ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

ΤΜΗΜΑ ΜΑΘΗΜΑΤΙΚΩΝ

Αντρέας Γκριμπαβιώτης

ΤΟ ΠΡΟΒΛΗΜΑ ΤΟΥ ΠΛΑΤΟΥΣ ΑΠΟΚΟΠΗΣ ΣΕ ΣΧΕΔΟΝ ΚΑΤΩΦΛΙΚΑ ΓΡΑΦΗΜΑΤΑ

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΑΤΡΙΒΗ

Ιωάννινα, 2015

Η παρούσα Μεταπτυχιακή Διατριβή εκπονήθηκε στο πλαίσιο των σπουδών για την απόκτηση του Μεταπτυχιακού Διπλώματος Ειδίκευσης στα Εφαρμοσμένα Μαθηματικά και την Πληροφορική που απονέμει το Τμήμα Μαθηματικών του Πανεπιστημίου Ιωαννίνων.

Εγκρίθηκε την 16/12/2015 από την εξεταστική επιτροπή:

Ονοματεπώνυμο	**Βαθμίδα**
Παπαδόπουλος Χάρης	Επίκουρος Καθηγητής
Γλυνός Νικόλαος	Επίκουρος Καθηγητής
Μπαλτζής Σωκράτης	Λέκτορας

ΥΠΕΥΘΥΝΗ ΔΗΛΩΣΗ

«Δηλώνω υπεύθυνα ότι η παρούσα διατριβή εκπονήθηκε κάτω από τους διεθνείς ηθικούς και ακαδημαϊκούς κανόνες δεοντολογίας και προστασίας της πνευματικής ιδιοκτησίας. Σύμφωνα με τους κανόνες αυτούς, δεν έχω προβεί σε ιδιοποίηση ξένου επιστημονικού έργου και έχω πλήρως αναφέρει τις πηγές που χρησιμοποίησα στην εργασία αυτή.»

Γκριμπαβιώτης Ανδρέας

ΕΥΧΑΡΙΣΤΙΕΣ

Η ολοκλήρωση της παρούσας εργασίας σημαίνει και την ολοκλήρωση του μεταπτυχιακού μου στο τμήμα Μαθηματικών του Πανεπιστημίου Ιωαννίνων. Νοιώθω την ανάγκη να ήθελα να ευχαριστήσω τους ανθρώπους που συνέλαβαν στη διεκπεραίωση αυτής. Κατά κύριο λόγο, οφείλω να εκφράσω τις θερμές μου ευχαριστίες στον επιβλέποντα Καθηγητή μου κ. Παπαδόπουλο, ο οποίος προσέφερε το ενδιαφέρον θέμα και την εμπιστοσύνη που μου έδειξε δίνοντάς μου τη δυνατότητα να εκπονήσω την διπλωματική μου εργασία στο συγκεκριμένο επιστημονικό τομέα. Τον ευχαριστώ επίσης για τις πολύτιμες γνώσεις και συμβουλές που μου παρείχε καθ' όλη τη διάρκεια της εργασίας, καθώς και για την απρόσκοπτη υποστήριξη και καθοδήγηση που μου παρείχε. Ιδιαίτερες ευχαριστίες θα ήθελα να απευθύνω σε όλο το διδακτικό προσωπικό του τμήματος μαθηματικών για την αμέριστη και απλόχερη βοήθειά. Το αμείωτο ενδιαφέρον, οι υποδείξεις, η καθοδήγηση, η προθυμία τους και η συμπαράστασή τους κατά την διάρκεια της διετούς συνεργασίας μας ήταν καθοριστική. Η συνεργασία μαζί τους υπήρξε μοναδική διδακτική εμπειρία σε ένα φιλικό κλίμα, γεμάτο θετική ενέργεια. Η διάθεση όλων για την στήριξή μου και η υπομονή τους με οδήγησαν στην ομαλή διεκπεραίωση των μεταπτυχιακών μου σπουδών. Ένα μεγάλο ευχαριστώ οφείλω επίσης και στους συμφοιτητές μου μεταπτυχιακούς για την άριστη συνεργασία που είχαμε κατά την διάρκεια της φοίτησής μου.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένειά μου για τα όσα έχει κάνει για μένα, την Κατερίνα για τη στήριξη, τη συμπαράσταση και την κατανόηση, και τους καρδιακούς μου φίλους, καθώς και τον αδελφό μου Παναγιώτη, μαθηματικό για την πολύτιμη συνεισφορά του στην συγγραφή αυτή της εργασίας.

ΠΕΡΙΛΗΨΗ

Στη παρούσα εργασία μελετάμε το NP-δύσκολο πρόβλημα της εύρεσης του ελάχιστου πλάτους αποκοπής σε σχεδόν κατωφλικα γραφήματα. Στο πρόβλημα αυτό σκοπός μας είναι να βρούμε μια διάταξη των κορυφών ενός γραφήματος που ελαχιστοποιεί το μέγιστο πλήθος ακμών που περνάνε από κάθε τομή των κορυφών. Ως κλασικό πρόβλημα διάταξης κορυφών βρίσκει ποικίλες εφαρμογές. Πρώτα είχε εισαχθεί ως μοντελοποίηση για την ελαχιστοποίηση του πλήθους καναλιών ενός κυκλώματος σχεδίασης, ενώ αργότερα βρήκε εφαρμογές σε άλλες περιοχές όπως την αξιοπιστία δικτύου, αυτόματη σχεδίαση γραφημάτων, ανάκτηση πληροφορίας, ακόμα και ως υπορουτίνα για τον αλγόριθμο αποκοπής μιας επίπεδης επιφάνειας στο κλασικό πρόβλημα του περιοδεύοντος πωλητή.

Καθώς το πρόβλημα παραμένει NP-δύσκολο ακόμα και σε κλάσεις γραφημάτων όπως τα διμερή (bipartite), διαχωρίσιμα (split), επίπεδα (planar) γραφήματα μελετάμε το πρόβλημα σε κλάσεις γραφημάτων που είναι ανοιχτή η υπολογιστική πολυπλοκότητα του προβλήματος. Από την θετική σκοπιά, είναι γνωστοί ορισμένοι αλγόριθμοι γραμμικού χρόνου για την κλάση των κατωφλικών (threshold) γραφημάτων και για την κλάση των διμερή μεταθετικών (bipartite permutation) γραφημάτων. Ως άμεση συνέπεια εξετάζουμε το πρόβλημα στην άμεση υπερκλάση των κατωφλικών γραφημάτων, γνωστή ως σχεδόν κατωφλικά γραφήματα.

Βασιζόμενοι σε δομικές ιδιότητες των σχεδόν κατωφλικών γραφημάτων, αποδεικνύουμε χαρακτηρισμούς της βέλτιστης διάταξης που μας επιτρέπουν την σχεδίαση αλγορίθμου που τρέχει σε χρόνο $\mathrm{O}(2^N)$ με $N \leq n$ που είναι γρηγορότερο από τον εκθετικό $\mathrm{O}(2^n)$ αλγόριθμο για γενικά γραφήματα με n κορυφές. Επίσης δίνουμε έναν πολυωνυμικό αλγόριθμο για την υποκλάση των σχεδόν κατωφλικών γραφημάτων, που την ονομάζουμε 1-επιπέδου σχεδόν κατωφλικά γραφήματα. Προς την κατεύθυνση ανάπτυξης πολυωνυμικού αλγορίθμου για την γενική περίπτωση, παρουσιάζουμε αντιπαραδείγματα για πολλές ιδιότητες που θα περιμέναμε να έχει μια βέλτιστη διάταξη. Για τον σκοπό αυτό υλοποιήσαμε αρκετές τεχνικές που βοηθάνε την αντιμετώπιση του προβλήματος. Ωστόσο, αξίζει να σημειώσουμε ότι το υπολογιστικό πρόβλημα της εύρεσης ελάχιστου πλάτους αποκοπής σε σχεδόν κατωφλικά γραφήματα παραμένει ανοικτό.

ABSTRACT

In this thesis we study the Cutwidth Minimization Problem in quasi-threshold graphs, an NP-hard problem that consists of finding a linear layout of a graph so that the maximum linear cut of edges (i.e., the number of edges that cut a line between consecutive vertices) is minimized. As a well-studied layout problem finds many applications. The Cutwidth problem was first used as a model for the number of channels in an optimal layout of a circuit, to give a measure of the area needed to represent the graph in a VLSI layout when nodes are laid out in a row and more recently we find applications of this problem in network reliability, automatic graph drawing, information retrieval and as a subroutine for the cutting plane algorithm to solve the TSP.

The problem remains NP-Hard even in graph classes such as the bipartite, the split and the planar graphs. Thus we study this problem in classes that it still remains open. In a positive side of view polynomial time algorithms exists for the subclass of thresholds and for the bipartite permutation graphs. As a consequence we study this problem in the direct superclass of quasi-threshold graphs.

Based on the structural properties of the quasi-threshold graphs we prove several acclaims for the form of an optimal linear ordering, thus enabling us to design an algorithm that runs in $O(2^N)$ time, where $N \leq n$, that is faster than the algorithm for general graphs that runs

in $O(2^n)$ time. We also design a polynomial time algorithm for a subclass of the quasi-threshold graphs, that we call 1-level quasi-threshold graphs. As we struggle to prove a polynomial time algorithm for the cutwidth problem in quasi-threshold graph we introduce counter examples for many properties one would expect to encounter in an optimal ordering. For this end we developed many techniques to tackle with the problem. We have to note, although, that the cutwidth problem for the quasi-threshold graphs remains open

ΠΕΡΙΕΧΟΜΕΝΑ

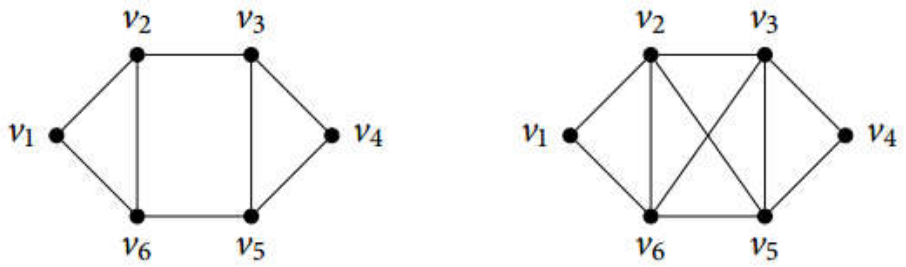
ΚΕΦΑΛΑΙΟ 1

1. Εισαγωγή

1.1. Θεωρία γραφημάτων(ορισμοί και συμβολισμοί)

Ένα *γράφημα* G είναι ένα διατεταγμένο ζεύγος $G = (V, E)$, όπου V ένα πεπερασμένο σύνολο που το καλούμε σύνολο *κορυφών* (ή σύνολο *κόμβων*) , και E ένα υποσύνολο του συνόλου των μη διατεταγμένων ζευγών του $V, V \times V$, το οποίο εκφράζει τις *ακμές* του γραφήματος. Το πλήθος των κορυφών του γραφήματος συμβολίζεται συνήθως με $n = |V(G)|$ και ονομάζεται ***τάξη*** του γραφήματος. Το πλήθος των ακμών του γραφήματος συμβολίζεται συνήθως με $m = |E(G)|$ και ονομάζεται *μέγεθος* του γραφήματος. Για μια κορυφή $v \in V(G)$ καλούμε *γειτονιά* του v το σύνολο των κορυφών που συνδέεται η v με ακμή, και συμβολίζεται ως $N(v)$ και $N(v) =_{def} \{u|\{v, u\} \in E(G)\}$. Η *γειτονιά* ενός συνόλου $S \subseteq V(G)$ συμβολίζεται $N(S)$ και ορίζεται ως το σύνολο των κορυφών $u \in S|\{u, v\}| \in E(G)$ για κάποια κορυφή v. Αντίστοιχα *κλειστή γειτονιά* της κορυφής v ορίζεται ως $N[v] =_{def} N(v) \cup \{v\}$, και αντίστοιχα

κλειστή γειτονία ενός συνόλου $S \in V(G), N[S] = N(S) \cup S$. Αν για μία κορυφή ισχύει ότι N(v) = τότε ονομάζεται *απομονωμένη*(*isolated*) ενώ αν $N[v] = V(G)$ τότε ονομάζεται *καθολική*(*universal*). Ένα γράφημα ονομάζεται *απλό* αν δεν υπάρχουν ακμές μεταξύ μιας κορυφής και του εαυτού της, καθώς και παραπάνω από μία ακμές με τα ίδια άκρα. Στην παρούσα εργασία όλα τα γραφήματα θεωρούνται απλά. Ο *βαθμός(degree)* μιας κορυφής v συμβολίζεται με $\deg(v)$ και είναι ο πληθικός αριθμός της ανοικτής γειτονιάς του v, $\deg(v) = |N(v)|$.Ο βαθμός ενός συνόλου $S \subseteq V(G)$ συμβολίζεται με $deg(S)$ και ορίζεται ως $\deg(S) = |\{u, v\} \in E(G)|u \in S \wedge v \in V(G) \backslash S\}|$. Ορίζουμε ως βαθμό μια κορυφής v ως προς ένα σύνολο $S \subseteq V(G)$, $\deg_S(v) = N(v) \cap S$. Ο *ελάχιστος* και ο *μέγιστος βαθμός* ενός γραφήματος είναι $\delta(G) = \min_{v \in V(G)} \deg(v)$ και $\Delta(G) = \max_{v \in V(G)} \deg(v)$.

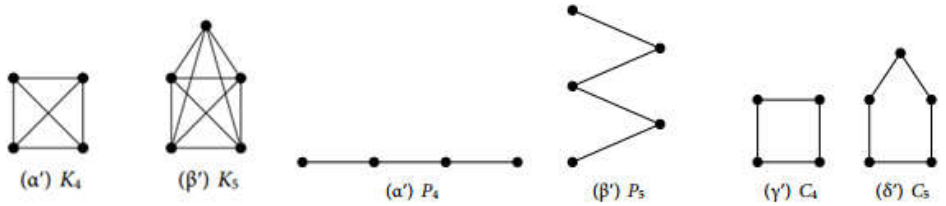


(α′) Ένα γράφημα G (β′) Ένα γράφημα H

Εικόνα 1 Δύο γραφήματα

Ένα γράφημα H είναι ένα *υπογράφημα* του γραφήματος G αν $V(H) \subseteq V(G)$ και $E(H) \subseteq E(G)$. Για ένα σύνολο S των κορυφών, με $G[S]$ συμβολίζουμε το *επαγόμενό* υπογράφημα του G που επάγεται από το S. Το $G[S]$ είναι ένα γράφημα που αποτελείται από τις κορυφές S και όλες

τις ακμές$\{u, v\} \in E(G)$: $u, v \in S$. Για ένα γράφημα H, το γράφημα G ονομάζεται *H-ελεύθερο (H-free) εάν δεν υπάρχει σύνολο* $S \subseteq V(G)$ *τέτοιο ώστε το* H *να είναι ισόμορφο με το* $G[S]$.



Εικόνα 2 Κλίκες, μονοπάτια και κύκλοι

Μια ακολουθία κορυφών $v_1, v_2, \ldots v_k \in V(G)$ ονομάζεται *μονοπάτι* μήκους $\kappa \quad 1$ στο G αν $(v_1, v2), (v_2, v_3) \ldots . (u_{k\ 1}, v_k) \in E(G)$ και συμβολίζεται P_k. Αν επίσης και $(v_k, v_1) \in E(G)$ τότε ονομάζεται κύκλος μεγέθους κ και συμβολίζεται με $\mathrm{C_k}$. Ένα γράφημα είναι *συνεκτικό* όταν υπάρχει ένα μονοπάτι μεταξύ δύο οποιονδήποτε κορυφών του $u, v \in V(G)$. Το γράφημα με $n \geq 1$ κορυφές και ακμές για κάθε ζεύγος κορυφών το ονομάζουμε *πλήρες γράφημα ή κλίκα (clique)* , το συμβολίζουμε με K_n και ισχύει $K_n = \Big\{\{v_1, \ldots, v_n\}, \{\{v_i, v_j\} | 1 \leq i, j \leq n\}\Big\}$. Άρα κλίκα είναι ένα σύνολο από κορυφές που ενώνονται όλες μεταξύ τους. Αντίστοιχα ένα *ανεξάρτητο σύνολο* είναι ένα σύνολο από όλα τα ζεύγη μη γειτονικών κορυφών. Ένα σύνολο καλείται *κάλυμμά κορυφών(vertex cover)* αν για όλες τις ακμές του γραφήματος ένα τουλάχιστον άκρο της ανήκει στο σύνολο αυτό. Μια *κλίκα ονομάζεται καθολική (universal clique)* αν η κλειστή γειτονιά της είναι ολόκληρο το γράφημα(«βλέπει» όλες της κορυφές του γραφήματος). Μια κλίκα K ονομάζεται *μεγιστοτική (maximal)* αν δεν υπάρχει καμία κορυφή $v \in V(G) \quad K$ τέτοια ώστε $K \cup \{v\}$ να είναι επίσης κλίκα. Μια μεγιστοτική

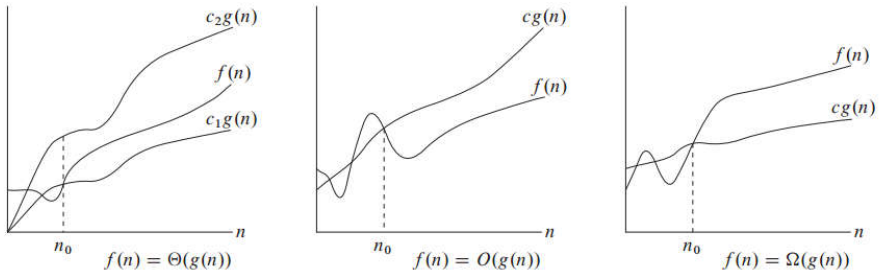
κλίκα είναι *μέγιστη* αν είναι η μεγαλύτερη μεγιστοτική κλίκα του γραφήματος [33].

Ορίζουμε ως *γραμμική διάταξη (linear ordering)* των κορυφών ενός γραφήματος G την αμφιμονοσήμαντη συνάρτηση $\sigma: V(G) \leftrightarrow \{1,2,\ldots,|V(G)|\}$. Τότε λέμε ότι μια κορυφή v είναι *αριστερά* μιας κορυφής u στην σ αν $\sigma(v) < \sigma(u)$, και αντίστοιχα λέμε είναι δεξιά αν $\sigma(v) > \sigma(u)$. Μερικές φορές συμβολίζουμε μια γραμμική διάταξη ως $\sigma = < v_1, v_2, \ldots v_n >$, εννοώντας ότι $s(v_i) = i \ \forall 1 \leq i \leq n$. Η λύση πολλών προβλημάτων στην θεωρία γραφημάτων απαιτεί την εύρεση μιας γραμμικής διάταξης που ελαχιστοποιεί/μεγιστοποιεί μια συνάρτηση $f(\sigma)$. Μια γραμμική διάταξη ονομάζεται *βέλτιστη* ως προς το πρόβλημα αυτό όταν δεν υπάρχει σ τέτοια ώστε $\mathrm{f}(\sigma) < f(\sigma)$ (ή αντίστοιχα μεγαλύτερη).Ορίζουμε σαν μια *διαμέρηση* ενός συνόλου S , τα σύνολα $S_1, S_2 \ldots S_i$ τέτοια ώστε $\mathrm{S}_1 \cup S_2 \ldots \cup S_i = S$ και $|S_1| + |S_2| + \quad |S_i| = |S|$.

1.2. Ασυμπτωτικός συμβολισμός

Έστω μία συνάρτηση $f(n)$ η οποία περιγράφει τον χειρότερο χρόνο εκτέλεσης ενός αλγορίθμου για δεδομένη είσοδο n. Ορίζουμε ότι η συνάρτηση $f(n)$ είναι $O\big(g(n)\big)$ (κεφαλαίο όμικρον του g(n)) αν υπάρχει σταθερά c και ακέραιος κ ώστε να ισχύει $f(n) \leq cg(n) \ \forall \ n \geq k$. Τότε λέμε ότι ο *χρόνος εκτέλεσης* του αλγορίθμου είναι $O(g(n))$. Ένας αλγόριθμος λέμε ότι είναι *αποδοτικός* αν υπάρχει πολυωνυμική συνάρτηση $f(n)$ τέτοια ώστε ό χρόνος εκτέλεσης να είναι $O(f(n))$. Ένας αλγόριθμος λέμε ότι είναι *γραμμικός* αν ο χρόνος εκτέλεσης του είναι $O(n)$.

