Πανεπιστήμιο Δυτικής Μακεδονίας

Πολυτεχνική Σχολή

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Διπλωματική Εργασία

Ανάπτυξη λογισμικού για τη δημιουργία σχηματικού σε αυτοματοποιημένη διαδικασία παραγωγής ψηφιακών κυκλωμάτων

Ευαγγελόπουλος Νικόλαος

Επιβλέπων Καθηγητής: Δρ. Μηνάς Δασυγένης (mdasyg(at)ieee.org) - Εργαστήριο Ψηφιακών Συστημάτων και Αρχιτεκτονικής Υπολογιστών

Ιούλιος 2020, Κοζάνη

Περίληψη

Η τεχνητή νοημοσύνη έχει μεταμορφώσει τις βιομηχανικές δραστηριότητες. Μία από τις πιο σημαντικές εφαρμογές της τεχνητής νοημοσύνης είναι η μείωση του υπολογιστικού κόστους βελτιστοποίησης.

Τα τελευταία χρόνια παρατηρείται η ολοένα και περισσότερο χρήση της τεχνητής νοημοσύνης για την επίλυση προβλημάτων με μεγάλη χρονική πολυπλοκότητα.

Με τον όρο βελτιστοποίηση εννοούμε ότι ελαχιστοποιούμε ή μεγιστοποιούμε κάποια μεγέθη ανάλογα με το πρόβλημα που έχουμε να αντιμετωπίσουμε.

Οι αλγόριθμοι βελτιστοποίησης ταξινομούνται σε δύο τύπους: ακριβείς αλγόριθμους και κατά προσέγγιση αλγόριθμους. Οι ακριβείς αλγόριθμοι μπορούν να βρουν με ακρίβεια βέλτιστες λύσεις, αλλά δεν ισχύουν για πολύπλοκα προβλήματα βελτιστοποίησης και ο χρόνος λύσης τους αυξάνεται εκθετικά σε τέτοια προβλήματα. Οι κατά προσέγγιση αλγόριθμοι μπορούν να βρουν σχεδόν βέλτιστες λύσεις για δύσκολα προβλήματα βελτιστοποίησης σε σύντομο χρονικό διάστημα.

Στην παρούσα διπλωματική εργασία λύνουμε κατά προσέγγιση ένα πρόβλημα βελτιστοποίησης υπό περιορισμούς με τη χρήση γενετικού αλγόριθμου. Πιο συγκεκριμένα, λύνουμε το πρόβλημα της δημιουργίας βελτιστοποιημένων σχηματικών ψηφιακών κυκλωμάτων. Δύο είναι τα βασικά υποπροβλήματα που έχουμε να αντιμετωπίσουμε, πρώτον είναι το πρόβλημα της τοποθέτησης αντικειμένων σε δισδιάστατο χώρο με βέλτιστο τρόπο και δεύτερον το πρόβλημα της εύρεσης συντομότερης διαδρομής για την διασύνδεση των στοιχείων μεταξύ τους.

Επιπλέον, θα χρησιμοποιήσουμε το εργαλείο Logisim που είναι ένα λογισμικό για σχεδίαση ψηφιακών κυκλωμάτων. Το Logisim είναι λογισμικό ανοιχτού κώδικα, χρησιμοποιείται από Πανεπιστημιακά τμήματα για την διεξαγωγή των μαθημάτων της Ψηφιακής Σχεδίασης και είναι διαθέσιμο για χρήση χωρίς χρέωση. Μπορεί κάποιος να το κατεβάσει ελεύθερα από http://www.cburch.com/logisim/download.html.

Για τον λόγο αυτό αναπτύχθηκε ένα εργαλείο γραμμής εντολών, που δέχεται σαν είσοδο ένα αρχείο (.dot) με τον ορισμό του ψηφιακού κυκλώματος και σαν έξοδο έχουμε ένα αρχείο (.circ) συμβατό με το Logisim, που περιέχει το σχηματικό του ψηφιακού κυκλώματος

μετά την βελτιστοποίηση. Η βελτιστοποίηση του σχηματικού μπορεί να επηρεαστεί σε μεγάλο βαθμό από τον χρήστη μέσω των παραμέτρων γραμμής εντολών του εργαλείου.

Επίσης, ένα μεγάλο και χρονοβόρο κομμάτι του εργαλείου ήταν η διασύνδεση του με τα κυκλωματικά στοιχεία του Logisim. Τα κυκλωματικά στοιχεία του Logisim με τα αντίστοιχα χαρακτηριστικά τους είναι αποδεκτά στο σύνολό τους με λίγες εξαιρέσεις σε ορισμένα χαρακτηριστικά.

Λέξεις κλειδιά: πρόβλημα βελτιστοποίησης, γενετικός αλγόριθμος, σχηματικό ψηφιακού κυκλώματος

Abstract

Software development for schematic design in an automated digital circuit production process

Artificial intelligence has transformed industrial activities. One of the important applications of artificial intelligence is the reduction of computing optimization costs.

In recent years, we have noticed the increasing use of artificial intelligence to solve problems with great time complexity.

By optimization we mean that we minimize or maximize some sizes depending on the problem we have to deal with.

Optimization algorithms are classified into two types: precise algorithms and approximate algorithms. Precise algorithms can accurately find optimal solutions, but they do not apply to complex optimization problems and their solution time increases exponentially in such problems. Approximate algorithms can find almost optimal solutions to difficult optimization problems in a short period of time.

In the present dissertation, we solve approximately a problem of optimization under constraints using genetic algorithm. More specifically, we solve the problem of creating optimized schematic digital circuits. There are two main sub-problems we have to deal with, firstly the problem of placing objects in two-dimensional space in an optimal way and secondly the problem of finding the shortest path to interconnect the components with each other.

In addition, we will use the Logisim tool which is software for designing digital circuits. Logisim is open source software, used by University departments to conduct Digital Design courses and is available for free use. You can download it for free from http://www.cburch.com/logisim/download.html.

For this reason, a command-line tool was developed, which accepts a file (.dot) as the input with the definition of the digital circuit, and as an output, we get a file (.circ) compatible with Logisim, which contains the schematic of the digital circuit after optimization. Schematic optimization can be greatly influenced by the user through the command line arguments.

Also, a big and time-consuming part of the tool creation was its interconnection with Logisim's circuit elements. Logisim's circuit elements with their corresponding characteristics are acceptable in their entirety with a few exceptions to certain features.

Keywords: optimization problem, genetic algorithm, digital circuit schematic

Δήλωση Πνευματικών Δικαιωμάτων

Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν. 1599/1986 και τα άρθρα 2,4,6παρ. 3 του Ν. 1256/1982, η παρούσα Διπλωματική Εργασία με τίτλο "Ανάπτυξη λογισμικού για τη δημιουργία σχηματικού σε αυτοματοποιημένη διαδικασία παραγωγής ψηφιακών κυκλωμάτων" καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας και αναφέρονται ρητώς μέσα στο κείμενο που συνοδεύουν, και η οποία έχει εκπονηθεί στο Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Πανεπιστημίου Δυτικής Μακεδονίας, υπό την επίβλεψη του μέλους του Τμήματος κ. Μηνά Δασυγένη αποτελεί αποκλειστικά προϊόν προσωπικής εργασίας και δεν προσβάλλει κάθε μορφής πνευματικά δικαιώματα τρίτων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή / και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα. Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και μόνο.

Υπογραφή Φοιτητή

Ευχαριστίες

Αρχικά, θα ήθελα να ευχαριστήσω τον Δρ. Μηνά Δασυγένη για την πολύτιμη συμβολή και καθοδήγηση του καθώς και για την υπομονή που έδειξε καθ‘όλη την διάρκεια εκπόνησης της παρούσας διπλωματικής εργασίας.

Επιπλέον, θα ήθελα να ευχαριστήσω την οικογένεια μου για την αμέριστη ηθική και υλική υποστήριξη που μου παρείχε κατά την διάρκεια των σπουδών μου.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Εξισώσεων

Κατάλογος Ψευδοκώδικα

1. Εισαγωγή

Η συνεχής ανάγκη για παραγωγή νέων ψηφιακών κυκλωμάτων μας έχει οδηγήσει στην αυτοματοποίηση αυτής της διαδικασίας. Μία αυτοματοποιημένη διαδικασία μας προσφέρει συνέπεια στην ποιότητα και ταχύτητα. Συνέπεια στην ποιότητα σημαίνει, ότι ανάλογα με τις προδιαγραφές που έχουμε ορίσει, το αποτέλεσμα θα είναι πάντα στο ίδιο επίπεδο ποιοτικώς. Αυτά τα δύο χαρακτηριστικά είναι αρκετά, ώστε να μας ωθήσουν να μελετήσουμε και να βρούμε αποδοτικότερους τρόπους αυτοματοποίησης μιας διαδικασίας.

1.1 Το πρόβλημα

Ως δεδομένα του προβλήματος που έχουμε να αντιμετωπίσουμε θεωρούμε ότι έχουμε τα κυκλωματικά στοιχεία ενός ψηφιακού κυκλώματος και τις διασυνδέσεις μεταξύ τους. Παράδειγμα: Το κύκλωμα έχει δύο πύλες AND με δύο εισόδους και μία πύλη OR με δύο εισόδους. Η έξοδος της πρώτης AND πύλης συνδέεται με την δεύτερη είσοδο της πύλης OR κ.ο.κ. Ο σκοπός μας είναι, έχοντας αυτά τα δεδομένα, να τοποθετήσουμε τα στοιχεία στον καμβά και να τα συνδέσουμε με τέτοιον τρόπο, ώστε να βελτιστοποιήσουμε κάποια μεγέθη, όπως το συνολικό εμβαδόν που καταλαμβάνει το κύκλωμα να είναι το ελάχιστο, το μήκος των καλωδίων να είναι το ελάχιστο και παράλληλα το κύκλωμα να διατηρεί την αναγνωσιμότητα του, την λειτουργικότητά του και την αισθητική του. Το πρόβλημα που αντιμετωπίσαμε είναι γενικό και θεωρητικά θα πρέπει η επίλυση του να ισχύει για κάθε κύκλωμα ανεξαρτήτου μεγέθους και πολυπλοκότητας. Πρακτικά όμως υπάρχουν περιορισμοί, όπως η πεπερασμένη υπολογιστική ισχύς και οι περιορισμοί που εισέρχονται από την χρήση του λογισμικού Logisim. Το Logisim επιβάλλει κάποια όρια ως προς τα μεγέθη των στοιχείων και τον τρόπο με τον οποίο μπορούν να γίνονται διασυνδέσεις μεταξύ τους. Όλα αυτούς τους περιορισμούς θα πρέπει να λάβουμε υπόψη και κατά την σχεδίαση του αλγόριθμου αλλά και ως προς την χρονική πολυπλοκότητα που θα προσθέσουν στο πρόγραμμα. Πιο συγκεκριμένα για να καταλάβουμε την απαίτηση για υπολογιστική ισχύ, εάν δοκιμάζαμε να επιλύσουμε το πρόβλημα δοκιμάζοντας όλους τους πιθανούς συνδυασμούς το πρόγραμμα δεν θα τερμάτιζε ποτέ. Γι αυτόν τον λόγο έπρεπε να βρούμε μια λύση προσεγγιστικά.

1.2 Ο Σκοπός

Σκοπός της παρούσας διπλωματικής εργασίας είναι η αυτοματοποίηση της διαδικασίας δημιουργίας του σχηματικού ενός ψηφιακού κυκλώματος. Αναπτύχθηκε σχετικό λογισμικό σε γλώσσα προγραμματισμού Java. Ποιο συγκεκριμένα, αναπτύχθηκε λογισμικό που με δεδομένα τα κυκλωματικά στοιχεία και τις διασυνδέσεις ενός ψηφιακού κυκλώματος, δημιουργεί το σχηματικό του σε αρχείο συμβατό με το Logisim. Η τοποθέτηση των στοιχείων με βέλτιστο τρόπο είναι ένα από τα σημαντικότερα προβλήματα που αντιμετωπίσαμε. Χρησιμοποιήθηκε τεχνητή νοημοσύνη και η τεχνική του γενετικού αλγόριθμου για την επίλυση του προβλήματος τοποθέτησης των αντικειμένων σε συγκεκριμένες θέσεις στον καμβά. Η εν λόγω μέθοδος μας έδωσε πολύ καλά αποτελέσματα με σχετικά γρήγορη εκτέλεση του προγράμματος. Στο σημείο αυτό να πούμε πως οι χρόνοι εκτέλεσης της τάξης των λεπτών και ωρών για το πρόβλημα μας είναι αποδεκτοί.

Με ενθουσιασμό εκτελέσαμε αυτό το εγχείρημα διότι οι δυνατότητες που ανοίγονται με την επιτυχή ολοκλήρωσή του είναι πολλές και μπορούν προκύψουν και ακόμη πιο ενδιαφέροντα και έξυπνα προγράμματα με την ενσωμάτωση του σε ήδη υπάρχουσες υποδομές. Όπως σύνδεση με VHDL ή text base language, έτσι ώστε να έχουμε ολοκληρωμένη αυτόματη διαδικασία παραγωγής ψηφιακών κυκλωμάτων από τον πίνακα αληθείας έως το σχηματικό του.

1.3 Σχετικές μελέτες

Ο σχεδιασμός ψηφιακών κυκλωμάτων είναι μια δύσκολη και πολύπλοκη διαδικασία που απαιτεί την εκπλήρωση ποικίλων στόχων ενώ ταυτόχρονα πρέπει να υπάρχει συμμόρφωση με περιορισμούς. Στόχος του μηχανικού είναι να δημιουργήσει ένα διάγραμμα κυκλώματος που ικανοποιεί συγκεκριμένους σχεδιαστικούς στόχους και συμμορφώνεται με τους παγκόσμιους σχεδιαστικούς κανόνες γνωστούς ως IEC (International Electrotechnical Commission) πρότυπα (McCabe, December 1976). Ο σχεδιασμός περιλαμβάνει τρία κύρια στάδια: επιλογή τοπολογίας, δημιουργία μεγέθους στοιχείων και διάταξης. Τόσο η επιλεγμένη τοπολογία όσο και η το μέγεθος πρέπει να διασφαλίζει ότι το προκύπτον κύκλωμα ικανοποιεί τους σχεδιαστικούς στόχους.

Η διαδικασία σχεδιασμού χαρακτηρίζεται σε συνδυασμό από εμπειρία και διαίσθηση και απαιτεί πλήρη γνώση των χαρακτηριστικών της διαδικασίας και τις λεπτομερείς προδιαγραφές του πραγματικού προϊόντος. Ο σχεδιασμός ψηφιακών κυκλωμάτων είναι μια εργασία που απαιτεί βαθειά γνώση του αντικειμένου, είναι μία πολυεπίπεδη, επαναληπτική και χρονοβόρα διαδικασία, η οποία εκτελείται από έμπειρους σχεδιαστές με υψηλά προσόντα. Θεωρείται επομένως από πολλούς να είναι μια μορφή τέχνης παρά επιστήμης. Τα τελευταία χρόνια οι ερευνητές έχουν διερευνήσει εκτενώς το σχεδιασμό και τον έλεγχο σχεδιασμού αναλογικών και ψηφιακών κυκλωμάτων. Ο σύγχρονος αυτοματισμός της διαδικασίας συνεπάγεται ευρετικές, γενετικούς αλγόριθμους, πολλαπλή αντικειμενική βελτιστοποίηση και νοημοσύνη σμήνους.

Μία σχετική μελέτη που βρέθηκε με τίτλο “Λογισμικό για σχεδίαση αναλογικών κυκλωμάτων με γενετικό αλγόριθμο” (Miri Weiss Cohen, 2015), προσέγγισε το πρόβλημα με μία πολύ ενδιαφέρουσα τεχνική. Αφού γινόταν η τμηματοποίηση του χώρου σε ίσα μικρότερα τμήματα και η εισαγωγή των στοιχείων σε αυτά, τότε έλυνε τον γενετικό αλγόριθμο για κάθε τμήμα ξεχωριστά και στο τέλος γινόταν η ένωση των επιμέρους τμημάτων. Αυτό είχε σαν αποτέλεσμα την βελτίωση του χρόνου εκτέλεσης λόγο της διαίρεσης του προβλήματος σε μικρότερα υποπροβλήματα. Το αρνητικό είναι βέβαια ότι με αυτόν τον τρόπο η τμηματοποίηση ήταν εμφανής στο τελικό κύκλωμα. Φαινόταν δηλαδή με το μάτι ότι τα στοιχεία έχουν τοποθετηθεί σε ομάδες με μικρές κατά κύριο λόγο καλωδιώσεις μεταξύ των στοιχείων των ομάδων και οι ενώσεις εκτός της ίδιας ομάδας ήταν σχετικά μεγάλες καλωδιώσεις.

Γι αυτό τον λόγο εμείς ακολουθήσαμε μια διαφορετική προσέγγιση που ταιριάζει περισσότερο στα ψηφιακά κυκλώματα. Αντί να διαιρέσουμε τις διαστάσεις σε ίσα τμήματα, χωρίσαμε τον χώρο σε κάθετα μακρόστενα τμήματα που χωράνε οριζοντίως ένα στοιχείο (το μεγαλύτερο του τμήματος) αλλά καθέτως πολλά. Αυτή η τεχνική δεν δίνει μόνο μια φυσική ροή στο κύκλωμα από τα inputs προς τα outputs αλλά δίνει και περισσότερη ελευθερία στον γενετικό αλγόριθμο να προσαρμόσει την διάρθρωση του κυκλώματος.

1.4 Διάρθρωση του κειμένου

Τα υπόλοιπα κεφάλαια οργανώνονται ως εξής: Στο κεφάλαιο 2 παρουσιάζεται το θεωρητικό υπόβαθρο των εννοιών που χρησιμοποιούνται. Ακολουθεί η περιγραφή του λογισμικού μέρους στο κεφάλαιο 3. Έπειτα, στο κεφάλαιο 4 γίνεται η εκτέλεση του προγράμματος και δίνονται ορισμένα πειραματικά αποτελέσματα. Στο κεφάλαιο 5 καταγράφονται ορισμένες μετρικές του πηγαίου κώδικα. Τέλος, στο κεφάλαιο 6 αναλύουμε τα συμπεράσματα που προέκυψαν και προτείνουμε μελλοντικές επεκτάσεις-βελτιώσεις.

2. Θεωρητικό υπόβαθρο

2.1 Η γλώσσα προγραμματισμού Java

Η Java είναι μια γλώσσα προγραμματισμού γενικού σκοπού που βασίζεται σε κλάσεις, αντικειμενοστρέφια και έχει σχεδιαστεί για να έχει όσο το δυνατόν λιγότερες εξαρτήσεις. Σκοπός είναι να επιτρέπεται στους προγραμματιστές εφαρμογών να γράφουν μία φορά κώδικα και να εκτελείται οπουδήποτε (Langley, 2002) που σημαίνει ότι ο μεταγλωττισμένος κώδικας Java μπορεί να εκτελεστεί σε όλες τις πλατφόρμες που υποστηρίζουν Java χωρίς την ανάγκη επαναμεταγλώττισης (Sakaiya, 1999). Οι εφαρμογές Java μεταγλωττίζονται σε γλώσσα μηχανής και μπορούν να εκτελεστούν σε οποιαδήποτε εικονική μηχανή Java (JVM) ανεξάρτητα από την αρχιτεκτονική του υπολογιστή. Η σύνταξη της Java είναι παρόμοια με C και C ++, αλλά έχει λιγότερες λειτουργίες χαμηλού επιπέδου από αυτές. Από το 2019, η Java ήταν μία από τις πιο δημοφιλείς γλώσσες προγραμματισμού που χρησιμοποιούνται σύμφωνα με το GitHub ειδικά για εφαρμογές ιστού-διακομιστή-πελάτη, με 9 εκατομμύρια εφαρμογές (Chan, 2019).

Η Java αναπτύχθηκε αρχικά από τον James Gosling στην Sun Microsystems (η οποία έκτοτε εξαγοράστηκε από την Oracle) και κυκλοφόρησε το 1995 ως βασικό συστατικό της πλατφόρμας Java της Sun Microsystems (Oracle, 2013). Οι αρχικοί μεταγλωττιστές Java και οι εφαρμογές αναφοράς, οι εικονικές μηχανές και οι βιβλιοθήκες κλάσεων κυκλοφόρησαν αρχικά από την Sun με ιδιόκτητες άδειες . Από τον Μάιο του 2007, σύμφωνα με τις προδιαγραφές της Java Community Process, η Sun είχε παραιτηθεί από τις περισσότερες από τις τεχνολογίες της Java βάσει της άδειας GNU General Public License. Εν τω μεταξύ, άλλοι έχουν αναπτύξει εναλλακτικές εφαρμογές αυτών των τεχνολογιών της Sun, όπως το GNU Compiler για Java (bytecode compiler), GNU Classpath (τυπικές βιβλιοθήκες) και το IcedTea-Web (πρόγραμμα περιήγησης για μικροεφαρμογές) (Sakaiya, 1999).

Οι πιο πρόσφατες εκδόσεις είναι η Java 14, κυκλοφόρησε τον Μάρτιο του 2020 και η Java 11, μια τρέχουσα υποστηριζόμενη μακροπρόθεσμη υποστήριξη (LTS), η οποία κυκλοφόρησε στις 25 Σεπτεμβρίου 2018. Η Oracle κυκλοφόρησε για το παλαιό Java 8 LTS την τελευταία δωρεάν δημόσια ενημέρωση τον Ιανουάριο του 2019 για εμπορική χρήση, ενώ διαφορετικά θα υποστηρίζει το Java 8 με δημόσιες ενημερώσεις για προσωπική χρήση έως τουλάχιστον τον Δεκέμβριο του 2020 (McMillan, 2013).

Πλεονεκτήματα γλώσσας Java

- Φορητότητα: Εκτέλεση σε οποιαδήποτε υποδομή αρκεί να υπάρχει το JRE.
- Απόδοση: Δημιουργείται εκτελέσιμο σε γλώσσα μηχανής.
- Ασφάλεια τύπων: Βοηθάει στην αποφυγή σφαλμάτων.
- Αυτόματη διαχείριση μνήμης: Δεν μας απασχολεί η δέσμευση και η απελευθέρωση μνήμης. Το κάνει για εμάς ο Garbage Collector.
- Υποστήριξη αντανάκλασης: Αλλαγή συμπεριφοράς του προγράμματος από το ίδιο κατά την εκτέλεση.

2.2 Εφαρμογές Java

Μια εφαρμογή Java SE είναι μια εφαρμογή γραμμένη στην Java Platform Standard Edition (Java SE). Οι ίδιες μη τροποποιημένες εφαρμογές Java SE μπορούν να εκτελεστούν σε σχεδόν οποιονδήποτε υπολογιστή, είτε αυτός ο υπολογιστής χρησιμοποιεί λειτουργικά συστήματα Microsoft Windows, Solaris, Linux ή OS X. Το κλειδί αυτής της φορητότητας εφαρμογών είναι το Java Runtime Environment, το οποίο διατίθεται δωρεάν για τα περισσότερα λειτουργικά συστήματα, συμπεριλαμβανομένων όλων αυτών που αναφέρονται παραπάνω.

Εκτός από την πλατφόρμα για εφαρμογές πολλαπλών πλατφόρμων για επιτραπέζιους υπολογιστές, η πλατφόρμα Java SE αποτελεί τη βάση για άλλες τεχνολογίες όπως τη Java Platform Enterprise Edition. Μπορούμε να γράψουμε κώδικα Java που παρέχει λογική back-end για εφαρμογές ιστού και υποδομής επιχειρήσεων.

2.3 Η Πλατφόρμα NetBeans

Η πλατφόρμα NetBeans είναι ένα πλαίσιο για την απλοποίηση της ανάπτυξης εφαρμογών Swing για επιτραπέζιους υπολογιστές. Το πακέτο NetBeans IDE για Java SE περιέχει ότι χρειάζεται για να ξεκινήσει η ανάπτυξη εφαρμογών NetBeans και εφαρμογών που βασίζονται στην πλατφόρμα NetBeans. Δεν απαιτείται πρόσθετο SDK (The Apache Software Foundation Announces Apache® NetBeans™ as a Top-Level Project, July 12, 2019).

Οι εφαρμογές μπορούν να εγκαταστήσουν επεκτάσεις δυναμικά. Οποιαδήποτε εφαρμογή μπορεί να περιλαμβάνει τη λειτουργική μονάδα του κέντρου ενημερώσεων για να επιτρέπει στους χρήστες της εφαρμογής να κατεβάζουν ψηφιακά υπογεγραμμένες αναβαθμίσεις και νέες δυνατότητες απευθείας στην εφαρμογή που εκτελείται. Η επανεγκατάσταση μιας αναβάθμισης ή μιας νέας έκδοσης δεν αναγκάζει τους χρήστες να κάνουν ξανά λήψη ολόκληρης της εφαρμογής.

Η πλατφόρμα προσφέρει επαναχρησιμοποιήσιμες υπηρεσίες κοινές για εφαρμογές επιτραπέζιων υπολογιστών, επιτρέποντας στους προγραμματιστές να επικεντρώνονται στη συγκεκριμένη λογική για την εφαρμογή τους. Μεταξύ των χαρακτηριστικών της πλατφόρμας είναι:

- Διαχείριση διεπαφής χρήστη (π.χ. μενού και γραμμές εργαλείων)
- Διαχείριση ρυθμίσεων χρήστη
- Διαχείριση αποθήκευσης (πραγματοποιεί αποτελεσματική αποθήκευση)
- Διαχείριση παραθύρων
- Πλαίσιο οδηγού (υποστηρίζει διαλόγους βήμα προς βήμα)
- Οπτική βιβλιοθήκη NetBeans
- Ολοκληρωμένα εργαλεία ανάπτυξης

2.4 Το Ολοκληρωμένο περιβάλλον ανάπτυξης NetBeans (IDE)

Το NetBeans IDE είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης ανοιχτού κώδικα. Το NetBeans IDE υποστηρίζει την ανάπτυξη όλων των τύπων εφαρμογών Java (Java SE (συμπεριλαμβανομένου του JavaFX), Java ME, web, EJB και εφαρμογών για κινητά)

απευθείας. Μεταξύ των άλλων χαρακτηριστικών είναι ένα σύστημα έργου που βασίζεται σε Ant (εργαλείο αυτοματοποίησης της διαδικασίας της μεταγλώττισης), υποστήριξη Maven (επίσης εργαλείο αυτοματοποίησης της διαδικασίας της μεταγλώττισης), refactorings (αλλαγές στον κώδικα που βοηθούν στην συντήρηση και αναγνωσιμότητα χωρίς να επηρεάζουν την λειτουργία του) και έλεγχο εκδόσεων.

