Τεχνητή Νοημοσύνη
Εργασία 1

Γεώργιος Σύρος ΑΜ 3190193 – Αναστάσιος Τουμαζάτος 3190198 - Ευγένιος Γκρίτσης 3190045

Οδηγίες για την μεταγλώττιση και εκτέλεση του κώδικα βρίσκονται στο ξεχωριστό αρχείο README.txt.

Άσκηση 1: **Το πρόβλημα των κανιβάλων και ιεραποστόλων**

Περιεχόμενα Κώδικα

- MisCan.java – το κύριο πρόγραμμα με το οποίο αλληλεπιδρά ο χρήστης.
- State.java – υλοποίηση μια κατάστασης του προβλήματος.
- StateComparator.java – υλοποίηση μιας κλάσης “συγκριτή” (java comparator) για την ευριστική συνάρτηση του αλγορίθμου A*.
- SpaceSearcher.java – υλοποίηση του αλγορίθμου A* με κλειστό σύνολο για την εύρεση της βέλτιστης λύσης του προβλήματος.
- Bank.java – κλάση enumerator που χρησιμεύει για την απεικόνιση της θέσης της βάρκας στις όχθες του ποταμού.

Χρήση

Ο χρήστης εκτελεί το κύριο πρόγραμμα (*MisCan*) περνώντας τις απαραίτητες παραμέτρους που αφορούν το πλήθος των κανιβάλων και τον ιεραποστόλων, την χωρητικότητα της βάρκας και το μέγιστο δυνατό αριθμό διασχίσεων του ποταμού από της βάρκας. Το πρόγραμμα επιστρέφει την βέλτιστη λύση μαζί με τα βήματα που χρειάστηκαν. Αν δεν υπάρχει λύση ή δεν βρει λύση με αριθμό διασχίσεων μικρότερο ή ίσο από τον μέγιστο αριθμό διασχίσεων που έχει εισάγει ο χρήστης, τότε το πρόγραμμα εμφανίζει σχετικό μήνυμα στον χρήστη.

Αρχιτεκτονική

Βασικός παράγοντας του προγράμματος είναι η υλοποίηση-απεικόνιση ενός στιγμιοτύπου του προβήματος. Αυτό επιτυγχάνεται μέσω της κλάσης State. Η κλάση αυτή περιέχει όλες τις απαραίτητες πληροφορίες και μεθόδους που χρειάζονται (1) για την αποθήκευση πληροφορίας σχετικά με την κατάσταση του προβλήματος (υπολογισμός σκορ ευριστικής συνάρτησης, αποθήκευση κατάστασης-πατέρα από την οποία προέκυψε, υπολογισμός μοναδικού αναγνωριστικού για ίδιες καταστάσεις με σκοπό τον προσδιορισμό τους στο κλειστό σύνολο, κτλ)

και (2) για την παραγωγή των δυνατών καταστάσεων-απογόνων (μετακίνηση της βάρκας αριστερά και δεξιά).

Πυρήνας του προγράμματος αποτελεί η εξερεύνηση του χώρου καταστάσεων, που γίνεται μέσω της κλάσης SpaceSearcher. Σε αυτή την κλάση υλοποιείται ο αλγόριθμος A* με κλειστό σύνολο. Για την βέλτιστη λειτουργία του χρησιμοποιείται η παρακάτω απλή ευρετική συνάρτηση h:

$$h(s) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n > 0,\ \text{Bank} = \text{LEFT} \\ 2 & \text{if } n > 0,\ \text{Bank} = \text{RIGHT} \end{cases}$$

** s μια κατάσταση του προβλήματος, n ο αριθμός των κανιβάλων και ιεραπόστολων στην αριστερή όχθη*

Η συγκεκριμένη ευρετική παρουσιάστηκε στην άσκηση 4.3 του φροντιστηρίου του μαθήματος και προκύπτει από την διαδικασία αφαίρεσης περιορισμών από το αρχικό πρόβλημα. Αναλυτικότερα, έχει αφαιρεθεί ο περιορισμός (1) να μην υπερβαίνουν οι κανίβαλοι τους ιεραπόστολους στην βάρκα και σε οποιαδήποτε όχθη του ποταμού και (2) η βάρκα να μην έχει συγκεκριμένη χωρητικότητα και έτσι να χωράνε όσα άτομα θέλουμε.