Παρόμοια ορίζουμε και το συμβολισμό $\Omega\big(g(n)\big)$ (κεφαλαίο ωμέγα του g(n)) αν υπάρχει σταθερά $c > 0$ και ακέραιος κ τέτοια ώστε $f(n) \geq cg(n)\ \forall\ n \geq k$. Μια συνάρτηση λέμε ότι είναι $\Theta\big(g(n)\big)$ αν υπάρχουν σταθερές $\mathrm{c}_1, \mathrm{c}_2 > 0$ και ακέραιος κ τέτοιες ώστε $c_1 g(n) \leq f(n) \leq c_2 g(n)\ \forall\ n \geq k$. Από το ορισμό των συμβολισμών προκύπτει ότι μια συνάρτηση $f(n)$ είναι $\Theta\big(g(n)\big)$ όταν ισχύει ταυτόχρονα $O\big(g(n)\big)$ και $\Omega\big(g(n)\big)$.



Εικόνα 3 Ασυμπτωτικοί συμβολισμοί

Ο συμβολισμός $f(n) = O\big(g(n)\big)$ εισάγει ένα *ασυμπτωτικό άνω φράγμα* του χρόνου εκτέλεσης ενός αλγορίθμου, και αντίστοιχα ο συμβολισμός $\Omega\big(g(n)\big)$ ορίζει ένα *ασυμπτωτικό κάτω φράγμα, ενώ* ο συμβολισμός $\Theta\big(g(n)\big)$ *ένα ασυμπτωτικό αυστηρό φράγμα.* Αφού οι συμβολισμοί αυτοί εισάγουν φράγματα στο χρόνο εκτέλεσης χειρότερης περίπτωσης ενός αλγορίθμου, εξασφαλίζουμε ότι τα φράγματα αυτά ισχύουν για οποιαδήποτε είσοδο. Με τον τρόπο αυτό μπορούμε να συγκρίνουμε την ταχύτητα εκτέλεσης και την απόδοση αλγορίθμων μεταξύ τους. Στην πράξη απλοποιούμε το φράγμα ενός αλγορίθμου κρατώντας μόνο το μεγιστοβάθμιο όρο του χρόνου εκτέλεσης, για παράδειγμα ένας αλγόριθμος με χρόνο εκτέλεσης $f(n) = 10n^2 + 30n + 20$ συνήθως λέμε ότι είναι $\Theta(n^2)$, αγνοώντας τους υπόλοιπους όρους.

Με βάση τα παραπάνω μπορούμε να ορίσουμε μια διάταξη ως προς τον χρόνο εκτέλεσης των αλγορίθμων και να ορίσουμε κλάσεις προβλημάτων βάση του χρόνου αυτού όπως αναφέρεται παρακάτω.

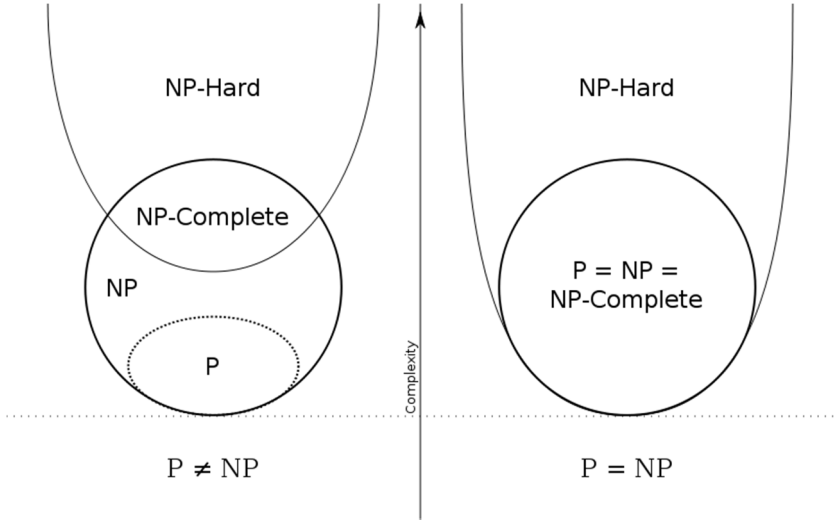
$$O(\log(n)) \leq O(n) \leq O\big(nlog(n)\big) \leq O(n^2) \leq O(n^k) \leq O(2^n) \leq O(k^n)$$

1.3. Κλάσεις πολυπλοκότητας

Τα προβλήματα χωρίζονται σε κλάσεις ανάλογα με τους αλγόριθμούς που έχουν βρεθεί να τα λύνουν και τον χρόνο που αυτοί απαιτούν για να το λύνουν. Οι κλάσεις αυτές ονομάζονται *κλάσεις πολυπλοκότητας*. Έτσι όλα τα προβλήματα για τα οποία υπάρχει γνωστός αλγόριθμος χρόνου $O(n^k)$ για κάποια σταθερά κ λέμε ότι ανήκουν στην κλάση P(polynomial time solvable). Όταν για δύο προβλήματα Α και Β υπάρχει τρόπος ώστε να μετατρέψουμε όλες τις εμφανίσεις του προβλήματος Α σε εμφανίσεις του προβλήματος Β *αποδοτικά*, έτσι ώστε μια λύση για το πρόβλημα Β να ισοδυναμεί με λύση για το πρόβλημα Α , τότε λέμε ότι το πρόβλημα Α *ανάγεται* στο Β. Τέτοιοι μετασχηματισμοί ονομάζονται *αναγωγές*, ουσιαστικά είναι σχέσεις ισοδυναμίας δύο προβλημάτων αφού μας δείχνει ότι το πρόβλημα Β είναι τουλάχιστον τόσο δύσκολο όσο και το Α(μια αποδοτική λύση για το Β μας δίνει και αποδοτική λύση για το Α).

Υπάρχουν προβλήματα για τα οποία δεν είναι γνωστός αλγόριθμος πολυωνυμικού χρόνου αλλά μπορούμε να επιβεβαιώσουμε ότι μια δοθείσα λύση είναι όντως λύση του προβλήματος αυτού σε πολυωνυμικό χρόνο. Ο αλγόριθμος που επιβεβαιώνει την λύση αυτή λέγεται *πιστοποιητής* και τα προβλήματα που έχουν πολυωνυμικό πιστοποιητή ανήκουν στην κλάση NP(non-deterministic polynomial time solvable). Είναι προφανές ότι

όλα τα προβλήματα της κλάσης P είναι και NP αρά $P \subseteq NP$ αλλά δεν έχει αποδειχτεί ακόμα αν οι κλάσεις αυτές ταυτίζονται δηλαδή αν $P = NP$, και αυτό είναι ένα από τα Millenium Prize Problems που έχει δώσει το Ινστιτούτο Μαθηματικών Clay(και η απόδειξη του χαρίζει ένα εκατομμύριο δολάρια) [20]. Τα προβλήματα τα οποία *ανήκουν στην κλάση NP* για τα οποία επίσης *υπάρχει αναγωγή σε πολυωνυμικό χρόνο σε όλα τα υπόλοιπα προβλήματα της κλάσης NP* λέμε ότι ανήκουν στην κλάση *NP-Complete.* Το 1971 ο Stephen Cook απέδειξε ότι όλα τα προβλήματα της κλάσης NP μπορούν να αναχθούν πολυωνυμικά στο Satisfiability Problem, γνωστό και ως SAT. Το πρόβλημα αυτό έχει πολυωνυμικό πιστοποιητή και έγινε το πρώτο πρόβλημα της κλάσης NP-Complete.Έτσι μια αναγωγή του προβλήματος SAT σε ένα πρόβλημα δείχνει ότι όλα τα προβλήματα της κλάσης NP ανάγονται σε αυτό. Η κλάση NP-Complete υποδηλώνει την πλήρη ισοδυναμία των προβλημάτων που ανήκουν σε αυτή και ότι αν βρισκόταν πολυωνυμικός αλγόριθμος για ένα από αυτά τότε θα υπήρχε πολυωνυμικός αλγόριθμος για όλα τα προβλήματα της κλάσης NP και άρα θα ισοδυναμούσε με την απόδειξη του $P = NP$. Τα προβλήματα για τα οποία υπάρχει ένα πρόβλημα της κλάσης NP-Complete το οποίο μπορεί να αναχθεί πολυωνυμικά σε αυτό λέγονται *NP-Hard*, και φυσικά αφού όλα τα προβλήματα της κλάσης NP-Complete είναι ισοδύναμα κάθε ένα από αυτά θα μπορούσε να αναχθεί σε πολυωνυμικό χρόνο. Ουσιαστικά η κλάση NP-Hard αντιπροσωπεύει τα προβλήματα που είναι τουλάχιστον τόσο δύσκολα όσο το πιο δύσκολο πρόβλημα της κλάσης NP-Complete. Ένα πρόβλημα NP-Hard μπορεί να μην ανήκει στην κλάση NP(να μην έχει πολυωνυμικό πιστοποιητή), εξάλλου, ισοδύναμα ένα πρόβλημα ανήκει στην κλάση NP-Complete αν και μόνο αν είναι NP και NP-Hard.



Εικόνα 4 Κλάσεις πολυπλοκότητας

Όμοια με τις κλάσεις πολυπλοκότητας για τον χρόνο μπορούμε να χωρίσουμε τα προβλήματα σε αντίστοιχες κλάσεις για τον χώρο που απαιτούν οι λύσεις τους(*Space Complexity*). Με το όρο χώρο συνήθως εννοούμε την μνήμη που απαιτείται για την υλοποίηση του αλγορίθμου επίλυσης τους. Ο χώρος που απαιτεί ένας αλγόριθμος περιγράφεται σαν μια συνάρτηση $f(n)$ και εκφράζουμε τις κλάσεις πολυπλοκότητας χώρου με τον ασυμπτωτικό συμβολισμό($O(g(n))$. Έτσι έχουμε τις κλάσεις P-Space(όπου ο χώρος που απαιτείται είναι $O(n^k)$) , Exp-Space $(O(k^n))$ $\kappa.\alpha.$

Η αυστηρή διατύπωση των κλάσεων προβλημάτων απαιτεί τα προβλήματα να είναι *προβλήματα απόφασης*, δηλαδή προβλήματα που απαντώνται με Ναι ή Όχι, αλλά τα περισσότερα προβλήματα στην πράξη είναι *προβλήματα βελτιστοποίησης,* δηλαδή εύρεσης ενός μεγίστου ή ενός ελαχίστου. Τα προβλήματα βελτιστοποίησης όμως μπορούν να επαναδιατυπωθούν ως προβλήματα απόφασης εισάγοντας ένα φράγμα

στην τιμή την οποία βελτιστοποιούν. Για παράδειγμα το πρόβλημα συντομότερου μονοπατιού μπορεί να επαναδιατυπωθεί στο αν υπάρχει συντομότερο μονοπάτι μεγέθους το πολύ κ.

Για τα προβλήματα της κλάσης NP δεν υπάρχουν γνωστοί πολυωνυμικοί αλγόριθμοι που να τα λύνουν και άρα δεν μπορούν να επιλυθούν αποδοτικά. Υπάρχουν όμως τεχνικές με τις οποίες μπορούμε να προσεγγίσουμε λύσεις για τα προβλήματα αυτά. Οι *προσεγγιστικοί αλγόριθμοι* είναι αλγόριθμοι οι οποίοι δεν υπολογίζουν την ακριβή λύση ενός προβλήματος αλλά μια προσέγγιση τους. Για να θεωρηθεί όμως ένας αλγόριθμος προσεγγιστικός πρέπει να εγγυάται ότι η προσέγγιση δεν θα απέχει περισσότερο από r από την ακριβή λύση όπου r μια σταθερά ή μια συνάρτηση της εισόδου. Έτσι ένας αλγόριθμος είναι r-προσεγγιστικός(r-approximate) [2].

Οι *ευριστικοί* (Heuristics) είναι αλγόριθμοι οι οποίοι δίνουν αρκετά γρήγορες λύσεις στην πράξη αλλά δεν είναι γνωστή η προσέγγιση που έχουν ως προς την βέλτιστη λύση. Οι αλγόριθμοι αυτοί μπορεί να έχουν πολύ κακή συμπεριφορά σε συγκεκριμένες εισόδους αλλά στην πράξη να μην εμφανίζονται πολύ συχνά. Ένας πολύ γνωστός τέτοιος αλγόριθμος είναι ο A^*, που επιλύει το πρόβλημα ελάχιστων μονοπατιών σε αραιά γραφήματα σε γρήγορο χρόνο.

Οι αλγόριθμοι *περιορισμένης εισόδου* είναι αλγόριθμοι για τους οποίους έχουμε εκ των προτέρων περιορίσει τις πιθανές εισόδους για το πρόβλημα. Για παράδειγμα περιορίζοντας ένα πρόβλημα από τα γενικά γραφήματα, σε γραφήματα με συγκεκριμένες δομικές ιδιότητες μπορούμε να επιτύχουμε πιο γρήγορες λύσεις όπως θα δούμε παρακάτω.

Ένας σύγχρονος κλάδος της πολυπλοκότητας ασχολείται με την κατηγοριοποίηση των προβλημάτων βάσει της εγγενής τους δυσκολίας να λυθούν βάσει κάποιων παραμέτρων της εισόδου. Η πολυπλοκότητα του προβλήματος τότε μελετάται βάσει αυτών των παραμέτρων. Αυτό επιτρέπει την διάταξη των προβλημάτων της κλάσης NP σε μια ακριβέστερη κλίμακα από ότι η κλασσική πολυπλοκότητα. Η πρώτη συστηματική εργασία πάνω σε αυτό τον τομέα έγινε από τους [13].

Υπό την υπόθεση ότι $P \neq NP$ υπάρχουν πολλά προβλήματα που απαιτούν περισσότερο από πολυωνυμικό χρόνο εκτέλεσης όταν αυτός μετράτε με βάση το μέγεθος εισόδου και μόνο, αλλά μπορούν να υπολογιστούν σε χρόνο που είναι πολυωνυμικός ως προς το μέγεθος εισόδου αλλά εκθετικός ή χειρότερος βάση μιας παραμέτρου κ. Συνεπώς αν το κ είναι σταθερό σε μια μικρή σχετικά τιμή και η αύξηση της συνάρτησης του κ είναι μικρή τα προβλήματα μπορούν να θεωρηθούν *επιλύσιμα(tractable)* πάρα την κλασσική τους κατηγοριοποίηση ως *μη επιλύσημα(intractable).* Άρα λοιπόν για πολλά προβλήματα υπάρχουν αλγόριθμοι που έχουν χρόνο εκτέλεσης $O\big(f(k)\ poly(n)\big)$, όπου η f είναι συνήθως μια εκθετική συνάρτηση. Αυτοί οι αλγόριθμοι ονομάζονται *σταθερής παραμέτρου επιλύσιμοι, Fixed Parameter Tractable ή FPT αλγόριθμοι* επειδή επιλύουν το πρόβλημα αποδοτικά για μικρές τιμές της παραμέτρου κ. Τα προβλήματα για τα οποία υπάρχει τέτοιος γνωστός αλγόριθμος λέμε ότι ανήκουν στην κλάση FPT. Για παράδειγμα υπάρχει αλγόριθμος που επιλύει το πρόβλημα του καλύμματός κορυφών σε χρόνο $O(kn + 1.274^k)$ χρόνο [8], όπου το n είναι ο αριθμός των κορυφών και k το μέγεθος του καλύμματός. Αυτό σημαίνει ότι το πρόβλημα του καλύμματός κορυφών είναι FPT πρόβλημα με το μέγεθος της λύσης ως παράμετρο.

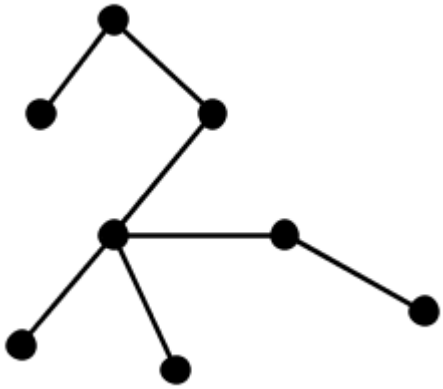
Ένα ορόσημο για τους ακριβής αλγόριθμους είναι η εύρεση αλγορίθμου με χρόνο εκτέλεσης $O((2 \quad e)^n), e > 0$, δηλαδή ταχύτερο από $O(2^n)$. Για το πρόβλημα του Πλάτους Αποκοπής(Cutwidth), που είναι και το αντικείμενο της παρούσας εργασίας , δεν έχει βρεθεί τέτοιος αλγόριθμος.

1.4. Κλάσεις γραφημάτων

Πολλά από τα προβλήματα που αντιμετωπίζουμε στην Αλγοριθμική θεωρία Γραφημάτων είναι δύσκολα(NP) στην γενική τους μορφή. Για το λόγο αυτό τα γραφήματα χωρίζονται σε κλάσεις βάσει των δομικών τους χαρακτηριστικών. Έτσι πολλά από τα προβλήματα αυτά όταν μελετηθούν για κάθε κλάση γραφημάτων χωριστά και χρησιμοποιηθούν οι ιδιότητες της κάθε κλάσης μπορεί να γίνουν πιο εύκολα. Ήδη έχουμε αναφέρει μερικά πολύ απλά γραφήματα όπως το μονοπάτι(P_n) και ο κύκλος(C_n) και στην παράγραφο αυτή θα παρουσιάσουμε μερικές από τις πιο γνωστές κλάσεις γραφημάτων. [7]

Δένδρα

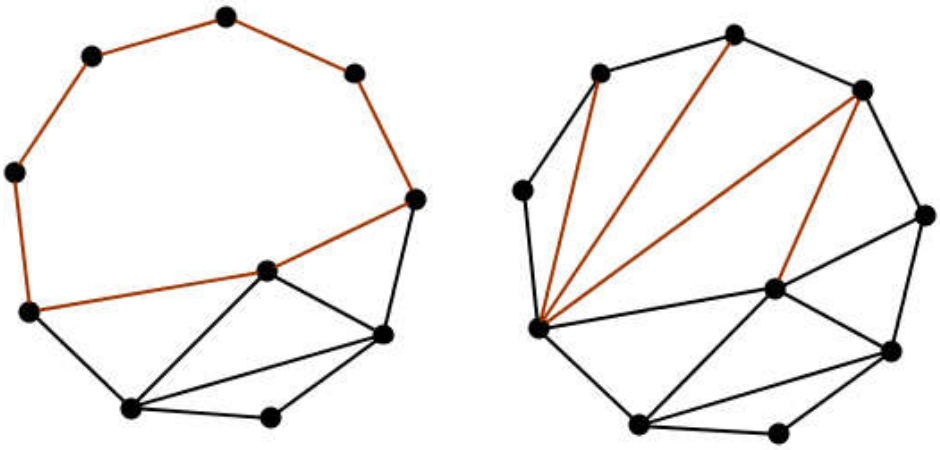
Τα δέντρα είναι τα συνεκτικά γραφήματα που δεν περιέχουν κύκλους. Ένα γράφημα είναι δέντρο αν μεταξύ δύο κορυφών του υπάρχει ακριβώς ένα μονοπάτι. Η ένωση δύο δέντρων είναι ένα *δάσος*. Τα δένδρα είναι μια απλή κλάση όπου πολλά προβλήματα των γραφημάτων έχουν πολυωνυμικούς αλγορίθμους για την λύση τους.



Εικόνα 5 Δέντρο

Χορδικά γραφήματα(Chordal)

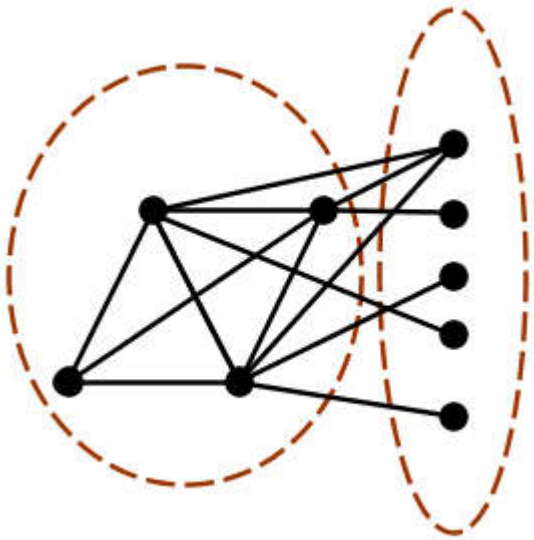
Ένα γράφημα είναι χορδικό αν δεν έχει C_k επαγόμενα γραφήματα για κάθε $\kappa > 3$. Από τον ορισμό τους τα χορδικά γραφήματα είναι υπερκλάση των δέντρων.



Εικόνα 6 Α) ένα γράφημα με κύκλο C_7 *Β) Ένα χορδικό γράφημα*

Split γραφήματα

Split ονομάζονται τα γραφήματα που μπορούν να διαχωριστούν ακριβώς σε μία κλίκα και ένα ανεξάρτητο σύνολο. Όλα τα split γραφήματα είναι χορδικά, αφού κανένα τέτοιο γράφημα δεν μπορεί να περιέχει κύκλους με μήκος μεγαλύτερο από τρία.

