Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Πληροφορικής και Υπολογιστών

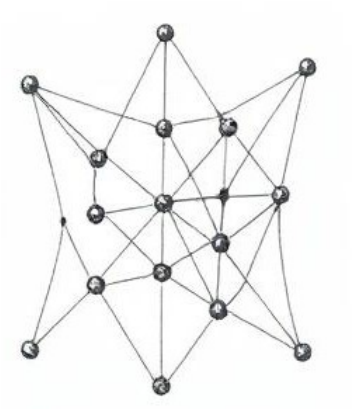
Βέλτιστη Offline Πλήρως Δυναμική 2-Ακμική Συνεκτικότητα

Μελέτη και υλοποίηση

Διπλωματικη Εργασια

του

ΙΩΑΝΝΗ ΡΙΖΑ



Επιβλέπων: Δημήτριος Φωτάκης
Καθηγητής, Ε.Μ.Π.

Αθήνα, Ιούλιος 2025

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Πληροφορικής και Υπολογιστών

Βέλτιστη Offline Πλήρως Δυναμική 2-Ακμική Συνεκτικότητα

Μελέτη και υλοποίηση

Διπλωματικη Εργασια

του

ΙΩΑΝΝΗ ΡΙΖΑ

Επιβλέπων: Δημήτριος Φωτάκης
Καθηγητής, Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την ΞΞ ΞΞΞΞ 2025.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Δημήτριος Φωτάκης	Λουκάς Γεωργιάδης	Αριστείδης Παγουρτζής
Καθηγητής, Ε.Μ.Π.	Καθηγητής, Π.Ι.	Καθηγητής, Ε.Μ.Π.

Αθήνα, Ιούλιος 2025

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Πληροφορικής και Υπολογιστών

ΔΗΛΩΣΗ ΜΗ ΛΟΓΟΚΛΟΠΗΣ ΚΑΙ ΑΝΑΛΗΨΗΣ ΠΡΟΣΩΠΙΚΗΣ ΕΥΘΥΝΗΣ

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ενυπογράφως ότι είμαι αποκλειστικός συγγραφέας της παρούσας Πτυχιακής Εργασίας, για την ολοκλήρωση της οποίας κάθε βοήθεια είναι πλήρως αναγνωρισμένη και αναφέρεται λεπτομερώς στην εργασία αυτή. Έχω αναφέρει πλήρως και με σαφείς αναφορές, όλες τις πηγές χρήσης δεδομένων, απόψεων, θέσεων και προτάσεων, ιδεών και λεκτικών αναφορών, είτε κατά κυριολεξία είτε βάσει επιστημονικής παράφρασης. Αναλαμβάνω την προσωπική και ατομική ευθύνη ότι σε περίπτωση αποτυχίας στην υλοποίηση των ανωτέρω δηλωθέντων στοιχείων, είμαι υπόλογος έναντι λογοκλοπής, γεγονός που σημαίνει αποτυχία στην Πτυχιακή μου Εργασία και κατά συνέπεια αποτυχία απόκτησης του Τίτλου Σπουδών, πέραν των λοιπών συνεπειών του νόμου περί πνευματικών δικαιωμάτων. Δηλώνω, συνεπώς, ότι αυτή η Πτυχιακή Εργασία προετοιμάστηκε και ολοκληρώθηκε από εμένα προσωπικά και αποκλειστικά και ότι, αναλαμβάνω πλήρως όλες τις συνέπειες του νόμου στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής άλλης πνευματικής ιδιοκτησίας.

(Υπογραφή)

.....
Ιωάννης Ριζάς

14 Ιουλίου 2025

Περίληψη

Έστω ένα ακατεύθυντο, αβαρές κι απλό γράφημα $G = (V, E)$, όπου V είναι το σύνολο των κορυφών και E το σύνολο των ακμών. Επίσης, δίνεται μία ακολουθία λειτουργιών (operations) τριών διαφορετικών τύπων: εισαγωγές ακμών, διαγραφές ακμών και ερωτήματα συνεκτικότητας (queries) σχετικά με το αν ένα ζεύγος κορυφών είναι 2-ακμικά συνδεδεμένο (2-edge connected). Δεδομένης μιας ακολουθίας τέτοιων λειτουργιών που θα εφαρμοστούν πάνω στο αρχικό γράφημα G και είναι όλες γνωστές εκ των προτέρων (offline), χρειάζεται να απαντηθούν όλα τα ερωτήματα συνεκτικότητας βέλτιστα.

Στόχος είναι η υλοποίηση ενός αλγοριθμικού μοντέλου που να μπορεί να απαντήσει σε χρόνο $O(t \log n)$, όπου t είναι το μήκος της ακολουθίας λειτουργιών και n το πλήθος κορυφών του γραφήματος. Το μοντέλο εφαρμόζει μια αναδρομική στρατηγική διαίρει-και-βασίλευε πάνω στις λειτουργίες, επιτρέποντας τη συμπύκνωση της πληροφορίας του γραφήματος σε κάθε αναδρομική κλήση. Όταν το μέγεθος των λειτουργιών γίνει τετριμμένο, αυτά πλέον θα μπορούν να απαντηθούν σε σταθερό χρόνο.

Λέξεις Κλειδιά

γράφοι, δυναμικοί αλγόριθμοι, 2-ακμική συνεκτικότητα, συνεκτικότητα ακμών, δυναμική συνεκτικότητα, πλήρως δυναμικά γραφήματα, διαίρει και βασίλευε, offline, αραιοποίηση γραφημάτων, τεχνικές αραιοποίησης

Abstract

Consider an undirected, unweighted, and simple graph denoted as $G = (V, E)$, where V represents the set of vertices and E represents the set of edges. We are given a sequence of operations of three different types: edge insertions, edge deletions, and connectivity queries that ask whether a given pair of vertices are 2-edge connected. Given that all operations are known in advance (offline), our objective is to answer all connectivity queries optimally.

We aim to design and implement an efficient algorithmic framework capable of processing the entire sequence of operations in $O(t \log n)$ time, where t is the number of operations and n is the number of vertices in the original graph. The proposed model follows a recursive divide-and-conquer approach over the operations sequence, allowing for the progressive compression and sparsification of the graph structure at each level of recursion. By reducing the graph size and eliminating irrelevant data, the model guarantees that queries on small instances can be answered in constant time. This approach ensures both theoretical efficiency and practical scalability for offline fully dynamic 2-edge connectivity testing.

Keywords

graphs, dynamic algorithms, 2-edge connectivity, dynamic connectivity, fully dynamic graphs, divide and conquer, offline, graph sparsification, sparsification techniques

στη Λένα, το Λεωνίδα, το Δημήτρη και τη Νίτσα

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Πρόλογος

Η παρούσα διπλωματική εργασία εκπονήθηκε στο πλαίσιο των σπουδών στο Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου για ερευνητική ενασχόληση με το πεδίο των δυναμικών γραφημάτων και αλγορίθμων σε συνεργασία με το Πανεπιστήμιο Ιωαννίνων.

Συγκεκριμένα, μου ανατέθηκε η μελέτη και υλοποίηση μιας θεωρητικής εργασίας πάνω στο offline δισυνεκτικό πρόβλημα, με στόχο την πειραματική επιβεβαίωση της θεωρητικής ανάλυσης ενός σύγχρονου αλγορίθμου για το πρόβλημα αυτό. Η υλοποίησή έγινε σε γλώσσα C++ αποτελώντας και το κύριο και πιο απαιτητικό μέρος του εγχειρήματος αυτής της εργασίας.

0.1 Θεωρητική Εργασία

Η θεωρητική βάση της υλοποίησης βασίστηκε στην εργασία των R. Peng, B. Sandlund & D. Sleator[2, 3], η οποία πραγματεύεται βέλτιστους offline αλγορίθμους για δυναμική δισυνεκτικότητα και τρισυνεκτικότητα ακμών και κορυφών σε ακατεύθυντα γραφήματα, καθώς και για τετραπλή ακμική και κομβική συνεκτικότητα. Η μελέτη αυτή αποτελεί σημείο αναφοράς για αμιγώς offline δυναμικά προβλήματα συνεκτικότητας και προσφέρει αποδοτικές λύσεις με θεμελιώδη θεωρητική τεκμηρίωση.

0.2 Κώδικας

Η υλοποίηση του αλγοριθμικού μοντέλου σε γλώσσα C++ για το βέλτιστο, αμιγώς offline, πλήρως δυναμικό, 2-ακμικό, πρόβλημα συνεκτικότητας είναι διαθέσιμη στο GitHub:

https://github.com/dmiw189/offline-2-edge-connectivity

Κεφάλαιο 1

Εισαγωγή

1.1 Ιστορική Εξέλιξη της Δυναμικής Συνεκτικότητας

Η σημασία των δυναμικών προβλημάτων συνεκτικότητας έγκειται στην ικανότητά τους να εντοπίζουν κρίσιμες συνδέσεις και να μετρούν την *ανθεκτικότητα* του γραφήματος έναντι αλλαγών. Όταν η δομή ενός γράφου μεταβάλλεται δυναμικά — μέσω προσθήκης ή διαγραφής ακμών και κορυφών — το πρόβλημα της συνεκτικότητας γίνεται πολύ πιο απαιτητικό από τη στατική του εκδοχή και απαιτεί εξειδικευμένες αλγοριθμικές τεχνικές.

Οι πρώτες σημαντικές εξελίξεις εντοπίζονται στα μέσα της δεκαετίας του 1990. Το 1994, παρουσιάστηκε αλγόριθμος που επιλύει το πρόβλημα του *Ελάχιστου Συνδετικού Δέντρου* (minimum spanning tree) σε offline περιβάλλον με χρόνο $O(\log n)$ ανά λειτουργία [4]. Το 1996 ακολούθησε η πρώτη σημαντική πρόοδος σε γενικά προβλήματα δυναμικής συνεκτικότητας, με αλγορίθμους για απλή και δισυνεκτικότητα (τόσο ακμική όσο και κομβική) που επιτυγχάνουν χρόνο $\tilde{O}(\sqrt{n})$ ανά πράξη [5, 6, 7, 8, 9].

Το 1997 έγιναν οι πρώτες προσπάθειες για αντιμετώπιση πιο σύνθετων μορφών συνεκτικότητας, όπως η 3-ακμική και η 3-κομβική online συνεκτικότητα, με πολυπλοκότητες $O(n^{2/3})$ και $O(n)$ αντίστοιχα [10].

Το 2001, παρουσιάστηκαν βελτιώσεις που κατέβασαν σημαντικά τις πολυπλοκότητες: $O(\log^2 n)$ για απλή συνεκτικότητα, $O(\log^4 n)$ για ακμική δισυνεκτικότητα και $O(\log^5 n)$ για κομβική δισυνεκτικότητα [11].

Το 2004, οι Patrascu & Demaine απέδειξαν ότι κάθε δυναμικό πρόβλημα συνεκτικότητας — τόσο στην online όσο και στην offline εκδοχή του — υπόκειται σε θεωρητικό κάτω φράγμα χρόνου $\Omega(\log n)$ ανά πράξη [12]. Αυτό έθεσε ένα ορόσημο που καθοδηγεί μέχρι και σήμερα την έρευνα στον τομέα (βλ. Παράρτημα Β').

Το 2015 υπήρξε αναζωπύρωση του ενδιαφέροντος για υβριδικά μοντέλα. Προτάθηκαν αλγόριθμοι που υποστηρίζουν offline ενημερώσεις και online ερωτήματα για απλή και 2-ακμική συνεκτικότητα με χρόνο $O(\log n)$ ανά λειτουργία [13]. Την ίδια χρονιά, προτάθηκε και ένας αλγόριθμος για το online πρόβλημα του Ε.Σ.Δ. με πολυπλοκότητα $O(\log^4 n)$ [14].

Το 2022 παρουσιάστηκε ο έως τώρα ταχύτερος αλγόριθμος για απλή online συνεκτικότητα, με χρόνο $O(\log n(\log\log n)^2)$ ανά λειτουργία [15].

Τέλος, το 2025 προτάθηκε αλγόριθμος για την 2-κομβική online συνεκτικότητα με πολυπλοκότητα $\tilde{O}(\log^2 n)$, ακολουθώντας την ίδια πολυλογαριθμική προσέγγιση με την αντίστοιχη ακμική έκδοση [16]. Το $\tilde{O}(\cdot)$ συμβολίζει ότι αγνοούνται πολυλογαριθμικοί πα-

ράγοντες — συνήθως δυνάμεις του $\log n$ — στην εκτίμηση της πολυπλοκότητας, δηλαδή $\tilde{O}(f(n)) = O(f(n) \cdot \log^k n)$ για κάποια σταθερά $k \geq 0$.

Συνολικά, η έρευνα έχει εστιάσει κυρίως στα online προβλήματα δυναμικής συνεκτικότητας. Παρά την ύπαρξη σχεδόν βέλτιστων θεωρητικών αλγορίθμων, οι απαιτήσεις σε κατανόηση, ανάλυση και αποδοτική υλοποίηση παραμένουν υψηλές, καθιστώντας πολλές από τις προτεινόμενες λύσεις μη πρακτικές σε πραγματικά συστήματα.

1.2 Αντικείμενο της διπλωματικής

Αντίθετα από το online, το offline πρόβλημα προσφέρει ένα πιο εφικτό και υλοποιήσιμο πλαίσιο από τη φύση του ως μια περίπτωση του online προβλήματος, επιτρέποντας την ανάπτυξη απλών και αποτελεσματικών αλγορίθμων με βέλτιστη πολυπλοκότητα.

Η παρούσα εργασία προτείνει έναν αλγόριθμο για το αμιγώς offline πρόβλημα, πλήρως δυναμικής, 2-ακμικής, συνεκτικότητας, ο οποίος συνδυάζει απλότητα και αποδοτικότητα, καθιστώντας τον ιδανικό για πρακτική εφαρμογή. Αντικείμενο μελέτης της αποτελεί το πρόβλημα offline πλήρως δυναμικής ακμικής δισυνεκτικότητας σε γραφήματα, όπως αυτό διατυπώνεται στο θεωρητικό μοντέλο του άρθρου των Peng, Sandlund και Sleator [2, 3].

Το πρόβλημα αφορά τη διαχείριση πλήρως δυναμικών γραφημάτων, όπου δίνεται μια ακολουθία λειτουργιών, που περιέχει όλων των ειδών τις αλλαγές (updates) - τόσο διαγραφές όσο και εισαγωγές ακμών - και ερωτήματα (queries) ακμικής δισυνεκτικότητας, γνωστής εκ των προτέρων, δηλαδή ένα *αμιγώς* offline πρόβλημα, και ένα αρχικό γράφημα, όπου στόχος είναι η απάντηση όλων των ερωτημάτων βέλτιστα. Η υλοποίηση βασίστηκε στην ακριβή αποτύπωση του θεωρητικού μοντέλου, ενώ πραγματοποιήθηκαν και πειραματικές μελέτες για την επιβεβαίωση της αποδοτικότητας του αλγορίθμου και τη σύγκρισή της με τη θεωρητική του πολυπλοκότητα.

1.3 Οργάνωση του Τόμου

Η παρούσα εργασία είναι οργανωμένη σε 5 κεφάλαια, καθένα εκ των οποίων επιτελεί διακριτό ρόλο στη συνολική ανάπτυξη και τεκμηρίωση της μελέτης:

- **Κεφ. 2:** Παρουσιάζεται το θεωρητικό υπόβαθρο που αφορά τα δυναμικά γραφήματα και τα βασικά προβλήματα που τα συνοδεύουν. Δίνονται οι απαραίτητες έννοιες και ορισμοί που είναι θεμελιώδεις για την κατανόηση του αντικειμένου της εργασίας.

- **Κεφ. 3:** Περιγράφεται το αλγοριθμικό μοντέλο που μελετήθηκε και υλοποιήθηκε στο πλαίσιο της εργασίας. Παρουσιάζονται αναλυτικά όλες οι βασικές υπορουτίνες και τα δομικά στοιχεία του μοντέλου.

