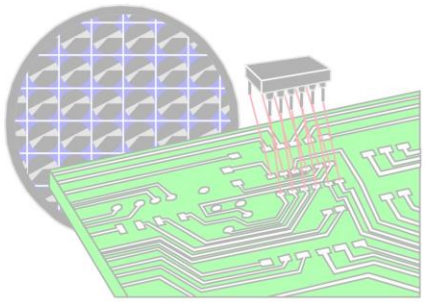
ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΕΠΕΞΕΡΓΑΣΤΩΝ & ΥΛΙΚΟΥ

«Ανάπτυξη Αναδιατασσόμενων Αρχιτεκτονικών για εξόρυξη συχνών υπογράφων»

«DEVELOPMENT OF FPGA ARCHITECTURES FOR FREQUENT SUBGRAPH MINING»

Αθανάσιος Στρατικόπουλος

Επιτροπή
Δόλλας Απόστολος, Καθηγητής (Επιβλέπων)
Παπαευσταθίου Ιωάννης, Αναπληρωτής Καθηγητής
Γαροφαλάκης Μίνως, Καθηγητής



Χανιά, Οκτώβριος 2013

Στην οικογένειά μου και στον αγαπημένο μου
θείο Δημήτρη, που έφυγε νωρίς.

Ευχαριστίες

Θα ήθελα πρωτίστως να ευχαριστήσω τον επιβλέποντα καθηγητή μου κ. Δόλλα Απόστολο, για την εμπιστοσύνη που έδειξε στις ικανότητές μου και τις συμβουλές του όλο αυτό το διάστημα. Επίσης, θα ήθελα να ευχαριστήσω τον Καθηγητή κ. Γαροφαλάκη Μίνωα και τον Αναπληρωτή Καθηγητή κ. Παπαευσταθίου Ιωάννη, για τη συμβολή τους στην αναζήτηση του θέματος της διπλωματικής μου εργασίας καθώς και για τη συμμετοχή τους ως μέλη της εξεταστικής επιτροπής.

Στη συνέχεια, θα ήθελα να ευχαριστήσω όλα τα μέλη του εργαστηρίου, τον Καθηγητή κ. Πνευματικάτο Διονύση, τον κ. Κιμιωνή Μάρκο και όλους τους προπτυχιακούς, μεταπτυχιακούς και διδακτορικούς φοιτητές του εργαστηρίου, με τους οποίους είχα καθημερινή επικοινωνία τον τελευταίο χρόνο και με βοήθησαν σε πολλά θέματα με την εμπειρία και τις συμβουλές τους. Επίσης, ευχαριστώ πολύ το Γιώργο Μαντάκο και τον Δημήτρη Σκαρλάτο, για τη συνεργασία που είχαμε τον τελευταίο χρόνο και για την υποστήριξή τους, όλο αυτό το διάστημα.

Ένα μεγάλο ευχαριστώ και στην παρέα μου, με τους οποίους σκοράραμε πολλά γκολ και κατακτήσαμε τρία κύπελλα αυτά τα έξι χρόνια, τον Μιχάλη, τον Νίκο, τον Κώστα, τον Δημήτρη, τον Δημήτρη και τον Βασίλη!! Είναι τα πρώτα παιδιά, που γνώρισα στα Χανιά και πραγματικά τους εκτιμώ πολύ για τη φιλία τους. Φυσικά είχα τη χαρά να έχω και πολλούς ακόμη φίλους, συναδέλφους και συναθλητές, τους οποίους εκτιμώ απεριόριστα.

Επίσης, ένα ευχαριστώ στους μακρινούς μου φίλους, Θανάση και Παναγιώτη, για τη στήριξή τους. Όπως και στους κοντινούς μου φίλους, Χρήστο και Φωτεινή, για την αμέριστη συμπαράσταση και για την ανοχή τους στα πειράγματά μου.

Επίσης, ένα πολύ μεγάλο ευχαριστώ στο φίλο μου Γρηγόρη Χρυσό, για την καθοδήγηση, τη στήριξη και την άψογη συνεργασία που είχαμε όλο αυτό το διάστημα. Νιώθω πραγματικά, πολύ τυχερός για αυτή τη συνεργασία.

Τέλος, θα ήθελα να εκφράσω την αγάπη και την ευγνωμοσύνη μου στους γονείς μου Δημήτρη και Σοφία και την αδερφή μου Στέλα, που είναι πάντα δίπλα μου και με στηρίζουν στα όνειρα και τις αποφάσεις μου.

Περιεχόμενα

Κεφάλαιο 4 : Αρχιτεκτονικές

Κεφάλαιο 5 : Αποτελέσματα

Κεφάλαιο 6 : Συμπεράσματα – Μελλοντικές επεκτάσεις

ΒΙΒΛΙΟΓΡΑΦΙΑ

Περίληψη

Τα τελευταία χρόνια η περιοχή του data mining έχει κεντρίσει το ενδιαφέρον της βιομηχανίας και της επιστημονικής κοινότητας, εξαιτίας της ύπαρξης μεγάλου όγκου δεδομένων σε πολλές εφαρμογές και της ανάγκης μετατροπής αυτών των δεδομένων σε χρήσιμη πληροφορία και γνώση.

Μία ειδική περίπτωση του data mining είναι το graph mining. Αντικείμενο αυτού του πεδίου είναι η εξόρυξη χρήσιμης πληροφορίας από δεδομένα, που χρησιμοποιούν ως βασική δομή δεδομένων τους γράφους. Οι γράφοι από τη φύση τους έχουν ιδιότητες, οι οποίες συχνά οδηγούν σε υψηλή αλγοριθμική πολυπλοκότητα. Παρ' όλα αυτά, έχουν αναπτυχθεί υλοποιήσεις σε γνωστά προβλήματα της περιοχής, όπως είναι η εξόρυξη συχνών μοτίβων (frequent pattern mining), η ομαδοποίηση των δεδομένων (clustering) και η ταξινόμησή τους (classification), οι οποίες καθιστούν τους γράφους ως μία κατάλληλη δομή δεδομένων για την επίλυση γνωστών προβλημάτων της περιοχής του data mining. Σε αυτά τα προβλήματα, η υψηλή πολυπλοκότητα καθώς και η ύπαρξη δεδομένων σε μορφή τεράστιων γράφων, αυξάνει εκθετικά τον χρόνο εκτέλεσης των αλγορίθμων. Για το λόγο αυτό έχουν παρουσιασθεί πολλές παράλληλες υλοποιήσεις, σε hardware και σε software, για πολλούς από τους αλγορίθμους του graph mining.

Η παρούσα διπλωματική εργασία είναι η πρώτη στην οποία ένας από τους πιο αποδοτικούς αλγορίθμους στο πεδίο του frequent subgraph mining (gSpan αλγόριθμος) απεικονίζεται σε ένα σύστημα αναδιατασσόμενης λογικής, όπως είναι η FPGA. Επίσης, στην εργασία αυτή αναπτύσσεται μία νέα δομή δεδομένων κατάλληλη για την απεικόνιση και την επεξεργασία ενός γράφου σε αναδιατασσόμενη λογική και γίνεται σύγκριση απέναντι στις άλλες «παραδοσιακές» τεχνικές αναπαράστασης γράφων. Η αρχιτεκτονική, που σχεδιάστηκε πετυχαίνει επιδόσεις, οι οποίες ξεπερνούν τις επιδόσεις της αυθεντικής έκδοσης του αλγορίθμου, κατά μία τάξη μεγέθους.
Συνεπώς, αναδεικνύεται η καταλληλόλητα της χρήσης των FPGAs, σε τόσο πολύπλοκα προβλήματα.

ΚΕΦΑΛΑΙΟ 1: Εισαγωγή

1.1 Γενικά

Στο πρώτο κεφάλαιο αυτής της διπλωματικής εργασίας γίνεται εισαγωγή στο επιστημονικό πεδίο της εξόρυξης δεδομένων, γνωστό ως Data Mining. Τα τελευταία χρόνια η περιοχή του data mining έχει κεντρίσει το ενδιαφέρον της βιομηχανίας της πληροφορίας και της επιστημονικής κοινότητας, εξαιτίας της ύπαρξης μεγάλου όγκου δεδομένων σε πολλές εφαρμογές και της ανάγκης μετατροπής αυτών των δεδομένων σε χρήσιμη πληροφορία και γνώση. Τα αποτελέσματα αυτά, μπορούν να χρησιμοποιηθούν σε πολλές εφαρμογές όπως είναι η διαδικασία λήψης αποφάσεων, ο έλεγχος της παραγωγής, η διαχείριση χρήσιμης πληροφορίας και η επεξεργασία των queries σε μία βάση δεδομένων. Για το λόγο αυτό, το data mining, θεωρείται ως ένα πολλά υποσχόμενο “εργαλείο”, στην επιστήμη της πληροφορικής.

Στη βιβλιογραφία, το data mining περιγράφεται με τον εξής ορισμό: «Η σύνθετη διαδικασία εξαγωγής συγκεκριμένης, προηγουμένως άγνωστης και δυνητικά ωφέλιμης, γνώσης από δεδομένα»[1]. Εναλλακτικά, συναντάται και ως «η επιστήμη της εξόρυξης χρήσιμης πληροφορίας από σύνολα ή βάσεις δεδομένων μεγάλου μεγέθους»[2]. Πιο απλά, το data mining μπορεί να οριστεί ως η διαδικασία που περιλαμβάνει την αναζήτηση, τη συλλογή, την επεξεργασία και την ανάλυση των δεδομένων. Είναι σημαντικό να ξεκαθαριστεί ότι ο ορισμός αυτός δεν είναι ευρέως αποδεκτός, ωστόσο περιγράφει εύστοχα την όλη διαδικασία. Πολλές ιστοσελίδες και βάσεις δεδομένων μπορεί να περιλαμβάνουν τεράστια ποσότητα από δεδομένα. Η πληροφορία των δεδομένων, μπορεί να έχει πολλές μορφές, όπως σχέσεις δεδομένων(data relationships), συσχετίσεις(co-relation) και πρότυπα(patterns). Με την εμφάνιση του διαδικτύου, των supercomputers και των μεγάλων βάσεων δεδομένων, ξεκίνησε να συσσωρεύεται μεγάλος όγκος δεδομένων. Τα δεδομένα αυτά μπορεί να χρειαστεί να αναλύονται συνεχώς, με σκοπό τον προσδιορισμό της σχέσης τους και την αναζήτηση λύσεων σε γνωστά προβλήματα. Τα τελευταία χρόνια, έχουν αναπτυχθεί αρκετές εφαρμογές, όπως είναι το Facebook και το LinkedIn, στις οποίες καθημερινά δημιουργείται ένας πολύ μεγάλος όγκος νέων δεδομένων[1]. Για την αναπαράσταση όλων αυτών των δεδομένων έχει χρησιμοποιηθεί η δομή των γράφων. Το μέγεθος αυτών των γράφων υπολογίζεται σε μεγέθη των PetBytes.

Πολλές κυβερνήσεις, γνωστές εταιρείες, μεγάλοι οργανισμοί και αρκετές επιχειρήσεις, συλλέγουν μεγάλη ποσότητα δεδομένων, με σκοπό την επιχειρησιακή και επιστημονική ανάπτυξη. Ωστόσο, πολλές φορές τα δεδομένα συλλέγονται για μελλοντική χρήση. Σε αυτή την περίπτωση, χρειάζεται να γίνει η αποθήκευση τους. Αυτή η διαδικασία είναι πολύ σημαντική, όταν απαιτείται η μελλοντική χρήση των δεδομένων. Επίσης είναι πολύ σημαντικό, ότι μπορεί να χρειαστεί μεγάλο χρονικό διάστημα για την αναζήτηση και την εύρεση χρήσιμης πληροφορίας από ιστοσελίδες, βάσεις δεδομένων και άλλες διαδικτυακές πηγές. Υπάρχουν πολλές εφαρμογές, οι οποίες εξυπηρετούνται από τις υπηρεσίες που παρέχει το data mining. Ορισμένες από αυτές είναι:

- **Έρευνα και μελέτες.** Το data mining μπορεί να χρησιμοποιηθεί για την έρευνα προϊόντων και για την αναζήτηση και την ανάλυση της αγοράς. Με αυτό τον τρόπο, μπορούν να προκύψουν χρήσιμες πληροφορίες, οι οποίες θα οδηγήσουν με τη σειρά τους στην επιτυχημένη προώθηση του προϊόντος στην αγορά.
- **Συλλογή πληροφορίας.** Με τη βοήθεια της διαδικασίας web scraping καθίσταται δυνατή η συλλογή χρήσιμης πληροφορίας αναφορικά με επενδύσεις, επενδυτές και κεφάλαια, μέσα από σχετικές ιστοσελίδες και βάσεις δεδομένων.
- **Γνώμες πελατών.** Οι γνώμες και οι προτάσεις των πελατών, παίζουν καθοριστικό ρόλο στον τρόπο με τον οποίο λειτουργεί μία εταιρεία. Οι πληροφορίες αυτές μπορούν εύκολα να συλλεχθούν μέσα από forums, blogs και άλλες διαδικτυακές πηγές, στις οποίες οι πελάτες εκφράζουν ελεύθερα τη γνώμη τους.

- **Σάρωση δεδομένων.** Η συλλογή και η αποθήκευση των δεδομένων δεν θα είχαν μεγάλη σημασία, αν δεν υπήρχε η λειτουργία της σάρωσης των δεδομένων. Η λειτουργία αυτή είναι χρήσιμη για την αναγνώριση προτύπων και ομοιοτήτων μεταξύ των δεδομένων.
- **Εξαγωγή της πληροφορίας.** Αυτό περιλαμβάνει την επεξεργασία της αναγνώρισης χρήσιμων προτύπων μέσα στα δεδομένα, η οποία μπορεί να χρησιμοποιηθεί στη διαδικασία λήψης μία απόφασης. Αυτό συμβαίνει επειδή η διαδικασία αυτή πρέπει να οφείλεται σε αξιόπιστα στοιχεία και γεγονότα.
- **Προεπεξεργασία δεδομένων.** Τα δεδομένα που συλλέγονται συνήθως αποθηκεύονται σε ειδικές «αποθήκες» δεδομένων, γνωστές ως data warehouse. Πολλές φορές τα δεδομένα αυτά χρειάζεται να περάσουν από ένα είδος προεπεξεργασίας. Ο όρος προεπεξεργασία σημαίνει ότι κάποια δεδομένα που μπορούν να θεωρούνται ασήμαντα, μπορούν να αφαιρεθούν.
- **Ανάλυση ανταγωνιστών.** Συχνά υπάρχει η ανάγκη μίας επιχείρησης, να κατανοήσει τη συμπεριφορά των ανταγωνιστών της μέσα στην αγορά, προκειμένου να παραμείνει και η ίδια ανταγωνιστική. Για το λόγο αυτό, χρειάζεται να γνωρίζει τις αδυναμίες και τις δυνατότητες τους. Οι τεχνικές της προώθησης και της διανομής μπορούν να συλλεχθούν. Επίσης, εξίσου σημαντικός είναι και ο τρόπος με τον οποίο οι ανταγωνιστές μειώνουν το συνολικό κόστος της επιχείρησης τους.
- **Online αναζήτηση.** Το διαδίκτυο είναι ευρέως γνωστό για το μεγάλο όγκο πληροφοριών που παρέχει. Έτσι, καθίσταται δυνατή η συγκέντρωση πληροφορίων, σχετικά με διάφορες εταιρείες και πελάτες. Με αυτό τον τρόπο, μπορεί να ανιχνευθούν απάτες, μόνο με τη χρησιμοποίηση του διαδικτύου.
- **Ενημέρωση δεδομένων.** Η συλλογή δεδομένων μπορεί να είναι μία άχρηστη διαδικασία, αν τα δεδομένα δεν είναι ενημερωμένα. Αυτό γίνεται για να εξασφαλιστεί ότι οι πληροφορίες είναι συναφείς, έτσι ώστε να ληφθούν σωστές αποφάσεις από αυτές.

Μία ειδική περίπτωση του data mining είναι το graph mining. Αντικείμενο αυτού του πεδίου είναι η εξόρυξη χρήσιμης πληροφορίας από δεδομένα, που χρησιμοποιούν ως βασική δομή δεδομένων τους γράφους. Οι γράφοι από τη φύση τους έχουν ιδιότητες, οι οποίες συχνά οδηγούν σε υψηλή αλγοριθμική πολυπλοκότητα. Παρ' όλα αυτά, έχουν αναπτυχθεί αρκετές τεχνικές, όπως είναι η εξόρυξη συχνών μοτίβων (frequent pattern mining**)**, η ομαδοποίηση των δεδομένων (clustering**)** και η ταξινόμησή τους (classification**)**, οι οποίες καθιστούν τους γράφους ως μία κατάλληλη δομή δεδομένων για την επίλυση γνωστών προβλημάτων της περιοχής του data mining.

1.1.1 Graph Frequent Pattern Mining

Το πρόβλημα του frequent pattern mining είναι ένα γνωστό πρόβλημα στην περιοχή του data mining. Όπως αναφέρθηκε και πιο πάνω, οι γράφοι είναι μία ιδιαίτερα πολύπλοκη δομή δεδομένων. Η χρησιμοποίηση τους στο συγκεκριμένο πρόβλημα, αλλάζει λίγο τη διαδικασία μελέτης του ελάχιστου ορίου συχνότητας, που ονομάζεται support. Το πρόβλημα μπορεί να οριστεί με διαφορετικούς τρόπου ανάλογα με την εφαρμογή στην οποία απευθύνεται. Στην πρώτη περίπτωση, υπάρχει μία ομάδα από γράφους, στην οποία επιθυμείται η εξόρυξη όλων των patterns, τα οποία περνούν το όριο της συχνότητας των αντίστοιχων γράφων [3,4,5]. Στη δεύτερη περίπτωση, υπάρχει ένας μεγάλος γράφος και επιθυμείται η εξόρυξη εκείνων των pattern, που συμβαίνουν ένας συγκεκριμένος αριθμός από Ν φορές μέσα στον ίδιο γράφο [5,6,7]. Και στις δύο περιπτώσεις χρειάζεται να γίνει έλεγχος για τον ισομορφισμό, προκειμένου να μην εξεταστούν πολλαπλές φορές όμοια patterns. Ο συγκεκριμένος έλεγχος αποτελεί ένα κρίσιμο σημείο του συνολικού προβλήματος, αν επιτρέπονται οι επικαλύψεις μεταξύ όμοιων και διαφορετικών patterns μέσα στο σύνολο των εξεταζόμενων γράφων.

1.1.2 Graph Clustering

Σε αυτό το σημείο θα γίνει μία αναφορά σε αλγορίθμους της περιοχής της «ομαδοποίησης» ή αλλιώς clustering των δεδομένων μέσα σε γράφους. Αυτό περιλαμβάνει τους κλασικούς αλγορίθμους για graph clustering, όπως και τους αλγορίθμους για clustering XML data. Οι αλγόριθμοι αυτοί χρησιμοποιούνται σε πολλές εφαρμογές, όπως είναι η κυκλοφοριακή συμφόρηση, η χωροθέτηση εγκαταστάσεων και η ολοκλήρωση XML δεδομένων[8]. Με τη χρησιμοποίηση των γράφων, η περιοχή αυτή χωρίζεται σε δύο είδη αλγορίθμων. Το πρώτο είδος είναι οι Node Clustering αλγόριθμοι, στους οποίους υπάρχει ένας μεγάλος γράφος και επιχειρείται η ομαδοποίηση των βασικών κόμβων, με κριτήριο την απόσταση ή την ομοιότητα των τιμών που έχουν οι ακμές. Στην περίπτωση αυτή οι ακμές έχουν ως βάρος διάφορες αριθμητικές τιμές που συμβολίζουν την αντίστοιχη απόσταση των κόμβων, μέσα στο γράφο. Με αυτόν τον τρόπο δημιουργούνται ομάδες των κόμβων, γνωστές και ως clusters. Μία ιδιαίτερη περίπτωση είναι εκείνη κατά την οποία η παρουσία μίας ακμής μέσα στο cluster αναφέρεται σε μια τιμή ομοιότητας του 1, ενώ η απουσία μιας ακμής αναφέρεται σε μια τιμή ομοιότητας του 0. Πιο απλά μία ακμή μπορεί να υπάρχει ανάμεσα σε οποιοδήποτε ζεύγος από κόμβους με μεγάλη πιθανότητα. Σε αυτό το πρόβλημα, εξετάζονται ομάδες κόμβων, οι οποίες αναφέρονται με τον όρο «almost cliques». Το δεύτερο είδος είναι οι Graph Clustering αλγόριθμοι, στους οποίους υπάρχει ένας μεγάλος αριθμός από γράφους και εξετάζεται η ομαδοποίησή τους, με κριτήριο τη συμπεριφορά της δομής τους. Η χρησιμοποίηση της ομοιότητας μεταξύ των δομών κάθε γράφου ως κριτήριο για την ομαδοποίηση όλων των γράφων, αναδεικνύει από μόνη της, την πολυπλοκότητα του συγκεκριμένου προβλήματος.

1.1.3 Graph Classification

Το classification αποτελεί ένα σημαντικό εργαλείο, στον τομέα της εξόρυξης δεδομένων και τη μηχανική μάθηση. Οι γράφοι χρησιμοποιούνται για την αναπαράσταση των οντοτήτων και της σχέσης μεταξύ αυτών, σε ένα μεγάλο εύρος εφαρμογών του επιστημονικού και του βιομηχανικού τομέα. Για παράδειγμα, στη φαρμακευτική και το σχεδιασμό των φαρμάκων, χρειάζεται να γνωρίζουμε τη σχέση μεταξύ της δραστηριότητας μιας χημικής ένωσης και τη δομή της ένωσης, η οποία αντιπροσωπεύεται από ένα γράφο. Στην ανάλυση των κοινωνικών δικτύων, γίνεται μελέτη για τη σχέση μεταξύ της υγείας της κοινότητας (π.χ. αν επεκτείνεται ή συρρικνώνεται) και τη δομή της, η οποία για μία ακόμη φορά αναπαρίσταται από ένα γράφο. Το Graph classification περιλαμβάνει δύο διαφορετικές προσεγγίσεις, οι οποίες παρουσιάζουν μερική ομοιότητα. Η πρώτη ονομάζεται Label propagation και περιλαμβάνει ένα σύνολο από κόμβους μέσα στο γράφο, οι οποίοι είναι ονομασμένοι με συγκεκριμένες ετικέτες(labels). Σκοπός της είναι η εξαγωγή ενός μοντέλου από τους ονομασμένους κόμβους, με βάση το οποίο θα ταξινομηθούν όλοι οι υπόλοιποι κόμβοι του γράφου που δεν έχουν ονομαστεί. Η δεύτερη προσέγγιση ονομάζεται Graph classification και περιλαμβάνει ένα σύνολο γράφων, οι οποίοι είναι ονομασμένοι με συγκεκριμένες ετικέτες(labels). Σκοπός της είναι η εξαγωγή ενός μοντέλου από τους ονομασμένους γράφους, με βάση το οποίο θα ταξινομηθούν όλοι οι υπόλοιποι γράφοι που δεν έχουν ονομαστεί.

1.2 Επιστημονική συνεισφορά

Στην παρούσα διπλωματική εργασία, έγινε η μελέτη ενός από τους πιο αποδοτικούς και διαδεδομένους αλγορίθμους στην επιστημονική κοινότητα, για το πρόβλημα της εξόρυξης συχνών υπογράφων μέσα από ένα σύνολο πολλών γράφων ή ενός μεγάλου γράφου, όπως είναι ο αλγόριθμος gSpan[9]. Το πρόβλημα αυτό είναι γνωστό ως Frequent Subgraph Mining και αποτελεί ένα από τα προβλήματα της πρώτης κατηγορίας που αναφέρθηκαν στην προηγούμενη ενότητα.

Ο βασικός στόχος της εργασίας, είναι η ανάδειξη της «παραλληλίας», που παρουσιάζουν αυτά τα προβλήματα και η αξιοποίησή της, με τη σχεδίαση κατάλληλης αρχιτεκτονικής υλικού, η οποία επιταχύνει την εκτέλεση του αλγορίθμου κατά ένα σημαντικό παράγοντα. Η αποτύπωση της αρχιτεκτονικής έγινε σε μία multiple-FPGA πλατφόρμα, με την χρήση ενός συνεπεξεργαστή.

Σε αυτά τα προβλήματα, η υψηλή πολυπλοκότητα καθώς και η ύπαρξη δεδομένων σε μορφή τεράστιων γράφων, αυξάνει εκθετικά τον χρόνο εκτέλεσης των αλγορίθμων. Για το λόγο αυτό έχουν παρουσιασθεί πολλές παράλληλες υλοποιήσεις, είτε σε hardware είτε σε software, για πολλούς από τους αλγορίθμους του graph mining. Η συνεισφορά της συγκεκριμένης εργασίας στο πρόβλημα του graph mining είναι η εξής:

- ✓ Η συγκεκριμένη εργασία είναι η πρώτη στην οποία ένας από τους πιο αποδοτικούς αλγορίθμους στο πεδίο του frequent subgraph mining (gSpan αλγόριθμος) απεικονίζεται σε ένα σύστημα αναδιατασσόμενης λογικής, όπως είναι η FPGA.
- ✓ Επίσης, στην εργασία αυτή αναπτύσσεται μία νέα δομή δεδομένων κατάλληλη για την απεικόνιση και την επεξεργασία ενός γράφου σε αναδιατασσόμενη λογική και γίνεται σύγκριση απέναντι στις άλλες «παραδοσιακές» τεχνικές αναπαράστασης γράφων.
- ✓ Τέλος, παρουσιάζεται η απόδοση της αρχιτεκτονικής του αλγορίθμου gSpan, που σχεδιάστηκε σε ένα σύστημα με πολλαπλές FPGAs απέναντι στις επιδόσεις της αυθεντικής έκδοσης του αλγορίθμου.

1.3 Δομή της διατριβής

Το κείμενο που ακολουθεί, χωρίζεται στις παρακάτω ενότητες:

- ◆ Στο **κεφάλαιο 2** της παρούσας εργασίας, αναλύεται το θεωρητικό υπόβαθρο για την περιοχή του Graph Mining. Επίσης, παρουσιάζονται διάφορες παράλληλες υλοποιήσεις στην περιοχή του Graph Mining.
- ◆ Στο **κεφάλαιο 3**, περιγράφεται ο αλγόριθμος gSpan καθώς και το τμήμα του κώδικα που σχεδιάστηκε και υλοποιήθηκε σε αναδιατασσόμενη λογική.
- ◆ Στο **κεφάλαιο 4**, παρουσιάζεται η προτεινόμενη αρχιτεκτονική και γίνεται ανάλυση των επιμέρους υποσυστημάτων που υλοποιήθηκαν, εξηγώντας ταυτόχρονα και τα κριτήρια με τα οποία εκπονήθηκε η συγκεκριμένη αρχιτεκτονική.
- ◆ Στο **κεφάλαιο 5**, παρατίθενται πληροφορίες για την απόδοση της αρχιτεκτονικής υλοποιημένης σε multiple-FPGA πλατφόρμα και γίνεται σύγκριση με τους χρόνους εκτέλεσης που έχουν καταμετρηθεί σε προηγούμενες παράλληλες υλοποιήσεις καθώς και στην πρωτότυπη υλοποίηση του αλγορίθμου σε software.
- ◆ Τέλος, στο **κεφάλαιο 6**, αναφέρονται τα συμπεράσματα από τη συγκεκριμένη διατριβή και προτείνονται ιδέες για μελλοντική επέκταση ή και βελτιώσεις της υπάρχουσας αρχιτεκτονικής.

ΚΕΦΑΛΑΙΟ 2: Σχετικές εργασίες

2.1 Γενικά

Στο κεφάλαιο αυτό περιγράφεται όλο το θεωρητικό υπόβαθρο για την περιοχή του frequent subgraph mining προβλήματος. Επιπλέον, γίνεται μία αναφορά στους αλγορίθμους που έχουν αναπτυχθεί για το πρόβλημα αυτό καθώς επίσης και στις υπάρχουσες παράλληλες υλοποιήσεις και μελέτες που έχουν παρουσιασθεί μέχρι σήμερα.

2.2 Frequent Subgraph Mining

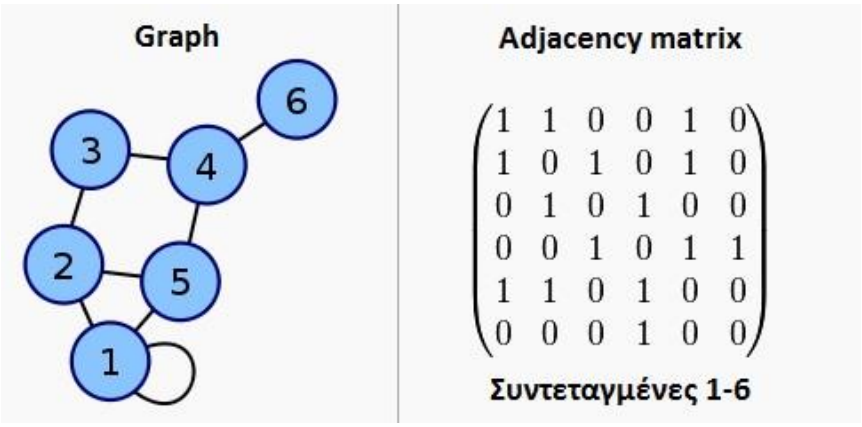
Το graph mining θεωρείται σημαντικό πρόβλημα καθώς χρησιμοποιείται, τόσο σε καθημερινές εφαρμογές, όσο και σε επιστημονικές περιοχές. Ένα από τα προβλήματα αυτής της περιοχής είναι το Frequent Subgraph Mining, το οποίο γίνεται σύμφωνα με ένα κριτήριο, που καθορίζεται από τον χρήστη και είναι η συχνότητα εμφάνισης του συγκεκριμένου υπογράφου μέσα στο αρχικό data set. Η παράμετρος αυτή ονομάζεται κατώφλι συχνότητας και είναι καθοριστική για τον χρόνο εκτέλεσης όλης της διαδικασίας. Στην ουσία αποτελεί ένα ποσοστό συχνότητας(%), που προσδιορίζει ότι κάθε υποψήφιος υπογράφος πρέπει να παρουσιάζεται σε ένα συγκεκριμένο αριθμό γράφων, από το συνολικό αριθμό των γράφων που δόθηκαν στην είσοδο.

Κάθε αλγόριθμος που έχει αναπτυχθεί στην περιοχή του Frequent Subgraph Mining, καθορίζεται από τρεις πτυχές, i) τον τρόπο αναπαράστασης των γράφων, ii) την απαρίθμηση όλων των υπογράφων του και iii) το μέτρημα της συχνότητας εμφάνισης τους.

2.2.1 Αναπαράσταση γράφων

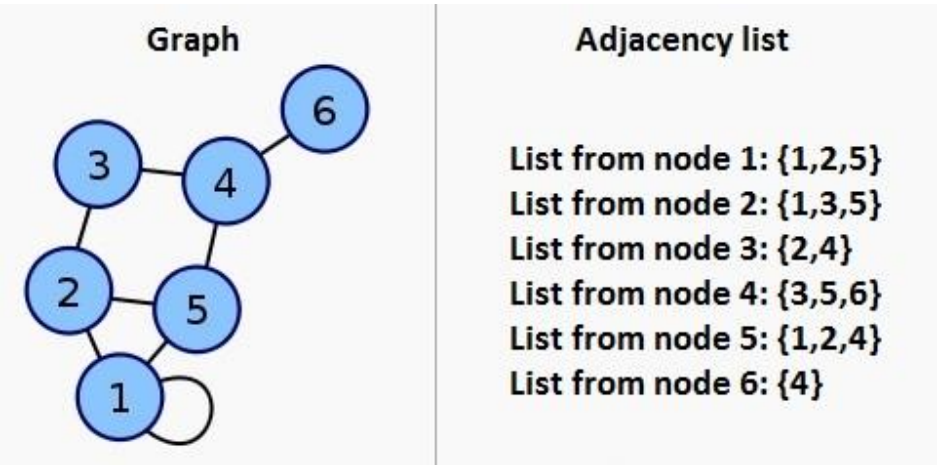
Η συνηθισμένη αναπαράσταση ενός γράφου γίνεται με δύο τρόπους, είτε με adjacency matrix, ή με adjacency list.

Στην πρώτη μέθοδο αναπαράστασης, ο adjacency matrix είναι ένας πίνακας διάστασης nxn, όπου n ο συνολικός αριθμός των κόμβων μέσα στο γράφο. Αυτός ο πίνακας περιέχει σε κάθε θέση του, την πληροφορία αν υπάρχει σύνδεση μεταξύ των δύο κόμβων του γράφου. Συνεπώς, τα στοιχεία πάνω στην κύρια διαγώνιο, δηλώνουν αν ένας κόμβος συνδέεται με τον εαυτό του. Ένα παράδειγμα φαίνεται στην εικόνα 2.2.1.



Εικόνα 2.2.1: Απεικόνιση ενός γράφου με τη χρήση μίας adjacency matrix δομής
(*Πηγή: http://en.wikipedia.org/wiki/Adjacency_matrix*)

Στη δεύτερη μέθοδο αναπαράστασης, για κάθε κόμβο δημιουργείται μία λίστα η οποία κρατάει το σύνολο των κόμβων με τους οποίους συνδέεται κάθε κόμβος μέσα στο γράφο. Η λίστα αυτή μπορεί να έχει μέγιστο μέγεθος, όσο είναι η τάξη του γράφου. Πιο συγκεκριμένα, ένα παράδειγμα φαίνεται στην εικόνα 2.2.2.



Εικόνα 2.2.2: Απεικόνιση ενός γράφου με τη χρήση μίας adjacency list δομής
(*Πηγή*: *http://en.wikipedia.org/wiki/Adjacency_list*)

2.2.2 Απαρίθμηση γράφων - Ισομορφισμός

Κατά τη διαδικασία της απαρίθμησης των υπογράφων, ένα εμπόδιο είναι η ανάπτυξη του ίδιου γράφου από πολλές συχνές ακμές. Αν γίνει η επεξεργασία και για όλους τους γράφους, τότε όχι μόνο θα υπάρξουν πολλοί ίδιοι γράφοι σαν έξοδο, αλλά και ο χρόνος επεξεργασίας όλων των δεδομένων εισόδου θα είναι σημαντικά μεγαλύτερος. Το πρόβλημα αυτό είναι γνωστό ως ισομορφισμός των γράφων. Για την αντιμετώπισή αυτού του προβλήματος, κάθε αλγόριθμος επιχειρεί έναν έλεγχο ισομορφισμού έπειτα από κάθε προέκταση που επιχειρεί. Ωστόσο, προκειμένου να αποφευχθεί αυτός ο έλεγχος αναπτύχθηκε μία μέθοδος, η οποία είναι γνωστή ως Canonical Labeling System. Η τεχνική αυτή περιορίζει την πιθανότητα της πολλαπλής επεξεργασίας όμοιων γράφων κατά τον έλεγχο του ισομορφισμού, καθώς αν ένα ζεύγος γράφων είναι ισομορφικό, τότε τα CLS τους θα είναι πανομοιότυπα [10]. Το σύστημα αυτό το χρησιμοποιεί ο αλγόριθμος gSpan[9], στον οποίο προεκτείνονται μόνο όσοι γράφοι έχουν το ελάχιστο Canonical Label κωδικό, τον οποίο ονομάζει DFS CODE.

Η απαρίθμηση των υπογράφων, μπορεί να γίνει με δύο τρόπους, είτε με τη μέθοδο της ένωσης (join operation), ή με τη μέθοδο της προέκτασης (extension operation). Στην πρώτη μέθοδο, μία και μόνο ένωση παράγει πολλαπλούς υποψήφιους γράφους και ο κάθε ένας υποψήφιος γράφος μπορεί να προκύπτει από πολλαπλές ενώσεις. Ενώ στη δεύτερη μέθοδο, ο αριθμός των κόμβων, που μία νέα ακμή μπορεί να επισυνάψει περιορίζεται σε μεγάλο βαθμό.

2.2.3 Συχνότητα εμφάνισης γράφων

Η πληροφορία για τη συχνότητα εμφάνισης των υπογράφων υπολογίζεται με τη χρήση δύο δομών των embedding lists και των recomputed embeddings. Η πρώτη δομή απεικονίζει σε πιο σημείο μέσα στη βάση δεδομένων, βρίσκεται κάθε υπογράφος και επιτρέπει τη γρήγορη αναζήτηση των υποψήφιων παιδιών του. Όμως, η επεξεργασία των παιδιών ενός πατέρα πρέπει να έχει ολοκληρωθεί, πριν ο ίδιος ο πατέρας ελευθερωθεί. Ο περιορισμός αυτός απαιτεί μεγάλο αποθηκευτικό χώρο. Η δεύτερη δομή διατηρεί ένα σύνολο από ενεργούς υπογράφους. Για κάθε έναν από αυτούς επαναπροσδιορίζει επαναληπτικά τις συχνότητες εμφάνισης τους[10].

2.3 Κατηγορίες Frequent Subgraph Mining αλγορίθμων

Για την επίλυση του frequent subgraph mining προβλήματος, έχουν αναπτυχθεί πολλοί αλγόριθμοι βασισμένοι σε διαφορετικές ευριστικές μεθόδους, ανάλογα με την εφαρμογή για την οποία χρησιμοποιούνται. Οι αλγόριθμοι αυτοί μπορούν να χωριστούν σε τρεις κατηγορίες με βάση τη μέθοδο που χρησιμοποιούν για να ξεπεράσουν το πρόβλημα του ισομορφισμού των υπογράφων[10]. Οι κατηγορίες αυτές είναι η Graph theory based, η Inductive Logging Programming και η Greedy search[11,12].

2.3.1 Graph theory based αλγόριθμοι

Η πρώτη προσέγγιση (GT) ερευνά ένα σύνολο υπογράφων, είτε με τη χρήση μίας παραμέτρου που δίνεται από το χρήστη και καλείται support, είτε με την μέτρηση “συχνών” υπογράφων. “Συχνός” χαρακτηρίζεται ο υπογράφος, ο οποίος εμφανίζεται με συχνότητα μεγαλύτερη από την παράμετρο support που έχει δώσει ο χρήστης. Η παράμετρος αυτή χρησιμοποιείται ως ένα κατώφλι (support threshold). Σε αυτή την προσέγγιση, η διαδικασία της εξόρυξης συχνών υπογράφων αποτελείται από δύο βήματα. Αρχικά δημιουργούνται οι υποψήφιοι συχνοί υπογράφοι και στη συνέχεια ελέγχεται η συχνότητα καθενός από αυτούς.

Οι αλγόριθμοι, οι οποίοι ακολουθούν την προσέγγιση Graph theory based, μπορούν να χωριστούν σε δύο κατηγορίες τους apriori αλγορίθμους και τους pattern growth αλγορίθμους.

Στους apriori αλγορίθμους οι υπογράφοι με αριθμό κόμβων k+1 προκύπτουν από τους υπογράφους με αριθμό κόμβων k. Αυτοί οι αλγόριθμοι χρησιμοποιούν τη Breadth First Search (BFS) στρατηγική αναζήτησης, μιας και οι υποψήφιοι υπογράφοι δημιουργούνται με ένα πολυεπίπεδο τρόπο. Για να δημιουργηθεί ένας υποψήφιος υπογράφος με μέγεθος k+1, δύο ή περισσότεροι συχνοί υπογράφοι μεγέθους k συγχωνεύονται. Οι αλγόριθμοι που ακολουθούν αυτή την προσέγγιση είναι οι AGM[4], FSG[5], ο AcGM [13], ο FFSM[14], ο SPIN[15] και ο PATH[3]. Το βασικό μειονέκτημα, όμως αυτών των αλγορίθμων είναι ότι η διαδικασία της σύστασης ενός νέου μεγαλύτερου υπογράφου από δύο συχνούς μικρότερους υπογράφους είναι μία πολύπλοκη και ακριβή διαδικασία. Γι’αυτό το λόγο, η κοινότητα δεν ασχολείται πλέον, σε μεγάλο βαθμό με τους apriori αλγόριθμους.

