HELLENIC REPUBLIC
National and Kapodistrian University of Athens
EST. 1837

DEPARTMENT OF INFORMATICS & TELECOMMUNICATIONS

K23a

Ανάπτυξη λογισμικού για πληροφοριακά συστήματα

Ισίδωρος Καλαμάρης
Κυριακή Καλαμάρη

Contents

1 Περίληψη

Η παρακάτω παρουσιάση αποτελεί μέρος του τελικού παραδοτέου του μαθήματος ¨Ανάπτυξη λογισμικού για πληροφοριακά συστήματα' και αποτελεί την ανάλυση των μεθοδολογίων που χρησιμοποιήθηκαν στα πλαίσια των προγραμματιστικών εργασίων. Οι προγραμματιστικές εργασίες είχαν ως στόχο τη παραγωγή $Vamana$ γράφων για την εύρεση των -περίπου- K-εγγύτερων γειτόνων, ελαχιστοποιώντας τη χρήση μνήμης που απαιτείται για την αποθήκευση τους διατηρώντας παράλληλα το *accuracy* άνω του 90%. Ολόκληρη η παρακάτω ανάλυση έχει βασιστεί στην εξής ερευνητική παρουσίαση: $Filtered_Vamana$

2 Εισαγωγή

2.1 Γενική περιγραφή

Το παραδοτέο της πρώτης εργασία εξήγαγε από μια βάση δεδομένων έναν αριθμό σημείων $128d$ τα οπόια αποτελούσαν τους εν δυνάμει γείτονες. Από αντίστοιχη διαφορετική βάση εξήγαγε τα *querypoints*, δηλαδή τα σημεία στα οποία αναζητούσαμε τους πιο κοντινούς γείτονες του. Για τα σημεία αυτά δεν υπήρχε κάποιος περιορισμός για το αν υπήρχαν και ως γείτονες ή οχι, αλλά ήταν βέβαιο πως και τα *queries* θα είχαν 128 διαστάσεις.

Το δεύτερο παραδοτέο εισήγαγε την έννοια των φίλτρων και της κατηγορίας κάποιου σημείου. Η λογική παρέμεινε όμοια με πριν, μόνο που αυτή τη φορά η αναζήτηση των κοντινότερων γειτόνων γινόταν με βάση τη κατηγορία και το φίλτρο που άνηκε το *query*. Για να ικανοποιηθούν οι παραπάνω περιορισμοί χρειάστηκε διαφορετική προσέγγιση στην αρχικοποίηση του $Filtered\ Vamana$ γράφου στην οποία θα γίνει αναφορά αργότερα.

Η προγραμματιστική προσέγγιση και των 2 ζητουμένων έγινε με βάση των ψευδοκώδικα που δινόταν στο *paper* που επισυνάπτηκε παραπάνω. Αν έχει γίνει κάποια παραδοχή θα αναφέρεται ρητά, αλλιώς εννοείται πως έχει ακολουθηθεί η δομή του ψευδοκώδικα για τις συναρτήσεις που συμβάλλουν στην αρχικοποίηση των $Vamana$ γράφων.

2.2 Περιγραφή δεδομένων

Τα δεδομένα που χρησιμοποιήθηκαν είναι ίδια με αυτά του *sigmod* διαγωνισμού. Όλα τα σημεία έχουν τον ίδιο αριθμό διαστάσεων και ακολουθούν την ίδια δομή. Συγκεκριμένα στα πλαίσια του παραδοτέου χρησιμοποιήθηκαν δεδομένα τύπου(στην επίσημη περιγραφή του διαγωνισμού αναφέρεται ως *query type*) 0 ή 1, καθιστώντας τα υπόλοιπα *queries* τύπου 2,3 απηρχαιωμένα(Ο τελικός κώδικας φροντίζει να τα αφαιρεί με το που τα εξάγει από τη βάση).

Τα δεδομένα που ανήκουν στη κατηγορία 0 θεωρούνται $unfiltered$ δεδομένα και ως *category attribute*(στα πλαίσια του μαθήματος αποτελούσε τη 2η διάσταση από συνολικά τις 104 διαστάσεις) έχουν τη τιμή -1. Τα δεδομένα που ανήκουν στη κατηγορία 1 θεωρούνται φιλτερεδ και ως *category attributes* μπορεί να έχουν κάποιους συγκεκριμένους $>=0$ $float$ αριθμούς (εντέλει είναι ακέραιοι αλλά η επίσημη περιγραφή του διαγωνισμού μιλούσε για $float$).

Ο υπολογισμός των εγγύτερων γειτόνων για τα *query type* 1, γίνεται με βάση το *category attribute* στο οποίο ανήκουν. Για παράδειγμα αν ένα σημείο ανήκει στο *category attribute* 0 τότε οι γείτονες του πρέπει αναγκαία να ανήκουν και εκείνοι στο ίδιο *category attribute*. Στη περίπτωση που κάποιο σημείο έχει *query type* $= 0$ τότε το θεωρούμε $unfiltered$ και οι γείτονες του μπορεί να ανήκουν σε οποιοδήποτε *category attribute* χωρίς κάποιο περιορισμό.

3 Ανάλυση κώδικα

Ο τελικός κώδικας αυτή τη στιγμή περιλαμβάνει τον απλό $Vamana$ αλγόριθμο, που αν και στο παραδοτέο που χρησιμοποιήθηκε δεν είχαν υπάρξει σημεία με συγκεκριμένες κατηγορίες και φίλτρα,χρησιμοποιείται από τον *stitched* γράφο και λειτουργεί κανονικά και για $filtered$ σημεία λόγο του ιδιαίτερου τρόπου που ο $Stitched\ Vamana$ αρχικοποιεί τον αντίστοιχο $Vamana$ γράφο. Επίσης κατασκευάζεται και ο $Filtered\ Vamana$ γράφος με τον αντίστοιχο αλγόριθμο. Οι δύο

γράφοι έχουν κοινό στόχο και εντέλει επιτυγχάνουν την ίδια λειτουργία, παρόλα αυτά έχουν διαφορετική υλοποίηση. Πριν γίνει παρουσίαση των 2 διαφορετικων $Vamana$ υλοποιήσεων ακολουθεί μια σύντομη ανάλυση των δεδομένων στα οποία θα εκτελεστούν.

3.1 Δομές που χρησιμοποιήθηκαν

Οι δύο κεντρικοί γράφοι είναι της δομής $map < int >, set < int >>$. Συνεπώς για κάθε κόμβο που χρειάζεται να γίνει αναζήτηση των γειτόνων του δίνουμε ως key αυτόν τον κόμβο και μας επιστρέφει το set που περιέχει τους κόμβους -γείτονες. Η χρήση σετ καθιστά σίγουρο πως δεν θα υπάρχουν διπλότυπα, αφαιρώντας έτσι την ανάγκη για έλεγχο διπλοτύπων κάθε φορά που εισαγάγαμε κάποιο νέο γείτονα μειώνοντας έτσι τον αριθμό των $branches$.

Επίσης,έγινε εκτενής χρήση $set < pair < double, int >>$ καθιστώντας έτσι τις $greedySearch$ συναρτήσεις, καθώς και την $FilteredRobust$ συνάρτηση πολύ πιο αποδοτικές, καθώς στη περίπτωση των $Greedy$ το $Lset$ ήταν μονίμως ταξινομημένο με βάση την ευκλείδια απόσταση του τωρινού σημείου που γινόταν η προσπέλαση με του $query$.Με αυτόν τον τόπο η εύρεση του ανεξερευνητου πιο κοντινού γείτονα μετατράπηκε σε μια αρκετά λιγότερο δαπανηρή διαδικασία. Ομοίως στην $Robust$ κατέστησε την εύρεση της υποψήφιας πιο κοντινής ακμής πολύ πιο αποδοτική.