Όλες οι λειτουργίες του IDE παρέχονται από επεκτάσεις (Modules). Κάθε επέκταση παρέχει μια καλά καθορισμένη λειτουργία, όπως υποστήριξη για τη γλώσσα Java, επεξεργασία ή υποστήριξη για το σύστημα εκδόσεων CVS και SVN. Το NetBeans περιέχει όλες τις λειτουργικές επεκτάσεις που απαιτούνται για την ανάπτυξη Java προγραμμάτων σε μία λήψη μόνο, επιτρέποντας στον χρήστη να αρχίσει να εργάζεται αμέσως. Οι επεκτάσεις επιτρέπουν επίσης την επέκταση του NetBeans με νέες δυνατότητες, όπως υποστήριξη για άλλες γλώσσες προγραμματισμού, μπορούν να προστεθούν με την εγκατάσταση πρόσθετων επεκτάσεων. Για παράδειγμα, τα Sun Studio, Sun Java Studio Enterprise και Sun Java Studio Creator από τα Sun Microsystems βασίζονται στο NetBeans IDE.

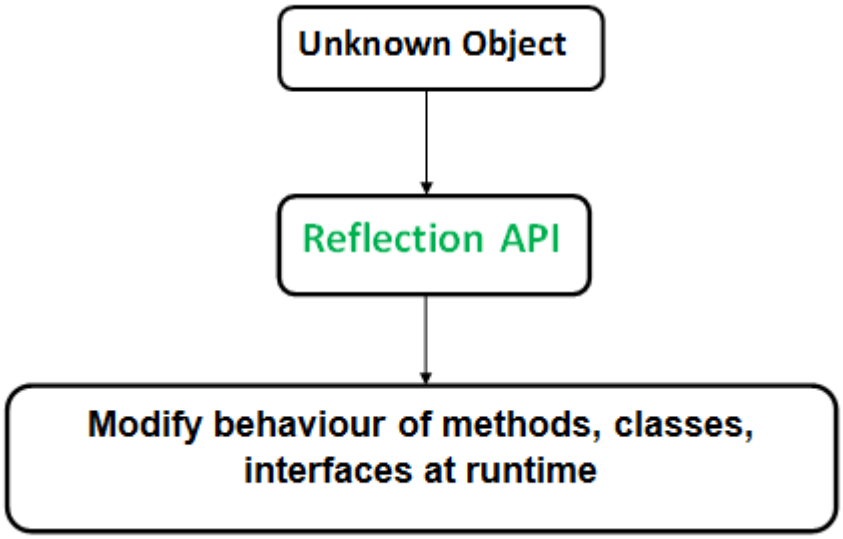
2.5 Η Αντανάκλαση στην Java (Reflection)

Η αντανάκλαση είναι μία διεπαφή προγραμματισμού εφαρμογών (API) που χρησιμοποιείται για να εξετάσει ή να τροποποιήσει τη συμπεριφορά μεθόδων, κλάσεων, διεπαφών κατά το χρόνο εκτέλεσης .

Οι απαιτούμενες κλάσεις παρέχονται στο πακέτο java.lang.reflect.

Η αντανάκλαση μας δίνει πληροφορίες σχετικά με την κλάση στην οποία ανήκει ένα αντικείμενο και επίσης τις μεθόδους αυτής της κλάσης που μπορούν να εκτελεστούν χρησιμοποιώντας το αντικείμενο (Laplante, 25 April 2007).

Μέσω της αντανάκλασης μπορούμε να επικαλεστούμε μεθόδους κατά το χρόνο εκτέλεσης, ανεξάρτητα από τον προσδιοριστή πρόσβασης που χρησιμοποιείται (Malenfant, 21 August 2017).



Εικόνα 1: Διάγραμμα λειτουργίας αντανάκλασης

Η αντανάκλαση μπορεί να χρησιμοποιηθεί για τη λήψη πληροφοριών σχετικά με -

1. **Class**: Η μέθοδος getClass() χρησιμοποιείται για να εξάγει το όνομα της κλάσης στην οποία ανήκει ένα αντικείμενο.

2. **Constructors**: Η μέθοδος getConstructors() χρησιμοποιείται για να εξάγει τους δημόσιους κατασκευαστές της κλάσης στην οποία ανήκει ένα αντικείμενο.

3. **Methods**: Η μέθοδος getMethods() χρησιμοποιείται για να εξάγει τις δημόσιες μεθόδους της κλάσης στην οποία ανήκει ένα αντικείμενο.

2.5.1 Χρήσεις της αντανάκλασης

Η αντανάκλαση χρησιμοποιείται συνήθως από προγράμματα που απαιτούν τη δυνατότητα εξέτασης ή τροποποίησης της συμπεριφοράς τους κατά τον χρόνο εκτέλεσης τους στην εικονική μηχανή Java. Αυτό είναι ένα σχετικά προηγμένο χαρακτηριστικό και πρέπει να χρησιμοποιείται μόνο από προγραμματιστές που έχουν ισχυρή κατανόηση των βασικών στοιχείων της γλώσσας. Έχοντας υπόψη αυτήν την προειδοποίηση, η αντανάκλαση είναι μια ισχυρή τεχνική και επιτρέπει στις εφαρμογές να εκτελούν λειτουργίες που διαφορετικά θα ήταν αδύνατες (Brian Cantwell Smith, 1982).

Χαρακτηριστικά επεκτασιμότητας

Μια εφαρμογή μπορεί να κάνει χρήση εξωτερικών, καθορισμένων από το χρήστη κλάσεων, δημιουργώντας παρουσίες αντικειμένων επεκτασιμότητας χρησιμοποιώντας τα πλήρως αναγνωρισμένα ονόματά τους.

Περιηγητές και περιβάλλοντα οπτικής ανάπτυξης

Ένα πρόγραμμα περιήγησης πρέπει να μπορεί να απαριθμεί τα μέλη των κλάσεων. Τα οπτικά περιβάλλοντα ανάπτυξης μπορούν να επωφεληθούν από τη χρήση πληροφοριών τύπου διαθέσιμων στην αντανάκλαση για να βοηθήσουν τον προγραμματιστή να γράψει σωστό κώδικα.

Εργαλεία εντοπισμού σφαλμάτων και δοκιμής

Οι εντοπιστές σφαλμάτων πρέπει να είναι σε θέση να εξετάζουν ιδιωτικά μέλη σε κλάσεις. Τα εργαλεία δημιουργίας προφίλ (Profiling Tools) να μπορούν να κάνουν χρήση της αντανάκλασης για να καλούν συστηματικά ένα εύχρηστο σύνολο από διεπαφές προγραμματισμού εφαρμογών (API) που ορίζονται σε μια κλάση, για να διασφαλιστεί ένα υψηλό επίπεδο κάλυψης κώδικα.

2.5.2 Μειονεκτήματα της αντανάκλασης

Η αντανάκλαση είναι ισχυρή τεχνική, αλλά δεν πρέπει να χρησιμοποιείται άσκοπα. Εάν είναι δυνατή η εκτέλεση μιας λειτουργίας χωρίς χρήση αντανάκλασης, είναι προτιμότερο να αποφεύγεται η χρήση της. Τα ακόλουθα μειονεκτήματα πρέπει να ληφθούν υπόψη κατά την πρόσβαση στον κώδικα μέσω αντανάκλασης.

Επιδόσεις

Επειδή η αντανάκλαση περιλαμβάνει τύπους που επιλύονται δυναμικά, ορισμένες βελτιστοποιήσεις της εικονικής μηχανής Java δεν μπορούν να πραγματοποιηθούν. Κατά συνέπεια, οι ανακλαστικές λειτουργίες έχουν βραδύτερη απόδοση από τις αντίστοιχες μη αντανακλαστικές τους, και θα πρέπει να αποφεύγονται σε τμήματα του κώδικα που καλούνται συχνά σε εφαρμογές ευαίσθητες στην απόδοση.

Περιορισμοί ασφαλείας

Η αντανάκλαση απαιτεί άδεια εκτέλεσης που ενδέχεται να μην υπάρχει όταν εκτελείται υπό διαχείριση ασφαλείας. Αυτό είναι ένα σημαντικό ζήτημα για κώδικα που πρέπει να εκτελείται σε περιορισμένο περιβάλλον ασφαλείας, όπως σε ένα Applet.

Έκθεση εσωτερικών τμημάτων

Δεδομένου ότι η αντανάκλαση επιτρέπει στον κώδικα να εκτελεί λειτουργίες που θα ήταν παράνομες σε μη ανακλαστικό κώδικα, όπως πρόσβαση σε ιδιωτικά πεδία και μεθόδους, η χρήση αντανάκλασης μπορεί να οδηγήσει σε απροσδόκητες παρενέργειες, οι οποίες μπορεί να καταστήσουν τον κώδικα δυσλειτουργικό και να καταστρέψουν τη φορητότητα. Ο ανακλαστικός κώδικας μπορεί να διακόψει την αφαιρετητκότητα και επομένως μπορεί να αλλάξει συμπεριφορά με τις αναβαθμίσεις της πλατφόρμας (Smith, January 1982).

Η χρήση της αντανάκλασης στην παρούσα διπλωματική εργασία

Η αντανάκλαση στην παρούσα διπλωματική εργασία χρησιμοποιήθηκε κατά την δημιουργία των κυκλωματικών στοιχείων από το αρχείο εισόδου. Έχει δημιουργηθεί μία βιβλιοθήκη κλάσεων για τα κυκλωματικά στοιχεία, που η κάθε κλάση περιγράφει τα χαρακτηριστικά και τις λειτουργίες των αντίστοιχων κυκλωματικών στοιχείων, όπως το μέγεθος τους, τις σχετικές θέσεις των pins, αν επιτρέπεται η αλλαγή προσανατολισμού και διάφορα άλλα χαρακτηριστικά. Ο λόγος είναι για να μπορούμε όταν διαβάζουμε το αλφαριθμητικό από το αρχείο εισόδου να δημιουργούμε δυναμικά και κατά την εκτέλεση του προγράμματος το αντίστοιχο αντικείμενο της κλάσης.

2.6 Προβλήματα βελτιστοποίησης και ο γενετικός αλγόριθμος

Ο γενετικός αλγόριθμος είναι μια ευρετική αναζήτηση που εμπνέεται από τη θεωρία της φυσικής εξέλιξης του Charles Darwin. Αυτός ο αλγόριθμος αντικατοπτρίζει τη διαδικασία της φυσικής επιλογής όπου τα πιο κατάλληλα άτομα, με τα καλύτερα χαρακτηριστικά δηλαδή, επιλέγονται για αναπαραγωγή προκειμένου να παράγουν απόγονους της επόμενης γενιάς.

Σε έναν γενετικό αλγόριθμο, ένας πληθυσμός υποψήφιων λύσεων (που ονομάζονται άτομα, πλάσματα ή φαινότυποι) σε ένα πρόβλημα βελτιστοποίησης εξελίσσεται προς καλύτερες λύσεις. Κάθε υποψήφια λύση έχει ένα σύνολο ιδιοτήτων (χρωμοσώματα ή

γονότυπους) που μπορούν να μεταλλαχθούν και να τροποποιηθούν. Παραδοσιακά, οι λύσεις αντιπροσωπεύονται σε δυαδική μορφή ως συμβολοσειρές των 0 και 1, αλλά είναι επίσης δυνατές και άλλες κωδικοποιήσεις (Whitley, 1994).

Η εξέλιξη ξεκινά συνήθως από έναν πληθυσμό ατόμων που δημιουργούνται τυχαία και είναι μια επαναληπτική διαδικασία, με τον πληθυσμό σε κάθε επανάληψη να ονομάζεται γενιά. Σε κάθε γενιά αξιολογείται η καταλληλότητα κάθε ατόμου στον πληθυσμό. η φυσική κατάσταση είναι συνήθως η αξία της αντικειμενικής συνάρτησης στο πρόβλημα βελτιστοποίησης που επιλύεται. Τα πιο κατάλληλα άτομα επιλέγονται στοχαστικά από τον τρέχοντα πληθυσμό και το γονίδιο κάθε ατόμου τροποποιείται (ανασυνδυάζεται και πιθανώς τυχαία μεταλλάσσεται) (Crossover and Mutation) για να σχηματίσει μια νέα γενιά. Η νέα γενιά υποψήφιων λύσεων χρησιμοποιείται στη συνέχεια στην επόμενη επανάληψη του αλγορίθμου. Συνήθως, ο αλγόριθμος τερματίζεται όταν είτε έχει παραχθεί ένας μέγιστος αριθμός γενεών, είτε έχει επιτευχθεί ικανοποιητικό επίπεδο φυσικής κατάστασης για τον πληθυσμό.

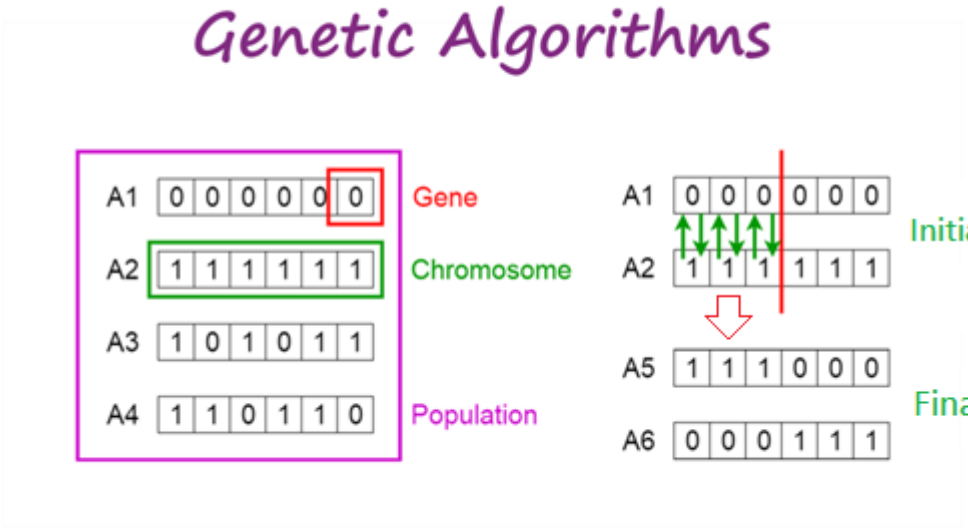
Ένας τυπικός γενετικός αλγόριθμος απαιτεί:

Α) Μια γενετική αναπαράσταση του τομέα της λύσης,

Β) Μια λειτουργία φυσικής κατάστασης για την αξιολόγηση του τομέα λύσης.

Μια τυπική αναπαράσταση κάθε υποψήφιας λύσης είναι ως μια σειρά bit. Οι πίνακες άλλων τύπων και δομών μπορούν να χρησιμοποιηθούν ουσιαστικά με τον ίδιο τρόπο. Η κύρια ιδιότητα που καθιστά αυτές τις γενετικές αναπαραστάσεις βολικές είναι ότι τα μέρη τους ευθυγραμμίζονται εύκολα λόγω του σταθερού τους μεγέθους, το οποίο διευκολύνει τις απλές εργασίες crossover (Surry, 1995). Μπορούν επίσης να χρησιμοποιηθούν παραστάσεις μεταβλητού μήκους, αλλά η εφαρμογή crossover είναι πιο περίπλοκη σε αυτήν την περίπτωση. Οι δενδροειδείς αναπαραστάσεις διερευνώνται σε γενετικό προγραμματισμό και οι αναπαραστάσεις μορφής γραφημάτων διερευνούνται στον εξελικτικό προγραμματισμό. Ένας συνδυασμός γραμμικών χρωμοσωμάτων και δέντρων διερευνάται στον προγραμματισμό γονιδιακής έκφρασης (Carter, 1995).

Μόλις καθοριστεί η γενετική αναπαράσταση και η λειτουργία φυσικής κατάστασης, ένας γενετικός αλγόριθμος προχωρά στην αρχικοποίηση ενός πληθυσμού λύσεων και στη συνέχεια στη βελτίωσή του μέσω επαναλαμβανόμενης εφαρμογής των τελεστών μετάλλαξης, διασταύρωσης, αντιστροφής και επιλογής.



Εικόνα 2: Αναπαράσταση γονιδίου, χρωμοσώματος και πληθυσμού σε έναν γενετικό αλγόριθμο

Η έννοια της φυσικής επιλογής

Η διαδικασία που αρχίζει με την επιλογή των πιο κατάλληλων ατόμων από έναν πληθυσμό, παράγουν απογόνους που κληρονομούν τα χαρακτηριστικά των γονέων και θα προστεθούν στην επόμενη γενιά. Εάν οι γονείς έχουν καλύτερη φυσική κατάσταση, οι απόγονοι τους θα είναι καλύτεροι από τους γονείς και θα έχουν περισσότερες πιθανότητες να επιβιώσουν. Αυτή η διαδικασία συνεχίζεται και στο τέλος θα βρεθεί μια γενιά με τα πιο ικανά άτομα.

Αυτή η έννοια μπορεί να εφαρμοστεί για ένα πρόβλημα αναζήτησης. Θεωρούμε μια σειρά λύσεων για ένα πρόβλημα και επιλέγουμε το σύνολο των καλύτερων από αυτές.

Οι πέντε φάσεις του γενετικού αλγόριθμου περιγράφονται παρακάτω:

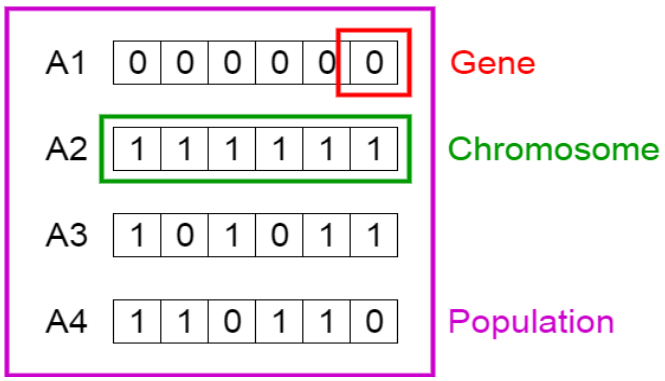
- Αρχικός Πληθυσμός
- Συνάρτηση Αξιολόγησης
- Επιλογή
- Διασταύρωση
- Μετάλλαξη

2.6.1 Αρχικός πληθυσμός

Η διαδικασία ξεκινά με ένα σύνολο ατόμων που ονομάζεται Αρχικός Πληθυσμός. Κάθε άτομο είναι μια λύση στο πρόβλημα που θέλουμε να λύσουμε.

Ένα άτομο χαρακτηρίζεται από ένα σύνολο παραμέτρων (μεταβλητών) γνωστών ως γονιδίων. Τα γονίδια ενώνονται σε ένα αλφαριθμητικό για να σχηματίσουν ένα χρωμόσωμα (μία λύση).

Σε έναν γενετικό αλγόριθμο, το σύνολο των γονιδίων ενός ατόμου αντιπροσωπεύεται χρησιμοποιώντας μια συμβολοσειρά, από την άποψη ενός αλφάβητου. Συνήθως, χρησιμοποιούνται δυαδικές τιμές (συμβολοσειρά 0 και 1). Λέμε ότι κωδικοποιούμε τα γονίδια σε ένα χρωμόσωμα.



Εικόνα 3: Πληθυσμός, Χρωμόσωμα και Γονίδιο

2.6.2 Λειτουργία φυσικής κατάστασης

Η λειτουργία φυσικής κατάστασης καθορίζει το πόσο κατάλληλο είναι ένα άτομο (η ικανότητα ενός ατόμου να ανταγωνίζεται με άλλα άτομα). Δίνει μια βαθμολογία fitness σε κάθε άτομο. Η πιθανότητα να επιλεγεί ένα άτομο για αναπαραγωγή βασίζεται στο σκορ ικανότητάς του.

2.6.3 Επιλογή

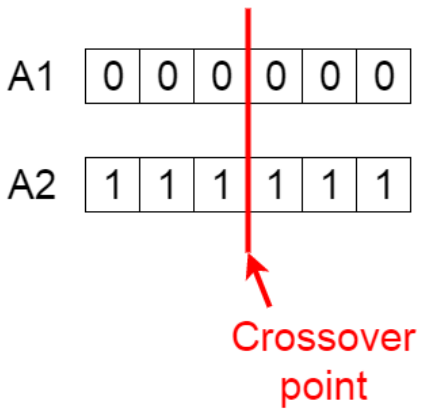
Η ιδέα της φάσης της επιλογής είναι να επιλέξουμε τα καταλληλότερα άτομα (με βάση την βαθμολογία της φυσικής τους κατάστασης) και να τα αφήσουμε να περάσουν τα γονίδιά τους στην επόμενη γενιά.

Δύο ζεύγη ατόμων (γονέων) επιλέγονται με βάση τις βαθμολογίες της φυσικής τους κατάστασης. Τα άτομα με υψηλή φυσική κατάσταση έχουν περισσότερες πιθανότητες να επιλεγούν για αναπαραγωγή. Τα άτομα με χαμηλή φυσική κατάσταση επιλέγονται και αυτά, αλλά με μικρότερη πιθανότητα. Αυτό το κάνουμε γιατί μπορεί ένα άτομο με χαμηλή φυσική κατάσταση να έχει ένα πολύ καλό χαρακτηριστικό και γι αυτό τον λόγο θέλουμε να του δώσουμε μια πιθανότητα αυτό το χαρακτηριστικό να περάσει και στις επόμενες γενιές (McMahon, 1991).

2.6.4 Διασταύρωση

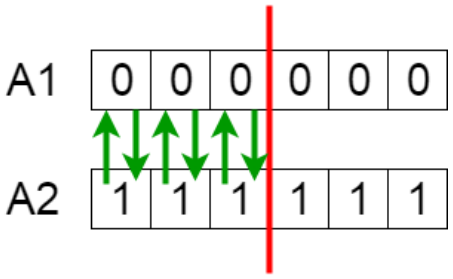
Η διασταύρωση είναι η πιο σημαντική φάση ενός γενετικού αλγορίθμου. Για κάθε ζεύγος γονέων που πρόκειται να ζευγαρώσουν, επιλέγεται τυχαία μέσα στα γονίδια ένα σημείο διασταύρωσης.

Για παράδειγμα, θεωρήστε το σημείο διασταύρωσης να είναι 3 όπως φαίνεται παρακάτω.



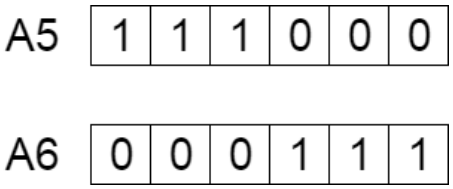
Εικόνα 4: Σημείο διασταύρωσης

Οι απόγονοι δημιουργούνται με την ανταλλαγή των γονιδίων των γονέων μεταξύ τους μέχρι να επιτευχθεί το σημείο διασταύρωσης.



Εικόνα 5: Ανταλλαγή γονιδίων μεταξύ των γονέων

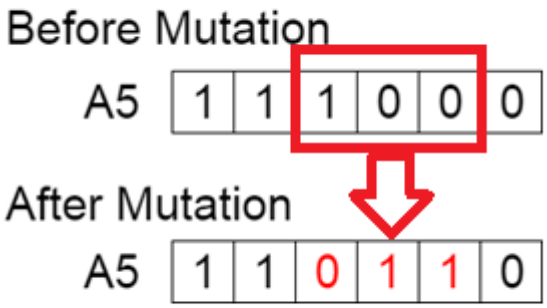
Ο απόγονοι προστίθενται στον πληθυσμό (στη νέα γενιά).



Εικόνα 6: Απόγονοι

2.6.5 Μετάλλαξη

Σε ορισμένους νέους απογόνους που σχηματίζονται, μερικά από τα γονίδιά τους μπορούν να υποβληθούν σε μια μετάλλαξη με χαμηλή τυχαία πιθανότητα. Αυτό σημαίνει ότι ορισμένα από τα δυαδικά ψηφία της συμβολοσειράς δυαδικών ψηφίων μπορούν να αναστραφούν.



Εικόνα 7: Μετάλλαξη: Πριν και Μετά

Η μετάλλαξη συμβαίνει για να διατηρηθεί η ποικιλομορφία εντός του πληθυσμού και να αποτραπεί η πρόωρη σύγκλιση.

Λήξη αλγορίθμου

Ο αλγόριθμος τερματίζει εάν ο πληθυσμός έχει συγκλίνει (δεν παράγει απογόνους οι οποίοι διαφέρουν σημαντικά από την προηγούμενη γενιά). Τότε λέγεται ότι ο γενετικός αλγόριθμος έδωσε ένα σύνολο λύσεων στο πρόβλημά μας.

Σχόλια

Ο πληθυσμός έχει σταθερό μέγεθος. Καθώς δημιουργούνται νέες γενιές, τα άτομα με λιγότερη φυσική κατάσταση πεθαίνουν χωρίς να περάσουν τα γονίδια τους στις επόμενες γενιές, παρέχοντας χώρο για νέους απογόνους.

