ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΒΕΛΤΙΩΣΗ ΤΗΣ ΠΟΙΟΤΗΤΑΣ ΤΩΝ ΑΠΟΤΕΛΕΣΜΑΤΩΝ ΤΟΥ ΑΛΓΟΡΙΘΜΟΥ ΝΟΜΙΜΟΠΟΙΗΣΗΣ TETRIS ΜΕΣΩ ΕΥΡΕΤΙΚΩΝ ΠΡΟΣΕΓΓΙΣΕΩΝ

ΑΡΓΥΡΟΣ ΑΓΓΕΛΟΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

Αντώνιος Δαδαλιάρης

Επίκουρος Καθηγητής

Τμήματος Πληροφορικής & Τηλεπικοινωνιών Πανεπιστημίου Θεσσαλίας

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΒΕΛΤΙΩΣΗ ΤΗΣ ΠΟΙΟΤΗΤΑΣ ΤΩΝ ΑΠΟΤΕΛΕΣΜΑΤΩΝ ΤΟΥ ΑΛΓΟΡΙΘΜΟΥ ΝΟΜΙΜΟΠΟΙΗΣΗΣ TETRIS ΜΕΣΩ ΕΥΡΕΤΙΚΩΝ ΠΡΟΣΕΓΓΙΣΕΩΝ

ΑΡΓΥΡΟΣ ΑΓΓΕΛΟΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

Αντώνιος Δαδαλιάρης

Επίκουρος Καθηγητής

Τμήματος Πληροφορικής & Τηλεπικοινωνιών Πανεπιστημίου Θεσσαλίας

Λαμία έτος 2024

UNIVERSITY OF
THESSALY

SCHOOL OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE &
TELECOMMUNICATIONS

IMPROVING THE QUALITY OF THE RESULTS OF THE TETRIS REGULARIZATION ALGORITHM THROUGH HEURISTIC APPROACHES

ARGYROS AGGELOS

FINAL THESIS

ADVISOR
Antonios Dadaliaris

Associate Professor

DEPARTMENT OF COMPUTER SCIENCE & TELECOMMUNICATIONS
UNIVERSITY OF THESSALY
Lamia year 2024

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

1. *Δεν παραθέτω κομμάτια βιβλίων ή άρθρων ή εργασιών άλλων αυτολεξεί **χωρίς να τα περικλείω σε εισαγωγικά** και χωρίς να αναφέρω το συγγραφέα, τη χρονολογία, τη σελίδα. Η αυτολεξεί παράθεση χωρίς εισαγωγικά χωρίς αναφορά στην πηγή, είναι λογοκλοπή. Πέραν της αυτολεξεί παράθεσης, λογοκλοπή θεωρείται και η παράφραση εδαφίων από έργα άλλων, συμπεριλαμβανομένων και έργων συμφοιτητών μου, καθώς και η παράθεση στοιχείων που άλλοι συνέλεξαν ή επεξεργάσθηκαν, χωρίς αναφορά στην πηγή. Αναφέρω πάντοτε με πληρότητα την πηγή κάτω από τον πίνακα ή σχέδιο, όπως στα παραθέματα.*
2. *Δέχομαι ότι η αυτολεξεί **παράθεση χωρίς εισαγωγικά**, ακόμα κι αν συνοδεύεται από αναφορά στην πηγή σε κάποιο άλλο σημείο του κειμένου ή στο τέλος του, είναι αντιγραφή. Η αναφορά στην πηγή στο τέλος π.χ. μιας παραγράφου ή μιας σελίδας, δεν δικαιολογεί συρραφή εδαφίων έργου άλλου συγγραφέα, έστω και παραφρασμένων, και παρουσίασή τους ως δική μου εργασία.*
3. *Δέχομαι ότι υπάρχει επίσης περιορισμός στο μέγεθος και στη συχνότητα των παραθεμάτων που μπορώ να εντάξω στην εργασία μου εντός εισαγωγικών. Κάθε μεγάλο παράθεμα (π.χ. σε πίνακα ή πλαίσιο, κλπ), προϋποθέτει ειδικές ρυθμίσεις, και όταν δημοσιεύεται προϋποθέτει την άδεια του συγγραφέα ή του εκδότη. Το ίδιο και οι πίνακες και τα σχέδια*
4. *Δέχομαι όλες τις συνέπειες σε περίπτωση λογοκλοπής ή αντιγραφής.*

Ημερομηνία: 27/10/2024

Ο – Η Δηλ.

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

ΠΕΡΙΛΗΨΗ

Ένα από τα σημαντικότερα στάδια στην διαδικασία του φυσικού σχεδιασμού κυκλωμάτων VLSI, είναι οι αλγόριθμοι νομιμοποίησης. Αναλαμβάνουν να μετατρέψουν μία πιθανά μη εφαρμόσιμη αρχική τοποθέτηση σε έγκυρη και κατασκευάσιμη εκδοχή. Οι αλγόριθμοι τύπου Tetris, θεωρούνται καθιερωμένοι για την αποδοτικότητα και την απλότητά τους. Ωστόσο, υστερούν σε δείκτες αξιολόγησης της ποιότητάς τους, όπως την συνολική μετακίνηση των κελιών ή το συνολικό μήκος καλωδίου. Αυτό μπορεί να επηρεάσει καθοριστικά την κατανάλωση ενέργειας του κυκλώματος και την συνολική απόδοσή του. Έτσι, κρίνεται αναγκαία η βελτιστοποίηση των αποτελεσμάτων τέτοιων αλγορίθμων, μέσω ευρετικών προσεγγίσεων, για αυξημένη αξιοπιστία, μικρότερο χρόνο εκτέλεσης και λιγότερη κατανάλωση ενέργειας. Η παρούσα εργασία επικεντρώνεται σε αυτό το πεδίο και παρουσιάζει με λεπτομέρεια διαφορετικές εκδοχές του αλγορίθμου, καθώς και τον τρόπο με τον οποίο αλλάζουν τα αποτελέσματά του όταν εφαρμόσουμε διαφορετικές αρχικές συνθήκες.

ABSTRACT

One of the most important stages in the physical design process of VLSI circuits is the regularization algorithms. They undertake to transform a potentially infeasible initial placement into a valid and manufacturable version. Tetris-type algorithms are considered well-established for their efficiency and simplicity. However, they lag behind in quality evaluation indicators, such as the total movement of cells or the total cable length. This can decisively affect the energy consumption of the circuit and its overall performance. Thus, it is necessary to optimize the results of such algorithms, through heuristic approaches, for increased reliability, shorter execution time and less energy consumption. The present work focuses on this field and presents in detail different versions of the algorithm, as well as the way in which its results change when we apply different initial conditions.

Λαμία έτος 2024

Περιεχόμενα

ΚΕΦΑΛΑΙΟ 1

Εισαγωγή

Η ολοκλήρωση πολύ μεγάλης κλίμακας ή Very Large Scale Integration (VLSI) είναι η διαδικασία δημιουργίας ενός ολοκληρωμένου κυκλώματος (Integrated Circuit) συνδυάζοντας εκατομμύρια ή δισεκατομμύρια τρανζίστορ MOS σε ένα μόνο τσιπ. Το VLSI ξεκίνησε τη δεκαετία του 1970 όταν αναπτύχθηκαν τσιπ ολοκληρωμένων κυκλωμάτων MOS (ημιαγωγών οξειδίου μετάλλου) και στη συνέχεια υιοθετήθηκαν ευρέως, επιτρέποντας πολύπλοκες τεχνολογίες ημιαγωγών και τηλεπικοινωνιών. Ο μικροεπεξεργαστής και τα τσιπ μνήμης είναι συσκευές VLSI. Πριν από την εισαγωγή της τεχνολογίας VLSI, τα περισσότερα ολοκληρωμένα κυκλώματα είχαν ένα περιορισμένο σύνολο λειτουργιών που μπορούσαν να εκτελέσουν. Ένα ηλεκτρονικό κύκλωμα μπορεί να αποτελείται από CPU, ROM, RAM και άλλα. Το VLSI επιτρέπει στους σχεδιαστές κυκλωμάτων να προσθέσουν όλα αυτά σε ένα τσιπ. Η διαδικασία σχεδίασης VLSI περιλαμβάνει διάφορα στάδια: απαιτήσεις λογισμικού, αρχιτεκτονικό σχεδιασμό, συμπεριφορικό ή λειτουργικό σχεδιασμό, λογική σχεδίαση, σχεδιασμό κυκλώματος, φυσικό σχεδιασμό, υποδειγματικό σχεδιασμό, κατασκευή και συσκευασία και δοκιμή και συσκευασία. Όλα αυτά είναι απαραίτητα για τη δημιουργία του νέου κυκλώματος, ξεκινώντας από την ιδέα μέχρι και την λειτουργία του.

Στη συνολική διαδικασία σχεδιασμού ολοκληρωμένων κυκλωμάτων VLSI, το στάδιο της τοποθέτησης (placement) είναι ένα ζωτικό βήμα του φυσικού σχεδιασμού. Σε αυτή τη φάση καθορίζονται οι ακριβείς θέσεις όλων των βασικών ψηφιακών κυψελών (standard cells) μέσα στην περιοχή του πυρήνα του chip, έτσι ώστε να μην υπάρχουν επικαλύψεις και να τηρούνται κρίσιμες προδιαγραφές όπως ο χρονισμός, η κατανάλωση ισχύος και η δυνατότητα δρομολόγησης.

Στο βήμα του φυσικού σχεδιασμού (physical design) των κυκλωμάτων VLSI, υπάρχουν δύο φάσης. Αυτή της συνολικής τοποθέτησης (global placement) και αυτή της νομιμοποίησης της τοποθέτησης (legalization). Στην πρώτη όλα τα στοιχεία τοποθετούνται προσεγγιστικά στις θέσεις του μέσα στον πυρήνα του chip. Δίνεται έμφαση σε μία συνολική βέλτιστη διάταξη, χωρίς να εφαρμόζονται ακόμη όλοι οι περιορισμοί θέσης. Στην δεύτερη φάση, όλα τα

στοιχεία του κυκλώματος (π.χ. cells) μετακινούνται ώστε να εξαλειφθούν οι επικαλύψεις. Το εργαλείο νομιμοποίησης μετακινεί τοπικά τις κυψέλες ώστε κάθε μία να καταλάβει μια έγκυρη θέση στο πλέγμα (rows) του chip, ευθυγραμμίζοντας π.χ. τις επαφές ισχύος με τις αντίστοιχες ράγες τροφοδοσίας και τηρώντας όλους τους κανόνες σχεδίασης.

Η τοποθέτηση των στοιχείων έχει άμεση επίδραση στην απόδοση του τελικού chip. Με την βελτιστοποίηση της τοποθέτησης μπορούμε να μειώσουμε σημαντικά τις καθυστερήσεις, να εξοικονομήσουμε ενέργεια και να αξιοποιήσουμε καλύτερα τον διαθέσιμο χώρο. Βασικοί στόχοι στον σχεδιασμό κυκλωμάτων είναι: η βελτίωση χρονισμού, η αποδοτικότερη κατανάλωση ισχύος, η μείωση της συμφόρησης στην δρομολόγηση και η ελαχιστοποίηση της έκτασης (επιφάνειας) του chip.

Ο αλγόριθμος νομιμοποίησης Tetris είναι μία από τις πιο διαδεδομένες μεθόδους για την εργασία της νομιμοποίησης. Πρόκειται για έναν άπληστο (greedy) και ταχύ αλγόριθμο[2] , ο οποίος χρησιμοποιείται ευρέως στην βιομηχανία για την τακτοποίηση των κελιών χωρίς επικαλύψεις. Η βελτιστοποίηση των αποτελεσμάτων του είναι κρίσιμη, όχι μόνο για την απόδοση του κυκλώματος αλλά και για την διευκόλυνση των επόμενων σταδίων του σχεδιασμού (δρομολόγηση, υλοποίηση).

Συγκεκριμένα, όταν ο αλγόριθμος καταφέρνει να τοποθετήσει τα στοιχεία πιο βέλτιστα πετυχαίνουμε σημαντική μείωση του μισού περιμέτρου καλωδίωσης (HPWL) σε σύγκριση με τις απλές άπληστες μεθόδους. Έτσι, το τελικό chip μπορεί να τρέξει σε υψηλότερη συχνότητα ή στον ίδιο χρονισμό με λιγότερες απαιτήσεις. Επίσης, βελτιωμένες εκδοχές του Tetris ελαχιστοποιούν τις περιττές μετακινήσεις (displacement) των κελιών από τις θέσεις της συνολικής τοποθέτησης. Έχουμε λοιπόν, μία διάταξη η οποία είναι σε μεγάλο βαθμό παρόμοια με την αρχική βέλτιστη λύση που είχε βρεθεί στο global placement, κάτι που διευκολύνει την δρομολόγηση. Με μία αποδοτική και ομοιόμορφη διάταξη επιτυγχάνουμε επίσης καλύτερη κατανομή ισχύος και θερμότητας στο chip, αποφεύγοντας σημεία υπερβολικής πυκνότητας.

Συμπερασματικά, η βελτιστοποίηση των αποτελεσμάτων του αλγορίθμου τοποθέτησης Tetris έχει καθοριστική σημασία για το **τελικό αποτέλεσμα του VLSI σχεδιασμού**. Ένας πιο αποδοτικός αλγόριθμος τοποθέτησης μπορεί να προσφέρει **ταχύτερα, οικονομικότερα και μικρότερα** ολοκληρωμένα κυκλώματα. Καθώς οι απαιτήσεις για επιδόσεις και πυκνότητα ολοκληρωμένων αυξάνονται, τέτοιες βελτιστοποιήσεις στην τοποθέτηση γίνονται ολοένα πιο

σημαντικές, συμβάλλοντας στην δυνατότητα να υλοποιούμε σύνθετα συστήματα σε ένα μόνο chip με επιτυχία. Οι συνεχιζόμενες βελτιώσεις στον αλγόριθμο Tetris και σε συναφείς αλγόριθμους τοποθέτησης αποτελούν σημαντικό κομμάτι της εξέλιξης των εργαλείων αυτοματοποιημένου σχεδιασμού (EDA) και θα συνεχίσουν να παίζουν κεντρικό ρόλο στη σχεδίαση αποδοτικών κυκλωμάτων VLSI.

Για την καλύτερη ανάλυση και σύγκριση των αποτελεσμάτων αναπτύχθηκαν αρκετές εκδοχές του αλγορίθμου κάθε μία από τις οποίες υιοθετούν την δική τους τεχνική τοποθέτησης αξιοποιώντας όμως πάντοτε την βασική λογική του κλασικού αλγορίθμου. Πραγματοποιήθηκαν πειράματα πάνω σε σύνολα δεδομένων τα οποία προσομοιώνουν ρεαλιστικές συνθήκες σχεδιασμού ενός τσιπ αξιολογώντας έτσι την αποτελεσματικότητα κάθε μεθόδου που εφαρμόστηκε. Η εργασία είναι δομημένη με τρόπο ώστε να γίνεται σαφής η προσέγγιση στην βελτίωση των αποτελεσμάτων του αλγορίθμου. Συγκεκριμένα στο δεύτερο κεφάλαιο θα παρουσιάσουμε τις βασικές έννοιες και την θεωρητική βάση του VLSI Design και του αλγόριθμου TETRIS. Στο τρίτο κεφάλαιο θα δούμε με λεπτομέρεια τις ευρετικές προσεγγίσεις που χρησιμοποιήθηκαν για την βελτίωση. Ύστερα, θα αναλύσουμε τα πειραματικά αποτελέσματα και θα συγκρίνουμε τις εκδόσεις του αλγόριθμου. Τέλος, στο τελευταίο κεφάλαιο, συνοψίζονται τα ευρήματα της έρευνας και προτείνονται πιθανές κατευθύνσεις για μελλοντική βελτίωση. Η παρούσα εργασία μπορεί να αποτελέσει τη βάση για περαιτέρω έρευνα και βελτίωση των τεχνικών τοποθέτησης με σκοπό την δημιουργία αποδοτικότερων σχεδιασμών στο πεδίο του VLSI Design.

ΚΕΦΑΛΑΙΟ 2

Tetris – Like Αλγόριθμοι Νομιμοποίησης και Παραλλαγές

Στο κεφάλαιο αυτό θα αναλύσουμε τις απλές παραλλαγές του αλγόριθμου νομιμοποίησης Tetris που υλοποιήσαμε. Για την αξιολόγηση της αποτελεσματικότητας, τόσο των απλών παραλλαγών του αλγόριθμου, όσο και των ευρετικών προσεγγίσεων (που θα αναλύσουμε σε επόμενο κεφάλαιο), έχουμε χρησιμοποιήσει ένα σύνολο από 18 benchmarks της σειράς IBM, τα οποία περιέχουν τυπικά είδη αρχείων όπως .nodes, .nets, .scl, .pl κ.α. Αυτά τα αρχεία, προσομοιώνουν ρεαλιστικά σενάρια σχεδίασης κυκλωμάτων. Περιέχουν πληροφορίες για τα χαρακτηριστικά και τις αρχικές θέσεις των κελιών, τις διαστάσεις του πυριτίου, καθώς και τις δικτυακές συνδέσεις των κελιών. Η αξιοποίηση τέτοιων benchmarks αποτελεί καθιερωμένη πρακτική επειδή παρέχουν μία αντικειμενική βάση σύγκρισης μεταξύ διαφορετικών εκδοχών του αλγορίθμου. Οι αρχικές θέσεις των κελιών προήλθαν από την εφαρμογή τριών global placement αλγόριθμων. Συγκεκριμένα των: EPLACE, GORDIAN και MPL6. Αυτοί οι αλγόριθμοι δημιουργούν μία αρχική εκτίμηση των θέσεων των κελιών. Δεν υπολογίζουν όμως, περιορισμούς νομιμότητας. Συνολικά ελέγχθηκαν τρεις «18-άδες» από benchmarks, μία για κάθε placer. Με αυτό τον τρόπο μπορέσαμε να αξιολογήσουμε την γενικότητα των παραλλαγών, εφόσον τις δοκιμάσαμε σε διαφορετικά αρχικά σενάρια. Εκτός από την υλοποίηση των παραλλαγών του αλγόριθμου, έχουμε αναπτύξει ένα σύστημα ανάγνωσης και επεξεργασίας των αρχείων εισόδου. Την βάση του αποτελούν 4 αναλυτές (parsers), για τους κύριους τύπους των αρχείων και οι αντίστοιχες δομές δεδομένων. Τέλος, έχουμε υλοποιήσει βοηθητικές συναρτήσεις για τον υπολογισμό του displacement και του THPWL, καθώς και συνάρτηση απεικόνισης της τελικής διάταξης για επαλήθευση του αποτελέσματος.

(2.1 Αναλυτές)

Για να μπορέσουμε να επεξεργαστούμε τα δεδομένα που μας παρέχουν τα IBM benchmarks έχουμε υλοποιήσει μία σειρά από συναρτήσεις ανάλυσης (parsers), οι οποίες εξάγουν πληροφορίες από τα αρχεία των test cases και τις αποθηκεύουν σε κατάλληλες δομές. Από τα έξι συνολικά αρχεία που περιέχει ο κάθε φάκελος ibm / test case, χρειαζόμαστε δεδομένα μόνο από τα τέσσερα, γι' αυτό έχουμε υλοποιήσει τέσσερις αναλυτές τους οποίους εξηγούμε παρακάτω:

(2.1.1 Αναλυτής αρχείων .nodes)

Για τον αναλυτή των αρχείων .nodes έχουμε υλοποιήσει την συνάρτηση ***parse_nodes_file***, η οποία δέχεται σαν παραμέτρους το όνομα του αρχείου προς ανάλυση και δείκτη με όνομα ***nodes_file*** προς την δομή ***NodesFile***, όπου και θα αποθηκευτούν τα δεδομένα. Η δομή ***NodesFile*** έχει τρία πεδία:

i. Την ακέραια μεταβλητή ***num_nodes*** στην οποία αποθηκεύεται ο συνολικός αριθμός κελιών του αρχείου
ii. Η ακέραια μεταβλητή ***num_terminals*** στην οποία αποθηκεύεται ο συνολικός αριθμός κελιών τύπου ***terminal***
iii. Και ο δείκτης nodes, που δείχνει σε έναν πίνακα από δομές ***Node***

Η δομή Node με την σειρά της έχει τα εξής πεδία:

i. Την μεταβλητή name τύπου char για την αποθήκευση του ονόματος του κελιού
ii. Την ακέραια μεταβλητή ***width*** για το πλάτος του κελιού
iii. Την ακέραια μεταβλητή ***height*** για το ύψος του κελιού
iv. Την ακέραια μεταβλητή ***is_terminal*** που περιέχει 0 ή 1 ανάλογα με το αν το κελί είναι κανονικό ή ***terminal*** αντίστοιχα.

Ο αναλυτής αρχικά, προσπαθεί να ανοίξει το αρχείο .nodes. Σε περίπτωση αποτυχίας εμφανίζεται κατάλληλο μήνυμα σφάλματος. Γίνεται αρχικοποίηση των μεταβλητών ***num_nodes***, ***num_terminals*** και ***nodes*** σε 0. Πάντα στην αρχή όλων των .nodes αρχείων περιέχονται πληροφορίες άχρηστες προς την λειτουργία του αλγορίθμου γι' αυτό επιλέγουμε να τις αγνοήσουμε. Με την χρήση της ***sscanf*** εντοπίζουμε τα δεδομένα: ***NumNodes*** και ***NumTerminals*** στο αρχείο, από τα οποία εξάγουμε τον συνολικό αριθμό κελιών και terminal κελιών αντίστοιχα. Δεσμεύουμε δυναμικά μνήμη για τον πίνακα δομών Node, ανάλογα με τον συνολικό αριθμό κελιών που θα εξάγουμε. Σε περίπτωση

αποτυχίας το αρχείο κλείνει. Μέσω ενός ***do-while*** βρόχου, διαβάζουμε τις επόμενες γραμμές του αρχείου. Για κάθε κελί, δημιουργούμε μια τοπική μεταβλητή ***node*** και αρχικοποιούμε το ***flag***: ***is_terminal*** σε 0. Ελέγχουμε αν υπάρχει η ένδειξη: ***terminal*** δίπλα από το κελί. Αν ναι, τότε το ***flag*** γίνεται 1. Τέλος, το αρχείο κλείνει και η συνάρτηση επιστρέφει 0 για ένδειξη επιτυχίας.

(2.1.2 Αναλυτής αρχείων .nets)

Ο αναλυτής των αρχείων .nets υλοποιείται στην συνάρτηση ***parse_nets_file*** και γεμίζει την δομή ***NetsFile***. Δέχεται σαν παραμέτρους το όνομα του αρχείου προς ανάλυση και τον δείκτη ***nets_file***, που δείχνει στην δομή ***NetsFile***. Η δομή ***NetsFile*** περιέχει τρία πεδία:

i. Την ακέραια μεταβλητή ***num_nets*** στην οποία αποθηκεύεται το σύνολο των δικτύων μεταξύ των κελιών του κυκλώματος
ii. Την ακέραια μεταβλητή ***num_pins*** στην οποία αποθηκεύεται το σύνολο των ***pins***
iii. Ο δείκτης ***nets***, προς της δομή ***Net***

Η δομή ***Net*** αντιπροσωπεύει ένα δίκτυο σύνδεσης στο .net αρχείο και περιέχει τα εξής πεδία:

i. Την ακέραια μεταβλητή ***degree***, στην οποία αποθηκεύεται το πλήθος των ***pins*** στο συγκεκριμένο δίκτυο
ii. Τον δείκτη ***pins*** προς μία δομή ***Pin***
iii. Και τον δείκτη ***cell_names***, ο οποίος δείχνει σε έναν πίνακα συμβολοσειρών με τα ονόματα των κελιών

Όπως και στον προηγούμενο αναλυτή παραβλέπουμε τις άχρηστες πληροφορίες και αναζητούμε τα δεδομένα: ***NumNets*** και ***NumPins***, τα οποία μας δίνουν τον συνολικό αριθμό των δικτύων και των ***pins***. Δεσμεύουμε μνήμη για την δομή ***Net***, ανάλογη με τον συνολικό αριθμό δικτύων. Μέσω ενός βρόχου ***while***, διαβάζουμε κάθε γραμμή και εξάγουμε το πλήθος των pins στο τρέχων δίκτυο από το ***NetDegree***. Ακολουθεί η επεξεργασία των ***pins***. Διαβάζεται μία νέα γραμμή, εξάγουμε το όνομα του κελιού και τον χαρακτήρα που αντιπροσωπεύει την διεύθυνση του. Το ***flag***: ***is_pin*** παίρνει τιμή 1 σε περίπτωση που ο χαρακτήρας είναι I ή O (Input / Output Port), διαφορετικά 0. Αντιγράφουμε

το όνομα στον πίνακα ***cell_names***. Αν υπάρξει αποτυχία μνήμης για κάποιο όνομα, γίνεται αποδέσμευση και επιστρέφεται σφάλμα. Το αρχείο κλείνει και επιστρέφει 0 για επιτυχία.

(2.1.3 Αναλυτής αρχείων .scl)

Ο τρίτος αναλυτής βασίζεται στην συνάρτηση ***parse_scl_file***, και κάνει ανάγνωση των αρχείων .scl. Γεμίζει την δομή ***SclFile***. Αυτή, περιέχει δύο πεδία:

i. Την ακέραια μεταβλητή ***num_rows***, στην οποία αποθηκεύεται το πλήθος των σειρών του τρέχοντος ***test case***
ii. Ο δείκτης ***rows***, που δείχνει στην δομή ***Row***

Η δομή ***Row*** περιέχει τα εξής πεδία:

i. Την ακέραια μεταβλητή coordinate για την «θέση» της σειράς
ii. Την ακέραια μεταβλητή ***height*** (ύψος της σειράς)
iii. Την ακέραια μεταβλητή ***site_width*** (πλάτος κάθε site)
iv. Την ακέραια μεταβλητή ***site_spacing*** (απόσταση μεταξύ των ***site***)
v. Την μεταβλητή τύπου ***char***: ***site_orient*** (προσανατολισμός των ***sites***)
vi. Την μεταβλητή τύπου ***char***: ***site_symmetry*** (συμμετρία των ***sites***)
vii. Την ακέραια μεταβλητή ***subrow_origin*** (η θέση στην οποία μπορεί να τοποθετηθεί ένα κελί)
viii. Την ακέραια μεταβλητή ***num_sites*** (ο συνολικός αριθμός των κελιών που μπορεί να χωρέσει η σειρά)

Πρώτα, ανοίγουμε το αρχείο προς ανάλυση και αρχικοποιούμε τα πεδία ***num_rows*** και ***rows*** της δομής. Διαβάζουμε δεδομένα μέχρι να βρούμε το ***NumRows*** από το οποίο εξάγουμε το πλήθος των σειρών. Το αποθηκεύουμε στο ***num_rows***. Δεσμεύουμε μνήμη για τον πίνακα δομών ***Row***, ανάλογη με το πλήθος των σειρών. Στη συνέχεια, αναζητούμε την φράση: ***CoreRow Horizontal***, η οποία υποδηλώνει ότι ακολουθεί μπλοκ πληροφοριών σχετικά με μία σειρά. Με βρόχο ***while***, περνάμε όλα τα δεδομένα του μπλοκ και τα αποθηκεύουμε στα κατάλληλα πεδία της δομής ***Row***. Όταν συναντήσουμε την λέξη: ***End***, σημαίνει πως το μπλοκ πληροφορίων της σειράς τελείωσε και την

αποθηκεύουμε στην δομή. Ο βρόχος σταματάει όταν έχουν διαβαστεί όλα τα μπλοκ με πληροφορίες από το αρχείο. Τέλος, κλείνει και επιστρέφει 0.

