ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

A Novel Genetic Algorithm for I/O Pad Planning Retaining Former Cell Positions.

ΚΥΡΙΑΚΟΣ ΠΑΝΑΡΕΤΟΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

Αντώνιος Ν. Δαδαλιάρης
Επίκουρος Καθηγητής

Λαμία, Μάρτιος 2024

«Με ατομική μου ευθύνη και γνωρίζοντας τις κυρώσεις [(1)], που προβλέπονται από της διατάξεις της παρ. 6 του άρθρου 22 του Ν. 1599/1986, δηλώνω ότι:

1. Δεν παραθέτω κομμάτια βιβλίων ή άρθρων ή εργασιών άλλων αυτολεξεί χωρίς να τα περικλείω σε εισαγωγικά και χωρίς να αναφέρω το συγγραφέα, τη χρονολογία, τη σελίδα. Η αυτολεξεί παράθεση χωρίς εισαγωγικά χωρίς αναφορά στην πηγή, είναι λογοκλοπή. Πέραν της αυτολεξεί παράθεσης, λογοκλοπή θεωρείται και η παράφραση εδαφίων από έργα άλλων, συμπεριλαμβανομένων και έργων συμφοιτητών μου, καθώς και η παράθεση στοιχείων που άλλοι συνέλεξαν ή επεξεργάσθηκαν, χωρίς αναφορά στην πηγή. Αναφέρω πάντοτε με πληρότητα την πηγή κάτω από τον πίνακα ή σχέδιο, όπως στα παραθέματα.

2. Δέχομαι ότι η αυτολεξεί παράθεση χωρίς εισαγωγικά, ακόμα κι αν συνοδεύεται από αναφορά στην πηγή σε κάποιο άλλο σημείο του κειμένου ή στο τέλος του, είναι αντιγραφή. Η αναφορά στην πηγή στο τέλος π.χ. μιας παραγράφου ή μιας σελίδας, δεν δικαιολογεί συρραφή εδαφίων έργου άλλου συγγραφέα, έστω και παραφρασμένων, και παρουσίασή τους ως δική μου εργασία.

3. Δέχομαι ότι υπάρχει επίσης περιορισμός στο μέγεθος και στη συχνότητα των παραθεμάτων που μπορώ να εντάξω στην εργασία μου εντός εισαγωγικών. Κάθε μεγάλο παράθεμα (π.χ. σε πίνακα ή πλαίσιο, κλπ), προϋποθέτει ειδικές ρυθμίσεις, και όταν δημοσιεύεται προϋποθέτει την άδεια του συγγραφέα ή του εκδότη. Το ίδιο και οι πίνακες και τα σχέδια

4. Δέχομαι όλες τις συνέπειες σε περίπτωση λογοκλοπής ή αντιγραφής.

Ημερομηνία: 4/2/2024

Ο – Η Δηλ.
Κυριάκος Πανάρετος

(1) «Όποιος εν γνώσει του δηλώνει ψευδή γεγονότα ή αρνείται ή αποκρύπτει τα αληθινά με έγγραφη υπεύθυνη δήλωση
του άρθρου 8 παρ. 4 Ν. 1599/1986 τιμωρείται με φυλάκιση τουλάχιστον τριών μηνών. Εάν ο υπαίτιος αυτών των πράξεων
σκόπευε να προσπορίσει στον εαυτόν του ή σε άλλον περιουσιακό όφελος βλάπτοντας τρίτον ή σκόπευε να βλάψει άλλον, τιμωρείται με κάθειρξη μέχρι 10 έτη

ΠΕΡΙΕΧΟΜΕΝΑ

Περίληψη

Η ελαχιστοποίηση του μήκους του καλωδίου σε ένα **chip** αποτελεί ένα από τα πιο σημαντικά προβλήματα που αντιμετωπίζει ένας σχεδιαστής κατά την Φυσική Σχεδίαση. Οι σχεδιαστές εστιάζουν στην αλλαγή των θέσεων των κελίων για να επιτύχουν αυτό τον στόχο. Παρόλα αυτά, στην παρούσα εργασία θα επιχειρηθεί η αλλαγή των θέσεων των **I/O Pads,** ώστε να επιτευχθεί η ελαχιστοποίηση του συνολικού μήκους καλωδίου. Ο τρόπος πραγματοποίησης αυτού του εγχειρήματος είναι η χρήση ενός γενετικού αλγορίθμου (η ονομασία αντλείται από τη Δαρβίνεια θεωρία της εξέλιξης, πιο συγκεκριμένα από την επιβίωση του δυνατότερου στη φύση).

Η δομή της εργασίας είναι η ακόλουθη: Στο 1ο κεφάλαιο παρουσιάζεται η διαδικασία της σχεδίασης ολοκληρωμένων κυκλωμάτων πολύ μεγάλης κλίμακας (**Very Large Scale of Integration-VLSI**), με ιδιαίτερη έμφαση στο κομμάτι της Φυσικής Σχεδίασης, διότι σε εκείνο το σημείο πραγματοποιούνται και οι αλλαγές των θέσεων των **I/O Pads**. Το 2ο κεφάλαιο περιλαμβάνει το θεωρητικό υπόβαθρο του γενετικού αλγόριθμου με αναλυτική περιγραφή του κάθε βήματος σε ξεχωριστή υπό-ενότητα.

Στο επόμενο κεφάλαιο, ακολουθείται ανάλογη μεθοδολογία με το 2ο, με τη διαφορά ότι παρουσιάζεται η υλοποίηση του αλγορίθμου με χρήση της γλώσσας **Python**. Ο παραγόμενος κώδικας έχει εστιαστεί στην ορθή λειτουργικότητα του αλγορίθμου και τη γρήγορη παραγωγή ποιοτικών αποτελεσμάτων. Η πολυπλοκότητα του αλγόριθμου και του όγκου των **dataset** (για την εκτέλεση του προγράμματος απαιτείται κατά μέσο όρο μισή ημέρα) αποτέλεσε την αφορμή για να προστεθεί στην εργασία το κομμάτι του **multiprocessing**, το οποίο σε ορισμένες εκτελέσεις πέτυχε μείωση του χρόνου της εκτέλεσης κατά 50%.

Πίνακες

Εικόνες

Κώδικες

Σχεδίαση VLSI

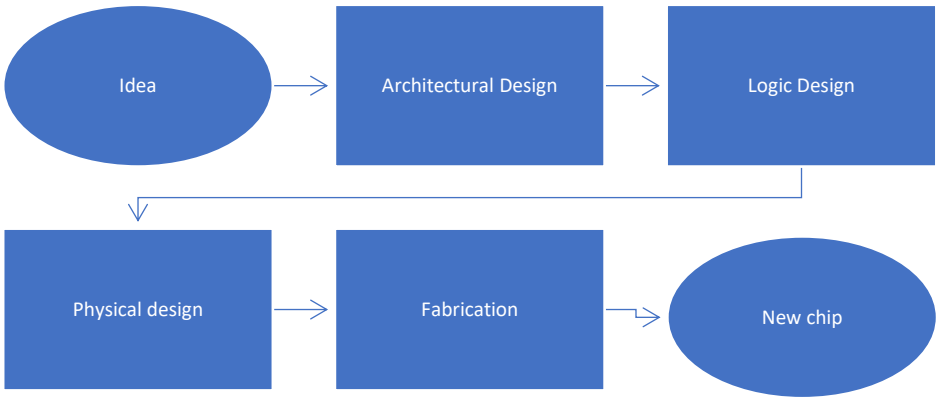
Εισαγωγή

Η σχεδίαση **VLSI** είναι η διαδικασία δημιουργίας ενός ολοκληρωμένου κυκλώματος (**Integrated Circuit-IC**) συνδυάζοντας χιλιάδες τρανζίστορ σε ένα **chip**. Τα πρώτα **Project** ξεκίνησαν τη δεκαετία του 1970 όταν και άρχισαν να αναπτύσσονται οι τηλεπικοινωνίες και το υλικό των υπολογιστών. Οι σημερινοί επεξεργαστές των υπολογιστών που χρησιμοποιούμε αποτελούν ένα παράδειγμα **VLSI** σχεδίασης. Πριν την εισαγωγή της τεχνολογίας **VLSI**, τα περισσότερα **IC** είχαν ένα περιορισμένο εύρος δυνατότητων. Ένα ολοκληρωμένο ηλεκτρονικό κύκλωμα μπορούσε αποτελούνταν από μία Κεντρική Μονάδα Επεξεργασίας (**Central Processing Unit-CPU**), Μία Μνήμη Μόνο για Ανάγνωση (**Read Only Memory-ROM)** και την Μνήμη Τυχαίας Προσπέλασης (**Random Access Memory-RAM)**. Το **VLSI** επέτρεψε στους σχεδιαστές των **IC** να προσθέσουν και να ενώσουν όλα αυτά σε ένα μόνο **chip**. Η ραγδαία εξέλιξη της βιομηχανίας ηλεκτρονικών τις τελευταίες δεκαετίες οφείλεται ως επί το πλείστων στην εξέλιξη της μεγάλης κλίμακας συστημάτων τεχνολογίας και εφαρμογών στη σχεδίαση συστημάτων.

Ροή Σχεδίασης Ολοκληρωμένων Κυκλωμάτων (VLSI Design Flow)

Ένα κύκλωμα **VLSI** αποτελείται από εκατομμύρια τρανζίστορ, καθιστώντας την υλοποίησή του κυκλώματος μια πολύπλοκη διαδικασία. Προκειμένου να ελαχιστοποιηθεί η πολυπλοκότητα, ο σχεδιασμός του κυκλώματος επιμερίζεται σε υπό-κατηγορίες. Η σχεδίαση ξεκινάει από την αρχική ιδέα του κυκλώματος μέχρι την κατασκευή του στο εργοστάσιο με την βοήθεια διάφορων Εργαλείων Αυτοματοποίησης της σχεδίασης (**Computer Aided Design-CAD**). Αυτό συντελείται εξαιτίας της αδυναμίας των ανθρώπινων δυνατοτήτων να ανταπεξέλθουν σε ένα τόσο μεγάλο σε αριθμό τρανζίστορ. Οι ανθρώπινες δυνατότητες ανταποκρίνονται αποτελεσματικότερα σε γνωστικές διεργασίες, όπως οι αριθμητικές και μνήμης μονάδες, οι διασυνδέσεις των δικτυών και οι χειρισμοί τους. Η σχεδίαση αποτελεί ένα καταλυτικό στάδιο και είναι γνωστό ως Καθορισμός Αρχιτεκτονικής.

Figure 1. VLSI Design Process



Ο καθορισμός Αρχιτεκτονικής ενός **chip** γίνεται από ειδικούς μηχανικούς. Οι αποφάσεις σε αυτό το σημείο επηρεάζουν το κόστος και την απόδοση της σχεδίασης. Οι αποφάσεις αυτές αντλούνται με βάση τις εξής ερωτήσεις:

- "Ποια πρέπει να είναι η αρχιτεκτονική συνόλου εντολών του επεξεργαστή;"
- "Ποιοι τρόποι διευθυνσιοδότησης εντολών στη μνήμη θα υποστηρίζονται; "
- "Θα πρέπει το σετ εντολών να είναι συμβατό με κάποιο άλλο επεξεργαστή που βρίσκεται ήδη στην αγορά;"
- "Θα πρέπει ο επεξεργαστής να έχει κρυφή μνήμη; "
- "Ποσό μεγάλη θα είναι η κρυφή μνήμη;"
- "Ποια θα είναι η οργάνωση της κρυφής μνήμης;"
- "Ποιες θα είναι οι διαφορές της κρυφής μνήμης σε σχέση με τη μνήμη δεδομένων;"
- "Ποια θα είναι η διεπαφή του επεξεργαστή με τον εξωτερικό κόσμο;"
- "Υπάρχουν κάποια διεθνή **standards** που πρέπει να ληφθούν υπόψη;"

Ο καθορισμός αρχιτεκτονικής δεν μπορεί να λειτουργήσει αυτοματοποιημένα, δηλαδή να υλοποιηθεί αποκλειστικά από έναν υπολογιστή, όμως δύναται να συνδράμει στη λήψη αποφάσεων του σχεδιαστή. Μια από αυτές είναι η ρύθμιση μιας παραμέτρου, όπως το μέγεθος της κρυφής μνήμης μέσω προσομοίωσης. Προσομοιωτές και εργαλεία πρόβλεψης της απόδοσης συνιστούν χρήσιμους παράγοντες για πειραματικούς ελέγχους [1].

Αφού οριστεί η σχεδίαση του συστήματος, είναι αναγκαία η υλοποίηση των παρακάτω βημάτων:

- Η αναλυτική λογική σχεδίαση των κυκλωμάτων.
- Να ρυθμιστούν τα σήματα ελέγχου που είναι απαραίτητα για την ενεργοποίηση και απενεργοποίηση των κυκλωμάτων.

Το πρώτο βήμα είναι γνωστό ως διαδρομή δεδομένων (**data path**) και περιέχει διάφορα λειτουργικά **block**, στοιχεία αποθήκευσης και **hardware** κομμάτια που επιτρέπουν τη μεταφορά δεδομένων μεταξύ των **block** και των στοιχείων αποθήκευσης. Κάποια από αυτά είναι οι μνήμες τυχαίας πρόσβασης, οι καταχωρητές ολίσθησης, οι στοίβες και οι ουρές. Η μεταφορά των δεδομένων γίνεται συνήθως μέσω ενός συνδυασμού πολυπλεκτών και αποπολυπλεκτών. Το δεύτερο βήμα ονομάζεται διαδρομή ελέγχου (**control path**), το οποίο δημιουργεί τους ελέγχους για τα σήματα που είναι απαραίτητα για τη λειτουργία του κυκλώματος. Τα σήματα ελέγχου είναι απαραίτητα για την εκκίνηση της μεταφοράς δεδομένων μεταξύ των **block**. Η διαδρομή δεδομένων συνήθως υλοποιείται μέσω **hardwired control (random logic)** ή **microprogrammed control.**

Λογική Σχεδίαση

Η διαδρομή δεδομένων και η διαδρομή ελέγχου (που διεξάγονται αυτόματα ή χειροκίνητα) έχουν στοιχεία όπως αριθμητικές/λογικές μονάδες, καταχωρητές μετατόπισης, πολυπλέκτες, **buffer** και ούτω καθεξής. Τα επόμενα βήματα σχεδιασμού εξαρτώνται από τους ακόλουθους παράγοντες:

(1) Υλοποίηση του κυκλώματος είτε ως **VLSI chip** είτε ως **Printed Circuit Board (PCB)**.

(2) Διαθεσιμότητα εξαρτημάτων.

Εάν το κύκλωμα σχεδιαστεί ως ένα **PCB** χρησιμοποιώντας **off-the-shelf** υπό-κυκλώματα, τότε το επόμενο στάδιο στο σχεδιασμό είναι η επιλογή των εξαρτημάτων ώστε να ελαχιστοποιηθεί το συνολικό κόστος και ταυτόχρονα να μεγιστοποιηθεί η απόδοση. Μετά τη διαδικασία επιλογής, τα **IC chip** τοποθετούνται σε μια ή περισσότερες πλακέτες κυκλωμάτων και οι απαραίτητες διασυνδέσεις δημιουργούνται χρησιμοποιώντας ένα ή περισσότερα στρώματα μεταλλικών αποθέσεων. Παρόμοια διαδικασία μπορεί να ακολουθηθεί και στη περίπτωση ενός **VLSI chip** χρησιμοποιώντας έτοιμες δομικές μονάδες από κάποια τεχνολογική βιβλιοθήκη. Οι μονάδες αυτές είναι γνωστές ως κελιά (**cells**) και τοποθετούνται στην προσφερόμενη επιφάνεια όπου καλωδιώνονται μεταξύ τους χρησιμοποιώντας διασυνδέσεις μετάλλου και πολυπυριτίου .

