ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ ΠΑΡΑΓΩΓΗΣ ΚΑΙ ΔΙΟΙΚΗΣΗΣ

ΕΞΕΛΙΚΤΙΚΟΣ ΑΛΓΟΡΙΘΜΟΣ ΒΕΛΤΙΣΤΟΠΟΙΗΣΗΣ ΤΟΥ ΠΡΟΒΛΗΜΑΤΟΣ ΧΡΟΝΟΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΠΑΡΑΓΩΓΗΣ

ΓΙΩΡΓΟΣ ΖΩΓΡΑΦΟΣ

Επιβλέπων: Ιωάννης Μαρινάκης

Χανιά, 2013

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή, κ. Ιωάννη Μαρινάκη για την πολύτιμη συμβολή του στην εκπόνηση αυτής της εργασίας, την οικογένεια μου για την υλική και πνευματική τους υποστήριξη και τους φίλους μου για πολλούς και διάφορους λόγους.

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΕΡΙΛΗΨΗ

Τα προβλήματα χρονοπρογραμματισμού παραγωγής έχουν δημιουργήσει έντονο ενδιαφέρον στην ερευνητική κοινότητα, επειδή συνδυάζουν πολλά διαφορετικά επιστημονικά πεδία. Ως χρονοπρογραμματισμός ορίζεται η κατανομή πόρων σε ένα συγκεκριμένο χρονικό διάστημα προκειμένου να περατωθεί ένας προαποφασισμένος αριθμός εργασιών.

Αυτή η εργασία επικεντρώνεται στο πρόβλημα χρονοπρογραμματισμού μηχανών κατά παραγγελία (*job shop scheduling problem*), που είναι ένα από τα πιο δύσκολα συνδυαστικά προβλήματα που έχουν διατυπωθεί με αποτέλεσμα την ύπαρξη πολλών διαφορετικών προσεγγίσεων. Ένα τέτοιο πρόβλημα ορίζεται, στη γενική του μορφή από τέσσερα δεδομένα: τον αριθμό των μηχανών, τον αριθμό των εργασιών, τις ακολουθίες των διεργασιών για την κάθε εργασία και τους χρόνους εκτέλεσης της κάθε εργασίας. Ένα χρονοπρόγραμμα είναι μία ανάθεση των εργασιών στις αντίστοιχες μηχανές, ώστε να ικανοποιούνται όλοι οι περιορισμοί. Ένα χρονοπρόγραμμα ονομάζεται εφικτό αν δεν περιέχει επικαλύψεις εργασιών στην ίδια μηχανή και βέλτιστο αν ελαχιστοποιεί την αντικειμενική συνάρτηση.

Οι Εξελικτικοί Αλγόριθμοι χρησιμοποιούν υπολογιστικά μοντέλα εξελικτικών διαδικασιών σαν βασικά στοιχεία σχεδιασμού και υλοποίησης υπολογιστικών συστημάτων επίλυσης προβλημάτων. Είναι αλγόριθμοι ανίχνευσης-αναζήτησης, βασισμένοι στη μηχανική της φυσικής επιλογής και της φυσικής γενετικής. Οι Γενετικοί Αλγόριθμοι πραγματοποιούν αναζήτηση βέλτιστης λύσης συνδυάζοντας υπάρχουσες λύσεις και ανήκουν στην κατηγορία των Εξελικτικών Αλγορίθμων. Όπως δηλώνει και το όνομα τους μιμούνται μηχανισμούς εξέλιξης της φύσης, όπως η φυσική επιλογή, συνδυασμός γονιδίων και μετάλλαξη για να ελέγξουν την διαδικασία αναζήτησης.

Σκοπός αυτής της εργασίας είναι η υλοποίηση ενός προγράμματος, σε προγραμματιστικό περιβάλλον Matlab, το οποίο επιλύνει το πρόβλημα χρονοπρογραμματισμού μηχανών χρησιμοποιώντας τη μέθοδο των γενετικών αλγορίθμων με αντικειμενικό κριτήριο προς ελαχιστοποίηση τον συνολικό χρόνο εκτέλεσης των εργασιών. Σαν δεδομένα χρησιμοποιήθηκαν παραδείγματα από την επιστημονική βιβλιογραφία, ώστε τα αποτελέσματα που προκύπτουν να είναι συγκρίσιμα.

ΚΕΦΑΛΑΙΟ 1

ΤΟ ΠΡΟΒΛΗΜΑ ΧΡΟΝΟΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΜΗΧΑΝΩΝ

1.1 ΕΙΣΑΓΩΓΗ

Το πρόβλημα χρονοπρογραμματισμού μηχανών περιγράφει μία κατάσταση στην οποία ένας αριθμός εργασιών, προκειμένου να περατωθεί, πρέπει να περάσει από ένα σύνολο σταδίων επεξεργασίας. Ένα παράδειγμα είναι η παραγωγή διαφόρων τύπων δοκιμίων τα οποία πρέπει να περάσουν από τόρνο, φρέζα και δράπανο. Σε περίπτωση που όλες οι εργασίες περνούν από κάθε στάδιο κατεργασίας με την ίδια σειρά το σύστημα χαρακτηρίζεται *συνεχούς ροής (flow shop)* ενώ αν η κάθε εργασία ακολουθεί τη δική της σειρά από διεργασίες το σύστημα χαρακτηρίζεται *κατά παραγγελία (job shop).* Σε αυτή την εργασία μελετάται το σύστημα παραγωγής κατά παραγγελία.

Τα προβλήματα χρονοπρογραμματισμού παραγωγής έχουν δημιουργήσει έντονο ενδιαφέρον στην ερευνητική κοινότητα επειδή συνδυάζουν επιστημονικά πεδία τόσο διαφορετικά όπως είναι αυτά του προγραμματισμού παραγωγής (*production planning*), της σχεδίασης ηλεκτρονικών υπολογιστών και αντίστοιχου λογισμικού (*computer and software design*) και της δημιουργίας χρονοδιαγραμμάτων (*timetabling*) και επιπρόσθετα έχουν τη δυνατότητα να μειώσουν δραματικά το κόστος και να αυξήσουν την παραγωγικότητα μιας μονάδας και, ως συνέπεια, τα κέρδη. Ένα σημαντικό πεδίο της έρευνας, τόσο από την πλευρά της επιχειρησιακής έρευνας όσο και από την πλευρά της τεχνητής νοημοσύνης, έχει αφιερωθεί στην θεωρία και τις εφαρμογές του χρονοπρογραμματισμού παραγωγής. Το πρόβλημα του χρονοπρογραμματισμού μπορεί να οριστεί ως «η κατανομή των πόρων σε ένα καθορισμένο χρονικό διάστημα ώστε να εκπληρωθεί μία προαποφασισμένη συλλογή εργασιών παραγωγής (Baker, 1974).».

Τα τελευταία χρόνια μεγάλο ποσοστό έρευνας έχει επικεντρωθεί στο πρόβλημα χρονοπρογραμματισμού παραγωγής κατά παραγγελία (*job-shop scheduling problem, JSSP*) που είναι ένα από τα πιο δύσκολα συνδυαστικά προβλήματα που έχουν διατυπωθεί με αποτέλεσμα την ύπαρξη πολλών διαφορετικών προσεγγίσεων. Ο σκοπός αυτής της εργασίας είναι να αναλύσει το πρόβλημα αυτό και να παρουσιάσει τις διάφορες τεχνικές επίλυσής του. Τα συστήματα παραγωγής κατά παραγγελία παράγουν μεγάλη ποικιλία προϊόντων σε μικρές ποσότητες και με προδιαγραφές που ορίζονται από τον πελάτη, ο οποίος αναθέτει στο σύστημα την παραγωγή ενός αριθμού ίδιων προϊόντων.

Πολύ σημαντικό είναι πως το σύστημα κατά παραγγελία είναι ένα συγκεκριμένο αλλά αρκετά συνηθισμένο μοντέλο χρονοπρογραμματισμού μηχανών. Υπάρχουν και

άλλα μοντέλα όπως τα συνεχούς ροής και ανοιχτής παραγωγής (*open shop*). Στην επιστημονική λογοτεχνία τα μοντέλα χρονοπρογραμματισμού μηχανών περιγράφονται και σχηματίζονται με ένα ασαφή τρόπο. Στην πραγματικότητα, τα προβλήματα αυτά είναι, τις περισσότερες φορές, πιο περίπλοκα και περιέχουν πολλούς περισσότερους παράγοντες και περιορισμούς. Εν τούτοις, για να δημιουργηθεί μία εικόνα για τις μεθευρετικές προσεγγίσεις του χρονοπρογραμματισμού μηχανών, η δομή ενός απλού προβλήματος είναι καταλληλότερη για να επισημανθούν οι σημαντικότερες έννοιες της εφαρμογής του συγκεκριμένου επιπέδου των μεθόδων αυτών. Τα ασαφή μοντέλα απασχολούν το μεγαλύτερο μέρος της ακαδημαϊκής βιβλιογραφίας και διευκολύνουν τη σύγκριση διάφορων μεθόδων επίλυσης.

Αναλυτικότερα, πρόβλημα χρονοπρογραμματισμού μηχανών κατά παραγγελία αποτελείται από μία ομάδα J πεπερασμένου αριθμού n εργασιών $J=\{J_1, J_2, \ldots, J_n\}$, οι οποίες εκτελούνται σε μία ομάδα M πεπερασμένου αριθμού m μηχανών $M=\{M_1, M_2, \ldots, M_n\}$. Κάθε εργασία είναι μία σειρά από διεργασίες o_{ik} (όπου ο δείκτης i υποδεικνύει τον αριθμό της εργασίας και ο δείκτης k τον αριθμό της μηχανής) και πρέπει να εκτελεστεί σε κάθε μία από τις μηχανές ακολουθώντας μία προκαθορισμένη σειρά, διαφορετική για κάθε εργασία. Κάθε εργασία μπορεί να εκτελείται σε μία μηχανή ανά πάσα στιγμή και αντίστοιχα μία μηχανή δεν μπορεί να εκτελεί περισσότερες από μία εργασίες ταυτόχρονα. Κάθε διεργασία o_{ik} έχει έναν ακέραιο χρόνο εκτέλεσης p_{ik}.

Σε ένα πρότυπο πρόβλημα χρονοπρογραμματισμού μηχανών, χάριν ευκολίας θεωρείται ότι υπάρχουν οι εξής υποθέσεις και περιορισμοί:

- Κάθε εργασία επισκέπτεται κάθε μηχανή ακριβώς μία φορά.
- Οι διεργασίες κάθε εργασίας πρέπει να εκτελεστούν αυστηρά με την προκαθορισμένη σειρά· δεν επιτρέπεται παράλληλη εκτέλεση ή επικάλυψη εργασιών.
- Οι χρόνοι προετοιμασίας των μηχανών και μεταφοράς από μία μηχανή σε μία άλλη θεωρούνται αμελητέοι.
- Κάθε διεργασία ανατίθεται σε ακριβώς μία μηχανή για εκτέλεση.
- Οι μηχανές είναι συνεχώς διαθέσιμες, δηλαδή δεν συμβαίνουν βλάβες ή άλλες διακοπές της παραγωγής.

Ο στόχος του προγραμματισμού είναι να καθοριστεί η ακολουθία των διεργασιών σε κάθε μηχανή M_k τέτοια ώστε οι περιορισμοί ακολουθίας κάθε εργασίας και χωρητικότητας κάθε μηχανής να ικανοποιούνται. Κάθε αντιμετάθεση δύο διεργασιών απεικονίζει μία λύση για το πρόβλημα, αυτό όμως δεν είναι πρακτικό μετά την επεξεργασία. Εκτός από τη θέση της διεργασίας στη σειρά ακολουθίας, σημαντικότερη είναι η χρονική στιγμή εκκίνησής της. Για αυτό το λόγο μία πιθανή λύση προτιμάται να απεικονίζεται ως μία ομάδα S από χρόνου εκκίνησης s_{ik}, $S=\{s_{ik} | 1 \leq I \leq n, 1 \leq k \leq m\}$.

Οι χρονικές πληροφορίες είναι σημαντικές όταν υπολογίζεται η ποιότητα μίας συγκεκριμένης ακολουθίας. Είναι προφανές πως ένα αυθαίρετο πρόγραμμα το οποίο να είναι εφικτό και ικανοποιεί τους περιορισμούς συνήθως δεν έχει ικανοποιητικό αποτέλεσμα, για αυτό το λόγο πρέπει να συνυπολογίζεται το κόστος. Ο στόχος πρέπει να είναι η επίτευξη ενός εφικτού προγράμματος με το μικρότερο δυνατό κόστος. Έτσι αποκτούμε ένα πρόβλημα βελτιστοποίησης, το οποίο συνήθως είναι πολύ δύσκολο να επιλυθεί. Η δυσκολία του προβλήματος εξαρτάται εν μέρει από την αντικειμενική συνάρτηση κόστους ή γενικότερα το αντικειμενικό κριτήριο της βελτιστοποίησης.

1.2 ΑΝΤΙΚΕΙΜΕΝΙΚΑ ΚΡΙΤΗΡΙΑ ΒΕΛΤΙΣΤΟΠΟΙΗΣΗΣ

Ο χρονοπρογραμματισμός μηχανών είναι στενά συνδεδεμένος με την έννοια του κόστους και της αποδοτικότητας. Παρόλα αυτά είναι πολλές φορές δύσκολο να αξιολογηθεί ένα πρόγραμμα με βάση αυστηρά οικονομικά μόνο κριτήρια Για το λόγο αυτό εισήχθησαν κριτήρια με βάση τους χρόνους εκτέλεσης των εργασιών και των επιμέρους διεργασιών, ενώ μερικά λαμβάνουν υπ' όψιν επιπλέον προθεσμίες και ειδικά βάρη. Προκειμένου το πρόβλημα να γίνει κατανοητό θα ακολουθήσει η περιγραφή του με μαθηματικούς όρους.

Με βάση ένα δεδομένο πρόγραμμα *S*, προκύπτουν αμέσως οι χρόνοι ολοκλήρωσης των εργασιών και των διεργασιών. Έστω c_{ik} ο χρόνος ολοκλήρωσης της διεργασίας o_{ik} (όπου ο δείκτης i υποδεικνύει τον αριθμό της εργασίας και ο δείκτης k τον αριθμό της μηχανής), στο πρόγραμμα *S* και C_i ο χρόνος ολοκλήρωσης της εργασίας J_i. Προκύπτει ότι $C_i = max_{1 \leq k \leq m} c_{ik}$. Σε περίπτωση που το πρόβλημα περιλαμβάνει περιορισμούς προθεσμιών d_i, αυτές θα πρέπει να είναι δεδομένες. Σε μερικές περιπτώσεις ίσως είναι σκόπιμο να ιεραρχηθούν οι εργασίες ανάλογα με το πόσο σημαντικές ή επείγουσες είναι. Σε αυτή αυτήν την περίπτωση σε κάθε εργασία ανατίθεται ένα βάρος w_i το οποίο λαμβάνει μη μηδενικές τιμές. Σε περίπτωση που δεν χρησιμοποιούνται βάρη θεωρείται ότι κάθε εργασία έχει βάρος ίσο με 1.

Τα αντικειμενικά κριτήρια με βάση τα οποία γίνεται η βελτιστοποίηση ενός προβλήματος είναι:

- Ελαχιστοποίηση του συνολικού χρόνου διάρκειας των εργασιών, δηλαδή του χρόνου που απαιτείται για την ολοκλήρωση όλων των εργασιών (*makespan or maximum completion time*)

Η ελαχιστοποίηση του makespan είναι το πιο συνηθισμένο αντικειμενικό κριτήριο βελτιστοποίησης στα προβλήματα χρονοπρογραμματισμού μηχανών. Το makespan ουσιαστικά είναι ο συνολικός χρόνος που απαιτείται για να ολοκληρωθεί μία συγκεκριμένη ομάδα από εργασίες. Κατά την ελαχιστοποίηση, ο προγραμματιστής λαμβάνει με ακρίβεια πληροφορίες για τα χρονικά όρια της παραγωγής και για το αν θα ολοκληρωθεί σύμφωνα μέσα σε αυτά. Το makespan ορίζεται ως:

$$C_{max} = \max\{1 \leq i \leq n | C_i\} \rightarrow Min \quad (1.1)$$

- Ελαχιστοποίηση του νεκρού χρόνου (*machine utilization)*

Ως νεκρός χρόνος ορίζεται ο λόγος του χρόνου κατά τον οποίο εργάζεται η μηχανή προς το σύνολο του χρόνου που είναι διαθέσιμη. Για μία μονάδα παραγωγής είναι επιθυμητό οι νεκροί χρόνοι των μηχανών να παραμένουν ελάχιστοι. Η μέση χρήση των μηχανών *MU* υπολογίζεται από τον τύπο:

$$MU = \frac{\sum_{i=1}^{n} \sum_{k=1}^{m} p_{ik}}{m \cdot C_{max}} \quad (1.2)$$

Όπως γίνεται κατανοητό σκοπός της βελτιστοποίησης είναι η μεγιστοποίηση της μέσης χρήσης μηχανών. Είναι ενδιαφέρον το γεγονός ότι η ελαχιστοποίηση του makespan και η μεγιστοποίηση της μέσης χρήσης μηχανών οδηγούν στο ίδιο βέλτιστο πρόγραμμα.