(2.1.4 Αναλυτής αρχείων .pl)

Ο τελευταίος αναλυτής υλοποιείται στην συνάρτηση: ***parse_pl_file***, είναι υπεύθυνος για την ανάγνωση των αρχείων .pl και το γέμισμα της δομής ***PlFile***. Αυτή περιέχει τα εξής πεδία:

i. Την ακέραια μεταβλητή ***num_placements***, που περιέχει το πλήθος των τοποθετήσεων (ή αρχικών θέσεων των κελιών)
ii. Τον δείκτη placements, προς την δομή ***Placement***

Η δομή Placement, αποτελείται από τα πεδία:

i. ***name***, ακέραια μεταβλητή για το όνομα του κελιού
ii. Τις μεταβλητές τύπου ***double***: x και y (συντεταγμένες της αρχικής θέσης του κελιού)
iii. Την μεταβλητή τύπου ***char***: ***orientation***, που υποδηλώνει τον προσανατολισμό του κελιού

Αφού ανοίξουμε το αρχείο προς ανάλυση, αρχικοποιούμε τα πεδία ***num_placements*** και ***placements***. Αγνοούμε την επικεφαλίδα του αρχείου, καθώς και τις γραμμές που ξεκινούν με δίεση, κενές γραμμές ή τέλος γραμμής. Με αυτό τον τρόπο κρατάμε μόνο το σύνολο έγκυρων τοποθετήσεων. Δεσμεύουμε μνήμη για τον πίνακα δομών Placement με μέγεθος ίσο με τις τοποθετήσεις. Εξάγει τα δεδομένα και τα αποθηκεύουμε στα πεδία της δομής. Σε περίπτωση που συναντήσουμε τον χαρακτήρα «:» σημαίνει ότι μετά τις συντεταγμένες του κελιού, ακολουθεί ο προσανατολισμός του. Αν λείπει, τίθεται ως προεπιλεγμένη τιμή «N». Ο μετρητής αρχικών θέσεων κελιών ***placement_index***, ενημερώνεται. Τέλος, κλείνει το αρχείο και επιστρέφει 0.

(2.2 Βοηθητικές Συναρτήσεις)

Εκτός από τους αναλυτές, υλοποιήσαμε και ορισμένες βοηθητικές συναρτήσεις οι οποίες επεξεργάζονται τα δεδομένα των αναλυτών και τα μετατρέπουν σε μία πιο χρήσιμη μορφή. Θα ξεκινήσουμε εξηγώντας τις συναρτήσεις που κάνουν γενικότερους υπολογισμούς και χρησιμοποιούνται από όλες τις εκδοχές του αλγορίθμου Tetris. Τα δύο βασικότερα παραδείγματα είναι οι συναρτήσεις για τον υπολογισμό των διαστάσεων του κάθε κυκλώματος:

(2.2.1 Υπολογισμός Πλάτους)

Για την διαδικασία τοποθέτησης των κελιών πάνω στο κάθε κύκλωμα ***/ test case***, είναι απαραίτητο να γνωρίζουμε το πλάτος του. Εφόσον τα αρχεία ***scl*** δεν μας παρέχουν άμεσα τις διαστάσεις του κυκλώματος, αλλά πληροφορίες από τις οποίες μπορούμε να το υπολογίσουμε, δημιουργήσαμε μία συνάρτηση ***calculate_chip_width*** με αυτό το σκοπό. Δέχεται σαν παράμετρο τον δείκτη ***scl_file***, προς την δομή ***SclFile***. Δηλώνουμε και αρχικοποιούμε την ακέραια μεταβλητή ***max_width*** με 0. Αυτή θα κρατήσει το τελικό x όλων των σειρών του κυκλώματος. Δηλαδή, το σημείο που βρίσκεται δεξιότερα από κάθε άλλο στο τσιπ. Μέσα σε έναν βρόχο ***for*** για κάθε σειρά του κυκλώματος, δηλώνουμε έναν δείκτη row προς την δομή ***Row*** για εύκολη πρόσβαση σε πληροφορίες για την σειρά. Τώρα υπολογίζουμε το τελικό x. Προσθέτουμε στην αρχική συντεταγμένη x της κάθε σειράς (***subrow_origin***) το γινόμενο των διαθέσιμων θέσεων στην σειρά που μπορούν να φιλοξενήσουν ένα κελί με το πλάτος κάθε μίας. Το αποτέλεσμα της παραπάνω πράξης το αποθηκεύουμε στην μεταβλητή ***row_end***. Τώρα, αν η τιμή που περιέχει το ***row_end*** είναι μεγαλύτερη από αυτή στο ***max_width***, την αποθηκεύουμε στην δεύτερη. Έτσι, όταν τερματίσει ο βρόχος, θα έχουμε στο ***max_width*** το μεγαλύτερο ***row_end*** από όλες τις σειρές. Τέλος, επιστρέφουμε το ***max_width*** που θα περιέχει το πλάτος του τσιπ.

(2.2.2 Υπολογισμός Ύψους)

Με ανάλογο τρόπο, έχουμε υλοποιήσει και την συνάρτηση calculate_chip_height, η οποία είναι υπεύθυνη για τον υπολογισμό του ύψους του κυκλώματος. Δέχεται όπως και η προηγούμενη, σαν παράμετρο τον δείκτη ***scl_file***. Δηλώνουμε και αρχικοποιούμε την μεταβλητή ***chip_height*** στην οποία θα αποθηκεύσουμε το ύψος του chip. Τώρα, στο βρόχο for που περνάει όλες τις σειρές, υπολογίζουμε το ύψος του τσιπ. Το παίρνουμε προσθέτοντας το πεδίο coordinate της σειράς (κάθετο σημείο εκκίνησης) της σειράς με το ύψος της σειράς. Αποθηκεύουμε το αποτέλεσμα στην ***row_top***. Όπως και πριν, συγκρίνουμε την τρέχουσα τιμή του ***row_top*** με αυτή στο ***chip_height***. Αν η πρώτη είναι μεγαλύτερη ενημερώνουμε την δεύτερη. Στο τέλος έχουμε το ύψος του τσιπ στο ***chip_height***.

(2.2.3 Οπτικοποίηση)

Η επόμενη βοηθητική συνάρτηση αναλαμβάνει την εργασία της εξαγωγής της τελικής τοποθέτησης σε ένα αρχείο SVG για οπτικοποίηση. Ουσιαστικά, η συνάρτηση ‘ζωγραφίζει’ οπτικά κάθε κελί πάνω στην διάταξη του τσιπ. Μας βοηθάει να εντοπίσουμε τυχόν λάθη και επικαλύψεις που έχουν προκύψει. Πρόκειται για συνάρτηση τύπου void η οποία δέχεται ως ορίσματα έναν δείκτη στο όνομα του αρχείου SVG που θα δημιουργηθεί, έναν δείκτη στον πίνακα δομών Cell, δείκτη στην δομή SclFile και μία ακέραια μεταβλητή num_cells. Αρχικά, δημιουργεί ένα αρχείο για εγγραφή με όνομα filename. Αν αποτύχει εκτυπώνεται μήνυμα σφάλματος με την perror. Στην συνέχεια καλεί και τις δύο βοηθητικές συναρτήσεις που υπολογίζουν τις διαστάσεις του τσιπ. Ορίζει το μέγεθος του καμβά στα ίδια πίξελς με το μέγεθος του τσιπ. Ξεκινάει έναν βρόχο for ο οποίος επαναλαμβάνεται για κάθε κελί της τρέχουσας σχεδίασης. Δημιουργούμε δείκτη (cell) προς την διεύθυνση του i-οστού στοιχείου του πίνακα cells για να μπορούμε να αναφερθούμε αργότερα σε αυτό. Χρειάζεται να κάνουμε ‘αντιστροφή’ του άξονα y, υπολογίζοντας πόσο απέχει το κάτω μέρος του cell από την κορυφή του καμβά επειδή το SVG έχει ανάποδο άξονα y (προς τα κάτω). Με την χρήση της fprintf σχεδιάζουμε ένα ορθογώνιο που αναπαριστά ένα κελί. Του δίνουμε πολύ λεπτό (stroke_width: 0.2), μαύρο περίγραμμα (stroke:black;) για περισσότερη λεπτομέρεια και μπλε γέμισμα (style=\"fill:lightblue;). Υπολογίζουμε το κέντρο του κελιού για να τοποθετήσουμε το κείμενο (το όνομα του). Χρησιμοποιούμε ξανά fprintf, για

το στυλ του κειμένου, το οποίο εμφανίζεται κάθετα στο κέντρο του κελιού (text-anchor="middle") με μικρή γραμματοσειρά (font-size="3"). Στο τέλος, κλείνουμε το αρχείο SVG και το αποθηκεύουμε.

(2.3 Μετρικές)

Για να αξιολογούμε τα αποτελέσματα αλγορίθμων νομιμοποίησης όπως ο Tetris, χρησιμοποιούμε μετρικές όπως το Total Half-Perimeter Wirelength (THPWL) και το Total Displacement. Αυτές, [1]αντιπροσωπεύουν την ποιότητα της ποιότητας των κελιών στο κύκλωμα. Το THPWL είναι η εκτίμηση του συνολικού μήκους καλωδίου που απαιτείται για την σύνδεση των στοιχείων μεταξύ τους. Εξαρτάται από την γεωμετρική απόσταση των κελιών / μελών ενός δικτύου. Όσο μικρότερο THPWL, τόσο χαμηλότερη καθυστέρηση και κατανάλωση ενέργειας στο τσιπ. Το Total Displacement από την άλλη, μετράει το άθροισμα των κινήσεων όλων των κελιών, από τις αρχικές θέσεις πριν την τοποθέτηση, στις τελικές. Πολύ μεγάλες κινήσεις, μπορούν να χαλάσουν αρχικά βελτιστοποιημένες τοποθετήσεις και να αυξήσουν το THPWL. Αναζητούμε τεχνικές νομιμοποίησης οι οποίες δημιουργούν μία ισορροπία μεταξύ των δύο. Πρόκειται για δύο αναπόσπαστα εργαλεία για την αξιολόγηση παραλλαγών και ευρετικών προσεγγίσεων του αλγορίθμου.

(2.3.1 Υπολογισμός THPWL)

Για τον υπολογισμό του THPWL, έχουμε υλοποιήσει την συνάρτηση ***compute_thpwl***. Δέχεται σαν παραμέτρους τον δείκτη ***nets_file*** προς την δομή με πληροφορίες για όλα τα δίκτυα του κυκλώματος. Τον δείκτη ***cell_positions***, που δείχνει στον πίνακα με τις αρχικές θέσεις των κελιών. Και τέλος, την μεταβλητή ***num_cells*** με το πλήθος των κελιών. Το τελικό άθροισμα του THPWL όλων των δικτύων θα αποθηκευτεί στην [5] μεταβλητή ***total_hpwl***. Για να μπορούμε να εκτελέσουμε δυαδική αναζήτηση στον πίνακα ***cell_positions***, χρειάζεται πρώτα να τον ταξινομήσουμε αλφαβητικά με την συνάρτηση ***qsort***. Ξεκινάμε βρόχο for για όλα τα δίκτυα στο κύκλωμα. Για καλύτερη απόδοση, χρησιμοποιούμε παράλληλη εκτέλεση του βρόχου

(OpenMP), και μοιράζουμε κάθε επανάληψή του σε διαφορετικό νήμα του επεξεργαστή. Αρχικοποιούμε τις μεταβλητές ***x_min***,***max*** και ***y_min***,***max*** οι οποίες θα αποθηκεύουν το ελάχιστο και μέγιστο x και y των κελιών του δικτύου. Ξεκινάμε δεύτερο βρόχο for, για όλα τα κελιά του δικτύου. Μέσα σε αυτόν κάνουμε την δυαδική αναζήτηση. Φτιάχνουμε προσωρινή δομή με όνομα ***CellPosition*** που κρατάει το όνομα του κάθε κελιού. Ψάχνουμε στον πίνακα ***CellPosition*** το όνομα καλώντας την συνάρτηση ***compare_placements***. Μόλις το βρει μας επιστρέφει δείκτη προς την θέση του κελιού. Ενημερώνουμε τις μεταβλητές με τα μέγιστα x και y όλων των κελιών στο δίκτυο. Υπολογίζουμε το half-perimeter wirelength του τρέχοντος δικτύου και το προσθέτουμε στο συνολικό. Στο τέλος, η συνάρτηση επιστρέφει το ολικό άθροισμα με το συνολικό THPWL.

(2.3.2 Υπολογισμός Total Displacement)

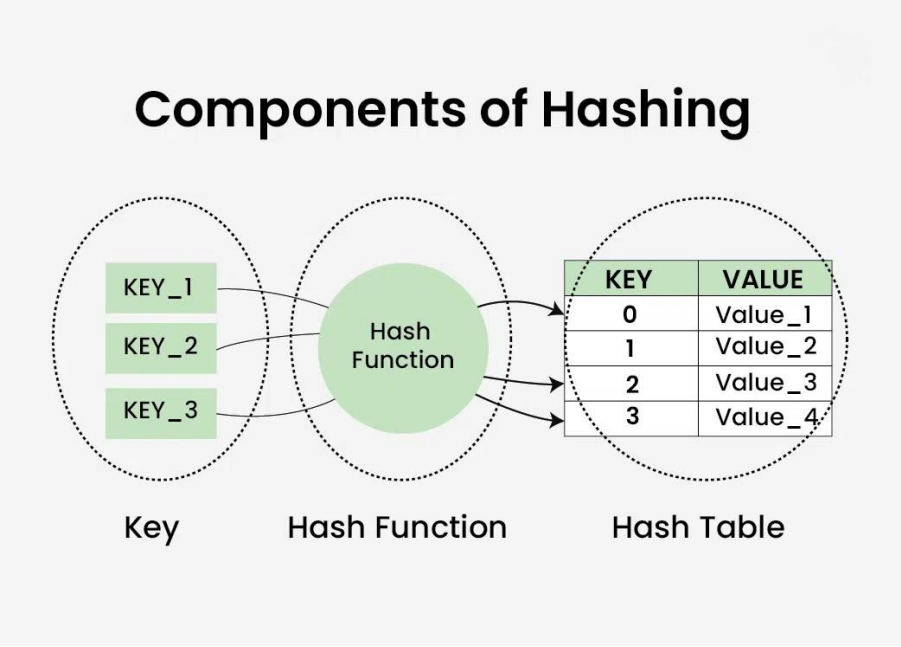
Αντίστοιχα, έχουμε υλοποιήσει την συνάρτηση ***compute_total_displacement*** για τον υπολογισμό της συνολικής μετακίνησης των κελιών. Δέχεται σαν παραμέτρους των δείκτη cells προς την δομή ***Cell***. Το πλήθος των κελιών ***num_cells***, και τον δείκτη pl_file προς την δομή ***PlFile*** που περιέχει τις αρχικές θέσεις των κελιών. Το τελικό άθροισμα της μετακίνησης θα αποθηκευτεί στην μεταβλητή ***total_displacement***. Θα χρειαστούμε πάλι δυαδική αναζήτηση, για αυτό ταξινομούμε αλφαβητικά τα κελιά του πίνακα ***placements***. Σε έναν βρόχο for για όλα τα κελιά του κυκλώματος εκτελούμε δυαδική αναζήτηση για να βρούμε τις αρχικές συντεταγμένες των κελιών. Όταν τις βρούμε, ενημερώνουμε τις μεταβλητές ***initial_x*** και ***y***. Επίσης ενημερώνουμε και τις μεταβλητές ***final_x*** και ***y*** από την δομή ***Cell*** με τις τελικές συντεταγμένες των κελιών μετά την νομιμοποίηση. Για τον υπολογισμό της μετακίνησης χρησιμοποιούμε απόσταση Manhattan:

$$\mathbf{|\,x\ final - x\ initial\,| + |\,y\ final - y\ initial\,|}$$

και προσθέτουμε το αποτέλεσμα στο συνολικό άθροισμα: ***total_displacement***. Στο τέλος το επιστρέφει.

(2.3 Πίνακας κατακερματισμού)

Έχουμε υλοποιήσει ένα hashtable το οποίο λειτουργεί ως χάρτης. Είναι υπεύθυνο για τον συσχετισμό του ονόματος κάθε κελιού με έναν αριθμό. Αυτός ο αριθμός έχει τον ρόλο του αναγνωριστικού. Το όνομα ενός κελιού (τύπου string), μετατρέπεται μέσω την συνάρτησης ***hash*** σε έναν δείκτη πίνακα. Είναι ιδιαίτερα χρήσιμος όταν έχουμε χιλιάδες κελιά με πολλές ιδιότητες. Ένα κελί έχει τα δικά του χαρακτηριστικά αλλά ταυτόχρονα ανήκει σε μεγαλύτερες ομάδες, όπως το πλήθος των δικτύων του κυκλώματος, γεγονός που του δίνει διαφορετικές ιδιότητες. Στην δικιά μας περίπτωση, το χρησιμοποιούμε όταν θέλουμε αποδώσουμε στο όνομα κάθε κελιού τις αρχικές συντεταγμένες του από το αρχείο PL.



(2.3.1 Συνάρτηση Hash)

Υπολογίζει έναν αριθμό για κάθε string, δηλαδή δημιουργεί ένα κλειδί για κάθε όνομα και σύμφωνα με την ταυτότητα:

$$h = h \times 33 + c$$

του αποδίδει ένα αναγνωριστικό. Όπου h, αρχικά είναι ίσο με 5381 (αριθμός για καλή διασπορά).

(2.3.2 Δημιουργία Hashtable)

Εδώ φτιάχνουμε και αρχικοποιούμε τον hash πίνακα. Χρησιμοποιεί ***calloc***, για τον πίνακα δεικτών HashNode, έναν για κάθε θέση στον πίνακα. Στο τέλος επιστρέφεται δείκτης στο νέο πίνακα.

(2.3.3 Εισαγωγή Κλειδιού)

Η συνάρτηση hash_table_insert, εισάγει ένα νέο ζευγάρι: όνομα κελιού – αναγνωριστικό στην λίστα. Φτιάχνει έναν καινούργιο κόμβο και τον συνδέει με την αρχή της λίστας του bucket.

(2.3.4 Αναζήτηση τιμής)

Με την συνάρτηση hash_table_search μπορούμε να αναζητήσουμε ένα κλειδί στον πίνακα και να επιστρέψουμε το αναγνωριστικό που του αντιστοιχεί μέσω ενός δείκτη. Ψάχνει τη λίστα για το κλειδί, αν το βρει επιστρέφει 1 και αποθηκεύει την τιμή στον δείκτη ***value***.

(2.3.5 Απελευθέρωση μνήμης)

Όταν δεν χρειαζόμαστε άλλο τον πίνακα, απελευθερώνουμε την δεσμευμένη μνήμη με την συνάρτηση free_hash_table. Διατρέχει την λίστα και απελευθερώνει τα κλειδιά και τους αντίστοιχους κόμβους.

(2.3 Αλγόριθμος Tetris και παραλλαγές)

Σε αυτό το υποκεφάλαιο θα παρουσιάσουμε τον κλασικό αλγόριθμο Tetris που υλοποιήθηκε και ορισμένες απλές παραλλαγές του. Οι παραλλαγές αυτές δεν αλλάζουν την βασική λογική του αλγορίθμου, όμως εξετάζουν το πως επηρεάζεται η ποιότητα της συνολικής τοποθέτησης όταν αλλάξει η στρατηγική διάταξης των κελιών πάνω στο κύκλωμα.

(2.3.1 Κλασικός Tetris)

Η λειτουργία του αλγόριθμου Tetris, στην παρούσα εργασία, βασίζεται στην συνάρτηση ***tetris_place_cells***, η οποία δέχεται μία εκτενή λίστα παραμέτρων:

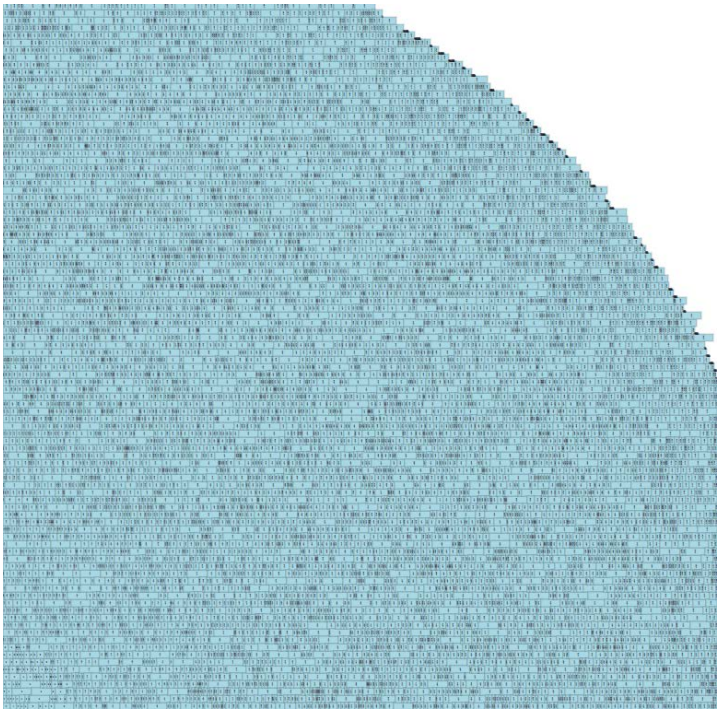
i. Δείκτη με όνομα ***cells*** προς τον πίνακα δομών ***Cell***
ii. Μεταβλητή με όνομα ***num_cells*** που περιέχει το πλήθος των κελιών
iii. Δείκτη με όνομα ***scl_file*** προς τον πίνακα δομών ***SclFile***
iv. Δείκτη με όνομα ***pl_file*** προς τον πίνακα δομών ***PlFile***

Για κάθε κελί προς τοποθέτηση, ο αλγόριθμος εκτελεί δυαδική αναζήτηση στο πεδίο ***placements*** του πίνακα ***PlFile*** για να βρει την αρχική θέση του κελιού με βάση το όνομά του. Χρησιμοποιούμε την συνάρτηση σύγκρισης ***compare_placements***, της οποίας η λειτουργία αναλύθηκε νωρίτερα. Αν βρεθεί η τοποθέτηση, ενημερώνουμε τις συντεταγμένες του κελιού από το αρχείο .pl, αλλιώς το κελί τοποθετείται προσωρινά στην θέση (0,0). Προσπελαύνουμε την τιμή στην οποία δείχνει ο δείκτης ***cells*** (dereference) για ευκολία και δημιουργούμε δύο μεταβλητές ***best_row_index*** και ***best_x_position*** τις οποίες αρχικοποιούμε με -1 και 0 αντίστοιχα. Αυτές οι μεταβλητές αποθηκεύουν τιμές που μας δείχνουν την «καλύτερη» γραμμή για τοποθέτηση του κελιού και την αντίστοιχη θέση x. Στο σημείο αυτό ξεκινάει η διαδικασία της τοποθέτησης των κελιών. Για κάθε διαθέσιμη σειρά του κυκλώματος εξάγουμε ορισμένες πληροφορίες από το πεδίο ***rows*** του πίνακα ***SclFile***:

i. Την επόμενη διαθέσιμη θέση στη σειρά (***subrow_origin***)

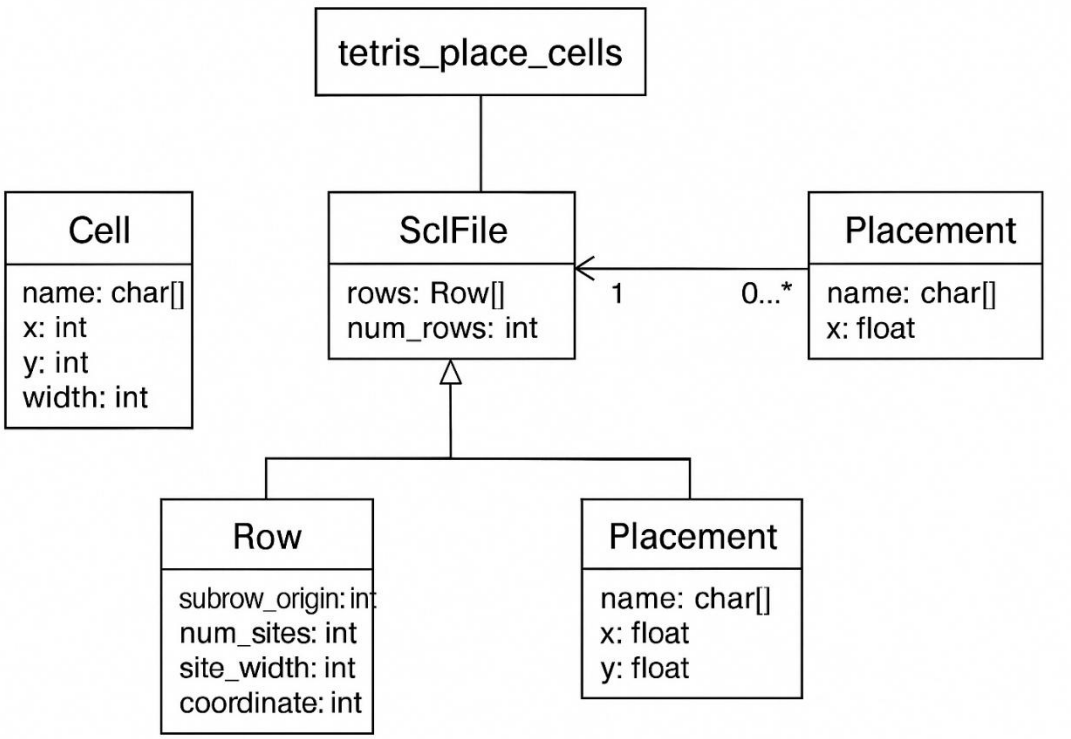
ii. Το τέλος της σειράς, το οποίο υπολογίζεται με βάση τον αριθμό θέσεων (***sites***) και το πλάτος του κάθε ***site*** (***site_width***)

Για τα δεδομένα αυτά έχουμε δημιουργήσει δύο μεταβλητές: ***row_start_x*** και ***row_end_x*** αντίστοιχα. Ελέγχουμε αν το κελί χωράει στην σειρά, αν όχι η σειρά αγνοείται και συνεχίζουμε στην επόμενη. [3]Αν χωράει, η σειρά αποθηκεύεται ως «καλύτερη» στην μεταβλητή ***best_row_index***. Αυτή η τακτική αποτελεί την χαρακτηριστική άπληστη (greedy) προσέγγιση του αλγορίθμου. Στη συνέχεια, αποφασίζουμε για το σημείο της σειράς όπου θα τοποθετήσουμε το κελί. Συγκρίνουμε την αρχική θέση του κελιού στον χώρο με την επόμενη διαθέσιμη θέση στην σειρά και διαλέγουμε αυτή που βρίσκεται πιο «δεξιά». Με τον τρόπο αυτό, αν το κελί είχε ήδη επιθυμητή θέση x και αυτή είναι ακόμη ελεύθερη, την επιλέγουμε. Αλλιώς, αν αυτή είναι κατειλημμένη, «σπρώχνουμε» το κελί πιο δεξιά για να μην έχουμε επικάλυψη με ήδη τοποθετημένα κελιά. Έτσι εξασφαλίζουμε ευθυγράμμιση με τις αρχικές θέσεις των ***benchmarks*** αλλά και προσαρμογή στα όρια του κυκλώματος. Τέλος, ενημερώνουμε την σειρά. Το ***subrow_origin*** προχωράει στο νέο κελί και μειώνουμε τα διαθέσιμα ***sites***. Σε περίπτωση που δεν έχει βρεθεί καμία κατάλληλη σειρά, εμφανίζουμε κατάλληλο μήνυμα σφάλματος.



