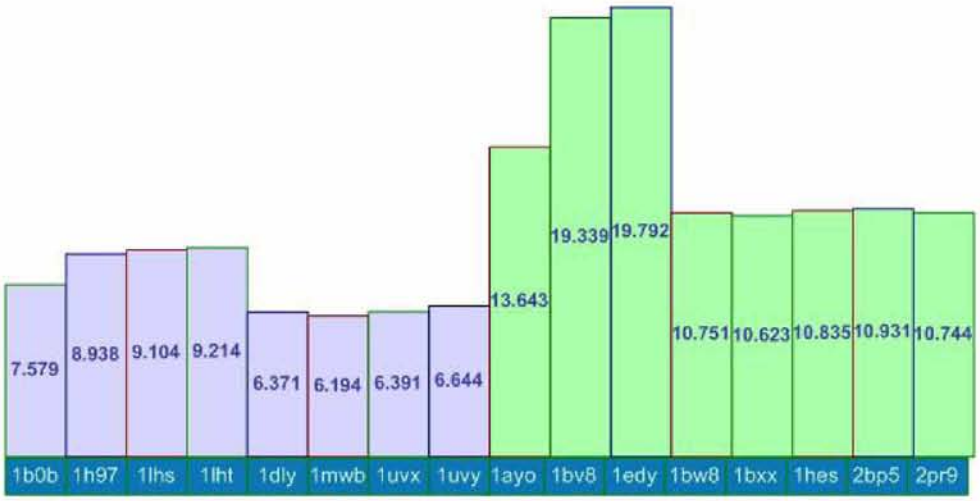
Στα πειράματα που εκτελέσαμε, πήραμε τα πρώτα 20 αμινοξέα από 16 διαφορετικές πρωτεΐνες, εκ των οποίων οι 8 ανήκουν στην ομάδα "beta" (1ayo, 1bv8, 1edy, 1bw8, 1bxx, 1hes, 2bp5 και 2pr9) και οι υπόλοιπες 8 στην ομάδα "alpha" (1b0b, 1h97, 1lhs, 1lht, 1dly, 1mwb, 1uvx και 1uvy).

Κάθε μία από τις ακολουθίες των 20 αμινοξέων την τοποθετήσαμε σε 16 διαφορετικά lattices. Οι διαφορές μεταξύ των lattices ήταν ως προς τα μήκη L_x, L_y, L_z των ακμών στους τρεις άξονες. Πιο συγκεκριμένα χρησιμοποιήσαμε τα παρακάτω lattices: lattice(1-1-1), lattice(2-1-1), lattice(1-2-1), lattice(1-1-2), lattice(2-2-2), lattice(1-2-3), lattice(1-3-2), lattice(2-1-3), lattice(2-3-1), lattice(3-1-2), lattice(3-2-1), lattice(3-1-1), lattice(1-3-1), lattice(1-1-3), lattice(3-3-3) και lattice(3.8-3.8-3.8).

Τα μήκη των ακμών όλων των lattices μετριούνται σε Å. Στη μεθόδο που φτιάξαμε επιλέξαμε lattices με μήκη ακμών κοντά στα 3.8 Å, αφενώς γιατί έχει μία σχετική σημασία για τις πρωτεΐνες και εφετέρου γιατί το PCLF πρόβλημα για απόκλιση CRMS και τρισδιάστατα lattices με μήκη ακμών 3.8 Å, έχει αποδειχτεί ότι είναι NP-complete.

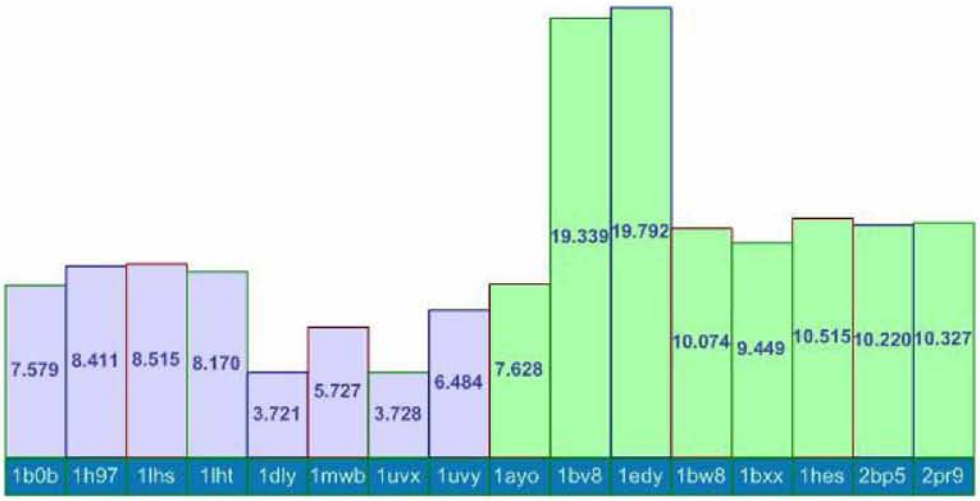
5.2 ΑΝΑΛΥΣΗ ΠΕΙΡΑΜΑΤΙΚΩΝ ΑΠΟΤΕΛΕΣΜΑΤΩΝ

Τα Σχήματα 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 5.10, 5.11, 5.12, 5.13, 5.14, 5.15 και 5.16 που ακολουθούν, περιγράφουν για κάθε ένα lattice ποιες είναι η τιμές του CRMS που υπολογίστηκαν με βάση τα αποτελέσματα που έδωσε ο αλγόριθμος του CPLEX για κάθε μία πρωτεΐνη. Η ομάδα "aplha" συμβολίζεται με γαλάζιο χρώμα, ενώ η ομάδα "beta" με πράσινο.



Σχήμα 5.1: Τιμές των αποκλίσεων CRMS για Lattice 1-1-1

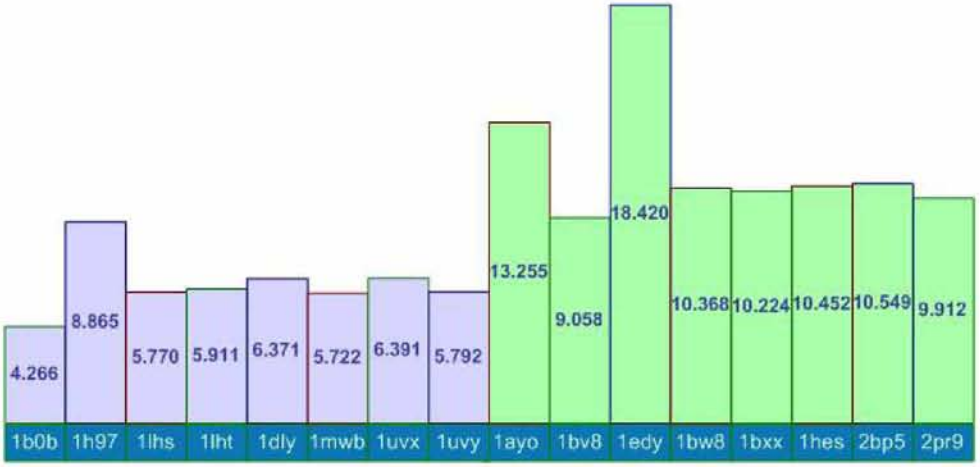
Όπως βλέπουμε στο Σχήμα 5.1 η μικρότερη τιμή για την ομάδα "alpha" είναι 6.194 και η μεγαλύτερη 9.214, δηλαδή το εύρος είναι περίπου 3 μονάδες. Ενώ για την ομάδα "beta" η μικρότερη τιμή είναι 10.623 και η μεγαλύτερη 19.792, δηλαδή το εύρος είναι περίπου 9 μονάδες. Υπολογίσαμε ότι ο μέσος όρος για την ομάδα "alpha" είναι 7.554, ενώ για την ομάδα "beta" είναι 13.332.



Σχήμα 5.2: Τιμές των αποκλίσεων CRMS για Lattice 2-1-1

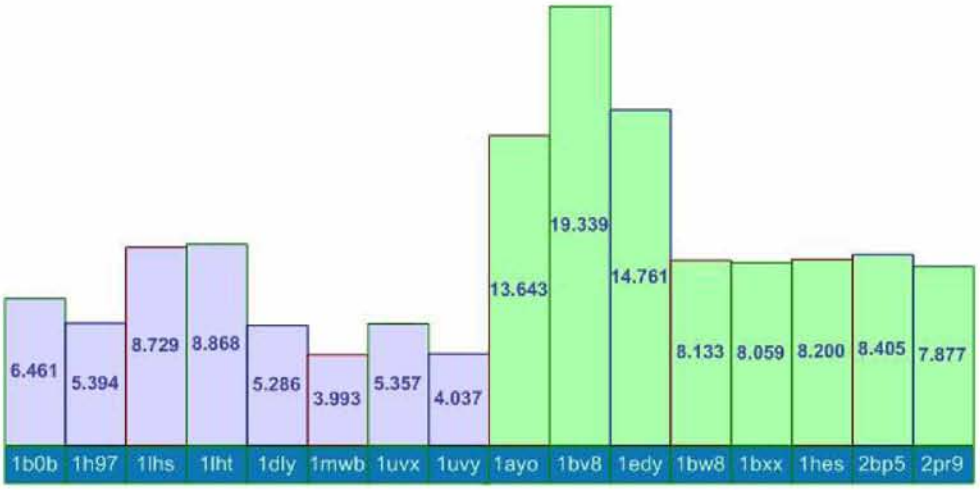
Όπως βλέπουμε στο Σχήμα 5.2 η μικρότερη τιμή για την ομάδα "alpha" είναι 3.721 και η μεγαλύτερη 8.515, δηλαδή το εύρος είναι περίπου 5 μονάδες. Ενώ για την ομάδα "beta" η μικρότερη τιμή είναι 7.628 και η μεγαλύτερη 19.792, δηλαδή το εύρος είναι περίπου 12 μονάδες. Υπολογίσαμε ότι ο μέσος όρος για την ομάδα "alpha" είναι 6.496, ενώ για την

ομάδα "beta" είναι 12.122.



Σχήμα 5.3: Τιμές των αποκλίσεων CRMS για Lattice 1-2-1

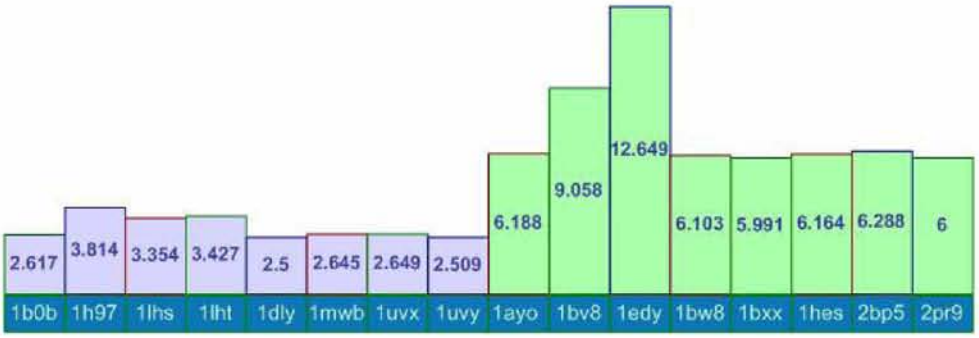
Όπως βλέπουμε στο Σχήμα 5.3 η μικρότερη τιμή για την ομάδα "alpha" είναι 4.266 και η μεγαλύτερη 8.865, δηλαδή το εύρος είναι περίπου 4 μονάδες. Ενώ για την ομάδα "beta" η μικρότερη τιμή είναι 9.058 και η μεγαλύτερη 18.420, δηλαδή το εύρος είναι περίπου 9 μονάδες. Υπολογίσαμε ότι ο μέσος όρος για την ομάδα "alpha" είναι 6.136, ενώ για την ομάδα "beta" είναι 11.529.



Σχήμα 5.4: Τιμές των αποκλίσεων CRMS για Lattice 1-1-2

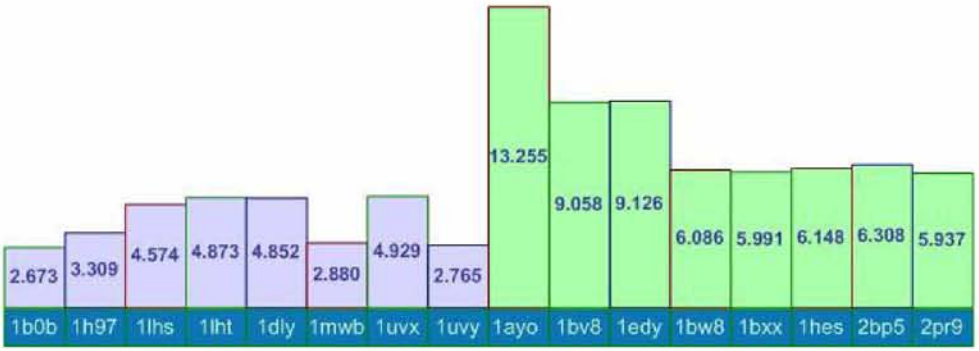
Όπως βλέπουμε στο Σχήμα 5.4 η μικρότερη τιμή για την ομάδα "alpha" είναι 3.993 και η μεγαλύτερη 8.868, δηλαδή το εύρος είναι περίπου 5 μονάδες. Ενώ για την ομάδα "beta" η μικρότερη τιμή είναι 7.877 και η μεγαλύτερη 19.339, δηλαδή το εύρος είναι περίπου 12

μονάδες. Υπολογίσαμε ότι ο μέσος όρος για την ομάδα "alpha" είναι 6.015, ενώ για την ομάδα "beta" είναι 11.052.



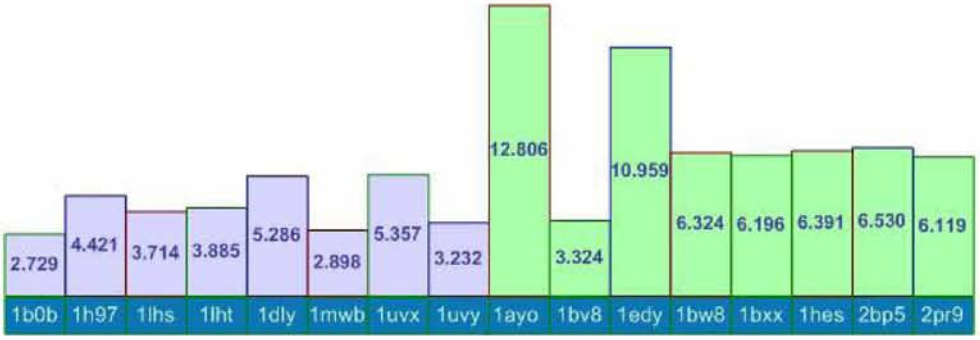
Σχήμα 5.5: Τιμές των αποκλίσεων CRMS για Lattice 2-2-2

Όπως βλέπουμε στο Σχήμα 5.5 η μικρότερη τιμή για την ομάδα "alpha" είναι 2.5 και η μεγαλύτερη 3.814, δηλαδή το εύρος είναι περίπου 1 μονάδα. Ενώ για την ομάδα "beta" η μικρότερη τιμή είναι 5.991 και η μεγαλύτερη 12.649, δηλαδή το εύρος είναι περίπου 7 μονάδες. Υπολογίσαμε ότι ο μέσος όρος για την ομάδα "alpha" είναι 2.939, ενώ για την ομάδα "beta" είναι 7.305.



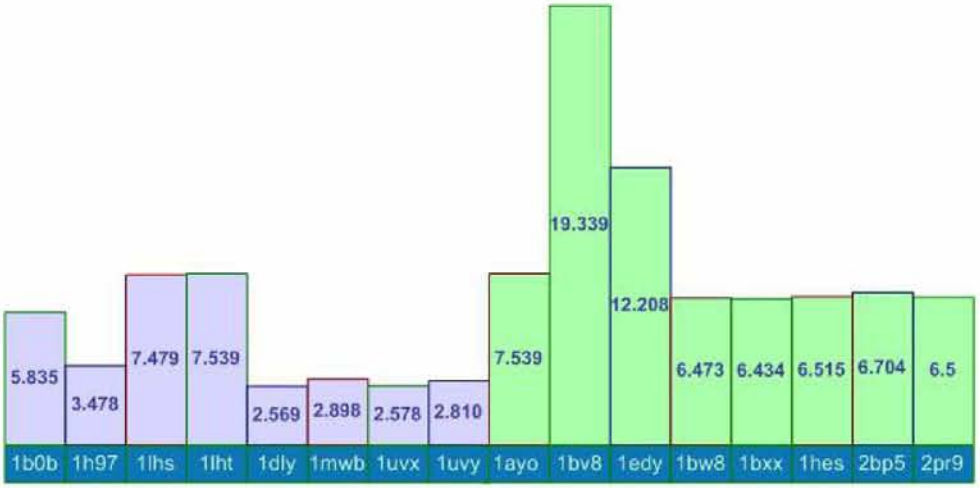
Σχήμα 5.6: Τιμές των αποκλίσεων CRMS για Lattice 1-2-3

Όπως βλέπουμε στο Σχήμα 5.6 η μικρότερη τιμή για την ομάδα "alpha" είναι 2.673 και η μεγαλύτερη 4.929, δηλαδή το εύρος είναι περίπου 2 μονάδες. Ενώ για την ομάδα "beta" η μικρότερη τιμή είναι 5.937 και η μεγαλύτερη 13.255, δηλαδή το εύρος είναι περίπου 8 μονάδες. Υπολογίσαμε ότι ο μέσος όρος για την ομάδα "alpha" είναι 3.869, ενώ για την ομάδα "beta" είναι 7.738.



