Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
Τμήμα Πληροφορικής και Τηλεπικοινωνιών

Ανάπτυξη λογισμικού για Πληροφοριακά συστήματα
Τελική Αναφορά

Ονοματεπώνυμο: Φραγκίσκος Φαρμάκης,Γιώργος Τσόμης Αριθμός Μητρώου: 1115202100201-198

Abstract

Αυτη η εργασία αφορά στην ανάπτυξη λογισμικού για την κατασκευή δομής μέσω της οποίας μπορεί να υπάρξει αποδοτική απάντηση για την εύρεση $k - nearest$ γειτόνων ενος δεδομένου σημείου.Στην πρώτη φάση της εργασίας η δομή μπορούσε να εξηπηρετήσει ερωτήματα βασισμένα μόνο στις συντεταγμένες του κάθε σημείου και σαν μετρική χρησιμοποιήθηκε το τετράγωνο της ευκλείδιας απόστασης των σημείων.Στο δέυτερο κομμάτι της εργασίας προστέθηκαν στα κρητίρια ετικέτες-ταμπέλες ανά σημείο η οποίες λειτουργούν σαν φίλτρα.Δηλαδή ακόμα και εάν δύο σημεία είχαν πολύ μικρή απόσταση μεταξύ τους μπορούσαν να μην θεωρούνται γείτονες καθώς τα φίλτρα τους δεν ταιριάζουν.Στο τρίτο κομμάτι της εργασίας χρησιμοποιήθηκαν διάφορες βελτιστοποιήσεις στους αλγορίθμους του δεύτερου κομματιού για την ταχύτερη σύγκλιση της δομής,αλλα και για την καλύτερη προσέγγιση των αποτελεσμάτων του προβλήματος των $k - nearest$ γειτόνων.Παράλληλα παρατίθενται ανάλυση των αποτελεσμάτων των αλγορίθμων σύμφωνα με τις διάφορες τιμές των παραμέτρων που μπορούν να προκύψουν.

---

1 Εἰσαγωγή

Η αναζήτηση του εγγύτερου γείτονα ($nearest\ neighbor\ search - NNS$) αποτελεί ένα πρόβλημα βελτιστοποίησης που αφορά την εύρεση του σημείου σε ένα δεδομένο σύνολο που βρίσκεται πιο κοντά (ή μοιάζει περισσότερο) σε ένα συγκεκριμένο σημείο αναφοράς. Η εγγύτητα συνήθως καθορίζεται μέσω μιας μετρικής ανομοιότητας ($dissimilarity$), όπου μεγαλύτερες τιμές δηλώνουν λιγότερη ομοιότητα μεταξύ αντικειμένων.

Το πρόβλημα του εγγύτερου γείτονα μπορεί να διατυπωθεί ως εξής: Δεδομένου ενός συνόλου S σημείων σε έναν χώρο M και ενός σημείου ερώτησης $q \in M$, ζητείται το σημείο στο S που βρίσκεται πλησιέστερα στο q. Συνήθως, ο χώρος M είναι μετρικός, και η έννοια της ανομοιότητας ορίζεται μέσω μιας μετρικής απόστασης, η οποία είναι συμμετρική και ικανοποιεί την τριγωνική ανισότητα. Επιπλέον, ο χώρος μπορεί να είναι d-διάστατος, όπου η ανομοιότητα υπολογίζεται συχνά μέσω της Ευκλείδειας απόστασης μεταξύ διανυσμάτων.

Μια προσεγγιστική λύση του προβλήματος αυτού για μεγάλα δεδομένα είναι η κατασκευή δομής $Vamana$, η οποία χρησιμοποιεί για την κατασκευή της άλλους 3 αλγορίθμους:

