2023-2024
Τεχνολογίες Υλοποίησης Αλγορίθμων

Δυναμική Συνεκτικότητα Γραφημάτων

Τελική Εργασία

Γιώργος Ράικος
ΑΜ: 1072558
up1072558@ac.upatras.gr
Έτος 5ο

Λεπτομέρειες υλοποίησης

Η υλοποίηση του δυναμικού αλγορίθμου Even & Shiloach για διατήρηση συνεκτικότητας έγινε με χρήση της βιβλιοθήκης Boost για γραφήματα. Ο αλγόριθμος περιλαμβάνει ορισμένες ιδιόμορφες τεχνικές κατά τη θεωρητική περιγραφή του, όπως η “παράλληλη” εκτέλεση διεργασιών, η ανάγκη αναίρεσης αλλαγών στη δομή και η ανάγκη διατήρησης συνόλων ακμών για κάθε κορυφή. Παρακάτω θα αναλυθεί η βασική δυναμική δομή με τις βασικές της μεθόδους, καθώς επίσης και οι τεχνικές υλοποίησης των παραπάνω ιδιόμορφων τεχνικών.

Παράλληλη εκτέλεση

Η δυναμική δομή περιλαμβάνει την εκτέλεση δύο βασικών διεργασίων (A και B κατά την δημοσίευση). Η διεργασία A ανιχνεύει αποδοτικά την περίπτωση που η αφαίρεση μιας ακμής προκαλεί τον χωρισμό μιας συνεκτικής συνιστώσας σε δύο, και αναλαμβάνει να ανανεώσει το σχετικό διάνυσμα συνεκτικών συνιστωσών. Η διεργασία B ανιχνεύει την περίπτωση που η συνεκτική συνιστώσα παραμένει ως έχει, ενώ παράλληλα ανανεώνει δυναμικά την δομή ώστε να διατηρεί συνεχώς μια BFS ιεραρχία.

Οι δύο αυτές διεργασίες πρέπει να εκτελούνται παράλληλα, και όταν η μια εξ αυτών τερματίσει υπό συγκεκριμένες συνθήκες, θα πρέπει και η άλλη να σταματήσει. Η παράλληλη εκτέλεση εδώ σημαίνει εναλλάξ εκτέλεση των στοιχειωδών βημάτων κάθε διεργασίας.

Για να εξυπηρετούν αυτό τον σκοπό, οι δύο βασικοί αλγόριθμοι των διεργασιών (DFS και δυναμική ανανέωση της BFS δομής) υλοποιούνται ως μια κλάση με πρόθεμα *Step*, η οποία διατηρεί την τρέχουσα κατάσταση της εκτέλεσης, τις τρέχουσες μεταβλητές για κάθε βήμα, και μια μεταβλητή κατάστασης, η οποία ανανεώνεται κατάλληλα σε κάθε βήμα.

Οι κλάσεις αυτές διαθέτουν τη μέθοδο *advance*, η οποία προχωρά την βηματική εκτέλεση κατά ένα βήμα. Εσωτερικά, η μέθοδος αυτή χρησιμοποιεί ένα switch statement για την τρέχουσα κατάσταση, εκτελεί τον κώδικα του τρέχοντος βήματος, και ανανεώνει την κατάσταση κατάλληλα. Εξωτερικά, όταν ολοκληρωθεί η εκτέλεση του αλγορίθμου, η μεταβλητή κατάστασης τίθεται σε κατάλληλη τιμή *Finished*, και διαθέτει κατάλληλη μεταβλητή αποτελέσματος της εκτέλεσης, όπου αυτό χρειάζεται.

Επομένως, η δυναμική δομή μετά από κάθε αφαίρεση καλεί εναλλάξ τις αντίστοιχες μεθόδους *advance*, μέχρις ότου επιτευχθούν οι τερματικές συνθήκες.

Αναίρεση αλλαγών

Κατά τη θεωρητική περιγραφή, στην περίπτωση που η διεργασία A ανιχνεύσει σπάσιμο της συνεκτικής συνιστώσας, θα πρέπει να σταματήσει η εκτέλεση της διεργασίας B. Θα πρέπει,

όμως, οι αλλαγές που έχουν γίνει στην δομή από την τρέχουσα εκτέλεση της διεργασίας Β να αναιρεθούν. Για τον λόγο αυτό, γίνεται χρήση μιας δομής στοίβας, στην οποία αποθηκεύονται εγγραφές κατά την εκτέλεση της διεργασίας Β. Στην περίπτωση που χρειαστεί, έτσι, αναίρεση, καλείτε η μέθοδος *rewind*, και εκτελείται η αντίστροφη διαδικασία για κάθε εγγραφή της στοίβας για την αναίρεση της εκάστοτε αλλαγής.

Εδώ να σημειωθεί πως για εξοικονόμηση χώρου, δεν αποθηκεύονται αντίγραφα των δομών εντός των εγγραφών της στοίβας, εκτός των απολύτως απαραίτητων. Για να επιτευχθεί αυτό, οι επαναφορές στα σύνολα *α, β, γ*, γίνονται εκμεταλλευόμενες την σειρά εκτέλεσης των βημάτων του avalanche, όπως περιγράφεται στη δημοσίευση. Εκτενέστερη περιγραφή αυτών υπάρχει στα σχόλια του συγκεκριμένου τμήματος κώδικα.

Σύνολα ακμών

Στην υλοποίηση του αλγορίθμου κομβικό ρόλο έχουν τα σύνολα ακμών *α, β, γ*. Για αποδοτική υλοποίηση είναι επιτακτική η ανάγκη χρήσης κατάλληλης δομής. Για αυτά τα σύνολα δημιουργείται η κλάση *EdgeSet,* η οποία εσωτερικά χρησιμοποιεί την δομή unordered_set, δηλαδή δομή τύπου Hash Map. Οι ακμές αποθηκεύονται ως ζεύγη ακεραίων. Επειδή η Boost σε μη κατευθυνόμενα γραφήματα εμφανίζει τις ακμές και ως (u,v) και ως (v,u), ανάλογα τον τρόπο προσπέλασης, η δομή *EdgeSet* φροντίζει να αποθηκεύει πάντα τα ζεύγη ως (min(v, u), max(v, u)). Η δομή υλοποιεί, επίσης, ειδική συνάρτηση κατακερματισμού για ζεύγη ακεραίων, η οποία χρησιμοποιεί την συνάρτηση *hash_combine* της Boost.

Σημειώνεται πως κατά την θεωρητική περιγραφή χρησιμοποιείται η έννοια των τεχνητών ακμών (artificial edges) για τη διεργασία Β. Όπως διαπιστώνεται και στη δημοσίευση, οι ακμές αυτές χρησιμεύουν στην θεωρητική ανάλυση, καθώς με αυτές εξασφαλίζεται η διατήρηση μιας ενιαίας δομής BFS, και άρα διευκολύνεται η ανάλυση ορθότητας και πολυπλοκότητας. Ωστόσο, στην πράξη δεν είναι αναγκαία.

Οι ακμές αυτές δεν επηρεάζουν την αναδιοργάνωση της επιμέρους συνεκτικής συνιστώσας στην οποία ανήκει μια ακμή πριν την αφαίρεσή της. Η επιμέρους BFS δομή θα παραμείνει, λόγω της διεργασίας Β, δομημένη ως έχει. Με την πρόοδο της εκτέλεσης θα χαθεί η ενιαία BFS δομή όλου του γραφήματος, καθώς αλλαγές στα levels συνεκτικής συνιστώσας διαφορετικής από την επιμέρους δεν θα επηρεάζει τα levels της επιμέρους. Η επιμέρους BFS δομή που θα παραμένει, ωστόσο, θα είναι αρκετή για την επίτευξη του σκοπού του αλγορίθμου.

