

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ



ΚΑΤΑΝΕΜΗΜΕΝΟΣ ΥΠΟΛΟΓΙΣΜΟΣ

Ράππη Μαρία
Α.Μ.: 4008

Επιβλέπων : Καθηγητής Παύλος Σπυράκης
Συνεπιβλέπων: Δρ. Ιωάννης Χατζηγιαννάκης

Πάτρα
Φεβρουάριος 2012

Περιεχόμενα

1. Εισαγωγή	5
1.1 Κίνητρο και Σημασία	5
1.2 Στόχοι Διπλωματικής Εργασίας	5
1.3 Συνεισφορά της Διπλωματικής Εργασίας	6
1.4 Δομή Διπλωματικής Εργασίας	6
 2. Αλγόριθμος MST Synch GHS	 8
Εισαγωγικοί Ορισμοί	8
2.1 Εισαγωγικά	11
2.2 Ανασκόπηση Spanning trees	13
2.3 Περιγραφή Κατανεμημένου Αλγορίθμου	15
2.4 Ανάλυση Κόστους Επικοινωνίας	20
2.5 Χρονική Ανάλυση	21
2.6 Παράρτημα Αλγορίθμου	22
 3. Self-Stabilizing MST αλγόριθμος	 26
3.1 Περίληψη & Εισαγωγή	26
3.2 Προκαταρκτικά	28
3.3 Περιγραφόμενο Μοντέλο	31
3.4 Ο Self Stabilizing MDST Αλγόριθμος	32
3.5 Ορθότητα Αλγορίθμου	42
3.6 Θέματα Πολυπλοκότητας	53
3.7 Συμπεράσματα	54
 4. Τεχνολογία Υλοποίησης	 56
4.1 Περιβάλλον Υλοποίησης – Shawn	56
4.2 Θέματα Υλοποίησης	64
4.3 Πειραματική Αξιολόγηση	74
 5. Συμπεράσματα και Μελλοντική Κατεύθυνση	 78
5.1 Συμπεράσματα	78
5.2 Μελλοντικές Κατευθύνσεις	79
 6. Βιβλιογραφία	 80

Περίληψη

Στην Παρούσα διπλωματική Εργασία παρουσιάζεται ο σχεδιασμός, η μελέτη και η κατασκευή στο περιβάλλον του *Shawn*, ενός κατανεμημένου αλγόριθμου με σκοπό την κατασκευή ενός δένδρου με ελάχιστο βάρος ακμών (ελάχιστο γεννητικό δένδρο *MST*) σε ένα συνδεδεμένο μη κατευθυνόμενο γράφο με διακριτά βάρη ακμών. Ένας επεξεργαστής υπάρχει σε κάθε κόμβο του γράφου, γνωρίζοντας αρχικά μόνο τα βάρη των παρακείμενων ακμών του. Οι επεξεργαστές υπακούν στον ίδιο αλγόριθμο και ανταλλάσσουν μηνύματα με τους επεξεργαστές των γειτονικών κόμβων έως ότου το δένδρο κατασκευαστεί.

Ευχαριστίες

Με τη διπλωματική εργασία αυτή ολοκληρώνεται ένας κύκλος σπουδών. Στο σημείο αυτό νιώθω την ανάγκη να ευχαριστήσω όλους όσους συνέβαλαν με το δικό τους τρόπο ο καθένας σε αυτή την ολοκλήρωση.

Καταρχάς, θα ήθελα να ευχαριστήσω θερμά τους επιβλέποντες μου Καθηγητή Παύλο Σπυράκη και Δρ. Ιωάννη Χατζηγιαννάκη για το ενδιαφέρον θέμα που μου έδωσαν την ευκαιρία να ασχοληθώ αλλά κυρίως για την εμπιστοσύνη και την καθοδήγησή τους.

Επίσης θα ήθελα να ευχαριστήσω τον ερευνητή Μάριο Λογαρά για την υπομονή και τις πολύτιμες συμβουλές του.

Τέλος θα ήθελα να ευχαριστήσω από τα βάθη της καρδιάς μου την οικογένεια μου, τους γονείς μου Πέτρο και Σπυριδούλα και τα αδέρφια μου Έφη και Γιάννη, αλλά και τους φίλους μου που με συντροφεύουν και με στηρίζουν σε κάθε μου βήμα.

Κεφάλαιο 1°

1.1 Κίνητρο Και Σημασία

Οι κατανεμημένοι MST αλγόριθμοι είναι χρήσιμοι στην επικοινωνία δικτύων όταν ένα δίκτυο επιθυμεί να μεταδώσει πληροφορία από έναν κόμβο σε όλους τους άλλους κόμβους και υπάρχει κόστος για κάθε κανάλι με το οποίο συνδέεται το δίκτυο. Στην υλοποίηση μας το κόστος της επικοινωνίας μεταξύ δυο κόμβων είναι το βάρος της μεταξύ τους ακμής, το οποίο σε πραγματικά δίκτυα θα μπορούσε να είναι είτε απόσταση, είτε οποιοδήποτε άλλο κόστος θα επηρέαζε την επικοινωνία.

Εκτός από την εφαρμογή της μετάδοσης, υπάρχουν πολλά δυνατά προβλήματα ελέγχου των οποίων οι πολυπλοκότητες επικοινωνίας ανάγονται στη γνώση του MST. Με αλλαγές της τοπολογίας προκαλούνται πιθανές αποτυχίες στο δίκτυο, αυτό είναι επιθυμητό ώστε να μπορεί να δημιουργηθεί ένα spanning tree ξεκινώντας από οποιοδήποτε κόμβο ή από οποιοδήποτε υποσύνολο κόμβων, και ο αλγόριθμος εδώ είναι τόσο αποδοτικός όσο κάθε άλλος που θα μπορούσαμε να βρούμε για τη δημιουργία ενός τυχαίου spanning tree.

Τελικά, υπάρχει ένας αριθμός εφαρμογών για κατανεμημένους αλγορίθμους που μπορούμε να επιλέξουμε τον κόμβο με το μεγαλύτερο αναγνωριστικό. Ένας αποδοτικός κατανεμημένος αλγόριθμος για το πρόβλημα αυτό ξεκινά με τον MST αλγόριθμο και έπειτα χρησιμοποιεί το δένδρο που λάβαμε ως αποτέλεσμα για να βρει τον κόμβο με το μεγαλύτερο αναγνωριστικό.

Από όλα τα παραπάνω κατανοούμε λοιπόν πόσο σημαντική γνώση αποτελεί η εύρεση του MST ενός γράφου - δικτύου το οποίο χρησιμοποιούμε για την επικοινωνία και τη μετάδοση πληροφορίας μεταξύ των κόμβων του. Καθώς στις μέρες μας η δύναμη της πληροφορίας και της γνώσης έχουν σημαντική θέση σε όλες τις επιστήμες, στόχος μας είναι να μπορούμε να την αποκτήσουμε με το λιγότερο πιθανό κόστος και με τη μέγιστη δυνατή ταχύτητα. Στην κατεύθυνση αυτή σημαντικό ρόλο παίζει ο αλγόριθμος που σας παρουσιάζουμε πιο κάτω.

1.2 Στόχοι Διπλωματικής Εργασίας

Στόχος της Διπλωματικής Εργασίας είναι ο σχεδιασμός, η μελέτη και η κατασκευή στο περιβάλλον του Shawn, ενός σύγχρονου κατανεμημένου αλγόριθμου με σκοπό την κατασκευή ενός δένδρου με ελάχιστο βάρος ακμών (ελάχιστο γεννητικό δένδρο MST) σε ένα συνδεδεμένο μη κατευθυνόμενο γράφο με διακριτά βάρη ακμών.

Ο αρχικός αλγόριθμος που μελετήσαμε αναφέρεται σε ασύγχρονα δίκτυα, έτσι εμείς κατασκευάσαμε τον αλγόριθμο αυτό να λειτουργεί συγχρονισμένα στο περιβάλλον υλοποίησης του Shawn.

Όσο αφορά τη λειτουργία του αλγορίθμου μας, κάθε κόμβος του γράφου γνωρίζει αρχικά μόνο τα βάρη των παρακείμενων ακμών του. Οι επεξεργαστές υπακούουν στον ίδιο αλγόριθμο και μέσω της ανταλλαγής διαφόρων τύπων μηνυμάτων, τα οποία υπαγορεύει ο αλγόριθμος, με τους επεξεργαστές των γειτονικών κόμβων και με τον τρόπο αυτό κατασκευάζεται το ζητούμενο δένδρο. Κάθε κόμβος – επεξεργαστής ακολουθεί ακριβώς τα ίδια βήματα έως ότου λάβουμε τα τελικά αποτελέσματα. Ο απώτερος σκοπός του αλγορίθμου μας είναι να αποτελέσει ένα πολύτιμο εργαλείο στην επικοινωνία των δικτύων και στη μείωση του κόστους αυτής.

1.3 Συνεισφορά της Διπλωματικής Εργασίας

Όπως αναφέρθηκα και πιο πάνω οι κατανεμημένοι MST αλγόριθμοι είναι χρήσιμοι στην επικοινωνία δικτύων όταν ένα δίκτυο επιθυμεί να μεταδώσει πληροφορία από έναν κόμβο σε όλους τους άλλους κόμβους και υπάρχει κόστος για κάθε κανάλι με το οποίο συνδέεται το δίκτυο. Γεγονός το οποίο καθιστά τη γνώση του MST ενός γράφου πολύ σημαντική για την καλύτερη και βέλτιστη επίλυση οποιουδήποτε τύπου επικοινωνίας ενός δικτύου.

Η συνεισφορά λοιπόν της παρούσας Διπλωματικής Εργασίας είναι να μας δώσει μια συγχρονισμένη υλοποίηση του κατανεμημένου MST αλγόριθμου GHS, με σκοπό να αποτελέσει η υλοποίηση αυτή ένα χρήσιμο εργαλείο για την επικοινωνία δικτύων.

1.4 Δομή Διπλωματικής Εργασίας

Η δομή της εργασίας αυτής είναι η ακόλουθη. Στο κεφάλαιο 2 αναλύεται ο αλγόριθμος βήμα προς βήμα, αφού αρχικά υπενθυμίσουμε κάποιους βασικούς ορισμούς για τους γράφους που ο αναγνώστης θα πρέπει να γνωρίζει.

Πιο αναλυτικά στο υποκεφάλαιο 2.1 περιγράφεται εν συντομία ο αλγόριθμος GHS. Στο υποκεφάλαιο 2.2 γίνεται μια ανασκόπηση των Spanning Trees, ορίζεται ο όρος fragment που χρησιμοποιούμε για την ανάπτυξη του αλγορίθμου, αλλά και πως ο συνδυασμός αυτών των επιμέρους fragment οδηγεί τελικά στο ζητούμενο MST. Στη συνέχεια το υποκεφάλαιο 2.3 μας αναλύει διεξοδικά κάθε λειτουργία του κατανεμημένου αλγορίθμου μας, ποιες και πόσες καταστάσεις κόμβων και ακμών έχουμε, πως γίνεται η κατηγοριοποίηση και η ταξινόμηση των ακμών μας και από ποια στάδια διέρχεται κάθε κόμβος – επεξεργαστής έως ότου ολοκληρωθεί ο αλγόριθμος. Στο υποκεφάλαιο 2.4 παρουσιάζεται η ανάλυση του κόστους επικοινωνίας και στο επόμενο υποκεφάλαιο 2.5 παρουσιάζεται η χρονική ανάλυση του αλγορίθμου αντίστοιχα. Στο τελευταίο τμήμα του κεφαλαίου αυτού, υποκεφάλαιο 2.6, υπάρχει το παράρτημα στο οποίο παρουσιάζεται η λειτουργία του αλγορίθμου υπό μορφή κώδικα.

Στο κεφάλαιο αυτό αναλύεται ένας αυτοσταθεροποιούμενος αλγόριθμος για την κατασκευή ενός MDST σε μη κατευθυνόμενους γράφους. Πιο αναλυτικά Στο 3.1 υποκεφάλαιο αναφέρονται τα εισαγωγικά και εν συντομία η λογική του αλγορίθμου καθώς και τα αποτελέσματα αυτού. Στο υποκεφάλαιο 3.2 δίνονται εισαγωγικές σημειώσεις, ορισμοί και συμβολισμοί που χρησιμοποιούνται για την ανάπτυξη του αλγορίθμου. Στο επόμενο υποκεφάλαιο, 3.3 παρουσιάζεται το μοντέλο του αλγορίθμου αυτού. Στο υποκεφάλαιο 3.4 παρουσιάζεται αναλυτικά ο αυτοσταθεροποιούμενος αλγόριθμος, όπου εξηγείται όλη η διαδικασία, οι μεταβλητές, τα μηνύματα που ανταλλάσσονται και ότι άλλο χρειάζεται για την ορθή λειτουργία του αλγορίθμου αυτού. Στην επόμενη ενότητα, υποκεφάλαιο 3.5 αναλύεται η ορθότητα του αλγορίθμου που παρουσιάστηκε πιο πάνω. Στο υποκεφάλαιο 3.6 παρουσιάζεται η ανάλυση κόστους καθώς και η ανάλυση της χρονικής πολυπλοκότητας του αλγορίθμου. Τέλος, στο υποκεφάλαιο 3.7 δίνονται τα συμπεράσματα που προέκυψαν από την ανάλυση αυτού του αλγορίθμου καθώς και μελλοντικές κατευθύνσεις έπειτα από την ανάλυση αυτή.

Εδώ, στο 4^ο κεφάλαιο παρουσιάζουμε τον αλγόριθμο που υλοποιήσαμε σε κώδικα Shawn σύμφωνα με την ανάλυση που έγινε στο 2^ο κεφάλαιο. Αναφέρουμε τις σχεδιαστικές υποθέσεις και τροποποιήσεις που κάναμε ώστε να καταφέρουμε να λειτουργεί συγχρονισμένα ο κατανεμημένος αλγόριθμος που παρουσιάστηκε στο 2^ο κεφάλαιο. Πιο αναλυτικά, στο υποκεφάλαιο 4.1 παρουσιάζουμε το περιβάλλον υλοποίησης μας, το οποίο δεν είναι άλλο από το Shawn. Στο υποκεφάλαιο 4.2 παρουσιάζουμε αναλυτικά τα θέματα της υλοποίησης μας, εξηγώντας κάθε λογική και μεθοδολογία που ακολουθήσαμε. Στο τελευταίο υποκεφάλαιο της ενότητας αυτής, υποκεφάλαιο 4.3, παρουσιάζονται και αναλύονται τα πειραματικά αποτελέσματα που λάβαμε από την συγκεκριμένη υλοποίηση του αλγορίθμου.

Το τελευταίο κεφάλαιο απαρτίζεται από δυο επιμέρους υποκεφάλαια 5.1 και 5.2 αντίστοιχα. Στο πρώτο παρουσιάζονται τα συμπεράσματα του αλγορίθμου που υλοποιήσαμε και στο επόμενο οι μελλοντικές κατευθύνσεις που μας δίνει η υλοποίηση αυτή.

Κεφάλαιο 2°

Πριν αναλύσουμε τον αλγόριθμο μας παραθέτουμε κάποιους χρήσιμους ορισμούς τους οποίους ο αναγνώστης θα πρέπει να γνωρίζει.

Εισαγωγικοί ορισμοί

Ένα **γράφημα** $G=(V,E)$ αποτελείται από ένα πεπερασμένο (μη κενό) σύνολο κορυφών V , και ένα πεπερασμένο σύνολο ακμών E .

Κάθε ακμή e είναι είτε ένα σύνολο δυο κορυφών $e=\{a,b\}$ όπου $a \neq b$, οπότε χαρακτηρίζεται **μη-κατευθυνόμενη**, ή ένα ζεύγος κορυφών $e=(a,b)$ (όχι αναγκαστικά διαφορετικών), οπότε χαρακτηρίζεται **κατευθυνόμενη**.

Μια ακμή, $\{a,b\}$ είτε (a,b) , προσπίπτει στις κορυφές a,b οι οποίες ονομάζονται **άκρα** της ακμής. Η κορυφή a ονομάζεται αρχή της κατευθυνόμενης ακμής (a,b) και η κορυφή b τέλος της αντίστοιχα. Οι κορυφές a,b ονομάζονται γειτονικές όταν το γράφημα έχει μια ακμή με άκρα τις a,b .

Παρατηρήσεις:

- Η μη κατευθυνόμενη ακμή $\{a,b\}$ είναι ίδια με την μη-κατευθυνόμενη ακμή $\{b,a\}$. Η κατευθυνόμενη ακμή (a,b) είναι διαφορετική από την κατευθυνόμενη ακμή (b,a) , αν $a \neq b$.
- Μια μη-κατευθυνόμενη ακμή $\{a,b\}$ παριστάνεται γραφικά με μια συνεχή γραμμή (οποιοδήποτε σχήματος) ανάμεσα στα άκρα της. Αντιθέτως μια κατευθυνόμενη ακμή (a,b) παριστάνεται γραφικά με ένα βέλος από την αρχή της προς το τέλος της.
- Ο βαθμός μιας κορυφής a , $\phi(a)$ είναι ο αριθμός των ακμών που προσπίπτουν στην a .
- Ένα γράφημα που όλες οι ακμές του είναι κατευθυνόμενες ονομάζεται κατευθυνόμενο, ενώ αντίστοιχα αν όλες οι ακμές του είναι μη-κατευθυνόμενες ονομάζεται μη-κατευθυνόμενο.
- Το γράφημα $G'=(V',E')$ λέγεται υπογράφημα του $G=(V,E)$, όταν $V' \subseteq V$ και $E' \subseteq E$. Το $G'=(V',E')$ λέγεται επαγόμενο υπογράφημα του $G=(V,E)$ όταν $V' \subseteq V$ και το E' περιέχει ακριβώς τις ίδιες ακμές του G που έχουν και τα δύο άκρα τους στο V' .

Σημείωση:

Παρακάτω όπου θα αναφέρουμε τον όρο **fragment** για τον αλγόριθμο μας εννοούμε πως κάθε fragment είναι ένα υπογράφημα του συνολικού δοσμένου γραφήματος.

Ονομάζουμε **διαδρομή** μια ακολουθία $(a_1, e_1, a_2, e_2, a_3, \dots, a_n, e_n, a_{n+1})$ με $n \geq 1$, όπου κάθε a_i είναι κορυφή, κάθε e_i είναι ακμή, και είναι $e_i = (a_i, a_{i+1})$ ή $e_i = \{a_i, a_{i+1}\}$, για κάθε $i = 1, 2, 3, \dots, n$.

Το **μήκος της διαδρομής** είναι ο αριθμός n .

Οι κορυφές a_1, a_{n+1} είναι τα **άκρα** της διαδρομής, και ονομάζονται αρχή και τέλος της αντίστοιχα. Όταν τα άκρα της ταυτίζονται η διαδρομή λέγεται κλειστή, αλλιώς ανοικτή.

Ονομάζουμε **ίχνος** μια διαδρομή $(a_1, e_1, a_2, e_2, a_3, \dots, a_n, e_n, a_{n+1})$ που δεν επαναλαμβάνει ακμές (δηλαδή, οι όροι της ακολουθίας που είναι ακμές διαφορετικές μεταξύ τους).

Ονομάζουμε **μονοπάτι** ένα ίχνος $(a_1, e_1, a_2, e_2, a_3, \dots, a_n, e_n, a_{n+1})$ που δεν επαναλαμβάνει κορυφές (δηλαδή, οι όροι της ακολουθίας που είναι ακμές διαφορετικές μεταξύ τους).

Ονομάζουμε **κύκλο** ένα κλειστό ίχνος $(a_1, e_1, a_2, e_2, a_3, \dots, a_n, e_n, a_{n+1})$ που δεν επαναλαμβάνει κορυφές, εκτός από την αρχική a_1 που ταυτίζεται με την τελική a_{n+1} . Οι κύκλοι $(a_1, e_1, a_2, e_2, a_3, \dots, a_n, e_n, a_{n+1})$ και $(a_i, e_i, a_{i+1}, \dots, a_n, e_n, a_{n+1}, a_1, e_1, a_2, \dots, a_{i-1}, e_{i-1}, a_i)$ που προκύπτουν ο ένας από τον άλλο με κυκλική μετάθεση, θεωρούνται ταυτόσημοι.

Ένα γράφημα που δεν περιέχει κύκλο ονομάζεται άκυκλο.

Οι κορυφές a, b ενός γραφήματος συσχετίζονται μέσω της **προσβασιμότητας**, R_δ , αν και μόνο αν υπάρχει μονοπάτι με αρχή το a και τέλος το b .

Σε κάθε μη-κατευθυνόμενο γράφημα $G=(V, E)$ η σχέση $\{(x, y) \mid x R_\delta y \text{ ή } x=y\}$ πάνω στο V είναι σχέση ισοδυναμίας, και οι κλάσεις ισοδυναμίας της αποτελούν ένα διαμερισμό $\{V_1, \dots, V_n\}$ του V . Τα επαγόμενα υπο-γράφηματα του G με σύνολα κορυφών τα $V_1, \dots, V_n$ ονομάζονται **συνεκτικές συνιστώσες** του G .

Η συνεκτική συνιστώσα μιας κορυφής a του G είναι το (μοναδικό) σύνολο V_i του V που περιέχει το a .

Το G λέγεται **συνεκτικό** όταν έχει μόνο μια συνεκτική συνιστώσα.

Μια ακμή e ενός μη-κατευθυνόμενου γραφήματος G είναι **γέφυρα** αν το $G-e$ έχει περισσότερες συνεκτικές συνιστώσες από το G .

Ονομάζουμε **δέντρο** ένα μη-κατευθυνόμενο γράφημα που είναι συνεκτικό και άκυκλο.

Ονομάζουμε **υπό-δέντρο** ενός δέντρου, ένα συνεκτικό υπογράφημα του.

Ονομάζουμε **δάσος** ένα μη-κατευθυνόμενο γράφημα, που κάθε συνεκτική συνιστώσα του είναι δέντρο.

Ένα **δέντρο επικάλυψης – S.T.** (Spanning Tree) ενός συνεκτικού γραφήματος G είναι ένα υπό-γράφημα του G που είναι δέντρο, και περιέχει όλες τις κορυφές του G .

Ένα δάσος επικάλυψης ενός μη-κατευθυνόμενου γραφήματος G είναι ένα υπό-γράφημα του G που είναι δάσος, περιέχει όλες τις κορυφές του G , και έχει τον ίδιο αριθμό συνεκτικών συνιστωσών με το G .

Ένα **ελάχιστο δέντρο επικάλυψης – M.S.T.** (Minimum Spanning Tree) ενός συνεκτικού γραφήματος G είναι ένα υπό-γράφημα του G που είναι δέντρο, περιέχοντας όλες τις κορυφές του G με βάρος μικρότερο ή ίσο με το βάρος του αρχικού Δέντρου.

Γενικότερα οποιοδήποτε μη-κατευθυνόμενο γράφημα (όχι απαραίτητα συνδεδεμένο) έχει ελάχιστο δάσος το οποίο είναι μια ένωση των ελαχίστων δέντρων που εκτείνονται για να συνδεθούν όλα τα συστατικά του.

Περισσότερους ορισμούς και διευκρινήσεις όσο αφορά τους γράφους μπορούμε να βρούμε στα ακόλουθα συγγράμματα [14], [15].

■

“Ένας Κατανεμημένος αλγόριθμος για την εύρεση δένδρου με το ελάχιστο βάρος ακμών”

Παρουσιάζεται ένας κατανεμημένος αλγόριθμος για την κατασκευή ενός δένδρου με ελάχιστο βάρος ακμών (ελάχιστο γεννητικό δένδρο MST) σε ένα συνδεδεμένο μη κατευθυνόμενο γράφο με διακριτά βάρη ακμών. Ένας επεξεργαστής υπάρχει σε κάθε κόμβο του γράφου, γνωρίζοντας αρχικά μόνο τα βάρη των παρακείμενων ακμών του. Οι επεξεργαστές υπακούν στον ίδιο αλγόριθμο και ανταλλάσσουν μηνύματα με τους επεξεργαστές των γειτονικών κόμβων έως ότου το δένδρο κατασκευαστεί. Ο συνολικός αριθμός μηνυμάτων που απαιτείται για ένα γράφο N κόμβων και E ακμών είναι το λιγότερο $5N \log_2 N + 2E$, και κάθε μήνυμα περιέχει το λιγότερο ένα βάρος ακμών της τάξης των $\log_2 8N$ bit. Ο αλγόριθμος μπορεί να αρχικοποιηθεί αυθόρμητα από οποιοδήποτε κόμβο ή από οποιοδήποτε υποσύνολο κόμβων.

Πριν την κατασκευή του αλγορίθμου μας μελετήσαμε κι άλλους κατανεμημένους αλγορίθμους που υλοποιούν αναζήτηση κατά βάθος όπως ο DFS αλλά και εύρεση ελαχίστων επαναληπτικών δένδρων όπως ο BFS. **[1], [3], [6], [7], [8].**

2.1 Εισαγωγή

Θεωρούμε ένα συνδεδεμένο γράφο ο οποίος έχει N κόμβους και E ακμές, με ένα διακριτό και πεπερασμένο βάρος να έχει οριστεί για κάθε ακμή. Περιγράφουμε ένα ασύγχρονο κατανεμημένο αλγόριθμο ο οποίος καθορίζει το ελάχιστο βάρος του γεννητικού δένδρου (MST) του αρχικού γράφου. Υποθέτουμε ότι κάθε κόμβος αρχικά γνωρίζει τα βάρη των παρακείμενων ακμών του.

Κάθε κόμβος εκτελεί τον ίδιο τοπικό αλγόριθμο, ο οποίος περιέχει την αποστολή μηνυμάτων πάνω από τις γειτονικές συνδέσεις, περιμένοντας για εισερχόμενα μηνύματα, και συνεχίζουν κάπως έτσι τη διαδικασία. Τα μηνύματα μπορούν να διαβιβαστούν ανεξάρτητα προς δυο κατευθύνσεις σε μια ακμή και να φθάσουν έπειτα από απρόβλεπτη αλλά πεπερασμένη καθυστέρηση, χωρίς κάποιο διαδοχικό λάθος. Έπειτα κάθε κόμβος ολοκληρώνει τον τοπικό του αλγόριθμο, γνωρίζοντας ποια από τις παρακείμενες ακμές του περιέχονται τελικά στο δένδρο και γνωρίζει επίσης ποια από τις ακμές του οδηγεί σε κάποια ιδιαίτερη ακμή η οποία αποτελεί και τον πυρήνα του δένδρου.

Θεωρούμε τους κόμβους του γραφήματος μας αρχικά να βρίσκονται στην κατάσταση *asleep*, δηλαδή να είναι αρχικά αδρανής. Ένας ή περισσότεροι από τους κόμβους έπειτα “ξυπνούν”, αυθόρμητα ή από τη λήψη μηνύματος από κάποιον μη-αδρανή γείτονα κόμβο, και έπειτα συνεχίζουν με τον τοπικό τους αλγόριθμο.

Κάτω υπό αυτές τις προϋποθέσεις, μπορούμε να δούμε πως ο συνολικός αριθμός των μηνυμάτων που ανταλλάσσεται από τους κόμβους του τελικού MST είναι το λιγότερο $5N \log_2 N + 2E$. Κάθε μήνυμα περιέχει το πολύ ένα βάρος ακμών, έναν ακέραιο αριθμό μεταξύ του μηδενός και του $\log_2 N$, και τρία πρόσθετα bit.

Ο αλγόριθμος αυτός είναι παρόμοιος με έναν αλγόριθμο που είχε ανακαλυφθεί νωρίτερα και περιγράφηκε από τον Spira [11], ο οποίος ακολούθησε έναν ακόμη προγενέστερο αλγόριθμο ο οποίος δόθηκε από τον Dalal [12]. Ο Spira δεν ανέλυσε με λεπτομέρεια τον αριθμό των μηνυμάτων που απαιτείται για τον αλγόριθμο του, αλλά έδωσε ένα ευρετικό επιχείρημα πως ο αναμενόμενος αριθμός μηνυμάτων (έπειτα από τυχαία επιλεγμένα γραφήματα) θα αυξάνονταν με τους παραμέτρους E και N , ως εξής: $E + N \log N$.

Εάν οι κόμβοι του δικτύου έχουν διακριτά αναγνωριστικά ώστε να μπορούν να διαταχθούν, έπειτα είναι εύκολο να επεκτείνουμε τον αλγόριθμο στην περίπτωση όπου τα βάρη των ακμών μας δεν είναι διακριτά. Στην περίπτωση αυτή η ακμή επισυνάπτει στο βάρος της τα αναγνωριστικά (ταυτότητες, ids) των δυο κόμβων που συνδέει, αριθμώντας για παράδειγμα πρώτα τον πιο χαμηλό διατεταγμένο κόμβο. Αυτή η επισύναψη βαρών έχει την ίδια ακολουθία όπως η προηγούμενη παραδοχή μας, με τους δεσμούς να σπάζουν από τα αναγνωριστικά των κόμβων. Αν οι κόμβοι δεν γνωρίζουν τα αναγνωριστικά των γειτονικών κόμβων, τότε κάθε κόμβος μπορεί να στείλει το αναγνωριστικό του σε όλους τους γείτονες με τη χρήση των ακμών τους, απαιτώντας κατά συνέπεια ένα σύνολο της τάξης $2E$ επιπρόσθετων μηνυμάτων. Έχουμε αναπτύξει επίσης έναν άλλο αλγόριθμο, ο οποίος εν συντομία περιγράφεται πιο κάτω, για τον οποίο επίσης δεν απαιτείται να υπάρχουν διακριτά βάρη ακμών αλλά δεν απαιτεί $2E$ επιπρόσθετα μηνύματα.

Αν το δίκτυο δεν έχει ούτε διακριτά βάρη ακμών αλλά ούτε και διακριτά αναγνωριστικά σε κάθε κόμβο, τότε κανένας κατανεμημένος αλγόριθμος (από τους τύπους αλγορίθμων που περιγράψαμε πιο πάνω) δεν υπάρχει για την εύρεση ενός MST με φραγμένο αριθμό μηνυμάτων. Αυτό μπορεί να γίνει πιο εύκολα εμφανές σε ένα τριπλό κόμβο πλήρως συνδεδεμένο γράφο με ίσα βάρη ακμών. Κάθε δυο ακμές θα συνθέτουν ένα MST, αλλά από τη στιγμή που όλοι οι κόμβοι ακολουθούν τον ίδιο τοπικό αλγόριθμο, δεν υπάρχει τρόπος να επιλέξουν. Αν οι κόμβοι επιλέξουν τυχαία αναγνωριστικά, τότε ο αλγόριθμος θα μπορούσε να λειτουργήσει μόνο αν όλα τα αναγνωριστικά είναι διαφορετικά μεταξύ τους, αλλά παρόλο αυτά δεν υπάρχει τρόπος να εγγυηθούμε τη λειτουργία του αλγορίθμου σε ένα πεπερασμένο αριθμό επιλογών. Φυσικά, ο αναμενόμενος αριθμός των απαιτούμενων επιλογών είναι μικρός, αλλά οποιοσδήποτε φραγμένος αριθμός μηνυμάτων θα αποτύχει με θετική πιθανότητα.

Οι κατανεμημένοι MST αλγόριθμοι είναι χρήσιμοι στην επικοινωνία δικτύων όταν ένα δίκτυο επιθυμεί να μεταδώσει πληροφορία από έναν κόμβο σε όλους τους άλλους κόμβους και υπάρχει κόστος για κάθε κανάλι με το οποίο συνδέεται το δίκτυο. Αν το κόστος που χρησιμοποιεί ένα κανάλι για τη μια κατεύθυνση είναι διαφορετικό από το κόστος της αντίθετης κατεύθυνσης, τότε

το MST δεν παρέχει την επιθυμητή λύση, αλλά ένα συνοδευτικό έγγραφο επεξεργάζεται αυτό το γενικό πρόβλημα [13]. Εκτός από την εφαρμογή της μετάδοσης, υπάρχουν πολλά δυνατά προβλήματα ελέγχου των οποίων οι πολυπλοκότητες επικοινωνίας ανάγονται στη γνώση του MST. Με αλλαγές της τοπολογίας προκαλούνται πιθανές αποτυχίες στο δίκτυο, αυτό είναι επιθυμητό ώστε να μπορεί να δημιουργηθεί ένα spanning tree ξεκινώντας από οποιοδήποτε κόμβο ή από οποιοδήποτε υποσύνολο κόμβων, και ο αλγόριθμος εδώ είναι τόσο αποδοτικός όσο κάθε άλλος που θα μπορούσαμε να βρούμε για τη δημιουργία ενός τυχαίου spanning tree. Τελικά, υπάρχει ένας αριθμός εφαρμογών για κατανεμημένους αλγορίθμους που μπορούμε να επιλέξουμε τον κόμβο με το μεγαλύτερο αναγνωριστικό. Ένας αποδοτικός κατανεμημένος αλγόριθμος για το πρόβλημα αυτό ξεκινά με τον MST αλγόριθμο και έπειτα χρησιμοποιεί το δένδρο που λάβαμε ως αποτέλεσμα για να βρει τον κόμβο με το μεγαλύτερο αναγνωριστικό.

