ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

Σχολή Μηχανικών Παραγωγής και Διοίκησης

ΑΛΓΟΡΙΘΜΟΣ ΑΝΑΖΗΤΗΣΗΣ ΚΟΥΚΟΥ ΓΙΑ ΤΟ ΠΡΟΒΛΗΜΑ ΤΩΝ Ν ΒΑΣΙΛΙΣΣΩΝ

Διπλωματική εργασία

Καράμπελα Κωνσταντίνα

Επιβλέπων: Δρ. Ιωάννης Μαρινάκης, Καθηγητής

Χανιά, Φεβρουάριος 2023

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να εκφράσω τις ευχαριστίες μου στον καθηγητή μου Ιωάννη Μαρινάκη για την πολύτιμη βοήθεια που μου παρείχε κατά την εκπόνηση της παρούσας διπλωματικής εργασίας, καθώς και για την εμπιστοσύνη που μου έδειξε.

Ακόμα, θα ήθελα να ευχαριστήσω του γονείς μου που με στηρίζουν πάντα και με όποιον τρόπο μπορούν ώστε να εκπληρώσω τα όνειρά μου, όπως και τους φίλους μου που είναι πάντα εκεί .

ΠΕΡΙΛΗΨΗ

Το πρόβλημα των Ν βασιλισσών αφορά την τοποθέτηση Ν βασιλισσών σε μία σκακιέρα NxN διαστάσεων, με τέτοιο τρόπο, ώστε καμία βασίλισσα να μην απειλείται από κάποια άλλη. Μία βασίλισσα απειλεί τις υπόλοιπες βασίλισσες που βρίσκονται στην ίδια γραμμή, στην ίδια στήλη ή στην ίδια διαγώνιο. Εξαιτίας του μεγάλου αριθμού συνδιασμών ακόμα και για μικρές τιμές του n, η λύση του προβλήματος είναι χρονοβόρα διαδικασία. Σκοπός της διπλωματικής εργασίας είναι η εφαρμογή του αλγορίθμου αναζήτησης κούκου για τη βέλτιστη και πιο γρήγορη τοποθέτηση των Ν βασιλισσών στην σκακιέρα. Ο συγκεριμένος αλγόριθμος είναι εμπνευσμένος από την παρασιτική συμπεριφορά που εμφανίζεται στη διαδικασία αναπαραγωγής ορισμένων ειδών κούκων τα οποία τοποθετούν τα αβγά τους σε φωλιές άλλων πτηνών και μετακινούν τα αβγά των άλλων πτηνών με στόχο να αυξήσουν την πιθανότητα επιβίωσης των δικών τους αβγών.

Λέξεις - κλειδιά: αλγόριθμος αναζήτησης κούκου, πρόβλημα Ν βασιλισσών,

Lévy flight, αλγόριθμος, τοπική αναζήτηση, 1-1 exchange

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΥΧΑΡΙΣΤΙΕΣ **2**

ΠΕΡΙΛΗΨΗ **3**

ΠΕΡΙΕΧΟΜΕΝΑ **4**

ΕΙΣΑΓΩΓΗ **6**

1.ΒΑΣΙΚΕΣ ΕΝΝΟΙΕΣ **7**

1.1 Επιχειρησιακη ερευνα 7
1.2 Συνδυαστικη Βελτιστοποιηση 8
1.3. Το προβλημα των N βασιλισσων 9
1.4. Αλγοριθμοι 12
1.4.1 Μεθευρετικοί αλγόριθμοι *13*
1.4.2 Εξελικτικοί Αλγόριθμοι *14*

2.ΑΛΓΟΡΙΘΜΟΙ **15**

2.1. Αλγοριθμος Αναζητησης του Κουκου 15
2.2 Levy flights 16
2.3.Αλγοριθμος Επαναληπτικης Τοπικης Αναζητησης 17
2.3.1 Γενική ανάλυση αλγορίθμου *17*
2.3.2 Μέθοδοι τοπικής αναζήτησης *21*

3.ΕΠΙΛΥΣΗ ΠΡΟΒΛΗΜΑΤΟΣ – ΕΦΑΡΜΟΓΗ ΑΛΓΟΡΙΘΜΩΝ **25**

3.1. Βασικη Συναρτηση 25
3.2 Αντικειμενικη συναρτηση και περιορισμοι 26
3.2.1 Αντικειμενική συνάρτηση *26*
3.2.2. Συγκρούσεις (penalties) *26*
3.3 Συναρτηση του Cuckoo 28
3.3.1 Μετατροπή διακριτών τιμών σε συνεχείς και αντίστροφα *30*
3.2.2. Υποσυναρτήσεις του Αλγόριθμου του Κούκου *32*
3.4. Συναρτησεις της Τοπικης Αναζητησης 35
3.5. Δημιουργια Σκακιερας 36

4. ΠΕΙΡΑΜΑΤΑ ΚΑΙ ΑΠΟΤΕΛΕΣΜΑΤΑ **37**

4.1. Πειραμα για N=4 38
4.2. Πειραμα για N=8 39
4.3 Πειραμα για N=10 40
4.4. Πειραμα για N=11 41

4.5. Πειραμα για N=15 42
4.6. Πειραμα για N=20 43
4.7.Πειραμα για N=25 45
4.8. Πειραμα για N=50 46
4.9. Πειραμα για N=100 και N=250 47
4.10. Πειραμα για συγκεκριμενο αριθμο βασιλισσων (N=15) και διαφορετικο πληθυσμο 49
5.ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΗ ΕΡΕΥΝΑ 51
ΠΗΓΕΣ 52

ΕΙΣΑΓΩΓΗ

Σκοπός της τρέχουσας διπλωματικής εργασίας είναι η εφαρμογή του αλγόριθμου αναζήτησης του Κούκου στο πρόβλημα των Ν βασιλισσών με στόχο τη βέλτιστη και πιο γρήγορη τοποθέτηση των βασιλισσών σε μία σκακιέρα διαστάσεων NxN. Το πρώτο κεφάλαιο αναφέρεται σε βασικές έννοιες, όπως, η επιχειρησιακή έρευνα, η βελτιστοποίηση, το πρόβλημα των N βασιλισσών, και οι μεθευρετικοί και εξελικτικοί αλγόριθμοι. Στη συνέχεια, στο δεύτερο κεφάλαιο της εργασίας παρουσιάζονται οι αλγόριθμοι οι οποιοι χρησιμοποιήθηκαν στην εκπόνηση της διπλωματικής και πιο συγκεκριμένα, ο αλγόριθμος αναζήτησης του κούκου, οι Lévy flights καθώς και ο αλγόριθμος τοπικής αναζήτησης. Στο τρίτο κεφάλαιο της εργασίας γίνεται ανάλυση καθώς και εφαρμογή των αλγορίθμων που ήδη αναφέρθηκαν, για την επίλυση του προβλήματος των Ν βασιλισσών. Τέλος, στο τέταρτο κεφάλαιο παρουσιάζονται τα αποτελέσματα των πειραμάτων.

1.ΒΑΣΙΚΕΣ ΕΝΝΟΙΕΣ

1.1 Επιχειρησιακή έρευνα

Η επιχειρησιακή έρευνα ασχολείται με την ανάπτυξη και εφαρμογή αναλυτικών μαθηματικών μοντέλων και τεχνικών για τη λήψη αποφάσεων σε θέματα σχεδιασμού και λειτουργίας Τεχνο-Οικονομικών και διοικητικών Συστημάτων. Απαιτεί γνώσεις από διάφορα αντικείμενα όπως τα μαθηματικά, τη μηχανολογία, τη διοίκηση, τον προγραμματισμό, και για αυτό το λόγο θεωρείται ως διεπιστημονικό πεδίο.

Η ανάπτυξη της επιχειρησιακής έρευνας ξεκίνησε κατά τον 2ο Παγκόσμιο πόλεμο ως εργαλείο στρατηγικού σχεδιασμού για το σχεδιασμό στρατηγικών επιχειρήσεων. Τα στρατιωτικά επιτελεία της Μεγάλης Βρετανίας και των ΗΠΑ οργάνωσαν ειδικές ομάδες αναλυτών από διάφορους επιστημονικούς κλάδους, όπως μηχανικούς, μαθηματικούς, φυσικούς, κ.α. Μετά το πέρας του πολέμου η επιχειρησιακή έρευνα αναπτύχθηκε σε θεωρητικό καθώς και σε πρακτικό επίπεδο. Μεθοδολογίες από το πεδίο της επιχειρησιακής έρευνας μπορούν να εφαρμοστούν στη σύγχρονη κοινωνία σε κάθε επιχείρηση ή οργανισμό. Οι κλάδοι στους οποίους είναι ευρέως διαδεδομένη είναι το εμπόριο, οι μεταφορές, η βιομηχανία, ο τομέας της υγείας, οι κατασκευές, κλπ. Επομένως <<αντικείμενο της επιχειρησιακής έρευνας είναι η υποστήριξη της διαδικασίας λήψης αποφάσεων σε θέματα ανάλυσης και σχεδιασμού τρόπων βελτίωσης των λειτουργικών διαδικασιών σε επιχειρήσεις και οργανισμούς. >> (Δούμπος, 2021)

Η ραγδαία ανάπτυξη των ηλεκτρονικών υπολογιστών τα τελευταία έτη, έχει συμβάλει στην τεράστια πρόοδο της επιχειρησιακής έρευνας σε πολύ μεγάλο βαθμό. Αυτό έχει ως αποτέλεσμα να επιλύονται προβλήματα μεγάλων διαστάσεων σε ελάχιστο χρόνο. (Μαρινάκης & Μυγδαλάς, Συνδυαστική Βελτιστοποίηση, 2016)

Μέσω ενός προβλήματος επιχειρησιακής έρευνας αναπτύσσεται ένα πρότυπο το οποίο παρομοιάζεται με μία φυσική κατάσταση. Δηλαδή ένα πρότυπο επιχειρησιακής έρευνας αναφέρεται σε μια εξιδανικευμένη αναπαράσταση ενός πραγματικού συστήματος. Ο πιο σημαντικός τύπος προτύπου επιχειρησιακής έρευνας είναι ο συμβολικός ή μαθηματικός τύπος. Στη μορφοποίηση αυτού του τύπου προβλημάτων υποθέτουμε ότι όλες οι συσχετιζόμενες μεταβλητές είναι ποσοτικές. Ετσι, για να περιγραφεί η συμπεριφορά του συστήματος, χρησιμοποιούνται μαθηματικά σύμβολα για την αναπαράσταση των μεταβλητών και εν συνεχεία συσχετίζονται με τις κατάλληλες μαθηματικές συναρτήσεις.

Ένα μαθηματικό πρότυπο περιλαμβάνει τρία βασικά σύνολα στοιχείων:

1. **Μεταβλητές απόφασης (decision variables)**: Ως μεταβλητές απόφασης ορίζονται τα οικονομικά ή φυσικά μεγέθη τα οποία προσδιορίζει ο λήπτης αποφάσεων και από τα οποία εξαρτάται το αποτέλεσμα. Οι παράμετροι αποτελούν τις μεταβλητές ελέγχου του συστήματος.

2. **Περιορισμοί (constraints)**: Για τη μέτρηση των φυσικών περιορισμών του συστήματος, το πρότυπο πρέπει να περιέχει περιορισμούς που να περιορίζουν τις μεταβλητές απόφασης στις εφικτές (επιτρεπτές) τιμές τους. Ένα πρόβλημα κατατάσσεται σε διαφορετικές κατηγορίες βάση της φύσης των περιορισμών του. (γραμμικοί, ακέραιοι, μη γραμμικοί, κλπ).

3. **Αντικειμενική συνάρτηση (objective function)**: Η αντικειμενική συνάρτηση καθορίζει το μέτρο της αποτελεσματικότητας του συστήματος σαν μαθηματική συνάρτηση των μεταβλητών απόφασής του. Για παράδειγμα, αν ο αντικειμενικός στόχος του συστήματος είναι η μεγιστοποίηση του συνολικού κέρδους, η αντικειμενική συνάρτηση πρέπει να καθορίζει το κέρδος σε όρους των μεταβλητών απόφασης. Γενικά, η βέλτιστη λύση ενός μοντέλου επιτυγχάνεται όταν οι αντίστοιχες τιμές των μεταβλητών απόφασης οδηγούν στη βέλτιστη τιμή της αντικειμενικής συνάρτησης, ικανοποιώντας ταυτόχρονα όλους τους περιορισμούς.

(Μαρινάκης & Μυγδαλάς, Συνδυαστική Βελτιστοποίηση, 2016)

1.2 Συνδυαστική Βελτιστοποίηση

Η συνδυαστική βελτιστοποίηση είναι ένα από τα πιο ενεργά πεδία στο χώρο της επιχειρησιακής έρευνας. Ο όρος συνδυαστική βελτιστοποίηση χρησιμοποιείται για να περιγράψει την περιοχή του μαθηματικού προγραμματισμού που ασχολείται με την επίλυση προβλημάτων βελτιστοποίησης που έχουν διακριτή ή συνδυαστική δομή. Προβλήματα συνδυαστικής βελτιστοποίησης εμφανίζονται σε ένα μεγάλο αριθμό από εφαρμογές, όπως στο σχεδιασμό τηλεπικοινωνιακών δικτύων, στα προβλήματα διανομής προϊόντων, στα προβλήματα μεταφορών, στα προβλήματα προγραμματισμού πληρωμάτων κ.λπ. Τα περισσότερα από αυτά τα προβλήματα λόγω της πολυπλοκότητας τους δεν μπορούν να αντιμετωπιστούν εύκολα και έτσι απαιτούν ειδικούς αλγόριθμους για την επίλυσή τους.

Ως πρόβλημα μεγιστοποίησης ή ελαχιστοποίησης μπορεί να δηλωθεί οποιοδήποτε πρόβλημα συνδυαστικής βελτιστοποίησης, ανάλογα με την εκάστοτε αντικειμενική συνάρτηση και την ελαχιστοποίηση ή την μεγιστοποίηση της.

Για κάθε πρόβλημα συνδυαστικής βελτιστοποίησης, διακρίνουμε δύο παραλλαγές:

Η παραλλαγή αναζήτησης: δεδομένου ενός προβλήματος, να βρεθεί μια λύση με ελάχιστη ή μέγιστη τιμή αντικειμενικς συνάρτησης.

Η παραλλαγή αξιολόγησης: δεδομένου ενός προβλήματος, να βρεθεί η βέλτιστη αντικειμενική τιμή συνάρτησης, δηλαδή την ποιότητα λύσης μιας βέλτιστης λύσης.

1.3. Το πρόβλημα των Ν βασιλισσών

Το πρόβλημα των 8 βασιλισσών είναι ένα κλασσικό παζλ που αρχικά προτάθηκε από τον Γερμανό παίκτη σκακιού Max Bezzel στην εφημερίδα " Berliner Schachzeitung" το 1848. Το πρόβλημα αφορά στην τοποθέτηση 8 βασιλισσών σε μία σκακιέρα 8x8 με τέτοιο τρόπο ώστε καμία βασίλισσα να μην απειλεί καμία άλλη. Επομένως δεν μπορούν να τοποθετηθούν δύο βασίλισσες στην ίδια γραμμή, στήλη ή διαγώνιο. Τον Ιούνιο του 1850 το πρόβλημα ξαναεμφανίζεται, αυτή τη φορά στην εφημερίδα "Illustrierten Zeitung", με τον Franz Nauck να δίνει την πρώτη λύση σε αυτό πρόβλημα τον Σεπτέμβριο του ίδιου έτους και να το γενικεύει σε πρόβλημα Ν βασιλισσών, όπου Ν βασίλισσες πρέπει να τοποθετηθουν σε σκακιέρα NxN χωρίς η μία να απειλεί την άλλη. Οι λύσεις που προκύπτουν για N=8 είναι 92. Ο ακριβής αριθμός των λύσεων είναι γνωστός για N< = 27. (Wang, Huang, Tan, Liu, Zhao, & Li, 2015)

Από τότε, πολλοί μαθηματικοί συμπεριλαμβανομένου του Carl Friederich Gauss, έχουν εργαστεί τόσο στον γρίφο των 8 βασιλισσών, όσο και στη γενικευμένη εκδοχή του για Ν βασίλισσες. Το 1874, ο S.Gunther πρότεινε μια μέθοδο που χρησιμοποιούσε προσδιοριστές για την εύρεση λύσεων ενώ ο Glaisher στη συνέχεια, βελτίωσε την προσέγγιση του Gunther. Το 1972, ο Edsger Dijkstra χρησιμοποίησε αυτό το πρόβλημα για να δείξει τη δύναμη αυτού που ονόμασε δομημένο προγραμματισμό. Δημοσίευσε μια πολύ λεπτομερή περιγραφή ενός αλγόριθμου οπισθοδρόμησης με βάση το βάθος. (Wikipedia)

Το πρόβλημα των Ν βασιλισσών ορίζεται ως η τοποθέτηση Ν βασιλισσών σε μία σκακιέρα NxN έχοντας τους εξής περιορισμούς:

- πρέπει να υπάρχει ακριβώς μία βασίλισσα σε κάθε στήλη
- πρέπει να υπάρχει ακριβώς μία βασίλισσα σε κάθε σειρά
- Δεν μπορούν να τοποθετηθούν περισσότερες από μία βασίλισσες σε οποιαδήποτε διαγώνιο

Το πρόβλημα της τοποθέτησης μη επιτιθέμενων n-βασιλισσών σε μια σκακιέρα n × n δεν είναι τετριμμένο. Υπάρχει μία λύση για N= 1, και καμία λύση για N = 2 και για N= 3. Για n = 4, . . . , 11 υπάρχουν αντίστοιχα 2, 10, 4, 40, 92, 352, 724, 2680 λύσεις. Ο Ahrens στο έργο του «Mathematische Unterhaltungen und Spiele» (Ahrens, 1901) σημειώνει ότι η αντανάκλαση γύρω από τη μεσαία σειρά του πίνακα μιας λύσης δημιουργεί μια ξεχωριστή λύση. Για παράδειγμα, για n = 4 όπου η [2 4 1 3] είναι λύση, η αντανάκλαση της [3 1 4 2] είναι μια ξεχωριστή λύση.