Έλεγχος ορθότητας

Για τον έλεγχο ορθότητας σε παραδείγματα των συνθετικών και των τυχαίων γραφημάτων, υλοποιούνται συναρτήσεις που εκτελούν τις αφαιρέσεις και ελέγχουν με το αναμενόμενο

αποτέλεσμα. Οι συναρτήσεις βρίσκονται στο αρχείο test.cpp, και για να δημιουργηθεί το executable αρκεί η εντολή make test, και make test_run για την εκτέλεσή τους.

Λεπτομέρειες υλοποίησης για διατήρηση σωστής πολυπλοκότητας

Παρακάτω παρατίθενται ορισμένες μικρές λεπτομέρειες της υλοποίησης, που έχουν όμως μεγάλη σημασία για τη διατήρηση της πολυπλοκότητας που περιγράφεται θεωρητικά.

Στην περίπτωση που η διεργασία Α αποφανθεί πως η η συνεκτική συνιστώσα δεν σπάει από την αφαίρεση της ακμής, η δημοσίευση αναφέρει πως συνεχίζεται η εκτέλεση της διεργασίας Β, ώστε να διατηρηθεί η δομή. Σε αυτή την περίπτωση, η υλοποίηση θα σταματήσει την καταγραφή αλλαγών στην στοίβα για εξοικονόμηση χώρου, εφόσον πλέον είναι βέβαιο πως δεν θα χρειαστεί αναίρεση.

Επίσης, για τα σύνολα ακμών υλοποιούνται κατάλληλοι move constructors και κατάλληλα move semantics ώστε να αποφευχθούν περιττά αντίγραφα δεδομένων που θα κόστιζαν γραμμικό χρόνο. Έτσι, η μεταφορά συνόλων που χρησιμοποιείται κατά κόρον στο avalanche γίνεται σε σταθερό χρόνο.

Παρόμοια μεταχείριση υπάρχει και στις εγγραφές της στοίβας, στην οποία πάντοτε προστίθενται εγγραφές με move, και όχι με copy, και αντίστοιχα κατά την ανάγνωση της κορυφής της στοίβας χρησιμοποιείται move.

Τέλος, στην υλοποίηση του αλγορίθμου DFS, που χρησιμοποιείται εσωτερικά από τη διεργασία Α, η δομή που διατηρεί την πληροφορία για τις visited κορυφές είναι δομή unordered_set. Αυτό επιτυγχάνει μέση περίπτωση με σταθερό χρόνο για lookup και insert, εφόσον πρόκειται για δομή hash map, και αποφεύγει πιθανή αρχικοποίηση διανύσματος μεγέθους ίσου με το πλήθος των κορυφών, η οποία θα κόστιζε γραμμικό χρόνο και θα υπερτερούσε της θεωρητικής λογαριθμικής θεωρητικής πολυπλοκότητας, καταστρέφοντας έτσι την ανάλυση του αλγορίθμου.

Συνθετικά γραφήματα

Για την πειραματική αξιολόγηση του αλγορίθμου, πέρα από τα τυχαία γραφήματα, χρησιμοποιούνται και συνθετικά γραφήματα. Αυτά είναι τα γραφήματα γραμμές, γραφήματα δακτύλιοι και πλήρως συνδεδεμένα γραφήματα.

Γραφήματα γραμμής

Πρόκειται για γραφήματα στα οποία κάθε κορυφή n συνδέεται με την αμέσως επόμενη κορυφή n+1. Χρησιμοποιούνται καθώς σε αυτά μπορεί να αποτυπωθεί η πολυπλοκότητα χειρότερης περίπτωσης O(|E|log|E|) για τη διεργασία Α.

Συγκεκριμένα, εφόσον πρόκειται για άκυκλο γράφημα, κάθε αφαίρεση θα σπάει την αντίστοιχη συνεκτική συνιστώσα. Επομένως θα εκτελείται “μόνο” η διεργασία Α. Επίσης, κατά τη θεωρητική μελέτη, επειδή η διεργασία Α σαρώνει τα υποδέντρα σταματώντας όταν σαρώσει πλήρως το μικρότερο εκ των δύο, η χειρότερη περίπτωση θα προκύπτει όταν, μετά τη διάσπαση, τα δυο επιμέρους υποδέντρα έχουν το ίδιο μέγεθος. Για να επιτευχθεί αυτό θα πρέπει αρχικά το πλήθος των κορυφών να είναι δύναμη του 2, και η αλληλουχία αφαιρέσεων να σπάει κάθε φορά στη μέση τα υποδέντρα. Για παράδειγμα, σε γράφημα γραμμή πλήθους 8 κορυφών, η αλληλουχία θα είναι 4, 2, 6, 1, 3, 5, 7, θεωρώντας ως ακμή 4 την ακμή (3,4) κ.ο.κ.

Γραφήματα δακτυλίου

Πρόκειται για γραφήματα στα οποία κάθε κορυφή n συνδέεται με την αμέσως επόμενη κορυφή n+1 και η τελευταία κορυφή συνδέεται με την κορυφή 0. Με αυτά μπορεί να αποτυπωθεί η πολυπλοκότητα χειρότερης περίπτωσης O(|V||E|) για τη διεργασία Β.

Συγκεκριμένα, η αφαίρεση της πρώτης ακμής θα είναι η μοναδική φορά που θα κυριαρχήσει η διεργασία Β. Με αυτό τον τρόπο θα μπορεί να παρατηρηθεί συγκεντρωμένο όλο το επιμερισμένο κόστος των εκτελέσεων που κυριαρχούνται από τη διεργασία Β. Μάλιστα, ο δακτύλιος και η αφαίρεση της μιας εκ των δύο ακμών που προσπίπτουν στην κορυφή που έχει επιλεγεί τυχαία ως ρίζα επιφέρει πολυπλοκότητα της τάξης της χειρότερης περίπτωσης.

Με την αφαίρεση της ακμής, θα πρέπει επαναληπτικά να εκτελεστεί η διαδικασία avalanche για κάθε μια από τις ακμές του ημικυκλίου. Έστω ρίζα η κορυφή 0. Η αφαίρεση της ακμής (0,1) σημαίνει πως το επίπεδο όλης της αλυσίδας του ημικυκλίου θα πρέπει να ανανεωθεί σταδιακά. Η αλληλουχία ανανεώσεων θα είναι η 1, 2, 1, 2, 3, ..., θα είναι δηλαδή πλήθος 1 + 2 + 3 + ... + n/2 = $O(n^2)$, με σταθερό όρο ¼ * c, εφόσον η ανανέωση θα αφορά μόνο τις μισές κορυφές.

