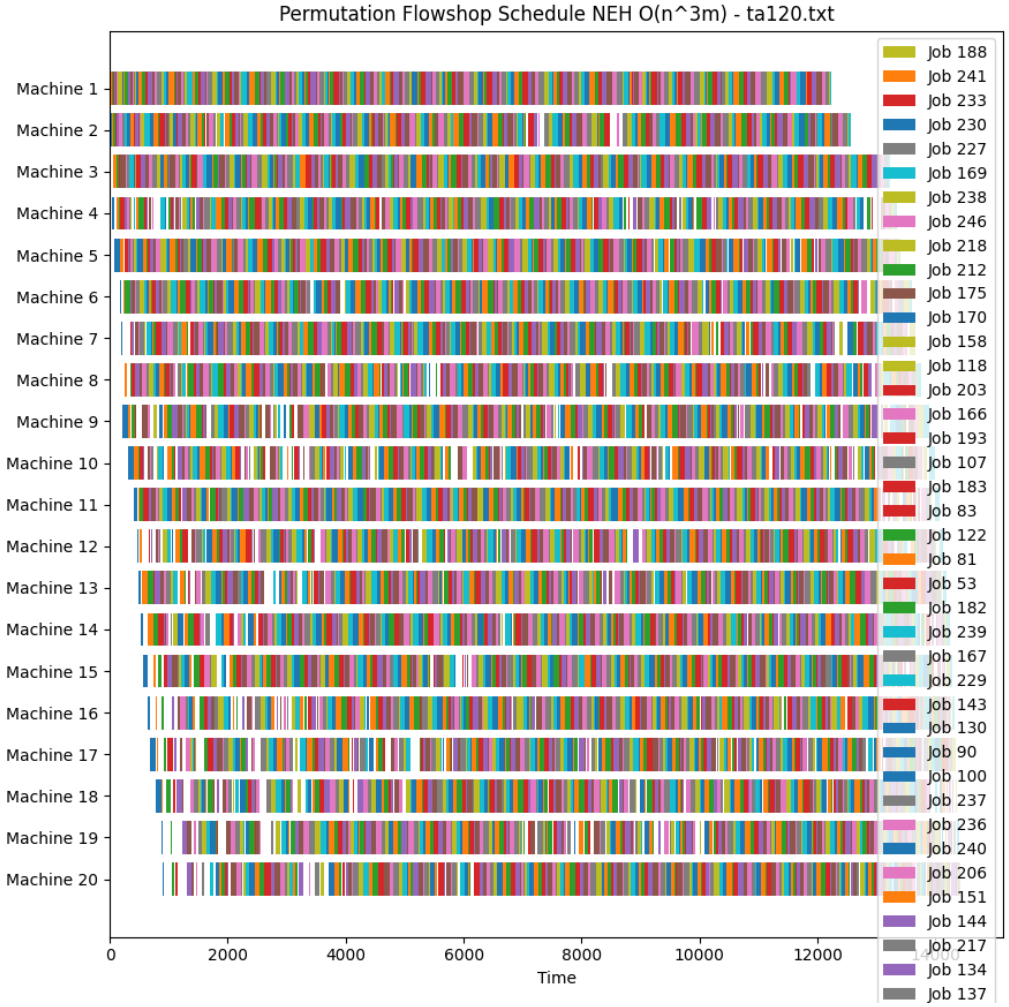
Πανεπιστήμιο Ιωαννίνων Τμήμα Πληροφορικής & Τηλεπικοινωνίων
Αλγόριθμοι & Προχωρημένες Δομές Δεδομένων

Permutation Flow-shop Scheduling Problem(PFSP)

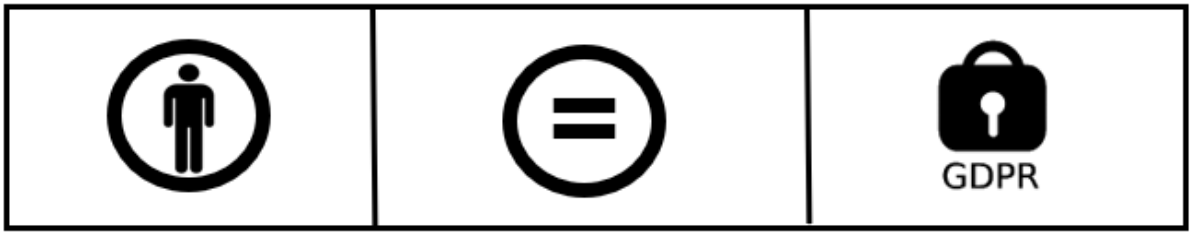


Αλγόριθμοι & Προχωρημένες Δομές Δεδομένων

Παναγιώτα Βίννη Α.Μ. 170

14 Ιανουαρίου 2024

Πίνακας περιεχομένων



Περίληψη

Η παρούσα εργασία αναλύει το πρόβλημα PFSP (Permutation Flow Shop Scheduling Problem) με στόχο την εξέταση και την επίλυσή του μέσω διαφόρων αλγορίθμων. Προσεγγίζοντας αυστηρά το πρόβλημα, περιλαμβάνονται όλες οι σημαντικές παραμέτρους, ενώ σχολιάζεται τη σημασία του PFSP σε πραγματικές εφαρμογές χρονοπρογραμματισμού παραγωγικών διεργασιών.

Εφαρμόζεται ο ευρετικός αλγόριθμος NEH για την αρχική επίλυση, ακολουθούμενη από τη βελτιστοποίηση του αλγορίθμου σύμφωνα με προηγούμενες προτάσεις. Η ανάλυση χρόνων εκτέλεσης επιβεβαιώνει την αποτελεσματικότητα της βελτιωμένης προσέγγισης.

Στη συνέχεια, προτείνεται μια προσωπική προσέγγιση, συγκρίνοντάς την με τα αποτελέσματα του NEH. Η σύγκριση αυτή αποκαλύπτει τα χαρακτηριστικά και τις δυνατότητες της κάθε προσεγγίσεως, εμπλουτίζοντας την κατανόησή μας για την απόδοση των αλγορίθμων στο πρόβλημα του χρονοπρογραμματισμού.

Συνοψίζοντας, η εργασία παρέχει ενδιαφέρουσες προσεγγίσεις για την αντιμετώπιση του PFSP, συνδυάζοντας θεωρητική ανάλυση με πρακτικές εφαρμογές και πειραματικές εκτιμήσεις.

Εισαγωγή

Η παρούσα εργασία αφορά την επίλυση του προβλήματος PFSP (Permutation Flow Shop Scheduling Problem) μέσω διάφορων αλγορίθμων, με έμφαση στη χρήση της γλώσσας προγραμματισμού Python. Η επιλογή της Python αντανακλά την επιθυμία για μια ευέλικτη και ισχυρή γλώσσα που διευκολύνει την υλοποίηση αλγορίθμων και την ανάλυση δεδομένων.

Η εφαρμογή των αλγορίθμων προγραμματίστηκε και εκτελέστηκε σε έναν υπολογιστή με τα ακόλουθα χαρακτηριστικά: Intel Core i7-1065G7 (1.30GHz - 1.50 GHz), 8 GB DDR3 (2667 MHz), λειτουργικό σύστημα Windows 11 Home x64.. Η εκτέλεση στον εν λόγω υπολογιστή επέτρεψε την αξιολόγηση και σύγκριση των αλγορίθμων, ενώ η Python προσέφερε εύκολη διεπαφή για την υλοποίηση και τον πειραματισμό με διάφορες τεχνικές.

Στο τέλος της εργασίας, παρουσιάζεται μια προσωπική προσέγγιση στην επίλυση του PFSP, συνδυάζοντας θεωρητική ανάλυση και πρακτική υλοποίηση με τη χρήση Python.

Ερώτημα 3.1

Το πρόβλημα Permutation Flow-shop Scheduling Problem (PFSP) είναι γνωστό ως είναι πρόβλημα χρονοπρογραμματισμού. Σε αυτό το σενάριο, ένα σύνολο n εργασιών (jobs) πρέπει να επεξεργαστεί με συγκεκριμένη σειρά σε m μηχανές (machines). Ο χρόνος που απαιτείται για την επεξεργασία της εργασίας i στη μηχανή j συμβολίζεται με schedule[i j], όπου το i κυμαίνεται από 1 έως n και το j από 1 έως m. Αυτοί οι χρόνοι επεξεργασίας είναι σταθεροί, μη αρνητικοί και μπορούν να είναι μηδενικοί εάν μια εργασία δεν επεξεργάζεται σε μια συγκεκριμένη μηχανή.

