ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΜΗΧΑΝΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ

ΤΟΜΕΑΣ ΒΙΟΜΗΧΑΝΙΚΗΣ ΔΙΟΙΚΗΣΗΣ ΚΑΙ ΕΠΙΧΕΙΡΗΣΙΑΚΗΣ ΕΡΕΥΝΑΣ

Διπλωματική Εργασία

Πρωτούλης Σπυρίδων

Σχεδιασμός και Ανάπτυξη του Αλγορίθμου Πυγολαμπίδας (Firefly Algorithm) για την επίλυση του προβλήματος Flexible Job-Shop Scheduling Problem

Επιβλέπων: Επίκουρος Καθηγητής Σ. Πόνης

Αθήνα, 2015

Πυγολαμπίδες στη φύση

Δηλώνω υπεύθυνα πως έχω διαβάσει και κατανοήσει τους κανόνες για τη λογοκλοπή και τον τρόπο αναφοράς των πηγών όπως αυτοί περιέχονται στον Οδηγό συγγραφής διπλωματικών εργασιών. Η παρούσα διπλωματική εργασία αποτελεί προϊόν προσωπικής εργασίας και όλες οι πηγές που χρησιμοποιούνται έχουν δηλωθεί κατάλληλα στις βιβλιογραφικές παραπομπές και αναφορές.

Πρωτούλης Σπυρίδων

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή κ. Σ.Πόνη για την καθοδήγηση και τη βοήθειά του για την εκπόνηση της παρούσας διπλωματικής εργασίας.

Επίσης ευχαριστώ την Ε.Ρόκου για την πολύτιμη καθοδήγησή της.

Τέλος θα ήθελα να ευχαριστήσω την οικογένειά μου για τη στήριξή τους καθ' όλη τη διάρκεια των σπουδών μου.

Έποψη

Το Flexible Job Shop Scheduling Problem (F-JSSP) είναι ένα πρόβλημα βελτιστοποίησης που αφορά την ανάθεση ενός συνόλου εργασιών σε ένα σύνολο μηχανών για την εκτέλεσή τους. Οι Εξελικτικοί Αλγόριθμοι είναι πλέον διαδεδομένη προσέγγιση για την επίλυση πολύπλοκων προβλημάτων βελτιστοποίησης, όπως το F-JSSP. Ο Αλγόριθμος Πυγολαμπίδας (Firefly Algorithm) είναι ένας σχετικά νέος εξελικτικός αλγόριθμος εμπνευσμένος από τη συμπεριφορά των πυγολαμπίδων στη φύση. Στα πλαίσια της παρούσας διπλωματικής εργασίας σχεδιάστηκε και αναπτύχθηκε ο Αλγόριθμος Πυγολαμπίδας για την επίλυση του F-JSSP.

Abstract

The Flexible Job Shop Scheduling Problem (F-JSSP) is an optimization problem that involves the assignment of a set of given jobs to a set of given machines in order to be executed. Evolutionary Algorithms are widely accepted as an effective approach to complex optimization problems such as the F-JSSP. The Firefly Algorithm is a relatively new evolutionary algorithm inspired by the behavior of fireflies in nature. For the purpose of this diploma thesis a Firefly Algorithm was designed and developed in order to solve the F-JSSP.

Περιεχόμενα

Κατάλογος Σχημάτων και Πινάκων

1. Εισαγωγή

Η ανάγκη για αυξημένη παραγωγικότητα στο σύγχρονο κόσμο των επιχειρήσεων ωθεί τους υπεύθυνους για την οργάνωση της παραγωγικής διαδικασίας στην αναζήτηση κάθε δυνατής ευκαιρίας για μείωση του κόστους με σκοπό την αύξηση του περιθωρίου κέρδους. Στην προσπάθεια αυτή ο χρονικός προγραμματισμός εργασιών και η ανάθεσή τους σε μηχανές αποτελεί βασική διαδικασία που μπορεί να διαδραματίσει σημαντικό ρόλο στην επίτευξη των ως άνω στόχων. Πράγματι, κατά τη διάρκεια των έξι τελευταίων δεκαετιών, οι αποδοτικοί μηχανισμοί για τον χρονικό προγραμματισμό (scheduling) των εργασιών έχουν αναγνωριστεί για την ικανότητά τους να αυξήσουν την παραγωγικότητα και τον βαθμό χρησιμοποίησης των μηχανών και γενικότερα των πόρων που χρησιμοποιούνται κατά την παραγωγική διαδικασία. Η σημασία του χρονικού προγραμματισμού ως καταλύτη των συστημάτων παραγωγής έχει αυξηθεί τα τελευταία χρόνια λόγω της αυξημένης ζήτησης της αγοράς για ποικιλία στα προϊόντα, του μειωμένου κύκλου ζωής των προϊόντων, των αλλαγών στις αγορές λόγω του διεθνούς ανταγωνισμού και της ταχείας ανάπτυξης νέων τεχνολογιών (Blazewicz, Ecker, Pesch, Schmidt, & Weglarz, 2007). Οι οικονομικές αυτές πιέσεις από τις αγορές κάνουν επιτακτική την ανάγκη για την ελαχιστοποίηση των αποθεμάτων διατηρώντας παράλληλα υψηλό το επίπεδο ικανοποίησης των πελατών και την έγκαιρη παραγωγή και παράδοση των προϊόντων. Η επίτευξη του στόχου αυτού απαιτεί ακρίβεια και αποτελεσματικότητα στον χρονικό προγραμματισμό των εργασιών ούτως ώστε τα προϊόντα να παίρνουν την τελική τους μορφή και να είναι έτοιμα προς παράδοση εντός των προβλεπόμενων χρονικών ορίων.

Ο χρονικός προγραμματισμός αλγοριθμικής και επιστημονικής φύσης εμφανίστηκε στο προσκήνιο με την ανάπτυξη και παρουσίαση της πρώτης ευρετικής τεχνικής για τον προγραμματισμό των εργασιών το 1954 (Dantzig, Fulkerson, & Johnson, 1954). Η έρευνα στον τομέα του χρονικού προγραμματισμού της παραγωγής (production scheduling) ξεκινά συνήθως από τον προγραμματισμό εργασιών (jobs) σε μία μηχανή και επεκτείνεται στη συνέχεια σε σύνθετους χώρους παραγωγής όπως τα μεταβλητά, παραμετροποιήσιμα και ευέλικτα συστήματα κατασκευών (Flexible Manufacturing Systems – FMS).

Το υπό εξέταση πρόβλημα της παρούσας διπλωματικής εργασίας, γνωστό ως Flexible Job Shop Scheduling Problem (F-JSSP), ενσωματώνει στοιχεία των ευέλικτων αυτών συστημάτων, τα οποία συναντώνται ως επί το πλείστον σε βιομηχανίες με μεγάλη ποικιλία προϊόντων και με μέτρια ζήτηση για καθένα από τα πολυάριθμα αυτά προϊόντα. Για την επίλυση πολύπλοκων προβλημάτων βελτιστοποίησης, όπως είναι το F-JSSP, χρησιμοποιούνται κατά κόρον μεθευρετικές τεχνικές αναζήτησης λύσεων (metaheuristics). Ορισμένες από τις πλέον διαδεδομένες και αποδεδειγμένα αποδοτικές τέτοιες τεχνικές βασίζονται στη νοημοσύνη σμήνους. Σε αυτή την κατηγορία ανήκει και ο Αλγόριθμος Πυγολαμπίδας (Firefly Algorithm). Με την εξέλιξη των σύγχρονων ηλεκτρονικών υπολογιστών, είναι πλέον δυνατή η χρήση ευφυών συστημάτων με πολλαπλές, αλληλοεπηρεαζόμενες συνιστώσες αναζήτησης, επιτυγχάνοντας έτσι την εξερεύνηση πολύ μεγάλων χώρων πιθανών λύσεων, επιτρέποντας έτσι την εύρεση ποιοτικών λύσεων για τα προβλήματα βελτιστοποίησης σε ικανοποιητικό χρόνο, κατά πολύ μικρότερο εκείνου που απαιτούσαν οι πολύπλοκες και χρονοβόρες παραδοσιακές τεχνικές επίλυσης.

Στα πλαίσια αυτής της διπλωματικής εργασίας αναπτύχθηκε ένας εξελικτικός Αλγόριθμος Πυγολαμπίδων για την επίλυση του F-JSSP, του οποίου ο σχεδιασμός, η ανάπτυξη και ο τρόπος λειτουργίας περιγράφονται εκτενώς στη συνέχεια.

Στο κεφάλαιο 2 καταγράφεται μία επισκόπηση της υπάρχουσας βιβλιογραφίας, τόσο για το Flexible Job-Shop Scheduling Problem, όσο και για τον Αλγόριθμο Πυγολαμπίδων.

Το κεφάλαιο 3 περιέχει τον ακριβή ορισμό του προβλήματος, συμπεριλαμβανομένων των συμβολικών όρων που το περιγράφουν, καθώς και τη μαθηματική μοντελοποίηση του προβλήματος.

Στο κεφάλαιο 4 παρουσιάζεται αναλυτικά η διαδικασία σχεδιασμού του αλγορίθμου, καθώς και η υλοποίηση του κώδικα για την εκτέλεση του αλγορίθμου.

Στο κεφάλαιο 5 παρουσιάζονται αρχικά τα δοκιμαστικά προβλήματα αναφοράς που αντλήθηκαν από τη βιβλιογραφία για τις πειραματικές δοκιμές του αλγορίθμου που αναπτύχθηκε. Ακολουθούν τα πειραματικά αποτελέσματα από την εκτέλεση του αλγορίθμου για τις προαναφερθείσες εκδοχές προβλημάτων F-JSSP.

Τέλος, στο κεφάλαιο 6 παρουσιάζονται τα συμπεράσματα που προέκυψαν από την εκτέλεση του αλγορίθμου.

2. Βιβλιογραφική επισκόπηση

2.1 Χρονικός προγραμματισμός εργασιών (scheduling)

Ο χρονικός προγραμματισμός (scheduling) είναι μια διαδικασία λήψης αποφάσεων η οποία παίζει πολύ σημαντικό ρόλο στις περισσότερες βιομηχανίες παραγωγής προϊόντων αλλά και στις επιχειρήσεις παροχής υπηρεσιών. Χρησιμοποιείται στις προμήθειες και την παραγωγή, στις μεταφορές και τη διανομή αλλά και στην επεξεργασία πληροφοριών και την επικοινωνία. Ο χρονικός προγραμματισμός μιας εταιρίας χρησιμοποιεί μαθηματικές τεχνικές ή ευρετικές μεθόδους (heuristics) με στόχο την ανάθεση περιορισμένων πόρων για την εκτέλεση εργασιών. Κατάλληλη ανάθεση αυτών επιτρέπει στην επιχείρηση να βελτιστοποιήσει τη λειτουργία της και να επιτύχει τους στόχους της. Ως πόροι μπορούν να θεωρηθούν οι μηχανές σε ένα μηχανουργείο, οι αεροδιάδρομοι σε ένα αεροδρόμιο, οι εργάτες σε ένα οικοδομικό έργο αλλά και οι μονάδες επεξεργασίας σε ένα υπολογιστικό περιβάλλον. Ως εργασίες μπορούν να θεωρηθούν οι κατεργασίες σε ένα μηχανουργείο, οι προσγειώσεις και οι απογειώσεις σε ένα αεροδρόμιο, οι φάσεις κατασκευής ενός οικοδομικού έργου ή η εκτέλεση ενός προγράμματος από τον ηλεκτρονικό υπολογιστή. Κάθε εργασία μπορεί να έχει ένα επίπεδο προτεραιότητας, μια χρονική στιγμή από την οποία και μετά είναι δυνατό να ξεκινήσει καθώς και μια χρονική στιγμή μέχρι την οποία θα πρέπει να έχει ολοκληρωθεί (due date). Οι στόχοι παίρνουν διαφορετικές μορφές, όπως για παράδειγμα την ελαχιστοποίηση του χρόνου στον οποίο θα έχουν ολοκληρωθεί όλες οι εργασίες ή την ελαχιστοποίηση του αριθμού των εργασιών που θα ολοκληρωθούν εκπρόθεσμα.

2.2 Ταξινόμηση προβλημάτων: P , NP, NP-Complete, NP-hard

Κρίνεται σκόπιμο σε αυτό το σημείο να γίνει μία αναφορά στην κατηγοριοποίηση των υπολογιστικών προβλημάτων με βάση την πολυπλοκότητά τους. Η κατηγοριοποίηση αυτή βοηθά στο να γίνουν κατανοητοί οι λόγοι που επιλέγονται οι εκάστοτε τεχνικές επίλυσης για πολύπλοκα προβλήματα βελτιστοποίησης στα οποία γίνεται αναφορά σε πολλά σημεία του κειμένου της παρούσας εργασίας.

P

Ένα πρόβλημα απόφασης είναι πρόβλημα «P» όταν αυτό μπορεί να λυθεί σε πολυωνυμικό χρόνο. Για μια δεδομένη εκδοχή (instance) του προβλήματος, η απάντηση «ναι» ή «όχι» μπορεί να καθοριστεί σε πολυωνυμικό χρόνο.

NP

Ένα πρόβλημα απόφασης χαρακτηρίζεται ως NP όταν δεδομένης της απάντησης «ναι» για μια οποιαδήποτε εκδοχή του προβλήματος, είναι δυνατό να επαληθευθεί η εγκυρότητα της σε πολυωνυμικό χρόνο.

NP-Complete

Τα NP-Complete προβλήματα είναι υποκατηγορία των NP προβλημάτων. Πρόκειται για προβλήματα στα οποία μπορεί οποιοδήποτε NP πρόβλημα να αναχθεί σε πολυωνυμικό χρόνο. Τα προβλήματα αυτά είναι ιδιαίτερης σημασίας, αφού εάν βρεθεί αιτιοκρατικός αλγόριθμος ο οποίος μπορεί να λύσει ένα οποιοδήποτε από αυτά σε πολυωνυμικό χρόνο, τότε όλα τα NP προβλήματα μπορούν να λυθούν σε πολυωνυμικό χρόνο. Παρά τις εκτεταμένες προσπάθειες, τέτοιος αλγόριθμος δεν έχει βρεθεί, ωστόσο δεν έχει αποδειχθεί η μη ύπαρξή του.

NP-hard

Τα προβλήματα αυτά είναι γενικά ακόμη δυσκολότερα από τα NP-complete προβλήματα. Να σημειωθεί εδώ πως τα NP-hard προβλήματα δεν ανήκουν κατ' ανάγκη στα NP και δεν είναι κατ' ανάγκη προβλήματα απόφασης τα οποία μπορούν να απαντηθούν με «ναι» ή «όχι». Ο ακριβής ορισμός είναι ο εξής: ένα πρόβλημα «Χ» είναι NP-hard αν υπάρχει NP-complete πρόβλημα «Υ» το οποίο μπορεί να αναχθεί στο «Χ» σε πολυωνυμικό χρόνο. Βέβαια, αφού είναι γνωστό πως κάθε NP-complete πρόβλημα μπορεί να αναχθεί σε πολυωνυμικό χρόνο σε οποιοδήποτε άλλο NP-complete πρόβλημα, αυτό σημαίνει πως όλα τα NP-complete προβλήματα μπορούν σε πολυωνυμικό χρόνο να αναχθούν σε NP-hard προβλήματα. Το πρόβλημα της λήξης ενός προγράμματος (halting problem) είναι ένα κλασικό NP-hard πρόβλημα. Διατυπώνεται ως εξής: δεδομένου ενός προγράμματος «Π» και των εισόδων του «Ε», το πρόγραμμα αυτό θα σταματήσει κάποια στιγμή ή θα τρέχει για πάντα; Ακόμη, το πρόβλημα του περιοδεύοντος πωλητή (travelling salesman problem) καθώς και το Job shop scheduling problem ανήκουν σε αυτή την κατηγορία προβλημάτων (Brucker, 2001) (Garey & Johnson, 1990).

2.3 Προβλήματα χρονικού προγραμματισμού εργασιών

Τα προβλήματα χρονικού προγραμματισμού εργασιών είναι από τα πλέον γνωστά προβλήματα βελτιστοποίησης. Τα προβλήματα αυτά υφίστανται όταν απαιτείται ανάθεση συγκεκριμένων εργασιών για την ολοκλήρωσή τους με χρήση πόρων των οποίων η διαθεσιμότητα είναι περιορισμένη. Γνωστά προβλήματα βελτιστοποίησης αυτού του είδους με ιδιαίτερο ενδιαφέρον είναι το Job-Shop Scheduling Problem (JSSP), το flow-shop scheduling problem (FSP), το πρόβλημα περιοδεύοντος πωλητή (traveling salesman problem-TSP), το πρόβλημα δρομολόγησης οχημάτων (vehicle routing problem-VRP), το πρόβλημα του σακιδίου (knapsack problem) και άλλα.

Το Job Shop Scheduling Problem είναι ένα από τα προβλήματα βελτιστοποίησης με εκτενή βιβλιογραφία. Θεωρείται ως ένα από τα πλέον δύσκολα NP-hard προβλήματα (Allahverdi, Ng, Cheng, & Kovalyov, 2008). Ακόμα και ένα πρόβλημα αναφοράς (benchmark) μεγέθους 10x10 από τους Muth και Thomson (Muth & Thompson, 1963) παρέμεινε άλυτο ως το 1985, ενώ για κανένα πρόβλημα αναφοράς μεγέθους 20x20 δεν έχει βρεθεί επιβεβαιωμένα βέλτιστη λύση ως σήμερα (Zhang, Li, Rao, & Guan, 2008).

Το Flow-shop, το οποίο αποτελεί μια εκδοχή του JSSP, μπορεί να αναχθεί στο traveling salesman problem (TSP), το οποίο TSP είναι και αυτό NP-hard (Nakano & Yamada, 1991).

Η πλέον απλουστευμένη εκδοχή προβλημάτων ανάθεσης εργασιών είναι το πρόβλημα ανάθεσης εργασιών σε μια μόνο μηχανή, πρόβλημα το οποίο, αν και αρκετά παλιό, μπορεί να εμφανίσει δυσκολίες στην επίλυση του (Sidney, 1977). Στο πρόβλημα αυτό, δίνεται ένα σύνολο εργασιών καθεμία από τις οποίες έχει ένα ζεύγος χρονικών στιγμών (a_i,b_i), όπου a_i ο επιθυμητός χρόνος εκκίνησης της εργασίας και b_i ο επιθυμητός χρόνος ολοκλήρωσής της. Υπάρχει ποινή τόσο για την πρόωρη εκκίνηση όσο και για την καθυστέρηση της εργασίας. Στόχος είναι η ανάθεση των εργασιών στη μηχανή με σειρά τέτοια ώστε να ελαχιστοποιηθούν τόσο οι καθυστερήσεις όσο και οι πρόωρες εκκινήσεις. Για ένα πρόβλημα μίας μηχανής με N εργασίες υπάρχουν N! διαφορετικές λύσεις το σύνολο των οποίων ονομάζεται χώρος λύσεων. Επομένως η πολυπλοκότητα του προβλήματος είναι O(N!).

Η επόμενη εκδοχή είναι το πρόβλημα flow-shop scheduling το οποίο αποτελεί ουσιαστικά ένα σύνολο προβλημάτων ανάθεσης εργασιών σε μια και μόνο μηχανή. Το FSP περιλαμβάνει Μ μηχανές και Ν εργασίες (jobs). Κάθε εργασία χρειάζεται να περάσει από όλες τις μηχανές, ενώ η σειρά με την οποία θα συμβεί αυτό είναι κοινή για όλες τις εργασίες του προβλήματος. Ο χρόνος κατεργασίας κάθε εργασίας από κάθε μηχανή είναι γνωστός. Στόχος είναι η ανάθεση των εργασιών στις μηχανές με τρόπο τέτοιο ώστε να ελαχιστοποιηθεί ο συνολικός χρόνος στον οποίο θα έχουν ολοκληρωθεί όλες οι εργασίες (makespan). Οι προσπάθειες για την εύρεση βέλτιστης λύσης των FSP ξεκίνησαν με συμβατικές μεθόδους βελτιστοποίησης πριν από πάνω από τέσσερις δεκαετίες (Ignall & Schrage, 1965; McMahon & Burton, 1967). Το FSP αναφέρεται ως ισχυρό NP-hard πρόβλημα όσον αφορά τη πολυπλοκότητά του (Lin & Cheng, 2001).

Το Job Shop Scheduling Problem είναι γενικότερο του FSP, αφού η σειρά με την οποία χρησιμοποιούνται οι μηχανές διαφέρει ανά εργασία. Είναι όμοιο με το FSP μόνο όσον αφορά το μέγεθος του προβλήματος. Στο JSSP, Ν εργασίες πρέπει να επεξεργαστούν από Μ μηχανές, ενώ η σειρά των μηχανών ανά εργασία δίνεται από τον ορισμό του προβλήματος. Ο χώρος λύσεων είναι $(N!)^M$ και τα προβλήματα JSSP έχουν εκθετική πολυπλοκότητα.

2.3.1 Το πρόβλημα Job-Shop Scheduling (JSSP)

Σε ένα κλασικό job-shop scheduling problem υπάρχουν N εργασίες (jobs) και M μηχανές (machines). Κάθε εργασία αποτελείται από μια αλυσίδα έως M λειτουργιών (operations). Κάθε λειτουργία μπορεί να εκτελεστεί αποκλειστικά από μία συγκεκριμένη μηχανή. Πέρα από αυτά, άλλα βασικά χαρακτηριστικά ενός κλασικού JSSP είναι τα παρακάτω:

- Ο αριθμός των λειτουργιών κάθε εργασίας είναι περατός.
- Ο χρόνος εκτέλεσης κάθε λειτουργίας σε μια συγκεκριμένη μηχανή είναι ορισμένος από το πρόβλημα.
- Η σειρά εκτέλεσης των λειτουργιών για την ολοκλήρωση μίας εργασίας είναι προκαθορισμένη.
- Δεν ορίζονται χρόνοι παράδοσης των προϊόντων/ χρόνοι ολοκλήρωσης των εργασιών.
- Δεν υπάρχει κόστος προετοιμασίας (setup) ούτε κόστος καθυστέρησης μιας εργασίας.
- Μια μηχανή μπορεί να εκτελεί μονάχα μία λειτουργία σε δεδομένη χρονική στιγμή.
- Κάθε εργασία περνάει από μια μηχανή μόνο μια φορά. (Δεν υπάρχει επανακυκλοφορία (recirculation)).
- Για την εκκίνηση της εκτέλεσης μίας λειτουργίας απαιτείται να έχει ολοκληρωθεί η εκτέλεση της προηγούμενης κατά σειρά λειτουργίας της ίδιας εργασίας.
- Το σύστημα δεν μπορεί να διακοπεί πριν την ολοκλήρωση όλων των εργασιών.
- Δεν είναι δυνατή η παύση μίας λειτουργίας πριν την ολοκλήρωσή της.
- Η αποδοτικότητα των μηχανών είναι σταθερή.
- Τη χρονική στιγμή 0 όλες οι μηχανές είναι ελεύθερες.

2.3.1.1 Διαγράμματα Gantt

Στα προβλήματα χρονικού προγραμματισμού εργασιών, ο πλέον διαδεδομένος τρόπος αναπαράστασης των χρονικών προγραμμάτων (δηλαδή των λύσεων για ένα τέτοιο πρόβλημα) είναι τα διαγράμματα Gantt, τα οποία έλαβαν την ονομασία τους από τον Henry Gantt ο οποίος τα εισήγαγε το 1910 (Wilson, 2003).

Στα διαγράμματα αυτά κάθε λειτουργία αναπαριστάται από μία μπάρα. Ο οριζόντιος άξονας του διαγράμματος αντιστοιχεί στον χρόνο και έτσι η αρχή της μπάρας που αντιστοιχεί σε μια λειτουργία υποδεικνύει τη χρονική στιγμή έναρξης της λειτουργίας ενώ το τέλος της μπάρας τη χρονική στιγμή ολοκλήρωσης της λειτουργίας. Ο κατακόρυφος άξονας μπορεί να αντιστοιχεί είτε στους πόρους που χρησιμοποιούνται για τις εργασίες (π.χ. μηχανές) είτε στις ίδιες τις εργασίες οι οποίες αποτελούνται από ένα σύνολο λειτουργιών. Καθεμία από τις δύο αυτές διαφορετικές περιπτώσεις σχηματικής απεικόνισης εξυπηρετεί την κατανόηση του χρονικού προγράμματος με προσανατολισμό στις μηχανές ή τις εργασίες του προβλήματος.

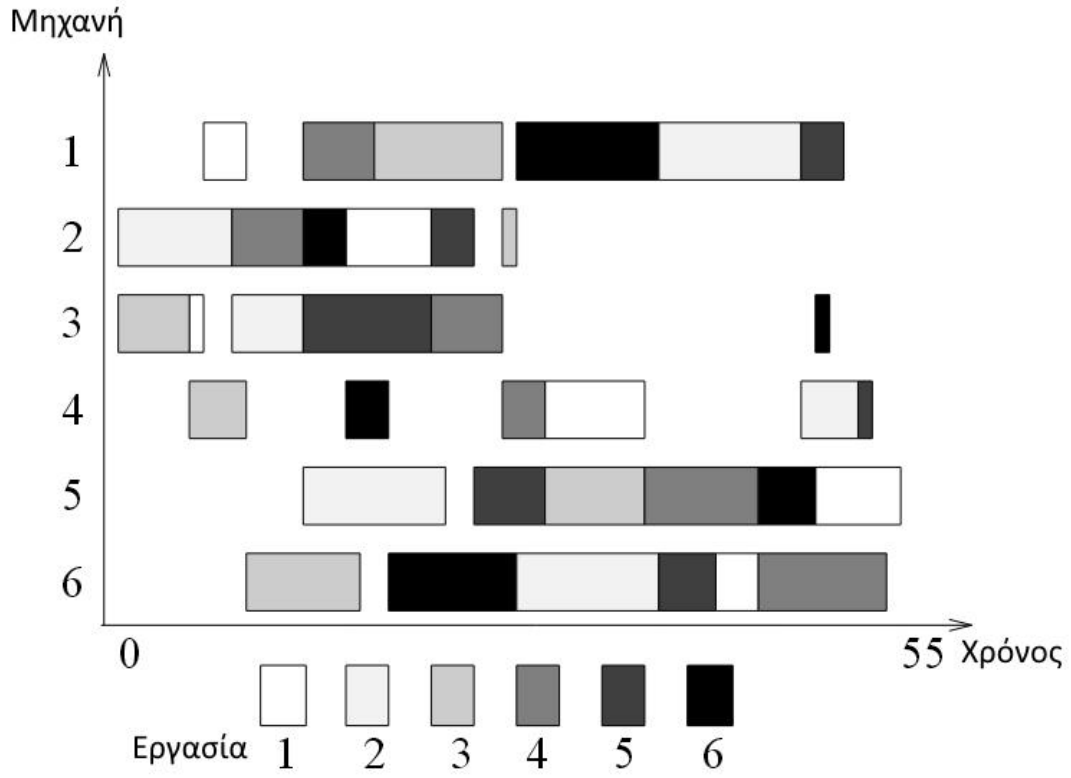
2.3.1.2 Παράδειγμα JSSP

Για την καλύτερη κατανόηση του JSSP, ακολουθεί μια γνωστή από τη βιβλιογραφία περίπτωση προβλήματος αναφοράς. Η εκδοχή αυτή εισήχθη από τους Muth & Thompson (1963). Περιλαμβάνει 6 εργασίες οι οποίες αποτελούνται από 6 λειτουργίες η καθεμία και 6 διαφορετικές μηχανές για την εκτέλεση των λειτουργιών. Στον παρακάτω πίνακα, οι πρώτοι αριθμοί σε κάθε ζεύγος αριθμών αντιστοιχούν στη μηχανή στην οποία πρέπει να εκτελεστεί η λειτουργία, ενώ οι δεύτεροι αριθμοί του ζεύγους αντιστοιχούν στη διάρκεια εκτέλεσης της λειτουργίας από τη μηχανή αυτή.

	Λειτουργία 1	Λειτουργία 2	Λειτουργία 3	Λειτουργία 4	Λειτουργία 5	Λειτουργία 6
	(μηχ,χρ)	(μηχ,χρ)	(μηχ,χρ)	(μηχ,χρ)	(μηχ,χρ)	(μηχ,χρ)
Εργασία 1	3,1	1,3	2,6	4,7	6,3	5,6
Εργασία 2	2,8	3,5	5,10	6,10	1,10	4,4
Εργασία 3	3,5	3,3	6,8	1,9	2,1	5,7
Εργασία 4	2,5	1,5	3,5	4,3	5,8	6,9
Εργασία 5	3,9	2,3	5,5	6,4	1,3	4,1
Εργασία 6	2,3	4,3	6,9	1,10	5,4	3,1

Σχήμα 1: Πίνακας προβλήματος Job Shop Muth & Thompson 6x6

Παρακάτω φαίνεται το διάγραμμα Gantt εκτέλεσης των λειτουργιών του προβλήματος 6x6 των Muth & Thompson όπως αυτό επιλύθηκε με τη χρήση Γενετικού Αλγορίθμου (Fang et al., 1993), με συνολικό χρόνο για την ολοκλήρωση όλων των εργασιών 55 μονάδες χρόνου.



Σχήμα 2: Διάγραμμα Gantt μίας εφικτής λύσης του προβλήματος του Πίνακα 1 (Fang, Ross, & Corne, 1993)

2.3.2 Παραλλαγές JSSP

Το κλασικό JSSP περιλαμβάνει τους βασικούς περιορισμούς, οι οποίοι αφορούν την απαίτηση για ολοκλήρωση της προηγούμενης λειτουργίας πριν την έναρξη της επόμενης κατά σειρά λειτουργίας της εργασίας, τη δυνατότητα μιας μηχανής να εκτελέσει μία λειτουργία, τη δυνατότητα μιας μηχανής να εκτελεί μόνο μία λειτουργία κάθε χρονική στιγμή και να μην διακόπτει μια λειτουργία αφού ξεκινήσει να την εκτελεί πριν ολοκληρωθεί αυτή, το κατά πόσο μια μηχανή είναι ελεύθερη σε μια δεδομένη χρονική στιγμή κλπ. Στην πράξη, τα προβλήματα ανάθεσης εργασιών αντιμετωπίζουν πολλούς ακόμη περιορισμούς καθώς και παράγοντες που μπορούν να οδηγήσουν στη διακοπή μιας διεργασίας. Υπάρχουν για παράδειγμα ημερομηνίες παράδοσης προϊόντων, ενώ μπορεί μια διεργασία να διακοπεί λόγω βλάβης σε μια μηχανή ή λόγω αλλαγών στις ημερομηνίες παράδοσης που καθιστούν επείγουσα την ολοκλήρωση μιας άλλης εργασίας. Οι μεθοδολογίες για την επίλυση προβλημάτων στη βιομηχανία λαμβάνουν υπόψη αυτούς και πολλούς ακόμη παράγοντες. Έτσι, πέρα από το κλασικό JSSP έχουν εμφανιστεί παραλλαγές του προβλήματος που συμπεριλαμβάνουν κάποιους από αυτούς για την καλύτερη προσέγγιση των πραγματικών συνθηκών.

2.3.2.1 Επιπλέον περιορισμοί στα JSSP

Σε συνδυασμό με τους βασικούς περιορισμούς τα προβλήματα Job-Shop μπορούν να συμπεριλάβουν και άλλες αρχικές θεωρήσεις και περιορισμούς ώστε να ανταποκρίνονται σε διαφορετικές εκφάνσεις του ιδίου κατά βάση προβλήματος. Για παράδειγμα, οι Nuijten και Aarts (1996) εισήγαγαν μια παραλλαγή του προβλήματος στην οποία κάθε μηχανή διαθέτει ένα όριο πληρότητας. Κάθε λειτουργία διαθέτει ένα «βάρος», το οποίο αντικατοπτρίζει το ποσοστό στο οποίο απασχολεί τη μηχανή. Σε αυτή την παραλλαγή του προβλήματος, κάθε μηχανή μπορεί να εκτελεί περισσότερες από μια εργασίες ταυτόχρονα, αρκεί το συνολικό τους «βάρος» να μην υπερβαίνει το όριο πληρότητας της μηχανής.

Οι Xi et al. (2009) και Huiyuan et al. (2009) διερεύνησαν την επίλυση παραλλαγών του προβλήματος με πολλαπλούς περιορισμούς πόρων. Σε αυτή την περίπτωση, οι μηχανές και η διαθεσιμότητα αυτών δεν είναι ο μοναδικός πόρος, αλλά εισάγονται και περιορισμοί όπως η διαθεσιμότητα καλουπιών σε ένα εργοστάσιο παραγωγής πλαστικών.

Οι Balas et al. (2008) ασχολήθηκαν με την επίλυση του JSSP με χρόνους προετοιμασίας (setup times) εξαρτημένους από την αλληλουχία των εργασιών και την ύπαρξη ημερομηνιών εμφάνισης των εργασιών (release dates). Κατέληξαν στο ότι η παρουσία τέτοιων περιορισμών επηρεάζει σημαντικά τη συμπεριφορά του προβλήματος.

2.3.2.2 Χρονικά παράθυρα (Time windows)

Με την εισαγωγή χρονικών παραθύρων(time windows), δίνονται συγκεκριμένα χρονικά περιθώρια για την ολοκλήρωση ή και την έναρξη των εργασιών. Σε αντίθεση με τους αυστηρά καθορισμένους χρόνους ολοκλήρωσης, αυτή η παραλλαγή του προβλήματος λαμβάνει υπόψη και το κόστος πρόωρης ολοκλήρωσης των εργασιών, αφού το παράθυρο χρόνου λειτουργεί ως διπλός περιορισμός, με το κάτω χρονικό του όριο να προλαμβάνει την πρόωρη ολοκλήρωση εργασιών που μπορεί να οδηγήσει σε προβλήματα όπως η έλλειψη διαθέσιμου χώρου στην αποθήκη.

2.3.2.3 Διακοπή εργασιών

Σε άλλες παραλλαγές του προβλήματος λαμβάνεται υπόψη η πιθανότητα διακοπής κάποιας διεργασίας. Στην περίπτωση αυτή πραγματοποιείται επαναπρογραμματισμός των εργασιών. Παρότι οι διακοπές δεν είναι περιοδικά γεγονότα, έχουν γίνει απόπειρες για την ενσωμάτωση προβλέψεων με βάση ιστορικά δεδομένα χρησιμοποιώντας στατιστικές κατανομές (Suwa & Sandoh, 2007).

