ΠΑΝΕΠΙΣΤΗΜΙΟ
ΙΩΑΝΝΙΝΩΝ

Πανεπιστήμιο Ιωαννίνων
Τμήμα Πληροφορικής και Τηλεπικοινωνιών
Πτυχιακή Εργασία

# Χρονοπρογραμματισμός εργασιών σε μια μηχανή με περιορισμούς χωρητικότητας εξαρτώμενους από το χρόνο

Αναστάσιος Βήττας

**Επιβλέπων καθηγητής:** Χρήστος Γκόγκος
Αναπληρωτής καθηγητής Πανεπιστημίου Ιωαννίνων

11 Δεκεμβρίου 2022

**Εγκρίθηκε από τριμελή εξεταστική επιτροπή**
Άρτα, 8 Σεπτεμβρίου 2022

# ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1. **Επιβλέπων Καθηγητής**
   Χρήστος Γκόγκος
   Αναπληρωτής Καθηγητής

2. **Μέλος Επιτροπής**
   Νικόλαος Γιαννακέας
   Επίκουρος Καθηγητής

3. **Μέλος Επιτροπής**
   Ευριπίδης Γλαβάς
   Καθηγητής

## Δήλωση μη λογοκλοπής

Δηλώνω υπεύθυνα και γνωρίζοντας τις κυρώσεις του Ν. 2121/1993 περί Πνευματικής Ιδιοκτησίας, ότι η παρούσα πτυχιακή εργασία είναι εξ ολοκλήρου αποτέλεσμα δικής μου ερευνητικής εργασίας, δεν αποτελεί προϊόν αντιγραφής ούτε προέρχεται από ανάθεση σε τρίτους. Όλες οι πηγές που χρησιμοποιήθηκαν (κάθε είδους, μορφής και προέλευσης) για τη συγγραφή της περιλαμβάνονται στη βιβλιογραφία.

Βήττας Αναστάσιος

Υπογραφή

# Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέπων καθηγητή μου Χρήστο Γκόγκο για την πολύτιμη βοήθεια και καθοδήγηση στην συγγραφή της πτυχιακής μου εργασίας, καθώς και τον Άγγελο Δήμητσα για την βοήθεια του καθ΄ όλη αυτή τη διάρκεια.

## Περίληψη

Τα προβλήματα Χρονοπρογραμματισμού έχουν ως στόχο τους να υλοποιήσουν τον προγραμματισμό ενός συνόλου εργασιών με την χρήση διαδικασιών που θα βελτιστοποιήσουν το τελικό αποτέλεσμα, αυξάνοντας την απόδοση της λύσης του προβλήματος. Επίσης βασικός στόχος των προβλημάτων αυτών αποτελεί και η ικανοποίηση των περιορισμών που έχουν τεθεί στο συγκεκριμένο πρόβλημα.
Στην παρούσα εργασία διαπραγματευόμαστε ένα πρόβλημα Χρονοπρογραμματισμού, υπολογιστικής πολυπλοκότητας NP-Hard. Στην διάθεση μας έχουμε μια απλή ή μοναδική μηχανή (Single ή One Machine Shop) στην οποία θα πραγματοποιηθεί ακολουθιακά η επεξεργασία των εργασιών. Παράλληλα διαθέτουμε περιορισμούς που αφορούν τόσο τις εργασίες όσο και την χωρητικότητα αυτής της μηχανής και πρέπει να διαμορφώσουμε το μοντέλο μας σύμφωνα με αυτούς. Πιο συγκεκριμένα, κάποιοι από τους περιορισμούς που πρέπει να ακολουθήσουμε κατά την επίλυση του προβλήματος είναι το να μην ξεπεραστεί αυστηρά η χωρητικότητα της μηχανής, η οποία είναι χρονικά μεταβαλλόμενη, και η κάθε εργασία να τοποθετείται έως την καθορισμένη ημερομηνία λήξης ή νωρίτερα έτσι ώστε να μην δημιουργείται καθυστέρηση. Όσο αυξάνεται η συνάρτηση της καθυστέρησης τόσο χειρότερη γίνεται η λύση.
Τέλος, παρουσιάζονται τα αποτελέσματα για τα προβλήματα των συνόλων δεδομένων, τα οποία προέκυψαν έπειτα από την εφαρμογή του επιλυτή IBM ILOG CP Optimizer σε σύγκριση με πρόσθετα αποτελέσματα που έχουν βασιστεί στα ίδια σύνολα δεδομένων αλλά με την χρήση άλλων τεχνικών και αλγορίθμων.



**Λέξεις Κλειδιά:** Χρονοπρογραμματισμός, Μοντέλο Απλής Μηχανής, Χωρητικότητα Μηχανής, CPLEX, CP Optimizer

**Abstract**

Scheduling problems aim to implement the scheduling of a set of tasks using processes that will optimize the final result, increasing the efficiency of the problem's solution. The main objective of scheduling problems is to meet the constraints placed upon a specific problem.
In this paper we are approaching to solve a NP-Hard Scheduling problem. At our disposal we have a Single or One Machine in which the processing of tasks will be carried out sequentially. Simultaneously, we have restrictions that concern both the operation and the capacity of this machine, and we must configure our model according to them. More specifically, some of the limitations that we must follow when solving the problem are to strictly adhere to the capacity of the machine, which is time-varying, and to allocate to each task a specified completion date or earlier so as not to create a delay. The longer the delay function, the worse the solution becomes.
Finally, the results of the problems of the database, which arose after the application of the solver IBM ILOG CP Optimizer are presented compared to additional results based on the same datasets but using other techniques and algorithms.

**Keywords: Scheduling, One Machine Scheduling, Capacity, CPLEX, CP Optimizer**

# Περιεχόμενα

# Κατάλογος Σχημάτων

# Κατάλογος Πινάκων

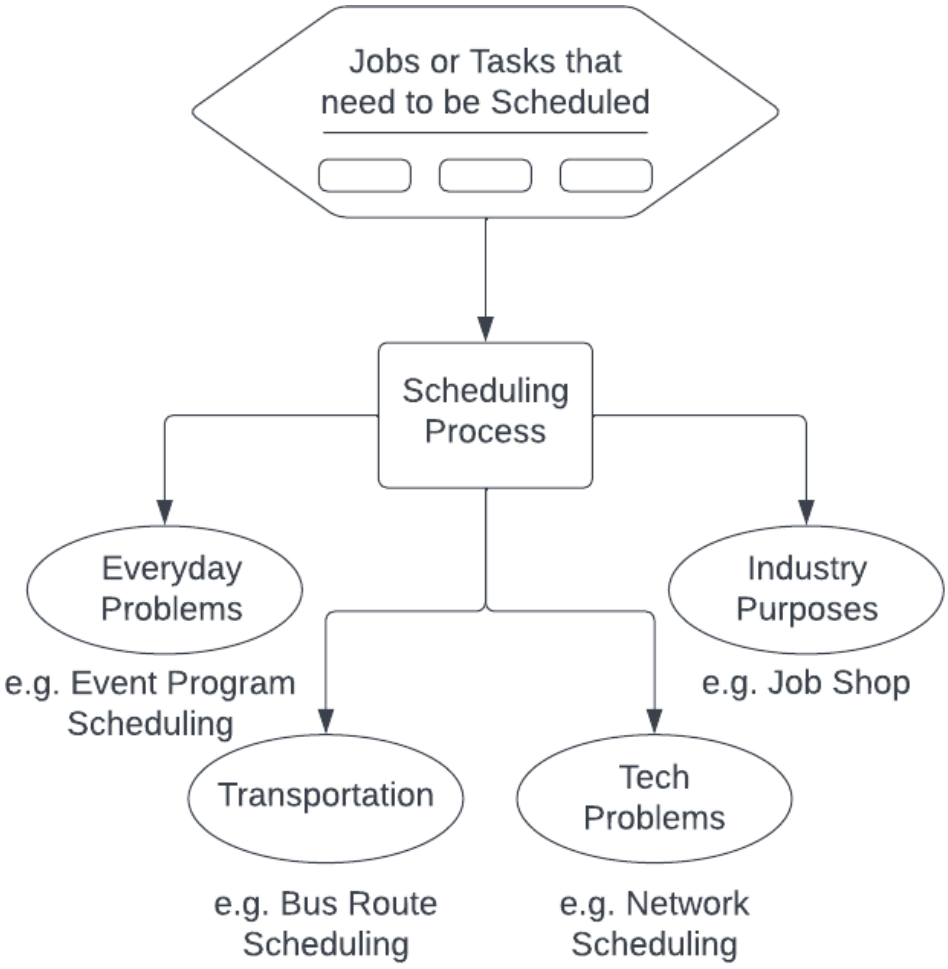
# List of Algorithms

# Κεφάλαιο 1

# Εισαγωγή

Τις τελευταίες δεκαετίες τα προβλήματα χρονοπρογραμματισμού έχουν κάνει αισθητή την παρουσία τους σε πολλούς τομείς της πληροφορικής, με βασικό τους στόχο την επίλυση προβλημάτων είτε της καθημερινότητας, είτε βιομηχανικού, είτε τεχνολογικού περιεχομένου με μεγάλη επιτυχία. Το γεγονός αυτό καθιστά την παρούσα ερευνητική περιοχή πολύ σημαντική σε μια κοινωνία με ανάγκη για υψηλή παραγωγικότητα, επιφέροντας μεγάλες οικονομικές επιπτώσεις, λόγω της αυξημένης αποδοτικότητας - ταχύτητας των διεργασιών μετά τον προγραμματισμό τους [3]. Κάποια παραδείγματα ανάγκης τεχνικών χρονοπρογραμματισμού αποτελούν, ο προγραμματισμός εκδηλώσεων και δραστηριοτήτων, δρομολόγιων Μέσων Μαζικής Μεταφοράς ή επαγγελματικών αυτοκινήτων, προγραμμάτων παρακολούθησης παρουσιάσεων, βιομηχανικών δραστηριοτήτων, ωραρίου υπαλλήλων.
Ο χρονοπρογραμματισμός καθίσταται ένα πολύ ενδιαφέρον αντικείμενο μελέτης καθώς συνδέει τρεις βασικές επιστήμες, την τεχνητή νοημοσύνη (artificial intelligence), την επιχειρησιακή έρευνα (operations research) αλλά και τα εφαρμοσμένα μαθηματικά (applied mathematics).
Επίσης τα προβλήματα χρονοπρογραμματισμού πολλές φορές παρουσιάζουν υψηλή υπολογιστική πολυπλοκότητα, όπως το πρόβλημα που θα επιλύσουμε με υπολογιστική πολυπλοκότητα NP-hard. Ένας από του παράγοντες που επηρεάζουν την πολυπλοκότητα των προβλημάτων σαν αυτό που θα διαπραγματευτούμε είναι οι περιορισμοί. Επειδή κατά βάση τα προβλήματα αυτά αφορούν χρονικό προγραμματισμό βιομηχανικών διεργασιών ή προβλημάτων καθημερινότητας όπως η σύνταξη ενός προγράμματος μεταφορών, διαμορφώνονται περιορισμοί σύμφωνα με τις απαιτήσεις του δημιουργού τους για να μην ξεπεραστούν κάποιες συνθήκες. Για παράδειγμα κάποιες από τις συνθήκες αυτές μπορεί να είναι, σε μια περίπτωση βιομηχανικής επεξεργασίας να μην ξεπεραστεί ο αριθμός των εργασιών που μπορεί να διαχειριστεί μια μηχανή παραγωγής ταυτόχρονα ή σε ένα πρόγραμμα προγραμματισμού δρομολογίων στον τομέα των μεταφορών να μην ξεπεραστεί ένας αριθμός οχημάτων που θα περνάει από μια συγκεκριμένη στάση. Έτσι κατά την επίλυση των προβλημάτων ο αλγόριθμος βελτιστοποίησης σχεδιάζεται και παραμετροποιείται έτσι ώστε να ικανοποιούνται οι περιορισμοί, συνεπώς όσο περισσότεροι οι περιορισμοί τόσο μεγαλύτερος ο βαθμός δυσκολίας του προβλήματος.
Κάποιες τεχνικές επίλυσης τέτοιων προβλημάτων είναι ο μαθηματικός προγραμματισμός (mathematical programming), ο προγραμματισμός με περιορισμούς (constraint programming), ενώ επίσης είναι γεγονός ότι και οι μεταευρετικοί αλγόριθμοι είναι πολύ αποτελεσματικοί στην επίλυση των προβλημάτων υψηλής υπολογιστικής πολυπλοκότητας σε μικρό χρονικό διάστημα.

Ένας ακόμη όρος που πρέπει να αναφερθεί, ο οποίος αποτελεί σημαντικό κομμάτι της συγκεκριμένης ερευνητικής περιοχής, είναι ο χρονοπρογραμματισμός απλής μηχανής. Σε αυτή την μέθοδο κάθε εργασία αποτελείται μόνο από μια λειτουργία η οποία πρέπει να εκτελεστεί στην μια και μοναδική μηχανή. Παρόμοιες κατηγορίες με το μοντέλο απλής μηχανής είναι ο χρονοπρογραμματισμός παράλληλων μηχανών και ο χρονοπρογραμματισμός συνεχούς ροής, στους οποίους δεν θα αναφερθούμε ιδιαίτερα καθώς το πρόβλημα μας εστιάζει στο πρώτο μοντέλο [7].




Σχήμα 1.1: Ο μηχανισμός του Χρονοπρογραμματισμού

# Κεφάλαιο 2

# Χρονοπρογραμματισμός

Ο βασικός στόχος του χρονοπρογραμματισμού είναι, όπως προαναφέραμε, η μείωση του χρόνου και του κόστους παραγωγής, βελτιστοποιώντας την αποδοτικότητα των διάφορων λειτουργιών της διαδικασίας παραγωγής ή ενός προγράμματος.


Σχήμα 2.1: Σχέση ποιότητας - κόστους - χρόνου

Για να πραγματοποιηθεί αυτή η βελτιστοποίηση όμως θα πρέπει να λάβουμε υπόψιν μας ότι κάποιες λειτουργίες των προβλημάτων αυτών παρουσιάζουν περιορισμούς που τα θεωρητικά μοντέλα δεν μπορούν να υπολογίσουν. Κατά αυτόν τον τρόπο καταλαβαίνουμε ότι, για να καλυφθούν αυτοί οι περιορισμοί, αυξάνεται στο σύνολο της η πολυπλοκότητα των προβλημάτων που καλούμαστε να αντιμετωπίσουμε, ώστε να βρούμε την βέλτιστη τους λύση [8].

## 2.1 Τα πεδία του Χρονοπρογραμματισμού

Τα προβλήματα Χρονοπρογραμματισμού διαφέρουν μεταξύ τους ανάλογα με την φύση των εργασιών και τα χαρακτηριστικά τους, τους περιορισμούς του προβλήματος, την συμπεριφορά της κάθε μηχανής αλλά και την αντικειμενική συνάρτηση τους. Έτσι, κυρίως για λόγους ευκολίας τα προβλήματα Χρονοπρογραμματισμού έχουν χωριστεί σε 3 πεδία σημειογραφικά το $\alpha$, $\beta$ και $\gamma$. Το πεδίο $\alpha$ διαπραγματεύεται το περιβάλλον που αφορά την μηχανή ή τις μηχανές που θα χρησιμοποιηθούν για την επίλυση του προβλήματος, το $\beta$ σχετίζεται με τα χαρακτηριστικά και τους περιορισμούς των διεργασιών του συστήματος, για παράδειγμα τα χρονικά όρια που θα επιτρέπει η μηχανή στις εργασίες να πραγματοποιηθούν. Τέλος, το πεδίο $\gamma$ αφορά την αντικειμενική συνάρτηση που θα χρησιμοποιήσουμε, όπως φαίνεται στο παρακάτω σχήμα.


Σχήμα 2.2: Αντικειμενική Συνάρτηση [1]

## 2.2 Βαθμος Δυσκολιας Προβλημάτων Χρονοπρογραμματισμού

Σε αυτό το κεφάλαιο θα ασχοληθούμε με την αποδοτικότητα των αλγορίθμων, δηλαδή αν σε ένα πρόβλημα ο χρόνος εκτέλεσης του είναι λογικός σε σχέση με τον όγκο δεδομένων και τις παραμέτρους που χειριζόμαστε. Ειδικότερα, ένας αλγόριθμος ονομάζεται αποδοτικός όταν η εύρεση της λύσης του προβλήματος στο οποίο εφαρμόζεται, πραγματοποιείται σε πολυωνυμικό χρόνο.
Για τον καλύτερο διαχωρισμό, ταξινομούμε τα προβλήματα σε κλάσεις πολυπλοκότητας ανάλογα με την δυσκολία τους. Οι δυο αυτές κλάσεις πολυπλοκότητας είναι η κλάση P στην οποία ανήκουν απλούστερα προβλήματα τα οποία επιλύονται σε πολυωνυμικό χρόνο και η κλάση NP όπου αφορά δυσκολότερα προβλήματα [9].

### 2.2.1 Η κλάση P

Η κλάση P περιέχει όλα τα προβλήματα που μπορούν να λυθούν σε πολυωνυμικό χρόνο και αποτελεί υποσύνολο της κλάσης NP. Αναλυτικότερα η κλάση P είναι άρρηκτα συνδεδεμένη με την έννοια της αποδοτικότητας στον τομέα των αλγορίθμων καθώς βασικός στόχος της είναι η επίλυση των προβλημάτων σε μικρό χρονικό διάστημα. Ο χειρότερος χρόνος στον οποίο μπορεί να εξαντληθεί ένα πρόβλημα το οποίο ανήκει στην κλάση είναι ο $O(n^k)$.