Ο πρωταρχικός στόχος του συγκεκριμένου προβλήματος είναι να ελαχιστοποιηθεί ο χρόνος από την έναρξη της πρώτης εργασίας στην πρώτη μηχανή έως την ολοκλήρωση της τελευταίας εργασίας στην τελευταία μηχανή, που αναφέρεται ως χρόνος παραγωγής ή αλλιώς makespan. Για το πρόβλημα αυτό εξετάζονται διάφορες παραδοχές, όπως:

- Κάθε job πρέπει να υποβληθεί σε επεξεργασία το πολύ μία φορά σε κάθε machine με προκαθορισμένη σειρά. (με αυτή τη σειρά: 1, 2, ..., m).
- Κάθε machine επεξεργάζεται μόνο μία job κάθε φορά.
- Κάθε job επεξεργάζεται ταυτόχρονα σε ένα μόνο machine.
- Οι jobs δεν μπορούν να διακοπούν.
- Οι χρόνοι προετοιμασίας των jobs ενσωματώνονται στο χρόνο επεξεργασίας και είναι ανεξάρτητοι από την ακολουθία.
- Η σειρά των jobs είναι η ίδια για κάθε job σε κάθε machine και αυτή η κοινή ακολουθία (job sequence) πρέπει να καθοριστεί.

Είναι σημαντικό να σημειωθεί ότι το πρόβλημα αυτό χαρακτηρίζεται ως NP-hard, πράγμα που σημαίνει ότι η εύρεση μιας βέλτιστης λύσης γίνεται υπολογιστικά δύσκολη, ειδικά όσο αυξάνεται ο αριθμός των θέσεων εργασίας και των μηχανών. Έτσι, μπορεί να επιλυθεί με ακρίβεια μόνο για μικρά μεγέθη. Ο κύριος στόχος είναι να βρεθεί μια ακολουθία , που συμβολίζεται ως job sequence, η οποία ελαχιστοποιεί το makespan χρόνο.

Κατά συνέπεια, ο αριθμός των πιθανών χρονοδιαγραμμάτων είναι n!, τονίζοντας τη συνδυαστική πολυπλοκότητα του έργου.

Όσον αφορά τώρα τις πραγματικές εφαρμογές του προβλήματος PFSP, αυτές περιλαμβάνουν:

- **Manufacturing (Βιομηχανία):** Ο προγραμματισμός των μηχανών για διάφορες εργασίες με τη βέλτιστη αλληλουχία μπορεί να επηρεάσει σημαντικά την αποδοτικότητα και να μειώσει το χρόνο παραγωγής. Το PFSP βοηθά στην οργάνωση της παραγωγικής διαδικασίας για την επίτευξη καλύτερης συνολικής απόδοσης. Παραδείγματα βιομηχανιών είναι τα εξής: Printing Industry, Food Processing Industry, Chemical Production Industry κ.α.
- **Logistics and Transportation (Μεταφορές):** Εδώ διαφορετικές εργασίες (π.χ. παραδόσεις ή παραλαβές) πρέπει να εκτελεστούν με τη βέλτιστη σειρά για την ελαχιστοποίηση του χρόνου ταξιδιού ή του κόστους.
- **Job Scheduling in Computing (Χρονοπρογραμματισμός εργασιών στην πληροφορική):** Σε αυτό το πλαίσιο, οι μηχανές μπορεί να αντιπροσωπεύουν επεξεργαστές ή υπολογιστικούς πόρους και οι εργασίες αντιστοιχούν σε υπολογιστικές εργασίες που πρέπει να προγραμματιστούν αποτελεσματικά.

Σε γενικές γραμμές, οι αποτελεσματικές λύσεις στο PFSP μπορούν να οδηγήσουν σε εξοικονόμηση κόστους, βελτιωμένη αξιοποίηση των πόρων και αυξημένη παραγωγικότητα σε διάφορες βιομηχανικές και επιχειρησιακές ρυθμίσεις. Ωστόσο, η εύρεση βέλτιστων λύσεων γίνεται όλο και πιο δύσκολη καθώς το μέγεθος του προβλήματος αυξάνεται, γεγονός που παρακινεί την ανάπτυξη ευρετικών και προσεγγιστικών αλγορίθμων για την αντιμετώπιση πραγματικών περιπτώσεων του PFSP.

Ερώτημα 3.2

Στα πλαίσια της επίλυσης του προβλήματος PFSP πραγματοποιήθηκε η ανάγνωση και η συγκέντρωση δεδομένων από τα προβλήματα που περιγράφονται στο άρθρο του Eric Taillard, με τίτλο "Benchmarks for basic scheduling problems", που δημοσιεύτηκε στο European Journal of Operational Research.

Τα στιγμιότυπα προβλημάτων προήλθαν από τον φάκελο που διατίθεται στον παρακάτω σύνδεσμο στο GitHub: https://github.com/chgogos/dituoi_msc_aads/tree/main/2023f/project/Taillard-PFSP

Όσον αφορά την μεθοδολογία της ανάγνωσης των προβλημάτων, πραγματοποιήθηκε σύμφωνα με τις οδηγίες που παρέχονται στο προαναφερόμενο άρθρο. Καταγράφηκαν οι παράμετροι των προβλημάτων, όπως ο αριθμός των εργασιών, οι χρόνοι επεξεργασίας και οι περιορισμοί ροής.

Στους κώδικες της συγκεκριμένης εργασίας, υπάρχουν δύο τρόποι για να διαβάσει ο χρήστης τα στιγμιότυπα των προβλημάτων, ανάλογα με τις ανάγκες και τις προτιμήσεις του:

- **Ανάγνωση από το GitHub:** Ο χρήστης μπορεί να ανακτήσει τα στιγμιότυπα προβλημάτων PFSP από τον φάκελο στο GitHub, προσφέροντας ευκολία και αξιοπιστία, καθώς εξασφαλίζει την πρόσβαση σε ενημερωμένες εκδόσεις των δεδομένων.
- **Ανάγνωση από Τοπικό Φάκελο:** Ενώ για επιπλέον ευελιξία, ο χρήστης μπορεί να χρησιμοποιήσει τοπικό φάκελο για την ανάγνωση των στιγμιοτύπων. Αυτό του επιτρέπει να διαχειρίζεται τα δεδομένα τοπικά και να πραγματοποιεί τυχόν προσαρμογές ανάλογα με τις ανάγκες του.

Ερώτημα 3.3

Ο αλγόριθμος NEH, ο οποίος πήρε το όνομά του από τους εφευρέτες του Nawaz, Enscore και Ham (NEH) το 1982, είναι ένας ευρετικός αλγόριθμος που έχει σχεδιαστεί για την αντιμετώπιση του προβλήματος προγραμματισμού ροής-καταστήματος μεταβολής (PFSP). Το PFSP, όπως αναφέρθηκε και πιο πάνω, έχει στόχο την ελαχιστοποίηση του makespan, δηλαδή του συνολικού χρόνου που απαιτείται για την ολοκλήρωση όλων των εργασιών σε όλες τις μηχανές. Ο αλγόριθμος NEH είναι γνωστός για την απλότητα και την αποτελεσματικότητά του στη δημιουργία σχεδόν βέλτιστων λύσεων για περιπτώσεις PFSP. Ακολουθεί ένα βασικό περίγραμμα του τρόπου λειτουργίας του αλγορίθμου NEH:

1. **Initialization (Αρχικοποίηση)**
   i) Αρχικά, ο αλγόριθμος ξεκινά με ένα άδειο πρόγραμμα.
   ii) Επιλέγει την εργασία με τον μέγιστο χρόνο επεξεργασίας σε όλες τις μηχανές και την εισάγει στο πρόγραμμα.
2. **Iterative Procedure (Επαναληπτική διαδικασία)**
   - Για κάθε επόμενη εργασία, ο αλγόριθμος εισάγει την εργασία σε όλες τις πιθανές θέσεις του τρέχοντος χρονοδιαγράμματος και αξιολογεί την προκύπτουσα χρονική απόσταση.
   - Η εργασία εισάγεται στη θέση που ελαχιστοποιεί το makespan.
3. **Job Selection**
   - Οι εργασίες επεξεργάζονται με μη αύξουσα σειρά του συνολικού χρόνου επεξεργασίας τους σε όλες τις μηχανές. Με τον τρόπο αυτό δίνεται προτεραιότητα στις εργασίες με υψηλότερους συνολικούς χρόνους επεξεργασίας, με στόχο την ελαχιστοποίηση των επιπτώσεων στο makespan.
4. **Schedule Evaluation (Αξιολόγηση χρονοδιαγράμματος)**

- Μετά την εισαγωγή κάθε εργασίας σε μια πιθανή θέση, υπολογίζεται το makespan για να εκτιμηθεί ο συνολικός αντίκτυπος στο χρονοδιάγραμμα.