Πρόκειται για ένα κλασικό σύνθετο πρόβλημα ικανοποίησης περιορισμών στον τομέα της τεχνητής νοημοσύνης (AI). Έχει χρησιμοποιηθεί ως σημείο αναφοράς για την ανάπτυξη νέων τεχνικών αναζήτησης τεχνητής νοημοσύνης. Κατά τη διάρκεια των τριών τελευταίων δεκαετιών, το πρόβλημα αυτό έχει χρησιμοποιηθεί σαν παράδειγμα και σημείο αναφοράς για αλγόριθμους οπισθοιχνηλασίας, αντιμεταθέσεων, τη μέθοδο διαίρει και βασίλευε, προβλήματα ικανοποίησης περιορισμών, νευρωνικά δίκτυα και γενετικούς αλγόριθμους.

Επίσης, το πρόβλημα των N-queens έχει πολλές πρακτικές εφαρμογές, όπως δοκιμές VDSl (διαδικασίες που πραγματοποιούνται μετά την κατασκευή τσιπ ώστε να εντοπιστούν πιθανά κατασκευαστικά ελαττώματα), έλεγχος εναέριας κυκλοφορίας, συστήματα επικοινωνίας με μόντεμ, δρομολόγηση μηνυμάτων/ δεδομένων, εξισορρόπηση φορτίου σε υπολογιστές πολλαπλών επεξεργαστών, συμπίεση δεδομένων, προγραμματισμός εργασιών υπολογιστών και οπτική παράλληλη επεξεργασία. (Hu, Eberhart, & Shi, 2003)

Το πρόβλημα N-queens έχει τρεις παραλλαγές: εύρεση μιας λύσης, εύρεση μιας οικογένειας λύσεων και εύρεση όλων των λύσεων.

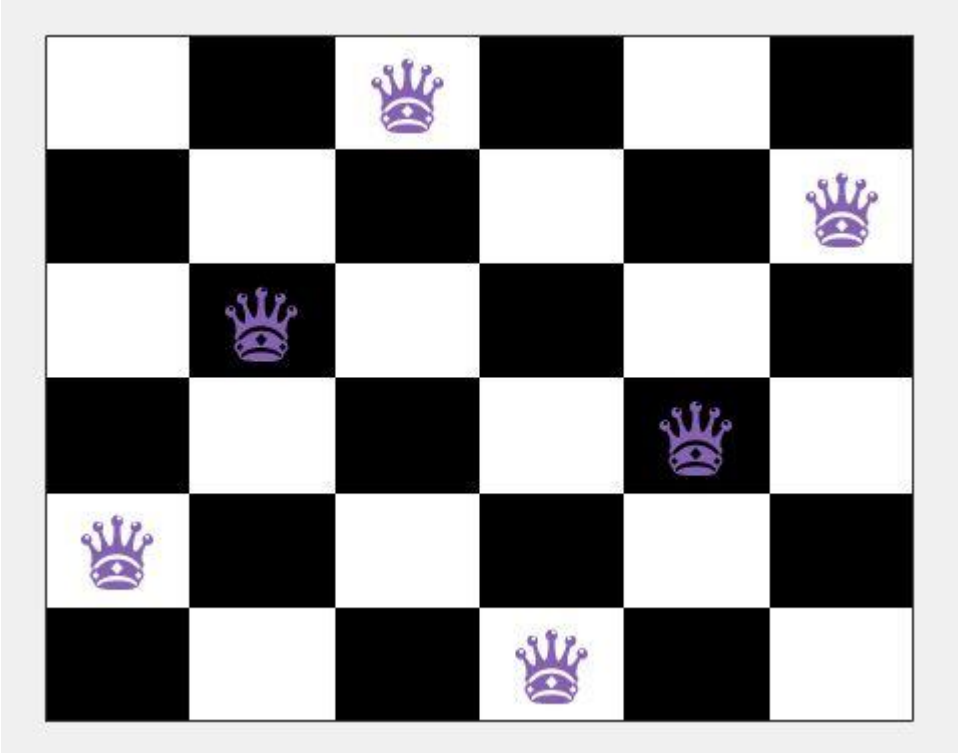
Στην παρούσα μελέτη, οι μεταθέσεις N-διαστάσεων χρησιμοποιούνται για να παρουσιάσουν τη λύση του προβλήματος των N βασιλισσών. Κάθε ακολουθία χρησιμοποιεί μια μετάθεση N αριθμών από το 1 έως το N ως πιθανή λύση. Ο i-οστός αριθμός της ακολουθίας αντιπροσωπεύει τη θέση στήλης στην i-οστή σειρά της σκακιέρας. Για να απεικονίσουμε πώς εμφανίζεται για την περίπτωση τοποθέτησης 6 βασιλισσών σε μία σκακιέρα 6x6 η ακολουθία μπορεί να είναι τα εξής: 3 6 2 5 1 4. Ο πρώτος αριθμός σημαίνει ότι η πρώτη βασίλισσα βρίσκεται στην τρίτη θέση της πρώτης σειράς, ο δεύτερος αριθμός σημαίνει ότι η δεύτερη βασίλισσα βρίσκεται στην έκτη θέση της δεύτερης σειράς, και ούτω καθεξής. Το παρακάτω σχήμα δείχνει την μετάβαση από την ακολουθία στις θέσεις της σκακιέρας.

Ακολουθία:

3	6	2	5	1	4

⇳

Σκακιέρα :



Με τη χρήση των αντιμεταθέσεων, οι οριζόντιες και κάθετες συγκρούσεις εξαλείφονται. Έτσι για να βρεθεί μια λύση, στόχος είναι η εξάλειψη των διαγώνιων συγκρούσεων. Ο αριθμός των βασιλισσών που συγκρούονται ορίζεται ως ο αριθμός των συγκρούσεων κατά μήκος των διαγωνίων της σκακιέρας. Η αντικειμενική συνάρτηση μεταβάλλεται ώστε να ελαχιστοποιηθούν οι συγκρούσεις βασιλισσών. Σκοπός είναι η αντικειμενική τιμή να φτάσει την ιδανική λύση, δηλαδή το 0.

Η αντικειμενική συνάρτηση του προβλήματος ορίζεται ως εξής:

$$O(board) = (\left(\frac{N^2}{2}\right) - L)$$

όπου ως N ορίζεται το πλήθος των βασιλισσών και L τα ζευγάρια των βασιλισσών τα οποία τρακάρουν μεταξύ τους.

Οι περιορισμοί οι οποίοι πρέπει να ισχύουν είναι οι εξής:

- Πρέπει να μεγιστοποιείται η αντικειμενική ώστε να βρεθεί η λύση
- πρέπει να μην παίρνει αρνητική τιμή η αντικειμενική συνάρτηση

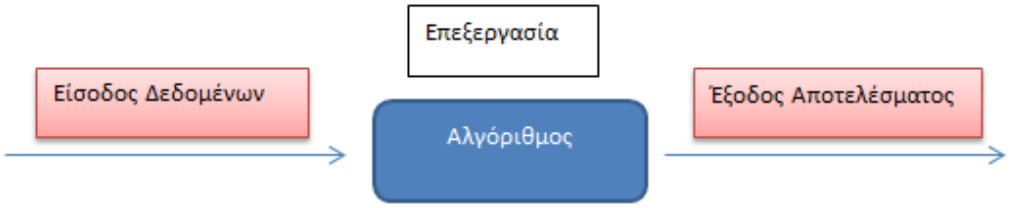
Όσον αφορά τη διαγώνιο, θεωρώντας δύο βασίλισσες Q1 και Q2 , υπολογίζεται η απόλυτη τιμή της απόστασης τους στο σκάκι για τη γραμμή και τη στήλη ξεχωριστά, από τους παρακάτω μαθηματικούς τύπους:

- deltaRow=abs(Q1row-Q2row)
- deltaCol=abs(Q1col-Q2col)

Αν deltaRow=deltaCol (ή deltaRow/deltaCol=1) τότε οι βασίλισσες βρίσκονται στην ίδια διαγώνιο. Αυτός ο έλεγχος πρέπει να επαναληφθεί για όλες τις βασίλισσες.

1.4. Αλγόριθμοι

Ως αλγόριθμος ορίζεται μια πεπερασμένη σειρά ενεργειών, αυστηρά καθορισμένων και εκτελέσιμων σε πεπερασμένο χρόνο, που στοχεύουν στην επίλυση ενός προβλήματος. Συγκεκριμένα, αλγόριθμο ονομάζουμε μια σειρά από εντολές που έχουν αρχή και τέλος, είναι σαφείς και έχουν ως σκοπό την επίλυση ενός προβλήματος. Η λέξη "αλγόριθμος", η οποία έχει κοινή ρίζα με τη λέξη "άλγεβρα", προέρχεται από μία διατριβή του Άραβα μαθηματικού Mohammed Al-Khwarizmi η οποία περιείχε συστηματικές τυποποιημένες λύσεις αλγεβρικών προβλημάτων και αποτελεί ίσως την πρώτη πλήρη πραγματεία άλγεβρας. Έτσι η λέξη αλγόριθμος καθιερώθηκε τα επόμενα χρόνια με την έννοια «συστηματική διαδικασία αριθμητικών χειρισμών». (Al-Khwarizmi) Τη σημερινή της σημασία την οφείλει στη γρήγορη ανάπτυξη των ηλεκτρονικών υπολογιστών στα μέσα του 20ου αιώνα. Η κοινή πλέον αντίληψη ενός αλγορίθμου είναι ως σύνολο εντολών που ο ηλεκτρονικός υπολογιστής εκτελεί σε δεδομένα, για να παράξει κάποιο ζητούμενο αποτέλεσμα.



Σχηματική απεικόνιση αλγορίθμου

Οι συμβατικοί αιτιοκρατικοί αλγόριθμοι εγγυώνται την εύρεση της βέλτιστης λύσης στα προβλήματα που καλούνται να επιλύσουν. Όταν όμως χρησιμοποιηθούν σε δύσκολα (NP-hard) προβλήματα βελτιστοποίησης απαιτούνται τόσο μεγάλοι υπολογιστικοί χρόνοι για την παραγωγή αποτελεσμάτων ώστε πρακτικά είναι αδύνατη η εφαρμογή τους. Έτσι, η έρευνα στράφηκε στην ανάπτυξη στοχαστικών μεθόδων οι οποίες μπορούν να βρουν μία πολύ καλή λύση (όχι πάντα τη βέλτιστη) σε εύλογο χρονικό διάστημα, πράγμα το οποίο είναι και το ζητούμενο της τρέχουσας διπλωματικής εργασιας και συγκεκριμένα για το πρόβλημα των Ν βασιλισσών. (Μαρινάκης & Μυγδαλάς, Συνδυαστική Βελτιστοποίηση, 2016)

1.4.1 Μεθευρετικοί αλγόριθμοι

Οι μεθευρετικοί αλγόριθμοι είναι μέθοδοι επίλυσης που συνδυάζουν διαδικασίες τοπικής αναζήτησης και υψηλότερου επιπέδου στρατηγικές για να δημιουργήσουν μια διαδικασία που είναι ικανή να ξεφύγει από κάποιο τοπικό ελάχιστο. Οι περισσότεροι αλγόριθμοι που έχουν αναπτυχθεί τα τελευταία χρόνια για την επίλυση προβλημάτων συνδυαστικής βελτιστοποίησης είναι αυτής της κατηγορίας. Μερικά από τα χαρακτηριστικά των μεθευρετικών αλγορίθμων τα οποία τους καθιστούν χρήσιμους για την επίλυση τέτοιων προβλημάτων είναι τα ακόλουθα:

1.Μοντελοποιούν ένα φαινόμενο που υπάρχει στη φύση.

2.Μπορούν να μεταφερθούν εύκολα σε παράλληλη μορφή.

3.Είναι προσαρμοστικοί αλγόριθμοι.

Ορισμένες μέθοδοι που ανήκουν σε αυτή τη κατηγορία είναι:

- ✓ Αλγόριθμος Βελτιστοποίησης Σμήνους Σωματιδίων (Particle Swarm Optimization).
- ✓ Προσομοιωμένη Ανόπτηση (Simulated Annealing)
- ✓ Περιορισμένη Αναζήτηση (Tabu Search)
- ✓ Γενετικοί και Εξελικτικοί Αλγόριθμοι (Genetic and Evolutionary algorithms)
- ✓ Νευρωνικά Δίκτυα (Neural Nets)
- ✓ Αλγόριθμος Βελτιστοποίησης Αποικίας των Μυρμηγκιών (Ant Colony Optimization)
- ✓ Αλγόριθμος Διασκορπισμένης Αναζήτησης (Scatter Search)
- ✓ Διαδικασία Άπληστης Τυχοποιημένης Προσαρμοστικής Αναζήτησης (Greedy Randomized Adaptive Search Procedure)
- ✓ Αλγόριθμος της Διαφορικής Εξέλιξης (Differential Evolution)

Σε αυτές τις μεθόδους εξερευνάται το πεδίο της λύσης με σκοπό να βρεθεί μία καλύτερη λύση. Στη συγκεκριμένη εργασία θα γίνει αναφορά σε μία από τις προηγμένες μεθόδους με την οποία θα επιλύσουμε το πρόβλημα των Ν βασιλισσών: τον Αλγόριθμο Αναζήτησης του κούκου (Cuckoo search algorithm). (Δελήμπαση, 2010)

Οι μεθευρετικοί αλγόριθμοι χωρίζονται σε τρείς βασικές κατηγορίες. Στους μεθευρετικούς μίας λύσης, στους εξελικτικούς (evolutionary algorithms) και στους αλγορίθμους νοημοσύνης σμήνους (swarm intelligence based algorithms).

1.4.2 Εξελικτικοί Αλγόριθμοι

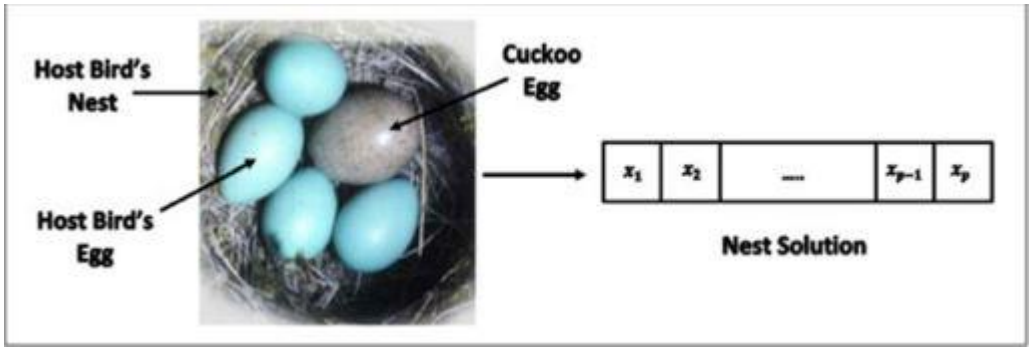
Οι εξελικτικοί αλγόριθμοι (evolutionary algorithms) σαν τεχνικές βελτιστοποίησης είναι εμπνευσμένοι από την βιολογία και τις συμπεριφορές που παρατηρούνται στην πραγματική ζωή. Οι βιολογικές συμπεριφορές χαρακτηριζονται από συγκεκριμένους μηχανισμούς που ορίζουν τη συμπεριφορά και εξέλιξη ενός πληθυσμού. Συγκεκριμένα, μιμούνται τις φυσικές διαδικασίες της επιλογής (selection) ή της αναπαραγωγής (re-production), της μετάλλαξης (mutation) και της διασταύρωσης (crossover) και τους χρησιμοποιούν ως μηχανισμό για την εύρεση καλύτερων λύσεων σε προβλήματα βελτιστοποίησης. Τέτοιοι μηχανισμοί όντας ρεαλιστικοί και παρατηρήσιμοι, είναι ιδιαίτερα ικανοί σε πολλές περιπτώσεις να περιγράψουν με έναν επαρκή τρόπο ένα πρόβλημα και να αποδώσουν εξαιρετικά όσον αφορά την βελτιστοποίηση του. Οι εξελικτικοί αλγόριθμοι διαχειρίζονται λοιπόν και οργανώνουν πληθυσμούς, οι οποίοι δρουν με μεθευρετικές (meta-heuristic) μεθόδους. Στην κατηγορία των εξελικτικών αλγορίθμων ανήκουν οι γενετικοί αλγόριθμοι (genetic algorithms), ο αλγόριθμος διαφορικής εξέλιξης (differential evolution) καθώς και άλλοι. Αξίζει βεβαίως να αναφερθεί ότι ο πιο αντιπροσωπευτικός εξελικτικός αλγόριθμος είναι ο Γενετικός Αλγόριθμος ο οποίος ακριβώς προσπαθεί να προσομοιώσει τον κύκλο της αναπαραγωγής και εξέλιξης ενός έμβιου πληθυσμού βασισμένος πολύ ισχυρά στην θεωρία του Δαρβίνου. Ωστόσο και οι υπόλοιποι αλγόριθμοι δεν παύουν να είναι εξελικτικοί όσον αφορά την κατηγοριοποίηση τους, απλώς είναι βασισμένοι σε διαφορετικούς βιολογικούς μηχανισμούς.

2.ΑΛΓΟΡΙΘΜΟΙ

2.1. Αλγόριθμος Αναζήτησης του Κούκου

Ο αλγόριθμος αναζήτησης του κούκου βασίζεται στην παρασιτική συμπεριφορά κατά τη διάρκεια αναπαραγωγής ορισμένων ειδών κούκου. Οι εμπνευστές του συγκεκριμένου αλγόριθμου, (Yang & Deb, Researchgate , 2009) παρατήρησαν τη διαδικασία αναπαραγωγής των κούκων και δημιούργησαν τον παραπάνω αλγόριθμο.

