ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΑΡΑΓΩΓΗΣ
ΚΑΙ ΔΙΟΙΚΗΣΗΣ

ΤΟΜΕΑΣ: ΕΠΙΧΕΙΡΗΣΙΑΚΗ ΕΡΕΥΝΑ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Μεθοδολογία επίλυσης προβλημάτων χρονοπρογραμματισμού εργασιών με στοχαστικό χρόνο άφιξης ή εξυπηρέτησης»

ΠΙΚΟΥΛΗ ΘΕΟΔΩΡΑ

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ: ΜΑΡΙΝΑΚΗΣ ΙΩΑΝΝΗΣ

ΧΑΝΙΑ 2016

ΠΕΡΙΛΗΨΗ

Στην παρούσα διπλωματική εργασία παρουσιάζεται το πρόβλημα αντιμετάθεσης χρονοπρογραμματισμού εργασιών, ένα πρόβλημα με το οποίο έρχονται αντιμέτωπες όλες οι σύγχρονες βιομηχανίες, προκειμένου να αυξήσουν την παραγωγή τους, ελαχιστοποιώντας ταυτόχρονα το κόστος λειτουργίας τους. Πρόκειται για ένα πρόβλημα που αφορά την κατανομή των διαφόρων εργασιών μιας γραμμής παραγωγής στις διαθέσιμες μηχανές με αντικειμενικό στόχο την ελαχιστοποίηση του συνολικού χρόνου επεξεργασίας των εργασιών από τις μηχανές. Ως βασικοί κανόνες αυτού του προβλήματος τίθενται οι εξής: όλες οι εργασίες πρέπει να επεξεργαστούν από όλες τις μηχανές με την ίδια σειρά, οι μηχανές δε μπορούν να διακόψουν τη λειτουργία τους, κάθε χρονική στιγμή μία μηχανή μπορεί να επεξεργαστεί το πολύ μία εργασία και τέλος μία εργασία μπορεί να επεξεργαστεί μόνο από μία μηχανή. Για την επίλυση του συγκεκριμένου προβλήματος επιλέχθηκε ο μεθευρετικός αλγόριθμος της Τεχνητής Αποικίας Μελισσών (Artificial Bee Colony - ABC), ένας αλγόριθμος ο οποίος βασίζεται στην ευφυΐα του σμήνους και όχι στα χαρακτηριστικά του εκάστοτε προβλήματος προκειμένου να βρει την βέλτιστη λύση. Ο αλγόριθμος αυτός, ιδιαίτερα διαδεδομένος στα συνεχή προβλήματα, τροποποιήθηκε κατάλληλα εδώ προκειμένου να επιλύσει το προαναφερθέν διακριτό πρόβλημα χρονοπρογραμματισμού εργασιών.

Στην παρούσα εργασία, το πρόβλημα αντιμετάθεσης χρονοπρογραμματισμού εργασιών, αρχικά, εμφανίζεται με τη μορφή θεωρητικών αιτιοκρατικών προβλημάτων από την βιβλιογραφία. Σε αυτά εφαρμόζεται ο διακριτός αλγόριθμος ABC προκειμένου να διαπιστωθεί η αξιοπιστία και ευρωστία του. Εν συνεχεία, μοντελοποιείται ένα πραγματικό σύστημα παραγωγής ενός εργαστηρίου διακρίβωσης και επιλύεται από τον ABC, αρχικά θεωρούμενο ως αιτιοκρατικό πρόβλημα και στη συνέχεια ως στοχαστικό. Από την επίλυση αυτού του συστήματος προκύπτουν αποτελέσματα, μέσα από τα οποία καταλήγουμε σε συμπεράσματα για τη βέλτιστη λειτουργία του συγκεκριμένου συστήματος παραγωγής.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΚΕΦΑΛΑΙΟ 1
ΧΡΟΝΟΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΕΡΓΑΣΙΩΝ

1.1 Πρόβλημα Χρονοπρογραμματισμού Εργασιών

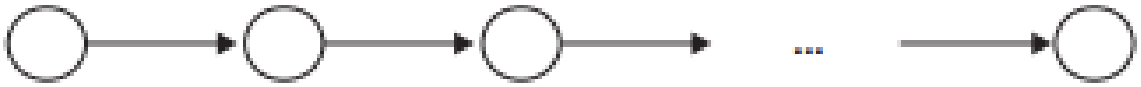
Στον σύγχρονο κόσμο, οι βιομηχανίες αντιμπετωπίζουν ένα συνονθύλευμα προβλημάτων και περιορισμών σε πολλούς τομείς της παραγωγικής διαδικασίας τους λόγω της πολυπλοκότητας του συστήματος παραγωγής αλλά και της δυναμικής φύσης του. Τέτοια προβλήματα και περιορισμοί μπορεί να αφορούν στον χρόνο ζωής και λειτουργίας των μηχανών, στο κόστος ηλεκτρικού ρεύματος, στο ανθρώπινο δυναμικό αλλά και σε περιορισμούς που θέτουν οι πρώτες ύλες ή ακόμα και οι ίδιοι οι πελάτες. Επομένως, είναι αναγκαίος ο προγραμματισμός της γραμμής παραγωγής, ο οποίος σε πρώτο στάδιο θα συντελέσει στην αποτελεσματική αξιοποίηση μηχανών και προσωπικού καθώς και στην ελαχιστοποίηση του χρόνου αναμονής των πελατών, αποθήκευσης και χρόνου εκτέλεσης. Σε επόμενο στάδιο θα επηρεάσει άμεσα τη ροή των οικονομικών εισροών στο σύστημα, που αποτελεί και τον απώτερο στόχο του χρονικού προγραμματισμού παραγωγής. Επομένως, ως χρονοπρογραμματισμός εργασιών (job shop *scheduling*) καθίσταται η κατανομή των μηχανών στις διάφορες εργασίες *(jobs)* σε ένα εύλογο χρονικό διάστημα.[1] Η σειρά με την οποία εκτελούνται οι εργασίες από τις μηχανές μπορεί να διαφέρει από παραγγελία σε παραγγελία, έχοντας όμως πάντα ως αντικειμενικό σκοπό την εύρεση μιας βέλτιστης κατανομής των διαφορετικών εργασιών στις διαθέσιμες μηχανές.

Μία ειδική κατηγορία του προβλήματος χρονοπρογραμματισμού εργασιών αποτελεί το πρόβλημα χρονοπρογραμματισμού συνεχούς ροής *(flow-shop scheduling problem).* Σε αυτήν την περίπτωση, η σειρά των επεξεργασιών είναι πάντα ίδια από παραγγελία σε παραγγελία για κάθε μονάδα προϊόντος. Επομένως, ένα τέτοιο πρόβλημα καθορίζει μία βέλτιστη σειρά n εργασιών που επεξεργάζονται από m μηχανές με την ίδια σειρά.

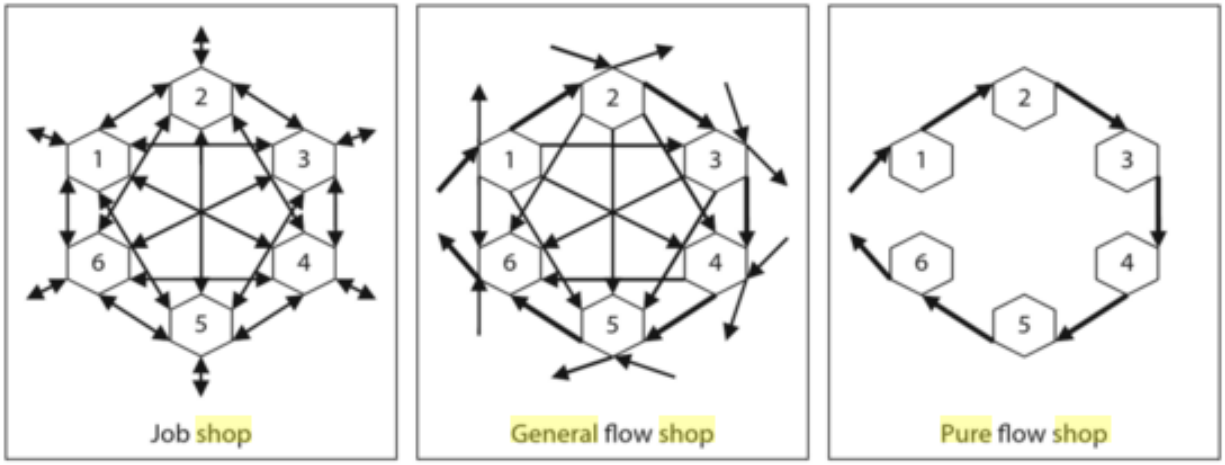
Γενικά υπάρχουν δύο κατηγορίες συστημάτων χρονοπρογραμματισμού συνεχούς ροής. Η πρώτη ονομάζεται «καθαρό» σύστημα flow-shop, όπου όλες οι εργασίες για να εκτελεστούν περνούν από όλες τις μηχανές, και η δεύτερη ονομάζεται «γενικό» flow-shop, όπου κάθε εργασία, αν και ακολουθεί την ίδια κατεύθυνση μέσα στο σύστημα, δεν περνάει υποχρεωτικά από όλες τις μηχανές [2].

Παράδειγμα καθαρού συστήματος flow-shop είναι μια γραμμή συναρμολόγησης, όπου η παραγωγή πραγματοποιείται από φάση σε φάση στην ίδια πάντα κατεύθυνση και περνώντας από όλους τους σταθμούς παραγωγής. Σε άλλες περιπτώσεις, όπως στην περίπτωση παραγωγής ολοκληρωμένων κυκλωμάτων ή στην περίπτωση παραγωγής ενδυμάτων, η ίδια διαδοχή επεξεργασιών απαιτείται για ένα μεγάλο αριθμό κομματιών μιας

παραγγελίας, ενώ η διαδοχή αυτή μπορεί να αλλάζει από παραγγελία σε παραγγελία. Ένα τέτοιο σύστημα θεωρείται γενικό σύστημα flow-shop. Σε άλλες περιπτώσεις, πάλι, η παραγωγή μπορεί να θεωρηθεί ότι διεξάγεται σε δύο φάσεις, που αντιστοιχούν στους δυο τύπους συστημάτων, job-shop και flow-shop. Παράδειγμα τέτοιου συστήματος αποτελεί μια βιομηχανία επίπλων, όπου η βασική διαμόρφωση του επίπλου πραγματοποιείται στις αρχικές φάσεις της παραγωγής, όμοια για όλα τα προϊόντα (flow-shop κομμάτι) ενώ στις τελευταίες φάσεις (τοποθέτηση ταπετσαρίας, βαφή, κ.λπ.) η παραγωγή διαφοροποιείται και εξατομικεύεται (job-shop κομμάτι).



Εικόνα 1.1.1: *Η αλληλουχία των εργασιών σε ένα σύστημα συνεχούς ροής.*



Εικόνα 1.1.2: *Απεικόνιση των τριών τύπων συστημάτων παραγωγής σε σχέση με τα χαρακτηριστικά δρομολόγησης τους. (Η πιθανότητα μετάβασης από μία διεργασία σε μία άλλη απεικονίζεται από το πάχος που έχουν τα βελάκια.)*

1.2 Πρόβλημα Χρονοπρογραμματισμού Εργασιών Συνεχούς Ροής

Στην παρούσα διπλωματική, θα επικεντρωθούμε στο «καθαρό» σύστημα *flow-shop scheduling problem*, όπου **n** εργασίες υπόκεινται επεξεργασία από **m** διαφορετικές μηχανές. Κάθε μία από τις **n** εργασίες απαιτεί **m** λειτουργίες προκειμένου να ολοκληρωθεί και κάθε

λειτουργία πραγματοποιείται από μία διαφορετική μηχανή. Το διάγραμμα ροής εργασιών είναι αμφίδρομο, αφού κάθε εργασία πρέπει να υποστεί επεξεργασία από κάθε μηχανή, σε μία προκαθορισμένη σειρά. Θεωρούμε ότι μία μηχανή δεν μπορεί να επεξεργαστεί ταυτόχρονα πάνω από μία εργασία και ότι κάθε εργασία εκτελείται μόνο σε μία μηχανή. Υποθέτουμε επίσης ότι κάθε εργασία δεν επισκέπτεται την ίδια μηχανή πάνω από μία φορά και ότι οι χρόνοι εκκίνησης και προετοιμασίας των μηχανών περιλαμβάνονται στους χρόνους επεξεργασίας κάθε εργασίας από μία μηχανή. Τέλος, ότι οι εργασίες που υφίστανται επεξεργασία από κάθε μηχανή δεν μπορούν να διακοπούν.

Επομένως, σε ένα σύστημα χρονοπρογραμματισμού συνεχούς ροής μπορούμε να αριθμήσουμε τις μηχανές από 1,2, ..., **m** και τις λειτουργίες κάθε εργασίας n ως εξής: (1,**n**), (2,**n**), ... ,(**m,n**) , ώστε να τις αντιστοιχήσουμε σε κάθε μηχανή. Επίσης, μπορούμε να συμβολίσουμε τον χρόνο επεξεργασίας κάθε εργασίας **n** στη μηχανή m ως p_{mn}. Για παράδειγμα, το p_{53} αποδίδει το χρόνο που χρειάστηκε η μηχανή 5 να επεξεργαστεί την εργασία 3. [6]

Ο αντικειμενικός σκοπός του προβλήματος είναι η εύρεση μιας σειράς εργασιών - λύσης - τέτοιας που θα ελαχιστοποιεί τον συνολικό χρόνο επεξεργασίας ή makespan. Ως makespan *Cmax* ορίζεται ο συνολικός χρόνος που απαιτείται για να επεξεργαστούν όλες οι εργασίες από όλες τις μηχανές. Με άλλα λόγια, το makespan ισούται με τον χρόνο περάτωσης των εργασιών στην τελευταία μηχανή:

$Cmax = max(R_{iM})$

όπου R_{iM} ο χρόνος επεξεργασίας κάθε εργασίας i στην τελευταία μηχανή M.

Φυσικά, δεν είναι δυνατό να υπολογιστούν όλες οι πιθανές λύσεις ώστε να ευρεθεί αυτή που ελαχιστοποιεί το makespan, δεδομένου ότι ο αριθμός όλων των δυνατών λύσεων του προβλήματος είναι $(\boldsymbol{n}!)^{\boldsymbol{m}}$ διαφορετικοί χρονοπρογραμματισμοί εργασιών, ένα μέγεθος που οδηγεί σε υπολογιστικό αδιέξοδο.

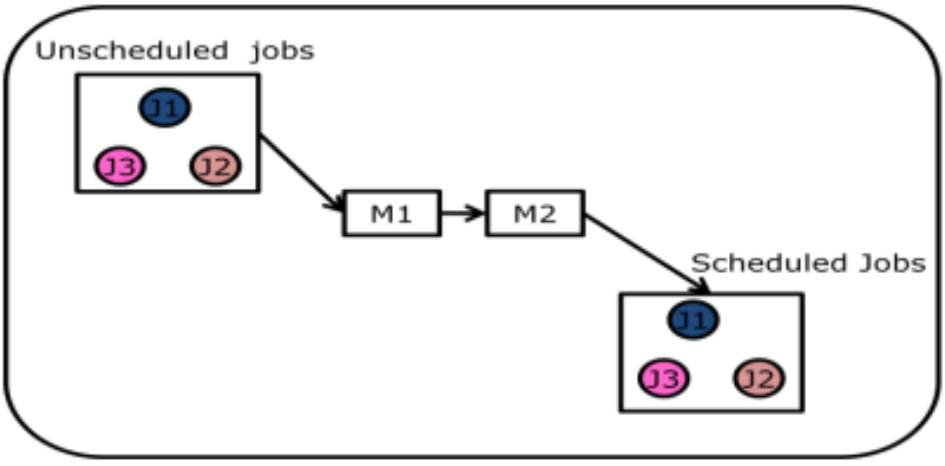
1.3 Πρόβλημα Χρονοπρογραμματισμού Συνεχούς Ροής Αντιμετάθεσης- The Permutation Flow Shop Problem

Ας θεωρήσουμε τώρα ένα σύστημα χρονοπρογραμματισμού συνεχούς ροής, όπως αυτό που αναφέρθηκε παραπάνω, στο οποίο ισχύουν όλοι οι προαναφερθείς περιορισμοί. Αν σε αυτό το σύστημα παραγωγής προσθέσουμε και τον ισχυρισμό ότι όλες οι εργασίες πρέπει να εκτελεστούν με την ίδια ακριβώς σειρά από κάθε μία από τις **m** μηχανές, οδηγούμαστε στο *Permutation Flow Shop Problem (PFSP).*

Αυτό το πρόβλημα ακολουθεί τους κάτωθι ισχυρισμούς:

- Όλες οι εργασίες είναι έτοιμες για επεξεργασία τη χρονική στιγμή 0.
- Οι μηχανές είναι συνεχώς διαθέσιμες από τη στιγμή 0 και έπειτα (δεν υπάρχουν βλάβες).
- Κάθε χρονική στιγμή, μία μηχανή μπορεί να επεξεργαστεί το πολύ μία εργασία και κάθε εργασία μπορεί να υφίσταται επεξεργασία το πολύ από μία μηχανή.
- Δεν επιτρέπονται προτεραιότητες. Αυτό σημαίνει ότι από τη στιγμή που μία μηχανή επεξεργάζεται μία εργασία, αυτή δε μπορεί να διακοπεί για να επεξεργαστεί άλλη εργασία με υψηλότερη προτεραιότητα.
- Μόνο προγραμματισμοί αντιμετάθεσης επιτρέπονται. Δηλαδή όλες οι εργασίες έχουν την ίδια σειρά σε όλες τις μηχανές. [5]

Όπως και στο απλό πρόβλημα χρονοπρογραμματισμού συνεχούς ροής, έτσι και εδώ στόχος του προβλήματος είναι η ικανοποίηση της αντικειμενικής συνάρτησης. Αναφερόμαστε στην ελαχιστοποίηση του χρόνου επεξεργασίας της τελευταίας εργασίας στην τελευταία μηχανή, το οποίο όπως προαναφέραμε ονομάζεται makespan και συμβολίζεται με *Cmax*. Θεωρούμε τώρα ότι έχουμε ένα διακριτό σύνολο ***S*** και μία βέλτιστη λύση $x^* \in S$, τέτοια ώστε $F(x^*) = \min F(x) \ \forall x \in S$, όπου ***F*** καλείται η αντικειμενική συνάρτηση, ***S*** καλείται μία εφικτή περιοχή και ***x*** μία εφικτή λύση. Έτσι, επιλύοντας το *PFSP* και έχοντας το makespan ως την αντικειμενική συνάρτηση προς ελαχιστοποίηση, έχουμε ως αποτέλεσμα τον καθορισμό μιας σειράς των εργασιών,τέτοιας ώστε να δίνει τη μικρότερη τιμή της αντικειμενικής συνάρτησης δηλαδή του makespan. Ένα τέτοιο πρόβλημα συμβολίζεται ως εξής ***F\|prmu\|Cmax.*** Η εικόνα 1.3.1 δείχνει ένα παράδειγμα εκτέλεσης τριών εργασιών σε δύο μηχανές.



Εικόνα 1.3.1: *Παράδειγμα προβλήματος PFSP*

Ο καθορισμός μιας σειράς εργασιών (job permutation) μπορεί να συμβολιστεί ως εξής $\pi = \{\pi_1, \pi_2, ..., \pi_n\}$ όπου οι n εργασίες θα πρέπει να επεξεργαστούν με την ίδια σειρά από τις m μηχανές. Ορίζουμε με $C(\pi_j, m)$ τον χρόνο ολοκλήρωσης της εργασίας π_j στην

μηχανή m. Ο χρόνος ολοκλήρωσης του προβλήματος χρονοπρογραμματισμού συνεχούς ροής αντιμετάθεσης δίδεται σύμφωνα με τη σειρά επεξεργασίας $\pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ ως ακολούθως:

$$C(\pi, 1) = p_{\pi_1,1} \quad (1)$$

$$C(\pi_j, 1) = C(\pi_{j-1}, 1) + p_{\pi,1} \ , \ j = 2, \ldots, n \quad (2)$$

$$C(\pi_1, i) = C(\pi_1, i-1) + p_{\pi,i} \ , \ i = 2, \ldots, m \quad (3)$$

$$C(\pi_j, i) = \max(C(\pi_{j-1}, i), C(\pi_j, i-1)) + p_{\pi,i} \ , \ j = 2, \ldots, n \ , \ i = 2, \ldots, m \quad (4)$$

$$Cmax = C(\pi_n, m) \quad (5)$$

Έτσι, λοιπόν, ένα Permutation Flow Shop Problem μπορούμε να συναντήσουμε στις βιομηχανίες, όπου τα διάφορα εξαρτήματα κάποιου προϊόντος μετακινούνται από μηχανή σε μηχανή μέσω άλλων μηχανών που αναλαμβάνουν τη μεταφορά τους. Ένα τέτοιο πρόβλημα χαρακτηρίζεται ως NP-Hard, πράγμα που σημαίνει ότι:

- Κάθε πρόβλημα που ανήκει στο NP (=Non Deterministic Polynomial time- ένα σύνολο προβλημάτων απόφασης, των οποίων η λύση τους επαληθεύεται σε πολυωνυμικό χρόνο), μπορεί να μετασχηματιστεί σε πολυωνυμικό χρόνο στο πρόβλημα PFSP.
- Υπάρχει ένας αλγόριθμος με πολυπλοκότητα ψευδο-πολυωνυμικού χρόνου που επιλύει το πρόβλημα PFSP.
- Ανήκει στο σύνολο των προβλημάτων NP.[5]

1.4 Μέτρα Απόδοσης

Η θεωρητική προσέγγιση του προαναφερθέντος προβλλήματος, που ξεκινά με την περιγραφή των πόρων και των εργασιών, ολοκληρώνεται με την σύγκλιση των τελικών αποφάσεων προγραμματισμού εργασιών στην αντικειμενική συνάρτηση, όπως είναι το makespan που αναφέρθηκε στην προηγούμενη παράγραφο.

Ιδανικά, η αντικειμενική συνάρτηση θα έπρεπε να περιλαμβάνει όλα τα λειτουργικά έξοδα που απορρέουν από τον χρονοπρογραμματισμό των εργασιών, ώστε να έχουμε έναν συνολικό δείκτη ή μέτρο απόδοσης για να εκτιμήσουμε το πρόβλημα. Στην πράξη όμως, όλα τα λειτουργικά έξοδα ενός συστήματος παραγωγής είναι συχνά δύσκολο να μετρηθούν και πολλές φορές ακόμα περισσότερο δύσκολο να ανιχνευθούν. Τα κύρια λειτουργικά έξοδα που είναι και τα πιο εύκολα αναγνωρίσιμα, έχουν καθοριστεί κατά τη διάρκεια σχεδιασμού του

συστήματος παραγωγής, ενώ κόστη τα οποία σχετίχονται με τον χρονοπρογραμματισμό των εργασιών και διαδικασιών που προκύπτουν είναι συχνά δύσκολο να απομονωθούν ή μπορεί να είναι σταθερά. Παρ' όλα αυτά, έχουν επικρατήσει τρία μέτρα απόδοσης- λήψης αποφάσεων όσον αφορά τον χρονοπρογραμματισμό εργασιών. Αυτά είναι ο χρόνος διεκπεραίωσης (*turnaround*), η επικαιρότητα (*timeliness*) και η απόδοση (*throughput*).

Ο χρόνος διεκπεραίωσης είναι ο χρόνος που απαιτείται για να ολοκληρωθεί μία εργασία. Η επικαιρότητα μετρά τη συμμόρφωση μιας συγκεκριμένης εργασίας σε μία προκαθορισμένη προθεσμία, ενώ η απόδοση μετρά την παραγωγή έργου σε μία προκαθορισμένη χρονική περίοδο. Όσον αφορά τα δύο πρώτα μέτρα απόδοσης, αξίζει να αναφερθεί ότι ενώ μπορούμε να μιλάμε για χρόνο διεκπεραίωσης ή επικαιρότητα μιας συγκεκριμένης εργασίας, εντούτοις τα προβλήματα χρονοπρογραμματισμού εργασιών απαιτούν ένα μέτρο απόδοσης που θα περιγράφει όλο το σύστημα παραγωγής στο σύνολό του και όχι μια μεμονομένη εργασία. Ένα τέτοιο μέτρο είναι η απόδοση (throughput), που απευθύνεται σε όλο το σύστημα παραγωγής συνολικά.[3]

Σε ένα σύστημα παραγωγής χρονοπρογραμματισμού, η απόδοση σχετίζεται με την ελαχιστοποίηση του makespan. Δεδομένου ότι η απόδοση σχετίζεται με την ποσότητα παραγόμενου έργου στη μονάδα του χρόνου και ότι η ποσότητα παραγόμενου έργου σε ένα σύστημα **n** εργασιών είναι σταθερή, μεγιστοποιούμε τη απόδοση ελαχιστοποιώντας το makespan. Γι΄αυτόν το λόγο στα περισσότερα προβλήματα χρονοπρογραμματισμού εργασιών ως τελικός σκοπός τίθεται η ελαχιστοποίηση του makespan, πράγμα που αποτελεί και τον τελικό σκοπό της παρούσας εργασίας.

1.5 Χρονοπρογραμματισμός Εργασιών Συνεχούς Ροής με τυχαίους χρόνους επεξεργασίας

Όπως αναφέρθηκε, ο χρονοπρογραμματισμός εργασιών σε ένα σύστημα παραγωγής περιλαμβάνει τον κατάλληλο συντονισμό όλων των δραστηριοτήτων με στόχο την αύξηση της παραγωγικότητας και τη μείωση των εξόδων λειτουργίας. Αυτό αποτελεί το λεγόμενο πρόβλημα χρονοπρογραμματισμού εργασιών, του οποίου η παραδοσιακή προσέγγιση επίλυσης θεωρεί ως γνωστούς και αμετάβλητους τους χρόνους επεξεργασίας κάθε εργασίας από κάθε μηχανή. Δηλαδή, επιλύεται ένα αιτιοκρατικό πρόβλημα χρονοπρογραμματισμού εργασιών με αιτιοκρατικούς χρόνους επεξεργασίας.