- **Κεφ. 4:** Μετά τη θεωρητική θεμελίωση, παρουσιάζονται δύο αλγόριθμοι για την επίλυση του προβλήματος της ακμικής δισυνεκτικότητας. Ο πρώτος είναι ο προτεινόμενος αλγόριθμου του θεωρητικού μοντέλου που μελετήθηκε και ο δεύτερος ένας απλός εξαντλητικός αλγόριθμος αναζήτησης για την μετέπειτα σύγκρισή του με τον αρχικό ούτως

ώστε στα επικείμενα πειράματα να επιβεβαιωθεί η εγκυρότητα και η αποδοτικότητα του πρώτου αλγορίθμου.

- **Κεφ. 5:** Παρουσιάζονται τα συγκριτικά πειραματικά αποτελέσματα των δύο αλγορίθμων για διάφορα μεγέθη εισόδου. Η ανάλυση αποσκοπεί στην αξιολόγηση της απόδοσης και της θεωρητικής χρονικής πολυπλοκότητας που επιδιώκεται από το μοντέλο.
- **Κεφ. 6:** Περιγράφεται η συνεισφορά της διπλωματικής εργασίας και προτείνονται πιθανές μελλοντικές επεκτάσεις.

Μέρος I

Θεωρητικό Μέρος

Κεφάλαιο 2

Θεωρητικό υπόβαθρο

Στο κεφάλαιο αυτό παρουσιάζεται το απαραίτητο θεωρητικό υπόβαθρο που σχετίζεται με το αντικείμενο της παρούσας εργασίας. Εισάγονται βασικές έννοιες των δυναμικών γραφημάτων, γίνεται διάκριση μεταξύ offline και online εκδοχών των προβλημάτων, και αναλύονται τα προβλήματα συνδεσιμότητας σε αυτό το πλαίσιο. Επιπλέον, περιγράφονται οι βασικοί αλγόριθμοι και τεχνικές για τον υπολογισμό δισυνεκτικών ακμικών συνιστωσών, τόσο σε στατικά όσο και σε δυναμικά γραφήματα, προετοιμάζοντας το έδαφος για την αλγοριθμική προσέγγιση που ακολουθεί στα επόμενα κεφάλαια.

Για λόγους απλοποίησης, κάθε γράφημα θεωρείται απλό και μη κατευθυνόμενο.

2.1 Δυναμικά Γραφήματα

2.1.1 Στατικά και Δυναμικά Γραφήματα

Τα γραφήματα αποτελούν θεμελιώδες εργαλείο στη θεωρία γράφων, καθώς και σε πλήθος εφαρμογών που σχετίζονται με την επίλυση πρακτικών προβλημάτων. Ένα γράφημα ορίζεται ως ένα ζεύγος $G = (V, E)$ [17, 18], όπου:

- V: το σύνολο των κορυφών,
- $E \subseteq V \times V$: το σύνολο των ακμών, οι οποίες συνδέουν ζεύγη κορυφών.

Τα γραφήματα διακρίνονται σε κατευθυνόμενα ή μη, συνδεδεμένα ή ασύνδετα, και ενδέχεται να φέρουν βάρη (ή κόστη) στις ακμές. Στην περίπτωση αυτή, εισάγεται ένα επιπλέον σύνολο W, που περιέχει τις αντίστοιχες τιμές βαρών.

Αναλόγως της συμπεριφοράς τους στον χρόνο, τα γραφήματα κατατάσσονται σε *στατικά* και *δυναμικά*:

- *Στατικά γραφήματα*: η δομή του γράφου (κορυφές, ακμές και βάρη) παραμένει σταθερή κατά τη διάρκεια της επεξεργασίας.
- *Δυναμικά γραφήματα*: η δομή μπορεί να μεταβάλλεται με την πάροδο του χρόνου, επιτρέποντας εισαγωγές ή διαγραφές κορυφών και ακμών, καθώς και μεταβολές στα βάρη τους.

Τα δυναμικά γραφήματα είναι κατάλληλα για την ανάλυση συστημάτων που εξελίσσονται χρονικά, όπως τα κοινωνικά δίκτυα ή τα δίκτυα τηλεπικοινωνιών [19, 20].

2.1.2 Λειτουργίες

Σε δυναμικά γραφήματα, η κύρια επεξεργασία βασίζεται σε μια ακολουθία *λειτουργιών* (operations), οι οποίες χωρίζονται σε δύο κατηγορίες:

- *Αλλαγές* (updates): Τροποποιήσεις στη δομή του γράφου, όπως εισαγωγές ή διαγραφές ακμών, κορυφών [21], ή αλλαγές βαρών. Συνήθως, οι αλλαγές περιορίζονται στις ακμές για λόγους απλότητας.
- *Ερωτήματα* (queries): Ερωτήσεις που σχετίζονται με τη δομή ή τις ιδιότητες του γράφου, όπως το αν δύο κορυφές είναι 2-ακμικά συνδεδεμένες.

2.1.3 Πλήρως και Μερικώς Δυναμικά Προβλήματα

Ανάλογα με το είδος των αλλαγών που επιτρέπονται, τα δυναμικά προβλήματα διακρίνονται στις εξής κατηγορίες:

- *Πλήρως δυναμικά* (fully dynamic): Επιτρέπονται τόσο εισαγωγές όσο και διαγραφές ακμών.
- *Μερικώς δυναμικά* (partially dynamic): Επιτρέπεται μόνο ένα είδος τροποποίησης. Ειδικότερα:
  - *Αύξοντα* (incremental): Επιτρέπονται μόνο εισαγωγές ακμών [22].
  - *Φθίνοντα* (decremental): Επιτρέπονται μόνο διαγραφές ακμών [23].

Η παρούσα εργασία επικεντρώνεται σε πλήρως δυναμικά προβλήματα, επιτρέποντας κάθε είδους τροποποίηση στο γράφημα.

2.1.4 Online και Offline

Η χρονική γνώση των λειτουργιών επηρεάζει σημαντικά τη φύση του προβλήματος. Διακρίνονται δύο βασικές περιπτώσεις:

- *Online*: Η ακολουθία των λειτουργιών δεν είναι γνωστή εκ των προτέρων και αποκαλύπτεται σταδιακά κατά την εκτέλεση του αλγορίθμου.
- *Offline*: Η πλήρης ακολουθία των λειτουργιών είναι γνωστή πριν την έναρξη της εκτέλεσης και αποτελεί μέρος της εισόδου του προβλήματος.

Συνδυάζοντας αυτές τις έννοιες με την κατηγοριοποίηση των λειτουργιών, προκύπτουν διαφορετικά είδη δυναμικών προβλημάτων:

- *Αμιγώς online* [9, 11, 15, 24, 14, 16]: Τόσο οι αλλαγές όσο και τα ερωτήματα αποκαλύπτονται σειριακά κατά την εκτέλεση. Αποτελεί την πιο γενική και δύσκολη περίπτωση.
- *Offline αλλαγές και Online ερωτήματα* [13]: Η σειρά των αλλαγών είναι γνωστή εξαρχής, ενώ τα ερωτήματα εμφανίζονται σταδιακά.
- *Αμιγώς offline* [2, 3, 4]: Όλες οι λειτουργίες είναι γνωστές εκ των προτέρων. Η πιο απλοποιημένη μορφή του προβλήματος.

Η γνώση μέρους ή όλης της εισόδου επιτρέπει πιο αποδοτικές προεπεξεργασίες και σχεδιασμό εξειδικευμένων και, ταυτόχρονα, πιο απλών δομών δεδομένων. Έτσι, τα offline προβλήματα συχνά αποτελούν χρήσιμο σημείο αναφοράς για την αξιολόγηση ή προσέγγιση των online εκδοχών τους.

Αξιοσημείωτο είναι ότι, όπως δείχνει και η εργασία [3], στα k-ακμικά ή k-κομβικά συνεκτικά προβλήματα, όσο αυξάνεται το k, η απόσταση σε απόδοση μεταξύ των γνωστών offline και online αλγορίθμων αυξάνεται σημαντικά.

Η παρούσα εργασία θα απασχοληθεί με το αμιγώς offline πρόβλημα.

2.2 Συνεκτικότητα

2.2.1 Ακμική και Κομβική Συνεκτικότητα

Μια θεμελιώδες έννοια στη θεωρία γράφων είναι η έννοια της συνεκτικότητας. Ένα γράφημα θεωρείται συνεκτικό όταν κάθε ζεύγος κορυφών συνδέεται μέσω τουλάχιστον ενός μονοπατιού. Συνεπώς, όλες οι κορυφές ενός συνεκτικού γραφήματος είναι προσβάσιμες ανά μεταξύ τους, δηλαδή ότι δεν υπάρχουν απομονωμένες ομάδες κορυφών [17, 18].

Τα προβλήματα συνεκτικότητας χωρίζονται σε δύο βασικές κατηγορίες, την *ακμική* και την *κομβική* συνεκτικότητα. Η *ακμική* συνεκτικότητα αναφέρεται στην ικανότητα του γραφήματος να παραμένει συνεκτικό ακόμα και μετά από την αφαίρεση κάποιων ακμών, ενώ πλήρως αντίστοιχα συμβαίνει για τη *κομβική* συνεκτικότητα.

Πιο σωστά, οι έννοιες της είτε $k-$ακμικής είτε $k-$κομβικής συνεκτικότητας αναφέρονται στην ικανότητα ενός γραφήματος να παραμένει συνεκτικό ακόμα και μετά την αφαίρεση $k-1$ ακμών ή κορυφών αντίστοιχα. Κατά τον ορισμό, ένα γράφημα είναι k-ακμικά συνεκτικό εάν για οποιοδήποτε ζεύγος κορυφών, αν αφαιρεθεί οποιοδήποτε σύνολο ακμών λιγότερων από k, οι κορυφές παραμένουν συνδεδεμένες μεταξύ τους μέσω τουλάχιστον ενός μονοπατιού. Αντίστοιχα και για την k-κομβική συνεκτικότητα, αν αφαιρεθεί οποιοδήποτε σύνολο κορυφών λιγότερων από k, οι κορυφές παραμένουν συνδεδεμένες μεταξύ τους μέσω τουλάχιστον ενός μονοπατιού.

2.2.2 Ερωτήματα Συνεκτικότητας

k-ακμική συνεκτικότητα

Οι πλήρως δυναμικοί αλγόριθμοι γραφημάτων για k-ακμική συνεκτικότητα υποστηρίζουν το τρίπτυχο των λειτουργιών:

- *insert*(u, v): εισαγωγή ακμής (u, v)
- *delete*(u, v): διαγραφή ακμής (u, v)
- *query*(u, v): Αν τα u, v είναι k-ακμικά συνεκτικές

Δύο κορυφές $u, v \in V$ λέγονται *k-ακμικά συνεκτικές* αν και μόνο αν ισχύει:

$$\lambda(u, v) = \min_{\substack{S \subseteq E \\ u \nleftrightarrow v \text{ στο } G-S}} |S| \geq k,$$

όπου $\lambda(u, v)$ δηλώνει τον ελάχιστο αριθμό ακμών των οποίων η αφαίρεση αποσυνδέει την u από τη v, και $G-S$ είναι ο γράφος που προκύπτει μετά την αφαίρεση των ακμών του συνόλου S [25, 26].

Το ερώτημα $Q(u, v)$ επιστρέφει *θετική απάντηση* αν και μόνο αν οι κορυφές u και v είναι k-ακμικά συνεκτικές, δηλαδή $\lambda(u, v) \geq k$.

2-ακμική συνεκτικότητα

Η παρούσα εργασία επικεντρώνεται αποκλειστικά στην περίπτωση των ερωτημάτων *2-ακμικής συνεκτικότητας*. Δηλαδή, για δύο κορυφές $u, v \in V$, λέμε ότι είναι *2-ακμικά συνεκτικές* αν και μόνο αν ισχύει μία από τις ισοδύναμες παρακάτω συνθήκες:

- Υπάρχουν τουλάχιστον δύο *ακμικά ξένα* (edge-disjoint) μονοπάτια που συνδέουν την u με τη v.
- Οι κορυφές u και v ανήκουν στην ίδια *ακμική δισυνεκτική συνιστώσα* του γράφου.

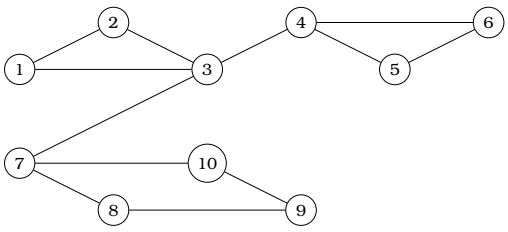
2.3 Υπολογισμός δισυνεκτικών συνιστωσών

Σε αυτό το κεφάλαιο θα αναλυθούν περιληπτικά οι μέχρι σήμερα τρόποι επίλυσης βασικών συνεκτικών προβλημάτων για στατικά και δυναμικά γραφήματα. Σε μια πρώτη ανάλυση, όπως και πριν, οι δισυνεκτικές συνιστώσες χωρίζονται σε δύο είδη προβλημάτων, τις *ακμικές* και τις *κομβικές*, σε πλήρη αντιστοιχία με τον ορισμό προηγουμένως.

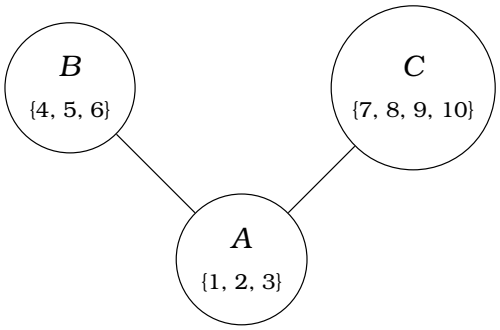
2.3.1 Ακμική δισυνεκτικότητα

Για στατικά γραφήματα, είναι γνωστοί οι γραμμικοί αλγόριθμοι για το πρόβλημα της απλής συνεκτικότητας και της δισυνεκτικότητας ακμών, βασισμένοι στην ιδέα της εύρεσης των *γεφυρών* ενός γραφήματος. *Γέφυρα* χαρακτηρίζεται μία ακμή που αν διαγραφεί αποσυνδέεται ένα μέρος του γραφήματος από το υπόλοιπο. Αν σε ένα μη κατευθυνόμενο γράφημα δεν υπάρχουν γέφυρες, τότε το γράφημα μπορεί να συμπυκνωθεί σε μια *δισυνεκτική ακμική συνιστώσα*, δηλαδή μια συνιστώσα κορυφών όπου κάθε κορυφή του γραφήματος συνδέεται με όλες τις υπόλοιπες με τουλάχιστον δύο διαφορετικά μονοπάτια.

Από την άλλη, αν υπάρχουν γέφυρες, τότε το γράφημα μπορεί νοητικά να μεταφραστεί σε ένα δενδρικό γράφημα και να χωριστεί σε δισυνεκτικές ακμικές συνιστώσες που ενώνονται μεταξύ τους μέσω των γεφυρών. Για να απαντηθεί ένα ερώτημα δισυνεκτικότητας μεταξύ δύο κορυφών σε στατικό γράφημα, αρκεί να υπολογιστεί το δενδρικό γράφημα με την εύρεση των δισυνεκτικών ακμικών συνιστωσών και ο αντιπρόσωπος των δύο κορυφών να είναι κοινός. Δηλαδή να βρίσκονται στην ίδια συνιστώσα.



Σχήμα 2.1: *Παράδειγμα ενός απλού μη-κατευθυνόμενου γραφήματος*



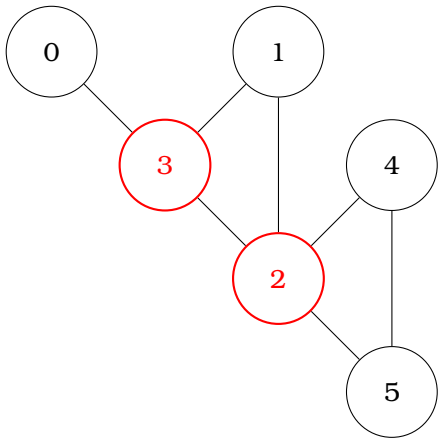
Σχήμα 2.2: *Αναπαράσταση του Παραγόμενου Δενδρικού Γραφήματος*

Η εύρεση του δένδρου γεφυρών, δηλαδή της εύρεσης των γεφυρών και του μετασχηματισμού του γραφήματος στο ισοδύναμό δενδρικό του γίνεται με έναν αλγόριθμο του Tarjan [27, 28], όπου βασίζεται πάνω στον DFS. Ένα παράδειγμα φαίνεται στο Σχήμα 2.1-2.2.