Το σκεπτικό είναι ότι αν δεν υπάρχουν άτομα στην αριστερή όχθη (n = 0), τότε η ευρετική συνάρτηση επιστρέφει 0 και βρισκόμαστε σε αποδεκτή τελική κατάσταση. Αν υπάρχουν άτομα στην αριστερή όχθη (n > 0) και η βάρκα βρίσκεται στα αριστερά, τότε ανεβαίνουν όλοι στην βάρκα και περνάνε στην απέναντι όχθη με 1 μόνο ταξίδι. Τέλος, αν η βάρκα βρίσκεται στην δεξιά όχθη, πρέπει να επιστρέψει στην αριστερή όχθη με τουλάχιστον ένα άτομο πάνω ως βαρκάρη, να ανέβουν όλοι στην βάρκα και τελικά να περάσουν στην απέναντι όχθη με 2 συνολικά ταξίδια.

Το κόστος κάθε κίνησης είναι θετικό και ο μέγιστος παράγοντας διακλάδωσης (συνάρτηση της χωρητικότητας της βάρκας *M*) , δηλαδή ο αριθμός δυνατών κινήσεων σε κάθε κατάσταση, είναι πεπερασμένος. Στην περίπτωση αυτή γνωρίζουμε ότι ο A* είναι πλήρης, δηλαδή αν υπάρχει λύση, σίγουρα τη βρίσκει. Γνωρίζουμε ότι το πρόβλημα έχει λύση για συγκεκριμένους συνδυασμούς των *N*, *M*.

Η παραπάνω ευρετική συνάρτηση που χρησιμοποιείται είναι **αποδεκτή**, διότι προκύπτει έπειτα από εφαρμογή αφαίρεσης περιορισμών του προβλήματος.

<u>Σε αναζήτηση για καλύτερη ευρετική</u>
Προσπαθώντας να γενικεύσουμε την πιο σύνθετη ευρετική που παρουσιάστηκε από την άσκηση 4.3 στο φροντιστήριο του μαθήματος, τροποποιήσαμε την συνάρτηση για να ισχύει για βάρκα χωρητικότητας M ατόμων:

$$h(s) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n \leq M,\ \text{Bank} = \text{LEFT} \\ 2 & \text{if } n \leq M - 1,\ \text{Bank} = \text{RIGHT} \\ 2 \cdot \lceil \frac{n}{M-1} \rceil & \text{if other, Bank} = \text{RIGHT} \\ 2 \cdot \lceil \frac{n-1}{M-1} \rceil - 1 & \text{if other, Bank} = \text{LEFT} \end{cases}$$

** s μια κατάσταση του προβλήματος, n ο αριθμός των κανιβάλων και ιεραπόστολων στην αριστερή όχθη*

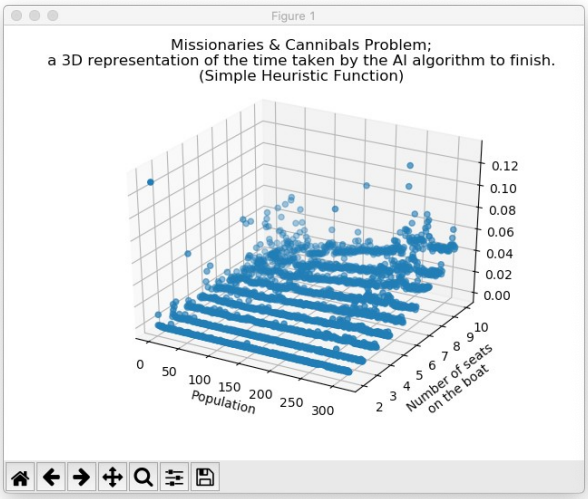
Η συνάρτηση αυτή προέκυψε από αφαίρεση του περιορισμού οι κανίβαλοι να μην υπερβαίνουν τους ιεραπόστολους σε οποιαδήποτε όχθη και πάνω στην βάρκα. Έχοντας απλοποιήσει λίγο το πρόβλημα, το σκεπτικό έχει ως εξής:

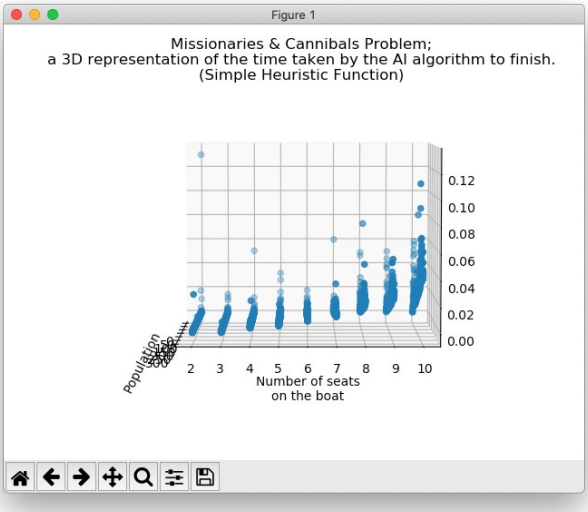
- Αν δεν υπάρχουν άτομα στην αριστερή όχθη, τότε η ευρετική επιστρέφει 0 κατά σύμβαση (δεν χρειάζεται καμία διάσχιση).
- Αν η βάρκα βρίσκεται στην αριστερή όχθη και τα άτομα στην αριστερή όχθη είναι λιγότερα ή ίσα με την χωρητικότητα της βάρκας, τότε χρειάζεται 1 διάσχιση και η ευρετική επιστρέφει 1 (όλα τα άτομα χωράνε στην βάρκα και περνάνε απλά απέναντι).
- Αν η βάρκα βρίσκεται στην δεξιά όχθη και τα άτομα στην αριστερή όχθη είναι λιγότερα ή ίσα με την χωρητικότητα της βάρκας πλην την μια θέση που καταλαμβάνει ο βαρκάρης, τότε χρειάζονται 2 διασχίσεις και η ευρετική επιστρέφει 2 (κάποιο άτομο από την δεξιά πλευρά περνάει στην αριστερή όχθη ως βαρκάρης και παίρνει τα υπόλοιπα άτομα).
- Αν η βάρκα βρίσκεται στην δεξιά όχθη και υπάρχουν παραπάνω άτομα από την χωρητικότητα της βάρκας πλην την θέση που καταλαμβάνει ο βαρκάρης, τότε περνάει ο βαρκάρης απέναντι, παίρνει Μ-1 άτομα, τα φέρνει στην δεξιά πλευρά, κ.ο.κ. Επομένως κάθε ταξίδι του βαρκάρη αποτελείται από 2 διασχίσεις και κάθε φορά το πλήθος των ατόμων στην αριστερή πλευρά μειώνεται κατά Μ-1. Άρα χρειάζονται τελικά 2*ceil(n / (M-1)) διασχίσεις.
- Αν η βάρκα βρίσκεται στην αριστερή όχθη και υπάρχουν παραπάνω άτομα από την χωρητικότητα της βάρκας, κάνουμε αναγωγή στο πρόβλημα της προηγούμενης περίπτωσης, που η βάρκα είναι στην δεξιά όχθη. Ουσιαστικά, είναι σαν να υπάρχουν n-1 άτομα στην αριστερή όχθη με τον βαρκάρη να έχει έρθει από την δεξιά όχθη για να μεταφέρει τους υπόλοιπους. Άρα, έχουμε 2*ceil((n-1)/(M-1)) – 1 διασχίσεις (έχουμε -1 διάσχιση γιατί ο βαρκάρης κάνει «μισό» ταξίδι για να πάει τα άτομα την πρώτη φορά απέναντι).

<u>Δυνατότητες και Χρονική πολυπλοκότητα</u>

Το πρόγραμμα είναι ικανό να βρίσκει την βέλτιστη δυνατή λύση του προβλήματος, εφ'όσον αυτή υπάρχει (με βάρκα χωρητικότητας 2 δεν υπάρχει λύση για πληθυσμούς κανιβάλων και ιεραπόστολων μεγαλύτερους του 3, κτλ).