2.2 Ανασκόπηση Spanning tree

Υποθέτουμε πως κάθε ακμή e ενός γράφου έχει βάρος $w(e)$ συνδεδεμένη με αυτό. Το βάρος ενός δένδρου σε ένα γράφο ορίζεται ως το άθροισμα των βαρών των επιμέρους ακμών του δένδρου, και το αντικείμενο μας είναι να βρούμε ένα δένδρο με ελάχιστο βάρος ακμών, το οποίο θα είναι ένα MST. Ένα **fragment** ενός MST ορίζεται ως ένα υποδένδρο του MST, το οποίο είναι συνδεδεμένο με ένα σύνολο κόμβων και ακμών του συνολικού MST. Ο αλγόριθμος ξεκινά με ένα επιμέρους κόμβο ως fragment και τερματίζει με το τελικό MST ως fragment. Καθορίζουμε μια ακμή ως εξερχόμενη ακμή ενός fragment αν ο ένας προσπίπτον σε αυτή κόμβος είναι μέρος του fragment ενώ ο δεύτερος όχι.

Ιδιότητα 1'

Δοσμένου ενός fragment ενός MST, έστω e η εξερχόμενη ακμή με το μικρότερο βάρος του fragment. Έπειτα συνδέοντας την ακμή e με τις προσπίπτον σε αυτήν κόμβους οι οποίοι δεν ανήκουν στο fragment δημιουργείται ένα άλλο fragment του MST.

Απόδειξη:

Υποθέτουμε ότι η προστιθέμενη ακμή e δεν περιέχεται στο MST το οποίο συμπεριλαμβάνει το πρότυπο fragment. Έπειτα υπάρχει ένας κύκλος ο οποίος διαμορφώνεται από την ακμή e και ένα υποσύνολο ακμών του MST. Το λιγότερο θα υπάρχει μια ακμή $x \neq e$ του κύκλου αυτού, Η ακμή x είναι επίσης εξερχόμενη ακμή του fragment, έτσι θα ισχύει $w(x) \geq w(e)$. Κατά συνέπεια, διαγράφοντας την ακμή x από το MST και προσθέτοντας την ακμή e δημιουργείται ένα νέο spanning tree το οποίο πρέπει να είναι ελάχιστο αν το πρότυπο δένδρο ήταν ελάχιστο. Το πρότυπο fragment με την προσθήκη της ακμής e είναι το fragment του νέου MST. ■

Ιδιότητα 1^η

Αν όλες οι ακμές ενός συνεκτικού γράφου έχουν διαφορετικά βάρη, τότε το MST είναι μοναδικό.

Απόδειξη:

Υποθέτουμε αντιθέτως, ότι υπάρχουν δυο διαφορετικά MSTs. Έστω e η ακμή με το μικρότερο βάρος η οποία είναι μια και μοναδική και δεν περιέχεται και στα δυο δένδρα, έστω T ένα σύνολο ακμών του MST το οποίο περιέχει την ακμή e , και T' ένα σύνολο ακμών του δεύτερου MST. Το σύνολο ακμών $\{e\} \cup \{T\}$ πρέπει να περιέχει κύκλο, και το λιγότερο μια ακμή αυτού του κύκλου, έστω e' , δεν περιέχεται στο σύνολο T (από τη στιγμή που το σύνολο T δεν περιέχει κύκλους). Από τη στιγμή που τα βάρη των ακμών είναι διαφορετικά και η ακμή e' είναι μοναδική αλλά δεν περιέχεται και στα δυο δένδρα, ισχύει $w(e) < w(e')$. Κατά συνέπεια το σύνολο: $\{e\} \cup \{T\} - \{e'\}$ είναι ένα σύνολο ακμών ενός spanning tree με μικρότερο βάρος από αυτό του T' , έτσι βλέπουμε πως δημιουργείται αντίφαση και καταλήγουμε σε άτοπο. ■

Οι παραπάνω δυο ιδιότητες άμεσα προτείνουν ένα γενικό τύπο αλγορίθμου για εύρεση ενός MST σε ένα γράφο με διαφορετικά βάρη ακμών. Κάποιος μπορεί να ξεκινήσει με ένα ή περισσότερα fragments τα οποία περιέχουν μονούς κόμβους. Χρησιμοποιώντας την πρώτη ιδιότητα, κάποιος μπορεί να διευρύνει αυτά τα fragments με οποιαδήποτε σειρά. Όποτε δυο fragments έχουν κοινό κόμβο, η δεύτερη ιδιότητα μας βεβαιώνει πως η ένωση των δυο αυτών fragments αποτελεί επίσης fragment, επιτρέποντας τα fragments να μπορούν να ενωθούν σε μεγαλύτερα fragments. Οι πρότυποι αλγόριθμοι για την δημιουργία MSTs αντιστοιχούν σε διαφορετικές διαταγές στις οποίες τα ανωτέρω fragments διευρύνονται και συνδυάζονται. Για παράδειγμα, ο αλγόριθμος του Prim-Dijkstra [9, 10] ξεκινά με ένα μονό κόμβο και διαδοχικά διευρύνει το fragment έως ότου εκτείνεται ο δοσμένος γράφος. Ο αλγόριθμος Kruskal ξεκινά με όλους τους κόμβους του ως ξεχωριστό fragment ο κάθε ένας από αυτούς και διαδοχικά εκτείνει το fragment με τη μικρότερη εξερχόμενη ακμή, συνδυάζοντας επιμέρους fragments όπου είναι πιθανό. Άλλοι αλγόριθμοι πάλι ξεκινούν με όλους τους κόμβους ως ένα fragment για κάθε έναν από αυτούς, επεκτείνοντας κάθε fragment, έπειτα συνδυάζοντας τα, και εκτείνοντας κάθε ένα από τα νέα διευρυμένα fragments, συνδυάζοντας τα ξανά, και συνεχίζοντας με τον ίδιο τρόπο μέχρι την επίτευξη του τελικού MST.

Οι αλγόριθμοι Prim-Dijkstra και Kruskal [16] εργάζονται εξίσου καλά αν κάποια από τα βάρη των ακμών τους είναι ίδια. Για να το διαπιστώσουμε αυτό, απλώς επιβάλλουμε μια αυθαίρετη διαταγή στις ακμές με το ίδιο βάρος με τις επιλογές που γίνονται κατά την εκτέλεση των αλγορίθμων. Άλλοι αλγόριθμοι [11, 12, 17] εκτείνουν διαφορετικά fragments χωρίς ενδιάμεσο συνδυασμό δεν λειτουργούν απαραίτητα σωστά για ακμές με ίσα βάρη. Για παράδειγμα, σε ένα πλήρως συνδεδεμένο δίκτυο με τρεις κόμβους με ίσα βάρη ακμών, κάθε κόμβος μπορεί να επεκτείνει μια διαφορετική ακμή, δίνοντας αφορμή για δημιουργία κύκλου από το συνδυασμό των επιμέρους fragment.

Στον ακόλουθο αλγόριθμο, κάθε fragment βρίσκει την εξερχόμενη ακμή του με το μικρότερο βάρος ασύγχρονα όσον αφορά τα υπόλοιπα fragment, και

έπειτα η ακμή αυτή μεταβαίνει στην κατάσταση found, το fragment προσπαθεί να συνδυάσει την ακμή αυτή με το fragment από την άλλη πλευρά της ακμής αυτής. Πως και πότε ο συνδυασμός αυτός των fragment λαμβάνει χώρα εξαρτάται από το επίπεδο στο οποίο βρίσκεται το κάθε fragment, το οποίο εξαρτάται από το συνδυασμό των fragment στον προηγούμενο γύρο. Συγκεκριμένα, ένα fragment περιέχει ένα μονό κόμβο ο οποίος αρχικά ορίζεται να βρίσκεται στο επίπεδο 0. Υποθέτοντας πως ένα δοσμένο fragment F βρίσκεται στο επίπεδο $L \geq 0$ και ένα fragment F' ,από την άλλη πλευρά της εξερχόμενης ακμής του fragment F με το μικρότερο βάρος, βρίσκεται στο επίπεδο L' . Αν $L > L'$, τότε το fragment F αμέσως απορροφάται ως τμήμα του fragment F' , και το επεκταμένο fragment βρίσκεται στο επίπεδο L' . Αν $L = L'$ και τα fragments F και F' έχουν την ίδια εξερχόμενη ακμή με το μικρότερο βάρος, τότε τα δυο fragments συνδυάζονται αμέσως σε ένα νέο fragment επιπέδου $L+1$, ο συνδυασμός της κοινής ακμής ονομάζεται πυρήνας (core) του νέου fragment. Σε όλες τις άλλες περιπτώσεις, το fragment F απλώς περιμένει έως ότου το fragment F' φθάσει ένα αρκετά υψηλό επίπεδο ώστε να μπορούν τα δυο fragments να συνδυαστούν υπό τις παραπάνω προϋποθέσεις.

Η εικόνα 1 πιο κάτω παρουσιάζει αυτούς τους κανόνες. Το fragment F βρίσκεται στο επίπεδο 1, το fragment F δημιουργήθηκε από την ένωση των κόμβων 1 και 2 μέσω της κοινής εξερχόμενης ακμής τους με το μικρότερο βάρος, και ο κόμβος 3 με την κοινή ακμή με το μικρότερο βάρος έπειτα απορροφώνται. Τα fragment F και F' έπειτα συνδυάζονται μέσω της κοινής εξερχόμενης ακμής τους με το μικρότερο βάρος και συνθέτουν ένα νέο fragment επιπέδου 2, και ο κόμβος 4 έπειτα απορροφάται. Ανάλογα με το συγχρονισμό, θα υπήρχε πιθανότητα ο κόμβος 4 να είχε απορροφηθεί νωρίτερα από το fragment F , πριν τη δημιουργία του τελικού fragment επιπέδου 2.

Δείχνουμε αργότερα, έπειτα από την αναλυτική περιγραφή του αλγορίθμου , πως η υπακοή και η αναμονή στους παραπάνω κανόνες έχει σαν αποτέλεσμα να μη βρεθούμε ποτέ σε αδιέξοδο (deadlock). Η αιτία της αναμονής είναι ότι για την επικοινωνία που απαιτείται μεταξύ των fragments, απαιτεί αρχικά για κάθε ένα επιμέρους fragment να έχει βρεθεί η εξερχόμενη ακμή της με το ελάχιστο βάρος, γεγονός το οποίο είναι ανάλογο προς το μέγεθος του επιμέρους fragment, και έτσι η επικοινωνία μειώνεται από τα μικρά fragment που συνδυάζονται και δημιουργούν τα μεγάλα fragment, παρά αντίστροφα.

2.3 Περιγραφή του κατανεμημένου αλγορίθμου

Ξεκινάμε περιγράφοντας πως ένα fragment βρίσκει την εξερχόμενη του ακμή με το ελάχιστο βάρος. Πρώτα θεωρούμε την τετριμμένη ειδική περίπτωση του μηδενικού fragment (π.χ. ένας μόνο κόμβος επιπέδου 1). Αρχικά κάθε κόμβος βρίσκεται σε μια αδρανή κατάσταση η οποία ονομάζεται sleeping. Υπάρχουν τρεις πιθανές καταστάσεις στις οποίες μπορούν να βρεθούν οι κόμβοι: η αρχική κατάσταση **sleeping**, η κατάσταση **Find** η οποία συμμετέχει

στην αναζήτηση του fragment για την εύρεση της εξερχόμενης ακμής με το ελάχιστο βάρος, και η κατάσταση **Found** για όλες τις υπόλοιπες πιθανές φάσεις. Όταν ένας κόμβος βρίσκεται στην κατάσταση sleeping και αυθόρμητα αφυπνίζεται και ξεκινά το συνολικό αλγόριθμο ή είναι ήδη αφυπνισμένος από την παραλαβή οποιουδήποτε μηνύματος του αλγορίθμου, από οποιοδήποτε γειτονικό κόμβο, ο κόμβος που θα επιλέξει πρώτα είναι εκείνος ο γειτονικός κόμβος που έχει το μικρότερο βάρος ακμής, έπειτα μαρκάρει την ακμή ως τμήμα του MST, και στέλνει ένα μήνυμα για επικοινωνία μέσω της ακμής αυτής, και έπειτα ο κόμβος αυτός μεταβαίνει στην κατάσταση found περιμένοντας απάντηση από το fragment για τον άλλο κόμβο που συνδέεται με την ίδια ακμή.

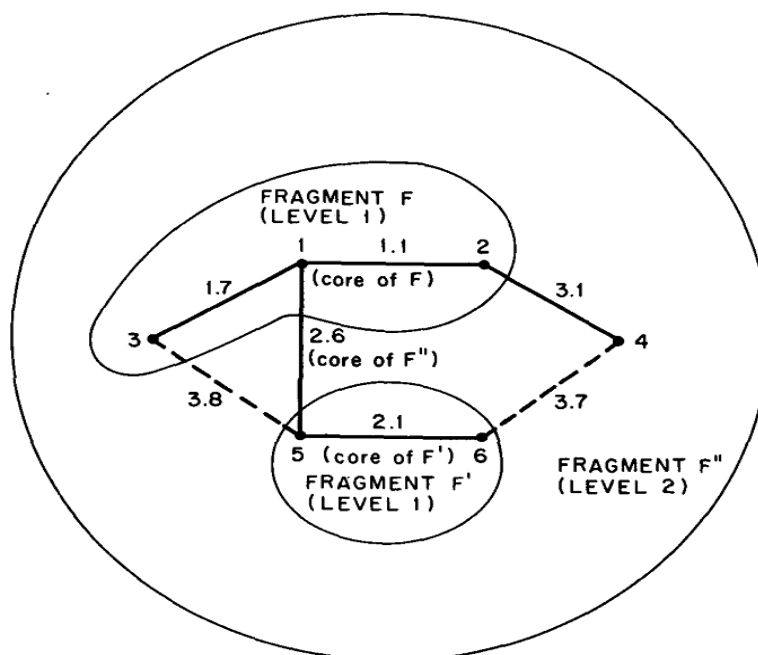


Fig. 1. Fragments and cores.

Τώρα έστω το πρόβλημα του πως ακριβώς οι κόμβοι του μηδενικού επιπέδου του fragment θα συνεργαστούν για να βρουν την εξερχόμενη ακμή με το μικρότερο βάρος. Υποθέτουμε ένα νέο fragment στο επίπεδο L το οποίο μόλις διαμορφώθηκε από το συνδυασμό fragments επιπέδου $L-1$, τα οποία είχαν κοινή την εξερχόμενη ακμή τους με το ελάχιστο βάρος (ακμή e), όπου η ακμή αυτή και οι δύο κόμβοι τους οποίους συνδέει θα γίνει ο πυρήνας του νέου fragment που δημιουργήθηκε από την συνένωση των δυο προηγούμενων. Το βάρος της ακμής αυτού του πυρήνα χρησιμοποιείται ως αναγνωριστικό του fragment που δημιουργήθηκε.

Οι δυο γειτονικοί κόμβοι του πυρήνα ξεκινούν ένα νέο γύρο επικοινωνίας στέλνοντας μήνυμα αρχικοποίησης στους υπόλοιπους κόμβους του fragment.

Αυτό το μήνυμα στέλνεται στις εξωτερικές ακμές του fragment και αναμεταδίδεται εξωτερικά από τους ενδιάμεσους κόμβους του fragment. Αυτό το αρχικό μήνυμα μεταφέρει το επίπεδο του νέου fragment και την ταυτότητα του όπως αναφέρθηκε πιο πάνω, παρέχοντας σε όλους τους κόμβους του φράγματος αυτή την πληροφορία. Το αρχικό μήνυμα επίσης περιέχει το επιχείρημα Find, το οποίο τοποθετεί τον εκάστοτε κόμβο στην κατάσταση Find. Αν τα άλλα fragments του επιπέδου L-1 περιμένουν να συνδεθούν με τους κόμβους του νέου fragment του επιπέδου L, το αρχικό μήνυμα δίνεται και σ' αυτούς επίσης, τοποθετώντας τους στο νέο fragment. Το αρχικό μήνυμα επίσης δίνεται στα fragments του επιπέδου L-1 τα οποία περιμένουν να συνδεθούν με τους κόμβους αυτούς, και ούτω κάθε εξής.

Όταν ένας κόμβος λάβει το μήνυμα της αρχικοποίησης, ξεκινά την αναζήτηση της εξερχόμενης ακμής με το ελάχιστο βάρος. Η δυσκολία εδώ είναι ότι ο κόμβος δεν γνωρίζει ποιές ακμές είναι εξερχόμενες. Η δυσκολία αυτή λύνεται ως ακολούθως: κάθε κόμβος ο οποίος ταξινομεί κάθε μια από τις προσπίπτουσες ακμές του σε μια από τις τρεις πιθανές καταστάσεις:

- Branch: Αν η ακμή αυτή είναι ακμή του τρέχοντος fragment.
- Rejected: Αν η ακμή αυτή δεν είναι ακμή του τρέχοντος fragment αλλά έχει ανακαλυφθεί προκειμένου να ενώσει δυο κόμβους του fragment.
- Basic: Αν η ακμή αυτή δεν ανήκει σε καμία από τις παραπάνω κατηγορίες αλλά δεν έχει απορριφθεί.

Προκειμένου να βρούμε την εξερχόμενη ακμή με το ελάχιστο βάρος, ένας κόμβος διαλέγει μια ακμή Basic και στέλνει ένα δοκιμαστικό μήνυμα σ' αυτή. Το μήνυμα αυτό περιέχει την ταυτότητα και το επίπεδο του fragment. Όταν ένας κόμβος λαμβάνει ένα δοκιμαστικό μήνυμα, ελέγχει αν η ταυτότητα (το αναγνωριστικό) του fragment στο οποίο βρίσκεται είναι ίδια με την ταυτότητα του μηνύματος. Αν οι δυο ταυτότητες συμφωνούν, τότε ο κόμβος στέλνει το μήνυμα Reject πίσω στον αποστολέα, και οι δυο αυτοί κόμβοι τοποθετούν την κοινή τους ακμή στην κατάσταση Rejected. Ο κόμβος ο οποίος έστειλε δοκιμαστικό μήνυμα, συνεχίζει την αποστολή τέτοιων μηνυμάτων με τις επόμενες ακμές. Η εξαίρεση εδώ είναι η ακόλουθη: αν ένας κόμβος στείλει αλλά και λάβει ένα δοκιμαστικό μήνυμα με την ίδια ταυτότητα μέσω της ίδιας ακμής, εύκολα απορρίπτει την ακμή αυτή χωρίς να απορρίπτει το μήνυμα, αυτό μειώνει λιγάκι την πολυπλοκότητα επικοινωνίας.

Αν ένας κόμβος λάβει ένα δοκιμαστικό μήνυμα και έχει διαφορετική ταυτότητα fragment από το fragment του κόμβου από τον οποίο έλαβε το μήνυμα, και αν το επίπεδο του fragment στο οποίο ανήκει είναι μεγαλύτερο ή ίσο με το επίπεδο του fragment του μηνύματος, τότε ένα μήνυμα Accept στέλνεται πίσω στον αποστολέα, πιστοποιώντας ότι η ακμή είναι εξερχόμενη ακμή του κόμβου του αποστολέα του fragment. Αν απ' την άλλη πλευρά, το επίπεδο του fragment του κόμβου που έλαβε το μήνυμα είναι μικρότερο από αυτό του μηνύματος, τότε ο κόμβος που έλαβε το μήνυμα καθυστερεί να δώσει οποιαδήποτε απάντηση έως ότου το επίπεδο του fragment στο οποίο ανήκει αυξηθεί ικανοποιητικά. Ο πιο σημαντικός λόγος αυτής της καθυστέρησης είναι ότι, έπειτα από ένα χαμηλό επίπεδο fragment συνδυάζονται σε ένα υψηλότερο επίπεδο, οι εξερχόμενοι κόμβοι του fragment δεν περιμένουν την αλλαγή του

επιπέδου για ένα αβέβαιο χρονικό διάστημα (στην πραγματικότητα περιμένουν έως ότου λάβουν ένα νέο μήνυμα αρχικοποίησης). Μια σημαντική ιδιοκτησία του αλγορίθμου είναι ότι η ταυτότητα του fragment ενός κόμβου αλλάζει όταν και μόνο όταν το επίπεδο αυξάνεται. Επιπλέον μια δοσμένη ταυτότητα fragment μπορεί να εμφανιστεί μόνο σε ένα επίπεδο. Αυτές οι προϋποθέσεις είναι διαισθητικά ξεκάθαρες από την προηγούμενη συζήτηση για το πώς συνδυάζονται τα fragments και πώς μπορούν να καθιερωθούν αυστηρά από την επαγωγή για επιτρεπόμενη χρονική περίοδο του αλγορίθμου. Από αυτές τις προϋποθέσεις για τα fragments, μπορούμε να δούμε πώς ένας κόμβος A στέλνει ένα μήνυμα Accept ως απάντηση σε ένα δοκιμαστικό μήνυμα του κόμβου B, τότε η ταυτότητα του fragment του A διαφέρει, και θα συνεχίσει να διαφέρει απ' αυτή του fragment του B.

Έχουμε μόλις περιγράψει πώς οι κόμβοι του fragment τελικά βρίσκουν τις εξερχόμενες ακμές με το ελάχιστο βάρος, αν υπάρχουν. Οι κόμβοι πρέπει τώρα να συνεργαστούν, στέλνοντας μηνύματα Report, για να βρουν την εξερχόμενη με το ελάχιστο βάρος από το δοσμένο fragment. Αν ένας κόμβος δεν έχει εξερχόμενες ακμές, ο αλγόριθμος ολοκληρώθηκε, και το fragment είναι το MST που ψάχνουμε. Πιο ειδικά κάθε φύλλο οποιουδήποτε κόμβου του fragment, το οποίο είναι, για κάθε γειτονικό κόμβο μόνο μια ακμή του fragment στέλνει μήνυμα Report (w) στις γειτονικές ακμές, όπου w: είναι το βάρος της εξερχόμενης ακμής του κόμβου με το ελάχιστο βάρος. Το βάρος w είναι ∞ όταν δεν υπάρχουν άλλες εξωτερικές ακμές. Παρομοίως κάθε εσωτερικός κόμβος του φράγματος περιμένει έως ότου βρει τη δική του εξερχόμενη ακμή με το ελάχιστο βάρος και λάβει μηνύματα από όλες τις εξερχόμενες ακμές του fragment. Ο κόμβος τότε δείχνει την ακμή (είτε εξερχόμενη ακμή του, είτε την εξερχόμενη ακμή του fragment) στην οποία το μικρότερο απ' αυτά τα βάρη, w, βρέθηκε ως καλύτερο άκρο, και ο κόμβος στέλνει Report(w) στις εισερχόμενες ακμές. Όταν ο κόμβος στείλει μήνυμα Report, πηγαίνει επίσης στην κατάσταση Found, με την ένδειξη της ολοκλήρωσης του ρόλου του για την εύρεση της εξερχόμενης ακμής του fragment με το ελάχιστο βάρος. Τελικά δυο γειτονικοί κόμβοι του πυρήνα στέλνουν report μήνυμα στον πυρήνα της ίδιας ακμής, επιτρέποντας σε κάθε έναν απ' αυτούς τους κόμβους να καθορίσουν από κοινού το βάρος της εξερχόμενης ακμής με το ελάχιστο βάρος και την πλευρά του πυρήνα στην οποία αυτή η άκρη βρίσκεται.

Έπειτα από δυο πυρήνες κόμβων οι οποίοι ανταλλάσσουν μηνύματα Report, η καλύτερη άκρη σωζόμενης κοντά στο fragment του κόμβου που κάνει πιθανή την επισήμανση του μονοπατιού από τον πυρήνα έως τον κόμβο ο οποίος περιέχει την εξερχόμενη ακμή με το ελάχιστο βάρος. Το μήνυμα change-core στέλνεται σε κάθε ακμή του μονοπατιού και οι εισερχόμενες ακμές για κάθε έναν από αυτούς τους κόμβους αλλάζουν για να ανταποκριθούν στην καλύτερη ακμή (best edge). Όταν το μήνυμα φτάνει στον κόμβο με την εξερχόμενη ακμή με το μικρότερο βάρος, η εισερχόμενη ακμή από ένα δένδρο με ρίζα, γίνεται ο κόμβος αυτός. Τελικά, αυτός ο κόμβος στέλνει το μήνυμα connect(L) σε όλες τις εξερχόμενες ακμές με το ελάχιστο βάρος, όπου L είναι το επίπεδο του fragment. Αν δυο fragments του επιπέδου L έχουν την ίδια εξερχόμενη ακμή με το μικρότερο βάρος, τότε κάθε ένας στέλνει μήνυμα

Connect(L) μέσω της κοινής ακμής, ένα σε κάθε κατεύθυνση, κι αυτό προκαλεί να γίνει η ακμή αυτή ο πυρήνας του fragment του επιπέδου L+1 και προκαλεί ένα νέο γύρο μηνυμάτων αρχικοποίησης με το νέο επίπεδο του fragment να στέλνει την ταυτότητα του. Αυτός ο κανόνας για τη διαμόρφωση νέων φραγμάτων εξασφαλίζει ότι το fragment του επιπέδου L+1 πάντα περιέχει το λιγότερο 2^L κόμβους, και ως συνέπεια αυτού το $\log_2 N$ είναι ένα άνω φράγμα για τα επίπεδα του fragment.

Τελικά θεωρούμε τι συμβαίνει όταν ένα μήνυμα connect από ένα κόμβο n, σε ένα κόμβο μικρότερου επιπέδου fragment, όπου L: το επίπεδο και F: το αναγνωριστικό (id) του fragment, φθάνει έναν κόμβο n' υψηλότερου επιπέδου fragment με L' και F'.

Εξαιτίας της στρατηγικής όπου ποτέ δεν κάνουμε τα χαμηλά επίπεδα των fragments να περιμένουν, ο κόμβος n' αμέσως στέλνει ένα μήνυμα αρχικοποίησης με τον αριθμό του επιπέδου του και την ταυτότητα του fragment F', το fragment F απλά ενώνει το fragment F' και συμμετέχει στην εύρεση της εξερχόμενης ακμής με το μικρότερο βάρος από το διευρυμένο fragment. Αν από την άλλη πλευρά ο κόμβος n' έχει ήδη στείλει μήνυμα report, τότε μπορούμε να συνάγουμε ότι η εξερχόμενη ακμή του κόμβου n' έχει πιο μικρό βάρος από την εξερχόμενη ακμή του F με το μικρότερο βάρος. Αυτό αποβάλλει την ανάγκη για το F να συμμετέχει στην αναζήτηση για την εξερχόμενη ακμή με το ελάχιστο βάρος. Αυτές οι δυο περιπτώσεις είναι κοντά στέλνοντας την κατάσταση του κόμβου, είτε Find είτε Found, στο αρχικό μήνυμα. Οι κόμβοι του fragment F πηγαίνουν στην κατάσταση Find ή Found βασιζόμενοι στην παράμετρο της αρχικοποίησης μηνύματος, και στέλνουν μηνύματα test (δοκιμής) μόνο στην κατάσταση Find.

Τώρα περιγράφουμε την απόδειξη πως ο αλγόριθμος μας είναι ορθός. Κατά την άποψη των ιδιοτήτων 1 και 2 πιο πάνω είναι ικανοποιητικό να ελεγχθεί ότι ο αλγόριθμος πράγματι βρίσκει την εξερχόμενη ακμή με το μικρότερο βάρος για κάθε fragment και πως η αναμονή για την εύρεση των ακμών αυτών δεν οδηγεί σε αδιέξοδο. Η προηγούμενη περιγραφή του αλγορίθμου πρέπει να πείσει τον αναγνώστη πως η οποία λαμβάνει ένα μήνυμα Connect(L) είναι η εξερχόμενη ακμή με το ελάχιστο βάρος για το fragment του επιπέδου L το οποίο απαρτίζεται από όλους τους κόμβους οι οποίοι έλαβαν το ίδιο μήνυμα αρχικοποίησης με τη δοσμένη ταυτότητα (σημειώστε ότι το fragment που αντιστοιχεί σε μια δεδομένη ταυτότητα fragment μπορεί να αυξηθεί αφού ως γνωστόν τα fragments χαμηλότερου επιπέδου ενώνονται με το δοσμένο fragment).

Για να δείξουμε ότι αδιέξοδα δεν υπάρχουν, θεωρούμε ένα σύνολο fragments τα οποία υπάρχουν σε οποιαδήποτε στιγμή, αποκλείοντας τα μηδενικά fragments (zero-level) τα οποία απαρτίζονται από sleeping κόμβους. Υποθέτουμε ότι ο αλγόριθμος έχει ήδη ξεκινήσει αλλά δεν έχει τερματίσει ακόμη, έτσι ώστε τα ανωτέρα σύνολα fragments δεν είναι κενά και κάθε ένα από αυτά έχει μια εξερχόμενη ακμή με ελάχιστο βάρος. Εκτός από το χαμηλότερο σε επίπεδο fragment του συνόλου, θεωρούμε ένα fragment με την μικρότερη εξερχόμενη ακμή από όλα τα υπόλοιπα. Οποιοδήποτε δοκιμαστικό μήνυμα (test message) από το fragment αυτό είτε θα αφυπνίσει ένα sleeping

zero-level fragment είτε θα αποκριθεί χωρίς αναμονή. Ομοίως, ένα μήνυμα επικοινωνίας (connect message) από το fragment αυτό είτε θα αφυπνίσει ένα sleeping zero-level fragment, είτε θα μεταβούμε σε ένα fragment υψηλότερου επιπέδου (με άμεση αρχική απόκριση), ή θα μεταβούμε σε ένα fragment του ίδιου επιπέδου με την ίδια εξερχόμενη ακμή, οδηγώντας πια τα δυο αυτά fragments σε ένα fragment μεγαλύτερου επιπέδου κατά 1. Δεδομένου ότι το υποτιθέμενο σύστημα καταστάσεων ήταν αυθαίρετο, βλέπουμε πως αδιέξοδα δεν υπάρχουν.

Το πρόγραμμα στο τμήμα του παραρτήματος πιο κάτω επίσης δοκιμάστηκε (χρησιμοποιώντας εφαρμογή σε γλώσσα προγραμματισμού Fortran) σε μια ποικιλία τοπολογιών δικτύου.

Σύντομα τώρα περιγράφουμε μια τροποποίηση του αλγορίθμου η οποία μπορεί να χρησιμοποιηθεί για μη διακριτά βάρη ακμών όπου δεν απαιτούνται 2Ε επιπρόσθετα μηνύματα για την επισύναψη στους κόμβους τα βάρη των παρακείμενων ακμών σε αυτούς. Στην τροποποίηση των fragments προσδιορίζεται από τις ταυτότητες των κόμβων, οι οποίες είναι ταξινομημένες και διακριτές. Μια εξερχόμενη ακμή ενός fragment με το ελάχιστο βάρος βρέθηκε όπως προηγουμένως, και ένα μήνυμα επικοινωνίας (connect message) στέλνεται μέσω της ακμής αυτής όμοια με προηγουμένως. Το νέο χαρακτηριστικό γνώρισμα εδώ είναι ένα μήνυμα επικοινωνίας της υποτιθέμενης ακμής e του fragment F που αποστέλλεται στο fragment F' αργότερα ακυρώνεται αν τα δυο fragments βρίσκονται στο ίδιο επίπεδο και $F > F'$, μερικά fragments F'' του ίδιου επιπέδου έχουν στείλει μήνυμα επικοινωνίας στο fragment F και ισχύει $F'' < F$, για ένα μήνυμα αρχικοποίησης που στάλθηκε από την ακμή e δεν έχει ακόμα σταλεί πίσω επιβεβαίωση. Όταν ένα μήνυμα επικοινωνίας ακυρώνεται, ο κόμβος ο οποίος το είχε στείλει μειώνει κατά ένα το επίπεδο του και στέλνει ένα νέο μήνυμα επικοινωνίας, στην περίπτωση αυτή τα fragments F και F'' ενώνονται.