Τί γίνεται όμως όταν οι παράμετροι σε ένα σύστημα παραγωγής είναι αβέβαοι ή και απρόβλεπτοι; Στην πραγματική ζωή, ένα σύστημα παραγωγής είναι εξαιρετικά πολύπλοκο να προσεγγιστεί με ένα αιτιοκρατικό μοντέλο, δεδομένου ότι οι αφίξεις των παραγγελιών μπορεί να έχουν καθυστερήσεις, ο ρυθμός ελαττωματικότητας των προϊόντων δεν είναι προκαθορισμένος αλλά τυχαίος, μπορεί να διακοπεί απροειδοποίητα η λειτουργία των

μηχανών για κάποιο λόγο, να αλλάξει η προτεραιότητα παραγωγής ενός προϊόντος κτλ. Έτσι, σε ένα τέτοιο σύστημα, η λύση ενός αιτιοκρατικού προβλήματος δεν μπορεί να αποφέρει την επιθυμητή ακρίβεια που αυτό απαιτεί. Γι' αυτό το λόγο, για να περιγράψουμε με μεγαλύτερη ακρίβεια το πρόβλημα, όπως αυτό εμφανίζεται στην καθημερινή ζωή, χρησιμοποιούμε το αιτιοκρατικό μοντέλο αλλά θεωρούμε ότι οι χρόνοι επεξεργασίας κάθε εργασίας από τις μηχανές είναι τυχαίοι και δεν είναι γνωστοί εξ΄αρχής παρά μόνο όταν ολοκληρωθεί η συγκεκριμένη εργασία. Ένα τέτοιο σύστημα ονομάζεται στοχαστικό σύστημα χρονοπρογραμματισμού εργασιών [7].

Επομένως, σε αυτό το σημείο μπορούμε να ορίσουμε το στοχαστικό σύστημα χρονοπρογραμματισμού εργασιών ως ένα σύστημα m μηχανών και n εργασιών στο οποίο ο χρόνος επεξεργασίας μιας ή περισσότερων εργασιών δεν είναι σταθερός και προκαθορισμένος αλλά δίδεται ως μια τυχαία μεταβλητή που ακολουθεί μία γνωστή συνάρτηση κατανομής. Η τυχαία αυτή μεταβλητή, που συμβολίζεται ως T_{ij} , έχει αθροιστική συνάρτηση κατανομής $F(t)_{ij}$, όπου ισχύει:

$$F(t)_{ij} = \Pr(T_{ij} \leq t) \tag{6}$$

Όμως, ένα τέτοιο σύστημα για να μελετηθεί, χρειάζεται να λάβουμε υπόψη μας μερικές θεωρίες. Για παράδειγμα, έστω ένα σύστημα παραγωγής, το οποίο εξελίσσεται με τον χρόνο, Στο αιτιοκρατικό σύστημα, έχοντας ορίσει έναν συγκεκριμένο προγραμματισμό εργασιών, ξέρουμε ακριβώς τί θα συμβεί από την αρχή έως το τέλος της όλης διαδικασίας. Αντιθέτως, μελετώντας μία απρόβλεπτη σειρά γεγονότων σε ένα στοχαστικό μοντέλο, καθώς οι τυχαίες μεταβλητές αποκτούν μία συγκεκριμένη τιμή αφού ολοκληρωθούν οι εργασίες, αποκτούμε νέες πληροφορίες που μπορούν να οδηγήσουν σε αλλαγές στον προγραμματισμό των εργασιών που εκκρεμούν [4]. Με αυτές τις νέες πληροφορίες μας δίνεται αρκετή ελευθερία και ευελιξία στη λήψη αποφάσεων και έτσι μπορούμε να καταλήξουμε σε δύο πολιτικές χρονοπρογραμματισμού εργασιών:

- Στατική πολιτική, η οποία περιγράφει τον χρονοπρογραμματισμό εργασιών ως σταθερό και αμετάβλητο με την πάροδο του χρόνου και ο οποίος καθορίζεται πριν την έναρξη της επεξεργασίας των εργασιών. Η πολιτική αυτή δεν επιτρέπει μία εργασία να διακοπεί στη μέση προκειμένου να γίνει επεξεργασία κάποιας άλλης εργασίας που επείγει περισσότερο, έστω και αν η σειρά όλων των εργασιών είναι προκαθορισμένη. Αντίθετα, στην πολιτική αυτή εντάσσεται η παραδοχή ότι όταν κάποια μηχανή ξεκινά να επεξεργάζεται κάποια εργασία, αυτή η εργασία θα επεξεργαστεί ως το τέλος.Αυτή η πολιτική είναι κατά μία έννοια παρόμοια με την πολιτική που ακολουθείται στα αιτιοκρατικά προβλήματα.

- Δυναμική πολιτική, κατά την οποία λαμβάνονται αποφάσεις κατά την εξέλιξη της επεξεργασίας των εργασιών. Αυτή μπορεί να εφαρμόζεται:

  - *Χωρίς διακοπές εργασιών* , δηλαδή μόνο όταν μία εργασία έχει ολοκληρωθεί, μπορούμε να εκτιμήσουμε την κατάσταση και να αναθέσουμε κάποια εφικτή εργασία σε έναν ελεύθερο επεξεργαστή.
  - *Με διακοπές εργασιών*, όπου οποιαδήποτε στιγμή μπορούμε να εκτιμήσουμε την κατάσταση και να κάνουμε οποιαδήποτε δυνατή αλλαγή στον προγραμματισμό των εργασιών [4].

Οι δυναμικές πολιτικές μπορεί να είναι πολύ δύκολο και πολύπλοκο να προσδιοριστούν, δεδομένου ότι περιλαμβάνουν πολλές υποθέσεις του τύπου "αν συμβεί αυτό, τότε κάνε εκείνο". Μπορούμε, ωστόσο, να τις ορίσουμε εκτενώς σε χρόνο μηδέν. Αυτό που δεν μπορούμε να κάνουμε είναι να εγγυηθούμε ότι θα δουλεύουμε πάντα μια εργασία που βρίσκεται πρώτη στη λίστα των προτεραιοτήτων μας.

Ποιά όμως είναι η καλύτερη πολιτική που χρειάζεται να ακολουθήσουμε στη λήψη αποφάσεων για να οδηγηθούμε στη βέλτιστη λύση; Με την αβεβαιότητα που εισάγεται πλέον στο σύστημά μας, δεδομένου ότι μιλάμε για στοχαστικά προβλήματα, δε μπορούμε να είμαστε σίγουροι ποιά πολιτική είναι η καλύτερη. Έτσι, συγκρίνοντας δύο πραγραμματισμούς εργασιών S_1 και S_2, μπορούμε να οδηγηθούμε στο ότι το S_1 έχει μικρότερο makespan με κάποια πιθανότητα p, ενώ το S_2 "κερδίζει" με πιθανότητα 1-p. Σε οποιαδήποτε από τις δύο περιπτώσεις, η διαφορά στα makespan μπορεί να είναι μικρή ή και μεγάλη. Γι' αυτό δεν μπορεί να είναι προφανές ποιός προγραμματισμός εργασιών είναι ο προτιμότερος. Έτσι, οδηγούμαστε στην προσπάθεια να κατατάξουμε τις διάφορες τιμές που μπορεί να πάρουν οι τυχαίες μεταβλητές, μία διαδικασία που ονομάζεται στοχαστική διάταξη ή στοχαστική κυριαρχία (stochastic ordering or stochastic dominance) [4].

Εν συνεχεία, έστω ότι το X είναι μία τυχαία μεταβλητή με συνάρτηση πυκνότητας πιθανότητας $f(x)_X$ και αθροιστική συνάρτηση κατανομής $F(x)_X$ και ισχύει το ίδιο για το Y. Αυτές οι δύο τυχαίες μεταβλητές μπορούν να συγκριθούν με τρεις τρόπους:

- Αναμενόμενη διάταξη (Expectation Ordering), που αποτελεί την απλούστερη και πιο πρακτική βάση σύγκρισης:

  Αναμένεται ότι το X είναι μικρότερο από το Y ↔ E(X) ≤ E(Y)

- Στοχαστική διάταξη (Stochastic Ordering) :

  Το X είναι τυχαία μικρότερο από το Y ↔ P(X > t) ≤ P(Y > t) ∀ t

Αυτό φυσικά υπονοεί ότι $f(t)_X \geq f(t)_Y$ ∀ t. Επίσης, γράφουμε ότι $X \leq_{st} Y$, πράγμα που σημαίνει ότι χονδρικά το X έχει μεγαλύτερη πιθανότητα να περιέχει μικρές τιμές (η

πιθανότητα μάζας είναι μεγαλύτερη στις μικρές τιμές) έτσι ώστε η αθροιστική συνάρτηση κατανομής να συσσωρεύεται γρηγορότερα. Πάντως, μία σίγουρη λύση σε τέτοιου είδους προβλήματα δε μπορεί να είναι σίγουρη δεδομένου ότι συχνά οι αθροιστικές συναρτήσεις κατανομής των τυχαίων μεταβλητών τέμνονται μεταξύ τους και έτσι δε μπορεί να κυριαρχήσει των υπολοίπων μία και μόνη λύση.

- Σχεδόν σίγουρη διάταξη (Almost sure ordering) :

Το X είναι σχεδόν σίγουρα μικρότερο από το Y $\leftrightarrow$ $P(X \leq Y) = 1$

Σε αυτήν τη περίπτωση, γράφουμε $X \leq_{as} Y$. Αν μπορούσαμε να εισάγουμε αυτή τη σχέση μεταξύ δύο makespan (κάτι το οποίο δεν είναι δυνατό), θα οδηγούμαστε σε ένα απίστευτα καλό αποτέλεσμα. Αν πάλι το υποθέσουμε ότι ισχύει μεταξύ των χρόνων επεξεργασίας, η κατάσταση δεν θα ήταν πολύ διαφορετκή από τα αποτελέσματα που παίρνουμε από το αιτιοκρατικό μοντέλο. Τέλος ισχύει ότι $X \leq_{st} Y \rightarrow E(X) \leq E(Y)$.

Θα πρέπει εν συνεχεία να αναφέρουμε ότι για το στοχαστικό πρόβλημα χρονοπρογραμματισμού εργασιών υπάρχει ένας περιορισμένος αριθμός αποτελεσμάτων και τα περισσότερα από αυτά εφαρμόζουν στατική πολιτική προκειμένου να βρούν μία βέλτιστη λύση. Ακόμα και για την περίπτωση του προβλήματος όπου έχουμε μόνο δύο μηχανές, δεν υπάρχουν αναλυτικά αποτελέσματα χωρίς να τεθούν αυστηρές προϋποθέσεις

Η απλούστερη ανάλυση της θεωρίας στοχαστικών προβλημάτων χρονοπρογραμματισμού εργασιών είναι αυτή της επέκτασης του προβλήματος των δύο μηχανών στην περίπτωση όπου οι χρόνοι επεξεργασίας A_j, B_j των εργασιών J_j στις μηχανές M_1 και M_2 είναι εκθετικά κατανεμημένοι και στοχεύουμε να ελαχιστοποιήσουμε το αναμενόμενο makespan (*F2|(pmtn), expPij| E(Cmax)*) . Έστω ότι οι χρόνοι επεξεργασίας των εργασιών έχουν μέση τιμή $a_j = E(A_j)$, $b_j = E(B_j)$ και ρυθμό $\alpha_j = {}^{1}/_{a_j}$ και $\beta_j = {}^{1}/_{b_j}$ [4]. Το makespan θα είναι και εδώ *Cmax* = $max_{j=1\ldots n}(R_j)$ αλλά το R_j είναι τώρα μία τυχαία μεταβλητή όπου: $R_j = \sum_{i=1}^{j} A_i + \sum_{i=j}^{n} B_j$ και *E(Cmax) = E(*$max_j R_j$*)* $\neq max_j E(R_j)$.

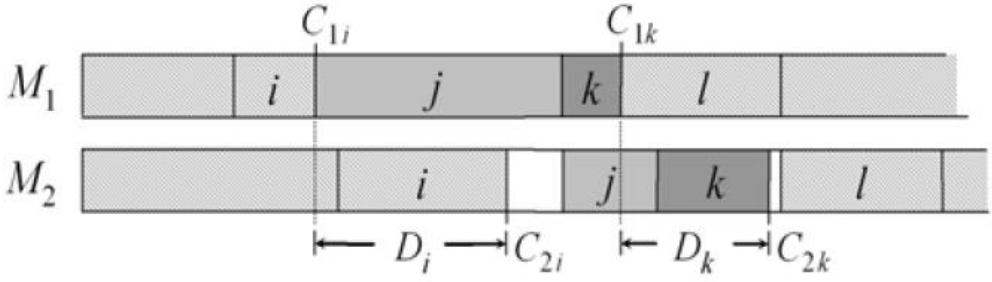
Παρ' όλα αυτά, αν οι χρόνοι επεξεργασίας είναι εκθετικά κατανεμημένοι, υπάρχει ακόμα μια απλή λύση του προβλήματος. Αυτή κάποιες φορές καλείται "το θεώρημα του Talwar", του οποίου η απόδειξη έγινε από τον Weiss το 1982.

ΘΕΩΡΗΜΑ: Για το πρόβλημα συνδυαστικής βελτιστοποίησης *F2|(pmtn),exp Pij|E(Cmax)* , χρησιμοποιώντας στατική ή δυναμική πολιτική, με ή χωρίς διακοπές, ισχύει ότι :

$$S^* = \searrow (a_j - \beta_j) \qquad (7)$$

όπου ↘ $(a_j - \beta_j)$ σημαίνει ότι οι διαφορές των ρυθμών των εργασιών είναι σε μη αύξουσα σειρά [4].

ΑΠΟΔΕΙΞΗ: Πρώτα από όλα, θα δείξουμε ότι ο προγραμματισμός S^* είναι βέλτιστος για κάθε στατική πολιτική δια της ατόπου απαγωγής. Θεωρούμε λοιπόν, έναν προγραμματισμό $S \neq S^*$. Θα πρέπει να υπάρχουν γειτονικές εργασίες J_j και J_k στο S, τέτοιες ώστε η εργασία J_j να προηγείται της J_k και $a_j - \beta_j < a_k - \beta_k$. Θα δείξουμε ότι αν αντιμεταθέσουμε τις εργασίες μεταξύ τους, αυτό οδηγεί στη μείωση του makespan.



Εικόνα 1.5.1: *Μία απεικόνιση του προγραμματισμού S.*

Όπως φαίνεται στην εικόνα 1.5.1, οι εργασίες που προηγούνται της J_j στο S καταλαμβάνουν τις μηχανές M_1 και M_2 μέχρι τις χρονικές στιγμές C_{1i} και C_{2i} αντίστοιχα, όπου η εργασία J_i είναι η τελευταία εργασία πριν να εκτελεστεί η J_j . Έστω ότι $D_i = C_{2i} - C_{1i}$ είναι η περίσσεια πληρότητα ή εναλλαγή του M_2 στο M_1 . Παρόμοια, ακολουθώντας τη J_k , η περίσσεια πληρότητα είναι $D_k = C_{2k} - C_{1k}$ και 'εστω ότι η επόμενη εργασία είναι η J_l .

Τώρα, αντιμεταθέτουμε τις εργασίες J_j και J_k και αφήνουμε τις υπόλοιπες εργασίες στη σειρά που ήδη βρίσκονται. Ονομάζουμε τον καινούριο προγραμματισμό S' και διακρίνουμε τους νέους χρόνους επεξεργασίας από τους πρώτους. Η πληρότητα του M_1 πριν την εκτέλεση της εργασίας J_l δεν μεταβάλλεται, αφού το $C'_{1j} = C_{1k} = C_{1i} + A_j + A_k$. Επομένως, τα C'_{1j} και C_{1k} είναι ταυτόσημα και έχουν την ίδια κατανομή. Τι γίνεται όμως με την πληρότητα του M_2 ; Πώς συγκρίνεται το C_{2k} με το C'_{2j}; Και επειδή δεν αλλάζει τίποτα στην πληρότητα του M_1, η προηγούμενη ερώτηση είναι ισοδύναμη με το πώς συγκρίνεται το D_k με το D'_j ; Στόχος μας είναι να δείξουμε ότι η εναλλαγή αυτή των εργασιών βελτιώνει τον προγραμματισμό ή πιο συγκεκριμένα ότι η πληρότητα μετά την εναλλαγή είναι στοχαστικά μικρότερη από αυτή στον προγραμματισμό S δηλαδή ότι $D'_j <_{st} D_k$. Υπάρχουν δύο περιπτώσεις:

- Αν $D_i \geq A_j + A_k$ ή $C_{2i} \geq C_{1k}$, τότε η αντιμετάθεση των δύο εργασιών δεν θα επιφέρει καμία μεταβολή αφού : $D'_j = D_k = D_i + B_j + B_k - A_j - A_k$.

- Αν $D_i \leq A_j + A_k$, τότε **πριν** γίνει η αντιμετάθεση, η πληρότητα D_k είχε την κατανομή:

$$P(D_k > \mathrm{t} \mid D_i \leq A_j + A_k) = \frac{\beta_j}{\alpha_k + \beta_j} e^{-\beta_k t} + \frac{\alpha_k}{\alpha_k + \beta_j}\left(\frac{\beta_k}{\beta_k - \beta_j} e^{-\beta_j t} + \frac{\beta_j}{\beta_j - \beta_k} e^{-\beta_k t}\right) \qquad (8)$$

Αυτό συμβαίνει γιατί με $D_i \leq A_j + A_k$, τα A_k και B_j ξεκινούν ταυτόχρονα. (Στην πραγματικότητα, αν ισχύει ότι $D_i > A_j$ τότε ο χρόνος επεξεργασίας A_k ξεκινά νωρίτερα από τον B_j αλλά όταν η D_i τελειώνει , η ιδιότητα απώλειας μνήμης μας επιτρέπει να θεωρήσουμε ότι ο A_k ξεκινά εκ νέου.) Ο πρώτος όρος του δεξιού σκέλους της εξίσωσης (8), με πιθανότητα $\frac{\beta_j}{\alpha_k + \beta_j}$ προκύπτει όταν ο χρόνος επεξεργασίας B_j τελειώνει πρώτος με $D_k = B_k$. Στην δεύτερη περίπτωση, όταν $B_j > A_k$ τότε η πληρότητα D_k ισούται με το άθροισμα του εναπομένοντος B_j και του B_k.

Μετά την αντιμετάθεση, η πιθανότητα $P(D'_j > \mathrm{t} \mid D_i \leq A_j + A_k)$ παίρνει την ίδια μορφή με προηγουμένως αλλά με τους δείκτες j και k ο ένας στη θέση του άλλου. Μπορούμε τώρα να δείξουμε ότι αυτή η αντιμετάθεση έκανε την περίσσεια πληρότητα D_k στοχαστικά μικρότερη. Έτσι, μετά απο υπολογισμούς και δεδομένου ότι $D_i \leq A_j + A_k$, βρίσκουμε ότι :

$$P(D_k > t) - \mathrm{P}(D'_j > t) = \frac{\beta_\mathrm{j}\beta_\mathrm{k}}{(\alpha_\mathrm{j} + \beta_\mathrm{k})(\alpha_\mathrm{k} + \beta_\mathrm{j})} \frac{\mathrm{e}^{-\beta_\mathrm{k} t} - \mathrm{e}^{-\beta_\mathrm{j} t}}{\beta_\mathrm{j} - \beta_\mathrm{k}} (\alpha_\mathrm{k} - \beta_\mathrm{k} - [\alpha_\mathrm{j} - \beta_\mathrm{j}]) \geq 0 \qquad (9)$$

Επομένως, μετά από όλα αυτά, καταλήγουμε στο ότι για κάθε αλληλουχία εργασιών πριν και μετά από τις εργασίες J_j και J_k , αν η χρονική στιγμή C_{1k} δεν μεταβάλλεται και η C_{2k} μειώνεται, τότε το makespan μπορεί μόνο να μειώνεται. Έτσι, με D'_j στοχαστικά μικρότερη από την D_k, το αναμενόμενο makespan μειώνεται. Αυτή η απόδειξη ισχύει για στατικές πολιτικές.

Όσον αφορά τώρα δυναμικές πολιτικές χωρίς διακοπές εργασιών, θα πρέπει να αναφέρουμε ότι η αλληλουχία εργασιών στη μηχανή M_2 είναι επουσιώδης. Δεδομένης της αλληλουχίας εργασιών στη M_1 , οι εργασίες που φτάνουν στη M_2 θα πρέπει να επεξεργαστούν το συντομότερο δυνατό. Η ανακατανομή τους δεν θα επηρρεάσει το makespan. Ας υποθέσουμε τώρα ότι στη M_1 μία εργασία έχει ολοκληρωθεί. Τι πληροφορία θα μπορούσαμε να αντλήσουμε από αυτό που θα μας οδηγήσει να αλλάξουμε τον αρχικό προγραμματισμό; Η μόνη σχετική πληροφορία είναι το ποσό της εναπομείνουσας εργασίας που πρέπει να γίνει στη M_2 , ή αλλιώς η περίσσεια πληρότητα, που είναι τώρα μία τυχαία μεταβλητή που πρέπει να καθοριστεί. Αλλά για οποιαδήποτε πιθανή αλληλουχία εργασιών η

απόφαση είναι ίδια με αυτή της στατικής πολιτικής, γι' αυτό και θα πρέπει η τελευταία να παραμείνει η καλύτερη επιλογή.

ΚΕΦΑΛΑΙΟ 2
ΕΥΡΕΤΙΚΟΙ-ΜΕΘΕΥΡΕΤΙΚΟΙ ΑΛΓΟΡΙΘΜΟΙ

2.1 Ευρετικοί Αλγόριθμοι

Στο προηγούμενο κεφάλαιο αναφερθήκαμε σε προβλήματα δύσκολα, προβλήματα για τα οποία ο αριθμός των περιπτώσεων προς εξέταση ή των υποψήφιων λύσεων είναι απαγορευτικά μεγάλος για να εξεταστούν όλες οι λύσεις μία προς μία. Επομένως, για να επιλυθούν τέτοια προβλήματα χρησιμοποιούμε τεχνικές που μας οδηγούν σε μία σχεδόν βέλτιστη αλλά ικανοποιητική λύση. Αυτές οι τεχνικές ονομάζονται ευρετικοί αλγόριθμοι, δηλαδή, πρόκειται για αλγορίθμους που βρίσκουν λύσεις ανάμεσα σε πολλές εφικτές αλλά δεν εγγυώνται εξ΄αρχής την εύρεση μιας βέλτιστης λύσης. Άρα μπορούν να θεωρηθούν ως προσεγγιστικοί και μη ακριβείς αλγόριθμοι, συνήθως όμως βρίσκουν λύση "κοντά" στη βέλτιστη και τη βρίσκουν εύκολα και “γρήγορα” [8].

Βέβαια, μία λύση ενός αλγορίθμου για να γίνει αποδεκτή πρέπει να ικανοποιεί κάποια κριτήρια. Τέτοια είναι η ποιότητα της λύσης, δηλαδή η απόκλισή της από τη βέλτιστη, η ευκολία απόκτησης μιας λύσης όπως πχ. ο χρόνος που απαιτεί ένας αλγόριθμος για να την παράξει και η λογική πάνω στην οποία στηρίζονται οι κανόνες του ευρετικού αλγορίθμου που χρησιμοποιήθηκαν για να οδηγηθούμε στη λύση [9]. Επιπλέον, κάθε πρόβλημα βελτιστοποίησης μπορεί να επιλυθεί με μία πληθώρα ευρετικών αλγορίθμων που οδηγούν στη βέλτιστη λύση αλλά έχουν αναπτυχθεί και αρκετοί αλγόριθμοι που συγκρινόμενοι μεταξύ τους οδηγούν σε ολοένα και καλύτερες λύσεις.

Όλοι αυτοί οι ευρετικοί αλγόριθμοι που έχουν αναπτυχθεί, μπορούν να ταξινομηθούν σε τρεις κατηγορίες, η κάθεμια από τις οποίες έχει κάποια ιδιαίτερα χαρακτηριστικά. Αυτές είναι:

- Αλγόριθμοι Απληστίας (Greedy Algorithms). Οι Αλγόριθμοι Απλιστίας προσπαθούν να οδηγήσουν σε μία εφικτή λύση του προβλήματος αλλά πολλές φορές απαιτούν πολύ χρόνο γιατί είναι μυωπικοί αλγόριθμοι, δηλαδή μεγιστοποιούν το κέρδος σε κάθε ένα βήμα μιας διαδικασίας χωριστά, χωρίς να εξετάζουν τις επιπτώσεις που αυτό μπορεί να έχει στα επόμενα βήματα και επομένως στο τελικό αποτέλεσμα.

- Προσεγγιστικοί Αλγόριθμοι (Approximation Algorithms). Οι προσεγγιστικοί Αλγόριθμοι προσπαθούν να λύσουν ένα πρόβλημα χρησιμοποιώντας επιπλέον πληροφορία.

- Αλγόριθμοι Τοπικής Αναζήτησης (Local Search Algorithms). Οι Αλγόριθμοι Τοπικής Αναζήτησης προσπαθούν ξεκινώντας από μια αρχική εφικτή λύση να βελτιώσουν τη λύση με κάποια μέθοδο αναζήτησης στη γειτονιά της λύσης.

Η βασική διαφορά των τριών κατηγοριών έγκειται στο ότι οι δύο πρώτες κατηγορίες χρησιμοποιούνται για την δημιουργία μιας αρχικής λύσης, ενώ η τρίτη προσπαθεί να βελτιώσει μια ήδη υπάρχουσα λύση [9].

2.2 Μεθευρετικοί Αλγόριθμοι

Πολλές φορές ο υπολογισμός μιας βέλτιστης λύσης από έναν ευρετικό αλγόριθμο μπορεί να αποδειχθεί δύσχρηστος για πολλά προβλήματα συνδυαστικής βελτιστοποίησης με βιομηχανικές ή επιστημονικές εφαρμογές. Στην πράξη, ένας ευρετικός αλγόριθμος παράγει συνήθως "καλές" λύσεις, που όμως δεν είναι οι βέλτιστες και αυτό γιατί είναι πολύ πιθανό να προσκολληθεί σε κάποιο τοπικό ελάχιστο.