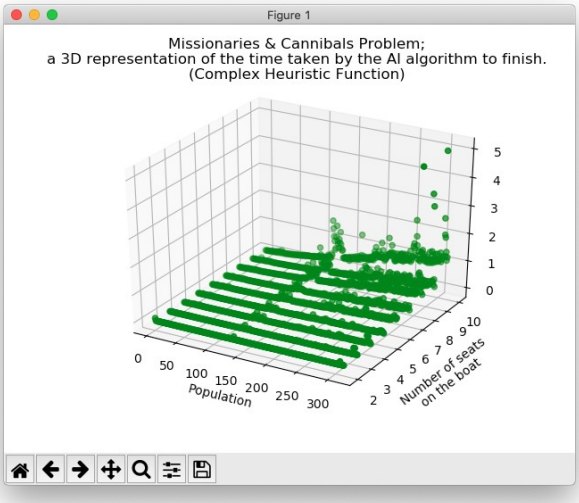
Έπειτα από την υποβολή του προγράμματος σε δοκιμαστικές εκτελέσεις με διαφορετικές παραμέτρους χρησιμοποιώντας την απλή ευρετική συνάρτηση που αναλύσαμε παραπάνω, προέκυψαν τα εξής αποτελέσματα:

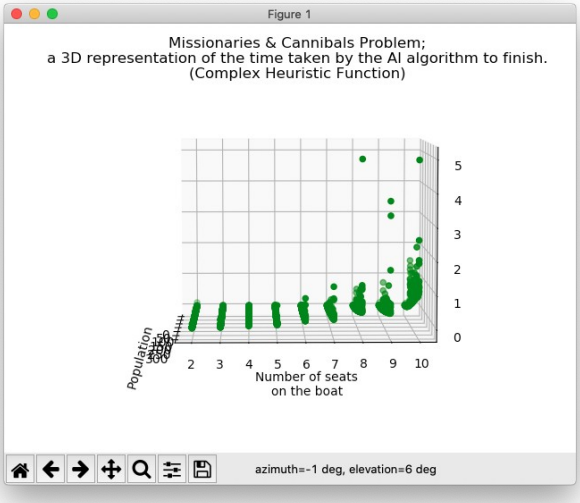




Παρατηρεί κανείς ότι όσο μεγαλύτερη είναι η χωρητικότητα της βάρκας M (ισοδύναμα: ο παράγοντας διακλάδωσης του δένδρου καταστάσεων που αναζητεί ο αλγόριθμος A*), η πολυπλοκότητα του αλγορίθμου είναι εκθετική, κάτι που είναι αναμενόμενο.

Θέλοντας να συγκρίνουμε τις δύο ευρετικές συναρτήσεις, υποβάλαμε το πρόγραμμα σε νέες δοκιμαστικές εκτελέσεις με τη διαφορά ότι χρησιμοποιούσε την σύνθετη ευρετική που παρουσιάσαμε. Προέκυψαν τα εξής αποτελέσματα:





Είναι προφανές ότι η σύνθετη ευρετική είναι πιο καλή από την απλή αφού simple_heuristic(s) ≤ complex_heuristic(s), για κάθε κατάσταση s. Δηλαδή η σύνθετη ευρετική προβλέπει με μεγαλύτερη ακρίβεια το ακριβές βέλτιστο κόστος από την s ως την τελική κατάσταση). **Γιατί όμως ο αλγόριθμος είναι πιο αργός σύμφωνα με τις μετρήσεις για τις ίδιες παραμέτρους;**

Το παράδοξο των δύο ευρετικών

Είναι γεγονός ότι, θεωρητικά, η σύνθετη ευρετική συνάρτηση είναι καλύτερη από την απλή. Ωστόσο, οι μικροί πληθυσμοί (N ≤ 313) που δοκιμάστηκε το πρόγραμμα “ξεγελούν” τον αναλυτή να νομίζει ότι η απλή ευρετική είναι καλύτερη από την σύνθετη, διότι η σύθνετη έχει μεγαλύτερο υπολογιστικό κόστος (υπολογισμός του μαθηματικού floor(), περισσότερα floating point operations), ενώ η απλή είναι απλά ένας βρόχος if για τον προσδιορισμό της θέσης της βάρκας και επιστρέφει τρεις σταθερές (0,1,2). Επιπλέον, προστίθεται παραπάνω κόστος κατά την ταξινόμηση των καταστάσεων στο μέτωπο της αναζήτησης ώστε ο

Ως αποτέλεσμα των παραπάνω μετρήσεων, το πρόγραμμα αποφασίστηκε από τα μέλη της ομάδας να περιέχει και τις δύο ευρετικές, δίνοντας την ικανότητα στον χρήστη να επιλέξει ποιά ευρετική θα χρησιμοποιήσει ο αλγόριθμος.