Φυσική Σχεδίαση

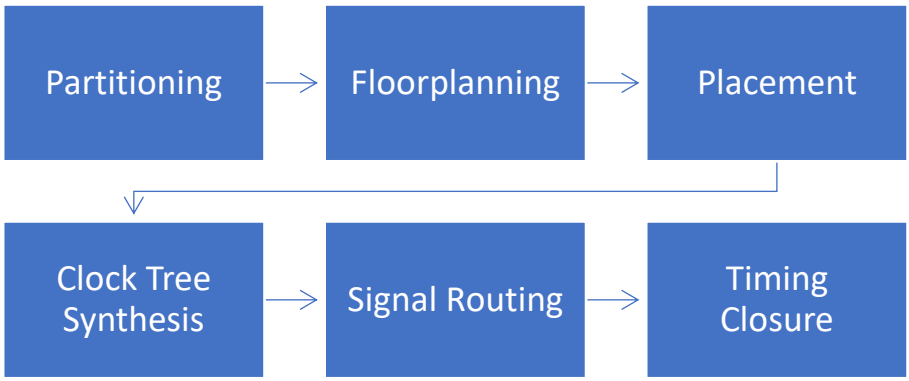
Κατά τη διάρκεια του φυσικού σχεδιασμού, όλα τα στοιχεία σχεδίασης παρουσιάζονται με τις γεωμετρικές τους αναπαραστάσεις τους. Τα κελία, οι πύλες, τα macros κ.λ.π παρουσιάζονται ως σταθερά σχήματα τα οποία τοποθετούνται στις προσφερόμενες θέσεις του chip (**placement**) και έχουν κατάλληλες συνδέσεις δρομολόγησης (**routing**) στα διαθέσιμα στρώματα μετάλλου του **chip.** Το αποτέλεσμα της φυσικής σχεδίασης είναι ένα σύνολο προδιαγραφών κατασκευής που πρέπει στη συνέχεια να επαληθευτεί. Η παρούσα διαδικασία επηρεάζει άμεσα την απόδοση του κυκλώματος, την καταλαμβανομένη περιοχή, την αξιοπιστία και την ισχύ:

- **Καταλαμβανόμενη Περιοχή**: Η τοποθέτηση των συνδεδεμένων μονάδων σε μεγάλη απόσταση οδηγεί σε μεγαλύτερα και πιο αργά **chip**.
- **Αξιοπιστία**: Μεγάλος αριθμός συνδέσεων μεταξύ διαφορετικών επιπέδων μετάλλου μπορεί να μειώσει σημαντικά την αξιοπιστία του κυκλώματος.
- **Απόδοση**: Η δρομολόγηση των καλωδίων διασύνδεσης σε κοντινές μεταξύ τους αποστάσεις μπορεί να οδηγήσει στη μείωση της απόδοσης λόγω πιθανών βραχυκυκλωμάτων στο στάδιο της κατασκευής. Η αντίθετη προσέγγιση μπορεί να οδηγήσει σε μακρύτερες συνδέσεις με αποτέλεσμα τον χειρότερο χρονισμό της σχεδίασης [2].

Η διαδικασία της Φυσικής Σχεδίασης παρουσιάζει μεγάλη πολυπλοκότητα με αποτέλεσμα να απαιτείται ο διαχωρισμός της σε επιμέρους βήματα:

- **Partinioning**: Διάσπαση του κυκλώματος σε μικρότερες υπό-διαστάσεις προκειμένου να είναι εφικτή η επιμέρους ανάλυση τους.
- **Floorplanning:** Καθορισμός των σχημάτων και της διάταξης των υπό-κυκλωμάτων, καθορισμός των σχετικών θέσεων των **I/O Pads** και των **macros**.
- **Power Planning:** Δρομολόγηση των δικτύων ισχύος (**VDD**) και γείωσης (**GND**) σε όλο το **chip**.
- **Placement:** Καθορισμός των συντεταγμένων όλων των κελιών.
- **Clock Tree Synthesis:** Δρομολόγηση των σημάτων ρολογιού.
- **Routing**: Δρομολόγηση των διασυνδέσεων μεταξύ όλων των μονάδων σχεδίασης.

Figure 2. Physical Design Flow



Partitioning

Ένα **chip** μπορεί να περιέχει πολλά εκατομμύρια τρανζίστορ. Λόγω των περιορισμών του χώρου μνήμης και της διαθέσιμης υπολογιστικής ισχύος μπορεί να μην είναι δυνατή η διάταξη ολόκληρου του **chip** στο ίδιο βήμα. Επομένως, το **chip** διαχωρίζεται σε υπό-κυκλώματα τα οποία ονομάζονται **block**. Το ***Partinioning*** λαμβάνει υπόψη πολλούς παράγοντες όπως το μέγεθος των **block**, τον αριθμό των **block** και τον αριθμό των διασυνδέσεων ανάμεσα τους. Το αποτέλεσμα της διαδικασίας είναι ένα σύνολο από **block** και οι μεταξύ τους διασυνδέσεις. Σε πολύ μεγάλα κυκλώματα, η διαδικασία του ***Partinioning*** είναι ιεραρχική, το οποίο σημαίνει ότι η σχεδίαση χωρίζεται αναδρομικά σε μικρότερα **block**. Κάποια από τα προβλήματα που δύνανται να βοηθήσει στην επίλυση τους η διαδικασία του Partitioning είναι τα ακόλουθα: [3]

- **Συσκευασία**: Ο διαχωρισμός διασπά το σύστημα προκειμένου να ικανοποιήσει τους φυσικούς περιορισμούς κατασκευής.
- **Στρατηγική Διαίρει και Βασίλευε (Divide And Conquer)**: Αυτή η στρατηγική υιοθετείται για την αποσύνθεση του έργου μεταξύ των μελών της ομάδας και την κατασκευή μιας λογικής ιεραρχίας για το στάδιο της λογικής σύνθεσης.
- **Εξομοίωση Συστήματος και Δημιουργία Πρωτοτύπου**: Υλοποίηση της σχεδίασης σε **FPGA (Field Programmable Gate Arrays)** υπό τη μορφή ενός πρωτοτύπου πάνω στο οποίο πάνω μπορούμε να κάνουμε πραγματικές μετρήσεις.

Floorplanning

Πριν από το στάδιο της χωροθέτησης, η σχεδίαση χωρίζεται σε μεμονωμένες μονάδες κυκλώματος. Μια μονάδα είναι ένα ορθογώνιο **block** στο οποίο έχουν εκχωρηθεί διαστάσεις ή κάποιο σχήμα. Αυτά τα **block** μπορούν να χαρακτηριστούν είναι είτε ως σκληρά είτε ως μαλακά, αναλόγως με το κατά πόσο οι διαστάσεις τους είναι σταθερές ή μπορούν να αλλάξουν.

Το Floorplanning, σε συνδυασμό με την ανάθεση **I/O Pads**, εξασφαλίζει ότι σε κάθε δομική μονάδα εκχωρείται ένα σχήμα και μια θέση, έτσι ώστε να διευκολύνεται η διαδικασία της χωροθέτησης. Σε ένα τυπικό παράδειγμα Floorplanning λαμβάνονται υπόψη οι ακόλουθοι παράμετροι:

- Η καταλαμβανόμενη περιοχή κάθε δομικής μονάδας.
- Οι πιθανές αναλογίες διαστάσεων κάθε δομικής μονάδας.
- Το πλήθος των διασυνδέσεων μεταξύ κάθε δομικής μονάδας.

Μετά την ολοκλήρωση του προκείμενου σταδίου έχουν βελτιστοποιηθεί τόσο οι τοποθεσίες όσο και οι διαστάσεις των **block**. χρησιμοποιώντας απλές αντικειμενικές συναρτήσεις για την αποτύπωση της πρακτικά επιθυμητής κάτοψης. Παρακάτω θα αναφερθούν μερικοί από τους στόχους του **Floorplanning**

- Όρια και σχήμα καταλαμβανόμενης περιοχής.
- Συνολικό μήκος καλωδίου διασύνδεσης.
- Καθυστερήσεις σήματος.

Placement

Η διαδικασία του **Placement** επηρεάζει άμεσα τον κύκλο σχεδίασης. Ακόμα και στις μέρες μας παραμένει μια σημαντική πρόκληση λόγω της αυξανομένης περιπλοκότητας του σχεδιασμού.

Η νόμος του **Moore** βρίσκεται σε ισχύ ακόμα όσο αναφορά την ανάπτυξη σχεδίασης ημιαγωγών. Ωστόσο στη μέτα-Moore εποχή η βιομηχανία θα αντιμετωπίσει σοβαρές προκλήσεις λόγω των πολύπλοκων στόχων και περιορισμών, γεγονός που καθιστά την αυτοματοποίηση της διαδικασίας της σχεδίασης αναγκαία. [4]

Η κακή ποιότητα χωροθέτησης ενός κυκλώματος μπορεί να προκαλέσει αποτυχία δρομολόγησης και την ανάγκη επανάληψης των τελευταίων βημάτων της σχεδίασης. Η

χωροθέτηση αποτελεί ένα πρόβλημα με πολλαπλούς στόχους βελτιστοποίησης το οποίο χαρακτηρίζεται ως **NP-complete**. Λόγω της πολυπλοκότητάς των σύγχρονων σχεδιάσεων, η διαδικασία διαιρείται στα ακόλουθα βήματα :

- Καθολική Χωροθέτηση (**Global Placement**)
  - Καθορισμός των θέσεων κατά προσέγγιση.
  - Επιτρέπονται φαινόμενα αλληλοεπικάλυψης.
- Νομιμοποίηση (**Legalization**)
  - Εξουδετέρωση όλων των αλληλοεπικαλύψεων.
  - Διόρθωση όλων των κανόνων σχεδίασης που έχουν παραβιαστεί.
- Λεπτομερής Χωροθέτηση (**Detailed Placement**)
  - Περαιτέρω αλλαγές για την βελτίωση της ποιότητας του τελικού αποτελέσματος.

Δεδομένης της σημασίας και της δυσκολίας του **Placement**, έχει διεξαχθεί εκτεταμένη έρευνα τα τελευταία 50 χρόνια. Διάφοροι αλγόριθμοι χρησιμοποιούνται για την επίλυση των προβλημάτων που παρουσιάζονται, συμπεριλαμβανομένων μεθόδων βασισμένων σε διαμερισμό, ευρετικές προσεγγίσεις, αναλυτικές μεθόδους και μηχανική μάθηση. Ακόμα και με όλες αυτές τις προσπάθειες, ωστόσο, η ποιότητα και η αποδοτικότητα των χωροθετήσεων απαιτεί περαιτέρω βελτιστοποίηση. Επιπλέον, η συρρίκνωση του μεγέθους των χαρακτηριστικών και η αυξανόμενη κλίμακα του σχεδιασμού επιφέρουν αυξημένη πολυπλοκότητα στο πρόβλημα, διατηρώντας το στην αιχμή της τεχνολογικής έρευνας.
(Yihang Qiu, 2023: 3-4)[5]

Ορισμός προβλήματος : Η τοποθέτηση είναι ένα πρόβλημα πολλαπλών στόχων βελτιστοποίησης που απαιτεί καθορισμό των θέσεων των **cells.**

Το πρόβλημα μπορεί να περιγραφεί ως εξής: Δεδομένης της περιοχής τοποθέτησης **P** του **netlist** εισόδου **G = (V, E)** (όπου **V** συμβολίζει τα cell, το **E** συμβολίζει τις συνδέσεις μεταξύ των κυκλωμάτων), και της συνάρτησης κόστους **F = (V, E),** βρείτε τη θέση (**xi, yi**) κάθε **vi ∈ V** ώστε:

- Να ελαχιστοποιηθεί μια συνάρτηση κόστους **F = (V, E)**.
- Να ικανοποιηθούν οι σχεδιαστικοί περιορισμοί.

Οι στόχοι σχετίζονται συνήθως με:

- Το συνολικό μήκος καλωδίωσης
- Τη δυνατότητα δρομολόγησης του κυκλώματος.
- Τη βελτίωση του χρονισμού της σχεδίασης.
- Την κατανάλωση ισχύος.
- …

Βήματα τοποθέτησης με χρήση βιομηχανικών εργαλείων :

1. **Global Placement**
2. **Legalization**
3. **Hign Fanout Net Synthesis**
4. **Iteration for Congestion**
5. **Iteration for Timing**
6. **Iteration for Design Rule Violations**
7. **Iteration for Power Optimization**
8. **Scan-Chain Reordering**
9. **Tie Cell insertion**

Κατασκευή

Η τελική διάταξη πρέπει να είναι **Design Rule Violations(DRC)-/ Layout Versus Schematic (LVS)-/ Electrical Rule Checks(ERC) clean**, και συνήθως αποθηκεύεται στο **format GDSII**, το οποίο αποστέλλεται για κατασκευή σε κάποιο χυτήριο πυρίτιού (**fab**). Η παράδοση του σχεδίου και η διαδικασία κατασκευής ονομάζεται **tapeout**, η μετάδοση δεδομένων από την ομάδα σχεδιασμού στο **silicon fab** πλέον δεν βασίζεται σε μαγνητική ταινία.

Γενετικός Αλγόριθμος

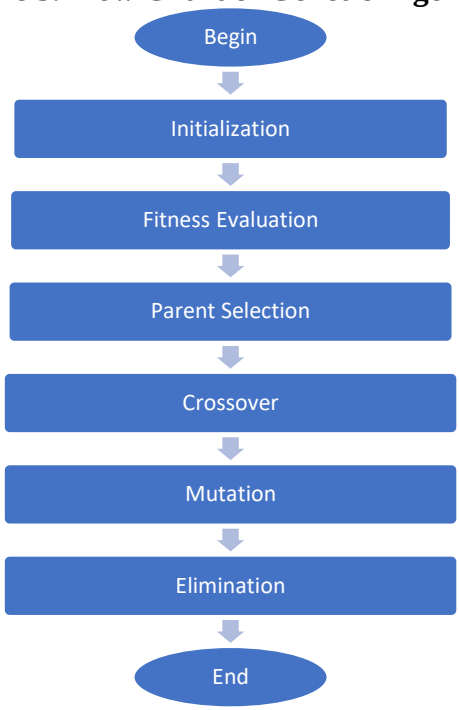
Ιστορική Αναδρομή

Από την δεκαετία του 1950 οι επιστήμονες μελετούν την τεχνητή νοημοσύνη και τον εξελεγκτικό υπολογισμό, προσπαθώντας να γράψουν προγράμματα τα οποία προσπαθούν να παρομοιώσουν διεργασίες που συμβαίνουν στον κόσμο. Το 1953, το πανεπιστήμιο **Princeton** προσκάλεσε τον **Nils Barricelli** για να μελετήσει την τεχνητή νοημοσύνη χρησιμοποιώντας τον ψηφιακό υπολογιστή, τον οποίο χρησιμοποίησε για να δημιουργήσει λογισμικό που μιμούνταν την φυσική αναπαραγωγή και τη μετάλλαξη. Στόχος του δεν ήταν να λύσει προβλήματα βελτιστοποίησης ή να προσομοιώσει την βιολογική εξέλιξη, αλλά να δημιουργήσει τεχνητή ζωή. Η δουλειά του συνεχίστηκε από τον **Alexander Fraser**, έναν βιολόγο από το Λονδίνο, ο οποίος είχε την ιδέα να δημιουργήσει ένα υπολογιστικό μοντέλο της εξέλιξης. Η ιδέα του προήλθε από το γεγονός ότι η απευθείας παρακολούθηση ενός θέματος χρειάζονταν εκατομμύρια χρονιά. Ήταν ο πρώτος που χρησιμοποίησε τον προγραμματισμό αποκλειστικά για να μελετήσει την εξέλιξη. Στη συνέχεια, τη δεκαετία του 1960 ο **John Holland** θα εφεύρει τον Γενετικό Αλγόριθμο. Η εκδοχή του συμπεριλάμβανε την Δαρβίνεια Θεωρία για την επιβίωση του δυνατότερου, αλλά και διεργασίες όπως η διασταύρωση (**Crossover**), ανασυνδυασμός (**Recombination**) και της μετάλλαξης (**Mutation**). Οι μέχρι τότε επιστήμονες χρησιμοποιούσαν μόνο την μετάλλαξη ως οδηγό στην ερευνά τους για την εξέλιξη. Κινητήρια δύναμη και έμπνευση του ήταν η θέληση για κατανόηση προσαρμογής των συστημάτων στον περίγυρο τους. Έπειτα από χρονιά συνεργασίας με μαθητές και συνεργάτες, δημοσιεύει το περίφημο βιβλίο του που ονομάζεται **Adaptation in Natural and Artificial Systems** το 1975. Αυτό το βιβλίο ήταν το πρώτο που πρότεινε θεωρητικά θεμέλια για την υπολογιστική εξέλιξη[6][7][8].