Στη πλειοψηφία του κώδικα χρησιμοποιήθηκε η δομή set καθώς η αναγκή για ταξινόμηση ήταν στη πλειοψηφία της εργασίας επιτακτική, καθώς και η εύρεση και η διαγραφή διπλοτύπων επίσης. Μια πολύ καλή εναλλάκτική,όπως λέει και το $paper$ θα ήταν η χρήση $pqueue$, αν και στη συγκεκριμένη υλοποίηση με τον τρόπο που χειρίζεται τα διάφορα $sets$ είναι πολύ πιθανό να είναι ισοδύναμες επιλογές.

Οι ευκλείδιες αποστάσεις υπολογίζονται και αποθηκεύονται σε έναν $vector < vector < double >> matrix$. Η συγκεκριμένη επιλογή από τα πειράματα που ακολουθούν φανερώθηκε πως δεν είναι καθόλου καλή καθώς χρησιμοποιεί υπερβολικά πολύ μνήμη περιορίζωντας δραματικά το μέγεθος των $dataset$ που μπορουν να επεξεργαστούν οι $Vamana$ αλγόριθμοι του παραδοτέου.

3.2 $StitchedVamana$

- $StitchedVamana(DataNodes, category\ attributes, a, L\ small, R\ small, R\ stitched, vecmatrix, M, threadNum)$

  Ακολουθεί μια σύντομη περιγραφή για το ρόλο χρήσης της κάθε μεταβλητής που δέχεται ο $StitchedVamana$:

  - $DataNodes$ που ισούται με το πλήθος των σημείων που διαβάστηκε από τη βάση δεδομένων(τα σημεία στα οποία θα γίνει η αναζήτηση για τους εγγύτερους γείτονες.)
  - $category\ attributes$ τα οποία ορίζουν τα διαφορετικά φίλτρα που μπορεί να έχει ένα σημείο(κάποιος $float$ αριθμός στη περίπτωση του μαθήματος).
  - $alpha$ που δείχνει πόσο πυκνός θα είναι ο γράφος.Όσο πιο κοντά το $alpha$ είναι στο 1 τόσο πιο $dense$ θα είναι ο αντίστοιχος $Vamana$ γράφος, ενώ όσο αυξάνεται τόσο πιο $sparse$ θα αρχικοπείται ο γράφος.
  - Το L_small δείχνει το όριο των διαφορετικών σημείων που θα γίνει αναζήτηση των γειτόνων τους κατά τη διάρκεια μιας επανάληψης(Περισσότερα για αυτό στην ανάλυση του $GreedySearchalgorÐjmou$).
  - Το R_small δείχνει πόσες τυχαίες διαφορετικές ακμές θα δημιουργηθούν ανάμεσα σε σημεία με το ίδιο φίλτρο (περισσότερα για αυτό στην ανάλυση του αμανα αλγορίθμου).
  - $R\_stitched$ είναι ο αριθμός που κατά τη διάρκεια του ¨κλαδέματος' στο $RobustPrune$ ορίζει το πόσες ακμές θα απομείνουν στον τελικό $Vamana$ γράφο πριν δωθεί ως όρισμα στο $GreedySearch$. Αν το $R\_stitched$ ’= R_small τότε ο αριθμός των τελικών ακμών θα ισούνται με το αρχικό R_small.
  - $vecmatrix$: Ένας $2d$ πίνακας όπου αποθηκεύονται οι ευκλείδιες αποστάσεις του κάθε σημείου που υπάρχει στο κυρίως $dataset$ που θα γίνει η αναζήτηση γειτόνων μεταξύ κάθε άλλου υποψήφιου γείτονα.

– M είναι μια δομή map όπου για κλειδί έχει το νούμερο ενός σημείου και ως κλειδί έχει το αντίστοιχο σημείο που έχει μια μέση απόσταση από αυτό (με βάση τη μικρότερη απόσταση του πιο κοντινού γείτονα του, έως και τη μεγαλύτερη απόσταση του).

– $thread\,num$ είναι ο αριθμός νημάτων που χρησιμοποιούνται σε σημεία που επιτυγχάνεται παραλληλία. Περισσότερα για αυτό στη συνέχεια.

Η κεντρική λογική του $Stitched\ Vamana$είναι πως αντιστοιχεί κάθε φίλτρο με τα αντίστοιχα σημεία που έχουν το συγκεκριμένο φίλτρο ως όρισμα. Έπειτα, για όλα αυτά τα σημεία κάθε φίλτρου καλεί τον $vamana_index_algorithm$ που φτιάχνει τον $Vamana$ γράφο και τον αντιστοιχεί συγκεκριμένα για αυτό το σημείο(Άρα αργότερα που θα γίνεται η αναζήτηση των κ-εγγύτερων γειτόνων, στη $Filtered-greedysearch$ θα δωθεί ως όρισμα ο γράφος που αντιστοιχεί στο συγκεκριμένο φίλτρο και η αναζήτηση θα γίνει αποκλειστικά σε αυτόν τον γράφο). Στο τέλος επιστρέφει ένα Map που για κάθε φίλτρο έχει αντιστοιχήσει τον αντίστοιχο γράφο. Συνεπώς ο $Stitched\ Vamana$ γράφος είναι πολλοί διαφορετικοί ανεξάρτητοι γράφοι μεταξύ τους που ο κάθε ένας αντιστοιχεί σε αποκλειστικά ένα φίλτρο.

– $vamana\quad index\quad algorithm$
Η συγκεκριμένη συνάρτηση είναι υπεύθυνη για τη κατασκευή του $vamana$ γράφου που θα περιέχει αποκλειστικά σημεία με το ίδιο φίλτρο και το κάθε σημείο θα έχει το πολύ R ακμές.Αρχικά παράγει έναν γράφο ορίζοντας τυχαίες ακμές για το κάθε σημείο. Έπειτα διασχίζει με τυχαίο τρόπο το κάθε σημείο που υπάρχει στον γράφο και καλεί τη $greedysearch$ συνάρτηση για αυτό αποκλειστικά το σημείο(η $greedysearch$ θα αναλυθεί περαιτέρω στη συνέχεια). Η $greedysearch$ συνάρτηση επιστρέφει τους πιο κοντινούς γείτονες για το σημείο που δόθηκε ως όρισμα καθώς και και ένα $setV$ που περιέχει όλα τα σημεία που προσπελάστηκαν κατά τη διάρκεια της αναζήτησης.Το συγκεκριμένο set δίνεται ως όρισμα στο $Robustprune$ και μέσα από αυτό αντικαθιστώνται οι παλιές τυχαίες ακμές του τυχαίου σημείου που αυτή τη στιγμή προσπελάζεται οι σωστές ακμές που δείχνουν όντως στους πιο κοντινούς γείτονες του.

