Ανάπτυξη εφαρμογών για πληροφοριακά συστήματα

Εργασία 3

Ηλίας Κωνσταντινίδης
Κωνσταντίνος Μυρτολλάρη
Άγγελος Γκουνέλας
A.M. 1115202000109
A.M. 1115202000148
A.M. 1115202000038

Όλα τα πειράματα εκτελέστηκαν στα linux του εργαστηρίου του τμήματος.

Σημαντικά χαρακτηριστικά του συστήματος που χρησιμοποιήθηκε:

- CPU(s): 4
- Thread(s) per core: 1
- Core(s) per socket: 4
- Socket(s): 1
- Model name: Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz
- OS version: Ubuntu 20.04.6 LTS
- gcc version: gcc (Ubuntu 9.4.0-1ubuntu1 20.04.2) 9.4.0

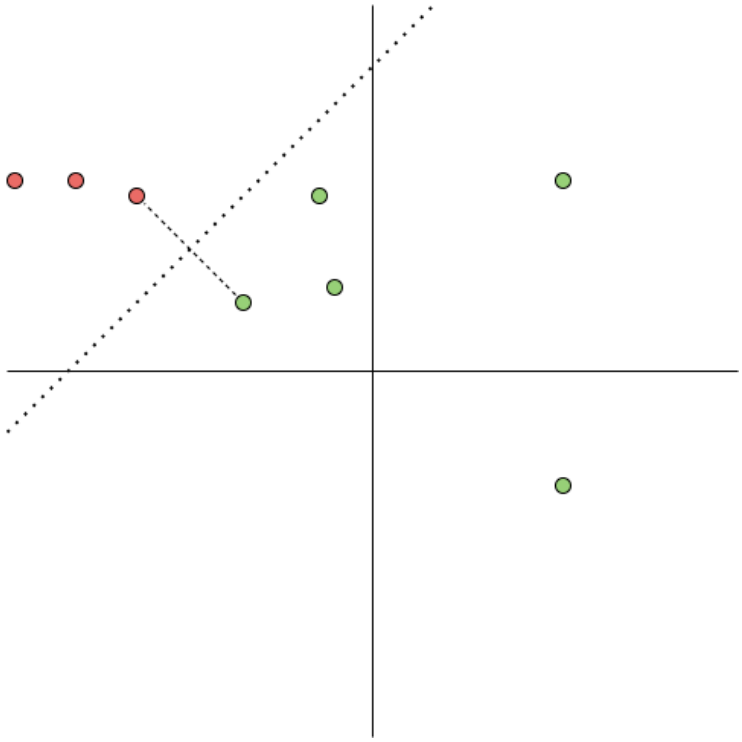
Αρχικοποίηση

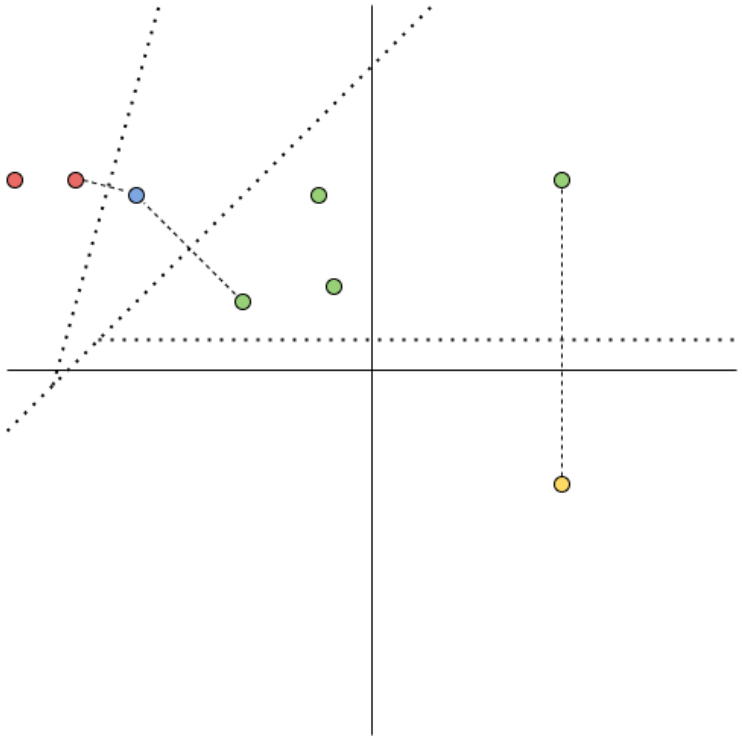
Η αρχικοποίηση του γράφου αφορά την εύρεση των αρχικών γειτόνων για όλα τα σημεία. Η πρώτη και προφανής υλοποίηση είναι η τυχαία αρχικοποίηση του γράφου. Σε αυτήν, κάθε heap αρχικοποιείται με ακριβώς k τυχαίους γείτονες. Μια άλλη αρχικοποιήση που υλοποιήθηκε είναι το δέντρο τυχαίας προβολής (random projection tree).