Ορολογία

- Κάθε κύτταρο ζωντανού οργανισμού εμπεριέχει χρωμοσώματα, Συμβολοσειρές από **DNA**
- Κάθε χρωμόσωμα έχει ένα σύνολο από γονίδια, κομμάτια του DNA
- Κάθε γονίδιο καθορίζει κάποιο χαρακτηριστικό του οργανισμού
- Αναπαραγωγή(**Crossover**) εμπεριέχει συνδυασμό γονιδίων από τους γονείς και μετά μικρές ποσότητες από παραλλαγές(**Mutation**) στην αντιγραφή.
- Η δυναμική(**Fitness**) ενός οργανισμού είναι το κατά πόσο μπορεί να αναπαράγει πριν το θάνατο του.
- Η εξέλιξη βασίζεται στην επιβίωση του δυνατότερου.

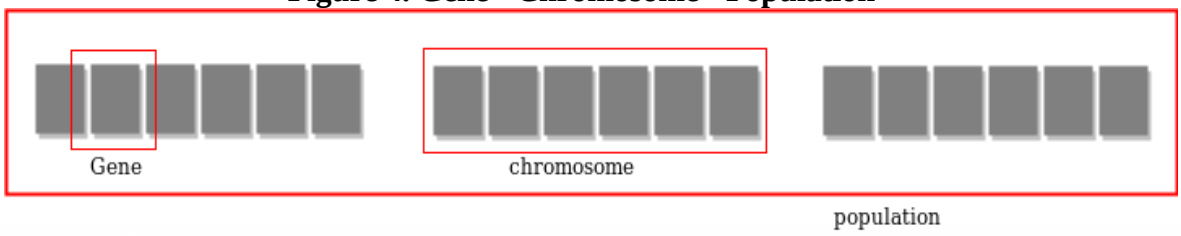
Figure 3. Flow Chart of Genetic Algorithm



Κωδικοποίηση

Υπάρχει ένας πληθυσμός από οντότητες και η κάθε οντότητα αποτελεί και μια λύση στο πρόβλημα. Η κάθε λύση στον γενετικό ονομάζεται χρωμόσωμα. Το χρωμόσωμα, όπως και στη φύση αποτελείται από γονίδια [11].

Figure 4. Gene - Chromosome - Population



Στον αλγόριθμο, η αναπαράσταση και υλοποίηση των χρωμοσωμάτων και των γονιδίων μπορεί να γίνει με διάφορους τρόπους, η επιλογή εξαρτάται κυρίως από το πρόβλημα που πρέπει να επιλυθεί διακατέχοντας πολύ σημαντικό ρόλο. Παρακάτω θα παρατεθούν μερικοί από τους τρόπους που μπορεί να γίνει αυτό.

Δυαδική Κωδικοποίηση: Είναι η πιο συχνή που συναντιέται στους γενετικούς αλγόριθμους. Το κάθε γονίδιο μπορεί να πάρει την τιμή 1 ή 0, όπου η κάθε τιμή θα αναπαριστά χαρακτηριστικά της λύσης. Η Δυαδική Κωδικοποίηση καθιστά την εκτέλεση των τελεστών εύκολη και γρήγορη, όμως χρειάζεται και απαιτεί συχνά στη συνέχεια μια τροποποίηση των αποτελεσμάτων σε μια πιο εύχρηστη μορφή. Για παράδειγμα στο πρόβλημα του **Knapsack**, το κάθε **bit** μας δείχνει αν το αντικείμενο βρίσκεται μέσα στην τσάντα ή όχι.

Table 1. Example og Binary Encoding

Chromosome1	110101110010
Chromosome2	011010011110

Οκταδική κωδικοποίηση: Τα γονίδια και το χρωμόσωμα παρουσιάζονται στην μορφή (0 – 7). Εξαιτίας όμως της δυσκολίας αναπαράστασης ενός χρωμοσώματος με αυτή τη μορφή, δεν επιλέγεται συχνά.

Table 2. Example of Octal Encoding

Chromosome1	06254524
Chromosome2	63725425

Δεκαεξαδική κωδικοποίηση: Αντίστοιχα υπάρχει η δεκαεξαδική κωδικοποίηση όπου οι τιμές είναι (0-9,A-F). Οπως στην οκταδική, συναντιέται πολύ σπάνια.

Table 3. Example of Hexadecimal Encoding

Chromosome1	9F2A
Chromosome2	A1B6

Permutation encoding: Αυτή η κωδικοποίηση χρησιμοποιείται για προβλήματα διάταξης. Το κάθε χρωμόσωμα είναι μια αλληλουχία αριθμών τα οποία εκπροσωπούν μια θέση στη σειρά. Για παράδειγμα, στο πρόβλημα του πλανόδιου πωλητή, ο κάθε αριθμός μας δείχνει ποια πόλη έχει επισκεφτεί και με ποια σειρά.

Table 4. Example of Permutation Encoding

Chromosome1	1 5 2 3 5 2 6 4 6 9 8
Chromosome2	8 6 3 6 3 9 6 3 1 5 8

Κωδικοποίηση τιμής: Εδώ το κάθε γονίδιο μπορεί να είναι ένας ακέραιος, πραγματικός, χαρακτήρας ή κάποιο αντικείμενο. Χρησιμοποιείται στα νευρωνικά δίκτυα για την αναζήτηση των βαρών [9].

Table 5. Example of Value Encoding

Chromosome1	1.23, 2.12, 3.14, 0.34, 4.62
Chromosome2	ABDJEIDHFFLDFLFEGT

Αρχικοποίηση

Η αρχικοποίηση του πληθυσμού είναι το πρώτο βήμα στην εκτέλεση του γενετικού αλγόριθμου. Σημαντικό στοιχείο είναι να αποφασιστεί το μέγεθος του πληθυσμού, καθώς ένας μεγάλος αριθμός μπορεί να οδηγήσει σε αύξηση της ώρας εκτέλεσης του προγράμματος, και αντίστροφα, ένας μικρός αριθμός μπορεί να οδηγήσει γρήγορα στην σύγκλιση και σε όμοιες λύσεις. Η αρχικοποίηση μπορεί να γίνει με τρεις τρόπους.

Τυχαία παραγωγή: Η τυχαία παραγωγή του αρχικού πληθυσμού, προσφέρει τη δυνατότητα να ερευνηθούν πάρα πολλές διαφορετικές λύσεις, αυξάνοντας όμως τον χρόνο εκτέλεσης.

Ευρετικού αλγορίθμου: Ο ευρετικός αλγόριθμος που θα χρησιμοποιηθεί εξαρτάται από το πρόβλημα. Για το πρόβλημα του πλανόδιου πωλητή (**Travelling Salesman Problem-TSP**) μπορεί να χρησιμοποιηθεί για παράδειγμα ο αλγόριθμος του πλησιέστερου γείτονα **(nearest neighbor)**. Με αυτό τον τρόπο αυξάνεται η ταχύτητα σύγκλισης του γενετικού αλγόριθμου. Αλλά επειδή μειώνεται ο χώρος των πιθανών λύσεων, υπάρχει κίνδυνος ο αλγόριθμος να εγκλωβιστεί σε ένα τοπικό ακρότατο[12][13].

Υβριδική αρχικοποίηση: Για να αυξηθεί η ταχύτητα σύγκλισης, αλλά και να έχει ο γενετικός αλγόριθμος ένα αρκετά μεγάλο χώρο αναζήτησης λύσεων, ένα μέρος του αρχικού πληθυσμού παράγεται τυχαία και ένα μέρος με τη χρήση κάποιου ευρετικού αλγορίθμου.

Συνάρτηση Αξιολόγησης

Στη φύση οι πιο ισχυροί οργανισμοί είχαν μεγαλύτερη πιθανότητα επιβίωσης σε σύγκριση με κάποιους άλλους. Η έννοια «ισχυρός» δε νοείται μόνο η φυσική του δύναμη, που παίζει και αυτή ένα σημαντικό ρόλο, αλλά και η δυνατότητα προσαρμογής σε ένα διαφορετικό κλίμα ή περιοχή. Αυτά τα χαρακτηριστικά επιτρέπουν σε κάποιες οντότητες να έχουν μεγαλύτερη πιθανότητα επιβίωσης σε σχέση με κάποιους άλλους. Με αποτέλεσμα, αυτοί που επιβιώνουν, σύμφωνα με την φυσική επιλογή, να κληρονομούν κάποια χαρακτηριστικά τους στις επόμενες γενιές, όπου σε βάθος χρόνου η νέα γενιά να γίνεται όλο και περισσότερο δυνατή και ανθεκτική.

Οι γενετικοί αλγόριθμοι προσπαθούν να παρομοιάσουν αυτή τη διαδικασία. Σε κάθε γενιά, το κάθε χρωμόσωμα πρέπει να αξιολογηθεί, δηλαδή να κριθεί κατά πόσο ο συνδυασμός των γονιδίων του είναι ισχυρός, ώστε να έχει και μεγαλύτερη πιθανότητα να περάσει τα γονίδια του στην επόμενη γενιά. Στον γενετικό, όπως και στη φύση, οι λύσεις που έχουν αξιολογηθεί ως δυνατότερες, θα είναι και αυτές που θα προτιμηθούν για να δημιουργηθεί η επόμενη γενιά.

Η αξιολόγηση γίνεται με μια συνάρτηση που έχει ως παράμετρο ένα χρωμόσωμα και επιστρέφει μια τιμή (**Fitness Value**) η οποία θα περαστεί στο χρωμόσωμα ως αξιολόγηση δύναμης. Συνήθως όσο μεγαλύτερη είναι η τιμή που επιστρέφεται, τόσο καλύτερη είναι και η βαθμολογία του. Όμως αυτό εξαρτάται και από το πρόβλημα, το οποίο μπορεί σε μια περίπτωση να έχουμε ως βέλτιστη τιμή την πιο μικρή [20][21].

Επιλογή Γονέων

Στη φάση της επιλογής είναι που επιλέγονται ποια χρωμοσώματα θα αναπαραχθούν με την ελπίδα να αποδώσουν παιδιά που θα έχουν μεγαλύτερο **Fitness Value** από τους γονείς. Για αυτό και διαμορφώνεται έτσι η επιλογή ώστε οι δυνατότεροι γονείς να έχουν μεγαλύτερη πιθανότητα να επιλεχθούν σε σύγκριση με τους πιο αδύναμους, όμως πρέπει να παρέχεται η δυνατότητα επιλογής και στα πιο αδύναμα μέλη ώστε να αποφευχθεί η κατάληξη σε ένα **Local Minimal**. Κύριος στόχος της επιλογής γονέων είναι η εξάλειψη των πιο αδύναμων λύσεων, χωρίς όμως να εξαφανιστεί η ποικιλομορφία που είναι ένα απαραίτητο στοιχείο για την επιτυχία ενός γενετικού αλγόριθμου. Παρακάτω, αναφέρονται μερικοί από τους τρόπους με τους οποίους γίνεται η επιλογή [14].

Ρουλέτα:

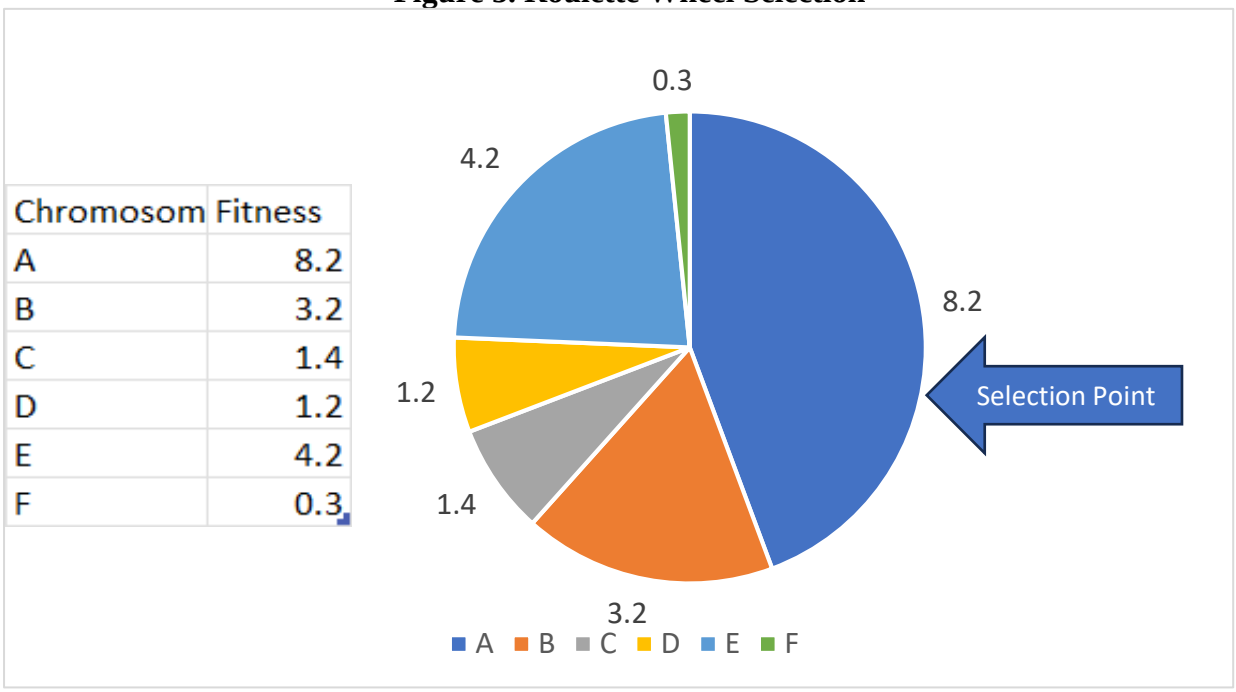
Στην διαδικασία της ρουλέτας, κάθε γονέας παίρνει από μια τιμή. Αυτή η τιμή υπολογίζεται από την συνάρτηση οπού **fi** είναι το **Fitness Value** του γονέα και η σειρά είναι το άθροισμα όλων τον **Fitness value** των γονέων.

$$p_i = \frac{f_i}{\sum_{j=1}^{n} f_j}$$

(pi είναι η πιθανότητα αυτός ο γονέας να εκλεχθεί για αναπαραγωγή.)

