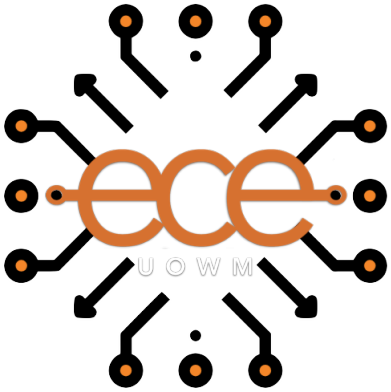


Πανεπιστήμιο Δυτικής Μακεδονίας

Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

Αλγόριθμοι για την επίλυση προβλημάτων χρονοδρομολόγησης

Κωνσταντίνος Λούγαρης (ΑΜ: 1500)

Επιβλέπων Καθηγητής: Νικόλαος Πλόσκας

Εργαστήριο Ευφυών Συστημάτων & Βελτιστοποίησης

28 Φεβρουαρίου 2024

Περίληψη

Η παρούσα διπλωματική εργασία μελετά και παρουσιάζει το πρόβλημα χρονοδρομολόγησης εργασιών (Job Shop Scheduling Problem - JSSP) και τον τρόπο επίλυσης του. Πρόκειται για ένα NP-Hard πρόβλημα συνδυαστικής βελτιστοποίησης, που βρίσκει εφαρμογή τόσο στον ερευνητικό όσο και στον επιχειρηματικό τομέα. Το πρόβλημα εμφανίζεται με διάφορες μορφές και περιορισμούς, ωστόσο στη συγκεκριμένη εργασία, ασχολούμαστε με την επίλυση του κλασικού προβλήματος χρονοδρομολόγησης εργασιών. Στα χρόνια που γίνονται έρευνες πάνω στο JSSP, έχουν προταθεί πολλές διαφορετικές τεχνικές για την επιλυσή του. Στην παρούσα εργασία παρουσιάζονται τέσσερις αλγόριθμοι, ένας ακριβής και τρείς ευρετικοί, που συνδυάζουν διάφορες μεθόδους για να λύσουν το πρόβλημα. Στόχος αυτής της εργασίας, είναι η ανάλυση των αποτελεσμάτων αυτών των αλγορίθμων καθώς επίσης και η σύγκρισή τους, με βάση την ποιότητα των αποτελεσμάτων αλλά και των χρόνων εκτέλεσης, ώστε να συμπεράνουμε ποιά είναι πιο αποδοτική.

Λέξεις κλειδιά: Πρόβλημα χρονοδρομολόγησης εργασιών, Ακριβείς αλγόριθμοι, Ευρετικοί αλγόριθμοι, Βελτιστοποίηση.

Abstract

This thesis studies and presents the Job Shop Scheduling Problem (JSSP) and its solution. It is an NP-Hard combinatorial optimization problem, which finds application in both research and business domains. The problem appears in various forms and constraints, however, in this paper, we are concerned with solving the classical job scheduling problem. In the years of research on the JSSP, many different techniques have been proposed to solve it. In this paper, we present four algorithms, one exact and three heuristics, that combine different methods to solve the problem. The aim of this paper, is to analyze the results of these algorithms and also to compare them, based on the quality of the results and the execution times, in order to conclude which one is more efficient.

Keywords: Job shop scheduling problem, Exact algorithms, Heuristic algorithms, Optimization.

Δήλωση Πνευματικών Δικαιωμάτων

Δήλωση Πνευματικών Δικαιωμάτων Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν. 1599/1986 και τα άρθρα 2,4,6 παρ. 3 του Ν. 1256/1982, η παρούσα Διπλωματική Εργασία με τίτλο "Αλγόριθμοι για την επίλυση προβλημάτων χρονοδρομολόγησης" καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας και αναφέρονται ρητώς μέσα στο κείμενο που συνοδεύουν, και η οποία έχει εκπονηθεί στο Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών του Πανεπιστημίου Δυτικής Μακεδονίας, υπό την επίβλεψη του μέλους του Τμήματος κ. Νικολάου Πλόσκα αποτελεί αποκλειστικά προϊόν προσωπικής εργασίας και δεν προσβάλλει κάθε μορφής πνευματικά δικαιώματα τρίτων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή / και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα. Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και μόνο.

Υπογραφή Φοιτητή

Περιεχόμενα

Κατάλογος σχημάτων

Κατάλογος αλγορίθμων

Κατάλογος πινάκων

Κεφάλαιο 1

Εισαγωγή

1.1 Ορισμός του προβλήματος

Το Πρόβλημα Χρονοδρομολόγησης Εργασιών (Job Shop Scheduling Problem, JSSP), είναι ένα από τα σημαντικότερα προβλήματα χρονοδρομολόγησης, αφορά τον καθορισμό της βέλτιστης σειράς και χρονικής διάταξης των εργασιών ή των εργασιών που πρέπει να εκτελεστούν σε διάφορα μηχανήματα ή πόρους, υπό συγκεκριμένους περιορισμούς και στόχους. Η πολυπλοκότητα του JSSP πηγάζει από τις αλληλεξαρτήσεις μεταξύ των εργασιών και των μηχανημάτων, καθώς και από περιορισμούς, όπως η διαθεσιμότητα πόρων, οι προθεσμίες και οι προτεραιότητες των εργασιών [1]. Το πρόβλημα του Job Shop Scheduling βρίσκει εφαρμογή σε πολλούς τομείς και βιομηχανίες όπου υπάρχει ανάγκη για αποδοτικό προγραμματισμό και κατανομή πόρων για την παραγωγή αγαθών ή υπηρεσιών. Μερικοί από τους τομείς όπου μπορεί να εφαρμοστεί το JSSP στον πραγματικό κόσμο περιλαμβάνουν τη βιομηχανία [2], τον σχεδιασμό παραγωγής [3], τον προγραμματισμό εργασιών [4], τη διαχείριση λειτουργιών [5] και τη διαχείριση έργων [6]. Συνολικά, το πρόβλημα του Job Shop Scheduling έχει ευρεία εφαρμογή σε διάφορους τομείς των βιομηχανιών και κλάδων, όπου υπάρχει ανάγκη για τη βελτιστοποίηση της χρήσης των πόρων, την ελαχιστοποίηση των χρόνων παραγωγής και τη βελτίωση της λειτουργικής αποτελεσματικότητας.

1.2 Κίνητρα και στόχοι υλοποίησης

Το Job Shop Scheduling Problem (JSSP) παρέχει ένα σημαντικό κίνητρο για τη συγγραφή της παρούσας διπλωματικής εργασίας λόγω της συνθετότητάς του, αλλά

και της ευρείας εφαρμογής του σε πολλούς τομείς. Από την άποψη της έρευνας, το JSSP αντιπροσωπεύει ένα αξιόλογο πρόβλημα για την ανάπτυξη και την αξιολόγηση νέων αλγορίθμων και μεθόδων επίλυσης στον τομέα της συνδυαστικής βελτιστοποίησης. Επιπλέον, η επίλυση του έχει σημαντικές εφαρμογές στη βιομηχανία και την πραγματική ζωή, όπου η βελτιστοποίηση του προγραμματισμού των εργασιών μπορεί να οδηγήσει σε βελτίωση της απόδοσης, μείωση των κόστων και αύξηση της αποτελεσματικότητας των διαδικασιών παραγωγής. Επομένως, η μελέτη του JSSP παρέχει ένα ευρύ φάσμα ευκαιριών για έρευνα και εφαρμογή.

Ο στόχος της παρούσας εργασίας είναι η κατασκευή αλγορίθμων που υλοποιούν διάφορες τεχνικές για να βελτιστοποιήσουν το πρόβλημα χρονοδρομολόγησης εργασιών, δηλαδή τη δρομολόγηση όλων των εργασιών στις μηχανές στο βέλτιστο χρόνο. Στόχος στην ουσία είναι να δημιουργήσουμε τα μικρότερα δυνατά ''δρομολόγια'', για διαφορετικά μεγέθη προβλήματων αναφοράς. Από μικρό αριθμό εργασιών-μηχανών έως και μεγάλο αριθμό εργασιών-μηχανών, μάλιστα ο αριθμός των διεργασιών για ορισμένα προβλήματα φτάνει τις δύο χιλιάδες. Για την επίτευξη του στόχου μας, λοιπόν θα μελετήσουμε δύο κατηγορίες αλγορίθμων. Η μια κατηγορία αφορά τους ακριβείς αλγορίθμους και συγκεκριμένα τη μέθοδο διακλάδωσης και δέσμευσης (branch and bound) που στοχεύει στην εύρεση της βέλτιστης λύσης μέσα από συνεχείς επαναλήψεις. Η δεύτερη κατηγορία, αφορά τους ευρετικούς αλγορίθμους που χρησιμοποιούν δίαφορες τεχνικές για να πλησιάσουν τη βέλτιστη λύση ή ακόμα και τη βρούν. Ειδικά για αυτή την κατηγορία θα παρουσιάσουμε τρείς μεθόδους: τη βελτιστοποίηση με προσομοιωμένο ανορθωτή, τη βελτιστοποίηση με σμήνος σωματιδίων και τη βελτιστοποίηση με αποικία μυρμηγκιών. Επιπλεόν, στόχος μας είναι να συγκρίνουμε τις παραπάνω τεχνικές ως προς τη ποιότητα της λύσης, δηλαδή την απόσταση από τη βέλτιστη, αλλά και ως προς τον χρόνο επίλυσης των προβλημάτων για να καταλήξουμε στο ποια είναι η καλύτερη.

1.3 Διάρθρωση κειμένου

Στην παρούσα διπλωματική εργασία θα εστιάσουμε στο κλασσικό πρόβλημα χρονοδρομολόγησης εργασίων (JSSP). Στο κεφάλαιο 2 θα παρουσιάσουμε και θα αναλύσουμε τις παραλλαγές του προβλήματος [7], τις διάφορες οντότητες του προβλήματος, τους περιορισμούς, τους στόχους προς επίτευξη αλλά και δύο τρόπους

αναπαράστασης του προβλήματος. Στο κεφάλαιο 3 θα αναφέρουμε πληθώρα βιβλιογραφικών πηγών που επιλύουν το πρόβλημα, ξεκινώντας από τις τεχνικές στα πρώτα χρόνια μελέτης του προβλήματος και συνεχίζοντας με πιο εξελιγμένες μεθόδους μέχρι σήμερα. Στο κεφάλαιο 4, θα παρουσιάσουμε τις δικές μας υλοποιήσεις για να λύσουμε το JSSP. Θα γίνει μια μικρή αναφορά στη γενική λειτουργία τους και έπειτα θα αναλύσουμε κάθε μέθοδο παρουσιάζοντας τα σημαντικότερα σημεία των κωδίκων μας. Στο κεφάλαιο 5, θα παρουσιάσουμε τα αποτελέσματα που δώσανε οι αλγόριθμοι μας και στη συνέχεια, αφού τα αναλύσουμε θα προχωρήσουμε στη σύγκριση των αλγορίθμων βάσει των αποτελεσμάτων που πήραμε τόσο στη λύση όσο και στο χρόνο. Μάλιστα, η παρουσίαση των αποτελεσμάτων θα φανεί μέσα από κατάλληλους πίνακες. Τέλος, στο κεφάλαιο 6 θα αναφέρουμε τα συμπεράσματα μας από τη σύγκριση των αλγορίθμων αλλά και τρόπους με τους οποίους μελλοντικά μπορούμε να βελτιώσουμε τη λειτουργία και την απόδοση των αλγορίθμων που υλοποιήσαμε στην παρούσα εργασία.

Κεφάλαιο 2

Παραλλαγές του προβλήματος χρονοδρομολόγησης

Τα προβλήματα χρονοδρομολόγησης εργασιών είναι από τα πιο σημαντικά προβλήματα βελτιστοποίησης. Πρόκειται για προβλήματα που απαιτούν την ολοκλήρωση συγκεκριμένων εργασίων με περιορισμένους πόρους για χρήση. Το Job-Shop Scheduling Problem (JSSP) ανήκει σε αυτή την κατηγορία προβλημάτων. Θεωρείται από τα σημαντικότερα και δυσκολότερα προβλήματα που έχουν απασχολήσει τους ερευνητές. Έχει γίνει γνωστό ως NP-hard πρόβλημα [8] κάτι το οποίο συνάδει με το γεγονός ότι στα αρχικά στάδια μελέτης και κατανόησης του παρέμεινε άλυτο για 15 χρόνια [9]. Μέχρι και σήμερα για πολύ μεγάλα προβλήματα δεν έχει υπάρξει επιβεβαιωμένη λύση.

2.1 Ορισμός του job shop scheduling problem

Ένα Job-Shop Scheduling problem αποτελείται από ένα σύνολο N εργασιών (jobs) που πρέπει επεξεργαστούν από ένα σύνολο M μηχανών (machines). Επιπλέον, κάθε εργασία φέρει μια σειρά από διεργασίες O (operations) που πρέπει να δρομολογηθούν με τη σειρά. Στόχος είναι να ελαχιστοποιηθεί ο χρόνος που χρειάζεται για να ολοκληρωθούν όλες οι εργασίες στο πλαίσιο ορισμένων περιορισμών.

Θα ήταν θεμιτό να εξηγήσουμε περαιτέρω τις παραπάνω οντότητες καθώς στην κάθε μια από αυτές αντιστοιχούν επιπλέον χαρακτηριστικά, όπως παρουσιάζονται στον Πίνακα 2.1. Σε μια εργασία j αντιστοιχεί ένας χρόνος αποδέσμευσης (r_j), ο χρόνος πρόωρης λήξης της εργασίας (d_j), το βάρος της εργασίας (w_j). Για τις μηχανές μπορούμε να ξεχωρίσουμε τη λειτουργικότητά τους, τη διαθεσιμότητα και

τη σύνδεση τους. Τέλος, τα χαρακτηριστικά της σχέσης εργασιών-μηχανών έχουν να κάνουν με το είδος επεξεργασίας, τις προτιμήσεις, την επανάληψη στις εισόδους και τη διακοπή των διεργασιών.

Πίνακας 2.1: Βασικά χαρακτηριστικά οντοτήτων ενός JSSP

Οντότητα	**Χαρακτηριστικό**	**Τιμές, Περιπτώσεις**
Εργασία	Χρόνος Αποδέσμευσης r_j	Μπορεί να είναι μηδέν ή να θεωρηθεί αυθαίρετος. Η κατανομή του μπορεί να γίνει ομοιόμορφα ή συμμετρικά (μέθοδος Gauss).
	Χρόνος Πρόωρης Λήξης d_j	Μπορεί να δοθεί αυθέραιτα, με κανόνες ή να μην υπάρχει καθόλου.
	Χρόνος Επεξεργασίας p_j	Παίρνει ντετερμινιστικές, ασαφείς, τυχαίες και ελεγχόμενες τιμές.
	Βάρος w_j	Η τιμή του είτε θα δίνεται είτε δεν θα λαμβάνεται υπόψην.
Μηχανή	Λειτουργικότητα	Θα δίνονται μηχανές μονής λειτουργίας είτε πολλαπλών λειτουργιών.
	Διαθεσιμότητα	Περίπτωση, όπου οι μηχανές θα είναι πάντα διαθέσιμες, περίπτωση που θα υπάρχει απρόβλεπτη αποτυχία ή συντήρηση και περίπτωση προγραμματισμένης επισκευής
	Σύνδεση	Ομοιόμορφη παράλληλη σύνδεση μηχανών, μονή-σε σειρά σύνδεση μηχανών.
Συνέχεια στην επόμενη σελίδα		

Πίνακας 2.1 – Συνέχεια από την προηγούμενη σελίδα

Οντότητα	**Χαρακτηριστικό**	**Τιμές, Περιπτώσεις**
Εργασία-Μηχανή	Είδος επεξεργασίας	Πλήρης, μερική ή καθόλου ευελιξία κατά τη δρομολόγηση. Είτε χρήση μονής μηχανής είτε χρήση ομάδας μηχανών.
	Προτίμηση	Με ή χωρίς προτίμηση στη δρομολόγηση.
	Επανάληψη είσοδου	Με επιτρεπόμενη επανάληψη ή χωρίς επανάληψη εισόδου διεργασιών.
	Διακοπή	Είτε επιτρέπεται η διακοπή των διεργασιών από άλλες είτε απαγορεύεται.

Με βάση τα παραπάνω χαρακτηριστικά μπορούμε να κάνουμε ορισμένες υποθέσεις και περιορισμούς για το πρόβλημα JSSP. Οι βασικότερες αυτών φαίνονται στον Πίνακα 2.2.

Πίνακας 2.2: Περιορισμοί προβλήματος

ΠΥ1.	Όλες οι εργασίες τη στιγμή 0 είναι έτοιμες για επεξεργασία, ο αριθμός τους είναι γνωστός και σταθερός. Ακόμη απαγορεύεται να υπάρξει διακοπή κάποιας εργασίας ενώ επεξεργάζονται.
ΠΥ3.	Όλες οι εργασίες έχουν το ίδιο κόστος επεξεργασίας.
ΠΥ4.	Δεν υπάρχει χρονική προθεσμία λήξης των εργασιών.
ΠΥ5.	Δεν υπάρχει αυθαίρετη προτεραιότητα μεταξύ των εργασιών.
ΠΥ6.	Κάθε διεργασία έχει μοναδικό και προκαθορισμένο χρόνο επεξεργασίας.
ΠΥ7.	Κάθε εργασία καθορίζεται προηγουμένως με μια μοναδική και σταθερή δρομολόγηση επεξεργασίας.
ΠΥ8.	Οι εργασίες είναι ανεξάρτητες μεταξύ τους.

ΠΥ9.	Κάθε διεργασία μπορεί εκ των προτέρων να ανατεθεί μόνο σε μια μηχανή για επεξεργασία. Δεν λαμβάνεται υπόψη η ευελιξία της διαδρομής.
ΠΥ10.	Κάθε διεργασία μπορεί να επεξεργαστεί από μια μόνο μηχανή.
ΠΥ11.	Κάθε μηχανή μπορεί να επεξεργαστεί μια διεργασία σε οποιαδήποτε εργασία.
ΠΥ12.	Επιτρέπεται η αναμονή της επεξεργασίας διεργασιών στην ίδια εργασία.
ΠΥ13.	Οι μηχανές είναι ανεξάρτητες η μια από την άλλη.
ΠΥ14.	Κατά την έναρξη μιας διεργασίας αυτή πρέπει να ολοκληρωθεί χωρίς να υπάρξει διακοπή.
ΠΥ15.	Κατά την έναρξη μιας διεργασίας αυτή πρέπει να ολοκληρωθεί χωρίς προτίμηση.
ΠΥ16.	Μια εργασία δεν μπορεί να χρησιμοποιήσει την ίδια μηχανή παραπάνω από μια φορά.
ΠΥ17.	Οι χρόνοι εγκατάστασης και μετάβασης είναι αμελητέοι.
ΠΥ18.	Όλες οι μηχανές είναι συνέχεια διαθέσιμες. Δεν λαμβάνονται υπόψη οι βλάβες και η συντήρηση των μηχανών.
ΠΥ19.	Δεν λαμβάνονται υπόψη για επανεπεξεργασία οι αποκλεισμένες και επείγουσες εργασίες.

2.2 Κατηγορίες προβλημάτων JSSP

Με βάση λοιπόν όλους αυτούς τους περιορισμούς και τις υποθέσεις μπορούμε να έχουμε διάφορες παραλλαγές του προβλήματος. Το κλασικό JSSP, όπως αναφέρθηκε, φέρει όλους τους περιορισμούς.

2.2.1 Dynamic job shop sceduling problem

Το Dynamic JSSP (DJSSP) χαλαρώνει μια ή περισσότερες παραδοχές των ΠΥ1, ΠΥ4, ΠΥ6 και ΠΥ19. Μάλιστα, ανάλογα τη χαλάρωση μπορούμε να διακρίνουμε και επιπλέον υποκατηγορίες της παραλλαγής αυτής.

- Υποκατηγορίες, που σχετίζονται με τις χρονικές ιδιότητες των εργασίων, στις οποίες παρουσιάζονται χαλαρώσεις στους περιορισμούς ΠΥ4 και ΠΥ6.
- Υποκατηγορίε,ς που αφορούν τον αριθμό των εργασιών, γίνεται χαλάρωση στους περιορισμούς ΠΥ1 και ΠΥ19. Αυτό σημαίνει πως επιτρέπεται η συνεχόμενη άφιξη εργασιών και η εκτέλεση έκτακτων εργασιών.
- Υποκατηγορίες, που λαμβάνεται υπόψη και επεξεργασία εργασιών που έχουν αποκλειστεί (ΠΥ19).

2.2.2 Κατάσταση μηχανής (machine state)

Το JSSP, όπου υπολογίζεται η κατάσταση των μηχανών. Σε αυτή την παραλλαγή του προβλήματος οι παράγοντες της συντήρησης των μηχανών ή των βλαβών δεν θεωρούνται αμεαλητέοι (ΠΥ18). Η περίπτωση, στην οποία οι μηχανές δεν είναι διαθέσιμες, μπορεί να είναι είτε περιοδική είτε να συμβαίνει υπό προϋποθέσεις. Τέλος, η κατάσταση των μηχανών είναι γνωστή εξαρχής.

2.2.3 Flexible job shop scheduling problem

Το Flexible JSSP (FJSSP), όπου διακρίνουμε δύο υποκατηγορίες του προβλήματος ανάλογα τις χαλαρώσεις στους περιορισμούς.

- με εναλλακτικές διαδρομές. Οι διεργασίες μιας εργασίας επιτρέπεται να επεξεργαστούν με οποιαδήποτε τεχνολογική ακολουθία (ΠΥ7).
- με εναλλακτικές μηχανές. Οι διεργασίες μιας εργασίας επιτρέπεται να επεξεργαστούν είτε από μια ομάδα μηχανών είτε και από όλες τις μηχανές (ΠΥ9).

2.2.4 Χρόνοι προετοιμασίας (setup times)

Το JSSP με χρόνους προετοιμασίας, στο οποίο υπάρχει χαλάρωση του κανόνα ΠΥ17 και συνήθως χαρακτηρίζεται από το JSSP με ανεξάρτητη ακολουθία και το JSSP με εξαρτημένη ακολουθία των χρόνων προετοιμασίας. Επιπλέον, το τελευταίο αναλογίζεται τη ρητή θεώρηση των χρονών προετοιμασίας και απομάκρυνσης μιας διεργασίας.

2.2.5 Παρτίδες / batch

Το JSSP που λαμβάνει υπόψη παρτίδες και μπορούμε να ξεχωρίσουμε δύο κατηγορίες.

- JSSP με παράλληλη παρτίδα, όπου μια ομάδα των μηχανών επιτρέπουν την πολλαπλή επεξεργασία διεργασιών σε αυτές ταυτόχρονα (ΠΥ11).
- JSSP με απόφαση προγραμματισμού της παρτίδας στην οποία υπολογίζεται η απομάκρυνση ή παράδοση της παρτίδας

2.2.6 Μη-ντετερμινιστικός χρόνος επεξεργασίας

Το JSSP με μη-ντετερμινιστικούς χρόνους επεξεργασίας. Σε αυτή την περίπτωση γίνεται χαλάρωση στον κανόνα ΠΥ6 δηλαδή, οι χρόνοι επεξεργασίας μπορεί να είναι είτε ελεγχόμενοι είτε ασαφείς είτε η αρχικοποίηση τους να είναι τυχαία. Σε αυτή την παραλλαγή ανήκει και το JSSP με εργασίες εξαρτώμενες από το χρόνο έναρξης.

2.3 Στόχος του προβλήματος (objective)

Υπάρχουν πολλοί στόχοι για επίτευξη που προκύπτουν από τα διάφορα μοντέλα και τις διαφορετικές κατηγορίες τέτοιων προβλημάτων. Οι Lawler et al. [10] αναφέρουν τέσσερα κριτήρια που στο βιβλιογραφικό χώρο, είναι τα βασικότερα για βελτιστοποίηση.