– $greedysearch(graph, s, query_point, k_neigh, L_sizelist, querymatrix)$

* $graph$: Ο προς το παρών τυχαίος γράφος που αρχικοποιήθηκε στη $Vamana$ συνάρτηση.
* $querypoint$ Είναι το σημείο για το οποίο θα βρεθούν οι πιο κοντινοί γείτονες. Δεν χρειάζεται να ανήκει απαραίτητα στο $query\ dataset$,κατά τη διάρκεια της δημιουργίας του τελικού $vamana$ γράφου το σημείο που δίνουμε ως όρισμα ανήκει στο $dataset$ των εν δυνάμει γειτόνων και όχι των χυεριες. Αυτό συμβαίνει γιατί ακριβώς ψάχνουμε τους πιο κοντινούς γείτονες αυτών των σημείων ώστε να αναδιαμορφώσουμε τις τυχαίες ακμές του συγκεκριμένου σημείου με τις οποίες είχε αρχικοποιηθεί στον $vamana$ με τους πιο κοντινούς του γείτονες.
* $k\ neigh$ Ορίζει τον αριθμό κοντινότερων γειτόνων που θα επιστραφούν από τη συνάρτηση.
* $L\ sizelist$ είναι ένας ακέραιος αριθμός $>= k$ και ορίζει το πόσο εύρος θα έχει η αναζήτηση του $greedysearch$. Κατά την αναζήτηση στο $L\ set$ υπάρχουν οι -προς το παρών- πιο κοντινοί γείτονες που έχουν βρεθεί και αναζητώνται οι γείτονες αυτών των σημείων ως υποψήφιοι πιο κοντινοί γείτονες στο αρχικό σημείο που δώθηκε ως όρισμα στη $greedysearch$. Συνεπώς αυξάνωντας το μέγεθος του $L_sizelist$ η αναζήτηση παίρνει υπόψη της και πιο μακρινούς υπογράφους οι οποίοι μπορεί να κρύβουν κάποιο γείτονα πιο κοντά στο σημείο για το οποίο γίνεται η αναζήτηση. Αντίθετα όσο πιο μικρό το $L_sizelist$ τόσο παραπάνω περιορίζεται η αναζήτηση με αποτέλεσμα να μη συμπεριληφθούν σημεία που μπορεί να οδηγούσαν στην εύρεση κοντινότερων γειτόνων.
* $querymatrix$ το όνομα της μεταβλητής είναι παραπλανητικό,κατά τη διάρκεια της δημιουργίας του $vamana$ γράφου δίνεται ως όρισμα το $vecmatrix$, που όπως ήδη

ειπώθηκε περιέχει τις ευκλείδιες αποστάσεις του κάθε σημείου της βάσης γειτόνων μεταξύ του άλλου.

Η $greedysearch$ συνάρτηση είναι υπεύθυνη για την εύρεση των εγγύτερων γειτόνων ενός συγκεκριμένου σημείου. Το σημείο αυτό μπορεί να ανήκει στη βάση δεδομένων των υποψήφιων γειτόνων ή να είναι το $query$ σημείο για το οποίο αναζητούμε τους γράφους. Κατά τη διάρκεια της δημιουργίας του $vamanaindex$ (κατέπέκταση του $stitchedVamana$) δίνεται ως όρισμα ο γράφος με τις -προς το παρών- τυχαίες ακμές και το τυχαίο σημείο στο οποίο βρισκόμαστε αυτή τη στιγμή, με τη τυχαία διάσχιση των ακμών, που ετκελείται στη $vamana$ συνάρτηση. Σταδιακά αναζητά όλα τα σημεία του γράφου και τα τοποθετεί με αύξουσα σειρά σε ένα $setL$, βάση την απόσταση που απέχουν από το σημείο που δώσαμε ως όρισμα.Συνεπώς σταδιακά το L περιέχει τους $L_sizelist$ πιο κοντινούς γείτονες του σημείου. Η $greedysearch$ συνάρτηση επιστρέφει το $setL$ που περιέχει τους k_neigh όπως και το $setV$ που περιέχει όλα τα σημεία που προσπελάστηκαν για την επίτευξη εύρεσης των εγγύτερων γειτόνων.

- $RobustPrune(graph, point, candidateSet, alpha, R, vecmatrix)$
- $graph$: Ο γράφος με τις τυχαία αρχικοποιημένες ακμές που δημιουργήθηκε στη $vamana$ συνάρτηση.
- $point$: Το σημείο στο οποίο βρίσκεται η τυχαία προσπέλαση που γίνεται στη $vamana$ συνάρτηση.
- $candidateset$: Το $Vset$ που επέστρεψε η $greedysearch$ συνάρτηση.
- $alpha$ η σταθερά που με βάση πόσο μεγάλη / μικρή είναι καθορίζεται το πόσο αυστηρό θα είναι το $pruning$
- R : Ακέραιος αριθμός που καθορίζει το μέγιστο πλήθος των ακμών που θα απομείνει μετά το κάλεσμα της συνάρτησης.
- $vecmatrix$: $2d\ matrix$ που περιέχει τις ευκλείδιες αποστάσεις του κάθε σημείου της βάσης γειτόνων μεταξύ του άλλου.

Η $RobustPrune$ συνάρτηση διαγράφει τις τυχαίες ακμές του σημείου που αρχικοποιήθηκε στη $vamana$, βρίσκει το πιο κοντινό σημείο που υπάρχει στο $candidateSet$ και δεν έχει ήδη επιλεχθεί και το τοποθετεί ως νέα ακμή στο σημείο που γίνεται η τυχαία προσπέλαση από τη $vamana$ συνάρτηση. Έπειτα διαγράφει όλα τα σημεία μέσα στο $candidateset$ που έχουν μεγαλύτερη απόσταση από το τωρινό σημείο που μόλις πρόσθεσε ως ακμή $\times a$ και επαναλαμβάνει τη διαδικασία είτε μέχρι να αδειάσει το $candidateset$ είτε να φτάσει στον αριθμό του R που δόθηκε ως όρισμα.

3.3 FilteredVamana

- $FilteredVamanaIndex(vecmatrix, DataNodes, alpha, R, category_attributes, M$:
  - $Vecmatrix$: $2d\ matrix$ που περιέχει τις ευκλείδιες αποστάσεις του κάθε σημείου της βάσης γειτόνων μεταξύ του άλλου.
  - $DataNodes$: Περιέχει όλα τα σημεία που διαβάστηκαν από τη κύρια βάση δεδομένων που ορίζει τους -εν δυνάμει- κοντινότερους γείτονες
  - $alpha$: Η λογική της συγκεκριμένης μεταβλητής παραμένει ακριβώς όπως και στον $StitchedVamana$
  - R :Ο μέγιστος αριθμός ακμών που μπορεί να περιέχει ένα σημείο στον $FilteredVamana$γράφο
  - $category\ attributes$: Τα διαφορετικά φίλτρα που μπορεί να έχει ένα σημείο(κάποιος ακέραιος αριθμός στη περίπτωση του μαθήματος).

– M : Αποτελεί το $medoid$και έχει τον ίδιο ρόλο με πριν στον $stitched\ Vamana$, περιέχει δηλαδή για κάθε σημείο έναν γείτονα που απέχει κατά Μ.Ο. τη μέση απόσταση.

Αυτή τη φορά η $Filtered\ Vamana$ συνάρτηση δεν αρχικοποιεί κάποιο γράφο με τυχαίες ακμές. Αντάυτού ξεκινάει άμεσα να διασχίζει με τυχαίο τρόπο το κάθε σημείο που υπάρχει στο $DataNodes$. Για αυτό το σημείο καλεί τη $FilteredGreedySearch$ και δίνει το $Vset$ που περιέχει όλα τα σημεία που επισκέφθηκε ως όρισμα στο $FilteredRobust$. Με βάση αυτά ο $Robust$ αρχικοποιεί σταδιακά τον τελικό $FilteredVamana$γράφο. Τέλος ο $FilteredVamana$ προσπαθεί οι περισσότερες ακμές να είναι συνεκτικές μεταξύ τους έτσι ώστε ο τελικός γράφος να είναι $well\ connected$,αν και δεν επιτυγχάνεται πάντα.