2.3.2.4 To Flexible Job Shop Scheduling Problem (F-JSSP)

Το Flexible Job Shop Scheduling Problem (F-JSSP) , του οποίου η επίλυση αποτελεί αντικείμενο της εργασίας αυτής, αποτελεί γενίκευση του κλασικού JSSP. Συγκεκριμένα, στην περίπτωση του F-JSSP οι λειτουργίες δεν έχουν μία προκαθορισμένη μηχανή στην οποία πρέπει να εκτελεστούν. Αντίθετα, μία λειτουργία είναι δυνατό να εκτελεστεί σε οποιαδήποτε μηχανή από ένα σύνολο μηχανών το οποίο καθορίζεται κατά τον ορισμό του προβλήματος για τη συγκεκριμένη λειτουργία. Να σημειωθεί πως ο απαιτούμενος χρόνος επεξεργασίας μιας λειτουργίας διαφέρει από μηχανή σε μηχανή. Το F-JSSP θα αναλυθεί σε βάθος στα επόμενα κεφάλαια της εργασίας αυτής.

2.3.4 Στόχοι προς επίτευξη (Objectives)

Υπάρχουν αρκετοί στόχοι – objectives των οποίων η επίτευξη μπορεί να επιδιώκεται κατά την επίλυση ενός JSSP. Ο πιο ευρέως χρησιμοποιούμενος στόχος είναι η ελαχιστοποίηση του συνολικού χρόνου στον οποίο θα έχουν ολοκληρωθεί όλες οι δοσμένες από το πρόβλημα εργασίες, γνωστός και ως makespan. Αυτός είναι και ο στόχος που έχει επιλεχθεί στα πλαίσια αυτής της διπλωματικής εργασίας.

Λόγω του ότι η πρόωρη ολοκλήρωση εργασιών καθώς και η καθυστέρησή τους έχουν ιδιαίτερη σημασία στα προβλήματα μίας μηχανής (Sidney, 1977), ορισμένοι ερευνητές χρησιμοποιούν αυτούς τους παράγοντες σαν κριτήριο βελτιστοποίησης των λύσεων (Feng & Lau, 2008); (Yau, Pan, & Shi, 2008); (Li, Mosheiov, & Yovel, 2008)). Αυτά και ορισμένα ακόμη κριτήρια βελτιστοποίησης περιγράφονται συνοπτικά παρακάτω.

2.3.4.1 Χρόνος ολοκλήρωσης όλων των εργασιών (makespan)

Το πιο συνηθισμένο κριτήριο βελτιστοποίησης είναι η ελαχιστοποίηση του χρόνου ολοκλήρωσης όλων των εργασιών του προβλήματος (makespan) (Adams, Balas, & Zawack, 1988);(Binato, Hery, Loewenstern, & Resende, 2002). Από τα πρώτα ακόμα βήματα των προβλημάτων χρονικού προγραμματισμού στη βιομηχανία, η ελαχιστοποίηση του makespan υπήρξε το πλέον δημοφιλής στόχος. Σε ένα χρονοδιάγραμμα, κάθε εργασία προσδιορίζεται από την απαιτούμενη από αυτή μηχανή, την χρονική στιγμή έναρξης και την χρονική στιγμή λήξης της εργασίας. Από αυτές τις πληροφορίες μπορεί να υπολογιστεί ο συνολικός χρόνος για την ολοκλήρωση όλων των εργασιών. Η συνολική διάρκεια είναι ο μέγιστος αριθμός από ένα σύνολο αριθμών C που περιέχει τις χρονικές στιγμές λήξης των εργασιών (jobs).

Η αντικειμενική συνάρτηση (objective function) αναπαριστάται ως εξής:

Ελαχιστοποίησε C_{max}

Όπου

$C_{max} \geq Cn \ \forall n$ και $C_{max} \in \mathbf{C}$

Με μια πρώτη ματιά η ελαχιστοποίηση του makespan φαίνεται παρόμοια με την ελαχιστοποίηση των καθυστερήσεων στις μηχανές. Ωστόσο στην πραγματικότητα, η επιτηδευμένη καθυστέρηση ορισμένων εργασιών μπορεί να βελτιώσει το makespan. (Gonçalves, de Magalhães Mendes, & Resende, 2005).

2.3.4.2 Χρόνος αποπεράτωσης (Throughput Time)

Ο χρόνος αποπεράτωσης (throughput) είναι το συνολικό άθροισμα των χρόνων ολοκλήρωσης των εργασιών. Η ελαχιστοποίηση του makespan επικεντρώνεται στη χρονική στιγμή λήξης της τελευταίας εργασίας που θα ολοκληρωθεί, ενώ ο throughput time λαμβάνει υπόψην όλες τις εργασίες. Ο χρόνος αποπεράτωσης είναι ένας απαιτητικός ως προς τη βελτιστοποίησή του παράγοντας τόσο στα προβλήματα job shop όσο και στα flow shop. Υπολογίζεται από τη σχέση:

$$G = \sum_{n=1}^{N} C_n$$

όπου Cn η χρονική στιγμή λήξης της εργασίας n.

2.3.4.3 Πρόωρη έναρξη ή λήξη εργασιών / Καθυστερημένη ολοκλήρωση εργασιών

Αυτοί οι παράγοντες βελτιστοποίησης αναφέρονται σε προβλήματα στα οποία δίνονται επιθυμητοί χρόνοι έναρξης ή ολοκλήρωσης για τις εργασίες.

Σε ορισμένες περιπτώσεις υπάρχει ποινή για την πρόωρη λήξη μίας εργασίας (Feng & Lau, 2008). Για την εύρεση της βέλτιστης λύσης στα προβλήματα αυτά λαμβάνεται υπόψην η συνολική ποινή και επιδιώκεται η ελαχιστοποίησή της.

Ο συνολικός χρόνος πρόωρης λήξης εργασιών υπολογίζεται από τη σχέση:

$$E = \sum_{n=1}^{N} (||C_n|| - C_n) \times a_n \text{ όπου } \alpha_n = \begin{cases} 1 \ \gamma\iota\alpha \ \left|||C_n||\right| > C_n \\ 0 \ \gamma\iota\alpha \ \left|||C_n||\right| \leq C_n \end{cases}$$

Όπου ||Fn|| και Fn ο επιθυμητός και ο πραγματικός χρόνος ολοκλήρωσης της εργασίας αντίστοιχα.

Με αντίστοιχο τρόπο υπολογίζεται και ο συνολικός χρόνος καθυστέρησης για την ολοκλήρωση των εργασιών. Εκφράζεται με τη σχέση:

$$T = \sum_{n=1}^{N} (C_n - ||C_n||) \times a_n \text{ όπου } \alpha_n = \begin{cases} 1 \ \gamma\iota\alpha \ C_n > \left|||C_n||\right| \\ 0 \ \gamma\iota\alpha \ C_n \leq \left|||C_n||\right| \end{cases}$$

Στην περίπτωση που τίθεται ένας τέτοιος στόχος, κάθε εργασία συνοδεύεται από απαιτήσεις των πελατών, περιορισμούς χωρητικότητας, διαθεσιμότητα πόρων και άλλους παράγοντες που χρειάζεται να ληφθούν υπόψη (Mattfeld & Bierwirth, 2004).

Ανάλογα με το μέγεθος της καθυστέρησης, συχνά εμφανίζεται και ένας παράγοντας «βάρους» Wn ο οποίος πολλαπλασιάζεται με τον α_n. Επειδή συχνά όσο μεγαλύτερη είναι μια καθυστέρηση π.χ. στην παράδοση ενός προϊόντος, ο παράγοντας αυτός εκφράζει τη μη γραμμικότητα στη σχέση μεταξύ καθυστέρησης και ζημιάς.

2.4 Κατηγορίες μεθόδων επίλυσης για το JSSP

Από τα πρώτα χρόνια της εμφάνισης του JSSP μέχρι σήμερα έχουν δοκιμαστεί για αυτό πληθώρα μεθόδων επίλυσης. Το JSSP ανήκει στα NP-hard προβλήματα όπως αυτά περιγράφηκαν σε προηγούμενο κεφάλαιο και ο χώρος έρευνας για λύσεις του προβλήματος μεγεθύνεται εκθετικά συναρτήσει του μεγέθους της εκδοχής του προβλήματος. Συγκεκριμένα, αν θεωρήσουμε πως κάθε εργασία μπορεί να εκτελεστεί από κάθε μηχανή, ο χώρος έρευνας των λύσεων ενός προβλήματος n εργασιών σε m μηχανές είναι της τάξης $(n!)^m$. Αυτό σημαίνει πως για ένα σύστημα μόλις 5 μηχανών και 5 εργασιών έχει 24,883,200,000 δυνατούς συνδυασμούς .

Έτσι, οι συμβατικές και αναλυτικές μέθοδοι επίλυσης προβλημάτων βρίσκουν εφαρμογή μόνο σε εκδοχές του προβλήματος με πολύ περιορισμένο αριθμό μηχανών και εργασιών.

2.4.1 Αναλυτικές μέθοδοι επίλυσης

Οι κλασικές-αναλυτικές μέθοδοι βελτιστοποίησης είναι μέθοδοι που εγγυώνται την εύρεση της βέλτιστης λύσης. Χρησιμοποιήθηκαν στα πρώτα βήματα για την επίλυση του JSSP. Ωστόσο, οι μέθοδοι αυτοί έχουν εκθετική πολυπλοκότητα. Το γεγονός αυτό, σε συνδυασμό με το ότι το JSSP είναι NP-hard πρόβλημα όπως αναλύθηκε παραπάνω, καθιστούν τις αναλυτικές μεθόδους επίλυσης πρακτικά χρήσιμες αποκλειστικά και μόνο για μικρές εκδοχές του προβλήματος. Σε κάθε άλλη περίπτωση είναι απαραίτητη η χρήση ευρετικών και μεθευρετικών μεθόδων, οι οποίες δεν εγγυώνται κατ' ανάγκη την εύρεση βέλτιστης λύσης. Η μέθοδος branch and bound είναι η πλέον διαδεδομένη από τις αναλυτικές μεθόδους αναζήτησης λύσεων για το JSSP. Εφαρμόσθηκε για το JSSP από τους (ASHOUR & Hiremath, 1973) και βασίζεται στην αρχή της απαρίθμησης όλων των εφικτών λύσεων για το πρόβλημα βελτιστοποίησης με τρόπο τέτοιο ώστε χαρακτηριστικά που δεν παρουσιάζονται σε καμία από τις επιτυχημένες λύσεις να απαλείφονται το νωρίτερο δυνατόν (Blazewicz et al., 2007).

2.4.2 Ευρετικές μέθοδοι επίλυσης

Ο όρος ευρετική μέθοδος επίλυσης (**heuristic**) αναφέρεται στην εύρεση μιας ικανοποιητικής λύσης για ένα πρόβλημα μέσα σε λογικό χρονικό περιθώριο ακολουθώντας εμπειρικές διαδικασίες. Η λύση αυτή δεν πρόκειται κατ' ανάγκη για τη βέλτιστη δυνατή, αλλά στις περισσότερες περιπτώσεις την προσεγγίζει σε ικανοποιητικό βαθμό δίχως να απαιτεί απαγορευτικά πολύ χρόνο για την εύρεσή της.

Υπάρχουν τέσσερα κριτήρια που καθορίζουν την καταλληλότητα ή μη των ευρετικών μεθόδων για την επίλυση ενός προβλήματος (Newell & Simon, 1976). Συνοπτικά, αυτά είναι:

- Αν υπάρχουν πολλές, εξίσου καλές, πιθανές λύσεις για το δοσμένο πρόβλημα πρέπει να εξετασθεί το πόσες από αυτές χρειαζόμαστε καθώς και το αν η ευρετική μπορεί να τις βρει όλες, καθώς πολλές μέθοδοι είναι σχεδιασμένες για να βρίσκουν μια και μόνο λύση.
- Πρέπει να εξετασθεί το αν η ευρετική μέθοδος που θα χρησιμοποιηθεί είναι η κατάλληλη για το ανατεθέν πρόβλημα από άποψη ταχύτητας εκτέλεσης. Αναλόγως το πρόβλημα μπορεί μια μέθοδος να είναι σαφώς ταχύτερη μιας άλλης, ενώ για ορισμένες προβλήματα είναι πιθανό κάποιες ευρετικές μέθοδοι να μην είναι παρά ελαφρώς ταχύτερες των ήδη πολύ αργών αναλυτικών μεθόδων επίλυσης.
- Σε προβλήματα με πολλές λύσεις, πρέπει να εξετασθεί αν είναι απαραίτητη η εύρεση της βέλτιστης από αυτές ή αν κρίνονται επαρκείς και άλλες. Εφόσον αναζητείται η βέλτιστη λύση, αξιολογείται το κατά πόσο η ευρετική μέθοδος εγγυάται την εύρεση αυτής.
- Τέλος, αξιολογείται η ευστοχία των προτεινόμενων λύσεων, το αν δηλαδή υπάρχει περιθώριο εμπιστοσύνης για το ότι η λύση προσεγγίζει σε ικανοποιητικό βαθμό κάποια από τις καλύτερες δυνατές λύσεις του προβλήματος.

2.4.3 Μεθευρετικές μέθοδοι επίλυσης

Οι ευρετικές μέθοδοι είναι τεχνικές που εξαρτώνται από το προς επίλυση πρόβλημα. Για το λόγο αυτό είναι συνήθως προσαρμοσμένες στο πρόβλημα αυτό, επιχειρώντας να εκμεταλλευτούν με τον καλύτερο δυνατό τρόπο τις ιδιαιτερότητες του. Ωστόσο, λόγω της τάσης των μεθόδων να εκμεταλλευτούν τη μέχρι τώρα καλύτερη λύση, είναι εύκολο να παγιδευτούν σε τοπικά ακρότατα.

Αντίθετα οι μεθευρετικές μέθοδοι είναι ανεξάρτητες του προβλήματος και μπορούν να χρησιμοποιηθούν για πολλά διαφορετικά προβλήματα λειτουργώντας έτσι σαν μαύρο κουτί (black box). Φυσικά παρότι οι μεθευρετικές τεχνικές είναι γενικά ανεξάρτητες του προβλήματος, να σημειωθεί πως είναι απαραίτητη η ρύθμιση των παραμέτρων της μεθόδου για την προσαρμογή στο προς επίλυση πρόβλημα (Talbi, 2009). Οι μέθοδοι αυτές δεν χαρακτηρίζονται από την απληστία που συχνά παρατηρείται στις ευρετικές μεθόδους, απεναντίας συχνά αποδέχονται μια λύση για το πρόβλημα χειρότερη της προηγούμενης, με σκοπό την καλύτερη εξερεύνηση όσο τη δυνατόν μεγαλύτερου μέρους του χώρου λύσεων.

Οι μεθευρετικές μέθοδοι χαρακτηρίζονται ως υψηλότερου επιπέδου, με την έννοια ότι αναζητούν ή δημιουργούν μια χαμηλότερου επιπέδου ευρετική μέθοδο, η οποία κρίνεται κατάλληλη για την εύρεση λύσεων. Ως επί το πλείστον είναι μέθοδοι στοχαστικές, αφού η προκύπτουσα λύση εξαρτάται από ένα σύνολο τυχαίων μεταβλητών. Έτσι, η εφαρμογή μιας μεθευρετικής μεθόδου στο ίδιο πρόβλημα είναι δυνατόν να παράγει διαφορετική λύση κάθε φορά (Bianchi, Dorigo, Gambardella, & Gutjahr, 2009). Το μεγαλύτερο μέρος της βιβλιογραφίας είναι πειραματικής φύσεως, περιγράφει δηλαδή εμπειρικά αποτελέσματα βασισμένα σε δοκιμές αλγορίθμων σε ηλεκτρονικούς υπολογιστές. Ωστόσο υπάρχουν και ορισμένα θεωρητικά αποτελέσματα, κυρίως όσον αφορά τη σύγκλιση των λύσεων και τη δυνατότητα εύρεσης της ολικά βέλτιστης λύσης (Blum & Roli, 2003).

Οι μεθευρετικές μέθοδοι μπορούν να διακριθούν με βάση τη στρατηγική αναζήτησης που χρησιμοποιούν. Από τη μια υπάρχουν μέθοδοι (π.χ. tabu search) των οποίων η αναζήτηση είναι μια βελτιωμένη εκδοχή των αλγορίθμων τοπικής αναζήτησης. Από την άλλη, άλλες μέθοδοι, όπως η Ant Colony Optimization, εμπεριέχουν κάποια τεχνική εκμάθησης από την προηγούμενη γενιά λύσεων.

Διακρίνονται ακόμη σε μεθόδους μιας λύσης (π.χ. simulated annealing) και σε μεθόδους με πληθυσμό λύσεων.

Κάθε μεθευρετική διαδικασία αναζήτησης καλείται να εξισορροπήσει δύο βασικές συνιστώσες: την εξερεύνηση και την εκμετάλλευση (Crepinsek,

Mernik, Shih\, \#45, & Liu, 2011). Και οι δύο επηρεάζονται από τις παραμέτρους ελέγχου του αλγορίθμου (Tashkova, Šilc, Atanasova, & Džeroski, 2012). Κατά την εξερεύνηση δοκιμάζονται χώροι λύσεων που δεν έχουν εξετασθεί ακόμη, ενώ με την εκμετάλλευση αναζητούνται λύσεις βασισμένες στις ήδη υπάρχουσες, αξιοποιώντας έτσι τις διαθέσιμες μέχρι στιγμής πληροφορίες.

2.4.4 Εξελικτικοί αλγόριθμοι

Η γεωμετρική αύξηση της υπολογιστικής ισχύος και το πιο προσιτό κόστος των μεγάλων και γρήγορων υπολογιστών από τη δεκαετία του 1990 και μετά, συντέλεσαν στην ευρεία αποδοχή και χρήση στοχαστικών μεθόδων βελτιστοποίησης με έναν από τους κυριότερους εκπροσώπους τους εξελικτικούς αλγόριθμους βελτιστοποίησης (Evolutionary Algorithms, EA). Βασικό στοιχείο που οδήγησε στη γρήγορη και ευρεία επικράτησή τους ήταν αφενός το ενδιαφέρον, μη-μαθηματικό υπόβαθρό τους, η ευκολία με την οποία προσαρμόζονται σε κάθε νέο πρόβλημα αρκεί να υπάρχει προγραμματισμένο λογισμικό αξιολόγησης κάθε υποψήφιας λύσης και, κυρίως, η δυνατότητά τους (ως στοχαστική μέθοδος) να μην εγκλωβίζονται σε τοπικά ακρότατα (Michalewicz, 1996). Σε αντίθεση με τις αιτιοκρατικές μεθόδους, σε ένα νέο πρόβλημα η χρήση των εξελικτικών αλγορίθμων είναι γενικά άμεση, χωρίς παρεμβάσεις στη διαδικασία βελτιστοποίησης, κάτι που αποτελεί και το μεγαλύτερο πλεονέκτημα των αλγορίθμων αυτών, αν και αναμφισβήτητα η γνώση του προβλήματος και η χρήση πληροφοριών από αυτό μέσω ειδικών τελεστών αυξάνει την ταχύτητα σύγκλισης. Ωστόσο, βασικό τους μειονέκτημα είναι το γεγονός ότι ο εντοπισμός της βέλτιστης λύσης απαιτεί συνήθως σημαντικό αριθμό αξιολογήσεων ειδικά όταν κάθε αξιολόγηση απαιτεί χρήση χρονοβόρου λογισμικού.

Οι εξελικτικοί αλγόριθμοι έλαβαν την ονομασία τους από την εξέλιξη των ειδών στη φύση, καθώς σε αυτή βασίζουν τη βασική φιλοσοφία στην οποία βασίζεται ο τρόπος λειτουργίας τους. Οι ιδέα των εξελικτικών αλγορίθμων δεν είναι νέα, αφού είχαν προταθεί από τη δεκαετία του 1960 (M. J. Fogel, Owens, & Walsh, 1966). Ωστόσο, τα τελευταία χρόνια η χρήση τους γνωρίζει τεράστια εξάπλωση την τελευταία δεκαετία, χάρη στην εξέλιξη των ηλεκτρονικών υπολογιστών.(L. J. Fogel, 1999)

Βασικό γνώρισμα των εξελικτικών αλγορίθμων είναι πως δεν χειρίζονται μια μεμονωμένη λύση σε κάθε τους επανάληψη όπως άλλες στοχαστικές μέθοδοι (π.χ. simulated annealing), αλλά έναν ολόκληρο πληθυσμό υποψηφίων λύσεων. Οι εξελικτικοί αλγόριθμοι παρότι αρχικά αναπτύχθηκαν για την επίλυση προβλημάτων ενός στόχου, είναι δυνατόν να αντιμετωπίσουν και

προβλήματα πολλών στόχων με τις κατάλληλες μετατροπές. Μάλιστα υπερτερούν των άλλων μεθόδων λόγω του ότι χειρίζονται έναν ολόκληρο πληθυσμό αποδίδοντας έτσι ένα σύνολο λύσεων με την ολοκλήρωσή τους.

Τα μέλη του πληθυσμού υφίστανται αλλαγές σε κάθε κύκλο ανανέωσης του πληθυσμού. Τα υπόλοιπα μέλη του πληθυσμού αλληλεπιδρούν μεταξύ τους, με τρόπο διαφορετικό ανάλογα τον υπό χρήση αλγόριθμο. Έτσι οι πιθανές λύσεις «δανείζονται» στοιχεία από τις υπόλοιπες, με έμφαση στις καλύτερες μέχρι στιγμής λύσεις.

2.4.4.1 Γενετικός Αλγόριθμος (Genetic Algorithm – GA)

Ο γενετικός αλγόριθμος αποτελεί έναν από τους παλαιότερους και πιθανότατα τον πιο πολυχρησιμοποιημένο σε προβλήματα βελτιστοποίησης εξελικτικό αλγόριθμο.

Χρησιμοποιεί τεχνικές επιλογής (selection), διασταύρωσης (crossover) και μετάλλαξης (mutation) εμπνευσμένες από τη φύση.

Στη φύση, διάφοροι παράγοντες μπορούν να προκαλέσουν μεταλλάξεις, πιο συνηθισμένα λόγω κάποιου λάθους στην αντιγραφή του κυττάρου. Έτσι και στον Γενετικό Αλγόριθμο, για κάθε λύση υπάρχει μια πιθανότητα μετάλλαξης από γενιά σε γενιά. Στην δυαδική κωδικοποίηση η διαδικασία της μετάλλαξης πραγματοποιείται πολύ απλά, αλλάζοντας (με μικρές πιθανότητες να συμβεί αυτό – τυπικά της τάξης του Pm=0.001 (Coley, 1998) κάποια τυχαία 0 σε 1 ή το αντίστροφο. Οι πιθανότητες μετάλλαξης βέβαια εξαρτώνται από το προς επίλυση πρόβλημα.

Ο τελεστής επιλογής καθορίζει τα μέλη του πληθυσμού που θα χρησιμοποιηθούν σαν γονείς, ώστε να δημιουργηθεί ο επόμενος πληθυσμός. Συχνά το 50% του πληθυσμού επιλέγεται, ενώ το άλλο 50% δεν συμμετέχει στη δημιουργία του νέου πληθυσμού, το ποσοστό όμως μπορεί να διαφοροποιηθεί για τις ανάγκες του εκάστοτε προβλήματος (Coley, 1998). Η επιλογή συνήθως είναι αναλογική της καταλληλότητας (fitness-proportional), δηλαδή όσο καλύτερη τιμή καταλληλότητας-fitness έχει ένα μέλος του πληθυσμού αυξάνονται οι πιθανότητες να επιλεγεί σαν γονιός.

Αφού επιλεχθούν τα ζεύγη των γονέων πραγματοποιείται η διασταύρωση τους, και δημιουργούνται τα παιδιά-μέλη του νέου πληθυσμού. Στη δυαδική κωδικοποίηση αυτά προκύπτουν λαμβάνοντας το πρώτο κομμάτι της δυαδικής αλληλουχίας από τον ένα γονιό και το δεύτερο από τον άλλο γονιό.

Στον αλγόριθμο αυτό εισάγεται και η έννοια του ελιτισμού (elitism), πρακτική η οποία μπορεί να αποδειχθεί χρήσιμη και σε άλλους εξελικτικούς αλγορίθμους.

Σύμφωνα με αυτή, ένας προκαθορισμένος αριθμός μελών του πληθυσμού – πιθανές λύσεις, και συγκεκριμένα αυτές με την καλύτερη τιμή καταλληλότητας-fitness,αντιγράφονται και τα αντίγραφα περνούν αυτούσια στην επόμενη γενιά χωρίς να υποστούν αλλαγές.

2.4.4.2 Βελτιστοποίηση Σμήνους Σωματιδίων (Particle Swarm Optimization)

Είναι σημαντικό να γίνει ιδιαίτερη αναφορά στον εξελικτικό αλγόριθμο Βελτιστοποίησης Σμήνους Σωματιδίων (Particle Swarm Optimization-PSO), καθώς αποτελεί τον πλέον συγγενικό αλγόριθμο του αλγορίθμου πυγολαμπίδων, ο οποίος χρησιμοποιήθηκε στα πλαίσια της διπλωματικής εργασίας αυτής. Ταυτόχρονα, ο Particle Swarm Optimization, ως αλγόριθμος παλαιότερος του ΑΠ διαθέτει μεγαλύτερη βιβλιογραφία η οποία μπορεί να χρησιμοποιηθεί ως σημείο αναφοράς, καθώς έχει χρησιμοποιηθεί σε αρκετά προβλήματα βελτιστοποίησης.

Αναπτύχθηκε από τους Kennedy και Eberhart (1995) και έχει τη δυνατότητα αναζήτησης πιθανών λύσεων σε πολύ μεγάλους χώρους λύσεων.

Στην κοινή του χρήση ο όρος σμήνος (swarm) αναφέρεται σε μια ανοργάνωτη ομάδα κινούμενων αντικειμένων, κυρίως εντόμων που κινούνται με τρόπο ακανόνιστο, χαοτικό, παραμένοντας κοντά μεταξύ τους παρότι κινούνται με τυχαίο τρόπο. Ωστόσο πρέπει να σημειωθεί ότι στη βελτιστοποίηση Particle Swarm, οι κινήσεις δεν είναι χαοτικές.

Στον αλγόριθμο αυτό, όταν ο πληθυσμός σωματιδίων (particles) αρχικοποιείται, πέρα από τις τυχαίες τιμές που λαμβάνουν οι μεταβλητές, ανατίθεται στοχαστικά και μια ταχύτητα σε κάθε σωματίδιο-particle. Σε κάθε επανάληψη, η ταχύτητα κάθε σωματιδίου επιταχύνεται στοχαστικά προς την προηγούμενη καλύτερη θέση του σωματιδίου αυτού (εκεί όπου η είχε την καλύτερη τιμή αντικειμενικής συνάρτησης (καταλληλότητα – fitness value) καθώς και προς την θέση του καλύτερου σωματιδίου της συγκεκριμένης περιοχής λύσεων (την θέση του σωματιδίου με την καλύτερη τιμή καταλληλότητας - fitness) (James Kennedy & Eberhart, 2001).

Πλεονέκτημα της βελτιστοποίησης με Particle Swarm είναι η δυσκολία του να παγιδευτούν πιθανές λύσεις-σωματίδια σε τοπικά ακρότατα, χάρη στη χρήση των περιοχών λύσεων και αποστάσεων μεταξύ των σωματιδίων αντί του ολικού μεγίστου ή ελαχίστου για τον καθορισμό της κίνησης του σωματιδίου. Ο τρόπος λειτουργίας της είναι εύκολος στην κατανόηση αλλά και στην υλοποίηση, προσφέροντας ταυτόχρονα ικανοποιητικά αποτελέσματα.

2.5 Ο Εξελικτικός Αλγόριθμος Πυγολαμπίδας - Firefly Algorithm (FA)

Η νοημοσύνη σμήνους ως κλάδος τεχνητής νοημοσύνης γίνεται όλο και περισσότερο δημοφιλής την τελευταία δεκαετία (Blum & Li, 2008). Πηγή έμπνευσής της είναι η συλλογική συμπεριφορά των σμηνών κοινωνικών εντόμων όπως τα μυρμήγκια και οι μέλισσες, αλλά και των πτηνών. Παρότι, μεμονωμένα, τα μέλη των σμηνών αυτά δεν επιδεικνύουν κάποιες ιδιαίτερες δεξιότητες, παρουσιάζουν συντονισμένες συμπεριφορές οι οποίες κατευθύνουν το σμήνος στους επιθυμητούς του στόχους. Για παράδειγμα τα μυρμήγκια και οι τερμίτες χτίζουν προηγμένες φωλιές, ενώ τα μυρμήγκια και οι μέλισσες ακολουθούν συλλογική συμπεριφορά και στην αναζήτηση τροφής. Στην περίπτωση των μελισσών το ρόλου του πληροφοριοδότη παίζουν οι αποκαλούμενοι ανιχνευτές, μέλισσες οι οποίες αναζητούν νέες πηγές τροφής, και καθοδηγούν τις μέλισσες σε αυτές. Η αποικία των μελισσών καλείται να εξισορροπήσει τις προσπάθειες της ανάμεσα στην αναζήτηση νέων πηγών τροφής και στην εκμετάλλευση των ήδη υπαρχόντων.

Η νοημοσύνη σμήνους εφαρμόζεται σε πεδία έρευνας που αφορούν αυτό-οργανωμένα και αποκεντρωμένα συστήματα, όπως για παράδειγμα στη δρομολόγηση και την εξισορρόπηση φορτίου στα δίκτυα τηλεπικοινωνιών.

Ο εξελικτικός αλγόριθμος Firefly αναπτύχθηκε από τον Xin-She Yang στα τέλη του 2007 και το 2008 στο πανεπιστήμιο του Cambridge (X.-S. Yang, 2009, 2014a; X. S. Yang, 2008) βασιζόμενος στο μοτίβο που ακολουθεί η λάμψη των πυγολαμπίδων (fireflies) και στην συμπεριφορά αυτών. Η βιβλιογραφία σχετικά με τον νέο αυτό αλγόριθμο έχει επεκταθεί σημαντικά τα τελευταία 6 χρόνια, με τη δημοσίευση εκατοντάδων σχετικών επιστημονικών κειμένων, για τα οποία οι Fister et al προσφέρουν μια κατατοπιστική επισκόπηση (Fister, JrFister, Yang, & Brest, 2013).

Στην ουσία, ο αλγόριθμος πυγολαμπίδων χρησιμοποιεί τρεις εξιδανικευμένους κανόνες:

1. Οι πυγολαμπίδες-fireflies δεν έχουν φύλο, που σημαίνει πως καθεμία μπορεί να προσελκυσθεί από οποιαδήποτε άλλη.
2. Η ελκυστικότητά τους είναι ανάλογη της λάμψης τους. Και τα δύο μειώνονται όσο μεγαλώνει η απόσταση μεταξύ των πυγολαμπίδων. Μεταξύ δύο πυγολαμπίδων, η λιγότερο λαμπερή θα κινηθεί προς την περισσότερο λαμπερή. Αν δεν υπάρχει καμία περισσότερο λαμπερή από μια συγκεκριμένη πυγολαμπίδα, τότε αυτή θα κινηθεί με τυχαίο τρόπο.
3. Η λαμπρότητα μιας πυγολαμπίδας καθορίζεται από την αντικειμενική συνάρτηση.

2.5.1 Μαθηματική έκφραση του Αλγορίθμου Πυγολαμπίδας για συνεχείς μεταβλητές

Καθώς η ελκυστικότητα μίας πυγολαμπίδας είναι ανάλογη της έντασης του φωτός όπως τον αντιλαμβάνονται οι γύρω πυγολαμπίδες, η ελκυστικότητα β μπορεί να εκφραστεί συναρτήσει της απόστασης r ως εξής:

$$\beta = \beta_0 e^{-\gamma r^2}$$

όπου β_0 η ελκυστικότητα για απόσταση r=0.

Η κίνηση της πυγολαμπίδας i , η οποία ελκύεται από μια άλλη, πιο λαμπερή-ελκυστική πυγολαμπίδα j δίνεται από τη σχέση:

$$x_i^{t+1} = x_i^t + \beta_0 e^{-\gamma r_{ij}^2}\left(x_j^t - x_i^t\right) + a_t \varepsilon_i^t$$

Ο πρώτος όρος του δευτέρου μέλους της σχέσης εκφράζει την αρχική θέση της i πριν την κίνησή της. Ο δεύτερος τη συνιστώσα της κίνησης που οφείλεται στην έλξη, ενώ ο τρίτος όρος εκφράζει την τυχαιοποίηση. α_t είναι η παράμετρος τυχαιοποίησης, ενώ το ε_i^t είναι ένα διάνυσμα αριθμών τυχαία επιλεγμένων από μια κατανομή Gauss ή μια διακριτή ομοιόμορφη κατανομή τη χρονική στιγμή t. Για β_0=0 η κίνηση γίνεται τυχαίος περίπατος (random walk) ενώ για γ=0 ο firefly algorithm μετατρέπεται σε μια παραλλαγή της Particle Swarm Optimization (X. S. Yang, 2008). Ακόμη, το διάνυσμα τυχαιοποίησης ε_i^t μπορεί να επεκταθεί και για άλλες κατανομές π.χ. Levy Flights (X. S. Yang, 2008).

Στον Αλγόριθμο Πυγολαμπίδων, η ελκυστικότητα (και η ένταση του φωτός) συνδέεται εγγενώς τόσο με την φυσική έννοια του νόμου της έντασης του φωτός, η οποία είναι αντιστρόφως ανάλογη της απόστασης από την πηγή του, όσο και με τον συντελεστή απορρόφησης γ, ο οποίος είναι πάντοτε μεγαλύτερος ή ίσος του 0. Προκύπτει έτσι ο μη γραμμικός όρος ελκυστικότητας $\beta_0 \exp[-\gamma r^2]$ (X. S. Yang, 2008).

Παράγοντες αντιστρόφως ανάλογοι του τετραγώνου της απόστασης έχουν χρησιμοποιηθεί και σε άλλους αλγορίθμους εμπνευσμένους από τη φύση. Για παράδειγμα ο charged system search (CSS) χρησιμοποιεί το νόμο του Coulomb, ενώ ο αλγόριθμος Gravitational Search (GSA) χρησιμοποιεί τον νόμο της βαρύτητας του Νεύτωνα.

Σκοπός του να χρησιμοποιείται τέτοιου είδους έλξη είναι να διευκολυνθεί η όσο το δυνατόν πιο επιτυχημένη και ταχεία σύγκλιση των πιθανών λύσεων. Καθώς ο πληθυσμός εξελίσσεται, αλληλεπιδρά και ελκύεται, γίνεται ολοένα και πιο πιθανό εκείνα τα μέλη του πληθυσμού που προκαλούν την έλξη στα υπόλοιπα, να βρίσκεται κοντά στην ολικά βέλτιστη κατάσταση-ακρότατο.