2.3.1 Ελαχιστοποίηση συνολικού χρόνου επεξεργασίας (Makespan)

Όλα τα χρόνια της μελέτης του προβλήματος JSSP, ο στόχος προς επίτευξη (objective) είναι να παραχθεί ένα πρόγραμμα, στο οποίο ο συνολικός χρόνος που απαιτείται ώστε να ολοκληρωθούν όλες οι εργασίες, ελαχιστοποιείται. Το μοντέλο που χρησιμοποιείται είναι το εξής: Έστω το πρόβλημα JSSP όπου J_n είναι ο αριθμός των εργασιών $(n = 1, 2, ..)$, M_m είναι ο αριθμός των μηχανών $(m = 1, 2, ..)$ και O_{ij}

είναι οι διεργασίες μια εργασίας.

$$\begin{aligned} t_{ik} - t_{ij} &\geq p_{ij}, & \forall[(i,j) \rightarrow (i,k)]\epsilon A \\ C_{max} - t_{ij} &\geq p_{ij}, & \forall(i,j)\epsilon N \\ t_{ij} - t_{lj} \geq p_{ij} \vee t_{lj} - t_{ij} &\geq p_{ij}, & \forall(l,j) and (i,j) = 1, ..., m \\ t_{i_j} &\geq 0, & \forall(i,j)\epsilon N \end{aligned} \tag{2.1}$$

Όπου C_{max} είναι ο συνολικός χρόνος που χρείαζεται για να ολοκληρωθούν όλες οι εργασίες, p_{ij} είναι ο χρόνος επεξεργασίας της (i,j) διεργασίας, με το (i) να δηλώνει την εκάστοτε εργασία και το j να δηλώνει την εκάστοτε μηχανή, t_{ij} είναι ο χρόνος έναρξης της (i,j) διεργασίας, N είναι το σύνολο των διεργασιών και A το σύνολο τών περιορισμών. Τέλος, $[(i,j) \rightarrow (i,k)]$ δείχνει τη σειρά που γίνεται η επεξεργασία μιας εργασίας στις μηχανές, δηλαδή ότι η (i) εργασία πρώτα χρησιμοποιεί τη μηχανή (j) και έπειτα την (k).

2.3.2 Ελαχιστοποίηση της μέγιστης καθυστέρησης (maximum lateness)

Ένας άλλος στόχος για βλετιστοποίση είναι αυτός των χρόνων πρόωρης λήξης των εργασιών. Η διατύπωση του στόχου περιγράφεται από τη σχέση $\mathbf{L_{max} = C_j - d_j}$ όπου L_{max} είναι η μέγιστη καθυστέρηση (lateness), C_{max} ο συνολικός χρόνος επεξεργασίας και d_j είναι ο χρόνος πρόωρης λήξης της εργασίας (j). Το L_{max} παίρνει θετικές τιμές όταν η ολοκλήρωση της εργασίας (j) είναι καθυστερημένη και αρνητικές τιμές όταν είναι πρόωρη. Το L_{max} υπολογίζει τη χειρότερη περίπτωση παραβίασης των χρόνων πρόωρης λήξης.

2.3.3 Ελαχιστοποίηση της συνολικής αργοπορίας (Total Tardiness)

Το επόμενο κριτήριο για βελτιστοποίηση είναι η συνολική αργοπορία που αποτελεί παραλλαγή της μέγιστης καθυστέρησης. Ο στόχος εδώ δίνεται από την εξίσωση $T_j = \max[0, (C_{max} - d_j)] = \max[0, L_{max}]$ όπου L_{max} είναι η συνολική αργοπορία (tardiness). Η διαφορά με τη μέγιστη καθυστέρηση βρίσκεται στο γεγονός ότι το L_{max} δεν δέχεται αρνητικές τιμές.

2.3.4 Unit Penalty

Τέλος, μια ακόμα περίπτωση βελτιστοποίησης είναι αυτή των αργοπορημένων εργασιών ή της ποινής ανά μονάδα (unit penalty). Και αυτός ο στόχος έχει σχέση με αυτόν της μέγιστης καθυστέρησης, και περιγράφεται ως εξής:

$$U_j = \begin{cases} 0, & C_{max} \leq d_j \\ 1, & C_{max} > d_j \end{cases} \tag{2.2}$$

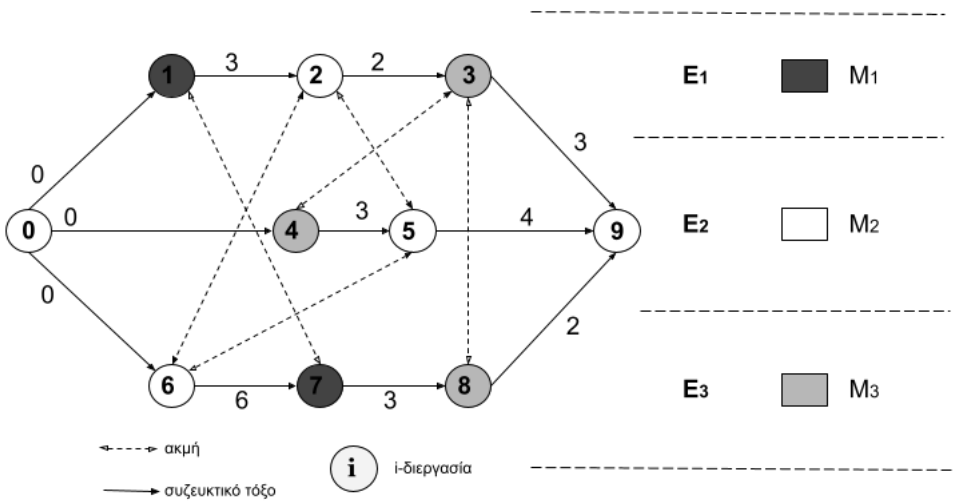
με το C_{max} να είναι ο συνολικός χρόνος επεξεργασίας και το d_j είναι ο χρόνος πρόωρης λήξης της εργασίας (j). Τέλος, από το άρθροισμα $\sum w_j U_j$ δίνεται το σύνολο των αργοπορημένων εργασιών, όπου w_j είναι το κόστος κάθε εργασίας.

2.4 Γράφοι για την αναπαράσταση του προβλήματος

Η αναπαράσταση του προβλήματος αποτελεί σημαντικό παράγοντα τόσο για την κατανόηση όσο και για την επίλυση του. Από τα αρχικά στάδια μελέτης του JSSP, οι ερευνητές έχουν επιλέξει να χρησιμοποιούν διαζευκτικούς γράφους καθώς η παρουσίαση του προβλήματος μέσα από αυτούς φαίνεται να είναι απλή. Όπως το παρουσιάζουν οι Ballas [11], Roy και Sussman (1964) [12] έστω ένας γράφος $G = (O, A, E)$, όπου O το σύνολο τών κόμβων, A το σύνολο των τόξων και E το σύνολο των ακμών. Στο Σχήμα 2.1 φαίνεται ένα πρόβλημα 3 μηχανών με 9 διεργασίες. Οι κόμβοι αντιστοιχούν στις διεργασίες, τα τόξα απεικονίζουν τη δρομολόγηση των διεργασιών της ίδιας δουλειάς και οι ακμές τα ζέυγη μεταξύ των διεργασιών που εκτελούνται αό την ίδια μηχανή. Για κάθε κόμβο (j) από το O(0, o+1) έχουμε μοναδικούς κόμβους (i), (l) έτσι ώστε (i,j) και (j,l) να είναι στοιχεία του A. Σε αυτή την περίπτωση εννοείται ότι (i) είναι πρόγονος του (j) ενώ (l) είναι ο διάδοχος του (j).

Έστω ότι έχουμε D = (O, A) ένα κατευθυνόμενο γράφο που προκύπτει από τον G, δηλαδή αφαιρούμε το σύνολο των ακμών. Επιπλέον, ας εννοειθεί ότι S είναι το σύνολο επιλογών στα ζέυγη διεργασιών από το E, με τέτοιο τρόπο ώστε να μην δημιουργούνται κύκλοι στον γράφο. Σκοπός είναι να αντικαταστήσουμε όλες τις ακμές από τον D, σχηματίζοντας ένα μη-κυκλικό γράφο $D_s = (O, A \cup S)$. Η καλύτερη λύση θα προκύψει σε ένα D_s που έχει κρίσιμο μονοπάτι, από τον αρχικό κόμβο μέχρι τον τελικό με το μικρότερο C_{max} (makespan). Όπου το C_{max} μέγιστο

Σχήμα 2.1: Παράδειγμα 3 μηχανών, 9 διεργασιών με τη χρήση γράφου

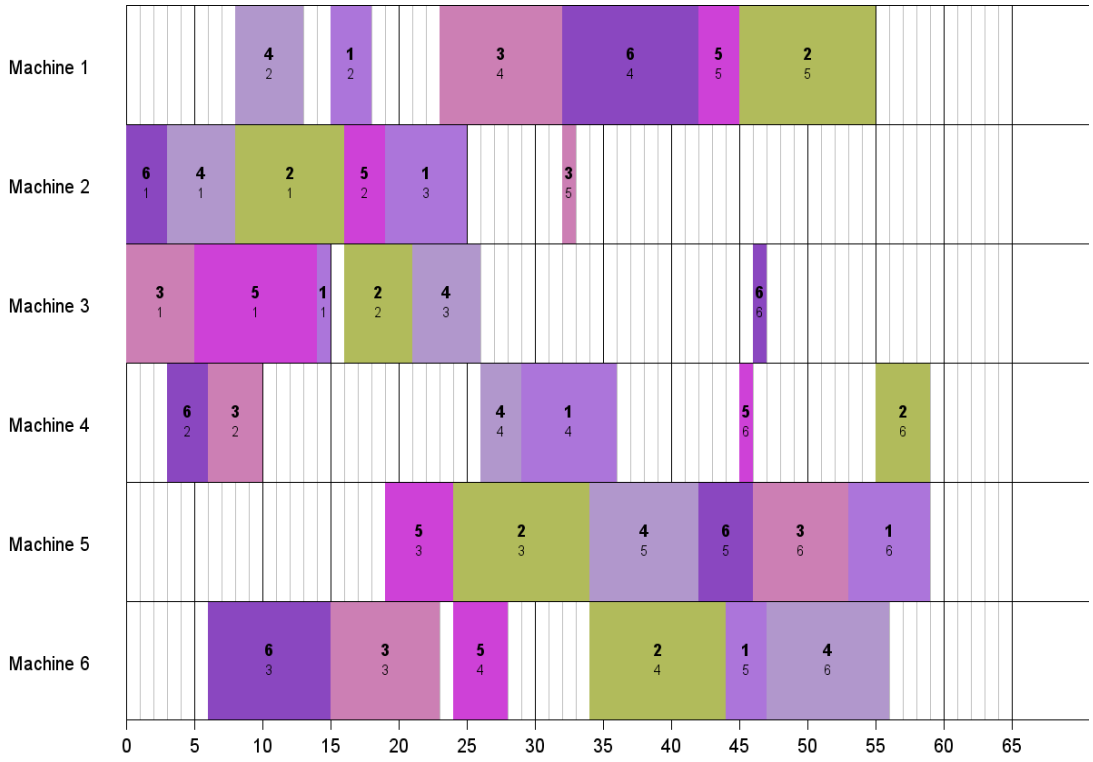


επιβαρυμένο μονοπάτι και αντιστοιχεί στο μέγιστο χρόνο επεξεργασίας όλων των εργασιών.

2.5 Διαγράμματα Gantt για την αναπαράσταση του προβλήματος

Τέλος, μια ακόμα τεχνική για την αναπαράσταση ενός προβλήματος χρονοπρογραμματισμού, και ιδιαίτερα του Job Shop, είναι τα διαγράμματα Gantt. Αυτά απεικονίζουν πληροφορίες σχετικά με το χρονοδιάγραμμα του εκάστοτε προβλήματος, δηλαδή μια λύση αυτού. Σε οριζόντιο άξονα απεικονίζει το χρόνο που χρειάζεται κάθε εργασία για να ολοκληρωθεί στη διάρκεια του χρόνου. Επιπλέον, παρέχει πληροφορίες αναφορικά με τις εξαρτήσεις που υπάρχουν μεταξύ των διεργασιών, αλλά και μια γενικότερη εικόνα του προγράμματος που παράγεται. Με αυτό τον τρόπο, οι διαχειριστές των προβλημάτων μπορούν με ευκολία και αποδοτικότητα να εντοπίζουν πόρους και να επιτηρούν όλη τη διαδικασία. Τα διαγράμματα Gantt αποτελούν έτσι ένα χρήσιμο εργαλείο για τη βελτιστοποίηση των διαδικασιών παραγωγής αλλά και της αποτελεσματικότητας σε Job Shop προβλήματα. Στο Σχήμα 2.2 δίνεται ένα τέτοιο διάγραμμα που απεικονίζει το ”δρομολόγιο” των εργασιών στις μηχανές, με τη χρήση ενός ευρετικού αλγόριθμου στο πρόβλημα MT06 (6x6), των Muth-Thompson. Σε αυτό το πρόβλημα, έχουμε 6 εργασίες και 6 μηχανές. Βλέπουμε ότι τον κάθετο άξονα αντιπροσωπεύουν οι μηχανές, κάθε γραμμή αντιστοι-

Σχήμα 2.2: Παράδειγμα διαγράμματος Gantt για το πρόβλημα FT06



χεί στην εκάστοτε μηχανή (στην 1η γραμμή βρίσκεται η μηχανή 1). Στον οριζόντιο άξονα δίνεται ο συνολικός χρόνος της ολοκλήρωσης των εργασιών. Κάθε ράβδος αντιπροσωπεύει μια εργασία (με διαφορετικό χρώμα) και η θέση της κατά μήκος του οριζόντιου άξονα την ώρα έναρξης της εργασίας στην αντίστοιχη μηχανή. Τέλος, το μήκος της ράβδου αντιπροσωπεύει τη διάρκεια του χρόνου επεξεργασίας της εργασίας στη συγκεκριμένη μηχανή. Παρατηρούμε πως μια εργασία που χρησιμοποιεί μια διαθέσιμη μηχανή, στη συνέχεια δεν ξαναχρησιμοποιεί την ίδια δεύτερη φορά, με σεβασμό στον περιορισμό ΠΥ16 του πίνακα 2.2. Αλλή μια πληροφορία που αντλούμε από το συγκεκριμένο διάγραμμα είναι το γεγονός ότι κάτα την επεξεργασία μιας διεργασίας επιτρέπεται η διακοπή της. Κάθε διεργασία ολοκλήρωνεται και στη συνέχεια γίνεται η ανάθεση της επόμενης στη μηχανή που είναι διαθέσιμη. Βλέπουμε, λοιπόν, πόσες σημαντικές πληροφορίες μπορούμε να αντλήσουμε από ένα διάγραμμα Gantt και την αναγκαιότητα του για την κατανόηση ενός προβλήματος χρονοπρογραμματισμού.

Κεφάλαιο 3

Βιβλιογραφική Ανασκόπηση

Σε αυτό το κεφάλαιο οπώς αναφέραμε και στην εισαγωγή ασχολούμαστε με τη βιβλιογραφία που υπάρχει για το πρόβλημα. Στο πλαίσιο αυτό, θα ήταν θεμιτό να γίνει μια ανάλυση και επεξήγηση των πληθώρων τεχνικών αλλα και των κατηγοριών που αυτές ανήκουν. Αυτό γιατί, όπως θα παρατηρήσουμε παρακάτω με το πέρασμα των χρόνων υπάρχει πολύ μεγάλη εξέλιξη στις υλοποιήσεις για το JSSP. Το υπόλοιπο κεφάλαιο θα χωριστεί σε τρία τμήματα ως εξής: κατηγορίες αλγορίθμων, αρχικές μελέτες και μεταγενέστερες προσεγγίσεις στο JSSP.

3.1 Κατηγορίες αλγορίθμων

Πριν ξεκινήσουμε με την ανασκόπηση της βιβλιογραφίας, είναι απαραίτητο να εξηγήσουμε τις κατηγορίες που ανήκουν οι διάφορες τεχνικές που θα παρουσιάσουμε στη συνέχεια. Γενικά, οι αλγόριθμοι μπορούν να διαχωριστούν σε τρείς κατηγορίες τους ακριβείς, τους προσεγγιστικούς και τους ευρετικούς. Οι ακριβείς αλγόριθμοι στοχεύουν κάθε φορά στο να βρίσκουν τη βέλτιστη λύση. Σε αυτή την κατηγορία ανήκουν κυρίως μέθοδοι που κάνουν εξαντλητικές επαναλήψεις, με μέγαλο χρόνο εκτέλεσης αλλά και μεγάλο κόστος υπολογιστικά. Τέτοιες τεχνικές είναι πολύ αποδοτικές για προβλήματα όπου η δυσκολία είναι μικρή, ωστόσο στα πολύπλοκα και δύσκολα προβλήματα δεν αποδίδουν με ανάλογο τρόπο και πολλές φορές δεν βρίσκουν καν λύση. Οι προσεγγιστικοί αλγόριθμοι είναι επίσης πολύ αποδοτικοί καθώς πάντα βρίσκουν μια λύση στο εκάστοτε πρόβλημα, αλλά αυτή δεν θα είναι η βέλτιστη. Βέβαια η λύση αυτή έχει εγγυημένη ποιότητα δηλαδή είναι πολύ κοντά στη βέλτιστη, συνήθως μια ή δύο τάξεις μικρότερη. Τέλος, οι ευρετικοί αλγόριθμοι δουλεύουν αρκετά ικανοποιητικά για διάφορα προβλήματα ανεξάρτητα από τη δυ-

σκολία και την πολυπλοκότητα τους. Σε αντίθεση όμως με τους προσεγγιστικούς η ποιότητα της λύσης δεν είναι εγγυημένη, πιο συγκεκριμένα τέτοιες μέθοδοι μπορούν να βρουν τη βέλτιστη λύση όμως μπορεί να επιστρέψουν και μια πολύ χειρότερη λύση απο τη βέλτιστη. Τέλος, αξίζει να αναφέρουμε ως υποκατηγορία τους μεταευρετικούς αλγορίθμους που έχουν σαν βάση τους ευρετικούς ωστόσο, πρόκειται για υψηλού επιπέδου τεχνικές.

3.2 Αρχικές μελέτες στο job scheduling problem

Τα προβλήματα χρονοπρογραμματισμού, και ιδιαίτερα το JSSP, όπως αναφέραμε και στη προηγούμενη ενότητα απασχολούν σε μεγάλο βαθμό τους ερευνητές, εξού και η εκτεταμένη έρευνα που έχει γίνει, με τη βιβλιογραφία του προβλήματος να είναι πολύ μεγάλη. Ο Johnson [13] ήταν ο πρώτος που άρχισε να κάνει έρευνα πάνω στο πρόβλημα και να πραγματοποιεί ανάλυση σε αυτό. Παρουσίασε την παραγωγή χρονοπρογραμμάτων για δύο και τρία στάδια παραγωγής, χρησιμοποιόντας ένα κανόνα απόφασης με στόχο αυτό να γίνεται στο λιγότερο συνολικό χρόνο. Επίσης, σύμφωνα με τους Arisha et al. [14], μαζί με τον Johnson (1954) [13] σημαντικές μελέτες διεξήγαγαν οι Smith (1956) [15], Jackson (1957) [16], Conway (1967) [17], Brooks (1965) [18], Roy & Sussmann (1976)[12].

Οι Muth και Thompson [9] με τη συνγραφή του βιβλίου τους συνέβαλαν σημαντικά στην εξέλιξη του JSSP. Στο μισό μέρος αυτού, εξέτασαν και συνέκριναν διάφορες τεχνικές ως προς την απόδοση τους. Με αυτό τον τρόπο κατάφεραν να προσεγγίσουν πολύπλευρα το πρόβλημα και να οδηγηθούν σε σημαντικά συμπεράσματα. Επιπλέον, κατασκεύασαν τρία προβλήματα για λύση, τα οποία αποτελούν σημεία αναφοράς μέχρι και σήμερα. Μάλιστα, το πρόβλημα των δέκα μηχανών-εργασιών (MT10X10) παρέμεινε χωρίς λύση για παραπάνω από 15 χρόνια.

Οι Giffler και Thompson [19], προσπαθώντας να επιλύσουν το πρόβλημα, εξέτασαν ακριβείς μεθόδους. Στη μελέτη τους αναλύουν ότι οι αλγόριθμοι παράγουν ένα αριθμό από δρομολόγια μέσα στα οποία υπάρχουν τα βέλτιστα. Αυτά τα ονομάζουν ως ''ενεργά δρομολόγια''. Τονίζουν πως η λειτουργικότητα αυτών για μικρά προβλήματα είναι αρκετά καλή καθώς, με τη δημιουργία πλήρων συνόλων από ενεργά δρομολόγια και στη συνέχεια, την επιλογή του καλύτερου χρονοδιαγράμματος απευθείας από αυτά. Στην περίπτωση που δεν είναι πρακτικοί επιλέγουν τυχαία κάποιο

δρομολόγιο απο τα "ενεργά". Με αυτό τον τρόπο βέβαια, πλησιάζουν κοντά στη βέλτιστη λύση αλλά δεν τη βρίσκουν.

Ο Ballas [11] αναπαρέστησε το πρόβλημα με τη χρήση διαζευκτικών γράφων με στόχο την εύρεση του ελάχιστου μονοπατιού. Η μέθοδος του παράγει μια σειρά από άκυκλους γράφους και έπειτα λύνει το πρόβλημα του κρίσιμου μονοπατιού (critical path) για κάθε έναν από αυτούς. Αυτή η διαδικασία επιτρέπει τη βελτίωση ενός αρχικού χρονοδιαγράμματος σταδιακά, με τη χρήση ευρετικών διαδικασιών και την αξιολόγηση υποψηφίων λύσεων για την καθοδήγηση της αναζήτησης. Με αυτόν τον τρόπο, μπορεί να επιτευχθεί μια "καλή" εφικτή λύση που πλησιάζει τη βέλτιστη, με περιορισμένη απαίτηση αποθήκευσης δεδομένων. Πλέον, οι ερευνητές επιλέγουν την υλοποίηση γράφων καθώς η αναπαράσταση των περιορισμών του προβλήματος είναι πολύ απλή.

Οι Graham et. al. [20] μετά από μια υπολογιστική μελέτη πάνω στα διάφορα προβλήματα χρονοπρογραμματισμού, πρότειναν το πεδίο τριών παραγόντων για την περιγραφή των προβλημάτων. Το μοντέλο αυτό έχει την εξής μορφή: $\alpha|\beta|\gamma$ όπου,

α: περιγράφει το τύπο του προβλήματος (Job, Flow, Open)

β: περιγράφει τα χαρακτηριστικά και τους περιορισμούς του προβλήματος

γ: περιγράφει τον στόχο βελτιστοποίησης

για παράδειγμα το $J|npmtn|Cmax$ περιγράφει ένα πρόβλημα Job-SSP όπου δεν επιτρέπεται η διακοπή επεξεργασίας μιας διεργασίας (non-preemptive), με στόχο την ελαχιστοποίηση του συνολικού χρόνου εκτέλεσης (makespan). Στη μελέτη τους ανέλυσαν την πολυπλοκότητα των προβλημάτων με τα οποία ασχολήθηκαν και την καταξή τους σε δύσκολα ή απλά. Παρατήρησαν, δε, πως για τα περισσότερα προβλήματα υπάρχουν δίαφορες λύσεις ενώ λίγα ήταν τελείως ανεξερεύνητα.

Ο Graves [21] πραγματοποίησε μελέτη πάνω στα προβλήματα χρονοδρομολόγησης, στοχεύοντας να κατηγοριοποιήσει τις διαφορετικές μορφές, στις οποίες αυτά εμφανίζονται. Έδωσε στο πρόβλημα τρείς διαστάσεις. Η πρώτη αναφέρεται στο είδος του προβλήματος, που μπορεί ανάλογα με τις απαιτήσεις παραγωγής να είναι Open-Shop ή Closed-Shop. Η δεύτερη σχετίζεται με τον αριθμό των σταδίων παραγωγής και της κατάστασης των μηχανών. Τέλος, η τρίτη απευθύνεται στα κριτήρια δρομολόγησης.