2.4 Ανάλυση κόστους επικοινωνίας

Καθορίζουμε εδώ ένα ανώτερο όριο του αριθμού των μηνυμάτων που ανταλλάσσονται κατά τη διάρκεια της εκτέλεσης του αλγορίθμου. Σημειώστε ότι το πιο σύνθετο μήνυμα περιέχει μια ακμή βάρους, ενός επιπέδου μεταξύ του μηδενικού και του $\log N$, και μερικά ακόμη bits τα οποία δείχνουν τον τύπο του μηνύματος.

Δεδομένου ότι μια ακμή μπορεί να απορριφθεί μόνο μια φορά, και κάθε απόρριψη απαιτεί δυο μηνύματα, υπάρχουν το λιγότερο 2Ε δοκιμαστικά μηνύματα ή μηνύματα απόρριψης που οδηγούν στις απορρίψεις.

Επειτα, ενώ ένας κόμβος βρίσκεται σε ένα δοσμένο επίπεδο εξαιρώντας το πρώτο και το τελευταίο, μπορεί να δεχθεί το πολύ ένα μήνυμα αρχικοποίησης και ένα μήνυμα αποδοχής. Μπορεί να διαβιβαστεί το πολύ

επιτυχημένο δοκιμαστικό μήνυμα, ένα μήνυμα αναφοράς, και ένα μήνυμα αλλαγής ρίζας ή μήνυμα επικοινωνίας. Δεδομένου ότι το $\log_2 N$ είναι το άνω φράγμα του υψηλότερου επιπέδου, ένας κόμβος μπορεί να περάσει το πολύ από $(-1 + \log N)$ επίπεδα χωρίς να υπολογίζουμε το πρώτο και το τελευταίο, και αυτό αποτελεί το πολύ $5N(-1 + \log N)$ μηνύματα.

Στο μηδενικό επίπεδο, κάθε κόμβος μπορεί να λάβει το πολύ ένα μήνυμα αρχικοποίησης και μπορεί να μεταδώσει το πολύ ένα μήνυμα επικοινωνίας. Στο τελευταίο επίπεδο, κάθε κόμβος μπορεί να στείλει το πολύ ένα μήνυμα αναφοράς (report message). Αυτό προσθέτει λιγότερο από $5N$ μηνύματα στο μεγάλο σύνολο μας, το οποίο γίνεται τελικά $5N \log N + 2E$.

Σημειώστε ότι, αν ο αριθμός των κόμβων του γράφου αρχικά δεν είναι γνωστός, όπως έχουμε υποθέσει σιωπηρά, τότε κανένας κατανεμημένος αλγόριθμος δεν θα μπορεί να βρει το MST με λιγότερα από E μηνύματα, αν υπάρχει ακμή πάνω από την οποία κανένα μήνυμα δεν έχει σταλεί, τότε εκεί μάλλον υπάρχει ένας κόμβος στο κέντρο της ακμής αυτής, ο οποίος προκαλεί την αποτυχία του αλγορίθμου.

2.5 Χρονική ανάλυση

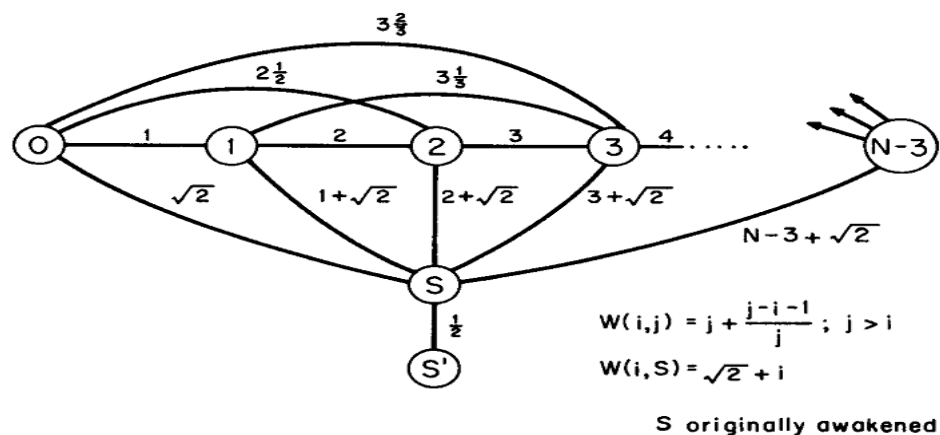


Fig. 2. Example when algorithm requires $N(N - 1)$ message time units.

Αν και εμφανίζεται ότι ο αλγόριθμος τυπικά επιτρέπει σε μεγάλο βαθμό παραλληλισμό των μηνυμάτων, δεν είναι δύσκολο να βρούμε παραδείγματα

όπως για παράδειγμα στην εικόνα 2 στην οποία βλέπουμε πως τα περισσότερα μηνύματα στέλνονται ακολουθιακά. Λεπτομερέστερα, αν ο κόμβος S είναι αρχικά αφυπνισμένος, κάθε κόμβος j ακολουθιακά στέλνει ένα δοκιμαστικό μήνυμα (test message) σε κάθε $i \leq j-2$ και λαμβάνει ένα μήνυμα απόρριψης πριν κάθε κόμβος $j' > j$ αφυπνιστεί. Το γεγονός αυτό οδηγεί σε $N(N-1)$ ακολουθιακά μηνύματα.

Αν ο χρόνος είναι σημαντικός, είναι προτιμότερο όλοι οι κόμβοι να είναι αρχικά αφυπνισμένοι, αυτό μπορεί να πραγματοποιηθεί το πολύ σε $N-1$ χρονικές μονάδες υποθέτοντας πως κάθε μετάδοση μηνυμάτων που απαιτείται πραγματοποιείται το πολύ σε μια χρονική μονάδα. Δείχνουμε ότι, με αυτή την υπόθεση της αρχικής αφύπνισης των κόμβων, ο αλγόριθμος χρειάζεται το πολύ $5N \log_2 N$ χρονικές μονάδες. Σημειώστε πρώτα ότι, μέχρι το χρόνο N , κάθε κόμβος θα είναι αφυπνισμένος και θα έχει στείλει ένα μήνυμα επικοινωνίας. Μέχρι το χρόνο $2N$, κάθε κόμβος πρέπει να στο επίπεδο ένα κατά τη μετάδοση των αρχικών σημάτων.

Παράγουμε από την αναγωγή του επιπέδου των αριθμών, δείχνοντας ότι χρειάζεται το πολύ $5(l+1)N - 3N$ χρονικές μονάδες έως ότου όλοι οι κόμβοι να βρίσκονται στο επίπεδο l . Αυτό είναι αληθές για $l=1$, υποθέτουμε ότι είναι αληθές για l . Στο επίπεδο l , κάθε κόμβος μπορεί να στείλει το πολύ N δοκιμαστικά μηνύματα (test messages) για τα οποία πρέπει να λάβουμε απάντηση πριν από χρόνο $5lN - N$. Η διάδοση της αναφοράς, της αλλαγής ρίζας και της επικοινωνίας, και των μηνυμάτων αρχικοποίησης μπορεί να πάρει περίπου $3N$ μονάδες, έτσι ώστε πριν από χρόνο $5(l+1)N - 3N$ όλοι οι κόμβοι να βρίσκονται στο επίπεδο $l+1$. Στο υψηλότερο επίπεδο, $l \leq \log_2 N$, μόνο δοκιμαστικά μηνύματα, μηνύματα απόρριψης και μηνύματα αναφοράς χρησιμοποιούνται, έτσι ώστε ο αλγόριθμος να έχει ολοκληρωθεί πριν από χρόνο $5N \log_2 N$.

Η χειρότερη περίπτωση $O(n \log N)$ είναι πιθανή, όπως παρουσιάστηκε στο παράδειγμα στην εικόνα 3, όταν η λαβή και η κεφαλή περιέχουν $N/2$ κόμβους. Αν τα βάρη ακμών στη λαβή αυξάνονται καθώς ένας κόμβος απομακρύνεται από την κεφαλή, τότε όλοι οι κόμβοι στη λαβή θα βρίσκονται στο ίδιο fragment του επιπέδου ένα. Αν η επεξεργασία της κεφαλής χρειάζεται $\log N/2$ επίπεδα και κάθε ένα από τα επίπεδα ενός fragment ενώνονται στη λαβή, το οποίο μπορεί να συμβεί, τότε ο χρόνος πολυπλοκότητας θα είναι $O(n \log N)$.

2.6 Παράρτημα

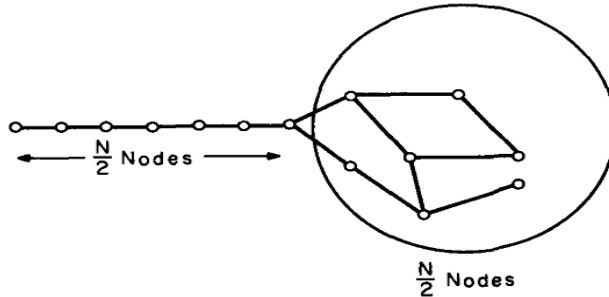


Fig. 3. Example when $O(N \log_2 N)$ message time units are required with initial general awakening.

Στον αλγόριθμο που ακολουθεί κάθε κόμβος υπακούει και αποτελείται από ένα κατάλογο απαντήσεων για κάθε ένα τύπο μηνύματος ο οποίος μπορεί να παραχθεί. Επιπλέον, η απάντηση για μια αυθόρμητη αφύπνιση ενός κόμβου δίνεται. Κάθε κόμβος υποτίθεται ότι περιμένει στη σειρά για τα εισερχόμενα μηνύματα και αποκρίνονται με τη σειρά FIFO δηλαδή αυτό που έρχεται πρώτο εξυπηρετείται και πρώτο. Μια ιδιαίτερη απάντηση είναι να τοποθετηθεί το μήνυμα πίσω στο τέλος της σειράς αναμονής για την καθυστερημένη συντήρηση, αλλά, εκτός από αυτό, κάθε απάντηση ολοκληρώνεται προτού να αρχίσει η επόμενη. Κάθε κόμβος, φυσικά, διατηρεί το δικό του σύνολο μεταβλητών, συμπεριλαμβανομένης και της κατάστασης του (σημειώστε ότι το συμβολίζουμε με SN και υποθέστε πιθανές τιμές: sleeping, Find και Found) και την κατάσταση των παρακείμενων ακμών. Η κατάσταση της ακμής j σημειώνεται συμβολίζεται με $SE(j)$ και οι πιθανές τιμές της είναι: Basic, Branch και Rejected. Είναι πιθανό για την κατάσταση των ακμών των δυο κόμβων οι οποίοι είναι προσκείμενοι στην ίδια ακμή να είναι προσωρινά ασυμβίβαστοι. Αρχικά για κάθε κόμβο έχουμε $SN=Sleeping$ και $SE(j)=Basic$ για κάθε προσκείμενη ακμή j . Κάθε κόμβος επίσης διατηρεί μια μεταβλητή για την ταυτότητα του fragment στο οποίο ανήκει FN και αντίστοιχα για το επίπεδο τη μεταβλητή LN, και τις μεταβλητές best-edge, best-wt, test-edge, in-branch και find-count, όπου οι αρχικές τιμές όλων αυτών είναι ασήμαντες. Υπάρχει επίσης μια αρχική άδεια ουρά first-come first-served για τα εισερχόμενα μηνύματα. Τελικά, το βάρος της κάθε προσκείμενης ακμής j σημειώνεται ως $w(j)$.

Ο αλγόριθμος είναι ο ακόλουθος:

- (1) Response to spontaneous awakening (can occur only at a node in the sleeping state)
execute procedure *wakeup*
- (2) **procedure** *wakeup*
begin let m be adjacent edge of minimum weight;
 $SE(m) \leftarrow Branch$;
 $LN \leftarrow 0$;
 $SN \leftarrow Found$;
 $Find-count \leftarrow 0$;
send *Connect(0)* on edge m
end
- (3) Response to receipt of *Connect(L)* on edge j
begin if $SN = Sleeping$ then **execute procedure** *wakeup*;
if $L < LN$
then begin $SE(j) \leftarrow Branch$;
send *Initiate(LN, FN, SN)* on edge j ;
if $SN = Find$ then
 $find-count \leftarrow find-count + 1$
end
else if $SE(j) = Basic$
then place received message on end of queue
else **send** *Initiate(LN + 1, w(j), Find)* on edge j
end
- (4) Response to receipt of *Initiate(L, F, S)* on edge j
begin $LN \leftarrow L$; $FN \leftarrow F$; $SN \leftarrow S$; $in-branch \leftarrow j$;
 $best-edge \leftarrow nil$; $best-wt \leftarrow \infty$;
for all $i \neq j$ such that $SE(i) = Branch$
do begin **send** *Initiate(L, F, S)* on edge i ;
if $S = Find$ then $find-count \leftarrow find-count + 1$
end;
if $S = Find$ then **execute procedure** *test*
end
- (5) **procedure** *test*
if there are adjacent edges in the state *Basic*
then begin $test-edge \leftarrow$ the minimum-weight adjacent edge in state *Basic*;
send *Test(LN, FN)* on $test-edge$
end
else begin $test-edge \leftarrow nil$; **execute procedure** *report* **end**
- (6) Response to receipt of *Test(L, F)* on edge j
begin if $SN = Sleeping$ then **execute procedure** *wakeup*;
if $L > LN$ then place received message on end of queue
else if $F \neq FN$ then **send** *Accept* on edge j
else begin if $SE(j) = Basic$ then $SE(j) \leftarrow Rejected$;
if $test-edge \neq j$ then **send** *Reject* on edge j
else **execute procedure** *test*
end
end

- (7) Response to receipt of *Accept* on edge j
begin $test-edge \leftarrow nil$;
 if $w(j) < best-wt$
 then begin $best-edge \leftarrow j$; $best-wt \leftarrow w(j)$ **end**;
 execute procedure *report*
end
- (8) Response to receipt of *Reject* on edge j
begin if $SE(j) = Basic$ **then** $SE(j) \leftarrow Rejected$;
 execute procedure *test*
end
- (9) **procedure** *report*
 if $find-count = 0$ **and** $test-edge = nil$
 then begin $SN \leftarrow Found$;
 send *Report*($best-wt$) on *in-branch*
 end
- (10) Response to receipt of *Report*(w) on edge j
 if $j \neq in-branch$
 then begin $find-count \leftarrow find-count - 1$
 if $w < best-wt$ **then begin** $best-wt \leftarrow w$; $best-edge \leftarrow j$ **end**;
 execute procedure *report*
 end
 else if $SN = Find$ **then** place received message on end of queue
 else if $w > best-wt$
 then execute procedure *change-root*
 else if $w = best-wt = \infty$ **then halt**
- (11) **procedure** *change-root*
 if $SE(best-edge) = Branch$
 then send *Change-root* on $best-edge$

 else begin send *Connect*(LN) on $best-edge$;
 $SE(best-edge) \leftarrow Branch$
 end
- (12) Response to receipt of *Change-root*
execute procedure *change-root*

Κεφάλαιο 3°

“Αυτοσταθεροποιούμενο MST μέσα από ένα βέλτιστο βαθμό”

Περίληψη

Προτείνουμε ένα αυτοσταθεροποιούμενο αλγόριθμο για την κατασκευή ενός MDST σε μη κατευθυνόμενα δίκτυα. Ξεκινάμε από μια αυθαίρετη (τυχαία) κατάσταση, και ο αλγόριθμος μας εγγυάται ότι θα συγκλίνει σε “νόμιμη” (επιθυμητή) κατάσταση περιγράφοντας ένα SPT του οποίου ο μέγιστος βαθμός κόμβων είναι το λιγότερο Δ^*+1 όπου Δ^* : είναι ο ελάχιστος πιθανός βαθμός για ένα SPT του δικτύου.

Για την καλύτερη γνώση μας ο αλγόριθμος αυτός αποτελεί την πρώτη λύση που έχει βρεθεί για αυτοσταθεροποιούμενα συστήματα στην κατασκευή MSPT σε μη κατευθυνόμενους γράφους. Ο αλγόριθμος χρησιμοποιεί μόνο τοπικές συνδέσεις (οι κόμβοι αλληλεπιδρούν μόνο με τους γειτονικούς κόμβους με τους οποίους έχουν μόνο ένα κόμβο απόσταση). Επιπλέον, ο αλγόριθμος έχει σχεδιαστεί να δουλεύει σε οποιοδήποτε σύστημα ασύγχρονης επικοινωνίας (ανταλλαγής μηνυμάτων) το οποίο χρησιμοποιεί αξιόπιστα FIFO κανάλια. Επιπρόσθετα, χρησιμοποιούμε ένα άριστο ατομικό μοντέλο (αυτόματο). Η χρονική πολυπλοκότητα της λύσης μας είναι $O(mn^2 \log n)$ όπου m : είναι ο αριθμός των ακμών, και n : είναι ο αριθμός των κόμβων. Η πολυπλοκότητα μνήμης για την αποστολή-λήψη μηνυμάτων του αυτομάτου (πολυπλοκότητα επικοινωνίας), είναι $O(\delta \log n)$, όπου δ : είναι ο μέγιστος βαθμός του δικτύου.

[2]

3.1 Εισαγωγή

Η κατασκευή ενός SP tree αποτελεί θεμελιώδες πρόβλημα στο πεδίο των κατανεμημένων δικτύων αλγορίθμων, όντας η δομική μονάδα για μια ευρεία κατηγορία θεμελιωδών υπηρεσιών όπως: πρωτόγονοι επικοινωνίας (π.χ. broadcast, multicast), ψήφισμα αδιεξόδου (deadlock) ή αμοιβαίο αποκλεισμό (mutual exclusion). Τα νέα προκύπτοντα κατανεμημένα συστήματα όπως τα ειδικά δίκτυα με αισθητήρες (ad-hoc networks) ακμοπυροδοτήτων δικτύων εστιάζουν στο κόστος αποδοτικότητας των κατανεμημένων επικοινωνιακών δομών σε παρόμοια SP trees. Το κύριο ζήτημα στα συστήματα των ειδικών δικτύων για παράδειγμα είναι το κόστος επικοινωνίας. Αν η κάλυψη της επικοινωνίας περιέχει κόμβους με μεγάλο βαθμό τότε ανεπιθύμητα αποτελέσματα μπορούν να συμβούν: συμφόρηση, συγκρούσεις ή έκρηξη

κυκλοφορίας. Παρόλο αυτά, οι κόμβοι αυτοί αποτελούν τον πρώτο στόχο στην ασφάλεια επιθέσεων. Στην περίπτωση αυτή, η κατασκευή ενός αξιόπιστου SP tree απαιτεί να έχει βαθμό κόμβων τον ελάχιστο δυνατό από όσο χρειάζεται.

Κατά ενδιαφέροντα τρόπο, στα ακμοφυροδοτικά δίκτυα το κίνητρο για την κατασκευή MSP trees παρακινείται από την ευημερία των κόμβων (χρηστών). Οποιαδήποτε συμπεριφορά στην επικοινωνία των άλλων κόμβων του δικτύου ελαχιστοποιεί την πιθανότητα ενός κόμβου να χρησιμοποιήσει μόνο όλο το εύρος των δικτύων που τον ενδιαφέρουν. Ως άμεση συνέπεια: οι κόμβοι είναι επιρρεπείς στο να εξαπατήσουν το πραγματικό εύρος ζώνης ή στο να υπάρξει απελευθέρωση στη συμπεριφορά τους και να χρησιμοποιηθεί παράνομα η τοπική κυκλοφορία.

Η αυτοσταθεροποίηση ως υπό -πεδίο της ανοχής σφαλμάτων εισήχθη αρχικά στην κατανομημένη περιοχή από τον Dijkstra το 1974 [18, 19]. Ένας κατανομημένος αλγόριθμος είναι αυτοσταθεροποιούμενος αν, ξεκινώντας από μια αυθαίρετη κατάσταση μας εγγυάται ότι θα συγκλίνει σε μια νόμιμη επιθυμητή κατάσταση, έπειτα από ένα πεπερασμένο αριθμό βημάτων και παραμένει σε ένα “νόμιμο” σύνολο καταστάσεων έπειτα από αυτό.

Μια ευρεία κλάση αυτοσταθεροποιούμενων SP trees έχουν προταθεί έως τώρα: για παράδειγμα BFS δένδρα, υπολογισμός ελάχιστης διαμέτρου SP tree και υπολογισμός ελαχίστου βάρους SP tree. Πρόσφατα προτάθηκαν λύσεις για κατασκευή σταθερού βαθμού SP trees με δυναμικές τοποθετήσεις. Από όσο γνωρίζουμε δεν υπάρχει ακόμη αυτοσταθεροποιούμενος αλγόριθμος για την κατασκευή SP tree.

Αυτή η εργασία εξοπλίζει την αυτοσταθεροποιούμενη κατασκευή MST σε μη κατευθυνόμενους γράφους. Πιο συγκεκριμένα, έστω $G=(V,E)$ ένας γράφος. Αντικείμενο μας είναι να υπολογίσουμε ένα SP tree του γράφου G το οποίο να έχει τον ελάχιστο βαθμό συγκριτικά με τα υπόλοιπα δένδρα του γράφου, όπου ο βαθμός του δένδρου είναι ο μέγιστος βαθμός των κόμβων του. Στην πραγματικότητα, δεδομένου ότι το πρόβλημα μας είναι NP-hard (από επαγωγή του προβλήματος της Hamiltonian διαδρομής), ενδιαφερόμαστε για την κατασκευή ενός Spanning tree του οποίου ο βαθμός είναι μέσα σε έναν από τους βέλτιστους. Αυτή η δέσμευση είναι επιτεύξιμη σε πολυωνυμικό χρόνο, όπως αποδείχθηκε από τους Furer και Raghavachari [20, 21], οι οποίοι περιγράφουν ένα επαυξητικό διαδοχικό αλγόριθμο ο οποίος κατασκευάζει ένα SP tree βαθμού το λιγότερο Δ^*+1 , όπου Δ^* : ο βαθμός ενός MDST. Οι Blin και Butelle [22] πρότειναν μια κατανομημένη εκδοχή του αλγορίθμου των Furer και Raghavachari. Ο αλγόριθμος αυτός έχει τεχνικές ομοιότητες με τον GHS αλγόριθμο ο οποίος αναφέρετε στον έλεγχο και την αναβάθμιση fragment υποδένδρων από τα πρόσφατα κατασκευασμένα SP trees, (παρουσιάστηκε στο προηγούμενο κεφάλαιο). Καμία όμως από τις προηγούμενες λύσεις δεν είναι λύσεις αυτοσταθεροποιούμενων συστημάτων.

Τα αποτελέσματα μας

Περιγράφουμε ένα αυτοσταθεροποιούμενο αλγόριθμο για την κατασκευή ενός MSP tree σε αυθαίρετα δίκτυα. Η συμβολή μας είναι διπλή. Πρώτα από όσο γνωρίζουμε ο αλγόριθμος μας είναι ο πρώτος αυτοσταθεροποιούμενος αλγόριθμος κατά προσέγγιση για την κατασκευή MSP tree σε μη κατευθυνόμενα δίκτυα (μη κατευθυνόμενους γράφους). Ο αλγόριθμος συγκλίνει σε μια “νόμιμη” κατάσταση η οποία περιγράφει ένα SP tree του οποίου ο μέγιστος βαθμός κόμβων είναι το λιγότερο Δ^*+1 , όπου Δ^* : ο μικρότερος πιθανός βαθμός ενός SP tree του δικτύου. Σημειώστε ότι ο υπολογισμός του Δ^* είναι NP-hard πρόβλημα. Ο αλγόριθμος χρησιμοποιεί μόνο τοπικές συνδέσεις – οι κόμβοι αλληλεπιδρούν μόνο με τους γείτονες κόμβους – γεγονός το οποίο είναι βολικό για μεγάλης κλίμακας και εξελικτικά συστήματα. Ο αλγόριθμος έχει σχεδιαστεί να δουλεύει για οποιαδήποτε ασύγχρονη ανταλλαγή μηνυμάτων σε αξιόπιστο FIFO κανάλι. Επιπροσθέτως, χρησιμοποιούμε ένα άριστο ατομικό μοντέλο (αυτόματο) το οποίο ορίζεται για πρώτη φορά στο 21^ο Διεθνές Συμπόσιο Κατανεμημένου Υπολογισμού από τους Burman και Kutten [23].

Δεύτερον, η προσέγγισή μας βασίζεται στην ανίχνευση θεμελιωδών κύκλων (π.χ. κύκλοι εμφανίζονται προσθέτοντας μια ακμή σε ένα δένδρο), αντίθετα με την τεχνική χρήση του αλγορίθμου που παρουσιάστηκε από τους Blin και Butelle, όπου συνεχώς αναβαθμίζονται οι πληροφορίες των χρηστών με σεβασμό στα διαφορετικά fragments (υποδένδρα) τα οποία συνυπάρχουν στο δίκτυο. Ως συνέπεια, και σε αντίθεση με τον αλγόριθμο αυτό, ο αλγόριθμος μας είναι ικανός να μειώσει συγχρονισμένα το βαθμό κάθε κόμβου μεγίστου βαθμού. Η χρονική πολυπλοκότητα της λύσης μας είναι $O(mn^2 \log n)$, όπου m : ο αριθμός των ακμών του δικτύου και n : ο αριθμός των κόμβων του δικτύου. Η πολυπλοκότητα μνήμης είναι $O(\delta \log n)$ σε μια ανταλλαγή μηνυμάτων send/receive ενός ατομικού μοντέλου (αυτομάτου), όπου δ : ο μέγιστος βαθμός του δικτύου. Το μέγιστο μέγεθος των μηνυμάτων που χρησιμοποιούνται απ’ τον αλγόριθμο μας είναι $O(n \log n)$ bits.

3.2 Προκαταρκτικά

3.2.1 Σημειώσεις

Έστω $G=(V,E)$ ένας μη κατευθυνόμενος συνδεδεμένος γράφος, όπου V : είναι το σύνολο των κόμβων και E : είναι το σύνολο των ακμών. Έστω T ένα Spanning tree, το T είναι για τον γράφο G ένας μη κυκλικά συνδεδεμένος υπογράφος του, άρα έχει ίδιο σύνολο κόμβων, επομένως $T=(V,E_T)$ και $E_T \subseteq E$, $|E_T|=|V|-1$. Ο βαθμός ενός κόμβου v του T σημειώνεται ως $\deg_T(v)$ και σημειώνουμε με $\deg(T)$ τον βαθμό του T , για παράδειγμα $\deg(T)=\max_v \deg_T(v)$.

Χρησιμοποιούμε τις ακόλουθες σημειώσεις οι οποίες εισάγονται πρώτα στον αλγόριθμο των Furer και Raghavachari, βελτιώνοντας τις ακμές και φράσσοντας τους κόμβους και ορίζονται ως εξής. Έστω G ένα δίκτυο και T ένα SP tree του G . Έστω $e = \{u, v\}$ μια ακμή του G η οποία δεν περιέχεται στο T . Ο κύκλος C_e δημιουργείται προσθέτοντας την ακμή e στο δένδρο T το οποίο ονομάζεται θεμελιώδες. Ανταλλάσσοντας την ακμή e με οποιαδήποτε άλλη ακμή του κύκλου C_e (ως εκ τούτου μια ακμή μέσα στο SP tree T) έχει αποτέλεσμα σε οποιοδήποτε SP tree T' του G . Έστω w ένας κόμβος του κύκλου C_e , ευδιάκριτο από τους u και v , ο οποίος έχει το μέγιστο βαθμό στο T μεταξύ όλων των άλλων κόμβων του C_e (π.χ. $\deg_T(w) = \deg(T)$). Αν η εναλλαγή μεταξύ της ακμής e και οποιασδήποτε ακμής C_e η οποία είναι εισερχόμενη στον κόμβο w μειώνει κατά ένα το βαθμό του κόμβου w , τότε η ακμή e ονομάζεται βελτιωμένη ακμή. Μια βελτιωμένη ακμή e ικανοποιεί την παρακάτω σχέση:

$$\deg_T(w) \geq \max(\deg(u), \deg(v) + 2) \quad (1)$$

Έστω k ο βαθμός του T (π.χ. $k = \deg(T)$). Αν $\deg_T(u) = k - 1$, τότε ο κόμβος u καλείται φραγμένος κόμβος (blocking) του κύκλου C_e . Αν οι κόμβοι u, v είναι φραγμένοι κόμβοι του C_e , τότε E_q μια κατάσταση όπου η ακμή e δεν είναι βελτιωμένη ακμή του T .

3.2.2 Ένας διαδοχικός αλγόριθμος με την καλύτερη προσέγγιση

Ένα πρόβλημα ενός MSP tree είναι NP-Hard πρόβλημα για να επιλυθεί βέλτιστα. Οι Furer και Raghavachari πρότειναν έναν επαυξητικό διαδοχικό προσεγγιστικό αλγόριθμο ο οποίος κατασκευάζει σε πολυωνυμικό χρόνο ένα SP tree βαθμού λιγότερο από $\Delta^* + 1$, όπου Δ^* : είναι ο ελάχιστος βαθμός του MDSP.

Η κύρια ιδέα του αλγορίθμου τους έχει ως εξής: (i) ξεκινώντας από ένα αυθαίρετο SP tree, (ii) υπολογίζουμε το βαθμό SP tree, (iii) αν υπάρχει βελτιωμένη ακμή e τότε εναλλάσσουμε την e με μια άλλη ακμή του κύκλου C_e η οποία είναι εισερχόμενη, από το προηγούμενο βήμα, σε ένα κόμβο ο οποίος έχει το μέγιστο βαθμό. Διαφορετικά, για να λάβουμε μια επαυξητική ακμή προσπαθούμε να φτιάξουμε μη φραγμένους κόμβους σε ένα θεμελιώδη κύκλο ο οποίος περιέχει μέγιστο βαθμό κόμβων (το γεγονός αυτό μπορεί να οδηγήσει σε ακολουθία φραγμένων κόμβων μειωμένου βαθμού). Αν μια βελτιωμένη ακμή εμφανίζεται, τότε επιστρέφουμε στο βήμα (iii), διαφορετικά δεν υπάρχει πιθανότητα βελτίωσης και έτσι επιστρέφουμε στο τρέχον δένδρο.

3.2.3 Πληροφορίες που απαιτούνται στο αποκεντρωτικό περιβάλλον

Ο διαδοχικός αλγόριθμος ο οποίος παρουσιάστηκε από τους Furer και Raghavachari απαιτεί τον υπολογισμό του κόμβου με το μεγαλύτερο βαθμό για το τρέχον SP tree κάθε φορά. Ο υπολογισμός της πληροφορίας αυτής με έναν αποκεντρωτικό τρόπο απαιτεί διάχυση στο δίκτυο. Κάποιος μπορεί να ρωτήσει αν είναι πιθανό να υπολογίσουμε σε πολυωνυμικό χρόνο μια λύση με τη ίδια προσέγγιση αλλά χωρίς την πληροφορία αυτή. Έως ότου ο αλγόριθμος λειτουργεί για θεμελιώδεις κύκλους, ένας φυσικός τρόπος να μειώσουμε το κόστος επικοινωνίας είναι να χρησιμοποιήσουμε τους κόμβους με το μέγιστο βαθμό ενός θεμελιώδη κύκλου ή τους κόμβους με το μεγαλύτερο βαθμό ενός δένδρου το οποίο έχει βελτιωμένες ακμές και ορίζονται φραγμένοι κόμβοι. Δυστυχώς, βλέπουμε για κάθε αλγόριθμο ο οποίος βασίζεται στην εναλλαγή ακμών θεμελιωδών κύκλων όπου αν ο κόμβος με το μεγαλύτερο βαθμό του δένδρου δεν χρησιμοποιείται για να λάβουμε μια λύση με μέγιστο βαθμό μεγαλύτερο του $\Delta^* + 1$.

