Οπτικά Δίκτυα Επικοινωνιών

3η Εργαστηριακή Άσκηση

Υλοποίηση Αλγορίθμου "*k* shortest path routing"

ΒΟΥΛΓΑΡΗΣ ΝΙΚΟΛΑΟΣ, 1084626

st1084626@ceid.upatras.gr

Link για GitHub Repository με τα αρχεία της εργασίας

Ερώτημα Α)

Αφορά τα αρχεία main a,b.py και dijkstra_shortest_path.py

Ο αλγόριθμος του Dijkstra βρίσκει τη συντομότερη διαδρομή από έναν δεδομένο κόμβο πηγής σε κάθε άλλο κόμβο, βελτιώνοντας επαναληπτικά τις εκτιμήσεις της απόστασης. Επεξεργάζεται τους κόμβους κατά σειρά αυξανόμενης απόστασης, επιλέγοντας τον κόμβο με τη μικρότερη απόσταση από το σύνολο που δεν έχει επισκεφτεί. Μετά την επεξεργασία των γειτόνων αυτού του κόμβου, ο κόμβος χαρακτηρίζεται ως επισκέψιμος και αφαιρείται από την εξέταση. Η διαδικασία επαναλαμβάνεται έως ότου επισκεφθούν όλοι οι κόμβοι ή παραμείνουν μόνο μη προσβάσιμοι κόμβοι, διασφαλίζοντας ότι η απόσταση κάθε επισκεπτόμενου κόμβου αντιπροσωπεύει τη συντομότερη διαδρομή από τον αρχικό κόμβο. Το k shortest path routing είναι δυνατό επεκτείνοντας τον αλγόριθμο του Dijkstra.

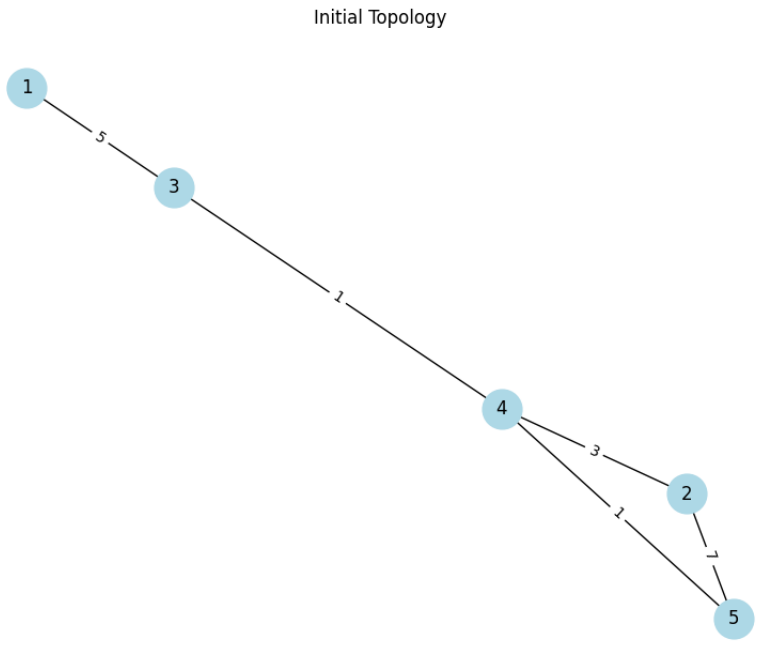
Μπαίνοντας τώρα στην υλοποίηση, στο script έχουμε μια μεταβλητή matrix, στην οποία αποθηκεύεται η τοπολογία του δικτύου που παρέχεται στην εκφώνηση. Κάνουμε την υπόθεση ότι οι αμφίδρομες ζεύξεις ανάμεσα σε δυο κόμβους δε θεωρούνται ξεχωριστές (π.χ. η ζεύξη (1,3) και (3,1) θεωρούνται ίδιες). Η μεταβλητή k ορίζει τον αριθμό των συντομότερων μονοπατιών που αναζητούμε.