Δέντρο Τυχαίας Προβολής

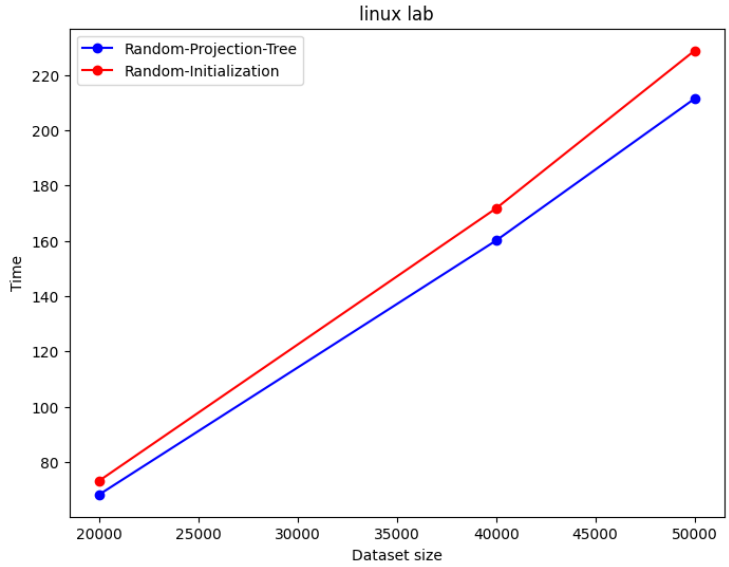
Το δέντρο τυχαίας προβολής πρόκειται για μια αρχικοποίηση που αφορά αποκλειστικά την ευκλείδια απόσταση. Στηρίζεται στην ιδέα της ομαδοποίησης των σημείων ανάλογα με το ημιεπίπεδο στο οποίο ανήκουν. Συγκεκριμένα, επιλέγονται τυχαία δύο σημεία και ορίζονται δύο ημιεπίπεδα που χωρίζονται στην μεσοκάθετο των δύο αυτών σημείων. Στη συνέχεια, κάθε σημείο εισέρχεται είτε στην ομάδα του ενός ημιεπιπέδου είτε στην ομάδα του άλλου, ανάλογα με το που ανήκουν γεωμετρικά. Αν ένα σημείο βρίσκεται πάνω στην μεσοκάθετο, τότε εισέρχεται τυχαία σε μια ομάδα. Ακόμη, αν μια ομάδα προσκομίσει όλα τα σημεία, τότε θεωρείται ακραία περίπτωση και τα σημεία χωρίζονται από την αρχή τυχαία. Η διαδικασία αυτή επαναλαμβάνεται για όλες τις ομάδες που δημιουργούνται μέχρις ότου μια από τις δύο ομάδες που επρόκειτο να δημιουργηθούν έχει πλήθος μικρότερο από μια επιλεγμένη ρήτρα. Τότε, σταματάει η περαιτέρω διάσπαση των δύο αυτών ομάδων και τα heaps αρχικοποιούνται με όλα τα σημεία και από τις δύο ομάδες ("φύλλο" του δέντρου). Η αρχικοποίηση αυτή μπορεί να επαναληφθεί, να δημιουργηθεί δηλαδή ένα νέο "δέντρο", έτσι ώστε να γίνει επιπλέον ενημέρωση των heaps και να επιτευχθεί ακόμη μεγαλύτερη ακρίβεια, θυσιάζοντας όμως ακόμη περισσότερο χρόνο.