Γι΄αυτόν το λόγο, για να ξεφύγουμε από ένα τέτοιο τοπικό ελάχιστο χρησιμοποιούμε προσεγγιστικές τεχνικές βελτιστοποίησης, τους μεθευρετικούς αλγορίθμους. Ο όρος "μεθευρετικός" επινοήθηκε από τον καθηγητή Fred W. Glover θέλοντας να περιγράψει "μια ανώτερη στρατηγική η οποία καθοδηγεί και τροποποιεί άλλους ευρετικούς αλγορίθμους στο να παράγουν λύσεις πέρα από αυτές που παράγονται κατά την έρευνα της ευνοϊκότερης τοπικής συνθήκης (Αλγόριθμοι Τοπικής Αναζήτησης).

Ένας μεθευρετικός αλγόριθμος αποτελεί μία ευφυή διαδικασία επαναληπτικής βελτίωσης. Αυτή χρησιμοποιεί μη εξαρτημένους από το εξεταζόμενο πρόβλημα μηχανισμούς καθοδήγησης υποδεέστερων ευρετικών, με σκοπό την επίτευξη ευρωστίας (robustness). Με τον όρο ευρωστία εννοούμε την ισορροπία ανάμεσα στην ικανότητα παραγωγής υψηλής ποιότητας λύσεων σε συγκεκριμένα προβλήματα και στην ευελιξία που απαιτείται για την επιβίωση σε πολλά διαφορετικά περιβάλλοντα όπως διαφορετικοί περιορισμοί, πόροι κτλ. Επομένως, στόχος ενός τέτοιου αλγορίθμου είναι ο όσο το δυνατόν ταχύτερος προσδιορισμός των περιοχών με υψηλής ποιότητας λύσεις και ταυτόχρονα η αποφυγή της σπατάλης χρόνου σε περιοχές όπου είτε έχουν εξερευνηθεί, είτε δεν παρέχουν υψηλής ποιότητας λύσεις.

Ποιοι όμως είναι οι τρόποι με τους οποίους οι μεθευρετικοί αλγόριθμοι επιτυγχάνουν το στόχο τους; Πρόκειται για στρατηγικές που σκοπό έχουν να εντατικοποιούν την έρευνα, ώστε να αυξήσουν τα περιθώρια βελτίωσης. Άλλες στρατηγικές που χρησιμοποιούνται, διαφοροποιούν την περιοχή ερεύνης, ενώ άλλες παρέχουν τρόπους απεγκλωβισμού από τα

τοπικά βέλτιστα, οδηγώντας είτε σε καλύτερες είτε σε χειρότερες λύσεις. Τέλος, άλλες μέθοδοι αξιοποιούν το ιστορικό της πορείας της έρευνας προκειμένου να παραχθούν νέες λύσεις.

Μιλώντας για εντατικοποίηση, αναφερόμαστε σε επικέντρωση της έρευνας σε περιοχές του χώρου λύσεων με υψηλής ποιότητας λύσεις. Επικεντρώνεται δηλαδή η έρευνα σε μικρά σημεία του χώρου λύσεων. Η διαφοροποίηση, εν συνεχεία, αφορά τη μετακίνηση της έρευνας σε ανεξερεύνητες περιοχές όταν κρίνεται απαραίτητο. Σκοπός είναι να εξεταστούν τελείως διαφορετικές περιοχές του χώρου έρευνας.

Οι μηχανισμοί Εντατικοποίησης και Διαφοροποίησης αποτελούν τις κινητήριες δυνάμεις της μεθευρετικής έρευνας. Είναι κατά κανόνα ανεξάρτητες από το υπό εξέταση πρόβλημα, αλληλοσυμπληρώνονται και αντικρούονται. Η δυναμική ισορροπία τους καθορίζει την αποτελεσματικότητα ενός μεθευρετικού αλγορίθμου, γι΄αυτό και οι μεθευρετικοί αλγόριθμοι διαφέρουν λόγω του διαφορετικού τρόπου που επιχειρούν να πετύχουν αυτήν την ιδανική ισορροπία ανάμεσα στη διαφοροποίηση και την εντατικοποίηση.

Επομένως, η αναζήτηση του βέλτιστου αποφεύγοντας ταυτόχρονα την παγίδα του τοπικού ελαχίστου οδήγησε στην δημιουργία πολλών διαφορετικών αλγορίθμων που έγιναν ιδιαίτερα δημοφιλείς κατά τη δεκαετία του 1980. Οι αλγόριθμοι αυτοί χωρίζονται σε τέσσερις κύριες κατηγορίες ανάλογα με τη μέθοδο που χρησιμοποιούν για να αποφύγουν το τοπικό βέλτιστο:

- Επαναληπτικές διαδικασίες που αρχίζουν από διαφορετικές αρχικές λύσεις. Σε αυτήν την κατηγορία ανήκουν οι αλγόριθμοι πολυεναρκτήριας τοπικής αναζήτησης (Multistart Local Search), της επαναληπτικής τοπικής αναζήτησης (Iterated Local Search), και η μέθοδος της διαδικασίας της άπληστης τυχαιοποιημένης προσαρμοστικής αναζήτησης (Greedy Randomized Adaptive Search Procedure - GRASP) [9].

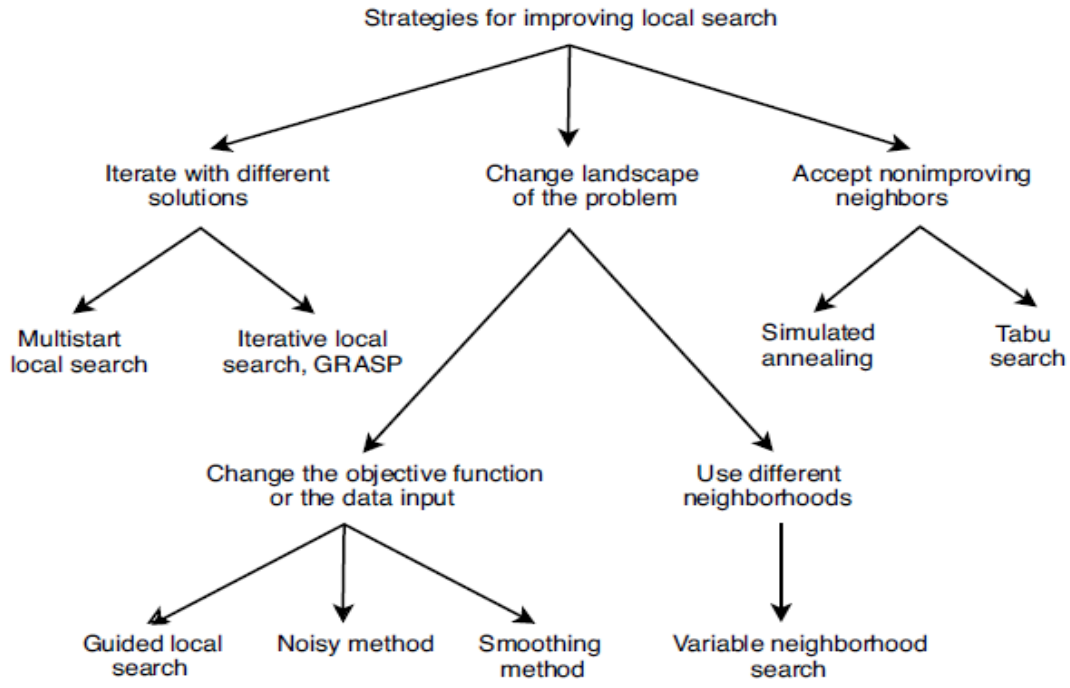
- Αλγόριθμοι οι οποίοι δέχονται γειτονικές κινήσεις που δε βελτιώνουν τη λύση. Αυτοί οι αλγόριθμοι μπορούν να κάνουν υπό κάποιες συνθήκες αποδεκτή μία κίνηση, η οποία δε βελτιώνει τη λύση. Έτσι, μπορεί σε κάποια επόμενη κίνηση να διαφύγουμε από το τοπικό ελάχιστο και να οδηγηθούμε σε κάποιο επόμενο τοπικό ελάχιστο, καλύτερο από το τρέχων. Χαρακτηριστικά παραδείγματα αυτής της κατηγορίας αποτελούν η προσομοιωμένη ανόπτηση (Simulated Annealing) και η περιορισμένη αναζήτηση (Tabu Search) [9].

- Αλγόριθμοι που αλλάζουν τη γειτονιά έρευνας. Αυτή η κατηγορία αποτελείται από αλγόριθμους οι οποίοι όταν κολλήσουν σε ένα τοπικό ελάχιστο αλλάζουν

τον αλγόριθμο που χρησιμοποιούν για την αναζήτηση σε γειτονικά σημεία του χώρου λύσεων. Οι πιο χαρακτηριστικές μέθοδοι αυτής της κατηγορίας είναι ο αλγόριθμος μεταβλητής γειτονιάς αναζήτησης (Variable Neighborhood Search - VNS) και ο αλγόριθμος επέκτασης της γειτονιάς αναζήτησης (Expanding Neighborhood Search - ENS) [9] .

- Αλγόριθμοι που αλλάζουν την αντικειμενική συνάρτηση ή κάποια από τα δεδομένα του προβλήματος. Σε αυτήν την κατηγορία των αλγορίθμων το πρόβλημα μετατρέπεται είτε αλλάζοντας την αντικειμενική συνάρτηση είτε αλλάζοντας τους περιορισμούς του προβλήματος. Η πιο χαρακτηριστική μέθοδος σε αυτή την κατηγορία είναι ο αλγόριθμος καθοδηγούμενης τοπικής αναζήτησης (Guided Local Search) [9].

Στην εικόνα 2.2.1 φαίνονται οι κατηγορίες αλγορίθμων που αναφέρθηκαν προηγουμένως.

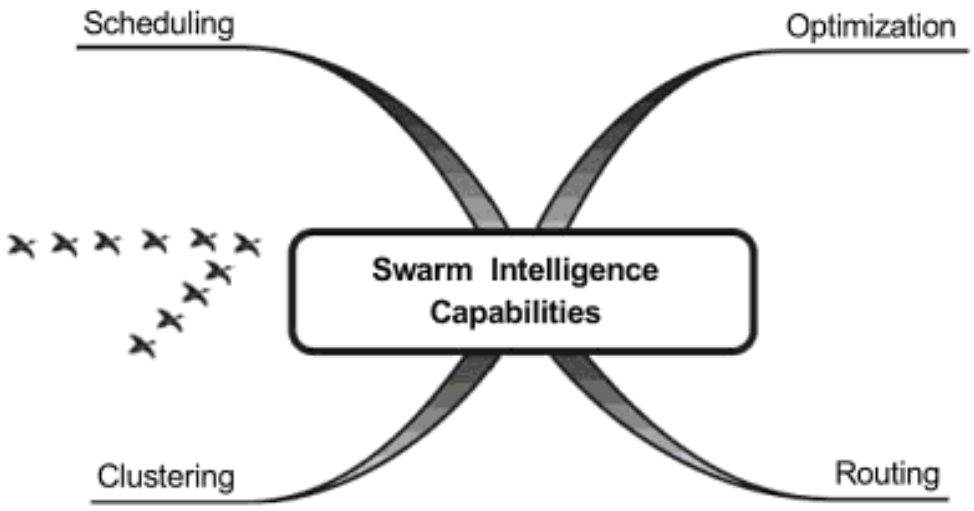


Εικόνα 2.2.1: *Δέντρο μεθόδων βελτιωσης της τοπικής αναζήτησης και αποφυγής του τοπικού βέλτιστου.*

2.3 Αλγόριθμοι που βασίζονται στην ευφυΐα του Σμήνους.

Αναφέραμε στην προηγούμενη παράγραφο ότι οι μεθευρετικοί αλγόριθμοι ταξινομούνται σε τέσσερις κατηγορίες ανάλογα με τη μέθοδο που χρησιμοποιούν για να αποφύγουν ένα τοπικό βέλτιστο. Στην παρούσα εργασία θα επικεντρωθούμε ιδιαίτερα στην κατηγορία αλγορίθμων που αλλάζουν τη γειτονιά αναζήτησης και μάλιστα σε αλγορίθμους που βασίζονται στην ευφυΐα του σμήνους.

Ως ευφυΐα σμήνους (Swarm Intelligence) ορίζεται το σύνολο των συλλογικών ικανότητων επίλυσης προβλημάτων των κοινωνικών ζώων. Η ευφυΐα σμήνους είναι το άμεσο αποτέλεσμα της συλλογικής οργάνωσης του σμήνους. Μέσα σε αυτή τη δομή οργάνωσης, οι αλληλεπιδράσεις των μελών χαμηλού επιπέδου δημιουργούν μία δυναμική δομή σε καθολικό επίπεδο, η οποία μπορεί να θεωρηθεί ως νοϋμοσύνη. Αυτές οι αλληλεπιδράσεις χαμηλού επιπέδου καθορίζονται από κάποιους απλούς κανόνες, τους οποίους τα μέλη χαμηλού επιπέδου ακολουθούν χωρίς να έχουν επίγνωση τις επιδράσεις αυτών των κανόνων σε καθολικό επίπεδο. Έτσι, τα μέλη ενός σμήνους ή μιας αποικίας έχουν μόνο γνώσεις τοπικού περιεχομένου για το περιβάλλον στο οποίο βρίσκονται. Στη συνέχεια, χρησιμοποιώντας άμεσες ή έμμεσες μεθόδους επικοινωνίας, οι αλληλεπιδράσεις χαμηλού επιπέδου επηρεάζουν όλη την οργάνωση της αποικίας [11].



Εικόνα 2.3.1: *Δυνατότητες της ευφυΐας του σμήνους*

Αυτή η συλλογική συμπεριφορά των μελών μιας αποικίας ή ενός σμήνους που είναι αποτέλεσμα της αλληλεπίδρασης που υπάρχει μεταξύ των μελών ή αλλιώς αυτή η αυτο-οργάνωση απαρτίζεται από τέσσερις συνιστώσες. Πρώτη συνιστώσα αποτελεί η θετική ανατροφοδότηση. Πρόκειται για ένα σύνολο απλών κανόνων οι οποίοι συντελούν στο να

παραχθεί μια πιο σύνθετη δομή. Για παράδειγμα, όταν μία μέλισσα ανακαλύψει μια περιοχή με αρκετά λουλούδια, μέσω της θετικής ανατροφοδότησης ειδοποιεί και τις υπόλοιπες μέλισσες να έρθουν σε αυτήν την περιοχή. Επόμενη συνιστώσα είναι αυτή τη φορά η αρνητική ανατροφοδότηση, η οποία μειώνει τις επιδράσεις της θετικής ανατροφοδότησης και βοηθά στην δημιουργία ενός μηχανισμού εξισορρόπησης. Παράδειγμα μηχανισμού εξισορρόπησης αποτελεί π.χ. ο περιορισμένος αριθμός ζωοτροφών, που έχει ως αποτέλεσμα να μην μετακινηθεί όλος ο πληθυσμός μελισσών στη συγκεκριμένη περιοχή, αφού υπάρχει τροφή μόνο για έναν συγκεκριμένο αριθμό μελισσών. Τρίτο στοιχείο της αυτο-οργάνωσης είναι η τυχαιότητα. Η τελευταία προσθέτει έναν συντελεστή αβεβαιότητας στο σύστημα και επιτρέπει στα σμήνη να ανακαλύψουν νέες λύσεις ακόμη και για τα πιο σύνθετα προβλήματά τους. Τέλος, αξίζει να αναφερθεί πως υπάρχουν πολλαπλές αλληλεπιδράσεις και μεταξύ των μεμονομένων μελών του σμήνους. Αναφερόμαστε σε έναν περιορισμένο αριθμό μεμονομένων μελών που αλληλεπιδρούν μεταξύ τους προκειμένου να μετατρέψουν τις ανεξάρτητες τοπικές τους δραστηριότητες σε έναν διασυνδεδεμένο ζωντανό οργανισμό [12].

Συνδυάζοντας, εν συνεχεία, όλα αυτά τα στοιχεία οδηγούμαστε σε ένα αποκεντρωμένο σύστημα, στο οποίο δεν υπάρχει κεντρικός έλεγχος [12]. Υπάρχει αντίθετα μια ιεραρχική δομή που χρησιμοποιείται για να αναθέσει καθήκοντα και έτσι δεν υπάρχει έλεγχος απευθείας πάνω στα άτομα του πληθυσμού αλλά εμμέσως μέσω των ενστίκτων τους. Αυτό συντελεί στη δημιουργία αποτελεσματικών και δυναμικών δομών οι οποίες βοηθούν το σύστημα να ανταπεξέλθει στις διάφορες προκλήσεις.

Υπάρχουν πολλά διαφορετικά ήδη ζώων που επωφελούνται από παρόμοιες διαδικασίες, οι οποίες τα βοηθούν να επιβιώσουν και να δημιουργήσουν νέες και καλύτερες γενιές. Τέτοια για παράδειγμα είναι οι μέλισσες, τα μηρμύγκια, τα σμήνη πουλιών και τα κοπάδια ψαριών. Μέσα σε αυτά τα αποτελεσματικά συστήματα, κάποια μεμονομένα μέλη βρίσκουν ασφάλεια και τροφή. Επιπλέον, και κάποιες άλλες πιο σύνθετες μορφές ζωής ακολουθώντας αυτούς τους κανόνες επωφελούνται από τη δύναμη του πλήθους. Άλλωστε, μέχρι ενός σημείου και το ανθρώπινο σώμα θα μπορούσε να θεωρηθεί ένα τέτοιο αυτοσυντηρούμενο σύστημα. Και αυτό γιατί όλα τα κύτταρα του σώματος επωφελούνται από τη δύναμη του ενός από το άλλο και μοιράζονται τις διάφορες εργασίες, ώστε να ανταπεξέλθουν σε συνθήκες τις οποίες θα αδυνατούσε να αντιμετωπίσει ένα και μοναδικό κύτταρο [12].

Οι αλγόριθμοι βελτιστοποίησης βασισμένοι στα σμήνη (Swarm-based optimization algorithms - SOAs) μιμούνται τις φυσικές μεθόδους οι οποίες οδηγούν την αναζήτηση στη βέλτιστη λύση. Αυτοί οι αλγόριθμοι διαφοροποιούνται από τους ευρετικούς ως προς το γεγονός ότι οι SOAs χρησιμοποιούν για κάθε επανάληψη έναν πληθυσμό από λύσεις και όχι μία μεμονωμένη λύση. Έτσι, ένας πληθυσμός λύσεων υφίσταται επεξεργασία κατά τη διάρκεια μιας επανάληψης με αποτέλεσμα τη δημιουργία ενός καινούριου πληθυσμού λύσεων. Και αν ένα πρόβλημα βελτιστοποίησης έχει ένα μόνο βέλτιστο, τότε τα μέλη του πληθυσμού θα συγκλίνουν προς αυτή τη βέλτιστη λύση. Αν όμως υπάρχουν περισσότερα του

ενός βέλτιστα, τότε ένας αλγόριθμος SO μπορεί να χρησιμοποιηθεί για να τα εγκλωβίσει στον τελικό πληθυσμό του [12].

Οι SO αλγόριθμοι περιλαμβάνουν τους Εξελικτικούς και τους Γενετικούς Αλγορίθμους, τους Αλγορίθμους Βελτιστοποίησης Σμήνους Σωματιδίων (Particle Swarm Optimization - PSO), τον Αλγόριθμο Τεχνητής Αποικίας Μελισσών (Artificial Bee Colony Optimization Algorithm - ABC), και τον Αλγόριθμο Βελτιστοποίησης Αποικίας Μυρμηγκιών (Ant Colony Optimization) [12]. Επομένως, οι μέθοδοι βασιζόμενοι σε πληθυσμούς αποτελούν στρατηγικές που παράγουν παραλλαγές της λύσης που αναζητείται. Κάποιες μέθοδοι αναζήτησης χρησιμοποιούν κάποιο κριτήριο απληστίας προκειμένου να αποφασίσουν ποιά παραγόμενη λύση να κρατήσουν. Ένα τέτοιο κριτήριο θα μπορούσε για παράδειγμα να είναι η αποδοχή μιας νέας λύσης αν και μόνο αν αυτή αυξάνει/μειώνει την τιμή της αντικειμενικής συνάρτησης.

2.4 Αλγόριθμος Τεχνητής Αποικίας Μελισσών για Συνεχή Προβλήματα (Artificial Bee Colony Optimization Algorithm - ABC)

Στη φύση, οι μέλισσες παρουσιάζουν πολλές πολύπλοκες συμπεριφορές όπως το ζευγάρωμα (mating), η ανατροφή (breeding) και η αναζήτηση τροφής (foraging). Τέτοιες συμπεριφορές έχουν μιμηθεί πολλοί αλγόριθμοι βελτιστοποίησης που βασίζονται στην ευφυΐα του σμήνους που αναφέραμε στην προηγούμενη παράγραφο. Στην παρούσα διπλωματική θα μελετήσουμε έναν συγκεκριμένο αλγόριθμο που βασίζεται σε συμπεριφορές μελισσών, τον Αλγόριθμο Τεχνητής Αποικίας Μελισσών (Artificial Bee Colony - ABC).

Ο Αλγόριθμος Τεχνητής Αποικίας Μελισσών αποτελεί έναν αλγόριθμο βασισμένο στην ευφυΐα του σμήνους και εμπνευσμένο από τη συμπεριφορά των μελισσών και προτάθηκε από τον Karaboga το 2005. Ο αλγόριθμος αυτός, που εφαρμόζεται κυρίως σε προβλήματα συνεχούς βελτιστοποίησης, προσομοιώνει την συμπεριφορά αναζήτησης τροφής των μελισσών. Έτσι, ένα σύνολο μελισσών δηλαδή ένα σμήνος μπορεί να εκτελέσει διάφορες εργασίες με επιτυχία μέσω της συνεργασίας με τα άλλα μέλη του σμήνους.

Εικόνα 2.4.1: *Η αναζήτηση νέκταρ από εργάτρια μέλισσα.*

Στον αλγόριθμο ABC υπάρχουν τρεις τύποι μελισσών : οι εργάτριες μέλισσες (employed bees), οι μέλισσες θεατές (onlooker bees) και οι ανιχνεύτριες (scout bees). Οι εργάτριες αναζητούν τροφή γύρω από ένα προκαθορισμένο σύνολο πηγών τροφής (πιθανές λύσεις του προβλήματος) και μοιράζονται αυτήν την πληροφορία με τις μέλισσες θεατές μέσω του χορού των μελισσών. Οι μέλισσες θεατές είναι μέλισσες που περιμένουν στην κυψέλη και με βάση την πληροφορία που παίρνουν από τις εργάτριες αναζητούν μία καλύτερη πηγή τροφής στην περιοχή που βρίσκεται η τροφή που τους υπέδειξαν οι εργάτριες [9]. Οι πηγές τροφής με την καλύτερη ποιότητα ή μεγαλύτερη ποσότητα (fitness) θα έχουν μεγαλύτερη πιθανότητα να επιλεγούν από τις μέλισσες θεατές από αυτές με τη χαμηλότερη ποιότητα τροφής. Τέλος, οι ανιχνεύτριες μετασχηματίζονται σε εργάτριες που εγαταλείπουν τις πηγές τροφής και αναζητούν καινούριες.

Πιο συγκεκριμένα, στον αλγόριθμο ABC το πρώτο μισό του σμήνους αποτελείται από εργάτριες και το δεύτερο μισό από ανιχνεύτριες. Ο αριθμός των εργατριών ή των ανιχνευτριών ισούται με τον αριθμό λύσεων του σμήνους (δεν είναι δεσμευτικό) δεδομένου ότι κάθε θέση των πηγών τροφής (νέκταρ) αντιπροσωπεύει και μία εφικτή λύση του προβλήματος. Σε αυτό το σημείο θα πρέπει όμως να ορίσουμε κάποιες παραμέτρους πριν προχωρήσουμε με την εξήγηση του αλγορίθμου ABC. Αυτές είναι : η διάσταση του χώρου λύσεων D, ο αρχικός αριθμός πηγών τροφής SN που είναι τυχαία κατανεμημένος, το όριο εξερεύνησης μιας λύσης που δε βελτιώνεται limit και ο μέγιστος αριθμό επαναλήψεων MCN. Επομένως, η θέση μιας πηγής τροφής αναπαρίσταται ως X_i = (x_{i1} , x_{i2} , ... , x_{iD}), όπου i = 1, 2, ... , SN [14].

Αρχικά, λοιπόν, οι εργάτριες διασκορπίζονται τυχαία στο χώρο προκειμένου να τον εξερευνήσουν και να υπολογίσουν την τιμή της αντικειμενικής συνάρτησης (νέκταρ) της αντίστοιχης θέσης που βρίσκονται (πηγή τροφής). Κάθε εργάτρια X_i παράγει μια καινούρια λύση V_i στη γειτονιά της παρούσας θέσης της, δηλαδή εξερευνεί την παρούσα θέση της για μία καλύτερη λύση. Αυτό αναπαρίσταται από την εξίσωση όπου :

$$V_{ij} = X_{ij} + \Phi_{ij} \cdot (X_{ij} - X_{rj}) \quad (10)$$

Στην παραπάνω εξίσωση το V_{ij} είναι η θέση μιας καινούριας παραγόμενης λύσης και το X_{ij} είναι η θέση της αρχικής i-οστής πηγής τροφής. Το r ϵ $\{ 1, 2, \ldots, SN \}$ και r ≠ i με j ϵ { 1,2,..., D }. Επιπλέον, τα i και j είναι τυχαίοι ακέραιοι, ενώ το Φ_{ij} ακολουθεί ομοιόμορφη κατανομή ($\Phi_{ij} \sim U[-1,1]$) [14].

