Πολυτεχνειο Κρητης

Σχολη Μηχανικων Παραγωγης και Διοικησης

Αλγοριθμος Βελτιστοποιησης Σμηνους Σωματιδιων Για Το Προβλημα Χρονοπρογραμματισμου Εργασιων Συνεχους Ροης

Διπλωματικη Εργασια

Κυριακου Σωτηριος

Επιβλεπων Καθηγητης:

Καθ. Μαρινακης Ιωαννης

Χανια, 2024

Περίληψη

Η παραγωγή ενός προϊόντος δεν είναι μια ενιαία διαδικασία αλλά απαρτίζεται από επιμέρους. Ο προγραμματισμός αυτών των διαδικασιών είναι πολύ σημαντικός προκειμένου η επιχείρηση να αξιοποιεί με βέλτιστο τρόπο το εργατικό δυναμικό και τον εξοπλισμό της. Έτσι, προκύπτει ένα πρόβλημα βελτιστοποίησης που ονομάζεται Βέλτιστος Προγραμματισμός Εργασίας (Optimal Job Scheduling). Μια παραλλαγή του παραπάνω προβλήματος είναι ο Χρονοπρογραμματισμός Εργασιών (Job-shop Scheduling). Ανάμεσα από τις πολλές πιθανές προσεγγίσεις στην διαδικασία εύρεσης βέλτιστης λύσης είναι και ο αλγόριθμος Βελτιστοποίησης Σμήνους Σωματιδίων. Στην παρούσα εργασία θα εφαρμοστεί ο αλγόριθμος αυτός υπό ορισμένες συνθήκες και σε δύο εκδοχές και θα αναλυθεί η επίδοσή του στο συγκεκριμένο πρόβλημα.

Ευχαριστίες

Θα ήθελα να απευθύνω ευχαριστίες στον επιβλέποντα καθηγητή μου, τον κ.Μαρινάκη Ιωάννη, που με την καθοδήγηση του μου έδωσε την ευκαιρία να εργαστώ και να εμβαθύνω σε έναν τόσο ενδιαφέρον τομέα. Επίσης, θέλω να ευχαριστήσω θερμά την οικογένεια μου για την ανιδιοτελή στήριξη που, με κάθε τρόπο, μου προσέφερε. Τέλος, απευθύνω ένα εγκάρδιο ευχαριστώ στους κοντινούς μου ανθρώπους που εμφυσήσανε πνοή στην καθημερινότητα και μορφοποιήσαν, με τον ωραιότερο τρόπο δυνατό, αυτό το στάδιο της ζωής.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

Είναι γεγονός πως οι ραγδαίες εξελίξεις και οι αλλαγές σε σύντομο χρονικό διάστημα είναι ένα από τα χαρακτηριστικά της σημερινής πραγματικότητας. Καθίσταται, επομένως, επιτακτική η ανάγκη σε όλους τους κλάδους να μπορούν να προσαρμοστούν στις αλλαγές αυτές. Κάτι τέτοιο υφίσταται και στον κλάδο της παραγωγής, με τις βιομηχανίες να έρχονται αντιμέτωπες καθημερινά με θέματα προσαρμογής στα ζητήματα της αγοράς, όπως ελαχιστοποίηση του χρόνου παραγωγής του προϊόντος ή του χώρου αποθήκευσης των αποθεμάτων, αλλά και γενικότερα δυναμικής φύσης προβλήματα. Για το λόγο αυτό είναι σημαντικό για μία βιομηχανία να προγραμματίζει ορθώς τη γραμμή παραγωγής της, συνυπολογίζοντας όλα αυτά τα προβλήματα. Η παραγωγή ενός προϊόντος δεν είναι μια ενιαία διαδικασία αλλά απαρτίζεται από επιμέρους διαδικασίες. Ο προγραμματισμός αυτών των διαδικασιών είναι πολύ σημαντικός προκειμένου η επιχείρηση, αρχικά να αξιοποιεί με βέλτιστο τρόπο το εργατικό δυναμικό και τον εξοπλισμό της που συνεπάγεται ελαχιστοποίηση του χρόνου παραγωγής του προϊόντος, του χρόνου αναμονής των πελατών της κ.α. και κατ' επέκταση να μεγιστοποιεί τα κέρδη της. Έτσι, η επιχείρηση έρχεται αντιμέτωπη με ένα πρόβλημα βελτιστοποίησης που ονομάζεται Βέλτιστος Προγραμματισμός Εργασίας (Optimal Job Scheduling). Μια παραλλαγή του παραπάνω προβλήματος είναι ο Χρονοπρογραμματισμός Εργασιών (Job-shop Scheduling). Στην παρούσα εργασία

θα εφαρμοστεί ο αλγόριθμος Βελτιστοποίησης Σμήνους Σωματιδίων [1] για την λύση τέτοιων προβλημάτων.

1.1 Δομή της εργασίας

Η παρούσα διπλωματική εργασία αποτελείται από πέντε κεφάλαια. Κατ'αρχάς, σε αυτό το κεφάλαιο γίνεται μία εισαγωγή στο αντικείμενο και στις έννοιες που θα συναντήσει ο αναγνώστης, ο λόγος συγγραφής και αναλύεται, συνοπτικά, το πρόβλημα και ο τρόπος προσέγγισής του. Στο δεύτερο κεφάλαιο γίνεται ανάλυση των προβλημάτων Χρονοπρογραμματισμού και πως αυτά λαμβάνουν χώρα στη σημερινή κοινωνία. Εν συνεχεία, στο τρίτο κεφάλαιο, παρουσιάζονται κάποιες κατηγορίες αλγορίθμων, όπως οι ευρετικοί αλγόριθμοι, και με πιο αναλυτικό τρόπο, οι αλγόριθμοι Βελτιστοποίησης Σμήνους Σωματιδίων (Particle Swarm Optimization), οι αλγόριθμοι τοπικής αναζήτησης 1-0 Επανατοποθέτηση και 1-1 Ανταλλαγή οι οποίοι χρησιμοποιήθηκαν για την αντιμετώπιση του προβλήματος Χρονοπρογραμματισμού Αντιμετάθεσης Συνεχούς Ροής (Permutation Flow-shop Scheduling Problem). Στο τέταρτο κεφάλαιο, αναλύεται η μοντελοποίηση του προβλήματος, παρατίθεται και εξηγείται η δομή του κώδικα. Τέλος, στο πέμπτο κεφάλαιο, παρατίθενται τα αποτελέσματα σε πίνακες, γίνεται σχολιασμός αυτών, αναφορά και ερμηνεία των συμπερασμάτων, και προτείνονται κάποια σημεία για μελλοντική μελέτη.

1.2 Μεθοδολογία

Οι αλγόριθμοι υλοποιήθηκαν σε γλώσσα προγραμματισμού Python. Η δοκιμή και ο έλεγχος της επίδοσής τους έγινε με την βοήθεια παραδειγμάτων της βιβλιογραφίας από τον Taillard [2] ως σημείο αναφοράς. Υλοποιήθηκαν επίσης, οι αλγόριθμοι 1-1 Ανταλλαγή και 1-0 Επανατοποθέτηση και χρησιμοποιήθηκαν για τη βελτίωση της λύσης σε κάθε βήμα του Αλγορίθμου Βελτιστοποίσης Σμήνους σωματιδίων.

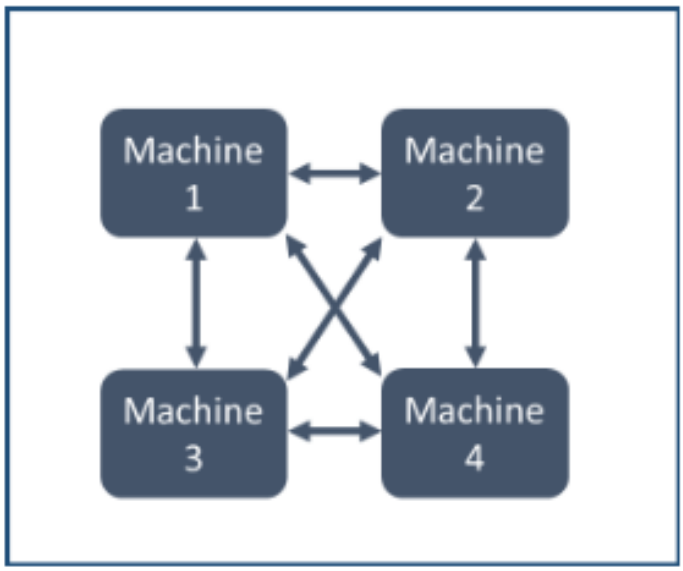
Κεφάλαιο 2

Χρονοπρογραμματισμός Εργασιών

Το πρόβλημα του Χρονοπρογραμματισμού Εργασιών (Job-shop Scheduling) μπορεί να διατυπωθεί ως εξής: Έστω ότι έχουμε να πραγματοποιήσουμε j εργασίες, με διαφορετικούς χρόνους διεκπεραίωσης μεταξύ τους, και m διαθέσιμες μηχανές με διαφορετική δύναμη επεξεργασίας των εργασιών αυτών. Σκοπός είναι να βρεθεί εκείνη η αλληλουχία ανάθεσης των εργασιών στις αντίστοιχες μηχανές ώστε όλες οι εργασίες να τελειώσουν στον ελάχιστο δυνατό χρόνο ή γενικότερα που να ικανοποιεί στο μέγιστο βαθμό τα κριτήρια που θέτει η εκάστοτε επιχείρηση.

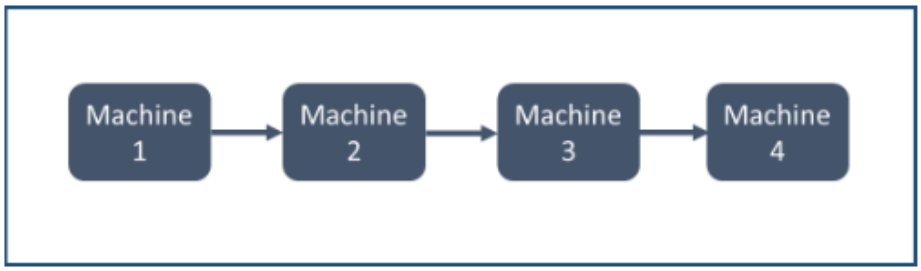
2.1 Χρονοπρογραμματισμός Συνεχούς Ροής

Μια ειδική περίπτωση του προβλήματος χρονοπρογραμματισμού εργασιών αποτελεί ο Χρονοπρογραμματισμός Συνεχούς Ροής [3] (Flow-shop Scheduling), όπως δημοσιεύτηκε από τον S.M. Johnson το 1954. Πρόκειται για ένα συνδυαστικό πρόβλημα βελτιστοποίησης NP δυσκολίας. Εδώ πρέπει να οριστούν και οι περιορισμοί του συστήματος αυτού. Κάθε εργασία του προβλήματος τίθεται προς εκτέλεση από τις m μηχανές κάθε φορά με την ίδια σειρά, με την πρώτη να εκτελείται από την πρώτη μηχανή, τη δεύτερη



Σχήμα 2.1: Ροή εργασιών στο σύστημα Job-shop

από τη δεύτερη μηχανή κ.ο.κ. Κάθε εργασία πρέπει να επεξεργαστεί από κάθε μηχανή και με μία σειρά που θα έχει οριστεί. Καμία μηχανή δεν μπορεί να εκτελέσει πάνω από μία επεξεργασίες ταυτόχρονα ενώ οι εργασίες μπορούν να εκτελεστούν και με διαφορετική σειρά κάθε φορά. Οι χρόνοι επεξεργασίας κάθε εργασίας από κάθε μηχανή περιλαμβάνουν και τους χρόνους προετοιμασίας και εκκίνησης των μηχανών και οι υπό επεξεργασία εργασίες δε γίνεται να διακοπούν.