Οι κούκοι αναπαράγονται γεννώντας τα αυγά τους σε φωλιές άλλων πουλιών. Μερικά είδη παρασιτικών θηλυκών κούκων μπορούν να μιμηθούν τα χρώματα και τα σχέδια των αυγών που προέρχονται από ορισμένα είδη ξενιστών. Αυτό σαν γεγονός, μειώνει την πιθανότητα να εγκαταλειφθουν τα αβγά, και ως εκ τούτου αυξάνει την πιθανότητα επιβίωσης. Στη συνέχεια όταν το αυγό εκκολαφθεί, καθώς το αυγό του κούκου εκκολάπτεται πιο γρήγορα, απωθεί τα άλλα αυγά από τη φωλιά ώστε να έχει μεγαλύτερες πιθανότητες επιβίωσης και μιμείται τη φωνή του πουλιού στο οποίο ανήκει η φωλιά. Σε ορισμένες περιπτώσεις δημιουργείται σύγκρουση μεταξύ του ξενιστή και του κουκου και ο ξενιστής, είτε απομακρύνει τα αυγά του κούκου, είτε καταστρέφει τη φωλιά και χτίσει νέα σε άλλο μέρος.



Αναπαράσταση μιας λύσης φωλιάς στον αλγόριθμο αναζήτησης Κούκου (Oshi, Kulkarni, Kakandikar , & Nandedkar, 2017)

Ο αλγόριθμος του κούκου βασίζεται στους τρεις ακόλουθους κανόνες:

- Κάθε κούκος επιλέγει τυχαία μια φωλιά και αφήνει ένα αυγό μέσα της.
- Οι καλύτερες φωλιές με τα υψηλής ποιότητας αυγά θα κρατηθούν για την επόμενη γενιά.

- Για ένα προκαθορισμένο αριθμό φωλιών, το αβγό του κούκου θα ανακαλυφθεί με πιθανότητα pa να ανήκει στο διάστημα [0,1].

Για τον τελευταίο κανόνα έχει επικρατήσει να αντικαθίσταται ένα ποσοστό pa των χειρότερων φωλιών (λύσεων) με νέες τυχαίες λύσεις. Η ποιότητα κάθε φωλιάς υπολογίζεται με βάση το πόσο καλή είναι η λύση σε σχέση με το πρόβλημα που καλούμαστε να λύσουμε. Από εκτελεστική άποψη του αλγόριθμου θεωρούμε ότι κάθε φωλιά έχει ένα αβγό το οποίο είναι και η λύση και ότι κάθε κούκος μπορεί να γεννήσει μόνο ένα αυγό.

Όταν δημιουργούμε νέες λύσεις $x^{(t+1)}$ για το κούκο i εφαρμόζεται μια Lèvy Flight

$$x_i^{(t+1)} = x_i^{(t)} + a \oplus Lèvy(\lambda)$$

όπου: $\alpha > 0$ είναι το μέγεθος του βήματος το οποίο σχετίζεται με την κλίμακα του προβλήματος το οποίο λύνουμε.

Παρακάτω βλέπουμε έναν χαρακτηριστικό ψευδοκώδικα του αλγορίθμου

Αλγόριθμος Αναζήτησης Κούκων
begin
Αντικειμενική συνάρτηση $f(x), x = (x_1, x_2, \cdots, x_d)$
Αρχικοποίηση του πληθυσμού από n φωλιές που θα φιλοξενήσουν
τα αυγά $x_i (i = 1, 2, \cdots, n)$
while ο μέγιστος αριθμός επαναλήψεων στη φάση μετακίνησης
δεν έχει επιτευχθεί
Επέλεξε ένα κούκο (i) τυχαία και δημιούργησε μία νέα
λύση με πτήση *Lévy*
Υπολόγισε την καινούρια αντικειμενική συνάρτηση F_i
Διάλεξε τυχαία μια φωλιά j
if $F_i < F_j$ (προβλήμα ελαχιστοποίησης)
Αντικατέστησε το j με την καινούρια λύση
endif
Εγκατέλειψε ένα αριθμό Pa από τις χειρότερες φωλιές
Αντικατέστησε τις φωλιές με καινούριες χρησιμοποιώντας
πτήση *Lévy*
Κράτα τις καλύτερες λύσεις και βρες την τρέχων βέλτιστη
endwhile
Επιστροφή της καλύτερης λύσης
end

2.2 Lévy flights

Μια πτήση Lévy μπορεί να θεωρηθεί ως ένας τυχαίος περίπατος όπου το μέγεθος (μήκος) του βήματος έχει πιθανότητα (κατανομή) Lévy, μια κατανομή πιθανότητας με βαρια ουρά (Heavy Tailed). Όταν ορίζεται σαν περίπατος σε χώρο διάστασης μεγαλύτερο της μονάδας, τα βήματα που γίνονται είναι σε ισότροπες τυχαίες κατευθύνσεις. Η ονομασία Lévy flight προήλθε από τον Γάλλο

μαθηματικό Paul Pierre Lévy. Ο όρος Lévy Flight επινοήθηκε από τον Benoit Mandelbrot ο οποίος χρησιμοποίησε συγκεκριμένο ορισμό της κατανομής των μεγεθών των βημάτων. Χρησιμοποίησε τον όρο πτήση Cauchy για την περίπτωση που η κατανομή των μεγεθώ βημάτων είναι κατανομή Cauchy, και τον όρο πτήση Rayleigh για την περίπτωση που η κατανομή είναι κανονική. (Η οποία δεν αποτελεί παράδειγμα κατανομής πιθανοτήτων Heavy Tailed). Τελικά ο όρος Lévy Flight χρησιμοποιήθηκε για να αναφερθεί σε διακριτό πλέγμα αντί για συνεχή χώρο. Μεταγενέστεροι ερευνητές επέκτειναν τη χρήση του όρου " Lévy flight" για να συμπεριλάβουν και τις περιπτώσεις όπου ο τυχαίος περίπατος πραγματοποιείται σε ένα διακριτό πλέγμα και όχι σε ένα συνεχή χώρο. Η Lévy flight είναι μια αλυσίδα Markov. Η εκθετική της ιδιότητα της δίνει μια αναλλοίωτη στην κλίμακα ιδιότητα και χρησιμοποιείται για τη μοντελοποίηση δεδομένων για την έκθεση και την προβολή συστάδων.

Στη φύση πολλά έντομα και ζώα ακολουθούν τις ιδιότητες της Lévy flight. Πρόσφατες έρευνες των Reynolds και Frye απέδειξαν ότι οι μύγες των φρούτων ή οι μύγες των καρπών (drosophila melanogaster) καλύπτουν τον ουρανό χρησιμοποιώντας πολυάριθμες σειρές από ευθείες διαδρομές/πτήσεις που ακολουθούνται από μια απότομη 90 μοιρών στροφή, η οποία είναι μοτίβο αναζήτησης τύπου Lévy Flight σε συνεχή κλίμακα. (Το μοτίβο κυνηγιού και συλλογής τροφής εκθέτει την τυπική μορφή του Lévy flight, που παρατηρήθηκε από τον Ju/'hoansi και στην ανθρώπινη συμπεριφορά.)

Έρευνες πάνω στην ανθρώπινη συμπεριφορά όπως αυτή του Ju/'hoasi που αφορά το κυνήγι και τη συλλογή τροφής δείχνουν επίσης το τυπικό χαρακτηριστικό των Lévy flights.Ακόμα και το φως μπορεί να σχετίζεται με Lévy flights.
Τέτοιες συμπεριφορές έχουν εφαρμοστεί στην βελτιστοποίηση και στην βελτιστη αναζήτηση, και τα πρώτα αποτελέσματα δείχνουν την πολλά υποσχόμενη ικανότητα της μεθόδου αυτής.

2.3.Αλγόριθμος Επαναληπτικής Τοπικής Αναζήτησης

Η τοπική αναζήτηση αποτελεί μία από τις αρχαιότερες μεθόδους βελτιστοποίησης και στηρίζεται στη μέθοδο δοκιμής και σφάλματος. Εχει αποδειχθεί πολύ επιτυχημένη εφαρμοσμένη στην πράξη και έχει λύσει πολλά προβλήματα συνδυαστικής βελτιστοποίησης.
Στη συγκεκριμένη διπλωματική χρησιμοποιήθηκε η μέθοδος 1-1 exchange για καλύτερα αποτελέσματα στον αλγόριθμο του κούκου.

2.3.1 Γενική ανάλυση αλγορίθμου

Στον αλγόριθμο της επαναληπτικής αναζήτησης γίνεται μια προσπάθεια να βελτιωθούν διαδοχικά τοπικά ελάχιστα.

Η μέθοδος της επαναληπτικής τοπικής αναζήτησης βελτιώνει τις κλασικές μεθόδους πολυεναρκτήριας τοπικής αναζήτησης εισάγοντας μια διαταραχή στη λύση που έχει οδηγήσει σε τοπικό ελάχιστο και χρησιμοποιώντας την καινούργια λύση ως αρχική λύση για το πρόβλημα.

Με αυτόν τον τρόπο δημιουργείται μία νέα αρχική λύση. Στη συνέχεια εφαρμόζεται μια διαδικασία τοπικής αναζήτησης στην αρχική λύση και σε κάθε επανάληψη, εφαρμόζεται μια διαταραχή στη λύση που προήλθε από τη διαδικασία της τοπικής αναζήτησης.

Στην καινούργια λύση εφαρμόζεται πάλι μια διαδικασία τοπικής αναζήτηση. Η διαδικασία τοπικής αναζήτησης που χρησιμοποιείται μπορεί να είναι η ίδια με την προηγούμενη ή και κάποια διαφορετική.

Η καινούργια λύση που προκύπτει γίνεται αποδεκτή κάτω από ορισμένες συνθήκες. Αυτή η διαδικασία συνεχίζεται μέχρι να επιτευχθεί το κριτήριο τερματισμού.

Τα βασικά χαρακτηριστικά του αλγορίθμου επαναληπτικής τοπικής αναζήτησης είναι τα εξής:

- **Ο αλγόριθμος τοπικής αναζήτησης**, δηλαδή οποιοσδήποτε αλγόριθμος ή συνδυασμός αλγορίθμων.
- **Η διαδικασία διαταραχής**. Αυτή η διαδικασία μπορεί να θεωρηθεί ως μια μεγάλη τυχαία κίνηση πάνω στην τρέχουσα λύση. Η διαδικασία διαταραχής πρέπει να κρατήσει κάποια στοιχεία της αρχικής λύσης, για να μη γίνει τελείως τυχαία η καινούργια λύση, αλλά θα πρέπει να μεταλλάξει πολύ σημαντικά τα υπόλοιπα σημεία ελπίζοντας να μεταφερθεί σε άλλο σημείο που μπορεί να οδηγήσει σε καλύτερο τοπικό ελάχιστο ή και στο βέλτιστο. Έχουν προταθεί αρκετές διαδικασίες διαταραχής:
    - Καθορισμένες ή διακριτές διαταραχές. Το μήκος της διαταραχής που εφαρμόζεται σε ένα τοπικό ελάχιστο μπορεί να καθοριστεί ως εξής:
        - *Στατικό*. Το μήκος είναι καθορισμένο εκ των προτέρων πριν ξεκινήσει η αναζήτηση
        - *Δυναμικό*. Το μήκος της διαταραχής καθορίζεται δυναμικά χωρίς να λαμβάνεται υπόψη η μνήμη κατά τη διάρκεια της αναζήτησης.
        - *Προσαρμοστικό*. Σε αυτή την περίπτωση το μήκος της διαταραχής προσαρμόζεται κατά τη διάρκεια της αναζήτησης λαμβάνοντας υπόψη πληροφορίες από τη μνήμη αναζήτησης.
    - Τυχαίες ή ημικαθορισμένες διαταραχές. Το μήκος της διαταραχής μπορεί να είναι τυχαίο όπου κάθε κίνηση δημιουργείται τυχαία στη γειτονιά αναζήτησης.

- **Το κριτήριο αποδοχής**. Το κριτήριο αποδοχής περιγράφει τις συνθήκες κάτω από τις οποίες το καινούγιο τοπικό ελάχιστο μπορεί να αντικαταστήσει το παλιό τοπικό ελάχιστο. Έτσι κάποιος θα μπορούσε να δεχτεί μόνο τις λύσεις που βελτιώνουν την αρχική λύση με στόχο να εντατικοποιηθεί όσο το δυνατό περισσότερο η αναζήτηση γύρω από κάποιο σημείο ενώ κάποιος άλλος θα

μπορούσε να δεχθεί όλες τις λύσεις με στόχο να εξερευνήσει όσο το δυνατό περισσότερο το χώρο λύσεων .

Επομένως, σε ένα γενικό αλγόριθμο τοπικής αναζήτησης αρχίζουμε από μια αρχική εφικτή λύση t ∈ F και χρησιμοποιούμε μία υπορουτίνα για να ψάξουμε για μια καλύτερη λύση στη γειτονιά της αρχικής λύσης.

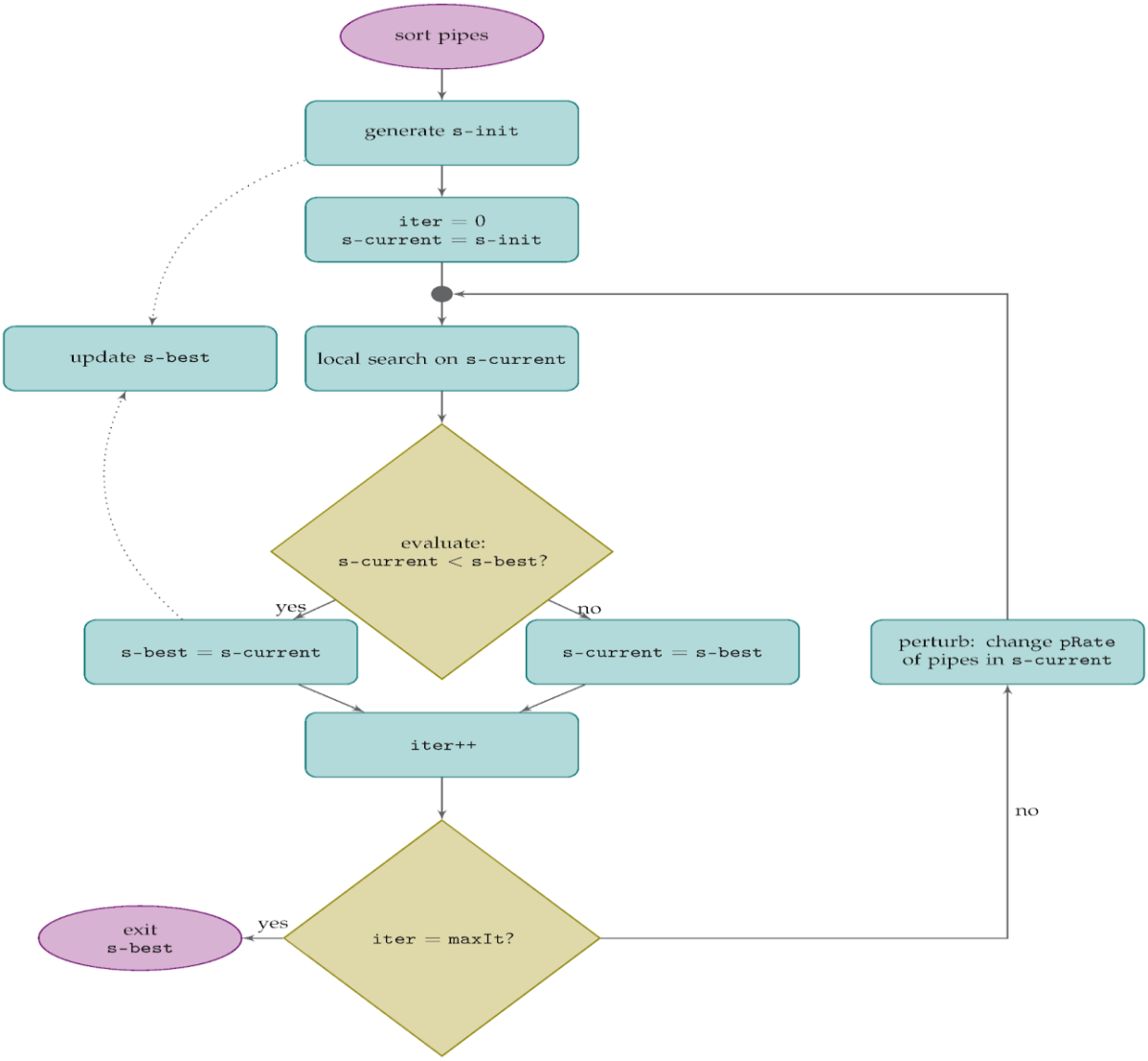
Όσο βρίσκεται μια βελτιωμένη λύση, την εφαρμόζουμε και επαναλαμβάνουμε τη διαδικασία αναζήτησης από τη νέα λύση. Όταν φτάσουμε σε τοπικό ελάχιστο σταματάμε.

Παρακάτω παρουσιάζεται ένας γενικός ψευδοκώδικας επαναληπτικής τοπικής αναζήτησης:

Αλγόριθμος Επαναληπτική Τοπική Αναζήτηση
Αρχικοποίηση
Επέλεξε μία μέθοδος τοπικής αναζήτησης LS
Επέλεξε μια αρχική λύση s_0
$s_1 = LS(s_0)$
repeat
$s' = Perturb(s_1, search\ history)$ (Διαταραχή της λύσης)
$s'_1 = LS(s')$
$s_1 = Accept(s_1, s'_1, search\ memory)$ (Κριτήριο Αποδοχής)
until Κριτήριο Τερματισμού
Επέστρεψε τη βέλτιστη λύση.

Παρακάτω παρουσιάζετε και ένα διάγραμμα ροής του αλγορίθμου τοπικής αναζήτησης.

Το σχήμα βοηθά στην καλύτερη κατανόηση της διαδικασίας εύρεσης καλύτερης λύσης.



Διάγραμμα ροής αλγορίθμου επαναληπτικής τοπκής αναζήτησης (De Corte & Sörensen, 2016)

Το σημαντικότερο ίσως στοιχείο για την επιτυχία της διαδικασίας είναι η επιλογή της μεθόδου που θα χρησιμοποιηθεί για την αναζήτηση.

