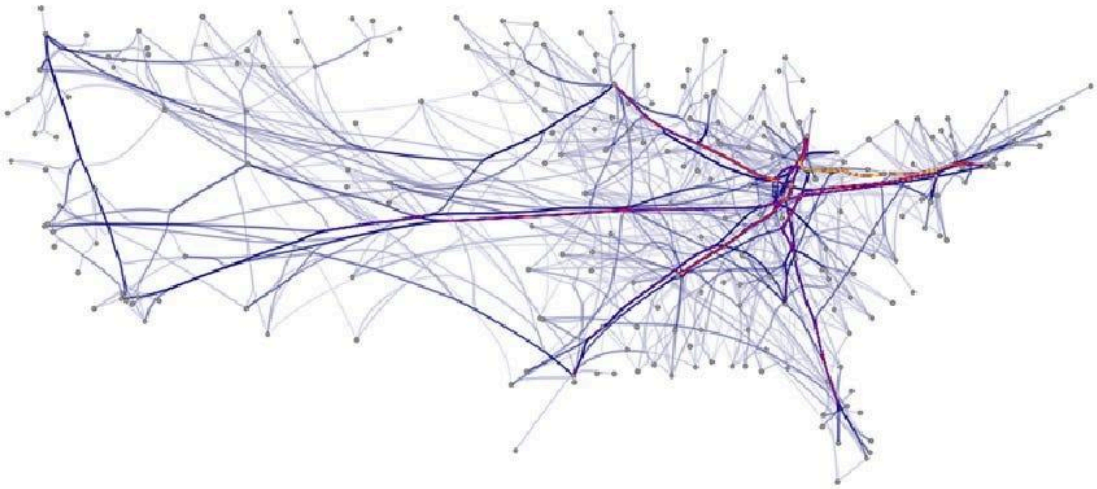
Ανάπτυξη Λογισμικού για Πληροφοριακά Προβλήματα

(3ο Παραδοτέο)

Αργύριος Λαζαρίδης - 1115202100083
Σταύρος Πρέντζας - 1115202100164
Στέφανος Φωτόπουλος - 1115202100209

GitHub repository:
https://github.com/argylaz/K-neighbors-project



Πίνακας Περιεχομένων

Σχεδιαστικές Επιλογές

- Στην υλοποίηση του γράφου χρησιμοποιείται η βιβλιοθήκη STL με την χρήση των δομών unordered set για την αναπαράσταση των κόμβων, vector<vector<>> για την αναπαράσταση του adjacency_list και map για την σύνδεση των κόμβων με τα αντίστοιχα indices
- Για την προσθήκη φίλτρων στο σύστημα φτιάξαμε μια δομή FilterGraph που κληρονομεί από την Graph και προσθέτει ένα mapping απο κόμβους σε φίλτρα.
- Όλο το project έχει υλοποιηθεί με templates για ευελιξία του κώδικα, στην πράξη όμως χρησιμοποιούνται vector<float> ως κόμβοι και float για τα αντίστοιχα φίλτρα.
- Το περισσότερο threading που έχει γίνει (εκτός από ό,τι εχει γινει με openmp), υλοποιήθηκε με στόχο την μέγιστη αξιοποίηση των δυνατοτήτων του μηχανήματος στο οποίο τρέχει το πρόγραμμα. Με την χρήση της std::thread διαβάζουμε το πλήθος των διαθέσιμων threads και μοιράζουμε το φόρτο του task μεταξύ όλων των threads ή μοιράζουμε το κάθε sub-task σε ενα thread και κάνουμε join.
- Η υλοποίηση των δεδομένων αλγορίθμων έχει γίνει στο αρχείο knn.hpp, οι βοηθητικές συναρτήσεις που χρειάστηκαν βρίσκονται όλες στο utils.hpp εκτός απο όσες έχουν να κανουν με read/write από/σε αρχεία οι οποίες βρίσκονται στο fileio.hpp
- Οι γράφοι (και τα αντίστοιχα medoid map των Filtered Graphs) αποθηκεύονται σε αρχεία .bin, τα οποία, όταν ήδη υπάρχουν, διαβάζονται για να εκτελεστούν τα queries.
- Τα ορίσματα εκτέλεσης δίνονται από τον χρήστη με command line arguments (αναλυτικές οδηγίες εκτέλεσης υπάρχουν στο README)
- Για την εκτέλεση των unfiltered queries επιλέξαμε να βάζουμε ως starting points αντί για το αντίστοιχο σημείο του medoid map, το κοντινότερο, για κάθε φίλτρο, σημείο στο query, ώστε η GreedySearch να ελέγξει και αυτές τις “γειτονιές” αν το αντίστοιχο starting point είναι σχετικά κοντά (ιδέα που αντλήσαμε από το piazza).
- Όλα τα δεδομένα είναι στα siftsmall και dummy datasets, το οποιο dummy_queries έχει average specificity ανα query 55.1953%. Ο αριθμός είναι τόσο μεγάλος λόγω των unfiltered queries τα οποία μετρήσαμε ως 100% specificity.
- Τα αποτελέσματα των k - κοντινότερων γειτόνων, το ποσοστό recall και το specificity value κάθε query, γίνονται append σε ένα αρχείο results.txt

Βελτιώσεις 3ου Παραδοτέου

- Παραλληλοποίηση της εκτέλεσης των filtered και unfiltered queries με την χρήση της βιβλιοθήκης openmp.

- Παραλληλοποίηση του υπολογισμού των αθροισμάτων στην συνάρτηση Euclidean Distance με την χρήση της βιβλιοθήκης openmp.

- Δοκιμάστηκε η παραλληλοποίηση της συνάρτησης find_min_Euclidean η οποία ωστόσο δεν επέφερε κάποια βελτίωση (απεναντίας επιβράδυνε τον χρόνο δημιουργίας των γράφων για αυτό και δεν υιοθετήθηκε στο τελικό παραδοτέο).

- Παράλληλη εκτέλεση του FilteredRobustPrune στην συνάρτηση FilteredVamana.

- Η τυχαιότητα στις FindMedoid και rDirectional υλοποιήθηκε τελικά με random indexing και όχι shuffling όλων των στοιχείων καθώς χρειαζόμασταν ένα υποσύνολο αυτών και όχι όλα τους σε τυχαία σειρά, αντίστοιχα στους vamana, το σ(i) υπολογίζεται με shuffle καθώς χρειαζόμασταν όλα τα στοιχεία. Αυτό οδήγησε σε μια βελτίωση του συνολικού χρόνου εκτέλεσης κατά κάποια δευτερόλεπτα καθώς η πολυπλοκότητα του random sampling στην rDirectional για παράδειγμα έγινε O(VlogV) έναντι του V^2 που απαιτούσε το shuffling όλων των στοιχείων κάθε φορα. Είναι επίσης και μια βελτίωση σε επίπεδο μνήμης καθως ξεφορτωθήκαμε το shuffled container.