### 2.2.2 Η κλάση NP

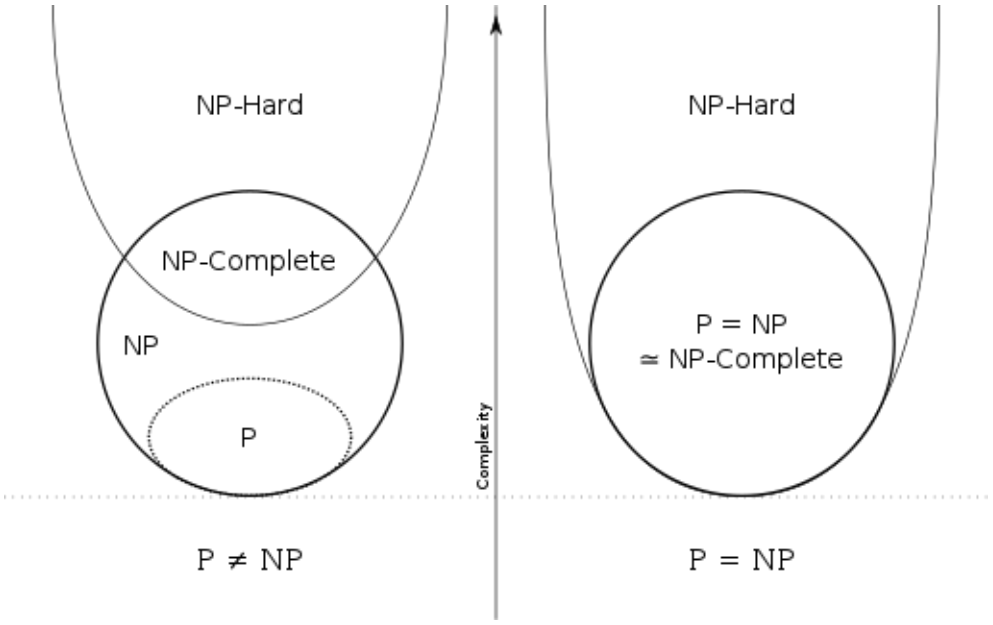
Η κλάση NP απευθύνεται σε δυσκολότερα προβλήματα τα οποία δεν μπορούν να επιλυθούν σε πολυωνυμικό χρόνο. Στην ουσία είναι τα προβλήματα τα οποία δέχονται διάφορες λύσεις και αναζητούν εντός πολυωνυμικού χρόνου την ορθότητα της κάθε λύσης. Προβλήματα τα οποία θα μπορούσαμε να πούμε ότι ανήκουν στην κλάση NP είναι το πρόβλημα του πλανόδιου πωλητή (TSP) ή το πρόβλημα του σακιδίου (Knapsack Problem).

### 2.2.3 NP-Hardness

Τα προβλήματα που χαρακτηρίζονται ως NP-hard αποτελούν προβλήματα ίδιου βαθμού δυσκολίας με τα δυσκολότερα προβλήματα της κλάσης NP [10]. Πιο συγκεκριμένα, τα προβλήματα που ανήκουν στην κλάση πολυπλοκότητας NP-hardness έχουν μεγαλύτερο βαθμό δυσκολίας από αυτά που μπορούν να επιλυθούν σε μια μη ντετερμινιστική μηχανή Turing [11], σε πολυωνυμικό χρόνο. Το πρόβλημα χρονοπρογραμματισμού εργασιών απλής μηχανής που θα υλοποιήσουμε χαρακτηρίζεται ως ένα πρόβλημα δυσκολίας NP-Hard.

### 2.2.4 NP-Completeness

Είναι γεγονός ότι ακόμη και σήμερα δεν γνωρίζουμε κανέναν αλγόριθμο επίλυσης προβλημάτων σε πολυωνυμικό χρόνο για προβλήματα NP-complete. Κατ΄ αυτόν τον τρόπο μπορούμε να προσδιορίσουμε την κλάση NP-complete ως μια κλάση που αντιπροσωπεύει




Σχήμα 2.3: Διάγραμμα τομής των κλάσεων P, NP, NP-hard, NP-complete

τα προβλήματα που η κατάσταση τους παραμένει άγνωστη [12].
Όπως παρατηρείται και στο σχήμα 2.2 η κλάση NP-complete αποτελεί την τομή των κλάσεων NP και NP-hard, και είναι υποσύνολο της κλάσης NP, άρα και υπό μια έννοια μεγαλύτερης δυσκολίας. Παραδείγματα προβλημάτων που ανήκουν στην κλάση NP-complete είναι το SAT (το πρόβλημα της ικανοποιησιμότητας) και το TSP (το πρόβλημα του περιοδεύοντος πωλητή) [13][14].

## 2.3 Κατηγοριες Προβλημάτων Χρονοπρογραμματισμού

Ο χρονοπρογραμματισμός χωρίζεται σε τέσσερις κατηγορίες, είναι είτε δυναμικός (Dynamic Scheduling), ή στατικός (Static Scheduling), ή ντετερμινιστικός (Deterministic Scheduling) ή στοχαστικός (Stochastic Scheduling). Συνήθως ο διαχωρισμός αυτός πραγματοποιείται ανάλογα με κάποια χαρακτηριστικά των κατηγοριών. Δηλαδή ο δυναμικός και ο στατικός χρονοπρογραμματισμός κατηγοριοποιούνται ανάλογα με τον χρόνο άφιξης της κάθε εργασίας, ενώ ο ντετερμινιστικός και ο στοχαστικός έχουν περισσότερες παραμέτρους.
Αναλυτικότερα, στον δυναμικό χρονοπρογραμματισμό η άφιξη των εργασιών πραγματοποιείται σε άγνωστο – τυχαίο χρονικό διάστημα ενώ στον στατικό χρονοπρογραμματισμό όλες οι εργασίες βρίσκονται σε διαθεσιμότητα από την αρχή.
Στον ντετερμινιστικό προγραμματισμό η μηχανή είναι κατάλληλα διαμορφωμένη έτσι ώστε οι εργασίες να εκτελούνται στον καλύτερο δυνατό χρόνο, χωρίς όμως να υπολογίζει άλλους εξωτερικούς παράγοντες. Επίσης, οι πληροφορίες της κάθε εργασίας, όπως ο χρόνος άφιξης τους και το άθροισμα τους, είναι γνωστές από την αρχή.
Αντίθετα, ο στοχαστικός χρονοπρογραμματισμός είναι αρκετά πιο ευέλικτος αφού το σύστημα είναι σε θέση να αντιμετωπίσει εξωτερικούς παράγοντες, όπως βλάβες ή ανθρώπινα λάθη. Οι πληροφορίες της κάθε εργασίας γίνονται γνωστές σε μεταβλητό χρόνο.
Για την ταξινόμηση των εργασιών στο σύστημα που κατηγοριοποιήσαμε παραπάνω χρεια-

ζόμαστε κάποιες μηχανές που θα πραγματοποιήσουν αυτόν τον χρονοπρογραμματισμό. Για την υλοποίηση ενός τέτοιου προβλήματος μπορούμε να χρησιμοποιήσουμε είτε το μοντέλο απλής μηχανής (single/one machine shop) ή το μοντέλο παράλληλων μηχανών (parallel machine shop) ή το μοντέλο συνεχούς ροής (flow shop) ή το κατά παραγγελία μοντέλο (job shop) [15].

### 2.3.1 Μοντέλο Απλής Μηχανής

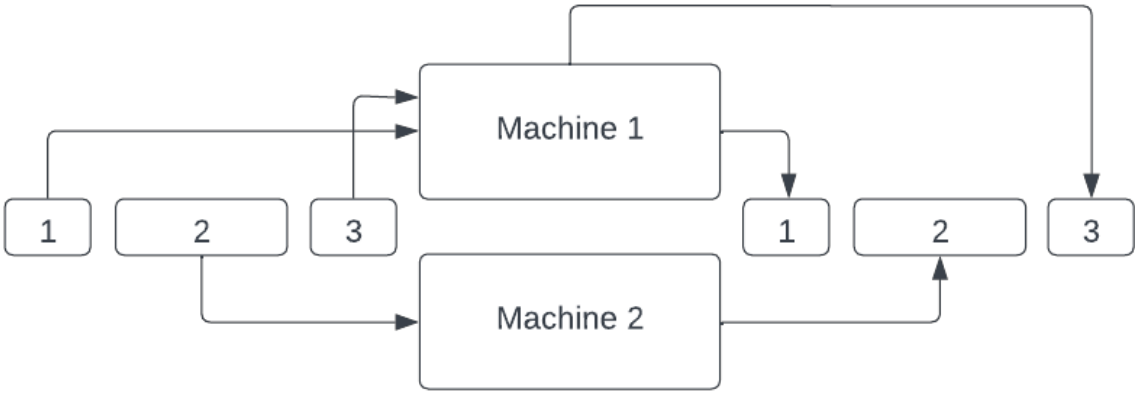
Το μοντέλο απλής μηχανής, συμφώνα με το οποίο θα επιλύσουμε ένα πρόβλημα βελτιστοποίησης στα επόμενα κεφάλαια, αποτελεί ένα μοντέλο μηχανής κατά το οποίο η κάθε εργασία εκτελείται από μια μοναδική μηχανή ακολουθιακά μαζί με άλλες εργασίες.


Σχήμα 2.4: Μοντέλο Απλής Μηχανής

### 2.3.2 Μοντέλο Παράλληλων Μηχανών

Το επίσης σημαντικό μοντέλο παράλληλων μηχανών αποτελείται από δυο ή περισσότερες μηχανές, στις οποίες οι εργασίες μπορούν να εκτελεσθούν παράλληλα. Το μοντέλο χωρίζεται σε τρεις υποκατηγορίες με βάση την τρόπο επεξεργασίας των εργασιών, που είναι οι εξής:

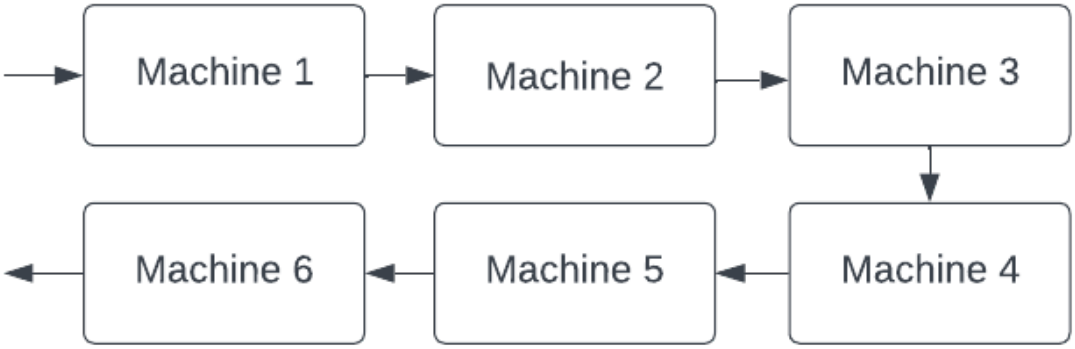
- Πανομοιότυπες παράλληλες μηχανές (Identical machines in parallel), όπου η κάθε εργασία j μπορεί να υποβληθεί σε επεξεργασία σε οποιαδήποτε από τις διαθέσιμες μηχανές m του συστήματος.
- Μηχανές παράλληλες με διαφορετικές ταχύτητες (Machines in parallel with different speeds), όπου παράγοντας της επεξεργασίας των εργασιών είναι και η ταχύτητα. Ο τύπος $p_j/v_i$ όπου vi η ταχύτητα του μηχανήματος συμβολίζει τον χρόνο pij που θα χρειαστεί για να διανύσει η εργασία j την μηχανή i. Αν η ταχύτητα όλων των μηχανών είναι ίδια τότε το περιβάλλον αυτό είναι πανομοιότυπο με το παραπάνω (Πανομοιότυπες παράλληλες μηχανές).
- Μη σχετιζόμενες παράλληλες μηχανές (Unrelated machines in parallel). Το συγκεκριμένο περιβάλλον είναι παρόμοιο με το παραπάνω, αλλά διαφέρει από αυτό ως προς το γεγονός ότι οι εργασίες αποτελούν παράγοντας που επιρρεάζει την ταχύτητα της μηχανής, η οποία στην συγκεκριμένη περίπτωση συμβολίζεται ως vij.




Σχήμα 2.5: Μοντέλο Παράλληλων Μηχανών

### 2.3.3 Μοντέλο Ροής

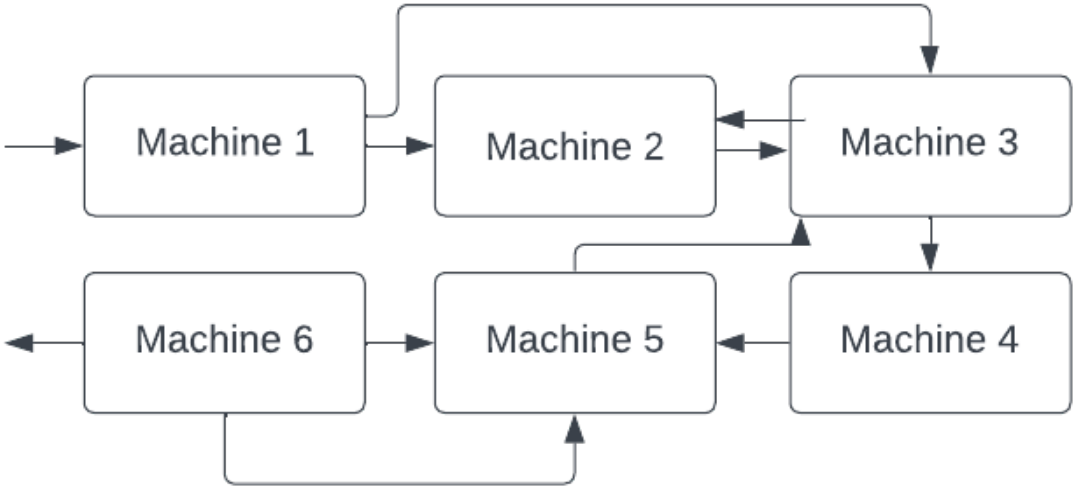
Το μοντέλο ροής, όπως και το μοντέλο παράλληλων μηχανών, αποτελείται από δυο ή περισσότερες μηχανές όμως όλες οι εργασίες του συστήματος θα πρέπει να διατηρούν μια συγκεκριμένη ακολουθία εκτέλεσης αλλά και μια συγκεκριμένη κατεύθυνση. Δηλαδή η κάθε εργασία j πρέπει από την στιγμή που θα περάσει από μια μηχανή j να περιμένει στην ούρα της μηχανής i+1 για να περάσει στο στάδιο επεξεργασίας. Συνήθως η ουρά στο σύστημα των μηχανών λειτουργεί με την ιεραρχία First in First out (FIFO).




Σχήμα 2.6: Μοντέλο Ροής

### 2.3.4 Μοντέλο Κατά Παραγγελία

Στο κατά παραγγελία μοντέλο κάθε εργασία καταλαμβάνει μια ξεχωριστή μηχανή, στην οποία αυτή καθορίζει την κατεύθυνση αλλά και άλλες παραμέτρους στο σύστημα. Τα συστήματα παραγωγής κατά παραγγελία κατηγοριοποιούνται στα συστήματα τα οποία κάθε εργασία επισκέπτεται κάθε μηχανή το πολύ μία φορά και σε εκείνα τα οποία μια εργασία μπορεί να επισκέπτεται κάθε μηχανή περισσότερες από μία φορές [16][7].




Σχήμα 2.7: Μοντέλο Κατά Παραγγελία

## 2.4 Τεχνικές Επίλυσης Προβλημάτων

Οι βασικές τεχνικές επίλυσης προβλημάτων χρονοπρογραμματισμού, είναι ο μαθηματικός προγραμματισμός (mathematical programming), ο γραμμικός προγραμματισμός (linear Programming), ο προγραμματισμός με περιορισμούς (constraint programming), και οι μεταευρετικοί αλγόριθμοι (metaheuristic algorithms), οι οποίοι αποτελούν μια τεχνική της κατηγορίας των ευρετικών αλγορίθμων. Η πρώτη κατηγορία αφορά τους αλγορίθμους που αναζητούν την βέλτιστη λύση. Αυτοί οι αλγόριθμοι λόγω της μεγάλης υπολογιστικής πολυπλοκότητας χρειάζονται αρκετό χρόνο για να φτάσουν τον στόχο τους, με αποτέλεσμα να τους χρησιμοποιούμε μόνο σε μικρά προβλήματα. Αντιθέτως, οι ευρετικές τεχνικές παρά τον μικρό υπολογιστικό χρόνο που απαιτούν δεν βρίσκουν πάντοτε βέλτιστες λύσεις. Όπως προαναφέρθηκε, ο κύριος στόχος των προβλημάτων χρονοπρογραμματισμού είναι η εύρεση βέλτιστων ή άριστων λύσεων, ωστόσο δεν είναι και ο μοναδικός. Για την χρήση μιας εκ των τεχνικών επίλυσης προβλημάτων ελέγχουμε το πόσο ευέλικτη και προσαρμόσιμη είναι σε προβλήματα διαφορετικών περιορισμών, τις απαιτήσεις της σε πόρους και τον χρόνο που χρειάζεται για την επίλυση του προβλήματος αναφορικά με τον όγκο δεδομένων που έχει να ταξινομήσει.

### 2.4.1 Μαθηματικός Προγραμματισμός - Mathematical Programming

Ο μαθηματικός προγραμματισμός αναφέρεται στα μοντέλα της επιστήμης των μαθηματικών που χρησιμοποιούνται με σκοπό την εύρεση της καλύτερης ή άριστης λύσης στα προβλήματα λήψης αποφάσεων. Τα προβλήματα αυτά αποτελούνται από μια αντικειμενική συνάρτηση της οποίας αναζητούμε το μέγιστο ή το ελάχιστο, αναλόγως του αποτελέσματος που επιζητούμε, και συνθήκες που οι μεταβλητές του συστήματος πρέπει να ικανοποιούν.[17]
Όπως και ο προγραμματισμός ηλεκτρονικών υπολογιστών έτσι και ο μαθηματικός προγραμματισμός σχεδιάζονται με σκοπό την επίλυση ενός συγκεκριμένου προβλήματος. Μπορεί το όνομα προγραμματισμός να παραπέμπει στον τομέα της πληροφορικής αλλά η σημασία του προγραμματισμού αφορά τις μεταβλητές του μοντέλου που θα προγραμμα-

τισθούν με σκοπό την βελτιστοποίηση της συνάρτησης.

Τα βήματα επίλυσης προβλημάτων με την χρήση μοντέλων του μαθηματικού προγραμματισμού:

- Μετατροπή ενός προβλήματος σε μαθηματικό μοντέλο που περιλαμβάνει τα στοιχεία του ως μεταβλητές.
- Αναζήτηση διαφορετικών λύσεων του προβλήματος.
- Εύρεση της άριστης ή της καλύτερης λύσης.

### 2.4.2 Γραμμικός Προγραμματισμός - Linear Programming

Ο γραμμικός προγραμματισμός χαρακτηρίζεται ως ένα ευρέως χρησιμοποιούμενο μοντέλο του μαθηματικού προγραμματισμού. Αποτελεί μια πολύ χρήσιμη και δυναμική τεχνική που αντλείται από προβλήματα ποικίλων τύπων και συναντάται σε πολλούς τομείς των επιστημών και της διοίκησης. Όπως υποδηλώνει και το όνομα του, ο γραμμικός προγραμματισμός αποτελείται από γραμμικές συναρτήσεις υπολογίζοντας την μικρότερη και την μεγαλύτερη τιμή του αντικειμενικού στόχου.
Αυτό που χαρακτηρίζει τον γραμμικό προγραμματισμό είναι κυρίως η χαμηλή ζήτηση σε πόρους και μεγάλη αποτελεσματικότητα, εφόσον στοχεύει την άριστη λύση του προβλήματος.

