Τεχνολογικό Εκπαιδευτικό Ίδρυμα Κρήτης

Σχολή Τεχνολογικών Εφαρμογών

Τμήμα Εφαρμοσμένης Πληροφορικής και Πολυμέσων

Πτυχιακή Εργασία

Τίτλος: Κάλυψη και Αποφυγή Εντός της Σκακιέρας

Καϊντάρης Ραβάνης Αργύριος (1729)

Επιβλέπων Καθηγητής: Καραγιαννάκης Δημήτριος

Επιτροπή Αξιολόγησης: Καραγιαννάκης Δημήτριος

Ξεζωνάκης Ιωάννης

Μαλάμος Αθανάσιος

Ημερομηνία Παρουσίασης: 7/12/2011

Abstract

One of the first problems proposed to be solved, demonstrating the potential of structured programming, is finding all the solutions of the so-called “8-queens problem”. Generalizing and with the help of our programming computational power we can find solutions for a chessboard with dimension up to 20x20; for a larger size though the execution time is prohibiting. Also, following the logic of a similar approach, we computed the minimum number of queens and other chess pieces that can be placed on a chessboard in such a way so that they “threaten” the whole board throughout.

Σύνοψη

Ένα από τα πρώτα προβλήματα που διατυπώθηκαν προς λύση και που να αποδεικνύουν τις δυνατότητες του δομημένου προγραμματισμού είναι η εύρεση όλων των λύσεων του λεγόμενου “Προβλήματος των οκτώ Βασιλισσών ”. Γενικεύοντας και με την βοήθεια τη υπολογιστικής δύναμης που διαθέτουμε σήμερα μπορούμε να βρούμε λύσεις για διαστάσεις σκακιέρας 20x20, ενώ για μεγαλύτερες διαστάσεις ο χρόνος εκτέλεσης είναι απαγορευτικός. Επίσης μπορούμε με την ίδια λογική να ερευνήσουμε τον ελάχιστο αριθμό των βασιλισσών ή άλλων σκακιστικών κομματιών που τοποθετούνται σε μια σκακιέρα έτσι ώστε να μπορούν να την «καλύπτουν» εξ’ ολοκλήρου.

Πίνακας Περιεχομένων

Πίνακας Εικόνων

Λίστα Πινάκων

1. Εισαγωγή

1.1. Περίληψη

Αρχική ιδέα της εργασίας αυτής ήταν η μελέτη τεχνικών υλοποίησης του λεγόμενου “8 Queens Problem”, το οποίο συνίσταται στην τοποθέτηση 8 βασιλισσών σε μία κλασσική σκακιέρα 8x8, χωρίς η μία να απειλεί (και προφανώς να απειλείται από) κάποια άλλη. Ο απώτερος σκοπός, όμως, ήταν η επίλυση του ίδιου προβλήματος αλλά για n βασίλισσες τοποθετημένες σε μία nxn σκακιέρα.

Αυτή η γενίκευση, όπως και η χρήση άλλων κομματιών του σκακιού αντί της βασίλισσας, μας βοήθησε να αναπτύξουμε γενικευμένους αλγόριθμους ώστε να είναι ικανοί να τελεσφορούν αλλάζοντας μόνο τις αρχικές ρυθμίσεις.

Επίσης σκεπτόμενοι με την ίδια λογική μελετήσαμε την nxn σκακιέρα από την οπτική γωνία της κυριαρχίας (domination) ενός δοσμένου είδους κομματιού, το οποίο συνιστάται στην τοποθέτηση ελαχίστου αριθμού τέτοιων κομματιών ώστε να «καλύπτουν» εξ’ ολοκλήρου την σκακιέρα.

Τέλος θεωρήσαμε σωστό να διαμορφώσουμε ένα οπτικό εργαλείο για την ορθή επισκόπηση και παρουσίαση των αποτελεσμάτων.

1.2. Κίνητρο για την Διεξαγωγή

Ένας λόγος για την ανάληψη της εργασίας αυτής ήταν η περιέργεια και η θέληση να δούμε, οι ίδιοι, τους αλγόριθμους να επιβεβαιώνουν τους αναρτημένους αριθμούς λύσεων, αλλά και τις ίδιες τις λύσεις οι οποίες είναι δύσκολο να αναρτηθούν καθότι είναι πολλές.

Επίσης η ανάπτυξη και μελέτη των αλγορίθμων με την προοπτική της βελτιστοποίησης τους πάνω σε αυτό το «παζλ» μας κέντρισε το ενδιαφέρον. Το ελάχιστο ή domination δεν έχει μελετηθεί καθόλου για την βασίλισσα, καθώς και για τα υπόλοιπα κομμάτια. Οι μόνες αναφορές αφορούν το πλήθος των λύσεων της κάθε περίπτωσης (1), (2), (3), (4). Αυτό επιτυγχάνεται μέσω μαθηματικών τύπων που βάση του n υπολογίζουν το πλήθος των λύσεων.

Επιπλέον δεν υπάρχει καμία προσπάθεια υλοποίησης οποιουδήποτε από τα παραπάνω προβλήματα στην πλατφόρμα του matlab.

Τέλος ένα τέτοιο πρόβλημα θα λυνόταν με κώδικα γραμμένο από το μηδέν, χωρίς ξένα τμήματα, με μόνο οδηγό - σκελετό τις γενικές μεθόδους υλοποίησης των είδους αλγορίθμου. Δεδομένου ότι τρέφω μία ιδιαίτερη αγάπη για τον προγραμματισμό η ιδέα ότι θα εργαζόμουν σε ένα τέτοιο εγχείρημα ήταν αρκετά ελκυστική ώστε να αναλάβω την εργασία.

1.3. Σκοπός και Στόχοι

Να αναπτυχθούν αναδρομικοί ή/και άπληστοι αλγόριθμοι, οι οποίοι να δίνουν όλες τις λύσεις σε κάθε ένα από τα προβλήματα κάλυψης ή/και αποφυγής, για ορισμένο είδος κομματιού και μεγέθους επιπέδου.

Θεωρήθηκε πρέπον για την σωστή παρουσίαση των αποτελεσμάτων, καθώς και για την παρακολούθηση του αλγορίθμου να δημιουργηθεί κατάλληλο εργαλείο οπτικής απεικόνισης της εξόδου του προγράμματος.

Να μελετηθούν ή/και παρουσιαστούν τεχνικές αλλά και εμπόδια τα οποία παρουσιάστηκαν κατά την διάρκεια της υλοποίησης.

Στο τέλος του εγχειρήματος να έχουμε στα χέρια μας κώδικα που να μπορεί να χρησιμοποιηθεί από οποιονδήποτε το επιθυμεί, που σημαίνει ότι πρέπει στην κλήση τους οι αρχικές συναρτήσεις να δέχονται σαφή και τυποποιημένα ορίσματα. Επίσης σε περίπτωση όπου χρειασθεί να αναφερθεί και να ανατρέξει κάποιος στον κώδικα, να είναι διαμορφωμένος έτσι ώστε να είναι ευανάγνωστος και κατανοητός.

1.4.Δομή

i. <u>Μεθοδολογία</u>
   Πλατφόρμα υλοποίησης, Μέθοδοι και τεχνικές που επιλέχθηκαν, Δυσκολίες και πώς αντιμετωπίσθηκαν τεχνικά.
ii. <u>Σχέδιο Δράσης</u>
   State of the Art, Στόχοι, Χρονοδιάγραμμα.
iii. <u>Κύριο Μέρος</u>
   Παρουσίαση των αλγορίθμων.
iv. <u>Αποτελέσματα</u>
   Παρουσίαση των αποτελεσμάτων

2. Μεθοδολογία Υλοποίησης

Ανάπτυξη αναδρομικού αλγορίθμου για τον υπολογισμό των λύσεων του μέγιστου αριθμού κομματιών σε δεδομένες διαστάσεις επιπέδου.