Με βάση αυτό το ποσοστό ο κάθε γονέας έχει ένα κομμάτι στην ρουλέτα, όσο μεγαλύτερο το ποσοστό, τόσο και μεγαλύτερο το κομμάτι στη ρουλέτα. Αντίστοιχα όσο μικρότερο το ποσοστό, τόσο μικρότερο και το κομμάτι. Προφανώς, ένας δυνατός γονέας έχει περισσότερες πιθανότητες να εκλεχθεί, και αυτές οι πιθανότητες αυξάνονται αναλόγως με το κομμάτι στην ρουλέτα. Το θετικό είναι πως αν και με μικρότερη πιθανότητα, όλοι οι γονείς έχουν την δυνατότητα να επιλεχθούν, ακόμα και αυτοί με το μικρότερο **Fitness Value**. Δηλαδή, η ποικιλομορφία διατηρείται ως ένα σημείο. Εντούτοις η ρουλέτα διακατέχει κάποιους κίνδυνους. Ο πρώτος κίνδυνος είναι η περίπτωση οπού ένας ή δυο γονείς έχουν πολύ μεγαλύτερο **Fitness value** σε σχέση με τους υπολοίπους, τότε θα επιλέγονται πολύ συχνά αυτοί οι δυο, με αποτέλεσμα να κυριαρχήσουν σαν επιλογές, κάτι που θα εξαλείψει κατά μεγάλο ποσοστό την ποικιλομορφία. Δεύτερος κίνδυνος είναι η περίπτωση που όλοι οι γονείς έχουν σχετικά ίδιο **fitness value**, τότε ο αλγόριθμος θα δυσκολευτεί να φτάσει σε καλύτερη λύση, αφού οι σχετικά αδύναμοι έχουν ιδιά δύναμη με τους πιο δυνατούς. Στο **Figure 5** υπάρχει η οπτικοποίηση της ρουλέτα χρησιμοποιώντας τυχαία δεδομένα.

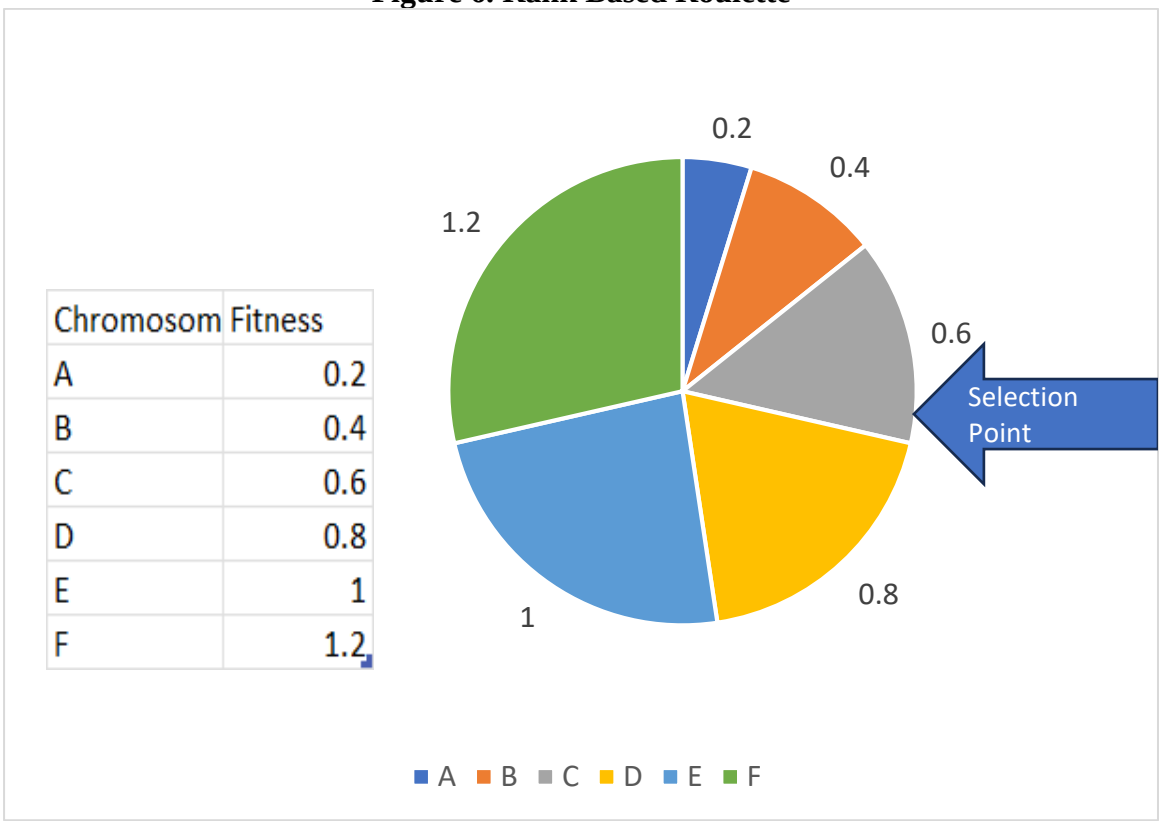
Figure 5. Roulette Wheel Selection



Rank Based ρουλέτα :

Η ρουλέτα με βάση το βαθμό, είναι ένας τρόπος επιλογής μόνο που οι πιθανότητες δεν καθορίζονται με βάση το **Fitness Value** του κάθε γονέα, άλλα με τη θέση του στην κατάταξη σε σχέση με τους άλλους γονείς. Στην αρχή γίνεται μια ταξινόμηση με βάση την αξία του κάθε γονέα, και έπειτα υπολογίζεται η τιμή που θα τους δοθεί, όσο υψηλοτέρα είναι στην κατάταξη, τόσο και μεγαλύτερη θα είναι η πιθανότητα να επιλεχθεί. Αυτή η στρατηγική μπορεί να βοηθήσει έτσι ώστε να καθυστερήσει η σύγκλιση, καθώς αποφεύγεται η πιθανότητα που υπάρχει στη ρουλέτα να επιλέγεται συνέχεια ένας γονιός που έχει πολύ μεγαλύτερη δύναμη σε σχέση με τους υπολοίπους. Όμως, υπάρχει και η πιθανότητα να καθυστερήσει υπερβολικά την σύγκλιση καθώς η πιθανότητα του πρώτου και πιο δυνατού γονιού θα είναι πάντα η ίδια, αντιστοιχία του δεύτερου και των υπολοίπων. Επίσης, αυτή η τεχνική έχει ένα αρνητικό ακόμα: στο κομμάτι της εκτέλεσης του αλγόριθμου, η συνεχή ταξινόμηση κάθε φορά που θα πρέπει να γίνει η επιλογή, απαιτεί αρκετά παραπάνω χρόνο και υπολογιστική ισχύ, κάτι που έχει ιδιαίτερη σημασία όταν οι επαναλήψεις είναι πολλές.

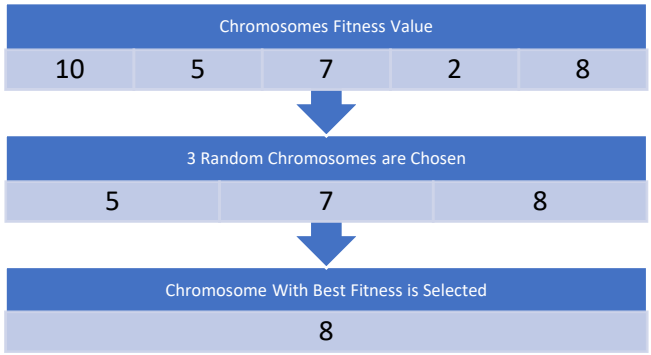
Figure 6. Rank Based Roulette



Τουρνουά:

Η επιλογή του τουρνουά ίσως είναι και η πιο γνωστή και διαδομένη, και αυτή με την οποία εκτελούνται οι περισσότεροι αλγόριθμοι. Αυτό οφείλεται στην αποτελεσματικότητα του αλλά και στην απλότητα υλοποίησης της. Σε αυτή την επιλογή διαλέγονται τυχαία γονείς οι οποίοι θα ανταγωνιστούν ο ένας τον άλλον, με νικητή αυτόν που έχει το μεγαλύτερο **fitness value**, ο οποίος θα είναι και αυτός που θα επιλεχτεί για να παράγει το παιδί. Το πόσοι γονείς μπορούν να ανταγωνιστούν ποικίλει. Η πιο συνηθισμένη επιλογή είναι το δυαδικό τουρνουά, οπού ανταγωνίζονται δυο γονείς ο ένας τον άλλον. Εντούτοις, θα μπορούσε και με τρεις, με αποτέλεσμα όμως να επισπεύσουμε την σύγκλιση. Επίσης, θετικό είναι ότι όλοι οι γονείς έχουν την πιθανότητα να επιλεχθούν, αλλά αντίστοιχα υπάρχει και η περίπτωση μια πολύ καλή λύση να μην επιλεχτεί ποτέ.

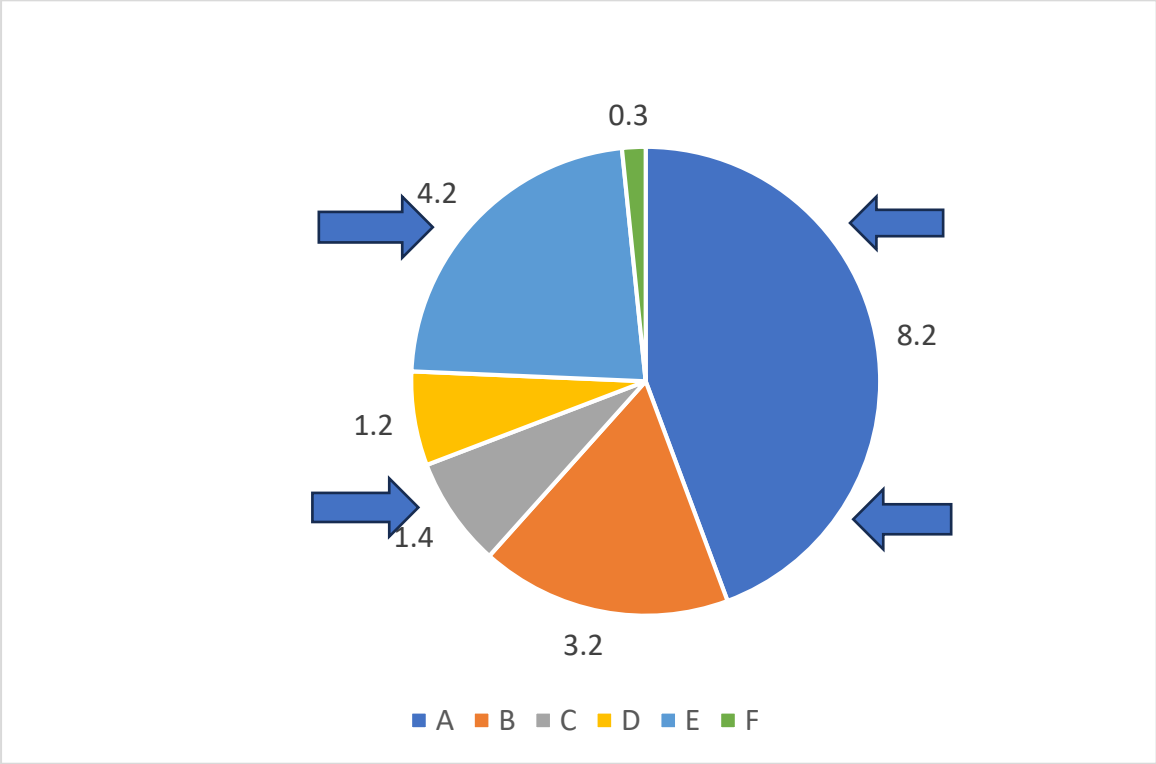
Figure 7. Tournament Selection Example with Size=3



Stochastic Universal Sampling (SUS):

Αποτελεί μια ελαφριά παραλλαγή της ρουλέτας που αναφέρθηκε στην αρχή. Χρησιμοποιεί ακριβώς την ίδια ρουλέτα και την ίδια λογική στην απόδοση ποσοστού πιθανότητας επιλογής. Υπάρχει μόνο μια διαφορά και αυτή είναι η εξής. Στην αρχική ρουλέτα υπήρχε ένας δείκτης όπου αφού σταματούσε το γύρισμα της ρουλέτας, ο δείκτης θα μας έδειχνε ποια οντότητα επιλέχθηκε. Σε αυτή την εκδοχή, αντί για ένα δείκτη, θα έχουμε Ν δείκτες, όπου Ν ισούται με το μέγεθος του αρχικού πληθυσμού. Δηλαδή σε μια μόνο περιστροφή θα επιλέγονται όλοι οι γονείς που χρειάζονται για να αναπαράγουν τα παιδιά. Επίσης να σημειωθεί πως οι δείκτες έχουν ίση απόσταση μεταξύ τους. Αυτή η παραλλαγή έχει ως θετικό, πως οριακά σιγουρεύει την επιλογή των δυνατότερων οντοτήτων και παράλληλα δίνει μια αυξημένη πιθανότητα στις πιο αδύναμες οντότητες.

Figure 8. Stochastic Universal Sampling (SUS)



Crossover

Στο προηγούμενο κεφάλαιο μελετήθηκαν διάφοροι τρόποι επιλογής γονέων, ο λόγος που έγινε αυτό είναι διότι η επιλογή γονέων χρησιμοποιείται στο επόμενο βήμα, την διασταύρωση γονίδιων (**Crossover**). Η διασταύρωση είναι ένας από τους τελεστές στον γενετικό αλγόριθμο που χρησιμοποιείται ώστε να διατηρηθεί η ποικιλομορφία και για να εξερευνηθούν άλλες καλύτερες λύσεις. Στόχος των τελεστών είναι να βρεθεί το καλύτερο δυνατό αποτέλεσμα στις όσο λιγότερες επαναλήψεις. Ο γενετικός αλγόριθμος παρομοιάζει τη λειτουργία του με την επιβίωση του δυνατότερου στην φύση, ο οποίος μεταφέρει τα γονίδια του στην επόμενη γενιά μαζί με τα χαρακτηριστικά που του επιτρέπουν να επιβιώνει. Έτσι και εδώ με την διασταύρωση, στόχος είναι μετά την επιλογή των γονέων, να διασταυρώσουμε αυτούς τους δυο, δηλαδή να κρατήσουμε γονίδια και από τους δυο και να τα συνδυάσουμε με την ελπίδα ότι θα αναπαραχθεί ένα παιδί δυνατότερο από τους γονείς. Αυτή η διαδικασία είναι που κρατάει και το περισσότερο βάρος στον γενετικό αλγόριθμο για διάφορους λογούς. Αρχικά είναι αυτή που έχει την μεγαλύτερη επιρροή στον αλγόριθμο, διότι αυτή κατά κόρον διαμορφώνει τον πληθυσμό. Για αυτό και η επιλογή του οποιουδήποτε τελεστή διασταύρωσης πρέπει να γίνει με προσοχή, διότι είναι και το κομμάτι στον αλγόριθμο που απαιτείται η μεγαλύτερη υπολογιστική ισχύ και ο περισσότερο χρόνος. Τέλος, είναι το μέρος οπού μπορεί να αποφευχθεί η γρήγορη σύγκλιση σε ένα **local minimal**. Παρακάτω θα αναφερθούν μερικοί από τους τρόπους με τους οποίους μπορεί να γίνει η διασταύρωση.