Στην βασική έκδοση του ABC ορίζουμε τη συνάρτηση ποιότητας ή fitness function ως εξής:

$$fit_i = \begin{cases} \frac{1}{1 + f_i}, & f_i \geq 0 \\ 1 + |f_i|, & f_i < 0 \end{cases} \quad (11)$$

όπου το f_i αποτελεί την τιμή της αντικειμενικής συνάρτησης της i-οστής πηγής τροφής ενώ το fit_i είναι η τιμή της αντικειμενικής συνάρτησης της i-οστής πηγής τροφής μετά από μετατροπή [14]. Δηλαδή, η fitness function είναι μια αριθμητική αναπαράσταση της τιμής της αντικειμενικής συνάρτησης που χρησιμοποιείται για να μετατρέψει την τιμή της αντικειμενικής συνάρτησης σε έναν μη αρνητικό αριθμό και μπορεί να έχει διαφορετικές τιμές από αυτές της αντικειμενικής συνάρτησης. Μας βοηθά, με άλλα λόγια, να εκτιμήσουμε την καινούρια τιμή της αντικειμενικής συνάρτησης. Σύμφωνα λοιπόν με την εξίσωση (11), η αντικειμενική συνάρτηση της καινούριας πηγής τροφής μπορεί να υπολογιστεί και να συγκριθεί με τον εαυτό της και αν η καινούρια λύση έχει καλύτερη αντικειμενική συνάρτηση, θα αντικαταστήσει την παλιά λύση αλλιώς θα απορριφθεί.

Όταν όλες οι εργάτριες έχουν ολοκληρώσει την αναζήτησή τους, μοιράζονται τις πληροφορίες των πηγών τροφής με τις θεατές μέλισσες μέσω του χορού των μελισσών. Κάθε μέλισσα θεατής αξιολογεί την πληροφορία του νέκταρ (δηλαδή την τιμή της αντικειμενικής συνάρτησης) που παίρνει από όλες τις εργάτριες και επιλέγει μία πηγή τροφής με πιθανότητα που συνδέεται άμεσα με την ποσότητα/ποιότητα του νέκταρ. Αυτή η επιλογή βάση πιθανότητας βασίζεται στον κανόνα της ρουλέτας που περιγράφεται από την παρακάτω εξίσωση:

$$P_i = \frac{fit_i}{\sum_j^{SN} fit_j} \quad (12)$$

όπου fit_i είναι η συνάρτηση ποιότητας της της i-οστής πηγής τροφής του σμήνους. Όπως φαίνεται, όσο καλύτερη είναι μία πηγή τροφής i, τόσο μεγαλύτερη πιθανότητα έχει η i-οστή πηγή τροφής να επιλεγεί.

Όταν όμως μια λύση δε βελτιώνεται για έναν αριθμό επαναλήψεων που ξεπερνάει το limit, τότε η συγκεκριμένη πηγή τροφής εγκαταλείπεται και τότε επιστρατεύονται οι ανιχνεύτριες μέλισσες ώστε να ανακαλύψουν μία καινούρια καλύτερη πηγή τροφής σύμφωνα με την παρακάτω εξίσωση:

$$X_{ij} = lb_j + \Psi_{ij} \cdot (ub_j - lb_j) \quad (13)$$

όπου τα lb και ub είναι τα κατώτατα και ανώτατα όρια των X_{ij} και Ψ_{ij} ~ U[0,1] [14].

Σε αυτό το σημείο πρέπει να τονιστεί ότι οι μέλισσες αποτελούν διαδικασίες που εφαρμόζονται σε κάποια πηγή τροφής - λύση προκειμένου να τη βελτιώσουν. Άρα αν σε κάποια πηγή τροφής ακολουθήσουν 10 μέλισσες την εργάτρια που βρήκε την πηγή τροφής, αυτό σημαίνει ότι θα πραγματοποιηθούν 10 προσπάθειες για εύρεση νέας πηγής τροφής βάσει της εξίσωσης (10) [9]. Παρακάτω δίνονται τα κύρια βήματα του αλγορίθμου ABC σε μορφή ψευδοκώδικα:

- Καθορίζεται ο αρχικός αριθμός πηγών τροφής.
- Καθορίζεται ο αριθμός των εργατριών.
- Καθορίζεται ο αριθμός των μελισσών θεατών.
- REPEAT
- Κάθε εργάτρια επιλέγει τυχαία μία πηγή τροφής και υπολογίζει την ποσότητα του νέκταρ σε αυτή (τιμή της αντικειμενικής συνάρτησης).
- Επιστρέφει στην κυψέλη και και μοιράζεται την πληροφορία με τις μέλισσες θεατές μέσω του χορού των μελισσών.
- Κάθε μέλισσα θεατής παρακολουθεί τον χορό των εργατριών και επιλέγει μία από τις πηγές τροφής, αναλόγως τον χορό και πηγαίνει στην επιλεγμένη πηγή τροφής.
- Αποφασίζεται ποιες πηγές τροφής θα εγκαταλειφθούν και αντικαθίστανται με τις καινούριες πηγές τροφής που εντοπίζουν οι ανιχνεύτριες.
- Σε όλες τις περιπτώσεις αποθηκεύεται η καλύτερη πηγή τροφής που έχει βρεθεί.
- UNTIL (μέχρι να πληρούνται οι απαιτήσεις)

2.5 Αλγόριθμος Τεχνητής Αποικίας Μελισσών για Διακριτά Προβλήματα (Discrete Artificial Bee Colony)

Ο αλγόριθμος ABC αρχικά σχεδιάστηκε να επιλύει συνεχή προβλήματα βελτιστοποίησης, στα οποία μάλιστα είναι και αρκετά αποτελεσματικός. Τί γίνεται όμως αν εφαρμόσουμε τον ABC σε διακριτά προβλήματα; Στην παρούσα διπλωματική θα χρησιμοποιήσουμε τον ABC για την επίλυση του διακριτού προβλήματος αντιμετάθεσης χρονοπρογραμματισμού εργασιών και μάλιστα με χρόνους επεξεργασίας των εργασιών στοχαστικούς, όπως θα δούμε σε επόμενη ενότητα. Σε αυτό το κεφάλαιο, θα παρουσιάσουμε μερικούς τρόπους μετατροπής του ABC από συνεχή σε διακριτό αλγόριθμο βελτιστοποίησης.

Αρχικά, θα πρέπει να αναφέρουμε ότι στον αλγόριθμο ABC κάθε πηγή τροφής, δηλαδή κάθε πιθανή λύση του προβλήματος , αναπαρίσταται με μία αλληλουχία εργασιών $\pi = \{\pi_1, \pi_2, ..., \pi_n\}$. Κάθε εργάτρια, λοιπόν, θα πρέπει αρχικά να παράξει μία καινούρια πηγή τροφής (αλληλουχία εργασιών) που βασίζεται σε αυτήν που είχε παράξει προηγουμένως και να κρατήσει αυτή με το περισσότερο νέκταρ. Στη συνέχεια, κάθε μέλισσα θεατής θα πρέπει να πάει στην επιλεγμένη (μέσω του χορού μελισσών) πηγή τροφής και να αναζητήσει στην περιοχή της μία ακόμη καλύτερη λύση. Τέλος, θα πρέπει και οι ανιχνεύτριες να αναζητήσουν καινούριες πηγές τροφής για να διευρύνουν τον χώρο λύσεων. Βλέπουμε δηλαδή τρεις λειτουργίες στις οποίες κάθε μέλισσα θα πρέπει να χρησιμοποιήσει κάποιες συγκεκριμένες μεθόδους για να παράξει μία καινούρια λύση και άρα να επιτύχει τον σκοπό της. Σε ένα διακριτό πρόβλημα βελτιστοποίησης μερικές από τις μεθόδους παραγωγής νέων λύσεων είναι οι εξής:

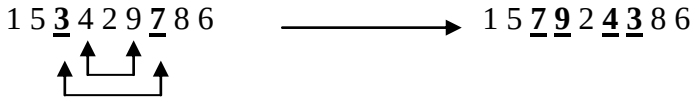
- *SWAP* : επιλέγει τυχαία δύο διαφορετικές εργασίες από μία αλληλουχία εργασιών (πηγή τροφής) και τις αντιμεταθέτει π.χ.

  1 5 **3** 4 2 9 **7** 8 6 ⟶ 1 5 **7** 4 2 9 **3** 8 6

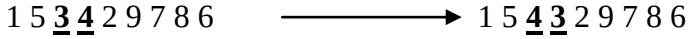
- *INSERT*: επιλέγει τυχαία δύο διαφορετικές εργασίες από μία αλληλουχία εργασιών και εισάγει την πρώτη ακριβώς πριν από την δεύτερη, μετατοπίζοντας παράλληλα και όλες τις προηγούμενες εργασίες της δεύτερης προς τα πίσω:

  1 5 **3** 4 2 9 **7** 8 6 ⟶ 1 5 4 2 9 **3** **7** 8 6

- *INVERSE*: επιλέγει τυχαία δύο διαφορετικές εργασίες από μία αλληλουχία εργασιών, τις αντιμεταθέτει και αντιμεταθέτει και όλες τις εργασίες που βρίσκονται εσωτερικά από τις πρώτες:



- *ADJACENT EXCHANGE*: επιλέγει τυχαία δύο γειτονικές εργασίες από μία αλληλουχία εργασιών και τις αντιμεταθέτει:



Κάθε μέθοδος από τις προαναφερθείσες μπορεί να έχει διαφορετική απόδοση κατά την δημιουργία μιας καινούριας λύσης. Παρ' όλα αυτά, αυτές οι μέθοδοι παρουσιάζουν ξεχωριστά πλεονεκτήματα που αν συνδυαστούν αποδοτικά μπορούν να επιλύσουν διάφορα προβλήματα αντιμετάθεσης χρονοπρογραμματισμού εργασιών. Για παράδειγμα θα μπορούσαν να συνδυαστούν ώστε να αποτελέσουν την τοπική αναζήτηση μιας εργάτριας μέλισσας για καλύτερη πηγή τροφής ή την τοπική αναζήτηση μιας ανιχνεύτριας για την εύρεση μιας καθ' όλα καινούριας λύσης. Ο τελικός στόχος είναι πάντα η εύρεση μιας καλύτερης τιμής της αντικειμενικής συνάρτησης (νέκταρ) από την προηγούμενη.

ΚΕΦΑΛΑΙΟ 3
ΒΙΒΛΙΟΓΡΑΦΙΚΗ ΑΝΑΣΚΟΠΗΣΗ

3.1 Βιβλιογραφική ανασκόπηση του αιτιοκρατικού PFSSP

Στην παρούσα ενότητα θα παρουσιάσουμε μερικές προηγούμενες μελέτες όσον αφορά την επίλυση του προβλήματος αντιμετάθεσης χρονοπρογραμματισμού εργασιών (PFSSP). Το πρόβλημα PFSSP έχει αποτελέσει ένα πολύ ενεργό πεδίο έρευνας πάνω στους ευρετικούς και μεθευρετικούς αλγορίθμους κυρίως λόγω της δυσκολίας που συναντούν οι ακριβείς αλγόριθμοι στην επίλυση μεσαίων ή μεγάλων προβλημάτων.

Το πρόβλημα αντιμετάθεσης χρονοπρογραμματισμού εργασιών μελετήθηκε πρώτα από τον Johnson το 1954, ο οποίος πρότεινε έναν απλό κανόνα για την εύρεση της βέλτιστης αλληλουχίας εργασιών σε ένα πρόβλημα PFSSP - δύο μηχανών. Αυτή η μελέτη έστρεψε το ενδιαφέρον πολλών ερευνητών προς το PFSSP, οι οποίοι προσπάθησαν να λύσουν το πρόβλημα αυτό με περισσότερες από δύο μηχανές χρησιμοποιώντας πολλούς και διαφορετικούς αλγορίθμους. Μια τέτοια προσπάθεια αποτελεί η μελέτη των Campbell et al. (1970) πάνω στον ευρετικό αλγόριθμο CDS (Campbell, Dudek and Smith), αλλά και οι βελτιώσεις του Dannenbring (1977) πάνω σε άλλους ευρετικούς αλγορίθμους. Επίσης, ο ευρετικός NEH (Nawaz, Enscore, Ham) που επινοήθηκε από τους Nawaz et al. (1983) θεωρείται ως η καλύτερη και πιο αποδοτική κατασκευαστική ευρετική μέθοδος. Επιπλέον, υπάρχουν εξίσου αποδοτικές μέθοδοι όπως ο βελτιωμένος ευρετικός αλγόριθμος του Suliman (2000) ή η μελέτη του ευρετικού NEH από τους Framinan et al. (2003). Αργότερα, οι Dong et al (2008) παρουσίασαν έναν βελτιωμένο ευρετικό αλγόριθμο NEH για το προβλημα αντιμετάθεσης χρονοπρογραμματισμού εργασιών, ενώ οι Kalczynski και Kamburowski (2008) πρότειναν έναν εξίσου βελτιωμένο NEH ευρετικό για την ελαχιστοποίηση του makespan.

Οι μεθευρετικοί αλγόριθμοι, από την άλλη μεριά, εμφανίστηκαν πολύ αργότερα σε σχέση με τους ευρετικούς. Μέσα από αυτούς ξεχωρίζουμε τον αλγόριθμο της προσομοιωμένης ανόπτησης (simulated annealing) των Osman και Potts (1989) που είναι αρκετά απλός, δεδομένου ότι χρησιμοποιεί μια θερμοκρασία που αναλόγως της τιμής της ο αλγόριθμος δέχεται και χειρότερες ή λιγότερο χειρότερες λύσεις και μπορεί με αυτόν τον τρόπο να διευρύνει τη γειτονιά αναζήτησής του. Επίσης, και ο αλγόριθμος SPIRIT των Widmer and Hertz (1989) είναι εξίσου αποδοτικός και αποτελείται από δύο φάσεις. Στην πρώτη φάση, παράγεται μια αρχική λύση που βασίζεται στο Open Traveling Salesman Problem (OTSP) και στη δεύτερη φάση χρησιμοποιείται η τεχνική του αλγορίθμου tabu search που λειτουργεί εναλλάσοντας τις γειτονιές αναζήτησης. Άλλοι αξιοσημείωτοι αλγόριθμοι που βασίζονται στον tabu search, προτάθηκαν από τον Taillard (1990) και τον Reeves (1993). Αυτοί οι δύο αλγόριθμοι εκμεταλλεύτηκαν την ιδιότητα της εισαγωγής

(insert) και χρησιμοποίησαν τον αλγόριθμο NEH για να παράξουν μια αρχική λύση. Επίσης, οι Nowicki και Smutnicki (1996) ανέπτυξαν έναν γρήγορο αλγόριθμο tabu search για το PFSSP.

Στην επίλυση του προβλήματος PFSSP έχουν συμβάλλει σημαντικά και οι γενετικοί αλγόριθμοι, οι οποίοι εμφανίστηκαν στις εργασίες των Chen et al. (1995), του Reeves (1995) αλλά και των Wang και Zheng (2003) και των Aldowaisan και Allahvedi (2003). Οι γενετικοί αλγόριθμοι που ανέπτυξαν οι προαναφερθέντες μελετητές έχουν το κοινό χαρακτηριστικό ότι η δημιουργία του αρχικού πληθυσμού λύσεων δεν γίνεται τυχαία όπως σε πολλούς άλλους γενετικούς αλγορίθμους. Αντίθετα, η δημιουργία του αρχικού πληθυσμού των λύσεων επιτυγχάνεται από τους ευρετικούς αλγορίθμους που αναφέρθηκαν παραπάνω, τον αλγόριθμο NEH ή και με τυχαίες αντιμεταθέσεις εργασιών. Άλλες μελέτες περιλαμβάνουν τον αλγόριθμο επαναληπτικής τοπικής αναζήτησης (iterated local search algorithm) όπως η μελέτη που πραγματοποιήθηκε από τον Stutzle (1998) ή τον αλγόριθμο βελτιστοποίησης της αποικίας μυρμηγκιών (ant colony optimization) που προτάθηκε από τον Marco Dorigo (1992). Αυτοί οι δύο τελευταίοι αλγόριθμοι επίσης βασίζονται στην αρχικοποίηση μέσω του αλγορίθμου NEH. Αργότερα, οι Ying και Liao (2004) πρότειναν ένα σύστημα βασισμένο στην αποικία των μυρμηγκιών προκειμένου να ελαχιστοποιήσουν το makespan του PFSSP. Οι Tasgetiren et al. (2007) παρουσίασαν έναν αλγόριθμο βελτιστοποίησης σμήνους σωματιδίων (particle swarm optimization) για το makespan ή την ελαχιστοποίηση του συνολικού χρόνου ροής (total flow time) του PFSSP. Τέλος, οι Liao et al (2007) ανέπτυξαν μια διακριτή εκδοχή του αλγορίθμου βελτιστοποίησης σμήνους σωματιδίων για το PFSSP ενώ οι Vallada και Ruiz (2009) έλυσαν το πρόβλημα αντιμετάθεσης με μία συνεργατική μεθευρετική μέθοδο.

Ο Αλγόριθμος Τεχνητής Αποικίας Μελισσών (ABC) προτάθηκε από τον Karaboga (2005) για πολυδιάστατα και πολύτροπα προβλήματα βελτιστοποίησης και βασίζεται στην συμπεριφορά των μελισσών για αναζήτηση τροφής. Ο αλγόριθμος αυτός έχει πολύ πρόσφατα χρησιμοποιηθεί από τους Pan et al (2011). Οι τελευταίοι πρότειναν μια διακριτή εκδοχή του αλγορίθμου (discrete artificial bee colony - DABC) για την επίλυση του lot-streaming flow shop scheduling problem. Όσον αφορά το πρόβλημα της αντιμετάθεσης χρονοπρογραμματισμού εργασιών, οι Tasgetiren et al. (2011) πρότειναν έναν επίσης διακριτό αλγόριθμο τεχνητής αποικίας μελισσών αλλά με αντικειμενικό σκοπό την ελαχιστοποίηση του total flow time. Για την ελαχιστοποίηση του makespan του προβλήματος PFSSP δύο μελέτες έχουν παρουσιαστεί από όσο γνωρίζουμε. Η πρώτη αποδίδεται στους Yan-Feng Liu και San-Yang Liu (2011) οι οποίοι δημιούργησαν έναν υβριδικό διακριτό αλγόριθμο τεχνητής αποικίας μελισσών στον οποίο η δημιουργία του αρχικού πληθυσμού λύσεων γίνεται από τον αλγόριθμο GRASP που βασίζεται στον NEH. Η δεύτερη πραγματοποιήθηκε από τους X. Li και M. Yin (2012) οι οποίοι χρησιμοποίησαν διάφορες τεχνικές μετάλλαξης προκειμένου να ελαχιστοποιήσουν το makespan του PFSSP.

3.2 Βιβλιογραφική ανασκόπηση του στοχαστικού PFSSP

Το στοχαστικό πρόβλημα αντιμετάθεσης χρονοπρογραμματισμού εργασιών έχει μελετηθεί πολύ λιγότερο από το αντίστοιχο αιτιοκρατικό μοντέλο και είναι σαφώς πιο σύνθετο από το τελευταίο. Πιο συγκεκριμένα, εκτός από τον κανόνα υπεροχής (dominance rule) που εισήχθηκε από τον Makino (1965) για το πρόβλημα των δύο μηχανών, δεν έχει δοθεί, μέχρι τώρα, καμία ακριβής λύση χωρίς να γίνουν παραδοχές για για την κατανομή των χρόνων επεξεργασίας των εργασιών. Για το πρόβλημα των δύο μηχανών και δεδομένου ότι οι χρόνοι επεξεργασίας των εργασιών ακολουθούν εκθετική κατανομή, ο Talwar (1967) συμπέρανε μία ακριβής λύση για το στοχαστικό πρόβλημα χρονοπρογραμματισμού εργασιών. Αυτή η λύση, η οποία αργότερα αποδείχτηκε ότι είναι και βέλτιστη από τους Cunningham και Dutta (1973) είναι πλέον γνωστή ως ο κανόνας του Talwar.

Παρ' όλη αυτήν την πρόοδο, εντούτοις, δεν έχουν βρεθεί βέλτιστες διαδικασίες για τις υπόλοιπες περιπτώσεις στοχαστικών προβλημάτων χρονοπρογραμματισμού. Για την περίπτωση του προβλήματος με τις δύο μηχανές, έχουν προταθεί τρεις προσεγγιστικές λύσεις από τους Baker and Trietsch (2011). Αυτές οι λύσεις βασίζονται στους κανόνες του Johnson (1954) και του Talwar για το αιτιοκρατικό μοντέλο και όλες παρουσιάζουν παρόμοια - κοντά στο βέλτιστο συμπεριφορά. Για το γενικό πρόβλημα των m μηχανών, οι Baker και Altheimer (2012) πρότειναν τρεις ευρετικούς αλγορίθμους, βασιζόμενους σε τροποποιήσεις των αλγορίθμων CDS και NEH, με συμπεριφορές και εδώ παρόμοιες και κοντά στο βέλτιστο.

Τέλος, όσον αφορά τον αλγόριθμο ABC, απ' όσο γνωρίζουμε, δεν υπάρχει κάποιος ερευνητής που να τον έχει χρησιμοποιήσει για την επίλυση του στοχαστικού προβλήματος αντιμετάθεσης χρονοπρογραμματισμού εργασιών μέχρι σήμερα.

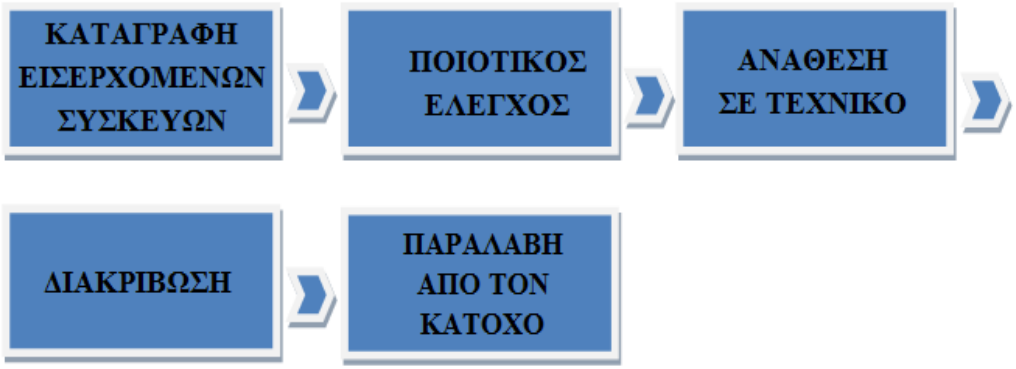
ΚΕΦΑΛΑΙΟ 4
ΜΟΝΤΕΛΟΠΟΙΗΣΗ ΠΡΟΒΛΗΜΑΤΟΣ

4.1 Μοντελοποίηση Πραγματικού Συστήματος Παραγωγής

Στην παρούσα ενότητα θα μοντελοποιήσουμε ένα πραγματικό σύστημα παραγωγής που προέρχεται από τον εργασιακό μου χώρο. Πρόκειται για ένα εργαστήριο που παρέχει υπηρεσίες διακρίβωσης οργάνων μέτρησης.

Με τον όρο δικρίβωση εννοούμε τη μέθοδο με την οποία προσδιορίζουμε την ικανότητά ενός οργάνου μέτρησης να μετράει σωστά. Αυτό συνήθως γίνεται συγκρίνοντας τις μετρήσεις του με αυτές ενός οργάνου μεγαλύτερης ακρίβειας, ή χρησιμοποιώντας ένα πρότυπο αναφοράς οι ιδιότητες του οποίου έχουν προσδιοριστεί με ακρίβεια (άρα είναι και αυτό διακριβωμένο). Η διαδικασία της διακρίβωσης πραγματοποιείται στα πλαίσια του συστήματος διασφάλισης ποιότητας που μπορεί μια εταιρεία να εφαρμόζει, προκειμένου να πιστοποιήσει την ποιότητα των οργάνων, εργαλείων και συσκευών που διαθέτει.

Το σύστημα παραγωγής του συγκεκριμένου εργαστηρίου διακρίβωσης περιλαμβάνει μια σειρά επιμέρους διεργασιών που ακολουθεί μία συσκευή προκειμένου να διακριβωθεί. Δηλαδή η παραγωγική διαδικασία απαρτίζεται από πέντε φάσεις οι οποίες είναι:



Εικόνα 4.1.1: *Ροή που ακολουθεί μια εργασία από την είσοδό της στο εργαστήριο διακρίβωσης μέχρι την έξοδό της.*

Δηλαδή, μία συσκευή όταν εισέρχεται για πρώτη φορά σε ένα εργαστήριο διακρίβωσης, καταγράφεται από τον εργαζόμενο που είναι υπεύθυνος για την παραλαβή της και στη συνέχεια περνάει από ποιοτικό έλεγχο για να διαπιστωθεί η κατάστασή της και να

γραφούν οποιεσδήποτε δυσλειτουργίες ή φθορές που μπορεί να έχει εξ' αρχής. Έπειτα, περνά από το τμήμα ελέγχου παραγωγής, όπου ανατίθεται η συσκευή στον κατάλληλο τεχνικό. Αυτό συμβαίνει γιατί ανάλογα με το είδος της συσκευής (ηλεκτρονική ή μηχανική) την αναλαμβάνει και άλλος τεχνικός. Τέλος, η συσκευή διακριβώνεται και επιστρέφει στον κάτοχο μαζί με κατάλληλο πιστοποιητικό που αποδεικνύει ότι ελέγχθηκε.

Για τη μοντελοποίηση του συγκεκριμένου συστήματος παραγωγής, θα θεωρήσουμε ότι εισέρχονται στο εργαστήριο **20** συσκευές και είναι όλες ηλεκτρονικές. Άρα μόνο ένας τεχνικός διακριβωτής θα αναλαμβάνει τη ρύθμισή τους. Έτσι, κάθε συσκευή που εισέρχεται στο εργαστήριο θα περνά υποχρεωτικά από **5** εργαζομένους (παραλαβή-ποιοτικός έλεγχος-ανάθεση σε τεχνικό-διακρίβωση-επιστροφή στον χρήστη) και η σειρά με την οποία θα περνά από αυτούς είναι συγκεκριμένη και δεν μπορεί να αλλάξει.

Ένα τέτοιο σύστημα παραγωγής, λοιπόν, μπορεί να συγκριθεί με το μοντέλο αντιμετάθεσης χρονοπρογραμματισμού εργασιών για τους εξής λόγους:

1. Περιλαμβάνει m εργαζομένους που πρέπει να επεξεργαστούν - διακριβώσουν n συσκευές.

2. Η σειρά με την οποία περνά κάθε συσκευή από κάθε εργαζόμενο είναι προκαθορισμένη και δεν μπορεί να μεταβληθεί λόγω της συγκεκριμένης δομής του συστήματος παραγωγής.