Ως αναδρομικοί αλγόριθμοι ορίζονται εκείνοι οι οποίοι κατά την διάρκεια εκτέλεσής τους καλούν τον εαυτό τους. Εάν κατανοήσουμε την αναδρομή ως έννοια στις θετικές επιστήμες θα είναι πιο εύκολο να αντιληφθούμε και να κατανοήσουμε την δομή ενός αναδρομικού αλγορίθμου. Ένα εύκολο παράδειγμα είναι η εύρεση του παραγοντικού (factorial, μαθηματικό σύμβολο: «!») ενός αριθμού. Ως βάση έχουμε ότι $n_0 = 0! = 1$ και για όλους τους υπόλοιπους αριθμούς ισχύει ότι $n_k = k! = k(n_{k-1}!)$, το οποίο αποτελεί και ένα εύκολο παράδειγμα προς υλοποίηση για την κατανόηση της αναδρομής στον προγραμματισμό.

Κατά την ίδια λογική θα δομήσουμε τον κώδικά μας με τέτοιο τρόπο ώστε να υπολογίζει κάθε λύση ως άθροισμα επί μέρους κλήσεων του ίδιου του του εαυτού. Πιο συγκεκριμένα, θα υλοποιήσουμε έναν backtracking αλγόριθμο, διότι μας προσφέρει την ικανότητα να διερευνήσουμε όλο το πεδίο τιμών για πιθανές λύσεις. Λεπτομερέστερη ανάλυση και επεξήγηση του τρόπου λειτουργίας του αλγορίθμου υπάρχει στο 2.1.1 κεφάλαιο.

Ανάπτυξη άπληστου (greedy) αναδρομικού αλγορίθμου για τον υπολογισμό των λύσεων του ελαχίστου αριθμού κομματιών σε δεδομένες διαστάσεις επιπέδου.

Ο αναδρομικός αλγόριθμος επεξηγήθηκε στο πάνω κείμενο, ο άπληστος είναι μία καινούρια έννοια. Ως άπληστος κατηγοριοποιείται ένας αλγόριθμος όταν ακολουθώντας ένα μονοπάτι επιλογών, κάθε φορά κάνει την καλύτερη δυνατή επιλογή (με βάση δεδομένα κριτήρια που έχουμε ορίσει). Ένα εύκολο παράδειγμα είναι: Να δώσουμε το ποσό των 97 λεπτών του ευρώ σε όσο το δυνατόν λιγότερα κέρματα. Η συνθήκη σε αυτή την περίπτωση είναι να δίνουμε όσα περισσότερα κέρματα μεγαλύτερης αξίας μπορούμε. (0,5)+(0,2)+(0,2)+(0,05)+(0,02)=0,97. (Κάθε παρένθεση αναπαριστά ένα βήμα.)

Ανάπτυξη εργαλείου οπτικής απεικόνισης αποτελεσμάτων χρησιμοποιώντας τους ενσωματωμένους βοηθούς του matlab. Σε αυτό το εργαλείο θα εμφανίζονται επίσης και άλλα πληροφοριακά στοιχεία για την λύση η οποία απεικονίζεται την δεδομένη στιγμή, όπως και κουμπιά πλοήγησης για εύκολη πρόσβαση στις λύσεις.

2.1.Μέθοδος Ανάλυσης και Ανάπτυξης

Χρήση πλατφόρμας τεχνικού υπολογισμού Matlab λόγω της ικανότητας της να χειρίζεται πίνακες. Ακόμη, το πρόβλημα ορίζεται ως υπολογιστικό. Τεχνικά επιλύεται με αλγορίθμους που αποτυπώνουν τις ορθές λύσεις ανάμεσα από το σύνολο όλων των πιθανών αποτελεσμάτων. Το πεδίο ορισμού αποτελεί ένα δένδρο αναζήτησης. Στην περίπτωση μας, στο μέγιστο

είναι n-αδικό, ένώ στο ελάχιστο n^2-αδικό (στην χειρότερη περίπτωση). Στην πραγματικότητα και τα δύο δένδρα είναι n^2 αλλά με ορθούς περιορισμούς απλοποιήσαμε το δένδρο του μεγίστου, βελτιστοποιώντας τον αλγόριθμο.

Ο περιορισμός που τέθηκε αφορά το ότι είναι αδύνατο να μπορέσουμε να τοποθετήσουμε δύο βασίλισσες στην ίδια γραμμή, έτσι αντί σε κάθε επανάληψη να έχουμε να διασχίσουμε $n^2 - k$ κόμβους, έχουμε $n(n-k)$ (k $το$ $βάθος$ $της$ $επανάληψης$). Επιπλέον επιλέγουμε μία βασίλισσα ανά σειρά, μειώνοντας έτσι ακόμα περισσότερο (σε n) τις επαναλήψεις που θα γίνουν Η συγκεκριμένη βελτίωση προφανώς δεν ελαττώνει την ασυμπτωτική πολυπλοκότητα, διότι για μεγάλα n ο όρος n^2 κυριαρχεί, αλλά επιτρέπει την εκτέλεση του προγράμματος σε αισθητά καλύτερους χρόνους για μικρά n.

Το ελάχιστο, δυστυχώς, δεν έχει κάποια σχέση που να ορίζει τον αριθμό των κομματιών που θα χρειαστούν με το μέγεθος της σκακιέρας. Αυτό μας οδηγεί στην ανάπτυξη πιο γενικών εργαλείων υπολογισμού που είναι πιο δυσκίνητα, διότι ερευνούμε μεγαλύτερο δένδρο.

Όσον αφορά την υλοποίηση της εφαρμογής οπτικής παρουσίασης, η εν λόγω πλατφόρμα έχει ενσωματωμένη βιβλιοθήκη ψηφιακής επεξεργασίας εικόνας και οδηγό για δημιουργία γραφικού περιβάλλοντος χρήστη (GUI) τα οποία θα χρησιμοποιήσουμε για την δημιουργία εικόνων-αποτελεσμάτων.

2.1.1. Αλγόριθμοι

Θα χρησιμοποιηθούν δύο τύποι αλγορίθμων, άπληστοι (greedy) και αναδρομικοί backtracking (recursive), μόνοι τους ή σε συνδυασμό.

Οι άπληστοι αλγόριθμοι έχουν την ικανότητα να δίνουν άμεσα αποτελέσματα, με την χρήση ελάχιστου υπολογιστικού χρόνου. Μόνοι τους δεν έχουν την ικανότητα να δώσουν όλες τις λύσεις. Αυτό το κενό το καλύπτουν οι αναδρομικοί αλγόριθμοι με την ικανότητά τους να ανατρέχουν σε όλο το φάσμα των πιθανών συνδυασμών και να σταχυολογούν τις λύσεις.

Συνδυάζοντας αυτές τις τεχνικές έχουμε έναν αναδρομικό αλγόριθμο, που επιλέγει τις λύσεις του βάση άπληστου τρόπου. Πιο συγκεκριμένα στην περίπτωση του μέγιστου θα χρησιμοποιηθεί αποκλειστικά backtracking αλγόριθμος διότι με «ικανοποίηση περιορισμών» (constraint satisfaction) επιτυγχάνουμε βελτιστοποίηση και απλοποίηση παράλληλα. Στο ελάχιστο όμως είναι επιτακτικός ο συνδυασμός και των δύο τεχνικών ώστε να ελέγχει κάθε πιθανός συνδυασμός – διάνυσμα, δεδομένου ότι ο αριθμός «κυριαρχίας» (domination number) δεν έχει σχέση με το n.