Στην διασταύρωση, μετά την επιλογή θα έχουμε δυο γονείς για αναπαραγωγή δυο παιδιών. Ένας γονέας είναι μια λύση, δηλαδή ένας πίνακας Ν, που σε κάθε θέση στον πίνακα υπάρχει ένα γονίδιο[15] .

Διασταύρωση σε ένα σημείο (Single Point Crossover):

Η διασταύρωση σε ένα σημείο είναι η πιο κοινή και διαδομένη στους γενετικούς αλγόριθμους. Επίσης, είναι και ο πιο απλός στην υλοποίηση. Επιλέγεται τυχαία ένας αριθμός Χ οποίος είναι προφανώς μικρότερος από το μέγεθος του πίνακα και θα συμβολίζει μια θέση στον πίνακα. Η διασταύρωση έχει ως εξής, το πρώτο παιδί θα αποτελείται από τα γονίδια 0 – Χ του πρώτου γονέα και τα γονίδια Χ μέχρι Ν του δευτέρου γονέα. Αντίστοιχα, το δεύτερο παιδί θα έχει 0-Χ γονίδια του $2^{ου}$ γονέα και Χ μέχρι Ν του $1^{ου}$ γονέα.

Διασταύρωση σε δύο σημεία(Two Point Crossover):

Η διαδικασία είναι σχετικά ιδιά με την προηγουμένη, μόνο που τώρα αντί να επιλεγεί ένα μέρος για να κοπεί ο πίνακας, θα παρθούν δυο. Επιλέγονται δυο τυχαίοι ακέραιοι αριθμοί Χ και Υ, ας θεωρηθεί για την περαιτέρω κατανόηση του αλγορίθμου ότι το Υ είναι μεγαλύτερο του Χ. Το $1^{ο}$ παιδί αποτελείται από τα γονίδια Χ-Υ του $2^{ο}$ γονέα και τα γονίδια 0-Χ και Υ-Ν είναι από τον $1^{ο}$ γονέα. Αντίστοιχα, το 2^{0} παιδί έχει τα γονίδια Χ-Υ του $1^{ου}$ γονέα και 0-Χ και Υ-Ν του $2^{ου}$.

Διασταύρωση σε Κ σημεία(K Point Crossover):

Η φιλοσοφία είναι παρόμοια, μόνο που τώρα ο αριθμός των κοψίματων είναι Κ, δηλαδή ο γονέας μπορεί να διαχωριστεί σε όσα μέρη είναι επιθυμητό και έπειτα με μια εναλλαγή να περαστούν στο παιδί. Για παράδειγμα, αν το Κ = 5, τότε το πρώτο παιδί θα έχει τα γονίδια του $1^{ου}$,$3^{ου}$,$5^{ου}$ κομματιού του πρώτου γονέα, και του $2^{ου}$ και $4^{ου}$ κομματιού από τον δεύτερο γονέα.

Uniform Crossover:

Σε αυτή τη περίπτωση το $1^{ο}$ παιδί σχετίζεται αποκλειστικά με τον $1^{ο}$ γονέα, και αντίστοιχα το $2^{ο}$ παιδί με τον $2^{ο}$ γονέα. Ας υποτεθεί ότι ένα κέρμα ρίχνεται Ν φορές, οπού Ν το μέγεθος του πίνακα. Αν η μια πλευρά αντιστοιχεί στον αριθμό 1 και η άλλη 0, τότε για κάθε φορά που βγαίνει 1 τότε θα αλλάζει το **bit** του γονέα και θα μεταφέρεται στο παιδί, αν το κέρμα είναι 0 τότε θα παραμένει το **bit** όπως είναι.

Discrete Crossover:

Εδώ υπάρχει μια παρόμοια περίπτωση με την προηγουμένη. Θα μπορούσε όπως προηγουμένως να υποτεθεί ότι υπάρχει ένα κέρμα το οποίο ρίχνεται Ν φορές, το οποίο θα μπορεί να φέρει την τιμή 1 ή 2. Η διαφορά είναι πως τώρα από τους δυο γονείς θα δημιουργηθεί μόνο ένα παιδί, εκ του οποίου τα γονίδια θα εξαρτώνται από την τιμή του κέρματος. Αν η τιμή είναι 1,τότε θα πάρει το γονίδιο του $1^{ου}$ γονέα, αντίστοιχα αν η τιμή είναι 2 τότε θα πάρει το γονίδιο του $2^{ου}$ γονέα, πχ την $5^{η}$ φορά που θα ριχτεί το κέρμα και έχει την τιμή 2, τότε το $5^{ο}$ γονίδιο του $2^{ου}$ γονέα θα περαστεί στο παιδί [16][17].

Mutation:

Ο τελεστής σε έναν γενετικό αλγόριθμο χρησιμοποιείται κυρίως για να διατηρηθεί η ποικιλομορφία στον πληθυσμό και να μην καταλήξει σε μια μεγάλη σύγκλιση των λύσεων. Σε αντίθεση με την συνδυασμό χρωμοσωμάτων, που συναντιέται στον τελεστή **Crossover**, εδώ η παράλλαξη έχει να κάνει με έναν μόνο γονέα την κάθε φορά, ο οποίος είναι ανεξάρτητος με όλους τους υπολοίπους. Ο τελεστής αυτός μπορεί να μην συμβάλει τόσο πολύ όσο ο προηγούμενος, όμως σε συνδυασμό αυτοί οι δυο μπορούν να βοηθήσουν σε πάρα πολύ μεγάλο ποσοστό να εξερευνηθούν καινούριες λύσεις με καλυτέρα αποτελέσματα διατηρώντας την ποικιλομορφία. Ο συγκεκριμένος έχει μια βοηθητική λειτουργία. Αν και εξαρτάται και από τον αλγόριθμο, αυτός ο τελεστής έχει ένα μικρό ποσοστό εκτέλεσης, δηλαδή δεν εκτελείται απαραίτητα σε κάθε επανάληψη, η οποία είναι χαμηλή του εύρους συνήθως 20-25% πιθανότητα να εκτελεστεί. Ο λόγος είναι ότι αν εκτελούνταν συνέχεια, ο αλγόριθμος πιθανόν να έφτανε το σημείο να έψαχνε τυχαία. Παρακάτω θα παραθέσουμε μερικούς τρόπους με τους οποίους μπορεί να γίνει η μετάλλαξη [18].

Bit Flip Mutation:

Σε αυτή την περίπτωση, επιλέγεται τυχαία ένα **bit** της λύσης, δηλαδή ένα γονίδιο του χρωμοσώματος, και αλλάζει η τιμή του. Αυτή η μετάλλαξη μπορεί να εφαρμοστεί μόνο σε γενετικούς που εφαρμόζουν την δυαδική κωδικοποίηση στην αναπαράσταση του χρωμοσώματος.

Figure 9. Example of Bit Flip Mutation

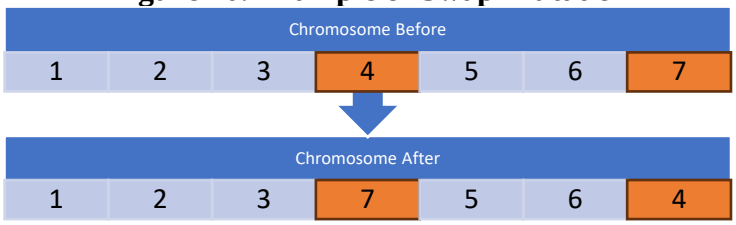
Random Resetting:

Σε αντίθεση με την προηγουμένη μετάλλαξη, η λογική είναι η ιδιά μόνο που αποτελεί μια παραλλαγή επειδή επιτρέπεται αντί να αλλάξουμε την τιμή ενός **bit**, να αλλάξουμε μια μεταβλητή που είναι αποδεκτή και λειτουργική για τον αλγόριθμο, όπως π.χ. ένας ακέραιος.

Swap Mutation :

Εδώ, διαλέγονται τυχαία δυο θέσεις γονίδιων και ανταλλάσσονται οι θέσεις μεταξύ τους στον πίνακα.

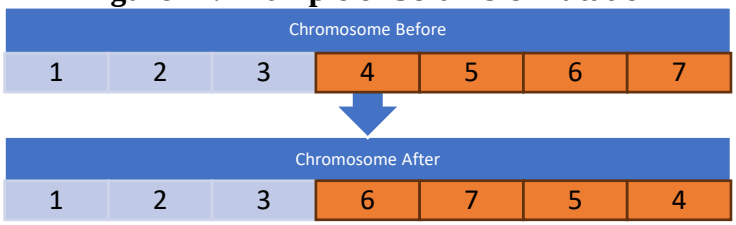
Figure 10. Example of Swap Mutation



Scramble Mutation:

Επιλέγεται ένας τυχαίος αριθμός γονίδιων στην σειρά από τον πίνακα και μπερδεύεται τυχαία η σειρά τους.

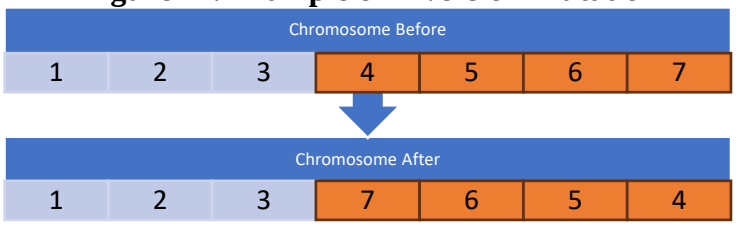
Figure 11. Example of Scramble Mutation



Inversion Mutation:

Το ίδιο με πριν μόνο που αντί να μπερδευτεί η σειρά, αντιστρέφεται[19].

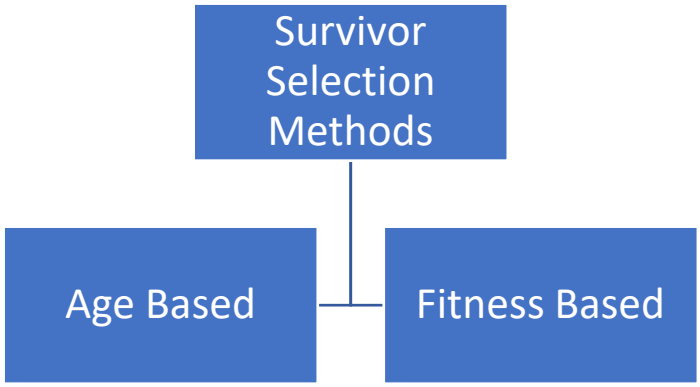
Figure 12. Example of Inversion Mutation



Επιλογή Επιβίωσης:

Πέρα από την επιλογή γονέων για αναπαραγωγή, στο σημείο αυτό πρέπει να επιλεχτούν ποιες λύσεις θα προχωρήσουν στην επόμενη γενιά. Στις περισσότερες περιπτώσεις γενετικών, τα παιδιά που δημιουργούνται σε μια επανάληψη αντιστοιχούν τον αριθμό του αρχικού πληθυσμού. Οπότε είναι σαν να διπλασιάζεται ο πληθυσμός. Αν ο πληθυσμός παρέμενε έτσι και αυξάνονταν συνεχώς τότε θα οδηγούσε σε πολύ μεγαλύτερο χρόνο εκτέλεσης και αλλοιώσεις των λύσεων, διότι στον πληθυσμό θα παραμένουν και οι πιο αδύναμες λύσεις. Στόχος είναι να η προσπάθεια να κρατηθούν οι πιο δυνατοί συνδυασμοί, όμως όπως θα φανεί και παρακάτω, μια μεθοδολογία του τύπου να επιβιώνουν οι πιο δυνατοί θα οδηγούσε σε αλλοίωση της ποικιλομορφίας. Οι υλοποιήσεις των επίλογων επιβίωσης μπορούν να κατηγοριοποιηθούν ως εξής.

Figure 13.Survivor Selection Methods



Age Based:

Σε αυτή τη περίπτωση το ποσό δυνατή μπορεί να είναι μια λύση δεν έχει καμία σημασία, καθώς το **Fitness** παραμελείτε εντελώς. Το μόνο κριτήριο στην επιλογή είναι η ηλικία ενός χρωμοσώματος, δηλαδή πόσες γενιές έχουν περάσει από τότε που δημιουργήθηκε. Η κάθε οντότητα είναι ελεύθερη να αναπαραγάγει μετά από πολλές γενιές αλλά μέχρι ένα σημείο. Το σημείο ορίζεται αναλόγως τους στόχους του προγράμματος. Η οντότητα θα διωχτεί από τον πληθυσμό όταν η ηλικία του ξεπεράσει το όριο.

Fitness Based:

Η λογική είναι ιδιά και με την επιλογή γονέων για αναπαραγωγή. Τα παιδιά τείνουν να αντικαθιστούν τις πιο αδύναμες οντότητες. Η επιλογή μπορεί να γίνει με τρόπους που έχουν ήδη αναδειχτεί, όπως η ρουλέτα και το τουρνουά. Σε αυτές τις περιπτώσεις οι πιο δυνατές οντότητες έχουν περισσότερες πιθανότητες να περάσουν στην επόμενη γενιά, αλλά και οι πιο αδύναμες. Σε αντίθεση με την άλλη επιλογή, τον ελιτισμό, όπου εκεί οι οντότητες που πηγαίνουν στην επόμενη γενιά είναι μόνο οι πιο δυνατές. Μετά τη δημιουργία των παιδιών αν ο πληθυσμός μας για παράδειγμα αποτελείται από 20 οντότητες, τότε στην επόμενη γενιά θα πάνε οι πρώτες 10 μετρά από ταξινόμηση με βάση το **Fitness**. Το θετικό είναι πως ο αλγόριθμος φτάνει πολύ γρηγορά σε πολύ καλά αποτελέσματα, το οποίο είναι ο στόχος θεωρητικά, οι καλύτερες λύσεις σε όσο λιγότερες γενιές. Όμως αυτή η προσέγγιση έχει ως αρνητικό ότι τα γονίδια των καλυτέρων λύσεων μπορεί να εξαπλωθούν πολύ γρηγορά στον πληθυσμό με αποτέλεσμα να χαθεί η ποικιλομορφία και να εγκλωβιστούμε σε μια τοπική λύση.

Genetic Algorithm for I/O Pad Planning

Εισαγωγή

Σε αυτό το μέρος θα παρουσιαστεί η μεθοδολογία και η υλοποίηση του γενετικού αλγορίθμου για να ελαχιστοποιηθεί το συνολικό μήκος καλωδίου. Θα συνεχιστεί όπως και στο προηγούμενο κεφάλαιο, η επεξήγηση του κάθε βήματος στον αλγόριθμο, ενώ παράλληλα θα αναλύονται έννοιες, οι οποίες είναι απαραίτητες για την κατανόηση τους.