Η ακολουθία των φάσεων επαναλαμβάνεται για να παράγονται άτομα σε κάθε νέα γενιά που είναι καλύτερα από την προηγούμενη γενιά.

```
START
Generate the initial population
Compute fitness
Repeat
        Selection
        Crossover
        Mutation
        Compute fitness
UNTIL the population has converged
STOP
```

Εικόνα 8: Ψευδοκώδικας Γενετικού Αλγορίθμου

2.7 Ο Αλγόριθμος αναζήτησης A-Star (A*)

Η επίτευξη ενός προορισμού μέσω της συντομότερης διαδρομής είναι μια καθημερινή δραστηριότητα που όλοι κάνουμε. Ο A-star (που επίσης αναφέρεται ως A*) είναι ένας από τους πιο επιτυχημένους αλγορίθμους αναζήτησης για να βρούμε τη διαδρομή με το μικρότερο κόστος μεταξύ κόμβων ή γραφημάτων (Sobey., 2006). Θα τον χρησιμοποιήσουμε για να βρούμε την συντομότερη διαδρομή για τις καλωδιώσεις μεταξύ των στοιχείων. Είναι ένας ενημερωμένος αλγόριθμος αναζήτησης, καθώς χρησιμοποιεί πληροφορίες σχετικά με το κόστος της διαδρομής αλλά και ευρετικά κριτήρια για να βρει τη λύση (Russell, 2018).

Ο A* είναι βέλτιστος και πλήρης αλγόριθμος, δύο πολύτιμες ιδιότητες των αλγορίθμων αναζήτησης.

Όταν ένας αλγόριθμος αναζήτησης έχει την ιδιότητα του βέλτιστου, σημαίνει ότι είναι εγγυημένη η βέλτιστη δυνατή λύση. Όταν ένας αλγόριθμος αναζήτησης έχει την ιδιότητα της πληρότητας αυτό σημαίνει ότι εάν υπάρχει λύση για ένα συγκεκριμένο πρόβλημα, ο αλγόριθμος εγγυάται ότι θα τη βρει (Delling, Sanders, Schultes, & Wagner, 2009).

Μερικές ορολογίες του A* για να καταλάβουμε πώς λειτουργεί:

- **Node** (λέγεται και **State**) — Ένας κόμβος στον χώρο.
- **Transition** — Ο τρόπος με τον οποίο ορίζεται η κίνηση στον χώρο.
- **Starting Node —** Αρχικός κόμβος.
- **Goal Node —** Τελικός κόμβος.
- **Search Space** — Όλοι οι πιθανοί κόμβοι του χώρου.
- **Cost** — Αριθμητική τιμή (κόστος) για την μετακίνηση από ένα κόμβο σε έναν άλλο.
- $\boldsymbol{g(n)}$— είναι το ακριβές κόστος από τον αρχικό κόμβο σε οποιονδήποτε άλλο.
- $\boldsymbol{h(n)}$— το εκτιμώμενο κόστος από οποιονδήποτε κόμβο προς τον κόμβο στόχο. Το κόστος αυτό δεν πρέπει ποτέ να υπερεκτιμάται για να λειτουργήσει σωστά ο αλγόριθμος.
- $\boldsymbol{f(n)}$— Το συνολικό κόστος.

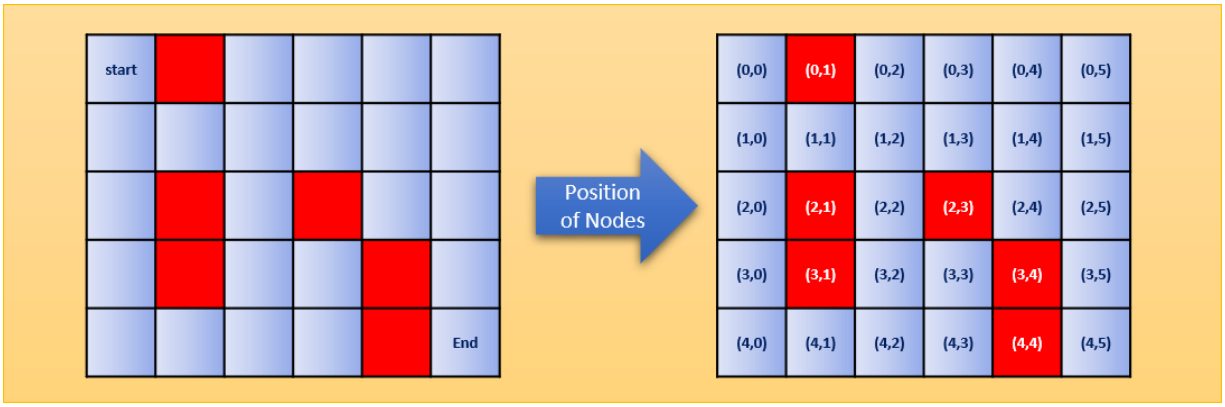
Κάθε φορά που A * εισέρχεται σε έναν κόμβο, υπολογίζει το κόστος $\boldsymbol{f(n)}$ (όπου n είναι ο γειτονικός κόμβος) και στη συνέχεια εισέρχεται στον κόμβο με τη χαμηλότερη τιμή της f(n).

Αυτές οι τιμές υπολογίζονται χρησιμοποιώντας τον ακόλουθο τύπο:

$$\boldsymbol{f(n) = g(n) + h(n)}$$

*Εξίσωση 1: Τύπος υπόλογισμού του κόστους A**

Για λόγους ευκολίας παρακάτω θα δούμε ένα παράδειγμα υλοποίησης σε γλώσσα Python. Η υλοποίηση αφορά ένα πρόβλημα λαβύρινθου και θα πρέπει να βρούμε τη συντομότερη διαδρομή από τον κόμβο έναρξης στον τελικό κόμβο του άκρου.



Εικόνα 9: Αριστερά: πρόβλημα λαβυρίνθου - Δεξιά: Θέση κάθε κόμβου (θέσεις δισδιάστατου πίνακα) του λαβυρίνθου

Για αυτό το πρόβλημα, υπάρχουν τέσσερις κινήσεις (αριστερά, δεξιά, πάνω και κάτω) από μια θέση λαβυρίνθου, εφόσον υπάρχει ένα έγκυρο βήμα. Στις κόκκινες θέσεις δεν επιτρέπεται καμία κίνηση (παράδειγμα: στην αρχική θέση είναι διαθέσιμη μόνο μία κίνηση προς τα κάτω, καθώς η κίνηση προς τα επάνω και προς τα αριστερά είναι μπλοκαρισμένη από τον τοίχο ενώ προς τα δεξιά υπάρχει κόκκινη θέση, επομένως δεν επιτρέπεται κίνηση).

Αρχικά θα δημιουργήσουμε την κλάση Node και μία βοηθητική συνάρτηση GetPath():

(1) Η Κλάση Node: που μπορεί να χρησιμοποιηθεί για τη δημιουργία ενός αντικειμένου για κάθε κόμβο με μεταβλητές, τον γονικό κόμβο, τρέχουσα θέση στο λαβύρινθο και τιμές κόστους (g, h & f).

(2) Η Συνάρτηση GetPath(): Έπειτα θα πρέπει να ορίσουμε μια συνάρτηση που θα επιστρέφει τη διαδρομή από τον κόμβο έναρξης προς τον τερματικό κόμβο.

(3) Θα δημιουργήσουμε μια συνάρτηση αναζήτησης που θα είναι η κυρίως λογική του αλγορίθμου:

(3.1) Αρχικοποιούμε όλες τις μεταβλητές.

(3.2) Προσθέτουμε τον κόμβο εκκίνησης στη λίστα "yetToVisit". Ορίζουμε μια κατάσταση διακοπής για να αποφύγουμε έναν ατέρμονα βρόχο. Καθορίζουμε την κίνηση σε σχέση με τον τρέχοντα κόμβο.

Επαναλαμβάνουμε τα παρακάτω έως ότου πληρούνται τα κριτήρια διακοπής:

(3.3) Αναζητούμε τον κόμβο με την χαμηλότερη τιμή της $\boldsymbol{f(n)}$ από την λίστα "yetToVisit". Αυτός ο κόμβος γίνεται ο τρέχον κόμβος. Ελέγχουμε επίσης τη μέγιστη τιμή επανάληψης που έχουμε ορίσει, αν έχει επιτευχθεί τερματίζουμε.

(3.4) Ελέγχουμε εάν ο τρέχον κόμβος είναι ίδιος με τον κόμβο στόχο (δηλαδή αν έχουμε βρει τη διαδρομή). Ο έλεγχος γίνεται με βάση τις συντεταγμένες των δύο κόμβων, αν είναι ίσες τότε είναι ο ίδιος κόμβος.

(3.5) Χρησιμοποιούμε τον τρέχοντα κόμβο και ελέγχουμε τους τέσσερις γειτονικούς του κόμβους. Αν δεν επιτρέπεται να κινηθούμε προς αυτόν ή αν βρίσκεται στη λίστα "visited", τον αγνοούμε. Διαφορετικά, δημιουργούμε νέο κόμβο με γονικό τον τρέχοντα κόμβο και ενημερώνουμε τη θέση του κόμβου.

(3.6) Ελέγχουμε όλους τους γειτονικούς κόμβους που δημιουργήθηκαν για να δούμε

• Εάν δεν βρίσκεται στην λίστα "yetToVisit", τον προσθέτουμε. Κάνουμε τον τρέχοντα κόμβο γονέα του γειτονικού και υπολογίζουμε τα κόστη f, g, και h του κόμβου.

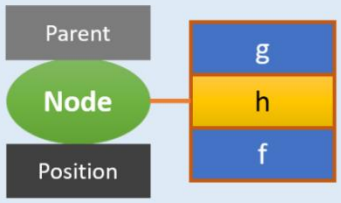
• Αν υπάρχει ήδη στον κατάλογο "yetToVisit", ελέγχουμε αν η διαδρομή προς αυτόν τον κόμβο είναι καλύτερη, χρησιμοποιώντας το κόστος g ως μέτρο. Ένα χαμηλότερο κόστος g σημαίνει ότι πρόκειται για μια καλύτερη πορεία. Αν ναι, αλλάζουμε τον γονέα του κόμβου στον τρέχον κόμβο και υπολογίζουμε εκ νέου τα αποτελέσματα g και f του κόμβου.

(4) Κυρίως πρόγραμμα: Θα ορίσουμε το λαβύρινθο, την αρχική και την τελική θέση. Στη συνέχεια θα χρησιμοποιήσουμε τη συνάρτηση αναζήτησης και εάν υπάρχει μια διαδρομή, τότε θα μπορούμε να την τυπώσουμε με την συνάρτηση GetPath().

Πρώτον, θα δημιουργήσουμε μια κλάση για έναν κόμβο που θα περιέχει όλα τα χαρακτηριστικά που σχετίζονται με τον κόμβο όπως ο γονέας του κόμβου, η θέση του κόμβου

και τα τρία κόστη (g, h & f) για τον κόμβο. Αρχικοποιούμε τον κόμβο και δημιουργούμε μια συνάρτηση για τον έλεγχο της ισότητας του κόμβου με έναν άλλο κόμβο.

```python
import numpy as np

class Node:
    """
        A node class for A* Pathfinding
        parent is parent of the current Node
        position is current position of the Node in the maze
        g is cost from start to current Node
        h is heuristic based estimated cost for current Node to end Node
        f is total cost of present node i.e. :  f = g + h
    """

    def __init__(self, parent=None, position=None):
        self.parent = parent
        self.position = position

        self.g = 0
        self.h = 0
        self.f = 0
    def __eq__(self, other):
        return self.position == other.position
```



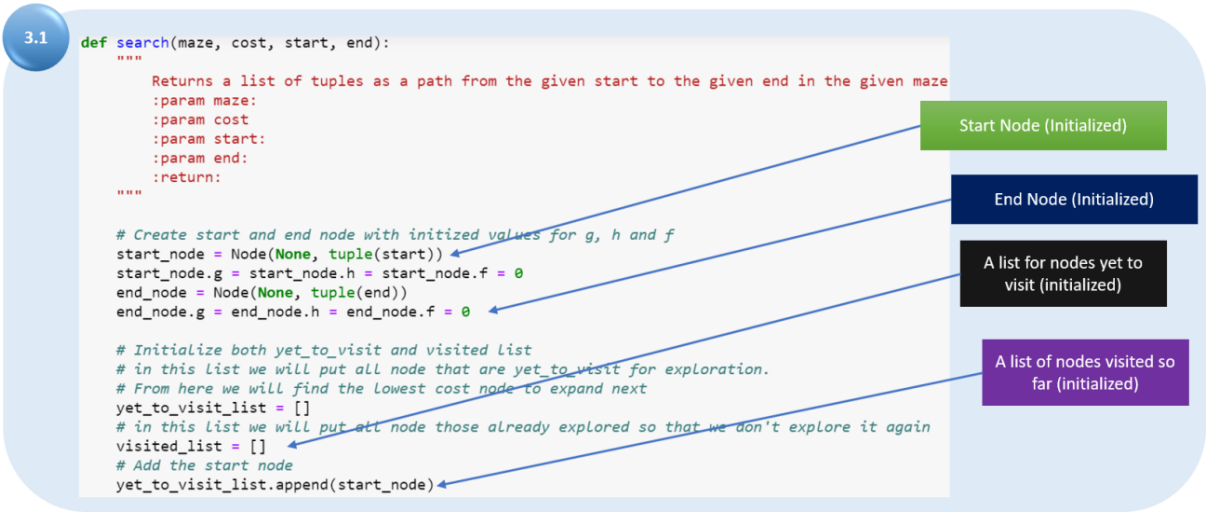
*Εικόνα 10: Η κλάση του κόμβου για την αναζήτηση A**

Τώρα θα φτιάξουμε τη συνάρτηση διαδρομής, η οποία θα χρησιμοποιηθεί για την επιστροφή της διαδρομής από τον κόμβο έναρξης στον κόμβο στόχου (end node).

```python
#This function return the path of the search
def return_path(current_node,maze):
    path = []
    no_rows, no_columns = np.shape(maze)
    # here we create the initialized result maze with -1 in every position
    result = [[-1 for i in range(no_columns)] for j in range(no_rows)]
    current = current_node
    while current is not None:
        path.append(current.position)
        current = current.parent
    # Return reversed path as we need to show from start to end path
    path = path[::-1]
    start_value = 0
    # we update the path of start to end found by A-star serch with every step incremented by 1
    for i in range(len(path)):
        result[path[i][0]][path[i][1]] = start_value
        start_value += 1
    return result
```

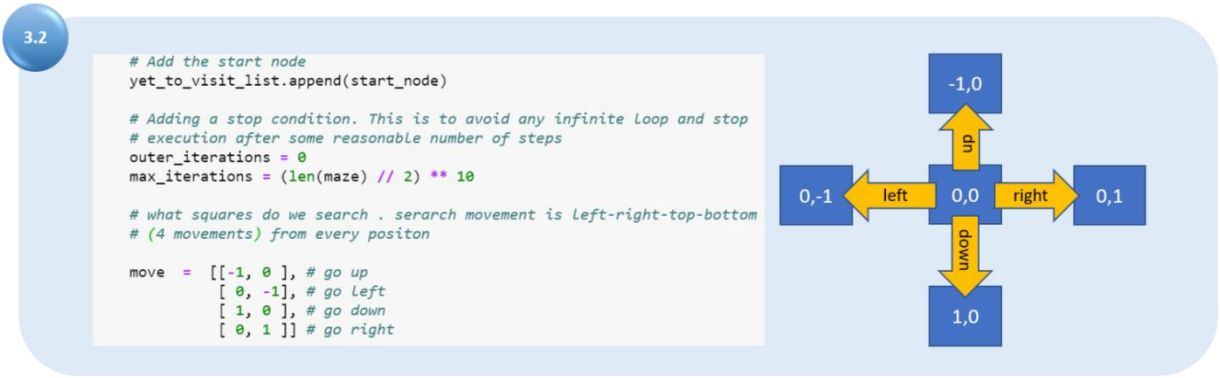
Εικόνα 11: Η συνάρτηση διαδρομής από τον τελικό κόμβο προς τον αρχικό

Τώρα θα ορίσουμε τη λειτουργία αναζήτησης, η οποία έχει πολλαπλά βήματα. Το πρώτο βήμα θα είναι να αρχικοποιήσουμε κόμβους και λίστες που θα χρησιμοποιήσουμε στη λειτουργία.



*Εικόνα 12: Αρχικοποίηση της αναζήτησης A**

Προσθέτουμε τον κόμβο εκκίνησης στη λίστα "yet_to_visit". Ορίζουμε μια κατάσταση διακοπής για να αποφευχθεί ένας ατέρμων βρόχος. Καθορίζουμε την σχετική κίνηση από τον τρέχοντα κόμβο, η οποία θα χρησιμοποιηθεί για την εύρεση των γειτονικών κόμβων.



Εικόνα 13: Κατάσταση διακοπής και ορισμός της κίνησης

Τώρα χρησιμοποιούμε ως τρέχοντα κόμβο τον κόμβο με την μικρότερη τιμή της f. Ελέγχουμε επίσης τη μέγιστη επανάληψη που έφτασε ή όχι, Ρύθμιση μηνύματος και διακοπή εκτέλεσης.

3.3

```
#find maze has got how many rows and columns
no_rows, no_columns = np.shape(maze)

# Loop until you find the end

while len(yet_to_visit_list) > 0:

    # Every time any node is referred from yet_to_visit list, counter of limit operation incremented
    outer_iterations += 1

    # Get the current node
    current_node = yet_to_visit_list[0]
    current_index = 0
    for index, item in enumerate(yet_to_visit_list):
        if item.f < current_node.f:
            current_node = item
            current_index = index

    # if we hit this point return the path such as it may be no solution or
    # computation cost is too high
    if outer_iterations > max_iterations:
        print ("giving up on pathfinding too many iterations")
        return return_path(current_node,maze)
```

*Εικόνα 14: Εκκίνηση της αναζήτησης A**

Καταργούμε τον τρέχοντα κόμβο από τη λίστα "yet_to_visit " και προσθέτουμε αυτόν τον κόμβο στη λίστα “visited”. Τώρα βάζουμε έναν έλεγχο αν βρεθεί ο κόμβος στόχος τότε καλούμε τη συνάρτηση επιστροφής διαδρομής και επιστρέφουμε.

3.4

```
# Pop current node out off yet_to_visit list, add to visited list
yet_to_visit_list.pop(current_index)
visited_list.append(current_node)

# test if goal is reached or not, if yes then return the path
if current_node == end_node:
    return return_path(current_node,maze)
```

Here we maintain which next node to visit and update visited list of nodes

Wow ...We have found the goal node . Terminate the search

Εικόνα 15: Επόμενος κόμβος και έλεγχος στόχου

Για τον τρέχοντα κόμβο, ανακαλύπτουμε όλους τους γειτονικούς (χρησιμοποιούμε όλες τις κινήσεις που μπορεί να κάνει ο κόμβος). Ο τρέχων ορίζεται ως πατρικός κόμβος στους γειτονικούς.

α) ελέγχουμε αν υπάρχει έγκυρη θέση (ο τοίχος περιορισμού θα κάνει κάποιες κινήσεις αδύνατες)

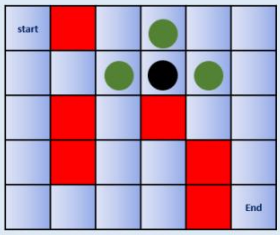
β) αν οποιαδήποτε θέση κόμβου είναι άκυρη (κόκκινη θέση) τότε αγνοείται

γ) προσθέτουμε στη λίστα όλους τους έγκυρους γειτονικούς κόμβους για τον επιλεγμένο γονέα

Εδώ στο διάγραμμα, δείχνουμε ότι ο κόμβος μαύρου κύκλου είναι ο τρέχων κόμβος και οι κόμβοι πράσινου κύκλου είναι οι αποδεκτοί γειτονικοί κόμβοι.

3.5

```
# Generate children from all adjacent squares
children = []

for new_position in move:

    # Get node position
    node_position = (current_node.position[0] + new_position[0],
                     current_node.position[1] + new_position[1])

    # Make sure within range (check if within maze boundary)
    if (node_position[0] > (no_rows - 1) or
        node_position[0] < 0 or
        node_position[1] > (no_columns -1) or
        node_position[1] < 0):
        continue

    # Make sure walkable terrain
    if maze[node_position[0]][node_position[1]] != 0:
        continue

    # Create new node
    new_node = Node(current_node, node_position)

    # Append
    children.append(new_node)
```



Εικόνα 16: Δημιουργία γειτονικών κόμβων

Για όλους τους γειτονικούς κόμβους:

α) εάν βρίσκεται στη λίστα “visited”, τότε τον αγνοούμε και δοκιμάζουμε τον επόμενο κόμβο.

β) υπολογίζουμε τις τιμές g, h, και f του γειτονικού κόμβου. Το ευρετικό κόστος h για την επίτευξη του κόμβου στόχου από τον τρέχοντα κόμβο υπολογίζεται χρησιμοποιώντας την Ευκλείδεια απόσταση (Hart, Nilsson, & Raphael, 1968).

γ) Εάν ο γειτονικός κόμβος βρίσκεται στη λίστα "yet_to_visit", τότε αγνοείται, αλλιώς προσθέτουμε τον γειτονικό κόμβο στη λίστα "yet_to_visit".

3.6

```
        # Loop through children
        for child in children:

            # Child is on the visited list (search entire visited list)
            if len([visited_child for visited_child in visited_list if visited_child == child]) > 0:
                continue

            # Create the f, g, and h values
            child.g = current_node.g + cost
            ## Heuristic costs calculated here, this is using eucledian distance
            child.h = (((child.position[0] - end_node.position[0]) ** 2) +
                       ((child.position[1] - end_node.position[1]) ** 2))

            child.f = child.g + child.h

            # Child is already in the yet_to_visit list and g cost is already lower
            if len([i for i in yet_to_visit_list if child == i and child.g > i.g]) > 0:
                continue

            # Add the child to the yet_to_visit list
            yet_to_visit_list.append(child)
```

Here Euclidean distance is used to calculate h cost. This is a major activity in real life problem to find appropriate heuristic.

Εικόνα 17: Εισαγωγή των γειτονικών κόμβων στην λίστα επίσκεψης

Τώρα θα τρέξουμε το πρόγραμμα ολοκληρωμένα για την επίλυση του προβλήματος βέλτιστης του λαβύρινθου και θα πάρουμε τη διαδρομή. Ανατρέξτε στη διαδρομή που εμφανίζεται χρησιμοποιώντας το κίτρινο βέλος.

4

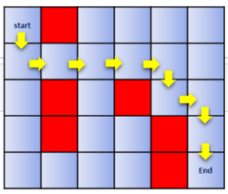
```
if __name__ == '__main__':

    maze = [[0, 1, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0],
            [0, 1, 0, 1, 0, 0],
            [0, 1, 0, 0, 1, 0],
            [0, 0, 0, 0, 1, 0]]

    start = [0, 0] # starting position
    end = [4,5] # ending position
    cost = 1 # cost per movement

    path = search(maze,cost, start, end)
    print(path)
```

```
[[0, -1, -1, -1, -1, -1], [1, 2, 3, 4, 5, -1], [-1, -1, -1, -1, 6, 7], [-1, -1, -1, -1, -1, 8], [-1, -1, -1, -1, -1, 9]]
```

Final result path

```
print('\n'.join(['  '.join(["{:" ">3d}".format(item) for item in row])
      for row in path]))
```

```
  0 -1 -1 -1 -1 -1
  1  2  3  4  5 -1
 -1 -1 -1 -1  6  7
 -1 -1 -1 -1 -1  8
 -1 -1 -1 -1 -1  9
```



*Εικόνα 18: Εκτέλεση αλγορίθμου A**

Συμπεράσματα

Ο A-star (A *) είναι ένας ισχυρός αλγόριθμος στην Τεχνητή Νοημοσύνη με ένα ευρύ φάσμα χρήσης. Ωστόσο, είναι τόσο καλός όσο και η ευρετική του συνάρτηση. Έχει βρει

εφαρμογές σε πολλά συστήματα λογισμικού, από τη Μηχανική Μάθηση και τη βελτιστοποίηση αναζήτησης, έως την ανάπτυξη παιχνιδιών όπου οι χαρακτήρες περιηγούνται σε σύνθετο έδαφος με εμπόδια για να φτάσουν στον στόχο τους (Zeng & Church, 2009). Επιλέχθηκε για χρήση λόγο της ταχύτητας του, χρησιμοποιεί ευρετική συνάρτηση και δεν δοκιμάζει όλους τους πιθανούς συνδυασμούς μέχρι να βρει το μονοπάτι με το ελάχιστο κόστος. Στην συγκεκριμένη διπλωματική εργασία θα τον χρησιμοποιήσουμε για την κατασκευή των καλωδιώσεων του κυκλώματος βρίσκοντας το μονοπάτι με το μικρότερο κόστος και σε δεύτερο στάδιο και με τις λιγότερες δυνατές αλλαγές κατεύθυνσης. Το δεύτερο στάδιο του αλγορίθμου χρησιμοποιείται μόνο όταν εξάγουμε ένα κύκλωμα και όχι συνεχώς, διότι προθέτει αρκετή χρονική πολυπλοκότητα στο πρόγραμμά μας. Στην θέση του ενός κόμβου του προηγούμενου παραδείγματος δημιουργούνται τέσσερις κόμβοι, που υποδηλώνουν και τις τέσσερις κατευθύνσεις που μπορεί να κινηθεί μία καλωδίωση.

Στο κεφάλαιο που ακολουθεί θα αναλύσουμε την λειτουργία του εργαλείου που δημιουργήθηκε και θα δείξουμε τμήματα όχι μόνο της σχεδίασης αλλά και της υλοποίησης του λογισμικού μέρους.

3. Υλοποίηση του λογισμικού

3.1 Προεπισκόπηση

Παρακάτω περιγράφονται τα βασικά βήματα εκτέλεσης του αλγορίθμου τα οποία θα αναλυθούν εκτενώς στη συνέχεια.