– $FilteredGreedy(graph, xq, knn, L_sizelist, M, Fq, querymatrix, datanodes, category\ attributes)$

  * $graph$: Στις αρχικές εκτελέσεις ο γράφος θα είναι κενός, χρησιμοποιείται στην $greedysearch$ για να αναζητήσει τους γείτονες του πιο κοντινού σημείου που εντόπισε και να προσθέσει τον πιο κοντινό από αυτούς στο L.Στις πρώτες εκτελέσεις το Λ θα περιέχει απλά το $medoid$ του φίλτρου που ανήκει το σημείο που δώθηκε ως $query(xq)$.
  * xq : Το σημείο για το οποίο γίνεται η αναζήτηση των εγγύτερων γειτόνων. Όπως και πριν αυτό το σημείο μπορεί να ανήκει και στο $query\ dataset$ αλλά και στο $dataset$ των γειτόνων.
  * knn : Ο αριθμός εγγύτερων γειτόνων που ο χρήστης θέλει να επιστραφεί.
  * $L\ sizelist$: Χρησιμοποιείται ακριβώς με τον ίδιο τρόπο και επηρεάζει με τον ίδιο τρόπο την αναζήτηση. Όσο μεγαλύτερο, τόσο πιο ¨μακριά' θα αναζητήσει για γείτονες. Αντίθετα όσο πιο μικρό τόσο πιο ¨κοντά' θα παραμείνει η αναζήτηση.
  * M : Περιέχει το $medoid$ για κάθε διαφορετικό φίλτρο που υπάρχει στο $dataset$ των γειτόνων.
  * Fq : είναι το φίλτρο στο οποίο ανήκει το σημείο για το οποίο εκτελείται η αναζήτηση
  * $querymatrix$ Πάλι ατυχής επιλογή ονόματος, κατά τη διάρκεια της αρχικοποίησης του $FilteredVamana$ γράφου δίνεται το $vecmatrix$ το οποίο όπως προ αναφέρθηκε είναι $2d$ πίνακας που περιέχει τις ευκλείδιες αποστάσεις μεταξύ των κεντρικού $dataset$.
  * $datanodes$ Περιέχει όλα τα σημεία που προήλθαν από το κεντρικό $dataset$.
  * $category\ attributes$ Περιέχει όλα τα διαφορετικά φίλτρα που υπάρχουν για τα συγκεκριμένα $datasets$($float$ αριθμοί)

  Η κύρια λειτουργία της $FilteredGreedySearch$ συνάρτησης είναι να βρίσκει το πιο κοντινό σημείο από το L(L είναι set που αποθηκεύει τα σημεία με αύξουσα σειρά με βάση την ευκλείδια απόσταση τους). Έπειτα βρίσκει τους γείτονες τους και τους εισάγει στο L. Συνεπώς παράγει σταδιακά το $Lset$ που περιέχει τους κ εγγύτερους γείτονες.

– $FilteredRobustPrune(graph, sigma, V, alpha, R, vectormatrix, DataNodes)$

  * $graph$ Είναι ο κενός γράφος που δημιουργείται στο $FilteredVamana$.Χτίζεται σταδιακά στο $RobustPrune$, συνεπώς όσο προχωράει και η τυχαία προσπέλαση στο $FilteredVamana$ θα αυξάνονται τα στοιχεία που περιέχει.
  * $sigma$ Είναι το σημείο στο οποίο γίνεται η τυχαία προσπέλαση στο $FilteredVamana$ αυτή τη στιγμή.
  * V Περιέχει τα σημεία που επισκέφτηκε ακριβώς πριν ο $FilteredGreedySearch$ ώσπου να βρει τους κ εγγύτερους γείτονες.
  * $alpha$ Ίδια λογική με αυτή που περιγράφηκε και πριν στον απλό $robust$ που καλούσε ο $vamana\ index\ algorithm$
  * R Αποτελεί το όριο των ακμών που μπορεί να έχει ένα σημείο στο γράφο.

* *vectormatrix* Επιτέλους,σωστή ονομασία! Το περιεχόμενο του παραμένει ίδιο με πριν
* *DataNodes* Επίσης το περιεχόμενο του παραμένει ίδιο με πριν.

Η *FilteredRobustPrune* αυτή τη φορά όχι μόνο κρατάει το πλήθως των ακμών ενός σημείου μέσα σε κάποιο όριο αλλά προσθέτει ακμές και στον αρχικά κενό γράφο. Από ένα *candidateSet* που περιέχει τα στοιχεία του V που προήλθαν από την εκτέλεση του *FilteredGreedySearch* επιλέγει το σημείο που απέχει λιγότερο από το *sigma* και το προσθέτει στον γράφο. Επειτα διαγράφει όλα τα σημεία μέσα στο *candidateset* που έχουν μεγαλύτερη απόσταση από το τωρινό σημείο που μόλις πρόσθεσε ως ακμή×a και επαναλαμβάνει τη διαδικασία είτε μέχρι να αδειάσει το *candidateset* είτε να φτάσει στον αριθμό του P που δόθηκε ως όρισμα.

3.4 main

Η *main* συνάρτηση έχει τρεις βασικούς ρόλους:

1. Προετοιμασία και αρχικοποίηση των ορισμάτων που δίνονται ως ορίσματα στη *Stitched* συνάρτηση που με τη σειρά της καλεί την *vamana indexing algortihm* και στη $FilteredVamana$ συνάρτηση.

   Αρχικά η *main* συνάρτηση δέχεται ορίσματα από τον χρήστη για το πόσες φορές θέλει να εκτελεστεί το τωρινό *instance*, πόσα *threads* επιθυμεί να χρησιμοποιηθούν κατά την εκτέλεση του προγράμματος και αν θέλει να γίνει ο υπολογισμός του *groundtruth* αρχείου *manually* ή να διαβαστεί από κάποιο *txt* αρχείο που σε κάποιες περιπτώσεις μπορεί ήδη να υπάρχει. Επίσης ο χρήστης ως τελευταίο όρισμα μπορεί να δώσει το όνομα κάποιας συγκεκριμένης συνάρτησης που επιθυμεί να εκτελέσει μεμονομένα για τυχόν $profiling$. Στη συνέχεια υπολογίζει τις ευκλείδιες αποστάσεις, αρχικά για τον δισδιάστατο πίνακα *vecmatrix* που είναι οι αποστάσεις μεταξύ των σημείων από το *dataset* των γειτόνων και έπειτα το *querymatrix* που αντικατροπτρίζει την απόσταση του κάθε *query* σημείου μεταξύ κάποιου συγκεκριμένου γείτονα, π.χ. το $querymatrix[2][4]$ δείχνει την απόσταση του *query* σημείου 2 από το σημείο γείτονα 4. Τέλος υπολογίζει τα μοναδικά φίλτρα που ένα από αυτά μπορεί να περιέχει κάποιο σημείο.

2. Αρχικοποίηση των δύο διαφορετικών $Vamana$ γράφων *stitched* και $filtered$. Με βάση τα ορίσματα που υπολογίστηκαν στο προηγούμενο μέρος της μαιν συνάρτησης καλεί σειριακά τη *stitched* και τη $FilteredVamana$ αντίστοιχα αρχικοποιώντας έτσι τους αντίστοιχους γράφους στους οποίους στη συνέχεια θα εκτελέσει την *greedysearch* αναζήτηση.

3. Έυρεση των κ εγγύτερων γειτόνων για κάθε σημείο που διαβάστηκε από το *querydataset* και υπολογισμός του συνολικού *recall* και για τους 2 γράφους που δημιουργήθηκαν

   Ο υπολογισμός γίνεται σειριακά για τον κάθε γράφο. Πρώτα δημιουργείται ο *stitched* γράφος και με βάση αυτόν για κάθε *query* σημείο εκτελείται η συναρτηση *FilteredGreedySearch* επιστρέφωντας έτσι τους πιο κοντινούς γείτονες για αυτό το σημείο. Η συγκεκριμένη διαδικασία επαναλαμβάνεται για όλα τα *query* σημεία και συγκρίνωντας κάθε φορά τους γείτονες που επέστρεψε η *FilteredGreedySearch* με τους γείτονες του σημείου που υπάρχουν στο *groundtruth* υπολογίζεται το *recall*. Η ίδια διαδικασία επαναλαμβάνεται και για το $FilteredVamana$ γράφο στη συνέχεια.