Καλούμε την συνάρτηση find_all_k_shortest_paths(matrix, k). Στην συνάρτηση εξετάζουμε ένα βρόχο όλους τους πιθανούς συνδυασμούς κόμβου πηγής και στόχου (ευρετηρίαση με βάση το 1). Κρατάμε μόνο τους συνδυασμούς κόμβου-στόχου όπου οι στόχοι είναι μεγαλύτεροι από τις πηγές. Αν θέλαμε να εξετάσουμε τα ζεύγη κόμβων όπου ο κόμβος στόχος έχει μεγαλύτερη τιμή από την πηγή, θα είχαμε τα ίδια μονοπάτια. Δε συμπεριλαμβάνονται εδώ για λόγους απλότητας. Για κάθε ζεύγος καλείται η συνάρτηση dijkstra_k_shortest_paths(matrix, source, target, k), η λειτουργία της οποίας εξηγείται

παρακάτω. Τα αποτελέσματα αποθηκεύονται στο all_paths dictionary. Κάθε key του dictionary είναι ένα tuple (πηγή, στόχος) και κάθε value είναι ένα list των k συντομότερων μονοπατιών μεταξύ αυτών των κόμβων.

Στην συνάρτηση dijkstra_k_shortest_paths(), source και target είναι οι δείκτες των κόμβων πηγής και στόχου με βάση το 1. Αρχικοποιείται μια paths list. Αρχικοποιείται επίσης μια μεταβλητή heap με ένα tuple που περιέχει (κόστος, πηγή, λίστα μονοπατιών). Το κόστος αρχικά είναι μηδέν και η λίστα περιέχει μόνο το source. Επεξεργαζόμαστε τα μονοπάτια με αύξουσα σειρά κόστους, κάνοντας τα pop από το heap με χρήση της βιβλιοθήκης heapq. Εάν το node που βρισκόμαστε είναι ίσο με το target μας, προστίθεται στον κατάλογο μονοπατιών και ο αλγόριθμος συνεχίζει να ψάχνει περισσότερα μονοπάτια. Αλλιώς για τον τρέχοντα κόμβο, ο αλγόριθμος ελέγχει όλους τους γείτονες: Εάν υπάρχει έγκυρη ζεύξη (κόστος > 0 στο matrix) και ο γείτονας δεν είναι ήδη στο μονοπάτι (για την αποφυγή βρόχων), υπολογίζεται το κόστος του νέου μονοπατιού και προστίθεται ο γείτονας στο μονοπάτι. Το νέο μονοπάτι και το κόστος του γίνονται push στο heap. Ο αλγόριθμος σταματά όταν είτε ο heap είναι άδειος είτε έχουν βρεθεί τα k συντομότερα μονοπάτια. Η paths list επιστρέφει στο τέλος της συνάρτησης.

Στη συνέχεια εκτυπώνουμε για οπτική βοήθεια ένα plot του γραφήματος της τοπολογίας με χρήση των συναρτήσεων create_graph() και plot_graph(). Η πρώτη μετατρέπει τα δεδομένα του πίνακα γειτνίασης σε γράφημα με χρήση της networkx βιβλιοθήκης. Η δεύτερη εμφανίζει το γράφημα και τονίζει τις διαδρομές με χρήση της matplotlib.pyplot βιβλιοθήκης. Στην συνέχεια εκτυπώνονται τα αποτελέσματα από το k shortest path routing. Λόγω της τοπολογίας του δικτύου και του μικρού αριθμού κόμβων διαδρομών παρατηρούμε πως τα συντομότερα πιθανά μονοπάτια για όλα τα ζεύγη είναι λιγότερα από 3.



Αρχικό Γράφημα

```
C:\Users\nickv\AppData\Local\Programs\Python\Python311\python.exe C:\Users\nickv\Documents\fiber_optics_project\main_a,b.py
All k shortest paths (skipping bidirectional edges):
From 1 to 2:
  1: Cost = 9, Path = [1, 3, 4, 2]
  2: Cost = 14, Path = [1, 3, 4, 5, 2]
From 1 to 3:
  1: Cost = 5, Path = [1, 3]
From 1 to 4:
  1: Cost = 6, Path = [1, 3, 4]
From 1 to 5:
  1: Cost = 7, Path = [1, 3, 4, 5]
  2: Cost = 16, Path = [1, 3, 4, 2, 5]
From 2 to 3:
  1: Cost = 4, Path = [2, 4, 3]
  2: Cost = 9, Path = [2, 5, 4, 3]
From 2 to 4:
  1: Cost = 3, Path = [2, 4]
  2: Cost = 8, Path = [2, 5, 4]
From 2 to 5:
  1: Cost = 4, Path = [2, 4, 5]
  2: Cost = 7, Path = [2, 5]
From 3 to 4:
  1: Cost = 1, Path = [3, 4]
From 3 to 5:
  1: Cost = 2, Path = [3, 4, 5]
  2: Cost = 11, Path = [3, 4, 2, 5]
From 4 to 5:
  1: Cost = 1, Path = [4, 5]
  2: Cost = 10, Path = [4, 2, 5]

Process finished with exit code 0
```

k=3 shortest paths

Ερώτημα Β)