Την εφαρμογή διακριτού γραμμικού προγραμματισμού στο κλασικό πρόβλημα JSSP πρότεινε για πρώτη φορά ο Manne [22]. Λαμβάνοντας υπόψην όλους τους περιορισμούς του προβλήματος, έδειξε ότι σε σχέση με ήδη υπάρχουσες τεχνικές οι χρήση μεταβλητών ήταν μικρότερη. Αυτός ήταν και ο λόγος για τον οποίο έκρινε πως μια τέτοια προσέγγιση άξιζε περαιτέρω έρευνα. Ωστόσο, δεν παρέλειψε να σημειώσει ότι για προβλήματα μεγαλύτερου μεγέθους και πολυπλοκότητας, το υπολογιστικό κόστος θα ήταν πολύ μεγάλο.

Ο Carlier [23] πρότεινε μια μέθοδο βασισμένη σε δέντρα, που χρησιμοποιεί τον πίνακα διαδρομών μέγιστης αξίας μεταξύ οποιουδήποτε ζεύγους κορυφών. Παρά το γεγονός πως η τεχνική του ήταν αρκετά γενική, η εφαρμογή της σε προβλήματα αναφοράς ήταν αποδοτική επιλύοντας τα βέλτιστα. Παράλληλα, οι McMahon και Florian [24] κατάφεραν να λύσουν ένα από τα προβλήματα υλοποιώντας τη μέθοδο δρομολόγησης εργασιών σε έναν επεξεργαστή. Λειτουργεί όπως ένας Branch and Bound αλγόριθμος με τη διαφορά ότι η λύση σχετίζεται με κάθε κόμβο δέντρου αρίθμησης.

Σε συνέχεια της τεχνικής [23] οι Pinson και Carlier [25] ανέπτυξαν μια μέθοδο branch and bound βασισμένη στο πρόβλημα μιας μηχανής. Μέσα από τη σωστή επιλογή δομής δεδομένων και τον πλεονασμό πληροφοριών μπόρεσαν να ελαχιστοποιήσουν τον αποθηκευτικό χώρο αλλά και την πολυπλοκότητα του αλγορίθμου. Εξεραιτικά αποδοτική μέθοδος με την οποία έγιναν οι πρώτοι ερευνητές που έλυσαν το πολύ γνωστό πρόβλημα MT10 (10x10), ενώ για μεγαλύτερα προβλήματα πλησίασαν στο 5% από τη βέλτιστη λύση.

3.3 Μεταγενέστερες προσεγγίσεις στο πρόβλημα

Οι τεχνικές που αναφέρθηκαν, παρότι έκεινη την εποχή συνέβαλλαν σημαντικά στην κατανόηση και τελικά στην επίλυση του προβλήματος, ήταν πολύ κοστοβόρες υπολογιστικά σε χρόνο ενώ, για πιο μεγάλα προβλήματα δεν ήταν αποδοτικές ή δεν έδιναν καμία λύση. Απόλυτα λογικό, αν κανείς αναλογιστεί πως όλες οι παραπάνω προσεγγίσεις έπρεπε να εξετάσουν όλες τις πιθανές περιπτώσεις του προβλήματος. Στα χρόνια που ακολούθησαν και μαζί με την ανάπτυξη της τεχνολογίας υπήρξαν νέες μέθοδοι για τη βελτιστοποίηση του προβλήματος.

Ο Davis [26] ήταν από τους πρώτους που επιχείρησαν να εφαρμόσουν γενετι-

κούς αλγορίθμους για τη λύση ενός απλού JSSP. Θεωρώντας ότι οι προσπάθειες να χρησιμοποιήσουν expert συστήματα και η θεώρηση γενικά του προβλήματος ντετερμινιστικά, αύξαναν τις πιθανότητες να εμφανίζονται τοπικά ελάχιστα, αποφάσισε να το αντιμετωπίσει με μη-ντετερμινιστικό τρόπο χρησιμοποιώντας γενετικούς αλγορίθμους.

Οι Adams et al. [27] χρησιμοποίησαν μια προσεγγιστική μέθοδο για να λύσουν το κλασικό JSSP, το πλέον γνωστό Shifting Bottleneck Procedure. Σε αυτή τη μέθοδο, οι μηχανές δρομολογούνται με τη σειρά επιλέγοντας κάθε φορά τη μηχανή που προκαλεί τη μεγαλύτερη καθυστέρηση, ανάμεσα στις μηχανές που δεν έχουν δρομολογηθεί ακόμα. Παράλληλα, μετά από κάθε δρομολόγηση πραγματοποιείται τοπική βελτιστοποίηση στις μηχανές. Με τη συγκεκριμένη μέθοδο, κατάφεραν να επιλύσουν ένα μεγάλο αριθμό προβλημάτων εκείνης της περιόδους, συμπεριλαμβανομένου και του περιβόητου πρόβληματος MT10. Τέλος, δεν παρέλειψαν να αναφέρουν πως συγκριτικά με άλλες τεχνικές εκείνης της εποχής, η τεχνική τους υπερτερούσε σημαντικά σε χρονική διάρκεια παραγωγής λύσεων.

Οι Applegate και Cook [28] ερευνώντας το JSSP, σχεδίασαν μια ευρετική διαδικασία για την κατασκευή προγραμμάτων, με τη χρήση ενός συνδυαστικού Branch and Bound αλγορίθμου. Το βασικό κορμό του αλγορίθμου αποτέλεσε ο branch and bound των Carlier και Pinson [25]. Όμως, μαζί με αυτόν χρησιμοποίησαν και την ευρετική μέθοδο των Adam et al. [27] που αναφέραμε και προηγουμένως. Επιπλέον, εμπλούτισαν τον αλγόριθμο την τεχνική κοπής-επιπέδου για τον υπολογισμό του κάτω ορίου. Αυτός λοιπόν ο συνδυαστικός branch and bound αλγόριθμος ήταν εξεραιτικά αποδοτικός καθώς, κατάφεραν να λύσουν το πρόβλημα των Muth and Thompson (1963) [9] (MT10X10) σε 7 λεπτά ένω μόλις σε 7 από τα 53 προβλήματα που χρησιμοποίησαν, δεν πήραν κάποια λύση.

Οι van Laarhoven et al. [29] εφάρμοσαν την τεχνική του Προσομοιωμένου Ανορθωτή-Simulated Annealing (SA) μέσα από ένα προσεγγιστικό αλγόριθμο με στόχο να μειώσουν το συνολικό χρόνο ολοκλήρωσης των εργασιών (makespan). Αυτό το πέτυχαν γενικεύοντας τον αλγόριθμο με τέτοιο τρόπο ώστε να αποφεύγονται τοπικά ελάχιστα. Παρότι βρίσκει καλύτερα makespan, έχει αρκετά μεγάλο χρονικό κόστος.

Ο Taillard [30] υλοποίησε την τεχνική Tabu search για το Job Shop Scheduling Problem. Έδειξε ότι λειτουργεί καλύτερα από το Shifting Bottlenect Procedure και

από το Simulated Annealing. Επιπλέον, έδωσε και ένα γρήγορο παράλληλο αλγόριθμο για τη λύση πιο σύνθετων προβλημάτων. Παράλληλα, ο ίδιος [31] ανέπτυξε μεθόδους για τη δημιουργία προβλημάτων σημεία αναφοράς για τα Job, Flow και Open Shop Scheduling Problem. Αξίζει να σημειωθεί ότι η δημιουργία όλων αυτών των προβλημάτων έγινε με τυχαίο τρόπο.

Οι Ow και Morton [32] μελέτησαν τη χρήση της ευρετικής τεχνικής Beam Search και τις επιδόσεις αυτής σε σύγκριση με άλλες διαφορετικές τεχνικές. Παράλληλα, παρατήρησαν την επίδραση των αλλαγών που έκαναν στη συνάρτηση αξιολόγησης, η οποία καθοδηγεί την αναζήτηση. Τα αποτελέσματα έδειξαν την αποδοτικότητα αυτής της απλής τεχνικής, αφού οι λύσεις ήταν πολύ ποιοτικές.

Οι Sabuncuoglu και Gurgun (1996) [33] πρότειναν μία τεχνική βασισμένη στα νευρωνικά δίκτυα (Neural Networks) για την επίλυση του προβλήματος μιας μηχάνης με συνολική αργοπορία αλλά, και το κλασικό πρόβλημα JSSP (makespan). Η απόδοση της συγκρίθηκε με τις ήδη υπάρχουσες τεχνικές και τα αποτελέσματα βρέθηκαν ιδιαίτερα ενθαρρυντικά καθώς για αρκετά προβλήματα, ο αλγόριθμος τους έβρισκε τη βέλτιστη λύση.

Οι Lin et al. (2010) [34] σχεδίασαν έναν ευφυή υβριδικό αλγόριθμο, βασισμένο στην τεχνική βελτιστοποίησης με σμήνος σωματιδίων (Particle Swarm Optimization, PSO) σε συνδυασμό με τη μέθοδο simulated annealing και ένα πολλαπλών τύπων ατομικό σχέδιο για να επιλύσουν το JSSP. Ο προτεινόμενος αλγόριθμος, σε αντίθεση με άλλους εξελικτικούς, δεν απαιτεί κάποια ευρετική συνάρτηση για να δημιουργήσει τον πληθυσμό που χρειάζεται για να αυξηθεί ο ρυθμός σύγκλισης. Τα αποτελέσματα τους έδειξαν ότι ο MPSO, όπως τον ονόμασαν, βρίσκει βέλτιστες λύσεις για μικρό μέγεθος πληθυσμού και επαναλήψεις, σε σχέση με άλλους αλγορίθμους.

Οι Sakawa και Kuboka [35] εισήγαγαν τα ασαφή προβλήματα Job Shop πολλαπλών στόχων, αναφορικά με τους ασαφείς χρόνους επεξεργασίας και πρόωρης λήξης εργασίας (due dates). Για τους ασαφείς στόχους του λήπτη αποφάσεων, συνδύασαν γραμμικές συναρτήσεις μελών μαζί με τον ασαφή τρόπο απόφασης των Bellman και Zadeh. Για την επίλυση τους, παρουσίασαν ένα γενετικό αλγόριθμο στον οποίο ενσωμάτωσαν την έννοια της ομοιότητας μεταξύ ατόμων. Ανέφεραν πως συγκριτικά με τη μέθοδο simulated annealing για τα προβλήματα που χρησιμοποίησαν, ο γενετικός τους αλγόριθμος λειτούργησε καλύτερα σε όλες τις περιπτώσεις.

τη μέθοδο βελτιστοποίησης αποικίας μυρμηγκιών (Ant Colony Optimization) χρησιμοποίησαν οι Colorni et al. [36] πάνω στο JSSP. Η αποστολή αναζήτησης γίνεται από πράκτορες που αλληλεπιδρούν μεταξύ τους. Η επιτυχία του μοντέλου απονέμεται στη συνεργασία των πρακτόρων, η οποία πραγματοποιείται περιοδικά με την τροποποίηση μιας δομής μνήμης του προβλήματος ονομαζόμενη trail matrix. Για τη μετακίνηση των πρακτόρων χρησιμοποίησαν έναν άπληστο ευρετικό μηχανισμό. Στα αποτελέσματα φάνηκε η σταθερότητα της μεθόδου και ότι για την υπολογιστική της απλότητα, σε δύσκολες καταστάσεις είναι αποδοτική.

Τέλος, οι Çaliş et al. [37] έχουν πραγματοποιήσει μια εκτενή έρευνα πάνω στη επίλυση του προβλήματος με τη χρήση στρατηγικών τεχνητής νοημοσύνης. Αρχικά, κάνουν μια επέξηγηση και στο πρόβλημα χρονοδρομολόγησης εργασιών αλλά και στο θέμα της τεχνητής νοημοσύνης και στη συνέχεια αναλύουν μια σειρά από διάφορες τεχνικές για τη βελτιστοποίηση του JSSP.

Κεφάλαιο 4

Παρουσίαση Αλγορίθμων

Στο κεφάλαιο αυτό θα παρουσιάσουμε τους κώδικες που υλοποιήσαμε για την επίλυση του προβλήματος χρονοδρομολόγησης εργασιών JSSP. Οι αλγόριθμοι που χρησιμοποιήθηκαν για την επίλυση του προβλήματος είναι ένας ακριβής αλγόριθμος διακλάδωσης και δέσμευσης (Branch and Bound) και τρείς ευρετικοί αλγόριθμοι. Ο πρώτος είναι ο αλγόριθμος Προσομοιωμένου Ανορθωτή, ο δεύτερος είναι ο αλγόριθμος Σμήνους Σωματιδίων (Particle Swarm Optimization) και ο τρίτος είναι ο αλγόριθμος Αποικίας Μυρμυγκιών (Ant Colony Optimization). Για τη δημιουργία όλων των αλγορίθμων, συνολικά κατασκευάσαμε 23 κλάσεις (java αρχεία). Στη συνέχεια, θα παρουσιάσουμε με τη σειρά που αναφέραμε τους τέσσερις αλγορίθμους, θα τους αναλύσουμε σε θεωρητικό επίπεδο και θα παρουσιάσουμε με παραδείγματα μεθόδων τη λειτουργία του καθενός με σκοπό να γίνουν απόλυτα κατανοητοί.

4.1 Διακλάδωση και δέσμευση / branch and bound

Ο αλγόριθμος Branch and Bound (B&B) παρουσιάστηκε για πρώτη φορά από τους Land και Doig το 1960 [38]. Ανέπτυξαν αυτόν τον αλγόριθμο ως μέθοδο επίλυσης προβλημάτων ακέραιου γραμμικού προγραμματισμού. Η μέθοδος Branch and Bound (B&B), είναι πλέον ευρέως γνωστή και χρησιμοποιείται πολύ συχνά ως τεχνική βελτιστοποίησης για την επίλυση προβλημάτων συνδυασμών. Ανήκει στη κατηγορία των αλγορίθμων που εξερευνούν συστηματικά τον χώρο αναζήτησης του προβλήματος, με στόχο να βρεί τη βέλτιστη λύση. Αυτό σημαίνει πως για προβλήματα που το επίπεδο της πολυπλοκότητας είναι αρκετά υψηλό, μια τέτοια τεχνική θα είναι αρκετά κοστοβόρα υπολογιστικά. Ωστόσο, όπως θα δούμε και στη συνέχεια, με τη σωστή δομή ένας τέτοιος αλγόριθμος μπορεί να είναι πολύ αποδοτικός

ακόμα και για τα σύνθετα προβλήματα.

4.1.1 Γενικός τρόπος λειτουργίας B&B

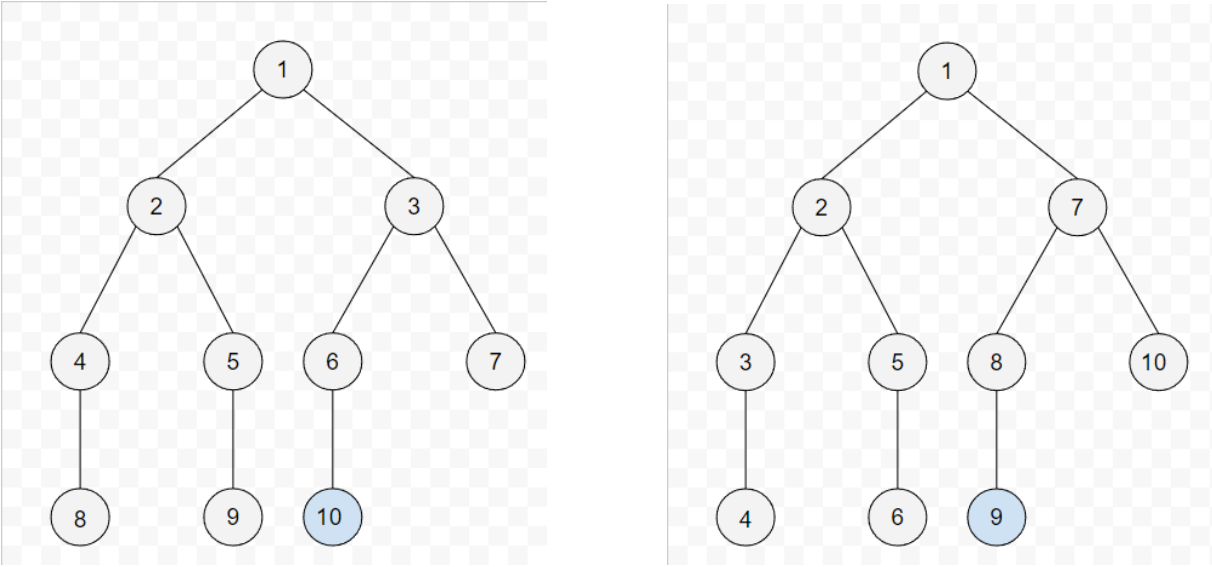
Στο πυρήνα του, ο B&B εφαρμόζει στρατηγική ''διαίρει και βασίλευε'' για την αποτελεσματική εξερεύνηση του χώρου λύσεων με τη συστηματική κατάτμησή του σε μικρότερους υποχώρους, το κλάδεμα των άγονων κλάδων και την οριοθέτηση της αναζήτησης για την αποφυγή περιττής εξερεύνησης. Με αυτό τον τρόπο μπορεί και βελτιώνει την υπολογιστική προσπάθεια, μέχρι να βρεί τη βέλτιστη λύση του προβλήματος, σε σύγκριση με άλλες εξαντλητικές μεθόδους αναζήτησης. Το χώρο αναζήτησης συνήθως αποτελεί ένα δέντρο αναζήτησης, όπου σε αυτό, εφαρμόζεται κάποια στρατηγική αναζήτησης. Επιπλέον, η καλύτερη λύση αποθηκεύεται εξαρχής και σε κάθε επανάληψη αν βρεθεί κάποια καλύτερη τότε γίνεται ενημέρωση αυτής. Στη συνέχεια αυτού του τμήματος, θα αναλύσουμε τα τρία στάδια ενός αλγόριθμου Branch and Bound, το στάδιο της εξερεύνησης, το στάδιο της διακλάδωσης και το στάδιο της οριοθέτησης.

Εξερεύνηση

Στο στάδιο της εξερεύνησης, ο αλγόριθμος εξερευνά συστηματικά τους κλάδους του δέντρου αναζήτησης με στόχο να βρεί την καλύτερη λύση. Αυτό περιλαμβάνει τη διάσχιση του δέντρου αλλά και τον τερματισμό αυτής όταν ικανοποιείται κάποιο κριτήριο. Για αυτό το στάδιο συνήθως γίνεται η χρήση κάποιας στρατιγικής εξερεύνησης. Η επιλογή μιας μεθόδου αναζήτησης είναι πολύ κρίσιμη καθώς επηρεάζει την αναζήτηση σημαντικά. Αρχικά, ο χώρος αναζήτησης, δηλαδή, το δέντρο μπορεί να διαμορφωθεί διαφορετικά για διαφορετικές μεθόδους αναζήτησης. Στο Σχήμα 4.1 φαίνεται πως αλλάζει ο χώρος αναζήτησης με τη χρήση της αναζήτησης πρώτα σε βάθος και πώς με την αναζήτηση πρώτα σε πλάτος. Αυτό συμβαίνει γιατί έχουμε διαφορετική σειρά στη δημιουργία/εξερεύνηση των υποπροβλημάτων.

Διακλάδωση

Ο αλγόριθμος ξεκινά με μια αρχική λύση (ή μια μερική λύση) και διαιρεί το πρόβλημα σε μικρότερα υποπροβλήματα, που ονομάζονται κλάδους. Κάθε κλάδος αντιπροσωπεύει μια πιθανή επέκταση της τρέχουσας λύσης. Η σωστή επιλογή δια-



(α′) Αναζήτηση πρώτα σε πλάτος (β′) Αναζήτηση πρώτα σε βάθος

Σχήμα 4.1: Παράδειγμα Στρατηγικών Αναζήτησης

κλάδωσης μπορεί να καθορίσει πόσα υπο-προβλήματα θα παρουσιαστούν. Υπάρχουν πολλές τεχνικές διακλάδωσης, ωστόσο πρέπει να τονίσουμε πως η επιλογή της εκάστοτε στρατηγικής εξαρτάται και από τους περιορισμούς του προβλήματος μας. Στη δική μας περίπτωση με το Job Shop Scheduling Problem δύο πολύ γνωστές στρατηγικές είναι οι διακλαδώσεις βάσει των εργασιών και βάσει των μηχανών.

Κλάδεμα (Pruning)

Το ”κλάδεμα” περιλαμβάνει την εξάλειψη των μη υποσχόμενων κλάδων του δέντρου αναζήτησης, με στόχο τη μείωση του χώρου αναζήτησης και τη βελτίωση της αποτελεσματικότητας. Αυτό επιτυγχάνεται μέσα από δύο βήματα την οριοθέτηση (bounding) και την οπισθοδρόμηση (backtracking). Η οριοθέτηση περιλαμβάνει την εκτίμηση των φραγμών της βέλτιστης λύσης για κάθε υποπρόβλημα και τη χρήση αυτών για την εξάλειψη των κλάδων που δεν είναι δυνατόν να οδηγήσουν σε βέλτιστη λύση. Τα φράγματα είναι στην ουσία δύο: το κάτω και το άνω φράγμα. Το άνω φράγμα αντιπροσωπεύει ένα ανώτατο όριο στην τιμή του βέλτιστου στόχου. Λειτουργεί ως σημείο αναφοράς για την αξιολόγηση της ποιότητας των λύσεων που βρέθηκαν κατά τη διάρκεια της διαδικασίας αναζήτησης. Ο στόχος είναι να βελτιωθεί διαρκώς το ανώτερο φράγμα βρίσκοντας καλύτερες λύσεις μέχρι να συγκλίνει προς τον βέλτιστο χρόνο ολοκλήρωσης. Το κάτω φράγμα είναι μια εκτίμηση για κατώτερο όριο στην τιμή του βέλτιστου στόχου, που μπορεί να επιτευχθεί από

οποιαδήποτε εφικτή λύση. Λειτουργεί ως οδηγός για την περικοπή κλαδιών του δέντρου αναζήτησης που δεν μπορούν να οδηγήσουν σε καλύτερες λύσεις από αυτήν που έχει βρεθεί μέχρι στιγμής. Όσο πιο κοντά είναι το κατώτερο φράγμα στον βέλτιστο στόχο , τόσο πιο αποτελεσματική είναι η περικοπή και τόσο πιο γρήγορα μπορεί να συγκλίνει σε αυτόν. Για το βήμα της οπισθοδρόμησης, όταν συναντάται αδιέξοδο, περιλαμβάνει την επιστροφή στο προηγούμενο σημείο απόφασης στο δέντρο αναζήτησης και τη διερεύνηση εναλλακτικών κλαδιών, κλαδεύοντας με αυτό τον τρόπο τα μη ελπιδοφόρα μονοπάτια.