Ενδεικτικοί Χρόνοι

Απλή Ευρετική

Πλήθος N	**Χωρητικότητα M**	**Χρόνος Εκτέλεσης (seconds)**
14	4	0.001
48	10	0.024
81	7	0.003
151	10	0.021
285	10	0.043
313	7	0.015

Σύνθετη Ευρετική

Πλήθος N	**Χωρητικότητα M**	**Χρόνος Εκτέλεσης (seconds)**
14	4	0.002
48	10	0.058
81	7	0.034
151	10	0.395
285	10	0.869
313	7	0.257

Άσκηση 2: **Το πρόβλημα των 8 βασιλισσών (γενίκευση για NxN σκακιέρα).**

Περιεχόμενα Κώδικα

- Chromosome.java – υλοποίηση του χρωμοσώματος για την αναπαράσταση μιας κατάστασης του αλγορίθμου
- GeneticAlgorithm.java – υλοποίηση γενετικού αλγόριθμου για την επίλυση του προβλήματος των βασιλισσών
- Main.java – μαιν

Ανάλυση Κώδικα

Για το συγκεκριμένο πρόβλημα πρέπει να ξεκινήσουμε αναλύοντας την υλοποίηση του χρωμοσώματος ως οντότητα, το οποίο θα χρησιμοποιείται για την αναπαράσταση μιας τρέχουσας εκδοχής του προβλήματος. Αυτό γίνεται μέσα από την κλάση Chromosome, η οποία στηρίζεται στις παρακάτω βασικές μεθόδους:

Chromosome: Ο Constructor της κλάσης, ο οποίος δέχεται ως είσοδο τον αριθμό **N** των βασιλισσών. Οι βασίλισσες αυτές κατανέμονται τυχαία στη σκακιέρα, με την προϋπόθεση πως κάθε στήλη θα περιέχει μία ακριβώς βασίλισσα. Η θέση της κάθε βασίλισσας στη σκακιέρα μεγέθους NxN του δεδομένου χρωμοσώματος αναπαρίσταται μέσω ενός μονοδιάστατου πίνακα **genes**, στον οποίο κάθε θέση k περιέχει τον αριθμό της γραμμής στην οποία βρίσκεται η βασίλισσα της στήλης k. Ο δείκτης **fitness** αναπαριστά το βαθμό καταλληλότητας του συγκεκριμένου χρωμοσώματος, ο οποίος υπολογίζεται μέσω της ευρετικής συνάρτησης καταλληλότητας. Το fitness score είναι ο μέγιστος

αριθμός ζευγαριών βασιλισσών που δεν απειλούνται μεταξύ τους (π.χ Για n=8 το fitness score θα είναι (n-1)+(n-2)+(n-3)+…+(n-n)=28 .Υπάρχει και ένας αντίστοιχος copy constructor, ο οποίος χρησιμοποιείται στην παραγωγή νέων χρωμοσωμάτων.
Mutate: Η μέθοδος αυτή υλοποιεί τη ‘’μετάλλαξη’’ ενός χρωμοσώματος, αλλάζοντας τη θέση μιας βασίλισσας που βρίσκεται σε μια τυχαία στήλη, και τοποθετώντας τη σε μια νέα τυχαία γραμμή στην ίδια στήλη. Προφανώς υπάρχει μια πιθανότητα 1/n πως το νέο χρωμόσωμα που προέκυψε από τη μετάλλαξη θα είναι το ίδιο με το παλιό.
CalculateFitness: Μέθοδος υπολογισμού της καταλληλότητας ενός χρωμοσώματος. Ξεκινώντας με καταλληλότητα 0, η μέθοδος διατρέχει το μήκος του πίνακα genes, ελέγχοντας για κάθε ζεύγος βασιλισσών (1) ότι δεν είναι στην ίδια γραμμή, και στη συνέχεια (2), ότι δεν είναι στην ίδια διαγώνιο. Για κάθε ζεύγος στο οποίο ισχύουν τα δύο παραπάνω, προστίθεται ένας βαθμός στο fitness score του χρωμοσώματος.
(1) δύο βασίλισσες είναι στην ίδια γραμμή μόνο αν στον πίνακα genes υπάρχουν δύο θέσεις με την ίδια τιμή (θυμίζουμε ότι η θέση στον πίνακα αντιπροσωπεύει τη στήλη και η τιμή σε αυτή τη θέση αναπαριστά τη γραμμή). Δεν χρειάζεται να ελέγξουμε αν είναι και στην ίδια στήλη, καθώς αυτό είναι εξασφαλισμένο από τον ορισμό του προβλήματος ότι δεν θα συμβεί.
(2) για να βρίσκονται δύο βασίλισσες στην ίδια διαγώνιο, πρέπει η απόλυτη τιμή της διαφοράς των στηλών τους να είναι ίση με την απόλυτη τιμή της διαφοράς των γραμμών τους. Αυτή η συνθήκη ισχύει για οποιαδήποτε διαγώνιο μπορούν να σχηματίζουν δύο βασίλισσες μεταξύ τους.