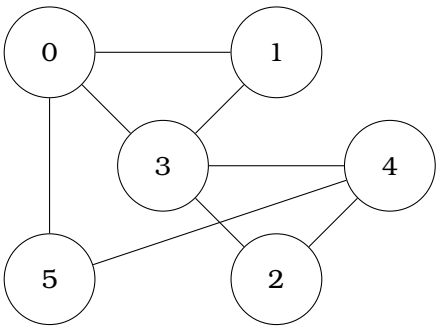
2.3.2 Κομβική δισυνεκτικότητα

Το δισυνεκτικό κομβικό πρόβλημα, από την άλλη, βασίζεται στην ύπαρξη ενός χαρακτηρισμού κορυφών, ως *αρθρώσεις* (articulation points), όπου η αφαίρεσή τους προκαλεί αποσύνδεση του γραφήματος, αντίστοιχη έννοια με τις γέφυρες [17, 18].

Για παράδειγμα, στον συνεκτικό γράφο που παρουσιάζεται στο Σχήμα 2.3, αν αφαιρεθούν οι κορυφές 2 ή 3, ο γράφος θα διασπαστεί σε δύο συνεκτικές συνιστώσες. Επομένως, οι κορυφές αυτές είναι αρθρώσεις και ο γράφος δεν είναι κομβικά δισυνεκτικός. Εάν προστεθούν όμως οι ακμές $(0, 1)$, $(0, 5)$ και $(2, 4)$, ο γράφος καθίσταται κομβικά δισυνεκτικός, αφού δεν υπάρχουν αρθρώσεις, δηλαδή κορυφές που αν αφαιρεθούν το γράφημα σπάσει η συνεκτικότητά του.



Σχήμα 2.3: *Παράδειγμα μη κομβικού δισυνεκτικού γραφήματος. Με κόκκινο φαίνονται οι αρθρώσεις του*



Σχήμα 2.4: *Παράδειγμα κομβικού δισυνεκτικού γραφήματος*

Ο τρόπος επίλυσης αυτού του προβλήματος είναι αρχικά η εύρεση των αρθρώσεων σε ένα γράφημα χρησιμοποιώντας γραμμική αναζήτηση με DFS. Ένας γράφος είναι κομβικά δισυνεκτικός αν και μόνο αν για κάθε κορυφή του γράφου, υπάρχει τουλάχιστον μία πίσω ακμή που βγαίνει από το υποδέντρο που ριζώνει στην κορυφή u προς κάποιον πρόγονο της κορυφής u. Όταν γίνεται αναφορά σε υποδέντρο ριζωμένο στην κορυφή u, εννοούνται όλοι οι απόγονοι της u (εκτός της ίδιας της κορυφής u). Με άλλα λόγια, όταν γίνει επιστροφή από μια κορυφή u, πρέπει να διασφαλιστεί ότι υπάρχει μια πίσω ακμή από κάποιον απόγονο της u προς κάποιον πρόγονο (γονέα ή ανώτερο) της u. Υπάρχει μια εξαίρεση για τον ριζικό κόμβο του δέντρου. Εάν ο ριζικός κόμβος έχει περισσότερους από έναν απογόνους, τότε είναι άρθρωση, αλλιώς όχι [28].

Κεφάλαιο 3

Περιγραφή θέματος

Στο κεφάλαιο αυτό παρουσιάζεται αρχικά η ακριβής περιγραφή του προβλήματος και γίνεται εισαγωγή στις βασικές έννοιες που θα χρησιμοποιηθούν εκτενώς κατά την ανάλυση του αλγοριθμικού μοντέλου επίλυσης που αναπτύσσεται στα επόμενα κεφάλαια της εργασίας.

Αναλύονται διεξοδικά έννοιες όπως ενεργές κορυφές (active vertices), μόνιμες ακμές (permanent edges) και κορυφές αντιπρόσωποι (representative vertices). Στη συνέχεια, παρουσιάζεται η κεντρική ιδέα του αλγορίθμου, καθώς και οι βασικές βοηθητικές υπο-ρουτίνες που τη συνοδεύουν.

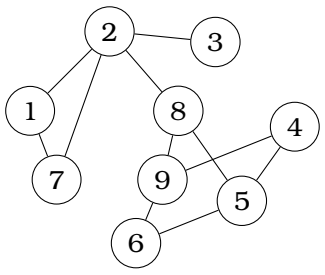
3.1 Ορισμοί

3.1.1 Βέλτιστο Offline Πλήρως Δυναμικό 2-Ακμικής Συνεκτικότητας Πρόβλημα

Έστω ένα αρχικό γράφημα $G(V, E)$ με V το σύνολο των κορυφών του και E το σύνολο των ακμών του, και μία ακολουθία λειτουργιών που είναι γνωστή εκ των προτέρων (offline) και αποτελείται από τις εξής:

- insert(u, v): εισαγωγή ακμής (u, v)
- delete(u, v): διαγραφή ακμής (u, v)
- query(u, v): έλεγχος αν οι u, v είναι 2-ακμικά συνδεδεμένες

Σκοπός του προβλήματος είναι η απάντηση όλων των ερωτημάτων σε χρόνο $O(t \log n)$, σύμφωνα με το γνωστό κάτω φράγμα [12], όπου t είναι το πλήθος των λειτουργιών και n το πλήθος των αρχικών κορυφών του γραφήματος.



1	2	3	4	...	60	61	...
I(5,6)	D(4,3)	Q(2,1)	I(5,4)	...	D(8,9)	I(2,5)	...

Σχήμα 3.1: *Παράδειγμα εισόδου με: 1.αρχικό γράφημα και, 2.ακολουθία λειτουργιών.*

3.1.2 Απλοποίηση Δεδομένων Εισόδου

Για λόγους απλοποίησης, μια ισοδύναμη είσοδος είναι να δίνεται ένα κενό γράφημα $G(V, E = \emptyset)$ με γνωστό το πλήθος των κορυφών του και μια ακολουθία από λειτουργίες στις οποίες συμπεριλαμβάνονται στην αρχή της ακολουθίας και όλες οι ακμές του αρχικού γραφήματος ως εισαγωγές ακμών.

1	2	3	...	11	12	13	14	...	71	72	...
I(1,2)	I(3,2)	I(4,5)	...	I(8,9)	I(5,6)	D(4,3)	Q(2,1)	...	D(8,9)	I(2,5)	...

Σχήμα 3.2: *Παραλλαγή εισόδου: Εκτεταμένη ακολουθία λειτουργιών από το Σχήμα 3.1.*

3.1.3 Θεωρητικό Μοντέλο

Στην παρούσα μελέτη αναλύεται το υπολογιστικό μοντέλο (*framework*) των R. Peng, B. Sandlund & D. Sleator[3], όπως αναφέρθηκε και αρχικά, για την επίλυση offline προβλημάτων συνεκτικότητας, το οποίο έχει εφαρμοστεί επιτυχώς σε διάφορες περιπτώσεις.

Η βασική προσέγγιση στηρίζεται στη στρατηγική *διαίρει και βασίλευε*, όπου η ακολουθία λειτουργιών διασπάται σε μικρότερα, διαχειρίσιμα υπο-προβλήματα. Σε κάθε στάδιο, δημιουργείται ένα ισοδύναμο γράφημα, αξιοποιώντας τεχνικές αραιοποίησης (*sparsification*) [5, 10, 6, 22], οι οποίες μειώνουν σημαντικά την πληροφορία που απαιτείται ανά αναδρομική κλήση. Μέσω αυτής της διαδικασίας επιτυγχάνεται συνολικά το βέλτιστο κάτω φράγμα, όπως αυτό έχει οριστεί από τους Patracscu & Demaine [12] (βλ. Παράρτημα Β'). Δηλαδή, πετυχαίνει χρονική πολυπλοκότητα:

$$O(t \cdot logn)$$

όπου n είναι το πλήθος των αρχικών κορυφών του γραφήματος και t το πλήθος των λειτουργιών.

3.1.4 Ανάλυση Χρονικής Πολυπλοκότητας

Στην ανάλυση του μοντέλου, όπου t αντιπροσωπεύει το μέγεθος της ακολουθίας λειτουργιών, προκύπτει η εξής σχέση αναδρομής για τον χρόνο εκτέλεσης [3]:

$$T(t) = T\left(\frac{t}{2}\right) + O(t) \Rightarrow T(t) = O(t \cdot \log t).$$

Η πολυπλοκότητα αυτή ισχύει υπό την προϋπόθεση ότι, για κάθε υποπρόβλημα μεγέθους t, ότι οι επιμέρους υπορουτίνες:

- για την επεξεργασία και την αραιοποίηση (sparsification) του γραφήματος στην εκάστοτε αναδρομική κλήση, και
- για την επεξεργασία και εξαγωγή πληροφοριών από τις εκάστοτε λειτουργίες

θα έχουν γραμμική πολυπλοκότητα ως προς το μέγεθος της εισόδου τους και το παραγώμενο γράφημα θα έχει μεγέθος $O(t)$.

Ωστόσο,η αποδοτικότητα του αλγορίθμου διαφοροποιείται ανάλογα με τη σχέση του πλήθους λειτουργιών t σε σχέση με το πλήθος των αρχικών κορυφών n.

πολυωνυμικά φραγμένο t ως προς n

Αν t είναι πολυωνυμικά φραγμένο ως προς n ($t = O(n^c)$) για κάποια σταθερά c, τότε ισχύει ότι:

$$T(t) = O(t \cdot \log n), \quad \textit{βέλτιστη}$$

εκθετικά φραγμένο t ως προς n

Αν όχι, δηλαδή t είναι εκθετικό ως προς n, τότε ο χρόνος εκτέλεσης αυξάνεται σε:

$$T(t) = O(t \cdot \log t) >> O(t \cdot logn), \quad \textit{μη βέλτιστη}$$

Σε αυτή την περίπτωση, η πολυπλοκότητα παύει να είναι βέλτιστη, αφού υπάρχουν γνωστοί διαδικτυακοί (online) αλγόριθμοι με χρόνο $O(t \cdot \text{polylog}(n))$, που είναι πρακτικά πιο αποδοτικοί.

Για να διατηρηθεί η αποδοτικότητα σε περιπτώσεις πολύ μεγάλου t, είναι δυνατή η εφαρμογή στρατηγικής ομαδοποίησης: η ακολουθία των λειτουργιών $x_1, x_2, \ldots, x_t$ χωρίζεται σε μπλοκ μεγέθους n^2. Δεδομένου ότι κανένα μπλοκ δεν περιέχει γράφο μεγαλύτερου μεγέθους από $O(n^2)$, κάθε υποπρόβλημα μπορεί να επιλυθεί ανεξάρτητα. Συνολικά, επιτυγχάνεται χρόνος:

$$T(t) = O(t \cdot \log n).$$

Η τεχνική αυτή καθιστά τον αλγόριθμο αποδοτικό και εφαρμόσιμο ακόμη και σε περιπτώσεις με πολύ μεγάλες ακολουθίες λειτουργιών.

Σημείωση

Η παραπάνω στρατηγική δεν υλοποιήθηκε στην παρούσα εργασία και δεν εξετάστηκε πειραματικά η πρακτική της επίδραση. Θεωρήθηκε δεδομένο ότι το πλήθος των λειτουργιών είναι πολυωνυμικά φραγμένο ως προς το μέγεθος του αρχικού γραφήματος.

Κεφάλαιο 4

Βασικές Υπορουτίνες

Στο κεφάλαιο αυτό παρουσιάζονται οι βασικοί αλγόριθμοι που αποτελούν τα δομικά στοιχεία του αλγοριθμικού μοντέλου για την επίλυση του offline ακμικού δισυνεκτικού προβλήματος. Οι υπορουτίνες αυτές απασχολούνται αρχικά με την διαχείρηση της ακολουθίας των λειτουργιών, καθορίζοντας τον τρόπο με τον οποίο θα εκτελεστούν τα βήματα της επίλυσης. Κατά δεύτερον, στη διαχείρηση της ανάλυσης και της επεξεργασίας του δοθέντος γραφήματος, εξασφαλίζοντας την κατάλληλη προετοιμασία της δομής του για την εφαρμογή της λύσης, παράγοντας αυτό που παρακάτω καλείται *ισοδύναμο γράφημα*.

4.1 Επεξεργασία Ακολουθίας Λειτουργιών

4.1.1 Ενεργές Κορυφές

Δοθέντος ενός χρονικού διαστήματος λειτουργιών όλες οι κορυφές που εμπλέκονται έστω και σε μία λειτουργία στο εν λόγω διάστημα ονομάζονται *ενεργές* (active), ενώ οι υπόλοιπες χαρακτηρίζονται ως *ανενεργές* (inactive). Έστω μία ακολουθία από λειτουργίες. Τότε μπορεί να απαντηθεί σε γραμμικό χρόνο για κάθε κορυφή ποιες κορυφές είναι *ενεργές* (active). Πίνακα 4.1.

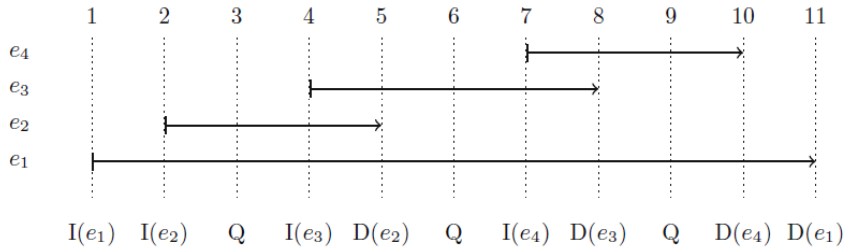
Χρόνος	···	5	6	7	8	9	10	···
Διάστημα	···	$I(1,3)$	$D(3,9)$	$Q(2,3)$	$D(9,2)$	$Q(1,9)$	$I(1,2)$	···

Πίνακας 4.1: *Ενεργές κορυφές: {1,2,3,9}*

Ο πιο απλός και αποδοτικός τρόπος για τον υπολογισμό τους είναι ένα απλό *γραμμικό πέρασμα* στο δοθέν διάστημα των λειτουργιών, όπου όποια κορυφή βρίσκεται σημειώνεται ως ενεργή. Όλες οι υπόλοιπες που δεν χρησιμοποιήθηκαν θεωρούνται ανενεργές για αυτό το διάστημα.

4.1.2 Διάρκεια Ζωής μίας Ακμής

Κάθε ακμή e έχει μία περίοδο ζωής $[I(e), D(e)]$, όπου $I(e)$ είναι η χρονική στιγμή εισαγωγής της και $D(e)$ η χρονική στιγμή διαγραφής της. Για παράδειγμα (βλ. Σχήμα 4.1, μία ακμή e με $I_e = 4$ και $D_e = 8$ έχει περίοδο ζωής το διάστημα $[4, 8]$.



Σχήμα 4.1: *Παράδειγμα διάρκειας ζωής ακμών*

4.1.3 Μόνιμες Ακμές

Ορισμός

Έστω ένα διάστημα $[l, r]$, το οποίο αναφέρεται σε χρονικό υποδιάστημα εντός της συνολικής ακολουθίας λειτουργιών. Οι ακμές κατηγοριοποιούνται ως εξής:

- **Μόνιμες ακμές** (*permanent*): Πρόκειται για ακμές που είναι ενεργές καθ' όλη τη διάρκεια του διαστήματος $[l, r]$, δηλαδή ισχύει:

$$I(e) \leq l \leq r \leq D(e)$$

  Με άλλα λόγια, η ακμή έχει εισαχθεί πριν ή στην αρχή του διαστήματος και δεν έχει αφαιρεθεί μέχρι το τέλος του. Οι ακμές αυτές δεν τροποποιούνται καθόλου εντός του διαστήματος.

