Τεχνητή Νοημοσύνη

Αναφορά Εξαμηνιαίας Προγραμματιστικής Άσκησης

Αποστολος Νικολαος Βαϊλακης
Α.Μ.:2014030174
Γιωργος Διαλεκτακης
Α.Μ.:2014030178
Γιωργος Σταματακης
Α.Μ.:2013030154
Χρηστος Σπυριδακης
Α.Μ.:2014030022

Technical University of Crete

Μέρος Α

1. Το πρόβλημα αναφέρεται στην εύρεση μιας θέσης για n βασίλισσες πάνω σε μια σκακιέρα $n \times n$, έτσι ώστε καμία από τις βασίλισσες να μην επιτίθεται σε καμία άλλη. Δύο βασίλισσες πάνω σε μια σκακιέρα επιτίθενται η μία στην άλλη αν βρίσκονται στην ίδια γραμμή, στήλη ή στην ίδια διαγώνιο. Επομένως θεωρούμε ως δομή δεδομένων έναν μονοδιάστατο πίνακα **board**$[n]$ ο οποίος και αναπαριστά μια σκακιέρα $n \times n$ διαστάσεων. Η τιμή του $board[i]$ ορίζει και την θέση μιας βασίλισσας στη στήλη i της σκακιέρας. Έτσι έχουμε $j = board[i]$ $i, j \in [1, n]$.

   Η συνάρτηση *cost* έχει ως σκοπό την λήψη ενός τέτοιου πίνακα *board* και την εύρεση του αριθμού απειλών ανάμεσα στις βασίλισσες.

2. Για την σωστή αξιολόγηση των πρακτόρων μας θα πρέπει να οριστούν συγκεκριμένα μέτρα απόδοσης. Αυτά είναι:
   α. Χρονική Πολυπλοκότητα
   β. Χωρική Πολυπλοκότητα
   γ. Μετακινήσεις Βασιλισσών

Μέρος Β

1. Για την επίλυση του προβλήματος με την μέθοδο τοπικής αναζήτησης χρησιμοποιήθηκε ο αλγόριθμος **Προσομοιωμένης Ανόπτησης** (Simulated Annealing) ο οποίος αποτελεί ένα συνδυασμό της **Αναρρίχησης Λόφων** (Hill Climbing) με έναν **τυχαίο περίπατο** (Random Walk) με τέτοιο τρόπο ώστε να προσφέρει και αποδοτικότητα και πληρότητα. Αρχικά, παράγουμε τη σκακιέρα τοποθετώντας τις ν βασίλισσες σε τυχαίες θέσεις. Στη συνέχεια, επιλέγουμε τυχαία μια βασίλισσα την οποία αποδεχόμαστε σίγουρα εάν μειώνει τον αριθμό των συγκρούσεων μεταξύ των βασιλισσών. Εάν δεν τον μειώνει δηλαδή δεν φτάνουμε πιο κοντά στον στόχο, τότε η συγκεκριμένη βασίλισσα γίνεται αποδεκτή μόνο με κάποια πιθανότητα μικρότερη του 1. Η πιθανότητα αυτή μειώνεται εκθετικά με το "πόσο κακή" είναι η κίνηση, δηλαδή πόσο μεγαλύτερος είναι ο αριθμός των συγκρούσεων σε σχέση με την προηγούμενη επιλεγμένη βασίλισσα. Η πιθανότητα επίσης μειώνεται καθώς κατεβαίνει η "θερμοκρασία" T. Έτσι, αρχικά είναι πιθανότερο να επιλεχθούν βασίλισσες που προκαλούν περισσότερες συγκρούσεις, ενώ όσο προχωράμε αυτές οι επιλογές "κακών" βασιλισσών γίνονται ολοένα και λιγότερο πιθανές. Η παραπάνω διαδικασία εκτελείται έως ότου ο αλγόριθμος φτάσει στον τελικό στόχο ο οποίος είναι η εύρεση μίας σκακιέρας χωρίς απειλούμενες βασίλισσες. Ο αλγόριθμος καταγράφει τον χρόνο που χρειάστηκε μέχρι να φτάσει στο στόχο καθώς και τον αριθμό των κινήσεων που απαιτήθηκαν.