Δίνουμε αριθμητικά παραδείγματα τα οποία δείχνουν ότι όλες οι εναλλαγές δεν είναι ισοδύναμες. Θεωρούμε το παράδειγμα της εικόνας 1(a). Λαμβάνοντας τους κόμβους u_i και v_i φραγμένοι κόμβοι ενός θεμελιώδη κύκλου $\{u_i, v_i\}$. Για να μειώσουμε το βαθμό ενός κόμβου x , θα χρειαστούμε το λιγότερο ένα ζευγάρι κόμβων u_i, v_i το οποίο θα πρέπει να μεταβεί γρήγορα σε μια μη φραγμένη κατάσταση. Οι ακμές b_i, c_i είναι αντίστοιχα βελτιωμένες ακμές των C_{b_i} και C_{c_i} κύκλων αντίστοιχα. Υποθέστε ακμές c_i οι οποίες χρησιμοποιούνται για εκτέλεση βελτίωσης. Το γεγονός αυτό κάνει μόνο τον u_i μη φραγμένο κόμβο και περισσότερη βελτίωση δεν είναι πιθανή εξαιτίας των ακμών b_i και a_i οι οποίες περιέχουν ένα φραγμένο κόμβο, όπως φαίνεται στην εικόνα 1(c). Από την άλλη πλευρά η ακμή c_i αντικαθίσταται από την b_i , αυτό οδηγεί τους κόμβους u_i και v_i σε μη φραγμένους και έπειτα μια πρόσθετη βελτίωση μειώνει το βαθμό του κόμβου x που φαίνεται στην εικόνα 1(c). Ο βαθμός του x θα είναι ένας από τους βέλτιστους όπως βλέπουμε στην εικόνα 1(d).

Επομένως οποιοσδήποτε αλγόριθμος βασίζεται σε εναλλαγές θεμελιωδών κύκλων χρειάζεται να χρησιμοποιεί τους κόμβους με το μέγιστο βαθμό του τρέχον δένδρου για να είναι σε θέση να επιστρέψουν μια λύση με μέγιστο βαθμό το λιγότερο $\Delta^* + 1$ από το MDST πρόβλημα. Διαφορετικά, όπως εξηγήσαμε ο αλγόριθμος μπορεί να λάβει κάποιες καλές τοπικές αποφάσεις οι οποίες να μην είναι καλές για τον αλγόριθμο γενικότερα.

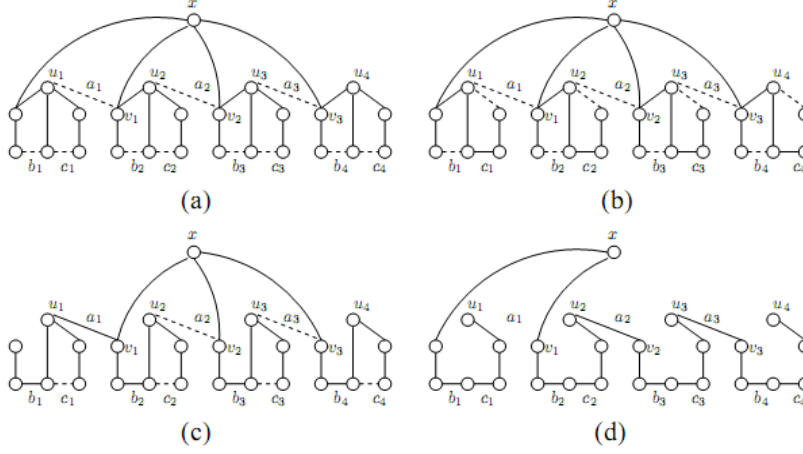


Figure 1: All swaps are not equivalent.

3.3 Μοντέλο

Δανειζόμαστε το μοντέλο το οποίο προτάθηκε στο 21^ο Διεθνές Συμπόσιο Κατανεμημένου Υπολογισμού από τους Burman και Kutten. Θεωρούμε ένα μη κατευθυνόμενο συνδεδεμένο δίκτυο $G=(V,E)$, όπου V : είναι το σύνολο των κόμβων και E : είναι το σύνολο των ακμών. Οι κόμβοι αντιπροσωπεύουν τους επεξεργαστές και οι ακμές τις συνδετικές γραμμές (συνδέσμους). Κάθε κόμβος του δικτύου έχει μια μοναδική ταυτότητα (αναγνωριστικό). Για κάθε κόμβο $v \in V$, σημειώνουμε το σύνολο των γειτονικών του κόμβων $N(v) = \{u, \{u,v\} \in E\}$ (όπου $\{u,v\}$ σημειώνεται η ακμή μεταξύ του κόμβου u και του γειτονικού του v). Το μέγεθος του συνόλου $N(v)$ είναι ο βαθμός του v .

Θεωρούμε μια στατική τοπολογία. Το επικοινωνιακό μοντέλο είναι ασύγχρονη ανταλλαγή μηνυμάτων με κανάλια FIFO (σε κάθε ακμή τα μηνύματα παραδίδονται με την ίδια σειρά με την οποία στάλθηκαν). Χρησιμοποιούμε έναν καθορισμό του μοντέλου: το ατομικό send/receive (αυτόματο). Στο μοντέλο αυτό κάθε κόμβος αναφέρεται σε ένα τοπικό αντίγραφο των μεταβλητών των γειτόνων του, το οποίο ανανεώνεται μέσω ειδικών μηνυμάτων (αναφέρεται στη συνέχεια ως (InfoMsg) που εναλλάσσονται περιοδικά από τους γείτονες κόμβους. Υποθέτουμε αξιόπιστα κανάλια, ούτως ώστε κανένα μήνυμα να μη χάνεται ή να μη παραδίδεται αλλοιωμένο. Μια τοπική κατάσταση ενός κόμβου είναι η αξία των τοπικών μεταβλητών του κόμβου, το αντίγραφο των τοπικών μεταβλητών των γειτονικών του κόμβων και η κατάσταση του μετρητή προγράμματος (program counter). Μια διαμόρφωση του συστήματος είναι το διαγώνιο προϊόν τοπικών καταστάσεων όλων των κόμβων στο σύστημα μαζί με το περιεχόμενο των συνδέσεων. Ένα ατομικό βήμα σε ένα κόμβο p είναι ένας εσωτερικός υπολογισμός ο οποίος βασίζεται στην τρέχουσα τιμή της τοπικής κατάστασης του p και σε μια ενιαία λειτουργία

επικοινωνίας (send/receive) στον κόμβο p . Ο υπολογισμός του συστήματος είναι μια άπειρη ακολουθία διαμορφώσεων, $e = (c_0, c_1, \dots, c_i, \dots)$ όπου κάθε διαμόρφωση c_{i+1} προκύπτει από την c_i μέσω εκτέλεσης ενός μόνο ατομικού βήματος.

Ελαττώματα & Αυτοσταθεροποίηση

Σαν συνέπεια θεωρούμε ότι το σύστημα μπορεί να ξεκινήσει με οποιαδήποτε διαμόρφωση. Αυτό είναι η τοπική κατάσταση του κόμβου η οποία μπορεί να αλλοιωθεί. Σημειώστε ότι δεν μπορούμε να κάνουμε οποιαδήποτε υπόθεση στο φράγμα των αλλοιωμένων κόμβων. Στη χειρότερη περίπτωση όλοι οι κόμβοι του συστήματος πρέπει να ξεκινήσουν από μια αλλοιωμένη διαμόρφωση. Για να εξοπλίσουμε τέτοιου είδους σφάλματα χρησιμοποιούμε τεχνικές αυτοσταθεροποίησης.

Ορισμός 1 (Αυτοσταθεροποίηση)

Έστω L_A ένα νόμιμο κατηγορήμα ενός αλγορίθμου A , όσον αφορά μια προδιαγραφή κατηγορήματος προδιαγραφών έτσι ώστε κάθε διαμόρφωση του L_A να ικανοποιεί τις προδιαγραφές. Ο αλγόριθμος A είναι αυτοσταθεροποιούμενος όσον αφορά την προδιαγραφή αν πληρούνται οι ακόλουθοι όροι: (i) κάθε υπολογισμός μιας έναρξης διαμόρφωσης ικανοποιεί ότι το L_A συντηρεί το L_A (περάτωση). (ii) Κάθε υπολογισμός του A ξεκινά από μια αυθαίρετη διαμόρφωση η οποία περιέχει μια διαμόρφωση που ικανοποιεί το L_A (σύγκλιση). ■

Minimum Degree Spannic Tree (MDST)

Μια νόμιμη διαμόρφωση για το MDST είναι η διαμόρφωση η οποία εξάγει ένα μοναδικό SP tree ελαχίστου βαθμού. Έως ότου υπολογίζοντας το Δ^* ως ελάχιστο βαθμό σε ένα SP tree σε δοσμένο NP-Hard δίκτυο, προτείνουμε στο ακόλουθο αυτοσταθεροποιούμενο MDST με μέγιστο βαθμό κόμβων το λιγότερο $\Delta^* + 1$.

3.4 Ο Self-Stabilizing MDST αλγόριθμος

Τα κύρια συστατικά που χρησιμοποιούνται από τον MDST προσεγγιστικά είναι: (1) μια ενότητα που περιέχει ένα SP tree, (2) μια ενότητα για τον υπολογισμό του κόμβου με το μέγιστο βαθμό του SP tree, (3) μια ενότητα για τον υπολογισμό θεμελιωδών κύκλων και (4) μια ενότητα για την μείωση του βαθμού του κόμβου, με το μέγιστο βαθμό του δένδρου βασισμένο

πάνω στους θεμελιώδεις κύκλους οι οποίοι υπολογίστηκαν σε προηγούμενες ενότητες.

Μια πρόκληση είναι να σχεδιάσουμε και να τρέξουμε τέτοιες ενότητες ταυτόχρονα, με ένα αυτοσταθεροποιούμενο τρόπο. Σημειώστε ότι ο πυρήνας του αλγορίθμου μας αποτελεί τη διαδικασία μείωσης η οποία στοχεύει επαναληπτική μείωση του βαθμού του SP tree έως ότου λάβουμε ένα SP tree βαθμού το λιγότερο $\Delta^* + 1$.

Το ακόλουθο τμήμα προτείνει μια αναλυτική περιγραφή του αλγορίθμου. Η επίσημη περιγραφή του αλγορίθμου φαίνεται στην εικόνα 3, και υλοποιείται με τις υπορουτίνες που φαίνονται στις εικόνες 2 και 6 πιο κάτω.

3.4.1 Δομές Δεδομένων και Στοιχειώδεις διαδικασίες

Μεταβλητές. Αυτό το μικρό τμήμα κατηγοριοποιεί όλες τις μεταβλητές του αλγορίθμου. Για κάθε κόμβο $v \in V(G)$, $N(v)$ (υποθέτουμε κρυφό αυτοσταθεροποιούμενο πρωτόκολλο το οποίο τακτικά αναβαθμίζει – ανανεώνει το σύνολο των γειτόνων) σημειώστε το σύνολο από όλους τους γειτονικούς κόμβους του v στο δίκτυο G , $ID_v \in V$ είναι το μοναδικό αναγνωριστικό (ταυτότητα) του v και deg_v είναι ο βαθμός του v στο δένδρο (π.χ. $deg_v \equiv |\{u : u \in N(v) \wedge is_tree_edge(u, v)\}|$). Οι κόμβοι επαναληπτικά στέλνουν τις μεταβλητές τους σε κάθε έναν από τους γείτονες τους στέλνοντας InfoMsg και αναβαθμίζοντας τις τοπικές μεταβλητές επάνω στην υποδοχή αυτού του τύπου μηνύματος από έναν γείτονα. Κάθε κόμβος v διατηρεί τις ακόλουθες μεταβλητές:

Μεταβλητές τύπου Integer:

- $root_v$: Είναι το ID της ρίζας του Spanning tree.
- $parent_v$: Το ID του πατέρα του κόμβου v .
- $distance_v$: Η απόσταση του κόμβου v από τη ρίζα .
- $dmax_v$: Η τοπική εκτίμηση του k , ανανεώνεται κάθε φορά που λαμβάνουμε ένα InfoMsg. Μια αλλαγή του k ανιχνεύεται με μια $color_tree_v$ μεταβλητή. Η πληροφορία αυτή αφορά περισσότερο το δίκτυο InfoMsg.
- $dmax_t_v$: Τοπική εκτίμηση του μεγίστου βαθμού σε ένα υποδένδρο με ρίζα τον κόμβο v , ανανεώνεται κάθε φορά που λαμβάνει InfoMsg.

Μεταβλητές τύπου Boolean:

- $edge_status_v[u]$: Είναι true όταν η $\{u, v\}$ είναι ακμή του δένδρου.
- $color_tree_v$: Χρησιμοποιείται για τη διαδρομή οποιασδήποτε αλλαγής στο $dmax_v$.

Μηνύματα. Τα μηνύματα χρησιμοποιούνται από τον αλγόριθμο ως ακολούθως:

- $\langle InfoMsg, root_v, parent_v, distance_v, dmax_v, dmax_t_v, deg_v, edge_status_v[u], color_tree_v \rangle$: αυτό το μήνυμα χρησιμοποιείται για να ενημερώσει τις τοπικές μεταβλητές του κόμβου.
- $\langle search, init_edge, idblock, path \rangle$: Χρησιμοποιείται για να βρει θεμελιώδη κύκλους.
- $\langle Remove, deg_max, target, path \rangle$: Χρησιμοποιείται για να μειώσει το βαθμό του κόμβου με το μεγαλύτερο βαθμό διαγράφοντας την προσπίπτουσα ακμή η οποία έπειτα αλλάζει προσανατολισμό άκρων στο θεμελιώδη κύκλο, όταν η $init_edge$ είναι η αρχική ακμή η οποία δεν ανήκει στο δένδρο, στόχος είναι η ακμή αυτή να διαγραφεί, όπου deg_max είναι ο μέγιστος βαθμός του στόχου των άκρων και μονοπατιών στο θεμελιώδη κύκλο.
- $\langle Back, init_edge, target, path \rangle$: Αλλάζει προσανατολισμό άκρων στο θεμελιώδη κύκλο μετά από μια ανάπτυξη όταν η $init_edge$ είναι η αρχική ακμή που δεν ανήκει στο δένδρο, στόχος είναι να διαγραφεί η ακμή και το μονοπάτι του θεμελιώδη κύκλου.
- $\langle Deblock, idblock \rangle$: Χρησιμοποιείται για να αλλάξει την κατάσταση ενός μπλοκαρισμένου κόμβου, όταν το $idblock$ είναι το id ενός μπλοκαρισμένου κόμβου.
- $\langle Reverse, target, path \rangle$: Χρησιμοποιείται για να σβήσει τις τροποποιήσεις που γίνονται στο θεμελιώδη κύκλο εξαιτίας της ταυτόχρονης βελτίωσης, όταν ο στόχος είναι η διαγεγραμμένη ακμή.
- $\langle UpdateDist, dist \rangle$: Ενημερώνει την απόσταση θεμελιωδών κύκλων των κόμβων, όταν η $dist$ είναι η απόσταση από τη ρίζα του δένδρου στον αποστολέα.

Σημειώστε ότι το μονοπάτι πληροφοριών ποτέ δεν αποθηκεύεται σε ένα μόνο κόμβο, παρόλο αυτά δεν ταξινομείται ως τοπική μεταβλητή ενός ιδιαίτερου κόμβου. Εντούτοις αυτή η πληροφορία μεταφέρει διαφορετικά μηνύματα. Παρόλο αυτά η πολυπλοκότητα της λύσης μας στο μήκος του αποθηκευτικού χώρου του δικτύου είναι το λιγότερο $O(n \log n)$ (το μέγιστο μήκος του path chain) όπως αναλύεται στο τμήμα της πολυπλοκότητας.

3.4.2 Στοιχειώδεις Δομικές Μονάδες

Σε αυτό το τμήμα, παρέχουμε πρώτα μια λεπτομερή περιγραφή των θεμελιωδών μονάδων για τον αλγόριθμο μείωσης του βαθμού μας, δηλαδή μια ενότητα αναφέρεται σε ένα spanning tree που ανιχνεύει τους θεμελιώδεις κύκλους. Ολοκληρώνουμε το τμήμα με τη λεπτομερή παρουσίαση της ενότητας μείωσης βαθμού.

3.4.2.1 Ενότητα Spanning Tree

Ο αλγόριθμος που περιγράφεται παρακάτω είναι μια απλοποίηση του αλγορίθμου BFS. Κάθε κόμβος ν αναφέρεται σε τρεις μεταβλητές: στο τοπικό γνωστό ID της ρίζας του δένδρου, σε ένα δείκτη του γονιού του ν (parent) και της απόστασης του ν από τη ρίζα του spanning tree. Η έξοδος του δένδρου δρομολογείται στον κόμβο με την ελάχιστη τιμή.

Ο αλγόριθμος χρησιμοποιεί τυπικά δυο κανόνες οι οποίοι διευκρινίζονται πιο κάτω. Ο πρώτος κανόνας “διόρθωση γονιού”, επιτρέπει την αναβάθμιση της τοπικής γνωστής ρίζας: Για ένα κόμβο ν αν ένας γειτονικός u έχει μικρότερο ID από τον ν τότε ρίζα του δένδρου γίνεται ο κόμβος u, και μάλιστα ο u γίνεται γονιός του κόμβου ν. Αν αρκετοί γειτονικοί κόμβοι ελέγξουν αυτή την ιδιοκτησία τότε ο u θα επιλεγεί ως πατέρας του κόμβου με το μικρότερο ID. Αυτή η επιλογή πραγματοποιείται από τη λειτουργία argmin. Ο δεύτερος κανόνας, “διόρθωση ρίζας”, δημιουργεί μια ρίζα αν ο γειτονικός μας κόμβος βρίσκεται σε ασυνάρτητη (μη αποδεκτή) κατάσταση. Σε μια ασυνάρτητη κατάσταση, η απόσταση του BFS δένδρου με σεβασμό στη ρίζα θα γίνει η απόσταση του γονιού της συν ένα. Στην περίπτωση όπου η απόσταση αυτή της ρίζας θα είναι μηδενική. Επιπροσθέτως, ο γονιός του κόμβου θα μπορούσε να είναι γείτονας.

Δίνουμε πιο κάτω τα κατηγορήματα, τις διαδικασίες και τους κανόνες που χρησιμοποιούνται για τη μέτρηση και την κατασκευή των spanning trees.

Κατηγορήματα. Τα κατηγορήματα σε ένα Spanning tree ενός τυχαίου κόμβου ν είναι τα ακόλουθα:

- $better_parent(v) \equiv V_{u \in N(v)} root_v > root_u$
- $coherent_parent(v) \equiv (parent_v \in N(v) \cup \{v\}) \wedge (root_v = root_{parent_v})$
- $coherent_distance(v) \equiv (parent \neq ID_v \wedge distance_v = distance_{parent_v} + 1) \vee (parent_v = ID_v \wedge distance_v = 0)$
- $new_root_vandidate(v) \equiv \neg coherent_parent(v) \vee \neg coherent_distance(v)$
- $is_tree_edge(v, u) \equiv parent_v = ID_u \vee parent_u = ID_v$

Διαδικασίες. Οι διαδικασίες ενός Spanning tree είναι:

- $change_parent_to(v, u) \equiv root_v \leftarrow root_u; parent_v \leftarrow ID_u;$
 $distance_v \leftarrow distance_u + 1$
- $create_new_root(v) \equiv root_v \leftarrow ID_v; parent_v \leftarrow ID_v; distance_v \leftarrow 0$

Κανόνες. Οι κανόνες σε ένα Spanning tree για ένα τυχαίο κόμβου v είναι οι ακόλουθοι:

RCorrect_p: (correction parent)

If $\neg \text{new_root_candidate}(v) \wedge \text{better_parent}(v)$ **then**
 $\text{change_parent_to}(v, \arg \max_{u \in N(v)} \text{root}_u);$

RCorrect_r: (correction root)

If $\text{new_root_candidate}(v)$ **then** $\text{create_new_root}(v)$

3.4.2.2. Ενότητα Minimum Degree

Η κατασκευή μας για τη δημιουργία ενός MSP tree ελαχίστου βαθμού από ένα τυχαίο Spanning tree απαιτεί για κάθε κόμβο v την ακόλουθη γνώση: ποιες από τις παρακείμενες ακμές ανήκουν στο δένδρο και το μέγιστο βαθμό του τρέχοντος Spanning tree.

Σημειώστε ότι και οι δυο πληροφορίες πρέπει να υπολογισθούν χρησιμοποιώντας μόνο τοπικές συνδέσεις όπως περιγράφεται στη συνέχεια. Η συνεπής συνάρτηση των ακμών του τμήματος του spanning tree πραγματοποιείται μέσω της μεταβλητής $\text{edge_status}_v[u]$, ενημερώνεται μέσω των InfoMsg μηνυμάτων. Αυτή η διαδικασία είναι παρόμοια με την αναφορά στα δυναμικά δένδρα, όπως αναφέρεται από τους Bianchi, Datta και Felber στην εργασία τους “Stabilizing peer-to-peer spatial filters”. Το πρωτόκολλο μας χρησιμοποιεί διάδοση της πληροφορίας με στρατηγική ανατροφοδότησης (PIF), (για περισσότερες λεπτομέρειες δες τις εργασίες των Blin, Cournier και Villain με τίτλο “An improved snap-stabilizing pif algorithm” καθώς και των Cournier, Datta, Petit και Villain με τίτλο “Optimal snap-stabilizing pif algorithms in un-oriented trees”). Με σκοπό να προτείνουμε μια αυτοπεριεχόμενη λύση, δίνουμε τον ακόλουθο τρόπο για υπολογισμό και για τη διάδοση του κόμβου με το μεγαλύτερο βαθμό του τρέχοντος spanning tree. Οι προτεινόμενοι κανόνες χρησιμοποιούν δυο μεταβλητές για τον κόμβο v : τις $d_{\max_t_v}$ και $d_{\max_v}$. Η πρώτη χρησιμοποιείται για να ενημερώσει τον πατέρα του κόμβου v , ο οποίος είναι ο κόμβος με το μεγαλύτερο βαθμό, στο υποδένδρο με ρίζα τον κόμβο v . Ο τελευταίος χρησιμοποιείται για να διαδώσει στο δένδρο το σωστό μέγιστο βαθμό του τρέχοντος spanning tree. Επιπλέον οι κανόνες οι οποίοι περιγράφηκαν πιο πριν χρησιμοποιούν το κατηγορημα $\text{local_max_degree}(v)$. Αυτό το κατηγορημα επιστρέφει το βαθμό του δένδρου για ένα κόμβο φύλλων ή τη μέγιστη τιμή μεταξύ του βαθμού του κόμβου v στο δένδρο και το μέγιστο βαθμό όπως υπολογίστηκε στο υποδένδρο του v (μέγιστη τιμή μεταξύ των κόμβων v και u της μεταβλητής $d_{\max_t_v}$). Για να υπολογίσουμε το μέγιστο βαθμό του τρέχοντος spanning tree, ενός κόμβου v πρέπει να σταθεροποιήσουμε πρώτα τη μεταβλητή $d_{\max_t_v}$. Αν η μεταβλητή

$d_{\max_t_v}$ έχει διαφορετική τιμή από αυτή που δόθηκε από το κατηγορήμα $local_max_degree(v)$ τότε ο κόμβος u χρησιμοποιεί τον κανόνα $RMax_dFeedB$ για να διαδώσει την τιμή της μεταβλητής $d_{\max_t_v}$. Όταν η μεταβλητή $d_{\max_t_v}$ σταθεροποιηθεί σε ένα κόμβο v , τότε ο κόμβος αυτός μπορεί να εκτελέσει τους κανόνες $RMax_dRoot$ και $RMax_dPro$ για να μειώσει το μέγιστο βαθμό του δένδρου ανανεώνοντας την τιμή των μεταβλητών $d_{\max}$. Ο κανόνας $RMax_dRoot$ χρησιμοποιείται από τη ρίζα του δένδρου για να ανανεώνει τις τιμές των $d_{\max}$ μεταβλητών και να ενημερώνει τα παιδιά της για το μέγιστο βαθμό του δένδρου. Αυτός ο κανόνας χρησιμοποιείται μόνο στην περίπτωση όπου $d_{\max_v} \neq d_{\max_t_v}$. Οποιοσδήποτε άλλος κόμβος του δένδρου χρησιμοποιεί τον κανόνα $RMax_dPro$ για να ενημερώσει την τιμή των μεταβλητών $d_{\max}$ και να ενημερώσει τα παιδιά του για το μέγιστο βαθμό του δένδρου. Ο κανόνας αυτός χρησιμοποιείται μόνο στην περίπτωση όπου ο κόμβος v έχει διαφορετική εκτίμηση του μεγίστου βαθμού με το γονιό του, π.χ. όταν έχουμε $d_{\max_v} \neq d_{\max_{parent_v}}$.

Κατηγορήματα. Τα κατηγορήματα της ενότητας με το μέγιστο βαθμό κόμβων για ένα τυχαίο κόμβο v , είναι τα ακόλουθα:

- $tree_stabilized(v) \equiv \neg better_parent(v) \wedge \neg new_root_candidate(v)$
- $local_max_degree(v) \equiv \max(deg_v, \max\{d_{\max_t_v} : u \in N(v) \wedge parent_u = v\})$
- $degree_stabilized(v) \equiv \bigwedge_{u \in N(v)} d_{\max_v} = d_{\max_u}$
- $color_stabilized(v) \equiv \bigwedge_{u \in N(v)} color_tree_v = color_tree_u$
- $locally_stabilized(v) \equiv tree_stabilized(v) \wedge color_stabilized(v)$

Κανόνες. Οι κανόνες της ενότητας με το μέγιστο βαθμό κόμβων για ένα τυχαίο κόμβο v , είναι οι ακόλουθοι:

$RMax_dFeedB$: (max degree feedback)

If $tree_stabilized(v) \wedge d_{\max_t_v} \neq local_max_degree(v)$
then $d_{\max_t_v} \leftarrow local_max_degree(v);$

$RMax_dRoot$: (max degree of root)

If $tree_stabilized(v) \wedge parent_v = root_v \wedge d_{\max_t_v} = local_max_degree(v) \wedge d_{\max_v} \neq d_{\max_t_v}$ **then** $d_{\max_v} \leftarrow d_{\max_t_v}$

$RMax_dPro$: (max degree propagation)

If $tree_stabilized(v) \wedge parent_v \neq root_v \wedge d_{\max_t} = local_max_degree(v) \wedge d_{\max_v} \neq d_{\max_{parent_v}}$ **then** $d_{\max_v} \leftarrow d_{\max_{parent_v}}$

Σημειώστε ότι ο μέγιστος βαθμός του δένδρου αλλάζει κατά τη διάρκεια της εκτέλεσης του αλγορίθμου κατά τη διαδικασία μείωσης βαθμού. Όταν ένας κόμβος με μέγιστο βαθμό έχει μειώσει το βαθμό του κατά ένα, ο αλγόριθμος χρησιμοποιεί τη μεταβλητή $color_tree_v$ για να εντοπίσει την αλλαγή αυτή.

Επιπροσθέτως ένας κόμβος ο οποίος εντοπίζει μια ασυναρτησία στους γειτονικούς κόμβους η οποία σχετίζεται με μέγιστο βαθμό που εμποδίζει την κατασκευή του δένδρου με τον ελάχιστο βαθμό SP tree έως ότου οι γείτονες γίνουν *locally_stabilized* (όλοι οι γείτονες έχουν την ίδια τιμή για τη μεταβλητή τους “*color_tree_v*”).

3.4.2.3 Θεμελιώδης Ανίχνευση Κύκλων

Κάνουμε ανάκληση στο δοσμένο δίκτυο G , όπου T είναι ένα Spanning tree του G και $e = \{u, v\}$ μια ακμή του G η οποία δεν ανήκει στο T , ο κύκλος C_e , δημιουργείται προσθέτοντας την ακμή e στο T το οποίο ονομάζεται θεμελιώδες.

Υπολογίζοντας ένα θεμελιώδη κύκλο για ένα αυθαίρετο κόμβο v εκτελείται από τη διαδικασία $Cycle_Search(v)$. Για κάθε ακμή $\{u, v\}$ η οποία δεν ανήκει στο δένδρο όπου $ID_v < ID_u$ (v είναι ο ιδρυτής της ακμής $\{u, v\}$), ο κόμβος v αρχικοποιεί ένα DFS στο τρέχον Spanning tree T για να ανακαλύψει το μονοπάτι μέσα στο δένδρο μεταξύ των κόμβων u και v . Αυτό το DFS χρησιμοποιεί μόνο μηνύματα τύπου *Search*. Η DFS διάδοση ανακαλύπτει μόνο τις ακμές του δένδρου των θεμελιωδών κύκλων $\{u, v\}$. Ανανεώνει το μονοπάτι επισυνάπτοντας τον κόμβο που ανακαλύφθηκε και τον βαθμό του δένδρου.

Η DFS αναζήτηση διάδοσης τερματίζει όταν το άλλο άκρο της ακμής (π.χ. u) επιτυγχάνεται και το μονοπάτι περιέχει μόνο τις ακμές του δένδρου των θεμελιωδών κύκλων των $\{u, v\}$.

```

1 Cycle_Search(id)
2 if locally_stabilized(v) then
3    $\forall u \in N(v)$  with  $edge\_status_v[u] = \text{false}$  and  $ID_v < ID_u$ : initiate a DFS propagation of a Search message crossing
   only tree edges by sending  $(\text{Search}, (u, v), id, \{\{v, deg_v\}\})$  to  $w \in N(v)$  s.t.  $edge\_status_v[w] = \text{true}$ 
4 Upon receipt  $(\text{Search}, (x, y), id, path) \wedge locally\_stabilized(v)$  from  $u$ :
5 if  $v \neq x$  then
6   DFS propagation on tree edges of  $(\text{Search}, (x, y), id, path \oplus \{\{v, deg_v\}\})$ 
7 else
8   if  $\neg edge\_status_v[y]$  then
9     Action_on_Cycle(id, y, path, u);

```

Figure 2: Fundamental cycle detection at node v

```

1 Do forever: send  $\text{InfoMsg}_v$  to all  $u \in N(v)$ 
2 Upon receipt of  $\text{InfoMsg}_u$  from  $u$ :  $\text{Update\_State}(u)$ ;  $\text{Cycle\_Search}(\text{NIL})$ ;
3 Upon receipt of  $\langle \text{Remove}, (x, y), \text{deg\_max}, (w, z), \text{path} \rangle \wedge \text{locally\_stabilized}(v)$  from  $u$ :
4 if  $\text{target\_remove}((w, z))$  then
5   if  $\text{remove\_ok}(\text{deg\_max}, (w, z))$  then  $\text{Reverse\_Orientation}((x, y), \text{deg\_max}, (w, z), u, \text{path})$ ;
6 else
7   if  $\text{source\_remove}(v, (x, y))$  then let  $v = x$ .  $\text{parent}_v \leftarrow y$ ;  $\text{distance}_v \leftarrow \text{distance}_{\text{parent}_v} + 1$ ;
8   else let  $\text{path} = \text{list}_1 \oplus v \oplus \text{list}_2$  and  $p = \text{head}(\text{reverse}(\text{list}_1))$ .
9     if  $w, z \notin \text{list}_1$  then
10       if  $\neg \text{edge\_status}_v[p]$  then  $\text{Reverse\_Aux}((w, z), \text{path})$ ;
11       else  $\text{parent}_v \leftarrow p$ ;  $\text{distance}_v \leftarrow \text{distance}_{\text{parent}_v} + 1$ ;
12       if  $\text{edge\_status}_v[p]$  then send  $\langle \text{Remove}, (x, y), \text{deg\_max}, (w, z), \text{path} \rangle$  to  $p$ ;
13  $\forall q \in N(v), q \neq \text{parent}_v \wedge \text{edge\_status}_v[q]$ , send  $\langle \text{UpdateDist}, \text{distance}_{\text{parent}_v} + 1 \rangle$  to  $q$ ;
14 send  $\text{InfoMsg}_v$  to all  $q \in N(v)$ 
15 Upon receipt of  $\langle \text{Back}, (x, y), (w, z), \text{path} \rangle \wedge \text{locally\_stabilized}(v)$  from  $u$ :
16 if  $\text{source\_remove}(v, (x, y))$  then let  $v = x$ .  $\text{parent}_v \leftarrow y$ ;  $\text{distance}_v \leftarrow \text{distance}_{\text{parent}_v} + 1$ ;
17 else let  $\text{path} = \text{list}_1 \oplus v \oplus \text{list}_2$  and  $p = \text{head}(\text{list}_2)$ .
18   if  $\neg \text{edge\_status}_v[p]$  then  $\text{Reverse\_Aux}((w, z), \text{reverse}(\text{path}))$ ;
19   else  $\text{parent}_v \leftarrow p$ ;  $\text{distance}_v \leftarrow \text{distance}_p + 1$ ; send  $\langle \text{Back}, (x, y), (w, z), \text{path} \rangle$  to  $p$ ;
20  $\forall q \in N(v), q \neq \text{parent}_v \wedge \text{edge\_status}_v[q]$ , send  $\langle \text{UpdateDist}, \text{distance}_{\text{parent}_v} + 1 \rangle$  to  $q$ ;
21 send  $\text{InfoMsg}_v$  to all  $q \in N(v)$ 
22 Upon receipt of  $\langle \text{Deblock}, \text{idblock} \rangle \wedge \text{locally\_stabilized}(v)$  from  $u$ :  $\text{Broadcast}(\text{idblock}, u)$ ;
23 Upon receipt of  $\langle \text{Reverse}, (w, z), \text{path} \rangle \wedge \text{locally\_stabilized}(v)$ :
24 let  $\text{path} = \text{list}_1 \oplus v \oplus \text{list}_2$  and  $p = \text{head}(\text{list}_2)$ .
25  $\text{parent}_v \leftarrow p$ ;  $\text{distance}_v \leftarrow \text{distance}_{\text{parent}_v} + 1$ ;
26 if  $v \neq w \vee v \neq z$  then send  $\langle \text{Reverse}, (w, z), \text{path} \rangle$  to  $\text{parent}_v$ ;
27  $\forall q \in N(v), q \neq \text{parent}_v \wedge \text{edge\_status}_v[q]$ , send  $\langle \text{UpdateDist}, \text{distance}_{\text{parent}_v} + 1 \rangle$  to  $q$ ;
28 send  $\text{InfoMsg}_v$  to all  $u \in N(v)$ 
29 Upon receipt of  $\langle \text{UpdateDist}, \text{dist} \rangle \wedge \text{locally\_stabilized}(v)$ :
30  $\text{distance}_v \leftarrow \text{dist} + 1$ ;  $\forall u \in N(v), u \neq \text{parent}_v \wedge \text{edge\_status}_v[u]$ , send  $\langle \text{UpdateDist}, \text{distance}_v \rangle$  to  $u$ .
31 send  $\text{InfoMsg}_v$  to all  $u \in N(v)$ 

```

Figure 3: Algorithm at node v

3.4.2.4 Ενότητα Μείωσης Βαθμού

Για όλες τις ακμές οι οποίες δεν ανήκουν στο τρέχον δένδρο T του αλγορίθμου μας ως ακολούθως. Για κάθε μια από τις ακμές e , ένας από τους παρακείμενους κόμβους υπολογίζει το θεμελιώδη κύκλο C_e , και ελέγχει αν η ακμή e είναι βελτιωμένη ακμή, και αν ο μέγιστος βαθμός του C_e είναι ίσος με το βαθμό $\text{deg}(T)$. Αν οι δυο προηγούμενες συνθήκες είναι ικανοποιητικές, τότε

η ακμή e ανταλλάσσετε με μια ακμή του κύκλου C_e η οποία είναι εισερχόμενη σε ένα κόμβο του T με μέγιστο βαθμό. Αυτή η λειτουργία επαναλαμβάνεται έως ότου δεν υπάρχει καμία βελτιωμένη ακμή e με βαθμό ίσο με το βαθμό του τρέχοντος δένδρου. Αν ένας μπλοκαρισμένος κόμβος του C_e αντιμετωπίζετε, τότε ο αλγόριθμος προσπαθεί να μειώσει το βαθμό των μπλοκαρισμένων κόμβων, και επανάληψη, με παρόμοιο τρόπο με αυτόν που χρησιμοποιήσαμε για να μειώσουμε το βαθμό των κόμβων με το μέγιστο βαθμό. Όταν δεν υπάρχει πιθανή περαιτέρω ανάπτυξη τότε ο αλγόριθμος τερματίζει. Σημειώστε ότι αποδείχθηκε ότι ο βαθμός ενός δένδρου για το οποίο περαιτέρω ανάπτυξη δεν είναι πιθανή, είναι τουλάχιστον $\Delta^* + 1$, όπου Δ^* είναι ο βαθμός ενός MDST του G .