Παρακάτω φαίνεται ένα απλό παράδειγμα δέντρου τυχαίας προβολής σε ένα δισδιάστατο επίπεδο. Στην πρώτη εικόνα, τα σημεία χωρίζονται σε κόκκινα και πράσινα. Στην δεύτερη, τα σημεία που ήταν πριν στην κόκκινη ομάδα τώρα είναι χωρισμένα σε κόκκινα και μπλε, ενώ αυτό που ήταν πράσινα τώρα είναι χωρισμένα σε πράσινα και κίτρινα.





Παρακάτω φαίνεται η διαφορά στον χρόνο εκτέλεσης του αλγορίθμου nn-descent με τυχαία αρχικοποίηση και αρχικοποίηση δέντρου τυχαίας προβολής. Μεταξύ των εκτελέσεων, όλες οι υπόλοιποι παράμετροι παρέμειναν σταθερές:



Τέλος, να σημειωθεί ότι παρατηρήθηκε καλύτερη απόδοση όταν ο αριθμός των δέντρων βρισκόταν στο διάστημα $[8, 16]$ και η ρήτρα ήταν λίγο πιο ψηλά από το K.

Heap

Έπρεπε να επιλεγεί μια δομή για την αναπαράσταση των k-πλησιέστερων γειτόνων για κάθε αντικείμενο. Η ζητούμενη δομή θα πρέπει να παρέχει γρήγορες αναζητήσεις και εισαγωγές. Για τους παραπάνω λόγους αναπτύχθηκε ένα max-heap.

Έστω ένα αντικείμενο i και το αντίστοιχο heap του. Κάθε στοιχείο του heap θα περιλαμβάνει το index $j \neq i$ ενός αντικειμένου στο dataset, την τιμή της μετρικής μεταξύ των i, j και ένα flag αρχικοποιημένο σε 1. Το flag προστέθηκε για την υποστήριξη της υλοποίησης του incremental search και θα αγνοηθεί στην τρέχουσα ενότητα. Στην ρίζα του heap βρίσκεται το αντικείμενο j με την μεγαλύτερη απόσταση από το i, δηλαδή το αντικείμενο με την μεγαλύτερη τιμή μετρικής. Το heap είναι σταθερού μεγέθους k και διπλότυπα αντικείμενα δεν επιτρέπονται. Παράλληλα, γίνεται χρήση ενός AVL δένδρου που αποθηκεύει τα indexes των αντικειμένων που βρίσκονται στο heap ανα πάσα στιγμή, ώστε να εξασφαλίζονται γρήγορες αναζητήσεις. Ενημέρωση του heap, επιφέρει και αντίστοιχη ενημέρωση του δένδρου.

Με βάση τα παραπάνω, επιτυγχάνεται πολυπλοκότητα $O(logn)$ για πράξεις αναζήτησης, εισαγωγής/αντικατάστασης και διαγραφής.

NN Descent αλγόριθμος

Δημιουργήθηκαν δύο υλοποιήσεις του αλγορίθμου NN-Descent: η σειριακή υλοποίηση και η παράλληλη υλοποίηση. Εδώ αναλύουμε την σειριακή εκδοχή καθώς η παραλληλία αναλύεται πιο μετά. Ο σκοπός του αλγορίθμου είναι δοθέντος ένος dataset με σημεία στον n-διάστατο χώρο να βρεθεί η ομάδα των K κοντινότερων στοιχείων του dataset, με κριτήριο μια μετρική (εμείς χρησιμοποιούμε την ευκλείδια μετρική), για κάθε στοιχείο του dataset. Τα K κοντινότερα στοιχεία για ένα σημείο n, λέγονται γείτονες του n. Οι γείτενος κάθε σημείου αποθηκεύονται σε ένα max heap ώστε η ενημέρωση τους να γίνεται εύκολα.