1ο βήμα - Αρχικοποίηση Προγράμματος: Σημείο εκκίνησης του προγράμματος, διάβασμα παραμέτρων από την γραμμή εντολών και αρχικοποίηση. Παράμετροι όπως: το αρχείο .dot με τα δεδομένα του κυκλώματος, το μέγεθος του πληθυσμού στον γενετικό αλγόριθμο (Ενότητα 3.3).

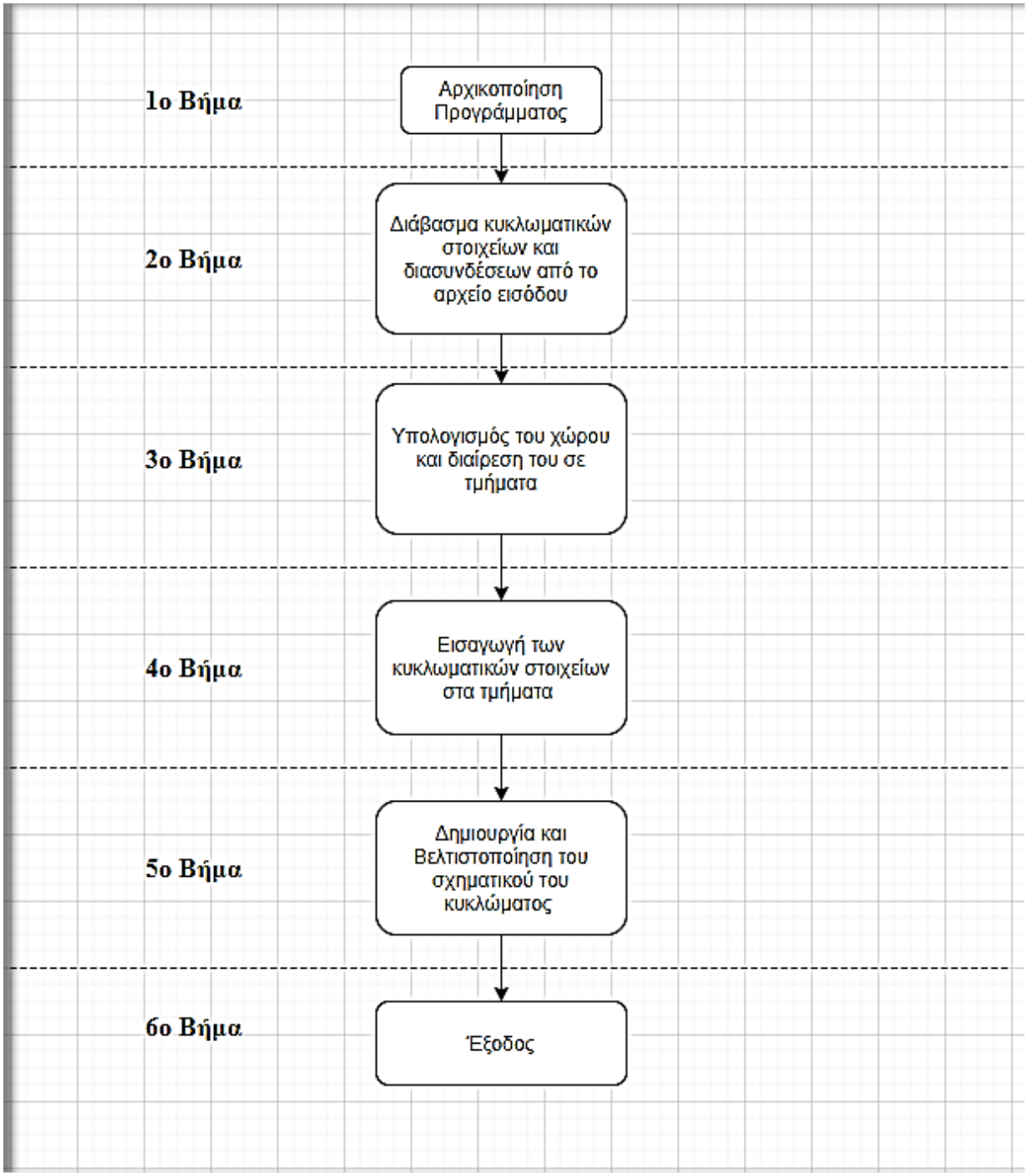
2ο βήμα - Διάβασμα κυκλωματικών στοιχείων και διασυνδέσεων από το αρχείο εισόδου: Σε αυτό το σημείο στο αρχείο εισόδου γίνεται διάσχιση, έλεγχος και δημιουργία των αντικειμένων (Ενότητα 3.4).

3ο βήμα - Εισαγωγή των κυκλωματικών στοιχείων στα τμήματα: Στη συνέχεια τοποθετούμε τα αντικείμενα στα τμήματα που δημιουργήθηκαν στο προηγούμενο βήμα (Ενότητα 3.5).

4ο βήμα - Υπολογισμός του χώρου και διαίρεση του σε τμήματα: Αφού έχουμε διαβάσει και γνωρίζουμε τα φυσικά χαρακτηριστικά των αντικειμένων, συνεχίζουμε με τον υπολογισμό του ελάχιστου δυνατού χώρου και την τμηματοποίηση του (Ενότητα 3.6).

5ο βήμα - Δημιουργία και Βελτιστοποίηση του σχηματικού του κυκλώματος: Σε αυτό το σημείο ξεκινάει το κυρίως τμήμα του αλγορίθμου. Δημιουργείται ο αρχικός πληθυσμός του γενετικού αλγόριθμου και ξεκινούν οι επαναλήψεις βελτιστοποίησης (ορίζεται στο 1ο βήμα για πόσες γενιές θέλουμε να εκτελεστεί ο γενετικός αλγόριθμος) (Ενότητα 3.7).

6ο βήμα - Έξοδος Προγράμματος: Όταν ολοκληρωθεί η προηγούμενη διαδικασία τότε το πρόγραμμα τερματίζει και έχει δημιουργήσει ένα αρχείο .circ συμβατό με το Logisim το οποίο περιέχει ένα ψηφιακό κύκλωμα το οποίο είναι πολύ κοντά στο βέλτιστο κύκλωμα που θα μπορούσε να δημιουργηθεί. Αυτό προϋποθέτει βέβαια ο αριθμός των γενεών να επαρκεί.

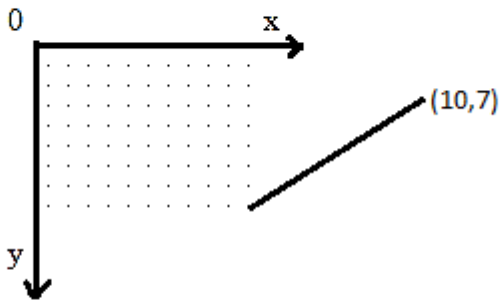


Εικόνα 19: Διάγραμμα ροής του αλγόριθμου βελτιστοποίησης σχηματικών ψηφιακών κυκλωμάτων

3.2 Περιγραφή Περιβάλλοντος

3.2.1 Περιγραφή του Χώρου

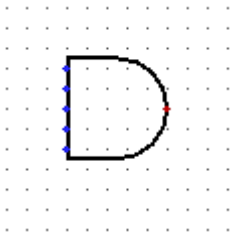
Ο χώρος του περιβάλλοντος μας είναι δισδιάστατος και διακριτός. Οπότε μπορεί να αναπαρασταθεί με δύο φυσικούς αριθμούς (x, y). Επιλέχθηκε αυτή η φορά, διότι αυτή χρησιμοποιείται και στο Logisim. Το σημείο (0,0) βρίσκεται στην επάνω αριστερή γωνία και ο καμβάς μεγαλώνει ελεύθερα προς τα κάτω και δεξιά.



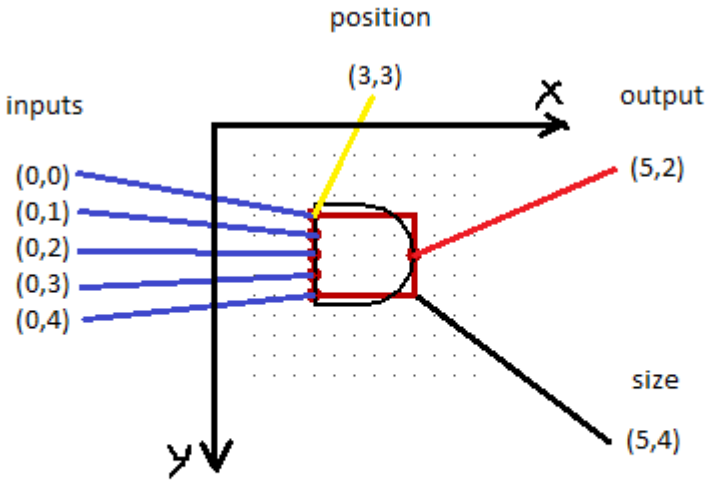
Εικόνα 20: Περιγραφή του δισδιάστατου χώρου. Ο παραπάνω χώρος αναπαριστάται ως (10, 7)

3.2.2 Περιγραφή Αντικειμένων

Τα αντικείμενα για ευκολία θεωρούμε ότι είναι ορθογώνια παραλληλόγραμμα. Οπότε τα αναπαριστούμε με τον ίδιο τρόπο.



Εικόνα 21: Η Πύλη AND στο Logisim. Το μέγεθος της παραπάνω πύλης AND των 5 εισόδων αναπαριστάται ως (5, 4)



Εικόνα 22: Αναπαράσταση της Πύλης AND

Για την θέση του στοιχείου χρησιμοποιείται πάντα το επάνω αριστερά σημείο του στοιχείου, είτε είναι pin είτε όχι.

Η κόκκινη περιοχή μας δείχνει την συνολική αναπαράσταση της πύλης AND: το μέγεθός της, τη θέση της, τα σημεία που υπάρχουν PINS εισόδου και εξόδου, όπως και τον προσανατολισμό της:

Οπότε στην Εικόνα 22 έχουμε μία πύλη AND στο σημείο (3, 3)

Με μέγεθος: (5, 4)

PINS εισόδου: (0, 0), (0, 1), (0, 2), (0, 3), (0, 4)
PIN εξόδου: (5, 2)

Προσανατολισμός: Ανατολικός (ορίζεται στο Logisim)

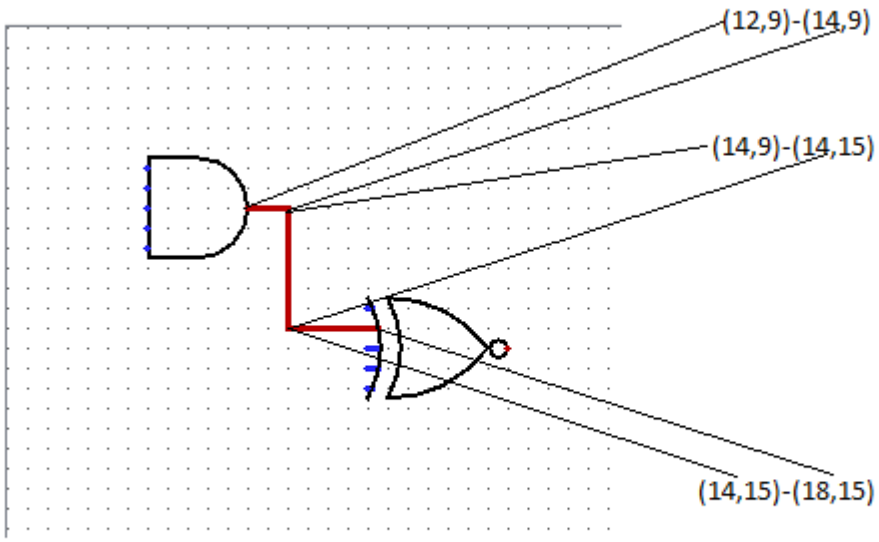
Εδώ να σημειώσουμε πως οι είσοδοι και έξοδοι των στοιχείων μπορούν να βρίσκονται μόνο πάνω στις κουκίδες του καμβά. Ενδιάμεσα από τις κουκίδες δεν υπάρχει τίποτα.

Επίσης, το σημείο (0, 0) για τον ορισμό του χώρου αλλά και για τα αντικείμενα είναι η επάνω αριστερή γωνία. Ο λόγος είναι ότι μας διευκολύνει στο να έχουμε κοινές αναπαραστάσεις με το Logisim.

Ο προσανατολισμός των αντικειμένων κατά την είσοδο είναι πάντα **Ανατολικός**

3.2.3 Περιγραφή Καλωδίωσης

Η καλωδίωση στο Logisim επιτρέπει μόνο οριζόντιες και κάθετες κινήσεις. Ένα καλώδιο ορίζεται από πολλά ευθύγραμμα τμήματα όπως φαίνεται και στην Εικόνα 23. Στις άκρες των ευθύγραμμων αυτών τμημάτων εάν υπάρχει σημείο άλλου ευθύγραμμου τμήματος ή PIN εισόδου-εξόδου τότε γίνεται διασύνδεσή τους.



Εικόνα 23: Περιγραφή της καλωδίωσης στο Logisim

```
<wire from="(12,9)" to="(14,9)"/>
```

```
<wire from="(14,9)" to="(14,15)"/>
<wire from="(14,15)" to="(18,15)"/>
```

3.3 Αρχικοποίηση Προγράμματος

Πρόκειται για ένα εργαλείο που εκτελείται από τη γραμμή εντολών. Όλοι οι παράμετροι είναι προαιρετικοί εκτός από το *Input Filename* που δέχεται το αρχείου εισόδου που είναι υποχρεωτικό.

Οι παράμετροι που δέχεται το πρόγραμμα μας είναι:

Input Filename
-FI:[string]: Το .dot αρχείο εισόδου με τα κυκλωματικά στοιχεία και τις διασυνδέσεις του κυκλώματος. Παρακάτω θα αναφερθούμε αναλυτικότερα στο περιεχόμενο του αρχείου.

Output Filename
-OF:[string]: Το επιθυμητό όνομα του αρχείου εξόδου.

Library
-LB:[Library]: Το αρχείο (.circ) που λειτουργεί σαν βιβλιοθήκη και μπορούμε να επαναχρησιμοποιήσουμε κυκλώματα μέσα στο νέο κύκλωμα (κυκλώματα από κυκλώματα δηλαδή). Όταν χρησιμοποιείται βιβλιοθήκη, τότε το εξαγόμενο κύκλωμα προστίθεται εκεί.

Circuit Name
-CN:[Circuit Name]: Το επιθυμητό όνομα του κυκλώματος. Όταν χρησιμοποιείται βιβλιοθήκη, τότε το Circuit Name γίνεται υποχρεωτικό.

Canvas Padding
-CP:[50>integer >= 2]: Ο διακόπτης αυτός περιγράφει το εσωτερικό περιθώριο του συνολικού χώρου. Παίρνει ακέραιες τιμές μεγαλύτερες του δύο. Το δύο ορίζεται ως ελάχιστο περιθώριο, έτσι ώστε ένα κυκλωματικό στοιχείο να μην μπορεί να τοποθετηθεί στα όρια του χώρου για να μπορούν να περνάνε οι καλωδιώσεις.
Part Padding

-PP:[50>integer >= 0]: Ο διακόπτης αυτός περιγράφει το εξωτερικό περιθώριο του συνολικού χώρου. Παίρνει ακέραιες τιμές μεγαλύτερες του μηδενός.

Object Margin

-OM:[20>integer >= 1]: Ο διακόπτης αυτός περιγράφει το εξωτερικό περιθώριο του αντικειμένου. Παίρνει ακέραιες τιμές μεγαλύτερες του ένα, λόγο καλωδιώσεων πάλι.

Number of Generations

-NG:[1.000.000>integer > 0]: Ο διακόπτης αυτός περιγράφει τον αριθμό των γενεών που θέλουμε να φτάσει ο γενετικός αλγόριθμος. Παίρνει ακέραιες τιμές μεγαλύτερες του μηδενός.

Elements of Population

-EP:[1.000.000>integer > 0]: Ο διακόπτης αυτός περιγράφει τον αριθμό των στοιχείων του πληθυσμού μιας γενιάς. Στο δικό μας πρόβλημα κάθε στοιχείο του πληθυσμού είναι ένα ψηφιακό κύκλωμα. Παίρνει ακέραιες τιμές μεγαλύτερες του μηδενός.

Number of Threads

-TH:[500>integer > 0]: Ο αριθμός των νημάτων που θέλουμε να τρέξει το πρόγραμμα. Παίρνει ακέραιες τιμές μεγαλύτερες του μηδενός.

Minimum Wire Length

-MI:[100> integer > 1]: Περιγράφει το ελάχιστο μήκος καλωδίων για την αξιολόγηση του κυκλώματος.

Maximum Wire Length

-MA:[500> integer > 1]: Περιγράφει το μέγιστο μήκος καλωδίων για την αξιολόγηση του κυκλώματος.

Crossover Rate

-CR:[1> double > 0]: Περιγράφει την πιθανότητα διακλάδωσης των γονέων κατά την αναπαραγωγή στον γενετικό αλγόριθμο.

Mutation Rate

-MU:[1> double > 0]: Περιγράφει την πιθανότητα μετάλλαξης ενός γονιδίου κατά την διασταύρωση στον γενετικό αλγόριθμο.

3.4 Αρχείο Εισόδου

Στο αρχείο εισόδου έχουμε δύο τμήματα που διαχωρίζονται μεταξύ τους με ετικέτες. Στο πρώτο τμήμα ορίζονται τα κυκλωματικά στοιχεία και στο δεύτερο οι διασυνδέσεις. Η σειρά είναι αυστηρή διότι οι διασυνδέσεις σχετίζονται με το είδος και το πλήθος των στοιχείων. Κάθε ένα από τα παραπάνω ορίζεται σε ξεχωριστή σειρά.

3.4.1 Περιγραφή εισόδου Κυκλωματικών Στοιχείων στο αρχείο .dot

Για να ορίσουμε ένα κυκλωματικό στοιχείο εισάγουμε το όνομα του και μετά μέσα σε αγκύλες τα χαρακτηριστικά του, τα οποία είναι προαιρετικά. Τα χαρακτηριστικά διαχωρίζονται μεταξύ τους με υποδιαστολή. Αν δεν οριστούν χαρακτηριστικά τότε ενεργοποιούνται οι προεπιλεγμένες τιμές τους που είναι αυτές που ορίζονται και στο Logisim σαν προεπιλογή, εκτός από το χαρακτηριστικό **inputs** (αριθμός εισόδων) που η προεπιλεγμένη τιμή για το συγκεκριμένο λογισμικό είναι η μικρότερη αποδεκτή τιμή ενώ για το logisim είναι κάποια ενδιάμεση. Με τον διακόπτη –LB μπορούμε να εισάγουμε μία βιβλιοθήκη κυκλωμάτων τα οποία μπορούμε να χρησιμοποιήσουμε στο αρχείο με το αντίστοιχο όνομα του κυκλώματος.

Π.χ. Αν θέλουμε να ορίσουμε μια πύλη AND με δύο εισόδους, μετά μία NOR με πέντε εισόδους και μετά ένα custom κύκλωμα με όνομα «CustomCircuitFromLirary» που φτιάξαμε και βάλαμε στην βιβλιοθήκη, τότε θα γράψουμε στο αρχείο .dot

```
//1. Components

AndGate

NorGate[inputs=5]

CustomCircuitFromLirary
```

Με τη σειρά που εισάγουμε τα στοιχεία ορίζονται και τα αναγνωριστικά τους. Στο παραπάνω παράδειγμα η πύλη And είναι το στοιχείο 0 και η πύλη Nor το στοιχείο 1 κ.ο.κ. Αυτό θα μας χρησιμεύσει παρακάτω για να ορίσουμε τις διασυνδέσεις αυτών.

Παρακάτω θα βρείτε την πλήρη λίστα των αποδεκτών κυκλωματικών στοιχείων με τις ονομασίες τους και τα χαρακτηριστικά τους. Οι ονομασίες των στοιχείων θα πρέπει να εισάγονται ακριβώς όπως είναι στην αναλυτική λίστα.

3.4.2 Περιγραφή εισόδου Διασυνδέσεων

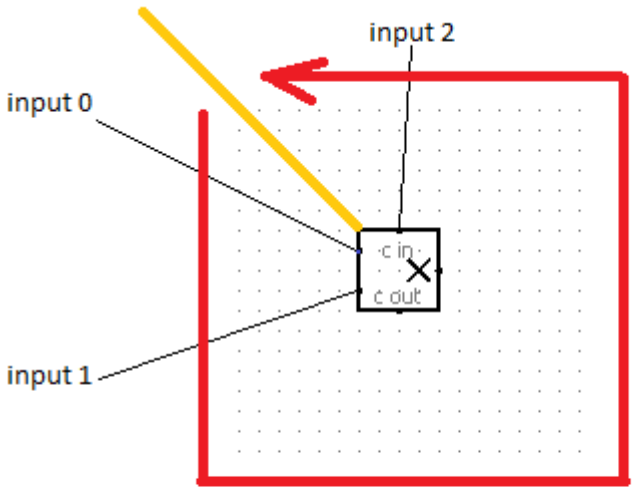
Μια διασύνδεση ορίζεται ως η έξοδος ενός στοιχείου με την είσοδο ενός άλλου.

```
outport{Κυκλωματικό-Στοιχείο:PIN-Εξόδου}\inport{Κυκλωματικό-Στοιχείο:PIN-Εισόδου}
```

Όπως ορίσαμε αναγνωριστικά στα στοιχεία, έτσι θα πρέπει να βρούμε και έναν τρόπο να αναγνωρίζουμε και τις εισόδους-εξόδους τους. Ένας τρόπος είναι ο παρακάτω.

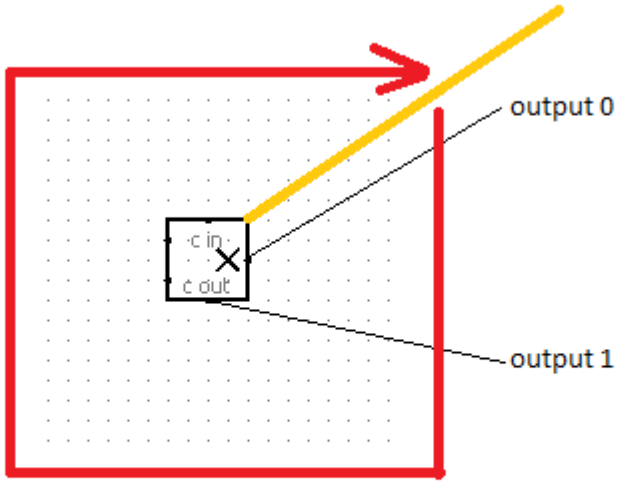
Οι είσοδοι και έξοδοι έχουν ξεχωριστή αρίθμηση γιατί πρέπει να γνωρίζουμε το κάθε στοιχείο πόσες εισόδους και εξόδους έχει.

Όπως προείπαμε ο προσανατολισμός των στοιχείων κατά την είσοδο στο αρχείο .dot θεωρείται ότι είναι πάντα ανατολικός. Οπότε για την αρίθμηση των εισόδων ξεκινάμε από την επάνω αριστερή γωνία του στοιχείου και με φορά αντίστροφη των δεικτών του ρολογιού. Η αρίθμηση ξεκινάει από το μηδέν.



Εικόνα 24: Παράδειγμα αναγνωριστικών εισόδων ενός Multiplier

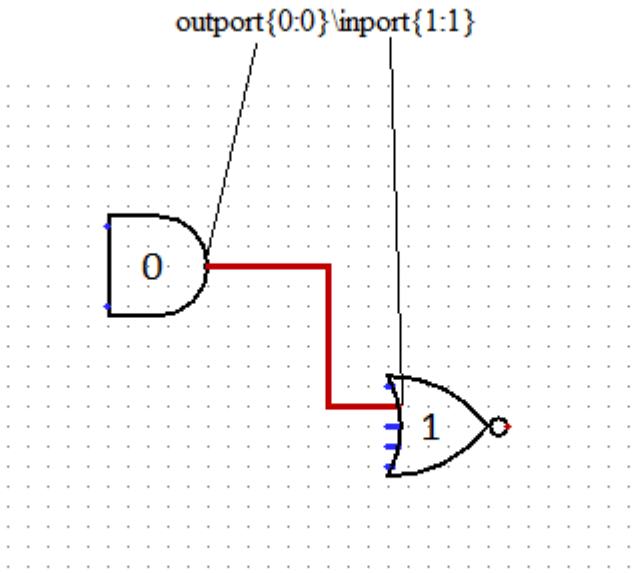
Για την αρίθμηση των εξόδων ξεκινάμε από την επάνω δεξιά γωνία του στοιχείου και με φορά ίδια με αυτή των δεικτών του ρολογιού. Η αρίθμηση ξεκινάει από το μηδέν.