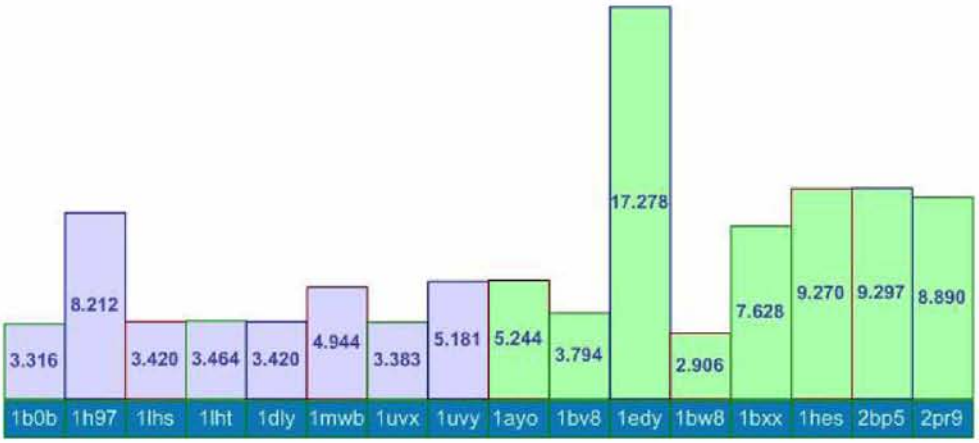
Σχήμα 5.7: Τιμές των αποκλίσεων CRMS για Lattice 1-3-2

Όπως βλέπουμε στο Σχήμα 5.7 η μικρότερη τιμή για την ομάδα "alpha" είναι 2.729 και η μεγαλύτερη 5.357, δηλαδή το εύρος είναι περίπου 3 μονάδες. Ενώ για την ομάδα "beta" η μικρότερη τιμή είναι 3.324 και η μεγαλύτερη 12.806, δηλαδή το εύρος είναι περίπου 9 μονάδες. Υπολογίσαμε ότι ο μέσος όρος για την ομάδα "alpha" είναι 3.940, ενώ για την ομάδα "beta" είναι 7.331.



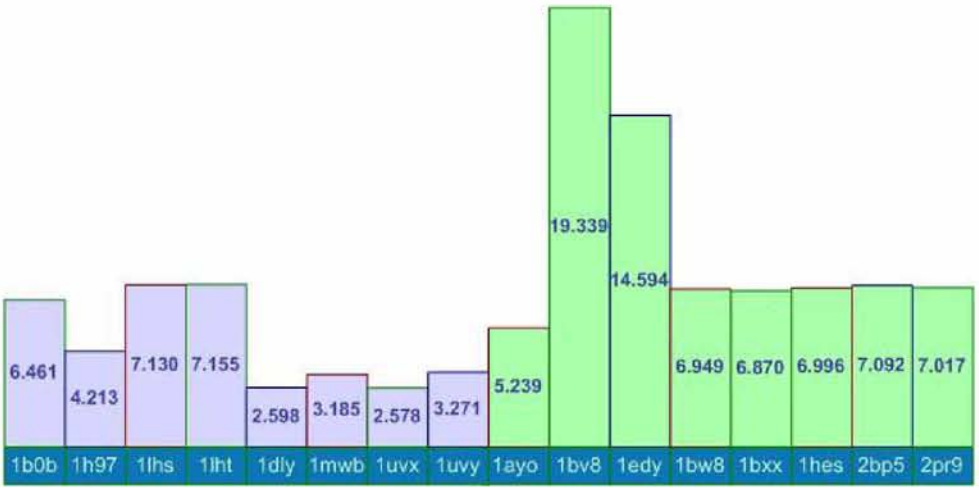
Σχήμα 5.8: Τιμές των αποκλίσεων CRMS για Lattice 2-1-3

Όπως βλέπουμε στο Σχήμα 5.8 η μικρότερη τιμή για την ομάδα "alpha" είναι 2.569 και η μεγαλύτερη 7.539, δηλαδή το εύρος είναι περίπου 5 μονάδες. Ενώ για την ομάδα "beta" η μικρότερη τιμή είναι 6.434 και η μεγαλύτερη 19.339, δηλαδή το εύρος είναι περίπου 13 μονάδες. Υπολογίσαμε ότι ο μέσος όρος για την ομάδα "alpha" είναι 4.398, ενώ για την ομάδα "beta" είναι 8.151.



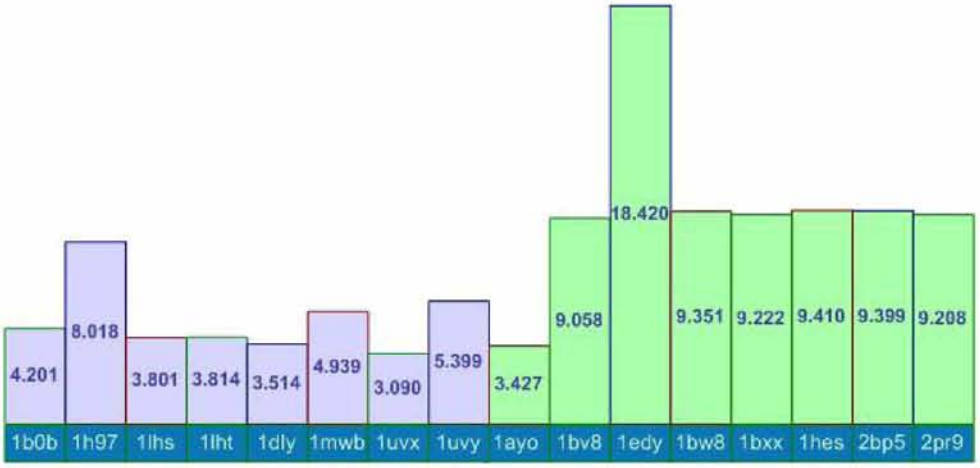
Σχήμα 5.9: Τιμές των αποκλίσεων CRMS για Lattice 2-3-1

Όπως βλέπουμε στο Σχήμα 5.9 η μικρότερη τιμή για την ομάδα "alpha" είναι 3.316 και η μεγαλύτερη 8.212, δηλαδή το εύρος είναι περίπου 5 μονάδες. Ενώ για την ομάδα "beta" η μικρότερη τιμή είναι 2.906 και η μεγαλύτερη 17.278, δηλαδή το εύρος είναι περίπου 15 μονάδες. Υπολογίσαμε ότι ο μέσος όρος για την ομάδα "alpha" είναι 4.417, ενώ για την ομάδα "beta" είναι 8.038.



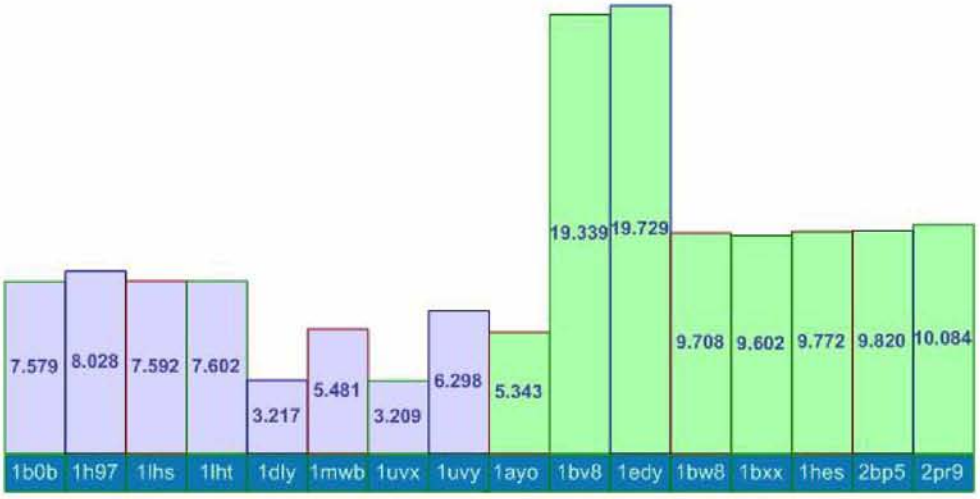
Σχήμα 5.10: Τιμές των αποκλίσεων CRMS για Lattice 3-1-2

Όπως βλέπουμε στο Σχήμα 5.10 η μικρότερη τιμή για την ομάδα "alpha" είναι 2.578 και η μεγαλύτερη 7.155, δηλαδή το εύρος είναι περίπου 5 μονάδες. Ενώ για την ομάδα "beta" η μικρότερη τιμή είναι 5.239 και η μεγαλύτερη 19.339, δηλαδή το εύρος είναι περίπου 14 μονάδες. Υπολογίσαμε ότι ο μέσος όρος για την ομάδα "alpha" είναι 4.573, ενώ για την ομάδα "beta" είναι 9.262.



Σχήμα 5.11: Τιμές των αποκλίσεων CRMS για Lattice 3-2-1

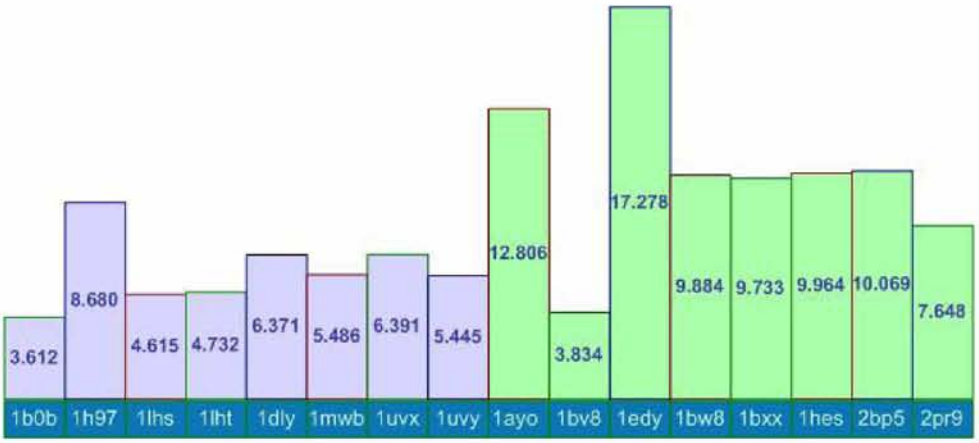
Όπως βλέπουμε στο Σχήμα 5.11 η μικρότερη τιμή για την ομάδα "alpha" είναι 3.090 και η μεγαλύτερη 8.018, δηλαδή το εύρος είναι περίπου 5 μονάδες. Ενώ για την ομάδα "beta" η μικρότερη τιμή είναι 3.427 και η μεγαλύτερη 18.420, δηλαδή το εύρος είναι περίπου 15 μονάδες. Υπολογίσαμε ότι ο μέσος όρος για την ομάδα "alpha" είναι 4.552, ενώ για την ομάδα "beta" είναι 9.686.



Σχήμα 5.12: Τιμές των αποκλίσεων CRMS για Lattice 3-1-1

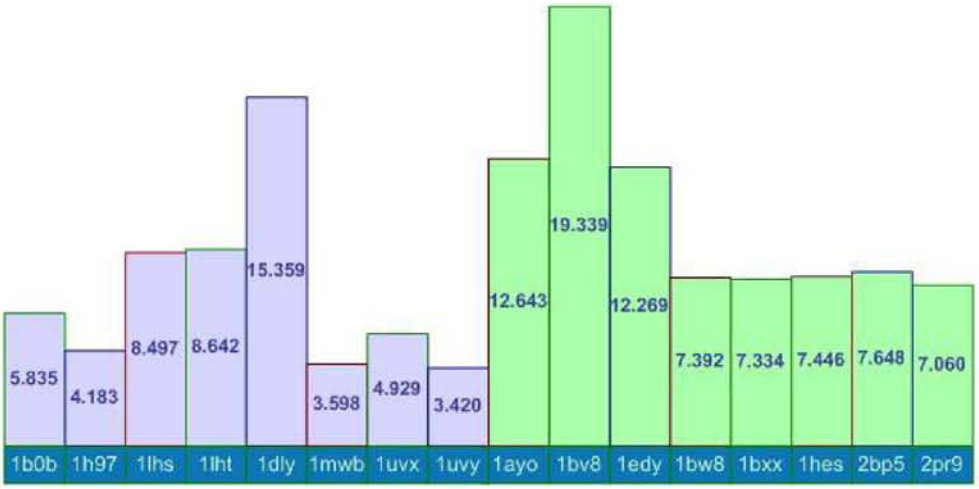
Όπως βλέπουμε στο Σχήμα 5.12 η μικρότερη τιμή για την ομάδα "alpha" είναι 3.209 και η μεγαλύτερη 8.028, δηλαδή το εύρος είναι περίπου 5 μονάδες. Ενώ για την ομάδα "beta" η μικρότερη τιμή είναι 5.343 και η μεγαλύτερη 19.729, δηλαδή το εύρος είναι περίπου 14 μονάδες. Υπολογίσαμε ότι ο μέσος όρος για την ομάδα "alpha" είναι 6.125, ενώ για την

ομάδα "beta" είναι 11.674.



Σχήμα 5.13: Τιμές των αποκλίσεων CRMS για Lattice 1-3-1

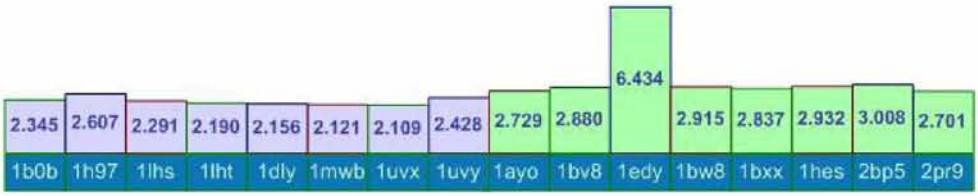
Όπως βλέπουμε στο Σχήμα 5.13 η μικρότερη τιμή για την ομάδα "alpha" είναι 3.612 και η μεγαλύτερη 8.680, δηλαδή το εύρος είναι περίπου 5 μονάδες. Ενώ για την ομάδα "beta" η μικρότερη τιμή είναι 3.834 και η μεγαλύτερη 17.278, δηλαδή το εύρος είναι περίπου 14 μονάδες. Υπολογίσαμε ότι ο μέσος όρος για την ομάδα "alpha" είναι 5.666, ενώ για την ομάδα "beta" είναι 10.151.



Σχήμα 5.14: Τιμές των αποκλίσεων CRMS για Lattice 1-1-3

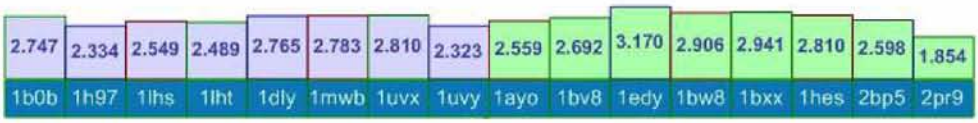
Όπως βλέπουμε στο Σχήμα 5.14 η μικρότερη τιμή για την ομάδα "alpha" είναι 3.420 και η μεγαλύτερη 15.359, δηλαδή το εύρος είναι περίπου 12 μονάδες. Ενώ για την ομάδα "beta" η μικρότερη τιμή είναι 7.060 και η μεγαλύτερη 19.339, δηλαδή το εύρος είναι περίπου

12 μονάδες. Υπολογίσαμε ότι ο μέσος όρος για την ομάδα "alpha" είναι 6.807, ενώ για την ομάδα "beta" είναι 10.266.



Σχήμα 5.15: Τιμές των αποκλίσεων CRMS για Lattice 3-3-3

Όπως βλέπουμε στο Σχήμα 5.15 η μικρότερη τιμή για την ομάδα "alpha" είναι 2.109 και η μεγαλύτερη 2.607, δηλαδή το εύρος είναι λιγότερο από 1 μονάδα. Ενώ για την ομάδα "beta" η μικρότερη τιμή είναι 2.701 και η μεγαλύτερη 6.434, δηλαδή το εύρος είναι περίπου 4 μονάδες. Υπολογίσαμε ότι ο μέσος όρος για την ομάδα "alpha" είναι 2.280, ενώ για την ομάδα "beta" είναι 3.429.



Σχήμα 5.16: Τιμές των αποκλίσεων CRMS για Lattice 3.8-3.8-3.8

Όπως βλέπουμε στο Σχήμα 5.16 η μικρότερη τιμή για την ομάδα "alpha" είναι 2.323 και η μεγαλύτερη 2.810, δηλαδή το εύρος είναι λιγότερο από 1 μονάδα. Ενώ για την ομάδα "beta" η μικρότερη τιμή είναι 1.854 και η μεγαλύτερη 3.170, δηλαδή το εύρος είναι περίπου 2 μονάδες. Υπολογίσαμε ότι ο μέσος όρος για την ομάδα "alpha" είναι 2.6, ενώ για την ομάδα "beta" είναι 2.816.

Ο Πίνακας 5.2 περιγράφει με αύξουσα σειρά ποιος είναι ο μέσος όρος των αντικειμενικών συναρτήσεων του προβλήματος που έδωσε ο αλγόριθμος του CPLEX και για τις δύο ομάδες των πρωτεϊνών μαζί. Είναι προφανές ότι το lattice που δίνει τη μικρότερη CRMS τιμή και άρα τη βέλτιστη απεικόνιση των πρωτεϊνών είναι το $lattice(3, 8-3, 8-3, 8)$

Οι Πίνακες 5.3 και 5.4 περιγράφουν με αύξουσα σειρά ποιος είναι ο μέσος όρος των αντικειμενικών συναρτήσεων του προβλήματος που έδωσε ο αλγόριθμος του CPLEX για κάθε μία ομάδα αντίστοιχα. Είναι προφανές ότι το lattice που δίνει τη μικρότερη απόκλιση CRMS και άρα τη βέλτιστη απεικόνιση των πρωτεϊνών για την ομάδα "alpha" είναι το $lattice(3-3-3)$, ενώ για την ομάδα "beta" είναι το $lattice(3, 8-3, 8-3, 8)$.