Σχήμα 2.2: Ροή εργασιών στο σύστημα Flow-shop

Σκοπός είναι να εξαχθεί εκείνη η σειρά εκτέλεσης των j εργασιών από τις m μηχανές, η οποία θα ελαχιστοποιεί το συνολικό χρόνο που χρειάζεται ώστε όλες οι εργασίες να ολοκληρωθούν ή αλλιώς θα ελαχιστοποιείται το makespan. Ως makespan ορίζεται το χρονικό διάστημα από την αρχή έως το τέλος επεξεργασίας όλων των εργασιών. Το

σύνολο των πιθανών λύσεων είναι $(j!)m$, άρα όσο μεγαλύτερος ο αριθμός των εργασιών αλλά και των μηχανών, τόσο πιο δύσκολο είναι να υπολογιστούν όλες οι πιθανές λύσεις και συνεπώς, να βρεθεί η βέλτιστη εξ'αυτών. Όπως αναφέρθηκε και προηγουμένως, στην περίπτωση του flow-shop συστήματος παραγωγής, η σειρά με την οποία εκτελούνται οι επεξεργασίες από τις μηχανές είναι πάντα η ίδια για κάθε προϊόν, ενώ στο job-shop η σειρά αυτή μπορεί να διαφέρει ανά παραγγελία. Συνεπώς, το σύστημα flow-shop μπορεί να θεωρηθεί ως υποκατηγορία του job-shop. Υπάρχει, όμως, και η περίπτωση τα δύο αυτά συστήματα να συνυπάρχουν στη ίδια διαδικασία παραγωγής προϊόντων, εφόσον αυτή χωρίζεται σε δύο στάδια. Για παράδειγμα, σε μία βιομηχανία επίπλων έχουμε το πρώτο στάδιο, που είναι η βασική διαμόρφωση του επίπλου που είναι κοινή διαδικασία για όλες τις παραγγελίες και είναι σύστημα flow-shop, και το δεύτερο στάδιο που είναι η βαφή, το λουστράρισμα κλπ., όπου έχουμε διαφοροποίηση των εργασιών παραγωγής ανάλογα με την κάθε περίπτωση και είναι σύστημα job-shop. Με τη σειρά του, το flow-shop σύστημα παραγωγής χωρίζεται σε δύο κατηγορίες το καθαρό σύστημα flow-shop, όπου όλες οι διαθέσιμες μηχανές χρησιμοποιούνται για την εκτέλεση των εργασιών και το γενικό flow-shop, όπου κάθε εργασία δεν περνάει αναγκαστικά από όλες τις μηχανές. Μια γραμμή παραγωγής αναψυκτικών είναι μια περίπτωση καθαρού flow-shop συστήματος, καθώς η κατεύθυνση εκτέλεσης των εργασιών είναι ορισμένη για κάθε προϊόν και επίσης χρησιμοποιούνται κάθε φορά όλα τα μηχανήματα. Αντίθετα, σε περιπτώσεις παραγωγής ενδυμάτων έχουμε γενικό σύστημα flow-shop, καθώς η διαδοχή των εργασιών παραγωγής μπορεί να είναι ίδια για μια συγκεκριμένη παραγγελία αλλά γενικότερα διαφέρει ανά παραγγελία, με ορισμένα μηχανήματα να μη χρειάζονται να δουλέψουν για την παραγωγή του κάθε προϊόντος. Τα συστήματα flow-shop και job-shop διαφέρουν σε αρκετά σημεία μεταξύ τους. Στην περίπτωση του flow-shop έχουμε σταθερότητα στη ζήτηση των πελατών, αφού οι εργασίες είναι ορισμένες, η ροή εργασιών είναι απλή και γραμμική, το εργατικό δυναμικό δεν είναι αναγκαστικό να έχει υψηλό επίπεδο δεξιοτήτων και η παραγωγή προϊόντων σε ένα συγκεκριμένο χρόνο είναι μεγαλύτερη συγκριτικά με το

job-shop. Από την άλλη, στο job-shop υπάρχει θεωρητικά άπειρη ποικιλία στα προϊόντα, η χρήση των μηχανημάτων γίνεται με ευελιξία και ακολουθείται η στρατηγική που επιτρέπει στους πελάτες να προσαρμόζουν το προϊόν στις ανάγκες τους (make to order), και για αυτό το λόγο δεν είναι απαραίτητη η ύπαρξη αποθεμάτων. Τέλος, οι flow-shop επιχειρήσεις είναι μεγαλύτερες από άποψη εργατικού δυναμικού, εξοπλισμού και εσόδων σε σχέση με τις job-shop.

2.2 Χρονοπρογραμματισμός Αντιμετάθεσης Συνεχούς Ροής

Αν σε ένα σύστημα χρονοπρογραμματισμού σταθερής ροής προσθέσουμε έναν επιπλέον περιορισμό, ότι δηλαδή κάθε μία από τις m επεξεργασίες πρέπει να εκτελεστεί με την ίδια ακριβώς σειρά από καθεμία από τις m μηχανές, προκύπτει ο Χρονοπρογραμματισμός Αντιμετάθεσης Συνεχούς Ροής (Permutation Flow-shop Scheduling Problem ή PFSP για συντομία). Ο χρονοπρογραμματισμός αντιμετάθεσης συνεχούς ροής είναι ένα πρόβλημα X, NP δυσκολίας, που σημαίνει ότι κάθε επιμέρους πρόβλημα $A = (A_1, A_2, \ldots, A_Z)$ μπορεί να μειωθεί σε πολυωνυμικό χρόνο του X.

Για να κατανοήσουμε πλήρως το πρόβλημα, θα ορίσουμε τις n εργασίες ως $\pi = (\pi_1, \pi_2, ..., \pi_n)$. Κάθε εργασία αποτελείται από o λειτουργίες και κάθε λειτουργία εκτελείται από διαφορετική μηχανή. Αν ορίσουμε ως $\pi_{i,j}$ το χρόνο ολοκλήρωσης της j εργασίας στη μηχανή i, με τη συγκεκριμένη μέθοδο βρίσκουμε τη βέλτιστη αντιμετάθεση εργασιών $\pi^* = (\pi_1^*, \pi_2^*, ..., \pi_n^*)$ – ανάμεσα στις $n!$ πιθανές λύσεις – ώστε να ικανοποιείται το κριτήριο μας με τον καλύτερο δυνατό τρόπο. Αν ορίσουμε επίσης ως makespan τη συνάρτηση $C(\pi_j, m)$ ολοκλήρωσης της π_j εργασίας από την m μηχανή, τότε η μαθηματική διατύπωση του προβλήματος έχει ως εξής:

$$C_{1,\pi_1} = p_{1,\pi_1} + s_1 \tag{2.1}$$

$$C_{1,\pi_j} = C_{1,\pi_{j-1}} + p_{1,\pi_j} + s_1, j = 2, ..., n \tag{2.2}$$

$$C_{i,\pi_1} = max\{s_i, C_{i-1,\pi_1}\} + p_{i,\pi_1}, i = 2, ..., m \tag{2.3}$$

$$C_{i,\pi_j} = max\{C_{i,\pi_{j-1}} + s_i, C_{i-1,\pi_j}\} + p_{i,\pi_j}, i = 2, ..., m, j = 2, ..., n \tag{2.4}$$

$$C_{max} = max\{C_{m,\pi_j}\}, 1 \leq j \leq n \tag{2.5}$$

Οι εξισώσεις 2.1 και 2.2 υπολογίζουν το χρόνο ολοκλήρωσης της πρώτης εργασίας και όλων των υπόλοιπων εργασιών στο πρώτο μηχάνημα αντίστοιχα. Οι 2.3 και 2.4 αντίστοιχα υπολογίζουν το χρόνο ολοκλήρωσης της πρώτης εργασίας και όλων των υπόλοιπων εργασιών σε όλα τα υπόλοιπα μηχανήματα. Τέλος, η 2.5 υπολογίζει το makespan C_{max}. Επομένως, σύμφωνα με τους παραπάνω τύπους, η βέλτιστη λύση είναι η εύρεση της π^* από το σύνολο αντιμεταθέσεων, τέτοια ώστε να ισχύει:

$$C_{max}(\pi^*) \leq C_{max}(\pi), \pi \in \Pi \tag{2.6}$$

Συνοψίζοντας, επιλύοντας το παραπάνω πρόβλημα βελτιστοποίησης PFSP , καταλήγουμε σε μια σειρά εργασιών τέτοια ώστε να ελαχιστοποιείται η τιμή της αντικειμενικής συνάρτησης του makespan, η ελαχιστοποίηση δηλαδή του χρόνου επεξεργασίας της τελευταίας εργασίας στην τελευταία μηχανή. Γενικότερα, η αντικειμενική συνάρτηση δεν περιλαμβάνει αποκλειστικά το χρόνο επεξεργασίας των εργασιών αλλά συνήθως περιλαμβάνει και άλλους παράγοντες που πρέπει να συνυπολογίσει και να βελτιστοποιήσει κάποιος κατά την παραγωγή. Ο χρόνος που τα μηχανήματα είναι σε κατάσταση αναμονής, το κόστος λειτουργίας των μηχανημάτων καθώς και η αποδοτικότητα του εργατικού δυναμικού είναι μερικοί από τους παράγοντες που θεωρητικά πρέπει κάποιος να συνυπολογίσει στην αντικειμενική συνάρτηση. Επειδή όμως στην πράξη είναι αρκετά δύσκολο αυτό να συμβεί, έχουν οριστεί τρία κριτήρια λήψης αποφάσεων κατά το χρονοπρογραμματισμό εργασιών. Αυτά είναι ο χρόνος διεκπεραίωσης (turnaround) , ο χρόνος δηλαδή που χρειάζεται για να ολοκληρωθεί μια εργασία, η επικαιρότητα (timeliness) που είναι η

συμμόρφωση μιας εργασίας σε μία προκαθορισμένη προθεσμία και η απόδοση (throughput), η οποία μετράει το έργο που παράγεται σε ένα χρονικό διάστημα. Καταλαβαίνει κάποιος ότι τα πρώτα δύο κριτήρια αφορούν μεμονωμένες εργασίες ενώ το τρίτο αφορά ολόκληρη τη διαδικασία παραγωγής. Η βελτιστοποίηση της απόδοσης σχετίζεται με την ελαχιστοποίηση του makespan, καθώς, αν ορίσουμε την απόδοση ως:

$$Efficiency = \frac{Produced\ Work}{Time} \tag{2.7}$$

και γνωρίζοντας πως το παραγόμενο έργο σε n εργασίες είναι σταθερό, τότε όσο πιο πολύ μικραίνει το χρονικό διάστημα, άρα το makespan, τόσο αυξάνεται η απόδοση. Εν κατακλείδι, στα περισσότερα προβλήματα χρονοπρογραμματισμού εργασιών, κυριότερος σκοπός είναι η ελαχιστοποίηση του makespan. Ο αριθμός των πιθανών λύσεων των προβλημάτων χρονοπρογραμματισμού εργασιών είναι αρκετά μεγάλος ώστε να εξεταστεί μία προς μία κάθε περίπτωση. Για την επίλυση τέτοιων προβλημάτων χρησιμοποιούνται τεχνικές που ονομάζονται ευρετικοί αλγόριθμοι. Τέτοιοι αλγόριθμοι βρίσκουν λύσεις ανάμεσα σε αρκετές εφικτές αλλά δεν εγγυώνται εξ'αρχής ότι θα βρουν τη βέλτιστη, μπορούμε δηλαδή να πούμε ότι προσεγγίζουν τη βέλτιστη λύση. Παρόλα αυτά, οι λύσεις που βρίσκουν είναι πολύ κοντά στη βέλτιστη και μάλιστα, η προσέγγιση γίνεται με εύκολο τρόπο και σε σύντομο χρονικό διάστημα. Για αυτούς τους αλγορίθμους αλλά και για βελτιωμένες εκδοχές τους, όπως είναι οι μεθευρετικοί αλγόριθμοι, θα γίνει λόγος στο επόμενο κεφάλαιο.

Κεφάλαιο 3

Αλγόριθμοι Βελτιστοποίησης

Στην υπολογιστική θεωρία, συχνά, αντιμετωπίζουμε προβλήματα πολύ μεγάλου όγκου. Αυτά μπορεί να είναι προβλήματα τα οποία έχουν μη πεπερασμένο αριθμό πιθανών λύσεων, δεν είναι γνωστός ο τρόπος να βρεθεί η βέλτιστη λύση ή το υπολογιστικό κόστος για τον προσδιορισμό όλων των πιθανών λύσεων είναι τεράστιο, και συνεπώς, ο υπολογισμός ανέφικτος. Οι ευρετικοί αλγόριθμοι είναι ένας τρόπος προσέγγισης αυτού του είδους των υπολογιστικών προβλημάτων. Είναι ικανοί να παρουσιάσουν εφικτές και κοντά στον βέλτιστο χώρο, λύσεις. Ακόμη και σε περιπτώσεις στις οποίες υπάρχει βέλτιστη λύση, δεν υπάρχει η εγγύηση πως θα βρεθεί. Για αυτούς τους λόγους θεωρούνται προσεγγιστικές και μη ακριβείς μέθοδοι. Είναι εξαιρετικά πρακτικοί, καθώς, θυσιάζοντας σε ακρίβεια, μειώνουν αισθητά την υπολογιστική ισχύ που απαιτείται για την προσέγγιση της λύσης. Οι ευρετικοί αλγόριθμοι χωρίζονται σε:

- Αλγόριθμοι Τοπικής Αναζήτησης (Local Search Algorithms)
- Αλγόριθμοι Απληστίας (Greedy Algorithms)

3.1 Αλγόριθμοι Απληστίας

Οι αλγόριθμοι απληστίας είναι ένας τύπος ευρετικών αλγορίθμων όπου σε κάθε βήμα επιλέγεται η τοπική βέλτιστη λύση με την ελπίδας εύρεσης της ολικής βέλτιστης λύσης. Αυτός ο αλγόριθμος λειτουργεί επιλέγοντας σε κάθε βήμα την καλύτερη επιλογή χωρίς να λαμβάνεται υπόψη ο αντίκτυπος που θα έχει αυτή η επιλογή στην λύση. Αυτή η προσέγγιση συχνά χρησιμοποιείται όταν η εύρεση της ολικής βέλτιστης λύσης είναι υπολογιστικά ακριβή ή αδύνατη.

