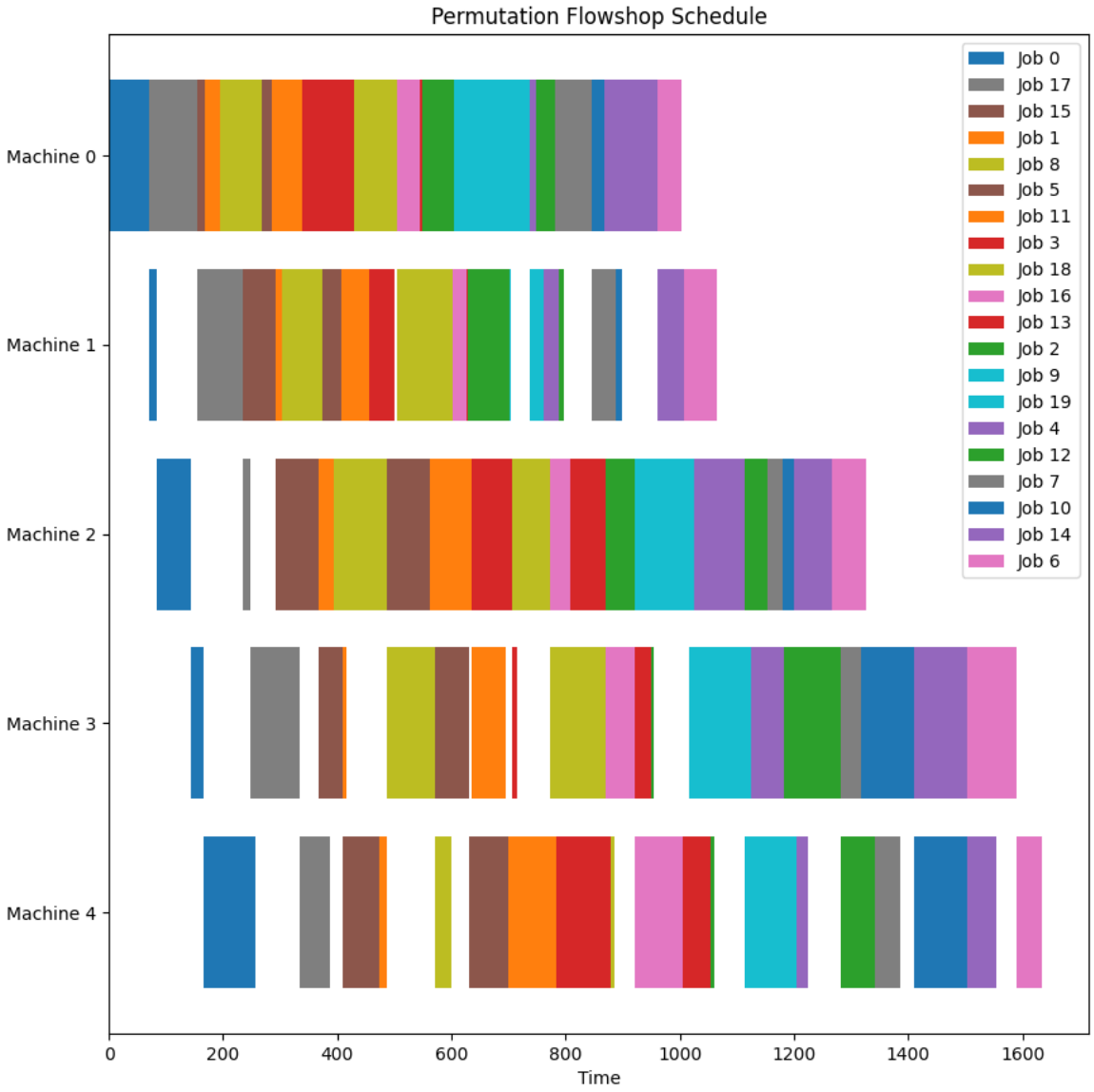
Πανεπιστήμιο Ιωαννίνων Τμήμα Πληροφορικής & Τηλεπικοινωνίων

Πρόγραμμα Μεταπτυχιακών Σπουδών "Πληροφορικής & Δικτύων"

Μάθημα : Αλγόριθμοι & Προχωρημένες Δομές Δεδομένων

Θέμα Εργασίας : Πρόβλημα PFSP



Ονοματεπώνυμο : Περσεφόνη Πατσέα

Α.Μ. : 185

1ο Εξάμηνο

Ημερομηνία Παράδοσης : 13 Ιανουαρίου 2024

Πίνακας περιεχομένων

ΠΕΡΙΛΗΨΗ
Η αντιμετώπιση του Permutation Flow-Shop Scheduling Problem (PFSP), αφορά τον προγραμματισμό χρόνου εκτέλεσης ενός συνόλου εργασιών σε ένα σύνολο μηχανών ούτως ώστε να επιτευχθεί ο συντομότερος χρόνος ολοκλήρωσης για την εργασία που ολοκληρώνει την επεξεργασία της τελευταία. Σ' αυτή την τεχνική αναφορά, εξετάζεται η υλοποίηση αλγορίθμων και υποστηρικτικών δομών δεδομένων για την αποδοτική επίλυση του προβλήματος καθώς και η εμπειρική εκτίμηση της απόδοσης της προσέγγισης μέσω της επίλυσης δημόσια δημοσιευμένων στιγμιότυπων προβλημάτων.

1. Εισαγωγή
Το πρόβλημα PFSP μπορεί να περιγραφεί απλά ως εξής:

Υπάρχουν n εργασίες (jobs) και m μηχανήματα (machines). Κάθε εργασία πρέπει να περάσει από όλες τις μηχανές με την ίδια σειρά (από την πρώτη μέχρι την τελευταία μηχανή) και επιπλέον κάθε μηχανή πρέπει να επεξεργάζεται την ίδια διάταξη (permutation) εργασιών με όλες τις άλλες μηχανές. Ο περιορισμός αυτός μπορεί να επιβάλλεται αν κάθε εργασία παράγει σταδιακά ένα προϊόν σε μια γραμμή παραγωγής με ιμάντα, που δεν επιτρέπει σε μια εργασία να πάρει τη σειρά μιας άλλης εργασίας σε κάποιο από τα μηχανήματα. Ο χρόνος επεξεργασίας κάθε εργασίας, σε κάθε μηχανή δίνεται, και έτσι ζητείται να βρεθεί η διάταξη των εργασιών που ελαχιστοποιεί το λεγόμενο makespan, δηλαδή το χρόνο ολοκλήρωσης της εργασίας που ολοκληρώνει την επεξεργασία της τελευταία.

2. Αποτελέσματα
Για την συγγραφή των πειραμάτων χρησιμοποιήθηκε η **Python 3.10.0** και το Visual Studio Code. Τα χαρακτηριστικά του υπολογιστή είναι i5 7600 (3.50 GHz), 16.0 GB RAM. Για την οπτικοποίηση των αποτελεσμάτων χρησιμοποιήθηκε η βιβλιοθήκη matplotlib, η seaborn και η tkinter.

Οδηγίες Εκτέλεσης του Κώδικα:
Για να εκτελέσουμε τον κώδικα, κάνουμε τα εξής βήματα:

i) Εκκινούμε το Visual Studio Code
ii) Ανοίγουμε το φάκελο που είναι αποθηκευμένοι οι κώδικες (**File** → **Open Folder**)
iii) Διαλέγουμε τον κώδικα που επιθυμούμε να τρέξουμε

iv) Πατάμε από το πληκτρολόγιο τα εξής κουμπιά (**CTRL + SHIFT + C**) προκειμένου να ανοίξει το Command Window

v) Όταν ανοίξει εισάγουμε την εντολή:

python file_name.py

Όπου **file_name**, το όνομα του εκάστοτε κώδικα που θέλουμε να εκτελέσουμε

Ειδικότερα, για τους κώδικες που έχουν δημιουργηθεί, οι εντολές εκτέλεσης είναι οι εξής:

i) Για τον κώδικα: **pfsp.py** → **python pfsp.py**
ii) Για τον κώδικα: **window.py** → **python window.py**
iii) Για τον κώδικα: **NEH.py** → **python NEH.py**
iv) Για τον κώδικα: **On2m.py** → **python On2m.py**
v) Για τον κώδικα: **tabu.py** → **python tabu.py**

3. Απαντήσεις Ερωτημάτων Εργασίας

3.1. Ορίστε το πρόβλημα τυπικά (αυστηρά), συμπεριλαμβανομένων όλων των παραμέτρων. Σχολιάστε τη σημασία του προβλήματος σε πραγματικές εφαρμογές.

