# N-Queens Problem

Γιακουμάκης Παυλος Πάρις 2013030045
Καδίτης Μανώλης 2014030000
Πορτοκαλάκης Πέτρος 2014030021

30 Μαΐου 2018

**Περίληψη**

Σε αυτήν την εργασία μελετήθηκε το πρόβλημα n-queens. Υλοποιήθηκαν 2 διαφορετικού τύπου αλγόριθμοι για την επίλυση του προβλήματος, καθώς και μελετήθηκαν ως προς την απόδοσή τους.

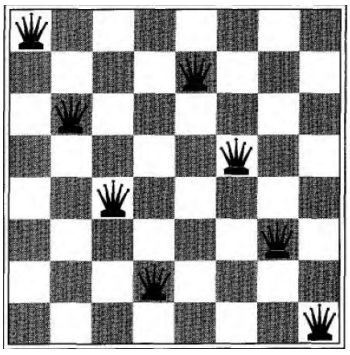
## 1 Περιγραφή του προβλήματος

Ο στόχος του προβλήματος των ν-Βασιλισσών είναι να τοποθετηθούν σε μία ν-επι-ν σκακιέρα με τέτοιο τρόπο ώστε να μην απειλείται καμία βασίλισσα από καμία άλλη. Αν και υπάρχουν αρκετοί ειδικού σκοπού αλγόριθμοι για το συγκεκριμένο πρόβλημα, παραμένει ένας ενδιαφέρον τρόπος να δοκιμάζονται οι αλγόριθμοι αναζήτησης που χρησιμοποιούνται στον τομέα της τεχνητής νοημοσύνης.

## 2 Μοντελοποίηση του προβλήματος

### 2.1 Γενικά

Όπως γίνεται εύκολα αντιληπτό, η αναπαράσταση σε κώδικα και οι πράξεις πάνω σε μία nxn σκακιέρα είναι τα πρώτα πράγματα που επιλύθηκαν. Ορίσαμε την εκάστοτε κατάσταση του προβλήματος ως ένα απλό vector. Σε αυτό το vector, ο i-οστός αριθμός υποδεικνύει την θέση της εκάστοτε βασίλισσας στην i-οστή στήλη της σκακίερας.



Για παράδειγμα, η παραπάνω σκακιέρα στις υλοποιήσεις μας απεικονίζεται ως [0,2,4,6,1,3,5,7]

Ακολουθεί ο κώδικας που αναπτύχθηκε για την εξαγωγή των απειλούμενων βασιλισσών σε οποιαδήποτε κατάσταση του προβλήματος

```
def find_conflicts(arr, N):
    return [hits(arr, N, col, arr[col]) for col in range(N)]


def hits(arr, N, col, row):
    total = 0
    for i in range(N):
        if i == col:
            continue
        if arr[i] == row or abs(i - col) == abs(arr[i] - row):
            total += 1
    return total


def findTotalConflicts(arr):
    return sum(find_conflicts(arr, len(arr)))


print(findTotalConflicts([2, 3, 2, 6, 6, 0, 1, 3]))
```

## 2.2 Μαθηματική Μοντελοποίηση

Η σκακιέρα περιγράφηκε προηγουμένως. Έστω M το μέγεθος του vector, και [M] η λίστα των στοιχείων του. Περιορισμοί:

Δεν γίνεται να υπάρχουν 2 ή περισσότερες βασίλισσες στην ίδια σειρά:

$$\forall i, j \in [M] : M_i \neq M_j$$

Εξόρισμού της λίστας [M], εφόσον είναι vector, δεν γίνεται να υπάρχουν 2 μεταβλητές στην ίδια θέση του vector, άρα συνεπως αυτό μεταφράζεται στο πρόβλημα των βασιλισσών, ότι δεν γίνεται να υπάρχουν 2 βασίλισσες στην ίδια στήλη.