Αφορά τα αρχεία main_a,b.py και path_analysis.py

Από τα εναλλακτικά μονοπάτια θέλουμε να επιλέξουμε το συντομότερο για κάθε ζεύγος κόμβων πηγής-στόχου. Καλούμε την συνάρτηση analyze_paths(all_k_shortest_paths). Στην συνάρτηση αυτή εισάγουμε το dictionary με όλα ζεύγη κόμβων πηγής-στόχου και τα συντομότερα μονοπάτια μεταξύ τους. Αρχικοποιούμε δυο dictionary edge_counts (που παρακολουθεί τη συχνότητα κάθε ζεύξης σε επιλεγμένα μονοπάτια) και selected_paths (που περιέχει το επιλεγμένο μονοπάτι με τις λιγότερες ζεύξεις).

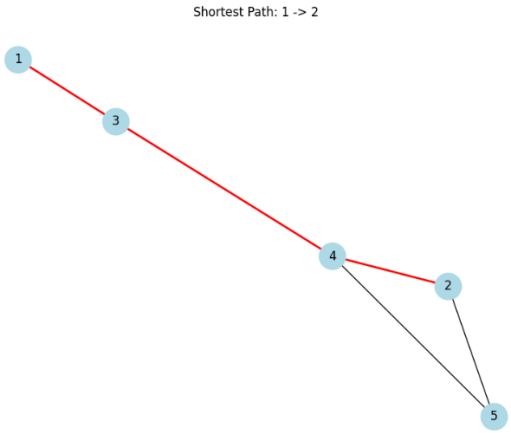
Για κάθε ζεύγος πηγής-στόχου βρίσκουμε το μονοπάτι με τις λιγότερες ζεύξεις με χρήση της min(), βάσει το len του μονοπατιού. Αποθηκεύουμε το επιλεγμένο μονοπάτι στο selected_paths. Στο επιλεγμένο μονοπάτι μετράμε τις εμφανίσεις κάθε ζεύξης, αντιμετωπίζοντας τις αμφίδρομες ως ίδιες (με χρήση του tuple(sorted())). Οι τιμές για κάθε ζεύξη αποθηκεύονται στο edge_counts. Υπολογίζουμε το μέγιστο, το ελάχιστο και το μέσο αριθμό εμφανίσεων τους και τα αποθηκεύουμε στα max_appearance, min_appearance, avg_appearance αντίστοιχα. Η συνάρτηση επιστρέφει τα selected_paths, τα edges_counts και τα υπολογισμένα στατιστικά στοιχεία ζεύξεων.

Πέρα από το να εκτυπώσουμε τα αποτελέσματα, εξάγουμε και ένα ξεχωριστό plot για κάθε ζεύγος κόμβων με το μονοπάτι που ακολούθησε. Εφόσον πλέον δε ασχολούμαστε με τα κόστη, δε

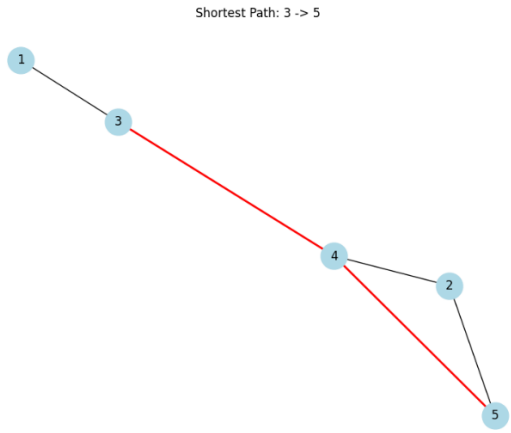
φαίνονται και πάνω στις ζεύξεις. Παραθέτω τα αποτελέσματα και δυο από τα γραφήματα ενδεικτικά:

```
C:\Users\nickv\AppData\Local\Programs\Python\Python311\python.exe C:\Users\nickv\Documents\fiber_optics_project\main_a,b.py
Selected paths with the least edges:
From 1 to 2: Path = [1, 3, 4, 2]
From 1 to 3: Path = [1, 3]
From 1 to 4: Path = [1, 3, 4]
From 1 to 5: Path = [1, 3, 4, 5]
From 2 to 3: Path = [2, 4, 3]
From 2 to 4: Path = [2, 4]
From 2 to 5: Path = [2, 5]
From 3 to 4: Path = [3, 4]
From 3 to 5: Path = [3, 4, 5]
From 4 to 5: Path = [4, 5]

Edge appearance statistics:
Maximum appearance: 6
Minimum appearance: 1
Average appearance: 3.40

Process finished with exit code 0
```