Τα σημαντικότερα προβλήματα που πρέπει να αντιμετωπιστούν κατά τη διάρκεια της φάσης της σχεδίασης ενός αλγορίθμου τοπικής αναζήτησης είναι:

1. Το πρώτο είναι η επιλογή της γειτονιάς, και αυτό σχετίζεται με την ανησυχία που υπάρχει για το αν είναι ή όχι εφικτή η λύση.

2. Το δεύτερο πολύ σημαντικό στοιχείο που πρέπει να αντιμετωπίσουμε είναι η ποιότητα της αρχικής λύσης. Είναι ένα πολύ σημαντικό στοιχείο, γιατί όσο καλύτερη είναι η αρχική λύση τόσο περισσότερες πιθανότητες υπάρχουν να οδηγηθούμε ευκολότερα και γρηγορότερα σε βελτίωση της λύσης με τη χρήση της τοπικής αναζήτησης.

3. Το τρίτο και σημαντικότερο στοιχείο για την επιτυχία του αλγορίθμου είναι η μέθοδος που θα χρησιμοποιηθεί για να βελτιώσει την αρχική λύση. Οι περισσότερες μέθοδοι που έχουν αναπτυχθεί στη βιβλιογραφία παρουσιάζουν σημαντικά προβλήματα όσον αφορά την ποιότητα της λύσης που επιστρέφουν και αυτό γιατί αν κάποια στιγμή πέσουν σε τοπικό ελάχιστο είναι πολύ δύσκολο να ξεκολλήσουν από εκεί.

2.3.2 Μέθοδοι τοπικής αναζήτησης

Οι αλγόριθμοι τοπικής αναζήτησης λειτουργούν πάνω σε μία ήδη υπάρχουσα λύση, δηλαδή πάνω σε ένα ήδη υπάρχον δρομολόγιο και λειτουργούν σταδιακά βελτιώνοντας τη λύση.
Έτσι μειώνουν το μήκος του δρομολογίου μέχρι να μην υπάρχουν πλέον αλλαγές που να οδηγούν σε περαιτέρω βελτίωση. (Κινικλής, 2009)

Σε αυτό το σημείο θα παρουσιαστούν αναλυτικά οι πιο διαδεδομένοι μέθοδοι τοπικής αναζήτησης καθώς και η μέθοδος 1-1 exchange η οποία επιλέχθηκε για τη συγκεκριμένη διπλωματική εργασία. Έγινε και εφαρμογή της μεθόδου 2-2 exchange, όμως τα αποτελέσματα ήταν υποδεέστερα της 1-1 exchange. Να αναφερθεί πως οι αλγόριθμοι αυτοί εφαρμόστηκαν στην συνάρτηση του αλγόριθμου του κούκου για το πρόβλημα των Ν βασιλισσών.

Μέθοδος 2-opt

Έστω ότι δημιουργείται μία τυχαία λύση.

Η μέθοδος 2-opt αποτελείται από τη διαγραφή 2 ακμών και την επανασύνδεση δύο μονοπατιών με διαφορετικό τρόπο για να καθορίσουμε μια

καινούργια διαδρομή. Παρατηρείται ότι υπάρχει ένας μόνο τρόπος για να επανασυνδέσουμε τα μονοπάτια. Η διαδικασία είναι η εξής:

Βήμα 1 : Έστω T η τρέχουσα διαδρομή

Βήμα 2 : Για κάθε κόμβο i=1,2...,n εξετάζουμε όλες τις πιθανές 2-opt κινήσεις που μπορεί να γίνουν από την i και την επόμενή της μέσα στην διαδρομή. Αν με αυτόν τον τρόπο μπορούμε να μειώσουμε το κόστος της διαδρομής, τότε επιλέγουμε την καλύτερη 2-opt κίνηση και εφαρμόζουμε τις αλλαγές στην διαδρομή T.

Βήμα 3 : Αν δεν μπορούμε να έχουμε επιπλέον βελτίωση, σταματάμε.

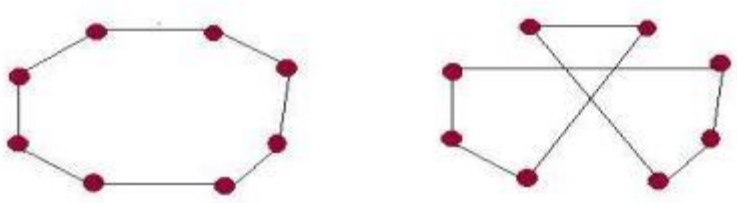
Στη χειρότερη περίπτωση, μπορεί να εγγυηθεί ότι μια κίνηση βελτίωσης μειώνει το μήκος της διαδρομής τουλάχιστον μία μονάδα.

Εφαρμογή 2-opt

Μέθοδος 3-opt

Ο παραπάνω αλγόριθμος είναι αδιαμφισβήτητα ένας αλγόριθμος που επιφέρει βελτίωση στο μήκος της διαδρομής.

Αν θέλουμε να είμαστε πιο ευέλικτοι θα μπορούσαμε να σπάσουμε τον αλγόριθμο σε τρία μέρη αντί για δύο και να συνδυάσουμε τα μονοπάτια με τον καλύτερο δυνατό τρόπο. Η διαδικασία είναι η εξής:
Βήμα 1 : Έστω T η τρέχουσα διαδρομή
Βήμα 2 : Για κάθε κόμβο ieV υπολογίζουμε ένα σύνολο από κόμβους N(i) .
Βήμα 3 :Για κάθε κόμβο i=1,2,...,n εξετάζουμε όλες τις πιθανές 3-opt κινήσεις που μπορεί να γίνουν και οι οποίες διαγράφονται από τις τρεις πλευρές έχοντας η κάθε μία από αυτές μια πλευρά στην N(i). Αν με αυτόν τον τρόπο μπορούμε να μειώσουμε το κόστος της διαδρομής τότε επιλέγουμε την καλύτερη 3-opt κίνηση και εφαρμόζουμε τις αλλαγές στη διαδρομή T.
Βήμα 4 : Αν δεν μπορούμε να βρούμε επιπλέον βελτίωση, σταματάμε.



εφαρμογή 3-opt

1-0 relocate

Στη συγκεκριμένη μέθοδο ένας κόμβος επανατοποθετειται από το σημείο που βρίσκεται σε ένα άλλο σημείο, μεταξύ δύο άλλων. Δηλαδή η μέθοδος relocate βασίζεται στην ιδέα της διαγραφής ενός πελάτη από μια διαδρομή και επανατοποθέτησή του σε μια άλλη διαδρομή με καλύτερο κόστος.

Έστω μια αρχική διαδρομή

1 7 4 9 5 8 3 10 2 6 1

Αν πάρουμε τον κόμβο 5 και τον μεταφέρουμε μία θέση θα έχουμε την λύση

1 7 4 5 9 8 3 10 2 6 1

Αν πάρουμε τον κόμβο 6 ανάμεσα στον 7 και τον 4 παίρνουμε την ακόλουθη λύση

1 7 6 4 5 9 8 3 10 2 1

2-0 relocate

Στη συγκεκριμένη μέθοδο ακολουθείται η ίδια διαδικασία με την 1-0 relocate αλλά αντί για έναν κόμβο, μετακινούνται και επανατοποθετούνται δύο κόμβοι σε διαδοχική σειρά.

Έστω μια αρχική διαδρομή

1 7 4 9 5 8 3 10 2 6 1

Αν πάρουμε τους κόμβους 8 και 3 και τους μεταφέρουμε μαζί σε άλλη θέση θα έχουμε την λύση

1 8 3 7 4 9 5 10 2 6 1

1-1 exchange

Στο συγκεκριμένο αλγόριθμο 2 κόμβοι αλλάζουν θέση μεταξύ τους.

Έστω αρχική διαδρομή

1 7 4 9 5 8 3 10 2 6 1

Αντιμεταθέτουμε τους κόμβους 2 και 8

1 7 4 9 5 2 3 10 8 6 1

2-2 exchange

Στο συγκεκριμένο αλγόριθμο αλλάζουν θέση 2 δυάδες κόμβων μεταξύ τους.

Έστω αρχική διαδρομή

1 7 4 9 5 8 3 10 2 6 1

Αντιμεταθέτουμε τον κόμβο 2 με τον 8 και τον κόμβο 7 με τον 6

1 6 4 9 5 2 3 10 8 7 1

Η συγκεκριμένη μέθοδος είναι και αυτή που εφαρμόστηκε στη συγκεκριμένη διπλωματική εργασία. Στο προηγούμενο υποκεφάλαιο παρουσιάστηκαν διαφορετικοί μέθοδοι της τοπικής αναζήτησης και τα βήματα που ακολουθούνται σε κάθε περίπτωση. Η δουλειά αυτής της διπλωματικης επικεντρώθηκε στην χρήση δύο εξ αυτών, την 1-1 exchange και την 2-2 exchange. Αυτες οι δύο μέθοδοι βασίζονται στην αντιμετάθεση στοιχείων κάτι το οποίο ταιριαζει στην κίνηση των βασιλισσών στη σκακιέρα και δίνει βέλτιστα αποτελέσματα στη λύση του προβλήματος.

Στο επόμενο κεφάλαιο της εργασίας θα γίνει ανάλυση καθώς και εφαρμογή των αλγορίθμων που ήδη αναφέρθηκαν, για την επίλυση του προβλήματος των N βασιλισσών.

3.Επίλυση προβλήματος – Εφαρμογή αλγορίθμων

Εισαγωγή

Στο παρακάτω κεφάλαιο θα γίνει ανάλυση του κώδικα ο οποίος εφαρμόστηκε για την εφαρμογή του αλγόριθμου αναζήτησης του κούκου στο πρόβλημα των N βασιλισσών.

Οι συναρτήσεις εφαρμόστηκαν στην έκδοση 2017a του προγράμματος της MATLAB και σε υπολογιστή με επεξεργαστή Intel® Core™ i3-4005U.

3.1. Βασική Συνάρτηση

Στην κύρια συνάρτηση αρχικά ζητείται από τον χρήστη να δώσει τον αριθμό των βασιλισσών. Αρχικά ελέγχεται αν ο αριθμός είναι θετικός ή αρνητικός. Αν ο αριθμός είναι αρνητικός εμφανίζεται μήνυμα σφάλματος και ο αλγόριθμος τερματίζεται. Αν ο αριθμός είναι θετικός συνεχίζεται η διαδικασία.

Στην περίπτωση όπου ο χρήστης δώσει τον αριθμό 1, ο αλγόριθμος ειδοποιεί τον χρήστη ότι υπάρχει μόνο μια λύση, ενώ αντίστοιχα όταν ο χρήστης δίνει τον αριθμό 2 ή 3, η λύση είναι αδύνατη.

Στη συνέχεια καλείται η συνάρτηση "cuckoo_search_my" μέσα στην οποία πραγματοποιούνται όλες οι διαδικασίες του κούκου, όπως θα δούμε σε επόμενο κεφάλαιο.

Μετά την εφαρμογή της συνάρτησης του κούκου, γίνεται επιστροφή της καλύτερης λύσης σε μορφή διανύσματος [best solution], η μέγιστη τιμή της αντικειμενικής συνάρτησης [fmax] καθώς και η μέγιστη τιμή που θα έπρεπε να έχει η αντικειμενική συνάρτηση αν δεν υπήρχε καμία σύγκρουση βασιλισσών [f optimal].

Τέλος, καλείται η συνάρτηση chessboard η οποία δημιουργεί την σκακιέρα μέσα στην οποία τοποθετούνται οι βασίλισσες με βάση το διάνυσμα [best solution].

3.2 Αντικειμενική συνάρτηση και περιορισμοί

Σε αυτό το υποκεφάλαιο θα γίνει παρουσίαση της αντικειμενικής συνάρτησης του προβλήματος καθώς και των περιορισμών. Ακόμα θα γίνει ανάλυση της συνάρτησης penalties η οποία αφορά τα ζευγάρια βασιλισσών τα οποία «τρακάρουν» μεταξύ τους.

3.2.1 Αντικειμενική συνάρτηση

Στη συγκεκριμένη συνάρτηση ορίζεται η αντικειμενική συνάρτηση του προβλήματος. Σκοπός είναι η συγκεκριμένη συνάρτηση να έχει τη βέλτιστη τιμή. Για να γίνει αυτό πρέπει ο αριθμός των βασιλισσών που τρακάρουν να είναι μηδενικός, δηλαδή η μεταβλητή L να είναι ίση με το μηδέν.

Η αντικειμενική συνάρτηση όπως έχει ήδη αναφερθεί στο 1ο κεφάλαιο, είναι η εξής:

$$O = \left(\frac{N^2}{2}\right) - L$$

Όπου N ο αριθμός βασιλισσών που δίνεται από τον χρήστη και L ο αριθμός των βασιλισσών που τρακάρουν μεταξύ τους.

3.2.2. Συγκρούσεις (penalties)

Η συνάρτηση penalties όπως έχει ήδη αναφερθεί, βοηθά στην εύρεση των ζευγαριών βασιλισσών που τρακάρουν. Η μεταβλητή L είναι υψηλής σημασίας για την επιλυση του προβλήματος.

Αρχικά δημιουργώ έναν πίνακα με μηδενικά στοιχεία και στη συνέχεια του δίνω την τιμή 1, σε όποιο κελί υπάρχει βασίλισσα.

Αρχικοποιούμε το L=0 ώστε να γίνει σωστή καταμέτρηση των ζευγαριών που τρακάρουν.

Στη συνέχεια, μέσω μιας επανάληψης, δημιουργώ έναν πίνακα «conflict queens» ο οποίος έχει 2 στήλες και τόσες γραμμές, όσες είναι η μεταβλητή pairs. Αυτό συμβαίνει γιατί η μεταβλητή αυτή εντοπίζει πόσα είναι συνολικά τα ζευγάρια βασιλισσών, όπως φαίνεται στην παρακάτω εικόνα.

```
function [L, conflict_queens] = penalties(N, p)
%PENALTIES Computes the penalty for N queens
    queens = zeros(N);
    for i = 1:N
        queens(i, p(i)) = 1;
    end
    L = 0;
    pairs = 0;
    for i = 1 : N
        pairs = pairs + N-i;
    end
    conflict_queens = zeros(pairs, 2);
    for i_q1 = 1:N-1
        Q1_row = i_q1;
        Q1_col = find(queens(i_q1, :) == 1);
        for i_q2 = (i_q1+1):N
            Q2_row = i_q2;
            Q2_col = find(queens(i_q2, :) == 1);
            deltaRow = abs(Q1_row-Q2_row);
            deltaCol = abs(Q1_col-Q2_col);
            if deltaRow == deltaCol
                L = L + 1;
                conflict_queens(L, :) = [i_q1, i_q2];
            end
        end
    end
end
```

Ακολούθως, έχουμε μια επανάληψη η οποία μέσω της εντολής «find(queens))» ψάχνει στον πίνακα queens πού υπάρχει μονάδα και τοποθετεί αυτή τη θέση στη μεταβλητή «Q1_col». Το ίδιο κάνει και για την αμέσως επόμενη βασίλισσα σε άλλη στήλη, και εκχωρεί την θέση στη μεταβλητή «Q2_col». Εν συνεχεία, υπολογίζεται η απόλυτη τιμή για τις γραμμές και τις στήλες αντίστοιχα. Η σύγκριση αυτή γίνεται μέσω των εντολών :

deltaRow=abs(Q1_row-Q2_row)

deltaCol=abs(Q1_col-Q2_col)

Στην περίπτωση που οι παραπάνω δύο μεταβλητές είναι ίσες (δηλαδή η deltaRow με την delta_Col) σημαίνει πως οι βασίλισσες βρίσκονται στην ίδια διαγώνιο και τότε ο αριθμός των βασιλισσών που τρακάρουν αυξάνεται κατά μία μονάδα (L) και στον πίνακα «conflict_queens» τοποθετούνται οι θέσεις των βασιλισσών που τρακάρουν.

Επομένως, η συγκεκριμένη συνάρτηση επιστρέφει στον κώδικα τον αριθμό των βασιλισσών που τρακάρουν καθώς και τα ζευγαρια που τρακάρουν και τη θέση τους.

3.3 Συνάρτηση του Cuckoo

Η συνάρτηση του αλγόριθμου αναζήτησης του κούκου είναι η πιο σημαντική συνάρτηση, καθώς μέσα σε αυτή ακολουθούνται οι περισσότερες διαδικασίες για την επίλυση του προβλήματος των N βασιλισσών.

Αρχικά ορίζεται η μεταβλητή n=100 η οποία δίνει τις πιθανές λύσεις του προβλήματος. Επίσης ορίζεται και η πιθανότητα pa=0.25 που μας δείχνει το ποσοστό πιθανότητας να ανακαλυφθεί το αυγό του κούκου από τον ξενιστή. Ακόμα ορίζονται και τα όρια της σκακιέρας τα οποία ήταν δεδομένα ως Lb=-5*ones(1,N) και Ub=5*ones(1,N). Στη συνέχεια δημιουργείται ένας πίνακας «nest» όπου αρχικά ορίζεται ως μηδενικός, ο οποίος αποτελείται από n γραμμές και N στήλες. Δηλαδή ο nest περιλαμβάνει στην δική μας περίπτωση 100 γραμμές , οι οποίες στη συνέχεια θα «γεμίσουν» από 100 διαφορετικές τυχαίες λύσεις- διανύσματα. Έγιναν διάφορες δοκιμές για την επιλογή του n. Στην περίπτωση που το n είναι πολύ μικρό , ο αλγόριθμος έχει λιγότερες πιθανότητες να πλησιάσει την βέλτιστη λύση. Στην περίπτωση που το n είναι πολύ μεγάλο, ο αλγόριθμος αργεί πολύ να βγάλει αποτελέσματα.