Το πρόβλημα αλληλουχίας της κατάστασης ροής, συγκροτεί ένα πρόβλημα, όσον αφορά τον προγραμματισμό μίας παραγωγής, η οποία αποτελείται από n (jobs) και m (machines). Πιο συγκεκριμένα, τα jobs, ειδάλλως αντικείμενα, εργασίες, κ.ο.κ., οφείλουν να διεκπεραιωθούν με την ίδια σειρά σε machines. Ο τύπος που ορίζεται για το χρόνο επεξεργασίας της εργασίας i στη μηχανή j, είναι ο κάτωθι:

$$t_{ij}(i = 1, \dots, n; j = 1, \dots, m)$$

Οι χρόνοι αυτοί είναι μη αρνητικοί, σταθεροί και ορίζονται από τους χρόνους οι οποίοι ενδεχομένως να είναι μηδενικοί, εφόσον κάποια εργασία δε δέχεται επεξεργασία από μία μηχανή.

Το πρόβλημα συνίσταται στην ελαχιστοποίηση του χρόνου, τη λεγόμενη ακολουθία (makespan). Δηλαδή έγκειται μεταξύ της έναρξης της εκτέλεσης της πρώτης εργασίας στην πρώτη μηχανή, και της ολοκλήρωσης της εκτέλεσης της τελευταίας εργασίας στην τελευταία μηχανή. Για το εν λόγω πρόβλημα συνίστανται οι παρακάτω παραδοχές:

- Κάθε εργασία διεκπεραιώνεται το πολύ μία φορά στις μηχανές 1, 2 , ... , m (με αυτήν ακριβώς τη σειρά).

- Ο αριθμός των εργασιών που δέχεται επεξεργασία από κάθε μηχανή, κάθε φορά, είναι μόνο μία.
- Κάθε φορά, κάθε εργασία, υπόκειται σε επεξεργασία, το πολύ σε ένα μηχάνημα.
- Οι εργασίες δε δύναται να διακοπούν.
- Οι χρόνοι προετοιμασίας των εργασιών περιλαμβάνονται στο χρόνο επεξεργασίας και δεν εξαρτώνται από την ακολουθία.
- Οι ακολουθίες λειτουργίας των εργασιών είναι οι εξής: ίδιες σε κάθε μηχάνημα και οι κοινές οφείλουν να προσδιορίζονται.

Συνοψίζοντας, το εν λόγω πρόβλημα είναι NP-hard, ειδάλλως μη ντετερμινιστική πολυωνυμική σκληρότητα χρόνου, και μπορεί να επιλυθεί μόνο για μικρά μεγέθη. Τέλος, αποτελείται από την εύρεση μιας ακολουθίας που ελαχιστοποιεί το makespan, το λεγόμενο M(σ). Επομένως, ο αριθμός των πιθανών χρονοδιαγραμμάτων είναι n!, τουτέστιν 1 x 2 x … x n.

Η σημασία του προβλήματος σε πραγματικές εφαρμογές, έχει την κάτωθι σημασία:

Δίνονται 5 jobs: **A, B, C, D, E**

1) Ταξινόμηση της λίστας των job
   Ταξινομημένη Λίστα: **A, E, B, C, D**

2) Επιλογή των A & E, με πιθανές ακολουθίες τις A → E ή E → A.
   Εντοπίζεται η καλύτερη ακολουθία, ανάμεσα σ' αυτές τις δύο.
   Αν η καλύτερη είναι η **E → A**, δεν αλλάζει θέση ξανά.

3) Επιλογή του B και εισαγωγή του στις 3 πιθανές θέσεις:
   B → E → A ,
   E → B → A ,
   E → A → B.
   Εντοπίζεται η καλύτερη: **B → E → A**

4) Επιλογή του C και εισαγωγή του στις 4 πιθανές θέσεις:
   C → B → E → A ,
   B → C → E → A ,
   B → E → C → A ,
   B → E → A → C.
   Εντοπίζεται η καλύτερη: **B → E → A → C**

5) Επιλογή του D και εισαγωγή του στις 5 πιθανές θέσεις:
   D → B → E → A → C ,
   B → D → E → A → C ,
   B → E → D → A → C ,

B → E → A → D → C ,
B → E → A → C → D.
Εντοπίζεται η καλύτερη: **D → B → E → A → C**

Στην καθημερινή ζωή, η υλοποίηση του Αλγορίθμου NEH, χρησιμοποιείται στην επεξεργασία τροφίμων, στις χημικές παραγωγές, και στις βιομηχανίες γενικότερα, εξαιτίας της πολύ καλής λύσης που παρέχει, χωρίς όμως να είναι βέλτιστη.

Ο μαθηματικός τύπος που εκφράζει το συγκεκριμένο αλγόριθμο, είναι:

$$\boldsymbol{NEH \rightarrow \frac{n(n+1)}{2} sequences}$$

Αφού:

- 1 x 2 x 3 x 4 x … x n = n!
- 1 + 2 + 3 + 4 + … + n = $\frac{n(n+1)}{2}$

3.2. Διαβάστε τα στιγμιότυπα προβλημάτων από το [1].

Για την υλοποίηση του συγκεκριμένου υπό-ερωτήματος, υλοποιήθηκαν οι κώδικες **pfsp.py** και **window.py**.

Στον κώδικα **pfsp.py**, υλοποιείται ένας απλός αλγόριθμος εύρεσης λύσης, ο οποίος δημιουργεί μία τυχαία διάταξη εργασιών. Ο συγκεκριμένος απλοϊκός επιλυτής, δέχεται δύο παραμέτρους: τον αριθμό των εργασιών και έναν προαιρετικό για την αρχικοποίηση με τυχαίους αριθμούς. Έτσι δημιουργείται μία τυχαία σειρά αριθμών που έχει εύρος τιμών από το 0 έως τον (αριθμό_εργασιών – 1), και κατόπιν αυτή η σειρά αριθμών αντιστοιχίζεται σε μία πιθανή λύση για το εν λόγω πρόβλημα.

Στον κώδικα **window.py**, δημιουργείται ένα παράθυρο, όπου κατά την εκτέλεσή του, ο χρήστης επιθυμεί ποιο αρχείο dat επιθυμεί να διαβάσει, ούτως ώστε να οπτικοποιηθούν σε ένα γράφημα τα jobs και τα machines, αντίστοιχα.

3.3. Υλοποιήστε τον ευρετικό αλγόριθμο NEH (Nawaz, Enscore, Ham) [3] για την επίλυση του προβλήματος.

Το 1982, προτάθηκε από τους Nawaz, Enscore και Ham, η έννοια του ευρετικού αλγορίθμου δημιουργίας λύσης, για το λεγόμενο PFSP πρόβλημα, ειδάλλως το flow-shop πρόβλημα. Ο συγκεκριμένος αλγόριθμος βρίσκεται στη βιβλιογραφία και ως NEH, ονομασία που προκύπτει από τα αρχικά ονόματα των δημιουργών του.

Τα βήματα του εν λόγω αλγορίθμου, είναι τα εξής:

i) **1ο Βήμα:** Για κάθε εργασία i, υπολογίζεται ο συνολικός χρόνος επεξεργασίας που απαιτείται απ' όλες τις μηχανές m, τουτέστιν:
$$T_i = \sum_{j=1}^{m} t_{ij},$$
με t_{ij} να αντιστοιχίζεται στο χρόνο που η εργασία i στη μηχανή j.

ii) **2ο Βήμα:** Οι εργασίες ταξινομούνται με φθίνουσα σειρά βάσει του T_i.

iii) **3ο Βήμα:** Από το **2ο Βήμα**, αντλούνται οι δύο πρώτες εργασίες της λίστας, και κατόπιν εντοπίζεται η διάταξη των δύο συγκεκριμένων εργασιών, οι οποίες δίνουν το μικρότερο makespan. Κατόπιν του αλγορίθμου, δεν αλλάζει η σχετική θέση των δύο συγκεκριμένων εργασιών. Έτσι, θέτεται το $i=3$.

iv) **4ο Βήμα:** Επιλέγεται από τη λίστα του **2ου Βήματος**, η εργασία που τοποθετείται στη θέση i, και έτσι βρίσκεται η θέση της στην προηγούμενη διάταξη, μέσω των i δυνατών θέσεων, οι οποίες δίνουν το μικρότερο makespan για το σύνολο εργασιών i. Αξίζει να σημειωθεί πως δεν αλλάζει η σχετική θέση των εργασιών που έχουν απομείνει.

v) **5ο Βήμα:** Αν το $i=n$, τότε ο αλγόριθμος τερματίζει. Ειδάλλως, θέτεται $i=i+1$, και πραγματοποιείται ανακατεύθυνση στο **4ο Βήμα**.