Πλήρως συνδεδεμένα γραφήματα

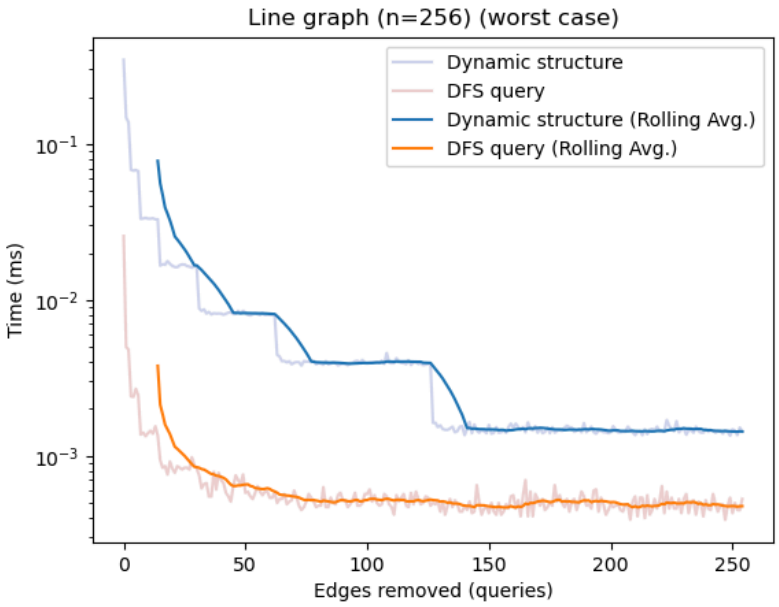
Παρουσιάζει ενδιαφέρον η πειραματική αξιολόγηση του αλγορίθμου σε πλήρως συνδεδεμένα γραφήματα, όπου τυχαία αφαίρεση ακμών έχει μικρή πιθανότητα να σπάσει κάποια συνεκτική συνιστώσα.

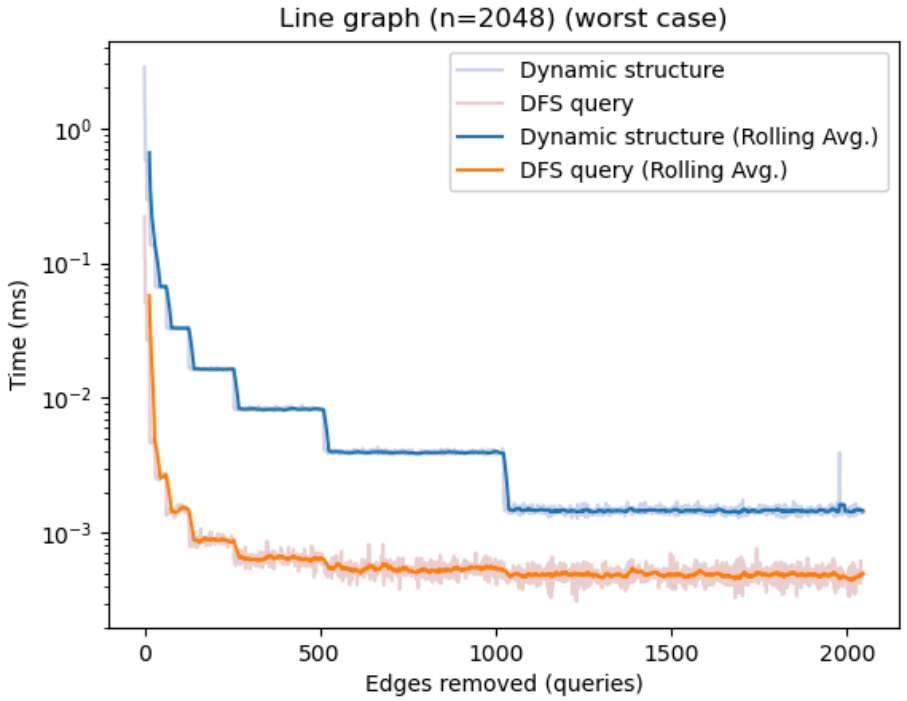
Πειραματική αξιολόγηση

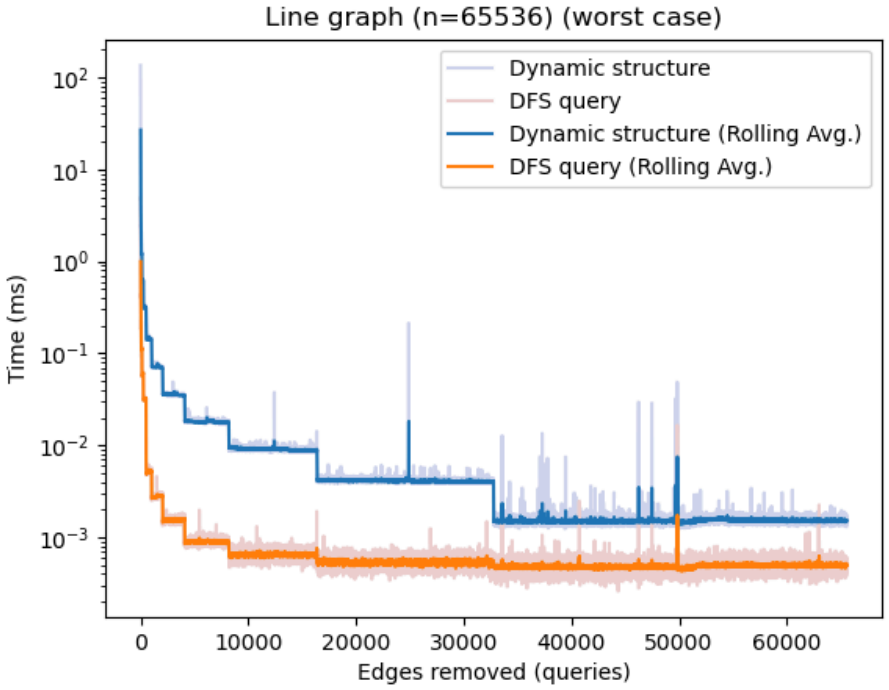
Παρακάτω παρουσιάζονται διαγράμματα που δημιουργήθηκαν με Python scripts και συγκρίνουν τους χρόνους απάντησης σε ερωτήματα συνεκτικότητας. Συγκεκριμένα, συγκρίνεται ο αλγόριθμος Even-Shiloach με τον απλό αλγόριθμο σάρωσης DFS. Για τον πρώτο, η μέτρηση αφορά το χρόνο δυναμικής αναδιοργάνωσης της δομής μετά την αφαίρεση, ενώ για τον δεύτερο η μέτρηση αφορά την εκτέλεση του αλγορίθμου σε αναζήτηση του άλλου άκρου.