2.5.2 Τι κάνει τον Αλγόριθμο Πυγολαμπίδας αποτελεσματικό

Καθώς η σχετική με τον αλγόριθμο βιβλιογραφία επεκτείνεται και νέες παραλλαγές του κάνουν την εμφάνισή τους, αποδεικνύεται ιδιαίτερα αποδοτικός στην επίλυση πολλών προβλημάτων.

Καθώς ο ΑΠ βασίζεται στη νοημοσύνη σμήνους, διαθέτει παρόμοια πλεονεκτήματα με όλους τους αλγορίθμους που βασίζονται σε αυτή. Στην πραγματικότητα, αναλύοντας τις παραμέτρους γίνεται φανερό ότι ορισμένες παραλλαγές της βελτιστοποίησης Particle Swarm όπως η επιταχυμένη - Accelerated PSO (X.-S. Yang, Deb, & Fong, 2011) αποτελούν ειδική περίπτωση του αλγορίθμου πυγολαμπίδων για γ=0 (X. S. Yang, 2008).

Ωστόσο ο ΑΠ διαθέτει δύο ξεχωριστά πλεονεκτήματα έναντι των υπολοίπων αλγορίθμων. Πρώτον, προσφέρει αυτόματο διαχωρισμό του πληθυσμού σε ομάδες. Ο αλγόριθμος βασίζεται στην ελκυστικότητα, η οποία μειώνεται όσο η απόσταση αυξάνεται. Αυτό αποδεικνύεται ιδιαίτερα χρήσιμο για την ταυτόχρονη αναζήτηση και αξιολόγηση διαφορετικών τοπικών ακροτάτων Δεύτερον, αυτός ο διαχωρισμός σε υπο-ομάδες μπορεί να αποδειχθεί ιδιαίτερα χρήσιμος σε προβλήματα στα οποία μας ενδιαφέρει το πλήθος των ακροτάτων, και όχι μονάχα η μέγιστη ή η ελάχιστη τιμή. Αυτό προϋποθέτει βέβαια το μέγεθος του πληθυσμού να είναι σημαντικά μεγαλύτερο του αριθμού των ακροτάτων. Για παράδειγμα, σε ένα πρόβλημα στο οποίο υπάρχουν πολλαπλά τοπικά ακρότατα με κοντινές μεταξύ τους τιμές της αντικειμενικής συνάρτησης, ο ΑΠ παρέχει την δυνατότητα εύρεσης πολλών από αυτά και στη συνέχεια ο χρήστης έχει τη δυνατότητα να τα συγκρίνει και ενδεχομένως να επιλέξει εκείνο που είναι προτιμότερο για εκείνον.

Παρά την αποτελεσματικότητα που παρουσιάζουν οι αλγόριθμοι νοημοσύνης σμήνους όπως ο αλγόριθμος πυγολαμπίδων, ένα ζήτημα που προκύπτει είναι η απόσταση που χωρίζει την πράξη από την θεωρία. Όπως συμβαίνει στους περισσότερους μεθευρετικούς αλγόριθμους, παρότι οι εφαρμογές τους είναι συχνά επιτυχημένες, το θεωρητικό τους υπόβαθρο και η μαθηματική τους ανάλυση βρίσκεται πολλά βήματα πίσω. Έτσι, παρότι είναι γνωστό ότι μπορούν να δουλέψουν αποδοτικά στην πράξη, οι λόγοι για τους οποίους συμβαίνει αυτό δεν είναι πάντοτε σαφείς, κάτι που καθιστά δύσκολη τη βελτίωση τους αφού οι μηχανισμοί λειτουργίας τους δεν είναι πλήρως κατανοητοί.

Ένα ακόμα σημαντικό ζήτημα αφορά την παραμετροποίηση των μεθευρετικών αλγορίθμων. Οι επιλογή των τιμών των παραμέτρων επηρεάζει σημαντικά το τελικό αποτέλεσμα, ενώ οι κατάλληλες τιμές εξαρτώνται και από το προς επίλυση πρόβλημα. Έτσι η επιλογή των παραμέτρων αποτελεί από μόνη της ένα πρόβλημα βελτιστοποίησης. Η ρύθμιση παραμέτρων αποτελεί

από μόνη της ένα σημαντικό αντικείμενο έρευνας (Eiben & Smit, 2011), το οποίο αξίζει περισσότερη προσοχή.

Επιπρόσθετα, παρότι υπάρχει πληθώρα επιτυχημένων εφαρμογών πολλών αλγορίθμων, στις περισσότερες περιπτώσεις τα προβλήματα προς επίλυση εμπεριέχουν ως επί το πλείστον έως λίγες εκατοντάδες μεταβλητές σχεδιασμού. Για την πρόοδο σε εφαρμογές που αφορούν τον πραγματικό κόσμο, θα ήταν χρήσιμη η δυνατότητα αύξησης του αριθμού των μεταβλητών στην τάξη των αρκετών χιλιάδων (X.-S. Yang, 2014a).

2.5.3 Χρήσεις και εφαρμογές του Αλγορίθμου Πυγολαμπίδας

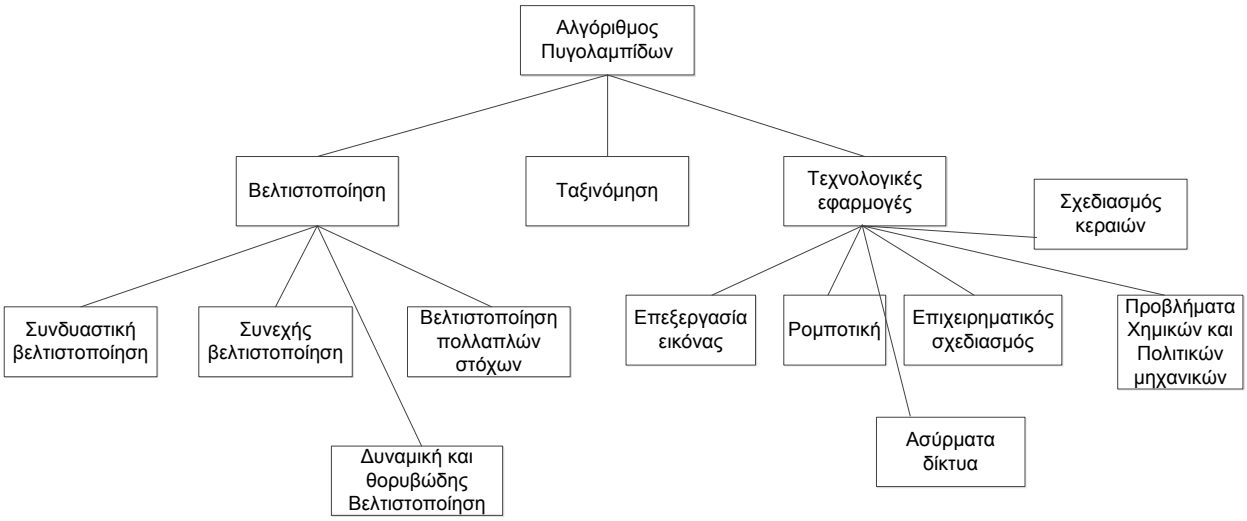
Ο Αλγόριθμος Πυγολαμπίδων έχει χρησιμοποιηθεί για την επίλυση πολλών προβλημάτων συνεχούς βελτιστοποίησης (Xin, #45, & Yang, 2010, 2011; X.-S. Yang, 2011). Στις περισσότερες περιπτώσεις, χρησιμοποιήθηκαν σημεία αναφοράς γνωστών συναρτήσεων βελτιστοποίησης. Ο ΑΠ αποδεικνύεται ιδιαίτερα αποτελεσματικός σε τέτοιου είδους προβλήματα συγκριτικά με άλλους μεθευρετικούς αλγορίθμους όπως ο γενετικός ή ο Simulated Annealing.

Έχει χρησιμοποιηθεί ακόμα για την επίλυση του προβλήματος task graph scheduling, πρόβλημα το οποίο είναι NP-hard, καθώς και για το Travelling salesman Problem. Στη δημοσίευση του Kwiecien (Kwiecień & Filipowicz, 2012) εφαρμόσθηκε ο Αλγόριθμος Πυγολαμπίδας για τη βελτιστοποίηση ουρών αναμονής.

Οι Sayadi et al.(2010) παρουσίασαν μια μεθευρετική αναζήτηση βασισμένη στον ΑΠ για την ελαχιστοποίηση του χρόνου ολοκλήρωσης των εργασιών στο flow shop scheduling problem με προκαθορισμένη σειρά έναρξης εργασιών. Τα αποτελέσματα πάνω σε προβλήματα αναφοράς αποδείχθηκαν καλύτερα σε σύγκρισή με υπάρχουσες λύσεις που έχουν προκύψει με χρήση της βελτιστοποίησης αποικίας μυρμηγκιών (ant colony).

Όσον αφορά τις τεχνολογικές εφαρμογές του αλγορίθμου, έχει χρησιμοποιηθεί (συνοπτικά) :

- Στη βιομηχανική βελτιστοποίηση (Zamuda & Brest, 2012)
- Στην επεξεργασία εικόνας (Hassanzadeh, Vojodi, & Mahmoudi, 2011; Horng, 2012)
- Στον σχεδιασμό κεραιών και προβόλων (Chatterjee, Mahanti, & Chatterjee, 2012)
- Στη ρομποτική (Jakimovski, Meyer, & Maehle, 2010)



Σχήμα 3: Εφαρμογές του Αλγορίθμου Πυγολαμπίδας (Fister et al., 2013)

2.5.3.1 Εφαρμογή Αλγορίθμου Πυγολαμπίδας για συνεχείς μεταβλητές

Για ένα πρόβλημα μεγιστοποίησης, η λαμπρότητα μιας πυγολαμπίδας μπορεί απλά να είναι αναλογική της αντικειμενικής συνάρτησης του προβλήματος. Άλλες μορφές λαμπρότητας μπορούν να ορισθούν με τρόπο αντίστοιχο της συνάρτησης fitness των γενετικών αλγορίθμων ή του bacterial foraging algorithm (BFA) (Gazi & Passino, 2004)

Ακολουθεί ο ψευδοκώδικας του Αλγορίθμου Πυγολαμπίδας, όπως αυτός χρησιμοποιείται για την αναζήτηση ακροτάτων σε συνεχείς συναρτήσεις(X.-S. Yang, 2010)

Αλγόριθμος Πυγολαμπίδα
Αντικειμενική συνάρτηση f(x), x = (x1, . . . , xd)T
Αρχικοποίησε πληθυσμό πυγολαμπίδων xi (i = 1, 2, . . . , n)
Όρισε το συντελεστή απορρόφησης φωτός
While (t <ΜέγιστηΓενιά)
for i = 1 : n ολες οι n πυγολαμπίδες
for j = 1 : i όλες οι n πυγολαμπίδες
Η ένταση φωτός Ii στο xi ορίζεται από την τιμή f(xi)
Αν (Ij > Ii)
Μετακίνησε την πυγολαμπίδα i προς την j σε όλες τις d διαστάσεις
Τέλος αν
Η ελκυστικότητα διαφοροποιείται συναρτήσει της αποστ. r με τάξηexp[−r]
Αξιολόγησε της νέες λύσεις και ανανέωσε τις εντάσεις του φωτός
Τέλος for j
Τέλος for i
Ταξινόμησε τις πυγολαμπίδες και βρες την καλύτερη της γενιάς
Τέλος while
Επεξεργασία αποτελεσμάτων και γραφική αναπαράσταση

Στα προβλήματα συνεχών μεταβλητών, η απόσταση μεταξύ δύο πυγολαμπίδων μπορεί να οριστεί ως η ευκλείδεια-καρτεστιανή απόσταση μεταξύ τους, βάσει των συντεταγμένων τους. Σε προβλήματα δύο ή τριών διαστάσεων, η απόσταση αυτή είναι η απόσταση στο επίπεδο ή στον τρισδιάστατο χώρο αντίστοιχα, ενώ για προβλήματα περισσοτέρων διαστάσεων, παρότι η απεικόνιση της απόστασης γίνεται δυσκολότερη, ισχύουν και πάλι οι ίδιοι κανόνες για τον προσδιορισμό της στον ευκλείδειο-καρτεσιανό χώρο. Όπως θα διαπιστωθεί στη συνέχεια, αυτός ο ορισμός της απόστασης δεν μπορεί να χρησιμοποιηθεί για προβλήματα scheduling.

Ο παράγοντας γ ουσιαστικά καθορίζει τη διαφοροποίηση της ελκυστικότητας. Η επιλογή της τιμής του γ είναι καθοριστικής σημασίας για την ταχύτητα συγκέντρωσης των λύσεων στα ακρότατα του προβλήματος και για την συμπεριφορά του ΑΠ γενικότερα. Θεωρητικά, ισχύει $\gamma \in [0, \infty]$, ωστόσο στην πράξη για την τάξη μεγέθους του γ είναι $\gamma = O(1)$ και η ακριβής τιμή του προσδιορίζεται με βάση το μέγεθος του προς βελτιστοποίηση συστήματος.

2.5.3.1 Αριθμητικά παραδείγματα

Δεδομένου του παραπάνω ψευδοκώδικα μπορεί να γίνει άμεση εφαρμογή σε συναρτήσεις συνεχών μεταβλητών.

Ακολουθούν τα αποτελέσματα της εφαρμογής του αλγορίθμου σε MATLAB, πάνω σε συναρτήσεις αναφοράς.(X.-S. Yang, 2010)

Συνάρτηση Ackley:

$$f(\mathbf{x}) = -20\exp\Big[-\frac{1}{5}\sqrt{\frac{1}{d}\sum_{i=1}^{d}x_i^2}\Big]$$

$$-\exp[\frac{1}{d}\sum_{i=1}^{d}\cos(2\pi x_i)] + 20 + e,$$

Η συνάρτηση αυτή έχει ένα ολικό ελάχιστο $f_0 = 0$ στο (0,0,…,0). Έπειτα από μελέτη της παραμετροποίησης, ο Yang κατέληξε στις εξής τιμές παραμέτρων για την εύρεση ακροτάτων των δοκιμαστικών συναρτήσεων: α=0.2, γ=1, β_0=1 (X.-S. Yang, 2010)
Για παράδειγμα, χρησιμοποιήθηκε ο ΑΠ για την εύρεση μεγίστων της παρακάτω συνάρτησης:

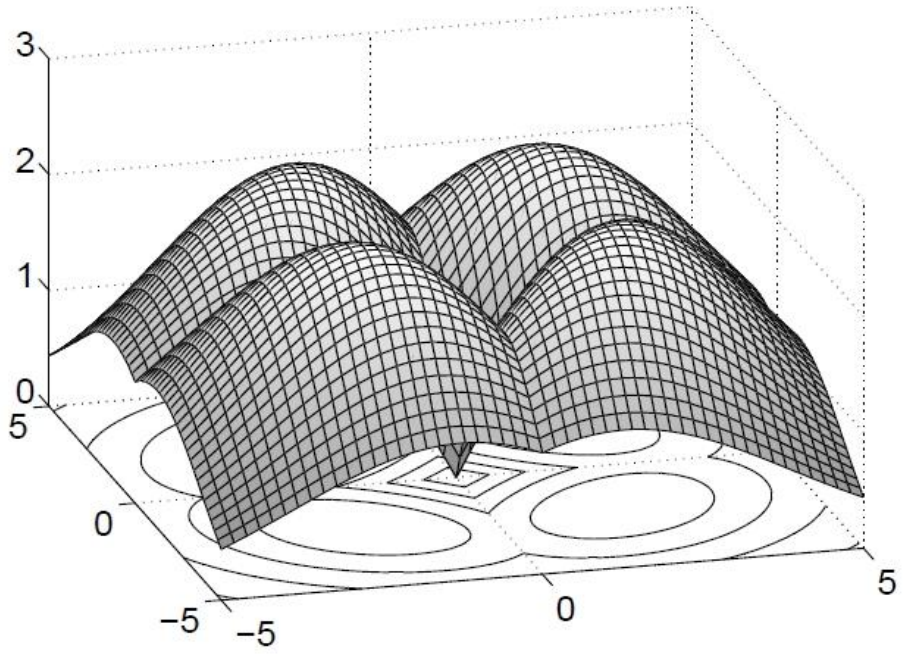
$$f(\mathbf{x}) = \Big(\sum_{i=1}^{d}|x_i|\Big)\exp\Big(-\sum_{i=1}^{d}x_i^2\Big) \quad (1)$$

Με $-10 \le x_i \le 10 \ \ για\ κάθε\ (i = 1,2,\dots,d)$ όπου d ο αριθμός των διαστάσεων. Η συνάρτηση αυτή έχει πολλαπλά ολικά μέγιστα. Για d=2 εμφανίζονται 4 ίσα μεταξύ τους ολικά μέγιστα με τιμή
$f_* = \frac{1}{\sqrt{e}} \approx 0.6065\ στα\ σημεία\ \left(\frac{1}{2},\frac{1}{2}\right),\left(\frac{1}{2},-\frac{1}{2}\right),\left(-\frac{1}{2},\frac{1}{2}\right),\left(-\frac{1}{2},-\frac{1}{2}\right)$

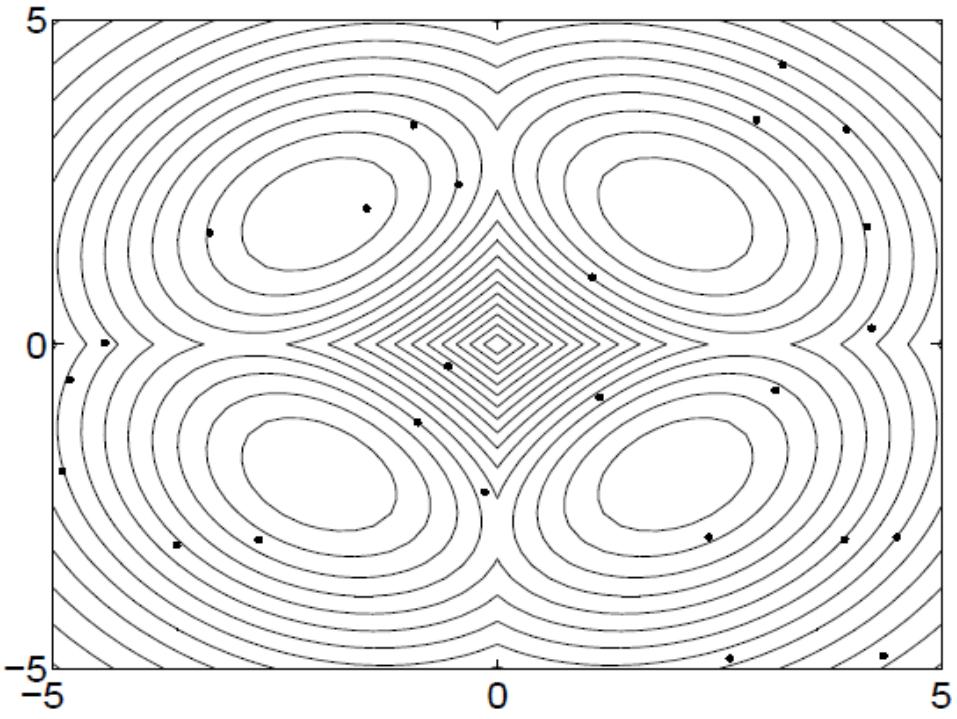
καθώς και ένα μοναδικό ολικό ελάχιστο με τιμή 0 στο σημείο (0,0)

Τα 4 ολικά μέγιστα και το ολικό ελάχιστο αναπαρίστανται γραφικά στο παρακάτω γράφημα, όπως αυτά βρέθηκαν με χρήση του Αλγορίθμου Πυγολαμπίδας μετά από 500 αξιολογήσεις της συνάρτησης. Οι 500 αξιολογήσεις αντιστοιχούν σε 25 πυγολαμπίδες οι οποίες εξελίσσονται για 20 γενιές (επαναλήψεις). Οι αρχικές θέσεις των πυγολαμπίδων φαίνονται στο σχήμα [4] ενώ οι τελικές στο σχήμα [5]. Γίνεται φανερό πως ο ΑΠ είναι αποτελεσματικός για τέτοιου είδους προβλήματα, ενώ πρόσφατες μελέτες

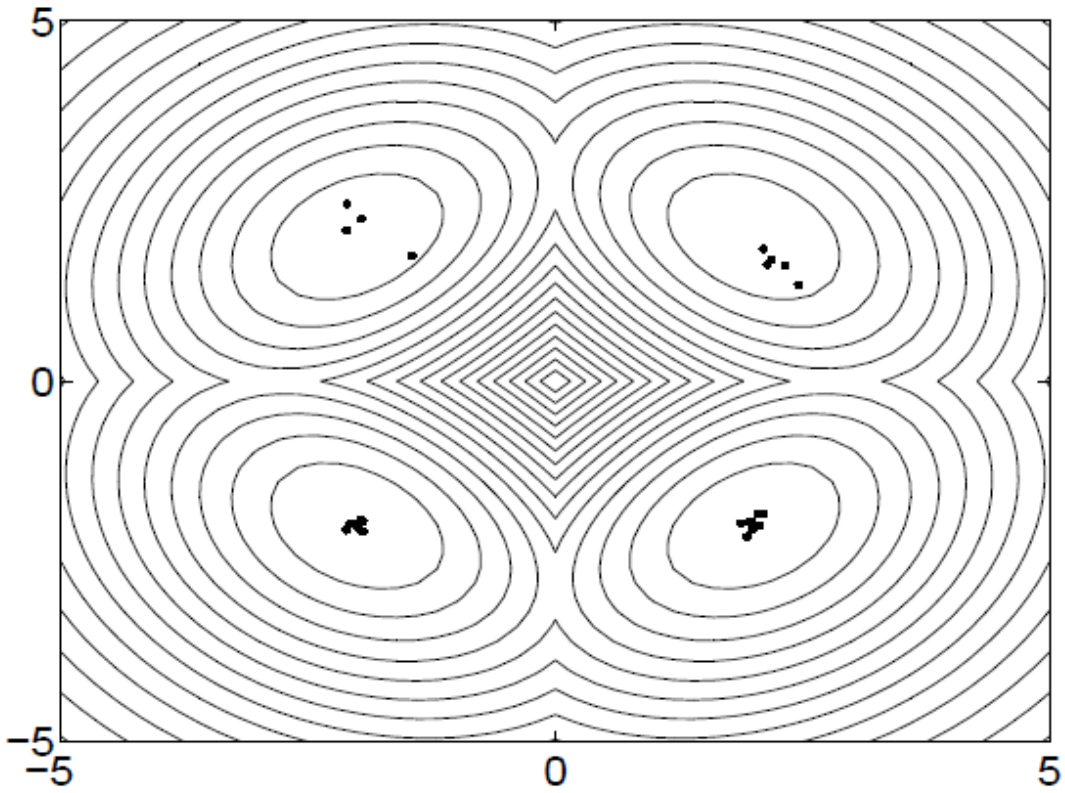
αποδεικνύονται ενθαρρυντικές για τη χρήση του ΑΠ σε προβλήματα μη γραμμικής βελτιστοποίησης με περιορισμούς (Łukasik & Żak, 2009).



Σχήμα 4: Γραφική παράσταση της συνάρτησης (1)



Σχήμα 5: Αρχική θέση των μελών πληθυσμού για τη συνάρτηση (1)



Σχήμα 6: Τελική θέση των μελών του πληθυσμού για τη συνάρτηση (1) μετά από 20 επαναλήψεις

2.5.3.2 Παράδειγμα εφαρμογής του Αλγορίθμου Πυγολαμπίδας για δοχεία πίεσης

Για την ανάδειξη της δυνατότητας εφαρμογής του ΑΠ σε πρακτικά προβλήματα βελτιστοποίησης στον πραγματικό κόσμο, ακολουθεί ένα παράδειγμα στο οποίο χρησιμοποιείται ο ΑΠ για την επίλυση του προβλήματος βέλτιστου σχεδιασμού ενός δοχείου πίεσης.(X.-S. Yang, 2010)

Τα ντεπόζιτα καυσίμων και τα μπουκάλια σαμπάνιας αποτελούν περιπτώσεις δοχείων πίεσης στον πραγματικό κόσμο. Με δοσμένο τον όγκο και την μέγιστη επιτρεπτή τιμή της πίεσης, ο βασικός στόχος κατά τον σχεδιασμό ενός κυλινδρικού δοχείου είναι η ελαχιστοποίηση του συνολικού κόστους. Στις περισσότερες περιπτώσεις οι μεταβλητές σχεδιασμού του προβλήματος είναι το πάχος d_1 της κεφαλής του δοχείου, το πάχος d_2 του σώματος του δοχείου, η εσωτερική ακτίνα r και το μήκος L του κυλινδρικού τμήματος (Coello Coello, 2000), (Cagnina, Esquivel, & Coello, 2008). Το γνωστό αυτό πρόβλημα βελτιστοποίησης μπορεί να διατυπωθεί μαθηματικά ως εξής:

$$ελαχιστοποίησε\ f(x) = 0.6224d_1rL + 1.7781d_2r^2 + 3.1661d_1^2L + 19.84d_1^2r,$$

όπου f το κόστος,

υπό τους παρακάτω περιορισμούς

$$g_1(x) = -d_1 + 0.0193r \leq 0$$

$$g_2(x) = -d_2 + 0.00954r \leq 0$$

$$g_3(x) = -\pi r^2 L - \frac{4\pi}{3}r^3 + 1296000 \leq 0$$

$$g_4(x) = L - 240 \leq 0$$

Πρέπει να ικανοποιούνται επίσης τα όρια

$$0.0625 \leq d_1, d_2 \leq 99 \times 0.0625$$

και

$$10.0 \leq r,\ \ L \leq 200.0$$

Το 2008, βρέθηκε από τους Cagnina et al με χρήση της βελτιστοποίησης particle swarm καλύτερη λύση για το πρόβλημα η

$$f_* = 6059.714$$

στο σημείο

$$x_* = (0.8125,\ \ 0.4375,\ \ 42.0984,\ \ 176.6366).$$

Η ελάχιστη δηλαδή τιμή κόστους για δοχείο που να ικανοποιεί τους παραπάνω περιορισμούς είναι 6059.71$.

Για το ίδιο πρόβλημα, με χρήση του ΑΠ με πληθυσμό 40 πυγολαμπίδες και μετά από 20 επαναλήψεις, βρέθηκε καλύτερη λύση (Xin et al., 2010), στο σημείο

$$x_* = (0.7782,\ \ 0.3846,\ \ 30.3196,\ \ \ 200.0)$$

με τιμή κόστους

$$f_* = 5885.33$$

η οποία είναι σημαντικά καλύτερη από εκείνη των Cagnina et al (2008).

Το παράδειγμα αυτό είναι ενδεικτικό του πόσο αποτελεσματικός μπορεί να αποδειχθεί ο Αλγόριθμος Πυγολαμπίδων. Είναι βέβαια απαραίτητη η περαιτέρω διερεύνηση της συμπεριφοράς και της αποτελεσματικότητας του αλγορίθμου σε διαφορετικής φύσεως προβλήματα βελτιστοποίησης.

2.6 Επιλογή εξελικτικού αλγορίθμου – No Free Lunch

Δεδομένης της ύπαρξης πολλών διαφορετικών αλγορίθμων αναζήτησης σήμερα, τίθεται το ερώτημα της ύπαρξης αλγορίθμου που να υπερτερεί των υπολοίπων γενικά, ή για ένα συγκεκριμένο πρόβλημα. Το να δοθεί απάντηση σε αυτό το ερώτημα αποδεικνύεται ιδιαίτερα δύσκολο.

Το θεώρημα “No Free Lunch” των David Wolpert και William Macready (1997) προκάλεσε πολλές συζητήσεις και αντιπαραθέσεις.

Υποστηρίζει ότι για κάθε ζεύγος μεθόδων αναζήτησης A,B , εφόσον π.χ. η A υπερτερεί της B όσον αφορά τη βελτιστοποίηση με βάση ένα κριτήριο για ένα είδος προβλημάτων, τότε αναπόφευκτα η B θα υπερτερεί της A για τη βελτιστοποίηση με βάση κάποιο διαφορετικό κριτήριο ή για κάποιο άλλο είδος προβλήματος ή παραλλαγή του ίδιου προβλήματος (*Evolutionary Algorithms in Engineering and Computer Science: Recent Advances in Genetic Algorithms, Evolution Strategies, Evolutionary Programming, GE*, 1999; James Kennedy & Eberhart, 2001). Για παράδειγμα, έστω πως έχουμε δύο μεθόδους αναζήτησης. Η μια αναζητά ακολουθώντας την παράγωγο, ενώ η άλλη αναζητά μεταπηδώντας τυχαία στον χώρο λύσεων. Έστω πως έχουμε δύο προβλήματα, το ένα με ένα ακρότατο σε μορφή ημισφαιρίου και το άλλο με πολλά διαφορετικά τοπικά ακρότατα, ορισμένα μεγαλύτερα σε τιμή και ορισμένα μικρότερα. Είναι προφανές ότι καθεμία από τις δύο μεθόδους αναζήτησης υπερτερεί της άλλης ανάλογα το πρόβλημα.

Πρέπει λοιπόν να σημειωθεί ότι κάθε πόρισμα για την κατάλληλη μέθοδο αναζήτησης και τις παραμέτρους της αφορά μονάχα το πρόβλημα που εξετάζεται, και ενδεχομένως τα παρόμοια σε αυτό προβλήματα. Δεν υπάρχει ένα σύνολο παραμέτρων βέλτιστο για πάσης φύσεως προβλήματα.

Ακόμη όμως και για ένα δεδομένο πρόβλημα με συγκεκριμένα κριτήρια βελτιστοποίησης, η επιλογή του κατάλληλου αλγορίθμου δεν αποτελεί εύκολη υπόθεση. Λόγω του ότι οι εξελικτικοί αλγόριθμοι βασίζονται ως επί το πλείστον σε πρακτικές, εμπειρικές εφαρμογές και λιγότερο σε θεωρητικό υπόβαθρο, είναι σχεδόν αδύνατο να προβλέψει κανείς με βεβαιότητα ποιος αλγόριθμος θα αποδειχθεί καταλληλότερος για ένα ορισμένο πρόβλημα.

Επιπρόσθετα, ακόμη και για τον ίδιο κατά βάση αλγόριθμο, διαφορετική παραμετροποίηση μπορεί να οδηγήσει σε πολύ διαφορετικά αποτελέσματα, κάτι που καθιστά την επιλογή αλγορίθμου ακόμη δυσκολότερη.

Έτσι, κρίνεται σκόπιμος ο πειραματισμός με διαφορετικούς αλγορίθμους και παραμέτρους αυτών. Η αξιολόγηση των αποτελεσμάτων θα δείξει ποιος υπερτερεί ανά περίπτωση, ενώ τα αποτελέσματα συχνά απέχουν από τις αρχικές προσδοκίες, είτε προς το καλύτερο είτε προς το χειρότερο.

2.7 Μεθευρετικές τεχνικές που έχουν χρησιμοποιηθεί για την επίλυση του F-JSSP

Οι Brucker και Schlie (1990) πρωτοπόρησαν με την παρουσίαση ενός πολυωνυμικού αλγορίθμου για πρόβλημα flexible job shop με δύο εργασίες. Ο Brandimarte (1993) πρότεινε έναν ιεραρχικό αλγόριθμο βασισμένο στο Tabu Search για το F-JSSP. Χρησιμοποίησε μια ιεραρχική στρατηγική ούτως ώστε να διασπάσει το F-JSSP σε δύο επίπεδα αποφάσεων: την επιλογή των διαδρομών και τον χρονικό προγραμματισμό (routing – scheduling). Τα δύο υποπροβλήματα που προέκυψαν επιλύθηκαν με την αναζήτηση Tabu. Οι Hurink, Jurisch et al. (1994) υλοποίησαν μια νέα εφαρμογή της αναζήτησης Tabu στο F-JSSP με πολύ ικανοποιητικά αποτελέσματα στα προβλήματα αναφοράς - benchmark.

Οι Dauzere-Peres και Paulli (1997) ασχολήθηκαν με την επίλυση ενός F-JSSP θεωρώντας ως αντικειμενική συνάρτηση προς ελαχιστοποίηση το makespan. Εισήγαγαν μια επεκταμένη εκδοχή του μοντέλου διαζευκτικών γράφων η οποία είχε τη δυνατότητα να λαμβάνει υπόψη το γεγονός ότι χρειάζεται να γίνει επιλογή μηχανών για τις λειτουργίες. Αυτό τους επέτρεψε να δημιουργήσουν μια ολοκληρωμένη - integrated προσέγγιση, ορίζοντας μια δομή γειτονικών λύσεων του προβλήματος στην οποία δεν υπάρχει διαφοροποίηση μεταξύ της επανα-ανάθεσης και του εκ νέου χρονικού προγραμματισμού μίας λειτουργίας. Τελικά, με χρήση της αναζήτησης Tabu τα υπολογιστικά αποτελέσματα έδειξαν την αποδοτικότητα της τεχνικής αυτής.

Οι Hussain και Joshi (1998) ανέλυσαν το JSSP με εναλλακτική επιλογή διαδρομών (routing) για την εκτέλεση των εργασιών. Για την αντιμετώπιση αυτού του προβλήματος χρησιμοποιήθηκε ένας γενετικός αλγόριθμος δύο περασμάτων. Στο πρώτο πέρασμα επιλέγονται οι εναλλακτικές με χρήση γενετικού αλγορίθμου, ενώ με το δεύτερο πέρασμα καθορίζεται η σειρά και οι χρόνοι εκκίνησης των εργασιών της επιλεχθείσας εναλλακτικής με την επίλυση ενός μη γραμμικού προβλήματος. Ένας μη γραμμικός περιορισμός περιορίζει την πολυπλοκότητα του προβλήματος, γι’ αυτό και χρησιμοποιείται στο δεύτερο πέρασμα του αλγορίθμου, ο οποίος κατάφερε να επιτύχει βέλτιστες λύσεις για μικρά δοκιμαστικά προβλήματα. Οι Dauzere-Peres, Roux et al. (1998) ασχολήθηκαν με ένα πρακτικό πρόβλημα στο οποίο μία λειτουργία μπορεί να έχει περισσότερες από μία προαπαιτούμενες (predecessor) λειτουργίες καθώς και περισσότερες από μία λειτουργίες που να το διαδέχονται στη δρομολόγηση της εργασίας. Για την ελαχιστοποίηση του χρόνου ολοκλήρωσης των εργασιών (**makespan**) έκαναν χρήση της

ιεραρχικής προσέγγισης τόσο για την ανάθεση των λειτουργιών στους πόρους όσο και για την διαδοχή (sequencing) των λειτουργιών στις μηχανές. Παρουσιάζεται αναπαράσταση του προβλήματος με διαζευκτικούς γράφους και προτείνεται μια δομή γειτονικών λύσεων. Με την δοκιμή σε προβλήματα αναφοράς επιδεικνύεται η αποτελεσματικότητα του αλγορίθμου τους. Οι Brucker και Neyer (1998) ασχολήθηκαν με την επίλυση του multi mode JSSP με τη χρήση Tabu search. Οι Chen, Ihlow et al. (1999) παρουσίασαν ένα νέο γενετικό αλγόριθμο για την επίλυση του F-JSSP με κριτήριο το makespan. Κωδικοποίησαν το πρόβλημα με χρωμοσώματα αποτελούμενα από δύο μέρη: α) το πρώτο μέρος καθορίζει την διαδρομή της εργασίας και β) το δεύτερο την αλληλουχία των λειτουργιών σε κάθε μηχανή. Νέοι γενετικοί τελεστές χρησιμοποιήθηκαν για τη διαδικασία αναπαραγωγής του αλγορίθμου. Αριθμητικά πειράματα έδειξαν πως ο γενετικός αλγόριθμος ήταν ικανός να βρει υψηλής ποιότητας χρονοπρογράμματα. Οι Mastrolilli και Gambardella (1996) για την ελαχιστοποίηση του makespan του F-JSSP δημιούργησαν μια τεχνική τοπικής αναζήτησης μαζί με δύο συναρτήσεις γειτονικότητας. Η βασική τους προσφορά ήταν ο περιορισμός του συνόλου των πιθανών γειτονικών λύσεων σε ένα υποσύνολο το οποίο περιείχε πάντοτε τη γειτονική λύση με το μικρότερο makespan.