Τα παραπάνω βήματα του αλγορίθμου NEH, έχουν υλοποιηθεί στον κώδικα ***NEH.py***. Ο συγκεκριμένος κώδικας υλοποιεί το δεδομένο ευρετικό αλγόριθμο για τη δημιουργία λύσης του προβλήματος Permutation Flow-Shop Scheduling Problem (PFSP).

Τα δεδομένα των αρχείων βρίσκονται σε μορφή .dat, και αντλούνται online από έναν public φάκελο στο GitHub, που βρίσκεται στον κάτωθι σύνδεσμο:

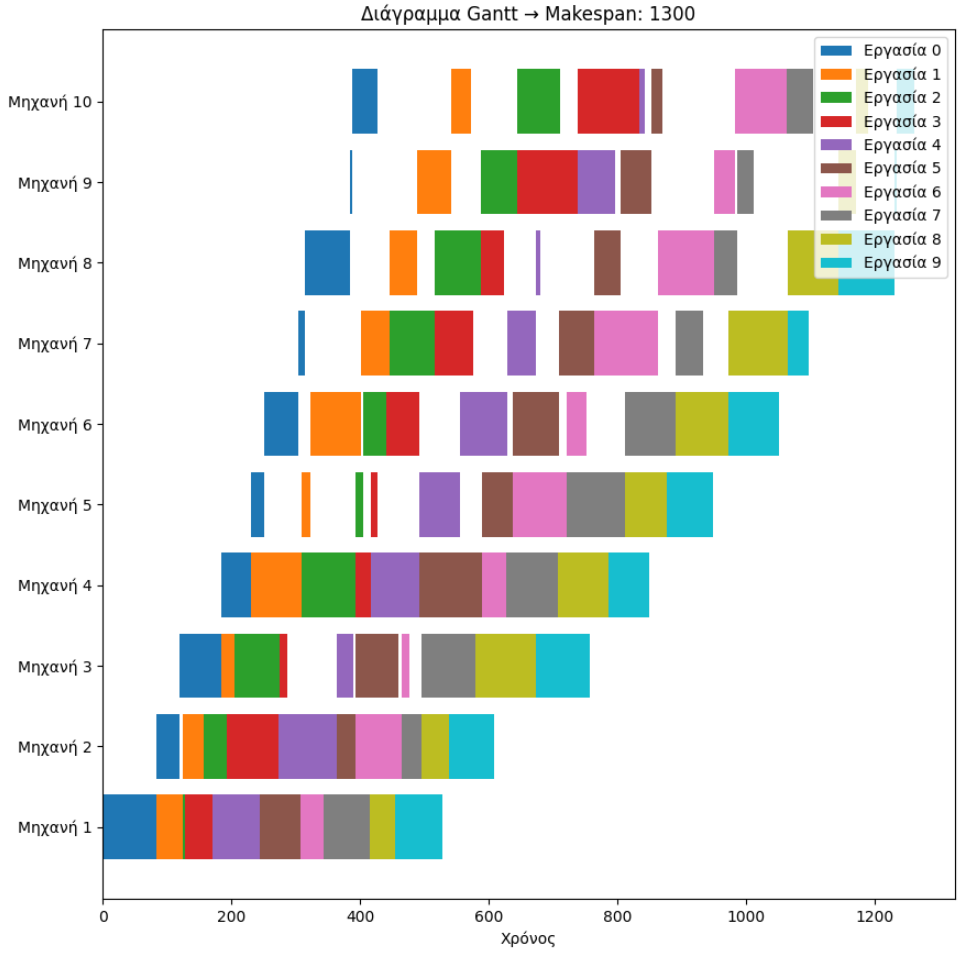
https://github.com/ppatsea/Algorithms

Επιπροσθέτως, κατά την εκτέλεση του κώδικα, εμφανίζονται στο Command Window *, τα εξής:

i) Βέλτιστη Διάταξη Εργασιών του αλγορίθμου NEH,
ii) Βέλτιστη Σειρά Εργασιών του αλγορίθμου NEH,
iii) Χρόνος Εκτέλεσης του αλγορίθμου NEH,
iv) Συνολικός Χρόνος Εκτέλεσης του κάθε αρχείου dat.

Μετά το πέρας εκτέλεσης των αρχείων, εμφανίζεται το τελικό Διάγραμμα Gantt των συγκεκριμένων αρχείων, και τέλος όταν ο χρήστης κλείσει το παράθυρο του γραφήματος, στο Command Window εμφανίζεται ο Συνολικός Χρόνος Εκτέλεσης όλου του Προγράμματος.

* : Ο χρήστης από τις κατάλληλες γραμμές κώδικα επιλέγει αν επιθυμεί τα αποτελέσματα να εμφανίζονται αποκλειστικά και μόνο στην οθόνη, ή εκτός από αυτό, να εγγράφονται αυτόματα και όχι χειροκίνητα, σε ένα αντίστοιχο txt αρχείο με την ονομασία **results_NEH.txt**.



Εικόνα 1: Διάγραμμα Gantt, αλγορίθμου NEH, για την 20η εργασία

```
File: ./Taillard-PFSP/ta013.txt

Βέλτιστη Διάταξη Εργασιών NEH: [[46, 19, 97, 1, 15, 62, 47, 39, 31, 2], [15, 59, 15, 46, 60, 47, 41, 38, 34, 22], [1, 24
, 55, 91, 72, 32, 26, 11, 94, 3], [49, 3, 60, 12, 75, 31, 70, 20, 88, 9], [85, 22, 2, 67, 41, 66, 7, 50, 4, 74], [65, 65
, 11, 43, 27, 34, 47, 64, 21, 83], [10, 26, 81, 73, 48, 1, 17, 68, 73, 75], [93, 54, 13, 55, 15, 31, 63, 38, 61, 90], [7
7, 89, 92, 12, 74, 19, 59, 74, 82, 57], [79, 51, 55, 25, 49, 98, 7, 65, 99, 89]]

Βέλτιστη Σειρά Εργασιών NEH: [9, 8, 7, 6, 5, 4, 3, 2, 0, 1]

Χρόνος Εκτέλεσης NEH: 1124

Συνολικός Χρόνος Εκτέλεσης Αρχείου: 0.0009944438934326172 seconds



File: ./Taillard-PFSP/ta014.txt

Βέλτιστη Διάταξη Εργασιών NEH: [[19, 18, 85, 44, 27, 24, 24, 40, 67, 19], [45, 83, 86, 3, 15, 8, 73, 6, 55, 8], [31, 87,
 21, 89, 61, 22, 13, 2, 36, 27], [54, 61, 9, 31, 7, 69, 58, 88, 8, 27], [31, 52, 77, 38, 4, 40, 50, 29, 88, 13], [82, 16
, 45, 85, 25, 85, 44, 17, 3, 30], [7, 55, 67, 79, 31, 39, 42, 13, 74, 42], [6, 43, 28, 83, 50, 19, 85, 12, 68, 66], [73,
 27, 85, 51, 33, 8, 95, 3, 42, 92], [94, 3, 39, 1, 63, 86, 44, 19, 55, 67]]

Βέλτιστη Σειρά Εργασιών NEH: [9, 8, 7, 6, 5, 4, 3, 2, 0, 1]

Χρόνος Εκτέλεσης NEH: 1052

Συνολικός Χρόνος Εκτέλεσης Αρχείου: 0.0009121894836425781 seconds



File: ./Taillard-PFSP/ta015.txt

Βέλτιστη Διάταξη Εργασιών NEH: [[18, 68, 9, 17, 28, 47, 24, 5, 50, 34], [13, 34, 52, 84, 66, 2, 40, 20, 7, 54], [90, 46,
 42, 34, 25, 69, 35, 7, 8, 40], [54, 87, 33, 87, 40, 5, 40, 50, 7, 49], [17, 12, 32, 87, 90, 93, 29, 61, 6, 31], [16, 11
, 55, 37, 89, 21, 56, 58, 86, 55], [64, 48, 30, 84, 9, 93, 71, 72, 33, 85], [51, 24, 39, 59, 81, 95, 63, 97, 35, 46], [5
2, 87, 30, 51, 90, 92, 99, 19, 62, 28], [65, 92, 83, 31, 98, 13, 20, 41, 81, 72]]

Βέλτιστη Σειρά Εργασιών NEH: [9, 8, 7, 6, 5, 4, 3, 2, 0, 1]

Χρόνος Εκτέλεσης NEH: 1262

Συνολικός Χρόνος Εκτέλεσης Αρχείου: 0.0009989738464355469 seconds



File: ./Taillard-PFSP/ta016.txt

Βέλτιστη Διάταξη Εργασιών NEH: [[36, 39, 46, 58, 36, 46, 14, 23, 65, 30], [34, 74, 35, 69, 33, 28, 47, 34, 41, 1], [85,
20, 23, 22, 72, 27, 76, 13, 93, 25], [47, 80, 51, 63, 60, 70, 39, 33, 5, 9], [99, 1, 17, 19, 2, 76, 57, 89, 97, 5], [66,
 37, 35, 47, 13, 56, 66, 59, 72, 37], [29, 22, 73, 35, 39, 66, 90, 65, 47, 27], [42, 2, 71, 94, 51, 98, 58, 6, 46, 42],
[28, 3, 97, 62, 5, 66, 95, 75, 55, 70], [77, 85, 18, 72, 67, 44, 56, 1, 90, 14]]

Βέλτιστη Σειρά Εργασιών NEH: [9, 8, 7, 6, 5, 4, 3, 2, 0, 1]

Χρόνος Εκτέλεσης NEH: 1224

Συνολικός Χρόνος Εκτέλεσης Αρχείου: 0.000997304916381836 seconds
```