Αρχικά, η ιδέα του αλγόριθμου είναι η δημιουργία μιας αρχικής λύσης, μέσω της τεχνικής του random projection tree ή μέσω τυχαίας αρχικοποίησης. Στην συνέχεια, η αρχική λύση βελτιώνεται σταδικά μέχρι να συγκλίνει στην βέλτιστη.

Για να βελτιωθεί η εκάστοτε λύση χρησιμοποιήθηκε η τεχνική του local join. Αυτή η τεχνική επενεργεί σε κάθε σημείο n του dataset με τον εξής τρόπο: Προσπελαύνει όλα τα ζευγάρια (i,j) γειτόνων του n και ελέγχει αν ο i βελτιώνει την ομάδα K γειτόνων του j και το αντίστροφο. Κάθε φορά όταν τελειώνει μια επανάληψη, δηλαδή όταν έχουν προσπελαστεί όλα τα ζευγάρια γειτόνων όλων των σημείων του dataset, η λύση που προκύπτει είναι καλύτερη από την προηγούμενη αν γίνει έστω και μια αλλαγή στους γείτονες κάποιου σημείου. Όταν σε κάποια επανάληψη δεν βελτιωθεί η λύση, δηλαδή δεν γίνει καμία αλλαγή σε καμία γειτονιά, τότε η τρέχουσα λύση πιθανόν να συγκλίνει στην βέλτιστη και τερματίζει ο αλγόριθμος.

Για να βελτιωθεί ο χρόνος εκτέλεσης του αλγόριθμου έχουν προστεθεί και οι εξής τεχνικές βλετιστοποίησης: **Incremental Search**, **Sampling**, **Early Termination**.

Incremental Search

Σε κάθε επανάληψη του local join όλο και λιγότερες αλλαγές λαμβάνουν χώρα. Αυτό συμβαίνει γιατί η λύση συγκλίνει στην βέλτιστη. Όμως, γίνονται έλεγχοι σε πολλά ζευγάρια που έχουν ήδη ελεγχθεί σε προηγούμενη επανάληψη. Η μέθοδος του Incremental Search προσπαθεί να αποφύγει αυτούς τους περιττούς ελέγχους προσθέτοντας ένα flag σε κάθε γείτονα κάθε σημείου του dataset. Το flag, που δηλώνει ότι ένας γείτονας είναι new, αρχικοποιείται με την τιμή true όταν εισέρχεται ένας νέος γείτονας στο heap. Σε κάθε επανάληψη του local join γίνονται έλεγχοι μόνο στα ζευγάρια γειτόνων, των οποίων τουλάχιστον ο ένας είναι new. Κάθε φορά που ένας γείτονας συμμετέχει σε έναν έλεγχο, το flag του γίνεται false και ο ίδιος μετατρέπεται από new γείτονας σε old γείτονας. Με αυτή την τεχνική αποφεύγονται πολλοί περιττοί έλεγχοι και μειώνεται αισθητά ο χρόνος εκτέλεσης.

Sampling

Το γεγονός ότι η τεχνική local join λαμβάνει υπόψην και τους αντίστροφους γείτονες, δημιουργεί το πρόβλημα της μεγάλης επιβάρυνσης για ορισμένα σημεία. Ακόμη, δύο σημεία είναι πιθανό να συμμετάσχουν σε διαφορετικά local join, με αποτέλεσμα να πραγματοποιούνται περιττοί υπολογισμοί. Για αυτούς τους λόγους, υλοποιήθηκε η τεχνική του sampling.