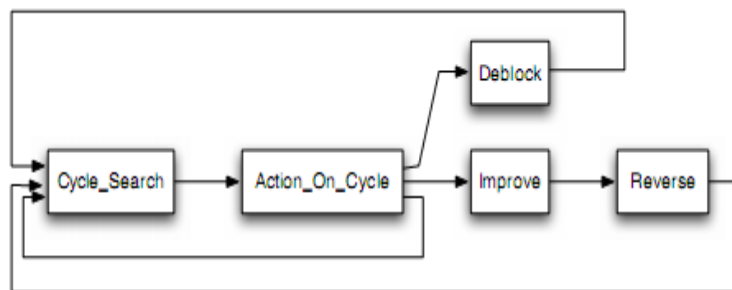


Figure 4: Degree reduction process

Cycle_Search: Επαναλαμβανόμενα κάθε ακμή η οποία δεν ανήκει στο δένδρο για το θεμελιώδη κύκλο της, χρησιμοποιεί τη διαδικασία “*Cycle_Search*” η οποία περιγράφηκε στο 4.2. Όταν μια ακμή $e = \{u, v\}$ η οποία δεν ανήκει στο δένδρο ανακαλύπτει το θεμελιώδη κύκλο της, ελέγχει αν είναι βελτιωμένη ακμή μέσω της διαδικασίας *Action_on_Cycle*.

Action_on_Cycle: Αν ο θεμελιώδης κύκλος περιέχει κόμβους χωρίς να έχουν μέγιστο βαθμό, καμία εξέλιξη δεν είναι πιθανή για το θεμελιώδη κύκλο. Αντίθετα αν ο θεμελιώδης κύκλος επίσης περιέχει μη μπλοκαρισμένους κόμβους κάποια βελτίωση είναι πιθανή. Αν δεν υπάρχουν μπλοκαρισμένοι κόμβοι η διαδικασία *Deblock* ξεκινά. Η διαδικασία αυτή προσπαθεί να βελτιώσει τους μπλοκαρισμένους κύκλους, ή μέσω της διαδικασίας *Improve* ή μέσω της διαδικασίας *Deblock*.

Improve: Η διαδικασία *Improve* στέλνει ένα μήνυμα τύπου *Remove*, το οποίο διαδίδεται κατά μήκος του θεμελιώδη κύκλου. Όταν το *Remove* μήνυμα φθάσει στον προορισμό του (ακμή $e' = \{w, z\}$), ο βαθμός και η κατάσταση της ακμής e' ελέγχονται. Αν ο μέγιστος βαθμός της ακμής ή η κατάσταση της έχουν αλλάξει, τότε το μήνυμα *Remove* απορρίπτεται. Διαφορετικά, η κατάσταση της ακμής e' τροποποιείται (γίνεται ακμή η οποία δεν ανήκει στο δένδρο), και ένα μήνυμα στέλνεται κατά μήκος του κύκλου για να διορθώσει τον προσανατολισμό των γονιών (διαδικασία *Reverse_Orientation*). Όταν το μήνυμα *Remove* φθάσει την ακμή e η οποία αρχικοποίησε τη διαδικασία τότε η e προστίθεται στο δένδρο. Στην περίπτωση όπου το μήνυμα *Remove* συναντά μια διαγεγραμμένη ακμή στο μονοπάτι του, την "κουβαλάει" αν η ακμή αυτή πρόκειται να παραμείνει ζωντανή.

Reverse_Orientation: Έπειτα από τη μετακίνηση της ακμής e' , ο προσανατολισμός του θεμελιώδη κύκλου πρέπει να έχει διορθωθεί. Αυτό επιτυγχάνεται με την κυκλοφορία δυο μηνυμάτων, των *Back* και *Remove*. Η διαδικασία *Reverse_Orientation* διαγράφει την ακμή e' και ελέγχει ακολουθώς τον προσανατολισμό της e' όταν ένα μήνυμα *Back* ή *Remove* πρέπει να χρησιμοποιηθεί για να διορθώσει τον προσανατολισμό. Αν η ακμή e' είναι προσανατολισμένη στην αντίθετη κατεύθυνση ακολουθούμενη από ένα μήνυμα *Remove*, τότε ένα μήνυμα *Remove* χρησιμοποιείται (διαφορετικά ένα *Back* μήνυμα χρησιμοποιείται). Το αποτέλεσμα της διαδικασίας *Reverse_Orientation* στο μονοπάτι από την ακμή e' έως την ακμή e πάνω στο θεμελιώδη κύκλο της e , φαίνεται στην εικόνα 5c.

Σημειώστε ότι ένα μήνυμα *Remove* (ή *Back*) διαδίδεται κατά μήκος του θεμελιώδη κύκλου μπορεί να συναντήσει μια διαγεγραμμένη ακμή από προηγούμενη βελτίωση. Αυτό υπονοεί ότι ο θεμελιώδης κύκλος της ακμής e έχει αλλάξει. Στην περίπτωση αυτή, ένα μήνυμα *Reverse* στέλνεται πίσω για να σβήσει τις τροποποιήσεις που έγιναν στο θεμελιώδη κύκλο, ο οποίος τελειώνει προσθέτοντας την διαγεγραμμένη ακμή e' . Αν οι αλλαγές που έγιναν από τη βελτίωση δεν είχαν σβήσει την ακμή αυτή τότε μπορούμε να διαχωρίσουμε τη συμπεριφορά του δένδρου. *Παρατήρηση:* Εάν μια ακμή δεν διαγράφει από μια άλλη βελτιωμένη ακμή αντιμετωπίζεται μέσα στο θεμελιώδη κύκλο πριν τη διαγραφή της ακμής e' , τότε το μήνυμα *Remove* απορρίπτεται διότι ο θεμελιώδης κύκλος έχει αλλάξει και η βελτίωση πρέπει να αλλάξει.

Deblock: Για να εκτελέσει μια εναλλαγή ακμής με ένα μπλοκαρισμένο κόμβο $w \in \{u, v\}$, ο αλγόριθμος μας ξεκινάει μειώνοντας κατά ένα το βαθμό του w . Για το λόγο αυτό, ο w μεταδίδει ένα *Deblock* μήνυμα το οποίο περιέχει το αναγνωριστικό του, σε όλα τα υποδένδρα. Όταν ένας απόγονος κόμβος w' του w , παρακείμενος σε μια ακμή η οποία δεν περιέχεται στο δένδρο, λαμβάνει ένα μήνυμα *Deblock*, προχωρά με την ανακάλυψη του θεμελιώδη κύκλου C_f του f όπως περιγράφηκε. Αν ο θεμελιώδης κύκλος C_f είναι ικανός να μειώσει το βαθμό του μπλοκαρισμένου κόμβου w , τότε ο w δεν είναι πια

μπλοκαρισμένος, και η εναλλαγή μεταξύ των ακμών e και e' μπορεί να εκτελεστεί. Εντούτοις, αν ο θεμελιώδης κύκλος C_f δεν είναι ικανός να μειώσει το βαθμό του μπλοκαρισμένου κόμβου w , τότε η διαδικασία τον κουβαλά κατ' επανάληψη μεταδίδοντας

Deblock μηνύματα από τον w' .

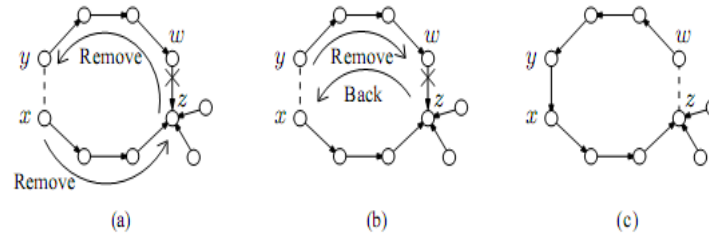


Figure 5: Illustration of Reverse_Orientation procedure: (a) when $ID_x > ID_y$, (b) when $ID_x < ID_y$, (c) the resulted network given by (a) and (b).

Σημειώστε ότι με σκοπό να αναφέρουμε ότι το δένδρο σταθεροποιείται πρέπει να ενημερώσουμε την απόσταση των κόμβων στο μονοπάτι αντιστρέφοντας έπειτα από την μετακίνηση της ακμής e' , όπως οι αποστάσεις τους στο δένδρο έχουν αλλάξει. Επομένως ένα μήνυμα *UpdateDist* διασκορπίζεται σε όλα τα παιδιά του ανεστραμμένου μονοπατιού.

3.5 Ορθότητα αλγορίθμου

Στην ενότητα αυτή αποδεικνύουμε τη ορθότητα του πρωτοκόλλου μας. Η απόδειξη χωρίζεται σε τρία τμήματα: αρχικά δείχνουμε ότι η μονάδα του spanning tree μας τελικά επιστρέφει ένα spanning tree από το δίκτυο, έπειτα δείχνουμε ότι κάθε κόμβος του δικτύου τελικά γνωρίζει το τρέχον spanning tree με το μέγιστο βαθμό από την ενότητα μεγίστου βαθμού. Τελικά αποδεικνύουμε ότι η ενότητα μείωσης βαθμού εκτελεί σωστά τις βελτιώσεις βαθμού και φθάνει στην επιθυμητή προσέγγιση στο μέγιστο βαθμό του δέντρου συγκριτικά με τη βέλτιστη λύση.

Ξεκινάμε πιο κάτω δίνοντας τον ορισμό μιας νόμιμης κατάστασης του συστήματος.

Ορισμός 2 (Νόμιμη - Αποδεκτή κατάσταση ενός MDST) .Μια διαμόρφωση του αλγορίθμου μας είναι νόμιμη όταν για κάθε διαδικασία $v \in V$ ικανοποιούνται οι ακόλουθες συνθήκες:

1. Ένα spanning tree T έχει κατασκευαστεί.

2. Ο μέγιστος βαθμός του T είναι το λιγότερο $\Delta^* + 1$, όπου Δ^* είναι ο μέγιστος βαθμός της βέλτιστης λύσης.

Σαν συνέπεια αυτού εξετάζουμε προσεκτικά τις διαφορετικές συνθήκες σχετικά με τη λανθασμένη τοπική εκτίμηση της κατάστασης ενός κόμβου με τη βοήθεια του μηνύματος InfoMsg. Το μήνυμα InfoMsg εξασφαλίζει τη συνοχή μεταξύ των αντιγράφων των μεταβλητών των κόμβων και των αρχικών τιμών τους. Επομένως ακόμα κι αν ένας κόμβος διατηρεί αλλοιωμένα αντίγραφα οι τιμές τους θα διορθωθούν μόλις ο κόμβος λάβει InfoMsg μηνύματα από κάθε έναν από τους γειτονικούς του.

Η διαδικασία μείωσης και η αναζήτηση των θεμελιωδών κύκλων είναι παγωμένοι έως ότου ένας κόμβος v σταθεροποιείται τοπικά. Δηλαδή οι ενότητες που εκτελούνται σε έναν τυχαίο κόμβο ταξινομούνται τοπικά με σειρά προτεραιότητας: Ο κόμβος με τη μεγαλύτερη προτεραιότητα γίνεται το spanning tree της κατασκευής μας, που ακολουθείται από τη μέγιστη ενότητα βαθμού και την ενότητα μείωσης βαθμού.

Στο ακόλουθο αποδεικνύουμε ότι η εκτέλεση της ενότητας μείωσης βαθμού είναι μπλοκαρισμένη για ένα πεπερασμένο χρονικό διάστημα (π.χ. για το χρόνο που χρειάζεται ο αλγόριθμος για να υπολογίσει ένα spanning tree). Δηλαδή αποδεικνύουμε ότι αλγόριθμος υπολογίζει με ένα αυτοσταθεροποιούμενο τρόπο ένα spanning tree σε πεπερασμένο αριθμό βημάτων.

Λήμμα 1. Ξεκινώντας από μια τυχαία διαμόρφωση, τελικά όλοι οι κόμβοι μοιράζονται την ίδια ρίζα.

Απόδειξη. Οι κόμβοι περιοδικά ανταλλάσσουν μηνύματα τύπου InfoMsg με τους γειτονικούς τους. Επομένως ακόμα κι αν το τοπικό αντίγραφο των γειτονικών μεταβλητών δεν ταυτίζεται με το πρωτότυπο, με βάση την αποδοχή ότι ένα InfoMsg έχει τη δυνατότητα να διορθώσει την τιμή του τοπικού αντιγράφου ενός κόμβου (μέσω της διαδικασίας *Update_State*). Υποθέστε w.r.g. δύο διαφορετικές τιμές ρίζας συνυπάρχουν στο δίκτυο. Έστω $v_1 < v_2$ οι τιμές αυτές. Έστω k ο αριθμός των κόμβων οι οποίοι έχουν ως ρίζα τον κόμβο v_2 . Δεδομένου ότι το δίκτυο είναι συνδεδεμένο υπάρχουν δυο γειτονικοί κόμβοι p_1 και p_2 τέτοιοι ώστε η τοπική ρίζα του κόμβου p_1 είναι ο κόμβος v_1 και η τοπική ρίζα του p_2 ο v_2 . Έπειτα από τη λήψη ενός μηνύματος InfoMsg από τους κόμβους p_1, p_2 ανιχνεύεται η ασυνέπιά του και αλλάζει η τοπική αξία ρίζας για τον κόμβο v_1 και γονιός του κόμβου p_1 μεταβάλλεται σύμφωνα με την εκτέλεση του κανόνα “*correction parent*”. Ως εκ τούτου η τιμή της μεταβλητής της ρίζας του δικτύου εκχωρείται στον κόμβο v_2 , μειώνοντας την κατά ένα. Επαναλαμβάνεται συνεχώς η διαδικασία αυτή έως ότου τελικά η τιμή της ρίζας για τον κόμβο v_2 πέσει στην τιμή μηδέν.

Σημειώστε ότι οι τιμές της ρίζας δεν τροποποιούνται από άλλη διαδικασία αλγορίθμου. Συνολικά, όλοι οι κόμβοι του δικτύου θα έχουν την ίδια τιμή ρίζας, δηλαδή την ίδια ρίζα.

Λήμμα 2. Ξεκινώντας από μια τυχαία διαμόρφωση, τελικά ένα μοναδικό *spanning tree* παραμένει ζωντανό μέσα στο δίκτυο.

Απόδειξη. Υποθέτουμε το αντίθετο: δυο ή περισσότερα *spanning trees* δημιουργούνται. Η παραδοχή αυτή οδηγεί είτε στην ύπαρξη δύο ή περισσότερων διαφορετικών τιμών ρίζας είτε στην παρουσία τουλάχιστον ενός κύκλου. Από τη στιγμή που οι διαφορετικές τιμές της ρίζας συνολικά ταξινομούνται τότε με βάση το λήμμα 1 μόνο μια από αυτές τις τιμές τελικά θα κερδίσει τον ανταγωνισμό ο οποίος ακυρώνει τις πολλαπλές υποθέσεις ριζών.

Υποθέτοντας ότι υπάρχει ένας κύκλος μέσα στο δίκτυο. Σημειώστε ότι υπάρχει μια μοναδική τιμή ρίζας και για κάθε κόμβο η απόσταση του από τη ρίζα είναι η απόσταση που έχει ο γονιός του συν ένα. Μέσα στον κύκλο υπάρχει το λιγότερο ένας κόμβος για τον οποίο το κατηγορημα *coherent_distance* επιστρέφει την τιμή *false*. Ο κόμβος αυτός τελικά θα εκτελέσει τον κανόνα “*correction root*”. Σύμφωνα με την παραπάνω απόδειξη του λήμματος 1 ο νέος κόμβος θα εκτελέσει ένα διαγωνισμό μεταξύ της υπάρχουσας τιμής ρίζας και της ρίζας με την χαμηλότερη τιμή ID, ένας κόμβος εξ αυτών τελικά θα παραμένει ρίζα ενώ η ο άλλος θα τροποποιήσει τις μεταβλητές της ρίζας και τις απόστασης αυτής. ■

Υποθέτοντας ότι ένα *spanning tree* του δικτύου κατασκευάζεται, τώρα θα αποδείξουμε ότι η μονάδα μεγίστου βαθμού υπολογίζει το μέγιστο μη-βαθμό του κατασκευασμένου *spanning tree*. Σημειώστε πως αν ένα *spanning tree* δεν κατασκευαστεί τότε υπάρχει ένας κόμβος *v* του οποίου οι κανόνες της μονάδας μεγίστου βαθμού δεν μπορούν να εκτελεστούν διότι το κατηγορημα *tree_stabilized(v)* δεν ικανοποιείται.

<pre> 1 Update.State(u) 2 update the local copy of u's variables. 3 use Rules 1 to 5 to correct the local state of v. 4 $\text{edge_status}_v[u] \leftarrow \text{is_tree_edge}(v, u)$; 5 $\text{color_tree}_v \leftarrow \text{degree_stabilized}(v)$; 6 Action_on_Cycle($\text{idblock}, y, \text{path}, s$) 7 if $\text{idblock} = \text{NIL}$ then 8 let $d_path = \max\{\deg_u : u \in \text{path}\}$ 9 if $d\max_v = d_path$ then 10 if $\max\{\deg_v, \deg_{v,y}\} = d\max_v - 1$ then 11 Deblock(y, s); 12 else 13 if $\max\{\deg_v, \deg_{v,y}\} < d\max_v - 1$ 14 then 15 let $w = \min\{\text{ID}_u : \deg_u =$ 16 $d_path, u \in \text{path}\} \text{ and } (w, z) \in \text{path}$ 17 Improve($y, \deg_w, (w, z), \text{path}$); 18 else if $\text{idblock} \in \text{path}$ then 19 if $\max\{\deg_v, \deg_{v,y}\} = d\max_v - 1$ then 20 Deblock(y, s); 21 else 22 if $\max\{\deg_v, \deg_{v,y}\} < d\max_v - 1$ then 23 let $e = (\text{idblock}, z) \in \text{path}$ 24 Improve($y, \deg_{\text{idblock}}, e, \text{path}$); 23 Broadcast($\text{idblock}, s$) 24 $\forall w \in N(v), w \neq s, \text{edge_status}_v[w],$ 25 send Deblock(idblock) to w. 26 Cycle_Search(idblock); 27 source_remove($v, (x, y)$) $\equiv (v = x$ $\wedge \text{edge_status}_v[y]) \vee (v = y \wedge \text{edge_status}_v[x])$; </pre>	<pre> 28 Improve($y, \deg, e, \text{path}$) 29 send Remove($(v, y), \deg, e, \text{path}$) to $\text{head}(\text{path})$ 30 Deblock(y, s) 31 if $\deg_v \geq \deg_y$ then Broadcast(v, s); 32 if $\deg_y \geq \deg_v$ then send Deblock(y) to y 33 Reverse_Orientation((x, y), $\deg, (w, z), u, \text{path}$) 34 let $\text{path} = \text{list}_1 \oplus v \oplus \text{list}_2$. 35 if $u = \text{parent}_v$ then 36 let $p = \text{head}(\text{reverse}(\text{list}_1))$. 37 if $\text{edge_status}_v[p]$ then 38 $\text{parent}_v \leftarrow p$; $\text{distance}_v \leftarrow \text{distance}_p + 1$; 39 $\text{color_tree}_v \leftarrow \neg \text{color_tree}_v$; 40 send Remove($(x, y), \deg, (w, z), \text{path}$) 41 to p 42 else 43 let $q = \text{head}(\text{list}_2)$. 44 if $\text{edge_status}_v[q]$ then 45 $\text{parent}_v \leftarrow q$; $\text{distance}_v \leftarrow \text{distance}_q + 1$; 46 $\text{color_tree}_v \leftarrow \neg \text{color_tree}_v$; 47 send Back($(x, y), (w, z), \text{path}$) to q 47 Reverse_Aux((w, z), path) 48 send Reverse($(w, z), \text{path}$) to parent_v. 49 remove_aux($v_1, v_2, \deg_max$) $\equiv \text{parent}_{v_1} = v_2$ $\wedge \deg_{v_2} = \deg_max \wedge \text{edge_status}_{v_1}[v_2] = \text{true}$; 50 remove_ok($\deg_max, (w, z)$) $\equiv (v = w$ $\wedge \text{remove_aux}(v, z, \deg_max)) \vee (v = z$ $\wedge \text{remove_aux}(v, w, \deg_max))$; 51 target_remove((w, z)) $\equiv (v = w$ $\wedge \text{parent}_v = z) \vee (v = z \wedge \text{parent}_v = w)$; </pre>
--	---

Figure 6: Algorithm's procedures at node v

Λήμμα 3. Ξεκινώντας από μια τυχαία διαμόρφωση η οποία περιέχει ένα spanning

tree T , τελικά για κάθε κόμβο v η μεταβλητή $d\max_t_v$ αποθηκεύει το μέγιστο βαθμό κόμβου του υποδένδρου με ρίζα τον κόμβο v .

Απόδειξη. Θεωρούμε ότι ένα spanning tree T έχει κατασκευαστεί, παραδειγματος χάρη, το κατηγορημα $\text{tree_stabilized}(v)$ ικανοποιείται για κάθε κόμβο v . Αποδεικνύουμε το λήμμα με επαγωγή στο ύψος του δένδρου T . Για κάθε κόμβο v , το κατηγορημα $\text{local_max_degree}(v)$ επιστρέφει τη μέγιστη τιμή μεταξύ του βαθμού του κόμβου v στο T και της μέγιστης τιμής της μεταβλητής $d\max_t_v$ των παιδιών του κόμβου v .

Εξετάστε τα φύλλα του κόμβου v του δένδρου T . Εάν ο κόμβος v δεν ικανοποιεί το λήμμα τότε αυτό υπονοεί ότι $d\max_t_v \neq 1$ διότι v είναι υποδένδρο του. Από τη στιγμή που ο κόμβος v είναι φύλλο, ο v δεν έχει παιδιά και το κατηγορημα $\text{local_max_degree}(v)$ επιστρέφει το βαθμό του v ο οποίος είναι ίσος με ένα. Κατά συνέπεια, η φρουρά του κανόνα $RMax_dFeedB$ ικανοποιείται διότι ισχύει $d\max_t_v \neq 1$ και ο κόμβος v μπορεί να εκτελέσει τον κανόνα

$RMax_dFeedB$ για να διορθώσει την τιμή της μεταβλητής $d_{max_t_v}$ η οποία ελέγχει το λήμμα. Θεωρώντας οποιοδήποτε εσωτερικό κόμβο v του T . Υποθέτουμε από επαγωγική υπόθεση πως κάθε κόμβος στο υποδένδρο το οποίο έχει ρίζα τον κόμβο v έχει αποθηκευμένη στη μεταβλητή $d_{max_t_v}$ τη σωστή τιμή. Θεωρώντας, πως ο κόμβος v δεν ικανοποιεί το λήμμα, αυτό υπονοεί ότι $d_{max_t_v} \neq local_max_degree(v)$. Ο κόμβος v μπορεί να εκτελέσει τον κανόνα $RMax_dFeedB$ για να διορθώσει τις τιμές της μεταβλητής $d_{max_t_v}$ και το λήμμα ικανοποιείται. Επομένως, έπειτα από ένα πεπερασμένο αριθμό βημάτων για κάθε κόμβο v στο δένδρο T η μεταβλητή $d_{max_t_v}$ αποθηκεύει το μέγιστο-βαθμό του υποδένδρου του κόμβου v .

Λήμμα 4. Ξεκινώντας από μια αυθαίρετη διαμόρφωση η οποία περιέχει ένα spanning tree T , τελικά για κάθε κόμβο v η μεταβλητή $d_{max_t_v}$ αποθηκεύει το μέγιστο βαθμό-κόμβου του δένδρου T .

Απόδειξη. Υποθέτουμε πως ένα spanning tree T έχει κατασκευαστεί, , παραδείγματος χάρη, το κατηγορήμα $tree_stabilized(v)$ ικανοποιείται για κάθε κόμβο v . Αποδεικνύουμε το λήμμα με επαγωγή στο ύψος του δένδρου T . Επιπλέον, σύμφωνα με το λήμμα 3 υποθέτουμε πως για κάθε κόμβο v του T η μεταβλητή $d_{max_t_v}$ αποθηκεύει το μέγιστο βαθμό-κόμβου του υποδένδρου με ρίζα τον κόμβο v , παραδείγματος χάρη έχουμε $d_{max_t_v} = local_max_degree(v)$.

Θεωρώντας τη ρίζα v του T στο μηδενικό ύψος. Από τη στιγμή που ο v είναι ρίζα του T , η μεταβλητή $d_{max_t_v}$ περιέχει το μέγιστο βαθμό-κόμβου με ρίζα τον v όπου είναι ο μέγιστος βαθμός-κόμβου του T . Αν ο κόμβος v δεν ικανοποιεί το λήμμα τότε έχουμε $d_{max_v} \neq d_{max_t_v}$. Κατά συνέπεια, ο v μπορεί να εκτελέσει τον κανόνα $RMax_dRoot$ για να διορθώσει τις τιμές της μεταβλητής d_{max_v} ώστε να ικανοποιείται το λήμμα. Υποθέτοντας από επαγωγική υπόθεση πως κάθε κόμβος v στο ύψος i έχει το μέγιστο βαθμό-κόμβου το οποίο αποθηκεύεται στη μεταβλητή d_{max_v} . Θεωρώντας οποιοδήποτε τυχαίο κόμβο v στο ύψος $i+1$. Αν ο κόμβος αυτός δεν ικανοποιεί το λήμμα υπονοείται πως ο κόμβος αυτός έχει αποθηκευμένη στη μεταβλητή του d_{max_v} μια τιμή η οποία είναι διαφορετική με εκείνη που έχει αποθηκευμένη ο γονιός του κόμβου ο οποίος έχει αποθηκευμένη και τη σωστή τιμή από επαγωγική υπόθεση, για παράδειγμα έχουμε $d_{max_v} \neq d_{max_{parent_v}}$. Κατά συνέπεια, ο κόμβος v μπορεί να εκτελέσει τον κανόνα $RMax_dPro$ για να διορθώσει τις τιμές της μεταβλητής d_{max_v} και να ικανοποιείται το λήμμα. Επομένως, έπειτα από ένα πεπερασμένο αριθμό βημάτων για κάθε κόμβο v του T η μεταβλητή d_{max_v} αποθηκεύει το μέγιστο βαθμό-κόμβου του spanning tree του T . ■

Τώρα θα αποδείξουμε ότι ο αλγόριθμος μας διατηρεί την ίδια απόδοση με εκείνη του αλγορίθμου που προτάθηκε από τους Furer και Raghavachari “Approximated distributed algorithm for the minimum-degree steiner tree to within one of optimal ” και συγκλίνει σε μια νόμιμη- επιθυμητή διαμόρφωση η οποία από επαγωγική υπόθεση ικανοποιεί το Θεώρημα 1.

Θεώρημα 1. (Από τους Furer και Raghavachari) Έστω ένα spanning tree T βαθμού Δ του γραφήματος G . Έστω Δ^* ο βαθμός ενός ελαχίστου-βαθμού spanning tree. Έστω S το σύνολο των κορυφών του βαθμού Δ του δένδρου T . Έστω B ένα αυθαίρετο υποσύνολο ακμών βαθμού $\Delta-1$ του δένδρου T . Έστω ότι το σύνολο $S \cup B$ αφαιρείται από το γράφημα, σπάζοντας το δένδρο T σε ένα δάσος (forest) F . Υποθέτουμε ότι το γράφημα G ικανοποιεί τη συνθήκη η οποία υποστηρίζει πως δεν υπάρχουν ακμές μεταξύ δυο διαφορετικών δένδρων μέσα στο F . Τότε θα ισχύει $\Delta \leq \Delta^* + 1$.

Καλούμε βελτίωση ένα βήμα του αλγορίθμου μας κατά το οποίο ο βαθμός το λιγότερο σε έναν από τους κόμβους μεγίστου βαθμού μειώνεται κατά 1, χωρίς να μειώνεται ο βαθμός του κόμβου ο οποίος έχει ήδη το μέγιστο βαθμό, αλλά ούτε και ο βαθμός ενός blocking κόμβου.

Λήμμα 5. Όταν κατά την εφαρμογή μιας διαδικασίας *Im prove*, ο αλγόριθμος μας εκτελεί κάποια βελτίωση.