Στην κλάση geneticAlgorithm υλοποιούνται οι παρακάτω μεθόδους:

- createPopulation: Φτιάχνει και επιστρέφει μια λίστα από χρωμοσώματα που αντιπροσωπεύουν εκδοχές του προβλήματος με NxN σκακιέρα.
- Reproduce: Δέχεται δύο χρωμοσώματα και επιστρέφει δύο απογόνους τους. Οι απόγονοι είναι νέα χρωμοσώματα τα οποία έχουν σχηματιστεί αντιγράφοντας ένα τυχαίο τμήμα από τον ένα γονέα και το υπόλοιπο από τον άλλο.
- updateOccurrenses: Ενημερώνει την λίστα με τα ενεργά χρωμοσώματα. Η λίστα occurrences περιέχει το index κάθε χρωμοσώματος της λίστας population ίσες φορές με το fitness score του. Για παράδειγμα ο δείκτης ενός χρωμοσώματος με αριθμό καταλληλότητας 7, θα εμφανιστεί 7 φορές στην λίστα occurrences. Με αυτόν τον τρόπο όταν επιλέγουμε δυο τυχαία χρωμοσώματα για αναπαραγωγή (reproduce) έχουμε μεγαλύτερη πιθανότητα αυτά να έχουν υψηλότερο βαθμό καταλληλότητας και άρα να παράξουν καλύτερους απογόνους.
- run: Δημιουργεί την επόμενη γενιά χρωμοσωμάτων επιλέγοντας κάθε φορά δύο διαφορετικά χρωμοσώματα του τωρινού πληθυσμού, αναπαράγοντας τα, μεταλλάσσοντάς τους απογόνους, και τοποθετώντας τους σε μια νέα λίστα. Η λίστα που προκύπτει πρέπει να έχει το ίδιο μέγεθος με τον τωρινό πληθυσμό. Στην συνέχεια, αυτή ταξινομείται βάσει του βαθμού καταλληλότητας των χρωμοσωμάτων της και εάν βρεθεί κάποιο χρωμόσωμα το οποίο πληροί τα κριτήρια καταλληλότητας του προβλήματος, τότε επιστρέφεται ως λύση. Τέλος, η λίστα με τους απογόνους αντικαθιστά τον παλαιό πληθυσμό και η λίστα με τα indexes του ενημερώνεται.

Χρήση

Ο χρήστης εκτελεί το κύριο πρόγραμμα (Main.java) περνώντας τις απαραίτητες παραμέτρους που αφορούν το μέγεθος της σκακιέρας (αριθμός βασιλισσών), το μέγεθος του πληθυσμού χρωμοσωμάτων, την πιθανότητα μετάλλαξης, τον μέγιστο αριθμό iterations και το ελάχιστο βαθμό καταλληλότητας. Το πρόγραμμα επιστρέφει το βέλτιστο χρωμόσωμα, τον βαθμό καταλληλότητας του (fitness), τον αριθμό των iterations που

χρειάστηκαν και τέλος εκτυπώνεται και η σκακιέρα με τις βέλτιστες τοποθετήσεις των βασιλισσών.