- Το χρωμόσωμα θα είναι ένας πίνακας ακέραιων που το μέγεθος του θα είναι το συνολικό μήκος της ταμπλέτας, δηλαδή το άθροισμα του μήκους της κάθε μιας από τις τέσσερις πλευρές του.
- Τα γονίδια εδώ θα συμβολίζουν τα **I/O Pads**, η μεταβλητή τους θα είναι ένας ακέραιος με τον αριθμό κάθε **I/O Pad.**
- Το **index** μια θέσης του πίνακα θα αντιπροσωπεύει μια τοποθεσία πάνω στην ταμπλέτα. Αυτή η τοποθεσία έχει να κάνει σε πιο συγκεκριμένο σημείο από τις τέσσερις πλευρές θα τοποθετηθεί το **I/O Pad**.
- Η τιμή κάθε θέσης του πίνακα θα μπορεί να πάρει δυο τιμές. Η πρώτη είναι το -1 που μας δείχνει ότι η συγκεκριμένη θέση είναι κενή και η δεύτερη παίρνει έναν ακέραιο που είναι το **I/O Pad**.

Κωδικοποίηση

Θα χρησιμοποιηθεί η κωδικοποίηση **Permutation**. Όπως αναφέρθηκε προηγουμένως, το χρωμόσωμα θα είναι μια σειρά ακέραιων αριθμών οπού κάθε θέση θα είναι μια τοποθεσία με τιμή το **I/O Pad**. Αν και ίσως θα μπορούσε να είναι μια μικρή παραλλαγή του **Permutation Encoding**, για το λόγο ότι ο πίνακας δεν αποτελείται μόνο από τα **I/O Pads**, αλλά και από τις κενές θέσεις. Η επιλογή αυτή επιτρέπει με τη χρήση μόνο ενός μονοδιάστατου πίνακα να υπάρχουν όσες πληροφορίες χρειάζονται. Επίσης, είναι πάρα πολύ χρήσιμη σε ζητήματα αναζήτησης, **Crossover**, **Mutation** διότι καθιστά αυτές τις διαδικασίες πολύ απλοϊκές στην υλοποίηση. Το μόνο αρνητικό αυτής της επιλογής θα μπορούσε να είναι η δυσκολία εύρεσης σε ποια πλευρά είναι το κάθε **I/O Pad**, όπως και που πρέπει να τοποθετηθεί. Η δυσκολία αυτή εμφανίστηκε στο κομμάτι που έπρεπε να υπολογιστεί το μήκος καλωδίου αλλά και στην οπτικοποίηση των αποτελεσμάτων. Για την επίλυση δημιουργήθηκε μια συνάρτηση με στόχο την εύρεση των συντεταγμένων X και Y που ονομάζεται **GetXY** (**Code 1.**), η οποία είναι η εξής: Η συνάρτηση έχει ως παράμετρο μια μεταβλητή, η οποία είναι μια οποιαδήποτε θέση στον πίνακα, αρά πρέπει να είναι από 0 μέχρι K, οπού K το μέγεθος του πίνακα. Αν θεωρηθεί πως ο πίνακας έχει μέγεθος 1000 και κάθε πλευρά αποτελείται από 250 θέσεις, τότε αν η θέση είναι μικρότερη του 250 σημαίνει πως βρίσκεται στην πάνω θέση, μικρότερη του 500 και μεγαλύτερη του 250 σημαίνει ότι βρίσκεται στην κάτω. Αντίστοιχα, μετά στην αριστερή και τέλος στη δεξιά.

Code 1. Function GetXY

```
def getXY(self, index):
        if index < X:
            coordinateX = index
            coordinateY = Y + 33
        elif index < 2 * X:
            coordinateX = index - X
            coordinateY = -33
        elif index < 2 * X + Y:
            coordinateY = index - 2 * X
            coordinateX = -33
        elif index < 2 * X + 2 * Y:
            coordinateY = index - (2 * X + Y)
            coordinateX = X + 32
        return coordinateX, coordinateY
```

Από μόνη της η συνάρτηση δεν απαιτεί κάποια υπολογιστική υψηλή ισχύ, όμως αν αναλογιστούμε ότι εκτελείται κάθε φορά που υπολογίζουμε το **Fitness**, τότε η αποφυγή της σιγουρά θα σήμαινε βελτίωση.

Αρχικοποίηση

Θα χρησιμοποιηθεί η **Random** επιλογή για την υλοποίηση του πληθυσμού. Εκτός από τον πρώτο **Individual** εκ του οποίου το χρωμόσωμα θα αποτελείται από τις αρχικές θέσεις των **I/O Pads** τα οποία θα παρθούν από το **Dataset**. Οι υπόλοιποι απλώς θα είναι το ίδιο αλλά με τυχαία σειρά τα γονίδια. Τέλος, όπως αναφέρθηκε προηγουμένως, καθοριστικό ρολό στην αρχικοποίηση παίζει η επιλογή για το μέγεθος του πληθυσμού. Στην περίπτωση αυτή ο πληθυσμός αποτελείται από 100 **Individuals**. Ο αριθμός αυτός επιλέχθηκε με βάση ότι δεν είναι πολύ μικρός, οπότε θα αποφευχθεί μια πρόωρη σύγκλιση, και ούτε τόσο μεγάλος για να χρειαστεί υπερβολικό χρόνο ο αλγόριθμος για να εκτελεστεί.

Fitness

Το **Fitness** είναι το μέτρο με το οποίο αξιολογούμε την κάθε λύση-χρωμόσωμα και το κατά πόσο ισχυρή είναι. Στην περίπτωση αυτή το **Fitness** θα αντιστοιχεί στο συνολικό μήκος καλωδίου, το οποίο στόχος μας είναι να το ελαχιστοποιήσουμε, δηλαδή να βρούμε την τοποθεσία των **I/O Pads** στην οποία το μήκος είναι το μικρότερο. Η αξιολόγηση γίνεται σε δυο περιπτώσεις στο πρόγραμμα:

- Όταν δημιουργείται ο πληθυσμός.
- Όταν αναπαράγεται ένα παιδί, δηλαδή έπειτα από το **Crossover** και το **Mutation**.

Η λογική στον υπολογισμό του μήκους καλωδίου είναι η εξής: ένα δίκτυο έχει στον περίγυρο του το καλώδιο, το δίκτυο εμπεριέχει μέσα του **Cells** και **I/O Pads,** οπότε στόχος μας είναι να βρούμε τα άκρα, εννοώντας τις γωνίες του καλωδίου. Αυτό θα γίνει ψάχνοντας

πιο **Cell** ή **I/O Pad** βρίσκεται στην πιο κάτω αριστερά, πάνω αριστερά, κάτω δεξιά και πάνω δεξιά γωνιά. Αφού εντοπίσουμε τις συντεταγμένες αυτών των γωνιών, είναι εύκολο να υπολογίσουμε την περίμετρο. Αυτό πρέπει να γίνει για κάθε δίκτυο ξεχωριστά, και στο τέλος το άθροισμα όλων των μηκών καλωδίων που υπολογίσαμε για κάθε δίκτυο, θα αποτελεί το **Fitness** του χρωμοσώματος. Παρακάτω στον **Code 2.** Παρουσιάζεται η προγραμματιστική υλοποίηση.

Code 2. Fitness Function

```
def Fitness(self):
    for i in range(len(sortedArr)):
            Xmax, Xmin, Ymax, Ymin = -500, 20000, -500, 20000

            port = Portals[sortedArr[i][0]]
            Xi, Yi = self.getXY(sortedArr[i][2])

            if Xi < Xmin:
                Xmin = Xi
            if Xi > Xmax:
                Xmax = Xi
            if Yi < Ymin:
                Ymin = Yi
            if Yi > Ymax:
                Ymax = Yi

            if port.x < Xmin:
                Xmin = port.x
            if port.x + port.mikos > Xmax:
                Xmax = port.x + port.mikos
            if port.y < Ymin:
                Ymin = port.y
            if port.y + port.ipsos > Ymax:
                Ymax = port.y + port.ipsos

            FitnessScore = FitnessScore + (Xmax - Xmin) + (Ymax - Ymin)
        self.Fitness_Score = FitnessScore
    return FitnessScore
```

Crossover

Το πρώτο βήμα στο **Crossover** είναι η επιλογή γονέων και έπειτα η επιλογή τελεστή **Crossover** για αναπαραγωγή, προηγουμένως σημειώθηκαν διάφοροι μέθοδοι τους οποίους μπορεί να χρησιμοποιήσει ένας Γενετικός Αλγόριθμος. Το πιο βασικό στοιχείο στην επιλογή γονέων όπως και συγκεκριμένου τελεστή **Crossover** είναι κατά ποσό αξίζει να ρισκάρουμε να εξερευνηθούν πολλές λύσεις, και αντίθετα κατά ποσό υπάρχει ο φόβος ότι σε μια περίπτωση, μια σε υψηλό ποσοστό πολυμορφία μπορεί να καταλήξει σε **random** επιλογές ή να χρειαστεί πολύ χρόνος για την εκτέλεση. Εδώ θα χρησιμοποιηθεί το **Two-Point Crossover** ως τη μέση τομή αναμεσά στα δυο προ αναφερθέντα κριτήρια και τη ρουλέτα για να επιλέξουμε γονείς.

Αρχικά, όπως παρουσιάζεται στο **Code 4** για την υλοποίηση της ρουλέτας πρέπει να γίνουν οι εξής διεργασίες:

- Υπολογισμός του συνολικού μήκους καλωδίου όπως φαίνεται παρακάτω στο **Code 3.**
- Παραγωγή των πιθανοτήτων για κάθε **individual**. Τα αποτελέσματα θα αποθηκευτούν σε ένα πίνακα.
- Τον οποίο χρησιμοποιώντας τον ως παράμετρο στην συνάρτηση **NumPy.random.choice** θα επιστρέψει έναν γονέα από τον πληθυσμό. Προφανώς, η συνάρτηση θα χρησιμοποιηθεί δυο φορές, διότι χρειάζονται δυο γονείς για την αναπαραγωγή.

Code 3. SumOfFitness Function

```
def SumOfFitness(self):
    summ = 0
    for sol in self.Population:
        summ = summ + sol.Fitness_Score
    return summ
```

Code 4. RouletteWheel Function

```
def RouletteWheel(self):

    summ = self.SumOfFitness()
    probabilities = [a.Fitness_Score / summ for a in self.Population]
    return np.random.choice(self.Population, p=probabilities)
```

Στη συνέχεια, για το **Two Point Crossover** θα αναλυθεί βήμα-βήμα τη μεθοδολογία υλοποίησης του. Αυτό θα γίνει διότι αποτελεί το κομμάτι στον αλγόριθμο που διακατέχει το περισσότερο υπολογιστικό βάρος, λόγω ότι απαιτεί συχνές αναζητήσεις στο χρωμόσωμα, αναθέσεις σε μεταβλητές και χρήση αρκετών συναρτήσεων. Επίσης είναι και το κομμάτι που ως επι τον πλείστων εξαρτάται η απόδοση του αλγορίθμου.

1ον. Με μια **Random** συνάρτηση θα παραχθούν δυο ακέραιοι 0<X1,X2<N οπού N το μέγεθος του χρωμοσώματος και X1,X2 οι μεταβλητές. Αυτές οι δυο μεταβλητές θα είναι τα σημεία στον πίνακα που θα γίνουν τα κοψίματα. Για την ευκολότερη επεξήγηση ας θεωρηθεί ότι X1>X2.

2ο.Με τη χρήση της συνάρτησης **Roulette Wheel** επιλέγονται δυο γονείς.

3ο.Έπειτα από τη δημιουργία δυο παιδιών, αντιγράφονται τα γονίδια του 1ο γονέα από X1 μέχρι X2 στο 2ο παιδί, και αντίστοιχα τα γονίδια X1 με X2 στο 1ο παιδί από τον 2ο γονέα.

4ο.Στη συνέχεια, αντιγράφονται τα γονίδια από 0-X1 και X1- N του 1ου γονέα στο 1ο παιδί. Εδώ χρειάζεται προσοχή να μην αντιγράφει κάποιο γονίδιο δεύτερη φορά στο ίδιο παιδί, το οποίο μπορεί να αποφευχθεί ευκολά με μια αναζήτηση πριν την αντιγραφή. Αντίστοιχα, η ιδιά διαδικασία για τον 2ο γονέα και το 2ο παιδί.

5ο.Στο τελικό βήμα, θεωρητικά αναζητούνται ποια γονίδια απομένουν να αντιγράφουν. Τα γονίδια που λείπουν είναι αυτά οπού τα γονίδια X1-X2 του πρώτο γονέα είναι κοινά με τα γονίδια 0-X1 και X2-N του 2ου γονέα. Έπειτα θα προθέτονταν στις κενές θέσεις τα γονίδια αυτά στη σειρά που εμφανίζονται στο γονέα 1 και η διαδικασία θα τέλειωνε. Αυτό θα γινόταν αν η κωδικοποίηση μας είχε την καθαρή μορφή του **Permutation Encoding**, όμως όπως αναφέρθηκε προηγουμένως, στην περίπτωση αυτή υπάρχει μια τροποποίηση, η οποία είναι τα κενά, δηλαδή οι τιμές με -1. Για αυτό αν χρησιμοποιούνταν η προηγουμένη μεθοδολογία, θα υπήρχε η κατάληξη να μπαίνανε όλα τα γονίδια του 5ου βήματος στην αρχή του χρωμοσώματος, κάτι που δεν θα βοηθούσε καθόλου. Για αυτό ως τροποποίηση στον αλγόριθμο θα αναπαραχθεί μια κενή θέση τυχαία στον πίνακα και θα προσθέσουμε σε αυτή το γονίδιο.

Code 5. Crossover Function

```
def Crossover(self, generation):

    x1 = random.randrange(len(self.Population[0].chromosome))
    x2 = random.randrange(len(self.Population[0].chromosome))

    if x1 > x2:
        x1, x2 = x2, x1

    child1 = Individual(0, 0)
    child2 = Individual(0, 0)
    parent1 = self.RouletteWheel()
    parent2 = self.RouletteWheel()
    child1.chromosome[x1:x2] = parent2.chromosome[x1:x2]
    child2.chromosome[x1:x2] = parent1.chromosome[x1:x2]
    kke = np.where(parent1.chromosome != -1)
    kke2 = np.where(parent2.chromosome != -1)

    for k in kke[0]:
        if k <= x1 or k > x2:
            if parent1.chromosome[k] not in child1.chromosome[x1:x2]:
                child1.chromosome[k] = parent1.chromosome[k]
    for k in kke2[0]:
        if k <= x1 or k > x2:
            if parent2.chromosome[k] not in child2.chromosome[x1:x2]:
                child2.chromosome[k] = parent2.chromosome[k]

    for k in kke[0]:
        if k > x1 and k <= x2:
            if parent1.chromosome[k] not in child1.chromosome:
                index = np.where(child1.chromosome == -1)
                kk = random.choice(index[0])
                child1.chromosome[kk] = parent1.chromosome[k]
    for k in kke2[0]:
        if k > x1 and k <= x2:
            if parent2.chromosome[k] not in child2.chromosome:
                index = np.where(child1.chromosome == -1)
                kk = random.choice(index[0])
                child2.chromosome[kk] = parent2.chromosome[k]

    return child1, child2
```

Mutation

Για το **Mutation** θα χρησιμοποιήσουμε την εναλλαγή **bits** ως μέθοδο. Θα επιλέξουμε δυο γονίδια και θα αλλάξουμε τη θέση μεταξύ τους. Η πιθανότητα**(Mutation Probability)** παίζει καθοριστικό ρολό. Είναι η πιθανότητα να διεξαχθεί ο τελεστής. Στην περίπτωση μας ο τελεστής εφαρμόζεται σε κάθε παιδί έπειτα από τη δημιουργία του. Δηλαδή κάθε οντότητα στον πληθυσμό έχει μια φορά την πιθανότητα να μεταλλαχθεί. Για αυτό και η πιθανότητα έχει οριστεί αρκετά υψηλοτέρα σε σχέση το σύνηθες.