Ακολουθεί ένα παράδειγμα για n=6 και N=4 :

1	2	3	4
3	2	4	1
2	3	1	4
4	1	2	3
2	4	3	1
4	3	1	2

Στη συνέχεια μέσω της εντολής «randperm» εκχωρούνται τυχαίες λύσεις για το δωθέν N στην μεταβλητή p και μέσω της συνάρτησης “discrete_to_continuous”, η οποία θα αναλυθεί παρακάτω, μετατρέπονται οι διακριτές τιμές σε συνεχείς και εκχωρούνται στον πίνακα «nest». Επιπλέον, αρχικοποιείται η μεταβλητή «fitness» με

μία πολύ μεγάλη αρνητική τιμή, ώστε να συγκριθεί στη συνέχεια με την αμέσως καλύτερη αντικειμενική συνάρτηση.

Στο συγκεκριμένο πρόβλημα, σκοπός είναι η μεγιστοποίηση της αντικειμενικής συνάρτησης. Για αυτό το λόγο ξεκινάμε με κάτι τόσο μικρό. Στην περίπτωση όπου το πρόβλημα μας θα ηταν ελαχιστοποίησης, θα έπρεπε η πρώτη τιμή της αντικειμενικής συνάρτησης να ήταν πολύ μεγάλη, ώστε να μην επηρεάζεται το αποτέλεσμα του αλγορίθμου. Ορίζεται ως «foptimal» η καλύτερη τιμή της αντικειμενικής συνάρτησης, δηλαδή η τιμή για την οποία ο αριθμός των ζευγαριών βασιλισσών που τρακάρει είναι μηδενικός. Αυτή η τιμή υπολογίζεται ώστε να έχουμε πιο ξεκάθαρη εικόνα, για το πόσο κοντά στο βέλτιστο βρίσκεται η τιμή που υπολογίζει ο δικός μας αλγόριθμος. Στη συνέχεια, μέσω της συνάρτησης "get_best_nest" υπολογίζεται η καλύτερη μέχρι στιγμής λύση. Ακολουθεί η συνάρτηση "get_cuckoos" η οποία κάνε εφαρμογή των Lévy flights και ανακαλύπτει νέες λύσεις. Εν συνεχεία, ξανά με τη βοήθεια της συνάρτησης "get_best_nest", γίνεται σύγκριση των νέων τιμών με τις προηγούμενες, ώστε να πάρουμε τα βέλτιστα αποτελέσματα μέχρι αυτή τη στιγμή.

Στη συνάρτηση "empty_nests" αντικαθίστανται ορισμένες όχι τόσο καλές λύσεις στον «nest». Αυτός ο έλεγχος γίνεται με την πιθανότητα pa, η οποία έχει ήδη οριστεί ως *0,25*.

Ακολούθως, γίνεται πάλι σύγκριση των προηγούμενων αποτελεσμάτων με τα καινούργια, ώστε να κρατήσουμε τα καλύτερα μέσα στη συνάρτηση "get_best_nest" . Δηλαδή συγκρίνει τα fitness(αντικειμενικές συναρτήσεις) των nest και best_nest και κρατάμε αυτό με το μεγαλύτερο fitness.

Οι παραπάνω συναρτήσεις, (get_best_nest,get_cuckoos,get_best_nest) είναι συγκεκριμένες και έχουν αντληθεί από το έργο του Xin-She Yang στο Cambridge University. Η διπλωματική εργασία, σε ορισμένα σημεία κώδικα είναι εμπνευσμένη από τον βασικό κώδικα του Xin - She Yang. Θα γίνει περαιτέρω ανάλυση αυτών των συναρτήσεων στη συνέχεια.

Βγαίνοντας από αυτές τις συναρτήσεις, οι τιμές του nest_init μετατρέπονται ξανά σε διακριτές, μέσω της συνάρτησης "continuous_to_discrete" και γίνεται εμφάνιση των βέλτιστων αποτελεσμάτων μετά τις διαδικασίες του κούκου, στο command window. Δηλαδή παρουσιάζεται η καλύτερη λύση σε διάνυσμα(nest_init) , η μέγιστη τιμή της αντικειμενικής (finit) και η foptimal.

Στη συνέχεια γίνεται ένας έλεγχος μεταξύ της finit και της foptimal.

Στην ουσία, αν η finit που βρίσκεται μέσω των υποσυναρτήσεων του κούκου ταυτίζεται με την foptimal, τότε finit=fmax, έχει βρεθεί βέλτιστο αποτέλεσμα και δεν υπάρχει λόγος να γίνει χρήση της τοπικής αναζήτησης. Σε αντίθετη περίπτωση

παίρνουμε τις καλύτερες τιμές που έχουν προκύψει από την τοπική αναζήτηση και τότε fex = fmax.

```
if finit == foptimal
    fmax = finit;
    best_nest_discrete = nest_init;
else
    %conflict queens=tsekarw tis theseis twn basilisswn pou trakaroun
    [L_new, conflict_queens] = penalties(N, nest_init);
    [nest_ex, fex] = exchange(N, nest_init, finit, conflict_queens);

    fmax = fex;
    best_nest_discrete = nest_ex;

end
```

3.3.1 Μετατροπή διακριτών τιμών σε συνεχείς και αντίστροφα

Η Συνάρτηση discrete_to_continuous

Η συγκεκριμένη συνάρτηση η οποία χρησιμοποιείται στον αλγόριθμο αναζήτησης του κούκου, μετατρέπει τις διακριτές τιμές του πίνακα nest σε συνεχείς. Αυτό συμβαίνει γιατί ο συγκεκριμένος αλγόριθμος χρειάζεται συνεχείς τιμές για να βγάλει αποτέλεσμα.

Η διαδικασία που ακολουθείται είναι η εξής. Αρχικά εντοπίζω τη μέγιστη τιμή του διανύσματος p μέσω της εντολής max(p). Στη συνέχεια διαιρώ την κάθε τιμή του διανύσματος p με τη μέγιστη τιμή ώστε να γίνουν συνεχείς.

```
function [ p ] = discrete_to_continuous( p )

%input p
% Find the maximum value of p.
A=max(p);
%divide each element of p with the maximum value
p=(p/A);

end
```

Η συνάρτηση continuous_to_discrete

Στη συγκεκριμένη συνάρτηση γίνεται μετατροπή των τιμών από συνεχείς σε διακριτές.

Αρχικά, υπολογίζουμε το μέγεθος του p_cont και συγκεκριμένα τις στήλες και το εκχωρούμε στη μεταβλητή n.

Στη συνέχεια με τη βοήθεια μίας επανάληψης από 1 έως n, και της εντολής [A,intex] = min(p_cont) εντοπίζω το μικρότερο στοιχείο ενώ στην μεταβλητή p_disc(index)=i εκχωρείται η θέση στην οποία βρίσκεται αυτό το i-οστό μικρότερο στοιχείο.

Τέλος, εκχωρούμε την τιμή άπειρο στην τιμή που ήδη έχουμε ελέγξει ώστε να βγει εκτός συναγωνισμού στην κατάταξη μέσω της εντολής p_cont(index)=inf.

```
function [ p_disc ] = continuous_to_discrete( p_cont )

    n = size(p_cont, 2);
    p_disc= zeros(1, n);

    for i=1:n

        [A, index] = min(p_cont);

        p_disc(index)=i;

        p_cont(index) = inf;
    end
end
```

3.2.2. Υποσυναρτήσεις του Αλγόριθμου του Κούκου

Η συνάρτηση get_best_nest

Η συνάρτηση αυτή βασίζεται στην υλοποίηση του Xin–She Yang (Yang, www.mathworks.com) και υπολογίζει την μέχρι στιγμής καλύτερη λύση.

Αρχικά υπολογίζεται το μέγεθος των στηλών του nest, δηλαδή ο αριθμός των βασιλισσών και με τη βοήθεια μίας επαναληπτικής διαδικασίας, από μονάδα έως το μέγεθος της δεύτερης διάστασης του nest θα υπολογιστούν οι νέες καλύτερες λύσεις.

Η συγκεκριμένη υποσυνάρτηση για να υπολογίσει σωστά τις νέες λύσεις θα πρέπει να γίνει μετατροπή των τιμών από συνεχείς ξανά σε διακριτές. Στη συνέχεια υπολογίζεται ο αριθμός των βασιλισσών που τρακάρει μέσω της συνάρτησης "penalnties" και με τον νέο αριθμό βασιλισσών που τρακάρουν (L_new)

υπολογίζεται η νέα αντικειμενική συνάρτηση μέσω της συνάρτησης "objective function".

Ακολούθως, γίνεται έλεγχος για το αν η νέα fnew είναι μεγαλύτερη ή ίση με την f_optimal. Αυτό για να συμβεί θα πρέπει να έχουμε βρεί το βέλτιστο αποτέλεσμα. Αν συμβεί αυτό, τότε η συνάρτηση σταματάει και επιστρέφει τη βέτιστη τιμή και εκεί τερματίζει ο αλγόριθμος.

Ο δεύτερος έλεγχος που γίνεται στην περίπτωση που η fnew είναι μεγαλύτερη από την ήδη υπάρχουσα λύση, αφορά την αντικατάσταση της αντικειμενικής καθώς και του πλήθους των λύσεων (fitness , nest) με τις νέες καλύτερες λύσεις (fnew, newnest).

Η συνάρτηση get_cuckoos (Lévy flights)

Η συνάρτηση Lévy flights είναι μία τυποποιημένη διαδικασία-υποσυνάρτηση η οποία βρίσκεται στην συνάρτηση του κούκου και υπολογίζει νέες λύσεις μέσω τυχαίων μονοπατιών. Η συγκεκριμένη εφαρμογή του Lévy flights αποτελεί έργο του εμπνευστή του αλγόριθμου του κούκου, X.S. Yang και έχει προγραμματιστεί από τον ίδιο στο πανεπιστήμιο του Cambridge. (Yang, www.mathworks.com)

Είναι μια συνάρτηση που χρησιμοποιεί έτοιμες εξισώσεις όπως η εξίσωση sigma και η gamma και μέσω μίας επαναληπτικής διαδιακασίας υπολογιζει νέες τυχαίες λύσεις και τις επιστρέφει στην βασική συνάρτηση του κούκου.

```
function nest=get_cuckoos(nest,best,Lb,Ub)

n = size(nest, 1);
    beta = 3/2;
    sigma = (gamma(1+beta)*sin(pi*beta/2)/(gamma((1+beta)/2)*beta*2^((beta-1)/2)))^(1/beta);

    for j = 1:n,
        s = nest(j,:);
        u = randn(size(s))*sigma;
        v = randn(size(s));
        step = u./abs(v).^(1/beta);
        stepsize = 0.01*step.*(s-best);
        s = s+stepsize.*randn(size(s));
        nest(j,:) = simplebounds(s,Lb,Ub);
    end
end
```

Η υποσυνάρτηση empty_nests

Η συνάρτηση αυτή , αντικαθιστά ορισμένες όχι τόσο καλές λύσεις δημιουργώντας νέες. Ένα ποσοστό των χειρότερων ανακαλύπτεται με μία πιθανότητα pa=0.25. Μέσω του stepsize ορίζει έναν σταθερό αριθμό που χρησιμοποιείται για τον υπολογισμό των νέων τιμών. Στη συνέχεια μέσω μίας επαναληπτικής διαδιακασίας υπολογιζει νέες τυχαίες λύσεις και τις επιστρέφει στην βασική συνάρτηση του κούκου.

```
function new_nest=empty_nests(nest,Lb,Ub,pa)

    n = size(nest,1);

    K = rand(size(nest))>pa;

    stepsize = rand*(nest(randperm(n),:)-nest(randperm(n),:));
    new_nest = nest+stepsize.*K;
    for j = 1:size(new_nest,1)
        s = new_nest(j,:);
        new_nest(j,:) = simplebounds(s,Lb,Ub);
    end
end
```

Η υποσυνάρτηση simplebounds

Η χρήση της συγκεκριμένης συνάρτησης αφορά τα όρια μέσα στα οποία πρέπει να βρίσκονται οι λύσεις. Εφαρμόζεται στην υποσυνάρτηση "get_cuckoos"

καθώς και στην υποσυνάρτηση "empty_nests" για τον τελικό υπολογισμό των νέων λύσεων.

```
function s=simplebounds(s,Lb,Ub)
    % Apply the lower bound
    ns_tmp=s;
    I=ns_tmp<Lb;
    ns_tmp(I)=Lb(I);

    % Apply the upper bounds
    J=ns_tmp>Ub;
    ns_tmp(J)=Ub(J);

    s=ns_tmp;
end
```

3.4. Συναρτήσεις της Τοπικής Αναζήτησης

Στην περίπτωση που τα αποτελέσματα από τον αλγόριθμο του κούκου δεν είναι ικανοποιητικά, γίνεται χρήση της τοπικής συνάρτησης.

Αρχικά, στην υποσυνάρτηση exchange ελεγχουμε αν και πόσα ζεύγη βασιλισσών τρακάρουν. Αν ο αριθμός αυτός είναι μηδενικός, σημαίνει πως έχουμε το βελτιστο αποτέλεσμα. Σε διαφορετική περίπτωση ακολουθείται η εξής διαδικασία:

Ο πρώτος έλεγχος αφορά την περίπτωση που η βασίλισσα που τρακάρει βρίσκεται στην πρώτη γραμμή της σκακιέρας. Τότε γίνεται έλεγχος και αντιμετάθεση της πρώτης γραμμής με όλες τις υπόλοιπες που βρίσκονται κάτω από αυτή και υπολογισμός των αντικειμενικών συναρτήσεων της κάθε περίπτωσης. Σε περίπτωση που η βασίλισσα που τρακάρει βρίσκεται σε οποιαδήποτε άλλη σειρά ο έλεγχος και η αντιμετάθεση της συγκεκριμένης γραμμής γίνεται με τις γραμμές που βρίσκονται είτε από πάνω είτε από κάτω αυτής.

Στην ουσία, σε κάθε μία των περιπτώσεων, γίνεται ανταλλαγή της γραμμής της βασίλισσας που τρακάρει με κάθε γραμμή της σκακιέρας μέσω της υποσυνάρτησης "exchange_lines". Κάθε φορά κρατάει τις λύσεις (nests) και αντίστοιχα την αντικειμενική τους συνάρτηση (fitnesses).

```
function [nest_new, fnew]=exchange_lines(N, nest, con_queen, i)

    nest_new = nest;

    nest_new(con_queen) = nest(i);
    nest_new(i) = nest(con_queen);

    [L_new, con_queen_tmp] = penalties(N, nest_new);
    fnew = objective_function(N, L_new);

end
```

Έπειτα επιλέγει τη μέγιστη τιμή από τα fitnesses και για αυτή την τιμή κρατά το βέλτιστο nest.

Τέλος, αν οι τιμές που προκύπτουν από την τοπική αναζήτηση είναι καλύτερες από τις λύσεις του κούκου, τότε παρουσιάζονται ως βέλτιστο αποτέλεσμα και τερματιζει ο αλγόριθμος.

3.5. Δημιουργία Σκακιέρας

Στη συγκεκριμένη συνάρτηση δημιουργείται η σκακιέρα πάνω στην οποία τοποθετούνται στην τελική τους θέση οι βασίλισσες. Ανάλογα με τον αριθμό των βασιλισσών, η σκακιέρα χρειάζεται άλλη μέθοδο για να δημιουργηθεί. Για να το πετύχουμε αυτό χρησιμοποιείται η συνθήκη “if mod(N,2)” η οποία ικανοποιείται όταν το υπόλοιπο της διαίρεσης είναι μονάδα, δηλαδή όταν ο αριθμός είναι περιττός. Όταν η συνθήκη ικανοποιείται, δημιουργείται ένας πίνακας με άσσους και στη συνέχεια, μετακινούμενη ανά δύο θέσεις τοποθετούνται μηδενικά στον πίνακα. Στη συνέχεια αντιστοιχίζεται το μηδέν με το μάυρο χρώμα και ο άσσος με το άσπρο, ώστε να δημιουργηθεί η σκακιέρα. Στην περίπτωση που ο αριθμός των βασιλισσών (N) είναι άρτιος η αρίθμηση στην σκακιέρα ξεκινά με μία θέση διαφορά.

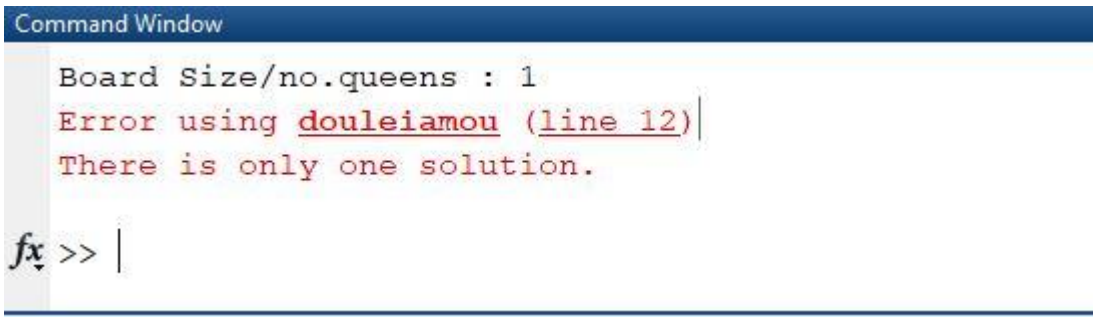
Η συνάρτηση αυτή είναι η τελευταία που καλείται στον κώδικα και δίνει το τελικό αποτέλεσμα σε σχηματική μορφή.

```
if mod(N,2) %Creating an array of 1s and 0s
    a = ones(N);%creating an array with 1s
    a(2:2:N*N) = 0; %every 2nd cell placing  0
else
    a = ones(N+1,N);
    a(2:2:(N+1)*N) = 0;
    a(N+1,:)=[];
end
```

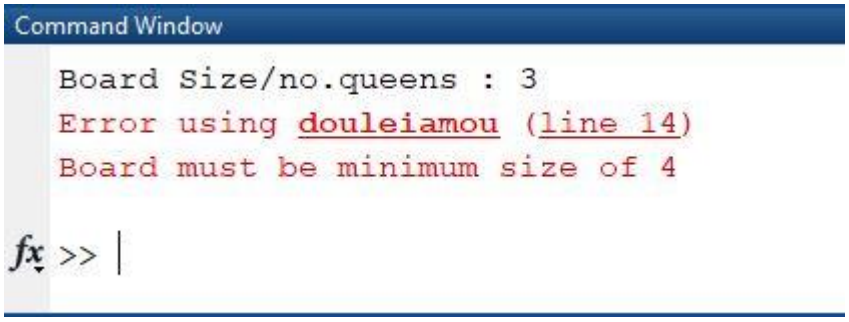
4. ΠΕΙΡΑΜΑΤΑ ΚΑΙ ΑΠΟΤΕΛΕΣΜΑΤΑ

Σε αυτό το κεφάλαιο παρατίθενται τα πειράματα που έγιναν για τυχαίο αριθμό βασιλισσών. Τα πειράματα αυτά έγιναν στην έκδοση 2017a της Matlab και εκτελέστηκαν για αριθμό βασιλισσών N=1 , N=3 και N>=4 .