Οι Kacem, Hammadi et al. (2002a) πρότειναν μια προσέγγιση Pareto βασισμένη στην υβριδοποίηση του fuzzy logic και των εξελικτικών αλγορίθμων. Η υβριδική αυτή προσέγγιση εκμεταλλεύεται τις δυνατότητες αναπαράστασης γνώσης του FL και τις δυνατότητες προσαρμογής των εξελικτικών αλγορίθμων. Ο Kacem (2003) παρουσίασε δύο νέες ενσωματωμένες προσεγγίσεις για την ταυτόχρονη λύση της ανάθεσης μηχανών και του JSSP (με ολική και μερική ευελιξία).

Οι Xia και Wu (2005) ανέπτυξαν έναν υβριδικό αλγόριθμο που προέκυψε από τη σύνθεση της Particle Swarm Optimization και του SA για το F-JSSP με πολλαπλούς στόχους βελτιστοποίησης. Τα αποτελέσματα της υπολογιστικής μελέτης έδειξαν πως ο αλγόριθμος είναι αποδοτικός, ιδιαίτερα για προβλήματα μεγάλης κλίμακας. Οι αντικειμενικοί στόχοι ήταν: χρόνος ολοκλήρωσης εργασιών , συνολικός φόρτος μηχανών, φόρτος εργασίας κρίσιμης μηχανής.

Οι Pezzella, Morganti et al. (2008) σχεδίασαν έναν γενετικό αλγόριθμο για το F-JSSP. Ο αλγόριθμος αυτός ενσωμάτωνε διαφορετικές στρατηγικές για την δημιουργία του αρχικού πληθυσμού, την επιλογή των μελών του πληθυσμού για την αναπαραγωγή και τη δημιουργία νέων μελών του πληθυσμού. Τα αποτελέσματα έδειξαν πως η ενσωμάτωση πολλαπλών στρατηγικών στα πλαίσια ενός γενετικού αλγορίθμου μπορεί να οδηγήσει σε καλύτερα αποτελέσματα συγκριτικά με τους άλλους γενετικούς αλγορίθμους.

Οι Liu, Abraham et al. (2009) χρησιμοποίησαν μια προσέγγιση πολλαπλών Particle Swarms (Multi-PSO) για την επίλυση του F-JSSP πολλαπλών στόχων. Η τεχνική αναζήτησης αποτελούταν από πολλαπλά σμήνη σωματιδίων-λύσεων, τα οποία αναζητούσαν πιθανές αλλαγές στην επιλογή των μηχανών για τις λειτουργίες καθώς και στη σειρά εκτέλεσης τους. Όλα τα

σμήνη αναζητούν τη βέλτιστη λύση συνεργατικά και διατηρούν την ισορροπία μεταξύ της διαφορετικότητας των σωματιδίων και του χώρου λύσεων. Τα αποτελέσματα έδειξαν πως μια τέτοια προσέγγιση είναι αποτελεσματική για το F-JSSP με πολλαπλά objectives και ειδικά για προβλήματα μεγάλης κλίμακας.

Οι Girish και Jawahar (2009) πρότειναν μια μέθοδο αναζήτησης βασισμένη στην Particle Swarm Optimization για το F-JSSP με κριτήριο την ελαχιστοποίηση του makespan. Μέσω συγκριτικών αξιολογήσεων απέδειξαν την αποτελεσματικότητα της προτεινόμενης PSO για την επίλυση του προβλήματος.

3. Ορισμός του προβλήματος / Μαθηματική μοντελοποίηση

3.1 Ορισμός του προβλήματος

Το F-JSSP μπορεί να οριστεί ως εξής:

Διατίθενται ***m*** μηχανές και θέλουμε να προγραμματίσουμε την εκτέλεση ***n*** εργασιών. Κάθε εργασία αποτελείται από μία αλληλουχία λειτουργιών $O_{j,h}$, $h=1,\ldots,h_j$, όπου $O_{j,h}$ η h κατά σειρά λειτουργία της εργασίας j και h_j ο συνολικός αριθμός λειτουργιών που αποτελούν την εργασία j.

Δίνεται επίσης το σύνολο $M=\{M_1,M_2,\ldots,M_m\}$ των μηχανών του προβλήματος. Εφόσον δε διευκρινίζεται διαφορετικά, ο δείκτης i αντιστοιχεί σε μηχανή, ο δείκτης j αντιστοιχεί σε εργασία και ο δείκτης h αντιστοιχεί σε λειτουργία στις μαθηματικές σχέσεις που θα ακολουθήσουν.

Για κάθε λειτουργία h της εργασίας j ορίζεται ως $M_{j,h} \subseteq M$ το σύνολο των μηχανών στις οποίες είναι δυνατό να εκτελεστεί αυτή η λειτουργία, καθώς και ο αντίστοιχος χρόνος επεξεργασίας της λειτουργίας Pi,j,h από κάθε μηχανή του συνόλου αυτού.

Το σύνολο Mj,h ορίζεται από τη μεταβλητή $a_{i,j,h}$ ως εξής: Ορίζεται ο δείκτης k για κάθε μηχανή, ο οποίος καθορίζει τη σειρά εκτέλεσης των ανατεθειμένων λειτουργιών σε αυτή τη μηχανή.

- Οι λειτουργίες μιας εργασίας αποτελούν μια αλυσίδα και πρέπει να εκτελούνται με συγκεκριμένη σειρά. Για την έναρξη της επεξεργασίας μίας λειτουργίας είναι απαραίτητο να έχει ολοκληρωθεί η επεξεργασία της προηγούμενης λειτουργίας της ίδιας εργασίας.

- Κάθε μηχανή μπορεί να επεξεργάζεται ανά πάσα στιγμή μία μόνο λειτουργία.
- Τη χρονική στιγμή 0 όλες οι μηχανές είναι ελεύθερες.

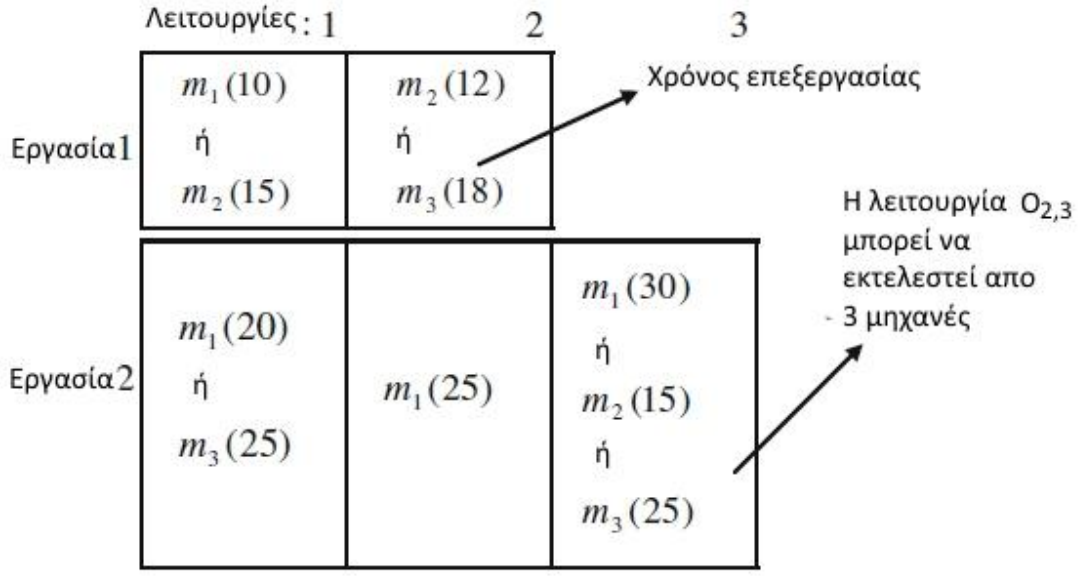
- Δεν απαιτείται χρόνος προετοιμασίας για τις μηχανές πριν την εκτέλεση των λειτουργιών (setup time).
- Δεν επιτρέπεται η προεκχώρηση (*preemption*), γεγονός που υποδηλώνει ότι από τη στιγμή της έναρξης μιας δραστηριότητας δεν επιτρέπεται η διακοπή της μέχρι και την ολοκλήρωσή της (χρόνος λήξης = χρόνος έναρξης + διάρκεια).

- Δεν υπάρχει ενδεχόμενο διακοπής λειτουργίας των μηχανών.

Η εύρεση μιας λύσης για το πρόβλημα έχει δύο σκέλη:

- Την επιλογή μηχανής για τις λειτουργίες των εργασιών του προβλήματος
- Την επιλογή της σειράς εκτέλεσης των ανατεθειμένων λειτουργιών σε κάθε μηχανή.

Δεδομένων αυτών των επιλογών, και εφόσον αυτές ικανοποιούν τους περιορισμούς του προβλήματος, δημιουργείται ένα χρονικό πρόγραμμα για την εκτέλεση των εργασιών του προβλήματος.



Πίνακας 1: Παράδειγμα προβλήματος Flexible Job Shop (2 εργασίες με 2 και 3 λειτουργίες αντίστοιχα , 3 μηχανές) (Fattahi, Mehrabad, & Jolai, 2007)

3.2 Μαθηματική μοντελοποίηση

Στη συνέχεια ορίζονται οι παράμετροι για τη μαθηματική μοντελοποίηση (MILP) του F-JSSP (Fattahi et al., 2007):

i=1,…..,m (μηχανές)

j=1,…..,n (εργασίες)

h=1,…..,hj (λειτουργίες εντός εργασίας)

n ο αριθμός των εργασιών

m ο αριθμός των μηχανών

$$a_{i,j,h} = \begin{cases} 1 \ \alpha\nu \ \eta \ O_{j,h} \ \mu\pi o\rho\epsilon\acute{\iota} \ \nu\alpha \ \epsilon\kappa\tau\epsilon\lambda\epsilon\sigma\tau\epsilon\acute{\iota} \ \sigma\tau\eta \ \mu\eta\chi\alpha\nu\acute{\eta} \ i \\ 0 \ \sigma\epsilon \ \alpha\nu\tau\acute{\iota}\theta\epsilon\tau\eta \ \pi\epsilon\rho\acute{\iota}\pi\tau\omega\sigma\eta \end{cases}$$

pi,j,h ο χρόνος επεξεργασίας της λειτουργίας Oj,h εφόσον αυτό εκτελείται στη μηχανή i ($p_{i,j,h} > 0$)

L ένας μεγάλος αριθμός

Μεταβλητές απόφασης (***i=1,…..,m; j=1,….,n; h=1,….,hj; , k=1,….,ki;*** *)*

Cmax : Ο συνολικός χρόνος για την ολοκλήρωση όλων των εργασιών (makespan)

$$y_{i,j,h} = \begin{cases} 1 \ \alpha\nu \ \eta \ \mu\eta\chi\alpha\nu\acute{\eta} \ i \ \acute{\epsilon}\chi\epsilon\iota \ \epsilon\pi\iota\lambda\epsilon\gamma\epsilon\acute{\iota} \ \gamma\iota\alpha \ \tau o \ operation \ O_{j,h} \\ 0 \ \sigma\epsilon \ \alpha\nu\tau\acute{\iota}\theta\epsilon\tau\eta \ \pi\epsilon\rho\acute{\iota}\pi\tau\omega\sigma\eta \end{cases}$$

$$x_{i,j,h,k} = \begin{cases} 1 \ \alpha\nu \ \tau o \ O_{j,h} \ \epsilon\kappa\tau\epsilon\lambda\epsilon\acute{\iota}\tau\alpha\iota \ \sigma\tau\eta \ \mu\eta\chi\alpha\nu\acute{\eta} \ i \ \mu\epsilon \ \sigma\epsilon\iota\rho\acute{\alpha} \ \pi\rho o\tau\epsilon\rho\alpha\iota\acute{o}\tau\eta\tau\alpha\varsigma \ k \\ 0 \ \sigma\epsilon \ \alpha\nu\tau\acute{\iota}\theta\epsilon\tau\eta \ \pi\epsilon\rho\acute{\iota}\pi\tau\omega\sigma\eta \end{cases}$$

tj,h ο χρόνος έναρξης της εκτέλεσης της λειτουργίας Oj,h.

$Tm_{i,k}$ ο χρόνος έναρξης του k κατά σειρά λειτουργίας της μηχανής i

k_i ο συνολικός αριθμός των λειτουργιών που έχουν ανατεθεί στη μηχανή i.

$Ps_{j,h}$ ο χρόνος επεξεργασίας της λειτουργίας Oj,h στην ήδη επιλεγμένη για αυτό μηχανή.

Με ορισμένες αυτές τις παραμέτρους, το πρόβλημα είναι να βρεθεί ανάθεση των λειτουργιών σε μηχανές και ορισμός των χρόνων έναρξης και λήξης των λειτουργιών για την ελαχιστοποίηση του χρόνου ολοκλήρωσης όλων των εργασιών Cmax.

Ελαχιστοποίησε το Cmax

Υπό τους περιορισμούς:

$$C_{max} \geq t_{j,h_j} + Ps_{j,h_j}\ \gamma\iota\alpha\ j = 1, \ldots\ldots, n;\ (1)$$

$$\sum_i y_{i,j,h} \cdot p_{i,j,h} = Ps_{j,h}\ \ \gamma\iota\alpha\ \ j = 1, \ldots, n;\ \ h = 1, \ldots. h_j; (2)$$

$$t_{j,h} + Ps_{j,h} \leq t_{j,h+1}\ \gamma\iota\alpha\ j = 1, \ldots, n; h + 1, \ldots, h_j\ - 1; (3)$$

$$Tm_{i,k} + Ps_{j,h} \cdot x_{i,j,h,k} \leq Tm_{i,k+1}\ \gamma\iota\alpha\ i = 1, \ldots, m;$$

$$j = 1, \ldots, n; h = 1, \ldots, h_j; k = 1, \ldots, k_i - 1; (4)$$

$$Tm_{i,k} \leq t_{j,h} + (1 - x_{i,j,h,k}) \cdot L\ \ \gamma\iota\alpha\ i = 1, \ldots, m;$$

$$j = 1, \ldots, n; h = 1, \ldots, h_j; k = 1, \ldots, k_i; (5)$$

$$Tm_{i,k} + (1 - x_{i,j,h,k}) \cdot L \geq t_{j,h}\ \ \gamma\iota\alpha\ i = 1, \ldots, m;$$

$$j = 1, \ldots, n; h = 1, \ldots, h_j; k = 1, \ldots, k_i; (6)$$

$$y_{i,j,h} \leq a_{i,j,h}\ \ \gamma\iota\alpha\ i = 1, \ldots, m;$$

$$j = 1, \ldots, n; h = 1, \ldots, h_j; (7)$$

$$\sum_j \sum_h x_{i,j,h,k} = 1\ \gamma\iota\alpha\ \ i = 1, \ldots, m;\ k = 1, \ldots, k_i; (8)$$

$$\sum_i y_{i,j,h} = 1\ \ \gamma\iota\alpha\ \ j = 1, \ldots, n;\ \ h = 1, \ldots, h_j; (9)$$

$$\sum_k x_{i,j,h,k} = y_{i,j,h}\ \gamma\iota\alpha\ \ i = 1, \ldots, m;\ j = 1, \ldots, n;\ h = 1, \ldots, h_j; (10)$$

$$t_{j,h} \geq 0\ \gamma\iota\alpha\ j = 1, \ldots, n;\ \ h = 1, \ldots, h_j; (11)$$

$$PS_{j,h} \geq 0\ \gamma\iota\alpha\ j = 1, \ldots, n;\ \ h = 1, \ldots, h_j; (12)$$

$$Tm_{i,k} \geq 0\ \gamma\iota\alpha\ i = 1, \ldots, m;\ \ k = 1, \ldots, k_i; (13)$$

$$x_{i,j,h,k} \in \{0,1\} \ \gamma\iota\alpha \ i = 1, \dots, m;$$

$$j = 1, \dots, n; h = 1, \dots, h_j; k = 1, \dots, k_i; (14)$$

$$y_{i,j,h} \in \{0,1\} \ \gamma\iota\alpha \ i = 1, \dots, m;$$

$$j = 1, \dots, n; h = 1, \dots, h_j; \ (15)$$

Η σχέση (1) καθορίζει το συνολικό χρόνο για την ολοκλήρωση όλων των εργασιών (makespan). Ελέγχονται οι χρόνοι ολοκλήρωσης όλων των εργασιών του προβλήματος και ο μέγιστος εξ αυτών αποτελέι το makespan (Cmax)

Η σχέση (2) καθορίζει τον χρόνο επεξεργασίας της λειτουργίας $O_{j,h}$ από την επιλεγμένη για αυτή μηχανή.

Ο περιορισμός της σχέσης (3) αναγκάζει κάθε εργασία να ακολουθεί μια ορισμένη αλληλουχία λειτουργιών.

Ο περιορισμός της σχέσης (4) αναγκάζει κάθε μηχανή να επεξεργάζεται μόνο μια λειτουργία σε δεδομένη χρονική στιγμή.

Οι περιορισμοί των σχέσεων (5) και (6) καθιστούν την έναρξη μια λειτουργίας $O_{j,h}$ επιτρεπτή μόνο όταν η μηχανή στην οποία θα εκτελεστεί η λειτουργία είναι ελεύθερη.

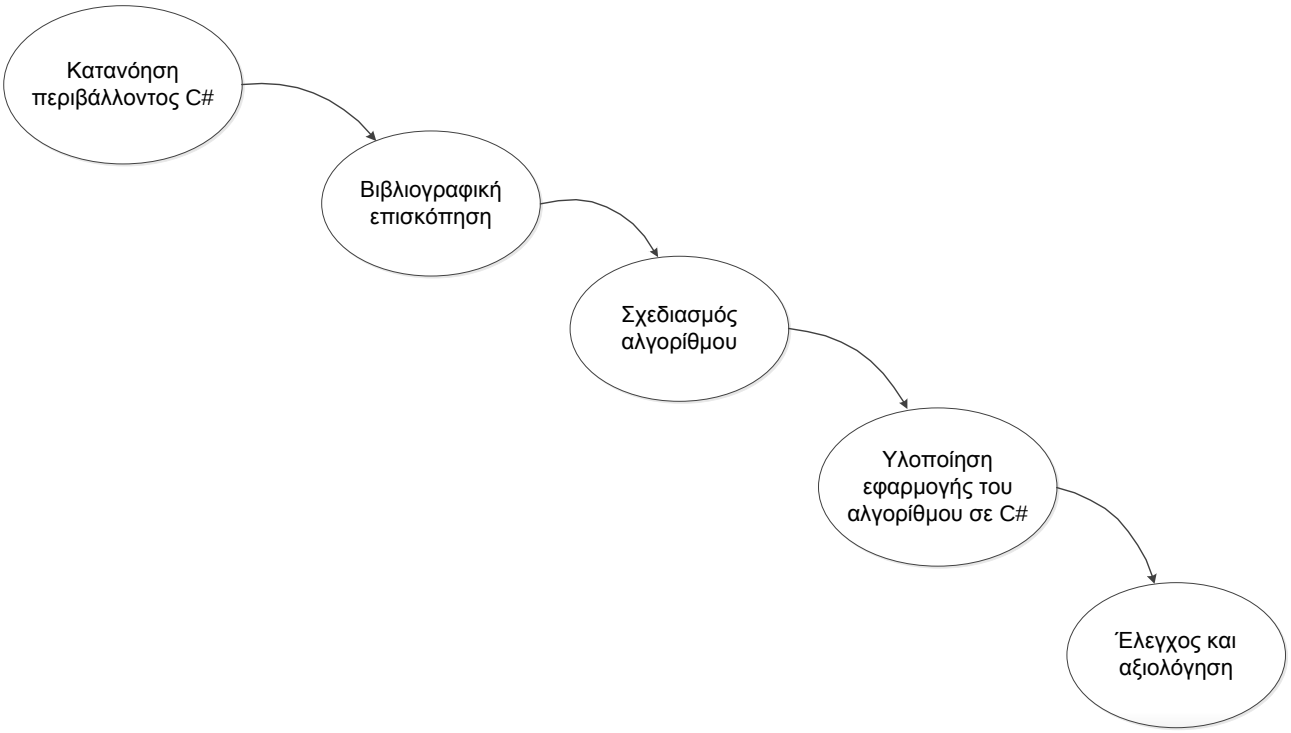
Η σχέση (7) καθορίζει τις μηχανές που έχουν τη δυνατότητα να εκτελέσουν κάθε λειτουργία.

Η σχέση (8) αφορά την ανάθεση των λειτουργιών σε μηχανές καθώς και τη σειρά εκτέλεσης των λειτουργιών ανά μηχανή.

Οι περιορισμοί των σχέσεων (9) και (10) εξασφαλίζουν πως κάθε λειτουργία θα εκτελεστεί τελικά από μία και μόνο μηχανή, και με μία συγκεκριμένη σειρά προτεραιότητας.

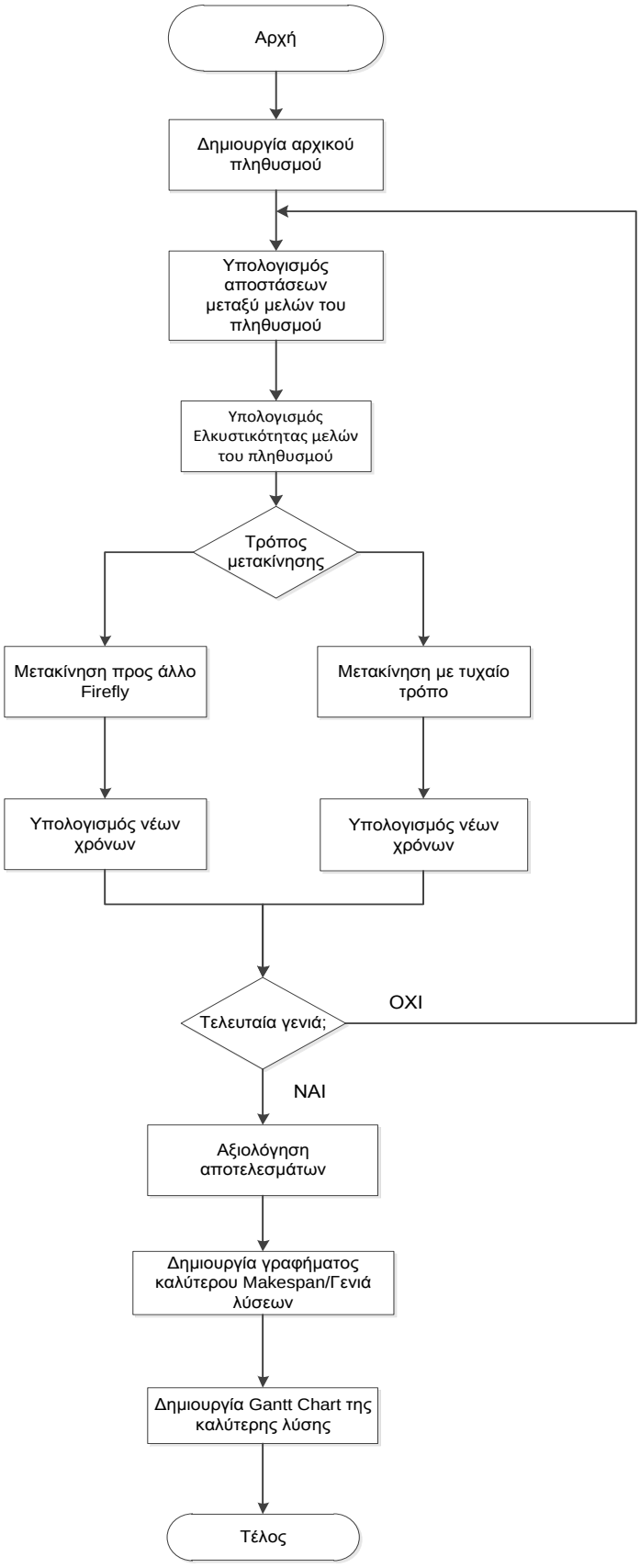
Η τιμή της μεταβλητής $x_{i,j,h,k}$ δείχνει την μηχανή στην οποία έχει ανατεθεί μια λειτουργία καθώς και τη σειρά εκτέλεσής της στη μηχανή αυτή.

4. Σχεδιασμός και ανάπτυξη του Αλγορίθμου Πυγολαμπίδας για το F-JSSP



Σχήμα 7: Διαδικασία μεθόδου έρευνας

Σχεδιασμός του αλγορίθμου - Σύνοψη βημάτων



Σχήμα 8: Γενικό διάγραμμα ροής για τον αλγόριθμο πυγολαμπίδων

Συνοπτικά, τα βήματα που ακολουθεί ο αλγόριθμος για την εύρεση λύσεων μπορούν να περιγραφούν ως εξής:

Σε πρώτη φάση, χρειάζεται να δημιουργηθεί, με τυχαίο τρόπο, ένας αρχικός πληθυσμός εφικτών λύσεων λαμβάνοντας υπόψη και τους περιορισμούς του προβλήματος.

Στη συνέχεια πρέπει να υπολογιστεί η ομοιότητα μεταξύ των λύσεων αυτών, το αν δηλαδή παρουσιάζουν μεταξύ τους κάποια κοινά στοιχεία. Το μέγεθος αυτό αντιστοιχεί στην απόσταση μεταξύ των πυγολαμπίδων.

Συναρτήσει του μεγέθους της ομοιότητας μεταξύ των λύσεων καθώς και την τιμή της αντικειμενικής συνάρτησης κάθε λύσης (δηλαδή του χρόνου ολοκλήρωσης των εργασιών-makespan) υπολογίζεται η τιμή της ελκυστικότητας κάθε πυγολαμπίδας ως προς μία άλλη πυγολαμπίδα.

Αφού αρχικοποιηθεί ο πληθυσμός των λύσεων, αυτές υπόκεινται σε αλλαγές. Αυτό συμβαίνει τόσες φορές όσες ο αριθμός των γενεών-επαναλήψεων που έχει οριστεί από τον χρήστη για τη λύση του προβλήματος.

Σε κάθε γενιά, οι πυγολαμπίδες – πιθανές λύσεις μετακινούνται είτε με τυχαίο τρόπο, είτε ακολουθώντας κάποια άλλη πυγολαμπίδα. Δημιουργείται έτσι μια νέα λύση, η οποία είναι μια παραλλαγμένη εκδοχή της λύσης που παρίστανε η ίδια πυγολαμπίδα στην προηγούμενη γενιά. Αφού καθοριστεί η νέα αλληλουχία λειτουργιών για κάθε μηχανή της λύσης, υπολογίζονται οι χρόνοι έναρξης και λήξεις της εκτέλεσης των λειτουργιών, με τρόπο τέτοιο ώστε να ικανοποιούνται οι περιορισμοί του προβλήματος.

Μετά την ολοκλήρωση των επαναλήψεων αποθηκεύονται τα δεδομένα της καλύτερης ευρεθείσας λύσης καθώς και στοιχεία για την πορεία του αλγορίθμου στην πάροδο των γενεών. Τέλος, εμφανίζεται το διάγραμμα Gantt της καλύτερης ευρεθείσας λύσης.

Υλοποίηση του αλγορίθμου

Ο Αλγόριθμος Πυγολαμπίδας για την επίλυση του F-JSSP αναπτύχθηκε στη γλώσσα προγραμματισμού C#, στο περιβάλλον του Visual Studio 2013, το οποίο διατίθεται δωρεάν στους φοιτητές της σχολής Μηχανολόγων Μηχανικών του ΕΜΠ μέσω της ιστοσελίδας DreamSpark[1] της Microsoft Η C#, καθότι αντικειμενοστραφής γλώσσα προγραμματισμού καθιστά χάρη στη χρήση αντικειμένων τον κώδικα του αλγορίθμου πιο ευέλικτο, ευκολότερα επεκτάσιμο για μελλοντική του βελτίωση και πιο κατανοητό στην ανάγνωση συγκριτικά με τις διαδικαστικές γλώσσες προγραμματισμού.

Ακόμη, η C# επιτρέπει τη χρήση πληθώρας εργαλείων από το .NET framework, όπως οι φόρμες WinForms και πολλά άλλα.

Για την ανάπτυξη του αλγορίθμου εγκαταστάθηκε ακόμη το πακέτο μαθηματικων Math.NET Numerics, το οποίο παρέχει πλήθος χρήσιμων υπολογιστικών εργαλείων (λ.χ. πράξεις πινάκων, νόρμες κ.α.). Το πακέτο αυτό διατίθεται δωρεάν[2].

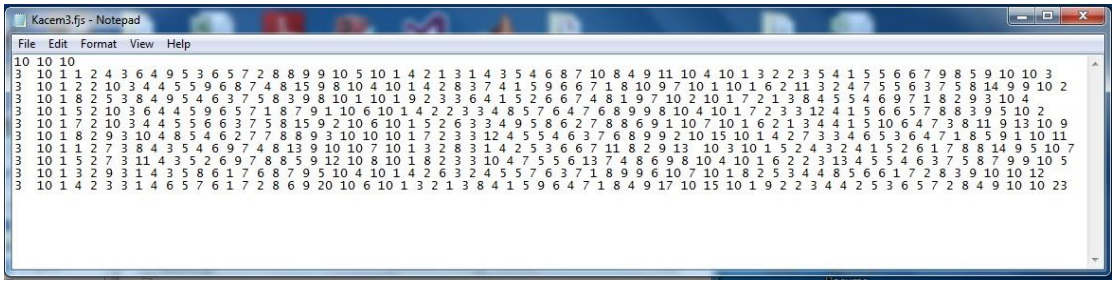
[1] https://www.dreamspark.com/
[2] http://www.mathdotnet.com/

4.1 Εισαγωγή δεδομένων

Η εισαγωγή δεδομένων γίνεται μέσω αρχείων .txt ή .fjs . Υποστηρίζονται δύο διαφορετικές μορφές εισαγωγής των δεδομένων.

Η πρώτη είναι εκείνη που εμφανίζεται στα προβλήματα αναφοράς (test instances – benchmarks) (Behnke & Geiger, 2012)[3].

Τα προβλήματα αυτά σε μορφή .fjs (προσβάσιμα με απλό text editor) είναι διαθέσιμα για download[4].



Σχήμα 9: Δείγμα μορφής δεδομένων εισόδου

Σε ένα τέτοιο αρχείο τα δεδομένα του προβλήματος αναπαρίστανται ως εξής:

Στην πρώτη γραμμή υπάρχουν το λιγότερο δύο αριθμοί. Ο πρώτος είναι ο αριθμός των εργασιών (jobs) του προβλήματος. Ο δεύτερος είναι ο αριθμός των μηχανών (machines) του προβλήματος. Ο τρίτος αριθμός, ο οποίος δεν είναι απαραίτητο να υπάρχει, είναι ο μέσος όρος των διαφορετικών μηχανών στις οποίες είναι δυνατό να ανατεθούν οι λειτουργίες του προβλήματος.

Οι υπόλοιπες γραμμές, από την δεύτερη και έπειτα, αναπαριστούν μία εργασία η καθεμία. Ο πρώτος αριθμός της γραμμής είναι πάντοτε ακέραιος και υποδεικνύει τον αριθμό των λειτουργιών (operations) της συγκεκριμένης εργασίας. Ο δεύτερος αριθμός, ο οποίος χωρίζεται από τον πρώτο με διπλό διάστημα, εκφράζει τον αριθμό των μηχανών στις οποίες μπορεί να ανατεθεί η πρώτη λειτουργία της εργασίας. Έστω k ο ακέραιος αυτός αριθμός. Αμέσως μετά τον k, ακολουθούν k ζεύγη αριθμών με τη μορφή του ζεύγους να είναι [μηχανή, χρόνος επεξεργασίας]. Τα ζεύγη αυτά υποδεικνύουν το ποιες είναι οι μηχανές στις οποίες μπορεί να ανατεθεί η λειτουργία καθώς και τον χρόνο

[3] http://edoc.sub.uni-hamburg.de/hsu/volltexte/2012/2982/

[4] http://edoc.sub.uni-hamburg.de/hsu/volltexte/2012/2982/pdf/FJSSPinstances.zip

επεξεργασίας που απαιτείται εφόσον ανατεθεί η λειτουργία στη μηχανή αυτή. Να σημειωθεί πως κάθε αριθμός μιας γραμμής που αναπαριστά ένα job χωρίζεται με τον επόμενο αριθμό με ένα διάστημα, με εξαίρεση τον πρώτο και τον δεύτερο αριθμό της γραμμής, που χωρίζονται μεταξύ τους με διπλό διάστημα.

Έτσι, ένα αρχείο .txt για την εισαγωγή δεδομένων ενός προβλήματος με 10 εργασίες (jobs) θα αποτελείται από 1+10 γραμμές.

Έστω μια γραμμή που αναπαριστά μία εργασία (job) με k λειτουργίες (operations) και m_i ο αριθμός των μηχανών στις οποίες μπορεί να ανατεθεί η λειτουργία i. Τότε η γραμμή θα έχει $1 + \sum_{i=1}^{k} 2 \cdot m_i$ αριθμούς.

Υποστηρίζεται ακόμη μία δεύτερη μορφή εισαγωγής δεδομένων, ο οποίος χρησιμοποιεί αρχεία .txt .

Στην περίπτωση αυτή υπάρχουν δύο αρχεία .txt τα οποία περιέχουν τα δεδομένα του προβλήματος.