3. Κάθε εργαζόμενος μπορεί να αναλάβει μόνο μία συσκευή και μία συσκευή μπορεί να ανατεθεί μόνο σε έναν εργαζόμενο.

4. Γίνεται η παραδοχή ότι όλοι οι εργαζόμενοι είναι διαθέσιμοι από τη χρονική στιγμή μηδέν και έπειτα δηλ. από τη στιγμή που εισέρχεται μια συσκευή στο εργαστήριο και μετά. Θεωρούμε ότι δεν κάνουν διάλειμμα κατά την εκτέλεση των εργασιών τους για ένα συγκεκριμένο αριθμό συσκευών (π.χ. 20).

5. Οι συσκευές από τη στιγμή που εισέρχονται στο εργαστήριο είναι έτοιμες προς επεξεργασία.

6. Δεν επιτρέπονται διακοπές. Δηλαδή από τη στιγμή που κάποιος εργαζόμενος αναλαμβάνει μια συσκευή, δεν μπορεί να διακόψει την εργασία του.

7. Αντικειμενικός σκοπός είναι η ελαχιστοποίηση του συνολικού χρόνου επεξεργασίας (makespan) των n συσκευών από τους m εργαζομένους.

Βλέπουμε, επομένως, ότι οι ομοιότητες που παρουσιάζει το συγκεκριμένο εργαστήριο διακρίβωσης με το θεωρητικό πρόβλημα αντιμετάθεσης χρονοπρογραμματισμού εργασιών

είναι πολλές και επομένως καθίσταται δυνατή η μοντελοποίηση και επίλυση του πραγματικού προβλήματος με βάση το θεωρητικό μοντέλο.

Τι γίνεται όμως με τους χρόνους επεξεργασίας κάθε συσκευής από κάθε εργαζόμενο; Είναι αιτιοκρατικοί ή στοχαστικοί; Στην παρούσα εργασία, θα μελετήσουμε τις εξής περιπτώσεις:

- Στο μεν αιτιοκρατικό μοντέλο θα θεωρήσουμε ότι οι χρόνοι επεξεργασίας των συσκευών από τους εργαζομένους είναι προκαθορισμένοι και σταθεροί και είναι αποτέλεσμα επεξεργασίας μακρόχρονων στατιστικών στοιχείων. Με αυτά τα δεδομένα θα εξάγουμε την αιτιοκρατική βέλτιστη λύση.

- Στο μεν στοχαστικό, θα γίνει μία προσπάθεια διαφοροποίησης των χρόνων επεξεργασίας, εκτιμώντας ότι αυτοί θα μεταβάλλονται από τους αντίστοιχους αιτιοκρατικούς κατά ±1. Έτσι, θα καταλήξουμε και στην διακύμανση της τιμής της αντικειμενικής συνάρτησης σε σχέση με την αιτιοκρατική τιμή της.

- Θα εξετάσουμε την περίπτωση κατά την οποία κάποια συσκευή μπορεί να έρθει με βλάβη. Δηλαδή θα χρειαστεί να μεταβληθεί ο χρόνος επεξεργασίας μόνο μίας συσκευής, ενώ οι χρόνοι επεξεργασίας των υπολοίπων συσκευών θα είναι αμετάβλητοι (αιτιοκρατικοί). Αυτήν την κατάσταση θα την προσομοιώσουμε, μελετώντας πέντε παραδείγματα όπου σε κάθε ένα εισέρχεται στο σύστημα μία συσκευή με βλάβη.

4.2 Επίλυση του πραγματικού συστήματος παραγωγής με τον ABC

Το πρόβλημα που αναφέραμε στην προηγούμενη παράγραφο επιλύεται εδώ με τον αλγόριθμο Artificial Bee Colony. Σε αυτή την παράφραφο θα περιγράψουμε τον τρόπο με τον οποίο υλοποιήθηκε ο διακριτός ABC, αρχικά για να επιλύσει το αιτιοκρατικό πρόβλημα 20 συσκευών - 5 εργαζομένων και ύστερα θα περιγράψουμε την μετατροπή του για την επίλυση του στοχαστικού προβλήματος.

- Για το αιτιοκρατικό πρόβλημα με σταθερούς χρόνους επεξεργασίας:

Ο αλγόριθμος ξεκινά διαβάζοντας, μέσω της συνάρτησης *readFile()*, από αρχείο txt τον αριθμό των συσκευών, τον αριθμό των εργαζομένων και τους χρόνους επεξεργασίας κάθε συσκευής από κάθε εργαζόμενο και τα αποθηκεύει στις μεταβλητές *jobs*, *machines* και *costs*.

Έπειτα, δημιουργεί έναν διδιάστατο πίνακα που περιέχει τις αρχικές πηγές τροφής - αρχικές λύσεις του προβλήματος. Οι αρχικές λύσεις δημιουργούνται από μία συνάρτηση που παράγει τυχαίους αριθμούς από 1 έως n, όσες είναι και οι εργασίες. Στον συγκεκριμένο αλγόριθμο δηλαδή, παράγεται ένας πίνακας 50 x 20 που δηλώνει την δημιουργία 50 πιθανών λύσεων, η κάθεμία από αυτές είναι ένας μονοδιάστατος πίνακας 20 θέσεων με τιμές από 1 έως 20.

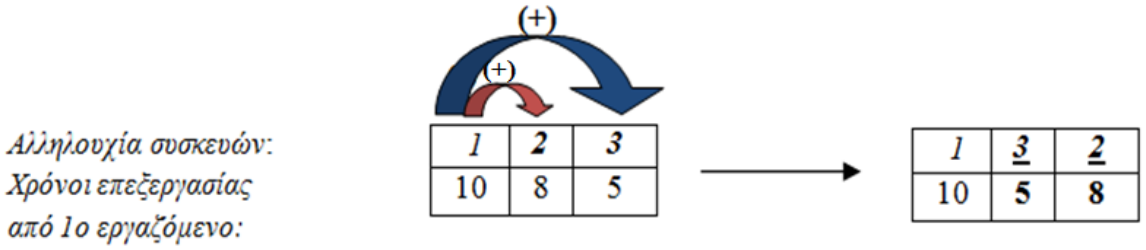
Στη συνέχεια, χρησιμοποιώντας τη συνάρτηση *computeNektar()*, η οποία καλεί μια άλλη συνάρτηση την *object()*, υπολογίζει για κάθε μία από αυτές τις 50 λύσεις την τιμή της αντικειμενικής συνάρτησης. Η συνάρτηση *object()* δέχεται σαν παράμετρο εισόδου μία πιθανή αλληλουχία συσκευών και ύστερα χρησιμοποιώντας τις εξισώσεις (1) - (5) για να υπολογίσει τον συνολικό χρόνο ολοκλήρωσης επεξεργασίας των συσκευών δηλαδή το ζητούμενο makespan.

Ποιά όμως είναι η καλύτερη από αυτές τις 50 λύσεις; Ο αλγόριθμος για να απαντήσει σε αυτήν την ερώτηση, συγκρίνει τις τιμές της αντικειμενικής συνάρτησης που υπολογίστηκαν από την συνάρτηση *computeNektar()* και μέσω μιας άλλης συνάρτησης της *findBestSolution()* βρίσκει την μικρότερη τιμή, δηλαδή το βέλτιστο makespan των 50 αρχικών λύσεων.

Στη συνέχεια, περνάμε στην κύρια φάση, όπου οι εργάτριες μέλισσες αναζητούν γύρω από τις 50 αρχικές λύσεις, καλύτερες πηγές τροφής. Αυτό επιτυγχάνεται μέσω της συνάρτησης *localSearch1()* κατά την οποία επιλέγονται μία προς μία οι 50 αρχικές λύσεις από κάθε εργάτρια. Σε κάθε μια λύση, η εργάτρια μέλισσα εφαρμόζει κάποιες από τις διακριτές μεθόδους αλλαγής λύσης που αναφέραμε στο δεύτερο κεφάλαιο. Συγκεκριμένα, χρησιμοποιείται η μέθοδος αλλαγής λύσης *swap()*, η οποία εδώ αντιμεταθέτει δύο λύσεις αλλά με σταθερή απόσταση i μεταξύ τους. Αυτή η απόσταση i έχει οριστεί να ισούται με τον αριθμό των συσκευών διαιρεμένο διά δύο (π.χ. αν jobs=20, i=10). Επομένως, η μέθοδος *swap()* αντιμεταθέτει δύο συσκευές που έχουν απόσταση 10 μεταξύ τους, από μιά συγκεκριμένη αλληλουχία συσκευών. Έπειτα, η λύση υφίσταται μία επιπλέον αλλαγή από τη μέθοδο *swapBycost()*. Η μέθοδος αυτή δεν αναφέρθηκε στο δεύτερο κεφάλαιο γιατί δεν αποτελεί μία γενική μέθοδο αλλαγής λύσεων, αλλά μία μέθοδο που κατασκευάστηκε εδώ για να αλλάζει τη λύση εκμεταλλευόμενη τη γεωμετρία του συγκεκριμένου προβλήματος.

Η μέθοδος *swapBycost()* παίρνει ένα τυχαίο κομμάτι μιας πιθανής λύσης και ανακατατάσσει τις συσκευές ανάλογα με τον χρόνο επεξεργασίας τους από τους εργαζομένους. Δηλαδή, συγκρίνει το άθροισμα του χρόνου επεξεργασίας π.χ. της πρώτης και της δεύτερης συσκευής με το άθροισμα της πρώτης και της τρίτης. Αν το πρώτο άθροισμα είναι μεγαλύτερο από το δεύτερο, τότε η συνάρτηση αντιμεταθέτει την δεύτερη και την τρίτη συσκευή. Αυτό, φυσικά, δεν εφαρμόζεται σε όλο το εύρος της λύσης γιατί τότε

όλες οι λύσεις που θα παράγονταν από τη συνάρτηση *swapBycost()* θα ήταν πανομοιότυπες. Αντίθετα, εφαρμόζεται σε ένα κομμάτι της λύσης ώστε μαζί με την μέθοδο *swap()* να εντοπίσουν μια καλύτερη λύση. Στην εικόνα 4.2.1 απεικονίζεται η μέθοδος *swapBycost(),* όπου στην πρώτη σειρά των πινάκων φαίνεται η σειρά με την οποία θα επεξεργαστούν οι συσκευές και στην δεύτερη σειρά οι χρόνοι επεξεργασίας αυτών των συσκευών. Βλέπουμε ότι το άθροισμα των χρόνων επεξεργασίας των συσκευών "1" και "2" είναι μεγαλύτερο από το άθροισμα των χρόνων επεξεργασίας των συσκευών "1" και "3". Επομένως, οι συσκευές αντιμετατίθενται, όπως φαίνεται στον τελικό πίνακα.



Εικόνα 4.2.1: *Παράδειγμα μεθόδου swapBycost(), όπου 10+8 > 10+5 άρα αντιμετατίθενται οι συσκευές 2 και 3.*

Ύστερα από την επέμβαση των μεθόδων *swap()* και swapBycost() πάνω στη λύση, η *localSearch1()* υπολογίζει την τιμή της αντικειμενικής συνάρτησης της καινούριας πλέον λύσης. Αν η καινούρια λύση είναι καλύτερη από την προηγούμενη - και στην περίπτωση μας, αν είναι μικρότερη- τότε αυτή αντικαθιστά την παλιά λύση. Αλλιώς, ξαναγίνεται μια προσπάθεια βελτίωσης της λύσης μέσω της συνάρτησης *swapBycost()* και αν πάλι δεν βρεθεί μια καλύτερη λύση, τότε ο αλγόριθμος κρατάει την παλιά λύση και απορρίπτει την καινούρια.

Συνεχίζουμε με την φάση του χορού των μελισσών. Σε αυτό τη σημείο, από την τιμή της αντικειμενικής συνάρτησης κάθε πιθανής λύση και μέσω της συνάρτησης *computePossibilityOfchosenSolution()* υπολογίζεται ένα ποσοστό με βάση το οποίο οι μέλισσες θεατές θα ακολουθήσουν την πιθανή λύση. Δηλαδή, η προαναφερθείσα συνάρτηση παίρνει την τιμή της αντικειμενικής συνάρτησης μιας πιθανής λύσης και μέσω των τύπων (11) και (12) παράγει έναν αριθμό, ο οποίος όσο μεγαλύτερος είναι, τόσο περισσότερες προσπάθειες θα εφαρμοστούν στην πιθανή λύση για τη βελτίωσή της. Αυτό σημαίνει ότι τόσες περισσότερες επαναλήψεις (μέλισσες θεατές) θα υποστεί η συγκεκριμένη λύση και επομένως θα έχει περισσότερες πιθανότητες να βελτιωθεί.

Το ποσοστό p, που αναφέραμε στην προηγούμενη παράγραφο, χρησιμοποιείται έπειτα, από τη συνάρτηση *localSearch2()* , ώστε να υπολογιστεί ο αριθμός των επαναλήψεων που θα εφαρμοστεί σε κάθε λύση. Έτσι, με έναν προκαθορισμένο αριθμό από μέλισσες-θεατές με onlookers=1000, υπολογίζεται ο αριθμός των επαναλήψεων ως εξής: iterations =

onlookers*p. Επομένως, σε κάθε λύση εφαρμόζεται διαφορετικός αριθμός επαναλήψεων αφού διαφορετικός είναι και ο αριθμός p. Σε αυτήν την τοπική αναζήτηση, κάθε λύση, για αριθμό επαναλήψεων iterations, μετασχηματίζεται μέσω των μεθόδων *multiswap()* και *swapBycost().* Η μέθοδος *multiswap()* είναι ίδια με τη μέθοδο inverse που αναφέραμε στο δεύτερο κεφάλαιο και αντιμεταθέτει όλες τις συσκευές (ή εργασίες) μιας πιθανής λύσης. Βλέπουμε δηλαδή ότι η *multiswap()* αποτελεί μία μέθοδο αρκετά δραστική όσον αφορά την αλλαγή της λύσης. Τέλος, ισχύει και σε αυτήν την localSearch, ότι αν κάποια λύση είναι καλύτερη από την προηγούμενη, τότε η καινούρια λύση αντικαθιστά την προηγούμενη.

Στη συνέχεια, περνάμε στην τελευταία φάση του αλγορίθμου, αυτήν των ανιχνευτριών μελισσών. Σε αυτό το σημείο, υπολογίζεται κατά πόσο η *localSearch2()* βελτίωσε ή όχι το νέκταρ (τιμή της αντικειμενικής συνάρτησης) σε σχέση με τη *localSearch1()*. Η βελτίωση αυτή αποθηκεύεται στη μεταβλητή *delta* και αν είναι μικρότερη από μία προκαθορισμένη τιμή, δηλαδή αν η λύση δεν έχει βελτιωθεί αρκετά, τότε χρησιμοποιούνται οι μέθοδοι *swap()* και *insert()* για να πραγματοποιήσουν μια δραστική αλλαγή στη λύση. Βέβαια, υπάρχει και μία ποινή (*poinh_count)* ώστε ο αλγόριθμος να μην προσπαθεί να βελτιώσει μια λύση που δεν μπορεί να βελτιωθεί περαιτέρω.

Τέλος, από τις 50 βελτιωμένες λύσεις του προβλήματος επιλέγεται αυτή με την μικρότερη τιμή ως το ολικό βέλτιστο. Όλη η διαδικασία που αναφέρθηκε, ακολουθείται για άλλες 4 φορές, ώστε να υπάρξουν ακόμη καλύτερα αποτελέσματα αλλά ταυτόχρονα ο χρόνος εκτέλεσης να παραμείνει μικρός.

Ο κώδικας που χρησιμοποιήθηκε για την υλοποίηση του διακριτού αλγορίθμου Artificial Bee Colony δίνεται σε μορφή ψευδοκώδικα παρακάτω:

ΕΝΑΡΞΗ_ΑΛΓΟΡΙΘΜΟΥ ABC

Προσδιορισμός αριθμού πηγών τροφής
Προσδιορισμός αριθμού επαναλήψεων όλου του ABC
Προσδιορισμός βελτίωσης δέλτα
Προσδιορισμός ποινής
Προσδιορισμός αριθμού θεατών

Ανάγνωση αρχείου txt και προσδιορισμός αριθμού συσκευών, εργαζομένων και χρόνων επεξεργασίας
Δημιουργία αρχικού πληθυσμού λύσεων-πηγών τροφής με συνάρτηση παραγωγής τυχαίων λύσεων
Υπολογισμός της τιμής της αντικειμενικής συνάρτησης για κάθε παραγόμενη λύση (= nektar)
Υπολογισμός καλύτερης λύσης από αυτές του αρχικού πληθυσμού

ΓΙΑ ορισμένο Αριθμό Επαναλήψεων του ABC **ΕΠΑΝΕΛΑΒΕ**

Τοπική Αναζήτηση 1:
Επέλεξε μία πηγή τροφής από τον αρχικό πληθυσμό
ΓΙΑ ορισμένο αριθμό επαναλήψεων **ΕΠΑΝΕΛΑΒΕ**
Άλλαξε την πηγή τροφής (λύση) με τη μέθοδο *swap()*
Άλλαξε την πηγή τροφής (λύση) με τη μέθοδο *swapBycost()*
Υπολόγισε το nektar (τιμή της αντικειμενικής συνάρτησης για αυτήν τη λύση)
ΑΝ καινούρια πηγή τροφής < παλιά πηγή τροφής **ΤΟΤΕ**
πηγή τροφής = καινούρια πηγή τροφής
nektar = Καινούριο nektar
ΑΛΛΙΩΣ
Άλλαξε την πηγή τροφής με τη μέθοδο *swapBycost()*
ΑΝ καινούρια πηγή τροφής < παλιά πηγή τροφής **ΤΟΤΕ**
πηγή τροφής = καινούρια πηγή τροφής
nektar = Καινούριο nektar
ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ_ΑΝ

ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
Τέλος_Τοπικής_Αναζήτησης_1

ΓΙΑ κάθε πηγή τροφής **ΕΠΑΝΕΛΑΒΕ**
Αποθήκευσε στον πίνακα delta το nektar (τιμής της αντικειμενικής συνάρτησης της πηγής τροφής)
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

ΓΙΑ κάθε πηγή τροφής **ΕΠΑΝΕΛΑΒΕ**
Άθροισε τις τιμές των nektar (αντικειμ. συναρτήσεων όλων των πηγών τροφής)
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

ΓΙΑ κάθε πηγή τροφής **ΕΠΑΝΕΛΑΒΕ**
Υπολόγισε το ποσοστό επιλογής κάθε πηγής τροφής με τον τύπο:
p = (1 / τιμή αντικ. συνάρτησης) / (1 / άθροισμα τιμών αντικειμ. συν. όλων των πηγών τροφής)
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

Τοπική Αναζήτηση 2:
ΓΙΑ κάθε πηγή τροφής **ΕΠΑΝΕΛΑΒΕ**
iterations = onlookers * p
Επέλεξε μία πηγή τροφής από τον πληθυσμό
ΓΙΑ ορισμένο αριθμό επαναλήψεων μέχρι *iterations* **ΕΠΑΝΕΛΑΒΕ**
Άλλαξε την πηγή τροφής με τη μέθοδο *multiswap()*
Άλλαξε την πηγή τροφής με τη μέθοδο *swapBycost()*
Υπολόγισε το nektar (τιμή της αντικειμενικής συνάρτησης για αυτήν τη λύση)

```
                    ΑΝ καινούρια πηγή τροφής < παλιά πηγή τροφής ΤΟΤΕ
                        πηγή τροφής = καινούρια πηγή τροφής
                        nektar = Καινούριο nektar
                    ΤΕΛΟΣ_ΑΝ
                ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
            ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
            Τέλος_Τοπικής_Αναζήτησης_2

            ΓΙΑ κάθε πηγή τροφής ΕΠΑΝΕΛΑΒΕ
                Επέλεξε μία πηγή τροφής από τον πληθυσμό
                ΑΝ delta > nektar ΤΟΤΕ
                    delta = delta - nektar
                ΤΕΛΟΣ_ΑΝ
                ΟΣΟ delta < μία προκαθορισμένη βελτίωση ΕΠΑΝΕΛΑΒΕ
                    ΑΝ  μετρητής < ποινή ΤΟΤΕ
                        Άλλαξε την πηγή τροφής με τη μέθοδο swap()
                        Άλλαξε την πηγή τροφής με τη μέθοδο insert()
                        ΑΝ η λύση βελτιώθηκε περαιτέρω ΤΟΤΕ
                            Βάλε στο delta την καινούρια βελτίωση
                            πηγή τροφής = καινούρια πηγή τροφής
                            nektar = Καινούριο nektar
                        ΤΕΛΟΣ_ΑΝ
                        Αύξησε τον μετρητή
                ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
            ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

            Βρες την μικρότερη τιμή nektar από αυτές του πληθυσμού
            Αποθήκευσέ την στη μεταβλητή: totalbest

ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

ΤΕΛΟΣ_ΑΛΓΟΡΙΘΜΟΥ
```

- Για το στοχαστικό σύστημα:

Δημιουργήσαμε ένα ξεχωριστό πρόγραμμα, το οποίο θα δέχεται ως είσοδο το αρχείο txt με τους αρχικούς χρόνους επεξεργασίας και θα δημιουργεί αρχεία εξόδου txt. Κάθε ένα από τα αρχεία εξόδου θα περιέχει ό,τι και το αρχείο εισόδου με τη διαφορά ότι κάθε φορά θα αλλάζει ένας χρόνος επεξεργασίας κατά +1 ή κατά -1. Τα δημιουργούμενα αρχεία εξόδου, στη συνέχεια, θα προωθούνται στο πρόγραμμα ABC, που αναφέραμε παραπάνω, για να εξαχθεί η βέλτιστη τιμή για καθένα από αυτά.

Το πρόγραμμα, αρχικά, ανοίγει το αρχείο εισόδου και αποθηκεύει στις μεταβλητές jobs και machines τις δύο πρώτες τιμές του αρχείου. Στη συνέχεια, δημιουργεί ένα

μονοδιάστατο διάνυσμα costs, με διαστάσεις *jobs x machines* και εισάγει σε αυτό τους χρόνους επεξεργασίας από το αρχείο εισόδου.

Στη συνέχεια, ανάλογα με τον αριθμό των συσκευών που θα είναι τυχαίες εκτελείται από το πρόγραμμα μία από τις ακόλουθες συναρτήσεις:

- Όταν θέλουμε ***όλοι*** οι χρόνοι επεξεργασίας των συσκευών να είναι τυχαίοι:

  Τότε, χρησιμοποιούμε την ιδιότητα της αναδρομής ώστε να προσπελαστούν όλες οι τιμές του διανύσματος costs μέχρι την τελευταία. Όταν φτάσουμε στην τελευταία τιμή, προσθέτουμε +1 και εξάγουμε το διάνυσμα σε αρχείο txt με τη μορφή του αρχείου εισόδου. Στην συνέχεια αφαιρούμε -2 από την τελευταία τιμή του διανύσματος και ξαναεξάγουμε το διάνυσμα. Τέλος, προσθέτουμε +1 για να φέρουμε την τελευταία τιμή του διανύσματος στην αρχική της τιμή και συνεχίζει η προσπέλαση προς την αντίθετη φορά μέχρι να φτάσουμε στο πρώτο στοιχείο του διανύσματος. Έτσι, έχουμε προσπελάσει τα στοιχεία του διανύσματος ένα προς ένα και έχουμε προσθαφαιρέσει την τιμή 1 προκειμένου να δημιουργηθούν καινούρια αρχεία με καινούριους χρόνους επεξεργασίας

- Όταν μία συσκευή έχει βλάβη ή ο χρόνος επεξεργασίας της είναι τυχαίος (θα δημιουργηθούν πέντε παραδείγματα για αυτήν την περίπτωση):

  Τότε χρησιμοποιούμε πάλι την ιδιότητα της αναδρομής με τη διαφορά ότι τώρα τα στοιχεία δεν προσπελαύνονται ένα προς ένα. Εδώ, δίνοντας στη συνάρτηση της αναδρομής έναν αριθμό εκκίνησης (πρόκειται για την εργασία που θέλουμε να είναι τυχαία), προσπελαύνουμε κάθε φορά τους χρόνους επεξεργασίας της συγκεκριμένης εργασίας. Δηλαδή, αν θεωρήσουμε τον πίνακα των χρόνων επεξεργασίας 20 συσκευών - 5 εργαζομένων ως ένα μονοδιάστατο διάνυσμα 1 x 100 (και όχι ως έναν διδιάστατο πίνακα 20 x 5) και θεωρούμε τυχαία π.χ. την 4η εργασία τότε η προσπέλαση έχει ως εξής: Από το 4ο στοιχείο του διανύσματος (εργασία εκκίνησης) -μέσω της αναδρομής- πηγαίνουμε στο 24ο (= 4 + 20) και ύστερα από το 24ο πηγαίνουμε στο 44ο και ούτω καθεξής μέχρι να φτάσουμε στο 84ο. Εκτυπώνεται το αποτέλεσμα, προσθέτουμε +1 στο 84ο και ξαναεκτυπώνουμε το αποτέλεσμα. Ύστερα, αφαιρούμε -2 (ώστε από το αρχικό αποτέλεσμα ουσιαστικά να έχει αφαιρεθεί -1) και ξαναεκτυπώνουμε τον πίνακα. Τέλος, προσθέτουμε +1 για να επαναφέρουμε το 84ο στοιχείο του διανύσματος στην αρχική του τιμή. Στη συνέχεια, ακολουθώντας την αναδρομή προς τα πίσω, πηγαίνουμε στο 64ο στοιχείο και ακολουθείται η ίδια διαδικασία που αναφέρθηκε παραπάνω και έτσι δημιουργούνται όλοι οι δυνατοί συνδυασμοί των πέντε χρόνων επεξεργασίας της μιας συσκευής, παίρνοντας ο καθένας 3 διαφορετικές τιμές (t, t+1, t-1). Επομένως, με τους 5 χρόνους

επεξεργασίας μιας συσκευής δημιουργούνται 3^5 = 243 διαφορετικά αποτελέσματα.