Σε αντίθεση με τους apriori αλγορίθμους, οι pattern growth αλγόριθμοι προεκτείνουν ένα υποψήφιο υπογράφο με την προσθήκη μίας νέας ακμής σε κάθε δυνατή θέση. Οι αλγόριθμοι αυτής της μεθόδου είναι ο MoFA[16], ο gSpan[9] (2002) και ο Gaston[17]. Όλοι αυτοί οι αλγόριθμοι παρουσιάζουν τον κίνδυνο της εξέτασης των ίδιων γράφων πολλές φορές (πρόβλημα ισομορφισμού), το οποίο αντιμετωπίζουν με διαφορετικό τρόπο ο καθένας.

2.3.2 Inductive Loging Programming αλγόριθμοι

Αυτή η κατηγορία των αλγορίθμων, προϋποθέτει το συνδυασμό από πολλά παραδείγματα και τη συγκέντρωση ενός υπόβαθρου γνώσης. Η βασική λειτουργία στην ILP[18] μέθοδο είναι η χρήση της λογικής για την αναπαράσταση και η χρήση της γνώσης του υπόβαθρου, για την κατασκευή συντακτικά ορθών υποθέσεων. Δεν υπάρχουν πολλοί αλγόριθμοι από αυτή την κατηγορία. Οι πιο γνωστός είναι ο WARMR, ο οποίος στη συνέχεια εξελίχθηκε στον αλγόριθμο FARMER[19], που περιλαμβάνει στοιχεία και από την κατηγορία του ILP και από την apriori.

2.3.3 Greedy search αλγόριθμοι

Σε αυτήν την προσέγγιση, σε κάθε στάδιο λαμβάνεται μια απόφαση, που φαίνεται να είναι η καλύτερη σε εκείνη τη χρονική στιγμή. Η απόφαση που θα ληφθεί σε ένα στάδιο δεν μπορεί να μεταβληθεί σε μεταγενέστερα στάδια και ως εκ τούτου, κάθε απόφαση θα πρέπει να διασφαλίζεται ότι είναι η βέλτιστη δυνατή. Από αυτήν την κατηγορία αλγορίθμων ξεχωρίζουν δύο αλγόριθμοι, ο SUBDUE[20] από τον Cook και ο GBI[12] από τον Matsuda. Και οι δύο αλγόριθμοι χρησιμοποιούν πολύ ισχυρά κριτήρια για την εξαγωγή των δομών του γράφου, όπως είναι η εντροπία της πληροφορίας και το gini-index[13]. Ο

αλγόριθμος SUBDUE αναζητάει ένα υπογράφο, ο οποίος μπορεί να συμπιέσει σε μεγάλο βαθμό τον γράφο G, βασισμένος στη θεωρία του MDL[21]. Το MDL είναι μία τεχνική στην οποία η καλύτερη υπόθεση για ένα δοθέντα γράφο, είναι εκείνη που οδηγεί στην καλύτερη συμπίεση των δεδομένων. Συνήθως, χρησιμοποιείται software σε γλώσσα προγραμματισμού, για να καθοριστεί η συμπίεση των δεδομένων.

2.4 Παράλληλες υλοποιήσεις Frequent Subgraph Mining προβλήματος

Η φύση του προβλήματος της εξόρυξης των συχνών υπογράφων, οδηγεί στη μελέτη και υλοποίηση παράλληλων αλγορίθμων, δηλαδή αλγορίθμων που παρουσιάζουν μεγάλο παραλληλισμό. Η μελέτη τέτοιων υλοποιήσεων είναι σημαντική καθώς μπορεί να επιτευχθεί πολύ μεγάλη επιτάχυνση εκτέλεσης. Πιθανές κατευθύνσεις του προβλήματος αποτελούν η κατανεμημένη ή η παράλληλη αναζήτηση, με σκοπό να ξεπεραστούν τα εμπόδια στη χρήση της μνήμης και στην απόδοση[22].
Η χρησιμοποίηση κατανεμημένων μνημών (distributed memories) προσφέρει καλή απόδοση σπάζοντας τα δεδομένα σε πολλά κομμάτια. Παρ' όλα αυτά, απαιτείται τα δεδομένα του γράφου να σπάνε σε κομμάτια, το οποίο δεν είναι πάντα τόσο εύκολο σε γράφους χωρίς σταθερή δομή[23].
Από την άλλη μεριά, οι υλοποιήσεις, που χρησιμοποιούν διαμοιραζόμενες μνήμες (shared memories), χρησιμοποιούν μία καθολική μνήμη (global), η οποία είναι προσβάσιμη από όλους τους επεξεργαστές, συμπεριλαμβάνοντας τους παράλληλους υπολογιστές με cache-coherent και τα μαζικά πολυνηματικά μηχανήματα (massively multi-thread machines MMT). Στους πρώτους μπορεί να υπάρχει γρηγορότερη πρόσβαση στους γράφους, συγκριτικά με τις κατανεμημένες μνήμες, ωστόσο απαιτούνται πρωτόκολλα cache coherence για την ορθή λειτουργία τους. Από την άλλη μεριά τα MMT ανέχονται το latency της μνήμης, παρέχοντας υποστήριξη για πολλά ταυτόχρονα νήματα, τα οποία με τη σειρά τους μπορούν να ζητήσουν πολλαπλά αιτήματα προς την μνήμη [23].
Στη συνέχεια θα κάνουμε μία αναφορά σε υπάρχουσες εργασίες σχετικές με το πρόβλημα του Frequent Subgraph Mining που μελετήθηκε σε αυτή την διπλωματική εργασία, καθώς και τα αποτελέσματα που πέτυχε κάθε μία από αυτές τις υλοποιήσεις. Τις δουλειές αυτές θα τις διακρίνουμε σε δύο ενότητες, ανάλογα με την πλατφόρμα στην οποία αναπτύχθηκαν. Η πρώτη αναφέρει, όλες τις παράλληλες υλοποιήσεις σε software, όπως είναι τα συστήματα με πολλαπλούς επεξεργαστές(multiprocessor systems), ενώ η δεύτερη αναφέρει όλες τις δουλειές που έχουν υλοποιηθεί σε πλατφόρμες hardware, όπως είναι οι FPGAs ή οι GPUs.

2.4.1 Software-based υλοποιήσεις

Σε αυτή την παράγραφο θα παρουσιασθούν κάποιες υπάρχουσες δουλειές, οι οποίες υλοποίησαν κάποιους από τους προαναφερθέντες αλγορίθμους σε πολυπύρηνες πλατφόρμες ή clusters, χρησιμοποιώντας frameworks όπως το openMP[28], το MPI[32] και το MapReduce[27,31,35].
Στις πρώτες προσπάθειες για την αξιοποίηση του παραλληλισμού, τον οποίο διαθέτει το πρόβλημα του frequent subgraph mining, χρησιμοποιήθηκαν clusters με πολυεπεξεργαστές (multiprocessor platforms).
Οι Di Fatta και Berthold [24], σε μία από τις πρώτες παράλληλες υλοποιήσεις για το πρόβλημα της εξόρυξης συχνών υπογράφων, παρουσίασαν τα πλεονεκτήματα μίας κατανεμημένης υλοποίησης σε ένα σύστημα με 32 επεξεργαστές. Η υλοποίησή τους προσέφερε speedup πάνω από 15x, απέναντι σε μία δική τους υλοποίηση ενός single-thread αλγόριθμου εξόρυξης υπογράφων.
Οι Buehrer και Parthasarathy [25] παρουσίασαν το 2005, μία υλοποίηση ενός αλγορίθμου graph mining, στην οποία υιοθέτησαν τεχνικές από τον αλγόριθμο gSpan[9] και αξιολόγησαν κάποια σημεία περαιτέρω κατάτμησης μέσα στον αλγόριθμο καθώς και ορισμένα μοντέλα αναμονής των εργασιών. Ο αλγόριθμος υλοποιήθηκε σε γλώσσα C και τα αποτελέσματα της υλοποίησης ενός thread σε σύγκριση με την αυθεντική υλοποίηση του είναι ανταγωνιστικά. Η πλατφόρμα που χρησιμοποίησαν ήταν ένα multiprocessor system και σε κάθε processor αντιστοίχησαν ένα thread, το οποίο το ονόμασαν node. Με αυτό τον τρόπο κατάφεραν να πάρουν speedup κοντά στο 25x με τη χρήση 27 έως 32 processors, συγκρινόμενοι με την αυθεντική υλοποίηση του gSpan[9] σε ένα thread.

Την επόμενη χρονιά οι Meinl, Worlein, Fischer και Philippsen [26,27] παρουσίασαν παράλληλες υλοποιήσεις σε πολλά threads, για τους αλγόριθμους MoFa[16] και gSpan[9], πετυχαίνοντας speedup πάνω από 7x και 11x για την εκτέλεση σε ένα SMP σύστημα με 12 επεξεργαστές και χρήση κοινής μνήμης.
Με τη σειρά τους οι Reinhardt and Karypis [28] (2005) χρησιμοποίησαν το framework του openMP για να αξιοποιήσουν τον παραλληλισμό του αλγορίθμου VSIGRAM[47]. Τα αποτελέσματά τους έδειξαν speedup πάνω από 26x σε μία πλατφόρμα με 32 επεξεργαστές, σε σύγκριση με την πρώτη υλοποίηση του αλγορίθμου το 2004 σε μηχανήματα AMD Athlon MP 1800+ (1.53 GHz).
Το 2009 οι Ranu και Singh [29], παρουσίασαν μία καινούργια τεχνική για την εξόρυξη σημαντικών υπογράφων μέσα από πολύ μεγάλους γράφους. Η τεχνική τους εμφάνισε ως αποτέλεσμα γραμμικό speed-up σε σύγκριση με τις αυθεντικές υλοποιήσεις των αλγορίθμων gSpan[9] και FSG[5].
Από την άλλη μεριά, οι Bin Wu και YunLong Bai [30] παρουσίασαν μία κατανεμημένη μέθοδο για το πρόβλημα του subgraph mining χρησιμοποιώντας το MapReduce framework. Η μέθοδος αυτή πέτυχε speedup εκτέλεσης πάνω από περίπου 12x για 32 reducers. Επίσης, οι Hill, Srichandan και Sunderraman[31] χρησιμοποιώντας το MapReduce framework για το ίδιο πρόβλημα σε βιολογικά δεδομένα εισόδου πέτυχαν σχεδόν γραμμικό speedup.
Ο αλγόριθμος SUBDUE[9] των Cook και Holder είναι γνωστός στην επιστημονική κοινότητα, για τη συμπίεση υπογράφων στο πρόβλημα του frequent subgraph mining. Οι Ray και Holder[32] επεκτείνοντας την δουλειά του Cook(2001), υλοποίησαν με τη χρήση του MPI framework, μία παράλληλη έκδοση του SUBDUE αλγορίθμου, την οποία κάλεσαν SP-SUBDUE. Στη συνέχεια μετά από βελτιώσεις στην παράλληλη υλοποίησή τους, εξέλιξαν την υλοποίηση αυτή και κατέληξαν σε μία δεύτερη έκδοση την οποία ονόμασαν SP-SUBDUE-2. Το speed-up που παρουσιάζει ο SP-SUBDUE-2 σε σύγκριση με τον SP-SUBDUE είναι super-linear, επειδή ο χρόνος εκτέλεσης του αλγορίθμου SUBDUE[20] δεν είναι γραμμικός σε σύγκριση με το μέγεθος του γράφου. Όσο κάθε core τρέχει τον SUBDUE[20] για μικρούς γράφους, τόσο ο χρόνος εκτέλεσης του SP-SUBDUE σε σύγκριση με τον SUBDUE[20] θα μειώνεται.
Τέλος, το καλοκαίρι του 2013 οι Bhuiyan και Al Hasan[33], από το πανεπιστήμιο Purde της Indianapolis κατάφεραν χρησιμοποιώντας την πλατφόρμα του MapReduce να τρέξουν έναν αλγόριθμο που ονόμασαν ως MIRAGE και να έχουν σχεδόν γραμμικό speedup σε σύγκριση με την υλοποίηση που παρουσίασαν οι Hill, Srichandan και Sunderraman[31], ένα χρόνο νωρίτερα.

2.4.2 Hardware-based υλοποιήσεις

Σε αυτή την παράγραφο θα παρουσιασθούν επιστημονικές εργασίες, οι οποίες υλοποίησαν κάποιους από τους προαναφερθέντες αλγορίθμους σε τεχνολογίες hardware, όπως είναι οι GPUs και οι FPGAs. Το πρόβλημα του graph mining είναι ένα πολύ σημαντικό πρόβλημα, εξαιτίας του εκθετικού χρόνου εκτέλεσης όλων των αλγορίθμων. Για το λόγο αυτό υπάρχουν πολλές δουλειές σε μοντέρνες τεχνολογίες όπως είναι οι GPUs, οι οποίες στοχεύουν στην επιτάχυνση των αλγορίθμων αυτών.
Οι πρώτες προσπάθειες για την αξιοποίηση του παραλληλισμού των αλγορίθμων αυτής της περιοχής, έγιναν από τους Harish και Narayanan[34], χρησιμοποιώντας την Nvidia 8800GTX GPU. Στην εργασία τους υλοποίησαν δύο αλγορίθμους έναν για την BFS στρατηγική προσπέλασης πάνω στους γράφους και έναν για το πρόβλημα του Single Source Shortest Path(SSSP). Και οι δύο αλγόριθμοι υλοποιήθηκαν σε γλώσσα C++. Από τα αποτελέσματα και των δύο, αναδείχθηκε η καταλληλότητα των GPUs σε αυτά τα προβλήματα, εξαιτίας του μικρού κόστους τους και της εξαιρετικής απόδοσής τους. Για τον BFS αλγόριθμο και μεγάλους γράφους με εκατομμύρια κόμβους και ακμές, παρουσίασαν speedup έως 50x, σε σύγκριση με την υλοποίηση σε CPU. Ενώ για το δεύτερο αλγόριθμο, παρουσίασαν 70 φορές πιο γρήγορα αποτελέσματα σε σύγκριση με την υλοποίηση σε CPU. Ωστόσο για γράφους με χαμηλή τάξη (π.χ. 2-3) οι οποίοι χαρακτηρίζονται ως γραμμικοί, οι παράλληλοι αλγόριθμοι δεν μπορούν να κερδίσουν καθώς σε κάθε επανάληψη χρειάζεται να εξεταστεί κάθε κόμβος.
Οι Hong, Oguntebi, Olukotun [35] πρότειναν μία νέα μέθοδο για υλοποίηση παράλληλης breadth-first αναζήτησης (BFS) πάνω σε δεδομένα από γράφους. Η μέθοδος αυτή πέτυχε περίπου 2x speedup

απέναντι σε μία διαδεδομένη μέθοδο αναζήτησης.
Οι Merrill, Garland και Grimshaw[36] παρουσίασαν με τη σειρά τους, μία καλά προσαρμοσμένη BFS προσπέλαση για γράφους με τη χρήση των GPUs. Τα αποτελέσματα τους έδειξαν speedup έως 2.5x για τη διάσχυση ενός γράφου με τη χρήση, ενός υβριδικού συστήματος που ανέπτυξαν, το οποίο επιλέγει για κάθε επίπεδο του BFS την κατάλληλη πλατφόρμα που θα τρέξει. Με αυτόν τον τρόπο, το σύστημα κερδίζει και στους μικρούς και στους μεγάλους γράφους, ανεξάρτητα από την τάξη που έχουν.
Παρόλο που το πρόβλημα του frequent subgraph mining δεν είχε μελετηθεί με τη χρήση της τεχνολογίας GPU, οι επιδόσεις των GPUs' σε παρόμοια προβλήματα γράφων απέδειξε ότι ενδεχομένως υπάρχει η δυνατότητα για εξίσου εντυπωσιακά αποτελέσματα. Έτσι, φτάσαμε το καλοκαίρι του 2013, στην πρώτη υλοποίηση με την τεχνολογία των GPUs, στην οποία τα αποτελέσματα είναι πραγματικά εντυπωσιακά[37]. Για γράφους μέχρι 30 κόμβους, η συγκεκριμένη υλοποίηση πέτυχε speedup περίπου 80x.
Τα τελευταία χρόνια πολλές δουλειές γύρω από τις FPGAs έχουν παρουσιαστεί, οι οποίες επιταχύνουν σε σημαντικό βαθμό την επεξεργασία στα προβλήματα του graph mining.
Από τις πρώτες δουλειές που συνδύασαν το πρόβλημα του graph mining με τις FPGAs είναι των Ichikawa, Saito, Udorn, Konishi[38], οι οποίοι υλοποίησαν τον αλγόριθμο του Ullmann σε ένα σύστημα FPGA για την λύση του προβλήματος ισομορφισμού των υπογράφων. Η εκτέλεση σε αυτό το σύστημα, παρουσιάζει 20 φορές καλύτερη επίδοση σε σύγκριση με την πιο διαδεδομένη εκτέλεση σε έναν υπολογιστή.
Επίσης, οι Bondhugula, Devulapalli, Fernando, Wyckoff, Sadayappan[39] παρουσίασαν την πρώτη δουλειά βασισμένη στην τεχνολογία της FPGA για το all-pairs shortest-paths πρόβλημα. Το σύστημα αυτό αναπτύχθηκε σε μία πλατφόρμα Cray XD1 and ξεπέρασε την απόδοση ενός επεξεργαστή κοντά στις 22 φορές.
Οι Thomas, Luk, Stumpl[40] παρουσίασαν το 2007 μία αρχιτεκτονική σε τεχνολογία FPGA που υλοποιεί το canonical labeling των κόμβων ενός γράφου, το οποίο χρησιμοποιείται ευρέως στους αλγόριθμους επεξεργασίας γράφων. Το σύστημα τους μπορείς να προσφέρει επιτάχυνση εκτέλεσης τουλάχιστον 10x. Ενώ, με τη χρήση της μεθόδου canonical graph labeling πετύχαινε ένα speedup κατά ένα παράγοντα γύρω στο 100. Η μέθοδος αυτή προτείνεται για το πρόβλημα της αναγνώρισης των υπογράφων.
Οι Betkaoui, Thomas, Luk, Przulj[41] ανέπτυξαν μία τεχνική (framework) για την υλοποίηση αλγορίθμων του graph mining problem σε αναδιατασσόμενη λογική. Για τη μελέτη του framework, που σχεδίασαν, υλοποιήθηκε ένας αλγόριθμος graphlet counting. Τα αποτελέσματα που πήραν ήταν speedup περίπου 10x απέναντι σε εκτέλεση σε μία 4-Quad CPU.
Επιπλέον, οι Betkaoui, Wang, Thomas, Luk[42] σχεδίασαν τον BFS αλγόριθμο για αναζήτηση ενός υπογράφου χρησιμοποιώντας μία πλατφόρμα πολλαπλών FPGAs, i.e. Convey HC-1 server, πετυχαίνοντας πάνω από 2x επιτάχυνση σε σύγκριση με έναν 32core Xeon server.
Τελειώνοντας, οι Kobori και Maruyama [43] ανέπτυξαν μία υλοποίηση ενός αλγορίθμου graph-cut segmentation σε FPGAs. Η σχεδίασή τους προσέφερε speedup 3x έως 5x, σε σύγκριση με άλλες προτεινόμενες λύσεις σε software και μία υλοποίηση αναπτυγμένη σε GPU.

2.5 Μελέτες και συγκρίσεις των αλγορίθμων Frequent Subgraph Mining

Υπάρχουν πολλές εργασίες, οι οποίες έχουν ως στόχο τη μελέτη και τη σύγκριση των Frequent Subgraph Mining αλγορίθμων. Από αυτές τις εργασίες εξάγονται συμπεράσματα σχετικά με τη μέθοδο που χρησιμοποιεί κάθε αλγόριθμος καθώς επίσης και για τις περιπτώσεις που ένα αλγόριθμος είναι πιο αποδοτικός σε σχέση με τους υπόλοιπους.

Η εργασία [11] συγκρίνει τους αλγορίθμους SUBDUE[20] (Greedy search), AGM[4] (apriori based), gSpan[9] (Pattern Growth based), ως προς το τρόπο λειτουργίας τους και εξάγει συμπεράσματα σχετικά με την καταλληλότητα του καθενός σε σχέση με τους άλλους. Πιο συγκεκριμένα,

περιγράφονται οι λειτουργίες κάθε αλγορίθμου, όπως είναι ο τρόπος με τον οποίο αναπαριστούν την πληροφορία των γράφων, η στρατηγική αναζήτησης που ακολουθούν, ο τρόπος με τον οποίο εκτιμούν τη συχνότητα των υπογράφων και πώς δέχονται τα δεδομένα εισόδου. Η δουλειά αυτή καταλήγει στα εξής συμπεράσματα:

1) Οι αλγόριθμοι AGM[4] και gSpan[9] ολοκληρώνονται, απαριθμώντας όλους τους συχνούς γράφους για τα δεδομένα εισόδου, σε αντίθεση με τον SUBDUE[20], ο οποίος δεν απαριθμεί όλους τους συχνούς υπογράφους που εμφανίζονται.

2) Ο αλγόριθμος SUBDUE[20] προτιμάται σε σύγκριση με τους άλλους δύο, για datasets που αποτελούνται από ένα μόνο γράφο.

3) Οι αλγόριθμοι AGM[4] και gSpan[9] επιλέγονται για μεγάλες βάσεις δεδομένων, στις οποίες παρατηρείται υψηλή συσχέτιση μεταξύ των δεδομένων. Ο gSpan[9] μπορεί να ξεπεράσει σε ταχύτητα εκτέλεσης τον AGM[4] κατά μία τάξη μεγέθους και είναι πιο αξιόπιστος για την εξόρυξη πιο μεγάλων γράφων με πιο μικρή συχνότητα εμφάνισης.

4) Τα ILP συστήματα είναι καλύτερα στην εκμάθηση πολύπλοκης γνώσης, καθώς χρησιμοποιούν αποδοτικά το υπόβαθρο της γνώσης όπως είπαμε και παραπάνω. Για τις Graph-based μεθόδους ο στόχος είναι η εύρεση των πιο συχνών κομματιών του γράφου. Αυτές οι μέθοδοι , έχουν καλύτερη ακρίβεια, επειδή ολοκληρώνονται εξάγοντας όλους τους συχνούς υπογράφους.

Στην εργασία [22] έγινε σύγκριση μεταξύ τεσσάρων αλγόριθμων MoFa[16], gSpan[9], FFSM[14], Gaston[17], οι οποίοι προσεγγίζουν με διαφορετικές τεχνικές το πρόβλημα της εξόρυξης των συχνών υπογράφων. Όλοι οι αλγόριθμοι υλοποιήθηκαν με τη χρήση ίδιων δομών δεδομένων, για την εξαγωγή πιο ασφαλών συμπερασμάτων. Τα συμπεράσματα στα οποία καταλήγει η εργασία είναι τα εξής:

1) Αντίθετα με την κοινή άποψη, η δομή των embedding lists δεν παρουσιάζει πολύ καλό speedup στην αναζήτηση συχνών τμημάτων και επίσης δε λειτουργεί σωστά, όταν δεν υπάρχει πολύ μνήμη διαθέσιμη ή όταν υπάρχει μικρό throughput. Για το λόγο αυτό, ο αλγόριθμος gSpan[9] είναι ανταγωνιστικός απέναντι στους Gaston[17] και FFSM[14], καθώς χρησιμοποιεί ένα σύστημα κωδικής αναπαράστασης των γράφων.

2) Η χρησιμοποίηση αυτών των κωδικών, για την ανίχνευση ισομορφικών γράφων αποτελεί μία πολύ αποδοτική τεχνική για τον έλεγχο του ισομορφισμού των υπογράφων, σε σύγκριση με τις τεχνικές των άλλων αλγορίθμων. Ο αλγόριθμος gSpan[9] μπορεί να αποτύχει, μόνο στην περίπτωση που εξετάζει τεράστιους γράφους.

3) Ο αλγόριθμος MoFa[16] παρόλο που είναι ο πιο αργός από τους αλγορίθμους αυτής της κατηγορίας σε όλες τις δοκιμές, χρησιμοποιείται σε μεγάλο βαθμό για την εξόρυξη γράφων σε μοριακές βάσεις και βιοχημικές ερωτήσεις(biochemical questions).

4) Όλοι οι αλγόριθμοι έχουν γραμμική απόδοση σε σχέση με το μέγεθος της βάσης αν και έχουν διαφορετικούς δείκτες γραμμικής συσχέτισης.

Επίσης, στην εργασία [44] αναλύονται όλοι αλγόριθμοι και κατηγοριοποιούνται με βάση τα κριτήρια που αναφέρθηκαν και πιο πάνω, όπως είναι η στρατηγική αναζήτησης, ο τρόπος που δέχονται τις εισόδους και η ολοκλήρωση της απαρίθμησης κάθε συχνού υπογράφου. Μετά την ανάλυση, γίνεται σύγκριση τριών αλγορίθμων, του FSG[5], του gSpan[9] και του Gaston[17]. Η επιλογή των συγκεκριμένων αλγορίθμων έγινε, επειδή και οι τρείς ολοκληρώνουν πλήρως τη διαδικασία της απαρίθμησης κάθε συχνού υπογράφου μέσα σε ένα σύνολο γράφων. Επίσης, είναι σχεδιασμένοι να δέχονται και οι τρείς ένα σύνολο από γράφους, ως δεδομένα. Τέλος, καθένας από τους τρεις αλγορίθμους, έχει από ένα χαρακτηριστικό, που τον ξεχωρίζει από όλους τους άλλους. Ο αλγόριθμος FSG[5] είναι από τους πιο γνωστούς αλγορίθμους, που λειτουργεί χρησιμοποιώντας τη BFS στρατηγική αναζήτησης. Ο αλγόριθμος gSpan[9] είναι ο πρώτος αλγόριθμος που χρησιμοποίησε τη DFS στρατηγική αναζήτησης. Ενώ ο Gaston[17] είναι από τους πιο πρόσφατους αλγορίθμους και έχει

τη πιο γρήγορη εκτέλεση από όλους.

Από τις μετρήσεις που έγιναν σε τρία διαφορετικά data sets, βγήκε το συμπέρασμα ότι ο Gaston[17] είναι ο πιο γρήγορος από τους τρείς αλγορίθμους. Ο gSpan[9] είναι αρκετά ανταγωνιστικός ως προς το χρόνο εκτέλεσης του Gaston[17] και υπερτερεί του FSG[5]. Ο FSG[5] χάνει, επειδή χρησιμοποιεί την BFS στρατηγική.

Εν κατακλείδι, καταλήγουμε στα εξής συμπεράσματα:

- Δεν ολοκληρώνουν τη διαδικασία της απαρίθμησης των συχνών υπογράφων όλοι οι αλγόριθμοι.
- Οι αλγόριθμοι Gaston[17] και gSpan[9] έχουν τους μικρότερους χρόνους εκτέλεσης.
- Από αυτούς τους δύο, ο αλγόριθμος gSpan[9] χρησιμοποιεί την λιγότερη μνήμη. Αυτό οφείλεται στην χρησιμοποίηση του Canonical Labeling συστήματος. Από την άλλη μεριά, ο Gaston[17] χρησιμοποιεί τη δομή των embedding lists, η οποία μπορεί να μειώσει το χρόνο εκτέλεσης, αλλά απαιτεί επιπλέον κατανάλωση μνήμης.

Με βάση τα παραπάνω συμπεράσματα, στην παρούσα εργασία επιλέχθηκε να μελετηθεί ο αλγόριθμος gSpan[9], καθώς είναι από τους πιο ανταγωνιστικούς αλγορίθμους του Frequent Subgraph Mining προβλήματος και καταλαμβάνει τη μικρότερη κατανάλωση σε πόρους μνήμης.

ΚΕΦΑΛΑΙΟ 3: Ανάλυση αλγορίθμου

3.1Γενικά

Σε αυτό κεφάλαιο παρουσιάζεται η αναλυτική περιγραφή του αλγορίθμου gSpan[9]. Πιο συγκεκριμένα, το τρίτο κεφάλαιο χωρίζεται σε τέσσερα μέρη. Στο πρώτο μέρος, ακολουθεί μία σύντομη αναφορά σε όλους τους βασικούς ορισμούς που χρησιμοποιούνται μέσα στον αλγόριθμο gSpan[9]. Στο δεύτερο μέρος, γίνεται η ανάλυση του αλγορίθμου βήμα-βήμα. Στο τρίτο μέρος, παρουσιάζεται η απόδοση του αλγόριθμου για διαφορετικού τύπου και μεγέθους δεδομένα εισόδου. Τέλος, περιγράφονται τα τμήματα του αλγόριθμου που επιλέχθηκαν για την υλοποίηση τους σε hardware καθώς και η ανάλυση των συγκεκριμένων τμημάτων.

3.2Χρήσιμοι ορισμοί

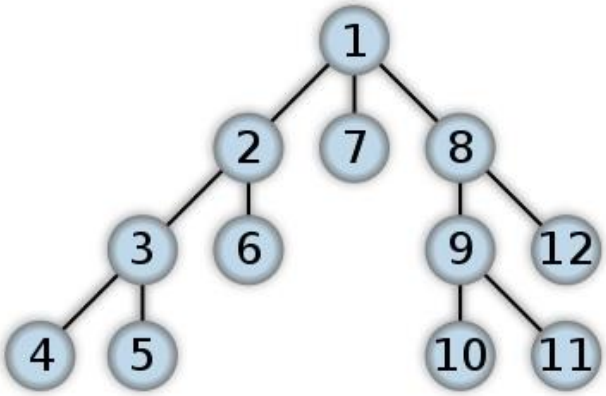
Όπως αναφέρθηκε στο προηγούμενο κεφάλαιο, ο αλγόριθμος gSpan θεωρείται ως ένας από τους πιο διαδεδομένους και γρήγορους αλγορίθμους για το πρόβλημα της εξόρυξης συχνών υπογράφων. Ο gSpan αλγόριθμος ανήκει στην κατηγορία των αλγορίθμων pattern-growth, έτσι σε κάθε βήμα μία καινούργια ακμή του γράφου προστίθεται. Στην συνέχεια ακολουθούν οι έλεγχοι για τη συχνότητα και τον ισομορφισμό των υπογράφων.

Ο αλγόριθμος gSpan είναι ο πρώτος αλγόριθμος από την κατηγορία pattern-growth, ο οποίος χρησιμοποιεί τη depth-first-search (DFS) στρατηγική αναζήτησης των προεκτάσεων. Υπάρχουν τρία κριτήρια που αφορούν κάθε πιθανή προέκταση του γράφου σύμφωνα με τον αλγόριθμο. Όλες οι προεκτάσεις πρέπει να γίνονται πάνω στο πιο «δεξιό» μονοπάτι του γράφου (rightmost path). Για κάθε πιθανή προέκταση που εξετάζει ο αλγόριθμος, μετονομάζει τους κόμβους και τις ακμές, που τους συνδέουν και δημιουργεί έναν κωδικό για το συγκεκριμένο γράφο, ο οποίος καλείται DFS CODE. Το επόμενο βήμα μετά την προέκταση ενός γράφου είναι ο έλεγχος για τον ισομορφισμού του. Για να περάσει ένας γράφος τον έλεγχο αυτό, θα πρέπει να είναι ο ελάχιστος λεξικογραφικά. Αν ο γράφος προκύψει ελάχιστος λεξικογραφικά, τότε συνεχίζεται η προέκτασή του. Σε διαφορετική περίπτωση, ο γράφος αυτός δεν προεκτείνεται περαιτέρω, μιας και δεν πέρασε επιτυχώς τον έλεγχο ισομορφισμού και ο αλγόριθμος συνεχίζει με backtrack στην επόμενη πιθανή προέκταση.

Για την πιο εύκολη κατανόηση του gSpan αλγορίθμου χρειάζεται να γίνει μία λεπτομερής περιγραφή ορισμένων χρήσιμων ορισμών, όπως είναι η DFS στρατηγική αναζήτησης, ο ισομορφισμός υπογράφων, το rightmost path, το όριο support, καθώς και το λεξικογραφικό κριτήριο.

- **DFS στρατηγική αναζήτησης**

  Η στρατηγική αναζήτησης σε ένα πρόβλημα με γράφους αποτελεί ένα βασικό κριτήριο, το οποίο καθορίζει τη φύση του αλγορίθμου και τον διαχωρίζει ή τον κατατάσσει στην ίδια οικογένεια με άλλους αλγορίθμους. Η διάσχιση ενός γράφου, αποτελεί μία διαδικασία, η οποία καθορίζει όλη τη λειτουργικότητα και το συνολικό χρόνο, που χρειάζεται ο αλγόριθμος για να ολοκληρώσει την εξέταση όλων των υπογράφων. Όπως αναφέραμε προηγουμένως, ο αλγόριθμος gSpan[9] είναι ο πρώτος, που χρησιμοποίησε τη μέθοδο depth-first-search (DFS). Η μέθοδος αυτή, εξετάζει ένα μονοπάτι σε βάθος και στην συνέχεια επιστρέφει και εξετάζει το επόμενο μονοπάτι από την αμέσως προηγούμενη διακλάδωση. Η διαδικασία αυτή περιγράφεται στην εικόνα 3.2.1.



Εικόνα 3.2.1 : Depth First tree
(*Πηγή: http://en.wikipedia.org/wiki/Depth-first_search*)

Στον αλγόριθμο gSpan[9], η DFS διάσχιση φαίνεται κατά την εξέταση των πιθανών προεκτάσεων των γράφων. Η διαδικασία αυτή γίνεται αναδρομικά. Σε κάθε αναδρομή, ο αλγόριθμος εξετάζει ένα μονοπάτι σε βάθος. Μόλις ολοκληρωθεί η εξέταση του μονοπατιού, επιστρέφει η αναδρομική κλήση και γίνεται κλήση για το επόμενο μονοπάτι. Πιο συγκεκριμένα, ο αλγόριθμος προεκτείνεται με τρείς τρόπους πάνω στο rightmost path (rmpath), οι οποίοι εξετάζονται με σειρά προτεραιότητας. Για κάθε προέκταση που αποφασίζει, γίνεται μετονομασία των νέων κόμβων και οι νέοι κόμβοι λαμβάνουν καινούργια ονόματα (ids).

- **Ισομορφισμός υπογράφων**

Δύο γράφοι G1(V1,L1,E1) και G2(V2,L2,E2) θεωρούνται ως ισομορφικοί ο ένας ως προς τον άλλον, αν υπάρχει μία αντιστοιχία ένα προς ένα από τους κόμβους του ενός γράφου (V1) στους κόμβους του άλλου (V2), η οποία διατηρεί τα βάρη των κόμβων, τα βάρη των ακμών και όλες τις συνδέσεις με τους γειτονικούς κόμβους.

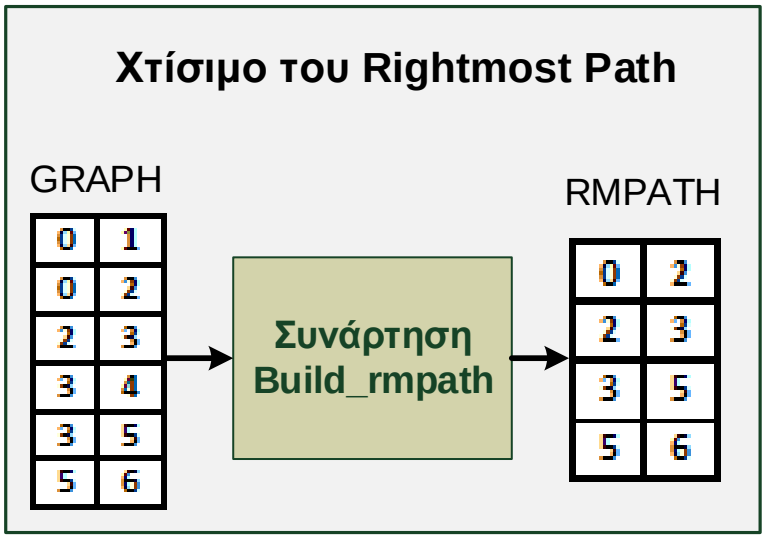
$f : V1 \rightarrow V2$**, τέτοια ώστε :** $(v1, v2) \in E1$ **αν και μόνο αν** $(f(v1), f(v2)) \in E2$

Για το λόγο αυτό, αν δύο γράφοι είναι μεταξύ τους είναι ισομορφικοί, τότε ο ένας γράφος μπορεί να θεωρηθεί ως υποσύνολο του άλλου γράφου. Πιο συγκεκριμένα:

ένας γράφος $GS(VS,LS,ES)$ είναι **υπογράφος** ενός γράφου $G(V,L,E)$ **αν και μόνο αν** $VS \subseteq V$ **και** $ES \subseteq E$**.**

- **Rightmost Path**

Μία πολύ σημαντική ιδιαιτερότητα, που διακρίνει τον αλγόριθμο gSpan από τους υπόλοιπους στο πρόβλημα του FSM (Frequent Subgraph Mining), είναι η προέκταση των γράφων πάνω στο rightmost path. Το μονοπάτι αυτό φτιάχνεται από τις υπάρχουσες ακμές, που έχει ο γράφος και ξαναχτίζεται, κάθε φορά που μία νέα ακμή εισάγεται στο γράφο. Ο αλγόριθμος έχει μία συνάρτηση, η οποία «χτίζει» κάθε φορά το rightmost path. Η διαδικασία αυτή είναι σημαντική, καθώς οι ακμές που περιλαμβάνονται στο μονοπάτι αυτό παίζουν καθοριστικό ρόλο στα κριτήρια με τα οποία θα επιλεγούν οι μελλοντικές προεκτάσεις του γράφου. Το «χτίσιμο» του rightmost path, ξεκινάει από την πιο πρόσφατη ακμή που μπήκε στον γράφο. Στη συνέχεια εξετάζονται όλες οι ακμές του γράφου και επιλέγεται εκείνη, που ενώνει τον κόμβο εκκίνησης της επιλεγμένης ακμής με κάποιον άλλον κόμβο και είναι και πιο πρόσφατη. Η διαδικασία αυτή ολοκληρώνεται μόλις εισαχθεί στο μονοπάτι μία ακμή, η οποία έχει ως τελευταίο κόμβο τη ρίζα του γράφου, δηλαδή τον πρώτο κόμβο του. Η διαδικασία αυτή περιγράφεται σχηματικά στο παρακάτω σχήμα 3.2.2.



Σχήμα 3.2.2: Η διαδικασία εξαγωγής του rmpath από έναν γράφο Graph.

- **Support Threshold**

  Στο FSM πρόβλημα, μία είσοδος του αλγορίθμου, η οποία καθορίζει τη συχνότητα εξόρυξης των υπογράφων είναι το support threshold. Η είσοδος αυτή δίνεται από το χρήστη και δημιουργεί ένα κάτω όριο, το οποίο χρησιμοποιεί ο αλγόριθμος για να αγνοήσει εκείνους τους υπογράφους που έχουν συχνότητα εμφάνισης χαμηλότερη από αυτό. Το κατώφλι αυτό συνήθως είναι ένα ποσοστό, το οποίο μετράει τη ποσοστιαία συχνότητα εμφάνισης ενός υπογράφου μέσα στους γράφους που δόθηκαν στο dataset εισόδου.