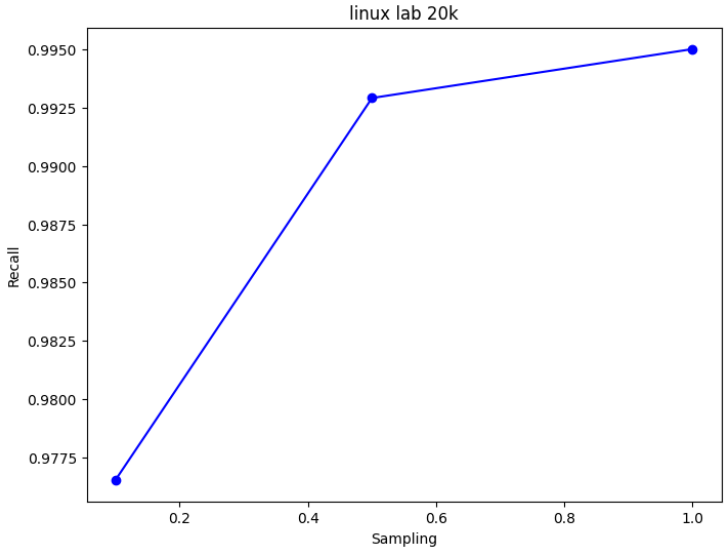
Προτού ξεκινήσει το local join γίνεται δειγματοληψία ένος πλήθους σημείων, τόσο από τους άμεσους όσο και από τους αντίστροφους γείτονες. Στη συνέχεια, πραγματοποιείται local join μόνο με αυτούς τους συγκεκριμένους γείτονες. Όσα σημεία ήταν new αλλά δεν έτυχε να επιλεγούν κατα την δειγματοληψία και δεν αντικαταστάθηκαν από κάποιο άλλο σημείο, πιθανόν να επιλεγούν σε κάποια επόμενη επανάληψη. Όσο μικρότερη είναι η δειγματοληψία, τόσο πιο γρήγορα γίνεται το local join αλλά και τόσο πιο μικρή είναι η ακρίβεια της λύσης.

Ο τρόπος με τον οποίο γίνεται η δειγματοληψία είναι ο εξής: έστω $\rho \in (0, 1]$. Σε κάθε επανάληψη, για κάθε κόμβο, δημιουργούνται δύο heaps: το old μεγέθους $K + \rho K$ και το new μεγέθους $2\rho K$, τα οποία αφορούν τόσο τους άμεσους όσο και τους αντίστροφους γείτονες. Τα heaps έχουν τα εξής χαρακτηριστικά:

- Είναι σταθερού μεγέθους, οπότε αποθηκεύουν πλήθος στοιχείων που προκύπτει από το ποσοστό sampling
- Ένα στοιχείο εισέρχεται ή όχι στο heap, με βάση την τιμή του που είναι ένα τυχαίο βάρος

Στον σειριακό αλγόριθμο, ο υπολογισμός των άμεσων και αντίστροφων γειτόνων γίνεται σε μια επανάληψη. Στα old heaps εισέρχονται όλοι οι old άμεσοι γείτονες και ρK old αντίστροφοι(εξού και $K + \rho K$ μέγεθος heap). Στα new heaps εισέρχονται new ρK άμεσοι γείτονες και ρK new αντίστροφοι(εξού και $2\rho K$ μέγεθος heap).

Παρακάτω φαίνεται η επίδραση του sampling στην ακρίβεια του αλγορίθμου:



Early Termination

Στην αρχή ο αλγόριθμος τερμάτιζε στην επανάληψη που δεν γινόταν καμία αλλαγή σε καμία γειτονιά. Όμως, στις τελευταίες επαναλήψεις που οι αλλαγές είναι ελάχιστες δεν γίνεται κάποια ουσιαστική δουλεία. Για αυτό και η τεχνική του early termination επιβάλλει έναν πρόωρο τερατισμό όταν οι αλλαγές γίνουν λιγότερες από ένα πολύ μικρό ποσοστό του πλήθους των σημείων του dataset. Με αυτόν τον τρόπο εξοικονομούνται αρκετές επαναλήψεις στο τέλος του local join που επιβάρυναν αρκετά τον αλγόριθμο χωρίς σημαντική συνεισφορά και μειώνεται ο χρόνος εκτέλεσης.