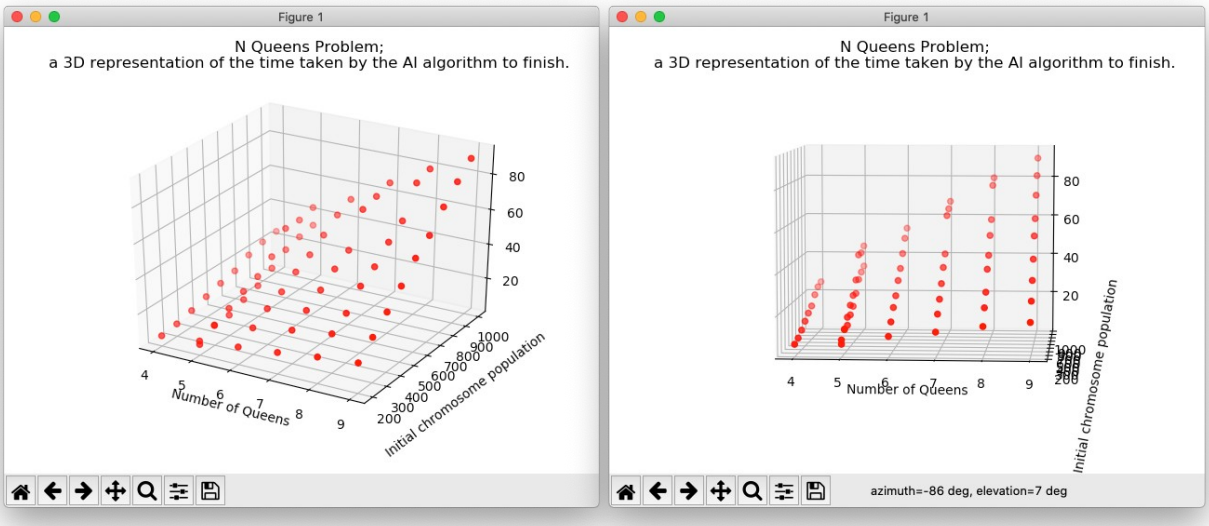
Αρχιτεκτονική

Η μέθοδος επίλυσης του προβλήματος των Ν βασιλισσών αξιοποιεί την τεχνική των γενετικών αλγορίθμων. Βασικό στοιχείο του αλγορίθμου είναι η υλοποίηση της έννοιας του χρωμοσώματος για την αναπαράσταση στιγμιότυπων της σκακιέρας. Αυτό επιτυγχάνεται μέσω της κλάσης Chromosome, η οποία περιέχει όλες τις απαραίτητες πληροφορίες για το κάθε χρωμόσωμα όπως την αναπαράσταση της σκακιέρας, καθώς και συναρτήσεις για τον υπολογισμό της καταλληλότητας και για την μετάλλαξη του χρωμοσώματος.

Η υλοποίηση των λειτουργιών του γενετικού αλγορίθμου γίνονται μέσω της κλάσης GeneticAlgorithm. Η κλάση αυτή διατηρεί λίστες που αναπαριστούν πληθυσμούς χρωμοσωμάτων και περιέχει μεθόδους για την αναπαραγωγή χρωμοσωμάτων και δημιουργία νέου πληθυσμού, ευνοώντας τα χρωμοσώματα με μεγαλύτερο βαθμό καταλληλότητας. Κάθε φορά που δημιουργείται ένας νέος πληθυσμός ελέγχει εάν υπάρχει κάποιο χρωμόσωμα που αναπαριστά κατάσταση που λύνει το πρόβλημα που έχουμε θέσει, και εάν βρεθεί το επιστρέφει. Σε αντίθετη περίπτωση, προχωρά σε ανανέωση του πληθυσμού μέσω της αναπαραγωγής χρωμοσωμάτων από τον τωρινό πληθυσμό και αυτή η διαδικασία επαναλαμβάνεται σύμφωνα με τα όρια που του έχουμε θέσει.

Δυνατότητες και Χρονική πολυπλοκότητα

Παρατηρεί κανείς ότι ο χρόνος εκτέλεσης (σε δευτερόλεπτα) αυξάνεται καθώς μεγαλώνει ο πληθυσμός των αρχικών χρωμοσωμάτων και όταν μεγαλώνει ο αριθμός βασιλισσών Ν που πρέπει να τοποθετήσουμε στην σκακιέρα.



Ενδεικτικοί Χρόνοι

Αριθμός Βασιλισσών (N)	**Αρχικός Πληθυσμός**	**Χρόνος Εκτέλεσης (seconds)**
4	300	7.4744
5	700	23.5155
6	200	10.4077
7	1000	67.2959
8	500	38.8749

* όλοι οι χρόνοι μετρήθηκαν έπειτα από εκτέλεση του προγράμματος σε περιβάλλον macOS 10.15.7, MacBook Pro (13-inch, Mid 2012), Processor; 2,5 GHz Dual-Core Intel Core i5, 16 GB RAM.