Ένα παράδειγμα ενός άπληστου αλγορίθμου είναι το πρόβλημα των κερμάτων. Ας υποθέσουμε ότι δίνεται ένα σετ νομισμάτων διαφορετικών αξιών και ζητείται ο ελάχιστος αριθμός νομισμάτων που αθροίζουν σε ένα συγκεκριμένο ποσό. Ο άπληστος αλγόριθμος για αυτό το πρόβλημα θα περιλαμβάνει την επανειλημμένη επιλογή του νομίσματος με τη μεγαλύτερη ονομαστική αξία που είναι μικρότερο από το υπόλοιπο ποσό μέχρι να επιτευχθεί το ποσό-στόχος.

Για παράδειγμα, δίνεται το σύνολο των νομισμάτων $1, 5, 10, 25$ το ποσό - στόχος ίσο με 63. Ο αλγόριθμος θα ξεκινήσει επιλέγοντας το μεγαλύτερο νόμισμα που είναι μικρότερο από το 63, δηλαδή το 25. Το υπόλοιπο ποσό είναι 38, οπότε ο αλγόριθμος θα επέλεγε ξανά το 25. Το υπόλοιπο ποσό είναι τώρα 13, επομένως ο αλγόριθμος θα επέλεγε το 10. Το υπόλοιπο ποσό είναι τώρα 3, επομένως ο αλγόριθμος θα επέλεγε το 1 τρεις φορές για να επιτύχει το πόσο - στόχο.

Χρησιμοποιώντας τον άπληστο αλγόριθμο, επιτεύχθηκε το ποσό-στόχος των 63 χρησιμοποιώντας έξι νομίσματα $(25 + 25 + 10 + 1 + 1 + 1)$. Αυτός είναι ο ελάχιστος αριθμός κερμάτων που απαιτείται για να επιτευχθεί το στοχευόμενο ποσό χρησιμοποιώντας αυτό το σύνολο κερμάτων.

Ωστόσο, αξίζει να σημειωθεί ότι οι άπληστοι αλγόριθμοι δεν βρίσκουν πάντα τη βέλτιστη λύση και μπορεί να υπάρχουν περιπτώσεις όπου η τοπικά βέλτιστη επιλογή σε κάθε βήμα δεν οδηγεί στην ολικά βέλτιστη λύση.

3.2 Αλγόριθμοι τοπικής αναζήτησης

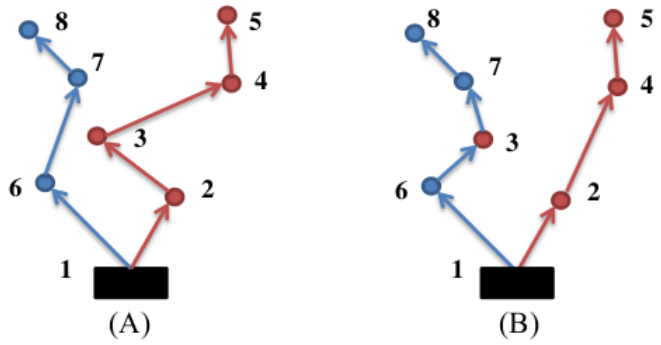
Οι αλγόριθμοι τοπικής αναζήτησης [4] (local search algorithms) αποτελούν μια υποκατηγορία των ευρετικών αλγορίθμων. Η τοπική αναζήτηση ξεκινάει από μία ήδη υπάρχουσα λύση του προβλήματος και προσπαθεί να βρει βέλτιστες λύσεις στη γειτονική περιοχή των εφικτών λύσεων. Ως γειτονιά λύσεων ορίζουμε τις εφικτές λύσεις με την μικρότερη δυνατή διαφορά από την υπάρχουσα. Λόγω του τρόπου δομής αυτών των αλγορίθμων, η επιλογή της αρχικής λύσης είναι εξέχουσας σημασίας, καθώς μπορεί να οδηγήσει, σύντομα, σε σύγκλιση. Εν αντιθέσει, η επιλογή μια λύσης που απέχει πολύ από το βέλτιστο, ως αρχική, θα καθυστερήσει να συγκλίνει. Σημαντική διαφορά σε σχέση με άλλους ευρετικούς αλγορίθμους, είναι πως οι αλγόριθμοι τοπικής αναζήτησης δεν παράγουν την αρχική τους λύση. Οπότε συχνά χρησιμοποιούνται σε συνδυασμό με κάποιον άλλον αλγόριθμο απληστίας, για την παραγωγή αυτής. Παρόλα αυτά, μπορεί να γίνει χρήση τυχαίας διαδικασίας επιλογής της αρχικής λύσης. Η εφαρμογή γίνεται σε ένα πρόβλημα βελτιστοποίησης, στο οποίο θα πρέπει να έχει δοθεί πρώτα ένα μέγεθος, που ζητάμε να μεγιστοποιήσουμε ή να ελαχιστοποιήσουμε (χρηματικό κόστος, χρόνος, απόσταση κτλ.).

3.2.1 Αλγόριθμος 1-0 Επανατοποθέτηση

Ο αλγόριθμος 1-0 επανατοποθέτηση (1-0 Relocate) είναι ένας αλγόριθμος τοπικής αναζήτησης. Η μέθοδος αυτή βασίζεται στην απλή ιδέα της διαγραφής ενός κόμβου από μία διαδρομή και επανατοποθέτησης του σε μια άλλη διαδρομή με καλύτερο κόστος. Σε αυτόν τον αλγόριθμο ένας κόμβος, επανατοποθετείται και έπειτα ελέγχεται αν η αλλαγή αυτή έχει βελτιώσει την λύση. Εάν την έχει βελτιώσει, ορίζεται η καινούρια λύση ως βέλτιστη, απορρίπτεται η παλαιά, και η διαδικασία επαναλαμβάνεται έως ότου δεν βρίσκονται άλλες τέτοιες επανατοποθετήσεις.

Παρακάτω παρουσιάζονται αναλυτικά τα βήματα του αλγορίθμου: Δεδομένης μιας αρχικής εφικτής λύσης.

- **Βήμα 1ο:** Επιλογή κόμβου i από το σύνολο των κόμβων που δεν έχουν ελεγχθεί.
- **Βήμα 2ο:** Τοποθέτηση του κόμβου i σε διαφορετική διαδρομή και αφαίρεση του από την παλαιότερη. Εάν δεν υπάρχει άλλη διαθέσιμη διαδρομή: ορισμός κόμβου i ως ελεγμένο και επιστροφή στο βήμα 1.
- **Βήμα 3ο:** Εάν με την επανατοποθέτηση αυτή παραβιάζεται κάποιος από τους περιορισμούς, επιστροφή στο βήμα 2.
- **Βήμα 4ο:** Υπολογισμός για τα νέα κόστη των διαδρομών. Αν τα νέα κόστη είναι χαμηλότερα: επανατοποθέτηση του κόμβου i στη θέση αυτή, ορισμός όλων των κόμβων ως μη ελεγμένων και επιστροφή στο βήμα 1. Αν είναι υψηλότερα: επιστροφή στο βήμα 2.
- **Βήμα 5ο:** Επανάληψη της διαδικασίας έως ότου ελεγχθούν όλοι οι κόμβοι.



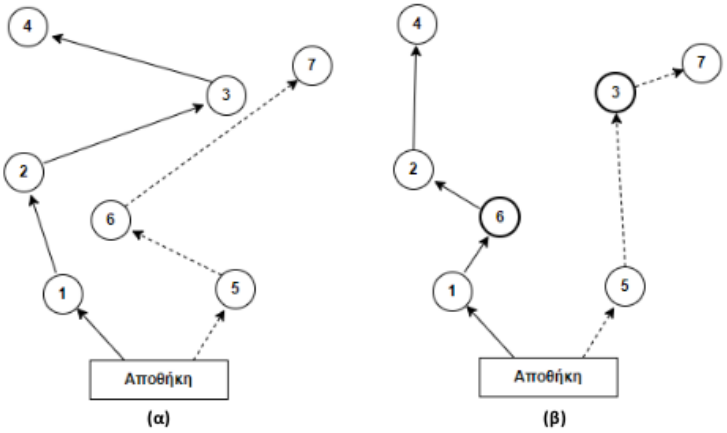
Σχήμα 3.1: Εφαρμογή του 1-0 Relocate

Στο παράδειγμα της εικόνας φαίνονται οι αρχικές διαδρομές (A) των δύο οχημάτων ξεκινώντας από τον κόμβο (1). Η διαδρομή του μπλε οχήματος είναι η [1→6→7→8] και η διαδρομή του κόκκινου η [1→2→3→4→5]. Ο αλγόριθμος 1-0 Relocate ξεκινάει μία επαναληπτική διαδικασία κατά την οποία επανατοποθετεί έναν κόμβο της μίας διαδρομής

σε κάθε δυνατή θέση της άλλης και ελέγχει αν η αλλαγή αυτή προσφέρει κάποια μείωση στο κόστος. Όπως είναι εύκολο να παρατηρηθεί και οπτικά η μόνη επανατοποθέτηση που μειώνει το κόστος είναι η μετακίνηση του κόμβου (3) ανάμεσα στους (6) και (7) (B). Οι διαδρομές καταλήγουν για το μπλε όχημα να είναι [1→6→3→7→8] και για το κόκκινο [1→2→4→5]. Ενώ το κόστος της μπλε διαδρομής αυξάνεται, συνολικά το κόστος και των δύο μειώνεται λόγω της μεγάλης διαφοράς στο κόστος της κόκκινης.

3.2.2 Αλγόριθμος 1-1 Ανταλλαγή

Ο αλγόριθμος 1-1 Ανταλλαγή (1-1 Exchange) έχει την ίδια φιλοσοφία με τον αλγόριθμο 1-0 Επανατοποθέτηση, ήτοι την επανατοποθέτηση κόμβων γραφήματος και έλεγχος αν αυτή η αλλαγή οδηγεί σε καλύτερο αποτέλεσμα. Συγκεκριμένα, σε αυτόν τον αλγόριθμο επιλέγονται δύο τυχαίοι κόμβοι από το γράφημα και γίνεται ανταλλαγή των θέσεων τους. Στην Εικόνα 3.2 παρουσιάζεται ένα απλό παράδειγμα 1-1 Ανταλλαγής για την καλύτερη κατανόηση της λειτουργίας.



Σχήμα 3.2: Εφαρμογή του 1-1 Exchange

Είναι εμφανές πως αν από τις διαδρομές τις αρχικής λύσης γίνει η ανταλλαγή των κόμβων 3 και 6 το συνολικό κόστος ελαχιστοποιείται.

3.3 Μεθευρετικοί Αλγόριθμοι

Στην επιστήμη των υπολογιστών και συγκεκριμένα στον τομέα της βελτιστοποίησης, μια μεθευρετική διαδικασία ονομάζεται μια διαδικασία που έχει σχεδιαστεί για να βρίσκει, να παράγει ή να συντονίζει μια ευρετική διαδικασία με σκοπό να παρέχει μια αρκετά καλή λύση σε ένα πρόβλημα βελτιστοποίησης, ειδικά με ελλιπείς ή ατελείς πληροφορίες ή περιορισμένη διαθέσιμη υπολογιστική ισχύ [5][6]. Οι μεθευρετικές διαδικασίες μπορεί να κάνουν σχετικά λίγες υποθέσεις σχετικά με το πρόβλημα βελτιστοποίησης που επιλύεται και έτσι μπορούν να χρησιμοποιηθούν για μια ποικιλία προβλημάτων ειδικά όταν ο χώρος του προβλήματος είναι πολύ μεγάλος. Στην παρούσα εργασία θα παρουσιαστούν: ο Γενετικός Αλγόριθμος [7], ο αλγόριθμος Προσομοιωμένης Ανόπτησης [8], ο αλγόριθμος Βελτιστοποίησης Αποικίας Μυρμηγκιών [9] και τέλος ο αλγόριθμος Βελτιστοποίησης Σμήνους Σωματιδίων [1] στον οποίο θα δωθεί και περισσότερη έμφαση.