Παραλληλοποίηση

Αρχικά, έγινε χρήση της openMP για την επίτευξη παραλληλίας. Η επιλογή έγινε λόγω του απλούστατου API που προσφέρει επιτρέποντας ομαλότερη και ταχύτερη ανάπτυξη κώδικα, σε σχέση με τα posix threads. Παρέχεται η συνάρτηση **nn_descent_parallel**, η οποία και περιλαμβάνει την υλοποίηση του παράλληλου αλγορίθμου.

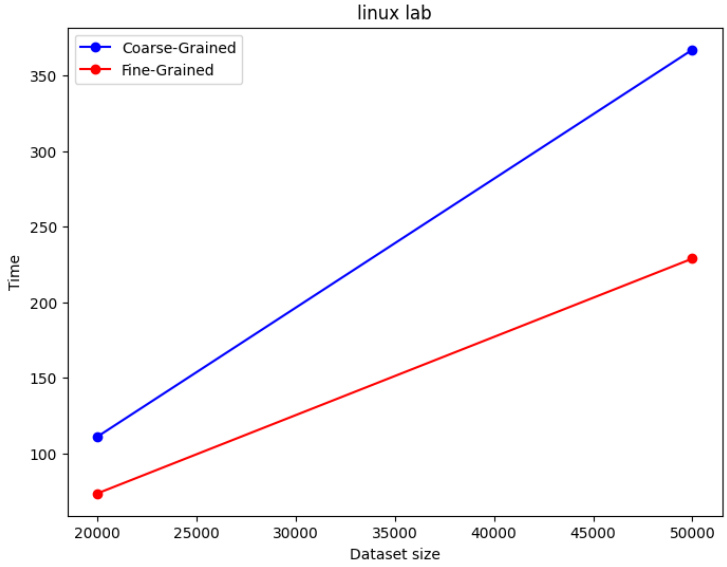
Σε κάποια σημεία δεν εμφανίζεται εξάρτηση δεδομένων και κάθε νήμα επενεργεί σε δικά του δεδομένα. Ωστόσο, όπου λαμβάνει χώρα ενημέρωση heap εμφανίζεται η ανάγκη συγχρονισμού. Έτσι, βρεθήκαμε αντιμέτωποι με την χρήση είτε fine-grained είτε coarse-grained locking. Στην περίπτωση του fine-grained locking αντιστοίχιζεται ένα κλείδωμα σε κάθε heap, ενώ στην περίπτωση του coarse-grained, ένα μοναδικό κλείδωμα χρησιμοποιείται για όλα τα heaps. Η επιλογή αυτή επηρέαζει τον υπολογισμό των αντιστρόφων γειτόνων. Στην fine-grained

υλοποίηση δύναται να γίνει παράλληλος υπολογισμός των αντιστρόφων, ενώ στην coarse-grained είναι προτιμότερο να γίνεται υπολογισμός των αντιστρόφων σειριακά από ένα νήμα.

Ωστόσο, τίθενται δύο ζητήματα στην fine-grained υλοποίηση. Πρώτον, το overhead από τα συνεχή κλειδώματα ενδέχεται να εκφυλίσει την απόδοση του προγράμματος και, δεύτερον, υπάρχει τεράστιο κόστος όσον αφορά την μνήμη. Θα πρέπει να δεσμευτούν τόσα κλειδώματα όσα και τα αντικείμενα του dataset. Για μεγάλα dataset, το κόστος ενδέχεται να είναι απαγορευτικό.

Στα πειραματικά μας αποτελέσματα παρατηρήσαμε πως η fine-grained προσέγγιση στα κλειδώματα επιφέρει καλύτερους χρόνους. Για αυτό τον λόγο και ακολουθήσαμε την προσέγγιση αυτή στο παραδοτέο μας. Ωστόσο, η επιβάρυνση στην μνήμη δεν πρέπει να διαφεύγει από τον νου μας.