   Όσον αφορά την **Successor function** του συγκεκριμένου αλγορίθμου, υλοποιήθηκε μία συνάρτηση η οποία λαμβάνει ως είσοδο τον αριθμό των βασιλισσών του προβλήματος που επιλύουμε και την τρέχουσα κατάσταση που είναι η μέχρι τώρα διάταξη τους πάνω στο board και επιστρέφει τις βασίλισσες που μπορούν να επιλεχθούν στο επόμενο βήμα. Στην περίπτωσή μας επιλέγεται πάντα μία τυχαία βασίλισσα από τη σκακιέρα η οποία στη συνέχεια θα γίνει αποδεκτή από τον αλγόριθμο με τη διαδικασία που περιγράφηκε παραπάνω.

   Να σημειωθεί ότι αρχικά, για την επίλυση του προβλήματος με τοπική αναζήτηση υλοποιήσαμε τον αλγόριθμο της **Αναρρίχησης λόφων** (Hill Climbing). Ωστόσο, στη συνέχεια παρατηρήσαμε μέσα από πειράματα ότι η απόδοση του είναι εξαιρετικά κακή και οδηγεί σε απαγορευτικούς χρόνους. Έτσι λοιπόν προχωρήσαμε στη επιλογή και υλοποίηση του αλγορίθμου **Simulated Annealing**, ο οποίος όπως θα δείξουμε παρουσιάζει πολύ καλύτερα αποτελέσματα.

Pseudo-code για το successor_function του SA

Algorithm 1 SA Successor_function

1: **Input:**
   1. A Board where the Queens are.
   2. Size of Board.
2: **Output:**
   1. State of conflicts existing there.
   2. Number of conflicts existing there.

state = Chose a random neighbour to jump to
cost = Counts the number of conflicts across the board for new state
Return state and cost

2. Για την επίλυση του προβλήματος με την χρήση μιας μεθόδου ικανοποίησης περιορισμών χρησιμοποιήθηκε ο **Αλγόριθμος Ελαχίστων Συγκρούσεων** (MinConflicts). Όντας αρκετά αποτελεσματικός και γρήγορος αλγόριθμος (αν και όχι πλήρης) λύνει το πρόβλημα με σημαντικό χρονικό πλεονέκτημα. Έχοντας υλοποιήσει τις μεθόδους check_conflicts() και successor_function() η σκακιέρα αρχικοποιείται με βασίλισσες τοποθετώντας κάθε φορά μια βασίλισσα σε σημείο όπου προκύπτουν ελάχιστες συγκρούσεις. Έπειτα ο αλγόριθμος λύνει το πρόβλημα επιλέγοντας διαδοχικά τυχαίες βασίλισσες και βρίσκοντας τις επόμενες πιθανές κινήσεις τους χρησιμοποιώντας την successor_function(). Στην περίπτωση που πάνω από μια κινήσεις επιστρέφονται από την successor_function(), τότε μια κίνηση από αυτές επιλέγεται τυχαία. Ο αλγόριθμος συνεχίζει έως ότου βρεθεί μια λύση καταγράφοντας τον αριθμό μετακινήσεων που χρειάστηκε για να λύσει το πρόβλημα. Η successor_function() είναι σχεδιασμένη ώστε να επιστρέφει τις πιθανές κινήσεις μιας βασίλισσας με το μικρότερο κόστος ελέγχοντας τις συγκρούσεις ανάμεσα στις βασίλισσες μετά από κάθε πιθανή κίνηση.