1. $Greedy-Search$,άπληστος αλγόριθμος ο οποίος έχει σαν είσοδο τις συντεταγμένες(ή και φίλτρα) του $query$ και τις παραμέτρους k και L.Ώς αποτέλεσμα επιστρέφει του $k-nearest$ του $query$ ύπο μια έννοια τυχαία(για την ακρίβεια η ποιότητα των αποτελεσμάτων βασίζετε σε ένα πολύ μεγάλο βαθμό στο σημείο εκκίνησης του αλγορίθμου).Ωστόσο αυτη η άπληστη μέθοδος δεν έχει ίδια απόδοση με την δομή,ούτε φέρνει τα επιθημητά αποτελέσματα.Σημαντική παρατήρηση είναι πως ο $Greedy-Search$ επιστρέφει καλά αποτελέσματα σε γράφου αραιής γειτονίας,ωστόσο τέτοιοι γράφοι είναι αρκετά μεγάλοι.Παρόλα αυτα είμαστε ένα βήμα πιο κόντα στην λύση του προβλήματος.

2. $Robust-Prune$, Όπως αναφέρθηκε προηγουμένως, οι γράφοι αραιής γειτονίας αποτελούν κατάλληλους υποψήφιους για τη διαδικασία αναζήτησης $GreedySearch$. Ωστόσο, ενδέχεται η διάμετρός τους να είναι αρκετά μεγάλη, γεγονός που μπορεί να δυσχεράνει την αναζήτηση. Σε τέτοιους γράφους, η αναζήτηση θα απαιτούσε πλήθος διαδοχικών αναγνώσεων για τον εντοπισμό των γειτόνων των κορυφών που επισκέπτεται ο αλγόριθμος κατά τη διάρκεια του μονοπατιού αναζήτησης.

   Για να περιοριστεί αυτό το πρόβλημα, προτείνεται η μείωση της απόστασης στην ερώτηση σε κάθε κορυφή κατά μήκος του μονοπατιού αναζήτησης, πολλαπλασιαστικά, αντί για την απλή μείωση που χρησιμοποιείται στους γράφους αραιής γειτονίας. Θεωρήστε έναν κατευθυνόμενο γράφο, όπου οι εξερχόμενοι γείτονες κάθε σημείου ορίζονται από τη συνάρτηση $RobustPrune()$. Σημειώνεται ότι αν οι εξερχόμενοι γείτονες καθορίζονται από τη $RobustPrune()$, τότε η διαδικασία $GreedySearch()$, ξεκινώντας από οποιοδήποτε σημείο, θα συγκλίνει στον στόχο σε λογαριθμικό αριθμό βημάτων, υπό την προϋπόθεση ότι ο γράφος πληροί συγκεκριμένες ιδιότητες.

   Παρά τα παραπάνω, η κατασκευή του ευρετηρίου μπορεί να είναι χρονοβόρα, με πολυπλοκότητα τάξης $O(N^2)$. Για να μειωθεί αυτός ο χρόνος, ο αλγόριθμος $Vamana$ εκτελεί τη διαδικασία $RobustPrune()$ μόνο σε έναν επιλεγμένο υποσύνολο κορυφών, το οποίο περιέχει σημαντικά λιγότερα από N σημεία. Με αυτόν τον τρόπο, ο χρόνος κατασκευής του ευρετηρίου βελτιώνεται αισθητά.

3. $FindMedoid$, η αναζήτηση του σημείου το οποίο έχει κατά μέσω όρο την χαμηλότερη απόσταση απο όλα τα άλλα σημεία.Στην περίπτωση των φίλτρων ο αλγόριθμος αυτός διαλέγει πιθανοκρατικά ένα σημείο έτσι ώστε να αποφευχθεί η έναρξη $query$ διαφορετικών φίλτρων απο το ίδιο σημείο.

Παρατηρήστε οτι αναφερόμαστε σε ποιότητα αποτελεσμάτων,με ποιο γνόμονα μετράμε αυτη τη απόδοση;Χρησιμοποιούμε την έννοια του $Recall$ η οποία αναλύετε ως εξής: Έστω X το σύνολο των σημείων που επιστρέφει η δομή μας και έστω T το σύνολο των σημείων τότε το $Recall$ ορίζεται

$$Recall = \frac{T \cup X}{X}$$

Σε αυτο το σημείο είναι καλή στιγμή να συζητήσουμε τις παραμέτρους που χρησιμοποιούντε στην κατασκευή του $index$,καθώς ο τρόπος που η κάθε μια επηρεάζει το $Recall$ και την ταχύτητα σύγκλισης είναι απο τα βασικά αντικείμενα αυτής της εργασίας.