- Η μέθοδος Medoid βελτιστοποιήθηκε με την χρήση std::thread, χωρίζοντας το σύνολο των κόμβων σε chunks αναλόγως με το πλήθος των threads του μηχανήματος, βρίσκοντας τα min σε κάθε chunk και συνδυάζοντας τα αποτελέσματα στο τέλος.

- Η μέθοδος FindMedoid έχει βελτιστοποιηθεί παρόμοια με την Medoid.

- Η κατασκευή των υπογράφων στον stitched vamana βελτιστοποιήθηκε με χρήση std::thread, ένα thread για κάθε φίλτρο, καθώς είναι όλοι ανεξάρτητοι μεταξύ τους.

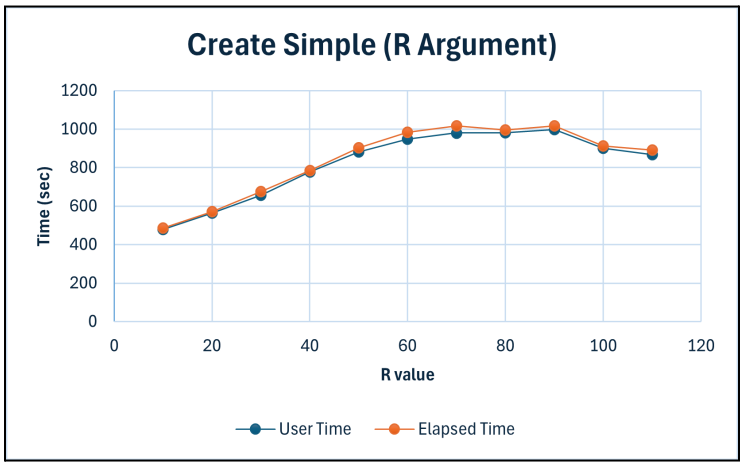
- Τα τμήματα των FilteredVamana και StitchedVamana που εκτελούν Robust Pruning διαδοχικά έχουν υλοποιηθεί με threading με την λογική ότι δεν έχει σημασία η σειρά εκτέλεσης και οι ακμές που πρόκειται να αποκοπούν θα αποκοπούν ούτως η άλλως. Η αλλαγή αυτή δεν επέφερε σημαντικές βελτιώσεις χρονικά στις εκτελέσεις με τα dummy datasets, όμως δεν επηρέασε αρνητικά την αξιοπιστία των αποτελεσμάτων οπότε τα κρατήσαμε.

- Αντικατάσταση της δομής set με Unordered Set για την αποθήκευση των κόμβων του γράφου.

Τελικοί Χρόνοι Υλοποίησης

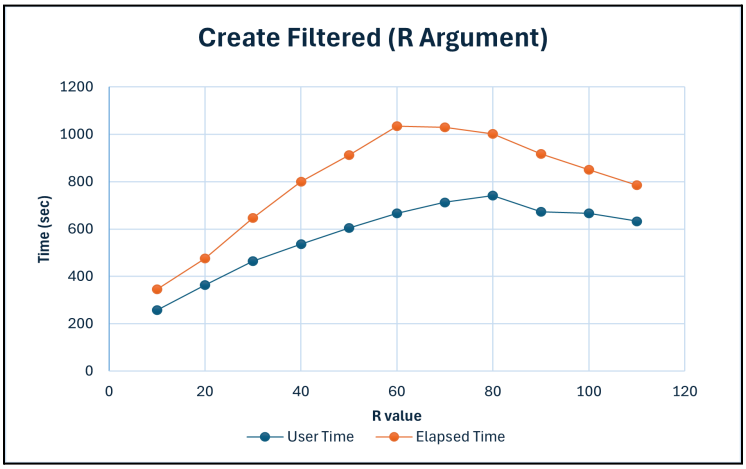
- **Χρόνοι Δημιουργίας Γράφων (αλλάζοντας το όρισμα R)**

Στην Εικόνα 1 παρατηρούμε ότι οι χρόνοι δημιουργίας του ευρετηρίου μεγιστοποιούνται για R από 60 έως 90 ενώ αρχίζουν να μειώνονται για R >= 100. Επίσης παρατηρούμε οτι τα UserTime και Elapsed Time είναι σχεδόν ίδια, το οποίο είναι αναμενόμενο καθως στον απλό vamana δεν υπάρχει παραλληλισμός.



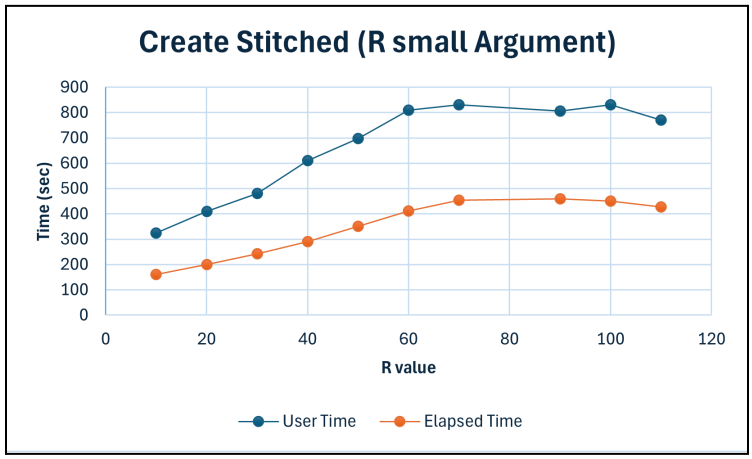
Εικόνα 1

Στην Εικόνα 2, παρόμοια με πριν, παρατηρούμε ότι ο χρόνος εκτέλεσης μεγιστοποιείται για R απο 60-80 και στην συνέχεια αρχίζει να μειώνεται για R > 80. Επίσης παρατηρούμε οτι το Elapsed Time είναι πάντα μεγαλύτερο από το User Time στην εκτέλεση του FilteredVamana.



Εικόνα 2

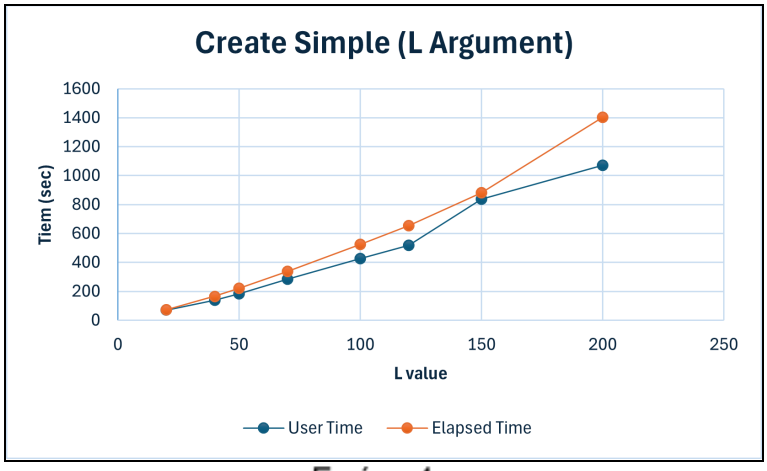
Στην Εικόνα 3, παρόμοια με τις 2 προηγούμενες, παρατηρούμε ότι ο χρόνος εκτέλεσης μεγιστοποιείται για R από 60 έως 90 ενω αρχίζει να μειώνεται για R > 100. Παρατηρούμε επίσης πως το Elapsed Time είναι πάντα μικρότερο από το User Time και περίπου το μισό της αντιστοιχης εκτέλεσης του filtered, το οποίο είναι αναμενόμενο καθώς στον StitchedVamana, τα πάντα εκτός από το merge των υπογράφων εκτελούνται παράλληλα.