Code 6. Mutation Function

```
def Mutation(self, child):

        if random.random() < 0.5:
            lista = np.where(child.chromosome != -1)

            x1 = random.choice(lista[0])
            x2 = random.choice(lista[0])

            child.chromosome[x1], child.chromosome[x2] = (
                child.chromosome[x2],
                child.chromosome[x1],
            )

        child.Fitness()
```

Survival Selection

Επιλέχθηκε η ελιτιστική μέθοδος για την επιλογή των χρωμοσωμάτων τα οποία θα προχωρήσουν στην επόμενη γενιά για αναπαραγωγή. Δηλαδή κρατάμε μόνο τις καλύτερες λύσεις. Στην υλοποίηση έπειτα από τη δημιουργία παιδιών ο πληθυσμός είναι πάντα διπλάσιος, εννοώντας ότι ο αριθμός των παιδιών αντιστοιχεί στον αριθμό των γονέων. Οπότε στο τέλος απλώς κρατάμε τις N καλύτερες λύσεις, οπού N το αρχικό μέγεθος του πληθυσμού.

Κριτήριο Τερματισμού

Η αρχική ιδέα ήταν να θέσω ένα μέγιστο σύνολο επαναλήψεων του ύψους 100000. Παρατηρήθηκε ότι από ένα σημείο και μετρά στον πληθυσμό είχε εξαφανιστεί η πολυμορφία, δηλαδή όλες οι λύσεις ήταν παρόμοιες μεταξύ τους το οποίο κατέληγε να μην υπάρχει καμία βελτίωση ούτε κατά μονάδα. Για αυτό και τέθηκε ένας όρος στις επαναλήψεις από το σημείο και μετέπειτα δεν υπήρχε καμία βελτίωση. Δηλαδή αν σε 500 επαναλήψεις η καλύτερη λύση παρέμενε ιδιά, τότε ο αλγόριθμος θα τερμάτιζε.

Multiprocessing

Η πολυεπεξεργασία (**Multiprocessing**) είναι η χρήση δυο ή παραπάνω επεξεργαστών σε έναν υπολογιστή. Ο ορισμός μπορεί και να διαφέρει ανάλογα με το περιεχόμενο, αλλά γενικά αναφέρεται στη δυνατότητα του υπολογιστή να μπορεί να καταμερίσει εργασίες στους επεξεργαστές.

Διεργασία(Process):

Η διεργασία είναι μια συλλογή οδηγιών που εκτελούνται από τον επεξεργαστή του υπολογιστή. Η εκτέλεση μιας διεργασίας πρέπει να προχωράει με μεθοδικό τρόπο. Για παράδειγμα, ένα προγραμματιστικό πρόγραμμα είναι γραμμένο σε ένα αρχείο κειμένου, αλλά αφού εκτελεστεί, μετατρέπεται σε μια διεργασία εκτελώντας όλα τα πράγματα που είναι γραμμένα στο πρόγραμμα με σειρά. Ένας απλός υπολογιστής τρέχει ένα σύνολο διεργασιών συνεχώς για να βοηθήσουν στη διαχείριση του λογισμικού, του **hardware** και των εγκατεστημένων εφαρμογών.

Μειονεκτήματα Multiprocessing:

- Κόστος: Προφανώς ένα σύστημα με πολλούς επεξεργαστές είναι ακριβότερο σε σχέση με τα συστήματα που έχουν ένα πυρήνα.
- **Deadlocks**: Είναι η περίπτωση που μια ή πολλές διεργασίες δεν μπορούν να συνεχίσουν την εκτέλεση τους επειδή περιμένουν να τελειώσει κάποια άλλη.
- Παραπάνω μνήμη: Ένα λειτουργικό απαιτεί περισσότερο μνήμη όσο περισσότεροι είναι και οι επεξεργαστές.

Πλεονεκτήματα Multiprocessing

- Ταχύτητα : Όσο περισσότεροι οι επεξεργαστές, συνήθως θα είναι και πιο γρήγορο το λειτουργικό σύστημα
- Σταθερότητα : Στην περίπτωση που υπάρξει κάποιο πρόβλημα με έναν από τους επεξεργαστές, ένας άλλος μπορεί να πάρει την θέση του και να τον καλύψει.

Multithreading vs Multiprocessing

Η πρωταρχική και κύρια διάφορά τους είναι να καταλάβουμε το επίπεδο αυτών των δυο εννοιών. Μια διεργασία έχει ένα κύριο νήμα, και μπορεί να έχει και ακόμα περισσότερα, και ότι μια διεργασία μπορεί να αποκτήσει διεργασίες παιδιά. Εν κατακλείδι, το νήμα ανήκει στη διεργασία.

Η επόμενη διάφορά τους έχει να κάνει με τον τρόπο με τον οποίο έχουν πρόσβαση στην κατάσταση του συστήματος. Η κατάσταση αναφέρεται στις ρυθμίσεις ή κατάσταση ενός συστήματος, προγράμματος, ή συσκευής μια συγκεκριμένη στιγμή. Τα νήματα μπορούν να μοιραστούν μνήμη μέσα σε μια διεργασία. Αυτό σημαίνει πως συναρτήσεις που εκτελούνται σε καινούρια νήματα μπορούν να έχουν πρόσβαση σε δεδομένα και στην κατάσταση του συστήματος. Αυτές πχ μπορεί να είναι **Global** μεταβλητές ή δεδομένα που μεταφέρθηκαν μέσω παραμέτρων της συνάρτησης. Απεναντίας, οι διεργασίες δεν μοιράζονται κοινή μνήμη όπως τα νήματα. Για αυτό και η κατάσταση πρέπει με σειρά να μεταδοθεί μεταξύ των

διεργασιών, η διαδικασία ονομάζεται **inter-process communication**. Παρόλο που γίνεται πίσω από τις σκηνές, έχει όρια σε τι δεδομένα και καταστάσεις μπορούν να μεταδοθούν και πέρα από αυτό, επίσης προσθέτει παραπάνω βάρος (**overhead**) στο πρόγραμμα. Τυπικά για να μεταφερθούν δεδομένα μεταξύ διεργασιών απαιτούνται ειδικοί μηχανισμοί όπως η χρήση του **multiprocessing.Pipe** ή **multiprocessing.queue**. Κλείνοντας, η μεταφορά δεδομένων μεταξύ νημάτων είναι εύκολο και ελαφρύ για το σύστημα, σε αντίθεση με τις διεργασίες που είναι δυσκολότερο και ποιο απαιτητικό για το σύστημα.

Ο συγχρονισμός που βασίζεται στα σήματα υποστηρίζει παραλληλισμό, σε αντίθεση με τον συγχρονισμό διεργασιών οπού υποστηρίζεται πλήρης παραλληλισμός. Τα πολλαπλά νήματα υπόκεινται στον **Global interpreter lock(GIL),** ενώ οι διεργασίες παιδιά, όχι. Το **GIL** είναι ένα προγραμματιστικό μοτίβο στον διερμηνέα της **Python.** Είναι σαν μια κλειδαριά που χρησιμοποιεί συγχρονισμό για να σιγουρέψει πως μόνο ένα νήμα εκτελεί κάτι τη στιγμή σε μια διεργασία. Αυτό σημαίνει πως παρόλο που μπορούμε να έχουμε πολλά νήματα σε ένα πρόγραμμα, πάντα εκτελείται μόνο ένα τη φορά. Αυτό όμως δεν συμβαίνει για τις διεργασίες, οι οποίες μπορούν να εκτελούνται εντελώς παράλληλα.

Multiprocessing στον Αλγόριθμο

Όπως είπαμε στην εισαγωγή, ο γενετικός ήταν το αποτέλεσμα της ανάγκης για πιο αποδοτικούς αλγορίθμους, ήρθε στο προσκήνιο όταν πλέον οι παραδοσιακοί μέθοδοι δεν αποδίδαν αρκετά. Κυρίως εφαρμόζεται σε προβλήματα με πολλά δεδομένα, και για αυτό ο χρόνος εκτέλεσης παίζει καθοριστικό ρολό. Για παράδειγμα, στην επιλογή του μεγέθους του πληθυσμού. Στόχος είναι η καλύτερη λύση, σε όσες λιγότερες γενιές, δηλαδή σε όσο λιγότερο χρόνο γίνεται. Ένα μεγάλο πλεονέκτημα του γενετικού αλγορίθμου είναι πως κατά την εκτέλεση του μπορεί να εφαρμοστεί παραλληλισμός, δηλαδή σημεία του αλγορίθμου που εκτελούνται ταυτόχρονα. Στην **Python** αυτό μπορεί να επιτευχθεί με μια βιβλιοθήκη που ονομάζεται **multiprocessing**. Έναυσμα για την αναζήτηση αυτής της μεθόδου ήταν το μεγάλο χρονικό διάστημα που χρειαζόταν για την εκτέλεση ο γενετικός. Σε μια απλή εκτέλεση το σύστημα θα χρησιμοποιούσε μόνο έναν επεξεργαστή. Με τη χρήση του **multiprocessing** έχουμε τη δυνατότητα να αξιοποιήσουμε όλους τους επεξεργαστές του υπολογιστή μας.

Στον γενετικό έχουμε τη δυνατότητα να χρησιμοποιήσουμε το **multiprocessing** για ένα μεγάλο κομμάτι του αλγορίθμου. Εμείς θα εστιάσουμε σε αυτά που σε μια επανάληψη εκτελούνται παραπάνω από μια φορά. Για παράδειγμα, η αρχικοποίηση του πληθυσμού και η διαδικασία επιλογής γονέων για επιβίωση είναι διεργασίες που εκτελούνται μια φορά, οπότε δεν υπάρχει λόγος να προστεθούν στο κομμάτι του **multiprocessing**. Απεναντίας, εμείς θα προσθέσουμε στη συνάρτηση για παραλληλοποίηση το **Crossover**, επειδή είναι αυτό που παίρνει τον περισσότερο χρόνο, αρά έχει και την μεγαλύτερη ανάγκη για παραλληλοποίηση και επειδή σε κάθε επανάληψη εκτελείται η συνάρτηση N/2 φορές, οπού N το μέγεθος του πληθυσμού. Είναι N/2 επειδή κάθε **Crossover** παράγει δυο παιδιά. Έπειτα, το **Mutation**, το οποίο θα μπορούσαμε και να το αποφύγουμε, το οποίο μεν εκτελείται στο μέγιστο N φορές, αλλά ο χρόνος εκτέλεσης του είναι ασήμαντος, διότι απλώς εμπεριέχει μόνο αναθέσεις κυρίως. Ως υπενθύμιση, εκτελείται στο μέγιστο N φορές επειδή στο **Mutation** υπάρχει η πιθανότητα

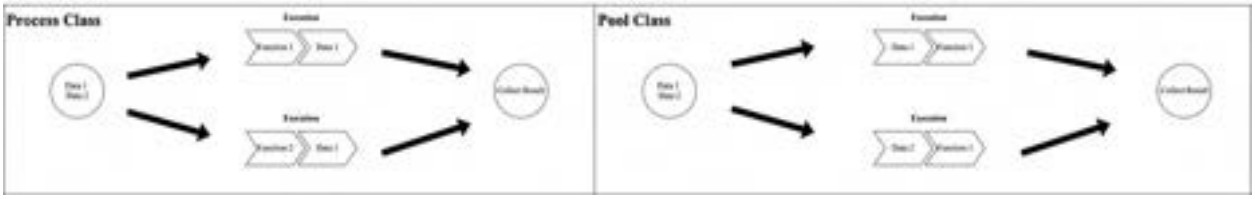
να εκτελεστεί. Τέλος, θα προστεθεί και η συνάρτηση **Fitness**, η οποία εκτελείται Ν φορές και αυτή, και απατή αρκετούς υπολογισμούς ώστε να φτάσει στο αποτέλεσμα της. Εν κατακλείδι, αυτές θα είναι οι τρεις κυρίες συναρτήσεις τις οποίες θα παραλληλοποιήσουμε κατά την εκτέλεση. Προσοχή υφίσταται στο γεγονός ότι οι διεργασίες δεν μοιράζονται κοινή μήνη, οπότε είναι βασικό στην υλοποίηση να έχουμε ανεξαρτητοποιήσει την κάθε συνάρτηση και την κάθε οντότητα μεταξύ τους.

Στην **Python** χρησιμοποιείται η βιβλιοθήκη **multiprocessing** για παραλληλισμό. Μας παρέχει δυο κλάσεις με τις οποίες μπορούμε να το επιτύχουμε αυτό, την **Process** και την **Pool**.

Process Class: Όταν έχουμε ένα σύνολο από διαφορετικά πράγματα τα οποία θέλουμε να εκτελέσουμε παράλληλα, τότε χρησιμοποιούμε αυτή τη κλάση. Συνήθως χρησιμοποιείται για προγραμματισμό βασισμένο σε συναρτήσεις οπού κάθε συνάρτηση έχει μια αρμοδιότητα. Κάθε διεργασία πρέπει να δημιουργείται ξεχωριστά κάτι που την κάνει διαχειρίσιμη από το πρόγραμμα-χρήστη.

Pool Class: Είναι μια προγραμματιστική πατέντα για αυτόματη διαχείριση ενός συνόλου από διεργασίες. Αυτόματα καταμερίζει τις δουλείες στους διαθέσιμους επεξεργαστές χρησιμοποιώντας μια σχεδίαση **FIFO**. Επιλέγεται όταν έχουμε παρόμοια καθήκοντα που θέλουμε να εκτελεστούν παράλληλα. Κυρίως, για παραλληλισμό που έχει να κάνει με δεδομένα με την ίδια συνάρτησης να εκτελείται παράλληλα πολλές φορές με διαφορετικό **input.**

Figure 14. Process and Pool Class



Παρακάτω θα αναδείξω την υλοποίηση του παραλληλισμού στην **Python** και θα εξηγηθεί βήμα η διαδικασία. Στον υπόλοιπο κώδικα για χάρη της παραλληλοποίησης υπήρξε μια παραλλαγή. Όπως θα δούμε με το **Pool.map** πρέπει να εκτελούμε μια συνάρτησης, και για αυτό μέσα στη συνάρτησης του **Crossover** αφού τελειώσει θα καλέσει τη συνάρτηση **Mutation**, και αυτή με τη σειρά της τη **Fitness**. Στην πράξη δεν αλλάζει τίποτα, αλλά μόνο στο οπτικό κομμάτι μπορεί να φανεί περίεργο. Η άλλη λύση θα ήταν να εκτελούμε ξεχωριστά τις τρεις αυτές συναρτήσεις για παραλληλισμό, το μόνο που θα άλλαζε είναι ότι θα βάραινε το πρόγραμμα και θα γινόταν ίσως λίγο πιο δυσνόητο, ενώ τώρα όλο το κομμάτι του παραλληλισμού γίνεται σε μια συνάρτησης που βελτιώνει τον χρόνο.

Code 7. Multiprocessing

```
1 pool = multiprocessing.Pool()
2 Children = pool.map(GA.Crossover, range(int(len(GA.Population) / 2)))
3 pool.close()
```

Αρχικά, δημιουργούμε ένα Instance της κλάσης **Pool**, η οποία έχεις τέσσερις προαιρετικούς παραμέτρους.