- **Μη μόνιμες ακμές** (*non-permanent*): Αυτές είναι οι ακμές των οποίων η περίοδος ζωής επικαλύπτει μερικώς το διάστημα $[l, r]$, αλλά δεν το καλύπτει πλήρως. Δηλαδή, ισχύει ότι:

$$I(e) \in (l, r) \quad \text{ή} \quad D(e) \in (l, r)$$

  Στην περίπτωση αυτή, η ακμή εισάγεται ή αφαιρείται εντός του διαστήματος και επομένως η κατάστασή της τροποποιείται κατά τη διάρκειά του.

Αυτή η ταξινόμηση είναι κρίσιμη για την κατανόηση των ακμών που παραμένουν αμετάβλητες σε ένα χρονικό παράθυρο, έναντι εκείνων που υφίστανται μεταβολές και πιθανώς επηρεάζουν την τοπολογία του γράφου κατά την εξέλιξη των λειτουργιών.

Υπολογισμός

Για τον υπολογισμό τους χρειάζεται πρώτα να γίνει μια προεπεξεργασία (preprocessing) στην ακολουθία λειτουργιών, ώστε για κάθε λειτουργία προσθήκης ή αφαίρεσης ακμής να υπάρχει μαζί και το πότε αυτές αφαιρούνται ή προστίθενται από το ή στο γράφημα, αντίστοιχα. Δηλαδή, χρειάζεται για κάθε ακμή e να χρησιμοποιηθεί η πληροφορία της διάρκειας ζωής της, $[I(e), D(e)]$, και να αποθηκευτεί πάνω στην στιγμή της λειτουργίας της εισαγωγής ή της διαγραφής.

Χρειάζεται επομένως ένας αποδοτικός τρόπος που να παίρνει μία ακολουθία από λειτουργίες, π.χ. $3 : \mathrm{I}(1, 6)$, και να επιστρέφει ένα ζεύγος της λειτουργίας και έναν αριθμό που

αντιπροσωπεύει το άλλο άκρο από το διάστημα ζωής της ακμή;, π.χ. 3 : (I(1, 6), 10). Δίνεται παράδειγμα:

Προεπεξεργασία Λειτουργιών

Υπάρχουν δύο βασικοί τρόποι επίλυσης του προβλήματος με πολυπλοκότητα της τάξης $O(t \log t)$, όπου t το πλήθος των λειτουργιών:

- **Ταξινόμηση:** Πρόκειται για έναν βέλτιστο αλγόριθμο με θεωρητική πολυπλοκότητα $O(t \log t)$, ο οποίος ταξινομεί τις λειτουργίες με βάση συγκεκριμένα κριτήρια, ώστε να διευκολύνει την αποδοτική επεξεργασία τους.
- **Hash Map:** Ένας αποδοτικός αλγόριθμος με γραμμική *αποσβεστική* (amortized) χρονική πολυπλοκότητα $O(t)$, ο οποίος χρησιμοποιεί έναν χάρτη κατακερματισμού (hash map) για να εξασφαλίζει προσπέλαση σε χρόνο $O(1)$ για τις χρονικές στιγμές εισαγωγής και διαγραφής των ακμών.

Αν και οι δύο αλγόριθμοι προεπεξεργασίας για εκθετικά φραγμένα t ως προς n δεν είναι βέλτιστοι, ωστόσο επειδή η εργασία δέχθηκε ως δεδομένο το πολυωνυμικό φράγμα τους, η χρήση και των δύο πρακτικά είναι βέλτιστη. Επίσης, μπορεί η χρήση του hash map να μη καλύπτεται πλήρως θεωρητικά ως προς την πολυπλοκότητα, στην πράξη όμως είναι σημαντικά πιο ευέλικτη λύση και δεν χάνει από θέμα ταχύτητας. Για αυτόν τον λόγο, επιλέχθηκε τελικά για την υλοποίηση του μοντέλου.

⋯	5	6	7	8	9	10	⋯
⋯	$I(1,3)$	$I(3,9)$	$Q(2,3)$	$D(9,3)$	$I(1,2)$	$D(1,3)$	⋯

Πίνακας 4.2: *Παράδειγμα αρχικής ακολουθίας λειτουργιών*

⋯	5	6	7	8	9	10	⋯
⋯	(I(1, 3), 10)	(I(3, 9), 8)	(Q(2, 3), –)	(D(9, 3), 5)	(I(1, 2), 24)	(D(1, 3), 5)	⋯

Πίνακας 4.3: *Επαυξημένη τελική ακολουθία λειτουργιών*

Αλγόριθμος Εύρεσης

Με δεδομένη την επαυξημένη ακολουθία λειτουργιών, για κάθε υποδιάστημα ενός αρχικού διαστήματος μπορεί να βρεθεί σε γραμμικό χρόνο ποιες ακμές είναι μόνιμες. Στο παράδειγμα του Πίνακα 4.3, για το υποδιάστημα $[6, 8]$, η μόνιμη ακμή είναι η $e_{1,3}$, αφού η προσθήκη της είναι πριν το υπο-διάστημα και η αφαίρεση μετά.

Έστω αρχικό χρονικό διάστημα λειτουργιών $[l, r]$. Το υποδιάστημα στο οποίο θα υπολογιστούν οι μόνιμες ακμές, έστω $[i, j]$, για κάθε ακμή e με χρονικό διάστημα ζωής $[I(e), D(e)]$ θεωρείται μόνιμη εάν πληρούνται οι συνθήκες:

$$l \leq I(e) < i \quad \wedge \quad j \leq D(e) < r,$$

Η άμεση εύρεση των μόνιμων ακμών για κάποιο υποδιάστημα είναι δύσκολη και υπολογιστικά κοστοβόρα χωρίς την κατάλληλη προ-επεξεργασία των λειτουργιών. Για το λόγο

αυτό, η υλοποίηση βασίζεται σε τεχνικές προεπεξεργασίας που επιτρέπουν την αποδοτική ανάκτηση των ακμών που πληρούν τις παραπάνω προϋποθέσεις, αξιοποιώντας τις χρονικές στιγμές εισαγωγής $I(e)$ και διαγραφής $D(e)$ της κάθε ακμής.

4.2 Επεξεργασία Γραφήματος (Ισοδύναμο Γράφημα)

Το υποκεφάλαιο αυτό ουσιαστικά παρουσιάζει όλες τις υπορουτίνες που χρησιμοποιούνται εντός της βασικής υπορουτίνας επεξεργασίας του γραφήματος για την παραγωγή του *ισοδύναμου γραφήματος* ως μια αποδοτική μέθοδος αραιοποίησης του γραφήματος την εκάστοτε χρονική στιμγή, με δεδομένη δηλαδή μια ακολουθία λειτουργιών.

4.2.1 Αντιπρόσωποι

Κορυφή αντιπρόσωπος είναι μια κορυφή που αναπαριστά κάποια ακμική δισυνεκτική συνιστώσα του γραφήματος. Λόγω της δυναμικής φύσης του γραφήματος, σε κάθε χρονική *κατάσταση* (state) υπάρχουν κορυφές *αντιπρόσωποι* (representatives), στις οποίες αντιστοιχούν διάφορες κορυφές του αρχικού γραφήματος. Ένα σχόλιο που θα αναλυθεί αργότερα είναι ότι για το πρόβλημα που απασχολείται η εργασία κάθε κορυφή αντιπρόσωπος αντιστοιχεί και σε μια δισυνεκτική συνιστώσα. Οπότε αν δύο ή περισσότερες κορυφές έχουν τον ίδιο αντιπρόσωπο, τότε ανήκουν στην ίδια ακμική δισυνεκτική συνιστώσα, επομένως είναι 2-ακμικά συνδεδεμένες.

Ο *πίνακας αντιπροσώπων* (representatives table) είναι μια δομή που καταγράφει τον αντιπρόσωπο κάθε αρχικής κορυφής του γραφήματος, δηλαδή σε ποια ακμική δισυνεκτική συνιστώσα ανήκει κάθε στιγμή.

Αξίζει να σημειωθεί ότι, αν το γράφημα τροποποιηθεί κατά την εκτέλεση του αλγορίθμου, απαιτείται ανανέωση του πίνακα αντιπροσώπων. Για να διατηρείται έγκυρος ο πίνακας, αρκεί μόνο η ενημέρωση των αντιπροσώπων για τις κορυφές ενδιαφέροντος, δηλαδή μόνο τις ενεργές κορυφές. Η ορθή ανανέωση αυτού εξασφαλίζει τη σωστή αντιστοίχιση των κορυφών στις νέες ακμικές δισυνεκτικές συνιστώσες που ενδέχεται να προκύψουν από τις τροποποιήσεις του γραφήματος.

4.2.2 Ακμικές Δισυνεκτικές Συνιστώσες (Δεντρικό Γράφημα)

Αρχικά, είναι απαραίτητη η περιγραφή του αλγορίθμου που χρησιμοποιείται για την εύρεση των γεφυρών (bridges) σε ένα γράφημα [27]. Βασίζεται στην αναδρομική γραμμική αναζήτηση κατά βάθος (DFS) και εκμεταλλεύεται δύο βασικές έννοιες:

1. Χρόνος Ανακάλυψης (Discovery Time): Ο χρόνος που μια κορυφή ανακαλύπτεται κατά τη διάρκεια της DFS.
2. Χρόνος Χαμηλότερης Επίσκεψης (Low Time): Η μικρότερη τιμή του χρόνου ανακάλυψης που είναι προσβάσιμη από την κορυφή μέσω των γειτονικών της κορυφών.

Έτσι, ο κύριος τρόπος εύρεσης των ακμικών δισυνεκτικών συνιστωσών (2-edge-connected components) γίνεται μέσω της εύρεσης του *δέντρου γεφυρών* (bridge tree), όπου καταγράφεται σε ποια ακμική δισυνεκτική συνιστώσα ανήκει κάθε κορυφή. Η διαδικασία αυτή διασφαλίζει ότι το γράφημα χωρίζεται σε συνιστώσες που δεν περιλαμβάνουν γέφυρες μεταξύ τους,

Αλγοριθμος 4.1: *Bridges*

```
1: function COMPUTE_BRIDGES(graphG)
2:     Initialize visited, disc, low, and parent arrays
3:     for each vertex i in graphG do
4:         if i is not visited then
5:             BRIDGEUTIL(i, visited, parent, low, disc, graphG)
6:         end if
7:     end for
8:     return results
9: end function
10:
11: function BRIDGEUTIL(u, visited, parent, low, disc, graphG)
12:     visited[u] ← True
13:     disc[u] ← Time
14:     low[u] ← Time
15:     Time ← Time + 1
16:     for each vertex v adjacent to u do
17:         if v is not visited then
18:             parent[v] ← u
19:             BRIDGEUTIL(v, visited, parent, low, disc, graphG)
20:             low[u] ← min(low[u], low[v])
21:             if low[v] > disc[u] then
22:                 mark edge(u, v) and edge(v, u) as a bridge
23:             end if
24:         else
25:             if v ≠ parent[u] then
26:                 low[u] ← min(low[u], disc[v])
27:             end if
28:         end if
29:     end for
30: end function
```

κάνοντας την ανάλυση του γραφήματος πιο αποτελεσματική [29]. Ο γραμμικός αλγόριθμος έχει τα εξής βήματα:

- **Εντοπισμός Γεφυρών** (Bridge Identification): Στο πρώτο βήμα, ο αλγόριθμος αναγνωρίζει όλες τις γέφυρες του γράφου.
- **Αρχικοποίηση Συνιστωσών** (Component Initialization): Ο αλγόριθμος αρχίζει με την αναγνώριση των συνιστωσών του γραφήματος, χρησιμοποιώντας DFS. Για κάθε κορυφή που δεν την έχει επισκεφτεί, εκτελεί DFS με έναν μοναδικό αριθμό συνιστώσας.
- **Εκτέλεση DFS** (DFS Execution): Η DFS εξετάζει όλες τις γειτονικές κορυφές που δεν έχουν ήδη επισκεφθεί. Όσες συνδέονται μέσω μη-γεφυρών θεωρούνται μέρος της ίδιας συνιστώσας. Αγνοούνται οι γέφυρες στο γράφημα.
- **Αύξηση Αριθμού Συνιστώσας** (Component Counter Incrementation): Μετά από κάθε DFS ακολουθία, αυξάνεται ο αριθμός συνιστώσας για τις επόμενες κορυφές.

Ο αλγόριθμος είναι βέλτιστος, αφού είναι γραμμικός ως προς το μέγεθος της εισόδου του - $O(N + M)$, όπου N το πλήθος των κορυφών και M το πλήθος των ακμών.

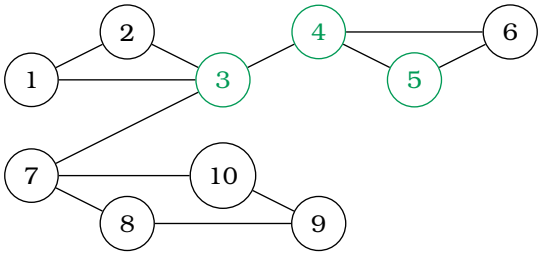
Αλγοριθμος 4.2: *Two Edge Connected Components - Bridge Tree*

```
1: function COMPUTE_BRIDGE_TREE(graphG)
2:     n ← vertices size of the graph
3:     unique_component_number ← 0
4:     Initialize visited arrays, all set to false
5:     //mark the bridges in the graphG
6:     COMPUTE_BRIDGES(graphG)
7:     for each vertex i = 1, 2, 3, . . . , n do
8:         if visited[i] = false then
9:             BRIDGE_TREE_UTIL(i, unique_component_number, visited, graphG)
10:            unique_component_number ← unique_component_number + 1
11:        end if
12:    end for
13: end function
14:
15: function BRIDGE_TREE_UTIL(vertex, visited, component_number, graphG)
16:    component[vertex] ← component_number
17:    visited[vertex] ← true
18:    for each next adjacent to vertex do
19:        if edge(next,vertex) is not a bridge and visited[next] = false then
20:            BRIDGE_TREE_UTIL(next, visited, component_number, graphG)
21:        end if
22:    end for
23: end function
```

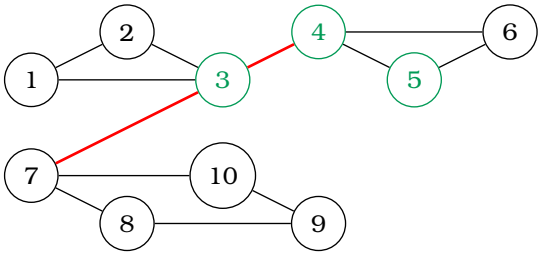
4.2.3 Ακμικές Δισυνεκτικές Συνιστώσες με Ενεργές Κορυφές

Μια παραλλαγή του προβλήματος αφορά κορυφές που είναι ενεργές. Σε αυτή την περίπτωση, μια ακμική δισυνεκτική συνιστώσα χαρακτηρίζεται ως ενεργή αν περιλαμβάνει τουλάχιστον μία ενεργή κορυφή.

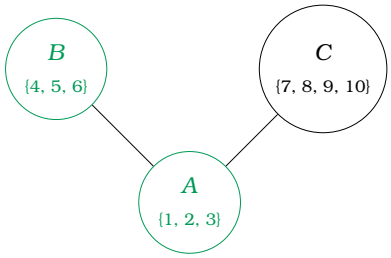
Ο αλγόριθμος είναι επέκταση της DFS και ελέγχει για την ύπαρξη ενεργών κορυφών κατά την κατασκευή των συνιστωσών. Ο αλγόριθμος επιστρέφει ένα δεντρικό γράφημα (bridge tree) με αντιπροσώπους κορυφών (component representatives) για κάθε συνιστώσα, τις γέφυρες ως ακμές και έναν πίνακα αντιπροσώπων (bridge tree component map) για την αντιστοίχηση.