Εικόνα 3

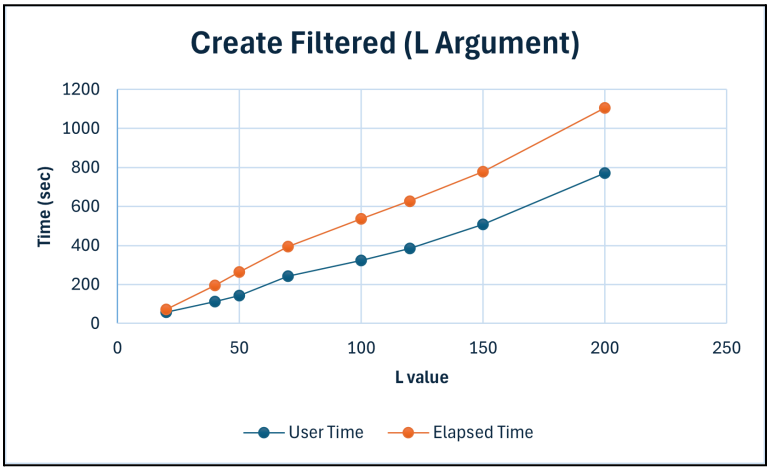
- **Χρόνοι Δημιουργίας Γράφων (αλλάζοντας το όρισμα L)**

Αντίστοιχα με τα προηγούμενα πειραματικά δεδομένα (δημιουργία γράφων με μεταβαλλόμενο όρισμα R), παρατηρούμε στις Εικόνες 4 & 5 ότι ο Elapsed χρόνος εκτέλεσης (τόσο στον Simple οσο και στον Filtered) παραμένει σταθερά μεγαλύτερος από τον User, χωρίς ωστόσο να έχει κάποια σημαντική απόκλιση . Στην δημιουργία των Simple και Filtered γράφων, ο χρόνος φαίνεται να μεταβάλλεται γραμμικά καθώς αυξάνεται η τιμή του ορίσματος L.



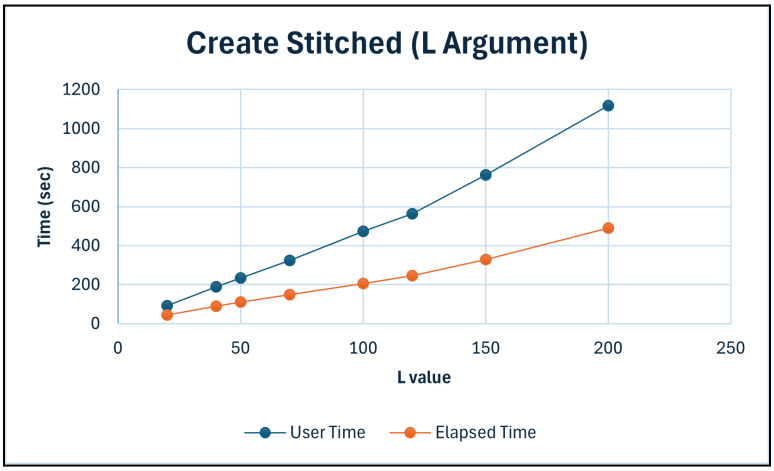
Εικόνα 4

Μεταξύ των χρόνων δημιουργίας του Filtered και του Simple γράφου, παρατηρούμε ότι για μεγαλύτερες τιμές του L, η User Time διαφορά χρόνου του Filtered από τον Simple μεγαλώνει, γεγονός που καθιστά τον Filtered Graph πιο αποδοτικό στο build time όσο μεγαλώνει η τιμή του ορίσματος L.



Εικόνα 5

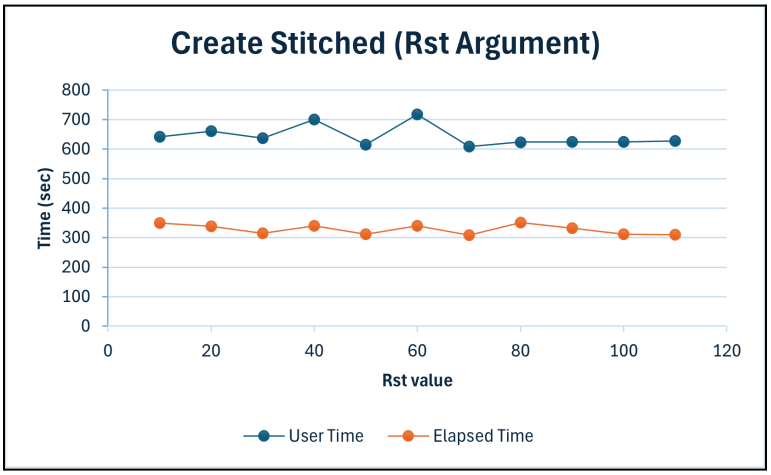
Στην Εικόνα 6 παρατηρούμε ότι ο Elapsed Time είναι αισθητά μικρότερος από τον User Time εκτέλεσης (περίπου μισός). Η παραλληλοποίηση μερών του αλγορίθμου δημιουργίας του Stitched γράφου έδωσε εμφανή αποτελέσματα στους χρόνους εκτέλεσης της δημιουργίας του γράφου, καθιστώντας έτσι τον Stitched Graph των πιο αποδοτικό εκ των τριών, από πλευράς build time. Ο Elapsed Time φαίνεται να αυξάνεται γραμμικά καθώς αυξάνεται η τιμή του L, ενώ παρατηρούμε ότι User Time αυξάνεται με μεγαλύτερο ρυθμό και ασυμπτωτικά η διαφορά τους μεγαλώνει αισθητά .