Εικόνα 2: Command Window, αλγορίθμου NEH, για τις εργασίες 1-20

3.4. Υλοποιήστε την απεικόνιση λύσεων προγραμμάτων ως γραφήματα με το χρόνο στον άξονα x, τις μηχανές στον άξονα y και τις εργασίες ως επιμέρους κουτάκια στις κατάλληλες θέσεις.

Η υλοποίηση απεικόνισης 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, επιτεύχθηκε με τη χρήση της βιβλιοθήκης της Python, τη **Matplotlib**. Όσον αφορά τη Matplotlib, πρόκειται για μία ολοκληρωμένη βιβλιοθήκη, που σχετίζεται με τη δημιουργία στατιστικών, κινούμενων και διαδραστικών απεικονίσεων στην Python.

Επιπροσθέτως, η Matplotlib, παράγει στοιχεία που σχετίζονται με την ποιότητα δημοσίευσης σε μία ποικιλία μορφών έντυπης εκτύπωσης και διαδραστικών περιβαλλόντων όλων των πλατφορμών.

Προκειμένου κάποιος να εγκαταστήσει τη συγκεκριμένη βιβλιοθήκη, αρκεί να ανοίξει το command line των windows **(Κουμπί Windows + R → cmd)** και να πληκτρολογήσει την εξής εντολή:

pip install matplotlib

Επίσης, η παραπάνω απεικόνιση επιτεύχθηκε με δεύτερο τρόπο, με τη χρήση της βιβλιοθήκης **Seaborn**. Αναλυτικότερα, η Seaborn είναι μια βιβλιοθήκη της Python, ώστε να οπτικοποιεί δεδομένα βασιζόμενη στη Matplotlib. Παρέχει μια διεπαφή υψηλού επιπέδου για τη σχεδίαση ελκυστικών και ενημερωτικών στατιστικών γραφικών.

Προκειμένου κάποιος να εγκαταστήσει τη συγκεκριμένη βιβλιοθήκη, αρκεί να ανοίξει το command line των windows **(Κουμπί Windows + R → cmd)** και να πληκτρολογήσει την εξής εντολή:

pip install seaborn

Αξίζει να σημειωθεί πως είναι η πιο πρόσφατη σταθερή έκδοση. Είναι επίσης δυνατό να συμπεριληφθούν προαιρετικές στατιστικές εξαρτήσεις:

pip install seaborn[stats]

Επιπλέον, για τον κώδικα ***window.py***, χρησιμοποιήθηκε η βιβλιοθήκη **Tkinter,** προκειμένου κατά την εκτέλεση του, να εμφανίζεται στο χρήστη ένα παράθυρο, όπου κάθε φορά που εκτελείται ο κώδικας, θα ερωτάται, ποιο γράφημα επιθυμεί να αναπαρίσταται, από τα 120 διαθέσιμα txt αρχεία που δόθηκαν. Το πακέτο tkinter συνιστά την τυπική διεπαφή της Python 3 για την εργαλειοθήκη Tcl/Tk GUI (Graphical User Interface), και σα βιβλιοθήκη είναι πλήρως παραμετροποιήσιμη, επεκτάσιμη και εύχρηστη.

Τέλος, για την υλοποίηση του κώδικα window.py, πραγματοποιήθηκε μετατροπή των 120 αρχείων txt, σε αρχεία dat, και κατόπιν δημιουργήθηκε ένας public φάκελος στο GitHub που περιέχει αυτά τα 120 dat αρχεία, ώστε να διαβάζονται online τα δεδομένα.

Προκειμένου κάποιος να εγκαταστήσει τη συγκεκριμένη βιβλιοθήκη, αρκεί να ανοίξει το command line των windows **(Κουμπί Windows + R → cmd)** και να πληκτρολογήσει την εξής εντολή:

pip install tkintertable

3.5. Ο αλγόριθμος NEH έχει πολυπλοκότητα $O(n^3m)$, αλλά με τις βελτιώσεις που προτείνονται στο [2] η πολυπλοκότητα του μπορεί να μειωθεί σε $O(n^2m)$. Υλοποιήστε το βελτιωμένο αλγόριθμο και πραγματοποιήστε ανάλυση χρόνων εκτέλεσης που να δείχνει την υπεροχή του βελτιωμένου NEH αλγορίθμου.

Τα βήματα που ακολουθεί ο αλγόριθμος NEH, είναι τα κάτωθι:

i) **1ο Βήμα:** Ταξινομεί τις n θέσεις εργασίας κατά φθίνουσα σειρά αθροισμάτων, των χρόνων επεξεργασίας, στις μηχανές.

ii) **2ο Βήμα:** Παίρνει τις δύο πρώτες εργασίες και τις προγραμματίζει, προκειμένου να ελαχιστοποιηθεί το μερικό makespan, εφόσον υπάρχουν μόνο δύο από αυτές τις εργασίες.

iii) **3ο Βήμα:** Για k=3, το n:

iv) **4ο Βήμα:** Εισάγει την k-οστή εργασία στη θέση, μεταξύ των k πιθανών εργασιών, η οποία ελαχιστοποιεί το μερικό makespan.

Η πολυπλοκότητα του βήματος (i) είναι **O(n log(n))**, από εκείνη του βήματος (ii) που είναι **O(m)**. Τέλος, το βήμα (iv) του αλγορίθμου NEH έχει πολυπλοκότητα **O(km)**. Οπότε συμπεραίνεται ότι οι αλγόριθμοι NEH εκτελούνται σε χρόνο **$O(n^2m)$**.

Τα παραπάνω βήματα του βελτιωμένου αλγορίθμου NEH, έχουν υλοποιηθεί στον κώδικα ***On2m.py***. Ο συγκεκριμένος κώδικας υλοποιεί το δεδομένο ευρετικό αλγόριθμο για τη δημιουργία βέλτιστης λύσης του προβλήματος Permutation Flow-Shop Scheduling Problem (PFSP). Στην ουσία απαρτίζει τη βελτιωμένη εκδοχή του αλγορίθμου NEH.