Lattices	Μέσος Όρος
$Lattice(3, 8-3, 8-3, 8)$	2.708
$Lattice(3-3-3)$	2.828
$Lattice(2-2-2)$	5.122
$Lattice(1-3-2)$	5.635
$Lattice(1-2-3)$	5.803
$Lattice(2-3-1)$	6.227
$Lattice(2-1-3)$	6.274
$Lattice(3-1-2)$	6.917
$Lattice(3-2-1)$	7.119
$Lattice(1-3-1)$	7.908
$Lattice(1-1-2)$	8.533
$Lattice(1-1-3)$	8.536
$Lattice(1-2-1)$	8.832
$Lattice(3-1-1)$	8.899
$Lattice(2-1-1)$	9.309
$Lattice(1-1-1)$	10.443

Πίνακας 5.2: Μέσος Όρος των αντικειμενικών συναρτήσεων και για τις δύο ομάδες πρωτεΐνων

Το εύρος στους Πίνακες 5.3 και 5.4 δείχνει πόσο καλά αντιπροσωπεύει η τιμή του μέσου όρου το δείγμα των πρωτεϊνών, για κάθε ένα lattice. Όσο πιο μικρή είναι η τιμή αυτή, τόσο περισσότερες πρωτεΐνες δίνουν απόκλιση CRMS κοντά στο μέσο όρο. Αντίθετα, όταν το εύρος είναι μεγάλο τότε ο μέσος όρος δεν αντιπροσωπεύει με επιτυχία το δείγμα, όπως συμβαίνει για παράδειγμα στο lattice(3-1-2), όπου ο μέσος όρος για την ομάδα "beta" είναι 9.262 και το εύρος του δείγματος είναι 14 μονάδες. Πραγματικά, από το Σχήμα 5.10 είναι ξεκάθαρο ότι 5 πρωτεΐνες από τις 8 πρωτεΐνες έχουν απόκλιση CRMS πολύ κοντά στην τιμή 7. Το γεγονός όμως ότι δύο πρωτεΐνες δίνουν πολύ μεγάλες τιμές στην απόκλιση CRMS ανεβάζει κατά πολύ το μέσο όρο του δείγματος.

Lattices	Μέσος Όρος	Εύρος>0
$Lattice(3-3-3)$	2.280	<1
$Lattice(3,8-3,8-3,8)$	2.6	<1
$Lattice(2-2-2)$	2.939	1
$Lattice(1-2-3)$	3.869	2
$Lattice(1-3-2)$	3.940	3
$Lattice(2-1-3)$	4.398	5
$Lattice(2-3-1)$	4.417	5
$Lattice(3-2-1)$	4.552	5
$Lattice(3-1-2)$	4.573	5
$Lattice(1-3-1)$	5.666	5
$Lattice(1-1-2)$	6.015	1
$Lattice(3-1-1)$	6.125	5
$Lattice(1-2-1)$	6.136	4
$Lattice(2-1-1)$	6.496	5
$Lattice(1-1-3)$	6.807	12
$Lattice(1-1-1)$	7.554	3

Πίνακας 5.3: Μέσος Όρος των αντικειμενικών συναρτήσεων της ομάδας "alpha"

Lattices	Μέσος Όρος	Εύρος>0
$Lattice(3, 8-3, 8-3, 8)$	2.816	2
$Lattice(3-3-3)$	3.429	4
$Lattice(2-2-2)$	7.305	7
$Lattice(1-3-2)$	7.331	9
$Lattice(1-2-3)$	7.738	8
$Lattice(2-3-1)$	8.038	15
$Lattice(2-1-3)$	8.151	13
$Lattice(3-1-2)$	9.262	14
$Lattice(3-2-1)$	9.686	15
$Lattice(1-3-1)$	10.151	14
$Lattice(1-1-3)$	10.266	13
$Lattice(1-1-2)$	11.052	7
$Lattice(1-2-1)$	11.529	9
$Lattice(3-1-1)$	11.674	14
$Lattice(2-1-1)$	12.122	12
$Lattice(1-1-1)$	13.332	9

Πίνακας 5.4: Μέσος Όρος των αντικειμενικών συναρτήσεων της ομάδας "beta"

Κεφάλαιο 6

ΣΥΜΠΕΡΑΣΜΑΤΑ-ΑΝΟΙΧΤΑ ΠΡΟΒΛΗΜΑΤΑ

Ο στόχος αυτής της πτυχιακής εργασίας ήταν:

1. Η σχεδίαση και υλοποίηση ενός ολοκληρωμένου λογισμικού το οποίο παίρνει σαν είσοδο την τρισδιάστατη δομή μίας πρωτεΐνης και ένα τρισδιάστατο lattice και επιστρέφει μία βέλτιστη αναδίπλωση της πρωτεΐνης πάνω στο lattice.

2. Η εφαρμογή αυτού του νέου λογισμικού σε μία σειρά πρωτεϊνών και lattices. Η επιλογή των πρωτεϊνών δεν έγινε τυχαία αλλά από συγκεκριμένες ομάδες.

Ουσιαστικά αυτό που επιχειρήσαμε ήταν να δώσουμε απάντηση στο ερώτημα κατά πόσο πρωτεΐνες που κατατάσσονται στην ίδια ομάδα λόγω κοινών (π.χ. μακροσκοπικών) χαρακτηριστικών τους, δίνουν βέλτιστες αναδιπλώσεις σε όχι πολύ διαφορετικά lattices.

Τα πρώτα αποτελέσματα της έρευνας αυτής δείχνουν ότι: Τμήματα από πρωτεΐνες που ανήκουν σε δύο διαφορετικές ομάδες, την ομάδα "alpha" και την ομάδα "beta", δεν αναδιπλώνονται το ίδιο καλά στο ίδιο lattice. Πιο συγκεκριμένα, για την ομάδα "alpha" βρίκαμε ότι το lattice που δίνει την καλύτερη απεικόνιση είναι το lattice(3-3-3), ενώ για τη ομάδα "beta" είναι το lattice(3.8-3.8-3.8). Για τα τμήματα των πρωτεϊνών που ανήκουν στην ομάδα "alpha", 6 από αυτές συγκλίνουν στο lattice(3-3-3) και 2 στο lattice(3.8,3.8,3.8), ενώ για τα τμήματα των πρωτεϊνών που ανήκουν στην ομάδα "beta", 7 από αυτές συγκλίνουν στο lattice(3.8,3.8,3.8) και μόνο 1 στο lattice(3-3-3). Έτσι λοιπόν φαίνεται από τις πρώτες ενδείξεις ότι πρωτεϊνες που ανήκουν στην ίδια ομάδα τείνουν να δίνουν βέλτιστες αναδιπλώσεις στο ίδιο lattice. Βέβαια τονίζουμε εδώ ότι τα πειράματα έγιναν σε μικρά μέρη των πρωτεϊνικών αλυσίδων (20 πρώτα αμινοξέα) και έτσι η πλήρης εικόνα (για ολόκληρες τις πρωτεϊνικές αλυσίδες) μπορεί να είναι αρκετά διαφορετική.

Επεκτείνοντας την έρευνα αυτή, θα μπορούσαμε να μελετήσουμε τα παρακάτω "ανοιχτά" προβλήματα:

1. Την υλοποίηση ενός προγράμματος που οπτικοποιεί την αναδίπλωση μίας πρωτεΐνης σε ένα lattice. Κάτι τέτοιο θα μπορούσε να βοηθήσει στο ερώτημα πόσο κοντά "οπτικά" είναι η αναδίπλωση της πρωτεΐνης στο lattice με την πραγματική αναδίπλωση. Με άλλα λόγια, η ελαχιστοποιήση του CRMS μεταξύ δύο αναδιπλώσεων με πόση ακρίβεια μετρά τις διαφορές των δύο αναδιπλώσεων;

2. Ένα δεύτερο πρόβλημα αποτελεί η μελέτη της ακρίβειας διαφορετικών κριτηρίων από το CRMS, για παράδειγμα του DRMS. Επίσης θα μπορούσαν να συγκριθούν τα αποτελέσματα που δίνουν οι δύο προσεγγίσεις (CRMS,DRMS).

3. Εξαιρετικό ενδιαφέρον επίσης θα είχε η αλλαγή των παραμέτρων του προβλήματος. Για παράδειγμα, η ταυτόχρονη εύρεση της θέσης στο χώρο ενός δεδομένου lattice και της βέλτιστης αναδίπλωσης της πρωτεΐνης σε αυτό.

4. Τέλος, η μελέτη των αποτελεσμάτων σε δισδιάστατα lattices θα είχε ιδιαίτερη σημασία καθώς το PCLF δεν έχει αποδειχτεί αν είναι NP-complete για δισδιάστατα lattices. Συνεπώς, σε αυτή την περίπτωση υπάρχει μεγάλο ενδιαφέρον αφενός για την απόδειξη της NP-πληρότητας και αφετέρου για τη σχεδίαση πολυωνυμικού χρόνου αλγόριθμου. Η πρακτική εφαρμογή μίας μεθόδου για δισδιάστατα lattices θα μπορούσε να ήταν πάνω στην καλή οπτικοποίηση της τρισδιάστατης δομής μίας πρωτεΐνης στο επίπεδο (π.χ. σε μία οθόνη).

Για να γυρίσουμε στο διάσημο πρόβλημα αναδίπλωσης μίας πρωτεΐνης σε ένα lattice με δεδομένη μόνο την αλυσίδα των αμινοξέων της πρωτεΐνης και όχι την τρισδιάστατη δομή της (Protein Folding Problem) θα ήταν πολύ ενδιαφέρον να ερευνήσει κανείς τα εξής: Έστω L ένα lattice στο οποίο αναδιπλώνεται καλύτερα μία πρωτεΐνη P σύμφωνα με τη μέθοδο που αναπτύξαμε σε αυτή την εργασία. Αν τώρα πάρουμε μία αλυσίδα αμινοξέων και ψάξουμε να βρούμε μία αναδίπλωσή της πάνω σε αυτό το lattice (π.χ. HP-Model), πόσο κοντά θα είναι η αναδίπλωση αυτή σε σχέση με την φυσική αναδίπλωση της πρωτεΐνης στο χώρο; Το αποτέλεσμα της έρευνας αυτής θα αξιολογήσει ουσιαστικά την ακρίβεια της δικής μας μεθόδου.

Τέλος, για την επίλυση του PCLF προβλήματος οι μόνοι αλγόριθμοι που έχουν χρησιμοποιηθεί είναι εκθετικής χρονικής πολυπλοκότητας. Μάλιστα, όπως ήδη αναφέραμε από την εισαγωγή για ειδικές κατηγορίες τρισδιάστατων lattices έχει ήδη αποδειχτεί πως το πρόβλημα είναι NP-complete. Έτσι λοιπόν οι ελπίδες να υπάρχει αλγόριθμος πολυωνυμικού

χρόνου για τρισδιάστατα lattices είναι ελάχιστες. Συνεπώς, αποκτά μεγάλη σπουδαίοτητα η αναπαράσταση του προβλήματος (ορισμός του προβλήματος).

Για παράδειγμα, στη μέθοδο που αναπτύξαμε, για μία πρωτεΐνη που αποτελείται από n αμινοξέα και ένα lattice με μήκη ακμών L_x, L_y, L_z κατασκευάζουμε ένα πρόγραμμα ακέραιου Γραμμικού Προγραμματισμού με πολυωνυμικό πλήθος ($f(n) = O(n^4)$) μεταβλητών και περιορισμών. Έτσι λοιπόν ο εκθετικός αλγόριθμος που καλείται να λύσει το μετασχηματισμένο πρόβλημα έχει χρονική πολυπλοκότητα τάξης $O(2^{f(n)})$. Συνεπώς, επειδή το $f(n)$ βρίσκεται στον εκθέτη έχει μεγάλη σημασία η οποιαδήποτε μείωσή του καθώς με αυτόν τον τρόπο θα είναι πραγματοποιήσιμη η ανάλυση μεγαλύτερων αλυσίδων. Ενδεικτικά μάλιστα αναφέρουμε ότι η μέθοδος που αναπτύξαμε για το μετασχηματισμένο PCLF πρόβλημα σε πρόβλημα ακέραιου Γραμμικού Προγραμματισμού έχει πολύ μεγάλες απαιτήσεις σε μνήμη. Αποτελεί λοιπόν πρόκληση η αλλαγή του προγράμματος ή/και της τεχνικής για την παραγωγή ενός ισοδύναμου προβλήματος ακέραιου Γραμμικού Προγραμματισμού με πολύ λιγότερες μεταβλητές και λιγότερους περιορισμούς.

Με την ίδια λογική αποτελεί επίσης πρόκληση η αναζήτηση άλλων τεχνικών με τις οποίες θα μπορούσαμε πιο γρήγορα να πάρουμε αποτελέσματα. Για παράδειγμα έχει προταθεί από πολλούς ερευνητές το παρακάτω μοντέλο: Η πρωτεΐνη δεν αναδιπλώνεται παίρνοντας ως δεδομένο το σύνολο της αλυσίδας αλλά "χωρίζει" την αλυσίδα σε τμήματα και αναδιπλώνει κάθε ένα από αυτά ξεχωριστά. Στη συνέχεια τα τμήματα αυτά αλληλεπιδρούν μεταξύ τους με σκοπό να δώσουν το τελικό σχήμα της πρωτεΐνης. Αν το μοντέλο αυτό αναπαριστά καλά αυτό που γίνεται στη φύση, τότε οι αλγόριθμοι που έχουμε στη διάθεσή μας θα μπορούσαν να δώσουν λύσεις για πολύ μεγαλύτερα μήκη πρωτεϊνικών αλυσίδων. Για τους παραπάνω λόγους, η αξιολόγηση και η οριστικοποιήση των λεπτομερειών ενός τέτοιου μοντέλου είναι πολύ ενδιαφέρον πρόβλημα.

Παράρτημα Α΄

Πλήρες Πρόγραμμα

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#define SIZE 200
#define Lx 3.8
#define Ly 3.8
#define Lz 3.8
#define MAX 5
#define L 1
/*********************************************************************/
typedef struct node
       {
           int x;
           int y;
           int z;
           struct node *next;
       }NODE;

/*********************************************************************/
typedef struct array_node
       {
            int info;
            struct node *Head;
       }ARRAY_NODE;
/*********************************************************************/
typedef struct lat_node
```

```
      {
           int var;
           double x;
           double y;
           double z;
      }LAT_NODE;



/*****************************************************************/
void initial  (ARRAY_NODE *B);
/*****************************************************************/
int find_new_node(NODE *read, int C[][3], ARRAY_NODE *B);
/*****************************************************************/
void ln_create(ARRAY_NODE *B, int C[][3], int p_p);
/*****************************************************************/
int read_pdb_file(char initial_file[], char initial_elements[], int max);
/*****************************************************************/
int protein_points(char protein[], double protein_coordinates[][3], int max);
/*****************************************************************************/
/*****************************************************************************/
/*****************************************************************************/
/*****************************************************************************/
/*****************************************************************************/
/*****************************************************************************/
int main()
```

```
{
    double protein_coordinates[MAX][3];
    char initial_elements[SIZE];
    int pdb_elements;
    int p_p;
    int fnn;
    int i,j,k;
    int x,y,z;
    int line=0;
    int C[6][3];
    ARRAY_NODE B[MAX];
    LAT_NODE LAT_ARR[(2*MAX)-1][(2*MAX)-1][(2*MAX)-1];
    int middle;
    int l_l;
    double distance;
    int foundx=0;
    int foundy=0;
    int foundz=0;
    int new_foundx=0;
    int new_foundy=0;
    int new_foundz=0;

    FILE *cplex_file;
    NODE *read;
    NODE *minimize;
    NODE *binary;
```

```
    NODE *constraint1;
    NODE *constraint2;
    NODE *constraint3;


/*********************CALL read_pdb_file*********************************/
    pdb_elements = read_pdb_file("initial_file.txt", initial_elements, SIZE);
/*********************CALL protein_points********************************/
    p_p = protein_points("protein.txt", protein_coordinates, MAX);


    middle=p_p-1;

    l_l=(2*p_p)-1;

/************************INITIALING ARRAY C*************************/
/*******************************************************************/
      for (i=0; i<6; i++)
      {
         for (j=0; j<3; j++)
         {
             C[i][j]=0;
         }
      }
/*******************************************************************/
```

```
/*************************CALL initial*************************************/
/**************************************************************************/
      for (i=0; i<p_p; i++)
        {
          initial (&B[i]);
        }
/**************************************************************************/


/*************************CALL find_new_node AND ln_create*****************/
/**************************************************************************/
      for (i=0; i<p_p; i++)
        {
          if (i==0)
          {
                ln_create(&B[i],C,p_p);
          }
          else
          {
              if(i-1==0)
              {
                  read=B[i-1].Head;
```

```
                        find_new_node(read,C,&B[i-1]);
                        ln_create(&B[i],C,p_p);
                    }
                    else
                    {
                        read=B[i-1].Head;
                        while(read!=NULL)
                        {
                            fnn=0;
                            fnn=find_new_node(read,C,&B[i-1]);
                            if (fnn==1)
                            {
                                ln_create(&B[i],C,p_p);
                                printf("\n");
                                read=read->next;
                            }
                        }
                    }
                }
            }
/*********************************************************************/
/*********************************************************************/
/****************  INITIALIZING ARRAY LAT_ARR ********************/
    for (i=0; i<l_l; i++)
    {
       for (j=0; j<l_l; j++)
```

```
        {
            for (k=0; k<l_l; k++)
            {
                LAT_ARR[i][j][k].x==0.0;
                LAT_ARR[i][j][k].y==0.0;
                LAT_ARR[i][j][k].z==0.0;
                LAT_ARR[i][j][k].var=0;
            }
        }
    }
/*********************************************************************/
/*********************************************************************/
/**** SET THE CENTRAL POINT OF LATTICE ON THE FIRST OMINO ACID *****/

                    LAT_ARR[middle][middle][middle].x = protein_coordinates[0][0];
                    LAT_ARR[middle][middle][middle].y = protein_coordinates[0][1];
                    LAT_ARR[middle][middle][middle].z = protein_coordinates[0][2];
/*********************************************************************/