Σχήμα 4.2: *Παράδειγμα αρχικού γραφήματος. Με πράσινο φαίνονται οι ενεργές κορυφές του*



Σχήμα 4.3: *Εύρεση των γεφυρών του γραφήματος*



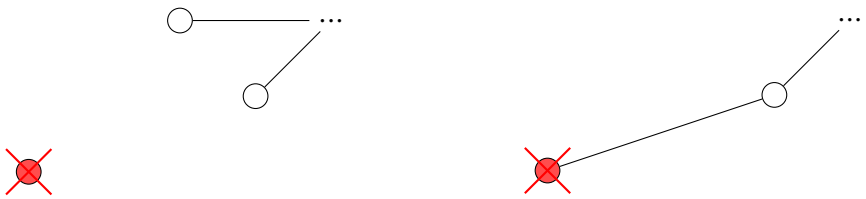
Σχήμα 4.4: *Παραγόμενο δεντρικό ισοδύναμο γράφημα από το αρχικό*

4.2.4 ¨Κλάδεμα¨ Δεντρικού Γραφήματος

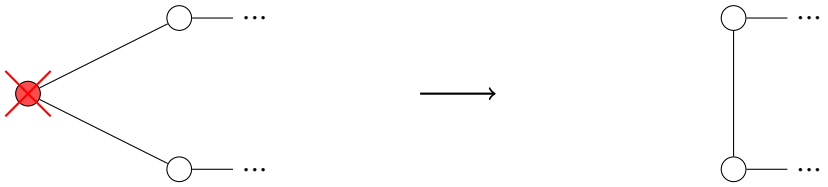
Η διαδικασία της σύντμησης, ή αλλιώς του *κλαδέματος* (pruning), εφαρμόζεται στο παραγόμενο δεντρικό γράφημα με σκοπό τη γραμμική μείωση του μεγέθους του. Αν μία κορυφή είναι ανενεργή και έχει βαθμό ≤ 2, τότε:

1. Αν ο βαθμός είναι 0 ή 1, αφαιρείται (removed)
2. Αν ο βαθμός είναι 2, οι γείτονές της ενώνονται με ακμή (edge contraction) και η ίδια αφαιρείται.

Ο αλγόριθμος είναι βέλτιστος, αφού και αυτός είναι γραμμικός ως προς το μέγεθος της εισόδου του - $O(N + M)$, όπου N το πλήθος των κορυφών και M το πλήθος των ακμών. Ο



Σχήμα 4.5: *Παράδειγμα αφαίρεσης ανενεργής κορυφής βαθμού 0 και 1*



Σχήμα 4.6: *Παράδειγμα αφαίρεσης ανενεργής κορυφής βαθμού 2*

αλγόριθμος επιστρέφει το νέο δεντρικό γράφημα και έναν νέο πίνακα αντιπροσώπων (pruning component map) με μέγεθος $O(t)$, όπου t το πλήθος των ενεργών κορυφών, ή αλλιώς το μέγεθος της ακολουθίας λειτουργιών που δίνεται μαζί με το γράφημα.

Αλγοριθμος 4.3: *Main Prune Function*

```
1:  function PRUNE(graph, activeNodes, oldMap, newMap)
2:      n ← graph.total_bridge_vertices()
3:      Initialize removed array of size n with false
4:      PRUNE_INACTIVE_WITH_DEGREE_1(graph, removed)
5:      PRUNE_INACTIVE_WITH_DEGREE_2(graph, removed)
6:      PRUNE_INACTIVE_WITH_DEGREE_0(graph, removed)
7:      Initialize prunedMap array of size n
8:      UPDATE_GRAPH(graph, prunedMap, removed)
9:      for all u in activeNodes do
10:         newMap[u] ← prunedMap[graph.bridgetreeMap[oldMap[u]]]
11:     end for
12: end function
13: function UPDATE_GRAPH(graph, prunedMap, removed)
14:     newId ← 0
15:     for v = 0 to graph.total_bridge_vertices() − 1 do
16:         if removed[v] = false then
17:             prunedMap[v] ← newId
18:             newId ← newId + 1
19:         else
20:             prunedMap[v] ← −1
21:         end if
22:     end for
23:     Initialize newGraph with newId vertices
24:     for v = 0 to graph.total_bridge_vertices() − 1 do
25:         if removed[v] = false then
26:             for all u in graph.neighbors(v) do
27:                 if removed[u] = false then
28:                     newGraph.add_edge(prunedMap[v], prunedMap[u])
29:                 end if
30:             end for
31:         end if
32:     end for
33:     graph.replace_with(newGraph)
34: end function
```

Αλγοριθμος 4.4: *Prune Inactive Vertices by Degree*

```
1: function PRUNE_INACTIVE_WITH_DEGREE_1(graph, removed)
2:     leaves ← graph.get_leaves()
3:     while leaves not empty do
4:         leaf ← leaves.dequeue()
5:         if leaf is inactive then
6:             for all neighbor in graph.neighbors(leaf) do
7:                 if neighbor not removed then
8:                     parent ← neighbor
9:                     parent.degree ← parent.degree − 1
10:                    if parent.degree = 1 and parent is inactive then
11:                        leaves.enqueue(parent)
12:                    end if
13:                    break
14:                end if
15:            end for
16:            PRUNING(graph, leaf, removed)
17:        end if
18:    end while
19: end function
20:
21: function PRUNE_INACTIVE_WITH_DEGREE_2(graph, removed)
22:    for v = 0 to graph.total_bridge_vertices() − 1 do
23:        if v is inactive and v.degree = 2 then
24:            neighbors ← unremoved neighbors of v
25:            PRUNING(graph, v, removed)
26:            graph.add_edge(neighbors[0], neighbors[1])
27:        end if
28:    end for
29: end function
30:
31: function PRUNE_INACTIVE_WITH_DEGREE_0(graph, removed)
32:    for v = 0 to graph.total_bridge_vertices() − 1 do
33:        if v is inactive and v.degree = 0 then
34:            PRUNING(graph, v, removed)
35:        end if
36:    end for
37: end function
38: function PRUNING(graph, vertex, removed)
39:    removed[vertex] ← true
40:    graph.clear_edges(vertex)
41:    vertex.degree ← vertex.degree − 1
42: end function
```

4.2.5 Ανανέωση του πίνακα αντιπροσώπων

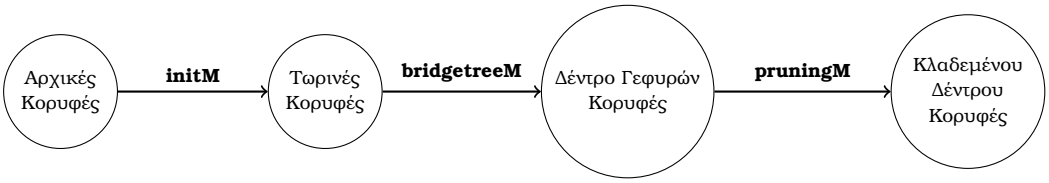
Αφού εκτελεστεί ο αλγόριθμος, χρειάζεται ανανέωση του πίνακα αντιπροσώπων (component map update) για τις ενεργές αρχικές κορυφές:

Αλγοριθμος 4.5: *update components (based only on initial active vertices)*

1: **for** each vertex $v \in V_{\text{active}}^{\text{original}}$ **do**
2: initM[v] ← pruningM[bridgeTreeM[initM[v]]]
3: **end for**

όπου:

- $V_{\text{active}}^{\text{original}}$: το σύνολο των αρχικά ενεργών κορυφών,
- *initM*: η απεικόνιση των αρχικών κορυφών σε κορυφές του τρέχοντος γράφου,
- *bridgeTreeM*: η απεικόνιση των συνιστωσών μετά την κατασκευή του δέντρου γεφυρών,
- *pruningM*: η απεικόνιση των συνιστωσών μετά το κλάδεμα.

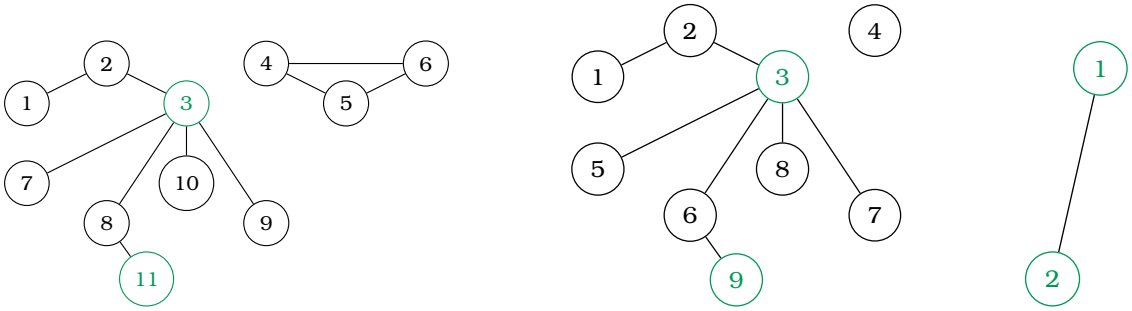


Σχήμα 4.7: *Αναπαράσταση των σχέσεων των διαδικασιών μεταξύ των επεξεργασιών ενός γραφήματος*

4.2.6 Ισοδύναμο Γράφημα

Η διαδικασία δημιουργίας του ισοδύναμου γραφήματος από το αρχικό γράφημα περιγράφεται σε τέσσερα στάδια, ως σύνθεση των υπορουτίνων που περιγράφτηκαν προηγουμένως. Ο αλγόριθμος ακολουθεί τα εξής βήματα:

1. Εντοπισμός και Μαρκάρισμα Γεφυρών
2. Κατασκευή του Δεντρικού Γραφήματος
3. ¨Κλάδεμα¨ Δέντρου Γραφήματος



Σχήμα 4.8: *Παράδειγμα από αρχικό γράφημα σε τελικό κλαδεμένο γράφημα*

Αρχικές	Ενδιάμεσες	Τελικές
3	3	1
11	9	2

Σχήμα 4.9: *Πίνακας Αντιπροσώπων Κορυφών Κλαδεμένου/Ισοδύναμου Γραφήματος*

4. Ανανέωση του πίνακα των αντιπροσώπων

Οπότε, στο τέλος παράγεται ένα αραιομένο γράφημα που καλείται *ισοδύναμο* και συνοδεύεται από έναν ανανεωμένο πίνακα αντιπροσώπων, δεδομένης μιας ακολουθίας λειτουργιών. Στο παράδειγμα του Σχήματος 4.8 παρατηρείται ότι η τελική ισοδύναμη μορφή του γραφήματος είναι πολύ πιο απλή σε σχέση με το αρχικό, εμπεριέχοντας μόνο 2 κορυφές και ένα πίνακα αντιπροσώπων μόνο 2 αρχικών κορυφών (βλ. Σχήμα 4.9), των 3 και 11, όπου αντιστοιχούν στις νέες τρέχοντες κορυφές αντιπροσώπους 1 και 2, αντίστοιχα.

Τέλος, ο αλγόριθμος αυτός καταφέρνει να είναι βέλτιστος, αφού τρέχει σε γραμμικό τρόπο ως προς το μέγεθος του τρέχοντας γραφήματος ($O(N + M)$) και φτιάχνει ένα γράφημα συνοδευόμενο με έναν πίνακα αντιπροσώπων μεγέθους $O(t)$.

Αλγοριθμος 4.6: *compute the equivalent graph*

```
1: input:
2:     graph - The initial graph.
3:     original_active_vertices - List of original active vertices.
4:     current_active_vertices - List of current active vertices.
5:     component_map - Mapping of components.
6: output:
7:     pruned-graph - The sparse equivalent graph.
8:     component_map - Updated component map.

9: procedure COMPUTE_EQUIVALENT_GRAPH
10:     function BRIDGES STAGE
11:         Find all bridges in graph
12:     end function
13:     function 2-EDGE CONNECTED COMPONENTS STAGE
14:         Construct the bridge – tree – graph from the bridges
15:         for each vertex original_vertex in graph do
16:             bridge_tree_mapping[original_vertex] = new_vertex
17:         end for
18:         Find bridge_tree – active_vertices
19:     end function
20:     function PRUNING STAGE
21:         Construct the pruned – graph from bridge-tree-graph and
22:         bridge_tree-active_vertices
23:         for each vertex v in bridge-tree-graph do
24:             pruning_mapping[v] = pruned_vertex
25:         end for
26:     end function
27:     function UPDATE COMPONENTS STAGE
28:         Update component_map
29:         for each vertex v in original_active_vertices do
30:             component_map[v] =
31:                 pruning_mapping[bridge_tree_mapping[component_map[v]]]
32:         end for
33:     end function
34:     return pruned-graph, component_map
35: end procedure
```

Part II

Πρακτικό Μέρος

Κεφάλαιο 5

Ανάλυση και σχεδίαση

Στο κεφάλαιο αυτό παρουσιάζεται η μελέτη που πραγματοποιήθηκε για την υλοποίηση, με κύρια έμφαση στον προτεινόμενο αλγόριθμο επίλυσης του προβλήματος της παρούσας εργασίας. Αρχικά, περιγράφεται η αρχιτεκτονική του αλγορίθμου, ενώ στη συνέχεια παρατίθεται η περιγραφή του σε ψευδοκώδικα. Ακολούθως, παρουσιάζεται και ένας εξαντλητικός αλγόριθμος, ο οποίος υλοποιήθηκε με σκοπό τη συγκριτική πειραματική αξιολόγηση, προκειμένου να ελεγχθεί η εγκυρότητα και η αποδοτικότητα του προτεινόμενου αλγορίθμου.

5.1 Βέλτιστο Αλγοριθμικό Μοντέλο

5.1.1 Δεδομένα Εισόδου

Μετά την ανάλυση του προηγούμενου κεφαλαίου, αναφέρθηκε ότι για την αποδοτικότερη εκτέλεση του αλγορίθμου χρειάζεται να επιτευχθεί πριν από αυτό μια προεπεξεργασία της ακολουθίας λειτουργιών. Οπότε τώρα, τα δεδομένα εισόδου για το αλγοριθμικό μοντέλο είναι τα εξής (βλ. Πίνακα 4.3):

- κενό γράφημα $G(V, E = \emptyset)$,
- ταυτοτικός πίνακας αντιπροσώπων $comp : \forall v \in V : comp[v] := v$,
- επαυξημένη ακολουθία λειτουργιών όπου περιέχει στην αρχή και τις αρχικές ακμές του γραφήματος ως *εισαγωγές ακμών*, με τα γνωστά στοιχεία:
  1. $I(i, j, D_{index} || \infty)$: εισαγωγής ακμής και το πότε έχει γίνει η διαγραφή της,
  2. $D(i, j, I_{index} || -\infty)$: διαγραφής ακμής και το πότε έχει γίνει η εισαγωγή της,
  3. $Q(i, j, -)$: ερωτήματα ακμικής δισυνεκτικότητας

5.1.2 Αρχιτεκτονική Αλγορίθμου

Δεδομένης μιας τέτοιας εισόδου, τα βήματα του αναδρομικού αλγορίθμου με λογική *διαίρει & βασίλευε* είναι αρχικά να χωρίσει το διάστημα λειτουργιών στη μέση και να έχει ένα γράφημα G_h, ένα πίνακα αντιπροσώπων $comp_h$ και μία ακολουθία λειτουγιών $events_h$, για κάθε μισό. Έτσι, σε κάθε μέρος γίνονται τα εξής:

1. Βρίσκει τις αρχικές ενεργές κορυφές στο $events_h$ και τις αντίστοιχες ενεργές κορυφές στο τρέχον γράφημα μέσω του $comp_h$,
2. Βρίσκει τις αρχικές μόνιμες ακμές μέσω του $events_h$ κι έπειτα μέσω του $comp_h$ τις μόνιμες ακμές του τρέχοντος γραφήματος και τις προσθέτει G_h,

3. Υπολογίζει το ισοδύναμο γράφημα $G_h^{"}$ από το G_h και το καινούργιο ανανεωμένο $comp_h^{"}$ από το $comp_h$,
4. καλείται η διαδικασία αυτή πάλι αναδρομικά με εισόδο τώρα:
   - $events_h$: τις μισές λειτουργίες,
   - $G_h^{"}$: το ισοδύναμο γράφημα και
   - $comp_h^{"}$: τον ανανεωμένο πίνακα αντιπροσώπων

έως ότου το μέγεθος του διαστήματος των λειτουργιών να γίνει τετριμμένο, δηλαδή $|events_h| = 1$, τότε αν η λειτουργία είναι ερώτημα $Q(i,j,-)$ η απάντησή του βρίσκεται στον $comp_h$ όπου ελέγχεται αν οι δύο κορυφές έχουν τον ίδιο αντιπρόσωπο, ή αλλιώς, βρίσκονται στην ίδια ακμική δισυνεκτική συνιστώσα:

$$απάντηση := comp[i] == comp[j]$$

Έτσι, ο αλγόριθμος απαντά έγκυρα και βέλτιστα σε χρόνο εκτέλεσης $O(t \cdot logt)$ που συεπάγεται με το βέλτιστο $O(t \cdot logn)$ με πολυωνυμικά φραγμένο t (βλ. 3.1.4).