5. **Final Schedule**
    - Η διαδικασία επαναλαμβάνεται μέχρι να εισαχθούν όλες οι εργασίες στο πρόγραμμα.
6. **Makespan Minimization (Ελαχιστοποίηση του χρονικού διαστήματος)**
    - Το τελικό χρονοδιάγραμμα που παράγεται από τον αλγόριθμο NEH αναμένεται να έχει σχετικά χαμηλό makespan, αν και μπορεί να μην εγγυάται τη βέλτιστη λύση.

Ο αλγόριθμος NEH είναι ιδιαίτερα ελκυστικός λόγω της απλότητας και της αποτελεσματικότητάς του στη γρήγορη παραγωγή καλών λύσεων. Ωστόσο, είναι σημαντικό να σημειωθεί ότι ο NEH είναι ευρετικός και οι λύσεις του δεν είναι εγγυημένα βέλτιστες. Συχνά χρησιμοποιείται ως αφετηρία για πιο εξελιγμένους αλγορίθμους ή σε συνδυασμό με άλλες τεχνικές βελτιστοποίησης για την περαιτέρω βελτίωση της ποιότητας των λύσεων.

Τέλος, ο αλγόριθμος NEH έχει εφαρμοστεί ευρέως σε διάφορες βιομηχανίες όπου η βελτιστοποίηση του χρονοπρογραμματισμού είναι ζωτικής σημασίας, όπως η παραγωγή, η εφοδιαστική και ο προγραμματισμός εργασιών σε υπολογιστικά περιβάλλοντα. Η ικανότητά του να παράγει καλές λύσεις με σχετικά χαμηλή υπολογιστική πολυπλοκότητα τον καθιστά πρακτική επιλογή για ορισμένες περιπτώσεις PFSP.

Όσον αφορά το συγκεκριμένο πρόβλημα PFSP, ο αλγόριθμος NEH έχει υλοποιηθεί στο αρχείο **`neh_algorithm.py`** και περιλαμβάνει τις εξής λειτουργίες:

- **Υλοποίηση του αλγορίθμου NEH (neh_algorithm):** Ο αλγόριθμος εκτελεί τη σειρά των εργασιών με βάση το άθροισμα των χρόνων επεξεργασίας τους, σε φθίνουσα σειρά. Στη συνέχεια, προτείνει διάφορες θέσεις για κάθε εργασία και επιλέγει το πρόγραμμα που οδηγεί στο ελάχιστο makespan.
- **Υπολογισμός του makespan (calculate_makespan):** Υπολογίζει τον χρόνο ολοκλήρωσης (makespan) για ένα δεδομένο πρόγραμμα εκτέλεσης.
- **Χρόνος Εκτέλεσης Αλγορίθμου (count_time):** Υπολογίζει τον χρόνο εκτέλεσης του αλγορίθμου για κάθε αρχείο ξεχωριστά.
- **Διάβασμα Αρχείων από το GitHub (read_data_from_github):** Διαβάζει τα περιεχόμενα αρχείων από ένα δημόσιο Repository στο GitHub, χειρίζεται διάφορα σφάλματα και επιστρέφει τα δεδομένα.
- **Διάβασμα Αρχείων από το Τοπικό Σύστημα (read_data_from_file):** Διαφορετικά, υπάρχει και η επιλογή να διαβάζει τα περιεχόμενα αρχείων από το τοπικό σύστημα και να επιστρέφει τα δεδομένα.
- **Εκτέλεση Αλγορίθμου NEH για Όλα τα Αρχεία (run_neh_algorithm):** Καλεί και εκτελεί τον αλγόριθμο NEH για ένα συγκεκριμένο αρχείο, εμφανίζοντας τα αποτελέσματα.
- **Συνολικός Χρόνος Εκτέλεσης (Total Execution Time):** Υπολογίζει τον συνολικό χρόνο εκτέλεσης όλου του προγράμματος.
- **Εκτύπωση των αποτελεσμάτων σε μορφή Gantt chart (display_schedule):** Εμφανίζει γραφικά το πρόγραμμα εκτέλεσης σε μορφή Gantt chart, δίνοντας οπτική αναπαράσταση της σειράς εκτέλεσης των εργασιών σε κάθε μηχάνημα.

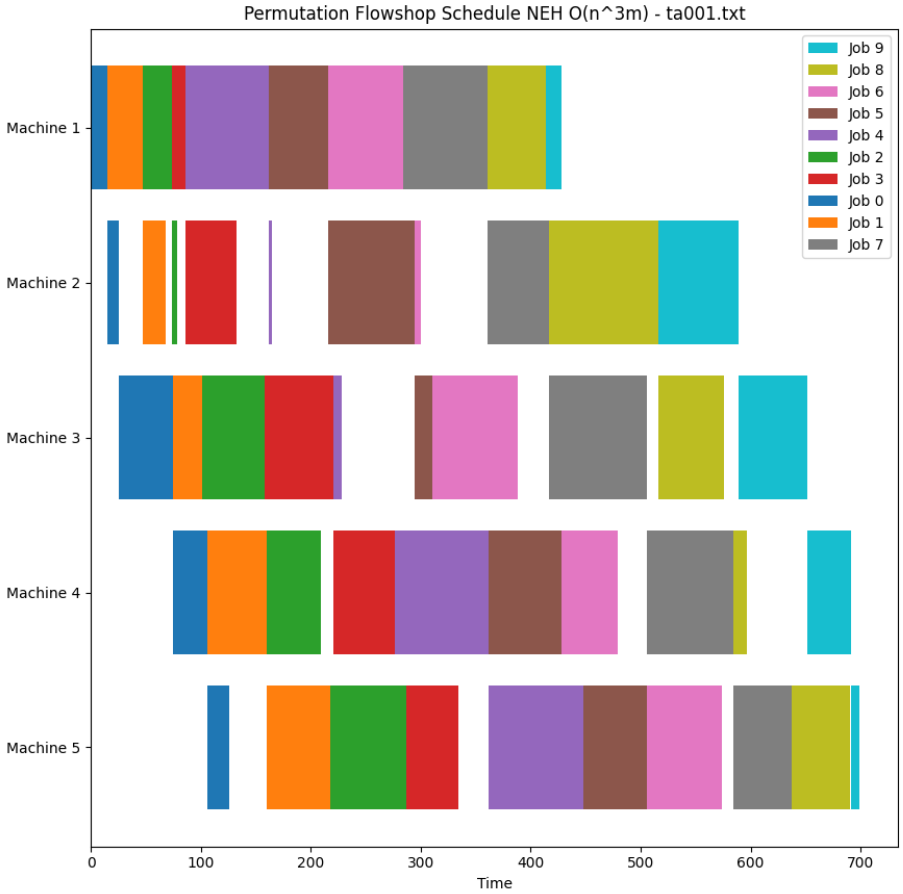
Κατά την εκτέλεση του προγράμματος, θα εμφανίζονται τα κάτωθι μηνύματα:

- **Εκτύπωση Ονόματος Αρχείου:** Εμφανίζει το όνομα του αρχείου που αντιστοιχεί στον κώδικα.
- **Εκτύπωση Βέλτιστης Σειράς Εργασιών (Schedule):** Εμφανίζει τη βέλτιστη σειρά εργασιών που προέκυψε από τον αλγόριθμο NEH.
- **Εκτύπωση Σωστής Σειράς Εργασιών (Job Indices):** Εμφανίζει τη σωστή σειρά εργασιών, δηλαδή τη σειρά των εργασιών μετά την εκτέλεση του αλγορίθμου NEH, με αριθμημένες τις εργασίες.
- **Εκτύπωση Χρόνου Επεξεργασίας (Makespan):** Εμφανίζει τον χρόνο ολοκλήρωσης (makespan) για το βέλτιστο πρόγραμμα εκτέλεσης.
- **Εκτύπωση Χρόνου Εκτέλεσης Αλγορίθμου (Execution Time):** Εμφανίζει τον χρόνο εκτέλεσης του αλγορίθμου για το συγκεκριμένο αρχείο.
- **Εκτύπωση Πίνακα Χρόνων Ολοκλήρωσης (Completion Times Table):** Εμφανίζει τον πίνακα χρόνων ολοκλήρωσης, που δείχνει τους χρόνους τερματισμού κάθε εργασίας σε κάθε μηχάνημα.