1. a, η μεταβλητή a ορίζετε $a \geq 1$ χαρακτηρίζει το πόσο ανοχή έχει ο αλγόριθμος $Robust-Prune$,δηλαδή όσο πιο κοντά στο 1 βρίσκεται τόσο λιγότερη ανοχή έχουμε στο κλάδεμα.

2. L,η μεταβλήτη $L \geq k$ χρησιμοποιείτε στον αριθμό απο σημεία που επιτρέπεται η $Greedy-Search$ να εξετάσει πριν φιλτράρει,το σημαντικό εδω είναι οτι όσο πιο μεγάλο είναι το L θα έπρεπε το $Recall$ αλλα και ο χρόνος σύγλισης να αυξάνετε.

3. R,τέλος το R ορίζει την τάξη κάθε κόμβου μόλις η κατασκευή της δομής έχει ολοκληρωθεί.Σαν συμπεριφορά θα περιμέναμε τα ίδια με το L.

Θα μηλίσουμε πρώτα για υλοποίηση και βελτιστωποιήσεις και μετά θα εξετάσουμε πειραματικά, εάν οι υποθέσεις που κάνουμε αντικατοπτρίζονται και στην πράξη.

2 Υλοποίηση-Βελτιστοποιήσεις

Η υλοποίηση χωρίζετε στα εξής αρχεία:

1. $functions.cpp$, $functions.h$,το αρχείο της βιβλιοθήκης που περιέχει όλες τις συναρτήσεις και αλγορίθμους που προαναφέραμε,αλλα και επιπλέον βοηθητικές συναρτήσεις

2. $vamana.cpp$,το αρχείο που περιέχει την $main$ συνάρτηση που δημιουργεί την δομή και εκτελεί τα ερωτήματα επιστρέφοντας το $Recall$ και για $filtered$ αλλα και για $unfiltered$ ερωτήματα.Τα ερωτήματα που εκτελούνται είναι τυχαία επιλεγμένα και σαν μεγεθός είναι 100.Για κάθε ένα απο αυτα έχει υπολογιστεί με $brute\ force$ το $groundtruth$ τους.

3. $testvamana.cpp$,το αρχείο που περιέχει όλα τα $unit\ tests$ συναρτήσεων του $functions.cpp$ και χρησιμοποιούντε για έλεγχο κώδικα μέσω $github\ actions$.

2.1 R-τυχαίος γράφος

Μια απο τις προτεινόμενες βελτιστοποιήσεις,συνοπτικά οι φιλτραρισμένες δομές αρχικοποιούντε ως κενοί γράφοι και έπειτα προσθέτουμε ακμές.Η βελτιστοποίηση ξεκινάει με ήδη υπάρχουσες ακμές κάτι το οποίο περιμένουμε να αυξήσει το $recall$ εφόσον τα νέα μονοπάτια που δημιουργούντε δίναντε να βοηθήσουν στην έυρεση καλών γειτόνων.Κυριώς στην $stitched\ vamana$ κάτι τέτοιο περιμένουμε οτι θα αυξήσει σίγουρα το $recall$ καθώς πριν απο αυτο θα είναι ενα πλήθως απο ανεξάρτητους γράφους.Θα δούμε τις επιδράσεις του στο $Recall$ και πειραματικά παρακάτω.

2.2 Αποθήκευση δομής σε αρχείο

Ο τίτλος είναι αρκετά περιεκτικός,αντί να χρειάζετε κάθε φορά που εκτελούμε το πρόγραμμα να φτιάχνετε απο την αρχή ο γράφος τον αποθηκεύουμε σε ένα αρχείο $index.bin$,και μπορεί να χρησιμοποιηθεί ξανά για άλλα ερωτήματα(απο την ίδια βάση) χωρίς να ξαναφτιαχτεί,βελτιώνοντας κατά πολύ την ταχύτητα εκτέλεσης ερωτημάτων.