4.1.2 Βελτιστοποίηση του JSSP με τον B&B

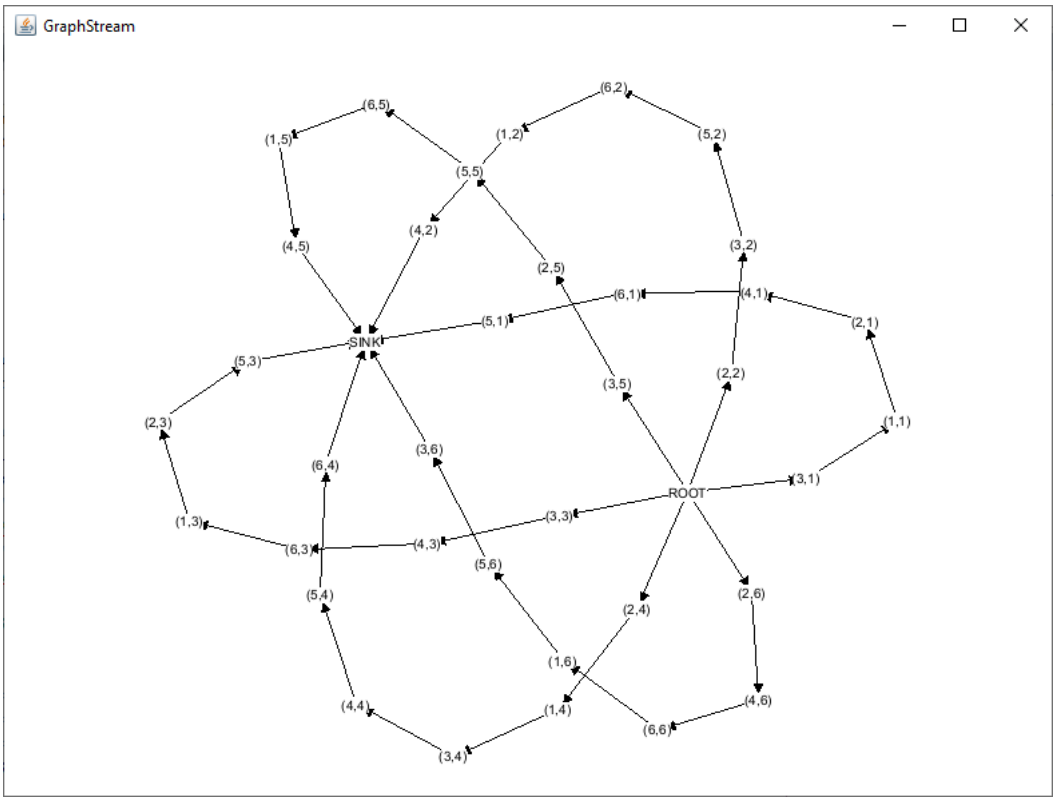
Η εφαρμογή ενός Branch and Bound αλγόριθμου για την επίλυση του JSSP είναι ιδιαίτερα αποτελεσματική όταν το πρόβλημα αντιπροσωπεύεται από ένα διαζευκτικό γράφο και συγκεκριμένα στο μοντέλο των Roy και Sussmann [12]. Όπως αναφέραμε και στο κεφάλαιο 2 για την αποτύπωση ενός JSSP μέσα από ένα γράφο, έχουμε V το σύνολο των κόμβων, που αντιπροσωπεύουν τον αριθμό των λειτουργιών του προβλήματος. Κάθε κόμβος επίσης φέρει ένα βάρος το οποίο αντιστοιχεί στο χρόνο επεξεργασίας κάθε λειτουργίας. Στη δική μας υλοποίηση της μεθόδου, αξιοποιόντας τη βιβλιοθήκη της Java multigraphs.streams δημιουργούμε το γράφο του εκάστοτε προβλήματος. Στο Σχήμα 4.2 φαίνεται ένας γράφος που δημιουργήσαμε για το πρόβλημα MT06(6x6), στην αρχική του κατάσταση.

Ακολουθώντας τα τρία βήματα των Nababan et. al. [39] σχεδίασαμε τον αλγόριθμο ως εξής:

Στο πρώτο βήμα, όπως βλέπουμε και στο 1, γίνεται η αρχικοποίηση των απαραίτητων δομών δεδομένων. Ένας νέος γράφος με αρχικό κόμβο το ”ROOT”, τη ρίζα του δέντρου. Επιπλέον, δημιουργούμε μια λίστα η οποία περιέχει τις συνδέσεις των υπόλοιπων κόμβων με τη ρίζα (ROOT), τη λίστα αυτή την ονομάσαμε ”omega”. Τέλος, καλούμε τη συνάρτηση που μας οδηγεί στο επόμενο βήμα, στην επιλογή μηχανής (machine selection).

Το δεύτερο βήμα ξεκινά με έλεγχο για να δει εάν η λίστα omega είναι άδεια. Εάν είναι, σημαίνει ότι όλοι οι κόμβοι έχουν επεξεργαστεί, και ελέγχει αν υπάρχει εφικτή λύση. Αν βρεί κάποια τότε ενημερώνει την αντίστοιχη μεταβλητή (upperbound). Σε διαφορετική περίπτωση, δύο λίστες, (tOmega και mOmega), αρχικοποιούνται για να

Σχήμα 4.2: Παράδειγμα γράφου στη java



Input : BBInstance instance

```
// reset bbTree graph
bbTree ← new MultiGraph(”bbTree”);
newNode ← bbTree.addNode(”ROOT”);
newNode.addAttribute(”ui.label”, ”ROOT”);
newNode.addAttribute(”ui.style”, ”text-background-mode: rounded-box;”);
newNode.addAttribute(”lowerBound”, 0);
// get the graph
G ← instance.getInitialState();
omega ← new ArrayList<MultiNode>();
// get all nodes that are directly connected to from ROOT
for each Edge e in G.getNode(”ROOT”).getLeavingEdgeSet() do
    e.getTargetNode().changeAttribute(”releaseDate”, 0);
    omega.add(e.getTargetNode());
end
// call step 2 of the B&B alg
BBStep2(instance, omega, bbTree.getNode(”ROOT”),
 topologicalSort(instance));
```

Αλγόριθμος 1: Βήμα 1ο: αρχικοποίηση δομών δεδομένων

αποθηκεύσουν τα αθροίσματα του χρόνου επεξεργασίας και της ημερομηνίας απελευθέρωσης, και τους αριθμούς των μηχανών αντίστοιχα. Για κάθε κόμβο, εξάγει τον αριθμό της μηχανής και υπολογίζει το άθροισμα του χρόνου επεξεργασίας και

της ημερομηνίας απελευθέρωσης, αποθηκεύοντάς τα στις δύο λίστες. Εξαιρούνται από αυτή τη διαδικασία ο αρχικός (ROOT) και τελικός (SINK) κόμβος. Μετά τη συλλογή των αθροισμάτων, βρίσκει την ελάχιστη τιμή από τη λίστα tOmega, που αντιπροσωπεύει το ελάχιστο του (χρόνου επεξεργασίας + ημερομηνία απελευθέρωσης). Στη συνέχεια αναγνωρίζει τις μηχανές στις οποίες συμβαίνει αυτό το ελάχιστο και τις αποθηκεύει στη λίστα minMachines. Τέλος, για κάθε μηχανή καλεί το τρίτο βήμα αναδρομικά, όπου θα εξερευνήσει περαιτέρω κλαδιά του δέντρου αναζήτησης. τη διαδικασία που μόλις αναφέραμε βλέπουμε στο 2.

Στο τρίτο 3 και τελευταίο βήμα, ξεκινάμε με τη δημιουργία μιας νέας λίστας με όνομα ”omegaPrime”, η οποία θα αποθηκεύει κόμβους που χρησιμοποιούν την ελάχιστη μηχανή (minMachine) και έχουν ημερομηνίες απελευθέρωσης μικρότερες από το (minTOmega). Για κάθε κόμβο στη λίστα omega, ελέγχει αν πληροί ορισμένα κριτήρια και τον προσθέτει στη λίστα ”omegaPrime”. Για κάθε κόμβο στην omegaPrime, δημιουργείται μια νέα λίστα ”newOmega” που είναι ένα αντίγραφο του ”omega” με τον τρέχοντα κόμβο αφαιρεμένο. Στη συνέχεια, ελέγχονται οι εξερχόμενες ακμές του τρέχοντος κόμβου για να βρεθούν εργασίες που εξαρτώνται από αυτόν και προστίθενται στη ”newOmega”. Έπειτα, ενημερώνονται οι ημερομηνίες απελευθέρωσης των εργασιών βάσει του τρέχοντος κόμβου και δημιουργείται μια νέα περίπτωση (newInstance) του προβλήματος. Εάν το κάτω φράγμα της νέας περίπτωσης είναι μικρότερο από το ανώτατο φράγμα, προστίθεται ένας νέος κόμβος στο δέντρο (bbTree), ενημερώνονται τα χαρακτηριστικά, δημιουργείται μια ακμή από την τελευταία περίπτωση στον νέο κόμβο και γίνεται αναδρομική κλήση του τρίτου βήματος, με τη νέα περίπτωση και το ενημερωμένο ”Omega”. Τέλος, επαναφέρεται η αρχική κατάσταση της τρέχουσας περίπτωσης και επαναφέρεται η προηγούμενη ταξινόμηση πριν προχωρήσει στον επόμενο κόμβο στο ”omegaPrime”.

Πρέπει να αναφέρουμε πως για τη στρατηγική αναζήτησης επιλέξαμε την αναζήτηση πρώτα σε βάθος (DFS) και συγκεκριμένα δύο αναζητήσεις σε βάθος DFSF και DFSB ξεκινώντας από τους κόμβους που πηγαίνουν (to) και από αυτούς που έρχονται (from) αντίστοιχα, για να εντοπιστούν οι κόμβοι που είναι προσβάσιμοι από ένα κόμβο ”to” και εκείνους που μπορούν να φτάσουν σε ένα κόμβο ”from”. Τέλος, για τον υπολογισμό της καλύτερης λύσης χρησιμοποιήσαμε το μεγαλύτερο μονοπάτι, το οποίο αντιστοιχεί στις ημερομηνίες απελευθέρωσης για κάθε κόμβο

```
Input : BBInstance currInstance, ArrayList<MultiNode> omega, Node
        lastInstance, ArrayList<Node> previousSort

BBSolver(currInstance, omega, lastInstance, previousSort) // Check if omega
    is empty
if omega.isEmpty() then
    lastInstance.setAttribute("ui.label", lastInstance.getAttribute("ui.label") + "
     FEASIBLE SOLUTION");
    if getLowerBound(currInstance) < upperBound then
        upperBound = getLowerBound(currInstance);
        upperBoundGraph = currInstance.getInitialStateSimple();
        System.out.println("New upperBound found with value: " +
         upperBound);
    end
    return;
end
// Initialize arrays to store processing times and machines
ArrayList<Integer> tOmega = new ArrayList<>();
ArrayList<Integer> mOmega = new ArrayList<>();
// Calculate the sum of processing time and release date for each
    node
for MultiNode n : omega do
    if !n.getId().equals("SINK") || !n.getId().equals("ROOT") then
        mOmega.add((Integer) n.getAttribute("machine"));
        tOmega.add((Integer) n.getAttribute("processingTime") + (Integer)
         n.getAttribute("releaseDate"));
    end
end
// Find the minimum sum of processing time and release date
int minTOmega = Collections.min(tOmega);
// Get machines where the minimum sum occurs
ArrayList<Integer> minMachines = new ArrayList<>();
for int i = 0; i < tOmega.size(); i++ do
    if tOmega.get(i) == minTOmega then
        minMachines.add(mOmega.get(i));
    end
end
// Recursively call BBSolver for each minimum machine
for Integer minMachine : minMachines do
    BBSolver(currInstance, omega, minTOmega, minMachine, lastInstance,
     previousSort);
end
```

Αλγόριθμος 2: Βήμα 2ο: επιλογή μηχανής

```
ArrayList<MultiNode> omegaPrime = new ArrayList<>();
for MultiNode n : omega do
    if n.getAttribute("machine") == minMachine && (Integer)
     n.getAttribute("releaseDate") < minTOmega then
        omegaPrime.add(n);
    end
end
// Store the current instance's initial state
MultiGraph initialState = currInstance.getInitialStateSimple();
ArrayList<Node> topSort = new ArrayList<>(previousSort);
for MultiNode n : omegaPrime do
    ArrayList<MultiNode> newOmega = new ArrayList<>(omega);
    newOmega.remove(n);
    for Edge e : n.getEdgeSet() do
        if e.getAttribute("type").equals("conjunctive") &&
         e.getSourceNode().equals(n) &&
         !e.getTargetNode().getId().equals("SINK") then
            newOmega.add((MultiNode) e.getTargetNode());
        end
    end
    // Create a new instance with updated release dates
    newInstance.updateReleaseDates(currInstance, n, topSort);
    ArrayList<MultiNode> updatedOmega = new ArrayList<>();
    for MultiNode mn : newOmega do
        updatedOmega.add((MultiNode)
         newInstance.getInitialStateSimple().getNode(mn.getId()));
    end
    if getLowerBound(newInstance) < upperBound then
        // Add a new node to the bbTree and recursively call BBSolver
        String id = n.getId() + bbTree.getNodeCount();
        bbTree.addNode(id);
        for String s : n.getAttributeKeySet() do
            Object attributeValue = n.getAttribute(s);
            if attributeValue instanceof Integer then
                attributeValue = new Object[]attributeValue;
            end
            bbTree.getNode(id).addAttribute(s, attributeValue);
        end
        bbTree.addEdge(lastInstance.getId() + " → " + id, lastInstance.getId(),
         id);
        bbTree.UpdateAttributes;
        BBSolver(newInstance, updatedOmega, minTOmega, minMachine,
         bbTree.getNode(id), topSort);
    end
end
```

Αλγόριθμος 3: Βήμα 3ο: διακλάδωση

στη δεδομένη τοπολογική σειρά.

4.2 Προσομοιωμένος ανορθωτής / simulated annealing

Βασισμένος στις αρχές της στατιστικής μηχανικής και εμπνευσμένος από τη φυσική διαδικασία της ανόρθωσης στη μεταλλουργία, ο Προσομοιωμένος Ανορθωτής (SA) αποτελεί ένα θεμέλιο στον τομέα των στοχαστικών αλγορίθμων βελτιστοποίησης. Προτείνεται από τους Kirkpatrick et al. [40] στις αρχές της δεκαετίας του 1980, ο SA μιμείται τη βαθμιαία ψύξη της διαδικασίας λιωσίματος μετάλλου για να επιτύχει μια κατάσταση χαμηλής ενέργειας, μεταφράζοντας αυτή την έννοια σε ένα πλαίσιο βελτιστοποίησης. Βασική ιδέα του SA είναι η προσεκτική διάσχιση του χώρου λύσεων, με την πιθανοτική αποδοχή νέων υποψήφιων λύσεων, παρόμοια με την εξερεύνηση του ενεργειακού τοπίου ενός υλικού κατά τη διάρκεια της ψύξης. Παρά την απλότητά του, η ικανότητα του SA να αποφύγει τα τοπικά ελάχιστα και να διασχίζει ανώμαλα τοπία το καθιστούν αναντικατάστατο στην επίλυση ποικίλων προβλημάτων βελτιστοποίησης, από συνδυαστικά προβλήματα μέχρι τη ρύθμιση παραμέτρων σε αλγόριθμους μηχανικής μάθησης. Με την ανθεκτικότητά του και την ευελιξία του, ο SA συνεχίζει να κατακτά ερευνητές και επαγγελματίες, προσφέροντας μια αιώνια απόδειξη της δύναμης της αναλογικής σκέψης στο σχεδιασμό αλγορίθμων.

Η διαδικασία της ανόρθωσης συνοπτικά δίνεται ως εξής:

- Αύξηση της θερμοκρασίας του θερμικού λουτρού σε μια μέγιστη τιμή, με την οποία το μέταλλο αρχίζει να λίωνει.
- Στη συνέχεια, μείωση αυτής με προσοχή μέχρι τα σωματίδια να οργανωθούν στο βασικό τους στάδιο, δηλαδή να πάρει πάλι στερεά μορφή το μέταλλο.

Βεβαίως στην ενδιάμεση κατάσταση που προκύπτει, δηλαδή στο στάδιο της υγρής μορφής του μετάλλου, όλα τα σωματίδια ταξινομούνται με τυχαίο τρόπο σε αντίθεση με το στάδιο της τήξης, όπου τα σωματίδια παρουσιάζουν ένα εξαιρετικά δομημένο πλέγμα και η αντίστοιχη ενέργεια είναι ελάχιστη. Αξίζει να αναφέρουμε ότι το βασικό στάδιο τήξης επιτυγχάνεται εάν η μέγιστη θερμοκρασία είναι υψηλή και ο ρυθμός ψύξης είναι αρκετά χαμηλός. Με αυτό το τρόπο, διασφαλίζεται το

ενδεχόμενο της ολικής ψύξης του στερεού και της μετάβασης σε μια μετά-βασική κατάσταση.

4.2.1 Γενικός Τρόπος Λειτουργίας Προσομοιωμένος Ανορθωτής

Στη γενική του μορφή ένας αλγόριθμος προσομοιωμένου ανορθωτή λειτουργεί με τον παρακάτω τρόπο.

- Αρχικοποίηση μιας σειράς καταστάσεων του στερεού.
- Για το τρέχον σταδίο i του στερεού με ενέργεια E_i, δημιουργείται επόμενη κατάσταση j μέσα από το μηχανισμό διαταραχών. Η επόμενη κατάσταση φέρει μικρή παραμόρφωση σε σχέση με την τρέχουσα και έχει ενέργεια E_j.
- Αποδοχή ή απόρριψη της νέας κατάστασης βάση του κριτηρίου Metropolis: $[p = \exp\left(-\frac{E_j - E_i}{kT}\right)]$ όπου, K είναι η σταθερά Boltzmann και T η θερμοκρασία.
- Το κόστος της λύσης αντιπροσωπεύεται από την ενέργεια της εκάστοτε κατάστασης.
- Επάναληψη της διαδικασίας μέχρι να επιτευχθεί η καλύτερη κατάσταση ή να έχουμε επιθυμητή θερμοκρασία ή να φτάσουμε το μέγιστο αριθμό επαναλήψεων.

Τέλος,γεγονός αποτελεί πως αν η μείωση της θερμοκρασίας είναι αργή, το στερεό μπορεί να πετύχει θερμική ισορροπία σε κάθε θερμοκρασία.

4.2.2 Βελτιστοποίηση του JSSP με τον προσομοιώμενο ανορθωτή

Στη δική μας υλοποίηση του Προσομοιωμένου Ανορθωτή, για την επίλυση του JSSP, ο αλγόριθμος αρχικά ξεκινάει με την αρχικοποίηση μιας πρώτης κατάστασης που δημιουργείται με τυχαίο τρόπο. Μια κατάσταση (state) αντιπροσωπεύει μια αποδεκτή λύση στο πρόβλημα μας. Μαζί με το αρχικό στάδιο υπολογίζει το συνολικό χρόνο εκτέλεσης του (makespan), ενώ αρχικοποιεί τη θερμοκρασία και το ρυθμό ψύξης που παραμένει σταθερός σε όλη τη διάρκεια εκτέλεσης του αλγορίθμου. Από το αρχικό στάδιο, ξεκινάει μια σειρά επαναλήψεων, όπου κάθε φορά παράγονται γειτονικές καταστάσεις. Αυτές οι νέες καταστάσεις δημιουργούνται μέσα από μικρές

τυχαίες αλλαγές στην τρέχουσα κατάσταση. Για κάθε γειτονική κατάσταση υπολογίζεται ο συνολικός χρόνος εκτέλεσης και με βάση τη συνθήκη αποδοχής Metropolis κρίνεται αν θα γίνει αποδεκτή ή όχι. Σε περίπτωση που βρεθεί καλύτερη κατάσταση, τότε ενημερώνεται η καλύτερη με αυτή και στη συνέχεια γίνεται η ενημέρωση της θερμοκρασίας. Η διαδικασία ενημέρωσης της θερμοκρασίας είναι ουσιαστικά η σταδιακή μείωση της και δίνεται από την πράξη: $Temp = Temp*(1-CR)$ όπου CR είναι ο ρυθμός ψύξης. Ο αλγόριθμος έπειτα συνεχίζει με την επόμενη επανάληψη μέχρι να φτάσει είτε κοντά στη βέλτιστη λύση είτε το όριο των επαναλήψεων. Στο 4 δίνεται η κύρια λειτουργία του αλγορίθμου μας, βασισμένη σε αυτά που αναφέραμε προηγουμένως. Στη συνέχεια θα αναλύσουμε ορισμένα σημεία του αλγόριθμου, όπως την αρχικοποίηση της πρώτης κατάστασης ή τη διαδικασία μετάβασης σε μια επόμενη.

```
Data: Temperature T, Cooling rate α, Makespan M
Result: Updated state and makespan
begin
    Initialize the first state, temperature, and the cooling rate;
    currentState ← generateRandomState();
    temperature ← T;
    coolingRate ← α;
    makespan ← M;
    Generate a neighboring solution;
    Integer[] neighborState ← generateNeighborState(currentState);
    Calculate makespan for the current and neighbor solutions;
    int neighborMakespan ← computeMakespan(neighborState);
    Calculate the change in makespan;
    int deltaMakespan ← neighborMakespan - makespan;
    Decide whether to accept the neighbor solution;
    if deltaMakespan < 0 or Math.exp(-deltaMakespan / temperature) > random()
     then
        currentState ← Arrays.copyOf(neighborState, neighborState.length);
        makespan ← neighborMakespan;
        // Update the best solution if needed;
        if makespan < bestMakespan then
            bestState ← Arrays.copyOf(currentState, currentState.length);
            bestMakespan ← makespan;
        end
    end
    // Update temperature and iteration counter;
    updateTemperature();
    ranIterations++;
end
```

Αλγόριθμος 4: Αλγόριθμος προσομοιωμένου ανορθωτή για το JSSP.

Όπως αναφέραμε και προηγουμένως, για την εκκίνηση του αλγορίθμου χρησιμοποιούμε μία αρχική τυχαία κατάσταση. Στη συνέχεια, είτε από την αρχική είτε και από μία τρέχουσα κατάσταση, δημιουργούμε ένα αριθμό από γειτονικές καταστάσεις που αποτελούν πιθανές καλύτερες λύσεις. Αυτές οι νεές καταστάσεις στην ουσία προκύπτουν από μικρές αλλαγές στην κατάσταση που βρίσκεται ο αλγόριθμος εκείνη τη στιγμή. Αυτές τις διαδικασίες παρουσιάζουμε στα 5 και 6.

1: **Input**: None
2: **Output**: Random state $randomState$ of type Integer[]
// Use Fisher-Yates shuffle algorithm to obtain a random permutation of the operation indices $randomState \leftarrow$ Array of size $getProblemInstance().getTotalOperations()$
3: **for** $i = 0$ to $randomState.length - 1$ **do**
4: $randomState[i] \leftarrow i$
5: **end for**
6: **for** $i = randomState.length - 1$ downto 1 **do**
7: $j \leftarrow$ Random integer between 0 and i
8: Swap $randomState[i]$ and $randomState[j]$
9: **end for**
10: **return** $randomState$

Αλγόριθμος 5: Δημιουργία τυχαίας κατάστασης

Στη παραπάνω μέθοδο, χρησιμοποιούμε τον αλγόριθμο Fisher-Yates ώστε να λάβουμε μια τυχαία διάταξη των λειτουργιών του προβλήματος που θα χρησιμοποιήσουμε για την αρχική μας κατάσταση. Αρχικοποιούμε ένα πίνακα με τους δείκτες των λειτουργιών. Έπειτα, ξεκινώντας από την τελευταία θεση του πίνακα με επαναληπτικό ρυθμό μέχρι να φτάσουμε στην αρχή, δημιουργούμε ένα τυχαίο δείκτη από 0 έως τον τρέχοντα δείκτη. Στη συνέχεια, ανταλλάζουμε τα στοιχεία της τρέχουσας θέσης με εκείνης του τυχαίου δείκτη. Αυτή η διαδικασία ανακατεύει τον πίνακα τυχαία, εξασφαλίζοντας ότι κάθε μετάθεση είναι εξίσου πιθανή.

Η συγκεκριμένη μέθοδος, δημιουργεί ένα αντίγραφο του πίνακα τρέχουσας κατάστασης (currentState) για να αποφύγει την τροποποίηση του αρχικού πίνακα. Επιλέγει δύο τυχαίους δείκτες (index1 και index2) εντός των ορίων του πίνακα. Έπειτα, ανταλλάσσει τα στοιχεία στους επιλεγμένους δείκτες για να δημιουργήσει μια νέα γειτονική κατάσταση (neighborState). Τέλος, επιστρέφει τον νέο πίνακα γειτονικής κατάστασης.

```
Input: Current state currentState of type Integer[]
Output: Neighbor state neighborState of type Integer[]

// Perform a small random change to the current state
neighborState ← Copy of currentState
index1 ← Random integer between 0 and length of neighborState - 1
index2 ← Random integer between 0 and length of neighborState - 1

// Swap two random indices to create a neighboring state
temp ← neighborState[index1]
neighborState[index1] ← neighborState[index2]
neighborState[index2] ← temp

return neighborState
```

Αλγόριθμος 6: Δημιουργία γειτονικής κατάστασης

4.3 Βελτιστοποίηση με σμήνος σωματιδίων / particle swarm optimization

Η βελτιστοποίηση με σμήνος σωματιδίων πρόκειται για μεθόδους που εμπνέονται απο τη φύση, και συγκεκριμένα από τη συρροή των πουλιών. Ανήκει στην κατηγορία των εξελικτικών αλγορίθμων με ιδιαίτερα καλή απόδοση. Οι J. Kennedy και G. Eberhart [41] ήταν οι πρώτοι που τον παρουσίασαν. Σε αντίθεση με άλλους παρόμοιους αλγορίθμους αυτού του είδους, δεν εκτελείται το στάδιο φιλτραρίσματος και έτσι γίνεται διάδοση της πληροφορίας ανάμεσα στα άτομα, ώστε να οδηγηθεί η αναζήτηση στη καλύτερη λύση.