Μελετώντας πιο αναλυτικά τους αλγορίθμους θα κατανοήσουμε τον τρόπο λειτουργίας τους. Ο άπληστος αλγόριθμος δίνει πάντα μια υποψήφια λύση, σε όποιο πρόβλημα και να τον χρησιμοποιήσουμε, αρκεί να του έχουμε ορίσει σωστά τα εργαλεία του. Για παράδειγμα αν

στο πρόβλημα με τα κέρματα (βλ. Κεφ. 2) δεν υπήρχε κέρμα των δύο και ενός λεπτών ο αλγόριθμος δεν θα έδινε ποτέ αποτέλεσμα. Αν δεν είχε κέρμα μόνο των δύο λεπτών θα έδινε αποτέλεσμα (δίνοντας ρέστα δύο μονόλεπτα), αλλά δεν θα ήταν η βέλτιστη λύση.

Η δεύτερη περίπτωση αποτελεί λογικό λάθος προγραμματισμού σε μια τέτοια περίπτωση και περιορίζει στην ουσία το πεδίο ορισμού απογυμνώνοντας το από τιμές που θα δίναν βέλτιστα αποτελέσματα. Όμως αυτή η ιδιότητα καθιστά τον αλγόριθμο ευέλικτο, μπορεί κάθε αποτέλεσμα να μην είναι το βέλτιστο, αλλά είναι ορθό. Όπως αναφέρθηκε παραπάνω αυτή η ιδιότητα σε συνδυασμό με τον backtracking αλγόριθμο μας επιτρέπει να σταχυολογούμε όλες τις λύσεις καταγράφοντας μόνο τις βέλτιστες.

Ο κορμός και των δύο προγραμμάτων που θα γράψουμε θα είναι backtracking. Αυτοί οι αλγόριθμοι διατρέχουν ένα δένδρο κατά βάθος (depth first), του οποίου κάθε κόμβος είναι μία εν δυνάμει λύση (ή μέρος της) και αφού ελέγξουν την ορθότητά της, είτε σταματούν (αν είναι λύση) είτε συνεχίζουν - αυτό εξαρτάται και από τον σκοπό του προγράμματος. Στην περίπτωση μας θα συνεχίζει για να προσπελάσει όλες τις λύσεις.

Στην περίπτωση όπου φτάσει σε κάποιο φύλλο ή κλαδί του δένδρου όπου δεν μπορεί να συνεχίσει, επιστρέφει στο προηγούμενο κλαδί από αυτό που ήταν και δοκιμάζει να προσπελάσει το υποδένδρο που (ίσως) κρέμεται από κάποιο αδελφό κλαδί του πρώτου. Σε περίπτωση που η έρευνα είναι ατελέσφορη επιστρέφει κι άλλο ένα κλαδί πίσω και η συγκεκριμένη διαδικασία επαναλαμβάνεται μέχρι να γίνει η προσπέλαση όλου του δένδρου ή να τερματιστεί η έρευνα με κάποια συνθήκη (5).

Επιπλέον σε κάθε κόμβο που επισκέπτεται ο αλγόριθμος ελέγχει αν τα δεδομένα μέχρι την δεδομένη στιγμή είναι ορθά Αν όχι σημαίνει ότι ο κόμβος που μόλις επισκέφθηκε δεν είναι λύση σίγουρα όπως επίσης και κανένα από τα παιδιά. Έτσι κλαδεύοντας (pruning) το δένδρο μειώνουμε τους κόμβους τους οποίους θα προσπελάσουμε, αυξάνοντας την απόκριση του αλγορίθμου ενώ παράλληλα κάνουμε τον αλγόριθμο πιο αξιόπιστο.

2.1.2. Θεωρία

Επιλέγοντας ορθούς περιορισμούς (constraints) μειώνουμε την απαιτούμενη υπολογιστική ισχύ για την επίλυση του προβλήματος. Οι βασίλισσες ,όπως και οι πύργοι απειλούν μία ολόκληρη γραμμή, άρα δεν υπάρχει λόγος να προσπαθήσουμε να τοποθετήσουμε ένα τέτοιο κομμάτι σε μία γραμμή/στήλη αν υπάρχει ήδη κάποιο άλλο. Ακόμη στις περιπτώσεις του αναζήτησης μεγίστου ο προηγούμενος περιορισμός μας επιτάσσει να αναζητήσουμε το πολύ n βασίλισσες σε μία nxn σκακιέρα.

Η πολιτική εύρεσης και ικανοποίησης περιορισμών (constraint satisfaction) είναι άρρηκτα δεμένη με τους backtracking αλγορίθμους καθώς σε κάθε περίπτωση τους βελτιστοποιεί και ως τεχνική είναι αυτή που χαρακτηρίζει έναν αλγόριθμο εύρεσης ως καλύτερο έναντι του χειρότερου δυνατού brute-force. Ένας constraint satisfaction αλγόριθμος έχει ασυμπτωτική πολυπλοκότητα $O(c^n), n > 1$, σε αντίθεση με τον brute-force που έχει $O(n!)$ (5), (6). (Λεπτομέρειες για την ασυμπτωτική πολυπλοκότητα στο Κεφάλαιο 3)

3. Σχέδιο Δράσης

3.1. State of the Art

Οι αλγόριθμοι που θα χρησιμοποιηθούν, στην φιλοσοφία τους είναι απλοί και αποτελούν ένα από τα πρώτα και βασικά πεδία της εκπαίδευσης πάνω στους αλγορίθμους. Η χρήση τους είναι διευρυμένη, όμως λόγω του μεγάλου χρόνου εκτέλεσης τους πάντα γίνεται πάντα προσπάθεια να αποφεύγονται. Αυτό ισχύει για τον αναδρομικό αλγόριθμο, ενώ ο άπληστος είναι ταχύς και αποδοτικός. Βέβαια οφείλουμε να αναφέρουμε ότι οι backtracking αλγόριθμοι έχουν κληρονομήσει την «δυσκινησία» τους από τους brute force αλγόριθμους, τους οποίους σχεδιάστηκαν να αντικαταστήσουν.

Παρόλα αυτά λόγω της ικανότητάς τους να διατρέχουν όλο το δεδομένο σύνολο ορισμού τους και να αποτυπώνουν κάθε λύση που ταιριάζει στα ορίσματα, κάνει αυτούς τους αλγορίθμους χρήσιμους σε τεχνολογίες νέας γενιάς όπως η AI.

Η κατηγοριοποίηση των αλγορίθμων με βάση την ασυμπτωτική τους πολυπλοκότητα είναι ένα μέτρο σύγκρισης μεταξύ τους. Δεδομένου ενός προβλήματος και μίας συγκεκριμένης εισόδου, αν ένας αλγόριθμος χρειάζεται λιγότερο χρόνο από τον άλλο για να φέρει εις πέρας το πρόβλημα, θεωρείτε ασυμπτωτικά ταχύτερος.

Όμως για να έχουμε μέτρο σύγκρισης πιο αντικειμενικό και πιο σαφές, χρησιμοποιούμε μαθηματικές συναρτήσεις για να το εκφράσουμε αυτό. Άλλωστε η ασυμπτωτική πολυπλοκότητα υπήρχε ως έννοια στα μαθηματικά πριν χρησιμοποιηθεί στην επιστήμη των υπολογιστών. Ορίζουμε λοιπόν $O(f(n))$ το άνω όριο αύξησης του χρόνου εκτέλεσης ενός αλγορίθμου, βάση μίας εισόδου n. Έτσι λοιπόν ένας αλγόριθμος $O(n^2)$ είναι πάντα πιο αργός (έχει μεγαλύτερο χρόνο εκτέλεσης) από κάποιον $O(n)$. Να σημειώσουμε ότι αν η παράσταση είναι πολυωνυμική κρατάμε μόνο τον όρο με τον υψηλότερο βαθμό και χωρίς τον συντελεστή του, καθώς οι όροι μικρότερου βαθμού και ο συντελεστής υποσκελίζονται για μεγάλα n λόγω του βαθμού του υψηλότερου όρου (7).