3.3.1 Γενετικοί Αλγόριθμοι

Οι Γενετικοί Αλγόριθμοι [7] είναι ένας τύπος ευρετικού αλγορίθμου που χρησιμοποιεί τις αρχές της φυσικής επιλογής και της γενετικής για την εύρεση βέλτιστων λύσεων σε προβλήματα. Ο αλγόριθμος λειτουργεί δημιουργώντας έναν πληθυσμό υποψήφιων λύσεων και στη συνέχεια βελτιώνοντας επαναληπτικά αυτές τις λύσεις μέσω επιλογής, διασταύρωσης και μετάλλαξης.

Σε έναν γενετικό αλγόριθμο, κάθε υποψήφια λύση αναπαρίσταται ως χρωμόσωμα, το οποίο αποτελείται από μια ακολουθία γονιδίων που κωδικοποιούν τις μεταβλητές του προβλήματος. Ο αλγόριθμος ξεκινά με τη δημιουργία ενός αρχικού πληθυσμού χρωμοσωμάτων, τα οποία στη συνέχεια αξιολογούνται και κατατάσσονται με βάση την καταλληλότητά τους ή την καταλληλότητά τους για το εκάστοτε πρόβλημα.

Στη συνέχεια, ο αλγόριθμος εφαρμόζει τελεστές επιλογής, διασταύρωσης και μετάλλαξης για τη δημιουργία νέων υποψήφιων λύσεων από τον τρέχοντα πληθυσμό. Η

επιλογή περιλαμβάνει την επιλογή των καταλληλότερων ατόμων από τον πληθυσμό που θα χρησιμοποιηθούν ως γονείς για την επόμενη γενιά. Η διασταύρωση περιλαμβάνει την ανταλλαγή γενετικών πληροφοριών μεταξύ δύο γονικών χρωμοσωμάτων για τη δημιουργία νέων απογόνων χρωμοσωμάτων. Η μετάλλαξη περιλαμβάνει την τυχαία τροποποίηση των γονιδίων ενός χρωμοσώματος για την εισαγωγή νέων γενετικών πληροφοριών.

Η διαδικασία της επιλογής, της διασταύρωσης και της μετάλλαξης επαναλαμβάνεται για πολλές γενιές μέχρι να βρεθεί μια ικανοποιητική λύση ή να ικανοποιηθεί ένα κριτήριο διακοπής.

Το πρόβλημα του πλανόδιου πωλητή μπορεί να προσεγγιστεί με έναν γενετικό αλγόριθμο. Σε αυτό το πρόβλημα, ένας πωλητής πρέπει να επισκεφθεί ακριβώς μία φορά ένα σύνολο πόλεων και να επιστρέψει στην πόλη εκκίνησης, ελαχιστοποιώντας τη συνολική απόσταση που διανύει. Για την επίλυση αυτού του προβλήματος με τη χρήση γενετικού αλγορίθμου, κάθε υποψήφια λύση θα αναπαρίσταται ως χρωμόσωμα που αποτελείται από μια ακολουθία επισκέψεων σε πόλεις.

Ο γενετικός αλγόριθμος θα ξεκινούσε δημιουργώντας έναν αρχικό πληθυσμό υποψήφιων λύσεων, οι οποίες θα αξιολογούνταν με βάση τη συνολική απόσταση που διένυσαν. Στη συνέχεια, ο αλγόριθμος θα επέλεγε τα καταλληλότερα άτομα από τον πληθυσμό που θα χρησιμοποιούνταν ως γονείς για την επόμενη γενιά. Οι επιλεγμένοι γονείς θα υποβάλλονταν σε διασταύρωση και μετάλλαξη για τη δημιουργία νέων υποψήφιων λύσεων, οι οποίες θα αξιολογούνταν και θα κατατάσσονταν με βάση την καταλληλότητά τους.

Η διαδικασία επιλογής, διασταύρωσης και μετάλλαξης θα επαναλαμβανόταν για πολλές γενιές μέχρι να βρεθεί μια ικανοποιητική λύση. Η τελική λύση θα αντιπροσωπεύει τη συντομότερη δυνατή διαδρομή που θα επισκέπτεται όλες τις πόλεις και θα επιστρέφει στην αρχική πόλη.

Οι γενετικοί αλγόριθμοι χρησιμοποιούνται σε πολλούς τομείς, όπως η βελτιστοποίηση, η μηχανική μάθηση και η τεχνητή νοημοσύνη. Είναι ιδιαίτερα χρήσιμοι όταν η εύρεση μιας βέλτιστης λύσης με άλλες τεχνικές βελτιστοποίησης είναι δύσκολη ή ανέφικτη.

3.3.2 Προσομοιωμένη Ανόπτηση

Η προσομοιωμένη ανόπτηση [8] είναι ένας τύπος ευρετικού αλγορίθμου που χρησιμοποιείται για την εύρεση βέλτιστων λύσεων σε πολύπλοκα προβλήματα. Ο αλγόριθμος είναι εμπνευσμένος από τη διαδικασία της ανόπτησης στη μεταλλουργία, όπου ένα μέταλλο θερμαίνεται και στη συνέχεια ψύχεται αργά για να επιτευχθεί η επιθυμητή κρυσταλλική δομή.

Στην προσομοιωμένη ανόπτηση, ο αλγόριθμος ξεκινά με μια αρχική λύση και στη συνέχεια διαταράσσει επανειλημμένα τη λύση και αξιολογεί τη νέα λύση. Οι διαταραχές γίνονται με μικρές αλλαγές στην τρέχουσα λύση και ο αλγόριθμος αποδέχεται τη νέα λύση εάν είναι καλύτερη από την τρέχουσα λύση. Εάν η νέα λύση είναι χειρότερη από την τρέχουσα λύση, ο αλγόριθμος μπορεί και πάλι να την αποδεχθεί με μια ορισμένη πιθανότητα, η οποία μειώνεται με την πάροδο του χρόνου.

Η πιθανότητα αποδοχής μιας χειρότερης λύσης ελέγχεται από μια παράμετρο που ονομάζεται θερμοκρασία, η οποία ξεκινάει υψηλή και μειώνεται με την πάροδο του χρόνου. Η υψηλή θερμοκρασία επιτρέπει στον αλγόριθμο να εξερευνήσει το χώρο λύσεων και να ξεφύγει από τα τοπικά βέλτιστα, ενώ η μειούμενη θερμοκρασία εξασφαλίζει ότι ο αλγόριθμος συγκλίνει σε ένα ολικό βέλτιστο.

Ένα παράδειγμα προσομοιωμένης ανόπτησης είναι το πρόβλημα του πλανόδιου πωλητή. Σε αυτό το πρόβλημα, ένας πωλητής πρέπει να επισκεφθεί ένα σύνολο πόλεων ακριβώς μία φορά και να επιστρέψει στην πόλη εκκίνησης, ελαχιστοποιώντας τη συνολική απόσταση που διανύει. Ο αλγόριθμος προσομοιωμένης ανόπτησης για αυτό το πρόβλημα θα ξεκινούσε με μια αρχική λύση, όπως ένας τυχαία διατεταγμένος κατάλογος επισκέψεων σε πόλεις. Στη συνέχεια, ο αλγόριθμος θα διατάρασσε τη λύση κάνοντας

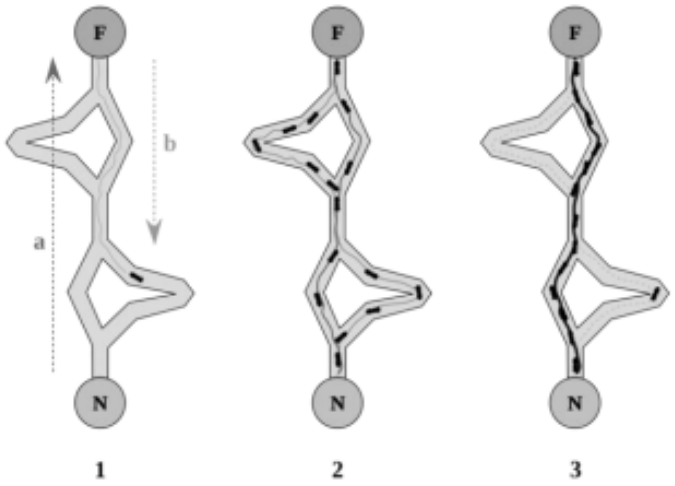
μικρές αλλαγές, όπως η ανταλλαγή δύο πόλεων στη λίστα. Η νέα λύση θα αξιολογείται με βάση τη συνολική απόσταση που διανύει και ο αλγόριθμος θα αποδέχονταν τη νέα λύση εάν ήταν καλύτερη από την τρέχουσα λύση. Εάν η νέα λύση είναι χειρότερη από την τρέχουσα λύση, ο αλγόριθμος μπορεί να την αποδεχτεί με μια ορισμένη πιθανότητα, η οποία μειώνεται με την πάροδο του χρόνου καθώς μειώνεται η θερμοκρασία.

Η διαδικασία της διαταραχής της λύσης και της αποδοχής νέων λύσεων θα επαναλαμβανόταν για πολλές επαναλήψεις, με τη θερμοκρασία να μειώνεται με την πάροδο του χρόνου. Η τελική λύση θα αντιπροσωπεύει τη συντομότερη δυνατή διαδρομή που επισκέπτεται όλες τις πόλεις και επιστρέφει στην αρχική πόλη. Η προσομοιωμένη ανόπτηση χρησιμοποιείται συχνά όταν η εύρεση μιας βέλτιστης λύσης με τη χρήση παραδοσιακών τεχνικών βελτιστοποίησης είναι δύσκολη ή ανέφικτη. Είναι ένα ισχυρό εργαλείο για την επίλυση σύνθετων προβλημάτων βελτιστοποίησης και έχει χρησιμοποιηθεί σε πολλούς τομείς, όπως η επιχειρησιακή έρευνα, η μηχανική και η τεχνητή νοημοσύνη.

3.3.3 Αλγόριθμος Βελτιστοποίησης Αποικίας Μυρμηγκιών

Ο Αλγόριθμος Βελτιστοποίησης Αποικίας Μυρμηγκιών [9] (Ant Colony Optimization) είναι εμπνευσμένος από τη μέθοδο που χρησιμοποιούν τα μυρμήγκια για να βρουν την πιο κοντινή διαδρομή για την τροφή τους. Αυτό επιτυγχάνεται χρησιμοποιώντας ίχνη φερομόνης σαν μια μορφή έμμεσης επικοινωνίας. Τα μυρμήγκια εναποθέτουν ίχνη φερομόνης καθώς ταξιδεύουν προς την πηγή της τροφής και άλλα μυρμήγκια ακολουθούν αυτό το μονοπάτι της φερομόνης. Τα μυρμήγκια που τυχαίνει να ακολουθήσουν τη συντομότερη διαδρομή προς την τροφή θα δημιουργήσουν ένα ισχυρό μονοπάτι με υψηλή συγκέντρωση φερομόνης γρηγορότερα από αυτά που επέλεξαν μια μακρύτερη διαδρομή. Αφού η περισσότερη φερομόνη προσελκύει καλύτερα τα μυρμήγκια, όλο και περισσότερα επιλέγουν την πιο σύντομη διαδρομή, μέχρι τελικά όλα ή σχεδόν όλα να την ακολουθούν.

Στην εικόνα 3.3 φαίνεται η διαδικασία εντοπισμού της συντομότερης διαδρομής προς την τροφή των μυρμηγκιών με την υψηλή συγκέντρωση φερομόνης.



Σχήμα 3.3: Απεικόνιση της διαδικασίας εντοπισμού της συντομότερης διαδρομής προς την τροφή των πραγματικών μυρμηγκιών

Οι βασικές διαδικασίες ενός αλγορίθμου ACO είναι η επιλογή μίας εκ των πιθανών εναλλακτικών επιλογών σε κάθε σημείο απόφασης, η διαδικασία ενημέρωσης της φερομόνης των πιθανών επιλογών (εξάτμιση και προσθήκη φερομόνης) και η διαδικασία υπολογισμού της καλύτερης λύσης του προβλήματος βάσει των τιμών φερομόνης.