Αφού εμφανιστούν τα συγκεκριμένα μηνύματα στο Command Window, ανοίγει δεύτερο παράθυρο παρουσιάζοντας το διάγραμμα Gantt για το συγκεκριμένα αρχείο που εκτελείται. Παρακάτω παρουσιάζονται δύο συγκεκριμένα παραδείγματα εκτέλεσης του κώδικα και των διαγραμμάτων Gantt, για το αρχείο ta001.txt και για το αρχείο ta020.txt :

```
File: ./Taillard-PFSP/ta001.txt

NEH Optimal Jobs Order (Schedule):
 [[15, 11, 49, 31, 20], [32, 21, 26, 54, 58], [27, 5, 57, 49, 69], [12, 47, 63, 56, 47], [76, 3, 7, 85, 86], [54, 79, 16
, 66, 58], [68, 5, 77, 51, 68], [77, 56, 89, 78, 53], [53, 99, 60, 13, 53], [14, 73, 63, 39, 8]]

NEH Correct Jobs Order (Job Indices): [9, 8, 6, 5, 4, 2, 3, 0, 1, 7]

NEH Processing Time (Makespan): 699

Execution Time for ta001.txt file: 0.0007220000000000004 seconds

Completion Times Table:
[15, 26, 75, 106, 126]
[47, 68, 101, 160, 218]
[74, 79, 158, 209, 287]
[86, 133, 221, 277, 334]
[162, 165, 228, 362, 448]
[216, 295, 311, 428, 506]
[284, 300, 388, 479, 574]
[361, 417, 506, 584, 637]
[414, 516, 576, 597, 690]
[428, 589, 652, 691, 699]


--------------------------------------------------------------------------------


Total Execution Time: 0.019909143447875977 seconds
```

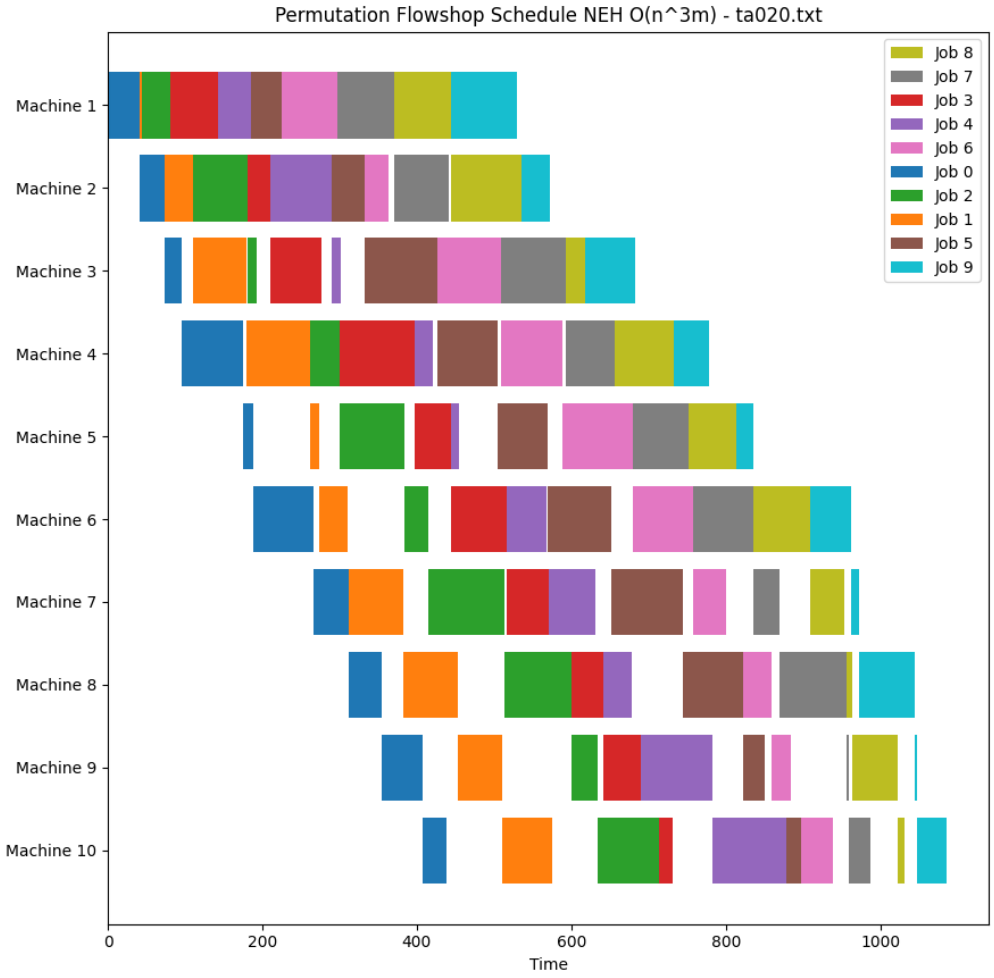
Εικόνα 1: Εκτέλεση NEH Αλγορίθμου για το αρχείο ta001.txt



Εικόνα 2: Διάγραμμα Gantt του NEH Αλγορίθμου για το αρχείο ta001.txt

```
File: ./Taillard-PFSP/ta020.txt

NEH Optimal Jobs Order (Schedule):
 [[41, 32, 22, 80, 14, 78, 45, 42, 53, 31], [3, 37, 69, 83, 12, 36, 70, 71, 57, 66], [37, 71, 12, 38, 84, 31, 99, 87, 33,
 80], [62, 29, 66, 97, 47, 73, 54, 41, 48, 17], [43, 80, 12, 24, 11, 51, 60, 36, 93, 96], [39, 42, 94, 78, 65, 83, 92, 79
, 28, 18], [72, 31, 83, 80, 91, 78, 43, 37, 25, 42], [74, 70, 84, 63, 72, 78, 33, 87, 3, 28], [74, 91, 25, 76, 62, 74, 44
, 8, 58, 9], [84, 36, 64, 46, 21, 53, 10, 71, 3, 39]]

NEH Correct Jobs Order (Job Indices): [8, 7, 3, 4, 6, 0, 2, 1, 5, 9]

NEH Processing Time (Makespan): 1086

Execution Time for ta020.txt file: 0.0011098000000000496 seconds

Completion Times Table:
[41, 73, 95, 175, 189, 267, 312, 354, 407, 438]
[44, 110, 179, 262, 274, 310, 382, 453, 510, 576]
[81, 181, 193, 300, 384, 415, 514, 601, 634, 714]
[143, 210, 276, 397, 444, 517, 571, 642, 690, 731]
[186, 290, 302, 421, 455, 568, 631, 678, 783, 879]
[225, 332, 426, 504, 569, 652, 744, 823, 851, 897]
[297, 363, 509, 589, 680, 758, 801, 860, 885, 939]
[371, 441, 593, 656, 752, 836, 869, 956, 959, 987]
[445, 536, 618, 732, 814, 910, 954, 964, 1022, 1031]
[529, 572, 682, 778, 835, 963, 973, 1044, 1047, 1086]


--------------------------------------------------------------------------------


Total Execution Time: 0.0181119441986084 seconds
```

Εικόνα 3: Εκτέλεση ΝΕΗ Αλγορίθμου για το αρχείο ta020.txt



Εικόνα 4: Διάγραμμα Gantt του ΝΕΗ Αλγορίθμου για το αρχείο ta020.txt

Ερώτημα 3.4

Η απεικόνιση των λύσεων προγραμμάτων ως γραφήματα Gantt με το χρόνο στον άξονα x, τις μηχανές στον άξονα y και τις εργασίες ως επιμέρους κουτάκια στις κατάλληλες θέσεις, υλοποιήθηκε με τη χρήση της βιβλιοθήκης **Matplotlib** στην Python. Η Matplotlib είναι μία από τις πιο δημοφιλείς βιβλιοθήκες γραφικών στην Python και παρέχει ευέλικτα εργαλεία για τη δημιουργία διαγραμμάτων και γραφημάτων.