Αλγοριθμος 5.1: *Two Edge Connectivity*

```
1:  procedure TWO_EDGE_CONNECTIVITY(operations, graph)
2:      events:
3:          compute the augmented operations (graph now is empty)
4:      results:
5:          initialize an empty list for the queries' results
6:      comp:
7:          define an empty array
8:      // initialize the components
9:      for each vertex v in graph do
10:         comp[v] = v
11:     end for
12:     call compute_two_edge_connectivity (
13:         events,
14:         graph,
15:         comp,
16:         results
17:     )
18: end procedure
```

Αλγοριθμος 5.2: *compute two edge connectivity*

```
1: function COMPUTE_TWO_EDGE_CONNECTIVITY(operations, graph, component_map, results)
2:     if |operations| = 1 and operation's type = query then
3:         if component_map[u] = component_map[v] then
4:             res ← true
5:         else
6:             res ← false
7:         end if
8:         Add res to results
9:     else
10:        // split operations into two halves
11:        for each half do
12:            half_graph:
13:                a copy of the graph
14:            half_component_map:
15:                a copy of the component_map
16:            half_operations:
17:                compute half operations
18:            half_active_vertices:
19:                compute active vertices for the half_operations
20:            half_component_active_vertices:
21:                compute component active vertices from half_active_vertices
22:                through component_map
23:            half_permanent_edges:
24:                compute permanent edges
25:            half_component_permanent_edges:
26:                compute component permanent edges from half_permanent_edges
27:                through component_map
28:            add half_component_permanent_edges to graph
29:            call compute_equivalent_graph(
30:                half_graph,
31:                half_active_vertices,
32:                half_component_active_vertices,
33:                half_component_map
34:            )
35:            call compute_two_edge_connectivity (
36:                half_operations,
37:                half_graph,
38:                half_component_map
39:            )
40:        end for
41:    end if
42: end function
```

5.2 Εξαντλητική Αναζήτηση

Ο αλγόριθμος με τον οποίο θα συγκριθεί το μοντέλο το οποίο αξιολογείται σε αυτή την εργασία έχει χρονική πολυπλοκότητα $O(t \cdot (N+M))$. Ο αλγόριθμος παίρνει το αρχικό γράφημα και τη σειρά των λειτουργιών και εκτελεί με τη σειρά κάθε λειτουργία που του δίνεται. Κάθε λειτουργία έχει την εξής πολυπλοκότητα:

- insert(u, v): $O(1)$
- delete(u, v): $O(\min\{N, M\})$
- query(u, v): $O(N + M)$

Τα ερωτήματα μεταξύ των κορυφών βασίζονται στην εύρεση σε γραμμικό χρόνο του δεντρικού γραφήματος όπως περιγράφτηκε νωρίτερα. Πιο συγκεκριμένα:

- **Εντοπισμός Γεφυρών**: Χρησιμοποιείται ένας αλγόριθμος βασισμένος σε DFS για τον εντοπισμό γεφυρών στο αρχικό γράφημα [27].
- **Κατασκευή του Δεντρικού Γραφήματος**: Ένα δεντρικό γράφημα κατασκευάζεται με βάση τις εντοπισμένες γέφυρες. Κάθε κορυφή-αντιπρόσωπος αντιστοιχεί σε μία ακμικά δισυνεκτική συνιστώσα του δοθέντος γραφήματος, και οι ακμές του δέντρου αποτελούνται από τις εντοπισμένες γέφυρες. Η κατασκευή του ακολουθεί την υπάρχουσα υλοποίηση [29].
- **Απαντήσεις Ερωτημάτων (Ακμικής) Δισυνεκτικότητας**: Αν οι δύο κορυφές βρίσκονται στην ίδια ακμική δισυνεκτική συνιστώσα, δηλαδή ο αντιπρόσωπός τους στο καινούργιο δεντρικό γράφημα είναι ο ίδιος, τότε είναι 2-ακμικά συνδεδεμένες..

Η επιλογή του αλγορίθμου εξαντλητικής αναζήτησης εξυπηρετεί διττό σκοπό: αφενός, αποτελεί μέτρο σύγκρισης για την αποδοτικότητα του υπό εξέταση μοντέλου, και αφετέρου λειτουργεί ως μέθοδος επιβεβαίωσης της ορθότητας των παραγόμενων αποτελεσμάτων.

Αλγοριθμος 5.3: *Brute Force Algorithm*

```
1: Input: Initial graph: G, operations sequence: L
2: Output: Query results
3: Initialize: An empty list for the query results: results
4: for each operation in L do
5:     if operation = insert(u, v) then
6:         Insert edge (u, v) into G
7:     else if operation = delete(u, v) then
8:         Delete edge (u, v) from G
9:     else if operation = query(u, v) then
10:        Find bridges in G
11:        Construct the bridge tree
12:        if (u, v) are in the same connected component in the bridge tree then
13:            res ← TRUE
14:        else
15:            res ← FALSE
16:        end if
17:        Add res to results
18:    end if
19: end for
20: return results
```

Κεφάλαιο 6

Πειράματα & Αποτελέσματα

Στο κεφάλαιο αυτό περιγράφονται τα πειράματα που διεξήχθησαν με τους δύο προαναφερθείσες αλγορίθμους στο προηγούμενο κεφάλαιο. Στόχος της πειραματικής διαδικασίας είναι η διερεύνηση της αποδοτικότητας του προτεινόμενου αλγορίθμου συγκρινόμενο με αυτή ενός εξαντλητικού αλγορίθμου.

6.1 Παραγωγή Εισόδου

Για την αξιολόγηση των αλγορίθμων, δημιουργήθηκαν ειδικά γραφήματα και αντίστοιχες ακολουθίες γεγονότων, με στόχο τη σύγκριση τόσο της αποδοτικότητας όσο και της ορθότητας των αποτελεσμάτων.

Η διαδικασία κατασκευής της εισόδου περιλάμβανε τα εξής:

- **Αρχικά (Πυκνά) Γραφήματα:** Δημιουργήθηκαν 7 διαφορετικά απλά γραφήματα χωρίς βρόχους, με αυξανόμενο πλήθος κορυφών και ακμών μέχρι τάξης του 10^4. Τα γραφήματα ικανοποιούν τη συνθήκη $N < M < \frac{N^2}{2}$ ώστε να εξασφαλίζεται η πυκνότητά τους. Από το 6ο γράφημα και μετά, ο αλγόριθμος αναφοράς (brute force) αδυνατεί να ανταποκριθεί χρονικά, σε αντίθεση με τον βελτιστοποιημένο αλγόριθμό μας.

- **Λειτουργίες:** Για κάθε γράφημα παράχθηκαν 5 διαφορετικές ακολουθίες λειτουργιών, των οποίων το μέγεθος αυξάνεται γραμμικά. Συγκεκριμένα, το i-οστό σύνολο περιέχει $i \times V$ γεγονότα, όπου V το πλήθος των κορυφών του αντίστοιχου γραφήματος. Οι ακολουθίες λειτουργιών κατασκευάστηκαν ώστε να περιλαμβάνουν κατά το ήμισυ ερωτήματα σχετικά με τη συνδεσιμότητα του γράφου και κατά το υπόλοιπο ήμισυ μεταβολές στη δομή του, όπως εισαγωγές ή διαγραφές ακμών, ώστε να πετυχαίνεται πυκνότητα ως προς τα ερωτήματα ακμικής δισυνεκτικότητας.

- **Συνδυασμός Δεδομένων:** Κάθε πείραμα αποτελείται από ένα γράφημα και τα 5 αντίστοιχα σύνολα γεγονότων, επιτρέποντας συνεπή και συγκρίσιμη αξιολόγηση μεταξύ των δύο αλγορίθμων.

Η εν λόγω κατασκευή επιτρέπει την εμπεριστατωμένη πειραματική μελέτη, υπό ελεγχόμενες και κλιμακούμενες συνθήκες, των δυνατοτήτων κάθε αλγορίθμου τόσο ως προς την ταχύτητα όσο και ως προς την ακρίβεια των απαντήσεών του.

6.2 Διαγράμματα Αποτελεσμάτων

Στόχος του πειραματικού μέρους είναι η επιβεβαίωση της θεωρητικής ανάλυσης ως προς την υπολογιστική πολυπλοκότητα των δύο αλγορίθμων. Συγκεκριμένα, αναμένεται ότι καθώς αυξάνεται το πλήθος των εκτελούμενων λειτουργιών, η διαφορά στην απόδοσή τους θα γίνεται εντονότερη, αντανακλώντας την ασυμπτωτική διαφορά τους, μεταξύ $O(t \cdot \log n)$ και $O(t \cdot (N + M))$, τόσο θεωρητική όσο και πρακτική (βλ. Παράρτημα Γ').

Στα επόμενα διαγράμματα παρουσιάζονται συγκριτικά οι χρόνοι εκτέλεσης των δύο αλγορίθμων για κάθε διαφορετική περίπτωση εισόδου. Η οπτικοποίηση συνοδεύεται από συγκεντρωτικούς πίνακες με τις αντίστοιχες μετρήσεις χρόνου σε δευτερόλεπτα (*seconds*), επιτρέποντας άμεση και ποσοτική σύγκριση της αποδοτικότητας των συγκριθέντων αλγορίθμων.

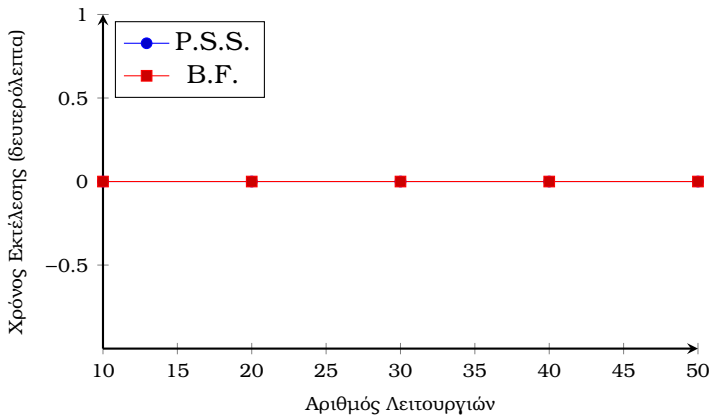
6.2.1 Περιπτώσεις Μικρής Κλίμακας ($N \in \{10, 40, 100\}$)

Αρχικά, στα πρώτα πειραματικά αποτελέσματα που αντιστοιχούν σε μικρού μεγέθους γραφήματα και ακολουθίες λειτουργιών ($N \in \{10, 40, 100\}$), παρατηρείται συγκλίνουσα συμπεριφορά μεταξύ του προτεινόμενου αλγορίθμου **P.S.S.** (Peng, Sandlund & Sleator) και του αλγορίθμου **B.F.** (Brute Force), ή αλλιώς εξαντλητικού αλγορίθμου.

Η παρατήρηση αυτή ενισχύεται τόσο εμπειρικά όσο και θεωρητικά: όταν το μέγεθος του γραφήματος είναι μικρό ($|V| := N \approx O(1)$), οι χρονικές πολυπλοκότητες των δύο προσεγγίσεων συγκλίνουν, καθώς ισχύει $O(t \cdot \log n) = O(t) = O(t \cdot (N + M))$.

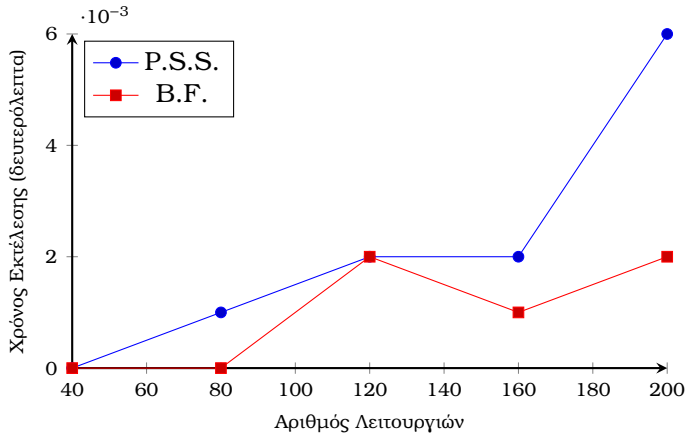
Επιπλέον, καταγράφεται ότι στις περιπτώσεις αυτές ο εξαντλητικός αλγόριθμος εμφανίζει ελαφρώς καλύτερους χρόνους εκτέλεσης συγκριτικά με τον προτεινόμενο (βλ. Σχήμα 6.2). Το φαινόμενο αυτό αποδίδεται πιθανόν στην απλότητα της υλοποίησης του εξαντλητικού αλγορίθμου, η οποία περιλαμβάνει περιορισμένο αριθμό επαναλήψεων και χωρίς σύνθετες δομές. Αντίθετα, ο προτεινόμενος αλγόριθμος βασίζεται σε πιο περίπλοκες αναδρομικές διαδικασίες και επιπρόσθετο λογισμικό φορτίο, το οποίο ενδέχεται να επιβαρύνει τον χρόνο εκτέλεσης σε περιπτώσεις μικρής κλίμακας.

Παρακάτω παρέχονται τα πειραματικά αποτελέσματα σε περιπτώσεις μικρής κλίμακας:



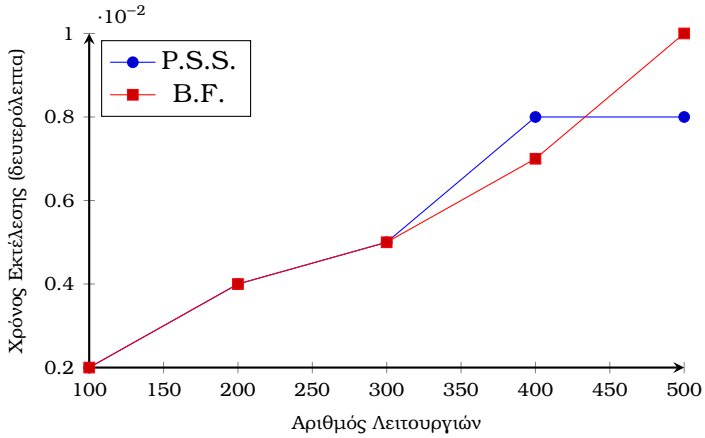
Αρ. Λειτουργιών	P.S.S.	B.F.
10	0	0
20	0	0
30	0	0
40	0	0
50	0	0

Σχήμα 6.1: *Γράφημα 1:* $G(V = 10, E = 19)$



Αρ. Λειτουργιών	P.S.S.	B.F.
40	0	0
80	0.001	0
120	0.002	0.002
160	0.002	0.001
200	0.006	0.002

Σχήμα 6.2: *Γράφημα 2:* $G(V = 40, E = 71)$



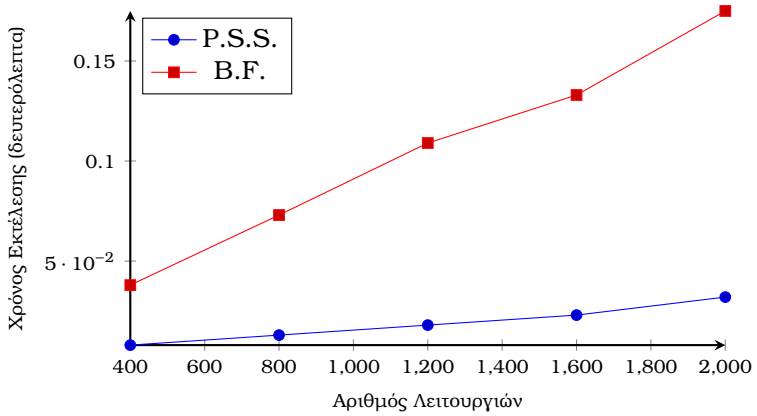
Αρ. Λειτουργιών	P.S.S.	B.F.
100	0.002	0.002
200	0.004	0.004
300	0.005	0.005
400	0.008	0.007
500	0.008	0.01

Σχήμα 6.3: *Γράφημα 3:* $G(V = 100, E = 175)$

6.2.2 Περιπτώσεις Μεσαίας Κλίμακας ($N \in \{400, 1000, 3000\}$)

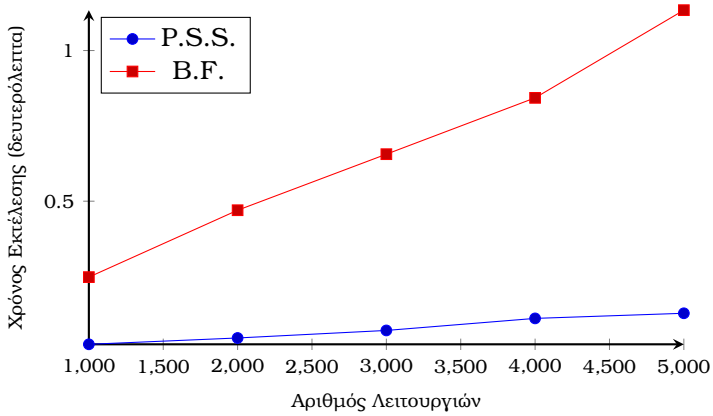
Σε αυτή τη φάση της αξιολόγησης, παρατηρείται πλέον σαφής διαφοροποίηση στον χρόνο εκτέλεσης των δύο αλγορίθμων καθώς αυξάνεται το μέγεθος της εισόδου. Ήδη από το τελευταίο διάγραμμα της παρούσας ενότητας, είναι εμφανές ότι ο εξαντλητικός αλγόριθμος αρχίζει να εμφανίζει σημαντική χρονική υστέρηση, φτάνοντας στα όρια της πρακτικής εφαρμοσιμότητάς του.