Εικόνα 10 : Τελική τοποθέτηση του test case ibm01 με κλασικό Tetris

Στην παραπάνω εικόνα φαίνεται η οπτικοποίηση της τελικής τοποθέτησης που έγινε με την βοήθεια της συνάρτησης ***write_svg***. Παρατηρούμε πως δεν υπάρχουν επικαλύψεις και τηρούνται τα όρια που έχουν οριστεί για το κύκλωμα από το αρχείο ***.scl***.



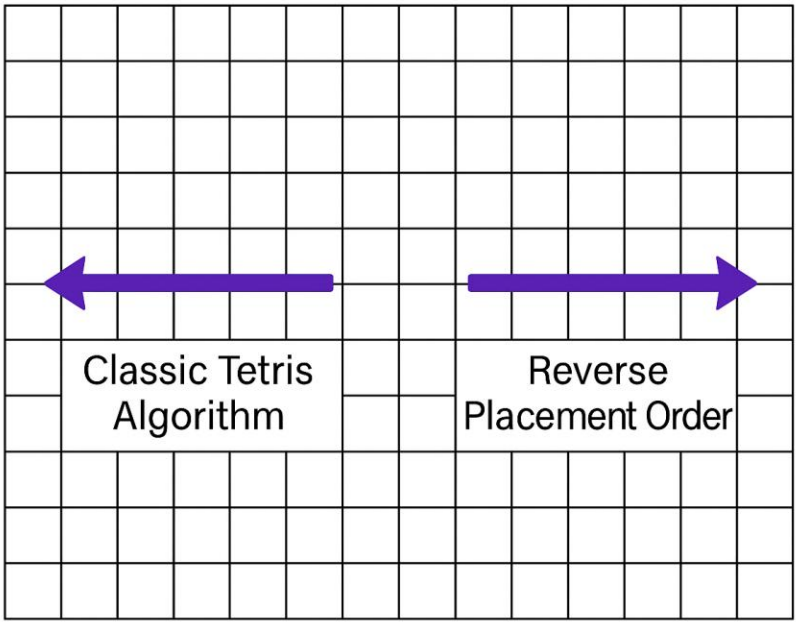
Εικόνα 11 : Διάγραμμα κλάσεων για classic Tetris

Αυτό το UML Class Diagram μας βοηθάει να καταλάβουμε ποια δεδομένα που χρησιμοποιούμε στον αλγόριθμο επηρεάζονται από άλλα. Μας δείχνει ξεκάθαρα τις ιδιότητες των βασικών δομών: ***Cell***, ***Row*** κ.λπ..

(2.3.1 Tetris με δεξιά φορά τοποθέτησης)

Η πρώτη από τις απλές παραλλαγές του αλγορίθμου είναι η «αντίθετη» του κλασικού. Σε αυτή, η τοποθέτηση ξεκινάει από το δεξιό άκρο της κάθε σειράς

και «γεμίζει» προς τα αριστερά. Όπως και οι υπόλοιπες απλές παραλλαγές, αποτελεί μία καθαρής γεωμετρικής φοράς μεταβολή του κλασικού Tetris, δηλαδή διατηρεί την αρχική λογική νομιμοποίησης.

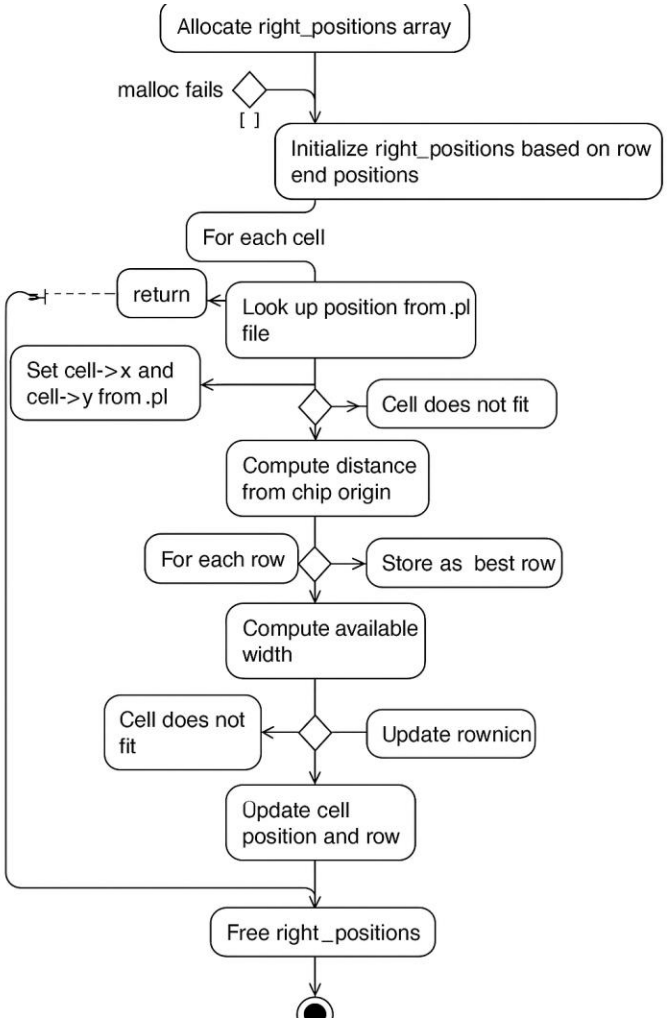


Εικόνα 12 : Πλευρά του κυκλώματος από την οποία ξεκινάει

η τοποθέτηση στις δύο εκδόσεις του αλγορίθμου

Η συνάρτηση έχει ονομαστεί ***tetris_place_cells_from_right*** και δέχεται σαν παραμέτρους τα αρχεία .pl και .scl, το πλήθος των κελιών (***num_cells***) και τον δείκτη ***cells*** προς την δομή ***Cell***. Αρχικά, δημιουργούμε τον πίνακα ***right_positions*** στον οποίο θα αποθηκεύουμε την δεξιότερη διαθέσιμη θέση για την τοποθέτηση του κελιού. Δεσμεύουμε μνήμη για τον πίνακα. Για κάθε σειρά του κυκλώματος, υπολογίζουμε την δεξιά της άκρη: ***αρχή της σειράς + πλήθος των διαθέσιμων θέσεων * πλάτος θέσης***. Την αποθηκεύουμε στον πίνακα ***right_positions***. Σε βρόχο ***for*** ο οποίος περνάει όλα τα κελιά, κάνουμε δυαδική αναζήτηση στο αρχείο .pl. Αναζητούμε τις αρχικές συντεταγμένες του κάθε κελιού με βάση το όνομα του. Αν βρεθούν, τις κρατάμε αλλιώς ορίζουμε ως αρχική θέση την 0,0. Αρχικοποιούμε τις μεταβλητές ***best_row_index*** και ***best_x_position***, που αφορούν την ιδανική θέση στην σειρά για τοποθέτηση του κελιού. Στο σημείο αυτό, ξεκινάει η αναζήτηση της καλύτερης σειράς. Το βήμα αυτό προηγείται της τελικής επιλογής για καλύτερη θέση στην σειρά. Σε ένα βρόχο ***for***, για κάθε σειρά του κυκλώματος υπολογίζουμε το διαθέσιμο

πλάτος αφαιρώντας από το ***right_positions[j]*** (τρέχουσα δεξιά άκρη) το ***subrow_origin*** (αριστερή άκρη). Εφόσον σε αυτή την έκδοση του αλγορίθμου τοποθετούμε κελιά από τα δεξιά, η μεταβλητή ***right_positions[j]*** μειώνεται σταδιακά όσο έχουμε τοποθετήσεις. Ελέγχουμε επίσης, αν το πλάτος του κελιού προς τοποθέτηση είναι μεγαλύτερο από το διαθέσιμο πλάτος για να διασφαλίσουμε πως δεν θα υπάρξει υπερχείλιση έξω από την σειρά. Στον ίδιο βρόχο υπολογίζουμε την ευκλείδεια απόσταση του δεξιού ορίου της σειράς από την αρχή του τσιπ. Αν η απόσταση είναι μικρότερη, την θεωρούμε «καλύτερη». Αφού βρήκαμε την σειρά που μας βολεύει, τοποθετούμε το κελί με την ίδια λογική όπως και στον κλασικό αλγόριθμο. Αν δεν βρεθεί σειρά, προειδοποιούμε τον χρήστη. Ενημερώνουμε το νέο δεξί όριο της σειράς και αποδεσμεύουμε τον πίνακα ***right_positions***.



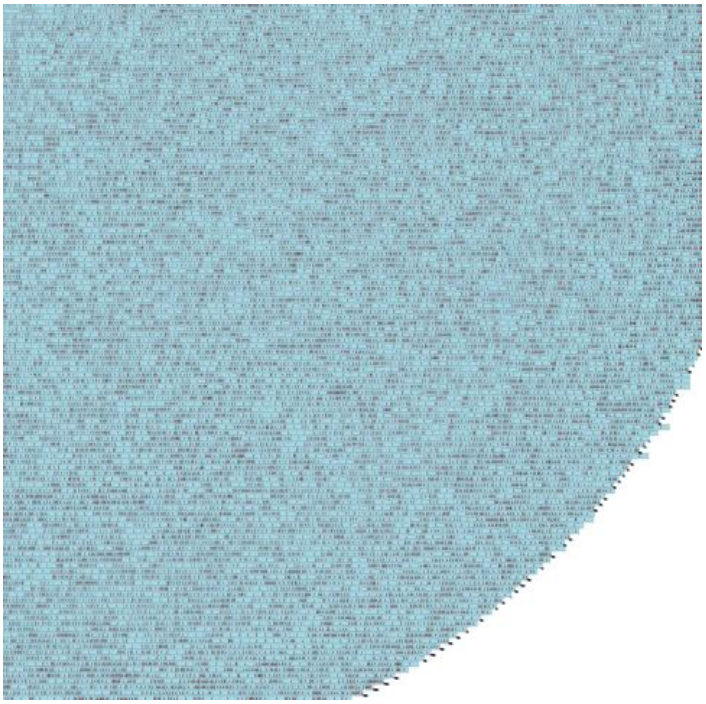
Εικόνα 13 : Διάγραμμα ροής του αλγορίθμου

(2.3.3 Tetris με φορά προς τα κάτω)

Η επόμενη παραλλαγή του αλγορίθμου προσφέρει μια εναλλακτική στρατηγική τοποθέτησης, ιδιαίτερα χρήσιμη σε περιπτώσεις που:

i. Οι «ψηλότερες» σειρές στο κύκλωμα έχουν ειδική σημασία (π.χ. απόδοση ή θερμοκρασία)
ii. Η κατανομή που ξεκινάει από το πάνω μέρος του κυκλώματος θεωρείται πιο οικονομική

Η συγκεκριμένη παραλλαγή νομιμοποιεί τα κελιά, έχοντας σαν κριτήριο επιλογής σειράς για τοποθέτηση την πιο «ψηλή» διαθέσιμη. Γεμίζει το τσιπ από πάνω προς τα κάτω.

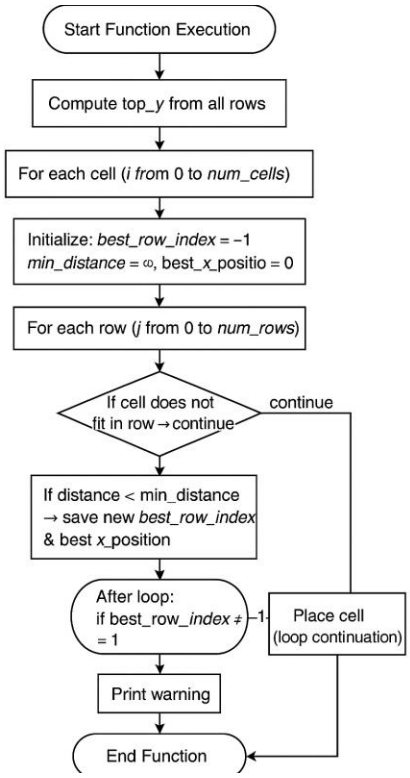


Εικόνα 14 : Τελική τοποθέτηση του test case ibm03

με φορά προς τα κάτω

Η συνάρτηση ***tetris_place_cells_from_top*** χρησιμοποιεί τα αρχεία .pl και .scl, όπως και οι προηγούμενες για τις αρχικές θέσεις των κελιών και τις

διαστάσεις του τσιπ. Αρχικά, δηλώνουμε την ακέραια μεταβλητή ***top_y*** και την αρχικοποιούμε με 0. Σε αυτή θα αποθηκεύσουμε το «ψηλότερο» σημείο του τσιπ. Για κάθε σειρά του κυκλώματος, υπολογίζουμε το άθροισμα του ***coordinate*** της, (δηλαδή του κάτω άκρου της) με το ***height*** της. Το τελικό άθροισμα θα είναι η πάνω αριστερή άκρη του τσιπ. Για κάθε κελί προς τοποθέτηση, αρχικοποιούμε τις μεταβλητές ***best_row_index***, ***min_distance*** και ***best_x_position*** στις οποίες θα αποθηκευτούν η αρχή της σειράς που θα τοποθετήσουμε το κελί, η ελάχιστη απόσταση από την κορυφή και η θέση της τοποθέτησης μέσα στην σειρά. Μέσα στον βρόχο για τα κελιά, φτιάχνουμε δεύτερο βρόχο για τις σειρές. Για κάθε σειρά υπολογίζουμε το διαθέσιμο εύρος της. Σε περίπτωση που το κελί είναι πολύ φαρδύ για την σειρά, την παραλείπουμε. Επίσης, υπολογίζουμε την απόσταση ***dy***, αφαιρώντας από την μεταβλητή ***top_y***, το άθροισμα του coordinate της σειράς με ύψος της. Αυτή η ευκλείδεια απόσταση μας δείχνει πόσο «χαμηλά» βρίσκεται η σειρά σε σχέση με την κορυφή του τσιπ. Συγκρίνουμε την απόσταση που υπολογίσαμε με την ελάχιστη μέχρι τώρα. Αν είναι μικρότερη, αποθηκεύουμε την σειρά και την θέση. Τοποθετούμε το κελί στην διαθέσιμη θέση και ενημερώνουμε το ***subrow_origin*** έτσι ώστε τα επόμενα κελιά να μπούνε πιο δεξιά του.



Εικόνα 15 : Διάγραμμα ροής Tetris με φορά κάτω

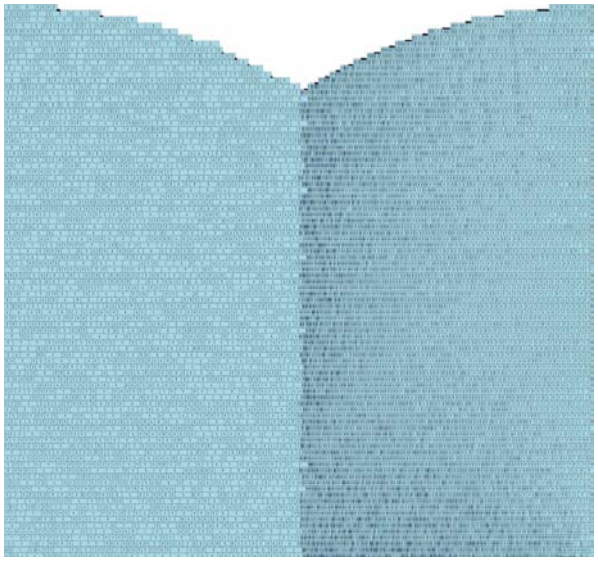
(2.3.4 Tetris με φορά προς τα πάνω)

Η συγκεκριμένη παραλλαγή του Tetris, παρά τις ομοιότητες της με τον κλασικό αλγόριθμο και την φαινομενικά απλή δομή της, εμφανίζει μία σημαντική ιδιαιτερότητα. Επιτρέπει την κάθετη στοίβαξη κελιών. Σε αντίθεση με τον κλασικό Tetris, όπου η τοποθέτηση είναι αυστηρά οριζόντια και ευθυγραμμισμένη, εδώ ένα κελί μπορεί να τοποθετηθεί πάνω από ένα άλλο και όχι αναγκαστικά δίπλα του, αρκεί να είναι κοντά στην χαμηλότερη και «αριστερότερη» δυνατή σειρά. Παρατηρούμε λοιπόν, ότι είναι πιθανή η αλλαγή σειράς για την τοποθέτηση ενός καινούργιου κελιού χωρίς απαραίτητα η προηγούμενη να είναι γεμάτη. Στον κλασικό αλγόριθμο, δεν αλλάζουμε σειρά αν υπάρχει ακόμα διαθέσιμος χώρος. Ας δούμε λοιπόν που λειτουργεί. Η συνάρτηση ***tetris_place_cells_from_bottom*** ανακτά τις αρχικές συντεταγμένες όλων των κελιών με δυαδική αναζήτηση. Μόλις τις βρει, τις αναθέτει σε κάθε κελί. Κάνουμε προετοιμασία των μεταβλητών οι οποίες θα αποθηκεύσουν πληροφορίες για την βέλτιστη σειρά. Όπως και στις προηγούμενες εκδοχές του αλγορίθμου, υπολογίζουμε την αρχή και το τέλος της κάθε σειράς για να δούμε αν χωράει ένα κελί. Υπολογίζουμε την ευκλείδεια απόσταση της «αρχής» του τσιπ (η πιο κάτω αριστερή γωνία στην συγκεκριμένη εκδοχή) από την αρχική θέση του κελιού. Δηλαδή αυτή που το ανατέθηκε από τον ***global placer*** και την έχουμε ήδη βρει με την δυαδική αναζήτηση. Αν αυτή είναι η κοντινότερη που έχουμε βρει μέχρι τώρα την αποθηκεύουμε σαν υποψήφια στην μεταβλητή: ***best_x_position***. Εφόσον έχουμε βρει την βέλτιστη σειρά, τοποθετούμε το κελί στο σημείο που δείχνει η μεταβλητή ***start_x*** (δηλαδή στην ***best_x_position***). Τέλος, ενημερώνουμε την καινούργια «αρχή» της σειράς και τα διαθέσιμα ***sites***.

(2.3.4 Tetris με συνδυασμό αριστερής και δεξιάς φοράς)

Η επόμενη παραλλαγή του Tetris εμφανίζει ιδιαίτερο ενδιαφέρον καθώς συνδυάζει την δεξιά φορά τοποθέτησης με τον κλασικό αλγόριθμο. Δηλαδή, η συνάρτηση αποφασίζει από ποια μεριά του τσιπ θα τοποθετήσει το επόμενο κελί (αριστερή ή δεξιά) με βάση το «κόστος» της. Ας δούμε πιο συγκεκριμένα:

Αρχικά, δημιουργούμε και δεσμεύουμε μνήμη για δύο δείκτες σε πίνακες ακεραίων. Τους ***left_positions*** και ***right_positions***. Εκεί θα αποθηκεύσουμε το αριστερό και δεξί όριο τοποθέτησης αντίστοιχα. . Έχουμε δημιουργήσει την μεταβλητή ***row_start_x*** που περιέχει πάντα το ενημερωμένο ***subrow_origin***. Άρα, ο δείκτης ***left_positions*** θα περιέχει πάντα την καινούργια αρχή της σειράς. Φτάνουμε στο τέλος μίας σειράς του κυκλώματος όταν έχουμε διανύσει την απόσταση: ***num_sites * site_width***. Η μεταβλητή ***row_end_x*** θα αποθηκεύει το τέλος κάθε σειράς, άρα το: ***subrow_origin + num_sites * site_width***. Ο ***right_positions*** παίρνει την τιμή της ***row_end_x***. Ξεκινάμε την τοποθέτηση των κελιών, αφού βρούμε τις αρχικές θέσεις τους από το .pl αρχείο. Αρχικοποιούμε τις μεταβλητές ***best_candidate_x*** και ***best_candidate_type*** στις οποίες θα αποθηκεύσουμε την υποψήφια θέση και το αν αυτή είναι αριστερά ή δεξιά στο τσιπ. Σε κάθε επανάληψη ελέγχουμε πάντα το διαθέσιμο χώρο που έχει μείνει στην σειρά: ***right_positions[j] – left_positions[j]***. Ορίζουμε τους δύο υποψήφιους Α και Β ως το αριστερό και δεξί άκρο της σειράς αντίστοιχα. Στο σημείο αυτό υπολογίζεται το κόστος της κάθε τοποθέτησης με βάση το οποίο παίρνουμε την τελική απόφαση. Ενδεικτικά, το κόστος τοποθέτησης ενός κελιού στην αριστερή θέση Α, υπολογίζεται από το άθροισμα της κάθετης απόστασης της θέσης από την αρχική του κελιού, με της οριζόντιας. Πιο συγκεκριμένα, για να βρούμε την κάθετη απόσταση αφαιρούμε από την συντεταγμένη y της αρχικής θέσης του κελιού που πήραμε από το αρχείο .pl (pl_y), την συντεταγμένη y της υποψήφιας θέσης (row_y). Αντίστοιχα υπολογίζουμε και το κόστος μίας δεξιάς τοποθέτησης. Εφόσον μία σειρά δεν έχει γεμίσει ακόμη, συγκρίνουμε τα αποτελέσματα και επιλέγουμε την τοποθέτηση με το μικρότερο κόστος. Ενημερώνουμε τους δείκτες και τις διαθέσιμες θέσεις που έχουν μείνει στην σειρά.



Εικόνα 15 : Τελική τοποθέτηση test case ibm01

(2.3.4 Tetris με συνδυασμό πάνω και κάτω φοράς)

Εδώ ακολουθούμε την ίδια τακτική συνδυαστικής τοποθέτησης με πριν, αλλά στην πάνω και κάτω πλευρά του τσιπ. Δεσμεύουμε πάλι μνήμη για τους δείκτες ***left*** και ***right_positions***, για τα περιθώρια της κάθε σειράς. Τους αρχικοποιούμε με το περιεχόμενο των μεταβλητών ***row_start_x*** και ***row_end_x*** αντίστοιχα όπως και στην προηγούμενη παραλλαγή. Εξάγουμε τις αρχικές συντεταγμένες των κελιών με δυαδική αναζήτηση στο .pl αρχείο. Ξεκινάμε τον βρόχο για την τοποθέτηση των κελιών, μέσα στον οποίο αρχικοποιούμε τις μεταβλητές για εύρεση της βέλτιστης θέσης. Δηλώνουμε δύο βασικές μεταβλητές. Την ***distance_to_top***, στην οποία θα αποθηκευτεί η απόσταση του κελιού από την κορυφή του τσιπ, δηλαδή το αποτέλεσμα της πράξης: συνολικό ύψος του τσιπ – (αρχή της γραμμής + ύψος της τρέχουσας σειράς) – συντεταγμένη y της αρχικής θέσης του κελιού. Και την ***distance_to_bottom***, στην οποία αποθηκεύουμε την διαφορά: αρχή της γραμμής – συντεταγμένη y. Επίσης, δηλώνουμε τις μεταβλητές ***candidate_x_left*** και ***candidate_x_right*** που θα αποθηκεύσουν το περιεχόμενο των ***left*** και ***right_positions*** – το πλάτος του κελιού αντίστοιχα. Δηλαδή κρατάνε τις υποψήφιες θέσης στην σειρά. Στην παρούσα εκδοχή του αλγορίθμου το κόστος της τοποθέτησης θα είναι η ευκλείδεια απόσταση του κελιού από την πιο πάνω ή την πιο κάτω σειρά. Άρα το άθροισμα της κάθετης και της οριζόντιας απόστασης του κελιού με την σειρά. Επομένως έχουμε:

total_cost_bottom = distance_to_bottom + horizontal_cost_left και ***total_cost_top = distance_to_top + horizontal_cost_right***. Συγκρίνουμε τις τιμές τους και τοποθετούμε το κελί στην σειρά με το μικρότερο κόστος. Ενημερώνουμε το χώρο στην σειρά και απελευθερώνουμε τον χώρο των δεικτών.