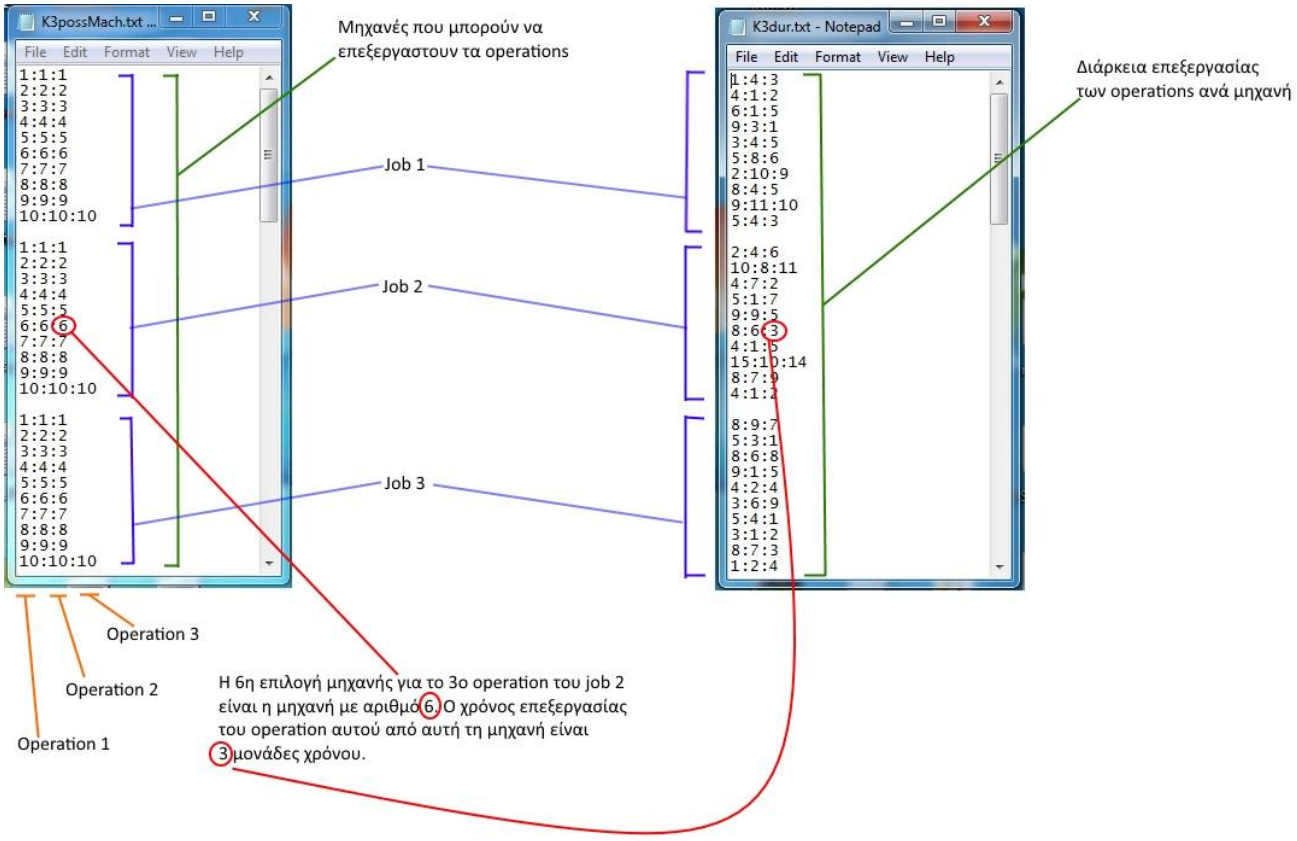
Το πρώτο περιέχει τις πιθανές μηχανές για τις λειτουργίες, οι οποίες αναπαρίστανται ως εξής:

Κάθε εργασία χωρίζεται από το επόμενο με μία κενή γραμμή. Μία εργασία καταλαμβάνει τόσες γραμμές όσες ο μέγιστος αριθμός πιθανών μηχανών στις οποίες μπορεί να ανατεθεί μία λειτουργία της εργασίας αυτής. Μία γραμμή που αφορά την εργασία αυτή, περιλαμβάνει τους αριθμούς-ID των μηχανών στις οποίες μπορούν να ανατεθούν οι λειτουργίες της εργασίας. Στη γραμμή αυτή οι αριθμοί είναι τοποθετημένοι με την σειρά των λειτουργιών, δηλαδή 1η μία πιθανή μηχανή της 1ης λειτουργίας της εργασίας, 2η μια πιθανή μηχανή της 2ης λειτουργίας της εργασίας κ.ο.κ. Εφόσον δεν υπάρχουν άλλες μηχανές που μπορούν να εκτελέσουν μία λειτουργία, στη θέση του αριθμού-ID της μηχανής υπάρχει η placeholder τιμή -1. Φυσικά η τιμή -1 είναι αδύνατο να εμφανιστεί στην πρώτη γραμμή μίας εργασίας, αφού κάτι τέτοιο θα σήμαινε πως κάποια λειτουργία δεν μπορεί να εκτελεστεί σε καμία μηχανή. Επίσης δεν είναι δυνατό κάποια γραμμή μίας εργασίας να έχει μονάχα τιμές -1, αφού κάτι τέτοιο θα σήμαινε πως δεν υπάρχουν άλλες μηχανές για καμία από τις λειτουργίες της εργασίας και άρα η γραμμή αυτή δεν έχει λόγο ύπαρξης. Οι αριθμοί χωρίζονται μεταξύ τους με άνω κάτω τελεία (:) .

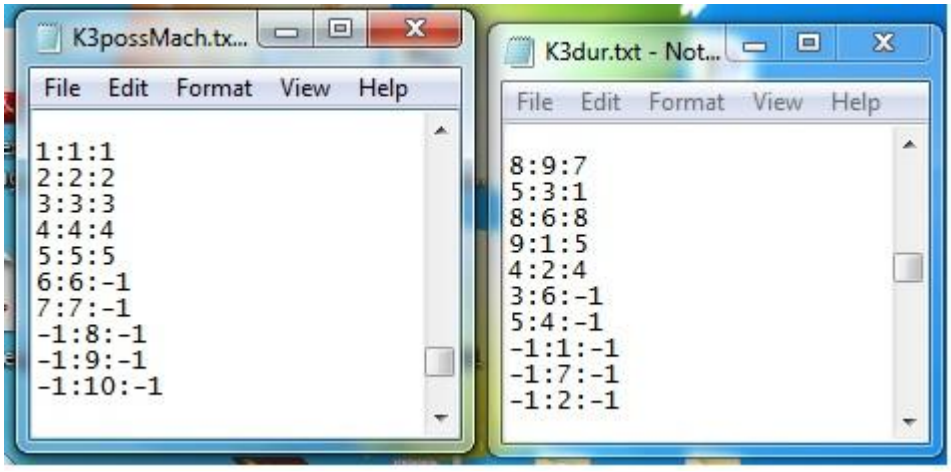
Το δεύτερο αρχείο .txt έχει ακριβώς την ίδια μορφή με το πρώτο, αλλά αντί για τον αριθμό της πιθανής μηχανής για την εκτέλεση μίας λειτουργίας, περιέχει τη διάρκεια της επεξεργασίας της λειτουργίας αυτής στη μηχανή που αναφέρεται στο 1ο αρχείο. Όπως είναι αναμενόμενο, στις θέσεις στις οποίες το πρώτο αρχείο έχει την placeholder τιμή της μη ύπαρξης μηχανής -1, στο

δεύτερο αρχείο υπάρχει επίσης η τιμή -1. Τα δύο αρχεία περιέχουν βέβαια ίσο πλήθος αριθμών.

Αυτή η μορφή αναπαράστασης των δεδομένων εισόδου διευκολύνει οπτικά με την έννοια ότι κάθε job αναπαριστάται στα .txt με τρόπο που θυμίζει πίνακες.



Σχήμα 10: Εναλλακτική μορφή δεδομένων εισόδου



Ακόμη, υπάρχει ένα αρχείο .txt μέσω του οποίου ο χρήστης εισάγει το μέγεθος του πληθυσμού καθώς και τον αριθμό των γενεών για τον οποίο θα εκτελεστεί ο αλγόριθμος.

Από τα αρχεία .txt με τα δεδομένα εισόδου δημιουργούνται τα αντικείμενα τύπου job1. Ένα αντικείμενο job1 αντιπροσωπεύει τις πληροφορίες για τις πιθανές μηχανές των λειτουργιών μίας εργασίας, καθώς και τις αντίστοιχες διάρκειες.

Αυτή η κλάση αντικειμένων περιέχει δύο πίνακες δύο διαστάσεων:

- Τον possibleMachForOper, ο οποίος περιέχει τις πιθανές μηχανές για τις λειτουργίες
- Τον durationsPerMachine, ο οποίος περιέχει τις αντίστοιχες διάρκειες

Τα αντικείμενα της κλάσης job1 περιέχουν ακόμη δύο μεταβλητές

- Την ακέραιη μεταβλητή numberOfOper που δείχνει των αριθμό των λειτουργιών της εργασίας
- Την ακέραιη μεταβλητή maxDiffMach που δείχνει τον μέγιστο αριθμό διαφορετικών μηχανών στις οποίες μπορεί να εκτελεστεί λειτουργία της εργασίας αυτής

Στην πρώτη διάσταση των πινάκων βρίσκεται ο δείκτης του αριθμού της επιλογής της μηχανής, ενώ στη δεύτερή τους διάσταση ο αριθμός σειράς της λειτουργίας στην αλυσίδα των λειτουργιών αυτής της εργασίας.

Για παράδειγμα, στη θέση possibleMachForOper[2,0] βρίσκεται το ID της $3^{ης}$ επιλογής μηχανής για την $1^{η}$ λειτουργία της εργασίας. Να σημειωθεί εδώ πως όλοι οι μετρητές στην γλώσσα προγραμματισμού C# ξεκινούν από την τιμή 0 και όχι από την τιμή 1.

Στην θέση durationsPerMachine[2,0] βρίσκεται η διάρκεια επεξεργασίας της λειτουργίας εφόσον επιλεχθεί για αυτό η μηχανή της οποίας το ID βρίσκεται στη θέση possibleMachForOper[2,0].

Στις θέσεις στις οποίες δεν υπάρχει διαθέσιμη επιλογή υπάρχει η placeholder τιμή -1, όπως περιγράφηκε και για την μία εκ των δύο μορφή των αρχείων .txt παραπάνω.

Με την χρήση ενός StreamReader γίνεται ανάγνωση των .txt αρχείων και στη συνέχεια με χρήση της μεθόδου Regex.Split το string που προκύπτει από ένα .txt αρχείο χωρίζεται σε strings τα οποία αντιπροσωπεύουν μία εργασία το καθένα.

Με τη χρήση της μεθόδου createJob (η οποία βρίσκεται στην κλάση job1) δημιουργούνται τα αντικείμενα των εργασιών για το δοσμένο πρόβλημα και τοποθετούνται στον μονοδιάστατο πίνακα myJobs[ ]. Η μέθοδος αυτή δέχεται σαν είσοδο μια μεταβλητή τύπου string η οποία περιέχει τις πληροφορίες για ένα job, και δίνει ως έξοδο ένα αντικείμενο τύπου job1.

Στην εφαρμογή εμφανίζεται η κλάση Global , στην οποία περιέχονται μεταβλητές τύπου public static οι οποίες μπορούν να χρησιμοποιηθούν σε οποιοδήποτε σημείο του κώδικα.

Ο συνολικός αριθμός των εργασιών, των λειτουργιών και των μηχανών αποτελούν χρήσιμες πληροφορίες κατά την εκτέλεση του προγράμματος και αποθηκεύονται στις μεταβλητές Global.totalNumberOfJobs , Global.totalNumberOfOper και Global.totalNumberOfMach αντίστοιχα.

Κάθε πιθανή λύση – πυγολαμπίδα - μέλος του πληθυσμού αναπαριστάται με ένα αντικείμενο της κλάσης firefly1.

Σε ένα τέτοιο αντικείμενο περιέχονται τα εξής:

Η ακέραιη μεταβλητή fireflyID που είναι ο αριθμός-ταυτότητα της πυγολαμπίδας

Η μεταβλητή πραγματικού αριθμού totalTime η οποία περιέχει το συνολικό χρόνο για την ολοκλήρωση όλων των εργασιών στη συγκεκριμένη λύση-πυγολαμπίδα, δηλαδή το makespan.

Η λίστα πραγματικών αριθμών timeEachMachineStops η οποία περιέχει τον χρόνο ολοκλήρωσης του τελευταίου κατά σειρά λειτουργίας σε κάθε μηχανή για αυτή τη λύση.

Η λίστα theMachines με αντικείμενα τύπου Machine.

Ένα αντικείμενο τύπου Machine περιέχει τα εξής:

Την ακέραια μεταβλητή ID. Πρόκειται για τον αριθμό-ταυτότητα της μηχανής.

Τη λίστα lAssignedJobOperationsToMachine, με αντικείμενα τύπου JobOperationFirefly, η οποία περιέχει τις πληροφορίες για τις ανατεθειμένες λειτουργίες στη μηχανή.

Ένα αντικείμενο τύπου JobOperationFirefly αποτελείται από τα εξής:

Την ακέραιη μεταβλητή iJobID που περιέχει την τιμή-ταυτότητα της εργασίας στο οποίο ανήκει η λειτουργία.

Την ακέραιη μεταβλητή iOperID που περιέχει τον σειριακό αριθμό της λειτουργίας στην αλυσίδα με τις λειτουργίες της εργασίας.

Την ακέραιη μεταβλητή iMachineID που περιέχει τη μηχανή στην οποία ανατίθεται η λειτουργία.

Την ακέραιη μεταβλητή iWhichChoiceID που περιέχει τον δείκτη επιλογής μηχανής, αν είναι δηλαδή η $1^{η}$, η $2^{η}$, η $3^{η}$ κ.ο.κ. επιλογή από τον πίνακα possibleMachinesForOper. Η μεταβλητή αυτή αν και δεν έχει σημασία στον πραγματικό κόσμο, είναι απαραίτητη ως δείκτης για την εκτέλεση του προγράμματος και για να αποφεύγονται περιττές αναζητήσεις που θα επιβράδυναν το πρόγραμμα.

Τη μεταβλητή πραγματικού αριθμού dStartTime που δείχνει τη χρονική στιγμή εκκίνησης της επεξεργασίας της λειτουργίας.

Τη μεταβλητή πραγματικού αριθμού dFinishTime που δείχνει τη χρονική στιγμή ολοκλήρωσης της επεξεργασίας της λειτουργίας.

Τη μεταβλητή τύπου bool bPending η οποία δείχνει αν η λειτουργία έχει ανατεθεί σε κάποια μηχανή ή αν αυτό εκρεμμεί.

Ο πίνακας jobOriented τύπου Job[ ]

Τα αντικείμενα τύπου Job περιέχουν μια ακέραιη μεταβλητή με τον αριθμό-ταυτότητα της εργασίας καθώς και την λίστα lAssignedInfoForOpersOfthisJob με αντικείμενα τύπου JobOperationFirefly.

Έτσι στον πίνακα jobOriented περιέχονται οι πληροφορίες για την μηχανή στην οποία έχει ανατεθεί κάθε λειτουργία των εργασιών, καθώς και οι χρόνοι εκκίνησης και ολοκλήρωσης της επεξεργασίας τους από τις μηχανές.

Είναι χρήσιμο να υποστηρίζεται αναπαράσταση των πληροφοριών ανάθεσης των λειτουργιών τόσο με τρόπο προσανατολισμένο στις εργασίες όσο και στις μηχανές. Με το να διατίθενται και οι δύο αυτοί προσανατολισμοί αποφεύγονται άσκοπες αναζητήσεις. Αν π.χ. δεν υπήρχε αναπαράσταση προσανατολισμένη στις εργασίες, για να βρεθεί το σε ποια μηχανή έχει ανατεθεί ένα συγκεκριμένο operation θα απαιτούταν σάρωση ολόκληρου του machine-oriented πίνακα αναθέσεων, κάτι που φυσικά δεν είναι επιθυμητό.

Επίσης η αναπαράσταση και με τους δύο τρόπους μπορεί να βοηθήσει στην εύρεση αδυναμιών σε μια λύση και κατ' επέκταση στην βελτίωσή της.

4.2 Αρχικοποίηση πληθυσμού

Σε πρώτη φάση, ο αλγόριθμος απαιτεί τη δημιουργία ενός αρχικού συνόλου λύσεων του προβλήματος, οι οποίες στη συνέχεια θα εξελιχθούν για την εύρεση ολοένα και καλύτερων λύσεων βάσει του στόχου που έχει τεθεί. Η αρχικοποίηση πρέπει να γίνεται σε σημαντικό βαθμό με τυχαίο τρόπο, ούτως ώστε σε κάθε εκτέλεση του αλγορίθμου να γίνεται εκκίνηση από διαφορετικά σημεία του χώρου λύσεων. Για να συμβεί αυτό, η διαδικασία της αρχικοποίησης του πληθυσμού πρέπει να εμπεριέχει στοχαστικές μεταβλητές.

Το μέγεθος του πληθυσμού των λύσεων όπως αυτό έχει δοθεί από τον χρήστη σε αρχείο .txt αποθηκεύεται στην μεταβλητή Global.totNumberOfFF.

Η δημιουργία του αρχικού πληθυσμού γίνεται με την μέθοδο createFirefly, η οποία δέχεται ως είσοδο τον πίνακα myJobs με τις πληροφορίες για τα jobs, καθώς και τη μεταβλητή Global.totalNumberOfMach με τον συνολικό αριθμό των μηχανών του προβλήματος.

Σε πρώτη φάση η μέθοδος createFirefly καλεί τη μέθοδο InitializeMachines. Αυτή αρχικοποιεί τις μηχανές και επιστρέφει μία λίστα lMachines με αντικείμενα τύπου Machine.

Χρησιμοποιείται ένας πίνακας ακεραίων με όνομα upcomingOper, ο οποίος μας δείχνει ποια είναι η επόμενη προς ανάθεση λειτουργία της κάθε εργασίας.

Δημιουργείται ακόμα μια λίστα αντικειμένων τύπου Job με όνομα pendingJobs στην οποία περιέχονται όλες οι εργασίες των οποίων εκκρεμούν λειτουργίες προς ανάθεση σε μηχανές. Αρχικά η λίστα αυτή περιέχει όλες τις εργασίες του προβλήματος.

Στη συνέχεια ξεκινάει ένας πρώτος κύκλος αναθέσεων εργασιών σε μηχανές, στον οποίο επιλέγεται τυχαία από μία φορά κάθε μηχανή. Με τον τρόπο αυτό επιδιώκεται, εφόσον είναι δυνατό, να ξεκινήσουν όλες οι μηχανές να επεξεργάζονται κάποια λειτουργία τη χρονική στιγμή 0, μιας και από τον ορισμό του προβλήματος τη χρονική στιγμή 0 όλες οι μηχανές είναι ελεύθερες.

Με τη χρήση της μεθόδου GiveMeANumber επιλέγεται τυχαία μία μηχανή που δεν έχει κληρωθεί προηγουμένως.

Στη συνέχεια, δημιουργείται η λίστα possibleJobsForMachRightNow. Σε αυτή προστίθενται οι εργασίες των οποίων τα προς ανάθεση upcoming operations μπορούν να εκτελεστούν στην κληρωθείσα μηχανή.

Εφόσον η λίστα αυτή δεν είναι κενή, δηλαδή εφόσον υπάρχουν λειτουργίες οι οποίες μπορούν να επεξεργαστούν από την κληρωθείσα μηχανή τη χρονική στιγμή 0, επιλέγεται με τυχαίο τρόπο μία εργασία από τη λίστα possibleJobsForMachRightNow και η λειτουργία της εργασίας αυτής ανατίθεται στη μηχανή.

Για να συμβεί αυτό, δημιουργείται αντικείμενο της κλάσης JobOperationFirefly στο οποίο περιέχονται οι απαραίτητες πληροφορίες για την ανάθεση της λειτουργίας.

Το αντικείμενο αυτό εισάγεται τόσο στη λίστα με το πρόγραμμα εργασιών της μηχανής στην οποία ανατίθεται (lAssignedJobOperationsToMachine) όσο και στη λίστα jobOriented η οποία δείχνει με τρόπο προσανατολισμένο στις εργασίες τις πληροφορίες ανάθεσης των λειτουργιών.

Μετά την επιτυχή ανάθεση της λειτουργίας μίας εργασίας και εφόσον η λειτουργία αυτή δεν ήταν η τελευταία της εργασίας, ο δείκτης της επερχόμενης λειτουργίας για αυτή την εργασία upcomingOper αυξάνεται κατά 1. Η διαδικασία αυτή επαναλαμβάνεται μέχρις ότου να έχουν επιλεχθεί από μία φορά όλες οι μηχανές.

Μετά από αυτόν τον κύκλο γύρο αναθέσεων ξεκινάει η γενική μορφή αναθέσεως εργασιών.

Αρχικά επιλέγεται με τυχαίο τρόπο μία μηχανή μέσω της μεθόδου GetMachine, στην οποία θα ανατεθεί λειτουργία προς εκτέλεση. Στη συνέχεια, σαρώνονται όλες οι προς ανάθεση λειτουργίες των εργασιών και όσες από αυτές μπορούν να εκτελεστούν από την κληρωθείσα μηχανή προστίθενται στη λίστα possibleJobsForMachRightNow. Αφού ελεγχθεί εάν υπάρχει μία τουλάχιστον λειτουργία που μπορεί να ανατεθεί στην κληρωθείσα μηχανή, επιλέγεται με τυχαίο τρόπο μία από της δυνατές λειτουργίες και ανατίθεται στη μηχανή.

Ο δείκτης upcomingOper της εργασίας αυξάνεται κατά ένα εφόσον υπάρχουν και άλλες προς ανάθεση λειτουργίες σε αυτή την εργασία. Σε αντίθετη περίπτωση η εργασία αυτή είναι πλέον ολοκληρωμένη, και αφαιρείται από την λίστα pendingJobs των εργασιών που εκκρεμούν.

Η διαδικασία αυτή επαναλαμβάνεται μέχρις ότου να έχουν ανατεθεί σε μηχανές όλες οι εργασίες όλων των λειτουργιών, οπότε και η λίστα pendingJobs είναι πλέον κενή. Τότε, η τιμή flagTelos παίρνει την τιμή true, και η αρχικοποίηση της λύσης-μέλους του πληθυσμού έχει ολοκληρωθεί.

4.3 Υπολογισμός χρόνων

Ακόμα και αν γνωρίζουμε τη σειρά με την οποία μία λειτουργία εκτελείται μία λειτουργία από μία μηχανή, η πληροφορία αυτή δεν επαρκεί από μόνη της για τον προσδιορισμό της χρονικής στιγμής στην οποία ξεκινά η επεξεργασία της λειτουργίας αυτής, αφού ο χρόνος εκκίνησης υπόκειται σε περιορισμούς.

Για τον υπολογισμό του χρόνου εκκίνησης μίας λειτουργίας σε μία μηχανή υλοποιήθηκε η μέθοδος getStartTime.

Δέχεται τις εξής εισόδους:

- Ένα αντικείμενο τύπου Machine, με την μηχανή στην οποία θα ανατεθεί η λειτουργία.
- Έναν πίνακα με αντικείμενα τύπου job1 στον οποίο περιέχονται οι πληροφορίες για τη διάρκεια επεξεργασίας των λειτουργιών ανά μηχανή

Τρία αντικείμενα τύπου JobOperationFirefly:

1. Το NewJobOperation, το οποίο είναι η προς ανάθεση λειτουργία.
2. Το previousOperOfJob το οποίο είναι η προηγούμενη κατά σειρά λειτουργία της εργασίας στην οποία ανήκει η προς ανάθεση λειτουργία.
3. Το previousOperOfMach το οποίο είναι η λειτουργία η οποία θα εκτελεστεί στη μηχανή αμέσως πριν την προς ανάθεση λειτουργία.

Επίσης δύο μεταβλητές τύπου bool.

1. Η μεταβλητή specialCaseMach εκφράζει την ειδική περίπτωση κατά την οποία η προς ανάθεση λειτουργία είναι η πρώτη που εκτελεί η επιλεγμένη μηχανή.
2. Η μεταβλητή specialCaseJob εκφράζει την ειδική περίπτωση κατά την οποία η προς ανάθεση λειτουργία είναι η πρώτο λειτουργία της εργασίας στην οποία ανήκει.

Για να ξεκινήσει η επεξεργασία της προς ανάθεση λειτουργίας, πρέπει να έχει ολοκληρωθεί η επεξεργασία της προηγούμενης λειτουργίας της εργασίας καθώς και η επιλεγμένη για τη λειτουργία μηχανή να είναι ελεύθερη προς χρήση, δηλαδή να έχει ολοκληρωθεί η επεξεργασία της προηγούμενης λειτουργίας που έχει ανατεθεί σε αυτή. Έτσι, ο χρόνος έναρξης της προς ανάθεση λειτουργίας είναι ο μέγιστος από τους δύο χρόνους λήξης των λειτουργιών previousOperOfMach και previousOperOfJob, εφόσον αυτές υπάρχουν.

Ο χρόνος λήξης της επεξεργασίας βρίσκεται εύκολα αθροίζοντας τον χρόνο έναρξης της λειτουργίας με την διάρκεια επεξεργασίας της στη συγκεκριμένη μηχανή, όπως αυτή αναγράφεται στον πίνακα myJobs.

Η εύρεση του previousOperOfJob γίνεται πολύ απλή χάρη στην ύπαρξη της λίστας αναθέσεων προσανατολισμένης στις εργασίες (jobOriented). Σε αντίθετη περίπτωση, αν δεν υπήρχε τέτοια λίστα, θα απαιτούταν σάρωση ολόκληρης της λίστας αναθέσεων στις μηχανές για την εύρεση της προηγούμενης λειτουργίας μιας εργασίας.

4.4 Υπολογισμός αποστάσεων

Στα προβλήματα συνεχών μεταβλητών, ο υπολογισμός της απόστασης μεταξύ δύο λύσεων είναι μία αρκετά απλή υπόθεση. Ιδιαίτερα όταν πρόκειται για προβλήματα 2 ή 3 μεταβλητών, η απόσταση μπορεί να αναπαρασταθεί και οπτικά στο επίπεδο ή στο χώρο.

Το Flexible Job Shop Scheduling Problem ωστόσο, διαφέρει σημαντικά, καθώς δεν πρόκειται για πρόβλημα συνεχών μεταβλητών. Έτσι, ο καθορισμός των ομοιοτήτων μεταξύ των πιθανών λύσεων και ο ορισμός της απόστασης μεταξύ τους γίνονται πολύ πιο περίπλοκες διαδικασίες.

Διερευνήθηκαν για το λόγο αυτό οι παράγοντες ομοιότητας μεταξύ των λύσεων. Όσο μεγαλύτεροι είναι οι παράγοντες αυτή, τόσο μειώνεται η απόσταση μεταξύ δύο λύσεων. Βασικός τέτοιος παράγοντας είναι η επιλογή ανάθεσης της ίδιας λειτουργίας στην ίδια μηχανή σε δύο λύσεις. Ακόμη, ισχυρός παράγοντας ομοιότητας είναι η εμφάνιση κοινών αλληλουχιών από λειτουργίες σε δύο λύσεις.

Για την καταγραφή των ομοιοτήτων μεταξύ των μελών του πληθυσμού αναπτύχθηκε η μέθοδος GetDistance, η οποία δέχεται ως είσοδο δύο πυγολαμπίδες. Αυτή απαριθμεί τις λειτουργίες οι οποίες εκτελούνται στην ίδια μηχανή και στις δύο λύσεις, ενώ σαρώνει ακόμα ολόκληρο το πρόγραμμα ανατεθειμένων λειτουργιών στις μηχανές και καταγράφει τις κοινές αλληλουχίες που εμφανίζονται καθώς και το μήκος αυτών (αν δηλαδή πρόκειται για 2, 3, 4 ή και περισσότερες λειτουργίες που εκτελούνται συνεχόμενα σε μία κοινή μηχανή και στα δύο μέλη του πληθυσμού των λύσεων).

Στην παρούσα υλοποίηση δίνεται ένας «πόντος» ομοιότητας για κάθε λειτουργία που εκτελείται από την ίδια μηχανή και στις δύο πυγολαμπίδες. Δύο «πόντοι» για κάθε δυάδα λειτουργιών που εκτελούνται συνεχόμενες στην

ίδια μηχανή. Τρεις «πόντοι» για κάθε τριάδα συνεχόμενων ίδιων λειτουργιών κ.ο.κ.

Προκύπτει έτσι ο εξής τύπος που υπολογίζει τους πόντους ομοιότητας μεταξύ δύο πυγολαμπίδων:

$$S = 1 * x_1 + 2 * x_2 + (3 * x_3 + 2 * 2 * x_3) + \cdots +$$

$$(n * x_n + 2 * (n-1) * x_n + \cdots + (n-1) * 2 * x_n) \Leftrightarrow$$

$$\Leftrightarrow S = 1 * x_1 + 2 * x_2 + (3 + 2 * 2) * x_3$$

$$+ \cdots + (n + 2 * (n-1) + \cdots + (n-1) * 2) * x_n$$

Βάσει του παραπάνω αναδρομικού τύπου προκύπτει η βαρύτητα ομοιότητας των κοινών αλληλουχιών αυξάνεται παραγοντικά ως προς το μήκος της κοινής αλληλουχίας ίδιων συνεχόμενων λειτουργιών.

Όπου x_1 ο αριθμός των λειτουργιών που έχουν ανατεθεί στην ίδια μηχανή, x_2 ο αριθμός των κοινών «δυάδων» λειτουργιών που έχουν ανατεθεί συνεχόμενες στην ίδια μηχανή, x_n ο αριθμός των αλληλουχιών με μήκος n που εκτελούνται συνεχόμενες στην ίδια μηχανή και στις δύο πυγολαμπίδες-λύσεις.

Όπως είναι φανερό από τον παραπάνω τύπο, επιλέχθηκε οι μεγάλες κοινές αλληλουχίες να «πριμοδοτούνται» ιδιαίτερα σε πόντους ομοιότητας, καθώς είναι λογικό μία μεγάλη κοινή αλληλουχία λειτουργιών μεταξύ δύο πιθανών λύσεων να καθιστά τις δύο αυτές λύσεις πολύ περισσότερο συγγενικές μεταξύ τους συγκριτικά με το να είχαν απλώς λειτουργίες που έχουν ανατεθεί στις ίδιες μηχανές, χωρίς όμως να υπάρχει κάποιος κοινή αλληλουχία χρονικού προγραμματισμού.

Οι αποστάσεις εισάγονται στον άνω τριγωνικό πίνακα theDistances, ενώ οι αντίστοιχες ελκυστικότητες στον άνω τριγωνικό πίνακα theAttr. Οι πίνακες αυτοί είναι άνω τριγωνικοί διότι η απόσταση μεταξύ δύο fireflies π.χ. του firefly 1 και του firefly 2 ταυτίζεται με την απόσταση μεταξύ του firefly 2 και του firefly1. Φυσικά, απόσταση ενός firefly από τον εαυτό του δεν υφίσταται. Στις κενές θέσεις του άνω τριγωνικού πίνακα εισάγεται η placeholder τιμή -1.

4.5 Ελκυστικότητα

Το κατά πόσο ένα μέλος του πληθυσμού ελκύει τα υπόλοιπα να κινηθούν προς το μέρος του, δηλαδή να δανειστούν στοιχεία από αυτό, εξαρτάται από δύο παράγοντες: την λαμπρότητά του και την απόσταση του από ένα άλλο μέλος του πληθυσμού που ενδέχεται να κινηθεί προς τη θέση του.

Αδιστατοποίηση

Για τον υπολογισμό του βαθμού ελκυστικότητας μίας πυγολαμπίδας προς μία άλλη, απαιτείται να λαμβάνεται υπόψη το μέγεθος του προβλήματος σε λειτουργίες. Σε αντίθετη περίπτωση, αν δηλαδή δεν εφαρμοζόταν κάποιου είδους αδιαστατοποίηση, ο ίδιος βαθμός ελκυστικότητας που θα ήταν μεγάλος στα πλαίσια ενός προβλήματος μικρού ή μεσαίου μεγέθους, θα λογιζόταν ως σημαντικός και στην περίπτωση ενός προβλήματος μεγάλου μεγέθους με πολλές λειτουργίες, κάτι που δεν ανταποκρίνεται στην πραγματικότητα.

Ορίστηκε έτσι ο βαθμό ελκυστικότητας **E** μίας πυγολαμπίδας i2 ως προς μία πυγολαμπίδα i1 (δηλαδή το πόσο ελκύει η i2 την i1 να την ακολουθήσει).

wDist είναι το βάρος που δίνεται στην ομοιότητα των δύο πυγολαμπίδων.

wBright είναι το βάρος που δίνεται στη λαμπρότητα της ακολουθούμενης πυγολαμπίδας, δηλαδή στο χρόνο ολοκλήρωσης όλων των εργασιών. Όσο πιο γρήγορα ολοκληρώνονται οι εργασίες σε μία λύση, τόσο πιο λαμπερή είναι η πυγολαμπίδα.

Αρχικά οι δύο παράμετροι βάρους wDist και wBright ορίστηκαν στην τιμή 1. Είναι στην ευχέρεια του χρήστη να αλλάξει τις τιμές αυτές ανάλογα με την εκδοχή του προς επίλυση προβλήματος, δίνοντας μεγαλύτερο βάρος όπου εκείνος το κρίνει σημαντικότερο.

Οι πόντοι **S** κοινών λειτουργιών μεταξύ των λύσεων αδιαστατοποιούνται με βάση το συνολικό αριθμό λειτουργιών στο πρόβλημα, αφού όσο περισσότερες λειτουργίες έχει ένα πρόβλημα, τόσο πιο πιθανό να βρεθούν κοινές αναθέσεις λειτουργιών μεταξύ δύο λύσεων του.

Η λαμπρότητα αδιαστατοποιείται με βάση τον χρόνο, διαιρώντας τον χρόνο ολοκλήρωσης των εργασιών στην ακολουθούμενη πυγολαμπίδα i2 (έστω **time_i2** με τον μέσο όρο των χρόνων ολοκλήρωσης των εργασιών των πυγολαμπίδων του αρχικού πληθυσμού (έστω **avgTimeInitPop**).

Προκύπτει τελικά η ελκυστικότητα:

$$E = \frac{S * wDist}{αριθμός\ λειτουργιών\ προβλήματος} * \frac{wBright}{(\frac{time_{i2}}{\text{avgTimeInitPop}})}$$

4.6 Μετακίνηση των Πυγολαμπίδων

Σε κάθε γενιά λύσεων, τα μέλη του πληθυσμού (πυγολαμπίδες) μετακινούνται. Αυτό σημαίνει πως υπόκεινται σε αλλαγές με σκοπό την εξερεύνηση του χώρου λύσεων του προβλήματος, με σκοπό την εύρεση ολοένα και καλύτερων λύσεων.

