ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
Σχολή Τεχνολογιών Πληροφορικής και Τηλεπικοινωνιών
Τμήμα Ψηφιακών Συστημάτων

Παράλληλη Επεξεργασία Ερωτημάτων Αντίστροφων Κορυφαίων-Κ στην Κύρια Μνήμη

Πτυχιακή Εργασία

Παπαϊωάννου Ιωάννης, Α.Μ. Ε14146

Επιβλέπων Καθηγητής: Χρήστος Δουλκερίδης

Περίληψη

Σε αυτή την πτυχιακή, εξετάζεται και αντιμετωπίζεται το πρόβλημα του αποτελεσματικού υπολογισμού ενός ερωτήματος αντίστροφων κορυφαίων-κ (reverse top-k queries) χρησιμοποιώντας πολλαπλές διεργασίες, με δεδομένα και προτιμήσεις που βρίσκονται στην κύρια μνήμη. Στόχος είναι η υλοποίηση μιας βιβλιοθήκης η οποία θα παρέχει μια προγραμματιζόμενη διεπαφή (API) μέσα από την οποία θα είναι δυνατό να πραγματοποιηθεί ο παράλληλος υπολογισμός του ερωτήματος. Απο τα πειράματα που εκτελούνται, αναγνωρίζεται σημαντική βελτίωση σε σχέση με μια λύση η οποία τρέχει σε μια διεργασία. Οι υλοποιημένοι αλγόριθμοι και μέθοδοι κατανομής των δεδομένων στις διεργασίες συγκρίνονται μεταξύ τους, ώστε να εντοπιστεί ο πιο αποδοτικός τρόπος επεξεργασίας του ερωτήματος.

Keywords --- Reverse Top-K, Multiprocessing, Parallel, Performance

Δήλωση Περί μη υποκλοπής

Εγώ ο Παπαϊωάννου Ιωάννης, δηλώνω υπεύθυνα ότι η παρούσα πτυχιακή εργασία με τίτλο *Παράλληλη Επεξεργασία Ερωτημάτων Αντίστροφων Κορυφαίων-Κ στην Κύρια Μνήμη* είναι δική μου και βεβαιώνω ότι:

- Σε όσες περιπτώσεις έχω συμβουλευτεί δημοσιευμένη εργασία τρίτων, αυτό επισημαίνεται με σχετική αναφορά στα επίμαχα σημεία.
- Σε όσες περιπτώσεις μεταφέρω λόγια τρίτων, αυτό επισημαίνεται με σχετική αναφορά στα επίμαχα σημεία. Με εξαίρεση τέτοιες περιπτώσεις, το υπόλοιπο κείμενο της πτυχιακής αποτελεί δική μου δουλειά.
- Αναφέρω ρητά όλες τις πηγές βοήθειας που χρησιμοποίησα.
- Σε περιπτώσεις που τμήματα της παρούσας πτυχιακής έγιναν από κοινού με τρίτους, αναφέρω ρητά ποια είναι η δική μου συνεισφορά και ποια των τρίτων.
- Γνωρίζω πως η λογοκλοπή αποτελεί σοβαρότατο παράπτωμα και είμαι ενήμερος για την επέλευση των νομίμων συνεπειών

Ευχαριστίες

Θα ήθελα να εκφράσω τις ιδιαίτερες ευχαριστίες μου στον επιβλέποντα μου κ. Δουλκερίδη Χρήστο για την καθοδήγηση και υπομονή που μου προσέφερε απλόχερα. Η ανάθεση της πτυχιακής είχε ως αποτέλεσμα την ενασχόληση μου, την αύξηση των γνώσεων μου επί του θέματος, την βαθύτερη κατανόηση μου για τον κόσμο του πειραματισμού και να γίνω ένας καλύτερος άνθρωπος.

Είμαι βαθιά υποχρεωμένος στον καθηγητή κ. Τελέλη Ορέστη, του οποίου η ενθάρρυνση μέσα από τις τεχνικές αλλά και μη τεχνικές συμβουλές αποτελούσαν πάντα μια σημαντική υποστήριξη για την πτυχιακή μου.

Τέλος θα ήθελα να ευχαριστήσω τους γονείς μου και την αδελφή μου. Χωρίς την βοήθεια και υποστήριξη τους δεν θα μου ήταν δυνατό να καταφέρω όσα έχω επιτύχει.

ΠΑΠΑΪΩΑΝΝΟΥ ΙΩΑΝΝΗΣ xx/xx/2019

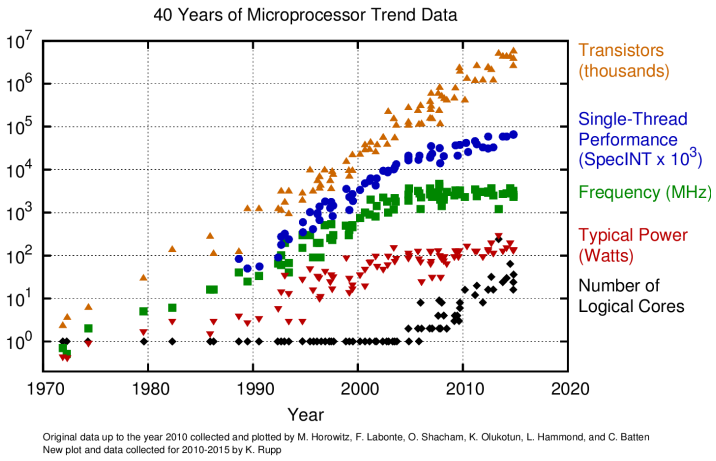
Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

Ένα πολύ ενδιαφέρον ερώτημα κατάταξης είναι το ερώτημα αντίστροφων κορυφαίων-k (reverse top-k query) [18] το οποίο ασχολείται με την εύρεση της απήχησης που έχει σε ένα κοινό η εισαγωγή ενός αντικειμένου στο χώρο των διαθέσιμων αντικειμένων. Το μειονέκτημα του ερωτήματος είναι πως για τον υπολογισμό του, απαιτούνται πολλαπλά ερωτήματα κορυφαίων-k (top-k queries) με αποτέλεσμα να είναι ιδιαίτερα απαιτητικό στον επεξεργαστή (CPU). Για τον εξής λόγο, σε αυτή την εργασία προσπαθούμε να δούμε αν μπορούμε να υπολογίσουμε ερωτήματα αντίστροφων κορυφαίων-k χρησιμοποιώντας παραλληλισμό, δηλαδή αντί να το υπολογίσουμε με μια μόνο διεργασία, να διανείμουμε το πρόβλημα σε πολλούς πυρήνες ώστε να εκτελεστεί πιο γρήγορα.

1.1 Πολλαπλοί Πυρήνες



Source: https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/

Σχήμα 1.1: Τάσεις πάνω στην ανάπτυξη επεξεργαστών

Η τάση προς τους πολλαπλούς πυρήνες είναι μια τεχνική προσέγγιση που βοηθά τους σχεδιαστές επεξεργαστών να αποφύγουν το πρόβλημα της υψηλής κατανάλωσης ενέργειας που εμφανίζεται με την κλιμάκωση των επεξεργαστών σε μεγαλύτερες συχνότητες. Καθώς οι ταχύτητες των επεξεργαστών αυξήθηκαν στην περιοχή $3 - 4$ GHz, η ποσότητα ηλεκτρικής ενέργειας που απαιτείται για να φτάσει

υψηλότερες συχνότητες άρχισε να γίνεται υπερβολικά υψηλή.[10] Αυτός είναι ο λόγος για τον οποίο ξεκίνησε η μετάβαση σε πολλούς πυρήνες μια τάση που μπορούμε να παρατηρήσουμε στο σχήμα 1.1.

Ειδικότερα η αύξηση των ταχυτήτων του ρολογιού συνεπάγεται αύξηση της τάσης, η οποία οδηγεί σε κυβική αύξηση της κατανάλωσης ρεύματος για το κύκλωμα. Έτσι, καθώς οι ταχύτητες ρολογιού ανεβαίνουν, παράγεται περισσότερη θερμότητα, απαιτώντας πιο ισχυρές λύσεις ψύξης. Για να θέσουμε σε λειτουργία τα τρανζίστορ και να αυξήσουμε την ταχύτητα του ρολογιού απαιτείται περισσότερη τάση, οδηγώντας σε δραματικά μεγαλύτερη κατανάλωση ενέργειας. Διαπιστώνουμε ότι η κατανάλωση θερμότητας και ενέργειας αυξάνεται δραματικά, καθώς προσπαθούμε να αυξήσουμε την ταχύτητα του ρολογιού, με αποτέλεσμα οι απαιτήσεις ισχύος και η παραγωγή θερμότητας να ξεπερνούν την ταχύτητα του ρολογιού.

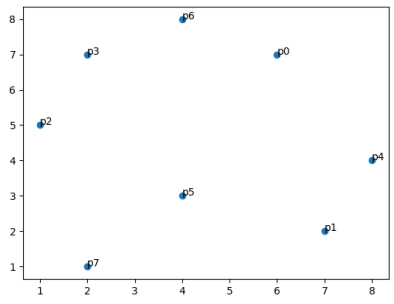
Παρόλα αυτά οι μεγαλύτερες ταχύτητες ρολογιού δεν σημαίνουν απαραίτητα ταχύτερους και καλύτερους υπολογιστές. Η ικανότητα των υπολογιστών μπορεί ακόμα να αυξηθεί ακόμα και αν η ταχύτητα του ρολογιού του επεξεργαστή πέφτει. Οι τάσεις στην επεξεργασία πολλαπλών πυρήνων θα παρέχουν μεγαλύτερη ισχύ επεξεργασίας με τις ίδιες ταχύτητες, ειδικά καθώς βελτιώνεται η παραλληλισμός του λογισμικού.[9]

1.2 Κλιμάκωση του ερωτήματος αντίστροφων κορυφαίων-k

Για να εξηγήσουμε τις προκλήσεις που εμφανίζονται κατά την κλιμάκωση ενός ερωτήματος αντίστροφων κορυφαίων-k θα κάνουμε αρχικά μια εισαγωγή στο τρόπο λειτουργίας των ερωτημάτων κορυφαίων-k. Ένα ερώτημα κορυφαίων-κ επιστρέφει για μια προτίμηση w τα k σημεία με τις καλύτερες βαθμολογίες. Η προτίμηση w σε ένα ερώτημα κορυφαίων-k λειτουργεί ως συνάρτηση, για παράδειγμα $w(x, y) = w_0 \cdot x + w_1 \cdot y$ όπου w_0 η πρώτη διάσταση της προτίμησης, w_1 η δεύτερη διάσταση της και η τιμή του $w(x, y)$ είναι η βαθμολογία του αντικείμενου με χαρακτηριστικά (x,y). Μέσα από αυτή την συνάρτηση απαντάμε το ερώτημα των κορυφαίων-k.

Πίνακας 1.1: Δεδομένα

Point	X	Y
p0	6	7
p1	7	2
p2	1	5
p3	2	7
p4	8	4
p5	4	3
p6	4	8
p7	2	1



Σχήμα 1.2: Χωρική αναπαράσταση

Πίνακας 1.2: Προτιμήσεις

User	w[1]	w[2]	top-2
John	0.6	0.4	[p7, p2]
George	0.3	0.7	[p7, p5]
Thanos	0.5	0.5	[p7, p2]

Για παράδειγμα από την εικόνα(1.2) και τους πίνακες(1.1, 1.2) το αποτέλεσμα κάθε χρήστη ως top_2 είναι για τον John και τον Thanos το [p7, p2] ενώ για τον George είναι [p7, p5].

Για να βρούμε πόση απήχηση, ως προς τις κορυφαίες δύο επιλογές των χρηστών, έχει το αντικείμενο p0 κοιτάμε σε ποιους χρήστες το p0 βρίσκεται στα top_2 με βάση τον πίνακα 1.2. Βλέποντας ότι κανένας από τους χρήστες δεν έχει το p0 στις top_2 επιλογές τους, ένα $RTOP_2(p0)$ επιστρέφει $\emptyset$ αποτέλεσμα. Αντίστοιχα το $RTOP_2(p2)$ επιστρέφει [John, Thanos] και το $RTOP_2(p7)$ επιστρέφει [John, Thanos, George].

Είναι φυσικό, αυξάνοντας το πλήθος του συνόλου δεδομένων, να αυξάνεται ο χρόνος που χρειάζεται για να εκτελεστεί το κάθε $TOP_k(w)$ για κάθε διαθέσιμη προτίμηση, ενώ όταν αυξάνονται οι προτιμήσεις να αυξάνεται το πλήθος από $TOP_k(w)$ που χρειάζεται για να υπολογίσουμε το $RTOP_k(Q)$. Παρατηρούμε ότι, το ερώτημα καθώς κλιμακώνεται γίνεται πολύ απαιτητικό ως προς την χρήση του επεξεργαστή γιατί χρειάζονται πολλές παραπάνω πράξεις για να υπολογιστεί.

Οπότε το παραπάνω γεγονός σε συνδυασμό με τις τάσεις στον χώρο των επεξεργαστών όπως είδαμε στην ενότητα 1.1 συγκροτεί την ιδέα του να χρησιμοποιήσουμε παραλληλισμό για να λύσουμε το ερώτημα αντίστροφων κορυφαίων-k με μεγαλύτερη ταχύτητα.

1.3 Στόχοι

Η παρούσα πτυχιακή έχει ως αντικείμενο την ανάπτυξη λύσεων για τον υπολογισμό ερωτημάτων αντίστροφων κορυφαίων-k χρησιμοποιώντας παραλληλισμό και πιο συγκεκριμένα την υλοποίηση μιας βιβλιοθήκης η οποία θα εκτελεί αυτά τα ερωτήματα σε πολλές διεργασίες. Αναλυτικότερα αυτή η βιβλιοθήκη θα εκθέτει, μέσα από μια φιλική προς τον χρήστη προγραμματιζόμενη διεπαφή (API), μια πληθώρα από αλγόριθμους και στρατηγικές για την κατανομή προτιμήσεων σε διεργασίες. Θα μας παρέχει την δυνατότητα και τα εργαλεία που θα χρειαστούμε για να μελετήσουμε και να αξιολογήσουμε τους αλγόριθμους και την αποδοτικότητα τους ώστε να εντοπίσουμε τον καλύτερο συνδυασμό αλγόριθμου και στρατηγικής διαχωρισμού προτιμήσεων. Εκτός αυτού θα μελετηθούν αλγόριθμοι που υπάρχουν από προηγούμενες εργασίες και αφορούν την επίλυση των αντίστροφων κορυφαίων ερωτημάτων. Τέλος θα γίνει μια προσπάθεια να προσδιοριστεί ένα πλήθος από πιθανές μελλοντικές επεκτάσεις.

1.4 Διάρθρωση εγγράφου

Η πτυχιακή αυτή στην παρούσα μορφή της αποτελείται από 6 κεφάλαια, των οποίων το περιεχόμενο αναλύεται παρακάτω:

Στο κεφάλαιο 1, Εισαγωγή: Ορίζουμε την θεματική περιοχή και τους στόχους αυτής της πτυχιακής εργασίας.

Στο κεφάλαιο 2, Θεωρητικό υπόβαθρο: Κάνουμε μια μικρή αναφορά στο θεωρητικό υπόβαθρο της εργασίας. Αναλύονται έννοιες που αφορούν τον παραλληλισμό, ερωτήματα κορυφαίων-κ και αντίστροφα ερωτήματα κορυφαίων-κ

Στο κεφάλαιο 3, Ανασκόπηση βιβλιογραφίας: Γίνεται μια παρουσίαση του ήδη υπάρχοντος ερευνητικού έργου, πάνω σε ερωτήματα αντίστροφων κορυφαίων-κ.

Στο κεφάλαιο 4, Μοντέλο επίλυσης ερωτημάτων αντίστροφων κορυφαίων-k: Παρουσιάζονται οι αλγόριθμοι που επιλέχθηκαν και υλοποιήθηκαν για τον πειραματισμό. Επίσης αναλύονται οι στρατηγικές που εφαρμόστηκαν για την επίτευξη παραλληλισμού πάνω σε ερωτήματα αντίστροφων κορυφαίων-κ.

Στο κεφάλαιο 5, Πειραματική αξιολόγηση: Παρατίθεται και αναλύεται η διαδικασία αξιολόγησης των αλγόριθμων και στρατηγικών που προτάθηκαν στα προηγούμενα κεφάλαια.

Στο κεφάλαιο 6, Συμπεράσματα και Μελλοντική έρευνα: Προσδιορίζονται μελλοντικά ερευνητικά θέματα πάνω στο αντικείμενο.

Κεφάλαιο 2

Θεωρητικό υπόβαθρο

Σε αυτό το κεφάλαιο, γίνεται εισαγωγή στις βασικές έννοιες για την εκτέλεση ερωτημάτων αντίστροφων κορυφαίων-k χρησιμοποιώντας παράλληλο υπολογισμό (parallel computing). Αρχικά περιγράφονται τα οφέλη που προσφέρει ο παραλληλισμός και οι προκλήσεις που εμφανίζονται κατά την εφαρμογή του ώστε να πετύχουμε υψηλή αποδοτικότητα. Στην συνέχεια γίνεται μια εισαγωγή στα ερωτήματα κατάταξης και στα πεδία εφαρμογής τους. Τέλος γίνεται μια θεωρητική ανάλυση σε ότι αφορά ερωτήματα κορυφαίων-k και ερωτήματα αντίστροφων κορυφαίων-k.

2.1 Παράλληλος υπολογισμός

Ο κατανεμημένος υπολογισμός (distributed computing) είναι η διαδικασία συσσωμάτωσης της δύναμης πολλών υπολογιστικών οντοτήτων, οι οποίοι είναι λογικά κατανεμημένοι και μπορεί ακόμη και να είναι γεωλογικά κατανεμημένοι, να εκτελούν από κοινού ένα ενιαίο υπολογιστικό έργο με διαφάνεια και συνεκτικό τρόπο, έτσι ώστε να εμφανίζονται ως ένα ενιαίο συγκεντρωτικό σύστημα. Ο παράλληλος υπολογισμός (parallel computing), ο οποίος αποτελεί κλάδο του κατανεμημένου υπολογισμού, είναι η ταυτόχρονη εκτέλεση της ίδιας εργασίας σε πολλούς επεξεργαστές προκειμένου να επιτευχθούν ταχύτερα αποτελέσματα.

Ο παραλληλισμός χρησιμοποιείται ήδη σε υπολογιστές υψηλών επιδόσεων, αλλά κερδίζει ευρύτερο ενδιαφέρον εξαιτίας των φυσικών περιορισμών που εμποδίζουν την κλιμάκωση συχνοτήτων. Καθώς η κατανάλωση ενέργειας (και συνεπώς η παραγωγή θερμότητας) από τους υπολογιστές έχει καταστεί προβληματική τα τελευταία χρόνια, ο παράλληλος υπολογισμός έχει γίνει το κυρίαρχο πρότυπο στην αρχιτεκτονική υπολογιστών, κυρίως με τη μορφή επεξεργαστών πολλαπλών πυρήνων, όπως είδαμε στην ενότητα 1.1.

Ιδιαίτερα σημαντικές για τον παραλληλισμό είναι οι έννοιες της διεργασίας και του νήματος. Διεργασία (process) ονομάζεται το στιγμιότυπο ενός προγράμματος υπολογιστή που εκτελείται. Κάθε διαδικασία έχει το δικό της χώρο μνήμης (address space) που χρησιμοποιεί για να αποθηκεύσει τις οδηγίες που εκτελούνται, καθώς και όλα τα δεδομένα που χρειάζεται για αποθήκευση και πρόσβαση για εκτέλεση. Σε μία διεργασία μπορεί να υπάρχουν πολλά νήματα (threads) που έχουν τη δυνατότητα να εκτελούνται ταυτόχρονα με την ίδια την διεργασία. Τα νήματα μιας κοινής γονικής διεργασίας μοιράζονται τον ίδιο χώρο μνήμης, δηλαδή τον χώρο μνήμης της γονικής διαδικασίας και αυτό έχει ως αποτέλεσμα ένα νήμα να είναι πολύ πιο ελαφρύ σε σχέση με μια διεργασία.[12]

2.1.1 Κίνητρο για παράλληλο υπολογισμό

Ο κύριος σκοπός της εκτέλεσης παράλληλων υπολογισμών είναι η ταχύτερη επίλυση προβλημάτων ή η επίλυση μεγαλύτερων προβλημάτων. Ο παράλληλος υπολογισμός χρησιμοποιείται ευρέως για τη μείωση του χρόνου υπολογισμού για πολύπλοκες εργασίες. Πολλές βιομηχανικές και επιστημονικές έρευνες για να έρθουν εις πέρας έχουν ανάγκη σύνθετους υπολογιστές μεγάλης κλίμακας, όπου χωρίς αυτούς θα χρειάζονταν χρόνια ή και δεκαετίες για να ολοκληρωθούν. Σε πολλές εφαρμογές, είναι πολύ επιθυμητό τα αποτελέσματα να είναι διαθέσιμα το συντομότερο δυνατό, καθώς τα καθυστερημένα αποτελέσματα συχνά συνεπάγονται άχρηστα αποτελέσματα.[20]

Ένα χαρακτηριστικό παράδειγμα είναι η πρόγνωση του καιρού, η οποία χαρακτηρίζεται από εξαιρετικά περίπλοκο υπολογισμό και μεγάλο σύνολο δεδομένων. Έχει επίσης αυστηρή απαίτηση χρόνου, λόγω του χαρακτήρα της διαδικασίας που είναι η πρόβλεψη του καιρού, καθώς οι παρελθοντικές προβλέψεις σπάνια αποτελούν αντικείμενο ενδιαφέροντος για το ευρύτερο κοινό.