Δεν γίνεται να υπάρχουν 2 ή περισσότερες βασίλισσες στην ίδια διαγώνιο. Για μία βασίλισσα στην θέση a,b δεν πρέπει να ισχύει ότι:

$$\forall i \in 1...M :\mid i - a \mid = \mid M_i - b \mid$$

## 2.3 Μέτρα απόδοσης

Οι πράκτορες που υλοποιήθηκαν αξιολογούνται με βάση την χρονική και χωρική πολυπλοκότητα τους. Επίσης, ένα ακόμα μέτρο αξιολόγησης της απόδοσης είναι αν τελικά κατάφεραν να επιλύσουν το πρόβλημα των N-Queens.. Αυτό το μέτρο απόδοσης αναφέρεται στον αλγόριθμο ελάχιστων συγκρούσεων, ο οποίος υπάρχει

πιθανότητα να μην επιστέψει αποτέλεσμα. Αυτό το μέτρο απόδοσης δεν έχει υλοποιηθεί σε κώδικα, άλλα έχουν γίνει οι απαραίτητες μετρήσεις για να υπάρξει μία αίσθηση της απόδοσης του πράκτορα που χρησιμοποιεί ελάχιστες συγκρούσεις.

# 3 Ανάπτυξη μεθόδων επίλυσης

## 3.1 Μέθοδος τοπικής αναζήτησης

Ως μέθοδος τοπικής αναζήτησης υλοποήθηκε ο αλγόριθμος Min-conflicts. Εδώ αξίζει να αναφερθεί ότι ο αλγόριθμος αυτός υπάγεται και στην κατηγορία των Constraint Satisfaction Problems, αλλά παρ΄όλα αυτά δεν παύει να είναι αλγόριθμος τοπικής αναζήτησης. Η αρχή λειτουργίας του αλγόριθμου είναι η επιλογή τυχαίων μεταβλητών του προβλήματος που παραβιάζουν τους περιορισμούς που έχουν δωθεί και έπειτα η ανάθεση σε αυτές τις μεταβλητές, τιμών που ελαχιστοποιούν τον αριθμό των παραβιάσεων περιορισμών. Συγκεκριμένα, στο πρόβλημα των n-queens ο αλγόριθμος επιλέγει τυχαία μία βασίλισσα από αυτές που απειλούνται από τουλάχιστον άλλη μία βασίλισσα, και ψάχνει να την τοποθετήσει σε σημείο τέτοιο ώστε να ελαχιστοποιούνται οι συγκρούσεις σε αυτήν τη βασίλισσα. Ο αλγόριθμος αυτός γεννά μία έννοια στοχαστικότητας, καθώς αφένός διαλέγει τυχαία τη βασίλισσα που θα ¨πειράξει¨, αλλά και την νέα θέση αυτής της, αν υπάρχουν πάνω από 1 τιμή που ελαχιστοποιεί τις ¨συγκρούσεις-απειλές¨.

## 3.2 Μέθοδος ικανοποίησης περιορισμών

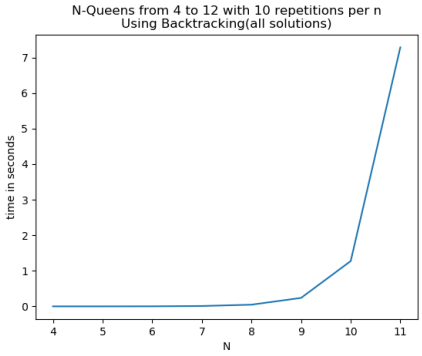
Ως μέθοδος ικανοποίησης περιορισμών υλοποιήθηκε ο αλγόριθμος υπαναχώρησης(Backtracking). Αυτός ο αλγόριμος επιλέχθηκε να υλοποιηθεί καθώς έχει την δυνατότητα να βρει όλες τις λύσεις του προβλήματος που μελετά. Στην δική μας περίπτωση έχουν υλοποιηθεί 2 εκδοχές του αλγόριθμου υπαναχώρησης, μία που βρίσκει όλες τις λύσεις και μία που μόλις βρει την πρώτη λύση σταματάει. Αυτή η δεύτερη εκδοχή έγινε κυρίως για να συγκριθούν οι διαφορές αφένός της εύρεσης όλων των λύσεων σε σχέση με την έυρεση μίας λύσης, και αφέτέρου να συγκριθεί ο αλγόριθμος ελάχιστων συγκρούσεων με τον αλγόριθμο υπαναχώρησης, ως 2 αλγόριθμοι που βρίσκουν μία λύση. Ο αλγόριθμος αυτός εκτελεί πρώτα μία αναζήτηση σε βάθος και επιλέγει νόμιμη θέση για μία βασίλισσα κάθε φορά, έως ότου δεν υπάρχουν άλλες νόμιμες θέσεις που μπορούν να της ανατεθούν. Όταν βρεθεί σε αυτή την κατάσταση επιστρέφει στην προηγούμενη βασίλισσα, μεταφέροντάς τη στην επόμενη νόμιμη θέση. Η διαδικασία αυτή, επαναλαμβάνεται όσες φορές χρειαστεί, έως ότου η τελευταία βασίλισσα βρει νόμιμη θέση.