Απόδειξη. Έστω $e = \{u, v\}$ μια βελτιωμένη ακμή του τρέχοντος δένδρου T μεγίστου βαθμού k . Προτού προσθέσουμε την ακμή e στο δένδρο T μέσω της διαδικασίας *Im prove*, κάποιο από τα άκρα της ακμής e στέλνει ένα μήνυμα *Remove* κατά μήκος του κύκλου C_e με σκοπό να διαγραφούν κάποιες ακμές $e' = \{x, y\}$ του κύκλου, προσκείμενες σε έναν κόμβο βαθμού k ή $k-1$. Το μήνυμα *Remove* δρομολογείται κατά μήκος του C_e . Όταν το μήνυμα φθάσει σε ένα άκρο της ακμής e' , διακρίνουμε δυο περιπτώσεις:

Περίπτωση 1: Η ακμή e' εξακολουθεί να είναι ακμή κάποιου δένδρου βαθμού k ή $k-1$. Τότε, σύμφωνα με τη διαδικασία *Im prove*, η ακμή e' αφαιρείται από το δένδρο T , και ο κόμβος u ή ο κόμβος v ενημερώνεται μέσω ενός *Remove* ή ενός *Back* μηνύματος για το γεγονός του ότι η ακμή e μπορεί να προστεθεί στο δένδρο T . Στην περίπτωση αυτή, έχουμε μια βελτίωση.

Περίπτωση 2: Η ακμή e' δεν είναι πια ακμή κάποιου δένδρου βαθμού k ή $k-1$. Αυτό υπονοεί ότι ο βαθμός του κύκλου C_e έχει μειωθεί (από μια ταυτόχρονα βελτιωμένη ακμή), παράγοντας μια βελτίωση. Στην περίπτωση αυτή, σύμφωνα με τη διαδικασία *Im prove*, το μήνυμα *Remove* απορρίπτεται,

για τη συντήρηση της ιδιοκτησίας των spanning trees. Και στις δυο περιπτώσεις, εμφανίζεται μια βελτίωση. ■

Για να λάβει την ίδια απόδοση με εκείνη του αλγορίθμου που παρουσίασαν οι Furer και Raghavachari, είναι απαραίτητο σε κάποια διαμόρφωση να γίνει ένας blocking κόμβος non blocking διότι αυτό οδηγεί στη μείωση του βαθμού του μέγιστου βαθμού κόμβου. Το ακόλουθο λήμμα δηλώνει αν είναι δυνατόν να γίνει ένας blocking κόμβος non blocking, έπειτα ο αλγόριθμος μας κάνει non blocking έναν blocking κόμβο χωρίς να μειώνεται ο βαθμός του δένδρου.

Λήμμα 6. Έστω w ένας blocking κόμβος ο οποίος μπορεί να γίνει non blocking. Τότε η διαδικασία *Deblock*(w) μειώνει το βαθμό του w κατά 1, χωρίς να μειώνεται ούτε ο βαθμός του μέγιστου βαθμού κόμβου, ούτε ο βαθμός του blocking κόμβου.

Απόδειξη. Η απόδειξη πραγματοποιείται μέσω επαγωγής στον αριθμό των κλήσεων της διαδικασίας *Deblock*. Αν στην πρώτη κλήση της διαδικασίας αυτής υπάρχει μια βελτιωμένη ακμή σε ένα υποδένδρο του w (σύμφωνα με το θεώρημα των Furer και Raghavachari είναι ικανοποιητικό να ψάχνουμε για βελτιωμένη ακμή στο υποδένδρο του w) τότε μια βελτίωση πραγματοποιείται από τη διαδικασία *Improve* μειώνοντας το βαθμό του w σύμφωνα με το λήμμα 5. Εάν δεν υπάρχει βελτιωμένη ακμή αλλά το λιγότερο ένας blocking κόμβος παρακείμενος σε μια ακμή η οποία δεν ανήκει στο δένδρο του υποδένδρου w το οποίο περιέχει το w μέσα στο θεμελιώδη κύκλο του, τότε η διαδικασία *Deblock* καλείται κατ' επανάληψη με τη χορήγηση αυτών των διαδικασιών *Deblock*. Έτσι αν μια βελτίωση είναι πιθανή, αυτή η βελτίωση διαδίδεται στο w από ακολουθίες βελτιώσεων. Σημειώστε ότι κάθε βελτίωση μπορεί μόνο να μειώσει το βαθμό των κόμβων με βαθμό $\leq k-2$ σύμφωνα με το λήμμα 5, έτσι η διαδικασία *Deblock*(w) μπορεί να μην μειώσει το βαθμό ούτε του κόμβου με το μέγιστο βαθμό, αλλά ούτε και του blocking κόμβου. ■

Κατά τη διάρκεια μιας βελτίωσης ενός θεμελιώδη κύκλου, ο αλγόριθμος μας αλλάζει ένα τμήμα του spanning tree με την τροποποίηση του γονέα κάποιων κόμβων του θεμελιώδη κύκλου (διαδικασία αντιστροφής άκρων). Το ακόλουθο λήμμα αποδεικνύει ότι κατά τη διάρκεια μιας βελτίωσης, ο αλγόριθμος μας δεν σταθεροποιεί την spanning tree διαδικασία.

Λήμμα 7. Κατά τη διάρκεια μιας διαδικασίας αντιστροφής άκρων, το κατηγορημα *tree_stabilized*(v) ικανοποιείται για κάθε κόμβο v του δένδρου.

Απόδειξη. Πρέπει να θεωρούμε κάθε κόμβο v του δένδρου (εκτός της ρίζας) οι οποίοι λαμβάνουν μέρος στη διαδικασία αντιστροφής άκρων. Η διαδικασία

αντιστροφής άκρων επιτυγχάνεται μέσω των μηνυμάτων *Remove*, *Back* και *Reverse*. Όταν ένα από τα μηνύματα αυτά λαμβάνεται από έναν κόμβο v , διαχειριζόμαστε το μήνυμα αν και μόνο αν το κατηγορημα $locally_stabilized(v)$ είναι αληθές. Υποθέτουμε ότι ένα *spanning tree* κατασκευάζεται (βλέπε λήμμα 2) και υπολογίζεται ο κόμβος μεγίστου βαθμού, αυτό υπονοεί ότι κάθε κόμβος v έχει ένα κόμβο γονέα και μια συνεπή απόσταση από αυτόν. Για να απλοποιήσουμε την απόδειξη, εστιάζουμε στο κατηγορημα $tree_stabilized(v)$ αντί αυτού στο κατηγορημα $locally_stabilized(v)$ για κάθε κόμβο v . Πράγματι, θεωρούμε ότι το κατηγορημα $color_stabilized(v)$ είναι αρχικά αληθές και παραμένει αληθές στο θεμελιώδη κύκλο κατά τη διάρκεια της διαδικασίας αντιστροφής άκρων διότι θεωρούμε ότι έχουμε κανάλια FIFO. Κατά συνέπεια, η επαγωγή του κόμβου μεγίστου βαθμού μέσα στο δένδρο λαμβάνεται υπόψη στο τέλος της διαδικασίας αντιστροφής άκρων (βλέπε το χειρισμό των *InfoMsg* μηνυμάτων). Όταν ένας κόμβος v εκτελεί τη διαδικασία αντιστροφής άκρων τότε ο κόμβος αυτός λαμβάνει ένα νέο κόμβο γονέα και ενημερώνει την απόσταση από αυτόν. Για να αποδείξουμε το λήμμα, πρέπει να δείξουμε ότι κατά τη διάρκεια της διαδικασίας αντιστροφής ακμής κάθε κόμβος v έχει ένα γονέα στη γειτονιά του και έχει μια απόσταση ίση με αυτή της απόστασης του από τον κόμβο γονέα του συν ένα.

Θεωρήστε οποιονδήποτε κόμβο v ο οποίος να ικανοποιεί το κατηγορημα $tree_stabilized(v)$ και εκτελεί τη διαδικασία αντιστροφής άκρων. Σύμφωνα με την περιγραφή του αλγορίθμου για τη διαδικασία της αντιστροφής άκρων, ένας κόμβος λαμβάνει ως νέο γονέα τον προκατόχο του στο θεμελιώδη κύκλο, με συνέπεια να μπορεί να λάβει μόνο ένα γείτονα ως νέο γονέα. Τώρα επιλέγουμε αν η συνοχή απόστασης αναφέρεται. Υπάρχουν δυο περιπτώσεις, η ακμή μεταξύ του κόμβου v και του προκατόχου του p στο θεμελιώδη κύκλο είναι (i) ακμή η οποία δεν ανήκει σε κάποιο δένδρο, (ii) ακμή δένδρου. Στην πρώτη περίπτωση, ο κόμβος v στέλνει στον κόμβο γονέα του ένα μήνυμα *Reverse* για να αντιστρέψει τον προσανατολισμό των άκρων μεταξύ του κόμβου v και της ακμής στόχου. Κατά συνέπεια, το κατηγορημα $tree_stabilized(v)$ ακόμη ικανοποιείται διότι ο κόμβος v δεν έχει αλλάξει τον κόμβο γονέα του. Η ανανέωση του υπολοίπου μονοπατιού μεταξύ του κόμβου v και της ακμής στόχου βρίσκεται στην ίδια περίπτωση την (ii) που αναφέρεται πιο πάνω (Δέστε τον χειρισμό του μηνύματος *Reverse* και της διαδικασίας *Reverse_Aux*). Στην περίπτωση (ii), ο κόμβος v αλλάζει το γονέα του και ανανεώνει την απόσταση από αυτόν. Έπειτα, ο v στέλνει αρχικά στο νέο γονέα του p μήνυμα *Remove* ή *Back* και έπειτα στέλνει στους γείτονες του ένα *InfoMsg*. Κατά συνέπεια, ο v να ικανοποιεί το κατηγορημα $tree_stabilized(v)$ έπειτα από την εκτέλεση της διαδικασίας αντιστροφής άκρων, αλλά θα πρέπει να δείξουμε ότι και ο p ικανοποιεί το κατηγορημα $tree_stabilized(p)$ επίσης έως ότου δημιουργήσουμε ένα προσωρινό κύκλο μεταξύ των κόμβων v και p .

Δεδομένου ότι χρησιμοποιούμε ένα άρτιο ατομικό μοντέλο, ο p διατηρεί ένα αντίγραφο της παλιάς απόστασης από το γονέα του v . Επιπλέον, θεωρούμε κανάλια FIFO στο δίκτυο μας με αποτέλεσμα τα *Reverse*, *Back* ή *Remove* μηνύματα λαμβάνονται από τον p πριν από το μήνυμα *InfoMsg*. Όσο θεωρούμε πως το κατηγορήμα $tree_stabilized(p)$ ικανοποιείται αρχικά, έπειτα από τη διαμόρφωση του κόμβου γονέα και της απόστασης από αυτόν για τις μεταβλητές του κόμβου v ο κόμβος p εξακολουθεί να ικανοποιεί το κατηγορήμα $tree_stabilized(p)$ διότι ο p χρησιμοποιεί ακόμη την παλιά κατάσταση του v . Κατά συνέπεια, ο p μπορεί να εκτελέσει τη διαδικασία αντιστροφής άκρων όπως περιγράφηκε για τον κόμβο v . Όταν ο p λαμβάνει ένα *InfoMsg* από τον v , έπειτα ο p ανανεώνει το τοπικό του αντίγραφο για την κατάσταση του v αλλά ο κόμβος v δεν είναι πλέον γονέας του p κι αυτό έχει επιπτώσεις στην ικανοποίηση του κατηγορήματος $tree_stabilized(p)$. Σημειώστε ότι για να διατηρήσουμε τη συνοχή της απόστασης, το μήνυμα *UpdateDist* διαδίδεται στο υποδένδρο ενός κόμβου ο οποίος έχει αλλάξει το γονέα του και κάθε κόμβος που λαμβάνει αυτό το μήνυμα ανανεώνει τη δική του απόσταση ανάλογα με τον κόμβο γονέα του (Κοιτάξτε το χειρισμό του μηνύματος *UpdateDist*).

Χρησιμοποιώντας το ίδιο επιχείρημα, ένα μπορούμε να δείξουμε από την επαγωγή στο μήκος του θεμελιώδη κύκλου ότι το κατηγορήμα $tree_stabilized(v)$ ικανοποιείται για κάθε κόμβο v . ■

Δεδομένου του ότι εκτελούμε βελτιώσεις σε όλους τους θεμελιώδεις κύκλους ταυτόχρονα πρέπει να εξασφαλίσουμε ότι το λιγότερο μια βελτίωση επιτυγχάνεται. Το ακόλουθο λήμμα δηλώνει εάν βελτιώσεις πραγματοποιούνται σε non-disjoint θεμελιώδεις κύκλους (π.χ. θεμελιώδη κύκλοι με κοινά μονοπάτια) τότε το λιγότερο μια βελτίωση εκτελείται.

Λήμμα 8. Έστω $C1$ και $C2$ δυο non-disjoint θεμελιώδη κύκλοι. Αν η διαδικασία *Improve* εφαρμόζεται παράλληλα στους $C1$ και $C2$ τότε το λιγότερο μια βελτίωση εκτελείται.

Απόδειξη. Υποθέτουμε ότι ένα συνεκτικό spanning tree έχει κατασκευαστεί, διαφορετικά οι διαδικασίες βελτίωσης δεν γίνεται να εφαρμοστούν διότι το κατηγορήμα *locally_stabilized* δεν ικανοποιείται. Κατά τη διάρκεια της ανάπτυξης ενός θεμελιώδη κύκλου, ο προσανατολισμός ενός τμήματος του θεμελιώδη κύκλου αντιστρέφεται χρησιμοποιώντας τα μηνύματα *Remove* ή *Back*. Βελτιώσεις σε non-disjoint θεμελιώδεις κύκλους μπορούν να παρέμβουν κατά τη διάρκεια της διαδικασίας αντιστροφής άκρων. Έστω C ένας θεμελιώδης κύκλος και $w, x \in C$ να είναι αντίστοιχα οι κόμβοι από τους οποίους ξεκινά (διαγράφεται η προσκείμενη ακμή του κόμβου μεγίστου βαθμού) και

τελειώνει (προσθ  εται μια ακμ   στο δ  νδρο η οποία δεν   ταν ακμ   δ  νδρου) η διαδικασία αντιστροφ  ς προσανατολισμο  .   στω $P'(C)$ το μονοπάτι του spanning tree που αντιστρέφεται μεταξύ των w και x κατά τη διάρκεια μιας βελτίωσης στο θεμελιώδη κύκλο C . Για να δείξουμε το λ  μμα, θεωρούμε δυο non-disjoint θεμελιώδης κύκλους $C1$, $C2$ και αποδεικν  ουμε   τι οι ακόλουθες διάφορες συνθήκες εκτελούνται το λιγότερο σε μια βελτίωση: Οι δυο βελτι  σεις (i) προσπαθούν να μειώσουν τον ίδιο κόμβο μεγίστου βαθμού, (ii) αντιστρέφουν μονοπάτια τέτοια   στε $P'(C1) \cap P'(C2) = \emptyset$ ή (iii) $P'(C1) \cap P'(C2) \neq \emptyset$.

Σημει  στε   τι κατά τη διάρκεια μιας βελτίωσης και πριν φθάσουμε στην ακμ   στόχο (στην προσκείμενη ακμ   του μεγίστου βαθμού κόμβου),   νας κόμβος v ο οποίος λαμβάνει   να μήνυμα *Remove* και ανιχνεύει   τι η ακμ   με τη διάδοχη ακμ   της p στο θεμελιώδη κύκλο δεν αν  κουν πλέον στο δ  νδρο ($edge_status_v[p] = false$)   τσι απορρίπτει το *Remove* μήνυμα. Το μήνυμα   υτό απορρίφθηκε δι  τι ο θεμελιώδης κύκλος   χει αλλάξει.

Στην περίπτωση (i),   σο θεωρούμε   να ασ  γχρονο σ  στημα   που μια   πό δυο βελτι  σεις οι οποίες πραγματοποιούνται φθάνει πρώτη στην ακμ   στόχο και διαγράφει αυτήν η οποία μειώνει το βαθμό του μεγίστου βαθμού κόμβου.   ταν η δεύτερη βελτίωση φθάσει στην ακμ   στόχο τότε καμία βελτίωση δεν πραγματοποιείται δι  τι το κατηγορημα *remove_ok* δεν ικανοποιείται   φ   ο βαθμός του μεγίστου βαθμού κόμβου   χει   δη μειωθεί. Κατά συνέπεια, σύμφωνα με την περιγραφή του αλγορίθμου το μήνυμα απορρίπτεται. Επομένως, στην περίπτωση αυτή η πρώτη βελτίωση μόνο πραγματοποιείται.

Στην περίπτωση (ii),   ως   του ισχύει $P'(C1) \cap P'(C2) = \emptyset$ μια διαδικασία αντιστροφ  ς ακμ  ς δεν μπορεί να παρέμβει με την   λλη διαδικασία αντιστροφ  ς ακμ  ς. Κατά συνέπεια, οι δυο βελτι  σεις μπορούν να πραγματοποιηθούν χωριστά σύμφωνα με το λ  μμα 5.

Στην περίπτωση (iii),   χουμε $P'(C1) \cap P'(C2) \neq \emptyset$, το κοινό μονοπάτι των $P'(C1)$ και $P'(C2)$ αντιστρέφεται   πό τις δυο διαδικασίες βελτίωσης. Οι δυο διαδικασίες αντιστροφ  ς ακμ  ς δεν μπορούν να αντιστρέψουν τις ακμ  ς του κοινού μονοπατιο   σε διαφορετικές κατευθ  νσεις, αντι  τως υπονοεί   τι ο κόμβος   χει δυο κόμβους ως γονέα ή   τι υπάρχει κακ  ς προσανατολισμός στο spanning tree γεγονός το οποίο   ρχεται σε αντίθεση με την υπόθεση   τι   να συνεκτικό spanning tree   χει κατασκευαστεί.   τσι, οι δυο διαδικασίες αντιστροφ  ς ακμ  ς αντιστρέφουν την ακμ   προς την ίδια κατεύθυνση. Η βελτίωση η οποία πρώτη αντιστρέφει τις ακμ  ς του κοινού μονοπατιο   των $P'(C1)$ και $P'(C2)$ συντηρείται (Λ  μμα 5). Πράγματι, η δεύτερη διαδικασία βελτίωσης   τε συντηρείτε   τε ακυρώνετε. Στην τελευταία περίπτωση, η διαδικασία αντιστροφ  ς ακμ  ς διασχι  ζει μια ακμ   (u, v) η οποία δεν   ναι πια ακμ   του δ  νδρου. Αυτό υπονοεί πως ο θεμελιώδης κύκλος της δεύτερης διαδικασίας βελτίωσης   χει αλλάξει.   πειτα,   να μήνυμα *Reverse* στέλνεται πίσω για να αντιστρέψει τον προσανατολισμό   πό τον τρέχοντα κόμβο της ακμ  ς στόχου, ο οποίος οδηγεί στο να προστεθεί ξανά η ακμ   στόχου στο δ  νδρο. Στην προηγούμενη περίπτωση, η διαδικασία αντιστροφ  ς ακμ  ς

διασχιζει μόνο ακμές του δένδρου και αυτό συνεπάγεται πως η δεύτερη διαδικασία βελτίωσης πραγματοποιείται σωστά ως το τέλος. Επομένως, μια τουλάχιστον διαδικασία βελτίωσης πραγματοποιείται σε κάθε περίπτωση. ■

Θεώρημα 2. Ο αλγόριθμος επιστρέφει ένα spanning tree του G το λιγότερο βαθμού $\Delta^* + 1$.

Απόδειξη. Στα ακόλουθα θα αποδείξουμε ότι μια νόμιμη διαμόρφωση ικανοποιεί την υπόθεση του θεωρήματος 1. Αυτό συμβαίνει, όταν ο αλγόριθμος μας φθάσει σε μια νόμιμη διαμόρφωση, τότε δεν υπάρχει πιθανότητα για περαιτέρω βελτίωση. Υποθέστε την αντίθεση. Υποθέτουμε ότι υπάρχει μια δομή δένδρου T με μέγιστο βαθμό k . Υποθέτουμε πως υπάρχει κάποια πιθανή βελτίωση για το T , από την εκτέλεση ή εναλλαγή μεταξύ των ακμών $e1 \notin T$ και $e2 \in T$ για να λάβουμε το δένδρο $T' = T \cup \{e1\} \setminus \{e2\}$ και υποθέτουμε πως ο αλγόριθμος δεν μπορεί να εκτελέσει κάποια βελτίωση. Αυτό υπονοεί πως η ακμή $e1$ έχει βαθμό ίσο με k , διαφορετικά όπως δείξαμε στα λήμματα 5 και 6 ο κόμβος με το μικρότερο id που πρόσκειται στην ακμή $e1$ θα εκτελέσει μια βελτίωση μέσω της διαδικασίας *Improve* ή τρέχοντας τη διαδικασία *Deblock* με σκοπό να μειώσει το βαθμό της ακμής $e1$, σύμφωνα με τη διαδικασία *Action_on_Cycle*. Υπάρχει αντίθεση με το γεγονός του ότι η ακμή $e1$ μπορεί να χρησιμοποιηθεί για να εκτελέσει βελτίωση, και του γεγονότος πως αν μπορεί να υπάρξει κάποια βελτίωση για το T τότε ο αλγόριθμος θα εκτελέσει τη βελτίωση.

Λήμμα 9. Ξεκινώντας από μια τυχαία διαμόρφωση, ο αλγόριθμος επιστρέφει μια νόμιμη διαμόρφωση.

Απόδειξη. Ο αλγόριθμος μας είναι μια παράλληλη σύνθεση τριών επιπέδων: την κατασκευή ενός spanning tree, τον υπολογισμό μεγίστου βαθμού και του βαθμού της μείωσης επιπέδου. Σύμφωνα με τα λήμματα 2 και 4, ένα spanning tree κατασκευάζεται και ο μέγιστος βαθμός του υπολογίζεται ξεκινώντας από μια τυχαία διαμόρφωση. Σημειώστε πως η μονάδα μεγίστου βαθμού δεν μπορεί να σταθεροποιήσει τη μονάδα του spanning tree διότι δεν μπορεί να τροποποιήσει τη μεταβλητή γονέα για οποιοδήποτε κόμβο. Παρόλο αυτά, η μονάδα μείωσης βαθμού τροποποιεί το δένδρο (μεταβλητές γονέων) και τους μεγίστους βαθμούς (μεταβλητές d_{max}). Σύμφωνα με το λήμμα 7, η μονάδα μείωσης βαθμού δεν μπορεί να σταθεροποιήσει τη μονάδα spanning tree. Έπειτα από τη μείωση μεγίστου βαθμού ενός spanning tree, οι κόμβοι μπορούν να εκτελέσουν βελτιώσεις χρησιμοποιώντας ψευδή πληροφορία σχετικά με το μέγιστο βαθμό, αλλά ένα spanning tree πάντα συντηρείται. Έπειτα από ένα πεπερασμένο αριθμό βημάτων κάθε κόμβος γνωρίζει τη σωστή τιμή για το μέγιστο βαθμό του δένδρου (Λήμμα 4). Επιπλέον, το θεώρημα 2 δηλώνει πως ξεκινώντας από μια τυχαία διαμόρφωση ένα spanning tree

βαθμού το λιγότερο $\Delta^* + 1$ επιστρέφεται από τον αλγόριθμο, με $\Delta^* + 1$ να είναι ο βαθμός της βέλτιστης λύσης. Επομένως, όταν κάθε επίπεδο σταθεροποιείται το αποτέλεσμα ακολουθεί.

Λήμμα 10. (Κλειστότητα) Ξεκινώντας από μια νόμιμη διαμόρφωση, ο αλγόριθμος παρουσιάζει διατηρεί τη νόμιμη αυτή διαμόρφωση του.

Απόδειξη. Σύμφωνα με τα Λήμματα 2 και 4, ένα spanning tree και ο μέγιστος βαθμός του διατηρούνται. Τώρα, πρέπει να δείξουμε πως ξεκινώντας από μια διαμόρφωση η οποία περιγράφει ένα spanning tree βαθμού τουλάχιστον $\Delta^* + 1$ καμία βελτίωση δεν εκτελείτε από τον αλγόριθμο, με $\Delta^* + 1$ να είναι ο βαθμός της βέλτιστης λύσης. Το θεώρημα 1 δείχνει πως όταν ένα ελαχίστου βαθμού spanning tree με βαθμό το λιγότερο $\Delta^* + 1$ επιτυγχάνεται, αυτό υπονοεί πως δεν υπάρχει άλλη πιθανή βελτίωση για το δένδρο αυτό. Σε μια νόμιμη διαμόρφωση, ο βαθμός κάθε κόμβου από τις ακμές οι οποίες δεν ανήκουν στο δένδρο είναι το λιγότερο ίσος με το μέγιστο βαθμό των υπολοίπων μείον 1. Κατά συνέπεια, ο αλγόριθμος δεν μπορεί να εκτελέσει βελτίωση σε κάποιο θεμελιώδη κύκλο διότι η συνθήκη βελτιωμένης ακμής δεν ικανοποιείται, καθέναν μέσω της διαδικασίας *Improve* ή *Deblock*. Επομένως, ξεκινώντας από μια νόμιμη διαμόρφωση ο αλγόριθμος διατηρεί τη νόμιμη αυτή διαμόρφωση. ■

3.6 Θέματα πολυπλοκότητας

Το ακόλουθο λήμμα δείχνει ότι η χρονική πολυπλοκότητα είναι πολυωνυμική όσο η πολυπλοκότητα μνήμης είναι λογαριθμική στο μέγεθος του δικτύου.

Λήμμα 11. Ο αλγόριθμος μας συγκλίνει σε μια νόμιμη κατάσταση όπου το MDST πρόβλημα σε $O(mn^2 \log n)$ γύρους χρησιμοποιεί $O(\delta \log n)$ bit μνήμης για την αποστολή/λήψη μηνυμάτων του μοντέλου, όπου δ είναι ο μέγιστος βαθμός του δικτύου.

Απόδειξη. Ο αλγόριθμος μας είναι σύνθεση τριών επιπέδων: την κατασκευή ενός spanning tree, τον υπολογισμό μεγίστου βαθμού και του βαθμού της μείωσης επιπέδου. Τα δυο πρώτα επίπεδα έχουν χρονική πολυπλοκότητα $O(n^2)$ και $O(n)$ γύρους αντίστοιχα. Η χρονική πολυπλοκότητα εξουσιάζεται από το τρίτο και τελευταίο επίπεδο. Σύμφωνα με τους Furer και Raghavachari, υπάρχουν $O(n \log n)$ φάσεις. Μια φάση περιέχει τη μείωση κατά ένα του βαθμού του κόμβου με το μέγιστο βαθμό. Όσο ισχύει $2 \leq k \leq n-1$,

προσθέτοντας τις αρμονικές σειρές σε αντιστοιχία με τις τιμές του k οδηγεί σε $O(n \log n)$ φάσεις. Κατά τη διάρκεια κάθε φάσης μια ακμή $\{u, v\}$ η οποία δεν περιέχεται στο δένδρο εκτελεί τις ακόλουθες διαδικασίες: βρίσκει θεμελιώδη κύκλους και πετυχαίνει βελτιώσεις (αν υπάρχει βελτιωμένη ακμή). Για να βρει κάθε θεμελιώδη κύκλο, η $\{u, v\}$ χρησιμοποιεί τη διαδικασία *Cycle_Search* η οποία διαδίδει ένα *Search* μήνυμα μέσω ενός DFS το οποίο διασχίζει το τρέχον δένδρο. Έτσι η πρώτη διαδικασία χρειάζεται το λιγότερο $O(n)$ γύρους. Για τη δεύτερη διαδικασία υπάρχουν δυο πιθανές περιπτώσεις: άμεση και έμμεση βελτίωση. Η άμεση βελτίωση εκτελείται αν η $\{u, v\}$ είναι βελτιωμένη ακμή, στην περίπτωση αυτή ένα μήνυμα *Remove* στέλνεται και ακολούθως ο θεμελιώδης κύκλος κάνει τις διάφορες αλλαγές, απαιτώντας $O(n)$ γύρους. Η έμμεση βελτίωση εκτελείται όταν κάποιος από τους u, v είναι μπλοκαρισμένοι κόμβοι. Στην χειρότερη περίπτωση, οι μπλοκαρισμένες αλυσίδες ακμών είναι μεγέθους το πολύ $m - (n - 1)$. Επομένως $m - (n - 1)$ εναλλαγές είναι απαραίτητες για τη μείωση του βαθμού του δένδρου. Αυτό απαιτεί $O(mn)$ γύρους (σύμφωνα με την άμεση βελτίωση). Έτσι αν η δεύτερη διαδικασία χρειάζεται το πολύ $O(mn)$ γύρους, το τρίτο επίπεδο χρειάζεται συνολικά $O(mn^2 \log n)$ γύρους.

Στα ακόλουθα αναλύουμε την πολυπλοκότητα επικοινωνίας (μνήμης) για τη λύση μας. Κάθε κόμβος περιέχει ένα σταθερό αριθμό τοπικών μεταβλητών μεγέθους $O(\log n)$ bits. Ωστόσο, εξαιτίας της ιδιαιτερότητας του μοντέλου (send/receive model) η πολυπλοκότητα επικοινωνίας περιέχει αντίγραφα των τοπικών γειτόνων μεγέθους $O(\delta \log n)$, όπου δ είναι ο μέγιστος βαθμός του δικτύου. ■

Σημειώστε ότι τα μηνύματα ανταλλάσσονται κατά τη διάρκεια της εκτέλεσης του αλγορίθμου διατηρώντας πληροφορία σχετική με το μέγεθος των θεμελιωδών κύκλων. Επομένως, η πολυπλοκότητα του μήκους των καταχωρητών της λύσης μας είναι

3.7 Συμπεράσματα και ανοιχτά προβλήματα

Προτείνουμε ένα αυτοσταθεροποιούμενο αλγόριθμο για την κατασκευή ενός ελαχίστου βαθμού spanning tree σε μη κατευθυνόμενα δίκτυα. Ξεκινώντας από μια τυχαία κατάσταση, ο αλγόριθμος μας εγγυάται πως θα συγκλίνει σε μια νόμιμη κατάσταση περιγράφοντας ένα spanning tree του οποίου ο μέγιστος βαθμός είναι το πολύ $\Delta^* + 1$, όπου Δ^* είναι ο ελάχιστος πιθανός βαθμός ενός spanning tree του δικτύου.

Για την καλύτερη γνώση μας ο αλγόριθμος μας είναι η πρώτος λύση αυτοσταθεροποιούμενου αλγορίθμου για την κατασκευή ενός ελαχίστου βαθμού spanning tree σε μη κατευθυνόμενα δίκτυα. Ο αλγόριθμος μπορεί εύκολα να εφαρμοστεί σε μεγάλης κλίμακας συστήματα από τη στιγμή που αυτά χρησιμοποιούν μόνο τοπικές συνδέσεις και όχι κόμβους οι οποίοι να συγκεντρώνουν τον υπολογισμό του MDST. Αυτό σημαίνει, πως κάθε κόμβος ανταλλάσει μηνύματα με τους γείτονες του με απόσταση ενός κόμβου και οι υπολογισμοί είναι συνολικά κατανεμημένοι. Επιπροσθέτως ο αλγόριθμος έχει σχεδιαστεί ώστε να μπορεί να λειτουργήσει σε οποιαδήποτε ασύγχρονη ανταλλαγή μηνυμάτων του δικτύου χρησιμοποιώντας μόνο αξιόπιστα FIFO κανάλια. Η χρονική πολυπλοκότητα της λύσης μας είναι $O(mn^2 \log n)$ όπου n, m είναι αποκλειστικά ο αριθμός των κόμβων και των ακμών του δικτύου αντίστοιχα. Η πολυπλοκότητα επικοινωνίας είναι $O(\delta \log n)$ για το ατομικό μοντέλο send/receive.

Διάφορα προβλήματα παραμένουν ανοικτά. Αρχικά, ο υπολογισμός ενός ελαχίστου βαθμού spanning tree σε κατευθυνόμενες τοπολογίες φαίνεται να είναι μια υποσχόμενη κατεύθυνση έρευνας με ευρεία περιοχή εφαρμογών (π.χ. δίκτυα αισθητήρων, ad-hoc δίκτυα και robot δίκτυα). Έπειτα, τα νέα προκύπτων δίκτυα χρειάζονται εξελιγμένες λύσεις προκειμένου να αντιμετωπίσουν κόμβους churn. Μια ελκυστική κατεύθυνση έρευνας θα ήταν ο σχεδιασμός αυτοσταθεροποιούμενων αλγορίθμων για να προσεγγίσουμε ένα MDST. Τέλος, ένας κατανεμημένος αλγόριθμος έχει προταθεί από τον Lavault και λουπούς ("A distributed approximation algorithm for the minimum degree minimum weight spanning trees", C.Lavault, M.Valencia-Pabon) για την επίλυση του ακόλουθου bicriteria προβλήματος: Κατασκευάστε ένα minimum spanning tree βαθμού το πολύ $b\Delta^* \log_b n$, για κάθε σταθερά $b > 1$, όπου Δ^* είναι ο μέγιστος βαθμός της βέλτιστης λύσης. Θα ήταν ενδιαφέρον η σχεδίαση ενός αυτοσταθεροποιούμενου αλγορίθμου για την επίλυση του δοσμένου αυτού προβλήματος.