Εικόνα 6

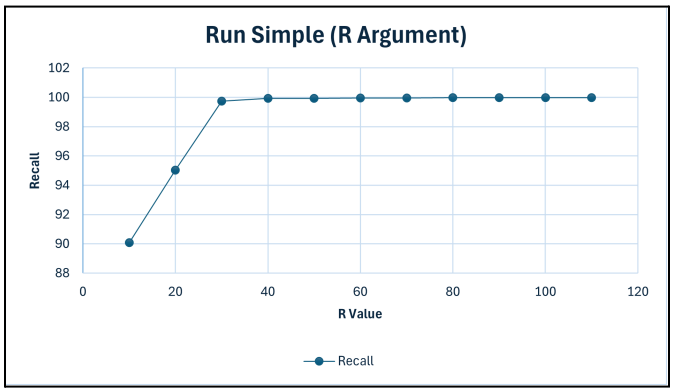
- **Χρόνοι Δημιουργίας Γράφου (αλλάζοντας το όρισμα Rst)**

Στην Εικόνα 7 παρατηρούμε -αντίστοιχα με την εικόνα 6- ότι ο Elapsed Time είναι αισθητά μικρότερος από τον User Time εκτέλεσης (περίπου μισός), το οποίο οφείλεται σε παραλληλοποίηση μερών του αλγορίθμου δημιουργίας του Stitched γράφου. Και οι δύο χρόνοι φαίνονται να μένουν σχετικά σταθεροί (σε ένα εύρος τιμών μεταξύ 600 με 700 sec στον User Time και 300 με 350 sec στον Elapsed Time). Συμπερασματικά, το όρισμα Rst φαίνεται να μην επηρεάζει σημαντικά τον χρόνο δημιουργίας του Stitched Vamana Γράφου καθώς απαιτείται αφαίρεση μικρού ή μηδενικού πλήθους ακμών το οποίο γίνεται σε γραμμικό χρόνο ως προς το πλήθος των φίλτρων (υπογράφων).

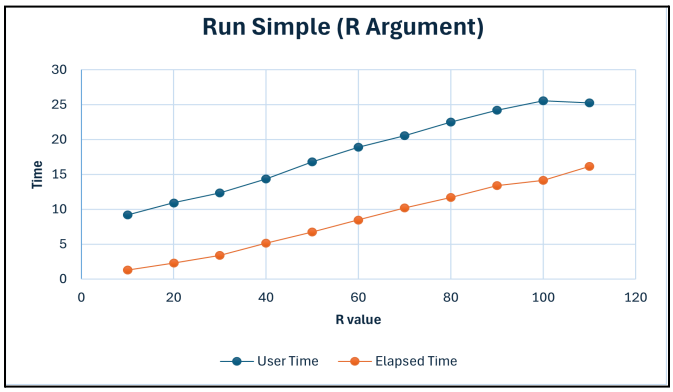


Εικόνα 7

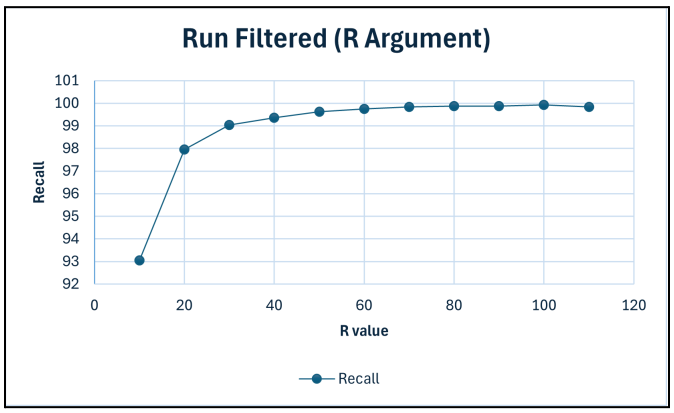
- **Χρόνοι Εκτέλεσης Ερωτημάτων (αλλάζοντας το όρισμα R)**