2.1.2 Αποδοτικότητα του παράλληλου υπολογισμού

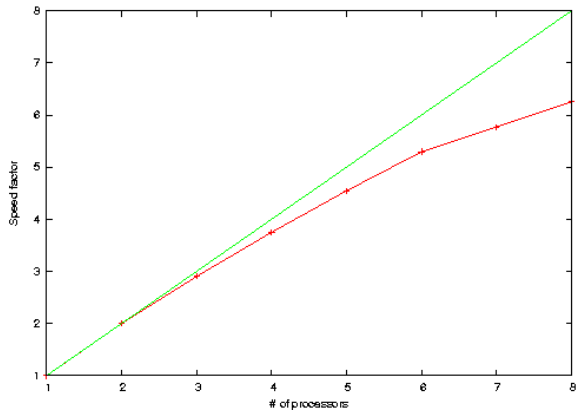
Προκειμένου να αποδειχθεί η αποτελεσματικότητα της παράλληλης επεξεργασίας για ένα πρόβλημα σε κάποια πλατφόρμα, έχουν οριστεί αρκετά μέτρα. Αυτά τα μέτρα θα χρησιμοποιηθούν κατά το κεφάλαιο 5 για την αξιολόγηση της αποτελεσματικότητας των παράλληλων προγραμμάτων.

Περιλαμβάνουν την *επιτάχυνση (Speedup)*, η οποία περιγράφει το κέρδος που έχουμε ως προς τον χρόνο που απαιτείται για έναν επεξεργαστή σε σχέση με τον χρόνο που χρειάζεται το πρόβλημα για να λυθεί σε πολλαπλούς επεξεργαστές.

Ο χαρακτηριστικός τύπος της επιτάχυνσης είναι:

$$S(n) = \frac{T(1)}{T(n)}$$

όπου $S(n)$ είναι η επιτάχυνσης που επιτυγχάνεται με n επεξεργαστές, (1) είναι ο απαιτούμενος χρόνος για έναν επεξεργαστή και (n) είναι ο απαιτούμενος χρόνος για τους N επεξεργαστές.



Source: https://docs.abinit.org/tutorial/basepar_assets/basepar_speedup.png

Σχήμα 2.1: Επιτάχυνση ανά αριθμό επεξεργαστών

Παρόλα αυτά, όπως βλέπουμε από το διάγραμμα 2.1 ενώ θα έπρεπε να έχουμε μια γραμμική επιτάχυνση καθώς προστίθενται επεξεργαστές, η γραμμική επιτάχυνση παραμένει ιδανική περίπτωση,

αφού στην πράξη η επιτάχυνση εμφανίζεται ως μια καμπύλη η οποία βρίσκεται κάτω από την ευθεία $y = x$.

Επιπλέον καθώς ο αριθμός των επεξεργαστών αυξάνεται, η επιτάχυνση εντείνεται επίσης μέχρι να επιτευχθεί ένα σημείο κορεσμού. Μετά από αυτό το σημείο, η προσθήκη περισσότερων επεξεργαστών δεν θα αποφέρει περαιτέρω κέρδη απόδοσης. Αυτό είναι το συνδυασμένο αποτέλεσμα του ότι απαιτείται παραπάνω χρόνος για να δημιουργηθούν διεργασίες/νήματα καθώς διαμερίζεται το πρόβλημα σε περισσότερα μέρη, αυξάνονται οι πιθανοί διπλοί υπολογισμοί σε κάποια προβλήματα και χρειάζεται παραπάνω χρόνος για να συγχρονιστούν και να επικοινωνήσουν οι διεργασίες μεταξύ τους.

2.1.3 Νόμος του Amdahl

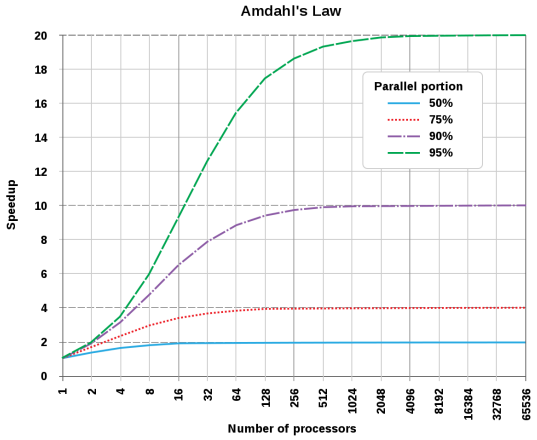
Όπως είναι φανερό από το προηγούμενο διάγραμμα 2.1 ο ρυθμός αύξησης της επιτάχυνσης μειώνεται καθώς προστίθενται περισσότεροι επεξεργαστές. Αυτό το φαινόμενο συνεπάγεται το όριο της απόδοσης που μπορούμε να λάβουμε από τον παραλληλισμό μια διαδικασίας, γιατί όταν ο αριθμός των επεξεργαστών ξεπεράσει κάποιο όριο, η προσθήκη περισσότερων επεξεργαστών δεν θα παράγει περαιτέρω βελτίωση της απόδοσης και θα οδηγήσει ακόμη και σε υποβάθμιση της απόδοσης.

Οι λόγοι όπως προαναφέρθηκαν είναι μείωσης της εξοικονόμησης χρόνου που προκύπτει από περαιτέρω κατανομή εργασιών, αύξηση σε επιβάρυνση της επικοινωνίας μεταξύ διαδικασιών και πιθανόν διπλούς υπολογισμούς.

Ο Gene Amdahl παρουσιάζει μια αρκετά απλή αν και πεσσιμιστική ανάλυση σχετικά με αυτό, που αργότερα αναφέρεται ως νόμος του Amdahl. Είναι σημαντικό βέβαια να αναφέρουμε ότι αυτή η ανάλυση παραμελεί άλλα πιθανά σημεία συμφόρησης, όπως το εύρος ζώνης μνήμης (memory) και το εύρος ζώνης εισόδου / εξόδου (I/O). Εάν οι πόροι αυτοί δεν κλιμακώνονται με τον αριθμό των επεξεργαστών, τότε απλώς προσθέτοντας επεξεργαστές υποβαθμίζεται παραπάνω η απόδοση.

Ο Gene Amdahl έδωσε την επιτάχυνση ενός παράλληλου προγράμματος με τον τύπο που ακολουθεί:

$$S(n) = \frac{1}{s + \frac{p}{n}}$$



Source: https://en.wikipedia.org/wiki/File:AmdahlsLaw.svg

Σχήμα 2.2: Επιτάχυνση ανά αριθμό επεξεργαστών με βάση το τμήμα που μπορεί να παραλληλιστεί

S(n) είναι η ταχύτητα που επιτυγχάνεται με n επεξεργαστές, όπου p είναι το ποσοστό του κώδικα το οποίο είναι παραλληλοποιήσιμο ενώ το s που είναι το υπόλοιπο μέρος του κώδικα είναι σειριακό καθώς δεν παραλληλίζεται και είναι ίσο με $1-p$. Από το παραπάνω τύπο καταλήγουμε στο συμπέρασμα ότι η μέγιστη επιτάχυνση είναι ως προς το ποσοστό του σειριακού κώδικα.[3]

$$S(n) < \frac{1}{s}$$

Για παράδειγμα αν παραλληλίζονται τα 4/5 τότε γνωρίζουμε ότι το 1/5 ενός προγράμματος δεν παραλληλίζεται και κατά συνέπεια η μέγιστη επιτάχυνση είναι 5.

2.1.4 Νόμος του Gustafson

Ο νόμος του Gustafson αντιμετωπίζει τον προβληματισμό που προκύπτει από τον νόμο του Amdahl, ο οποίος βασίζεται στην υπόθεση ότι το πρόβλημα έχει σταθερό μέγεθος, δηλαδή ο φόρτος εργασίας που χρειάζεται να εκτελεστεί δεν αλλάζει σε σχέση με τη βελτίωση των πόρων. Ο νόμος του Gustafson προτείνει ότι οι προγραμματιστές τείνουν να καθορίζουν το μέγεθος των προβλημάτων για να αξιοποιήσουν πλήρως την υπολογιστική ισχύ που διατίθεται καθώς βελτιώνονται οι πόροι. Επομένως, εάν είναι διαθέσιμος ταχύτερος εξοπλισμός, μεγαλύτερα προβλήματα μπορούν να επιλυθούν ταυτόχρονα.[4]

Ο Gustafson υπολόγισε ότι η ταχύτητα S(n) είναι η ταχύτητα που επιτυγχάνεται με n επεξεργαστές, όπου s είναι το ποσοστό του κώδικα το οποίο δεν είναι παραλληλοποιήσιμο.

$$S(n) = n + (1 - n)s$$

2.2 Ερωτήματα κατάταξης

Τα ερωτήματα κατάταξης (Ranking) είναι θεμελιώδη στην επιστήμη της ανάκτησης πληροφοριών, καθώς χρησιμοποιούνται από τις μηχανές αναζήτησης. Λαμβάνοντας υπόψη ένα ερώτημα `q` και μια συλλογή `D` εγγράφων που ταιριάζουν με το ερώτημα, το πρόβλημα είναι να κατατάξουμε, δηλαδή να ταξινομήσουμε, τα έγγραφα στην `D` σύμφωνα με κάποιο κριτήριο έτσι ώστε τα «καλύτερα» αποτελέσματα να εμφανίζονται στην αρχή της λίστας αποτελεσμάτων που βλέπει ο χρήστης. Η πλειοψηφία των μηχανών αναζήτησης χρησιμοποιούν αλγόριθμους κατάταξης για να παρέχουν στους χρήστες ακριβή και συναφή αποτελέσματα.[6]

Για παράδειγμα, οι εφαρμογές σε βάσεις δεδομένων πολυμέσων και ιστού έχουν ιδέες εγγύτητας, κατάταξης και ανατροφοδότησης των χρηστών. Όλα αυτά έχουν σοβαρό αντίκτυπο στο μοντέλο ερωτήματος σε μια παραδοσιακή βάση δεδομένων, ο οποίος δεν μπορεί να αντιμετωπιστεί επαρκώς από τις τρέχουσες προσεγγίσεις επεξεργασίας ερωτημάτων. Δύο από τις προκλήσεις διαχείρισης δεδομένων που θέτουν αυτές οι αναδυόμενες εφαρμογές είναι οι εξής:

- Η παραδοσιακή επεξεργασία ερωτημάτων με βάση το παραδοσιακό σχεσιακό (relational) μοντέλο που βασίζεται στην χρήση boolean συμβόλων (AND, OR, NOT, etc.) δεν μπορεί να αντικατοπτρίζει τις προτιμήσεις των χρηστών όσον αφορά τόσο τη σύνταξη του ερωτήματος όσο και την επεξεργασία ερωτημάτων.

- Σε πολλές εφαρμογές, η «ταξινόμηση» των αποτελεσμάτων των ερωτημάτων σύμφωνα με ορισμένα κριτήρια του χρήστη αποτελεί αναπόσπαστο μέρος της σημασιολογίας ερωτημάτων.

2.2.1 Ερωτήματα κορυφαίων-K

Το πιο γνωστό ερώτημα κατάταξης είναι τα ερωτήματα κορυφαίων-κ (Top-K queries). Η χρήση των ερωτημάτων κορυφαίων-k είναι ιδιαίτερα αισθητή σε αρκετά σύγχρονα συστήματα τα οποία προτείνουν εξατομικευμένα προϊόντα ή πληροφορία με βάση τις προτιμήσεις του χρήστη. Δεδομένης μιας βάσης δεδομένων αντικειμένων που περιγράφονται από ένα σύνολο αριθμητικών χαρακτηριστικών βαθμολόγησης και ενός χρήστη με μια συνάρτηση προτίμησης (βαθμολόγησης) που ορίζεται πάνω από αυτά τα χαρακτηριστικά, ένα ερώτημα κορυφαίων-k ανακτά τα αντικείμενα k με τις καλύτερες βαθμολογίες για τη συγκεκριμένη λειτουργία προτιμήσεων. Με άλλα λόγια τα ερωτήματα κορυφαίων-κ αναζητούν τις πλειάδες οι οποίες μεγιστοποιούν ή ελαχιστοποιούν μια προτίμηση υπό την μορφή συνάρτησης.

Ορισμός (Ερωτήματα κορυφαίων-K): Δεδομένου ενός θετικού ακεραίου k και ενός καθορισμένου από τον χρήστη συντελεστή στάθμισης ως διάνυσμα w, το αποτέλεσμα ενός $TOP_K(w)$ τέτοιο ώστε $TOP_K(w) \subseteq S$ και $|TOP_K(w)| \leqslant k$ ενός ερωτήματος κορυφαίων-k είναι μια συλλογή από σημεία $\forall p1 \in TOP_K(w)$ τέτοια ώστε $\forall p2 \in S - TOP_K(w)$ να ισχύει $f_w(p1) \leqslant f_w(p2)$.[18]

Στο μοντέλο που χρησιμοποιείται ευρέως σε σχετικές εργασίες και στην πράξη, οι χρήστες εκφράζουν τις προτιμήσεις τους μέσω γραμμικών ερωτήσεων κορυφαίων-k, που καθορίζονται με την ανάθεση ενός βάρους σε κάθε ένα από τα χαρακτηριστικά βαθμολόγησης, υποδεικνύοντας τη σημασία κάθε χαρακτηριστικού για τον χρήστη.[14]

2.2.2 Ερωτήματα αντίστροφων κορυφαίων-K

Το ερώτημα των αντίστροφων κορυφαίων-κ (Reverse Top-K Queries) όπως είδαμε και στην ενότητα 1.2 είναι ένα ερώτημα που έχει εισαχθεί στον επιστημονικό χώρο τα τελευταία χρόνια μέσα από τους Vlachou et al. το οποίο χρησιμοποιείται για να εντοπίσει τους χρήστες για τους οποίους ένα αντικείμενο q θα αποτελεί μια από τις κορυφαίες-κ επιλογές τους.[18]

Ορίζεται το ερώτημα αντίστροφων κορυφαίων-k σε δύο μορφές την μονοχρωματική (Monochromatic) και την διχρωματική (Bichromatic) μορφή υπό τους παρακάτω ορισμούς:

Ορισμός (Μονοχρωματικό ερώτημα αντίστροφων κορυφαίων-k): Έχοντας ως δεδομένο ένα σημείο Q και έναν θετικό αριθμό k μαζί με ένα σύνολο δεδομένων S, το αποτέλεσμα του μονοχρωματικού ερωτήματος αντίστροφων κορυφαίων-k $mRTOP_k(q)$ είναι μια συλλογή από διανύσματα. Στην συλλογή αυτή βρίσκονται διανύσματα $w[i]$ για τα οποία $\exists p \in TOP_k(w[i])$ τέτοια ώστε $f_{w_i}(q) \leqslant f_{w_i}(p)$.[18]

Ορισμός (Διχρωματικό ερώτημα αντίστροφων κορυφαίων-k): Έχοντας ως δεδομένο ένα σημείο Q και έναν θετικό αριθμό k μαζί με δυο σύνολα δεδομένων S και W, όπου το S αντιπροσωπεύει σημεία δεδομένων και το W αντιπροσωπεύει διαφορετικούς συντελεστές στάθμισης, το αποτέλεσμα του διχρωματικού ερωτήματος αντίστροφων κορυφαίων-k $bRTOP_k(q)$ είναι μια συλλογή από διανύσματα. Στην συλλογή αυτή βρίσκονται διανύσματα $w[i] \in W$ για τα οποία είναι απαραίτητο $\exists p \in TOP_k(w[i])$ τέτοια ώστε $f_{w_i}(q) \leqslant f_{w_i}(p)$.[18]

Στα μονοχρωματικά ερωτήματα δεν υπάρχει πληροφορία για τις προτιμήσεις των χρηστών οπότε ο στόχος είναι να αναγνωρίσουμε ποια μέρη της αγοράς θα επηρεάσει η εισαγωγή ενός προϊόντος. Σε αντίθεση με τα μονοχρωματικά, στα διχρωματικά ερωτήματα κοιτάμε στοχευμένα ποιες προτιμήσεις θα επηρεάσουμε καθώς γνωρίζουμε ποιες είναι οι προτιμήσεις οι οποίες κυριαρχούν στην αγορά.

Κεφάλαιο 3

Ανασκόπηση βιβλιογραφίας

Σε αυτό το κεφάλαιο παρουσιάζεται το σύνολο του ερευνητικού έργου από το οποίο αντλήθηκαν οι πληροφορίες σχετικά με την υλοποίηση ενός συστήματος που εφαρμόζει παραλληλισμό πάνω σε ερωτήματα αντίστροφων κορυφαίων-κ. Συγκεκριμένα αναλύονται οι συνεισφορές που προσέφερε η κάθε εργασία στον επιστημονικό χώρο και το πως θα χρησιμοποιηθούν για την επίτευξη του στόχου αυτής της εργασίας.

3.1 Reverse Top-K Queries

Η εργασία των Akrivi Vlachou et al με τίτλο *Reverse Top-K Queries*[18] αποτελεί την πρώτη εργασία στην οποία μελετήθηκαν εκτενώς και ορίστηκαν τα ερωτήματα αντίστροφων κορυφαίων-κ, όπου αυτά κατατάσσονται σε δυο διαφορετικές εκδόσεις την μονοχρωματική και την διχρωματική όπως αναφέρθηκαν και ορίστηκαν στο κεφάλαιο 2.2.2.

Αναλύονται οι γεωμετρικές ιδιότητες της δισδιάστατης περίπτωσης του μονοχρωματικού ερωτήματος αντίστροφων κορυφαίων-κ και περιγράφει ένα αλγόριθμο ο οποίος τις εκμεταλλεύεται επιτυχώς.

Μια από αυτές τις ιδιότητες είναι το κριτήριο της κυριαρχίας Pareto. Μέσα από αυτό το κριτήριο βγαίνει το συμπέρασμα πως όταν ένα σημείο p κυριαρχεί στον χώρο έναντι ενός άλλου σημείου q τότε το σημείο p θα έχει καλύτερη βαθμολόγηση από το q για κάθε προτίμηση που προκύπτει μέσα από μια μονότονη συνάρτηση. Αντίθετα αν το σημείο p κυριαρχείται από το σημείο q τότε γνωρίζουμε ότι το σημείο p θα έχει χειρότερη βαθμολόγηση από το q για κάθε προτίμηση.

Για να αντιμετωπίσουν τα μονοχρωματικά ερωτήματα δεν εξετάζονται τα σημεία p τα οποία κυριαρχούν το σημείο q ή κυριαρχούνται από το σημείο q γιατί γνωρίζουμε ότι ανήκουν ή δεν ανήκουν στο αποτέλεσμα του $TOP_K(w)$ χωρίς να χρειάζεται να το υπολογίσουμε. Συγκεκριμένα ο προτεινόμενος αλγόριθμος εξετάζει διακριτές προτιμήσεις w για τις οποίες εξετάζει αν το $q \in TOP_K(w)$ και εφόσον ανήκει στο αποτέλεσμα προσπαθεί να βρει το σύνορα του διαστήματος $[w_l, w_u]$ τέτοιο ώστε $w \in [w_l, w_u]$ και $q \in TOP_K(w_l)$ και $q \in TOP_K(w_u)$.

Για τα διχρωματικά ερωτήματα κορυφαίων-κ παρουσιάζεται ένας αποδοτικός αλγόριθμος Reverse top-k Threshold Algorithm (RTA) ο οποίος ορίζει ένα άνω όριο (threshold) το οποίο χρησιμοποιείται για να απορρίπτονται πολλές υποψήφιες προτιμήσεις χωρίς να χρειάζεται να εκτελέσουμε όλα τα ερωτήματα κορυφαίων-κ που θα εκτελούσε ένας αντίστοιχος «αφελής» αλγόριθμος. Πιο αναλυτικά ένας αφελής αλγόριθμος θα έτρεχε ένα $TOP_K(w) \forall w \in W$, όπου W είναι το σύνολο προτιμήσεων που εισάγουμε. Αντίθετα ο RTA κρατάει σε ένα buffer τα $TOP_K(w_i)$ και μέσα από αυτό υπολογίζει ένα threshold το οποίο είναι ίσο με τη μεγαλύτερη βαθμολογία από τα σημεία του buffer ως προς την επόμενη προτίμηση w_{i+1}. Αυτό το threshold το συγκρίνουμε στο επόμενο βήμα με τη βαθμολογία

του q για την προτίμηση w_{i+1}. Αν το q ξεπερνάει το threshold τότε γνωρίζουμε ότι υπάρχουν k σημεία τα οποία έχουν καλύτερη βαθμολογία δηλαδή όλα τα σημεία που εμπεριέχονται στο buffer οπότε γνωρίζουμε ότι το $w_{i+1} \notin bRTOP_k(q)$ χωρίς να τρέξουμε το $TOP_K(w_{i+1})$. Ένα ακόμα σημαντικό στοιχείο του παραπάνω αλγόριθμου είναι ότι οι προτιμήσεις εξετάζονται με μια σειρά όπου η προτίμηση w_i και η προτίμηση w_{i+1} είναι όσο πιο πολύ γίνεται παρόμοιες. Αυτό έχει ως στόχο να βελτιωθεί η αποδοτικότητα του threshold, δηλαδή να απορρίπτονται όσο το δυνατόν περισσότερες προτιμήσεις και επιτυγχάνεται μέσα από την ταξινόμηση ως προς την ομοιότητα του συνημιτόνου τους. Επιπλέον παρουσιάζουν μια δομή ευρετηρίασης που βασίζεται στην τμηματοποίηση του χώρου μέσω ενός πλέγματος η οποία βελτιώνει περαιτέρω τον χρόνο εκτέλεσης των ερωτημάτων.