Shortest Path και Στατιστικά



Μονοπάτι για 1,2



Μονοπάτι για 3,5

Τώρα θέλουμε να εγκαταστήσουμε και lightpaths και να αναθέσουμε μήκη κύματος στην τοπολογία του δικτύου. Στην εκφώνηση αναφέρεται πως αν μια ζεύξη εμφανίζεται σε N μονοπάτια, τότε η χρήση της είναι N, δηλαδή ζητείται η ελαχιστοποίηση των μηκών κύματος που περνάνε από εκείνη την ζεύξη Πρέπει να υπακούσουμε στους βασικούς περιορισμούς σχετικά με την εγκατάσταση lightpaths. Δύο lightpaths που διατρέχουν κοινή ζεύξη δε μπορούν να έχουν το ίδιο μήκος κύματος. Ένα lightpath πρέπει να έχει το ίδιο μήκος κύματος σε όλες τις ζεύξεις που διατρέχει. Τα μήκη κύματος μπορούν να επαναχρησιμοποιηθούν εφόσον δεν καταπατούν αυτούς τους περιορισμούς.

Καλούμε την συνάρτηση allocate_bandwidths(selected_paths). Αναθέτει μήκη κύματος (πόρους) σε μονοπάτια στο δίκτυο για την αποφυγή συγκρούσεων σε κοινές ζεύξεις. Αρχικοποιούμε

τα dictionary wavelengths (παρακολουθεί το σύνολο των μηκών κύματος που χρησιμοποιούνται από κάθε ζεύξη) και lightpaths (αποθηκεύει το εκχωρημένο μήκος κύματος για κάθε ζεύγος πηγής-στόχου). Αποθηκεύουμε τον συνολικό αριθμό των μηκών κύματος που χρησιμοποιούνται στην μεταβλητή l. Για κάθε ζεύγος, αν υπάρχει έγκυρη διαδρομή αποθηκεύουμε στη unavailable_wavelengths τη συλλογή μηκών κύματος που έχουν ήδη χρησιμοποιηθεί στις ζεύξεις του. Με βάσει το συνολικό αριθμό μηκών που έχουν χρησιμοποιηθεί μέχρι τώρα, προσπαθούμε να βρούμε το πρώτο διαθέσιμο που δεν περιλαμβάνεται στο unavailable_wavelengths και να το αποθηκεύσουμε στην μεταβλητή assigned_wavelength.

Εάν δεν υπάρχει διαθέσιμο μήκος και η μεταβλητή assigned_wavelength δεν έχει πάρει τιμή, αυξάνουμε το l και το αναθέτουμε ως νέο μήκος κύματος. Το καταγράφουμε στο lightpaths dictionary για το τρέχον ζεύγος πηγής-στόχου. Για κάθε ζεύξη στην τρέχουσα διαδρομή ενημερώνουμε τον συνολικό αριθμό εκχωρημένων μηκών στο wavelength dictionary. Η συνάρτηση επιστρέφει τα δυο dictionary. Παρακάτω φαίνονται τα αποτελέσματα:

```
C:\Users\nickv\AppData\Local\Programs\Python\Python311\python.exe C:\Users\nickv\Documents\fiber_optics_project\main_a,b.py

Lightpaths and assigned wavelengths:
From 1 to 2: λ1
From 1 to 3: λ2
From 1 to 4: λ3
From 1 to 5: λ4
From 2 to 3: λ2
From 2 to 4: λ3
From 2 to 5: λ1
From 3 to 4: λ5
From 3 to 5: λ6
From 4 to 5: λ1

Wavelength usage per edge:
Edge (1, 3): Wavelengths = [1, 2, 3, 4]
Edge (3, 4): Wavelengths = [1, 2, 3, 4, 5, 6]
Edge (2, 4): Wavelengths = [1, 2, 3]
Edge (4, 5): Wavelengths = [1, 4, 6]
Edge (2, 5): Wavelengths = [1]

Process finished with exit code 0
```