Τα δεδομένα των αρχείων βρίσκονται σε μορφή .dat, και αντλούνται online από έναν public φάκελο στο GitHub, που βρίσκεται στον κάτωθι σύνδεσμο:

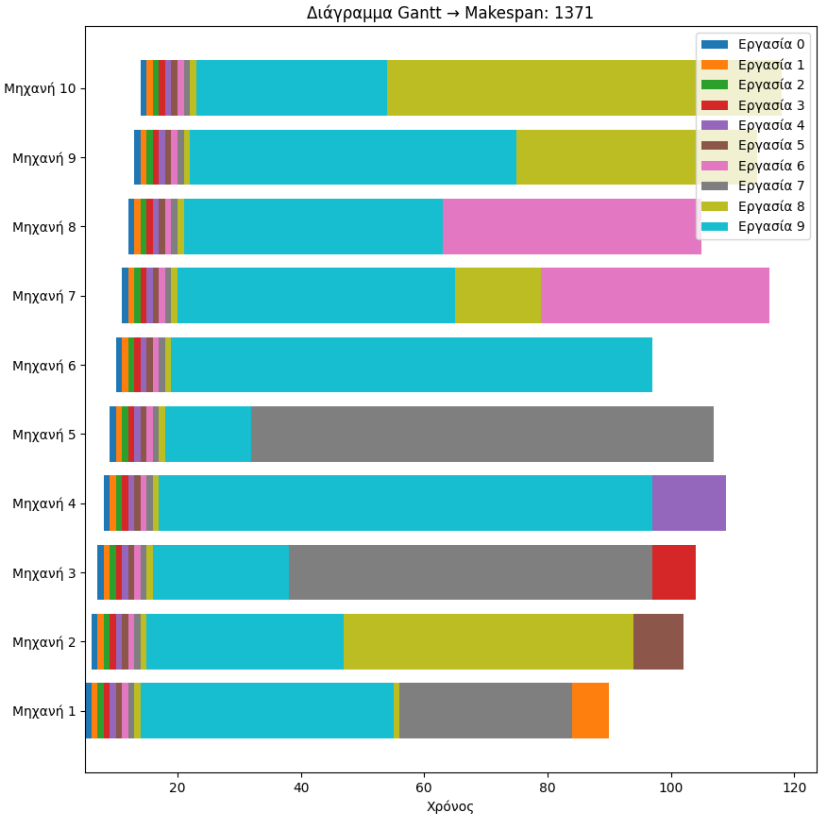
https://github.com/ppatsea/Algorithms

Επιπροσθέτως, κατά την εκτέλεση του κώδικα, εμφανίζονται στο Command Window *, τα εξής:

i) Βέλτιστη Διάταξη Εργασιών του αλγορίθμου NEH,
ii) Βέλτιστη Σειρά Εργασιών του αλγορίθμου NEH,
iii) Χρόνος Εκτέλεσης του αλγορίθμου NEH,
iv) Συνολικός Χρόνος Εκτέλεσης του κάθε αρχείου dat.

Μετά το πέρας εκτέλεσης των αρχείων, εμφανίζεται το τελικό Διάγραμμα Gantt των συγκεκριμένων αρχείων, και τέλος όταν ο χρήστης κλείσει το παράθυρο του γραφήματος, στο Command Window εμφανίζεται ο Συνολικός Χρόνος Εκτέλεσης όλου του Προγράμματος.

* : Ο χρήστης από τις κατάλληλες γραμμές κώδικα επιλέγει αν επιθυμεί τα αποτελέσματα να εμφανίζονται αποκλειστικά και μόνο στην οθόνη, ή εκτός από αυτό, να εγγράφονται αυτόματα και όχι χειροκίνητα, σε ένα αντίστοιχο txt αρχείο με την ονομασία **results_On2m.txt**.



Εικόνα 3: Διάγραμμα Gantt, αλγορίθμου $O(n^2m)$, για την $20^η$ εργασία

```
File: ./Taillard-PFSP/ta004.txt
Βέλτιστη Διάταξη Εργασιών O(n^2 * m): [1, 3, 0, 2, 7, 8, 9, 4, 6, 5]
Χρόνος Εκτέλεσης O(n^2 * m): 1114
Συνολικός Χρόνος Εκτέλεσης Αρχείου: 0.00020989999757148325 seconds

File: ./Taillard-PFSP/ta005.txt
Βέλτιστη Διάταξη Εργασιών O(n^2 * m): [5, 2, 9, 3, 0, 1, 4, 8, 7, 6]
Χρόνος Εκτέλεσης O(n^2 * m): 856
Συνολικός Χρόνος Εκτέλεσης Αρχείου: 0.00021570001263171434 seconds

File: ./Taillard-PFSP/ta006.txt
Βέλτιστη Διάταξη Εργασιών O(n^2 * m): [1, 9, 2, 4, 3, 7, 6, 8, 0, 5]
Χρόνος Εκτέλεσης O(n^2 * m): 1169
Συνολικός Χρόνος Εκτέλεσης Αρχείου: 0.00021189998369663954 seconds

File: ./Taillard-PFSP/ta007.txt
Βέλτιστη Διάταξη Εργασιών O(n^2 * m): [4, 9, 3, 6, 7, 0, 2, 1, 8, 5]
Χρόνος Εκτέλεσης O(n^2 * m): 986
Συνολικός Χρόνος Εκτέλεσης Αρχείου: 0.00021199998445808887 seconds

File: ./Taillard-PFSP/ta008.txt
Βέλτιστη Διάταξη Εργασιών O(n^2 * m): [0, 5, 6, 1, 4, 3, 9, 7, 2, 8]
Χρόνος Εκτέλεσης O(n^2 * m): 1008
Συνολικός Χρόνος Εκτέλεσης Αρχείου: 0.00021070000366307795 seconds

File: ./Taillard-PFSP/ta009.txt
Βέλτιστη Διάταξη Εργασιών O(n^2 * m): [9, 2, 4, 7, 5, 3, 6, 1, 8, 0]
Χρόνος Εκτέλεσης O(n^2 * m): 1105
Συνολικός Χρόνος Εκτέλεσης Αρχείου: 0.00023880001390352845 seconds
```

Εικόνα 4: Command Window, αλγορίθμου $O(n^2m)$, για τις εργασίες 1-20

3.6. Υλοποιήστε μια δική σας προσέγγιση για το πρόβλημα. Συγκρίνετε τα αποτελέσματά της με τα αποτελέσματα που παράγει ο ΝΕΗ.

Όπως αναφέρθηκε και παραπάνω, το Permutation Flow-Shop Scheduling Problem (PFSP), αφορά τον προγραμματισμό χρόνου εκτέλεσης ενός συνόλου εργασιών σε ένα σύνολο μηχανών ούτως ώστε να επιτευχθεί ο συντομότερος χρόνος ολοκλήρωσης για την εργασία που ολοκληρώνει την επεξεργασία της τελευταία. Το συγκεκριμένο πρόβλημα μπορεί να οριστεί με αλγορίθμους που σχετίζονται με την Εύρεση Βέλτιστης Λύσης. Ενδεικτικά, τέτοιοι αλγόριθμοι είναι οι:

i) Γενετικοί Αλγόριθμοι
ii) Metaheuristic
iii) Tabu Search
iv) Αλγόριθμος Τυχαίας Αναζήτησης

Για την εξαγωγή του συγκεκριμένου ερωτήματος υλοποιήθηκε ο κώδικας του Tabu Search (***tabu.py***). Η Αναζήτηση Tabu (TS), αφορά έναν επαναληπτικό αλγόριθμο αναζήτησης γειτονιάς, όπου αυτή η γειτονιά αλλάζει δυναμικά. Η TS ενισχύει την τοπική αναζήτηση, αποφεύγοντας έτσι σημεία στο χώρο αναζήτησης, που έχουν ήδη επισκεφθεί. Κατά αυτόν τον τρόπο, αποφεύγοντας τα ήδη επισκέψιμα σημεία, αποφεύγονται και οι βρόχοι στο χώρο αναζήτησης.

Εν κατακλείδι, το κύριο χαρακτηριστικό της Αναζήτησης Tabu (Tabu Search – TS), είναι η χρήση ρητής μνήμης, με δύο στόχους:

i) Αποτροπή επανεξέτασης της αναζήτησης σε λύσεις που έχουν ήδη επισκεφθεί στο παρελθόν
ii) Εξερεύνηση των μη επισκέψιμων περιοχών του χώρου των λύσεων

Τα βήματα του εν λόγω αλγορίθμου, είναι τα εξής:

i) **1ο Βήμα:** Επιλέγεται μία αρχική (τρέχουσα) λύση.

ii) **2ο Βήμα:** Η τρέχουσα λύση καθορίζεται από μία γειτονιά τοπικών λύσεων.

iii) **3ο Βήμα:** Επιλέγεται η βέλτιστη λύση, από τις τοπικές λύσεις (νέα τρέχουσα λύση).

iv) **4ο Βήμα:** Τοποθέτηση χαρακτηριστικών βέλτιστης λύσης στον πίνακα μνήμης.

v) **5ο Βήμα:** Επανάληψη της συγκεκριμένης διαδικασίας από τη νέα τρέχουσα λύση, έως ότου ικανοποιηθεί το κριτήριο τερματισμού της διαδικασίας.