- Κριτήρια συνεχούς ροής (*flow time related criteria)*

Η χρόνος ροής μια εργασίας (*flow time*) ορίζεται ως το χρονικό διάστημα μεταξύ της απελευθέρωσης της r_i (δηλαδή της χρονικής στιγμής κατά την οποία περατώνεται η τελευταία προαπαιτούμενη της) και της ολοκλήρωσης της C_i:

$$F_i = C_i - r_i \quad (1.3)$$

Αυτός ο δείκτης απόδοσης είναι σημαντικός όταν λαμβάνονται υπ' όψιν τα αποθέματα κατά τη διάρκεια της εκτέλεσης του προγράμματος. Όσο μεγαλύτερος είναι ο χρόνος ροής μιας εργασίας, τόσο μεγαλύτερο θα είναι το κόστος διατήρησης του αποθέματος. Μία κοινή πρακτική για την ελαχιστοποίηση αυτού του κόστους είναι η ελαχιστοποίηση του συνολικού χρόνου ροής του προγράμματος, συμπεριλαμβάνοντας και βάρη αν κριθεί απαραίτητο:

$$\sum_{i=1}^{n} w_i F_i \rightarrow Min \quad (1.4)$$

Σε αντιστοιχία με την περίπτωση ελαχιστοποίησης του makespan, ελαχιστοποιώντας τον συνολικό σταθμισμένο χρόνο ροής, ελαχιστοποιείται ταυτόχρονα ο συνολικά σταθμευμένος χρόνος ολοκλήρωσης των εργασιών ($\sum^{n}_{i=1} w_i F_i$ -> *Min*).

- Κριτήρια προθεσμιών (*Due date related criteria*)

Όταν περιλαμβάνονται προθεσμίες d_i, καθυστερήσεις ή οποιουδήποτε είδους αποκλίσεις επηρεάζουν τη λύση του προβλήματος. Το πιο απλό κριτήριο είναι η ολική καθυστέρηση (*maximum lateness*) L_{max}:

$$L_{max} = \max\{1 \leq i \leq n | C_i - d_i\} \rightarrow Min \tag{1.5}$$

Ένα πρόβλημα ελαχιστοποίησης της ολικής καθυστέρησης είναι μία γενίκευση του προβλήματος ελαχιστοποίησης του συνολικού χρόνου ολοκλήρωσης των εργασιών. Με βάση αυτή τη σχέση ένα πρόβλημα ελαχιστοποίησης του L_{max} μπορεί να επιλυθεί σαν πρόβλημα ελαχιστοποίησης του C_{max} κάνοντας μια μετατροπή.

Ένα ακόμα πρακτικό αντικειμενικό κριτήριο είναι ελαχιστοποίηση της ολικής σταθμισμένης καθυστέρησης (*total weighted tardiness*). Η ολική σταθμισμένη καθυστέρηση T_i μιας εργασίας ορίζεται ως:

$$T_i = \max(0, C_i - d_i) \tag{1.6}$$

Όπως γίνεται κατανοητό μόνο θετικές καθυστερήσεις σε σχέση με την προθεσμία έχουν νόημα και εξετάζονται. Το κριτήριο διαμορφώνεται ως εξής:

$$\sum_{i=1}^{n} w_i T_i \rightarrow Min \tag{1.7}$$

Χρησιμοποιώντας την καθυστέρηση ως κριτήριο αγνοούνται οι εργασίες που τελείωσαν νωρίτερα από την προθεσμία τους και δεν έχουν καμία επίπτωση στην τιμή της αντικειμενικής συνάρτησης. Σε μερικά συστήματα, οπως για παράδειγμα στα Just-In-Time (JIT), είναι σημαντικό οι εργασίες να τελειώνουν όσο δυνατόν πιο κοντά στις δεδομένες προθεσμίες. Σε αυτή την περίπτωση η αντικειμενική συνάρτηση προσαρμόζεται ώστε να λαμβάνει την απόκλιση σε απόλυτη τιμή:

$$\sum_{i=1}^{n} w_i |C_i - d_i| \rightarrow Min \tag{1.8}$$

Άλλη μία επιλογή είναι η δημιουργία δύο διαφορετικών συντελεστών για κάθε εργασία, τον συντελεστή νωρίτερης ολοκλήρωσης α_i και τον συντελεστή βραδύτερης ολοκλήρωσης β_i, δημιουργώντας το κριτήριο *earliness-tardiness* (ET). Η αντικειμενική του συνάρτηση είναι της μορφής:

$$\sum_{i=1}^{n} \alpha_i \cdot \max(0, d_i - C_i) + \sum_{i=1}^{n} \beta_i \cdot \max(0, C_i - d_i) \rightarrow Min \tag{1.9}$$

Από τα παραπάνω επιλέχθηκε το κριτήριο ελαχιστοποίησης του συνολικού χρόνου ολοκλήρωσης των εργασιών (*makespan*). Το πλεονέκτημα που προσφέρει είναι ότι σε αντίθεση με τα περισσότερα κριτήρια είναι εύκολη η οπτικοποίηση των λύσεων που προσφέρει με τη χρήση διαγραμμάτων κάνοντας ευκολότερα κατανοητές τις αλλαγές σε ένα πρόγραμμα. Επιπλέον οι περισσότερες εργασίες πάνω στον τομέα του χρονοπρογραμματισμού χρησιμοποιούν το makespan ως κριτήριο βελτιστοποίησης, οπότε υπάρχει μία πληθώρα προσεγγίσεων αλλά και αποτελεσμάτων και σύγκριση.

1.3 ΜΕΘΟΔΟΙ ΑΝΑΠΑΡΑΣΤΑΣΗΣ

Οι μέθοδοι που ακολουθούνται για την αναπαράσταση λύσεων του προβλήματος χρονοπρογραμματισμού κατά παραγγελία είναι οι εξής:

- Αναπαράσταση με βάση τις διεργασίες (*Operation-based representation*)
- Αναπαράσταση με βάση τις εργασίες (*Job-based representation*)
- Λίστα προτίμησης (*Preference list-based representation*)
- Αναπαράσταση με βάση τις σχέσεις εργασιών (*Job pair relation-based representation*)
- Αναπαράσταση με βάση τους κανόνες προτεραιότητας (*Priority rule-based representation*)
- Αναπαράσταση με βάση τυχαία κλειδιά (*Random keys-based representation*)
- Αναπαράσταση διαζευκτικών γράφων (*Disjunctive graph-based representation*)
- Διάγραμμα Gantt (*Gantt-based representation*)
- Αναπαράσταση με βάση τον χρόνο ολοκλήρωσης (*Completion time-based representation*)
- Αναπαράσταση με βάση τις μηχανές (*Machine-based representation*)

Στη συνέχεια της ενότητας θα αναλυθούν οι πιο διαδεδομένες μέθοδοι: τα διαγράμματα Gantt και οι διαζευκτικοί γράφοι (*disjunctive graph-based representation*).

1.3.1 Διάγραμμα Gantt

Το διάγραμμα Gantt είναι ένα οριζόντιο ιστόγραμμα που περιγράφηκε από τους Gantt, Clark (1922) και Porter (1968). Παρέχει μια γραφική απεικόνιση ενός έργου που βοηθά το σχεδιασμό, τον συντονισμό και την εξειδίκευση των εργασιών σε ένα έργο.

Στον χρονοπρογραμματισμό χρησιμοποιείται συνήθως η εξής διαμόρφωση: κατασκευάζεται με έναν οριζόντιο άξονα που αντιπροσωπεύει την εξέλιξη του

προγράμματος συναρτήσει του χρόνου(π.χ., ημέρες, εβδομάδες, ή μήνες) και ένα κάθετο άξονα που αντιπροσωπεύει τις εργασίες που αποτελούν το έργο. Κάθε μπάρα μέσα στο διάγραμμα αναπαριστά μία διεργασία, και συνήθως περιέχει τον αριθμό της μηχανής ή της διεργασίας. Η προβολή του δεξιού άκρου της μπάρας στον οριζόντιο άξονα είναι το χρονικό σημείο από το οποίο αρχίζει η εκτέλεση της εργασίας και αντίστοιχα η προβολή του αριστερού άκρου της στον ίδιο άξονα το σημείο που περατώνεται η εκτέλεση της. Διαγράμματα Gantt απεικονίζονται στα σχήματα 1.3 και 1.4.

1.3.2 Αναπαράσταση διαζευκτικού γραφήματος (Disjunctive graph representation)

Μολονότι το διάγραμμα Gantt παραδοσιακά αποτέλεσε την πιο δημοφιλή μέθοδο αναπαράστασης της λύσης ενός προβλήματος χρονοπρογραμματισμού εργασιών, οι Blazewicz et al. (1996) έδειξαν ότι το μοντέλο διαζευκτικών γράφων (disjunctive graph model), των Roy και Sussmann (1964) είναι πλέον αυτό που επικρατεί.

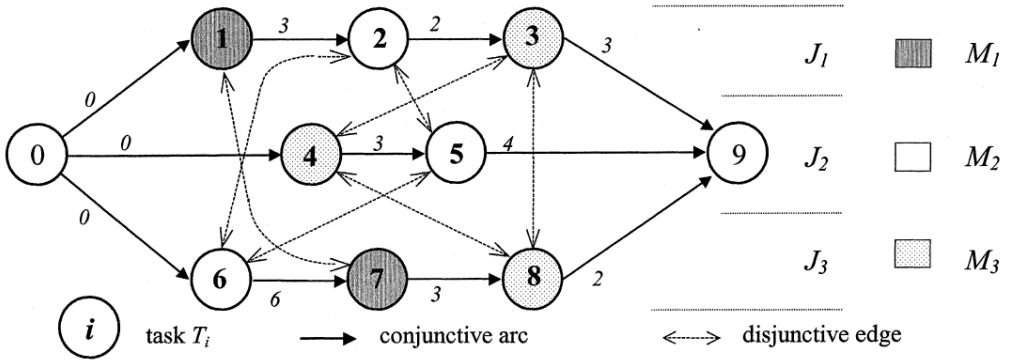
Το διαζευκτικό γράφημα είναι ένα από τα πιο δημοφιλή μοντέλα που χρησιμοποιούνται για να περιγράφει περιπτώσεις του προβλήματος προγραμματισμού σε περιβάλλον job shop, που στην πραγματικότητα είναι ένα από τα πιο διευρυμένα σε επίπεδο έρευνας, πρόβληματα προγραμματισμού. Το διαζευκτικό γράφημα είναι ένα κατευθυνόμενο γράφημα $G = (V, C \cup D)$. Το V δηλώνει ένα σύνολο κορυφών που αντιστοιχούν σε διεργασίες. Το σετ περιλαμβάνει δύο επιπλέον κορυφές: μια πηγή (*source*) και μία δίνη (*sink*), τα οποία αντιπροσωπεύουν την έναρξη και το τέλος του προγράμματος. Η πηγή είναι ισοδύναμη με ένα εικονικό T_0 την διεργασία που προηγείται όλων των άλλων διεργασιών και η δίνη με ένα εικονικό T_{n+1}, που ακολουθεί όλες τις διεργασίες. Και οι δύο διεργασίες έχουν μηδενικό χρόνο επεξεργασίας. Το C είναι ένα σύνολο τόξων τα οποία αντανακλούν τους περιορισμούς προτεραιότητας, συνδέοντας δύο διαδοχικές διεργασίες της ίδιας εργασίας. Οι μη προσδιορισμένες διαζευκτικές ακμές που ανήκουν στο σύνολο D συνδέουν διεργασίες οι οποίες απαιτούν την ίδια μηχανή για την εκτέλεσή τους. Κάθε τόξο είναι επισημασμένο με θετικό βάρος ίσο προς το χρόνο επεξεργασίας του έργου, όπου το τόξο αρχίζει.

Το πρόβλημα χρονοπρογραμματισμού κατά παραγγελία απαιτεί μια βέλτιστη σειρά εργασιών σχετικών με τις μηχανές, με αποτέλεσμα ένα χρονοδιάγραμμα με το ελάχιστο μήκος. Στο διαζευκτικό μοντέλο, αυτό είναι ισοδύναμο με το να επιλέγεται ένα τόξο σε κάθε διάζευξη. Με τον καθορισμό κατευθύνσεων σε όλες τις διαζευκτικές ακμές, η σειρά εκτέλεσης όλων των αντικρουόμενων διεργασιών που απαιτούν την ίδια μηχανή, προσδιορίζεται και ένα πλήρες πρόγραμμα αποκτάται. Επιπλέον, το γράφημα που προκύπτει πρέπει να είναι ακυκλικό και αν είναι βέλτιστο, το μήκος του μακρύτερου μονοπατιού από την πηγή στην δίνη είναι ελάχιστο. Αυτή η μεγαλύτερη διαδρομή καθορίζει το συνολικό χρόνο εκτέλεσης των εργασιών, είναι η διάρκεια της μακρύτερης αλυσίδας των διεργασιών σε ένα πρόβλημα κατά παραγγελία.

Παράδειγμα: Ένα πρόβλημα κατά παραγγελία αποτελείται από ένα σύνολο τριών μηχανών $M = \{M_1, M_2, M_3\}$ και ένα σύνολο τριών εργασιών $J = \{J_1, J_2, J_3\}$ που περιγράφονται από τις ακόλουθες σειρές εργασιών $J_1: T_1 \rightarrow T_2 \rightarrow T_3$, και $J_2: T_4 \rightarrow T_5$, $J_3: T_6 \rightarrow T_7 \rightarrow T_8$. Για κάθε εργασία T_i η απαιτούμενη μηχανή $M(T_i)$ και ο χρόνος επεξεργασίας p_i παρουσιάζονται στον πίνακα 1.1. Το διαζευκτικό γράφημα για τη δεδομένη περίσταση παρουσιάζεται στο σχήμα 1.1.

Πίνακας 1.1: Χρόνοι επεξεργασίας διεργασιών

	M_1	M_2	M_3
J_1	3(T_1)	2(T_2)	3(T_3)
J_2	-	4(T_5)	3(T_4)
J_3	3(T_7)	6(T_6)	2(T_8)



Σχήμα 1.1: Διάγραμμα διαζευκτικών γράφων του παραδείγματος

[Πηγή: Jacek Blazewicz, Erwin Pesch, Malgorzata Sterna: The disjunctive graph machine representation of the job shop scheduling problem]

1.4 ΠΡΟΒΛΗΜΑ ΑΝΑΦΟΡΑΣ

Προκειμένου το πρόβλημα να γίνει πιο κατανοητό θα χρησιμοποιηθεί ένα πρόβλημα μικρού μεγέθους. Το πρόβλημα αποτελείται από 3 εργασίες οι οποίες πρέπει να εκτελεστούν σε 3 μηχανές. Η ακολουθία των μηχανών για κάθε εργασία καθώς και οι χρόνοι επεξεργασίας της κάθε διεργασίας παρουσιάζονται στους πίνακες 1.2 και 1.3 αντίστοιχα.

Πίνακας 1.2: Ακολουθίες εργασιών

J_1	1	2	3
J_2	3	1	2
J_3	2	3	1

Πίνακας 1.3: Χρόνοι επεξεργασίας διεργασιών

	M_1	M_2	M_3
J_1	12	15	6
J_2	7	20	8
J_3	8	9	12

Ας υποθέσουμε το ακόλουθο αυθαίρετο εφικτό πρόβλημα, με βάση τον περιορισμό ακολουθίας. Προκύπτει για κάθε μηχανή:

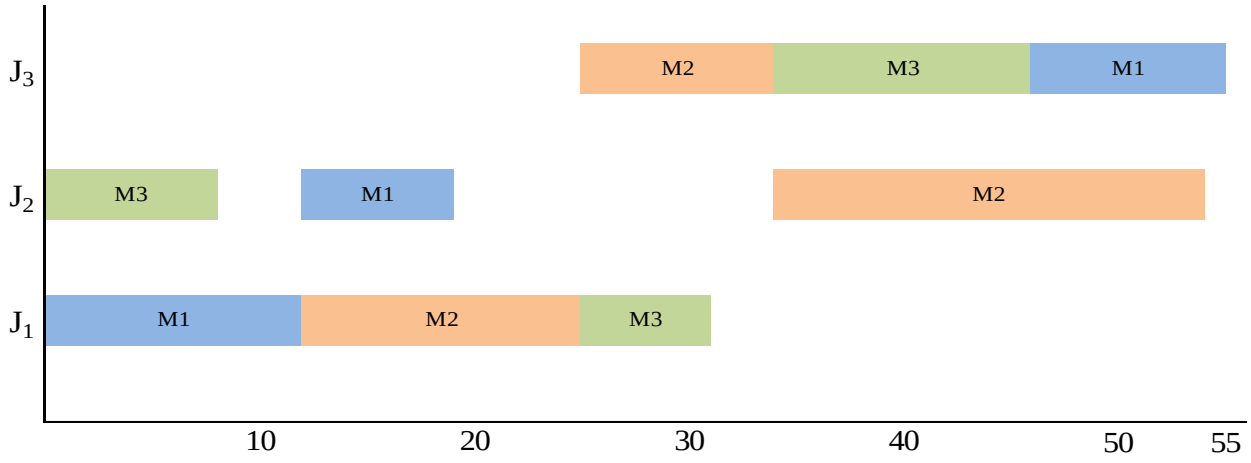
M_1: (J_1, J_2, J_3)
M_2: (J_1, J_3, J_2)
M_3: (J_2, J_1, J_3)

Ή διαφορετικά χρησιμοποιώντας διεργασίες:

M_1: (o_{11}, o_{21}, o_{31})
M_2: (o_{12}, o_{32}, o_{22})
M_3: (o_{23}, o_{13}, o_{33})

Ο παραπάνω τρόπος απεικόνισης έχει περιορισμένες πληροφορίες. Καθορίζεται σαφώς η σειρά των διεργασιών σε κάθε μηχανή, δεν ορίζονται όμως οι χρόνοι που αρχίζει η κάθε διεργασία.