Τέλος διεξάγουν μια πειραματική αξιολόγηση η οποία συγκρίνει και αναδεικνύει την αποδοτικότητα των αλγορίθμων τους.

Η ερευνητική εργασία πάνω στα ερωτήματα αντίστροφων κορυφαίων-κ συνιστά τις βάσεις πάνω στις οποίες βασίστηκε όλη η πτυχιακή εργασία, το A και το Ω. Ο αλγόριθμος που εισήχθηκε στον χώρο με βάση το άνω όριο αποτελεί τον αλγόριθμο ο οποίος χρησιμοποιείται για την εξέταση του παραλληλισμού πάνω στα ερωτήματα.

3.2 Parallel and Distributed Processing of Reverse Top-K Queries

Στην πιο πρόσφατη εργασία των Panagiotis Nikitopoulos et al με τίτλο *Parallel and Distributed Processing of Reverse Top-k Queries*[14] αντιμετωπίζουν για πρώτη φορά το πρόβλημα της επεξεργασίας των ερωτημάτων αντίστροφων κορυφαίων-k σε παράλληλα και κατανεμημένα περιβάλλοντα. Ο λόγος είναι ότι παρά τους όποιους αποδοτικούς αλγόριθμους που υπάρχουν, ο υπολογισμός ενός ερωτήματος αντίστροφων κορυφαίων-κ παραμένει παραμένει ιδιαίτερα εντατικός στον επεξεργαστή, με αποτέλεσμα να μην είναι βιώσιμο σε μεγάλα σύνολα δεδομένων λόγο του ακριβού κόστος επεξεργασίας. Προτείνουν έναν αλγόριθμο παράλληλης επεξεργασίας, που ονομάζεται Di-PaRT (Distributed and Parallel Reverse Top-k algorithm), το οποίο βασίζεται στο μοντέλο Map-Reduce και υπολογίζει ερωτήματα αντίστροφων κορυφαίων-k σε κατανεμημένα περιβάλλοντα και στην συνέχεια παρέχουν μια υλοποίηση του σε Hadoop την οποία υλοποίηση αξιολογούν σε μεγάλα σύνολα δεδομένων.

Η συγκεκριμένη ερευνητική εργασία αποτέλεσε την πηγή έμπνευσης πίσω από την έρευνα σχετικά με την υλοποίηση αλγόριθμων αντίστροφων κορυφαίων-κ πάνω σε συστήματα με πολλούς επεξεργαστές / πυρήνες.

3.3 Continuous Monitoring of Top-k Queries over Sliding Windows

Στην εργασία των Kyriakos Mouratidis et al το 2006 με τίτλο *Continuous Monitoring of Top-k Queries over Sliding Windows*[13] αντιμετώπιστηκε το πρόβλημα της μεθόδου υπολογισμού ερωτημάτων κορυφαίων k σημείων σε ροές δεδομένων. Συγκεκριμένα μελετούν τη συνεχή παρακολούθηση των ερωτήσεων κορυφαίων-k σε ένα παράθυρο σταθερού μεγέθους W των πιο πρόσφατων δεδομένων. Το μέγεθος του παραθύρου μπορεί να εκφραστεί είτε ως προς τον αριθμό των ενεργών πλειάδων είτε ως μονάδες χρόνου. Είναι σημαντικό να αναφέρουμε ότι τα δεδομένα που βρίσκονται στο παράθυρο W προσπελαύνονται μέσα από την κύρια μνήμη του συστήματος, ότι τα έγκυρα αρχεία είναι ευρετηριασμένα από ένα ευρετήριο πλέγματος και πως η υψηλότερη βαθμολογία είναι καλύτερη σε σχέση με μία χαμηλότερη βαθμολογία.

Στην συνέχεια ορίζει αυτό που ονομάζεται περιοχή επιρροής ενός σημείου. Δεδομένου ενός ερωτήματος q το οποίο χαρακτηρίζεται με την μορφή μιας συνάρτησης $f(x_1, x_2)$ η οποία αποκαλείται και αλλιώς μέθοδος υπολογισμού της βαθμολογίας και ενός σημείου p_k όπου k είναι η χαμηλότερη-k βαθμολογία η γραμμή που προκύπτει από αυτή την συνάρτηση $score(p_k) = f(x_1, x_2)$ χωρίζει τον χώρο σε δύο μέρη εκ των οποίων ο χώρος με τα σημεία που υπερβαίνουν την τιμή της βαθμολογίας είναι η περιοχή επιρροής του σημείου. Οι αλλαγές στην περιοχή επιρροής επηρεάζουν το αποτέλεσμα του ερωτήματος q, και επιπλέον η περιοχή επιρροής έχει σταθερό μέγεθος οπότε η προσθήκη μιας πλειάδας μειώνει το διαθέσιμο μέγεθος του χώρου, ενώ η αφαίρεση μια πλειάδας προκαλεί την αύξηση του χώρου.

Ορίζεται επίσης ένα σημείο το οποίο είναι η μέγιστη δυνατή βαθμολογία που μπορεί να λάβει μια οποιαδήποτε πλειάδα για κάθε scoring μέθοδο και κυριαρχεί έναντι οποιασδήποτε άλλης πλειάδας που υπάρχει στον χώρο.

Για να γενικεύσουμε σε αυθαίρετες τιμές του k, χρησιμοποιείται η έννοια του k-skyband. Ειδικότερα το k-skyband περιέχει τις πλειάδες που κυριαρχούνται από τα περισσότερα k-1 άλλα σημεία. Σύμφωνα με αυτόν τον ορισμό, το skyline είναι μια ειδική περίπτωση του skyband, όπου k=1.

Η αναγωγή από τα ερωτήματα κορυφαίων-k σε k-skyband εφαρμόζεται και στα δύο είδη συρόμενων παραθύρων (δηλαδή, με βάση την καταμέτρηση και με βάση το χρόνο) και είναι ανεξάρτητη από τη διαστατικότητα d, που σημαίνει ότι το skyband υπολογίζεται πάντοτε στο δισδιάστατο χώρο χρόνου / βαθμολογίας ακόμη και όταν τα δεδομένα έχουν πολλά (> 2) χαρακτηριστικά κάτι που εκμεταλλεύεται ο αλγόριθμος SMA που περιγράφεται παρακάτω.

Ο πρώτος αλγόριθμος ο οποίος προτείνεται είναι ο Top-k Monitoring Algorithm(TMA). Ο αλγόριθμος αποτελείται ουσιαστικά από δυο modules το computation module και το maintenance module. Το πρώτο είναι υπεύθυνο για τον αρχικό υπολογισμό του σετ των κορυφαίων-k ενώ το δεύτερο είναι ένας αποτελεσματικός μηχανισμός για την ενημέρωση του αποτελέσματος με την παρουσία εισαγωγών / διαγραφών. Πέρα από τους δυο μηχανισμούς υπάρχει μια ουρά η οποία οφείλει να διαχειρίζεται τα σημεία τα οποία εισάγονται και να διαγράφει τα σημεία που «έληξαν», και ένα ευρετήριο πλέγματος όπου για κάθε κελί του πλέγματος, συντηρείται η λίστα με τα ερωτήματα εκείνα που το συγκεκριμένο κελί (ή χώρος δεδομένων) αποτελεί περιοχή επιρροής. Στην αρχή εκτελείται το computation module το οποίο υπολογίζει το σύνολο των πλειάδων που αποτελούν σύνολο του αποτελέσματος και κατά την διάρκεια που τρέχει ο αλγόριθμος το computation module υπολογίζει καινούργια ερωτήματα q τα οποία μπορεί να εισαχθούν ενώ το maintenance module για κάθε καινούργια πλειάδα ή πλειάδα η οποία έχει λήξη ανανεώνει τα αποτελέσματα και τις περιοχές επιρροής για κάθε q εφόσον χρειάζεται
.

Ο SMA κάνει το εξής παραπάνω βήμα έτσι ώστε να βελτιώσει την απόδοση του TMA το οποίο είναι να συντηρεί στην μνήμη του για κάθε q ένα skyband ερώτημα μεγέθους. Αυτό έχει ως αποτέλεσμα μαζί με τις περιοχές επιρροής για κάθε ερώτημα να χρειάζεται να ανανεώνεται το skyband του όταν εισάγεται/εξάγεται μια πλειάδα η οποία επηρεάζει μια περιοχή επιρροής.

3.4 Most Influential Data Objects with Reverse Top-K Queries

Στην εργασία με τίτλο *Most Influential Data Objects with Reverse Top-K Queries*[19] των Akrivi Vlachou et al οι συγγραφείς ασχολούνται με το πρόβλημα της αναγνώρισης των σημείων που κατέχουν την μεγαλύτερη επιρροή έναντι των υπόλοιπων σημείων που περιέχει το σύνολο δεδομένων. Συγκεκριμένα παρουσιάζουν δυο αλγόριθμους για τον εντοπισμό των πιο σημαντικών αντικείμενων στα δεδομένα. Ο πρώτος αλγόριθμος ονομάζεται *Skyband-Based* και εκμεταλλεύεται τις ιδιότητες του skyband-συνόλου με στόχο τον περιορισμό του αριθμού των υποψήφιων αντικειμένων. Ο δεύτερος

είναι ένας branch and bound αλγόριθμος ο οποίος εφαρμόζει άνω όρια πάνω στο σκορ επιρροής και την κατανομή των αποτελεσμάτων για την αναφορά του αποτελέσματος σταδιακά. Επιπλέον εισάγουν παραλλαγές των κορυφαίων-m πιο σημαντικών αντικειμένων (δηλαδή που έχουν πιο πολλή επιρροή) οι οποίες έχουν νόημα για πολλές εφαρμογές, είναι χρήσιμα στην πράξη και υποστηρίζονται μέσα από τον branch and bound αλγόριθμο. Τέλος αποδεικνύουν θεωρητικά και πρακτικά μέσω πειραματισμού σχετικά με την ορθότητα των αλγορίθμων και την αποδοτικότητα τους.

Αν και σε αυτή την εργασία δεν χρησιμοποιήθηκαν R-trees ή skylines, καθώς χρησιμοποιήθηκαν κυρίως grids και ο τρόπος με τον οποίο εντόπιζα τα σημεία με την μεγαλύτερη επιρροή ήταν μέσω της απόστασης του κέντρου του grid από την αρχή των αξόνων, ήταν ενδιαφέρον ο τρόπος με τον οποίο χρησιμοποιήθηκαν οι δομές δεδομένων για να τα εντοπίσουν.

3.5 Reverse Top-K for streaming data

Στην Διπλωματική διατριβή [15] για το Π.Μ.Σ. «Διδακτική της Τεχνολογίας και Ψηφιακών Συστημάτων» του Νικητόπουλου Παναγιώτη εισάγεται και αναλύεται για πρώτη φορά το πρόβλημα της μεθόδου επίλυσης ερωτήματα αντίστροφων κορυφαίων k σημείων σε ροές δεδομένων. Συγκεκριμένα προτείνει ένα μοντέλο το οποίο θα ειδικεύεται στην επίλυση ερωτημάτων τέτοιου τύπου το οποίο θα εφαρμόζει του περιορισμούς αλλά και τα πλεονεκτήματα των ροών δεδομένων. Αναλυτικότερα σχεδιάζει μια αρχιτεκτονική η οποία θα χειρίζεται δεδομένα μεγάλου όγκου (Big Data) η οποία αποτελείται από 3 μέρη:

- Μονάδα προ-επεξεργασίας, η οποία αφορά τα σημεία για τα οποία θα υπολογίζουμε τα ερωτήματα αντίστροφων κορυφαίων-κ.
- Μονάδα Εισόδου, η οποία αφορά τον χειρισμό των δεδομένων που προστίθενται στο σύνολο δεδομένων.
- Μονάδα Εξόδου, η οποία αφορά τον χειρισμό των δεδομένων τα οποία «λήγουν» δηλαδή παύουν να αποτελούν μέλος το συνόλου δεδομένων.

Μαζί με την αρχιτεκτονική σχεδιάζει 3 αλγόριθμους όπου ο καθένας υλοποιεί με διαφορετικό τρόπο τις παραπάνω μονάδες με στόχο την μεγαλύτερη αποδοτικότητα τους οποίους συγκρίνει και αξιολογεί.

Η ερευνητική διατριβή του Νικητόπουλου Παναγιώτη αποτέλεσε πηγή έμπνευσης για τον Layered αλγόριθμο (βλέπε κεφάλαιο 4.2.5) ο οποίος αναλύεται στο κεφάλαιο των αλγορίθμων και για την αρχιτεκτονική της υλοποίησης (βλέπε κεφάλαιο 4.1) η οποία αναλύεται εκτενώς στο κεφάλαιο του πειραματισμού.

3.6 Determining the impact regions of competing options in preference space

Οι Bo Tang et al στην εργασία τους με τίτλο *Determining the impact regions of competing options in preference space*[16] σε αντίθεση με τις προηγούμενες εργασίες που ασχολούνται με ερωτήματα όπου δίνεται ένα πεπερασμένο σύνολο διανυσμάτων προτιμήσεων και ο στόχος είναι να προσδιοριστούν τα διανύσματα που κατατάσσουν μια καταγραφή ως την υψηλότερη, εξετάζουν στην θέση των διακριτών προτιμήσεων συνεχόμενα διαστήματα προτιμήσεων.

Συγκεκριμένα ο στόχος της εργασίας τους είναι να αντιμετωπιστεί το πρόβλημα εύρεσης όλων των περιφερειών στον χώρο προτιμήσεων όπου μια καταγραφή αρχείο p ανήκει στη σύσταση κορυφαίων-k ή αλλιώς k-Shortlist Preference Region identification (kSPR) όσο πιο αποτελεσματικά γίνεται.

Μοντελοποιούν το kSPR ως υπολογιστικό γεωμετρικό πρόβλημα σε μια διάταξη από hyperplanes και αναπτύσσουν μια δομή δεδομένων που ονομάζουν Cell Tree για να διατηρήσουν αυτή τη διάταξη. Μια βασική αρχή στην προσέγγισή τους είναι ότι η ακριβής γεωμετρία του κάθε Cell στο Cell-Tree της διάταξης δεν υπολογίζεται, εκτός αν είναι εγγυημένη ότι είναι στο αποτέλεσμα του kSPR .

Κάθε καταγραφή r και p συσχετίζεται με ένα hyperplane το οποίο είναι υπεύθυνο για τον διαχωρισμό του χώρου των προτιμήσεων σε δυο συμπληρωματικές περιοχές το θετικό hyperspace και το αρνητικό hyperspace. Το θετικό hyperspace είναι η περιοχή των προτιμήσεων για τις οποίες το r έχει μεγαλύτερη βαθμολογία από το p, ενώ το αρνητικό hyperspace είναι η περιοχή των προτιμήσεων για τις οποίες το p έχει μεγαλύτερη βαθμολογία από το r.

Το Cell-Tree είναι ένα δυαδικό δέντρο με όσα hyperplanes έχουν εισαχθεί μέχρι τώρα. Η ρίζα του δέντρου αντιστοιχεί σε ολόκληρο τον μετασχηματισμένο χώρο προτιμήσεων. Ο ρόλος του Cell-Tree είναι να διατηρεί διαδοχικά τη διευθέτηση Γ καθώς εισάγονται νέα hyperplanes. Ένα Cell (c) που ανήκει στο διευθέτηση Γ έχει ως rank(c) ένα συν το πλήθος των θετικών hyperspace τα οποία καλύπτει.

Ο πρώτος αλγόριθμος που παρουσιάζουν ονομάζεται Cell Tree Approach (CTA). Η κύρια ιδέα στο CTA είναι να καθορίσει κάθε εγγραφή $r[i] \in D$ σε ένα hyperplane h[i] και να εισάγει τα hyperplanes ένα προς ένα στο Cell-Tree. Όταν η χαρτογράφηση ολοκληρωθεί, τα κελιά στο Γ με την $rank(c) \leqslant k$ σχηματίζουν το αποτέλεσμα kSPR .

Ο δεύτερος αλγόριθμος που περιγράφουν ονομάζεται Progressive Cell Tree Approach(P-CTA). Αυτός ο αλγόριθμος γλιτώνει υπολογισμούς με (i) τον έλεγχο της σειρά επεξεργασίας των εγγραφών στο D, δηλαδή τη σειρά με την οποία τα hyperplanes εισάγονται στο Cell-Tree, (ii) αγνοεί τις καταγραφές που δεν μπορούν να επηρεάσουν το αποτέλεσμα kSPR και (iii) επιταχύνοντας τον αλγόριθμο εισαγωγής βασισμένο σε κρίσιμες παρατηρήσεις.

Στην συνέχεια για να ενισχύσουν περαιτέρω την απόδοση του P-CTA, προτείνουν τεχνικές προεπισκόπησης που επιτρέπουν (i) τον έγκαιρο κλάδεμα των μη ενθαρρυντικών Cells, (ii) την έγκαιρη ανίχνευση των Cells που ανήκουν στο αποτέλεσμα του kSPR . Ονομάζουν την παραγόμενη μέθοδο ως Look-ahead Progressive Cell Tree Approach (LP-CTA). Τέλος αποδεικνύουν ότι παρά την κοινή ασυμπτωτική πολυπλοκότητα των δύο αλγορίθμων, στην πράξη ο LP-CTA είναι δύο φορές σε μια τάξη μεγέθους ταχύτερος από ότι ο P-CTA .

3.7 Processing a Large Number of Continuous Preference Top-k Queries

Στην εργασία με τίτλο *Processing a Large Number of Continuous Preference Top-k Queries*[21] οι Yu Albert et al θεωρούν το πρόβλημα της επεξεργασίας ενός μεγάλου αριθμού συνεχών ερωτημάτων κορυφαίων-k, το καθένα με δική του προτίμηση. Στόχος τους είναι μια κλιμακούμενη και ολοκληρωμένη λύση στο πρόβλημα της επεξεργασίας ενός μεγάλου αριθμού συνεχόμενων ερωτημάτων κορυφαίων-k, όπου τα αποτελέσματα των ερωτημάτων θα ενημερωθούν, όταν αλλάζουν αντικείμενα ή προτιμήσεις χρήστη.

Σε αυτή την εργασία, παρέχουν μια διαισθητική ερμηνεία του k-οστού επιπέδου ως ένα query response surface (QRS), το οποίο αντιπροσωπεύει γεωμετρικά το αντικείμενο που έχει ταξινομηθεί ως k-οστό πάνω από το χώρο όλων των πιθανών προτιμήσεων. Το δεύτερο θεμελιακό στοιχείο πάνω στο οποίο βασίζεται η εργασία είναι τα halfspace range queries δηλαδή queries πάνω σε περιοχές που χωρίζονται μέσα από hyperplanes.

Επιπλέον παρουσιάζουν ένα δυναμικό ευρετήριο που υποστηρίζει το ερώτημα αντίστροφο κορυφαίων-k, το οποίο έχει ανεξάρτητο ενδιαφέρον. Συνδυάζοντας αυτό το ευρετήριο με ένα άλλο για τα ερωτήματα κορυφαίων-k, αναπτύσσουν μια κλιμακούμενη λύση για την επεξεργασία πολλών συνεχόμενων ερωτημάτων κορυφαίων-k που εκμεταλλεύονται τη συσσωμάτωση στις προτιμήσεις των χρηστών. Συγκεκριμένα συσχετίζουν τα ερωτήματα αντίστροφων κορυφαίων-k με τα halfspace range queries ώστε να αξιοποιήσουν τα αποτελέσματα της, για να δημιουργήσουν ένα ευρετήριο για τα αντίστροφα ερωτήματα κορυφαίων-k, τα οποία, χρησιμεύουν ως κρίσιμο συστατικό της λύσης στο πρόβλημα της κλιμακούμενης επεξεργασίας συνεχόμενων ερωτημάτων κορυφαίων-k.

Εκτός αυτού προτείνουν μια υβριδική προσέγγιση η οποία συνδυάζει τα θετικά της επεξεργασία με προτίμηση και τα θετικά της επεξεργασία με QRS, όπου χρησιμοποιούν επεξεργασία ως προς τις προτιμήσεις για περιοχές όπου υπάρχουν λίγες ή καθόλου προτιμήσεις και επεξεργασία μέσω QRS για πυκνές συστοιχίες προτιμήσεων

Τέλος καθορίζουν μια προσεγγιστική έκδοση για το πρόβλημα και παρουσιάζουν μια λύση που είναι σημαντικά πιο αποτελεσματική από την ακριβή με μικρή απώλεια ακρίβειας. Συγκεκριμένα, χρησιμοποιούν την έννοια των coresets, η οποία έχει χρησιμοποιηθεί με επιτυχία για τους αλγόριθμους γεωμετρικής προσέγγισης για να διατηρηθεί ένα μικρό υποσύνολο αντικειμένων που προκαλούν ένα QRS τέτοιο ώστε να προσεγγίζει το QRS που επηρεάζεται από ολόκληρο το σύνολο αντικειμένων. Παρατηρούν ότι το μέγεθος του υποσυνόλου εξαρτάται μόνο από το k και το σφάλμα προσέγγισης, και όχι από τον αριθμό των αντικειμένων.