### 2.4.3 Προγραμματισμός με Περιορισμούς - Constraint Programming

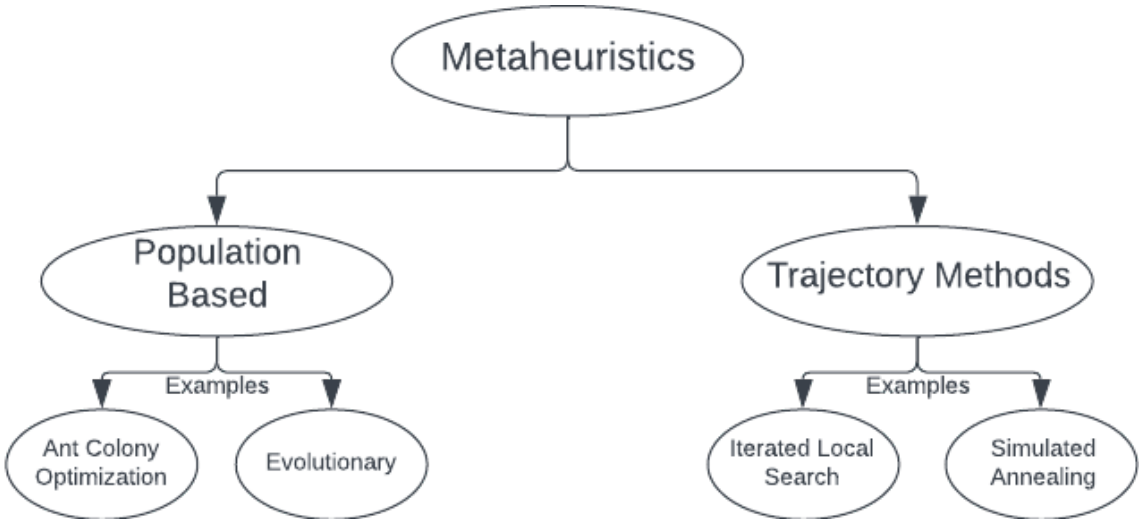
Ο προγραμματισμός με περιορισμούς βρίσκει εφαρμογή σε πλήθος βιομηχανικών προβλημάτων, όπως προβλήματα χρονοπρογραμματισμού με τα οποία θα ασχοληθούμε και εμείς. Ο προγραμματισμός με περιορισμούς ανήκει στην κατηγορία του λογικού προγραμματισμού, δηλαδή ο τρόπος αναζήτησης της λύσης του προβλήματος πραγματοποιείται με την χρήση λογικών συλλογισμών, προσπαθώντας να πλησιάσει όσο περισσότερο γίνεται την ανθρώπινη σκέψη [18].
Πιο συγκεκριμένα τα προβλήματα ικανοποίησης περιορισμών αποτελούνται από ένα σύνολο μεταβλητών και πεδίων τιμών όπως και όλα τα προβλήματα γενικά, αλλά και από ένα σύνολο περιορισμών που καλούνται να ξεπεράσουν. Ως ένα προϊόν του λογικού προγραμματισμού οι περιορισμοί αυτοί συνδέονται λογικά με τις μεταβλητές του προβλήματος και η τιμή τους είναι είτε αληθής είτε ψευδής [19].

Οι περιορισμοί του προβλήματος είναι διακριτοί στις εξής κατηγορίες:

- **Δηλωτικοί**, δηλαδή ορίζουν μια σχέση μεταξύ των οντοτήτων του προβλήματος χωρίς να ορίζουν μια συγκεκριμένη υπολογιστική διαδικασία.
- **Προσθετικοί**, γιατί ενδιαφέρει συνήθως η σύζευξη των περιορισμών και όχι η σειρά με την οποία τέθηκαν.
- **Σπανίως ανεξάρτητοι**, καθώς στη συνηθέστερη περίπτωση οι περιορισμοί έχουν κοινές μεταβλητές.

### 2.4.4 Μεταευρετικοί Αλγόριθμοι - Metaheuristic Algorithms

Οι Μεταευρετικοί αλγόριθμοι (Metaheuristic Algorithms) αποτελούν εξέλιξη των Ευρετικών αλγορίθμων με βασική τους διαφορά ότι δεν βρίσκουν απευθείας την λύση του προβλήματος αλλά καταλήγουν στην αναζήτηση ευρετικών μεθόδων για την βελτίωση της λύσης αυτής. Γενικά θεωρούνται καλύτεροι από τους Ευρετικούς Αλγορίθμούς, καθώς η συμπεριφορά τους είναι πιο ομαλή και τα αποτελέσματα τους συνήθως βέλτιστα. Επίσης σε αντίθεση με τους Ευρετικούς αλγορίθμους, οι Μεταυερετικοί δεν εγκλωβίζονται σε τοπικά ελάχιστα δίνοντας συνήθως υψηλότερης ποιότητας λύσεις, εφόσον εξετάζουν μεγαλύτερο χώρο δεδομένων και λύσεων [20].
Οι Μεταευρετικοί αλγόριθμοι σχεδιάζονται με σκοπό την αποδοτική και αποτελεσματική εξερεύνηση του χώρου των λύσεων. Αποτελούνται από δυο μηχανισμούς, την εντατικοποίηση και την διαφοροποίηση. Ο μηχανισμός εντατικοποίησης αφορά την ευφυία που πρέπει να έχουν οι Μεταευρετικοί αλγόριθμοι έτσι ώστε να δραστηριοποιούνται σε μεγαλύτερο εύρος χώρου λύσεων. Ο μηχανισμός διαφοροποίησης με την σειρά του αφορά την μετακίνηση του αλγόριθμου σε ανεξερεύνητες περιοχές με σκοπό την εύρεση άγνωστων, πιθανώς καλύτερων λύσεων.
Οι Μεταευρετικοί Αλγόριθμοι χωρίζονται σε δυο κατηγορίες με βάση τη λογική που διεξάγουν έρευνα εντός του χώρου λύσεων. Η πρώτη κατηγορία ονομάζεται μέθοδοι βασιζόμενοι στον πληθυσμό, και δραστηριοποιούνται πραγματοποιώντας στιγμιαίο έλεγχο σε ένα πλήθος λύσεων. Η δεύτερη κατηγορία ονομάζεται μέθοδοι τροχιάς και το χαρακτηριστικό τους γνώρισμα είναι ότι, σε αντίθεση με τις μεθόδους πληθυσμού, ελέγχουν στιγμιαία μια μοναδική λύση.



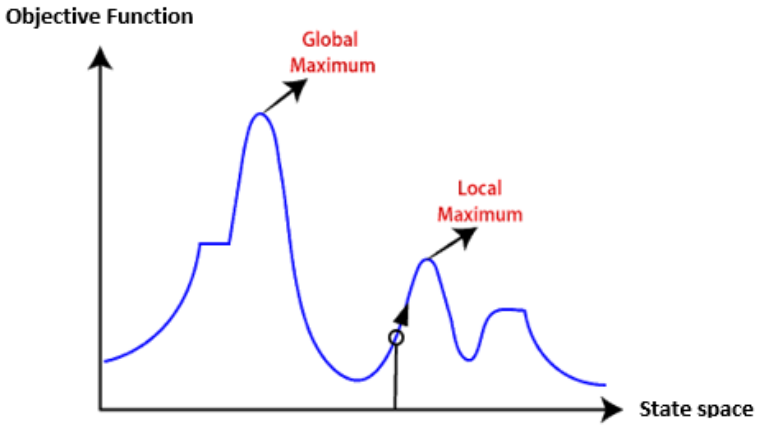

Σχήμα 2.8: Κατηγοριοποίηση Μεταευρετικών Αλγορίθμων

Μπορεί οι Μεταευρετικοί αλγόριθμοι να είναι ευρέως γνωστοί, παρά ταύτα δεν υπάρχει όρος 'Μεταευρετικοί' και 'Ευρετικοί' στο Ελληνικό και Λατινικό λεξιλόγιο. Ο όρος μεταευρετικός προέρχεται από τις λέξεις «μετά» και «ευρίσκω» και η έννοια που του αποδίδεται «είναι αναζήτηση σε υψηλότερο επίπεδο».
Κάποιες γνωστές Μεταευρετικές τέχνικες είναι η Αναζήτηση με Απαγορευμένες Καταστάσεις (Tabu Search), η Προσομοιωμένη Ανόπτηση (Simulated Annealing), την οποία θα αναλύσουμε και στο Κεφάλαιο 6, οι Αλγόριθμοι Τοπικής Αναζήτησης (Guided Local Search) και οι Γενετικοί Αλγόριθμοι (Genetic Algorithms) οι οποίοι αναφέρονται

και στον τομέα της Τεχνητής Νοημοσύνης και της Υπολογιστικής Νοημοσύνης και στις Μεταευρετικές τεχνικές.

## 2.5 Μέθοδοι Τοπικής Αναζήτησης

Οι Αλγόριθμοι Τοπικής Αναζήτησης είναι μέθοδοι εύρεσης υποβέλτιστων λύσεων, που λειτουργούν βελτιώνοντας σταδιακά μια αρχική λύση. Οι αλγόριθμοι αυτοί λειτουργούν εκτελώντας μια πολύ απλή αλληλουχία βημάτων [21]. Πρωτίστως διαλέγουν μια αρχική λύση από τον χώρο αναζήτησης και την θέτουν ως τρέχουσα, έπειτα πραγματοποιούν έναν μετασχηματισμό στην τρέχουσα λύση με σκοπό την παραγωγή μιας νέας και την αποτίμηση αυτής της νέας λύσης. Αν η νέα λύση είναι καλύτερη από την τρέχουσα τότε τις ανταλλάσσει έτσι ώστε η καλύτερη λύση να γίνει η τρέχουσα, αλλιώς την απορρίπτει. Ο αλγόριθμος επαναλαμβάνεται μέχρι κανένας νέος μετασχηματισμός να μην βελτιώνει την τρέχουσα λύση. Αυτή η τεχνική ονομάζεται επαναληπτική βελτίωση.
Η αποτελεσματικότητα των Αλγόριθμων Τοπικής Αναζήτησης εξαρτάται από τέσσερις βασικούς παράγοντες: την τιμή της αρχικής λύση, τις γειτονικές λύσεις, την συνάρτηση μετασχηματισμού η οποία μπορεί να πραγματοποιεί αναζήτηση νέων λύσεων με τυχαίο ή με άπληστο (Greedy) τρόπο, αλλά και την συνθήκη τερματισμού του αλγορίθμου. Επίσης, σε αντίθεση με πολλούς αλγορίθμους, οι Αλγόριθμοι Τοπικής Αναζήτησης είναι αποτελεσματικοί στην επίλυση προβλημάτων με μεγάλους ή άπειρους χώρους λύσεων.
Παρά την ευχρηστία τους όμως οι Αλγόριθμοι Τοπικής Αναζήτησης παρουσιάζουν προβλήματα στον τομέα της αναζήτησης καλύτερων λύσεων σε βάθος. Αυτό το πρόβλημα παρουσιάζεται καθώς δεν διαθέτουν μηχανισμούς εκτεταμένης αναζήτησης, αλλά λειτουργούν συνήθως όπως προαναφέρθηκε είτε τυχαία είτε άπληστα, εντοπίζοντας κάθε φορά την πλησιέστερη καλύτερη λύση. Αυτό έχει ως αποτέλεσμα να τερματίζουν στο κοντινότερο τοπικό μέγιστο που εξετάζουν, εφόσον οι επόμενες κοντινές λύσεις δεν είναι καλύτερες [22].




Σχήμα 2.9: Τοπική Αναζήτηση [2]

# Κεφάλαιο 3

# Ανάλυση του προβλήματος

Η συγκεκριμένη εργασία μελετά το πρόβλημα (1,Cap(t)||ΣTi), κατά το οποίο ένα σύνολο N εργασιών J = {1,...,N}, πρέπει να προγραμματιστεί με την χρήση μιας απλής μηχανής. Αυτή η μηχανή με την οποία θα πραγματοποιηθεί η παρούσα διαδικασία έχει μεταβαλλόμενη χωρητικότητα κατά την πάροδο του χρόνου. Τελικός μας στόχος είναι η ελαχιστοποίηση της αντικειμενικής συνάρτησης συνολικής καθυστέρησης.

**Χωρητικότητα απλής μηχανής:**

$$Cap(t) \tag{3.1}$$

**Αντικειμενική συνάρτηση ολικής καθυστέρησης:**

$$T(S) = \sum_{i \in j} T(i) \tag{3.2}$$

Ακόμη, είναι γνωστό ότι παρόλο που η χωρητικότητα μεταβάλλεται συνεχώς, ισχύει οτι Cap(t) > 0 για κάθε χρονική στιγμή t≥0.
Η κάθε εργασία που ανήκει στο σύνολο J είναι διαθέσιμη από την χρονική στιγμή t = 0 και έχει διάρκεια (duration) συμβολιζόμενη ως pi και ημερομηνία λήξης (due date) ως di.
Για να πραγματοποιηθεί σωστά ένα χρονοδιάγραμμα S θα πρέπει να πληροί τις παρακάτω προϋποθέσεις:

- Ο χωρητικότητα του μηχανήματος δεν πρέπει να ξεπεραστεί σε καμία χρονική στιγμή. Πρέπει πάντοτε να ισχύει X(t) < Cap(t), για όλα τα t≥0, όπου X(t) είναι η συνολική κατανάλωση του μηχανήματος στο διάστημα [t, t+1], δηλαδή ο αριθμός εργασιών που πραγματοποιείται επεξεργασία στο συγκεκριμένο διάστημα.

- Πρώτα πρέπει να υπολογίζεται ο χρόνος που απαιτείται για την ολοκλήρωση μιας εργασίας, $Ci = si + pi$, όπου si ο χρόνος έναρξης που δίνεται από την μηχανή στην κάθε εργασία.

Έπειτα για να υπολογιστεί η αντικειμενική συνάρτηση ολικής καθυστέρησης, υπολογίζουμε την καθυστέρηση της κάθε εργασίας που ολοκληρώνεται μετά την καθορισμένη ημερομηνία λήξης της di [3].

Δηλαδή:

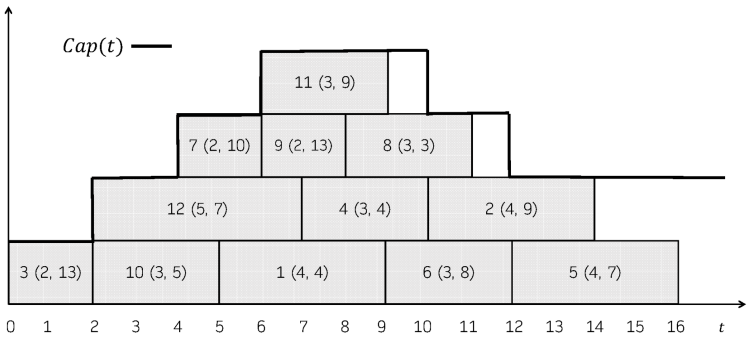
$$T(i) = max\{0, Ci - di\} \tag{3.3}$$

Και τέλος μας μένει απλώς να υπολογίσουμε την αντικειμενική συνάρτηση:

$$T(S) = \sum_{i \in j} T(i) \tag{3.4}$$

## 3.1 Schedule Builder

Για την εισαγωγή των εργασιών J = {1,...,N} στην μηχανή με βέλτιστο τρόπο θα χρησιμοποιήσουμε έναν Schedule Builder ή Schedule Generation Schema. Η μη ντετερμινιστική κατασκευαστική αυτή μέθοδος εξειδικεύεται σε προβλήματα αναζήτησης εντός ενός κενού χώρου, υπολογίζοντας βέλτιστα ή έστω εφικτά την σειρά των εργασιών και τοποθετώντας τις ακολουθιακά στον χώρο αναζήτησης [23].
Για την καλύτερη κατανόηση της έννοιας Schedule Builder μπορούμε να παρατηρήσουμε την γραφική της απεικόνιση, Σχήμα 3.1, η οποία αποτελείται από τέσσερις γραμμές χωρητικότητας και δώδεκα εργασίες συστήματος που συμβολίζονται ως εξής: αριθμός εργασίας (διάρκεια ζωής εργασίας, ημερομηνία λήξης εργασίας).




Σχήμα 3.1: Schedule Builder ή Schedule Generation Schema [3]

Η συνάρτηση ολικής καθυστέρησης του Σχήματος 3.1 είναι 37 και υπολογίζεται από την καθυστέρηση της κάθε εργασίας που ολοκληρώνεται μετά την καθορισμένη ημερομηνία λήξης της. Στο συγκεκριμένο πρόβλημα προκύπτει εφόσον 1 (T1=5), 2 (T2=5), 4 (T4=6), 5 (T5=9), 6 (T6=4) και 8 (T8=8).
Το πρόβλημα μας δεν υπάρχει προκαθορισμένος χρόνος έναρξης για την κάθε εργασία του συστήματος ξεχωριστά, συνεπώς η ταξινόμηση τους εντός της μηχανής πρέπει να γίνει σειριακά, με βάση άλλα κριτήρια. Σε προβλήματα όπως αυτό, όπου όλες οι εργασίες είναι διαθέσιμες κατά το σύνολο τους από την αρχή ($st_i = 0$) είναι αναγκαία η χρήση μεθόδων κατασκευής όπως ο Schedule Builder.
Για την σωστή κατανόηση της λειτουργίας του Schedule Builder θα πρέπει να γίνει σαφές ότι όλες οι κινήσεις του κατασκευαστή εκτελούνται από τα αριστερά προς τα δεξιά. Μη ντετερμινιστικά λοιπόν κάθε εργασία τοποθετείται στην μηχανή στην καλύτερη θέση, δηλαδή στην θέση με τον μικρότερο δυνατό χρόνο έναρξης, έτσι ώστε να μην ξεπεραστεί η χωρητικότητα της μηχανής και το πρόγραμμα μας να παραμείνει feasible δηλαδή εφικτά επιλύσιμο.