(2.3.4 Tetris με τοποθέτηση από κάθε πλευρά)

Η τελευταία από τις απλές παραλλαγές του Tetris είναι ένας συνδυασμός όλων των προηγούμενων. Εδώ, για την διαδικασία της νομιμοποίησης λαμβάνουμε υπόψη μας όλες τις πλευρές του τσιπ. Αρχικά, σε ένα βρόχο ***for*** διενεργούμε δυαδική αναζήτηση στο πεδίο ***placements*** του αρχείου pl για να βρούμε τις αρχικές θέσεις των κελιών. Ενημερώνουμε τις συντεταγμένες του κελιού με αυτές τις αρχικής θέσης. Σε καινούργιο βρόχο, ο οποίος επαναλαμβάνει για όλα τα κελιά του ***test case***, ξεκινάμε την διαδικασία της τοποθέτησης. Πρώτα, αρχικοποιούμε τις μεταβλητές για καλύτερη θέση. Σε έναν δεύτερο, εμφωλευμένο βρόχο, υπολογίζουμε τον διαθέσιμο χώρο για τοποθέτηση σε κάθε σειρά. Στην συνέχεια, υπολογίζουμε τις αποστάσεις από κάθε πλευρά του τσιπ και τις αποθηκεύουμε σε κατάλληλες μεταβλητές. Αναλυτικά: στην συνάρτηση ***distance_left*** αποθηκεύουμε το περιεχόμενο της ***row_start_x***, στην οποία νωρίτερα έχουμε «περάσει» την αρχή της σειράς (***subrow_origin***). Στην συνάρτηση ***distance_right*** αποθηκεύουμε το αποτέλεσμα της πράξης: πλάτος του τσιπ – (αρχή της σειράς + πλάτος κελιού). Στην ***distance_top***, το αποτέλεσμα της πράξης: ύψος το τσιπ – (η συντεταγμένη y της σειράς + ύψος της σειράς). Τέλος, στην ***distance_bottom***, αποθηκεύουμε το ***coordinate*** της σειράς, δηλαδή το κάθετο σημείο του κυκλώματος που βρίσκεται η σειρά. Αυτές οι μεταβλητές μας δείχνουν πόσο απέχει το κελί από τις πλευρές του τσιπ αν έχει τοποθετηθεί στην σειρά. Επιλέγουμε την μικρότερη από τις τέσσερις και την αποθηκεύουμε στην ***chosen_distance***. Αν βρούμε άλλη μικρότερη, την αντικαθιστούμε. Εφόσον έχουμε βρει την κατάλληλη σειρά και θέση, τοποθετούμε το κελί και ενημερώνουμε την χωρητικότητα της σειράς.

ΚΕΦΑΛΑΙΟ 3

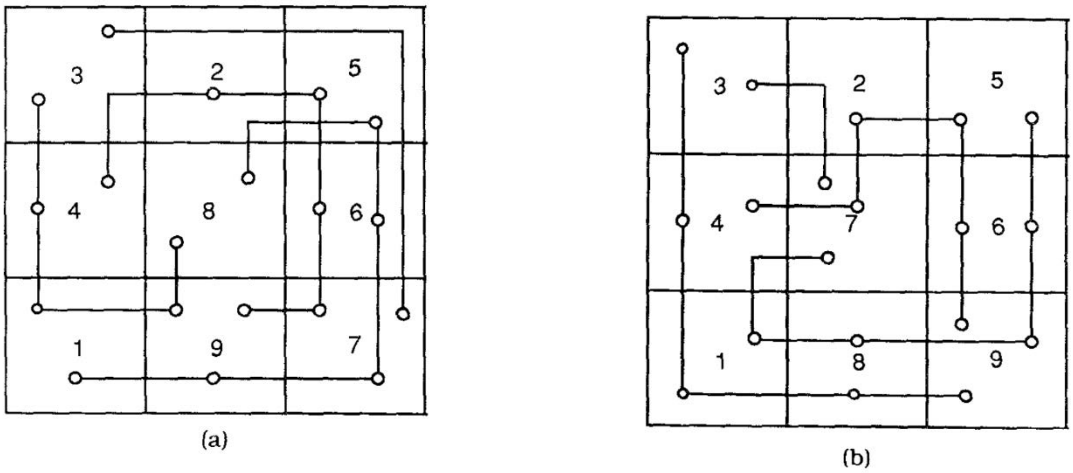
Υλοποίηση Ευρετικών Προσεγγίσεων

(3.1 Εισαγωγή)

Στο κεφάλαιο αυτό θα αναλύσουμε με λεπτομέρεια τις ευρετικές προσεγγίσεις που υλοποιήθηκαν. Αυτές οι εκδοχές του αλγορίθμου Tetris εμφανίζουν σημαντικές διαφορές στην δομή από αυτές του δεύτερου κεφαλαίου. Εδώ, έχουμε εκμεταλλευτεί πλήρως όλα τα χαρακτηριστικά των παραγόντων στην νομιμοποίηση. Για παράδειγμα, τα δίκτυα που ενώνουν τα κελιά μεταξύ τους, παίζουν καθοριστικό ρόλο σε πολλές από τις ευρετικές παραλλαγές, όπως επίσης και ο τρόπος που είναι δομημένο το κάθε κύκλωμα. Παράγοντες που είχαμε επιλέξει να αγνοήσουμε στις απλές παραλλαγές. Με απώτερο σκοπό την βελτιστοποίηση των αποτελεσμάτων του αλγορίθμου, προτείνουμε τις εξής ευρετικές τακτικές νομιμοποίησης:

(3.2 Παραλλαγές με Δίκτυα)

Οι πρώτες παραλλαγές που θα αναλύσουμε σε αυτό το κεφάλαιο, ασχολούνται με την δικτυακή συνδεσιμότητα των κελιών. Δηλαδή, δεν εξετάζουμε πλέον το κάθε κελί σαν αυτόνομο κομμάτι του κυκλώματος, αλλά σαν ένα μέρος ενός μεγαλύτερου συνόλου ή net. Με αυτό τον τρόπο αξιοποιούμε το τοπολογικό πλαίσιο του κελιού, επιλέγοντας θέσεις τοποθέτησης που ευνοούν την μείωση συμφόρησης και κατανάλωσης ενέργειας.

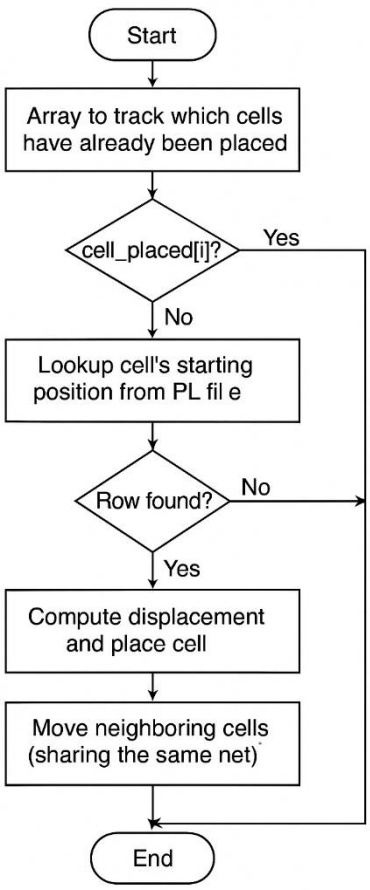


Εικόνα 16 : Παραδείγματα δικτυακών συνδέσεων μεταξύ των κελιών δύο κυκλωμάτων

(3.2.1 Tetris με Δικτυακή Συνδεσιμότητα)

Η πρώτη βασική ευρετική, βασίζεται στην κίνηση των κελιών που ανήκουν στο ίδιο δίκτυο σαν ένα σύνολο. Η συνάρτηση ονομάζεται ***tetris_place_cells_with_net_sync*** και δέχεται σαν παραμέτρους: Τον δείκτη ***cells***, προς την δομή ***Cell*** με πληροφορίες για κάθε κελί, το σύνολο των κελιών (***num_cells***), των δείκτη ***scl_file*** προς την δομή ***SclFile*** με πληροφορίες για τις σειρές του κυκλώματος, τον δείκτη ***nets_file*** προς την δομή ***NetsFile*** με πληροφορίες για κάθε δίκτυο στο κύκλωμα, τον δείκτη ***cell_map*** για πρόσβαση στο ***hashtable*** των κελιών. Τέλος, τον δείκτη ***pl_file***, προς την δομή ***PlFile*** με πληροφορίες για τις αρχικές θέσεις των κελιών. Αρχικά, δηλώνουμε και δεσμεύουμε μνήμη για έναν πίνακα ακεραίων ***cell_placed***. Αυτός, θα μας βοηθήσει να παρακολουθούμε αν τα κελιά που συναντάμε έχουν τοποθετηθεί ήδη ή όχι. Ξεκινάμε έναν βρόχο ***for*** για όλα τα κελιά. Στην συνέχεια εκτελούμε δυαδική αναζήτηση στο αρχείο ***pl*** για να βρούμε τις αρχικές θέσεις που δόθηκαν στα κελιά από τον ***global placer***. Αν τις βρούμε, περνάμε τις συντεταγμένες σε αυτές του κελιού: ***cells.x = placement.x***. Αλλιώς, δίνουμε στο κελί την αρχική θέση 0,0. Τώρα ξεκινάμε την αναζήτηση για διαθέσιμη σειρά στην οποία θα τοποθετήσουμε τα κελιά. Με τον συνολικό αριθμό σειρών που έχουμε λάβει από το αρχείο scl, ξεκινάμε έναν δεύτερο βρόχο for που διατρέχει όλες τις σειρές. Για κάθε μία, ελέγχουμε αν ***cells.width > row_end_x - row_start_x***, δηλαδή αν το πλάτος του κελιού προς τοποθέτηση

είναι αρκετά μικρό για να χωρέσει στην σειρά. Αν είναι, την κρατάμε. Σε δύο ακέραιες μεταβλητές ***dx***, ***dy***, αποθηκεύουμε τις ευκλείδειες αποστάσεις από τα αρχικά x, y. Δηλαδή, τα ***dx*** και ***dy*** είναι η συνολική μετατόπιση του κελιού. Τοποθετούμε το κελί στην διαθέσιμη θέση στην σειρά με τον ίδιο τρόπο όπως και στον κλασικό Tetris. Στο σημείο αυτό πρέπει να βρούμε τα υπόλοιπα κελιά που ανήκουν στο ίδιο δίκτυο με αυτό που μόλις τοποθετήσαμε. Εδώ γίνεται ιδιαίτερα χρήσιμη η συνάρτηση ***cell_to_nets*** με πληροφορίες για την συνδεσιμότητα των κελιών, την οποία αναλύσαμε νωρίτερα. Σε έναν βρόχο for που διασχίζει όλα τα δίκτυα του κυκλώματος, ψάχνουμε το δίκτυο στο οποίο άνηκε το κελί. Επιπλέον στο σημείο αυτό χρησιμοποιούμε και την συνάρτηση ***hashtable***. Ας δούμε πως δουλεύει. Ο πίνακας ***cell_names*** είναι ένα πεδίο της δομής ***Net***, το οποίο γεμίζει από τον ***parser*** των ***Nodes*** με τα ονόματα των κελιών ενός ***benchmark***. Μπορούμε να προσπελάσουμε αυτό τον πίνακα μόνο με βάση τον αριθμό θέσης ενός κελιού (index) και όχι με βάση το όνομα του. Για να μπορέσουμε να μετατρέψουμε το όνομα του κάθε κελιού σε index χρησιμοποιούμε την συνάρτηση ***cell_map***. Αυτή εκτελεί την μετατροπή: π.χ. a1023 ->32. Στην συνέχεια με την συνάρτηση ***hash_table_search*** έχουμε άμεση πρόσβαση στο ***index*** του κελιού αν έχουμε το όνομά του. Σε benchmarks όπως τα δικά μας, με εκατοντάδες χιλιάδες κελιά, η γραμμική αναζήτηση που θα εκτελούσαμε αντί της δυαδικής θα ήταν καταστροφική για την απόδοση. Αφού βρούμε το index κάθε κελιού που ανήκει στο δίκτυο, ελέγχουμε αν έχει τοποθετηθεί. Αν όχι, τα μετακινούμε το ίδιο (dx, dy) με αυτό που τοποθετήσαμε, κρατώντας την σχετική απόσταση του δικτύου σταθερή. Στο τέλος, αποδεσμεύουμε των πίνακα ***cell_placed***. Συνοπτικά, η συνάρτηση καταφέρνει να «συγχρονίσει» τα κελιά του ίδιου δικτύου μετακινώντας τα κατά την ίδια απόσταση.

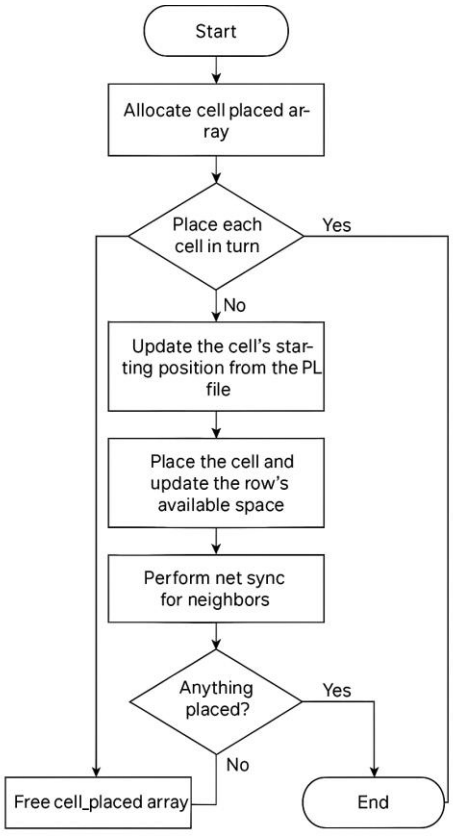


Εικόνα 17 : Διάγραμμα ροής της συνάρτησης

(3.2.2 Δικτυακή Συνδεσιμότητα με Fallback)

Η προηγούμενη έκδοση του Tetris, εκτελεί μία μόνο διέλευση από όλα τα κελιά και αν κάποιο δεν χωράει σε σειρά το προσπερνάει. Επίσης, δεν μας παρέχει κάποιο τρόπο να γνωρίζουμε πόσα κελιά προσπεράσαμε. Αυτό την καθιστά γρήγορη αλλά πρόχειρη. Για αυτό το λόγο έχουμε δημιουργήσει και μία πιο ακριβής και «επίμονη» εκδοχή. Η ***tetris_place_cells_with_net_sync_full*** κάνει όσες επαναλήψεις χρειάζεται έτσι ώστε να είναι σίγουρο ότι έχουν τοποθετηθεί όλα τα κελιά. Αρχικά, δηλώνουμε τις δύο μεταβλητές ***placed_count*** και ***anything_placed***. Στην πρώτη θα αποθηκεύουμε το σύνολο των κελιών που έχουν ήδη τοποθετηθεί και στην δεύτερη 0 ή 1 ανάλογα

με το αν τοποθετήσαμε κάποιο κελί σε αυτό το πέρασμα. Το βασικό μπλοκ κώδικα της συνάρτησης τώρα περιέχεται σε έναν βρόχο do-while, με προϋπόθεση: σε κάθε επανάληψη, αν έχουν τοποθετηθεί κελιά και το σύνολο αυτών που έχουν τοποθετηθεί είναι μικρότερο από όλα τα κελιά, συνέχισε. Η τοποθέτηση ενός κελιού καθώς και ο συγχρονισμός του δικτύου του, εκτελούνται με τον ίδιο τρόπο. Όταν εξασφαλίζουμε την συμμετοχή όλων των κελιών του test case στην τοποθέτηση, βελτιώνουμε έμμεσα το μετρικό της μετατόπισης (displacement). Αυτό συμβαίνει διότι δεν έχουν μείνει πίσω κελιά τα οποία στην συνέχεια ίσως να χρειαστούν ξανά νομιμοποίηση. Επίσης, η ομαδική μετακίνηση των κελιών μειώνει το μήκος καλωδίου, αφού τα κελιά του ίδιου δικτύου βρίσκονται πιο κοντά μεταξύ τους. Στην δεύτερη εκδοχή λοιπόν, έχουμε βελτίωση του displacement κρατώντας το THPWL εξίσου χαμηλό με της πρώτης. Θυσιάζουμε όμως χρόνο εκτέλεσης, καθώς η πολυπλοκότητα της δεύτερης συνάρτησης είναι κοντά σε O(N^2).

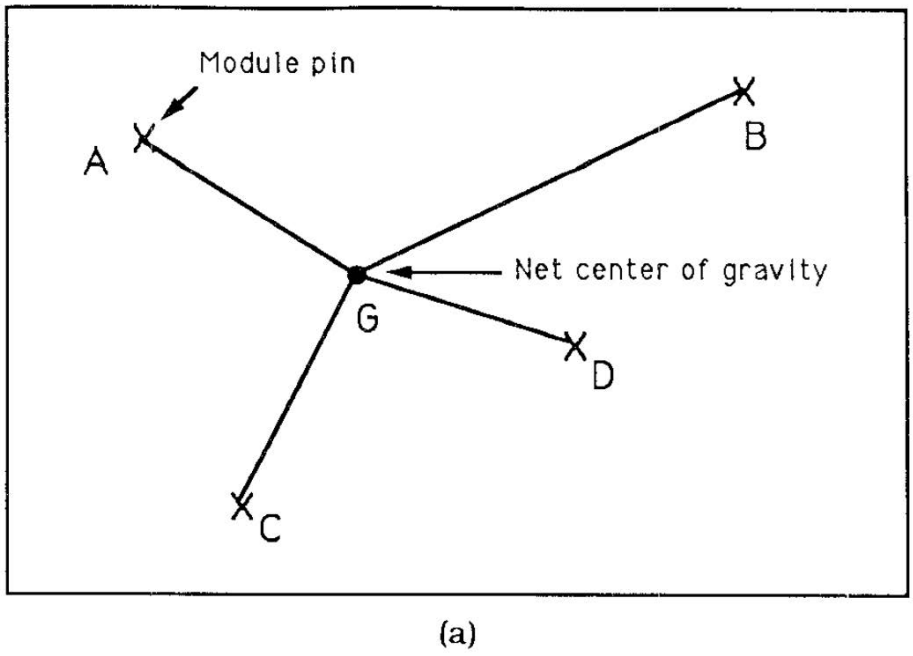


Εικόνα 17 : Διάγραμμα ροής της συνάρτησης

(3.3 Προσεγγίσεις με παράγοντα τοποθέτησης το Δικτυακό Κέντρο)

Για κάθε δίκτυο του κυκλώματος μπορούμε να ορίσουμε ως κέντρο του (net center), το γεωμετρικό κέντρο βάρους όλων των κελιών – μελών του. Για να το υπολογίσουμε, βρίσκουμε το μέσο όρο των συντεταγμένων όλων των κελιών του δικτύου. Δηλαδή το σημείο του κέντρο προκύπτει από τις εξισώσεις:

$$\boldsymbol{Net\ Center}\ (\boldsymbol{x}) = \frac{\boldsymbol{1}}{\boldsymbol{N}}\sum_{\boldsymbol{i=1}}^{\boldsymbol{N}} \boldsymbol{x_i} \qquad \boldsymbol{Net\ Center}\ (\boldsymbol{y}) = \frac{\boldsymbol{1}}{\boldsymbol{N}}\sum_{\boldsymbol{i=1}}^{\boldsymbol{N}} \boldsymbol{y_i}$$



(a)

Εικόνα 18 : Κέντρο δικτύου με τα κελιά: A,B,C,D

Η γνώση του net center κάθε δικτύου μπορεί να χρησιμέψει στην παραγωγή ενός κυκλώματος με μικρότερο κόστος δρομολόγησης καθώς αποτελεί κρίσιμο

δείκτη της ιδανικής περιοχής τοποθέτησης. Για αυτό, έχουμε υλοποιήσει μία σειρά παραλλαγών του αλγορίθμου, οι οποίες εκμεταλλεύονται το net center και δίνουν προτεραιότητα σε πιο αποδοτικές κινήσεις με βάση αυτό.

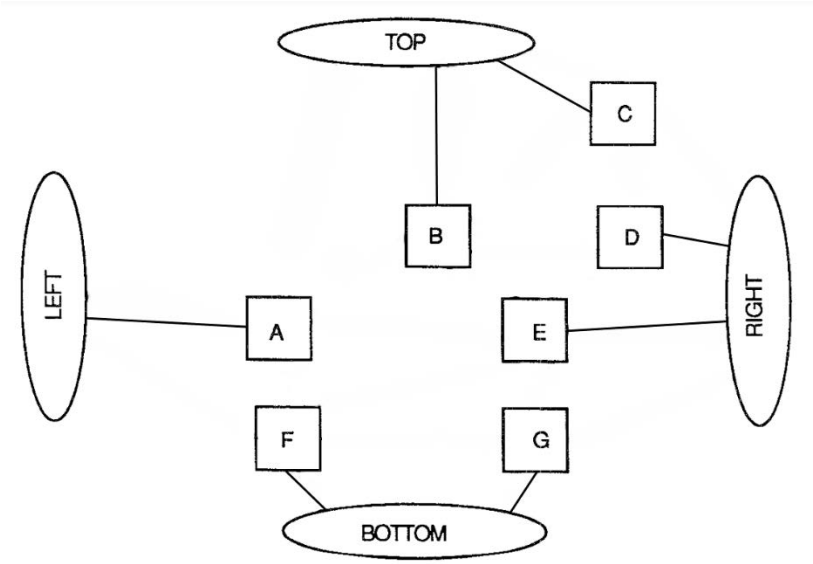
(3.3.1 Net Center Tetris)

Η πρώτη συνάρτηση που βασίζεται στο κέντρο των δικτύων για την νομιμοποίηση ονομάζεται ***tetris_place_cells_net_center***. Πρόκειται για μία ευρετική έκδοση του απλού Tetris, στην οποία η απόσταση του κέντρου κάθε δικτύου από την αρχή των σειρών του κυκλώματος καθορίζει το σημείο τοποθέτησης. Η συνάρτηση δέχεται σαν παραμέτρους δείκτες προς τα αρχεία scl, pl και nets για εύκολη πρόσβαση σε πληροφορίες. Δείκτη προς την δομή Cell και τον δείκτη ***cell_map*** για το ***hashtable***. Όπως και στις εκδοχές με τον συγχρονισμό των δικτύων, δημιουργούμε πάλι τον πίνακα ***cell_placed*** για να παρακολουθούμε ποια κελιά έχουν τοποθετηθεί. Εκτελούμε δυαδική αναζήτηση στο pl αρχείο για να βρούμε τις αρχικές θέσεις των κελιών. Ξεκινάμε τον υπολογισμό του κέντρου κάθε δικτύου. Είναι σημαντικό να διευκρινίσουμε ότι στον υπολογισμό του κέντρου ενός δικτύου χρησιμοποιούμε μόνο τα κανονικά κελιά (***standard cells***) και όχι τα κελιά εισόδου-εξόδου (I/O Pins). Εφόσον τα I/O Pins έχουν σταθερές θέσεις στις άκρες του τσιπ, μετακινούν το κέντρο βάρους εκεί με αποτέλεσμα τα κανονικά κελιά να τοποθετούνται σε λάθος θέσεις. Για αυτό το λόγο, σε ένα βρόχο for για όλα τα δίκτυα του benchmark, δημιουργούμε έναν πίνακα ***net_cell_indices***, ο οποίος θα αποθηκεύσει τα κανονικά κελιά που θα συμμετέχουν στον υπολογισμό. Η δομή Pin μας παρέχει το χρήσιμο πεδίο ***is_pin***. Αυτή η ακέραια μεταβλητή / ***flag***, βοηθάει να γνωρίζουμε αν το κελί είναι εισόδου-εξόδου ή κανονικό. Μέσω του ***hashtable*** βρίσκουμε το ***index*** του κάθε κελιού. Επειδή τα κελιά συνδέονται μεταξύ τους από τα κέντρα τους, τα υπολογίζουμε πριν βρούμε το κέντρο του δικτύου. Το κέντρο του κάθε κελιού είναι το σημείο ***cx***, ***cy***, όπου: ***cx = cells.x + cells.width/2*** και ***cy = cells.y + cells.height/2*** και ***cells.x***, ***cells.y***, οι συντεταγμένες της κάτω αριστερής γωνίας του κελιού. Η συντεταγμένη x του κέντρου του κάθε δικτύου είναι το σύνολο όλων των ***cx*** των κελιών του δικτύου διά το πλήθος τους. Στην προκειμένη περίπτωση χρησιμοποιούμε μόνο την απόσταση του κέντρου του δικτύου από την αρχή της σειράς στον x άξονα. Αυτό γίνεται επειδή οι σειρές έχουν σταθερό ύψος και το

μόνο που μας ενδιαφέρει είναι το net center να είναι κοντά οριζόντια στην αρχή της σειράς. Στο ***sum_x*** έχουμε αποθηκεύσει το άθροισμα των ***cx***. Στην μεταβλητή dx αποθηκεύουμε την απόσταση κάθε net center από την αρχή κάθε σειράς. Τοποθετούμε τα κελιά του ίδιου δικτύου στην σειρά που απέχει λιγότερο από το net center. Κάποιο κελί μπορεί να ανήκει ταυτόχρονα σε παραπάνω από ένα δίκτυο, για αυτό, μετά από κάθε επανάληψη ενημερώνουμε κατάλληλα τον πίνακα ***cell_placed***, για να μην έχουμε διπλή τοποθέτηση. Σε περίπτωση που κάποιο από τα μέλη ενός δικτύου δεν χωράνε στην σειρά, τοποθετούνται αργότερα σε ένα fallback πέρασμα. Αυτό το πέρασμα, ακολουθεί την τακτική του απλού Tetris και αναλαμβάνει τα κελιά που έχουν «ξεμείνει».

(3.3.2 Net Center Tetris με πολλαπλές κατευθύνσεις)

Ακόμη και σε πιο περίπλοκες ευρετικές προσεγγίσεις σαν αυτές που παρουσιάζουμε σε αυτό το κεφάλαιο, είναι σημαντικό να εξετάσουμε διαφορετικές στρατηγικές τοποθέτησης. Έχουμε υλοποιήσει εκδοχές του αλγορίθμου οι οποίες συνδυάζουν τις καινούργιες ευρετικές προσεγγίσεις με φορές τοποθέτησης όπως αυτές που είδαμε στο κεφάλαιο 2. Έτσι, καταφέρνουμε να αξιολογούμε πολλαπλές εκδοχές και μας είναι πιο εύκολο να εντοπίσουμε διατάξεις με καλύτερα χαρακτηριστικά φυσικού σχεδιασμού.