Η πιθανότητα να επιλεγεί κάθε ένα από τα πιθανά τόξα υπολογίζεται με τη σχέση 3.1

$$p_{i,j} = \frac{[\tau_{i,j}]^{\alpha}[\eta_{i,j}]^{\beta}}{\sum_{l=1}^{M}[\tau_{i.j}]^{\alpha}[\eta_{i,j}]^{\beta}} \tag{3.1}$$

όπου: $p_{i,j}$ η πιθανότητα να επιλεγεί η μετάβαση στον κόμβο j από τον κόμβο i, $\tau_{i,j}$ η συγκέντρωση φερομόνης στο τόξο (i,j), $\eta_{i,j}$ ένας ευρετικός παράγοντας που ευνοεί τόξα χαμηλότερου κόστους, α και β είναι εκθετικές παράμετροι που ελέγχουν τη σχετική σημαντικότητα της φερομόνης και του ευρετικού παράγοντα στην τυχαία επιλογή.

Ο ευρετικός παράγοντας $\eta_{i,j}$ ονομάζεται ορατότητα ή οπτικό εύρος και η τιμή του υπολογίζεται ως το αντίστροφο του κόστους, $c_{i,j}$, του τόξου (i,j):

$$\eta_{i,j} = \frac{1}{c_{i,j}} \tag{3.2}$$

Οι παράμετροι α και β, ελέγχουν το πόσο σημαντικό ρόλο παίζουν η ποσότητα της φερομόνης και η ορατότητα αντίστοιχα, για την απόφαση κάθε μυρμηγκιού. Αν $\alpha \gg \beta$ τότε ο αλγόριθμος θα επιλέγει βασιζόμενος κυρίως στις πληροφορίες που αντιπροσωπεύει η φερομόνη, ενώ αν $\beta \gg \alpha$, τότε ο αλγόριθμος θα δρα ως ένας άπληστος ευρετικός, που επιλέγει τις φθηνότερες εναλλακτικές αδιαφορώντας για τις επιπτώσεις που αυτές οι επιλογές έχουν στην ποιότητα της τελικής λύσης.

Η εξάτμιση φερομόνης από όλο το γράφημα λαμβάνει χώρα με το πέρας της επανάληψης, όταν δηλαδή θα έχουν κατασκευάσει λύση όλα τα μυρμήγκια. Αυτή γίνεται με τη βοήθεια της παραμέτρου ρ και με τη σχέση 3.3

$$\tau_{i,j} \leftarrow (1-\rho)\tau_{i,j} \tag{3.3}$$

όπου: ρ είναι ο παράγοντας εξάτμισης ή επιμονής φερομόνης($0 \leq \rho \leq 1$) . Η εξάτμιση δίνει τη δυνατότητα στην αποικία να ξεχάσει τις κακές ή λιγότερο καλές λύσεις και αυξάνει την πιθανότητα να επιλέγονται οι καλύτερες λύσεις. Για τιμές του $\rho \to 0$, μικρή ποσότητα φερομόνης εξατμίζεται, και η ταχύτητα σύγκλισης είναι μικρότερη. Για τιμές $\rho \to 1$, εξατμίζεται μεγάλη ποσότητα με αποτέλεσμα γρηγορότερη σύγκλιση.

Η προσθήκη φερομόνης μπορεί να γίνει επιπρόσθετα ή και μόνο στις επιλεγμένες διαδρομές του μυρμηγκιού που έδωσε την καλύτερη λύση στην τρέχουσα επανάληψη, στις επιλογές που έχουν δώσει την καλύτερη λύση συνολικά έως την τρέχουσα επανάληψη, στις επιλεγμένες διαδρομές των n καλύτερων λύσεων, όπως επίσης μπορεί να γίνει και συνδυασμός των παραπάνω. Η προσθήκη φερομόνης υπολογίζεται από τη σχέση 3.4.

$$\tau_{i,j} = \tau_{i,j} = \sum_{k=1}^{m} \Delta\tau_{i,j}^{k} \tag{3.4}$$

Όπου το $\Delta\tau_{i,j}^k$ είναι η ποσότητα φερομόνης που αφήνει κάθε μυρμήγκι, k, στα τόξα που πέρασε και δίνεται από τη σχέση 3.5.

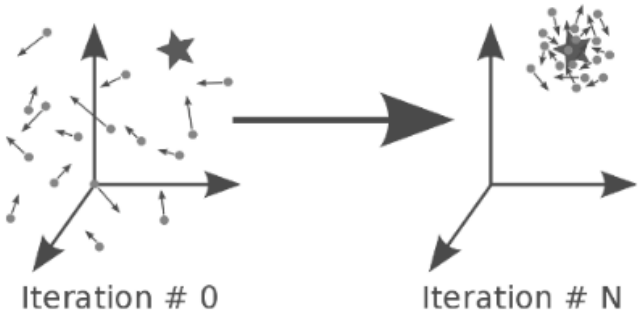
$$\Delta\tau_{i,j}^k = \begin{cases} \frac{1}{C^k}, & \text{Έαν το τόξο } (i,j) \text{ έχει επιλεγεί από το } k \\ 0, & \text{Αλλιώς} \end{cases} \tag{3.5}$$

Όπου C^k είναι το κόστος της λύσης που έχει δημιουργήσει το καλύτερο μυρμήγκι k. Όσο καλύτερη είναι η λύση, τόσο περισσότερη φερομόνη αφήνεται και άρα έχει περισσότερες πιθανότητες να επιλεγεί στην επόμενη επανάληψη.

3.3.4 Αλγόριθμος βελτιστοποίησης σμήνους σωματιδίων

Ο αλγόριθμος βελτιστοποίησης σμήνους σωματιδίων [1] (Particle Swarm Optimization ή PSO) είναι μια υπολογιστική τεχνική που αναπτύχθηκε από τους Kennedy και Eberhart το 1995, επηρεασμένοι από τις κοινωνικές συμπεριφορές μεμονωμένων οργανισμών εντός ενός ευρύτερου συνόλου. Πιο συγκεκριμένα, πηγή έμπνευσης αποτέλεσε ένα σμήνος πουλιών, ώστε να αναζητήσουν τον τρόπο με τον οποίο τα πουλιά πετάνε σε ομάδες αλλά και της δυνατότητας της ξαφνικής αλλαγής κατεύθυνσης του σμήνους χωρίς να χαλάσει ο σχηματισμός του. Χαρακτηριστικό αυτού του μοντέλου είναι η ιδιότητα των επιμέρους οργανισμών ή – γενικότερα – σωματιδίων να θυμούνται τη βέλτιστη θέση που έχουν βρεθεί αλλά και την ικανότητά τους να μεταδίδουν αυτή την πληροφορία στα υπόλοιπα σωματίδια του σμήνους. Συνεπώς κάθε σωματίδιο παίρνει ατομικές αποφάσεις αλλά ταυτόχρονα, εξ'αιτίας της επικοινωνίας μεταξύ τους, παίρνονται και ομαδικές αποφάσεις. Ας ορίσουμε το πρόβλημα που καλούμαστε να λύσουμε σαν ένα κλειστό χώρο, εντός του οποίου βρίσκεται μία βέλτιστη λύση. Ο αλγόριθμος αρχικά βρίσκει κάποιες πιθανές λύσεις, τις οποίες θα φανταστούμε σαν επιμέρους διανύσματα ενός συνόλου διανυσμάτων που βρίσκονται στο χώρο αυτό. Στη συνέχεια, οι λύσεις αυτές κινούνται στο χώρο με βάση ένα μαθηματικό τύπο που εξαρτάται από τη θέση, την κατεύθυνση και την ταχύτητα του κάθε διανύσματος. Για να διαλέξει το προς τα που θα κινηθεί το κάθε

διάνυσμα, έχοντας μνήμη της βέλτιστης θέσης που βρέθηκε, ταυτόχρονα συνυπολογίζει και τις βέλτιστες θέσεις του που βρίσκουν τα υπόλοιπα διανύσματα στο χώρο. Αυτή η πληροφορία ανανεώνεται κάθε φορά που κάποιο διάνυσμα βρίσκει μια νέα βέλτιστη θέση, ώστε να ενημερώνονται όλα τα διανύσματα για κάθε εξέλιξη. Με τον τρόπο αυτόν, το σύνολο κινείται ολοένα και πιο κοντά προς τη βέλτιστη λύση.



Σχήμα 3.4: Οπτικοποίηση επίλυσης ενός προβλήματος μέσω του PSO, μετά από: (Αριστερά) 0 επαναλήψεις και (Δεξιά) N επαναλήψεις

Συνοψίζοντας τα παραπάνω, κάθε σωματίδιο του σμήνους έχει μια θέση ξ και μια ταχύτητα u, τα οποία σε κάθε i βήμα επανάληψης του αλγορίθμου μεταβάλλονται. Επίσης σε κάθε βήμα, το σωματίδιο αποθηκεύει την τρέχουσα θέση του, την αμέσως προηγούμενη ταχύτητά του, την καλύτερη θέση που αυτό έχει βρεθεί (personal best) καθώς και τη βέλτιστη θέση που έχει βρεθεί από οποιοδήποτε σωματίδιο του σμήνους (global best). Ως βέλτιστη θεωρούμε τη θέση x^* η οποία δίνει την καλύτερη τιμή στην αντικειμενική συνάρτηση του προβλήματός μας. Οι εξισώσεις που περιγράφουν τη θέση x και την ταχύτητα u ενός σωματιδίου σε μία διάσταση κατά επανάληψη k, είναι:

$$u_{id}(k) = \omega u_{id}(k-1) + c_1 r_1 \big(p_{id}(k-1) - x_{id}(k-1)\big) + c_2 r_2 \big(p_{gd}(k-1) - x_{id}(k-1)\big) \quad (3.6)$$

$$x_{id}(k) = x_{id}(k-1) + u_{id}(k) \quad (3.7)$$

Όπου:

$i = 1, 2, ..., p$: ο αριθμός των σωματιδίων του σμήνους

$d = 1, 2, ..., D$: η διάσταση που βρίσκεται το σωματίδιο

p_{id}: η καλύτερη θέση που έχει βρεθεί το συγκεκριμένο σωματίδιο

p_{gd}: η καλύτερη θέση που έχει βρεθεί οποιοδήποτε σωματίδιο

ω: η σταθερά αδράνειας

c_1, c_2: σταθερές επιτάχυνσης

r_1, r_2: τυχαίοι αριθμοί του συνόλου $[0, 1]$

Η διαδικασία αυτή επαναλαμβάνεται έως ότου βρεθεί η τιμή της αντικειμενικής συνάρτησης με το επιθυμητό περιθώριο σφάλματος ($\varepsilon <$ Περιθώριο Σφάλματος) ή ο αριθμός των επαναλήψεων φτάσει στο μέγιστο που έχει οριστεί από τον προγραμματιστή. Το κριτήριο για να σταματήσει ο αλγόριθμος ορίζεται ανάλογα με το πρόβλημα που τίθεται προς επίλυση. Αναλύοντας τους παραπάνω τύπους, προκύπτει πως η θέση του σωματιδίου εξαρτάται από την προηγούμενη του θέση και την ταχύτητα του – πράγμα που είναι λογικό για την κίνηση ενός σώματος – ενώ η ταχύτητα του κάθε σωματιδίου υπολογίζεται από τρεις παράγοντες:

- Την ταχύτητα του κατά την προηγούμενη επανάληψη, που, όπως ειπώθηκε, έχει αποθηκευτεί στη μνήμη του και δυσκολεύει το σωματίδιο στο να κάνει μεγάλες αλλαγές κατεύθυνσης ανά επανάληψη.

- Τη βέλτιστη προσωπική του θέση μέχρι και την προηγούμενη επανάληψη, ώστε να μην απομακρυνθεί πολύ από τη γειτονιά που αυτή βρέθηκε και κάνει άστοχες κινήσεις.

- Τη βέλτιστη θέση του σμήνους μέχρι και την προηγούμενη επανάληψη. Έτσι θα ξέρει προς τα που θα βρει μια καλύτερη θέση από τη βέλτιστη προσωπική του.

Ο αλγόριθμος αυτός χρησιμοποιείται αρκετά ως μέθοδος βελτιστοποίησης , κυρίως τα τελευταία χρόνια, λόγω των πλεονεκτημάτων που έχει έναντι των υπόλοιπων αλγόριθμων βελτιστοποίησης. Κάποια από αυτά είναι:

- Δε χρειάζεται ο υπολογισμός ολοκληρωμάτων ή παραγώγων. Η λύση επέρχεται από την ιδιότητα των σωματιδίων να κρατάνε πληροφορίες σχετικές με τη βέλτιστη λύση και να τις μοιράζονται μεταξύ τους, κάτι που αποτρέπει τον εγκλωβισμό της τελικής λύσης σε τοπικά ελάχιστα καθώς υπάρχει μια συνολική εικόνα των πιθανών λύσεων .

- Ο κώδικας είναι απλός και εύκολος στην κατάστρωσή του, αποτελείται από λίγες γραμμές κώδικα και δε χρειάζονται εξεζητημένες συναρτήσεις για να υλοποιηθεί.