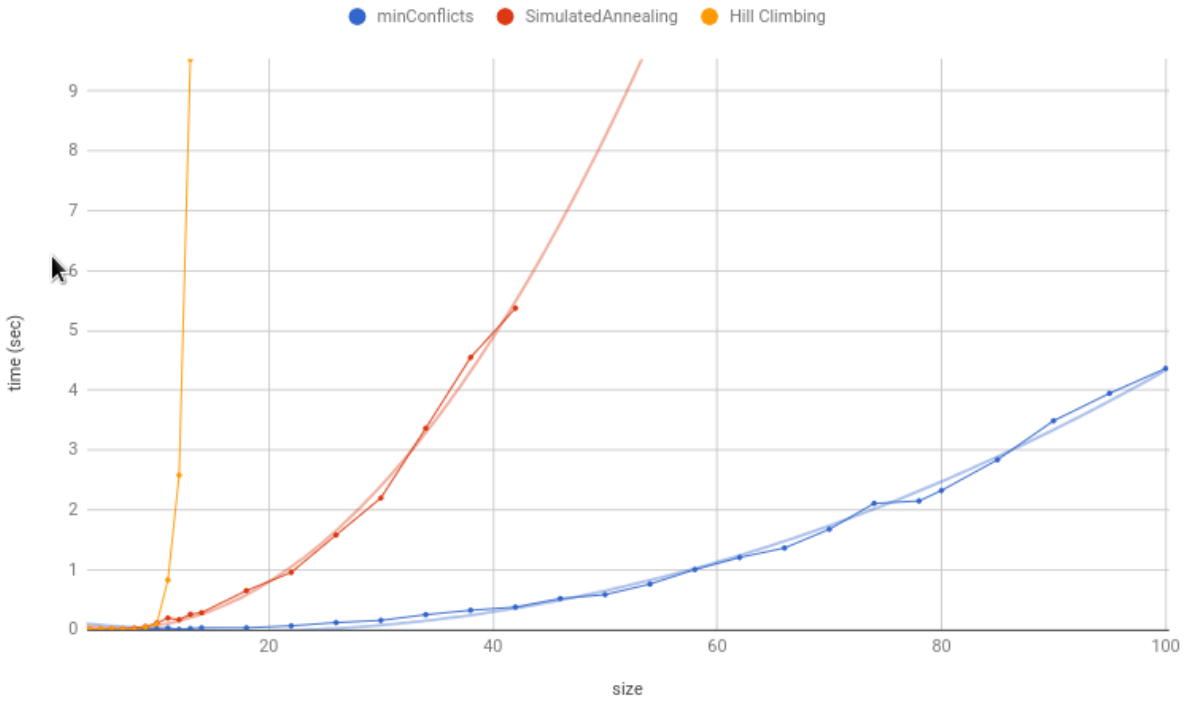
Ο Αλγόρθμος:

Algorithm 2 Successor_function

1: **Input:**
   1. A Queen to move.
   2. A Board where the Queens are.
2: **Output:** Moves with min conflicts
Initialise candidates[ ] as empty array
min_conflicts = maximum conflicts possible
for pos in board
    if conflicts == min_conflicts
        add pos to candidates[ ]
    if conflicts < min_conflicts
        flush candidates
Return candidates

Μέρος Γ

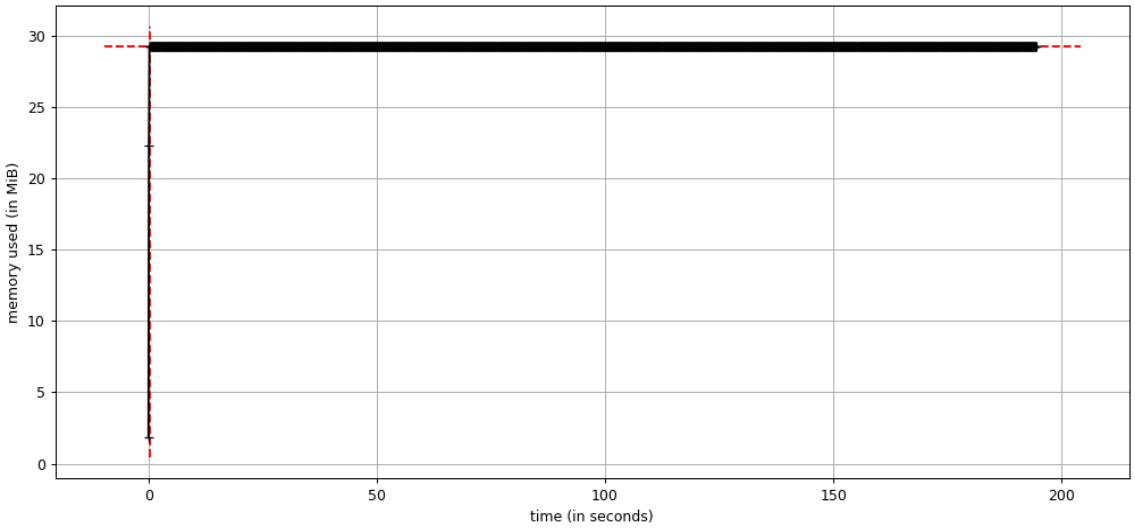
Για τον έλεγχο απόδοσης των μεθόδων μας επαναλάβαμε τους αλγόριθμους **Προσομοιωμένης Ανόπτησης** και **Ελαχίστων Συγκρούσεων** 20 φορές για διάφορα μεγέθη n βασιλισσών και κατά συνέπεια για μεγέθη σκακιέρας $n \times n$ για $n = 10, 14, 18, ..., 38$ και καταγράφουμε το μέσο όρο εκτέλεσης για κάθε n.