Εικόνα 19 : Στρατηγική τοποθέτησης δικτύων με βάση την μικρότερη απόσταση

(3.3.3 Net Center Tetris με τοποθέτηση από αριστερά και δεξιά)

Η συνάρτηση ***tetris_place_cells_net_left_right*** δανείζεται την τακτική τοποθέτησης της απλής παραλλαγής με υπολογισμό της απόστασης από το αριστερό και το δεξί άκρο του τσιπ. Η διαφορά εδώ βρίσκεται στο γεγονός ότι η απόσταση μετριέται από το κέντρο του κάθε δικτύου και όχι από την «άκρη» του κάθε κελιού. Συγκεκριμένα, εδώ δημιουργούμε δύο μεταβλητές για να γνωρίζουμε την αρχή κάθε σειράς και από αριστερά και από δεξιά. Η ***left_edge*** θα περιέχει το ***subrow_origin*** της επιλεγμένης σειράς για τοποθέτηση, ενώ η ***right_edge***, το ***subrow_origin*** συν το γινόμενο του πλήθους των διαθέσιμων θέσεων με το πλάτος τους. Αποθηκεύουμε την απόσταση και από τις δύο πλευρές στις μεταβλητές ***dist_to_left*** και ***dist_to_right***. Τις συγκρίνουμε και επιλέγουμε την πλευρά με αρκετό διαθέσιμο χώρο και μικρότερη απόσταση.

(3.3.4 Net Center Tetris με τοποθέτηση από πάνω και κάτω)

Με παρόμοιο τρόπο έχουμε υλοποιήσει και την εκδοχή με φορά τοποθέτησης από την πάνω και την κάτω πλευρά του τσιπ ταυτόχρονα. Εδώ οι αποστάσεις υπολογίζονται από την πάνω και κάτω άκρη του τσιπ, δηλαδή: top_edge = coordinate (της πιο «ψηλής» σειράς) και bottom_edge = coordinate + height (της πιο «χαμηλής» σειράς). Η Ευκλείδεια απόσταση χρησιμοποιείται δύο φορές σε αυτές τις συναρτήσεις. Πρώτα

(3.3.5 Net Center Tetris με φορά «ρολογιού»)

Η παρούσα εκδοχή χρησιμοποιεί κάθε πλευρά του τσιπ. Ωστόσο, η τοποθέτηση γίνεται με προτεραιότητα φοράς ρολογιού: ΑΡΙΣΤΕΡΑ – ΠΑΝΩ – ΔΕΞΙΑ – ΚΑΤΩ. Πρώτα, υπολογίζουμε τα άκρα του τσιπ και τα αποθηκεύουμε στις μεταβλητές ***chip_left, right, top, bottom***. Έτσι καθορίζουμε το περίγραμμα του κυκλώματος. Εξάγουμε τις αρχικές θέσεις των κελιών από το αρχείο PL.

Για να υπολογίσουμε το κέντρο κάθε δικτύου, πρέπει πρώτα να δεσμεύσουμε χώρο στην μνήμη για να κρατήσουμε τα indices για κάθε κελί που ανήκει στο δίκτυο. Όπως εξηγήσαμε νωρίτερα, αυτό γίνεται μέσω της συνάρτησης hash_table_search. Καταγράφουμε κάθε κελί που βρίσκουμε για να μην έχουμε διπλές εγγραφές. Αγνοούμε I/O κελιά. Το κέντρο κάθε δικτύου προκύπτει από το άθροισμα όλων των κέντρων των κελιών του, διά το πλήθος τους. Άρα, ***net_center_x = sum_x / count*** και αντίστοιχα για ***net_center_y***. Οι αποστάσεις των δικτυακών κέντρων από τις πλευρές του τσιπ θα αποθηκευτούν στις μεταβλητές: ***dist_left, right, top, bottom***. Διαλέγουμε την κοντινότερη πλευρά για τοποθέτηση. Σε περίπτωση ισοπαλίας, επιλέγουμε με βάση την σειρά προτεραιότητας. Μετά από κάθε τοποθέτηση, ενημερώνουμε τις ***offset_left, top, right, bottom*** για να ξέρουμε που είναι η επόμενες διαθέσιμες θέσεις.

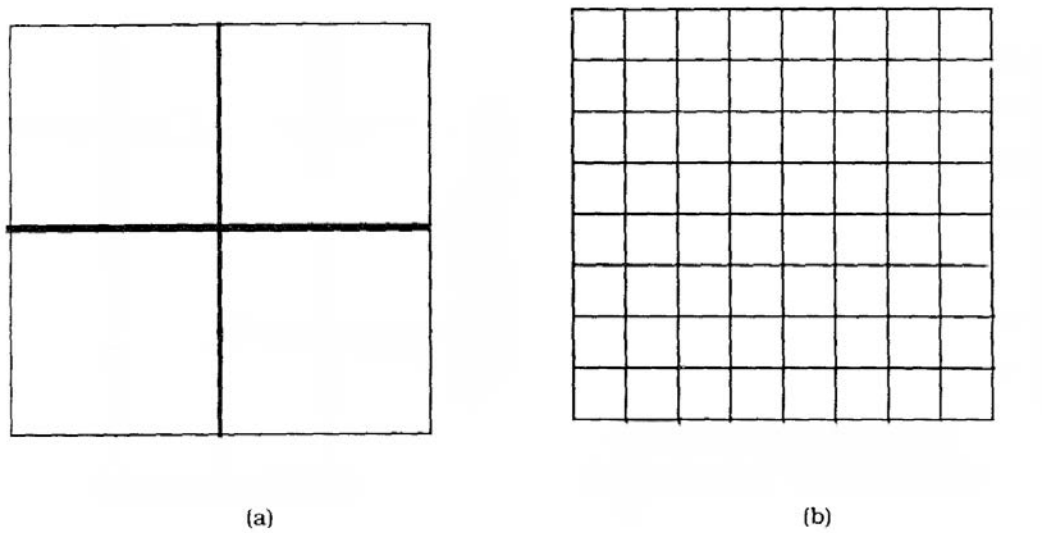
(3.3.6 Net Center Tetris με φορά αντίθετη «ρολογιού»)

Η αντίθετη εκδοχή από την προηγούμενη, όπου ακολουθούμε προτεραιότητα «αντι-ωρολογιακά» περιμετρικά του τσιπ. Στην επιλογή της πλησιέστερης σειράς, επιλέγουμε με σειρά: ΑΡΙΣΤΕΡΑ – ΚΑΤΩ – ΔΕΞΙΑ – ΠΑΝΩ.

(3.4 Grid-Based Προσεγγίσεις)

Ιδιαίτερο ενδιαφέρον εμφανίζουν οι grid-based εκδοχές του αλγορίθμου[4]. Σε αυτές διαιρούμε το layout του τσιπ σε μικρότερα πεδία και εκτελούμε διαδοχική νομιμοποίηση εντός αυτών. Με αυτό τον τρόπο είναι πιο εύκολο να

εντοπίσουμε τοπικές ιδιαιτερότητες του κυκλώματος και να μειώσουμε τον χρόνο εκτέλεσης, καθώς η νομιμοποίηση γίνεται σε πιο ελεγχόμενους χώρους. Ειδικά σε περιπτώσεις που έχουμε να αντιμετωπίσουμε κυκλώματα πολύ μεγάλης κλίμακας, χωρίζοντας το πρόβλημα σε υποπροβλήματα μειώνει την πολυπλοκότητα. Στο γενικότερο επίπεδο του φυσικού σχεδιασμού, τέτοιες τακτικές μας βοηθάνε στην πρόωρη αναγνώριση σημείων συμφόρησης στο κύκλωμα. Έτσι, έχουμε τοποθετήσεις ακρίβειας και μειώνουμε την ανάγκη για ολική επεξεργασία του κυκλώματος αργότερα. Τέτοιες στρατηγικές μπορούν να προσφέρουν ποιότητα αποτελεσμάτων και καλύτερη διαχείριση της επιφάνειας του τσιπ.



Εικόνα 20 : Παραδείγματα διαχωρισμού του διαθέσιμου χώρου σε υποπεδία

Παραπάνω έχουμε δύο βασικά παραδείγματα χωρισμού του τσιπ σε πεδία. Στο παράδειγμα a έχουμε διαχωρισμό σε τέσσερα μέρη (2x2 grids) και στο b σε 64 μέρη (8x8 grids). Η πρώτη περίπτωση διατηρεί μία ελευθερία τοποθέτησης διότι δεν έχουμε πολλές επιμέρους νομιμοποιήσεις να εκτελέσουμε. Η δεύτερη, είναι μία αρκετά περιοριστική τεχνική, η οποία επιτρέπει λεπτομερή τοπικό έλεγχο. Είναι σημαντικό να εξετάσουμε τόσο αυτές τις «ακραίες» τεχνικές, όσο και τις ενδιάμεσές τους για να έχουμε μία πλήρη εικόνα του πως επηρεάζουν την ποιότητα των αποτελεσμάτων.

(3.4.1 Grid-Based Tetris 2x2)

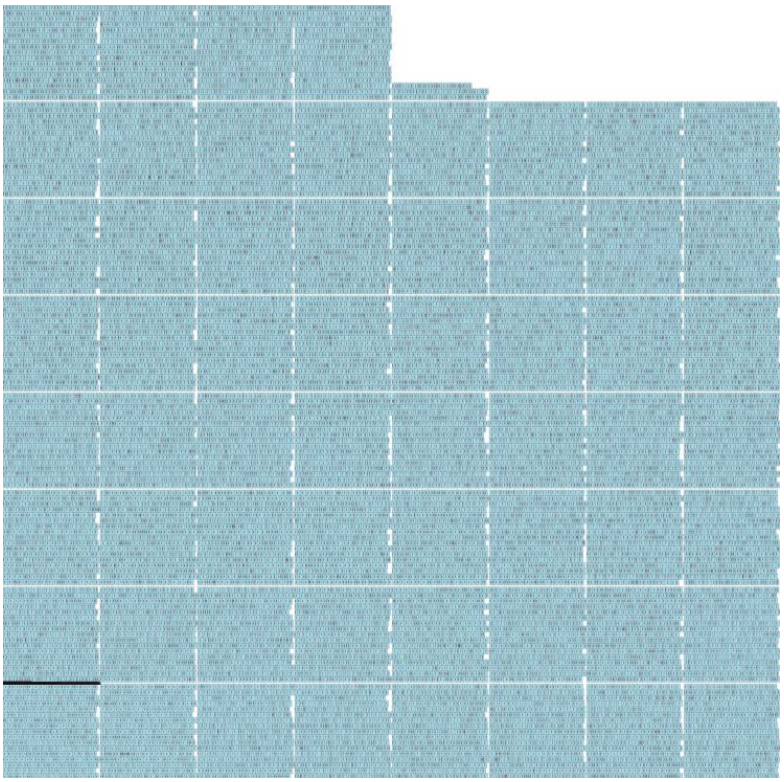
Η συνάρτηση ***tetris_place_cells_in_2x2_with_fallback*** χωρίζει το τσιπ σε τέσσερα τεταρτημόρια και κάνει διαδοχική νομιμοποίηση σε καθένα από αυτά. Πρώτα, εξάγουμε τις αρχικές θέσεις των κελιών στο layout με δυαδική αναζήτηση στο pl αρχείο. Καλούμε τις συναρτήσεις υπολογισμού των διαστάσεων του τσιπ: ***calculate_chip_width*** και ***calculate_chip_height***, αποθηκεύουμε τα αποτελέσματά τους στις μεταβλητές ***chip_width*** και ***height*** αντίστοιχα. Για να ορίσουμε τις διαστάσεις κάθε τεταρτημόριου, διαιρούμε το πλάτος και το ύψος του τσιπ διά 2. Άρα, έχουμε ***region_width*** = ***chip_width***/2 και ***region_height*** = ***chip_height***/2. Για να ξεκινήσουμε την διαδοχική νομιμοποίηση πρέπει πρώτα να αναθέσουμε τα κελιά στα σωστά τεταρτημόριά τους. Δημιουργούμε έναν πίνακα ***region_cells*** που θα περιέχει τέσσερις δείκτες, έναν σε κάθε τεταρτημόριο. Επίσης, τον πίνακα ***region_cell_counts*** με τέσσερις θέσεις, ο οποίος θα αποθηκεύσει πόσα κελιά έχουν ανατεθεί σε κάθε τεταρτημόριο. Η ανάθεση των κελιών στα τεταρτημόρια εξαρτάται από την αρχική τους θέση που πήραμε από το αρχείο pl. Βρίσκουμε το κέντρο των κελιών (πλάτος και μήκος κελιού διά 2) και ελέγχουμε τις συντεταγμένες του. Σε όποιο τεταρτημόριο βρίσκονται, εκεί ανήκει το κελί. Στους πίνακες ***quadrant_current_x***, ***quadrant_current_y***, ***quadrant_row_max_height*** αποθηκεύουμε πληροφορίες για την τρέχουσα κατάσταση τοποθέτησης. Το τρέχον x μέσα στο τεταρτημόριο στην πρώτη, δηλαδή την αρχή της σειράς, το τρέχον y, δηλαδή σε ποιο ύψος του τεταρτημόριου ξεκινάει η σειρά που βρισκόμαστε. Ο τελευταίος πίνακας κρατάει το μέγιστο ύψος του κάθε τεταρτημόριου. Στο πρώτο πέρασμα, το quadrant_current_x ενημερώνεται κάθε φορά που προσθέτουμε ένα κελί με το πλάτος του. Αν η σειρά γεμίσει, προχωράμε στην από πάνω και ενημερώνουμε το quadrant_current_y. Μόλις αυτό γίνει ίσο με το quadrant_row_max_height σταματάμε την νομιμοποίηση σε αυτό το τεταρτημόριο. Το δεύτερο πέρασμα, ή fallback προσπαθεί να τοποθετήσει όσα κελιά δεν τοποθετήθηκαν στο πρώτο, σε όποιο τεταρτημόριο φαίνεται να έχει ακόμη διαθέσιμο χώρο. Αν, δεν υπάρχει άλλος χώρος προειδοποιείται ο χρήστης.

(3.4.2 Grid-Based Tetris 4x4)

Συνεχίζουμε με ένα πιο λεπτομερές μοίρασμα του layout. Τώρα, διαιρούμε το πλάτος και το μήκος του τσιπ διά 4 και καταλήγουμε να έχουμε 16 περιοχές. Το πλάτος και το μήκος κάθε πεδίου (region_width και region_height) ισούται με το πλάτος και το μήκος του τσιπ διά 4. Η ανάθεση των κελιών στα σωστά πεδία γίνεται πάλι με βάση την αρχική τους θέση. Σε ένα βρόχο for, δηλώνουμε τις συντεταγμένες του κέντρου κάθε κελιού cx και cy. Η στήλη στην οποία βρίσκεται ένα κελί είναι το cx του κελιού διά το πλάτος του πεδίου. Αντίστοιχα, η γραμμή είναι το cy διά το ύψος του πεδίου. Για να ορίσουμε τις διαστάσεις κάθε πεδίου χρησιμοποιούμε ένα βρόχο for με όριο το num_regions, δηλαδή 16. Χρησιμοποιούμε τις μεταβλητές ***col*** και ***row*** για να ξέρουμε που ξεκινάει και που καταλήγει το κάθε πεδίο. Κάθε πεδίο έχει το δικό του flag r. Για να το καταλάβουμε καλύτερα ας δούμε ένα παράδειγμα. Έστω ότι το τσιπ με το οποίο δουλεύουμε έχει διαστάσεις 6000x6000. Άρα ***region_width*** = 1500 και ***region_height*** = 1500. Έχουμε ***col*** = r % ***regions_per_row*** και ***row*** = r / ***regions_per_row***. Για το πεδίο r = 8 θα ισχύει: col = 8 % 4 = 0 και row = 8 / 4 = 2. Άρα, ***region_x_start***[8] = 0*6000 = 0, ***region_y_start*** = 2*1500 = 3000. Επιπλέον, ***region_x_end***[8] = 0 + 1500 = 1500, ***region_y_end***[5] = 3000 + 1500 = 4500. Τελικά, το πεδίο 8 καλύπτει το παραλληλόγραμμο από (0, 3000) έως (1500, 4500). Η νομιμοποίηση γίνεται όπως και στην προηγούμενη έκδοση. Έχουμε δύο βρόχους for, ο εξωτερικός για το πλήθος των πεδίων και ο εσωτερικός για το πλήθος των κελιών σε κάθε πεδίο.

(3.4.3 Grid-Based Tetris 8x8)

Ο διαμοιρασμός σε 8x8 grids, αποτελεί μία μέση λύση. Πολύ χρήσιμος όταν έχουμε σχέδια μεσαίου προς μεγάλου μεγέθους, στα οποία θέλουμε να εκμεταλλευτούμε τις τοπικές ιδιαιτερότητές τους. Επιπλέον, αποτελεί καλή επιλογή για πειραματικές συγκρίσεις, καθώς δεν επιβαρύνει σημαντικά την απόδοση του αλγόριθμου.



Εικόνα 21 : Τελική τοποθέτηση από τον Grid-Based Tetris 8x8

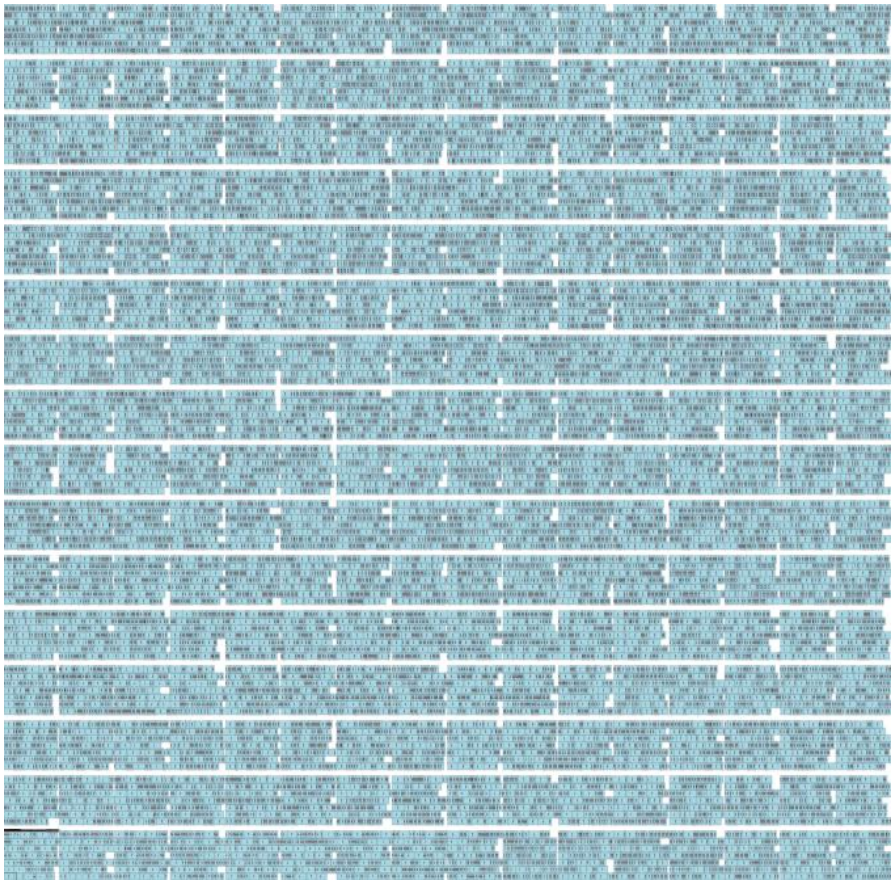
Σε τέτοιες διαμερίσεις, αυξάνονται κατά πολύ οι πιθανότητες ένα κελί να βρίσκεται αρχικά πάνω σε μία γραμμή ή στήλη του σχεδίου. Σε αυτές τις περιπτώσεις, τα κελιά ανήκουν στο πεδίο δεξιά ή κάτω από την γραμμή / στήλη. Στην 8x8 εκδοχή θα χωρίσουμε το πλάτος και μήκος του τσιπ σε 8 πεδία το καθένα. Καταλήγουμε να έχουμε χωρίσει το τσιπ σε 64 πεδία συνολικά, όπως φαίνεται στον παρακάτω πίνακα:

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
[16]	[17]	[18]	[19]	[20]	[21]	[22]	[23]
[24]	[25]	[26]	[27]	[28]	[29]	[30]	[31]
[32]	[33]	[34]	[35]	[36]	[37]	[38]	[39]
[40]	[41]	[42]	[43]	[44]	[45]	[46]	[47]
[48]	[49]	[50]	[51]	[52]	[53]	[54]	[55]
[56]	[57]	[58]	[59]	[60]	[61]	[62]	[63]

Αν υποθέσουμε πως το τσιπ έχει διαστάσεις 8000x8000, το πεδίο [56] αρχίζει στο (0, 0) και τελειώνει στο (1000, 1000). Έστω ότι η αρχική θέση ενός κελιού είναι η (500, 1000), δηλαδή η μέση της σειράς που χωρίζει τα πεδία: [48] και [56]. Το κελί θεωρούμε ότι ανήκει στο πεδίο [56]. Άρα το προσμετράμε στο ***region_cell_counts*** του [56]. Επιπλέον, όσο πιο λεπτομερές το διαμέρισμα του τσιπ, τόσο πιο αναγκαίο γίνεται το fallback pass. Δηλαδή έναν τρόπος να τοποθετούμε τα κελιά που δεν συμμετείχαν στην πρώτη τοποθέτηση. Ελέγχουμε αρχικά αν έχει τοποθετηθεί στο πρώτο πέρασμα. Αν όχι, δημιουργούμε ένα flag placed_in_fallback, για να ξέρουμε αν τοποθετήθηκε μέχρι το τέλος του δεύτερου περάσματος. Ψάχνουμε αν έχει μείνει χώρος σε κάποιο από τα 64 πεδία. Αν ναι, το τοποθετούμε εκεί και ενημερώνουμε το ύψος και το πλάτος της σειράς.

(3.4.4 Grid-Based Tetris 16x16)

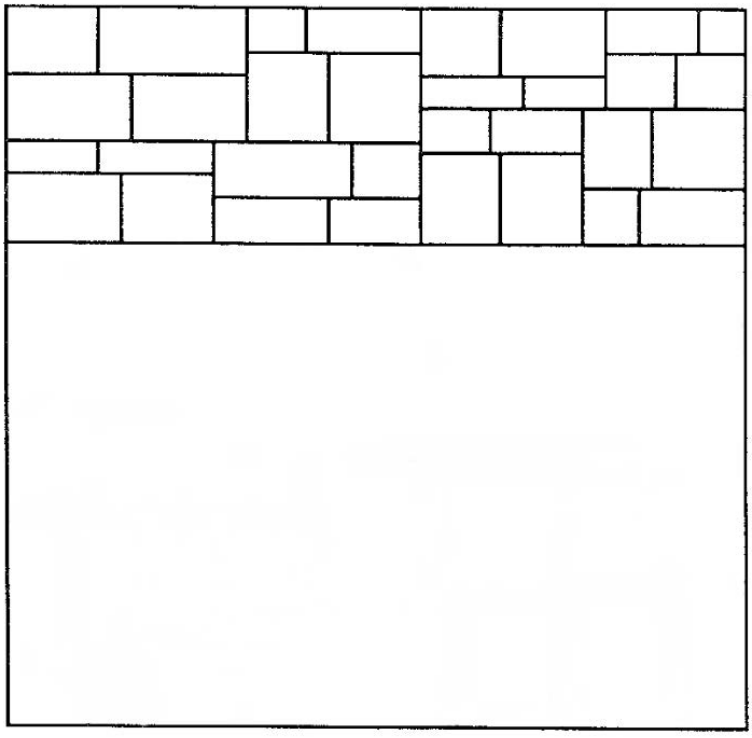
Η τελευταία από τις Grid-Based προσεγγίσεις εκτελεί διαμοιρασμό του τσιπ σε 256 πεδία. Στρατηγικές νομιμοποίησης σαν αυτή, αυξάνουν την γεωμετρική αυστηρότητα και συχνά επιφέρουν μεγάλο αριθμό αποτυχημένων τοποθετήσεων. Για αυτό το λόγο κρίνεται απαραίτητο ξανά, ένα τελικό στάδιο επανατοποθέτησης. Όταν χρησιμοποιούμε τέτοιες τεχνικές, ενδέχεται να οδηγηθούμε σε περιορισμούς και συμφόρηση σε σημεία του τσιπ. Παρ' όλα αυτά, μπορούν να μειώσουν το συνολικό displacement όταν έχουμε αρχικά καλές κατανομές.



Στην παραπάνω εικόνα έχουμε ένα παράδειγμα της προσέγγισης. Το test case που χρησιμοποιήσαμε ήταν το ibm06, το οποίο περιέχει 33000 κελιά. Με διαστάσεις 2016 x 2037, είχαμε συνολικά 1953 κελιά τα οποία δεν μπορούσαν να τοποθετηθούν, ακόμη και μετά την επανατοποθέτηση. Δηλαδή 6% των κελιών προς τοποθέτηση δεν χρησιμοποιήθηκαν.

(3.5 Περιοριστικές Προσεγγίσεις)

Η τελευταία κατηγορία ευρετικών προσεγγίσεων που μελετήσαμε, ήταν οι περιοριστικές. Βασικό χαρακτηριστικό τους είναι ότι ο χρήστης, εκ των προτέρων καθορίζει το εύρος των σειρών του κυκλώματος στις οποίες μπορεί να χρησιμοποιηθεί ο αλγόριθμος Tetris. Ουσιαστικά, εμπλουτίζουμε τον αλγόριθμο με την δυνατότητα φιλτραρίσματος, η οποία του επιτρέπει να αγνοεί όλες τις σειρές εκτός αυτών που του έχουμε επιτρέψει. Αυτές οι τεχνικές μπορούν να είναι χρήσιμες σε περιπτώσεις που υπάρχουν δεσμευμένες περιοχές του κυκλώματος, ή τεχνολογικοί περιορισμοί μας εμποδίζουν να εκμεταλλευτούμε ολόκληρο το layout. Επίσης, τέτοιες περιοριστικές στρατηγικές μας βοηθούν να αξιολογήσουμε την απόδοση του αλγορίθμου όταν λειτουργεί υπό συνθήκες μειωμένου χώρου.