Παρακάτω δίνεται η υλοποίηση του αλγορίθμου για τους τυχαίους χρόνους επεξεργασίας των εργασιών. Ο αλγόριθμος περιλαμβάνει 2 συναρτήσεις που χρησιμοποιούνται αναλόγως του αριθμού των εργασιών που επιθυμούμε να είναι τυχαίες. Η πρώτη συνάρτηση δημιουργεί αρχεία εξόδου, δεδομένου ότι όλες οι εργασίες είναι τυχαίες, ενώ η δεύτερη θεωρεί ότι μόνο μία εργασία είναι τυχαία. Ο αλγόριθμος υλοποιείται σε μορφή ψευδοκώδικα:

ΕΝΑΡΞΗ_ΑΛΓΟΡΙΘΜΟΥ_ΤΥΧΑΙΩΝ_ΧΡΟΝΩΝ_ΕΠΕΞΕΡΓΑΣΙΑΣ

Αποθήκευσε τις εργασίες στη μεταβλητή jobs από το αρχείο εισόδου
Αποθήκευσε τις μηχανές στη μεταβλητή machines από το αρχείο εισόδου
Δημιούργησε ένα vector *costs*
Βάλε στο vector *costs* τους χρόνους επεξεργασίας από το αρχείο εισόδου

// *Επιλογή 1 - Αναδρομή με όλες τις εργασίες τυχαίες*
Βάλε στη μεταβλητή start_job την αρχική εργασία (start_job = 0)
Συνάρτηση_Αναδρομής_1 (costs , start_job)
//Τέλος Επιλογής 1
// *Επιλογή 2 - Αναδρομή με μία εργασία τυχαία*
Βάλε στη μεταβλητή start_job την εργασία που θέλω να είναι τυχαία (π.χ. start_job = 3)
Συνάρτηση_Αναδρομής_2 (costs , start_job)
//Τέλος Επιλογής 2

ΤΕΛΟΣ_ΑΛΓΟΡΙΘΜΟΥ_ ΤΥΧΑΙΩΝ_ΧΡΟΝΩΝ_ΕΠΕΞΕΡΓΑΣΙΑΣ

<u>ΥΛΟΠΟΙΗΣΗ ΣΥΝΑΡΤΗΣΕΩΝ</u>

Συνάρτηση_Αναδρομής_1 (costs , start_job)

ΑΝ start_job + 1 < μέγεθος του costs **ΤΟΤΕ**
    Συνάρτηση_Αναδρομής_1 (costs , start_job + 1)
ΑΛΛΙΩΣ
    Εκτύπωσε τον πίνακα costs στη μορφή των αρχείων εισόδου
ΤΕΛΟΣ_ΑΝ

Αύξησε τον πίνακα costs στη θέση start_job κατά 1

ΑΝ start_job + 1 < μέγεθος του costs **ΤΟΤΕ**
    Συνάρτηση_Αναδρομής_1 (costs , start_job + 1)

ΑΛΛΙΩΣ
    Εκτύπωσε τον πίνακα costs στη μορφή των αρχείων εισόδου
ΤΕΛΟΣ_ΑΝ

Μείωσε τον πίνακα costs στη θέση start_job κατά 2

ΑΝ start_job + 1 < μέγεθος του costs **ΤΟΤΕ**
    Συνάρτηση_Αναδρομής_1 (costs , start_job + 1)
ΑΛΛΙΩΣ
    Εκτύπωσε τον πίνακα costs στη μορφή των αρχείων εισόδου
ΤΕΛΟΣ_ΑΝ

Αύξησε τον πίνακα costs στη θέση start_job κατά 1
Τέλος_Συνάρτησης_Αναδρομής_1

Συνάρτηση_Αναδρομής_2 (costs , start_job)

Όρισε μεταβλητή stop_job τέτοια ώστε stop_job = start_job + ((machines-1) * jobs)

ΑΝ start_job < stop_job **ΤΟΤΕ**
    Συνάρτηση_Αναδρομής_2 (costs , start_job + jobs)
ΑΛΛΙΩΣ
    Εκτύπωσε τον πίνακα costs στη μορφή των αρχείων εισόδου
ΤΕΛΟΣ_ΑΝ

Αύξησε τον πίνακα costs στη θέση start_job κατά 1

ΑΝ start_job < stop_job **ΤΟΤΕ**
    Συνάρτηση_Αναδρομής_2 (costs , start_job + jobs)
ΑΛΛΙΩΣ
    Εκτύπωσε τον πίνακα costs στη μορφή των αρχείων εισόδου
ΤΕΛΟΣ_ΑΝ

Μείωσε τον πίνακα costs στη θέση start_job κατά 2

ΑΝ start_job < stop_job **ΤΟΤΕ**
    Συνάρτηση_Αναδρομής_2 (costs , start_job + jobs)
ΑΛΛΙΩΣ
    Εκτύπωσε τον πίνακα costs στη μορφή των αρχείων εισόδου
ΤΕΛΟΣ_ΑΝ

Αύξησε τον πίνακα costs στη θέση start_job κατά 1
Τέλος_Συνάρτησης_Αναδρομής_2

ΚΕΦΑΛΑΙΟ 5
ΥΠΟΛΟΓΙΣΤΙΚΑ ΑΠΟΤΕΛΕΣΜΑΤΑ

5.1 Αποτελέσματα επίλυσης προβλημάτων βιβλιογραφίας με τον αλγόριθμο ABC

Στο κεφάλαιο αυτό, προκειμένου να αξιολογήσουμε την αποτελεσματικότητα του αλγορίθμου Artificial Bee Colony ώστε να τον εφαρμόσουμε στο πραγματικό σύστημα, θα παραθέσουμε αρχικά, τα αποτελέσματα από την εφαρμογή του αλγορίθμου στα 120 θεωρητικά προβλήματα (benchmark instances) του Taillard (1993) με αντικειμενό στόχο πάντα την ελαχιστοποίηση του συνολικού χρόνου επεξεργασίας των εργασιών (makespan). Σε αυτά τα προβλήματα, διακρίνουμε διαφορετικές ομάδες των 20, 50, 100, 200 και 500 εργασιών, τα οποία μπορεί να έχουν 5, 10 ή 20 μηχανές. Κάθε ομάδα προβλημάτων περιλαμβάνει 10 παραδείγματα. Συνολικά, υπάρχουν 12 ομάδες προβλημάτων οι οποίες είναι: 20 x 5 (δηλ. 20 εργασίες και 5 μηχανές), 20 x 10, 20 x 20, 50 x 5, 50 x 10, 50 x 20, 100 x 5, 100 x 10, 100 x 20, 200 x 10, 200 x 20 και 500 x 20.

Η υλοποίηση του αλγορίθμου Artificial Bee Colony περιλαμβάνει τον καθορισμό κάποιων παραμέτρων, οι οποίες επιλέχτηκαν ύστερα από έναν μεγάλο αριθμό δοκιμών ώστε να διαπιστωθεί ότι οι συγκεκριμένες δίνουν το καλύτερο αποτέλεσμα στον μικρότερο δυνατό χρόνο. Αυτές οι παράμετροι είναι οι εξής:

- Ο αριθμός των αρχικών πηγών τροφής food_sources = 50.
- Ο αριθμός των συνολικών επαναλήψεων του αλγορίθμου r = 5.
- Το μέγεθος της βελτίωσης delta = 20.
- Η ποινή count = 100 πάνω από την οποία εγκαταλείπεται κάθε προσπάθεια περαιτέρω βελτίωσης της λύσης
- Η παράμετρος της συνάρτησης Swap(), i = 10 δηλαδή η απόσταση που έχουν δύο εργασίες που αντιμετατίθενται.
- Ο αριθμός επαναλήψεων στην Τοπική Αναζήτηση 1, k = 10000
- Ο αριθμός των θεατών, onlookers = 1000.

Για κάθε θεωρητικό παράδειγμα, ο αλγόριθμος εκτελείται τρεις φορές προκειμένου να εξαχθεί η μικρότερη λύση. Επίσης θα παραθέσουμε και την σχετική απόκλιση αυτής της μικρότερης λύσης από την γνωστή -έως τώρα- καλύτερη λύση. Δηλαδή η σχετική απόκλιση ορίζεται ως εξής: deviation = $\frac{C_{ABC} - C_{BEST}}{C_{BEST}} \cdot \%$, όπου C_{ABC} είναι η μικρότερη λύση που έδωσε ο Artificial Bee Colony και C_{BEST} είναι η γνωστή -έως τώρα- καλύτερη λύση.

Στον παρακάτω πίνακα παρουσιάζονται αναλυτικά τα αποτελέσματα του αλγορίθμου Artificial Bee Colony για κάθε ένα από τα θεωρητικά παραδείγματα καθώς και η σχετική απόκλισή τους από την γνωστή -έως τώρα- καλύτερη λύση.

Πίνακας 5.1.α: Αποτελέσματα του ABC για τα παραδείγματα του Taillard για το PFSSP

Problem	BKS	Run 1	Run 2	Run 3	Min	Deviation
20x5	1278	1290	1288	1278	1278	0.00%
20x5	1359	1366	1365	1366	1365	0.44%
20x5	1081	1114	1125	1114	1114	3.05%
20x5	1293	1347	1340	1351	1340	3.63%
20x5	1235	1279	1260	1277	1260	2.02%
20x5	1195	1225	1224	1225	1224	2.43%
20x5	1239	1252	1259	1255	1252	1.05%
20x5	1206	1250	1242	1239	1239	2.74%
20x5	1230	1261	1266	1278	1261	2.52%
20x5	1108	1140	1140	1145	1140	2.89%
20x10	1582	1684	1682	1667	1667	5.37%
20x10	1659	1771	1752	1760	1752	5.61%
20x10	1496	1590	1585	1577	1577	5.41%
20x10	1377	1438	1439	1446	1438	4.43%
20x10	1419	1523	1521	1510	1510	6.41%
20x10	1397	1471	1483	1491	1471	5.30%
20x10	1484	1542	1543	1544	1542	3.91%
20x10	1538	1639	1628	1628	1628	5.85%
20x10	1593	1668	1659	1664	1659	4.14%
20x10	1591	1672	1684	1661	1661	4.40%
20x20	2297	2421	2414	2392	2392	4.14%
20x20	2099	2218	2188	2213	2188	4.24%
20x20	2326	2453	2441	2450	2441	4.94%
20x20	2223	2330	2323	2339	2323	4.50%
20x20	2291	2421	2417	2412	2412	5.28%
20x20	2226	2319	2324	2333	2319	4.18%
20x20	2273	2391	2350	2372	2350	3.39%

20x20	2200	2314	2308	2301	2301	4.59%
20x20	2237	2354	2381	2368	2354	5.23%
20x20	2178	2314	2344	2308	2308	5.97%
50x5	2724	2744	2752	2752	2744	0.73%
50x5	2834	2912	2943	2944	2912	2.75%
50x5	2621	2666	2680	2673	2666	1.72%
50x5	2751	2802	2832	2797	2797	1.67%
50x5	2863	2894	2882	2883	2882	0.66%
50x5	2829	2870	2878	2865	2865	1.27%
50x5	2725	2775	2771	2777	2771	1.69%
50x5	2683	2723	2718	2711	2711	1.04%
50x5	2552	2638	2639	2614	2614	2.43%
50x5	2782	2786	2789	2792	2786	0.14%
50x10	2991	3339	3256	3306	3256	8.86%
50x10	2867	3169	3132	3109	3109	8.44%
50x10	2839	3163	3176	3247	3163	11.41%
50x10	3063	3277	3303	3281	3277	6.99%
50x10	2976	3260	3252	3277	3252	9.27%
50x10	3006	3259	3222	3220	3220	7.12%
50x10	3093	3275	3271	3310	3271	5.75%
50x10	3037	3264	3269	3258	3258	7.28%
50x10	2897	3120	3137	3198	3120	7.70%
50x10	3065	3318	3329	3311	3311	8.03%
50x20	3850	4305	4232	4244	4232	9.92%
50x20	3704	4174	4165	4105	4105	10.83%
50x20	3640	4117	4099	4061	4061	11.57%
50x20	3720	4097	4115	4172	4097	10.13%
50x20	3610	4093	4111	4078	4078	12.96%
50x20	3681	4086	3998	4050	3998	8.61%
50x20	3704	4096	4206	4155	4096	10.58%
50x20	3691	4178	4155	4204	4155	12.57%
50x20	3743	4284	4188	4157	4157	11.06%
50x20	3756	4315	4171	4139	4139	10.20%

100x5	5493	5563	5539	5574	5539	0.84%
100x5	5268	5360	5358	5316	5316	0.91%
100x5	5175	5269	5271	5257	5257	1.58%
100x5	5014	5035	5131	5077	5035	0.42%
100x5	5250	5375	5312	5349	5312	1.18%
100x5	5135	5188	5192	5193	5188	1.03%
100x5	5246	5305	5314	5349	5305	1.12%
100x5	5094	5239	5160	5176	5160	1.30%
100x5	5448	5513	5509	5666	5509	1.12%
100x5	5322	5422	5451	5451	5422	1.88%
100x10	5770	6068	6122	6110	6068	5.16%
100x10	5349	5715	5697	5688	5688	6.34%
100x10	5676	6007	5902	6017	5902	3.98%
100x10	5781	6249	6237	6175	6175	6.82%
100x10	5467	5946	5967	5967	5946	8.76%
100x10	5303	5763	5653	5659	5653	6.60%
100x10	5595	5860	5853	5863	5853	4.61%
100x10	5617	5935	6029	5992	5935	5.66%
100x10	5871	6137	6144	6096	6096	3.83%
100x10	5845	6162	6209	6168	6162	5.42%
100x20	6202	7077	7071	6999	6999	12.85%
100x20	6183	6923	6972	6964	6923	11.97%
100x20	6271	7050	6959	7079	6959	10.97%
100x20	6269	7096	7123	7059	7059	12.60%
100x20	6314	6947	7135	7090	6947	10.03%
100x20	6364	7140	7124	7124	7124	11.94%
100x20	6268	7177	7081	7312	7081	12.97%
100x20	6401	7161	7212	7252	7161	11.87%
100x20	6275	7189	7158	7146	7146	13.88%
100x20	6434	7167	7134	7146	7134	10.88%
200x10	10862	11374	11366	11410	11366	4.64%
200x10	10480	11325	11506	11403	11325	8.06%
200x10	10922	11311	11360	11530	11311	3.56%

200x10	10889	11239	11230	11341	11230	3.13%
200x10	10524	11228	11279	11211	11211	6.53%
200x10	10329	11117	11034	11154	11034	6.83%
200x10	10854	11779	11794	11626	11626	7.11%
200x10	10730	11301	11328	11371	11301	5.32%
200x10	10438	11228	11380	11212	11212	7.42%
200x10	10675	11410	11241	11350	11241	5.30%
200x20	11195	12591	12670	12739	12591	12.47%
200x20	11203	12821	12825	12820	12820	14.43%
200x20	11281	12613	12798	12684	12613	11.81%
200x20	11275	12564	12674	12542	12542	11.24%
200x20	11259	12555	12620	12581	12555	11.51%
200x20	11176	12482	12500	12539	12482	11.69%
200x20	11360	12642	12545	12646	12545	10.43%
200x20	11334	12813	12760	12786	12760	12.58%
200x20	11192	12639	12647	12627	12627	12.82%
200x20	11288	12860	12837	12748	12748	12.93%
500x20	26059	28605	28807	29153	28605	9.77%
500x20	26520	29419	29488	29221	29221	10.18%
500x20	26371	29004	29154	29118	29004	9.98%
500x20	26456	29139	29109	29085	29085	9.94%
500x20	26334	29083	29066	29027	29027	10.23%
500x20	26477	29289	29263	29322	29263	10.52%
500x20	26389	29029	29101	29074	29029	10.00%
500x20	26560	29302	29330	29293	29293	10.29%
500x20	26005	28899	28867	28940	28867	11.01%
500x20	26457	29085	29039	29109	29039	9.76%

Στον Πίνακα 5.1.α, η πρώτη στήλη περιέχει τον τύπο του προβλήματος, δηλαδή ότι έχουμε για παράδειγμα ένα μοντέλο με 20 εργασίες και 5 μηχανές. Η δεύτερη στήλη αποτελεί το καλύτερο αποτέλεσμα που έχει βρεθεί μέχρι στιγμής από τη βιβλιογραφία (BKS) και οι τρεις επόμενες είναι τα αποτελέσματα που έδωσε ο συγκεκριμένος ABC για το πρόβλημα PFSSP. Τέλος, στην τελευταία στήλη δίνεται η σχετική απόκλιση της μικρότερης τιμής των τριών λύσεων του ABC (Run1, Run2, Run3) από την BKS.

Παρατηρούμε ότι σε όλα τα παραδείγματα η απόκλιση της λύσης δεν ξεπερνά το 14% καθώς επίσης και ότι στα προβλήματα με μικρό αριθμό μηχανών, οι παραγόμενες λύσεις είναι πολύ καλύτερες ακόμα και για μεγάλο αριθμό εργασιών. Επομένως, ο αλγόριθμος είναι αρκετά αποτελεσματικός προκειμένου να χρησιμοποιηθεί και στο πραγματικό σύστημα παραγωγής, τόσο με αιτιοκρατικούς χρόνους επεξεργασίας όσο και με στοχαστικούς.

5.2 Αποτελέσματα του αλγορίθμου ABC στο πραγματικό σύστημα με αιτιοκρατικούς χρόνους επεξεργασίας.

Στο προηγούμενο κεφάλαιο πραγματοποιήθηκε η μοντελοποίηση της παραγωγικής διαδικασίας του εργαστηρίου διακρίβωσης σε ένα πρόβλημα αντιμετάθεσης χρονοπρογραμματισμού εργασιών (PFSSP). Στο παρόν κεφάλαιο, θα παρουσιάσουμε τους αιτιοκρατικούς χρόνους επεξεργασίας κάθε συσκευής από κάθε εργαζόμενο, όπως έχουν προκύψει από μελέτη μακρόχρονων στατιστικών στοιχείων. Έτσι, η επίλυση του προβλήματος αντιμετάθεσης χρονοπρογραμματισμού εργασιών περιλαμβάνει τους εξής χρόνους επεξεργασίας οι οποίοι μετριούνται σε λεπτά:

Πίνακας 5.2.a: Χρόνοι επεξεργασίας συσκευών από τους εργαζομένους του εργαστηρίου διακρίβωσης.

Συσκευές:	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Εργαζόμενος1 (καταγραφή)	45	38	51	17	77	63	35	83	72	78	67	19	41	92	21	77	23	78	86	49
Εργαζόμενος2 (ποιοτικός ελ.)	97	3	11	99	65	7	99	6	5	65	30	16	37	57	74	41	12	68	50	77
Εργαζόμενος3 (ανάθεση)	61	98	94	51	98	54	6	32	75	46	70	10	36	14	36	74	62	57	77	4
Εργαζόμενος4 (διακρίβωση)	66	85	13	86	87	19	31	95	94	58	58	90	93	14	65	4	45	77	15	13
Εργαζόμενος5 (παραλαβή)	85	65	2	58	35	53	35	14	96	31	68	27	80	94	74	78	85	81	86	82

Οι παραπάνω χρόνοι εισήχθησαν στον αλγόριθμο ABC προκειμένου να εξαχθεί ο βέλτιστος συνολικός χρόνος επεξεργασίας των συσκευών αλλά και η σειρά με την οποία πρέπει να επεξεργαστούν για να επιτύχουν τον συγκεκριμένο χρόνο. Το πρόγραμμα εκτελέστηκε 50 φορές και τα αποτελέσματά του φαίνονται στον πίνακα 5.2.b

Πίνακας 5.2.b: Makespan και αλληλουχία συσκευών του πραγματικού συστήματος παραγωγής

Makespan	**Job Sequence**
1380	16 12 14 10 15 3 5 8 0 18 6 1 19 4 9 17 2 7 13 11
1380	16 12 19 1 3 5 17 18 9 11 10 6 15 14 8 4 0 7 13 2
1382	16 14 12 19 1 4 15 17 0 8 10 6 11 5 2 3 9 18 13 7
1374	16 14 19 12 1 0 6 10 8 15 17 5 11 4 9 18 3 7 13 2
1382	16 14 12 5 8 3 0 11 13 18 1 19 15 4 17 9 10 6 7 2
1377	16 14 19 11 1 0 9 5 8 3 10 15 6 12 4 17 7 18 13 2
1382	11 16 14 12 17 19 5 1 3 6 10 8 0 4 15 9 7 18 13 2
1380	16 12 14 11 19 1 15 3 10 8 4 5 17 9 0 6 7 18 13 2

1379	16 14 11 19 1 10 0 8 6 9 15 12 3 5 4 17 7 18 13 2
1371	16 14 19 12 1 0 17 18 7 3 6 10 5 8 11 15 4 9 13 2
1380	16 14 5 12 8 0 3 19 11 10 13 1 4 15 17 9 7 2 18 6
1378	16 14 5 3 15 19 11 1 12 10 13 0 8 4 9 17 7 2 18 6
1377	16 14 10 12 0 19 5 11 3 1 6 8 4 15 9 17 7 18 13 2
1382	16 12 19 11 1 0 14 2 3 10 13 8 4 15 9 17 7 5 18 6
1377	16 14 19 11 8 0 5 10 3 1 13 17 12 4 15 9 6 7 18 2
1379	16 12 19 11 10 14 1 18 3 0 13 5 8 15 4 9 17 2 7 6
1379	16 12 19 8 10 3 1 14 13 18 11 2 4 15 17 9 5 0 7 6
1384	16 12 14 11 19 1 10 6 8 3 0 5 4 15 17 9 7 18 13 2
1382	16 14 12 10 6 15 8 0 1 3 13 19 5 11 17 4 9 7 18 2
1379	16 12 19 14 8 1 6 18 17 0 11 3 10 5 9 4 15 7 13 2
1382	16 14 12 10 1 19 3 8 11 6 5 4 15 0 17 9 7 18 13 2
1379	16 12 14 5 0 15 11 3 13 18 4 19 10 8 9 17 1 6 7 2
1378	16 14 10 12 0 19 8 5 3 1 6 11 15 4 17 9 7 18 13 2
1371	16 14 19 12 1 5 6 10 3 8 18 11 0 15 4 9 17 7 13 2
1371	16 14 5 12 10 0 8 3 13 18 4 17 11 15 9 1 19 2 7 6
1372	16 14 19 1 0 5 3 10 8 15 13 11 12 4 9 17 7 2 18 6
1379	16 12 14 10 15 11 19 8 1 3 5 4 0 6 17 9 7 18 13 2
1382	16 14 12 19 1 15 3 0 18 7 13 6 11 5 10 8 4 17 9 2
1382	11 16 14 19 8 3 12 1 0 6 5 10 15 4 17 9 7 18 13 2
1378	16 14 5 3 15 19 11 1 12 10 13 0 8 4 17 9 7 2 18 6
1379	16 12 13 8 1 0 17 2 19 5 3 10 14 4 15 6 9 7 18 11
1373	16 14 19 11 10 8 6 1 13 18 3 15 12 4 9 17 0 5 7 2
1371	16 14 19 11 1 0 17 18 3 5 13 12 2 10 8 4 15 9 7 6
1379	16 14 11 19 1 0 5 10 3 8 15 6 12 4 17 9 7 18 13 2
1382	11 16 14 12 0 19 1 5 10 3 6 8 4 15 17 9 7 18 13 2
1377	16 14 10 12 5 8 3 1 19 4 17 0 6 15 11 9 7 18 13 2
1379	16 12 19 11 1 15 14 10 3 8 5 6 17 4 0 9 7 18 13 2
1372	16 14 19 10 12 1 0 5 6 3 13 8 15 4 17 9 7 2 18 11
1371	16 14 19 12 1 0 10 3 18 5 11 6 8 4 15 9 17 7 13 2
1382	16 14 12 19 10 8 3 18 7 17 6 1 0 5 15 11 4 9 13 2
1379	16 14 11 19 8 10 15 1 3 0 13 12 5 4 17 9 7 2 18 6
1382	11 16 14 12 5 10 17 3 13 18 0 1 19 8 15 4 6 9 7 2
1373	16 14 15 19 1 17 12 0 18 7 13 10 8 3 4 5 6 11 9 2
1371	16 14 5 12 8 3 19 1 0 6 13 11 10 15 17 7 4 2 18 9
1371	16 14 13 11 12 1 15 0 8 3 10 4 19 6 2 9 17 7 18 5
1377	16 11 19 8 14 1 0 5 10 3 13 12 6 15 9 17 7 4 18 2

1379	16 14 5 3 19 10 8 0 13 18 11 1 4 15 9 12 6 17 7 2
1371	16 14 5 11 19 8 3 1 13 18 4 15 10 6 17 9 12 0 7 2
1379	16 12 19 1 0 6 8 17 13 18 11 3 15 4 5 14 10 9 7 2
1382	11 16 14 19 8 3 12 1 0 10 15 5 6 4 17 9 7 18 13 2

Παρατηρούμε ότι ο ελάχιστος συνολικός χρόνος επεξεργασίας των συσκευών (minimum makespan) είναι **1371** λεπτά και η αλληλουχία με την οποία πρέπει να επεξεργαστούν οι συσκευές φαίνεται στη δεύτερη στήλη του πίνακα 5.2.b.

5.3 Αποτελέσματα του αλγορίθμου ABC στο πραγματικό σύστημα με στοχαστικούς χρόνους επεξεργασίας.

Μελετώντας ένα πραγματικό σύστημα, είναι δυνατόν να παρατηρήσουμε μεταβολές στους χρόνους επεξεργασίας των εργασιών από τις μηχανές. Έτσι, για το σύστημα παραγωγής του εργαστηρίου διακρίβωσης θα μελετήσουμε τις εξής περιπτώσεις;

- *Όταν όλες οι συσκευές έχουν τυχαίους χρόνους επεξεργασίας.*

Μέσω του αλγορίθμου τυχαίων χρόνων επεξεργασίας, τον οποίο εφαρμόσαμε στα πραγματικά δεδομένα του πίνακα 5.2.a, παράξαμε αρχεία εξόδου, σε κάθε ένα από τα οποία αλλάζει κάθε φορά κάποιος χρόνος επεξεργασίας μιας συσκευής κατα ±1. Όμως, ο αριθμός των πιθανών συνδυασμών 100 αριθμών (20 x 5), οι οποίοι μάλιστα παίρνουν 3 διαφορετικές τιμές (t, t+1, t-1), είναι 3^{100} = 5,15378E+47. Επειδή, επομένως δεν είναι δυνατό να παραχθούν και κατ' επέκταση να επεξεργαστούν τόσα αρχεία, επιλέξαμε να παράγουμε και να επεξεργαστούμε τα 200 πρώτα από αυτά. Επίσης, υπολογίσαμε το makespan στην περίπτωση όπου όλοι οι χρόνοι επεξεργασίας είναι αυξημένοι κατά +1 και στην περίπτωση όπου όλοι οι χρόνοι επεξεργασίας έχουν μειωθεί κατά -1. Στους παρακάτω πίνακες φαίνονται τα αποτελέσματα που εξάγαμε, μεταβάλλοντας τους χρόνους επεξεργασίας κάθε συσκευής από κάθε εργαζόμενο καθώς επίσης και η απόκλιση του καλύτερου αποτελέσματος από την αιτιοκρατική λύση **1371**.