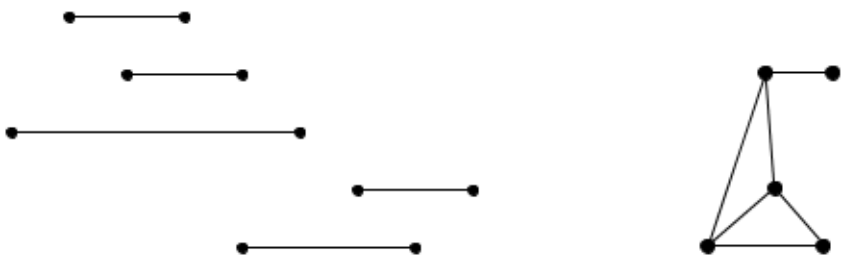


Εικόνα 7 Ένα split γράφημα, με μια κλίκα K_4, και το ανεξάρτητο σύνολο

Interval γραφήματα

Ένα γράφημα είναι interval αν υπάρχει μια αντιστοιχία μεταξύ των κορυφών του σε διαστήματα πάνω στην γραμμή των πραγματικών αριθμών τέτοια ώστε κάθε κορυφή να αντιστοιχεί σε ακριβώς ένα συνεχές διάστημα και αν υπάρχει ακμή μεταξύ 2 κορυφών τότε τα αντίστοιχα διαστήματα τους να τέμνονται. Η αναπαράσταση των γραφήματων αυτών με τα διαστήματα του πάνω στον άξονα των πραγματικών αριθμών ονομάζεται *μοντέλο διαστημάτων(interval model)*.Τα interval γραφήματα

δεν έχουν επαγόμενους κύκλους με μήκος μεγαλύτερο του 3 και άρα όλα είναι χορδικά.



Εικόνα 8 Ένα interval γράφημα και η αναπαράσταση του

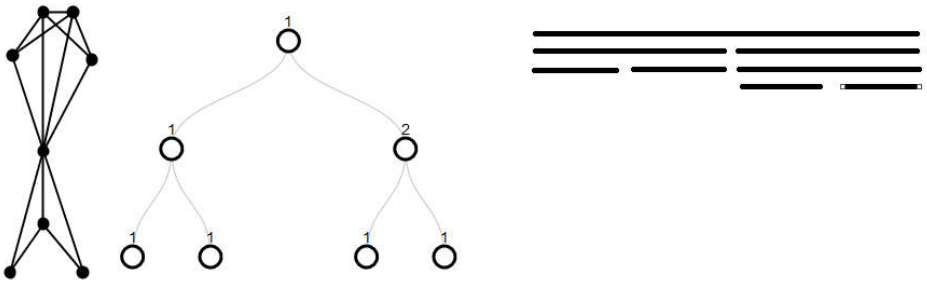
Proper interval γραφήματα

Είναι τα γραφήματα όπου στην αναπαράσταση τους κανένα διάστημα δεν περιέχεται εξολοκλήρου σε κανένα άλλο διάστημα. Αυτή η κλάση είναι ισοδύναμη με τα *unit interval γραφήματα* όπου κάθε διάστημα έχει μήκος ακριβώς ένα.

Σχεδον-Κατωφλικά γραφήματα(quasi-threshold)

Είναι γνωστά και ως *Trivially Perfect Γραφήματα*, και είναι η κλάση των γραφημάτων που το μοντέλο διαστημάτων τους δεν έχει επικαλυπτόμενα διαστήματα. Κάθε συνεκτικό γράφημα της κλάσης αυτής μπορεί να αναπαρασταθεί από ένα ριζωμένο δένδρο όπου κάθε μονοπάτι από την ρίζα μέχρι ένα φύλλο αντιστοιχεί σε μια μεγιστοτική κλίκα. Η αναπαράσταση ως δένδρο των γραφημάτων αυτών ονομάζεται *δενδρική αναπαράσταση*, και απαιτεί $O(n)$ χώρο(ενώ ένα τυχαίο γράφημα απαιτεί $O(n+m)$). Ως *συμπαγής δεντρική αναπαράσταση* ενός σχεδόν κατωφλικού γραφηματος ορίζουμε ένα ριζωμένο δέντρο από σύνολα

κόμβων τουλάχιστον μεγεθούς 1 που αποτελούν κλίκες του γραφήματος και κάθε κορυφή του γραφήματος υπάρχει σε ένα και μόνο τέτοιο σύνολο. Οι κορυφές που ανήκουν στα σύνολα ενός μονοπατιού από την ρίζα προς ένα φύλλο ορίζουν μια μεγιστοτική κλίκα στο γράφημα.



Εικόνα 9 Ένα σχεδόν κατωφλικό γράφημα, η δεντρική του αναπαράσταση και η interval αναπαράσταση του

Δεν υπάρχουν ακμές μεταξύ δύο κορυφών αν δεν υπάρχει μονοπάτι από την ρίζα προς ένα φύλλο που να περιέχει και τις δύο αυτές κορυφές. Από τις ιδιότητες των γραφημάτων αυτών αποδεικνύεται ότι υπάρχει μια τέτοια αναπαράσταση και *είναι μοναδική* [31].

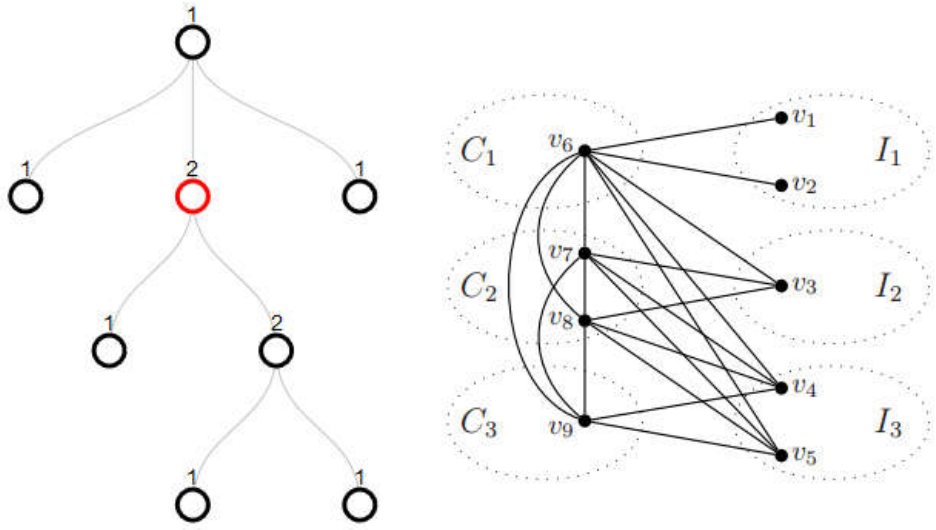
Τα γραφήματα που έχουν συμπαγής δεντρική αναπαράσταση βάθους 1 λέγονται *υπερ*εύθραυστα *(superfragile)* και το πρόβλημα του πλάτους αποκοπής σε αυτά αποδεικνύεται ότι είναι πολυωνυμικού χρόνου. [26]

Τα σχεδόν κατωφλικά γραφήματα είναι και το αντικείμενο μελέτης της παρούσας εργασίας και θα δούμε αναλυτικά τις ιδιότητες τους στο επόμενο κεφάλαιο.

Κατωφλικά γραφήματα(Threshold)

Είναι τα γραφήματα τα οποία μπορούν να κατασκευαστούν είτε προσθέτοντας μια απομονωμένη κορυφή ή μια καθολική κορυφή.

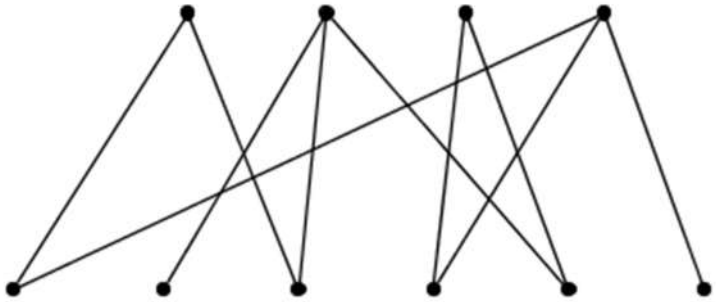
Επομένως είναι τα γραφήματα για τα οποία υπάρχει μια διάταξη των κορυφών $\{v_1, v_2, ... v_n\}$ τέτοια ώστε κάθε κορυφή v_i είναι είτε καθολική είτε απομονωμένη στο επαγόμενο από τα $\{v_1, v_2, ... v_i\}$ γράφημα. Αυτή η διάταξη καλείται *διάταξη κατασκευής*. Είναι υπό-κλάση των interval γραφημάτων. Με γνωστή την διάταξη κατασκευής ενός κατωφλικού γραφήματος μπορεί να κατασκευαστεί το μοντέλο διαστημάτων ως εξής: κάθε κόμβος v_j αντιστοιχεί στο διάστημα $[2j \quad 1,2j]$ αν είναι απομονωμένος ή στο $[1, 2j]$ αν είναι καθολικός. Τα κατωφλικά γραφήματα είναι επίσης και Split γραφήματα, αφού οι καθολικές κορυφές σχηματίζουν μια κλίκα ενώ οι απομονωμένες κορυφές απαρτίζουν το ανεξάρτητο σύνολο. Η δεντρική αναπαράσταση των κατωφλικών γραφημάτων παίρνει την μορφή της «σαρανταποδαρούσας» για αυτό ονομάζονται και caterpillar γραφήματα.



Εικόνα 10 Ένα κατωφλικό γράφημα και η δεντρική αναπαράσταση του

Διμερή γραφήματα

Διμερή (bipartite graphs) ονομάζονται τα γραφήματα τα οποία μπορούν να διαχωριστούν σε δύο ξένα μεταξύ τους σύνολα U και V (άρα τα σύνολα αυτά είναι ανεξάρτητα σύνολα) τέτοια ώστε κάθε ακμή του γραφήματος να ενώνει μια κορυφή του U με το V. Ισοδύναμα είναι τα γραφήματα που δεν έχουν κύκλους περιττού μήκους.



Εικόνα 11 Διμερές γράφημα

Τέλεια γραφήματα

Τέλεια ονομάζονται τα γραφήματα για τα οποία ο χρωματικός αριθμός κάθε επαγόμενου υπό-γραφήματος ισούται με το μέγεθος της μεγαλύτερης κλίκας του υπό-γραφήματος αυτού. Θεωρήθηκαν για πολλά χρόνια ισοδύναμα με τα γραφήματα τα οποία ούτε αυτά ούτε το συμπλήρωμα τους δεν περιέχουν κύκλους περιττού μήκους 5 ή μεγαλύτερους, κάτι που τελικά αποδείχθηκε το 2002 και έγινε γνωστό ως το *Ισχυρό Θεώρημα Τέλειων Γραφημάτων*. Η κλάση των τέλειων γραφημάτων περιλαμβάνει πολλές μεγάλες οικογένειες γραφημάτων και ενοποιεί τα αποτελέσματα αλγορίθμων χρωματισμού και εύρεσης κλικών σε αυτές. Έτσι σε όλα τα τέλεια γραφήματα το *πρόβλημα του*

χρωματισμού, της εύρεσης μεγιστοτικής κλίκας και *μέγιστοτικού ανεξάρτητου συνόλου* μπορούν να λυθούν σε *πολυωνυμικό χρόνο.*

Comparability Γραφήματα

Comparability ονομάζονται τα μη κατευθυνόμενα γραφήματα που συνδέουν ζευγάρια στοιχείων που είναι συγκρίσιμα μεταξύ τους σε μια μερική διάταξη. Ονομάζονται επίσης και *μεταβατικά κατευθυνόμενα γραφήματα, μερικώς* διατάξιμα *γραφήματα(transitively orientable graphs, partially orderable graphs).* Για κάθε αυστηρά διατεταγμένο σύνολο$(\mathrm{S}, <)$, το comparability γράφημα του είναι ένα γράφημα με κορυφές τα στοιχεία του συνόλου S και ακμές για κάθε ζευγάρι $\{\mathrm{u}, \mathrm{v}\}$ για τα οποία ισχύει $\mathrm{u} < \mathrm{v}$. Ισοδύναμα είναι τα γραφήματα που έχουν την *μεταβατική ιδιότητα(transitive orientation),* δηλαδή υπάρχει μια ανάθεση κατευθύνσεων στις ακμές του γραφήματος τέτοια ώστε αν υπάρχουν κατευθυνόμενες ακμές (x, y) και (y, z) τότε υπάρχει και η ακμή (x, z).

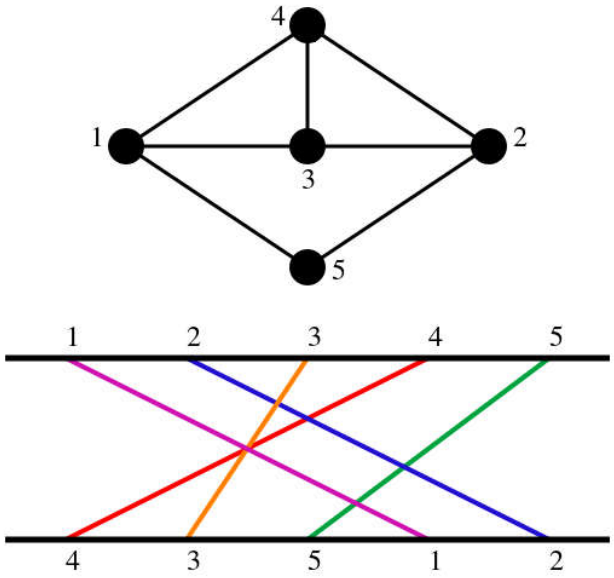
Η κλάση των comparability γραφημάτων είναι μια μεγαλύτερη κλάση που περιέχει τα *πλήρη γραφήματα, τα διμερή γραφήματα, τα συμπληρώματα των interval γραφημάτων, τα permutation γραφήματα, τα σχεδόν κατωφλικά γραφήματα, τα cographs και τα κατωφλικά γραφήματα.* Όλα τα comparability γραφήματα είναι *τέλεια γραφήματα.*

Εικόνα 12 Comparabilty γράφημα και κατευθύνσεις με την μεταβατική ιδιότητα

Permutation γραφήματα

Αν $\sigma = (\sigma_1, \sigma_2 \ldots \sigma_\nu)$ είναι ένα μια μετάθεση των αριθμών από το 1 μέχρι το ν, τότε ορίζουμε ως permutation γράφημα ένα γράφημα με ν κορυφές $\mathrm{v}_1, \mathrm{v}_2 \ldots \mathrm{u}_\mathrm{n}$, και ακμές $\mathrm{v_i v_j}$ για κάθε i, j με $\mathrm{i} < \mathrm{j}$ και $\sigma_\mathrm{i} > \sigma_\mathrm{j}$. Δηλαδή έχουμε ακμή για κάθε αντιστροφή στην μετάθεση. Ορίζοντας τις δύο παράλληλες γραμμές που ορίζουν οι $\mathrm{y} = 0$ και $\mathrm{y} = 1$ και τα ευθύγραμμα τμήματα $\mathrm{s_i}$ με άκρα τα σημεία $(\mathrm{i}, 0)$ και $(\sigma_\mathrm{i}, 1)$ έχουμε την αναπαράσταση τομών (intersection model) του γραφήματος. Η αναπαράσταση αυτή ορίζει ακμές στο γράφημα αν δύο ευθύγραμμά τμήματα τέμνονται.

Τα permutation γραφήματα είναι τέλεια γραφήματα, υπό-κλάση των comparability γραφημάτων και υπέρ-κλάση των διμερών γραφημάτων.

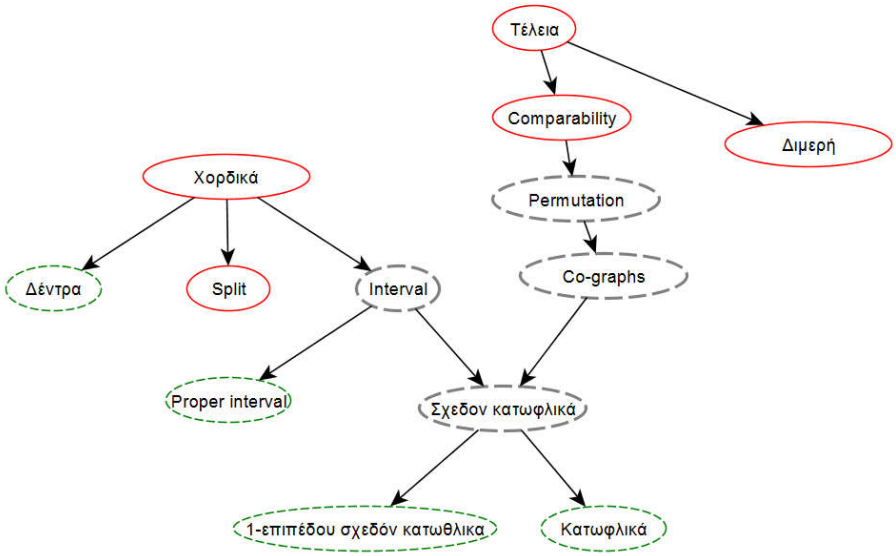


Εικόνα 13 Ένα permutation γράφημα και η αναπαράσταση του

Αν ένα πρόβλημα βρεθεί ότι είναι NP-Hard σε μια κλάση γραφημάτων τότε είναι και σε όλες τις μεγαλύτερες κλάσεις αυτής. Ομοίως αν βρεθεί πολυωνυμικός αλγόριθμος για ένα πρόβλημα μιας κλάσης τότε ο ίδιος ισχύει και για όλες τις μικρότερες κλάσεις αυτής και άρα το πρόβλημα ανήκει στην κλάση P και για αυτές τις κλάσεις γραφημάτων.

Στην παρακάτω εικόνα βλέπουμε την συγγένεια των κλάσεων γραφημάτων που αναφέραμε παραπάνω, το βέλος δείχνει από την μεγαλύτερη στην μικρότερη κλάση. Με κόκκινο χρώμα(και συνεχόμενη γραμμή) είναι σημειωμένες οι κλάσεις για τις οποίες έχει αποδειχθεί ότι το πρόβλημα του πλάτους αποκοπής είναι NP-Complete, με πράσινο είναι οι κλάσεις για τις οποίες το πρόβλημα λύνεται σε πολυωνυμικό χρόνο(και ελαφρώς διακεκομμένη γραμμή), ενώ με γκρι χρώμα (και έντονα

διακεκομμένη γραμμή) είναι σημειωμένες οι κλάσεις για τις οποίες το πρόβλημα παραμένει ανοιχτό.



Εικόνα 14 Κλάσεις γραφημάτων

1.5. Προβλήματα διατάξεων και το πρόβλημα του πλάτους αποκοπής

Στην αλγοριθμική θεωρία γραφημάτων μια μεγάλη κατηγορία προβλημάτων αναζητούν μια γραμμική διάταξη τέτοια ώστε να βελτιστοποιείται(ελαχιστοποιείται ή μεγιστοποιείται) κάποια συνάρτηση κόστους του γραφήματος. Έστω ένα γράφημα $\mathrm{G} = (\mathrm{V}, \mathrm{E})$, μια γραμμική διάταξη φ, και ένα ακέραιο i ορίζουμε τα παρακάτω:

Την *αποκοπή ακμών(edge cut)*: $\theta(i, \varphi, G) = |\{uv \in E, \varphi(v) \leq i \wedge \varphi(v) > i\}|$

Την *τροποποιημένη αποκοπή ακμών(modified edge cut)*:$\zeta(i, \varphi, G) = |\{uv \in E, \varphi(v) \leq i \wedge \varphi(v) > i \wedge \varphi(u) \neq i\}|$

Την *απόσταση κορυφών(vertex separation)*: $\delta(i, \varphi, G) = \{u\colon \varphi(u) \leq i\colon \exists v\colon \varphi(v) > i\colon uv \in E\}|$

Το *μήκος μιας ακμής(length)* $uv \in E$: $\lambda(uv, \varphi, G) = |\varphi(u) \quad f(v)|$

Ως συνάρτηση κόστους ορίζουμε μια συνάρτηση F που συσχετίζει μια γραμμική διάταξη φ στον ακέραιο $F(\varphi, G)$. Ένα πρόβλημα διάταξης επομένως ορίζεται ως η εύρεση μια βέλτιστης διάταξης φ τέτοια ώστε $\mathrm{F}(\varphi\ , G) = \min_{\varphi \in \Phi(G)} \mathrm{F}(\varphi, G) = \mathrm{F}(G)$.