Για να καλυφθεί αυτή η ανάγκη χρησιμοποιούνται τα διαγράμματα Gantt, τα οποία είναι πολύ δημοφιλή όσον αφορά την οπτικοποίηση στον τομέα του χρονοπρογραμματισμού. Στο σχήμα 1.2 απεικονίζεται το αυθαίρετο πρόγραμμα που περιγράφηκε παραπάνω.



Σχήμα 1.2: Διάγραμμα Gantt του προγράμματος

Η απεικόνιση του διαγράμματος προσφέρει πολλές ευκολίες στο χρήστη. Γίνεται αμέσως προφανές ότι ικανοποιούνται οι περιορισμοί ακολουθίας, καθώς όλες οι διεργασίες εκτελούνται με την προκαθορισμένη σειρά. Ικανοποιούνται επίσης οι περιορισμοί χωρητικότητας όλων των μηχανών. Οι χρόνοι αρχής εκτέλεσης της κάθε διεργασίας υπολογίστηκαν με βάση την σειρά εκτέλεσης που προέκυψε από το πρόγραμμα, λαμβάνοντας υπ' όψιν όλους τους περιορισμούς.

Άλλη μία παρατήρηση που προκύπτει από τη μελέτη του διαγράμματος είναι ότι όλες οι διεργασίες ξεκινούν το νωρίτερο δυνατό. Καμία διεργασία δεν μπορεί να ξεκινήσει νωρίτερα χωρίς να παραβιαστεί κάποιος περιορισμός ή να αλλάξει η σειρά εκτέλεσης. Θα μπορούσε όμως να καθυστερήσει η έναρξη της εργασίας 2 στην μηχανή 1 για 15 χρονικές μονάδες χωρίς να μεταβάλει το συνολικό χρόνο εκτέλεσης C_{max}.

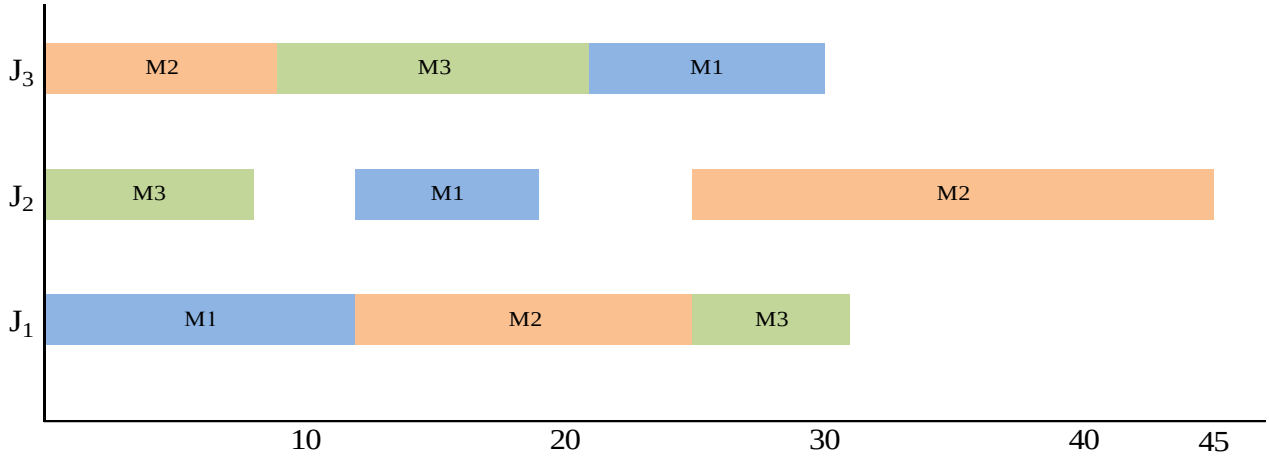
1.5 ΕΙΔΗ ΠΡΟΓΡΑΜΜΑΤΩΝ

Στη θεωρία του προγραμματισμού, τα προγράμματα είναι ταξινομημένα σύμφωνα με κάποιες συγκεκριμένες ιδιότητες. Σχετικά με το διάγραμμα 1.2 σημειώθηκε ότι καμία εργασία δεν μπορεί να ξεκινήσει νωρίτερα χωρίς να αλλάξει η σειρά εκτέλεσης κάποιων διεργασιών. Έτσι προκύπτει ο εξής ορισμός:

Ορισμός 1.1: Ένα εφικτό πρόγραμμα ονομάζεται ημιενεργό (*semi-active*) αν καμία διεργασία δεν μπορεί να ξεκινήσει νωρίτερα χωρίς να αλλάξει πρώτα η σειρά επεξεργασίας στην αντίστοιχη μηχανή.

Μελετώντας το διάγραμμα 1.2 φαίνεται ότι όντως κάθε διεργασία εκκινεί το συντομότερο δυνατό τηρώντας, για κάθε μηχανή, την σειρά που έχει οριστεί από το αυθαίρετο πρόγραμμα που ορίστηκε στην ενότητα 1.4. Παρατηρείται όμως ότι κάποιες διεργασίες μπορούν να μετακινηθούν νωρίτερα χωρίς να επηρεάσουν τον χρόνο

εκκίνησης των υπόλοιπων. Αυτό ισχύει για την διεργασία o_{23} η οποία μπορεί να μετακινηθεί νωρίτερα και να εκκινήσει τη χρονική στιγμή 0. Ομοίως μετακινούνται νωρίτερα οι διεργασίες o_{33}, o_{31}. Μετά από αυτές της αλλαγές η διεργασία o_{22} δεν εκκινεί το νωρίτερο δυνατό οπότε μετακινείται και αυτή νωρίτερα. Το αποτέλεσμα όλων αυτών των αλλαγών είναι η μείωση του συνολικού χρόνου εκτέλεσης. Έτσι προκύπτει το διάγραμμα 1.3.



Σχήμα 1.3: Διάγραμμα Gantt του ενεργού προγράμματος

Το νέο πρόγραμμα εξακολουθεί να ικανοποιεί όλους τους περιορισμούς. Σε αυτό το πρόγραμμα δεν μπορεί να γίνει μετατόπιση άλλης εργασίας χωρίς να καθυστερήσει η εκκίνηση μιας άλλης εργασίας. Αυτό το πρόγραμμα ονομάζεται ενεργό.

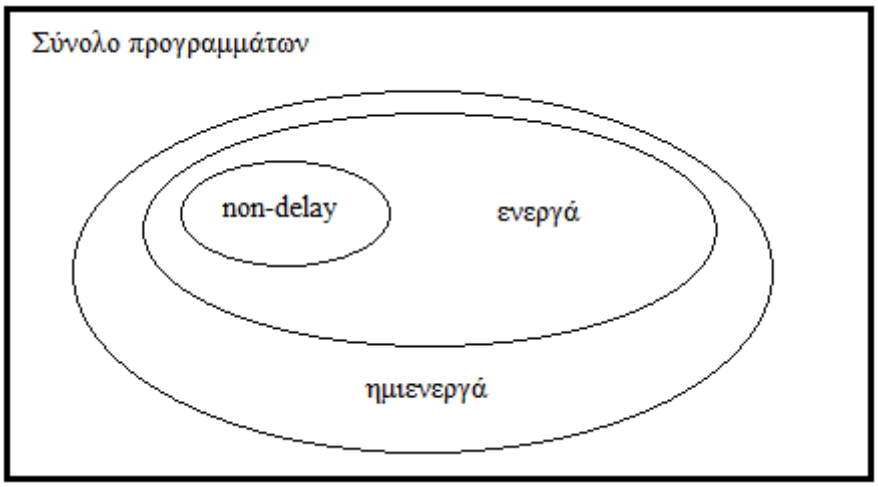
Ορισμός 1.2: Ένα εφικτό πρόγραμμα ονομάζεται ενεργό (*active*) αν καμία διεργασία δεν μπορεί να ξεκινήσει νωρίτερα αλλάζοντας τη σειρά επεξεργασίας σε οποιαδήποτε από τις μηχανές χωρίς να καθυστερήσει κάποια άλλη διεργασία.

Αυτό σημαίνει πως για να την εύρεση του βέλτιστου προγράμματος ενός προβλήματος, είναι αρκετό να ψάξουμε το σύνολο των ενεργών προγραμμάτων το οποίο ουσιαστικά είναι ένα υποσύνολο από το σύνολο των ημιενεργών προγραμμάτων και συνήθως είναι αρκετά μικρότερο. Φυσικά δεν είναι πρακτική η κατασκευή ενεργών προγραμμάτων από ημιενεργά τροποποιώντας τις σειρές των διεργασιών.

Μία τρίτη κατηγορία προγραμμάτων προκύπτει από την απαλοιφή των περιττών χρόνων αδράνειας των μηχανών:

Ορισμός 1.3: Ένα εφικτό πρόγραμμα ονομάζεται *non-delay* αν καμία μηχανή δεν παραμένει αδρανής όταν υπάρχει τουλάχιστον μία διεργασία έτοιμη για εκτέλεση.

Οι προϋποθέσεις των προγραμμάτων non-delay είναι πιο απαιτητικές οδηγώντας στις σχέσεις που φαίνονται στο διάγραμμα Venn στο σχήμα 1.4.



Σχήμα 1.4: Σχέση κατηγοριών προγραμμάτων

1.6 ΜΑΘΗΜΑΤΙΚΗ ΜΟΝΤΕΛΟΠΟΙΗΣΗ

Το πρόβλημα χρονοπρογραμματισμού μηχανών κατά παραγγελία μπορεί να μοντελοποιηθεί μαθηματικά με διάφορες μεθόδους. Η πιο διαδεδομένη είναι αυτή που εισήγαγε ο Manne (1960) με τη χρήση μικτού ακέραιου προγραμματισμού (*mixed integer programming*).

Έστω ότι οι όλες οι διεργασίες λαμβάνουν ένα σειριακό αριθμό από το 1 έως το N, όπου N ο συνολικός αριθμός των διεργασιών. Ορίζονται οι εξής ομάδες: $\bar{V}$ στην οποία ανήκουν όλες οι διεργασίες, $\bar{W}$= {(i, j)| όπου το j προηγείται αμέσως του i στην ίδια εργασία} η ομάδα περιορισμών ακολουθίας και $\bar{V}^k$ οι διεργασίες που θα εκτελεστούν στην μηχανή M_k. Χρησιμοποιώντας την ακέραια μεταβλητή $\bar{s_i}$ που συμβολίζει τη χρονική στιγμή εκκίνησης της εργασίας i και την σταθερά $\bar{p_i}$ που συμβολίζει τη διάρκεια της έχουμε το εξής πρόγραμμα:

$$min\ C_{max}$$

υπό περιορισμούς

$$\bar{s_i} \geq \bar{s_j} + \bar{p_j}, \forall (i,j) \in \bar{W} \quad (1.10)$$

$$\bar{s_i} \geq \bar{s_j} + \bar{p_j} \text{ ή } \bar{s_j} \geq \bar{s_i} + \bar{p_i}, \forall\, 1 \leq k \leq m, \forall (i,j) \in \bar{V}^k, i \neq j \quad (1.11)$$

$$C_{max} \geq \bar{s_i} + \bar{p_i}, \forall\, 1 \leq i \leq N \quad (1.12)$$

$$\bar{s_i} \geq 0, \forall\, 1 \leq i \leq N \quad (1.13)$$

Η σχέση 1.11 μπορεί να αναλυθεί με τον ακόλουθο τρόπο χρησιμοποιώντας την δυαδική μεταβλητή y_{ij}:

$$y_{ij} = \begin{cases} 1, αν\ το\ i\ προηγείται\ του\ j\ στην\ συγκεκριμένη\ μηχανή \\ 0, αλλιώς \end{cases} \quad (1.14)$$

Με δεδομένο ότι το ζευγάρι (i, j) διεργασιών σε μία μηχανή οι περιορισμοί παίρνουν την μορφή:

$$M \cdot y_{ij} + \overline{s_i} \geq \overline{s_j} + \overline{p_j} \quad (1.15)$$

$$M \cdot \left(1 - y_{ij}\right) + \overline{s_j} \geq \overline{s_i} + \overline{p_i} \quad (1.16)$$

όπου Μ είναι ένας πολύ μεγάλος αριθμός.

ΚΕΦΑΛΑΙΟ 2

ΓΕΝΕΤΙΚΟΙ ΑΛΓΟΡΙΘΜΟΙ

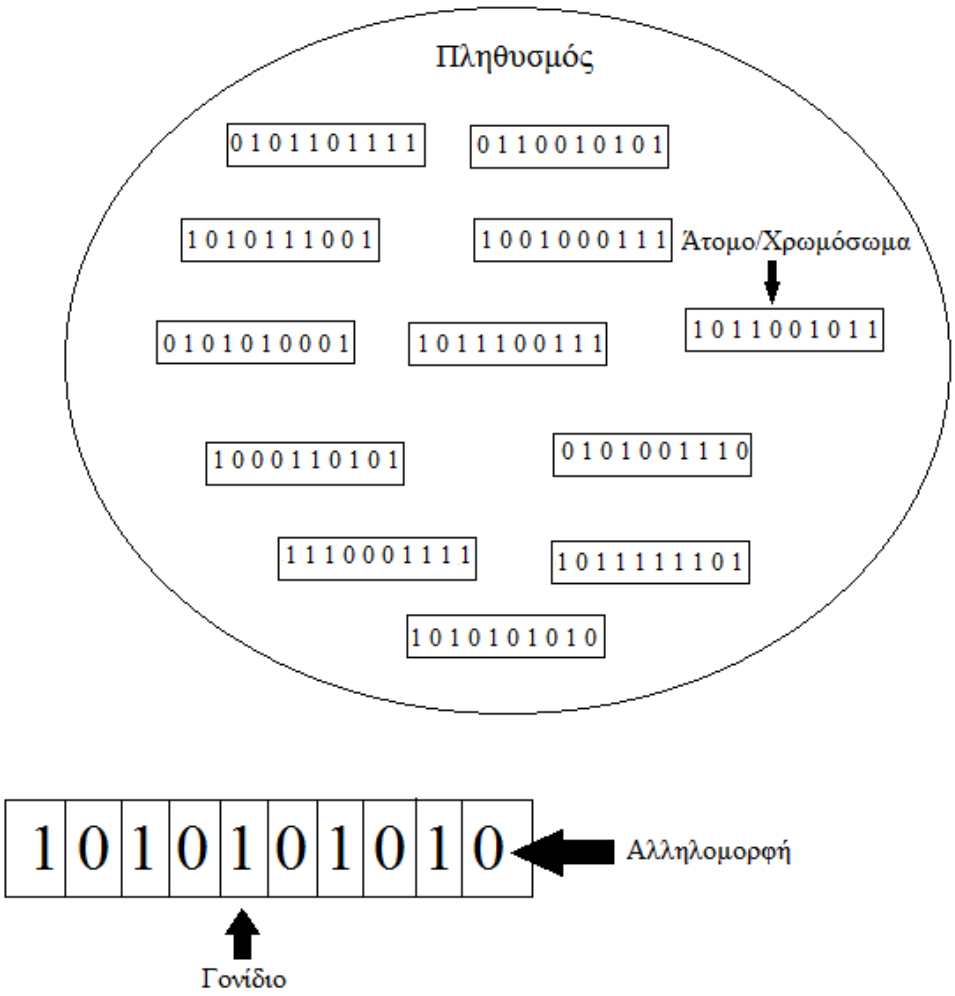
2.1 ΕΙΣΑΓΩΓΗ

Οι Γενετικοί Αλγόριθμοι (*Genetic Algorithms*) πραγματοποιούν αναζήτηση βέλτιστης λύσης συνδυάζοντας υπάρχουσες λύσεις και ανήκουν στην κατηγορία των Εξελικτικών Αλγορίθμων (*Evolutionary Algorithms*). Όπως δηλώνει και το όνομα τους μιμούνται μηχανισμούς εξέλιξης της φύσης, όπως η φυσική επιλογή, συνδυασμός γονιδίων και μετάλλαξη για να ελέγξουν την διαδικασία αναζήτησης.