3.8 Efficient All Top-k Computation

Οι Shen Ge et al στην εργασία τους *Efficient All Top-k Computation*[11] εξετάζουν λύσεις ώστε να αξιολογούν κορυφαία-k ερωτήματα σε παρτίδες, εκμεταλλευόμενοι το γεγονός ότι παρόμοια ερωτήματα μοιράζονται κοινά αποτελέσματα για να επιτύχουν καλύτερες ταχύτητες, έναντι του να εξετάσουν τα ερωτήματα κορυφαίων-k ένα προς ένα, κάτι το οποίο δεν κλιμακώνει καλά. Συγκεκριμένα αυτό επεκτείνεται στα ερωτήματα αντίστροφων κορυφαίων-k τα οποία δεν κλιμακώνουν καλά. Ο λόγος όπως έχει ήδη αναφερθεί είναι επειδή κάθε ερώτημα αντίστροφων κορυφαίων-k απαντά με βάση την εκτέλεση ενός συνόλου βασικών ερωτημάτων κορυφαίων-k. Εάν έχουν εκδοθεί πολλαπλά ερωτήματα αντίστροφων κορυφαίων-k, ορισμένα από αυτά τα ερωτήματα κορυφαίων-k ενδέχεται να εκτελούνται ακόμη και πολλές φορές.

Ο πρώτος αλγόριθμος τους εφαρμόζει το λεγόμενο block indexed nested loops paradigm. Αυτή η μέθοδος μπορεί να θεωρηθεί ως το αντίστοιχο block indexed nested-loops σε σχεσιακές βάσεις δεδομένων και spatial join queries σε χωρικές βάσεις δεδομένων. Σε αυτή την τεχνική ευρετηριάζονται όχι μόνο τα αντικείμενα του συνόλου δεδομένων με ένα πολυδιάστατο δείκτη όπως το R*-Tree αλλά και τις προτιμήσεις τις οποίες διαμερίζουμε σε ομάδες. Ειδικότερα για να ομαδοποιήσουμε τις προτιμήσεις τις ταξινομούμε αρχικά με βάση την θέση τους στην καμπύλη Hilbert, που ευρετηριάζει το χώρο των συντελεστών λειτουργίας.

Στη συνέχεια, διαιρούμε την καμπύλη σε δευτερεύοντα υποδείγματα, καθένα από τα οποία ορίζει μια ομάδα, έτσι ώστε κάθε ομάδα να περιέχει μόνο μια αναλογία δ των προτιμήσεων. Διαισθητικά, μια ομάδα περιέχει έναν μικρό αριθμό παρόμοιων προτιμήσεων που θα μοιράζονταν μερικά αποτελέσματα. Η επεξεργασία των προτιμήσεων της ομάδας ταυτόχρονα θα ήταν ταχύτερη από την εκτέλεση των ερωτημάτων ξεχωριστά, καθώς κάποιο κόστος αναζήτησης θα μοιραζόταν μεταξύ των προτιμήσεων της ομάδας.

Η δεύτερη τεχνική τους είναι ένας αλγόριθμος με βάση τα views, όπου ένα view είναι πολύ απλά το αποτέλεσμα ενός ερωτήματος κορυφαίων-k. Συγκεκριμένα εφαρμόζει ένα καλά αποδεκτό γενικό

πρότυπο για την αποτελεσματική επεξεργασία ερωτημάτων, για διαφορετικά είδη δεδομένων και ερωτημάτων, δηλαδή να επωφεληθούν από υλοποιημένα views με προ-υπολογισμένα αποτελέσματα. Για την υποστήριξη της επεξεργασίας παρτίδας, όταν ένα αντικείμενο p έχει πρόσβαση από μια προβολή, μπορούμε να αξιολογήσουμε τις βαθμολογίες της προβολής για πολλαπλά ερωτήματα κορυφαίων-k. Έτσι για κάθε ερώτημα κορυφαίων-k που αξιολογείται, ενημερώνουμε το τρέχον σύνολο αποτελεσμάτων, εάν είναι απαραίτητο. Μια προτίμηση έχει επισημανθεί ως σταματημένη εάν ο k-οστός υποψήφιος δεν είναι χειρότερος από τη μέγιστη δυνατή βαθμολογία. Σε κάθε επανάληψη του BLPTA, φέρουμε το επόμενο αντικείμενο p από μία από τις προβολές με την μορφή round-robin και ενημερώνουμε τα υποψήφια κορυφαία-k για κάθε μια από τις τρέχουσες λειτουργίες.

Κεφάλαιο 4

Μοντέλο επίλυσης ερωτημάτων αντίστροφων κορυφαίων-k

Στα πλαίσια της πτυχιακής θέλουμε να επιλύσουμε όσο πιο αποδοτικά γίνεται το ερώτημα αντίστροφων κορυφαίων-k με παραλληλισμό, οπότε υλοποιήθηκε μια λύση η οποία υπολογίζει ένα ερώτημα αντίστροφων κορυφαίων-k αναθέτοντας σε κάθε διεργασία ένα υποσύνολο των προτιμήσεων. Σκοπός είναι να επιτύχουμε όσο πιο υψηλό speedup γίνεται και για αυτό το λόγο τα δεδομένα και οι προτιμήσεις βρίσκονται φορτωμένα στην κύρια μνήμη (RAM memory) ώστε να είναι γρήγορα προσβάσιμες.

Συγκεκριμένα σε αυτό το κεφάλαιο αναλύονται τα επιμέρους τμήματα από τα οποία αποτελείται το μοντέλο επίλυσης ερωτημάτων αντίστροφων κορυφαίων-k. Αυτά είναι η αρχιτεκτονική του συστήματος, οι αλγόριθμοι που υλοποιήθηκαν και οι μέθοδοι με τους οποίους διαχωρίζουμε το πρόβλημα ώστε να τρέχει όσο πιο αποδοτικά γίνεται, δεδομένου των υπολογιστικών πόρων που είναι διαθέσιμοι.

Τέλος αναλύονται τα εργαλεία που χρησιμοποιήθηκαν, προκλήσεις που εμφανίστηκαν κατά την υλοποίηση και περιγραφή της διεπαφή προγραμματισμού εφαρμογών (API) που παρέχει η βιβλιοθήκη που υλοποιήθηκε στα πλαίσια της εργασίας.

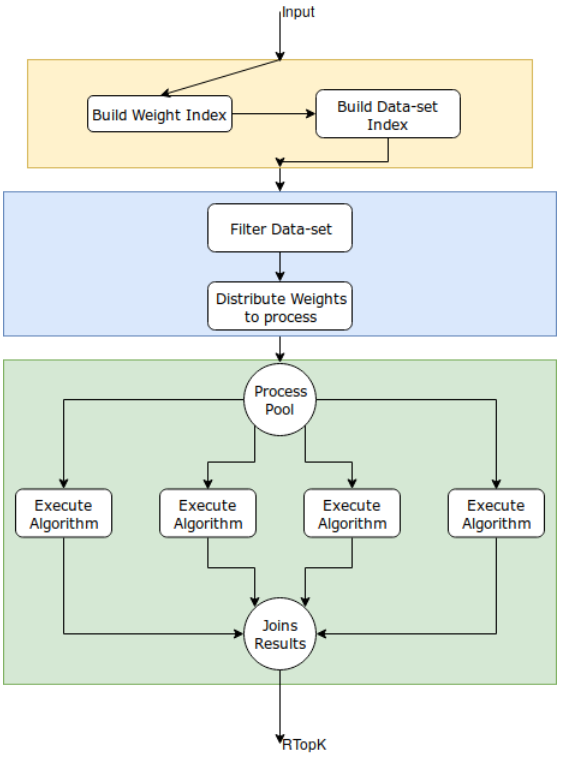
4.1 Αρχιτεκτονική συστήματος

Στο παρακάτω διάγραμμα 4.1 περιγράφεται η ροή της αρχιτεκτονικής που σχεδιάστηκε και υλοποιήθηκε στα πλαίσια της πτυχιακής εργασίας.

Στο πρώτο της στάδιο, που είναι η αρχικοποίηση της, τρέχει τους αλγόριθμους που είναι απαραίτητοι για την κατασκευή των ευρετηρίων που χρειάζονται για την διεξαγωγή του αλγορίθμου. Για παράδειγμα όσον αφορά το σύνολο δεδομένων χρησιμοποιείται ένα ευρετήριο πλέγματος σε αρκετούς αλγόριθμους που αναλύονται παρακάτω. Η αρχικοποίηση δεν αφορά όμως, μόνο την ευρετηρίαση του συνόλου δεδομένων αλλά και την ομαδοποίηση του συνόλου των προτιμήσεων σε υποσύνολα. Ειδικότερα συνήθως γίνεται μια ταξινόμηση των προτιμήσεων και στην συνέχεια δημιουργούνται τα υποσύνολα. Αυτά τα υποσύνολα κατά την διάρκεια εκτέλεσης του ερωτήματος θα διανεμηθούν στην αντίστοιχη διεργασία. Η εκτέλεση του πρώτου σταδίου γίνεται μόνο μια φορά πριν εκτελέσουμε για πρώτη φορά το ερώτημα αντίστροφων κορυφαίων-k και για αυτό τον λόγο δεν την προσμετράμε στον χρόνο εκτέλεσης.

Το δεύτερο στάδιο είναι η εκτέλεση του ερωτήματος. Αρχικά φιλτράρονται τα δεδομένα τα οποία σε σχέση με το αντικείμενο Q δεν επηρεάζουν το αποτέλεσμα ή δεν χρειάζεται να περιέχονται σε ένα ερώτημα κορυφαίων-k για να βγει κάποιο συμπέρασμα ως προς αυτά. Η διαδικασία του φιλτραρί-

σματος περιγράφεται πιο αναλυτικά στην ενότητα 4.2.3. Στην συνέχεια κατανέμονται οι προτιμήσεις ανά διεργασία, όπου η κάθε διεργασία αναλαμβάνει να λύσει το ερώτημα αντίστροφων κορυφαίων-k ως προς το πλήθος των προτιμήσεων που της αναλογεί. Τέλος συνδέονται τα αποτελέσματα της κάθε διεργασίας στο τελικό αποτέλεσμα που παραδίδεται στον χρήστη.



Σχήμα 4.1: Αναπαράσταση της αρχιτεκτονικής του συστήματος

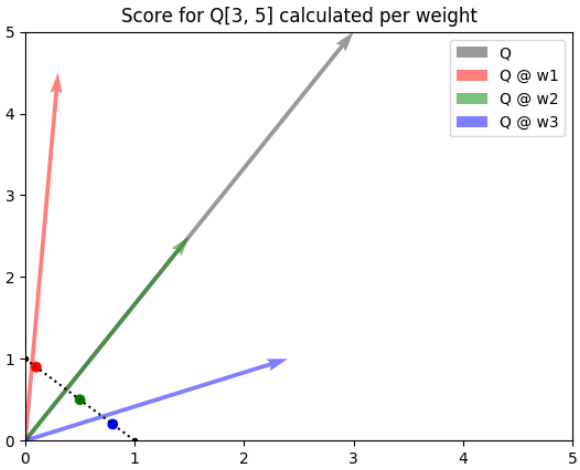
Με **κίτρινο** είναι μαρκαρισμένη η περιοχή του προγράμματος η οποία εκτελείται μόνο μια φορά ώστε να κατασκευάσει τα απαραίτητα indexes που χρειάζονται για τον υπολογισμό του αποτελέσματος. Με **γαλάζιο** είναι μαρκαρισμένες οι ενέργειες που δεν αποτελούν μέρος της εξέτασης ως προς την αποδοτικότητα των αλγορίθμων όπως το φιλτράρισμα. Η ιδέα είναι ότι αυτές είναι οι ενέργειες οι οποίες αποτελούν το σειριακό κομμάτι του κώδικα, δηλαδή το κομμάτι το οποίο δεν παραλληλίζεται. Με **πράσινο** είναι οι ενέργειες που αφορούν τον διαχωρισμό του προβλήματος σε υποπροβλήματα, η εκτέλεση των υποπροβλημάτων και η σύμπτυξη του αποτελέσματος. Αποτελεί το κομμάτι του κώδικα το οποίο είναι παραλληλοποιήσιμο και μαζί με τις ενέργειες που είναι μαρκαρισμένες με **γαλάζιο** αποτελούν το μέρος που μετράμε ως προς την αποδοτικότητα.

4.2 Αλγόριθμοι

Σε αυτή την ενότητα αναλύονται οι αλγόριθμοι που υλοποιήθηκαν μαζί με τα επιμέρους τμήματα τους όπως η βαθμολόγηση, ερωτήματα κορυφαίων-k και φιλτράρισμα στα πλαίσια αυτής της πτυχιακής. Συγκεκριμένα εξηγούμε πως λειτουργεί ένα απλός αφελής (*Naive*) αλγόριθμος ο οποίος προσπαθεί εξαντλητικά να λύσει το ερώτημα που του δίνεται. Εκτός του αφελή αλγόριθμου διατυπώνεται η ιδέα πίσω από τον *Layered* αλγόριθμο ο οποίος απορρίπτει προτιμήσεις καθώς εξετάζει το σύνολο

δεδομένων σε μια προσπάθεια να γλιτώσει πράξεις. Τέλος αναλύεται ο *Reverse top-k Threshold Algorithm* μαζί με μια παραλλαγή του που ονομάζεται *Reverse top-k Threshold Algorithm* με Grid η οποία χρησιμοποιεί πλέγμα ώστε να αποφύγουμε να προσπελάσουμε όλο το σύνολο δεδομένων.

4.2.1 Βαθμολογία



Πίνακας 4.1: Προτιμήσεις

Weight	w[1]	w[2]	Βαθμολογία
w_1	0.1	0.9	4.8
w_2	0.5	0.5	4.0
w_3	0.8	0.2	3.4

Σχήμα 4.2: Αναπαράσταση υπολογισμού της βαθμολογίας

Για τον υπολογισμό της βαθμολογίας μεταξύ ενός σημείου $Q = (x_1, x_2 \dots x_n)$ και ενός βάρους $W = (y_1, y_2 \dots y_n)$ χρησιμοποιούμε το εσωτερικό γινόμενο αυτών:

$$F(Q, W) = Q \cdot W = (x_1 \cdot y_1 + x_2 \cdot y_2 + \dots + x_n \cdot y_n)$$

Στο διάγραμμα 4.2 βλέπουμε ως σημεία στον χώρο τις προτιμήσεις w_1, w_2, w_3 με κόκκινο, πράσινο και μπλε αντίστοιχα. Με μαύρο βέλος αναπαριστούμε το σημείο αναφοράς $Q = (3, 5)$ και με χρωματιστά βέλη έχουμε την βαθμολογία του Q για την αντίστοιχη προτίμηση. Για παράδειγμα έστω ότι έχουμε σημείο στον χώρο $Q = (3, 5)$ και βάρος $w_3 = (0.8, 0.2)$, τότε η βαθμολογία τους θα είναι ίση με $Q.w_3 = (3 \cdot 0.8 + 5 \cdot 0.2) = 3.4$ όπως βλέπουμε στον πίνακα 4.1.

Αυτό σε κώδικα μεταφράζεται ως εξής:

```
1 def score(w, p, D):
2     score_value = 0
3     for d in D:
4         score_value += p[d] * w[d]
5     return score_value
```

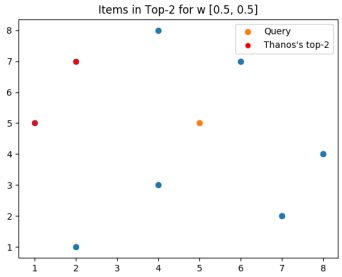
Όμως στον ψευδοκώδικα που ακολουθεί παρακάτω θα το χρησιμοποιούμε με την εξής μορφή **p @ w**. Μια βαθμολογία θεωρείται καλύτερη από μια άλλη δεδομένου ότι έχει μικρότερη τιμή, με άλλα λόγια οι βαθμολογίες κατατάσσονται χρησιμοποιώντας αύξουσα ταξινόμηση.

4.2.2 Top-K

Παρά το ότι κανονικά ένας αλγόριθμος κορυφαίων-k επιστρέφει τα k καλύτερα σημεία που περιέχει το σύνολο δεδομένων εμείς χρησιμοποιούμε μια πιο «ελαφριά» έκδοση του αλγόριθμου η οποία αναζητεί απλά να βρει k σημεία τα οποία είναι καλύτερα από το Q και όχι απαραίτητα τα καλύτερα k. Η υλοποίηση που χρησιμοποιήθηκε είναι η εξής:

```
1  def Top_K(w, S, q, k):
2      q_score = q @ w
3      buffer = list()
4      for p in S:
5          point_score = p @ w
6          # if a tuple from the data-set has a better score
7          # we add it to the buffer of tuples with better
8          # score than q
9          if point_score < q_score:
10             buffer.append(p)
11         # if we have k tuples with better score there is no need
12         # for us to keep processing the data-set
13         if len(buffer) >= k:
14             break
15
16     return buffer
```

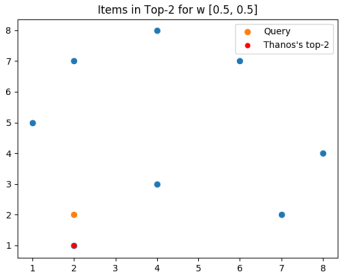
Χρησιμοποιώντας το σύνολο δεδομένων από την προηγούμενη ενότητα 1.1 κοιτάμε για τις ακόλουθες δυο διακριτές περιπτώσεις πως θα αντιδράσει η έκδοση μας του κορυφαίου-k.



Σχήμα 4.3: Query [5, 5]

Point	X	Y
p0	6	7
p1	7	2
p2	1	5
p3	2	7
p4	8	4
p5	4	3
p6	4	8
p7	2	1

Πίνακας 4.2: Δεδομένα



Σχήμα 4.4: Query [2, 2]

Στην πρώτη περίπτωση στο διάγραμμα αριστερά 4.3, βλέπουμε το top-2 για τον χρήστη Thanos από το αρχικό πίνακα με τις προτιμήσεις 1.2 και ένα αντικείμενο με χαρακτηριστικά $[5, 5]$. Βλέπουμε ότι ο αλγόριθμος αναγνωρίζει ότι στον χώρο υπάρχουν τουλάχιστον 2 πλειάδες με καλύτερη βαθμολογία και συγκεκριμένα οι πλειάδες $[1, 5], [2, 7]$.

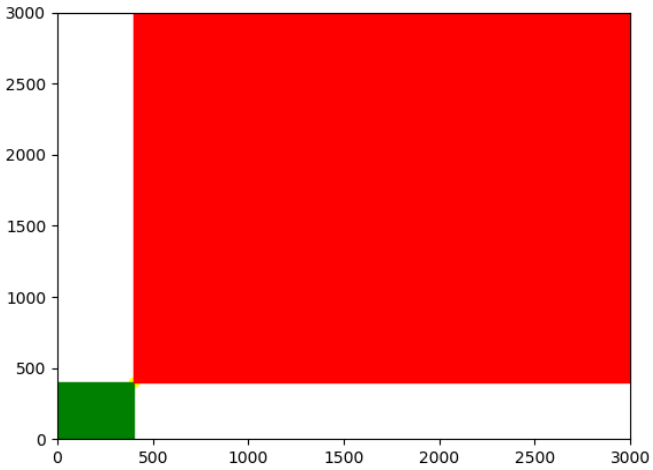
Παρόλα αυτά ενώ υπάρχουν πλειάδες ακόμα καλύτερες από τις 2 παραπάνω και συγκεκριμένα η πλειάδα $[2, 1]$ δεν μας επηρεάζει ως προς το αποτέλεσμα και επιτύχαμε να αποφύγουμε να προσπελάσουμε όλες τις πλειάδες για να γνωρίζουμε ότι το $[5, 5]$ δεν ανήκει στα top-2 του Thanos.

Στην δεύτερη περίπτωση στο διάγραμμα δεξιά 4.4, βλέπουμε το top-2 πάλι για τον χρήστη Thanos, όμως έχουμε ένα αντικείμενο με πολύ πιο ανταγωνιστικά χαρακτηριστικά $[2, 2]$ σε σχέση με την πρώτη

περίπτωση. Αυτή την φορά το μόνο αντικείμενο που επιστρέφει ο αλγόριθμος είναι η πλειάδα $[2, 1]$ και επειδή είναι μόνο μια η πλειάδα γνωρίζουμε ότι το αντικείμενο $[2, 2]$ ανήκει και αυτό στο αποτέλεσμα του top-2 ερωτήματος.

4.2.3 Φιλτράρισμα του συνόλου δεδομένων

Στο κομμάτι της πτυχιακής χρησιμοποιήθηκαν 2 κυρίως φίλτρα για το «κοσκίνισμα» σημείων στα οποία δεν χρειάζεται να κάνουμε περιττούς Top K υπολογισμούς



Σχήμα 4.5: Περιοχές Dominance(κόκκινο) και Anti-Dominance(πράσινο)

Ορισμός Κυριαρχία (Dominance): Δεδομένου δυο σημείων $p, q \in R$, σε d διαστάσεων χώρο, το p κυριαρχεί στο q δηλαδή $p < q$, εάν $\forall i \in 1 \dots d$ όπου $i \neq j$, ισχύει $p[i] \leqslant q[i]$ και $\exists$ τουλάχιστον ένα j για το οποίο ισχύει $p[j] < q[j]$.