  Ο αλγόριθμος gSpan δέχεται έναν ακέραιο αριθμό **σ**, ως είσοδο για το όριο support. Επίσης, δέχεται και τους γράφους που υπάρχουν μέσα στο dataset. Διαβάζοντας το dataset εισόδου απαριθμεί το πλήθος των διακριτών γράφων **G**. Οπότε το ποσοστό με βάση το οποίο ο αλγόριθμος gSpan κόβει τους υπογράφους που δεν είναι συχνοί, προκύπτει ως το πηλίκο:

  support threshold = σ/G (*100) %

- **Λεξικογραφικό κριτήριο**

  Το λεξικογραφικό κριτήριο είναι ένα κριτήριο, το οποίο χρησιμοποιεί ο αλγόριθμος gSpan, με σκοπό να δώσει προτεραιότητα στις υποψήφιες ακμές, οι οποίες προεκτείνουν ένα γράφο. Σαν αποτέλεσμα, ο αλγόριθμος καταφέρνει να μειώσει σημαντικά το χώρο αναζήτησης (search space)των εξεταζόμενων μονοπατιών μέσα στους γράφους, αφού διαχωρίζει πολύ εύκολα ποια μονοπάτια έχουν ήδη εξεταστεί. Πιο συγκεκριμένα, η προτεραιότητα αφορά το βάρος των κόμβων που συνδέει μία ακμή, καθώς και το βάρος της ίδιας της ακμής. Η μικρότερη «λεξικογραφικά» ακμή, είναι εκείνη που έχει το μικρότερο βάρος κόμβου εκκίνησης, το μικρότερο βάρος ακμής και το μικρότερο βάρος κόμβου προορισμού, με αυτή τη σειρά. Το λεξικογραφικό κριτήριο χρησιμοποιείται σε τρία σημεία του αλγορίθμου. Το πρώτο είναι στην αρχή, όταν ψάχνει τις μικρότερες ακμές, που θα αποτελέσουν τη ρίζα κάθε εξεταζόμενου γράφου. Το δεύτερο είναι στον έλεγχο ισομορφισμού του γράφου. Και το τελευταίο είναι, όταν εξετάζει τις υποψήφιες προεκτάσεις του γράφου και υπάρχουν πολλές προεκτάσεις από τον ίδιο κόμβο. Στην περίπτωση αυτή, διαλέγει πρώτα τη μικρότερη «λεξικογραφικά» και μόλις επιστρέψει η αναδρομή - δηλαδή τελειώσει εκείνο το μονοπάτι - επιλέγει την επόμενη.

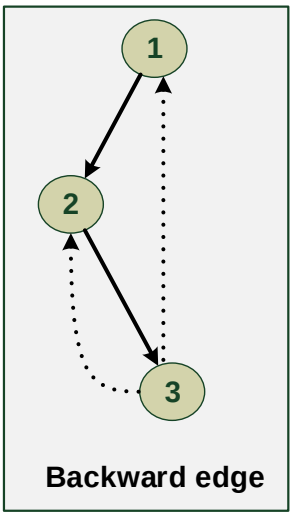
- **Τα τρία κριτήρια προέκτασης των ακμών**

Όπως είπαμε πιο πάνω, στον αλγόριθμο χρησιμοποιούνται τρία κριτήρια για την προέκταση των

ακμών. Και τα τρία κριτήρια αναζητούν ακμές με συγκεκριμένα χαρακτηριστικά, πάνω στο right-most path (rmpath). Τα χαρακτηριστικά αυτά διαφέρουν ανάλογα με το είδος των προεκτάσεων (Backward, Forward pure, Forward rmpath).

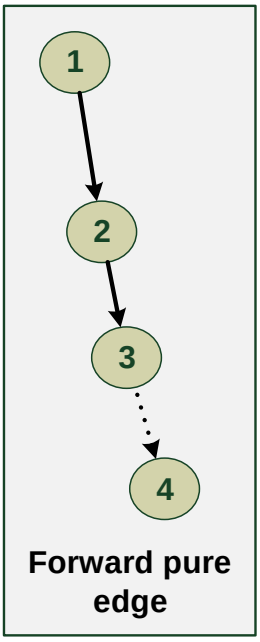
- Backward

Μία προέκταση χαρακτηρίζεται ως backward, αν ξεκινάει από τον τελευταίο κόμβο που βρίσκεται μέσα στο rmpath και καταλήγει σε κόμβο, που επίσης είναι μέσα στο rmpath. Κατά τη διαδικασία εύρεσης αυτών των ακμών, επιστρέφονται όλες οι ακμές που ικανοποιούν την παραπάνω συνθήκη και δίνεται προτεραιότητα για την εξέταση σε αυτήν που πηγαίνει προς τον πιο παλιό κόμβο μέσα στο rmpath. Ένας κόμβος όσο πιο μικρό id έχει, τόσο πιο παλιός θεωρείται.



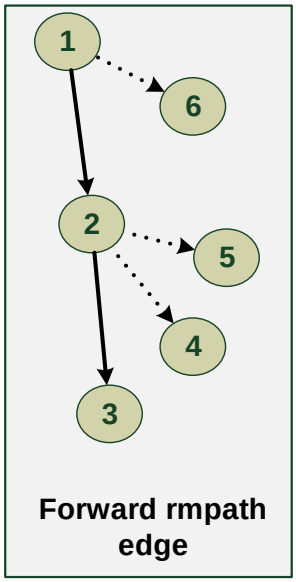
- Forward pure

Μία προέκταση χαρακτηρίζεται ως forward pure, αν ξεκινάει από τον τελευταίο κόμβο μέσα στο rmpath και ο κόμβος προορισμού δεν υπάρχει μέσα στο rmpath. Κατά τη διαδικασία εύρεσης αυτών των ακμών, επιστρέφονται όλες οι ακμές που ικανοποιούν την παραπάνω συνθήκη και δίνεται προτεραιότητα σε αυτήν που είναι μικρότερη λεξικογραφικά.



- Forward rmpath

Μία προέκταση χαρακτηρίζεται ως forward rmpath, αν ξεκινάει από οποιονδήποτε κόμβο μέσα στο rmpath εκτός του τελευταίου κόμβου και ο κόμβος προορισμού δεν υπάρχει μέσα στο rmpath. Κατά τη διαδικασία εύρεσης αυτών των ακμών, επιστρέφονται όλες οι ακμές που ικανοποιούν την παραπάνω συνθήκη και δίνεται προτεραιότητα σε αυτήν που ξεκινάει από τον πιο πρόσφατο κόμβο μέσα στο rmpath. Ένας κόμβος χαρακτηρίζεται ως πιο πρόσφατος, όσο πιο μεγάλο id έχει.



3.3 Ανάλυση του αλγορίθμου gSpan

Στο κεφάλαιο αυτό, παρουσιάζεται αναλυτικά η λειτουργία του αλγορίθμου gSpan. Μία σύντομη περιγραφή των βημάτων του αλγορίθμου, που θα αναλυθούν παρακάτω, φαίνεται στην εικόνα 3.3.

Αλγόριθμος gSpan
Είσοδος : όριο support **σ**, γράφοι εισόδου **G**
Έξοδος : απαριθμημένοι συχνοί υπογράφοι

- Διαβάζει όλους τους γράφους.
- Μετονομάζει τους κόμβους και τις ακμές κάθε γράφου.
- Βρίσκει κάθε μονή ακμή από τους αρχικούς γράφους.
- Τις ταξινομεί, σύμφωνα με τα λεξικογραφικά κριτήρια.
- Εξετάζει όλες τις πιθανές επεκτάσεις, καλώντας επαναληπτικά την **Subgraph_Mining Διαδικασία**.

Subgraph_Mining Διαδικασία:
- Ελέγχει το support όριο του υπογράφου.
- Εκτελεί το test ισομορφισμού του υπογράφου.
- Εκτυπώνει τον συχνό υπογράφο.
- Εξετάζει κάθε πιθανή επέκταση.
- Για κάθε νέα επέκταση, καλεί αναδρομικά την **Subgraph_Mining Διαδικασία**.

Εικόνα 3.3 : Βασικά βήματα αλγόριθμου gSpan

Όπως αναφέρθηκε στο προηγούμενο κεφάλαιο, ο αλγόριθμος gSpan είναι ένας από τους πιο διαδεδομένους στο πρόβλημα της εξόρυξης συχνών υπογράφων. Ο αλγόριθμος χρειάζεται δύο εισόδους, για να ξεκινήσει τη λειτουργία της εξόρυξης, όπως φαίνεται στην παραπάνω εικόνα. Η πρώτη είσοδος είναι ένας ακέραιος αριθμός **σ**, το οποίο αποτελεί την ελάχιστη συχνότητα εμφάνισης κάθε υπογράφου. Η δεύτερη είσοδος είναι οι γράφοι του dataset, μέσα από τους οποίους ο αλγόριθμος θα αναζητήσει κάθε μικρότερο συχνό υπογράφο.

Στο πρώτο βήμα ο αλγόριθμος διαβάζει το dataset εισόδου και κρατάει την πληροφορία για τα βάρη των κόμβων και των ακμών, κάθε γράφου. Η πληροφορία αυτή κρατείται σε δομές δεδομένων τύπου λίστας. Για κάθε κόμβο μέσα στο γράφο, υπάρχει μία λίστα, η οποία περιέχει τις ακμές που ξεκινούν από τον κόμβο αυτό. Κάθε ακμή, αποκτά δύο ταυτότητες ids. Η μία ταυτότητα ονομάζεται TRANS-id και προσδιορίζει σε ποιον γράφο ανήκει. Ενώ, η δεύτερη ονομάζεται edge-id και προσδιορίζει τη σειρά, που έχει η ακμή μέσα στο γράφο.

Στη συνέχεια, ο αλγόριθμος αναζητεί όλες τις μονές ακμές, που υπάρχουν μέσα σε όλους τους γράφους και τις ταξινομεί με βάση το λεξικογραφικό κριτήριο, το οποίο αναλύθηκε στην προηγούμενη ενότητα. Αφού ταξινομήσει όλες τις ακμές, τότε προεκτείνει μία κάθε φορά, ξεκινώντας από τις προεκτάσεις της μικρότερης «λεξικογραφικά» ακμής. Η διαδικασία αυτή όπως φαίνεται στην παραπάνω εικόνα, ονομάζεται **Subgraph_Mining**.

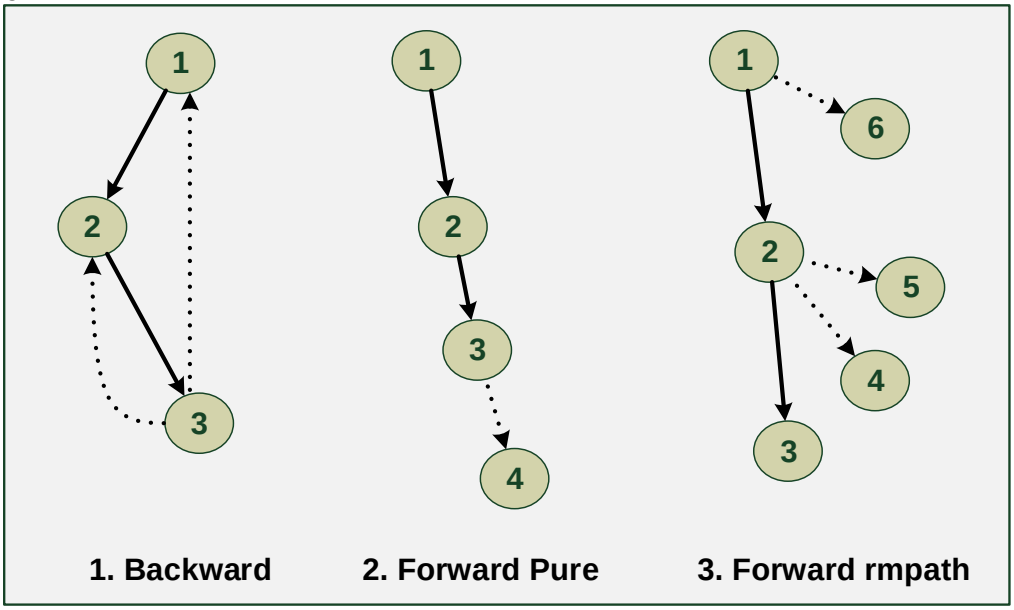
Η διαδικασία αυτή είναι μία αναδρομική διαδικασία, η οποία εξετάζει σε βάθος κάθε μονοπάτι που ξεκινάει από μία μονή ακμή ή όπως την αναφέρει ο αλγόριθμος root edge. Όμως, η αναδρομική αυτή διαδικασία, καλείται επαναληπτικά για κάθε μονή ακμή που υπάρχει μέσα στους γράφους εισόδου. Για το λόγο αυτό, έχει μία σταθερή δομή, η οποία αποτελείται από τέσσερα βασικά μέρη.
Τα δύο πρώτα μέρη αποτελούν ουσιαστικά δύο κριτήρια, με βάση τα οποία ο αλγόριθμος κόβει μονοπάτια που είτε δεν είναι συχνά, ή έχουν ήδη εξεταστεί. Αν έστω ένα από τα δύο κριτήρια αποτύχει, τότε ο υπογράφος σταματάει να εξετάζεται και η αναδρομική διαδικασία επιστρέφει ένα βήμα πίσω. Με αυτό τον τρόπο, ο αλγόριθμος μειώνει σημαντικά το χώρο αναζήτησης (search space) και καταφέρνει να γίνει ανταγωνιστικά γρήγορος στη διαδικασία της εξόρυξης συχνών υπογράφων. Η σειρά με την οποία εξετάζονται τα δύο κριτήρια είναι εξίσου σημαντική για την απόρριψη ενός μονοπατιού. Πρώτο εξετάζεται το κριτήριο της συχνότητας, καθώς είναι γρήγορο και στη συνέχεια το κριτήριο του ισομορφισμού.

Στο κριτήριο της συχνότητας, εξετάζεται αν ο συγκεκριμένος υπογράφος που έχει προεκταθεί, βρίσκεται σε τουλάχιστον **σ** αριθμό από τους γράφους εισόδου.

Αν ο υπογράφος είναι συχνός, τότε εξετάζεται το κριτήριο του ισομορφισμού. Το κριτήριο αυτό ελέγχει αν ο υπογράφος αυτός μπορεί να προκύψει από κάποιο άλλο υπογράφο εισόδου. Στον έλεγχο αυτό, δημιουργείται ένας νέος γράφος, με τις ελάχιστες «λεξικογραφικά» συνδέσεις. Οι συνδέσεις αυτές τηρούν το λεξικογραφικό κριτήριο, όπως αναλύθηκε στην παράγραφο 3.2 καθώς και την προτεραιότητα για την εξέταση των πιθανών προεκτάσεων. Η προτεραιότητα αυτή, αποτελεί ουσιαστικά έναν κανόνα του αλγορίθμου gSpan και φαίνεται στην εικόνα 3.3.1. Πρώτα, αναζητείται αν υπάρχει μία ακμή τύπου backward. Αν υπάρχουν πολλές backward ακμές, προτεραιότητα έχει εκείνη που πηγαίνει στον πιο παλιό κόμβο. Στη συνέχεια αναζητείται μία ακμή τύπου forward_pure και αν δεν υπάρχει καμία ακμή από τα δύο αυτά είδη, τότε θα εισαχθεί μία ακμή τύπου forward_rmpath. Αν υπάρχουν πολλές forward_rmpath ακμές, προτεραιότητα έχει εκείνη που ξεκινάει από τον πιο πρόσφατο κόμβο στο rightmost path. Στην περίπτωση που ο νέος γράφος είναι ο ίδιος με τον γράφο που εξετάζεται, τότε ο γράφος αυτός ορίζεται ως «λεξικογραφικά» ελάχιστος. Αν ισχύει αυτό, τότε η

διαδικασία της προέκτασής του συνεχίζεται. Σε διαφορετική περίπτωση, σταματάει η διαδικασία αυτή, γιατί ο υπογράφος αυτός θα προεκταθεί από άλλο μονοπάτι.

Στο τρίτο μέρος εκτυπώνεται ο υπογράφος που έχει εξεταστεί και έχει περάσει τον έλεγχο για τη συχνότητα και τον ισομορφισμό του. Στη συνέχεια, ο αλγόριθμος εξετάζει όλες τις πιθανές προεκτάσεις που έχει ο συγκεκριμένος υπογράφος. Για να μεγαλώσει ένας υπογράφος, θα πρέπει οι προεκτάσεις του να έχουν αποφασιστεί με βάση τους κανόνες προέκτασης που έχει ο αλγόριθμος gSpan. Όπως είπαμε προηγουμένως ο αλγόριθμος gSpan, εξετάζει τις προεκτάσεις ενός υπογράφου, μόνο προς το πιο δεξί μονοπάτι (rightmost path). Οι προεκτάσεις αυτές χωρίζονται σε τρία είδη, τα οποία φαίνονται στην εικόνα 3.3.1. Προκειμένου ο αλγόριθμος να εξετάσει όλες τις προεκτάσεις για όλους τους γράφους με την ίδια σειρά, εξετάζει όλες τις πιθανές προεκτάσεις(και από τα τρία είδη προέκτασης) που μπορεί να έχει ο συγκεκριμένος υπογράφος μέσα σε κάθε γράφο από τους γράφους εισόδου.
Η επιλογή της σειράς με την οποία θα εξεταστούν οι πιθανές προεκτάσεις, γίνεται με προτεραιότητα ως προς: α) το είδος της ακμής, β) προτεραιότητα μέσα στις συναρτήσεις.
α) Δηλαδή, πρώτα επιλέγονται οι ακμές που έχουν προκύψει από τη συνάρτηση backward, στη συνέχεια αυτές που έχουν προκύψει από τη συνάρτηση forward_pure και τέλος αυτές από τη συνάρτηση forward_rmpath.
β) Κάθε συνάρτηση ακολουθεί μία σειρά προτεραιότητας για την εξέταση των ακμών που προκύπτουν από αυτήν. Πιο συγκεκριμένα, η backward συνάρτηση επιλέγει πρώτα την επέκταση προς τον παλαιότερο κόμβο. Έτσι, θα γίνει πρώτα η επέκταση του γράφου με την ακμή (3-1). Από την άλλη μεριά, για τη συνάρτηση forward_pure, θα γίνει πρώτα η επέκταση του γράφου με την ακμή, της οποίας ο κόμβος προορισμού έχει το μικρότερο βάρος (λεξικογραφικό κριτήριο). Όπως φαίνεται στην εικόνα 3.3.1, η forward_rmpath συνάρτηση επιλέγει να προεκτείνει πρώτα από τον πιο πρόσφατο κόμβο του rightmost path.



Εικόνα 3.3.1 : Τα τρία είδη επέκτασης του αλγορίθμου gSpan.

Στο τέλος, για κάθε νεοσύστατο υπογράφο καλείται η **Subgraph_Mining** διαδικασία, στην οποία εξετάζεται ξανά η συχνότητα και ο ισομορφισμός του. Μόλις ολοκληρωθεί η διαδικασία επέκτασης του συγκεκριμένου μονοπατιού σε βάθος, τότε επιστρέφει η αναδρομική διαδικασία και ξανακαλείται για κάθε μία από τις ακμές που είχαν αποθηκευτεί στην προηγούμενη αναδρομή. Όταν ολοκληρωθεί η εξέταση όλων των μονοπατιών, για κάθε μία από αυτές τις επεκτάσεις, τότε η διαδικασία επιστρέφει στο προηγούμενο επίπεδο, στο οποίο ξανακαλείται με όρισμα την επόμενη μονή ακμή (root edge).

3.4 Ανάλυση αλγορίθμου gSpan για υλοποίηση σε αναδιατασσόμενη λογική

Ο αλγόριθμος gSpan, παρόλο που είναι ένας DFS αλγόριθμος, δίνει τη δυνατότητα για παράλληλες κλήσεις, λόγω της φύσης των γράφων. Όπως είδαμε στην ενότητα 3.4, υπάρχουν πολλά κομμάτια που είναι κρίσιμα κατά την εκτέλεση του αλγορίθμου. Τα πιο σημαντικά είναι η διαδικασία επέκτασης του κυρίως υπογράφου, και ο έλεγχος για τον ισομορφισμό του. Κατά τη διαδικασία επέκτασης ενός υπογράφου, εξετάζονται όλες οι εμφανίσεις του μέσα στους γράφους του αρχικού data set εισόδου. Για όλες αυτές τις εμφανίσεις, αναζητούνται όλες οι πιθανές επεκτάσεις και στη συνέχεια απαριθμούνται όλες οι κοινές επεκτάσεις και κρατούνται όσες ξεπερνούν το όριο της συχνότητας, που έχει ορίσει ο χρήστης. Έπειτα, οι ακμές αυτές, προχωρούν στον έλεγχο για τον ισομορφισμό κάθε υπογράφου. Αν ένας υπογράφος είναι λεξικογραφικά ελάχιστος, τότε συνεχίζεται η επέκτασή του, αναδρομικά. Σε διαφορετική περίπτωση, η διαδικασία επέκτασης του υπογράφου σταματάει και ξεκινάει η εξέταση ενός άλλου υπογράφου. Στη συνέχεια, θα παρουσιάσουμε αναλυτικά τις διαδικασίες του αλγορίθμου gSpan καθώς επίσης και το διάγραμμα ροής ολόκληρης της εφαρμογής.

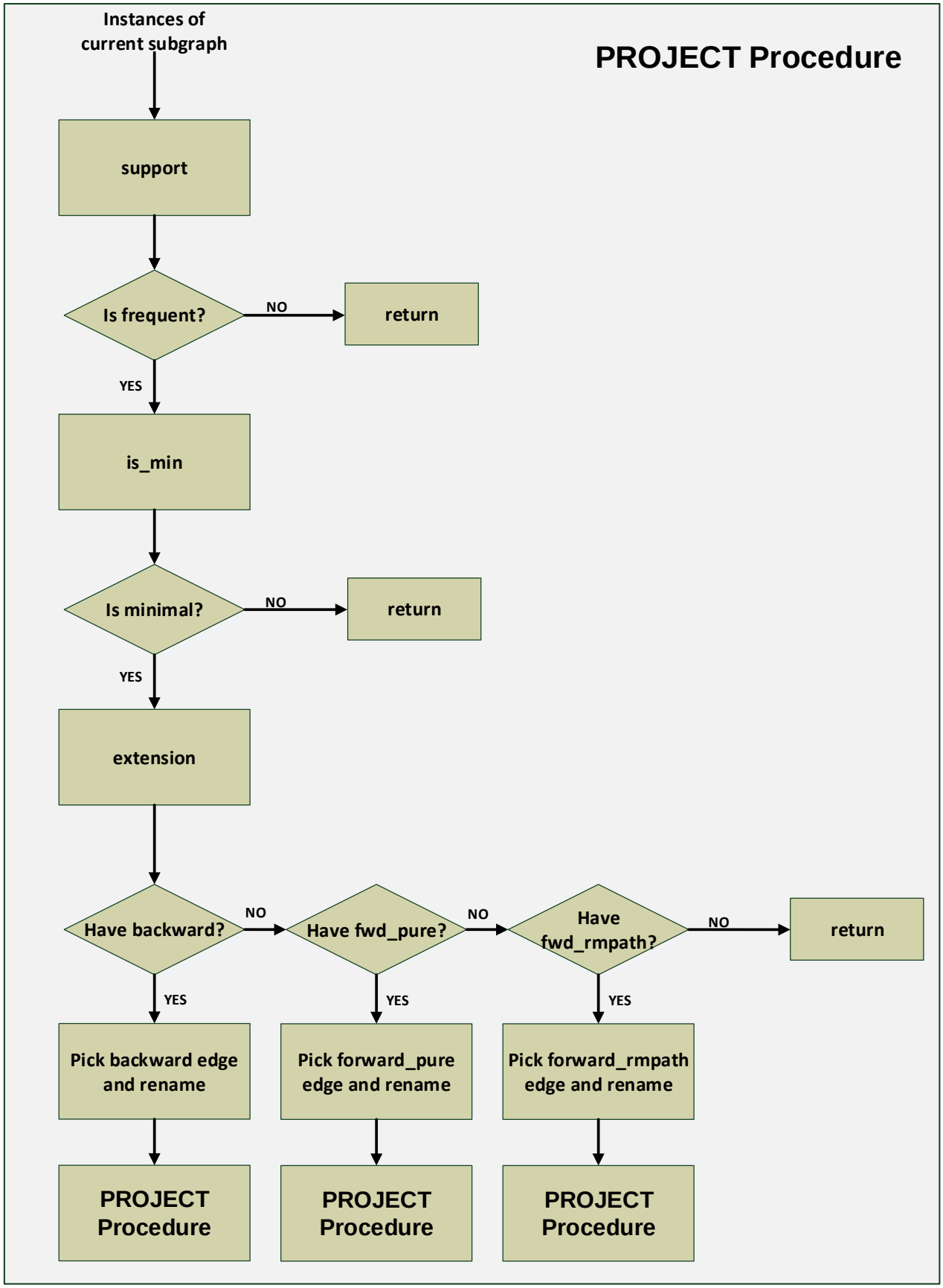
3.4.1 Διαδικασία επέκτασης υπογράφων (project)

Η διαδικασία της επέκτασης των υπογράφων, δέχεται ως είσοδο μία λίστα με τους εξεταζόμενους υπογράφους, που υπάρχουν μέσα στους αρχικούς γράφους του dataset. Έχοντας αυτά τα στοιχεία και εξετάζοντας τα κατάλληλα δεδομένα για τους κόμβους από τη δομή των γράφων, ο αλγόριθμος επεκτείνει κάθε πιθανό υπογράφο σε βάθος. Όπως φαίνεται στην εικόνα 3.5.1, η λειτουργία αυτή αναλύεται σε τρία βασικά βήματα:

i. support: έλεγχος για τη συχνότητα του εξεταζόμενου υπογράφου.

ii. is_min: έλεγχος για τον ισομορφισμό του εξεταζόμενου υπογράφου.

iii. extension: αναζήτηση όλων των πιθανών επεκτάσεων του γράφου. Οι επεκτάσεις αυτές προκύπτουν από το data set εισόδου, χωρίς να εξετάζονται οι καθρεφτικές ακμές. Κάθε ακμή που μπορεί να επεκτείνει τον υπογράφο επιλέγεται για την αναδρομική κλήση της project διαδικασίας. Η προτεραιότητα που ακολουθούν οι ακμές είναι ως προς: α) το είδος της ακμής, β) προτεραιότητα μέσα στις συναρτήσεις, γ) λεξικογραφικό κριτήριο.



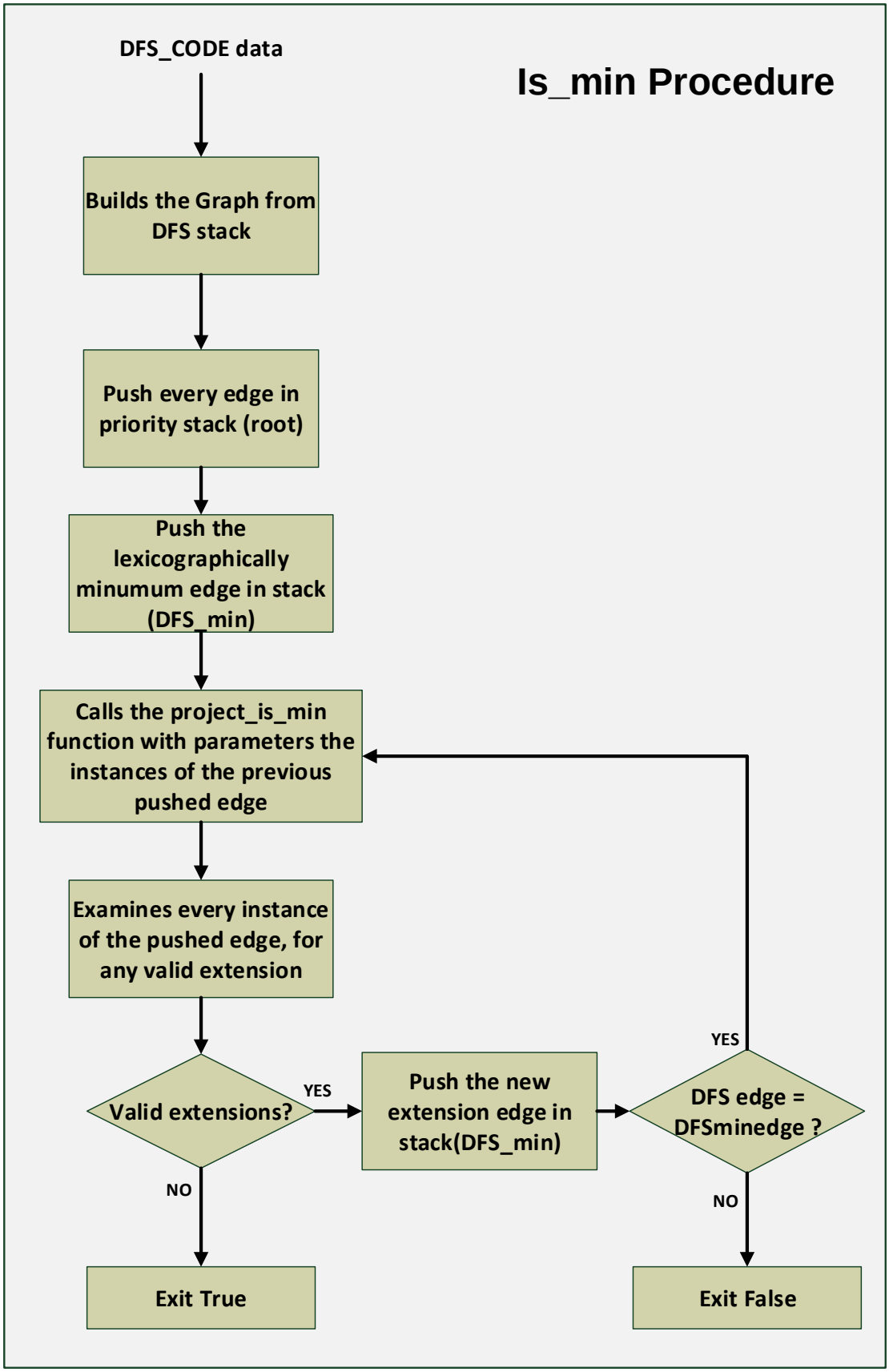
Εικόνα 3.5.1 : Το διάγραμμα ροής της διαδικασίας προέκτασης στον gSpan.

3.4.1.1 Διαδικασία ελέγχου συχνότητας υπογράφων (support)

Η διαδικασία ελέγχου της συχνότητας των υπογράφων, γίνεται με το μέτρημα εκείνων των υποστάσεων του συγκεκριμένου υπογράφου, οι οποίες βρίσκονται μέσα σε διαφορετικό γράφο εισόδου. Πιο συγκεκριμένα, στο σύνολο των υποστάσεων μπορούν να υπάρχουν και πολλαπλές υποστάσεις από τον ίδιο αρχικό γράφο. Αυτές οι υποστάσεις μετρούνται ως μία για τον έλεγχο της συχνότητας. Έτσι, όταν το σύνολο των αρχικών γράφων εισόδου μέσα στους οποίους υπάρχει τουλάχιστον μία φορά ο συγκεκριμένος υπογράφος, είναι μεγαλύτερο ή ίσο με το όριο της συχνότητας που έχει δώσει ο χρήστης, τότε ο υπογράφος είναι συχνός. Σε αντίθετη περίπτωση, ο υπογράφος δεν είναι συχνός και σταματάει η διαδικασία της επέκτασης του.

3.4.1.2 Διαδικασία ελέγχου ισομορφισμού υπογράφων (is_min)

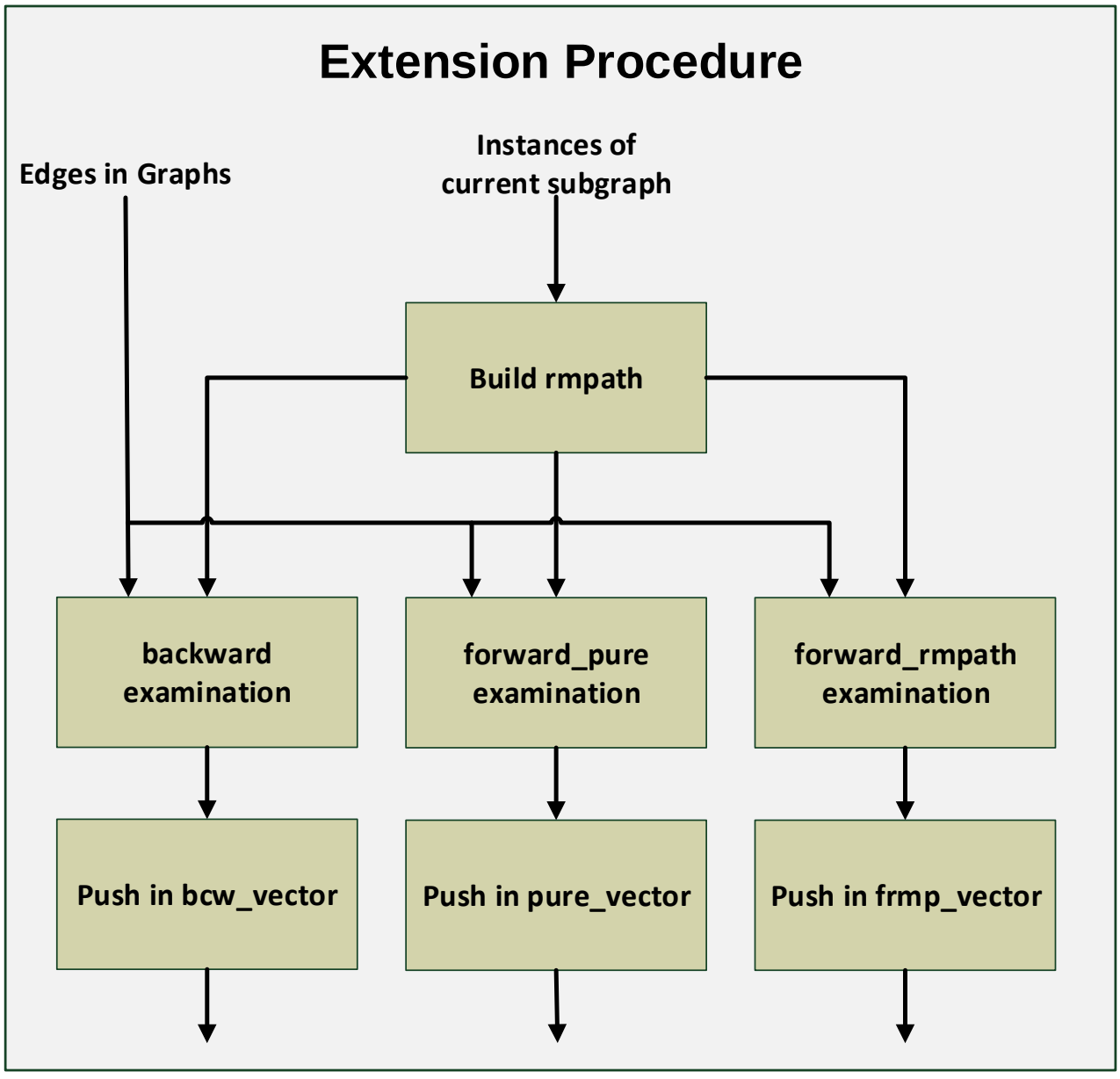
Η διαδικασία ελέγχου του ισομορφισμού των υπογράφων, ελέγχει αν ο συγκεκριμένος υπογράφος είναι ο ελάχιστος λεξικογραφικά. Ο έλεγχος αυτός γίνεται, δημιουργώντας από την αρχή έναν καινούργιο γράφο, με δεδομένες τις ακμές, τους κόμβους και τα βάρη τους. Κατά τη δημιουργία του ελάχιστου γράφου, χρησιμοποιούνται κάποια κριτήρια. Πρώτον, βρίσκεται η μικρότερη ακμή ως προς τα βάρη, με βάση το λεξικογραφικό κριτήριο που αναλύσαμε προηγουμένως. Στη συνέχεια, αναζητούνται όλες οι επεκτάσεις και επιλέγονται με σειρά προτεραιότητας πρώτα η backward επέκταση, στη συνέχεια η forward_pure επέκταση και τελευταία η forward_rmpath επέκταση. Αν υπάρχουν πολλές backward επεκτάσεις επιλέγεται πρώτα αυτή της οποίας ο κόμβος προορισμού έχει το μικρότερο id, δηλαδή είναι ο παλαιότερος μέσα στο rmpath. Αντίστοιχα, αν υπάρχουν πολλές forward_pure επεκτάσεις επιλέγεται πρώτα αυτή που είναι μικρότερη λεξικογραφικά. Τέλος, αν υπάρχουν πολλές forward_rmpath επεκτάσεις επιλέγεται πρώτα αυτή της οποίας ο κόμβος εκκίνησης έχει το μεγαλύτερο id μέσα στο rmpath, δηλαδή είναι ο πιο πρόσφατος μέσα στο rmpath. Η παραπάνω διαδικασία περιγράφεται στην εικόνα 3.5.1.2.



Εικόνα 3.5.1.2 : Το διάγραμμα ροής της διαδικασίας ελέγχου ισομορφισμού στον gSpan.

3.4.1.3 Διαδικασία αναζήτησης επεκτάσης των υπογράφων (extension)

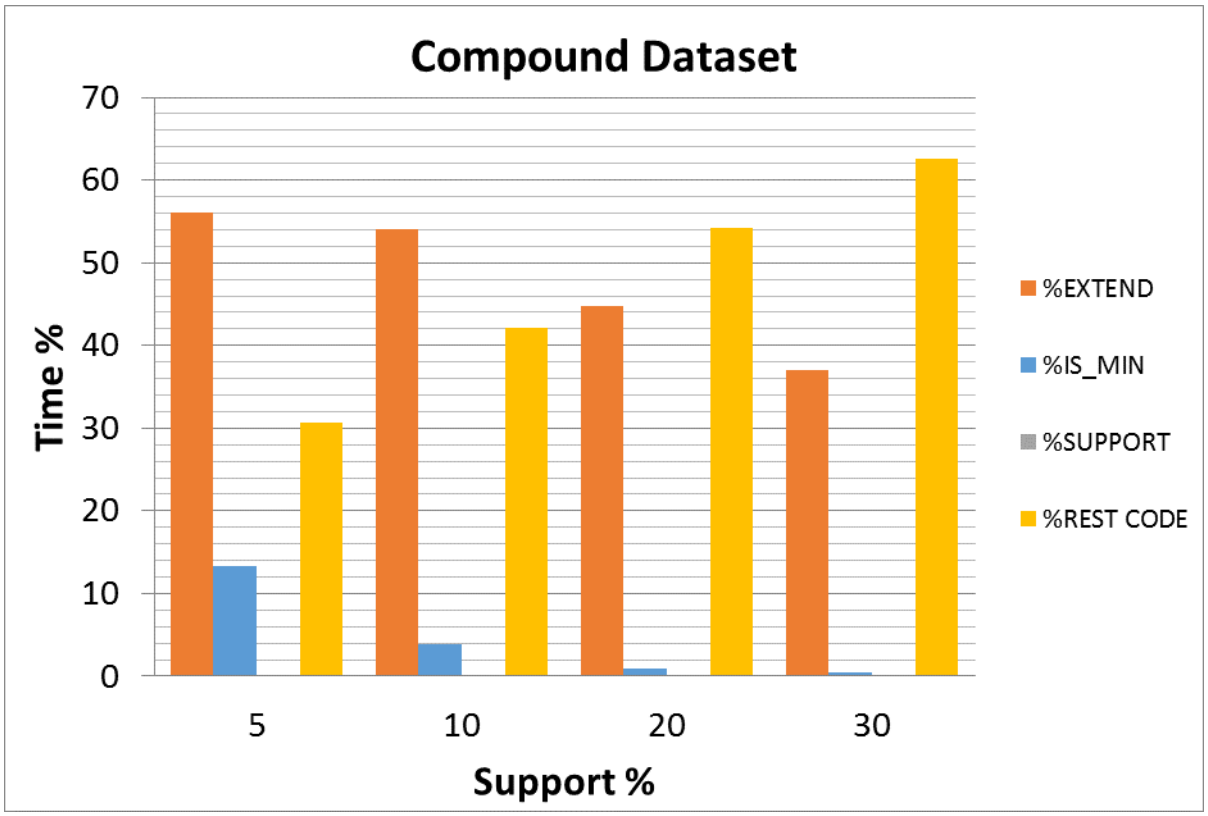
Η διαδικασία επέκτασης των υπογράφων, αρχικά δημιουργεί το rightmost path συγκεκριμένου υπογράφου. Στη συνέχεια, εξετάζει όλες τις πιθανές επεκτάσεις για κάθε είδος, backward, forward_pure και forward_rmpath. Κάθε ακμή που προκύπτει, ανάλογα με το είδος της τοποθετείται σε ένα vector. Στο τέλος, επιστρέφει κάθε ακμή από κάθε είδος. Μετά την ολοκλήρωση της διαδικασίας αναζήτησης των επεκτάσεων, το κομμάτι του project, είναι υπεύθυνο να επιλέξει μία μία όλες τις ακμές, να τις μετονομάσει και να τις προεκτείνει. Η παραπάνω διαδικασία περιγράφεται στην εικόνα 3.5.1.3.