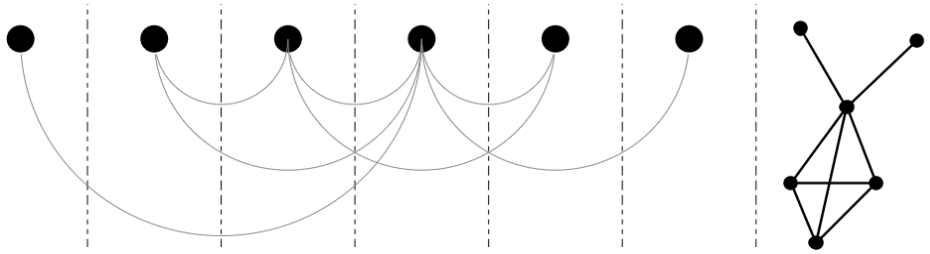
Με βάση αυτές τις συναρτήσεις μπορούμε να ορίσουμε τα παρακάτω γνωστά προβλήματα γραμμικών διατάξεων

Εύρος ζώνης(Bandwidth)	$BW(\varphi, G) = \max_{(uv \in E)} \lambda(uv, \varphi, G)$
Ελάχιστη γραμμική διάταξη(Min Linear Arrangment)	$LA(\varphi, G) = \sum_{uv \in E} \lambda(uv, \varphi, G)$
Πλάτος αποκοπής(cutwidth)	$CW(\varphi, G) = \max\{\theta(i, \varphi, G)\}$
Τροπ. Πλάτος αποκοπής (Modified Cut)	$MC(\varphi, G) = \sum_{i=1}^{n} \zeta(i, \varphi, G)$
Απόσταση κορυφών(Vertex Seperation)	$VS(\varphi, G) = \max\{\delta(i, \varphi, G)\|$
Άθροισμα αποκοπής(Sum Cut)	$SC(\varphi, G) = \sum d(i, \varphi, G)$

Αν και ορίσαμε όλα τα παραπάνω προβλήματα ως προβλήματα βελτιστοποίησης, δηλαδή την εύρεση μιας βέλτιστης διάταξης φ τέτοια ώστε να βελτιστοποιείται κάποια από τις παραπάνω τιμές, εύκολα μπορούμε να θέσουμε τα προβλήματα αυτά και σαν απόφασης αν εισάγουμε και έναν ακέραιο K στην διατύπωση: «Υπάρχει διάταξη φ τέτοια ώστε η συγκεκριμένη συνάρτηση κόστους να είναι $F(\varphi, G) \leq K$».

Όλα τα παραπάνω προβλήματα διατάξεων θα μπορούσαν να λυθούν με τον προφανή αλγόριθμο, να δοκιμάζουμε όλες τις πιθανές διατάξεις. Ο αριθμός των διατάξεων αυτών όμως είναι $O(n!)$ και αμέσως γίνεται εμφανής η μεγάλη δυσκολία που παρουσιάζουν τα προβλήματα αυτά. Πράγματι έχει αποδειχθεί ότι όλα τα παραπάνω προβλήματα είναι NP-Complete για γενικά γραφήματα και άρα δύσκολα στην επίλυση τους. Η αναγωγή για το πρόβλημα του *εύρους ζώνης(bandwidth)* έγινε από τον Παπαδημητρίου το 1976 [32], για το πρόβλημα της *ελάχιστης γραμμικής*

διάταξης(Min LA) από τον Garey το 1976 [14], για το πρόβλημα του *πλάτους αποκοπής(cutwidth)* από τον Gavril το 1977 [15], για το *τροποποιημένο πλάτος αποκοπής(Mod.cutwidth)* από τους Monien και Sudborough το 1988 [29], για την *απόσταση κορυφών(Vertex Seperation)* από τον Lengauer το 1981 [24] και η απόδειξη για το πρόβλημα του *αθροίσματος αποκοπής (sum cut)* μπορεί να βρεθεί στα [10], [27], [16]



Εικόνα 15 Ένα γράφημα και μια γραμμική διάταξη του

Η παρούσα εργασία εστιάζει στο πρόβλημα του πλάτους αποκοπής. Θα ορίσουμε ξανά το πρόβλημα με έναν ισοδύναμο ορισμό ο οποίος θα χρησιμοποιείται στο υπόλοιπο της εργασία: Έστω ένα γράφημα G και ένας ακέραιος k, υπάρχει γραμμική διάταξη σ του G έτσι ώστε ο μέγιστος αριθμός ακμών που τέμνει μια νοητή κάθετη γραμμή ανάμεσα σε δύο κορυφές να είναι μικρότερος ή ίσος με κ. Τυπικά το πλάτος αποκοπής μιας γραμμικής διάταξης κορυφών π ενός γραφήματος $G = (V, E)$ είναι $\max_{u \in V} |\{\{w, x\} \in E | \pi(w) \leq \pi(u) \leq \pi(x)\}|$.

1.6. Εφαρμογές του προβλήματος του πλάτους αποκοπής

Αυτό το πολύ σημαντικό πρόβλημα διάταξης γραφημάτων, προτάθηκε για πρώτη φορά για να ελαχιστοποιήσει τον αριθμό των καναλιών σε ένα κύκλωμα [1] [28], και πιο πρόσφατα βρήκε εφαρμογές σε τομείς όπως η μηχανική πρωτεϊνών [3], η αξιοπιστία δικτύων [22], η αυτόματη αναπαράσταση γραφημάτων [30], η ανάκτηση πληροφορίας [6] και σαν υπό πρόβλημα σε έναν αλγόριθμο για το πρόβλημα TSP(Traveling Salesman Problem) [21]

1.7. Πολυπλοκότητα του προβλήματος σε γενικά γραφήματα

Ο κύριος όγκος της έρευνας πάνω στο πρόβλημα επικεντρώνεται γύρω από προσεγγιστικούς και σταθερής παραμέτρου αλγορίθμους. Υπάρχει ένας προσεγγιστικός πολυωνυμικός αλγόριθμος $O(\log^2 n)$ για γενικά γραφήματα [23] ενώ ο ταχύτερος μέχρι στιγμής αλγόριθμος σταθερής παραμέτρου είναι γραμμικός αλλά εκθετικός ως προς την παράμετρο κ. [35].

Ένας αφελής αλγόριθμος που θα μπορούσε να επιλύσει όλα τα προβλήματα διατάξεων όπως είπαμε είναι της τάξης του $O(n!)$, όμως το 2011 οι Bodlaender, Fomin, Koster, Kratsch και Thilikos [5] παρουσίασαν έναν εκλεπτυσμένο αλγόριθμο που επιλύει πολλά από τα προβλήματα διατάξεων(και μερικά ακόμα) σε χρόνο $O\ (2^n)$ και χώρο $O\ (2^n)$ με δυναμικό προγραμματισμό. Επίσης στην ίδια εργασία

παρουσιάζεται ένας αλγόριθμος τύπου «*διαίρει και βασίλευε*» που επιλύει τα ίδια προβλήματα σε χρόνο $O\ (4^n)$ και πολυωνυμικό χώρο.

Θα δούμε πιο αναλυτικά τον αλγόριθμο αυτό γιατί σε αυτόν βασίζονται πολλές από τις ιδέες που χρησιμοποιήθηκαν στην παρούσα εργασία καθώς και θα δώσουμε μια υλοποίηση του για το πρόβλημα του πλάτους αποκοπής. Πρώτα όμως θα δώσουμε κάποιους σύντομους ορισμούς για τους αλγόριθμους τύπου «διαιρεί και βασίλευε» και δυναμικού προγραμματισμού.

Οι αλγόριθμοι τύπου *«διαίρει και βασίλευε»* διαιρούν το πρόβλημα σε διάφορα υπό-προβλήματα που είναι παρόμοια με αρχικό αλλά μικρότερα σε μέγεθος, επιλύουν αυτά τα υπό-προβλήματα αναδρομικά και στην συνέχεια συνδυάζουν τις διάφορες λύσεις ώστε να συνθέσουν μια λύση του αρχικού προβλήματος. Το μοντέλο *«διαίρει και βασίλευε»* περιλαμβάνει τρία βήματα σε κάθε επίπεδο αναδρομής:

- *Διαιρούμε* το πρόβλημα σε διάφορα υπό-προβλήματα
- «*Κυριεύουμε*» τα υπό-προβλήματα, επιλύοντας τα αναδρομικά
- *Συνδυάζουμε* τις λύσεις των υπό-προβλημάτων ώστε να συνθέσουμε μια λύση του αρχικού προβλήματος.

Ο *δυναμικός προγραμματισμός* είναι μια μέθοδος επίλυσης προβλημάτων μέσω του συνδυασμού των λύσεων κάποιων υπό-προβλημάτων, που δεν είναι όμως ανεξάρτητα μεταξύ τους. Ένας αλγόριθμος δυναμικού προγραμματισμού επιλύει το κάθε υπό-πρόβλημα μόνο μία φορά και αποθηκεύει την λύση αποφεύγοντας τον εκ νέου υπολογισμό κάθε φορά που το υπό-πρόβλημα αυτό συναντάται. Η ιδιότητα αυτή των επικαλυπτόμενων *υπό-προβλημάτων(overlapping sub problems)* βασική για τον δυναμικό προγραμματισμό και αποτελεί την ισχυρότερη ένδειξη

ότι ένα πρόβλημα μπορεί να λυθεί με χρήση δυναμικού προγραμματισμού. Συνήθως χρησιμοποιείται σε προβλήματα βελτιστοποίησης. Η ανάπτυξη ενός αλγορίθμου δυναμικού προγραμματισμού συνήθως ακολουθεί τα παρακάτω βήματα: [9]

- Χαρακτηρίζουμε την δομή μιας βέλτιστης λύσης
- Ορίζουμε αναδρομικά την τιμή μιας βέλτιστης λύσης
- Υπολογίζουμε την τιμή της βέλτιστης λύσης «αναβιβαστικά»(δηλαδή από «κάτω προς τα πάνω» ή από τα «μερικά στα γενικά»)
- Κατασκευάζουμε μια βέλτιστη λύση από τα δεδομένα που έχουμε υπολογίσει.

Έστω $G = (V, E)$ ένα γενικό γράφημα ,π μια γραμμική διάταξη του και $u \in V$ μια κορυφή του. Με $\pi_{<,u}$ συμβολίζουμε το σύνολο των κορυφών που εμφανίζονται αριστερά της u στην διάταξη π : $\pi_{<,u} = \{w \in V |\ \pi(w) \leq \pi(u)\}$. Στην εργασία αυτή [5] παρατηρήθηκε ότι πολλά από τα πολλά από προβλήματα διατάξεων (και μερικά ακόμα που μπορούν να τυποποιηθούν σαν προβλήματα διατάξεων) έχουμε μια συνάρτηση f που σε πολυωνυμικό χρόνο μπορεί να υπολογιστεί και αντιστοιχεί 3 τιμές, το γράφημα $\mathrm{G} = (\mathrm{V}, \mathrm{E})$, ένα υποσύνολο των κορυφών του γραφήματος $S \subseteq V$, και μια κορυφή $\mathrm{u} \in \mathrm{V}$ σε ένα ακέραιο. Τα προβλήματα τότε ισοδυναμούν με τον υπολογισμό των τιμών:

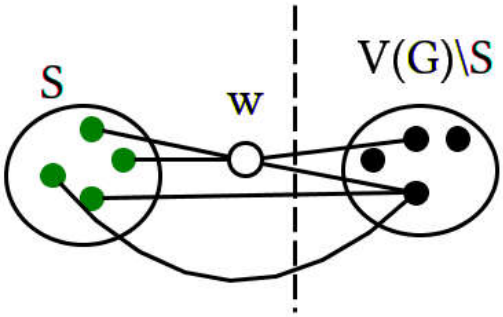
$$\min_{\pi \in \Pi(V)} \max_{u \in V} f(G, \pi_{<,u}, u)$$

$$\min_{\pi \in \Pi(V)} \sum_{u \in V} f\left(G, \pi_{<,u}, u\right)$$

Για το πρόβλημα το πλάτους αποκοπής η συνάρτηση f παίρνει την μορφή:

$$f(G,S,u) = \Big|\{\{w,x\} \in E \mid w \in S \cup \{u\} \wedge x \in V \quad S \quad \{u\}\Big\}|$$

Απέδειξαν ότι προβλήματα που μπορούν να τυποποιηθούν στην παραπάνω μορφή μπορούν να λυθούν σε $O\ (2^n)$ χρόνο και $O\ (2^n)$ χώρο με βάση έναν αλγόριθμο δυναμικού προγραμματισμού. Τυποποιήσαν πολλά προβλήματα στην παραπάνω μορφή δίνοντας για κάθε ένα την συνάρτηση f, όπως *της ελάχιστης γραμμικής διάταξης, της απόστασης κορυφών , του πλάτους αποκοπής, του πλάτους μονοπατιού, του πλάτους δέντρου, του αθροίσματος αποκοπής* κ.α. Η απόδειξη ορθότητας του παραπάνω αλγορίθμου βασίζεται στο γεγονός ότι η συνάρτηση f εξαρτάται από το *σύνολο* S και άρα *όχι από την σειρά των κορυφών μέσα στο* S. Άρα γίνεται άμεσα εμφανές ότι από την στιγμή που δεν εξαρτάται από την σειρά των στοιχείων οι πιθανοί συνδυασμοί είναι $O(2^n)$.



Εικόνα 16 Τυχαίο βήμα του δυναμικού αλγορίθμου $O(2^n)$

Περιγραφικά έστω ένα τυχαίο βήμα του αλγορίθμου όπου έστω $S \subseteq V$ βρίσκονται αριστερά μιας κορυφής $\mathrm{w} \in \mathrm{V} \quad \mathrm{S}$, και φυσικά οι υπόλοιπες $V \quad S \quad \{w\}$ βρίσκονται δεξιά της w , επίσης με άγνωστη σειρά. Μέχρι το σημείο αυτό η χειρότερη τιμή της συνάρτησης f είναι είτε αυτό που υπολογίζουμε για την κορυφή w είτε αυτό που έχουμε υπολογίσει για το σύνολο S σε προηγούμενο βήμα. Άρα δηλαδή αν με A

συμβολίσουμε τον πίνακα που αποθηκεύουμε τις προηγούμενες τιμές έχουμε ότι $A_G(S) = \min_{w \in S} \max\{f(G, S \quad \{w\}, w\}, A_G(S \quad \{w\})\}$. Τελικά παρουσίασαν τον ακόλουθο αλγόριθμο σε μορφή ψευδό-κώδικα:

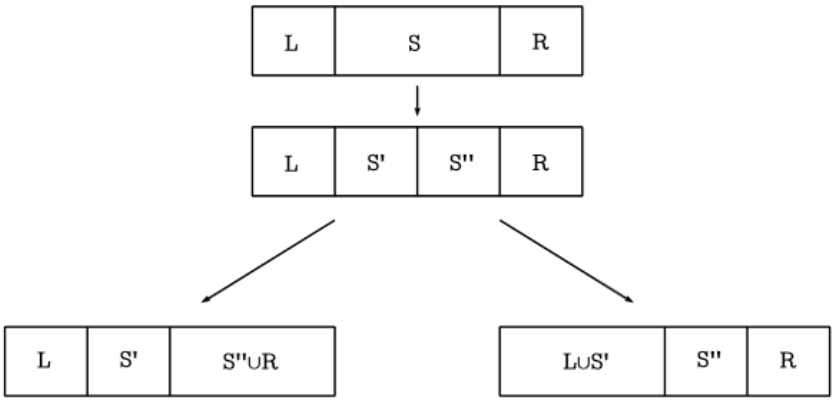
Αλγόριθμος 1: Αλγόριθμος δυναμικού προγραμματισμού, γράφημα G = (V, E)

Set $A(\ ) = \quad \infty$
for $i = 1$ *to* n:
  for *all vertex sets* $S \subset V : \ |S| = i :$
    Set $A(S) = \min_{w \in S} \max\{f(G, S \quad \{w\}, w), A(S \quad \{w\})\}$
  end for
end for
return $A(V)$

Για τον δεύτερο αλγόριθμο που παρουσίασαν στην εργασία βασιστήκαν στην παρατήρηση ότι αν διαχωρίσουμε μια γραμμική διάταξη σε τρία σύνολα, L, S, R έτσι ώστε οι κορυφές L να εμφανίζονται πρώτα (πιο αριστερά), μετά οι κορυφές του S και τέλος οι κορυφές του R, μπορούμε να βρούμε μια διαμέρηση του S σε S', S'' έτσι ώστε οι χειροτέρη τιμή της συνάρτησης f να βρίσκεται είτε στο $L \cup S'$ είτε στο $S'' \cup R$, και άρα μπορούμε να δοκιμάσουμε όλες τις πιθανές διαμερίσεις του S με $|S'| = \lfloor \frac{|S|}{2} \rfloor$, λόγω συμμετρίας. Ο ψευδό-κώδικας για τον αλγόριθμο αυτό δίνεται παρακάτω και εκτελείται σε χρόνο της τάξης $O\ (4^n)$ και πολυωνυμικό χώρο:

Αλγόριθμος 2: Recursive(Γράφημα G, σύνολο κορυφών L, σύνολο κορυφών S)

if $|S| = 1$ **:**
  $S = \{u\}$
  return $f(G, L, u)$
end if
$opt = \infty$
for *all vertex sets* $S' \subseteq S, |S'| = \left\lfloor \frac{|S|}{2} \right\rfloor$ **:**
  $u_1 = Recursive(G, L, S')$
  $u_2 = Recursive(G, L \cup S', S \quad S')$
  $opt = \min\{opt, max\{u_1, u_2\}\}$
end for
return *opt*



Εικόνα 17 Η αναδρομική κλήση του αλγόριθμου διαίρει-και-βασίλευε

1.8. Το πρόβλημα του πλάτους αποκοπής σε κλάσεις γραφημάτων

Για τις κλάσεις των meshes και των πλήρων κ-μερών γραφημάτων υπάρχουν κλειστοί τύποι για τον υπολογισμό του πλάτους αποκοπής. Για τα proper interval γραφήματα υπάρχει ένας απλός αλγόριθμος που ακολουθεί τη φυσική διάταξη των κορυφών [40]. Για τα δέντρα μπορεί να υπολογιστεί σε χρόνο $O(nlogn)$ με έναν αρκετά πολύπλοκο αλγόριθμο [39]. Το πλάτος αποκοπής για γραφήματα με φραγμένο πλάτος δέντρου και μέγιστο βαθμό, μπορεί να υπολογιστεί σε πολυωνυμικό χρόνο μέσω πολύπλοκων μεθόδων [36]. Για διμερή permutation γραφήματα υπάρχει αλγόριθμος γραμμικού χρόνου [18]. Για τα *χορδικά γραφήματα* το πλάτος αποκοπής είναι *NP-complete* [34] λόγω του ότι είναι μεγαλύτερη κλάση των *split γραφημάτων.* Τέλος το πρόβλημα είναι άγνωστο αν είναι NP-Hard για τα interval γραφήματα καθώς και για τα σχεδόν κατωφλικά γραφήματα. Στα κατωφλικά γραφήματα το πρόβλημα του πλάτους αποκοπής αποδεικνύεται ότι είναι γραμμικού χρόνου(με άπληστο αλγόριθμο) [34] Ο αλγόριθμος αυτός είναι απλός και η υλοποίηση του δεν απαιτεί καμία ιδιότητα των κατωφλικών γραφημάτων, άρα μπορεί να εκτελεστεί σε οποιαδήποτε κλάση γραφημάτων ως ευριστικό. Θα δούμε τον αλγόριθμο αυτό στο κεφάλαιο 3 και αντιπαράδειγμα γιατί δεν ισχύει για την κλάση των σχεδόν κατωφλικών γραφημάτων.

Στο παρακάτω πίνακα βλέπουμε συνοπτικά για ποιες από τις παραπάνω κλάσεις γραφημάτων είναι γνωστό αν το κάθε πρόβλημα ανήκει στην κλάση *NP-Complete.*

Πρόβλημα	**Κλάση γραφημάτων**
Εύρος ζώνης(bandwidth)	γενικά δέντρα με μέγιστο βαθμό 3 κατωφλικά
Ελαχ. Γραμ. διάταξη(MinLA)	γενικά διμερή γραφήματα
Πλάτος αποκοπής (cutwidth)	γενικά γραφήματα με μέγιστο βαθμό 3 επίπεδα με μέγιστο βαθμό 3
Τρ. Πλάτος αποκοπής (mod. Cutwidth)	επίπεδα με μέγιστο βαθμό 3
Απόσταση κορυφών(vertex separation)	γενικά χορδικά διμερή
Άθροισμα αποκοπής(sum cut)	γενικά

Στο παρακάτω πίνακα βλέπουμε κλάσεις γραφημάτων για τις οποίες είναι γνωστός πολυωνυμικός αλγόριθμος για κάθε πρόβλημα.