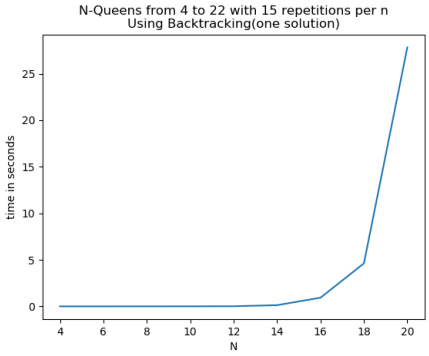
## 3.3 Υλοποίηση-Σύγκριση

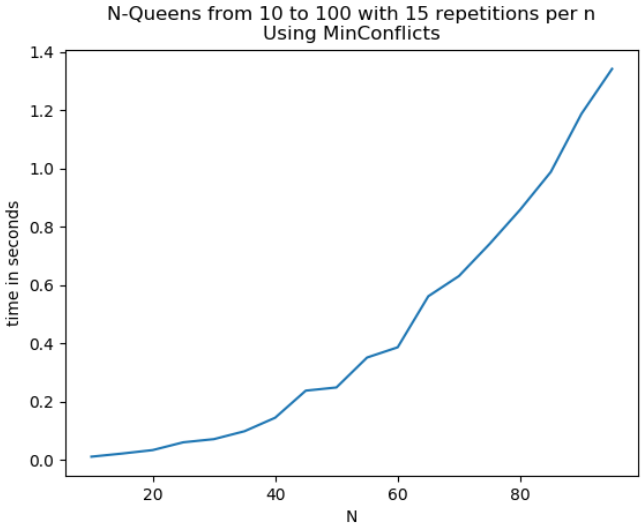
Πριν προχωρήσουμε στις μετρήσεις και τους σχολιασμούς, να αναφερθεί ότι ο αλγόριθμος ελάχιστων συγκρούσεων αναμένεται να είναι πιο αποδοτικός, ωστόσο δεν υπάρχει καμία εγγύηση ότι θα επιστρέψει αποδεκτή λύση. Αντιθέτως, ο αλγόριθμος υπαναχώρησης, αναμένεται να έχει αρκετά μικρότερη απόδοση, αλλά είναι εγγυημένο ότι θα επιστρέψει λύση.

# 4 Πειραματισμός και σύνοψη αποτελεσμάτων

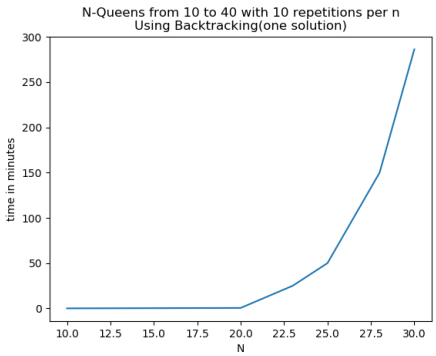
## 4.1 Γραφήματα για γρήγορες εκτελέσεις αλγορίθμων