Πίνακας 5.3.a: Makespan με όλες τις εργασίες τυχαίες (200 συνδυασμοί).

Problem	Run1	Run2	Run3	Min	Deviation
output_001.txt	1378	1382	1379	1378	0.51%
output_002.txt	1383	1380	1383	1380	0.66%
output_003.txt	1378	1376	1370	1370	-0.07%
output_004.txt	1383	1380	1385	1380	0.66%

output_005.txt	1381	1375	1381	1375	0.29%
output_006.txt	1378	1382	1379	1378	0.51%
output_007.txt	1378	1378	1370	1370	-0.07%
output_008.txt	1379	1382	1371	1371	0.00%
output_009.txt	1369	1377	1380	1369	-0.15%
output_011.txt	1384	1380	1381	1380	0.66%
output_012.txt	1376	1375	1377	1375	0.29%
output_013.txt	1381	1386	1381	1381	0.73%
output_014.txt	1374	1374	1385	1374	0.22%
output_015.txt	1385	1377	1379	1377	0.44%
output_016.txt	1378	1371	1379	1371	0.00%
output_017.txt	1383	1380	1379	1379	0.58%
output_018.txt	1370	1381	1378	1370	-0.07%
output_019.txt	1376	1378	1378	1376	0.36%
output_020.txt	1382	1382	1382	1382	0.80%
output_021.txt	1380	1377	1369	1369	-0.15%
output_022.txt	1378	1382	1379	1378	0.51%
output_023.txt	1380	1383	1380	1380	0.66%
output_024.txt	1370	1383	1370	1370	-0.07%
output_025.txt	1380	1383	1377	1377	0.44%
output_026.txt	1370	1379	1378	1370	-0.07%
output_027.txt	1368	1379	1374	1368	-0.22%
output_028.txt	1379	1377	1382	1377	0.44%
output_029.txt	1373	1375	1376	1373	0.15%
output_030.txt	1376	1382	1371	1371	0.00%
output_031.txt	1384	1384	1380	1380	0.66%
output_032.txt	1384	1377	1379	1377	0.44%
output_033.txt	1378	1372	1379	1372	0.07%
output_034.txt	1376	1376	1374	1374	0.22%
output_035.txt	1376	1377	1372	1372	0.07%
output_036.txt	1373	1380	1379	1373	0.15%
output_037.txt	1373	1383	1380	1373	0.15%
output_038.txt	1381	1381	1381	1381	0.73%
output_039.txt	1374	1378	1382	1374	0.22%
output_040.txt	1379	1379	1374	1374	0.22%
output_041.txt	1384	1382	1375	1375	0.29%
output_042.txt	1383	1380	1383	1380	0.66%
output_043.txt	1378	1383	1377	1377	0.44%
output_044.txt	1378	1377	1380	1377	0.44%
output_045.txt	1372	1381	1371	1371	0.00%
output_046.txt	1372	1373	1378	1372	0.07%
output_047.txt	1383	1376	1382	1376	0.36%
output_048.txt	1370	1380	1375	1370	-0.07%
output_049.txt	1378	1379	1372	1372	0.07%
output_050.txt	1385	1382	1380	1380	0.66%
output_051.txt	1378	1379	1378	1378	0.51%
output_052.txt	1375	1381	1370	1370	-0.07%
output_053.txt	1384	1381	1371	1371	0.00%
output_054.txt	1373	1374	1375	1373	0.15%
output_055.txt	1381	1372	1379	1372	0.07%
output_056.txt	1382	1380	1377	1377	0.44%

output_057.txt	1372	1369	1378	1369	-0.15%
output_058.txt	1380	1378	1378	1378	0.51%
output_059.txt	1381	1378	1381	1378	0.51%
output_060.txt	1379	1379	1381	1379	0.58%
output_061.txt	1378	1380	1381	1378	0.51%
output_062.txt	1384	1384	1381	1381	0.73%
output_063.txt	1381	1380	1376	1376	0.36%
output_064.txt	1374	1377	1383	1374	0.22%
output_065.txt	1381	1372	1372	1372	0.07%
output_066.txt	1381	1381	1382	1381	0.73%
output_067.txt	1372	1379	1372	1372	0.07%
output_068.txt	1384	1385	1385	1384	0.95%
output_069.txt	1375	1383	1371	1371	0.00%
output_070.txt	1379	1379	1379	1379	0.58%
output_071.txt	1382	1374	1383	1374	0.22%
output_072.txt	1380	1378	1378	1378	0.51%
output_073.txt	1377	1381	1381	1377	0.44%
output_074.txt	1381	1370	1370	1370	-0.07%
output_075.txt	1368	1379	1368	1368	-0.22%
output_076.txt	1382	1382	1379	1379	0.58%
output_077.txt	1371	1380	1382	1371	0.00%
output_078.txt	1369	1381	1377	1369	-0.15%
output_079.txt	1377	1373	1377	1373	0.15%
output_080.txt	1378	1377	1378	1377	0.44%
output_081.txt	1376	1376	1376	1376	0.36%
output_082.txt	1383	1380	1382	1380	0.66%
output_083.txt	1384	1384	1378	1378	0.51%
output_084.txt	1378	1374	1382	1374	0.22%
output_085.txt	1384	1373	1381	1373	0.15%
output_086.txt	1382	1382	1382	1382	0.80%
output_087.txt	1380	1383	1381	1380	0.66%
output_088.txt	1372	1371	1385	1371	0.00%
output_089.txt	1383	1380	1380	1380	0.66%
output_090.txt	1382	1377	1370	1370	-0.07%
output_091.txt	1381	1381	1380	1380	0.66%
output_092.txt	1381	1382	1382	1381	0.73%
output_093.txt	1372	1372	1379	1372	0.07%
output_094.txt	1384	1374	1382	1374	0.22%
output_095.txt	1385	1375	1385	1375	0.29%
output_096.txt	1384	1373	1381	1373	0.15%
output_097.txt	1383	1383	1383	1383	0.88%
output_098.txt	1381	1381	1383	1381	0.73%
output_099.txt	1373	1383	1382	1373	0.15%
output_10.txt	1383	1378	1380	1378	0.51%
output_100.txt	1371	1380	1379	1371	0.00%
output_101.txt	1380	1381	1380	1380	0.66%
output_102.txt	1381	1379	1380	1379	0.58%
output_103.txt	1380	1383	1383	1380	0.66%
output_104.txt	1378	1381	1382	1378	0.51%
output_105.txt	1382	1379	1384	1379	0.58%
output_106.txt	1379	1378	1375	1375	0.29%

output_107.txt	1382	1372	1377	1372	0.07%
output_108.txt	1376	1377	1377	1376	0.36%
output_109.txt	1380	1377	1381	1377	0.44%
output_110.txt	1381	1380	1374	1374	0.22%
output_111.txt	1379	1377	1378	1377	0.44%
output_112.txt	1379	1384	1374	1374	0.22%
output_113.txt	1378	1375	1382	1375	0.29%
output_114.txt	1385	1383	1381	1381	0.73%
output_115.txt	1382	1383	1376	1376	0.36%
output_116.txt	1374	1373	1379	1373	0.15%
output_117.txt	1377	1382	1381	1377	0.44%
output_118.txt	1384	1384	1376	1376	0.36%
output_119.txt	1382	1385	1382	1382	0.80%
output_120.txt	1379	1379	1380	1379	0.58%
output_121.txt	1376	1384	1380	1376	0.36%
output_122.txt	1378	1382	1376	1376	0.36%
output_123.txt	1384	1384	1380	1380	0.66%
output_124.txt	1380	1379	1384	1379	0.58%
output_125.txt	1381	1385	1374	1374	0.22%
output_126.txt	1378	1379	1379	1378	0.51%
output_127.txt	1372	1375	1382	1372	0.07%
output_128.txt	1380	1373	1378	1373	0.15%
output_129.txt	1378	1378	1378	1378	0.51%
output_130.txt	1380	1378	1383	1378	0.51%
output_131.txt	1381	1374	1382	1374	0.22%
output_132.txt	1379	1382	1373	1373	0.15%
output_133.txt	1378	1378	1383	1378	0.51%
output_134.txt	1379	1377	1382	1377	0.44%
output_135.txt	1380	1377	1375	1375	0.29%
output_136.txt	1378	1371	1382	1371	0.00%
output_137.txt	1375	1383	1386	1375	0.29%
output_138.txt	1382	1381	1379	1379	0.58%
output_139.txt	1378	1381	1382	1378	0.51%
output_140.txt	1386	1379	1385	1379	0.58%
output_141.txt	1376	1381	1383	1376	0.36%
output_142.txt	1370	1381	1379	1370	-0.07%
output_143.txt	1371	1371	1382	1371	0.00%
output_144.txt	1378	1369	1379	1369	-0.15%
output_145.txt	1374	1375	1372	1372	0.07%
output_146.txt	1373	1382	1383	1373	0.15%
output_147.txt	1383	1380	1382	1380	0.66%
output_148.txt	1386	1382	1384	1382	0.80%
output_149.txt	1383	1386	1377	1377	0.44%
output_150.txt	1376	1381	1375	1375	0.29%
output_151.txt	1371	1380	1371	1371	0.00%
output_152.txt	1381	1383	1378	1378	0.51%
output_153.txt	1381	1379	1370	1370	-0.07%
output_154.txt	1377	1386	1379	1377	0.44%
output_155.txt	1382	1380	1380	1380	0.66%
output_156.txt	1380	1380	1372	1372	0.07%
output_157.txt	1380	1378	1371	1371	0.00%

output_158.txt	1381	1383	1381	1381	0.73%
output_159.txt	1381	1379	1379	1379	0.58%
output_160.txt	1376	1378	1370	1370	-0.07%
output_161.txt	1381	1375	1379	1375	0.29%
output_162.txt	1377	1377	1372	1372	0.07%
output_163.txt	1379	1370	1381	1370	-0.07%
output_164.txt	1379	1376	1379	1376	0.36%
output_165.txt	1376	1380	1380	1376	0.36%
output_166.txt	1379	1383	1379	1379	0.58%
output_167.txt	1380	1380	1385	1380	0.66%
output_168.txt	1370	1378	1370	1370	-0.07%
output_169.txt	1375	1377	1376	1375	0.29%
output_170.txt	1370	1376	1378	1370	-0.07%
output_171.txt	1378	1379	1377	1377	0.44%
output_172.txt	1373	1382	1371	1371	0.00%
output_173.txt	1377	1372	1380	1372	0.07%
output_174.txt	1379	1378	1381	1378	0.51%
output_175.txt	1372	1380	1381	1372	0.07%
output_176.txt	1377	1375	1378	1375	0.29%
output_177.txt	1379	1376	1371	1371	0.00%
output_178.txt	1381	1374	1378	1374	0.22%
output_179.txt	1377	1375	1374	1374	0.22%
output_180.txt	1376	1380	1378	1376	0.36%
output_181.txt	1372	1380	1377	1372	0.07%
output_182.txt	1378	1377	1375	1375	0.29%
output_183.txt	1379	1373	1378	1373	0.15%
output_184.txt	1378	1370	1378	1370	-0.07%
output_185.txt	1381	1382	1379	1379	0.58%
output_186.txt	1377	1379	1369	1369	-0.15%
output_187.txt	1376	1379	1377	1376	0.36%
output_188.txt	1382	1380	1370	1370	-0.07%
output_189.txt	1375	1377	1374	1374	0.22%
output_190.txt	1376	1384	1381	1376	0.36%
output_191.txt	1381	1382	1379	1379	0.58%
output_192.txt	1377	1377	1375	1375	0.29%
output_193.txt	1372	1379	1377	1372	0.07%
output_194.txt	1378	1380	1373	1373	0.15%
output_195.txt	1376	1376	1378	1376	0.36%
output_196.txt	1377	1380	1377	1377	0.44%
output_197.txt	1381	1374	1381	1374	0.22%
output_198.txt	1373	1375	1376	1373	0.15%
output_199.txt	1379	1382	1379	1379	0.58%
output_200.txt	1380	1373	1377	1373	0.15%

Πίνακας 5.3.b: Αποτελέσματα προβλήματος με χρόνους επεξεργασίας ±1.

Execution Times+1		**Execution Times-1**	
Makespan	Time	Makespan	Time
1406	34	1357	36
1404	33	1357	34
1404	34	1355	34

1403	32	1358	36
1403	33	1352	34
1397	34	1350	35
1395	34	1347	35
1401	34	1348	33
1403	34	1349	32
1403	33	**1347**	33
Min	Dev	Min	Dev
1395	**1.75%**	1347	**-1.75%**

- Όταν μία συσκευή έρχεται με βλάβη δηλ. μόνο μία συσκευή έχει τυχαίο χρόνο επεξεργασίας.

Σε αυτήν την περίπτωση, προσομοιώνουμε την κατάσταση όπου όλες οι συσκευές έχουν αιτιοκρατικούς χρόνους επεξεργασίας, εκτός από μία συσκευή που έρχεται με βλάβη και επομένως δεν γνωρίζουμε πόσο χρόνο χρειάζεται για να επισκευαστεί. Επομένως, ο χρόνος επεξεργασίας της είναι στοχαστικός. Θα μεταβάλλουμε λοιπόν τους χρόνους επεξεργασίας της συγκεκριμένης συσκευής κατά ±1 αλλά επειδή και εδώ όλοι οι πιθανοί συνδυασμοί θα είναι $3^5 = 243$, θα τρέξουμε τον αλγόριθμο θεωρώντας ότι υπάρχουν 3 εργαζόμενοι αντί για 5 (καταγραφή συσκευής - διακρίβωση - παραλαβή από τον χρήστη). Έτσι, θα δημιουργηθούν $3^3 = 27$ αρχεία για την συγκεκριμένη συσκευή, τα οποία στη συνέχεια θα δοθούν ως αρχεία εισόδου στον αλγόριθμο ABC για να βρει το μικρότερο makespan. Το παράδειγμα της συσκευής με βλάβη, θα εφαρμοστεί συνολικά σε πέντε τυχαίες συσκευές, μία διαφορετική κάθε φορά. Ως συσκευές με βλάβη θεωρούνται οι: $1^η$, $6^η$, $13^η$, $15^η$ και $20^η$. Οι παρακάτω πίνακες απεικονίζουν τα αποτελέσματα που έδωσε ο ABC για το αιτιοκρατικό πρόβλημα 3 μηχανών αλλά και για κάθε μία περίπτωση συσκευής με βλάβη.

Πίνακας 5.3.c: Αποτελέσματα αιτιοκρατικού προβλήματος με 3 μηχανές.

3 Machines		
	Makespan	**Time**
	1141	20
	1141	22
	1141	22
	1141	21
	1141	21
	1141	22
	1150	22
	1141	21
	1141	22
	1141	21
Min:	**1141**	**20**

Πίνακας 5.3.d: Αποτελέσματα με την 1η συσκευή τυχαία και αποκλίσεις από τη βέλτιστη τιμή 1141.

Problem	Run1	Run2	Run3	Min	Deviation
output_01.txt	1141	1141	1142	1141	0.00%
output_02.txt	1143	1141	1141	1141	0.00%
output_03.txt	1141	1141	1141	1141	0.00%
output_04.txt	1141	1148	1150	1141	0.00%
output_05.txt	1141	1141	1141	1141	0.00%
output_06.txt	1148	1141	1145	1141	0.00%
output_07.txt	1141	1141	1141	1141	0.00%
output_08.txt	1146	1141	1141	1141	0.00%
output_09.txt	1141	1141	1141	1141	0.00%
output_10.txt	1142	1142	1142	1142	0.09%
output_11.txt	1147	1147	1143	1143	0.18%
output_12.txt	1142	1142	1142	1142	0.09%
output_13.txt	1142	1142	1142	1142	0.09%
output_14.txt	1142	1147	1142	1142	0.09%
output_15.txt	1142	1142	1142	1142	0.09%
output_16.txt	1150	1142	1149	1142	0.09%
output_17.txt	1145	1148	1142	1142	0.09%
output_18.txt	1142	1142	1142	1142	0.09%
output_19.txt	1140	1140	1140	1140	-0.09%
output_20.txt	1140	1140	1140	1140	-0.09%
output_21.txt	1149	1140	1140	1140	-0.09%
output_22.txt	1145	1140	1140	1140	-0.09%
output_23.txt	1149	1140	1140	1140	-0.09%
output_24.txt	1140	1140	1140	1140	-0.09%
output_25.txt	1140	1140	1140	1140	-0.09%
output_26.txt	1140	1140	1147	1140	-0.09%
output_27.txt	1140	1140	1140	1140	-0.09%

Πίνακας 5.3.e: Αποτελέσματα με την 6η συσκευή τυχαία και αποκλίσεις από τη βέλτιστη τιμή 1141.

Problem	Run1	Run2	Run3	Min	Deviation
output_01.txt	1141	1141	1150	1141	0.00%
output_02.txt	1150	1143	1141	1141	0.00%
output_03.txt	1141	1141	1149	1141	0.00%
output_04.txt	1141	1141	1141	1141	0.00%
output_05.txt	1150	1141	1149	1141	0.00%
output_06.txt	1141	1147	1141	1141	0.00%
output_07.txt	1141	1141	1142	1141	0.00%
output_08.txt	1142	1141	1141	1141	0.00%
output_09.txt	1141	1141	1141	1141	0.00%
output_10.txt	1146	1142	1148	1142	0.09%
output_11.txt	1142	1142	1142	1142	0.09%
output_12.txt	1142	1148	1142	1142	0.09%
output_13.txt	1142	1142	1142	1142	0.09%
output_14.txt	1142	1142	1143	1142	0.09%
output_15.txt	1142	1142	1142	1142	0.09%
output_16.txt	1142	1142	1142	1142	0.09%
output_17.txt	1142	1142	1142	1142	0.09%

output_18.txt	1142	1142	1142	1142	0.09%
output_19.txt	1141	1140	1146	1140	-0.09%
output_20.txt	1140	1140	1140	1140	-0.09%
output_21.txt	1140	1149	1140	1140	-0.09%
output_22.txt	1140	1140	1140	1140	-0.09%
output_23.txt	1140	1140	1149	1140	-0.09%
output_24.txt	1140	1141	1140	1140	-0.09%
output_25.txt	1140	1140	1140	1140	-0.09%
output_26.txt	1140	1142	1140	1140	-0.09%
output_27.txt	1144	1140	1144	1140	-0.09%

Πίνακας 5.3.f: Αποτελέσματα με την 13η συσκευή τυχαία και αποκλίσεις από τη βέλτιστη τιμή 1141.

Problem	Run1	Run2	Run3	Min	Deviation
output_01.txt	1141	1141	1148	1141	0.00%
output_02.txt	1141	1141	1141	1141	0.00%
output_03.txt	1141	1141	1147	1141	0.00%
output_04.txt	1150	1141	1141	1141	0.00%
output_05.txt	1150	1141	1141	1141	0.00%
output_06.txt	1141	1149	1149	1141	0.00%
output_07.txt	1141	1141	1143	1141	0.00%
output_08.txt	1146	1141	1150	1141	0.00%
output_09.txt	1147	1141	1141	1141	0.00%
output_10.txt	1147	1147	1142	1142	0.09%
output_11.txt	1149	1142	1142	1142	0.09%
output_12.txt	1142	1147	1145	1142	0.09%
output_13.txt	1149	1142	1142	1142	0.09%
output_14.txt	1142	1148	1142	1142	0.09%
output_15.txt	1142	1142	1143	1142	0.09%
output_16.txt	1142	1151	1142	1142	0.09%
output_17.txt	1142	1142	1144	1142	0.09%
output_18.txt	1151	1142	1142	1142	0.09%
output_19.txt	1140	1145	1149	1140	-0.09%
output_20.txt	1140	1140	1140	1140	-0.09%
output_21.txt	1140	1140	1140	1140	-0.09%
output_22.txt	1140	1140	1142	1140	-0.09%
output_23.txt	1142	1140	1148	1140	-0.09%
output_24.txt	1140	1140	1140	1140	-0.09%
output_25.txt	1144	1140	1140	1140	-0.09%
output_26.txt	1140	1140	1140	1140	-0.09%
output_27.txt	1140	1145	1146	1140	-0.09%

Πίνακας 5.3.g: Αποτελέσματα με την 15η συσκευή τυχαία και αποκλίσεις από τη βέλτιστη τιμή 1141.

Problem	Run1	Run2	Run3	Min	Deviation
output_01.txt	1141	1141	1141	1141	0.00%
output_02.txt	1141	1141	1144	1141	0.00%
output_03.txt	1148	1141	1141	1141	0.00%
output_04.txt	1146	1150	1141	1141	0.00%
output_05.txt	1141	1144	1147	1141	0.00%

output_06.txt	1141	1141	1143	1141	0.00%
output_07.txt	1141	1149	1141	1141	0.00%
output_08.txt	1141	1141	1141	1141	0.00%
output_09.txt	1141	1141	1141	1141	0.00%
output_10.txt	1142	1146	1142	1142	0.09%
output_11.txt	1142	1146	1142	1142	0.09%
output_12.txt	1145	1142	1148	1142	0.09%
output_13.txt	1142	1142	1151	1142	0.09%
output_14.txt	1151	1142	1151	1142	0.09%
output_15.txt	1142	1142	1142	1142	0.09%
output_16.txt	1142	1142	1142	1142	0.09%
output_17.txt	1151	1142	1142	1142	0.09%
output_18.txt	1142	1144	1142	1142	0.09%
output_19.txt	1146	1140	1141	1140	-0.09%
output_20.txt	1140	1140	1149	1140	-0.09%
output_21.txt	1140	1142	1145	1140	-0.09%
output_22.txt	1140	1140	1140	1140	-0.09%
output_23.txt	1147	1140	1146	1140	-0.09%
output_24.txt	1140	1148	1147	1140	-0.09%
output_25.txt	1140	1140	1148	1140	-0.09%
output_26.txt	1141	1140	1140	1140	-0.09%
output_27.txt	1140	1140	1144	1140	-0.09%

Πίνακας 5.3.h: Αποτελέσματα με την 20^{η} συσκευή τυχαία και αποκλίσεις από τη βέλτιστη τιμή 1141.

Problem	Run1	Run2	Run3	Min	Deviation
output_01.txt	1141	1141	1142	1141	0.00%
output_02.txt	1147	1150	1141	1141	0.00%
output_03.txt	1150	1141	1141	1141	0.00%
output_04.txt	1141	1141	1141	1141	0.00%
output_05.txt	1149	1141	1141	1141	0.00%
output_06.txt	1141	1141	1143	1141	0.00%
output_07.txt	1141	1141	1141	1141	0.00%
output_08.txt	1147	1141	1143	1141	0.00%
output_09.txt	1141	1142	1141	1141	0.00%
output_10.txt	1142	1147	1142	1142	0.09%
output_11.txt	1142	1142	1142	1142	0.09%
output_12.txt	1142	1142	1142	1142	0.09%
output_13.txt	1142	1142	1142	1142	0.09%
output_14.txt	1142	1151	1142	1142	0.09%
output_15.txt	1142	1149	1145	1142	0.09%
output_16.txt	1142	1142	1142	1142	0.09%
output_17.txt	1142	1142	1147	1142	0.09%
output_18.txt	1146	1142	1142	1142	0.09%
output_19.txt	1140	1140	1140	1140	-0.09%
output_20.txt	1140	1143	1140	1140	-0.09%
output_21.txt	1147	1140	1143	1140	-0.09%
output_22.txt	1144	1140	1140	1140	-0.09%
output_23.txt	1149	1144	1140	1140	-0.09%
output_24.txt	1140	1141	1140	1140	-0.09%
output_25.txt	1140	1145	1140	1140	-0.09%

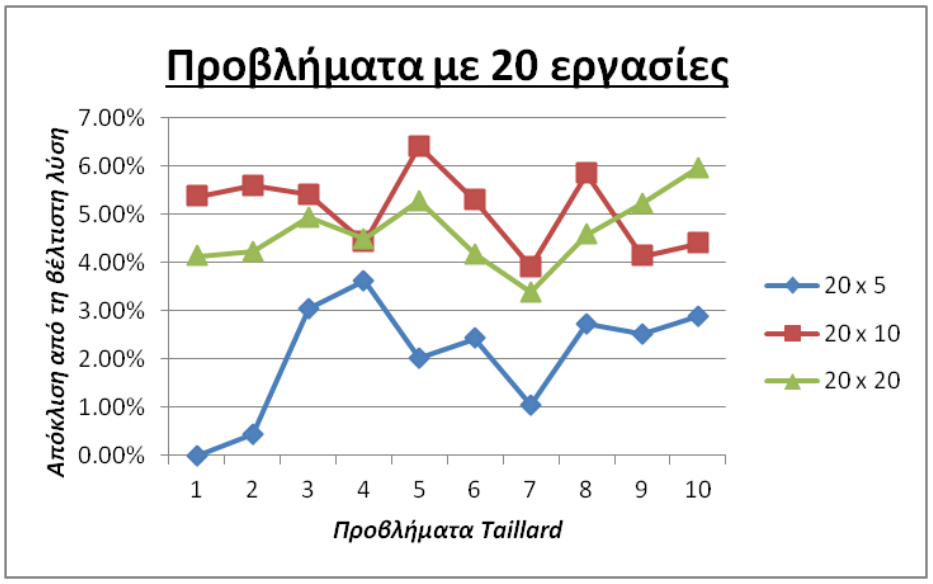
output_26.txt	1149	1140	1147	1140	-0.09%
output_27.txt	1140	1140	1140	1140	-0.09%

ΚΕΦΑΛΑΙΟ 6
ΠΑΡΑΤΗΡΗΣΕΙΣ-ΣΥΜΠΕΡΑΣΜΑΤΑ

6.1 Επίλυση των προβλημάτων Taillard από τον ABC

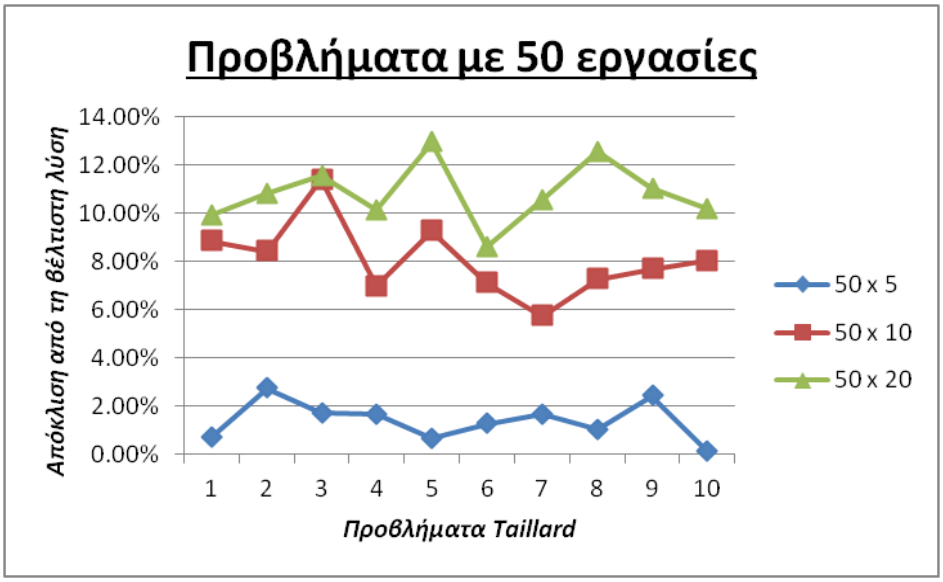
Στο 6^{ο} και τελευταίο κεφάλαιο της παρούσας διπλωματικής εργασίας, θα σχολιάσουμε τα αποτελέσματα που προέκυψαν από την επίλυση τόσο των θεωρητικών προβλημάτων του Taillard όσο και του πραγματικού συστήματος, χρησιμοποιώντας τον αλγόριθμο Artificial Bee Colony.