**Algorithm 1:** Schedule Builder

**Data:** A (1,Cap(t)||ΣTi) problem instance P.
**Result:** A feasible schedule S for P.
$US \leftarrow \{1, 2, ..., n\}$;
$X(t) \leftarrow 0; \forall t \geq 0$;
**while** $US \neq \emptyset$ **do**
  Non-deterministically pick job $u \in US$;
  Assign $st_u = min\{t' | \forall t \; with \; t' \leq t < t' + p_u : X(t) < Cap(t)\}$;
  Update $X(t) \leftarrow X(t) - 1; \forall t \; with \; st_u \leq t < st_u + p_u$;
  $US \leftarrow US - \{u\}$;
**end**
**return** *Feasible schedule* $S = (st1, st2, ..., stn)$;

Αν και ο βασικός μας στόχος είναι η μείωση της συνάρτησης ολικής καθυστέρησης (total tardiness), ο Schedule Builder δεν εκτελεί κάποιον έλεγχο - λειτουργία για την βελτιστοποίηση της. Συνεπώς ο ρόλος του Schedule Builder είναι καθαρά κατασκευαστικός και ο στόχος του να μην ξεπεραστεί η χωρητικότητα της μηχανής. Για την μείωση της συνάρτησης ολικής καθυστέρησης μπορούμε να εκτελέσουμε αλγόριθμους τοπικής αναζήτησης, έπειτα από την δημιουργία του κατασκευαστή, όπως ο Hill Climbing ή ο αλγόριθμος Simulated Annealing και να συγκρίνουμε τα αποτελέσματα.

Ακολουθεί ο κώδικας για την κατασκευή του Schedule Builder σε γλώσσα προγραμματισμού Python. Όπως αναλύθηκε και παράπανω ο Schedule builder που κατασκευάσαμε, με τυχαία σειρά εισάγει για την κάθε ξεχωριστή γραμμή χωρητικότητας απο δεξιά προς τα αριστερά τις εργασίες με την καλύτερο χρόνο έναρξης. Πιο συγκεκριμένα με την χρήση της εντολής random.shuffle(jobs), όλες οι εργασίες τοποθετούνται σε τυχαία σειρά. Έπειτα πραγματοποιούνται έλεγχοι για την σωστή τοποθέτηση της κάθε εργασίας, όπως εάν υπάρχει αρκετή χωρητικότητα στην θέση που η μηχανή θέλει να τοποθετήσει την εργασία που έχει σειρα, ή αν κάποια άλλη εργασία έχει τοποθετηθεί στην συγκεκριμένη θέση. Εφόσον ικανοποιηθούν οι παραπάνω έλεγχοι τότε η εργασία τοποθετείται από την μηχανή σε μια εφικτή θέση.

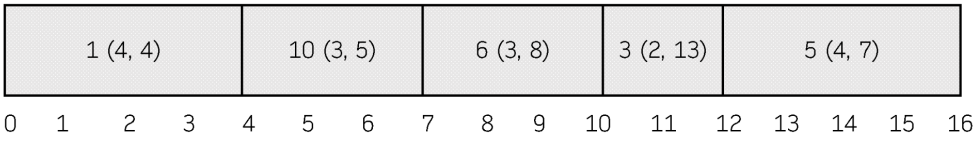
```
def schedule_builder(self, jobs=None):
    if jobs == None:
        jobs = [job_id for job_id in self.problem.jobs]
        random.shuffle(jobs)
    x = [0] * self.problem.flat_duration()
    left_bound = 0
    for job_id in jobs:
        t1 = left_bound
        job = self.problem.jobs[job_id]
        while True:
            capacity_pulse = self.problem.
  capacities_detailed[
                t1 : t1 + job.duration
            ]
```

```
14              current_demand = x[t1 : t1 + job.duration]
15              flag = True
16              for cap, dem in zip(capacity_pulse,
    current_demand):
17                  if dem >= cap:
18                      flag = False
19                      break
20              if flag:
21                  self.solution.schedule(job_id, t1)
22                  for t in range(t1, t1 + job.duration):
23                      x[t] += 1
24                      if x[t] == self.problem.
    capacities_detailed[t]:
25                          left_bound = t + 1
26                  break
27              t1 += 1
```

Listing 3.1: Schedule Builder Example

## 3.2 C-Path

Για την βελτιστοποίηση της ολικής καθυστέρησης, οι αλγόριθμοι που θα χρησιμοποιήσουμε αποτελούνται από μηχανισμούς που θα πραγματοποιήσουν εναλλαγές των εργασιών εντός της μηχανής του συστήματος. Για την απλοποίηση της διαδικασίας αυτής είναι εξαιρετικά χρήσιμη η ομαδοποίηση των εργασιών του συστήματος ανάλογα με κάποια κοινά τους χαρακτηριστικά. Από εδώ και στο εξής θα ονομάζουμε την αλληλουχία ομαδοποιημένων εργασιών ως γραμμή χωρητικότητας (C-Path). Η ονομασία αυτή προκύπτει από την λέξη χωρητικότητα και την λέξη μονοπάτι, εφόσον το κάθε C-Path αντιπροσωπεύει ένα μονοπάτι εργασιών σε μια αλληλουχία με βάση την χωρητικότητα της μηχανής.
Ειδικότερα, η δημιουργία ενός C-Path αποτελείται από δυο στάδια. Το πρώτο στάδιο αφορά την εύρεση ενός αρχικού κόμβου, δηλαδή μια εργασία εκκίνησης, η όποια μπορεί να εμφανίζεται σε ένα ή περισσότερα C-Paths. Έπειτα, στο δεύτερο και τελευταίο στάδιο, επιλέγεται η κάθε επόμενη εργασία μετά την πρώτη να έχει ίδιο χρόνο έναρξης με τον χρόνο λήξης της προηγούμενης.




Σχήμα 3.2: C-Path [3]

Ο λόγος για τον οποίο ομαδοποιούμε τις εργασίες σε γραμμές χωρητικότητας είναι για την καλύτερη αντίληψη του χώρου στον οποίο ανήκει η κάθε εργασία αλλά και για να παραμένει συνεχώς το πρόγραμμα εφικτό. Οι εναλλαγές που μπορούν να πραγματοποιηθούν μεταξύ των εργασιών μπορεί να είναι τόσο εντός του C-Path που ανήκουν, όσο και μεταξύ δυο διαφορετικών C-Path, εφόσον όμως δεν ξεπερνούν τα όρια της μηχανής.

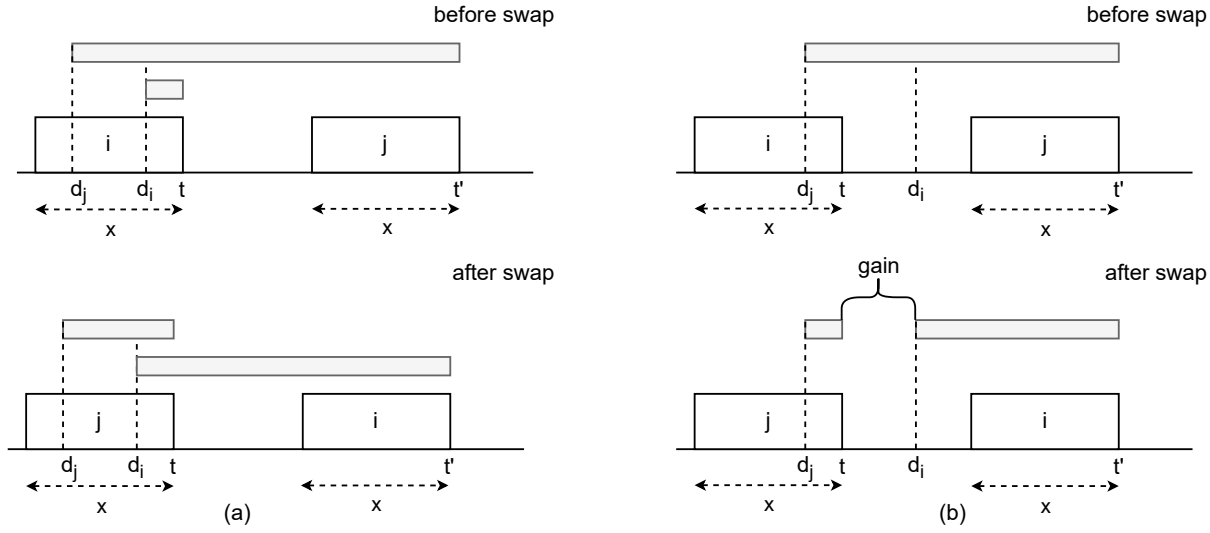
Για παράδειγμα στο Σχήμα 3.2 παρατηρούμε την ακολουθία $\pi_1 = (1, 10, 6, 3, 5)$ που προκύπτει από εσωτερικές εναλλαγές των εργασιών στο πρώτο C-Path του Σχήματος 5.1. Με αυτή την βελτίωση του C-Path παρατηρούμε ότι η συνάρτηση ολικής καθυστέρησης παίρνει την τιμή 32 σε αντίθεση με την αρχική, κατά την οποία η συνολική τιμή ήταν 37. Στον κώδικα που ακολουθεί εφαρμόζεται ο διαχωρισμός του κάθε C-path εφόσον έχουν τοποθετηθεί οι εργασίες με την χρήση του Schedule Builder στην μηχανή. Η διαδικασία αυτή περιλαμβάνει την δημιουργία μιας άδειας λίστας στην γραμμή 3 του κώδικα η οποία ονομάζεται cpaths. Ύστερα τοποθετούμε στην άδεια λίστα το κάθε C-Path, το οποίο αποτελείται από την πρώτη έως και την τελευταία εργασία για κάθε γραμμή χωρητικότητας της μηχανής ξεχωριστά.

```
1  # returns cpaths
2      def get_cpaths(self):
3          cpaths=[]
4          graph = nx.DiGraph()
5          for lane in self.lanes:
6              edges = []
7              for start_time1, job_id1 in lane:
8                  job1 = self.problem.jobs[job_id1]
9                  finish_time1 = start_time1 + job1.duration
10                 for lane2 in self.lanes:
11                     for start_time2, job_id2 in lane2:
12                         if finish_time1 == start_time2:
13                             edges.append((job_id1, job_id2)
   )
14             graph.add_edges_from(edges)
15         print(graph)
16
17         roots = []
18         leaves = []
19         for node in graph.nodes :
20             if graph.in_degree(node) == 0 :
21                 roots.append(node)
22             elif graph.out_degree(node) == 0 :
23                 leaves.append(node)
24         logging.info(f"{roots=}")
25         logging.info(f"{leaves=}")
26         for root in roots :
27             for leaf in leaves :
28                 for path in nx.all_simple_paths(graph, root
   , leaf) :
29                     cpaths.append(path)
30         return cpaths
```

Listing 3.2: C-path Example

## 3.3 Due times rule - Κανόνας του κατάλληλου χρόνου λήξης εργασιών

Ένα πολύ βασικό κομμάτι της παρούσας εργασίας αποτελούν οι αποφάσεις που λαμβάνει ο Schedule Builder όσον αφορά την χρονική ταξινόμηση των εργασιών στην μηχανή. Πιο συγκεκριμένα, για την βέλτιστη ταξινόμηση των εργασιών στην μηχανή με την ελάχιστη καθυστέρηση του συστήματος πραγματοποιείται έλεγχος που αφορά την ημερομηνία λήξης της κάθε εργασίας. Σε πολλές περιπτώσεις συναντούμε εργασίες που έχουν την ίδια διάρκεια $p_i$ επεξεργασίας και σύμφωνα λοιπόν με τον κανόνα οι εργασίες αυτές θα πρέπει να ταξινομηθούν με γνώμονα την ημερομηνία λήξης $d_i$ τους.
Εάν για παράδειγμα δύο εργασίες i και j, με ίδια διάρκεια ζωής, ταξινομηθούν με αντίστοιχη σειρά και η ημερομηνία λήξης της εργασίας j είναι μικρότερη από της i, δηλαδή $d_j < d_i$, τότε θα πρέπει να αντιστραφούν. Αυτή η αντιστροφή θα πραγματοποιηθεί αν και μόνο αν δεν επηρεάζει αρνητικά την ολική καθυστέρηση του συστήματος, δηλαδή αν ισχύει ότι $t - d_i \geq 0$ και $t' - d_j \geq 0$. Έτσι μετά την αντιστροφή η καθυστέρηση της εργασίας j προκύπτει από τον τύπο $t - d_j > t - d_i \geq 0$ και της εργασίας i είναι $t' - d_i$. Τελικά η ολική συνάρτηση πριν την αντιστροφή είναι $(t - d_i) + (t' - d_j)$ και είναι ίση με την ολική καθυστέρηση μετά την αντιστροφή που είναι $(t - d_j) + (t' - d_i)$ [4].
Στο πάνω μέρος του Σχήματος 3.3 παριστάνεται η συνθήκη που επικρατεί κατά την διάρκεια που η εργασία i βρίσκεται μπροστά από την j, ενώ στο χαμηλότερο μέρος απεικονίζεται η κατάσταση μετά από την αντιστροφή των δυο εργασιών. Η γκρι μπάρα που εμφανίζεται και στα τέσσερα σχήματα αποτελεί την καθυστέρηση του συστήματος. Μετά την αντιστροφή των δυο εργασιών παρατηρούμε μείωση της ολικής καθυστέρησης, όπως ήταν αναμενόμενο.




Σχήμα 3.3: Αναπαράσταση της διαδικάσιας αναστροφής δυο εργασιών [4]

# Κεφάλαιο 4

# Περιγραφή Δεδομένων Προβλήματος

Τα δεδομένα που χρειαζόμαστε [24] για την επίλυση του προβλήματος (1,Cap(t)||ΣTi) χωρίζονται σε δυο διαφορετικές κατηγορίες, τα συστατικά στοιχεία της κάθε εργασίας και την χωρητικότητα της μηχανής την κάθε χρονική στιγμή. Οι κατηγορίες αυτές στο σύνολο δεδομένων και πιο συγκεκριμένα στις πρώτες δυο γραμμές εμφανίζονται ως "NOP" και "NINT", και συμβολίζουν τον αριθμό των γραμμών που θα ακολουθήσουν για την αναπαράσταση των εργασιών και της χωρητικότητας αντίστοιχα.

Πίνακας 4.1: Παράδειγμα Προβολής Συνόλου Δεδομένων

| NOP: 12<br>NINT: 6 | | |
|---|---|---|
| 0 | 2 | 1 |
| 2 | 4 | 2 |
| 4 | 6 | 3 |
| 6 | 10 | 4 |
| 10 | 12 | 3 |
| 12 | 300000 | 2 |
| 1 | 4 | 4 |
| 2 | 4 | 9 |
| 3 | 2 | 13 |
| 4 | 3 | 4 |
| 5 | 4 | 7 |
| 6 | 3 | 8 |
| 7 | 2 | 10 |
| 8 | 3 | 3 |
| 9 | 2 | 13 |
| 10 | 3 | 5 |
| 11 | 3 | 9 |
| 12 | 5 | 7 |

Σύμφωνα με τον Πίνακα 4.1 οι πρώτες δυο γραμμές περιέχουν το "NOP" και το "NINT", που όπως προαναφέραμε, συμβολίζουν τον αριθμό των γραμμών που θα ακολουθήσουν για την αναπαράσταση των εργασιών που στο παρόν παράδειγμα είναι 12 και της χωρητικότητας είναι 6. Στην τρίτη γραμμή μέχρι και την όγδοη ακολουθούν τα δε-

δομένα που αφορούν τη χωρητικότητα της μηχανής και στις υπόλοιπες δώδεκα γραμμές τα δεδομένα των εργασιών. Τα δεδομένα του Πίνακα 4.1 αναπαριστώνται γραφικά στο Σχήμα 5.1 με την χρήση του Schedule Builder.

Στο παρακάτω κομμάτι του κώδικα βλέπουμε στην πράξη πως γίνεται η ανάγνωση του κάθε αρχείου δεδομένων. Αναλυτικά, για τις πρώτες δυο γραμμές του αρχείου ο κώδικας αγνοεί τις λέξεις "NOP" και "NINT" και διαβάζει μόνο τους αριθμούς που τις συνοδεύουν. Έπειτα με βάση τους αριθμούς αυτούς υπολογίζει τον αριθμό των γραμμών που οι εργασίες και η χωρητικότητα της μηχανής καταλαμβάνουν. Στις τρεις τελευταίες γραμμές του κώδικα που ακολουθεί διαβάζουμε τα συστατικά στοιχεία της κάθε εργασίας, αγνοώντας τις πρώτες 6 γραμμές που αφορούν την χωρητικότητα της μηχανής.

```
def __init__(self, filename):
    with open(filename) as f:
        lines = f.readlines()
        self.nop = int(lines[0].split(":")[1])
        self.nint = int(lines[1].split(":")[1])
        self.jobs = {}
        for i in range(self.nop):
            line = [int(x) for x in lines[i + 2 + self.nint
  ].split()]
            self.jobs[line[0]] = Job(*line)
```

Listing 4.1: Data Import Example

## 4.1 Αναγνωριστική Ονομασία Συνόλων Δεδομένων

Για την ευκολότερη εύρεση του κάθε συνόλου δεδομένων η κάθε ονομάσια αποτελείται από την εξής ακολουθία "i< $n$ > _ < $MC$ > _ < $id$ >.txt" κατά την οποία γίνονται διακριτά στοιχεία όπως ο συνολικός αριθμός των εργασιών του προβλήματος (n), η μέγιστη χωρητικότητα της μηχανής (MC) και την αρίθμηση του κάθε συνόλου δεδομένων (id), αντίστοιχα. Το γράμμα "i" στην αρχή της κάθε ονομασίας προκύπτει ως συντομογραφία από την λέξη "instances" ως αναγνωριστικό και όχι ως κάποια πληροφορία για το σύνολο δεδομένων.
Για παράδειγμα η ονομασία του συνόλου δεδομένων του Πίνακα 4.1 θα ήταν i12_4_1.txt.

## 4.2 Εργασίες Συστήματος

Στόχος του προβλήματος (1,Cap(t)||ΣTi) είναι η ταξινόμηση της κάθε εργασίας εντός των πλαισίων της διαθέσιμης μηχανής με τον καλύτερο δυνατό τρόπο χρονικά. Στο κάθε σύνολο δεδομένων, N εργασίες J = {1,...,N} αποτελούνται από την αρίθμηση τους (index), την χρονική διάρκεια επεξεργασίας τους από την ενεργή μηχανή (duration) και την ημερομηνία λήξης τους (due date), όπως φαίνεται και στον Πίνακα 4.2.