Κεφάλαιο 4°

4.1 Περιβάλλον Υλοποίησης - Shawn

Το περιβάλλον υλοποίησης του αλγορίθμου μας ήταν το Shawn.

Το shawn είναι ένα περιβάλλον open-source (BSD Lisence) διακριτό κατά την προσομοίωση, όπου η ανάπτυξη πραγματοποιείται σε C++. Στο περιβάλλον αυτό μπορούμε να αναπτύξουμε εύκολα high-level αλγορίθμους και κατανεμημένα πρωτόκολλα. Επίσης μπορούμε να εξομοιώσουμε πολύ μεγάλα δίκτυα (100κ κόμβοι).

[4], [5]

Πιο Αναλυτικά:

- Το shawn είναι ένα *Συνεχές μοντέλο*. Δηλαδή οι μεταβλητές κατάστασης αλλάζουν με ένα συνεχή τρόπο, και όχι απότομα από τη μία κατάσταση σε μια άλλη (άπειρο αριθμό καταστάσεων).
- Υπάρχει *Συνεχής προσομοίωση* όσο αφορά την μοντελοποίηση κατά την πάροδο του χρόνου ενός συστήματος από μια αναπαράσταση σε μεταβλητές κατάστασης που μεταβάλλονται συνεχώς ως προς τον χρόνο.
- Συνήθως, χρησιμοποιούμε διαφορικές εξισώσεις, που δίνουν σχέσεις για τα ποσοστά μεταβολής των καταστάσεων μεταβλητών με το χρόνο. Έτσι χρησιμοποιούμε τα *Μοντέλα προσομοίωσης*.
- Είναι αναμφισβήτητα ένα *Διακριτό* περιβάλλον.

Ο όρος *διακριτό μοντέλο* αναφέρεται στις μεταβλητές κατάστασης οι οποίες αλλάζουν μόνο σε συγκεκριμένα χρονικά σημεία.

Τα χρονικά σημεία αυτά είναι εκείνα κατά τα οποία πραγματοποιείται αλλαγή κατάστασης.

Κάθε γεγονός πραγματοποιείται σε μια στιγμή στο χρόνο και σηματοδοτεί μια αλλαγή κατάστασης στο σύστημα.

Η λειτουργία του συστήματος αναπαρίσταται ως χρονολογική σειρά γεγονότων.

Τα Βασικά Components του Shawn

Clock

Παρακολουθώντας την τρέχουσα ώρα προσομοίωσης.

Το ρολόι μπορεί να λειτουργήσει με δύο τρόπους:

Χρόνος κοπής σε διαστήματα, όπου η πρόοδος του μοντέλου πραγματοποιείται κάθε φορά από ένα σταθερό ποσό ανεξάρτητα από την απουσία εκδηλώσεων για την ορθή εκτέλεση.

Επόμενο Συμβάν, όπου αναπτύσσουμε το μοντέλο για το επόμενο γεγονός το οποίο θα εκτελεστεί ανεξάρτητα από το χρονικό διάστημα που απαιτείται. Έτσι ο χρόνος εδώ κινείται από το ένα γεγονός στο άλλο.

Event List

Η προσομοίωση διατηρεί τουλάχιστον μια λίστα, τη λίστα προσομοίωσης γεγονότων.

Παραθέτει τα γεγονότα για τα οποία εκκρεμεί ως αποτέλεσμα της προηγούμενης προσομοίωσης, μόνο στην περίπτωση που δεν έχουν προσομοιωθεί οι ίδιοι.

Ένα γεγονός περιγράφεται από την στιγμή κατά την οποία αυτό συμβαίνει και ένα είδος, το οποίο φέρει τον κωδικό που θα χρησιμοποιηθεί στην προσομοίωση για την περίπτωση αυτή.

Random-Number Generation

Η προσομοίωση πρέπει να δημιουργεί τυχαίες μεταβλητές διαφόρων ειδών, ανάλογα με το μοντέλο του συστήματος.

Μια προσομοίωση θα μπορούσε να επιστρέψει με την ίδια ακριβώς συμπεριφορά.

Statistics

Η προσομοίωση κρατά συνήθως ένα κομμάτι του συστήματος των στατιστικών στοιχείων, τα οποία αξιολογούν ποσοτικά τις πτυχές του ενδιαφέροντος.

Είναι σημαντικές για την επικύρωση του μοντέλου και την παροχή του μέτρων απόδοσης.

Ending Condition

Θεωρητικά ένα μοντέλο διακριτών συμβάντων προσομοίωσης θα μπορούσε να τρέξει για πάντα, όμως πότε θα σταματήσουμε?

Σε ένα συγκεκριμένο χρονικό διάστημα T.

Μετά από την επεξεργασία N αριθμό συμβάντων (γεγονότων).

Τέλος θα μπορούσαμε να τερματίσουμε την προσομοίωση για διάφορες άλλες συνθήκες.

Λογική Κινήτρων Προσομοίωσης

Ο κύριος βρόγχος των διακριτών γεγονότων προσομοίωσης είναι περίπου ο ακόλουθος:

Έναρξη

- Αρχικοποίηση της μεταβλητής τερματισμού κατάστασης σε FALSE.
- Αρχικοποίηση μεταβλητών κατάστασης του συστήματος.
- Προετοιμασία ρολογιού/Clock (συνήθως ξεκινά από το μηδενικό χρόνο προσομοίωσης).
- Προγραμματισμός ενός αρχικού γεγονότος (θέτουμε κάποια αρχική εκδήλωση στην Event List).

Έως ότου η συνθήκη τερματισμού κατάστασης παραμένει FALSE, εκτελέστε τα ακόλουθα:

- Ρυθμίστε το ρολόι για το επόμενο γεγονός.
- Εκτελέστε το επόμενο γεγονός και αφαιρέστε το από την Event List.
- Ενημέρωση στατιστικών.

Σχεδιαστικοί Στόχοι

Όπως αναφέραμε και πιο πάνω, το shawn είναι ένα περιβάλλον open-source (BSD Licence) διακριτό κατά την προσομοίωση, γραμμένο σε C++. Παρέχει Υποστήριξη για την ανάπτυξη και τη δοκιμή High Level αλγορίθμων και πρωτοκόλλων που διανέμεται.

Μπορεί να προσομοιώσει πολύ μεγάλα δίκτυα (>100κ κόμβοι).

Οι προσομοιωτές γενικά ξοδεύουν αρκετό χρόνο επεξεργασίας σε αποτελέσματα τα οποία δεν παρουσιάζουν ενδιαφέρον για τον αλγόριθμο. Χαρακτηριστικό παράδειγμα αποτελούν η προσομοίωση φυσικών φαινομένων, όπως ραδιοφωνικό σήμα χαρακτηριστικά διάδοσης και ISO/OSI επιπέδου πρωτόκολλα, π.χ. Media Access Control (MAC).

Μία κεντρική προσέγγιση του Shawn είναι η εξομοίωση του αποτελέσματος που προκαλείται από ένα φαινόμενο, όχι το φαινόμενο του συστήματος.

Το Shawn αντικαθιστά χαμηλού επιπέδου επιπτώσεις με γενικά και ανταλλάξιμα μοντέλα, έτσι η προσομοίωση μπορεί να χρησιμοποιηθεί για τεράστια δίκτυα σε εύλογο χρονικό διάστημα.

Για παράδειγμα, αντί για την προσομοίωση ενός πλήρους επιπέδου MAC, συμπεριλαμβάνοντας το ραδιοφωνικό μοντέλο διάδοσης, οι επιδράσεις των αποτελεσμάτων αυτών (δηλαδή η απώλεια πακέτων και τα διάφορα σφάλματα) είναι μοντελοποιημένες στο Shawn.

Επεκτασιμότητα & Πεδίο Εφαρμογής

Όσο αφορά το *πεδίο εφαρμογής* του Shawn διακρίνεται σε πεδίο εφαρμογής για τον αριθμό των προσομοιωτών και σε πεδίο εφαρμογής του επιπέδου αφαίρεσης.

Τμήμα του πεδίου εφαρμογής του επιπέδου αφαίρεσης αποτελεί η προσομοίωση φυσικών φαινομένων, το χαμηλότερο επίπεδο του OSI/ISO πρωτοκόλλου και η αλγοριθμική εφαρμογή επιπέδου.

Όσο αφορά την επεκτασιμότητα του shawn αυτή είναι ο αριθμός των κόμβων του δικτύου.

Πρωτόκολλο & Κύκλος Ανάπτυξης

Για το πρωτόκολλο θα επικεντρωθούμε στην γρήγορη εκδοχή. Δηλαδή θα δούμε τα πρώτα αποτελέσματα και τις εντυπώσεις των επιδόσεων αυτών.

Για την αποκεντρωμένη απλοποιημένη εκδοχή, θα αποδείξουμε ότι τρέχει με ένα αποκεντρωμένο τρόπο και παράγει έγκυρα αποτελέσματα.

Για την πλήρης κατανεμημένη εκτέλεση, θα μετρήσουμε την απόδοση του αριθμού των μηνυμάτων, την κατανάλωση ενέργειας, τον χρόνο εκτέλεσης και την ανθεκτικότητα στην απώλεια μηνυμάτων.

Αρχιτεκτονική

Η αρχιτεκτονική του Shawn απαρτίζεται από τρία βασικά τμήματα, τα Sequencer, Model και Simulation Environment.

Sequencer

Αποτελεί την κεντρική συντονιστική μονάδα. Ρυθμίζει τις παραμέτρους της προσομοίωσης, εκτελεί τις εργασίες και ελέγχει την εξομοίωση.

Model

Το τμήμα αυτό ελέγχει και επιδρά στον τρόπο που επικοινωνούν οι κόμβοι μεταξύ τους.

Simulation Environment

Το περιβάλλον προσομοίωσης συγκρατεί τον εικονικό κόσμο στον οποίο κατοικούν οι αισθητήρες των κόμβων.

Models

Τρία μοντέλα αποτελούν τα θεμέλια του Shawn, αυτά είναι το Communication Model, το Edge Model και το Transmission Model .

Υπάρχουν επίσης κάποια μοντέλα για τον έλεγχο της συμπεριφοράς για την προσομοίωση κάποιας συγκεκριμένης διεργασίας.

Το Shawn διακρίνει τα μοντέλα μεταξύ τους και τις αντίστοιχες υλοποιήσεις τους, σε τρεις γενικές κατηγορίες:

- Απλοποιημένη και γρήγορη εφαρμογή.
- Πλησιέστερα στις πραγματικές υλοποιήσεις.
- Το Shawn διατηρεί μια αποθήκη πολλών διαφορετικών μοντέλων υλοποίησης.

Communication Model

Παρέχει μεθόδους για να διαπιστωθεί αν ο κόμβος B βρίσκεται σε κάποια απόσταση από τον κόμβο A.

Με την εφαρμογή της διεπαφής αυτής η οποία ορίζεται από το χρήστη όσο αφορά τον κώδικα, αυθαίρετα μοτίβα επικοινωνίας μπορούν να χρησιμοποιηθούν.

Παρέχει ένα σύνολο διαφορετικών εφαρμογών επικοινωνίας όπως είναι το μοντέλο UDG, Q-UDG, RIM. Πιο αναλυτικά:

UDG (Quasi-Unit Disk Graph): Στο πρωτόκολλο αυτό δυο κόμβοι μπορούν να επικοινωνούν αμφίδρομα αν η Ευκλείδεια απόσταση d μεταξύ των κόμβων είναι μικρότερη από r_{max} .

Q-UDG (Μονάδα Γράφημα Disk): Για $0 < d < r_1$ και $d > r_2$, η συμπεριφορά είναι ισοδύναμη με αυτή του UDG μοντέλου. Για $r_1 < d < r_2$, η πιθανότητα λήψης πακέτου μειώνεται γραμμικά από 1 σε 0.

RIM (Radio Model παρατυπία): Η γωνία του εύρους εξαρτάται μεταξύ ενός ελάχιστου και ενός μέγιστου σημείου της εμβέλειας επικοινωνίας.

Edge Model

Το μοντέλο αυτό παρέχει μια γραφική απεικόνιση του δικτύου.

Οι κόμβοι οι οποίοι λαμβάνουν μέρος στην προσομοίωση είναι ακμές του γραφήματος.

Μια ακμή μεταξύ δυο κόμβων, προστίθεται κάθε φορά που το μοντέλο επικοινωνίας (communication model) επιστρέφει true.

Το μοντέλο αυτό χρησιμοποιείται από το Shawn για το δυναμικό προσδιορισμό του παραλήπτη με την επανάληψη στους γείτονες του αποστέλλοντος κόμβου.

Μέσω του μοντέλου αυτού, οι κεντρικοποιημένοι αλγόριθμοι μπορούν να πάρουν άμεσα τη λίστα των γειτονικών τους κόμβων χωρίς ανταλλαγή μηνυμάτων.

Το Edge Model διακρίνεται σε τρία διαφορετικά είδη, τα οποία είναι τα ακόλουθα:

- *Lazy Model Edge (Απλό):* Το μοντέλο αυτό δεν αποθηκεύει οποιαδήποτε πληροφορία. Χρησιμοποιείται κυρίως για μικρές προσομοιώσεις με λίγη επικοινωνία ή για υψηλή κινητικότητα.
- *Edge List Model:* Στο μοντέλο αυτό αποθηκεύεται το αρχικό πλήρες γράφημα. Όμως η αρχική αυτή κατασκευή είναι αρκετά χρονοβόρα. Αλλά για κάθε επόμενο κόμβο η επανάληψη της διαδικασίας είναι ταχύτερη.
- *Edge Grid Model:* Εδώ χρησιμοποιούμε ένα πλέγμα δυο διαστάσεων για την ταξινόμηση των κόμβων, σύμφωνα με τη γεωμετρική τους θέση. Με τη χρήση του μοντέλου αυτού πετυχαίνουμε βελτίωση της ταχύτητας αναζήτησης αλλά και πλήρη υποστήριξη κινητικότητας.

Transmission Model

Το μοντέλο αυτό λειτουργεί ως ακολούθως. Κάθε φορά που ένας κόμβος μεταδίδει ένα μήνυμα, η συμπεριφορά του καναλιού μετάδοσης μπορεί να είναι εντελώς διαφορετική από οποιοδήποτε μήνυμα είχε μεταδοθεί νωρίτερα. Το μοντέλο αυτό επιτρέπει διαγώνια κίνηση από άλλους κόμβους καθώς και μετακίνηση κόμβων.

Το μοντέλο μετάδοσης μπορεί να καθυστερήσει αυθαίρετα ή να τροποποιήσει μηνύματα.

Η επιλογή μιας εφαρμογής εξαρτάται σε μεγάλο βαθμό από τον αρχικό στόχο προσομοίωσης.

Εργασίες Προσομοίωσης

Οι εργασίες προσομοίωσης είναι τμήματα του κώδικα τα οποία καλούνται από τη διαμόρφωση της προσομοίωσης που παρέχονται από τον χρήστη. Οι εργασίες αυτές δεν συνδέονται άμεσα με την εφαρμογή προσομοίωσης αλλά έχουν πρόσβαση σε όλο το περιβάλλον προσομοίωσης.

Οι εργασίες αυτές χρησιμοποιούνται, παραδείγματος χάρη στις παρακάτω περιπτώσεις:

Στη διαχείριση προσομοιώσεων, για τη συλλογή δεδομένων από μεμονωμένους κόμβους (`dump_transmission_stats`), για το τρέξιμο κεντριοποιημένων αλγορίθμων, για τη δημιουργία και την παραμετροποίηση των νέων κόσμων προσομοίωσης (`prepare_world edge_model=simple comm_model=disk_graph range =2 rect_world width=N height=M count=K processor=flood_the_worldr`) ακόμα και για την πραγματική προσομοίωση η οποία ενεργοποιείται με μια εργασία (`max_iterations` προσομοίωσης = `N`).

Sequencer

Το τμήμα του Sequencer είναι το κέντρο ελέγχου της προσομοίωσης, διότι προετοιμάζει τον κόσμο στον οποίο βρίσκονται οι προσομοιωμένοι κόμβοι καθώς και επειδή υποδεικνύει και παραμετροποιεί τις υλοποιήσεις των μοντέλων, όπως ορίζονται από τη διαμόρφωση εισόδου και τον έλεγχο της προσομοίωσης.

Το τμήμα του Sequencer απαρτίζεται από τρία τμήματα:

- Εργασίες Προσομοίωσης (Simulation Tasks).
- Ελεγκτής Προσομοίωσης (Simulation Controller).
- Event Scheduler.

Κόμβοι & Επεξεργαστές

Η προσομοίωση κόμβων διαμένει σε ένα single World instance. Οι ίδιοι οι κόμβοι χρησιμεύουν ως “δοχεία” ώστε να τους αποκαλούμε επεξεργαστές. Οι προγραμματιστές που χρησιμοποιούν το Shawn αναπτύσσουν την εφαρμογή τους λογικά ως τμήματα των επεξεργαστών αυτών.

Πολλαπλές εφαρμογές μπορούν να συνδυαστούν εύκολα σε μια ενιαία προσομοίωση .

Processor’s API:

- void boot();
- bool process_message(MessageHandle&);
- void work();
- Node & owner();
- void send(MessageHandle&);

Event Scheduler

Το Shawn οργανώνει λογικά την προσομοίωση σε γύρους ($r=0,1,2,\dots,n$) . Ο χρήστης μπορεί να καταχωρήσει Καθήκοντα Προσομοίωσης ως προ-βήμα και μετά-βήμα εργασιών οι οποίες εκτελούνται αμέσως πριν. Στην αρχή κάθε γύρου γίνεται επίκληση της μεθόδου *work()* κάθε κόμβου. Επίσης οι κόμβοι ή τα στοιχεία προσομοίωσης μπορούν να εγγραφούν ένα γεγονός σε οποιοδήποτε σημείο εγκαίρως.

Αισθητήρες Μοντελοποίησης & Αναγνώσεις

Τα μοντέλα αισθητήρων και οι αντίστοιχες αξίες αισθητήρων τους, είναι ένα γενικό πλαίσιο το οποίο παρέχεται και ονομάζεται αναγνώσεις και αισθητήρες. Οι αναγνώσεις παραδίδουν τις εξαρτώμενες θέσεις και τις χρονικά εξαρτώμενες τιμές οι οποίες μπορούν αυθόρμητα να πληκτρολογηθούν. Οι αισθητήρες συνδέονται σε ένα συγκεκριμένο κόμβο αισθητήρα και παρέχουν ενδείξεις ανάγνωσης αισθητήρα. Αναγνώσεις και αισθητήρες μπορούν να επικυρωθούν από τον χρήστη και γίνεται παραπομπή στο εσωτερικό της προσομοίωσης χρησιμοποιώντας μοναδικά ονόματα.

Μελέτη Περίπτωσης

Συγκρίνοντας το περιβάλλον εξομοίωσης του Shawn με αυτό του ns2, βγάζουμε τα ακόλουθα συμπεράσματα:

- Οι προσομοιώσεις εκτελούνται σε μια υπορουτίνα που χρησιμοποιείται σε συγκεκριμένο χρονικό διάστημα, χρησιμοποιώντας πρωτόκολλα

συγχρονισμού. Πιο συγκεκριμένα, κάθε κόμβος μεταδίδει περιοδικά ένα μήνυμα το οποίο περιέχει time stamps τα οποία μετατρέπουν τον κόμβο σε receiving node.

- Ένα σύνολο 380 μηνυμάτων αποστέλλονται από κάθε κόμβο. Αυτό προσφέρει μια εικόνα για την ικανότητα του προσομοιωτή να αποστείλει ένα μεγάλο μέρος της κυκλοφορίας.
- Περιβάλλον αποτελείται από μια τετράγωνη περιοχή του οποίου το μέγεθος είναι καθορισμένο από το πολλαπλάσιο του εύρους επικοινωνίας των κόμβων του.

Αντιθέτως το περιβάλλον υλοποίησης του ns2:

- Έχει πολύ πιο λεπτομερείς υπολογισμούς από Shawn προκειμένου να επιτευχθούν τα ίδια αποτελέσματα.
- Έχουμε διαφορά στις μετρήσεις απόδοσης όσο αφορά τη διάρκεια αλλά και τη χρήση μνήμης.
- Η μεταβλητή node density περιγράφει τον μέσο αριθμός των κόμβων σε μια περιοχή εκπομπής, δηλαδή την πυκνότητα των κόμβων.
- Το ns-2 hits το one-day barrier για περιπτώσεις που Shawn τελειώνει σε
- λιγότερο από ένα λεπτό.

Απόδοση Συστήματος Shawn

Η Προσομοίωση μιας απλής εφαρμογής μεταδίδει ένα μήνυμα κάθε 250ms της προσομοίωσης του χρόνου. Η εμβέλεια επικοινωνίας των κόμβων αισθητήρων έχει οριστεί σε 50 μονάδες μήκους και κάθε προσομοίωση τρέχει για 60 μονάδες προσομοίωσης χρόνου. Το μέγεθος της προσομοιωμένης περιοχής είναι 500x500 μονάδες μήκους.

Το *Shawn* ξεπερνά τα δύο άλλα εργαλεία προσομοίωσης με τάξεις μεγέθους.

Το *ns-2* εκτελεί μια πολύ λεπτομερή προσομοίωση των χαμηλότερων στρώματων, όπως η σωματική και το στρώμα ζεύξης δεδομένων, ενώ το *Shawn* απλά αποστέλλει τα μηνύματα χρησιμοποιώντας ένα απλοποιημένο μοντέλο.

Το περιβάλλον υλοποίησης *TOSSIM* που προσομοιώνει ένα υποκείμενο *TinyOSsupported* της πλατφόρμας υλικού.

4.2 Θέματα Υλοποίησης

Οι σχεδιαστικές υποθέσεις υλοποίησης τις οποίες χρησιμοποιήσαμε κατά την σύγχρονη υλοποίηση του αλγορίθμου μας σε πρακτικό επίπεδο, για οποιαδήποτε τοπολογία είναι οι ακόλουθες:

Για κάθε μια από τις διαφορετικές τοπολογίες για τις οποίες τρέξαμε τον αλγόριθμο μας δημιουργήσαμε ένα τρισδιάστατο πίνακα με όνομα *Neighborhood* ο οποίος περιέχει στην πρώτη στήλη κάθε μια από τις διεργασίες του εκάστοτε δικτύου, στην επόμενη στήλη τις γειτονικές για αυτή διεργασίες και στην τρίτη το βάρος της μεταξύ τους ακμής. Επομένως με βάση αυτό τον αρχικό πίνακα της τοπολογίας δημιουργούνται και όλοι οι υπόλοιποι που απαιτούνται για την ολοκλήρωση του αλγορίθμου.

Αρχικά κάθε κόμβος επεξεργαστής γνωρίζει τα βάρη των παρακείμενων ακμών του, δημιουργώντας τοπικούς πίνακες για κάθε έναν από αυτούς με όνομα *my_neighbor*. Οι τοπικοί αυτοί πίνακες είναι δύο διαστάσεων όπου στην πρώτη στήλη βρίσκεται η ταυτότητα του κόμβου με τον οποίο γειτνιάζει ο εκάστοτε επεξεργαστής και στη δεύτερη περιέχεται το βάρος της κοινής ακμής των δυο αυτών επεξεργαστών.

Υποθέτουμε πως κάθε ένας επεξεργαστής όταν ενεργοποιείται αποτελεί ένα ξεχωριστό *fragment* με ταυτότητα *fragment* (*id_fragment*) ίση με την ταυτότητα του, μιας και χρησιμοποιούμε μοναδικά αναγνωριστικά. Έτσι σταδιακά τα ξεχωριστά *fragments* – κόμβοι συνδέονται μεταξύ τους έως ότου λάβουμε ένα μόνο *fragment*, το οποίο θα είναι και το MST της τοπολογίας μας.

Κάθε κόμβος εκτός από τον αρχικό κόμβο (για την υλοποίηση μας είναι ο κόμβος V_0) ο οποίος ξεκινά τον αλγόριθμο, ενεργοποιείται μόλις λάβει μήνυμα αρχικοποίησης (*initiate message*) από κάποιο γείτονα κόμβο. Τότε έχοντας ήδη τον τοπικό πίνακα ώστε να γνωρίζει τις γειτονικές του διεργασίες και τα βάρη των μεταξύ τους ακμών ξεκινά την κατηγοριοποίηση όλων των παρακείμενων ακμών του. Αυτό γίνεται προκειμένου κάθε διεργασία να γνωρίζει τις εξερχόμενες ακμές της και έτσι να επιλέξει να συνδεθεί με την εξερχόμενη ακμή που έχει το ελάχιστο βάρος. Δημιουργήσαμε λοιπόν τρεις ακόμη τοπικούς πίνακες για κάθε μια διεργασία για την κατηγοριοποίηση αυτή, οι οποίοι είναι:

- *my_merge*: Ο πίνακας αυτός περιέχει τις εξερχόμενες ακμές της διεργασίας οι οποίες μπορούν να χρησιμοποιηθούν για επιμέρους συγχώνευση διαφορετικών *fragments*. Η πρώτη ακμή που περιέχει ο πίνακας αυτός για κάθε διεργασία είναι η ακμή που ενώνει τη διεργασία αυτή με τη διεργασία που την ενεργοποίησε.

- *my_basic*: Εδώ περιέχονται όλες οι εξερχόμενες ακμές μέσω των οποίων μπορώ να συνενωθώ με τις γειτονικές μου διεργασίες και να αποτελέσουμε μαζί ένα νέο *fragment*. Από τον πίνακα αυτό κάθε διεργασία επιλέγει τη μικρότερη ακμή και στέλνει μήνυμα συνένωσης στον παρακείμενο σε αυτή κόμβο.

Στέλνουμε επομένως *connect message* στον γείτονα κόμβο με τον οποίο μας συνδέει η ακμή με το μικρότερο βάρος.

- *my_branch*: Κάθε διεργασία εδώ αποθηκεύει τις ακμές με τις οποίες έχει συνδεθεί μαζί τους. Ουσιαστικά δηλαδή στον πίνακα αυτόν κρατάμε όλες τις ακμές των επιμέρους *fragments*, και τελικά τις ακμές του ζητούμενου *fragment* (τελικού *MST*) .

Επομένως, σύμφωνα με όλα τα παραπάνω όταν μια διεργασία ενεργοποιηθεί στέλνει *initiate messages* σε όλες τις γειτονικές της σύμφωνα με τον τοπικό της πίνακα *my_neighbor*.

Έπειτα από τον πίνακα *my_basic* επιλέγει την ακμή με το μικρότερο για αυτήν βάρος και στέλνει *connect message* στην διεργασία η οποία έχει κοινή την ακμή αυτή, προκειμένου να συνδεθούν και να αποτελέσουν ένα κοινό *fragment*.

Επίσης αφού επιλέξει την ακμή αυτή στέλνει και μήνυμα *report message* στην διεργασία γονέα της προκειμένου να την ενημερώσει για την προσθήκη της διεργασίας αλλά και της ακμής αυτής στο *fragment*. Κατά τη διάρκεια ανταλλαγής αυτών των μηνυμάτων μεταξύ των διεργασιών ενημερώνονται ταυτόχρονα όλοι οι παραπάνω τοπικοί πίνακες για κάθε αλλαγή που πραγματοποιείται στα επιμέρους *fragments*.

Υποθέτουμε ακόμη πως τα μηνύματα για τη συνένωση των επιμέρους (*test messages*) *fragment* ανταλλάσσονται μεταξύ των κόμβων οι οποίοι αποτελούν ρίζα στο *fragment* το οποίο ανήκουν, δηλαδή δεν έχουν γονέα αλλά είναι η ίδια γονέας. Έτσι όταν κάποια διεργασία αποτελεί ρίζα στο *fragment* το οποίο ανήκει σύμφωνα με τον τοπικό της πίνακα *my_merge* θα στείλει *test message* στον κόμβο της ακμής με το μικρότερο βάρος από τις πιθανές ακμές.

Τώρα αν η διεργασία που θα λάβει το μήνυμα αυτό είναι επίσης ρίζα στο *fragment* στο οποίο ανήκει αλλά και η μεταξύ τους ακμή είναι και για εκείνη, σύμφωνα με τον πίνακα *my_merge*, αυτή με το μικρότερο βάρος θα απαντήσει με ένα *accept message* στην διεργασία αυτή και θα έχουμε συνένωση επιμέρους *fragment* με νέα ταυτότητα ίση με το βάρος της κοινής ακμής.

Διαφορετικά, δηλαδή αν η διεργασία η οποία έλαβε το *test message* δεν αποτελεί ρίζα στο *fragment* το οποίο ανήκει ή η κοινή ακμή δεν είναι και για εκείνη αυτή με το ελάχιστο βάρος, θα απαντήσει με ένα *reject message* στη διεργασία που της έστειλε *test message*. Ενώ ταυτόχρονα θα αποστείλει μήνυμα *change core message* στη διεργασία στην οποία η μεταξύ τους ακμή είναι αυτή με το ελάχιστο βάρος, ενώ η προσκείμενη διεργασία δεν αποτελεί ρίζα στο *fragment* στο οποίο ανήκει. Έτσι θα έχουμε αλλαγή ρίζας για το *fragment* αυτό προκειμένου να επιτευχθεί συνένωση δυο επιμέρους *fragment* μεταξύ των ριζών τους μέσω της κοινής ακμής ελαχίστου βάρους.

Τέλος, ως συνθήκη τερματισμού για την κάθε διεργασία θεωρούμε να έχει ταυτότητα του *fragment* στο οποίο ανήκει (*id_fragment*) την τελική ταυτότητα, δηλαδή το βάρος της ακμής μέσω της οποίας ενώθηκαν τα δυο επιμέρους *fragment* της αρχικής μας τοπολογίας. Έτσι ο τερματισμός των

διεργασιών μας ξεκινά από τα φύλλα του δένδρου και τους ενδιάμεσους κόμβους έως ότου καταλήξει στους κόμβους πυρήνες (core) του τελικού μας fragment.

Σημαντική παρατήρηση στο σημείο αυτό αποτελεί η χρονική στιγμή κατά την οποία ελέγχουμε κατά πόσο χρειάζεται να πραγματοποιηθεί merge των επιμέρους fragments. Έπειτα από παρατήρηση κατά το τρέξιμο του αλγορίθμου μας καταλήξαμε ότι σύμφωνα με το συγχρονισμό του αλγορίθμου που υλοποιήσαμε και καθώς οι τοπικοί μας πίνακες διαμορφώνονται μέσω της ανταλλαγής των διαφόρων αυτών μηνυμάτων, δεν υπάρχει λόγος να γίνει ο έλεγχος αυτός πριν τον 4^ο γύρο. Θεωρητικά και αφού κάθε κόμβος αρχικά αποτελεί για την υλοποίηση μας ένα ξεχωριστό fragment θα μπορούσε ο έλεγχος αυτός να γίνεται σε κάθε γύρο, αφού σκοπός μας είναι να μην υπάρχουν επιμέρους fragment αλλά μόνο ένα. Από άποψη όμως πολυπλοκότητας μηνυμάτων κάτι τέτοιο δεν θα συνέφερε καθόλου καθώς δεν επηρεάζει καθόλου την εκτέλεση του αλγορίθμου και το συνδυασμό των επιμέρους fragment του τελικού αποτελέσματος. Θα ήταν όμως και άσκοπο μιας και οι τοπικοί πίνακες για κάθε διεργασία αρχικά θα ήταν κενοί. Για την ελαχιστοποίηση και μόνο λοιπόν του κόστους επικοινωνίας και για πρακτικούς λόγους ορίζουμε τον έλεγχο για την συγχώνευση των επιμέρους fragment να πραγματοποιείται ανά 4 γύρους. Είναι μια παρατήρηση η οποία εντοπίστηκε κατά την πειραματική εκτέλεση του αλγορίθμου προκειμένου να επιτύχουμε τη βέλτιστη υλοποίηση αυτού.