Εικόνα 22 : Παράδειγμα περιορισμού της τοποθέτησης στο πάνω μέρος του τσιπ

(3.5.1 Περιοριστικός Tetris με input αριθμού)

Η πρώτη εκδοχή από τις περιοριστικές δέχεται σαν παράμετρο την ακέραια μεταβλητή ***row_range***. Αυτή είναι μία hard coded παράμετρος η οποία εκπροσωπεί το περιθώριο των σειρών πάνω και κάτω από την «βασική σειρά» που θα επεκταθεί ο αλγόριθμος. Η «βασική σειρά» ή base row, είναι η σειρά που επιλέξει ο χρήστης και λειτουργεί σαν αφετηρία για την νομιμοποίηση. Είναι πάντα η μεσαία σειρά του παραλληλογράμμου στο οποίο θα εκτελέσουμε τον αλγόριθμο. Ας το δούμε με μεγαλύτερη λεπτομέρεια. Η συνάρτηση ***tetris_place_cells_with_row_restriction_user*** πρώτα ανακτά τις αρχικές θέσεις των κελιών στον χώρο. Ζητάει από τον χρήστη να εισάγει το base row. Ελέγχουμε την εγκυρότητα του input. Αν δεν είναι ακέραιος αριθμός εκτυπώνουμε κατάλληλο σφάλμα. Αν ο χρήστης εισάγει αριθμό σειράς μεγαλύτερο ή μικρότερο από αυτές που έχουμε διαθέσιμες, χρησιμοποιούμε την πρώτη σειρά (σειρά 0) σαν base row. Στην συνέχεια υπολογίζουμε τα περιθώρια πάνω και κάτω από το base row. Δηλαδή σε πόσες σειρές πάνω και κάτω από αυτή θα τρέξει ο αλγόριθμος. Η μεταβλητή ***min_allowed*** εκπροσωπεί τις κάτω σειρές, σε αυτή αποθηκεύουμε το αποτέλεσμα του ***base_row – row_range***. Αν αυτό είναι αρνητικό, σημαίνει πως ο χρήστης επέλεξε ένα base row αρκετά κοντά στο κάτω μέρος του τσιπ. Τότε θεωρούμε πως ***min_allowed*** = 0. Η ***max_allowed*** είναι το αποτέλεσμα του ***base_row + row_range***. Ενημερώνουμε τον χρήστη τα όρια στα οποία θα εκτελέσουμε τον αλγόριθμο. Η τοποθέτηση γίνεται με δύο βρόχους for, ο εξωτερικός για όλα τα κελιά του test case και ο εσωτερικός για τις επιτρεπτές σειρές (από ***min_allowed*** μέχρι ***max_allowed***). Για κάθε κελί που τοποθετούμε ενημερώνουμε την αρχή της σειράς και το flag placed. Αν δεν υπάρχει χώρος σε καμία σειρά για τοποθέτηση αυξάνουμε τον μετρητή unplaced. Στο τέλος εμφανίζουμε μήνυμα με το σύνολο των κελιών τα οποία δεν τοποθετήθηκαν. Σε αυτές τις τεχνικές δεν χρησιμοποιούμε fallback pass, διότι καθιστά τον αρχικό περιορισμό που θέσαμε, ανούσιο.

(3.5.2 Περιοριστικός Tetris με input ποσοστού)

Μία άλλη εκδοχή της της περιοριστικής τακτικής, είναι η εισαγωγή του base row σαν ποσοστό. Τώρα δηλαδή, εκτός από το base row, ζητάμε από τον χρήστη να εισάγει και το ποσοστό των σειρών που επιθυμεί να χρησιμοποιήσουμε. Έστω ότι έχουμε ένα κύκλωμα με 96 σειρές. Ο χρήστης δίνει σαν base row την σειρά 40 και ποσοστό 30%. Το σύνολο των σειρών πάνω και κάτω από την βασική που μπορούμε να χρησιμοποιήσουμε ισούται με: ποσοστό/100*σύνολο σειρών του τσιπ (+0.5 για στρογγυλοποίηση). Άρα, 30/100*96+0.5 = 29.3 (29). Το μισό του 29 (14 και όχι 14.5), θα το προσθαφαιρήσουμε από το base row για να βρούμε πως το περιθώριο που έχουμε είναι το [26,54].

ΚΕΦΑΛΑΙΟ 4

Πειραματικά αποτελέσματα – Συμπεράσματα και μελλοντικές Προεκτάσεις

Στο παρόν κεφάλαιο θα παρουσιάσουμε τα αποτελέσματα της εφαρμογής των διάφορων παραλλαγών του αλγορίθμου νομιμοποίησης Tetris. Κάθε εκδοχή του, ευρετική ή απλή, θα αξιολογηθεί με βάση κάποια καθορισμένα κριτήρια. Συγκεκριμένα, την συνολική μετακίνηση (displacement), τον χρόνο εκτέλεσης (runtime) καθώς και την μετρική THPWL. Έτσι θα έχουμε μία ξεκάθαρη εικόνα της επίδρασης των τροποποιήσεων στην ποιότητα της τελικής τοποθέτησης. Όλα τα αποτελέσματα θα παρουσιαστούν συγκριτικά, τόσο μεταξύ των παραλλαγών όσο και απέναντι στον κλασικό αλγόριθμο. Με την ανάλυση των δεδομένων, επιχειρούμε να εξάγουμε συμπεράσματα και ποιοτικές παρατηρήσεις σχετικά με την αποτελεσματικότητα κάθε εκδοχής σε διαφορετικά benchmarks. Στο τέλος του κεφαλαίου θα παρουσιάσουμε προτάσεις για μελλοντικές βελτιώσεις και επεκτάσεις.

(4.1 Περιβάλλον Εκτέλεσης)

Όλες οι παραλλαγές του αλγορίθμου που έχουμε παρουσιάσει μέχρι τώρα, όπως επίσης και οι αναλυτές υλοποιήθηκαν και εκτελέστηκαν στον ίδιο υπολογιστικό εξοπλισμό για να διασφαλίσουμε την συγκρισιμότητα των αποτελεσμάτων. Συγκεκριμένα, οι πειραματικές δοκιμές πραγματοποιήθηκαν σε φορητό υπολογιστή με τα εξής τεχνικά χαρακτηριστικά:

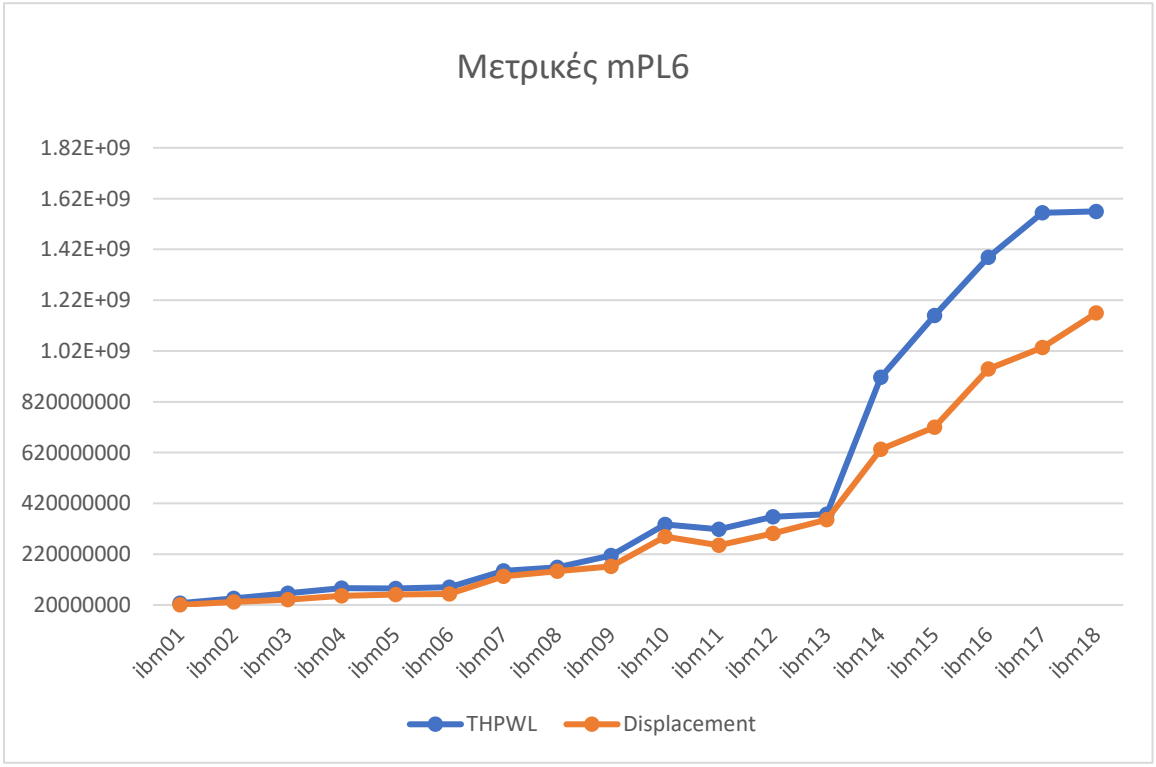
- **Επεξεργαστής (CPU)**: Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz
- **Μνήμη (RAM)**: 32 GB DDR4 @ 3200MHz
- **Μονάδα Αποθήκευσης**: NVMe SSD *BC501 NVMe SK hynix 250 GB*
- **Λειτουργικό Σύστημα**: Windows 11 24H2 (64-bit)
- **Μεταγλωττιστής**: GCC 13.2.0

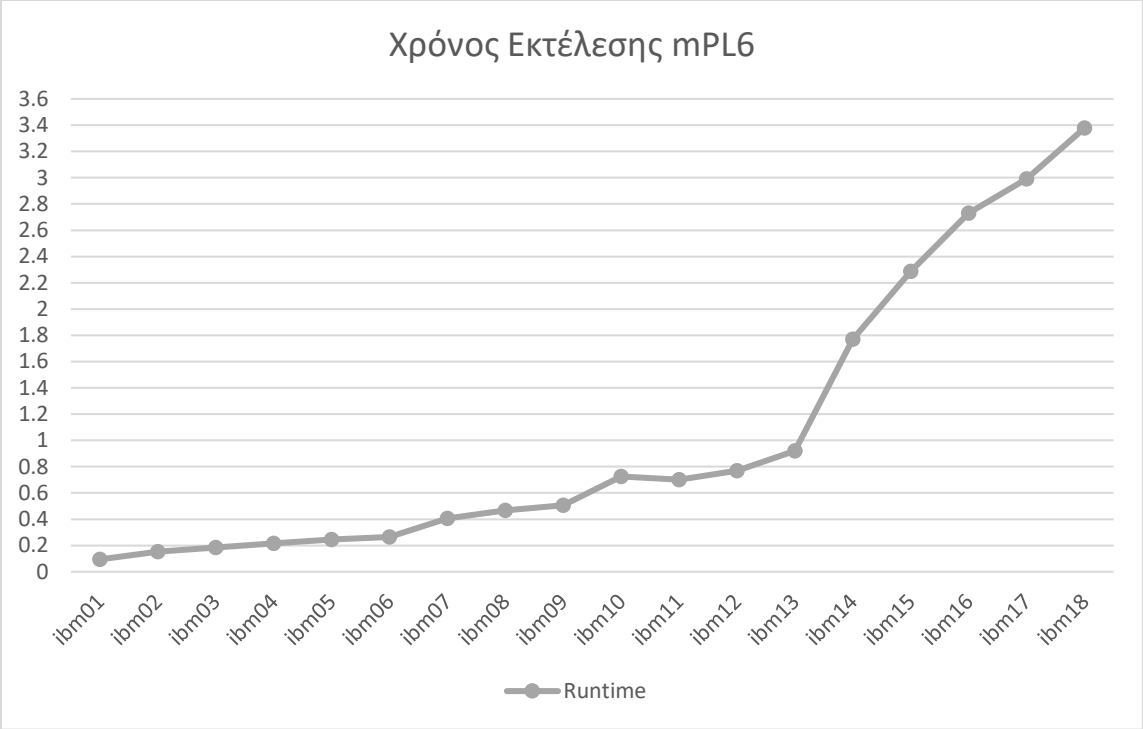
Η μέτρηση του χρόνου εκτέλεσης (runtime), έγινε μέσω της συνάρτησης ***clock_gettime*** με το flag ***CLOCK_MONOTONIC***, ώστε να εξασφαλίσουμε μονοτονία και ακρίβεια. Ο υπολογισμός του runtime συμβαίνει ακριβώς μετά τον υπολογισμό του συνολικού displacement και του THPWL. Ο χρήστης έχει την επιλογή να χρησιμοποιήσει κάποια από τις εκδοχές του αλγορίθμου είτε σε όλα τα IBM benchmarks, είτε σε κάποιο συγκεκριμένο. Και στις δύο περιπτώσεις τα αποτελέσματα αποθηκεύονται σε αρχείο ***results.csv***.

(4.2 Ανάλυση Αποτελεσμάτων Απλών Παραλλαγών – Γενικές Παρατηρήσεις)

(4.2.1 Αποτελέσματα κλασικού Tetris)

Πρώτα θα παρουσιάσουμε τα αποτελέσματα του κλασικού αλγορίθμου, θα προχωρήσουμε σε αυτά των απλών παραλλαγών και τέλος, των ευρετικών . Θα εμβαθύνουμε στις ιδιαιτερότητές τους και τον τρόπο που επηρεάζονται από παράγοντες όπως ο global placer αλγόριθμος που χρησιμοποιήθηκε.





Αναμενόμενα, παρατηρούμε σημαντική αύξηση του χρόνου εκτέλεσης μετά το benchmark ibm13. Από 0.92 δευτερόλεπτα για το ibm13, σε 1.77 για το ibm14, σχεδόν διπλασιασμός.

Παρακάτω έχουμε τις τρεις δεκαοχτάδες αποτελεσμάτων του κλασικού Tetris.

Global Placer: Mpl6

Benchmark	THPWL	Displacement	Runtime
ibm01	26862670	20229126	0.094421
ibm02	45811518	31431617	0.151885
ibm03	65749776	41020428	0.184664
ibm04	85792604	55437369	0.216899
ibm05	84949889	60423030	0.2452
ibm06	90180001	63979569	0.264763
ibm07	154203340	132252524	0.405714
ibm08	167650657	153082893	0.468066
ibm09	214955043	171828450	0.505839

ibm10	336533843	289241689	0.725865
ibm11	317351600	254769357	0.701013
ibm12	366561163	301181752	0.768393
ibm13	377708946	355733658	0.92076
ibm14	915967042	633332896	1.77102
ibm15	1159533808	720201431	2.285823
ibm16	1388729163	949168503	2.731621
ibm17	1563553887	1034019776	2.991178
ibm18	1569650817	1169347545	3.378368

Global Placer: GORDIAN

Benchmark	THPWL	Displacement	Runtime
ibm01	27144347	19721334	0.101813
ibm02	46265454	30975998	0.161078
ibm03	65468425	41545958	0.194139
ibm04	86895141	56037303	0.232868
ibm05	85753789	59316849	0.260888
ibm06	90572957	62980047	0.285369
ibm07	154775710	133996782	0.445027
ibm08	168565619	153158844	0.514252
ibm09	214526744	170529760	0.537339
ibm10	335829354	286310322	0.781844
ibm11	317467514	258835770	0.743342
ibm12	366033091	303195426	0.808336
ibm13	378080270	354266855	0.969731
ibm14	918109472	639377016	1.855408
ibm15	1162303650	722040026	2.35852
ibm16	1384923626	938357763	2.851834
ibm17	1565625388	1034483013	3.150813
ibm18	1570594296	1170615574	3.42941

Global Placer: ePLACE

Benchmark	THPWL	Displacement	Runtime
ibm01	26367276	19985698	0.093713
ibm02	43899325	30894853	0.167913
ibm03	64373428	40569410	0.204924

Benchmark	THPWL	Displacement	Runtime
ibm04	59483441	56863710	0.208147
ibm05	85334904	60736094	0.259754
ibm06	90081979	62268673	0.270108
ibm07	153956425	130723230	0.413317
ibm08	167705881	149532946	0.468945
ibm09	214424659	168771177	0.505155
ibm10	333867989	279663805	0.725916
ibm11	315742716	254181918	0.699348
ibm12	365336730	298849118	0.769536
ibm13	377756728	353390754	0.920287
ibm14	915046704	627774751	1.763818
ibm15	1158121507	707176206	2.269953
ibm16	1379520729	932209643	2.732247
ibm17	1561056444	1018748021	2.986614
ibm18	1563841965	1153368867	3.317628

Από τις τρεις ομάδες αποτελεσμάτων εξάγουμε τα εξής συμπεράσματα. Ο eplace έχει το μικρότερο μέσο όρο στο THPWL με 493106600, άρα και την καλύτερη απόδοση στην καλωδίωση από τους τρεις. Ο gordian είναι ο χειρότερος με μέσο όρο THPWL 496607500, αλλά οι διαφορές είναι ελάχιστες. Όσον αφορά την συνολική μετακίνηση ο eplace είναι πάλι πρώτος με μέσο όρο displacement 352539400. Ο mpl6 και ο gordian είναι πολύ κοντά με μέσους όρους 357593400 και 357541400 αντίστοιχα. Στον χρόνο εκτέλεσης παρατηρούμε ελάχιστες διαφορές, με τον eplace να προηγείται πάλι με μέσο χρόνο εκτέλεσης 1.043 δευτερόλεπτα. Ακολουθεί ο mpl6 με 1.045 και τελευταίος ο gordian με 1.093.

Όταν μιλάμε για τον κλασικό αλγόριθμο νομιμοποίησης Tetris, οι λεπτές διαφορές ανάμεσα σε διαφορετικές αρχικές συνθήκες είναι απολύτως αναμενόμενες και αποτελούν χαρακτηριστικό γνώρισμα των ντετερμινιστικών ευρετικών αλγορίθμων. Οι ελάχιστα διαφοροποιημένες θέσεις που δέχεται σαν είσοδο κάθε φορά οδηγούν σε διαφορετικές αποφάσεις τοποθέτησης και σε ελαφρώς αλλαγμένα αποτελέσματα. Οι διαφορές παραμένουν της τάξη του 1-3% και επιβεβαιώνουν την σταθερότητα και προβλεψιμότητα του αλγορίθμου.

(4.2.2 Αποτελέσματα εκδοχής «Bottom» Tetris)

Global Placer: Mpl6

Benchmark	THPWL	Displacement	Runtime
ibm01	24466473	21414856	0.108567
ibm02	38213042	39356644	0.173139
ibm03	59624418	35493034	0.245701
ibm04	77061997	48157652	0.25141
ibm05	76336696	53083971	0.283134
ibm06	78773442	68301282	0.303852
ibm07	156198590	120561279	0.478572
ibm08	168964988	137997969	0.529345
ibm09	215951736	155478817	0.569353
ibm10	319389855	274475393	0.841427
ibm11	295933329	242347872	0.857682
ibm12	342281382	287298761	1.00032
ibm13	395749818	335270763	1.11035
ibm14	781633067	797735800	2.550766
ibm15	945605508	914321926	3.648217
ibm16	1133387661	1202824598	4.780173
ibm17	1287051606	1311176764	4.439328
ibm18	1430129119	1018482491	5.856212

Global Placer: GORDIAN

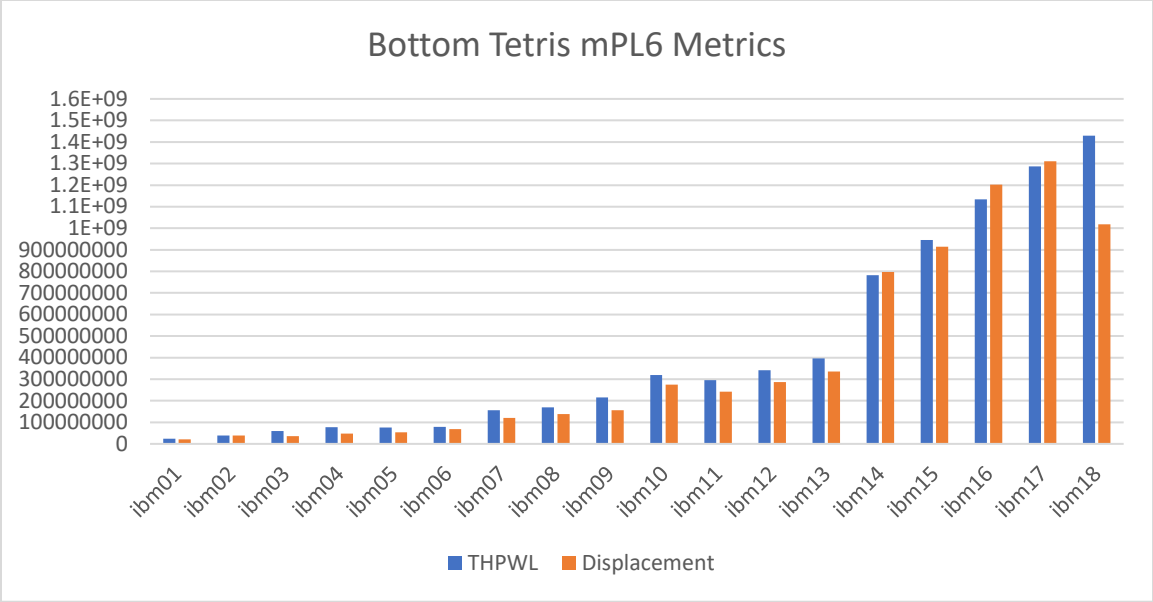
Benchmark	THPWL	Displacement	Runtime
ibm01	24512753	20629999	0.114614
ibm02	38283847	38248161	0.177849
ibm03	59919180	36082048	0.270263
ibm04	79002305	48678456	0.270506
ibm05	77519491	51289807	0.297125
ibm06	76877838	69921224	0.33223
ibm07	156720031	121595281	0.497674
ibm08	170188762	139161348	0.560669
ibm09	217210474	156258087	0.616477
ibm10	317727304	268008661	0.888435

Benchmark	THPWL	Displacement	Runtime
ibm11	297448470	244843127	0.858392
ibm12	345497328	286845310	0.934382
ibm13	395971132	333245483	1.142417
ibm14	780802722	805544268	2.223655
ibm15	948081006	916248632	2.732206
ibm16	1131756206	1194602450	3.354947
ibm17	1286902382	1317728276	3.700528
ibm18	1431333449	1020225816	3.959748

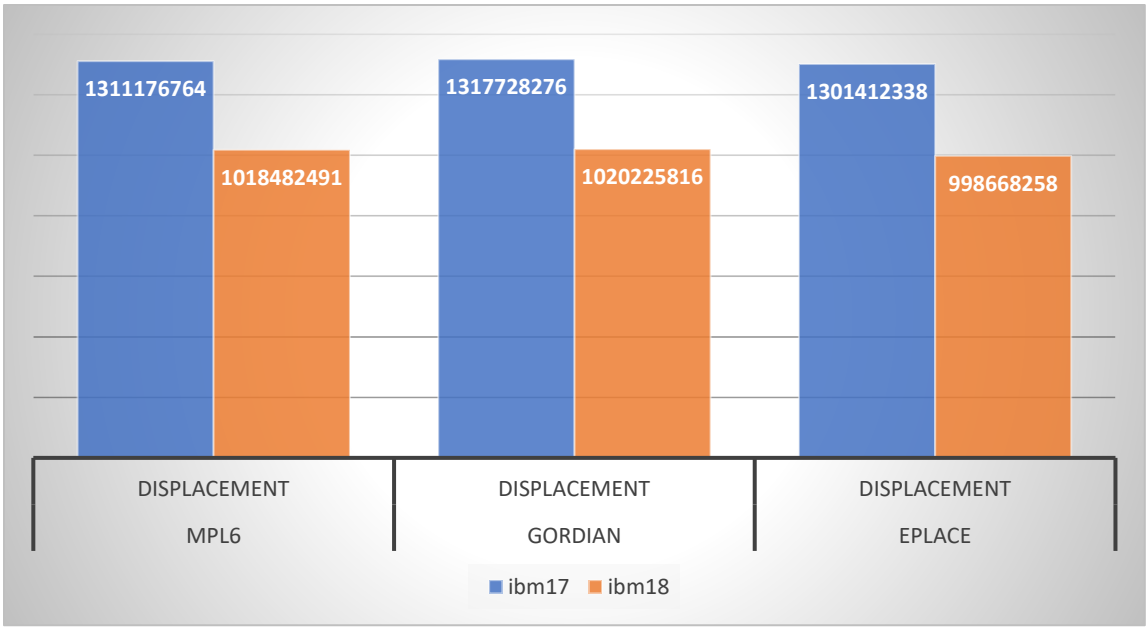
Global Placer: ePlace

Benchmark	THPWL	Displacement	Runtime
ibm01	24073354	20818881	0.108717
ibm02	37493828	38070107	0.168846
ibm03	58224700	34859693	0.211685
ibm04	59784049	54828029	0.244629
ibm05	75798453	52564733	0.282227
ibm06	76611535	69678072	0.319674
ibm07	155448097	119004125	0.516412
ibm08	169359143	136527895	0.546893
ibm09	216868114	154179740	0.602295
ibm10	316336833	268055793	0.862427
ibm11	297502104	239646302	0.809589
ibm12	343041081	281848191	1.09293
ibm13	395930001	331983186	1.262249
ibm14	775092884	797386952	2.243695
ibm15	945656845	897840999	2.780346
ibm16	1132808746	1191844695	3.408359
ibm17	1288561853	1301412338	3.856737
ibm18	1423038503	998668258	4.51666

Παραπάνω βλέπουμε τα αποτελέσματα της «Bottom» εκδοχής του αλγορίθμου. Εμφανίζει ομαλή εξέλιξη του THPWL με αξιοπρεπείς τιμές, καλύτερες από αυτές του «δεξιόστροφου» Tetris και της «top» έκδοσης.



Η συνολική μετατόπιση ξεκινάει σε λογικά επίπεδα αλλά υποβαθμίζεται σημαντικά σε μεγαλύτερα benchmarks, όπως το ibm15. Ο χρόνος εκτέλεσης κλιμακώνεται μέτρια, με μέση τιμή ~5.8 δευτερόλεπτα. Παρατηρούμε μία δραστική μείωση του συνολικού displacement στο τελευταίο benchmark. Αυτή η μείωση, όπως βλέπουμε και παρακάτω, είναι εμφανής στα αποτελέσματα όλων των placers.