4.3.1 Γενική λειτουργία Σμήνους Σωματιδίων

Στη βελτιστοποίηση με σμήνος σωματιδίων κάθε λύση στο χώρο αναζήτησης αποτελεί ένα σωματίδιο, το οποίο ορίζεται από δύο ιδιότητες, τη θέση x και την ταχύτητα v του. Η θέση προσδιορίζει τη λύση στο πρόβλημα ενώ η ταχύτητα την απόσταση και την κατεύθυνση της αναζήτησης, καθοδηγώντας έτσι το σωματίδιο. Τα σωματίδια στον χώρο αναζήτησης ακολουθούν ως πρότυπα τα καλύτερα σωματίδια. Γενικά, υπάρχουν τοπικά καλύτερα $pbest$ αλλά και ολικά καλύτερα $gbest$ σωματίδια. Το τοπικά καλύτερο σωματίδιο αντιπροσωπεύει την καλύτερη λύση ως τώρα και το ολικά καλύτερο την καλύτερη λύση ως τώρα στο σμήνος.

Υπάρχουν διάφορα μοντέλα για αλγορίθμους βελτιστοποίησης με σμήνος σωματιδίων, οπώς χαρακτηριστικά αναλύουν οι Yuhui Shi και Russell C. Eberhart [42] .

Στη γενικότερη μορφή του ένας αλγόριθμος σμήνους σωματιδίων λειτουργεί όπως φαίνεται στον πίνακα 4.1.

Πίνακας 4.1: Παράδειγμα διαδικασίας αλγορίθμου σμήνους σωματιδίων

Βήμα 1.	Αρχικοποίηση ενός πληθυσμού σωματιδίων με τυχαίες θέσεις και ταχύτητες σε n-διαστάσεις του χώρου του προβλήματος.
Βήμα 2.	Αξιολόγηση της επιθυμητής συνάρτησης καταλληλότητας βελτιστοποίησης σε n μεταβλητές, για κάθε σωματίδιο.
Βήμα 3.	Σύγκριση της καταλληλότητας των τρεχόντων σωματιδίων με αυτή του $pbest$. Αν $pbest < current$, τότε θέσε την τιμή και τη θέση του $pbest$ με αυτή του τρέχοντος σωματιδίου στο χώρο n-διαστάσεων.
Βήμα 4.	Σύγκριση της αξιολογημένης καταλληλότητας των τρεχόντων σωματιδίων με τη συνολική καλύτερη του πληθυσμού. Αν $gbest < current$, τότε θέσε το $gbest$ στο δείκτη και την τιμή του τρέχοντος σωματιδίου στο πίνακα.
Βήμα 5.	Αλλάγη ταχύτητας και θέσης με βάση τις εξισώσεις (1), (2). $v_{in} = v_{in} + c_1 * rand() * (p_{in} - x_{in} + c_2 * rand() * (g_n - x_{in}), (1)$ $x_{in} = x_{in} + v_{in}.(2)$
Βήμα 6.	Επανάληψη από το Βήμα 2 μέχρι να βρεθεί κριτήριο τερματισμού.

Τις ταχύτητες των σωματιδίων σε κάθε διάσταση τις περιορίζουμε σε μια μέγιστη ταχύτητα V_{max}, η οποία είναι σημαντική οντότητα. Αυτό συμβαίνει καθώς επηρεάζει παραμέτρους όπως την καταλληλότητα μεταξύ της θέσης που βρισκόμαστε και εκείνης που θέλουμε να πετύχουμε. Οπότε η τιμή του V_{max} επιλέγεται σε μέσες τιμές δηλαδή όχι πολύ υψηλές αλλά ούτε και πολύ χαμηλές. Με αυτόν τον τρόπο αποφεύγουμε το ενδεχόμενο τα σωματίδια να αγνοούν καλές λύσεις αλλά και εξερευνούν πολύ μικρές περιοχές. Θα μπορούσαμε να πούμε πως ο ρόλος της μέγιστης ταχύτητας είναι να ελέγχει τη συνολική δυνατότητα του σωματιδίου κατά την αναζήτηση.

4.3.2 Βάρος αδράνειας

Όλα όσα αναφέραμε προηγουμένως αφορούν τη γενική μορφή βελτιστοποίησης με σμήνος σωματιδίων. Μια βελτιωμένη μορφή της, την οποία θα υλοποιήσουμε στη συνέχεια, είναι αυτή που συμπεριλαμβάνει το βάρος αδράνειας. Το βάρος αυτό προσφέρει περισσότερο έλεγχο στην εξερεύνηση και στην αξιοποίηση του χώρου αναζήτησης. Με τη χρήση αυτού μπορούμε να εξαλείψουμε την έννοια της μέγιστης ταχύτητας V_{max}, καθώς με τη σωστή επιλογή του βάρους αδράνειας επιτυγχάνεται ισορροπία ανάμεσα στην τοπική και ολική αναζήτηση, το οποίο έχει ως αποτέλεσμα μικρότερους αριθμούς επαναλήψεων.

Η διαδικασία με την οποία λειτουργεί το συγκεκριμένο μοντέλο δεν αλλάζει τελείως σε σχέση με αυτή που παρουσιάσαμε στον πίνακα 4.1. Η μόνη αλλαγή που υπάρχει, βρίσκεται στο βήμα 5 και συγκεκριμένα στην εξίσωση (1). Εφόσον πρέπει να λάβουμε υπόψην μας το βάρος αδράνειας w, η εξίσωση (1) δίνεται ως εξής:

$$\begin{aligned} v_{in} &= w * v_{in} + c_1 * rand() * (p_{in} - x_{in} + c_2 * rand() * (g_n - x_{in}), (3) \\ v_{in} &= (v_{11}, ..., v_{in}), \\ x_{in} &= (x_{11}, ..., x_{in}), \\ p_{in} &= (p_{11}, ..., p_{in}), \\ g_n &= (g_1, ..., g_n). \end{aligned} \tag{4.1}$$

όπου, v_{in} η ταχύτητα του σωματιδίου, x_{in} η θέση του σωματιδίου, p_{in} το καλύτερο τοπικά σωματίδιο και g_n το καλύτερο ολικά σωματίδιο. Τέλος, τα c_1, c_2 αντιπροσωπεύουν τα βάρη της επιτάχυνσης με την οποία τα σωματίδια έλκονται από τα $gbest$ και $pbest$.

4.3.3 Βελτιστοποίηση του JSSP με το σμήνος σωματιδίων

Παρότι η βελτιστοποίηση σμήνους σωματιδίων είναι γνωστή για την αποτελεσματικότητα της στην επίλυση προβλημάτων συνεχούς βελτιστοποίησης, η εφαρμογή της σε προβλήματα όπως το JSSP, που είναι πρόβλημα συνδυασμών, είναι λιγότερο συμβατική. Αυτό συμβαίνει καθώς δεν μπορεί να δωθεί σαν λύση η θέση του σωματιδίου. Οπότε, η αντιπροσώπευση των σωματιδιών πρέπει να γίνει με τέτοιο τρόπο, ώστε να γίνει αντιστοίχιση της θέσης του σωματιδίου με τη λύση στο πρόβλημα και έπειτα να αποκωδικοποιηθεί αυτή και να πάρουμε τη τελική λύση.

Για να το πετύχουμε αυτό, πρέπει να αλληλοσυνδέσουμε σειρές από δείκτες που αντιπροσωπεύουν τις θέσεις των σωματιδίων, με εκείνες των λειτουργιών του JSSP. Για ένα τέτοιο πρόβλημα, με n εργασίες, m μηχανές και $l = n * m$ διεργασίες. Με τις γνωστές συνθήκες του προβλήματος, ότι κάθε εργασία πρέπει να επεξεργαστεί από μια μηχανή και κάθε μια μπορεί να έχει διαφορετική σειρά διεργασιών, ο αλγόριθμος μας θα έχει l-διάσταση. Στον πίνακα, φαίνεται ο τρόπος που οι διεργασίες αντιστοιχούν στις θέσεις των σωματιδίων.

Στη δική μας υλοποίηση για την αναπαράσταση των σωματιδίων εφαρμόσαμε τη μέθοδο OPPS (Operation-Particle Position Sequence). Στο 7 δίνεται ο αλγόριθμος

με τη βασική λειτουργία του PSO μας για το Job-Shop πρόβλημα. Η αρχικοποίηση της θέσης και της ταχήτητας του σωματιδίου γίνεται παίρνοντας μια τυχαία τιμή από x_{min} έως x_{max} για το x_{ij} και από v_{min} έως v_{max} για το v_{ij}. Επιπλέον, το εύρος τιμών $[x_{min}, x_{max}]$ είναι αποδεκτό μόνο για την αρχικοποίηση της θέσης, ενώ το εύρος τιμών $[v_{min}, v_{max}]$ χρησιμοποιείται και κατά τη διάρκεια της εκτέλεσης του PSO, περιορίζοντας την ταχύτητα μέσα σε αυτό.

```
Result: Updated swarm and inertia
begin
    // Update the swarm's global best position and fitness;
    getSwarm().updateGlobalBest();
    // Update all particles in the swarm;
    foreach Particle p in swarm.getParticles() do
        p.update(swarm.getGlobalBestPosition(), inertia, c1, c2, vmin, vmax);
    end
    // Update inertia;
    if inertia > minInertia then
        // Adjusted to spend more time exploring with lower inertia;
        double x = ranIterations/(float)maxIterations;
        inertia = initialInertia - 1.35f * (float) Math.pow(Math.log10(x + 1),
          1.0/4.0) * (initialInertia - minInertia);
        if inertia < minInertia then
            inertia = minInertia;
        end
    end
    ranIterations++;
end
```

Αλγόριθμος 7: Αλγόριθμος σμήνους σωματιδίων

Συνολικά, η κύρια λειτουργία αυτού του τμήματος κώδικα είναι να εκτελέσει μια επανάληψη του αλγορίθμου PSO. Κατά τη διάρκεια κάθε επανάληψης, τα σωματίδια ενημερώνονται με βάση την καλύτερη παγκόσμια θέση και τις προηγούμενες τους θέσεις, ενώ η παράμετρος του βάρους αδράνειας προσαρμόζεται κατάλληλα για να ισορροπήσει την εξερεύνηση και την εκμετάλλευση στον χώρο αναζήτησης. Ας αναλύσουμε λίγο παραπάνω τη διαδικασία ενημέρωσης των σωματιδίων, με σκοπό να κατανοήσουμε πλήρως πως επιλέγεται το καλύτερο με βάση την καταλληλότητα του. Στο 8 δίνεται το τμήμα κώδικα που αντιστοιχεί σε αυτή τη διαδικασία.

Αναλύοντας τον κώδικα, έχουμε τα ακόλουθα βήματα. Ξεκινάει με μια αρχικοποίηση της καλύτερης παγκόσμιας καταλληλότητας σε μία τιμή που εξαρτάται από την τιμή της καλύτερης καταλληλότητας ενός σωματιδίου. Στη συνέχεια, ελέγχει

```
Data: None
Result: Updated global best position and fitness
begin
    foreach Particle p in this.getParticles() do
        if p.getLocalBestFitness() > globalBestFitness then
            globalBestFitness ← p.getLocalBestFitness();
            float[] pos ← p.getLocalBestPosition();
            for j ← 0 to pos.length do
                globalBestPosition[j] ← pos[j];
            end
        end
    end
end
```

Αλγόριθμος 8: Ενημέρωση του καλύτερου σωματιδίου

κάθε σωματίδιο στο σμήνος. Εάν η τοπική καλύτερη καταλληλότητα του σωματιδίου είναι μεγαλύτερη από την τρέχουσα καλύτερη παγκόσμια, τότε ενημερώνεται η καλύτερη παγκόσμια και η αντίστοιχη θέση του σμήνους. Οι ενημερώσεις αυτές πραγματοποιούνται για κάθε σωματίδιο στο σμήνος, εξετάζοντας την τοπική καλύτερη καταλληλότητα και ενημερώνοντας την καλύτερη παγκόσμια ανάλογα. Τέλος, ένα ακόμα σημείο που πρέπει να δώσουμε προσοχή είναι το πως ενημερώνονται τα σωματίδια μέσα στο σμήνος στην εξέλιξη των επαναλήψεων.

```
Data: globalBestPosition, inertia, c1, c2, vmin, vmax
Result: Updated position and velocity
begin
    foreach j in 1 to position.length do
        // Calculate new velocity;
        float v ← inertia * velocity[j] + c1 * alg.random() *
          (localBestPosition[j] - position[j]) + c2 * alg.random() *
          (globalBestPosition[j] - position[j]);
        // Keep velocity in bounds;
        velocity[j] ← Math.min(vmax, Math.max(vmin, v));
        // Update position;
        position[j] ← position[j] + velocity[j];
    end
    fitness.needsUpdating();
    updateLocalBest();
end
```

Αλγόριθμος 9: Ενημέρωση των σωματιδίων

Στο 9 φαίνεται αυτή η διαδικασία. Αυτή η μέθοδος ενημερώνει τη θέση και την ταχύτητα ενός σωματιδίου στο σμήνος βάσει της καλύτερης παγκόσμιας θέσης, του

παράγοντα του βάρους αδράνειας(inertia), των σταθερών c_1 και c_2, καθώς και των ορίων της ταχύτητας (v_{min}, v_{max}). Για κάθε θέση, υπολογίζεται η νέα ταχύτητα με βάση την τρέχουσα ταχύτητα του σωματιδίου, τη διαφορά μεταξύ της καλύτερης τοπικής θέσης και της τρέχουσας θέσης, καθώς και τη διαφορά μεταξύ της καλύτερης παγκόσμιας θέσης και της τρέχουσας θέσης. Η νέα ταχύτητα περιορίζεται στα όρια vmin και vmax και η θέση ενημερώνεται ανάλογα με τη νέα ταχύτητα. Τέλος, ενημερώνεται η τιμή της καταλληλότητας και η καλύτερη τοπική θέση του σωματιδίου.

4.4 Βελτιστοποίηση με αποικία μυρμηγκίων / ant colony optimization

Μια ακόμα τεχνική επιρεασμένη από τη φύση είναι η βελτιστοποιήση με Αποικία Μυρμηγκιών. Εμπνευσμένη από την εντυπωσιακή συμπεριφορά της αναζήτησης τροφής των μυρμηγκιών, η Βελτιστοποίηση με Αποικία Μυρμηγκιών (ACO) έχει αναδειχθεί ως ισχυρή μετα-ευρετική μέθοδος για την επίλυση πολύπλοκων προβλημάτων βελτιστοποίησης. Είναι αλγόριθμοι που μιμούνται τη συλλογική ευφυία των αποικιών μυρμηγκιών και το πως κινούνται από τη βάση τους προς την τροφή δημιουργώντας τα μικρότερα μονοπάτια. Ο πρώτος που πρότεινε αυτό το μοντέλο είναι ο Dorigo [43] τη δεκαετία του 1990 και έκτοτε, η ACO προσφέρει μια συναρπαστική ματιά στις έξυπνες λύσεις της φύσης, φέρνοντας νέες προσεγγίσεις των ερευνητών για την αντιμετώπιση των προβλημάτων βελτιστοποίησης.

4.4.1 Γενικός τρόπος λειτουργίας

Η ACO έχει βρει ευρεία εφαρμογή σε διάφορους τομείς, από τη δρομολόγηση προβλημάτων έως και τη βελτιστοποίηση δικτύων. Όπως αναφέραμε και προηγουμένως, η βασική αρχή της τεχνικής αυτής είναι ο τρόπος με τον οποίο τα μυρμήγκια βρίσκουν τα συντομότερα μονοπάτια από τη φωλιά τους προς την τροφή. Τα μυρμήγκια όταν αναζητούν τροφή σε περιοχή γύρω από την αποικία τους, επιλέγουν τυχαίες κατευθύνσεις με πιθανότητα να βρουν τροφή βάσει της ποσότητας της φερομόνης που αντιλαμβάνονται. Αυτή την ουσία την αφήνουν και τα ίδια τα μυρμήγκια. Έτσι όταν βρίσκουν τροφή, αξιολογούν τόσο την ποιότητα αλλά και την ποσότητα της. Στη διαδικασία της επιστροφής τους πίσω στη φωλιά, αφήνουν ποσότητα φερομόνης ανάλογα με την αξιολόγηση της πηγής τροφής που έχει προηγηθεί. Έτσι,

τα επόμενα μυρμήγκια μπορούν να ακολουθήσουν το ίδιο μονοπάτι και με τη συνεχή αυτή διαδικασία να αυξάνεται η ποσότητα της φερομόνης εώς ότου όλα τα μυρμήγκια να ακολουθούν αυτό το μονοπάτι.

4.4.2 Βελτιστοποιήση του JSSP με αποικία μυρμηγκιών

Για την εφαρμογή αύτης της μεθόδου σε προβήματα χρονοδρομολόγησης, υλοποιήσαμε τον αλγόριθμο να λειτουργεί με τον παρακάτω τρόπο ακολουθώντας τη λογική των Zhou et al. [44]. Με επαναληπτικό ρυθμό ξεκινούν ομάδες από μυρμήγκια-πράκτορες μέσα στον χώρο αναζήτησης, τον οποίο συνήθως αναπαραστά κάποιος γράφος. Κάθε μυρμήγκι επισκέπτεται κάθε κόμβο τον ένα μετά τον άλλο, και με αυτό το τρόπο δημιουργεί ένα μονοπάτι, το οποίο στα πλαίσια του προβλήματος ερμηνεύεται ως ένα δρομολόγιο. Τα μυρμήγκια καθώς προχωράνε από κόμβο σε κόμβο αφήνουν μια ποσότητα φερομόνης, η οποία αλλάζει το περιβάλλον ολικά. Στη συνέχεια, η επόμενη γενιά από μυρμήγκια θα ακολουθήσει μια διαδρομή με βάση την πιθανότητα P_{ig} που δίνεται από την εξίσωση (1) όπου, h είναι ο μετρητής των επαναλήψεων, τ_{ij} είναι η ποσότητα φερομόνης στην ακμή και d_{ij} η ευρετική απόσταση μεταξύ δύο κόμβων. Επιπλέον, τα α και β είναι παράμετροι που ελέγχουν τη σχετική σημασία της φερομόνης και της ευρετικής απόστασης στη λήψη αποφάσεων, αντίστοιχα. Αυτή η πιθανότητα είναι γνωστή ως State Transition Rule, και εξαρτάται τόσο από την απόσταση αλλά και από την ποσότητα της φερομόνης που υπάρχει σε αυτή.

Στη δική μας υλοποίηση επιλέξαμε να παρουσιάσουμε τον χώρο αναζήτησης με τη χρήση ενός πίνακα. Ο πίνακας αυτός αποτελείται από τις διάφορες ποσότητες φερομόνης και ενημερώνεται σε κάθε επανάληψη αλλά και μετά από την εύρεση καλύτερης λύσης. Η διαδικασία της ενημέρωσης δίνεται από τις εξισώσεις (2), (3). Στην εξίσωση (3) να εξηγήσουμε πως το Q αναπαραστά την ποιότητα της φερομόνης ανά μονάδα απόστασης και το είναι ο συντελεστής εξάτμισης. Στο 10 παρουσιάζουμε τη βασική διαδικασία που ακολουθεί ο αλγόριθμος μας με βάση όλα τα προηγούμενα.

$$
\begin{aligned}
p_{ij} &= \frac{(\tau_{ij})^{\alpha} \cdot (\eta_{ij})^{\beta}}{\sum_{k \in allowed\_paths} (\tau_{ik})^{\alpha} \cdot (\eta_{ik})^{\beta}} \quad (1) \\
\tau_{ij}(h+1) &= (1-\rho)\tau_{ij}(h) + \Delta\tau_{ij}(h+1) \quad (2) \\
\Delta\tau_{ij}(h+1) &= \begin{cases} \frac{q}{\text{f\_evaluation(best\_so\_far)}}, & \text{if best\_so\_far is found} \\ 0, & \text{otherwise} \end{cases} \quad (3)
\end{aligned}
\tag{4.2}
$$

Input: Ant Colony *colony*
Output: Updated Ant Colony and Pheromone Matrix

Function `runIteration()`:

```
// Re-generate ants
colony.nextGeneration();
// Update pheromone matrix
colony.updatePheromones();
ranIterations++;
```

Αλγόριθμος 10: Αλγόριθμος Αποικίας Μυρμιγκίων

Αυτή η μέθοδος υλοποιεί τα κύρια βήματα του αλγορίθμου, όπου τα μυρμήγκια κατασκευάζουν λύσεις, οι φερομόνες ενημερώνονται με βάση αυτές τις λύσεις και η διαδικασία επαναλαμβάνεται μέχρι να ικανοποιηθεί μια συνθήκη τερματισμού. Στη συνέχεια, παρουσιάζουμε της δύο βασικές συναρτήσεις που καλούνται σε αυτή τη μέθοδο. Με αυτό τον τρόπο θα έχουμε πλήρη επίγνωση για το πώς δημιουργούνται οι επόμενες γενιές αλλά και πώς ενημερώνεται ο πίνακας φερομόνων.

Στο 11 φαίνεται ο τρόπος με τον οποίο δημιουργείται μια γενία μυρμηγκιών. Η συγκεκριμένη μέθοδος είναι ένα κρίσιμο μέρος της διαδικασίας βελτιστοποίησης. Για κάθε μυρμήγκι της αποικίας, ελέγχει τον αριθμό της επανάληψης και το αν το τρέχον μυρμήγκι είναι το καλύτερο και έπειτα προχωρά στη δημιουργία μιας λύσης. Μετά την πιθανή δημιουργία νέων λύσεων από όλα τα μυρμήγκια, ενημερώνει την καλύτερη λύση που έχει βρεθεί μέχρι στιγμής. Επαναλαμβάνει όλα τα μυρμήγκια ξανά για να βρει το μυρμήγκι με τον καλύτερο χρόνο εκτέλεσης. Εάν ένα μυρμήγκι έχει μικρότερο χρόνο εκτέλεσης από τον καλύτερο μέχρι στιγμής, την ενημερώνει και αντιγράφει τις προγραμματισμένες λειτουργίες του μυρμηγκιού στον πίνακα που περιέχει τα καλύτερα μυρμήγκια μέχρι εκείνη τη στιγμή. Σε συνέχεια αυτής της μεθόδου στο 12 δίνεται το κομμάτι κώδικα όπου δημιουργείται μια λύση. Αρχικά, αρχικοποιεί διάφορες μεταβλητές και πίνακες που χρησιμοποιούνται για τον προγραμματισμό με βάση τις πληροφορίες του εκάστοτε προβλήματος, για παράδειγμα

```
Input: Array of Ants ants
Output: Updated Ants with Generated Solutions
Function nextGeneration():
    for i ← 0 to ants.length do
        if alg.getRanIterations() == 0 or ants[i] ≠ getBestAnt() then
            ants[i].generate();
        end
    end
    // Update the best solution so far
    for a in this.getAnts() do
        if a.getMakespan() < bestMakespanSoFar then
            bestMakespanSoFar ← a.getMakespan();
            for i ← 0 to a.getScheduledOperations().length do
                bestSoFar[i] ← a.getScheduledOperations()[i];
            end
        end
    end
```

Αλγόριθμος 11: Next Generation

των συνολικό αριθμό λειτουργιών.Στη συνέχεια, ορίζει μια τυχαία πιθανότητα αγνόησης των φερομονών. Το όριο ανέρχεται στο 5% (0,05f). Έπειτα, επιλέγει την πρώτη λειτουργία με βάση το αν θα αγνοήσει τις φερομόνες ή όχι. Μέσα σε έναν βρόγχο, συνεχίζει να επιλέγει την επόμενη λειτουργία και ενημερώνει το χρονοδιάγραμμα μέχρι να προγραμματιστούν όλες οι λειτουργίες. Για κάθε επιλεγμένη λειτουργία, ενημερώνει το χρονοδιάγραμμα, τις συνδέσεις και υπολογίζει το συνολικό χρόνο εκτέλεσης (makespan). Τελικά, ο βρόγχος τερματίζεται όταν όλες οι λειτουργίες προγραμματιστούν (το scheduleIndex φτάνει το συνολικό αριθμό των λειτουργιών).