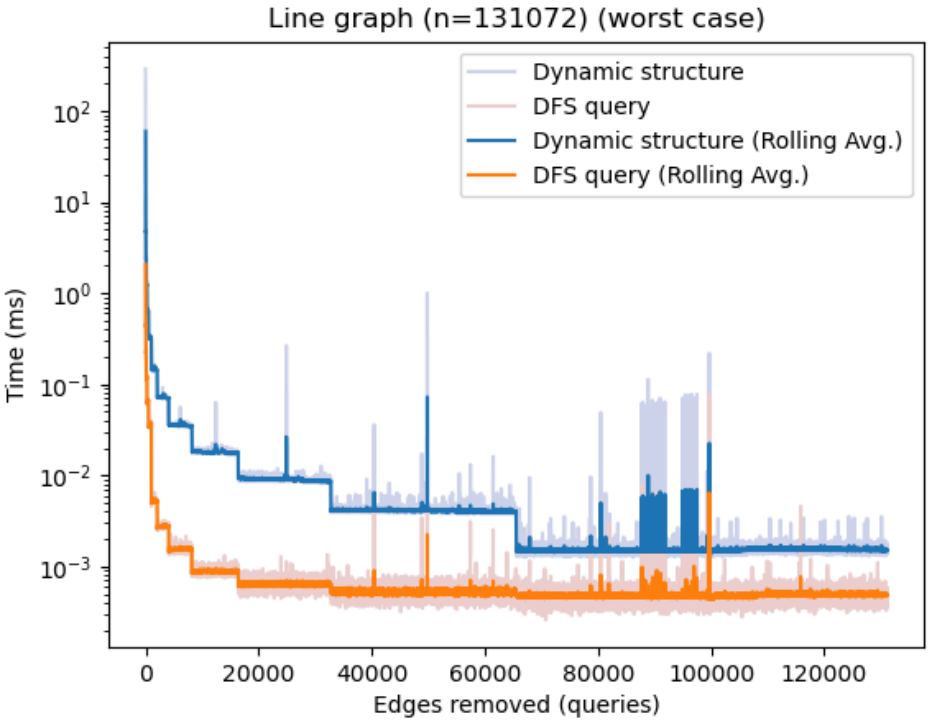
Γραφήματα γραμμής

Αρχικά παρουσιάζονται μετρήσεις για την συγκεκριμένη αλληλουχία αφαίρεσης ακμών, και έπειτα για τυχαία αλληλουχία αφαίρεσης ακμών.



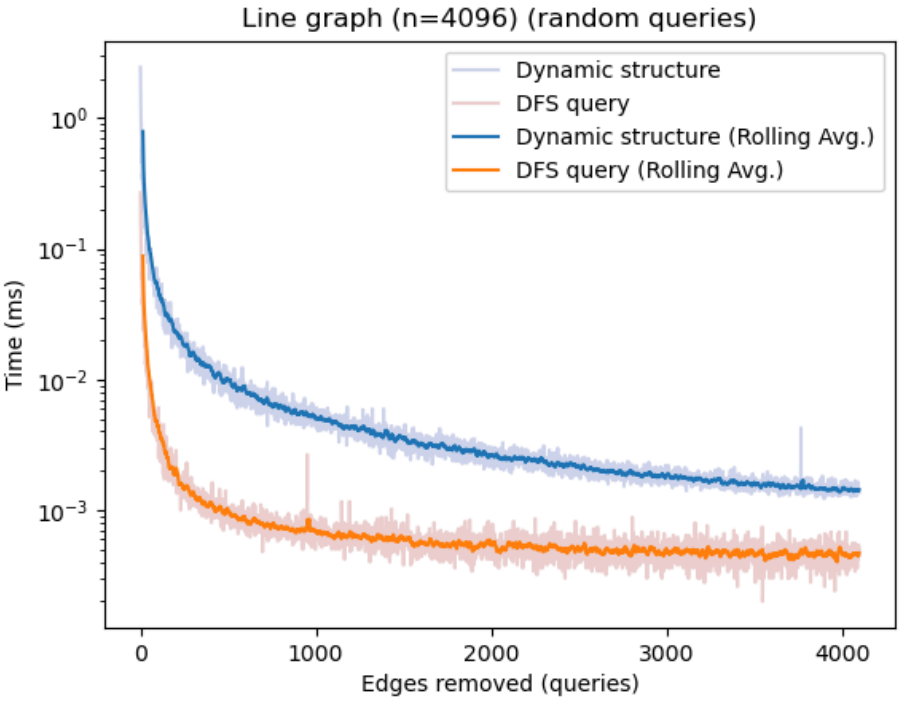

Line graph (n=2048) (worst case)
Dynamic structure
DFS query
Dynamic structure (Rolling Avg.)
DFS query (Rolling Avg.)
Time (ms)
10^{0}
10^{-1}
10^{-2}
10^{-3}
0
500
1000
1500
2000
Edges removed (queries)

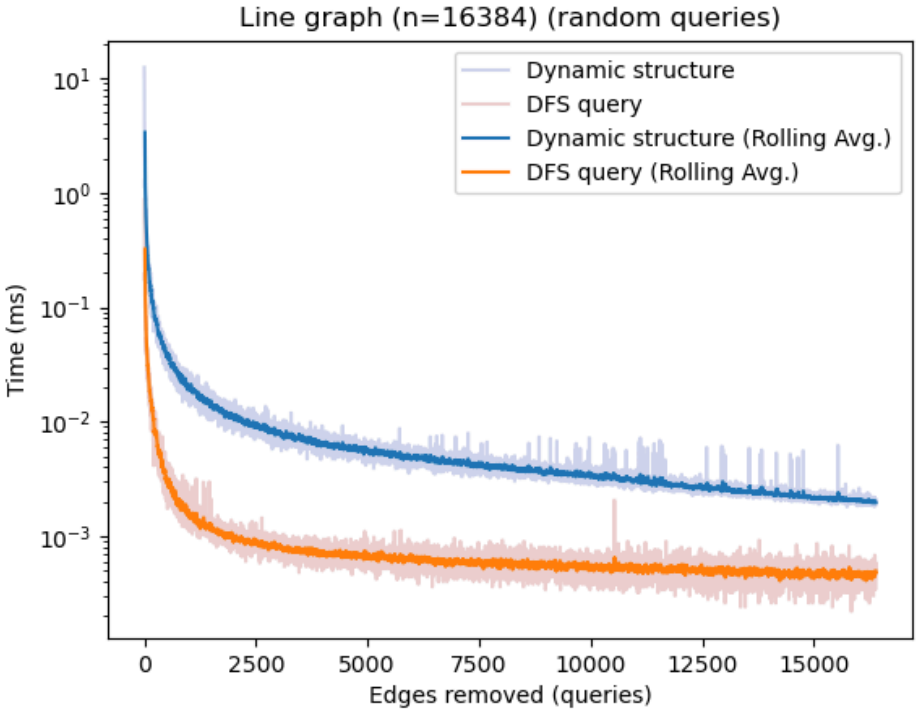

Line graph (n=65536) (worst case)
Dynamic structure
DFS query
Dynamic structure (Rolling Avg.)
DFS query (Rolling Avg.)
Time (ms)
10^{2}
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
0
10000
20000
30000
40000
50000
60000
Edges removed (queries)



Όπως είναι αναμενόμενο, παραπάνω αποτυπώνεται η μείωση στο μισό, κάθε φορά που αλληλουχία συνεχίζει σε μικρότερα υποδέντρα. Ενδιαφέρον παρουσιάζουν, επίσης, τα spikes που παρατηρούνται για μεγάλα δεδομένα εισόδου. Εδώ αρχίζουν να εμφανίζονται και ζητήματα μνήμης, λόγω του μεγέθους. Αξιοσημείωτα είναι, επίσης, τα spikes που παρατηρούνται στον δυναμικό αλγόριθμο. Μετά από σχετική εξέταση, έγινε αντιληπτό πως προέρχονται από το rehashing που επιτελούν εσωτερικά οι οι δομές unordered_set (που υπάρχουν στα EdgeSet *α, β, γ*) μετά από συγκεκριμένο αριθμό εισαγωγών. Αυτό εξηγεί και το ύψος των spikes, το οποίο είναι σχεδόν ανάλογο του πλήθους των ακμών.