/*********************************************************************/
/***********************OBJECTIVE FUNTION*****************************/
cplex_file = fopen("input_cplex.lp", "w");
fprintf(cplex_file,"Minimize\n");
fprintf(cplex_file,"Obj:\n");
   for(i=0; i<p_p; i++)
```

```
   {
      if (i!=p_p-1)
      {
          minimize=B[i].Head;
          while(minimize!=NULL)
          {
                    j=0;
                    distance=0;
                    foundx=0;
                    foundy=0;
                    foundz=0;
                    foundx=minimize->x - middle;
                    foundy=minimize->y - middle;
                    foundz=minimize->z - middle;
                    LAT_ARR[minimize->x][minimize->y][minimize->z].x =
LAT_ARR[middle][middle][middle].x + (foundx*Lx);
                    LAT_ARR[minimize->x][minimize->y][minimize->z].y =
LAT_ARR[middle][middle][middle].y + (foundy*Ly);
                    LAT_ARR[minimize->x][minimize->y][minimize->z].z =
LAT_ARR[middle][middle][middle].z + (foundz*Lz);
                    while(j<=2)
                    {
                              if (j==0)
                              {
distance=distance+(protein_coordinates[i][j]-
LAT_ARR[minimize->x][minimize->y][minimize->z].x)*
```

```
(protein_coordinates[i][j]-
LAT_ARR[minimize->x][minimize->y][minimize->z].x));
                        j++;
                    }
                    if (j==1)
                    {
distance=distance+((protein_coordinates[i][j]-
        LAT_ARR[minimize->x][minimize->y][minimize->z].y)*
(protein_coordinates[i][j]-
LAT_ARR[minimize->x][minimize->y][minimize->z].y));
                        j++;
                    }
                    if (j==2)
                    {
distance=distance+((protein_coordinates[i][j]-
LAT_ARR[minimize->x][minimize->y][minimize->z].z)*
(protein_coordinates[i][j]-
LAT_ARR[minimize->x][minimize->y][minimize->z].z));
                        j++;
                    }
                }
                fprintf(cplex_file,"%fX(%d,(%d,%d,%d)) + ",distance,i+1,minimize->x,minimize->y,minimize->z);
                minimize=minimize->next;
            }
        }
        else
```

```
    {
          minimize=B[i].Head;
          while(minimize!=NULL)
          {
              if (minimize->next!=NULL)
              {
                    j=0;
                    distance=0;
                    foundx=0;
                    foundy=0;
                    foundz=0;
                    foundx=minimize->x-middle;
                    foundy=minimize->y-middle;
                    foundz=minimize->z-middle;
                    LAT_ARR[minimize->x][minimize->y][minimize->z].x = LAT_ARR[middle][middle][middle].x +
 (foundx*Lx);
                    LAT_ARR[minimize->x][minimize->y][minimize->z].y = LAT_ARR[middle][middle][middle].y +
 (foundy*Ly);
                    LAT_ARR[minimize->x][minimize->y][minimize->z].z =LAT_ARR[middle][middle][middle].z +
 (foundz*Lz);
                    while(j<=2)
                    {
                              if (j==0)
                              {
                                       distance=distance+((protein_coordinates[i][j]-
LAT_ARR[minimize->x][minimize->y][minimize->z].x)*
```

```
(protein_coordinates[i][j]-
LAT_ARR[minimize->x][minimize->y][minimize->z].x));
                        j++;
                    }
                    if (j==1)
                    {
distance=distance+((protein_coordinates[i][j]-
LAT_ARR[minimize->x][minimize->y][minimize->z].y)*
(protein_coordinates[i][j]-
LAT_ARR[minimize->x][minimize->y][minimize->z].y));
                        j++;
                    }
                    if (j==2)
                    {
  distance=distance+((protein_coordinates[i][j]-
LAT_ARR[minimize->x][minimize->y][minimize->z].z)*
(protein_coordinates[i][j]-
LAT_ARR[minimize->x][minimize->y][minimize->z].z));
                        j++;
                    }
                }
                fprintf(cplex_file,"%fX(%d,(%d,%d,%d)) + ",distance,i+1,minimize->x,minimize->y,minimize->z);
                minimize=minimize->next;
            }
            else
            {
```

```
                    j=0;
                    distance=0;
                    foundx=0;
                    foundy=0;
                    foundz=0;
                    foundx=minimize->x-middle;
                    foundy=minimize->y-middle;
                    foundz=minimize->z-middle;
                    LAT_ARR[minimize->x][minimize->y][minimize->z].x = LAT_ARR[middle][middle][middle].x +
(foundx*Lx);
                    LAT_ARR[minimize->x][minimize->y][minimize->z].y = LAT_ARR[middle][middle][middle].y +
(foundy*Ly);
                    LAT_ARR[minimize->x][minimize->y][minimize->z].z = LAT_ARR[middle][middle][middle].z +
(foundz*Lz);
                    while(j<=2)
                    {
                              if (j==0)
                              {
distance=distance+((protein_coordinates[i][j]-
LAT_ARR[minimize->x][minimize->y][minimize->z].x)*
(protein_coordinates[i][j]-
LAT_ARR[minimize->x][minimize->y][minimize->z].x));
                                       j++;
                              }
                              if (j==1)
                              {
```

```
distance=distance+((protein_coordinates[i][j]-
LAT_ARR[minimize->x][minimize->y][minimize->z].y)*
(protein_coordinates[i][j]-
LAT_ARR[minimize->x][minimize->y][minimize->z].y));
                        j++;
                    }
                    if (j==2)
                    {
distance=distance+((protein_coordinates[i][j]-
LAT_ARR[minimize->x][minimize->y][minimize->z].z)*
(protein_coordinates[i][j]-
LAT_ARR[minimize->x][minimize->y][minimize->z].z));
                        j++;
                    }
                }
                fprintf(cplex_file,"%fX(%d,(%d,%d,%d))",distance,i+1,minimize->x,minimize->y,minimize->z);
                minimize=minimize->next;
            }
        }
    }
fprintf(cplex_file,"\n");
fprintf(cplex_file,"\n");
/*************************************************************************/

/*************************************************************************/
```

```
/********** This is the First Constraint***************/
 fprintf(cplex_file,"Subject To \n");
 fprintf(cplex_file,"c1: \n");
 for(j=1; j<=p_p; j++)
 {
     constraint1=B[j-1].Head;
     while(constraint1!=NULL)
     {
             foundx=0;
             foundy=0;
             foundz=0;
             foundx=constraint1->x-middle;
             foundy=constraint1->y-middle;
             foundz=constraint1->z-middle;
             LAT_ARR[constraint1->x][constraint1->y][constraint1->z].x = LAT_ARR[middle][middle][middle].x +
(foundx*Lx);
             LAT_ARR[constraint1->x][constraint1->y][constraint1->z].y = LAT_ARR[middle][middle][middle].y +
(foundy*Ly);
             LAT_ARR[constraint1->x][constraint1->y][constraint1->z].z = LAT_ARR[middle][middle][middle].z +
(foundz*Lz);
             fprintf(cplex_file,"X(%d,(%d,%d,%d)) ",j,constraint1->x,constraint1->y,constraint1->z);
             constraint1=constraint1->next;
             if(constraint1!=NULL)
             {
                fprintf(cplex_file,"+  ");
             }
```

```
            else
            {
                fprintf(cplex_file," = 1\n");
            }
    }
}
/*************************BINARY VARIABLES*********************************/
/**************************************************************************/
   fprintf(cplex_file,"Binaries\n");
   for (i=0; i<p_p; i++)
   {

       binary=B[i].Head;
       while(binary!=NULL)
       {
             foundx=0;
             foundy=0;
             foundz=0;
             foundx=binary->x-middle;
             foundy=binary->y-middle;
             foundz=binary->z-middle;
             LAT_ARR[binary->x][binary->y][binary->z].x = LAT_ARR[middle][middle][middle].x +
(foundx*Lx);
             LAT_ARR[binary->x][binary->y][binary->z].y = LAT_ARR[middle][middle][middle].y +
(foundy*Ly);
             LAT_ARR[binary->x][binary->y][binary->z].z = LAT_ARR[middle][middle][middle].z +
```

```
(foundz*Lz);

            fprintf(cplex_file,"X(%d,(%d,%d,%d))\n",i+1,binary->x,binary->y,binary->z);
            binary=binary->next;
      }
      fprintf(cplex_file,"\n");
   }
  fprintf(cplex_file,"\n");
  fprintf(cplex_file,"End\n");
  fclose(cplex_file);
  cplex_file=fopen("input_cplex1.lp","w");
/*********************************************************************/
 /************************************************************/
 /***************This is the second constraint***************/
 fprintf(cplex_file,"c2:\n");
 for(i=0; i<p_p; i++)
 {
      constraint2=B[i].Head;
      while(constraint2!=NULL)
      {
              foundx=0;
              foundy=0;
              foundz=0;

              foundx=constraint2->x-middle;
              foundy=constraint2->y-middle;
```

```
foundz=constraint2->z-middle;

LAT_ARR[constraint2->x][constraint2->y][constraint2->z].x = LAT_ARR[middle][middle][middle].x +
(foundx*Lx);

LAT_ARR[constraint2->x][constraint2->y][constraint2->z].y = LAT_ARR[middle][middle][middle].y +
(foundy*Ly);

LAT_ARR[constraint2->x][constraint2->y][constraint2->z].z = LAT_ARR[middle][middle][middle].z +
(foundz*Lz);

if (LAT_ARR[constraint2->x][constraint2->y][constraint2->z].var==0)
{
    fprintf(cplex_file,"X(%d,(%d,%d,%d))",i+1,constraint2->x,constraint2->y,constraint2->z);
    j=i+2;
    while(j<=p_p-1)
    {
        read=B[j].Head;
        while(read!=NULL)
        {
            new_foundx=0;
            new_foundy=0;
            new_foundz=0;

            new_foundx=read->x-middle;
            new_foundy=read->y-middle;
            new_foundz=read->z-middle;
```

```
                                LAT_ARR[read->x][read->y][read->z].x = LAT_ARR[middle][middle][middle].x +
(new_foundx*Lx);
                                LAT_ARR[read->x][read->y][read->z].y = LAT_ARR[middle][middle][middle].y +
(new_foundy*Ly);
                                LAT_ARR[read->x][read->y][read->z].z = LAT_ARR[middle][middle][middle].z +
(new_foundz*Lz);
                if((LAT_ARR[constraint2->x][constraint2->y][constraint2->z].x==
      LAT_ARR[read->x][read->y][read->z].x)&&
(LAT_ARR[constraint2->x][constraint2->y][constraint2->z].y==
         LAT_ARR[read->x][read->y][read->z].y)&&
(LAT_ARR[constraint2->x]
    [constraint2->y][constraint2->z].z==LAT_ARR[read->x][read->y][read->z].z))
                           {
                                    fprintf(cplex_file," + ");
                            fprintf(cplex_file,"X(%d,(%d,%d,%d))",j+1,constraint2->x,constraint2->y,constraint2->z);
                                    LAT_ARR[constraint2->x][constraint2->y][constraint2->z].var=1;
                                    read=read->next;
                            }
                                  else
                                  {
                                        read=read->next;
                                  }
                            }
                            j++;
                    }
```

```
                fprintf(cplex_file," <=1\n");
                j=i+2;
                constraint2=constraint2->next;
            }
            else if (LAT_ARR[constraint2->x][constraint2->y][constraint2->z].var==1)
            {
                constraint2=constraint2->next;
            }
        }
 }
fprintf(cplex_file,"\n");
fprintf(cplex_file,"End\n");
fclose(cplex_file);
cplex_file=fopen("input_cplex2.lp","w");
 /*************************************************************/
 /***************This is the third constraint***************/
fprintf(cplex_file,"c3:\n");
 for (i=0; i<p_p-1; i++)
 {

    constraint3=B[i].Head;
    while(constraint3!=NULL)
    {
                foundx=0;
                foundy=0;
                foundz=0;
```

```
                foundx=constraint3->x-middle;
                foundy=constraint3->y-middle;
                foundz=constraint3->z-middle;
                LAT_ARR[constraint3->x][constraint3->y][constraint3->z].x = LAT_ARR[middle][middle][middle].x +
(foundx*Lx);
                LAT_ARR[constraint3->x][constraint3->y][constraint3->z].y = LAT_ARR[middle][middle][middle].y +
(foundy*Ly);
                LAT_ARR[constraint3->x][constraint3->y][constraint3->z].z = LAT_ARR[middle][middle][middle].z +
(foundz*Lz);


                fprintf(cplex_file,"X(%d,(%d,%d,%d)) + ",i+2,constraint3->x-1,constraint3->y,constraint3->z);
                fprintf(cplex_file,"X(%d,(%d,%d,%d)) + ",i+2,constraint3->x+1,constraint3->y,constraint3->z);
                fprintf(cplex_file,"X(%d,(%d,%d,%d)) + ",i+2,constraint3->x,constraint3->y-1,constraint3->z);
                fprintf(cplex_file,"X(%d,(%d,%d,%d)) + ",i+2,constraint3->x,constraint3->y+1,constraint3->z);
                fprintf(cplex_file,"X(%d,(%d,%d,%d)) + ",i+2,constraint3->x,constraint3->y,constraint3->z-1);
                fprintf(cplex_file,"X(%d,(%d,%d,%d)) - ",i+2,constraint3->x,constraint3->y,constraint3->z+1);
                fprintf(cplex_file,"X(%d,(%d,%d,%d))",i+1,constraint3->x,constraint3->y,constraint3->z);
                fprintf(cplex_file," >= 0");
                fprintf(cplex_file,"\n");

                constraint3=constraint3->next;
    }
 }
   fprintf(cplex_file,"\n");
   fprintf(cplex_file,"End\n");
```

```
    fclose(cplex_file);
    system("pause");
}
/*************************FUNTCION read_pdb_file*****************************/
/****************************************************************************/
int read_pdb_file(char initial_file[], char initial_elements[], int max)

{
      FILE *fp;
      FILE *data;
      char protein[SIZE];
      int lines=0;
      if ((fp = fopen("initial_file.txt", "r")) == NULL)
      {
        perror("fopen");
        return 0;
      }
      data = fopen("protein.txt", "w");
          while (fgets(initial_elements, SIZE, fp)!= NULL)
          {
              while(lines!=MAX)
              {
              if((strncmp(initial_elements,"ATOM",4) == 0)&&(strncmp(initial_elements+13,"CA", 2)) ==0)
                  {
                      strncpy(protein, initial_elements+31, 23);
```

```
                    protein[23] = '\n';
                    fputs(protein, data);
                    lines++;
                  }
                  fgets(initial_elements, SIZE, fp)== NULL;
                }
              }
              fclose(fp);
              fclose(data);
              return 1;

}
/*****************************************************************************/
/*************************FUNCTION protein_points*************************/
/*****************************************************************************/
int protein_points(char protein[], double protein_coordinates[MAX][3], int max)
{



    FILE *data;
    int row;
    int col;
    int var = 0;
    int i = 0;
    int j = 0;
```

```
    double x;
    double y;
    double z;
    char ch;
    int prosimo;

    double value1 = 0.0;
    double value2 = 0.0;
    double value;

    int dig = 1;
    int number = 0;
    int comps = 0;

    int decimal = 0;



if ((data = fopen("protein.txt", "r")) == NULL)
{
   perror("fopen");
   return 0;
}
    while((ch = fgetc(data))!= EOF)
    {
                if ((ch == ' ') || (ch == '\n'))
```

```
{
                if (number == 0)
                {
                                prosimo = 1;
                }
                else
                {

                                for (x=0; x<comps; x++)
                                {
                                    dig = dig*10;
                                }
                                value2 = (value2/dig);
                                value = value1 + value2;
                                value = value*prosimo;
                                if(i<MAX)
                                {
                                      protein_coordinates[i][j] = value;
                                }
                                      i = i + ((j+1)/3);
                                      j = (j+1)%3;
                                value1 = 0.0;
                                value2 = 0.0;
                                value = 0.0;
                                dig = 1;
                                comps = 0;
```

```
                                number = 0;
                                decimal = 0;
                }
}
else
{
        number = 1;
}
if(ch == '.')
{
        decimal = 1;
}
 if (ch == '-')
{
         prosimo = -1;
}
if(decimal == 0)
{
          if ((ch >= '0') && (ch <= '9'))
         {
                  var = ch-'0';
                  if (value1 == 0.0)
                  {
                          value1 = var;
                  }
                 else
```

```
                        {
                                value1 = (value1*10) + var;
                        }
                }
        }
        else
        {
                if ((ch >= '0') && (ch <= '9'))
                {
                        var = ch - '0';
                        if (value2 == 0.0)
                        {
                                value2 = var;
                                comps++;
                        }
                        else
                        {
                                value2 = (value2*10) + var;
                                comps++;
                        }
                }
        }
    }
    fclose(data);

x = protein_coordinates[i/2][0];
```

```
  y = protein_coordinates[i/2][1];
  z = protein_coordinates[i/2][2];

return i;
}
/*************************************************************/
/**********FUNCTION initial*********/
/*************************************/
void initial  (ARRAY_NODE *B)
{

        B->info=0;
        B->Head=NULL;
}
/*************************************/
/**********FUNCTION find_new_node**********/
/*************************************/
int find_new_node(NODE *read, int C[6][3],ARRAY_NODE *B)
{

    int i;
    int j;

    /*briskw ta x' y' z' kai ta apothikeuw se enan pinaka ton C*/

    /******INITIALIZE*********/
```

```
    for (i=0; i<6; i++)
    {
            for (j=0; j<3; j++)
            {
                    C[i][j]=0;
            }
    }

if (B->info==1)
{
        i=0;
        while(i!=6)
        {
              if (i==0)
              {
                     for (j=0; j<3; j++)
                     {
                                if (j==0)
                                {
                                   C[i][j]=read->x-1;
                                }
                                if (j==1)
                                {
                                   C[i][j]=read->y;
                                }
                                if (j==2)
```