Συγκεκριμένα, η χρήση της Matplotlib στον κώδικα περιλαμβάνει την εισαγωγή των εξής μετασχηματιστών:

```
import matplotlib.pyplot as plt # Βιβλιοθήκη για την σχεδίαση γραφημάτων
import matplotlib.colors as mcolors # Βιβλιοθήκη για την προσθήκη χρωμάτων στα γραφήματα
```

Με τη χρήση αυτών των βιβλιοθηκών, η συνάρτηση **`display_schedule`** δημιουργεί γραφήματα Gantt για την οπτικοποίηση των προγραμμάτων εκτέλεσης των εργασιών, βελτιώνοντας την κατανόηση των αποτελεσμάτων του αλγορίθμου Flowshop Scheduling. Ειδικότερα, η διαδικασία υλοποίησης της απεικόνισης μέσω της **`display_schedule`** είναι η εξής:

1) **Δημιουργία Πίνακα Χρόνων Ολοκλήρωσης (C_table):**
   - Πριν από την απεικόνιση, υπολογίζεται ο πίνακας χρόνων ολοκλήρωσης **C_table** για το βέλτιστο πρόγραμμα εκτέλεσης. Αυτός ο πίνακας αποτυπώνει τους χρόνους τερματισμού εκτέλεσης κάθε εργασίας σε κάθε μηχάνημα.
2) **Αντιστοίχιση Εργασιών στο Gantt:**
   - Η συνάρτηση αντιστοιχίζει κάθε εργασία σε ένα ξεχωριστό χρωματισμένο κουτάκι στο γράφημα Gantt.
   - Οι εργασίες τοποθετούνται στην κατάλληλη θέση στον άξονα Y (μηχανές) και στην κατάλληλη χρονική στιγμή στον άξονα X, σύμφωνα με τον πίνακα C_table.

3) **Χρήση Χρωμάτων για Κάθε Εργασία:**
    - Χρησιμοποιούνται διάφορα χρώματα για κάθε εργασία, ώστε να είναι ευανάγνωστο το γράφημα και να διακρίνονται εύκολα οι διάφορες εργασίες.
4) **Εμφάνιση Διαγράμματος:**
    - Το διάγραμμα Gantt παράγεται με χρήση της βιβλιοθήκης Matplotlib.
    - Εμφανίζονται τα χρονικά διαστήματα κάθε εργασίας σε κάθε μηχάνημα.

Έτσι, κατά την εκτέλεση του προγράμματος για κάθε αρχείο εισόδου, προβάλλεται ένα γράφημα Gantt που απεικονίζει το πρόγραμμα εκτέλεσης των εργασιών, καθιστώντας ευανάγνωστο και κατανοητό το αποτέλεσμα του αλγορίθμου Flowshop Scheduling.

Ερώτημα 3.5

Σύμφωνα με την αναφορά "Some efficient heuristic methods for the flow shop sequencing problem" τα βήματα του NEH αλγορίθμου είναι τα εξής:

1) Ταξινόμηση των n εργασιών με βάση τα φθίνοντα αθροίσματα των χρόνων επεξεργασίας στις μηχανές.
2) Προγραμματισμός των δύο πρώτων εργασιών (schedule) με σκοπό την ελαχιστοποίηση του current makespan σαν να υπήρχαν μόνο αυτές οι δύο εργασίες.
3) Για k=3 έως n τότε:
4) Τοποθέτηση της k-οστής εργασίας στη θέση, μεταξύ των k πιθανών, που ελαχιστοποιεί τη μερική διάρκεια (current makespan).

Όσον αφορά την πολυπλοκότητα σε αυτόν τον αλγόριθμο, στο Βήμα 1 είναι **O(n log(n))** και στο Βήμα 2 είναι **O(m).** Για τον υπολογισμό του current makespan στο Βήμα 4 χρειάστηκαν **O(km)** πράξεις.

Εδώ, στο πρόγραμμα **`neh_square.py`** η συγκεκριμένη υλοποίηση επιτυγχάνει τη βελτιστοποίηση του αλγορίθμου NEH, πετυχαίνοντας πολυπλοκότητα O(n^2m). Η κύρια λειτουργία του αλγορίθμου περιγράφεται στη συνάρτηση **neh_square()**, όπου εκτελείται για κάθε αρχείο εισόδου (πρόβλημα) με τη χρήση της συνάρτησης **run_neh_square()**. Η αποτελεσματικότητα και η απόδοση του αλγορίθμου αξιολογούνται μέσω της εκτύπωσης των βέλτιστων διατάξεων, των χρόνων εκτέλεσης και άλλων στατιστικών πληροφοριών για κάθε πρόβλημα. Επιπλέον, ο κώδικας παρέχει και τη δυνατότητα οπτικοποίησης της βέλτιστης διάταξης με τη χρήση γραφήματος Gantt μέσω της συνάρτησης **display_schedule()**, που επισημαίνει την πορεία της κάθε εργασίας σε κάθε μηχάνημα.

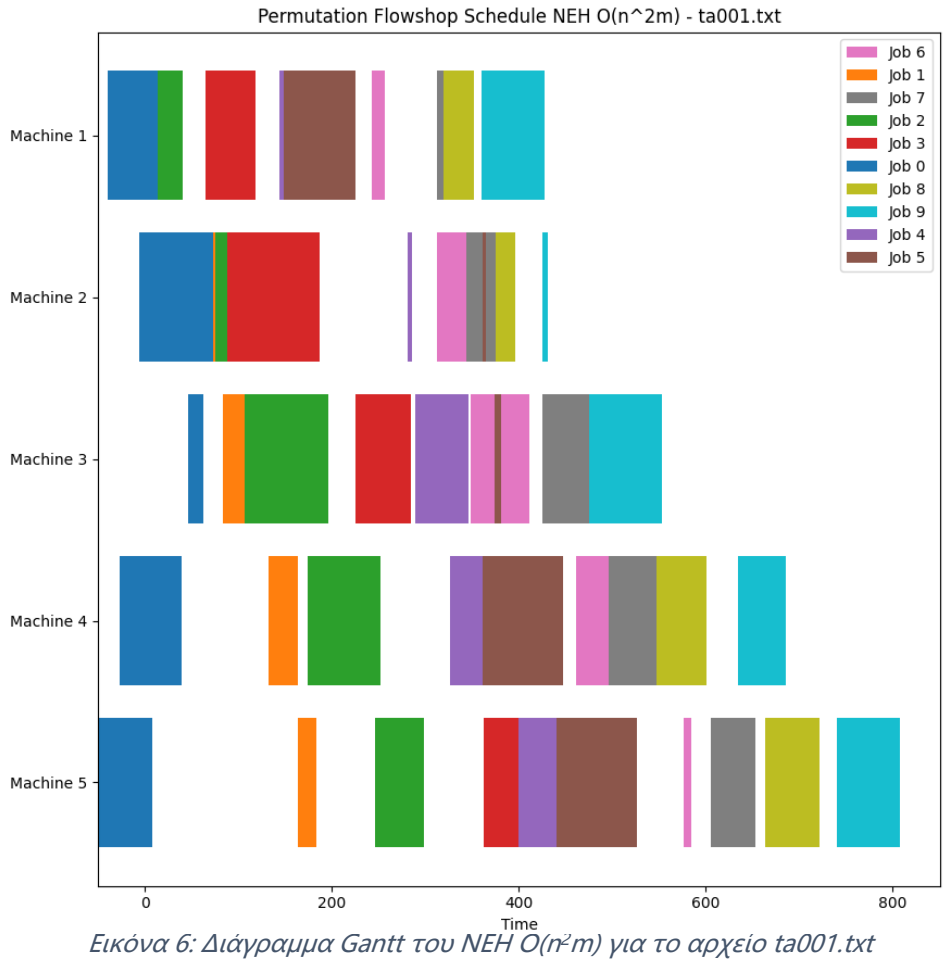
Είναι σημαντικό να τονιστεί όμως, ότι η βελτιωμένη υλοποίηση του NEH αλγορίθμου προσφέρει αξιοσημείωτη απόδοση και αποτελεσματικότητα,

καθιστώντας τον αλγόριθμο κατάλληλο για εφαρμογές μεγάλης κλίμακας στον τομέα της χρονοπρογραμματισμού παραγωγικών διεργασιών.