Όπως αναφέρθηκε οι γενετικοί αλγόριθμοι είναι βασισμένοι στον τρόπο που λειτουργεί η φύση. Έτσι πρέπει να εισαχθούν όροι που χρησιμοποιούνται από την επιστήμης της βιολογίας. Οι γενετικοί αλγόριθμοι χρησιμοποιούν μια αρχική ομάδα λύσεων για να παράγουν νέες λύσεις, η οποία ονομάζεται πληθυσμός (*population*). Κάθε ένα μέλος του πληθυσμού (*individual*) αντιπροσωπεύει μία λύση. Τα χαρακτηριστικά του κάθε ατόμου αναπαριστώνται από ένα χρωμόσωμα (*chromosome*). Τα χρωμοσώματα αποτελούνται από γονίδια (*genes*) και οι τιμές που παίρνουν ονομάζονται αλληλομορφές (*alleles*). Το γονίδιο, που είναι η μικρότερη ποσότητα πληροφορίας, είναι μία ιδιότητα του χρωμοσώματος η οποία παίρνει τιμές από ένα προκαθορισμένο σύνολο. Το σύνολο των γονιδίων ενός χρωμοσώματος περιγράφουν τον γενότυπο (*genotype*) του. Ο φαινότυπος (*phenotype*) περιγράφει τον τρόπο με τον οποία εκφράζονται τα γονίδια του ατόμου.

Άτομα του πληθυσμού τα οποία συνδυάζονται ώστε να προκύψουν νέες λύσεις ονομάζονται γονείς (*parents*) και κάθε λύση που προκύπτει ονομάζεται παιδί (*child*) ή απόγονος (*offspring*). Η διαδικασία του συνδυασμού ονομάζεται διασταύρωση (*crossover*).

Στο προηγούμενο κεφάλαιο αναφέρθηκε ότι η ποιότητα μίας λύσης προκύπτει από την τιμή μιας αντικειμενικής συνάρτησης. Στους γενετικούς αλγορίθμους η καταλληλότητα ενός απογόνου (*fitness*) μπορεί να καθορίζεται με διαφορετικό τρόπο. Σε κάθε περίπτωση όμως, πρέπει να υπολογίζεται η καταλληλότητα κάθε μέλους του πληθυσμού για να χρησιμοποιείται σαν κριτήριο για επιλογή και αντικατάσταση.



Σχήμα 2.1: Οπτικοποίηση των όρων των γενετικών αλγορίθμων

Κατά την εκτέλεση ενός εξελικτικού αλγορίθμου συμβαίνει ένα από τα εξής δυο σενάρια: είτε ο αλγόριθμος συγκλίνει σε ένα βέλτιστο πληθυσμό, είτε όχι. Το φαινόμενο κατά το οποίο ο αλγόριθμος συγκλίνει σε ένα τοπικό ελάχιστο ονομάζεται πρόωρη σύγκλιση (*premature convergence*). Η αιτία αυτού του προβλήματος εντοπίζεται στην απώλεια πληροφοριών, οπότε η κάθε μέθοδος πρέπει να διατηρεί τις απαραίτητες πληροφορίες μέσα στον πληθυσμό.

2.2 ΚΥΡΙΑ ΣΤΟΙΧΕΙΑ ΕΝΟΣ ΓΕΝΕΤΙΚΟΥ ΑΛΓΟΡΙΘΜΟΥ

Ένας γενετικός αλγόριθμος αποτελείται από αρκετά στοιχεία τα οποία μαζί συνθέτουν μία αξιόπιστη και γρήγορη μέθοδο. Σε αυτή την ενότητα θα παρουσιαστούν αυτά τα στοιχεία καθώς και ο τρόπος με τον οποία αλληλεπιδρούν.

Για να λειτουργήσει ένας γενετικός αλγόριθμος χρειάζεται έναν πληθυσμό (*P*). Έτσι το πρώτο βήμα είναι η αρχικοποίηση αυτού του πληθυσμού (*initialization*). Έπειτα γίνεται η αξιολόγηση της καταλληλότητας των ατόμων του πληθυσμού (*evaluation*). Αυτά τα δύο βήματα είναι η αρχική φάση του αλγορίθμου. Αφού ολοκληρωθούν ξεκινά μία επαναληπτική διαδικασία.

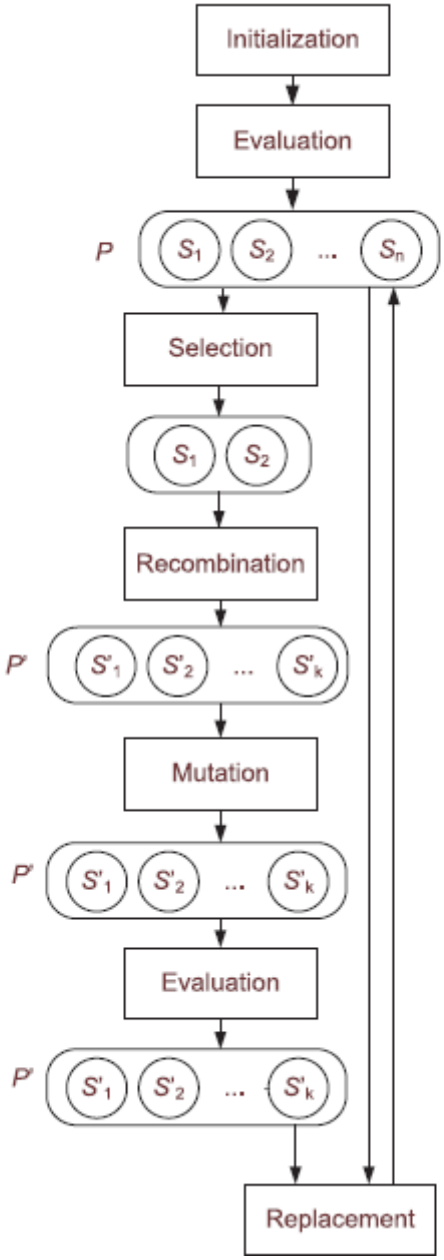
Από τον υπάρχοντα πληθυσμό μία διαδικασία επιλογής (*selection*) θα καθορίσει ποια από τα άτομα του πληθυσμού θα αναπαραχθούν (*recombination*) για να δημιουργήσουν νέα άτομα. Στους κλασσικούς γενετικούς αλγόριθμους επιλέγονται δύο άτομα και διασταυρώνονται. Μετά την διασταύρωση ένα μέρος των νέων ατόμων μεταλλάσσεται (mutation), δηλαδή υπόκειται σε μικρές βελτιώσεις. Τελικά τα νέα άτομα αξιολογούνται και συγκεντρώνονται σε ένα νέο σύνολο, ένα προσωρινό πληθυσμό(*P'*).

Με αυτό τον τρόπο δημιουργείται μία ομάδα νέων λύσεων μέχρι ο νέος πληθυσμός να περιέχει αρκετές λύσεις για αντικατάσταση (replacement). Το μέγεθος του προσωρινού πληθυσμού παρουσιάζει σημαντικές διαφορές μεταξύ διαφορετικών γενετικών αλγορίθμων. Όταν λοιπόν τελειώσει η δημιουργία του πληθυσμού *P'*, ο αλγόριθμος επιλέγει από το σύνολο $P \cup P'$ τα άτομα που θα απαρτίσουν το πληθυσμό *P* στην επόμενη επανάληψη. Επιλέγονται δηλαδή οι λύσεις από το νέο πληθυσμό που θα αντικαταστήσουν λύσεις από τον παλιό.

Συνοψίζοντας τα επτά στοιχεία που απαρτίζουν τους γενετικούς αλγόριθμους είναι:

- Πληθυσμός
- Αρχικοποίηση
- Αξιολόγηση
- Επιλογή
- Αναπαραγωγή
- Μετάλλαξη
- Αντικατάσταση

Η σειρά με την οποία εκτελούνται και η σχέση τους συνοψίζονται σε ένα διάγραμμα ροής ενός κλασσικού γενετικού αλγόριθμου στο σχήμα 2.2.



Σχήμα 2.2: Διάγραμμα ροής ενός γενετικού αλγόριθμου

[Πηγή: Günther Zäpfel, Roland Braune, Michael Bögl: Metaheuristic Search Concepts]

2.3 ΠΕΡΙΓΡΑΦΗ ΤΟΥ ΑΛΓΟΡΙΘΜΟΥ

Σε αυτήν την ενότητα θα παρουσιαστεί ο αλγόριθμος ενός κλασσικού γενετικού αλγορίθμου σε ψευδοκώδικα (αλγόριθμος 2.1). Αυτός ο αλγόριθμος, όπως και οι περισσότεροι, δεν λειτουργεί με άτομα αλλά με ολόκληρο τον πληθυσμό. Όταν, για

παράδειγμα, καλείται η συνάρτηση αναπαραγωγής δέχεται σαν όρισμα τον πληθυσμό *P* και επιστρέφει σαν αποτέλεσμα τον νέο πληθυσμό *P'*.

Αλγόριθμος 2.1: Κλασσικός Γενετικός Αλγόριθμος
Δημιούργησε αρχικό πληθυσμό *P*; Αξιολόγησε *P*; **Όσο** κριτήριο όχι ικανοποιημένο *P'* ← Συνδύασε *P*; Μετάλλαξε *P'*; Αξιολόγησε *P'*; *P* ← Αντικατάστησε *P* ∪ *P'*; **Τέλος επανάληψης**

Αρχικά, είναι απαραίτητος ένας αρχικός πληθυσμός. Η κατασκευή του πληθυσμού αυτού είναι ένα εξαρτημένο πρόβλημα. Μία λύση είναι η δημιουργία τυχαίων ατόμων, όμως αυτό είναι αποτελεσματικό μόνο σε συγκεκριμένα προβλήματα. Για άλλα προβλήματα είναι πιο εύκολη η κατασκευή με συγκεκριμένες ευρετικές μεθόδους. Όπως επισήμανε ο Reeves (2003), ο αρχικός πληθυσμός πρέπει να καλύπτει το εύρος των πιθανών λύσεων ομοιόμορφα, δηλαδή πρέπει να προτιμούνται ποικίλες διαφορετικές λύσεις. Ακόμα και ο καλύτερα δομημένος αλγόριθμος μπορεί να παράγει κακές λύσεις αν ο αρχικός πληθυσμός δεν έχει την απαραίτητη ποικιλία. Το τελευταίο βήμα κατά την αρχική φάση είναι να υπολογιστεί η καταλληλότητα των ατόμων, χρησιμοποιώντας μία κατάλληλη συνάρτηση αξιολόγησης.

Μετά την αρχική φάση ξεκινά το επαναληπτικό τμήμα. Οι επαναλήψεις συνεχίζονται μέχρι να ικανοποιηθεί ένα συγκεκριμένο κριτήριο το οποίο μπορεί να είναι ένας μέγιστος ή σταθερός αριθμός επαναλήψεων, μέγιστος χρόνος εκτέλεσης, σύγκλιση του πληθυσμού είτε κάποια άλλη προϋπόθεση.

Στην επανάληψη ένας αριθμός λύσεων πρέπει να επιλεχθεί ώστε να αναπαραχθούν και να προκύψουν νέες λύσεις. Η επιλογή γίνεται από μία αντίστοιχη συνάρτηση, η οποία καθορίζει τις λύσεις που θα επεξεργαστούν. Η διαδικασία της επιλογής μαζί με αυτήν της αντικατάστασης παίζουν σημαντικό ρόλο στην ταχύτητα της σύγκλισης στον βέλτιστο πληθυσμό.

Ο κύριος σκοπός της επιλογής είναι να προτιμά τις καλές λύσεις και να αποφεύγει τις κακές, αν όμως είναι πολύ ισχυρή η προτίμηση στις καλύτερες λύσεις ο αλγόριθμος μπορεί να συγκλίνει πρόωρα σε κάποιο τοπικό ελάχιστο. Αντιθέτως, αν η προτίμηση δεν είναι αρκετά ισχυρή ο αλγόριθμος μπορεί να μην βρει την βέλτιστη λύση. Για να είναι επιτυχημένη η αναζήτηση, πρέπει να είναι ισορροπημένη. Η ισορροπία αυτή εξαρτάται από το μέγεθος του πληθυσμού, από την συνάρτηση της αναπαραγωγής, της μετάλλαξης και της αντικατάστασης.

Η συνάρτηση της αναπαραγωγής είναι το σημείο, όπου δυο υπάρχουσες λύσεις διασταυρώνονται για να δημιουργήσουν μία νέα. Ο απόγονος των δύο προηγούμενων λύσεων είναι ένας συνδυασμός των δυνατοτήτων των δύο γονέων και πρέπει να είναι καλύτερη από τους δύο γονείς, επειδή η βελτίωση των λύσεων συμβαίνει κυρίως σε αυτό το τμήμα. Όπως είναι λογικό η συνάρτηση διασταύρωσης πρέπει να σχεδιαστεί με πολλή προσοχή. Θα πρέπει να διατηρεί τις καλές ιδιότητες των γονέων και να τις συνδυάζει με τέτοιο τρόπο ώστε ο απόγονος να είναι ανώτερος των γονέων.

Όταν παραχθεί ο πληθυσμός των νέων λύσεων, ένα μέρος του μεταλλάσσεται, δηλαδή μία συνάρτηση επεμβαίνει για να βελτιώσει μερικούς από αυτούς. Μέσω της μετάλλαξης επανεισάγονται στους απογόνους χαμένες πληροφορίες, η έλλειψη των οποίων θα μπορούσε να οδηγήσει σε πρόωρη σύγκλιση.

Τέλος, αφού δημιουργηθεί ο νέος πληθυσμός θα πρέπει να αξιολογηθεί η καταλληλότητα του, ώστε να αντικατασταθούν μερικά μέλη του παλαιού πληθυσμού με νέα ποιοτικότερα, μέσω της συνάρτησης επιλογής. Ο πληθυσμός που προκύπτει θα γίνει ο νέος P και θα χρησιμοποιηθεί στην επόμενη επανάληψη.

2.4 ΠΕΡΙΓΡΑΦΗ ΣΤΟΙΧΕΙΩΝ ΤΟΥ ΓΕΝΕΤΙΚΟΥ ΑΛΓΟΡΙΘΜΟΥ

Στην ενότητα 2.2 καταγράφηκαν τα στοιχεία που απαρτίζουν ένα γενετικό αλγόριθμο και στην ενότητα 2.3 περιγράφηκε η μεταξύ τους σχέση και ο τρόπος που η ακολουθία αυτών των εντολών κατασκευάζουν ένα γενετικό αλγόριθμο. Σε αυτή την ενότητα τα στοιχεία θα περιγραφούν αναλυτικότερα.

3.5.1 Πληθυσμός

Ο πληθυσμός είναι το σύνολο που περιέχει όλες τις λύσεις υποψήφιες για επεξεργασία. Στην αρχή της αναζήτησης ο πληθυσμός πρέπει να περιέχει τη μεγαλύτερη δυνατή ποικιλία λύσεων. Κατά τη διάρκεια της αναζήτησης η ποιότητα των λύσεων πρέπει να βελτιώνεται και να συγκλίνει στο βέλτιστο και στο τέλος να δίνει τη βέλτιστη λύση.

Μία σημαντική παράμετρος είναι το μέγεθος του πληθυσμού. Δεν πρέπει να είναι πολύ μικρό, ώστε να υπάρχει αρκετά μεγάλη ποικιλία λύσεων και αν αποτραπεί η πρόωρη σύγκλιση σε ένα τοπικό ελάχιστο, δεν μπορεί όμως να είναι πολύ μεγάλη, επειδή το πρόβλημα θα έχει μεγάλο υπολογιστικό φόρτο.

Δεν υπάρχει κάποια μέθοδος για τον υπολογισμό του βέλτιστου μεγέθους του πληθυσμού. Εξαρτάται κυρίως από μήκος του χρωμοσώματος και την μέθοδο που είναι

σχεδιασμένες οι υπόλοιπες συναρτήσεις. Συνήθως προκύπτει εμπειρικά είτε πειραματικά.