Ενδεικτικά αναφέρουμε ότι ένας βελτιστοποιημένος backtracking έχει πολυπλοκότητα $O(c^n)$, ενώ ένας brute-force $O(n!)$.

Πάνω στον τομέα της προσπάθειας για βελτιστοποίηση των αλγορίθμων έχουν αναπτυχθεί και άλλες τεχνικές, όπως το backjumping κατά αντιστοιχία με το backtracking. Σε αυτή την τεχνική όταν ο αλγόριθμος βρεθεί σε αδιέξοδο επιστρέφει όχι κατά ένα, αλλά κατά δύο ή και περισσότερους κόμβους προς τα πίσω στο δένδρο. Εδώ βέβαια χρειάζονται περισσότεροι περιορισμοί και ορισμοί ώστε αυτή η ανακατεύθυνση να αποκλείει την περίπτωση να υφίστανται λύσεις από και κλαδιά που παρακάμφθηκαν. Αυτός είναι και ο λόγος που η συγκεκριμένη τεχνική κρίθηκε ανεφάρμοστη στον κώδικά μας.

3.2. Κεντρικοί Στόχοι

Έρευνα State of the Art	60
Ανάλυση του προβλήματος	60
Συγγραφή κώδικα μεγίστου αριθμού	60
Αποσφαλμάτωση κώδικα μεγίστου αριθμού	120
Δημιουργία εργαλείου οπτικής απεικόνισης	20
Συγγραφή κώδικα ελαχίστου αριθμού	60
Αποσφαλμάτωση κώδικα ελαχίστου αριθμού	120
Συγγραφή αναφοράς	60
Υποβολή αίτησης αξιολόγησης	1
Προετοιμασία παρουσίασης αναφοράς	5
Παρουσίαση αναφοράς	1

3.2.1. Προτεινόμενο Χρονοδιάγραμμα

Το προτεινόμενο χρονοδιάγραμμα βρίσκετε συνημμένο λόγω μεγάλων διαστάσεων.

4. Κύριο Μέρος

4.1. Ανάλυση Προβλήματος

Σκοπός του όλου εγχειρήματος είναι στο τέλος να έχουμε ένα σύνολο αποτελεσμάτων ικανού όγκου για να βγάλουμε συμπεράσματα για τους αλγορίθμους μας. Με την χρήση κατάλληλου software (Matlab) θα υπολογίσουμε αριθμούς λύσεων καθώς και χρόνους εκτέλεσης.

Θέλουμε έναν αλγόριθμο που να υπολογίζει όλες τις θέσεις-διανύσματα που να επιλύουν το πρόβλημα των n-βασιλισσών και όλα τα διανύσματα που επιλύουν το ελάχιστο. Γενικεύοντας το πρόβλημα θα αναζητήσουμε τα ίδια αποτελέσματα για διάφορα κομμάτια του σκακιού.

4.1.1. Απαιτήσεις Συστήματος

Το συγκεκριμένο λογισμικό επιλέχθηκε διότι αποτελεί μία ολοκληρωμένη υπολογιστική πλατφόρμα, μιας και το πρόβλημά μας ορίζεται ως υπολογιστικό και όχι προγραμματιστικό.
Ακόμη δεδομένης της δομής και της πολυπλοκότητας του αλγορίθμου θα χρειαστούν αρκετές ώρες εκτέλεσης για να βρεθούν αποτελέσματα, ειδικά για μεγάλα n.

4.2. Σχεδιασμός Υλοποίησης

a) Το κομμάτι του μεγίστου αποτελείται από δύο συναρτήσεις, η μία είναι η συνάρτηση αρχικοποίησης και η άλλη η καρδιά του αλγορίθμου.
b) Το μεγαλύτερο κομμάτι του κώδικα δημιουργήθηκε από το ίδιο το Matlab, εμείς απλώς το εμπλουτίσαμε.
c) Το κομμάτι του ελαχίστου αποτελείται από επτά συναρτήσεις.

4.3. Υλοποίηση

4.3.1. Μέγιστο

```
function out=InitQueens(boardsize)
%Ορίσμός και αρχικοποίηση μεταβλητών
%κάποιες είναι global για να είναι προσβάσιμες από όλα τα instances
%(δυστυχώς το Matlab δεν έχει poiters)
clc;
tic;
out.mode='Queen';
global n;
global solution;
global solution_array;
```

```
global column;
global forward_diagonals;
global backward_diagonals;
global queen;
level=1;
n=boardsize;
solution=0;
%Αυτές οι μεταβλητές εξαρτώνται από το μέγεθος της σκακιέρας
solution_array=zeros(n,n,1);
column=zeros(1,n)+n+1;
forward_diagonals=zeros(1,2*n-1)+n+1;
backward_diagonals=zeros(1,2*n-1)+n+1;
queen=zeros(1,n);
%Αρχή αναδρομής
Queens(level);
%Μεταβλητές εξόδου
out.solution_count=solution;
out.solution_array=solution_array;
out.execution_time=toc;
end


function Queens(level)
%Ορίσμός και αρχικοποίηση μεταβλητών
%κάποιες είναι global για να είναι προσβάσιμες από όλα τα instances
%(δυστυχώς το Matlab δεν έχει poiters)
global n;
global solution;
global solution_array;
global column;
global forward_diagonals;
global backward_diagonals;
global queen;
column_index=1;
%Αυτές οι μεταβλητές σε κάθε κλήση - instance έχουν διαφορετική τιμή
forward_diagonal_index=level;
backward_diagonal_index=level+n-1;
%Αν έχει βρεί λύσει την αποτυπώνει, αλλιώς συνεχίζει και προσθέτει
και
%άλλη βασιλισσα
if level == n+1
    solution=solution+1;
    for i = 1:n
        solution_array(i,queen(i),solution)=1;
    end
%Η καρδιά της αναδρομής
else
    for col = 1:n
        %Μέσα στην if γίνεται ο έλεγχος για να αποκλειστεί μια
βασίλισσα
        %να απειλεί μία άλλη
        if (column(column_index)>=level)...
                &&
(forward_diagonals(forward_diagonal_index)>=level)...
                &&
(backward_diagonals(backward_diagonal_index)>=level)
            %Ο κώδικας μεσα στην if εκτελείται μόνο αν υπάρχει ορθή
θέση
            %για βασίλισσα και ορίζει της θέση της βασίλισσας στη
μνήμη
            queen(level)=col;
```

```
            column(column_index)=level;
            forward_diagonals(forward_diagonal_index)=level;
            backward_diagonals(backward_diagonal_index)=level;
            %Αναδρομή
            Queens(level+1);
            %Συνθήκες για σωστό "ένα βήμα πίσω"
            column(column_index)=n+1;
            forward_diagonals(forward_diagonal_index)=n+1;
            backward_diagonals(backward_diagonal_index)=n+1;
        end
    %Συνθήκες για σωστό "ένα βήμα πίσω"
    column_index=column_index+1;
    forward_diagonal_index=forward_diagonal_index+1;
    backward_diagonal_index=backward_diagonal_index-1;
    end
end
end
```