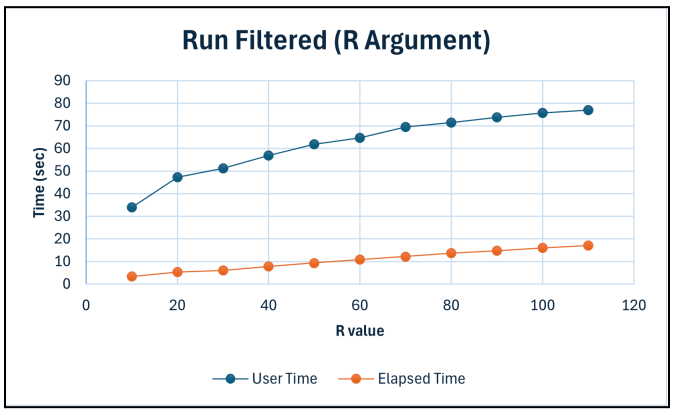
Εικόνα 8



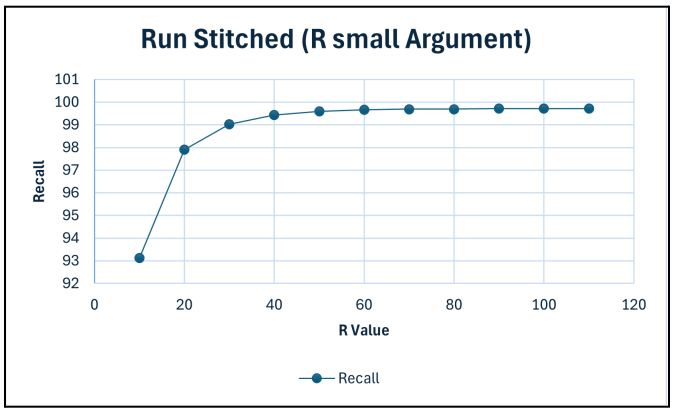
Εικόνα 9



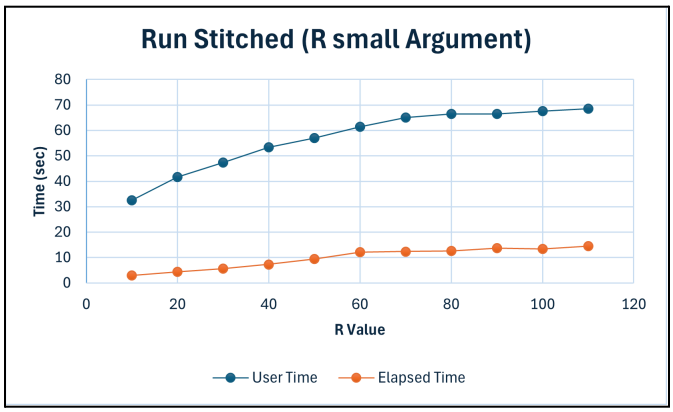
Εικόνα 10



Εικόνα 11



Εικόνα 12

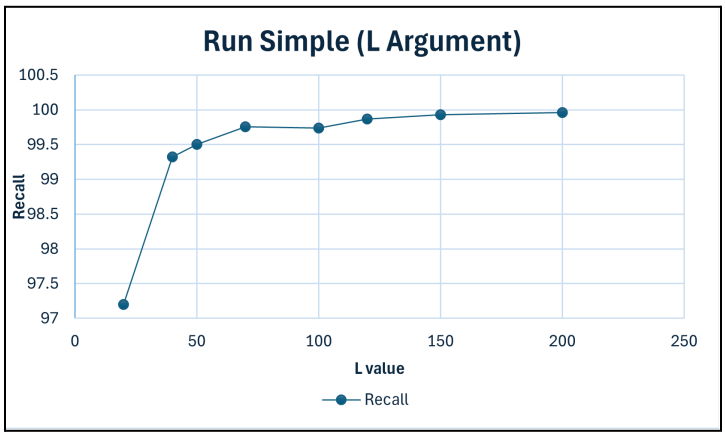


Εικόνα 13

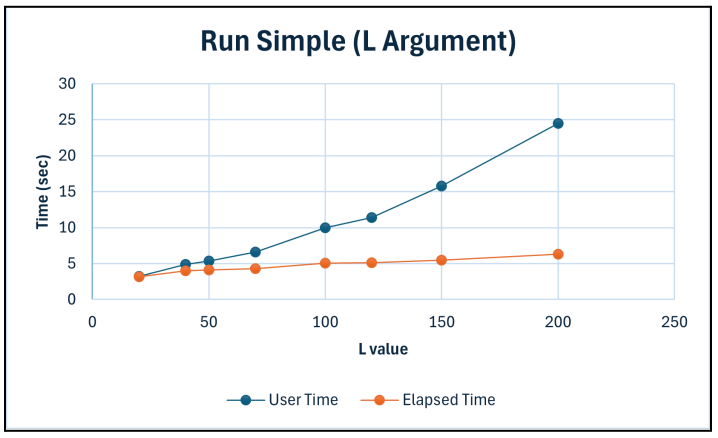
Από τις Εικόνες 9, 11 και 13 παρατηρούμε ότι το συνολικό recall φτάνει μια επιθυμητή ακρίβεια (>95%) για R > 15 ενώ για R > 30 η ακρίβεια είναι μεγαλύτερη του 99% και καθώς το R τείνει στο 100 το ίδιο κάνει και το recall. Αυτο είναι αναμενόμενο καθώς το R αντιπροσωπεύει το μέγιστο πλήθος εξερχόμενων ακμών του κάθε κόμβου στον γράφο.

Από τα διαγράμματα των χρόνων εκτέλεσης των ερωτημάτων για διαφορτικά R (για k = 10, L = 150 και a = 1.2) παρατηρούμε ότι ενώ στον Simple Vamana οι User και Elapsed Time έχουν απόκλιση περίπου 10 δευτερόλεπτα σε όλες τις περιπτώσεις, τα αντίστοιχα διαγράμματα στους Filtered και Stitched η απόκλιση είναι μεγαλύτερη και ανάλογη της τιμής του R, αφού το Elapsed Time δεν αυξάνεται σημαντικά και δεν ξεπερνάει ποτέ τα 20 δευτερόλεπτα ενώ το User Time ξεπερνάει και το ένα λεπτό σε κάποιες περιπτώσεις.

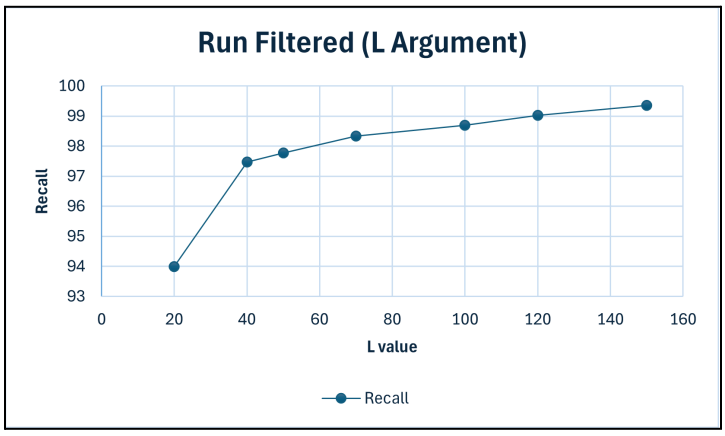
- **Χρόνοι Εκτέλεσης Ερωτημάτων (αλλάζοντας το όρισμα L)**