Ανάθεση Μηκών Κύματος στα Lightpath

Ερώτημα Γ)

Αφορά τα αρχεία main_c.py και wavelength_assignment.py

Από την εκφώνηση δίνεται ένα traffic matrix που υποδηλώνει τα μήκη κύματος που χρειάζονται για την εγκατάσταση σύνδεσης ανάμεσα στους κόμβους του δικτύου. Σκοπός είναι να αναθέσουμε μήκη κύματος στα ζευγάρια κόμβων που ζητούν σύνδεση, με βάσει τους κανόνες για τα lightpath που παρατέθηκαν στο ερώτημα Β. Θεωρούμε όπως και στα παραπάνω ερωτήματα ότι οι αμφίδρομες ζεύξεις, όπως (2,4) και (4,2), είναι ίδιες. Επομένως αν έχει ανατεθεί ένα μήκος κύματος για την σύνδεση των κόμβων 2,4 και φτάσουμε στο στοιχείο (4,2) του traffic matrix B, θα θεωρήσουμε ότι πρόκειται για τη ίδια ζεύξη. Αναθέτουμε νέο μήκος κύματος στο μονοπάτι για να

μην παραβούμε τους κανόνες. Κρατάμε όλες τις ζεύξεις σε μια links list. Ο αριθμός των διαθέσιμων μηκών αποθηκεύεται στην μεταβλητή lambda_count. Στο num_nodes κρατάμε τον αριθμό των κόμβων. Τέλος, κρατάμε τα μονοπάτια από το B ερώτημα στη μεταβλητή paths.

Τα μήκη κύματος ανατίθενται με βάσει 3 αλγόριθμους, η υλοποίηση των οποίων παρατίθεται παρακάτω:

- Random Fit

Καλούμε την συνάρτηση random_fit(paths, lambda_count, num_nodes, B, links).Αρχικοποιούμε 2 dictionary link_wavelengths (παρακολουθεί τη διαθεσιμότητα των μηκών κύματος για κάθε ζεύξη με Boolean True για διαθέσιμο) και wavelength_allocation (παρακολουθεί τα ζεύγη πηγής-στόχου που αντιστοιχούν σε κάθε μήκος κύματος). Αρχικοποιούμε και δυο μεταβλητές total_requests και blocked_requests. Σε βρόχο για όλα τα ζεύγη κόμβων, ελέγχουμε την αίτηση για κάθε ζεύγος με βάσει το πίνακα B. Εφόσον η αίτηση είναι μη μηδενική μετατρέπει το μονοπάτι σε κατάλογο ζεύξεων χρησιμοποιώντας την συνάρτηση path_to_links().

Τώρα σε βρόχο με βάσει τον απαιτούμενο αριθμού μηκών για το ζεύγος βρίσκουμε ποια από τα παρεχόμενα μήκη είναι ακόμα διαθέσιμα σε όλες τις ζεύξεις της διαδρομής. Αυξάνεται η τιμή της μεταβλητής total_requests. Εάν δεν υπάρχουν διαθέσιμα κοινά μήκη κύματος αυξάνεται η τιμή του blocked_requests και καταγράφεται το μπλοκαρισμένο αίτημα. Διαφορετικά ακολουθεί ανάθεση μήκους κύματος. Επιλέγεται τυχαία ένα διαθέσιμο μήκος κύματος (βιβλιοθήκη random). Σημειώνεται το επιλεγμένο μήκος κύματος ως μη διαθέσιμο σε όλες τις ζεύξεις της διαδρομής και προστίθεται στο wavelength_allocation. Τέλος υπολογίζεται το ποσοστό των μπλοκαρισμένων αιτήσεων ως (blocked_requests/total_requests)×100. Επιστρέφεται το blocked_percentage και το wavelength_allocation.