## 4.3 Χωρητικότητα Μηχανής

Η χωρητικότητα της μηχανής είναι ένας πολύ σημαντικός περιορισμός του προβλήματος μας, καθώς πρέπει αυστηρά να μην ξεπεραστεί σε καμία χρονική στιγμή από τις εργασίες

Πίνακας 4.2: Πίνακας εργασιών

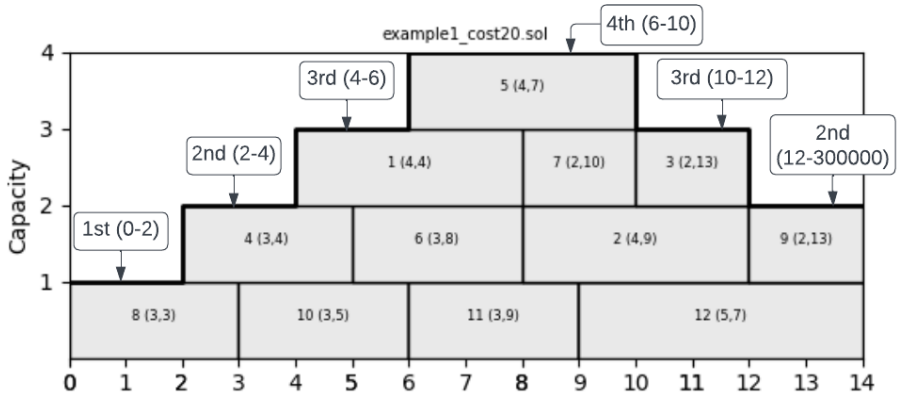
| i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| pi | 4 | 4 | 2 | 3 | 4 | 3 | 2 | 3 | 2 | 3 | 3 | 5 |
| di | 4 | 9 | 13 | 4 | 7 | 8 | 10 | 3 | 13 | 5 | 9 | 7 |

του συστήματος. Έτσι λοιπόν καλούμαστε να προσαρμόσουμε των κώδικα μας έτσι ώστε να μην ξεπεραστεί σε καμία χρονική στιγμή.
Το κάθε ξεχωριστό διάστημα χωρητικότητας της μηχανής, όπως φαίνεται και στον Πίνακα 4.3, αποτελείται από την έναρξή του (start), την λήξη του (end) και την χωρητικότητα της μηχανής την παρούσα χρονική στιγμή (capacity).

Πίνακας 4.3: Πίνακας Χωρητικότητας Μηχανής

| Διάστημα | Έναρξη | Λήξη | Χωρητικότητα |
|---|---|---|---|
| 1ο | 0 | 2 | 1 |
| 2ο | 2 | 4 | 2 |
| 3ο | 4 | 6 | 3 |
| 4ο | 6 | 10 | 4 |
| 5ο | 10 | 12 | 3 |
| 6ο | 12 | 300000 | 2 |

Ο Πίνακας 4.3 αποτελεί ένα απόσπασμα του Πίνακα 4.1, και πιο συγκεκριμένα το κομμάτι «NINT» που αφορά την χωρητικότητα της μηχανής και όχι τις εργασίες. Στην τελευταία στήλη του συγκεκριμένου πίνακα, μπορούμε να παρατηρήσουμε την κάθετη χωρητικότητα της μηχανής για το κάθε διάστημα ξεχωριστά, δηλαδή πόσες εργασίες μπορούν να επεξεργαστούν παράλληλα την ίδια χρονική στιγμή. Τέλος, στην δεύτερη και στην τρίτη στήλη αναγράφεται η έναρξη και η λήξη του κάθε διαστήματος. Στο Σχήμα 4.1 μπορούμε να παρατηρήσουμε τι αντιπροσωπεύει το κάθε στοιχείο του Πίνακα 4.3 γραφικά.




Σχήμα 4.1: Σχηματισμός της χωρητικότητας

Ο κώδικας που αναπτύσσεται παρακάτω αφορά την αναγνώριση της χωρητικότητας, ΄διαβάζοντας' από το σύνολο δεδομένων τις γραμμές που αντιστοιχούν στην κατηγορία «NINT». Ο κώδικας μέσω επαναλήψεων ΄διαβάζει' την κάθε γραμμή, που ισοδυναμεί με ένα διάστημα, και με την χρήση της εντολής split(), στην γραμμή 5 και 6 του κώδικα, διαχωρίζει και

τοποθετεί τα περιεχόμενα των τριών στηλών εντός μιας ονομαστικής πλειάδας (Named Tuple), στα πεδία start, end, capacity, που συμβολίζουν την έναρξη, λήξη και την χωρητικότητα του κάθε διαστήματος ξεχωριστά, με την σειρά που πραγματοποιείται η ανάγνωση τους.

```
# self.capacities is a list of tuples of the form: start, end, capacity
    self.capacities = []
    CapacityInterval = namedtuple("CapacityInterval", "start, end, capacity")
    for i in range(self.nint):
        start, end, cap = (int(n) for n in lines[i + 2].split())
        self.capacities.append(CapacityInterval(start, end, cap))
    flat_duration = self.flat_duration()
    # adjust finish time of last interval
    self.capacities[-1] = CapacityInterval(
        self.capacities[-1].start, flat_duration, self.capacities[-1].capacity
    )
# self.capacities_detailed is a list of values that at each index t correspond to the capacity at time t
    self.capacities_detailed = []
    intervals = []
    capacities = []
    for _, end, cap in self.capacities:
        intervals.append(end)
        capacities.append(cap)
    for t in range(flat_duration):
        self.capacities_detailed.append(capacities[np.digitize(t, intervals)])
```

Listing 4.2: Capacity Example

# Κεφάλαιο 5

# Εργαλεία και Βιβλιοθήκες

## 5.1 Dataclasses - Κλάσεις Δεδομένων

Στον κώδικα μας θα χρησιμοποιήσουμε τις κλάσεις δεδομένων της Python. Οι κλάσεις δεδομένων αποτελούν ένα εργαλείο δημιουργίας δομημένων κλάσεων με στόχο την αποθήκευση των δεδομένων των αντικειμένων αυτόματα, δηλαδή χωρίς την συγγραφή μεθόδων, όπως οι κατασκευαστές. Στην Python οι κλάσεις δεδομένων ήταν διαθέσιμες από την έκδοση 3.6 μέσω του pip, ενώ από την 3.7 και μετά ήταν έτοιμες για χρήση χωρίς επιπλέον εγκατάσταση.
Παρόλο που μια κλάση δεδομένων παρέχει όλες τις δυνατότητες μιας απλής κλάσης αντικειμένων, λόγω της έλλειψης κατασκευαστών και μεθόδων αναπαράστασης καθίσταται απλούστερη και πιο εύκολη στον χειρισμό της.

Οι κλάσεις δεδομένων γράφονται με τον εξής τρόπο:

```
@dataclass
class Job:
    id: int
    duration: int
    due: int
```

Όπως παρατηρείται και στο παράδειγμα η συγγραφή τους είναι πολύ λιτή. Η κλάση αποτελείται μόνο από το αναγνωριστικό της "@dataclass" και τα πεδία id, duration και due.

## 5.2 Logging

Στον χώρο της πληροφορικής ένα log αποτελεί ένα αρχείο που καταγράφει είτε συμβάντα ενός λειτουργικού συστήματος, είτε προβλήματα ενός λογισμικού, είτε μηνύματα μεταξύ χρηστών ενός λογισμικού επικοινωνίας. Πιο συγκεκριμένα logging ονομάζεται η διαδικασία εντοπισμού συμβάντων του λογισμικού που εκτελείται και αποτελεί ένα εργαλείο της Python σύμφωνα με το οποίο ο χρήστης έχει την δυνατότητα, με την χρήση προκαθορισμένων εντολών, να εμφανίζει αυτά τα συμβάντα στον χρήστη κατά την εκτέλεση του κώδικα.

Το logging αποτελείται από 5 επίπεδα δυνατοτήτων:

- Debug: Χρησιμοποιείται μόνο για την διάγνωση προβλημάτων.
- Info: Βεβαιώνει τον χρήστη ότι όλα δουλεύουν όπως προβλέπεται.
- Warning: Προειδοποιεί το σύστημα για απρόβλεπτα λάθη που συνέβησαν.
- Error: Ενημερώνει τον χρήστη ότι συγκεκριμένες λειτουργίες του προγράμματος δεν εκτελέστηκαν λόγω κάποιου σφάλματος.
- Critical: Εμφανίζει τα σοβαρά σφάλματα του συστήματος, που δεν μπορούν να παρακαμφθούν.

Πίνακας 5.1: Πίνακας Επιπέδων-Εντολών Logging

| Επίπεδο | Εντολή |
|---|---|
| Debug | Logging.debug(message) |
| Info | Logging.info(message) |
| Warning | Logging.warning(message) |
| Error | Logging.error(message) |
| Critical | Logging.critical(message) |

## 5.3 Named Tuples - Πλειάδα Ονομάτων

Οι πλειάδες ονομάτων (Named Tuples) είναι βασικές λειτουργίες της Python συμφώνα με τις οποίες ο χρήστης έχει την δυνατότητα να δημιουργήσει πλειάδες (tuples) με πεδία που περιέχουν λέξεις. Ο λόγος για τον οποίο δημιουργήθηκαν αφορά την βελτίωση της ανάγνωσης του κώδικα, εφόσον πλέον για την πρόσβαση στα πεδία της πλειάδας θα πρέπει να κληθεί το όνομα του πεδίου και όχι η θέση του [25].

Παράδειγμα δήλωσης και κλήσης μιας πλειάδας ονομάτων αντίστοιχα:

- CapacityInterval = namedtuple("CapacityInterval", "start, end, capacity")
- print(CapacityInterval.start)

## 5.4 Unittest - PyUnit Framework

Το unittest ή PyUnit χαρακτηρίζεται ως το πρώτο framework δοκιμής λογισμικού της Python. Η δομή λειτουργίας είναι παρόμοια με όλα τα frameworks δοκιμής λογισμικού των υπόλοιπων γλωσσών προγραμματισμού ένω την βασική του έμπνευση αποτελεί το JUnit της Java. Η νέα έκδοση του unittest είναι το unittest2 στο οποίο υπάρχουν πρόσθετα χαρακτηριστικά και παροχές σε σχέση με το πρωτότυπο. Αντίστοιχα framework παρόμοια με το unittest είναι το PyTest και το DocTest.
Έχοντας εμπνευστεί από το JUnit της Java, εμπεριέχει στοιχεία του όπως την αυτοματοποίηση των δοκιμών λογισμικού, υποστηρίζοντας δοκιμές σε ανεξάρτητο αλλά και σε

ομαδοποιημένο κώδικα.
Η βασική ροή εργασιών του framework είναι η εξής:

1. Εισάγουμε το unittest στο πρόγραμμα μας με την χρήση της import.

```
import unittest
```

2. Δημιουργούμε μια ειδική κλάση και καλούμε το framework.

```
class TestModel(unittest.TestCase):
```

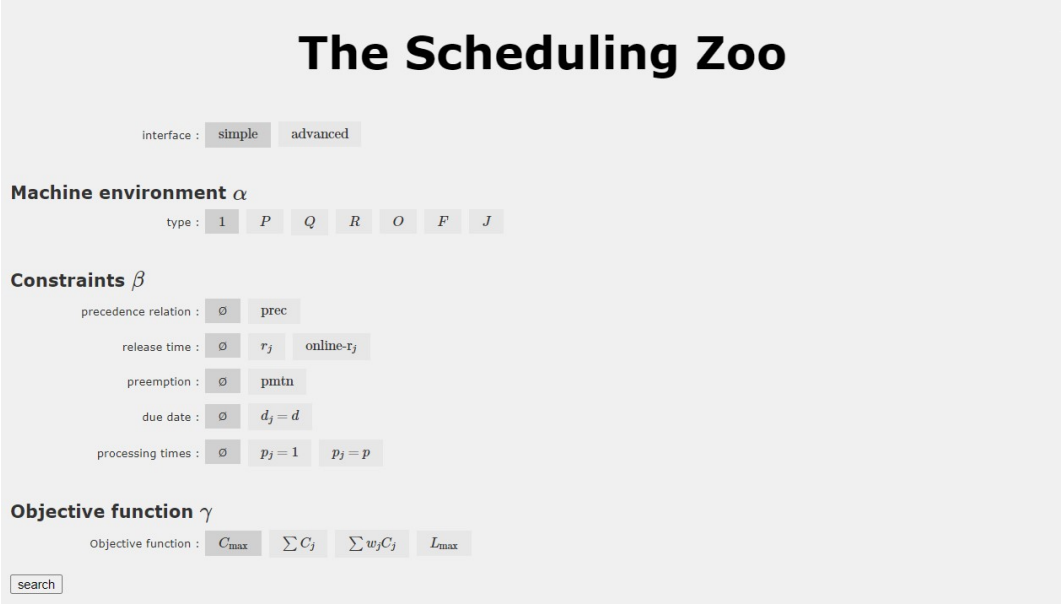
3. Κατασκευάζουμε συναρτήσεις εντός της κλάσης για την εκτέλεση εργασιών δοκιμής κώδικα.

4. Χρησιμοποιούμε μια συνάρτηση τύπου main() κατά αυτόν τον τρόπο:

```
unittest.main()
```

## 5.5 The Scheduling Zoo

Το Scheduling Zoo αποτελεί ένα βοηθητικό εργαλείο βιβλιογραφίας, το οποίο λειτουργεί εισάγοντας τις παραμέτρους του προβλήματος χρονοπρογραμματισμού που θέλουμε να επιλύσουμε στην διεπαφή χρήστη της εφαρμογής, όπως φαίνεται στο σχήμα που ακολουθεί.




Σχήμα 5.1: The Scheduling Zoo Interface [1]

Οι παράμετροι δηλαδή που μπορεί ο χρήστης να επιλέξει, είναι το περιβάλλον της μηχανής την οποία θα χρησιμοποιήσει στο πρόβλημα του, τα χρονικά όρια που θα επιτρέπει η μηχανή στις εργασίες να πραγματοποιηθούν και την αντικειμενική συνάρτηση που θα χρησιμοποιήσει. Στην συνέχεια η εφαρμογή μας εμφανίζει επιστημονικά άρθρα και βιβλία τα οποία διαπραγματεύονται προβλήματα Χρονοπρογραμματισμού ίδια ή παρόμοια με το πρόβλημα μας.

## 5.6 IBM ILOG CPLEX

Ο CPLEX αποτελεί έναν υψηλής απόδοσης επιλυτή προβλημάτων μαθηματικού προγραμματισμού. Αρχικά είχε δημιουργηθεί με σκοπό να εξυπηρετεί μόνο την γλώσσα προγραμματισμου C αλλά πλέον εφαρμόζεται και σε άλλες γλώσσες προγραμματισμού όπως την Python, την οποία χρησιμοποιούμε. Οι λόγοι που η χρήση του είναι εξαιρετικά σημαντική είναι η αποδοτικότητα, το χαμηλό κόστος ή αλλιώς χρήση λιγότερων πόρων και αύξηση του κέρδους. Πιο συγκεκριμένα θα λέγαμε ότι, λόγω των μηχανισμών που ο επιλυτής CPLEX περιέχει, είναι ικανός να λύσει πολύ μεγάλα προβλήματα με μεγάλη αποδοτικότητα σε πολύ μικρό χρονικό διάστημα και με ελάχιστους πόρους. Συνεπώς, η μεγάλη ταχύτητα εύρεσης λύσεων σε συνδυασμό με την χαμηλή ζήτηση σε πόρους καθιστούν την διαδικασία αρκετά κερδοφόρα [26].
Οι κατηγορίες προβλημάτων μαθηματικού προγραμματισμού που επιλύει ο CPLEX είναι ο προγραμματισμός με περιορισμούς (constraint programming), ο ακέραιος προγραμματισμός (mixed integer programming) και ο τετραγωνικός προγραμματισμός (quadratic programming).
Η εισαγωγή του CPLEX στην γλώσσα προγραμματισμού Python είναι πολύ απλή και πραγματοποιείται με την δήλωση της στην αρχή του κώδικα με σκοπό να χρησιμοποιηθούν έπειτα οι μηχανισμοί του. Στο παράδειγμα που ακολουθεί, πιο συγκεκριμένα, δηλώνεται το μοντέλο CP Optimizer του επιλυτή CPLEX, το οποίο θα χρησιμοποιήσουμε για την επίλυση του προβλήματος μας.

```
1 import docplex.cp.model as cpx
```

## 5.7 Google OR-Tools

Ένας παρόμοιος επιλυτής με τον IBM ILOG CPLEX είναι ο επιλυτής ανοιχτού κώδικα OR-Tools της Google. Ο επιλυτής OR-Tools αποτελεί μια δωρεάν σουίτα λογισμικού που δημιουργήθηκε από την Google για την επίλυση προβλημάτων Γραμμικού Προγραμματισμού (Linear Programming), Ακέραιου Προγραμματισμού (Mixed Integer Programming), Προγραμματισμού με Περιορισμούς (Constraint Programming), Δρομολόγησης Οχημάτων (Vehicle Routing) αλλά και για προβλήματα Βελτιστοποίησης [27].

Ο επιλυτής OR-Tools είναι προγραμματισμένος σε γλώσσα προγραμματισμού C++ αλλά χρησιμοποιείται και σε εφαρμογές της Java, της Python και σε C#. Η εγκατάσταση του είναι πολύ απλή και πιο συγκεκριμένα για την γλώσσα προγραμματισμού Πψτηον, στην οποία επικεντρωνόμαστε αρκεί στον υπολογιστή μας να υπάρχει εγκατεστημένη η γλώσσα από την έκδοση 3.6 και μετα, να έχουμε εγκαταστήσει τον διαχειριστή πακέτων λογισμικού ΠΙΠ και έπειτα να εκτελέσουμε στο τερματικό μας την παρακάτω εντολή.

```
1 python -m pip install --upgrade --user ortools
```

Στην συνέχεια για να χρησιμοποιήσουμε τα μοντέλα του Google OR-Tools και πιο συγκεκριμένα το μοντέλο του Προγραμματισμού με Περιορισμούς(Constraint Programming) αρκεί απλώς να το δηλώσουμε στο αρχείο που θέλουμε να χρησιμοποιήσουμε με την παρακάτω εντολή.

```
1 from ortools.sat.python import cp_model
```

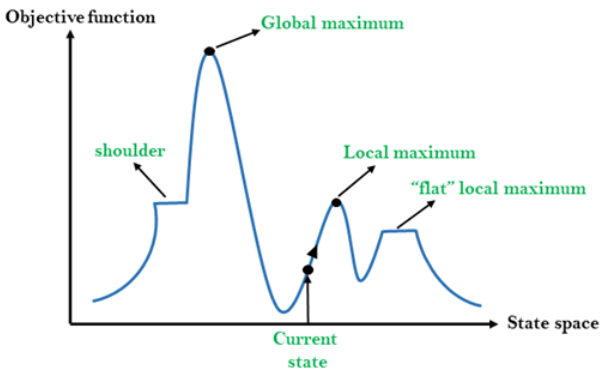
# Κεφάλαιο 6

# Επίλυση του Προβήματος με την χρήση αλγορίθμων Βελτιστοποίησης

Όπως προαναφέρθηκε και στο Κεφάλαιο 3 για την εισαγωγή τον εργασιών στην διαθέσιμη μηχανή με τουλάχιστον εφικτό τρόπο χρησιμοποιούμε τον Schedule Builder ή Schedule Generation Schema. Στα περισσότερα προβλήματα όμως δεν αναζητούμε μια εφικτή λύση αλλά μια αποτελεσματική ή βέλτιστη, για αυτόν τον λόγο λοιπόν χρησιμοποιούμε και τους αλγορίθμους βελτιστοποίησης.
Στην υλοποίηση του προβλήματος που καλούμαστε να επιλύσουμε θα χρησιμοποιήσουμε επιπλέον από τον Schedule Builder ή Schedule Generation Schema, τον αλγόριθμο Αναρρίχησης Λόφων (Hill Climbing) και τον αλγόριθμο Προσομοιωμένης Ανόπτησης (Simulated Annealing).