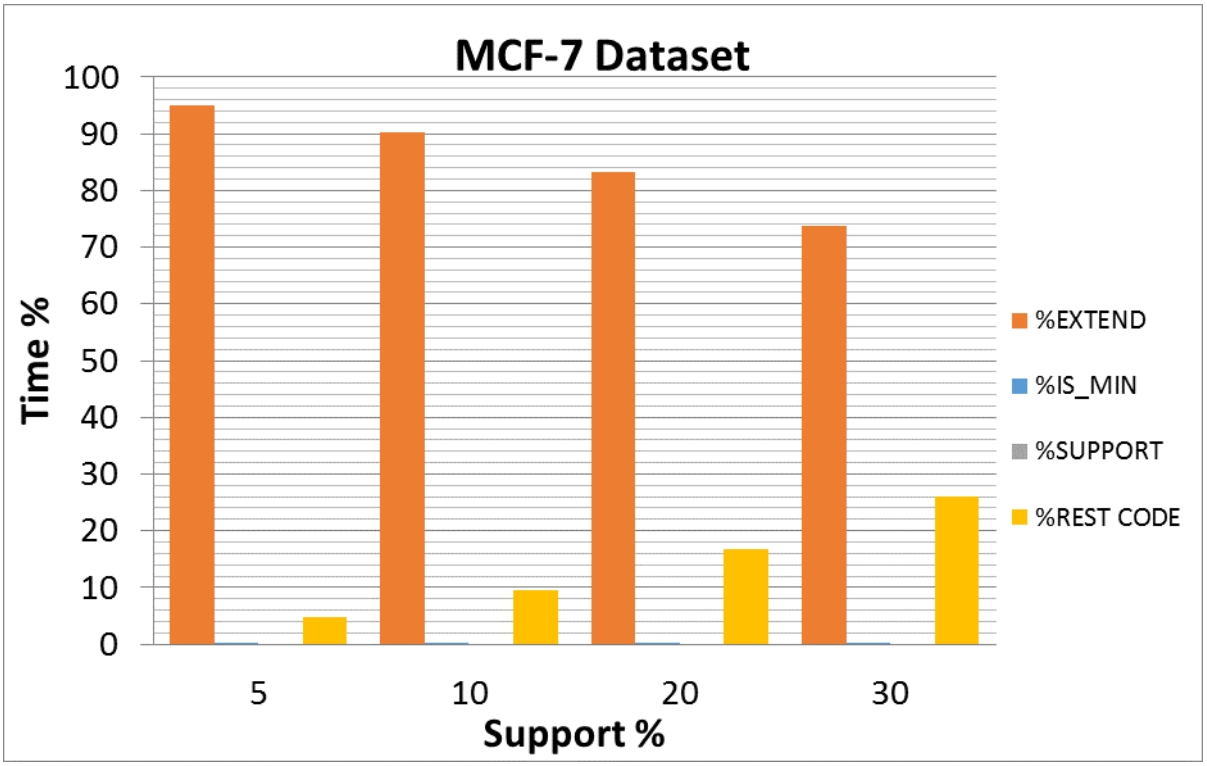
Εικόνα 3.5.1.3 : Το διάγραμμα ροής της διαδικασίας αναζήτησης των επεκτάσεων κάθε υπογράφου.

3.5 Profiling του αλγορίθμου gSpan

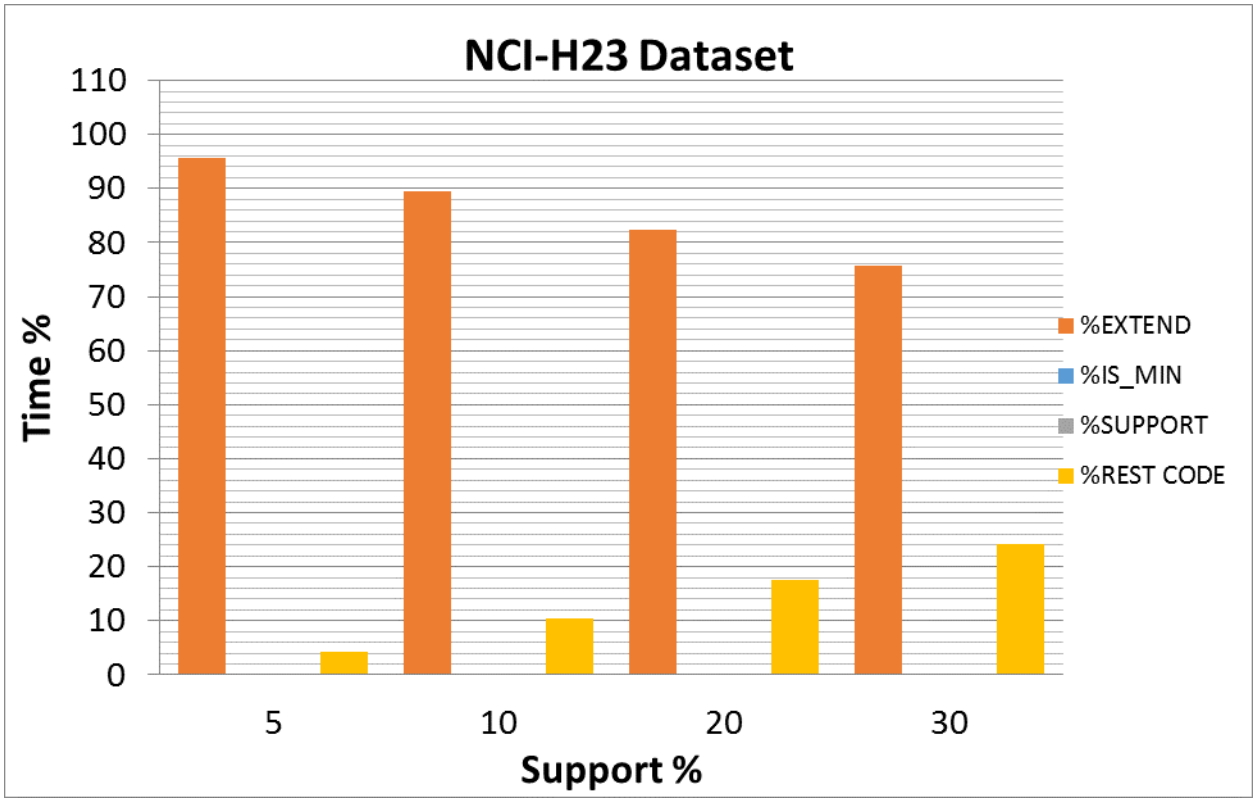
Για την ανίχνευση των σημείων, όπου ο αλγόριθμος gSpan καταναλώνει τον περισσότερο χρόνο εκτέλεσης(hot spots), χρησιμοποιήθηκαν αρχικά τα εργαλεία Vtune και gprof. Τα εργαλεία αυτά, έδειξαν ως πιο επεξεργαστικά βαρύ κομμάτι του κώδικα, τη διαδικασία για τον έλεγχο του ισομορφισμού (is_min). Έπειτα, από την επικύρωση της αρχιτεκτονικής του is_min και τις μετρήσεις, είδαμε ότι η ανίχνευση της βαριάς συνάρτησης με τη χρήση των δύο εργαλείων δεν ήταν αξιόπιστη. Το συμπέρασμα, στο οποίο καταλήξαμε είναι ότι οι profilers δε δίνουν αξιόπιστα αποτελέσματα λόγω της μεγάλης αναδρομής του αλγορίθμου.
Στη συνέχεια, για πιο αξιόπιστα αποτελέσματα έγινε χρήση της συνάρτησης της C get_time_of_day, η οποία έχει πολύ μεγάλη ακρίβεια στη χρονομέτρηση. Η παραπάνω συνάρτηση χρησιμοποιήθηκε επίσης στη διαδικασία μέτρησης των τελικών αποτελεσμάτων. Τα αποτελέσματα από αυτή τη μέτρηση φαίνονται στις παρακάτω εικόνες.



Εικόνα 3.5.1 : Καταμερισμός του χρόνου εκτέλεσης ανά συνάρτηση, με βάση τη παράμετρο support για το Compound Dataset (422 graphs).

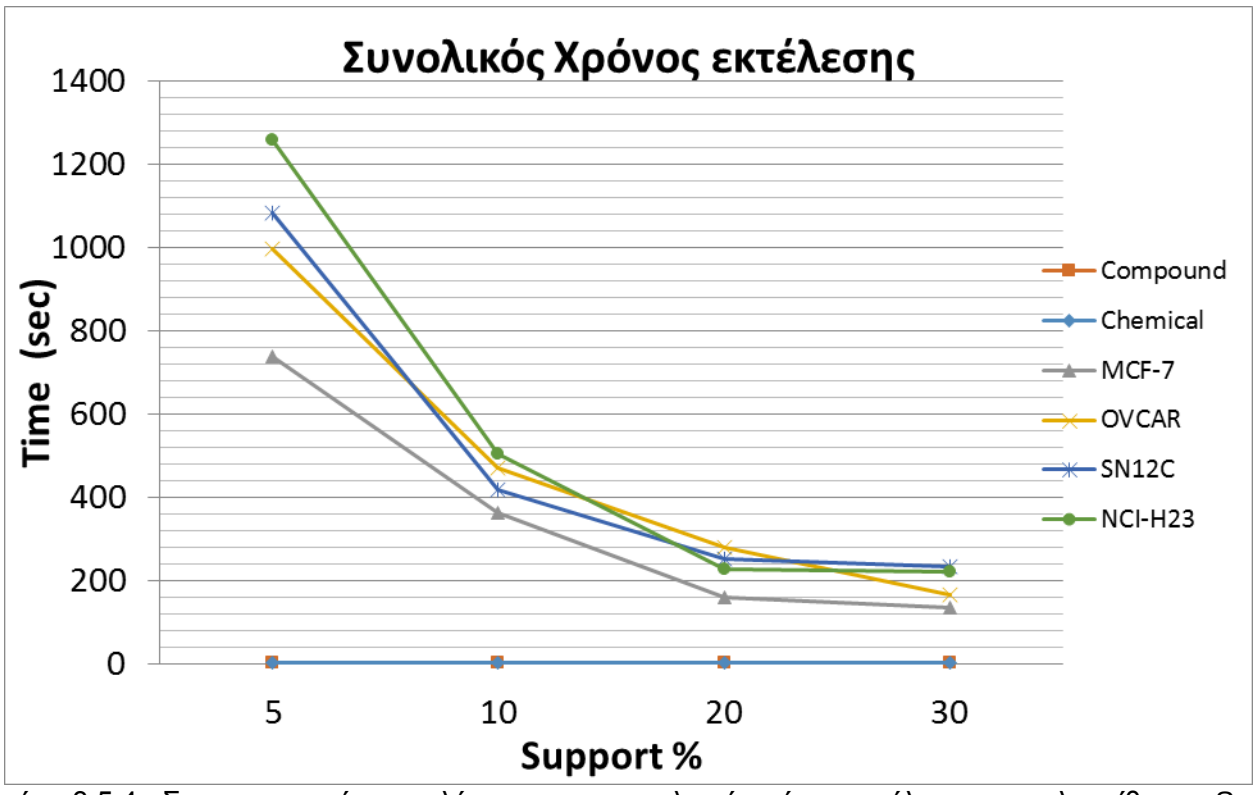


Εικόνα 3.5.2 : Καταμερισμός του χρόνου εκτέλεσης ανά συνάρτηση, με βάση τη παράμετρο support για το MCF-7 Dataset (25.475 graphs).



Εικόνα 3.5.3 : Καταμερισμός του χρόνου εκτέλεσης ανά συνάρτηση, με βάση τη παράμετρο support για το NCI-H23 Dataset (38.295 graphs).

Τα αποτελέσματα δείχνουν ότι ο συνολικός χρόνος εκτέλεσης του προγράμματος gSpan καταναλώνεται σε τρεις βασικές συναρτήσεις. Από αυτές το μεγαλύτερο ποσοστό, για διάφορες τιμές του ποσοστού συχνότητας (support) αποτελεί η συνάρτηση extend. Ακολουθεί στη συνέχεια, η συνάρτηση is_min που έβγαλαν οι profilers με ποσοστό κοντά στο 10%, και τέλος η συνάρτηση του support που υπολογίζει αν υπάρχει ο συγκεκριμένος υπογράφος περισσότερες φορές από ένα threshold. Ο υπόλοιπος κώδικας αντιστοιχεί στην δέσμευση της απαιτούμενης μνήμης καθώς και στην αρχικοποίηση των δεδομένων. Συνεπώς, προχωρήσαμε στην ανάπτυξη μίας δεύτερης αρχιτεκτονικής στην οποία αποτυπώσαμε τη λειτουργία της συνάρτησης extend, μαζί με το κομμάτι του support.



Εικόνα 3.5.4 : Συγκεντρωτικά αποτελέσματα του συνολικού χρόνου εκτέλεσης του αλγορίθμου gSpan για έξι Datasets.

ΚΕΦΑΛΑΙΟ 4: Αρχιτεκτονικές

4.1 Γενικά

Στο κεφάλαιο αυτό, παρουσιάζεται ο τρόπος με τον οποίο αναπτύχθηκε η τελική αρχιτεκτονική του συστήματος σε αναδιατασσόμενη λογική. Η αρχιτεκτονική του συστήματος λειτουργεί σε συνεργασία με έναν συνεπεξεργαστή για την υλοποίηση του αλγορίθμου gSpan στο πρόβλημα του Frequent Subgraph Mining. Με λίγα λόγια η υλοποίησή μας ακολουθεί το εξής σενάριο: ο συνεπεξεργαστής εκτελώντας μία κλήση στο hardware για έναν υπογράφο, λαμβάνει όλες τις συχνές επεκτάσεις του συγκεκριμένου υπογράφου, μέσα σε όλους τους γράφους της εισόδου. Η διαδικασία της αναζήτησης όλων των συχνών επεκτάσεων ενός γράφου καθώς επίσης και ο έλεγχος του ισομορφισμού του υπογράφου, είναι από τις πιο ακριβές διαδικασίες του αλγορίθμου, όπως αναφέραμε στο προηγούμενο κεφάλαιο. Οι δύο διαδικασίες (is_min-extension) έχουν όμοια δομή, επειδή χρησιμοποιούν τις ίδιες συναρτήσεις. Γι'αυτό το λόγο, υλοποιήσαμε πρώτα τη διαδικασία του is_min και στη συνέχεια, κάνοντας ορισμένες μετατροπές στην αρχιτεκτονική του is_min, καταφέραμε να υλοποιήσουμε την αρχιτεκτονική για τη διαδικασία του extension. Με κάποιες επιπλέον προσθήκες, χρησιμοποιώντας λογική από τους διαθέσιμους πόρους της FPGA πλατφόρμας, υλοποιήσαμε και τη διαδικασία του support, η οποία διασφαλίζει ότι μία επέκταση είναι συχνή. Με την παραπάνω σχεδίαση, καταφέρνουμε να αναδείξουμε την παραλληλία που υπάρχει στα προβλήματα των γράφων και την καταλληλότητα της χρήσης μίας multi-FPGA πλατφόρμας για το πρόβλημα του Frequent Subgraph Mining. Πιο συγκεκριμένα, χρησιμοποιώντας πολλά αντίγραφα της σχεδίασης για το τμήμα του extension, καταφέρνουμε να εξετάσουμε παράλληλα την επέκταση του ίδιου υπογράφου, μέσα σε διαφορετικούς γράφους. Παρακάτω, παρουσιάζεται κάθε τμήμα της αρχιτεκτονικής που σχεδιάστηκε από το τμήμα is_min και όλα τα υποτμήματά του, μέχρι και το τμήμα του extension και την ένωσή τους για την τελική αρχιτεκτονική του συστήματος, όπως αυτή υλοποιήθηκε με τη χρήση μίας multi-FPGA πλατφόρμας.

4.2 Αρχιτεκτονική τμήματος is_min

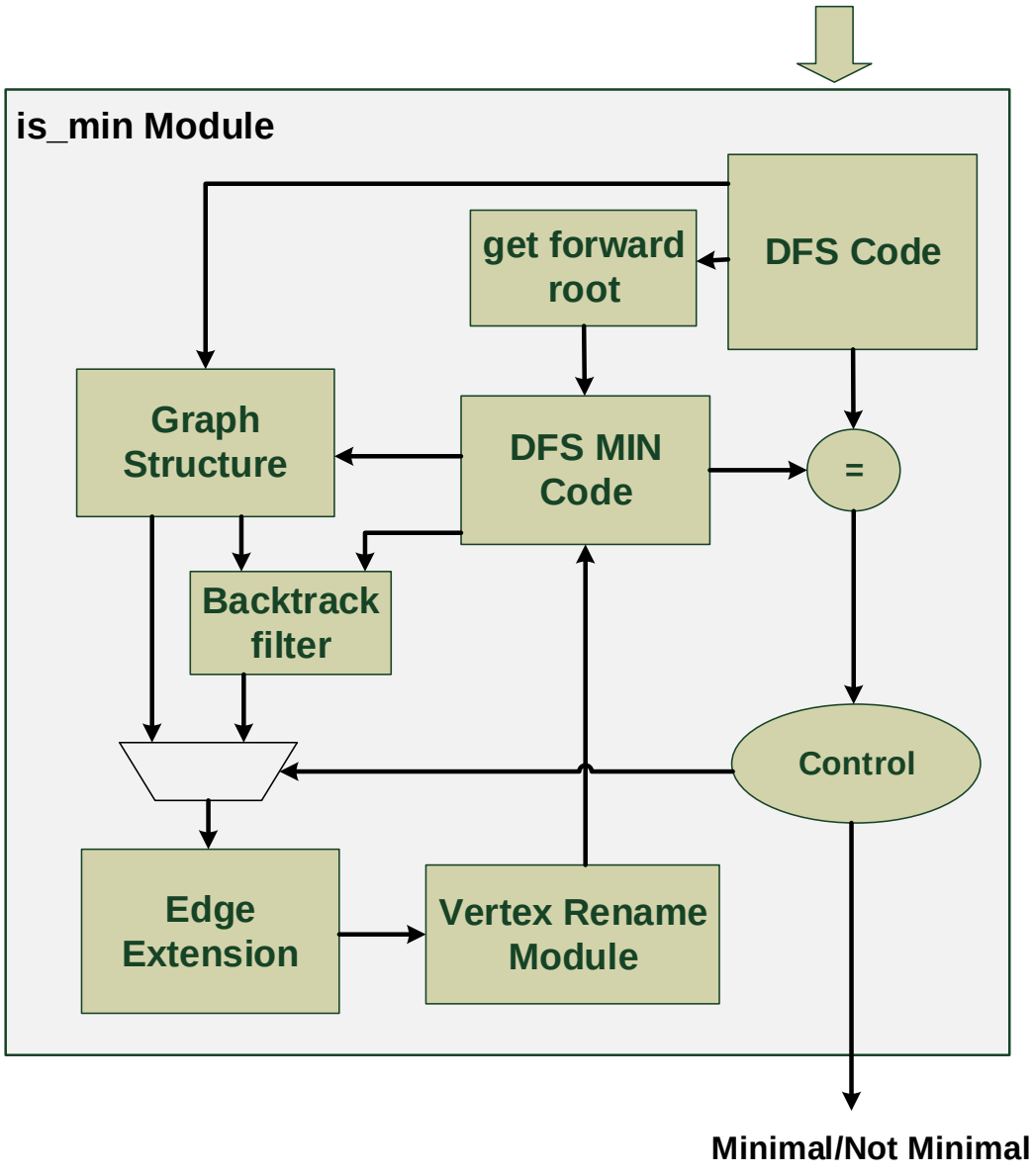
Αρχικά περιγράφεται η αρχιτεκτονική του τμήματος is_min, η οποία απεικονίζεται στην εικόνα 4.2. Αυτό το module είναι υπεύθυνο για τον έλεγχο ισομορφισμού, ο οποίος πραγματοποιείται για κάθε πιθανή προέκταση ενός υπογράφου. Το τμήμα αυτό, δέχεται ως είσοδο τις ακμές του εξεταζόμενου γράφου και δημιουργεί τις λεξικογραφικά μικρότερες δυνατές συνδέσεις μεταξύ αυτών. Αν ο γράφος που προέκυψε από τη σχεδίαση είναι λεξικογραφικά ίσος με τον αρχικό γράφο που δόθηκε στην είσοδο, τότε επιτρέπεται η περαιτέρω προέκταση του τελευταίου. Αλλιώς, η διαδικασία της επέκτασης σταματάει αμέσως. Με αυτό τον τρόπο ο αλγόριθμος κόβει γρήγορα, την επέκταση των μονοπατιών, τα οποία έχουν ήδη εξεταστεί ή θα εξετασθούν στη συνέχεια.

Προκειμένου να αποφασιστεί το αποτέλεσμα της εξέτασης του ισομορφισμού ενός γράφου, η προτεινόμενη σχεδίαση εξετάζει εξαντλητικά όλα τα πιθανά μονοπάτια που μπορούν να σχηματιστούν από τον γράφο της εισόδου. Όπως αναφέρεται και στο βιβλίο [45], η Brute force τεχνική είναι πολύ αποδοτική στην διαδικασία depth-first αναζήτησης σε ένα δέντρο. Με αυτό τον τρόπο, δεδομένο ότι ο γράφος δεν έχει αρχικούς κόμβους, κάθε κόμβος μπορεί να θεωρηθεί ως ρίζα και επομένως μπορούν να σχηματιστούν πολλά δέντρα προς εξέταση. Μέσα στο τμήμα is_min, γίνεται μία προεργασία με την οποία εξετάζονται οι πιθανές αρχικές ρίζες όλων των δέντρων. Η διαδικασία αυτή γίνεται στο τμήμα get_forward_root, από το οποίο προκύπτει η ακμή μπαίνει πρώτη μέσα στη στοίβα DFS_MIN_CODE. Ταυτόχρονα με την εξέταση για την πρώτη ακμή, όλες οι ακμές περνούν στο τμήμα Graph_Structure, στο οποίο κρατούνται όλες οι χρήσιμες πληροφορίες σχετικά με τον γράφο εισόδου. Στη συνέχεια, η μονάδα του CONTROL εξετάζει να δει αν η πρώτη ακμή είναι η ίδια λεξικογραφικά με αυτήν που προέκυψε από το get_forward_root τμήμα. Αν οι ακμές έχουν ίσα βάρη, τότε η διαδικασία της περαιτέρω επέκτασης συνεχίζεται με το τμήμα Edge_Extension να είναι υπεύθυνο για την εύρεση της

επόμενης ακμής και το τμήμα Rename_Vertex να κάνει την μετονομασία των κόμβων, προτού η ακμή τοποθετηθεί στη στοίβα DFS_MIN CODE. Διαφορετικά, στην περίπτωση που έχει προκύψει μία ακμή με μικρότερο βάρος τότε τερματίζει με αποτέλεσμα Not Minimal, γιατί ο γράφος εισόδου δεν είναι ο ελάχιστος λεξικογραφικά δυνατός. Το βάρος της κάθε ακμής εξετάζεται με προτεραιότητα ως προς το βάρος του κόμβου εκκίνησης(from_label), το βάρος της ακμής(edge_label) και το βάρος του κόμβου προορισμού(to_label).

Μόλις ένα μονοπάτι δεν μπορεί να επεκταθεί πλέον, τότε ενεργοποιείται η λειτουργία Backtrack από τη μονάδα του CONTROL. Σε αυτή την λειτουργία, όλες οι ακμές που είναι στη στοίβα του DFS_MIN_CODE και δεν έχουν δεύτερη πιθανή προέκταση επιστρέφουν στον DOUBLE BUFFER, το οποίος βρίσκεται στο Graph_Structure module. Μόλις βρεθούν ακμές ακμές με τα ίδια ακριβώς χαρακτηριστικά (labels, ids κόμβων) τότε περνούν από το τμήμα του Backtrack_Filter και εξετάζονται από το τμήμα Edge_Extension. Με αυτό τον τρόπο, υλοποιείται η εξαντλητική αναζήτηση όλων των μονοπατιών, η οποία αναφέρθηκε προηγουμένως. Η συγκεκριμένη τεχνική επιλέχθηκε για δύο λόγους. Ο πρώτος είναι ότι σύμφωνα με τον αλγόριθμο του Ullmann, η τεχνική του backtracking μειώνει σημαντικά το χώρο αναζήτησης[46]. Ο δεύτερος λόγος είναι, ότι η συγκεκριμένη τεχνική σε συνδυασμό με την δομή δεδομένων που χρησιμοποιήσαμε στην αρχιτεκτονική μας, η οποία είναι η στοίβα, μπορούν εύκολα να καλύψουν την αναδρομική λειτουργία του αλγορίθμου στο software.

DFS_CODE graph				
from_id	to_id	from_label	edge_label	to_label
0	1	0	0	1
0	2	-1	0	1
0	3	-1	3	0
3	4	-1	0	1
3	5	-1	3	0
5	6	-1	3	0
6	7	-1	3	0
7	0	-1	3	-1
7	8	-1	0	1
7	9	-1	0	1

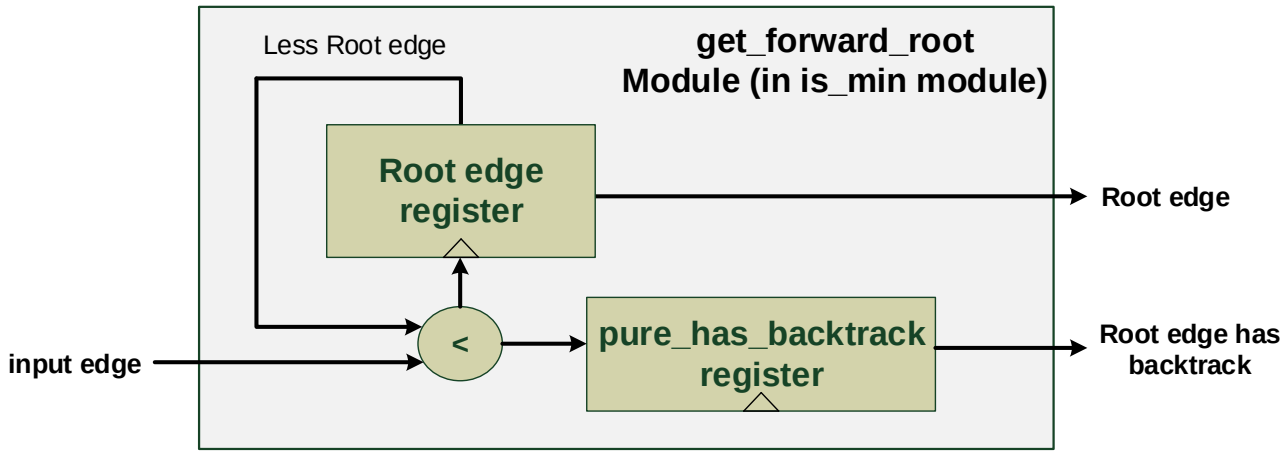


Εικόνα 4.2 : Αρχιτεκτονική του τμήματος is_min.

4.2.1 Τμήμα get_forward_root

Αυτό το module είναι υπεύθυνο για την αναζήτηση της πρώτης ακμής που θα μπει στη στοίβα του DFS_CODE_MIN. Το κριτήριο με βάση το οποίο επιλέγεται η πρώτη ακμή είναι το βάρος της. Ως βάρος κάθε ακμής, όπως αναφέρθηκε παραπάνω, είναι ο συνδυασμός με σειρά προτεραιότητας του βάρους του κόμβου εκκίνησης, στη συνέχεια του βάρους της ακμής και στο τέλος του βάρους του κόμβου προορισμού.

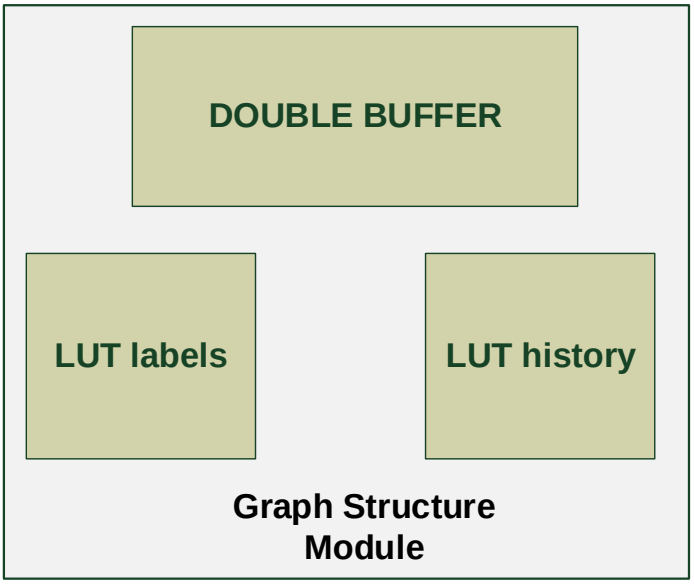
Το τμήμα αυτό είναι ενεργό ταυτόχρονα με το γέμισμα του τμήματος Graph_Structure. Επομένως, όσο έρχονται καινούργια δεδομένα εισόδου, αυτά φιλτράρονται στο τμήμα get_forward_root και στη συνέχεια τοποθετείται η ελάχιστη ακμή στη στοίβα DFS_CODE_MIN. Το φίλτρο που χρησιμοποιείται σε αυτό το τμήμα, έχει ένα ακόμη χαρακτηριστικό. Σε περίπτωση που υπάρχουν παραπάνω από μία αρχικές ακμές που είναι λεξικογραφικά ισοδύναμες, τότε γράφεται μαζί με την ακμή ένα bit επιπλέον το οποίο προσδιορίζει ότι υπάρχει διαφορετικό μονοπάτι για την συγκεκριμένη ακμή. Η πληροφορία αυτή θα χρειαστεί, μόλις ενεργοποιηθεί η διαδικασία Backtrack-αναδρομή στο κύκλωμα ώστε να ελεγχθούν όλα τα δυνατά μονοπάτια. Η παραπάνω διαδικασία φαίνεται σχηματικά στο σχήμα 4.2.1.



Σχήμα 4.2.1 : Αρχιτεκτονική τμήματος get_forward_rmpath (στο τμήμα is_min).

4.2.2 Τμήμα Graph_Structure

Το τμήμα Graph_Structure είναι υπεύθυνο για τη συγκέντρωση όλης της χρήσιμης πληροφορίας του γράφου εισόδου, την οποία χρειάζεται η υπόλοιπη σχεδίαση για την σωστή λειτουργία της. Το τμήμα αυτό, αποτελείται από τρείς βασικές δομές δεδομένων: τον DOUBLE BUFFER, το LUT_LABELS και το LUT_HISTORY, όπως φαίνεται στην Εικόνα 4.2.2.1.

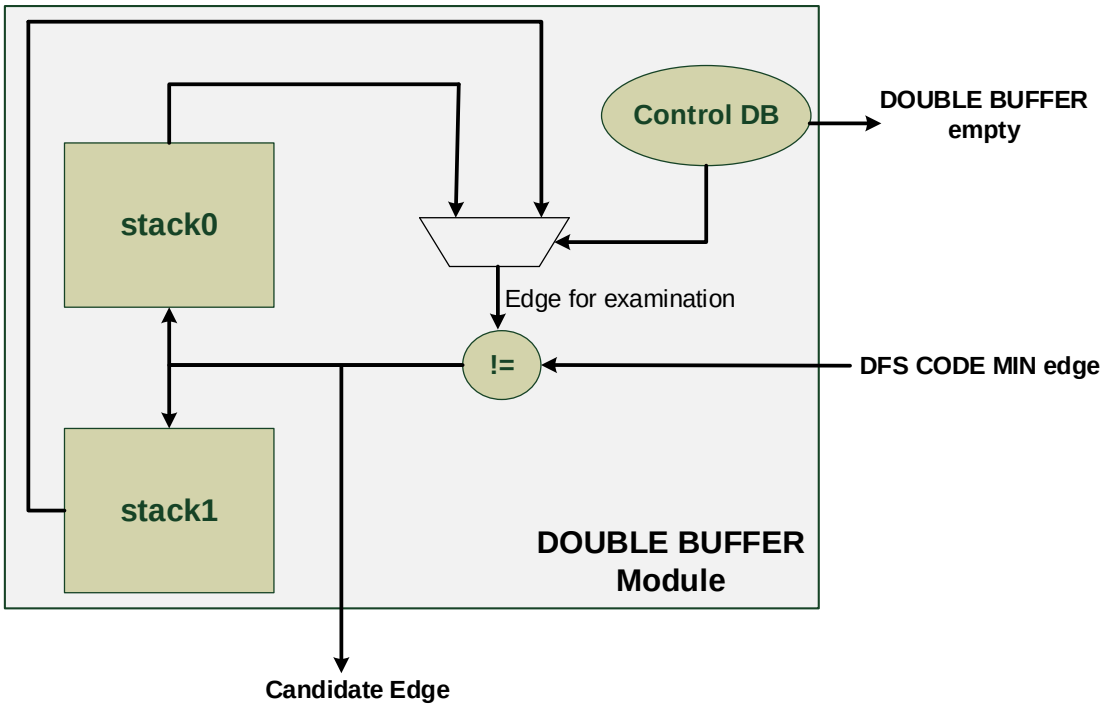


Εικόνα 4.2.2.1 : Σχηματική περιγραφή των δομών δεδομένων, που περιλαμβάνονται στο τμήμα Graph_Structure.

Στην αυθεντική υλοποίηση του αλγορίθμου gSpan στη γλώσσα C++, γίνεται χρήση των vectors, καθώς επίσης και της δομής Adjacency List και της Linked List. Η επιλογή των vectors, έγινε με σκοπό την ομαδοποίηση των εμφανίσεων κάθε ακμής που περιέχει τα ίδια βάρη. Με αυτόν τον τρόπο, ο αλγόριθμος εξετάζει επαναληπτικά μια ακμή η οποία μπορεί να είναι λεξικογραφικά ισοδύναμη, με πολλές άλλες ακμές μέσα σε διαφορετικούς γράφους, αλλά ακόμα και στον ίδιο γράφο εισόδου. Τα περιεχόμενα των vectors, περιέχουν τη Linked List, η οποία περιλαμβάνει όλες τις συνδεδεμένες ακμές από τη ρίζα του μονοπατιού μέχρι το φύλλο, καθώς επίσης και την πληροφορία σε ποιον γράφο ανήκει. Είναι εύκολο, να κατανοήσει κανείς ότι η συγκεκριμένη υλοποίηση είναι σχετικά εύκολη, σε επίπεδο C++ και ταυτόχρονα ακατόρθωτη σε επίπεδο hardware. Επιπλέον στο software, γίνεται χρήση δυναμικής μνήμης, και ο χώρος για τη μνήμη που χρειάζεται είναι πολύ μεγάλος, για την υλοποίηση σε μια FPGA. Για το λόγο αυτό, η αρχιτεκτονική που σχεδιάστηκε σε αυτή την διπλωματική εργασία, άλλαξε την αλγοριθμική δομή που χρησιμοποιείται από το software για την εξέταση των επεκτάσεων και τον έλεγχο του ισομορφισμού των υπογράφων. Στη νέα αλγοριθμική υλοποίηση, έγινε χρήση μιας καινοτόμου δομής δεδόμενων, ως προς τη γνώση μας. Η δομή αυτή καλείται DOUBLE BUFFER, και χρησιμοποιεί δυο στοίβες, για να υλοποιήσει την τεχνική του Double Buffering. Το όφελος της συγκεκριμένης δομής είναι ότι η πρώτη στοίβα τροφοδοτείται αρχικά με όλες τις ακμές που υπάρχουν σε ένα γράφο, και στη συνέχεια σε κάθε κύκλο εξάγει όλες τις υποψήφιες ακμές, τις οποίες τις γράφει στη δεύτερη στοίβα. Μόλις επιλεχθεί μια από αυτές τις ακμές ως κατάλληλη για το χτίσιμο του γράφου, τότε γράφονται στην πρώτη στοίβα, όλες οι ακμές της δεύτερης εκτός από αυτήν. Έτσι, όσο επεκτείνεται το επίπεδο επέκτασης του υπογράφου, τόσο μειώνεται ο αριθμός των υποψήφιων ακμών. Στο software, η αντίστοιχη μείωση, για να αποφευχθεί η διπλή χρησιμοποίηση κάποιας ακμής, γίνεται με τη συνάρτηση history, η οποία καταναλώνει σημαντικό μέρος του συνολικού χρόνου εκτέλεσης.

Ο DOUBLE BUFFER, που περιγράψαμε φαίνεται σχηματικά στο σχήμα 4.2.2.2. Πιο συγκεκριμένα οι δύο στοίβες είναι ίδιας διάστασης (1024x48) και επίσης χρησιμοποιείται ένα φίλτρο που εξετάζει τις ακμές που γράφονται σε κάθε μία στοίβα εναλλάξ και μία μονάδα ελέγχου, η οποία συντονίζει όλη την διαδικασία. Η δομή του DOUBLE BUFFER είναι υπεύθυνη για να εξάγει τις εναπομείναντες ακμές που δεν έχουν μπει ακόμη στη στοίβα του DFS_CODE_MIN. Η διαδικασία αυτή της εξαγωγής ενεργοποιείται από τη μονάδα του Control DB και γίνεται κάθε φορά εξάγοντας τις ακμές της γεμάτης στοίβας και περνώντας τες από το φίλτρο. Το φίλτρο αυτό είναι υπεύθυνο να κόψει την ακμή που μπήκε πιο πρόσφατα στη στοίβα DFS_CODE_MIN και να περάσει τις υπόλοιπες ακμές, ώστε να

γραφτούν στην άλλη στοίβα και να συνεχιστεί η διαδικασία επέκτασης. Με αυτό τον τρόπο, μειώνεται κάθε φορά το πλήθος των υποψήφιων ακμών για να μπούν στη στοίβα DFS_CODE_MIN.



Σχήμα 4.2.2.2 : Αρχιτεκτονική του τμήματος DOUBLE BUFFER, τον οποίο υλοποιήσαμε με τη χρήση της δομής της στοίβας.