3.5 Σύγκριση $Vamana$ συναρτήσεων

Όπως προ αναφέρθηκε οι 2 $Vamana$ συναρτήσεις, αν και χρησιμοποιούνται για τον ίδιο σκοπό(την δημιουργία αποδοτικών και εύκολα προσπελάσιμων γράφων για την αναζήτηση εγγύτερων γειτόνων) ακολουθούν διαφορετική λογική στο τρόπο αρχικοποίησης τους:

- Ο $FilteredVamana$ ξεκινάει την δημιουργία του τελικού γράφου από έναν αρχικά κενό γράφο. Σταδιακά βρίσκει τους πιο κοντινές γείτονες του κάθε σημείου σχηματίζει ακμές και επιβεβαιώνει πως στη πλειοψηφία τους τα σημεία που έχουν ίδιες ακμές θα είναι $well - connected$ μεταξύ τους.
- Ο $StitchedVamana$ από την άλλη αρχικά έχει έναν τυχαία αρχικοποιημένο γράφο που σταδιακά αλλάζει αυτόν τον γράφο και τον φέρνει στη τελική $Vamana$ μορφή του. Κατά τη δημιουργία τους όμως παίρνει υπόψη του και πιο μακρινά σημεία που ο $FilteredVamana$ λόγο της έλλειψης τυχαιότητας στον αρχικό του γράφο δεν υπολογίζει.

Στη συνέχεια φαίνεται πως εντέλει οι 2 διαφορετικές τεχνικές έχουν όμοια αποτελέσματα αν και ο χρόνος εκτέλεσης τους καθώς και το $recall$ τους βάλλωνται από διάφορα $trade - offs$ που έχουν να κάνουν με τον αντίστοιχο τρόπο που επιλέγουν να αρχικοποιήσουν το τελικό $Vamana$ γράφο.

3.6 Παραλληλία

Για $optimization$ του κώδικα προστέθηκε παραλληλία σε σημεία που γινόντουσαν μεγάλου μεγέθους υπολογισμοί καθώς και στη παράλλη δημιουργία του γράφου $Vamana$ στο $stitched.cpp$. Για την εφαρμογή της παραλληλίας χρησιμοποιήθηκε η διεπαφή του $OpenMp$ και κάποιες $simd$ εντολές.

1. Υπολογισμός ευκλείδιας απόστασης:

   Όπως προ αναφέρθηκε, υπολογίζουμε ευκλείδιες αποστάσεις για 2 μεγάλους $2d\ matrixes$ συνεπώς η παραλληλία είναι ιδανική για μείωση του χρόνου των υπολογισμών. Έχει χρησιμοποιηθεί η διεπαφή $OpenMp$ καθώς και για το $vecmatrix$ έχουν χρησιμοποιηθεί $simd$ μεταβλητές $m256$ και αντίστοιχες συναρτήσεις. Επίσης έχει χρησιμοποηθεί η τεχνική $blocking$ για να μειωθούν τα $cache\ misses$ που αναπόφευκτα λόγο του μεγάλου μεγέθους του πίνακα θα συνέβαιναν. Για τον πίνακα $querymatrix$ χρησιμοποιήθηκαν με αντίστοιχη λογική $simdinstructions$ καθώς και $openMp$ παραλληλία, αλλά αποφεύχθηκε η χρήση της μεθόδου $blocking$ λόγο ότι προσέφερε ελάχιστη βελτίωση -πολλές φορές καθόλου- κάνωντας την ίδια στιγμή τον κώδικα αρκετά πιο δυσανάγνωστο. Η έλλειψη της βελτίωσης πιθανώς οφείλεται στο μειωμένο μέγεθος του συγκεκριμένου πίνακα σε σχέση με τον $vecmatrix$.

2. Υπολογισμός $groundtruth$:

   Γίνεται με απλό $brute - force$ τρόπο δηλαδή για κάθε $node$ συγκρίνει τις αποστάσεις του με τα σημεία των γειτόνων που διαβάστηκαν από το άλλο $dataset$ και βρίσκει έτσι σταδιακά τα 100 μικρότερα σημεία.

3. παραλληλία στο $graph\ creation$ της $Vamana$ συνάρτησης.

   Η συνάρτηση $graph\ creation$ δέχεται ένα σύνολο σημείων το οποίο το διασχίζει με παράλληλο τρόπο ορίζωντας την ίδια στιγμή για το κάθε ένα τυχαίους γείτονες. Η παραλληλία περιορίζεται τη στιγμή που ένα $thread$ πρέπει να περάσει το τοπικό γράφο στη κεντρική $shared$ μεταβλητή $graph$. Επειδή το κάθε σημείο ακολουθεί αυτή την τακτική, αν και υπάρχει βελτίωση είναι αρκετά περιορισμένη λόγο ότι η εγγραφή στη $shared$ μεταβλητή γίνεται σε μεγάλο βαθμό σειριακά.

3.7 παραδοχές / βελτιώσεις

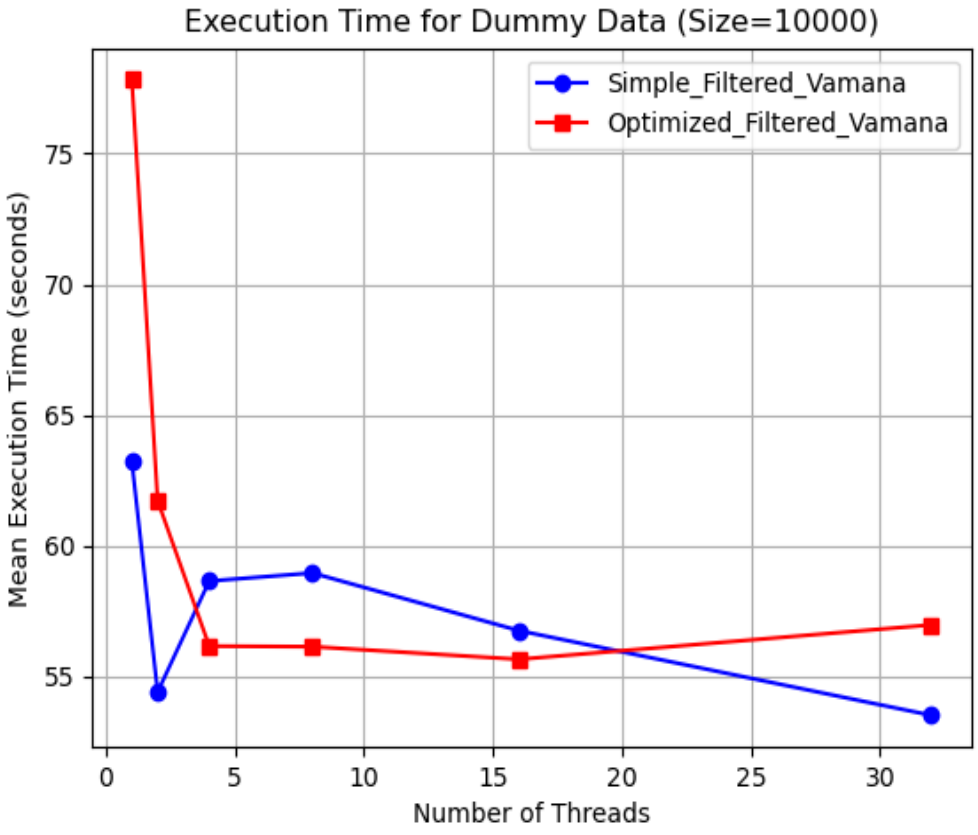
- unfiltered nodes

  Στον $FilteredVamana$ γράφο τα $unfilterednodes$ χρειάστηκαν ειδική μεταχείριση λόγο του ότι ο γράφος που παράγεται δεν είναι $well$ - $connected$, συνεπώς η αρχική αναζήτηση πριν τη παραδοχή περιοριζόταν σε κάποιο υπό-μέρος του $Vamana$ γράφου ανάλογα το $medoid$που