Στην εικόνα έχουμε ως $Q = (400, 400)$ το σημείο στον χώρο οπότε προκύπτουν οι δύο χώροι anti-dominance region (ADR) με πράσινο και dominance region (DR) με κόκκινο αντίστοιχα.

Από τις ιδιότητες της κυριαρχίας, ισχύει για το εσωτερικό γινόμενο ότι για το ίδιο W βάρος και δυο διαφορετικά σημεία $Q_1 = (3, 5)$ και $Q_2 = (7, 7)$ το $F(Q_1, W) < F(Q_2, W)$ για κάθε W.

Άρα όταν ένα σημείο το οποίο ανήκει στο anti-dominance region του Q κυριαρχεί έναντι του Q, οπότε γνωρίζουμε ότι θα έχει καλύτερη βαθμολογία από το Q για κάθε προτίμηση με αποτέλεσμα να έχει μεγαλύτερη θέση σε όλα τα ερωτήματα κορυφαίων-k που θα τρέξουν. Κατά συνέπεια δεν χρειάζεται να το εξετάσουμε και απλά μειώνουμε την τιμή του K κατά 1 για κάθε σημείο που ανήκει στο anti-dominance region. Αντίθετα τα σημεία που ανήκουν στο dominance region είναι αυτά στα οποία κυριαρχεί το Q, με αποτέλεσμα να έχουν χειρότερη βαθμολογία για όλες τις προτιμήσεις, οπότε απλά δεν χρειάζεται να εξετάσουμε εκείνα τα σημεία.

4.2.4 Naive Reverse Top K

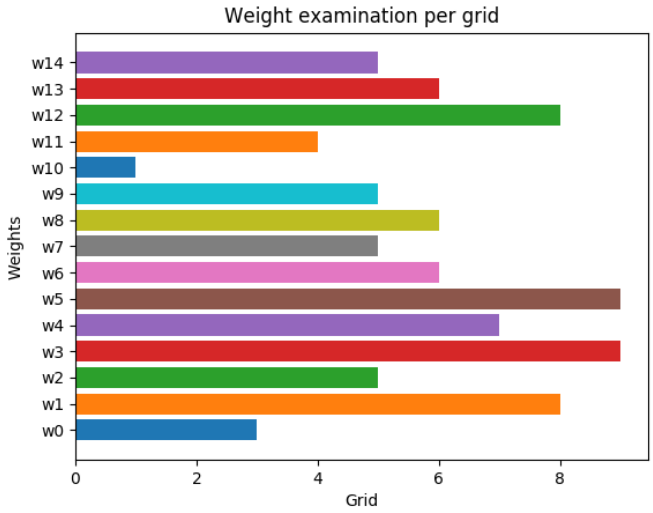
Δεδομένου ενός πλήθους από προτιμήσεις W, ενός συνόλου δεδομένων S από σημεία στον χώρο και ενός σημείου Q ο στόχος του αλγόριθμου είναι να βρει για την κάθε προτίμηση w, αν ως προς

το υπόλοιπο σύνολο δεδομένων η βαθμολογία του Q ανήκει στις κορυφαίες-k βαθμολογίες. Αν ανήκει στις κορυφαίες-k βαθμολογίες τότε η προτίμηση w ανήκει στο αποτέλεσμα του ερωτήματος αντίστροφων κορυφαίων-k. Το αποτέλεσμα αυτού είναι να εκτελεί ένα ερώτημα κορυφαίων-κ για κάθε προτίμηση το οποίο τον καθιστά «αφελή». Αυτό σε κώδικα μεταφράζεται ως εξής:

```
def naive(W, S, q, k):
    # list containing the weights the rtopk returns
    r_top_k_result = list()
    for w in W:
        # find up to k tuples from the data-set with better score
        # for a certain weight
        top_k_points = Top_K(w, S, q, k)
        # if the number of tuples found is less than K
        # then the weight belongs to the reverse top-k result
        if len(top_k_points) < k:
            r_top_k_result.append(w)

    return r_top_k_result
```

4.2.5 Layered

Ο Layered αλγόριθμος είναι ένας απλός αλγόριθμος εμπνευσμένος από τον πίνακα Counter που χρησιμοποιεί στους αλγόριθμους του για δεδομένα ροής ο κ, Νικητόπουλος.[15] Η ιδέα είναι ότι δημιουργούμε ένα ευρετήριο πλέγματος και καθώς το προσπελαύνουμε, από τα καλύτερα σημεία (σημεία που θα φέρουν χαμηλότερη βαθμολόγηση) στα χειρότερα, του συνόλου δεδομένων που δίνεται, παρατηρούμε από νωρίς ότι κάποιες προτιμήσεις δεν θα αποτελούν κομμάτι του αποτελέσματος του ερωτήματος. Οπότε αυτές οι προτιμήσεις απορρίπτονται σύντομα και δεν εξετάζονται για ολόκληρο το σύνολο δεδομένων.



Σχήμα 4.6: Εξεταζόμενες περιοχές ανά πλέγμα

Χρησιμοποιώντας το διάγραμμα 4.6 ως παράδειγμα μπορούμε να δούμε ότι στο πρώτο πλέγμα από το ευρετήριο πλεγμάτων εξετάζουμε όλες τις προτιμήσεις αν ανήκουν στο αποτέλεσμα του ερωτήματος αντίστροφων κορυφαίων-k. Καθώς όμως προχωράμε στο δεύτερο πλέγμα δεν εξετάζουμε την προτίμηση w_{10} γιατί από ότι φαίνεται στο πρώτο πλέγμα υπήρχαν περισσότερες από k πλειάδες οι οποίες είχαν καλύτερη βαθμολογία από την εξεταζόμενη πλειάδα Q. Καθώς φτάνουμε στο 3ο πλέγμα απορρίπτεται και η w_0 προτίμηση, μέχρι που στο τέλος στο 9 πλέγμα οι μόνες προτιμήσεις που έχουν απομείνει είναι η προτίμηση w_3 και w_5.

Είναι σημαντικό να αναφέρουμε, ότι καθώς μετακινούμαστε από πλέγμα σε πλέγμα μεταφέρουμε το πλήθος των πλειάδων που έχουν βρεθεί να έχουν καλύτερη βαθμολογία από το Q. Πηγαίνοντας πίσω στο προηγούμενο παράδειγμα, έστω ότι το αρχικό ερώτημα ήταν να βρούμε το $RTOP_{10}(Q)$ οπότε το w_0 πριν το πρώτο πλέγμα χρειαζόταν 10 πλειάδες καλύτερες από το Q, και αν βρήκε στο 1ο πλέγμα 6 πλειάδες καλύτερες από το Q τότε στο δεύτερο πλέγμα χρειαζόταν 4 πλειάδες για να απορριφθεί από το αποτέλεσμα.

Σε ψευδοκώδικα αυτό μεταφράζεται ως:

```
def layered(W, GridIndex, q, k):
    weight_matrix = list()
    for w in W:
        weight_matrix.append([w, k])

    # The matrix should look like this
    # weight_matrix = [
    #     [w_1, k],
    #     [w_2, k],
    #     [w_3, k],
    #     ...,
    #     [w_4, k],
    # ]


    for grid in GridIndex:
        # this is a flag for checking if we have to
        # prune the matrix
        should_delete = False

        for row in weight_matrix:
            w = weight_matrix[row][0]
            k_remaining = weight_matrix[row][1]
            # we maintain the number of needed tuples in order
            # to invalidate the weight from the result
            top_k_points = Top_K(w, grid, q, k_remaining)
            # updates the number of needed tuples to
            # invalidate the weight
            weight_matrix[row][1] = weight_matrix[row][1] - size(
    top_k_points)
            # if one of the weights need to be removed
            # then we need to prune the matrix
```

```
            if weight_matrix[row][1] <= 0:
                should_delete = True

        # we need to delete the rows where the second column
        # is less or equal to zero since we found enough tuples
        # in the data-set to invalidate the weight from being in
    the result.
        if should_delete:
            remove rows in weight_matrix where rows[1] <= 0

        # if the matrix is empty there are no more weights
        # to examine
        if len(weight_matrix) == 0:
            break
     # we return the weights in the weight matrix
     return [row[0] for row in weight_matrix]
```

4.2.6 Reverse top-k Threshold Algorithm

Ο Reverse top-k Threshold Algorithm (RTA) είναι ένας πιο εκλεπτυσμένος αλγόριθμος ο οποίος έχει ως στόχο να μειώσει το πλήθος των κορυφαίων-k πράξεων που γίνονται κατά την διάρκεια του ερωτήματος καθώς αυτά είναι το πιο δαπανηρό κομμάτι ως προς τον χρόνο της διαδικασίας.[18]

Όπως περιγράψαμε τον αλγόριθμο στο κεφάλαιο Reverse Top-K Queries, ο RTA ορίζει ένα κατώφλι (threshold) το οποίο χρησιμοποιείται για να απορρίπτονται πολλές υποψήφιες προτιμήσεις οι οποίες είναι παρόμοιες με προηγούμενες ήδη εξετασμένες προτιμήσεις και δεν αποτελούν μέρος του αποτελέσματος του ερωτήματος.

Συγκεκριμένα δημιουργούμε ένα buffer το οποίο περιέχει πλειάδες, οι οποίες είχαν την μεγαλύτερη επιρροή στο αποτέλεσμα σχετικά με την προηγούμενη προτίμηση, οπότε τις χρησιμοποιούμε για να δούμε αν έχουν αρκετή επιρροή στην επόμενη προτίμηση για να την απορρίψουν.

```
def rta(W, S, q, k):
    # list containing the weights the rtopk returns
    r_top_k_result = list()
    # initial threshold is infinite since its not set
    threshold = infinite
    for w in W:
        # we calculate the top_k query only if
        # the score of q doesn't cross the threshold
        q_score = q @ w
        if q_score < threshold:
            buffer = Top_K(w, S, q, k)
            # if the number of tuples is less than k
            # we can add the weight at the result of the query
            if len(buffer) < k:
                r_top_k_result.append(w)

        # we get the next weight and we calculate the threshold
```

```
        # based on the worst score inside the buffer
        next_weight = get_next_weight()
        if next_weight and len(buffer) == k:
            threshold = max(buffer @ next_weight)

    return r_top_k_result
```

Χρησιμοποιώντας το σύνολο δεδομένων 1.1 και το σύνολο προτιμήσεων 1.2 από την εισαγωγή εξετάζουμε πως θα αντιμετωπίσει ο RTA το $RTOP_2(Q)$ όπου η πλειάδα Q έχει τα χαρακτηριστικά $[2, 5]$. Για την πρώτη προτίμηση η βαθμολογία του Q είναι 3.2 και το threshold είναι άπειρο. Τρέχοντας το ερώτημα $TOP_2(Q, w_1)$ μας επιστρέφει τις εξής πλειάδες [[1, 5],[2, 1]] που σημαίνει ότι υπάρχουν 2 πλειάδες με καλύτερη βαθμολογία από το Q. Υπολογίζουμε τη βαθμολογία της κάθε πλειάδας για την επόμενη προτίμηση οπότε έχουμε τις εξής βαθμολογίες 3 και 1.5 και επιλέγουμε το μεγαλύτερο από τα δυο δηλαδή θέτουμε ως threshold το 3. Για την δεύτερη προτίμηση η βαθμολογία του Q είναι 3.5 και το threshold είναι 3.0. Δεν εκτελούμε λοιπόν το $TOP_2(Q, w_2)$ και επαναχρησιμοποιούμε το προηγούμενο buffer. Υπολογίζουμε τη βαθμολογία της κάθε πλειάδας για την επόμενη προτίμηση οπότε έχουμε τις εξής βαθμολογίες 3.8 και 1.3 και επιλέγουμε το μεγαλύτερο από τα δυο δηλαδή θέτουμε ως threshold το 3.8. Για την τρίτη προτίμηση η βαθμολογία του Q είναι 4.1 και το threshold είναι 3.8. Δεν εκτελούμε λοιπόν το $TOP_2(Q, w_3)$.
Ο RTA στο παραπάνω παράδειγμα εκτέλεσε ένα ερώτημα κορυφαίων-k για ένα ερώτημα αντίστροφων κορυφαίων-k με 3 προτιμήσεις.

Απόδειξη: Αν ένα σημείο ανήκει στα κορυφαία-κ της προτίμησης τότε πρέπει να είναι τουλάχιστον καλύτερο από το χειρότερο κορυφαίο-k σημείο της προηγούμενης επανάληψης αλλιώς υπάρχουν τουλάχιστον k σημεία καλύτερα από αυτό άρα δεν μπορεί να είναι στα κορυφαία-κ της προτίμησης.

4.2.7 Reverse top-k Threshold Algorithm με Grid

Ο RTA με Grid είναι μια απλή εξέλιξη του RTA . Αρχικά τοποθετούμε το data-set σε ένα Grid Index. Η ιδέα είναι ότι προσπαθούμε να δημιουργήσουμε το buffer που θα χρησιμοποιηθεί για τον υπολογισμό του threshold κάνοντας ένα ερώτημα κορυφαίων-κ σε κάθε grid ώστε να αποφύγουμε να κάνουμε iterate για το ερώτημα κορυφαίων-k όλο το data-set.

Αποτελεί μια απλουστευμένη παραλλαγή του GRTA(Grid-based Reverse Top-k Algorithm) χωρίς να χρησιμοποιεί cells τα οποία έχουν προ-υπολογισμένα αντίστροφα κορυφαία-κ ερωτήματα.

Η υλοποίηση σε ψευδοκώδικα μοιάζει ως εξής:

```
def grid_rta(W, GridIndex, q, k):
    # list containing the weights the rtopk returns
    r_top_k_result = list()
    # initial threshold is infinite since its not set
    threshold = infinite
    for w in W:
        item_score = w @ q
        # similarly to rta we calculate the top_k query only if
        # the score of q doesn't cross the threshold
        if item_score < threshold:
            buffer = list()
            remaining_k = k
```

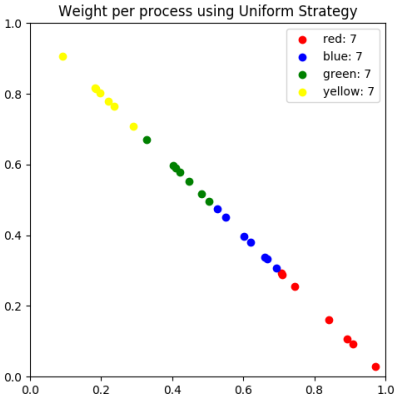
```
            # we try to get the most influential data objects by
            # selecting the grids closer to the weight.
            for grid in GridIndex:
                # here we build up the buffer. By building the
                # buffer using grid by grid we avoid making
                # calculations on tuples that are not
                # influential enough
                buffer += Top_K(w, grid, q, k)
                remaining_k = remaining_k - len(buffer)
                if remaining_k <= 0:
                    # if there are enough tuple to invalidate
                    # the weight there is no need to keep
                    # iterating the GridIndex
                    break

            # if the number of tuples is less than k
            # we can add the weight at the result of the query
            if len(buffer) < k:
                r_top_k_result.append(w)

        # we get the next weight and we calculate the threshold
        # based on the worst score inside the buffer
        next_weight = get_next_weight()
        if next_weight and len(buffer) == k:
            threshold = max(buffer @ next_weight)

    return r_top_k_result
```

4.3 Στρατηγικές Παραλληλισμού

Ο κύριος τρόπος με τον οποίο διαχωρίζουμε τα δεδομένα ανά process είναι μέσω του διαμοιρασμού των βαρών που έχει το κάθε process να εξετάσει. Ο κύριος λόγος που επιλέγουμε αυτή την στρατηγική είναι επειδή δεν χρειάζεται να γνωρίζουμε οποιαδήποτε στιγμή όλα το set από τα weights για να βγάλουμε συμπέρασμα αν ένα βάρος αποτελεί μέλος του αποτελέσματος του Reverse Top-K query.

Ενδιαφέρον αποτελεί η περίπτωση του RTA καθώς το κάθε weight κοιτάει το buffer από το προηγούμενο Weight οπότε πρέπει να γίνει ένα καλός διαχωρισμός των προτιμήσεων ώστε να είναι optimal ο αλγόριθμος. Παρακάτω παρατίθενται οι κύριοι μέθοδοι διαχωρισμού των βαρών για τον RTA/Grid-RTA:

4.3.1 Ομοιόμορφος διαχωρισμός



Σχήμα 4.7: Ομοιόμορφη στρατηγική διαχωρισμού

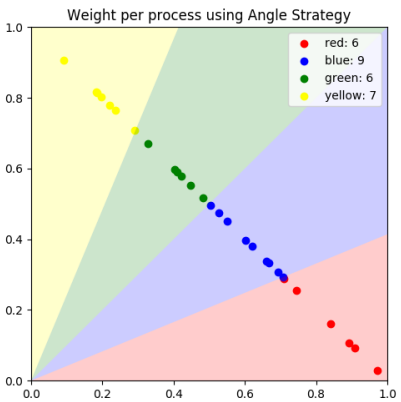
Σε αυτή την στρατηγική υπολογίζουμε την γωνία κάθε προτίμησης w ως προς την αρχή των αξόνων και τις ταξινομούμε. Στην συνέχεια όμως διαχωρίζουμε το ταξινομημένο set από προτιμήσεις σε λ subsets.

Η ιδέα πίσω από αυτή την στρατηγική είναι ότι αναθέτουμε σε κάθε διεργασία ίσο πλήθος από προτιμήσεις για εξέταση και σε μια συνέχεια ώστε να συσχετίζεται το w_i με το w_{i+1} και να είναι βέλτιστο για τον RTA η εξέταση της προτίμησης σε σχέση με το threshold.

4.3.2 Διαχωρισμός ως προς την γωνία

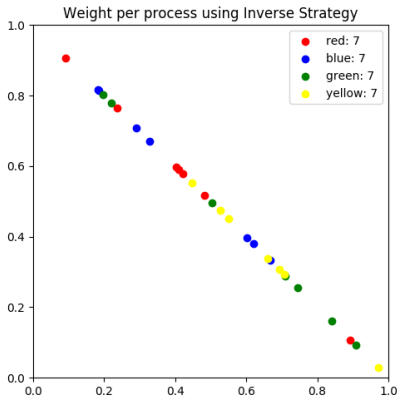
Όμοια με την προηγούμενη στρατηγική υπολογίζουμε την γωνία που δημιουργεί η κάθε προτίμηση w ως προς την αρχή των αξόνων, τις ταξινομούμε και χωρίζουμε το τόξο που δημιουργείται από τις 0 μοίρες ως τις 90 μοίρες στο πρώτο τεταρτημόριο σε λ ίσα τόξα. Στην συνέχεια η κάθε διεργασία αναλαμβάνει ένα από τα υπό-τόξα μαζί με τις προτιμήσεις των οποίων η γωνία είναι εντός αυτών.

Η διαφορά με την προηγούμενη τεχνική είναι ότι έχει ανατεθεί διαφορετικό πλήθος από προτιμήσεις σε κάθε διεργασία ενώ στο 4.3.1 η κάθε διεργασία θα έχει ίσο πλήθος από προτιμήσεις.



Σχήμα 4.8: Στρατηγική διαχωρισμού δια της γωνίας

4.3.3 Αντίστροφος διαχωρισμός



Σχήμα 4.9: Αντίστροφη στρατηγική διαχωρισμού

Αντίθετα με τις προηγούμενες στρατηγικές στο *Inverse Strategy* πρώτα χωρίζουμε τις προτιμήσεις σε λ subsets και στην συνέχεια ταξινομούμε το κάθε subset αυτόνομα, δηλαδή πάλι υπολογίζουμε την γωνία που δημιουργεί η κάθε προτίμηση w ως προς την αρχή των αξόνων και τις ταξινομούμε. Αυτό έχει ως αποτέλεσμα να διαμοιράζουμε το βάρος του υπολογισμού και στις υπόλοιπες διεργασίες και να μην καταλήγουμε να έχουμε όλο τον φόρτο εργασίας σε μια διεργασία ενώ οι υπόλοιπες είναι σε $idle$ κατάσταση.

4.4 Υλοποίηση

Σε αυτή την ενότητα εξετάζονται προκλήσεις που εμφανίστηκαν κατά την υλοποίηση και τεκμηριώνεται η διεπαφή που παρέχει η βιβλιοθήκη σε όσους θέλουν να ενσωματώσουν ερωτήματα αντίστροφων κορυφαίων-k στις εφαρμογές τους. Υλοποιούνται οι αλγόριθμοι που περιγράφηκαν παραπάνω μαζί με τις διαφορετικές εκδοχές τους για όποιον αλγόριθμο χρησιμοποιεί στρατηγικές διαχωρισμού.

Συγκεκριμένα υλοποιείται ο *Naive* και ο *Layered* οι οποίοι δεν υποστηρίζουν διαφορετικές εκδοχές και ο *Reverse top-k Threshold Algorithm* με τον *Reverse top-k Threshold Algorithm με Grid* οι οποίοι εκτίθενται με τρεις διαφορετικές εκδοχές, μια για κάθε διαχωρισμό (*Angle*, *Uniform*, *Inverse*).