```
                {
                    C[i][j]=read->z;
                }
            }
            i++;
        }
        if (i==1)
        {
            for (j=0; j<3; j++)
                {
                    if (j==0)
                    {
                        C[i][j]=read->x+1;
                    }
                    if (j==1)
                    {
                        C[i][j]=read->y;
                    }
                    if (j==2)
                    {
                        C[i][j]=read->z;
                    }
                }
            i++;
        }
        if (i==2)
```

```
{
        for (j=0; j<3; j++)
        {
                    if (j==0)
                    {
                       C[i][j]=read->x;
                    }
                    if (j==1)
                    {
                       C[i][j]=read->y-1;
                    }
                    if (j==2)
                    {
                       C[i][j]=read->z;
                    }
        }
        i++;
}
if (i==3)
{
        for (j=0; j<3; j++)
        {
                    if (j==0)
                    {
                       C[i][j]=read->x;
                    }
```

```
                if (j==1)
                {
                    C[i][j]=read->y+1;
                }
                if (j==2)
                {
                    C[i][j]=read->z;
                }
            }
            i++;
        }
        if (i==4)
        {
            for (j=0; j<3; j++)
            {
                if (j==0)
                {
                    C[i][j]=read->x;
                }
                if (j==1)
                {
                    C[i][j]=read->y;
                }
                if (j==2)
                {
                    C[i][j]=read->z-1;
```

```
                                        }
                                }
                                i++;
                        }
                        if (i==5)
                        {
                                for (j=0; j<3; j++)
                                {
                                        if (j==0)
                                        {
                                           C[i][j]=read->x;
                                        }
                                        if (j==1)
                                        {
                                           C[i][j]=read->y;
                                        }
                                        if (j==2)
                                        {
                                           C[i][j]=read->z+1;
                                        }
                                }
                                i++;
                        }
                }
        }
else if((B->info!=1)&&(read!=NULL))
```

```
{
    i=0;
    while(i!=6)
    {
        if (i==0)
        {
            for (j=0; j<3; j++)
            {
                if (j==0)
                {
                    C[i][j]=read->x-1;
                }
                if (j==1)
                {
                    C[i][j]=read->y;
                }
                if (j==2)
                {
                    C[i][j]=read->z;
                }
            }
            i++;
        }
        if (i==1)
        {
            for (j=0; j<3; j++)
```

```
        {
                    if (j==0)
                    {
                       C[i][j]=read->x+1;
                    }
                    if (j==1)
                    {
                       C[i][j]=read->y;
                    }
                    if (j==2)
                    {
                       C[i][j]=read->z;
                    }
        }
        i++;
}
if (i==2)
{
         for (j=0; j<3; j++)
         {
                     if (j==0)
                     {
                        C[i][j]=read->x;
                     }
                     if (j==1)
                     {
```

```
            C[i][j]=read->y-1;
        }
        if (j==2)
        {
            C[i][j]=read->z;
        }
    }
    i++;
}
if (i==3)
{
    for (j=0; j<3; j++)
    {
        if (j==0)
        {
            C[i][j]=read->x;
        }
        if (j==1)
        {
            C[i][j]=read->y+1;
        }
        if (j==2)
        {
            C[i][j]=read->z;
        }
    }
```

```
        i++;
}
if (i==4)
{
        for (j=0; j<3; j++)
        {
                    if (j==0)
                    {
                       C[i][j]=read->x;
                    }
                    if (j==1)
                    {
                       C[i][j]=read->y;
                    }
                    if (j==2)
                    {
                       C[i][j]=read->z-1;
                    }
        }
        i++;
}
if (i==5)
{
        for (j=0; j<3; j++)
        {
                    if (j==0)
```

```
    {
        C[i][j]=read->x;
    }
    if (j==1)
    {
        C[i][j]=read->y;
    }
    if (j==2)
    {
        C[i][j]=read->z+1;
    }
            }
            i++;
        }
    }
    return 1;
}
else if((B->info!=1)&&(read==NULL))
{
    return 2;
}
}
}
/*****************************************************/
/*************FUNCTION ln_create**********************/
/*****************************************************/
void ln_create(ARRAY_NODE *B, int C[6][3], int p_p)
```

```
{
    NODE *ptr;
    NODE *ptrB;
    NODE *control;
    NODE *prev_control;
    int i=0;
    int j;
    int empty;
    int middle;
    int check;


            while (i!=6)
            {
                  j=0;
                  while(j!=3)
                  {
                              if (C[i][j]==0)
                              {
                                     empty=0;
                                     j++;
                              }
                              else
                              {
                                     empty=1;
                                     j=3;
```

```
                                    i=5;
                               }
                    }
                    i++;
          }

if ((B->Head==NULL)&&(empty==0))
{
     middle=(p_p-1);
     struct node *ptr = (struct node *)malloc(sizeof(struct node));
     B->Head=ptr;
     ptr->x=middle;
     ptr->y=middle;
     ptr->z=middle;
     ptr->next=NULL;
     B->info++;
}
else if ((B->Head==NULL)&&(empty==1))
{
      struct node *ptr = (struct node *)malloc(sizeof(struct node));
      i=0;
      while(i!=6)
      {
                if (i==0)
                {
                     B->Head=ptr;
```

```
                    ptr->x=C[i][0];
                    ptr->y=C[i][1];
                    ptr->z=C[i][2];
                    ptr->next=NULL;
                    i++;
                    B->info++;
               }
               else
               {
                    struct node *ptrB = (struct node *)malloc(sizeof(struct node));
                    ptr->next=ptrB;
                    ptrB->x=C[i][0];
                    ptrB->y=C[i][1];
                    ptrB->z=C[i][2];
                    ptrB->next=NULL;
                    ptr=ptrB;
                    i++;
                    B->info++;
               }
     }
}
else if ((B->Head!=NULL)&&(empty==1))
{
     i=0;
     while(i!=6)
     {
```

```
        control=B->Head;
        check=0;
        while ((control!=NULL)&& (!check))
        {
              if ((control->x==C[i][0]) && (control->y==C[i][1]) && (control->z==C[i][2]))
              {
                    check=1;
              }
              else
              {
                    prev_control=control;
                    control=control->next;
              }
        }
        if(check==0)
        {
                    struct node *ptr=(struct node *)malloc(sizeof(struct node));
                    prev_control->next=ptr;
                    ptr->x=C[i][0];
                    ptr->y=C[i][1];
                    ptr->z=C[i][2];
                    ptr->next=NULL;
                    B->info++;
        }
        i++;
}
```

```
    }
}
/*************************************************************/
```

Παράρτημα Β΄

Παραδείγματα Αρχείων

"input_cplex.lp"

```
Minimize Obj:
0.000000X(1,(4,4,4)) + 48.392643X(2,(3,4,4)) + 8.918243X(2,(5,4,4)) +
47.997443X(2,(4,3,4)) +
9.313443X(2,(4,5,4)) + 21.078243X(2,(4,4,3)) + 36.232643X(2,(4,4,5)) +
155.179148X(3,(2,4,4)) +
32.636748X(3,(4,4,4)) + 116.327948X(3,(3,3,4)) + 71.487948X(3,(3,5,4)) +
112.163148X(3,(3,4,3)) +
75.652748X(3,(3,4,5)) + 25.614348X(3,(6,4,4)) + 51.545548X(3,(5,3,4)) +
6.705548X(3,(5,5,4)) +
47.380748X(3,(5,4,3)) + 10.870348X(3,(5,4,5)) + 135.236748X(3,(4,2,4)) +
102.191948X(3,(4,3,3)) +
65.681548X(3,(4,3,5)) + 45.556748X(3,(4,6,4)) + 57.351948X(3,(4,5,3)) +
20.841548X(3,(4,5,5)) +
126.907148X(3,(4,4,2)) + 53.886348X(3,(4,4,6)) + 269.218537X(4,(1,4,4)) +
83.520137X(4,(3,4,4)) +
171.338137X(4,(2,3,4)) + 181.400537X(4,(2,5,4)) + 202.954137X(4,(2,4,3)) +
149.784537X(4,(2,4,5)) +
13.341737X(4,(5,4,4)) + 43.399737X(4,(4,3,4)) + 53.462137X(4,(4,5,4)) +
75.015737X(4,(4,4,3)) +
21.846137X(4,(4,4,5)) + 131.217737X(4,(3,2,4)) + 133.953737X(4,(3,3,3)) +
80.784137X(4,(3,3,5)) +
151.342537X(4,(3,6,4)) + 144.016137X(4,(3,5,3)) + 90.846537X(4,(3,5,5)) +
194.449737X(4,(3,4,2)) +
88.110537X(4,(3,4,6)) + 58.683337X(4,(7,4,4)) + 30.981337X(4,(6,3,4)) +
41.043737X(4,(6,5,4)) +
62.597337X(4,(6,4,3)) + 9.427737X(4,(6,4,5)) + 61.039337X(4,(5,2,4)) +
63.775337X(4,(5,3,3)) +
10.605737X(4,(5,3,5)) + 81.164137X(4,(5,6,4)) + 73.837737X(4,(5,5,3)) +
20.668137X(4,(5,5,5)) +
124.271337X(4,(5,4,2)) + 17.932137X(4,(5,4,6)) + 148.857337X(4,(4,1,4)) +
122.713337X(4,(4,2,3)) +
69.543737X(4,(4,2,5)) + 154.329337X(4,(4,3,2)) + 47.990137X(4,(4,3,6)) +
179.044537X(4,(4,7,4)) +
142.838137X(4,(4,6,3)) + 89.668537X(4,(4,6,5)) + 164.391737X(4,(4,5,2)) +
```

```
58.052537X(4,(4,5,6)) +
243.705337X(4,(4,4,1)) + 84.196537X(4,(4,4,7)) + 460.484762X(5,(0,4,4)) +
197.783162X(5,(2,4,4)) +
301.257162X(5,(1,3,4)) + 357.010762X(5,(1,5,4)) + 341.240762X(5,(1,4,3)) +
317.027162X(5,(1,4,5)) +
50.601562X(5,(4,4,4)) + 96.315562X(5,(3,3,4)) + 152.069162X(5,(3,5,4)) +
136.299162X(5,(3,4,3)) +
112.085562X(5,(3,4,5)) + 199.789562X(5,(2,2,4)) + 210.893162X(5,(2,3,3)) +
186.679562X(5,(2,3,5)) +
311.296762X(5,(2,6,4)) + 266.646762X(5,(2,5,3)) + 242.433162X(5,(2,5,5)) +
279.756762X(5,(2,4,2)) +
231.329562X(5,(2,4,6)) + 18.939962X(5,(6,4,4)) + 6.893962X(5,(5,3,4)) +
62.647562X(5,(5,5,4)) +
46.877562X(5,(5,4,3)) + 22.663962X(5,(5,4,5)) + 52.607962X(5,(4,2,4)) +
63.711562X(5,(4,3,3)) +
39.497962X(5,(4,3,5)) + 164.115162X(5,(4,6,4)) + 119.465162X(5,(4,5,3)) +
95.251562X(5,(4,5,5)) +
132.575162X(5,(4,4,2)) + 84.147962X(5,(4,4,6)) + 156.081962X(5,(3,1,4)) +
138.305562X(5,(3,2,3)) +
114.091962X(5,(3,2,5)) + 178.289162X(5,(3,3,2)) + 129.861962X(5,(3,3,6)) +
323.342762X(5,(3,7,4)) +
249.812762X(5,(3,6,3)) + 225.599162X(5,(3,6,5)) + 234.042762X(5,(3,5,2)) +
185.615562X(5,(3,5,6)) +
276.032762X(5,(3,4,1)) + 203.391962X(5,(3,4,7)) + 102.798362X(5,(8,4,4)) +
32.992362X(5,(7,3,4)) +
88.745962X(5,(7,5,4)) + 72.975962X(5,(7,4,3)) + 48.762362X(5,(7,4,5)) +
20.946362X(5,(6,2,4)) +
32.049962X(5,(6,3,3)) + 7.836362X(5,(6,3,5)) + 132.453562X(5,(6,6,4)) +
87.803562X(5,(6,5,3)) +
63.589962X(5,(6,5,5)) + 100.913562X(5,(6,4,2)) + 52.486362X(5,(6,4,6)) +
66.660362X(5,(5,1,4)) +
48.883962X(5,(5,2,3)) + 24.670362X(5,(5,2,5)) + 88.867562X(5,(5,3,2)) +
40.440362X(5,(5,3,6)) +
233.921162X(5,(5,7,4)) + 160.391162X(5,(5,6,3)) + 136.177562X(5,(5,6,5)) +
144.621162X(5,(5,5,2)) +
96.193962X(5,(5,5,6)) + 186.611162X(5,(5,4,1)) + 113.970362X(5,(5,4,7)) +
```

```
170.134362X(5,(4,0,4)) +
123.477962X(5,(4,1,3)) + 99.264362X(5,(4,1,5)) + 134.581562X(5,(4,2,2)) +
86.154362X(5,(4,2,6)) +
203.445162X(5,(4,3,1)) + 130.804362X(5,(4,3,7)) + 393.148762X(5,(4,8,4)) +
290.738762X(5,(4,7,3)) +
266.525162X(5,(4,7,5)) + 246.088762X(5,(4,6,2)) + 197.661562X(5,(4,6,6)) +
259.198762X(5,(4,5,1)) +
186.557962X(5,(4,5,7)) + 330.068762X(5,(4,4,0)) + 233.214362X(5,(4,4,8))
Subject To
c1:
X(1,(4,4,4))  = 1
X(2,(3,4,4)) +  X(2,(5,4,4)) +  X(2,(4,3,4)) +  X(2,(4,5,4)) +  X(2,(4,4,3)) +
X(2,(4,4,5))  = 1
X(3,(2,4,4)) +  X(3,(4,4,4)) +  X(3,(3,3,4)) +  X(3,(3,5,4)) +  X(3,(3,4,3)) +
X(3,(3,4,5)) +  X(3,(6,4,4)) +
X(3,(5,3,4)) +  X(3,(5,5,4)) +  X(3,(5,4,3)) +  X(3,(5,4,5)) +  X(3,(4,2,4)) +
X(3,(4,3,3)) +  X(3,(4,3,5)) +
X(3,(4,6,4)) +  X(3,(4,5,3)) +  X(3,(4,5,5)) +  X(3,(4,4,2)) +  X(3,(4,4,6))  =
1
X(4,(1,4,4)) +  X(4,(3,4,4)) +  X(4,(2,3,4)) +  X(4,(2,5,4)) +  X(4,(2,4,3)) +
X(4,(2,4,5)) +  X(4,(5,4,4)) +
X(4,(4,3,4)) +  X(4,(4,5,4)) +  X(4,(4,4,3)) +  X(4,(4,4,5)) +  X(4,(3,2,4)) +
X(4,(3,3,3)) +  X(4,(3,3,5)) +
X(4,(3,6,4)) +  X(4,(3,5,3)) +  X(4,(3,5,5)) +  X(4,(3,4,2)) +  X(4,(3,4,6)) +
X(4,(7,4,4)) +  X(4,(6,3,4)) +
X(4,(6,5,4)) +  X(4,(6,4,3)) +  X(4,(6,4,5)) +  X(4,(5,2,4)) +  X(4,(5,3,3)) +
X(4,(5,3,5)) +  X(4,(5,6,4)) +
X(4,(5,5,3)) +  X(4,(5,5,5)) +  X(4,(5,4,2)) +  X(4,(5,4,6)) +  X(4,(4,1,4)) +
X(4,(4,2,3)) +  X(4,(4,2,5)) +
X(4,(4,3,2)) +  X(4,(4,3,6)) +  X(4,(4,7,4)) +  X(4,(4,6,3)) +  X(4,(4,6,5)) +
X(4,(4,5,2)) +  X(4,(4,5,6)) +
X(4,(4,4,1)) +  X(4,(4,4,7))  = 1
X(5,(0,4,4)) +  X(5,(2,4,4)) +  X(5,(1,3,4)) +  X(5,(1,5,4)) +  X(5,(1,4,3)) +
X(5,(1,4,5)) +  X(5,(4,4,4)) +
X(5,(3,3,4)) +  X(5,(3,5,4)) +  X(5,(3,4,3)) +  X(5,(3,4,5)) +  X(5,(2,2,4)) +
X(5,(2,3,3)) +  X(5,(2,3,5)) +
```

```
X(5,(2,6,4)) +  X(5,(2,5,3)) +  X(5,(2,5,5)) +  X(5,(2,4,2)) +  X(5,(2,4,6)) +
X(5,(6,4,4)) +  X(5,(5,3,4)) +
X(5,(5,5,4)) +  X(5,(5,4,3)) +  X(5,(5,4,5)) +  X(5,(4,2,4)) +  X(5,(4,3,3)) +
X(5,(4,3,5)) +  X(5,(4,6,4)) +
X(5,(4,5,3)) +  X(5,(4,5,5)) +  X(5,(4,4,2)) +  X(5,(4,4,6)) +  X(5,(3,1,4)) +
X(5,(3,2,3)) +  X(5,(3,2,5)) +
X(5,(3,3,2)) +  X(5,(3,3,6)) +  X(5,(3,7,4)) +  X(5,(3,6,3)) +  X(5,(3,6,5)) +
X(5,(3,5,2)) +  X(5,(3,5,6)) +
X(5,(3,4,1)) +  X(5,(3,4,7)) +  X(5,(8,4,4)) +  X(5,(7,3,4)) +  X(5,(7,5,4)) +
X(5,(7,4,3)) +  X(5,(7,4,5)) +
X(5,(6,2,4)) +  X(5,(6,3,3)) +  X(5,(6,3,5)) +  X(5,(6,6,4)) +  X(5,(6,5,3)) +
X(5,(6,5,5)) +  X(5,(6,4,2)) +
X(5,(6,4,6)) +  X(5,(5,1,4)) +  X(5,(5,2,3)) +  X(5,(5,2,5)) +  X(5,(5,3,2)) +
X(5,(5,3,6)) +  X(5,(5,7,4)) +
X(5,(5,6,3)) +  X(5,(5,6,5)) +  X(5,(5,5,2)) +  X(5,(5,5,6)) +  X(5,(5,4,1)) +
X(5,(5,4,7)) +  X(5,(4,0,4)) +
X(5,(4,1,3)) +  X(5,(4,1,5)) +  X(5,(4,2,2)) +  X(5,(4,2,6)) +  X(5,(4,3,1)) +
X(5,(4,3,7)) +  X(5,(4,8,4)) +
X(5,(4,7,3)) +  X(5,(4,7,5)) +  X(5,(4,6,2)) +  X(5,(4,6,6)) +  X(5,(4,5,1)) +
X(5,(4,5,7)) +  X(5,(4,4,0)) +
X(5,(4,4,8))  = 1
Binaries
X(1,(4,4,4))
X(2,(3,4,4))
X(2,(5,4,4))
X(2,(4,3,4))
X(2,(4,5,4))
X(2,(4,4,3))
X(2,(4,4,5))
X(3,(2,4,4))
X(3,(4,4,4))
X(3,(3,3,4))
X(3,(3,5,4))
X(3,(3,4,3))
X(3,(3,4,5))
X(3,(6,4,4))
```