Πρόβλημα	**Κλάση γραφημάτων**	**Πολυπλοκότητα**
Εύρος ζώνης(bandwidth)	Κατωφλικά (με hair-length < 3) interval	$O(nlogn)$ $O(nlogn)$
Ελαχ. Γραμ. διάταξη(MinLA)	Δέντρα Πλήρη k-μερή	$O(n^{\frac{log3}{log2}})$ $O(n + klogk)$
Πλάτος αποκοπής (cutwidth)	Δέντρα Πλήρη k-μερή	$O(nlogn)$ $O(n + klogk)$
Απόσταση κορυφών(vertex separation)	Δέντρα permutation	$O(n)$ $O(n^2)$
Άθροισμα αποκοπής(sum cut)	Δέντρα	$O(n^{1.722})$

ΚΕΦΑΛΑΙΟ 2

2. Εισαγωγικά σε Quasi-Threshold και βέλτιστες διατάξεις

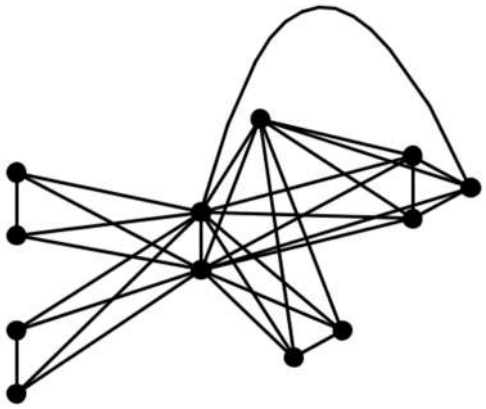
2.1. QT γραφήματα

Ένα γράφημα G ονομάζεται *σχεδόν κατωφλικό (Quasi Threshold)* [38] αν για κάθε επαγόμενο υπογράφημα H του G το πλήθος των μεγιστοτικών κλικών του H είναι ίσος με το μέγιστο μέγεθος ενός ανεξάρτητου συνόλου του H. Ονομάζονται επίσης *Trivially Perfect γραφήματα ή Comparability γραφήματα δέντρων.*

Ισοδύναμα είναι τα γραφήματα που δεν περιέχουν P_4 ή C_4 ως επαγόμενα υπογραφήματα. [17]

Κάθε συνεκτικό σχεδόν κατωφλικό γράφημα έχει μια καθολική κλίκα. [37] Κάθε επαγόμενο υπογράφημα του είναι σχεδόν κατωφλικό γράφημα [7] και επομένως κάθε συνεκτικό επαγόμενο υπογράφημα έχει μια καθολική κλίκα. [17]

Κάθε σχεδόν κατωφλικό γράφημα έχει μια αναπαράσταση ως ριζωμένο δέντρο όπου κάθε μονοπάτι από την ρίζα σε ένα φύλλο είναι μια μεγιστοτική κλίκα



Εικόνα 18 Ένα σχεδόν κατωφλικό γράφημα

Η δεντρική αναπαράσταση ενός σχεδόν κατωφλικού γραφήματος G θα την συμβολίζουμε στο εξής $T(G)$ και έχει τις παρακάτω ιδιότητες: [31]

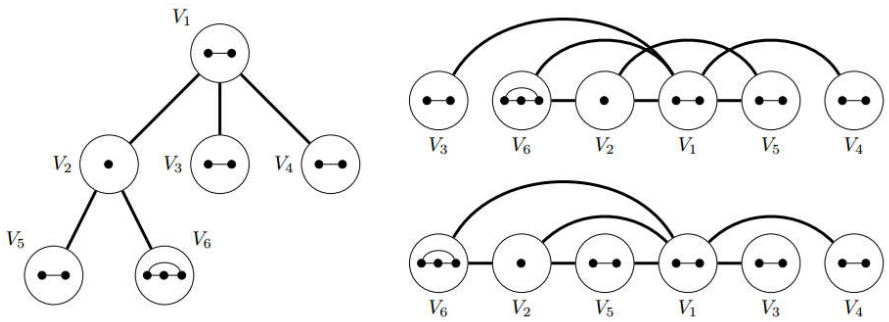
i. Διαμερίζουν το V(G) σε κ κόμβους του $T(G)$ έτσι ώστε $\mathrm{V(G)} = V_1 + \quad V_k$

ii. Κάθε κόμβος $\mathrm{V_i}$ του T(G) είναι είτε φύλλο είτε έχει *τουλάχιστον* δύο παιδιά.

iii. Κάθε κόμβος $\mathrm{V_i}$ του T(G) επάγει μια *κλίκα* στο G

iv. Δύο κορυφές u και v είναι γειτονικές στο G αν και μόνο αν $u, v \in V_i$ για κάποιο κόμβο $\mathrm{V_i}$ στο $T(G)$, είτε $u \in V_i$ και $\mathrm{v} \in \mathrm{V_j}$ και ο κόμβος $\mathrm{V_i}$ είναι πρόγονος του $\mathrm{V_j}$ για δύο κόμβους V_i, V_j του T(G).

Ένα γράφημα είναι σχεδόν κατωφλικό αν έχει μια δεντρική αναπαράσταση με όλες τις παραπάνω ιδιότητες και αυτή η αναπαράσταση είναι μοναδική [31]



Εικόνα 19 Η δεντρική αναπαράσταση του παραπάνω σχεδόν κατωφλικού γραφήματος, και δύο γραμμικές διατάξεις του, η πρώτη με πλάτος αποκοπής 14, η δεύτερη με πλάτος αποκοπής 12

Για έναν κόμβο V_i του T(G) συμβολίζουμε με T_i το υποδέντρο του T(G) με ρίζα το κόμβο V_i.

Όπως αναφέραμε και προηγουμένως για ένα γράφημα G, καλούμε *διάταξη κορυφών(ή γραμμική διάταξη κορυφών)* σ μια αμφιμονοσήμαντη ανάθεση $\sigma: V(G) \rightarrow \{1,2,\dots,n\}$. Για μια τέτοια διάταξη $\sigma =_{def} v_1, v_2, \dots, v_n$ συμβολίζουμε με $v_i \ _\sigma \ v_j$ αν $i < j$ ή ισοδύναμα αν η θέση του v_i είναι μικρότερη από την θέση του v_j.Θεωρούμε τον v_1 ως το πιο αριστερό κόμβο της διάταξης σ και τον u_n ως τον πιο δεξιό αντίστοιχα. Επίσης λέμε ότι k κορυφές εμφανίζονται *διαδοχικά* σε μια διάταξη σ,αν οι θέσεις των κορυφών αυτών είναι οι $\{i+1, i+2, \dots, i+k\}$ για κάποιο $i \in \{1,2,\dots,n \quad k\}$. Γενικότερα λέμε ότι ένα σύνολο $A \subseteq$

$V(G)$ έχει την *διαδοχική ιδιότητα* στη σ αν οι κορυφές που ανήκουν στο A εμφανίζονται συνεχόμενα στη σ.

Με $\Sigma(G)$ συμβολίζουμε το σύνολο όλων των διατάξεων κορυφών ενός γραφήματος G. Για μια τέτοια διάταξη σ και έναν ακέραιο i, $1 \leq i < n$, λέμε ότι μια ακμή *«διαπερνά»* τις θέσεις i και $i+1$ αν το ένα άκρο της είναι στο $G[v_1, \ldots, v_i]$ και το άλλο στο $G[v_{i+1}, \ldots, v_n]$. Με $cut_\sigma(i)$ συμβολίζουμε το πλάτος αποκοπής μεταξύ των θέσεων i και $i+1$ και ισούται με το πλήθος των ακμών που διαπερνούν τις θέσεις i και $i+1$ Επομένως το *πλάτος αποκοπής(cutwidth)* της διάταξης σ είναι $cw_\sigma(G) =_{\mathrm{d}ef} \max_i cut_\sigma(i)$ και το *πλάτος αποκοπής του γραφήματος* G είναι το ελάχιστο από όλα τα δυνατά πλάτη αποκοπής δηλαδή $cw(G) =_{\mathrm{d}ef} \min_{\sigma \in \Sigma(G)} cw_\sigma(G)$.Μια *βέλτιστη διάταξη* σ_{opt} για ένα γράφημα G ορίζουμε ως την διάταξη που έχει πλάτος αποκοπής ίσο με το πλάτος αποκοπής του γραφήματος G.

Με σ^R συμβολίζουμε την *ανάστροφη διατάξη* μια διάταξης σ που ορίζεται ως $\sigma^R(v) =_{\mathrm{d}ef} n \quad \sigma(v) + 1$. Είναι προφανές ότι $cw_\sigma(G) = cw_{\sigma^R}(G)$.

Για μια διάταξη σ και μια κορυφή u στην θέση i, συμβολίζουμε ως $L_\sigma(u)$ τους γείτονες του u που εμφανίζονται πιο αριστερά στηνσ, δηλαδή $L_\sigma(u) =_{\mathrm{d}ef} \{v | v \quad_\sigma u, v \in N(u)\}$.Αντίστοιχα με $R_\sigma(u)$ συμβολίζουμε τους γείτουνες της u που εμφανίζονται δεξιότερα. Για ευκολία επίσης συμβολίζουμε με μικρά γράμματα τον πληθικο αριθμό των συνόλων δηλαδη, $l_\sigma(u) = |L_\sigma(u)|$ και $r_\sigma(u) = |R_\sigma(u)|$. Έστω u και v δύο κορυφές στη σ. Όταν λέμε ότι εναλλάσσουμε τις κορυφές u και v στη σ, παίρνουμε μια νέα διάταξη σ' τέτοια ώστε $\sigma'(u) = \sigma(v)$, $\sigma'(v) = \sigma(u)$, καθώς και $\sigma'(w) = \sigma(w)$ για κάθε $w \neq u, v$.

Για δύο κορυφές u και v του G, ορίζουμε την τιμή $\beta(u,v) =_{def} 1$, αν $uv \in E(G)$ και $\beta(u,v) =_{def} 0$ αν $uv \quad E(G)$.

2.2. Βέλτιστες διατάξεις αποκοπής

Βάση των παραπάνω ορισμών μπορούμε να αποδείξουμε το παρακάτων λήμμα για τις προυποθέσεις με τις οποίες μπορούμε να εναλλάξουμε δύο κορυφές σε μια διάταξη χωρίς να μεγαλώσουμε το πλατος αποκοπής.

Λήμμα 2.1 (Εναλλαγή κορυφών) *Έστω σ μια διάταξη κορυφών του γραφήματος G και έστω u και v δύο διαδοχικές κορυφές στη σ τέτοιες ώστε $u \quad_{\sigma} v$. Για την διάταξη σ' που προκύπτει αν εναλλάξουμε τις κορυφές u και v αν ισχύει ότι:*

- $r_{\sigma}(u) \geq l_{\sigma}(u) + 2\beta(u,v)$ ή $l_{\sigma}(v) \geq r_{\sigma}(v) + 2\beta(u,v)$

τότε $cw_{\sigma'}(G) \leq cw_{\sigma}(G)$.

Αποδειξη. Εφόσον η v υπάρχει σημαίνει ότι $\sigma(u) = i$ γι κάποιο $1 \leq i < n$. Προφανώς *μόνο* το πλάτος αποκοπής μεταξύ των θέσεων i και $i+1$ στη σ' επηρεάζεται από την εναλλαγή των u και v. Επομένως ισχύει ότι $cut_{\sigma'}(j) = cut_{\sigma}(j)$ για κάθε $j \neq i$ και άρα $cw_{\sigma'}(G) \leq cw_{\sigma}(G)$ αν και μόνο αν $cut_{\sigma'}(i) \leq cut_{\sigma}(i)$. Έστω θ_i το σύνολο των ακμών της σ που διαπερνάνε τις θέσεις i και $i+1$ και δεν προσπίπτουν στην u ή στην v. Τότε $cut_{\sigma}(i) = |\theta_i| + r_{\sigma}(u) + l_{\sigma}(v) \quad \beta(u,v)$ και $cut_{\sigma'}(i) = |\theta_i| + r_{\sigma}(v) + l_{\sigma}(u) + \beta(u,v)$.

Έτσι ισχύει $cw_{\sigma}(G) < cw_{\sigma'}(G)$ και επομένως $cut_{\sigma}(i) < cut_{\sigma'}(i)$ από το προηγουμενο επιχείρημα. Τότε $n \geq 3$, αφού για $n = 2$ έχουμε $r_{\sigma}(u) = l_{\sigma}(v) = \beta(u,v)$, $l_{\sigma}(u) = r_{\sigma}(v) = 0$, και το $\sigma' = \sigma^R$, κάτι που

δείχνει ότι $cw_{\sigma'}(G) = cw_{\sigma}(G)$. Διακρίνουμε τις δύο παρακάτω περιπτώσεις.

Έστω u η πιο αριστερή κορυφή στη σ, δηλαδή $\sigma(u) = 1$ και $i = 1$. Τότε $\theta_i = l_{\sigma}(u) = 0$ και $l_{\sigma}(v) = \beta(u, v)$. Παρατηρούμε ότι για την v, αν ισχύει $\beta(u, v) = 0$ και $r_{\sigma}(v) = 0$ το γράφημα G είναι μη συνεκτικό και η κορυφή v είναι απομονομένη κορυφή και άρα $cw_{\sigma'}(G) = cw_{\sigma}(G)$. Επομένως $\beta(u, v) = 1$ ή $r_{\sigma}(v) > 0$ και άρα $r_{\sigma}(v) > \beta(u, v)$ που δείχνει ότι δεν μπορεί να ισχύει η σχέση της υπόθεσης. Αφού $n \geq 3$, η θέση $i + 1$ τόσο στην διάταξη σ όσο και στη σ' υπάρχει. Συγκεκριμένα $cut_{\sigma}(i + 1) = r_{\sigma}(u) + r_{\sigma}(v) \beta(u, v)$. Από την υπόθεση ότι $cut_{\sigma}(i) < cut_{\sigma'}(i)$ έχουμε ότι $cut_{\sigma'}(i) > cut_{\sigma'}(i + 1) = cut_{\sigma}(i + 1)$. Επομένως $r_{\sigma}(v) + \beta(u, v) > r_{\sigma}(u) + r_{\sigma}(v) \beta(u, v)$ που δείχνει ότι $r_{\sigma}(u) < 2\beta(u, v)$ και επομένως οι συνθήκες τις υπόθεσης δεν ισχύουν για τις u και v. Εντελώς συμμετρικά μπορούμε να απόδείκουμε την περίπτωση που η κορυφή v είναι η δεξιότερη στη σ, κάτι που αποδεικνύει ότι αν μια από τις κορυφές u ή v είναι τελική κορυφή στη σ τότε δεν ισχύει καμία από τις υποθέσεις. □

Για την τελευταία περίπτωση υποθέτουμε ότι $1 < i < n \quad 1$ και $n \geq 4$. Κάτι που σημαίνει ότι οι θέσεις $i \quad 1$ και $i + 1$ υπάρχουν στη σ. Από το ορισμό του θ_i , έχουμε ότι $cut_{\sigma}(i \quad 1) = |\theta_i| + l_{\sigma}(u) + l_{\sigma}(v) \quad \beta$ και $cut_{\sigma}(i + 1) = |\theta_i| + r_{\sigma}(u) + r_{\sigma}(u) \quad \beta$. Η γενική υπόθεση ότι$cut_{\sigma}(i) < cut_{\sigma'}(i)$ σημαίνει ότι $cut_{\sigma'}(i) > cut_{\sigma'}(i \quad 1) = cut_{\sigma}(i \quad 1)$ και $cut_{\sigma'}(i) > cut_{\sigma'}(i + 1) = cut_{\sigma}(i + 1)$. Με αντικατάσταση των τιμών έχουμε ότι $l_{\sigma}(v) < r_{\sigma}(v) + 2\beta(u, v)$ και $r_{\sigma}(u) < l_{\sigma}(u) + 2\beta(u, v)$, και άρα καταλήγουμε σε άτοπο με βάση την υπόθεση.

Το παραπάνω λήμμα δεν είναι δύσκολο αλλά παρακάτω θα το εφαρμόσουμε επανηλλημένως για τα επόμενα θεωρήματα. Έστω σ μια διάταξη του G και έστω $A \subseteq V(G)$. Για μια κορυφή $v \in A$, έστω $S_v = \{u | u \quad A, u \quad _\sigma v\}$. Παίρνουμε μια διάταξη $\sigma[A]$ από τη σ τέτοια ώστε για κάθε $v \in A$, $\sigma[A](v) =_{def} \sigma(v) \quad |S_v|$. Πιο απλα η $\sigma[A]$ είναι η διάταξη που προκύπτει από την σ αν διαγράψουμε τις κορυφές $V(G) \backslash A$ και διατηρήσουμε τις σχετικές θέσεις των κορυφών του A. Θεωρούμε ως z μια κορυφή του A που έχει $\sigma[A](z) = \frac{A}{2}$, και τον z τον ονομάζουμε *μεσσαία κορυφή* του A στη σ.

Λήμμα 2.2 *Για ένα γράφημα* G *και μια κορυφή του* v ∈ V(G)*, έστω* z *η μεσαία κορυφή της* N[v] *σε μια βέλτιστη διάταξη για το* G*. Υπάρχει μια βέλτιστη διάταξη στην οποία οι κορυφές* v *και* z *εμφανίζονται συνεχόμενες.*

Απόδειξη. Έστω σ_{opt} μια βέλτιστη διάταξη του G. Αν $v = z$ ή οι v και z εμφανίζονται συνεχόμενα στην σ_{opt} τότε προφανώς ισχύει. Επομένως έστω ότι $v \quad _{\sigma_{opt}} z$. Η περίπτωση $z \quad _{\sigma_{opt}} v$ είναι απολύτως συμμετρική. Έστω L το σύνολο των κορυφών της $N[v]$ που εμφανίζονται αριστερά της z και έστω R το σύνολο των κορυφών της $N[v]$ που εμφανίζονται δεξιά της z. Από τον ορισμό της z, $|L| + \delta = |R|$ για $\delta \in \{0,1\}$. Για $1 \leq i < j \leq n$, ορίζουμε ως $N_{i,j}(v)$ τους γείτονες της v που εμφανίζονται μεταξύ των θέσεων i και j στη σ_{opt} . Έστω $p = \sigma_{opt}(v)$ και $q = \sigma_{opt}(z)$.Τότε $r_{\sigma_{opt}}(v) = \left|N_{p,q}(v)\right| + |R|$ και $l_{\sigma_{opt}}(v) = |L| \quad \left|N_{p,q}(v)\right|$. Αντικαθιστώντας το $|R|$ έχουμε ότι $r_{\sigma_{opt}}(v) = l_{\sigma_{opt}}(v) + 2|N_{p,q}(v)| + \delta$. Εναλλάσσουμε την v με την κορυφή u στην θέση $\sigma_{opt}(v) + 1$ έως ότου $u = z$ και επιβεβαιώνουμε ότι $r_{\sigma_{opt}}(v) \geq l_{\sigma_{opt}}(v) + 2\beta(v,u)$. Αν η u δεν

είναι γειτονική της v τότε $r_{\sigma_{opt}}(v) \geq l_{\sigma_{opt}}(v)$. Έστω τώρα ότι η u είναι γειτονική της v. Αφού όμως $u \neq z$, $N_{p,q}(v) \neq$ και $r_{\sigma_{opt}}(v) \geq l_{\sigma_{opt}}(v) + 2$. Επομένως από το Λήμμα 2.1 παίρνουμε μια νέα διάταξη σ από την σ_{opt} τέτοια ώστε $cw_{\sigma}(G) = cw_{\sigma_{opt}}(G)$ και οι κορυφές v και z να εμφανίζονται συνεχόμενα. □

ΚΕΦΑΛΑΙΟ 3

3. Αλγόριθμος πολυωνυμικού χρόνου για την εύρεσή του πλάτους αποκοπής σε Quasi-Threshold γραφήματα

3.1. Χαρακτηρισμός βέλτιστης διάταξης

Θα προσπαθήσουμε να αναλύσουμε τις δομικές ιδιότητες μιας βέλτιστης διάταξης βάση της δενδρικής αναπαράστασης ενός σχεδόν κατωφλικού γραφήματος. Για την ενότητα αυτή θεωρούμε ότι G είναι ένα σχεδόν κατωφλικό γράφημα και $T(G)$ η δενδρική του αναπαράσταση, με κόμβους διαμέρησης $V_1, \dots, V_k$.

Αρχικά θα δείξουμε ότι κάθε V_i εμφανίζεται συνεχόμενα. Θα τροποποιήσουμε την βέλτιστη σ_{opt} ώστε να αποκτήσουμε μια σ' επίσης βέλτιστη, $cw_{\sigma'}(G) \leq cw_{\sigma_{opt}}(G)$ όπου όμως οι κορυφές κάθε κόμβου V_i να εμφανίζονται συνεχόμενα. Για αυτό ξεκινάμε από τα φύλλα του $T(G)$ και δείχνουμε αναδρομικά ότι κάθε V_i έχει αυτή την ιδιότητα.