3.5.2 Αρχικοποίηση

Η αρχικοποίηση είναι το τμήμα στο οποίο δημιουργείται ο αρχικό πληθυσμός *P*. Είναι ένα καίριο σημείο επειδή η τελική λύση εξαρτάται κατά μεγάλο μέρος από τον αρχικό πληθυσμό. Όποια μέθοδος και να χρησιμοποιηθεί είναι πολύ σημαντικό, όπως αναφέρθηκε παραπάνω, να παράγει λύσεις οι οποίες διαφέρουν πολύ μεταξύ τους. Για αυτό το λόγο οι μέθοδοι που παράγουν τυχαίες λύσεις είναι πολύ δημοφιλείς, δεν είναι όμως πάντα εφαρμόσιμες καθώς μερικά προβλήματα έχουν πολύ αυστηρούς περιορισμούς και υπάρχει πιθανότητα να παράγουν μη εφικτές λύσεις. Αυτές οι παράμετροι πρέπει να λαμβάνονται πάντα υπ' όψιν όταν σχεδιάζεται η μέθοδος αρχικοποίησης.

3.5.3 Αξιολόγηση

Η συνάρτηση αξιολόγησης είναι αυτή που καθορίζει την ποιότητα των λύσεων, αυτό που στην ορολογία των γενετικών αλγορίθμων ονομάζεται καταλληλότητα. Η τιμή της ποιότητας χρειάζεται για την επιλογή των γονέων και την αντικατάσταση, που θα αναλυθούν παρακάτω.

Ανάλογα με τις μεθόδους που χρησιμοποιούνται στην αλγόριθμο και το πρόβλημα, η αξιολόγηση μπορεί να είναι ανάλογη της αντικειμενικής συνάρτησης. Σίγουρα είναι κάποιο είδους μέτρησης της ποιότητας, καθώς λύσεις με καλύτερη καταλληλότητα συνήθως προτιμούνται κατά της διαδικασία της επιλογής. Η καταλληλότητα των λύσεων κατευθύνει τη διαδικασία αναζήτησης, για αυτό θα πρέπει να δίνει πληροφορίες για την συγκριτική ποιότητα των λύσεων.

3.5.4 Επιλογή

Η συνάρτηση επιλογής καθορίζει ποιες από τις λύσεις θα χρησιμοποιηθούν για αναπαραγωγή. Συνήθως η επιλογή βασίζεται στην καταλληλότητα των λύσεων. Συχνά η τιμή της καταλληλότητας απεικονίζει την ποιότητα της λύσης και χρησιμοποιείται για την σύγκριση της με τις υπόλοιπες λύσεις του πληθυσμού. Όσο καλύτερη είναι η λύση, σύμφωνα με την αντικειμενική συνάρτηση του προβλήματος, τόσο καλύτερη είναι η καταλληλότητα της και είναι πιο πιθανό να επιλεχθεί για αναπαραγωγή.

Η διαδικασία συνήθως προτιμά της καλύτερες λύσεις για να κατευθύνουν την αναζήτηση, όμως υπάρχουν αρκετές μέθοδοι που διαφέρουν. Παρακάτω παρουσιάζονται οι πιο γνωστές:

- Αναπαραγωγή ανά δύο

Σε αυτή τη μέθοδο όλοι οι γονείς σχηματίζουν ζευγάρια ανά δύο και αναπαράγονται. Για παράδειγμα, σε ένα πληθυσμό αναπαράγονται το πρώτο άτομο με το δεύτερο, το τρίτο με το τέταρτο και ούτω καθεξής. Είναι μία εύκολη και κατανοητή μέθοδος. Μεγάλο της μειονέκτημα είναι ότι αγνοεί την ποιότητα των ατόμων. Όμως εκμεταλλεύεται ολόκληρο τον πληθυσμό και παράγει ποικίλες λύσεις, άρα είναι πιο δύσκολο να συγκλίνει πρόωρα.

- Κανόνας της ρουλέτας (Roulette wheel Selection)

Στον κανόνα της ρουλέτας επιλέγονται λύσεις ανάλογα με τη συγκριτική τους καταλληλότητα, δηλαδή όσο μεγαλύτερη καταλληλότητα έχει μία λύση τόσο πιθανότερο είναι να επιλεγεί για αναπαραγωγή. Έστω το ακόλουθο παράδειγμα μεγιστοποίησης: Ένας πληθυσμός αποτελείται από πέντε λύσεις οι οποίες έχουν τις εξής τιμές καταλληλότητας: $f(s_1) = 10$, $f(s_2) = 20$, $f(s_3) = 30$, $f(s_4) = 40$, $f(s_5) = 50$.

Η πιθανότητα να επιλεγεί η λύση s_1 είναι πολύ μικρότερη από την s_5, συγκεκριμένα είναι υπόπενταπλάσια. Η πιθανότητα επιλογής υπολογίζεται με τον ακόλουθο τύπο:

$$p(s_i) = \frac{f(s_i)}{\sum_{k=1}^{n}(s_k)} \tag{2.1}$$

Για τις πέντε λύσεις του παραδείγματος η εξίσωση αυτή δίνει τις εξής πιθανότητες: $p(s_1) = \frac{1}{15}$, : $p(s_2) = \frac{2}{15}$, $p(s_3) = \frac{3}{15}$, : $p(s_4) = \frac{4}{15}$, : $p(s_5) = \frac{5}{15}$ οι οποίες αθροίζουν στο 1.

Ο κανόνας της ρουλέτας είναι μία πολύ δημοφιλής μέθοδος, όμως έχει κάποια μειονεκτήματα. Κατά την εκτέλεση του αλγορίθμου μπορεί να προκύψει μία λύση η οποία είναι πολύ καλύτερη σε σχέση με τον υπόλοιπο πληθυσμό. Αυτό έχει σαν αποτέλεσμα αυτή η λύση να υπερισχύει και να έχει υπερβολικά μεγάλη πιθανότητα να επιλεγεί σαν γονέας. Ένα τέτοιο σενάριο είναι πολύ πιθανό να οδηγήσει σε πρόωρη σύγκλιση.

- Γραμμική Κατάταξη (Linear Rank Selection)

Η μέθοδος της γραμμικής κατάταξης μοιάζει αρκετά με την μέθοδο της ρουλέτας. Ταξινομεί κάθε λύση συγκρίνοντας την με τις υπόλοιπες και χρησιμοποιεί την θέση τις στην κατάταξη σαν κριτήριο για επιλογή.

Για παράδειγμα, έστω οι λύσεις οι οποίες έχουν τις εξής τιμές καταλληλότητας: $f(s_1) = 1$, $f(s_2) = 50$, $f(s_3) = 12$, $f(s_4) = 13$, $f(s_5) = 21$. Είναι φανερό ότι το μεγάλο εύρος τιμών θα προκαλούσε πρόβλημα στη μέθοδο της ρουλέτας. Ταξινομούνται, λοιπόν, οι λύσεις κατά αύξουσα σειρά (s_1, s_3, s_4, s_5, s_2). Από αυτό το στάδιο και έπειτα αγνοείται η τιμή της καταλληλότητας κάθε λύσης και χρησιμοποιείται σαν κριτήριο για την ποιότητα της η θέση της στην κατάταξη. Με βάση αυτή υπολογίζεται η πιθανότητα επιλογής της κάθε λύσης για αναπαραγωγή συμφώνα με τον τύπο: $p(s_i) = \frac{r(s_i)}{\sum_{k=1}^{n} r(s_k)}$, όπου r είναι η θέση της λύσης s_k στην κατάταξη. Άρα οι πιθανότητες διαμορφώνονται ως εξής: $p(s_1) = \frac{1}{15}$, : $p(s_2) = \frac{5}{15}$, $p(s_3) = \frac{2}{15}$, : $p(s_4) = \frac{3}{15}$, : $p(s_5) = \frac{5}{15}$.

Είναι προφανές ότι λύσεις με πολύ μεγαλύτερη καταλληλότητα συγκριτικά με τη υπόλοιπες δεν υπερέχουν υπερβολικά.

- Μέθοδος τουρνουά

Άλλη μία ενδιαφέρουσα μέθοδος είναι η μέθοδος τουρνουά. Σε αυτή τη μέθοδο επιλέγονται τυχαία από τον πληθυσμό ένας αριθμός από ομάδες λύσεων και από την κάθε ομάδα αναδεικνύεται η καλύτερη. Οι λύσεις αυτές αναπαράγονται μεταξύ τους.

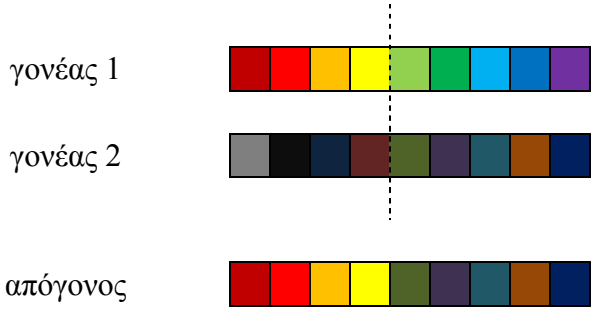
Για παράδειγμα έστω ο πληθυσμός των πέντε ατόμων: $\{s_1, s_2, s_2,, s_4, s_5\}$. Οι ποιότητα τις κάθε λύσης είναι αδιάφορη στο πρώτο βήμα. Αρχικά πρέπει να επιλεχθεί το μέγεθος του τουρνουά, για αυτό το παράδειγμα έστω δύο. Άρα δημιουργούνται δύο τυχαίες ομάδες $\{s_2, s_3\}$ και $\{s_3, s_4\}$. Σε αυτό το στάδιο καθορίζεται η καλύτερη λύση για κάθε σύνολο. Για τιμές καταλληλότητας $f(s_2) = 20$, $f(s_3) = 30$, $f(s_4) = 40$ προκύπτουν οι λύσεις s_3 από την πρώτη ομάδα και s_4 από την δεύτερη, οι οποίες θα αναπαραχθούν μεταξύ τους. Παρατηρείται ότι κάθε άτομο μπορεί να συμμετέχει σε περισσότερες από μία ομάδες. Είναι, επίσης, δυνατό το μέγεθος κάθε ομάδας να διαφέρει.

3.5.5 Αναπαραγωγή

Η αναπαραγωγή είναι η διαδικασία κατά την οποία οι υπάρχουσες λύσεις (άτομα ή χρωμοσώματα) συνδυάζονται ώστε να παράγουν νέες λύσεις. Όπως αναφέρθηκε η διαδικασία ονομάζεται και διασταύρωση στην ορολογία των γενετικών αλγορίθμων. Η ιδέα είναι απλή: Αν δύο λύσεις συνδυάσουν τα «καλά» χαρακτηριστικά

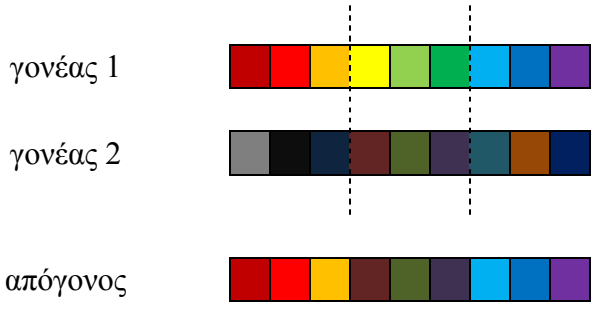
τους σε μία νέα λύση τότε η λύση αυτή θα είναι καλύτερη από τις δύο που την παρήγαγαν. Για να επιτευχθεί αυτό θα πρέπει η συνάρτηση να έχει σχεδιαστεί προσεκτικά και η κωδικοποίηση των λύσεων να είναι σωστή και προσαρμοσμένη στο πρόβλημα.

Κατά τη διασταύρωση ενός σημείου (*1-point crossover*) δύο χρωμοσωμάτων, αρχικά, επιλέγεται ένα σημείο κοπής (*crossover point*) για τους δύο γονείς. Το σημείο κοπής συνήθως επιλέγεται τυχαία, όμως σε μερικά προβλήματα μπορεί να επιλέγονται με κάποια συγκεκριμένη λογική. Ο απόγονος που προκύπτει από τα δύο χρωμοσώματα αποτελείται από τον πρώτο γονέα από την αρχή του μέχρι το σημείο κοπής και από τον δεύτερο γονέα από το σημείο κοπής μέχρι το τέλος του. Για παράδειγμα στο σχήμα 2.3 επιλέγεται ως σημείο κοπής το σημείο μετά το τέταρτο γονίδιο. Οπότε ο απόγονος που προκύπτει κληρονομεί τα τέσσερα πρώτα γονίδια από τον πρώτο γονέα και τα υπόλοιπα πέντε από τον δεύτερο γονέα.



Σχήμα 2.3: Διασταύρωση ενός σημείου

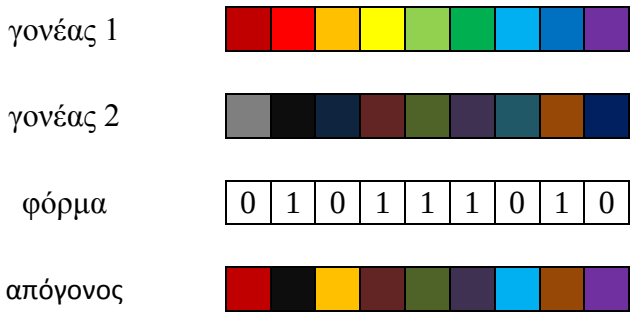
Αντίστοιχα μπορεί να γίνει και διασταύρωση δύο σημείων (*2-point crossover*) όπως στο σχήμα 2.4. Ο απόγονος που προκύπτει κληρονομεί τα τρία πρώτα γονίδια του από τον πρώτο γονέα, τα επόμενα τρία από τον δεύτερο γονέα και τα τελευταία τρία πάλι από τον πρώτο γονέα.



Σχήμα 2.4: Διασταύρωση δύο σημείων

Με αυτή τη λογική μπορεί η διαδικασία να γενικευτεί σε διασταύρωση *m* σημείων, όπου *m* > 1. Τυχαία σημεία κοπής επιλέγονται και μία λύση κατασκευάζεται συνδυάζοντας εναλλάξ τα γονίδια των δύο γονέων.

Άλλη μία εναλλακτική είναι η ομοιόμορφη διασταύρωση (*uniform crossover*) η όποια αποφασίζει από ποιο γονέα θα προέλθει το κάθε γονίδιο του απογόνου, ανεξάρτητα από τα υπόλοιπα γονίδια. Για αυτό το σκοπό δημιουργείται μία φόρμα (*uniform mask*) η οποία υποδεικνύει για κάθε γονίδιο τον γονέα από τον οποίο προέρχεται. Παράδειγμα φαίνεται στο σχήμα 2.5. Όταν σε μία θέση της φόρμας υπάρχει 0 σημαίνει ότι στην αντίστοιχη θέση του απόγονου χρησιμοποιείται το γονίδιο που βρίσκεται στην αντίστοιχη θέση του γονέα 1. Αντίστοιχα 1 σημαίνει ότι χρησιμοποιείται το γονίδιο του γονέα 2.



Σχήμα 2.5: Ομοιόμορφη διασταύρωση

Η διαδικασία της αναπαραγωγής χρησιμοποιεί ιδιότητες από υπάρχουσες λύσεις, δεν δημιουργεί νέες πληροφορίες. Για αυτό το λόγο η απώλεια μερικών πληροφοριών μπορεί να οδηγήσει σε πρόωρη σύγκλιση σε τοπικό ελάχιστο. Για αυτό χρησιμοποιείται και η διαδικασία της μετάλλαξης.

3.5.6 Μετάλλαξη

Η μετάλλαξη είναι μία μικρή τυχαία διαταραχή στην λύση. Σκοπός της διαδικασίας είναι να επανακτήσει πληροφορίες που πιθανώς χάθηκαν με την διασταύρωση και αποτρέψει τον πληθυσμό να υποπέσει σε πρόωρη σύγκλιση.

Γενικά η μετάλλαξη είναι μία δευτερεύουσα διαδικασία που συμβαίνει συνήθως σε μικρό ποσοστό του πληθυσμού. Σε μερικούς αλγορίθμους είναι πιθανό να απουσιάζει. Παράδειγμα μετάλλαξης ακολουθεί στο σχήμα 2.6 όπου το γονίδιο στη θέση 3 αντιμετατίθεται με αυτό στη θέση 4.

Σχήμα 2.6: Μετάλλαξη

3.5.7 Αντικατάσταση

Μετά την αναπαραγωγή, την μετάλλαξη και την αξιολόγηση υπάρχουν δύο πληθυσμοί. Σε έναν κλασσικό γενετικό αλγόριθμο το μέγεθος του πληθυσμού είναι σταθερό, οπότε πρέπει να αποφασιστεί ποια άτομα θα διατηρηθούν και ποια θα χαθούν. Η διαδικασία της αντικατάστασης επιλέγει τα άτομα που θα επιβιώσουν, δηλαδή αυτά που θα διατηρηθούν για την επόμενη επανάληψη.

Υπάρχουν κυρίως δύο σχέδια για να γίνει η αντικατάσταση σε έναν γενετικό αλγόριθμο: η γενεαλογική αντικατάσταση (*generational replacement*) και η αντικατάσταση σταθερής κατάστασης (*steady state replacement*).