Τα παραπάνω βήματα του αλγορίθμου TS (Tabu Search), έχουν υλοποιηθεί στον κώδικα ***tabu.py***.

Τα δεδομένα των αρχείων βρίσκονται σε μορφή .dat, και αντλούνται online από έναν public φάκελο στο GitHub, που βρίσκεται στον κάτωθι σύνδεσμο:

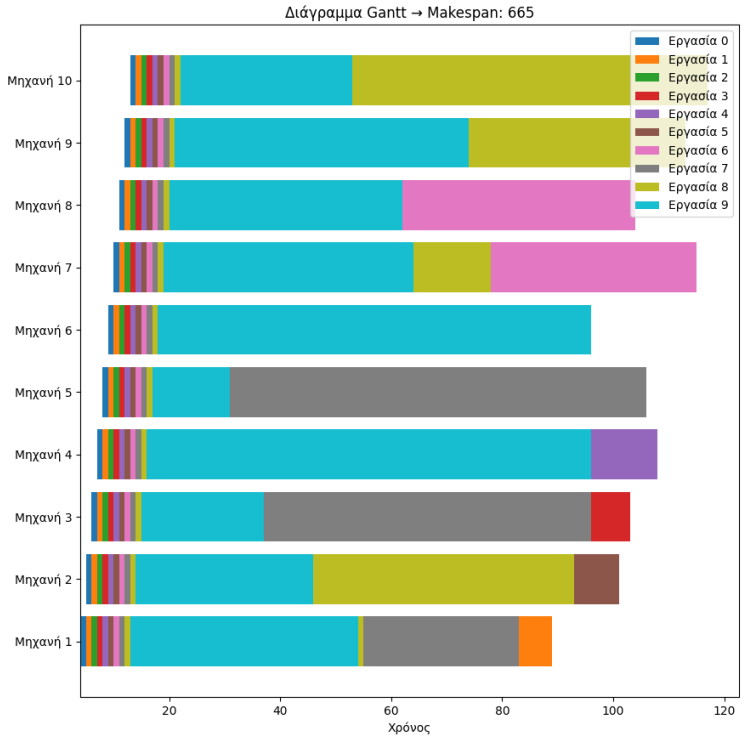
https://github.com/ppatsea/Algorithms

Επιπροσθέτως, κατά την εκτέλεση του κώδικα, εμφανίζονται στο Command Window *, τα εξής:

i) Βέλτιστη Διάταξη Εργασιών του αλγορίθμου NEH,
ii) Βέλτιστη Σειρά Εργασιών του αλγορίθμου NEH,
iii) Χρόνος Εκτέλεσης του αλγορίθμου NEH,
iv) Συνολικός Χρόνος Εκτέλεσης του κάθε αρχείου dat.

Μετά το πέρας εκτέλεσης των αρχείων, εμφανίζεται το τελικό Διάγραμμα Gantt των συγκεκριμένων αρχείων, και τέλος όταν ο χρήστης κλείσει το παράθυρο του γραφήματος, στο Command Window εμφανίζεται ο Συνολικός Χρόνος Εκτέλεσης όλου του Προγράμματος.

* : Ο χρήστης από τις κατάλληλες γραμμές κώδικα επιλέγει αν επιθυμεί τα αποτελέσματα να εμφανίζονται αποκλειστικά και μόνο στην οθόνη, ή εκτός από αυτό, να εγγράφονται αυτόματα και όχι χειροκίνητα, σε ένα αντίστοιχο txt αρχείο με την ονομασία **results_tabu_search.txt**.



Εικόνα 5: Διάγραμμα Gantt, αλγορίθμου Tabu Search (TS), για την 20η εργασία

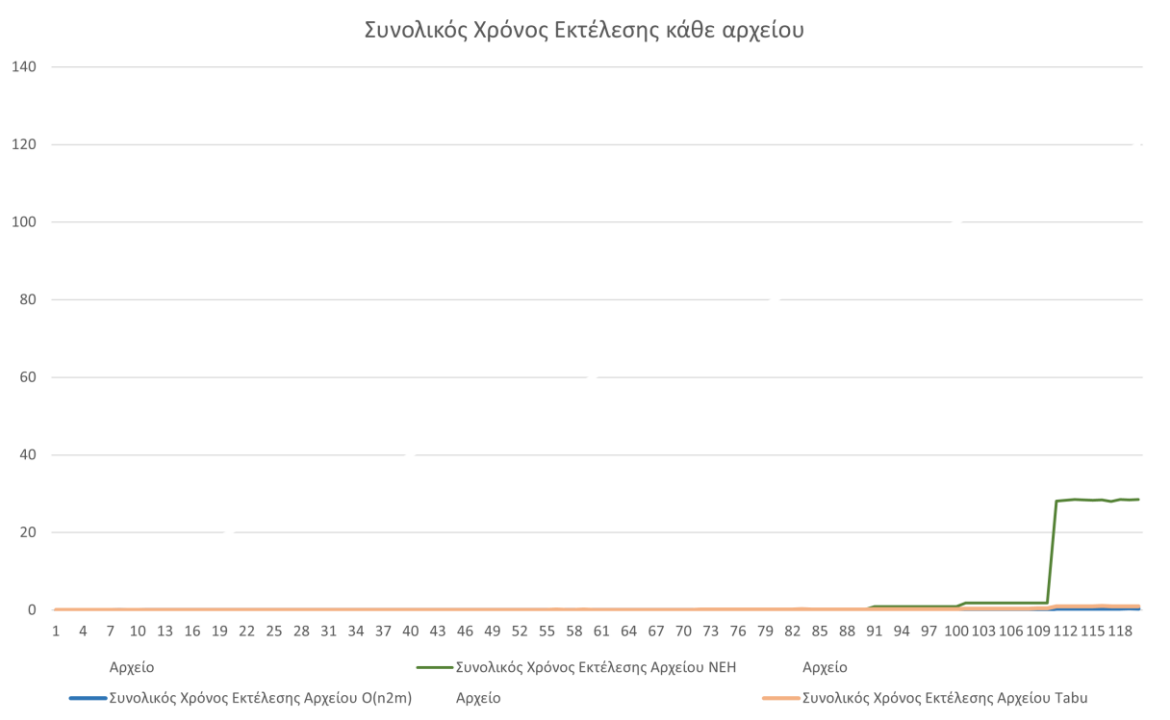
```
File: ./Taillard-PFSP/ta015.txt
Βέλτιστη Διάταξη Εργασιών Tabu Search: [1, 5, 4, 6, 2, 0, 7, 3, 8, 9]
Χρόνος Εκτέλεσης Tabu Search: 616
Συνολικός Χρόνος Εκτέλεσης Αρχείου: 0.019950151443481445 seconds


File: ./Taillard-PFSP/ta016.txt
Βέλτιστη Διάταξη Εργασιών Tabu Search: [1, 5, 3, 9, 4, 2, 7, 0, 8, 6]
Χρόνος Εκτέλεσης Tabu Search: 611
Συνολικός Χρόνος Εκτέλεσης Αρχείου: 0.020928382873535156 seconds


File: ./Taillard-PFSP/ta017.txt
Βέλτιστη Διάταξη Εργασιών Tabu Search: [2, 4, 9, 5, 8, 7, 0, 6, 3, 1]
Χρόνος Εκτέλεσης Tabu Search: 625
Συνολικός Χρόνος Εκτέλεσης Αρχείου: 0.020963668823242188 seconds


File: ./Taillard-PFSP/ta018.txt
Βέλτιστη Διάταξη Εργασιών Tabu Search: [1, 6, 7, 0, 5, 2, 9, 4, 3, 8]
Χρόνος Εκτέλεσης Tabu Search: 563
Συνολικός Χρόνος Εκτέλεσης Αρχείου: 0.02194046974182129 seconds


File: ./Taillard-PFSP/ta019.txt
Βέλτιστη Διάταξη Εργασιών Tabu Search: [9, 8, 5, 3, 6, 2, 7, 4, 1, 0]
Χρόνος Εκτέλεσης Tabu Search: 666
Συνολικός Χρόνος Εκτέλεσης Αρχείου: 0.02094745635986328 seconds


File: ./Taillard-PFSP/ta020.txt
Βέλτιστη Διάταξη Εργασιών Tabu Search: [7, 8, 2, 4, 9, 3, 0, 6, 1, 5]
Χρόνος Εκτέλεσης Tabu Search: 665
Συνολικός Χρόνος Εκτέλεσης Αρχείου: 0.020936012268066406 seconds
```