```
X(3,(5,3,4))
X(3,(5,5,4))
X(3,(5,4,3))
X(3,(5,4,5))
X(3,(4,2,4))
X(3,(4,3,3))
X(3,(4,3,5))
X(3,(4,6,4))
X(3,(4,5,3))
X(3,(4,5,5))
X(3,(4,4,2))
X(3,(4,4,6))
X(4,(1,4,4))
X(4,(3,4,4))
X(4,(2,3,4))
X(4,(2,5,4))
X(4,(2,4,3))
X(4,(2,4,5))
X(4,(5,4,4))
X(4,(4,3,4))
X(4,(4,5,4))
X(4,(4,4,3))
X(4,(4,4,5))
X(4,(3,2,4))
X(4,(3,3,3))
X(4,(3,3,5))
X(4,(3,6,4))
X(4,(3,5,3))
X(4,(3,5,5))
X(4,(3,4,2))
X(4,(3,4,6))
X(4,(7,4,4))
X(4,(6,3,4))
X(4,(6,5,4))
X(4,(6,4,3))
X(4,(6,4,5))
```

```
X(4,(5,2,4))
X(4,(5,3,3))
X(4,(5,3,5))
X(4,(5,6,4))
X(4,(5,5,3))
X(4,(5,5,5))
X(4,(5,4,2))
X(4,(5,4,6))
X(4,(4,1,4))
X(4,(4,2,3))
X(4,(4,2,5))
X(4,(4,3,2))
X(4,(4,3,6))
X(4,(4,7,4))
X(4,(4,6,3))
X(4,(4,6,5))
X(4,(4,5,2))
X(4,(4,5,6))
X(4,(4,4,1))
X(4,(4,4,7))
X(5,(0,4,4))
X(5,(2,4,4))
X(5,(1,3,4))
X(5,(1,5,4))
X(5,(1,4,3))
X(5,(1,4,5))
X(5,(4,4,4))
X(5,(3,3,4))
X(5,(3,5,4))
X(5,(3,4,3))
X(5,(3,4,5))
X(5,(2,2,4))
X(5,(2,3,3))
X(5,(2,3,5))
X(5,(2,6,4))
X(5,(2,5,3))
```

```
X(5,(2,5,5))
X(5,(2,4,2))
X(5,(2,4,6))
X(5,(6,4,4))
X(5,(5,3,4))
X(5,(5,5,4))
X(5,(5,4,3))
X(5,(5,4,5))
X(5,(4,2,4))
X(5,(4,3,3))
X(5,(4,3,5))
X(5,(4,6,4))
X(5,(4,5,3))
X(5,(4,5,5))
X(5,(4,4,2))
X(5,(4,4,6))
X(5,(3,1,4))
X(5,(3,2,3))
X(5,(3,2,5))
X(5,(3,3,2))
X(5,(3,3,6))
X(5,(3,7,4))
X(5,(3,6,3))
X(5,(3,6,5))
X(5,(3,5,2))
X(5,(3,5,6))
X(5,(3,4,1))
X(5,(3,4,7))
X(5,(8,4,4))
X(5,(7,3,4))
X(5,(7,5,4))
X(5,(7,4,3))
X(5,(7,4,5))
X(5,(6,2,4))
X(5,(6,3,3))
X(5,(6,3,5))
```

```
X(5,(6,6,4))
X(5,(6,5,3))
X(5,(6,5,5))
X(5,(6,4,2))
X(5,(6,4,6))
X(5,(5,1,4))
X(5,(5,2,3))
X(5,(5,2,5))
X(5,(5,3,2))
X(5,(5,3,6))
X(5,(5,7,4))
X(5,(5,6,3))
X(5,(5,6,5))
X(5,(5,5,2))
X(5,(5,5,6))
X(5,(5,4,1))
X(5,(5,4,7))
X(5,(4,0,4))
X(5,(4,1,3))
X(5,(4,1,5))
X(5,(4,2,2))
X(5,(4,2,6))
X(5,(4,3,1))
X(5,(4,3,7))
X(5,(4,8,4))
X(5,(4,7,3))
X(5,(4,7,5))
X(5,(4,6,2))
X(5,(4,6,6))
X(5,(4,5,1))
X(5,(4,5,7))
X(5,(4,4,0))
X(5,(4,4,8))
End
```

"input_cplex1.lp"

```
c2:
X(1,(4,4,4))  + X(3,(4,4,4))  + X(5,(4,4,4))  <=1
X(2,(3,4,4))  + X(4,(3,4,4))  <=1
X(2,(5,4,4))  + X(4,(5,4,4))  <=1
X(2,(4,3,4))  + X(4,(4,3,4))  <=1
X(2,(4,5,4))  + X(4,(4,5,4))  <=1
X(2,(4,4,3))  + X(4,(4,4,3))  <=1
X(2,(4,4,5))  + X(4,(4,4,5))  <=1
X(3,(2,4,4))  + X(5,(2,4,4))  <=1
X(3,(3,3,4))  + X(5,(3,3,4))  <=1
X(3,(3,5,4))  + X(5,(3,5,4))  <=1
X(3,(3,4,3))  + X(5,(3,4,3))  <=1
X(3,(3,4,5))  + X(5,(3,4,5))  <=1
X(3,(6,4,4))  + X(5,(6,4,4))  <=1
X(3,(5,3,4))  + X(5,(5,3,4))  <=1
X(3,(5,5,4))  + X(5,(5,5,4))  <=1
X(3,(5,4,3))  + X(5,(5,4,3))  <=1
X(3,(5,4,5))  + X(5,(5,4,5))  <=1
X(3,(4,2,4))  + X(5,(4,2,4))  <=1
X(3,(4,3,3))  + X(5,(4,3,3))  <=1
X(3,(4,3,5))  + X(5,(4,3,5))  <=1
X(3,(4,6,4))  + X(5,(4,6,4))  <=1
X(3,(4,5,3))  + X(5,(4,5,3))  <=1
X(3,(4,5,5))  + X(5,(4,5,5))  <=1
X(3,(4,4,2))  + X(5,(4,4,2))  <=1
X(3,(4,4,6))  + X(5,(4,4,6))  <=1
X(4,(1,4,4))  <=1
X(4,(2,3,4))  <=1
X(4,(2,5,4))  <=1
X(4,(2,4,3))  <=1
X(4,(2,4,5))  <=1
X(4,(3,2,4))  <=1
X(4,(3,3,3))  <=1
```

```
X(4,(3,3,5))  <=1
X(4,(3,6,4))  <=1
X(4,(3,5,3))  <=1
X(4,(3,5,5))  <=1
X(4,(3,4,2))  <=1
X(4,(3,4,6))  <=1
X(4,(7,4,4))  <=1
X(4,(6,3,4))  <=1
X(4,(6,5,4))  <=1
X(4,(6,4,3))  <=1
X(4,(6,4,5))  <=1
X(4,(5,2,4))  <=1
X(4,(5,3,3))  <=1
X(4,(5,3,5))  <=1
X(4,(5,6,4))  <=1
X(4,(5,5,3))  <=1
X(4,(5,5,5))  <=1
X(4,(5,4,2))  <=1
X(4,(5,4,6))  <=1
X(4,(4,1,4))  <=1
X(4,(4,2,3))  <=1
X(4,(4,2,5))  <=1
X(4,(4,3,2))  <=1
X(4,(4,3,6))  <=1
X(4,(4,7,4))  <=1
X(4,(4,6,3))  <=1
X(4,(4,6,5))  <=1
X(4,(4,5,2))  <=1
X(4,(4,5,6))  <=1
X(4,(4,4,1))  <=1
X(4,(4,4,7))  <=1
X(5,(0,4,4))  <=1
X(5,(1,3,4))  <=1
X(5,(1,5,4))  <=1
X(5,(1,4,3))  <=1
X(5,(1,4,5))  <=1
```

```
X(5,(2,2,4))  <=1
X(5,(2,3,3))  <=1
X(5,(2,3,5))  <=1
X(5,(2,6,4))  <=1
X(5,(2,5,3))  <=1
X(5,(2,5,5))  <=1
X(5,(2,4,2))  <=1
X(5,(2,4,6))  <=1
X(5,(3,1,4))  <=1
X(5,(3,2,3))  <=1
X(5,(3,2,5))  <=1
X(5,(3,3,2))  <=1
X(5,(3,3,6))  <=1
X(5,(3,7,4))  <=1
X(5,(3,6,3))  <=1
X(5,(3,6,5))  <=1
X(5,(3,5,2))  <=1
X(5,(3,5,6))  <=1
X(5,(3,4,1))  <=1
X(5,(3,4,7))  <=1
X(5,(8,4,4))  <=1
X(5,(7,3,4))  <=1
X(5,(7,5,4))  <=1
X(5,(7,4,3))  <=1
X(5,(7,4,5))  <=1
X(5,(6,2,4))  <=1
X(5,(6,3,3))  <=1
X(5,(6,3,5))  <=1
X(5,(6,6,4))  <=1
X(5,(6,5,3))  <=1
X(5,(6,5,5))  <=1
X(5,(6,4,2))  <=1
X(5,(6,4,6))  <=1
X(5,(5,1,4))  <=1
X(5,(5,2,3))  <=1
X(5,(5,2,5))  <=1
```

```
X(5,(5,3,2))  <=1
X(5,(5,3,6))  <=1
X(5,(5,7,4))  <=1
X(5,(5,6,3))  <=1
X(5,(5,6,5))  <=1
X(5,(5,5,2))  <=1
X(5,(5,5,6))  <=1
X(5,(5,4,1))  <=1
X(5,(5,4,7))  <=1
X(5,(4,0,4))  <=1
X(5,(4,1,3))  <=1
X(5,(4,1,5))  <=1
X(5,(4,2,2))  <=1
X(5,(4,2,6))  <=1
X(5,(4,3,1))  <=1
X(5,(4,3,7))  <=1
X(5,(4,8,4))  <=1
X(5,(4,7,3))  <=1
X(5,(4,7,5))  <=1
X(5,(4,6,2))  <=1
X(5,(4,6,6))  <=1
X(5,(4,5,1))  <=1
X(5,(4,5,7))  <=1
X(5,(4,4,0))  <=1
X(5,(4,4,8))  <=1
End
```

“input_cplex2.lp”

```
c3:
X(2,(3,4,4)) + X(2,(5,4,4)) + X(2,(4,3,4)) + X(2,(4,5,4)) + X(2,(4,4,3)) +
X(2,(4,4,5)) - X(1,(4,4,4)) >= 0
X(3,(2,4,4)) + X(3,(4,4,4)) + X(3,(3,3,4)) + X(3,(3,5,4)) + X(3,(3,4,3)) +
X(3,(3,4,5)) - X(2,(3,4,4)) >= 0
X(3,(4,4,4)) + X(3,(6,4,4)) + X(3,(5,3,4)) + X(3,(5,5,4)) + X(3,(5,4,3)) +
X(3,(5,4,5)) - X(2,(5,4,4)) >= 0
X(3,(3,3,4)) + X(3,(5,3,4)) + X(3,(4,2,4)) + X(3,(4,4,4)) + X(3,(4,3,3)) +
X(3,(4,3,5)) - X(2,(4,3,4)) >= 0
X(3,(3,5,4)) + X(3,(5,5,4)) + X(3,(4,4,4)) + X(3,(4,6,4)) + X(3,(4,5,3)) +
X(3,(4,5,5)) - X(2,(4,5,4)) >= 0
X(3,(3,4,3)) + X(3,(5,4,3)) + X(3,(4,3,3)) + X(3,(4,5,3)) + X(3,(4,4,2)) +
X(3,(4,4,4)) - X(2,(4,4,3)) >= 0
X(3,(3,4,5)) + X(3,(5,4,5)) + X(3,(4,3,5)) + X(3,(4,5,5)) + X(3,(4,4,4)) +
X(3,(4,4,6)) - X(2,(4,4,5)) >= 0
X(4,(1,4,4)) + X(4,(3,4,4)) + X(4,(2,3,4)) + X(4,(2,5,4)) + X(4,(2,4,3)) +
X(4,(2,4,5)) - X(3,(2,4,4)) >= 0
X(4,(3,4,4)) + X(4,(5,4,4)) + X(4,(4,3,4)) + X(4,(4,5,4)) + X(4,(4,4,3)) +
X(4,(4,4,5)) - X(3,(4,4,4)) >= 0
X(4,(2,3,4)) + X(4,(4,3,4)) + X(4,(3,2,4)) + X(4,(3,4,4)) + X(4,(3,3,3)) +
X(4,(3,3,5)) - X(3,(3,3,4)) >= 0
X(4,(2,5,4)) + X(4,(4,5,4)) + X(4,(3,4,4)) + X(4,(3,6,4)) + X(4,(3,5,3)) +
X(4,(3,5,5)) - X(3,(3,5,4)) >= 0
X(4,(2,4,3)) + X(4,(4,4,3)) + X(4,(3,3,3)) + X(4,(3,5,3)) + X(4,(3,4,2)) +
X(4,(3,4,4)) - X(3,(3,4,3)) >= 0
X(4,(2,4,5)) + X(4,(4,4,5)) + X(4,(3,3,5)) + X(4,(3,5,5)) + X(4,(3,4,4)) +
X(4,(3,4,6)) - X(3,(3,4,5)) >= 0
X(4,(5,4,4)) + X(4,(7,4,4)) + X(4,(6,3,4)) + X(4,(6,5,4)) + X(4,(6,4,3)) +
X(4,(6,4,5)) - X(3,(6,4,4)) >= 0
X(4,(4,3,4)) + X(4,(6,3,4)) + X(4,(5,2,4)) + X(4,(5,4,4)) + X(4,(5,3,3)) +
X(4,(5,3,5)) - X(3,(5,3,4)) >= 0
X(4,(4,5,4)) + X(4,(6,5,4)) + X(4,(5,4,4)) + X(4,(5,6,4)) + X(4,(5,5,3)) +
X(4,(5,5,5)) - X(3,(5,5,4)) >= 0
X(4,(4,4,3)) + X(4,(6,4,3)) + X(4,(5,3,3)) + X(4,(5,5,3)) + X(4,(5,4,2)) +
```

```
X(4,(5,4,4)) - X(3,(5,4,3)) >= 0
X(4,(4,4,5)) + X(4,(6,4,5)) + X(4,(5,3,5)) + X(4,(5,5,5)) + X(4,(5,4,4)) +
X(4,(5,4,6)) - X(3,(5,4,5)) >= 0
X(4,(3,2,4)) + X(4,(5,2,4)) + X(4,(4,1,4)) + X(4,(4,3,4)) + X(4,(4,2,3)) +
X(4,(4,2,5)) - X(3,(4,2,4)) >= 0
X(4,(3,3,3)) + X(4,(5,3,3)) + X(4,(4,2,3)) + X(4,(4,4,3)) + X(4,(4,3,2)) +
X(4,(4,3,4)) - X(3,(4,3,3)) >= 0
X(4,(3,3,5)) + X(4,(5,3,5)) + X(4,(4,2,5)) + X(4,(4,4,5)) + X(4,(4,3,4)) +
X(4,(4,3,6)) - X(3,(4,3,5)) >= 0
X(4,(3,6,4)) + X(4,(5,6,4)) + X(4,(4,5,4)) + X(4,(4,7,4)) + X(4,(4,6,3)) +
X(4,(4,6,5)) - X(3,(4,6,4)) >= 0
X(4,(3,5,3)) + X(4,(5,5,3)) + X(4,(4,4,3)) + X(4,(4,6,3)) + X(4,(4,5,2)) +
X(4,(4,5,4)) - X(3,(4,5,3)) >= 0
X(4,(3,5,5)) + X(4,(5,5,5)) + X(4,(4,4,5)) + X(4,(4,6,5)) + X(4,(4,5,4)) +
X(4,(4,5,6)) - X(3,(4,5,5)) >= 0
X(4,(3,4,2)) + X(4,(5,4,2)) + X(4,(4,3,2)) + X(4,(4,5,2)) + X(4,(4,4,1)) +
X(4,(4,4,3)) - X(3,(4,4,2)) >= 0
X(4,(3,4,6)) + X(4,(5,4,6)) + X(4,(4,3,6)) + X(4,(4,5,6)) + X(4,(4,4,5)) +
X(4,(4,4,7)) - X(3,(4,4,6)) >= 0
X(5,(0,4,4)) + X(5,(2,4,4)) + X(5,(1,3,4)) + X(5,(1,5,4)) + X(5,(1,4,3)) +
X(5,(1,4,5)) - X(4,(1,4,4)) >= 0
X(5,(2,4,4)) + X(5,(4,4,4)) + X(5,(3,3,4)) + X(5,(3,5,4)) + X(5,(3,4,3)) +
X(5,(3,4,5)) - X(4,(3,4,4)) >= 0
X(5,(1,3,4)) + X(5,(3,3,4)) + X(5,(2,2,4)) + X(5,(2,4,4)) + X(5,(2,3,3)) +
X(5,(2,3,5)) - X(4,(2,3,4)) >= 0
X(5,(1,5,4)) + X(5,(3,5,4)) + X(5,(2,4,4)) + X(5,(2,6,4)) + X(5,(2,5,3)) +
X(5,(2,5,5)) - X(4,(2,5,4)) >= 0
X(5,(1,4,3)) + X(5,(3,4,3)) + X(5,(2,3,3)) + X(5,(2,5,3)) + X(5,(2,4,2)) +
X(5,(2,4,4)) - X(4,(2,4,3)) >= 0
X(5,(1,4,5)) + X(5,(3,4,5)) + X(5,(2,3,5)) + X(5,(2,5,5)) + X(5,(2,4,4)) +
X(5,(2,4,6)) - X(4,(2,4,5)) >= 0
X(5,(4,4,4)) + X(5,(6,4,4)) + X(5,(5,3,4)) + X(5,(5,5,4)) + X(5,(5,4,3)) +
X(5,(5,4,5)) - X(4,(5,4,4)) >= 0
X(5,(3,3,4)) + X(5,(5,3,4)) + X(5,(4,2,4)) + X(5,(4,4,4)) + X(5,(4,3,3)) +
X(5,(4,3,5)) - X(4,(4,3,4)) >= 0
X(5,(3,5,4)) + X(5,(5,5,4)) + X(5,(4,4,4)) + X(5,(4,6,4)) + X(5,(4,5,3)) +
```