4.4.1 Περιορισμοί από την υλοποίηση της CPython

Ένα από τα μεγάλα προβλήματα του να γράφει κάποιος παράλληλο κώδικα σε python είναι ότι τα νήματα (threads) της cpython δεν είναι αποδοτικά για έντονα CPU προβλήματα. Ο λόγος για αυτό είναι ότι στην υλοποίηση της cpython υπάρχει το λεγόμενο GIL (Global Interpreter Lock). Το GIL εισήχθη αρχικά ως μέρος της προσπάθειας να υποστηριχθεί ο παραλληλισμός νημάτων στην python. Η python χρησιμοποιεί την αυτόματη διαχείριση μνήμης μέσω της συλλογής απορριμμάτων, η οποία υλοποιείται με μια τεχνική που ονομάζεται μέτρηση αναφοράς (reference count). Το reference count είναι μια δομή δεδομένων που περιέχει όλες τις αναφορές αντικειμένων που μπορούν να προσεγγιστούν από ένα πρόγραμμα και όταν ένα αντικείμενο έχει μηδενικές αναφορές, μπορεί να ελευθερωθεί.

Ωστόσο, οι συνθήκες αγώνα (race conditions) στον παραλληλισμό το έκαναν έτσι ώστε η καταμέτρηση αυτών των αναφορών να μπορεί να ανανεώνεται λανθασμένα, καθιστώντας έτσι τα αντικείμενα να μπορούν να απελευθερωθούν εσφαλμένα ή να μην απελευθερωθούν καθόλου. Ένας τρόπος για να λυθεί αυτό το πρόβλημα είναι με granular locks, για παράδειγμα γύρω από κάθε κοινό αντικείμενο, αλλά αυτό θα δημιουργούσε ζητήματα όπως η αυξημένη επιβάρυνση λόγω πολλών αιτημάτων απόκτησης / αποδέσμευσης κλειδώματος, καθώς και η αύξηση της πιθανότητας αδιεξόδου.

Οι προγραμματιστές της python επέλεξαν να λύσουν αυτό το πρόβλημα τοποθετώντας μια κλειδαριά γύρω από ολόκληρο το διερμηνέα, κάνοντας κάθε νήμα να αποκτήσει αυτή την κλειδαριά όταν τρέχει το bytecode της Python. Αυτό αποφεύγει πολλά από τα προβλήματα επιδόσεων γύρω από το υπερβολικό κλείδωμα, αλλά αποτελεσματικά σειριοποιεί την εκτέλεση bytecode. Με απλά λόγια, το GIL είναι μια κλειδαριά γύρω από τον διερμηνέα. Κάθε νήμα που επιθυμεί να εκτελέσει τον bytecode της python πρέπει να κρατήσει το GIL για να το κάνει. Αυτό σημαίνει ότι το πολύ ένα νήμα μπορεί να εκτελέσει python bytecode οποιαδήποτε στιγμή.[17]

Το αποτέλεσμα της ύπαρξης του GIL είναι να μην είναι δυνατό να χρησιμοποιήσουμε νήματα σε εντατικά επεξεργαστικά προβλήματα όπως το $RTOP_k(q)$ οπότε χρειάστηκε να καταφύγουμε στην χρήση διεργασιών. Το πρόβλημα που εισάγεται με τις διεργασίες είναι ότι κάθε διεργασία έχει το δικό της ανεξάρτητο memory space με αποτέλεσμα όταν κάνουμε spawn μια καινούργια διεργασία να αντιγράφουμε όλο το data-set από την κύρια διεργασία στις υπόλοιπες το οποίο καταλήγει σε μια επιπλέον επιβάρυνση της μνήμης τυχαίας προσπέλασης. Παρόλα αυτά το πρόβλημα το οποίο μελετάμε αφορά την επεξεργαστική αποδοτικότητα οπότε η επιπρόσθετη μνήμη δεν μας επηρεάζει στις μετρήσεις μας.

4.4.2 Χρήση του API

Ο στόχος κατά την συγγραφή της βιβλιοθήκης ήταν να παρέχει όμορφα και καθαρά bindings ώστε να είναι εύκολο κάποιος άλλος προγραμματιστής να μπορεί να την χρησιμοποιήσει εύκολα χωρίς να χρειάζεται να γνωρίζει τι συμβαίνει στο παρασκήνιο. Όπως μπορεί κάποιος να παρατηρήσει παρακάτω χρησιμοποιούμε 4 γραμμές κώδικα για να εκτελέσουμε το ερώτημα.

Αρχικά πρέπει να οριστεί το σύνολο δεδομένων, το σύνολο προτιμήσεων και το σημείο ως προς το οποίο θα διεξαχθεί το ερώτημα αντίστροφων κορυφαίων-κ. Χρησιμοποιούμε το NumPy για να κατασκευάσουμε τους πίνακες που αναπαριστούν το σύνολο δεδομένων και προτιμήσεων έτσι ώστε να μπορούμε να επωφεληθούμε από την βιβλιοθήκη για να έχουμε ταχύτερες πράξεις.

```
import NumPy as np
# set of data defined using a simple
# NumPy array of two dimensional arrays
dataset = np.array([
    [9482.012,  1883.1473],
    [425.9226,  3913.8147],
    ...
])
# set of weights
weights = np.array([
    [0.4211, 0.5789],
    [0.4839, 0.5161],
    ...
])
# query point
q_point = np.array([150, 200])
k = 10
# number of processes available
processes = 4
# number of grids if they are required
```

```
21 grids = 10
```

Κάνουμε import την κύρια κλάση που χρησιμοποιείται για την εκτέλεση ερωτημάτων αντίστροφων κορυφαίων-κ.

```
1 from rtopk import ReverseTopK
```

Πέρα από την κύρια κλάση είναι απαραίτητο να επιλέξουμε έναν αλγόριθμο από τους αλγόριθμους που ορίσαμε στις αρχές αυτού του κεφάλαιου.

Οι διαθέσιμοι αλγόριθμοι είναι οι εξής:

NaiveRTopK
: Αφελής / Εξαντλητικός

LayeredRTopK
: Ο layered αλγόριθμος

ReverseTopKThresholdAngleBased
: RTA με διαχωρισμό γωνίας

ReverseTopKThresholdUniform
: RTA με διαχωρισμό ομοιόμορφο

ReverseTopKThresholdRandomUniform
: RTA με διαχωρισμό αντίστροφο

RTAGridAngleRTopK
: RTA με grid και διαχωρισμό γωνίας

RTAGridUniformRTopK
: RTA με grid και διαχωρισμό ομοιόμορφο

RTAGridRandomRTopK
: RTA με grid και διαχωρισμό αντίστροφο

```
1 from rtopk import ReverseTopKThresholdRandomUniform
```

Στην συνέχεια κάνουμε initialize την κλάση και τρέχουμε το ερώτημα μας.

```
1 rtopk = ReverseTopK(
2     algorithm=ReverseTopKThresholdRandomUniform,
3     weights=weights,
4     dataset=dataset,
5     processes=processes,
6 )
7 results = rtopk.run_query(q_point, k)
8 print(results)
```

Μόλις τελειώσει μας επιστρέφει μια απάντηση σε μορφή json:

```
1 {
2     "result": array([
3         [3.317e-01, 6.683e-01],
4         [3.312e-01, 6.688e-01],
5         [3.306e-01, 6.694e-01],
6         ...,
7         [1.510e-02, 9.849e-01],
```

```
        [6.900e-03, 9.931e-01]
    ]),
    "time": 1.9327211380004883,
    "weight_skipped": 181,
    "average_weights_skipped": 45.25,
    "deviation_weights_skipped": 5.0682837331783235,
    "calculations": 819,
    "average_calculations": 204.75,
    "deviation_calculations": 5.0682837331783235,
    "cpu_activity": {
        0: 1.7876198291778564,
        1: 1.8035612106323242,
        2: 1.713073492050171,
        3: 1.7967305183410645
    },
    "weight_count": 251
}
```

Όπου το κλειδί με όνομα result περιέχει τις προτιμήσεις που ανήκουν στο αποτέλεσμα, ενώ τα υπόλοιπα κλειδιά περιέχουν στατιστικές πληροφορίες σχετικά με την εκτέλεση του αλγόριθμου.

Κεφάλαιο 5

Πειραματική αξιολόγηση

Ο στόχος σε αυτή την ενότητα ήταν να πειραματιστούμε χρησιμοποιώντας μεγάλο πλήθος δεδομένων, προτιμήσεων και μεγάλο k, να δούμε αν θα υπάρχει αποδοτική επιτάχυνση όταν χρησιμοποιούμε πολλές διεργασίες αντί για μόνο μια διεργασία. Έγινε δηλαδή μια προσπάθεια να δούμε πως αντιδρούσαν οι αλγόριθμοι όταν παραλληλίζονται υπό συνθήκες προβλημάτων που χρειάζοντουσαν πολύ υπολογιστική δύναμη. Συγκεκριμένα ο στόχος είναι να βρεθεί ποιος είναι ο πιο αποδοτικός τρόπος να διαχωρίσουμε τις προτιμήσεις ανά υπό-διεργασία και ποιος αλγόριθμος έχει το μέγιστο κέρδος.

Τα εργαστηριακά πειράματα εκτελέστηκαν χρησιμοποιώντας ένα ηλεκτρονικό υπολογιστή με Intel Core i7-6700 CPU (4 cores, 8 threads, 3.40 GHz) και 7.7 GB RAM σε περιβάλλον Ubuntu 18.04.

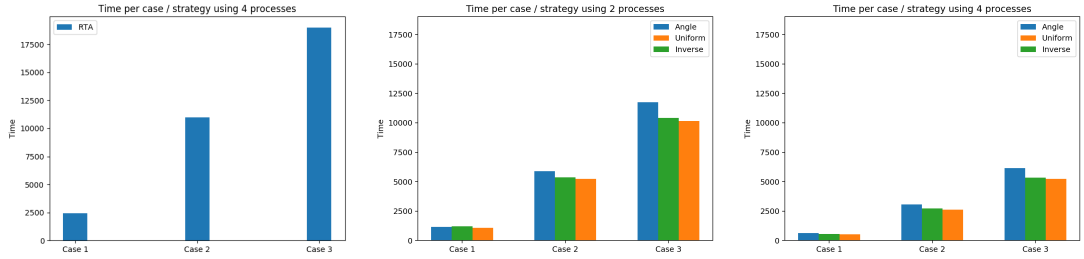
5.1 Κλιμάκωση αλγορίθμων ανά διεργασίες

Βλέποντας τα παρακάτω διαγράμματα είναι ξεκάθαρο ότι ο αλγόριθμος RTA κλιμακώνεται (scale) με βάση το πλήθος των processes για τα ακόλουθα cases όπως παρουσιάζονται στον παρακάτω πίνακα. Παρατηρούμε ότι επιστρέφει το κάθε case ένα μεγάλο ποσοστό των προτιμήσεων που εισήχθησαν με στόχο ο αλγόριθμος να είναι όσο πιο CPU intensive γίνεται, καθώς όταν μια προτίμηση απορρίπτεται, παύουμε να ψάχνουμε σημεία τα οποία θα αποτελούν μέρος των κορυφαίων-k της προτίμησης.

Οπότε βλέπουμε ότι το όλο εγχείρημα είναι εφικτό και έχει αποδοτικά αποτελέσματα. Ακολουθεί ο πίνακας με τον καλύτερο χρόνο ανά περίπτωση δηλαδή τον χρόνο τον οποίο χρειάστηκε η καλύτερη στρατηγική. Στο συγκεκριμένο παράδειγμα φαίνεται ότι η καλύτερη στρατηγική είναι ο Ομοιόμορφος διαχωρισμός(βλέπε κεφάλαιο 4.3.1).

Περίπτωση	Data-Set	Weights	k	Πλήθος Αποτελέσματος
1	100000	50000	5000	50000
2	500000	50000	10000	47774
3	1000000	50000	15000	46972

Πίνακας 5.1: Πειράματα με σημείο Q [400, 400]



Σχήμα 5.1: Scaling per process count

Διεργασίες	Περίπτωση 1 (seconds)	Περίπτωση 2 (seconds)	Περίπτωση 3 (seconds)
1	2448	10993	19015
2	1070	5231	10145
4	535	2635	5231

Υπολογίζοντας το speedup με τον τύπο που ορίστηκε στο κεφάλαιο 2.1.2 ανά περίπτωση βλέπουμε ότι το speedup για 2 διεργασίες είναι:

Περίπτωση 1: S(2) = 2448/1070 = 2.29 S(4) = 2448/535 = 4.58
Περίπτωση 2: S(2) = 10993/5231 = 2.10 S(4) = 10993/2635 = 4.17
Περίπτωση 3: S(2) = 19015/10145 = 1.87 S(4) = 19015/5231 = 3.64

Παρατηρούμε ότι σε αρκετά από τα αποτελέσματα υπάρχει το λεγόμενο *super-linear speedup*, δηλαδή speedup το οποίο είναι πάνω από την ευθεία $y = x$. Αυτό μπορεί να οφείλεται για παράδειγμα στην κρυφή μνήμη καθώς κάθε πυρήνας μπορεί να έχει, πέρα από την L2 κρυφή μνήμη που μοιράζονται όλοι οι πυρήνες, την δική του ιδιωτική L1 κρυφή μνήμη. Με το μεγαλύτερο συσσωρευμένο μέγεθος κρυφής μνήμης, ο χρόνος πρόσβασης στη μνήμη μειώνεται δραματικά, το οποίο ως γεγονός, προκαλεί επιπλέον επιτάχυνση πέρα από τον πραγματικό υπολογισμό.

5.2 Κλιμάκωση σύνολου δεδομένων

Ο πρώτος πειραματισμός αφορά την αποδοτικότητα των αλγορίθμων μέσα από την κλιμάκωση του πλήθους του σύνολου δεδομένων. Εξετάζονται δύο περιπτώσεις;

- Ομοιόμορφο σύνολο δεδομένων
- Αντί-συσχετιζόμενο σύνολο δεδομένων

Είναι σημαντικό να αναφέρουμε ότι οι προτιμήσεις που χρησιμοποιήθηκαν καλύπτουν ομοιόμορφα το ευθύγραμμο τμήμα $x + y = 1$ με άκρα τα $(x, y) = 1, 0$ και $(x, y) = 0, 1$.

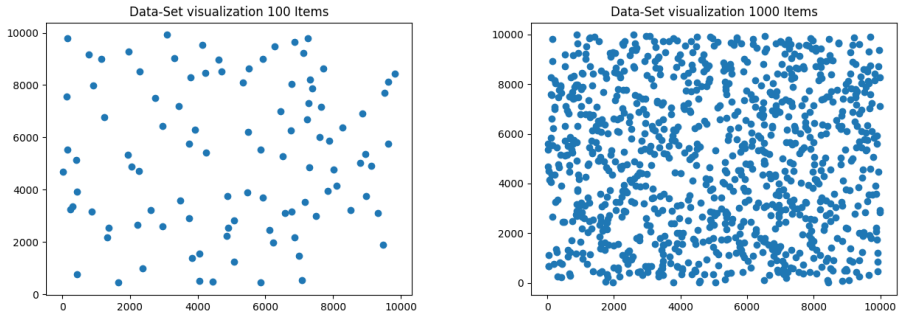
5.2.1 Ομοιόμορφα δεδομένα

Σε αυτό το σημείο ασχολούμαστε με το πως αντιδρούν όλοι οι αλγόριθμοι σε ένα περιβάλλον όπου τα δεδομένα είναι ομοιόμορφα κατανεμημένα στον χώρο. Είναι σημαντικό ότι λόγο της κατανομής και του τρόπου με τον οποίο φιλτράρουμε τον χώρο, πολλά σημεία που προστίθενται ομοιόμορφα αποτελούν μέλος του *anti-dominance region* και του *dominance region* όπως ορίστηκαν στο κεφάλαιο

Περίπτωση	Data-Set	Weights	k	Πλήθος Αποτελέσματος
1	100000	50000	1500	45625
2	500000	50000	6000	44300
3	1000000	50000	10000	42874

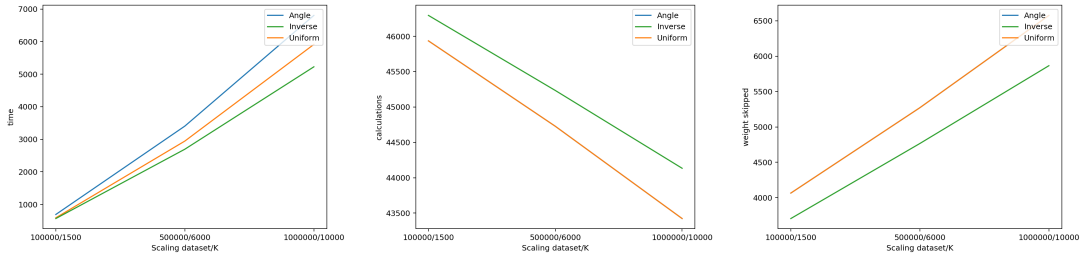
Πίνακας 5.2: Πειράματα με σημείο Q [300, 600]

4.2.3. Για αυτό τον λόγο όταν αυξάνεται το σύνολο δεδομένων μαζί με αυτό αυξάνουμε και το μέγεθος k. Επειδή πολλά από τα καινούργια σημεία μπαίνουν στο *anti-dominance region*, άμα το k παραμείνει σταθερό, κάποια στιγμή το πλήθος των σημείων που θα είναι στο anti-dominance region θα ξεπεράσει το k. Οπότε το μοντέλο επίλυσης των ερωτημάτων θα γνωρίζει ότι το $RTop_K(Q, S, W)$ επιστρέφει $\emptyset$ σύνολο χωρίς να εκτελέσει ούτε ένα ερώτημα κορυφαίων-k.



Σχήμα 5.2: Ομοιόμορφη κατανομή του πληθυσμού

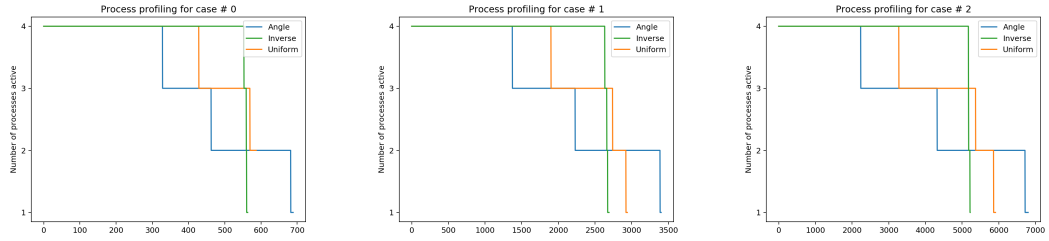
Επιπλέον σε αυτό το πείραμα άλλαξε το σημείο q και τα k έτσι ώστε το $RTop_K(q, S, W)$ να επιστρέφει μικρότερο σύνολο σε σχέση με το σύνολο που επέστρεφε το πείραμα στο κεφάλαιο 5.1.



Σχήμα 5.3: Πειραματισμός πάνω στον RTA με ομοιόμορφα δεδομένα

Το ενδιαφέρον του πειράματος είναι ότι αυτή την φορά η στρατηγική, που εκτελείται πιο γρήγορα και υπερτερεί είναι ο *αντίστροφος διαχωρισμός* (βλέπε κεφάλαιο 4.3.3). Ιδιαίτερα ενδιαφέρον είναι συγκεκριμένα το γεγονός ότι ο *αντίστροφος διαχωρισμός* κάνει παραπάνω calculations από τον *ομοιόμορφο* (δηλαδή εκτελεί πιο πολλά ερωτήματα κορυφαίων-k) και παραλείπει λιγότερες προτιμήσεις (δηλαδή η επόμενη προτίμηση ξεπερνάει το threshold).

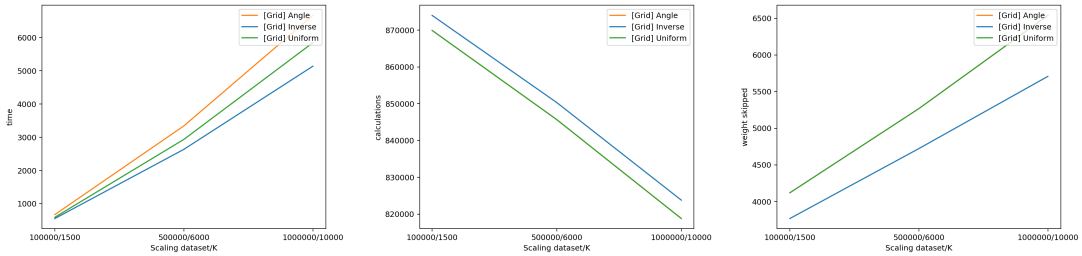
Η απάντηση σε αυτό το παράδοξο βρίσκεται μέσα από CPU profiling που εμφανίζεται στα παρακάτω γραφήματα. Μέσα από ένα CPU profiling βλέπουμε πόσους λογικούς πυρήνες χρησιμοποιεί ο κάθε αλγόριθμος καθόλη την διάρκεια της εκτέλεσης του.