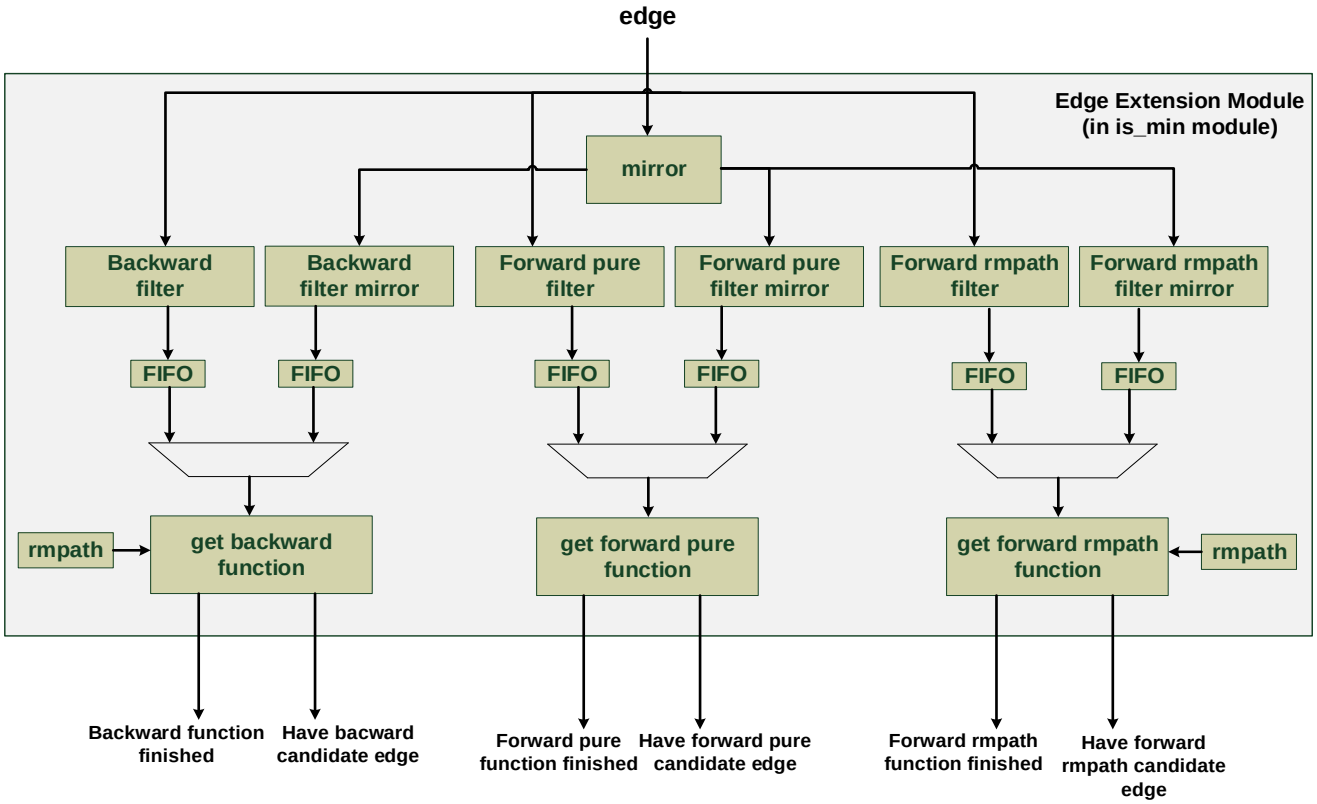
Οι υπόλοιπες δομές δεδομένων που χρησιμοποιήθηκαν είναι δύο Lookup tables. Η πρώτη δομή καλείται LUT_LABELS, είναι true dual port ram με διαστάσεις(1024x16) και κρατάει πληροφορία για τα βάρη όλων των κόμβων του γράφου εισόδου. Η δεύτερη δομή καλείται LUT_HISTORY, είναι επίσης true dual port ram με διαστάσεις(1024x1) και κρατάει το ιστορικό όλων των κόμβων κατά τη διαδικασία της προέκτασης. Και οι δύο αυτές δομές, προσπελαύνονται από το τμήμα Edge_Extension και παίζουν καθοριστικό ρόλο στη διαδικασία επέκτασης κάθε μονοπατιού. Στην υλοποίηση του αλγορίθμου επιλέχτηκε, το σύστημα να μπορεί να δεχθεί δεδομένα που έχουν μέχρι 1024 κόμβους και 1024 ακμές, καθώς το συγκεκριμένο πρόβλημα της εξόρυξης των συχνών υπογράφων δέχεται μία πολύ μεγάλη ποσότητα από σχετικά μικρούς γράφους. Επίσης, η χρησιμοποίηση της δεύτερης πόρτας των μνημων, ήταν πολύ βολική καθώς μπορούσε να ζητηθεί ταυτόχρονα από την ίδια μνήμη, πληροφορία και για τους δύο κόμβους μίας ακμής.

4.2.3 Τμήμα Edge_Extension

Αυτό το τμήμα είναι υπεύθυνο να εξετάζει όλες τις ακμές που του έρχονται είτε από τη δομή δεδομένων του DOUBLE BUFFER μέσα στο τμήμα Graph_Structure, είτε από το Backtrack_Filter και να εξάγει την επόμενη προεκτεινόμενη ακμή. Η διαδικασία της προέκτασης του γράφου, γίνεται με βάση τα τρία κριτήρια επέκτασης του αλγορίθμου gSpan, όπως αυτά περιγράφονται στο κεφάλαιο 3 και φαίνονται στην εικόνα 4.2.3.

Στη προτεινόμενη σχεδίαση, η παραπάνω λειτουργία υλοποιείται με την χρήση έξι φίλτρων, δύο για κάθε διαφορετικό είδος προέκτασης και τρία τμήματα, τα οποία εξετάζουν παράλληλα τις ακμές που πέρασαν από τα φίλτρα τους και κρατούν τη μικρότερη λεξικογραφικά ακμή. Κάθε είδος προέκτασης, χρειάζεται δύο φίλτρα, τα οποία θα εξετάζουν ταυτόχρονα την ίδια ακμή για τις δύο κατευθύνσεις της. Με αυτό τον τρόπο, η υλοποίηση που προτείνουμε υποστηρίζει την εξέταση και επέκταση μην κατευθυντικών γράφων. Κάθε ένα από τα δύο φίλτρα και των τριών ειδών, εξετάζει την εισερχόμενη ακμή και την καθρεφτική της. Τα φίλτρα χρησιμοποίησαν μέρος της λογικής των τριών συναρτήσεων, προκειμένου να γράψουν τις υποψήφιες ακμές στις ουρές (FIFOs) που ακολουθούν. Με αυτό τον τρόπο, καταφέραμε να μειώσουμε σημαντικά, τον αριθμό των εξεταζόμενων ακμών από κάθε μία συνάρτηση. Στη συνέχεια για όσο διάστημα τα δύο φίλτρα είναι γεμάτα, κάθε τμήμα που αντιστοιχεί σε μία από τις τρεις συναρτήσεις, είναι υπεύθυνο να διαβάσει και να εξετάσει όλες τις ακμές μέχρι να αδειάσουν. Μόλις ολοκληρωθεί και η εξέταση των ακμών από κάθε τμήμα, εκείνο ενημερώνει την μονάδα ελέγχου CONTROL και το τμήμα που είναι υπεύθυνο για την μετονομασία των κόμβων Rename_Vertex, για την ύπαρξη ακμής του συγκεκριμένου είδους επέκτασης καθώς επίσης και για την ολοκλήρωση της όλης διαδικασίας.

Στις επόμενες ενότητες περιγράφονται αναλυτικά όλα τα υποτμήματα, καθώς και οι σχεδιαστικές προδιαγραφές με τις οποίες σχεδιάστηκαν.

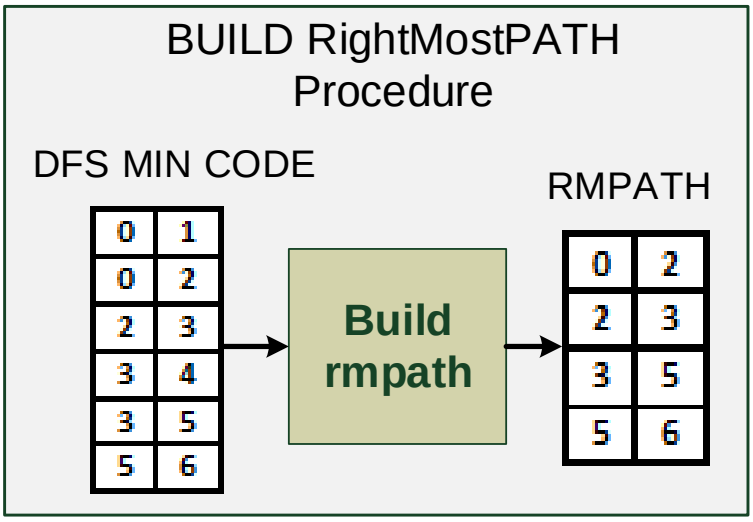


Εικόνα 4.2.3 : Αρχιτεκτονική του τμήματος Edge_Extension (μέσα στο τμήμα is_min).

4.2.3.1 Τμήμα Build_rmpath

Μία σημαντική αρχή του gSpan αλγορίθμου, όπως αναφέραμε και στο κεφάλαιο 3, είναι το κριτήριο με βάση το οποίο γίνεται η επέκταση των γράφων. Το κριτήριο αυτό είναι η επέκταση πάνω στη ραχοκοκαλιά του γράφου, η οποία αναφέρεται στον αλγόριθμο ως rightmost path[9]. Επομένως, καθίσταται εξαιρετικά σημαντική η πληροφορία του rightmost path. Η πληροφορία αυτή δημιουργείται από το τμήμα του Build_rmpath και κρατείται σε μία στοίβα, η οποία είναι μέσα στο τμήμα Edge_Extension και καλείται rmpath.

Για κάθε αλλαγή που παρατηρείται στη στοίβα DFS_CODE_MIN, μεταβάλεται και το rightmost path. Οι μεταβολές αυτές πραγματοποιούνται από το τμήμα Build_rmpath, το οποίο αδειάζει και ξαναγεμίζει τη στοίβα κάθε φορά. Η διαδικασία με την οποία χτίζεται το rightmost path κάθε φορά είναι η ακόλουθη. Κάθε ακμή εξάγεται από τη στοίβα DFS_CODE_MIN και οδηγείται στο Build_rmpath τμήμα. Εκεί το τμήμα αυτό τοποθετεί στη στοίβα rmpath την τελευταία ακμή που είχε μπει στη στοίβα DFS_CODE_MIN, δηλαδή την πιο πρόσφατη. Και στην συνέχεια συνεχίζει τη στοίβαξη των ακμών, με τέτοιο τρόπο, ώστε να μπαίνει η αμέσως επόμενη δυνατή σύνδεση, με τη σειρά που βγαίνουν από τη στοίβα DFS_CODE_MIN. Ένα παράδειγμα φαίνεται στην εικόνα 4.2.3.1. Η μνήμη, η οποία κρατάει την πληροφορία του rightmost path είναι μία single port block ram με διαστάσεις (256x80). Το πλάτος είναι 80 bits, επειδή κρατάει και την πληροφορία της νέας και της παλιάς ονομασίας, η οποία χρειάζεται κατά την εξέταση μίας backward και μίας forward_rmpath ακμής.



Εικόνα 4.2.3.1 : Σχηματική περιγραφή της λειτουργίας του τμήματος Build_rmpath.

4.2.3.2 Τμήμα Filter_Backward

Αυτό το φίλτρο είναι υπεύθυνο για την επιλογή εκείνων των ακμών μόνο που ξεκινούν από τον τελευταίο κόμβο του rmpath και έχουν κόμβο προορισμού, μέσα στο rmpath.

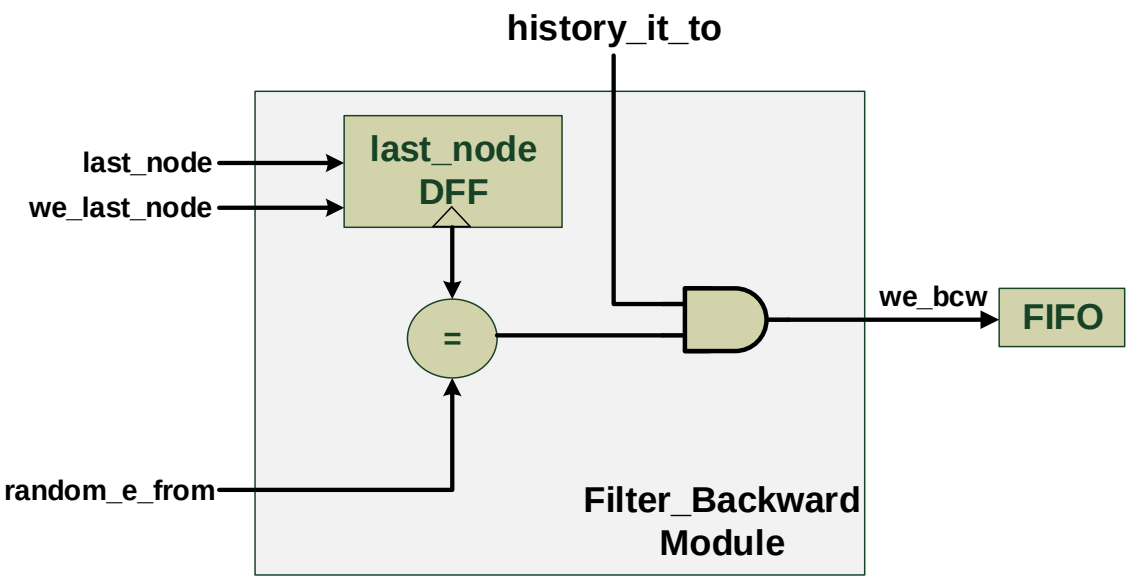
Η ενεργοποίηση της μονάδας του Filter_Backward γίνεται από τη μονάδα του CONTROL

Μέσα στο φίλτρο, υπάρχει ένας καταχωρητής, ο οποίος κρατάει κάθε φορά που γράφεται μία νέα ακμή στη στοίβα DFS_CODE_MIN, τον τελευταίο κόμβο του rmpath. Η έξοδος του καταχωρητή, συνδέεται στη μία είσοδο ενός συγκριτή(comparator). Στην άλλη είσοδο, συνδέεται ο κόμβος εκκίνησης της εισερχόμενης ακμής. Το αποτέλεσμα είναι ένα σήμα 1 bit, το οποίο ενεργοποιείται μόνο αν ισχύει η ισότητα των δύο εισόδων.

Η δεύτερη συνθήκη που πρέπει να ικανοποιείται είναι η πληροφορία της ύπαρξης του κόμβου προορισμού της εισερχόμενης ακμής, μέσα στο rmpath. Η πληροφορία αυτή προκύπτει από ένα

lookup table (LUT_HISTORY).

Η αρχιτεκτονική του Filter_Backward είναι pipelined και μπορεί να εξυπηρετούνται εισερχόμενες ακμές ανά κύκλο ρολογιού. Η παραπάνω σχεδίαση φαίνεται στο σχήμα 4.2.3.2.



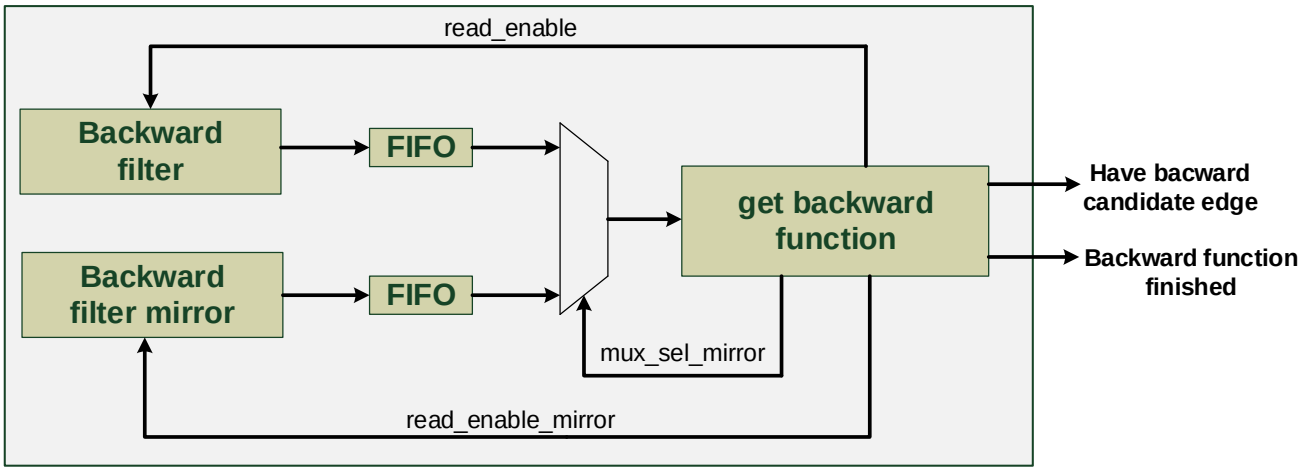
Σχήμα 4.2.3.2 : Αρχιτεκτονική τμήματος Filter_Backward.

4.2.3.3 Τμήμα Filter_Backward_mirror

Η λειτουργία του συγκεκριμένου τμήματος έχει να κάνει με την εξέταση των καθρεφτικών ακμών, δηλαδή των ακμών που έρχονται από τη δομή του DOUBLE BUFFER και έχουν την αντίθετη κατεύθυνση. Οι συνθήκες που πρέπει να ικανοποιούνται για να γραφτεί η εισερχόμενη καθρεφτική ακμή στη δεύτερη ουρά(fifo) του τμήματος Backward, είναι οι ίδιες με την ενότητα 4.2.3.2. Το μόνο που αλλάζει είναι το αποτέλεσμα για το ιστορικό του κόμβου εκκίνησης αυτή τη φορά, το οποίο προκύπτει από την δεύτερη πόρτα του LUT_HISTORY.

4.2.3.4 Τμήμα Backward

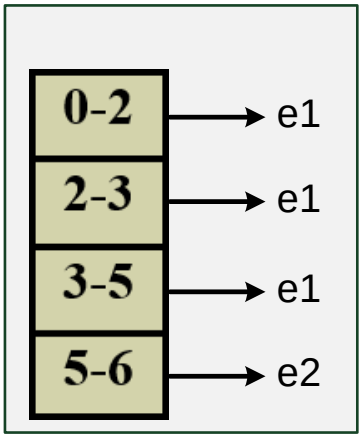
Το τμήμα Backward είναι υπεύθυνο για την εξέταση όλων των ακμών που έχουν γραφτεί στα δύο φίλτρα του. Η εξέταση των ακμών γίνεται στο τμήμα Backward_examination, το οποίο βρίσκεται εντός του τμήματος get backward function και στο οποίο εισέρχονται εκτός των άλλων και οι ακμές της στοίβας Backward_rmpath. Ο συγχρονισμός, προκειμένου να εξαχθεί ένα στοιχείο από τη στοίβα του Backward_rmpath, καθώς επίσης και η διαδικασία του διαβάσματος των εισερχόμενων ακμών από τα δύο φίλτρα, γίνεται με τη βοήθεια μίας μονάδας ελέγχου με όνομα Control_backward.



Εικόνα 4.2.3.4 : Σχηματική αναπαράσταση του τμήματος Forward_Pure.

4.2.3.4.1 Στοίβα Backward_rmpath

Η στοίβα Backward_rmpath, περιλαμβάνει όλες τις ακμές και τους κόμβους που βρίσκονται μέσα στο rightmost path του εξεταζόμενου γράφου. Η στοίβα είναι στην ουσία μία true dual-port block Ram (1024x80bits), την οποία διαχειρίζεται ένας Memory Controller με όμοιο τρόπο, σε σχέση με την υλοποίηση στο software. Η συγκεκριμένη ιδιατερότητα έχει να κάνει με τον τρόπο που διαχειρίζεται το τμήμα Bacward_examination τις ακμές που εξάγονται κάθε φορά από τη μνήμη Backward_rmpath. Πιο συγκεκριμένα η έξοδος της πρώτης πόρτας (e1), βγάζει όλες τις ακμές της μνήμης που έχουν στοιβαχθεί από την πρώτη που μπήκε στην μνήμη, μέχρι την προτελευταία. Ενώ η έξοδος της δεύτερης πόρτας (e2) βγάζει την πιο πρόσφατη ακμή, που μπήκε στην μνήμη. Η παραπάνω λειτουργία της μνήμης, μαρτυρά γιατί έγινε αναφορά στη δομή της στοίβας. Ένα παράδειγμα δίνεται στην παρακάτω εικόνα.



Σχήμα 4.2.3.4.1: Ένα παράδειγμα, για τη λειτουργικότητα της dual-port ram. Οι ακμές τύπου e1 εξάγονται μία ανα κύκλο από την πρώτη πόρτα, ενώ η τελευταία ακμή του rmpath εξάγεται από τη δεύτερη πόρτα. Αυτό γίνεται, επειδή στο τμήμα Backward_examination γίνεται έλεγχος αν η υποψήφια ακμή ξεκινάει από τον κόμβο προορισμού της e2 (6) και καταλήγει σε κάποιον από τους κόμβους εκκίνησης των e1 (0,2,3).

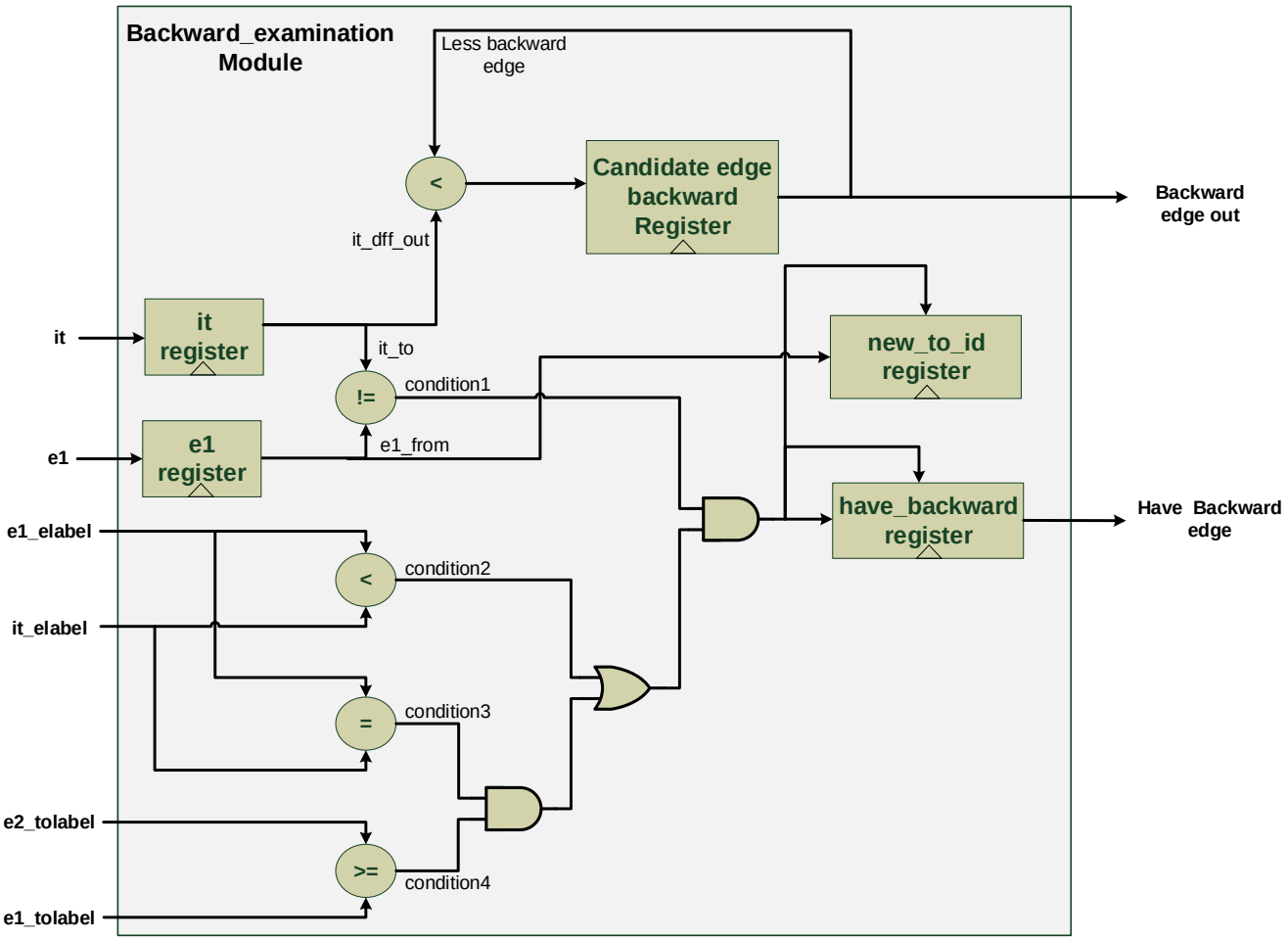
4.2.3.4.2 Τμήμα Backward_examination

Το module αυτό εξετάζει κάθε φορά μία εισερχόμενη ακμή από τα φίλτρα μαζί με όλες τις ακμές που εξάγονται από τη στοίβα rmpath.

Αρχικά διαβάζεται η πρώτη ακμή από το rmpath,πχ. e1. Η εισερχόμενη ακμή γράφεται σε έναν καταχωρητή, μόνο αν ικανοποιούνται δύο συνθήκες. Η πρώτη είναι να έχει κόμβο προορισμού, ο οποίος είναι ο κόμβος εκκίνησης της e1 και η δεύτερη, το βάρος της εισερχόμενης ακμής να είναι μεγαλύτερο από το βάρος της e1 ακμής, ή να είναι ίσο και να ισχύει επίσης ότι το βάρος του κόμβου εκκίνησης της εισερχόμενης ακμής είναι μεγαλύτερο ή ίσο με εκείνο του κόμβου προορισμού της e1 ακμής. Τα βάρη των κόμβων προκύπτουν από το lookup table LUT_LABELS.

Στη συνέχεια η έξοδος του καταχωρητή περνάει σε έναν συγκριτή μαζί με τα κατάλληλα βάρη των κόμβων και της ακμής και εξετάζεται αν είναι η μικρότερη λεξικογραφικά ακμή τύπου backward. Στο συγκριτή αυτό γίνεται μία επιπλέον λειτουργία, καθώς από τη φύση του αλγορίθμου δίνεται προτεραιότητα σε εκείνες τις ακμές που πηγαίνουν προς τον πιο παλιό κόμβο του rmpath. Η διαδικασία επαναλαμβάνεται για όλες τις ακμές του rmapth. Στο τέλος, κρατείται εκείνη η ακμή, που είναι μικρότερη και ως προς τα ονόματα(id) των κόμβων. Η παραπάνω διαδικασία επαναλαμβάνεται, για όλες τις ακμές εισόδου, δηλαδή μέχρις ότου αδειάσουν και οι δύο ουρές οι οποίες αντιστοιχούν στο τμήμα Backward.

Η λειτουργία του τμήματος Backward_examination φαίνεται σχηματικά στην εικόνα 4.2.3.4.2.

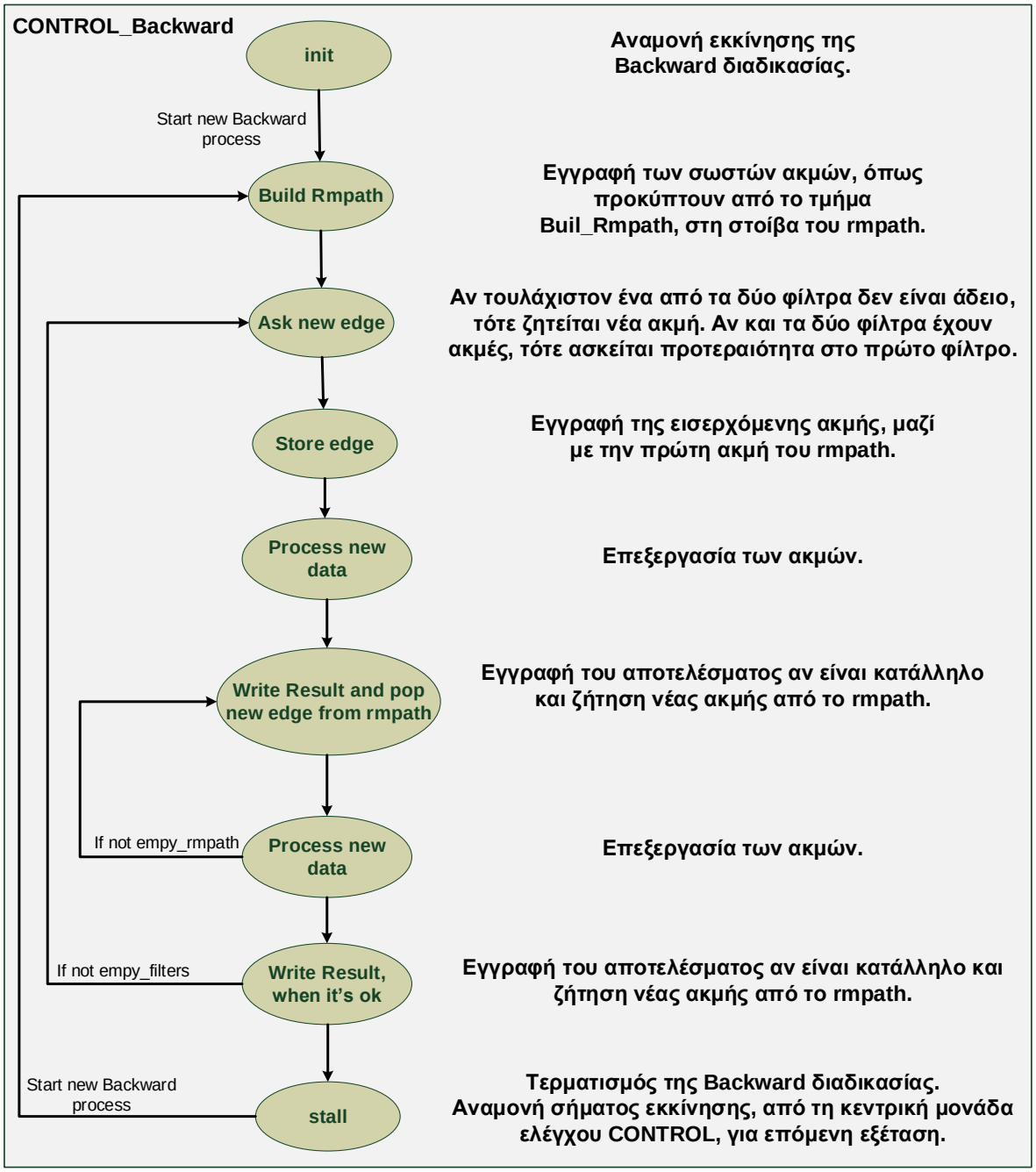


Εικόνα 4.2.3.4.2: Αρχιτεκτονική του τμήματος που εξετάζει τις υποψήφιες ακμές τύπου backward.

4.2.3.4.3 Τμήμα Control_Backward

Μόλις ολοκληρωθεί η διαδικασία εξέτασης των υποψήφιων ακμών, η μονάδα ελέγχου είναι υπεύθυνη να εξάγει τα κατάλληλα αποτελέσματα. Το ένα είναι αν βρέθηκε ακμή τύπου backward και το δεύτερο ότι ολοκληρώθηκε η διαδικασία εξέτασης των ακμών που υπήρχαν μέσα στα φίλτρα. Τα αποτελέσματα αυτά πηγαίνουν στην κεντρική μονάδα ελέγχου CONTROL καθώς και στο τμήμα της μετονομασίας των κόμβων.

Η μονάδα ελέγχου αποτελείται από ένα σύνολο πεπερασμένων καταστάσεων (Finite State Machines-FSMs), κάθεμία εκ των οποίων υλοποιεί τις λειτουργίες που περιγράφηκαν προηγουμένως. Οι καταστάσεις της μονάδας ελέχου του Control_Backward περιγράφονται στο σχήμα 4.2.3.4.3.



Σχήμα 4.2.3.4.3: Περιγραφή της FSM του τμήματος ελέγχου για το τμήμα Backward.

4.2.3.5 Τμήμα Filter_Forward_Pure

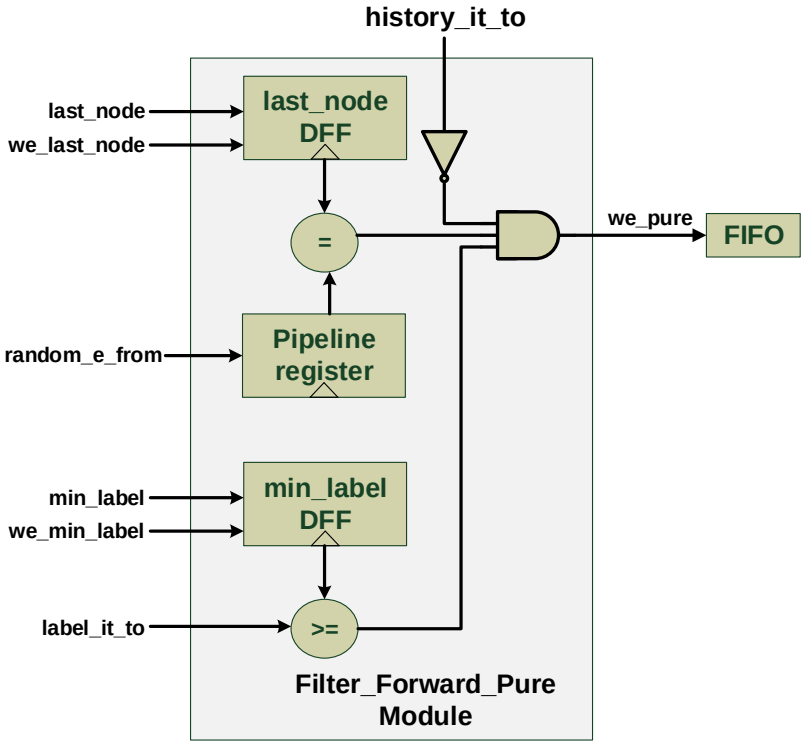
Σε αυτό το τμήμα γράφονται στην πρώτη ουρά, που αντιστοιχεί στο τμήμα του Forward_Pure, οι ακμές εκείνες που ξεκινούν από τον τελευταίο κόμβο του rmpath και έχουν κόμβο προορισμού, ο οποίος δεν είναι μέσα στο rmpath, μόλις δωθεί το αντίστοιχο σήμα ενεργοποίησης από τη μονάδα του CONTROL.

Η ενεργοποίηση της μονάδας του Filter_Forward_Pure γίνεται από τη μονάδα ελέγχου, με την ενεργοποίηση του σήματος filter_pure_active.

Μέσα στο φίλτρο, υπάρχουν δύο καταχωρητές. Ο πρώτος κρατάει κάθε φορά που γράφεται μία ακμή στη στοίβα DFS_CODE_MIN τον τελευταίο κόμβο του rmpath. Η έξοδος του καταχωρητή αυτού, συνδέεται στη μία είσοδο ενός συγκριτή. Στην άλλη είσοδο, συνδέεται ο κόμβος εκκίνησης της εισερχόμενης ακμής. Το αποτέλεσμα είναι ένα σήμα 1 bit, το οποίο ενεργοποιείται μόνο αν ισχύει η ισότητα των δύο εισόδων. Ο δεύτερος καταχωρητής κρατάει την πληροφορία του minlabel, το οποίο είναι το βάρος του κόμβου εκκίνησης της πρώτης ακμής μέσα στο DFS_CODE_MIN. Η έξοδός του πηγαίνει ως είσοδος σε ένα συγκριτή, ο οποίος ελέγχει αν το βάρος του κόμβου προορισμού κάθε εισερχόμενης ακμής είναι μεγαλύτερο ή ίσο από αυτό.

Η τρίτη συνθήκη που πρέπει να ικανοποιείται είναι η πληροφορία αν ο κόμβος προορισμού της εισερχόμενης ακμής δεν βρίσκεται μέσα στο rmpath. Η πληροφορία αυτή προκύπτει από το lookup table LUT_HISTORY.

Η αρχιτεκτονική του Filter_Forward_Pure module είναι pipelined, καθώς χρειάζεται να εξυπηρετούνται εισερχόμενες ακμές ανά κύκλο. Επειδή τα δύο lookup tables, για το ιστορικό των κόμβων και για τα βάρη τους, είναι Memories και βγάζουν το επιθυμητό αποτέλεσμα στον επόμενο κύκλο, για το λόγο αυτό τοποθετήθηκε ένας pipeline register στην αρχή πριν την πρώτη συνθήκη, ώστε να συγχρονίζει τη μεταξύ τους λειτουργία. Το Filter_Forward_Pure module φαίνεται στο σχήμα 4.2.3.5.



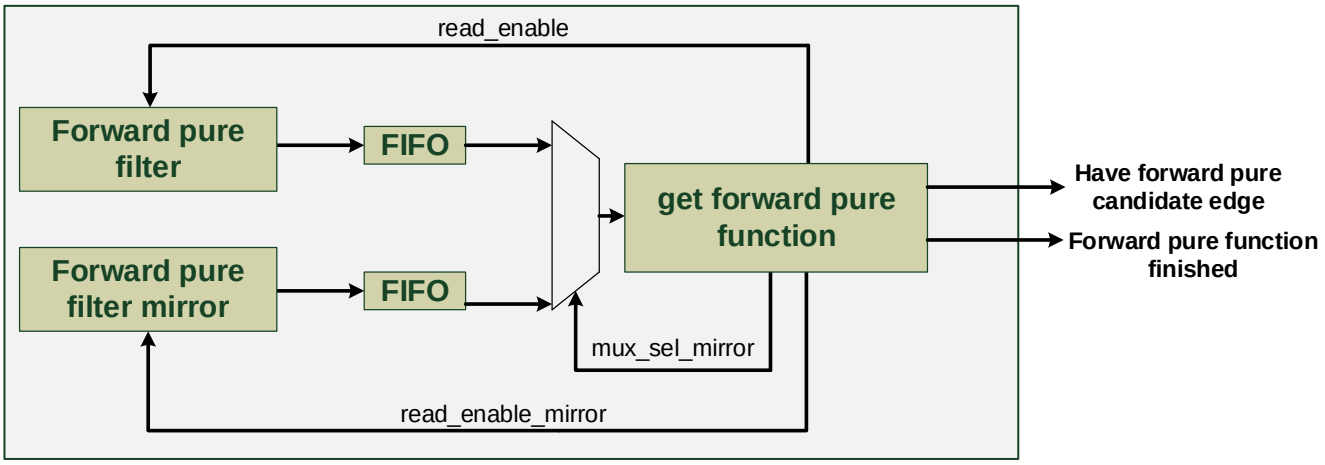
Σχήμα 4.2.3.5: Αρχιτεκτονική του φίλτρου για τις υποψήφιες ακμές, τύπου Forward_Pure.

4.2.3.6 Τμήμα Filter_Forward_Pure_mirror

Η λειτουργία του συγκεκριμένου τμήματος έχει να κάνει με την εξέταση των καθρεφτικών ακμών, οι οποίες πληρούν τις συνθήκες που καθορίζει το forward_pure είδος προέκτασης. Οι συνθήκες που πρέπει να ικανοποιούνται για να γραφτεί η εισερχόμενη καθρεφτική ακμή στη δεύτερη ουρά(fifo) του τμήματος Forward_Pure, είναι οι ίδιες με την ενότητα 4.2.3.5. Το μόνο που αλλάζει είναι το αποτέλεσμα για το ιστορικό και το βάρος του κόμβου εκκίνησης, τα οποία αυτή τη φορά προκύπτουν από την δεύτερη πόρτα καθενός Lookup table.