```
X(5,(4,5,5)) - X(4,(4,5,4)) >= 0
X(5,(3,4,3)) + X(5,(5,4,3)) + X(5,(4,3,3)) + X(5,(4,5,3)) + X(5,(4,4,2)) +
X(5,(4,4,4)) - X(4,(4,4,3)) >= 0
X(5,(3,4,5)) + X(5,(5,4,5)) + X(5,(4,3,5)) + X(5,(4,5,5)) + X(5,(4,4,4)) +
X(5,(4,4,6)) - X(4,(4,4,5)) >= 0
X(5,(2,2,4)) + X(5,(4,2,4)) + X(5,(3,1,4)) + X(5,(3,3,4)) + X(5,(3,2,3)) +
X(5,(3,2,5)) - X(4,(3,2,4)) >= 0
X(5,(2,3,3)) + X(5,(4,3,3)) + X(5,(3,2,3)) + X(5,(3,4,3)) + X(5,(3,3,2)) +
X(5,(3,3,4)) - X(4,(3,3,3)) >= 0
X(5,(2,3,5)) + X(5,(4,3,5)) + X(5,(3,2,5)) + X(5,(3,4,5)) + X(5,(3,3,4)) +
X(5,(3,3,6)) - X(4,(3,3,5)) >= 0
X(5,(2,6,4)) + X(5,(4,6,4)) + X(5,(3,5,4)) + X(5,(3,7,4)) + X(5,(3,6,3)) +
X(5,(3,6,5)) - X(4,(3,6,4)) >= 0
X(5,(2,5,3)) + X(5,(4,5,3)) + X(5,(3,4,3)) + X(5,(3,6,3)) + X(5,(3,5,2)) +
X(5,(3,5,4)) - X(4,(3,5,3)) >= 0
X(5,(2,5,5)) + X(5,(4,5,5)) + X(5,(3,4,5)) + X(5,(3,6,5)) + X(5,(3,5,4)) +
X(5,(3,5,6)) - X(4,(3,5,5)) >= 0
X(5,(2,4,2)) + X(5,(4,4,2)) + X(5,(3,3,2)) + X(5,(3,5,2)) + X(5,(3,4,1)) +
X(5,(3,4,3)) - X(4,(3,4,2)) >= 0
X(5,(2,4,6)) + X(5,(4,4,6)) + X(5,(3,3,6)) + X(5,(3,5,6)) + X(5,(3,4,5)) +
X(5,(3,4,7)) - X(4,(3,4,6)) >= 0
X(5,(6,4,4)) + X(5,(8,4,4)) + X(5,(7,3,4)) + X(5,(7,5,4)) + X(5,(7,4,3)) +
X(5,(7,4,5)) - X(4,(7,4,4)) >= 0
X(5,(5,3,4)) + X(5,(7,3,4)) + X(5,(6,2,4)) + X(5,(6,4,4)) + X(5,(6,3,3)) +
X(5,(6,3,5)) - X(4,(6,3,4)) >= 0
X(5,(5,5,4)) + X(5,(7,5,4)) + X(5,(6,4,4)) + X(5,(6,6,4)) + X(5,(6,5,3)) +
X(5,(6,5,5)) - X(4,(6,5,4)) >= 0
X(5,(5,4,3)) + X(5,(7,4,3)) + X(5,(6,3,3)) + X(5,(6,5,3)) + X(5,(6,4,2)) +
X(5,(6,4,4)) - X(4,(6,4,3)) >= 0
X(5,(5,4,5)) + X(5,(7,4,5)) + X(5,(6,3,5)) + X(5,(6,5,5)) + X(5,(6,4,4)) +
X(5,(6,4,6)) - X(4,(6,4,5)) >= 0
X(5,(4,2,4)) + X(5,(6,2,4)) + X(5,(5,1,4)) + X(5,(5,3,4)) + X(5,(5,2,3)) +
X(5,(5,2,5)) - X(4,(5,2,4)) >= 0
X(5,(4,3,3)) + X(5,(6,3,3)) + X(5,(5,2,3)) + X(5,(5,4,3)) + X(5,(5,3,2)) +
X(5,(5,3,4)) - X(4,(5,3,3)) >= 0
X(5,(4,3,5)) + X(5,(6,3,5)) + X(5,(5,2,5)) + X(5,(5,4,5)) + X(5,(5,3,4)) +
```

```
X(5,(5,3,6)) - X(4,(5,3,5)) >= 0
X(5,(4,6,4)) + X(5,(6,6,4)) + X(5,(5,5,4)) + X(5,(5,7,4)) + X(5,(5,6,3)) +
X(5,(5,6,5)) - X(4,(5,6,4)) >= 0
X(5,(4,5,3)) + X(5,(6,5,3)) + X(5,(5,4,3)) + X(5,(5,6,3)) + X(5,(5,5,2)) +
X(5,(5,5,4)) - X(4,(5,5,3)) >= 0
X(5,(4,5,5)) + X(5,(6,5,5)) + X(5,(5,4,5)) + X(5,(5,6,5)) + X(5,(5,5,4)) +
X(5,(5,5,6)) - X(4,(5,5,5)) >= 0
X(5,(4,4,2)) + X(5,(6,4,2)) + X(5,(5,3,2)) + X(5,(5,5,2)) + X(5,(5,4,1)) +
X(5,(5,4,3)) - X(4,(5,4,2)) >= 0
X(5,(4,4,6)) + X(5,(6,4,6)) + X(5,(5,3,6)) + X(5,(5,5,6)) + X(5,(5,4,5)) +
X(5,(5,4,7)) - X(4,(5,4,6)) >= 0
X(5,(3,1,4)) + X(5,(5,1,4)) + X(5,(4,0,4)) + X(5,(4,2,4)) + X(5,(4,1,3)) +
X(5,(4,1,5)) - X(4,(4,1,4)) >= 0
X(5,(3,2,3)) + X(5,(5,2,3)) + X(5,(4,1,3)) + X(5,(4,3,3)) + X(5,(4,2,2)) +
X(5,(4,2,4)) - X(4,(4,2,3)) >= 0
X(5,(3,2,5)) + X(5,(5,2,5)) + X(5,(4,1,5)) + X(5,(4,3,5)) + X(5,(4,2,4)) +
X(5,(4,2,6)) - X(4,(4,2,5)) >= 0
X(5,(3,3,2)) + X(5,(5,3,2)) + X(5,(4,2,2)) + X(5,(4,4,2)) + X(5,(4,3,1)) +
X(5,(4,3,3)) - X(4,(4,3,2)) >= 0
X(5,(3,3,6)) + X(5,(5,3,6)) + X(5,(4,2,6)) + X(5,(4,4,6)) + X(5,(4,3,5)) +
X(5,(4,3,7)) - X(4,(4,3,6)) >= 0
X(5,(3,7,4)) + X(5,(5,7,4)) + X(5,(4,6,4)) + X(5,(4,8,4)) + X(5,(4,7,3)) +
X(5,(4,7,5)) - X(4,(4,7,4)) >= 0
X(5,(3,6,3)) + X(5,(5,6,3)) + X(5,(4,5,3)) + X(5,(4,7,3)) + X(5,(4,6,2)) +
X(5,(4,6,4)) - X(4,(4,6,3)) >= 0
X(5,(3,6,5)) + X(5,(5,6,5)) + X(5,(4,5,5)) + X(5,(4,7,5)) + X(5,(4,6,4)) +
X(5,(4,6,6)) - X(4,(4,6,5)) >= 0
X(5,(3,5,2)) + X(5,(5,5,2)) + X(5,(4,4,2)) + X(5,(4,6,2)) + X(5,(4,5,1)) +
X(5,(4,5,3)) - X(4,(4,5,2)) >= 0
X(5,(3,5,6)) + X(5,(5,5,6)) + X(5,(4,4,6)) + X(5,(4,6,6)) + X(5,(4,5,5)) +
X(5,(4,5,7)) - X(4,(4,5,6)) >= 0
X(5,(3,4,1)) + X(5,(5,4,1)) + X(5,(4,3,1)) + X(5,(4,5,1)) + X(5,(4,4,0)) +
X(5,(4,4,2)) - X(4,(4,4,1)) >= 0
X(5,(3,4,7)) + X(5,(5,4,7)) + X(5,(4,3,7)) + X(5,(4,5,7)) + X(5,(4,4,6)) +
X(5,(4,4,8)) - X(4,(4,4,7)) >= 0
End
```

"output1bw8.txt"

```
<?xml version = "1.0" encoding="UTF-8" standalone="yes"?>
<CPLEXSolution version="1.2">
 <header
   problemName="C:\Users\Froso\Desktop\input_cplex.lp"
   solutionName="incumbent"
   solutionIndex="-1"
   objectiveValue="36.25469"
   solutionTypeValue="3"
   solutionTypeString="primal"
   solutionStatusValue="101"
   solutionStatusString="integer optimal solution"
   solutionMethodString="mip"
   primalFeasible="1"
   dualFeasible="1"
   MIPNodes="0"
   MIPIterations="7"
   writeLevel="1"/>
 <quality
   epInt="1e-05"
   epRHS="1e-06"
   maxIntInfeas="0"
   maxPrimalInfeas="0"
   maxX="1"
   maxSlack="1"/>
 <linearConstraints>
  <constraint name="c1" index="0" slack="0"/>
  <constraint name="c3" index="1" slack="0"/>
  <constraint name="c4" index="2" slack="0"/>
  <constraint name="c5" index="3" slack="0"/>
  <constraint name="c6" index="4" slack="0"/>
  <constraint name="c2" index="5" slack="0"/>
  <constraint name="c8" index="6" slack="1"/>
  <constraint name="c9" index="7" slack="0"/>
  <constraint name="c10" index="8" slack="1"/>
```

```
<constraint name="c11" index="9" slack="0"/>
<constraint name="c12" index="10" slack="1"/>
<constraint name="c13" index="11" slack="1"/>
<constraint name="c14" index="12" slack="1"/>
<constraint name="c15" index="13" slack="1"/>
<constraint name="c16" index="14" slack="1"/>
<constraint name="c17" index="15" slack="1"/>
<constraint name="c18" index="16" slack="1"/>
<constraint name="c19" index="17" slack="1"/>
<constraint name="c20" index="18" slack="0"/>
<constraint name="c21" index="19" slack="0"/>
<constraint name="c22" index="20" slack="1"/>
<constraint name="c23" index="21" slack="1"/>
<constraint name="c24" index="22" slack="1"/>
<constraint name="c25" index="23" slack="1"/>
<constraint name="c26" index="24" slack="1"/>
<constraint name="c27" index="25" slack="1"/>
<constraint name="c28" index="26" slack="1"/>
<constraint name="c29" index="27" slack="1"/>
<constraint name="c30" index="28" slack="1"/>
<constraint name="c31" index="29" slack="1"/>
<constraint name="c32" index="30" slack="1"/>
<constraint name="c33" index="31" slack="1"/>
<constraint name="c34" index="32" slack="1"/>
<constraint name="c35" index="33" slack="1"/>
<constraint name="c36" index="34" slack="1"/>
<constraint name="c37" index="35" slack="1"/>
<constraint name="c38" index="36" slack="1"/>
<constraint name="c39" index="37" slack="1"/>
<constraint name="c40" index="38" slack="1"/>
<constraint name="c41" index="39" slack="1"/>
<constraint name="c42" index="40" slack="1"/>
<constraint name="c43" index="41" slack="1"/>
<constraint name="c44" index="42" slack="1"/>
<constraint name="c45" index="43" slack="1"/>
<constraint name="c46" index="44" slack="1"/>
```

```
<constraint name="c47" index="45" slack="1"/>
<constraint name="c48" index="46" slack="1"/>
<constraint name="c49" index="47" slack="1"/>
<constraint name="c50" index="48" slack="1"/>
<constraint name="c51" index="49" slack="1"/>
<constraint name="c52" index="50" slack="1"/>
<constraint name="c53" index="51" slack="1"/>
<constraint name="c54" index="52" slack="1"/>
<constraint name="c55" index="53" slack="1"/>
<constraint name="c56" index="54" slack="1"/>
<constraint name="c57" index="55" slack="1"/>
<constraint name="c58" index="56" slack="1"/>
<constraint name="c59" index="57" slack="1"/>
<constraint name="c60" index="58" slack="1"/>
<constraint name="c61" index="59" slack="1"/>
<constraint name="c62" index="60" slack="1"/>
<constraint name="c63" index="61" slack="1"/>
<constraint name="c64" index="62" slack="1"/>
<constraint name="c65" index="63" slack="1"/>
<constraint name="c66" index="64" slack="1"/>
<constraint name="c67" index="65" slack="1"/>
<constraint name="c68" index="66" slack="1"/>
<constraint name="c69" index="67" slack="1"/>
<constraint name="c70" index="68" slack="1"/>
<constraint name="c71" index="69" slack="1"/>
<constraint name="c72" index="70" slack="1"/>
<constraint name="c73" index="71" slack="1"/>
<constraint name="c74" index="72" slack="1"/>
<constraint name="c75" index="73" slack="1"/>
<constraint name="c76" index="74" slack="1"/>
<constraint name="c77" index="75" slack="1"/>
<constraint name="c78" index="76" slack="1"/>
<constraint name="c79" index="77" slack="1"/>
<constraint name="c80" index="78" slack="1"/>
<constraint name="c81" index="79" slack="1"/>
<constraint name="c82" index="80" slack="1"/>
```

```
<constraint name="c83" index="81" slack="1"/>
<constraint name="c84" index="82" slack="1"/>
<constraint name="c85" index="83" slack="1"/>
<constraint name="c86" index="84" slack="1"/>
<constraint name="c87" index="85" slack="1"/>
<constraint name="c88" index="86" slack="1"/>
<constraint name="c89" index="87" slack="1"/>
<constraint name="c90" index="88" slack="1"/>
<constraint name="c91" index="89" slack="1"/>
<constraint name="c92" index="90" slack="1"/>
<constraint name="c93" index="91" slack="1"/>
<constraint name="c94" index="92" slack="1"/>
<constraint name="c95" index="93" slack="1"/>
<constraint name="c96" index="94" slack="1"/>
<constraint name="c97" index="95" slack="1"/>
<constraint name="c98" index="96" slack="1"/>
<constraint name="c99" index="97" slack="1"/>
<constraint name="c100" index="98" slack="1"/>
<constraint name="c101" index="99" slack="1"/>
<constraint name="c102" index="100" slack="1"/>
<constraint name="c103" index="101" slack="1"/>
<constraint name="c104" index="102" slack="1"/>
<constraint name="c105" index="103" slack="1"/>
<constraint name="c106" index="104" slack="1"/>
<constraint name="c107" index="105" slack="1"/>
<constraint name="c108" index="106" slack="1"/>
<constraint name="c109" index="107" slack="1"/>
<constraint name="c110" index="108" slack="1"/>
<constraint name="c111" index="109" slack="1"/>
<constraint name="c112" index="110" slack="1"/>
<constraint name="c113" index="111" slack="1"/>
<constraint name="c114" index="112" slack="1"/>
<constraint name="c115" index="113" slack="1"/>
<constraint name="c116" index="114" slack="1"/>
<constraint name="c117" index="115" slack="1"/>
<constraint name="c118" index="116" slack="1"/>
```

```
<constraint name="c119" index="117" slack="1"/>
<constraint name="c120" index="118" slack="1"/>
<constraint name="c121" index="119" slack="1"/>
<constraint name="c122" index="120" slack="1"/>
<constraint name="c123" index="121" slack="1"/>
<constraint name="c124" index="122" slack="1"/>
<constraint name="c125" index="123" slack="1"/>
<constraint name="c126" index="124" slack="1"/>
<constraint name="c127" index="125" slack="1"/>
<constraint name="c128" index="126" slack="1"/>
<constraint name="c129" index="127" slack="1"/>
<constraint name="c130" index="128" slack="1"/>
<constraint name="c131" index="129" slack="1"/>
<constraint name="c132" index="130" slack="1"/>
<constraint name="c133" index="131" slack="1"/>
<constraint name="c134" index="132" slack="1"/>
<constraint name="c135" index="133" slack="1"/>
<constraint name="c3" index="134" slack="0"/>
<constraint name="c137" index="135" slack="0"/>
<constraint name="c138" index="136" slack="-1"/>
<constraint name="c139" index="137" slack="0"/>
<constraint name="c140" index="138" slack="0"/>
<constraint name="c141" index="139" slack="0"/>
<constraint name="c142" index="140" slack="0"/>
<constraint name="c143" index="141" slack="0"/>
<constraint name="c144" index="142" slack="-1"/>
<constraint name="c145" index="143" slack="0"/>
<constraint name="c146" index="144" slack="0"/>
<constraint name="c147" index="145" slack="0"/>
<constraint name="c148" index="146" slack="0"/>
<constraint name="c149" index="147" slack="-1"/>
<constraint name="c150" index="148" slack="-1"/>
<constraint name="c151" index="149" slack="0"/>
<constraint name="c152" index="150" slack="-1"/>
<constraint name="c153" index="151" slack="-1"/>
<constraint name="c154" index="152" slack="0"/>
```