Εικόνα 25: Παράδειγμα αναγνωριστικών εξόδων ενός Multiplier

Οπότε στο προηγούμενο παράδειγμα που ορίσαμε την And και Nor στο αρχείο εισόδου. Αν θέλουμε να συνδέσουμε την έξοδο της And με την δεύτερη είσοδο της Nor, τότε στο τμήμα των διασυνδέσεων θα γράψουμε

```
//2. Interconnections
outport{0:0}\inport{1:1}
```

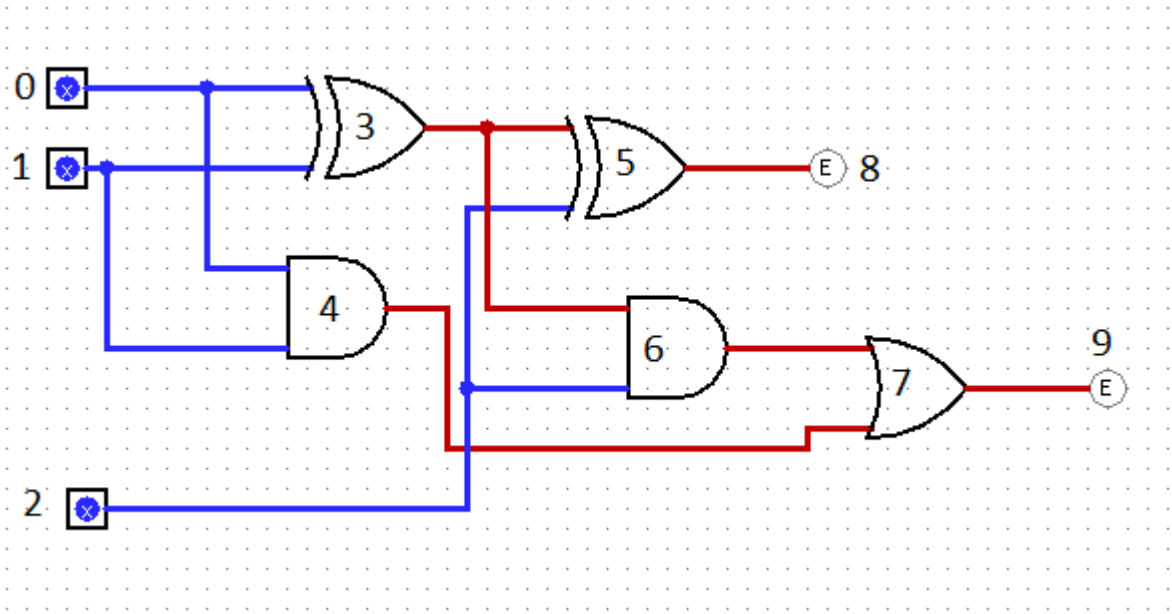


Εικόνα 26: Παράδειγμα διασύνδεσης δύο κυκλωματικών στοιχείων

Παρακάτω παρατίθεται το αρχείο εισόδου ενός πλήρη αθροιστή

```
//1. Components
Pin
Pin
Pin
XorGate[label=xorGateLabel]
AndGate
XorGate
AndGate
OrGate
Probe
Probe[label=probeLabel]


//2. Interconnections
outport{0:0}\inport{3:0}
outport{1:0}\inport{3:1}
outport{0:0}\inport{4:0}
outport{1:0}\inport{4:1}
outport{2:0}\inport{6:1}
outport{3:0}\inport{5:0}
outport{4:0}\inport{7:1}
outport{3:0}\inport{6:0}
outport{2:0}\inport{5:1}
outport{5:0}\inport{8:0}
outport{6:0}\inport{7:0}
outport{7:0}\inport{9:0}

//3. End
```

Εικόνα 27: επάνω: Αρχείο εισόδου ενός πλήρη αθροιστή, κάτω: σχηματικό Πλήρη αθροιστή με τα αναγνωριστικά των στοιχείων



3.5 Εισαγωγή των στοιχείων σε τμήματα

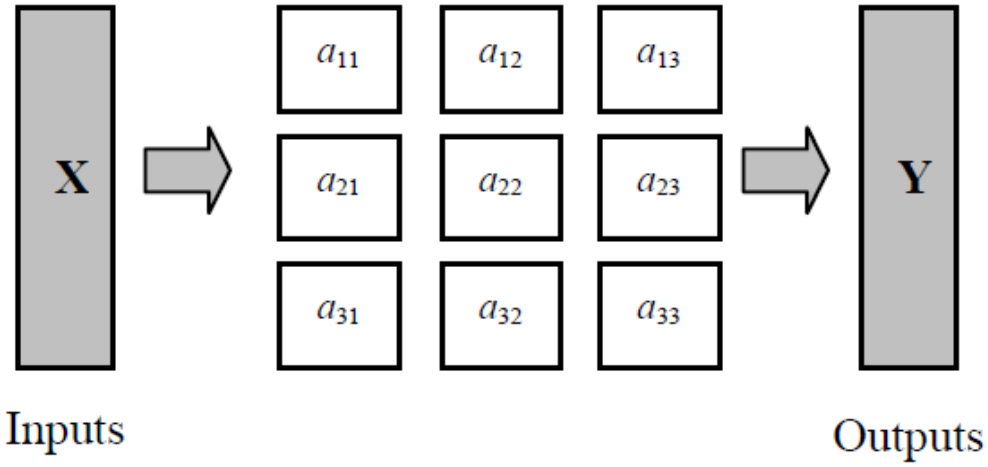
Η εισαγωγή των στοιχείων σε τμήματα γίνεται με έναν αλγόριθμο ελάχιστων αλμάτων. Εδώ να επισημάνουμε πως γνωρίζουμε για κάθε στοιχείο αν είναι στοιχείο εισόδου, ενδιάμεσο ή στοιχείο εξόδου. Στοιχείο εισόδου σημαίνει ότι έχει μόνο output pins, ενδιάμεσο σημαίνει ότι έχει και input και output pins ενώ τα εξόδου έχουν μόνο input pins.

Ξεκινάμε με τα στοιχεία εισόδου και τα τοποθετούμε όλα στο πρώτο group. Τα στοιχεία εισόδου βρίσκονται στον αριθμό αλμάτων μηδέν. Έπειτα για κάθε ένα από αυτά βλέπουμε με ποιο στοιχείο συνδέεται και αν δεν έχει ήδη εισαχθεί σε κάποιο άλλο τμήμα τότε το τοποθετούμε στο επόμενο τμήμα (βλέπε Ψευδοκώδικας 1). Εδώ αξίζει να τονίσουμε πως αν ένα στοιχείο μπορεί να βρεθεί με περισσότερους τρόπους αλμάτων τότε θα προτιμηθεί ο ελάχιστος αριθμός αλμάτων.

```
//put objects in groups, 0 hops (inputs) - 1 hop...

Groups add all_inputs

Foreach element in Group
        Get_connections
        If connected_element_is_not_inserted
        Insert_it_in_NewGroup
```

Ψευδοκώδικας 1: Εισαγωγή των στοιχείων σε τμήματα

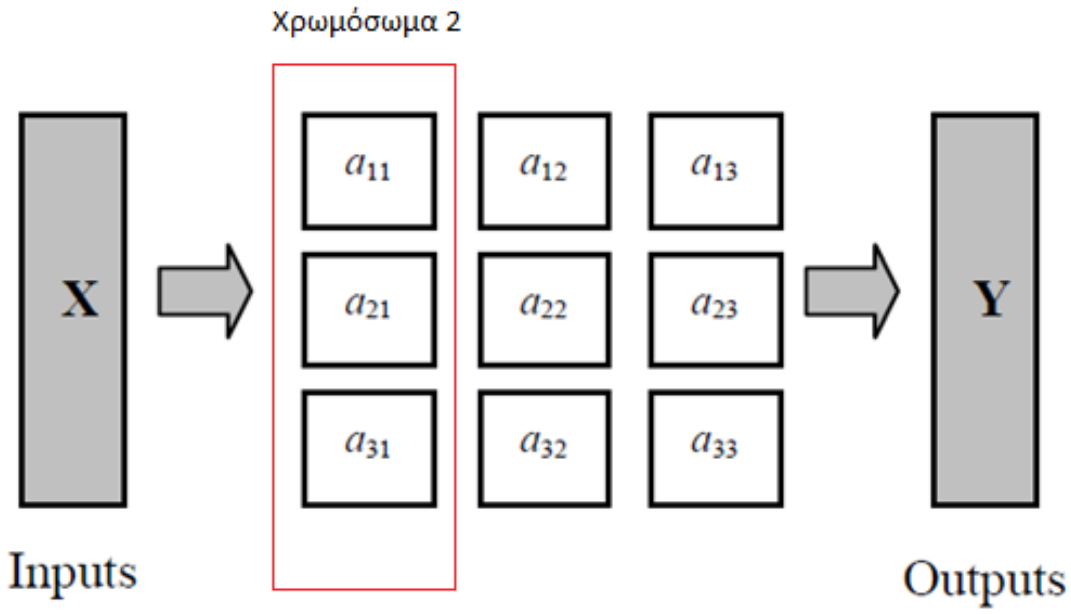
3.6 Τμηματοποίηση του χώρου και υπολογισμός των φυσικών μεγεθών



Εικόνα 28: Τμηματοποίηση του χώρου

Ένα οποιοδήποτε ψηφιακό κύκλωμα μπορεί να θεωρηθεί ότι έχει το παραπάνω σχήμα.

Γι αυτό το λόγο η τμηματοποίηση του χώρου γίνεται σε κάθετα ορθογώνια παραλληλόγραμμα. Ο λόγος είναι ότι αυτό θα μας βοηθήσει αργότερα στο να ορίσουμε τα χρωμοσώματα του γενετικού αλγορίθμου και καθώς ανταλλάζουμε χρωμοσώματα μεταξύ των κυκλωμάτων κατά την διαδικασία της διασταύρωσης, τα στοιχεία τους δεν θα συμπίπτουν από άποψη θέσης (δηλαδή δεν θα πέσει το ένα πάνω στο άλλο) και επιπλέον μετά την αλλαγή θα συνεχίσουμε να έχουμε το ίδιο πλήθος και είδος των στοιχείων. Τα τμήματα κατά τον άξονα x είναι μεταβλητού μεγέθους ανάλογα με τις διαστάσεις των στοιχείων που περιέχουν, ενώ κατά τον άξονα y όλα τα τμήματα έχουν το ίδιο μέγεθος. Σκοπός μας εδώ είναι να εξασφαλίσουμε ότι δίνοντας τυχαίες θέσεις στα στοιχεία, θα πρέπει ο χώρος να επαρκεί έτσι ώστε, όλα τα στοιχεία να μπορούν να εισαχθούν στα τμήματα.



Εικόνα 29: Ορισμός του χρωμοσώματος στο ψηφιακό κύκλωμα

Οπότε, για να βρούμε το ελάχιστο x του κάθε τμήματος αρκεί να βρούμε την μεγαλύτερη πλευρά από όλα τα στοιχεία που περιέχονται στο συγκεκριμένο τμήμα. Εδώ να σημειώσουμε ότι ο προσανατολισμός των στοιχείων μπορεί να αλλάζει κατά την λειτουργία. Οπότε για να βρούμε την μεγαλύτερη πλευρά θα πρέπει να λάβουμε υπόψη μας όλες τις πλευρές των στοιχείων και όχι μόνο την x πλευρά τους.

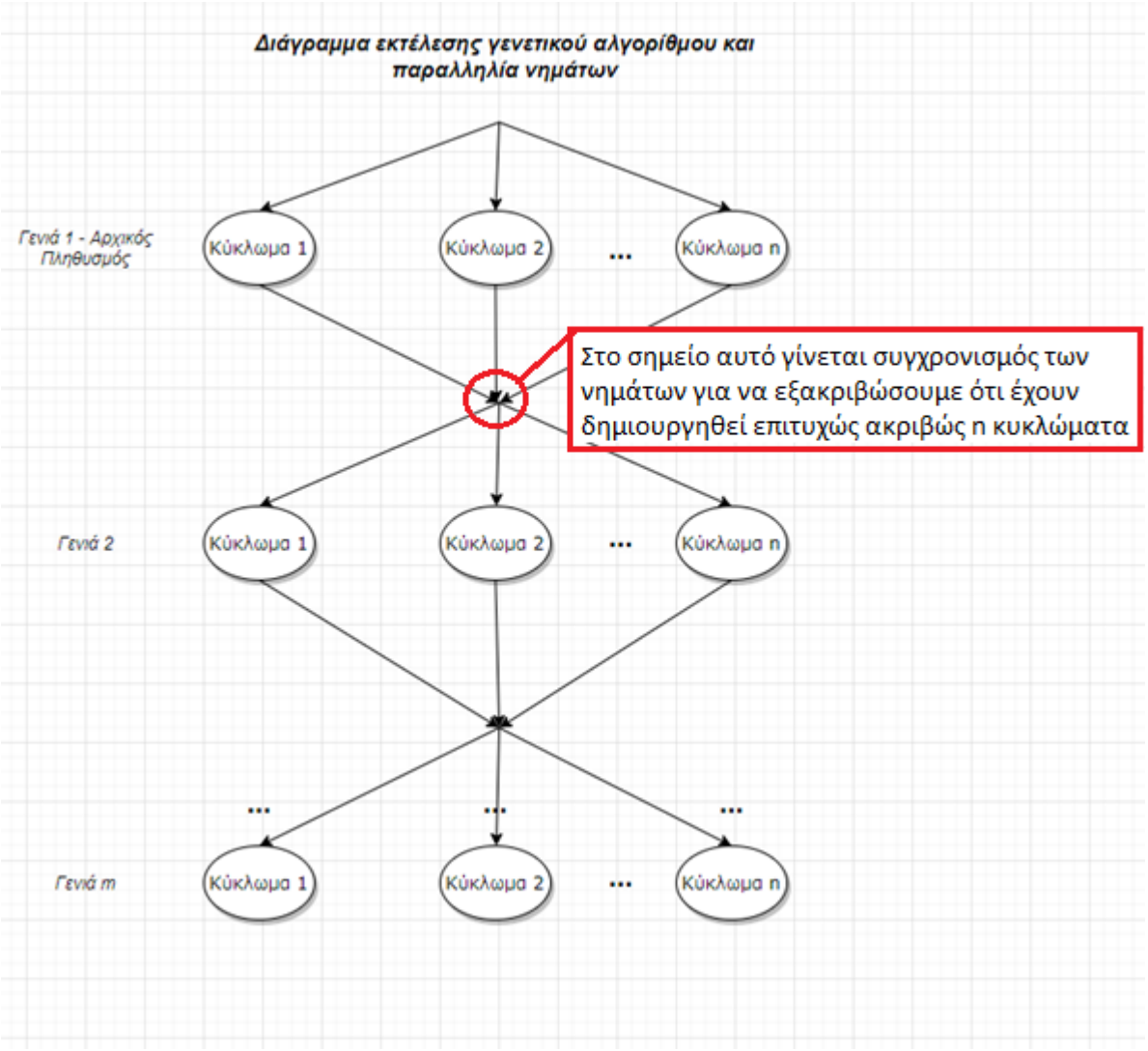
Για να βρούμε την y πλευρά που είναι κοινή για όλα τα τμήματα, θα υπολογίσουμε για όλα τα τμήματα το άθροισμα των μεγαλύτερων πλευρών των στοιχείων και θα το πολλαπλασιάσουμε επί δύο. Ο λόγος που πολλαπλασιάζεται επί δύο είναι ότι μόνο έτσι εξασφαλίζεται ότι καθώς δίνουμε τυχαίες θέσεις στα στοιχεία θα χωρέσουν όλα στο τμήμα.

Το y θα μπορούσε να υπολογιστεί όπως και το x του τμήματος αλλά θα έτσι θα περιορίζαμε κατά πολύ την ελευθερία του γενετικού αλγορίθμου και τα κυκλώματα που θα μας έδινε θα ήταν κατά έναν μεγάλο βαθμό ίδια. Ο λόγος που το y είναι κοινό για όλα τα τμήματα είναι να δώσουμε ελευθερία στον γενετικό αλγόριθμο κατά τον άξονα y, έτσι ώστε να υπάρχει ποικιλομορφία στα αποτελέσματα, το οποίο μεταφράζεται και σαν μεγαλύτερη πιθανότητα να έχει δημιουργηθεί ένα καλύτερο κύκλωμα σε μία γενιά κυκλωμάτων. Γενικά,

θέλουμε να δώσουμε αρκετή ελευθερία στον γενετικό αλγόριθμο, έτσι ώστε να μπορεί να βρει την βέλτιστη λύση αλλά όχι περισσότερη από όσο πρέπει, διότι θα κάνει περισσότερες άσκοπες δοκιμές και θα έχει αυξημένη χρονική πολυπλοκότητα.

3.7 Γενετικός αλγόριθμος και βελτιστοποίηση

3.7.1 Αρχικός Πληθυσμός



Εικόνα 30: Διάγραμμα εκτέλεσης γενετικού αλγορίθμου

3.7.2 Εισαγωγή των στοιχείων στον καμβά

Ξεκινώντας, δημιουργούμε τον αρχικό πληθυσμό των κυκλωμάτων, των αριθμό των οποίων έχουμε ορίσει στην είσοδο. Για κάθε κύκλωμα εισάγουμε τα στοιχεία στον καμβά με τυχαίο τρόπο.

Τα στοιχεία έχουν κατανεμηθεί σε τμήματα σε προηγούμενο βήμα

```
Foreach group in Groups
Foreach object in group
        inserted = false
```

Ψευδοκώδικας 2: Για κάθε κυκλωματικό στοιχείο

Δίνουμε τυχαίο προσανατολισμό σε όσα στοιχεία επιτρέπουν την αλλαγή του προσανατολισμού.

```
If OrientationCanChange
        Change_objects_orientation_randomly
```

Ψευδοκώδικας 3: Δίνουμε τυχαίο προσανατολισμό

Δημιουργούμε τυχαία x, y μέχρι να εισάγουμε το στοιχείο στον καμβά, λαμβάνοντας πάντα υπόψη το τμήμα στο οποίο βρισκόμαστε, το μέγεθος του συγκεκριμένου στοιχείου και το επιθυμητό περιθώριο που έχουμε ορίσει στη είσοδο.

```
While !inserted
    x = random(g.maxX,g.Get_minX)
    y = random(g.maxY,g.Get_minY)
```

Ψευδοκώδικας 4: Δίνουμε τυχαία θέση

Όταν βρεθεί διαθέσιμη θέση γίνεται εισαγωγή.

```
If Position_is_available
    Insert_object_into_canvas(obj);
```

Ψευδοκώδικας 5: Αν υπάρχει διαθέσιμη θέση κάνε εισαγωγή

3.7.3 Καλωδίωση

3.7.3.1 Εύρεση σημείου έναρξης

Για κάθε διασύνδεση, γίνεται έλεγχος αν από το output pin υπάρχουν άλλες καλωδιώσεις. Αν υπάρχουν τότε υπολογίζεται η ευκλείδεια απόσταση από όλα τα σημεία των καλωδιώσεων και σαν σημείο έναρξης ορίζεται αυτό με την μικρότερη απόσταση από το σημείο που βρίσκεται το input pin.

```
Foreach wire
        If it_is_connected_to_source

        Get_all_points
```

Ψευδοκώδικας 6: Εύρεση ήδη υπάρχουσας καλωδίωσης και εισαγωγή όλων των σημείων σε λίστα.

```
SortPointsByGoalDist(ArrayList<Point> po, Point goal)
```

Ψευδοκώδικας 7: Ταξινόμηση όλων των σημείων με βάση την μικρότερη απόσταση από τον στόχο.

Αν η καλωδίωση αποτύχει ή δεν βρεθεί η υπάρχουσα χρησιμοποιούμε το επόμενο σημείο έναρξης με την μικρότερη απόσταση.

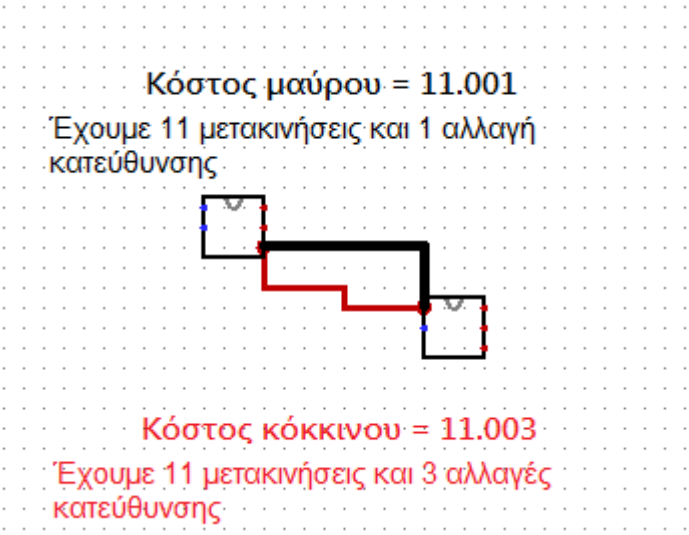
3.7.3.2 Καλωδίωση από σημείο σε σημείο

Αφού τοποθετηθούν τα στοιχεία στον καμβά και αποφασίσουμε για τα σημεία έναρξης και σημείο στόχου τότε δημιουργούμε την καλωδίωση. Η καλωδίωση γίνεται σύμφωνα με τον αλγόριθμο A*. Ο αλγόριθμος A* είναι δύο σταδίων. Το πρώτο στάδιο είναι αυτό της εύρεσης συντομότερης διαδρομής μεταξύ δύο σημείων που αναφερθήκαμε και στο Κεφ. 2.7. Το δεύτερο στάδιο είναι με τις λιγότερες δυνατές αλλαγές κατεύθυνσης. Το δεύτερο στάδιο επειδή προσθέτει χρονική και χωρική πολυπλοκότητα εφαρμόζεται μόνο στα κυκλώματα με την καλύτερη τιμή της συνάρτησης αξιολόγησης για λόγους αισθητικής έτσι ώστε, να είναι ευδιάκριτα τα κυκλώματα εξόδου.

Η κυριότερη διαφορά για τον υπολογισμό της συντομότερης διαδρομής με τις λιγότερες δυνατές στροφές, είναι ότι στη θέση του ενός κόμβου που αναφερθήκαμε προηγουμένως δημιουργούμε τέσσερις. Ο καθένας εκ των οποίων αντιστοιχίζεται και σε μία κατεύθυνση. Τέσσερις είναι και οι κινήσεις που μας επιτρέπει και το Logisim. Η λογική είναι πως για το κόστος εύρεσης του μονοπατιού χρησιμοποιούμε ακέραιους αριθμούς για την μετακίνηση από κόμβο σε κόμβο ενώ δεκαδικούς για την αλλαγή κατεύθυνσης. Συγκεκριμένα, το κόστος για την μεταφορά σε γειτονικό κόμβο έχει οριστεί σε μονάδα ενώ για την αλλαγή κατεύθυνσης το 0,001 (βλέπε Εικόνα 31). Άρα λαμβάνεται πρώτα υπόψη το κόστος της μεταφοράς και έπειτα το πόσες αλλαγές κατεύθυνσης πραγματοποίησε κατά την διάρκεια. Σε αυτό το σημείο να αναφέρουμε πως η καλωδίωση μπορεί να αποτύχει να δημιουργηθεί λόγο της τυχαιότητας των θέσεων των στοιχείων. Μπορεί δηλαδή μία συγκεκριμένη τοποθέτηση ή ήδη υπάρχουσα καλωδίωση να έχει αποκλείσει τελείως ένα pin και ο τρόπος διασύνδεσης να μην είναι δυνατός Σε αυτήν την περίπτωση το κύκλωμα αφαιρείται και προχωράμε σε δημιουργία νέου στη θέση του.

```
If parent.direction == this.direction
        Περίπτωση ίδιας κατευθυνσης
        Create_successor(1);
}else{
        Περίπτωση αλλαγής κατεύθυνσης
        Create_successor(1+0,001);
}
```

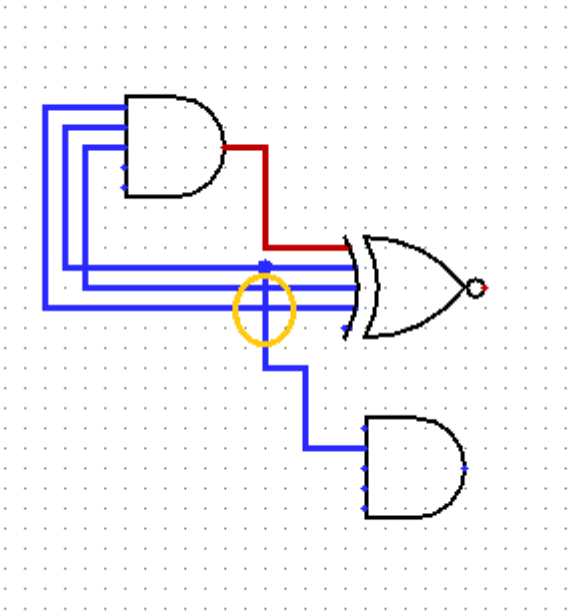
Ψευδοκώδικας 8: Δημιουργία γειτονικών κόμβων και κόστος μετακίνησης.



Εικόνα 31: Παράδειγμα κόστους A με τις λιγότερες δυνατές αλλαγές κατεύθυνσης. Επιλέγεται το μαύρο καλώδιο*

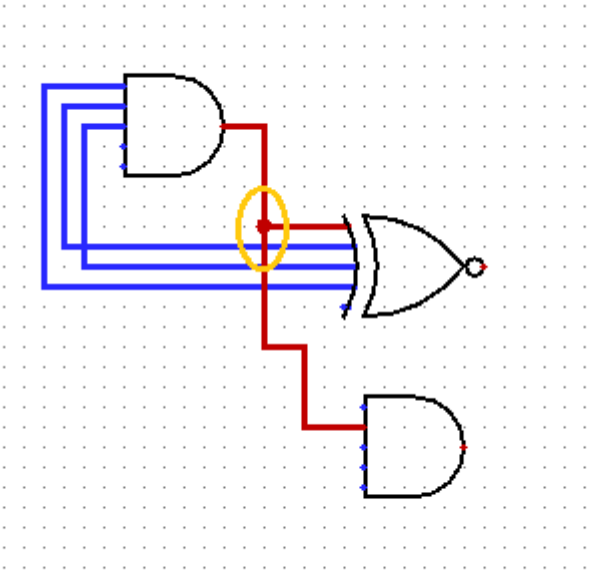
3.7.4 Εγκυρότητα γειτονικών κόμβων - Περιορισμοί Logisim

Το Logisim μας επιβάλλει κάποιους περιορισμούς σχετικά με την καλωδίωση. Γενικά ένα καλώδιο μπορεί να διαπεράσει ένα άλλο, μόνο κάθετα. Αυτό αυξάνει αρκετά την χρονική πολυπλοκότητα του προγράμματος διότι αυτοί οι έλεγχοι περιορισμών θα πρέπει να γίνονται κατά την δημιουργία των γειτονικών κόμβων στην αναζήτηση A* που είναι και η μέθοδος που χρησιμοποιείται περισσότερο από όλες.