- Η ευελιξία του, καθώς μπορεί να χρησιμοποιηθεί και για στοχαστικά προβλήματα, αφού δεν εξαρτάται εξ'ολοκλήρου από την αντικειμενική συνάρτηση.

- Εξάγει ελάχιστα έως και καθόλου συμπεράσματα για τη φύση του προβλήματος, γεγονός που επιτρέπει την αναζήτηση λύσεων σε ένα μεγάλο σύνολο υποψήφιων λύσεων.

Παρά τα πολλά θετικά του, ο αλγόριθμος σμήνους σωματιδίων δεν εγγυάται ότι θα βρεθεί η βέλτιστη λύση του προβλήματος. Επίσης, για πολυδιάστατους χώρους πιθανών λύσεων μπορεί να εγκλωβιστεί σε τοπικά βέλτιστα που απέχουν από τη βέλτιστη λύση και τέλος έχει χαμηλό ποσοστό σύγκλισης κατά την επαναληπτική διαδικασία.

Κεφάλαιο 4

Εφαρμογή

Υλοποιήθηκε ο αλγόριθμος PSO με τοπική αναζήτηση 1-0 Επανατοποθέτηση και 1-1 Ανταλλαγή για το πρόβλημα χρονοπρογραμματισμού αντιμετάθεσης συνεχούς ροής με κριτήριο το makespan. Για τον έλεγχο της επίδοσης, ο αλγόριθμος εφαρμόστηκε στα παραδείγματα αναφοράς της βιβλιογραφίας του Taillard [2]. Τα παραδείγματα αυτά αποτελούν σημείο αναφοράς (Benchmark) για προβλήματα χρονοπρογραμματισμού αντιμετάθεσης συνεχούς ροής αφού σε κάποια από αυτά έχουν βρεθεί οι βέλτιστες λύσεις και έτσι μπορεί να μετρηθεί με ακρίβεια η απόκλιση από τα ολικά ελάχιστα.

Τα παραδείγματα αυτά αποτελούνται από τους χρόνους επεξεργασίας των J εργασιών (jobs) από τις M μηχανές (machines). Οι συνδυασμοί $J \times M$ που χρησιμοποιήθηκαν είναι οι εξής: 20x5, 20x10, 20x20, 50x5, 50x10, 50x20, 100x5, 100x10, 100x20, 200x10, 200x20, 500x20.

4.1 Μοντελοποίηση προβλήματος και υπολογισμός makespan

Κάθε συνδυασμός $J \times M$ περιέχει 10 διακριτά παραδείγματα και καθένα είναι της παρακάτω μορφής:

$$T = \underbrace{\begin{bmatrix} t_{1,1} & t_{2,1} & \cdots & t_{J,1} \\ t_{1,2} & t_{2,2} & \cdots & t_{J,2} \\ \vdots & \vdots & \ddots & \vdots \\ t_{1,M} & t_{2,M} & \cdots & t_{J,M} \end{bmatrix}}_{Number of Jobs} \Bigg\} Number of Machines \tag{4.1}$$

Όπου κάθε στοιχείο $t_{i,j}$ του πίνακα αντιστοιχεί στον χρόνο που χρειάζεται η μηχανή j να ολοκλήρωση την εργασία i. Η επεξεργασία γίνεται σειριακά και κάθε εργασία πρέπει να περάσει από όλες τις μηχανές πριν θεωρηθεί ολοκληρωμένη. Έτσι το πρόβλημα είναι να βρεθεί εκείνη η σειρά των εργασιών $[J_1, J_2, ...]$ η οποία ελαχιστοποιεί το συνολικό χρόνο ολοκλήρωσης.

Για να ξεκινήσει η i μηχανή την επεξεργασία της j εργασίας θα πρέπει να τηρούνται δύο προϋποθέσεις: (α) η i μηχανή να είναι διαθέσιμη, δηλαδή να μην επεξεργάζεται την $j-1$ εργασία, εφόσον αυτή υπάρχει και (β) η εργασία j να έχει ολοκληρωθεί από την $i-1$ μηχανή. Συνεπώς, ο χρόνος που θα χρειαστεί η i μηχανή για να ολοκληρώσει την j εργασία είναι ο χρόνος που χρειάζεται η μηχανή για την εργασία αυτή συν τον χρόνο που απαιτείται για να φτάσει η εργασία j στην μηχανή i. Ο χρόνος αυτός καθορίζεται από το μεγαλύτερο χρόνο των δύο προϋποθέσεων.

Ο πίνακας $C \in \mathbb{R}^{M \times J}$ είναι ο πίνακας κόστους, τα στοιχεία του οποίου αποτελούνται από το συνολικό χρόνο που απαιτείται για την ολοκλήρωση των μέχρι τότε εργασιών.

Παραδείγματος χάριν, το στοιχείο $C(i,j)$ είναι ο συνολικός χρόνος που χρειάζεται για να ολοκληρωθούν όλες οι εργασίες μέχρι την $j-1$ συν το χρόνο μέχρι την ολοκλήρωση της j εργασίας από την i μηχανή. Η διαδικασία υπολογισμού του makespan βασίζεται στους μαθηματικούς τύπους 2.1 - 2.5 και έχει ως εξής: ξεκινώντας από την πρώτη μηχανή της πρώτης εργασίας όπου ο χρόνος που απαιτείται είναι ο δεδομένος, για κάθε στοιχείο του πίνακα $C(i,j)$ η τιμή που λαμβάνει βασίζεται στον τύπο 2.4 και είναι ο χρόνος που χρειάζεται η μηχανή i για να τελειώσει την εργασία j συν το μέγιστο από τα στοιχεία $C(i,j-1)$ και $C(i-1,j)$. Δηλαδή:

$$C_{i,j} = max\{c_{i,j-1}, c_{i-1,j}\} + t_{i,j} \tag{4.2}$$

Ο πίνακας 4.3 αναπαριστά έναν πίνακα κόστους για 3 μηχανές και 3 εργασίες , κάθε στοιχείο $C(i,j)$ υπολογίζεται από την πρόσθεση του $T(i,j)$ (πίνακας 4.1) συν το μέγιστο από τα στοιχεία του C των οποίων τα βέλη καταλήγουν στο στοιχείο $C(i,j)$. π.χ. το στοιχείο $c(2,2)$ υπολογίζεται ως εξής: $C(2,2) = T(2,2) + max\{c_{2,1}, c_{1,2}\}$

$$C = \begin{bmatrix} t_{1,1} & \rightarrow & c_{1,2} & \rightarrow & c_{1,3} \\ \downarrow & & \downarrow & & \downarrow \\ c_{2,1} & \rightarrow & c_{2,2} & \rightarrow & c_{2,3} \\ \downarrow & & \downarrow & & \downarrow \\ c_{3,1} & \rightarrow & c_{3,2} & \rightarrow & c_{3,3} \end{bmatrix} \tag{4.3}$$

Στον αλγόριθμο 1 παρουσιάζεται ο ψευδοκώδικας της παραπάνω διαδικασίας υπολογισμού του makespan.

Algorithm 1 Υπολογισμός χρόνου ολοκλήρωσης

```
Input T: data table
J = number of jobs
M = number of machines
Initiate C: cost table
for i = 1, 2, ..., M do
    for j = 1, 2, ..., J do
        if i = 1 then
            if j = 1 then
                C(i, j) ← T(i, j)
            else
                C(i, j) ← C(i − 1, j) + T(i, j)
            end if
        else if i = 1 then
            C[i, j] ← C(i, j − 1) + T(i, j)
        else
            C[i, j] ← max{C(i, j − 1), C(i − 1, j)} + T(i, j)
        end if
    end for
end for
Print resutls
```

4.2 Δομή κώδικα

Για κάθε συνδυασμό $J \times M$ εφαρμόζεται ο αλγόριθμος PSO για κάθε ένα από τα 10 παραδείγματα των δεδομένων. Δεδομένου ότι ο αλγόριθμος είναι στοχαστικός η εφαρμογή γίνεται τρεις φορές με τρία διαφορετικά RNG seed και επιλέγεται η καλύτερη λύση από τις τρεις. Σε κάθε επανάληψη εφαρμόζονται στην καθολική, έως εκείνη τη στιγμή, βέλτιστη λύση 10 αλλαγές 1-0 Επανατοποθέτηση και 10 αλλαγές 1-1 Ανταλλαγή, εάν η λύση είναι καλύτερη γίνεται αποδεκτή. Στον αλγόριθμο 2 παρουσιάζεται ο ψευδοκώδικας της βασικής δομής του προγράμματος.

Algorithm 2 Main loop

```
Initiate maximum iterations, number of particles, number of 1-0 and 1-1 local searches.
for i = 1, 2, ..., Number of Taillard Tables do
    Load data from .txt file for Table i
    Get number of jobs and machines from Table i
    for j = 1, 2, ..., Number of examples in each set-up do
        Get lower Bound from data
        Initiate array
        for k = 1, 2, ..., number of different RNG seeds do
            Run PSO and get the best solution
        end for
        Save best solution and lower bound
    end for
end for
Print resutls
```

Παρακάτω ακολουθούν αναλυτικά οι αλγόριθμοι PSO, 1-0 Επανατοποθέτηση και 1-1 Ανταλλαγή.

4.2.1 Υλοποίηση Αλγόριθμου Βελτιστοποίησης Σμήνους Σωματιδίων

Στον αλγόριθμο PSO, αρχικά δημιουργούνται τα σωματίδια. Τα σωματίδια περιέχουν τις εργασίες με τη σειρά που θα επεξεργαστούν δηλαδή κάθε σωματίδιο είναι της μορφής $x = [j_1, j_2, ..., j_J]$ όπου η εργασία j_1 θα επεξεργαστεί πρώτη, η j_2 δεύτερη κ.ο.κ. Η αρχικοποίηση των σωματιδίων γίνεται δίνοντας κάθε σωματίδιο μία τυχαία ακολουθία ακέραιων αριθμών από το 1 έως τον αριθμό των εργασιών του προβλήματος. Έπειτα, αρχικοποιείται και το διάνυσμα ταχύτητας κάθε σωματιδίου το οποίο είναι ένα διάνυσμα της μορφής $v = [s_1, s_2, ..., s_J]$ και αρχικοποιείται με τυχαίες τιμές ομοιόμορφα κατανεμημένες στο εύρος $[0.1 - 0.3]$

Στη συνέχεια, ξεκινάει μια επαναληπτική διαδικασία όπου σε κάθε επανάληψη υπολογίζεται το makespan κάθε σωματιδίου και αποθηκεύεται η καλύτερη λύση κάθε σωματιδίου και το καλύτερο σωματίδιο από όλο το σμήνος. Σύμφωνα με τον τύπο 3.6

ενημερώνεται η ακολουθία των εργασιών κάθε σωματιδίου ως εξής: Υπολογίζονται οι 3 συνιστώσες που συμβάλουν στην αλλαγή της ακολουθίας (α) η ταχύτητα του ιδίου σωματιδίου, (β) η διεύθυνση που προκύπτει από την αφαίρεση των διανυσμάτων της καλύτερης θέσης του σωματιδίου με την θέση του στην συγκεκριμένη επανάληψη και (γ) η διεύθυνση που προκύπτει από την αφαίρεση των διανυσμάτων της θέσης του καλύτερου σωματιδίου με την θέση του στην συγκεκριμένη επανάληψη. Αυτή η ταχύτητα προστίθεται στο διάνυσμα ακολουθίας εργασιών και υπολογίζεται η ακολουθία που προκύπτει μετά τις αλλαγές αυτές. Η διαδικασία ολοκληρώνεται όταν παρέλθει ο μέγιστος αριθμός επαναλήψεων.

Algorithm 3 PSO

Initialize maximum iterations, number of particles, number of jobs, number of machines.
Set speed coefficients, c_1, c_2 and w
Initialize particles with a random sequence of jobs
for i = 1, 2, ..., maximum iterations **do**
    Compute makespan for each particle and save the best and the global best.
    for 1,2,..., number of particles **do**
        Calculate Speed Vector and update particle's speed
        Change order of jobs based on the changes.
    end for
    Run 1-0 Relocate for n iterations and keep solution if it is better
    Run 1-1 Exchange for n iterations and keep solution if it is better
    Save best solution and lower bound
end for
Print resutls

4.2.2 Υλοποίηση αλγορίθμων τοπικής αναζήτησης

Στους αλγόριθμους 4 και 5 παρουσιάζεται η βασική λειτουργία των αντίστοιχων συναρτήσεων που υλοποιήθηκαν. Ο έλεγχος κόστους της καινούργιας λύσης γίνεται στον αλγόριθμο PSO, έτσι η δομή τους είναι πολύ απλή.