## 6.1 Hill Climbing

Ο αλγόριθμος της Αναρρίχησης Λόφων (Hill Climbing) αποτελεί μια διαδικασία βελτιστοποίησης και ανήκει στην κατηγορία αλγορίθμων τοπικής αναζήτησης όπως και ο αλγόριθμος της Προσομοιωμένης Ανόπτησης (Simulated Annealing) που θα αναλύσουμε στο επόμενο υποκεφάλαιο.




Σχήμα 6.1: Hill Climbing Algortihm [5]

Ο αλγόριθμος δραστηριοποιείται αναζητώντας με άπληστο τρόπο πάντοτε την καλύτερη πιθανή λύση. Έτσι σε κάθε κίνηση του αναζητεί στον χώρο, τον οποίο καλείται να εξετάσει την λύση, που θα μειώσει το κόστος της αντικειμενικής συνάρτησης. Αν η υποψήφια λύση που βρήκαμε στον χώρο συγκριθεί με την ήδη υπάρχουσα λύση και βελτιώνει το κόστος λύσης της συνάρτησης τότε την αντικαθιστά με την υπάρχουσα. Για την σύγκριση των δύο λύσεων, δηλαδή της βέλτιστης και της υποψήφιας λύσης αλλά και την παραγωγή νέων, ο αλγόριθμος χρησιμοποιεί τους τελεστές γειτνίασης. Οι τελεστές γειτνίασης ορίζουν ποιες λύσεις είναι γειτονικές, με σκοπό ο αλγόριθμος να επιλέξει ποια θα είναι η επόμενη λύση που θα εξετάσει, αλλά συντελεί επίσης και στην σύγκριση της με την βέλτιστη λύση.

**Algorithm 2:** Hill Climbing

```
Function HillClimbing(individual): Solution
newIndividual ⟵ emptysolution
begin while Individual is not improved do
    for Number of new solution do
        newIndividual ⟵ NeighbourhoodSelectionHeuristic(individual)
        if newIndividual is better than individual then
            individual ⟵ newIndividual
        end
    end
    return individual
end
```

Λόγω της άπληστης συμπεριφοράς του ως προς την αναζήτηση λύσης αλλά και λόγω της έλλειψης κάποιου μηχανισμού ελέγχου λύσεων, ο αλγόριθμος της Αναρρίχησης λόφων αδυνατεί να αναζητήσει εντός του χώρου αναζήτησης μια καλύτερη λύση από την ήδη υπάρχουσα και κατά συνέπεια παραμένουν σε τοπικά βέλτιστα και τερματίζουν εκεί. Αυτό έχει ως αποτέλεσμα να μην βρίσκουν πάντα την βέλτιστη λύση του προβλήματος και γι' αυτό τον λόγο δεν είναι τόσο αποδοτικοί όσο άλλοι αλγόριθμοι τοπικής αναζήτησης. Ο αλγόριθμος λειτουργεί για δυο κατηγορίες προβλημάτων, τα προβλήματα ελαχιστοποίησης και τα προβλήματα μεγιστοποίησης. Τα πιο συνηθισμένα προβλήματα που χρησιμοποιούνται από τις παραπάνω κατηγορίες είναι τα προβλήματα ελαχιστοποίησης, που έχουν σκοπό την μείωση του κόστους λύσης.

Στον αλγόριθμο που ακολουθεί βλέπουμε μια απλή υλοποίηση του αλγορίθμου βελτιστοποίησης Αναρρίχησης Λόφων (Hill Climbing), στην γλώσσα προγραμματισμού Python.

```
1 def hillClimbing(tsp):
2     #create a random solution and calculate its route
    length
3     currentSolution = randomSolution(oms)
4     currentRouteLength = routeLength(oms, currentSolution)
5
6     #find a neighbour
7     neighbours = getNeighbours(currentSolution)
8     bestNeighbour, bestNeighbourRouteLength =
    getBestNeighbour(oms, neighbours)
9
```

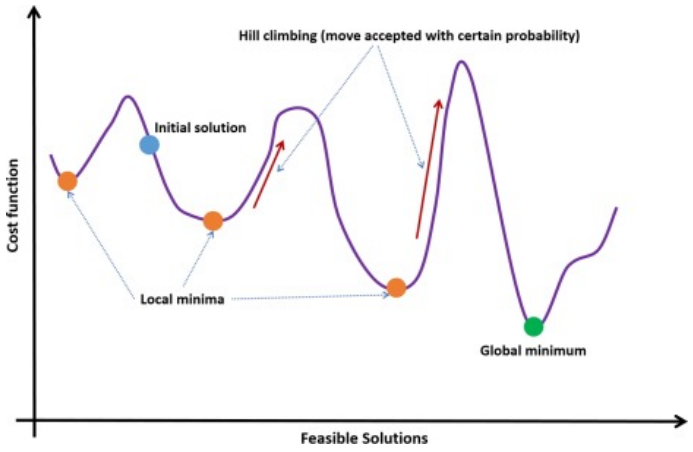
```
    #if the current solution is better than the best one
    #add the current solution as the best solution
    while bestNeighbourRouteLength < currentRouteLength:
        currentSolution = bestNeighbour
        currentRouteLength = bestNeighbourRouteLength
        neighbours = getNeighbours(currentSolution)
        bestNeighbour, bestNeighbourRouteLength =
getBestNeighbour(oms, neighbours)

    return currentSolution, currentRouteLength
```

Listing 6.1: Hill Climbing Algorithm

## 6.2 Simulated Annealing

Η Προσομοιωμένη Ανόπτηση αποτελεί μια από τις βασικές μεταευρετικές μεθόδους που χρησιμοποιούνται στον χώρο τις βελτιστοποίησης για την επίλυση μεγάλων προβλημάτων και οφείλει την ύπαρξη της στην φυσική και την επιστήμη υλικών. Ο αλγόριθμος είναι εμπνευσμένος από την ανόπτηση των μετάλλων, μια διαδικασία προσαρμογής του μετάλλου στην κατάλληλη σκληρότητα και ευθραυστότητα που ζητείται. Αυτό επιτυγχάνεται θερμαίνοντας το υλικό και έπειτα ψύχοντας το σταδιακά, αυστηρά με σταθερό ρυθμό.




Σχήμα 6.2: Simulated Annealing Algortihm [6]

Ο τρόπος με τον οποίον δρα ο αλγόριθμος της Προσομοιωμένης Ανόπτησης (Simulated Annealing) είναι πολύ πιο πολύπλοκος από αυτόν της Αναρρίχησης Λόφων (Hill Climbing). Αρχικά ο αλγόριθμος ξεκινάει με μια αρκετά μεγάλη τιμή θερμοκρασίας T. Ο αλγόριθμος λειτουργεί με την χρήση διαδοχικών επαναλήψεων, όπου στην κάθε μια κινείται σε μια νέα γειτονία υποψήφιων λύσεων. Η κάθε λύση όπου η ο Μεταευρετικός αλγόριθμος επιλέγει ονομάζεται τρέχουσα και συμβολίζεται ως $s_0$, ενώ η βέλτιστη πιθανή λύση συμβολίζεται ως $s_{best}$. Για κάθε επανάληψη που πραγματοποιείται, επιλέγεται μια υποψήφια λύση με τυχαίο τρόπο και αξιολογείται με βάση το κόστος της από την αντικειμενική συνάρτηση του αλγορίθμου. Αν το κόστος λύσης της είναι μικρότερο από το κόστος της βέλτιστης λύσης $s_{best}$, τότε ορίζουμε αυτή ως βέλτιστη. Σε περίπτωση που το κόστος λύσης της δεν είναι μικρότερο από την βέλτιστη $s_{best}$ εξετάζεται η διαφορά d στο κόστος

της υποψήφιας και της τρέχουσας λύσης. Αν $d > 0$ τότε ορίζεται μια πιθανότητα κόστους λύσης και γίνεται αποδεκτή σύμφωνα με αυτή την πιθανότητα αποδοχής. Η πιθανότητα αποδοχής μειώνεται εκθετικά σύμφωνα με την ακαταλληλότητα της εκάστοτε κίνησης, είτε σύμφωνα με την θερμοκρασία που αναφέρεται στον τύπο σχηματισμού της πιθανότητας.

$$p = e^{(-d/t)} \tag{6.1}$$

Στην συνέχεια όπως και στην Ανόπτηση των μετάλλων κατά την οποία ψύχουμε με σταθερό ρυθμό για να σταθεροποιήσουμε στο επιθυμητό σχήμα το μέταλλο που έχουμε θερμάνει, έτσι και στον αλγόριθμο της Προσομοιωμένης Ανόπτησης με την χρήση του συντελεστή ψύξης a και τον συντελεστή θερμοκρασίας t, τους οποίους πολλαπλασιάζουμε σε κάθε επανάληψη, μειώνουμε σταθερά την θερμοκρασία. Με αυτή την διαδικασία σταδιακά παίρνουμε πιο σταθερές τιμές κόστους λύσης. Ο αλγόριθμος τερματίζεται είτε μετά από κάποιον συγκεκριμένο αριθμό επαναλήψεων είτε μετά από ένα συγκεκριμένο χρονικό διάστημα το οποίο έχουμε ορίσει εμείς από πριν.

**Algorithm 3:** Simulated Annealing

```
Select the best solution vector x_0 to be optimized
Initialize the parameters: temperature T, Boltzmann's constant k, reduction
 factor c
Data: T,k,c.
while Termination criteria is not satisfied do
    for Number of new solution do
        Select a new solution x_0+Δx
        if f(x_0 + Δx) > f(x_0) then
            f_new() = f(x_0 + Δx);  x_0 = x_0 + Δx
        end
        else
            Δf = f(x_0 + Δx) − f(x_0)
            random r(0,1)
        end
        if r > exp(−Δf/kT) then
            f_new() = f(x_0 + Δx),  x_0 = x_0 + Δx
        end
        else
            f_new = f(x_0)
        end
        f = f_new Decrease the temperature periodically: T = cxT
    end
end
```

Ένα από τα μεγαλύτερα προβλήματα που εντοπίζεται στους περισσότερους, κυρίως απλούς αλγόριθμους βελτιστοποίησης, είναι η παραμονή και ο τερματισμός τους σε τοπικά μέγιστα, καθώς εξετάζουν μόνο αν το γειτονικό αποτέλεσμα είναι καλύτερο από το παρόν και όχι κάποιο επόμενο από τα γειτονικά τους. Αντίθετα ο αλγόριθμος της Προσομοιωμένης Ανόπτησης περιέχει μηχανισμό ελέγχου διαδοχικών επαναλήψεων, με στόχο την εξέταση καλύτερων λύσεων και αποτελεσμάτων. Ο μηχανισμός αυτός πιο συγκεκριμένα επιτρέπει μετακινήσεις προς χειρότερες λύσεις, δίνοντας του την ελευθερία να μην στα-

ματήσει στην πρώτη καλύτερη λύση, αλλά να συνεχίσει να ψάχνει για κάποια καλύτερη αργότερα. Συμπερασματικά, αν ο αλγόριθμος βρίσκει συνεχώς χειρότερες λύσεις από την τρέχουσα, κάποια στιγμή η πιθανότητα να βρεθεί επόμενη χειρότερη λύσει θα είναι μηδενική, έτσι ο αλγόριθμος θα τερματίσει σε αυτό το σημείο πιθανός με μια βέλτιστη λύση. Τέλος, σε περίπτωση που βρίσκει συνεχώς καλύτερες λύσεις φθάνει σε ολικό μέγιστο [28].

Στον αλγόριθμο που ακολουθεί βλέπουμε μια απλή υλοποίηση του αλγορίθμου βελτιστοποίησης Προσομοιωμένης Ανόπτησης (Simulated Annealing), στην γλώσσα προγραμματισμού Python.

```
#finding a solution using simulated annealing algorithm
def simulated_annealing(initial_state):
    initial_temp = 90
    final_temp = .1
    alpha = 0.01
    current_temp = initial_temp

    #initializing the current state with the initial state
    current_state = initial_state
    solution = current_state

    while current_temp > final_temp:
        neighbor = random.choice(get_neighbors())
        # Check if neighbor solution is best so far
        cost_diff = get_cost(self.current_state) - get_cost
   (neighbor)
        # if the new solution is better, accept it
        if cost_diff > 0:
            solution = neighbor
        # if the new solution is not better, accept it with
    a probability of e^(-cost/temp)
        else:
            if random.uniform(0, 1) < math.exp(-cost_diff /
    current_temp):
                solution = neighbor
        # decrement the temperature
        current_temp -= alpha

    return solution
```

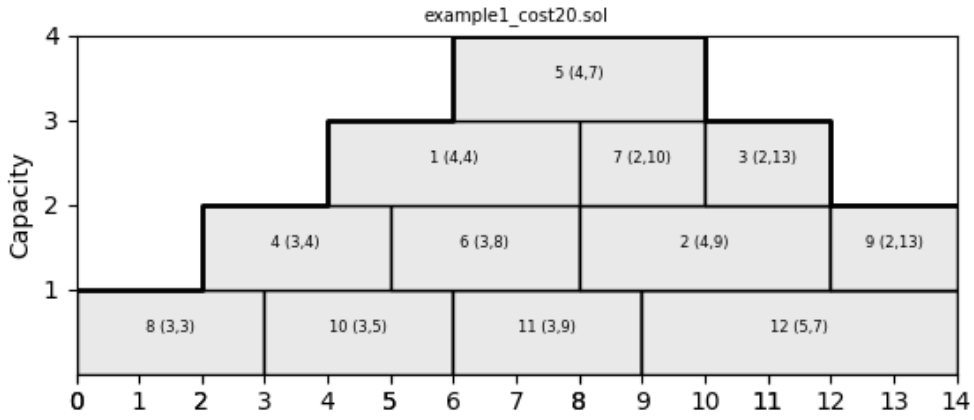
Listing 6.2: Simulated Annealing Algorithm

# Κεφάλαιο 7

# Επίλυση & Αποτελέσματα

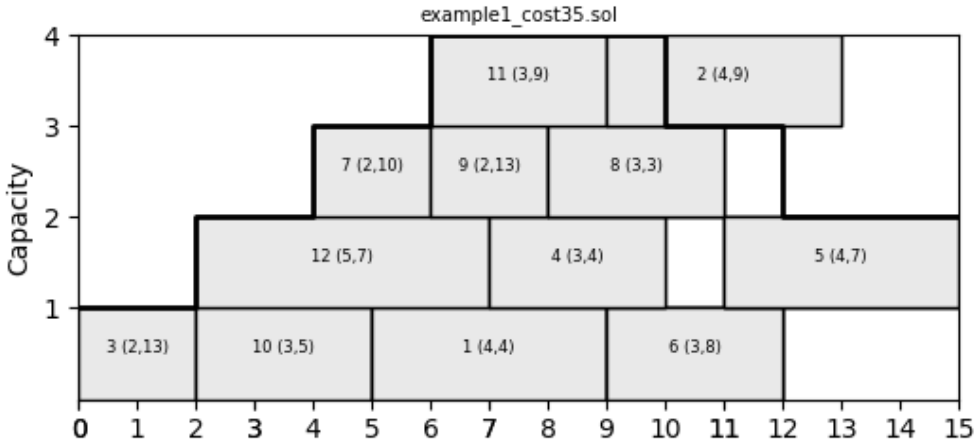
## 7.1 Επίλυση του προβλήματος

Τα προβλήματα σαν αυτό που καλούμαστε να επιλύσουμε δεν αποτελούνται από μια συγκεκριμένη λύση, ούτε επίσης πρέπει να αρκεστούμε σε μια δοκιμή με έναν συγκεκριμένο αλγόριθμο η τεχνική βελτιστοποίησης. Για την επίλυση τους λοιπόν είναι αναγκαίο το να πραγματοποιήσουμε πολλές δοκιμές. Σε κάθε δοκιμή μπορούν να αλλάξουν πολλοί παράμετροι. Άλλοτε ο αλγόριθμος που θα χρησιμοποιηθεί για την βελτιστοποίηση της λύσης, άλλοτε διαφορετικά αρχεία δεδομένων τα οποία θα αποτελούνται από λιγότερα η περισσότερα δεδομένα, έτσι ώστε ο χρήστης να συγκρίνει εάν ο συγκεκριμένος αλγόριθμος αποδίδει καλύτερα σε μεγάλο ή μικρό φόρτο εργασιών. Επίσης, όσον αφορά τα μεγάλα αρχεία δεδομένων συνδυαστικά με τα προβλήματα που αποτελούνται από πολλούς περιορισμούς, ο χρήστης πρέπει να διαθέτει καλό "hardware", κυρίως κάρτα γραφικών, επεξεργαστή και καλή ψύξη, εφόσον θα εκτελέσει πολύ απαιτητικές διεργασίες σε μεγάλο διάστημα χρόνου.