- First Fit

Η συνάρτηση είναι ίδια με την random_fit, το μόνο που αλλάζει είναι ο τρόπος που επιλέγονται τα μήκη κύματος. Για κάθε μονάδα αίτησης των ζεύξεων ελέγχουμε όλα τα μήκη κύματος με αύξουσα σειρά. Το πρώτο μήκος κύματος που είναι διαθέσιμο σε όλες τις ζεύξεις της διαδρομής, ανατίθεται και σημειώνεται ως μη διαθέσιμο σε αυτές. Καταγράφουμε την κατανομή στο wavelength_allocation.

- Least Used

Παρομοίως η least_used είναι πολύ κοντά στις άλλες δυο με τις παρακάτω αλλαγές. Χρησιμοποιείται ο array wavelength_usage (για κάθε μήκος κύματος αποθηκεύει πόσες φορές έχει εκχωρηθεί). Μεταξύ των διαθέσιμων μηκών, επιλέγεται αυτό με τη μικρότερη συχνότητα χρήσης χρησιμοποιώντας τη μέθοδο min() με κλειδί το wavelength_usage. Το σημειώνουμε ως μη διαθέσιμο σε όλες τις ζεύξεις της διαδρομής, αυξάνουμε τον αριθμό χρήσης της στο wavelength_usage και καταγράφουμε την κατανομή στο wavelength_allocation.

Εκτυπώνονται η λίστα των συνδέσεων, το ποσοστό των αιτήσεων που δεν εξυπηρετείται δλδ το ποσοστό των αιτήσεων που έγινε “blocked” από τον κάθε αλγόριθμο και ο χρόνος εκτέλεσης του κάθε αλγορίθμου. Για τον υπολογισμό του χρόνου εκτέλεσης των αλγορίθμων χρησιμοποιούμε την βιβλιοθήκη time και την συνάρτηση time.perf_counter(). Αυτή η λειτουργία παρέχει μεγαλύτερη ακρίβεια, ειδικά για μικρά χρονικά διαστήματα, καθώς χρησιμοποιεί το ακριβέστερο ρολόι που διαθέτει το σύστημα.

Ο Random Fit παρατηρούμε πως παίρνει το περισσότερο χρόνο. Αυτό συμβαίνει γιατί περιλαμβάνει την συνάρτηση συνάρτησης random.choice, η οποία εισάγει πρόσθετη επιβάρυνση σε σύγκριση με έναν απλό διαδοχικό έλεγχο. Η random.choice μετατρέπει επίσης το σύνολο των διαθέσιμων μηκών κύματος σε λίστα, προσθέτοντας περαιτέρω υπολογιστικό κόστος. Ο First Fit πραγματοποιεί διαδοχικό έλεγχο των μηκών κύματος, σταματώντας μόλις βρεθεί ο πρώτος διαθέσιμος. Η απλότητα του ελέγχου με αύξουσα σειρά και η πρόωρη διακοπή το κάνει πιο γρήγορο από όλους. Ο Least Used είναι πιο αργός αλλά κοντά στο First Fit. Αυτό θα μπορούσε να οφείλεται στον αριθμό των διαθέσιμων μηκών κύματος που είναι μικρός, καθιστώντας αμελητέα την επιβάρυνση από την εύρεση του λιγότερο χρησιμοποιούμενου μήκους κύματος. Επίσης η ενσωματωμένη συνάρτηση min() της Python είναι ιδιαίτερα βελτιστοποιημένη για μικρά μεγέθη δεδομένων, γεγονός που ελαχιστοποιεί την αναμενόμενη διαφορά απόδοσης.

```
C:\Users\nickv\AppData\Local\Programs\Python\Python311\python.exe C:\Users\nickv\Documents\fiber_optics_project\main_c.py

Wavelength to Connections Mapping based on Random Fit Algorithm:
λ1 = 1-3, 2-4, 2-5
λ2 = 1-3, 4-2, 4-3, 4-5, 5-2
λ3 = 2-5, 3-1, 3-4, 5-4

Percentage of requests blocked: 7.69%
Time taken by Random Fit: 0.000085400 seconds

Wavelength to Connections Mapping based on First Fit Algorithm:
λ1 = 1-3, 2-4, 2-5, 3-4, 4-5
λ2 = 1-3, 2-5, 4-2, 4-3, 5-4
λ3 = 3-1, 5-2

Percentage of requests blocked: 7.69%
Time taken by First Fit: 0.000048600 seconds

Wavelength to Connections Mapping based on Least Used Algorithm:
λ1 = 1-3, 2-5, 3-4, 4-5
λ2 = 1-3, 2-5, 4-2, 5-4
λ3 = 2-4, 3-1, 4-3, 5-2

Percentage of requests blocked: 7.69%
Time taken by Least Used: 0.000055000 seconds

Execution Time Summary:
Random Fit: 0.000085400 seconds
First Fit: 0.000048600 seconds
Least Used: 0.000055000 seconds

Process finished with exit code 0
```