Παρακάτω παρουσιάζονται δύο συγκεκριμένα παραδείγματα εκτέλεσης του κώδικα και των διαγραμμάτων Gantt, για το αρχείο ta001.txt και για το αρχείο ta020.txt :

```
C:\Users\penyv\Desktop\DIT\Μεταπτυχιακό\Αλγόριθμοι & Προχωρημένες Δομές Δεδομένων\AADS_Vinni_170\code>python neh_square.
py

File: ./Taillard-PFSP/ta001.txt

Jobs from File: [[54, 79, 16, 66, 58], [15, 11, 49, 31, 20], [77, 56, 89, 78, 53], [53, 99, 60, 13, 53], [27, 5, 57, 49,
 69], [76, 3, 7, 85, 86], [14, 73, 63, 39, 8], [12, 47, 63, 56, 47], [32, 21, 26, 54, 58], [68, 5, 77, 51, 68]]

NEH Square O(n^2m) Optimal Jobs Order (Schedule):
 [6, 1, 7, 2, 3, 0, 8, 9, 4, 5]

NEH Square O(n^2m) Processing Time (Makespan): 808

Execution Time for ta001.txt file: 0.00019340000000001023 seconds


------------------------------------------------------------------------------


Total Execution Time: 0.0040280818939208984 seconds
```

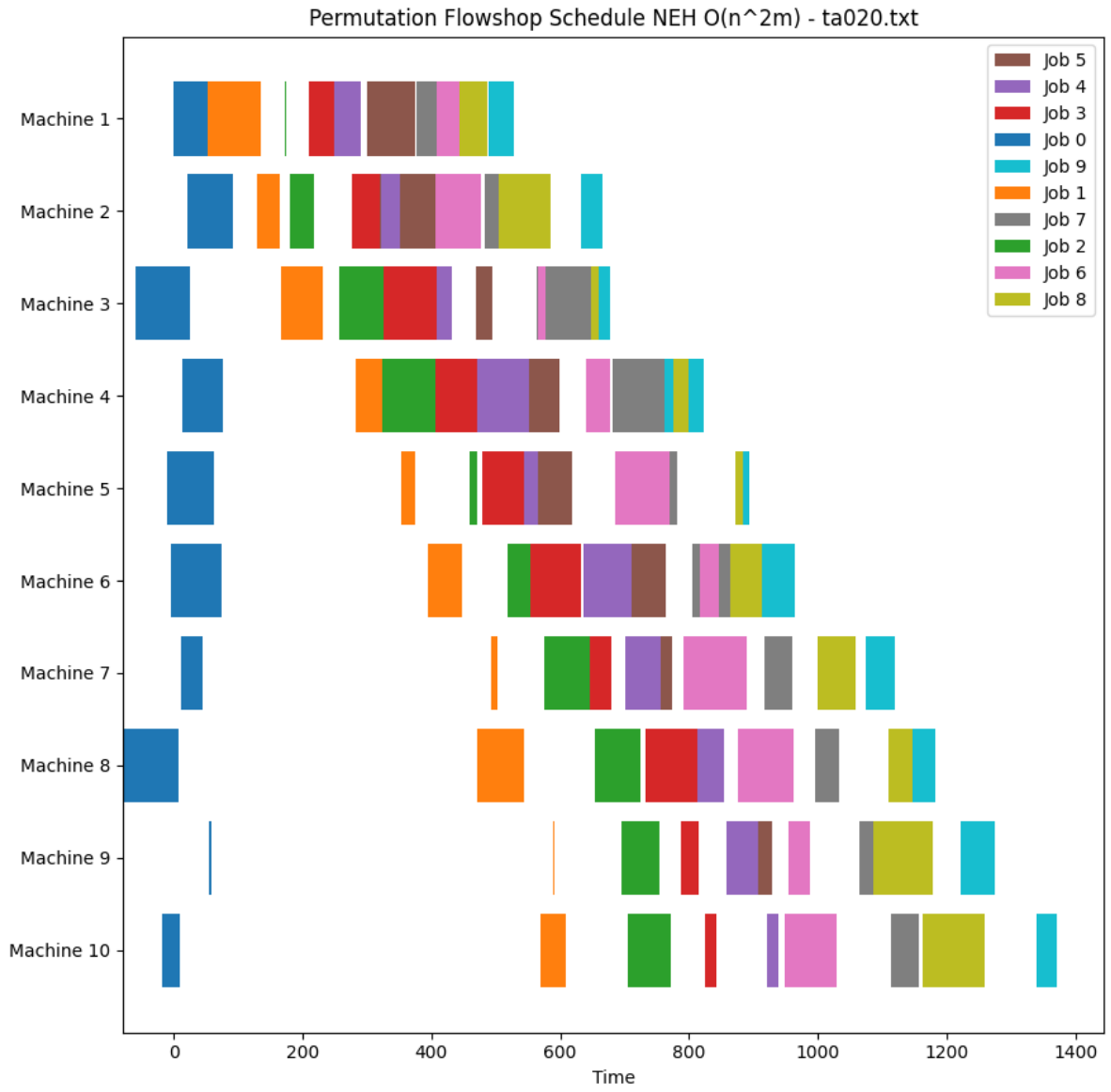
Εικόνα 5: Εκτέλεση NEH O(n²m) για το αρχείο ta001.txt



Εικόνα 6: Διάγραμμα Gantt του NEH O(n²m) για το αρχείο ta001.txt

```
Microsoft Windows [Version 10.0.22621.3007]
(c) Microsoft Corporation. All rights reserved.

C:\Users\penyv\Desktop\DIT\Μεταπτυχιακό\Αλγόριθμοι & Προχωρημένες Δομές Δεδομένων\AADS_Vinni_170\code>python neh_square.
py

File: ./Taillard-PFSP/ta020.txt

Jobs from File: [[74, 70, 84, 63, 72, 78, 33, 87, 3, 28], [84, 36, 64, 46, 21, 53, 10, 71, 3, 39], [3, 37, 69, 83, 12, 3
6, 70, 71, 57, 66], [39, 42, 94, 78, 65, 83, 92, 79, 28, 18], [62, 29, 66, 97, 47, 73, 54, 41, 48, 17], [74, 91, 25, 76,
 62, 74, 44, 8, 58, 9], [37, 71, 12, 38, 84, 31, 99, 87, 33, 80], [72, 31, 83, 80, 91, 78, 43, 37, 25, 42], [43, 80, 12,
 24, 11, 51, 60, 36, 93, 96], [41, 32, 22, 80, 14, 78, 45, 42, 53, 31]]

NEH Square O(n^2m) Optimal Jobs Order (Schedule):
 [5, 4, 3, 0, 9, 1, 7, 2, 6, 8]

NEH Square O(n^2m) Processing Time (Makespan): 1371

Execution Time for ta020.txt file: 0.0003103000000000966 seconds


------------------------------------------------------------------------


Total Execution Time: 0.0030074119567871094 seconds
```

Εικόνα 7: Εκτέλεση NEH O(n2m) για το αρχείο ta020.txt



Εικόνα 8: Διάγραμμα Gantt του NEH O(n2m) για το αρχείο ta020.txt

Ερώτημα 3.6

Στην προσέγγισή μου για το πρόβλημα PFSP, επιλέγω να υλοποιήσω τον αλγόριθμο **Tabu Search**, έναν βελτιωμένο μεταευριστικό (metaheuristic) αλγόριθμο που επιδιώκει την εύρεση βέλτιστων λύσεων σε προβλήματα συνδυαστικής βελτιστοποίησης.

Ο αλγόριθμος Tabu Search λειτουργεί με τη χρήση μιας λίστας "Tabu" που περιέχει κινήσεις που έχουν προσφάτως εξεταστεί, αποφεύγοντας έτσι την επανάληψη των ίδιων κινήσεων. Επιπλέον, χρησιμοποιεί μηχανισμούς όπως η λειτουργία κόστους για την επιλογή βέλτιστων κινήσεων.