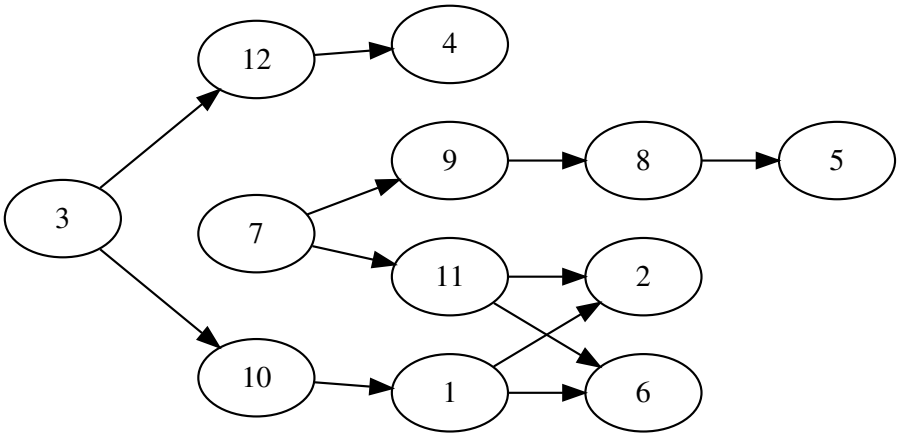

Σχήμα 7.1: Γραφικό αποτέλεσμα ταξινόμησης εργασιών στην μηχανή με κόστος ολικής καθυστέρησης 20 [4]

Η πρώτη διαδικασία που θα ακολουθήσουμε αποτελεί το να "διαβάσουμε" ένα από τα αρχεία δεδομένων τα οποία έχουμε στην διάθεση μας, όπως φαίνεται στον κώδικα που βρίσκεται στο Κεφάλαιο 4. Έπειτα με την χρήση του Schedule Builder ή Schedule Generation Schema που εξειδικεύεται στην αναζήτηση εντός κενού χώρου, τοποθετούμε




Σχήμα 7.2: Γραφικό αποτέλεσμα ταξινόμησης εργασιών στην μηχανή με κόστος ολικής καθυστέρησης 35 [4]

ακολουθιακά τις εργασίες στις γραμμές χωρητικότητας της μηχανής από δεξιά προς τα αριστερά με βάση τον χρόνο έναρξης τους, έτσι ώστε να μην ξεπεραστεί η χωρητικότητα της μηχανής και το πρόγραμμα μας να παραμείνει feasible, δηλαδή εφικτά επιλύσιμο. Σημαντικό είναι να αναφερθεί ότι κάθε φορά που μια νέα εργασία τοποθετείται σε μια θέση εντός της μηχανής πραγματοποιούνται έλεγχοι αναφορικά με την ακεραιότητα της κάθε κίνησης και αν η κίνηση αυτή βελτιώνει ή όχι το αποτέλεσμα.
Εφόσον έχουμε ταξινομήσει εφικτά τις εργασίες στην μηχανή με την χρήση του Schedule Builder χωρίζουμε τις γραμμές χωρητικότητας (C-paths), κυρίως για διαχειριστικούς λόγους. Κατά αυτόν τον τρόπο μπορούμε να κάνουμε αλλαγές ταξινόμησης των εργασιών εσωτερικά του κάθε C-path χωρίς να επηρεαστούν οι εργασίες εξωτερικά από αυτό.
Για παράδειγμα στο Σχήμα 7.2 με ολική καθυστέρηση 35, η μηχανή αποτελείται από έξι γραμμές χωρητικότητας. Η πρώτη γραμμή χωρητικότητας στο διάγραμμα του παραδείγματος αποτελείται από τις εργασίες (3,12,4), η δεύτερη από τις εργασίες (3,10,1,6), ή τρίτη από τις (3,10,1,2), η τέταρτη από τις (7,9,8,5), η πέμπτη από τις εργασίες (7,11,2) και η τελευταία από τις (7,11,6). Πιο αναλυτικά βλέπουμε το Σχήμα 7.3 όπου παριστάνονται γραφικά τα μονοπάτια χωρητικότητας του Σχήματος 7.2.




Σχήμα 7.3: Διάγραμμα προβολής των C-Paths [4]

Μπορεί ο Schedule Builder να κάνει μια πρώτη προσέγγιση στην λύση του προβλήματος, αλλά η προσέγγιση αυτή δεν είναι βέλτιστη ούτε και πάντοτε αρκετή. Γι' αυτό τον λόγο χρησιμοποιούμε αλγόριθμους βελτιστοποίησης ή επιλυτές όπως ο IBM ILOG CPLEX ή ο Google OR-Tools που, όπως προαναφέραμε, περιέχουν εμφωλευμένα μαθηματικά μοντέλα και άλλες μεθόδους βελτιστοποίησης. Συνεπώς με την επιπλέον βελτιστοποίηση των εργασιών έχουμε συνήθως πολύ καλύτερα αποτελέσματα αναζητώντας πάντοτε το βέλτιστο.

Το σημαντικότερο εργαλείο που θα χρησιμοποιήσουμε για την επίλυση του προβλήματος και την εύρεση λύσεων είναι ο IBM ILOG CP Optimizer που αποτελεί ένα σύστημα μοντελοποίησης και επίλυσης προβλημάτων χρονοπρογραμματισμού, χρησιμοποιώντας ως βασικό μηχανισμό του την μέθοδο του προγραμματισμού με περιορισμούς (Constraint Programming), έχοντας πάρει και το όνομα του από αυτή. Ο συγκεκριμένος επιλυτής CPLEX χρησιμοποιεί μαθηματικά μοντέλα της αλγεβρικής γλώσσας για τα προβλήματα που καλείται να επιλύσει, ώστε να αποτυπώσει την χρονική διάσταση των προβλημάτων χρονοπρογραμματισμού. Ο βασικός μηχανισμός του CP Optimizer είναι σχεδιασμένος να λειτουργεί με την τεχνική "model & run" έτσι ώστε ο χρήστης να μην χρειάζεται να αφοσιωθεί τόσο στην κατανόηση σε βάθος της μεθόδου του προγραμματισμού με περιορισμούς και των αλγορίθμων χρονοπρογραμματισμού, αλλά να επικεντρωθεί στην μοντελοποίηση [29]. Ο CP Optimizer είναι ευρέως γνωστός στον χώρο του χρονοπρογραμματισμού και έχει χρησιμοποιηθεί σε ποικίλα προβλήματα που καλύπτουν πολλούς τομείς του. Ο επιλυτής έχει εφαρμοστεί μέχρι στιγμής στους εξής τομείς:

- Στον χώρο της βιομηχανίας
- Στα αυτοματοποιημένα συστήματα
- Στα δίκτυα
- Από πολιτικούς μηχανικούς
- Στον χώρο των μεταφορών
- Σε αποστολές στο διάστημα
- Σε λιμένες

## 7.2 Αποτελέσματα

Το αρχείο δεδομένων που παρουσιάστηκε στο Κεφάλαιο 4 είναι κυρίως ενδεικτικό, αποτελούμενο από μικρό αριθμό εργασιών και μικρή χωρητικότητα μηχανής. Τα αρχεία που καλούμαστε να δοκιμάσουμε αποτελούνται από 120 έως και 1000 εργασίες που πρέπει να ταξινομηθούν χρονικά, και με σαφώς μεγαλύτερη χωρητικότητα μηχανής.
Για να έχουμε μια καλύτερη εικόνα των αποτελεσμάτων που θα πάρουμε από την επίλυση του κάθε αρχείου δεδομένων, θα συγκρίνουμε την κάθε λύση με την αντίστοιχη λύση του επιστημονικού άρθρου [3], το οποίο διαπραγματεύεται το ίδιο πρόβλημα με την χρήση διαφορετικών αλγορίθμων βελτιστοποίησης.
Στο συγκεκριμένο άρθρο οι συγγραφείς εμβαθύνουν στην βελτιστοποίηση των διεργασιών με βάση τις γραμμές χωρητικότητας της μηχανής δηλαδή των C-paths, τα οποία είχαμε αναλύσει στο Κεφάλαιο 3. Κάποιοι από τους αλγόριθμους που χρησιμοποιούνται στο επιστημονικό άρθρο [3] είναι οι εξής:

- SCP (Single C-path local search) : Τοπική αναζήτηση εντός του C-path
- SCP+: SCP με την χρήση του αλγόριθμου Hill Climbing
- iSCP: SCP με κάποιες βελτιώσεις
- CB (Cover-Based Local Search)
- CPO (IBM ILOG CP Optimizer), παρόμοια με εμάς
- MA_HYB (Hybrid Memetic Algorithm)

Στον πίνακα που ακολουθεί παριστάνονται όλες οι περιπτώσεις των δεδομένων που θα αναλύσουμε. Στο σύνολό τους τα αρχεία δεδομένων είναι 190 και χωρίζονται σε πέντε κατηγορίες με βάση τον αριθμό των εργασιών που περιέχουν και σε υποκατηγορίες με βάση την χωρητικότητα της μηχανής. Στην πρώτη στήλη εμφανίζονται αυτές οι πέντε κατηγορίες που αφορούν τον αριθμό των εργασιών (120,250,500,750,1000). Στην δεύτερη στήλη του πίνακα αναγράφονται οι χωρητικότητες της μηχανής για κάθε αρχείο δεδομένων ξεχωριστά.

| Number of jobs | Capacity |
|---|---|
| 120 | 3, 5, 7, 10 |
| 250 | 10, 20, 30 |
| 500 | 10, 20, 30 |
| 750 | 10, 20, 30, 50 |
| 1000 | 10, 20, 30, 50, 100 |

Πίνακας 7.1: Ο αριθμός των εργασίων για την κάθε περίπτωση του αρχειού δεδομένων και η μέγιστη χωρητικότητα μηχανής

Τα αναλυτικά αποτελέσματα μετά την βελτιστοποίηση των δεδομένων φαίνονται στους πίνακες δεδομένων που ακολουθούν. Οι πίνακες αυτοί αποτελούνται από τέσσερις στήλες. Η πρώτη στήλη αντιπροσωπεύει το όνομα του αρχείου δεδομένων που έχουμε χρησιμοποιήσει για την υλοποίηση του προβλήματος. Η δεύτερη και η τρίτη στήλη περιέχουν τα καλύτερα αποτελέσματα από το επιστημονικό άρθρο που προαναφέρθηκε και τα δικά μας πιο εύστοχα αποτελέσματα αντίστοιχα, βασισμένα στο ίδιο αρχείο δεδομένων. Τέλος, η τέταρτη στήλη αποτελεί την απόκλιση επί τοις εκατό που έχουν οι δυο στήλες των αποτελεσμάτων. Αν η τιμή της στήλης με την ονομασία “distance” είναι μηδέν, τότε τα δυο αποτελέσματα είναι ισάξια. Σε περίπτωση που η τιμή της “distance” είναι μικρότερη από το μηδέν, τότε τα δικά μας αποτελέσματα έχουν μικρότερο κόστος, ενώ αν είναι μεγαλύτερη του μηδενός το αντίθετο.

Πίνακας 7.2: Αποτελέσματα Μηχανής με Χωρητικότητα 120

| Dataset | Mencia_Best | Ours_Best | distance% |
|---|---|---|---|
| i120_3_1 | 848 | 848 | 0.000000 |
| i120_3_2 | 3568 | 3570 | 0.056054 |
| i120_3_3 | 2410 | 2410 | 0.000000 |
| i120_3_4 | 1019 | 1019 | 0.000000 |
| i120_3_5 | 3858 | 3858 | 0.000000 |
| i120_3_6 | 3120 | 3120 | 0.000000 |
| i120_3_7 | 720 | 720 | 0.000000 |
| i120_3_8 | 4084 | 4084 | 0.000000 |
| i120_3_9 | 1663 | 1663 | 0.000000 |
| i120_3_10 | 1268 | 1268 | 0.000000 |
| i120_5_1 | 1030 | 1030 | 0.000000 |
| i120_5_2 | 1511 | 1514 | 0.198544 |
| i120_5_3 | 959 | 959 | 0.000000 |
| i120_5_4 | 496 | 496 | 0.000000 |
| i120_5_5 | 3061 | 3061 | 0.000000 |
| i120_5_6 | 455 | 455 | 0.000000 |
| i120_5_7 | 519 | 519 | 0.000000 |
| i120_5_8 | 1214 | 1214 | 0.000000 |
| i120_5_9 | 3246 | 3231 | 0.248216 |
| i120_5_10 | 1131 | 1131 | 0.000000 |
| i120_7_1 | 1120 | 1121 | 0.089286 |
| i120_7_2 | 2725 | 2725 | 0.000000 |
| i120_7_3 | 3497 | 3505 | 0.343544 |
| i120_7_4 | 4028 | 4028 | 0.174086 |
| i120_7_5 | 3065 | 3078 | 0.621118 |
| i120_7_6 | 2640 | 2640 | 0.000000 |
| i120_7_7 | 1655 | 1655 | 0.000000 |
| i120_7_8 | 3225 | 3225 | 0.000000 |
| i120_7_9 | 4470 | 4474 | 0.089485 |
| i120_7_10 | 493 | 493 | 0.000000 |
| i120_10_1 | 749 | 749 | 0.402145 |
| i120_10_2 | 1125 | 1125 | 0.088968 |
| i120_10_3 | 1453 | 1453 | 0.762829 |
| i120_10_4 | 887 | 887 | 0.000000 |
| i120_10_5 | 1454 | 1450 | 0.207326 |
| i120_10_6 | 1118 | 1118 | 0.000000 |
| i120_10_7 | 2463 | 2463 | 0.000000 |
| i120_10_8 | 721 | 721 | 0.000000 |
| i120_10_9 | 984 | 983 | 0.614125 |
| i120_10_10 | 820 | 820 | 0.000000 |

Πίνακας 7.3: Αποτελέσματα Μηχανής με Χωρητικότητα 250

| Dataset | Mencia_Best | Ours_Best | distance% |
|---|---|---|---|
| i250_10_1 | 4098 | 4103 | 0.219834 |
| i250_10_2 | 506 | 506 | 0.000000 |
| i250_10_3 | 1349 | 1349 | 0.000000 |
| i250_10_4 | 4731 | 4731 | 0.000000 |
| i250_10_5 | 6442 | 6391 | 0.015649 |
| i250_10_6 | 6280 | 6284 | 0.063694 |
| i250_10_7 | 4503 | 4511 | 0.311319 |
| i250_10_8 | 5900 | 5887 | 0.102023 |
| i250_10_9 | 5324 | 5327 | 0.112761 |
| i250_10_10 | 1293 | 1293 | 0.000000 |
| i250_20_1 | 5573 | 5573 | 0.000000 |
| i250_20_2 | 1882 | 1889 | 0.371945 |
| i250_20_3 | 2813 | 2813 | 0.000000 |
| i250_20_4 | 2525 | 2525 | 0.000000 |
| i250_20_5 | 1054 | 1054 | 0.000000 |
| i250_20_6 | 1585 | 1606 | 1.452937 |
| i250_20_7 | 1567 | 1570 | 0.319489 |
| i250_20_8 | 2190 | 2190 | 0.000000 |
| i250_20_9 | 1553 | 1553 | 0.000000 |
| i250_20_10 | 6532 | 6541 | 0.153116 |
| i250_30_1 | 3013 | 3013 | 0.000000 |
| i250_30_2 | 3054 | 3054 | 0.000000 |
| i250_30_3 | 4757 | 4753 | -0.084087 |
| i250_30_4 | 4096 | 4093 | -0.073242 |
| i250_30_5 | 4194 | 4194 | 0.000000 |
| i250_30_6 | 5031 | 5019 | -0.238521 |
| i250_30_7 | 3641 | 3641 | 0.000000 |
| i250_30_8 | 686 | 686 | 0.000000 |
| i250_30_9 | 1502 | 1502 | 0.000000 |
| i250_30_10 | 5188 | 5193 | 0.096376 |

Πίνακας 7.4: Αποτελέσματα Μηχανής με Χωρητικότητα 500

| Dataset | Mencia_Best | Ours_Best | distance% |
|---|---|---|---|
| i500_10_1 | 4614 | 4614 | 0.000000 |
| i500_10_2 | 822 | 822 | 0.000000 |
| i500_10_3 | 951 | 953 | 0.210305 |
| i500_10_4 | 2105 | 2116 | 0.666032 |
| i500_10_5 | 610 | 610 | 0.000000 |
| i500_10_6 | 5981 | 5981 | 0.000000 |
| i500_10_7 | 2768 | 2783 | 0.541908 |
| i500_10_8 | 4460 | 4460 | 0.000000 |
| i500_10_9 | 462 | 462 | 0.000000 |
| i500_10_10 | 2003 | 2008 | 0.500501 |
| i500_20_1 | 7602 | 7569 | -0.434096 |
| i500_20_2 | 790 | 790 | 0.000000 |
| i500_20_3 | 8942 | 8970 | 0.324349 |
| i500_20_4 | 1272 | 1272 | 0.000000 |
| i500_20_5 | 1717 | 1744 | 1.572510 |
| i500_20_6 | 9099 | 9097 | -0.021980 |
| i500_20_7 | 523 | 523 | 0.000000 |
| i500_20_8 | 3180 | 3192 | 0.377358 |
| i500_20_9 | 6289 | 6338 | 0.779138 |
| i500_20_10 | 5618 | 5662 | 0.783197 |
| i500_30_1 | 2670 | 2670 | 0.000000 |
| i500_30_2 | 477 | 477 | 0.000000 |
| i500_30_3 | 6005 | 5974 | -0.016736 |
| i500_30_4 | 4307 | 4307 | 0.000000 |
| i500_30_5 | 5869 | 5873 | 0.068155 |
| i500_30_6 | 1618 | 1626 | 0.743494 |
| i500_30_7 | 3795 | 3795 | 0.000000 |
| i500_30_8 | 8895 | 8888 | -0.078696 |
| i500_30_9 | 862 | 862 | 0.000000 |
| i500_30_10 | 352 | 352 | 0.000000 |

Πίνακας 7.5: Αποτελέσματα Μηχανής με Χωρητικότητα 750

| Dataset | Mencia_Best | Ours_Best | distance% |
|---|---|---|---|
| i750_10_1 | 4337 | 4312 | -0.576435 |
| i750_10_2 | 7744 | 7744 | 0.000000 |
| i750_10_3 | 5819 | 5821 | 0.034370 |
| i750_10_4 | 5078 | 5082 | 0.078771 |
| i750_10_5 | 1520 | 1500 | -1.120633 |
| i750_10_6 | 7895 | 7895 | 0.000000 |
| i750_10_7 | 948 | 943 | -0.527426 |
| i750_10_8 | 7956 | 7962 | 0.075415 |
| i750_10_9 | 12779 | 12814 | 0.273887 |
| i750_10_10 | 3840 | 3840 | 0.000000 |
| i750_20_1 | 5874 | 5979 | 1.787538 |
| i750_20_2 | 700 | 700 | 0.000000 |
| i750_20_3 | 1314 | 1314 | 0.000000 |
| i750_20_4 | 9632 | 9562 | -0.726744 |
| i750_20_5 | 9073 | 9073 | 0.000000 |
| i750_20_6 | 11442 | 11434 | 0.000000 |
| i750_20_7 | 4855 | 4855 | 0.000000 |
| i750_20_8 | 11298 | 11284 | -0.123916 |
| i750_20_9 | 4393 | 4393 | 0.000000 |
| i750_20_10 | 2632 | 2632 | 0.000000 |
| i750_30_1 | 4707 | 4767 | 1.274697 |
| i750_30_2 | 5128 | 5128 | 0.000000 |
| i750_30_3 | 2365 | 2364 | 0.000000 |
| i750_30_4 | 2948 | 2945 | -0.101764 |
| i750_30_5 | 670 | 667 | 0.907716 |
| i750_30_6 | 2577 | 2577 | 0.000000 |
| i750_30_7 | 1070 | 1070 | 0.000000 |
| i750_30_8 | 2911 | 2912 | 0.034352 |
| i750_30_9 | 2700 | 2706 | 0.222222 |
| i750_30_10 | 1994 | 1994 | 0.000000 |
| i750_50_1 | 5134 | 5134 | 0.000000 |
| i750_50_2 | 4959 | 4792 | -3.367614 |
| i750_50_3 | 12572 | 12147 | -3.380528 |
| i750_50_4 | 2356 | 2356 | 0.000000 |
| i750_50_5 | 9840 | 9847 | 0.071138 |
| i750_50_6 | 11480 | 11500 | 0.174216 |
| i750_50_7 | 4871 | 4868 | 0.000000 |
| i750_50_8 | 10581 | 10583 | 0.028355 |
| i750_50_9 | 5154 | 5154 | 0.000000 |
| i750_50_10 | 5150 | 5028 | -2.368932 |