Κατά την γενεαλογική αντικατάσταση η νέα γενιά αντικαθιστά την παλαιότερη. Αυτή η μέθοδος έχει ένα μεγάλο μειονέκτημα· όταν αντικαθιστάται μία ολόκληρη υπάρχει μεγάλη πιθανότητα να χαθούν κάποιες καλές λύσεις. Έτσι ο De Jong (1975) εισήγαγε την έννοια του ελιτισμού (elitism), όπου η καλύτερη λύση του παλαιότερου πληθυσμού διατηρείται και συμπεριλαμβάνεται στο νέο πληθυσμό. Αντίστοιχα μπορεί να επιλεγούν *n* λύσεις για να διατηρηθούν αντί για μία. Σε αυτή την περίπτωση έχουμε n-ελιτισμό (*n-elitism*).

Η αντίθετη ακραία περίπτωση είναι να αντικατασταθεί μόνο μία λύση από τον παλαιό πληθυσμό με μία από το νεότερο. Αυτή η μέθοδος ονομάζεται αντικατάσταση σταθερής κατάστασης. Οι Sastry et al. (2005) ορίζουν διαφορετικά την αντικατάσταση σταθερής κατάστασης· δεν αντικαθιστάται μόνο μία λύση του παλαιότερου πληθυσμού αλλά περισσότερες. Σε κάθε περίπτωση θα πρέπει να οριστεί το κριτήριο με το οποίο γίνεται η αντικατάσταση (χειρότερες λύσεις, τυχαία, γονείς κλπ.)

2.5 ΜΗ ΕΦΙΚΤΟΤΗΤΑ ΛΥΣΕΩΝ

Άλλο ένα σημαντικό στοιχείο στους γενετικούς αλγορίθμους είναι ο τρόπος διαχείρισης των μη εφικτών λύσεων. Μία εφικτή λύση δεν μπορεί να θεωρηθεί σαν λύση, καθώς παραβιάζει τουλάχιστον έναν περιορισμό, πράγμα που την καθιστά αδύνατη στην εκτέλεση της και συνεπώς άχρηστη.

Έστω ένα πρόβλημα προγραμματισμού μηχανών κατά παραγγελία. Είναι δυνατό να προκύψει σαν αποτέλεσμα ένα πρόγραμμα, το οποίο τοποθετεί την εκκίνηση μίας διεργασίας πριν από το τέλος μιας προαπαιτούμενης της. Αυτό συμβαίνει πολύ συχνά, ειδικά σε προβλήματα που έχουν πολλούς περιορισμούς που πρέπει να ικανοποιηθούν. Η ύπαρξη μη εφικτών λύσεων εξαρτάται από το σχεδιασμό των συναρτήσεων διασταύρωσης και μετάλλαξης.

Υπάρχουν τρεις τρόποι για να αντιμετωπιστούν οι μη εφικτές λύσεις:

1. Απόρριψη μη εφικτών λύσεων
2. Ποινικοποίηση μη εφικτών λύσεων
3. Επιδιόρθωση μη εφικτών λύσεων

Ο πιο εύκολος τρόπος διαχείρισης μη εφικτών λύσεων είναι η απόρριψη τους. Όμως η εύρεση μιας λύσης είναι υπολογιστικά ακριβή, οπότε αυτή η μέθοδος δεν προτιμάται. Κάτι ακόμα που πρέπει να ληφθεί υπ' όψιν είναι ότι μη εφικτές λύσεις μπορεί να οδηγήσουν σε καλές εφικτές λύσεις με τις οποίες γειτνιάζουν, επισπεύδοντας τη διαδικασία της αναζήτησης βέλτιστης λύσης. Σε περίπτωση μη αποδοχής των μη εφικτών λύσεων αυτές, οι λύσεις παραμένουν αποκλεισμένες.

Η δεύτερη προσέγγιση είναι η ποινικοποίηση των μη εφικτών λύσεων. Η μη εφικτότητα γίνεται μετρήσιμη και παίρνει μία τιμή που μειώνει την ποιότητα της λύσης, για παράδειγμα χειροτερεύει την καταλληλότητα της λύσης. Πρέπει όμως να βρεθεί μία συνάρτηση ποινής, η οποία να επιτρέπει τις μη εφικτές λύσεις αλλά να οδηγεί την αναζήτηση σε εφικτές λύσεις. Σε μερικές περιπτώσεις είναι δύσκολο να βρεθούν παράμετροι ποινής ώστε η αναζήτηση να καταλήξει σε εφικτή λύση.

Η τρίτη επιλογή είναι η επιδιόρθωση μιας εφικτής λύσης. Αυτό εξαρτάται από το πρόγραμμα και απαιτεί συγκεκριμένες αποφάσεις σχεδιασμού, για παράδειγμα αν θα χρησιμοποιηθεί μία αιτιοκρατική ή στοχαστική συνάρτηση επιδιόρθωσης. Σε μερικές δύσκολες περιπτώσεις η διόρθωση μιας λύσης είναι εξίσου δύσκολη με την κατασκευή μιας εφικτής λύσης. Σε τέτοιες περιπτώσεις η επιδιόρθωση λύσεων είναι υπολογιστικά ασύμφορη.

ΚΕΦΑΛΑΙΟ 3

ΠΕΡΙΓΡΑΦΗ ΠΡΟΓΡΑΜΜΑΤΟΣ

3.1 ΕΙΣΑΓΩΓΗ

Ο στόχος αυτής της εργασίας είναι ο σχεδιασμός ενός προγράμματος που δέχεται τα δεδομένα ενός προβλήματος χρονοπρογραμματισμού μηχανών κατά παραγγελία, τα επεξεργάζεται με την μέθοδο των γενετικών αλγορίθμων και εξάγει σαν αποτέλεσμα ένα βέλτιστο χρονοπρόγραμμα.

Το πρόγραμμα ξεκινά δημιουργώντας έναν αρχικό πληθυσμό από λύσεις. Στη συνέχεια καλεί τη συνάρτηση η οποία περιλαμβάνει το επαναληπτικό τμήμα ενός γενετικού αλγορίθμου. Σε κάθε επανάληψη επιλέγονται λύσεις και αναπαράγονται δημιουργώντας απογόνους. Οι απόγονοι διορθώνονται αν είναι ανέφικτοι και έπειτα μεταλλάσσονται με τοπική αναζήτηση. Τέλος, από τον αρχικό πληθυσμό και τους απογόνους επιλέγεται ο νέος πληθυσμός που θα χρησιμοποιηθεί στην επόμενη επανάληψη. Μετά τον πέρας των επαναλήψεων η λύση, μέσα από τον τελικό πληθυσμό που προέκυψε, με τον μικρότερο συνολικό χρόνο ολοκλήρωσης είναι η βέλτιστη λύση.

3.2 MATLAB

Το πρόγραμμα στο οποίο υλοποιήθηκε ο αλγόριθμος είναι το Matlab. Το Matlab αποτελεί ένα εμπορικό εργαλείο το οποίο προσφέρει ένα διαδραστικό προγραμματιστικό περιβάλλον στον χρήστη και χρησιμοποιείται σε ένα μεγάλο εύρος εφαρμογών. Ενσωματώνει μια υψηλού επιπέδου γλώσσα προγραμματισμού, κατάλληλη για τη μοντελοποίηση και επίλυση σύνθετων μαθηματικών, και όχι μόνο, προβλημάτων.

Το όνομά του προέρχεται από τις λέξεις MATrix LABoratory (εργαστήριο πινάκων) πράγμα που υποδηλώνει το γεγονός ότι η λειτουργία του βασίζεται εξ ολοκλήρου στη χρήση πινάκων, στοιχεία των οποίων μπορεί να είναι πραγματικοί ή μιγαδικοί αριθμοί (ακόμα και ένας μεμονωμένος αριθμός, όπως για παράδειγμα το 8, θεωρείται ως πίνακας με ένα στοιχείο). Ανάμεσα σε ένα πλήθος άλλων ευκολιών που προσφέρει, επιτρέπει τον εύκολο χειρισμό πινάκων, τη γραφική απεικόνιση (*plotting*) συναρτήσεων και δεδομένων, την υλοποίηση αλγορίθμων, την δημιουργία γραφικών περιβαλλόντων διεπαφής χρήστη (*graphic user interface*) και τη συνεργασία και διαλειτουργικότητα με προγράμματα γραμμένα σε άλλες γλώσσες προγραμματισμού.

Λόγω, δε, του ότι το Matlab βρίσκει εφαρμογή σε ποικίλα επιστημονικά πεδία (επεξεργασία σήματος, νευρωνικά δίκτυα, συστήματα ελέγχου κ.τ.λ.), πρόσθετα πακέτα, που ονομάζονται toolboxes, ενσωματώνονται σε αυτό και προσφέρουν χρήσιμες εξειδικευμένες συναρτήσεις.

3.3 ΤΟΠΙΚΗ ΑΝΑΖΗΤΗΣΗ

Η τοπική αναζήτηση (*local search*) βασίζεται στην παλαιότερη μέθοδο βελτιστοποίησης, αυτής της δοκιμής και σφάλματος (Papadimitriou και Steiglitz, 1982). Η ιδέα είναι απλή και εφαρμόσιμη σε μεγάλο εύρος προβλημάτων, για αυτό το λόγο είναι και πολύ δημοφιλής. Έστω ένα πρόβλημα βελτιστοποίησης (F, c), όπου F είναι ένα εφικτό σύνολο και c είναι το κόστος, επιλέγεται η γειτονιά

$$N : F \rightarrow 2^F \tag{3.1}$$

στην οποία εφαρμόζεται αναζήτηση για να βρεθεί ένα σημείο $s \in N$ που να έχει μικρότερο κόστος από το υπάρχον. Η αναζήτηση στην γειτονιά συνεχίζεται επαναληπτικά μέχρι να βρεθεί η καλύτερη λύση του συνόλου (για παράδειγμα αυτή που έχει την καλύτερη τιμή αντικειμενικής συνάρτησης). Η καλύτερη λύσης της γειτονιάς ονομάζεται τοπικό βέλτιστο (*local optimum*).

Ο αλγόριθμος αναζήτησης ξεκινά από μία εφικτή λύση $t \in F$ (η οποία έχει προκύψει από το υπόλοιπο πρόβλημα είτε κατασκευάζεται ευρετικά ή τυχαία) και επαναληπτικά αλλάζει μέρος της λύσης ή ολόκληρη τη λύση με κάτι καλύτερο μέσα από το ορισμένο σύνολο της γειτονιάς. Όταν βρεθεί καλύτερη λύση από την υπάρχουσα διατηρείται και η διαδικασία συνεχίζεται μέχρι να βρεθεί το τοπικό βέλτιστο. Προκειμένου να αντικατασταθούν μέρη της λύσης η τοπική αναζήτηση πραγματοποιεί αναζήτηση με βάση συγκεκριμένες μεθόδους μέσα στη γειτονιά για να ανακαλύψει νέες λύσεις. Οι πιο γνωστές μέθοδοι είναι 2-Opt, 1-1 exchange και 1-0 relocate. Η μέθοδος 2-Opt αντιστρέφει ένα σύνολο εργασιών τυχαίου μήκους σε μία μηχανή ενώ η μέθοδος 1-1 exchange αντιμεταθέτει δύο τυχαίες διεργασίες στην ίδια μηχανή. Η 1-0 relocate μεταφέρει μία διεργασία από τη θέση της σε μία άλλη θέση στην ίδια μηχανή. Οι αλλαγές που μπορούν να γίνουν σε μία λύση, χρησιμοποιώντας τοπική αναζήτηση, και οι γείτονες που προκύπτουν είναι πρακτικά άπειροι.

Το κύριο μειονέκτημα της τοπικής αναζήτησης είναι ότι μπορεί πολύ εύκολα να παγιδευτεί σε τοπικό βέλτιστο, καθώς εξ ορισμού είναι κοντόφθαλμες, αφού εξετάζουν τις κοντινές γειτονικές λύσεις. Αν σχεδιαστεί σωστά ένας αλγόριθμος τοπικής αναζήτησης μπορεί αποτελεσματικά να αναζητήσει την γειτονιά μιας αρχικής λύσης αλλά δεν μπορεί να οδηγηθεί σε άλλες γειτονιές για να βρει το ολικό βέλτιστο του προβλήματος (*global optimum*). Για αυτό το λόγο η τοπική αναζήτηση χρησιμοποιείται κυρίως επικουρικά ώστε να βελτιώσει λύσεις που προκύπτουν από ευρύτερη

αναζήτηση. Σε αυτή την εργασία χρησιμοποιείται ώστε να βελτιώσει λύσεις που προκύπτουν από τον γενετικό αλγόριθμο.

3.4 ΕΙΣΑΓΩΓΗ ΔΕΔΟΜΕΝΩΝ

Το πρόβλημα χρονοπρογραμματισμού μηχανών κατά παραγγελία έχει, στη γενική του μορφή τέσσερα δεδομένα: τον αριθμό των μηχανών, τον αριθμό των εργασιών, τις ακολουθίες των διεργασιών για την κάθε εργασία και τους χρόνους εκτέλεσης της κάθε εργασίας. Τα δύο πρώτα στοιχεία σε μερικές περιπτώσεις είναι πιθανό να μην δίνονται καθώς προκύπτουν από τα υπόλοιπα δύο. Για παράδειγμα, όταν υπάρχει σαν δεδομένο ένας πίνακας ακολουθιών των διεργασιών μεγέθους 4×3 και κάθε γραμμή απεικονίζει την ακολουθία για κάθε εργασία, είναι προφανές ότι υπάρχουν 3 μηχανές και 4 εργασίες.

Στην συγκεκριμένη εργασία τα δεδομένα προκύπτουν από τη βιβλιογραφία. Συγκεκριμένα είναι τα παραδείγματα των Adams et al. (1988), Fisher et al. (1963), Lawrence (1984), Applegate et al. (1991), Storer et al. (1992), Yamada et al. (1992). Είναι προβλήματα τα οποία έχουν χρησιμοποιηθεί από αρκετές εργασίες στο παρελθόν, πράγμα που επιτρέπει την σύγκριση των λύσεων που προκύπτουν, με τις καλύτερες γνωστές που έχουν βρεθεί σε προηγούμενες εργασίες. Στον πίνακα 3.1 δίνονται τα δεδομένα του παραδείγματος της ενότητας 1.4.

Πίνακας 3.1: Δεδομένα προβλήματος παραδείγματος

3	3				
0	12	1	15	2	6
2	7	0	20	1	8
1	8	2	9	0	12

Σε αυτόν τον πίνακα καταγράφονται όλες οι πληροφορίες που χρειάζονται για να οριστεί σαφώς το πρόβλημα. Στην πρώτη γραμμή φαίνονται ο αριθμός των εργασιών, στο πρώτο κελί, και ο αριθμός των μηχανών, στο δεύτερο κελί. Στις επόμενες γραμμές απεικονίζονται οι υπόλοιπες πληροφορίες. Κάθε γραμμή αναφέρεται σε μία εργασία. Στα μονά κελιά (πρώτο, τρίτο κ.ο.κ) περιέχεται ο αριθμός των μηχανών ξεκινώντας από το μηδέν, ενώ στα ζυγά κελιά (δεύτερο, τέταρτο κ.ο.κ) φαίνεται η διάρκεια της διεργασίας του κελιού της διεργασίας που προηγείται. Για παράδειγμα, για την εργασία η ακολουθία που φαίνεται είναι (0, 1, 2), που σημαίνει (M_1, M_2, M_3) και διεργασίες (o_{11}, o_{12}, o_{13}) με διάρκεια αντίστοιχα 12, 15 και 6 χρονικές μονάδες.

Τα δεδομένα για κάθε παράδειγμα αποθηκεύονται, στη μορφή του πίνακα 3.1, σε ένα αρχείο τύπου .xls, ώστε να εισάγονται σαν δεδομένα στο περιβάλλον του Matlab. Στην συνέχεια το πρόγραμμα τα αποθηκεύει σε μεταβλητές: n αριθμός

εργασιών, *m* αριθμός μηχανών, *sequence* ακολουθία διεργασιών και *times* χρόνοι εκτέλεσης των διεργασιών.

3.5 ΚΩΔΙΚΟΠΟΙΗΣΗ ΔΕΔΟΜΕΝΩΝ

Τα δεδομένα που εισάγονται στο πρόγραμμα, καθώς και οι πληροφορίες που προκύπτουν από την επεξεργασία τους, πρέπει να κωδικοποιούνται με συγκεκριμένο τρόπο, ώστε να είναι εύκολη η αποθήκευση και η περαιτέρω επεξεργασία τους. Επιπλέον είναι σημαντικό να είναι εύκολα κατανοητά από το χρήστη, ώστε να μπορεί να παρακολουθήσει την διαδικασία του προγράμματος.

Οι λύσεις του προβλήματος σε αυτήν την εργασία κωδικοποιούνται με δύο τρόπους, κάθε ένας από τους οποίους έχει συγκεκριμένη χρησιμότητα.