Αντιθέτως, ο προτεινόμενος αλγόριθμος εξακολουθεί να παρουσιάζει σταθερή και προβλέψιμη συμπεριφορά, ανταποκρινόμενος ικανοποιητικά στις αυξημένες απαιτήσεις των συγκεκριμένων παραδειγμάτων. Παρακάτω παρέχονται τα πειραματικά αποτελέσματα σε περιπτώσεις μεσαίας κλίμακας:



Αρ. Λειτουργιών	P.S.S.	B.F.
400	0.008	0.038
800	0.013	0.073
1200	0.018	0.109
1600	0.023	0.133
2000	0.032	0.175

Σχήμα 6.4: *Γράφημα 4:* $G(V = 400, E = 811)$

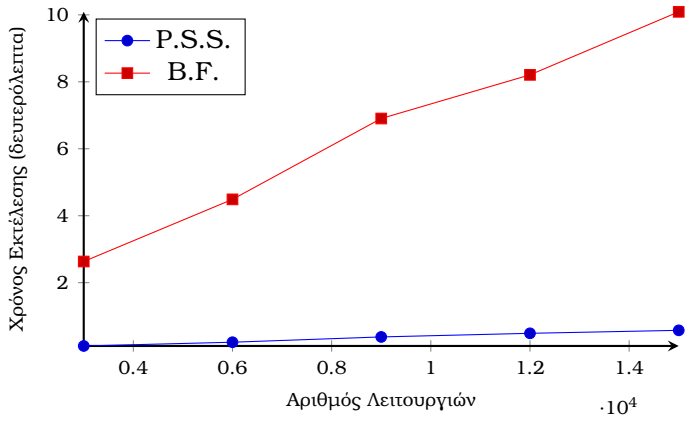


Αρ. Λειτουργιών	P.S.S.	B.F.
1000	0.025	0.248
2000	0.046	0.47
3000	0.071	0.656
4000	0.111	0.843
5000	0.128	1.134

Σχήμα 6.5: *Γράφημα 5:* $G(V = 1000, E = 1479)$

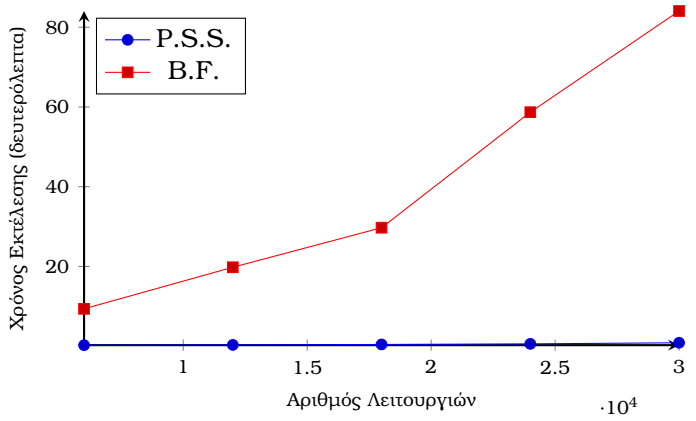
6.2.3 Περιπτώσεις Μεγάλης Κλίμακας ($N \in \{6000, 10000\}$)

Στη τελική αυτή φάση που αρχίζει να αντικατοπτρίζει το μέγεθος δεδομένων πραγματικών προβλημάτων, ο εξαντλητικός αλγόριθμος φαίνεται ότι έχει σταματήσει να είναι επιλογή χρήσης, ενώ ακόμη ο προτεινόμενος αλγόριθμος *P.S.S.* στέκεται αρκετά καλά υπό αυτές τις εισόδους, επιβεβαιώνοντας έτσι την θεωρητική του πολυπλοκότητα και την εργασία των δημιουργών τους. Παρακάτω παρέχονται τα πειραματικά αποτελέσματα σε περιπτώσεις πραγματικής κλίμακας:



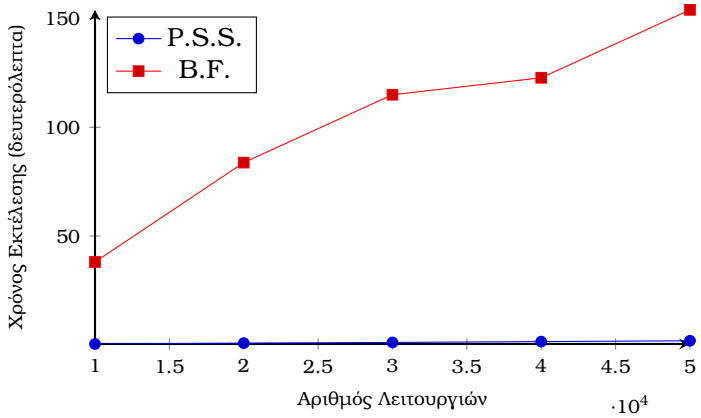
Αρ. Λειτουργιών	P.S.S.	B.F.
3000	0.11	2.633
6000	0.222	4.489
9000	0.38	6.904
12000	0.488	8.208
15000	0.577	10.091

Σχήμα 6.6: *Γράφημα 6:* $G(V = 3000, E = 3034)$



Αρ. Λειτουργιών	P.S.S.	B.F.
6000	0.265	9.377
12000	0.33	19.8
18000	0.441	29.717
24000	0.59	58.688
30000	0.86	84.064

Σχήμα 6.7: *Γράφημα 7:* $G(V = 6000, E = 9819)$



Αρ. Λειτουργιών	P.S.S.	B.F.
10000	0.407	38.09
20000	0.789	83.755
30000	1.113	114.954
40000	1.5	122.731
50000	1.882	153.965

Σχήμα 6.8: *Γράφημα 8:* $G(V = 10000, E = 10482)$

6.3 Συμπεράσματα

Τα πειραματικά αποτελέσματα επιβεβαιώνουν πλήρως τις θεωρητικές προσδοκίες. Σε περιπτώσεις μικρής εισόδου, ο αλγόριθμος εξαντλητικής αναζήτησης εμφανίζει ελαφρώς καλύτερη ή ισοδύναμη επίδοση σε σχέση με το προτεινόμενο μοντέλο, λόγω της απλότητας των λειτουργιών και του περιορισμένου υπολογιστικού φορτίου. Ωστόσο, καθώς το πλήθος των λειτουργιών αυξάνεται, οι χρόνοι εκτέλεσης των δύο μεθόδων αποκλίνουν σημαντικά, με τον προτεινόμενο αλγόριθμο να υπερέχει κατά πολύ σε απόδοση.

Η ραγδαία αυτή απόκλιση επιβεβαιώνει αφενός την ορθότητα της θεωρητικής ανάλυσης κι αφ' ετέρου τη λειτουργική επάρκεια της υλοποίησης, συμφωνώντας και με την εμπειρική σχέση μεταξύ πολυπλοκότητας χρόνου και το μέγεθος της εισόδου (βλ. Πίνακα Γ'.1). Η εν λόγω παρατήρηση ενισχύει τα συμπεράσματα της παρούσας μελέτης και συμφωνεί με τις θεωρητικές εργασίες της βιβλιογραφίας [2, 3] στις οποίες βασίστηκε.

Μέρος III

Επίλογος

Κεφάλαιο 7

Επίλογος

7.1 Μελλοντικές Επεκτάσεις

Το αλγοριθμικό μοντέλο που αναπτύχθηκε στο πλαίσιο της παρούσας διπλωματικής εργασίας μπορεί να αποτελέσει τη βάση για περαιτέρω εξέλιξη και διεύρυνση της μεθοδολογίας σε πολλαπλά επίπεδα. Συγκεκριμένα, οι βασικές κατευθύνσεις μελλοντικής έρευνας και ανάπτυξης περιλαμβάνουν:

1. **Υποστήριξη γραφημάτων με πολλαπλές ακμές (multi-edge graphs):** Με σχετικά μικρές τροποποιήσεις στη φάση προεπεξεργασίας των λειτουργιών (operations preprocessing), το υπάρχον μοντέλο μπορεί να προσαρμοστεί ώστε να διαχειρίζεται γραφήματα με πολλαπλές ακμές, χωρίς να απαιτούνται αλλαγές στους βασικούς αλγοριθμικούς μηχανισμούς.

2. **Πλήρης ολοκλήρωση συμπεριλαμβανομένου t εκθετικά φραγμένου από το n:** Η υπάρχουσα έκδοση του αλγορίθμου μπορεί να επεκταθεί και να ενισχυθεί με στόχο την υλοποίηση όλων των θεωρητικών παραλλαγών που αναλύθηκαν στο αντίστοιχο θεωρητικό μέρος. Επιπλέον, η ενσωμάτωση βελτιωμένων τεχνικών μπορεί να ενισχύσει περαιτέρω την απόδοση, ιδιαίτερα σε περιπτώσεις μεγάλων και πολύπλοκων εισόδων.

3. **Διεύρυνση πειραματικού πλαισίου ελέγχου:** Προτείνεται η ενίσχυση της πειραματικής αξιολόγησης με τη χρήση πιο ποικίλων και μεγάλης κλίμακας συνόλων δεδομένων, καθώς και η ενσωμάτωση εντατικών stress tests και ακραίων περιπτώσεων. Ένα πλήρες και ευέλικτο πλαίσιο ελέγχου θα συμβάλει σημαντικά στην αξιολόγηση της πρακτικής αξιοπιστίας και αποδοτικότητας της μεθόδου.

4. **Επέκταση σε πιο σύνθετα προβλήματα συνεκτικότητας:** Η υφιστάμενη αλγοριθμική προσέγγιση είναι αρκετά γενική ώστε να μπορεί να επεκταθεί σε προβλήματα όπως:

   - 2-κομβική συνεκτικότητα
   - 3-ακμική/κομβική συνεκτικότητα
   - Πιθανή μελλοντική υποστήριξη για 4-ακμική/κομβική συνεκτικότητα

   Το αλγοριθμικό μοντέλο λειτουργεί ως *σκελετός* επίλυσης για όλα τα παραπάνω προβλήματα. Αυτό υποδηλώνει ότι η μόνη διαφοροποίηση που χρειάζεται για να επιλυθεί

κάποιο άλλο πρόβλημα είναι η μόνη αλλαγή της εύρεσης του λεγόμενου *ισοδύναμου γραφήματος*, δηλαδή η αλλαγή της τεχνικής αραιοποίησης που χρησιμοποιείται σε κάθε αναδρομική κλήση.

Παραρτήματα

Παράρτημα Α΄

Ορισμοί, Λήματα & Θεωρήματα

Σε αυτό το κεφάλαιο παρουσιάζονται τα βασικά λήματα και θεωρήματα που θεμελιώνουν τη θεωρητική πολυπλοκότητα του προτεινόμενου αλγορίθμου, ο οποίος αναλύθηκε και υλοποιήθηκε στο πλαίσιο της παρούσας εργασίας. Η θεωρητική τεκμηρίωση βασίζεται στο έργο των P.S.S. [3], όπου περιλαμβάνονται οι πλήρεις αποδείξεις τους. Για λόγους πληρότητας και κατανόησης, παρατίθεται εδώ η ουσιώδης μαθηματική επιχειρηματολογία επί της οποίας στηρίζεται η αποδοτικότητα της προσέγγισης, χωρίς τις αποδείξεις τους.

Α΄.1 Γενικά

Θεώρημα 1.1. *Έστω $G = (V, E)$ μη κατευθυνόμενο γράφημα με $|V| = n$, στο οποίο εφαρμόζεται μία ακολουθία t λειτουργιών της μορφής:*

- Εισαγωγή-Ακμής(u, v),
- Διαγραφή-Ακμής(u, v),
- Ερώτημα(u, v): *αν οι u και v είναι 2-ακμικά ή κομβικά συνεκτικοί, 3-ακμικά ή κομβικά συνεκτικοί στο τρέχον γράφημα*

Τότε, υπάρχει αλγόριθμος που εκτελεί όλες τις t λειτουργίες σε συνολικό χρόνο $O(t \log n)$.

Ορισμός 1.1. *Έστω γράφος $G = (V_G, E_G)$ με υποσύνολο κορυφών $W \subseteq V_G$ και γράφος $H = (V_H, E_H)$ με $W \subseteq V_H$. Λέμε ότι οι G και H είναι **c-ακμικά ισοδύναμοι**, εάν για κάθε διάσπαση (A, B) του W, το μέγεθος της ελάχιστης ακμής-τομής που διαχωρίζει το A από το B είναι το ίδιο στους G και H, όταν αυτό το μέγεθος είναι μικρότερο του c.*

*Ομοίως, λέμε ότι οι G και H είναι **c-κομβικά ισοδύναμοι**, εάν για κάθε διάσπαση (A, B, C) του W με $|C| < c$, το μέγεθος της ελάχιστης κομβικής τομής D που διαχωρίζει το A από το B (όπου $C \subseteq D$, $D \cap A = \emptyset$, $D \cap B = \emptyset$) είναι το ίδιο στους G και H, όταν αυτό το μέγεθος είναι μικρότερο του c.*

Λήμμα 1.1. *Έστω ότι οι γράφοι $G = (V_G, E_G)$ και $H = (V_H, E_H)$ είναι c-ακμικά ή c-κομβικά ισοδύναμοι ως προς το σύνολο κορυφών W. Έστω E_W οποιοδήποτε σύνολο ακμών μεταξύ κορυφών του W. Τότε οι επεκτάσεις $G' = (V_G, E_G \cup E_W)$ και $H' = (V_H, E_H \cup E_W)$ παραμένουν c-ακμικά ή c-κομβικά ισοδύναμες, αντιστοίχως.*

Λήμμα 1.2. *Έστω γράφος G με m ακμές και σύνολο κορυφών W μεγέθους k. Αν υπάρχει αλγόριθμος χρόνου $O(m)$ που κατασκευάζει γράφο H μεγέθους $O(k)$ τέτοιο ώστε H και G να είναι c-ακμικά ή c-κομβικά ισοδύναμοι ως προς το W, τότε υπάρχει αλγόριθμος που, για κάθε*

ακολουθία λειτουργιών $x_1, \ldots, x_t$ (εισαγωγές/διαγραφές ακμών και ερωτήματα συνεκτικότητας), απαντά σε όλα τα c-ακμικά ή c-κομβικά ερωτήματα συνεκτικότητας σε συνολικό χρόνο:

$$O(t \log n)$$

.