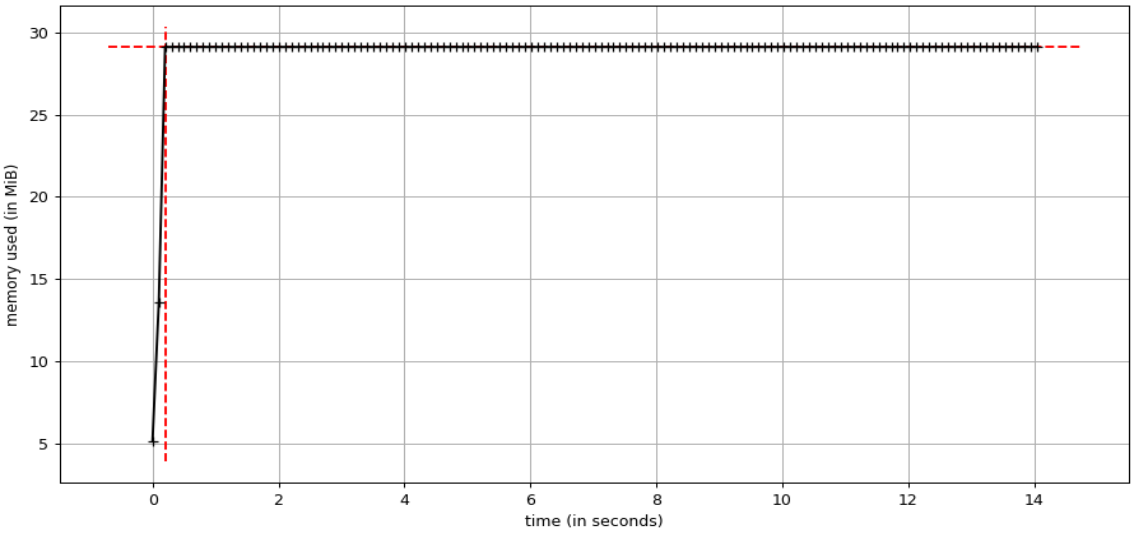
Σχήμα 1: Γράφημα 1

Στο γράφημα 1 εμφανίζεται η χρονική πολυπλοκότητα των αλγορίθμων. Εδώ μπορούμε να δούμε το σημαντικό πλεονέκτημα του αλγορίθμου **Ελαχίστων Συγκρούσεων** του οποίου και η πολυπλοκότητα αυξάνεται πολυωνυμικά, με κύριο overhead την τοποθέτηση των βασιλισσών. Σε αντίθεση το γράφημα του αλγορίθμου **Προσομοιωμένης Ανόπτησης** εμφανίζει πολυωνυμική πολυπλοκότητα με μεγαλύτερο βαθμό. Τέλος μπορουμε να παρατηρήσουμε την εκθετική πολυπλοκότητα του αλγοριθμου Hill Climbing With Random Restart.