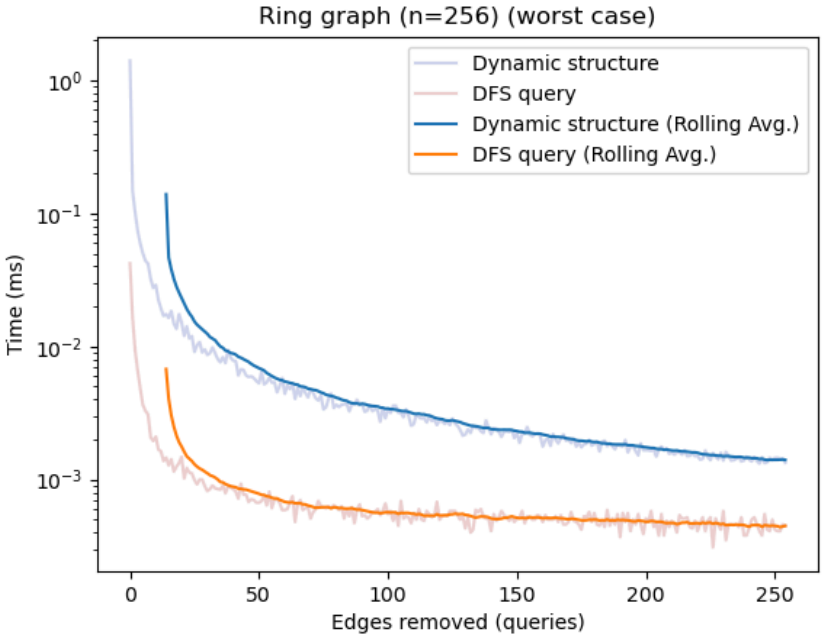
Παρακάτω οι μετρήσεις για τυχαία αλληλουχία. Όπως είναι αναμενόμενο, εδώ δεν υπάρχει η ξεκάθαρη μείωση στη μέση.

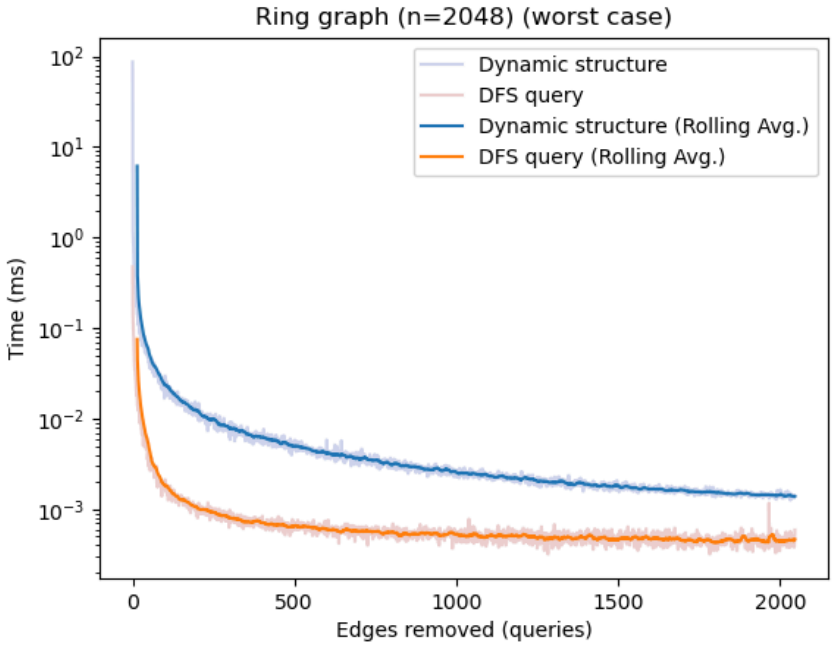

Line graph (n=4096) (random queries)
Dynamic structure
DFS query
Dynamic structure (Rolling Avg.)
DFS query (Rolling Avg.)
Time (ms)
10^{0}
10^{-1}
10^{-2}
10^{-3}
0
1000
2000
3000
4000
Edges removed (queries)

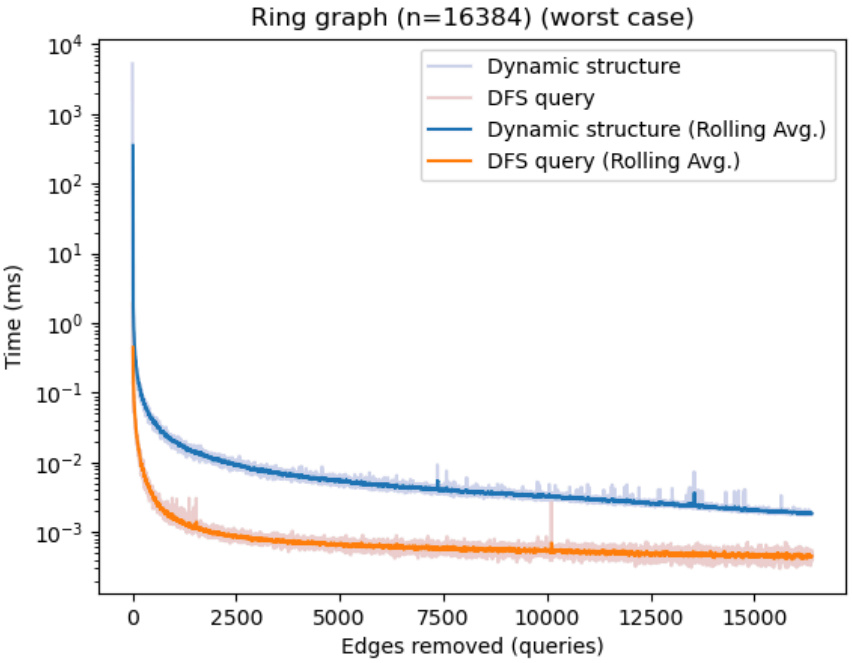

Line graph (n=16384) (random queries)
Dynamic structure
DFS query
Dynamic structure (Rolling Avg.)
DFS query (Rolling Avg.)
Time (ms)
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
0
2500
5000
7500
10000
12500
15000
Edges removed (queries)

Γραφήματα δακτυλίου

Πράγματι, όπως φαίνεται παρακάτω, ο δυναμικός αλγόριθμος έχει μεγάλο κόστος στο πρώτο ερώτημα, που είναι επιλεγμένο ώστε να επιφέρει τη χειρότερη δυνατή αναδιοργάνωση από τη διεργασία B. Και ο DFS, βέβαια, δαπανά σημαντικό μέγεθος χρόνου, καθώς τα άκρα είναι στη μέγιστη δυνατή απόσταση στο δακτύλιο μετά την αφαίρεση.