3 Πειράματικά αποτελέσματα

Για αρχή θα εξετάσουμε πειραματικά,την επίδραση των τριών παραμέτρων που παραθέσαμε παραπάνω στον χρόνο κατασκευής και στο $Recall$ της δομής,έχοντας για $queries$ 100 τυχαία επιλεγμένα απο το $dummy\ dataset$.Η κάθε μια παράμετρος θα μεταβάλετε ενώ οι υπόλοιπες θα παραμένουν σταθερές.

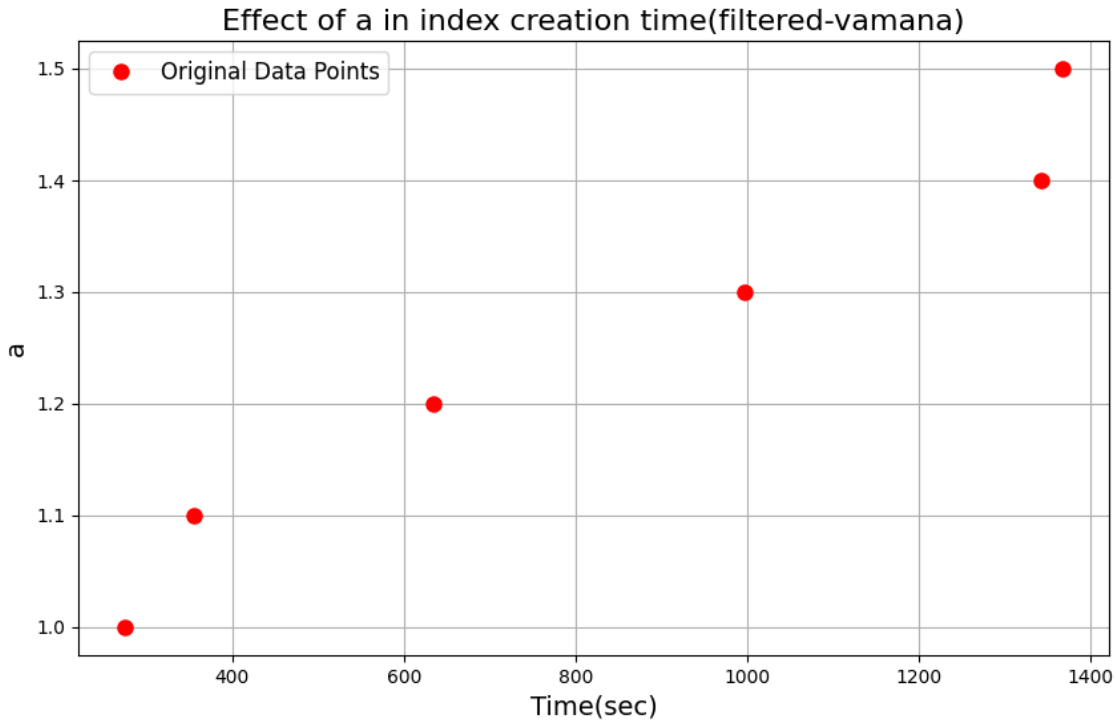
3.1 Μεταβλητό a

Παρακάτω παρατείθονται τα αποτελέσματα της επηρροής του a στον χρόνο σύγκλισης του αλγορίθμου κατασκευής της δομής.Το πρώτο γραφικό αφορά στην $filtered\ vamana$ και το δεύτερο στην $stitched\ vamana$.

Γενικά παρατηρούμε πως το a όντως επιρεάζει τον χρόνο κατασκευής με μια κάπως γραμμική σχέση στην περίπτωση του $filtered$,ενώ στην περίπτωση του $stitched$ φαίνετε να μην είναι τελείως μονότονη η σχέση,επιπλέον οι χρόνοι του πρώτου φαίνονται να καλύπτουν και τα δύο άκρα(και πολύ γρήγορη κατασκευή για χαμηλό α,αλλα και αργή κατασκευή για μεγαλύτερο α) ενώ τα αποτελέσματα του δεύτερου φαίνονται πιο μαζεμένα σε κεντικές τιμές.

3.2 Μεταβλητό L

Τώρα ας εξετάσουμε πως επηρεάζει τον χρόνο κατασκευής τον δομών το L ενώ οι υπόλοιπες παράμετροι μένουν σταθερές.