δεχόταν και δεν επεκτεινόταν παραπάνω. Η παραδοχή που διόρθωσε το συγκεκριμένο πρόβλημα ήταν η εύρεση του πιο κοντινού σημείου για κάθε ένα διαφορετικό φίλτρο που υπάρχει διαθέσιμο σε σχέση με το $unfiltered\ query$ σημείο. Συνεπώς αντί για το $medoid$ σημείο στη $FIlteredGreedy$ αύτη τη φορά δίνεται ως όρισμα ένα σετ με κάθε σημείο να έχει δαιφορετικό φίλτρο και να αποτελεί το πιο κοντινό γείτονα, για αυτό το συγκεκριμένο φίλτρο, στο $query\ point$. Με αυτό το τρόπο η αναζήτηση επεκτείνεται παραπάνω,ανάλογα φυσικά και το μέγεθος της μεταβλητής $L\ sizelist$

- Αρχικοποίηση γράφων με τυχαίες ακμές

  Αυτή η βελτιστοποίηση έγινε αποκλειστικά για το $FilteredVamana$ καθώς ο $stitched$ που χρησιμοποιεί τον απλό $Vamana$ αρχικοποεί εκ προεπιλογής έναν τυχαίο γράφο. Η συγκεκριμένη βελτιστοποίηση,όπως μπορεί να φανεί και στο παρακάτω διάγραμμα, δεν προσέφερε κάποια βελτίωση στο χρόνο εκτέλεσης, ούτε φάνηκε να βελτιώνει το $recall$. Συνεπώς στο τελικό παραδοτέο κρατήθηκε η $default$ επιλογή του ψευδοκώδικα με τον άδειο γράφο. Ακολουθεί το σχετικό διάγραμμα:



- Μεταβλητές L_small και $L\_sizelist$:

  Στο συγκεκριμένο παραδοτέο το L_small χρησιμοποιείται ως όρισμα κατά το κτίσιμο την αρχικοποίηση των $vamana\ indexes$ ενώ το $L\_sizelist$ δίνεται ως όρισμα κατά τη διάρκεια του υπολογισμού του $recall$. Αυτό συμβαίνει γιατί παρατηρήθηκε πως μέχρι ένα συγκεκριμένο μέγεθος του L_small το $recall$ των 2 $indexes$ αυξάνεται με αρκετά όμοιο τρόπο, συνεπώς κρίθηκε σωστό να χρησιμοποιηθεί ως όρισμα και για τα 2 $indexes$.

3.8 μεταβλητές

Οι μεταβλητές που κρίνεται να ορίσει σε κάθε εκτέλεση ο χρήστης είναι οι εξής: $alpha, R, KNN, L_sizelist, Rsmall$ $Lsmall, Rstitched$

- *alpha*: Έχει γίνει ήδη αναφορά στη συγκεκριμένη και όπως είχε αναφερθεί επηρεάζει σε πόσο μεγάλο βαθμό θα αφαιρούνται υποψήφια σημεία για εισαγωγή ως ακμή στο σημείο για το οποίο εκτελείται η αντίστοιχη *Robust* συνάρτηση.
- *R*: το όριο των ακμών για τον $Filtered\_Vamana$ γράφο.
- KNN Το πλήθος των κωντινότερων γειτόνων που θέλουμε να βρούμε για κάποιο σημείο
- *L_sizelist L_small*: Ισχύει η παραδοχή που περιγράφηκε πιο πάνω.
- *Rsmall*: ορίζει το πλήθος των ακμών στο τυχαίο γράφο που φτιάχνει αρχικά η $vamana\_indexing\_algorithm$ συνάρτηση.
- *Rstitched*: Ορίζει το πλήθος των ακμών του τελικού $Vamana$ γράφου που αρχικοποιείται σταδιακά από την *Robust* συνάρτηση.

4 Πειράματα

Δυστυχώς λόγο ατυχής επιλογής χρήσης δομής για τον υπολογισμό της ευκλείδιας απόστασης δεν μπορέσαμε να τρέξουμε δεδομένα 40000 λόγο του $2d\ matrix$ που χρησιμοποιήθηκε. Ακολουθούν μερικά παραδείγματα για τον χρόνο εκτέλεσης που χρειάζονται κάποια μεγαλύτερα *datasets*:

Τα παρακάτω πειράματα εκτελέστηκαν στο εξής configuration: article listings [margin=1in]geometry

```
Architecture:            x86_64
CPU op-mode(s):          32-bit, 64-bit
Address sizes:           39 bits physical, 48 bits virtual
Byte Order:              Little Endian
CPU(s):                  16
On-line CPU(s) list:     0-15
Vendor ID:               GenuineIntel
Model name:              13th Gen Intel(R) Core(TM) i5-13500H
  CPU family:            6
  Model:                 186
  Thread(s) per core:    2
  Core(s) per socket:    8
  Socket(s):             1
  Stepping:              2
  BogoMIPS:              6374.42
  Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep
                         mtrr pge mca cmov pat pse36 clflush mmx fxsr
                         sse sse2 ss ht syscall nx pdpe1gb rdtscp lm
                         constant_tsc rep_good nopl xtopology
                             tsc_reliable
                         nonstop_tsc cpuid pni pclmulqdq vmx ssse3 fma
                         cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt
                         tsc_deadline_timer aes xsave avx f16c rdrand
                         hypervisor lahf_lm abm 3dnowprefetch
                             invpcid_single
                         ssbd ibrs ibpb stibp ibrs_enhanced tpr_shadow
                         vnmi ept vpid ept_ad fsgsbase tsc_adjust bmi1
                         avx2 smep bmi2 erms invpcid rdseed adx smap
```

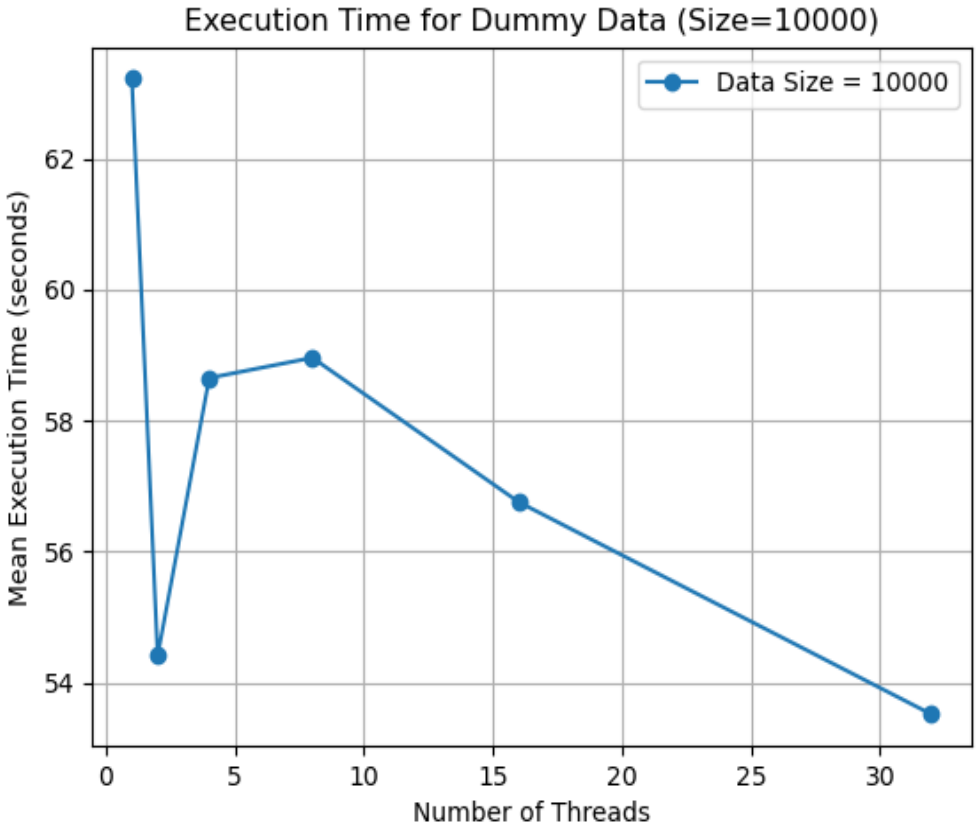
```
                        clflushopt clwb sha_ni xsaveopt xsavec
                           xgetbv1
                        xsaves umip waitpkg gfni vaes vpclmulqdq
                           rdpid
                        movdiri movdir64b fsrm md_clear serialize
                        flush_l1d arch_capabilities
Virtualization features:
Virtualization:         VT-x
Hypervisor vendor:      Microsoft
Virtualization type:    full
Caches (sum of all):
  L1d:                    384 KiB (8 instances)
  L1i:                    256 KiB (8 instances)
  L2:                     10 MiB (8 instances)
  L3:                     18 MiB (1 instance)
Vulnerabilities:
  Itlb multihit:          Not affected
  L1tf:                   Not affected
  Mds:                    Not affected
  Meltdown:               Not affected
  Spec store bypass:      Mitigation; Speculative Store Bypass
     disabled via prctl and seccomp
  Spectre v1:             Mitigation; usercopy/swapgs barriers and
     __user pointer sanitization
  Spectre v2:             Mitigation; Enhanced IBRS, IBPB conditional,
      RSB filling
  Srbds:                  Not affected
  Tsx async abort:        Not affected
```