Για την επιλογή των λειτουργιών χρησιμοποιούμε ακόμα μια συνάρτηση που αποτελεί κρίσιμο μέρος του αλγορίθμου μας. Η συνάρτηση λειτουργεί ως εξής: καθορίζει πρώτα τον κατάλογο των λειτουργιών που είναι επί του παρόντος προσβάσιμες ή επιλέξιμες για χρονοπρογραμματισμό. Εάν υπάρχει μόνο μία προσβάσιμη λειτουργία, επιστρέφει άμεσα την εν λόγω λειτουργία. Σε άλλη περίπτωση, συνεχίζει για κάθε προσβάσιμη λειτουργία, υπολογίζει μια τιμή πιθανότητας με βάση τα επίπεδα φερομόνης, τις αποστάσεις και τις παρεχόμενες παραμέτρους alpha και beta. Στη συνέχεια, χρησιμοποιεί έναν μηχανισμό επιλογής ρουλέτας για να επιλέξει την επόμενη λειτουργία με βάση τις υπολογισμένες πιθανότητες. Αυτή η μέθοδος επιλογής, δίνει μεγαλύτερες πιθανότητες στις λειτουργίες με υψηλότερα σκορ, ενώ παράλληλα επιτρέπει κάποια τυχαιότητα. Τέλος, επιστρέφει την επιλεγμένη

Input: Input algorithm alg
Output: Schedule

$pb \leftarrow alg.getProblemInstance()$;
Initialize variables;
$makespan \leftarrow 0$;
$scheduleIndex \leftarrow 0$;
Create array $scheduledOperations$ with size $pb.getTotalOperations()$;
Create array $connections$ with size $pb.getTotalOperations() + 1$;
Create array $lastJobOperation$ with size $pb.getNumberOfJobs()$;
for i *from 0 to* $pb.getNumberOfJobs() - 1$ **do**
  $lastJobOperation[i] \leftarrow -1$;
end
Random chance of ignoring pheromones $ignorePheromones \leftarrow$
`alg.getRandom().nextFloat() < 0.05f;`
Choose first operation $operation \leftarrow$
`chooseNextOperation(ignorePheromones);`
$connections[0] \leftarrow operation + 1$;
repeat
  $temp \leftarrow operation$;
  Choose next operation $operation \leftarrow$
  `chooseNextOperation(ignorePheromones);`
  $connections[temp + 1] \leftarrow operation + 1$;
  $scheduledOperations[scheduleIndex++] \leftarrow operation$;
  $lastJobOperation[operation/pb.getOperationsPerJob()] \leftarrow operation$;
  Compute makespan $makespan \leftarrow$
  `alg.computeMakespan(scheduledOperations);`
until $scheduleIndex \geq scheduledOperations.length$;

Αλγόριθμος 12: Μέθοδος δημιουργίας πιθανής λύσης

λειτουργία. Στο 13 παρουσιάζουμε και τον αντίστοιχο κώδικα της συγκεκριμένης συνάρτησης.

Input: Boolean $ignorePheromones$
Output: Next operation to choose

Compute eligible operations;
$accessibleOperations \leftarrow$ getAccessibleOperations();
if $accessibleOperations.size == 1$ **then**
  return $accessibleOperations.get(0)$;
end
$pb \leftarrow alg.getProblemInstance()$;
$\alpha \leftarrow alg.getAlpha(), \beta \leftarrow alg.getBeta()$;
Create array $probabilities$ with size $pb.getTotalOperations()$;
$currentNode \leftarrow$ `scheduleIndex == 0 ? 0 : (1 + scheduledOperations[scheduleIndex-1])`;
$denominator \leftarrow 0$;
for k *in* $accessibleOperations$ **do**
  $denominator \leftarrow denominator+$ (ignorePheromones ? 1 :
  Math.pow($alg.getColony()$.getPheromones($currentNode, k+1), \alpha$)) /
  Math.pow(distance$(k) + 0.5, \beta$);
end
for j *in* $accessibleOperations$ **do**
  $probabilities[j] \leftarrow$ (ignorePheromones ? 1 :
  Math.pow($alg.getColony()$.getPheromones($currentNode, j+1), \alpha$)) /
  Math.pow(distance$(j) + 0.5, \beta$) / $denominator$;
end
$operation \leftarrow$ `RouletteWheel.spinOnce(alg.getRandom(), probabilities);`
return $operation$;

Αλγόριθμος 13: Μέθοδος επιλογής Λειτουργίας

Κεφάλαιο 5

Αποτελέσματα

Σε αυτό το κεφάλαιο θα παρουσιάσουμε τα αποτελέσματα, τα οποία παράχθηκαν από τους κώδικες που χρησιμοποιήσαμε για την επίλυση των προβλημάτων. Η δημιουργία και εκτέλεση των 4 αλγορίθμων έγινε με το προγραμματιστικό περιβάλλον NetBeans IDE 15 και με τη γλώσσα προγραμματισμού Java. Τους αλγορίθμους τους τρέξαμε σε λειτουργικό σύστημα Windows 10x64bit. Για την υπολογιστική μας μελέτη επιλέξαμε 128 βασικά προβλήματα αναφοράς των Lawrence, Adam-Balas-Zawack, Muth-Thompson και Taillard, τα οποία καλύπτουν μεγάλο εύρος προβλημάτων Job Shop. Από μικρά προβλήματα με μικρό αριθμό εργασιών-μηχανών, όπως για παράδειγμα το πολύ γνωστό MT10 (10x10) αλλά και μεγαλύτερα προβλήματα κυρίως των Lawrence και Taillard. Επιπλέον, χρησιμοποιώντας τα απαραίτητα δεδομένα του Taillard για τη δημιουργία τυχαίων μεγάλων προβλημάτων, κατασκευάσαμε τα 80 προβλήματα τα οποία κυμαίνονται από 15 εώς 100 εργασίες και από 15 εώς 20 μηχανές. Τα προβλήματα που χρησιμοποιήσαμε για τη μελέτη μας τα αποκτήσαμε από τη δωρεάν διαδικτυακή βιβλιοθήκη OR-Library `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/jobshop1.txt` και `http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html`, ενώ τον αλγόριθμο για τη δημιουργία των τυχαίων προβλημάτων από τον σύνδεσμο `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/jobshop2.txt`.

Για την επίλυση των αρχείων των προβλημάτων θεσπίσαμε ένα χρονικό όριο μιας ώρας για την εύρεση κάποιου αποτελέσματος. Σε περίπτωση που κάποιος αλγόριθμος δεν επέλυε το πρόβλημα εντός αυτού του ορίου, τότε έδινε σαν αποτέλεσμα είτε τη λύση που είχε μέχρι εκείνη τη στιγμή είτε δεν επέστρεφε καμία λύση. Τον ακριβή αλγόριθμο B&B για κάθε πρόβλημα τον τρέξαμε από μια φορά

καθώς είναι ντετερμινιστικός και κάθε φορά επιστρέφει το ίδιο αποτέλεσμα. Τους 3 ευρετικούς αλγορίθμους ACO, PSO και SA, τους τρέξαμε για κάθε πρόβλημα από 10 φορές κρατώντας τον μέσο όρο της καλύτερης λύσης αλλά και τον μέσο όρο του χρόνου που χρειάστηκε κάθε επανάληψη για να ολοκληρωθεί. Επιπλέον για τους ACO, PSO και SA που είναι εξελικτικοί αλγόριθμοι, θέσαμε τις εσωτερικές επαναλήψεις (Iterations) στις 2000 και τον αριθμό των εποχών (epochs) στις 200. Για τις ξεχωριστές παραμέτρους των αλγορίθμων, τις ορίσαμε ως εξής:

Για τον ACO: Το μέγεθος της αποικίας ColonySize = 30, Q = 2.0f, rho = 0.025, a = 10.0f, b = 8.0f και τον αριθμό της αρχικής φερομόνης InitPheromone = 0.5f. Για τον PSO: το μέγεθος του σμήνους SwarmSize = 40, το αρχικό βάρος αδράνειας InitInertia = 1.2f, το χαμηλότερο βάρος αδράνειας minInertia = 0.3f, (c1,c2) = 2.0f, το έυρος τιμών [Xmin, Xmax] = [0.0f, 6.0f] και το εύρος τιμών [Vmin, Vmax] = [-6.0f, 6.0f]. Για τον SA: την αρχικη θερμοκρασία InitTemp = 10000 και τον ρυθμό ψύξης CR = 0.05f.

Ο πρώτος πίνακας που θα δούμε περιέχει τα δεδομένα των προβλημάτων, ο δεύτερος περιέχει τα αποτελέσματα με τα καλύτερα Cmax (makespan) που βρήκε ο κάθε αλγόριθμος, ο τρίτος περιέχει το gap των αλγόριθμων για κάθε πρόβλημα προς επίλυση και ο τέταρτος τα αποτελέσματα των χρόνων που χρειάστηκε κάθε ένας για να λύσει το πρόβλημα. Πάνω στο δεύτερο και τρίτο πίνακα θα γίνει και η σύγκριση των αποτελεσμάτων, για να αποφασίσουμε ποιός είναι τελικά ο πιο αποδοτικός αλγόριθμος ως προς την επίτευξη της βέλτιστης λύσης και τον χρόνο.

Στον Πίνακα 5.1 δίνονται τα δεδομένα ενός προβλήματος. Περίεχει δύο στήλες, όπου η πρώτη στήλη περιέχει το όνομα του προβλήματος που μας δείχνει ποιο πρόβλημα επιλύουμε. Ενώ η δεύτερη την καλύτερη λύση που έχει βρεθεί για το πρόβλημα, που αντιστοιχεί στο μικρότερο χρόνο ολοκλήρωσης όλων των εργασιών C_{max} (makespan).

Πίνακας 5.1: Πίνακας δεδομένων των προβλημάτων

Number	**Problem**	**Makespan**
1	MT06 (6X6)	55
2	MT10 (10X10)	930
Συνέχεια σε επόμενη σελίδα		

Πίνακας 5.1 – Συνέχεια από προηγούμενη σελίδα

Number	Problem	Makespan
3	MT20 (20x5)	1165
4	ABZ5 (10X10)	1234
5	ABZ6 (10X10)	651
6	ABZ7 (20x15)	651
7	ABZ8 (20x15)	627
8	ABZ9 (20x15)	650
9	LA01 (10X5)	666
10	LA02 (10X5)	655
11	LA03 (10X5)	597
12	LA04 (10X5)	590
13	LA05 (10X5)	593
14	LA06 (15X5)	926
15	LA07 (15X5)	890
16	LA08 (15X5)	863
17	LA09 (15X5)	951
18	LA10 (15X5)	958
19	LA11 (20X5)	1222
20	LA12 (20X5)	1039
21	LA13 (20X5)	1150
22	LA14 (20x5)	1292
23	LA15 (20x5)	1207
24	LA16 (10x10)	945
25	LA17 (10x10)	784
26	LA18 (10x10)	848
27	LA19 (10x10)	842
28	LA20 (10x10)	902
29	LA21 (15x10)	1040
30	LA22 (15x10)	927

Συνέχεια σε επόμενη σελίδα

Πίνακας 5.1 – Συνέχεια από προηγούμενη σελίδα

Number	Problem	Makespan
31	LA23 (15x10)	1032
32	LA24 (15x10)	935
33	LA25 (15x10)	977
34	LA26 (20x10)	1218
35	LA27 (20x10)	1235
36	LA28 (20x10)	1216
37	LA29 (20x10)	1120
38	LA30 (20x10)	1355
39	LA31 (30x10)	1784
40	LA32 (30x10)	1850
41	LA33 (30x10)	1719
42	LA34 (30x10)	1721
43	LA35 (30x10)	1888
44	LA36 (15x15)	1268
45	LA37 (15x15)	1397
46	LA38 (15x15)	1171
47	LA39 (15x15)	1233
48	LA40 (15x15)	1233
49-53	TA01-TA05 (15x15)	1231
54-58	TA06-TA10 (15x15)	1238
59-63	TA11-TA15 (20x15)	1323-1357
64-68	TA16-TA20 (20x15)	1304-1360
69-73	TA21-TA25 (20x20)	1573-1642
74-78	TA26-TA30 (20x20)	1558-1643
79-83	TA31-TA35 (30x15)	1764
84-88	TA36-TA40 (30x15)	1819
89-93	TA41-TA45 (30x20)	1876-2005
94-98	TA46-TA50 (30x20)	1940-2004

Συνέχεια σε επόμενη σελίδα

Πίνακας 5.1 – Συνέχεια από προηγούμενη σελίδα

Number	Problem	Makespan
99-103	TA51-TA55 (50x15)	2760
104-108	TA56-TA60 (50x15)	2781
109-113	TA61-TA65 (50x20)	2868
114-118	TA66-TA70 (50x20)	2845
119-123	TA71-TA75 (100x20)	5464
124-128	TA76-TA80 (100x20)	5342

Ο Πίνακας 5.2 περιέχει τα αποτελέσματα των αλγορίθμων ως προς την καλύτερη λύση που βρήκαν στο πρόβλημα. Αποτελείται από έξι στήλες, όπου η πρώτη αντιστοιχεί στο πρόβλημα προς επίλυση, η δεύτερη στην καλύτερη λύση του κάθε προβλήματος, η τρίτη στον ακριβή αλγόριθμο Branch and Bound (B&B), η τέταρτη στον αλγόριθμο βελτιστοποίησης με αποικία μυρμηγκιών (ACO), η πέμπτη στον αλγόριθμο σμήνους σωματιδίων (PSO) και η έκτη στη μέθοδο του προσομοιωμένου ανορθωτή (SA).

Πίνακας 5.2: Πίνακας αποτελεσμάτων των αλγορίθμων

Problem	Best Makespan	B&B	ACO	PSO	SA
	Makespan	Makespan	Makespan	Makespan	Makespan
MT06(6x6)	55	55	57.7	58.8	60.5
MT10(10x10)	930	1150	1151.8	1090.1	1229.8
MT20(20x5)	1165	1650	1308.5	1419.3	1395.1
ABZ5(10x10)	1234	1364	1342.7	1360.4	1361.0
ABZ6(10x10)	651	1005	1009.0	987.7	1023.0
ABZ7(20x15)	651	798	857.3	821.2	821.7
ABZ8(20x15)	627	816	913.5	843.3	853.8
ABZ9(20x15)	650	860	913.0	868.1	873.7
LA01(10x5)	666	668	672.2	666	669.9
LA02(10x5)	655	785	676.7	712.6	711.2
LA03(10x5)	597	726	627.9	640.8	647.9

Συνέχεια σε επόμενη σελίδα

Πίνακας 5.2 – Συνέχεια από προηγούμενη σελίδα

Problem	**Best Makespan**	**B&B**	**ACO**	**PSO**	**SA**
	Makespan	Makespan	Makespan	Makespan	Makespan
LA04(10x5)	590	677	609.0	611.5	619.2
LA05(10x5)	593	593	593.0	593.0	593.0
LA06(15x5)	926	964	926.0	926.0	926.0
LA07(15x5)	890	1076	895.3	901.2	898.8
LA08(15x5)	863	972	865.6	863.0	867.9
LA09(15x5)	951	951	951.0	951.0	953.4
LA10(15x5)	958	958	958.0	958.0	958.0
LA11(20x5)	1222	1222	1222.0	1222.0	1222.0
LA12(20x5)	1039	1039	1039.0	1039.0	1039.0
LA13(20x5)	1150	1171	1150.9	1150.0	1150.0
LA14(20x5)	1292	1292	1292.0	1292.0	1292.0
LA15(20x5)	1207	1587	1220.5	1222.8	1257.2
LA16(10x10)	945	1064	1014.8	1039.3	1017.3
LA17(10x10)	784	835	813.2	834.8	828.0
LA18(10x10)	848	932	909.9	931.8	923.7
LA19(10x10)	842	945	909.3	935.2	927.0
LA20(10x10)	902	1068	961.0	994.8	977.9
LA21(15x10)	1040	1348	1222.9	1182.0	1201.1
LA22(15x10)	927	1226	1122.6	1054.5	1091.5
LA23(15x10)	1032	1174	1120.9	1130.1	1085.2
LA24(15x10)	935	1095	1080.0	1085.0	1085.2
LA25(15x10)	977	1221	1146.9	1119.0	1115.1
LA26(20x10)	1218	1369	1400.5	1390.4	1414.3
LA27(20x10)	1235	1535	1473.2	1432.3	1458.8
LA28(20x10)	1216	1436	1408.5	1405.5	1436.4
LA29(20x10)	1120	1449	1445.1	1399.7	1425.5
LA30(20x10)	1355	1612	1557.8	1546.8	1599.3

Συνέχεια σε επόμενη σελίδα

Πίνακας 5.2 – Συνέχεια από προηγούμενη σελίδα

Problem	**Best Makespan**	**B&B**	**ACO**	**PSO**	**SA**
	Makespan	Makespan	Makespan	Makespan	Makespan
LA31(30x10)	1784	1940	1851.3	1922.2	1923.4
LA32(30x10)	1850	2081	1940.6	1999.0	2038.7
LA33(30x10)	1719	1863	1790.3	1864.2	1871.6
LA34(30x10)	1721	1969	1874.0	1899.0	1940.5
LA35(30x10)	1888	2081	2006.6	2054.7	2077.5
LA36(15x15)	1268	1512	1672.6	1457.2	1496.2
LA37(15x15)	1397	1672	1709.8	1673.1	1673.2
LA38(15x15)	1171	1471	1536.7	1423.3	1452.8
LA39(15x15)	1233	1440	1565.5	1480.5	1487.3
LA40(15x15)	1233	1369	1570.5	1418.6	1494.7
TA01(15x15)	1231	1461	1588.4	1446.0	1485.9
TA02(15x15)	1231	1491	1398.3	1443.0	1491.8
TA03(15x15)	1231	1482	1618.2	1439.0	1478.1
TA04(15x15)	1231	1475	1568.8	1417.2	1481.1
TA05(15x15)	1231	1567	1631.1	1438.2	1517.2
TA06(15x15)	1238	1501	1563.8	1452.2	1465.3
TA07(15x15)	1238	1465	1583.5	1428.6	1470.8
TA08(15x15)	1238	1437	1577.7	1442.8	1500.0
TA09(15x15)	1238	1527	1673.8	1501.1	1533.5
TA10(15x15)	1238	1478	1650.7	1446.0	1473.2
TA11(20x15)	1323-1357	1680	1697.4	1713.8	1755.0
TA12(20x15)	1323-1357	1666	1905.0	1736.0	1754.7
TA13(20x15)	1323-1357	1732	1904.6	1685.1	1700.3
TA14(20x15)	1323-1357	1594	1775.0	1645.7	1685.1
TA15(20x15)	1323-1357	1664	1845.6	1724.6	1756.5
TA16(20x15)	1304-1360	1688	1898.6	1709.9	1733.2
TA17(20x15)	1304-1360	1868	1911.3	1776.4	1808.0

Συνέχεια σε επόμενη σελίδα

Πίνακας 5.2 – Συνέχεια από προηγούμενη σελίδα

Problem	**Best Makespan**	**B&B**	**ACO**	**PSO**	**SA**
	Makespan	Makespan	Makespan	Makespan	Makespan
TA18(20x15)	1304-1360	1731	1909.6	1748.1	1771.6
TA19(20x15)	1304-1360	1732	1868.3	1698.7	1724.0
TA20(20x15)	1304-1360	1643	1931.3	1721.3	1751.2
TA21(20x20)	1573-1642	2023	2450.5	2130.6	2141.6
TA22(20x20)	1573-1642	2000	2411.5	2056.3	2151.5
TA23(20x20)	1573-1642	1944	2381.5	2055.1	2037.1
TA24(20x20)	1573-1642	1984	2375.5	2094.8	2089.1
TA25(20x20)	1573-1642	2040	2499.0	2058.9	2078.7
TA26(20x20)	1558-1643	2038	2428.5	2139.6	2119.1
TA27(20x20)	1558-1643	1971	2420.5	2097.0	2098.6
TA28(20x20)	1558-1643	2014	2428.0	2103.0	2147.1
TA29(20x20)	1558-1643	1945	2419.0	2077.6	2083.5
TA30(20x20)	1558-1643	2091	2464.0	2110.3	2133.0
TA31(30x15)	1764	2226	2406.0	2355.0	2346.4
TA32(30x15)	1764	2366	2479.0	2428.2	2427.0
TA33(30x15)	1764	2405	2510.0	2453.0	2429.8
TA34(30x15)	1764	2334	2436.0	2371.0	2400.6
TA35(30x15)	1764	2443	2464.0	2482.1	2425.9
TA36(30x15)	1819	2538	2441.0	2389.5	2450.7
TA37(30x15)	1819	2204	2344.0	2391.7	2410.4
TA38(30x15)	1819	2184	2308.0	2273.0	2307.2
TA39(30x15)	1819	2280	2517.0	2402.8	2390.4
TA40(30x15)	1819	2117	2275.0	2250.3	2255.5
TA41(30x20)	1876-2005	2568	3231.0	2882.0	2866.1
TA42(30x20)	1876-2005	2578	3083.0	2823.1	2765.1
TA43(30x20)	1876-2005	2444	3024.0	2747.4	2673.1
TA44(30x20)	1876-2005	2545	3172.0	2878.4	2886.1

Συνέχεια σε επόμενη σελίδα

Πίνακας 5.2 – Συνέχεια από προηγούμενη σελίδα

Problem	**Best Makespan**	**B&B**	**ACO**	**PSO**	**SA**
	Makespan	Makespan	Makespan	Makespan	Makespan
TA45(30x20)	1876-2005	2480	2947.0	2881.4	2819.2
TA46(30x20)	1940-2004	2656	3316.0	2937.5	2847.1
TA47(30x20)	1940-2004	2476	3026.0	2661.0	2730.8
TA48(30x20)	1940-2004	2474	3104.0	2776.9	2694.8
TA49(30x20)	1940-2004	2693	3004.0	2733.9	2742.0
TA50(30x20)	1940-2004	2515	3034.0	2820.6	2792.7
TA51(50x15)	2760	-	3507.0	3710.0	3624.4
TA52(50x15)	2760	-	3548.8	5657.0	3643.4
TA53(50x15)	2760	-	3176.0	3458.4	3427.4
TA54(50x15)	2760	-	3444.0	3575.0	3429.3
TA55(50x15)	2760	-	3348.0	3634.9	3569.3
TA56(50x15)	2781	-	3501.0	3647.4	3563.1
TA57(50x15)	2781	-	3740.0	3777.0	3632.9
TA58(50x15)	2781	-	3567.0	3714.1	3651.9
TA59(50x15)	2781	-	3320.0	3559.8	3515.2
TA60(50x15)	2781	-	3308.0	3546.9	3453.3
TA61(50x20)	2868	-	4249.0	4020.4	4003.8
TA62(50x20)	2868	-	4451.0	4220.3	4111.2
TA63(50x20)	2868	-	3993.0	4088.6	3852.2
TA64(50x20)	2868	-	4024.0	3934.9	3848.3
TA65(50x20)	2868	-	4101.0	4043.4	3932.4
TA66(50x20)	2845	-	-	4121.8	4001.3
TA67(50x20)	2845	-	-	4200.2	4065.6
TA68(50x20)	2845	-	-	4081.1	3897.9
TA69(50x20)	2845	-	-	4217.5	4121.6
TA70(50x20)	2845	-	-	4189.0	4131.4
TA71(100x20)	5464	-	-	7569.3	7268.5

Συνέχεια σε επόμενη σελίδα

Πίνακας 5.2 – Συνέχεια από προηγούμενη σελίδα

Problem	**Best Makespan**	**B&B**	**ACO**	**PSO**	**SA**
	Makespan	Makespan	Makespan	Makespan	Makespan
TA72(100x20)	5464	-	-	7341.1	6848.9
TA73(100x20)	5464	-	-	7555.5	7179.7
TA74(100x20)	5464	-	-	7387.7	6938.3
TA75(100x20)	5464	-	-	7521.6	7198.3
TA76(100x20)	5342	-	-	7596.3	6999.5
TA77(100x20)	5342	-	-	7582.2	7121.1
TA78(100x20)	5342	-	-	7502.7	6993.1
TA79(100x20)	5342	-	-	7578.7	7000.8
TA80(100x20)	5342	-	-	7355.1	6865.9

Στον πίνακα των αποτελεσμάτων των αλγόριθμων που παράχθηκαν παρατηρούμε ότι ο ακριβής αλγόριθμος (B&B) και ο ACO σε ορισμένα προβλήματα δεν κατάφεραν να βρουν κάποια λύση. Αυτά τα αποτελέσματα αντιπροσωπεύονται με τη χρήση της παύλας (-). Επίσης παρατηρούμε πως οι άλλοι δύο ευρετικοί αλγόριθμοι, δηλαδή οι PSO και SA, έλυσαν όλα τα προβλήματα. Συγκεντρωτικά, για τα αποτελέσματα των τεσσάρων αλγόριθμων στο χρονικό όριο της μίας ώρας παρατηρήσαμε ότι:

- Η μέθοδος B&B επέστρεψε λύση σε 98 προβλήματα.
- Η μέθοδος ACO επέστρεψε λύση σε 113 προβλήματα.
- Η μέθοδος PSO επέστρεψε λύση σε 128 προβλήματα.
- Η μέθοδος SA επέστρεψε λύση σε 128 προβλήματα.