Λήμμα 3.1 *Έστω z η μεσαία κορυφή της $N[V_i]$ σε μια βέλτιστη διάταξη σ_{opt} και έστω $z \in V_z$. Αν ο V_z ανήκει στο μονοπάτι από τη ρίζα του $T(G)$ στο V_i τότε υπάρχει μια βέλτιστη διάταξη σ στην οποία ο V_i έχει την διαδοχική ιδιότητα και το $\sigma[V(G) \quad V_i] = \sigma_{opt}[V(G) \quad V_i]$.*

Απόδειξη. Παρατηρούμε ότι για κάθε κορυφή v του V_i ισχύει ότι $N[v] = N[V_i]$ από τις ιδιότητες του $T(G)$. Αν $V_z = V_i$ τότε εφαρμόζουμε το Λήμμα 2.2 σε κάθε κορυφή του V_i και έχουμε την καινούργια διάταξη σ' από την σ_{opt} ώστε να ισχύει $cw_{\sigma'}(G) = cw_{\sigma_{opt}}(G)$ και κάθε κορυφή του V_i να εμφανίζεται συνεχόμενα στην σ'.

Αν ο V_z ανήκει στο μονοπάτι από την ρίζα του $T(G)$ στοV_i τότε ισχύει ότι $N[V_i] \subset N[V_z]$. Εφαρμόζουμε την μετατροπή όπως στο Λήμμα 2.2 στην σ_{opt} και παίρνουμε μια σ' με $cw_{\sigma'}(G) \leq cw_{\sigma_{opt}}(G)$ και το V_i να έχει διαχωριστεί στα σύνολα A_i και B_i έτσι ώστε οι κορυφές του A_i να εμφανίζονται συνεχόμενα στις θέσεις $\sigma_{opt}(z) \quad |A_i|, \ldots, \sigma_{opt}(z) \quad 1$ και οι κορυφές του B_i συνεχόμενα στις θέσεις $\sigma_{opt}(z)+1, \ldots, \sigma_{opt}(z)+|B_i|$. Πιο συγκεκριμένα για κάθε $a_j \in A_i$ και $b_{j'} \in B_i$ μπορούμε να γράψουμε την σ' ως εξής:

$$\sigma' = \ldots, L_i, \ldots, a_{|A_i|}, \ldots, a_1, z, b_1, \ldots, b_{|B_i|}, \ldots, R_i, \ldots,$$

όπου $L_i = N(V_i) \cap L_{\sigma_{opt}}(z)$ και $R_i = N(V_i) \cap R_{\sigma_{opt}}(z)$. Από τον ορισμό της z έχουμε ότι $|B_i| + |R_i| \quad 1 \leq |A_i| + |L_i| \leq |B_i| + |R_i|$. Επιπλεόν για την a_1 έχουμε ότι $r_{\sigma'}(a_1) = |B_i| + |R_i| + 1$ και $l_{\sigma'}(a_1) = |A_i| \quad 1 + |L_i|$. Επομένως $r_{\sigma'}(a_1) \geq l_{\sigma'}(a_1) + 2$ και το ισχύει το Λήμμα 2.1 για την κορυφή a_1. Εναλάσοντας την a_1 με την z, έχουμε μια νεά διάταξη σ'' για την οποία ισχύει ότι $cw_{\sigma''}(G) \leq cw_{\sigma_{opt}}(G)$ και η κορυφή a_1 είναι πλέον η μεσαία κορυφή της $N[V_i]$ (αφού ο α_1 πήρε την θέση του,

δηλαδή $\sigma'\left[N[V_i]\right](z) = \sigma''[N[V_i]](a_1))$. Επομένως η μεσαία κορυφή του $N[V_i]$ ανήκει στο V_i. Άρα για κάθε κορυφή a_j με $2 \leq j \leq |A_i|$ μπορούμε να εφαρμόσουμε το Λήμμα 2.2 και να πάρουμε μια νέα διάταξη σ''' για την οποία ισχύει ότι $cw_{\sigma'''}(G) \leq cw_{\sigma_{opt}}(G)$ και οι κορυφές του V_i εμφανίζονται συνεχόμενες στην σ'''. Ο δευτερος ισχυρισμός ισχύει λόγω της κατασκευής της σ''' αφού οι κορυφές του $V(G) \quad V_i$ διατηρούν τις σχετικές τους θέσεις. □

Λήμμα 3.2 *Έστω* z *η μεσαία κορυφή της* $\mathrm{N[V_i]}$ *σε μια βέλτιστη διάταξη* σ_{opt} *και έστω* $\mathrm{z} \in \mathrm{V_z}$ *. Αν* $\mathrm{V_z} \in \mathrm{T_i}\backslash\{\mathrm{V_i}\}$ *τότε υπάρχει μια βέλτιστη διάταξη* σ *στην οποία οι*$\mathrm{V_i}$ *και* $\mathrm{V_z}$ *έχουν την διαδοχική ιδιότητα και* $\sigma[\mathrm{V(G)} \quad \mathrm{V_i}] = \sigma_{\mathrm{opt}}[\mathrm{V(G)} \quad \mathrm{V_i}]$.

Απόδειξη. Παρατηρούμε ότι $N[V_z] \subset N[V_i]$ από τις ιδιότητες του $T(G)$. Από το Λήμμα 3.1 συμπεραίνουμε ότι οι κορυφές του V_z εμφανίζονται συνεχόμενα στη σ_{opt} και θέλουμε να την τροποποιήσουμε έτσι ώστε και η $\mathrm{V_i}$ να εμφανίζεται συνεχόμενα και δεν θα αλλάξουμε τις σχετικές θέσεις των $V(G) \quad V_i$.

Έστω ότι $A_i = V_i \cap L_{\sigma_{opt}}(z)$, $B_i = V_i \cap R_{\sigma_{opt}}(z)$. Εναλάσουμε κάθε κορυφή a_i του A_i με την κορυφή στην θέση $\sigma_{opt}(a_i) + 1$ μέχρι την θέση $\sigma_{opt}(z_1)$, όπου z_1 είναι η πιο αριστερή κορυφή του V_z στη σ_{opt} . Εφαρμόζοντας μια συμμετρική διαδικασία για το B_i, έχουμε μια σ' με $cw_{\sigma'}(G) \leq cw_{\sigma_{opt}}(G)$ από το Λήμμα 2.1 και $\sigma' = \ldots, A_i, V_z, B_i, \ldots$. Αυτό σημαίνει ότι για κάθε V_j (συμπεριλαμβάνοντας το V_z) που ανήκει στο T_i (χωρίς το V_i) όλες οι κορυφές εμφανίζονται συνεχομένες σε μια βέλτιστη διάταξη. Θέλουμε να βρούμε μια ακόμα βέλτιστη διάταξη σ'' με $cw_{\sigma''}(G) \leq cw_{\sigma'}(G)$ και είτε $\sigma'' = \ldots, V_z, A_i, B_i, \ldots$ είτε $\sigma'' =$

$... , A_i, B_i, V_z ...$. Την πρώτη παίρνουμε αν *μετακινήσουμε* το A_i *προς το* B_i *και την δευτερη αν μετακινήσουμε το* B_i *προς το* A_i.Διαχωρίζουμε επιπλέον την $N(V_i)$ ως εξής: $Z_1 =_{\mathrm{def}} V_z \cap L_{\sigma'}(z)$, $Z_2 =_{def} V_z \cap R_{\sigma'}(z)$, όπου $L_z =_{\mathrm{def}} N(V_z) \cap L_{\sigma'}(z)$, $R_z =_{\mathrm{def}} N(V_z) \cap R_{\sigma'}(z)$. Επίσης το $N(V_i)$ $N(V_z)$ δεν είναι κενό γιατι $N[V_z] \subset N[V_i]$. Έστω $L_i =_{\mathrm{def}} (N(V_i)$ $N(V_z)) \cap L_{\sigma'}(z)$ και $R_i =_{\mathrm{def}} (N(V_i)$ $N(V_z)) \cap R_{\sigma'}(z)$. Με αυτούς τους συμβολισμούς η σ' μπορεί να γραφτεί

$$\sigma' = \ ... , (L_i \cup L_z), ... , A_i, Z_1, z, Z_2, B_i, ... , (R_z \cup R_i), ...$$

Εφόσον η z είναι η μεσσαία κορυφή της $N[V_i]$ ισχύει η εξής εξίσωση:

$$|L_i| + |L_z| + |A_i| + |Z_1| + \delta = |Z_2| + |B_i| + |R_z| + |R_i| \qquad (1)$$

όπου $\delta \in \{0,1\}$. Αναλόγα με τις τιμές των $|L_i|$ και $|R_i|$ είτε μετακινούμε το A_i προς το B_i ή το ανάποδο στην σ'.

Σαν ενδιάμεσο προφανές βήμα θεωρούμε ότι αν μετακινήσουμε τον A_i προς το B_i τότε υποθέτουμε μια βέλτιστη σ_1 τέτοια ώστε $Z_1 \ _{\sigma_1} A_i \ _{\sigma_1} z$. Μια τέτοια διάταξη υπάρχει αφού η z είναι η μεσαία κορυφή της $N[V_i]$. Εξάλου όταν μετακινούμε τον B_i προς τον A_i τότε υποθέτουμε μια βέλτιστη διάταξη σ_2 τέτοια ώστε $z \ _{\sigma_2} B_i \ _{\sigma_2} Z_2$. Για απλούστευση στο εξής θεωρούμε την σ' είτε τη σ_1 είτε τη σ_2 κατά περίπτωση. Διακρίνουμε της παρακάτω περιπτώσεις ανάλογα με τις τιμές των $|L_i|$ και $|R_i|$.

- *Περίπτωση* $|\mathrm{R_i}| \geq |\mathrm{L_i}| + \delta$ *ή* $|\mathrm{R_i}| = |\mathrm{L_i}|$ *και* $\delta = 1$.

Τότε μετακινούμε το A_i προς το B_i. Θα αποδείξουμε ότι η σ'' που προκύπτει είναι όντως μια βέλτιστη διάταξη σε δύο στάδια. Πρώτα παίρνουμε μια ενδιάμεση διάταξη $\sigma_1{}'$ από την σ' στην οποία έχουμε μετακινήσει το A_i δεξιά του z. Έστω η δεξιότερη κορυφή $a_{|A_i|}$ του A_i

στη σ'. Τότε $r_{\sigma'}(a_{|A_i|}) = 1 + |Z_2| + |B_i| + |R_z| + |R_i|$ και $l_{\sigma'}(a_{|A_i|}) = |A_i| \quad 1 + |Z_1| + |L_z| + |L_i|$, και από την εξίσωση (1) έχουμε ότι $r_{\sigma'}(a_{|A_i|}) \geq l_{\sigma'}(a_{|A_i|}) + 2$ αφού $\delta \geq 0$. Επομένως μπορούμε να αλλάξουμε την $a_{|A_i|}$ με την z για να πάρουμε την βέλτιστη διάταξη ${\sigma'}_1$. Σε αυτήν πλέον μεσαία κορυφή της $N[V_i]$ είναι η $a_{|A_i|}$.Επομένως από το Λήμμα 2.2 κάθε κορυφή του $A_i \backslash \{a_{|A_i|}\}$ μπορεί να μετακινηθεί αριστερά της $a_{|A_i|}$ και η σ'_1 να γίνει:

$${\sigma'}_1 = \ \ldots, (L_i \cup L_z), \ldots, Z_1, z, A_i, Z_2, B_i, \ldots, (R_z \cup R_i), \ldots$$

Σε δευτερη φάση αλλάζουμε τις κορυφές του A_i με αυτές τουZ_2. Έστω $a_1, \ldots, a_p, z_1, \ldots, z_q$ είναι η ανάλυση του $A_i \cup Z_2$ στη ${\sigma'}_1$ όπου $p = |A_i|$ και $q = |Z_2|$.

Έστω ότι $|\mathrm{R_i}| \geq |\mathrm{L_i}| + \delta$. Εφαρμόζουμε το Λήμμα 2.1 και έχουμε ότι $l_{{\sigma'}_1}(z_1) \geq r_{{\sigma'}_1}(z_1) + 2$. Αναλυτικά $l_{{\sigma'}_1}(z_1) = |A_i| + 1 + |Z_1| + |L_z|$ και $r_{{\sigma'}_1}(z_1) = |Z_2| \quad 1 + |B_i| + |R_z|$ και από την εξίσωση (1) $l_{{\sigma'}_1}(z_1) \geq r_{{\sigma'}_1}(z_1) + 2$ σημαίνει ότι $|R_i| \geq |L_i| + \delta$ το οποίο ισχύει από την υπόθεση ότι μετακινούμε το A_i προς το B_i. Επομένως αλλάζουμε την z_1 με την a_p και παίρνουμε μια καινούργια βέλτιστη διάταξη ${\sigma'}_2$. Για κάθε κορυφή $z_j \in Z_2$ με $2 \leq j \leq q$, παρατηρούμε ότι: $l_{{\sigma'}_2}(z_j) = l_{{\sigma'}_1}(z_1) + j \quad 1$ και $r_{{\sigma'}_2}(z_j) = r_{{\sigma'}_1}(z_1) \quad (j \quad 1)$. Αυτό σημαίνει ότι $l_{{\sigma'}_2}(z_j) \geq r_{{\sigma'}_2}(z_j) + 2$ επομένως $|R_i| \geq |L_i| + \delta$, και επομένως από το Λήμμα 2.1 αλλάζουμε κάθεz_j με την a_p. Έστω ${\sigma'}_{q+1}$ είναι η διάταξη που προκύπτει και $a_1, \ldots, a_{p\ 1}, z_1, \ldots, z_q, a_p$ είναι η ανάλυση του $A_i \cup Z_2$ στην ${\sigma'}_{q+1}$. Τότε για κάθε κορυφή $a_{j'}$ με $1 \leq j' \leq p \quad 1$ έχουμε ότι $r_{{\sigma'}_{q+1}}(a_{j'}) = r_{\sigma'}(a_p) + (p \quad j')$ και $l_{{\sigma'}_{q+1}}(a_{j'}) = l_{\sigma'}(a_p) \quad (p \quad j')$. Αφού όμως $(p \quad j') > 0$ και $r_{\sigma'}(a_p) \geq l_{\sigma'}(a_p) + 2$, έχουμε ότι $r_{{\sigma'}_{q+1}}(a_{j'}) \geq l_{{\sigma'}_{q+1}}(a_{j'}) +$

2. Άρα αλλάζουμε κάθε κορυφή του $A_i \backslash a_p$ με κάθε κορυφή του Z_1 στη ${\sigma'}_{q+1}$ και παίρνουμε την ${\sigma'}_{p+q+1}$ που είναι και βέλτιστη από το Λήμμα 2.1. Επομένως η ${\sigma'}_{p+q+1}$ έχει τις απαραίτητες ιδιότητες και ολοκληρώνεται η περίπτωση του $|R_i| \geq |L_i| + \delta$.

Έστω ότι $|\mathrm{R_i}| = |\mathrm{L_i}|$ *και* $\delta = 1$. Τότε η εξίσωση (1) γίνεται $|L_z| + |A_i| + |Z_1| + 1 = |Z_2| + |B_i| + |R_z|$. Εδώ θα μετακινήσουμε όλες τις κορυφές του A_i με τις κορυφές του Z_2. Εδώ όμως η μετακίνηση θα γίνει σε ένα βήμα αφού δεν είναι εφαρμόσιμο το Λήμμα 2.1. Έστω ${\sigma'}_2$ ή τελική διάταξη:

$$ {\sigma'}_2 = \ \ldots, (L_i \cup L_z), \ldots, Z_1, z, Z_2, A_i, B_i, \ldots, (R_z \cup R_i), \ldots $$

Θα δείξουμε ότι $cw_{{\sigma'}_2}(G) = cw_{{\sigma'}_1}(G)$ και άρα η ${\sigma'}_2$ είναι μια βέλτιστη διάταξη. Στην διάταξη ${\sigma'}_1$ το μέγιστο πλάτος αποκοπής ανάμεσα στις θέσεις των $A_i \cup Z_2$ είναι στη δεξιότερη κορυφή του A_i και την αριστερότερη του Z_2 . Συγκεκριμένα θα δείξουμε ότι $cut_{{\sigma'}_1}(a_p) = \max_{1 \leq j \leq p, 1 \leq t \leq q}\{cut_{{\sigma'}_1}(a_j), cut_{{\sigma'}_1}(z_t)\}$. Έστω θ οι ακμές που δεν προσπίπτουν σε καμία κορυφή των $A_i \cup Z_2$. Έχουμε ότι $cut_{{\sigma'}_1}(a_p) = \theta + p(p + |L_i|) + (p + q)(1 + |Z_1| + |L_z|)$. Έστω μια τυχαία θέση ${\sigma'}_1(a_j)$ όπου $1 \leq j \leq p$. Με αυτές τις υποθέσεις εχουμε ότι:

$$\begin{aligned} cut_{{\sigma'}_1}(a_j) &= \theta + j(p \quad j) + j(q + |B_i| + |R_z| + |R_i|) \\ &+(p \quad j)(1 + |Z_1| + |L_z| + |L_i|) + q(1 + |Z_1| + |L_z|) \\ &= \theta + 2pj \quad j^2 + p|L_i| + (p + q)(1 + |Z_1| + |L_z|). \end{aligned}$$

Επομένως το $cut_{{\sigma'}_1}(a_j)$ έχει μεγιστο όταν $j = p$ και ότι$cut_{{\sigma'}_1}(a_p) = \max_{1 \leq j \leq p} cut_{{\sigma'}_1}(a_j)$. Ομοίως ορίζουμε το πλάτος αποκοπής και για τις

κορυφές του Z_2. Για ευκολία ορίζουμε $z_0 =_{def} a_p$ έτσι ώστε ${\sigma'}_1(z_0) = {\sigma'}_1(a_p)$. Τότε για μια τυχαία θέση ${\sigma'}_1(z_t)$ όπου $0 \leq t \leq q$ ισχύει ότι:

$$cut_{{\sigma'}_1}(z_t) = \theta + t(q \quad t) + t(|B_i| + |R_z|)$$
$$+(q \quad t)(p + 1 + |Z_1| + |L_z|) + p(|B_i| + |R_z| + |R_i|)$$
$$= \theta \quad t^2 + (p + q)(p + 1 + |Z_1| + |L_z|) + p(|R_i| \quad q).$$

Επομένως το $cut_{{\sigma'}_1}(z_t)$ παρουσιάζει μέγιστο για $t = 0$ και ισχύει ότι $cut_{{\sigma'}_1}(a_p) = \max_{0 \leq t \leq q} cut_{{\sigma'}_1}(z_t)$. Επομένως $cut_{{\sigma'}_1}(a_p) = \max cut_{{\sigma'}_1}(A_i \cup Z_2)$.

Στην θεμιτή διάταξη ${\sigma'}_2$ παρατηρούμε ότι $cut_{{\sigma'}_2}(v) = cut_{{\sigma'}_1}(v)$ για κάποια κορυφή $v \in V(G) \backslash (A_i \cup Z_2)$. Θα δείξουμε ότι $cw_{{\sigma'}_2}(G) = cw_{{\sigma'}_1}(G)$ αποδεικνύοντας ότι $\max cut_{{\sigma'}_2}(Z_2 \cup A) = cut_{{\sigma'}_1}(a_p)$. Τώρα ορίζουμε ως $z_0 =_{def} z$ και ${\sigma'}_2(z_0) = {\sigma'}_2(z)$. Για τις κορυφές του Z_2 στη ${\sigma'}_2$ σε μια τυχαία θέση ${\sigma'}_2(z_{t'})$ όπου $0 \leq t' \leq q$ ισχύει:

$$cut_{{\sigma'}_2}(z_{t'}) = \theta + t'(q \quad t') + t'(p + |B_i| + |R_z|)$$
$$+(q \quad t)(1 + |Z_1| + |L_z|) + p(1 + |Z_1| + |L_z| + |L_i|)$$
$$= \theta + 2pt' \quad {t'}^2 + (p + q)(1 + |Z_1| + |L_z|) + p|L_i|.$$

Επομένως το $cut_{{\sigma'}_2}(z_{t'})$ μεγιστοποιείται για $t' = 0$ ή $t' = p$ και άρα για ${t'}_{max} =_{def} \max\{p, q\}$,το $cut_{{\sigma'}_2}(z_{{t'}_{max}})$ μεγιστοποιείται για ολόκληρο το $cut_{{\sigma'}_2}(z_{t'})$. Θα πρέπει να επιβεβαιώσουμε ότι και οι δύο αυτές τιμές είναι $cut_{{\sigma'}_2}(z_{{t'}_{max}}) \leq cut_{{\sigma'}_1}(a_p)$.'Οντως για ${t'}_{max} = p$ έχουμε ότι

$$cut_{{\sigma'}_2}(z_p) = \theta + p^2 + (p + q)(1 + |L_z| + |Z_1|) + p|L_i|$$
$$= \theta + p(p + |L_i|) + (p + q)(1 + |Z_1| + |L_z|) =$$

$cut_{{\sigma'}_1}(a_p)$,

Και για $t'_{max} = q$ έχουμε

$$cut_{\sigma'_2}(z_q) = \theta + 2pq \quad q^2 + (p+q)(1 + |L_z| + |Z_1|) +$$

$p|L_i|$

$$\leq \theta + p(p + |L_i|) + (p+q)(1 + |Z_1| + |L_z|) =$$

$cut_{\sigma'_1}(a_p)$.