Σύγκριση Χρόνων

Όσο αφορά τώρα το αριθμό λ των διαθέσιμων μηκών κύματος και την αποδοτικότητα των αλγορίθμων, το traffic matrix δε ήταν τέτοιο ώστε να δίνει δυνατότητα σύγκρισης του block percentage. Όλες οι αιτήσεις για σύνδεση απαιτούν το πολύ 4 μήκη κύματος, άρα για λ=4 έχουμε αρκετά διαθέσιμα μήκη κύματος για μηδενισμό του ποσοστού. Λόγω της μορφής του μητρώου επίσης δε παρατηρούμε διαφορές στο ποσοστό από αλγόριθμο σε αλγόριθμο. Παρατηρούμε όμως

σημαντικές διαφορές στο πως ανατίθενται τα μήκη κύματος σε κάθε ζεύξη και ποιος αλγόριθμος διαχειρίζεται τα μήκη πιο αποδοτικά. Παρατίθενται μερικά πειράματα:

```
C:\Users\nickv\AppData\Local\Programs\Python\Python311\python.exe C:\Users\nickv\Documents\fiber_optics_project\main_c.py

Wavelength to Connections Mapping based on Random Fit Algorithm:
λ1 = 1-3, 2-5, 3-4
λ2 = 2-4, 2-5, 3-1, 4-5
λ3 = 1-3, 4-2, 4-3, 5-2, 5-4

Percentage of requests blocked: 7.69%
Time taken by Random Fit: 0.000081100 seconds

Wavelength to Connections Mapping based on First Fit Algorithm:
λ1 = 1-3, 2-4, 2-5, 3-4, 4-5
λ2 = 1-3, 2-5, 4-2, 4-3, 5-4
λ3 = 3-1, 5-2

Percentage of requests blocked: 7.69%
Time taken by First Fit: 0.000047600 seconds

Wavelength to Connections Mapping based on Least Used Algorithm:
λ1 = 1-3, 2-5, 3-4, 4-5
λ2 = 1-3, 2-5, 4-2, 5-4
λ3 = 2-4, 3-1, 4-3, 5-2

Percentage of requests blocked: 7.69%
Time taken by Least Used: 0.000053700 seconds

Execution Time Summary:
Random Fit: 0.000081100 seconds
First Fit: 0.000047600 seconds
Least Used: 0.000053700 seconds

Process finished with exit code 0
```

λ=3

```
C:\Users\nickv\AppData\Local\Programs\Python\Python311\python.exe C:\Users\nickv\Documents\fiber_optics_project\main_c.py

Wavelength to Connections Mapping based on Random Fit Algorithm:
λ1 = 3-4, 4-5, 5-2
λ2 = 1-3, 2-5, 5-4
λ3 = 1-3, 4-3
λ4 = 2-4, 3-1, 5-2
λ5 = 2-5, 4-2

Percentage of requests blocked: 0.00%
Time taken by Random Fit: 0.000098900 seconds

Wavelength to Connections Mapping based on First Fit Algorithm:
λ1 = 1-3, 2-4, 2-5, 3-4, 4-5
λ2 = 1-3, 2-5, 4-2, 4-3, 5-4
λ3 = 3-1, 5-2
λ4 = 5-2
λ5 = None

Percentage of requests blocked: 0.00%
Time taken by First Fit: 0.000050900 seconds

Wavelength to Connections Mapping based on Least Used Algorithm:
λ1 = 1-3, 3-4, 5-2
λ2 = 1-3, 4-2, 5-2
λ3 = 2-4, 3-1, 5-4
λ4 = 2-5, 4-3
λ5 = 2-5, 4-5

Percentage of requests blocked: 0.00%
Time taken by Least Used: 0.000062200 seconds

Execution Time Summary:
Random Fit: 0.000098900 seconds
First Fit: 0.000050900 seconds
Least Used: 0.000062200 seconds

Process finished with exit code 0
```