Εικόνα 6: Command Window, αλγορίθμου Tabu Search (TS), για τις εργασίες 1-20

4. Συμπεράσματα

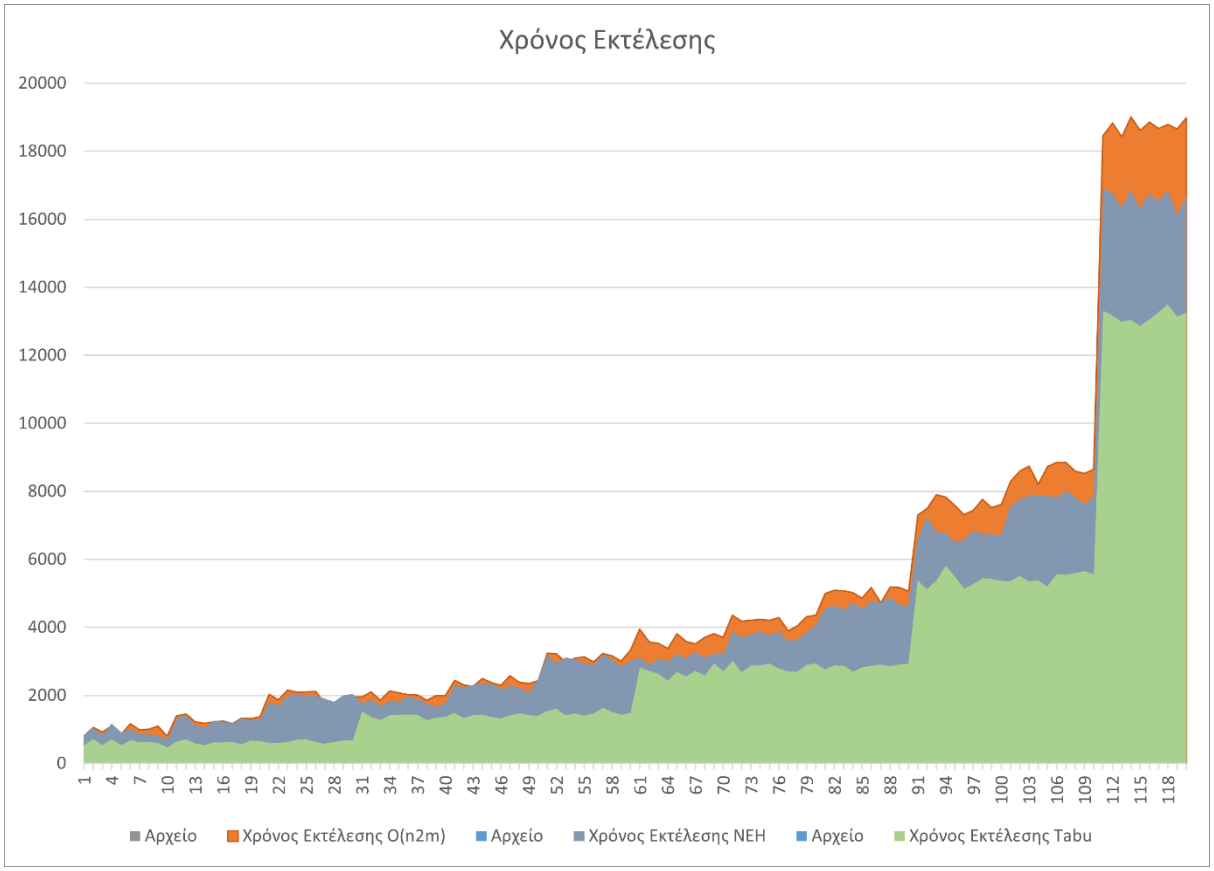
ΣΥΓΚΡΙΣΗ ΧΡΟΝΟΥ ΕΚΤΕΛΕΣΗΣ & MAKESPAN:

Από την εκτέλεση και τη σύγκριση των τριών αλγορίθμων, NEH, O(n^2m) και Tabu Search, αντίστοιχα, προκύπτουν:



Εικόνα 7: Σύγκριση των τριών Αλγορίθμων, ως προς το χρόνο εκτέλεσης του κάθε αρχείου (και για τις 120 εργασίες)

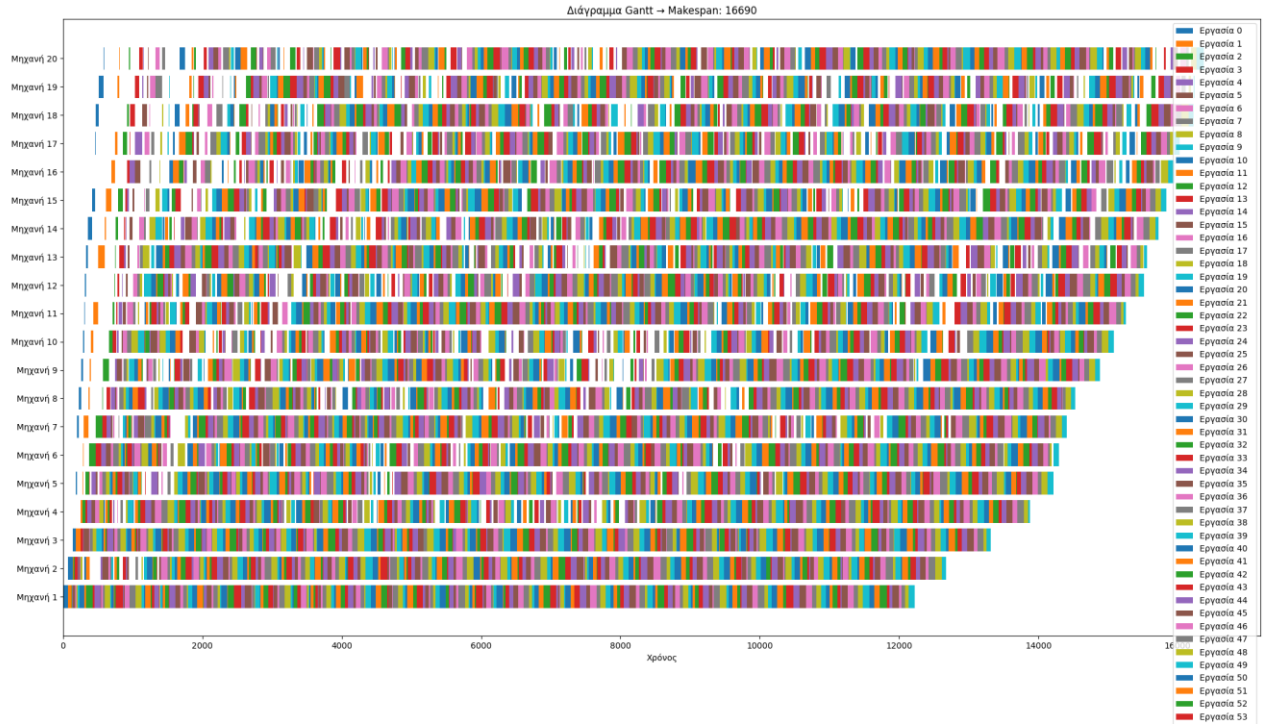
Από το παραπάνω γράφημα διαπιστώνεται, πως προκειμένου να εκτελεστεί κάθε αρχείο, του κάθε κώδικα, αντλείται κάποιος χρόνος. Το αποτέλεσμα είναι ο Αλγόριθμος NEH, να χρειάζεται πολύ περισσότερο χρόνο εκτέλεσης, συγκριτικά με αυτόν των άλλων δύο αλγορίθμων.