4.3.2. Οπτική Παρουσίαση

```
function myImageViewer_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to myImageViewer (see VARARGIN)

% Choose default command line output for myImageViewer
handles.s=varargin{1};
%Τοποθέτηση των ετικετών
handles.text1=uicontrol('Style','text','Position',[52 105 150
15],'HorizontalAlignment','left');
handles.text2=uicontrol('Style','text','Position',[52 90 150
15],'HorizontalAlignment','left');
handles.text3=uicontrol('Style','text','Position',[252 72 90 40]);
handles.i=1;
handles.array=handles.s.solution_array;
set(handles.text1,'String',horzcat('Solutions:',blanks(1),num2str(han
dles.s.solution_count)));
set(handles.text2,'String',horzcat('Time
Elapsed:',blanks(1),num2str(handles.s.execution_time),' s'));
set(handles.text3,'String',horzcat('Solution ',num2str(handles.i),'
from ',num2str(handles.s.solution_count)));
handles.array_size=size(handles.array,3);
handles.image=BoardSetter(handles.array(:,:,handles.i));
imshow(handles.image)
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);


function pushbutton1_Callback(hObject, eventdata, handles)
handles.i=handles.i-1;
if handles.i < 1
    handles.i=1;
end
```

```
set(handles.text3,'String',horzcat('Solution ',num2str(handles.i),'
from ',num2str(handles.s.solution_count)));
handles.image=BoardSetter(handles.array(:,:,handles.i));
imshow(handles.image)
guidata(hObject, handles);


function pushbutton2_Callback(hObject, eventdata, handles)
handles.i=handles.i+1;
if handles.i > handles.array_size
    handles.i=handles.array_size;
end
set(handles.text3,'String',horzcat('Solution ',num2str(handles.i),'
from ',num2str(handles.s.solution_count)));
handles.image=BoardSetter(handles.array(:,:,handles.i));
imshow(handles.image)
guidata(hObject, handles);


function [index] = BoardSetter(pin)
%Φόρτωση του sprite
load('QueenSymbol.mat');
n=size(pin,1);
%Με βάση τις διαστάσεις της σκακιέρας την δημιουργεί και τοποθετεί
τις
%βασίλισσες στις κατάλληλες συντεταγμένες
for i=1:n
    for j=1:n
        for a=1:32
            for b=1:32
                if pin(i,j)==1
                    if rem(i+j+1,2)==1
                        index(a+32*(i-1),b+32*(j-1))=symbol(a,b);
                    else
                        index(a+32*(i-1),b+32*(j-
1))=rem(symbol(a,b)+1,2);
                    end
                else
                    index(a+32*(i-1),b+32*(j-1))=rem(i+j+1,2);
                end
            end
        end
    end
end
end
```

4.3.3. Ελάχιστο

```
function [ output_args ] = InitMinQueens( boardsize )
%Ορίσμός και αρχικοποίηση μεταβλητών
%κάποιες είναι global για να είναι προσβάσιμες από όλα τα instances
%(δυστυχώς το Matlab δεν έχει poiters)
clc
tic
global out;
out.Solutions=0;
out.Solution_Array=zeros(boardsize,boardsize);
```

```
global variables;
out.Mode='Queen';
%Αυτές οι μεταβλητές εξαρτώνται από το μέγεθος της σκακιέρας
variables.Min_Queens=boardsize+1;
variables.n=boardsize;
%Αρχή αναδρομής
QueenWeights(ones(boardsize));
%Μεταβλητές εξόδου
output_args.mode=out.Mode;
output_args.solution_count=out.Solutions;
output_args.solution_array=out.Solution_Array;
output_args.execution_time=toc;
end


function QueenWeights( position_matrix )
%Ορίσμός και αρχικοποίηση μεταβλητών
%κάποιες είναι global για να είναι προσβάσιμες από όλα τα instances
%(δυστυχώς το Matlab δεν έχει poiters)
global out;
global variables;
%Θέτουμε βάρη, βάση των οποίων θα κάνουμε την επιλογή της βέλτιστης
θέσης
weights=ComputeTotalWeights(position_matrix);
Ma=max(max(weights));
%
%Οι γραμμές 15,16,36 και 52 είναι σε σχόλια διότι αν τις
χρησιμοποιήσουμε,
%ο κώδικας είναι πλήρης και αποτυπώνει όλες τις λύσεις, αλλά δέν
έχουμε την
%απαιτούμενη υπολογιστική ισχύ και το πρόγραμμα κολάει
%
%Mi=min(min(weights));
%diff=Ma-Mi;
%
%Αν έχει βρεί λύσει την αποτυπώνει, αλλιώς συνεχίζει και προσθέτει
και
%άλλη βασίλισσα
%Ελέγχει επίσης αν η καινούρια λύση είναι καλύτερη (μικρότερος
αριθμός
%βασιλισσών)
%Το ".*-1" είναι για λόγους συμαβατότητας με την myImageViewer()
if max(max(weights))==0
    if sum(sum(position_matrix.*-1))==variables.Min_Queens
        out.Solutions=out.Solutions+1;
        out.Solution_Array(:,:,out.Solutions)=position_matrix.*-1;
    end
    if sum(sum(position_matrix.*-1))<variables.Min_Queens
        out.Solutions=1;
        out.Solution_Array(:,:,out.Solutions)=position_matrix.*-1;
        variables.Min_Queens=sum(sum(position_matrix.*-1));
    end
%Η καρδιά της αναδρομής
else
%
%for k=1:(diff+1)
%
    for i=1:variables.n
         for j=1:variables.n
            if weights(i,j)==Ma
                %Ορισμός θέσης καινούριας βασίλισσας
```

```
                position_matrix=SetPosition(i,j,position_matrix);
                %Αναδρομή
                QueenWeights(position_matrix);
                %Πάμε "ένα βήμα πίσω"
                position_matrix=UnSetPosition(i,j,position_matrix);
            end
        end
    end
    Ma=Ma-1;
%
%end
%
end
end


function [ weights ] = ComputeTotalWeights( position_matrix )
%Ορίζμός και αρχικοποίηση μεταβλητών
%κάποιες είναι global για να είναι προσβάσιμες από όλα τα instances
%(δυστυχώς το Matlab δεν έχει poiters)
global variables
%Καλούμε σειριακά την ComputeWeights() ώστε να συγκεντρώσουμε βάρη
για κάθε
%κελί
for i=1:variables.n
    for j=1:variables.n
        weights(i,j)=ComputeWeights(i,j,position_matrix);
    end
end
end


function [ weight ] = ComputeWeights(i,j,position_matrix )
%Ορίσμός και αρχικοποίηση μεταβλητών
%κάποιες είναι global για να είναι προσβάσιμες από όλα τα instances
%(δυστυχώς το Matlab δεν έχει poiters)
%
%Υπολογίζει το βάρος, πόσες καινούριες θέσεις θα απειλούνταν αν
%τοποθετουσαμε μία βασίλισσα στη συγκεκριμένη θέση "i,j"
%
global variables
        weight=0;
        a=j-i;
        b=i+j;
        if position_matrix(i,j)==-1;
         weight=0;
        else
          for k=1:variables.n
               if (position_matrix(i,k)==1)&&(k~=j)
                  weight=weight+1;
               end
          end
         for l=1:variables.n
              if (position_matrix(l,j)==1)&&(l~=i)
                  weight=weight+1;
              end
         end
          for k=1:variables.n
              for l=1:variables.n
                 if (((-
k+l)==a)||((k+l)==b))&&(position_matrix(k,l)==1)
```

```
                    weight=weight+1;
                end
            end
        end
    end
end


function [ threat ] = ComputeThreat( i,j,position_matrix )
%Ορίσμός και αρχικοποίηση μεταβλητών
%κάποιες είναι global για να είναι προσβάσιμες από όλα τα instances
%(δυστυχώς το Matlab δεν έχει poiters)
%
%Ελέγχει αν το συγκεκριμένο κελί "i,j" απειλέιται
%
global variables
a=j-i;
b=i+j;
threat=0;
        for k=1:variables.n
            if (position_matrix(i,k)==-1)&&(k~=j)
                threat=threat+1;
            end
        end
        for l=1:variables.n
            if (position_matrix(l,j)==-1)&&(l~=i)
                threat=threat+1;
            end
        end
        for k=1:variables.n
            for l=1:variables.n
                if (((-
k+l)==a)||((k+l)==b))&&(position_matrix(k,l)==-1)
                    threat=threat+1;
                end
            end
        end
end


function position_matrix = SetPosition( i,j,position_matrix )
%Ορίσμός και αρχικοποίηση μεταβλητών
%
%Με βάση τις συντεταγμένες που ορίζουμε τοποθετεί μία βασίλισσα εκεί
και
%αλλάζει τα δεδομένα που χρειάζονται ώστε να σηματοδοτηθεί
 αυτό
%
global variables
a=j-i;
b=i+j;
for k=1:variables.n
    if (position_matrix(i,k)==1)&&(k~=i)
            position_matrix(i,k)=0;
    end
end
for l=1:variables.n
    if (position_matrix(l,j)==1)&&(l~=j)
            position_matrix(l,j)=0;
    end
end
```

```
for k=1:variables.n
    for l=1:variables.n
        if (((-k+l)==a)||((k+l)==b))&&(position_matrix(k,l)==1)
                position_matrix(k,l)=0;
        end
    end
end
position_matrix(i,j)=-1;
end


function [ position_matrix ] = UnSetPosition( i,j,position_matrix )
%Ορίσμός και αρχικοποίηση μεταβλητών
%κάποιες είναι global για να είναι προσβάσιμες από όλα τα instances
%(δυστυχώς το Matlab δεν έχει poiters)
%
%Με βάση τις συντεταγμένες που ορίζουμε αφαιρεί μία βασίλισσα από
εκεί και
%αλλάζει τα δεδομένα που χρειάζονται ώστε να σηματοδοτηθεί αυτό
%
global variables
position_matrix(i,j)=1;
if ComputeThreat(i,j,position_matrix)~=0
    position_matrix(i,j)=0;
end
for m=1:variables.n
    for n=1:variables.n
        if (position_matrix(m,n)~=-
1)&&(ComputeThreat(m,n,position_matrix)==0)
            position_matrix(m,n)=1;
        end
    end
end
end
```