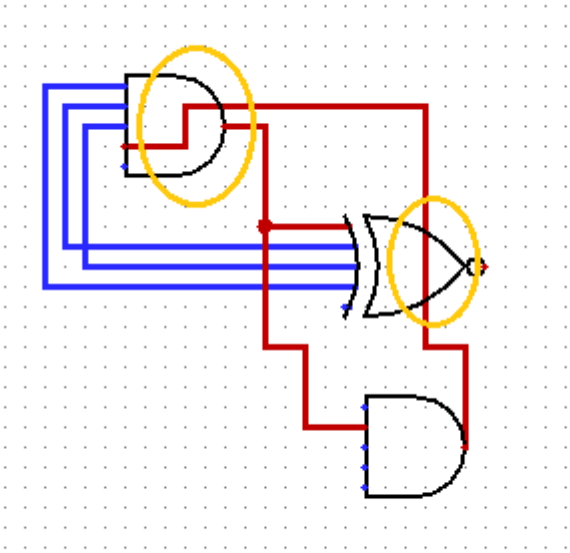
Εικόνα 32: Ένα καλώδιο επιτρέπεται να διαπερνά ένα άλλο κάθετα χωρίς να γίνεται σύνδεση

Αν προσπαθήσουμε να το διαπεράσουμε παράλληλα τότε δημιουργείται ένωση.



Εικόνα 33: Ένα καλώδιο δεν επιτρέπεται να διαπερνά ένα άλλο παράλληλα χωρίς να γίνεται σύνδεση

Το Logisim επιτρέπει τα καλώδια να περνούν μέσα από τα κυκλωματικά στοιχεία αλλά εμείς για λόγους αισθητικής αποφασίσαμε να μην το επιτρέψουμε.



Εικόνα 34: Το Logisim επιτρέπει σε ένα καλώδιο να περνάει μέσα από τα κυκλωματικά στοιχεία, αλλά εμείς το απαγορεύσαμε για λόγους αισθητικής

Άρα οι περιορισμοί για τους έγκυρους γειτονικούς κόμβους για την αναζήτηση A* προκύπτουν από τα παραπάνω.

Δηλαδή:

Α) Να μην είναι σημεία άλλου αντικειμένου εκτός μόνο από το τελικό σημείο στο αντικείμενο στόχο.

Β) Οι καλωδιώσεις να διασχίζουν άλλες καλωδιώσεις μόνο κάθετα.

3.7.5 Υπολογισμός της συνάρτησης αξιολόγησης (Fitness Function)

Αυτή η μελέτη βασίστηκε σε βασικούς κανόνες σχεδιασμού σχηματικών ψηφιακών κυκλωμάτων. Μετά την εξέταση των βασικών χαρακτηριστικών μιας αποδεκτής σχεδίασης,

ομαδοποιήσαμε όλους τους παράγοντες στα ακόλουθα χαρακτηριστικά: Εμβαδόν κυκλώματος, μήκος καλωδίων, ευθυγράμμιση καλωδίων και τυπική απόκλιση καλωδίων.

Α) Εμβαδόν κυκλώματος: Υπολογίζουμε το άθροισμα των εμβαδών του κάθε στοιχείου του κυκλώματος. Αυτό είναι ένα ιδεατό εμβαδό που θα θέλαμε να έχει το τελικό κύκλωμα. Έπειτα υπολογίζουμε το πραγματικό εμβαδό. Το πραγματικό εμβαδό υπολογίζεται προσεγγιστικά. Όπως είπαμε ο καμβάς είναι ένα ορθογώνιο παραλληλόγραμμο οπότε από κάθε πλευρά κάνουμε διάσχιση παράλληλα μέχρι να συναντήσουμε κάποιο σημείο του κυκλώματος. Αυτές θα είναι οι τιμές που θα χρησιμοποιήσουμε για να υπολογίσουμε το πραγματικό εμβαδόν. Σαν τιμή της συνάρτησης αξιολόγησης για το εμβαδόν του κυκλώματος θα πάρουμε τον λόγο πραγματικού προς ιδεατού εμβαδού, αφού βέβαια τα αφαιρέσουμε από το συνολικό εμβαδόν του καμβά.

Β) Μήκος καλωδίων: Στην αρχικοποίηση του προγράμματος έχει οριστεί το επιθυμητό εύρος καλωδίων. Σε αυτήν την τιμή μετρούμε πόσα καλώδια από τα συνολικά είναι μέσα στο εύρος.

C) Ευθυγράμμιση καλωδίου: Αν ένα καλώδιο σε όλα τα σημεία του έχει ίσα y τότε έχει ευθυγράμμιση κατά των άξονα x. Αλλιώς αν έχει όλα τα x ίσα τότε έχει ευθυγράμμιση κατά των άξονα y. Σε κάθε περίπτωση το καλώδιο είναι ευθυγραμμισμένο και σαν τιμή της συνάρτησης αξιολόγησης παίρνουμε το άθροισμα των ευθυγραμμισμένων καλωδίων προς τα συνολικά.

D) Τυπική απόκλιση καλωδίων: Υπολογίζεται η τυπική απόκλιση των καλωδίων.

$$s = \sqrt{\frac{1}{N-1}\sum_{i-1}^{N}(x_i - \bar{x})^2}$$

Εξίσωση 2: Τύπος υπολογισμού της τυπικής απόκλισης

Η συνάρτηση αξιολόγησης είναι επομένως:

$$F = (A + B + C - D)$$

Εξίσωση 3: Τύπος υπολογισμού της συνάρτησης αξιλόγησης

όπου

A - Ο Λόγος των εμβαδών κυκλώματος

B - Αριθμός γραμμών που εμπίπτουν στο εύρος [μικρότερο, μεγαλύτερο]

C - Αριθμός ευθυγραμμισμένων καλωδίων

D - Τυπική απόκλιση

Κάθε μία παράμετρος (A, B, C και D) πολλαπλασιάζεται με έναν παράγοντα για την ομαλοποίηση των τιμών.

Το άνω όριο της συνάρτησης αξιολόγησης είναι το 1000 και το κάτω το 0. Το 1000 αντικατοπτρίζει ένα τέλειο και ιδεατό κύκλωμα. Η τιμές αυτές δεν θα επιτευχθούν ποτέ αλλά χρειάζονται για να μας δείχνουν ότι πλησιάζοντας προς το 1000 παίρνουμε όλο και καλύτερα κυκλώματα, ενώ προς το 0 παίρνουμε όλο και χειρότερα.

Ο αριθμός χίλια (1000) επίσης μας δείχνει και την ακρίβεια του συγκεκριμένου γενετικού αλγόριθμου. Π.χ. Ένα κύκλωμα με τιμή 255 με ένα άλλο με τιμή 255 που μπορεί να είναι διαφορετικά κυκλώματα, για τον δικό μας αλγόριθμο αυτά θεωρούνται ίδια. Σύμφωνα με τον αλγόριθμό μας, όλες οι πιθανές τιμές αξιολόγησης ενός κυκλώματος είναι χίλιες, άρα αν θέλουμε μεγαλύτερη ακρίβεια θα πρέπει να αυξήσουμε αυτήν τιμή. Στην συγκεκριμένη εργασία το νούμερο αυτό κρίθηκε αρκετό.

3.7.6 Επιλογή (Selection)

Η επιλογή των γονέων για αναπαραγωγή γίνεται με βάση την τιμή της συνάρτησης αξιολόγησης. Όσοι έχουν μεγαλύτερη τιμή, έχουν και μεγαλύτερη πιθανότητα αναπαραγωγής. Η επιλογή των δύο γονέων γίνεται από ένα Pool με λαχνούς που δίνονται ανάλογα με την πιθανότητα αναπαραγωγής του καθενός. Η πιθανότητα επιλογής ενός ατόμου του πληθυσμού (κύκλωμα) για αναπαραγωγή είναι:

$$P = \text{Fitness function value} / \text{Sum of all fitness function values}$$

Εξίσωση 4: Πιθανότητα επιλογής ενός ατόμου για αναπαραγωγή

Σε κάθε γενιά ο αριθμός του πληθυσμού πρέπει να παραμένει σταθερός διότι αν αλλάζει μπορεί μετά από κάποιες γενιές να εξαφανιστεί ή να έχουμε υπερπληθυσμό αλλά όχι καλών λύσεων γιατί θα είναι λύσεις των αρχικών γενεών.

3.7.7 Διασταύρωση (Crossover)

Η διασταύρωση γίνεται με βάση την τμηματοποίηση. Το καθένα τμήμα αντιστοιχίζεται και σε ένα χρωμόσωμα και για κάθε ένα από αυτά, με τυχαίο τρόπο επιλέγεται ένα εκ των δύο γονέων.

```
Foreach group
    Pick randomly from parent1, parent 2
    Child.add parent.group
```

3.7.8 Μετάλλαξη (Mutation)

Κατά την διαδικασία της μετάλλαξης επιλέγουμε ένα αντικείμενο τυχαία και του δίνουμε νέα θέση και προσανατολισμό μέσα στο ίδιο τμήμα. Η μετάλλαξη πυροδοτείται σύμφωνα με την πιθανότητα που ορίσαμε στην μεταβλητή αρχικοποίησης. Μετά από δοκιμές είδαμε ότι συνήθως μία σχετικά μικρή πιθανότητα μετάλλαξης (μικρότερη του 25%) δίνει το καλύτερο αποτέλεσμα διότι προσθέτει ποικιλία στις λύσεις η οποία βοηθάει στο ξεφύγουμε από τοπικά ακρότατα για να πλησιάσουμε προς το ολικό. Από την άλλη, μία σχετικά μεγάλη πιθανότητα μετάλλαξης (μεγαλύτερη του 50%) δεν επιτρέπει στις γενιές να διατηρήσουν τα καλά χαρακτηριστικά τους κατά την διαδικασία της εξέλιξης. Αυτούς τους παράγοντες θα πρέπει να τους λάβουμε υπόψη για μπορέσουμε να βρούμε μία ικανοποιητική πιθανότητα μετάλλαξης.

Στο πρόγραμμά μας συγκεκριμένα, επιλέγεται τυχαία ένα αντικείμενο από ένα τμήμα του κυκλώματος και η καινούρια του θέση θα πρέπει να είναι στο ίδιο τμήμα. Αυτό το κάνουμε για να κρατήσουμε ανεπηρέαστη την διάρθρωση των τμημάτων που ορίσαμε.

Στο κεφάλαιο 4 θα εκτελέσουμε το πρόγραμμα και θα δούμε αναλυτικά τα αποτελέσματα και τα κυκλώματα εξόδου. Θα συγκρίνουμε αποτελέσματα και χρόνους εκτέλεσης για διάφορες τιμές των παραμέτρων.

4. Εκτέλεση και αποτελέσματα

Η διαδικασία του (Ενότητα 3.7) επαναλαμβάνεται έως ότου δημιουργηθεί ο αριθμός των γενεών που ορίσαμε κατά την αρχικοποίηση, όπου και τερματίζει το πρόγραμμα. Σαν έξοδο έχουμε δημιουργήσει ένα xml αρχείο με κατάληξη .circ το οποίο είναι συμβατό με το logisim και μπορούμε να το ανοίξουμε και να δούμε το αποτέλεσμα της διαδικασίας. Το κύκλωμα που έχει δημιουργηθεί είναι πλήρως λειτουργικό και μπορούμε να το εξακριβώσουμε και να το χρησιμοποιήσουμε μέσω Logisim.

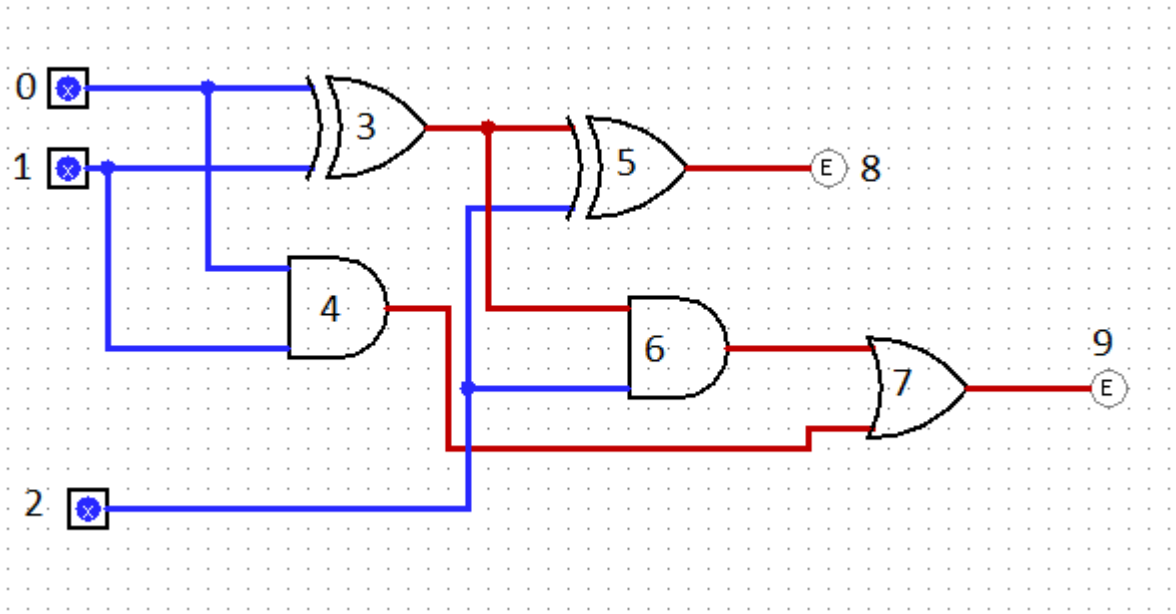
Εκτελούμε το πρόγραμμα για μεταβλητές εισόδου:

```
CANVAS_PADDING = 2
PART_PADDING = 0
OBJECT_MARGIN = 1
NUMBER_OF_GENERATIONS = 700
ELEMENTS_OF_POPULATION = 1000
NUMBER_OF_THREADS = 10
MIN_WIRE_RANGE = 1
MAX_WIRE_RANGE = 13
MUTATION_RATE = 0.15
CROSSOVER_RATE = 0.7
```

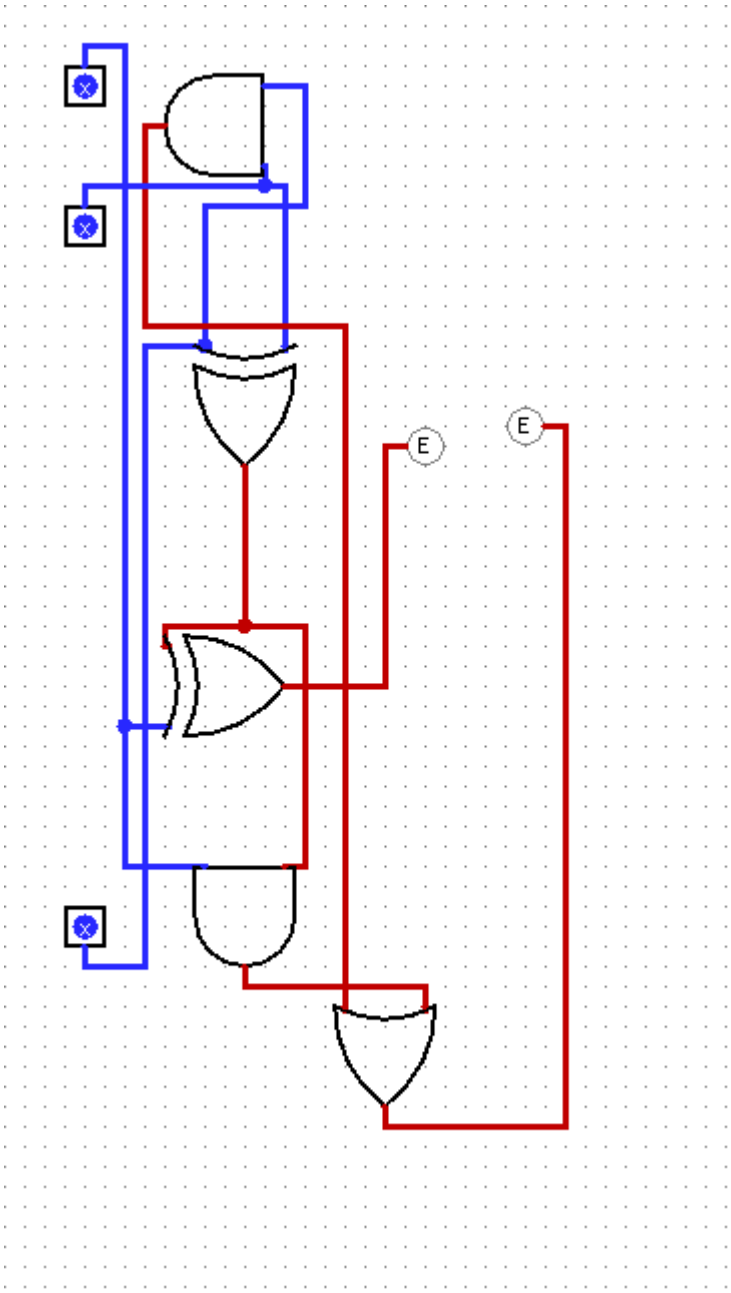
Και αρχείο εισόδου (Πλήρης αθροιστής)

```
//1. Components
Pin
Pin
Pin
XorGate
AndGate
XorGate
AndGate
OrGate
Probe
Probe


//2. Interconnections
outport{0:0}\inport{3:0}
outport{1:0}\inport{3:1}
outport{0:0}\inport{4:0}
outport{1:0}\inport{4:1}
outport{2:0}\inport{6:1}
outport{3:0}\inport{5:0}
outport{4:0}\inport{7:1}
outport{3:0}\inport{6:0}
outport{2:0}\inport{5:1}
outport{5:0}\inport{8:0}
outport{6:0}\inport{7:0}
outport{7:0}\inport{9:0}

//3. End
```

Εικόνα 35: επάνω: Αρχείο εισόδου ενός πλήρη αθροιστή, κάτω: σχηματικό Πλήρη αθροιστή με τα αναγνωριστικά των στοιχείων



Ένα από τα πρώτα κυκλώματα που δημιουργήθηκαν είναι:



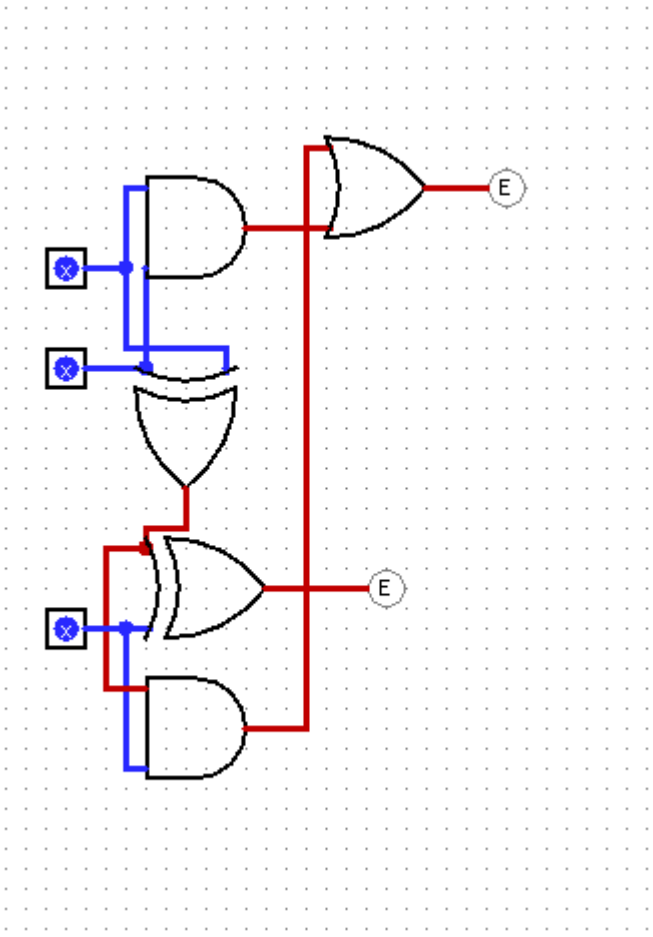
Εικόνα 36: Κύκλωμα με τιμή συνάρτησης αξιολόγησης ίση με 210

Το παραπάνω κύκλωμα θα το χρησιμοποιήσουμε ως βάση για να δούμε τον βαθμό βελτιστοποίησης που πετύχαμε κατά την ολοκλήρωση του προγράμματος. Το πρόγραμμα ολοκληρώνεται με την επιτυχή δημιουργία όλων των γενεών που έχουμε ορίσει. Επίσης θα τρέξουμε τον αλγόριθμο για διάφορες πιθανότητες μετάλλαξης και διασταύρωσης για να δούμε ποιος συνδυασμός από αυτές, μας δίνει τα καλύτερα αποτελέσματα.

4.1 Αποτελέσματα για διάφορες πιθανότητες μετάλλαξης και διασταύρωσης

Κατά την ολοκλήρωση το κύκλωμα με την μέγιστη τιμή συνάρτησης αξιολόγησης είναι:

Για πιθανότητα μετάλλαξης = 0.15 και πιθανότητα διασταύρωσης = 0.3



Εικόνα 37: Κύκλωμα με τιμή συνάρτησης αξιολόγησης ίση με 685

Γενιά = 699

Στοιχεία = 1000

Μέσος όρος τιμής της συνάρτησης αξιολόγησης για την 699 γενιά = 571.809

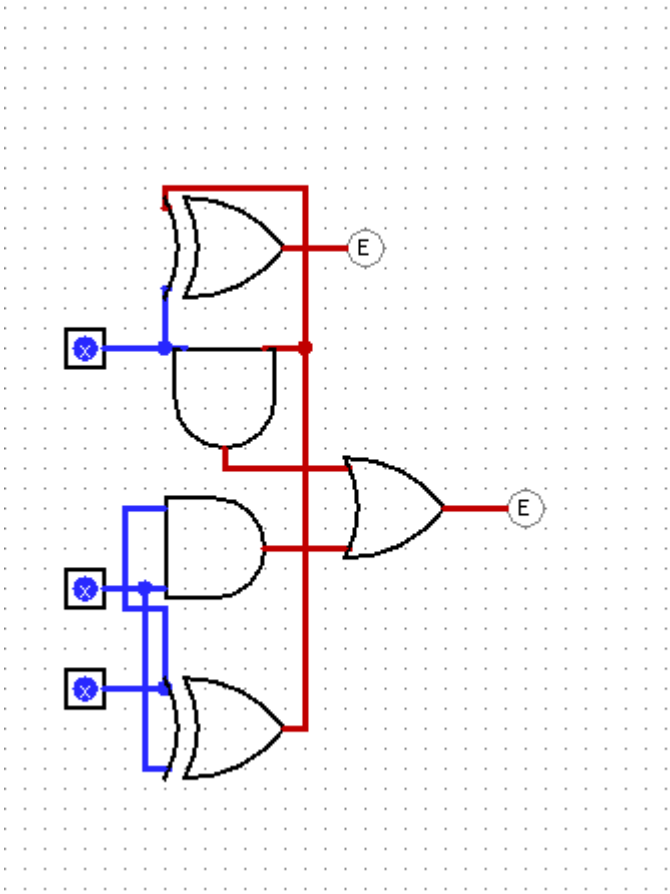
Τιμή της συνάρτησης αξιολόγησης = 685

Χρόνος εκτέλεσης = 9.16 λεπτά

Παρατηρούμε ότι για τιμή της συνάρτησης αξιολόγησης = 685 έχουμε πετύχει μια πολύ καλή βελτιστοποίηση, δεδομένου ότι έχουμε ορίσει να υπάρχει περιθώριο 2 σημείων μεταξύ των στοιχείων.

Όπως βλέπουμε το κύκλωμα έχει βελτιωθεί αρκετά ως προς τον χώρο που καταλαμβάνει επάνω στον καμβά, ως προς το μήκος των καλωδίων και των θέσεων των συνδεδεμένων στοιχείων. Το παραπάνω κύκλωμα δεν είναι τέλειο και θα μπορούσε να βελτιστοποιηθεί περισσότερο, σύμφωνα πάντα με τους περιορισμούς που έχουμε ορίσει, αλλά σίγουρα πλησιάζει το τέλειο.

Για πιθανότητα μετάλλαξης = 0.4 και πιθανότητα διασταύρωσης = 0.3



Εικόνα 38: Κύκλωμα με τιμή συνάρτησης αξιολόγησης ίση με 750

Γενιά = 699

Στοιχεία = 1000

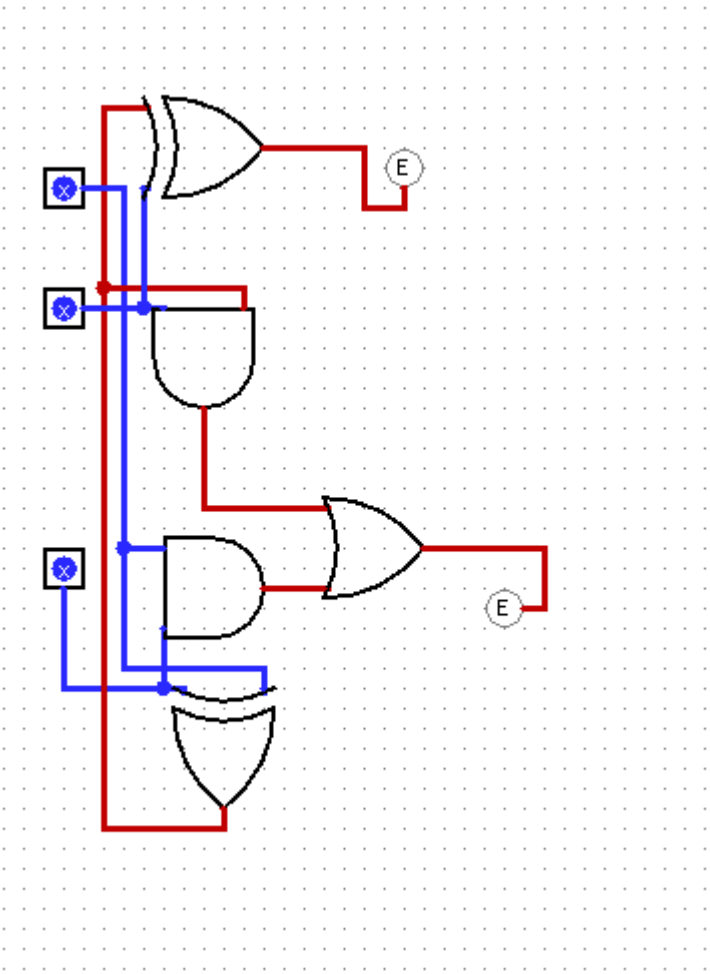
Μέσος όρος τιμής της συνάρτησης αξιολόγησης για την 699 γενιά = 501.532

Τιμή της συνάρτησης αξιολόγησης = 750

Χρόνος εκτέλεσης = 12.77 λεπτά

Εδώ η βελτιστοποίηση είναι ακόμα καλύτερη από το προηγούμενο, αλλά αυξήθηκε και ο χρόνος εκτέλεσης λόγο της αύξησης της πιθανότητα μετάλλαξης.