-16 GB ram (Dual-channel)

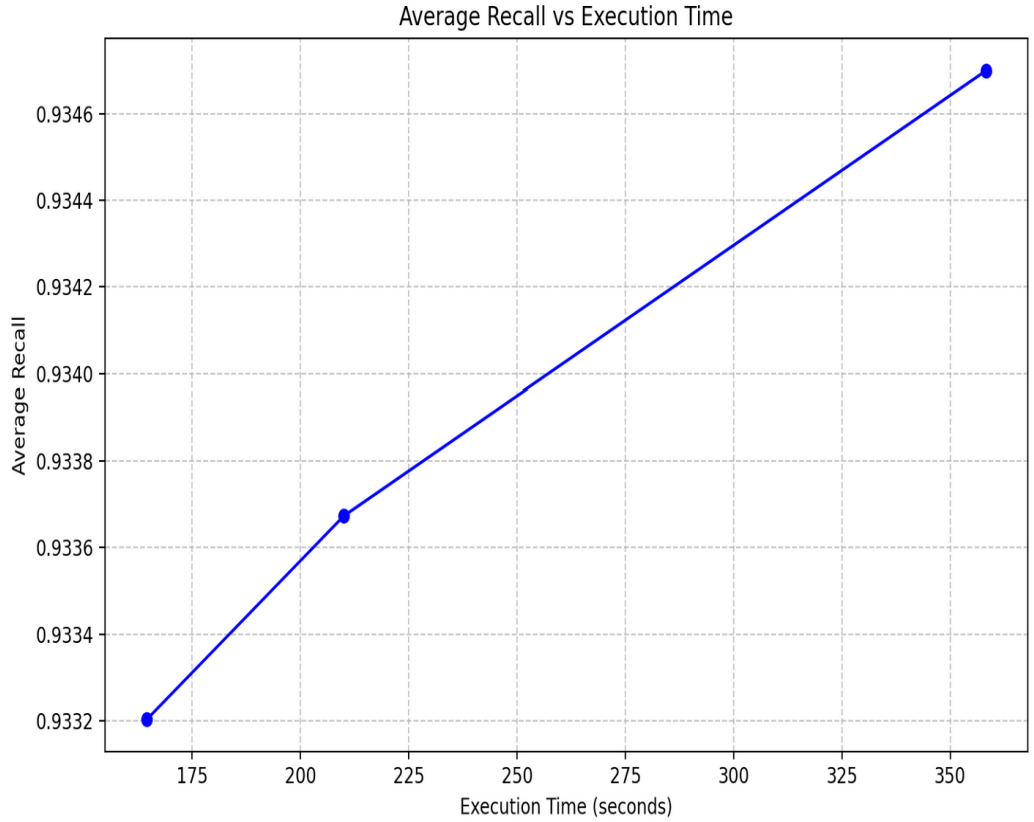
Στο παρακάτω διάγραμμα το κάθε *instance* έτρεξε 2 φορές και οι τιμές του διαγράμματος αποτελούν τον Μ.Ο. αυτών των δύο διαφορετικών εκτελέσεων.

- "alpha": 2.0,
- "R": 25,
- "knn": 100,
- "Lsizelist": 120,
- "Rsmall": 45,
- "Lsmall": 160,
- "Rstitched": 30
- "FilteredRecall": 0.923939
- "StitchedRecall":0.920822

- "queries$_{p}er_{s}econd$" : 682



Γίνεται ξεκάθαρο πως αν και η παραλληλία που εφαρμόστηκε ήταν πολύ περιορισμένη και σε συγκεκριμένα μέρη του κώδικα φαίνεται πως έχει αρκετό αντίκτυπο στο χρόνο εκτέλεσης κάποιου $instance$, με τη μεγαλύτερη απόδοση να βρίσκεται στα 32 νήματα.Είναι σημαντικό να αναφερθεί πως διαφορετικά $configurations$ έχουν διαφορετική ¨ευαισθησία' και αντίδραση στη παραλληλία. Μέσα και άλλως πειραμάτων παρατηρήθηκε πως στη μέση περίπτωση το πλήθος νήματων που δίνουν τη μεγαλύτερη μείωση χρόνου είναι τα 16.