4.2.3.7 Τμήμα Forward_Pure

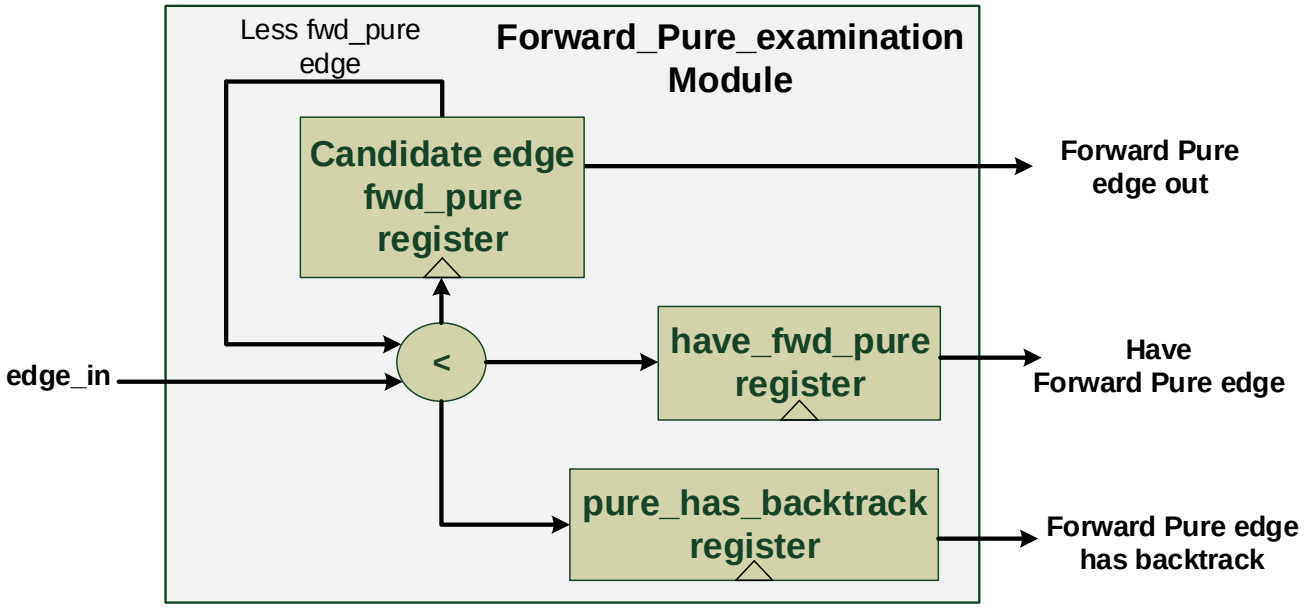
Το τμήμα Forward_Pure είναι υπεύθυνο για την εξέταση όλων των ακμών που έχουν γραφτεί στα δύο φίλτρα του. Η εξέταση των ακμών γίνεται στο τμήμα Forward_Pure_examination, το οποίο βρίσκεται εντός του τμήματος get forward pure function και στο οποίο εισέρχονται οι υποψήφιες ακμές από τα δύο φίλτρα.



Σχήμα 4.2.3.7 : Σχηματική αναπαράσταση του τμήματος Forward_Pure.

4.2.3.7.1 Τμήμα Forward_Pure_examination

Το τμήμα αυτό δέχεται ως είσοδο όλες τις ακμές που έρχονται από τα φίλτρα και κρατάει τη μικρότερη λεξικογραφικά ακμή. Σε περίπτωση που υπάρχουν πάνω από μία ίδιες λεξικογραφικά μικρές ακμές, τότε κρατείται αυτή που έχει το μικρότερο όνομα κόμβων(id) και ενεργοποιείται ένα σήμα εξόδου, που δείχνει ότι ο γράφος έφτασε σε ακμή που μπορεί να οδηγήσει σε δεύτερο μονοπάτι(Forward Pure has backtrack). Η λειτουργία του τμήματος Forward_Pure_examination φαίνεται σχηματικά στην εικόνα 4.2.3.7.1.



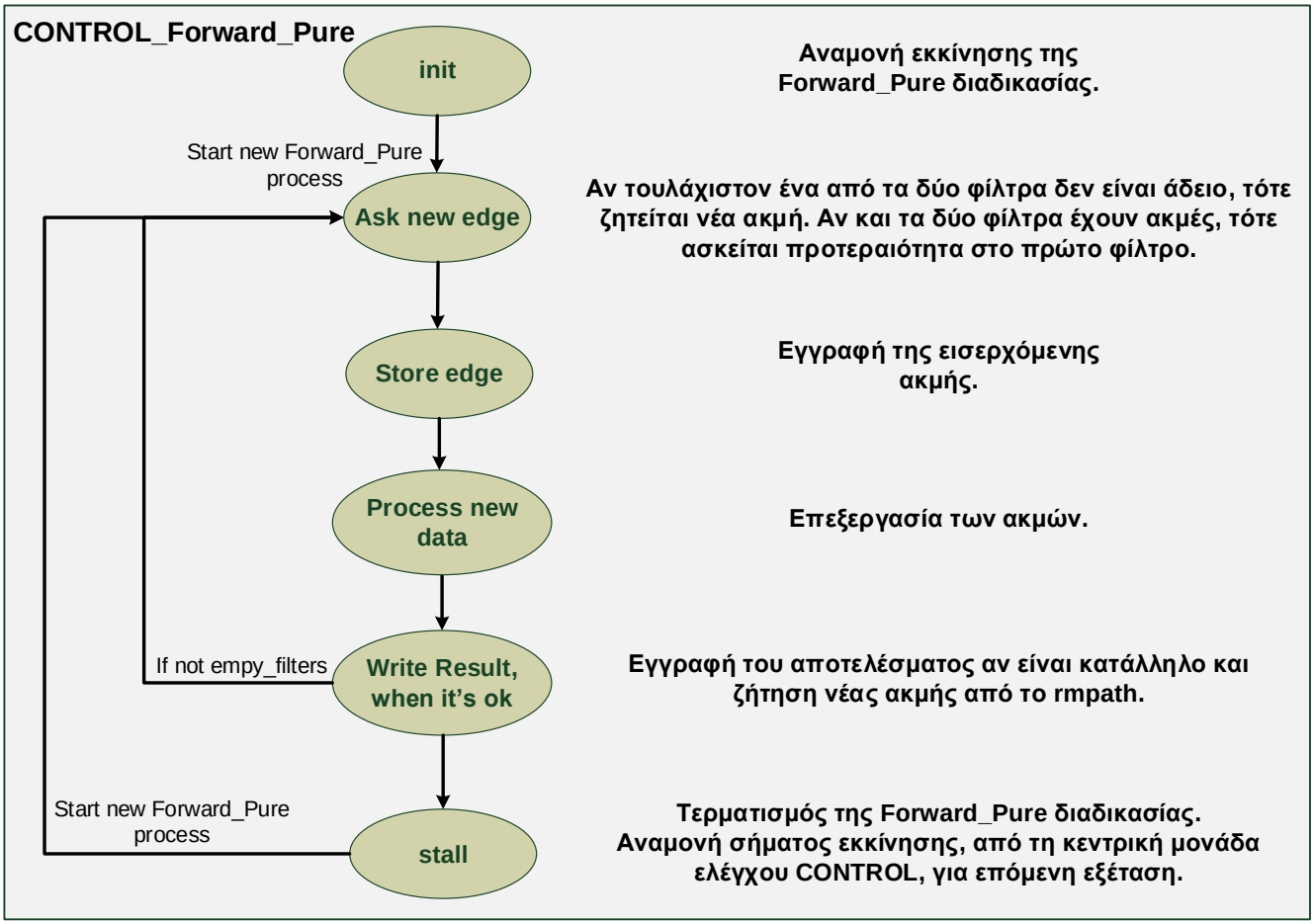
Εικόνα 4.2.3.7.1 : Αρχιτεκτονική του τμήματος που εξετάζει τις υποψήφιες ακμές τύπου forward_pure.

4.2.3.7.2 Τμήμα Control_Forward_Pure

Η μονάδα του Control_Forward_Pure είναι υπεύθυνη για τον συγχρονισμό της όλης διαδικασίας, προκειμένου να διαβαστούν όλες οι ακμές που υπάρχουν μέσα στις δύο ουρές και να ολοκληρωθεί το στάδιο της εξέτασής τους.

Μόλις ολοκληρωθεί και το στάδιο της εξέτασης, τότε η μονάδα ελέγχου εξάγει τα κατάλληλα αποτελέσματα. Το πρώτο είναι αν βρέθηκε ακμή τύπου forward_pure και το δεύτερο ότι ολοκληρώθηκε η διαδικασία εξέτασης των ακμών που υπήρχαν μέσα στα φίλτρα. Τα αποτελέσματα αυτά πηγαίνουν στην κεντρική μονάδα ελέγχου CONTROL καθώς και στο τμήμα της μετονομασίας των κόμβων.

Η μονάδα ελέγχου αποτελείται από ένα σύνολο πεπερασμένων καταστάσεων (Finite State Machines-FSMs), κάθεμία εκ των οποίων υλοποιεί τις λειτουργίες που περιγράφηκαν προηγουμένως. Οι καταστάσεις της μονάδας ελέχου του Control_Forward_Pure περιγράφονται στο σχήμα 4.2.3.7.2.



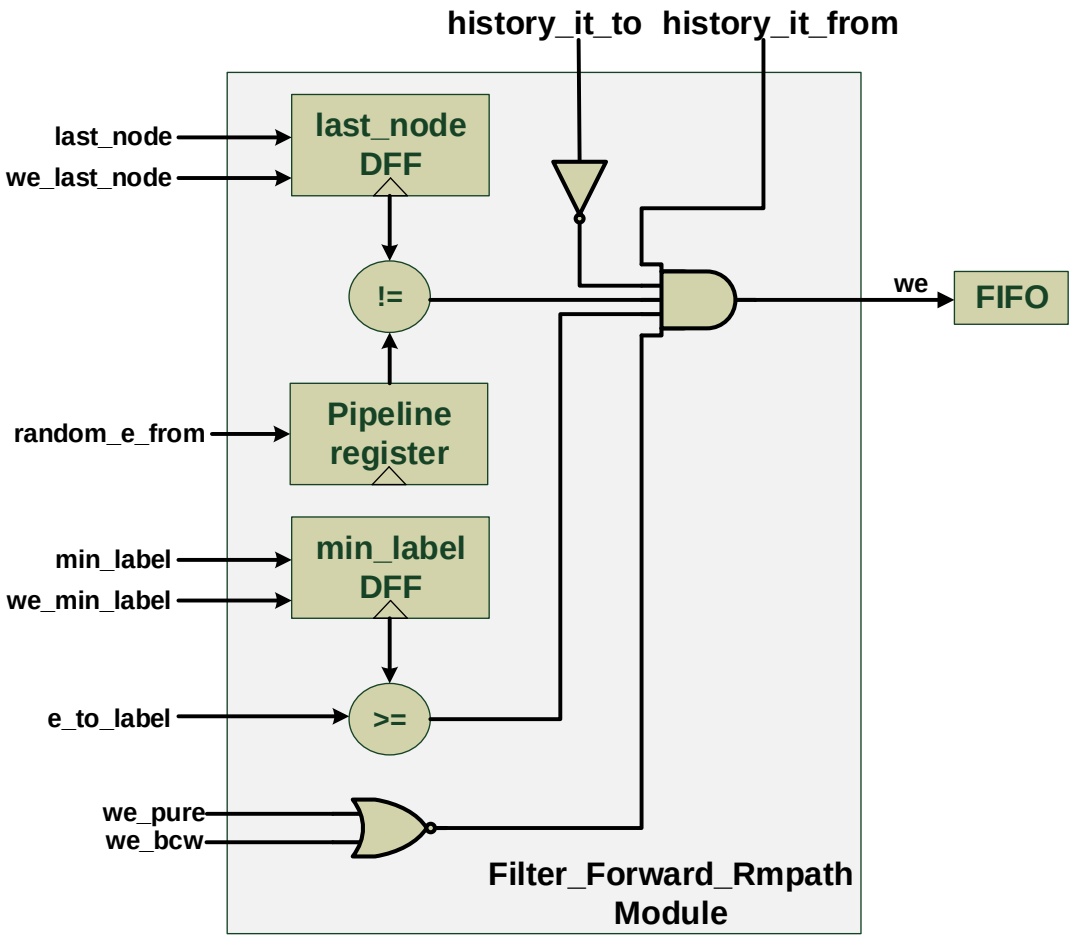
Σχήμα 4.2.3.7.2 : Περιγραφή της FSM του τμήματος ελέγχου για το τμήμα Forward_Pure.

4.2.3.8 Τμήμα Filter_Forward_Rmpath

Σε αυτή την μονάδα τοποθετούνται στην αντίστοιχη ουρά(fifo), που αντιστοιχεί στο τμήμα του Forward_Rmpath, οι ακμές εκείνες που δεν έχουν γραφτεί σε κάποιο από τα προαναφερθέντα φίλτρα και επίσης ικανοποιούν τέσσερις συνθήκες. Οι δύο πρώτες συνθήκες είναι το ιστορικό του κόμβου εκκίνησης της ακμής να υπάρχει στο rightmost path και ο κόμβος να μην είναι ο τελευταίος του rightmost path. Η τρίτη συνθήκη είναι το ιστορικό του κόμβου προορισμού να μην υπάρχει στο rmpath. Η τελευταία συνθήκη είναι ο κόμβος προορισμού της υποψήφιας ακμής να έχει βάρος μεγαλύτερο ή ίσο από το βάρος του πρώτου κόμβου στο DFS_CODE_MIN, το οποίο αναφέρεται ως minlabel και κρατείται σε καταχωρητή, κατά την εκχώρηση της πρώτης ακμής στη DFS_CODE_MIN στοίβα.

Οι παραπάνω συνθήκες λαμβάνονται υπόψιν, όταν ενεργοποιηθεί το αντίστοιχο σήμα από την κεντρική μονάδα ελέγχου.

Η αρχιτεκτονική του Filter_Forward_Rmpath είναι επίσης pipelined, καθώς πρέπει να εξυπηρετούνται εισερχόμενες ακμές ανά κύκλο ρολογιού. Η παραπάνω σχεδίαση απεικονίζεται στο σχήμα 4.2.3.8.



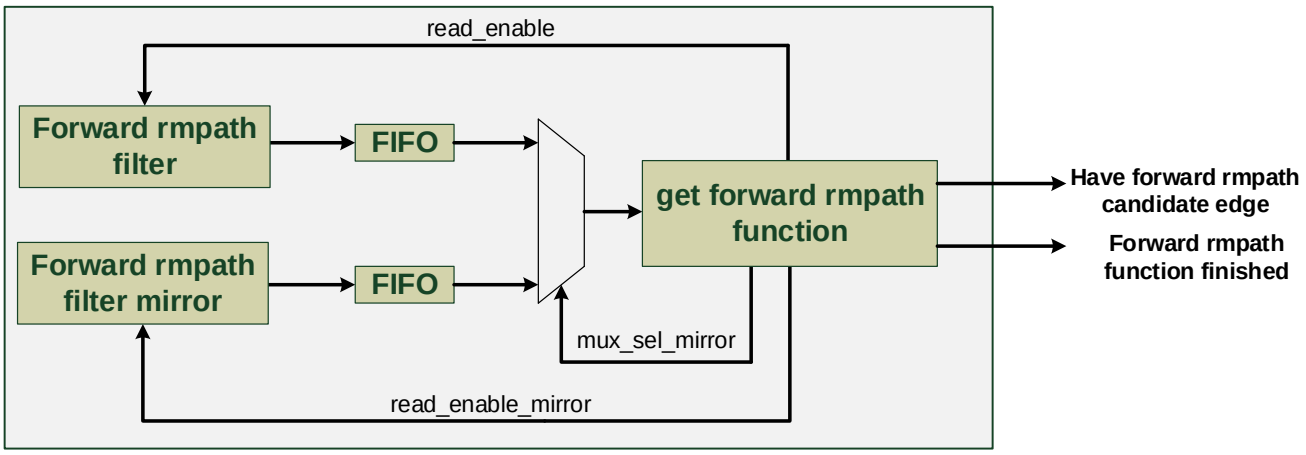
Σχήμα 4.2.3.8 : Αρχιτεκτονική του φίλτρου για τις υποψήφιες ακμές, τύπου Forward_Rmpath.

4.2.3.9 Τμήμα Filter_Forward_Rmpath_mirror

Η λειτουργία του συγκεκριμένου τμήματος έχει να κάνει με την εξέταση των καθρεφτικών ακμών, οι οποίες πληρούν τις συνθήκες που καθορίζει το forward_rmpath είδος προέκτασης. Οι συνθήκες που πρέπει να ικανοποιούνται για να γραφτεί η εισερχόμενη καθρεφτική ακμή στη δεύτερη ουρά(fifo) του τμήματος Forward_Rmpath, είναι οι ίδιες με την ενότητα 4.2.3.8. Το μόνο που αλλάζει είναι το αποτέλεσμα για το ιστορικό και το βάρος του κόμβου εκκίνησης, τα οποία αυτή τη φορά προκύπτουν από την δεύτερη πόρτα καθενός Lookup table.

4.2.3.10 Τμήμα Forward_Rmpath

Το τμήμα Forward_Rmpath είναι υπεύθυνο για την εξέταση όλων των ακμών που έχουν γραφτεί στα δύο φίλτρα του. Η εξέταση των ακμών γίνεται στο τμήμα Fwd_rmpath_examination, το οποίο βρίσκεται εντός του τμήματος get forward rmpath function και στο οποίο εισέρχονται εκτός των άλλων και οι ακμές της στοίβας Forward_rmpath. Ο συγχρονισμός, προκειμένου να εξαχθεί ένα στοιχείο από τη στοίβα του Forward_rmpath, καθώς επίσης και η διαδικασία του διαβάσματος των εισερχόμενων ακμών από τα δύο φίλτρα, γίνεται με τη βοήθεια μίας μονάδας ελέγχου με όνομα Control_Forward_Rmpath.



Σχήμα 4.2.3.10 : Σχηματική αναπαράσταση του τμήματος Forward_Rmpath.

4.2.3.10.1 Στοίβα Forward_rmpath

Η στοίβα Forward_rmpath, περιλαμβάνει όλες τις ακμές και τους κόμβους που βρίσκονται μέσα στο rightmost path του εξεταζόμενου γράφου. Η στοίβα ειναι όμοια με την Backward_rmpath μνήμη (1024x80bits), όμως διαφέρει στον τρόπο με τον οποίο την διαχειρίζεται ο Memory Controller της. Η συγκεκριμένη ιδιατερότητα έχει να κάνει με τον τρόπο που διαχειρίζεται το τμήμα Fwd_rmpath_examination τις ακμές που εξάγονται κάθε φορά από τη μνήμη Forward_rmpath. Πιο συγκεκριμένα κατά τη διαδικασία ανάγνωσης από τη μνήμη, σε κάθε κύκλο εξάγεται μια ακμή του Forward_rmpath, μέχρι να αδειάσει η στοίβα. Ένα παράδειγμα δίνεται στην εικόνα 4.2.3.10.1.

4.2.3.10.2 Τμήμα Fwd_rmpath_examination

Το τμήμα αυτό εξετάζει κάθε φορά μία εισερχόμενη ακμή από τα φίλτρα μαζί με όλες τις ακμές που εξάγονται από την πρώτη πόρτα της στοίβας rmpath, δηλαδή το πεδίο e, όπως αναφέρθηκε προηγουμένως.

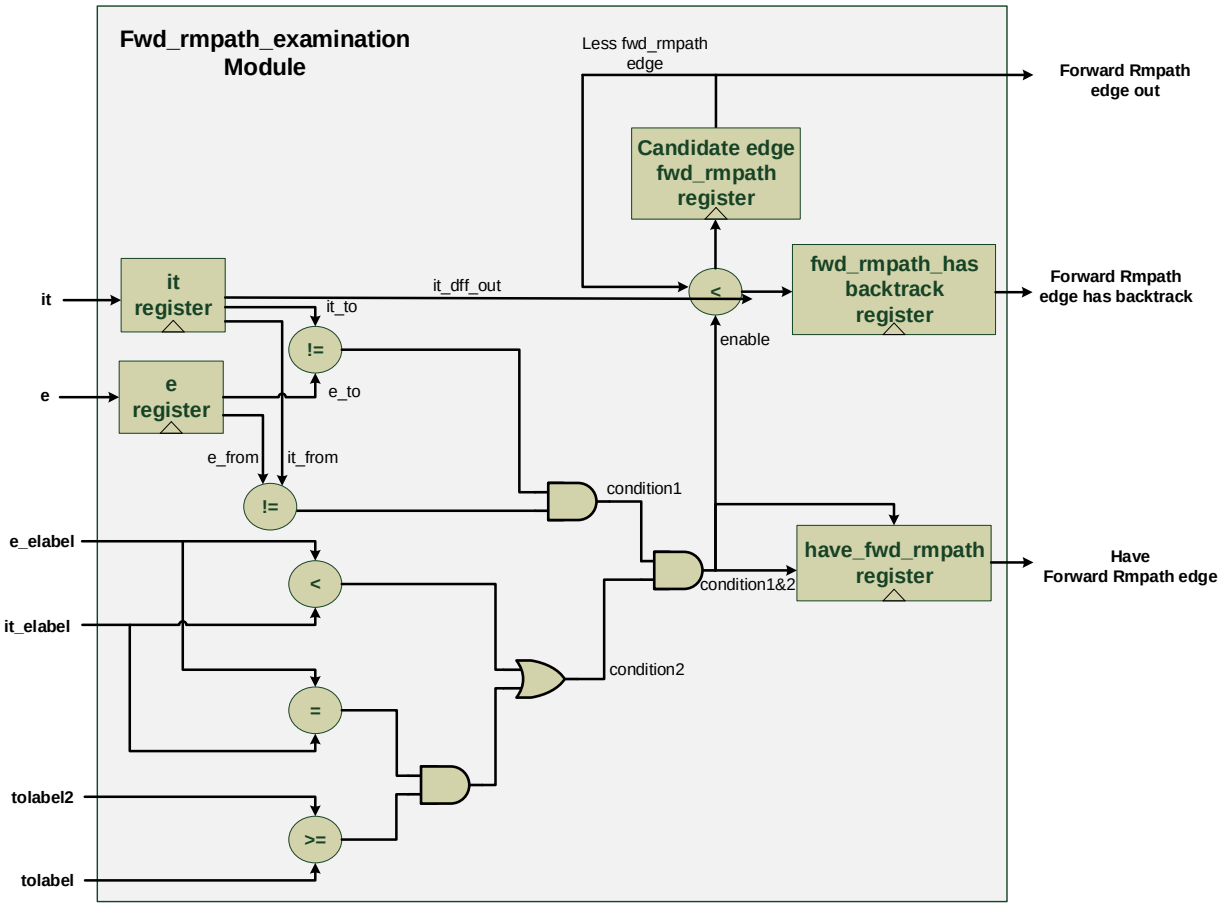
Αρχικά, η κάθε εισερχόμενη ακμή γράφεται σε έναν καταχωρητή, μόνο αν ικανοποιούνται τρείς συνθήκες. Οι δύο συνθήκες, είναι ο κόμβος προορισμού της εξεταζόμενης ακμής, να είναι διαφορετικός από τον κόμβο προορισμού των ακμών που βρίσκονται ήδη μέσα στο rmpath, όμως οι κόμβοι εκκίνησης των δύο ακμών να είναι οι ίδιοι. Η τελευταία συνθήκη αποτελείται από δύο σκέλη. Το ένα σκέλος είναι το βάρος της ακμής της εξεταζόμενης ακμή να είναι μεγαλύτερο από το βάρος της αντίστοιχης ακμής του rmpath. Το δεύτερο είναι αν δεν ικανοποιείται το πρώτο σκέλος, τότε τα δύο βάρη των ακμών να είναι ίσα και το βάρος του κόμβου προορισμού της εξεταζόμενης ακμής να είναι μεγαλύτερο ή ίσο από την ακμή του rmpath. Τα βάρη των κόμβων προκύπτουν από το Lookup table LUT_LABELS.

Αν ικανοποιούνται και οι τρείς συνθήκες, τότε η μονάδα ελέγχου ενεργοποιεί ένα συγκριτή (comparator), ο οποίος με τη σειρά του έχει μία ιδιαίτερη λειτουργία. Η λειτουργία του είναι να κρατήσει εκείνη την ακμή που ξεκινάει από τον χαμηλότερο κόμβο στο rmpath και η οποία είναι και η μικρότερη λεξικογραφικά ακμή από αυτόν τον κόμβο. Η ακμή που προκύπτει από το αποτέλεσμα γράφεται σε έναν καταχωρητή, ο οποίος κάθε φορά κρατάει την εκάστοτε υποψήφια ακμή και την περνάει στη δεύτερη είσοδο του συγκριτή. Σε περίπτωση που υπάρχουν πάνω από μία ίδιες λεξικογραφικά μικρές ακμές, τότε κρατείται αυτή που έχει το μικρότερο όνομα κόμβων(id) και ενεργοποιείται ένα σήμα εξόδου, που δείχνει ότι ο γράφος έφτασε σε δεύτερο μονοπάτι(Forward Rmpath has backtrack).

Η παραπάνω διαδικασία επαναλαμβάνεται, μέχρις ότου αδειάσουν και οι δύο ουρές οι οποίες αντιστοιχούν στο τμήμα Forward_Rmpath. Μόλις η διαδικασία εξέτασης ολοκληρωθεί, το αποτέλεσμα

του καταχωρητή αυτού είναι και η υποψήφια προς επέκταση ακμή.

Η λειτουργία του τμήματος Forward_Rmpath_examination αναπαρίσταται σχηματικά στην εικόνα 4.2.3.10.2.



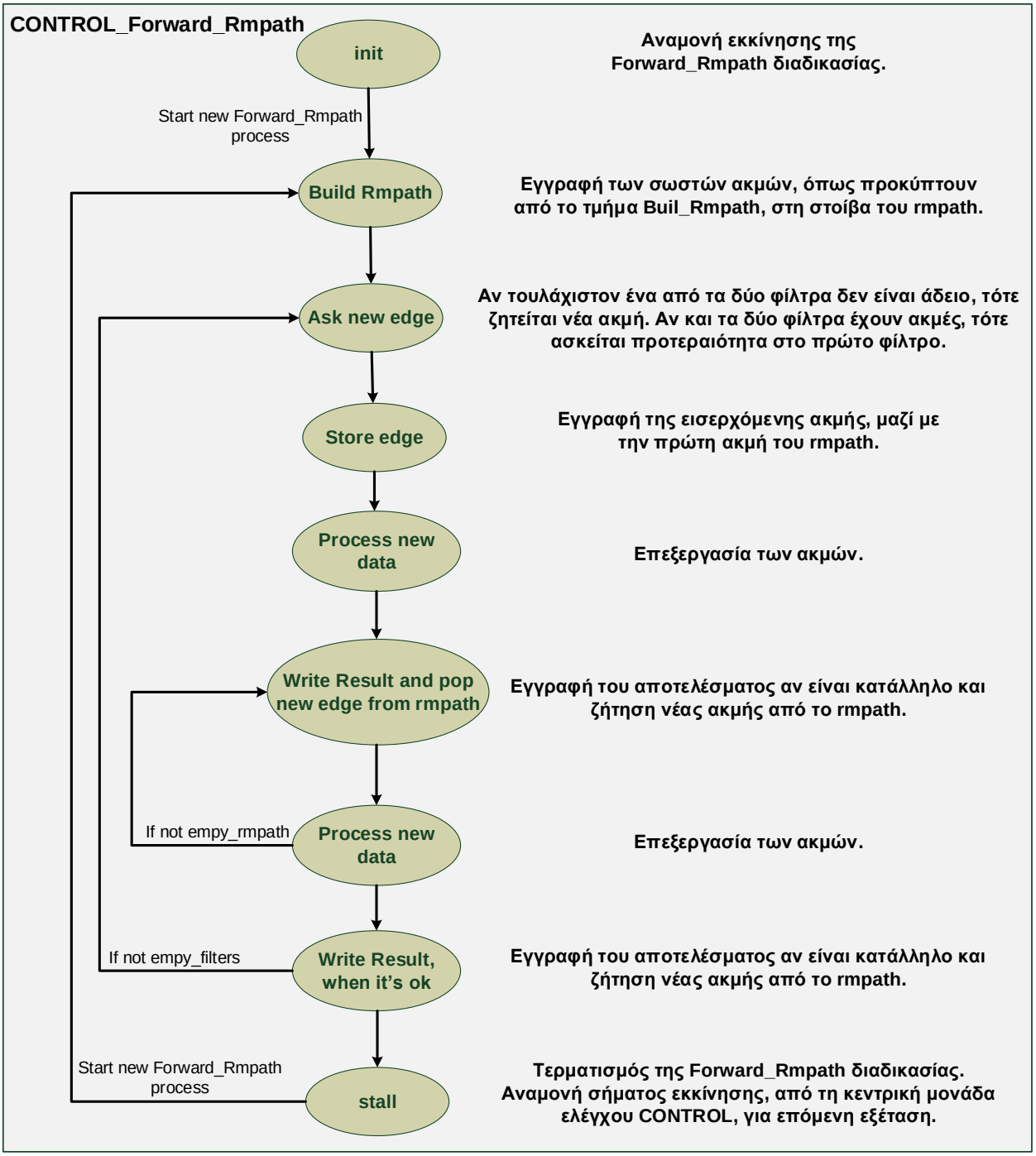
Εικόνα 4.2.3.10.2 : Αρχιτεκτονική του τμήματος που εξετάζει τις υποψήφιες ακμές τύπου forward_rmpath.

4.2.3.10.3 Τμήμα Control_Forward_Rmpath

Η μονάδα ελέγχου είναι υπεύθυνη τόσο για να φέρνει όλες τις ακμές της στοίβας του rmpath για κάθε καινούργια εξεταζόμενη ακμή, όσο και για να εξάγει τα κατάλληλα αποτελέσματα μόλις ολοκληρωθεί η διαδικασία εξέτασης των ακμών.

Τα εξαγόμενα αποτελέσματα από τη μονάδα ελέγχου είναι δύο.Το ένα είναι αν βρέθηκε ακμή τύπου Forward_Rmpath και το δεύτερο ότι ολοκληρώθηκε η διαδικασία εξέτασης των ακμών που υπήρχαν μέσα στα φίλτρα. Τα αποτελέσματα αυτά πηγαίνουν στην κεντρική μονάδα ελέγχου CONTROL καθώς και στο τμήμα της μετονομασίας των κόμβων.

Η μονάδα ελέγχου αποτελείται από ένα σύνολο πεπερασμένων καταστάσεων (Finite State Machines-FSMs), κάθε μία εκ των οποίων υλοποιεί τις λειτουργίες που περιγράφηκαν προηγουμένως. Οι καταστάσεις της μονάδας ελέχου του Control_Forward_Rmpatha περιγράφονται στο σχήμα 4.2.3.10.3.



Σχήμα 4.2.3.10.3 : Περιγραφή της FSM του τμήματος ελέγχου για το τμήμα Forward_Rmpath.

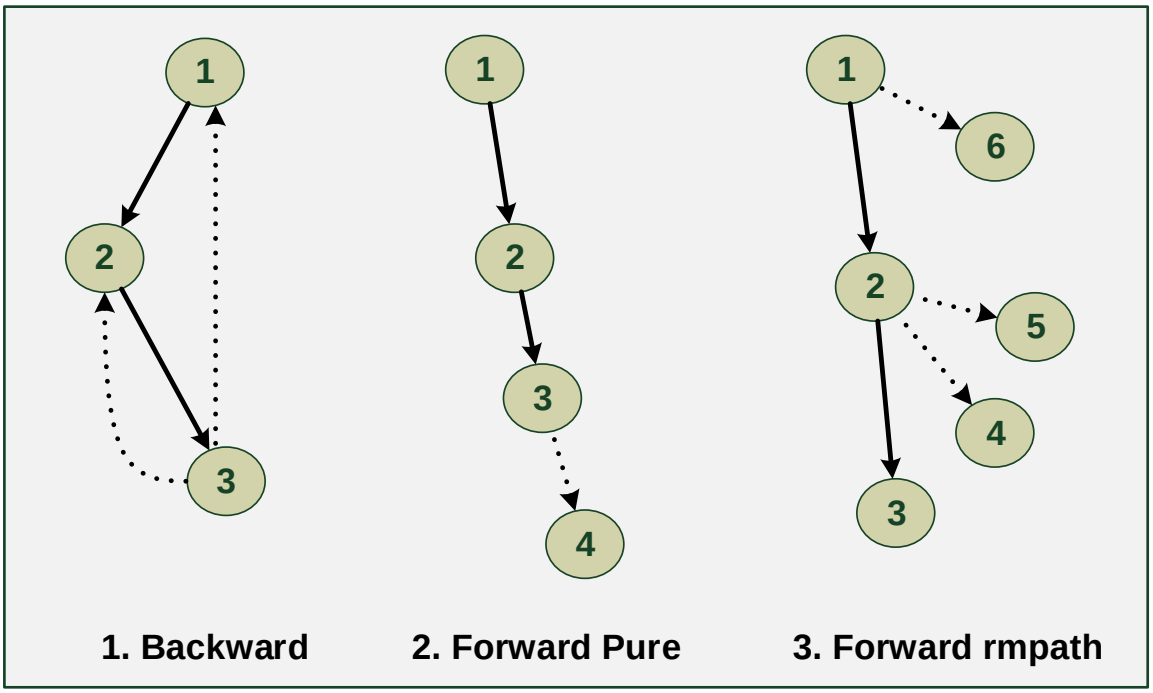
4.2.4 Τμήμα Rename_Vertex

Αυτό το τμήμα είναι υπεύθυνο να παίρνει τα αποτελέσματα του τμήματος Edge_Extension και ανάλογα με τις πιθανές προεκτάσεις και το είδος τους να δίνει την κατάλληλη ονομασία στους κόμβους. Η διαδικασία αυτή της μετονομασίας των κόμβων είναι πολύ σημαντική για την ορθή σύνδεση των ακμών του γράφου και τη σωστή εξέταση όλων των μονοπατιών. Επίσης είναι πολύ σημαντική καθώς

ο γράφος μπορεί να ξεκινήσει από πολλούς διαφορετικούς κόμβους, με διαφορετικά ids κάθε φορά.

Το τμήμα Rename_Vertex αποτελείται από δύο μέρη. Το πρώτο είναι ένας μετρητής(counter), ο οποίος μετράει τους νέους κόμβους που εισάγονται στην στοίβα DFS_CODE_MIN. Σαν αποτέλεσμα ο counter αυτός κρατάει κάθε φορά το όνομα του τελευταίου κόμβου που μπήκε στη στοίβα, το οποίο είναι ίδιο με το πλήθος των διακριτών κόμβων που έχουν ήδη εισαχθεί. Η λειτουργία του τμήματος Rename_Vertex περιγράφεται αναλυτικά παρακάτω.

Ο αλγόριθμος του gSpan προεκτείνει τις ακμές του, σύμφωνα με τα τρία πιθανά είδη προέκτασης, όπως φαίνεται και στο ακόλουθο σχήμα.



Τα τρία αυτά είδη λειτουργούν με βάση σειρά προτεραιότητας. Επομένως, αν υπάρχει υποψήφια ακμή από το Bacward τμήμα, τότε ο κόμβος εκκίνησης παίρνει ως όνομα την τιμή που έχει ο counter, ενώ ο κόμβος προορισμού παίρνει ως όνομα το όνομα του αντίστοιχου κόμβου από το rmpath. Το όνομα αυτό, έρχεται ως πληροφορία από το τμήμα Backward_examination, η οποία είχε κρατηθεί σε έναν καταχωρητή(register) κατά την εξέταση της κατάλληλης ακμής του Backward_rmpath. Εφόσον, δεν εισάγεται κάποια καινούργια ακμή τότε ο counter κρατάει την τελευταία του τιμή ενώ ταυτόχρονα, η ακμή είναι έτοιμη, να περάσει στην έξοδο του τμήματος, μέσα από τον πολυπλέκτη εξόδου και να καταλήξει να γραφτεί στη στοίβα του DFS_CODE_min.

Στην περίπτωση που υπάρχει υποψήφια ακμή από το Forward_Pure τμήμα, τότε ο κόμβος εκκίνησης παίρνει ως όνομα την τιμή που έχει ο counter, ενώ ο κόμβος προορισμού παίρνει ως όνομα την τιμή του counter αυξημένη κατά 1. Αφού, εισάγεται ακμή με έναν καινούργιο κόμβο, τότε ο counter παίρνει την τιμή του νέου κόμβου προορισμού, ενώ ταυτόχρονα, η ακμή είναι έτοιμη, να περάσει στην έξοδο του τμήματος, μέσα από τον πολυπλέκτη εξόδου και να καταλήξει να γραφτεί στη στοίβα του DFS_CODE_min.

Διαφορετικά, στην περίπτωση που υπάρχει υποψήφια ακμή από το Forward_Rmpath τμήμα, τότε ο κόμβος εκκίνησης παίρνει ως όνομα την πληροφορία που έρχεται από το τμήμα Forward_Rmpath_examination, η οποία είχε κρατηθεί σε έναν καταχωρητή(register) κατά την εξέταση της κατάλληλης ακμής του Forward_rmpath. Ενώ ο κόμβος προορισμού παίρνει ως όνομα την τιμή του counter αυξημένη κατά 1. Αφού, εισάγεται ακμή με έναν καινούργιο κόμβο, τότε ο counter παίρνει

την τιμή του νέου κόμβου προορισμού, ενώ ταυτόχρονα, η ακμή είναι έτοιμη, να περάσει στην έξοδο του τμήματος, μέσα από τον πολυπλέκτη εξόδου και να καταλήξει να γραφτεί στη στοίβα του DFS_CODE_min.

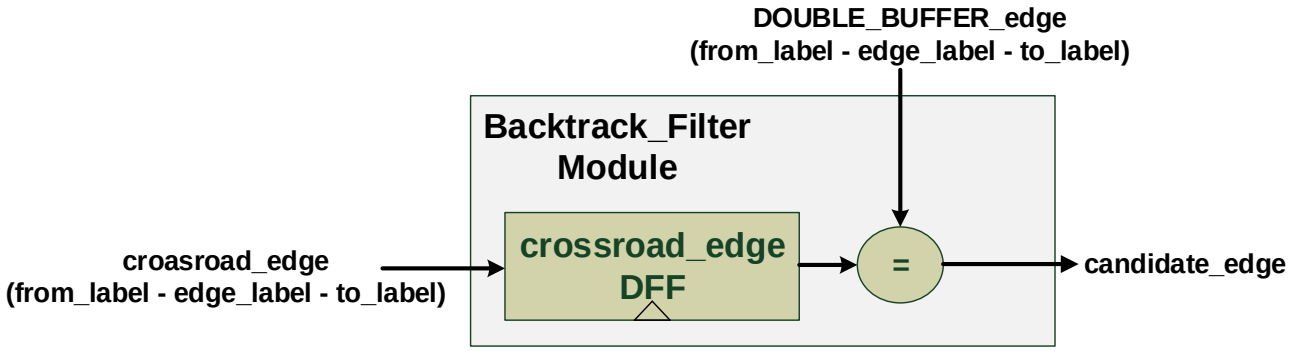
Αν δεν υπάρξει κάποια προέκταση, αυτό σημαίνει είτε ότι το μονοπάτι που εξετάστηκε οδηγήθηκε σε αδιέξοδο, ή τερματίστηκε. Σε αυτή την περίπτωση, θα ενεργοποιηθεί η λειτουργία του Backtrack από την κεντική μονάδα ελέγχου CONTROL και θα ξεκινήσει η εξαγωγή ακμών μέχρι να μπορεί να ακολουθηθεί νέα διαδρομή. Μόλις διαπιστωθεί ότι μία ακμή, μπορεί να ακολουθήσει νέα διαδρομή, τότε θα γραφτεί στον counter, το όνομα του κόμβου προορισμού αυτής της ακμής και η διαδικασία θα είναι έτοιμη να συνεχιστεί κανονικά.