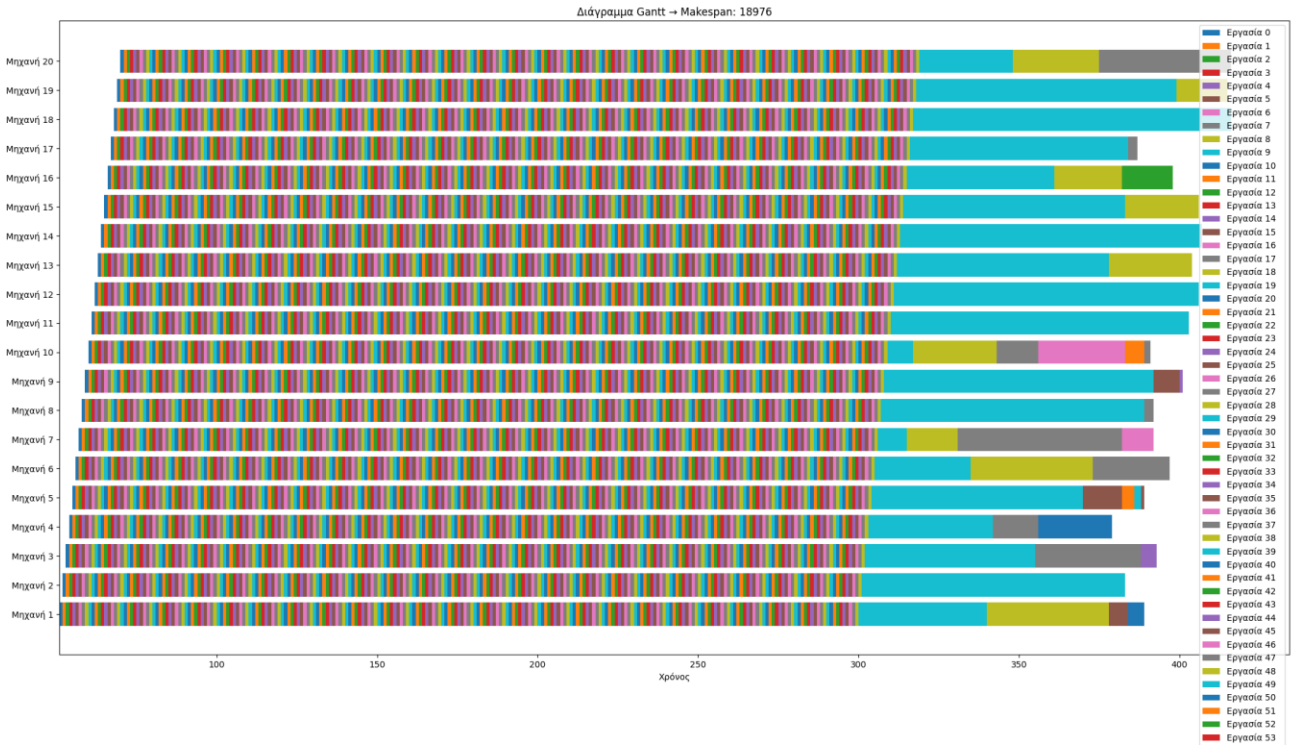
Εικόνα 8: Σύγκριση των τριών Αλγορίθμων, ως προς το χρόνο εκτέλεσης - makespan (και για τις 120 εργασίες)

Από το παραπάνω γράφημα διαπιστώνεται, πως προκειμένου να εκτελεστεί ο κάθε κώδικας, είναι απαραίτητο κάποιο makespan. Το αποτέλεσμα είναι ο Αλγόριθμος $O(n^2m)$, να χρησιμοποιεί πολύ μεγαλύτερο makespan σε σύγκριση με τον Tabu Search (TS), που χρησιμοποιεί το λιγότερο.

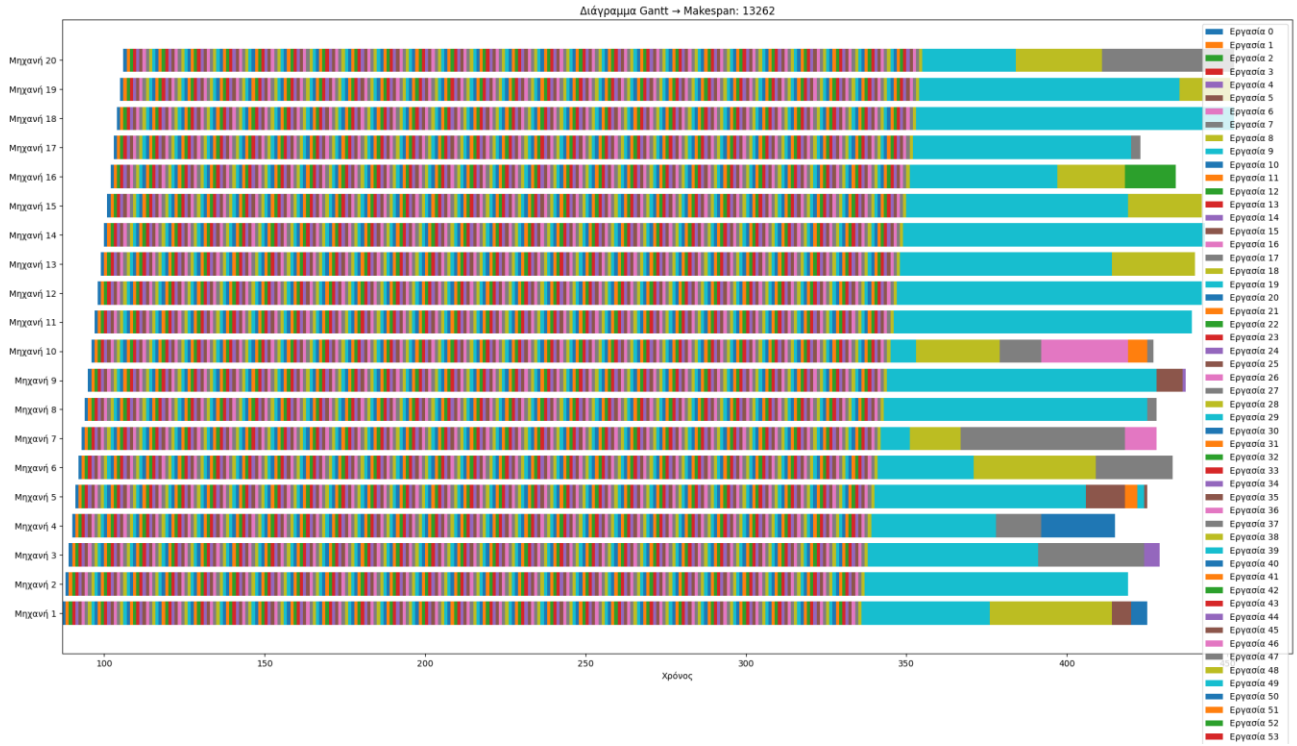
ΔΙΑΓΡΑΜΜΑΤΑ GANTT:



Εικόνα 10:Διάγραμμα Gantt, αλγορίθμου ΝΕΗ, για την 120η εργασία



Εικόνα 9:Διάγραμμα Gantt, αλγορίθμου $O(n^2m)$, για την 120η εργασία



Εικόνα 11:Διάγραμμα Gantt, αλγορίθμου Tabu Search (TS), για την 120η εργασία

Συνοψίζοντας, κατόπιν της εκτέλεσης των αλγορίθμων για το πρόβλημα του Flow-Shop Scheduling, παρατηρείται ότι ο NEH εμφανίζει το χρόνο εκτέλεσης που απαιτεί περισσότερο χρόνο συγκριτικά με τους Tabu Search και $O(n^2m)$. Ωστόσο, όσον αφορά το συνολικό χρόνο ολοκλήρωσης (makespan), ο Tabu Search παράγει τις βέλτιστες λύσεις, ακολουθούμενος από τον NEH και, τέλος, από τον αλγόριθμο $O(n^2m)$.

Ειδικότερα, ο NEH είναι ο πιο χρονοβόρος αλγόριθμος, αλλά οι λύσεις που παράγει δεν είναι πάντα οι βέλτιστες. Αντιθέτως, ο Tabu Search προσφέρει καλές λύσεις σε σύντομο χρονικό διάστημα, ενώ ο αλγόριθμος $O(n^2m)$, κινείται σε μια μεσαία γραμμή, προσφέροντας λύσεις με ευνοϊκή σχέση μεταξύ του χρόνου εκτέλεσης και της ποιότητας της λύσης.

Συμπερασματικά, οι Tabu Search και $O(n^2m)$, προσφέρουν ισορροπημένες επιδόσεις μεταξύ του χρόνου εκτέλεσης και της ποιότητας των λύσεων, με τον πρώτο να ξεχωρίζει στην ποιότητα των makespan. Τέλος, είναι αξιοσημείωτο να αναφερθεί ότι, ανάλογα με τις ανάγκες του εκάστοτε προβλήματος, η επιλογή του αλγορίθμου μπορεί να γίνει βάσει του συνολικού στόχου, τη βέλτιστη λύση ή την επίτευξη ισορροπημένων αποτελεσμάτων δηλαδή, σε λογικό χρονικό διάστημα.