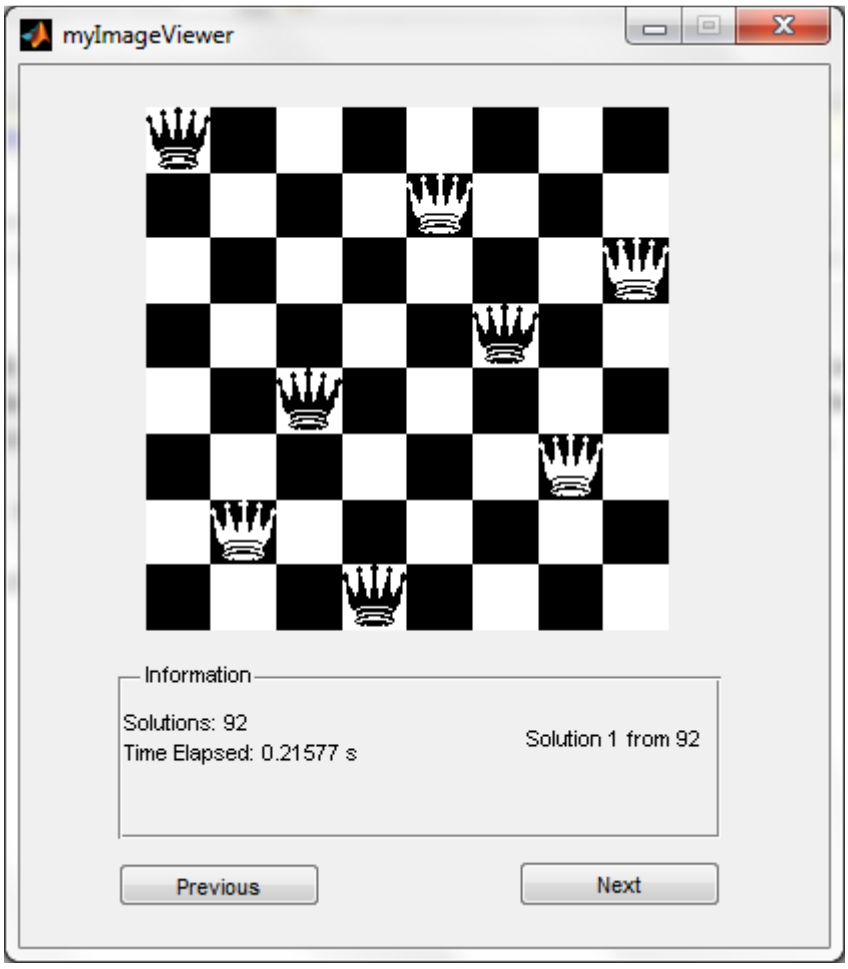
5. Αποτελέσματα

Παραθέτουμε πίνακες και στιγμιότυπα:

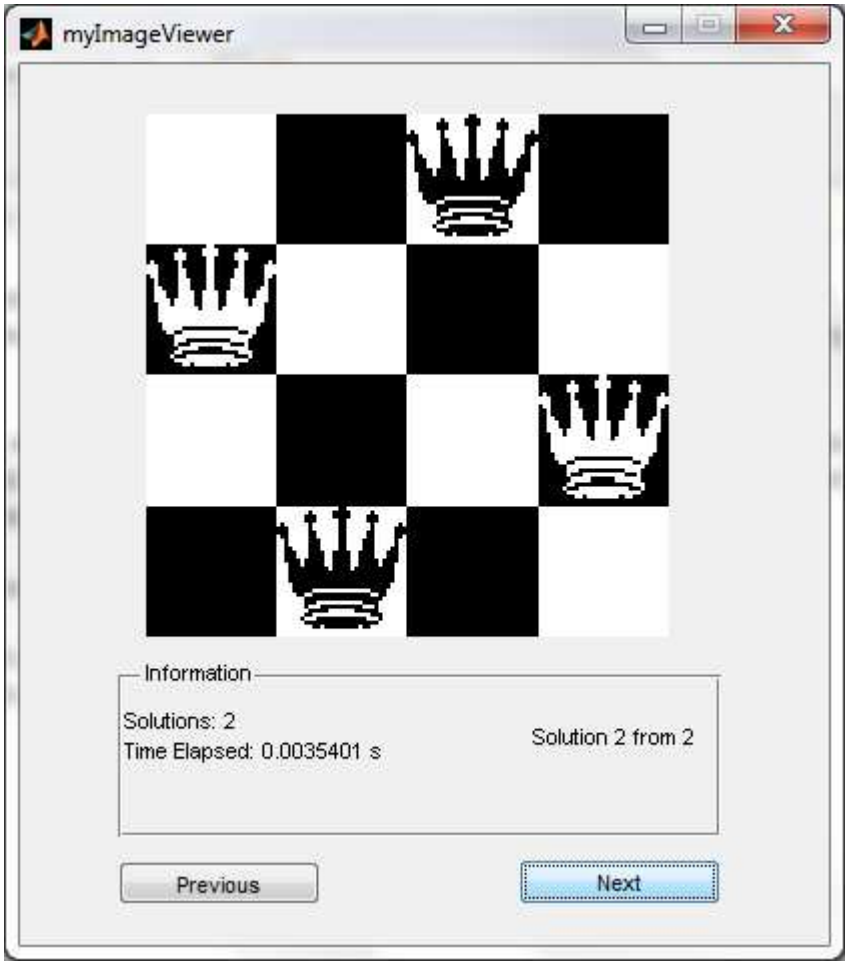
Μέγιστο (όσα περισσότερα κομμάτια μπορούμε να τοποθετήσουμε):