Πίνακας 7.6: Αποτελέσματα Μηχανής με Χωρητικότητα 1000

| Dataset | Mencia_Best | Ours_Best | distance% |
|---|---|---|---|
| i1000_10_1 | 641 | 641 | 0.000000 |
| i1000_10_2 | 23556 | 23521 | -0.148582 |
| i1000_10_3 | 814 | 812 | -0.245700 |
| i1000_10_4 | 15199 | 15211 | 0.078953 |
| i1000_10_5 | 15335 | 15180 | -1.010760 |
| i1000_10_6 | 2025 | 2025 | 0.000000 |
| i1000_10_7 | 732 | 729 | -0.409836 |
| i1000_10_8 | 20626 | 20574 | -0.252109 |
| i1000_10_9 | 22252 | 22162 | -0.404458 |
| i1000_10_10 | 3986 | 3982 | -0.100351 |
| i1000_20_1 | 9440 | 9440 | 0.000000 |
| i1000_20_2 | 16004 | 15971 | -0.206198 |
| i1000_20_3 | 20004 | 19986 | -0.089982 |
| i1000_20_4 | 7907 | 7975 | 0.859997 |
| i1000_20_5 | 14183 | 14183 | 0.000000 |
| i1000_20_6 | 24508 | 24507 | -0.004080 |
| i1000_20_7 | 9727 | 9726 | -0.010281 |
| i1000_20_8 | 16906 | 16789 | -0.692062 |
| i1000_20_9 | 10290 | 10301 | 0.106900 |
| i1000_20_10 | 10781 | 10781 | 0.000000 |
| i1000_30_1 | 6894 | 6780 | -1.653612 |
| i1000_30_2 | 1675 | 1668 | -0.417910 |
| i1000_30_3 | 18312 | 18087 | -1.228702 |
| i1000_30_4 | 12399 | 12411 | 0.096782 |
| i1000_30_5 | 3718 | 3707 | -0.295858 |
| i1000_30_6 | 12090 | 12090 | 0.000000 |
| i1000_30_7 | 9775 | 9765 | -0.102302 |
| i1000_30_8 | 2085 | 2085 | 0.000000 |
| i1000_30_9 | 15564 | 15528 | -0.231303 |
| i1000_30_10 | 18609 | 18483 | -0.677092 |
| i1000_50_1 | 2883 | 2883 | 0.000000 |
| i1000_50_2 | 16388 | 15977 | -2.507933 |
| i1000_50_3 | 11613 | 11618 | 0.043055 |
| i1000_50_4 | 1142 | 1142 | 0.000000 |
| i1000_50_5 | 1390 | 1390 | 0.000000 |
| i1000_50_6 | 16656 | 16667 | 0.066042 |
| i1000_50_7 | 19566 | 19566 | 0.000000 |
| i1000_50_8 | 1889 | 1889 | 0.159067 |
| i1000_50_9 | 13740 | 13747 | 0.050946 |
| i1000_50_10 | 3989 | 3989 | 0.000000 |
| i1000_100_1 | 71018 | 71012 | -0.008449 |
| i1000_100_2 | 75129 | 75181 | 0.106523 |
| i1000_100_3 | 25873 | 25594 | -1.078344 |
| i1000_100_4 | 36382 | 36376 | 0.005498 |
| i1000_100_5 | 67095 | 66419 | -1.007527 |
| i1000_100_6 | 47523 | 47541 | 0.037876 |
| i1000_100_7 | 44439 | 43437 | -2.254776 |
| i1000_100_8 | 52298 | 52268 | 0.003827 |
| i1000_100_9 | 23721 | 23690 | -0.059062 |
| i1000_100_10 | 15605 | 15230 | -2.403076 |

# Κεφάλαιο 8

# Συμπεράσματα & Μελλοντικές επεκτάσεις

## 8.1 Συμπεράσματα

Πλέον διαθέτοντας το σύνολο των αποτελεσμάτων που συλλέξαμε κατά την επίλυση του προβλήματος με την χρήση του επιλυτή IBM ILOG CP Optimizer και των αποτελεσμάτων που προέκυψαν από το επιστημονικό άρθρο [3], με την χρήση των ίδιων συνόλων δεδομένων, έχουμε την δυνατότητα να τα συγκρίνουμε και να βγάλουμε κάποια συμπεράσματα. Συνήθως σε προβλήματα ίδιας κατηγορίας με το πρόβλημα μας κατά την σύγκριση των αποτελεσμάτων μπορούμε να αντλήσουμε πολύ χρήσιμες πληροφορίες. Για παράδειγμα αν ο συγκεκριμένος αλγόριθμος βελτιστοποίησης που χρησιμοποιήσαμε διαχειρίζεται αποδοτικότερα, σύνολα με μεγαλύτερο όγκο δεδομένων ή σύνολα με μικρότερο όγκο δεδομένων, επίσης εάν δεν έχουμε θέσει κάποιο χρονικό όριο γίνεται εφικτή η σύγκριση της ταχύτητας των αλγορίθμων που θα χρησιμοποιηθούν πάνω στο ίδιο σύνολο δεδομένων.

Πίνακας 8.1: Πίνακας κατηγοριοποίησης σύνολων δεδομένων

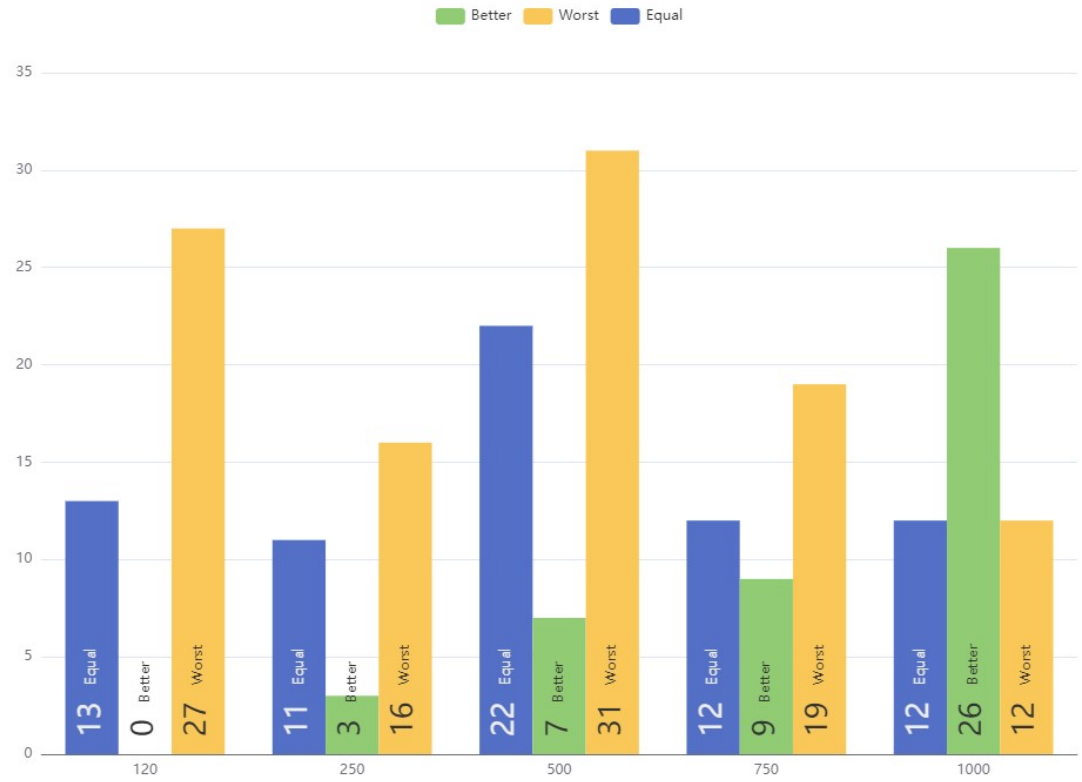
| Αριθμός Εργασιών | Σύνολα Δεδομένων |
|---|---|
| 120 | 40 |
| 250 | 30 |
| 500 | 30 |
| 750 | 40 |
| 1000 | 50 |
| Άθροισμα | 190 |

Στον Πίνακα 8.2 που ακολουθεί αλλά και στο παρακάτω σχήμα [8.1] βλέπουμε διαφορετικές εκδοχές αναπαράστασης των αποτελεσμάτων που πήραμε από τον IBM ILOG CP Optimizer συγκριτικά με τα αποτελέσματα του επιστημονικού άρθρου [3]. Στην πρώτη στήλη παριστάνεται το πλήθος των εργασιών που περιλαμβάνεται στα σύνολα δεδομένων. Έχουμε δηλαδή 5 κατηγορίες συνόλων (120,250,500,750,1000) χωρισμένες με βάση τον αριθμό εργασιών που περιέχουν, όπως φαίνεται στον Πίνακα 8.1. Επίσης στην δεύτερη στήλη παρατηρούμε το άθροισμα των περιπτώσεων που έχουμε αποτελέσματα καλύτερα από αυτά του επιστημονικού άρθρου [3], σε αντίθεση με την τρίτη στήλη που περιλαμβάνει τον αριθμό των χειρότερων αποτελεσμάτων. Στην τέταρτη στήλη αναγράφονται τα

δεδομένα που και στις δυο περιπτώσεις ήταν ίδια.

Πίνακας 8.2: Πίνακας συγκριτικών αποτελεσμάτων

| Αριθμός Εργασιών | Καλύτερα | Χειρότερα | Ίδια |
|---|---|---|---|
| 120 | 0/40 | 13/40 | 27/40 |
| 250 | 3/30 | 11/30 | 16/30 |
| 500 | 7/30 | 22/30 | 31/30 |
| 750 | 9/40 | 12/40 | 19/40 |
| 1000 | 26/50 | 12/50 | 12/50 |




Σχήμα 8.1: Ραβδόγραμμα σύγκρισης αποτελεσμάτων

Συμπερασματικά, όπως φαίνεται και στο σύνολο των αποτελεσμάτων που έχουμε στην κατοχή μας, η μέθοδος που χρησιμοποιήσαμε είναι λιγότερο αποδοτική σε μικρότερα σύνολα δεδομένων από τις μεθόδους αναζήτησης που χρησιμοποιήθηκαν από τους συγγραφείς του επιστημονικού άρθρου [3], αλλά είναι πολύ πιο αποδοτική σε μεγάλα σύνολα δεδομένων. Αυτό γίνεται αντιληπτό διότι στα σύνολα δεδομένων που αποτελούνται από 120 εργασίες το σκορ μας είναι 0/40 καλύτερα αποτελέσματα και 13/40 χειρότερα, ενώ στα σύνολα δεδομένων που αποτελούνται από 1000 εργασίες το σκορ που πετύχαμε είναι 26/50 καλύτερα και 12/50 χειρότερα.

## 8.2 Μελλοντικές επεκτάσεις

Όπως προαναφέρθηκε, σε προβλήματα παρόμοιας φύσης είναι πολύ υποκειμενικό το να έχει βρεθεί μια βέλτιστη λύση. Γενικότερα, δεν πρέπει να μένουμε στάσιμοι σε μια καλή λύση αλλά να δοκιμάζουμε νέους αλγόριθμους και τεχνικές. Έτσι λοιπόν, σε μελλοντικό στάδιο θα ήταν ιδανικό να δοκιμάσουμε και άλλες υλοποιήσεις, όπως:

- Με αλγόριθμους Τοπικής Αναζήτησης όπως ο αλγόριθμος Αναρρίχησης Λόφων Hill Climbing και ο αλγόριθμος Προσομοιωμένης Ανόπτησης Simulated Annealing που αναλύσαμε και παραπάνω.
- Με Γενετικούς αλγορίθμους.
- Με την χρήση του επιλυτή ανοιχτού κώδικα Google OR-Tools.
- Μιμητικούς, όπως στο επιστημονικό άρθρο [3].
- Με συνδυασμό αλγόριθμων Τοπικής Αναζήτησης και μεθόδων χρήσης των C-Paths, όπως ο SCP, ο SPC+ ή ο iSCP.

# Βιβλιογραφία


[1] "The scheduling zoo." [Online]. Available: http://schedulingzoo.lip6.fr/

[2] "Local search algorithms and optimization problem." [Online]. Available: https://www.tutorialandexample.com/local-search-algorithms-and-optimization-problem

[3] R. Mencía and C. Mencía, "One-machine scheduling with time-dependent capacity via efficient memetic algorithms." no. 23, Nov. 2021. [Online]. Available: https://www.mdpi.com/2227-7390/9/23/3030

[4] C. Valouxis, C. Gogos, A. Dimitsas, P. Potikas, and A. Vittas, "A hybrid exact–local search approach for one-machine scheduling with time-dependent capacity." no. 12, Nov. 2022. [Online]. Available: https://www.mdpi.com/1999-4893/15/12/450

[5] "Understanding hill climbing algorithm in artificial intelligence." [Online]. Available: https://www.section.io/engineering-education/understanding-hill-climbing-in-ai/

[6] M. Blocho, "Heuristics, metaheuristics, and hyperheuristics for rich vehicle routing problems." [Online]. Available: https://www.sciencedirect.com/topics/social-sciences/simulated-annealings

[7] L. Drougas, "Bachelor's thesis: Advertisement scheduling via evolutionary alorithms." Nov. 2009.

[8] K. Kydonieos, "Bachelor's thesis: Production scheduling in industry paper packaging using pso algorithm." Feb. 2015.

[9] S. Aaronson, "The scientific case for p $\neq$ *np.*"[*Online*].*Available* : *https* : *//scottaaronson.blog/?p* = 1720

[10] "Np-hard." [Online]. Available: https://xlinux.nist.gov/dads/HTML/nphard.html

[11] "Nondeterministic turing machine." [Online]. Available: https://encyclopediaofmath.org/wiki/Nondeterministic_Turing_machine

[12] V. Nastos, "Uncapacitated examination timetabling," p. 68, Oct. 2021. [Online]. Available: https://apothetirio.lib.uoi.gr/xmlui/handle/123456789/13105

[13] A. Sanjeev and B. Boaz, *Computational Complexity: A Modern Approach*. Cambridge University Press, no. 1. [Online]. Available: https://theory.cs.princeton.edu/complexity/book.pdf

[14] "Computers and intractability: a guide to the theory of np-completeness," no. 27. [Online]. Available: https://bohr.wlu.ca/hfan/cp412/references/ChapterOne.pdf

[15] C. Bo, C. Potts, and G. Woeginger, "Handbook of combinatorial optimization," pp. 1493–1641, 1998. [Online]. Available: https://link.springer.com/chapter/10.1007/978-1-4613-0303-9_25

[16] F. Hsiao-Lan, "Genetic algorithms in timetabling and scheduling," 1995. [Online]. Available: https://era.ed.ac.uk/handle/1842/30185

[17] O. Naud, J. Taylor, L. Colizzi, R. Giroudeau, S. Guillaume, E. Bourreau, T. Crestey, and B. Tisseyre, *Agricultural Internet of Things and Decision Support for Precision Smart Farming*, 2020, no. 1. [Online]. Available: https://www.sciencedirect.com/book/9780128183731/agricultural-internet-of-things-and-decision-support-for-precision-smart-farmingbook-info

[18] J. Hooker, "Logic, optimization, and constraint programming," Jan. 2002. [Online]. Available: http://pubsonline.informs.org/doi/abs/10.1287/ijoc.14.4.295.2828

[19] I. Sakellariou, N. Vasiliadis, P. Kefalas, and D. Stamatis, "Constraint logic programming." [Online]. Available: http://repository.kallipos.gr/handle/11419/789

[20] A. Gandomi, X.-S. Yang, S. Talatahari, and A. Alavi, "Metaheuristic algorithms in modeling and optimization," no. 1, Dec. 2013. [Online]. Available: https://www.researchgate.net/publication/259603698_Metaheuristic_Algorithms_in_Modeling_and_Optimization

[21] D. Papadakis, "Parallel tabu search algorithms for the flow-shop problem," no. 1, Dec. 1995. [Online]. Available: https://elocus.lib.uoc.gr/dlib/9/3/b/metadata-dlib-1995papadakis.tkl

[22] P. Meseguer, F. Rossi, and T. Schiex, "Soft constraints," 2008. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S1574652606800131

[23] C. Mencıa, M. R. Sierra, R. Mencıa, and R. Varela, *Genetic Algorithm for Scheduling Charging Times of Electric Vehicles Subject to Time Dependent Power Availability.* University of Oviedo, Campus of Gijon, Spain: Cambridge University Press, 2017, no. 1. [Online]. Available: https://bit.ly/3OVddYu

[24] R. Mencía and M. Carlos, "One-machine scheduling with time-dependent capacity via efficient memetic algorithms datasets on github." [Online]. Available: https://github.com/raulmencia/One-Machine-Scheduling-with-Time-Dependent-Capacity-via-Efficient-Memetic-Algorithms

[25] "Collections container datatypes, (Python Documentation)." [Online]. Available: https://docs.python.org/3/library/collections.html

[26] "Overview of mathematical programming, (IBM Decision Optimization CPLEX Modeling for Python documentation)." [Online]. Available: http://ibmdecisionoptimization.github.io/docplex-doc/mp.html

[27] "About or-tools." [Online]. Available: https://developers.google.com/optimization

[28] L. Kampas and N. Karikas, "Study, evaluation and implementation of heuristic and meta-heuristic algorithmus to optimize the time of the traffic lights," no. 1, Nov. 2017. [Online]. Available: http://okeanis.lib.puas.gr/xmlui/handle/123456789/3941

[29] P. Laborie, J. Rogerie, P. Shaw, and P. Vilím, "IBM ILOG CP optimizer for scheduling," no. 1, 2018. [Online]. Available: https://link.springer.com/article/10.1007/s10601-018-9281-x