Σχήμα 5.4: CPU Profiling ανά αλγόριθμο / περίπτωση

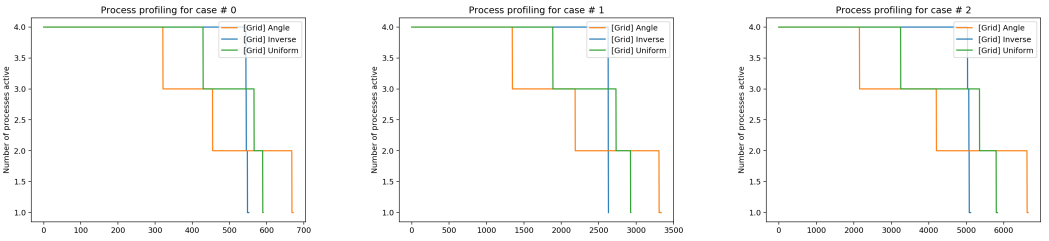
Βλέπουμε ότι σε κάθε περίπτωση ο *αντίστροφος διαχωρισμός* χρησιμοποιεί και τις 4 διεργασίες ενώ στις άλλες δυο στρατηγικές μετά από λίγη ώρα κάποιες από τις διεργασίες γίνονται άεργες. Το συμπέρασμα είναι ότι στις άλλες στρατηγικές δεν υπάρχει ισορροπημένος φόρτος εργασίας σε σχέση με τον *αντίστροφο διαχωρισμό*.

Στην συνέχεια θα κάνουμε τον ίδιο πειραματισμό και για τον αλγόριθμο RTA με Grid το οποίο είναι 10 επί 10 για να δούμε πως επιδράει το Grid ως δομή δεδομένων πάνω στις στρατηγικές διαχωρισμού.



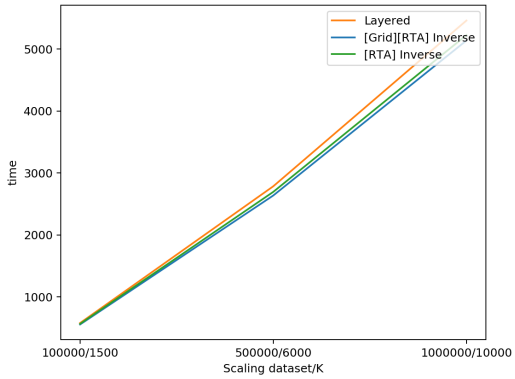
Σχήμα 5.5: Πειραματισμός πάνω στον RTA με Grid με ομοιόμορφα δεδομένα

Από ότι φαίνεται η εφαρμογή μιας δομής δεδομένων δεν αλλάζει την βέλτιστη στρατηγική διαχωρισμού. Παρατηρούμε ότι είναι πολύ υψηλός ο αριθμός από πράξεις στους RTA με Grid αλλά αυτό οφείλεται στο ότι μια πράξη κορυφαίων-κ εφαρμόζεται σε όλο το σύνολο δεδομένων ενώ στο RTA με Grid εφαρμόζεται στο 1/100 του συνόλου δεδομένων. Αυτό βέβαια δεν σημαίνει ότι η πράξη που κάνει ο RTA εφαρμόζεται απαραίτητα σε όλο το σύνολο καθώς ο αλγόριθμος απλά εξετάζει μέχρι να βρει κ σημεία καλύτερα από το σημείο του ερωτήματος όπως ορίστηκε στο κεφάλαιο 4.2.2. Ακολουθούν τα διαγράμματα χρήσης διεργασιών.



Σχήμα 5.6: CPU Profiling ανά αλγόριθμο / περίπτωση στον RTA με Grid

Τέλος θα συγκρίνουμε τους αλγόριθμους RTA και RTA με Grid που χρησιμοποιούν αντίστροφο διαχωρισμό με τον Layered αλγόριθμο.

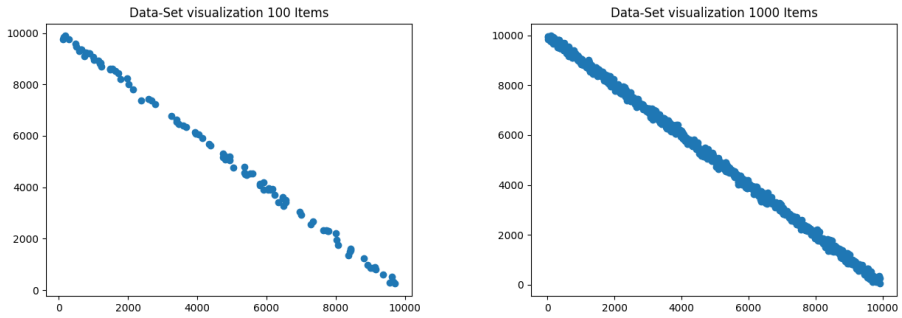


Σχήμα 5.7: Καλύτερος χρόνος εκτέλεσης

Συμπεραίνουμε ότι όλοι οι αλγόριθμοι έχουν παρόμοιο χρόνο εκτέλεσης με τον RTA που χρησιμοποιεί Grid και αντίστροφο διαχωρισμό να παίρνει τον προβάδισμα και να αποτελεί τον βέλτιστο αλγόριθμο σε σχέση με τους υπόλοιπους.

5.2.2 Αντί-συσχετισμένα δεδομένα

Σε αυτό το σημείο ασχολούμαστε με το πως αντιδρούν οι αλγόριθμοι σε ένα περιβάλλον όπου τα δεδομένα είναι αντί-συσχετισμένα (anti-correlated) κατανεμημένα στον χώρο. Τέτοιου είδους δεδομένα και από άλλες έρευνες στο παρελθόν αποτελούν συχνά την πιο δύσκολη περίπτωση για τους αλγόριθμους καθώς δυσκολεύουν το φιλτράρισμα δεδομένων.



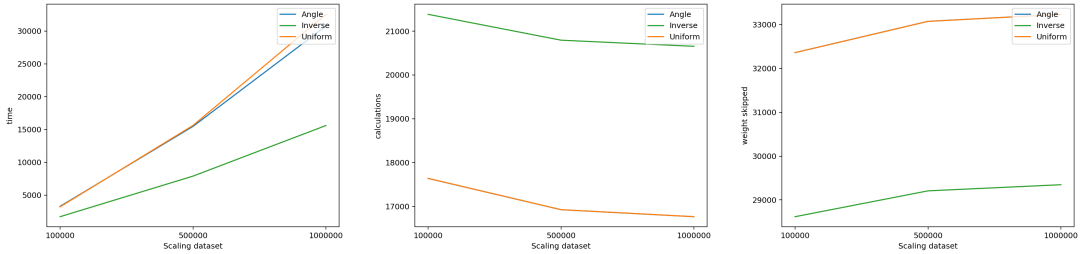
Σχήμα 5.8: Αντι-συσχετισμένη κατανομή του πληθυσμού

Συγκεκριμένα όπως βλέπουμε και από τα δεδομένα του παραδείγματος, όλα τα σημεία βρίσκονται κοντά στην ευθεία $x + y = 10000$. Το αποτέλεσμα είναι δεδομένου ενός αντικειμένου Q, ότι καθώς προσθέτουμε δεδομένα στον χώρο το πλήθος αυτών που θα ανήκουν στα anti-dominance region και dominance region να είναι πολύ μικρότερο σε σχέση με μια ομοιόμορφη κατανομή.

Περίπτωση	Data-Set	Weights	k	Πλήθος Αποτελέσματος
1	100000	50000	2000	15961
2	500000	50000	2000	15183
3	1000000	50000	2000	15009

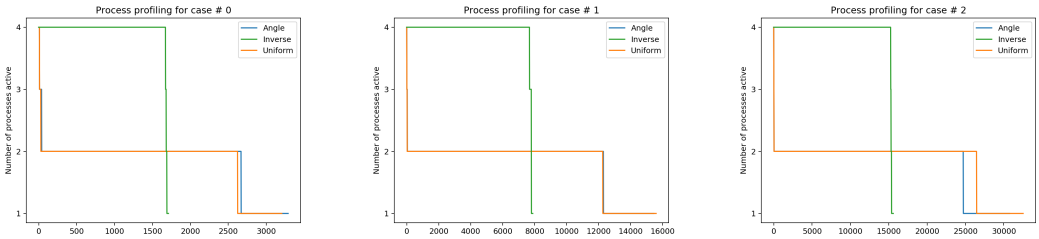
Πίνακας 5.3: Πειράματα με σημείο Q [4000, 3000]

Λόγω της εντατικής φύσης του αλγόριθμου με τέτοιου είδους δεδομένα επιλέχθηκε q και k έτσι ώστε πολλές από τις προτιμήσεις να απορρίπτονται με στόχο την ταχύτερη εκτέλεση των πειραμάτων.



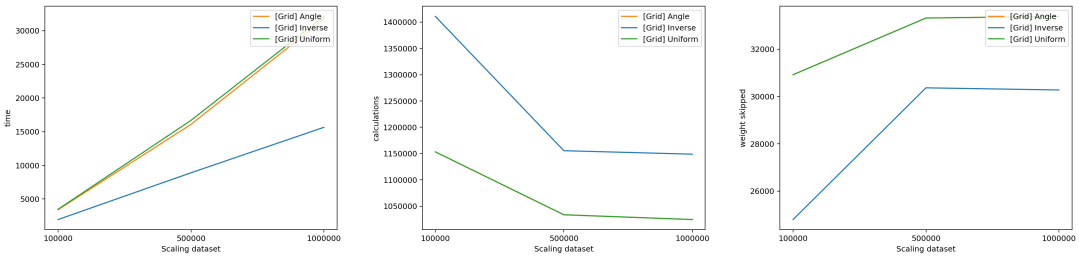
Σχήμα 5.9: Πειραματισμός πάνω στον RTA με αντί-συσχετισμένα δεδομένα

Βλέποντας όμοια αποτελέσματα με αυτά που παρήγαγε ο αλγόριθμος για τα ομοιόμορφα δεδομένα αναμένουμε και ένα παρόμοιο διάγραμμα διεργασιών όπου ο αντίστροφος διαχωρισμός χρησιμοποιεί και τις 4 διεργασίες που του δίνονται ενώ οι υπόλοιποι έχουν τουλάχιστον μια διεργασία σε κατάσταση αναμονής την πιο πολύ ώρα.

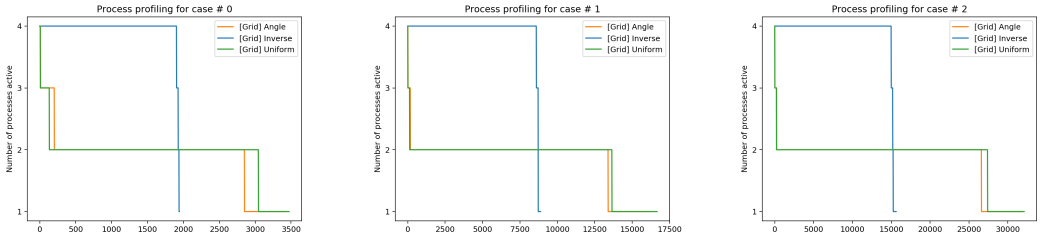


Σχήμα 5.10: CPU Profiling ανά αλγόριθμο / περίπτωση

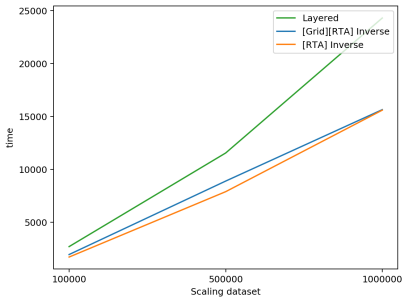
Στην συνέχεια θα κάνουμε τον ίδιο πειραματισμό και για τον αλγόριθμο RTA με Grid το οποίο είναι 10 επί 10 όπως κάναμε και με τα ομοιόμορφα δεδομένα για να δούμε πως επιδράει το Grid ως δομή δεδομένων πάνω στις στρατηγικές διαχωρισμού σε αντί-συσχετιζόμενα δεδομένα.



Σχήμα 5.11: Πειραματισμός πάνω στον RTA με Grid με αντί-συσχετισμένα δεδομένα



Σχήμα 5.12: CPU Profiling ανά αλγόριθμο / περίπτωση στον RTA με Grid



Σχήμα 5.13: Καλύτερος χρόνος εκτέλεσης

Τέλος θα συγκρίνουμε τους καλύτερους αλγόριθμους από RTA και RTA με Grid με τον layered αλγόριθμο, για να δούμε ποιος αλγόριθμος θα είναι αυτός με την καλύτερη επίδοση.
Σε αντίθεση με τα ομοιόμορφα δεδομένα παρατηρούμε ότι στα αντί-συσχετιζόμενα δεδομένα το Grid στον RTA επιβαρύνει την εκτέλεση του ερωτήματος, με αποτέλεσμα ο RTA χωρίς grid να αποκτά το προβάδισμα.

5.3 Κλιμάκωση Προτιμήσεων

Ο δεύτερος πειραματισμός αφορά την αποδοτικότητα των αλγορίθμων μέσα από την κλιμάκωση του πλήθους των προτιμήσεων. Όπως και στον προηγούμενο πειραματισμό εξετάζονται δύο περιπτώσεις πάνω στο σύνολο δεδομένων:

- Ομοιόμορφο σύνολο δεδομένων
- Αντισυσχετιζόμενο σύνολο δεδομένων

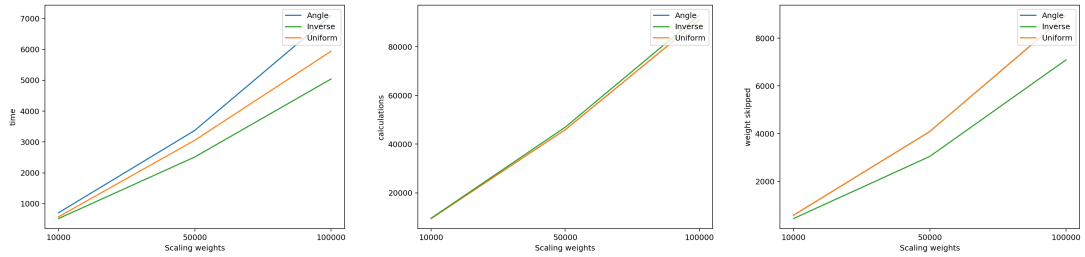
5.3.1 Ομοιόμορφα δεδομένα

Τα πειράματα που εκτελέστηκαν σε αυτή την ενότητα χρησιμοποιούν τις τιμές του παρακάτω πίνακα και έχουν ως στόχο να δούμε πως επηρεάζονται οι αλγόριθμοι από την αύξηση του πλήθους των προτιμήσεων σε σχέση με την αύξηση του σύνολου δεδομένων.

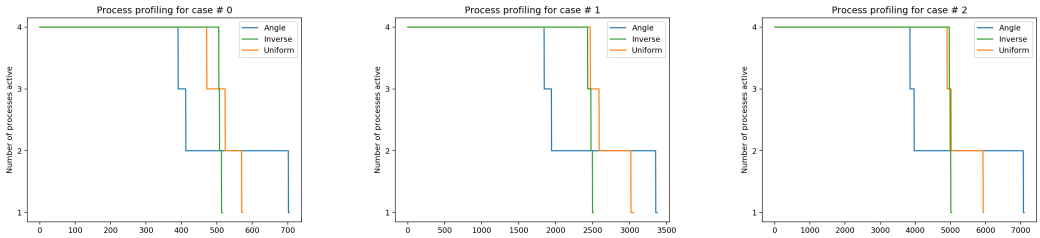
Case	Data-Set	Weights	k	Πλήθος Αποτελέσματος
1	500000	10000	5000	9015
2	500000	50000	5000	45057
3	500000	100000	5000	90133

Πίνακας 5.4: Πειράματα με σημείο Q [400, 400]

Ξεκινάμε και πάλι με την σύγκριση των παραλλαγών του RTA ως προς τον διαχωρισμό των προτιμήσεων τους.

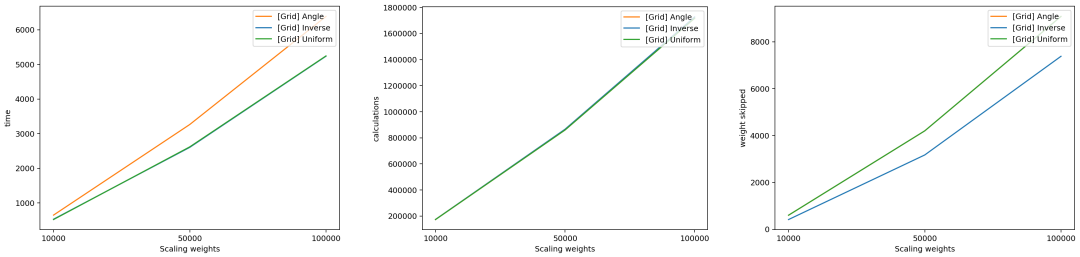


Σχήμα 5.14: Πειραματισμός πάνω στον RTA με ομοιόμορφα δεδομένα και προτιμήσεις που κλιμακώνονται

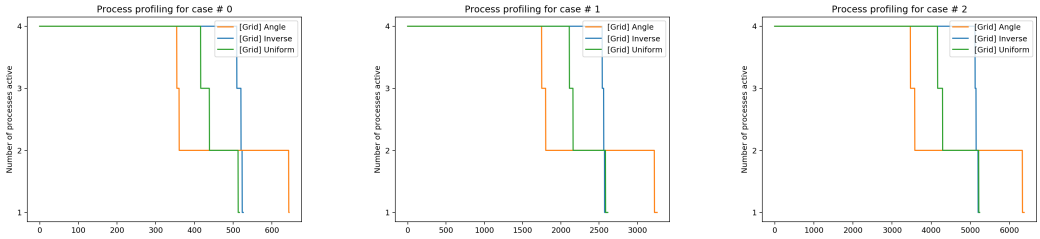


Σχήμα 5.15: CPU Profiling ανά αλγόριθμο / περίπτωση

Όμοια με τα πειράματα ως προς το σύνολο δεδομένων η στρατηγική του αντίστροφου διαχωρισμού παραμένει η κυρίαρχη στρατηγική διαχωρισμού των προτιμήσεων. Προχωράμε στην εξέταση των στρατηγικών του RTA με Grid.



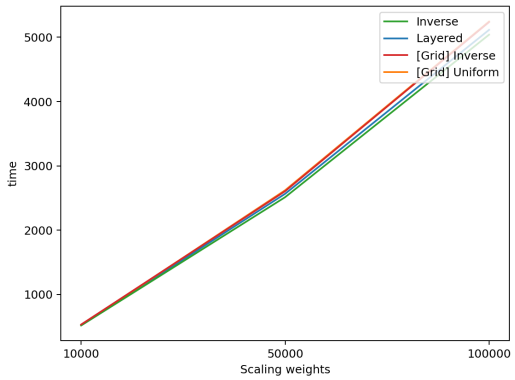
Σχήμα 5.16: Πειραματισμός πάνω στον RTA με Grid με ομοιόμορφα δεδομένα και προτιμήσεις που κλιμακώνονται



Σχήμα 5.17: CPU Profiling ανά αλγόριθμο / περίπτωση στον RTA με Grid

Το ενδιαφέρον είναι ότι αυτή την φορά δεν έχουμε ξεκάθαρη νικητήρια στρατηγική. Και η αντίστροφη και η ομοιόμορφη στρατηγική έχουν παρόμοια αποτελέσματα/αποδοτικότητα. Στον παρακάτω πίνακα φαίνεται πόσο κοντά είναι οι χρόνοι των αλγορίθμων καθώς είναι πολύ πιο δύσκολο να φανεί αυτό μέσα από τα διαγράμματα.

Τέλος συγκρίνουμε όλους τους ανταγωνιστικούς αλγόριθμους μεταξύ τους.



Σχήμα 5.18: Καλύτερος χρόνος εκτέλεσης

Επειδή όμως στο διάγραμμα είναι πολύ δύσκολο να δούμε τους χρόνους που είχαν οι αλγόριθμοι και να τους συγκρίνουμε θα χρησιμοποιήσουμε έναν απλό πίνακα.

Στρατηγική	1η περίπτωση	2η περίπτωση	3η περίπτωση
RTA με Grid Αντίστροφη	532	2605	5236
RTA με Grid Ομοιόμορφη	522	2620	5244
RTA Αντίστροφη	517	2512	5040
Layered	532	2605	5236

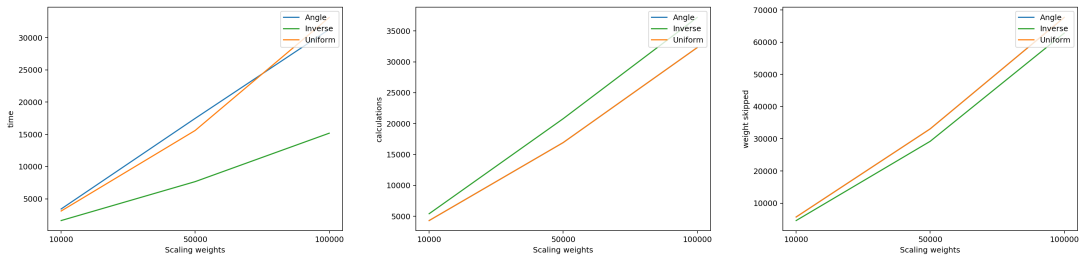
5.3.2 Αντί-συσχετισμένα δεδομένα

Όμοια με τα προηγούμενα πειράματα, τα πειράματα που εκτελέστηκαν σε αυτή την ενότητα χρησιμοποιούν τις τιμές του παρακάτω πίνακα και έχουν ως στόχο να δούμε πως επηρεάζονται οι αλγόριθμοι από την αύξηση του πλήθους των προτιμήσεων σε σχέση με την αύξηση του αντί-συσχετιζόμενου σύνολου δεδομένων.