Από τη παραπάνω ανάλυση μπορούμε να διακρίνουμε ότι οι μέθοδοι PSO και SA ήταν πολύ αποδοτικοί εφόσον λύσανε όλα τα προβλήματα, ένω οι άλλοι δύο δεν απέδωσαν ανάλογα. Επιπλέον, ένα ακόμα χαρακτηριστικό των PSO, SA είναι πως για κανένα πρόβλημα δεν πλησίασαν το χρονικό όριο. Αντιθέτως, και ο B&B και ο ACO στα περισσότερα προβλήματα χρειάστηκαν όλο τον χρόνο για την εύρεση ή τη μη εύρεση κάποιας λύσης. Ακόμη, υπήρξαν και τριάντα προβλήματα για τα οποία ο

ακριβής αλγόριθμος δεν βρήκε κάποια λύση στο διάστημα της μίας ώρας ενώ, από αυτά ο ACO δεν έλυσε τα δεκαπέντε. Τα προβλήματα αυτά είναι τα (TA51-TA80) και (TA65-TA80) αντίστοιχα. Ο λόγος για τον οποίο αυτές οι δύο μέθοδοι δεν έδωσαν καμία λύση, μπορεί να είναι είτε επειδή ο αριθμός των εργασιών-μηχανών ήταν πολύ μεγάλος, είτε επειδή οι χρόνοι επεξεργασίας ήταν πολύ μεγάλοι. Συνεπώς, μπορούμε να διαπιστώσουμε πως ο παράγοντας του χρόνου επηρεάζει σε μεγάλο βαθμό την παραγωγή αποτελεσμάτων. Πιθανόν, στη περίπτωση που το χρονικό όριο ήταν μεγαλύτερο της μίας ώρας, οι μέθοδοι αυτές να είχαν καλύτερη απόδοση λύνοντας περισσότερα προβλήματα αλλά και βελτιώνοντας τις ήδη υπάρχουσες λύσεις.

Ο Πίνακας 5.3 περιέχει τα gaps των αποτελεσμάτων για τους τέσερις αλγόριθμους. Το gap είναι το ποσοστό απόστασης από τη βέλτιστη λύση και δίνεται ως $(|Makespan(Solution) - OptimalMakespan|/OptimalMakespan) * 100\%$. Ο πίνακας των gap λοιπόν αποτελείται από πέντε στήλες, όπου έχει ως πρώτη το πρόβλημα προς λύση μεθόδων, ενώ οι υπόλοιπες τέσσερις το gap των μεθόδων. Συγκεκριμένα, στη δεύτερη είναι το gap του B&B, στην τρίτη το gap του ACO, στη τέταρτη το gap του PSO και στην πέμπτη το gap του SA.

Πίνακας 5.3: Πίνακας GAP των αλγορίθμων

Problem	**B&B**	**ACO**	**PSO**	**SA**
	Gap(%)	Gap(%)	Gap(%)	Gap(%)
MT06(6x6)	0.00%	4.91%	6.91%	10.00%
MT10(10x10)	23.66%	23.85%	17.22%	32.24%
MT20(20x5)	41.63%	12.32%	21.83%	19.75%
ABZ5(10x10)	10.53%	8.81%	10.24%	10.29%
ABZ6(10x10)	54.38%	54.99%	51.72%	57.14%
ABZ7(20x15)	22.58%	31.69%	26.14%	26.22%
ABZ8(20x15)	30.14%	45.69%	34.50%	36.17%
ABZ9(20x15)	32.31%	40.46%	33.55%	34.42%
LA01(10x5)	0.30%	0.93%	0.00%	0.59%
LA02(10x5)	19.85%	3.31%	8.79%	8.58%
LA03(10x5)	21.61%	5.18%	7.34%	8.53%
				Συνέχεια σε επόμενη σελίδα

Πίνακας 5.3 – Συνέχεια από προηγούμενη σελίδα

Problem	**B&B**	**ACO**	**PSO**	**SA**
	Gap(%)	Gap(%)	Gap(%)	Gap(%)
LA04(10x5)	14.75%	3.22%	3.64%	4.95%
LA05(10x5)	0.00%	0.00%	0.00%	0.00%
LA06(15x5)	4.10%	0.00%	0.00%	0.00%
LA07(15x5)	20.90%	0.60%	1.26%	0.99%
LA08(15x5)	12.63%	0.30%	0.00%	0.57%
LA09(15x5)	0.00%	0.00%	0.00%	0.25%
LA10(15x5)	0.00%	0.00%	0.00%	0.00%
LA11(20x5)	0.00%	0.00%	0.00%	0.00%
LA12(20x5)	0.00%	0.00%	0.00%	0.00%
LA13(20x5)	1.83%	0.08%	0.00%	0.00%
LA14(20x5)	0.00%	0.00%	0.00%	0.00%
LA15(20x5)	31.48%	1.12%	1.31%	4.16%
LA16(10x10)	12.59%	7.39%	9.98%	7.65%
LA17(10x10)	6.51%	3.72%	6.48%	5.61%
LA18(10x10)	9.91%	7.30%	9.88%	8.93%
LA19(10x10)	12.23%	7.99%	11.07%	10.10%
LA20(10x10)	18.40%	6.54%	10.29%	8.41%
LA21(15x10)	29.62%	17.59%	13.65%	15.49%
LA22(15x10)	32.25%	21.10%	13.75%	17.75%
LA23(15x10)	13.76%	8.61%	9.51%	5.16%
LA24(15x10)	17.11%	15.51%	16.04%	16.06%
LA25(15x10)	24.97%	17.39%	14.53%	14.14%
LA26(20x10)	12.40%	14.98%	14.15%	16.12%
LA27(20x10)	24.29%	19.29%	15.98%	18.12%
LA28(20x10)	18.09%	15.83%	15.58%	18.13%
LA29(20x10)	29.38%	29.03%	24.97%	27.28%
LA30(20x10)	18.97%	14.97%	14.15%	18.03%

Συνέχεια σε επόμενη σελίδα

Πίνακας 5.3 – Συνέχεια από προηγούμενη σελίδα

Problem	**B&B**	**ACO**	**PSO**	**SA**
	Gap(%)	Gap(%)	Gap(%)	Gap(%)
LA31(30x10)	8.74%	3.77%	7.75%	7.81%
LA32(30x10)	12.49%	4.90%	8.05%	10.20%
LA33(30x10)	8.38%	4.15%	8.45%	8.88%
LA34(30x10)	14.41%	8.89%	10.34%	12.75%
LA35(30x10)	10.22%	6.28%	8.83%	10.04%
LA36(15x15)	19.24%	31.91%	14.92%	18.00%
LA37(15x15)	19.69%	22.39%	19.76%	19.77%
LA38(15x15)	25.62%	31.23%	21.55%	24.06%
LA39(15x15)	16.79%	26.97%	20.07%	20.62%
LA40(15x15)	11.03%	27.37%	15.05%	21.22%
TA01(15x15)	18.68%	29.03%	17.47%	20.71%
TA02(15x15)	21.12%	13.59%	17.22%	21.19%
TA03(15x15)	20.39%	31.45%	16.90%	20.07%
TA04(15x15)	19.82%	27.44%	15.13%	20.32%
TA05(15x15)	27.29%	32.50%	16.83%	23.25%
TA06(15x15)	21.24%	26.32%	17.30%	18.36%
TA07(15x15)	18.34%	27.91%	15.40%	18.80%
TA08(15x15)	16.07%	27.44%	16.54%	21.16%
TA09(15x15)	23.34%	35.20%	21.25%	23.87%
TA10(15x15)	19.39%	33.34%	16.80%	19.00%
TA11(20x15)	23.80%	25.08%	26.29%	29.33%
TA12(20x15)	22.77%	40.38%	27.93%	29.31%
TA13(20x15)	27.63%	40.35%	24.18%	25.30%
TA14(20x15)	17.46%	30.80%	21.27%	24.18%
TA15(20x15)	22.62%	36.01%	27.09%	29.44%
TA16(20x15)	24.12%	39.60%	25.73%	27.44%
TA17(20x15)	37.35%	40.54%	30.62%	32.94%

Συνέχεια σε επόμενη σελίδα

Πίνακας 5.3 – Συνέχεια από προηγούμενη σελίδα

Problem	**B&B**	**ACO**	**PSO**	**SA**
	Gap(%)	Gap(%)	Gap(%)	Gap(%)
TA18(20x15)	27.28%	40.41%	28.54%	30.26%
TA19(20x15)	27.35%	37.38%	24.90%	26.76%
TA20(20x15)	20.81%	42.01%	26.57%	28.76%
TA21(20x20)	23.20%	49.24%	29.76%	30.43%
TA22(20x20)	21.80%	46.86%	25.23%	31.03%
TA23(20x20)	18.39%	45.04%	25.16%	24.06%
TA24(20x20)	20.83%	44.67%	27.58%	27.23%
TA25(20x20)	24.24%	52.19%	25.39%	26.60%
TA26(20x20)	24.04%	47.81%	30.23%	32.65%
TA27(20x20)	19.96%	51.58%	34.92%	37.93%
TA28(20x20)	22.58%	47.75%	29.56%	29.06%
TA29(20x20)	18.38%	49.42%	28.26%	29.93%
TA30(20x20)	27.27%	46.10%	27.80%	29.01%
TA31(30x15)	26.19%	36.39%	33.50%	33.02%
TA32(30x15)	34.13%	40.53%	37.65%	37.59%
TA33(30x15)	36.34%	42.29%	39.06%	37.74%
TA34(30x15)	32.31%	38.10%	34.41%	36.09%
TA35(30x15)	38.49%	39.68%	40.71%	37.52%
TA36(30x15)	39.53%	34.19%	31.36%	34.73%
TA37(30x15)	21.17%	28.86%	31.48%	32.51%
TA38(30x15)	20.07%	26.88%	24.96%	26.84%
TA39(30x15)	25.34%	38.37%	32.09%	31.41%
TA40(30x15)	16.38%	25.07%	23.71%	24.00%
TA41(30x20)	28.08%	61.15%	43.74%	42.95%
TA42(30x20)	28.58%	53.77%	40.80%	37.91%
TA43(30x20)	21.90%	50.82%	37.03%	33.32%
TA44(30x20)	26.93%	58.20%	43.56%	43.95%

Συνέχεια σε επόμενη σελίδα

Πίνακας 5.3 – Συνέχεια από προηγούμενη σελίδα

Problem	**B&B**	**ACO**	**PSO**	**SA**
	Gap(%)	Gap(%)	Gap(%)	Gap(%)
TA45(30x20)	23.69%	46.98%	43.71%	40.61%
TA46(30x20)	32.53%	65.47%	46.58%	42.07%
TA47(30x20)	23.55%	51.00%	32.78%	36.27%
TA48(30x20)	23.45%	54.89%	38.57%	34.47%
TA49(30x20)	034.38%	49.90%	36.42%	36.83%
TA50(30x20)	25.50%	51.40%	40.75%	39.36%
TA51(50x15)	100%	27.07%	34.42%	31.32%
TA52(50x15)	100%	28.58%	100.00%	32.01%
TA53(50x15)	100%	15.07%	25.30%	24.18%
TA54(50x15)	100%	24.78%	29.53%	24.25%
TA55(50x15)	100%	21.30%	31.70%	29.32%
TA56(50x15)	100%	25.89%	31.15%	28.12%
TA57(50x15)	100%	34.48%	35.81%	30.63%
TA58(50x15)	100%	28.26%	33.55%	31.32%
TA59(50x15)	100%	19.38%	28.00%	26.40%
TA60(50x15)	100%	18.95%	27.54%	24.17%
TA61(50x20)	100%	48.15%	40.18%	39.60%
TA62(50x20)	100%	55.20%	47.15%	43.35%
TA63(50x20)	100%	39.23%	42.56%	34.32%
TA64(50x20)	100%	40.31%	37.20%	34.18%
TA65(50x20)	100%	42.99%	42.99%	37.11%
TA66(50x20)	100%	100%	44.88%	40.64%
TA67(50x20)	100%	100%	47.63%	42.90%
TA68(50x20)	100%	100%	43.45%	37.01%
TA69(50x20)	100%	100%	48.24%	44.87%
TA70(50x20)	100%	100%	47.24%	45.22%
TA71(100x20)	100%	100%	38.53%	33.03%

Συνέχεια σε επόμενη σελίδα

Πίνακας 5.3 – Συνέχεια από προηγούμενη σελίδα

Problem	**B&B**	**ACO**	**PSO**	**SA**
	Gap(%)	Gap(%)	Gap(%)	Gap(%)
TA72(100x20)	100%	100%	34.35%	25.35%
TA73(100x20)	100%	100%	38.28%	31.40%
TA74(100x20)	100%	100%	35.21%	26.98%
TA75(100x20)	100%	100%	37.66%	31.74%
TA76(100x20)	100%	100%	42.20%	31.03%
TA77(100x20)	100%	100%	41.94%	33.30%
TA78(100x20)	100%	100%	40.45%	30.91%
TA79(100x20)	100%	100%	41.87%	31.05%
TA80(100x20)	100%	100%	37.68%	28.53%
Average Gap	39.11%	36.04%	25.17%	24.27%
Average Comm.Gap	20.47%	26.98%	20.51%	21.61%

Παρατηρούμε πώς για τα προβλήματα που οι μέθοδοι δεν έδωσαν λύση το gap το θεωρήσαμε ως 100% δηλαδή το μέγιστο και συμπεριλαμβάνουμε στη σύγκριση των μεθόδων. Το ίδιο ισχύει και για τις μεθόδους, όπου το gap του αποτελέσματος τους ξεπερνούσε το 100%. Επιπλεόν, στις δύο τελευταίες σειρές του πίνακα, έχουμε υπολογίσει τους μέσους όρους των gap των αλγόριθμων συνολικά. Η σειρά που περιέχει το Average Gap μας δείχνει, το μέσο όρο των gap και για τα 128 προβλήματα όλων των μεθόδων. Βλέπουμε ότι ο SA έχει το μικρότερο μέσο όρο με περίπου 24.27, ακολουθεί ο PSO με περίπου 25.17, ο ACO έχει μέσο όρο περίπου 36.04 ενώ ο B&B έχει το μεγαλύτερο μέσο όρο με περίπου 39.11. Συνεπώς, παρατηρούμε ότι σταδιακά και με την αύξηση της δυσκολίας των προβλημάτων ο SA έχει τον καλύτερο μέσο όρο σε σχέση με τους υπόλοιπους τρεις, καθώς οι λύσεις που παρήγαγε βρίσκονταν κοντά στο 20% απο τις βέλτιστες λύσεις.

Ακόμη, υπολογίστηκε και ο μέσος όρος των gap και για τα κοινά τους προβλήματα, δηλαδή στα προβλήματα που και οι τέσσερις μέθοδοι βρήκαν κάποια λύση. Αυτό το μέσο όρο τον παρουσιάζουμε στη γραμμή Average Com. Gap. Θεωρούμε πως τα κοινά προβλήματα των μεθόδων τελειώνουν με το τελευταίο πρόβλημα που έδωσε λύση ο ακριβής αλγόριθμος το (TA50). Για αυτά τα προβλήματα, λοιπόν ο

B&B είχε τον μικρότερο μέσο όρο με περίπου 20.47, ο PSO έχει περίπου 20.51, ο SA έχει περίπου 21.61 και ο ACO έχει περίπου 26.98 που είναι και ο μεγαλύτερος μέσος όρος. Άρα, για τα κοινά τους προβλήματα τον καλύτερο μέσο όρο τον έχει ο B&B.

Συνεπώς, από τους τέσσερις αλγόριθμους, ο PSO και ο SA επιλύσανε όλα τα προβλήματα σε σχέση με τους άλλους δύο, ωστόσο ο SA είχε καλύτερο μέσο όρο ως αναφορά το gap, άρα καταλήγουμε στο συμπέρασμα ότι είναι ο πιο αποδοτικός για την εύρεση της βέλτιστης λύσης και ο λιγότερο αποδοτικός είναι ο αλγόριθμος B&B με το μεγαλύτερο μέσο όρο gap και τις λιγότερες λύσεις σε 128 προβλήματα. Στο Σχήμα 5.1 παρουσιάζουμε το γράφημα του gap των αλγορίθμων σε σχέση με τον συνολικό αριθμό προβλημάτων προς επίλυση. Βλέπουμε και με τη χρήση του γραφήματος ότι ο SA (μπλέ χρώμα) παρουσιάζει μια στεθερότητα για όλα τα προβλήματα στην απόκλιση από τη βέλτιστη λύση, με τις τιμές του gap να είναι κοντά στο 20-35%. Ανάλογη πορεία εκ πρώτης όψεος, φαίνεται να έχει και ο PSO ώστοσο, παρατηρούμε πως για τα τελευταία προβλήματα, το gap του PSO είναι υπερβαίνει από αυτού του SA. Φαίνεται, λοιπόν ότι έκει υπάρχει η διαφορά που καθιστά τη μέθοδο του προσομοιωμένου ανορθωτή (SA) πιο αποδοτική απο τις υπόλοιπες.

Ο Πίνακας 5.4 περιέχει τα αποτελέσματα των χρόνων που χρειάστηκε η κάθε μέθοδος για να την παραγωγή και τη μη παραγωγή λύσεων. Ακόμη, πρέπει να σημειώσουμε πως για τις μεθόδους που δεν έδωσαν κάποιο αποτέλεσμα μέσα στα πλαίσια του χρονικού ορίου, ο χρόνος τους έχει κρατηθεί στο μέγιστο, και συμπεριλαμβάνονται για τη σύγκριση. Ο πίνακας λοιπόν αποτελείται από πέντε στήλες, μια για κάθε μέθοδο όπως και οι προηγούμενοι δύο πίνακες (Πίνακας αποτελεσμάτων και Πίνακας Gap). Άρα, ο πίνακας αποτελεσμάτων χρόνου, έχει στην πρώτη στήλη το πρόβλημα προς λύση, στη δεύτερη έχει τους χρόνους εκτέλεσης του B&B αλγόριθμου, στην τρίτη έχει τους χρόνους εκτέλεσης του ACO, στην τέταρτη έχει τους χρόνους εκτέλεσης του PSO και στην πέμπτη τους χρόνους εκτέλεσης του SA.

Πίνακας 5.4: Χρόνοι εκτέλεσης αλγορίθμων για μικρά και μεσαία προβλήματα.

Problem	**B&B**	**ACO**	**PSO**	**SA**
MT06(6x6)	1456.08s	6.788s	0.11s	0.007s
Συνέχεια σε επόμενη σελίδα				

Πίνακας 5.4 – Συνέχεια από προηγούμενη σελίδα

Problem	**B&B**	**ACO**	**PSO**	**SA**
MT10(10x10)	3600s	61.488s	0.22s	0.008s
MT20(20x5)	3600s	102.936s	0.204s	0.010s
ABZ5(10x10)	3600s	57.881s	0.56s	0.007s
ABZ6(10x10)	3600s	57.968s	0.700s	0.007s
ABZ7(20x15)	3600s	1063.459s	3.283s	0.012s
ABZ8(20x15)	3600s	3600s	3.496s	0.013s
ABZ9(20x15)	3600s	3600s	3.430s	0.013s
LA01(10x5)	3600s	15.296s	0.396s	0.008s
LA02(10x5)	3600s	16.400s	0.394s	0.009s
LA03(10x5)	3600s	15.633s	0.396s	0.009s
LA04(10x5)	3600s	15.908s	0.403s	0.007s
LA05(10x5)	3600s	15.350s	0.438s	0.009s
LA06(15x5)	3600s	44.800s	0.655s	0.008s
LA07(15x5)	3600s	44.548s	0.625s	0.009s
LA08(15x5)	3600s	46.322s	0.634s	0.008s
LA09(15x5)	3600s	44.155s	0.638s	0.008s
LA10(15x5)	3600s	43.339s	0.689s	0.009s
LA11(20x5)	3600s	95.551s	0.865s	0.008s
LA12(20x5)	3600s	97.036s	0.867s	0.008s
LA13(20x5)	3600s	101.359s	0.847s	0.009s
LA14(20x5)	3600s	95.121s	0.947s	0.011s
LA15(20x5)	3600s	99.553s	0.824s	0.008s
LA16(10x10)	3600s	57.771s	0.858s	0.009s
LA17(10x10)	3600s	58.302s	0.849s	0.010s
LA18(10x10)	3600s	58.684s	0.887s	0.009s
LA19(10x10)	3600s	59.071s	0.871s	0.008s
LA20(10x10)	3600s	55.822s	0.879s	0.010s
LA21(15x10)	3600s	215.225s	1.399s	0.009s
Συνέχεια σε επόμενη σελίδα				

Πίνακας 5.4 – Συνέχεια από προηγούμενη σελίδα

Problem	B&B	ACO	PSO	SA
LA22(15x10)	3600s	212.492s	1.373s	0.011s
LA23(15x10)	3600s	229.578s	1.420s	0.012s
LA24(15x10)	3600s	216.358s	1.470s	0.010s
LA25(15x10)	3600s	247.772s	1.502s	0.010s
LA26(20x10)	3600s	3600s	2.039s	0.011s
LA27(20x10)	3600s	3600s	1.939s	0.011s
LA28(20x10)	3600s	3600s	2.097s	0.010s
LA29(20x10)	3600s	3600s	2.125s	0.012s
LA30(20x10)	3600s	3600s	2.103s	0.012s
LA31(30x10)	3600s	3600s	3.461s	0.014s
LA32(30x10)	3600s	3600s	3.343s	0.013s
LA33(30x10)	3600s	3600s	3.455s	0.013s
LA34(30x10)	3600s	3600s	3.432s	0.013s
LA35(30x10)	3600s	3600s	3.291s	0.013s
LA36(15x15)	3600s	3600s	2.280s	0.012s
LA37(15x15)	3600s	3600s	2.390s	0.013s
LA38(15x15)	3600s	3600s	2.315s	0.013s
LA39(15x15)	3600s	3600s	2.365s	0.011s
LA40(15x15)	3600s	3600s	2.338s	0.011s
TA01(15x15)	3600s	3600s	2.465s	0.014s
TA02(15x15)	3600s	3600s	2.399s	0.019s
TA03(15x15)	3600s	3600s	2.434s	0.022s
TA04(15x15)	3600s	3600s	2.459s	0.014s
TA05(15x15)	3600s	3600s	2.446s	0.012s
TA06(15x15)	3600s	3600s	2.528s	0.015s
TA07(15x15)	3600s	3600s	2.449s	0.012s
TA08(15x15)	3600s	3600s	2.743s	0.012s
TA09(15x15)	3600s	3600s	2.399s	0.014s