Τα τμήματα του κώδικα που υλοποιούνται τα παραπάνω είναι τα ακόλουθα:

```
void GHSPProcessor::boot( void )
throw()
{
    node = id();
    int r1=0;
    int r2=0;

    //int Neighborhood[6][3] = {{0, 1, 12},{0, 4, 13},{1, 4, 18},{1, 2, 14},{2, 3,
    23},{3, 4, 15}};
    int Neighborhood[7][3] = {{0, 1, 22},{0, 5, 27},{1, 2, 21},{1, 5, 24},{2, 3,
    25},{4, 3, 26},{4, 5, 23}};

    const          shawn::SimulationEnvironment&          se          =
    owner().world().simulation_controller().environment();
    diam = se.optional_int_param("diam",0);

    j=0;
    for(int k=0;k<7;k++){
        for(int l=0;l<3;l++)
        {
            if(Neighborhood[k][l] == id())
            {
                if (l == 0)
                {
                    my_neighbor[j][0]=Neighborhood[k][1];
                    my_neighbor[j][1]=Neighborhood[k][2];
                    j++;
                }
                else
                {
                    my_neighbor[j][0]=Neighborhood[k][0];
                    my_neighbor[j][1]=Neighborhood[k][2];
                    j++;
                }
            }
        }
    } // telos dhmiourgias topikwn pinakwn
```

Έπειτα από τη δημιουργία των τοπικών πινάκων αρχίζει η ανταλλαγή μηνυμάτων μεταξύ των διεργασιών.

```

void GHSPProcessor::work(void)
throw()
{
    int r=0;
    if( valid != -1 && test_flag != 1)
    {
        send (new GHSConnectMessage(node, parent, neighbor,
id_fragment, level));
        send_connect_flag = 1;

        if(level < diam)
        {
            level++;
            if( parent != -1 && level_flag != 1)
            {
                send( new GHSreportMessage (minimum, parent,
level, id_fragment));
                level_flag = 1;
            }
        }
        //apo8hkevw tis akmes tou fragment
        my_branch[r][0]=neighbor;
        my_branch[r][1]=minimum;

        // enimerwnw ti lista basic, diagraftontas tin akmi pou pleon einai akmi tou
        fragment!

        for(int k=0;k<5;k++){
            for(int l=0;l<2;l++)
            {
                if(my_basic[k][0] == neighbor &&
my_basic[k][1] == minimum)
                {
                    my_basic[k][0]=0;
                    my_basic[k][1]=0;
                }
            }
        }

        branch_flag=1;
        r++;
        valid = -1;
    }
    if( id() == 0 && simulation_round() == 1 )
    {
        id_fragment=0;
        level=0;
        init_level=0;
        sleeping=false;
        find=true;
        init_flag = 1;
    }
}

```

```

//----- apostolh init message -----//
    if (find == true && init_flag == 1 && connect_flag != 1)
    {
        send( new GHSinitiateMessage(node, id_fragment, level));
        connect_flag = 1;
    }

// ----- apostolh connect message-----//
    if (find == true && connect_flag == 1 && init_flag == 1)
    {
        int r=0;
        minimum = 100;
        for(int m=0;m<5;m++){
            for(int n=0;n<2;n++)
            {
                if(my_basic[m][1] < minimum &&
my_basic[m][1] != 0)
                {
                    minimum=my_basic[m][1];
                    neighbor=my_basic[m][0];
                }
                if(id() == 0 && my_basic[m][1] ==
minimum)
                {
                    my_branch[0][0]=my_basic[m][0];
                    my_branch[0][1]=my_basic[m][1];
                    cout << "my_branch0=" <<
my_branch[m][1] << endl;

                    branch_flag=1;

                    // diagraw tin akmh auth apo
                    ton my_basic

                    my_basic[m][0]=0;
                    my_basic[m][1]=0;
                }
                if(id() == 0 && my_basic[m][1] != minimum
&& my_basic[m][1] != 0)
                {
                    my_merge[r][0]=my_basic[m][0];
                    my_merge[r][1]=my_basic[m][1];
                    cout << "my_merge0=" <<
my_merge[r][1] << endl;

                    r++;
                    merge_flag=1;

                    // diagraw tin akmh auth apo
                    ton my_basic

                    my_basic[m][0]=0;
                    my_basic[m][1]=0;
                }
            }
        }
    }
}

```

```

        if(minimum != 100 && count == 0)
        {
            valid = 1;
            cout << "minimum=" << "tou kombou v" << id() << "einai to" <<
minimum << endl;
            cout << "neighbor" << "tou kombou v" << id() << " einai o" <<
neighbor << endl;

            my_connect_list[0]= node;
            my_connect_list[1] = parent;
            my_connect_list[2] = neighbor;
            my_connect_list[3] = id_fragment;
            my_connect_list[4] = level;
            my_connect_list[5] = minimum;
            count++;
        }
    }
    // ----- send report message -----//
    if(find == true && send_connect_flag == 1 && count == 1)
    {
        cout << "o kombos v" << id() << " eginε Found, me
minimum akmh " << my_connect_list[5] << endl;
        // den stelnw report sthn periptwsh pou eimai riza!!
        if(parent != -1)
        {
            send( new GHSreportMessage (my_connect_list[5],
parent, level, id_fragment));
        }

        //end of 1st step - each node have found the minimum outgoing edge and send
report message!!

        found=true;
        send_report_message = 1;
    }
    // eleggxos gia sunenwsh fragments
    if(simulation_round()%4 == 0 && end_flag != 1)
    {
        /* gia opoiadh pote topologia exw sunenwsh duo fragment ka8e fora,
etsi kanw elegxo gia sunenwsh ana 4 gurous !!!! */

        if(found == true && parent == -1 && id_fragment !=
init_id_fragment )
        {
            for(int m=0;m<5;m++){
                for(int n=0;n<2;n++)
                {
                    if(my_merge[m][1] != 0 )
                    {
                        send(new
GHSmergeMessage(node, id_fragment , level, my_merge[m][0]));
                        test_flag = 1;
                    }
                }
            }
        }
    }
}

```

```

// enhmerwnw tous geitones mou gia teliko to id_fragment
if(simulation_round() > 2*diam+1 && id_fragment == best_edge
&& id_fragment != 0)
{
    for(int m=0;m<5;m++){
        for(int n=0;n<2;n++)
        {
            if(my_neighbor[m][1] != 0)
            {
                send( new GHSreportMessage
(best_edge, my_neighbor[m][0], level, id_fragment));
            }
        }
    }
}

// shn8hkh termatismou
if(simulation_round() > 2*diam+1)
{
    int edge=0;
    int edge2=0;
    for(int k=0;k<5;k++){
        for(int l=0;l<2;l++)
        {
            if(my_branch[k][1] != 0)
            {
                edge=my_branch[k][1];
            }
            if(my_merge[k][1] != 0)
            {
                edge2=my_merge[k][1];
            }
        }
    }

    if(found == true && simulation_round() > 2*diam+1 && parent !=
-1 && edge == 0 && edge2 == 0 && id_fragment == best_edge && id_fragment !=
0)
    {
        weight=1000;
        send( new GHSreportMessage (weight, parent, level,
id_fragment));

        cout << "oloklhpw8hke o algori8mos gia ti diergasia" <<
id() << endl;

        set_state(Inactive);
    }

    if(found == true && end_flag == 1 && id_fragment == best_edge
&& simulation_round() > 2*(2*diam+1) && id_fragment != 0)
    {
        weight=1000;
        send( new GHSreportMessage (weight, parent, level,
id_fragment));

        cout << "oloklhpw8hke o algori8mos gia ti diergasia" <<
id() << endl;

        set_state(Inactive);
    }
}
}

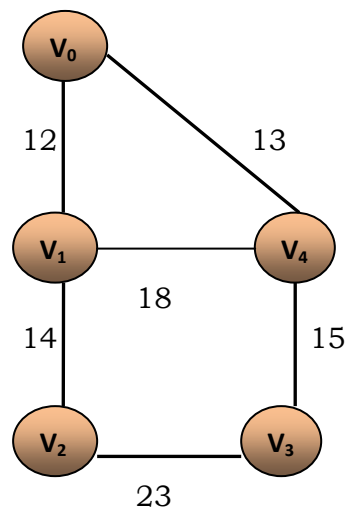
```

Έπειτα σε κάθε τύπο μηνύματος γίνονται οι κατάλληλες ενέργειες, ενημερώνονται οι τοπικοί πίνακες και οι γίνονται οι απαιτούμενες συνενώσεις κόμβων ή fragment, έως ότου ο αλγόριθμος μας ολοκληρωθεί με επιτυχία, έχοντας κατασκευαστεί το απαιτούμενο MST.

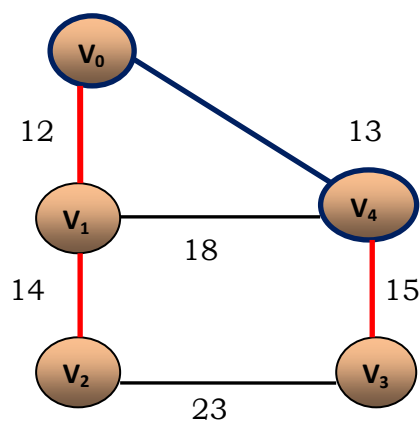
Παραδείγματα υλοποίησης:

Τρέξαμε τον αλγόριθμο μας για δυο διαφορετικές τοπολογίες και μελετήσαμε γύρο προς γύρο αλλά και μήνυμα προς μήνυμα την ορθή λειτουργία του λαμβάνοντας ως τελικό αποτέλεσμα το ζητούμενο MST. Οι δυο αυτές τοπολογίες είναι οι ακόλουθες:

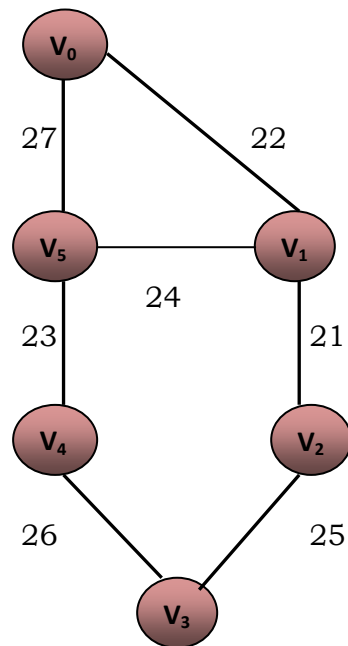
Η πρώτη τοπολογία μας είναι η ακόλουθη



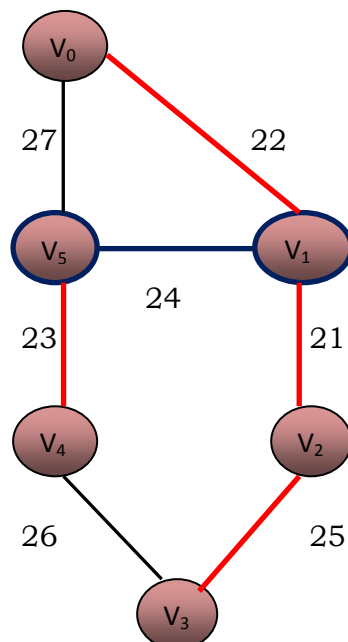
Και το τελικό MST που λάβαμε είναι το ακόλουθο, με πυρήνες τις διεργασίες κόμβους V_0 και V_4 και $id_fragment=13$.



Η δεύτερη τοπολογία μας καθώς και το τελικό MST αυτής είναι η ακόλουθη:



Στην τοπολογία αυτή πυρήνες του τελικού fragment αποτελούν οι διεργασίες V_5 και V_1 και το $id_fragment$ είναι ίσο με την κοινή ακμή των δυο αυτών κόμβων δηλαδή έχουμε $id_fragment=24$.



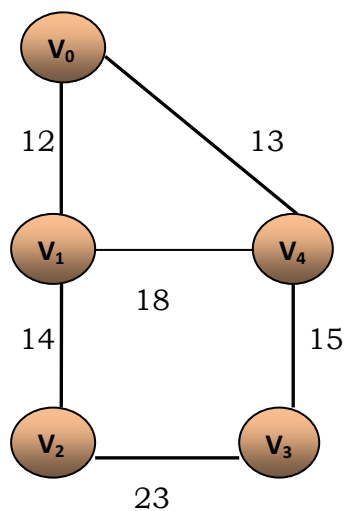
4.3 Πειραματική Αξιολόγηση

Όσο αφορά την πειραματική αξιολόγηση του αλγορίθμου τον οποίο κατασκευάσαμε, δουλέψαμε αναλυτικά στις δυο τοπολογίες που ακολουθούν και παρουσιάστηκαν και πιο πάνω.

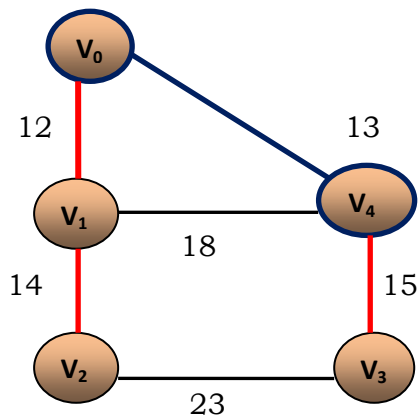
Πιο συγκεκριμένα παρακολουθήσαμε για κάθε μια από αυτές τις τοπολογίες για κάθε γύρο κάθε ανταλλαγή μηνύματος και το κατά πόσο το τελικό MST που είναι και το ζητούμενο κατασκευάζεται σύμφωνα με τις υποδείξεις του αλγορίθμου, ακολουθώντας τα χρονικά περιθώρια αλλά και το πλήθος μηνυμάτων που ορίζουν η χρονική πολυπλοκότητα και πολυπλοκότητα επικοινωνίας αντίστοιχα.

Ειδικότερα για κάθε τοπολογία λάβαμε τα ακόλουθα αποτελέσματα τα οποία επαλήθευσαν την ορθή λειτουργία του αλγορίθμου μας.

Για την πρώτη τοπολογία



Το MST που λάβαμε είναι το ακόλουθο, με πυρήνες τις διεργασίες κόμβους V_0 και V_4 .



Αρχικά ενεργή είναι η διαδικασία v_0 η οποία αποτελεί ένα ξεχωριστό fragment με `id_fragment` ίσο με την ταυτότητα της. Η διεργασία αυτή ξεκινά τον αλγόριθμο και στέλνοντας μηνύματα αρχικοποίησης στις γειτονικές της διεργασίες v_1 και v_4 οι οποίες ενεργοποιούνται και αυτές με τη σειρά τους και θεωρούνται ως ξεχωριστά fragment η κάθε μία ομοίως. Διαδοχικά έπειτα από την ανταλλαγή των μηνυμάτων αρχίζουν τα επιμέρους fragment (διεργασίες) να συγχωνεύονται έως ότου λάβουμε το τελικό αποτέλεσμα, δηλαδή ένα μόνο fragment το οποίο αποτελεί και το MST της τοπολογίας μας.

Για να κατασκευαστεί λοιπόν πλήρως το τελικό MST που λάβαμε για την τοπολογία αυτή χρειάστηκαν 11 γύροι και 169 μηνύματα. Τιμές πλήρως αποδεκτές καθώς σύμφωνα με τον αλγόριθμο μας η χρονική πολυπλοκότητα είναι στη χειρότερη περίπτωση $O(n \log N)$ ενώ η πολυπλοκότητα επικοινωνίας $5N \log_2 N + 2E$.

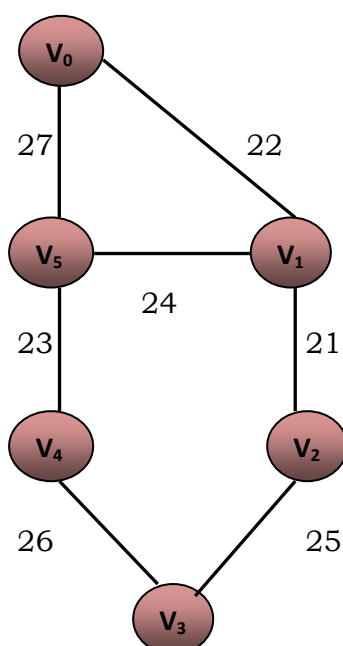
```

o1ok1hpw8hke o algori8mos gia ti diergasia4
----- DONE ITERATION 11
[ 0 active, 0 sleeping, 5 inactive ]

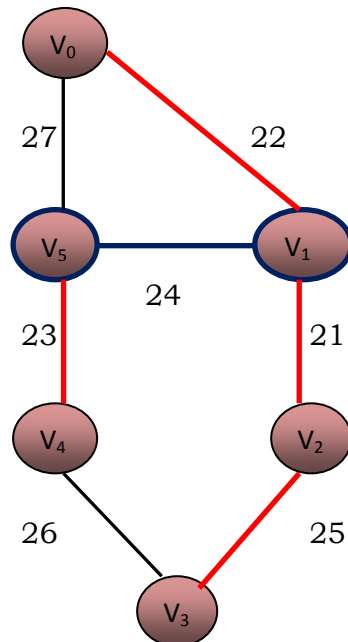
Simulation: Task done 'simulation'
Simulation: Running task 'dump_transmission_stats'
----- stats_chain transmission model information -----
general all_types messages 169
general all_types size 169
general N3GHS14GHS testMessageE messages 12
general N3GHS14GHS testMessageE size 12
general N3GHS16GHS acceptMessageE messages 2
general N3GHS16GHS acceptMessageE size 2
general N3GHS16GHS rejectMessageE messages 1
general N3GHS16GHS rejectMessageE size 1
general N3GHS16GHS reportMessageE messages 141
general N3GHS16GHS reportMessageE size 141
general N3GHS17GHS connectMessageE messages 8
general N3GHS17GHS connectMessageE size 8
general N3GHS18GHS initiateMessageE messages 5
general N3GHS18GHS initiateMessageE size 5
-----
Simulation: Task done 'dump_transmission_stats'

```

Η δεύτερη τοπολογία μας καθώς και το τελικό MST αυτής είναι η ακόλουθη:



Στην τοπολογία αυτή πυρήνες του τελικού fragment αποτελούν οι διεργασίες V_5 και V_1 και το `id_fragment` είναι ίσο με την κοινή ακμή των δυο αυτών κόμβων δηλαδή έχουμε `id_fragment=24`.



Όμοια και εδώ τον αλγόριθμο ξεκινά η διεργασία v_0 η οποία αποτελεί ένα ξεχωριστό fragment. Όπως και στο προηγούμενο παράδειγμα έπειτα από κάποιες ανταλλαγές μηνυμάτων τα ξεχωριστά fragment συνενώνονται σταδιακά έως ότου λάβουμε το τελικό αποτέλεσμα.

Εδώ όπως φαίνεται και πιο κάτω από την εικόνα που λάβαμε μετά την εκτέλεση του αλγορίθμου απαιτήθηκαν συνολικά 15 γύροι εκτέλεσης και έγινε ανταλλαγή 223 μηνυμάτων.

```

oloκληρω8hke o algori8mos gia ti diergasia5
----- DONE ITERATION 15
[ 0 active, 0 sleeping, 6 inactive ]

Simulation: Task done 'simulation'
Simulation: Running task 'dump_transmission_stats'
----- stats_chain transmission_model information -----
general all_types messages 223
general all_types size 223
general N3GHS14GHStestMessageE messages 12
general N3GHS14GHStestMessageE size 12
general N3GHS16GHSrejectMessageE messages 7
general N3GHS16GHSrejectMessageE size 7
general N3GHS16GHSreportMessageE messages 189
general N3GHS16GHSreportMessageE size 189
general N3GHS17GHSconnectMessageE messages 7
general N3GHS17GHSconnectMessageE size 7
general N3GHS18GHSinitiateMessageE messages 6
general N3GHS18GHSinitiateMessageE size 6
general N3GHS20GHSchangecoreMessageE messages 2
general N3GHS20GHSchangecoreMessageE size 2
-----
Simulation: Task done 'dump_transmission_stats'

```

Στο σημείο αυτό υπενθυμίζουμε ότι ο αλγόριθμος μας υποθέτει ότι κάθε τοπολογία έχει *διακριτά βάρη ακμών*, δηλαδή όλη η παραπάνω ανάλυση και κυρίως οι τιμές της χρονικής πολυπλοκότητας και της πολυπλοκότητας επικοινωνίας έχουν υπολογιστεί με βάση την υπόθεση αυτή.

Παρόλο αυτά ο αλγόριθμος μας μπορεί να λειτουργήσει και χωρίς διακριτά βάρη ακμών, αλλά με διακριτά αναγνωριστικά για κάθε μια από τις διεργασίες μας, δηλαδή η κάθε ακμή θα επισυνάπτει στο βάρος της τα αναγνωριστικά (ταυτότητες, *ids*) των δυο κόμβων που συνδέει. Σαν άμεση συνέπεια αυτού θα έχουμε επιβάρυνση της τάξης $2E$ επιπρόσθετων μηνυμάτων στις αναμενόμενες πολυπλοκότητες.

Όμως αν το δίκτυο δεν έχει ούτε διακριτά βάρη ακμών αλλά ούτε και διακριτά αναγνωριστικά σε κάθε κόμβο, τότε κανένας κατανεμημένος αλγόριθμος (από τους τύπους αλγορίθμων που περιγράψαμε στα προηγούμενα κεφάλαια) δεν υπάρχει για την εύρεση ενός MST με φραγμένο αριθμό μηνυμάτων.

Τρέξαμε επίσης τον αλγόριθμο μας για μια τυχαία τοπολογία 10 κόμβων και 12 ακμών.

Ακολουθεί ένας πίνακας με όλα τα αποτελέσματα εκτέλεσης τα οποία λάβαμε.

Κόμβοι (n)	Ακμές (E)	Χρονική Πολυπλοκότητα	Πολυπλοκότητα Επικοινωνίας
5	6	11 Γύροι	169 μηνύματα
6	7	15 Γύροι	223 μηνύματα
10	12	49 Γύροι	1052 μηνύματα

Συμπεράσματα:

Γνωρίζουμε ότι ο αλγόριθμος μας απαιτεί $O(n)$ γύρους εκτέλεσης για κάθε επίπεδο του δικτύου ή $O(n \log N)$ γύρους συνολικά. Έτσι αναμένουμε ανάλογα με τις διάφορες τοπολογίες που θα τρέξουμε τον αλγόριθμο αυτό οι γύροι να συγκλίνουν με το φράγμα αυτό.

Επίσης, όσο αφορά την πολυπλοκότητα επικοινωνίας αυτή είναι για τη χειρότερη περίπτωση ίση με $5N \log_2 N + 2E$. Επομένως και για την πολυπλοκότητα επικοινωνίας περιμένουμε να υπάρχει αύξηση του συνολικού αριθμού των μηνυμάτων ανάλογη με αυτή της αύξησης των κόμβων και των ακμών της κάθε τοπολογίας.

Οι δυο παρατηρήσεις αυτές βλέπουμε ότι από την σύγκριση των δύο παραπάνω τοπολογιών επαληθεύονται. Αν και οι τοπολογίες μας αυτές δεν έχουν και πολύ μεγάλες διαφορές στο πλήθος των κόμβων και των ακμών βλέπουμε όμως την ανάλογη αλλαγή στην πολυπλοκότητα επικοινωνίας αλλά και τη χρονική πολυπλοκότητα. Βεβαίως με πειραματισμό σε πολύ μεγαλύτερες τοπολογίες τα συμπεράσματα αυτά θα ήταν πιο εμφανή.

Κεφάλαιο 5°

5.1 Συμπεράσματα

Στην διπλωματική εργασία αυτή παρουσιάζεται ο κατανεμημένος αλγόριθμος GHS ο οποίος κατασκευάζει ένα δένδρο με ελάχιστο βάρος ακμών (ελάχιστο γεννητικό δένδρο MST) σε ένα συνδεδεμένο μη κατευθυνόμενο γράφο με διακριτά βάρη ακμών. Ένας επεξεργαστής υπάρχει σε κάθε κόμβο του γράφου, γνωρίζοντας αρχικά μόνο τα βάρη των παρακείμενων ακμών του. Οι επεξεργαστές υπακούουν στον ίδιο αλγόριθμο και ανταλλάσσουν μηνύματα με τους επεξεργαστές των γειτονικών κόμβων έως ότου το δένδρο κατασκευαστεί. Ο συνολικός αριθμός μηνυμάτων που απαιτείται για ένα γράφο N κόμβων και E ακμών είναι το λιγότερο $5N \log_2 N + 2E$. Ο αλγόριθμος μπορεί να αρχικοποιηθεί αυθόρμητα από οποιοδήποτε κόμβο ή από οποιοδήποτε υποσύνολο κόμβων. Κατά τη δική μας υλοποίηση ο αλγόριθμος ξεκινά πάντα από την αρχική διεργασία v_0 και κάθε διεργασία η οποία ενεργοποιείται θεωρείται αρχικά ως ένα fragment με `id_fragment` ίσο με την ταυτότητα της.

Η πορεία της κατασκευής του αλγορίθμου αυτού ώστε να λειτουργεί συγχρονισμένα στο κατανεμημένο περιβάλλον υλοποίησης μας, δηλαδή στο Shawn ήταν μια όμορφη εμπειρία για να κατανοήσουμε πόσο τρομακτικά διαφέρει η θεωρητική παρουσίαση του αλγορίθμου από την πρακτική του υλοποίηση. Οτιδήποτε θεωρείται θεωρητικά δεδομένο για τον αλγόριθμο μας πρέπει στην πρακτική υλοποίηση να καθορίζεται ρητά. Κατά αρχάς ο αλγόριθμος τον οποίο μελετήσαμε για να πραγματοποιήσουμε αυτή την κατασκευή αναφέρεται σε ασύγχρονα δίκτυα ενώ εμείς επιθυμούσαμε να κατασκευάσουμε τον κατανεμημένο αλγόριθμο μας να λειτουργεί συγχρονισμένα. Επίσης καλούμασταν να επιλέξουμε τον τρόπο κατά τον οποίο κάθε διεργασία θα είχε εξ αρχής γνώση των γειτονικών της διεργασιών αλλά και των βαρών των ακμών μεταξύ αυτών. Όλα αυτά ήταν μια διαδρομή γεμάτη εκπλήξεις και σε κάθε βήμα γινόταν όλο και πιο ξεκάθαρο πόσο δύσκολο είναι να αποδώσεις σωστά όλα αυτά που θεωρητικά περιγράφονται από τους ερευνητές που κατασκεύασαν τον αλγόριθμο.

Έγινε λοιπόν ξεκάθαρο μέσα από την πορεία μου αυτή πόσο τελικά σημαντική γνώση είναι η εύρεση του MST σε μια τοπολογία και όταν αυτή αναφέρεται σε πραγματικά δίκτυα πόσο μπορεί να ελαχιστοποιήσει το κόστος επικοινωνίας μέσω της γνώσης αυτής.

5.2 Μελλοντικές Κατευθύνσεις

Όσο αφορά τους μελλοντικούς στόχους πρωτίστως μιας και όπως αναφέρθηκε και πιο πάνω η γνώση του MST μιας τοπολογίας είναι σημαντική, μπορούμε να επεκτείνουμε τον αλγόριθμο που κατασκευάσαμε σε πραγματικές συσκευές. Στην περίπτωση αυτή θα παρατηρήσουμε άμεσα μείωση στην επικοινωνία μεταξύ των κόμβων του δικτύου μας αλλά και στην χρονική πολυπλοκότητα.

Μια άλλη ενδιαφέρουσα κατεύθυνση είναι εκείνη των αυτοσταθεροποιούμενων αλγορίθμων όπως παρουσιάστηκε στο 3^ο κεφάλαιο. Στην υλοποίηση του αλγορίθμου μας θεωρήσαμε ιδανικές συνθήκες (ιδανικά κανάλια μετάδοσης), δηλαδή δεν λάβαμε υπόψη μας τις διάφορες πιθανές αστοχίες οι οποίες είναι συχνές και αναμενόμενες. Για παράδειγμα κάποια μηνύματα αργούν να μεταδοθούν ή δεν φθάνουν και ποτέ στον προορισμό τους. Ανάγοντας τον κατανεμημένο αλγόριθμο αυτό σε έναν κατανεμημένο και αυτοσταθεροποιούμενο αλγόριθμο αποφεύγουμε όλες αυτές τις αστοχίες και είμαστε πλέον σίγουροι για την απόδοση του αλγορίθμου μας ακόμη και για την χειρότερη αστοχία όσο αφορά την επικοινωνία.

Οι δυο παραπάνω κατευθύνσεις αποτελούν ενδιαφέρον θέμα για περαιτέρω εμβάθυνση στους κατανεμημένους αλγορίθμους αλλά και να αποδείξουμε για ακόμη μια φορά πόσο σημαντική είναι η συμβολή τους σε πραγματικά δίκτυα.

ΒΙΒΛΙΟΓΡΑΦΙΑ

- [1] "A Distributed Algorithm for Minimum-Weight Spanning Trees ", by R.G. Gallager, P.A. Hamlet, and P.M. Spira.
- [2] "Self-stabilizing minimum-degree spanning tree withn One from the optimal degree", by Lelia Blin, Maria Gradinariu Potop-Butucaru, Stephane Rovedakis.
- [3] "Distributed Algorithms" (N.Lynch).
Κεφάλαιο 4: Algorithms in General Synchronous Networks
- [4] <http://shawn.sourceforge.net/>
- [5] A. Kr"oller, D. Pfisterer, C. Buschmann, S.P. Fekete, and S. Fischer, Shawn: A new approach to simulating wireless sensor networks, Proceedings of the Design, Analysis, and Simulation of Distributed Systems (DASD '05), San Diego, 2005.
(<http://arxiv.org/abs/cs/0502003v1>)
- [6] "Distributed Computing Fundamentals, Simulations, and Advanced Topics" (H.Attiya, J.Welch)
Κεφάλαιο 2: Basic Algorithms in Message Passing Systems
- [7] "Introduction to Distributed Algorithms" (G.Tel)
Κεφάλαιο 6: Wave and Traversal Algorithms
- [8] "Distributed Systems, Concepts and Design" (G.Coulouris, J.Dollimore, T.Kindberg)
Κεφάλαιο 11: Coordination and Agreement
- [9] Dijkstra, E. Two problems in connection with graphs. Numer. Math. 1 (1959), 269-271.
- [10] Prim R.C. Shortest connection and some generalizations. Bell Syst. Tech.J.36 (1957), 1389-1401.
- [11] Spira, P. Communication complexity of distributed minimum spanning tree algorithms. In Proceedings, 2th Berkley Conference on Distributed Data Management and Computer Networks, Berkley Calif., June 1977.
- [12] Dalal, Y. Broadcast Protocols in packet switched computer networks. Tech. Rep. 128, Dep. Of Electrical Engineering, Stanford Univ., Apr. 1977 (revised version for publication in preparation.)

- [13] Humblet P.A. A distributed Algorithm for minimum weight directed Spanning Trees. Rep. LIDS-P-1149, Laboratory for Information and Decision Systems, Massachusetts Inst. Of Technology, Cambridge Mass., Sept. 1981.
- [14] Lawler, E. Combinatorial Optimization-Networks and Matroids. Holt, Rinehart & Winston, New York 1976.
- [15] LIU, C.L. Introduction to Combinatorial Mathematics. McGraw Hill, New York, 1968.
- [16] KRUSKAL, J.B. On the Shortest Spanning Subtree of a graph and the traveling salesman problem.
Proc. Am. Math. Soc. 7 (1956), 48-50.
- [17] YAO, A.C.C. An $O(E \log \log V)$ algorithm for finding minimum spanning trees. Inf. Process. Lett. 4 (1975), 21-23.
- [18] E. W. Dijkstra, Self-stabilizing systems in spite of distributed control, Communications of the ACM 17 (11) (1974) 643-644.
- [19] S. Dolev, Self-stabilization, MIT Press 2000.
- [20] M. Fischer, B. Raghavachari, Approximating the minimum degree spanning tree to within one from the optimal degree, in: Proc. of the 3rd ACM-SIAM Symp. on Discr. Algo. (SODA), 1992, pp. 317-324.
- [21] M. Fischer, B. Raghavachari, Approximating the minimum-degree steiner tree to within one of optimal, Journal of Algorithms 17 (3) (1994) 409-423.
- [22] L. Blin, F. Butelle, The first approximated distributed algorithm for the minimum degree spanning tree problem on general graphs, Int. J. Found. Comput. Sci. 15 (3) (2004) 507-516.
- [23] J. Burman, S. Kutten, Time optimal asynchronous self-stabilizing spanning tree, in: 21st International Symposium on Distributed Computing (DISC), 2007, pp. 92-107.