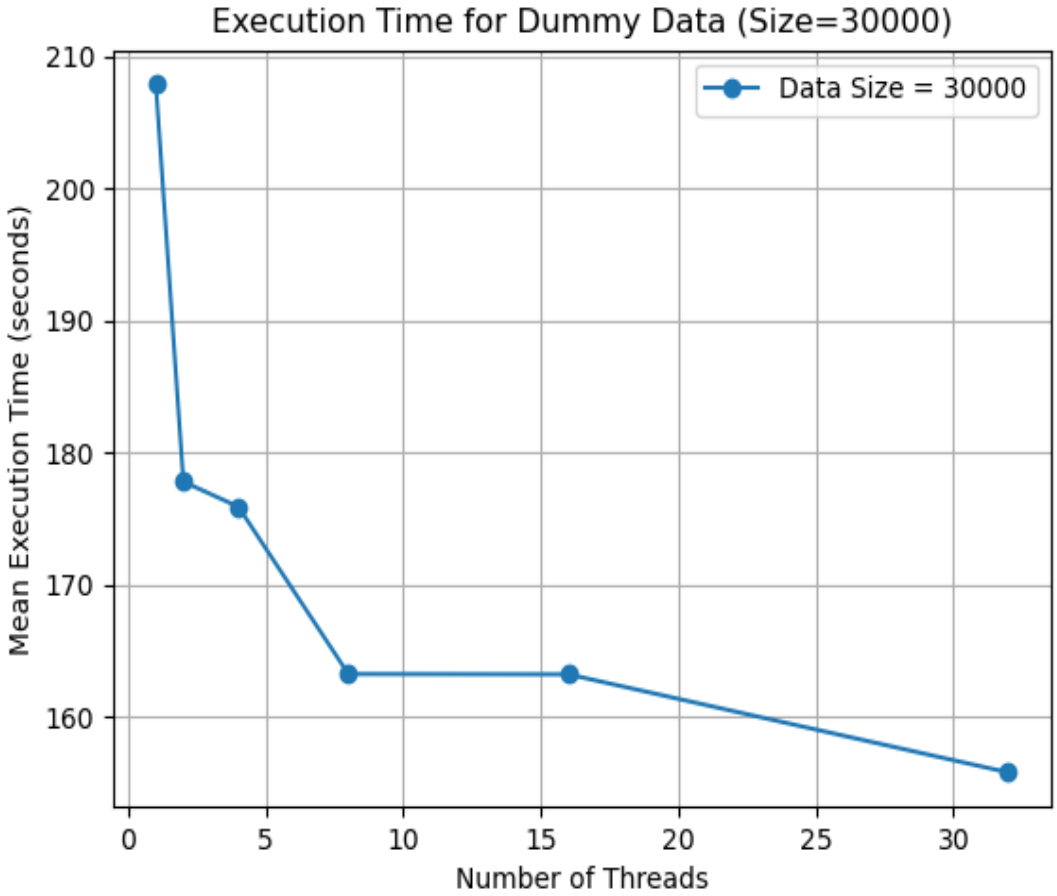


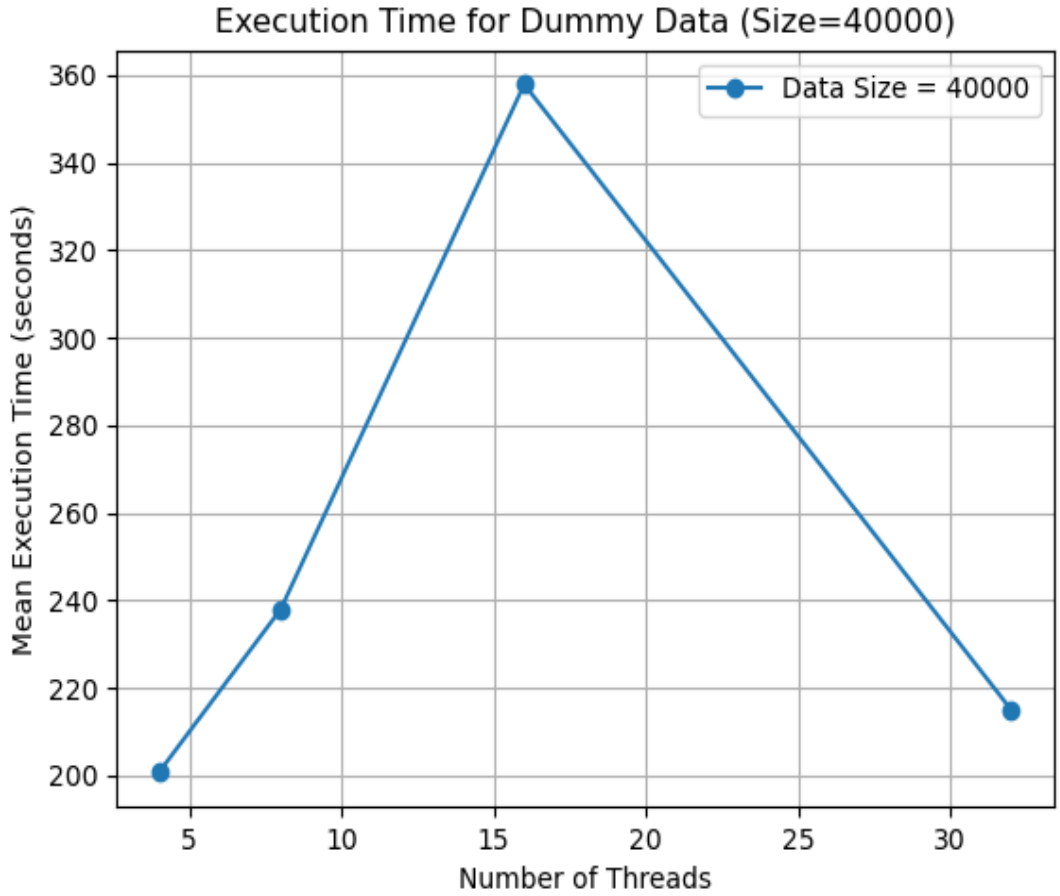
Αν και θα περίμενε κανείς πως όσο αυξάνονται τα ορίσματα - άρα και ο χρόνος εκτέλεσης- θα αυξανόταν και το *recall*, στο συγκεκριμένο παράδειγμα δεν πάει έτσι. Όπως γίνεται αντιληπτό από το παραπάνω διάγραμμα είναι πως το *recall* κάποια στιγμή έρχεται σε τέλμα περίπου στο 93.3%. Αν και ο χρόνος εκτέλεσης διπλασιάστηκε το *recall* αυξήθηκε λίγες μονάδες στα δέκατα των χιλιοστών, -οριακά μηδαμινή βελτιώση-.

Από τα δύο παραπάνω πειράματα γίνεται κατανοητό πως αν και για ένα αρκετά ικανοποιητικό μέγεθος *recall* το πρόγραμμα τρέχει πολύ γρήγορα -λιγότερο από 1 λεπτό- δεν είναι ικανό να επιστρέψει τους απόλυτα κοντινότερους γείτονες ανεξαρτήτως το πόσο μεγάλο μέγεθος μπορεί να έχουν τα ορίσματα του οποιοδήποτε *instance*.

Ακολουθούν πειράματα μεγαλύτερων δεδομένων με βάση το πως αυξάνεται ο χρόνος εκτέλεσης.

4.1 $datasets$>10000


Execution Time for Dummy Data (Size=30000)
Data Size = 30000
Mean Execution Time (seconds)
210
200
190
180
170
160
0
5
10
15
20
25
30
Number of Threads



Σημαντική παρατήρηση στα παραπάνω πειράματα ήταν η αύξηση του *recall* στο *stiched* στο 94% και του $Filtered$ στο 96

5 Συμπεράσματα

Από τις παραπάνω μετρήσεις φαίνεται ξεκάθαρα πως οι $Filtered$ και $Stitched\ Vamana$ αλγόριθμοι μπορούν να χαρακτηριστούν ως πολύ ανταγωνιστικές επιλογές στη κατηγοριά των $DiskANN$ αλγορίθμων. Αν και λόγο της χρήσης $2dmatrix$ η χρήση της μνήμης δεν γινόταν με καθόλου αποδοτικό τρόπο βλέπουμε πως εντέλει το ρεςαλλ μένει σταθερό ανεξαρτήτως το μέγεθος του *dataset* και ο χρόνος συνεχίζει να παραμένει σχετικά χαμηλός.

Συνεπώς, αν ληφθούν εξάρχής υπόψη οι περιορισμοί της μνήμης και η υλοποίηση χτιστεί γύρω από αυτό, επιλέγωντας μια πιο αποδοτική δομή αντί του $2dmatrix$, το αποτέλεσμα είναι 2 ιδιαίτερα αποδοτικοί αλγόριθμοι που παραμένουν *consistent* ανεξαρτήτως το πλήθος των δεδομένων που έχουν να αντιμετωπίσουν ενώ παράλληλα διατηρούν τον χρόνο υπολογισμού των γράφων και αντίστοιχα του *recall* στα χαμηλά όρια για τα δεδομένα $DISKANN$ αλγορίθμων.

5.1 Διαμοιρασμός καθηκόντων ανάμεσα στα μέλη της ομάδας

- Ισίδωρος Καλαμάρης:$Greedysearch, Robust, FilteredGreedySearch, FilteredRobust$, *some unit tests*
- Κυριακή Καλαμάρη: $Vamana, FilteredVamana$, *script*, *some unit tests*, *Review Paper*

5.2 πηγές

παραδοτέο1
παραδοτέο2