Συνέχεια σε επόμενη σελίδα

Πίνακας 5.4 – Συνέχεια από προηγούμενη σελίδα

Problem	B&B	ACO	PSO	SA
TA10(15x15)	3600s	3600s	2.343s	0.023s
TA11(20x15)	3600s	3600s	3.418s	0.015s
TA12(20x15)	3600s	3600s	3.401s	0.014s
TA13(20x15)	3600s	3600s	3.374s	0.015s
TA14(20x15)	3600s	3600s	3.578s	0.015s
TA15(20x15)	3600s	3600s	3.425s	0.017s
TA16(20x15)	3600s	3600s	3.560s	0.015s
TA17(20x15)	3600s	3600s	3.833s	0.015s
TA18(20x15)	3600s	3600s	3.467s	0.014s
TA19(20x15)	3600s	3600s	3.509s	0.014s
TA20(20x15)	3600s	3600s	3.723s	0.018s
TA21(20x20)	3600s	3600s	4.739s	0.017s
TA22(20x20)	3600s	3600s	4.708s	0.017s
TA23(20x20)	3600s	3600s	4.789s	0.018s
TA24(20x20)	3600s	3600s	4.765s	0.017s
TA25(20x20)	3600s	3600s	4.774s	0.018s
TA26(20x20)	3600s	3600s	4.606s	0.031s
TA27(20x20)	3600s	3600s	5.564s	0.018s
TA28(20x20)	3600s	3600s	4.856s	0.018s
TA29(20x20)	3600s	3600s	4.748s	0.017s
TA30(20x20)	3600s	3600s	5.340s	0.023s
TA31(30x15)	3600s	3600s	5.378s	0.019s
TA32(30x15)	3600s	3600s	5.374s	0.021s
TA33(30x15)	3600s	3600s	5.419s	0.021s
TA34(30x15)	3600s	3600s	5.737s	0.026s
TA35(30x15)	3600s	3600s	5.464s	0.023s
TA36(30x15)	3600s	3600s	5.389s	0.023s
TA37(30x15)	3600s	3600s	5.871s	0.022s

Συνέχεια σε επόμενη σελίδα

Πίνακας 5.4 – Συνέχεια από προηγούμενη σελίδα

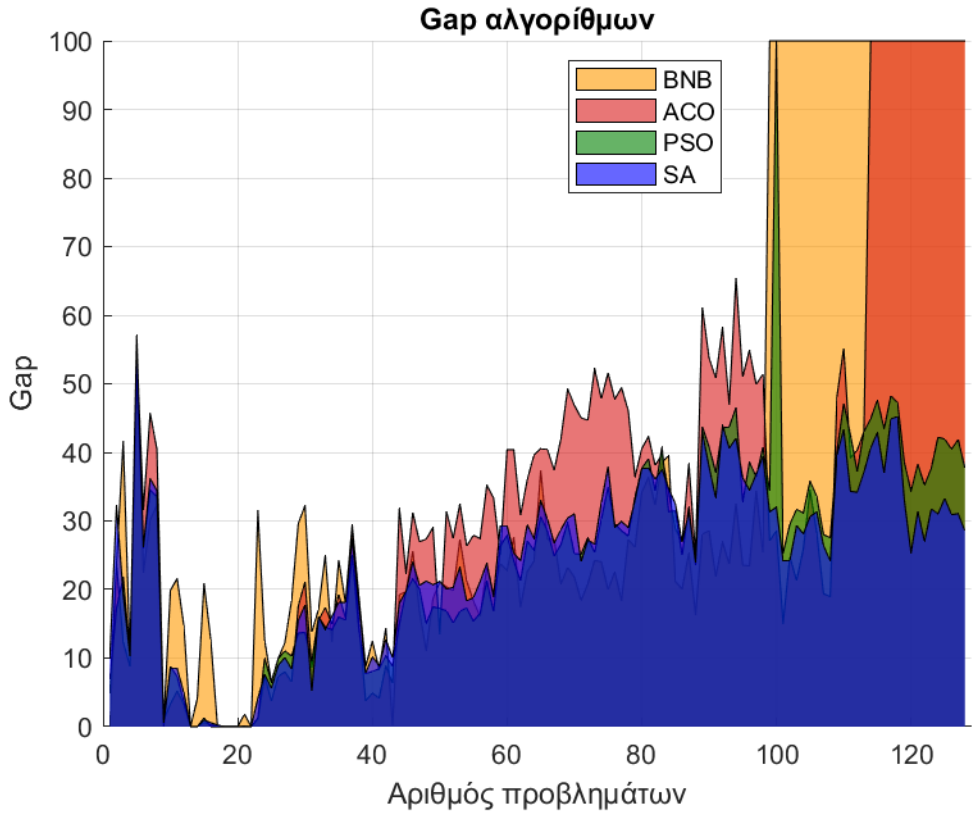
Problem	B&B	ACO	PSO	SA
TA38(30x15)	3600s	3600s	5.568s	0.023s
TA39(30x15)	3600s	3600s	5.534s	0.026s
TA40(30x15)	3600s	3600s	5.331s	0.024s
TA41(30x20)	3600s	3600s	8.068s	0.028s
TA42(30x20)	3600s	3600s	8.527s	0.030s
TA43(30x20)	3600s	3600s	8.032s	0.024s
TA44(30x20)	3600s	3600s	8.175s	0.025s
TA45(30x20)	3600s	3600s	8.204s	0.025s
TA46(30x20)	3600s	3600s	7.812s	0.026s
TA47(30x20)	3600s	3600s	8.149s	0.026s
TA48(30x20)	3600s	3600s	8.047s	0.025s
TA49(30x20)	3600s	3600s	7.720s	0.026s
TA50(30x20)	3600s	3600s	8.708s	0.026s
TA51(50x15)	3600s	3600s	9.943s	0.029s
TA52(50x15)	3600s	3600s	9.905s	0.031s
TA53(50x15)	3600s	3600s	10.902s	0.051s
TA54(50x15)	3600s	3600s	10.785s	0.032s
TA55(50x15)	3600s	3600s	9.719s	0.029s
TA56(50x15)	3600s	3600s	9.675s	0.029s
TA57(50x15)	3600s	3600s	9.707s	0.031s
TA58(50x15)	3600s	3600s	10.266s	0.032s
TA59(50x15)	3600s	3600s	10.422s	0.029s
TA60(50x15)	3600s	3600s	10.015s	0.031s
TA61(50x20)	3600s	3600s	13.211s	0.046s
TA62(50x20)	3600s	3600s	13.860s	0.039s
TA63(50x20)	3600s	3600s	13.908s	0.036s
TA64(50x20)	3600s	3600s	13.676s	0.036s
TA65(50x20)	3600s	3600s	13.732s	0.039s

Συνέχεια σε επόμενη σελίδα

Πίνακας 5.4 – Συνέχεια από προηγούμενη σελίδα

Problem	**B&B**	**ACO**	**PSO**	**SA**
TA66(50x20)	3600s	3600s	14.736s	0.038s
TA67(50x20)	3600s	3600s	14.928s	0.037s
TA68(50x20)	3600s	3600s	16.086s	0.055s
TA69(50x20)	3600s	3600s	14.765s	0.042s
TA70(50x20)	3600s	3600s	14.992s	0.047s
TA71(100x20)	3600s	3600s	33.057s	0.072s
TA72(100x20)	3600s	3600s	36.654s	0.070s
TA73(100x20)	3600s	3600s	50.992s	0.081s
TA74(100x20)	3600s	3600s	48.186s	0.140s
TA75(100x20)	3600s	3600s	48.463s	0.199s
TA76(100x20)	3600s	3600s	51.909s	0.106s
TA77(100x20)	3600s	3600s	50.238s	0.145s
TA78(100x20)	3600s	3600s	48.619s	0.068s
TA79(100x20)	3600s	3600s	51.370s	0.097s
TA80(100x20)	3600s	3600s	47.914s	0.151s
Average Time	3583.255s	2755.871s	8.047s	0.026s
Average Com.Time	3578.123s	2497.476s	3.239s	0.015s

Αυτό που παρατηρούμε για τους χρόνους στον πίνακα των αποτελεσμάτων χρόνου είναι ότι οι μέθοδοι B&B και ACO στα περισσότερα προβλήματα χρειάστηκαν όλο το χρονικό όριο που δόθηκε. Συγκεκριμένα, ο B&B χρησιμοποίησε όλο το χρονικό διάστημα της μίας ώρας σε 97 από τα 98 τα οποία έλυσε, ενώ ο ACO σε 97 από τα 113 τα οποία έλυσε. Σε αντίθεση, οι άλλοι δύο ευρετικοί αλγόριθμοι PSO και SA δεν χρειάστηκαν σε κανένα πρόβλημα το χρονικό όριο. Επιπλέον, υπολογίσαμε το μέσο των χρόνων για τις μεθόδους, όπως φαίνεται από τη σειρά Average Time. Παρατηρούμε λοιπόν ότι, ο B&B έχει μέσο όρο επίλυσης των προβλημάτων 3583,255 δευτερόλεπτα (περίπου 59 λεπτά), ο ACO έχει 2755,871 δευτερόλεπτα (περίπου 45 λεπτά), ο PSO έχει 8,047 δευτερόλεπτα, ενώ ο SA έχει μέσο όρο μόλις 0.026 δευτερόλεπτα για την επίλυση των προβλημάτων. Άρα, τον καλύτερο μέσο όρο επίλυσης όλων των προβλημάτων έχει ο SA με 0.026 δευτερόλεπτα, έπειτα ακολουθεί ο PSO



Σχήμα 5.1: Γράφημα gap αλγορίθμων για όλα τα προβλήματα

με 8 δευτερόλεπτα περίπου, μετά ο ACO με περίπου 45 λεπτά και τέλος ο B&B με περίπου 57 λεπτά. Όπως και για τον πίνακα των gap έτσι και σε αυτό τον πίνακα των χρονικών αποτελεσμάτων, προχωρήσαμε στον υπολογισμό του μέσου όρου και για τα κοινά προβλήματα των τεσσάρων μεθόδων. Παρατηρούμε λοιπόν, και πάλι πως ο SA είναι ο ταχύτερος με μέσο όρο 0.015 δευτερόλεπτα, ακολουθεί ο PSO με 3.239 δευτερόλεπτα, μετά ο ACO με μέσο όρο 2497.476 δευτερόλεπτα (περίπου 42 λεπτά) και τέλος ο B&B με μέσο όρο 3578.123 δευτερόλεπτα (περίπου 59 λεπτά).

Επιπλέον, πρέπει να αναφέρουμε ότι, με γνώμονα το Average Com.Gap του Πίνακα 5.3, οι δύο μέθοδοι B&B και ACO παρόλο που χρησιμοποίησαν όλο το χρόνο που δόθηκε, οι λύσεις που έδωσαν δεν απέχουν πολύ από τις βέλτιστες. Αυτό μας οδηγεί στο συμπέρασμα ότι οι μέθοδοι δεν θα σταματούσαν την αναζήτηση αν το χρονικό περιθώριο ήταν μεγαλύτερο, στην περίπτωση που δεν ολοκλήρωναν την αναζήτηση τους ολικά. Επίσης, για μεγαλύτερο χρονικό περιθώριο εκτέλεσης, οι αλγόριθμοι αυτοί θα μπορούσαν να πλησιάσουν ακόμη περισσότερο στη βέλτιστη λύση ή και να τη βρουν. Αυτό ισχύει περισσότερο για τον ακριβή αλγόριθμο που στόχος του είναι να παράγει την καλύτερη λύση. Συνεπώς, για τους τέσσερις αλ-

γορίθμους που παρουσιάσαμε, καταλήγουμε στο γεγονός ότι αποτελεσματικότερος είναι ο αλγόριθμος του προσομοιωμένου ανορθωτή και ως προς τον παράγοντα του χρόνου.

Κεφάλαιο 6

Συμπεράσματα

Σε αυτή τη διπλωματική εργασία, μελετήσαμε το πρόβλημα χρονοδρομολόγησης εργασιών, το οποίο βρίσκει εφαρμογές σε πολλούς κλάδους της βιομηχανίας, στο σχεδιασμό παραγωγής, στον προγραμματισμό εργασιών, στη διαχείριση λειτουργιών και έργων. Το πρόβλημα αυτό, αφορά τη δρομολόγηση εργασιών σε μηχανές ή πόρους με σκοπό την επεξεργασία αυτών στο μικρότερο χρονικό διάστημα, ώστε με αυτό τον τρόπο να παραχθεί το καλύτερο δρομολόγιο στον καλύτερο χρόνο.

Επιπλέον, αναφερθήκαμε στις διάφορες παραλλαγές που μπορεί να εμφανιστεί αυτό το πρόβλημα, τις οντότητες του αλλά και τους περιορισμούς που μπορεί να προκύψουν. Ακόμη, αναφέραμε διαφορετικούς στόχους που μπορούμε να βελτιστοποιήσουμε (makespan, unit penalty κλπ), αλλά και τρόπους αναπαράστασης του προβλήματος (διαγράμματα gantt, διαζευκτικό γράφο). Στη συνέχεια, έγινε μια ιστορική αναδρομή, δηλαδή από ποιόν ξεκίνησε η μελέτη αυτού του προβλήματος και συνεχίσαμε με τη βιβλιογραφική ανασκόπησή του, όπου παρουσιάστηκε πληθώρα τεχνικών επίλυσης του JSSP ξεκινώντας από τις αρχικές υλοποιήσεις μεθόδων, όπως ακριβείς αλγορίθμους έως και τις πιο εξελιγμένες μεθόδους, π.χ. γενετικούς αλγορίθμους, αλγόριθμους τοπικής αναζήτησης που επιλύουν το πρόβλημα. Επίσης, δεν πρέπει να παραβλέπουμε το γεγονός ότι το JSSP είναι από τα πλέον σημαντικότερα προβλήματα συνδυαστικής βελτιστοποίησης για αυτό και υπάρχουν παρά πολλές και διαφερετικές υλοποιήσεις για την επίλυση του.

Στην παρούσα εργασία, χρησιμοποιήθηκαν τέσσερις αλγόριθμοι για την επίλυση του προβλήματος, όπου ο πρώτος είναι ένας ακριβής αλγόριθμος διακλάδωσης και δέσμευσης (branch and bound) με μέθοδο αναζήτησης πρώτα σε βάθος. Ακόμη, υλοποιήσαμε τρείς ευρετικούς αλγορίθμους, έναν αλγόριθμο σμήνους σωματιδίων, έναν

αποικίας μυρμηγκιών και έναν προσομοιωμένου ανορθωτή. Αναλύοντας τα αποτελέσματα που παράχθηκαν και παρουσιάστηκαν στους πίνακες της εργασίας, αλλά και τη σύγκριση που έγινε για τις τέσσερις μεθόδους, καταλήξαμε στο συμπέρασμα ότι η βελτιστοποίηση με τον προσομοιωμένο ανορθωτή (SA) ήταν η πιο αποδοτική μέθοδος, επιλύοντας όλα τα προβλήματα που διατέθηκαν, στον μικρότερο χρόνο αλλά και με τη μικρότερη απόσταση (gap) από τη βέλτιστη λύση, διατηρώντας το χαμηλότερο ποσοστό και στις δύο περιπτώσεις. Αντιθέτως, διαπιστώθηκε ότι η χειρότερη μέθοδος ήταν η ακριβής μέθοδος (B&B) ως προς την παραγωγή λύσεων καθώς βρήκε αποτελέσματα στα λιγότερα προβλήματα, ενώ παράλληλα χρειάστηκε και τον περισσότερο χρόνο για να φτάσει σε αυτά.

Συμπερασματικά, θα λέγαμε ότι, παρόλο που αυτή η μελέτη επιτέλεσε με επιτυχία την υλοποίηση και αξιολόγηση τεσσάρων αλγορίθμων για την επίλυση προβλημάτων χρονοπρογραμματισμού εργασιών (JSSP), υπάρχει ακόμα περιθώριο μελλοντικής βελτίωσης αυτών με την εφαρμογή παράλληλης εκτέλεσης, χρησιμοποιώντας νήματα. Μέσω της παράλληλης εκτέλεσης, μπορούμε να διανείμουμε τις υπολογιστικές εργασίες των αλγορίθμων σε πολλαπλά νήματα, εκμεταλλευόμενοι το δυναμικό των σύγχρονων πολυπύρηνων επεξεργαστών. Με αυτή την προσέγιση, σε παρόμοιες μελλοντικές έρευνες θα μπορέσουμε να βελτιώσουμε συνολικά την αποτελεσματικότητα και την απόδοση των αλγορίθμων, οδηγώντας τους τελικά σε πιο γρήγορες και καλύτερες λύσεις.

Βιβλιογραφία

[1] M. L. Pinedo, *Scheduling*, vol. 29. Springer, 2012.

[2] K. R. Baker, "Introduction to sequencing and scheduling," *(No Title)*, 1974.

[3] J. A. Smith and R. E. Fogarty, "Job shop scheduling with finite capacities," *Operations Research*, vol. 6, no. 3, pp. 363–373, 1958.

[4] W. J. Stevenson, M. Hojati, and J. Cao, *Operations management*. McGraw-Hill Education New York, 2014.

[5] M. J. Brusco and L. M. Clover, "Stochastic optimization in health care delivery systems," *Annals of Operations Research*, vol. 35, no. 1, pp. 157–179, 1992.

[6] H. A. Taha, *Operations research: an introduction*. Pearson Education India, 2013.

[7] H. Xiong, S. Shi, D. Ren, and J. Hu, "A survey of job shop scheduling problem: The types and models," *Computers & Operations Research*, vol. 142, p. 105731, 2022.

[8] J. K. Lenstra, A. R. Kan, and P. Brucker, "Complexity of machine scheduling problems," in *Annals of discrete mathematics*, vol. 1, pp. 343–362, Elsevier, 1977.

[9] J. F. Muth, G. L. Thompson, and P. R. Winters, "Industrial scheduling," *(No Title)*, 1963.

[10] E. L. Lawler, J. K. Lenstra, A. H. R. Kan, and D. B. Shmoys, "Sequencing and scheduling: Algorithms and complexity," *Handbooks in operations research and management science*, vol. 4, pp. 445–522, 1993.

[11] E. Balas, "Machine sequencing via disjunctive graphs: an implicit enumeration algorithm," *Operations research*, vol. 17, no. 6, pp. 941–957, 1969.

[12] B. Roy and B. Sussmann, "Les problemes d'ordonnancement avec contraintes disjonctives," *Note ds*, vol. 9, 1964.

[13] S. M. Johnson, "Optimal two-and three-stage production schedules with setup times included," *Naval research logistics quarterly*, vol. 1, no. 1, pp. 61–68, 1954.

[14] A. Arisha, P. Young, and M. El Baradie, "Job shop scheduling problem: an overview," 2001.

[15] W. E. Smith, "Various optimizers for single-stage production," *Naval Research Logistics Quarterly*, vol. 3, no. 1-2, pp. 59–66, 1956.

[16] J. R. Jackson, "Simulation research on job shop production," *Naval Research Logistics Quarterly*, vol. 4, no. 4, pp. 287–295, 1957.

[17] R. W. Conway, "" theory of scheduling,"," *Addison Wesley*, 1967.

[18] G. H. Brooks, "An algorithm for finding optimal or near optimal solutions to the production scheduling problem," *The Journal of Industrial Engineering*, vol. 16, no. 1, pp. 34–40, 1969.

[19] B. Giffler and G. L. Thompson, "Algorithms for solving production-scheduling problems," *Operations research*, vol. 8, no. 4, pp. 487–503, 1960.

[20] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. R. Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," in *Annals of discrete mathematics*, vol. 5, pp. 287–326, Elsevier, 1979.

[21] S. C. Graves, "A review of production scheduling," *Operations research*, vol. 29, no. 4, pp. 646–675, 1981.

[22] A. S. Manne, "On the job-shop scheduling problem," *Operations research*, vol. 8, no. 2, pp. 219–223, 1960.

[23] J. Carlier, "Ordonnancements a contraintes disjonctives," *RAIRO-Operations Research*, vol. 12, no. 4, pp. 333–350, 1978.

[24] G. McMahon and M. Florian, "On scheduling with ready times and due dates to minimize maximum lateness," *Operations research*, vol. 23, no. 3, pp. 475–482, 1975.

[25] J. Carlier and É. Pinson, "An algorithm for solving the job-shop problem," *Management science*, vol. 35, no. 2, pp. 164–176, 1989.

[26] L. Davis, "Job shop scheduling with genetic algorithms," in *Proceedings of the first International Conference on Genetic Algorithms and their Applications*, pp. 136–140, Psychology Press, 2014.

[27] J. Adams, E. Balas, and D. Zawack, "The shifting bottleneck procedure for job shop scheduling," *Management science*, vol. 34, no. 3, pp. 391–401, 1988.

[28] D. Applegate and W. Cook, "A computational study of the job-shop scheduling problem," *ORSA Journal on computing*, vol. 3, no. 2, pp. 149–156, 1991.

[29] P. J. Van Laarhoven, E. H. Aarts, and J. K. Lenstra, "Job shop scheduling by simulated annealing," *Operations research*, vol. 40, no. 1, pp. 113–125, 1992.

[30] E. D. Taillard, "Parallel taboo search techniques for the job shop scheduling problem," *ORSA journal on Computing*, vol. 6, no. 2, pp. 108–117, 1994.

[31] E. Taillard, "Benchmarks for basic scheduling problems," *european journal of operational research*, vol. 64, no. 2, pp. 278–285, 1993.

[32] P. S. Ow and T. E. Morton, "Filtered beam search in scheduling," *The International Journal Of Production Research*, vol. 26, no. 1, pp. 35–62, 1988.

[33] I. Sabuncuoglu and B. Gurgun, "A neural network model for scheduling problems," *European Journal of operational research*, vol. 93, no. 2, pp. 288–299, 1996.

[34] T.-L. Lin, S.-J. Horng, T.-W. Kao, Y.-H. Chen, R.-S. Run, R.-J. Chen, J.-L. Lai, and I.-H. Kuo, "An efficient job-shop scheduling algorithm based on particle swarm optimization," *Expert Systems with Applications*, vol. 37, no. 3, pp. 2629–2636, 2010.

[35] M. Sakawa and R. Kubota, "Fuzzy programming for multiobjective job shop scheduling with fuzzy processing time and fuzzy duedate through genetic algorithms," *European Journal of operational research*, vol. 120, no. 2, pp. 393–407, 2000.

[36] A. Colorni, M. Dorigo, V. Maniezzo, and M. Trubian, "Ant system for job-shop scheduling," *JORBEL-Belgian Journal of Operations Research, Statistics, and Computer Science*, vol. 34, no. 1, pp. 39–53, 1994.

[37] B. Çaliş and S. Bulkan, "A research survey: review of ai solution strategies of job shop scheduling problem," *Journal of Intelligent Manufacturing*, vol. 26, pp. 961–973, 2015.

[38] A. H. Land and A. G. Doig, *An automatic method for solving discrete programming problems*. Springer, 2010.

[39] E. B. Nababan, A. R. Hamdan, S. Abdullah, and M. S. Zakaria, "Branch and bound algorithm in optimizing job shop scheduling problems," in *2008 International Symposium on Information Technology*, vol. 1, pp. 1–5, IEEE, 2008.

[40] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[41] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95-international conference on neural networks*, vol. 4, pp. 1942–1948, IEEE, 1995.

[42] Y. Shi *et al.*, "Particle swarm optimization: developments, applications and resources," in *Proceedings of the 2001 congress on evolutionary computation (IEEE Cat. No. 01TH8546)*, vol. 1, pp. 81–86, IEEE, 2001.

[43] M. Dorigo, "Optimization, learning and natural algorithms," *Ph. D. Thesis, Politecnico di Milano*, 1992.

[44] R. Zhou, H. P. Lee, and A. Y. Nee, "Applying ant colony optimisation (aco) algorithm to dynamic job shop scheduling problems," *International Journal of Manufacturing Research*, vol. 3, no. 3, pp. 301–320, 2008.