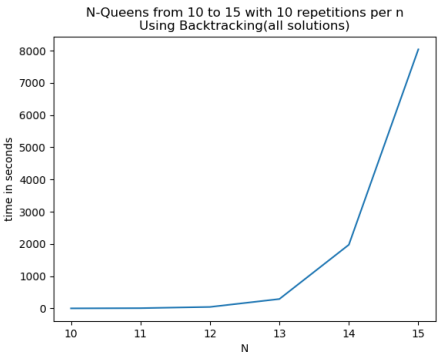


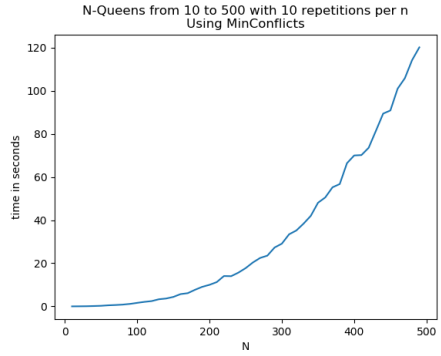



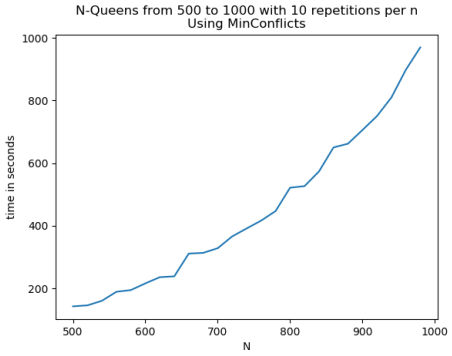




## 4.2 Γραφήματα






## 4.3 Συγκριτικός Πίνακας

Οι χρόνοι, όπου δεν αναφέρεται κάτι διαφορετικό, είναι σε δευτερόλεπτα

| N | Min-Conflicts | Backtracking(one solution) | Backtracking(all solutions) |
|---|---|---|---|
| 4 | insignificant | insignificant | insignificant |
| 6 | insignificant | insignificant | insignificant |
| 8 | insignificant | insignificant | 0.04 |
| 10 | 0.01 | 0.01 | 1.2 |
| 12 | 0.01 | 0.01 | 14 |
| 14 | 0.01 | 0.1 | 33 minutes |
| 16 | 0.01 | 0.8 | 133 minutes |
| 17 | 0.01 | 2.2 | not measured |
| 18 | 0.01 | 4.6 | not measured |
| 20 | 0.12 | 24.2 | not measured |
| 40 | 0.15 | 300 minutes | not measured |
| 60 | 0.4 | not measured | not measured |
| 100 | 1.3 | not measured | not measured |
| 150 | 3 | not measured | not measured |
| 200 | 10 | not measured | not measured |
| 400 | 75 | not measured | not measured |
| 800 | 440 | not measured | not measured |
| 1000 | 980 | not measured | not measured |

## 4.4 Σχολιασμός

Για την μελέτη των 3 αυτών αλγορίθμων θα παρουσιαστούν 2 ομάδες γραφημάτων. Η πρώτη που φαίνεται στην ενότητα 4.1, αφορά λύσεις των n-queens για τέτοια n ώστε να επιστρέφουν οι αλγόριθμοι αποτέλεσμα αρκετά γρήγορα. Παρ΄όλα αυτά, βλέπουμε ακόμα και σε σχετικά αρχικά στάδια των αλγορίθμων, φαίνονται οι τεράστιες διαφορές στην δυναμικότητα των 2 αλγορίθμων. Ενώ ο αλγόριθμος ελάχιστων συγκρούσεων για 100-queens χρειάζεται περίπου 1.35 δευτερόλεπτα, στον ίδιο χρόνο ο αλγόριθμος υπαναχώρησης έχει υπολογίσει όλες τις λύσεις για το πρόβλημα των 10-queens,. Ο αντίστοιχος αλγόριθμος υπαναχώρησης που υπο-

λογίζει μία λύση, σε 1.35 δευτερόλεπτα βγάζει αποτέλεσμα για 13-queens.