Στην περίπτωση που ο αριθμός των βασιλισσών είναι μονάδα ο αλγόριθμος βγάζει μήνυμα πως υπάρχει μόνο μία λύση.



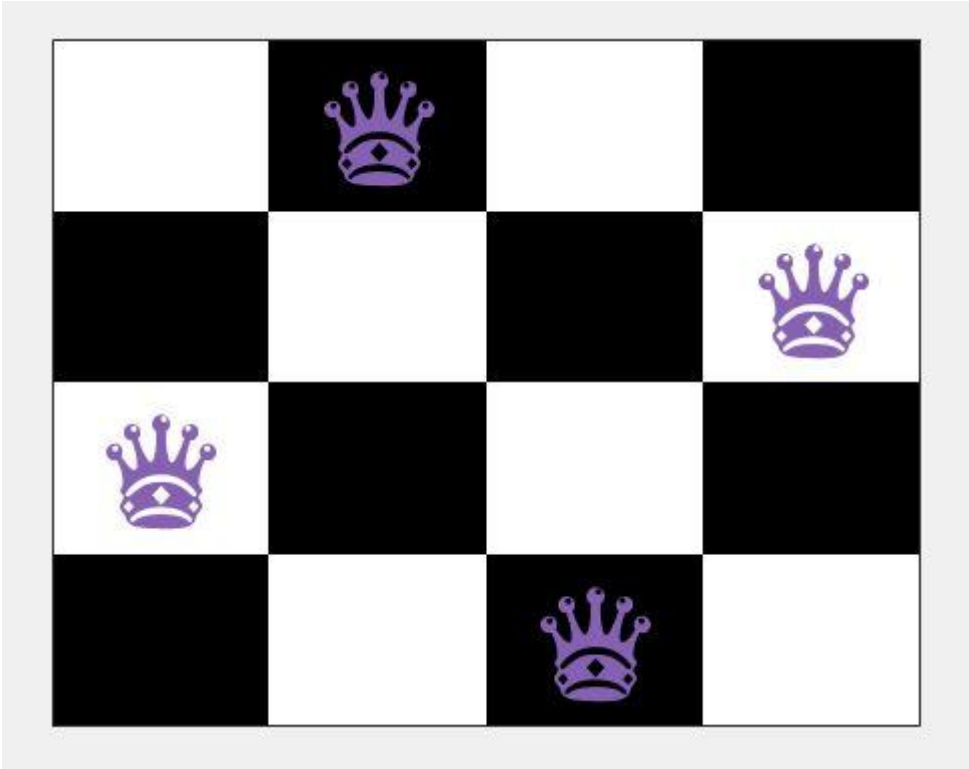
Αντίστοιχα για N=2 και N=3 η λύση είναι αδύνατη.



Για το κάθε πείραμα θα παρουσιάζεται η σκακιέρα καθώς και τα αριθμητικά αποτελέσματα της Matlab, δηλαδή η καλύτερη λύση του Κούκου, η καλύτερη λύση της τοπικής αναζήτησης καθώς και η ιδανική λύση. Ακόμα παρουσιάζεται το διάνυσμα θέσης των βασιλισσών και ο χρόνος που απαιτήθηκε για τις διαδικασίες.

4.1. Πείραμα για N=4

Παρακάτω βλέπουμε μία πιθανή λύση για αριθμό βασιλισσών ίσο με 4.



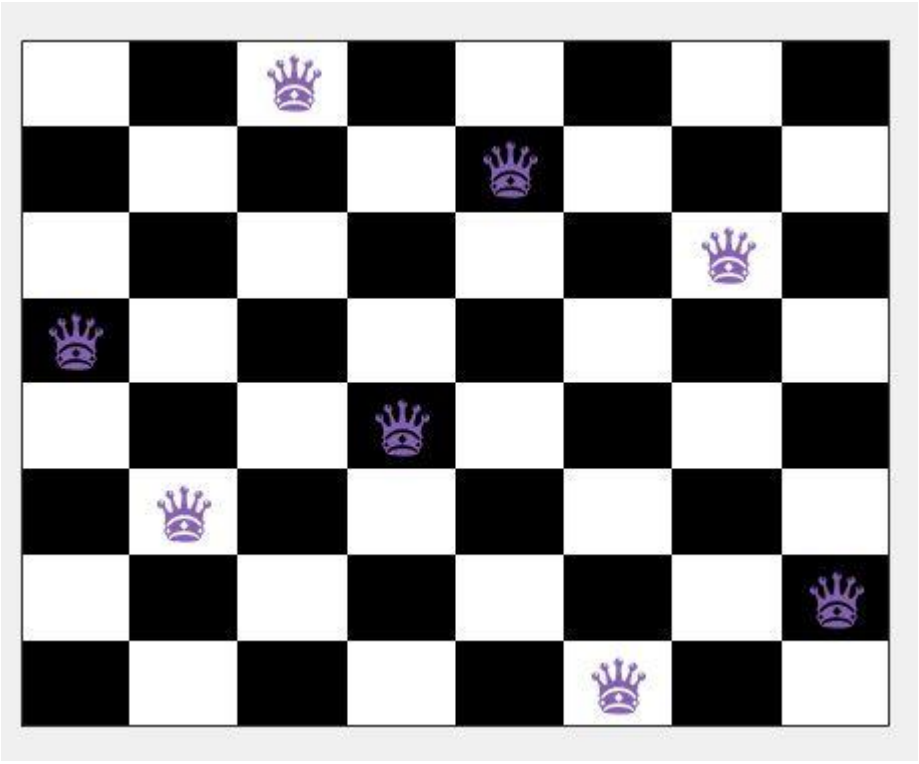
```
Command Window
Board Size/no.queens : 4
The best solution after cuckoos algorithm =2   4   1   3
The fmax after cuckoos algorithm =8
The foptimal after cuckoos algorithm =8
The best solution =2   4   1   3
The fmax =8
The foptimal =8
Elapsed time is 0.130659 seconds.
fx >>
```

Στη συγκεκριμένη περίπτωση ο αλγόριθμος του κούκου έβγαλε βέλτιστο αποτέλεσμα χωρίς να χρησιμοποιηθεί η τοπική αναζήτηση και όπως φαίνεται παραπάνω, η αντικειμενική συνάρτηση που προκύπτει από τον αλγόριθμο αναζήτησης του κούκου συμπίπτει με τη βέλτιστη λύση.

(Όπου βέλτιστη λύση, η τιμή της αντικειμενικής συνάρτησης με μηδενικό αριθμό ζευγαριών βασιλισσών που συγκρούονται).

4.2. Πείραμα για N=8

Παρακάτω βλέπουμε μία πιθανή λύση για αριθμό βασιλισσών ίσο με 8.



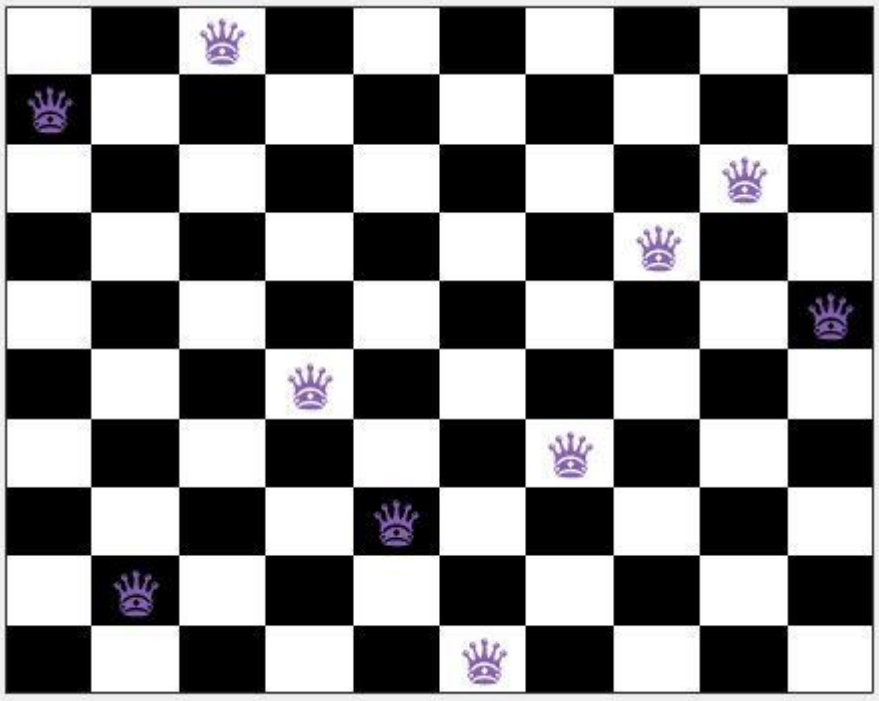
Η σκακιέρα του πειράματος για N=8

```
Command Window
  Board Size/no.queens : 8
  The best solution after cuckoos algorithm =3  5  8  1  4  2  7  6
  The fmax after cuckoos algorithm =31
  The foptimal after cuckoos algorithm =32
  The best solution after local search =3  5  7  1  4  2  8  6
  The fmax after local search =32
  The foptimal after local search =32
  The best solution =3  5  7  1  4  2  8  6
  The fmax =32
  The foptimal =32
  Elapsed time is 0.202607 seconds.
fx >>
```

Σε αυτή την περίπτωση, όπως φαίνεται και στην παραπάνω φωτογραφία, το αποτέλεσμα που βγαίνει από τον αλγόριθμο του κούκου δεν είναι το βέλτιστο. Επομένως καλείται η συνάρτηση τοπικής αναζήτησης και βρίσκει το βέλτιστο αποτέλεσμα. Δηλαδή συμπίπτει η αντικειμενική συνάρτηση που προκύπτει από την τοπική αναζήτηση με την βέλτιστη λύση.

4.3 Πείραμα για N=10

Στο συγκεκριμένο πείραμα ο αριθμός των βασιλισσών είναι ίσος με 10.

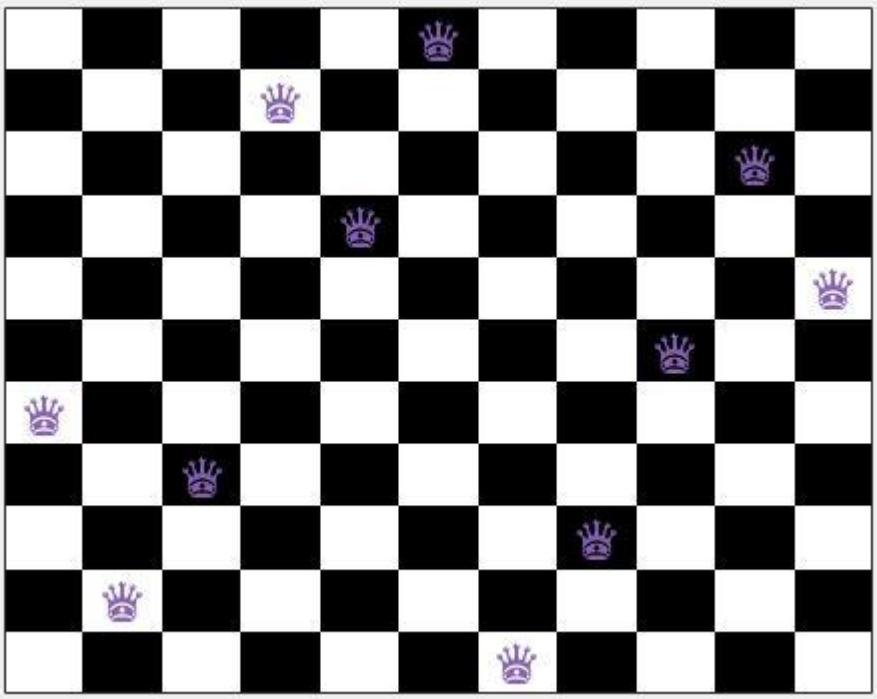


```
Command Window
  Board Size/no.queens : 10
  The best solution after cuckoos algorithm =3   1   9   8  10   4   7   5   2   6
  The fmax after cuckoos algorithm =49
  The foptimal after cuckoos algorithm =50
  The best solution after local search =3   1   9   8  10   4   7   5   2   6
  The fmax after local search =49
  The foptimal after local search =50
  The best solution =3   1   9   8  10   4   7   5   2   6
  The fmax =49
  The foptimal =50
  Elapsed time is 0.198445 seconds.
fx >>
```

Σε αυτό το πείραμα ο αριθμός που προκύπτει από τις διαδικασίες του κούκου δεν είναι ο βέλτιστος. Σε αυτή την περίπτωση καλείται η τοπική αναζήτηση της οποίας η λύση συμπίπτει με αυτή του κούκου. Επομένως δεν φτάνει σε βέλτιστο αποτέλεσμα.

4.4. Πείραμα για N=11

Σε αυτό το πείραμα ο αριθμός των βασιλισσών είναι ίσος με 11.

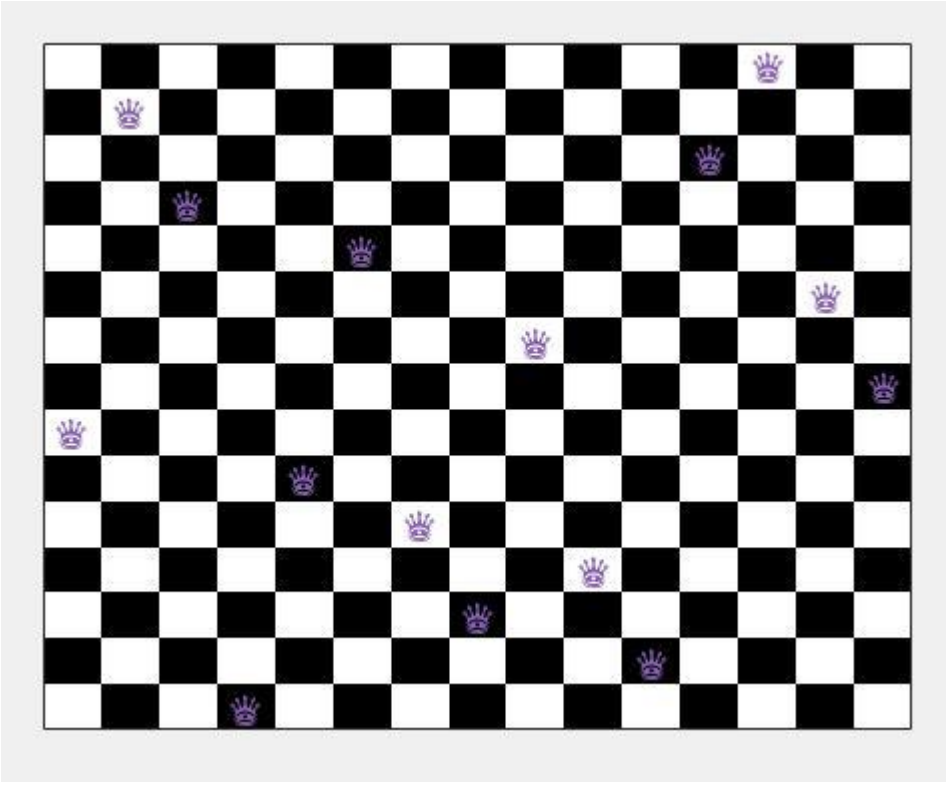


```
Command Window
  Board Size/no.queens : 11
  The best solution after cuckoos algorithm =3   4  10   5  11   9   1   6   8   2   7
  The fmax after cuckoos algorithm =59.5
  The foptimal after cuckoos algorithm =60.5
  The best solution after local search =6   4  10   5  11   9   1   3   8   2   7
  The fmax after local search =60.5
  The foptimal after local search =60.5
  The best solution =6   4  10   5  11   9   1   3   8   2   7
  The fmax =60.5
  The foptimal =60.5
  Elapsed time is 0.222433 seconds.
fx >>
```

Σε αυτή την περίπτωση ο αλγόριθμος του κούκου δεν καταφέρνει να βρει τη βέλτιστη λύση, επομένως καλείται η συνάρτηση exchange της τοπικής αναζήτησης. Μέσω της τοπικής αναζήτησης βρίσκεται η βέλτιστη λύση η οποία συμπίπτει με την βέλτιστη λύση (foptimal) του προβλήματος.

4.5. Πείραμα για N=15

Στο συγκεκριμένο πείραμα γίνεται εφαρμογή του αλγορίθμου για αριθμό βασιλισσών ίσο με 15.