Ακολουθεί διάγραμμα σύγκρισης των δύο προσεγγίσεων:

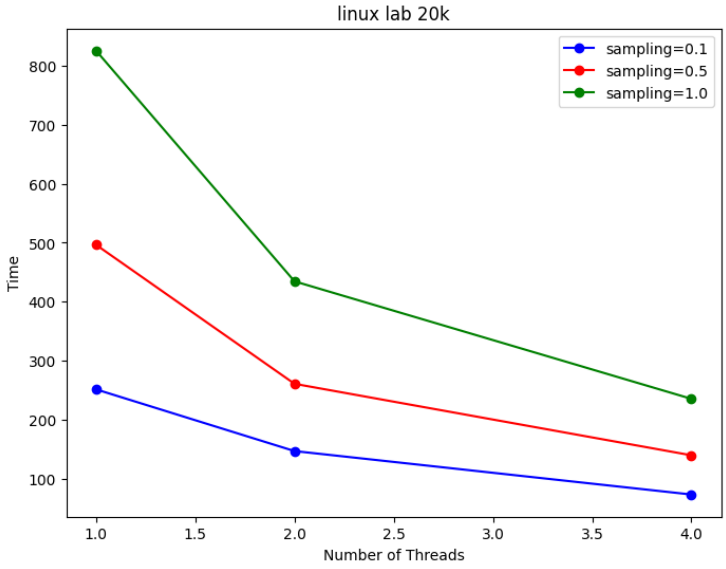


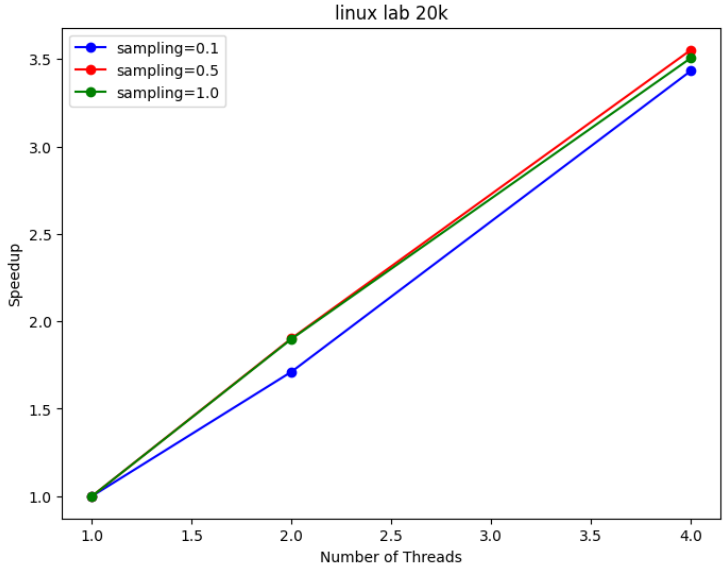
Πέραν του local join, παραλληλοποιήθηκαν και οι δύο μέθοδοι αρχικοποίησης του γράφου. Για το random projection tree παραλληλοποιείται η δημιουργία των δένδρων.