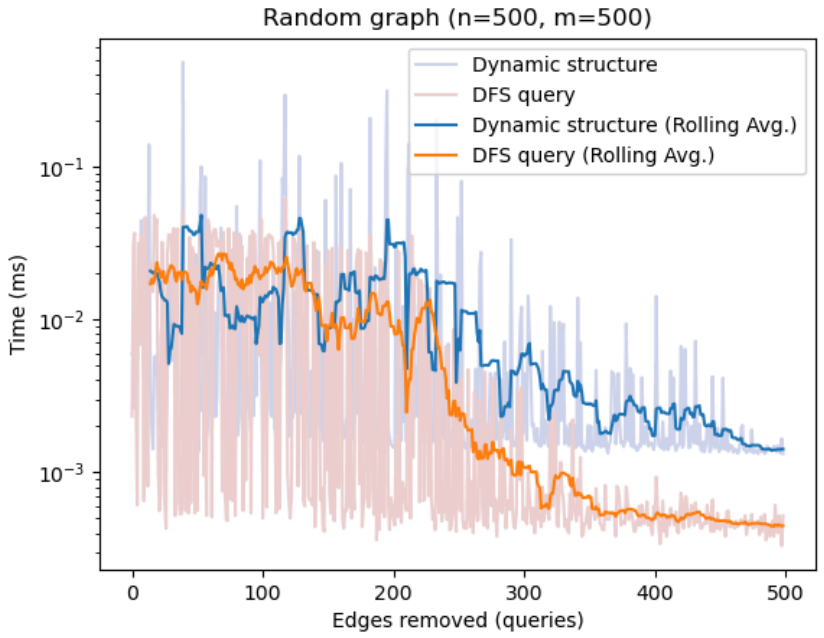


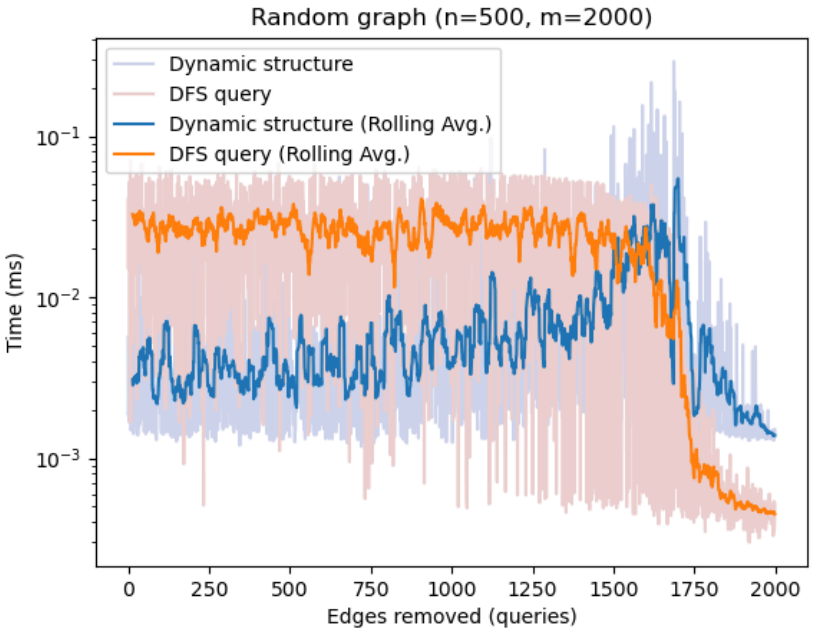


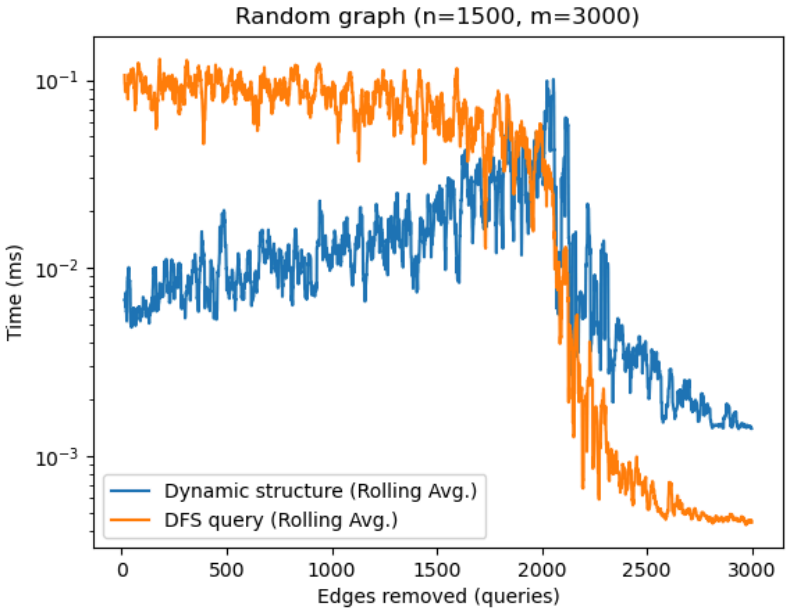

Ring graph (n=16384) (worst case)
Dynamic structure
DFS query
Dynamic structure (Rolling Avg.)
DFS query (Rolling Avg.)
Time (ms)
10^4
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
10^{-3}
0
2500
5000
7500
10000
12500
15000
Edges removed (queries)