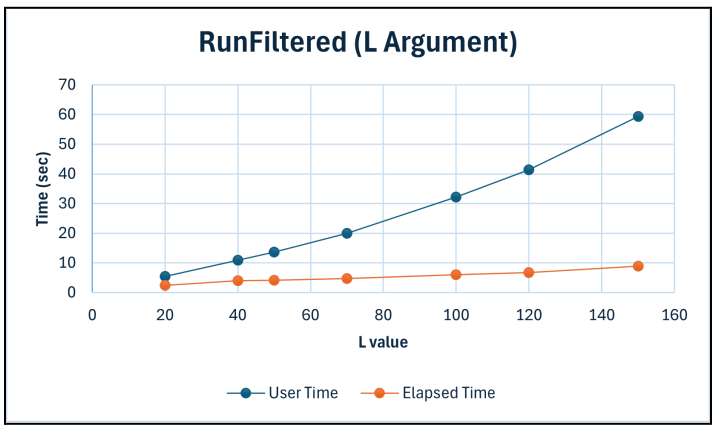
Εικόνα 16



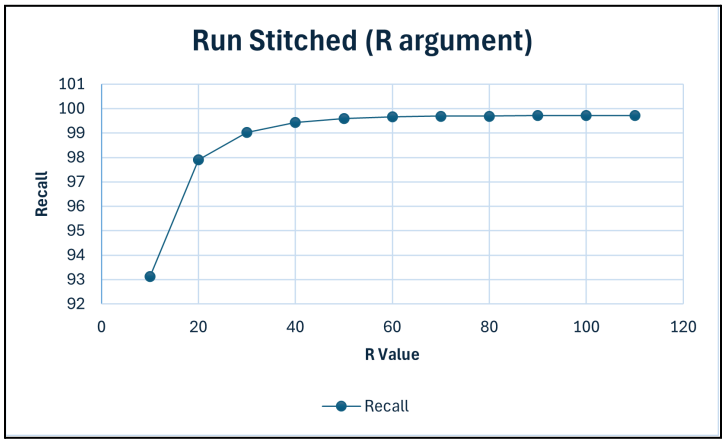
Εικόνα 17



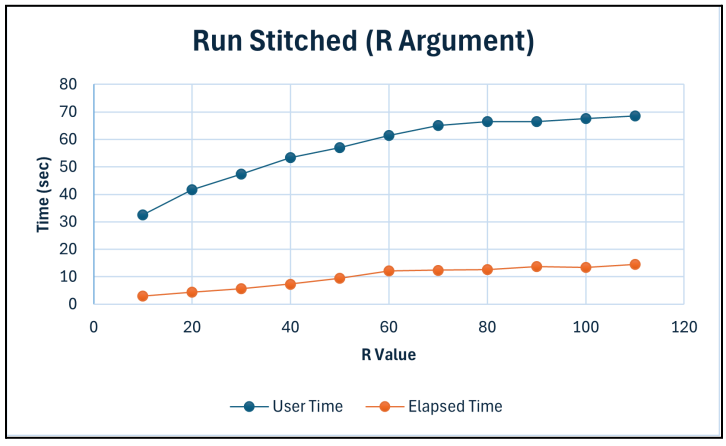
Εικόνα 18



Εικόνα 19



Εικόνα 20

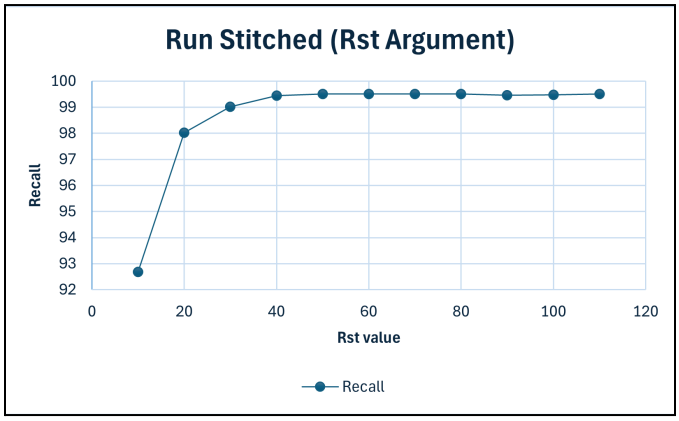


Εικόνα 21

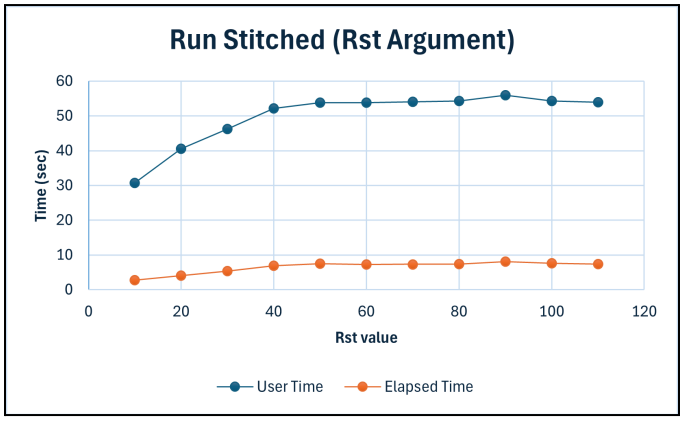
Από τις Εικόνες 17, 19 και 21 (καθώς και από τις εκτελέσεις για διαφορετικές τιμές του ορίσματος R) εξάγουμε το συμπέρασμα ότι ο Simple γράφος σημειώνει τους χαμηλότερους χρόνους εκτέλεσης των ερωτημάτων σε Elapsed και User Time. Καθώς αυξάνεται η τιμή του ορίσματος L, οι μεταβολές του του Elapsed Time είναι μικρές (συγκριτικά με αυτές του User Time), γεγονός αναμενόμενο, καθότι η εκτέλεση των ερωτημάτων γίνεται παράλληλα (με την χρήση της βιβλιοθήκη openmp).

Τα ποσοστά recall και στις τρεις υλοποιήσεις είναι ικανοποιητικά (>93%), διαπιστώνοντας όπως και στα προηγούμενα δεδομένα μελέτης (για μεταβαλλόμενη τιμή του ορίσματος R) ότι για μεγαλύτερες τιμές του L, παίρνουμε όλο και πιο ικανοποιητικά ποσοστά recall (το οποίο τείνει ασυμπτωτικά στο 100%).

- **Χρόνοι Εκτέλεσης Ερωτημάτων (αλλάζοντας το όρισμα Rst)**



Εικόνα 22



Εικόνα 23

Ικανοποιητικούς επίσης χρόνους (User Time & Elapsed Time) και ποσοστά Recall παρατηρούμε στις Εικόνες 22 & 23, από τα αποτελέσματα της εκτέλεσης των ερωτημάτων με μεταβαλλόμενο το όρισμα Rst. Η παραλληλοποίηση της εκτέλεσης των ερωτημάτων έχει ως αποτέλεσμα, για ακόμα μια φορά, ο Elapsed Time να είναι σημαντικά μικρότερος από τον User Time (εώς και 47 sec). Τέλος, και οι χρόνοι εκτέλεσης και το ποσοστό Recall παραμένουν σταθερά για τις τιμές του Rst μεγαλύτερες του R small (γεγονός αναμενόμενο, δεδομένου ότι τα σημεία που δίνονται έχουν ένα ακριβώς ένα φίλτρο, συνεπώς έπειτα από τον συνδυασμό των υπογράφων, δεν προκύπτει κόμβος με ακμές περισσότερες από R small ώστε ο FilteredRobustPrune με Rst > R small να τις αφαιρέσει, άρα και δημιουργούνται παρόμοιοι γράφοι).

Πίνακες Δεδομένων Διαγραμμάτων

Όπως φαίνεται και στους παρακάτω πίνακες, τα υπόλοιπα ορίσματα πέραν των L, R και Rst (δηλαδή τα k, a) κατά την εκτέλεση του προγράμματος παίρνουν λογικές τιμές, οι οποίες συμβάλλουν στην ορθή εκτέλεση και την σωστή αποτύπωση των αποτελεσμάτων και των συμπερασμάτων της παρούσας υλοποίησης. Κατά την εκτέλεση ερωτημάτων και την δημιουργία των γράφων (Simple, Filtered και Stitched) για μεταβαλλόμενες τιμές του L, αναγκαστικά η παράμετρος L λειτουργεί σαν bottleneck για την παράμετρο k. Συνεπώς, για τις εκτελέσεις με $L \leq 100$ θέσαμε L=k, ενώ για τις εκτελέσεις με $L > 100$ η τιμή k παρέμεινε 100. Το όρισμα a, έμεινε παντού ίσο με 1.2, η τιμή του ορίσματος Rstitched (όπου χρησιμοποιήθηκε) ήταν ίση με 96. Τέλος στα πειράματα για την μεταβολή του ορίσματος L, το R επιλέχθηκε να είναι ίσο με 40 ή 42. Τα δεδομένα των διάφορων εκτελέσεων του προγράμματος (αναλυτικοί χρόνοι, ορίσματα k, L, recall) εκτυπώθηκαν σε μορφή csv (μέσω εκτελέσεων που έγιναν σε bash scripting), και στην συνέχεια επεξεργάστηκαν με Excel για την δημιουργία των αντίστοιχων διαγραμμάτων.
Το περιβάλλον εκτέλεσης του προγράμματος είναι WSL 2.
Τα στοιχεία του μηχανήματος εκτέλεσης των προγραμμάτων είναι

- CPU: Ryzen 7 5800h (8 cores 16 threads)
- RAM: 16GB
- Το μηχάνημα επίσης διαθέτει γρήγορο SSD και GPU RTX 3060 αλλά δεν έχει σημασία στο συγκεκριμένο πρόγραμμα.