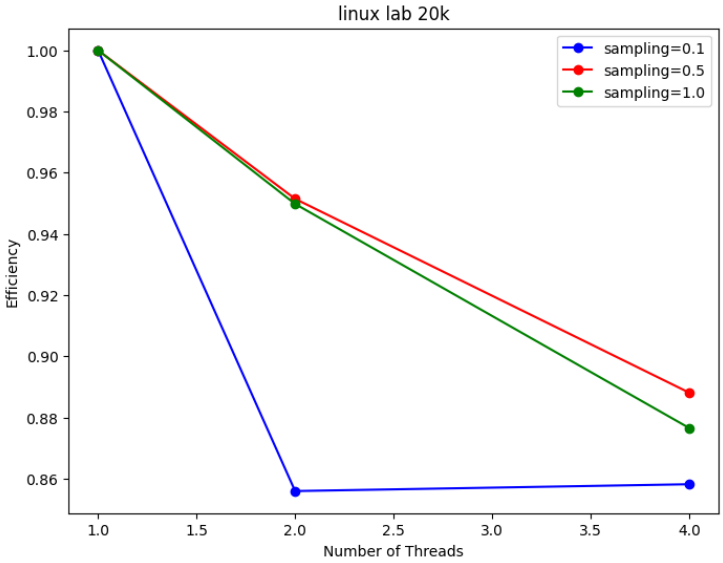
Συμπεράσματα

Οι δύο βασικοί μέθεδοι μείωσης του χρόνου εκτέλεσης του αλγορίθμου που αξίζει να μελετηθούν είναι το sampling και ο αριθμός των threads. Η παράμετρος d

του early termination μένει σταθερά στο 0.001, η αρχικοποίηση έγινε τυχαία και χρησιμοποιήθηκε το dataset των 20k. Ακολουθούν διαγράμματα χρόνου, επιτάχυνσης και αποδοτικότητας:

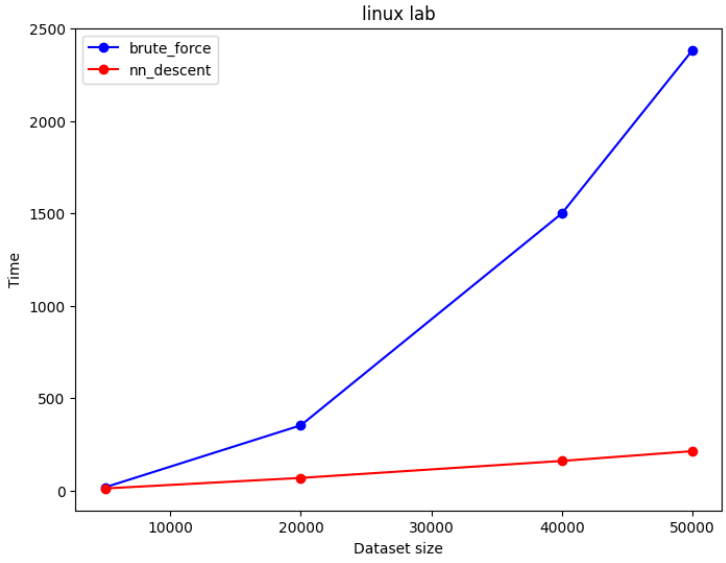






Παρατηρείται ότι η αύξηση του αριθμού των νημάτων επιφέρει γραμμική επιτάχυνση. Επίσης, όπως ήταν αναμενόμενο, η μείωση του παραγόντα ρ (sampling) επιφέρει μείωση του χρόνου εκτέλεσης. Να τονιστεί όμως πως η μείωση της τιμής του συγκεκριμένου παράγοντα προκαλεί μείωση στην ακρίβεια του αλγορίθμου. Αντιθέτως, ο αριθμός των threads δεν έχει καμία επίπτωση στην ακρίβεια.

Τέλος, παρουσιάζεται η διαφορά του αλγορίθμου brute force και του αλγορίθμου nn-descent για διάφορα μεγέθη dataset ως προς τον χρόνο εκτέλεσης. Τα αποτελέσματα του αλγορίθμου nn-descent είναι τα καλύτερα δυνατά για το οποία επιτυγχάνεται ακρίβεια $> 95\%$. Χρησιμοποιήθηκε η μέθοδος του random projection tree, χαμηλό sampling και παραλληλία



References

[1] `https://pynndescent.readthedocs.io/en/latest/how_pynndescent_works.html`.

[2] Wei Dong, Charikar Moses, and Kai Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, page 577–586, New York, NY, USA, 2011. Association for Computing Machinery.

[3] Dan Kluser, Jonas Bokstaller, Samuel Rutz, and Tobias Buner. Fast single-core k-nearest neighbor graph computation. *arXiv preprint arXiv:2112.06630*, 2021.