4.2.5 Τμήμα Backtrack_Filter

Ο αλγόριθμος gSpan είναι αναδρομικός και έχει σαν αποτέλεσμα σε κάθε αναδρομή του να εξετάζει και από μία προέκταση. Στο hardware, επιλέχθηκε η τεχνική του backtracking με σκοπό να καλυφθεί η εξέταση κάθε πιθανού μονοπατιού ενός γράφου. Η μέθοδος αυτή είναι στην ουσία μία Brute-force εξέταση των συνδέσεων των κόμβων ενός γράφου και στο κύκλωμά μας φαίνεται από το τμήμα Backtrack_Filter.

Πιο συγκεκριμένα, μόλις η κεντρική μονάδα ελέγχου CONTROL αντιληφθεί ότι δεν υπάρχει άλλη πιθανή προέκταση ενός μονοπατιού, τότε ενεργοποιεί τη λειτουργία BACKTRACK. Αμέσως οι ακμές που έχουν μπει στη στοίβα, ξεκινούν να βγαίνουν και να τοποθετούνται μέσα στη δομή του DOUBLE_BUFFER, μέχρις ότου ανιχνευθεί ότι μία ακμή μπορεί να ακολουθήσει και εναλλακτική διαδρομή. Τότε σταματάει η παραπάνω διαδικασία, και ξεκινάει η εξαγωγή των ακμών που υπάρχουν μέσα στον DOUBLE_BUFFER στο Backtrack_Filter. Σε αυτό το τμήμα, φιλτράρονται όλες οι πιθανές προεκτάσεις και περνούν, μόνο εκείνες οι οποίες είναι λεξικογραφικά ισοδύναμες με την ακμή που είχε μπει πρώτη στο προηγούμενο μονοπάτι και έχει και ίδιο είδος προέκτασης. Αυτό φαίνεται σχηματικά στο σχήμα 4.2.5.

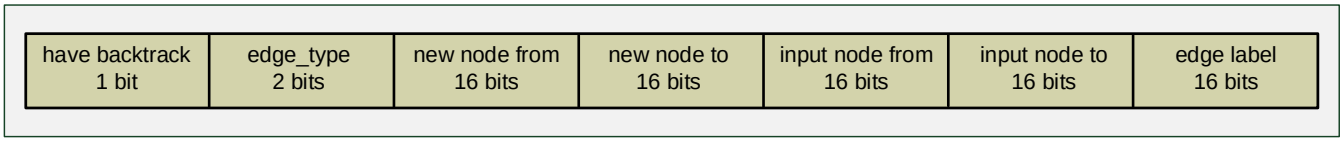
Το τμήμα του Backtrack_Filter δέχεται ως είσοδο την ακμή previous_edge, η οποία έχει γραφτεί σε έναν καταχωρητή, μόλις ανιχνεύθηκε ότι έχει αντικαταστάτρια ακμή καθώς και όλες τις ακμές που εξάγονται από τον DOUBLE_BUFFER. Πραγματοποιώντας τις παραπάνω συγκρίσεις που περιγράψαμε, τροφοδοτεί το τμήμα Edge_Extension με τις κατάλληλες ακμές. Με αυτό τον τρόπο, ξεκινάει η εξέταση ενός καινούργιου μονοπατιού.



Σχήμα 4.2.5 : Αρχιτεκτονική του τμήματος Backtrack_Filter.

4.2.6 Στοίβα DFS_CODE_MIN

Η στοίβα αυτή είναι μία single port ram με διαστάσεις (1024x82), στην οποία κρατούνται τα στιγμιότυπα του γράφου, μέχρι να ολοκληρωθεί η διαδικασία της επέκτασής του. Τα 82 bits, που αντιστοιχούν σε μία ακμή έχουν ένα συγκεκριμένο τύπο ακμής στην είσοδο, το οποίο προέκυψε με βάση τη λειτουργία του κυκλώματος και περιγράφεται στο σχήμα 4.2.6.1.



Σχήμα 4.2.6.1 : Format της πληροφορίας που κρατείται για μία ακμή του νέου γράφου.

Το πεδίο **have backtrack**, κρατάει πληροφορία για το αν η ακμή που εισέρχεται στη στοίβα έχει λεξικογραφικά ισοδύναμη ακμή, η οποία μπορεί να οδηγήσει σε διαφορετικό μονοπάτι.

Το πεδίο **edge type**, κρατάει πληροφορία για το είδος που είναι η συγκεκριμένη ακμή. Πιο συγκεκριμένα, υπάρχουν 4 είδη ακμών. Αυτές που έχουν προκύψει από το τμήμα get_forward_root και έχουν τον κωδικό 00. Αυτές που έχουν προκύψει από το τμήμα get_backward και έχουν τον κωδικό 01. Αυτές που έχουν προκύψει από το τμήμα get_forward_pure και έχουν τον κωδικό 10. Και τέλος, αυτές που έχουν προκύψει από το τμήμα get_forward_rmpath και έχουν τον κωδικό 11.

Το πεδίο **new node from**, αναφέρει το όνομα του κόμβου εκκίνησης, μετά τη μετονομασία που έχει γίνει στο τμήμα Rename_Vertex.

Το πεδίο **new node το**, αναφέρει το όνομα του κόμβου προορισμού, μετά τη μετονομασία που έχει γίνει στο τμήμα Rename_Vertex.

Το πεδίο **input node from**, αναφέρει το όνομα του κόμβου εκκίνησης, όπως είχε δωθεί κατά την είσοδο και έχει κρατηθεί στην αντίστοιχη θέση της μνήμης DFS_CODE.

Το πεδίο **input node to**, αναφέρει το όνομα του κόμβου προορισμού, όπως είχε δωθεί κατά την είσοδο και έχει κρατηθεί στην αντίστοιχη θέση της μνήμης DFS_CODE.

Το πεδίο **edge label**, αναφέρει το βάρος της συγκεκριμένης ακμής.

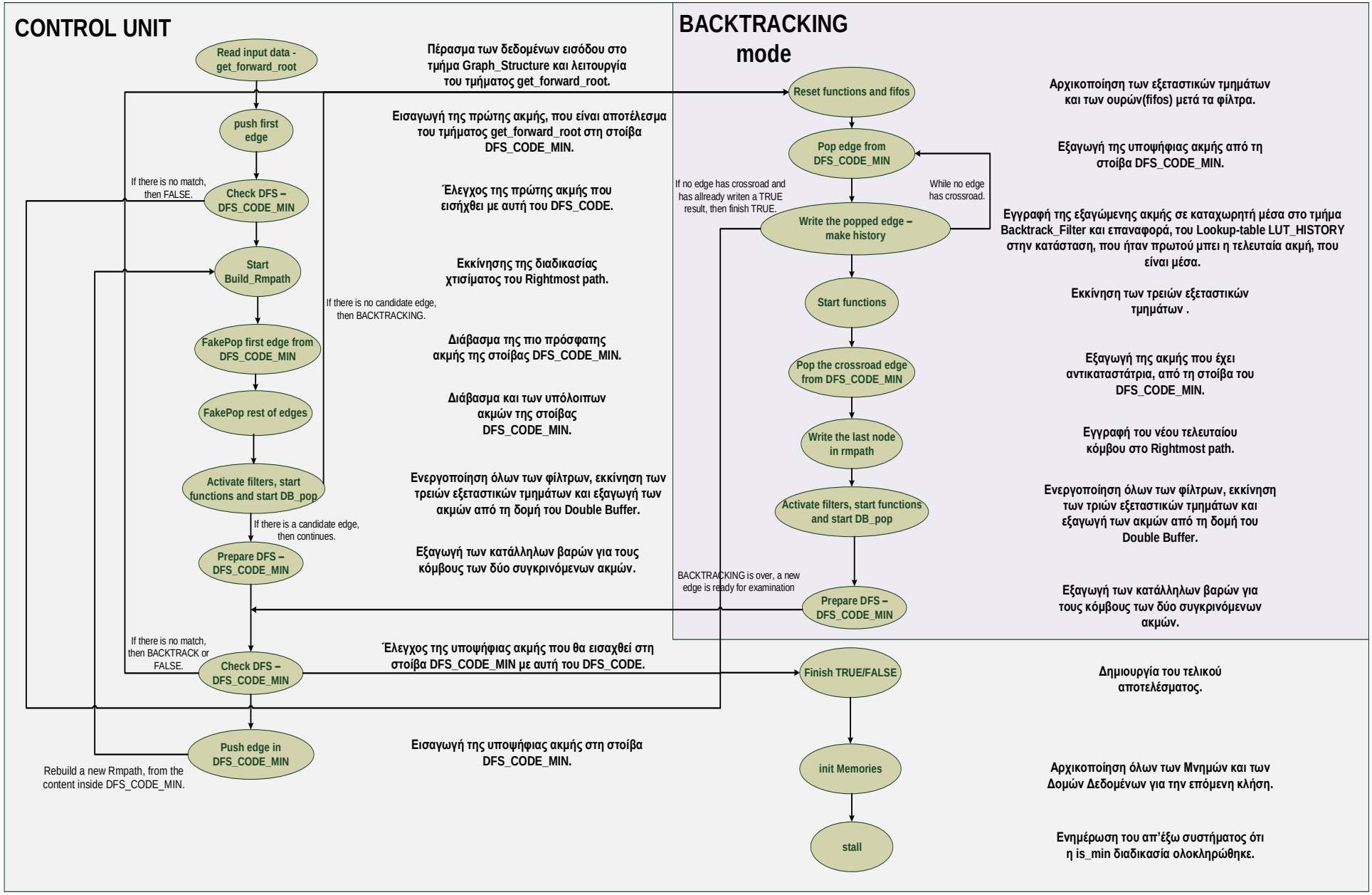
4.2.7 Μονάδα ελέγχου CONTROL

Αυτή η μονάδα είναι υπεύθυνη για το συγχρονισμό και την επικοινωνία κάθε τμήματος της προτεινόμενης αρχιτεκτονικής στο κύκλωμά μας.

Η κεντρική μονάδα ελέγχου, κάθε φορά που υπάρχει ακμή για να μπεί στη στοίβα DFS_CODE_MIN, εξετάζει αν είναι ίδια με την αντίστοιχη ακμή του γράφου, που είχε εισαχθεί στην αρχή. Αν είναι ίδια τόσο λεξικογραφικά όσο και με τα ονόματα των κόμβων, τότε την εισάγει στη στοίβα DFS_CODE_MIN(με τις κατάλληλες ονομασίες των κόμβων από το τμήμα Rename_Vertex) και συνεχίζει την προέκταση του μονοπατιού. Αλλιώς αν η ακμή που εισήχθει είτε είναι είδος υψηλότερης προτεραιότητας από της ακμής που έχει δωθεί στην είσοδο, ή είναι λεξικογραφικά μικρότερη, τότε η σχεδίαση τερματίζει με αποτέλεσμα ότι ο γράφος είναι Not Minimal.

Στην περίπτωση, που δεν υπάρχει προέκταση, αυτό μπορεί να οφείλεται σε δύο πιθανά σενάρια. Το πρώτο είναι να φτάσει σε αδιέξοδο το εξεταζόμενο μονοπάτι όπως προαναφέρθηκε στο υποκεφάλαιο 4-2-5, οπότε ενεργοποιείται η λειτουργία του BACKTRACK και εξετάζεται κάποιο εναλλακτικό σταυροδρόμι. Ενώ το δεύτερο είναι να έχουν εξεταστεί όλα τα πιθανά μονοπάτια του γράφου, και αφού δεν έχει βρεθεί μικρότερη λεξικογραφικά ακμή ή διαφορετική προέκταση, η σχεδίαση τερματίζει με αποτέλεσμα ότι ο γράφος είναι Minimal.

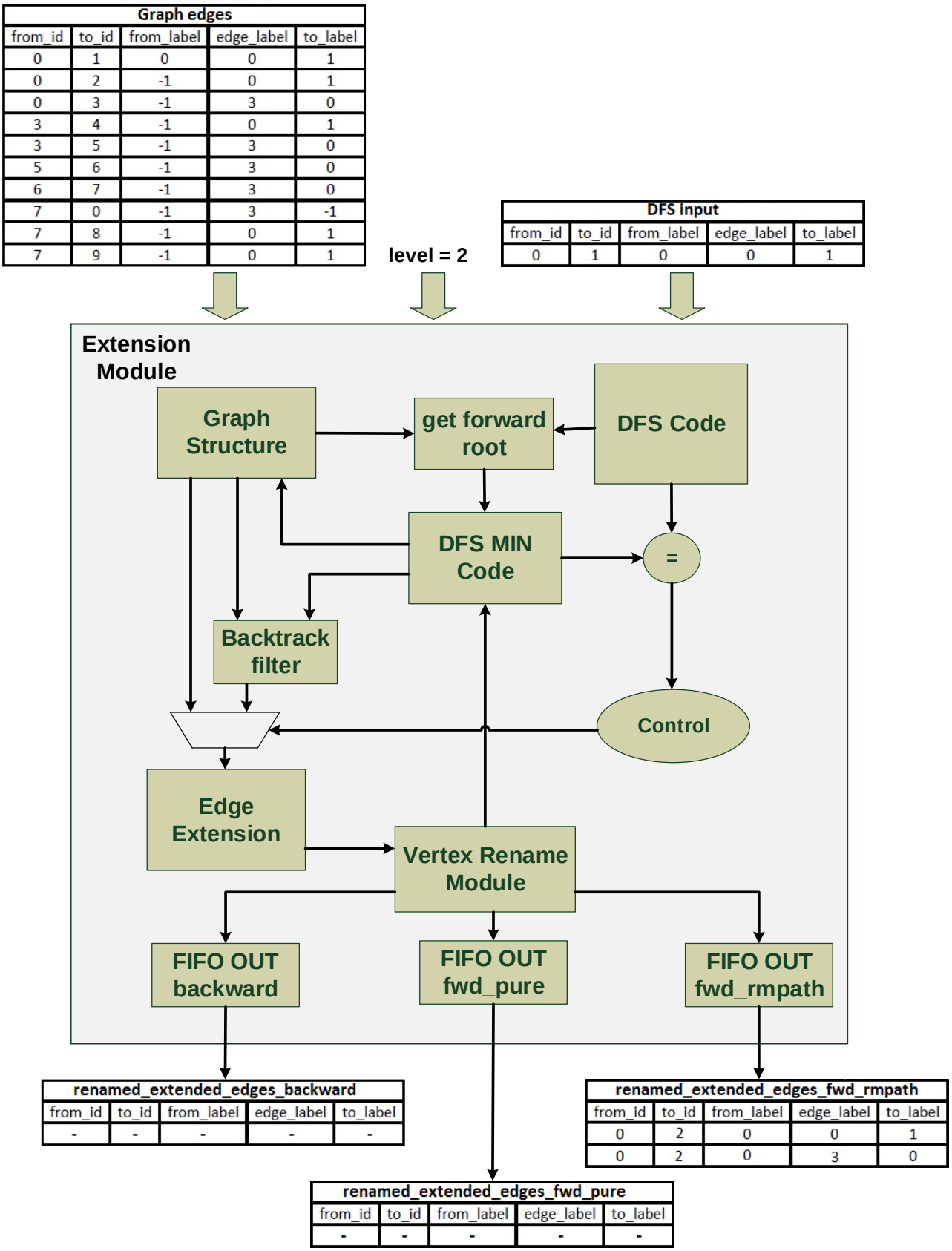
Οι βασικές λειτουργίες της μονάδας ελέγχου και οι κύριες καταστάσεις της FSM που υλοποιήθηκε, περιγράφονται σχηματικά στην επόμενη εικόνα.


CONTROL UNIT
BACKTRACKING mode
Read input data - get_forward_root
Πέρασμα των δεδομένων εισόδου στο τμήμα Graph_Structure και λειτουργία του τμήματος get_forward_root.
push first edge
Εισαγωγή της πρώτης ακμής, που είναι αποτέλεσμα του τμήματος get_forward_root στη στοίβα DFS_CODE_MIN.
If there is no match, then FALSE.
Check DFS – DFS_CODE_MIN
Έλεγχος της πρώτης ακμής που εισήχθει με αυτή του DFS_CODE.
Start Build_Rmpath
Εκκίνησης της διαδικασίας χτισίματος του Rightmost path.
If there is no candidate edge, then BACKTRACKING.
FakePop first edge from DFS_CODE_MIN
Διάβασμα της πιο πρόσφατης ακμής της στοίβας DFS_CODE_MIN.
FakePop rest of edges
Διάβασμα και των υπόλοιπων ακμών της στοίβας DFS_CODE_MIN.
Activate filters, start functions and start DB_pop
Ενεργοποίηση όλων των φίλτρων, εκκίνηση των τρειών εξεταστικών τμημάτων και εξαγωγή των ακμών από τη δομή του Double Buffer.
If there is a candidate edge, then continues.
Prepare DFS – DFS_CODE_MIN
Εξαγωγή των κατάλληλων βαρών για τους κόμβους των δύο συγκρινόμενων ακμών.
If there is no match, then BACKTRACK or FALSE.
Check DFS – DFS_CODE_MIN
Έλεγχος της υποψήφιας ακμής που θα εισαχθεί στη στοίβα DFS_CODE_MIN με αυτή του DFS_CODE.
Push edge in DFS_CODE_MIN
Εισαγωγή της υποψήφιας ακμής στη στοίβα DFS_CODE_MIN.
Rebuild a new Rmpath, from the content inside DFS_CODE_MIN.
Reset functions and fifos
Αρχικοποίηση των εξεταστικών τμημάτων και των ουρών(fifos) μετά τα φίλτρα.
Pop edge from DFS_CODE_MIN
Εξαγωγή της υποψήφιας ακμής από τη στοίβα DFS_CODE_MIN.
If no edge has crossroad and has allready writen a TRUE result, then finish TRUE.
While no edge has crossroad.
Write the popped edge – make history
Εγγραφή της εξαγώμενης ακμής σε καταχωρητή μέσα στο τμήμα Backtrack_Filter και επαναφορά, του Lookup-table LUT_HISTORY στην κατάσταση, που ήταν πρωτού μπει η τελευταία ακμή, που είναι μέσα.
Start functions
Εκκίνηση των τρειών εξεταστικών τμημάτων .
Pop the crossroad edge from DFS_CODE_MIN
Εξαγωγή της ακμής που έχει αντικαταστάτρια, από τη στοίβα του DFS_CODE_MIN.
Write the last node in rmpath
Εγγραφή του νέου τελευταίου κόμβου στο Rightmost path.
Activate filters, start functions and start DB_pop
Ενεργοποίηση όλων των φίλτρων, εκκίνηση των τριών εξεταστικών τμημάτων και εξαγωγή των ακμών από τη δομή του Double Buffer.
BACKTRACKING is over, a new edge is ready for examination
Prepare DFS – DFS_CODE_MIN
Εξαγωγή των κατάλληλων βαρών για τους κόμβους των δύο συγκρινόμενων ακμών.
Finish TRUE/FALSE
Δημιουργία του τελικού αποτελέσματος.
init Memories
Αρχικοποίηση όλων των Μνημών και των Δομών Δεδομένων για την επόμενη κλήση.
stall
Ενημέρωση του απ'έξω συστήματος ότι η is_min διαδικασία ολοκληρώθηκε.

4.3 Αρχιτεκτονική τμήματος Extension

Η σχεδίαση του τμήματος Extension, έγινε χρησιμοποιώντας κομμάτια τα οποία αναλύθηκαν προηγουμένως, κατά την περιγραφή του τμήματος is_min. Επίσης, εφαρμόσαμε μετατροπές στον τρόπο με τον οποίο το νέο τμήμα δέχεται τις εισόδους και στην διαδικασία επεξεργασίας των υποψήφιων επεκτάσεων. Αυτό το τμήμα είναι υπεύθυνο για την εξόρυξη των υποψήφιων ακμών, οι οποίες αποτελούν τις επεκτάσεις ενός υπογράφου μέσα σε έναν γράφο. Έτσι το τμήμα Extension δέχεται ως είσοδο τις ακμές, ενός από τους γράφους εισόδους (Graph), τον υπογράφο που επεκτείνει (DFS_CODE) και το επίπεδο στο οποίο αναζητούνται οι επεκτάσεις και επιστρέφει κάθε πιθανή επέκταση από τα τρία είδη επέκτασης. Οι επεκτάσεις ανάλογα με το είδος επέκτασης, γράφονται σε μία ουρά (FIFO), την οποία διαβάζει το top-level τμήμα, που ονομάζεται Project. Η top level αρχιτεκτονική του τμήματος Extension, απεικονίζεται στην εικόνα 4.3.

Υπάρχει μία βασική διαφορά μεταξύ της υλοποίησης στην αυθεντική έκδοση του gSpan στην C++ και στην υλοποίηση στο hardware, την οποία σχεδιάσαμε. Η διαφορά αυτή αφορά την εξέταση όλων των ίδιων ακμών, προκειμένου να εξεταστούν όλα τα μονοπάτια που ξεκινούν από την ίδια ακμή, μέσα σε όλους τους γράφους. Στο software, αυτό γίνεται περνόντας ως όρισμα ένα vector, το οποίο περιλαμβάνει όλες τις ίδιες ακμές και τα ids των γράφων στους οποίους εμπεριέχονται. Έτσι, εξετάζει σειριακά τις ακμές του vector και ανιχνεύει προεκτάσεις για κάθε μία ακμή. Σε αντίθετη περίπτωση, στη σχεδίαση στο hardware, επιλέξαμε ένα πιο βολικό τρόπο εξέτασης των ακμών. Πιο συγκεκριμένα, επιλέξαμε να περνάμε μόνο τα βάρη των κόμβων και της ακμής(DFS_CODE edge) αντί να περνάμε σαν είσοδο μία-μία, κάθε ακμή του vector. Στη συνέχεια, κάθε τμήμα Extension, αναλαμβάνει να δημιουργήσει όλα τα μονοπάτια που ξεκινούν από την ακμή του DFS_CODE και προεκτείνουν τον γράφο, επιλέγοντας τις κατάλληλες ακμές από το τμήμα Graph Structure, οι οποίες ταιριάζουν με τις ακμές στο DFS_CODE. Κάθε φορά που ο γράφος φτάνει στο επίπεδο που έχει ζητηθεί, αλλάζει η λειτουργία της σχεδίασης και ενεργοποιούνται οι ουρές εξόδου (FIFOs out) οι οποίες γράφουν όλες τις επεκτάσεις. Για την προέκταση κάθε δυνατού μονοπατιού, χρησιμοποιήθηκε η τεχνική του backtracking, όπως και στο τμήμα is_min.

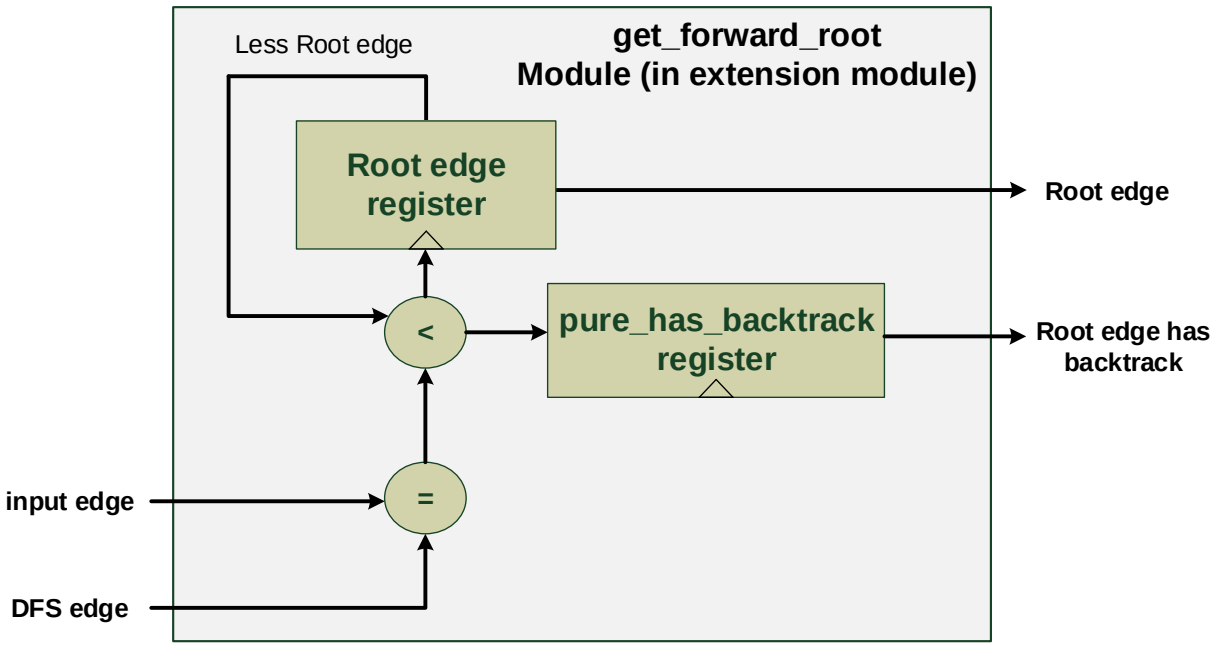


Εικόνα 4.3 : Αρχιτεκτονική τμήματος Extension.

4.3.1 Τμήμα get_forward_root

Προκειμένου να βρεθούν όλες οι πιθανές επεκτάσεις του γράφου που δίνεται στην είσοδο DFS Code για ένα συγκεκριμένο επίπεδο level, η σχεδίαση ξεκινάει την εξέταση κάθε πιθανού συνδυασμού απ'όλες τις ακμές, που μπορούν να συνθέσουν αυτόν τον γράφο. Με αυτόν τον τρόπο, το τμήμα get_forward_root είναι υπεύθυνο για την αναζήτηση της πρώτης ακμής που θα μπεί στη στοίβα του DFS_CODE_MIN, και η οποία θα ταιριάζει και με την πρώτη ακμή που είναι στο DFS Code.

Το τμήμα αυτό είναι ενεργό ταυτόχρονα με το γέμισμα του τμήματος Graph_Structure. Επομένως, όσο έρχονται όλες οι ακμές του γράφου στο τμήμα Graph_Structure, ταυτόχρονα φιλτράρονται στο τμήμα get_forward_root και στη συνέχεια τοποθετείται η κατάλληλη ακμή στη στοίβα DFS_CODE_MIN. Το φίλτρο που χρησιμοποιείται σε αυτό το τμήμα, έχει ένα ακόμη χαρακτηριστικό. Σε περίπτωση που υπάρχουν παραπάνω από μία κατάλληλες ακμές, τότε γράφεται μαζί με την ακμή ένα bit επιπλέον το οποίο προσδιορίζει ότι υπάρχει διαφορετικό μονοπάτι για την συγκεκριμένη ακμή. Η πληροφορία αυτή θα χρειαστεί, μόλις ενεργοποιηθεί η διαδικασία Backtrack στο κύκλωμα ώστε να ελεγχθούν όλα τα δυνατά μονοπάτια. Η παραπάνω διαδικασία φαίνεται σχηματικά στο σχήμα 4.3.1.



Σχήμα 4.3.1 : Αρχιτεκτονική τμήματος get_forward_rmpath (στο τμήμα Extension).

4.3.2 Τμήμα Graph_Structure

Το τμήμα Graph_Structure είναι υπεύθυνο για τη συγκέντρωση όλης της χρήσιμης πληροφορίας του γράφου εισόδου, την οποία χρειάζεται η υπόλοιπη σχεδίαση για την σωστή λειτουργία της. Το τμήμα αυτό, αποτελείται από τρείς βασικές δομές δεδομένων: τον DOUBLE BUFFER, το LUT_LABELS και το LUT_HISTORY, όπως φαίνεται στην Εικόνα 4.2.2.1.

Ο DOUBLE BUFFER του τμήματος extension, σε αντίθεση με την παράγραφο 4.2.2, περιλαμβάνει όλες τις ακμές ενός γράφου, οι οποίες έχουν δωθεί στο data set εισόδου. Η λειτουργικότητα παραμένει ίδια, με δύο στοίβες ίδιας διάστασης, ένα φίλτρο που εξετάζει τις ακμές που γράφονται σε κάθε μία στοίβα εναλλάξ και μία μονάδα ελέγχου, η οποία συντονίζει όλη την διαδικασία. Η δομή του DOUBLE BUFFER είναι υπεύθυνη για να εξάγει τις υποψήφιες ακμές που δεν έχουν μπει ακόμη στη στοίβα του DFS_CODE_MIN. Η διαδικασία αυτή της εξαγωγής ενεργοποιείται από τη μονάδα του Control DB και γίνεται κάθε φορά εξάγοντας τις ακμές της γεμάτης στοίβας και περνώντας τες μέσα από το φίλτρο. Το φίλτρο αυτό είναι υπεύθυνο να κόψει την ακμή που μπήκε πιο πρόσφατα στη στοίβα DFS_CODE_MIN και να περάσει τις υπόλοιπες ακμές, ώστε να γραφτούν στην άλλη στοίβα και να συνεχιστεί η διαδικασία επέκτασης. Με αυτό τον τρόπο, μειώνεται κάθε φορά το πλήθος των υποψήφιων ακμών για να μπούν στη στοίβα DFS_CODE_MIN.

Οι υπόλοιπες δομές δεδομένων είναι δύο Lookup tables, τα οποία έχουν την ίδια χρησιμότητα με το τμήμα is_min. Η πρώτη δομή καλείται LUT_LABELS και κρατάει πληροφορία για τα βάρη όλων των κόμβων του γράφου εισόδου. Η δεύτερη δομή καλείται LUT_HISTORY και κρατάει το ιστορικό όλων των κόμβων κατά τη διαδικασία της προέκτασης. Και οι δύο αυτές δομές, προσπελαύνονται από το τμήμα Edge_Extension και παίζουν καθοριστικό ρόλο στη διαδικασία επέκτασης κάθε μονοπατιού.

4.3.3 Τμήμα Edge_Extension

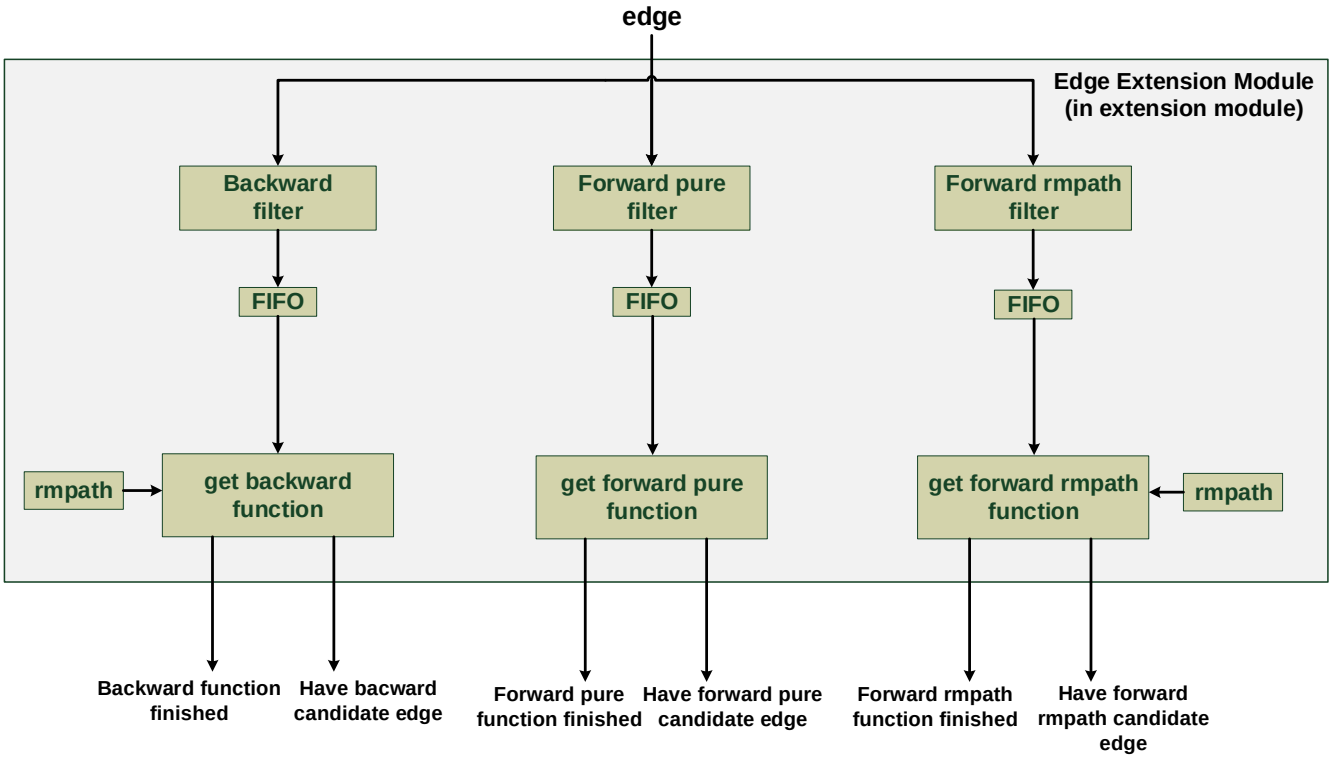
Αυτό το τμήμα είναι υπεύθυνο να εξετάζει όλες τις ακμές που του έρχονται είτε από τη δομή δεδομένων του DOUBLE BUFFER μέσα στο τμήμα Graph_Structure, είτε από το Backtrack_Filter και να εξάγει την επόμενη προεκτεινόμενη ακμή.

Όσο το πλήθος των εισαγόμενων ακμών μέσα στη στοίβα DFS_CODE_MIN είναι μικρότερο από τον αριθμό level-1, τότε το τμήμα Edge_Extension αναζητεί εκείνες τις υποψήφιες επεκτάσεις που συμφωνούν με τις ακμές της δομής DFS Code. Σε αντίθετη περίπτωση, ενεργοποιείται στο κύκλωμα η λειτουργία της εξέτασης των ακμών του επιπέδου level και κάθε υποψήφια ακμή αυτού του επιπέδου, τοποθετείται στην αντίστοιχη ουρά, ανάλογα με το είδος της ακμής.

Η διαδικασία της προέκτασης του γράφου, γίνεται με βάση τα τρία κριτήρια επέκτασης του αλγορίθμου gSpan, όπως αυτά περιγράψαμε σε προηγούμενες ενότητες.

Η βασική διαφορά του συγκεκριμένου τμήματος για τη λειτουργία στην αρχιτεκτονική του is_min και στην αρχιτεκτονική του extension, είναι ότι στη δεύτερη αρχιτεκτονική, αγνοούνται οι καθρεφτικές εξετάσεις των ακμών. Έτσι, απαιτείται η χρησιμοποίηση τριών φίλτρων, ενός για κάθε διαφορετικό είδος προέκτασης και τριών τμημάτων, τα οποία θα εξετάζουν ταυτόχρονα την ίδια ακμή για τις δύο κατευθύνσεις της. Τα φίλτρα χρησιμοποίησαν μέρος της λογικής των τριών συναρτήσεων, προκειμένου να γράψουν τις υποψήφιες ακμές στις ουρές (FIFOs) που ακολουθούν. Με αυτό τον τρόπο, καταφέραμε να μειώσουμε σημαντικά, τον αριθμό των εξεταζόμενων ακμών από κάθε μία συνάρτηση. Στη συνέχεια, για όσο διάστημα κάποιο από τα τρία φίλτρα δεν είναι άδειο, κάθε τμήμα που αντιστοιχεί σε μία από τις τρεις συναρτήσεις, είναι υπεύθυνο να διαβάσει και να εξετάσει όλες τις ακμές μέχρι να αδειάσουν. Μόλις ολοκληρωθεί και η εξέταση των ακμών από κάθε τμήμα, εκείνο ενημερώνει την μονάδα ελέγχου CONTROL και το τμήμα Rename_Vertex, που είναι υπεύθυνο για την μετονομασία των κόμβων, για την ύπαρξη ακμής του συγκεκριμένου είδους επέκτασης καθώς επίσης και για την ολοκλήρωση της όλης διαδικασίας.

Όλα τα παραπάνω αναπαρίστανται στο σχήμα 4.3.3.



Σχήμα 4.3.3 : Αρχιτεκτονική του τμήματος Edge_Extension (μέσα στο τμήμα Extension).

4.3.4 Μονάδα Ελέγχου CONTROL

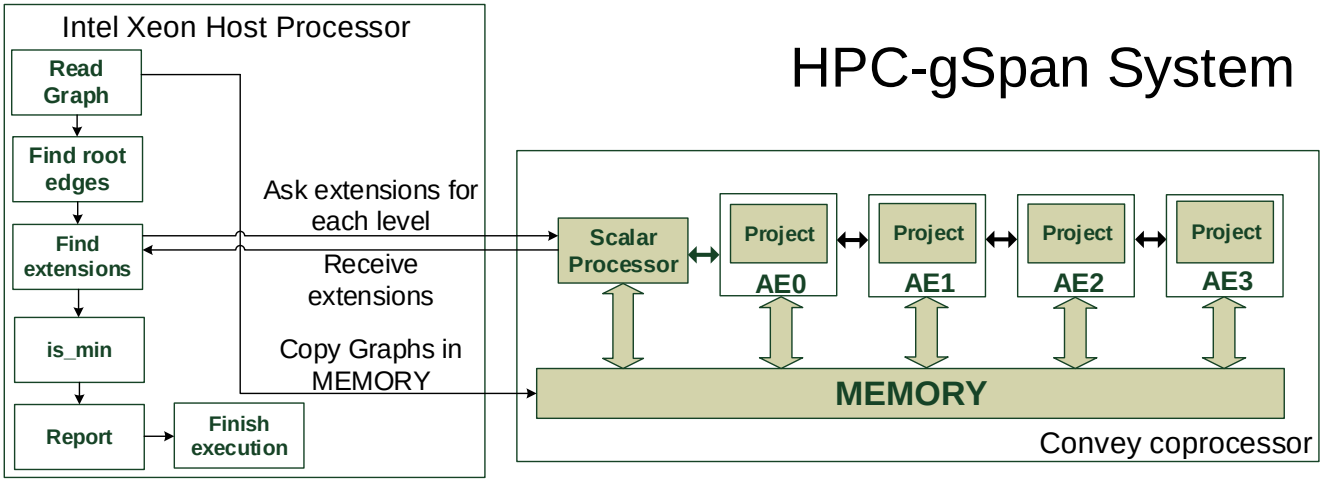
Αυτή η μονάδα είναι υπεύθυνη για το συγχρονισμό και την επικοινωνία κάθε τμήματος μέσα στο κύκλωμα του extension.

Είναι άξιο αναφοράς, το γεγονός ότι χρησιμοποιώντας κομμάτια από τη σχεδίαση του τμήματος της is_min, υλοποιήσαμε με τον πιο αποδοτικό τρόπο τη λειτουργικότητα του τμήματος extension. Το μεγαλύτερο κομμάτι της σχεδίασης που μεταλλάξαμε, είναι το κομμάτι του CONTROL, καθώς το κύκλωμα άλλαξε λειτουργικότητα. Η λειτουργία του περιγράφεται ως εξής, στην αρχή το κύκλωμα δημιουργεί το μονοπάτι που έχει δωθεί και μόλις φτάσει στο επίπεδο που έχει ζητηθεί, τότε ενεργοποιείται η εγγραφή όλων των πιθανών επεκτάσεων στις ουρές εξόδου. Μόλις ολοκληρωθεί η εγγραφή αυτή, ενεργοποιείται η λειτουργία του BACKTRACK, και το κύκλωμα αναζητεί ένα-ένα τα υπόλοιπα μονοπάτια μέσα στο γράφο, τα οποία μπορούν να επεκταθούν στο ζητούμενο επίπεδο.