Αρχικά, θα αξιολογήσουμε τη συμπεριφορά του αλγορίθμου ABC ως προς την επίλυση των 120 θεωρητικών προβλημάτων του Taillard. Τα στοιχεία του πίνακα 5.1.a του προηγούμενου κεφαλαίου, μας επιτρέπουν να σχεδιάσουμε τις ακόλουθες γραφικές παραστάσεις, προκειμένου να καταλήξουμε σε ένα συγκεντρωτικό συμπέρασμα για την αποδοτικότητα του αλγορίθμου.



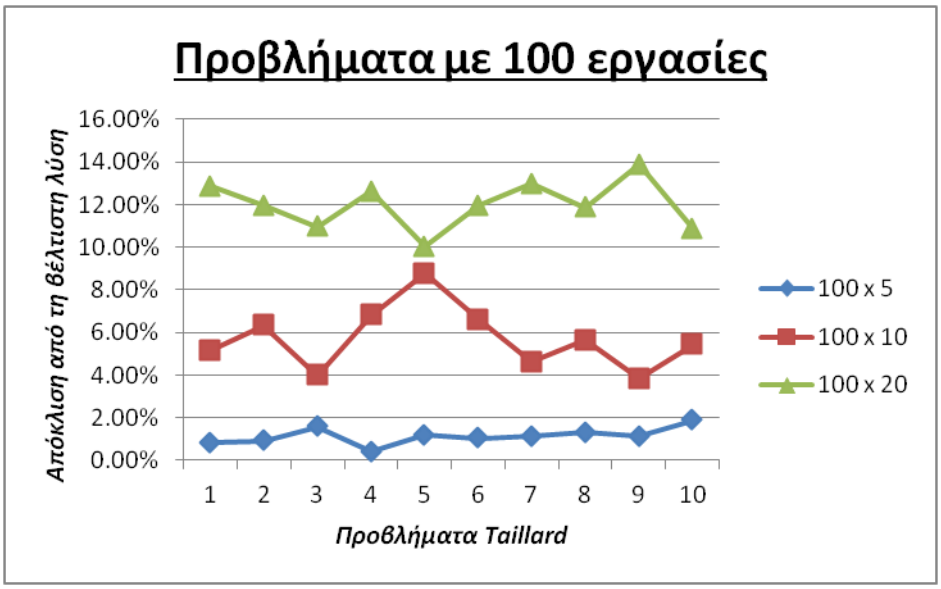
Εικόνα 6.1.1: *Γραφική Παράσταση προβλημάτων με 20 εργασίες και 5, 10, 20 μηχανές.*

Παρατηρούμε ότι ο αλγόριθμος ABC στα 30 προβλήματα του Taillard με 20 εργασίες και 5, 10, 20 μηχανές παράγει αρκετά αποδοτικά αποτελέσματα που η απόκλισή τους από τη βέλτιστη λύση δεν ξεπερνα το 7%. Ιδιαίτερα στα προβλήματα με μικρό αριθμό μηχανών, ο αλγόριθμος δίνει λύσεις με απόσταση 4% από την βέλτιστη. Αξιοσημείωτο αποτελεί το γεγονός ότι οι λύσεις των προβλημάτων 20 εργ.- 20 μηχ. είναι κατά βάση καλύτερες από αυτές των 20 εργ.- 10 μηχ.



Εικόνα 6.1.2: *Γραφική Παράσταση προβλημάτων με 50 εργασίες και 5, 10, 20 μηχανές.*

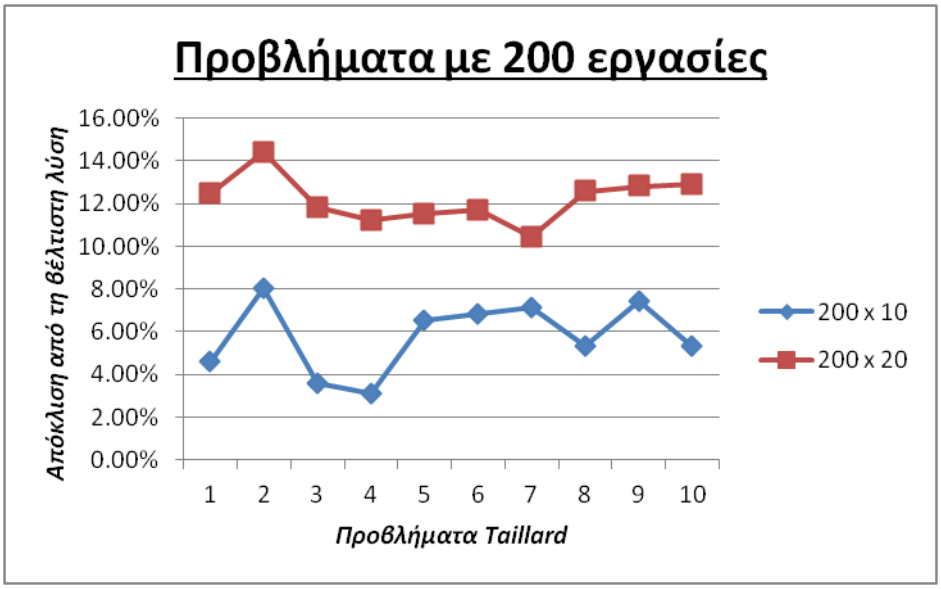
Σε αυτήν την γραφική παράσταση παρατηρούμε ότι, αν και αυξήθηκε ο αριθμός των εργασιών σε 50, εντούτοις οι αποκλίσεις των λύσεων από τις βέλτιστες δε φαίνεται να έχουν επηρεαστεί σημαντικά. Μάλιστα, για τα προβλήματα με 5 μηχανές, η απόκλιση από τη βέλτιστη λύση δεν ξεπερνά και εδώ το 4%. Παρ' όλα αυτά, τα προβλήματα με 10 μηχανές παρουσιάζουν καλύτερες λύσεις σε σχέση με αυτά με τις 20. Τέλος, οι αποκλίσεις των λύσεων που παράγει ο αλγόριθμος για προβλήματα των 50 εργασιών δεν ξεπερνούν το 14%.



Εικόνα 6.1.3: *Γραφική Παράσταση προβλημάτων με 100 εργασίες και 5, 10, 20 μηχανές.*

Τα προβλήματα με 100 εργασίες και 5 μηχανές έχουν και αυτά εξαιρετικές λύσεις που οι αποκλίσεις τους δεν ξεπερνούν το 2%. Εξίσου καλές λύσεις παρουσιάζουν και τα προβλήματα 100 x 10 με αποκλίσεις που δεν ξεπερνούν το 10%. Μάλιστα, οι λύσεις των

συγκεκριμένων προβλημάτων παρουσιάζουν μικρότερες αποκλίσεις από τα αντίστοιχα με προβλήματα 50 x 10. Παρ' όλα αυτά, όλα τα προβλήματα με 100 εργασίες έχουν μέγιστη απόκλιση από το βέλτιστο 14%.



Εικόνα 6.1.4: *Γραφική Παράσταση προβλημάτων με 200 εργασίες και 10, 20 μηχανές.*

Σε αυτήν την κατηγορία προβλημάτων, εξετάζονται οι περιπτώσεις με 200 εργασίες και 10 μέχρι 20 μηχανές. Όπως και στο προηγούμενο διάγραμμα, τα προβλήματα με 10 μηχανές έχουν μέγιστη απόκλιση από το βέλτιστο 8%, ενώ αυτά με τις 20 μηχανές δεν ξεπερνούν το 14%.

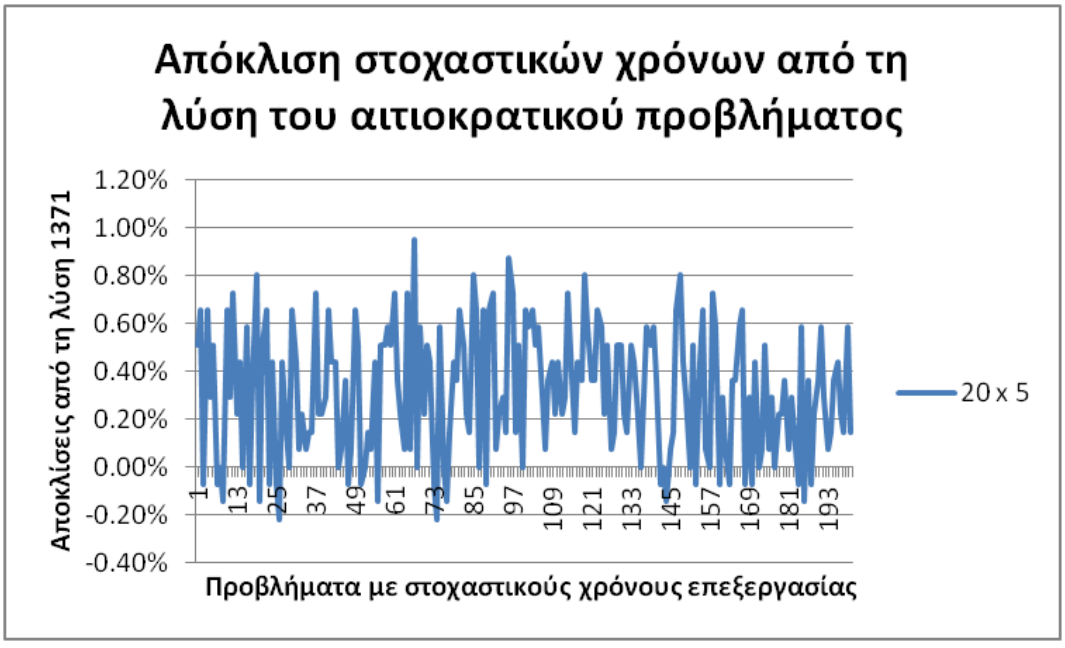
Εικόνα 6.1.5: *Γραφική Παράσταση προβλημάτων με 500 εργασίες και 20 μηχανές.*

Αυτή η γραφική παράσταση απεικονίζει τις αποκλίσεις λύσεων προβλημάτων με 500 εργασίες και 20 μηχανές. Εδώ οι αποκλίσεις λύσεων ξεκινούν από 9,8% και φτάνουν μέχρι 11% με μέση τιμή απόκλισης 10,4%.

Γενικά, παρατηρούμε ότι στην πλειοψηφία των προβλημάτων, οι αποκλίσεις δεν ξεπερνούν το 10% από τις βέλτιστες τιμές τους. Ιδιαίτερα όταν προσανατολιζόμαστε σε προβλήματα με μικρό αριθμό μηχανών, ο αλγόριθμος ABC δίνει εξαιρετικές λύσεις ακόμα και με μεγάλο αριθμό εργασιών (π.χ. 500) και επομένως μπορούμε να τον χρησιμοποιήσουμε και σε ένα πραγματικό σύστημα και να εξάγουμε ασφαλή συμπεράσματα.

6.2 Επίλυση του πραγματικού προβλήματος με στοχαστικούς χρόνους επεξεργασίας από τον ABC

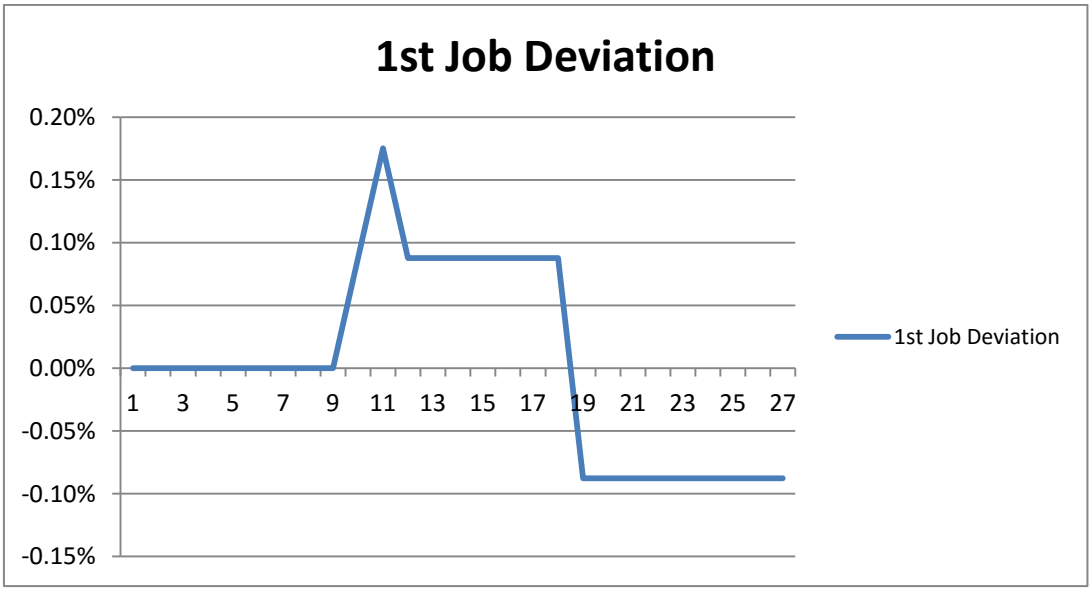
Στο πραγματικό σύστημα παραγωγής του εργαστηρίου διακρίβωσης, το βέλτιστο makespan του προβλήματος 20 συσκευών - 5 εργαζομένων με αιτιοκρατικούς χρόνους επεξεργασίας υπολογίστηκε ως **1371** λεπτά. Στη συνέχεια, οι συσκευές θεωρήθηκε ότι έχουν στοχαστικούς χρόνους επεξεργασίας και από τον πίνακα 5.3.a μπορούμε να καταλήξουμε στην ακόλουθη γραφική παράσταση:



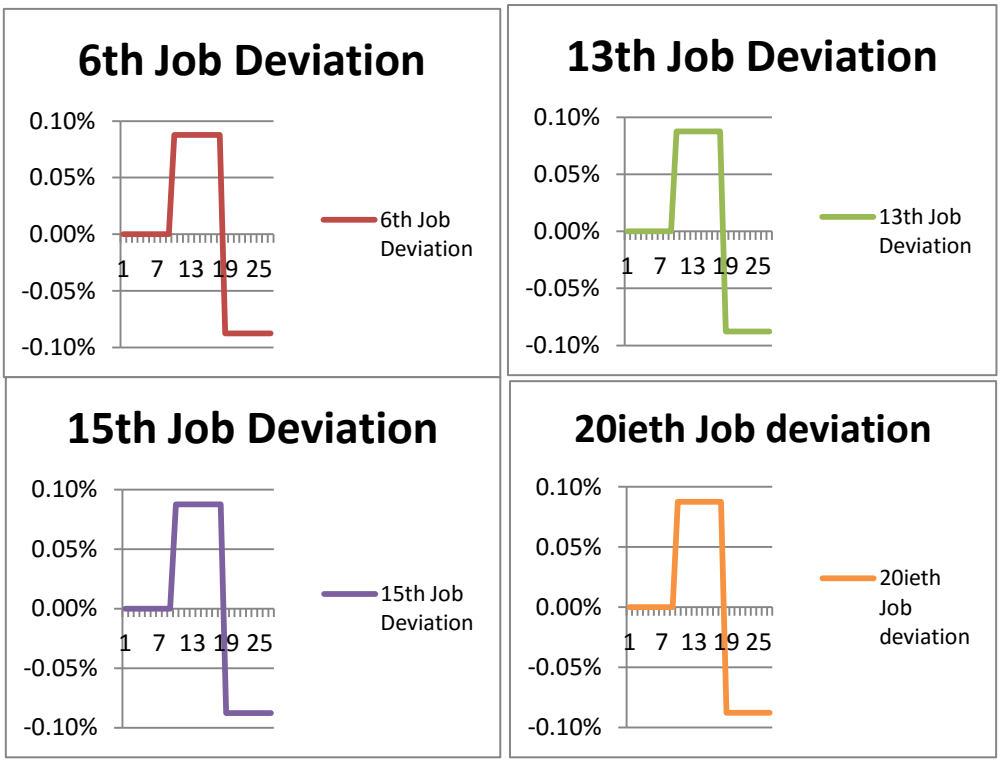
Εικόνα 6.2.1: *Απόκλιση στοχαστικών χρόνων από τη λύση 1371 του αιτιοκρατικού προβλήματος.*

Στην γραφική παράσταση της εικόνας 6.2.1 είναι φανερό ότι οι αποκλίσεις των στοχαστικών λύσεων από την αιτιοκρατική λύση (1371) κυμαίνονται μεταξύ -0,2% και +1% . Βέβαια, σε αυτό το σημείο πρέπει να αναφερθεί ότι αυτή η γραφική παράσταση απεικονίζει μόνο τα 200 από τα 3^{100} προβλήματα με στοχαστικούς χρόνους επεξεργασίας. Για να δούμε λοιπόν τα άνω και κάτω όρια όλων των αποκλίσεων από την αιτιοκρατική λύση, υπολογίσαμε την απόκλιση της λύσης, όταν σε όλους τους χρόνους επεξεργασίας προστέθηκε +1 αλλά και όταν αφαιρέθηκε -1 (πίνακας 5.3.b). Το αποτέλεσμα αυτών των υπολογισμών έδειξε ότι μεταβάλλοντας τους χρόνους επεξεργασίας των συσκευών κατά ±1, η απόκλιση της λύσης κυμαίνεται μεταξύ -1,75% και +1,75%. Μεταβάλλοντας, δηλαδή, ομοιόμορφα τους χρόνους επεξεργασίας, αντίστοιχα μεταβάλλεται ομοιόμορφα και η απόκλιση του makespan από την αιτιοκρατική λύση.

Τέλος, μελετήσαμε την περίπτωση του πραγματικού προβλήματος με 3 εργαζομένους αντί για 5, με αρχικά σταθερούς όλους τους χρόνους επεξεργασίας. Το ελάχιστο makespan, με αυτές τις παραδοχές, είναι 1141 λεπτά και από αυτήν την τιμή θα μελετήσουμε τις αποκλίσεις των λύσεων του στοχαστικού προβλήματος 20 συσκευών - 3 εργαζομένων. Στην περίπτωση αυτή, θεωρήσαμε ως στοχαστικούς τους χρόνους επεξεργασίας μιας μεμονωμένης συσκευής και όχι όλων των συσκευών, όπως στην προηγούμενη περίπτωση. Τρέξαμε αυτό το σενάριο 5 διαφορετικές φορές με αποτελέσματα που φαίνονται στις παρακάτω γραφικές παραστάσεις:



Εικόνα 6.2.2: *Αποκλίσεις λύσεων με στοχαστικούς χρόνους της $1^{ης}$ συσκευής.*



Εικόνα 6.2.3: *Αποκλίσεις λύσεων με στοχαστικούς χρόνους των συσκευών 6, 13, 15 και 20.*

Σε αυτές τις γραφικές παραστάσεις, παρατηρούμε ότι, εισάγοντας τυχαιότητα στις συγκεκριμένες συσκευές, αυτό δεν επηρεάζει την απόκλιση της λύσης τους. Πιο συγκεκριμένα, διαπιστώνουμε ότι σχεδόν όλες οι εργασίες έχουν τις ίδιες αποκλίσεις από το αιτιοκρατικό makespan, με εξαίρεση την πρώτη εργασία που η απόκλισή της έχει ανώτατη τιμή 0,18%. Στις υπόλοιπες περιπτώσεις, οι αποκλίσεις κυμαίνονται από -0,09% έως +0,09% με τη μεταβολή των στοχαστικών χρόνων επεξεργασίας μίας συσκευής κατά ±1. Βλέπουμε δηλαδή ότι η μεταβολή των χρόνων επεξεργασίας μιας συγκεκριμένης συσκευής δεν επιδρά σημαντικά στην αλλαγή της λύσης, αφού η τελευταία μεταβάλλεται μόνο κατά 0,09%.

Έχοντας πλέον μια γενική εικόνα των αποτελεσμάτων του προβλήματος, καταλήγουμε στο συμπέρασμα ότι η μεταβολή των χρόνων επεξεργασίας του προβλήματος αντιμετάθεσης χρονοπρογραμματισμού εργασιών κατά +1 ή -1 μεταβάλλει ομοιόμορφα το makespan κατά +1,75% ή -1,75%, αντίστοιχα. Επίσης, δεν επηρεάζει σημαντικά την τιμή του makespan η μεταβολή των χρόνων επεξεργασίας μιας μεμονωμένης συσκευής κατά ±1 καθώς επίσης και η επιλογή της συγκεκριμένης συσκευής. Είδαμε ότι επιλέγοντας τις εργασίες 1, 6, 13, 15 και 20, οι αποκλίσεις των λύσεών τους παρουσίασαν παρόμοια συμπεριφορά. Επομένως, προκύπτει το γενικό συμπέρασμα ότι το στοχαστικό πρόβλημα, με μεταβολή των χρόνων επεξεργασίας κατά ±1, μπορεί να επιλυθεί θεωρώντας τους χρόνους επεξεργασίας σταθερούς. Δηλαδή ο αλγόριθμος ABC μπορεί να χρησιμοποιηθεί για να επιλύσει το αιτιοκρατικό πρόβλημα, δεδομένου ότι η λύση του τελευταίου δεν διαφέρει σημαντικά από αυτήν του αντίστοιχου στοχαστικο.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Sandeep Kumar, Pooja Jadon, Sandeep Kumar et al (2014). A Novel Hybrid Algorithm for Permutation Flow. *International Journal of Computer Science and Information Technologies (IJCSIT)*, 5 (4), 5057-5061

[2] Matthias Thurer, Mark Stevenson, Charles Protzman (2016). *Card-Based Control Systems for a Lean Work Design*, CRC Press

[3] Kenneth R. Baker, Dan Trietsch (2009). *Principles of Sequencing and Scheduling*, New Jersey : A JOHN WILEY & SONS, INC. publication

[4] George Vairaktarakis (2013). *Flow Shop Scheduling: Theoritical Results, Algorithms and Applications*, New York: Springer Publishing Company

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

[6] Ibrahim M. Alharkan (2005). *Algorithms for Sequencing and Scheduling*, Riyadh, Saudi Arabia: Industrial Engineering Department College of Engineering King Saud University

[7] Byung Jun Joo, Yong Chan Choi, Paul Xirouchakis (2013). Dispatching rule-based algorithms for a dynamic flexible flow shop scheduling problem with time-dependent process defect rate and quality feedback, *Forty Sixth CIRP Conference on Manufacturing Systems*, 163-168

[8] Παπαθεοδώρου Θεόδωρος (2001). *Αλγόριθμοι: Εισαγωγικά Θέματα και Παραδείγματα*, Εκδόσεις Πανεπιστημίου Πατρών

[9] Ιωάννης Μαρινάκης, Μαγδαληνή Μαρινάκη (2010). *Σημειώσεις Μεταπτυχιακού Μαθήματος: Εξελικτικοί Αλγόριθμοι και Βελτιστοποίηση Συστημάτων Μεγάλης Κλίμακας*, Χανιά: Πολυτεχνείο Κρήτης

[10] Talbi, E.-G. (2009). *Metaheuristics: From Design to Implementation*, New Jersey : A JOHN WILEY & SONS, INC. publication

[11] Baris Yuce , Michael S. Packianather , Ernesto Mastrocinque , Duc Truong Pham and Alfredo Lambiase (2013). Honey Bees Inspired Optimization Method: The Bees Algorithm, *Insects*, 4(4), 646-662

[12] Bonabeau E , Dorigo M, Theraulaz G. (1999). *Swarm Intelligence: From Natural to Artificial Systems*, New York : Oxford University Press

[13] Dervis Karaboga (2005). An Idea Based On Honey Bee Swarm for Numerical Optimization, technical report-TR06, *Erciyes University, Engineering Faculty, Computer Engineering Department*

[14] De-Shuang Huang, Kang-Hyun Jo, Abir Hussain. Intelligent Computing Theories and Methodologies. *11th International Conference ICIC 2015*, August 20–23 2015, Fuzhou, China

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