Για πιθανότητα μετάλλαξης = 0.8 και πιθανότητα διασταύρωσης = 0.3



Εικόνα 39: Κύκλωμα με τιμή συνάρτησης αξιολόγησης ίση με 632

Γενιά = 699

Στοιχεία = 1000

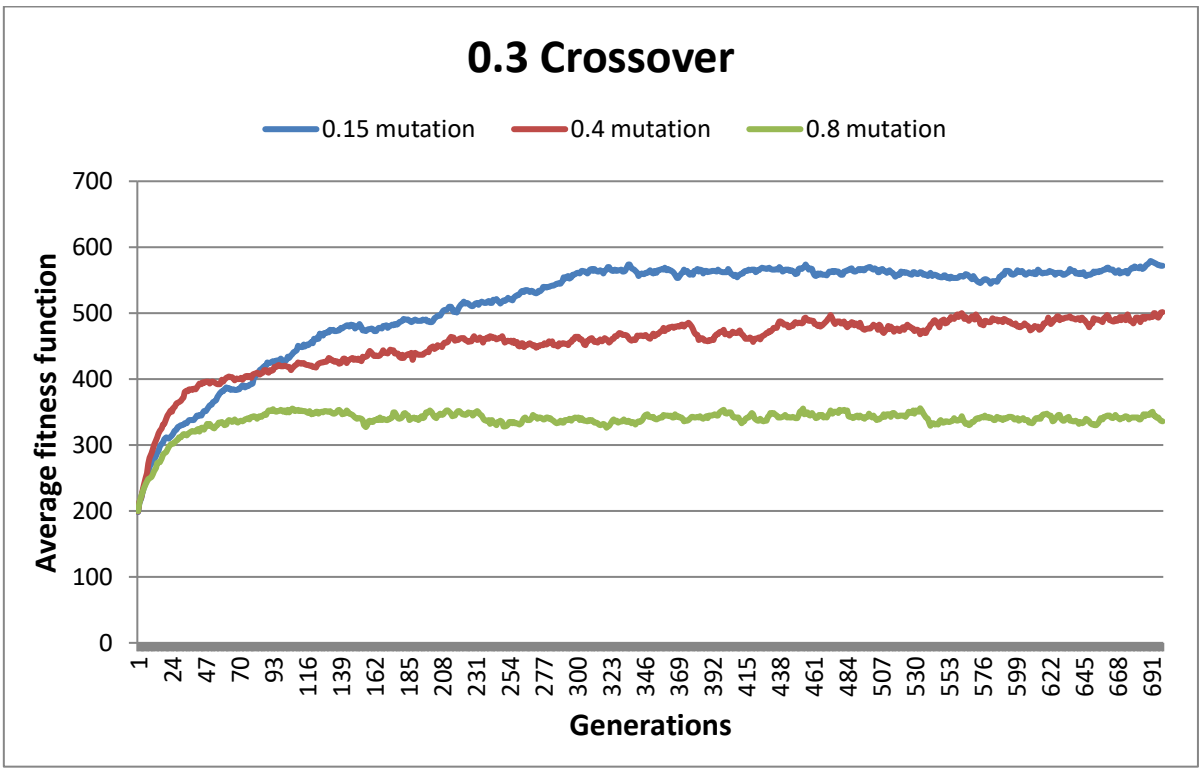
Μέσος όρος τιμής της συνάρτησης αξιολόγησης για την 699 γενιά = 336.304

Τιμή της συνάρτησης αξιολόγησης = 632

Χρόνος εκτέλεσης = 24.28 λεπτά

Εδώ παρατηρούμε ότι για ακόμα μεγαλύτερη πιθανότητα μετάλλαξης όχι μόνο ο χρόνος εκτέλεσης αυξάνεται ακόμα περισσότερο, αλλά ότι και ότι το επίπεδο της βελτιστοποίησης χειροτερεύει.

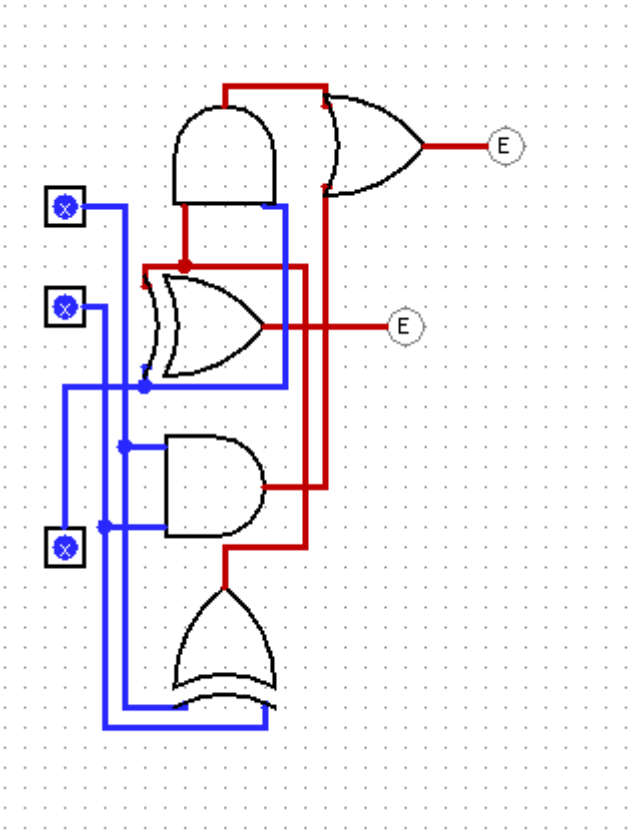
Σύγκριση αποτελεσμάτων για πιθανότητα διασταύρωσης 0.3



Εικόνα 40: Γράφημα εκτέλεσης για διαφορετικές τιμές μετάλλαξης και για πιθανότητα διασταύρωσης ίση με 0.3

Όπως βλέπουμε και στο γράφημα, για σχετικά μικρές πιθανότητες μετάλλαξης παίρνουμε τα καλύτερα αποτελέσματα.

Για πιθανότητα μετάλλαξης = 0.15 και πιθανότητα διασταύρωσης = 0.5



Εικόνα 41: Κύκλωμα με τιμή συνάρτησης αξιολόγησης ίση με 574

Γενιά = 699

Στοιχεία = 1000

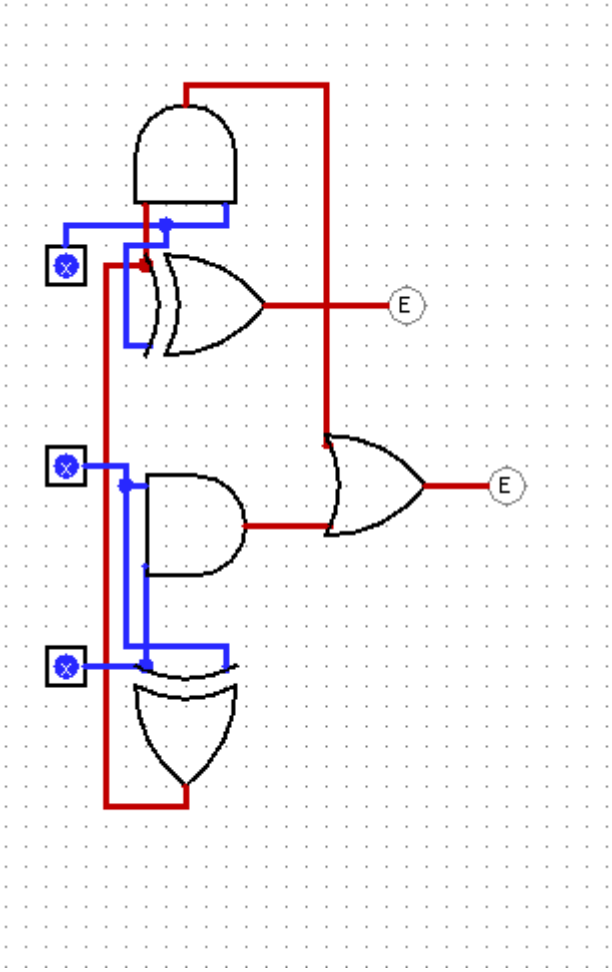
Μέσος όρος τιμής της συνάρτησης αξιολόγησης για την 699 γενιά = 487.365

Τιμή της συνάρτησης αξιολόγησης = 574

Χρόνος εκτέλεσης = 11.39 λεπτά

Γι αυτές τις τιμές δεν παρατηρούμε καλά αποτελέσματα. Έχουμε αρκετά μπερδεμένες και πολύπλοκες καλωδιώσεις.

Για πιθανότητα μετάλλαξης = 0.4 και πιθανότητα διασταύρωσης = 0.5



Εικόνα 42: Κύκλωμα με τιμή συνάρτησης αξιολόγησης ίση με 640

Γενιά = 699

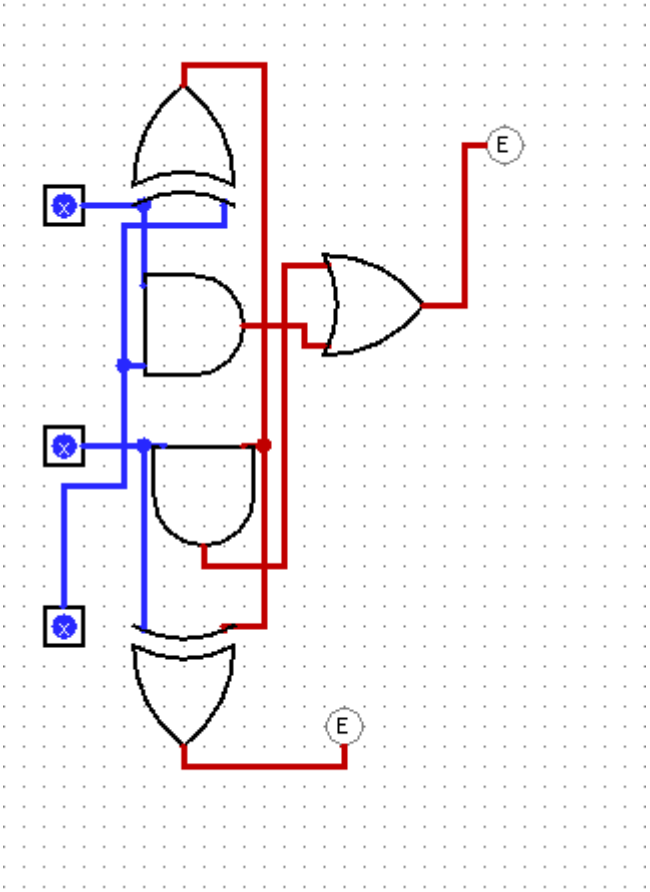
Στοιχεία = 1000

Μέσος όρος τιμής της συνάρτησης αξιολόγησης για την 699 γενιά = 436.633

Τιμή της συνάρτησης αξιολόγησης = 640

Χρόνος εκτέλεσης = 17.7 λεπτά

Για πιθανότητα μετάλλαξης = 0.8 και πιθανότητα διασταύρωσης = 0.5



Εικόνα 43: Κύκλωμα με τιμή συνάρτησης αξιολόγησης ίση με 618

Γενιά = 699

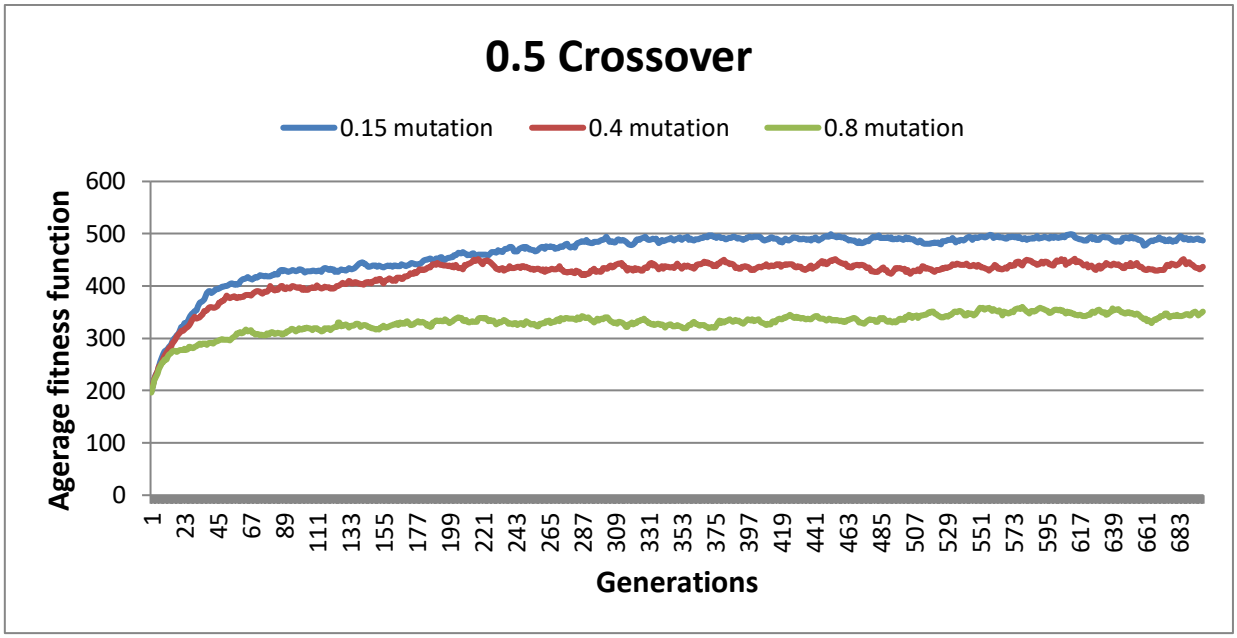
Στοιχεία = 1000

Μέσος όρος τιμής της συνάρτησης αξιολόγησης για την 699 γενιά = 350.481

Τιμή της συνάρτησης αξιολόγησης = 618

Χρόνος εκτέλεσης = 23.25 λεπτά

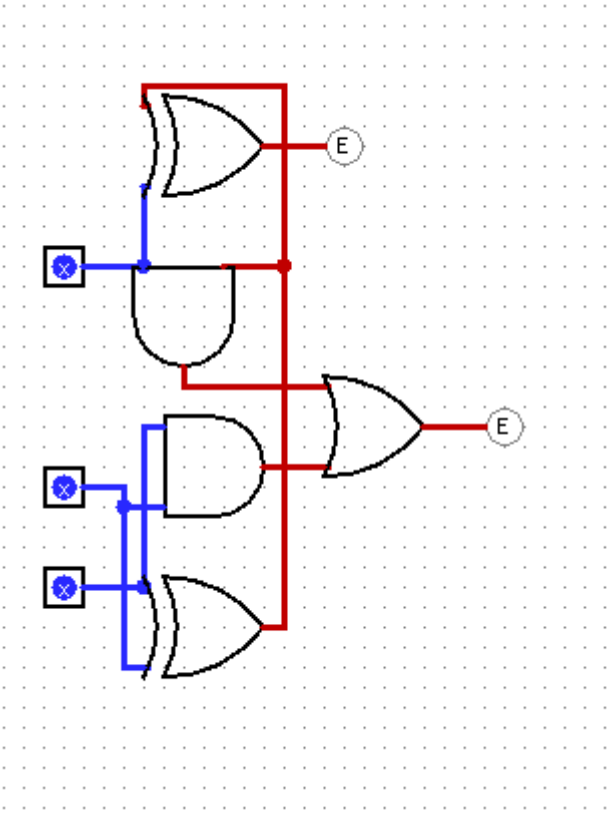
Σύγκριση αποτελεσμάτων για πιθανότητα διασταύρωσης 0.5



Εικόνα 44: Γράφημα εκτέλεσης για διαφορετικές πιθανότητες μετάλλαξης και για πιθανότητα διασταύρωσης ίση με 0.5

Παρατηρούμε κατά αρχάς ότι και πάλι για μικρές πιθανότητες μετάλλαξης έχουμε τα καλύτερα αποτελέσματα αλλά γενικά ο μέσος όρος έχει μειωθεί σε σχέση με προηγουμένως.

Για πιθανότητα μετάλλαξης = 0.15 και πιθανότητα διασταύρωσης = 0.7



Εικόνα 45: Κύκλωμα με τιμή συνάρτησης αξιολόγησης ίση με 750

Γενιά = 699

Στοιχεία = 1000

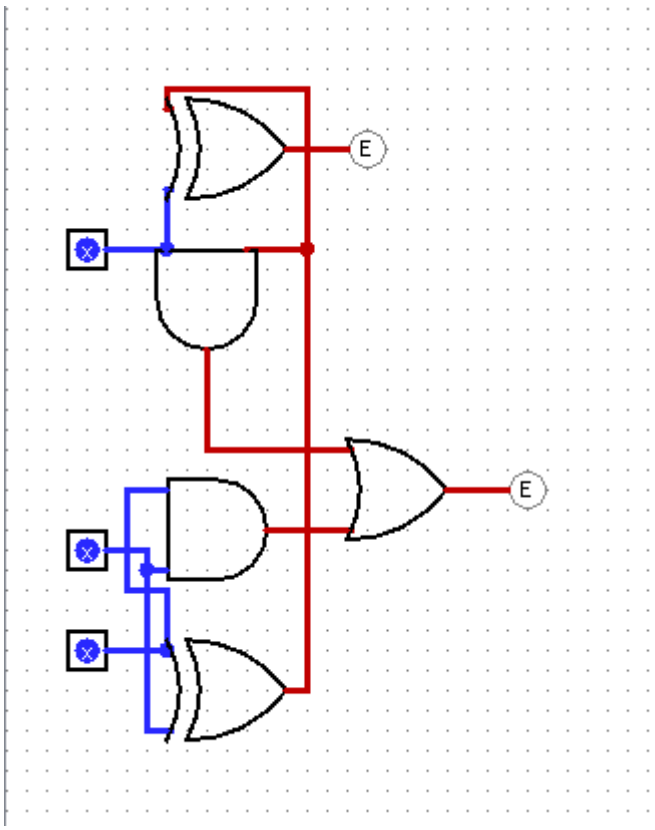
Μέσος όρος τιμής της συνάρτησης αξιολόγησης για την 699 γενιά = 627.817

Τιμή της συνάρτησης αξιολόγησης = 750

Χρόνος εκτέλεσης = 6.47 λεπτά

Εδώ η βελτιστοποίηση είναι πολύ καλή. Το κύκλωμα καταλαμβάνει τον ελάχιστο χώρο στον καμβά, οι καλωδιώσεις έχουν μικρό μήκος και πολλές από αυτές είναι και ευθυγραμμισμένες.

Για πιθανότητα μετάλλαξης = 0.4 και πιθανότητα διασταύρωσης = 0.7



Εικόνα 46: Κύκλωμα με τιμή συνάρτησης αξιολόγησης ίση με 736

Γενιά = 699

Στοιχεία = 1000

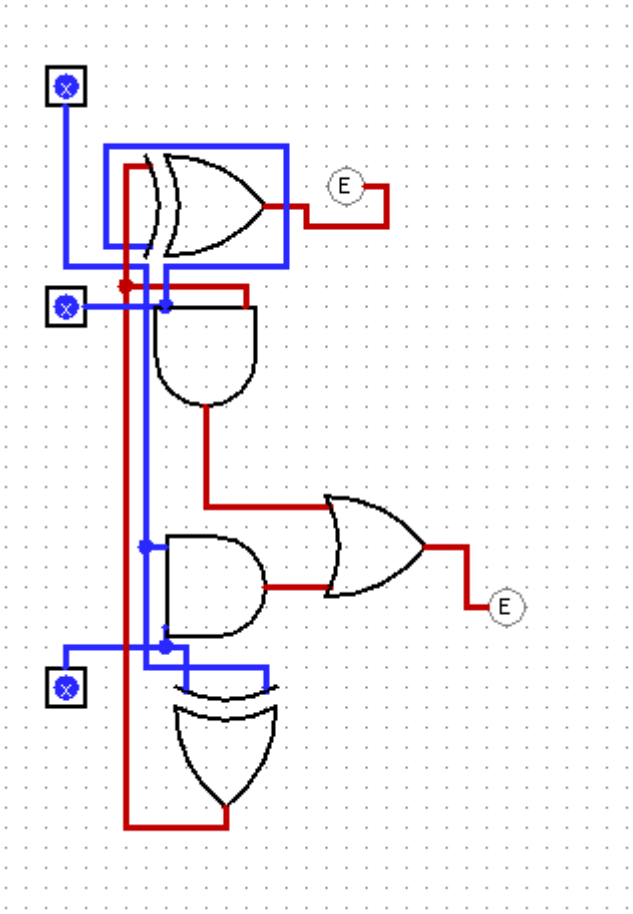
Μέσος όρος τιμής της συνάρτησης αξιολόγησης για την 699 γενιά = 484.636

Τιμή της συνάρτησης αξιολόγησης = 736

Χρόνος εκτέλεσης = 12.56 λεπτά

Όπως και εδώ η βελτιστοποίηση είναι πολύ καλή. Περίπου στα επίπεδα του προηγούμενου κυκλώματος.

Για πιθανότητα μετάλλαξης = 0.8 και πιθανότητα διασταύρωσης = 0.7



Εικόνα 47: Κύκλωμα με τιμή συνάρτησης αξιολόγησης ίση με 622

Γενιά = 699

Στοιχεία = 1000

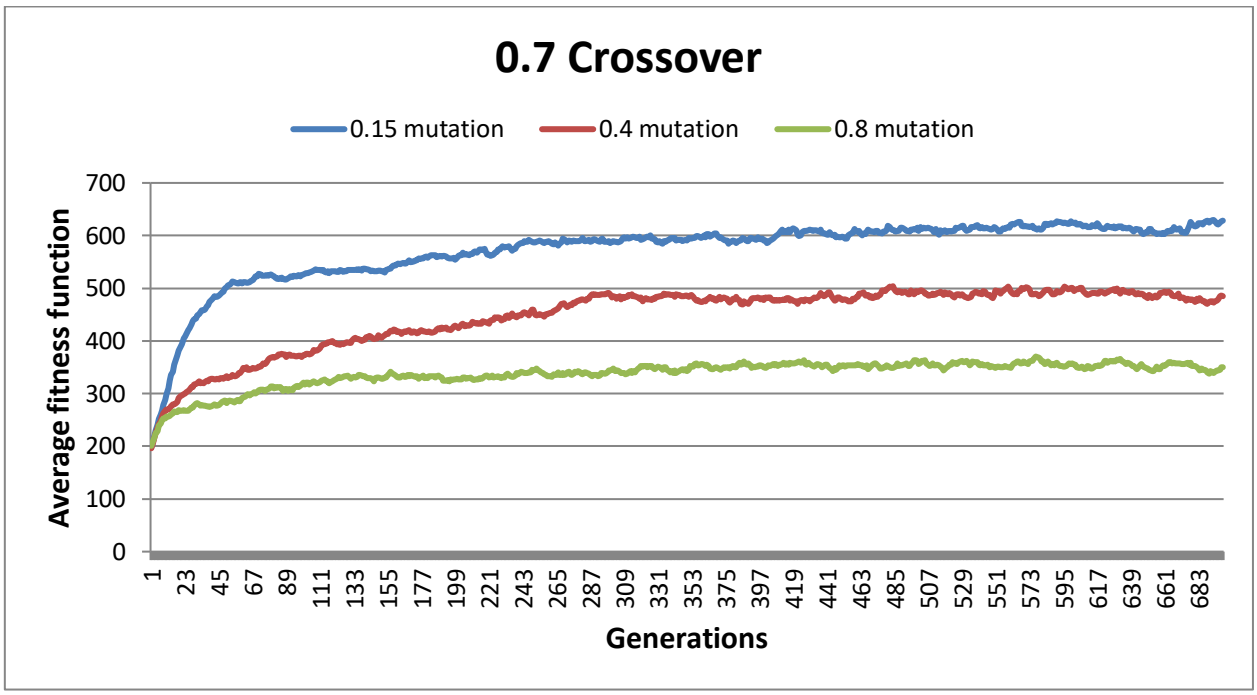
Μέσος όρος τιμής της συνάρτησης αξιολόγησης για την 699 γενιά = 349.618

Τιμή της συνάρτησης αξιολόγησης = 622

Χρόνος εκτέλεσης = 25.66 λεπτά

Για μεγάλη πιθανότητα μετάλλαξης βλέπουμε πάλι ότι το κύκλωμα είναι μπερδεμένο και με μεγαλύτερο χρόνο εκτέλεσης.