```
<constraint name="c155" index="153" slack="0"/>
<constraint name="c156" index="154" slack="0"/>
<constraint name="c157" index="155" slack="0"/>
<constraint name="c158" index="156" slack="0"/>
<constraint name="c159" index="157" slack="0"/>
<constraint name="c160" index="158" slack="0"/>
<constraint name="c161" index="159" slack="0"/>
<constraint name="c162" index="160" slack="0"/>
<constraint name="c163" index="161" slack="0"/>
<constraint name="c164" index="162" slack="0"/>
<constraint name="c165" index="163" slack="0"/>
<constraint name="c166" index="164" slack="0"/>
<constraint name="c167" index="165" slack="0"/>
<constraint name="c168" index="166" slack="0"/>
<constraint name="c169" index="167" slack="-1"/>
<constraint name="c170" index="168" slack="0"/>
<constraint name="c171" index="169" slack="0"/>
<constraint name="c172" index="170" slack="0"/>
<constraint name="c173" index="171" slack="0"/>
<constraint name="c174" index="172" slack="0"/>
<constraint name="c175" index="173" slack="0"/>
<constraint name="c176" index="174" slack="0"/>
<constraint name="c177" index="175" slack="0"/>
<constraint name="c178" index="176" slack="0"/>
<constraint name="c179" index="177" slack="0"/>
<constraint name="c180" index="178" slack="0"/>
<constraint name="c181" index="179" slack="0"/>
<constraint name="c182" index="180" slack="-1"/>
<constraint name="c183" index="181" slack="0"/>
<constraint name="c184" index="182" slack="0"/>
<constraint name="c185" index="183" slack="0"/>
<constraint name="c186" index="184" slack="-1"/>
<constraint name="c187" index="185" slack="-1"/>
<constraint name="c188" index="186" slack="-1"/>
<constraint name="c189" index="187" slack="0"/>
<constraint name="c190" index="188" slack="0"/>
```

```
  <constraint name="c191" index="189" slack="0"/>
  <constraint name="c192" index="190" slack="0"/>
  <constraint name="c193" index="191" slack="0"/>
  <constraint name="c194" index="192" slack="0"/>
  <constraint name="c195" index="193" slack="0"/>
  <constraint name="c196" index="194" slack="0"/>
  <constraint name="c197" index="195" slack="0"/>
  <constraint name="c198" index="196" slack="0"/>
  <constraint name="c199" index="197" slack="0"/>
  <constraint name="c200" index="198" slack="0"/>
  <constraint name="c201" index="199" slack="0"/>
  <constraint name="c202" index="200" slack="0"/>
  <constraint name="c203" index="201" slack="0"/>
  <constraint name="c204" index="202" slack="0"/>
  <constraint name="c205" index="203" slack="0"/>
 </linearConstraints>
 <variables>
  <variable name="X(1,(4,4,4))" index="0" value="1"/>
  <variable name="X(2,(3,4,4))" index="1" value="0"/>
  <variable name="X(2,(5,4,4))" index="2" value="0"/>
  <variable name="X(2,(4,3,4))" index="3" value="0"/>
  <variable name="X(2,(4,5,4))" index="4" value="1"/>
  <variable name="X(2,(4,4,3))" index="5" value="0"/>
  <variable name="X(2,(4,4,5))" index="6" value="0"/>
  <variable name="X(3,(2,4,4))" index="7" value="0"/>
  <variable name="X(3,(4,4,4))" index="8" value="0"/>
  <variable name="X(3,(3,3,4))" index="9" value="0"/>
  <variable name="X(3,(3,5,4))" index="10" value="0"/>
  <variable name="X(3,(3,4,3))" index="11" value="0"/>
  <variable name="X(3,(3,4,5))" index="12" value="0"/>
  <variable name="X(3,(6,4,4))" index="13" value="0"/>
  <variable name="X(3,(5,3,4))" index="14" value="0"/>
  <variable name="X(3,(5,5,4))" index="15" value="1"/>
  <variable name="X(3,(5,4,3))" index="16" value="0"/>
  <variable name="X(3,(5,4,5))" index="17" value="0"/>
  <variable name="X(3,(4,2,4))" index="18" value="0"/>
```

```
<variable name="X(3,(4,3,3))" index="19" value="0"/>
<variable name="X(3,(4,3,5))" index="20" value="0"/>
<variable name="X(3,(4,6,4))" index="21" value="0"/>
<variable name="X(3,(4,5,3))" index="22" value="0"/>
<variable name="X(3,(4,5,5))" index="23" value="0"/>
<variable name="X(3,(4,4,2))" index="24" value="0"/>
<variable name="X(3,(4,4,6))" index="25" value="0"/>
<variable name="X(4,(1,4,4))" index="26" value="0"/>
<variable name="X(4,(3,4,4))" index="27" value="0"/>
<variable name="X(4,(2,3,4))" index="28" value="0"/>
<variable name="X(4,(2,5,4))" index="29" value="0"/>
<variable name="X(4,(2,4,3))" index="30" value="0"/>
<variable name="X(4,(2,4,5))" index="31" value="0"/>
<variable name="X(4,(5,4,4))" index="32" value="1"/>
<variable name="X(4,(4,3,4))" index="33" value="0"/>
<variable name="X(4,(4,5,4))" index="34" value="0"/>
<variable name="X(4,(4,4,3))" index="35" value="0"/>
<variable name="X(4,(4,4,5))" index="36" value="0"/>
<variable name="X(4,(3,2,4))" index="37" value="0"/>
<variable name="X(4,(3,3,3))" index="38" value="0"/>
<variable name="X(4,(3,3,5))" index="39" value="0"/>
<variable name="X(4,(3,6,4))" index="40" value="0"/>
<variable name="X(4,(3,5,3))" index="41" value="0"/>
<variable name="X(4,(3,5,5))" index="42" value="0"/>
<variable name="X(4,(3,4,2))" index="43" value="0"/>
<variable name="X(4,(3,4,6))" index="44" value="0"/>
<variable name="X(4,(7,4,4))" index="45" value="0"/>
<variable name="X(4,(6,3,4))" index="46" value="0"/>
<variable name="X(4,(6,5,4))" index="47" value="0"/>
<variable name="X(4,(6,4,3))" index="48" value="0"/>
<variable name="X(4,(6,4,5))" index="49" value="0"/>
<variable name="X(4,(5,2,4))" index="50" value="0"/>
<variable name="X(4,(5,3,3))" index="51" value="0"/>
<variable name="X(4,(5,3,5))" index="52" value="0"/>
<variable name="X(4,(5,6,4))" index="53" value="0"/>
<variable name="X(4,(5,5,3))" index="54" value="0"/>
```

```
<variable name="X(4,(5,5,5))" index="55" value="0"/>
<variable name="X(4,(5,4,2))" index="56" value="0"/>
<variable name="X(4,(5,4,6))" index="57" value="0"/>
<variable name="X(4,(4,1,4))" index="58" value="0"/>
<variable name="X(4,(4,2,3))" index="59" value="0"/>
<variable name="X(4,(4,2,5))" index="60" value="0"/>
<variable name="X(4,(4,3,2))" index="61" value="0"/>
<variable name="X(4,(4,3,6))" index="62" value="0"/>
<variable name="X(4,(4,7,4))" index="63" value="0"/>
<variable name="X(4,(4,6,3))" index="64" value="0"/>
<variable name="X(4,(4,6,5))" index="65" value="0"/>
<variable name="X(4,(4,5,2))" index="66" value="0"/>
<variable name="X(4,(4,5,6))" index="67" value="0"/>
<variable name="X(4,(4,4,1))" index="68" value="0"/>
<variable name="X(4,(4,4,7))" index="69" value="0"/>
<variable name="X(5,(0,4,4))" index="70" value="0"/>
<variable name="X(5,(2,4,4))" index="71" value="0"/>
<variable name="X(5,(1,3,4))" index="72" value="0"/>
<variable name="X(5,(1,5,4))" index="73" value="0"/>
<variable name="X(5,(1,4,3))" index="74" value="0"/>
<variable name="X(5,(1,4,5))" index="75" value="0"/>
<variable name="X(5,(4,4,4))" index="76" value="0"/>
<variable name="X(5,(3,3,4))" index="77" value="0"/>
<variable name="X(5,(3,5,4))" index="78" value="0"/>
<variable name="X(5,(3,4,3))" index="79" value="0"/>
<variable name="X(5,(3,4,5))" index="80" value="0"/>
<variable name="X(5,(2,2,4))" index="81" value="0"/>
<variable name="X(5,(2,3,3))" index="82" value="0"/>
<variable name="X(5,(2,3,5))" index="83" value="0"/>
<variable name="X(5,(2,6,4))" index="84" value="0"/>
<variable name="X(5,(2,5,3))" index="85" value="0"/>
<variable name="X(5,(2,5,5))" index="86" value="0"/>
<variable name="X(5,(2,4,2))" index="87" value="0"/>
<variable name="X(5,(2,4,6))" index="88" value="0"/>
<variable name="X(5,(6,4,4))" index="89" value="0"/>
<variable name="X(5,(5,3,4))" index="90" value="1"/>
```

```
<variable name="X(5,(5,5,4))" index="91" value="0"/>
<variable name="X(5,(5,4,3))" index="92" value="0"/>
<variable name="X(5,(5,4,5))" index="93" value="0"/>
<variable name="X(5,(4,2,4))" index="94" value="0"/>
<variable name="X(5,(4,3,3))" index="95" value="0"/>
<variable name="X(5,(4,3,5))" index="96" value="0"/>
<variable name="X(5,(4,6,4))" index="97" value="0"/>
<variable name="X(5,(4,5,3))" index="98" value="0"/>
<variable name="X(5,(4,5,5))" index="99" value="0"/>
<variable name="X(5,(4,4,2))" index="100" value="0"/>
<variable name="X(5,(4,4,6))" index="101" value="0"/>
<variable name="X(5,(3,1,4))" index="102" value="0"/>
<variable name="X(5,(3,2,3))" index="103" value="0"/>
<variable name="X(5,(3,2,5))" index="104" value="0"/>
<variable name="X(5,(3,3,2))" index="105" value="0"/>
<variable name="X(5,(3,3,6))" index="106" value="0"/>
<variable name="X(5,(3,7,4))" index="107" value="0"/>
<variable name="X(5,(3,6,3))" index="108" value="0"/>
<variable name="X(5,(3,6,5))" index="109" value="0"/>
<variable name="X(5,(3,5,2))" index="110" value="0"/>
<variable name="X(5,(3,5,6))" index="111" value="0"/>
<variable name="X(5,(3,4,1))" index="112" value="0"/>
<variable name="X(5,(3,4,7))" index="113" value="0"/>
<variable name="X(5,(8,4,4))" index="114" value="0"/>
<variable name="X(5,(7,3,4))" index="115" value="0"/>
<variable name="X(5,(7,5,4))" index="116" value="0"/>
<variable name="X(5,(7,4,3))" index="117" value="0"/>
<variable name="X(5,(7,4,5))" index="118" value="0"/>
<variable name="X(5,(6,2,4))" index="119" value="0"/>
<variable name="X(5,(6,3,3))" index="120" value="0"/>
<variable name="X(5,(6,3,5))" index="121" value="0"/>
<variable name="X(5,(6,6,4))" index="122" value="0"/>
<variable name="X(5,(6,5,3))" index="123" value="0"/>
<variable name="X(5,(6,5,5))" index="124" value="0"/>
<variable name="X(5,(6,4,2))" index="125" value="0"/>
<variable name="X(5,(6,4,6))" index="126" value="0"/>
```

```
  <variable name="X(5,(5,1,4))" index="127" value="0"/>
  <variable name="X(5,(5,2,3))" index="128" value="0"/>
  <variable name="X(5,(5,2,5))" index="129" value="0"/>
  <variable name="X(5,(5,3,2))" index="130" value="0"/>
  <variable name="X(5,(5,3,6))" index="131" value="0"/>
  <variable name="X(5,(5,7,4))" index="132" value="0"/>
  <variable name="X(5,(5,6,3))" index="133" value="0"/>
  <variable name="X(5,(5,6,5))" index="134" value="0"/>
  <variable name="X(5,(5,5,2))" index="135" value="0"/>
  <variable name="X(5,(5,5,6))" index="136" value="0"/>
  <variable name="X(5,(5,4,1))" index="137" value="0"/>
  <variable name="X(5,(5,4,7))" index="138" value="0"/>
  <variable name="X(5,(4,0,4))" index="139" value="0"/>
  <variable name="X(5,(4,1,3))" index="140" value="0"/>
  <variable name="X(5,(4,1,5))" index="141" value="0"/>
  <variable name="X(5,(4,2,2))" index="142" value="0"/>
  <variable name="X(5,(4,2,6))" index="143" value="0"/>
  <variable name="X(5,(4,3,1))" index="144" value="0"/>
  <variable name="X(5,(4,3,7))" index="145" value="0"/>
  <variable name="X(5,(4,8,4))" index="146" value="0"/>
  <variable name="X(5,(4,7,3))" index="147" value="0"/>
  <variable name="X(5,(4,7,5))" index="148" value="0"/>
  <variable name="X(5,(4,6,2))" index="149" value="0"/>
  <variable name="X(5,(4,6,6))" index="150" value="0"/>
  <variable name="X(5,(4,5,1))" index="151" value="0"/>
  <variable name="X(5,(4,5,7))" index="152" value="0"/>
  <variable name="X(5,(4,4,0))" index="153" value="0"/>
  <variable name="X(5,(4,4,8))" index="154" value="0"/>
 </variables>
</CPLEXSolution>
```

Ευρετήριο

Γλωσσάρι

2D lattice 2 Dimensional lattice, 14
3D lattice 3 Dimensional lattice, 14
CRMS Coordinates Root Mean Squared deviation, 18
DRMS Distance Root Mean Squared deviation, 19
HP Hydrophobic-Polar, 21
PCLF Protein Chain Lattice Fitting Problem, 8
PDB Protein Data Bank, 27, 33

Βιβλιογραφία

[1] B. Alberts, D. Bray, K. Hopkin, A. Johnson, J. Lewis, M. Raff, K. Roberts, P. Walter. Essential cell Biology. N. Ανάγνου, Π. Παπαζαφείρη, Ι. Παπαματθαιάκης, Κ. Σταματόπουλος. Βασικές Αρχές Κυτταρικής Βιολογίας. Ιατρικές Εκδόσεις Π.Χ. Πασχαλίδης Ε.Π.Ε Τόμος 1. Δεύτερη Έκδοση. (2006).

[2] D.G. Covell , R.L. Jernigan. Conformations of folded proteins in restrected spaces. Biochemistry (1990);29(13):3287-3294.

[3] T. Dallas. Algorithms and Experiments for the Protein Chain Lattice Fitting Problem. Master Thesis. University of Lethbridge (2004).

[4] K.A. Dill. Theory for the folding and stability of globular proteins. Biochemistry (1985);24(6):1501-1509.

[5] T.S. Ferguson, Linear Programming (A Concise Introduction).

[6] A. Godzik, J. Skolnick, A. Kolinski. Regularities in interaction pattern of globular proteins. Protein Eng. (1993);6(8):801-810.

[7] A. Gupta, J. Manuch, L. Stacho. Inverse protein folding in 2D HP model, Proc. of IEEE Computational Systems Bioinformatics Conference (2004);311-318.

[8] X. Huang. Fitting Protein Chains to Lattice using integer programming approach. Master Thesis. Simon Fraser University (2007).

[9] Κ. Καστρίτσης, Β. Δημητριάδης , Α. Σιβροπούλου, Εισαγωγή στην Βιολογία, Εκδοτικός Οίκος Αδελφών Κυριακίδη Α.Ε. (2005)

[10] P. Koehl, M. Delarue. Building protein lattice models using self-consistent mean field theory. The Journal of Chemical Physics (1998);108(22):9540-9549.

[11] J. Manuch and D. Gaur. Fitting protein chains to cubic lattice is NP-complete. J. Bioinform. Comput. Biol., (2008);6:93-106.

[12] Nimro d Megiddo. Linear Programming (For the Encyclopedia of Microcomputers). (1991)

[13] BH. Park, M. Levitt. The complexity and accuracy of discrete state models of protein structure. J. Mol. Biol (1995);249(2):493-507.

[14] M. Sipser. Introduction to Theory of Computation. Χρήστος Καπουτσής, Γεώργιος Φρ. Γεωργακόπουλος, Ιωάννης Παπαδόγγονας. Εισαγωγή στην Θεωρία Υπολογισμού. Πανεπιστημιακές Εκδόσεις Κρήτης (2007).

[15] S.S. Skiena. The Algorithm Design Manual. Second Edition

[16] Susanna S. Epp. Βασίλης Μεταφτσής, Αντώνης Τσολαμύτης. Διακριτά Μαθηματικά με Εφαρμογές. Κλειδάριθμος 3^h Έκδοση (2004).

[17] ILOG CPLEX User Manual. Retrieved from eaton.math.rpi.edu/cplex90html/pdf/usrcplex.pdf

[18] IBM SOFTWARE. Retrieved from http://www-01.ibm.com/software/