Μία πυγολαμπίδα μπορεί να κινηθεί με δύο διαφορετικούς τρόπους: είτε ακολουθώντας μία άλλη, ελκυστική ως προς τη μετακινούμενη, πυγολαμπίδα, είτε με τυχαίο τρόπο.

Ορίζεται ένα όριο ελάχιστης ελκυστικότητας ως προς μία πυγολαμπίδα. Σε περίπτωση που καμία από τις υπόλοιπες πυγολαμπίδες του πληθυσμού δεν ξεπερνά ελκυστικότητα το όριο αυτό, τότε θεωρείται πως δεν υπάρχει καμία πυγολαμπίδα η οποία να ελκύει επαρκώς την προς μετακίνηση πυγολαμπίδα. Στην περίπτωση αυτή η μετακινούμενη πυγολαμπίδα θα κινηθεί οπωσδήποτε με τυχαίο τρόπο.

4.6.1 Μετακίνηση με τυχαίο τρόπο

Η μετακίνηση των πυγολαμπίδων (δηλαδή η αλλαγή των λύσεων) με τυχαίο τρόπο, εξυπηρετεί στην εξερεύνηση νέων χώρων λύσεων. Η διαδικασία αυτή (exploration) είναι απαραίτητο κομμάτι κάθε εξελικτικού αλγορίθμου. Ο κώδικας της μεθόδου τυχαίας μετακίνησης παρατίθεται στο Παράρτημα 1.

Για την μετακίνηση μίας πυγολαμπίδας με τυχαίο τρόπο αναπτύχθηκε η μέθοδος moveFFrandomly. Αυτή δέχεται ως είσοδο ένα firefly συνοδευόμενο από τη λέξη-κλειδί ref, η οποία επιτρέπει την επιστροφή του ίδιου του αντικειμένου αφού πραγματοποιηθούν σε αυτό αλλαγές εντός της μεθόδου. Επίσης δίνεται ως είσοδος ο πίνακας myJobs με τις πληροφορίες για τις εργασίες του προβλήματος.

Εντός της μεθόδου ορίζεται η ακέραιη μεταβλητή numberOfOperToMove, η οποία καθορίζει τον αριθμό των λειτουργιών στη λύση τα οποία θα υποστούν αλλαγές για την μετακίνηση της πυγολαμπίδας. Ο αριθμός αυτός ορίζεται ως ποσοστό του συνολικού αριθμού των λειτουργιών στο πρόβλημα, και στις περισσότερες περιπτώσεις ορίστηκε στην τάξη μεγέθους του 5% των συνολικών λειτουργιών.

Σε περίπτωση που ένα πρόβλημα είναι πολύ μικρό, η μεταβλητή αυτή δεν μπορεί να πάρει τιμή μικρότερη του 1, αφού σε αυτή την περίπτωση η πυγολαμπίδα δεν θα άλλαζε καθόλου.

Για την επιλογή της λειτουργίας η οποία θα υποστεί αλλαγές, κληρώνεται με τυχαίο τρόπο ένας ακέραιος αριθμός μεταξύ του 0 και του συνολικού αριθμού των λειτουργιών στο πρόβλημα μείον 1[5]. Στη συνέχεια, εντοπίζεται η λειτουργία που αντιστοιχεί στον κληρωθέντα αριθμό με τον εξής τρόπο:

Έστω ότι κληρώθηκε ο αριθμός 34. Αν στην πρώτη μηχανή υπάρχουν ανατεθειμένες 10 λειτουργίες, στη δεύτερη 15 και στην τρίτη 20, τότε ο αριθμός 34 αντιστοιχεί στη $10^{η}$ κατά σειρά λειτουργία που εκτελείται στην τρίτη μηχανή, και βρίσκεται στη θέση 9 της λίστας της μηχανής αυτής (αφού η αρίθμησή της ξεκινά από το 0). Αυτό ισχύει γιατί και η αρίθμηση του τυχαίου αριθμού που κληρώθηκε ξεκινά από το 0. Άρα ο αριθμός 34 αντιστοιχεί στην $35^{η}$ κατά σειρά λειτουργία συνολικά και 10 ($1^{η}$ μηχανή) + 15 ($2^{η}$ μηχανή) + 10 (<20 τρίτης μηχανής) = 35.

Η μετακίνηση μίας λειτουργίας μπορεί να γίνει με δύο διαφορετικούς τρόπους: είτε αλλάζοντας τη μηχανή στην οποία θα εκτελεστεί, είτε αλλάζοντας τη σειρά και επομένως το χρόνο έναρξής του στην ίδια μηχανή στην οποία είναι ανατεθειμένο. Οι πιθανότητες να αλλάξει μηχανή η λειτουργία είναι 90%, ενώ οι πιθανότητες να παραμείνει στην ίδια μηχανή με άλλη σειρά εκτέλεσης είναι 10%.

Αν τυχόν μία λειτουργία μπορεί να εκτελεστεί μόνο από μία συγκεκριμένη μηχανή, τότε αναγκαστικά θα αλλάξει η σειρά εκτέλεσής του.

4.6.1.1 Αλλαγή μηχανής εκτέλεσης

Για την αλλαγή μηχανής εκτέλεσης μίας λειτουργίας κληρώνεται μια μηχανή από το σύνολο εκείνων οι οποίες μπορούν να επεξεργαστούν τη λειτουργία, με εξαίρεση τη μηχανή στην οποία είναι ήδη ανατεθειμένο.

Στη συνέχεια η λειτουργία τοποθετείται σε αυτή τη μηχανή. Η επιλογή της σειράς εκτέλεσης της λειτουργίας στη μηχανή αυτή γίνεται με βάση το χρονικό πρόγραμμα πριν τις μετακινήσεις των λειτουργιών. Η λειτουργία τελικά τοποθετείται στη νωρίτερη δυνατή επιτρεπτή θέση βάσει των περιορισμών. Συγκεκριμένα, καταγράφεται η χρονική στιγμή στην οποία ολοκληρώνεται η προηγούμενη λειτουργία στην εργασία της μετακινούμενης λειτουργίας. Στη συνέχεια εξετάζεται το αν αυτή τη χρονική στιγμή, η μηχανή στην οποία πρόκειται να μετακινηθεί η λειτουργία είναι ελεύθερη. Αν είναι ελεύθερη τότε η λειτουργία ανατίθεται στη θέση του προγράμματος της μηχανής που αντιστοιχεί σε αυτή την χρονική στιγμή. Σε αντίθετη περίπτωση, αν τη χρονική στιγμή αυτή η μηχανή είναι απασχολημένη, η λειτουργία ανατίθεται στην θέση αμέσως μετά την ολοκλήρωση της λειτουργίας που απασχολεί τη μηχανή.

[5] http://msdn.microsoft.com/en-us/library/2dx6wyd4%28v=vs.110%29.aspx

4.6.1.2 Μετακίνηση στην ίδια μηχανή

Για την μετακίνηση μίας λειτουργίας χωρίς αυτή να αλλάξει μηχανή στην οποία εκτελείται, πραγματοποιείται με μετακίνηση κατά 10% του μήκους της αλυσίδας των λειτουργιών που είναι ανατεθειμένες στη μηχανή. Π.χ. για μία μηχανή η οποία έχει 12 λειτουργίες προς εκτέλεση, η μετακινούμενη λειτουργία θα μετακινηθεί 1 θέση πιο νωρίς (10%*12=1.2→1).

Για να γίνει μετακίνηση με αυτόν τον τρόπο ελέγχεται το αν ικανοποιούνται όλοι οι περιορισμοί του προβλήματος. Σε αντίθετη περίπτωση δεν πραγματοποιείται η μετακίνηση.

Ενημέρωση χρόνων μίας πυγολαμπίδας

Μία λύση του προβλήματος, δηλαδή μια πυγολαμπίδα, είναι ουσιαστικά η αλληλουχία των εργασιών που θα εκτελεστούν από κάθε μηχανή, καθώς και οι χρόνοι έναρξης και λήξης των εργασιών αυτών. Επομένως, όταν η αλληλουχίες αλλάζουν για τη δημιουργία της πυγολαμπίδας της επόμενης γενιάς, αλλάζουν και αυτοί οι χρόνοι. Για το λόγο αυτό αναπτύχθηκε η μέθοδος updateFFtimes, η οποία δοθέντων των αλληλουχιών των εργασιών (οι οποίες βέβαια πρέπει να ικανοποιούν τους περιορισμούς του προβλήματος) για κάθε μηχανή, υπολογίζει τους χρόνους έναρξης και λήξης όλων των εργασιών. Να σημειωθεί πως η μέθοδος αυτή δεν είναι στοχαστική, δηλαδή δεν περιέχει παράγοντες τυχαιότητας. Οι αλληλουχίες των εργασιών στις μηχανές είναι από μόνες τους ικανός παράγοντας για τον καθορισμό των χρόνων του προγράμματος εργασιών (schedule).

Η μέθοδος updateFFtimes αρχικά αναθέτει χρόνο εκκίνησης 0 στις λειτουργίες που είναι ταυτόχρονα πρώτες στη σειρά εκτέλεσης της μηχανής τους και οι πρώτες λειτουργίες της εργασίας στην οποία ανήκουν. Συνεχίζει με την ανάθεση χρόνων στις λειτουργίες των οποίων έχουν ανατεθεί τόσο οι προηγούμενες λειτουργίες της εργασίας στην οποία ανήκουν, όσο και αυτές που έχουν ανατεθεί πριν από αυτές στην ίδια μηχανή. Η διαδικασία αυτή επαναλαμβάνεται μέχρις ότου να έχουν ορισθεί χρόνοι εκκίνησης και ολοκλήρωσης για όλες τις λειτουργίες της νέας λύσης.

4.6.2 Μετακίνηση προς άλλη Πυγολαμπίδα

Η μετακίνηση μίας πυγολαμπίδας προς μία άλλη, πρόκειται ουσιαστικά για τον δανεισμό στοιχείων από άλλες λύσεις του πληθυσμού, με έμφαση στην αξιοποίηση των καλύτερων λύσεων που έχουν βρεθεί τη στιγμή της μετακίνησης. Αυτό εξυπηρετεί στην αξιοποίηση των χαρακτηριστικών που κάνουν μία λύση επιτυχημένη (exploitation) και είναι βασικό κομμάτι των εξελικτικών αλγορίθμων.

Μία πυγολαμπίδα προς μετακίνηση μπορεί να κινηθεί προς την κατεύθυνση ενός άλλου μέλους του πληθυσμού, με την προϋπόθεση να υπάρχει μέλος του οποίου η ελκυστικότητα ως προς τη μετακινούμενη πυγολαμπίδα να είναι μεγαλύτερη της ελάχιστης ορισμένης (cutoff attractiveness).

Εφόσον υπάρχει μία τουλάχιστον τέτοια πυγολαμπίδα, η προς μετακίνηση πυγολαμπίδα έχει πιθανότητες 70% να ακολουθήσει μία άλλη πυγολαμπίδα και 30% να κινηθεί με τυχαίο τρόπο. Είναι απαραίτητο μία πυγολαμπίδα να μπορεί να κινηθεί και με τυχαίο τρόπο, ανεξαρτήτως του αν υπάρχουν άλλες πυγολαμπίδες κοντά της, αφού αυτό είναι που επιτρέπει την εξερεύνηση νέων χώρων λύσεων, κάτι που είναι απαραίτητο για την καλύτερη δυνατή προσέγγιση της βέλτιστης λύσης. Φυσικά είναι στην ευχέρεια του χρήστη η αλλαγή των πιθανοτήτων αυτών.

Σε πρώτη φάση, καθορίζεται η ακολουθούμενη πυγολαμπίδα βάσει της ελκυστικότητάς της ως προς τη μετακινούμενη πυγολαμπίδα.

Για την μετακίνηση μίας πυγολαμπίδας προς μία άλλη χρησιμοποιείται η μέθοδος followAFirefly. Αυτή δέχεται ως είσοδο δύο αντικείμενα τύπου firefly1, ένα για το μετακινούμενο firefly και ένα για το ακολουθούμενο firefly, καθώς και τον πίνακα myJobs με τις πληροφορίες για τα jobs του προβλήματος.

Η μέθοδος αυτή, αρχικά προσδιορίζει τις κοινές αλληλουχίες εργασιών μεταξύ των δύο λύσεων καλώντας τη μέθοδο GetSeqInCommon. Αυτή, παίρνοντας ως είσοδο τα δύο fireflies επιστρέφει τις κοινές αλληλουχίες σε μορφή λίστας αντικειμένων JobOperationFirefly. Με τη σειρά τους οι αλληλουχίες αυτές εισάγονται σε λίστα στην οποία συγκεντρώνονται όλες οι κοινές αλληλουχίες εργασιών. Η λίστα αυτή δηλαδή είναι τύπου List<List<JobOperationFirefly>>.

Στο επόμενο βήμα της μεθόδου υπολογίζονται οι διαφορές μεταξύ των δύο πυγολαμπίδων στους χρόνους ολοκλήρωσης των jobs του προβλήματος και εισάγονται σε λίστα. Η πληροφορία αυτή είναι ιδιαίτερα χρήσιμη καθώς μπορεί να υποδείξει διαφορές μεταξύ των δύο πυγολαμπίδων και να επιτρέψει στην κινούμενη πυγολαμπίδα να προσεγγίσει την ακολουθούμενη καλύτερα.

Η ακέραιη μεταβλητή operLimit εκφράζει τον αριθμό των λειτουργιών της μετακινούμενης πυγολαμπίδας που θα υποστούν αλλαγές σε μία γενιά.

Για τον καθορισμό αυτών των λειτουργιών λαμβάνεται υπόψη η λίστα differenceInJobFinishTim.

Να σημειωθεί πως οι λειτουργίες που ανήκουν σε κάποιο από τις κοινές αλληλουχίες (sequences) των δυο πυγολαμπίδων δεν επιτρέπεται να υποστούν αλλαγές, αφού κάτι τέτοιο θα είχε ως αποτέλεσμα την αύξηση της απόστασης μεταξύ των δύο λύσεων και όχι την προσέγγιση της ακολουθούμενης λύσης από τη μετακινούμενη που είναι και το επιθυμητό.

Επίσης, εκτός της μη μετακίνησης λειτουργιών που ανήκουν σε κοινές αλληλουχίες εργασιών των δύο πυγολαμπίδων, λαμβάνεται υπόψη και η περίπτωση απόπειρας μετακίνησης μίας λειτουργίας σε θέση ενδιάμεση μιας υπάρχουσας κοινής αλληλουχίας εργασιών μεταξύ των δύο πυγολαμπίδων. Κάτι τέτοιο επίσης δεν είναι επιτρεπτό, για αυτό και σε περιπτώσεις κατά τις οποίες η νέα θέση μίας λειτουργίας μετά την μετακίνησή της θα ήταν τέτοια που θα διασπούσε ένα κοινό sequence, για την αποφυγή αυτής της διάσπασης η λειτουργία τοποθετείται τελικά αμέσως μετά την ολοκλήρωση της κοινής αλληλουχίας λειτουργιών.

Αφού επιλεχθεί λειτουργία προς μετακίνηση, πραγματοποιείται έλεγχος για το αν η λειτουργία αυτή εκτελείται από την ίδια μηχανή στις δύο λύσεις ή όχι. Σε περίπτωση που η λειτουργία εκτελείται σε διαφορετικές μηχανές, πραγματοποιείται μετακίνηση της λειτουργίας στην κινούμενη πυγολαμπίδα. Έτσι, η λειτουργία αφαιρείται από την μηχανή στην οποία βρισκόταν και τοποθετείται στη μηχανή στην οποία εκτελείται η λειτουργία αυτή στην ακολουθούμενη πυγολαμπίδα.

Η θέση στην οποία θα τοποθετηθεί η λειτουργία αυτή στην αλληλουχία της νέας μηχανής καθορίζεται με τρόπο παρόμοιο με αυτόν που περιγράφηκε για την περίπτωση της τυχαίας μετακίνησης της πυγολαμπίδας στις περιπτώσεις όπου μία λειτουργία αλλάζει μηχανή στην οποία εκτελείται.

Εφόσον μία λειτουργία προς μετακίνηση εκτελείται από την ίδια μηχανή τόσο στη μετακινούμενη όσο και στην ακολουθούμενη πυγολαμπίδα, η διαδικασία μετακίνησης πραγματοποιείται με διαφορετικό τρόπο. Δεδομένου ότι οι λειτουργίες που μετακινούμε ανήκουν σε εργασίες τις οποίες η ακολουθούμενη πυγολαμπίδα ολοκληρώνει νωρίτερα σε σύγκριση με τη μετακινούμενη πυγολαμπίδα, επιδιώκουμε την μείωση του χρόνου ολοκλήρωσης της εργασίας στη μετακινούμενη πυγολαμπίδα. Έτσι, οι λειτουργίες αυτής της εργασίας μετακινούνται σε νωρίτερες θέσεις της αλληλουχίας των λειτουργιών της μηχανής στην οποία βρίσκονται ανατεθειμένα. Ο αριθμός των θέσεων μετακίνησης στην αλληλουχία καθορίζεται με βάση τον συνολικό αριθμό των λειτουργιών που εκτελεί η συγκεκριμένη μηχανή σε αυτή τη λύση-μέλος του πληθυσμού. Η συνήθης τάξη μεγέθους του αριθμού των θέσεων μετακίνησης είναι 10% του συνολικού

αριθμού των λειτουργιών που εκτελεί η μηχανή. Ελάχιστος αριθμός θέσεων μετακίνησης είναι το 1, αφού για τιμή μετακίνησης 0 δεν θα είχαμε αλλαγή στο firefly. Σε περίπτωση κατά την οποία η μετακίνηση σε νωρίτερες θέσεις προκαλεί διάσπαση κοινού sequence εργασιών μεταξύ των δύο πυγολαμπίδων, αυτό προλαμβάνεται και ο αριθμός θέσεων μετακίνησης περιορίζεται σε τιμή τέτοια που να σταματά πριν τη διάσπαση της κοινής αλληλουχίας.

Μετά την μετακίνηση όλων των προς μετακίνηση λειτουργιών, ανανεώνονται οι χρόνοι του χρονοδιαγράμματος εκτέλεσης των εργασιών από τις μηχανές, με τη βοήθεια της μεθόδου updateFFtimes η οποία περιγράφηκε παραπάνω.

4.6.3 Επίλεκτα μέλη του πληθυσμού (Elites)

Στους εξελικτικούς αλγορίθμους, συχνά παίζει ιδιαίτερο ρόλο η ύπαρξη επίλεκτων μελών του πληθυσμού. Αυτά τα μέλη του πληθυσμού αποτελούν της καλύτερες λύσης κάθε γενιάς. Στα πλαίσια της ανάπτυξης του Αλγορίθμου Πυγολαμπίδων για το F-JSSP επιλέχθηκε να θεωρηθούν ως επίλεκτοι το 5% των καλύτερων λύσεων κάθε γενιάς. Οι λύσεις αυτές δεν αλλάζουν από γενιά σε γενιά, ωστόσο μπορούν να ακολουθηθούν από άλλα μέλη του πληθυσμού, ώστε αυτά να υιοθετήσουν στοιχεία που τις καθιστούν επιτυχημένες. Στην πορεία των επαναλήψεων όταν βρεθούν νέες, καλύτερες λύσεις, αυτές παίρνουν πλέον τον χαρακτηρισμό του επίλεκτου, και οι εκείνες που δεν είναι πλέον επίλεκτοι μπορούν να μετακινηθούν σε άλλους χώρους λύσεων.

Αφού μετακινηθούν όλες οι πυγολαμπίδες σε μια γενιά, πραγματοποιείται η αξιολόγηση των νέων ομοιοτήτων μεταξύ τους και καθορίζονται οι αποστάσεις μεταξύ τους με τη βοήθεια της μεθόδου getAllDistances.

Μέθοδος DeepClone

Σε αρκετές περιπτώσεις είναι αναγκαία η αποθήκευση μιας λύσης-firefly. Για το σκοπό αυτό αναπτύχθηκε η μέθοδος DeepClone, η οποία δημιουργεί βαθιά και ακριβή αντίγραφα του αντικειμένου ως έχει τη στιγμή της αντιγραφής. Οποιαδήποτε αλλαγή στο αρχικό αντικείμενο από το οποίο δημιουργήθηκε το αντίγραφο, δεν επηρεάζει με κανένα τρόπο το αντίγραφο εφόσον πραγματοποιηθεί μετά τη δημιουργία αυτού. Μιας και η μέθοδος αυτή χρησιμοποιεί εσωτερικά αντικείμενα της C# τύπου BinaryFormatter καθώς και την εσωτερική μέθοδο Serialize για αντικείμενα τύπου BinaryFormatter, είναι απαραίτητο οι κλάση firefly1 καθώς και όλες οι κλάσεις των αντικειμένων που περιέχονται μέσα στα αντικείμενα τύπου firefly1 να έχουν οριστεί ως

[Serializable], ειδάλλως δεν είναι δυνατή η δημιουργία αντιγράφων των fireflies με τη χρήση της μεθόδου DeepClone

Αποθήκευση αποτελεσμάτων και καλύτερης ευρεθείσας λύσης

Με την ολοκλήρωση της εκτέλεσης του αλγορίθμου, παράγονται τρία αρχεία τα οποία εξυπηρετούν στην καταγραφή του αποτελέσματος:

1. Ένα αρχείο .bin το οποίο περιέχει όλα τα χαρακτηριστικά της καλύτερης ευρεθείσας λύσης (αναθέσεις λειτουργιών σε μηχανές, αλληλουχίες εκτέλεσης λειτουργιών, χρόνους έναρξης και λήξης λειτουργιών κ.α). Το αρχείο αυτό μπορεί να αναγνωστεί από τον κώδικα για μελλοντική χρήση της λύσης.

2. Ένα αρχείο .txt με τον καλύτερο χρόνο που έχει βρεθεί ανά γενιά. Από το αρχείο αυτό προκύπτουν τα διαγράμματα σύγκλισης.

3. Ένα αρχείο .txt με την τιμή της νόρμας του πίνακα ομοιοτήτων μεταξύ των λύσεων ανά γενιά. Από αυτό το αρχείο προκύπτει το διάγραμμα της νόρμας ανά γενιά-επανάληψη για την απεικόνιση της συγκέντρωσης των λύσεων.

Διαγράμματα Gantt

Ο κώδικας υποστηρίζει την αυτόματη εξαγωγή του διαγράμματος Gantt της καλύτερης ευρεθείσας λύσης, αμέσως μετά το πέρας της εκτέλεσης των επαναλήψεων.

5. Αποτελέσματα και αξιολόγηση αποτελεσμάτων

Για να γίνει αξιολόγηση της αποτελεσματικότητας του Αλγορίθμου Πυγολαμπίδα, όπως αυτός αναπτύχθηκε στα πλαίσια αυτής της διπλωματικής εργασίας, για την επίλυση του F-JSSP, απαιτείται η εφαρμογή του αλγορίθμου σε προβλήματα αναφοράς (benchmark problems) της βιβλιογραφίας. Η βέλτιστη λύση για καθένα από τα προβλήματα αυτά είναι γνωστή και δίνεται από τους ίδιους τους δημιουργούς της κάθε εκδοχής του προβλήματος.

Τέτοιου είδους δοκιμαστικές εκδοχές του προβλήματος F-JSSP χρησιμοποιούνται για να ελεγχθεί πειραματικά το κατά πόσο προσεγγίζει η λύση που δίνει ένας εξελικτικός αλγόριθμος τη βέλτιστη λύση ενός προβλήματος. Επίσης γίνεται δυνατή η σύγκριση των αποτελεσμάτων του αλγορίθμου με άλλους αλγόριθμους της βιβλιογραφίας ως προς την προσέγγιση της βέλτιστης λύσης αλλά και ως προς τον χρόνο τρεξίματος και εσωτερικών επαναλήψεων (iterations) που απαιτούνται για την επίτευξη της παραγόμενης από τον αλγόριθμο λύσης.

Να σημειωθεί, βέβαια, πως ο χρόνος για την ολοκλήρωση ενός αριθμού επαναλήψεων του αλγορίθμου, πέρα από τη δομή του ίδιου του αλγορίθμου, εξαρτάται σε σημαντικό βαθμό από την ισχύ του επεξεργαστή και συνολικά το υλικό (hardware) του υπολογιστή στον οποίο εκτελείται ο αλγόριθμος.

5.1 Επιλογή προβλημάτων αναφοράς

Στη βιβλιογραφία εμφανίζονται αρκετές εκδοχές του F-JSSP. Πολλές από αυτές οποίες παρουσιάζουν ιδιαιτερότητες οι οποίες μπορούν να καταστήσουν διαφορετικούς αλγορίθμους περισσότερο ή λιγότερο αποτελεσματικούς για την επίλυσή τους. Ακόμη και η επιλογή των παραμέτρων στα πλαίσια ενός αλγορίθμου μπορεί να καταστήσει τον αλγόριθμο ιδιαίτερα αποτελεσματικό για την επίλυση μίας εκδοχής του F-JSSP και ταυτόχρονα αναποτελεσματικό για μια άλλη εκδοχή του προβλήματος.

Σε γενικές γραμμές είναι προτιμότερη η επιλογή παραμέτρων τέτοιων ώστε ο αλγόριθμος να παρουσιάζει ικανοποιητικά αποτελέσματα για ένα ευρύ φάσμα εκδοχών του προβλήματος, ακόμη και αν μία τέτοια φιλοσοφία στην παραμετροποίηση δύσκολα θα καταλήξει στην βέλτιστη απόδοση για εκδοχές του προβλήματος οι οποίες παρουσιάζουν ασυνήθιστες ιδιαιτερότητες.

Φυσικά, εφόσον κάποια εκδοχή του προβλήματος παρουσιάζει ιδιαίτερο ενδιαφέρον για τον ερευνητή, είναι δυνατό να εφαρμοστεί παραμετροποίηση τέτοια ώστε να ταιριάζει στη συγκεκριμένη εκδοχή. Με τον τρόπο αυτό υιοθετείται μια στενή (αντί ευρείας) σκοπιά έρευνας, η οποία πράγματι αποδεικνύεται συχνά ως η καλύτερη προσέγγιση για την επίλυση μεμονωμένων προβλημάτων.

Οι πρώτες δοκιμαστικές εκδοχές του F-JSSP εισήχθησαν από τον Brandimarte (Behnke & Geiger, 2012) και παρουσιάζουν καθεμία διαφορετικό βαθμό ευελιξίας της παραγωγής, δηλαδή διαφορετικό αριθμό πιθανών μηχανών ανά λειτουργία. Στη συνέχεια οι Hurink et al. και Chambers and Barnes δημιούργησαν εκδοχές στις οποίες οι λειτουργίες μπορούν να έχουν διαφορετική διάρκεια εκτέλεσης, ανάλογα με την επιλεχθείσα για τη λειτουργία μηχανή.

Τέλος, οι Kacem et al. παρουσίασαν εκδοχές του F-JSSP με πλήρη ευελιξία, στις οποίες κάθε λειτουργία του προβλήματος μπορεί να εκτελεστεί από οποιαδήποτε μηχανή, με διαφορετικούς χρόνους επεξεργασίας ανά μηχανή. Σε αυτές τις τέσσερις εκδοχές των Kacem et al. δοκιμάστηκε αρχικά ο Αλγόριθμος Πυγολαμπίδων που αναπτύχθηκε στα πλαίσια αυτής της εργασίας. Καθεμία από τις τέσσερις αυτές εκδοχές έχει διαφορετικό αριθμό εργασιών και λειτουργιών, και επομένως δραστικά διαφορετικό χώρο πιθανών λύσεων.

Προβλήματα αναφοράς – test instances

Οι Kacem et al. (2002) σχεδίασαν τέσσερις εκδοχές του F-JSSP με πλήρη ευελιξία και διαφοροποίηση στον αριθμό των λειτουργιών ανά εργασία. Τα δεδομένα των εκδοχών αυτών παρότι δε συμπεριλαμβάνονται στη βιβλιοθήκη OR library, βρίσκονται δημοσιευμένα σε σχετικό ιστότοπο (Kacem, Hammadi, & Borne, 2002b). Οι πίνακες που ακολουθούν δείχνουν τη διάρκεια εκτέλεσης κάθε λειτουργίας, ανάλογα με τη μηχανή που θα την εκτελέσει.

M→ Μηχανές (machines)

J→ εργασίες (jobs)

O→ λειτουργίες (operations)

		M1	M2	M3	M4	M5
J1	O1,1	2	5	4	1	2
	O1,2	5	4	5	7	5
	O1,3	4	5	5	4	5
J2	O2,1	2	5	4	7	8
	O2,2	5	6	9	8	5
	O2,3	4	5	4	54	5
J3	O3,1	9	8	6	7	9
	O3,2	6	1	2	5	4
	O3,3	2	5	4	2	4
	O3,4	4	5	2	1	5
J4	O4,1	1	5	2	4	12
	O4,2	5	1	2	1	2

Σχήμα 11:Πίνακας προβλήματος Kacem 1

		M1	M2	M3	M4	M5	M6	M7
J1	O1,1	1	4	6	9	3	5	2
	O1,2	8	9	5	4	1	1	3
	O1,3	4	8	10	4	11	4	3
J2	O2,1	6	9	8	6	5	10	3
	O2,2	2	10	4	5	9	8	4
J3	O3,1	15	4	8	4	8	7	1
	O3,2	9	6	1	10	7	1	6
	O3,3	11	2	7	5	2	3	14
J4	O4,1	2	8	5	8	9	4	3
	O4,2	5	3	8	1	9	3	6
	O4,3	1	2	6	4	1	7	2
J5	O5,1	7	1	8	5	4	3	9
	O5,2	2	4	5	10	6	4	9
	O5,3	5	1	7	1	6	6	2
J6	O6,1	8	7	4	56	9	8	4
	O6,2	5	14	1	9	6	5	8
	O6,3	3	5	2	5	4	5	7
J7	O7,1	5	6	3	6	5	15	2
	O7,2	6	5	4	9	5	4	3
	O7,3	9	8	2	8	6	1	7
J8	O8,1	6	1	4	1	10	4	3
	O8,2	11	13	9	8	9	10	8
	O8,3	4	2	7	8	3	10	7
J9	O9,1	12	5	4	5	4	5	5
	O9,2	4	2	15	99	4	7	3
	O9,3	9	5	11	2	5	4	2
J10	O10,1	9	3	13	10	7	6	8
	O10,2	4	4	25	3	8	1	2
	O10,3	1	2	6	11	13	3	5

Σχήμα 12:Πίνακας προβλήματος Kacem 2

		M1	M2	M3	M4	M5	M6	M7	M8	M9	M10
J1	O1,1	1	4	6	9	3	5	2	8	9	5
	O1,2	4	1	1	3	4	8	10	4	11	4
	O1,3	3	2	5	1	5	6	9	5	10	3
J2	O2,1	2	10	4	5	9	8	4	15	8	4
	O2,2	4	8	7	1	9	6	1	10	7	1
	O2,3	6	11	2	7	5	3	5	14	9	2
J3	O3,1	8	5	8	9	4	3	5	3	8	1
	O3,2	9	3	6	1	2	6	3	1	7	2
	O3,3	7	1	8	5	4	9	1	2	3	4
J4	O4,1	5	10	6	4	9	5	1	7	1	6
	O4,2	4	2	3	8	7	4	6	9	8	4
	O4,3	7	3	12	1	6	5	8	3	5	2
J5	O5,1	7	10	4	5	6	3	5	15	2	6
	O5,2	5	6	3	9	8	2	8	6	1	7
	O5,3	6	1	4	1	10	4	3	11	13	9
J6	O6,1	8	9	10	8	4	2	7	8	3	10
	O6,2	7	3	12	5	4	3	6	9	2	15
	O6,3	4	7	3	6	3	4	1	5	1	11
J7	O7,1	1	7	8	3	4	9	4	13	10	7
	O7,2	3	8	1	2	3	6	11	2	13	3
	O7,3	5	4	2	1	2	1	8	14	5	7
J8	O8,1	5	7	11	3	2	9	8	5	12	8
	O8,2	8	3	10	7	5	13	4	6	8	4
	O8,3	6	2	13	5	4	3	5	7	9	5
J9	O9,1	3	9	1	3	8	1	6	7	5	4
	O9,2	4	6	2	5	7	3	1	9	6	7
	O9,3	8	5	4	8	6	1	2	3	10	12
J10	O10,1	4	3	1	6	7	1	2	6	20	6
	O10,2	3	1	8	1	9	4	1	4	17	15
	O10,3	9	2	4	2	3	5	2	4	10	23

Σχήμα 13: Πίνακας προβλήματος Kacem 3

	K4	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10
J1	O1,1	1	4	6	9	3	5	2	8	9	4
	O1,2	1	1	3	4	8	10	4	11	4	3
	O1,3	2	5	1	5	6	9	5	10	3	2
	O1,4	10	4	5	9	8	4	15	8	4	4
J2	O2,1	4	8	7	1	9	6	1	10	7	1
	O2,2	6	11	2	7	5	3	5	14	9	2
	O2,3	8	5	8	9	4	3	5	3	8	1
	O2,4	9	3	6	1	2	6	4	1	7	2
J3	O3,1	7	1	8	5	4	9	1	2	3	4
	O3,2	5	10	6	4	9	5	1	7	1	6
	O3,3	4	2	3	8	7	4	6	9	8	4
	O3,4	7	3	12	1	6	5	8	3	5	2
J4	O4,1	6	2	5	4	1	2	3	6	5	4
	O4,2	8	5	7	4	1	2	36	5	8	5
	O4,3	9	6	2	4	5	1	3	6	5	2
	O4,4	11	4	5	6	2	7	5	4	2	1
J5	O5,1	6	9	2	3	5	8	7	4	1	2
	O5,2	5	4	6	3	5	2	28	7	4	5
	O5,3	6	2	4	3	6	5	2	4	7	9
	O5,4	6	5	4	2	3	2	5	4	7	5
J6	O6,1	4	1	3	2	6	9	8	5	4	2
	O6,2	1	3	6	5	4	7	5	4	6	5
J7	O7,1	1	4	2	5	3	6	9	8	5	4
	O7,2	2	1	4	5	2	3	5	4	2	5
J8	O8,1	2	3	6	2	5	4	1	5	8	7
	O8,2	4	5	6	2	3	5	4	1	2	5
	O8,3	3	5	4	2	5	49	8	5	4	5
	O8,4	1	2	36	5	2	3	6	4	11	2
J9	O9,1	6	3	2	22	44	11	10	23	5	1
	O9,2	2	3	2	12	15	10	12	14	18	16
	O9,3	20	17	12	5	9	6	4	7	5	6
	O9,4	9	8	7	4	5	8	7	4	56	2
J10	O10,1	5	8	7	4	56	3	2	5	4	1
	O10,2	2	5	6	9	8	5	4	2	5	4
	O10,3	6	3	2	5	4	7	4	5	2	1
	O10,4	3	2	5	6	5	8	7	4	5	2
J11	O11,1	1	2	3	6	5	2	1	4	2	1
	O11,2	2	3	6	3	2	1	4	10	12	1
	O11,3	3	6	2	5	8	4	6	3	2	5
	O11,4	4	1	45	6	2	4	1	25	2	4
J12	O12,1	9	8	5	6	3	6	5	2	4	2
	O12,2	5	8	9	5	4	75	63	6	5	21
	O12,3	12	5	4	6	3	2	5	4	2	5
	O12,4	8	7	9	5	6	3	2	5	8	4
J13	O13,1	4	2	5	6	8	5	6	4	6	2
	O13,2	3	5	4	7	5	8	6	6	3	2
	O13,3	5	4	5	8	5	4	6	5	4	2
	O13,4	3	2	5	6	5	4	8	5	6	4

J14	O14,1	2	3	5	4	6	5	4	85	4	5
	O14,2	6	2	4	5	8	6	5	4	2	6
	O14,3	3	25	4	8	5	6	3	2	5	4
	O14,4	8	5	6	4	2	3	6	8	5	4
J15	O15,1	2	5	6	8	5	6	3	2	5	4
	O15,2	5	6	2	5	4	2	5	3	2	5
	O15,3	4	5	2	3	5	2	8	4	7	5
	O15,4	6	2	11	14	2	3	6	5	4	8

Σχήμα 14:Πίνακας προβλήματος Kacem 4

5.2 Συγκέντρωση λυσεων (convergence)

Στον Αλγόριθμο Πυγολαμπίδων, όπως και σε όλες τις τεχνικές βελτιστοποίησης που βασίζονται στη φιλοσοφία της νοημοσύνης σμήνους, παρουσιάζει ιδιαίτερο ενδιαφέρον η συγκέντρωση των μελών του πληθυσμού, δηλαδή των λύσεων του προβλήματος, γύρω από ένα συγκεκριμένο χώρο λύσεων.