Για τον αλγόριθμο 1-1 Ανταλλαγή αρχικά γίνεται η εισαγωγή της ακολουθίας εργασιών $J = [j_1, j_2, ..., j_n]$, έπειτα επιλέγονται δύο τυχαίες εργασίες και η θέση τους ανταλλάσσεται. Στη συνέχεια, ελέγχεται η ποιότητα της ακολουθίας που προκύπτει.

Algorithm 4 1-1 Exchange

Input the sequence
Get 2 random integers a,b from [0, *length*(sequence)-1]
Swap the jobs dictated by a and b
return sequence

Για τον αλγόριθμο 1-0 Επανατοποθέτηση, η διαδικασία είναι παρόμοια με αυτή του 1-1 Ανταλλαγή με την διαφορά πως σε αυτόν τον αλγόριθμο επιλέγονται δύο τυχαίοι αριθμοί από το 1 εως τον αριθμό των εργασιών. Ο πρώτος τυχαίος αριθμός α ορίζει την εργασία που πρόκειται να επανατοποθετηθεί και ο 2ος τυχαίος αριθμός β ορίζει τη θέση στην οποία θα γίνει αυτή η επανατοποθέτηση.

Algorithm 5 1-0 Relocate

Input sequence
Get 2 random integers a, b from [0, *length*(sequence)-1]
Relocate the job indexed by a to position indexed by b.
return sequence

Κεφάλαιο 5

Αποτελέσματα

Οι αλγόριθμοι εφαρμόστηκαν τρεις φορές με τρία διαφορετικά RNG Seed για κάθε ένα από τους 12 $J \times M$ συνδυασμούς στους πίνακες Taillard και για κάθε ένα από τα 10 παραδείγματα του κάθε συνδυασμού. Αρχικά έγινε η εφαρμογή του PSO χωρίς τη χρήση αλγορίθμων τοπικής αναζήτησης και έπειτα με τη χρήση. Στη παράγραφο 5.2 παρατίθενται τα αποτελέσματα.

5.1 Τιμές παραμέτρων

Υπολογισμός του διανύσματος ταχύτητας γίνεται σύμφωνα με την εξίσωση 3.6 όπου w είναι η σταθερά αδράνειας, c_1, c_2 οι συντελεστές επιτάχυνσης. Οι τιμές των παραμέτρων που επιλέχθηκαν υπολογίστηκαν μέσω δοκιμής και σφάλματος και παρουσιάζονται στον πίνακα 5.1.

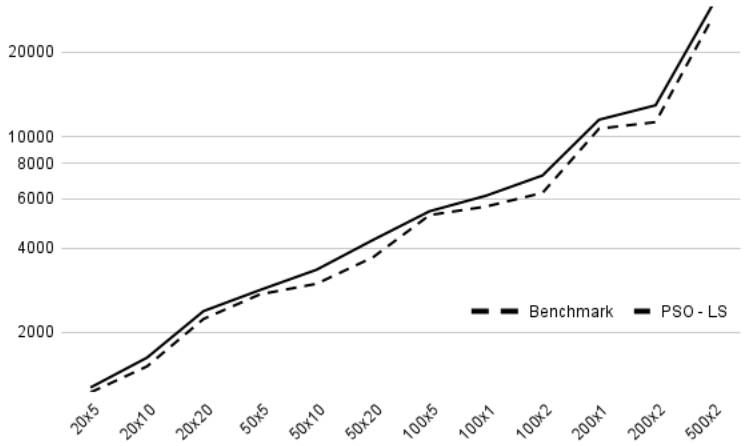
Παράμετρος	Τιμή
Επαναλήψεις	800
Αριθμός Σωματιδίων	500
Επαναλήψεις 1-1	10
Επαναλήψεις 1-0	10
c_1	2.05
c_2	2.05
w	0.5

Πίνακας 5.1: Τιμές παραμέτρων Αλγόριθμου PSO

Το πλήθος των σωματιδίων και των επαναλήψεων είναι αρκετά υψηλό έτσι ώστε να βρεθεί όσο το δυνατόν καλύτερη λύση. Ωστόσο, αυτό έχει ως αποτέλεσμα την σχετικά χρονοβόρο εκτέλεση του αλγορίθμου.

5.2 Αποτελέσματα

Στο γράφημα 5.1 φαίνεται το μέσο κόστος των λύσεων ανά συνδυασμό $J \times M$ για τον αλγόριθμο PSO με τη χρήση αλγορίθμων τοπικής αναζήτησης και για τα ολικά ελάχιστα (Benchmark). Σημειώνεται πως το γράφημα είναι σε λογαριθμική κλίμακα. Έτσι είναι καλύτερα εμφανή η αύξηση του κόστους συναρτήσει του αριθμού των εργασιών και του αριθμού των μηχανών. Είναι εύκολα διακριτό πως η αύξηση των εργασιών επηρεάζει το κόστος σε μεγαλύτερο βαθμό από την αύξηση του αριθμού των μηχανών. Ο αριθμός των εργασιών επηρεάζει άμεσα το συνολικό κόστος καθώς υπάρχει άμεση συσχέτιση μεταξύ τους. Ο αριθμός των μηχανών επηρεάζει το συνολικό κόστος με έμμεσο τρόπο. Όσο μεγαλύτερος είναι ο αριθμός των μηχανών τόσο μεγαλώνει και ο δειγματικός χώρος κάνοντας έτσι δυσκολότερη την εύρεση μιας καλής λύσης.



Σχήμα 5.1: Μέσο κόστος λύσεων κάθε συνδυασμού

Στους πίνακες 5.2 έως 5.4 παρουσιάζονται τα αποτελέσματα για τους 12 συνδυασμούς $J \times M$ και για κάθε ένα από τα 10 παραδείγματα. Στην στήλη BM (Benchmark) παρουσιάζονται τα ολικά ελάχιστα των προβλημάτων όπως προκύπτουν από τους πίνακες Taillard, στη στήλη PSO-LS παρουσιάζονται τα κόστη των λύσεων που υπολογίστηκαν από τον αλγόριθμο PSO κάνοντας χρήση των αλγορίθμων τοπικής αναζήτησης, και στην τρίτη στήλη είναι το επί τις εκατό σφάλμα μεταξύ των δύο τιμών. Στην τέταρτη στήλη είναι τα αποτελέσματα του αλγορίθμου PSO χωρίς τη χρήση των αλγορίθμων τοπικής αναζήτησης και στην πέμπτη το επι τις εκατό σφάλμα με τα ολικά ελάχιστα.

Problem	BM	PSO-LS	error (%)	PSO	error (%)
20×5	1278	1297	1.487	1336	4.538
20×5	1359	1367	0.589	1368	0.662
20×5	1081	1153	6.660	1163	7.586
20×5	1293	1370	5.955	1375	6.342
20×5	1235	1282	3.806	1287	4.211
20×5	1195	1239	3.682	1246	4.268
20×5	1239	1269	2.421	1276	2.986
20×5	1206	1274	5.638	1283	6.385
20×5	1230	1292	5.041	1294	5.203
20×5	1108	1167	5.325	1198	8.123
20×10	1582	1707	7.901	1727	9.166
20×10	1659	1754	5.726	1791	7.957
20×10	1496	1631	9.024	1631	9.024
20×10	1377	1508	9.513	1523	10.603
20×10	1419	1548	9.091	1575	10.994
20×10	1397	1492	6.800	1510	8.089
20×10	1484	1601	7.884	1614	8.760
20×10	1538	1664	8.192	1671	8.648
20×10	1593	1677	5.273	1682	5.587
20×10	1591	1677	5.405	1730	8.737
20×20	2297	2438	6.138	2456	6.922
20×20	2099	2240	6.717	2261	7.718
20×20	2326	2449	5.288	2463	5.890
20×20	2223	2386	7.332	2392	7.602
20×20	2291	2454	7.115	2462	7.464
20×20	2226	2374	6.649	2387	7.233
20×20	2273	2397	5.455	2399	5.543
20×20	2200	2319	5.409	2378	8.091
20×20	2237	2394	7.018	2446	9.343
20×20	2178	2338	7.346	2365	8.586
50×5	2724	2753	1.065	2773	1.799
50×5	2834	2915	2.858	2944	3.881
50×5	2621	2708	3.319	2794	6.601
50×5	2751	2868	4.253	2890	5.053
50×5	2863	2939	2.655	2945	2.864
50×5	2829	2915	3.040	2930	3.570
50×5	2725	2852	4.661	2853	4.697
50×5	2683	2824	5.255	2832	5.553
50×5	2552	2643	3.566	2680	5.016
50×5	2782	2869	3.127	2875	3.343
50×10	2991	3353	12.103	3385	13.173
50×10	2867	3246	13.219	3263	13.812
50×10	2839	3288	15.815	3289	15.851
50×10	3063	3390	10.676	3483	13.712
50×10	2976	3384	13.710	3409	14.550
50×10	3006	3384	12.575	3386	12.641
50×10	3093	3395	9.764	3473	12.286
50×10	3037	3343	10.076	3370	10.965
50×10	2897	3276	13.082	3293	13.669
50×10	3065	3426	11.778	3426	11.778

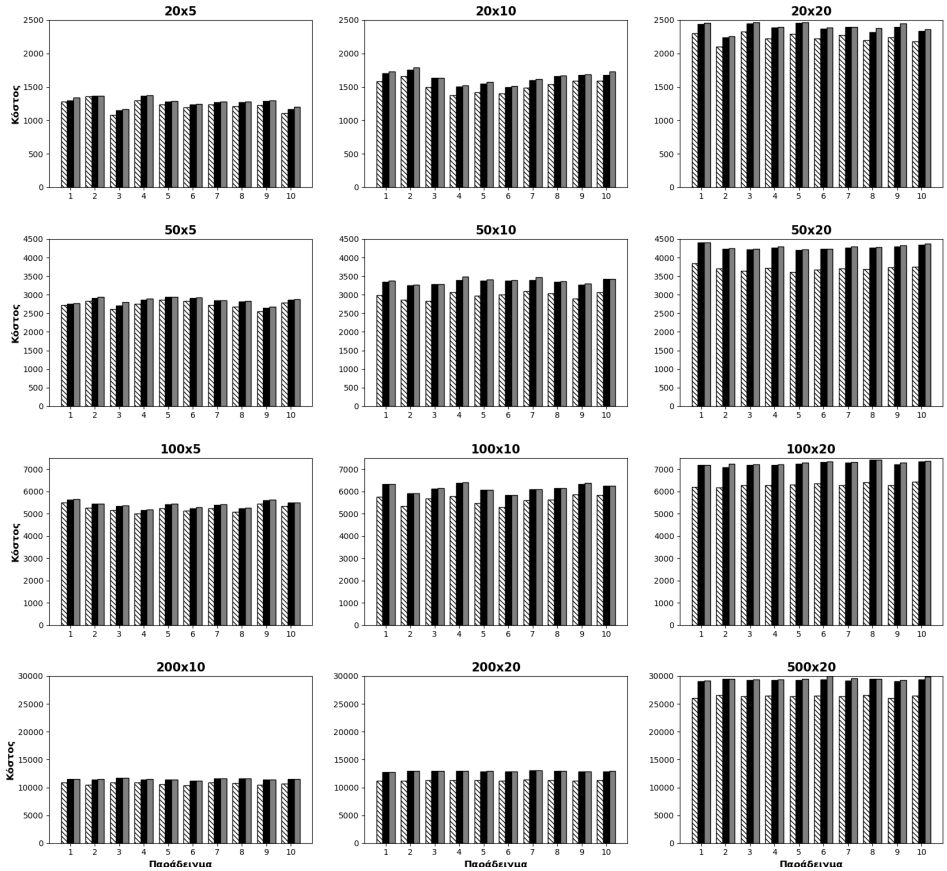
Πίνακας 5.2: Αναλυτικά Αποτελέσματα (1/3)