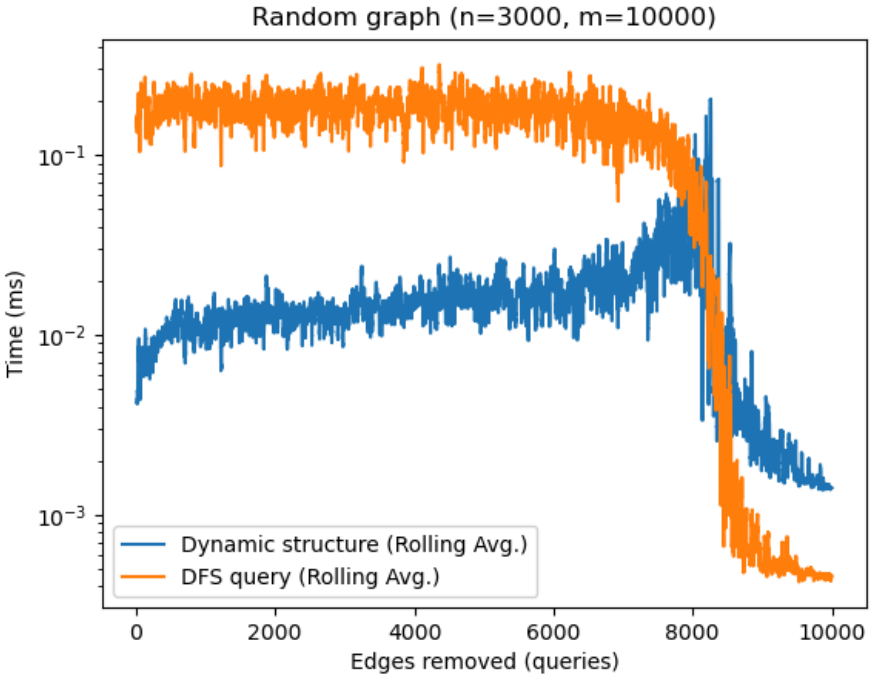
Τυχαία γραφήματα

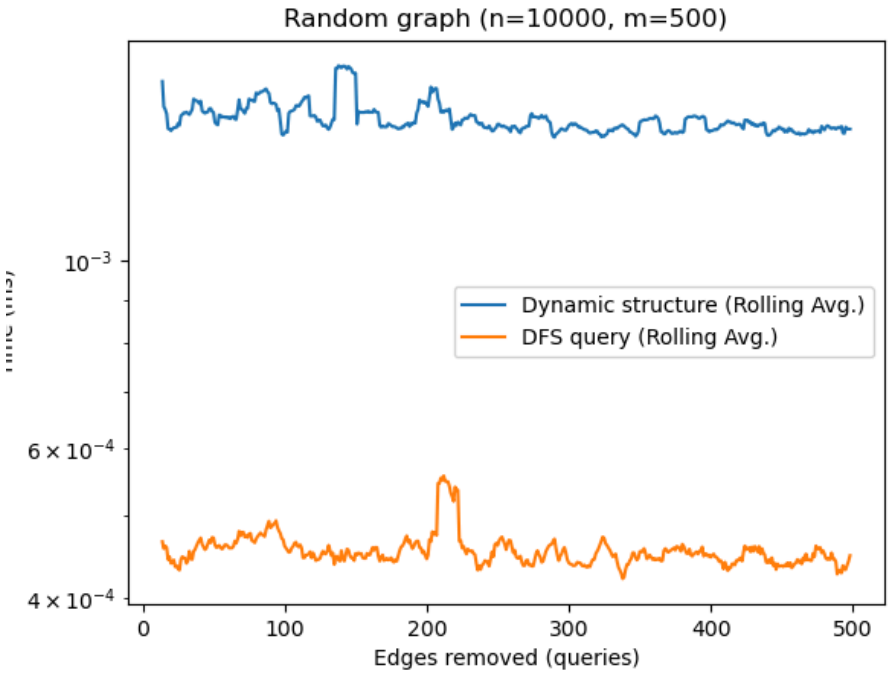
Όπως φαίνεται παρακάτω, λόγω της τυχαίοτητας, η διακύμανση μεταξύ τυχαίων ερωτημάτων είναι μεγάλη. Παρατηρούμε, ωστόσο, πως ο δυναμικός αλγόριθμος υπερτερεί σαφώς αρχικά, όταν το πλήθος των ακμών είναι μεγαλύτερο, και έπειτα ο DFS γίνεται πιο αποδοτικός, καθώς ο δυναμικός συνεχίζει να έχει το κόστος της αναδιοργάνωσης σε κάθε ερώτημα. Αυτό φαίνεται και στην τελευταία περίπτωση, που πρόκειται για αραιό γράφημα.





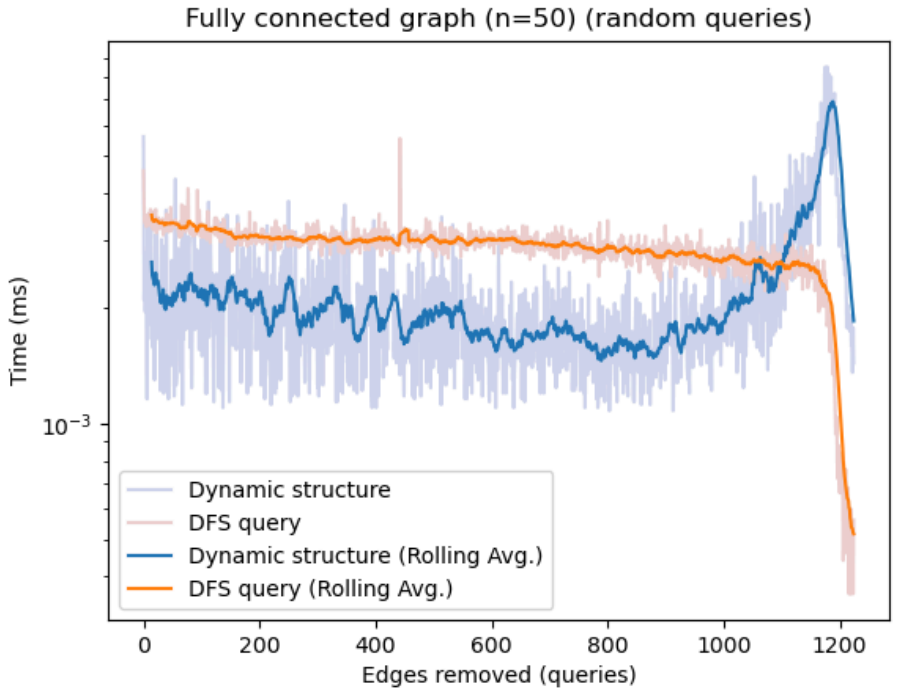

Random graph (n=1500, m=3000)
Time (ms)
10^{-1}
10^{-2}
10^{-3}
Dynamic structure (Rolling Avg.)
DFS query (Rolling Avg.)
0
500
1000
1500
2000
2500
3000
Edges removed (queries)

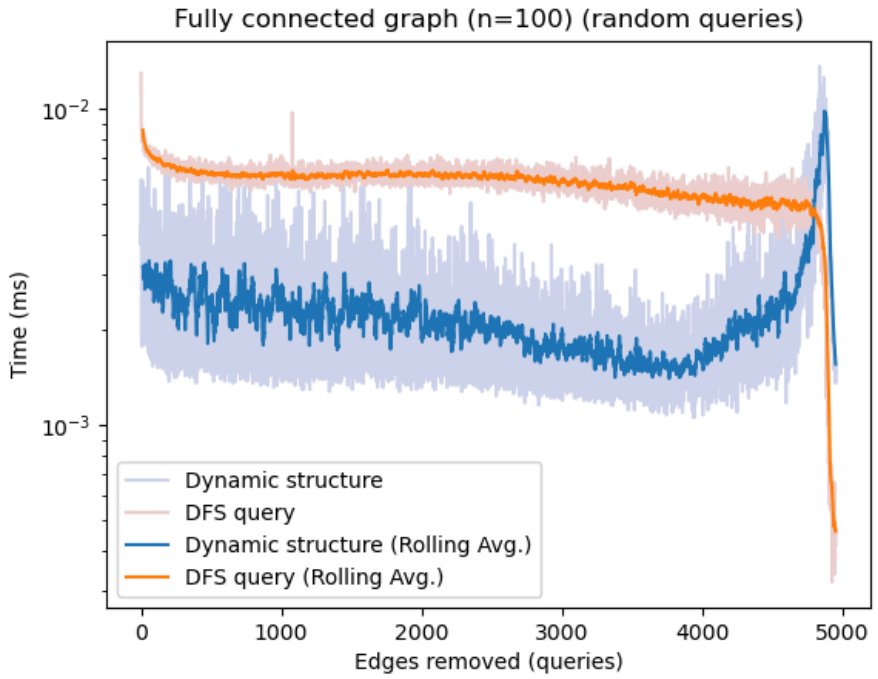

Random graph (n=3000, m=10000)
Time (ms)
10^{-1}
10^{-2}
10^{-3}
Dynamic structure (Rolling Avg.)
DFS query (Rolling Avg.)
0
2000
4000
6000
8000
10000
Edges removed (queries)