λ=5

```
Wavelength to Connections Mapping based on Random Fit Algorithm:
λ1 = 1-3, 2-4, 2-5, 4-3, 5-4
λ2 = None
λ3 = 5-2
λ4 = 1-3, 3-4
λ5 = 2-5, 3-1, 4-5
λ6 = None
λ7 = 4-2, 5-2

Percentage of requests blocked: 0.00%
Time taken by Random Fit: 0.000164800 seconds

Wavelength to Connections Mapping based on First Fit Algorithm:
λ1 = 1-3, 2-4, 2-5, 3-4, 4-5
λ2 = 1-3, 2-5, 4-2, 4-3, 5-4
λ3 = 3-1, 5-2
λ4 = 5-2
λ5 = None
λ6 = None
λ7 = None

Percentage of requests blocked: 0.00%
Time taken by First Fit: 0.000044700 seconds

Wavelength to Connections Mapping based on Least Used Algorithm:
λ1 = 1-3, 4-2
λ2 = 1-3, 4-3
λ3 = 2-4, 4-5
λ4 = 2-5, 5-4
λ5 = 2-5
λ6 = 3-1, 5-2
λ7 = 3-4, 5-2

Percentage of requests blocked: 0.00%
Time taken by Least Used: 0.000060600 seconds
```

λ=7

```
Wavelength to Connections Mapping based on Random Fit Algorithm:
λ1 = None
λ2 = 1-3
λ3 = 4-2
λ4 = 4-3
λ5 = 3-1
λ6 = 1-3, 2-5, 5-4
λ7 = 2-4, 5-2
λ8 = 5-2
λ9 = 4-5
λ10 = 2-5, 3-4

Percentage of requests blocked: 0.00%
Time taken by Random Fit: 0.000098500 seconds

Wavelength to Connections Mapping based on First Fit Algorithm:
λ1 = 1-3, 2-4, 2-5, 3-4, 4-5
λ2 = 1-3, 2-5, 4-2, 4-3, 5-4
λ3 = 3-1, 5-2
λ4 = 5-2
λ5 = None
λ6 = None
λ7 = None
λ8 = None
λ9 = None
λ10 = None

Percentage of requests blocked: 0.00%
Time taken by First Fit: 0.000050200 seconds

Wavelength to Connections Mapping based on Least Used Algorithm:
λ1 = 1-3, 5-2
λ2 = 1-3, 5-2
λ3 = 2-4, 5-4
λ4 = 2-5
λ5 = 2-5
λ6 = 3-1
λ7 = 3-4
λ8 = 4-2
λ9 = 4-3
λ10 = 4-5
```

Λ=10

Παρατηρούμε τα παρακάτω:

Στο Random Fit, η χρήση του μήκους κύματος είναι λιγότερο προβλέψιμη, γεγονός που θα μπορούσε να οδηγήσει σε αναποτελεσματικότητα σε πιο περιορισμένα σενάρια. Λειτουργεί όμως καλά όταν το μπλοκάρισμα δεν αποτελεί πρόβλημα. Το First Fit έχει προβλέψιμη και συμπαγής κατανομή, μεγιστοποιώντας τη χρήση των πόρων στα μήκη κύματος με χαμηλότερους δείκτες. Αυτή η προσέγγιση είναι αποτελεσματική σε σενάρια αραιής κυκλοφορίας όπως το συγκεκριμένο. Ωστόσο, κινδυνεύει να υπερφορτώσει τα χαμηλότερα μήκη κύματος σε σενάρια πυκνής σύνδεσης. Το Least Used ισορροπεί την χρήση μηκών κύματος και αποφεύγει την υπερφόρτωση οποιουδήποτε μεμονωμένο, γεγονός που είναι επωφελές και στη δική μας τοπολογία και σε δίκτυα μεγαλύτερης κυκλοφορίας.

Πηγές

1. *Wikipedia,* ***k shortest path routing*** *https://en.wikipedia.org/wiki/K_shortest_path_routing*
2. *Average-Case Analyses of First Fit and Random Fit Packing* https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=7711a27d76645c40f28194b9d1cde80e09cf108c
3. Wavelength Assignment in Optical Networks with Imprecise Network State Information https://www.researchgate.net/publication/4287125_Wavelength_Assignment_in_Optical_Networks_with_Imprecise_Network_State_Information#pf2