Σχήμα 1: Πως το α επηρεάζει χρονικά την κατασκευή του $filtered\ vamana$

Η σχέση του L σε σχέση με τον χρόνο κατασκευής της δομής $vamana$ στην περίπτωση της $filtered$ φαίνετε να είναι υπογραμμική,δηλαδή ο χρόνος που απαιτείτε μεγαλώνει πίο γρήγορα απο την τιμή του L.Για άλλη μια φορά παρατηρούμε οτι σαν διαγράματα μοιάζουν ώστοσο στο δεύτερο πολλες τιμές συσταδοποιούνται και υπάρχει έπειτα ραγδαία αύξηση.

4 Μεταβλητό R

Έδω στην πρώτη περίπτωση παρατηρούμε ένα ανάποδο $trend$ σε σχέση με τις υπόλοιπες παραμέτρους,δηλαδή όσο πιο μεγάλη είναι η τιμή της παραμέτρου τόσο πιο γρήγορα συγκλίνει η μέθοδος.Για άλλη μια φορά τα δεδομένα ακολουθούν παρόμοια συμπεριφορά ωστόσο οι τιμές είναι πιο απότομες με μεγαλύτερη συσταδοποίηση.

5 Βελτιστοποίηση R-τυχαίου γράφου

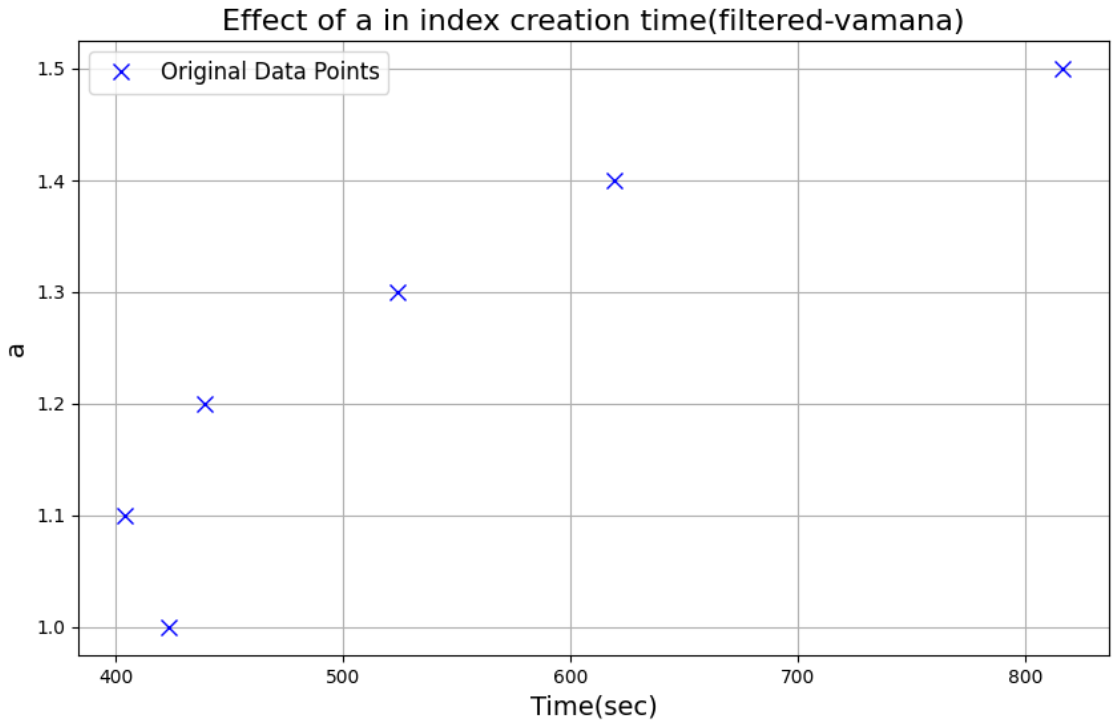
Αν και οι προηγούμενες παράμετροι δεν μεταβάλλαν ιδιαίτερα το $Recall$ της δομής, η αρχικοποίηση του R-τυχαίου γράφου αύξησε κατά πολύ το $Recall$ κυρίως των $unlabeled\ queries$.Για την ακρίβεια οι μεταβολές που σημειώθηκαν σε αυτα είναι:

$$filtered\ vamana \longrightarrow 0.4456 \longrightarrow 0.8881$$

και αντίστοιχα:

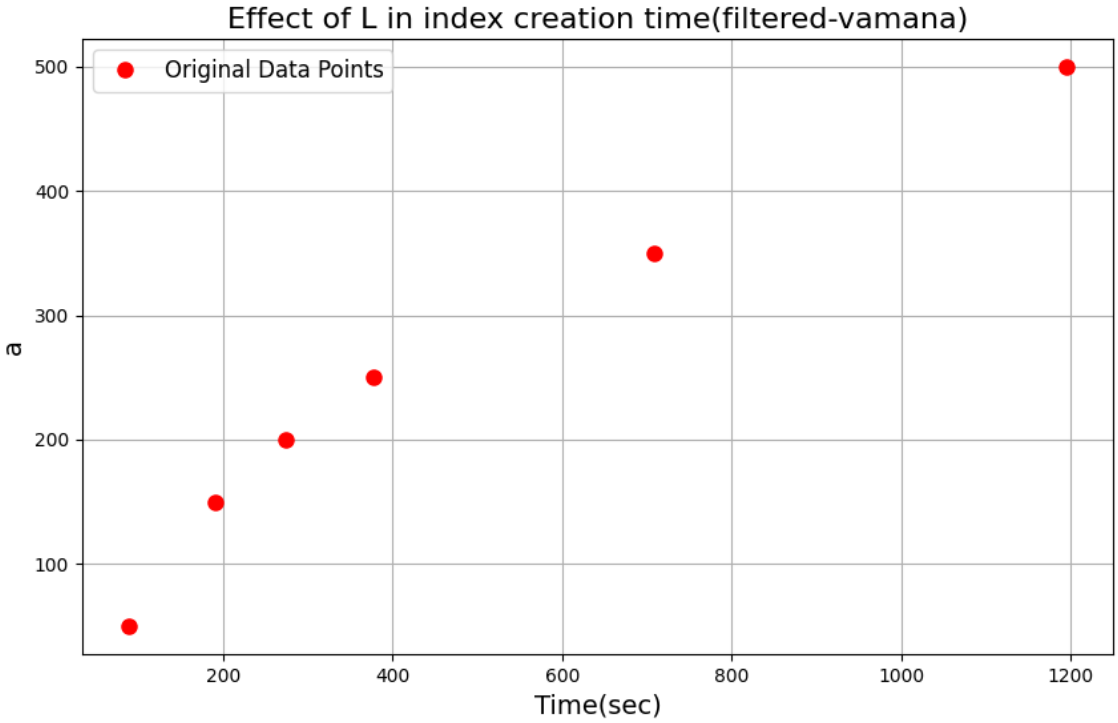
$$stitched\ vamana \longrightarrow 0.4231 \longrightarrow 0.9296$$