Παρατίθενται τα αναλυτικά δεδομένα εκτέλεσης των προγραμμάτων:

- ***Run Simple R***

K	L	R	Rstitched	a	Recall	UserTime	ElapsedTime
100	150	10	-1	1.2	90.08	9.171875	1.2674
100	150	20	-1	1.2	95.04	10.921875	2.287
100	150	30	-1	1.2	99.75	12.34375	3.40356
100	150	40	-1	1.2	99.94	14.34375	5.13701
100	150	50	-1	1.2	99.93	16.8125	6.75322
100	150	60	-1	1.2	99.96	18.90625	8.47043
100	150	70	-1	1.2	99.96	20.5625	10.2185
100	150	80	-1	1.2	99.97	22.515625	11.6751
100	150	90	-1	1.2	99.97	24.1875	13.4
100	150	100	-1	1.2	99.98	25.546875	14.1692
100	150	110	-1	1.2	99.97	25.234375	16.1746

- ***Run Filtered R***

K	L	R	Rstitched	a	Recall	UserTime	ElapsedTime
100	150	10	-1	1.2	93.05	33.96875	3.41637
100	150	20	-1	1.2	97.96	47.296875	5.2712
100	150	30	-1	1.2	99.03	51.15625	6.08536
100	150	40	-1	1.2	99.36	56.9375	7.89233
100	150	50	-1	1.2	99.62	61.828125	9.32191
100	150	60	-1	1.2	99.75	64.703125	10.7796
100	150	70	-1	1.2	99.84	69.46875	12.1668
100	150	80	-1	1.2	99.88	71.421875	13.641
100	150	90	-1	1.2	99.87	73.8125	14.7457
100	150	100	-1	1.2	99.92	75.75	15.9589
100	150	110	-1	1.2	99.84	76.875	16.9814

- ***Run Stitched R***

K	L	R	Rstitched	a	Recall	UserTime	ElapsedTime
100	150	10	96	1.2	93.12	32.46875	2.96279
100	150	20	96	1.2	97.91	41.625	4.39235
100	150	30	96	1.2	99.02	47.34375	5.58827
100	150	40	96	1.2	99.43	53.421875	7.29815
100	150	50	96	1.2	99.6	56.984375	9.42432
100	150	60	96	1.2	99.66	61.421875	12.112
100	150	70	96	1.2	99.69	64.96875	12.3545
100	150	80	96	1.2	99.69	66.46875	12.5717
100	150	90	96	1.2	99.72	66.453125	13.7087
100	150	100	96	1.2	99.71	67.578125	13.3846
100	150	110	96	1.2	99.71	68.53125	14.5599

- ***Create Simple R***

K	L	R	Rstitched	a	UserTime	ElapsedTime
100	150	10	-1	1.2	479.578125	486.626
100	150	20	-1	1.2	563.609375	570.995
100	150	30	-1	1.2	655.828125	674.439
100	150	40	-1	1.2	776.4375	784.79
100	150	50	-1	1.2	882.46875	903.087
100	150	60	-1	1.2	948.765625	984.39
100	150	70	-1	1.2	979.75	1016.97
100	150	80	-1	1.2	981.53125	996.195
100	150	90	-1	1.2	997.234375	1015.8
100	150	100	-1	1.2	900.203125	912.601
100	150	110	-1	1.2	868.171875	891.579

- ***Create Filtered R***

K	L	R	Rstitched	a	UserTime	ElapsedTime
100	150	10	-1	1.2	258.109375	344.71
100	150	20	-1	1.2	363.9375	475.743
100	150	30	-1	1.2	464.609375	646.18
100	150	40	-1	1.2	535.78125	800.549
100	150	50	-1	1.2	603.671875	911.643
100	150	60	-1	1.2	665.625	1034.96
100	150	70	-1	1.2	712.640625	1029.84
100	150	80	-1	1.2	741.78125	1001.99
100	150	90	-1	1.2	672.9375	916.548
100	150	100	-1	1.2	665.921875	850.051
100	150	110	-1	1.2	632.8125	785.643

- ***Create Stitched R***

K	L	R	Rstitched	a	UserTime	ElapsedTime
100	150	10	96	1.2	324.25	159.911
100	150	20	96	1.2	408.796875	200.482
100	150	30	96	1.2	480.21875	241.755
100	150	40	96	1.2	610.75	290.93
100	150	50	96	1.2	697.4375	351.709
100	150	60	96	1.2	809.109375	411.35
100	150	70	96	1.2	831.5625	453.564
100	150	90	96	1.2	806.171875	459.168
100	150	100	96	1.2	831.171875	449.637
100	150	110	96	1.2	770.140625	427.611

- ***Run Simple L***

K	L	R	Rstitched	a	Recall	UserTime	ElapsedTime
20	20	42	-1	1.2	97.2	3.203125	3.16472
40	40	42	-1	1.2	99.325	4.859375	3.97192
50	50	42	-1	1.2	99.5	5.359375	4.11547
70	70	42	-1	1.2	99.7571	6.609375	4.31386
100	100	42	-1	1.2	99.74	9.96875	5.04636
100	120	42	-1	1.2	99.87	11.375	5.13941
100	150	42	-1	1.2	99.93	15.8125	5.49389
100	200	42	-1	1.2	99.96	24.46875	6.28876

- ***Run Filtered L***

K	L	R	Rstitched	a	Recall	UserTime	ElapsedTime
20	20	40	-1	1.2	94	5.4375	2.47239
40	40	40	-1	1.2	97.475	10.84375	3.95462
50	50	40	-1	1.2	97.78	13.578125	4.13536
70	70	40	-1	1.2	98.3286	19.90625	4.66169
100	100	40	-1	1.2	98.7	32.1875	5.99273
100	120	40	-1	1.2	99.02	41.3125	6.77343
100	150	40	-1	1.2	99.36	59.34375	8.87636

- ***Run Stitched L***

K	L	R	Rstitched	a	Recall	UserTime	ElapsedTime
20	20	42	96	1.2	93.4	5.59375	2.88576
40	40	42	96	1.2	97.475	10.453125	3.8354
50	50	42	96	1.2	97.7	12.953125	3.88062
70	70	42	96	1.2	98.2	19.390625	4.56201
100	100	42	96	1.2	98.79	31.703125	6.05009
100	120	42	96	1.2	99.16	40.9375	6.37774
100	150	42	96	1.2	99.48	57.671875	7.82564
100	200	42	96	1.2	99.79	83.484375	9.5845

- ***Create Simple L***

K	L	R	Rstitched	a	UserTime	ElapsedTime
20	20	42	-1	1.2	69.5625	73.5322
40	40	42	-1	1.2	140.34375	165.728
50	50	42	-1	1.2	182.40625	220.787
70	70	42	-1	1.2	284.359375	338.414
100	100	42	-1	1.2	426.78125	523.088
100	120	42	-1	1.2	517.765625	654.35
100	150	42	-1	1.2	837.640625	881.493
100	200	42	-1	1.2	1071.296875	1401.78