Code 8. Class multiprocessing.Pool

```
class multiprocessing.Pool(
                [processes[,
                initializer[,
                initargs[,
                maxtasksperchild]]]]
                )
```

- **Processes**: Διαλέγουμε τον αριθμό των επεξεργαστών που θέλουμε να χρησιμοποιήσουμε, αν η τιμή είναι **None** τότε παίρνει ως παράμετρο το αποτέλεσμα του **cpu_count()**.
- **Initializer**: Παίρνει ως όρισμα μια συνάρτησης που θα εκτελείται και αρχικοποιεί τις δουλείες των ‘’εργατών’’ πριν ξεκινήσουν.
- **Initargs**: Τα ορίσματα της παραπάνω συνάρτησης.
- **Maxtaskperchild**: Επιλέγει τον αριθμό που μια διεργασία θα εκτελέσει πριν πάρει τη θέση της κάποια άλλη, αν δεν του δώσουμε τιμή τότε η κάθε διεργασία θα παραμείνει μέχρι το τέλος.

Στην περίπτωση μας δεν έχουμε ανάγκη κάποια από τις 4, οπότε θα είναι όλες κενές. Έπειτα, με την συνάρτησης **map** της κλάσης **Pool** θα ξεκινήσουμε την εκτέλεση των διεργασιών. Η συνάρτησης **Pool.map** έχει σύνταξη ως εξής.

Code 9. Function map

```
map(func, iterable[, chunksize])
```

- **Func**: Το όνομα της συνάρτησης που θέλουμε να εκτελεστεί
- **Iterable**: Εδώ δίνουμε τα ορίσματα στη συνάρτησης που πρέπει να είναι σε μορφή **Iterable**, δηλαδή πίνακα, λίστα, σετ κλπ. Π.Χ. αν στείλουμε έναν πίνακα ακέραιων που θα έχει μέγεθος 10, είναι σαν να λεμέ εκτέλεσε τον αλγόριθμο 10 φορές για καφέ διαφορετικό αριθμό μέσα στον πίνακα ως όρισμα. Ετήσιο πίνακας διαιρείται σε όσα μέρη είναι οι επεξεργαστές και ξεκινάει την εκτέλεση.
- **Chunksize**: Ο αριθμός των ορισμάτων που θα δοθεί σε κάθε επεξεργαστή ο οποίος είναι προαιρετικός.

Η συνάρτηση **Pool.map** είναι παρόμοια με τη συνάρτηση **Python map().** Σταματάει την κύρια διεργασία μέχρι να τελειώσουν και οι υπόλοιπες. Επίσης, δέχεται μόνο ένα όρισμα και αυτό πρέπει να είναι **iterable**. Τέλος, τα αποτελέσματα είναι ταξινομημένα.

Τελευταία συνάρτηση είναι το **Pool.close** το οποίο καλείται όταν έχουν τελειώσει οι διεργασίες και δεν υπάρχει κάτι άλλο να κάνουν.

Table 6. Multiprocessing Time Reduction

IBM	Cells	I/O Pads	Time
01	12505	246	33.43%
02	19341	259	16.70%
03	22852	283	34.61%
04	27219	287	18.95%
05	28145	1201	47.09%
06	32331	166	-1.313%
07	45368	287	14.48%
08	51022	286	21.93%
09	53109	286	17.31%
10	68684	744	37.59%

Αρχικά παρατηρούμε στο **Table 6** πως όσο περισσότερα τα **I/O Pad**, τόσο και μεγαλύτερη είναι και η ποσοστιαία μείωση του χρόνου. Ειδικά, στο IBM05 βλέπουμε οριακά μια μείωση που φτάνει στο μισό χρόνο εκτέλεσης. Αυτή η μείωση οφείλεται κυρίως στο **crossover** οπού γίνονται πολλές αναζητήσεις θέσεων και γονίδιων.

Έπειτα, σχετικά με τα κελιά, δεν υπάρχει κάποια ιδιαίτερη διάφορά διότι δεν συμβάλλουν και στον αλγόριθμο, πάρα μόνο στον υπολογισμό του συνολικού μήκους καλωδίου.

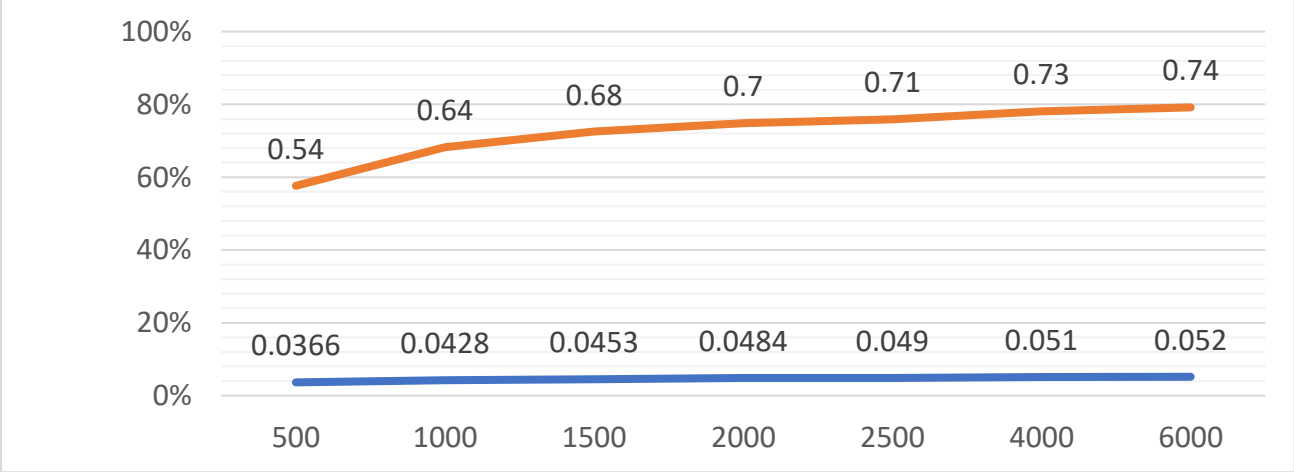
Παρατηρούμε στο **Table 6** ότι σε μια περίπτωση είχαμε αύξηση του χρόνου πάρα την αναμενομένη μείωση. Όπως έχουμε ήδη αναφέρει η ανάθεση και διαχείριση των διεργασιών δεν είναι μια εύκολη υπόθεση για το σύστημα, οπότε σε μια περίπτωση όπως η συνάρτηση να εκτελείται τόσο γρηγορά ώστε η εναλλαγή των επεξεργαστών να είναι εν τελεί πιο χρονοβόρα. Για αυτό και η μείωση παρατηρείται στο **Dataset** με τα λιγότερα **I/O Pads**, επειδή οι συναρτήσεις που παραλληλοποιούμαι εκτελούνται πάρα πολύ γρηγορά.

Τέλος, ένα πάρα πολύ σημαντικό σημείο στις δυο εκτελέσεις, με παραλληλοποίηση ή χωρίς είναι πως ο αλγόριθμος εκτελείται ξεχωριστά ο ένας από τον άλλον, και είναι γνωστό πως ο γενετικός επηρεάζεται σε πολύ μεγάλο κομμάτι από τις πιθανότητες και την τύχη. Οπότε πρέπει να λάβουμε υπόψιν αυτό στα αποτελέσματα, η επιρροή σιγουρά δεν διακατέχει κάποιο μεγάλο ποσοστό, διότι όπως και να έχει θα εκτελεστούν για τον ίδιο αριθμό επαναλήψεων, εντούτοις στη μια περίπτωση αν έβρισκε τυχαία μια πολύ καλή λύση και σταματούσε σε ένα τοπικό ακρότατο και αντίθετα η άλλη εκτέλεση αργούσε πολύ η σύγκλιση της, σιγουρά κάτι τέτοιο θα επηρέαζε.

Table 7. Wire Length Reduction

IBM	Cells	I/O Pads	Init_wl_impr	Init_io_wl_impr
01	12505	246	70.81%	10.26%
02	19341	259	74.65%	6.09%
03	22852	283	85.89%	6.95%
04	27219	287	74.59%	4.96%
05	28145	1201	79.53%	11.62%
06	32331	166	69.17%	2.86%
07	45368	287	64.46%	2.23%
08	51022	286	71.26%	2.38%
09	53109	286	84.34%	1.95%
10	68684	744	79.59%	2.48%

Figure 15. Results



Οι παρακάτω εικόνες αποτελούν αποτέλεσμα εκτέλεσης του Dataset IBM01.

Figure 16. Chromosome at 0 Generations

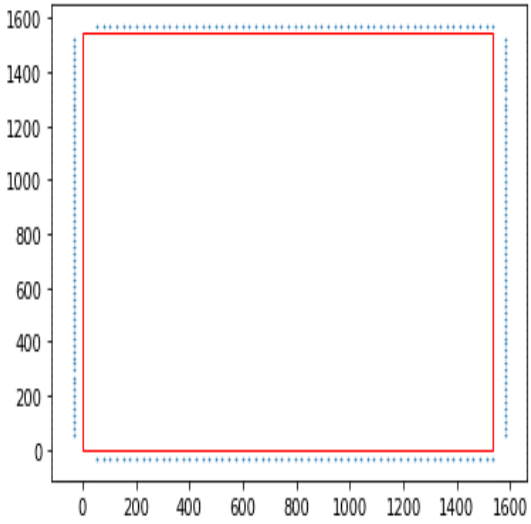


Figure 17. Chromosome at 500 Generations

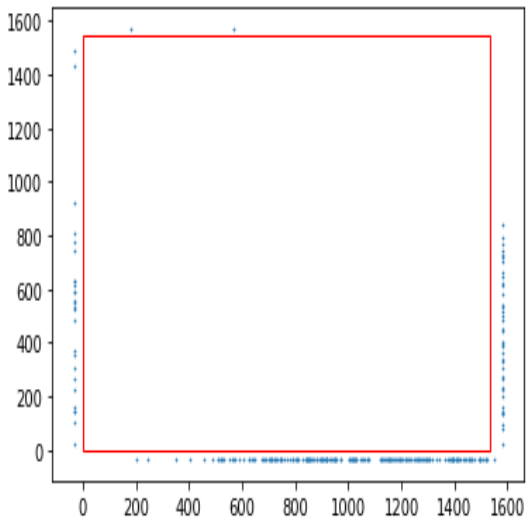


Figure 18.. Chromosome at 1000 Generations

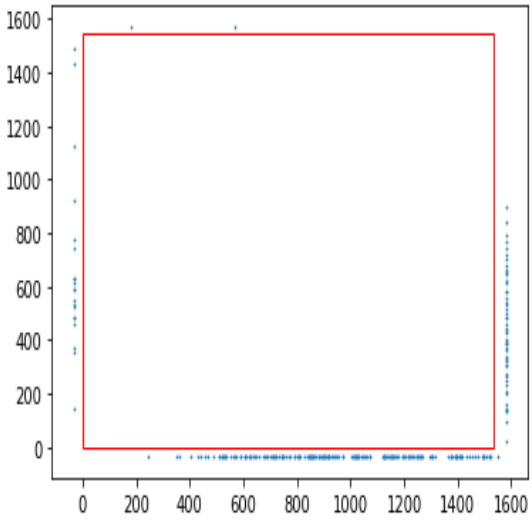
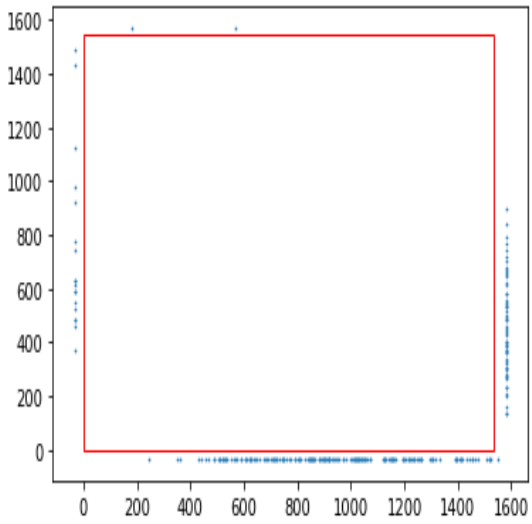


Figure 19. Chromosome at 1500 Generations



[1] Sait, S. M., & Youssef, H. (1999). *VLSI physical design automation: theory and practice* (Vol. 6). World Scientific.

[2] Kahng, A. B., Lienig, J., Markov, I. L., & Hu, J. (2011). *VLSI physical design: from graph partitioning to timing closure* (Vol. 312). Netherlands: Springer.

[3] Naveed A. Sherwani (1993). Algorithms for VLSI Physical Design Automation

[4] Schaller, R. R. (1997). Moore's law: past, present and future. IEEE spectrum, 34(6), 52-59.

[5] Qiu, Y., Xing, Y., Zheng, X., Gao, P., Cai, S., & Xiong, X. (2023). Progress of Placement Optimization for Accelerating VLSI Physical Design. *Electronics*, *12*(2), 337.

[6] Carr, J. (2014). An introduction to genetic algorithms. Senior Project, 1(40), 7.

[7] Barricelli, N. A. (1962). Numerical testing of evolution theories: part in theoretical introduction and basic tests. *Acta Biotheoretica*, *16*(1-2), 69-98.

[8] Holland, J. H. (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press.

[9] Kumar, A. (2013). Encoding schemes in genetic algorithm. *International Journal of Advanced Research in IT and Engineering*, *2*(3), 1-7.

[10] Goldberg, D. E., & Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. In Foundations of genetic algorithms (Vol. 1, pp. 69-93). Elsevier.

[11] Katoch, S., Chauhan, S. S., & Kumar, V. (2021). A review on genetic algorithm: past, present, and future. Multimedia tools and applications, 80, 8091-8126.

[12] Gavish, B., & Graves, S. C. (1978). The travelling salesman problem and related problems.

[13] Peterson, L. E. (2009). K-nearest neighbor. *Scholarpedia*, *4*(2), 1883.

[14] Umbarkar, A. J., & Sheth, P. D. (2015). Crossover operators in genetic algorithms: a review. ICTACT journal on soft computing, 6(1).

[15] Hasançebi, O., & Erbatur, F. (2000). Evaluation of crossover techniques in genetic algorithm based optimum structural design. Computers & Structures, 78(1-3), 435-448.

[16] Magalhaes-Mendes, J. (2013). A comparative study of crossover operators for genetic algorithms to solve the job shop scheduling problem. WSEAS transactions on computers, 12(4), 164-173.

[17] Spears, W. M., & Anand, V. (1991, October). A study of crossover operators in genetic programming. In International symposium on methodologies for intelligent systems (pp. 409-418). Berlin, Heidelberg: Springer Berlin Heidelberg.

[18] Καραγιαννίδη, Χ. Θ. (2007). Βελτιστοποίηση βιομηχανικής γραμμής παραγωγής με χρήση γενετικών αλγορίθμων.

[19] De Falco, I., Della Cioppa, A., & Tarantino, E. (2002). Mutation-based genetic algorithm: performance evaluation. Applied Soft Computing, 1(4), 285-299.

[20] Lambora, A., Gupta, K., & Chopra, K. (2019, February). Genetic algorithm-A literature review. In 2019 international conference on machine learning, big data, cloud and parallel computing (COMITCon) (pp. 380-384). IEEE.

[21] Schmitt, L. M. (2001). Theory of genetic algorithms. Theoretical Computer Science, 259(1-2), 1-61.