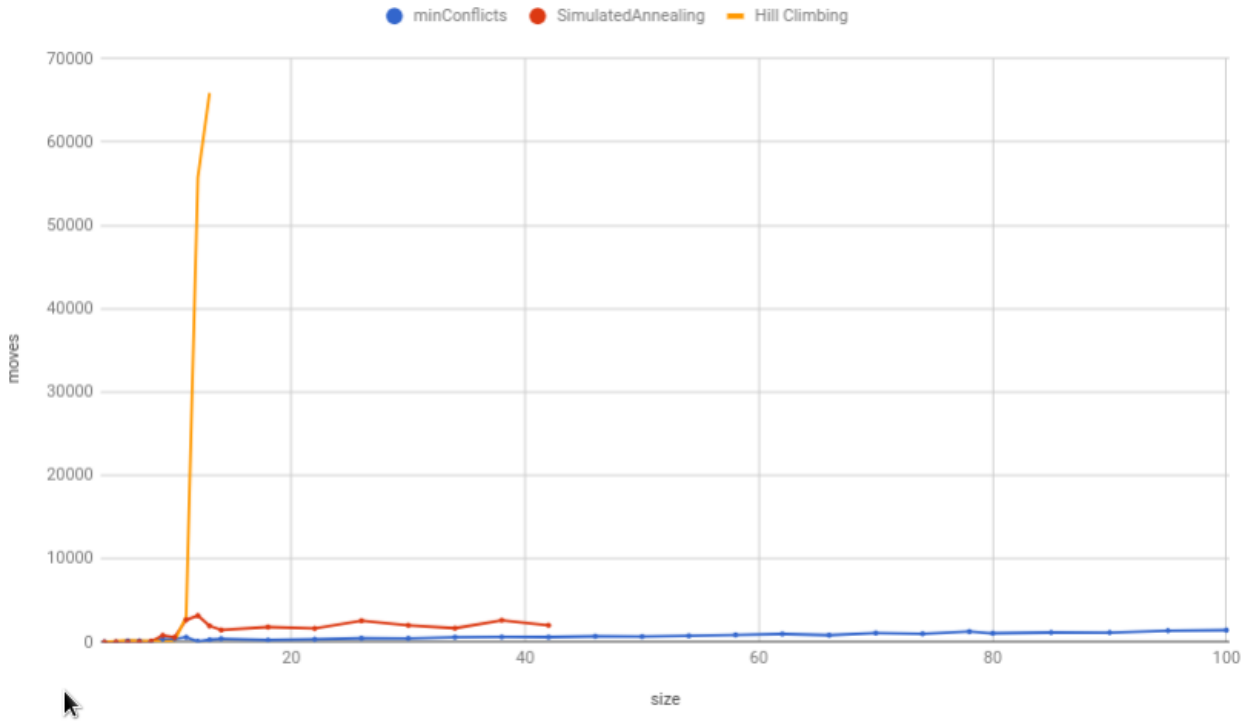
Κατά την διάρκεια των δοκιμασιών καταγράφτηκε και η μνήμη που χρειάστηκαν οι δύο αλγόριθμοι χρησιμοποιώντας την βιβλιοθήκη memory-profiler της python. Τα παρακάτω γραφήματα 2 και 3 δείχνουν την χωρική πολυπλοκότητα των δυο αλγορίθμων **Προσομοιωμένης Ανόπτησης** και **Ελαχίστων Συγκρούσεων** αντίστοιχα για τα διαφορετικά n. Αξίζει να σημειωθεί ότι στα γραφήματα καταγράφεται η μνήμη ανά χρονική στιγμή, και όχι ανά διαφορετικό n, όμως και στις δύο μετρήσεις ο αριθμός n των βασιλισσών αυξάνεται σταδιακά, χωρίς να παρατηρούμε καμία σημαντική αύξηση στην απαιτούμενη μνήμη.



Σχήμα 2: Γράφημα 2



Σχήμα 3: Γράφημα 3



Σχήμα 4: Γράφημα 4

Στο γράφημα 4 παρατηρούμε ξανά το πλεονέκτημα του αλγορίθμου **Ελαχίστων Συγκρούσεων** του οποίου ο αριθμός μετακινήσεων παραμένει πρακτικά σταθερός, ενώ ο αριθμός μετακινήσεων του αλγορίθμου **Προσομοιωμένης Ανόπτησης** είναι πάντα μεγαλύτερος, και τείνει να αυξάνεται όσο μεγαλώνει ο αριθμός βασιλισσών και το μέγεθος της σκακιέρας. Τέλος μπορούμε να παρατηρήσουμε τον γιγαντιαίο αριθμό κινήσεων που χρειάζεται ο αλγόριθμος Climbing With Random Restart.