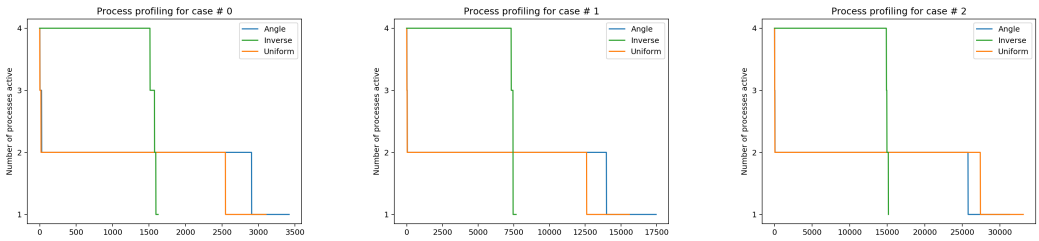
Περίπτωση	Data-Set	Weights	k	Πλήθος Αποτελέσματος
1	500000	10000	2000	2997
2	500000	50000	2000	15183
3	500000	100000	2000	30558

Πίνακας 5.5: Πειράματα με σημείο Q [4000, 3000]

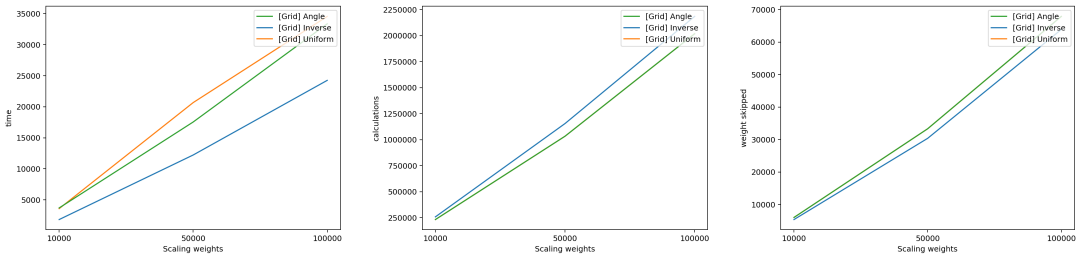
Αρχικά συγκρίνουμε τους RTA και RTA με Grid με τις παραλλαγές τους ως προς τις στρατηγικές διαχωρισμού.



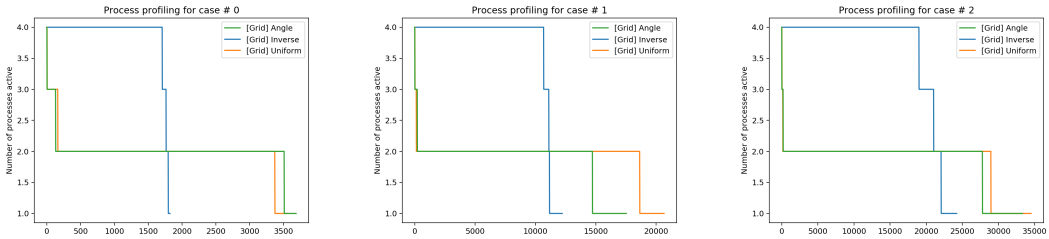
Σχήμα 5.19: Πειραματισμός πάνω στον RTA με αντί-συσχετισμένα δεδομένα και προτιμήσεις που κλιμακώνονται



Σχήμα 5.20: CPU Profiling ανά αλγόριθμο / περίπτωση

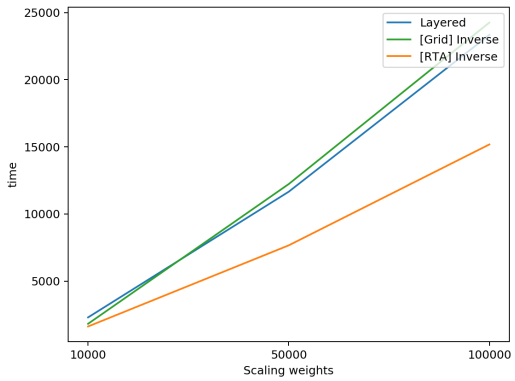


Σχήμα 5.21: Πειραματισμός πάνω στον RTA με Grid με αντί-συσχετισμένα δεδομένα και προτιμήσεις που κλιμακώνονται



Σχήμα 5.22: CPU Profiling ανά αλγόριθμο / περίπτωση στον RTA με Grid

Παρατηρούμε για άλλη μια φορά ότι η αντίστροφη στρατηγική υπερισχύει των υπολοίπων και για τον RTA και για τον RTA με Grid. Το ιδιαίτερο είναι ότι σε σχέση με το προηγούμενο πείραμα, οι χρονικές επιδόσεις των αλγόριθμων / στρατηγικών έχουν πάλι ένα χάσμα μεταξύ τους όπως στα αρχικά πειράματα όπου αυξανόταν το πλήθος του συνόλου δεδομένων και όχι οι προτιμήσεις. Προχωράμε στο τελευταίο διάγραμμα που συγκρίνει την καλύτερη έκδοση του κάθε αλγόριθμου με τις άλλες



Σχήμα 5.23: Καλύτερος χρόνος εκτέλεσης

Φαίνεται ότι πάλι για αντί-συσχετιζόμενα δεδομένα το grid επιβαρύνει τον RTA, με αποτέλεσμα ο RTA με αντίστροφο διαχωρισμό να είναι ο κυρίαρχος αλγόριθμος.

Κεφάλαιο 6

Συμπεράσματα και Μελλοντική έρευνα

6.1 Συμπεράσματα

Μέσα από την εκπόνηση της παρούσας ερευνητικής εργασίας και κυρίως κατά τη διαδικασία αξιολόγησης των αλγορίθμων και στρατηγικών διαχείρισης προτιμήσεων για την αποδοτική επίλυση αντίστροφων ερωτημάτων k κορυφαίων σημείων, προέκυψαν οι εξής παρατηρήσεις:

Γενικά σε όλα τα πειράματα η στρατηγική η οποία κυριάρχησε έναντι των υπολοίπων ήταν η αντίστροφη. Η στρατηγική γωνίας δεν μπόρεσε σε κανένα από τα πειράματα να ανταγωνιστεί καμία από τις άλλες στρατηγικής, καθώς την κυριαρχούσε συστηματικά η ομοιόμορφη στρατηγική, η οποία μάλιστα σε κάποια πειράματα ξεπερνούσε σε μικρό βαθμό όμως την αντίστροφη. Το κοινό αυτών των πειραμάτων ήταν ότι το set που επέστρεφε το $RTOP_k(q)$ περιείχε το 90% των προτιμήσεων που εισήχθηκαν.

Από τους αλγόριθμους ο σταθερά πιο αποδοτικός ήταν ο RTA, με τον RTA με grid να ακολουθεί και στο τέλος να είναι ο Layered αλγόριθμος. Το grid στον RTA δεν ήταν τόσο αποδοτικό στα δεδομένα μάλλον για λόγους παραμετροποίησης, δηλαδή η επιλογή του πλήθους των grids στον χώρο ήταν πιθανότατα άστοχη.

6.2 Μελλοντική έρευνα

Η βιβλιοθήκη που παρουσιάστηκε στην παρούσα εργασία, αποτελεί ένα πολύ αποδοτικό εργαλείο επίλυσης αντίστροφων ερωτημάτων k κορυφαίων σημείων. Ωστόσο ορισμένα ενδιαφέροντα ζητήματα παραμένουν ανοιχτά προς μελλοντική έρευνα:

Πέρα από την υλοποίηση αυτής της πτυχιακής, θα ήταν ιδιαίτερα ωφέλιμο να υπάρξει για την κοινότητα μια βιβλιοθήκη γραμμένη σε κάποια πιο χαμηλού επιπέδου γλώσσα όπως η C, η οποία θα παρέχει bindings διαθέσιμα σε άλλες γλώσσες. Έτσι θα υπάρχει μια δημόσια «state of the art» βιβλιοθήκη η οποία θα αποτελεί βάση για μελλοντικές έρευνες πάνω στα ερωτήματα k κορυφαίων σημείων.

Ιδιαίτερα δημοφιλής αυτήν την περίοδο στον επιστημονικό χώρο είναι η μηχανική μάθηση. Στην μηχανική μάθηση η πιο δύσκολη φάση είναι συνήθως η εκπαίδευση (training phase), καθώς είναι δύσκολο να κατασκευαστούν τα απαραίτητα σύνολα δεδομένων και ανάλογα με τον αλγόριθμο που χρησιμοποιείται μπορεί η συγκεκριμένη φάση να χρειαστεί πολλές ώρες εκπαίδευσης. Παρόλα αυτά

έχει το θετικό πως εφόσον παραχθεί το εκπαιδευμένο μοντέλο, μπορούν να πραγματοποιηθούν πολύ γρήγορα προβλέψεις με μεγάλο ποσοστό επιτυχίας. Θα μπορούσε να επιλεχθεί κάποιος αλγόριθμος τύπου gradient descent, ώστε να χρησιμοποιηθεί για να παράγει ένα μοντέλο το οποίο θα απαντάει σε τι ποσοστό από το πλήθος των προτιμήσεων θα επιστρέψει ένα $RTOP_k(q)$ δεδομένου του σημείου Q και του K.

Το μοντέλο που θα παρήγαγε μια τέτοια εργασία θα μπορούσε να χρησιμοποιηθεί επίσης ως ένα ευριστικό εργαλείο το οποίο θα μπορούσε να ευνοήσει και άλλες έρευνες ή συγκεκριμένα αυτήν την εργασία. Πριν υπολογίσουμε το $RTOP_k(q)$ θα είχαμε την δυνατότητα να ξέρουμε τι ποσοστό προτιμήσεων θα επιστρέψει το ερώτημα και εφόσον επιστρέψει παραπάνω από ένα κατώφλι, για παράδειγμα 90%, να επιλέγει ομοιόμορφο διαχωρισμό αλλιώς να επιλέγει αντίστροφο διαχωρισμό.

Παράρτημα Α

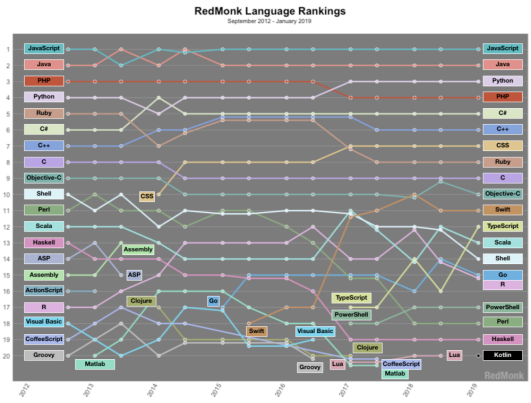
Εργαλεία που χρησιμοποιήθηκαν

Στο παράρτημα αναγράφονται όλα τα εργαλεία και οι βιβλιοθήκες που χρησιμοποιήθηκαν ώστε να έρθει εις πέρας αυτή η εργασία. Μαζί με κάθε αναφορά γίνεται μια γενική περιγραφή όπου εξηγείται η επιλογή του εργαλείου.

Α.1 Python

Η γλώσσα προγραμματισμού που επιλέχθηκε είναι η python και συγκεκριμένα η έκδοση 3.7, για λόγους ταχύτητας υλοποίησης και εξοικείωσης. Αποτελεί μια εύκολη στην εκμάθηση, ισχυρή γλώσσα προγραμματισμού. Διαθέτει ισχυρές δομές δεδομένων υψηλού επιπέδου και μια απλή αλλά αποτελεσματική προσέγγιση στον αντικειμενοστραφή προγραμματισμό. Η κομψή σύνταξη της, την καθιστούν ιδανική γλώσσα για scripting και γρήγορη ανάπτυξη εφαρμογών σε πολλές περιοχές στις περισσότερες πλατφόρμες.

Υπάρχουν πολλές ταξινομήσεις δημοτικότητας γλώσσας προγραμματισμού. Ενώ είναι δυνατό να επικρίνουμε ότι αυτοί οι οδηγοί δεν είναι ακριβείς, κάθε κατάταξη δείχνει την python ως κορυφαία γλώσσα προγραμματισμού μέσα στην πρώτη δεκάδα, αν όχι την πρώτη πεντάδα όλων των γλωσσών. Το IEEE ταξινόμησε την python ως τη # 1 γλώσσα προγραμματισμού το 2018, αφού ταξινομήθηκε ως η # 1 γλώσσα το 2017 και η κορυφαία γλώσσα προγραμματισμού # 3 το 2016[8]. Η κατάταξη του RedMonk τον Ιανουάριο του 2019 είχε την python στην 3η θέση[7].



Language Rank	Types	Spectrum Ranking
1. Python		100.0
2. C++		99.7
3. Java		97.5
4. C		96.7
5. C#		89.4
6. PHP		84.9
7. R		82.9
8. JavaScript		82.6
9. Go		76.4
10. Assembly		74.1

Σχήμα Α.1: Language popularity

Είναι ξεκάθαρο ότι η γλώσσα έχει υψηλή δημοτικότητα και ταυτόχρονα παρέχει ευελιξία, και

πολλές built-in βιβλιοθήκες out of the box. Έτσι η python δίνει τη δυνατότητα να αυτοματοποιήσουμε τα τετριμμένα πράγματα και να εστιάσουμε σε πιο συναρπαστικά και χρήσιμα πράγματα, όπως ο πειραματισμός.[1] Πέρα από αυτό όμως υπάρχουν πολλές third-party βιβλιοθήκες που έχουν φτιαχτεί από την κοινότητα οι οποίες είναι πολύ χρήσιμες για τον τομέα του Data Science όπως το NumPy, SciPy και το Matplotlib.

A.2 NumPy

Το NumPy είναι μια από τις θεμελιώδεις βιβλιοθήκες για την επιστημονική πληροφορική με την python. Η βιβλιοθήκη του προσφέρει ένα ισχυρό αντικείμενο που αναπαριστά έναν N-διαστάσεων πίνακα μέσα από τον οποίο μπορούν να εφαρμοστεί γραμμική άλγεβρα. Εκτός αυτού προσφέρει εκλεπτυσμένες λειτουργίες που χρησιμοποιούνται για τηλεπικοινωνίες όπως είναι ο μετασχηματισμός Fourier καθώς και γεννήτριες τυχαίων αριθμών. Τέλος προσφέρει εργαλεία για την ενσωμάτωση κώδικα από C, C++ και Fortran.

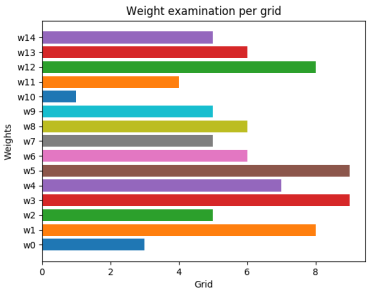
Εκτός από τις προφανείς επιστημονικές του χρήσεις, το NumPy μπορεί επίσης να χρησιμοποιηθεί ως αποτελεσματικό πολυδιάστατο δοχείο γενικών δεδομένων και να οριστούν αυθαίρετοι τύποι δεδομένων. Αυτό του επιτρέπει να ενσωματώνεται χωρίς προβλήματα και με μεγάλη ταχύτητα σε μια ποικιλία από βάσεις δεδομένων.[5]

Ο πρώτος λόγος για τον οποίο επιλέχθηκε το NumPy είναι πως έχει υλοποιηθεί σε C και εκτελείται πολύ γρήγορα ως αποτέλεσμα. Συγκριτικά, η python είναι μια δυναμική γλώσσα που ερμηνεύεται από τον διερμηνέα CPython, μετατρέπεται σε bytecode και εκτελείται. Εξαιτίας όλης αυτής της διαδικασίας το αποτέλεσμα είναι ο καταρτισμένος κώδικας C να είναι πάντα ταχύτερος.[2] Ο δεύτερος λόγος είναι, οι λειτουργίες που παρέχει η βιβλιοθήκη για πράξεις γραμμικής άλγεβρας πάνω σε πίνακες . Οι λειτουργίες αυτές χρησιμοποιούνται για το πολλαπλασιασμό και την μετάθεση των πινάκων ώστε να υπολογίζουμε γρήγορα τις βαθμολογίες των σημείων του σύνολου δεδομένων για κάθε μία από τις προτιμήσεις.

A.3 Matplotlib

Το matplotlib είναι μια βιβλιοθήκη σχεδίασης για python σε δύο διαστάσεις (2-dimensional), μέσα από την οποία μπορεί κάποιος να σχεδιάσει διαγράμματα ή διαδραστικά περιβάλλοντα για οποιαδήποτε πλατφόρμα. Χρησιμοποιείται συχνά για την αναπαράσταση δεδομένων, εξισώσεων, ιστογραμμάτων, φασμάτων ισχύος, διαγραμμάτων σφαλμάτων.

Όλα τα διαγράμματα που παρουσιάστηκαν στις προηγούμενες ενότητες, όπως το διάγραμμα στα δεξιά που αντικατοπτρίζει τις ενεργές προτιμήσεις ανά πλέγμα για τον Layered αλγόριθμο, κατασκευάστηκαν μέσα από λειτουργίες του matplotlib.



Είναι ιδιαίτερα διαδεδομένο σε επιστημονικές κοινότητες όπως για παράδειγμα στις κοινότητες που ασχολούνται με data science καθώς παρέχει την δυνατότητα της ενσωμάτωσης τους σε κελύφη ipython ή τετράδια jupyter. Το matplotlib έχει σχεδιαστεί για να είναι παρόμοια χρησιμοποιήσιμο με το MATLAB για λόγους συνήθειας των χρηστών, με το πλεονέκτημα του να είναι δωρεάν και ανοικτού κώδικα.

A.4 Decouple

Το Decouple είναι μια βιβλιοθήκη που βοηθά στην οργάνωση των ρυθμίσεων. Στόχος του είναι να υπάρχει ένα κοινό σημείο από το οποίο διαχειριζόμαστε τις ρυθμίσεις της εφαρμογής χωρίς να χρειάζεται να κάνουμε αλλαγές στον πηγαίο κώδικα αν θέλουμε να αλλάξουμε κάποια από αυτές. Συνήθως αυτό το σημείο είναι αρχεία τύπου dot env ή αλλιώς αρχεία περιβάλλοντος.

Ειδικότερα προσφέρει τις εξής δυνατότητες:

- αποθήκευση παραμέτρων σε αρχεία ini ή .env.
- καθορισμός πλήρων προεπιλεγμένων τιμών.
- μετατροπή των τιμών που λαμβάνουν οι παράμετροι στην κατάλληλη δομή δεδομένων.
- έχετε μόνο μία ενότητα διαμόρφωσης για να καθορίσετε όλες τις εμφανίσεις σας.

Βιβλιογραφία

[1] 10 reasons to learn python in 2019. https://hackernoon.com/10-reasons-to-learn-python-in-2018-f473dc35e2ee.

[2] 5 reasons you should know numpy. https://insights.dice.com/2016/09/01/5-reasons-know-numpy/.

[3] Amdahl's law. https://en.wikipedia.org/wiki/Amdahl%27s_law.

[4] Gustafson's law. https://en.wikipedia.org/wiki/Gustafson%27s_law.

[5] Numpy website. https://www.numpy.org/.

[6] Ranking in information retrieval. https://en.wikipedia.org/wiki/Ranking_(information_retrieval).

[7] Redmonk top 20 languages over time: January 2019. https://redmonk.com/rstephens/2019/03/20/redmonk-top-20-languages-over-time-january-2019/.

[8] Top programming languages 2018. https://spectrum.ieee.orga/at-work/innovation/the-2018-top-programming-languages.

[9] Why cpu clock speed isn't increasing. https://www.maketecheasier.com/why-cpu-clock-speed-isnt-increasing.

[10] Why multi core processors? https://superuser.com/a/152014.

[11] S. Ge, L. Hou U, N. Mamoulis, and D. W. Cheung. Efficient all top-k computation - a unified solution for all top-k, reverse top-k and top-m influential queries. *IEEE Transactions on Knowledge and Data Engineering*, 25(5):1015--1027, May 2013.

[12] Sumit Ghosh. Multiprocessing vs threading in python. https://blog.floydhub.com/multiprocessing-vs-threading-in-python-what-every-data-scientist-needs-to-

[13] Kyriakos Mouratidis, Spiridon Bakiras, and Dimitris Papadias. Continuous monitoring of top-k queries over sliding windows. pages 635--646, 2006.

[14] P. Nikitopoulos, G. A. Sfyris, A. Vlachou, C. Doulkeridis, and O. Telelis. Parallel and distributed processing of reverse top-k queries. pages 1586--1589, April 2019.

[15] Nikitopoulos Panagiotis. Δημιουργία μεθόδου λήψης απόφασης με χρήση αντίστροφων ερωτημάτων k κορυφαίων σημείωνσε συστήματα πραγματικού χρόνου. 2014.

[16] Bo Tang, Kyriakos Mouratidis, and Man Lung Yiu. Determining the impact regions of competing options in preference space. pages 805--820, 2017.

[17] Rohan Varma. The python gil. https://rohanvarma.me/GIL/.

[18] A. Vlachou, C. Doulkeridis, Y. Kotidis, and K. Nørvåg. Reverse top-k queries. pages 365--376, March 2010.

[19] Akrivi Vlachou, Christos Doulkeridis, Kjetil Nørvåg, and Yannis Kotidis. Identifying the most influential data objects with reverse top-k queries. *PVLDB*, 3:364--372, 09 2010.

[20] Cao Yiqun. Parallel and distributed computing techniques in biomedical engineering. 2005.

[21] Albert Yu, Pankaj K. Agarwal, and Jun Yang. Processing a large number of continuous preference top-k queries. pages 397--408, 2012.