Στην ενότητα 4.2 παρουσιάζονται γραφήματα τα οποία δείχνουν την απόδοση των αλγορίθμων όσο το n αυξάνεται. Αξίζει να σχολιαστεί η αξιοσημείωτη απόδοση του αλγορίθμου ελάχιστων συγκρούσεων για μεγέθή σκακίερας της τάξης του 500, που χρειάζεται μόλις 120 δευτερόλεπτα για να βρει λύση. Όσο αυξάνεται το n σε τιμές μεγαλέτερες από 500, έως και την τιμή 1000, ο αλγόριθμος έχει εξαιρετική απόδοση, εξάγοντας το αποτέλεσμα για 1000 βασίλισσες σε κάτι λιγότερο από 17 λεπτά Δυστυχώς λόγω έλλειψης χρόνου, δεν καταφέραμε να φτάσουμε στα όριά του τον αλγόριθμο. Αντιθέτως, ο αλγόριθμος υπαναχώρησης είναι τάξεις μεγέθους πιο αργός, αφού για N = 25 χρειάζεται 50 λεπτά, ενώ για να N = 30, χρειάζεται κοντά στις 3 ώρες υπολογισμών! Συνεπώς, αποδείχτηκε και πειραματικά η εκτίμιση που είχε αναφερθεί στην ενότητα 3.3.

# 5 Παράρτημα

## 5.1 Οδηγίες για το τρέξιμο του κώδικα

Η ανάπτυξη του κώδικα έγινε στην γλώσσα Python v.3.6. Το περιβάλλον ανάπτυξης ήταν το PyCharm COMMUNITY 2018.1. Ο κώδικας είναι πλήρως αυτόνομος, και με ένα πλήρως λειτουργικό μενού, ο χρήστης μπορεί να επιλέξει τον αλγόριθμο που θα μελετήσει, αλλά και διάφορες άλλες παραμέτρους όπως το αρχικό και το τελικό μέγεθος της σκακιέρας, οι επαναλήψεις σε κάθε n αλλά και το βήμα με το οποίο θα προσπερνά ενδιάμεσα μεγέθη σκακιέρας. Το πρόγραμμα μπορεί εύκολα να το τρέξει ο οποιοσδήποτε χρήστης, αρκεί να έχει την Python στον υπολογιστή του, αλλά και κάποιο περιβάλλον σύνταξης κώδικα σε αυτήν. Τα 4 αρχεία πρέπει να είναι στο ίδιο directory . Τρέχει το αρχείο NQueensSolver.py και εμφανίζεται tomenu χρήστη. Από εκεί το menu είναι πλήρως κατατοπιστικό. Ενδέχεται να χρειαστούν μερικά πακέτα της Python v.3.6. που χρησιμοποιήθηκαν κατά την διάρκεια της ανάπτυξης της εργασίας. Μερικά από αυτά είναι τα

```
cycler                  version 0.10.0
kiwisolver              version 1.0.1
matplotlib              version 2.2.2
numpy                   version 1.14.3
pip                     version 9.0.3
pyparsing               version 2.2.0
python-dateutil         version 2.7.3
pytz                    version 2018.4
setuptools              version 39.0.1
six                     version 1.11.0
```

## 5.2 Κώδικας-Υπαναχώρηση(Επιστρέφει όλες τις λύσεις)

```
import copy
import time


def nqueens(N):
```

```
    successorFunction(list(N for i in range(N)), 0, N)


def hits(board, N, col, row):
    total = 0
    for i in range(N):
        if i == col or board[i] == N:
            continue
        if board[i] == row or abs(i - col) == abs(board[i] - row):
            total += 1
    return total


def successorFunction(board, col, N):
    """Use backtracking to find all solutions"""
    global solutions
    # base case
    if col >= N:
        return

    for i in range(N):
        if hits(board, N, col, i) == 0:
            board[col] = i
            if col == N - 1:
                print(board)
                solutions = solutions + 1
                add_solution(board)
                board[col] = N
                return
            successorFunction(board, col + 1, N)
            # backtrack
            board[col] = N


def add_solution(board):
    """Saves the board state to the global variable 'solutions'"""
    saved_board = copy.deepcopy(board)
    allSolutions.append(saved_board)


def backtrackingAllMain(startingPoint, endingPoint, step, repetitions):

    executionTimes = []
    global solutions
    global allSolutions
    allSolutions = []

    for i in range(startingPoint, endingPoint, step):
        totalExecTime = 0
        for j in range(repetitions):
```

```
            solutions = 0
            start_time = time.time()
            nqueens(i)
            execTime = time.time() - start_time
            totalExecTime = totalExecTime + execTime
            print("%d queens has %d solutions found in %s seconds" %
                (i, solutions, execTime))
        executionTimes.append(totalExecTime/repetitions)

    print(executionTimes)
    print(allSolutions)
    return executionTimes
```