Αριθμός Βασιλισσών	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
Λύσεις	2	10	4	40	92	352	724	2680	14200	73712
Χρόνος	0.0035	0.012	0.017	0.056	0.216	0.912	4.0	21.61	185.4	3350

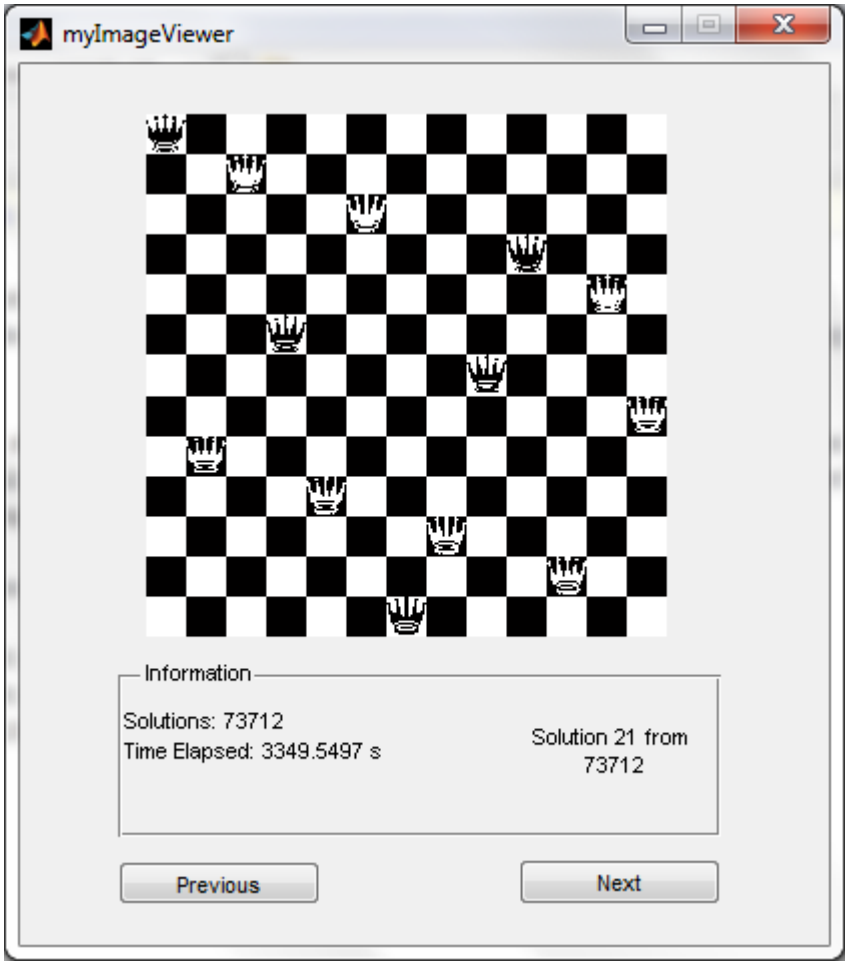
Πίνακας 1 – Αποτελέσματα Μεγίστου



Εικόνα 1 - Πρώτη Λύση για n=8, Μέγιστο



Εικόνα 2 - Δεύτερη Λύση για n=4, Μέγιστο



Εικόνα 3 - Εικοστή Πρώτη Λύση για n=13, Μέγιστο

Αναφέρουμε για την περίπτωση όπου n=8, που είναι η πιο μελετημένη, ότι από τις 92 λύσεις οι 12 είναι μοναδικές και οι υπόλοιπες σχηματίζονται με περιστροφές και αντιμεταθέσεις αυτών των 12. Αντίστοιχα το ίδιο συμβαίνει και για τα άλλα n.

Όσον αφορά τους άλλους τύπους πιονιών, οι πύργοι έχουν n! λύσεις που είναι όλοι οι δυνατοί συνδυασμοί, δεδομένου ότι κάθε στήλη και κάθε γραμμή έχει μόνο ένα πιόνι. Το πλήθος τον λύσεων για τους αξιωματικούς δίνεται από την εξίσωση:

$2^{n-3} + 2^{\left\lfloor \frac{n-1}{2} \right\rfloor - 1}, \lfloor n \rfloor = floor\ function$ (2)

Να σημειώσουμε ότι η κάθε λύση είναι ένας Costas array, και το σύνολο των λύσεων όλοι οι Costas array για το δοσμένο n (8).

Ο αριθμός των μεγίστων αξιωματικών είναι 2n-2.

<u>Ελάχιστο (Όσα λιγότερα κομμάτια μπορούμε να τοποθετήσουμε που να καλύπτουν-απειλούν όλη την σκακιέρα):</u>

Στην συγκεκριμένη περίπτωση αντιμετωπίσαμε ένα πρόβλημα το οποίο οφειλόταν στην ασυμπτωτική πολυπλοκότητα του αλγορίθμου. Συγκεκριμένα, προσπαθώντας να καλύψουμε όλο το εύρος των λύσεων εξαντλούσαμε όλη την μνήμη του υπολογιστή, λόγω αλλεπάλληλων αναδρομών. Χωρίς όμως να αφορούν ατέρμονες βρόγχους. Αυτός ο

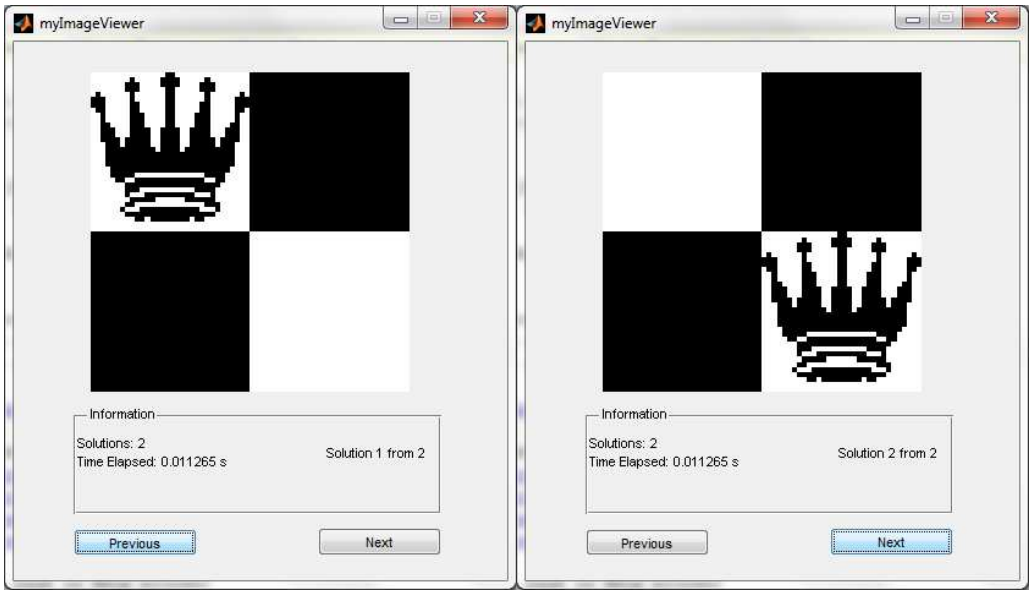
κώδικας, τέσσερις γραμμές παραπάνω, λειτουργεί (και αποδεικνύεται από τα σωστά αποτελέσματα για n=2).

Προσπαθώντας να λύσουμε το πρόβλημα καταλήξαμε στο να περικόψουμε το εύρος των λύσεων ώστε τουλάχιστον να έχουμε κάποια αποτελέσματα.