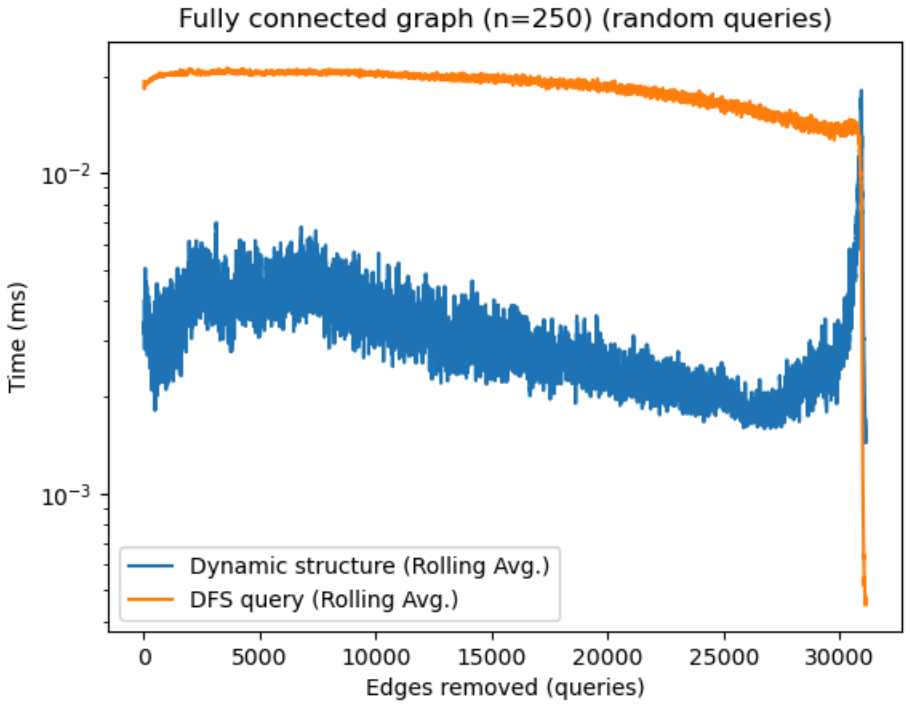

Random graph (n=10000, m=500)
10^{-3}
6×10^{-4}
4×10^{-4}
Dynamic structure (Rolling Avg.)
DFS query (Rolling Avg.)
0
100
200
300
400
500
Edges removed (queries)

Πλήρως συνδεδεμένα γραφήματα

Και εδώ φαίνεται η υπεροχή του δυναμικού αλγορίθμου όσο υπάρχουν πολλές ακμές και άρα λίγες συνεκτικές συνιστώσες. Προς το τέλος των ερωτημάτων, όταν και η αφαίρεση πιθανότητα επιφέρει αλλαγή στις συνεκτικές συνιστώσες, παρατηρείται αύξηση στο χρόνο του δυναμικού αλγορίθμου. Αντίθετα, ο DFS υπερτερεί σημαντικά σε αυτή την περίπτωση.

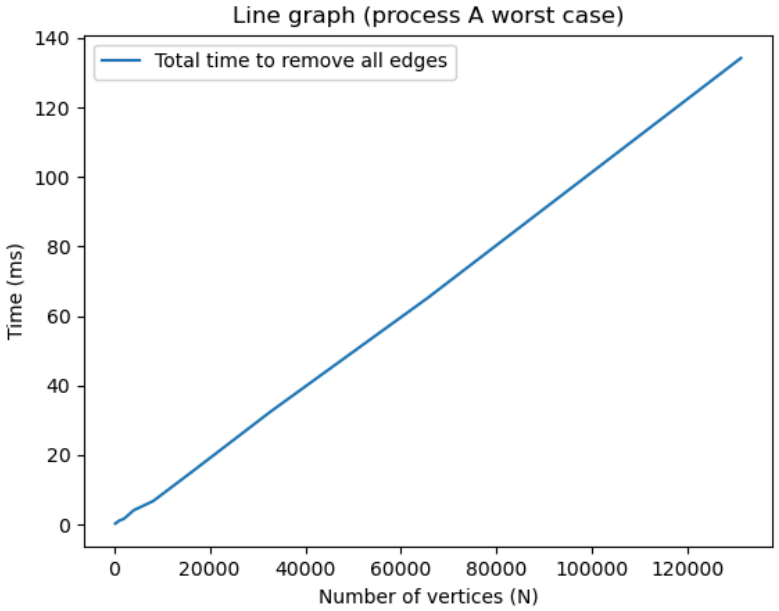


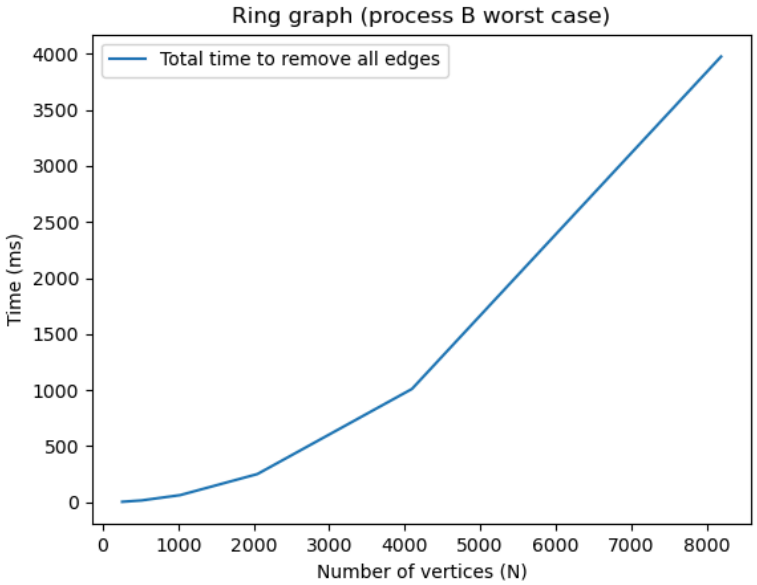

Fully connected graph (n=100) (random queries)
Time (ms)
10^{-2}
10^{-3}
Dynamic structure
DFS query
Dynamic structure (Rolling Avg.)
DFS query (Rolling Avg.)
0
1000
2000
3000
4000
5000
Edges removed (queries)


Fully connected graph (n=250) (random queries)
Time (ms)
10^{-2}
10^{-3}
Dynamic structure (Rolling Avg.)
DFS query (Rolling Avg.)
0
5000
10000
15000
20000
25000
30000
Edges removed (queries)

Πολυπλοκότητα χειρότερης περίπτωσης

Παρακάτω εμφανίζεται η κλιμάκωση των μετρήσεων σε συνάρτηση με το μέγεθος εισόδου. Στοχευμένα επιλέγεται η πρώτη μέτρηση, που αφορά τη χειρότερη περίπτωση διεργασίας Α, να αφορά γράφημα γραμμής, και κάθε σημείο να είναι *σύνολο* του χρόνου που απαιτήθηκε για την αφαίρεση όλων των ακμών σε γράφημα αυτού του μεγέθους. Η δεύτερη μέτρηση καταγράφει το χρόνο για την αφαίρεση της *πρώτης* μόνο ακμής σε γράφημα δακτυλίου, η οποία θα φέρει και το σύνολο του επιμερισμένου κόστους χειρότερης περίπτωσης για τη διεργασία Β.





Από τα παραπάνω επαληθεύεται πράγματι η πολυπλοκότητα της τάξης του O(|E|log|E|) και O(|E||V|). Για εναργέστερη παρουσίαση, παρακάτω παρουσιάζονται οι μετρήσεις:

```
256,512,1024,2048,4096,8192,16384,32768,65536,131072,
0.24134
0.45541
1.038
1.59551
4.06241
6.72982
15.2428
32.5064
65.1488
134.22
```

```
256,512,1024,2048,4096,8192,
4.23692
16.2503
63.5624
251.551
1010.67
3975.1
```

Πράγματι, για την πρώτη περίπτωση, μετά από ορισμένες μικρές τιμές, διπλασιασμός της εισόδου επιφέρει διπλασιασμό και αύξηση κατά έναν παράγοντα, όπως επιτάσσει ασυμπτωτική πολυπλοκότητα O(nlogn). Και για την δεύτερη, διπλασιασμός του μεγέθους της εισόδου επιφέρει τετραπλασιασμό του χρόνου, όπως επιτάσσει πολυπλοκότητα $O(n^2)$, εφόσον |V| - 1 = |E|.