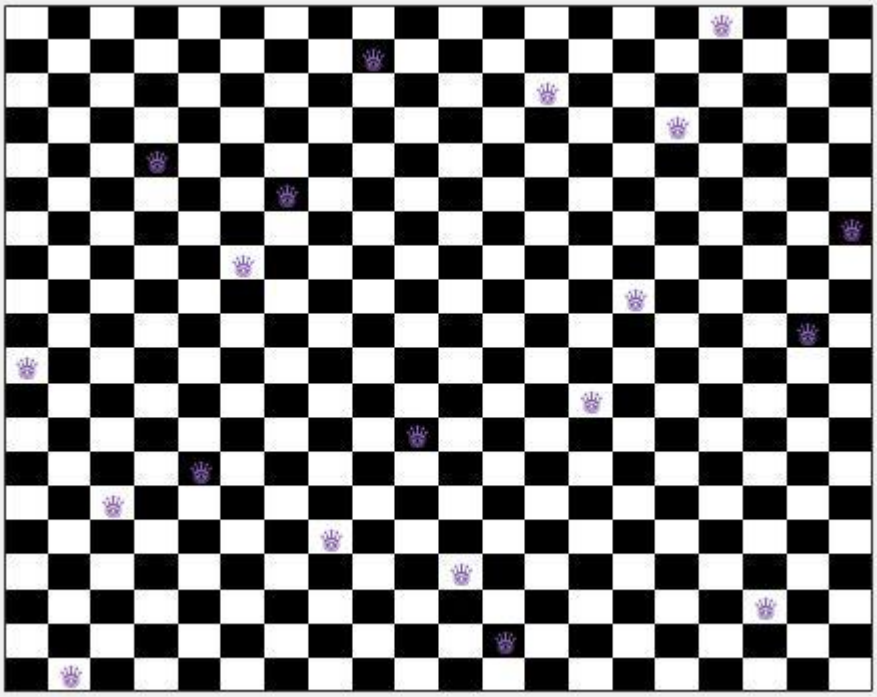


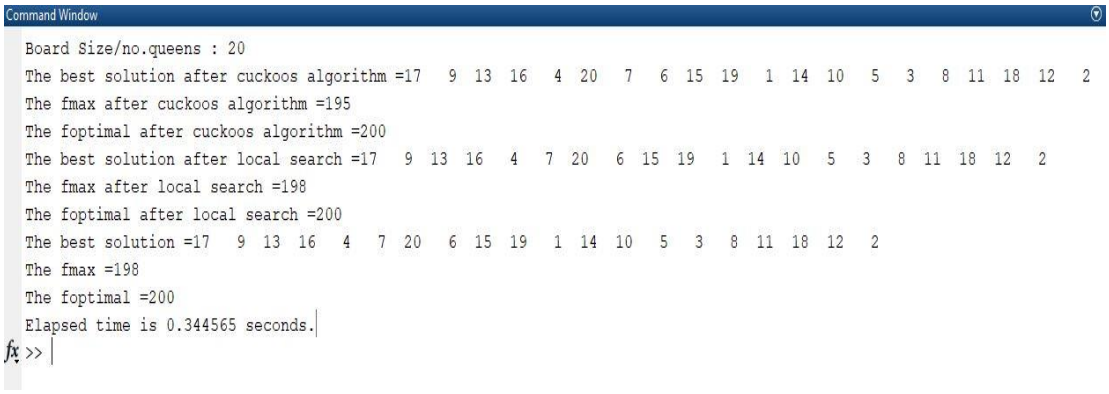
```
Command Window
  Board Size/no.queens : 15
  The best solution after cuckoos algorithm =13   4   6   9   7  14   3  15   1   5  12  10   8  11   2
  The fmax after cuckoos algorithm =108.5
  The foptimal after cuckoos algorithm =112.5
  The best solution after local search =13   2  12   3   6  14   9  15   1   5   7  10   8  11   4
  The fmax after local search =110.5
  The foptimal after local search =112.5
  The best solution =13   2  12   3   6  14   9  15   1   5   7  10   8  11   4
  The fmax =110.5
  The foptimal =112.5
  Elapsed time is 0.246922 seconds.
fx >>
```

Σε αυτό το πείραμα, ο αλγόριθμος του κούκου βρίσκει αρκετά μικρότερη τιμή στην αντικειμενική συνάρτηση από τη βέλτιστη λύση. Γι'αυτό το λόγο γίνεται χρήση της τοπικής αναζήτησης η οποία βρίσκει καλύτερη λύση από τον κούκο, κατά δύο μονάδες. Παρ'όλα αυτά δεν καταφέρνει να βρει τη βέλτιστη λύση. Όμως ως αποτέλεσμα στη λύση του προβλήματος, κρατείται η τιμή της τοπικής αναζήτησης καθώς προσεγγίζει περισσότερο την βέλτιστη λύση.

4.6. Πείραμα για N=20

Σε αυτό το πείραμα γίνεται εφαρμογή του αλγορίθμου για 20 βασίλισσες.

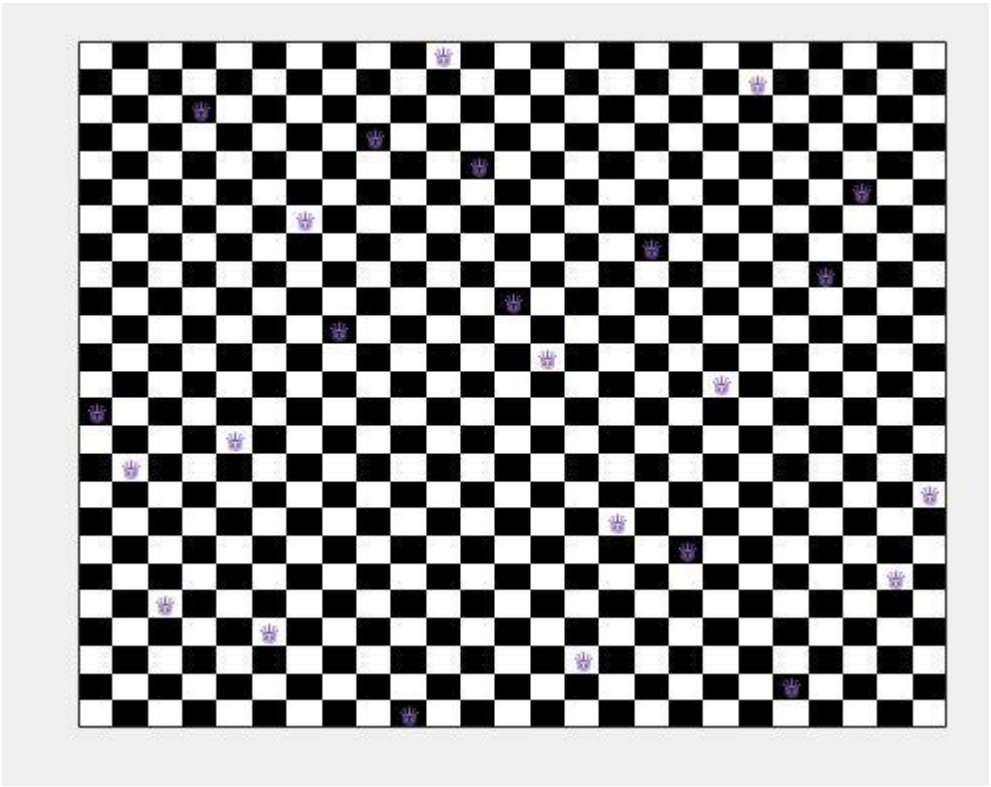




Σε αυτό το πείραμα, ο αλγόριθμος του κούκου βρίσκει αρκετά μικρότερη τιμή στην αντικειμενική συνάρτηση από τη βέλτιστη λύση και γίνεται χρήση της τοπικής αναζήτησης η οποία βρίσκει καλύτερη λύση από τον κούκο, κατά τρεις μονάδες. Παρ'όλα αυτά δεν καταφέρνει να βρει τη βέλτιστη λύση. Όμως ως αποτέλεσμα στη λύση του προβλήματος, κρατείται η τιμή της τοπικής αναζήτησης καθώς προσεγγίζει περισσότερο την βέλτιστη λύση.

4.7.Πείραμα για N=25

Στο συγκεκριμένο πείραμα ο αριθμός των βασιλισσών είναι ίσος με 25.



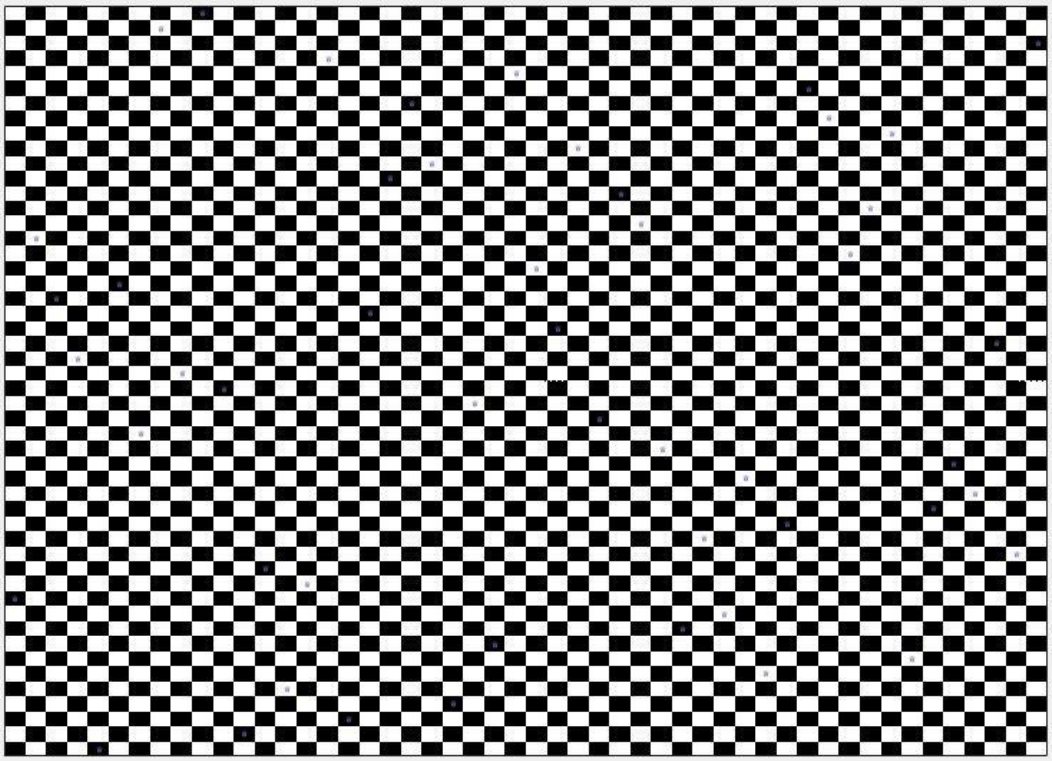
```
Command Window
  Board Size/no.queens : 25
  The best solution after cuckoos algorithm =
  9  17  11  20  12   4   7  25  22  13   8  14  19   1   5   2  23  16  18  24   3   6  15  21  10
  The fmax after cuckoos algorithm =304.5
  The foptimal after cuckoos algorithm =312.5
  The best solution after local search =
  11  20   4   9  12  23   7  17  22  13   8  14  19   1   5   2  25  16  18  24   3   6  15  21  10
  The fmax after local search =311.5
  The foptimal after local search =312.5
  The best solution =
  11  20   4   9  12  23   7  17  22  13   8  14  19   1   5   2  25  16  18  24   3   6  15  21  10
  The fmax =311.5
  The foptimal =312.5
  Elapsed time is 0.481018 seconds.
fx >>
```

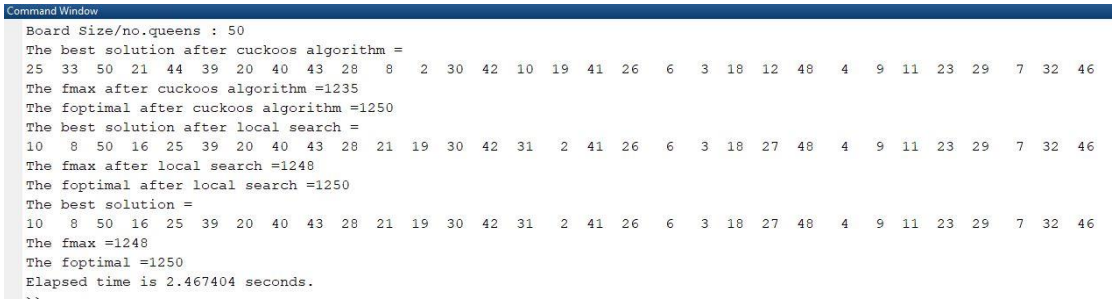
Σε αυτό το πείραμα, ο αλγόριθμος του κούκου βρίσκει μικρότερη τιμή στην αντικειμενική συνάρτηση από τη βέλτιστη λύση και γίνεται χρήση της τοπικής αναζήτησης η οποία βρίσκει καλύτερη λύση από τον κούκο, κατά 7 μονάδες. Παρ'όλα αυτά δεν καταφέρνει να βρει τη βέλτιστη λύση. Όμως ως αποτέλεσμα στη

λύση του προβλήματος, κρατείται η τιμή της τοπικής αναζήτησης καθώς προσεγγίζει περισσότερο την βέλτιστη λύση.

4.8. Πείραμα για N=50

Στο πείραμα που ακολουθεί ο αριθμός των βασιλισσών είναι ίσος με 50. Δυστυχώς, όσο μεγαλώνει ο αριθμός των βασιλισσών, είναι πιο δύσκολο να αποτυπωθει η λύση στη σκακιέρα. Γι' αυτό το λόγο θα παραλειφθεί στα επόμενα πειράματα.

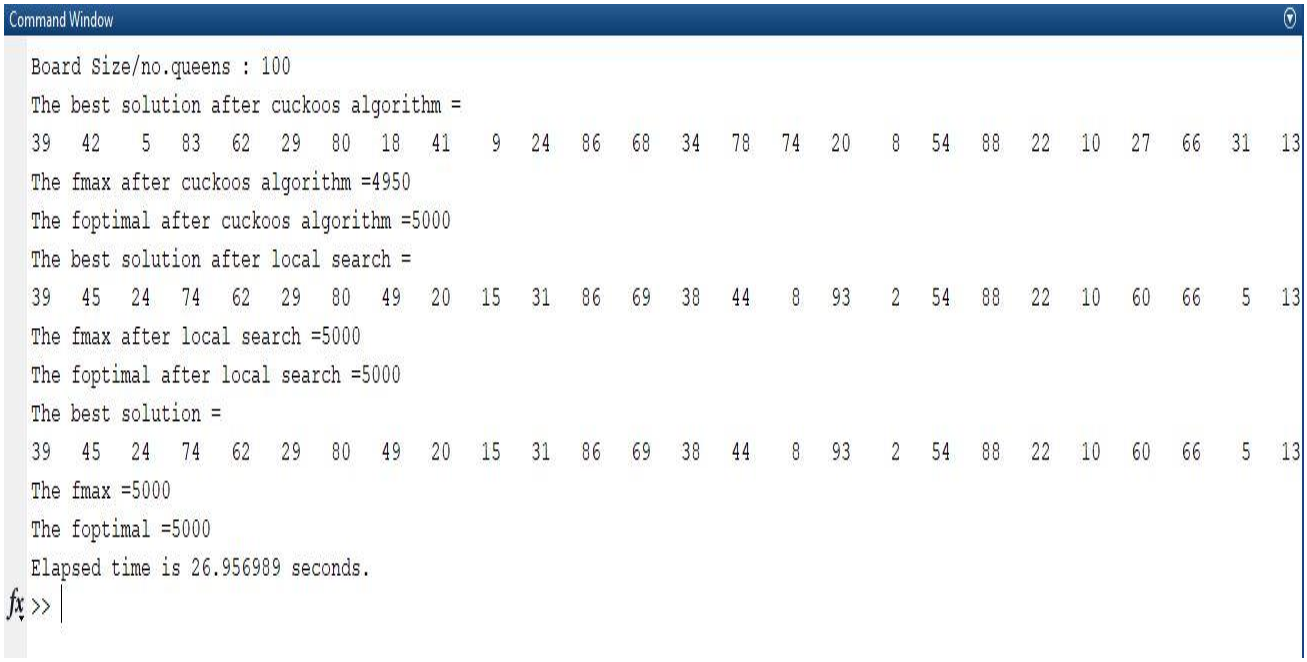


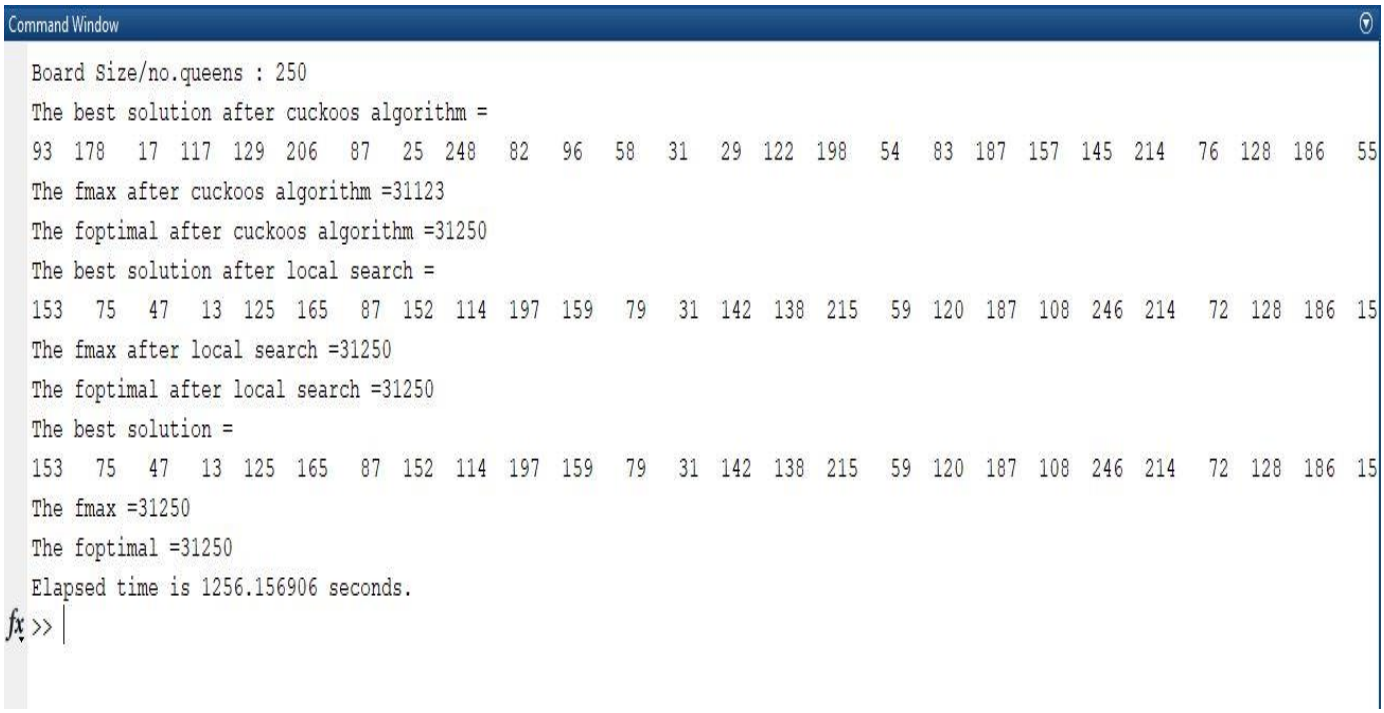


Στο συγκεκριμένο πείραμα ο αλγόριθμος του κούκου δεν καταφέρνει να βρει βέλτιστο αποτέλεσμα και γίνεται χρήση της τοπικής αναζήτησης. Ο αλγόριθμος της τοπικής αναζήτησης καταφέρνει να προσεγγίσει τη βέλτιστη λύση με απόκλιση 2 μονάδων. Για αυτό το λόγο κρατάμε σαν βέλτιστη λύση, την λύση της τοπικής αναζήτησης.