- ***Create Filtered L***

K	L	R	Rstitched	a	UserTime	ElapsedTime
20	20	40	-1	1.2	58.578125	73.3115
40	40	40	-1	1.2	113.21875	195.697
50	50	40	-1	1.2	143.546875	265.043
70	70	40	-1	1.2	242.515625	393.909
100	100	40	-1	1.2	322.75	537.495
100	120	40	-1	1.2	385.859375	627.858
100	150	40	-1	1.2	507.828125	779.403
100	200	40	-1	1.2	770.78125	1104.71

- ***Create Stitched L***

K	L	R	Rstitched	a	UserTime	ElapsedTime
20	20	42	96	1.2	92.34375	44.2499
40	40	42	96	1.2	187.9375	89.5652
50	50	42	96	1.2	233.15625	110.869
70	70	42	96	1.2	324.859375	147.536
100	100	42	96	1.2	472.328125	205.775
100	120	42	96	1.2	563.5625	245.331
100	150	42	96	1.2	762.203125	328.429
100	200	42	96	1.2	1118.875	490.736

- ***Create Stitched Rst***

K	L	R	Rstitched	a	UserTime	ElapsedTime
100	150	42	10	1.2	641.109375	349.174
100	150	42	20	1.2	660.640625	338.039
100	150	42	30	1.2	637.125	314.819
100	150	42	40	1.2	700.1875	340.407
100	150	42	50	1.2	615.34375	312.169
100	150	42	60	1.2	717.359375	339.779
100	150	42	70	1.2	608.875	308.857
100	150	42	80	1.2	623.578125	351.613
100	150	42	90	1.2	624.71875	332.792
100	150	42	100	1.2	623.625	312.124
100	150	42	110	1.2	627.3125	310.3871

- ***Run Stitched Rst***

K	L	R	Rstitched	a	Recall	UserTime	ElapsedTime
100	150	42	10	1.2	92.68	30.671875	2.78189
100	150	42	20	1.2	98.02	40.5	4.029
100	150	42	30	1.2	99.02	46.265625	5.36071
100	150	42	40	1.2	99.44	52.15625	6.952
100	150	42	50	1.2	99.5	53.8125	7.46871
100	150	42	60	1.2	99.5	53.78125	7.24322
100	150	42	70	1.2	99.5	54.109375	7.31006
100	150	42	80	1.2	99.5	54.296875	7.34931
100	150	42	90	1.2	99.46	55.984375	8.07712
100	150	42	100	1.2	99.48	54.296875	7.57576
100	150	42	110	1.2	99.5	53.984375	7.3551

Συμπεράσματα

Από τα διαγράμματα και τις παρατηρήσεις των χρόνων δημιουργίας ευρετηρίου, και των ερωτημάτων, για τους 3 αλγόριθμους με διαφορετικές τιμες του R συμπεραίνουμε ότι οι χειρότερες τιμές από άποψη απόδοσης χρόνου είναι αυτές από 60 έως 80. Μεγαλύτερες τιμές του R αυξάνουν τον Elapsed Time των run και create (και φυσικά και τον User) χωρίς μεγάλες αυξήσεις στο recall (ειδικά στον απλό vamana) καθώς αυτό είναι ήδη αρκετά μεγάλο (για R > 20 Recall >95%). Οπότε αν δεν απαιτείται απόλυτη ακρίβεια και υπάρχει ανοχή σε ελάχιστα σφάλματα, δεν απαιτούνται και μεγάλες τιμές για το R. Οι τιμές 20-40 προσφέρουν ένα ικανοποιητικά καλό trade-off χρόνου και ακρίβειας. Αυτό συμβαίνει επειδή το R αντιπροσωπεύει τον πραγματικό αριθμό εξερχόμενων ακμών για κάθε κόμβο και για το συγκεκριμένο dataset και groundtruth (με k = 100) περισσότερες από 20-40 ακμές επιβαρύνουν πολύ την GreedySearch χωρίς να χρειάζεται καθώς τα αποτελέσματα μπορούν να βρεθούν και με λιγότερες ακμές, σε ικανοποιητικό χρόνο και με ικανοποιητική ακρίβεια.

Από τις εκτελέσεις ερωτημάτων με διαφορετικές τιμές του ορίσματος Rstitched εξάγουμε το συμπέρασμα ότι σε περιπτώσεις που η τιμή του είναι μικρότερη από αυτήν του Rsmall έχουμε μειωμένο Recall. Αυτό συμβαίνει επειδή απο τις R ακμές που έχει κάθε κόμβος στους υπογράφους, στο τέλος κρατάμε μόνο τις Rstitched όταν γίνεται merge των γράφων, με αποτέλεσμα να δυσκολεύεται περισσότερο ο GreedySearch να βρει όλους τους γείτονες.

Από όλες τις εκτελέσεις κατασκευής ευρετηρίου με τις διάφορες τιμες για τις παραμέτρους συμπεραίνουμε ότι η προσπάθεια βελτιστοποίησης του Filtered Vamana με χρήση openmp δεν απέδωσε όπως αναμέναμε καθως το Estimated Time είναι ακόμη μεγαλύτερο του UserTime. Αυτό συμβαίνει μάλλον επειδη το μεγαλύτερο μέρος του αλγορίθμου τρέχει σειριακά και το μέρος που τρέχει παράλληλα περνάει αρκετό χρόνο περιμένοντας για πρόσβαση σε κοινόχρηστες δομές (με mutex) και άλλες I/O λειτουργίες που δεν έχουν να κάνουν με το User Time. Αντιθέτως, στον Stitched Vamana φαίνεται πως ο παραλληλισμός πέτυχε κρίνοντας από τους χρόνους, γεγονός που τον καθιστά τον πιο αποδοτικό εκ των τριών όσον αφορά τον χρόνο δημιουργίας του γράφου.

Αναφορές

1. Suhas Jayaram Subramanya, Devvrit, Rohan Kadekodi, Ravishankar Krishaswamy, and Harsha Vardhan Simhadri. 2019. DiskANN: fast accurate billion-point nearest neighbor search on a single node. Proceedings of the 33rd International Conference on Neural Information Processing Systems. Curran Associates Inc., Red Hook, NY, USA, Article 1233, 13766–13776. URL https://dl.acm.org/doi/abs/10.5555/3454287.3455520

2. Siddharth Gollapudi, Neel Karia, Varun Sivashankar, Ravishankar Krishnaswamy, Nikit Begwani, Swapnil Raz, Yiyong Lin, Yin Zhang, Neelam Mahapatro, Premkumar Srinivasan, Amit Singh, and Harsha Vardhan Simhadri. 2023. Filtered-DiskANN: Graph Algorithms for Approximate Nearest Neighbor Search with Filters. In Proceedings of the ACM Web Conference 2023 (WWW '23). Association for Computing Machinery, New York, NY, USA, 3406–3416. Filtered-DiskANN: Graph Algorithms for Approximate Nearest Neighbor Search with Filters | Proceedings of the ACM Web Conference 2023