- Πίνακας χρόνων εκκίνησης διεργασιών

Ένας τρόπος απεικόνισης μιας λύσης είναι ο πίνακας των χρόνων εκκίνησης των διεργασιών. Είναι ένας πίνακας δύο διαστάσεων που περιέχει τη χρονική στιγμή κατά την οποία εκκινεί η εκτέλεση κάθε διεργασίας. Παράδειγμα της απεικόνισης φαίνεται στον πίνακα 3.2, για μία τυχαία λύση του προβλήματος της ενότητας 1.4. Η κάθε σειρά του πίνακα αντιστοιχεί σε μία εργασία, ενώ κάθε στήλη σε μία μηχανή. Για παράδειγμα, βλέποντας το κελί (1,1), ο χρήστης καταλαβαίνει ότι η εργασία J_1 στην μηχανή M_1 εκκινεί τη χρονική στιγμή 0. Αν σε αυτόν τον πίνακα προσθέσουμε τον πίνακα που περιέχει τους χρόνους εκτέλεσης των εργασιών προκύπτει ο πίνακας με τους χρόνους περάτωσης διεργασιών (πίνακας 3.3). Από αυτόν τον πίνακα φαίνεται πολύ εύκολα ο συνολικός χρόνος εκτέλεσης όλων των εργασιών C_{max} (*makespan*), καθώς είναι το μέγιστο του πίνακα.

Πίνακας 3.2: Χρόνοι εκκίνησης εργασιών τυχαίας λύσης παραδείγματος

0	39	54
12	19	0
21	0	9

Πίνακας 3.3: Χρόνοι ολοκλήρωσης εργασιών λύσης πίνακα 3.2

12	54	60
19	39	8
29	9	21

- Διάνυσμα ακολουθίας εργασιών

Οι γενετικοί αλγόριθμοι πραγματοποιούν διασταυρώσεις γονιδίων με τον τρόπο που περιγράφηκε στην ενότητα 2.4.5. Η διαδικασία αυτή ευκολότερο να πραγματοποιηθεί αν τα γονίδια έχουν μορφή διανύσματος, παρά πίνακα, οπότε χρησιμοποιείται αυτός ο τρόπος κωδικοποίησης. Κάθε λύση καταγράφεται σε ένα διάνυσμα, το οποία περιγράφει την ακολουθία των διεργασιών του συγκεκριμένου προγράμματος.

Αρχικά η κάθε εργασία αποκτά ένα σειριακό αριθμό. Για παράδειγμα, σε ένα πρόβλημα με τρεις εργασίες και τρεις μηχανές η κωδικοποίηση των εργασιών γίνεται ως εξής:

Πίνακας 3.4: Σειριακοί αριθμοί εργασιών

o_{11}		1
o_{12}		2
o_{13}		3
o_{21}		4
o_{22}	⇨	5
o_{23}		6
o_{31}		7
o_{32}		8
o_{33}		9

Αφού έχουν κωδικοποιηθεί όλες οι διεργασίες, έτσι ώστε κάθε μία να απεικονίζεται από έναν ακέραιο αριθμό, μπορεί να δημιουργηθεί το διάνυσμα που απεικονίζει τη λύση. Στην πρώτη θέση του διανύσματος θα εισαχθεί η εργασία που εκκινεί τη χρονική στιγμή μηδέν, στη δεύτερη αυτή που ξεκινά αμέσως μετά και ούτω καθεξής. Σε περίπτωση που δύο ή περισσότερες εργασίες ξεκινούν ταυτόχρονα, βρίσκονται σε διπλανές θέσεις, αλλά η μεταξύ τους σειρά τους είναι αδιάφορη.

Πίνακας 3.5: Διάνυσμα ακολουθίας εργασιών

1	8	6	2	9	4	3	7	5

3.6 ΠΕΡΙΓΡΑΦΗ ΠΡΟΓΡΑΜΜΑΤΟΣ

Σε αυτήν την ενότητα θα περιγραφεί το πρόγραμμα που γράφτηκε στο Matlab σε ψευδοκώδικα. Οι διαδικασίες που χρησιμοποιούνται αρκετές φορές έχουν καταγραφεί σε συναρτήσεις ώστε να είναι γρηγορότερη η εκτέλεση του προγράμματος, αλλά και πιο κατανοητή στην ανάγνωση. Κάθε συνάρτηση θα καταγραφεί ξεχωριστά.

Αλγόριθμος 3.1: Κύριο μέρος Προγράμματος
Διάβασε δεδομένα *n, m, sequence, times* **Για** *counter* **από** 1 **έως** 10 Δημιούργησε τυχαία εφικτή λύση Πρόσθεσε τη λύση στον πληθυσμό *population* **Τέλος επανάληψης** **Για** *counter* **από** 1 **έως** 1000 *population* ← genetic_algorithm(*population*) **Τέλος επανάληψης** Αξιολόγησε *population* *solution* ← Αναζήτησε την καλύτερη λύση από *population* Εκτύπωση *solution*

Αρχικά δημιουργείται ένας πληθυσμός 10 τυχαίων εφικτών λύσεων. Το μέγεθος του πληθυσμού είναι αρκετά μεγάλο, ώστε να υπάρχει κάποια ποικιλία λύσεων, και αρκετά μικρό, ώστε να μην γίνεται μεγάλος ο υπολογιστικός φόρτος. Κατά την αρχικοποίηση των λύσεων λαμβάνονται υπ' όψιν όλοι οι περιορισμοί, με αποτέλεσμα να παράγονται μόνο εφικτές λύσεις. Στη συνέχεια καλείται η επαναληπτική διαδικασία του γενετικού αλγορίθμου, που περιλαμβάνεται στη συνάρτηση genetic_algorithm. Αφού τελειώσει αυτή η διαδικασία έχει προκύψει ο τελικός πληθυσμός. Από αυτόν τον πληθυσμό επιλέγεται η καλύτερη λύση, η οποία είναι και το τελικό χρονοπρόγραμμα που δίνει σαν αποτέλεσμα το πρόγραμμα. Σαν κριτήριο επιλογής των λύσεων θεωρείται ο συνολικός χρόνος διάρκειας των εργασιών (*makespan*).

Αλγόριθμος 3.2: Συνάρτηση genetic_algorithm
Για *counter* **από** 1 **έως** 5 *offsprings* ← Δημιούργησε 2 παιδιά συνδυάζοντας 2 άτομα του πληθυσμού **Αν** *offsprings* μη εφικτές λύσεις **τότε** Διόρθωσε *offsprings* **Τέλος Αν** *offsprings* ← local_search(*offsprings*) Πρόσθεσε τα παιδιά *offsprings* στον πληθυσμό *population* **Τέλος επανάληψης** Αξιολόγησε *population* *population* ← Διατήρησε τις 10 καλύτερες λύσεις του *population*

Η συνάρτηση genetic_algorithm παίρνει σαν όρισμα τον πληθυσμό που έχει προκύψει, αρχικά από την αρχικοποίηση του πληθυσμού και στη συνέχεια από την προηγούμενη επανάληψη, για να δημιουργήσει απογόνους. Από τα δέκα άτομα του πληθυσμού επιλέγονται όλα, ανά δύο, και δημιουργούν συνολικά δέκα απογόνους. Με αυτή τη μέθοδο δεν λαμβάνεται υπ' όψιν η καταλληλότητα των λύσεων, όμως δεν αγνοούνται οι κακές λύσεις, που είναι πιθανό να οδηγήσουν σε καλύτερες.

Λόγω τις ύπαρξης πολλών περιορισμών στο πρόβλημα job shop, η πιθανότητα ένας απόγονος να είναι μη εφικτός είναι αρκετά μεγάλη. Στους απόγονους που

δημιουργούνται κατά τη διαδικασία της διασταύρωσης είναι πιθανό μία διεργασία να εμφανίζεται δύο φορές, επειδή, για παράδειγμα, στον πρώτο γονέα βρισκόταν πριν το σημείο διασταύρωσης και στον δεύτερο γονέα μετά από το σημείο. Επίσης είναι πιθανό να παραβιάζεται η ακολουθία των διεργασιών κάποια μηχανής επειδή κατά την διαδικασία της διασταύρωσης κάποια διεργασία τοποθετήθηκε πριν από την προηγούμενη της. Για αυτό το λόγο για κάθε λύση που προκύπτει από τη διασταύρωση ελέγχονται οι διεργασίες που μπορεί να εμφανίζονται δύο φόρες στο διάνυσμα λύσης, καθώς και οι περιορισμοί ακολουθίας. Σε περίπτωση μη εφικτότητας οι απόγονοι διορθώνονται και καθίστανται εφικτοί.

Αναφέρθηκε σε προηγούμενη ενότητα, ότι αν το κόστος διόρθωσης της λύσης είναι μεγάλο, τότε είναι καλύτερα να απορριφθεί η λύση και να δημιουργηθεί μία νέα. Σε αυτή την περίπτωση παρόλο που το κόστος διόρθωσης είναι μεγάλο, η απόρριψη μίας λύσης δεν εξυπηρετεί το σκοπό της αναζήτησης, καθώς μία νέα αρχική λύση είναι τις περισσότερες φορές αρκετά χειρότερη και ουσιαστικά είναι ένα βήμα προς τα πίσω, προς τον αρχικό πληθυσμό. Με δεδομένο ότι το συντριπτικό ποσοστό των λύσεων που παράγεται, κυρίως στα πρώτα βήματα, είναι μη εφικτές, η μη αποδοχή αυτών των λύσεων θα οδηγούσε σε ένα απλούστερο και γρηγορότερο πρόγραμμα, που όμως θα έδινε χειρότερες λύσεις.

Μετά τον έλεγχο εφικτότητας οι απόγονοι περνάνε στο στάδιο της μετάλλαξης. Η μετάλλαξη πραγματοποιείται σε όλες τις λύσεις με την συνάρτηση local_search. Αφού βελτιωθεί η λύση εισάγεται στο πληθυσμό. Όταν όλοι οι απόγονοι εισέλθουν στον πληθυσμό γίνεται η διαδικασία της αντικατάστασης. Τα είκοσι άτομα που βρίσκονται στον πληθυσμό αξιολογούνται, με βάση το συνολικό χρόνο εκτέλεσης (*makespan*). Από αυτά τα δέκα καλύτερα (οι λύσεις με τον μικρότερο χρόνο) επιβιώνουν και περνούν στην επόμενη γενιά, δηλαδή στον πληθυσμό που θα χρησιμοποιηθεί στην επόμενη επανάληψη, ενώ οι υπόλοιπες απορρίπτονται.

Αλγόριθμος 3.3: Συνάρτηση local_search

Είσοδος (*individual*)
temp←*individual*
Για *local_search_counter* από 1 έως 200
    Επίλεξε τυχαία μηχανή *machine*
    Επίλεξε τυχαίες εργασίες *job1, job2*
    s1,s2←Υπολόγισε τους σειριακούς των διεργασιών (*machine, job1*), (*machine, job2*)
    temp←Αντιμετάθεσε *s1, s2* στη λύση *temp*
    Αξιολόγησε *temp*
    Αν *makespan*(*individual*)>*makespan*(*temp*) **τότε**
        individual←*temp*
    Αλλιώς Αν *makespan*(*individual*)>>*makespan*(*temp*)
        temp←*individual*
    Τέλος Αν
Τέλος επανάληψης

Η συνάρτηση local_search δέχεται σαν είσοδο ένα άτομο του πληθυσμού και χρησιμοποιεί τοπική αναζήτηση 1-1 exchange για να τον βελτιώσει. Αρχικά επιλέγει μία τυχαία μηχανή και δύο τυχαίες εργασίες. Έπειτα υπολογίζει τους σειριακούς αριθμούς των διεργασιών, τις εντοπίζει στο διάνυσμα λύσης και τις αντιμεταθέτει. Αφού προκύπτει μία νέα λύση αξιολογείται με βάση το makespan. Αν είναι καλύτερη της αρχικής τότε διατηρείται, ενώ αν είναι πολύ χειρότερη της αρχικής τότε απορρίπτεται. Σε περίπτωση που είναι χειρότερη της αρχικής αλλά η απόκλιση από την αρχική λύση είναι μικρή, για παράδειγμα μικρότερη του 10%, τότε η λύση διατηρείται και συνεχίζεται η τοπική αναζήτηση σε αυτήν μέχρι να γίνει καλύτερη της αρχικής και να την αντικαταστήσει ή αρκετά χειρότερη ώστε να υπερβεί το κατώφλι και να απορριφθεί. Με αυτό τον τρόπο περιορίζονται οι πιθανότητες να εγκλωβιστεί η τοπική αναζήτηση σε τοπικό βέλτιστο.

ΚΕΦΑΛΑΙΟ 4

ΑΠΟΤΕΛΕΣΜΑΤΑ ΠΡΟΓΡΑΜΜΑΤΟΣ

4.1 ΕΙΣΑΓΩΓΗ

Σε αυτό το κεφάλαιο θα παρουσιαστούν τα αποτελέσματα του προγράμματος. Για να εκτιμηθεί η ποιότητα των λύσεων θα χρησιμοποιηθούν τα εξής παραδείγματα τα οποία βρέθηκαν στην επιστημονική βιβλιογραφία:

- ABZ5 - ABZ9 που προτάθηκαν από τους Adams et al. (1988)
- FT6, FT10 ΚΑΙ FT20 που προτάθηκαν από τους Fisher και Thompson (1963)
- LA01 - LA40 που προτάθηκαν από τον Lawrence (1984)
- ORB01 - ORB10 που προτάθηκαν από τους Applegate και Cook (1991)
- SWV01 - SWV20 που προτάθηκαν από τους Storer et al. (1992)
- YN1 - YN4 που προτάθηκαν από τους Yamada και Nakano (1992)

Οι πίνακες 4.1, 4.2, 4.3, 4.4, 4.5 και 4.6 συνοψίζουν τα αποτελέσματα του προγράμματος. Σε κάθε πίνακα αναγράφεται το όνομα του παραδείγματος, το μέγεθος του προβλήματος (εργασίες × μηχανές), η καλύτερη γνωστή λύση (*best known solution, BKS*), η καλύτερη λύση που προέκυψε από το πρόγραμμα, μετά από 10 εκτελέσεις, η σχετική απόκλιση (*relative deviation, RD*) από την καλύτερη γνωστή, η σχετική απόκλιση του μέσου όρου των αποτελεσμάτων που προέκυψαν από τις 10 εκτελέσεις (RD_{av}) για κάθε παράδειγμα και την διαφορά των δύο τελευταίων. Στην τελευταία γραμμή κάθε πίνακα υπολογίζεται η μέση σχετική απόκλιση (*average relative deviation, ARD*) για τις καλύτερες λύσεις και η μέση σχετική απόκλιση για όλες τις λύσεις (ARD_{av}) για κάθε ομάδα δεδομένων. Οι αποκλίσεις υπολογίζονται με τους εξής τύπους:

$$RD = \frac{C_{max} - BKS}{BKS} \tag{4.1}$$

$$SD = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}} \tag{4.2}$$

$$ARD = \frac{\sum_{i=1}^{NOI} RD}{NOI} \tag{4.3}$$

όπου n είναι ο αριθμός παραδειγμάτων (δηλαδή 10), x_i η κάθε λύση, $\bar{x}$ ο μέσος όρος των λύσεων και NOI ο αριθμός των παραδειγμάτων (*number of instances*) για κάθε ομάδα.

Οι παράμετροι του προγράμματος είναι:

- Επαναλήψεις γενετικού αλγορίθμου: 1000
- Μέγεθος Πληθυσμού: 10
- Πλήθος απογόνων που προκύπτουν από αναπαραγωγή σε κάθε επανάληψη: 10
- Επαναλήψεις τοπικής αναζήτησης: 200
- Κατώφλι αποδοχής λύσης τοπικής αναζήτησης: 10%
- Επαναλήψεις προγράμματος (από τις οποίες διατηρήθηκε η καλύτερη λύση): 10

Όλες οι παράμετροι επιλέχθηκαν έτσι ώστε να είναι αρκετά μεγάλες, ώστε να συγκλίνει η διαδικασία, αλλά όχι τόσο μεγάλες ώστε να αυξάνουν υπέρμετρα τον υπολογιστικό φόρτο. Μετά από δοκιμή και σφάλμα προέκυψαν οι παραπάνω τιμές.