Για το σκοπό αυτό εισήχθη στον αλγόριθμο που αναπτύχθηκε η έννοια της νόρμας του πίνακα theDistances, ο οποίος δείχνει την ομοιότητα μεταξύ κάθε ζεύγους δύο μελών του πληθυσμού. Έτσι, όσο αυξάνονται οι ομοιότητες μεταξύ των μελών του πληθυσμού, κάτι που δείχνει συγκέντρωση των λύσεων σε σμήνη σε συγκεκριμένες περιοχές του χώρου λύσεων, η τιμή της νόρμας του πίνακα αυξάνεται και αυτή.

Συγκεκριμένα, επιλέχθηκε να υπολογίζεται η νόρμα του Frobenius.

Η νόρμα Frobenius ορίζεται ως εξής:

Για ένα πίνακα m x n , η νόρμα A ισούται με τη ρίζα του αθροίσματος του τετραγώνου της απόλυτης τιμής των στοιχείων του πίνακα.

$$\|\mathbf{A}\|_F \equiv \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}\left|a_{i\,j}\right|^2}$$

5.3 Παρουσίαση αποτελεσμάτων

Στα πλαίσια της εργασίας αυτής, επιλέχθηκε ως στόχος της βελτιστοποίησης με χρήση του Αλγορίθμου Πυγολαμπίδων η ελαχιστοποίηση του χρόνου στον οποίο θα έχουν ολοκληρωθεί όλες οι λειτουργίες όλων των εργασιών της εκάστοτε εκδοχής του προβλήματος (makespan). Ο στόχος αυτός αποτελεί και τον πλέον διαδεδομένο παράγοντα βελτιστοποίησης στην υπάρχουσα βιβλιογραφία για το F-JSSP και τα συγγενικά του προβλήματα.

Οι ελάχιστες τιμές για τον χρόνο ολοκλήρωσης όλων των εργασιών δίνονται από τους ίδιους τους σχεδιαστές του προβλήματος [] και παρατίθενται στον παρακάτω πίνακα:

Όνομα εκδοχής	Αριθμός εργασιών n	Αριθμός Μηχανών m	Συνολικός αριθμός λειτουργιών	Ελάχιστος χρόνος ολοκλήρωσης εργασιών
Εκδοχή Kacem 1 (μικρού μεγέθους)	4	5	12	11
Εκδοχή Kacem 2 (μεσαίου μεγέθους)	10	7	29	11
Εκδοχή Kacem 3 (μεσαίου μεγέθους)	10	10	30	7
Εκδοχή Kacem 4 (μεγάλου μεγέθους)	15	10	56	12

Σχήμα 15: Συνοπτική παρουσίαση προβλημάτων αναφοράς

Εκδοχή προβλήματος	Ελάχιστος χρόνος ολοκλήρωσης	Ελάχιστος χρόνος ΑΠ	Μέσο αποτέλεσμα ΑΠ	Μέσος όρος γενεών ευρεθήσας λύσης
Kacem 1	11	11	11	173
Kacem 2	11	12	13	342
Kacem 3	7	7	8.5	418
Kacem 4	12	16	18	6410

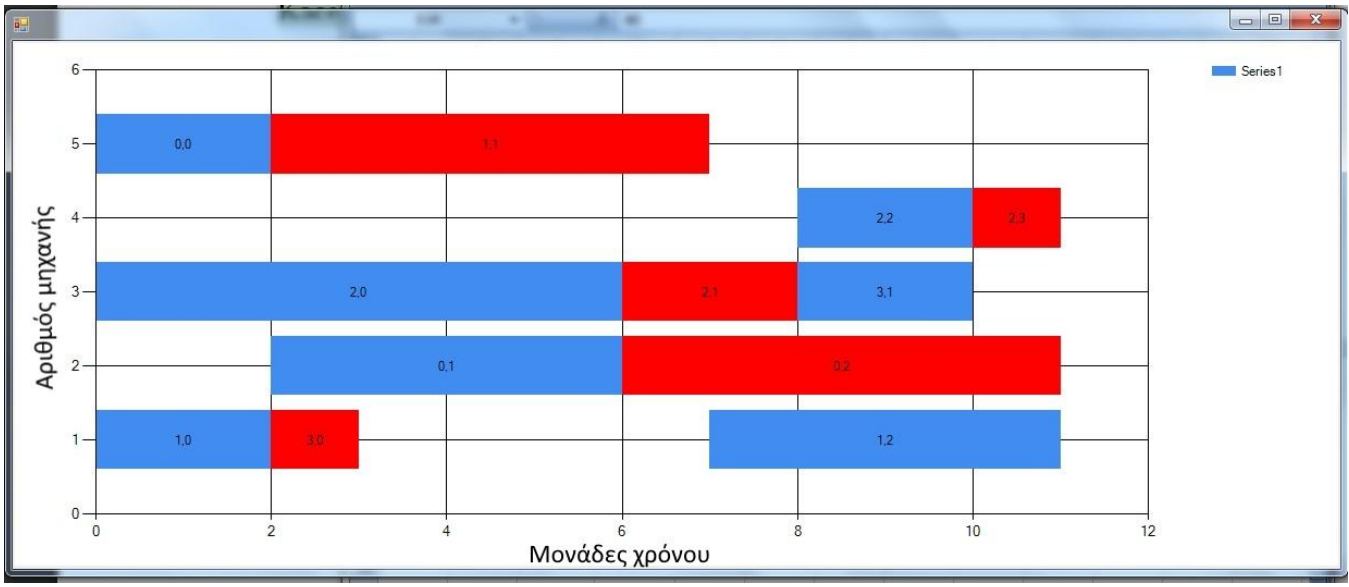
Σχήμα 16:Αποτελέσματα Αλγ. Πυγολαμπίδας στα προβλήματα αναφοράς

Η παρουσίαση των αποτελεσμάτων για κάθε εκδοχή του προβλήματος αποτελείται από:

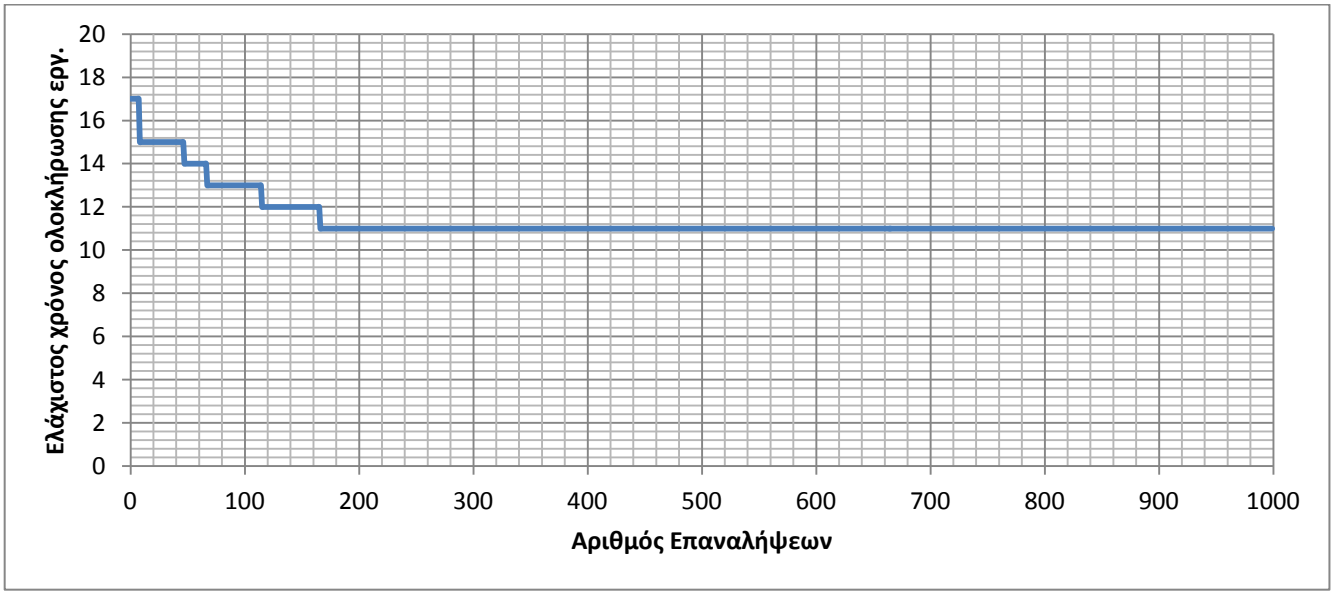
1. Ένα διάγραμμα Gantt της καλύτερης ευρεθείσας λύσης. Στον κατακόρυφο άξονα βρίσκεται ο αριθμός της μηχανής, ενώ στον οριζόντιο οι μονάδες χρόνου. Επάνω στις μπάρες αναφέρεται ο αριθμός εργασίας και λειτουργίας στην οποία αντιστοιχεί η κάθε μπάρα.
2. Ένα διάγραμμα σύγκλισης στο οποίο φαίνεται η καλύτερη λύση που έχει βρεθεί ανά γενιά-επανάληψη της εκτέλεσης του αλγορίθμου
3. Ένα διάγραμμα της νόρμας του πίνακα ομοιότητας μεταξύ των λύσεων. Το μέγεθος αυτό, όπως έχει περιγραφεί σε προηγούμενη παράγραφο, αντικατοπτρίζει τη συγκέντρωση των λύσεων.

Πρόβλημα Kacem1

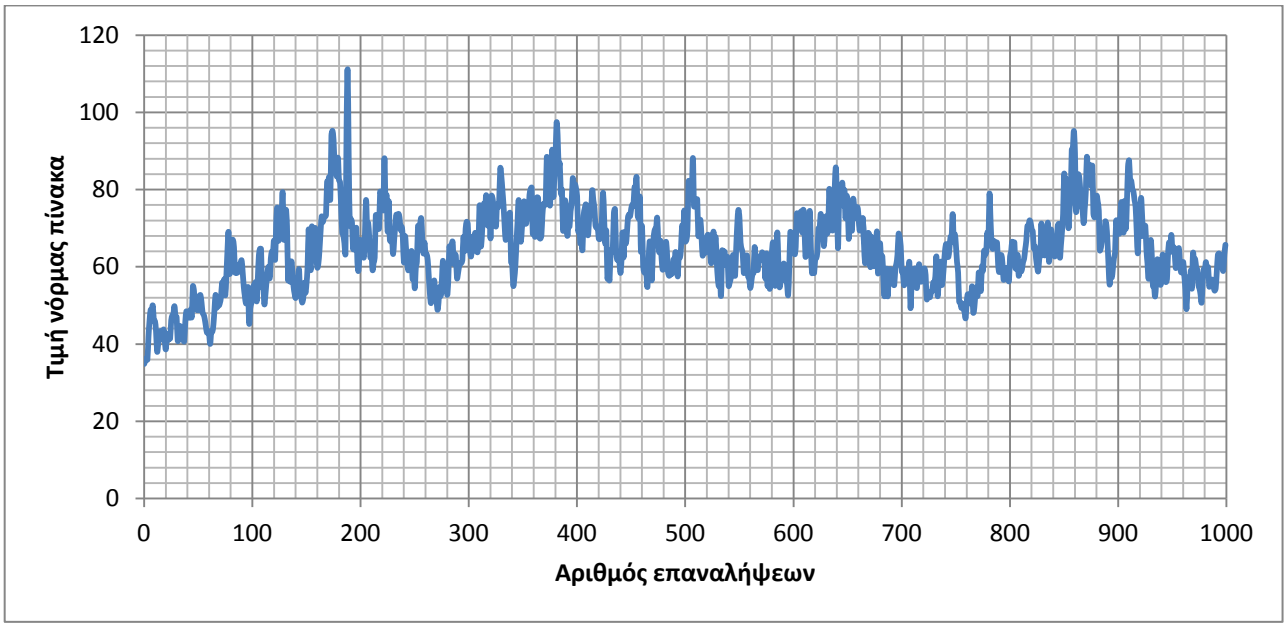
Διάγραμμα Gantt καλύτερης λύσης:



Διάγραμμα σύγκλισης (Καλύτερη λύση που είχε βρεθεί ανά γενιά):

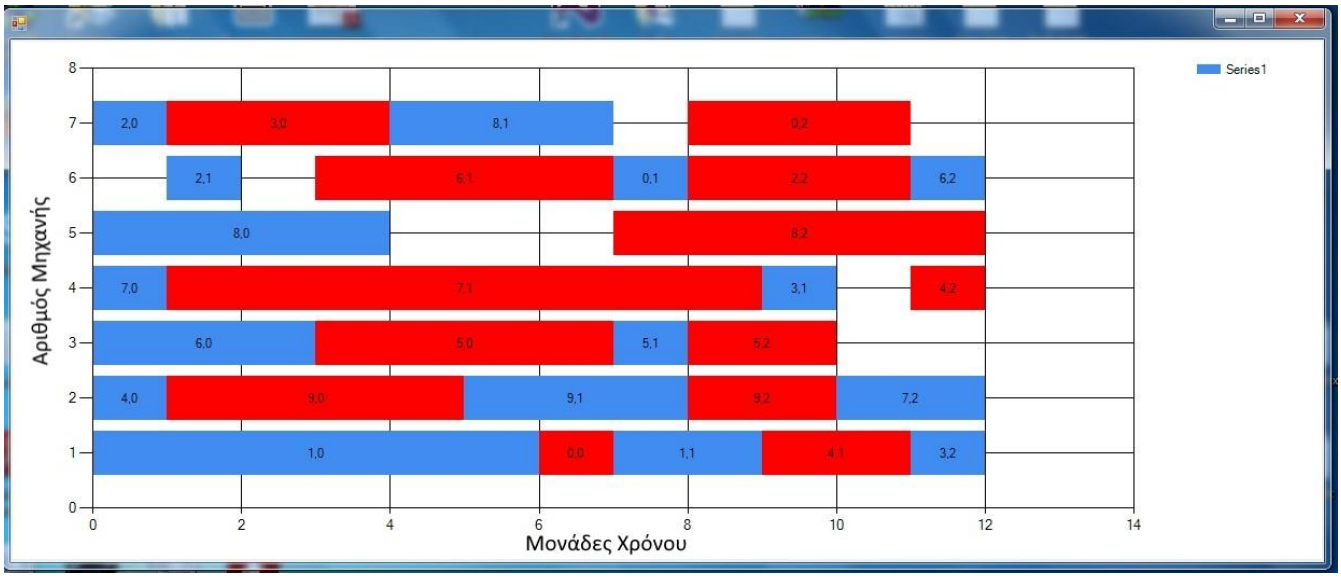


Διάγραμμα νόρμας πίνακα ομοιοτήτων μεταξύ των λύσεων (συγκέντρωση λύσεων):

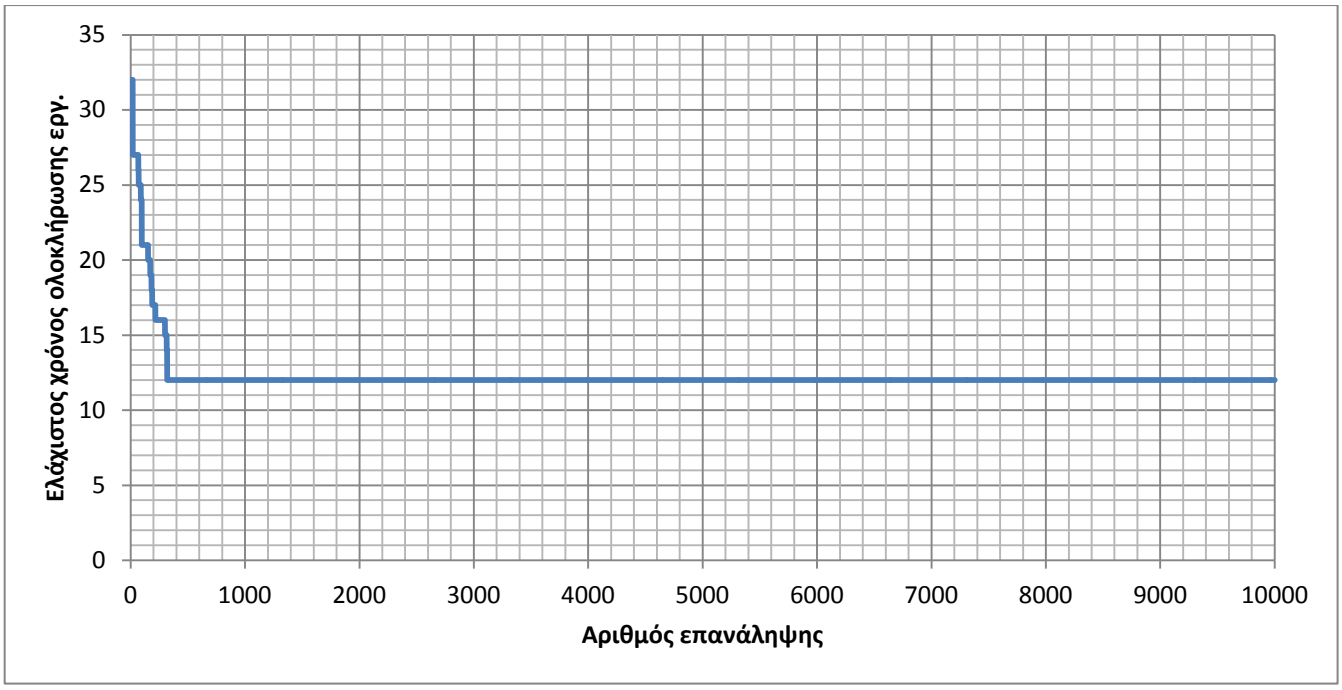


Πρόβλημα Kacem 2

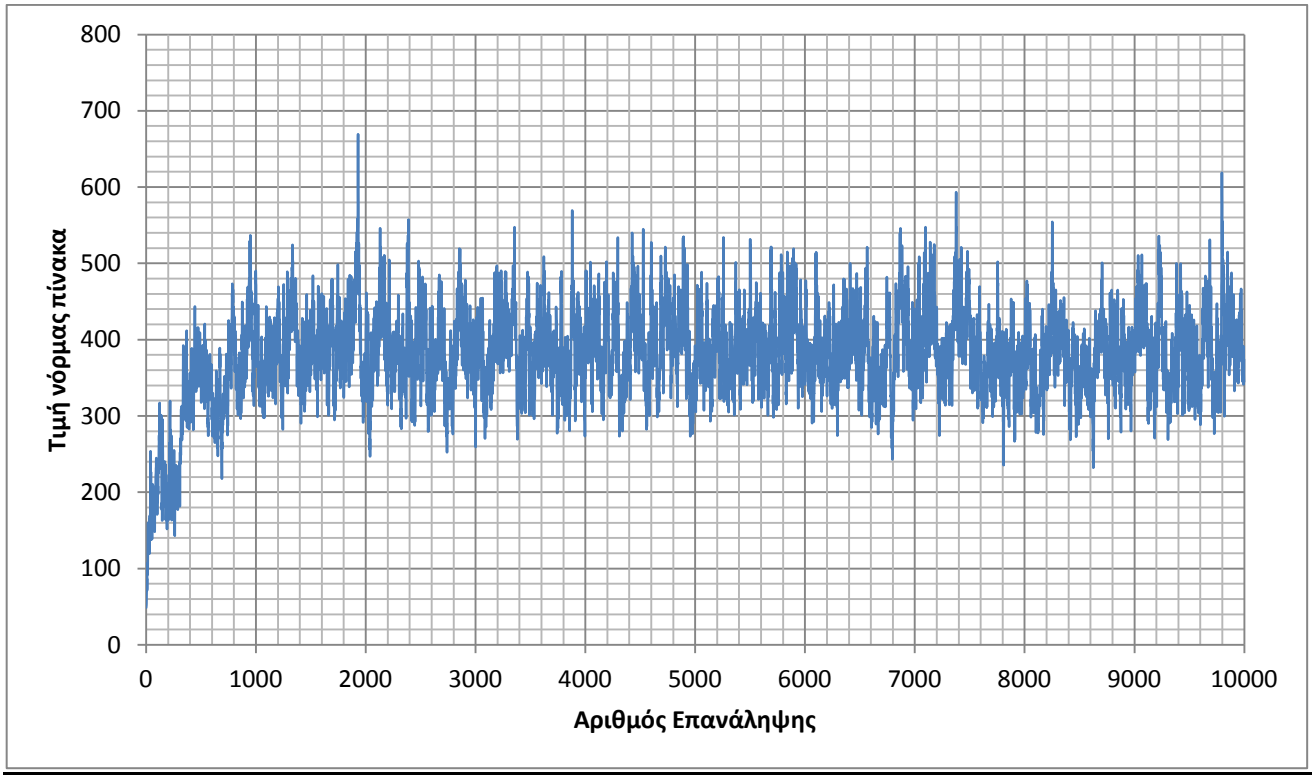
Διάγραμμα Gantt:



Διάγραμμα σύγκλισης (Καλύτερη λύση που είχε βρεθεί ανά γενιά):

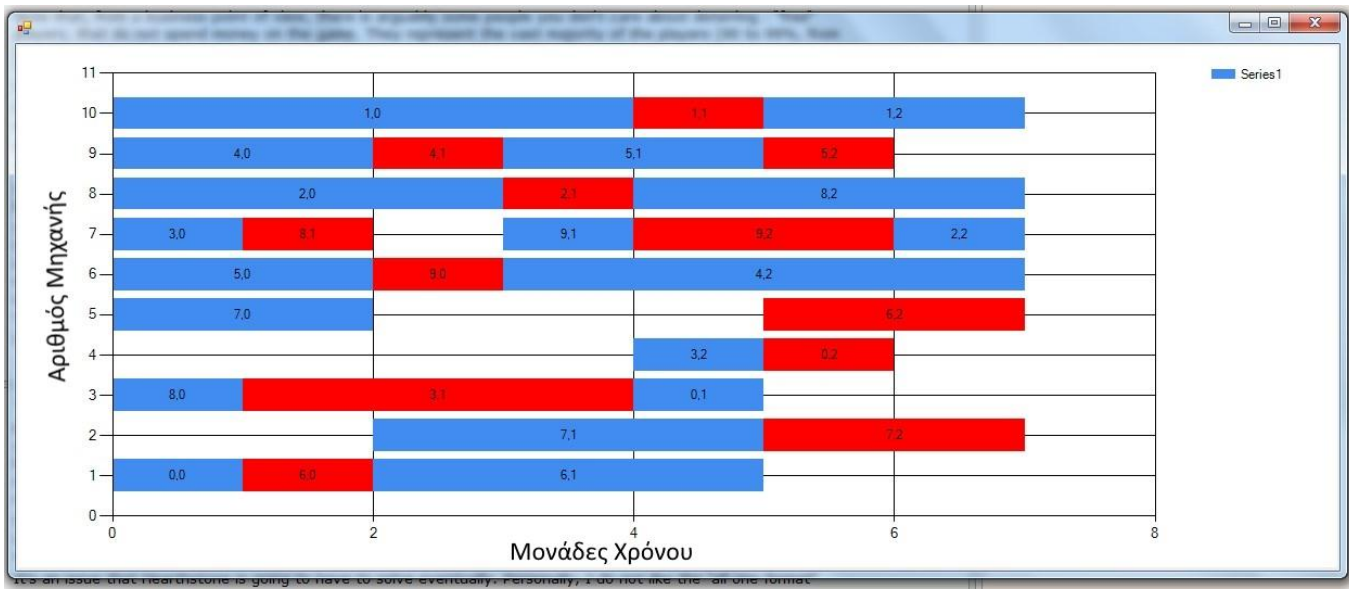


Διάγραμμα νόρμας πίνακα ομοιοτήτων μεταξύ των λύσεων (συγκέντρωση λύσεων):

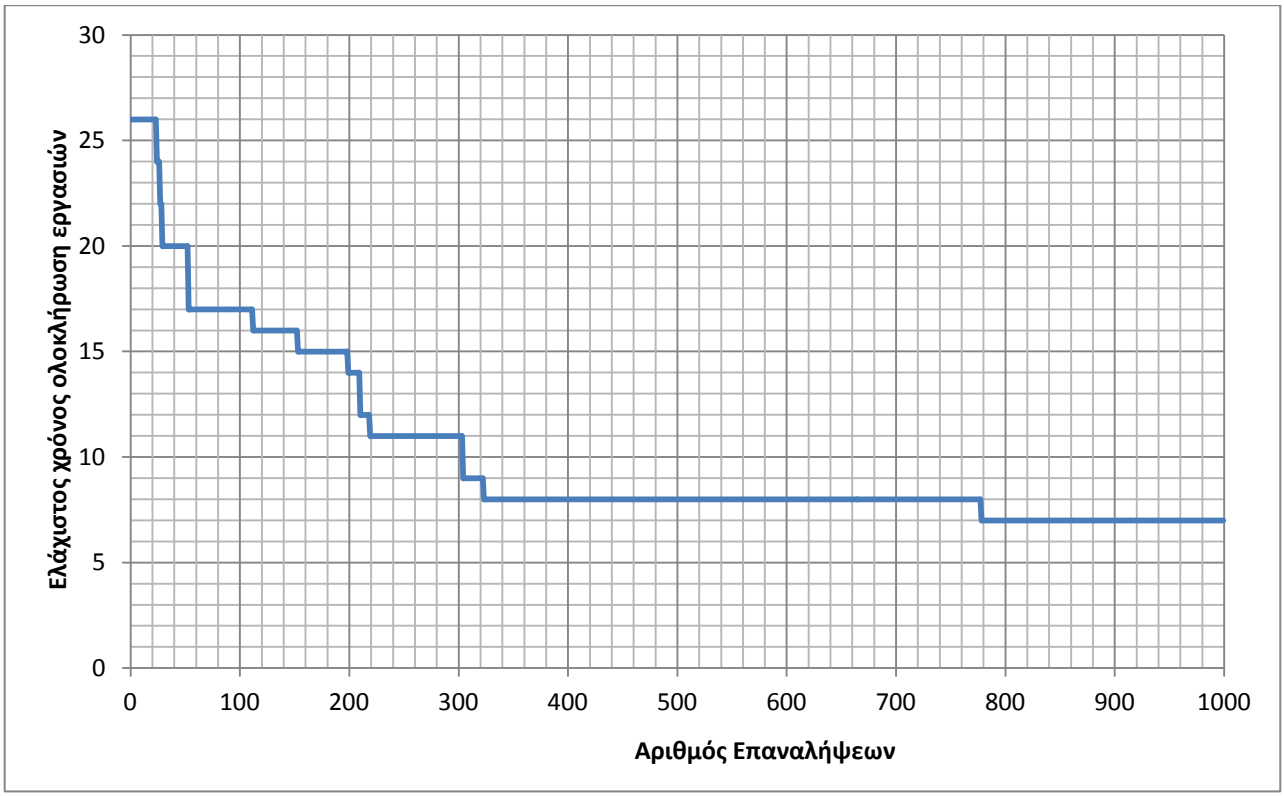


Πρόβλημα Kacem3

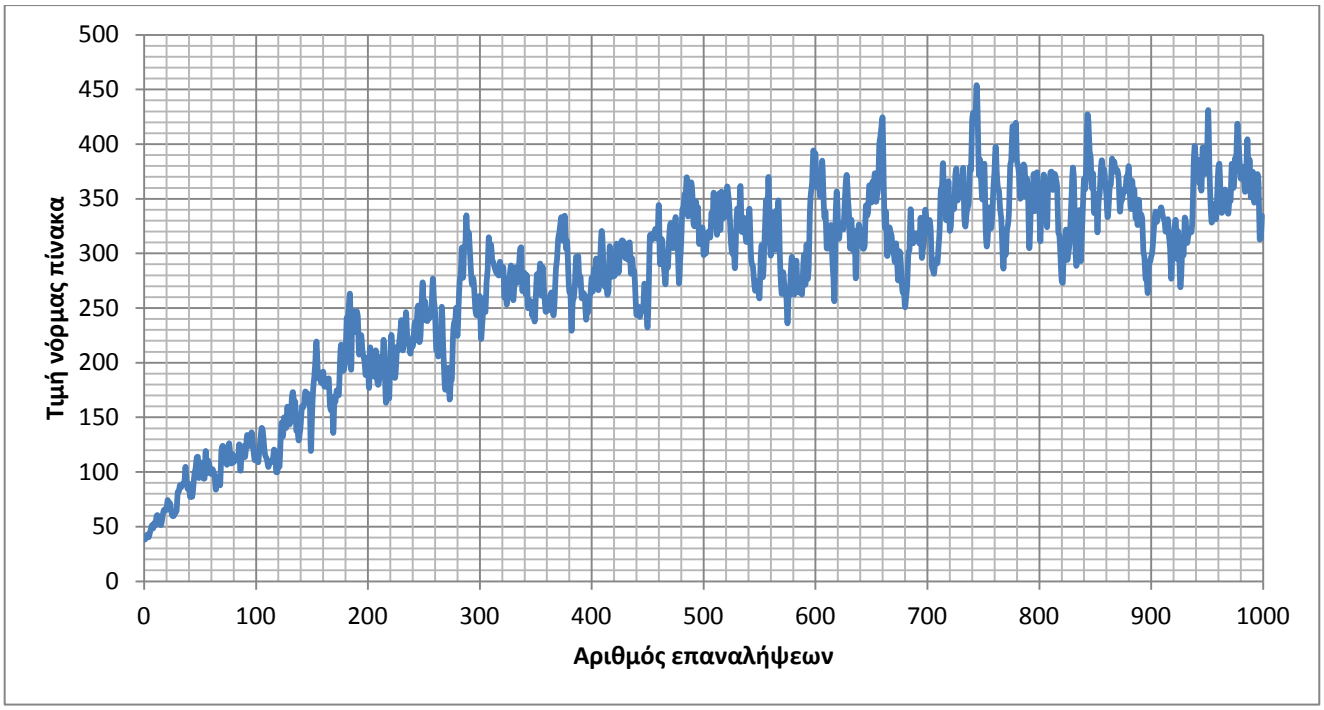
Διάγραμμα Gantt:



Διάγραμμα σύγκλισης (Καλύτερη λύση που είχε βρεθεί ανά γενιά):

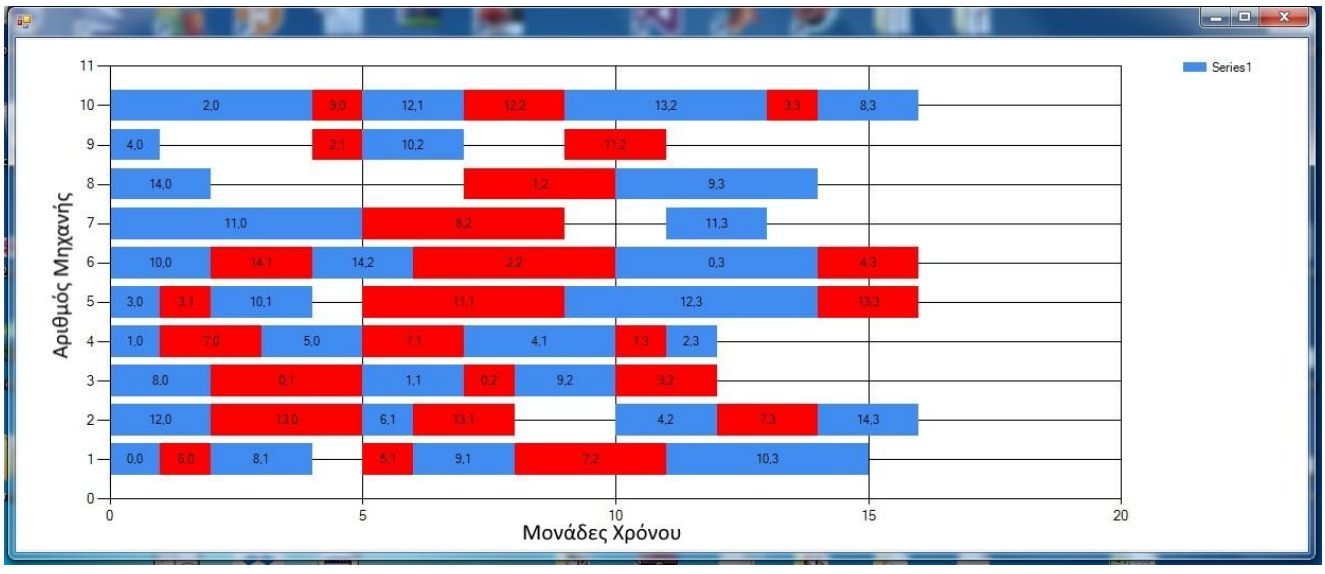


Διάγραμμα νόρμας πίνακα ομοιοτήτων μεταξύ των λύσεων:

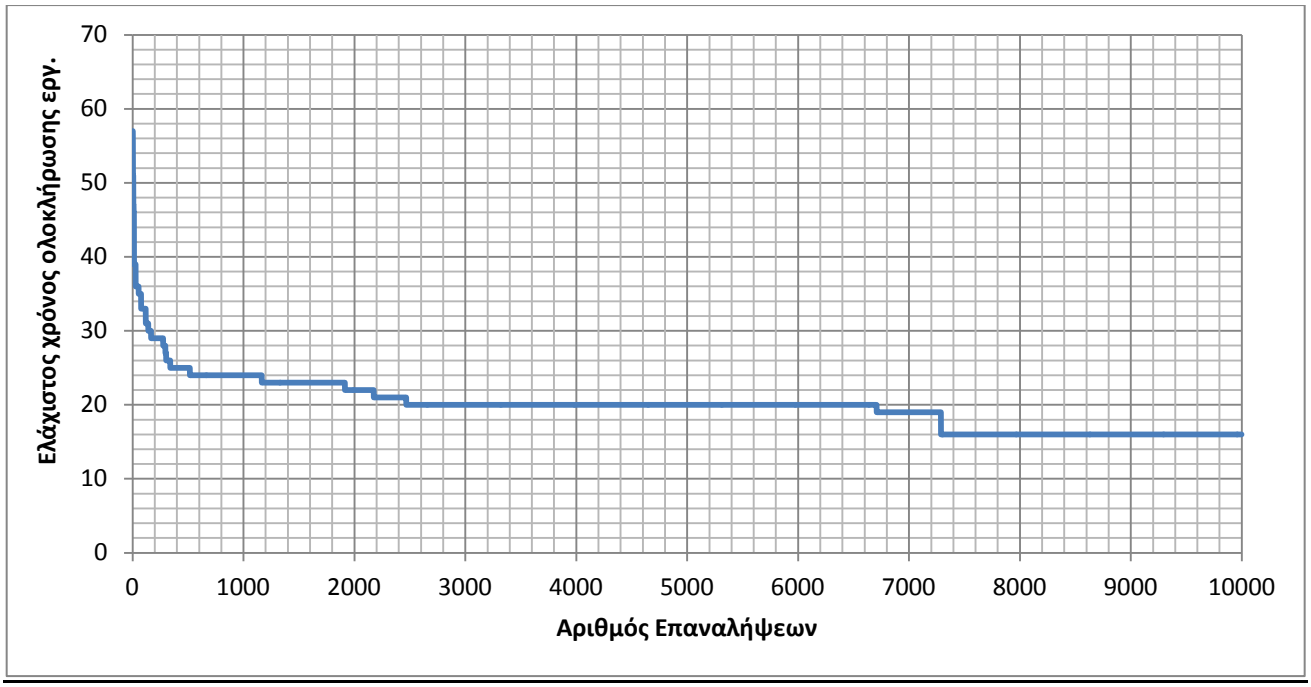


Πρόβλημα Kacem 4

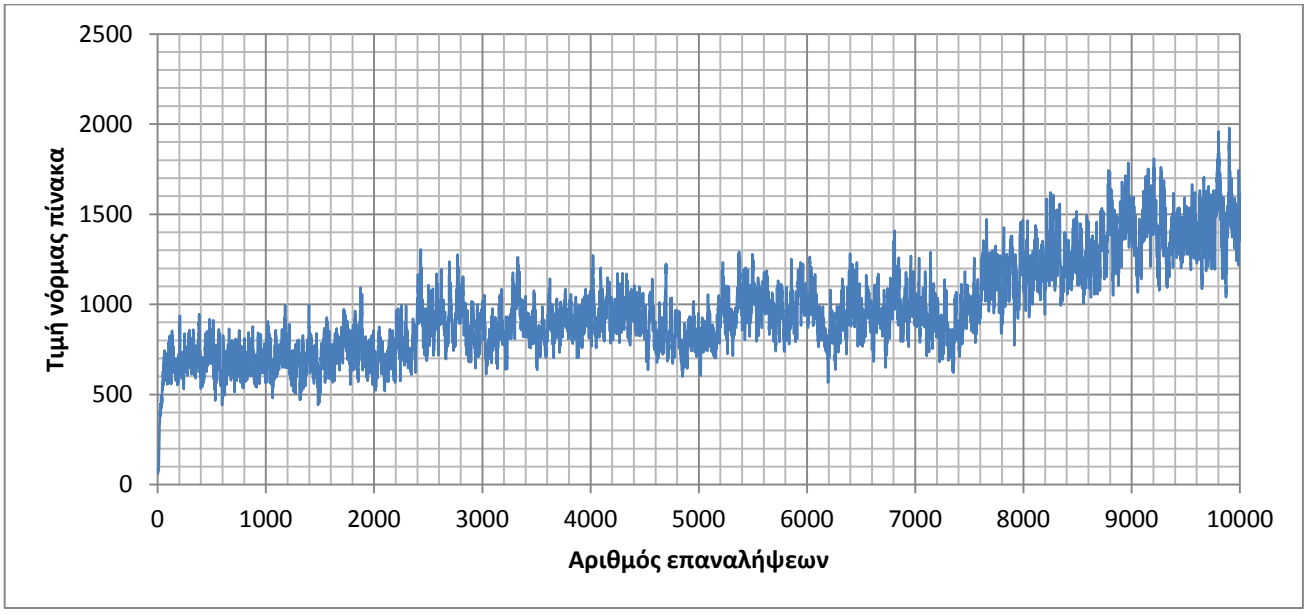
Διάγραμμα Gantt:



Διάγραμμα σύγκλισης (Καλύτερη λύση που είχε βρεθεί ανά γενιά):



Διάγραμμα νόρμας πίνακα ομοιοτήτων μεταξύ των λύσεων:



5.4 Επιλογή παραμέτρων του αλγορίθμου

Η επιλογή των τιμών για τις παραμέτρους που εμφανίζονται στον αλγόριθμο που σχεδιάστηκε έχει ιδιαίτερη σημασία. Ανάλογα με το μέγεθος και τις ιδιαιτερότητες που παρουσιάζει κάθε εκδοχή του προβλήματος, διαφορετικές επιλογές για τις τιμές των παραμέτρων μπορούν να οδηγήσουν σε δραστικά διαφορετικά αποτελέσματα.

Έτσι, δεν μπορεί να σταθεί ο ισχυρισμός πως υπάρχουν βέλτιστες τιμές για τις παραμέτρους οι οποίες ικανοποιούν όλες τις εκδοχές του F-JSSP. Για παράδειγμα, μία εκδοχή του F-JSSP μπορεί να παρουσιάζει πολύ μεγάλες διαφορές ανάμεσα στους χρόνους επεξεργασίας της ίδιας λειτουργίας από διαφορετικές μηχανές του προβλήματος, καθιστώντας έτσι την επιλογή μίας γρήγορης μηχανής ιδιαίτερα σημαντική. Από την άλλη, μία άλλη εκδοχή του F-JSSP μπορεί να διαθέτει μία μηχανή η οποία να είναι η ταχύτερη για την επεξεργασία πολλών λειτουργιών του προβλήματος. Αν όμως ανατεθεί στη μηχανή αυτή μεγάλος αριθμός λειτουργιών υπάρχει κίνδυνος η μηχανή αυτή να καταλήξει επιβαρυμένη με ένα μεγάλο φόρτο εργασίας, αφήνοντας τις άλλες μηχανές του προβλήματος με μεγάλους χρόνους στους οποίους δεν εκτελούν καμία λειτουργία (idle time), οδηγώντας τελικά σε μεγαλύτερο συνολικό χρόνο για την ολοκλήρωση των εργασιών. Γίνεται σαφές λοιπόν, πως τα δύο αυτά προβλήματα ανταποκρίνονται καλύτερα σε πολύ διαφορετικές επιλογές για τις τιμές των παραμέτρων του αλγορίθμου.

Μέγεθος πληθυσμού και αριθμός επαναλήψεων (iterations)

Οι πιο βασικοί παράγοντες που καθορίζουν το χρόνο εκτέλεσης του κώδικα για την εύρεση μίας λύσης είναι τρεις. Πρώτον, το μέγεθος της εκδοχής του προβλήματος, δηλαδή το πόσες μηχανές υπάρχουν, πόσες εργασίες προς ολοκλήρωση και πόσες λειτουργίες πρέπει να εκτελεστούν για την ολοκλήρωση της κάθε εργασίας. Ακόμη, το μέγεθος του προβλήματος εξαρτάται και από τον αριθμό των διαφορετικών μηχανών που δύνανται να εκτελέσουν την κάθε λειτουργία.

Οι άλλοι δύο παράγοντες είναι ο αριθμός των μελών του πληθυσμού και το πλήθος των γενεών για τις οποίες θα εκτελεστεί ο κώδικας. Οι δύο αυτοί παράγοντες είναι αλληλένδετοι, αφού το γινόμενο τους καθορίζει ουσιαστικά το πλήθος των διαφορετικών πιθανών λύσεων του προβλήματος που θα εξεταστούν και μεταξύ αυτών να επιλεγεί η καλύτερη.

Όπως προκύπτει και από τη βιβλιογραφία, το μέγεθος του πληθυσμού κυμαίνεται συνήθως στο εύρος από 15 έως 100 πυγολαμπίδες (X.-S. Yang, 2014b). Στα πλαίσια του αλγορίθμου της παρούσας εργασίας και για τις διάφορες εκδοχές του F-JSSP δοκιμάστηκαν τιμές από 30 έως 70.

Για τον αριθμό των γενεών μέχρι την ολοκλήρωση της εκτέλεσης του κώδικα χρειάστηκε να δοκιμαστεί μεγάλο εύρος τιμών. Αυτό γιατί, όπως αναμενόταν,

ενώ για τις μικρότερες εκδοχές του προβλήματος η βέλτιστη λύση με πληθυσμό 40 πυγολαμπίδων είχε βρεθεί σε λιγότερες από 800 επαναλήψεις, δεν ίσχυε το ίδιο για εκδοχές με μεγάλο αριθμό λειτουργιών προς ανάθεση και με πολλές μηχανές που δύνανται να τις εκτελέσουν.

6. Συμπεράσματα

Για όλες τις εκτελέσεις του αλγορίθμου για τις διαφορετικές εκδοχές του προβλήματος, παρατηρείται σημαντική μείωση του χρόνου ολοκλήρωσης των εργασιών με την πάροδο των γενεών-επαναλήψεων, όπως γίνεται εμφανές και από τα διαγράμματα σύγκλισης. Έτσι, ο αλγόριθμος επιβεβαιώνει τον εξελικτικό του χαρακτήρα.

Επίσης, για όλες τις εκτελέσεις του αλγορίθμου παρατηρείται αύξηση της νόρμας του πίνακα των ομοιοτήτων μεταξύ των πυγολαμπίδων, όπως φαίνεται και στα σχετικά διαγράμματα. Το γεγονός αυτό επιβεβαιώνει πως με την πάροδο τον γενεών οι πυγολαμπίδες αρχίζουν να παρουσιάζουν περισσότερες ομοιότητες μεταξύ τους, δηλαδή περισσότερα κοινά στοιχεία στην ανάθεση λειτουργιών σε μηχανές και στην αλληλουχία των λειτουργιών. Στην πράξη αυτό σημαίνει πως οι αποστάσεις μεταξύ των λύσεων γίνονται μικρότερες, δηλαδή με την πάροδο των γενεών οι πυγολαμπίδες-λύσεις αρχίζουν να δημιουργούν «σμήνη» και να συγκεντρώνονται γύρω από ακρότατα του χώρου λύσεων, κάτι που ήταν εξαρχής επιθυμητό και αποτελεί θεμελιώδες χαρακτηριστικό των εξελικτικών αλγορίθμων εν γένει. Όταν, σε μικρά κυρίως προβλήματα, παρατηρείται παύση της αυξητικής τάσης της νόρμας του πίνακα, συνήθως αυτό συμβαίνει διότι η βέλτιστη λύση έχει ήδη βρεθεί. Στις μεγάλες εκδοχές του προβλήματος παρατηρείται διαρκώς αυξητική τάση της νόρμας του πίνακα ομοιοτήτων μεταξύ των πυγολαμπίδων.

Παρατηρείται ότι στις μικρότερες εκδοχές του προβλήματος ο αλγόριθμος επιτυγχάνει με συνέπεια τη βέλτιστη λύση σε ικανοποιητικό αριθμό επαναλήψεων, οι οποίες αντιστοιχούν σε λίγα λεπτά εκτέλεσης του αλγορίθμου.

Σε μεγαλύτερες εκδοχές του προβλήματος (όπως η εκδοχή Kacem 4) εμφανίζονται σημαντικές αποκλίσεις από τη βέλτιστη λύση, κάτι που ως ένα σημείο αναμενόταν λόγω της ραγδαίας μεγέθυνσης του χώρου λύσεων όσο αυξάνονται οι εργασίες, οι λειτουργίες και οι μηχανές του προβλήματος. Να σημειωθεί βέβαια στο σημείο αυτό πως η συγκεκριμένη εκδοχή παρουσιάζει πολύ μεγάλο εύρος μηχανών που μπορούν να επιλεχθούν για κάθε λειτουργία, συγκριτικά με άλλα προβλήματα αναφοράς της βιβλιογραφίας (Behnke & Geiger, 2012). Συγκεκριμένα, κάθε λειτουργία μπορεί να εκτελεστεί από οποιαδήποτε από τις 10 διαφορετικές μηχανές του προβλήματος, υπάρχει δηλαδή πλήρης ευελιξία στην επιλογή μηχανών, κάτι που καθιστά την επίτευξη της βέλτιστης λύσης σαφώς δυσκολότερη. Πιθανόν μία προσαρμοσμένη παραμετροποίηση του αλγορίθμου προσαρμοσμένη στα δεδομένα της συγκεκριμένης εκδοχής να μπορεί να αποδώσει καλύτερα αποτελέσματα, ωστόσο κάτι τέτοιο δεν πραγματοποιήθηκε στα πλαίσια της παρούσας εργασίας, καθώς θα αναιρούσε το σκοπό της δημιουργίας ενός αλγορίθμου ικανού να ανταποκριθεί σε διαφορετικές εκδοχές του F-JSSP.

Σε γενικές γραμμές η λειτουργία του κώδικα που αναπτύχθηκε για την εκτέλεση του αλγορίθμου που σχεδιάστηκε κρίνεται ικανοποιητική, αφού η ελαχιστοποίηση του χρόνου ολοκλήρωσης των εργασιών επιτυγχάνεται είτε πλήρως, είτε σε ένα λογικό βαθμό, ανάλογα με την εκδοχή του προβλήματος.

Μελλοντικά κρίνεται σκόπιμη η εφαρμογή του αλγορίθμου σε περισσότερες, κυρίως μεγαλύτερες, εκδοχές του προβλήματος για τη διερεύνηση της απόδοσής του σε αυτές. Αναμένεται, μιας και στις περισσότερες εκδοχές η ευελιξία στις επιλογές των μηχανών είναι μικρότερη από εκείνη της εκδοχής Kacem4, η προσέγγιση της βέλτιστης λύσης να είναι ευκολότερη και ταχύτερη.

Τέλος, αναμένεται η δοκιμή περαιτέρω διαφορετικών τιμών για τις υπάρχουσες παραμέτρους του αλγορίθμου, ή και η εισαγωγή νέων παραμέτρων, να βελτιώσει τελικά την απόδοση του αλγορίθμου, τουλάχιστον για ορισμένες εκδοχές προβλημάτων.

Βιβλιογραφικές αναφορές

Adams, J., Balas, E., & Zawack, D. (1988). The shifting bottleneck procedure for job shop scheduling. *Management science, 34*(3), 391-401.

Allahverdi, A., Ng, C., Cheng, T. E., & Kovalyov, M. Y. (2008). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research, 187*(3), 985-1032.

ASHOUR, S., & Hiremath, S. (1973). A branch-and-bound approach to the job-shop scheduling problem. *International Journal of Production Research, 11*(1), 47-58.

Balas, E., Simonetti, N., & Vazacopoulos, A. (2008). Job shop scheduling with setup times, deadlines and precedence constraints. *Journal of Scheduling, 11*(4), 253-262.

Behnke, D., & Geiger, M. J. (2012). Test instances for the flexible job shop scheduling problem with work centers.

Bianchi, L., Dorigo, M., Gambardella, L., & Gutjahr, W. (2009). A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing, 8*(2), 239-287. doi: 10.1007/s11047-008-9098-4

Binato, S., Hery, W., Loewenstern, D., & Resende, M. (2002). A GRASP for job shop scheduling *Essays and surveys in metaheuristics* (pp. 59-79): Springer.

Blazewicz, J., Ecker, K. H., Pesch, E., Schmidt, G., & Weglarz, J. (2007). *Handbook on Scheduling: From Theory to Applications*: Springer.

Blum, C., & Li, X. (2008). *Swarm intelligence in optimization*: Springer.

Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv., 35*(3), 268-308. doi: 10.1145/937503.937505

Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research, 41*(3), 157-183.

Brucker, P. (2001). *Scheduling Algorithms*: Springer-Verlag New York, Inc.

Brucker, P., & Neyer, J. (1998). Tabu-search for the multi-mode job-shop problem. *Operations-Research-Spektrum, 20*(1), 21-28.

Brucker, P., & Schlie, R. (1990). Job-shop scheduling with multi-purpose machines. *Computing, 45*(4), 369-375.

Cagnina, L. C., Esquivel, S. C., & Coello, C. A. C. (2008). Solving Engineering Optimization Problems with the Simple Constrained Particle Swarm Optimizer. *Informatica (Slovenia), 32*(3), 319-326.

Chatterjee, A., Mahanti, G. K., & Chatterjee, A. (2012). Design of a fully digital controlled reconfigurable switched beam concentric ring array antenna using firefly and particle swarm optimization algorithm. *Progress In Electromagnetics Research B, 36*, 113-131.

Chen, H., Ihlow, J., & Lehmann, C. (1999). *A genetic algorithm for flexible job-shop scheduling.* Paper presented at the Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on.

Coello Coello, C. A. (2000). Use of a self-adaptive penalty approach for engineering optimization problems. *Computers in Industry, 41*(2), 113-127.

Coley, D. A. (1998). *An Introduction to Genetic Algorithms for Scientists and Engineers*: World Scientific Publishing Co., Inc.

Crepinsek, M., Mernik, M., Shih\, \#45, & Liu, H. (2011). Analysis of exploration and exploitation in evolutionary algorithms by ancestry trees. *Int. J. Innov. Comput. Appl., 3*(1), 11-19. doi: 10.1504/ijica.2011.037947

Dantzig, G., Fulkerson, R., & Johnson, S. (1954). Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America, 2*(4), 393-410.
Dauzère-Pérès, S., & Paulli, J. (1997). An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research, 70*, 281-306.
Dauzère-Pérès, S., Roux, W., & Lasserre, J. (1998). Multi-resource shop scheduling with resource flexibility. *European Journal of Operational Research, 107*(2), 289-305.
Eiben, A., & Smit, S. (2011). Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation, 1*(1).
Evolutionary Algorithms in Engineering and Computer Science: Recent Advances in Genetic Algorithms, Evolution Strategies, Evolutionary Programming, GE. (1999). John Wiley \\& Sons, Inc.
Fang, H.-L., Ross, P., & Corne, D. (1993). *A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems*: University of Edinburgh, Department of Artificial Intelligence.
Fattahi, P., Mehrabad, M. S., & Jolai, F. (2007). Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. *Journal of Intelligent Manufacturing, 18*(3), 331-342.
Feng, G., & Lau, H. C. (2008). Efficient algorithms for machine scheduling problems with earliness and tardiness penalties. *Annals of Operations Research, 159*(1), 83-95.
Fister, I., JrFister, I., Yang, X.-S., & Brest, J. (2013). A comprehensive review of firefly algorithms. *Swarm and Evolutionary Computation*.
Fogel, L. J. (1999). *Intelligence through simulated evolution: forty years of evolutionary programming*: John Wiley \& Sons, Inc.
Fogel, M. J., Owens, L. J., & Walsh, A. J. (1966). *Artificial Intelligence through Simulated Evolution*. Chichester, WS, UK: Wiley.
Gambardella, M. M.-L. M., & Mastrolilli, M. (1996). Effective neighborhood functions for the flexible job shop problem. *Journal of Scheduling, 3*(3).
Garey, M. R., & Johnson, D. S. (1990). *Computers and Intractability; A Guide to the Theory of NP-Completeness*: W. H. Freeman \& Co.
Gazi, V., & Passino, K. M. (2004). Stability analysis of social foraging swarms. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, 34*(1), 539-557.
Girish, B., & Jawahar, N. (2009). *A particle swarm optimization algorithm for flexible job shop scheduling problem.* Paper presented at the Automation Science and Engineering, 2009. CASE 2009. IEEE International Conference on.
Gonçalves, J. F., de Magalhães Mendes, J. J., & Resende, M. c. G. (2005). A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research, 167*(1), 77-95.
Hassanzadeh, T., Vojodi, H., & Mahmoudi, F. (2011). Non-linear grayscale image enhancement based on firefly algorithm *Swarm, Evolutionary, and Memetic Computing* (pp. 174-181): Springer.
Horng, M.-H. (2012). Vector quantization using the firefly algorithm for image compression. *Expert Systems with Applications, 39*(1), 1078-1091. doi: http://dx.doi.org/10.1016/j.eswa.2011.07.108
Huiyuan, R., Lili, J., Xiaoying, X., & Muzhi, L. (2009). *Heuristic optimization for dual-resource constrained job shop scheduling.* Paper presented at the Informatics in Control, Automation and Robotics, 2009. CAR'09. International Asia Conference on.
Hurink, J., Jurisch, B., & Thole, M. (1994). Tabu search for the job-shop scheduling problem with multi-purpose machines. *Operations-Research-Spektrum, 15*(4), 205-215.

Hussain, M. F., & Joshi, S. B. (1998). *A genetic algorithm for job shop scheduling problems with alternate routing.* Paper presented at the Systems, Man, and Cybernetics, 1998. 1998 IEEE International Conference on.

Ignall, E., & Schrage, L. (1965). Application of the branch and bound technique to some flow-shop scheduling problems. *Operations Research, 13*(3), 400-412.

Jakimovski, B., Meyer, B., & Maehle, E. (2010). Firefly Flashing Synchronization as Inspiration for Self-synchronization of Walking Robot Gait Patterns Using a Decentralized Robot Control Architecture. In C. Müller-Schloer, W. Karl & S. Yehia (Eds.), *Architecture of Computing Systems - ARCS 2010* (Vol. 5974, pp. 61-72): Springer Berlin Heidelberg.

Kacem, I. (2003). *Genetic algorithm for the flexible job-shop scheduling problem.* Paper presented at the Systems, Man and Cybernetics, 2003. IEEE International Conference on.

Kacem, I., Hammadi, S., & Borne, P. (2002a). Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, 32*(1), 1-13.

Kacem, I., Hammadi, S., & Borne, P. (2002b). Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic. *Mathematics and computers in simulation, 60*(3), 245-276.

Kennedy, J., & Eberhart, R. (1995, Nov/Dec 1995). *Particle swarm optimization.* Paper presented at the Neural Networks, 1995. Proceedings., IEEE International Conference on.

Kennedy, J., & Eberhart, R. C. (2001). *Swarm intelligence*: Morgan Kaufmann Publishers Inc.

Kwiecień, J., & Filipowicz, B. (2012). Firefly algorithm in optimization of queueing systems. *Bulletin of the Polish Academy of Sciences: Technical Sciences, 60*(2), 363-368.

Li, C.-L., Mosheiov, G., & Yovel, U. (2008). An efficient algorithm for minimizing earliness, tardiness, and due-date costs for equal-sized jobs. *Computers & Operations Research, 35*(11), 3612-3619.

Lin, B. M., & Cheng, T. (2001). Batch scheduling in the no-wait two-machine flowshop to minimize the makespan. *Computers & Operations Research, 28*(7), 613-624.

Liu, H., Abraham, A., & Wang, Z. (2009). A multi-swarm approach to multi-objective flexible job-shop scheduling problems. *Fundamenta Informaticae, 95*(4), 465-489.

Łukasik, S., & Żak, S. (2009). Firefly algorithm for continuous constrained optimization tasks *Computational Collective Intelligence. Semantic Web, Social Networks and Multiagent Systems* (pp. 97-106): Springer.

Mattfeld, D. C., & Bierwirth, C. (2004). An efficient genetic algorithm for job shop scheduling with tardiness objectives. *European Journal of Operational Research, 155*(3), 616-630.

McMahon, G., & Burton, P. (1967). Flow-shop scheduling with the branch-and-bound method. *Operations Research, 15*(3), 473-481.

Michalewicz, Z. (1996). *Genetic algorithms + data structures = evolution programs (3rd ed.)*: Springer-Verlag.

Muth, J. F., & Thompson, G. L. (1963). *Industrial Scheduling*: Prentice-Hall.

Nakano, R., & Yamada, T. (1991). *Conventional Genetic Algorithm for Job Shop Problems.* Paper presented at the ICGA.

Newell, A., & Simon, H. A. (1976). Computer science as empirical inquiry: Symbols and search. *Communications of the ACM, 19*(3), 113-126.

Nuijten, W. P., & Aarts, E. H. (1996). A computational study of constraint satisfaction for multiple capacitated job shop scheduling. *European Journal of Operational Research, 90*(2), 269-284.

Pezzella, F., Morganti, G., & Ciaschetti, G. (2008). A genetic algorithm for the flexible job-shop scheduling problem. *Computers & Operations Research, 35*(10), 3202-3212.

Sayadi, M., Ramezanian, R., & Ghaffari-Nasab, N. (2010). A discrete firefly meta-heuristic with local search for makespan minimization in permutation flow shop scheduling problems. *International Journal of Industrial Engineering Computations, 1*(1), 1-10.

Sidney, J. B. (1977). Optimal single-machine scheduling with earliness and tardiness penalties. *Operations Research, 25*(1), 62-69.

Suwa, H., & Sandoh, H. (2007). Capability of cumulative delay based reactive scheduling for job shops with machine breakdowns. *Computers & Industrial Engineering, 53*(1), 63-78.

Talbi, E. G. (2009). *Metaheuristics: From Design to Implementation*: Wiley.

Tashkova, K., Šilc, J., Atanasova, N., & Džeroski, S. (2012). Parameter estimation in a nonlinear dynamic model of an aquatic ecosystem with meta-heuristic optimization. *Ecological Modelling, 226*(0), 36-61. doi: http://dx.doi.org/10.1016/j.ecolmodel.2011.11.029

Wilson, J. M. (2003). Gantt charts: A centenary appreciation. *European Journal of Operational Research, 149*(2), 430-437.

Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *Trans. Evol. Comp, 1*(1), 67-82. doi: 10.1109/4235.585893

Xia, W., & Wu, Z. (2005). An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. *Computers & Industrial Engineering, 48*(2), 409-425.

Xin, #45, & Yang, S. (2010). Firefly algorithm, stochastic test functions and design optimisation. *Int. J. Bio-Inspired Comput., 2*(2), 78-84. doi: 10.1504/ijbic.2010.032124

Xin, #45, & Yang, S. (2011). Review of meta-heuristics and generalised evolutionary walk algorithm. *Int. J. Bio-Inspired Comput., 3*(2), 77-84. doi: 10.1504/ijbic.2011.039907

Yang, X.-S. (2009). Firefly Algorithms for Multimodal Optimization. In O. Watanabe & T. Zeugmann (Eds.), *Stochastic Algorithms: Foundations and Applications* (Vol. 5792, pp. 169-178): Springer Berlin Heidelberg.

Yang, X.-S. (2010). Firefly algorithm, stochastic test functions and design optimisation. *International Journal of Bio-Inspired Computation, 2*(2), 78-84.

Yang, X.-S. (2011). Metaheuristic Optimization: Algorithm Analysis and Open Problems. In P. Pardalos & S. Rebennack (Eds.), *Experimental Algorithms* (Vol. 6630, pp. 21-32): Springer Berlin Heidelberg.

Yang, X.-S. (2014a). Cuckoo Search and Firefly Algorithm: Overview and Analysis. In X.-S. Yang (Ed.), *Cuckoo Search and Firefly Algorithm* (Vol. 516, pp. 1-26): Springer International Publishing.

Yang, X.-S. (2014b). Cuckoo Search and Firefly Algorithm: Overview and Analysis. In X.-S. Yang (Ed.), *Cuckoo Search and Firefly Algorithm* (Vol. 516, pp. 13): Springer International Publishing.

Yang, X.-S., Deb, S., & Fong, S. (2011). Accelerated Particle Swarm Optimization and Support Vector Machine for Business Optimization and Applications. In S. Fong (Ed.), *Networked Digital Technologies* (Vol. 136, pp. 53-66): Springer Berlin Heidelberg.

Yang, X. S. (2008). *Nature-Inspired Metaheuristic Algorithms*: Luniver Press.

Yau, H., Pan, Y., & Shi, L. (2008). New solution approaches to the general single-machine earliness-tardiness problem. *Automation Science and Engineering, IEEE Transactions on, 5*(2), 349-360.

Zamuda, A., & Brest, J. (2012). Population reduction differential evolution with multiple mutation strategies in real world industry challenges *Swarm and Evolutionary Computation* (pp. 154-161): Springer.

Zhang, C. Y., Li, P., Rao, Y., & Guan, Z. (2008). A very fast TS/SA algorithm for the job shop scheduling problem. *Computers & Operations Research, 35*(1), 282-294. doi: http://dx.doi.org/10.1016/j.cor.2006.02.024

Παράρτημα 1: Κώδικας μεθόδου τυχαίας μετακίνησης Πυγολαμπίδας

```
 public void moveFFrandomly (ref firefly1 ourFF,job1[] myJobs)
        {
            int numberOfOperToMove = Convert.ToInt32(Global.totalNumberOfOper *
0.05); //change 5% oper of this FF
            if (numberOfOperToMove==0)
            {
                numberOfOperToMove = 1;
            }
            for (int i1 = 0; i1 < numberOfOperToMove; i1++)
            {
                int iSerial = Global.rand001.Next(0,
Global.totalNumberOfOper);// this and the vars below are used to spot the
random result in the lists.
                int iMach = 0;
                int iCounts =
ourFF.lMachines[iMach].lAssignedJobOperationsToMachine.Count;
                while (iSerial >= iCounts)
                {
                    iMach++;
                    if (iMach > ourFF.lMachines.Count - 1)
                    {
                        Console.WriteLine("Error, iMach out of bounds");
                    }
                    iCounts +=
ourFF.lMachines[iMach].lAssignedJobOperationsToMachine.Count;
                }
                int iOrderInMachFromEnd = iCounts - (iSerial + 1);
                int iOrderInMachNormal =
(ourFF.lMachines[iMach].lAssignedJobOperationsToMachine.Count - 1) -
iOrderInMachFromEnd;
                JobOperationFirefly changedJOF =
ourFF.lMachines[iMach].lAssignedJobOperationsToMachine[iOrderInMachNormal];
                //18jan:
                double changeMachPercent = 0.9; //90% chance to change mach,
10% to stay at same mach
                double randomChangeOrNot = Global.rand001.NextDouble();
                bool flagChangeMach = true;
                if (randomChangeOrNot>changeMachPercent)
                {
                    flagChangeMach = false;
                }
                else if (randomChangeOrNot<=changeMachPercent)
                {
                    flagChangeMach = true;
                }

                if (flagChangeMach == true)
                {
                    //move to another mach
                    List<int> possMachIDs = new List<int>();
```

```
                    List<int> poss_iWC = new List<int>();
                    for (int i = 0; i <
myJobs[changedJOF.iJobID].possibleMachForOper.GetLength(0); i++)
                    {
                        if (myJobs[changedJOF.iJobID].possibleMachForOper[i,
changedJOF.iOperID] >= 0
                            && myJobs[changedJOF.iJobID].possibleMachForOper[i,
changedJOF.iOperID] != changedJOF.iMachineID)
                        {

possMachIDs.Add(myJobs[changedJOF.iJobID].possibleMachForOper[i,
changedJOF.iOperID]);
                            poss_iWC.Add(i);
                        }


                    }
                    if (possMachIDs.Count == 0)
                    {
                        Console.WriteLine("no other machs to move this to");
                    }
                    else
                    {




                        int i_whichChoiceID;
                        int newMachID = GetMachID_WithChances(changedJOF,
possMachIDs, poss_iWC, myJobs,out i_whichChoiceID);
                        int newMachPos = determinePosInNewMach(ourFF,
changedJOF, newMachID, myJobs);



ourFF.lMachines[changedJOF.iMachineID].lAssignedJobOperationsToMachine.RemoveAl
l(item => item.iJobID == changedJOF.iJobID && item.iOperID ==
changedJOF.iOperID);
                        changedJOF.iMachineID = newMachID;
                        changedJOF.iWhichChoiceID = i_whichChoiceID;

ourFF.lMachines[changedJOF.iMachineID].lAssignedJobOperationsToMachine.Insert(n
ewMachPos, changedJOF);

ourFF.jobOriented[changedJOF.iJobID].lAssingedInfoForOpersOfthisJob[changedJOF.
iOperID].iMachineID = newMachID;

ourFF.jobOriented[changedJOF.iJobID].lAssingedInfoForOpersOfthisJob[changedJOF.
iOperID].iWhichChoiceID = changedJOF.iWhichChoiceID;
                    }
                }
                else if (flagChangeMach==false) //move to other position in
this mach
                {

                                int step_size =
Convert.ToInt32(ourFF.lMachines[changedJOF.iMachineID].lAssignedJobOperationsTo
Machine.Count
                                    * 0.1); //move by 10% of chain of opers in
this Mach
                                if (step_size==0)
```

```
                                {
                                    step_size = 1;
                                }
                                bool limitationsOK = true;

                                if (step_size > iOrderInMachNormal) //to
prevent it from pushing to negatives
                                {
                                    step_size = iOrderInMachNormal;
                                }


                                if (step_size >= 1)
                                {
                                    for (int i5 = 1; i5 <= step_size; i5++)
                                    {
                                        if
(ourFF.lMachines[changedJOF.iMachineID].lAssignedJobOperationsToMachine[iOrderI
nMachNormal - i5].iJobID
                                            == changedJOF.iJobID) // if its a
prev oper of the same job
                                        {
                                            limitationsOK = false;
                                            step_size = i5 - 1;
                                            break;
                                        }


                                    }
                                }
                                if (limitationsOK==true ||
limitationsOK==false)

                                {

ourFF.lMachines[changedJOF.iMachineID].lAssignedJobOperationsToMachine.RemoveAt
(iOrderInMachNormal);

ourFF.lMachines[changedJOF.iMachineID].lAssignedJobOperationsToMachine.Insert(i
OrderInMachNormal - step_size, changedJOF);
                                //Console.WriteLine("just for testing");
                                }
                            }
                        }


                        updateFFTimes(ourFF, myJobs);
                    }
```