Άρα $cut_{\sigma'_2}(Z_2) \leq cut_{\sigma'_1}(a_p)$. Για τις κορυφές του A_i στη σ'_2 ορίζουμε ότι $a_0 =_{def} z_q$ τέτοι ώστε $\sigma'_2(a_0) = \sigma'_2(z_q)$. Έστω $a_{j'}$ η κορυφή του $A_i \cup \{a_0\}$ όπου $0 \leq j' \leq p$. Τότε για μια τυχαία θέση $\sigma'_2(a_{j'})$ ισχύει ότι

$$cut_{\sigma'_2}(a_{j'}) = \theta + j'(p \quad j') + j(|B_i| + |R_z| + |R_i|)$$
$$+(p \quad j')(q + 1 + |Z_1| + |L_z| + |L_i|) + q(|B_i| + |R_z|)$$
$$= \theta + 2(p \quad q)j' \quad j'^2 + (p+q)(1 + |Z_1| + |L_z|)$$
$$+p(q + |L_i|) + q(p \quad q).$$

Όπου χρησιμοποιήσαμε την υπόθεση ότι $|R_i| = |L_i|$ και $\delta = 0$. Άρα αν $p > q$ τότε το $cut_{\sigma'_2}(a_{j'})$ παρουσιάζει μέγιστο για $j' = p \quad q$ και αν $p \leq q$ τότε το $cut_{\sigma'_2}(a_{j'})$ παρουσιάζει μέγιστο για $j' = 0$. Παρατηρούμε ότι $cut_{\sigma'_2}(a_0) = cut_{\sigma'_2}(z_q)$για το οποίο έχουμε ήδη δείξει ότι $cut_{\sigma'_2}(z_q) \leq cut_{\sigma'_1}(a_p)$. Επιβεβαιώνουμε ότι $cut_{\sigma'_2}(a_{p\ q}) = cut_{\sigma'_1}(a_p)$:

$$cut_{\sigma'_2}(a_{p\ q}) = \theta + (p \quad q)^2 + (p+q)(1 + |Z_1| + |L_z|)$$
$$+p(q + |L_i|) + q(p \quad q)$$
$$= \theta + p(p + |L_i|) + (p+q)(1 + |Z_1| + |L_z|) =$$

$cut_{\sigma'_1}(a_p)$.

Επομένως $cut_{\sigma'_2}(v) \leq cut_{\sigma'_1}(v)$ για καθε $v \in A_i \cup Z_2$ και αφού η σ'_1 είναι μια βέλτιστη διάταξη τότε και η σ'_2 είναι επίσης βέλτιστη με τις θεμιτές ιδιότητες.

– *Περίπτωση* $|R_i| \leq |L_i| + \delta \quad 2 \;$ *ή* $\; |R_i| = |L_i| \quad 1$ *και* $\delta = 0$.

Σε αυτή την περίπτωση μετακινούμε το B_i προς το A_i. Η διάταξη σ' έχει ως εξής:
$\sigma' = \; \ldots, (L_i \cup L_z), \ldots, A_i, z_1, \ldots, z_q, z, b_1, \ldots, b_p, Z_2, \ldots, (R_z \cup R_i), \ldots \;,$

Όπου $Z_1 = \{z_1, \ldots, z_q\}$ και $B = \{b_1, \ldots, b_p\}$. Επίσης βλέπουμε ότι $l_{\sigma'}(b_1) \leq r_{\sigma'_1}(b_1) + 2$. Από τις σχέσεις $l_{\sigma'}(b_1) = 1 + |Z_1| + |A_i| + |L_i| + |L_z|$ και $r_{\sigma'}(b_1) = |B_i| \quad 1 + |Z_2| + |R_z| + |R_i|$ και την εξίσωση (1) έχουμε ότι $l_{\sigma'}(b_1) \leq r_{\sigma'_1}(b_1) + 2$ πρέπει $\delta \geq \quad 1$ κάτι που ισχύει από τον ορισμό του δ. Άρα μπορούμε να αλλάξουμε τη b_1 με τη z στη σ' ώστε να πάρουμε μια σ'_1 που είναι βέλτιστη από Λήμμα 2.1. Τώρα η μεσαία κορυφή της $N[V_i]$ στη σ'_1 έγινε η b_1και από το Λήμμα 2.2 οι κορυφές του $B \backslash \{b_1\}$ μεταβαίνουν στις θέσεις $\sigma'_1(b_1) + 1, \ldots, \sigma'_1(b_1) + p \quad 1$ και έχουμε την καινούργια βέλτιστη διάταξη σ'_2.

Έστω ότι $|\mathrm{R_i}| \leq |\mathrm{L_i}| + \delta \quad 2$. Από την εξίσωση (1) έχουμε ότι $|B_i| + |R_z| + |Z_2| \geq |A_i| + |L_z| + |Z_1| + 2 \quad$. Θέλουμε να μετακινούμε επανηλήμενα την δεξιότερη κορυφή του Z_1 με τις κορυφές του B_i μέχρι μια κορυφή του B_i να γίνει η μεσαία της $N[V_i]$. Αναλυτικά για κάθε $j = 0, \ldots, q \quad 1$ έστω $t_j =_{def} \min\{j + 2, p\}$ όπου $p = |B_i|$. Ξεκινώντας από την z_q, θεωρούμε την κορυφή $z_{q \;\; j}$ και εφαρμόζουμε t_j δεξιές εναλλαγές με τις κορυφές του B_i. Θα δείξουμε ότι από κάθε τέτοια εναλλαγή προκύπτει μια βέλτιστη διάταξη. Έστω η k_j-ιοστή εναλλάγη για την κορυφή $z_{q \;\; j}$ όπου $1 \leq k_j \leq t_j$ ορίζοντας ως $\sigma_{k_j,j}$ την προκύπτουσα

διάταξη. Για λόγους πληρότητας ορίζουμε ότι $\sigma_{0,0} =_{def} \sigma'_2$. Στην $\sigma_{k_j\ 1,j}$ υπάρχουν $k_j \quad 1$ κορυφές του B_i αριστερότερα της $z_{q\ j}$ και $|Z_1| \quad j \quad 1$ κορυφές του Z_1 αριστερότερα της $z_{q\ j}$.Άρα έχουμε ότι $r_{\sigma_{k_j\ 1,j}}(z_{q\ j}) = B \quad (k_j \quad 1) + j + 1 + |Z_2| + |R_z|$ και $l_{\sigma_{k_j\ 1,j}}(z_{q\ j}) = (k_j \quad 1) + |Z_1| \quad j \quad 1 + |A_i| + |L_z|$.Επομένως μαζί με την σχέση $r_{\sigma_{k_j\ 1,j}}(z_{q\ j}) \geq l_{\sigma_{k_j\ 1,j}}(z_{q\ j}) + 2$ πρέπει να ισχύει ότι $|B_i| + |R_z| + |Z_2| \geq |A_i| + |L_z| + |Z_1| + 2(k_j \quad j \quad 1)$. Εφόσον $k_j \quad j \quad 1 \leq 1$, $|A_i| + |L_z| + |Z_1| + 2(k_j \quad j \quad 1) \leq |A_i| + |L_z| + |Z_1| + 2$ και από την υπόθεση ότι $|R_i| \leq |L_i| + \delta \quad 2$ ισχύει η ανισότητα για την $z_{q\ j}$. Απομένως από το Λήμμα 2.1 η $\sigma_{k_j,j}$ είναι μια βέλτιστη διάταξη. Έστω τώρα η διάταξη $\sigma_{t_j,j}$ για την $z_{q\ j}$.Απο την επιλογή του t_j, είτε όλες οι κορυφές του B_i είναι πιο αριστερα της $z_{q\ j}$ και συνεχίζουυμε με την επόμενη κορυφή $z_{q\ j\ 1}$, είτε η $z_{q\ j}$ και η b_{j+2} ήταν η τελαυταία εναλλαγή. Στην προκειμένη περίπτωση η $\sigma_{t_j,j}$ έχει ως εξής:

$$\sigma_{t_j,j} = \langle \dots, z_1, \dots, z_{q\ j\ 1}, b_1, \dots, b_{j+2}, z_{q\ j}, b_{j+3}, \dots, b_p, z_{q\ j+1}, \dots, z_q, z, \dots \rangle$$

Από εκεί παρατηρούμε ότι η b_{j+2} είναι η μεσαία κορυφή της $N[V_i]$ και εφαρμόζοντας το Λήμμα 2.2 έχουμε μια βέλτιστη διάταξη στην οποία η $z_{q\ j}$ είναι δεξιότερα της b_p. Εφαρμόζοντας την ίδια διαδικάσία για κάθε $z_{q\ j}$, καταλήγουμε σε μια βέλτιστη διάταξη όπου οι κορυφές των V_i και V_z εμφανίζονται συνεχόμενες.

Έστω ότι $|\mathrm{R_i}| = |\mathrm{L_i}| \quad 1$ *και* $\delta = 0$. Από την εξίσωση (1) έχουμε ότι $|A_i| + |L_z| + |Z_1| + 1 = |B_i| + |R_z| + |Z_2|$. Μετακινούμε είτε την δεξιότερη κορυφή του Z_1 δεξιά είτε την αριστερότερη κορυφή του B_i

αριστερά. Συμβολίζουμε $\sigma_{j,t}$ την ενδιάμεση διάταξη και $\sigma_{0,0} =_{def} \sigma'_2$. Έστω $z_{q\ \ j}$ η πιο δεξιά κορυφη του Z_1 τέτοια ώστε $R_{\sigma_{j,t}}(z_{q\ \ j}) \cap B_i \neq$ με $0 \leq j \leq q \quad 1$. Ομοίως έστω b_{t+1} η πιο αριστερή κορυφή του B_i τέτοια ώστε $L_{\sigma_{j,t}}(b_{t+1}) \cap Z_1 \neq \quad$ με $0 \leq t \leq p \quad 1$. Αυτό σημαίνει ότι δεξιά της $z_{q\ \ j}$ υπάρχουν $|B_i| \quad t$ κορυφές του B_i και j κορυφές του Z_1. Ανάλογα υπάρχουν t κορυφές του B_i και $|Z_1| \quad j$ κορυφές του Z_1 αριστερότερα της b_{t+1}. Αναλυτικότερα η $\sigma_{j,t}$ γράφετε ως εξής:

$$\sigma_{j,t} = \ ..., b_1, ..., b_t, z_1, ..., z_{q\ \ j}, b_{t+1}, ..., b_p, z_{q\ \ j+1}, ..., z_q, z, ...$$

Θα δείξουμε ότι για κάθε $0 \leq j \leq q \quad 1$ και $0 \leq t \leq p \quad 1$ είτε η $z_{q\ \ j}$ ή η b_{t+1} πληρούν τις προυποθέσεις του Λήμματος 2.1 και μπορούμε να έχουμε μια βέλτιστη διάταξη της μορφής $\sigma_{j+1,t}$ ή $\sigma_{j,t+1}$ αλλάζοντας την $z_{q\ \ j}$ με την b_{t+1}. Ισχύει ότι

$$r_{\sigma_{j,t}}(z_{q\ \ j}) = |B_i| \quad t + j + 1 + |Z_2| + |R_z| \ \kappa\alpha\iota$$

$$l_{\sigma_{j,t}}(z_{q\ \ j}) = |Z_1| \quad j \quad 1 + t + |A_i| + |L_z|$$

Αντικαθιστόντας με την υπόθεση $r_{\sigma_{j,t}}(z_{q\ \ j}) \geq l_{\sigma_{j,t}}(z_{q\ \ j}) + 2$ πρέπει να ισχύει ότι $2j + 1 \geq 2t$. Αρά για $2j + 1 \geq 2t$ έχουμε μια βέλτιστη διάταξη από την δευτερη υπόθεση του Λήμματος 2.1. Τα δείξουμε ότι αν $r_{\sigma_{j,t}}(z_{q\ \ j}) < l_{\sigma_{j,t}}(z_{q\ \ j}) + 2$ τότε παίρνουμε μια βέλτιστη διάταξη από την πρώτη υπόθεση του Λήμματος 2.1. Για b_{t+1} ισχύει ότι

$$\begin{aligned} l_{\sigma_{j,t}}(b_{t+1}) &= l_{\sigma_{j,t}}(z_{q\ \ j}) + 1 + |L_i| \ \kappa\alpha\iota \ r_{\sigma_{j,t}}(b_{t+1}) \\ &= r_{\sigma_{j,t}}(z_{q\ \ j}) \quad 1 + |R_i|. \end{aligned}$$

Με αντικατάσταση της υπόθεσης $|R_i| = |L_i| \quad 1$, έχουμε ότι $l_{\sigma_{j,t}}(b_{t+1}) \geq r_{\sigma_{j,t}}(b_{t+1}) + 2$. Επομένως σε κάθε περίπτωση αλλάζουμε την

$z_{q\ j}$ με την b_{t+1} εώς ότου $j = q \quad 1$ ή $t = p \quad 1$ και κάθε ενδιάμεση διάταξη είναι βέλτιστη και η τελευταιά αυτών να πληρεί τις θεμιτές ιδιότητες.

Άρα σε όλες τις περιπτώσεις μπορούμε να έχουμε μια βέλτιστη διάταξη όπου πληρουνται θεμιτές ιδιότητες για τα V_i και V_z. □

Λήμμα 3.3 *Υπάρχει μια βέλτιστη διάταξη τέτοια ώστε οι κορυφές κάθε* $\mathrm{V_i}$ *να εμφανίζονται συνεχόμενες*

Απόδειξη. Για κάθε V_i έστω z η μεσαία κορυφή της $\mathrm{N[V_i]}$, ενώ έστω $z \in V_z$. Ξεκινώντας από τα φύλλα του $T(G)$ θα δείξουμε ότι αναδρομικά κάθε κόμβος V_i έχει τις κορυφές τους συνεχόμενες. Από τις ιδιότητες του $T(G)$ το V_z ανήκει είτε στο μονοπάτι από την ρίζα του $T(G)$ στο V_i είτε από στο υποδέντρο του $T(G)$ με ρίζα το V_i. Στην πρώτη περίπτωση εφαρμόζουμε το Λήμμα 3.1 ενώ στην δεύτερη το Λήμμα 3.2.

□

Άρα για το πρόβλημα του πλάτους αποκοπής σε QT-γραφήματα μια βέλτιστη διάταξη παρουσιάζει *συνεχόμενες (και συγκεντωμένες)* τις κορυφές κάθε κόμβου της δεντρικής του αναπαράστασης. Έτσι για να λύσουμε πλήρως το πρόβλημα πρέπει να υπολογίσουμε τις θέσεις των κόμβων αυτών σε μια γραμμική διάταξη και όχι όλων των κορυφών του γραφήματος. Άρα έχουμε μειώσει την είσοδο του προβλήματος από $\mathrm{n} = |\mathrm{V}|$ στο $N = k$ όπου k ο αριθμός των κόμβων της διαμέρησης του γραφήματος στην δεντρική του αναπαράσταση $\mathrm{V}_1, V_2 \ldots, V_k$.

3.2. Πολυώνυμικος αλγόριθμος για 1 επίπεδο

Τα σχεδόν κατωφλικά γραφήματα που η δεντρική τους αναπαράσταση έχει βάθος ένα, ονομάζονται superfragile γραφήματα. Από το ορισμό τους τα γραφήματα αυτά έχουν σαν δεντρική αναπαράσταση έναν *καθολικό κόμβο(universal)*, που θα τον συμβολίζουμε στο εξής U, και $N \quad 1$ επιπλέον κόμβους που θα τους συμβολίζουμε $\mathrm{C}_1, C_2 \ldots C_{(N\ 1)}$. Επίσης από τον ορισμό τους παρατηρούμε ότι κάθε κορυφή του καθολικού κόμβου συνδέεται με κάθε κορυφή κάθε κόμβου C_i ενώ οι κορυφές των κόμβων αυτών συνδέονται με τις κορυφές του καθολικού κόμβο και μόνο με αυτές.

Σύμφωνα με το Λήμμα 3.3 υπάρχει μια βέλτιστη διάταξη για τα γραφήματα αυτά που όλες οι κορυφές των C_i και της U εμφανίζονται συνεχόμενες. Για ένα σχεδόν κατωφλικό γράφημα ενός επιπέδου G, με δεντρική αναπαράσταση $T(G) = (U, C_1 \ldots, C_{N\ 1})$ ορίζουμε ως *διάταξη κόμβων* σ μια αμφιμονοσήμαντη ανάθεση σ: T(G) $\rightarrow$ {1,2, ... , N}. Για μια τέτοια διάταξη $\sigma =_{def} C_1, C_2 \ldots, \mathrm{U}, \ldots, C_{N\ 1}$ συμβολίζουμε με $C_i \quad {}_{\sigma} C_j$ αν $i < j$ ή ισοδύναμα αν η θέση του C_i είναι μικρότερη από την θέση του C_j. Με το όρο *διάταξη* στο εξής θα εννοούμε μια διάταξη των κόμβων και όχι των κορυφών του γραφήματος. Για απλότητα και πάλι θα συμβολίζουμε με μικρά γράμματα το πληθικό αριθμό ενός κόμβου, δηλαδή $\mathrm{c_i} = |C_i|$.

Για να μπορέσουμε να λύσουμε το πρόβλημα του πλάτους αποκοπής σε πολυωνυμικό χρόνο χρειαζόμαστε να ξέρουμε την μορφή που θα έχει μια βέλτιστη διάταξη. Για την μορφή αυτή θα μας βοηθήσει το παρακάτω λήμμα.

Λήμμα 3.4 *Έστω* G *ένα σχεδόν κατωφλικό γράφημα ενός επιπέδου,* $\mathrm{T(G)}$ *ή δεντρική αναπαράσταση του και* σ *μια βέλτιστη διάταξη των κόμβων του* $\mathrm{T(G)}$ *και* $\mathrm{C_i, C_j}$ *δύο διαδοχικοί κόμβοι στην διάταξη αυτή. Για την διάταξη* σ *η οποία προκύπτει αν αλλάξουμε τις θέσεις των* $\mathrm{C_i, C_j}$ *ισχύει* $\mathrm{cw_{\sigma'}(G) \leq cw_{\sigma}(G)}$ *αν* $\mathrm{c_j \geq c_i}$ *και βρίσκονται αριστερά της* U *ή αν* $\mathrm{c_j \leq c_i}$ *και βρίσκονται δεξιά της* U

Απόδειξη. Εξετάζουμε την περίπτωση όπου οι κόμβοι βρίσκονται αριστερά της U αφού η δεύτερη περίπτωση είναι εντελώς συμμετρική. Παρατηρούμε ότι το πλάτος αποκοπής αλλάζει μόνο στους κόμβους $\mathrm{C_i, C_j}$ και σε καμία άλλη θέση αρά $\mathrm{cut_{\sigma'}(l) = cut_{\sigma}(l)}$ για $\mathrm{l \neq i, j}$. Αν με θ συμβολίσουμε τις ακμές που περνάνε από την θέση i αλλά δεν προσπίπτουν στις κορυφές των κόμβων $\mathrm{C_i, C_j}$ μπορούμε να υπολογίσουμε το πλάτος αποκοπής μέσα σε ένα κόμβο $\mathrm{cut(C_i) = \theta + \max\{x(c_i \quad x) + xu\}, 0 \leq x \leq c_i}$. Η συνάρτηση αυτή προκύπτει από την ιδιότητα των σχεδόν κατωφλικών γραφημάτων ότι κάθε κόμβος είναι μια κλίκα(και άρα κάθε κορυφή του κόμβου ενώνεται με όλες τις υπόλοιπες), και την ιδιότητα του ενός επιπέδου ότι κάθε κορυφή ενός κόμβου ενώνεται με τις κορυφές της U. Βάση αυτών για την διάταξη σ έχουμε $\mathrm{cut_{\sigma}(C_i) = \ \theta + \max\{x(c_i \quad x) + xu\}, 0 \leq x \leq c_i \quad , \quad cut_{\sigma}(C_j) = \theta + \max\{x(c_j \quad x) + xu + c_i u\}, 0 \leq x \leq c_j}$, ενώ για την διάταξη σ' εχουμε αντίστοιχα $\mathrm{cut_{\sigma'}(C_i) = \ \theta + \max\{x(c_i \quad x) + xu + c_j u\}, 0 \leq x \leq c_i \ , \ cut_{\sigma'}(C_j) = \theta + \max\{x(c_j \quad x) + xu\}, 0 \leq x \leq c_j}$.Για να αποδείξουμε ότι η σ' είναι βέλτιστη πρέπει να δείξουμε ότι $\mathrm{\max_{\sigma}\{cut_{\sigma}(C_i), cut_{\sigma}(C_j)\} \geq \max_{\sigma'}\{cut_{\sigma'}(C_i), cut_{\sigma'}(C_j)\}}$.