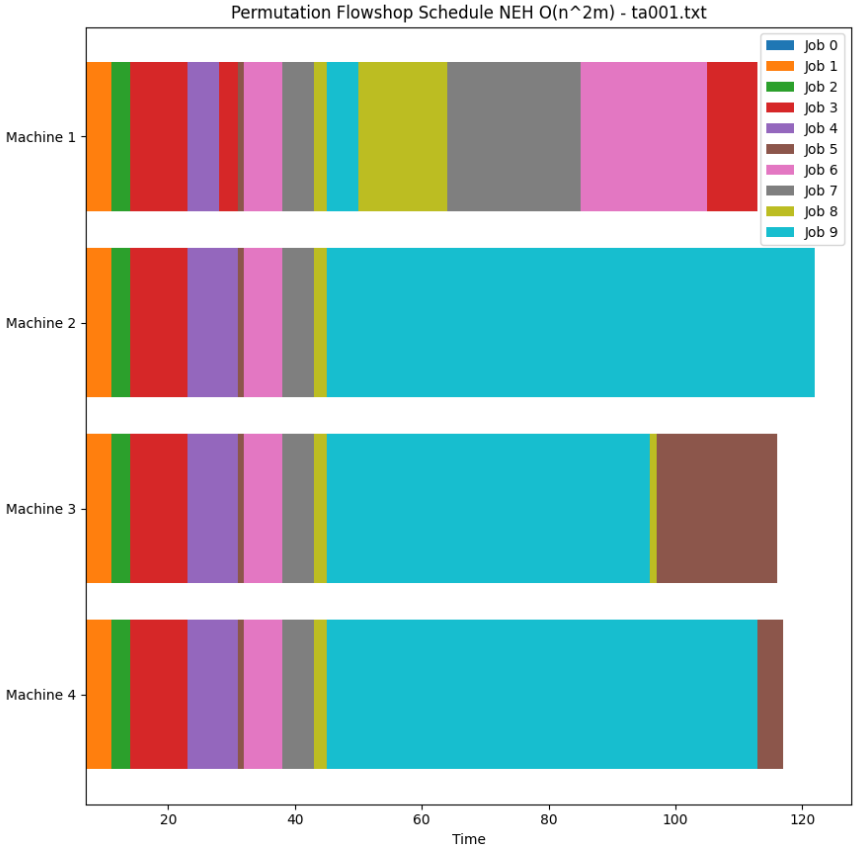
Στην πράξη, ο αλγόριθμος Tabu Search ξεκινάει από μια αρχική λύση και εξερευνά διάφορες γειτονικές λύσεις. Οι καλύτερες λύσεις διατηρούνται, ακόμη κι αν αυτές προσθέτουν κινήσεις που βρίσκονται στη λίστα Tabu. Ο αλγόριθμος εξερευνά διάφορες περιοχές του χώρου λύσεων, προσπαθώντας να αποφύγει τον εγκλωβισμό.

```
Microsoft Windows [Version 10.0.22621.3007]
(c) Microsoft Corporation. All rights reserved.

C:\Users\penyv\Desktop\DIT\Μεταπτυχιακό\Αλγόριθμοι & Προχωρημένες Δομές Δεδομένων\AADS_Vinni_170\code>python tabu_algori
thm.py
File: ./Taillard-PFSP/ta001.txt
Tabu Search Optimal Schedule: [6, 8, 7, 2, 3, 4, 0, 1, 9, 5]
Tabu Search Makespan: 522
Processing times list is: [[79, 16, 66, 58], [11, 49, 31, 20], [56, 89, 78, 53], [99, 60, 13, 53], [5, 57, 49, 69], [3,
7, 85, 86], [73, 63, 39, 8], [47, 63, 56, 47], [21, 26, 54, 58], [5, 77, 51, 68]]

Execution Time for ta001.txt file: 0.007394900000000093 seconds
--------------------------------------------------------------------------------


Total Execution Time: 0.021062612533569336 seconds
```

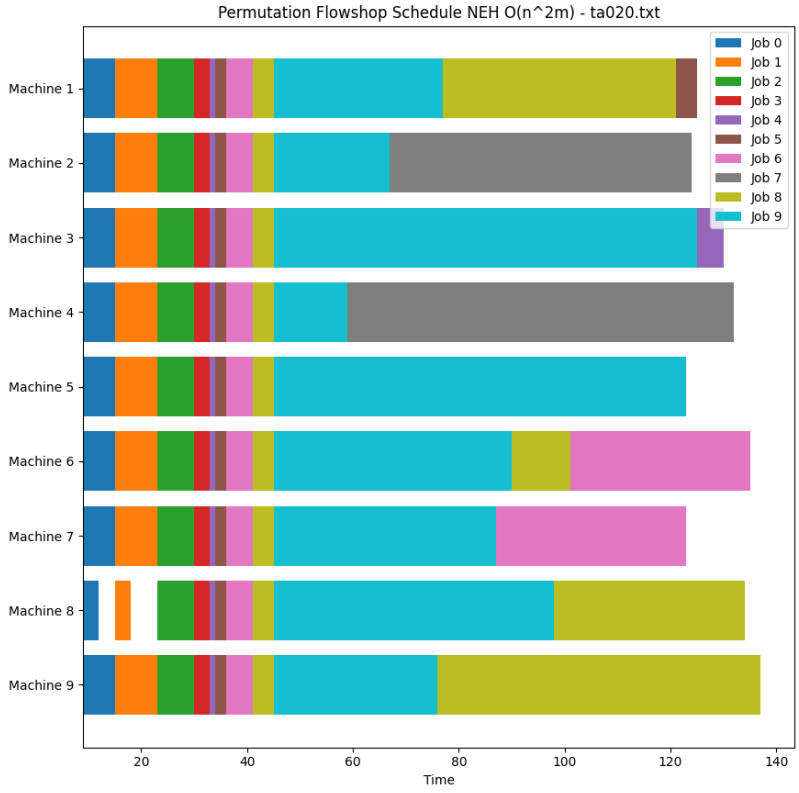
Εικόνα 9: Εκτέλεση Tabu Search για το αρχείο ta001.txt

```
C:\Users\penyv\Desktop\DIT\Μεταπτυχιακό\Αλγόριθμοι & Προχωρημένες Δομές Δεδομένων\AADS_Vinni_170\code>python tabu_algori
thm.py
File: ./Taillard-PFSP/ta020.txt
Tabu Search Optimal Schedule: [8, 6, 1, 2, 3, 7, 9, 5, 4, 0]
Tabu Search Makespan: 665
Processing times list is: [[70, 84, 63, 72, 78, 33, 87, 3, 28], [36, 64, 46, 21, 53, 10, 71, 3, 39], [37, 69, 83, 12, 36
, 70, 71, 57, 66], [42, 94, 78, 65, 83, 92, 79, 28, 18], [29, 66, 97, 47, 73, 54, 41, 48, 17], [91, 25, 76, 62, 74, 44,
8, 58, 9], [71, 12, 38, 84, 31, 99, 87, 33, 80], [31, 83, 80, 91, 78, 43, 37, 25, 42], [80, 12, 24, 11, 51, 60, 36, 93,
96], [32, 22, 80, 14, 78, 45, 42, 53, 31]]

Execution Time for ta020.txt file: 0.021835999999999967 seconds
--------------------------------------------------------------------------------


Total Execution Time: 0.055047273635864426 seconds
```