Παρακάτω παρατίθενται και οι τελικές μετρήσεις των πειραμάτων μας :

	Time			Moves		
Size	minConflicts	Simulated Annealing	Hill Climbing	minConflicts	Simulated Annealing	Hill Climbing
4	0.00020003	0.00051762	0.01041369	10	1	44
5	0.00019999	0.00082177	0.01535244	9	17	134
6	0.00379543	0.01229014	0.02133441	143	18	224
7	0.00469995	0.01222363	0.01359868	140	38	179
8	0.00400004	0.02464843	0.00849829	95	115	134
9	0.02295804	0.05332117	0.04432950	362	815	100
10	0.01781807	0.10907342	0.09591050	418	586	72
11	0.02929754	0.19616318	0.83176079	588	2681	3080
12	0.00595026	0.16793098	2.58324146	97	3186	55730
13	0.01955280	0.25835557	9.53337975	286	1946	65894
14	0.03274453	0.28553381		418	1452	
18	0.03194671	0.65348430		253	1811	
22	0.06550946	0.95955343		354	1645	
26	0.12023315	1.58358834		473	2570	
30	0.15655828	2.20244663		468	2021	
34	0.25235906	3.36802211		595	1677	
38	0.32566050	4.55150826		630	2630	
42	0.37819457	5.37487628		608	2024	
46	0.52114111			705		
50	0.58691858			677		
54	0.76382827			758		
58	1.00601343			869		
62	1.20792278			985		
66	1.36659228			838		
70	1.67956294			1094		
74	2.11115737			1014		
78	2.15222881			1270		
80	2.32636684			1069		
85	2.84045250			1163		
90	3.49127494			1158		
95	3.95104798			1385		
100	4.36559958			1441		

Οδηγίες Εκτέλεσης

Για να μπορέσουμε να εκτελέσουμε τον κώδικα χρειάζεται να έχουμε στην διάθεση μας την Python3.6 και από dependencies χρειαζόμαστε το numpy. Αφού υπάρχουν τα παραπάνω στο σύστημα που επιθυμούμε να τρέξουμε τον κώδικα αρκεί να εκτελέσουμε το runner.py χωρίς την χρήση ορισμάτων ώστε να εκτελεστεί ένα default demo. Εκτελόντας:

- $ python3.6 runner.py -h

Μπορούμε να δούμε το help menu ώστε να δούμε όλα τα δυνατά ορίσματα τα οποία μπορούμε να προσθέσουμε για να εκτελέσουμε τον κώδικα για συγκεκριμένες τιμές.
Στο αρχείο runner.py φαίνονται όλα τα δυνατά ορίσματα που μπορούν να μπουν στο πρόγραμμα. Για παράδειγμα αν κάποιος ήθελε να τρέξει το πρόγραμμα για τον αλγόριθμο minConflicts από 8 εως 16 με βήμα 2 θα έγραφε:

- $ python runner.py –start 8 –stop 16 –step 2 –algo MC

Πηγές

[1] Russel, S. & Norvig, P. (2015). Τεχνητή Νοημοσύνη Μία σύγχρονη προσέγγιση.
Αθήνα: Κλειδάριθμος.

[2] Katrina Ellison Geltman, (2014). The Simulated Annealing Algorithm. Διαθέσιμο σε:
http://katrinaeg.com/simulated-annealing.html (Ανακτήθηκε 27 Μαίου, 2018)

[3] N-Queens and Local Search. Διαθέσιμο σε:
https://academics.tjhsst.edu/compsci/ai/nqueens/NQueens.pdf (Ανακτήθηκε 26 Μαίου, 2018)

[4] Heuristics and Stochastic Algorithms. Διαθέσιμο σε:
https://ktiml.mff.cuni.cz/ bartak/constraints/stochastic.html (Ανακτήθηκε 26 Μαίου, 2018)

[5] Fahiem Bacchus, (x.x). Constraint Satisfaction Problems. Διαθέσιμο σε:
http://www.cs.toronto.edu/ fbacchus/Presentations/CSP-BasicIntro.pdf (Ανακτήθηκε 27 Μαίου,2018)

[6] Min-conflicts algorithm. Διαθέσιμο σε:
https://en.wikipedia.org/wiki/Min-conflicts_algorithm (Ανακτήθηκε 26 Μαίου, 2018)