4.2 ΠΙΝΑΚΕΣ ΑΠΟΤΕΛΕΣΜΑΤΩΝ

Πίνακας 4.1: Αποτελέσματα προγράμματος για παραδείγματα ABZ5-ABZ9

Παράδειγμα	**Μέγεθος**	**BKS**	**Λύση**	**RD**	**RD_{av}**	**Διαφορά**
ABZ5	10×10	1234	1252	1,46%	3,06%	1,60%
ABZ6	10×10	943	948	0,53%	2,07%	1,54%
ABZ7	20×15	656	968	47,56%	51,13%	3,57%
ABZ8	20×15	665, 646	959	44,21%	55,98%	11,77%
ABZ9	20×15	679, 662	922	35,79%	41,90%	6,11%
			ARD	25,91%	30,83%	4,92%

Πίνακας 4.2: Αποτελέσματα προγράμματος για παραδείγματα FT6, FT10, FT20

Παράδειγμα	**Μέγεθος**	**BKS**	**Λύση**	**RD**	**RD_{av}**	**Διαφορά**
FT06	6×6	55	55	0,00%	0,00%	0,00%
FT10	10×10	930	976	4,95%	8,28%	3,33%
FT20	20×5	1165	1340	15,02%	18,38%	3,36%
			ARD	6,66%	8,89%	2,23%

Πίνακας 4.3: Αποτελέσματα προγράμματος για παραδείγματα LA01-LA40

Παράδειγμα	**Μέγεθος**	**BKS**	**Λύση**	**RD**	**RD_{av}**	**Διαφορά**
LA01	10×5	666	795	19,37%	19,73%	0,36%
LA02	10×5	655	713	8,85%	11,62%	2,76%
LA03	10×5	597	755	26,47%	27,82%	1,36%
LA04	10×5	590	704	19,32%	23,69%	4,37%
LA05	10×5	593	643	8,43%	8,58%	0,15%
LA06	15×5	926	1036	11,88%	18,37%	6,49%

LA07	15×5	890	989	11,12%	13,11%	1,99%
LA08	15×5	863	1019	18,08%	20,19%	2,11%
LA09	15×5	951	1071	12,62%	14,14%	1,52%
LA10	15×5	958	1066	11,27%	12,03%	0,75%
LA11	20×5	1222	1418	16,04%	18,76%	2,73%
LA12	20×5	1039	1194	14,92%	16,96%	2,04%
LA13	20×5	1150	1278	11,13%	15,39%	4,26%
LA14	20×5	1292	1467	13,54%	16,15%	2,60%
LA15	20×5	1207	1434	18,81%	23,37%	4,57%
LA16	10×10	945	956	1,16%	3,83%	2,67%
LA17	10×10	784	784	0,00%	1,02%	1,02%
LA18	10×10	848	859	1,30%	2,10%	0,80%
LA19	10×10	842	866	2,85%	4,76%	1,91%
LA20	10×10	902	907	0,55%	2,96%	2,41%
LA21	15×10	1046	1659	58,60%	66,24%	7,64%
LA22	15×10	927	1576	70,01%	75,81%	5,80%
LA23	15×10	1032	1652	60,08%	66,15%	6,08%
LA24	15×10	935	1613	72,51%	76,36%	3,85%
LA25	15×10	977	1444	47,80%	59,62%	11,82%
LA26	20×10	1218	1543	26,68%	32,56%	5,88%
LA27	20×10	1235	1684	36,36%	40,57%	4,22%
LA28	20×10	1216	1589	30,67%	34,93%	4,26%
LA29	20×10	1153, 1142	1473	27,75%	32,72%	4,97%
LA30	20×10	1355	1644	21,33%	26,88%	5,55%
LA31	30×10	1784	2410	35,09%	39,64%	4,55%
LA32	30×10	1850	2489	34,54%	44,47%	9,93%
LA33	30×10	1719	2295	33,51%	37,41%	3,90%
LA34	30×10	1721	2284	32,71%	39,05%	6,34%
LA35	30×10	1888	2410	27,65%	31,64%	3,99%
LA36	15×15	1268	1337	5,44%	9,00%	3,56%
LA37	15×15	1397	1496	7,09%	11,12%	4,03%
LA38	15×15	1196	1314	9,87%	13,63%	3,76%
LA39	15×15	1233	1343	8,92%	12,85%	3,93%
LA40	15×15	1222	1297	6,14%	9,11%	2,97%
			ARD	22,01%	25,86%	3,85%

Πίνακας 4.4: Αποτελέσματα προγράμματος για παραδείγματα ORB01-ORB05

Παράδειγμα	**Μέγεθος**	**BKS**	**Λύση**	**RD**	**RD_{av}**	**Διαφορά**
ORB01	10×10	1059	1128	6,52%	9,62%	3,11%
ORB02	10×10	888	895	0,79%	3,92%	3,13%
ORB03	10×10	1005	1125	11,94%	15,95%	4,01%
ORB04	10×10	1005	1012	0,70%	5,23%	4,54%
ORB05	10×10	887	907	2,25%	5,56%	3,30%
			ARD	4,44%	8,06%	3,62%

Πίνακας 4.5: Αποτελέσματα προγράμματος για παραδείγματα SWV01-SWV20

Παράδειγμα	**Μέγεθος**	**BKS**	**Λύση**	**RD**	**RD_{av}**	**Διαφορά**
SWV01	20×10	1418, 1392	1726	21,72%	24,56%	2,83%
SWV02	20×10	1475	1664	12,81%	16,77%	3,96%
SWV03	20×10	1398, 1370	1663	18,96%	23,83%	4,87%
SWV04	20×10	1483, 1450	1726	16,39%	19,57%	3,18%
SWV05	20×10	1434, 1421	1638	14,23%	18,38%	4,16%
SWV06	20×15	1696, 1591	2294	35,26%	38,42%	3,16%
SWV07	20×15	1620, 1447	2386	47,28%	54,97%	7,69%
SWV08	20×15	1763, 1641	2695	52,86%	65,27%	12,40%
SWV09	20×15	1663, 1605	2445	47,02%	59,74%	12,71%
SWV10	20×15	1767, 1632	2538	43,63%	58,82%	15,19%
SWV11	50×10	3005, 2983	3740	24,46%	30,01%	5,55%
SWV12	50×10	3038, 2972	3857	26,96%	29,98%	3,02%
SWV13	50×10	3146, 3104	3926	24,79%	27,88%	3,08%
SWV14	50×10	2968	3698	24,60%	28,54%	3,95%
SWV15	50×10	2940, 2885	3711	26,22%	31,35%	5,12%
SWV16	50×10	2924	4174	42,75%	51,53%	8,78%
SWV17	50×10	2794	3928	40,59%	50,75%	10,16%
SWV18	50×10	2852	4051	42,04%	50,25%	8,21%
SWV19	50×10	2843	4223	48,54%	59,13%	10,59%
SWV20	50×10	2823	3896	38,01%	49,13%	11,12%
			ARD	32,46%	39,44%	6,99%

Πίνακας 4.6: Αποτελέσματα προγράμματος για παραδείγματα YN1-YN4

Παράδειγμα	**Μέγεθος**	**BKS**	**Λύση**	**RD**	**RD_{av}**	**Διαφορά**
YN1	20×20	888, 827	1001	12,73%	16,54%	3,82%
YN2	20×20	909, 862	1013	11,44%	15,48%	4,04%
YN3	20×20	893, 828	996	11,53%	19,07%	7,54%
YN4	20×20	968, 919	1114	15,08%	19,06%	3,98%
			ARD	12,70%	17,54%	4,84%

4.3 ΣΥΜΠΕΡΑΣΜΑΤΑ

Το πρόγραμμα επεξεργάστηκε 77 παραδείγματα του προβλήματος χρονοπρογραματισμού κατά παραγγελία, που βρέθηκαν από την επιστημονική βιβλιογραφία. Τα αποτελέσματα συνοψίζονται στους πίνακες της ενότητας 4.2. Τα μεγέθη των προβλημάτων παρουσιάζουν μεγάλη ποικιλία, με το μικρότερο να έχει μέγεθος 6×6, δηλαδή 6 εργασίες που εκτελούνται σε 6 μηχανές (36 διεργασίες), και το μεγαλύτερο 50×10, δηλαδή 50 εργασίες που εκτελούνται σε 10 μηχανές (500 διεργασίες).

Η μέση σχετική απόκλιση για όλα τα παραδείγματα είναι 20,74%. Αντίστοιχα για κάθε ομάδα παραδειγμάτων ήταν 25,91% για την ABZ, 6,66% για την FT, 22,01% για την LA, 4,44% για την ORB, 32,46% για την SWV και 12,70% για την YN. Καλύτερη απόδοση είχε στην ομάδα ORB, της οποίας τα παραδείγματα είχαν μέγεθος 50×10. Βρέθηκαν βέλτιστες λύσεις σε δύο παραδείγματα LA17 και FT06. Το πρόγραμμα της βέλτιστης λύσης του FT06 παρουσιάζεται στον πίνακα 4.7 και σε διάγραμμα Gantt στο σχήμα 4.1.

Γενικά παρατηρείται ότι ο αλγόριθμος είναι πιο αποδοτικός στα μικρότερα παραδείγματα. Για προβλήματα που περιλαμβάνουν μέχρι 10 εργασίες οι σχετικές αποκλίσεις έχουν χαμηλές τιμές, ενώ σε μεγαλύτερα παραδείγματα που φτάνουν τις 50 εργασίες οι λύσεις που εξάγει ο αλγόριθμος έχουν μεγάλη σχετική απόκλιση. Αντιθέτως, δεν επηρεάζεται από τον αριθμό των μηχανών που έχει το πρόβλημα.

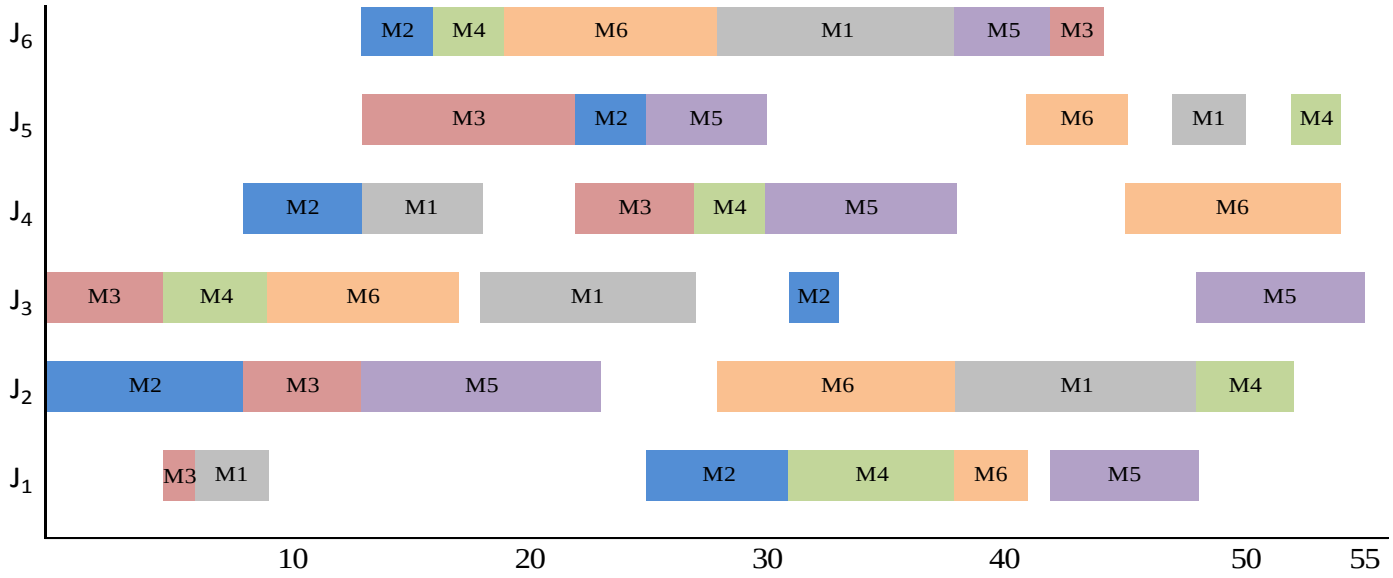
Το ίδιο ισχύει σε μικρότερο βαθμό και για τη μέση τιμή των λύσεων που προκύπτουν για κάθε παράδειγμα. Στα μικρότερα παραδείγματα ο μέσος όρος των λύσεων που προκύπτουν έχει μικρή διαφορά από την καλύτερη του συνόλου λύσεων, πράγμα που σημαίνει ότι ο αλγόριθμος παράγει λύσεις με μικρή απόκλιση μεταξύ τους. Αυτό είναι ένα καλό στοιχείο επειδή αν το πρόγραμμα αποτύχει να δώσει την καλύτερη λύση, τότε θα δώσει μία λύση αρκετά κοντά· δεν είναι αναγκαίο να τρέξει πολλές φορές για να δώσει μία καλή λύση. Όσο μεγαλώνουν τα αποτελέσματα η διαφορά μεγαλώνει

αλλά δεν φτάνει σε ποσοστό μεγαλύτερο του 15%, πράγμα που σημαίνει ότι το πρόγραμμα παράγει λύσεις με συνέπεια.

Πίνακας 4.7: Χρόνοι εκκίνησης και ολοκλήρωσης εργασιών του βέλτιστου προγράμματος του παραδείγματος FT06

Χρόνοι εκκίνησης					
6	25	5	31	42	38
38	0	8	48	13	28
18	31	0	5	48	9
13	8	22	27	30	45
48	22	13	52	25	41
28	13	42	16	38	19

Χρόνοι ολοκλήρωσης					
9	31	6	38	48	41
48	8	13	52	23	38
27	32	5	9	55	17
18	13	27	30	38	54
51	25	22	53	30	45
38	16	43	19	42	28



Σχήμα 4.1: Διάγραμμα Gantt της βέλτιστης λύσης του παραδείγματος FT06

ΒΙΒΛΙΟΓΡΑΦΙΑ

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

Applegate D., Cook W. (1991), A computational study of the job-shop scheduling instance, ORSA Journal on Computing 3, 149-156.

Baker K. R. (1974), Introduction to Sequencing and Scheduling, Wiley, New York.

Blazewicz J., Ecker K., Schimdt G., Weglarz J. (1996), Scheduling in Computer and Manufacturing Systems, Springer, Berlin, New York.

Blazewicz J., Pesch E., and Sterna M. (2000), The disjunctive graph machine representation of the job shop scheduling problem, European Journal of Operational Research 127(2):317-331.

Blum C. and Roli A. (2003), Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Computing Surveys, 35(3):268–308.

Clark W. and Gantt H. (1922), The Gantt chart, a working tool of management. New York, Ronald Press.

De Jong K.A. (1975), An Analysis of the behavior of a class of genetic adaptive systems (Doctoral dissertation, University of Michigan) Dissertation Abstracts International 36(10), 5140B University Microfilms No. 76/938.

Fisher H., Thompson G.L. (1963), Probabilistic learning combinations of local job-shop.

Lawrence S. (1984), Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (Supplement), Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania.

A. S. Manne A.S. (1960), On the job shop scheduling problem. *Operations Research*, 8:219–223.

Μαρινάκης Ι., Μυγδαλάς Α., (2008), Σχεδιασμός και Βελτιστοποίηση της Εφοδιαστικής Αλυσίδας, Εκδόσεις Σοφία, Θεσσαλονίκη.

Μαρινάκης Ι., Μαρινάκη Μ., Ματσατσίνης Ν., Ζοπουνίδης Κ. (2011), Μεθευρετικοί και Εξελικτικοί Αλγόριθμοι σε Προβλήματα Διοικητικής Επιστήμης, Εκδόσεις Κλειδάριθμος.

Michalewicz Z. and Fogel D. B.(2004), How to Solve It: Modern Heuristics. Springer-Verlag.

Papadimitriou C.H., Steiglitz K. (1982), Combinatorial Optimization: Algorithms and Complexity. Prentice-Hall 1982, ISBN 0-13-152462-3.

Porter, L. W., & Lawler, E. E. (1968), Managerial Attitudes and Performance. Homewood, IL: Richard D. Irwin, Inc.

Reeves C. (2003), Genetic algorithms. In F. Glover and G. Kochenberger, editors, Handbook of Metaheuristics, chapter 3, pages 55–82. Kluwer Academic Publishers.

Roy B. and Sussmann B. (1964). Les Probl`emes d'ordonnancement avec contraintes disjonctives, Note DS no. 9 bis, SEMA, Montrouge.

Sastry K., Goldberg D., and Kendall G. (2005), Genetic algorithms. In Edmund K. Burke and Graham Kendall, editors, Search Methodologies - Introductory Tutorials in Optimization and Decision Support Techniques, chapter 4, pages 97–125. Springer Science + Business Media Inc.

Storer R.H., Wu S.D., Vaccari R. (1992), New search spaces for sequencing instances with application to job shop scheduling, Management Science 38, 1495-1509.

Xhafa F., Abraham A. (2008): Metaheuristics for Scheduling in Industrial and Manufacturing Applications. Studies in Computational Intelligence 128, Springer, ISBN 978-3-540-78984-0.

Yamada T., Nakano R. (1992), A genetic algorithm applicable to large-scale job-shop instances.

Zapfel G., Braune R., and Bogl M. (2010), Metaheuristic Search Concepts: A Tutorial with Applications to Production and Logistics, Springer, Berlin, Germany.

K.A. De Jong, An Analysis of the behavior of a class of genetic adaptive systems (Doctoral dissertation, University of Michigan) Dissertation Abstracts International 36(10), 5140B University Microfilms No. 76/938.