Η μείωση κυμαίνεται περίπου:

- 23% για Mpl6,
- 22,6% για GORDIAN
- 23,3% για ePlace

Ενώ οι μείωση του displacement εμφανίζεται σε πολλές εκδοχές του αλγορίθμου στα τελευταία δύο benchmarks, γίνεται ιδιαίτερα έντονη στην bottom εκδοχή του. Πρόκειται για ένα συνεπές μοτίβο, που υποδεικνύει πως κάτι στην φύση του ibm18 διευκολύνει αρκετά το legalization και ταιριάζει εξαιρετικά καλά στην στρατηγική τοποθέτησης του bottom Tetris. Είναι πολύ πιθανό πως η μορφολογία του ibm18 επιτρέπει σε αυτή την εκδοχή του αλγορίθμου να εκμεταλλευτεί πλήρως τον χώρο στις κατώτερες σειρές του τσιπ και να τοποθετήσει πολλά κελιά κοντά στις αρχικές τους θέσεις.

THPWL	Runtime mpl6	Runtime gordian	Runtime ePlace
113M	4.78s	3.35s	3.40s
128M	4.43s	3.70s	3.85s
143M	5.85s	3.95s	4.51s

Από την ανάλυση των δεδομένων του bottom tetris, προκύπτει μία ενδιαφέρουσα παρατήρηση. Όπως βλέπουμε παραπάνω, ο μέσος χρόνος εκτέλεσης στα τελευταία 3 benchmarks είναι σημαντικά μεγαλύτερος όταν χρησιμοποιούμε αρχική κατάσταση του Mpl6 placer. Αυτό είναι εμφανές και σε άλλες εκδοχές του αλγορίθμου, ωστόσο εδώ γίνεται αρκετά αισθητό. Σημαντικός παράγοντας είναι ο τρόπος με τον οποίο τοποθετεί τα κελιά ο mpl6, ακολουθώντας μία πιο «ελεύθερη» προσέγγιση, επομένως υπάρχουν περισσότερα overlaps μεταξύ των κελιών που πρέπει να νομιμοποιήσουμε. Άρα ο bottom tetris χρειάζεται περισσότερους ελέγχους σύγκρουσης οι οποίοι οδηγούν σε αύξηση χρόνου.

(4.2.3 Αποτελέσματα εκδοχής «Clockwise» Tetris)

Global Placer: ePlace

Benchmark	THPWL	Displacement	Runtime
ibm01	14264698	6273391	0.176426
ibm02	28127404	13991640	0.188236
ibm03	39893848	20458523	0.228662
ibm04	60119321	54843590	0.226546
ibm05	53761557	31431658	0.277476
ibm06	55115825	29732885	0.290279
ibm07	76057665	35618499	0.414034
ibm08	82399173	41269227	0.47528
ibm09	105909188	46417265	0.508669
ibm10	164768420	72534825	0.71862
ibm11	153315514	66622290	0.697391
ibm12	180263044	80718542	0.724757
ibm13	146308564	55725401	0.847716
ibm14	583766760	308798515	1.514514
ibm15	701214173	310719848	1.771305
ibm16	843180030	406863594	2.075008
ibm17	963317353	447023239	2.224609
ibm18	940775895	552951018	2.381579

Global Placer: GORDIAN

Benchmark	THPWL	Displacement	Runtime
ibm01	14798087	6246802	0.194228
ibm02	28274698	13909677	0.222459
ibm03	40774773	20773104	0.272958
ibm04	53885042	27956851	0.328047
ibm05	53467718	29123607	0.354529
ibm06	55403068	29578376	0.376272
ibm07	77168764	35638123	0.577343
ibm08	84555488	41449708	0.649323
ibm09	106675516	46101938	0.72622
ibm10	168700804	73684452	1.039844

Benchmark	THPWL	Displacement	Runtime
ibm11	157269057	66291034	1.003905
ibm12	183128198	78338198	1.093927
ibm13	153998561	56985862	1.25934
ibm14	588216478	317694462	2.424326
ibm15	698611171	311829777	2.796591
ibm16	838217298	407325427	3.385263
ibm17	959385606	449136457	3.71741
ibm18	953994280	565040181	3.843885

Global Placer: Mpl6

Benchmark	THPWL	Displacement	Runtime
ibm01	14335763	6337700	0.18836
ibm02	27888786	13935552	0.220025
ibm03	39832460	20089924	0.267122
ibm04	54154873	28055152	0.317366
ibm05	52957003	30939181	0.350338
ibm06	55472123	30325225	0.379891
ibm07	76291018	35993461	0.586968
ibm08	81708223	41218555	0.652386
ibm09	104300415	45552786	0.676814
ibm10	161756620	74096845	1.025792
ibm11	151499075	65797690	0.967125
ibm12	176894938	77846646	1.053573
ibm13	144946543	55812888	1.202055
ibm14	582430340	314707525	2.304429
ibm15	690118316	310215001	2.706151
ibm16	830703826	405454670	3.421643
ibm17	945709474	446635843	3.947826
ibm18	953003240	568629301	4.210423

Η clockwise εκδοχή του αλγορίθμου εμφανίζει συστηματικά χαμηλότερες displacement τιμές από την bottom. Επίσης ο χρόνος εκτέλεσης αυξάνεται λογικά με τα μεγέθη των benchmarks, χωρίς να εμφανίζει το «runtime spike» που υπάρχει στα τελικά benchmarks του bottom (ειδικά στην mpl6 έκδοση).

Legalizer	Best THPWL	Best Displacement	Best Runtime
ePlace	✔	✔	✔
Gordian	✘	✘	✘
mpl6	✘ (σχεδόν)	✘ (κοντά)	✘

Παρατηρούμε πως ο ePlace βγάζει τις πιο «εύκολες» αρχικές συνθήκες για την clockwise εκδοχή. Ο mpl6 παραμένει ο πιο βαρύς αν και οι διαφορές χρόνου έχουν μειωθεί σε σχέση με τον bottom.

(4.2.4 Αποτελέσματα εκδοχής «Left-Right» Tetris)

Global Placer: GORDIAN

Benchmark	THPWL	Displacement	Runtime
ibm01	14878170	6314822	0.11307
ibm02	28274698	13909677	0.176594
ibm03	40748500	20768672	0.235137
ibm04	53822527	27943787	0.264046
ibm05	53460088	29124665	0.301091
ibm06	55455110	29572766	0.314637
ibm07	77168764	35638123	0.508543
ibm08	84609030	41468432	0.564004
ibm09	106675516	46101938	0.618986
ibm10	168700804	73684452	0.886621
ibm11	157269057	66291034	0.852445
ibm12	183087765	78355556	0.931552
ibm13	153741790	56847188	1.121215
ibm14	588845631	318068166	2.1698
ibm15	697004034	311798733	2.702589
ibm16	838217298	407325427	3.28881
ibm17	959385606	449136457	3.621292
ibm18	954199508	565007851	4.008586

Global Placer: ePlace

Benchmark	THPWL	Displacement	Runtime
ibm01	14264698	6273391	0.107556
ibm02	28127404	13991640	0.17788
ibm03	39893848	20458523	0.20682
ibm04	60119321	54843590	0.220564
ibm05	53761557	31431658	0.271596
ibm06	55115825	29732885	0.30319
ibm07	76057665	35618499	0.464677
ibm08	82399173	41269227	0.531638
ibm09	105909188	46417265	0.574676
ibm10	164768420	72534825	0.843308
ibm11	153315514	66622290	0.842407
ibm12	180263044	80718542	0.921118
ibm13	146308564	55725401	1.149694
ibm14	583766760	308798515	2.349663
ibm15	701214173	310719848	2.777279
ibm16	843180030	406863594	3.424578
ibm17	963317353	447023239	3.568851
ibm18	940775895	552951018	3.960965

Global Placer: mpl6

Benchmark	THPWL	Displacement	Runtime
ibm01	14298824	6336028	0.105744
ibm02	27888786	13935552	0.171742
ibm03	39860062	20090959	0.220062
ibm04	54143703	28060440	0.268061
ibm05	52942399	30935092	0.273747
ibm06	55396875	30332075	0.29855
ibm07	76291018	35993461	0.508318
ibm08	81706626	41182985	0.531876
ibm09	104300415	45552786	0.577273
ibm10	161756620	74096845	0.844346
ibm11	151499075	65797690	0.807338
ibm12	176798362	77770269	0.902345
ibm13	144714363	55674333	1.072087
ibm14	583502510	315189549	2.101877

Benchmark	THPWL	Displacement	Runtime
ibm15	690946851	310012806	2.617969
ibm16	830703826	405454670	3.340815
ibm17	945709474	446635843	3.481941
ibm18	952843006	568579201	3.912342

Η left-right έκδοση παρουσιάζει πολύ καλό runtime και στους τρεις placers. Καταφέρνει να διατηρήσει το displacement σε ανταγωνιστικά επίπεδα χωρίς ιδιαίτερες απότομες αυξήσεις.

Benchmark	Runtime στο benchmark 18 (τελευταίο)
Bottom	5.85s (mpl6)
Clockwise	4.21s (mpl6)
Left-to-Right	**3.91s** (mpl6) ☑

Σε πολύ μικρά benchmarks, όπως τα πρώτα 3, εμφανίζει εξαιρετική απόδοση. Σε benchmarks μεγάλου μεγέθους (ibm14-ibm18) φαίνονται ξεκάθαρα τα πλεονεκτήματα της εκδοχής, με ~20 – 25% καλύτερο runtime από τις άλλες εκδοχές.

Εκδοχή Tetris	Displacement	Runtime	Σχόλιο
Bottom	✗ Πολύ υψηλό	✗ Υψηλό	Συχνά κολλάει
Clockwise	☑ Πολύ καλό	☑ Καλό	Καλή σταθερότητα
Left-to-Right	☑ Πολύ καλό	☑ **Άριστο**	**Η ταχύτερη και πιο σταθερή εκδοχή**

(4.3 Ανάλυση Αποτελεσμάτων Ευρετικών Προσεγγίσεων – Γενικές Παρατηρήσεις)

(4.3.1 Αποτελέσματα Grid-Based Tetris με ePlace τοποθέτηση)

Global Placer: ePlace

2x2 Grid

Benchmark	THPWL	Displacement	Runtime
ibm01	16063071	8970156	0.093147
ibm02	29397604	17498503	0.148815
ibm03	40581249	24209216	0.176591
ibm04	60284751	55540780	0.197107
ibm05	53989499	36910319	0.260283
ibm06	55191233	36200261	0.270717
ibm07	90639503	57198346	0.405571
ibm08	97952210	65793739	0.469218
ibm09	125842249	73904531	0.496128
ibm10	196051013	118894839	0.717985
ibm11	182554785	107883725	0.720801
ibm12	216702586	130102824	0.745128
ibm13	212186802	120201107	0.899052
ibm14	595645456	379165445	1.737963
ibm15	728079348	398773134	2.247921
ibm16	869760266	515437026	2.750138
ibm17	987284850	567587795	3.072005
ibm18	940028478	668200619	3.446261

Global Placer: ePlace

4x4 Grid

Benchmark	THPWL	Displacement	Runtime
ibm01	14274000	7076217	0.093856
ibm02	28545478	15252051	0.148679
ibm03	38423801	18938532	0.191772
ibm04	60536960	55490114	0.20065
ibm05	52093825	29384327	0.245119
ibm06	52560463	28278201	0.280602
ibm07	72838798	38576423	0.406048
ibm08	79215392	44856058	0.486326
ibm09	101136531	50101189	0.613281
ibm10	158752521	77507932	0.765379
ibm11	145050671	73936178	0.802385
ibm12	174568852	89264238	0.825025
ibm13	156765823	77783375	0.939714
ibm14	573567670	337068288	1.965216
ibm15	696701227	345808768	2.334241
ibm16	834638398	451465122	3.405154
ibm17	950123335	494594602	3.710385
ibm18	889284057	516014903	3.592762

Global Placer: ePlace

8x8 Grid

Benchmark	THPWL	Displacement	Runtime
ibm01	13821513	5956053	0.095237
ibm02	28500357	14150704	0.151751
ibm03	37747815	16636340	0.247793
ibm04	61078299	56410042	0.211242
ibm05	51436311	25402099	0.246012
ibm06	52063294	24183259	0.270809
ibm07	67924126	28724777	0.412648
ibm08	71755579	32128363	0.464861
ibm09	94000120	36263419	0.498717
ibm10	145927705	58093945	0.770021
ibm11	131411830	51654787	0.81951
ibm12	159676663	63931512	0.805833
ibm13	126192503	51395264	0.987471
ibm14	562431459	307980306	2.069075

Benchmark	THPWL	Displacement	Runtime
ibm15	686136639	315265458	2.730532
ibm16	811699744	410309380	3.168287
ibm17	925633645	449087609	4.189974
ibm18	858597983	446171713	3.749986

Global Placer: ePlace

16x16 Grid

Benchmark	THPWL	Displacement	Runtime
ibm01	13466814	5367240	0.102165
ibm02	30775928	14252894	0.159497
ibm03	40590647	16670972	0.220583
ibm04	63457089	58272067	0.212059
ibm05	54270274	25220756	0.257741
ibm06	56256331	24503904	0.282045
ibm07	66692137	24968629	0.418294
ibm08	73608177	29167766	0.474951
ibm09	92751894	32260211	0.502903
ibm10	146031072	51670528	0.718995
ibm11	127060104	45077215	0.688698
ibm12	154242952	54550528	0.763038
ibm13	112604456	38190659	0.90335
ibm14	564895922	297703121	1.764159
ibm15	698063912	305222194	2.237905
ibm16	820217857	391322949	2.711117
ibm17	935368914	431172752	2.971676
ibm18	866777994	421977385	3.53249

Algorithm	THPWL Trend
2x2	Το μεγαλύτερο THPWL
4x4	Μικρή μείωση σε σχέση με το 2x2
8x8	Καλύτερο από το 4x4 σε μικρά benchmarks, χειρότερο σε μεγάλα
16x16	Γενικά, το μικρότερο THPWL σε όλες τις περιπτώσεις

Παρατηρούμε πως, όσο το πλέγμα μικραίνει (16x16), οι τιμές του μήκους καλωδίου μειώνονται και αυτές, που σημαίνει καλύτερη κατανομή των κελιών μέσα στα πλέγματα.

Algorithm	Displacement Trend
2x2	Υψηλές τιμές σχεδόν σε όλες τις περιπτώσεις
4x4	Μεγάλη βελτίωση (π.χ., το ibm14 πέφτει από 379M σε 337M)
8x8	Βελτίωση μέχρι και το ibm13 αλλά μετά παρόμοιες τιμές
16x16	Γενικά, η χαμηλότερη μετακίνηση στα μεγάλα benchmarks

Επίσης, όσο μειώνουμε τις διαστάσεις του πλέγματος, μειώνεται και η συνολική μετακίνηση των κελιών. Για παράδειγμα: Για το ibm18

2x2: 668M

4x4: 516M

8x8: 446M

16x16: 421M (μικρότερο)

Algorithm	Runtime Trend
2x2	Γρηγορότερος από άλλες εκδοχές
4x4	Ελάχιστα πιο αργός
8x8	Εξίσου ταχύς με τον 4x4, πιο αργός στα μεγάλα benchmarks
16x16	Ο χρόνος εκτέλεσης εμφανίζει απότομες αυξήσεις λόγω των περισσότερων πλεγμάτων που έχουν δημιουργηθεί

Όσο μειώνεται το μέγεθος του πλέγματος και αυξάνονται τα κομμάτια του τσιπ στα οποία πρέπει να κάνουμε νομιμοποίηση, αυξάνεται και ο μέσος χρόνος εκτέλεσης αλλά παραμένει πάντα σε αποδεκτά επίπεδα.

Η καλύτερη εκδοχή του grid-based Tetris με αρχική κατάσταση από τον ePlace αλγόριθμο είναι η 16x16. Χαμηλότερες τιμές THPWL στα μεγάλα becnhmarks, με τη χαμηλότερη μετακίνηση και αποδεκτό χρόνο εκτέλεσης (3.53 δευτερόλεπτα για το ibm18)

(4.3.2 Αποτελέσματα Grid-Based Tetris με Mpl6 τοποθέτηση)

Global Placer: Mpl6

2x2 Grid

Benchmark	THPWL	Displacement	Runtime
ibm01	15949986	9165159	0.096584
ibm02	29224830	17559734	0.147708
ibm03	40306144	24132716	0.177676
ibm04	53512265	33606470	0.219892
ibm05	53554081	36404110	0.240899
ibm06	55153607	36976920	0.265015
ibm07	89574542	58268497	0.405159
ibm08	98274006	66466854	0.475773
ibm09	124468472	75491626	0.49627
ibm10	198002397	123343056	0.723511
ibm11	182532719	109674371	0.694003
ibm12	212646920	129769121	0.742886
ibm13	211494347	122179952	0.908433
ibm14	599017800	388682612	1.968244
ibm15	727572577	404845747	2.797862
ibm16	870030200	527672292	3.574015
ibm17	987881022	580731435	3.473864
ibm18	946109103	690168989	4.169645

Global Placer: Mpl6

4x4 Grid

Benchmark	THPWL	Displacement	Runtime
ibm01	14209712	7320821	0.096097
ibm02	28088649	15140761	0.161712
ibm03	38001421	18596039	0.242318
ibm04	50745205	26154509	0.234797
ibm05	51858424	28862147	0.246788
ibm06	52407019	28932992	0.280355
ibm07	72738373	40161749	0.410684
ibm08	78488828	46212398	0.467118
ibm09	99919589	53149404	0.497742
ibm10	156577108	84175167	0.711365
ibm11	144154522	73193099	0.695596
ibm12	170952835	91565586	0.770185
ibm13	154721192	80948258	0.916564
ibm14	574422999	347875776	1.741221
ibm15	694309362	350114799	2.297313
ibm16	833215059	458854296	2.778689
ibm17	946906708	503036317	3.109353
ibm18	879195398	528907976	3.339928

Global Placer: Mpl6

8x8 Grid

Benchmark	THPWL	Displacement	Runtime
ibm01	13402238	5926368	0.093449
ibm02	27959518	13864210	0.151178
ibm03	37194193	16358433	0.214535
ibm04	48946800	22659221	0.237696
ibm05	51034671	25541052	0.25894
ibm06	51871599	24806984	0.398121
ibm07	66048851	29263230	0.448865
ibm08	70135153	32304179	0.484208
ibm09	90958718	38001302	0.524024
ibm10	141378983	59926958	0.771607
ibm11	128945765	53467201	0.782319
ibm12	154974310	65463383	0.77369
ibm13	120937742	49888674	0.983074
ibm14	560721770	312816903	2.149799

Benchmark	THPWL	Displacement	Runtime
ibm15	681296823	319332144	2.458741
ibm16	804763507	414829094	3.161784
ibm17	914108176	453965884	3.266507
ibm18	851765590	456619501	3.370157

Global Placer: Mpl6

16x16 Grid

Benchmark	THPWL	Displacement	Runtime
ibm01	13329834	5408267	0.102019
ibm02	30396661	14305145	0.168062
ibm03	40239234	16634051	0.238355
ibm04	51494994	21807348	0.240072
ibm05	53652079	25207727	0.253184
ibm06	56078875	24500642	0.275916
ibm07	63688255	24389927	0.426135
ibm08	71679203	28405826	0.481134
ibm09	87489709	31491592	0.505449
ibm10	137990816	50604302	0.734848
ibm11	122501738	44330461	0.711896
ibm12	148105948	53847357	0.771237
ibm13	107717709	37332368	0.909587
ibm14	562891558	301881263	1.75043
ibm15	689089858	305103023	2.240012
ibm16	815018674	395924540	2.734835
ibm17	926632841	434396799	3.660907
ibm18	854790439	425127449	4.448181

Παρατηρούμε αρκετές ομοιότητες στα αποτελέσματα με αρχικές συνθήκες από τον Mpl6. Ωστόσο κάποιες σημαντικές παρατηρήσεις είναι οι εξής: Η απόδοση στα μικρότερα κυκλώματα (ibm01-ibm05) είναι σχεδόν ίδια, ανεξάρτητα της εκδοχής του αλγόριθμου. Αυτό, δείχνει πως τα benchmarks δεν επωφελούνται σημαντικά από τον διαχωρισμό του τσιπ σε μικρότερα πεδία. Τα μεγαλύτερα benchmarks (ibm10-ibm18) παρουσιάζουν σημαντικές βελτιώσεις σε τιμές THPWL στις 8x8 και 16x16 εκδοχές. Η εκδοχή 2x2 έχει σταθερά την

χειρότερη απόδοση σε μεγάλα κυκλώματα και ειδικά στο displacement. Αναμενόμενο, από την στιγμή που οι ευρετικές grid-based ενσωματώνουν fallback πέρασμα για τοποθέτηση των κελιών τα οποία δεν βρήκαν θέση στο πρώτο πέρασμα, άρα σε μεγαλύτερα κυκλώματα έχουμε μεγάλες μετακινήσεις. Οι 16x16 και 8x8 εκδοχές είναι πολύ κοντά σε benchmarks μεσαίου μεγέθους, αλλά η 16x16 βελτιώνει αρκετά τα αποτελέσματά της στα μεγαλύτερα κυκλώματα.

(4.3.3 Αποτελέσματα Grid-Based Tetris με GORDIAN τοποθέτηση)

Global Placer: GORDIAN

2x2 Grid

Benchmark	THPWL	Displacement	Runtime
ibm01	16358510	8949462	0.109827
ibm02	29702703	17552206	0.160168
ibm03	41278831	25339209	0.190716
ibm04	54181260	33792800	0.228502
ibm05	54506543	35312238	0.258477
ibm06	55314200	36241911	0.280302
ibm07	93000139	58470551	0.424719
ibm08	100497359	67095376	0.493629
ibm09	127135139	74968219	0.525465
ibm10	203211744	122209920	0.755126
ibm11	190093757	111754588	0.726299
ibm12	222777394	130727148	0.782666
ibm13	215337784	120184284	0.948019
ibm14	604040155	391242350	1.872825
ibm15	730779668	405185723	2.320486
ibm16	876077364	527126593	3.486213
ibm17	1000212820	580991765	3.774477
ibm18	954311981	687517757	4.144521

Global Placer: GORDIAN

4x4 Grid

Benchmark	THPWL	Displacement	Runtime
ibm01	15262243	7406417	0.105366
ibm02	28804764	15329882	0.162753
ibm03	39106145	19790796	0.207857
ibm04	50556572	26063667	0.246207
ibm05	51983295	27772657	0.270834
ibm06	52977129	28558053	0.297019
ibm07	75152594	41165053	0.443817
ibm08	80375626	46816276	0.493994
ibm09	103488198	52958018	0.533259
ibm10	166507690	86621680	0.778464
ibm11	152771508	77902784	0.81319
ibm12	180525378	91846052	0.81842
ibm13	160920646	79705046	1.003917
ibm14	581660049	349150350	2.010812
ibm15	699554843	351163446	3.275757
ibm16	835685050	457606334	3.423628
ibm17	962294469	506303049	3.840037
ibm18	892392532	531781673	3.61014

Global Placer: GORDIAN

8x8 Grid

Benchmark	THPWL	Displacement	Runtime
ibm01	14689539	6185947	0.105486
ibm02	28851227	14065358	0.162724
ibm03	38566471	17121022	0.205238
ibm04	49563786	22696749	0.245016
ibm05	51107173	23697933	0.262009
ibm06	52907251	24886927	0.291818
ibm07	70078777	29849218	0.439428
ibm08	73068036	33089630	0.505718
ibm09	94900507	38286120	0.546453
ibm10	154605470	61520214	0.755988
ibm11	139136159	54812053	0.730176
ibm12	166604967	64459554	0.789404

Benchmark	THPWL	Displacement	Runtime
ibm13	130266030	50264514	0.970501
ibm14	568663782	316862923	1.93302
ibm15	690505144	322056179	2.472939
ibm16	811498356	415316262	2.833675
ibm17	931757016	458872615	3.165365
ibm18	865820181	460377895	3.417063

Global Placer: GORDIAN

16x16 Grid

Benchmark	THPWL	Displacement	Runtime
ibm01	14740634	5658538	0.107161
ibm02	31056242	14384382	0.178155
ibm03	41263589	16934343	0.247186
ibm04	52319269	21801306	0.249145
ibm05	53938873	23001597	0.271023
ibm06	56913450	24538161	0.309737
ibm07	68832206	25285569	0.462885
ibm08	74289716	28793640	0.506772
ibm09	93927409	32460756	0.538637
ibm10	153422197	52842959	0.792726
ibm11	133359033	45986703	0.736277
ibm12	161631177	54395188	0.800689
ibm13	121978170	38909426	0.983547
ibm14	571373955	305479656	1.888277
ibm15	699057556	308161738	2.352923
ibm16	821869422	396575547	2.865551
ibm17	943978193	438620102	3.170859
ibm18	871800817	429131305	3.757423

Εκδοχή	Μέσο THPWL	Μέσο Displacement	Μέσος Χρόνος Εκτέλεσης
2×2 Tetris	312,624,775.60	193,581,808.90	1.2519 sec
4×4 Tetris	291,589,314.90	170,445,840.10	1.2754 sec
8×8 Tetris	276,341,446.30	156,672,712.70	1.2017 sec
16×16 Tetris	276,552,314.70	148,354,912.90	1.2636 sec

Μετά από την ανάλυση των αποτελεσμάτων καταλαβαίνουμε πως η επιλογή της ευρετικής θα πρέπει να γίνει με βάση κάποιο trade-off. Συγκεκριμένα

- Αν χρειαζόμαστε μικρό μήκος καλωδίωσης τότε η 16x16 εκδοχή αποτελεί την πιο κατάλληλη
- Αν χρειαζόμαστε την μικρότερη μετακίνηση τότε πάλι διαλέγουμε την 16x16
- Για ισορροπία στα αποτελέσματα τότε η 4x4 είναι η καλύτερη λύση
- Για χρόνο εκτέλεσης η 8x8 και η 4x4 είναι προτιμότερες

(4.4.1 Αποτελέσματα Περιοριστικού Tetris)

Όπως έχουμε αναφέρει και στο προηγούμενο κεφάλαιο, η λειτουργία των περιοριστικών ευρετικών του αλγορίθμου στηρίζεται στον διαθέσιμο αριθμό σειρών που επιτρέπεται να χρησιμοποιήσουν. Αυτός δίνεται σαν input από τον χρήστη, στην πρώτη εκδοχή σαν αριθμός και στην δεύτερη σαν ποσοστό. Τα παρακάτω αποτελέσματα έχουν παραχθεί με διαθέσιμες 30 σειρές σε όλα τα benchmark της πρώτης εκδοχής και στην δεύτερη εκδοχή είναι διαθέσιμο το 30% από όλες τις σειρές. Και στις δύο περιπτώσεις, το base row είναι 50.