Α΄.2 Ακμική Δισυνεκτικότητα (2-edge connectivity)

Λήμμα 1.3. *Έστω S μια 2-ακμικά συνεκτική συνιστώσα ενός γράφου G. Αν συρρικνώσουμε όλες τις κορυφές του S σε μία κορυφή s και αναπροσαρμόσουμε ανάλογα τα άκρα των ακμών, προκύπτει γράφος H που είναι 2-ακμικά ισοδύναμος με τον G.*

Λήμμα 1.4. *Αν ο γράφος G είναι δέντρο, τότε οι παρακάτω δύο λειτουργίες παράγουν γράφους H που είναι 2-ακμικά ισοδύναμοι με τον G:*

- *Αφαίρεση μη ενεργού φύλλου (βαθμός 1).*
- *Αφαίρεση μη ενεργής κορυφής βαθμού 2 και προσθήκη ακμής μεταξύ των δύο γειτόνων της.*

Λήμμα 1.5. *Δεδομένου ενός γράφου G με m ακμές και k ενεργές κορυφές W, μπορεί να κατασκευαστεί ένας γράφος H, 2-ακμικά ισοδύναμος με τον G και μεγέθους $O(k)$, σε χρόνο $O(m)$.*

Παράρτημα Β΄

Θεμελιώδες Κάτω Φράγμα για Δυναμική Συνεκτικότητα

Στο πρόβλημα της *δυναμικής συνεκτικότητας*, με ένα μη κατευθυνόμενο γράφο $G(V, E)$ που υφίσταται δυναμικές τροποποιήσεις (εισαγωγές και διαγραφές ακμών) απαντώνται συνεκτικά ερωτήματα:

¨*είναι οι κορυφές u και v συνεκτικές*¨.

Ένα από τα σημαντικότερα θεωρητικά αποτελέσματα στον τομέα είναι το *κάτω φράγμα* που απέδειξαν οι Patracscu & Demaine, το οποίο θέτει έναν θεμελιώδη περιορισμό στο πόσο αποδοτικά μπορούν να γίνουν τέτοιοι δυναμικοί αλγόριθμοι συνεκτικότητας.

Θεώρημα 2.2 (Κάτω Φράγμα - $\Omega(logn)$ [12, 30])**.** *Κάθε αλγόριθμος που επιλύει το πρόβλημα της δυναμικής συνεκτικότητας, με πράξεις αλλαγών και ερωτημάτων, απαιτεί χρόνο τουλάχιστον*

$$\Omega(\log n) - \textit{ανά πράξη}$$

Το αποτέλεσμα αυτό ουσιαστικά δηλώνει ότι κανένας αλγόριθμος δεν μπορεί να απαντά σε ερωτήματα συνεκτικότητας γρηγορότερα από αυτό το κάτω φράγμα, αν θέλει ταυτόχρονα να υποστηρίζει δυναμικές τροποποιήσεις. Και αυτό αποδείχτηκε τόσο για το online όσο και για το offline πρόβλημα.

Παράρτημα Γ΄

Πολυπλοκότητα Χρόνου σε Σχέση με το Μέγεθος Εισόδου

Η εκτίμηση της αποδοτικότητας ενός αλγορίθμου ως προς το μέγεθος εισόδου μπορεί να πραγματοποιηθεί βάσει της αναμενόμενης πολυπλοκότητας χρόνου. Ο ακόλουθος πίνακας παρουσιάζει τις πολυπλοκότητες χρόνου που θεωρούνται αποδεκτές υπό χρονικό περιορισμό *1 δευτερολέπτο*, βασισμένος σε υπάρχουσα βιβλιογραφία [18, 1]:

Μέγεθος εισόδου n	**Αποδεκτή πολυπλοκότητα χρόνου**
$n \leq 10$	$O(n!)$
$n \leq 20$	$O(2^n)$
$n \leq 500$	$O(n^3)$
$n \leq 5\,000$	$O(n^2)$
$n \leq 10^6$	$O(n \log n)$ ή $O(n)$
Πολύ μεγάλο n	$O(1)$ ή $O(\log n)$

Πίνακας Γ΄.1: *Σχέση πολυπλοκότητας χρόνου με το μέγεθος εισόδου [1]*

Ενδεικτικά Παραδείγματα

- Για $n = 10^5$, ένας αλγόριθμος με πολυπλοκότητα $O(n^2)$ θα εκτελέσει περίπου 10^{10} πράξεις, αριθμός απαγορευτικός για χρονικό όριο ενός δευτερολέπτου.
- Στην περίπτωση πολυπλοκότητας $O(n \log n)$ και $n = 10^5$, το πλήθος των πράξεων εκτιμάται σε $10^5 \cdot \log_2(10^5) \approx 10^5 \cdot 17 \approx 1.7 \cdot 10^6$, ποσότητα που θεωρείται αποδεκτή.

Οι παραπάνω εκτιμήσεις συμβάλλουν στον έγκαιρο αποκλεισμό μη αποδοτικών στρατηγικών. Ενδεικτικά, για $n = 20$, η χρήση εξαντλητικής αναζήτησης με πολυπλοκότητα $O(2^n)$ δύναται να είναι αποδεκτή. Αντιθέτως, για $n = 10^6$, ενδείκνυνται μόνον αλγόριθμοι γραμμικής ή σχεδόν γραμμικής πολυπλοκότητας.

Επίδραση Σταθερών Παραγόντων

Η πολυπλοκότητα χρόνου παρέχει χρήσιμες εκτιμήσεις, χωρίς όμως να λαμβάνει υπόψη τους σταθερούς παράγοντες. Συγκεκριμένα:

- Για $n = 10^6$, ο Αλγόριθμος Α εκτελεί $100n$ πράξεις $\rightarrow 10^8$ συνολικές πράξεις
- Ο Αλγόριθμος Β εκτελεί $10n \log n$ πράξεις $\rightarrow$ περίπου $1.7 \cdot 10^7$ πράξεις

Παρά τη θεωρητικά χειρότερη ασυμπτωτική πολυπλοκότητα, ο δεύτερος αλγόριθμος ενδέχεται να είναι ταχύτερος στην πράξη λόγω μικρότερου συντελεστή.

Παράρτημα Δ΄

Παραδείγματα Βιβλιογραφικών Αναφορών

Τύπος βιβλιογραφικής πηγής	Αναφορά
Άρθρο σε επιστημονικό περιοδικό	[5, 6, 11, 28]
Τεχνική αναφορά / Επιστημονική αναφορά (misc)	[3, 12, 9, 15]
Ιστοσελίδα / Online υλικό (misc)	[27, 29]

Βιβλιογραφία

[1] Antti Laaksonen. *Competitive Programmer's Handbook*. 2018. https://cses.fi/book.html.

[2] Richard Peng, Bryce Sandlund και Daniel D. Sleator. *Offline Dynamic Higher Connectivity (v1)*, 2017. http://arxiv.org/abs/1708.03812v1.

[3] Richard Peng, Bryce Sandlund και Daniel D. Sleator. *Optimal Offline Dynamic 2,3-Edge/Vertex Connectivity (v2)*. *Algorithms and Data Structures - 16th International Symposium, WADS 2019, Edmonton, AB, Canada, August 5-7, 2019, Proceedings*, τόμος 11632 στο *Lecture Notes in Computer Science*, σελίδες 602–616. Springer, 2019. https://arxiv.org/abs/1708.03812v2.

[4] David Eppsteinet al. *Offline algorithms for dynamic minimum spanning tree problems*. *J. Algorithms*, 17(2):237–250, 1994.

[5] David Eppstein, Zvi Galil, Giuseppe F. Italiano και Thomas H. Spencer. *Separator Based Sparsification: I. Planarity Testing and Minimum Spanning Trees*. *J. Comput. Syst. Sci.*, 52(1):3–27, 1996. https://sciencedirect.com/science/article/pii/S0022000096900096.

[6] David Eppstein, Zvi Galil, Giuseppe F. Italiano και Thomas H. Spencer. *Separator-Based Sparsification II: Edge and Vertex Connectivity*. *SIAM J. Comput.*, 28(1):341–381, 1998. https://epubs.siam.org/doi/10.1137/S0097539793244356.

[7] Monika Rauch Henzinger. *Fully dynamic biconnectivity in graphs*. *Algorithmica*, 13(6):503–538, 1995.

[8] Monika Rauch Henzinger. *Improved data structures for fully dynamic biconnectivity*. *SIAM Journal on Computing*, 29(6):1761–1815, 2000.

[9] Kathrin Hanauer, Monika Henzinger και Christian Schulz. *Recent Advances in Fully Dynamic Graph Algorithms*, 2022. https://arxiv.org/abs/2102.11169.

[10] David Eppstein, Zvi Galil, Giuseppe F Italiano και Amnon Nissenzweig. *Sparsification—a technique for speeding up dynamic graph algorithms*. *J. ACM*, 44(5):669–696, 1997.

[11] Jacob Holm, Kristiande Lichtenberg και Mikkel Thorup. *Poly-Logarithmic Deterministic Fully Dynamic Algorithms for Connectivity, Minimum Spanning Tree, 2-Edge, and Biconnectivity*. *SIAM J. Comput.*, 31(3):720–740, 2001. https://dl.acm.org/doi/10.1145/502090.502095.

[12] Patracscu & Demaine. *Lower Bounds for Dynamic Connectivity*, 2004. https://people.csail.mit.edu/mip/docs/enc-algs/connect.pdf.

[13] Jakub Karczmarz, Adam & Łącki. *Fast and Simple Connectivity in Graph Timelines*. *Proceedings of the 14th International Symposium on Algorithms and Data Structures (WADS 2015)*, τόμος 9214 στο *Lecture Notes in Computer Science*, σελίδες 458–469. Springer, Cham, 2015. https://doi.org/10.1145/502090.502095.

[14] J. Holm, E. Rotenberg και C. Wulff-Nilsen. *Faster Fully-Dynamic Minimum Spanning Forest*. *Proceedings of the 23rd Annual European Symposium on Algorithms (ESA 2015)*Nikhil Bansal και Irene Finocchi, επιμελητές, τόμος 9294 στο *Lecture Notes in Computer Science*, σελίδες 742–753. Springer, Heidelberg, 2015. https://doi.org/10.1007/978-3-662-48350-3_62.

[15] Shang En Huang, Dawei Huang, Tsvi Kopelowitz και Seth Pettie. *Fully Dynamic Connectivity in $O(\log n(\log\log n)^2)$ Amortized Expected Time*, 2017. https://epubs.siam.org/doi/10.1137/1.9781611974782.32.

[16] Jacob Holm, Wojciech Nadara, Eva Rotenberg και Marek Sokołowski. *Fully dynamic biconnectivity in $\tilde{O}(\log^2 n)$ time*, 2025. https://arxiv.org/abs/2503.21733.

[17] Douglas B. West. *Introduction to Graph Theory*. Prentice Hall, 2νδη έκδοση, 2001.

[18] Thomas H. Cormen, Charles E. Leiserson και Ronald L. Rivest & Clifford Stein. *Introduction to Algorithms*. MIT Press, 2009.

[19] Joakim Skårding, Bogdan Gabrys και Katarzyna Musiał. *Foundations and Modelling of Dynamic Networks using Dynamic Graph Neural Networks: A Survey*. *arXiv preprint arXiv:2005.07496*, 2020. https://arxiv.org/abs/2005.07496.

[20] Farzad V. Farahani, Waldemar Karwowski και Nichole R. Lighthall. *Application of Graph Theory for Identifying Connectivity Patterns in Human Brain Networks: A Systematic Review*. *Frontiers in Neuroscience*, 13:585, 2019. https://www.frontiersin.org/articles/10.3389/fnins.2019.00585/full.

[21] Xuan Zhanget al. *Social Network Analysis and Churn Prediction in Telecommunications Using Graph Theory*. *Entropy*, 22(7):753, 2020. https://www.mdpi.com/1099-4300/22/7/753.

[22] Camil Demetrescu, David Eppstein, Zvi Galil και Giuseppe F. Italiano. *Dynamic Graph Algorithms*. *Algorithms and Theory of Computation Handbook: General Concepts and Techniques*M. J. Atallah και Marina Blanton, επιμελητές, κεφάλαιο 9, σελίδες 9–1–9–28. CRC Press, 2η έκδοση, 2009. https://dl.acm.org/doi/pdf/10.5555/1882757.1882766.

[23] Mikkel Thorup. *Decremental Dynamic Connectivity*. *Journal of Algorithms*, 33(2):229–243, 1999. https://doi.org/10.1006/jagm.1999.1033.

[24] Christian Wulff-Nilsen. *Fully-Dynamic Minimum Spanning Forest with Improved Worst-Case Update Time*, 2016. https://arxiv.org/abs/1611.02864.

[25] Frank K. Boesch, Franz T. & Chen. *On the Generalized Edge-Connectivity of a Graph*. *SIAM Journal on Applied Mathematics*, 35(3):433–438, 1978. Ιντροδυςες ςυτ-ερσιον οφ *k*-εδγε-ςοννεςτιιτψ.

[26] Wikipedia. *K-edge-connected Graph*, 2025. https://en.wikipedia.org/wiki/K-edge-connected_graph.

[27] GeeksforGeeks. *Find Bridges in a Graph Using Tarjan's Algorithm*. https://www.geeksforgeeks.org/bridge-in-a-graph/.

[28] R. E. Tarjan. *Depth-First Search and Linear Graph Algorithms*. *SIAM Journal on Computing*, 1(2):146–159, 1972.

[29] Codeforces. *Bridge Tree Algorithm - Implementation in C++*. https://codeforces.com/blog/entry/99259.

[30] Mihai Patrascu & Erik D. Demaine. *Logarithmic Lower Bounds in the Cell-Probe Model*, 2005.

Συντομογραφίες - Αρκτικόλεξα - Ακρωνύμια

βλ.	βλέπε
Π.Ι.	Πανεπιστήμιο Ιωαννίνων
Ε.Μ.Π.	Εθνικό Μετσόβιο Πολυτεχνείο
Κεφ.	Κεφάλαιο
Ε.Σ.Δ.	Ελάχιστο Συνδετικό Δέντρο
P.S.S.	Richard Peng, Bryce Sandlund, και Daniel D. Sleator
B.F.	Brute Force

Απόδοση ξενόγλωσσων όρων

γράφος / γράφημα	graph
ακατεύθυντο	undirected
αραιοποίηση	sparsification
σύνδεση / συνεκτικότητα / συνδεσιμότητα	connectivity
δυναμικός	dynamic
ακμή	edge
γέφυρα	bridge
συνεκτικότητα ακμών	edge connectivity
δισυνεκτικότητα ακμών	2-edge connectivity
τρισυνεκτικότητα ακμών	3-edge connectivity
μόνιμες ακμές	permanent edges
x-ακμική συνεκτικότητα	x-edge connectivity
x-ακμικά συνδεδεμένα	x-edge connected
κόμβος / κορυφή	node / vertex
άρθρωση	articulation point
συνεκτικότητα κορυφών	vertex connectivity
δισυνεκτικότητα κορυφών	2-vertex connectivity
τρισυνεκτικότητα κορυφών	3-vertex connectivity
ενεργές κορυφές	active vertices
x-κομβική συνεκτικότητα	x-vertex connectivity
x-κομβικά συνδεδεμένα	x-vertex connected
αντιπρόσωπος	representative
μοντέλο	framework
διαίρει και βασίλευε	divide and conquer
αλγόριθμος	algorithm
χρονική πολυπλοκότητα	time complexity
αποσβεστικός	amortized
λειτουργίες	operations
ακολουθία	sequence
ερώτημα	query
προεπεξεργασία	preprocessing
βέλτιστο	optimal
υλοποίηση	implementation
πλήθος	number
εξαντλητική αναζήτηση	brute force search

εισαγωγή	insertion
διαγραφή	deletion
πλήρως	fully
μερικώς	partially