Problem	BM	PSO-LS	error (%)	PSO	error (%)
50 × 20	3850	4408	14.494	4409	14.519
50 × 20	3704	4230	14.201	4246	14.633
50 × 20	3640	4212	15.714	4240	16.484
50 × 20	3720	4266	14.677	4293	15.403
50 × 20	3610	4211	16.648	4214	16.731
50 × 20	3681	4240	15.186	4240	15.186
50 × 20	3704	4267	15.200	4297	16.010
50 × 20	3691	4268	15.633	4277	15.876
50 × 20	3743	4297	14.801	4329	15.656
50 × 20	3756	4343	15.628	4381	16.640
100 × 5	5493	5634	2.567	5655	2.949
100 × 5	5268	5445	3.360	5448	3.417
100 × 5	5175	5332	3.034	5359	3.556
100 × 5	5014	5171	3.131	5181	3.331
100 × 5	5250	5427	3.371	5446	3.733
100 × 5	5135	5249	2.220	5305	3.311
100 × 5	5246	5395	2.840	5411	3.145
100 × 5	5094	5248	3.023	5262	3.298
100 × 5	5448	5610	2.974	5633	3.396
100 × 5	5332	5492	3.001	5511	3.357
100 × 10	5770	6325	9.619	6342	9.913
100 × 10	5349	5912	10.525	5917	10.619
100 × 10	5676	6128	7.963	6147	8.298
100 × 10	5781	6384	10.431	6402	10.742
100 × 10	5467	6072	11.066	6076	11.140
100 × 10	5303	5839	10.107	5843	10.183
100 × 10	5595	6090	8.847	6090	8.847
100 × 10	5617	6139	9.293	6162	9.703
100 × 10	5871	6335	7.903	6393	8.891
100 × 10	5845	6249	6.912	6250	6.929
100 × 20	6202	7180	15.769	7188	15.898
100 × 20	6183	7083	14.556	7245	17.176
100 × 20	6271	7177	14.447	7225	15.213
100 × 20	6269	7195	14.771	7213	15.058
100 × 20	6314	7236	14.602	7304	15.679
100 × 20	6364	7314	14.928	7340	15.336
100 × 20	6268	7297	16.417	7311	16.640
100 × 20	6401	7410	15.763	7423	15.966
100 × 20	6275	7216	14.996	7287	16.127
100 × 20	6434	7341	14.097	7380	14.703

Πίνακας 5.3: Αναλυτικά Αποτελέσματα (2/3)

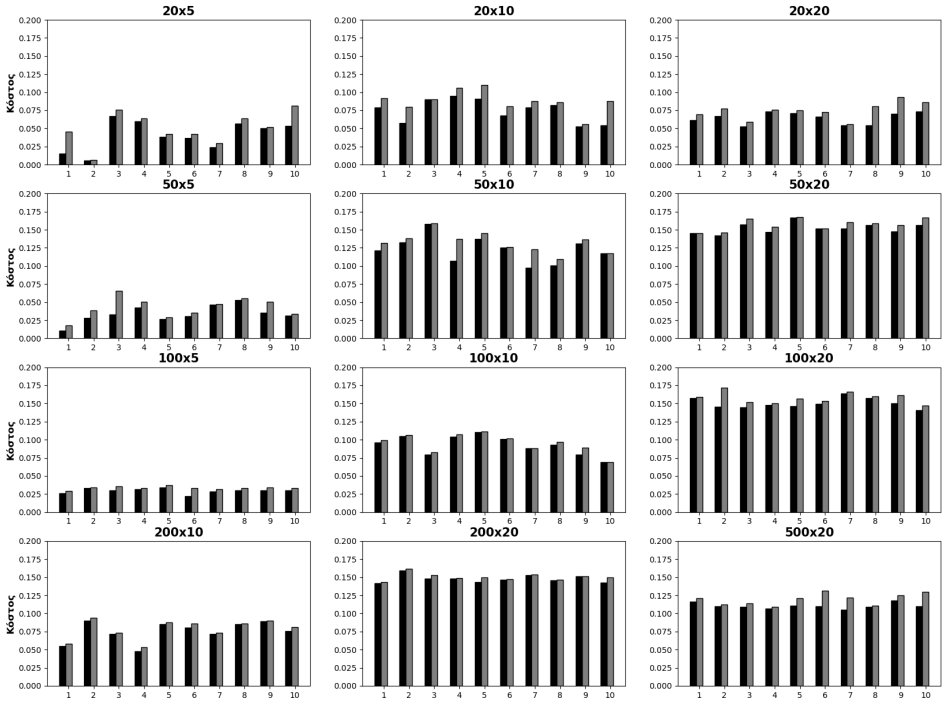
Problem	BM	PSO-LS	error (%)	PSO	error (%)
200×10	10862	11463	5.533	11492	5.800
200×10	10480	11422	8.989	11468	9.427
200×10	10922	11708	7.196	11724	7.343
200×10	10889	11409	4.775	11473	5.363
200×10	10524	11417	8.485	11449	8.789
200×10	10329	11157	8.016	11214	8.568
200×10	10854	11634	7.186	11648	7.315
200×10	10730	11642	8.500	11651	8.583
200×10	10438	11373	8.958	11379	9.015
200×10	10675	11480	7.541	11543	8.131
200×20	11195	12782	14.176	12804	14.372
200×20	11203	12984	15.898	13012	16.147
200×20	11281	12955	14.839	13003	15.265
200×20	11275	12948	14.838	12953	14.882
200×20	11259	12877	14.371	12943	14.957
200×20	11176	12814	14.656	12826	14.764
200×20	11360	13095	15.273	13104	15.352
200×20	11334	12982	14.540	12994	14.646
200×20	11192	12888	15.154	12890	15.172
200×20	11288	12896	14.245	12975	14.945
500×20	26059	29096	11.654	29208	12.084
500×20	26520	29433	10.984	29501	11.241
500×20	26371	29243	10.891	29368	11.365
500×20	26456	29284	10.689	29342	10.909
500×20	26334	29243	11.047	29528	12.129
500×20	26477	29384	10.979	29964	13.170
500×20	26389	29158	10.493	29615	12.225
500×20	26560	29460	10.919	29498	11.062
500×20	26005	29077	11.813	29264	12.532
500×20	26457	29362	10.980	29893	12.987

Πίνακας 5.4: Αναλυτικά Αποτελέσματα (3/3)



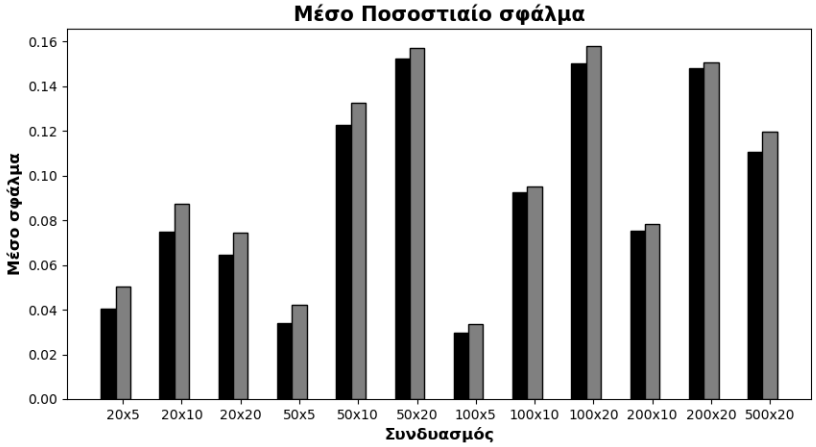
Σχήμα 5.2: Κόστη λύσεων κάθε συνδυασμού $J \times M$

Στα γραφήματα του σχήματος 5.2 παρουσιάζονται τα κόστη για κάθε συνδυασμό και για κάθε παράδειγμά τους. Οι λευκές μπάρες με τις πλάγιες γραμμές είναι τα ολικά ελάχιστα BM των πινάκων 5.2 - 5.4, με μαύρο είναι η εφαρμογή του PSO με την χρήση τοπικής αναζήτησης και με γκρί χωρίς. Ο αλγόριθμος φαίνεται να παρουσιάζει ικανοποιητικά αποτελέσματα στα προβλήματα με χαμηλό αριθμό μηχανών. Η χρήση αλγορίθμων τοπικής αναζήτησης μειώνει το κόστος σε κάθε παράδειγμα, με το σημαντικό μειονέκτημα του επιπλέον υπολογιστικού χρόνου. Είναι εμφανές ότι το σφάλμα αυξάνεται αισθητά για τα προβλήματα με αριθμό μηχανών ίσο με 20. Για την πρώτη εφαρμογή, με την χρήση δηλαδή των αλγορίθμων τοπικής αναζήτησης το μέσο σφάλμα για το πρόβλημα 20×20 είναι 6.45% , 15.22% για το πρόβλημα 50×20, 15.03% για το 100×20, 14.80% για το 200×20 και 11.04% για το 500×20. Στα προβλήματα με αριθμό μηχανών ίσο με 5 το μέσο σφάλμα είναι μικρότερο του 5% ενώ για αυτά με αριθμό μηχανών ίσο με 10 το μέσο σφάλμα κυμαίνεται στο εύρος 7.5% - 10% με το πρόβλημα 50×10 να παρουσιάζει μέσο σφάλμα της τάξης του 12%. Τα αποτελέσματα της δεύτερης εφαρμογής, χωρίς τη χρήση δηλαδή των αλγορίθμων τοπικής αναζήτησης παρουσιάζουν την ίδια κατανομή σφάλματος ως προς το πλήθος των μηχανών και των εργασιών με μια αύξηση στο εύρος 0.26 - 1.28% ως προς την πρώτη εφαρμογή. Τα μέσα ποσοστιαία σφάλματα για τους 12 συνδυασμούς $J \times M$ παρουσιάζονται στο γράφημα 5.4.



Σχήμα 5.3: Ποσοστιαία σφάλματα των λύσεων κάθε συνδυασμού $J \times M$

Στο γράφημα 5.3 φαίνεται το ποσοστιαίο σφάλμα από τα ολικά ελάχιστα για κάθε συνδυασμό $J \times M$ και για κάθε παράδειγμά τους. Είναι εμφανές πως το σφάλμα αυξάνεται καθώς αυξάνεται και ο δειγματικός χώρος του προβλήματος και πιο συγκεκριμένα το πλήθος των διαθέσιμων μηχανών. Επίσης, η χρήση αλγορίθμων τοπικής αναζήτησης, αν και προσφέρει μια κάποια βελτίωση τον λύσεων, δεν επαρκεί για να αντισταθμίσει τον επιπλέον υπολογιστικό χρόνο.



Σχήμα 5.4: Μέσο ποσοστιαίο σφάλμα λύσεων κάθε συνδυασμού $J \times M$

5.3 Συμπεράσματα

Συμπεραίνεται λοιπόν, ότι ο αλγόριθμος PSO ανταποκρίνεται επαρκώς στο πρόβλημα χρονοπρογραμματισμού αντιμετάθεσης εργασιών συνεχούς ροής , παράγοντας ικανοποιητικές λύσεις. Η χρήση των αλγορίθμων τοπικής αναζήτησης 1-0 Επανατοποθέτηση και 1-1 Ανταλλαγή προσφέρει ελάχιστη βελτίωση των λύσεων και έτσι δεν κρίνεται ζωτικής σημασίας για την παραγωγή αποτελεσμάτων. Δεδομένης και της επαύξησης του υπολογιστικού χρόνου που απαιτείται μπορεί να παραληφθεί η χρήση αυτή.

Βιβλιογραφία

[1] Yannis Marinakis, Magdalene Marinaki, *Particle swarm optimization with expanding neighborhood topology for the permutation flowshop scheduling problem.* Verlag Berlin Heidelberg: Springer, 17, 1159–1173, 2013.

[2] Taillard, E., 'Scheduling instances,' 2008.

[3] Samia kouki , Mohamed Jemni, Talel Ladhari, *Solving the Permutation Flow Shop Problem with Makespan Criterion using Grids, International Journal of Grid and Distributed Computing.* 4 (2), 53-64, 2011.

[4] Vos, S., *Meta-heuristics: The State of the Art. In: Nareyek A. (Eds.) Local Search for Planning and Scheduling. LSPS 2000. Lecture Notes in Computer Science.* 2148, Springer, Berlin, Heidelberg., 2001.

[5] El-Ghazali Talbi, *Metaheuristics: from design to implementation. New. Wiley Series on Parallel and Distributed Computing.* Wiley, 2009.

[6] Gendreau, M., Potvin, J.-Y., *Metaheuristics in Combinatorial Optimization. Annals of Operations Research.* 140, 190-197 ???, 2005.

[7] Sastry, Kumara and Goldberg, David and Kendall, Graham, *Genetic Algorithms.* Boston, MA: Springer US, 2005.

[8] Kirkpatrick, S. and Gelatt, C. D. and Vecchi, M. P., 'Optimization by Simulated Annealing,' *Science*, ολ. 220, νο. 4598, 1983.

[9] Dorigo, Marco and Birattari, Mauro and Stutzle, Thomas, 'Ant colony optimization,' *IEEE Computational Intelligence Magazine*, νο. 4, 2006.

[10] Xin-She Yang, *Nature-Inspired Optimization Algorithms. 1st ed. Elsevier Insights.* Elsevier, 2014.

[11] Christos H. Papadimitriou and Kenneth Steiglitz, *Combinatorial Optimization: Algorithms and Complexity. 2nd ed. Vol. 1. Mathematical optimization.* Prentice Hall, 1998.