Global Placer: ePlace

Number Ristriction

Benchmark	THPWL	Displacement	Runtime
ibm01	20261152	12741542	1.913819
ibm02	31901089	24922867	8.971399
ibm03	32363513	14318107	0.955348
ibm04	51671342	26709063	0.567223
ibm05	45268694	21781987	1.194518
ibm06	44518936	21249232	0.566693
ibm07	71332497	30831960	0.973306
ibm08	75637543	34815131	0.919382
ibm09	93109072	36470756	0.870975
ibm10	134791462	49630379	1.076058
ibm11	131264449	49026185	1.048213
ibm12	145127912	52422999	1.2021
ibm13	165415198	59616882	1.316725
ibm14	510261837	474015483	2.380021
ibm15	650873124	524140326	4.335369
ibm16	751811742	644562949	3.428289
ibm17	852071232	709621432	3.781941
ibm18	871904514	546225390	4.107662

Global Placer: ePlace

Perventage Ristriction

Benchmark	THPWL	Displacement	Runtime
ibm01	19546503	11039165	0.772847
ibm02	30990214	25505737	0.776851
ibm03	32363513	14318107	0.747601
ibm04	54985474	30174385	0.820399
ibm05	44946823	22867772	0.775277
ibm06	45432948	21987945	0.775029
ibm07	84857765	42453205	0.797547
ibm08	91254714	48972755	0.820428
ibm09	119385031	56647511	0.793908
ibm10	185301958	88279185	0.878149

Benchmark	THPWL	Displacement	Runtime
ibm11	178093954	83877405	0.806994
ibm12	200216701	93697518	0.802607
ibm13	229943293	105189905	0.94629
ibm14	484935875	587290241	1.780506
ibm15	609368313	648276650	3.501412
ibm16	727600785	810236720	3.800585
ibm17	829437675	891526398	3.884945
ibm18	833318768	441049568	4.504154

Global Placer: GORDIAN

Number Ristriction

Benchmark	THPWL	Displacement	Runtime
ibm01	20355348	13287749	0.683032
ibm02	31837031	25634358	0.584552
ibm03	33933333	15159526	0.705016
ibm04	43285100	18482111	0.645562
ibm05	45797540	19759467	0.71122
ibm06	45864688	21657253	0.762172
ibm07	74405451	31375800	0.974817
ibm08	79114545	35312926	0.868432
ibm09	97181537	37199682	0.856502
ibm10	144606715	49762572	1.06481
ibm11	141072456	50257961	1.075147
ibm12	156564661	51502279	1.086744
ibm13	173748520	60210305	0.965054
ibm14	513156207	471883939	2.282342
ibm15	643902000	512645912	2.621802
ibm16	755885029	649820971	3.149132
ibm17	862474723	705752639	3.40945
ibm18	886780246	543907634	3.935442

Global Placer: GORDIAN

Perventage Ristriction

Benchmark	THPWL	Displacement	Runtime
ibm01	19702267	11503724	1.623013
ibm02	30899324	26323036	0.987172
ibm03	33933333	15159526	1.000352
ibm04	46051676	21073816	1.033255
ibm05	47231602	22301040	0.983186
ibm06	46729592	22408694	0.983965
ibm07	87142078	43411752	1.172072
ibm08	94087316	49830052	1.09944
ibm09	122123444	57850457	0.96478
ibm10	192060174	89804226	1.033335
ibm11	183757142	86276450	0.98685
ibm12	207076712	94104169	1.00814
ibm13	236190186	106779326	1.084479
ibm14	483377828	587583732	1.855668
ibm15	606089033	638691108	2.349115
ibm16	725892665	818847132	3.838186
ibm17	826833120	891734544	3.083725
ibm18	853057773	444352714	4.449839

Global Placer: Mpl6

Number Ristriction

Benchmark	THPWL	Displacement	Runtime
ibm01	20280679	13318599	0.712193
ibm02	31901089	25926471	0.637408
ibm03	32876490	14623226	0.766155
ibm04	42834805	18803909	0.674592
ibm05	45808014	22108318	0.582718
ibm06	45947223	22298367	0.570327
ibm07	72754714	31604481	0.656397
ibm08	76443039	35039576	0.850267
ibm09	97219469	38300959	0.816838
ibm10	134778365	49690435	0.722276
ibm11	133737324	49766232	1.172395
ibm12	147278854	52698561	1.093246
ibm13	167856647	60599774	1.169278

Benchmark	THPWL	Displacement	Runtime
ibm14	511827391	469193541	1.765313
ibm15	634717960	509125937	2.778807
ibm16	750173999	645539033	3.11485
ibm17	852661664	709065282	2.958857
ibm18	879645587	546853748	3.873731

Global Placer: Mpl6

Perventage Ristriction

Benchmark	THPWL	Displacement	Runtime
ibm01	19701817	11546682	8.286447
ibm02	30990214	26707805	1.434763
ibm03	32876490	14623226	1.226755
ibm04	45377023	21141366	1.19275
ibm05	45469140	23278453	1.209491
ibm06	46825760	23026517	1.423082
ibm07	86354674	43685093	1.35467
ibm08	92100153	49357409	1.759267
ibm09	123852012	59185908	2.018638
ibm10	185662481	88671768	1.768282
ibm11	180230999	85227190	1.740445
ibm12	201844107	94250456	1.942205
ibm13	232954172	106686924	2.176335
ibm14	485705341	582953241	4.466025
ibm15	606251500	633415305	3.84002
ibm16	726703875	813421327	4.100684
ibm17	830844062	894591572	3.917949
ibm18	841101794	444875914	4.211189

ePlace

Μετρική	Καλύτερη Εκδοχή	Παρατηρήσεις
THPWL	Αριθμός	Σχεδόν πάντα καλύτερο (εκτός από ibm04)
Displacement	Ποσοστό	Χαμηλότερες τιμές, ειδικά στα μικρά benchmarks
Runtime	Ποσοστό	Αρκετά πιο γρήγορη εκδοχή

Οι αρχική κατάσταση του ePlace ευνοεί την ευρετική με βάσει ποσοστό, όσον αφορά τον χρόνο εκτέλεσης και την συνολική μετατόπιση. Η ευρετική με βάσει αριθμό υπερισχύει ελάχιστα στην ποιότητα του μήκους καλωδίου.

GORDIAN

Μετρική	Καλύτερη Εκδοχή	Παρατηρήσεις
THPWL	Αριθμός	Καλύτερο THPWL στα 15/18 benchmarks
Displacement	Και οι δύο	Η ευρετική ποσοστού αποδίδει καλύτερα σε μεγάλα benchmarks αλλά όχι τόσο στα μικρότερα
Runtime	Αριθμός	Μικρότερος μέσος όρος χρόνου εκτέλεσης

Οι αρχική κατάσταση του GORDIAN προσφέρει καλύτερη ποιότητα σε μήκος καλωδίου. Ο χρόνος εκτέλεσης είναι παρόμοιος και στις δύο εκδοχές. Η ποιότητα της συνολικής μετατόπισης εξαρτάται από το μέγεθος του benchmark.

Mpl6

Μετρική	Καλύτερη Εκδοχή	Παρατηρήσεις
THPWL	Αριθμός	Μέχρι και το ibm13
Displacement	Αριθμός	-
Runtime	Αριθμός	Πιο γρήγορος σε όλα τα benchmarks

Το mPL6 ευνοεί έντονα τον περιορισμό βάσει αριθμών σε όλες τις μετρήσεις. Ο ακραίος χρόνος εκτέλεσης για το ibm01 με βάση το ποσοστό μπορεί να υποδηλώνει αναποτελεσματικότητα στην αρχικοποίηση ή την κλιμάκωση εισόδου.

(4.4.1 Αποτελέσματα Tetris με Συγχρονισμό Δικτύων)

Global Placer: GORDIAN

Net-Sync

Benchmark	THPWL	Displacement	Runtime
ibm01	17993455	12638942	0.106771
ibm02	30020234	22039845	0.170604
ibm03	44039025	28734602	0.235327
ibm04	58947305	38620344	0.264987
ibm05	58082747	41102938	0.294607
ibm06	61723874	42034239	0.332901
ibm07	111233738	78762543	0.502119
ibm08	117772784	89039482	0.565984
ibm09	156387292	99938423	0.611409
ibm10	243957226	150934823	0.880009
ibm11	232204948	151293344	0.852907
ibm12	259384724	170923484	0.954299
ibm13	307923472	198394724	1.146649
ibm14	620384752	445834722	2.35087
ibm15	789843205	499398424	3.039573
ibm16	942223573	600495735	3.823915
ibm17	1057238942	680394724	4.222858
ibm18	1069374292	730934824	4.886612

Global Placer: ePlace

Net-Sync with Fallback

Benchmark	THPWL	Displacement	Runtime
ibm01	18719557	12461617	0.098671
ibm02	31263529	22256886	0.161971
ibm03	44113730	27585783	0.214872
ibm04	58608480	57551448	0.231509
ibm05	58433688	41550513	0.291901
ibm06	61720246	42112024	0.315879
ibm07	112698262	77532309	0.48016
ibm08	121598267	88656125	0.537575
ibm09	156584884	99081147	0.579015
ibm10	246906372	163505786	0.854291
ibm11	233049400	147846949	0.887114
ibm12	267045652	173628464	0.982214
ibm13	307609377	193185625	1.903471
ibm14	633774480	455013176	3.617334
ibm15	790701164	499895107	3.196364
ibm16	946516188	650452463	3.845633
ibm17	1066191721	713912628	4.427142
ibm18	1061318618	774903245	4.953229

Οι δύο βασικές εκδοχές του αλγορίθμου νομιμοποίησης με συγχρονισμό δικτύων είναι η κλασική παραλλαγή και η παραλλαγή με fallback πέρασμα, για τοποθέτηση των κελιών που έμειναν. Η στρατηγική της επαναληπτικής νομιμοποίησης (με κλασική λογική Tetris), έχει σημαντική επίδραση στα αποτελέσματα του αλγορίθμου. Συγκεκριμένα, το THPWL παρατηρείται αυξημένο, ασχέτως του αρχικού placer. Η συνολική μετατόπιση παραμένει γενικά η ίδια, εκτός από κάποιες περιπτώσεις (ePlace με fallback), όπου φαίνεται να αυξάνεται ελαφρώς. Ο χρόνος εκτέλεσης αυξάνεται σημαντικά στην fallback εκδοχή, ειδικά στα μεγαλύτερα κυκλώματα (ibm14-ibm18). Αυτό αποδεικνύει πως ο μηχανισμός επαναληπτικής τοποθέτησης εστιάζει στη σωστή νομιμοποίηση, αλλά δεν βελτιώνεται η ποιότητα του THPWL.

Mpl6

THPWL

- Το fallback αυξάνει το THPWL σε όλα τα benchmarks

Displacement

- Ελάχιστες αλλαγές, κάποια benchmarks όπως τα ibm01, ibm05, ibm06 εμφανίζουν μικρές αυξήσεις.

Runtime

- Γενικά, ελάχιστη αύξηση

GORDIAN

THPWL

- Παρόμοια συμπεριφορά με τον Mpl6. Μικρή αύξηση του THPWL στην fallback εκδοχή.

Displacement

- Τα αποτελέσματα δεν είναι σταθερά. Σε κάποιες περιπτώσεις καλύτερο (ibm16), κάποιες χειρότερο.

Runtime

- Πολύ σταθερό

ePlace

THPWL

- Πολύ μεγάλη αύξηση του THPWL στην fallback έκδοση (ibm03 από 22 σε 44 εκατομμύρια).

Displacement

- Σημαντική βελτίωση για πολλά benchmarks

Runtime

- Περίπου ίδιο με την έκδοση χωρίς fallback.

(4.4.2 Αποτελέσματα Tetris Δικτυακού Κέντρου)

Global Placer: Mpl6

Net-Center

Benchmark	THPWL	Displacement	Runtime
ibm01	18010268	12722446	0.095418
ibm02	30775011	22241157	0.204974
ibm03	43360150	27649689	0.183575
ibm04	58033273	38426860	0.221896
ibm05	58433688	42244746	0.249104
ibm06	61297974	43274927	0.272095
ibm07	111295248	79089412	0.418101
ibm08	120570075	90264287	0.477606
ibm09	155132362	101467592	0.515552
ibm10	243326440	166518127	0.849344
ibm11	230457602	150013582	0.729231
ibm12	262711750	176824142	0.778996
ibm13	304917874	196900007	1.050731
ibm14	630005109	465888186	1.794561
ibm15	788147850	508872970	2.319559
ibm16	941976750	668543922	2.803855
ibm17	1058182902	733418628	3.060106
ibm18	1058757904	796722649	3.338803

Global Placer: GORDIAN

Net-Center

Benchmark	THPWL	Displacement	Runtime
ibm01	18006794	12536157	0.106932
ibm02	30558884	21939715	0.16216
ibm03	43197165	28585755	0.196047
ibm04	57913298	38772613	0.240603
ibm05	58433688	41105257	0.263165
ibm06	61350652	42659720	0.290039

Benchmark	THPWL	Displacement	Runtime
ibm07	111370442	79086572	0.44004
ibm08	120578848	90519536	0.504932
ibm09	155279562	101396393	0.542472
ibm10	242165423	167305007	0.774127
ibm11	230646409	152068328	0.750542
ibm12	263712336	178311428	0.815802
ibm13	304368520	195438075	0.984471
ibm14	630898921	469254410	1.924768
ibm15	787308418	510848979	2.725497
ibm16	940474563	664404114	3.121484
ibm17	1056563904	731111091	3.173783
ibm18	1057728373	794065726	4.088655

Global Placer: ePlace

Net-Center

Benchmark	THPWL	Displacement	Runtime
ibm01	18064366	12316229	0.097732
ibm02	30722401	22310201	0.187316
ibm03	43411287	27721174	0.220192
ibm04	57643421	57920179	0.212654
ibm05	58433688	41550513	0.258153
ibm06	61333117	42013186	0.28428
ibm07	111317798	77272787	0.413914
ibm08	120322789	88576032	0.475578
ibm09	155329703	98659598	0.52455
ibm10	242874352	162596462	0.801741
ibm11	230736546	147149561	0.70846
ibm12	262804128	173358420	0.772676
ibm13	304924985	192600652	1.015413
ibm14	630917640	455053438	1.832017
ibm15	787095322	499208441	2.309002
ibm16	941939445	650645121	2.754014
ibm17	1058484458	712847914	3.054272
ibm18	1058735758	773872035	3.368655

Γενικά, το THPWL παραμένει αρκετά σταθερό σε όλα τα placers αλλά, ο GORDIAN placer παράγει ελαφρώς χαμηλότερο μήκος καλωδίου από τους υπόλοιπους. Ο ePlace, παράγει σταθερά την χαμηλότερη μετακίνηση. Τα πλεονεκτήματά του φαίνονται στα μεγαλύτερα κυκλώματα, που υποδηλώνει καλύτερη αρχική τοποθέτηση. Όσον αφορά τον χρόνο εκτέλεσης, ο Mpl6 παράγει τα καλύτερα αποτελέσματα. Γενικά η ευρετική εκδοχή του αλγορίθμου με δικτυακό κέντρο δείχνει ανθεκτική σε μη βέλτιστες εισόδους. Ακόμη και σε περιπτώσεις που το displacement είναι αρκετά αυξημένο, ο αλγόριθμος παραμένει γρήγορος και με εύλογο THPWL.

(4.4.3 Αποτελέσματα Tetris με αλλαγή φοράς Δικτυακού Κέντρου)

Global Placer: Mpl6

Clockwise Net-Center

Benchmark	THPWL	Displacement	Runtime
ibm01	23912319	17831601	0.099331
ibm02	42939476	30411273	0.152795
ibm03	50658488	30327892	0.218764
ibm04	74778329	48291098	0.233544
ibm05	58433688	42244746	0.252877
ibm06	73753700	48387672	0.279306
ibm07	122686037	83972933	0.427284
ibm08	159040977	120441763	0.484592
ibm09	182548359	111924609	0.524927
ibm10	306809902	217198811	0.743104
ibm11	261283377	164975680	0.703389
ibm12	296420356	205802554	0.772716
ibm13	386385764	280562680	0.93945
ibm14	652077655	472044489	1.859121
ibm15	1056101473	616160567	2.349019
ibm16	1184919669	776713196	2.819688
ibm17	1317723735	863740315	3.212784
ibm18	1193320362	846116025	3.422312

Global Placer: GORDIAN

Counterclockwise Net-Center

Benchmark	THPWL	Displacement	Runtime
ibm01	22312968	15660636	0.103929
ibm02	37221117	25758225	0.164538
ibm03	51406167	32634356	0.222455
ibm04	70516417	45630762	0.25558
ibm05	58433688	41105257	0.272696
ibm06	72267726	47239228	0.289561
ibm07	132163100	90840133	0.438986
ibm08	139265402	102060403	0.505566
ibm09	186887633	117619850	0.54174
ibm10	281901321	208712677	0.793027
ibm11	285002755	186933276	0.748128
ibm12	320049887	235172914	0.817217
ibm13	355380926	250477082	0.978654
ibm14	722202225	508722822	1.961303
ibm15	921050021	559120627	2.487331
ibm16	1016351208	688233955	2.955566
ibm17	1286904990	835995820	3.175714
ibm18	1144899706	823469264	3.500666

Global Placer: ePlace

Top Bottom Net-Center

Benchmark	THPWL	Displacement	Runtime
ibm01	18857173	12860092	0.101926
ibm02	31297372	22670483	0.158664
ibm03	44269091	28141831	0.189868
ibm04	58290928	56152266	0.227285
ibm05	58433688	41550513	0.269668
ibm06	61656959	42334263	0.304213
ibm07	112919212	78711307	0.418222
ibm08	120482960	89408888	0.492601
ibm09	155945150	99588803	0.529633
ibm10	247068928	168031093	0.773324

Benchmark	THPWL	Displacement	Runtime
ibm11	232300085	149228390	0.729197
ibm12	265303036	176275291	0.798701
ibm13	307050755	194529275	0.986595
ibm14	635028400	458026827	1.888892
ibm15	788965877	502053829	2.439507
ibm16	946342254	653741840	2.857017
ibm17	1065749759	719126376	3.232736
ibm18	1061920730	777695664	3.575875

Global Placer: GORDIAN

Left Right Net-Center

Benchmark	THPWL	Displacement	Runtime
ibm01	18548423	12599128	0.105686
ibm02	31324808	22132489	0.175419
ibm03	43937505	28781137	0.208344
ibm04	59379175	38981689	0.25058
ibm05	58433688	41105257	0.274465
ibm06	61686279	42805210	0.29948
ibm07	113025547	79627176	0.452948
ibm08	122057161	91034694	0.530079
ibm09	156117693	101657977	0.578891
ibm10	245873575	168183749	0.807916
ibm11	231876829	152504308	0.776547
ibm12	267233815	179087426	0.833477
ibm13	307563522	196530677	1.008561
ibm14	633087766	469134926	1.971576
ibm15	791810116	511733307	2.478985
ibm16	944179022	665360391	2.97087
ibm17	1064356843	733597320	3.25142
ibm18	1062519220	793846851	3.628817

Όσον αφορά τον χρόνο εκτέλεσης ο mpl6 δίνει τα καλύτερα αποτελέσματα, συγκεκριμένα ο μέσος όρος χρόνου εκτέλεσης στο ibm18 είναι 3.6 δεύτερα. Ο GORDIAN έχει ελάχιστα χειρότερη απόδοση στον χρόνο. Ο ePlace γίνεται

βραδύς στα μεγαλύτερα κυκλώματα, φτάνοντας και τα 4.4 δεύτερα στο ibm18. Στα μικρότερα benchmarks, όλες οι εκδοχές είναι συγκρίσιμες στον χρόνο εκτέλεσης. Ο mpl6 και ο GORDIAN με τον clockwise net center tetris παράγουν σχεδόν το ίδιο THPWL για το ibm18, 1193 και 1143 εκατομμύρια αντίστοιχα. Ο ίδιος αλγόριθμος με αρχική κατάσταση του ePlace, παράγει 1176 εκατομμύρια THPWL, αλλά με αυξημένο χρόνο κατά 1 δευτερόλεπτο περίπου.

(4.5 Συμπεράσματα)

Από την συγκριτική αξιολόγηση των 20 παραλλαγών του αλγορίθμου νομιμοποίησης Tetris, με 3 διαφορετικές αρχικές συνθήκες δοσμένες από global placers, αναδείχθηκαν σημαντικά συμπεράσματα σχετικά με την αποδοτικότητα, την σταθερότητα και την προσαρμοστικότητα τους. Αρχικά, παρατηρήθηκε πως η αρχική τοποθέτηση παίζει καίριο ρόλο στην ποιότητα της νομιμοποίησης. Συγκεκριμένα, ευρετικές υλοποιήσεις με net-aware χαρακτηριστικά εμφανίζουν σαφώς βελτιωμένα αποτελέσματα όταν ξεκινούν από τοποθετήσεις που έχουν δημιουργήσει οι αλγόριθμοι GORDIAN και ePlace, παρά ο Mpl6. Επιπλέον, αποδείχθηκε η χρησιμότητα των παραλλαγών, καθώς σε αρκετές περιπτώσεις κατέγραψαν καλύτερες επιδόσεις, με μειωμένο THPWL, displacement και χρόνο εκτέλεσης από την κλασική υλοποίηση του Tetris. Χαρακτηριστικό παράδειγμα αποτελεί ο Tetris με φορά left-right και net-aware στρατηγική. Έχουμε επίσης διαπιστώσει, πως ορισμένες παραλλαγές του αλγορίθμου εμφανίζουν σταθερότητα, ανεξάρτητα από τις αρχικές συνθήκες. Αυτό σημαίνει πως θα μπορούσαν να ενσωματωθούν σε πιο γενικευμένα εργαλεία τοποθέτησης. Γενικά, οι πειραματικές δοκιμές έδειξαν πως η ενίσχυση του κλασικού αλγορίθμου νομιμοποίησης Tetris με ευρετικές στρατηγικές κατάφερε να βελτιώσει την ποιότητα νομιμοποίησης και να κάνει τον αλγόριθμο πιο ευέλικτο.

(4.6 Μελλοντικές Επεκτάσεις)

(4.6.1 Tetris με region-aware (περιορισμός-περιοχής) χαρακτηριστικά)

Η ανάπτυξη μίας εκδοχής του αλγορίθμου νομιμοποίησης η οποία θα αποφασίζει ποια στρατηγική θα χρησιμοποιήσει ανάλογα με τα τοπικά χαρακτηριστικά του τσιπ. Συγκεκριμένα, σημεία στο layout τα οποία εμφανίζουν μεγάλη πυκνότητα θα αντιμετωπίζονται με ευρετικές οι οποίες δίνουν προτεραιότητα στην μείωση της συνολικής μετακίνησης. Αντίθετα, περιοχές με πιο αραιή τοποθέτηση θα μπορούσαν να εκμεταλλευτούν την σταθερότητα και την ταχύτητα του clockwise tetris. Αυτός ο αλγόριθμος θα αναλύει δυναμικά το τσιπ και θα ικανοποιεί ταυτόχρονα πολλαπλά κριτήρια. Μία τέτοια προσέγγιση θα είναι χρήσιμη σε κυκλώματα πολύ μεγάλης κλίμακας τα οποία δεν εμφανίζουν αρκετή ομοιογένεια.

(4.6.2 Tetris με Ευφυής επιλογή αρχικής τοποθέτησης)

Εξίσου ενδιαφέρουσα εκδοχή του αλγορίθμου, είναι αυτή που θα επιλέγει την αρχική τοποθέτηση πριν την νομιμοποίηση. Όπως έχουμε καταλάβει και από τα πειραματικά αποτελέσματα, οι αρχικές συνθήκες παίζουν καθοριστικό ρόλο στην ποιότητα του τελικού αποτελέσματος. Άρα, η δημιουργία ενός μηχανισμού ο οποίος θα αναλύει τα χαρακτηριστικά του κυκλώματος (π.χ. τα δίκτυα των κελιών) και με βάση αυτά θα αποφασίζει τον αρχικό placer φαίνεται χρήσιμη. Ο μηχανισμός μπορεί να στηρίζεται σε μοντέλο μηχανικής μάθησης το οποίο θα έχει εκπαιδευτεί σε benchmarks. Μία τέτοια παραλλαγή θα μείωνε την ανάγκη για παρέμβαση του προγραμματιστή στον καθορισμό των παραμέτρων, φτιάχνοντας ένα αυτοματοποιημένο σύστημα τοποθέτησης και νομιμοποίησης.

1. JUCEMAR LUIS MONTEIRO (2019). Algorithms to Improve Area Density Utilization, Routability and Timing During Detailed Placement and Legalization of VLSI Circuits. ***Digital Repository***. https://lume.ufrgs.br/handle/10183/197078

2. Panagiotis Oikonomou, Antonios N. Dadaliaris, Kostas Kolomvatsos, Thanasis Loukopoulos, Athanasios Kakarountas, Georgios I. Stamoulis (2018). Improved Parallel Legalization Schemes for Standard Cell Placement with Obstacles. *Technologies* **2019**, *7*(1), 3; **https://doi.org/10.3390/technologies7010003**

3. VÍCTOR FRANCO SÁNCHEZ (2024). COMBINATORIAL AND NUMERICAL OPTIMIZATION TECHNIQUES FOR FLOORPLANNING ***UPCommons.** Global access to UPC knowledge* https://upcommons.upc.edu/bitstream/handle/2117/410213/183175.pdf?sequence=2&isAllowed=y

4. Ulrich Brenner (2012). VLSI legalization with minimum perturbation by iterative augmentation. *IEE Xplore* https://ieeexplore.ieee.org/abstract/document/6176579

5. Hwapyong Kim (2024). Refining Algorithms for Placement Legalization and Global Routing. *S-Space* https://dcollection.snu.ac.kr/public_resource/pdf/000000180116_20250704195644.pdf