Ενώ τα $recall$ των $filtered$ ερωτημάτων είναι πάνω απο 0.95

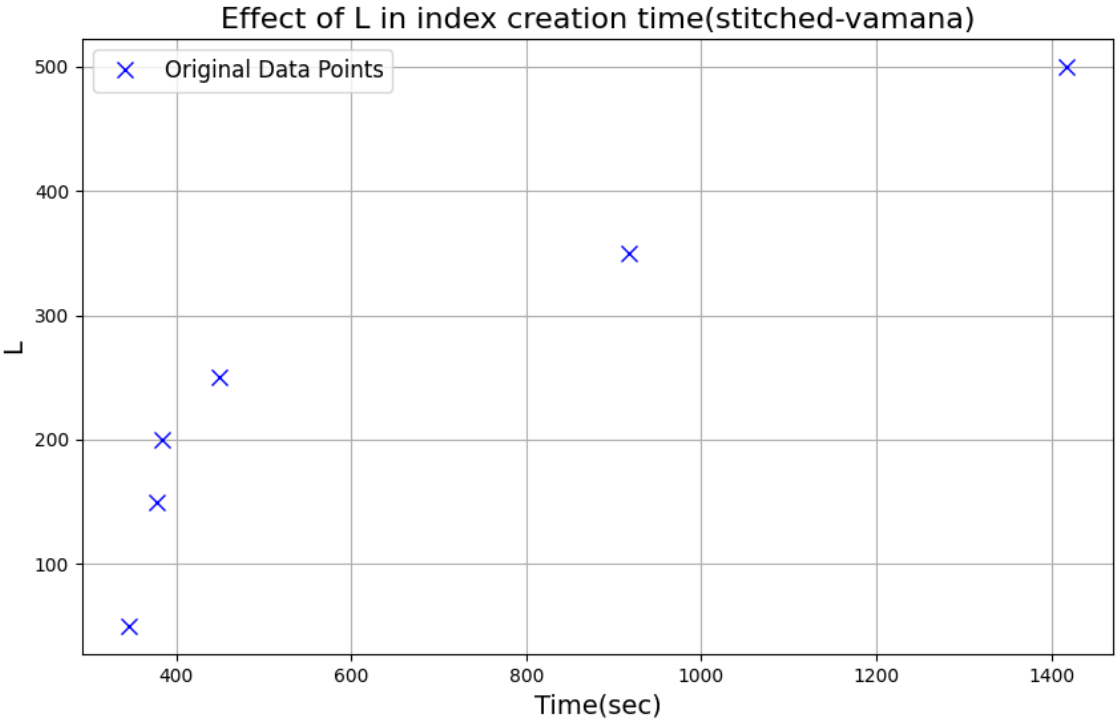


Σχήμα 2: Πως το α επηρεάζει χρονικά την κατασκευή του *stitched vamana*

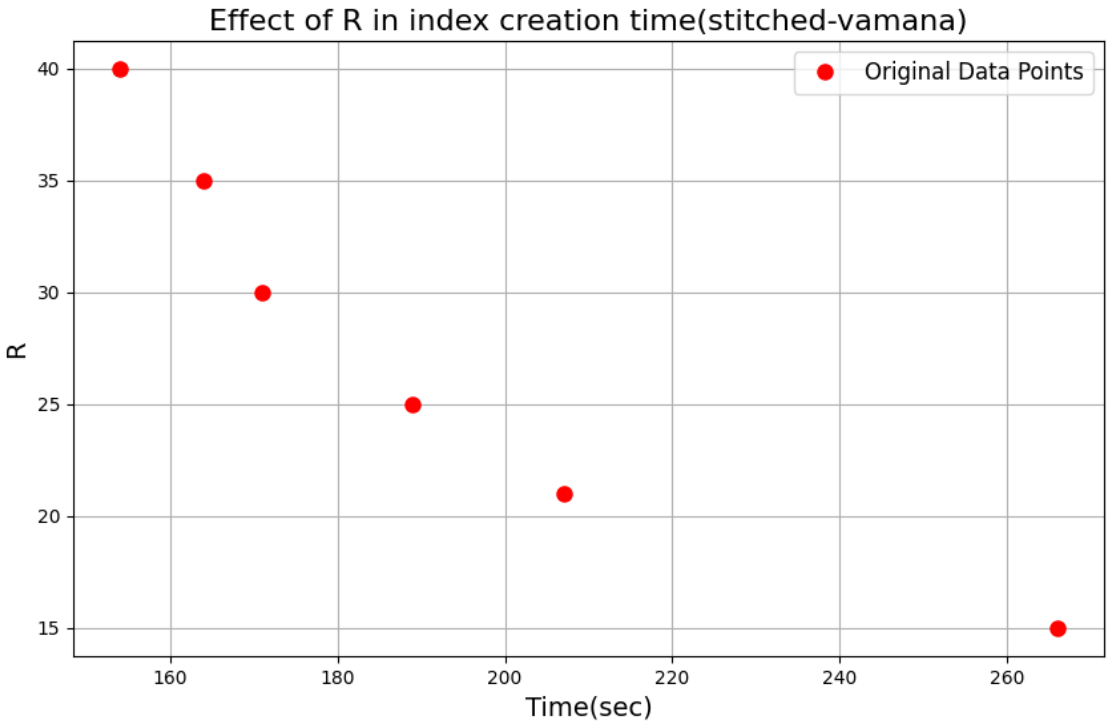
Αυτη ήταν η μελέτη των παραμέτρων αλλα και κυρίως η μέτρηση της βελτίωσης που μόλις αναφέρθηκε απο εκεί και έπειτα οποιαδήποτε άλλη βελτίωση έιτε ήδη χρησιμοποιούταν ή δεν είχε κάποιο μετρήσιμο αποτέλεσμα.Όλα τα προγράμματα τρέξαν σε εξαπύρινη CPU σε WSL.Κάτι το οποίο φυσικά δίναντε να επηρεάσει τα αποτελέσματα.Επιπλέον καταλήγουμε στον $Stiched\ Vamana$ σαν την ελαφρώς καλύτερη δομή για την επίλυση του προβλήματος μας.