Εικόνα 10: Εκτέλεση Tabu Search για το αρχείο ta020.txt



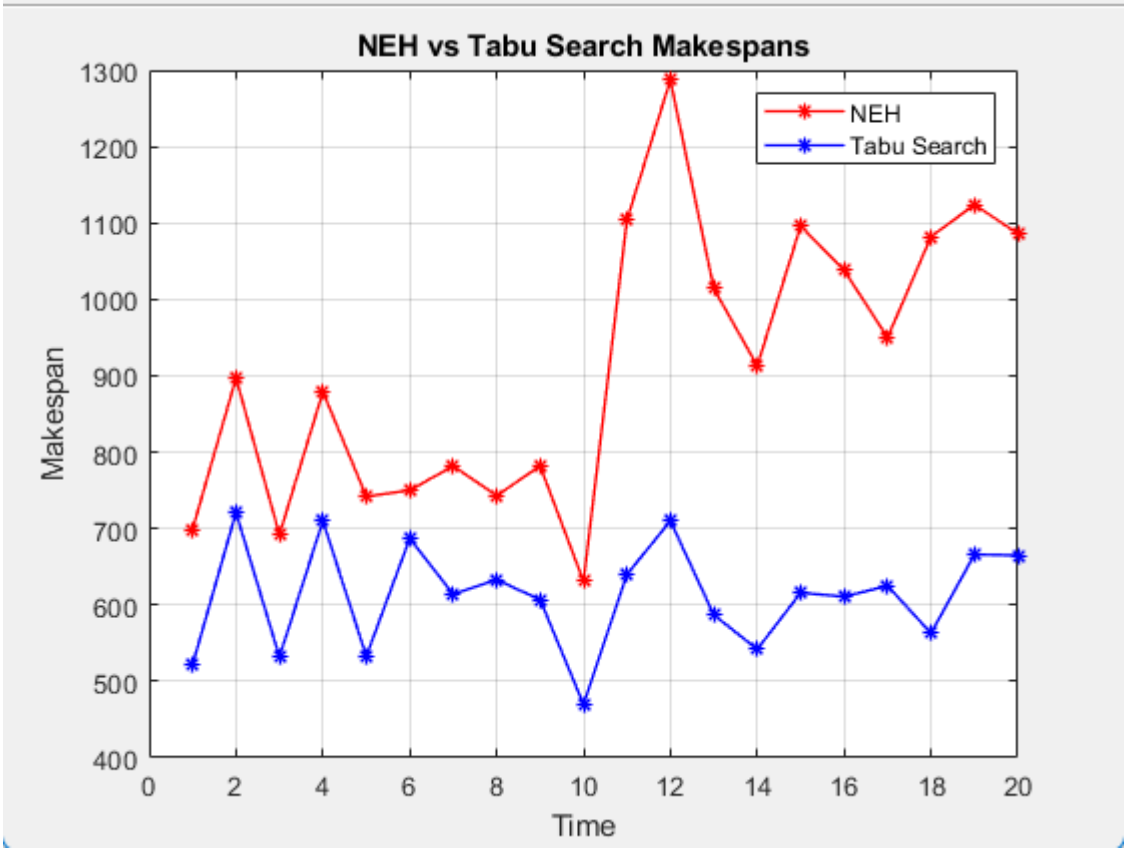
Εικόνα 11: Διάγραμμα Gantt του Tabu για το αρχείο ta001.txt



Εικόνα 12: Διάγραμμα Gantt του Tabu για το αρχείο ta020.txt

Για τη σύγκριση των αποτελεσμάτων του Tabu Search με αυτά του NEH, εκτελώ τους δύο αλγορίθμους για τα ίδια προβλήματα PFSP και αξιολογώ την απόδοσή τους βάσει του makespan. Προσέγγιση του ποιοτικού και ποσοτικού κριτηρίου των λύσεων μεταξύ των δύο αλγορίθμων θα με διαφωτίσει σχετικά με την αποτελεσματικότητά τους για το συγκεκριμένο πρόβλημα. Εδώ, χρησιμοποιήθηκε το **Matlab** για πιο λεπτομερής ανάλυση των αποτελεσμάτων στα γραφήματα. Τα δεδομένα για το makespan του NEH αλγορίθμου, καθώς και εκείνα του Tabu Αλγορίθμου, αποθηκευόταν αυτόματα κατά την εκτέλεση του κάθε κώδικα αντίστοιχα σε ένα αρχείο με το όνομα **`makespan_neh_algorithm.txt`** και **`makespan_tabu_algorithm.txt`** αντίστοιχα. Ο κώδικας είναι ο εξής:

```
close all, clear, clc;

% NEH Algorithm O(n^3m)
neh_makespan = [699 897 692 879 742 750 782 743 782 631
1104 1287 1014 913 1096 1038 950 1082 1124 1086];

% Tabu Search Algorithm
tabu_makespan = [522 721 532 710 533 687 614 633 607 470
640 710 587 542 616 611 625 563 666 665];


% Plot for NEH vs Tabu
figure;
plot(neh_makespan, 'r*-', 'LineWidth', 1, 'DisplayName',
'NEH');
hold on;
plot(tabu_makespan, 'b*-', 'LineWidth', 1, 'DisplayName',
'Tabu');
title('NEH vs Tabu Search Makespans');
xlabel('Time');
ylabel('Makespan');
grid on;
legend('NEH','Tabu Search');
```



Εικόνα 13: Σύγκριση των Makespans του NEH με του Tabu Search

Εκτέλεση Αρχείων

Η εκτέλεση της υλοποίησης των αλγορίθμων για την επίλυση του προβλήματος PFSP πραγματοποιήθηκε με χρήση της γλώσσας προγραμματισμού Python. Πριν από την εκτέλεση των αλγορίθμων, οι χρήστες πρέπει να σιγουρευτούν ότι έχετε εγκαταστήσει όλες τις απαιτούμενες βιβλιοθήκες Python που χρησιμοποιούνται στον κώδικα, χρησιμοποιώντας τον διαχειριστή πακέτων της Python (π.χ. pip).

Οι χρήστες μπορούν να εκτελέσουν τον κώδικα μέσω του τερματικού (Command Window) ή του περιβάλλοντος εκτέλεσης Python. Είναι σημαντικό να παρέχουν τις κατάλληλες παραμέτρους και διαδρομές αρχείων, όπως η τοποθεσία των στιγμιοτύπων των προβλημάτων PFSP. Πιο συγκεκριμένα:

- Εκτέλεση neh_algorithm.py : **`python neh_algorithm.py`**
- Εκτέλεση neh_square.py : **`python neh_square.py`**
- Εκτέλεση tabu_algorithm.py : **`python tabu_algorithm.py`**

Ακόμη, όλα τα αρχεία της συγκεκριμένης εργασίας είναι ανεβασμένα στο public repository στο GitHub : https://github.com/penyvinni/AADS_PFSP_170.
Η διαδικασία που ακολουθήθηκε είναι η εξής:

- Δημιουργία Νέου Αποθετηρίου (Repository) στο GitHub
- Προετοιμασία του Τοπικού Αποθετηρίου (Local Repository) μέσω του Git Bash.
- Εκτέλεση των εντολών:

```
git init  # Αρχικοποιήστε ένα νέο Git αποθετήριο
git add . # Προσθέστε όλα τα αρχεία στον προσωρινό
αποθετήριο (staging area)
git commit -m "First Commit"  # Δημιουργήστε ένα πρώτο
commit
```

- Σύνδεση του Τοπικού με το Κοινόχρηστο Αποθετήριο με την εντολή:

```
git remote add origin <repository_url>
```

- Ανέβασμα των Αλλαγών στο GitHub με την εντολή :

```
git push -u origin master  # Ανεβάστε τις αλλαγές στο
GitHub
```

Συμπεράσματα

Συμπερασματικά, στο πλαίσιο της συγκεκριμένης εργασίας, προσφέρθηκε μια εκτενής μελέτη επίλυσης του PFSP, εξετάζοντας διάφορες προσεγγίσεις. Αρχικά, προσεγγίθηκε το πρόβλημα με τον ευρετικό αλγόριθμο NEH, αποτυπώνοντας τις λύσεις μέσω γραφημάτων Gantt για καλύτερη κατανόηση. Εν συνεχεία, υλοποιήθηκε ή βελτίωση του αλγορίθμου όπως προτείνεται στη βιβλιογραφία, μειώνοντας σημαντικά την πολυπλοκότητά του. Η ανάλυση χρόνων εκτέλεσης κατέδειξε την αποτελεσματικότητα της βελτιωμένης προσέγγισης.

Επιπλέον, έγινε πρόταση μια προσωπικής προσέγγισης για την επίλυση του προβλήματος, η οποία συγκρίθηκε με τα αποτελέσματα του NEH. Η σύγκριση αυτή αποκάλυψε τα πλεονεκτήματα και τα χαρακτηριστικά κάθε προσεγγίσεως, ενισχύοντας την κατανόηση των δυνατοτήτων και των περιορισμών των αλγορίθμων που εξετάστηκαν. Η προσέγγιση αυτή συμβάλλει στην εμβάθυνση της γνώσης μας σχετικά με τις προκλήσεις του PFSP και ενισχύει την ευελιξία του αναλυτικού πλαισίου για παρόμοια προβλήματα χρονοπρογραμματισμού στο μέλλον.