4.9. Πείραμα για N=100 και N=250

Για τα παρακάτω πειράματα, δηλαδή για αριθμό βασιλισσών 100 και 250 αντίστοιχα, οι σκακιέρες είναι δυσδιάκριτες. Επομένως δεν υπήρχε λόγος να συμπεριληφθούν στην συγκεκριμένη διπλωματική και παρατίθενται μόνο τα αριθμητικά αποτελέσματα της MATLAB.





Στην περίπτωση όπου ο αριθμός των βασιλισσών είναι ίσος με 100, ο αλγόριθμος του κούκου βρίσκει μία ικανοποιητική λύση, αλλά όχι τη βέλτιστη. Επομένως γίνεται χρήση της τοπικής αναζήτησης η οποία καταφέρνει να βρεί τη βέλτιστη τιμή.

Ακριβώς το ίδιο παρατηρείται και στην περίπτωση που ο αριθμός των βασιλισσών είναι ίσος με 250. Ο κούκος προσεγγίζει αρκετά τη βέλτιστη τιμή, αλλά δεν καταφέρνει να συμπίπτει με αυτή. Γίνεται χρήση της τοπικής αναζήτησης και μέσω της exchange καταφέρνει να βρει το βέλτιστο αποτέλεσμα στην αντικειμενική συνάρτηση.

Συνοπτικός πίνακας αποτελεσμάτων

N	n	Fmax cuckoo	Fmax local search	F optimal	Elapsed time
4	100	8	X	8	0.130659
8	100	31	32	32	0.202607
10	100	49	49	50	0.198445
11	100	59.5	60.5	60.5	0.222433
15	100	108.5	110.5	112.5	0.246922
20	100	195	198	200	0.344565
25	100	304.5	311.5	312.5	0.481018

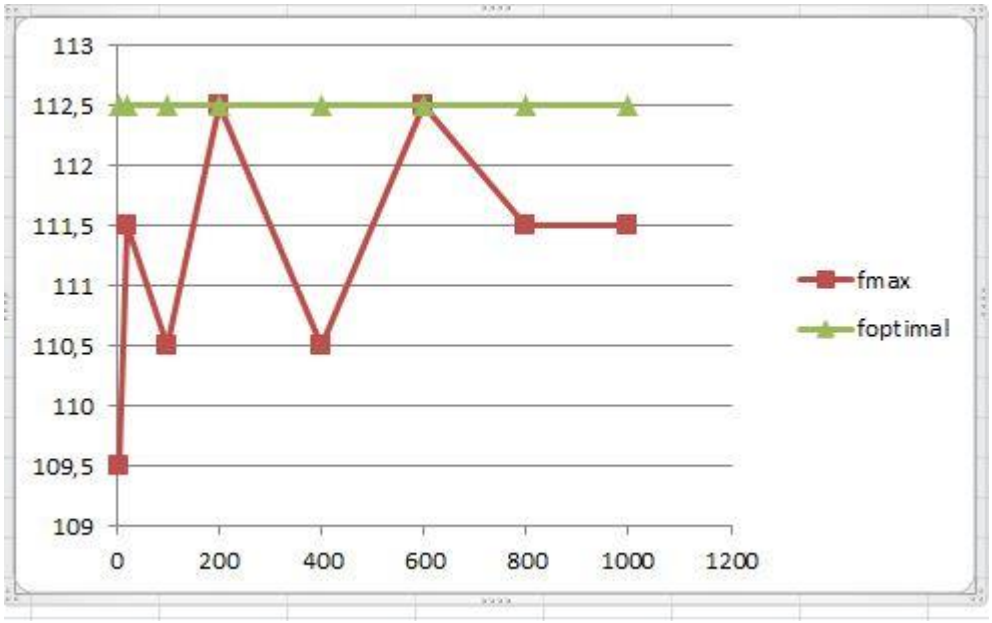
50	100	1235	1248	1250	0.467404
100	100	4950	5000	5000	26.956989
250	100	31123	31250	31250	1256.156906

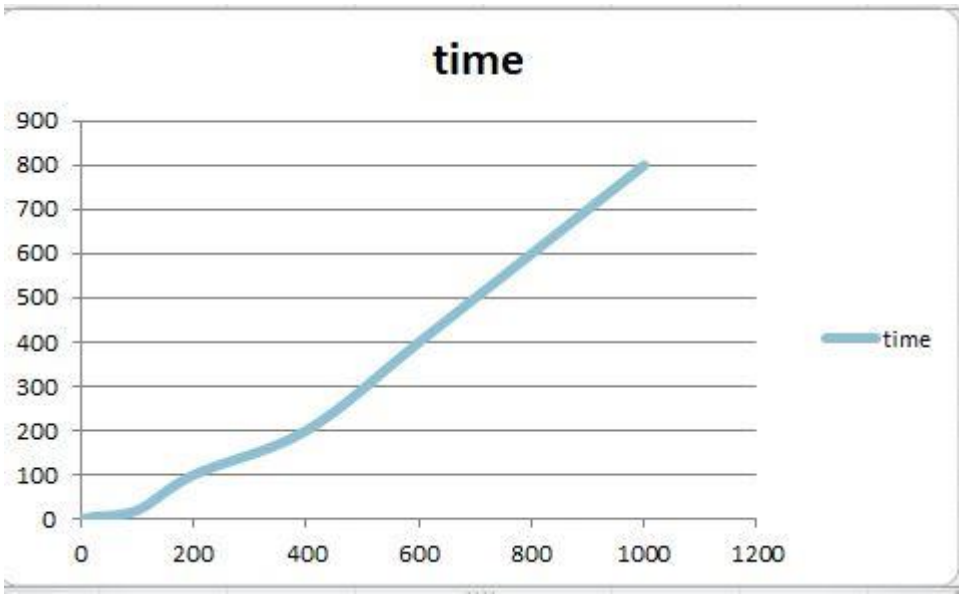
4.10. Πείραμα για συγκεκριμένο αριθμό βασιλισσών (N=15) και διαφορετικό πληθυσμό

Παρακάτω παρουσιάζεται ένα ακόμα πείραμα για σταθερό αριθμό βασιλισσών.

N	n	fmax	foptimal	time
15	5	109,5	112,5	0,207966
15	20	111,5	112,5	0,226994
15	100	110,5	112,5	0,271476
15	200	112,5	112,5	0,411714
15	400	110,5	112,5	0,535485
15	600	112,5	112,5	0,725908
15	800	111,5	112,5	0,98963
15	1000	111,5	112,5	1

Για αριθμό βασιλισσών ίσο με 1 , έγιναν 8 διαφορετικές επαναλήψεις αλλάζοντας κάθε φορά το πλήθος πιθανών λύσεων(population size, n). Οι δοκιμές έγιναν για n=5, n=20, n=100, n=200, n=400, n=600, n=800 και n=1000. Αυτό που παρατηρούμε είναι πως για μεγαλύτερο n αυξάνονται οι πιθανότητες να βρεθει βέλτιστη λύση, ενώ ο χρόνος υλοποίησης του κώδικα αυξάνεται κατά πολύ.





Επομένως συμπεραίνουμε ότι για n=100, που είναι και ο αριθμός που επιλέχθηκε για τα πειράματα, έχουμε ένα μέσο αποτέλεσμα σωστών αποτελεσμάτων σε σύντομο χρονικό διάστημα.

5.ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΗ ΕΡΕΥΝΑ

Σε αυτή τη διπλωματική εργασία επιλύσαμε το πρόβλημα των N βασιλισσών με τη βοήθεια του αλγόριθμου αναζήτησης του κούκου. Μέσω ενός συνόλου μεθόδων όπως η Lévy flights και η μέθοδος 1-1 exchange της τοπικής αναζήτησης προσπαθήσαμε να δημιουργήσουμε τις αρχικές μας λύσεις και να βελτιώσουμε τις λύσεις που προέκυπταν από την εφαρμογή του αλγορίθμου αναζήτησης του κούκου, έτσι ώστε να πετύχουμε όσο το δυνατόν καλύτερες λύσεις.

Το πρόγραμμα στο οποίο εκτελέσαμε τον αλγόριθμο της διπλωματικής εργασίας είναι η MATLAB και συγκεκριμένα η έκδοση 2017a και καταλήξαμε στις βέλτιστες λύσεις μέσω της τοπικής αναζήτησης. Συγκεκριμένα εκτελέσαμε τον αλγόριθμο για διαφορετικό αριθμό βασιλισσών κάθε φορά.

Τα συμπεράσματα που προέκυψαν είναι τα εξής:

- Η τοπική αναζήτηση βελτιστοποιεί σχεδόν κάθε φορά τα αποτελέσματα του κούκου
- Η τοπική αναζήτηση φτάνει σχεδόν πάντα στο βέλτιστο σποτέλεσμα

Συνολικά, για τα δέκα πειράματα που έγιναν, ο κούκος έβγαλε μία φορά βέλτιστο αποτέλεσμα χωρίς να γίνει χρήση της τοπικής αναζήτησης, μία φορά έγινε χρήση της τοπικής αναζήτησης, η οποία ταυτίστηκε με τον κούκο και δεν έφτασε στο βέλτιστο, τέσσερις φορές η τοπική αναζήτηση βελτιστοποίησε το αποτέλεσμα του κούκου αλλά χωρίς να φτάσει το βέλτιστο, ενώ τέσσερις φορές η τοπική αναζήτηση βελτίωσε τα αποτελέσματα του κούκου και έφτασε το βέλτιστο.

Επομένως η χρήση της τοπικής αναζήτησης είναι αυτή η οποία βελτιστοποιεί το αποτέλεσμα.

Παρόλα αυτά τα συγκεκριμένα δεδομένα προκύπτουν από τα παραπάνω παραδείγματα και σε πολλαπλή χρήση του αλγορίθμου τα ποσοστά μπορεί να μεταβάλονται, λόγω της τυχαιότητας που διαίπει το συγκεκριμένο πρόβλημα.

Ως μελλοντική έρευνα θα μπορούσε κανείς να εφαρμόσει άλλους μεθευρετικούς αλγόριθμους βελτιστοποίησης, όπως ο αλγόριθμος της νυχτερίδας, στο πρόβλημα των N βασιλισσών. Ακόμα μπορεί να γίνει εφαρμογή δοαφορετικής μεθόδου τοπικής αναζήτησης, όπως η 1-0 relocate, η οποία θα τοποθετεί την σειρά της σκακιέρας με την βασίλισσα που τρακάρει, ανάμεσα σε δύο άλλες σειρές.

Πηγές

1. Lourenço, H. R., Martin, O. C., & Stützle, T. (2001, Μάρτιος). *Iterated Local Search.* Ανάκτηση από ResearchGate : https://www.researchgate.net/publication/2098230_Iterated_Local_Search

2. Ahrens, W. (1901). *Mathematische Unterhaltungen und Spiele , vol 1, chapter IX.* Leipzig: B.G. Teubner.

3. Al-Khwarizmi, M. (n.d.). *Wikipedia.* Ανάκτηση από https://en.wikipedia.org/wiki/Muhammad_ibn_Musa_al-Khwarizmi

4. De Corte , A., & Sörensen, K. (2016, Αύγουστος). An Iterated Local Search Algorithm for Multi-Period Water Distribution Network Design Optimization. Belgium.

5. Hu, X., Eberhart, R. C., & Shi, Y. (2003, July). *researchgate*. Ανάκτηση από www.researchgate.net: https://www.researchgate.net/publication/2482316_Swarm_Intelligence_for_Permutation_Optimization_Case_Study_of_n-Queens_Problem

6. Oshi, A. S., Kulkarni, O., Kakandikar , G. M., & Nandedkar, V. M. (2017). *Cuckoo Search Optimization- A Review*. Ανάκτηση από ScienceDirect: https://reader.elsevier.com/reader/sd/pii/S2214785317313433?token=7C7361C049D48010061185BCA4845926AF00576D88EDC0DC06654CD9336BC820C9E65C911132C5489AD12FA2EE23A2F9&originRegion=eu-west-1&originCreation=20230207135959

7. Sangita , R., & Chaudhuri, S. S. (2013, Δεκέμβριος). *Cuckoo Search Algorithm using Lèvy Flight: A Review.* Ανάκτηση από https://j.mecs-press.net/ijmecs/ijmecs-v5-n12/IJMECS-V5-N12-2.pdf

8. Wang, Z., Huang, D., Tan, J., Liu, T., Zhao, K., & Li, L. (2015, May). *www.elsevier.com.* Ανάκτηση από elsevier: https://reader.elsevier.com/reader/sd/pii/S0303264715000374?token=C40B7FA1645417882E2D31E27422E8A7F07459D2C66482215EE73682637D69AEBC99C2DD35CFFDAEEB8E01566A62CC87&originRegion=eu-west-1&originCreation=20230206190223

9. Wikipedia. (n.d.). Ανάκτηση από https://en.wikipedia.org/wiki/Eight_queens_puzzle

10. Wikipedia. (n.d.). Ανάκτηση από wikipedia : https://en.wikipedia.org/wiki/L%C3%A9vy_flight

11. Yang, X.-S. (2010). *Nature-Inspired Metaheuristic Algorithms .* University of Cambridge , United Kingdom : Luniver Press.

12. Yang, X.-S. (n.d.). *www.mathworks.com*. Ανάκτηση από https://www.mathworks.com/matlabcentral/fileexchange/29809-cuckoo-search-cs-algorithm

13. Yang, X.-S., & Deb, S. (2009). *Researchgate .* Ανάκτηση από https://www.researchgate.net/publication/45918028_Engineering_Optimisation_by_Cuckoo_Search

14. Αγγελή , Β. (2015). Αλγόριθμος Μεταβλητής γειτονιάς αναζήτησης για το πρόβλημα δρομολόγησης οχημάτων περιορισμένης απόστασης . Χανιά: Πολυτεχνείο Κρήτης .

15. Αδαμίδης , Π. (1999, Μάιος). *Εξελικτικοί αλγόριθμοι.* Ανάκτηση από https://people.iee.ihu.gr/~adamidis/IntelSys/EA_notes.pdf

16. Γονιδάκης , Δ. (2014, Ιανουάριος). Εφαρμογές του αλγόριθμου της νυχτερίδας σε πολυκριτήρια προβλήματα βελτιστοποίησης. Πανεπιστήμιο Πειραιώς.

17. Δελήμπαση, Ε. (2010). Επίλυση του προβλήματος πλανόδιου πωλητή με πολλαπλές αντικειμενικές συναρτήσεις με χρήση του Αλγόριθμου. Χανιά: Πολυτεχνείο Κρήτης.

18. Δούμπος, Μ. (2021). *Πολυτεχνείο Κρήτης .* Ανάκτηση από Πολυτεχνείο Κρήτης , eclass : https://www.eclass.tuc.gr/modules/document/?course=MPD113

19. Κινικλής, Β. (2009, Ιούνιος). Αλγόριθμος επαναληπτικής επανασύνδεσης διαδρομών με χρήση τοπικής αναζήτησης για το πρόβλημα δρομολόγησης οχημάτων. Πολυτεχνείο Κρήτης.

20. Μαρινάκης , Ι. (n.d.). A Hybrid Particle Swarm Optimization Algorithm for Clustering Analysis. Χανιά: Πολυτεχνείο Κρήτης .

21. Μαρινάκης, Ι., & Μυγδαλάς, Α. (2016). *Συνδυαστική Βελτιστοποίηση*. Χανιά: NewTechPub.

22. Μιχάλογλου, Α. (2018, Σεπτέμβριος). Αλγόριθμοι Εξελικτικής Βελτιστοποίησης και εφαρμογές. Θεσσαλονίκη: Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης.

23. Ρηγάκης , Μ. (2016, Απρίλιος). Χρήση Εξελικτικών Αλγορίθμων για την Επίλυση του Διλήμματος του Φυλακισμένου . Χανιά : Πολυτεχνείο Κρήτης .

24. Στρατάκη, Χ. (2010). Μεθευρετικός αλγόριθμος για την επίλυση του σύνθετου προβλήματος χωροθέτησης εγκαταστάσεων και δρομολόγησης οχημάτων . Χανιά : Πολυτεχνείο Κρήτης .

25. Τσαμπουνάρη, Ε. (2017). Αλγόριθμος Βελτιστοποίησης Ζευγαρώματος Μελισσών για το ανοιχτό πρόβλημα Δρομολόγησης Οχημάτων . Χανιά: Πολυτεχνείο Κρήτης .

26. Φραγκογιάννης , Ι. (2018). Βέλτιστος σχεδιασμός γραμμής προϊόντων με χρήση αλγορίθμων εμπνευσμένων από τη φύση. Χανιά: Πολυτεχνείο Κρήτης .