Η κεντρική μονάδα ελέγχου, κάθε φορά που υπάρχει ακμή για να μπεί στη στοίβα DFS_CODE_MIN, εξετάζει αν είναι ίδια με την αντίστοιχη ακμή του γράφου, που είχε εισαχθεί στην μνήμη του DFS Code. Αν είναι ίδια τόσο λεξικογραφικά όσο και με τα ονόματα των κόμβων, τότε την εισάγει στη στοίβα DFS_CODE_MIN(με τις κατάλληλες ονομασίες των κόμβων από το τμήμα Rename_Vertex) και συνεχίζει την προέκταση του μονοπατιού. Σε διαφορετική περίπτωση, ενεργοποιείται η λειτουργία του BACKTRACK και εξετάζεται κάποιο εναλλακτικό σταυροδρόμι.

4.4 Τελικό Σύστημα High-Performance-Computing gSpan (HPC-gspan)

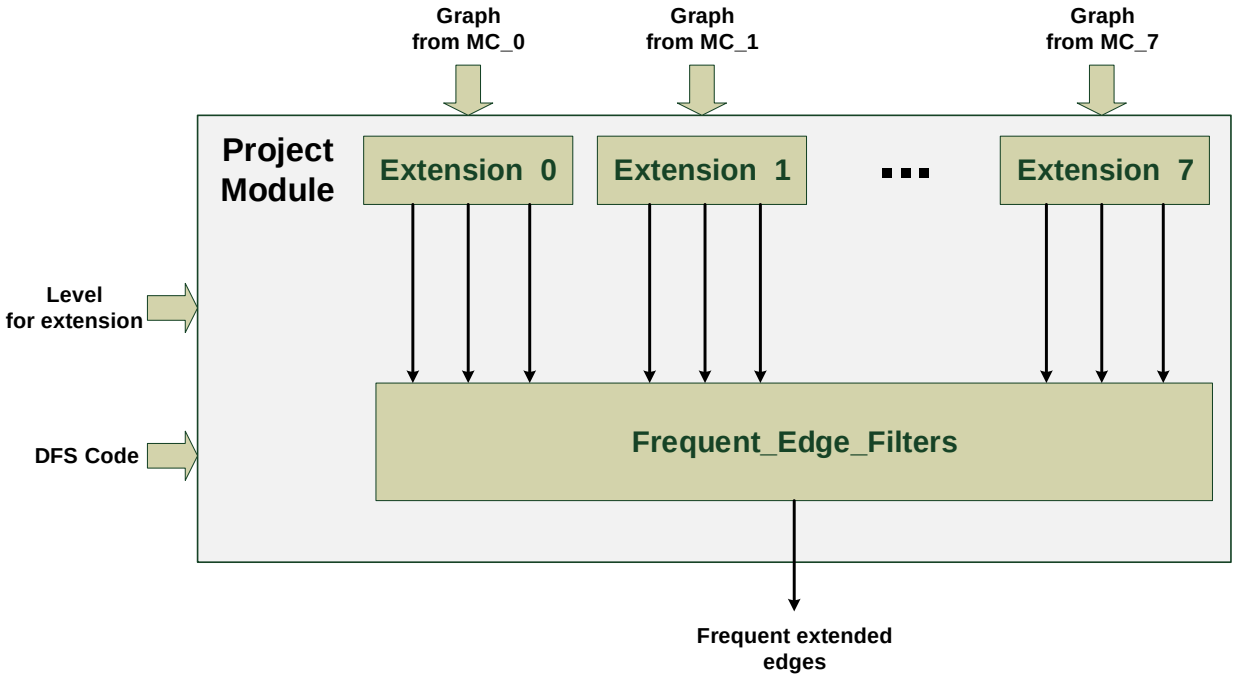
Σε αυτό το σημείο θα γίνει η τελική περιγραφή του συστήματος που σχεδιάστηκε, με σκοπό την επιτάχυνση του χρόνου εκτέλεσης του αλγορίθμου gSpan, με τη χρήση μίας multiple-FPGA πλατφόρμας, όπως φαίνεται στο Σχήμα 4.4.1. Στην αρχή ο αλγόριθμος περνάει όλους τους γράφους που δόθηκαν στην είσοδο στη μνήμη του Coprocessor. Στη συνέχεια, το τελικό σύστημα, αναθέτει τέσσερις διαφορετικές αρχικές ακμές (root edges), στις τέσσερις FPGAs της πλατφόρμας. Έπειτα, σε κάθε FPGA, το σύνολο των αρχικών γράφων χωρίζεται σε οχτώ υποσύνολα, ένα για κάθε παράλληλο τμήμα Extension, που χωράει μέσα σε μια FPGA. Στη συνέχεια, εξετάζονται όλοι οι γράφοι εισόδου από το αντίστοιχο μηχανάκι που έχουν ανατεθεί προηγουμένως. Μετά το πέρας της εξέτασης κάθε γράφου, επιστρέφονται όλες οι πιθανές συχνές προεκτάσεις οι οποίες κρατούνται σε μία ουρά. Έπειτα, ελέγχεται ο ισομορφισμός του υπογράφου πριν εκτυπωθεί και ακολουθώντας μία επαναληπτική διαδικασία, κάθε φορά επιλέγεται μία από τις προεκτάσεις και ξεκινάει η προέκταση του καινούργιου υπογράφου, για κάθε έναν από τους αρχικούς γράφους. Μόλις τελειώσει η επαναληπτική διαδικασία των προεκτάσεων, εξετάζονται όλες εκείνες που είχαν αποθηκευτεί σε κάθε επίπεδο προέκτασης. Όταν ολοκληρωθούν και αυτές, τότε ολοκληρώνεται και η εκτέλεση του αλγορίθμου.



Σχήμα 4.4.1: Αναπαράσταση του τελικού συστήματος HPC-gSpan.

Είναι πολύ σημαντικό να αναφέρουμε, ότι ο τρόπος με τον οποίο το τμήμα Project εξετάζει τις επεκτάσεις των υπογράφων, κλιμακώνεται ανάλογα με το data set εισόδου και η παραλληλία διατηρείται.

Όπως φαίνεται στο σχήμα 4.4.2, κάθε FPGA μπορεί να φέρει εις πέρας την παράλληλη εξέταση έως 8 γράφων για το ίδιο επίπεδο προέκτασης. Μόλις ολοκληρωθεί η προέκταση τους, περνούν όλες οι πιθανές μετονομασμένες προεκτάσεις που προέρχονται από τις τρεις fifo-εξόδου, σε μία μονάδα που ονομάζεται Frequent_Edge_Filters. Από αυτή τη μονάδα απορρίπτονται οι πολλαπλές όμοιες προεκτάσεις και αποθηκεύονται μόνο οι διαφορετικές προεκτάσεις. Στη συνέχεια, εξετάζονται οι επόμενοι οχτώ γράφοι και επιστρέφονται οι επεκτάσεις τους, που εισέρχονται επίσης στη μονάδα Frequent_Edge_Filters. Μόλις, ολοκληρωθεί η εξέταση όλων των γράφων από τη μία FPGA, επιστρέφονται στο software όλες οι συχνές προεκτάσεις του υπογράφου, για ένα συγκεκριμένο επίπεδο.



Σχήμα 4.4.2: Πρότυπη αρχιτεκτονική για κάθε μία FPGA.

ΚΕΦΑΛΑΙΟ 5: Αποτελέσματα

5.1 Γενικά

Το παρόν κεφάλαιο έχει ως αντικείμενο, την παρουσίαση των αποτελεσμάτων της απόδοσης της αρχιτεκτονικής του τελικού συστήματος, σε σύγκριση με την αυθεντική υλοποίηση του αλγόριθμου gSpan. Οι μετρήσεις έγιναν στον ίδιο επεξεργαστή, έναν 4-Core Intel Xeon processor @2,4GHz. Επίσης, παρατίθεται η αναλυτική κατανάλωση των πόρων που καταλαμβάνει κάθε υλοποιημένη αρχιτεκτονική.

5.2 Κατανάλωση πόρων

Σε αυτό το σημείο θα αναλύσουμε τους πόρους που καταναλώνει κάθε κομμάτι σχεδίασης του τελικού συστήματος ξεχωριστά, καθώς και το τελικό σύστημα με τη μέγιστη δυνατή παράλληλία σε μία FPGA Virtex 5 lx330t.

Το πρόβλημα του Frequent Subgraph Mining, προσφέρει μεγάλη παραλληλία, στην εξέταση των επεκτάσεων μέσα σε διαφορετικούς γράφους, για τον ίδιο υπογράφο. Για το λόγο αυτό, επιλέχθηκε το τελικό σύστημα να υλοποιηθεί στην πλατφόρμα Convey HC-1, το οποίο περιλαμβάνει μία πλατφόρμα τεσσάρων FPGAs - οι οποίες μοιράζονται την ίδια μνήμη - και έναν συνεπεξεργαστή, ο οποίος κάνει κλήση σε κάθε FPGA. Για την ανάπτυξη του συστήματος έγινε χρήση του εργαλείου Xilinx ISE Design Suite 13.1 και η σχεδίαση όλων των τμημάτων έγινε σε γλώσσα περιγραφής υλικού VHDL.

Η σύνθεση πραγματοποιήθηκε με το εργαλείο Xilinx ISE 13.1 και προέκυψαν τα ακόλουθα αποτελέσματα, όσον αφορά τη συχνότητα ρολογιού και τους πόρους που καταναλώνει το σύστημα για κάθε τμήμα που σχεδιάσαμε καθώς και για το τελικό σύστημα. Πιο συγκεκριμένα για την αρχιτεκτονική του τμήματος is_min τα αποτελέσματα δίνονται στον Πίνακας 5.2.1. Ενώ, αντίστοιχα τα αποτελέσματα του τμήματος extension και του τελικού συστήματος δίνονται στους Πίνακες 5.2.2 και 5.2.3.

5.2.1 Πόροι τμήματος is_min

Όπως φαίνεται στον Πίνακα 5.2.1, το κρίσιμο σημείο της αρχιτεκτονικής αυτού του τμήματος, είναι οι απαιτήσεις σε μνήμες, καθώς κάθε σχεδίαση για το is_min απαιτεί περίπου 8% των πόρων, που διαθέτει κάθε FPGA.

Clock Frequency : 158.892 MHz

Logic Utilization	Used	Available	Utilization(%)
Number of slice registers	1415	51840	3
Number of Block Ram/FIFO	22	288	8
Number of DSPs	0	192	0

Πίνακας 5.2.1: Πόροι που χρησιμοποιούνται σε μία FPGA για την αρχιτεκτονική του τμήματος is_min.

5.2.2 Πόροι τμήματος Extension

Στον Πίνακα 5.2.2, φαίνεται ότι εφόσον χρησιμοποιήσαμε την αρχιτεκτονική του τμήματος is_min για την ολοκλήρωση της Extension λειτουργίας, το κρίσιμο σημείο της αρχιτεκτονικής αυτού του τμήματος παραμένει η μνήμη, καθώς κάθε σχεδίαση για το τμήμα Extension απαιτεί περίπου 10% των πόρων, που διαθέτει κάθε FPGA.

Clock Frequency : 179.000 MHz

Logic Utilization	Used	Available	Utilization(%)
Number of slices	1545	51840	3
Number of Block Ram/FIFO	28	288	10
Number of DSPs	0	192	0

Πίνακας 5.2.2: Πόροι που χρησιμοποιούνται σε μία FPGA για την αρχιτεκτονική του τμήματος extension.

5.2.3 Πόροι τελικού συστήματος Project

Τέλος, όπως αναμέναμε από τα αποτελέσματα του προηγούμενου πίνακα, σε μία FPGA, χωράνε 8 τμήματα Extension, όσο αφορά τις απαιτήσεις σε μνήμη. Από τον πίνακα 5.2.3, παρατηρούμε όμως ότι το κρίσιμο σημείο της αρχιτεκτονικής του τελικού συστήματος μίας FPGA, είναι οι πόροι σε slices, καθώς το τμήμα Frequent_Edge_Filters που προστέθηκε καταναλώνει κοντά στα 74 % των πόρων των slices μέσα στην FPGA. Επομένως, σε κάθε FPGA να υπάρχει πλήρη κατανάλωση των slices (98%). Αυτό είχε σαν συνέπεια, η διαδικασία του Place&Route να φτάσει κοντά στις 14 ώρες και το ρολόι του συστήματος να πέσει στα 134 MHz.

Clock Frequency : 134.000 MHz

Logic Utilization	Used	Available	Utilization(%)
Number of slices	51140	51840	98
Number of Block Ram/FIFO	241	288	83
Number of DSPs	0	192	0

Πίνακας 5.2.3: Πόροι που χρησιμοποιούνται σε μία FPGA για την αρχιτεκτονική του τμήματος project.

5.3 Δεδομένα

Για την επικύρωση των επικύρωση και τη μέτρηση των αποτελεσμάτων χρησιμοποιήσαμε τα data sets, όπως φαίνονται στον Πίνακα 5.3. Τα data sets έχουν προκύψει από real-life δεδομένα, τα οποία αναπαριστούν χημικές ενώσεις.

Dataset	# Graphs
Chemical	340
Compound	422
MCF-7	25.475
OVCAR	38.436
SN12C	38.048
NCI-H23	38.295

Πίνακας 5.3 : Τα Datasets που μετρήσαμε για το τελικό σύστημα HPC-gSpan.

5.4 Αποτελέσματα τελικού συστήματος HPC-gSpan

Το τελικό σύστημα που περιγράφηκε στο προηγούμενο κεφάλαιο, αποτελείται από τέσσερις FPGAs, στις οποίες έχουν καλυφθεί όλα τα διαθέσιμα slices, όπως φαίνεται από τον Πίνακα 5.2.3. Έτσι, κάθε FPGA περιλαμβάνει από ένα τμήμα Project. Συνεπώς, το τελικό σύστημα περιλαμβάνει 32 παράλληλους εξυπηρετητές για την ανίχνευση κάθε δυνατής προέκτασης ενός υπογράφου. Η επικύρωση των αποτελεσμάτων του συστήματος έγινε με την διασταύρωση των αποτελεσμάτων με την αυθεντική υλοποίηση του αλγορίθμου.

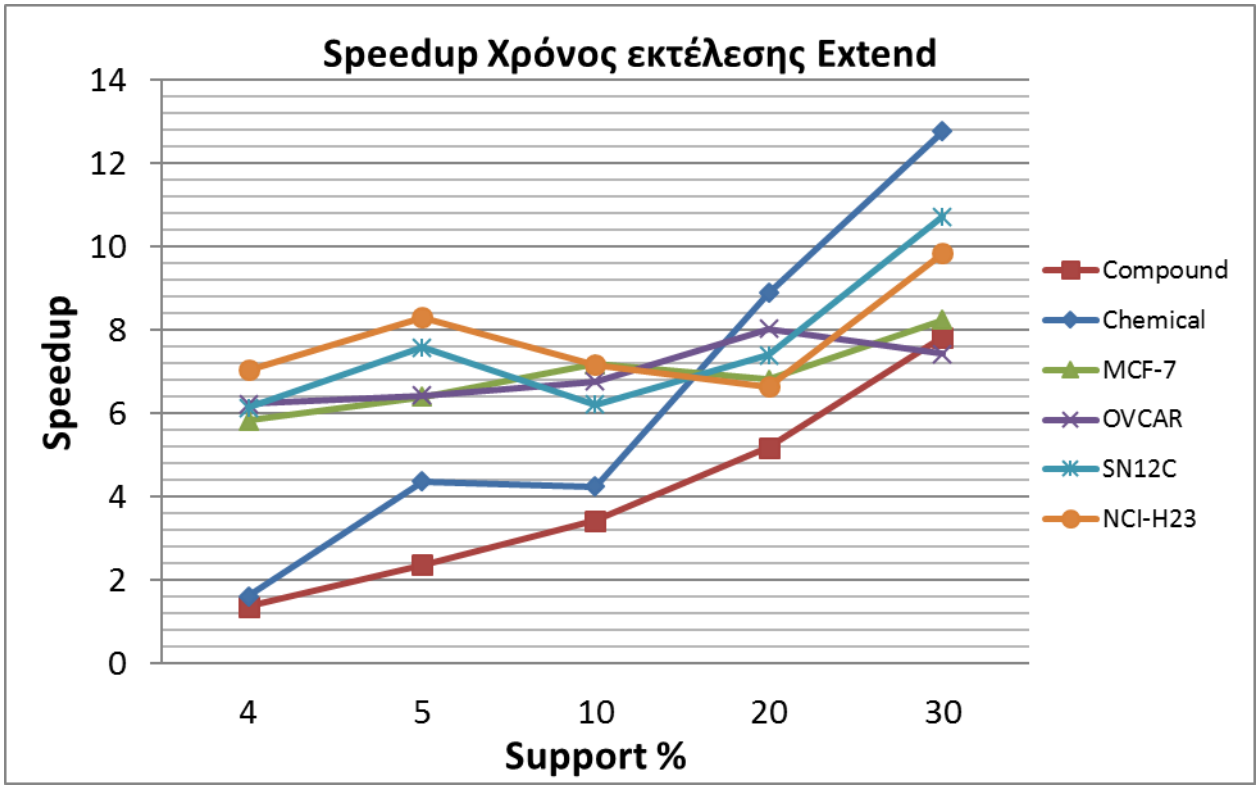
Για να ελέγξουμε την απόδοση του HPC-gSpan, σε σύγκριση με την αρχική υλοποίηση του αλγορίθμου, χρησιμοποιήθηκαν πολλά data sets, τα οποία συναντήθηκαν σε πολλές από τις υπάρχουσες υλοποιήσεις που έχουν γίνει και για τις οποίες έγινε αναφορά στο δεύτερο κεφάλαιο.

Στον πίνακα 5.4.1, παρουσιάζονται τα αποτελέσματα του χρόνου εκτέλεσης μόνο της «βαριάς» συνάρτησης, που είναι η extend τόσο για την υλοποίηση του αυθεντικού αλγορίθμου, το οποίο ονομάσαμε (Original-gSpan SW), όσο και για το προτεινόμενο τελικό σύστημα HPC-gSpan.

Επίσης στην εικόνα 5.4.2 φαίνεται σε γραφική αναπαράσταση το Speed-up που κερδίζει το σύστημα HPC-gSpan, σε σύγκριση με την Original-gSpan SW υλοποίηση, για την εκτέλεση της συνάρτησης extend.

Dataset	No of graphs	Support (%)	Χρόνος Εκτέλεσης Extend Original gSpan SW (sec)	Χρόνος Εκτέλεσης Extend HPC-gSpan (sec)	Speed up
Chemical	340	4	3.59	2.64	1.4x
		5	0.90	0.38	2.4x
		10	0.47	0.14	3.4x
		20	0.31	0.06	5.2x
		30	0.22	0.03	7.8x
Compound	422	4	4.27	2.64	1.6x
		5	1.94	0.44	4.4x
		10	0.73	0.17	4.2x
		20	0.50	0.06	8.9x
		30	0.45	0.04	12.8
MCF-7	25475	4	898.45	154.27	5.8x
		5	700.40	109.49	6.4x
		10	327.15	45.50	7.2x
		20	130.87	19.18	6.8x
		30	98.42	11.93	8.3x
OVCAR	38436	4	1265.74	203.12	6.2x
		5	955.17	148.38	6.4x
		10	430.40	63.61	6.8x
		20	225.09	28.00	8.0x
		30	126.27	16.98	7.4x
SN12C	38048	4	1141.43	186.00	6.1x
		5	1041.45	137.25	7.6x
		10	378.12	60.87	6.2x
		20	197.68	26.74	7.4x
		30	180.69	16.85	10.7x
NCI-H23	38295	4	1380.61	196.06	7.0x
		5	1201.98	144.69	8.3x
		10	451.66	62.98	7.2x
		20	185.51	27.90	6.7x
		30	165.83	16.84	9.9x

Πίνακας 5.4.1: Αποτελέσματα HPC-gSpan vs Original-gSpan για τη συνάρτηση extend.



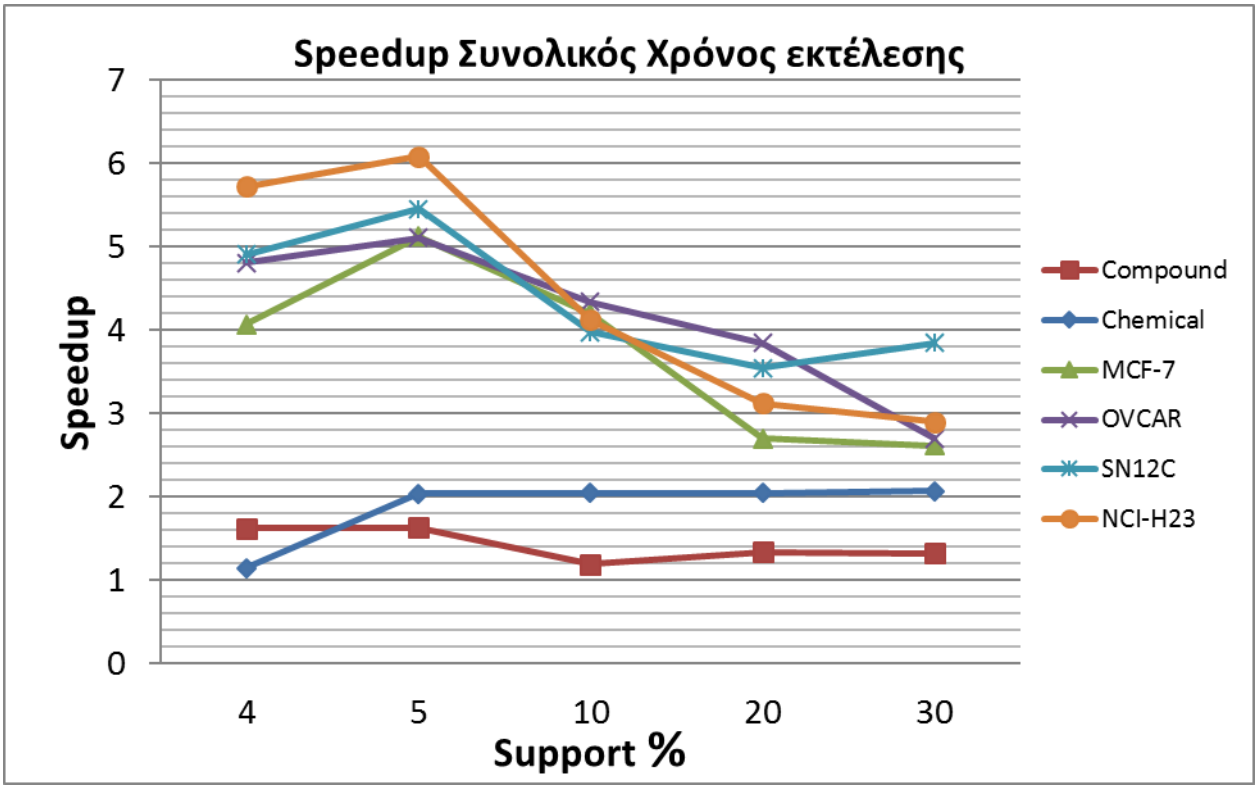
Εικόνα 5.4.2: Αποτελέσματα Speed-up για το χρόνο εκτέλεσης μόνο της συνάρτησης extend (HPC-gSpan vs Original-gSpan).

Επίσης, στον πίνακα 5.4.3, παρουσιάζονται τα αποτελέσματα του χρόνου εκτέλεσης ολόκληρης της εκτέλεσης του αλγορίθμου, τόσο για την υλοποίηση του αυθεντικού αλγορίθμου (Original-gSpan SW), όσο και για το προτεινόμενο τελικό σύστημα HPC-gSpan.

Τέλος, στην εικόνα 5.4.4 φαίνεται σε γραφική αναπαράσταση το Speed-up που κερδίζει το σύστημα HPC-gSpan, σε σύγκριση με την Original-gSpan SW υλοποίηση, για την εκτέλεση ολόκληρου του αλγορίθμου.

Dataset	No of graphs	Support (%)	Συνολικός Χρόνος Εκτέλεσης Original-gSpan SW (sec)	Συνολικός Χρόνος Εκτέλεσης HPC-gSpan (sec)	Speed up
Chemical	340	4	14.37	8.88	1.6x
		5	1.61	0.99	1.6x
		10	0.88	0.74	1.2.1x
		20	0.69	0.51	1.3x
		30	0.59	0.45	1.3x
Compound	422	4	8.27	7.24	1.1x
		5	2.60	1.28	2.0x
		10	1.03	0.51	2.1x
		20	0.79	0.39	2.1x
		30	0.74	0.36	2.1x
MCF-7	25475	4	935.91	229.75	4.1x
		5	737.06	143.83	5.1x
		10	362.54	86.35	4.2x
		20	157.20	58.18	2.7x
		30	133.29	50.97	2.6x
OVCAR	38436	4	1321.08	274.76	4.8x
		5	996.38	195.03	5.1x
		10	470.81	108.49	4.3x
		20	278.26	72.38	3.8x
		30	165.98	61.42	2.7x
SN12C	38048	4	1182.82	241.10	4.9x
		5	1082.57	198.44	5.5x
		10	417.70	105.11	4.0x
		20	249.97	70.55	3.5x
		30	233.01	60.62	3.8x
NCI-H23	38295	4	1422.87	248.46	5.7x
		5	1256.89	206.48	6.1x
		10	504.85	122.37	4.1x
		20	224.97	72.01	3.1x
		30	219.12	75.50	2.9x

Πίνακας 5.4.3: Αποτελέσματα HPC-gSpan vs Original-gSpan για την εκτέλεση ολόκληρου του αλγορίθμου.



Εικόνα 5.4.4: Αποτελέσματα Speed-up για το συνολικό χρόνο εκτέλεσης του αλγορίθμου (HPC-gSpan vs Original-gSpan).

Όπως φαίνεται από τις εικόνες 5.4.2 και 5.4.4, η εκτέλεση της συνάρτησης extend στο σύστημα HPC-gSpan, μπορεί να γίνει έως 12 φορές γρηγορότερα. Από την άλλη μεριά, η συνολική εκτέλεση ολόκληρου του αλγορίθμου, μπορεί να γίνει έως και 6 φορές γρηγορότερα. Αυτό, οφείλεται στο γεγονός ότι στο τελικό σύστημα περιλαμβάνεται ο χρόνος για να αντιγραφούν όλοι οι γράφοι εισόδου, μέσα στην MEMORY του συστήματος. Ένας ακόμη παράγοντας που επηρεάζει το αποτέλεσμα είναι ότι η συνάρτηση extend καλείται πάρα πολλές φορές από τον coprocessor, με αποτέλεσμα να υπάρχει κάθε φορά καθυστέρηση μέχρι να φτάσουν τα κατάλληλα δεδομένα.

ΚΕΦΑΛΑΙΟ 6: Συμπεράσματα – Μελλοντικές επεκτάσεις

6.1 Γενικά

Σε αυτό το κεφάλαιο αναφέρονται τα συμπεράσματα της παρούσας εργασίας και κάποιες επεκτάσεις της προτεινόμενης αρχιτεκτονικής.

6.2 Συμπεράσματα

Η διατριβή αυτή ασχολήθηκε με τη μελέτη του αλγορίθμου gSpan, ο οποίος εξάγει όλους τους συχνούς υπογράφους, μέσα από ένα σύνολο πολλών μικρών γράφων. Κατά τη μελέτη του αλγορίθμου, έγινε κατανοητό η πολυπλοκότητα της δομής των γράφων και η ανάγκη της αποθήκευσης μεγάλης ποσότητας πληροφορίας με σκοπό τη γρήγορη περάτωση όλης της διαδικασίας. Ο στόχος ήταν η δημιουργία ενός συστήματος, το οποίο να είναι πιο αποδοτικό σε σύγκριση με έναν 4-Core Intel Xeon processor @2,4GHz.

6.3 Μελλοντικές Επεκτάσεις

Η αρχιτεκτονική του συνολικού συστήματος, που παρουσιάστηκε σε αυτή τη διπλωματική εργασία, μπορεί να υποστεί ορισμένες αλλαγές με σκοπό την επέκτασή της σε μία πιο ολοκληρωμένη προσέγγιση του αλγορίθμου gSpan.

Πιο συγκεκριμένα οι επεκτάσεις, που προτείνουμε είναι οι εξής:

1) Η χρησιμοποίηση καθρεφτικών φίλτρων στο τμήμα Extension, με σκοπό την υλοποίηση του αλγορίθμου για μη κατευθυνόμενους γράφους. Τα φίλτρα αυτά έχουν δημιουργηθεί και αναλυθεί στο τέταρτο κεφάλαιο κατά την ανάλυση του τμήματος is_min. Ωστόσο, η συγκεκριμένη προσθήκη θα καταναλώσει επιπλέον πόρους σε Block Rams, και κατά συνέπεια θα μειωθεί ο αριθμός των παράλληλων κουτιών που χωρούν σε κάθε FPGA. Αλλά εκτιμάται από τα υπάρχουσα αποτελέσματα, ότι θα έχει εξίσου καλές επιδόσεις.
2) Η αλλαγή της αρχιτεκτονικής, με σκοπό να υποστηρίζεται η εκτέλεση των δεδομένων εισόδου για ποσοστά συχνότητας κάτω από 4%. Η εκτέλεση του αλγορίθμου δείχνει ότι ο περισσότερος χρόνος εκτέλεσης καταναλώνεται σε πολύ μικρές τιμές της συχνότητας.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Thusoo, A., Shao, Z., Anthony, S., Borthakur, D., Jain, N., Sarma, J.S., Murthy, R., and Liu, H. 2010. “Data warehousing and analytics infrastructure at facebook”. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. Pages 1013-1020.

[2] Aggarwal, C. and Wang, H. 2010. “Managing and Mining Graph Data”. Springer, Advances in Database Systems, Volume 40.

[3] Vanetik, N., Gudes, E. and Shimony, S.E. 2002. “Computing frequent graph patterns from semistructured data”. In Proceedings of the IEEE International Conference on Data Mining (ICDM 2002). Pages 458–465.

[4] Inokuchi, A., Washio, T. and Motoda, H. 2000. “An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data”. Pages 13-23.

[5] Kuramoshi, M. and Karypis, G. 2001. “Frequent Subgraph Discovery”. In Proceedings of the IEEE International Conference on Data Mining (ICDM 2001). Pages 313-320.

[6] Bringmann, B. and Nijssen, S. 2008. “What is frequent in a single graph?”. In Proceedings of the 12th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2008). Volume 5012. Pages 858-863.

[7] Fiedler, M. and Borgelt, C. 2007. “Support Computation for Mining Frequent Subgraphs in a Single Graph”. Workshop on Mining and Learning with Graphs (MLG’07).

[8] Lee, M., Yang, L. H., Hsu, W. and Yang, X. 2002. “XClust: Clustering XML Schemas for Effective Integration”. In Proceedings of the 11th International Conference on Information and Knowledge Management (CIKM 2002). Pages 292-299.

[9] Yan, X. and Han, J. 2002. “gSpan: Graph-Based Substructure Pattern Mining”. In Proceedings of the IEEE International Conference on Data Mining (ICDM 2002). Pages 721-724.

[10] Lakshmi, K. and Dr. Meyyappan, T. 2012. “FREQUENT SUBGRAPH MINING ALGORITHMS - A SURVEY AND FRAMEWORK FOR CLASSIFICATION”. Advances in the International Journal of Information Technology Convergence and Services. Volume 2, No.2. Pages 23-39.

[11] Gholami, M. and Salajegheh, A. 2012. “A Survey on Algorithms of Mining Frequent Subgraphs”. In International Journal of Engineering Inventions. Volume 1, Issue 5. Pages 60-63.

[12] Matsuda, T., Horiuchi, T., Motoda, H. and Washio, T. 2000. “Extension of graph-based induction for general graph structured data”. In Proceedings of the 4th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2000). Volume 1805. Pages 420–431.

[13] Inokuchi, A., Washio, T., and Motoda, H. 2003. “Complete mining of frequent patterns from graphs: Mining graph data”. In Machine Learning, Volume 50, Issue 3. Pages 321–354.

[14] Huan, J., Wang, W., and Prins, J. 2003. “Efficient mining of frequent subgraphs in the presence of isomorphism”. In Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM 2003). Pages 549-552.

[15] Huan, J., Wang W., Prins, J. and Yang, J. 2004. “SPIN: Mining Maximal Frequent Subgraphs from Graph Databases”. In Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD’04). Pages 581-586.

[16] Borgelt, C. and Berthold, M. 2002. “Mining Molecular Fragments: Finding Relevant Substructures of Molecules”. In Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM 2003). Pages 51-58.

[17] Nijssen, S. and Kok, J. 2004. “A quickstart in frequent structure mining can make a difference”. In Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD’04). Pages 647-652.

[18] Muggleton, S. 1996. “Stochastic logic programs”. Advances in Inductive Logic Programming. Pages 254-264.

[19] Nijssen, S. and Kok, J. 2001. “Faster association rules for multiple relations”. In Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI’01). Volume 2. Pages 891–896.

[20] Kethar, N., Holder, L. and Cook, D. 2005. “Subdue: Compression-Based Frequent Pattern Discovery in Graph Data”. In Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations (OSDM’05). Pages 71-76.

[21] Rissanen, J.1999. “Hypothesis selection and testing by the MDL principle”. Advances in the Computer Journal. Volume 42, Issue 4. Pages 260–269.

[22] Wörlein, M., Meinl, T., Fischer, I. and Philippsen, M. 2005. “A Quantitative Comparison of the Subgraph Miners MoFa, gSpan, FFSM, and Gaston”. In Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases. Volume 3721. Pages 392-403.

[23] Chakrabarti, D. and Faloutsos C. 2006. “Graph Mining: Laws, Generators and Algorithms”. In Journal ACM Computing Surveys (CSUR).Volume 38, Issue 1. Article No. 2.

[24] Di Fatta, G. and Berthold, M. 2005. “High Performance Subgraph Mining in Molecular Compounds”. In Proceedings of the 1st International Conference on High Performance Computing and Communications (HPCC 2005). Volume 3726. Pages 866-877.

[25] Buehrer, G. and Parthasarathy, S. 2005. “Parallel Graph Mining on Shared Memory Architectures”. Technical report, Ohio State University, Columbus, OH.

[26] Meinl, T., Wörlein, M., Fischer, I. and Philippsen, M. 2006. “Mining Molecular Datasets on Symetric Multiprocessor Systems”. In Proceedings of the IEEE International Conference on Systems, Man and Cybernetics (SMC 2006). Pages 1269-1274.

[27] Meinl, T., Fischer, I. and Philippsen, M. 2005. “Parallel Mining for Frequent Fragments on a Shared-Memory Multiprocessor – Results and Java-Obstacles-”. In Workshopwoche Lernen, Wissensentdeckung, AdaptivitÄat (LWA'05). Pages 196-201.

[28] Reinhardt, S. and Karypis, G. 2007. “A Multi-Level Parallel Implementation of a Program for Finding Frequent Patterns in a Large Sparse Graph”. In Proceedings of the IEEE International Conference on Parallel and Distributed Processing Symbosium (IPDPS 2007). Pages 1-8.

[29] Ranu, S. and Singh, A. 2009. “GraphSig: A Scalable Approach to Mining Significant Subgraphs in Large Graph Databases”. In Proceedings of the 25th IEEE International Conference on Data Engineering (ICDE’09). Pages 844-855.

[30] Wu, B. and Bai, Y. 2010. “An Efficient Distributed Subgraph Mining Algorithm in Extreme Large Graphs”. In Proceedings of the International Conference on Artificial Intelligence and Computational Intelligence (AICI 2010).Volume 6319. Pages 107-115.

[31] Hill, S., Srichandan, B. and Sunderraman, R. 2012. “An iterative mapreduce approach to frequent subgraph mining in biological datasets”. In Proceeding of the ACM Conference on Bioinformatics, Computational Biology and Biomedicine (BCB’12). Pages 661-666.

[32] Ray, A. and Holder, L.B.. 2012. “Efficiency Improvements for Parallel Subgraph Miners”. In the 25th Florida Artificial Intelligence Research Society Conference (FLAIRS 2012). Pages 74-79.

[33] Bhuiyan, M. and Al Hasan, M. 2013. “MIRAGE: An Iterative MapReduce based Frequent Subgraph Mining Algorithm”.

[34] Harish, P. and Narayanan, P.J. 2007. “Accelerating Large Graph Algorithms on the GPU using CUDA”. In Proceedings of the 14th International Conference on High Performance Computing (HiPC 2007). Volume 4873. Pages 197-208.

[35] Hong, S., Oguntebi T. and Olukotun K. 2011. “Efficient Parallel Graph Exploration on Multi-Core CPU and GPU”. In Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT 2011). Pages 78-88.

[36] Merrill, D., Garland, M. and Grimshaw, A. 2012. “Scalable GPU Graph Traversal”. In Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practise of Parallel Programming (PPOPP’12).Volume 47, Issue 8. Pages 117-128.

[37] Wang, W., Dong, J. and Yuan B. 2013. “Graph-Based Substructure Pattern Mining Using CUDA Dynamic Parallelism”.

[38] Ichikawa, S., Saito, H., Udorn, L. and Konishi, K. 2000. “Evaluation of Accelerator Designs for Subgraph Isomorphism Problem”. In Proceedings of 10th International Conference on Field-Programmable Logic and Applications: The Roadmap to Reconfigurable Computing (FPL 2000). Volume 1896. Pages 729-738.

[39] Bondhugula, U., Devulapalli, A., Fernando, J., Wyckoff, P. and Sadayappan, P. 2006. “Parallel FPGA-based All-Pairs Shortest-Paths in a Directed Graph”. In Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS 2006).

[40] Thomas, D., Luk, W. and Stumpl, M. 2007. “Reconfigurable Hardware Acceleration of Canonical Graph Labelling”. In Proceedings of the 3rd International Workshop in Reconfigurable Computing: Architectures, Tools and Applications (ARC 2007).

[41] Betkaoui, B., Thomas, D., Luk, W. and Przulj, N. 2011. “A Framework for FPGA Acceleration of Large Graph Problems: Graphlet Counting Case Study”. In Proceedings of the International Conference on Field-Programmable Technology (FPT 2011).

[42] Betkaoui, B., Wang,Y., Thomas, D. and Luk, W. 2012. “A Reconfigurable Computing Approach for Efficient and Scalable Parallel Graph Exploration”. In Proceedings of the IEEE 23rd International Conference on Application-Specific Systems, Architectures and Processors (ASAP 2012).

[43] Kobori, D. and Maruyama, T. 2012. “An Acceleration of a Graph Cut Segmentation With FPGA”. In Proceedings of the 22nd International Conference on Field-Programmable Logic and Applications (FPL 2012). Pages 407-413.

[44] Krishna, V., Ranga Suri, N. N. R., Athithan, G. 2011. “A comparative survey of algorithms for frequent subgraph discovery”. In Current Science. Volume 100,Issue 2. Pages 190-198.

[45] Voss, S. and Subhlok, J. 2010. “Performance of General Graph Isomorphism Algorithms”. Technical Report UH-CS-09-07, University of Houston.

[46] Ulman, J. T. 1976. “An algorithm for Subgraph Isomorphism”. Journal of the Association for Computing Machinery. Volume 23. Pages 31-42.

[47] Kuramochi, M. and Karypis, G. 2005. “Finding Frequent Patterns in a Large Sparse Graph”. Springer, Advances in Data Mining and Knowledge Discovery. Volume 11, Issue 3. Pages 243-271.