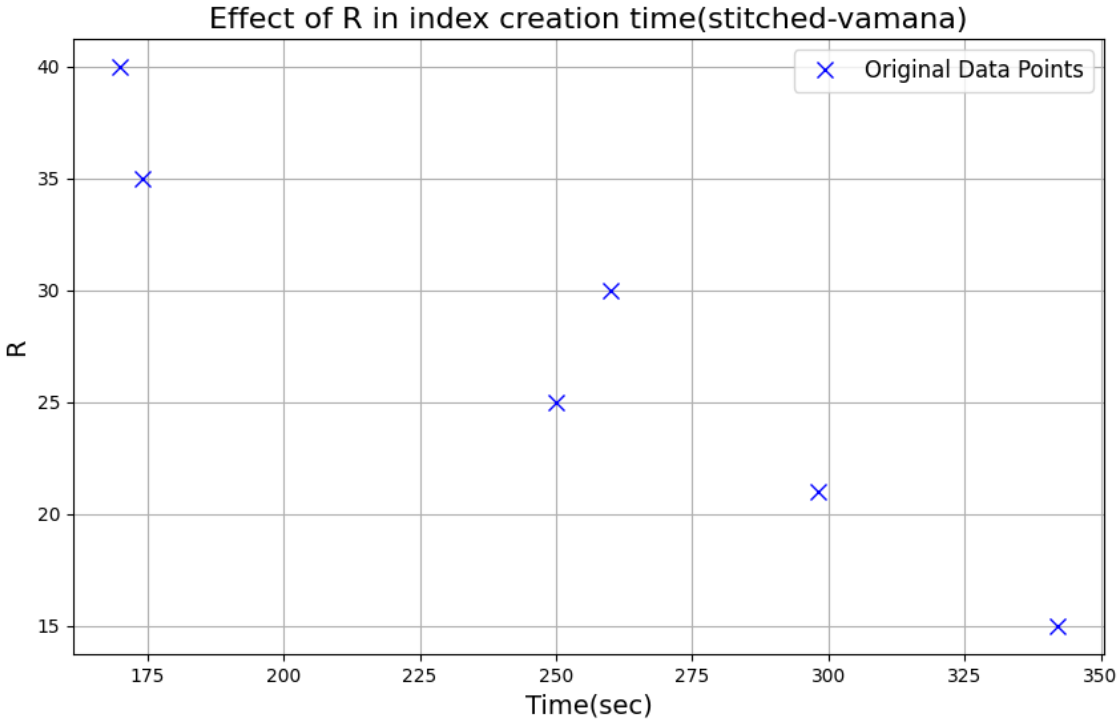
Σχήμα 3: Πως το L επηρεάζει χρονικά την κατασκευή του $filtered\ vamana$



Σχήμα 4: Πως το L επηρεάζει χρονικά την κατασκευή του $stiched\ vamana$



Σχήμα 5: Πως το R επηρεάζει χρονικά την κατασκευή του *filtered vamana*



Σχήμα 6: Πως το R επηρεάζει χρονικά την κατασκευή του *stitched vamana*