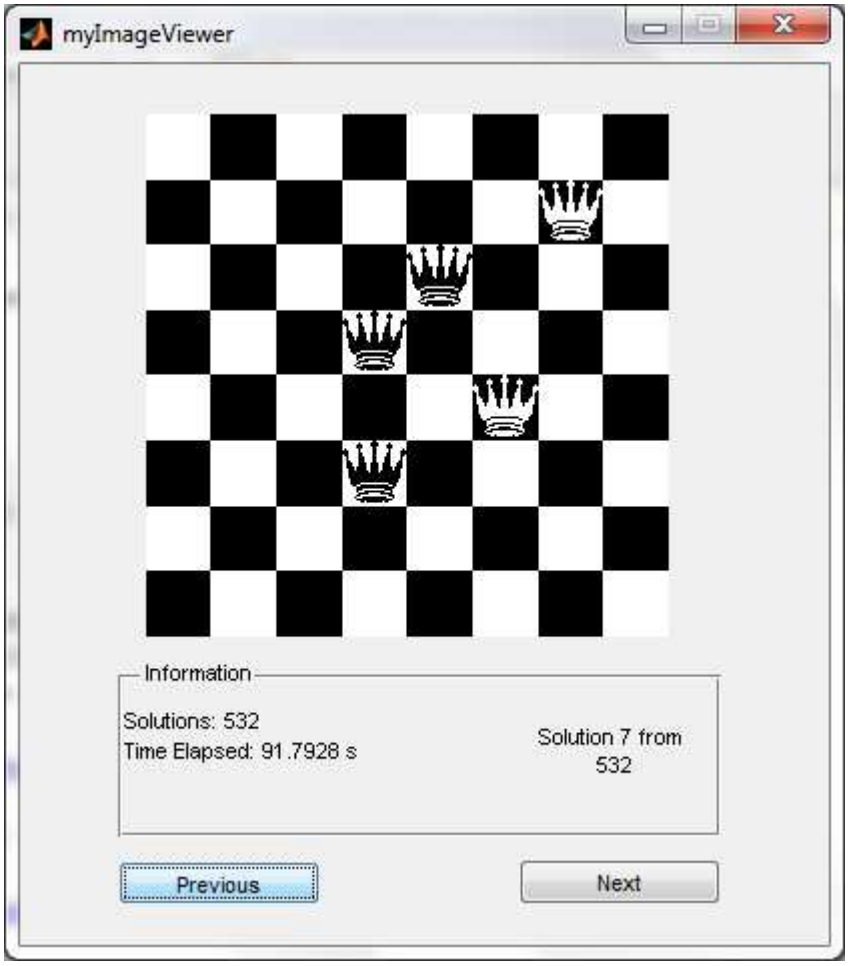
Μέγεθος Σκακιέρας	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Αριθμός Βασιλισσών	1	1	2	3	4	5	5	5	6
Λύσεις	2	1	10	32	496	608	532	64	72
Χρόνος	0.0113	0.0042	0.0324	0.067	1.06	1.63	91.8	40.5	44.6

Μέγεθος Σκακιέρας	**11**	**12**	**13**
Αριθμός Βασιλισσών	7	7	7
Λύσεις	2096	384	128
Χρόνος	158.7	255.5	226.9

Πίνακας 2 – Αποτελέσματα Ελαχίστου Μειωμένου Εύρους



Εικόνα 4 – Οι Δύο Λύσεις του Μειωμένου Φάσματος Κώδικα Ελαχίστου



Εικόνα 5 - Η Έβδομη Λύση του Μειωμένου Ελαχίστου

Ενδεικτικά αναφέρουμε και τα αποτελέσματα για τον πλήρη κώδικα:

Μέγεθος Σκακιέρας	**2**
Αριθμός Βασιλισσών	1
Λύσεις	4
Χρόνος	0.0245

Πίνακας 3 – Αποτελέσματα Ελαχίστου Όλου του Εύρους

5.1.Συμπεράσματα

Για δοθέν n μπορούμε τα τοποθετήσουμε n βασίλισσες, $\frac{n^2}{2}$ ίππους, 2n-2 αξιωματικούς, 2n βασιλείς ή n πύργους.

Στην πραγματικότητα το πραγματικό n του O(n) είναι το n^2 του n όπως το έχουμε ορίσει εμείς. Δηλαδή για σκακιέρα 8x8 εμείς θεωρούμε n=8, για τον υπολογισμό του O(n), όμως, το n είναι 64 διότι ο πίνακας εισόδου έχει 64 στοιχεία (9).

5.2.Μελλοντική Εργασία και Επεκτάσεις

Αξίζει να αναφέρουμε ότι το συγκεκριμένο πρόβλημα-ερώτημα υφίσταται από το 1848, όταν προτάθηκε από τον Max Bezzel. Ο Johann Carl Friedrich Gauss το γενίκευσε στην nxn εκδοχή του, με την οποία ασχοληθήκαμε στην παρούσα εργασία, και οι πρώτες λύσεις ήρθαν το 1850 από τον Nauk. Επίσης οι S. Günther και J.W.L. Glaisher πρότειναν μεθόδους λύσης βασισμένες σε ορίζουσες πινάκων. Τέλος ο Edsger Dijkstra το 1972 έθεσε το πρόβλημα ως **ένα παράδειγμα δομημένου προγραμματισμού με αναδρομή**.

Όπως ανακαλύψαμε στην πορεία ενώ μπορούμε να βρούμε τις λύσεις για το μέγιστο των βασιλισσών, είναι αδύνατο για ένα υπολογιστικό σύστημα γραφείου να υπολογίσει όλες τις «ελάχιστες» λύσεις.

Σε κάθε περίπτωση έχουν βρεθεί μαθηματικοί τύποι που δίνουν το πλήθος των λύσεων.

Το συγκεκριμένο πρόβλημα έχει επεκταθεί από διάφορους μελετητές σε τρείς διαστάσεις, σε nxnxn σκακιέρα δηλαδή (10), σε δακτυλιοειδείς σκακιέρες, όπως και στην τοποθέτηση 9 βασιλισσών και ενός στρατιώτη σε 8x8 σκακιέρα.

Βιβλιογραφία

(1) http://mathworld.wolfram.com/QueensProblem.html
(2) http://mathworld.wolfram.com/RooksProblem.html
(3) http://mathworld.wolfram.com/BishopsProblem.html
(4) Hoffman, et al. "*Construction for the Solutions of the m Queens Problem*". Mathematics Magazine, Vol. XX (1969), p. 66-72.
(5) http://www.cse.ohio-state.edu/~gurari/course/cis680/cis680Ch19.html#QQ1-51-128
(6) Riven, I. and Zabih, R. "An Algebraic Approach to Constraint Satisfaction Problems." In *Proc. Eleventh Internat. Joint Conference on Artificial Intelligence, Vol. 1, August 20-25, 1989.* Detroit, MI: IJCAII, pp. 284-289, 1989.
(7) N. G. de Bruijn (1958). *Asymptotic Methods in Analysis*. Amsterdam: North-Holland. pp. 5–7.
(8) Gilbert, E. N. (1965), "Latin squares which contain no repeated diagrams", *SIAM Review* **7**: 189–198
(9) Bell, Jordan and Stevens, Brett, "A survey of known results and research areas for **n**-queens", Discrete Mathematics, Vol. 309, Issue 1, 6 January 2009, 1-31.
(10) J. Barr and S. Rao, "*The n-Queens Problem in Higher Dimensions*", Elemente der Mathematik, vol 61 (2006), pp. 133–137

Παράρτημα

Η παρουσίαση σε Power Point, καθώς και το χρονοδιάγραμμα είναι συνημμένα σε ξεχωριστά αρχεία.