### 5.3 Κώδικας-Υπαναχώρηση(Βρίσκει μία λύση και σταματάει)

```
import time


def nqueens(N):
    successorFunction(list(N for i in range(N)), 0, N)


def hits(board, N, col, row):
    total = 0
    for i in range(N):
        if i == col or board[i] == N:
            continue
        if board[i] == row or abs(i - col) == abs(board[i] - row):
            total += 1
    return total


def successorFunction(board, col, N):
    """Use backtracking to find all solutions"""
    # base case
    global flag
    if flag == 0:
        if col >= N:
            return

        for i in range(N):
            if hits(board, N, col, i) == 0:
                board[col] = i
                if col == N - 1:
                    print(board)
                    flag = 1
                    return
                successorFunction(board, col + 1, N)
                # backtrack
```

```
                board[col] = N
    else:
        return


def backtrackingOneMain(startingPoint, endingPoint, step,repetitions):
    global flag
    executionTimes = []
    for i in range(startingPoint, endingPoint, step):
        totalExecTime = 0
        for j in range(repetitions):
            flag = 0
            start_time = time.time()
            nqueens(i)
            execTime = time.time() - start_time
            totalExecTime = totalExecTime + execTime
            print("%d queens solution found in %s seconds" % (i, execTime))
        executionTimes.append(totalExecTime/repetitions)

    print(executionTimes)
    return executionTimes


#backtrackingOneMain()
```

### 5.4 Κώδικας-Κεντρικό Αρχείο

```
import Backtracking
import BacktrackAll
import MinConflicts
import matplotlib.pyplot as plt

if __name__ == '__main__':

    while True:
        print("N-Queens solver")
        print("Choose algorithm to test:")
        print("\t Press 1 for Minimum Conflicts local search algorithm")
        print("\t Press 2 for Backtacking constraint satisfaction algorithm
              (finds all solutions)")
        print("\t Press 3 for Backtacking constraint satisfaction algorithm
              (finds one solutions)")
        answer = int(input("Your answer: "))

        startingPoint = int(input("Enter starting chessboard size: "))
        endingPoint = int(input("Enter ending chessboard size: "))
        step = int(input("Enter step value: "))

        repetitions = int(input("Enter repetitions for each value of n: "))
```

```
if answer == 1:
    execTimeList = MinConflicts.minConflictsMain(startingPoint,
        endingPoint, step, repetitions)
elif answer == 2:
    execTimeList = BacktrackAll.backtrackingAllMain(startingPoint,
        endingPoint, step, repetitions)
elif answer == 3:
    execTimeList = Backtracking.backtrackingOneMain(startingPoint,
        endingPoint, step, repetitions)

plt.plot(list(range(startingPoint, endingPoint, step)),
        execTimeList)
if answer == 1:
    plt.title("N-Queens from %d to %d with %d repetitions per n
        \nUsing MinConflicts" % (startingPoint, endingPoint,
                repetitions))
elif answer == 2:
    plt.title("N-Queens from %d to %d with %d repetitions per n
        \nUsing Backtracking(all solutions)" % (startingPoint,
                endingPoint, repetitions))
elif answer == 3:
    plt.title("N-Queens from %d to %d with %d repetitions per n
        \nUsing Backtracking(one solution)" % (startingPoint,
                endingPoint, repetitions))

plt.xlabel("N")
plt.ylabel("time in seconds")
plt.show()

exitAnswer = int(input('\n\nPress 0 to exit, 1-9 to continue\n\n'))
if exitAnswer == 0:
    break
```

# Αναφορές


[1] Τεχνητή Νοημοσύνη: Μία σύγχρονη προσέγγιση Stuart Russel, Peter Norvig

[2] Min-Conflicts algorithm https://gist.github.com/vedantk/747203

[3] Backtracking algorithm https://www.ploggingdev.com/2016/11/n-queens-solver-in-python-3/