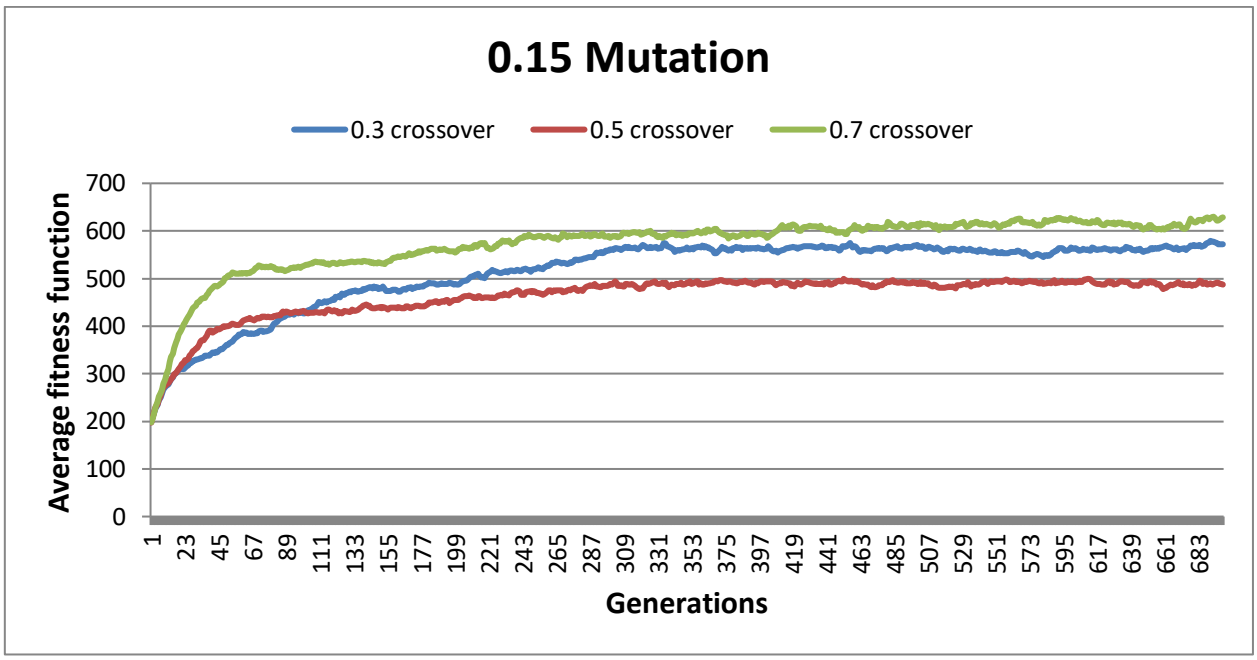
Σύγκριση αποτελεσμάτων για πιθανότητα διασταύρωσης 0.7



Εικόνα 48: Γράφημα εκτέλεσης για διαφορετικές πιθανότητες μετάλλαξης και για πιθανότητα διασταύρωσης ίση με 0.7

Για όλες τις πιθανότητες διασταύρωσης επιβεβαιώνεται ότι για μικρές πιθανότητες μετάλλαξης παίρνουμε τα καλύτερα αποτελέσματα.

Συνολική σύγκριση για πιθανότητα μετάλλαξης 0.15



Εικόνα 49: Γράφημα εκτέλεσης για διαφορετικές πιθανότητες διασταύρωσης και για πιθανότητα μετάλλαξης ίση με 0.15

Συγκρίνοντας τα γραφήματα για πιθανότητα μετάλλαξης 0.15 βλέπουμε ότι για πιθανότητα διασταύρωσης 0.7 έχουμε τα καλύτερα αποτελέσματα.

4.2 Κατανάλωση μνήμης

Για κύκλωμα των 10 στοιχείων

Για 100 κυκλώματα σε μία γενιά ο δεσμευμένος χώρος μνήμης είναι στα 0,5 GB.

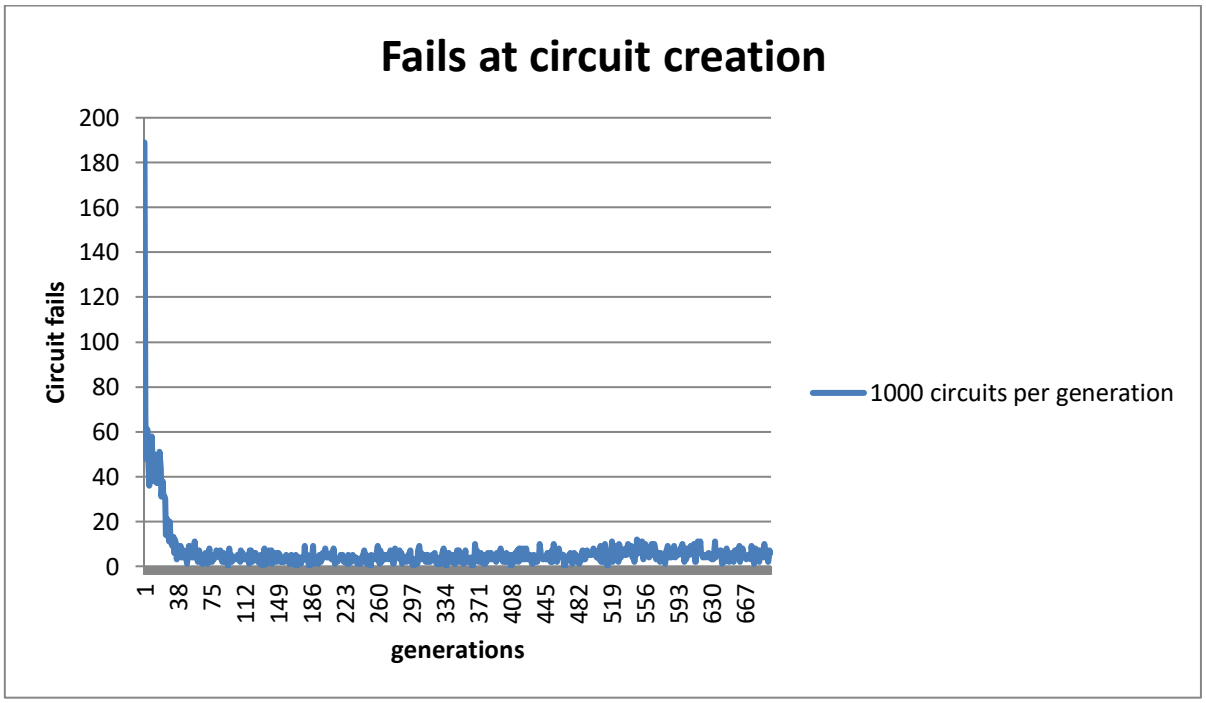
Για 1000 κυκλώματα σε μία γενιά ο δεσμευμένος χώρος μνήμης είναι στα 1,5 GB.

Για 10000 κυκλώματα σε μία γενιά ο δεσμευμένος χώρος μνήμης είναι στα 2,5GB.

4.3 Αποτυχίες δημιουργίας κυκλώματος

Αποτυχίες δημιουργίας κυκλώματος έχουμε όταν δεν είναι εφικτή η διασύνδεση μεταξύ των στοιχείων. Δύο είναι οι λόγοι που μπορούν να μας οδηγήσουν σε αποτυχία. Πρώτον, βάζοντας τα στοιχεία με τυχαίο τρόπο στον χώρο μπορεί να τύχει αυτά να

περικυκλώσουν ένα pin και η διασύνδεση του να μην είναι πλέον εφικτή και δεύτερον, καθώς δημιουργούνται οι διασυνδέσεις, μπορεί και αυτές να αποκλείσουν κάποιες διαδρομές και να απομονώσουν ένα pin. Στο παρακάτω γράφημα βλέπουμε τις αποτυχίες κυκλώματος ανά γενιά. Ο αλγόριθμος εκτελέστηκε για 1000 κυκλώματα ανά γενιά και για 700 γενιές. Είναι εμφανές ότι στην αρχή της εκτέλεσης του προγράμματος οι αποτυχίες βρίσκονται στο μεγαλύτερο ποσοστό, στο 16% περίπου. Αυτό οφείλεται στον πρώτο λόγο που περιγράψαμε νωρίτερα και στην τυχαία τοποθέτηση των στοιχείων στον χώρο. Στην συνέχεια το ποσοστό αυτό μειώνεται και παραμένει σταθερό κατά την υπόλοιπη διάρκεια εκτέλεσης διότι το κύκλωμα αρχίζει και βελτιστοποιείται.



Στο κεφάλαιο που ακολουθεί θα συγχωνεύσουμε όλα αυτά τα αποτελέσματα και τα συμπεράσματα που καταλήξαμε. Επίσης θα προτείνουμε τεχνικές βελτίωσης της απόδοσης και της λειτουργίας του γενετικού αλγόριθμου.

5. Μετρικές κώδικα και προδιαγραφές συστήματος

5.1 Κυκλωματική πολυπλοκότητα (Cyclomatic complexity)

Η κυκλωματική πολυπλοκότητα ενός τμήματος του πηγαίου κώδικα είναι ο αριθμός των γραμμικά ανεξάρτητων διαδρομών μέσα σε αυτό - όπου "γραμμικά ανεξάρτητο" σημαίνει ότι κάθε διαδρομή έχει τουλάχιστον ένα άκρο που δεν βρίσκεται σε μια από τις άλλες διαδρομές. Για παράδειγμα, εάν ο πηγαίος κώδικας δεν περιείχε δηλώσεις ροής ελέγχου (υπό όρους ή σημεία απόφασης), η πολυπλοκότητα θα ήταν 1, αφού θα υπήρχε μόνο μία διαδρομή μέσω του κώδικα. Εάν ο κώδικας είχε μια δήλωση IF μίας συνθήκης, θα υπήρχαν δύο διαδρομές μέσω του κώδικα: μία όπου η δήλωση IF αξιολογείται σε TRUE και μια άλλη όπου αξιολογεί FALSE, οπότε η πολυπλοκότητα θα είναι 2. Δύο ένθετα IF μίας συνθήκης , ή ένα IF με δύο συνθήκες, θα προκαλούσε πολυπλοκότητα 3.

5.2 Μετρικές πηγαίου κώδικα

Χρησιμοποιώντας το εργαλείο Simple Code Metrics πήραμε τα παρακάτω αποτελέσματα:

Total lines of code: 10552

Top 5 Classes Lines of Code:

Dc_schem	792
Circuit_solution	778
XMLReaderWriter	544
CustomComponent	500
Object	307

Packages Lines of Code:

dc_schematic	4183
comp	6369

Total lines with imports: 497

Top 5 classes imports:

XMLReaderWriter	25
Dc_schem	13
DotReader	8
XorGate	8
XnorGate	8

Packages imports:

dc_schematic	497
comp	433

Total blank lines: 1663

Top 5 classes blank lines:

Circuit_solution	169
Dc_schem	130
XMLReaderWriter	107
CustomComponent	86
Node	57

Total classes: 75

Packages with the biggest number of classes:

dc_schematic	75
comp	56

Total methods: 266

Top 5 classes with the biggest number of methods:

Circuit_solution	37
Object	31
Node	15
Group	13
XMLReaderWriter	12

Average cyclomatic complexity: 4.57

Top 5 methods with the highest cyclomatic complexity:

Dc_schem::main	57
CustomComponent::SetAllPins	56
Node::Get_successors	23
Dc_schem::Optimize_with_GA	18
CustomComponent::SetInputOutputPins	16

Όπως είδαμε στο (Ενότητα 5.1) όσο πιο μικρή η τιμή της κυκλωματικής πολυπλοκότητας τόσο πιο εύκολα συντηρείται ένα πρόγραμμα και έχει λιγότερες πιθανότητες σφαλμάτων. Η κυκλωματική πολυπλοκότητα στο δικό μας εργαλείο βρίσκεται στο 4.57 που είναι μια σχετικά καλή τιμή αν αναλογιστεί κανείς την πολυπλοκότητα του προβλήματος.

5.3 Χαρακτηριστικά του συστήματος

Τα χαρακτηριστικά του συστήματος που έγιναν οι παραπάνω μετρήσεις είναι:

CPU	AMD Ryzen 7 1700, 8 cores, 3Ghz, 16 Logical Processors
RAM	16 GB
OS	Microsoft Windows 10 PRO

6. Συμπεράσματα και βελτιώσεις για το μέλλον

6.1 Σύνοψη

Αφού εκτελέσαμε το πρόγραμμα για διάφορες πιθανότητες μετάλλαξης και αναπαραγωγής διαπιστώσαμε ότι λαμβάνουμε τα καλύτερα αποτελέσματα για σχετικά μικρές πιθανότητες μετάλλαξης και σχετικά μεγάλες πιθανότητες αναπαραγωγής. Συγκεκριμένα, μας ενδιαφέρει να βρούμε με ποιον συνδυασμό μεγιστοποιείται η μέση τιμή της συνάρτησης αξιολόγησης. Όπως φαίνεται και από τα γραφήματα για 0.15 πιθανότητα μετάλλαξης και 0.7 πιθανότητα διασταύρωσης η μέση τιμή της συνάρτησης αξιολόγησης έχει ξεπεράσει τις 600 μονάδες κατά μέσο όρο, με το καλύτερο κύκλωμα να είναι στις 750 μονάδες. Σ αυτό το σημείο να υπενθυμίσουμε ότι το 1000 της κλίμακας είναι ιδεατό και δεν υπάρχει περίπτωση να επιτευχθεί.

Αυτό το συμπέρασμα επιβεβαιώνεται και από άλλους επιστήμονες που μελέτησαν τους γενετικούς αλγόριθμους.

Γενικά το πρόβλημα έχει αυξημένη χρονική πολυπλοκότητα και η συχνά χρησιμοποιούμενη λειτουργία είναι αυτή της αναζήτησης A*. Αν ένα κύκλωμα έχει 10 διασυνδέσεις και τρέξουμε τον αλγόριθμο για πληθυσμό 1.000 και για 1.000 γενιές τότε η αναζήτηση A* θα τρέξει τουλάχιστον 10.000.000 φορές. Τουλάχιστον, διότι κάποια κυκλώματα όπως προείπαμε μπορεί να μην καταφέρουν να δημιουργήσουν επιτυχώς όλες τις διασυνδέσεις τους και να αχρηστευθούν. Γι αυτό τον λόγο και εμείς ενεργοποιούμε το $2^{ο}$ στάδιο του A* (αυτό με τις λιγότερες δυνατές αλλαγής κατεύθυνσης) μόνο στα καλύτερα κυκλώματα, τα οποία ενδέχεται να εξαχθούν στην έξοδο. Επιπλέον, καθεμία φορά που κάνουμε την αναζήτηση A*, χρειάζεται να γίνονται κοστοβόροι έλεγχοι περιορισμών ως προς την καλωδίωση, διότι το περιβάλλον αλλάζει δυναμικά.

Στο επόμενο κεφάλαιο θα δούμε κάποιες τεχνικές που θα μπορούσαν σε κάποια μελλοντική έκδοση να προστεθούν. Κάποιες τεχνικές που αφορούν την βελτίωση της απόδοσης και τον βαθμό της βελτιστοποίησης.

6.2 Βελτιώσεις για το μέλλον

Στο παρόν λογισμικό κατά την διαδικασία της μετάλλαξης αλλάζουμε τυχαία την θέση ενός αντικειμένου στον καμβά. Η αλλαγή θέσης γίνεται στο ίδιο τμήμα για να μην αλλάξει η σύσταση των χρωμοσωμάτων του γενετικού αλγόριθμου. Κατά την διαδικασία αυτή όταν υπάρχουν συγκρούσεις για μια θέση που είναι κατειλημμένη, τότε δίνεται νέα τυχαία θέση έως ότου το κυκλωματικό στοιχείο εισαχθεί στον καμβά. Η βελτίωση που θα μπορούσε να γίνει σε μελλοντική επανεξέταση του προβλήματος είναι να κρατάμε κάποιον χάρτη διαθέσιμων δυνατών θέσεων έτσι ώστε, να απαλειφθούν οι συγκρούσεις.

6.2.1 Προσαρμοστικός Γενετικός Αλγόριθμος (Adaptive Genetic Algorithm)

Μια άλλη σημαντική και πολλά υποσχόμενη παραλλαγή των γενετικών αλγορίθμων είναι οι γενετικοί αλγόριθμοι με προσαρμοστικές παραμέτρους (προσαρμοστικοί γενετικοί αλγόριθμοι). Οι πιθανότητες διασταύρωσης και μετάλλαξης καθορίζουν σε μεγάλο βαθμό τον βαθμό ακρίβειας της λύσης και την ταχύτητα σύγκλισης που μπορούν να αποκτήσουν οι γενετικοί αλγόριθμοι . Αντί να χρησιμοποιούν σταθερές τιμές, οι προσαρμοστικοί γενετικοί αλγόριθμοι χρησιμοποιούν τις πληροφορίες του πληθυσμού σε κάθε γενιά και προσαρμόζουν ανάλογα τις τιμές αυτές για τη διατήρηση της ποικιλομορφίας του πληθυσμού καθώς και για τη διατήρηση της ικανότητας σύγκλισης. Στον προσαρμοστικό γενετικό αλγόριθμο (AGA) (Srinivas & Patnaik, 1994), η προσαρμογή της πιθανότητας μετάλλαξης και της πιθανότητας διασταύρωσης εξαρτάται από τις τιμές καταλληλότητας των λύσεων. Στον προσαρμοσμένο γενετικό αλγόριθμο βάσει συμπλέγματος (CAGA) (Zhang, Chung, & Lo, 2007), μέσω της χρήσης ανάλυσης ομαδοποιήσεων για τις καταστάσεις βελτιστοποίησης του πληθυσμού, η προσαρμογή της πιθανότητας μετάλλαξης και της πιθανότητας διασταύρωσης εξαρτάται από αυτές τις καταστάσεις βελτιστοποίησης. Μπορεί να είναι αρκετά αποτελεσματικό να συνδυάσουμε τον γενετικό αλγόριθμο με άλλες μεθόδους βελτιστοποίησης. Ο γενετικός αλγόριθμος τείνει να είναι αρκετά καλός στην εύρεση γενικά καλών ολικών λύσεων, αλλά αρκετά αναποτελεσματικός στην εύρεση των τελευταίων μεταλλάξεων για να βρεθεί το απόλυτο βέλτιστο.

Παράρτημα Α – Οδηγίες εγκατάστασης και παραδείγματα χρήσης

Για να εκτελέσουμε το πρόγραμμα απαιτείται να έχουμε εγκατεστημένο το Java Runntime Enviroment 8, οποίο διατίθεται δωρεάν από https://www.java.com/en/download/win10.jsp.

Αφού το εγκαταστήσουμε, τότε μπορούμε να μεταβούμε στον φάκελο που υπάρχει το εκτελέσιμο dc_schematic.jar και με την εντολή

java -jar dc_schematic.jar -help

να δούμε τις διαθέσιμες επιλογές και να εκτελέσουμε το πρόγραμμα.

Παράρτημα Β - Αναλυτική λίστα αποδεκτών κυκλωματικών στοιχείων με τα χαρακτηριστικά τους

Τα χαρακτηριστικά των στοιχείων και οι προεπιλεγμένες τιμές τους μπορούν να βρεθούν και από το Logisim.

Όλα τα χαρακτηριστικά είναι προαιρετικά.

1. **Wiring**
    1. Splitter[No Attributes]
    2. Pin[width=1-32, output=true/false, tristate= true/false, pull= uncharged/up/down, label=1-100 chars]
    3. Probe[radix= bin/8/10signed/10unsigned/16, label=1-100 chars]
    4. Tunnel[width=1-32, label=1-100 chars]
    5. PullResistor[pull=0/1/X]
    6. Clock[highDuration=1-2.147.483.647, lowDuration=1-2.147.483.647, label=1-100 chars]
    7. Constant[width=1-32, value=Hex depending on width-0x..]
    8. Power[width=1-32]
    9. Ground[width=1-32]
    10. Transistor[width=1-32, type=p/n]

11. TransmissionGate[width=1-32]
12. BitExtender[in_width=1-32, out_width=1-32, type= zero/one/sign/input]

2. Gates

1. NotGate[width=1-32, out=01/0Z/ Z1, label=1-100 chars]
2. Buffer[width=1-32, out=01/ 0Z/ Z1, label=1-100 chars]
3. AndGate[width=1-32, inputs=2-32, out=01/0Z/ Z1, label=1-100 chars]
4. OrGate[width=1-32, inputs=2-32, out=01/0Z/ Z1, label=1-100 chars]
5. NandGate[width=1-32, inputs=2-32, out=01/0Z/ Z1, label=1-100 chars]
6. NorGate[width=1-32, inputs=2-32, out=01/0Z/ Z1, label=1-100 chars]
7. XorGate[width=1-32, inputs=2-32, out=01/0Z/ Z1, xor=one/odd, label=1-100 chars]
8. XnorGate[width=1-32, inputs=2-32, out=01/0Z/ Z1, xor=one/odd, label=1-100 chars]
9. OddParity[width=1-32, inputs=2-32, out=01/0Z/ Z1, label=1-100 chars]
10. EvenParity[width=1-32, inputs=2-32, out=01/0Z/ Z1, label=1-100 chars]
11. ControlledBuffer[width=1-32, label=1-100 chars]
12. ControlledInverter[width=1-32, label=1-100 chars]

3. Plexers

1. Multiplexer[width=1-32, select=1-5, disabled=floating/0, enable=true/false]
2. Demultiplexer[width=1-32, select=1-5, tristate=true/false, disabled=floating/0, enable= true/false]
3. Decoder[select=1-5, tristate=true/false, disabled=floating/0, enable=true/false]
4. PriorityEncoder[select=1-5, disabled=floating/0]
5. BitSelector[width=1-32, group=1-32]

4. Arithmetic

1. Adder[width=1-32]
2. Subtractor[width=1-32]
3. Multiplier[width=1-32]
4. Divider[width=1-32]
5. Negator[width=1-32]
6. Comparator[width=1-32, mode=2ndComplement/unsigned]
7. Shifter[width=1-32, shift=ll/lr/ar/rl/rr]
8. BitAdder[width=1-32, inputs=2-32]
9. BitFinder[width=1-32, type= low1/high1/low0/high0]

5. **Memory**
    1. DFlipFlop[trigger= rising/falling/high/low, label=1-100 chars]
    2. TFlipFlop[trigger=rising/falling, label=1-100 chars]
    3. JKFlipFlop[trigger=rising/falling, label=1-100 chars]
    4. SRFlipFlop[trigger=rising/falling/high/low, label=1-100 chars]
    5. Register[width=1-32, trigger=rising/falling/high/low, label=1-100 chars]
    6. Counter[width=1-32, max=Hex depending on width-0x.., ongoal=wrap/stay/continue/load, trigger=rising/falling, label=1-100 chars]
    7. ShiftRegister[width=1-32, parallel=true/false, length=1-32, trigger=rising/falling, label=1-100 chars]
    8. RandomGenerator[width=1-32, seed=INT.MIN-INT.MAX, trigger=rising/falling, label=1-100 chars]
    9. Ram[addrWidth=2-24, dataWidth=1-32, bus=synch/asynch/separate]
    10. Rom[addrWidth=2-24, dataWidth=1-32]
6. **Input/Output**
    1. Button[label=1-100 chars]
    2. Joystick[bits=2/3/4/5]
    3. Keyboard[buflen=1-256, trigger=rising/falling]
    4. Led[active=true/false, label=1-100 chars]
    5. SevenSegmentDisplay[active=true/false]
    6. HexDigitDisplay[No Attributes]
    7. LedMatrix[matrixcols=1-32, matrixrows=1-32, inputtype=col/row/select]
    8. Tty[No Attributes]

Βιβλιογραφία

Brian Cantwell Smith, P. d. (1982). *Procedural Reflection in Programming Languages.* Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology.

Carter, P. P. (1995). *Genetic algorithm crossover operators for ordering applications.* Computers & Operations Research.

Chan, R. (2019). *The 10 most popular programming languages, according to the Facebook for programmers.* Business Insider.

Delling, D., Sanders, P., Schultes, D., & Wagner, D. (2009). *Engineering Route Planning Algorithms.*

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*.

(July 12, 2019). *The Apache Software Foundation Announces Apache® NetBeans™ as a Top-Level Project.* blogs.apache.org.

Langley, N. (2002). Write once, run anywhere? *Computer Weekly*.

Laplante, P. A. (25 April 2007). What Every Engineer Should Know about Software Engineering. *CRC Press*.

Malenfant, J. (21 August 2017). A Tutorial on Behavioral Reflection and its Implementation.

McCabe. (December 1976). "A Complexity Measure". *IEEE Transactions on Software Engineering*, 308–320.

McMahon, B. F. (1991). *Genetic operators for sequencing problems.* San Mateo, CA: Foundations of Genetic Algorithms. Morgan Kaufmann.

McMillan, R. (2013). *Is Java Losing Its Mojo?* wired.com.

Miri Weiss Cohen, M. A. (2015). *Genetic Algorithm Software System for Analog Circuit Design.* ScienceDirect.

Oracle. (2013). *Java Takes on the Internet of Things.* www.oracle.com.

Russell, S. J. (2018). *Artificial intelligence a modern approach.* Boston: Pearson.

Sakaiya, T. (1999). *Design Goals of the Java™ Programming Language*. Retrieved from oracle.com.

Smith, B. C. (January 1982). *Reflection and semantics in a procedural language.* Massachusetts Institute of Technology, Cambridge, Massachusetts.

Sobey., A. J. (2006). *Basis Path Testing.* NIST Special Publication.

Srinivas, M., & Patnaik, L. (1994). Adaptive probabilities of crossover and mutation in genetic algorithms. *IEEE Transactions on System, Man and Cybernetics*.

Surry, N. R. (1995). *Format and the variance of fittness.* San Mateo, CA.

Whitley, D. (1994). In *A genetic algorithm tutoria* (pp. 65–85).

Zeng, W., & Church, R. L. (2009). Finding shortest paths on real road networks: the case for A*. *International Journal of Geographical Information Science*.

Zhang, J., Chung, H., & Lo, W. L. (2007). Clustering-Based Adaptive Crossover and Mutation Probabilities for Genetic Algorithms. *IEEE Transactions on Evolutionary Computation*.