Για την περίπτωση όπου $c_i < u$ και αφού $N[C_i] = C_i \cup U$ σημαίνει ότι η μεσαία κορυφή της $N[C_i] \in U$ και άρα από το Λήμμα 3.2 κάθε κορυφή της C_i μπορεί να μετακινηθεί προς την U και άρα να αλλάξει θέσεις με την $C_{(i+1)}$.

Αν $c_i \geq u$ τότε από την υπόθεση $c_j \geq c_i \geq u$, άρα $\max_{\sigma} = cut_{\sigma}(C_j), \max_{\sigma'} = cut_{\sigma'}(C_i)$. Επίσης από την ίδια σχέση ισχύει ότι $cut_{\sigma}(C_j) = \theta + \frac{(c_j+u)^2}{4} + c_i u$ και $cut_{\sigma'}(C_i) = \theta + \frac{(c_i+u)^2}{4} + c_j u$ και επομένως $cut_{\sigma'}(C_i) \quad cut_{\sigma}(C_j) = \frac{1}{4}((c_i \quad u)^2 \quad (c_j \quad u)^2) = \frac{1}{4}\Big((c_i + c_j \quad 2u)(c_i \quad c_j)\Big) \leq 0$, άρα η διάταξη σ' είναι βέλτιστη. □

Με βάση το προηγούμενο Λήμμα αποδείξαμε ότι για κάθε σχεδόν κατωφλικό γράφημα ενός επιπέδου υπάρχει μια βέλτιστη διάταξη των κόμβων του T(G) όπου οι κόμβοι εμφανίζονται σε φθίνουσα σειρά με το μέγεθος τους αριστερά της U και σε αύξουσα σειρά δεξιά της U. Μια τέτοια διάταξη θα την ονομάζουμε *διτονική (bitonic).* Αυτό όμως δεν μας εξασφαλίζει ποιοι κόμβοι θα εμφανιστούν αριστερά και ποιοι δεξιά της U. Άρα ένας απλοϊκός αλγόριθμος θα μπορούσε να δοκιμάζει όλες τις πιθανές διαμερίσεις των κόμβων αυτών φυσικά στην σωστή σειρά. Αυτός ο αλγόριθμος όμως δεν θα ήταν πολυωνυμικού χρόνου αφού όλες οι πιθανές τέτοιες διαμερίσεις είναι $O(2^N)$.

Ας εξετάσουμε ένα τυχαίο στιγμιότυπο του προβλήματος, έστω σ μια διάταξη κόμβων σε διτονική μορφή, $\sigma = (C_1, C_2, .. C_i, ... U, ... C_{N\ 1}), c_1 > c_2 > c_i,$ και C_i ένας τυχαίος κόμβος αριστερά της U. Όπως είδαμε και στην απόδειξη του προηγούμενου λήμματος το πλάτος αποκοπής στον κόμβο αυτό είναι $cut_{\sigma}(C_i) = \theta + \max\{x(c_i \quad x) + xu\}, 0 \leq x \leq c_i$, όπου ϑ είναι οι ακμές που περνάνε από

τις θέσεις του C_i και δεν προσπίπτουν στις κορυφές του. Από τον ορισμό των γραφημάτων αυτών ξέρουμε ότι όλοι οι κόμβοι συνδέονται με την U και μόνο με αυτή, και άρα οι ακμές αυτές είναι $\theta = (c_1 + c_2 + .. + c_{i\ 1})u$. Παρατηρούμε ότι οι ακμές αυτές *εξαρτώνται μόνο από το πλήθος των κορυφών* που εμφανίζονται πριν την C_i και *όχι από το ποιοι κόμβοι.* Αυτό σημαίνει ότι διατάξεις με το ίδιο άθροισμα είναι *ισοδύναμες* για το πρόβλημα του πλάτους αποκοπής σε τέτοια γραφήματα, και τώρα μπορούμε να δώσουμε ένα πολυωνυμικό αλγόριθμο για την επίλυση του.

Ο αλγόριθμος αυτός είναι ένας αλγόριθμος *δυναμικού προγραμματισμού* που δοκιμάζει όλες τις πιθανές διαμερίσεις των κόμβων σε δύο μέρη αριστερά και δεξιά της U, μη λαμβάνοντας όμως υπόψη τις ισοδύναμες περιπτώσεις όπως της περιγράψαμε παραπάνω. Για ευκολία θα συμβολίσουμε το πλάτος αποκοπής ενός κόμβου ανεξαρτήτως διάταξης ως $\mathrm{cut}(C_i) = \max\{x(c_i \quad x) + xu\}, 0 \le x \le c_i$, θυμίζουμε ότι με $n = |V| = \sum_1^{N\ 1} C_i + U$, και επίσης το πλάτος αποκοπής μέσα στον καθολικό κόμβο $\mathrm{cut}(U, l, r) = \max\{x(u \quad x) + xr + (u \quad x)l\}, 0 \le x \le u$, όπου με l συμβολίζουμε τις κορυφές αριστερά του U και αντίστοιχα με r τις κορυφές δεξιά του U. Αρχικά ο αλγόριθμος θεωρεί ως είσοδο τους κόμβους του T(G) σε φθίνουσα σειρά. Σε ένα τυχαίο βήμα του αλγορίθμου εξετάζουμε τον κόμβο C_i , έστω ότι η μέχρι στιγμής διαμέρηση των κόμβων είναι $\sigma = (C_{l1}, C_{l2}, \_, U, \_, C_{r2}, C_{r1})$, ξέρουμε ότι $c_{l1} \ge c_{l2}, c_{r2} \le c_{r1}$, από το Λήμμα 3.4 και ότι οι πιθανές θέσεις του C_i είναι οι σημειωμένες με «_». Άρα μπορούμε να υπολογίσουμε το πλάτος αποκοπής αν ο C_i είναι αριστερά του U, και αντίστοιχα αν είναι δεξιά του U. Το πλάτος αυτό θα είναι το μέγιστο από το πλάτος που έχουμε υπολογίσει στα προηγούμενα βήματα (δυναμικός προγραμματισμός) ή το πλάτος που επιβάλει ο κόμβος C_i. Φυσικά πρέπει να υπολογιστεί και το

πλάτος αποκοπής του κόμβου U σε κάθε βήμα. Με βάση όλα τα παραπάνω μπορούμε να δώσουμε το παρακάτω αναδρομικό τύπο για τον υπολογισμό του πλάτους αποκοπής με βάση τον οποίο γράφουμε και τον αλγόριθμο δυναμικού προγραμματισμού. Ο αλγόριθμος αυτός εκτελείται σε χρόνο $\Omega(nN)$, αφού $N \leq n$ δηλαδή το πολύ $O(n^2)$ και άρα πολυωνυμικός.

$$A(l, i) = \max\{cut(C_i) + r_i u, cut(U, l, r_i + c_i), A(l, i \quad 1)\}$$

$$A(l + c_i, i) = \max\{cut(C_i) + lu, cut(U, l + ci, r_i), A(l, i \quad 1)\}$$

$$r_i = \sum_{0}^{i \quad 1} c_k$$

Αλγόριθμος 3: Πλάτος αποκοπής ενός επιπέδου $(T(G)=(C_1 > .. C_{N \ 1}, U))$

$A[1..n, 1..N] = 0$
for $i = 1$ to $N \quad 1$:
  for $l = 0$ to n:
    if $A[l, i \quad 1] \neq 0$ or $i = 1$:
      if $A[l, i] = 0$:
        $A[l, i] = \max\{cut(C_i) + r_i u, cut(U, l, r_i + c_i), A(l, i \quad 1)\}$
      else:
        $A[l, i] = \min\{A[l, i], \max\{cut(C_i) + r_i u, cut(U, l, r_i + c_i), A(l, i \quad 1)\}\}$
    $A[l + c_i, i] = \max\{cut(C_i) + lu, cut(U, l + ci, r_i), A(l, i \quad 1)\}$
return $\min_{k \in [0..n]} A[k, N \quad 1]$

3.3. Προς την κατεύθυνση πολυωνυμικού αλγόριθμου για QT γραφήματα

Στην παρούσα παράγραφο θα προσπαθήσουμε να επεκτείνουμε τις ιδέες που οδήγησαν στον πολυωνυμικό αλγόριθμο για τα σχεδόν κατωφλικά γραφήματα ενός επιπέδου σε όλα τα σχεδόν κατωφλικά γραφήματα.

3.3.1. Αλγόριθμος για κατωφλικά γραφήματα

Αρχικά αφού τα κατωφλικά γραφήματα είναι υπό-κλάση των σχεδόν κατωφλικών και αφού υπάρχει γραμμικός αλγόριθμος για το πρόβλημα του πλάτους αποκοπής σε αυτά λογικό είναι να δοκιμάσουμε τον αλγόριθμο αυτό και να μελετήσουμε τι συμβαίνει.

Αρχικά αφού τα κατωφλικά γραφήματα είναι υπό-κλάση των σχεδόν κατωφλικών και αφού υπάρχει γραμμικός αλγόριθμος για το πρόβλημα του πλάτους αποκοπής σε αυτά λογικό είναι να δοκιμάσουμε τον αλγόριθμο αυτό και να μελετήσουμε τι συμβαίνει. Ο αλγόριθμος αυτός αρχικά ορίζει ως *τάξη(rank)* μιας κορυφής $v \in V$ ως προς ένα σύνολο $S \subseteq V$, $rank_S(v)$, το βαθμό της κορυφής αυτής ως προς το σύνολο $V(G)\backslash S$ μείον το βαθμό του v ως προς το S, δηλαδή $rank_S(v) = \deg_{V(G)\backslash S}(v) \quad \deg_{\mathrm{S}} v$. Σε ένα τυχαίο βήμα i ο αλγόριθμος έχει ήδη επιλέξει τις κορυφές $\mathrm{V_{i\ 1}}$, άρα εμφανίζονται πρώτες στην γραμμική διάταξη, και επιλέγει την επόμενη κορυφή v $\mathrm{V_{i\ 1}}$ που έχει την μικρότερη τάξη ως προς το σύνολο $V_{i\ 1}$. Αν υπάρχουν παραπάνω από μία κορυφές με την ίδια τάξη τότε διαλέγει αυτή με τον μεγαλύτερο βαθμό στο γράφημα. Θέτει την κορυφή αυτή ως επόμενη στην γραμμική διάταξη, $\sigma(v) = i$. Άρα σε κάθε βήμα διαλέγει την κορυφή που ελαχιστοποιεί τον βαθμό του $\mathrm{V_{i\ 1}} \cup \{u\}$, και σε περίπτωση ισότητας, την κορυφή που μειώνει την τάξη όσο το δυνατόν περισσότερων κορυφών ως προς το $\mathrm{V_{i\ 1}} \cup \{u\}$. [34]

Ο αλγόριθμος χρησιμοποιεί τον *βαθμό* και την *τάξη* μιας κορυφής για να επιλέξει την επόμενη σε κάθε βήμα, τιμές που ορίζονται για οποιοδήποτε γράφημα, επομένως ο αλγόριθμος μπορεί να εκτελεστεί σε όλα τα γραφήματα. Να σημειώσουμε ότι για ένα μη συνεκτικό γράφημα,

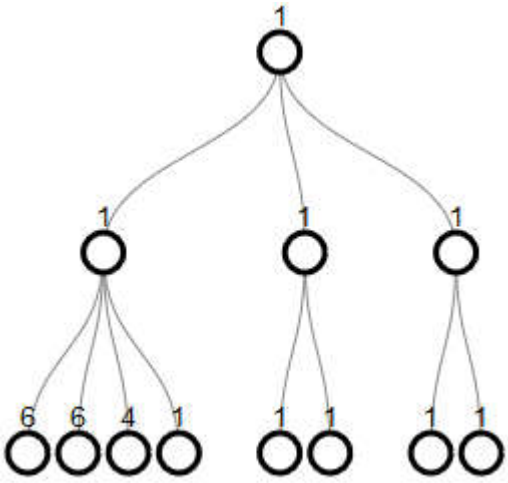
για το πρόβλημα του πλάτους αποκοπής προφανώς πάντα υπάρχει μια βέλτιστη διάταξη που διατηρεί τις συνεκτικές συνιστώσες του γραφήματος χωριστά, και άρα χωρίς βλάβη της γενικότητας μπορούμε να θεωρούμε *μόνο συνεκτικά γραφήματα*.

Από την απλότητα του κριτηρίου επιλογής της επόμενης κορυφής από τον συγκεκριμένο αλγόριθμο μπορούμε να κατασκευάσουμε ένα σχεδόν κατωφλικό γράφημα στο οποίο θα εξαναγκάζουμε τον αλγόριθμο να επιλέξει αυτές τις κορυφές πρώτα, δηλαδή κορυφές με πολύ μικρό βαθμό. Για παράδειγμα μπορούμε να κατασκευάσουμε ένα τέτοιο γράφημα με καθολικό κόμβο μεγέθους 1, με l παιδιά μεγέθους l, κάθε ένα από τα οποία έχει ένα παιδί μεγέθους c και τουλάχιστον άλλα l παιδιά μεγέθους 1 όπως στο παρακάτω σχήμα. Ο αλγόριθμος αυτός λοιπόν θα επέλεγε πρώτα αυτά τα φύλλα μεγέθους 1 (γιατί έχουν το μικρότερο βαθμό) και θα τα τοποθετούσε πρώτα σε μια γραμμική διάταξη με αποτέλεσμα ένα πλάτος αποκοπής τουλάχιστον $\Omega(l^3)$, ενώ σε μια διάταξη όπου χωρίζουμε τα παιδιά αυτά αριστερά-δεξιά του καθολικού κόμβου θα είχαμε ένα πλάτος αποκοπής $\Omega(l^2)$ (χωρίς καν να ξέρουμε αν η διάταξη αυτή είναι η βέλτιστη). Άρα ο άπληστος αλγόριθμος για τα κατωφλικά γραφήματα είναι τουλάχιστον $\Omega(\sqrt{n})$ φορές χειρότερος από έναν βέλτιστο, και αφού ήδη υπάρχει ένας προσεγγιστικός αλγόριθμος για γενικά γραφήματα με καλύτερη προσέγγιση, καταλαβαίνουμε ότι ο άπληστος αυτός αλγόριθμος δεν είναι κατάλληλος για τα σχεδόν κατωφλικά γραφήματα.

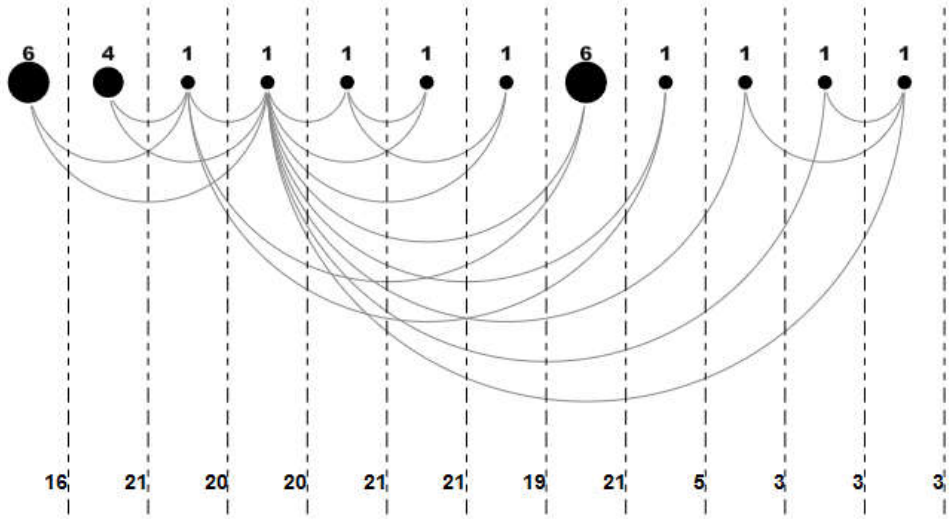
3.3.2. Μορφή βέλτιστης διάταξης

Στην προηγούμενη ενότητα αποδείξαμε ότι οι κορυφές των κόμβων που αποτελούν την δεντρική αναπαράσταση ενός σχεδόν κατωφλικού γραφήματος εμφανίζονται σε συνεχόμενες θέσεις(Λήμμα 3.3). Άρα

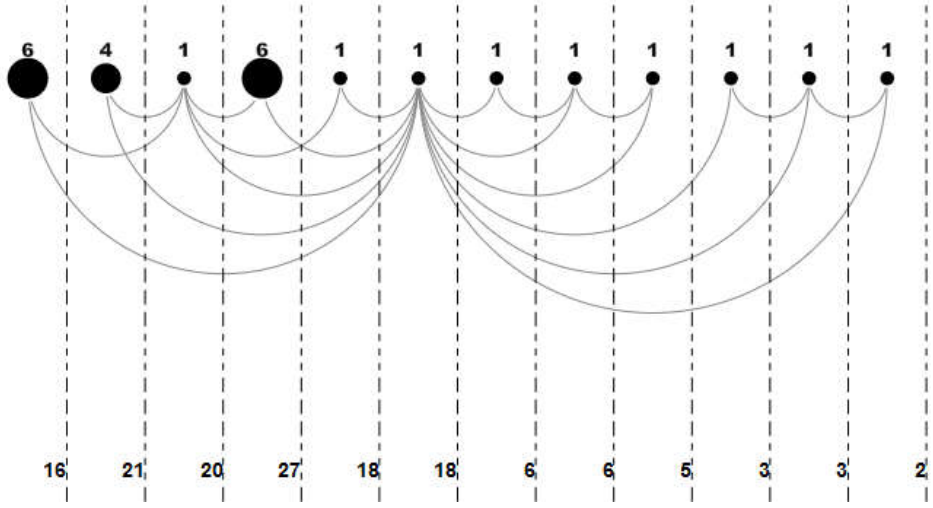
ξέρουμε ήδη ότι μπορούμε να εξετάζουμε διατάξεις των κόμβων αυτών και όχι όλων των κορυφών του γραφήματος. Το επόμενο λογικό βήμα είναι να θεωρήσουμε ότι πιθανώς μια βέλτιστη διάταξη θα εμφανίζει τα υπό-δέντρα που αποτελούν το γράφημα σε συνεχόμενες θέσεις, και άρα τα υπό-δέντρα αυτά θα εμφανίζονται χωριστά μεταξύ τους. Αναλυτικά αν θεωρήσουμε ότι U είναι ο καθολικός κόμβος και A, B είναι δύο υπό-δέντρα παιδιά του καθολικού κόμβου αυτού, και έστω μια βέλτιστη διάταξη των κόμβων αυτών $\sigma = (A_L, B_L, U, A_R, B_R)$, όπου $A_L, A_R \subseteq A$ και B_L $B_R \subseteq B$ είναι τα υποσύνολα των A, B που είναι αριστερά-δεξιά του καθολικού κόμβου. Τότε θα θέλαμε να αποδείξουμε ότι υπάρχει μια διάταξη σ' η οποία δεν «μεγαλώνει» το πλάτος αποκοπής και τα υποσύνολα A_L, A_R όπως και τα B_L, B_R εμφανίζονται σε συνεχόμενες θέσεις. Στην προσπάθεια απόδειξης της ιδιότητας αυτής όμως καταλήγουμε στο αντιπαράδειγμα που φαίνεται στην παρακάτω εικόνα.



Εικόνα 20 Αντιπαράδειγμα όπου τα υπό-δέντρα δεν μπορούν να βρεθούν σε συνεχόμενες θέσεις



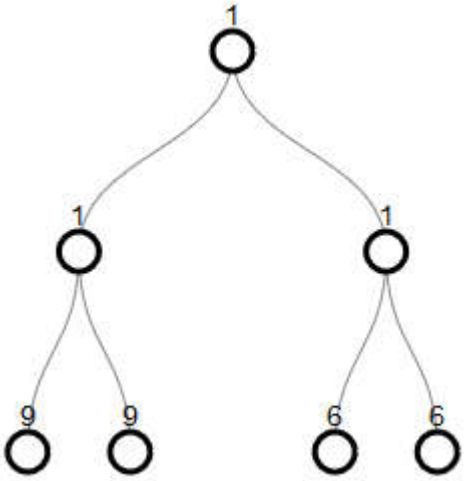
Εικόνα 21 Μια βέλτιστη διάταξη με πλάτος αποκοπής 21, τα υπό-δέντρα δεν είναι συνεχόμενα



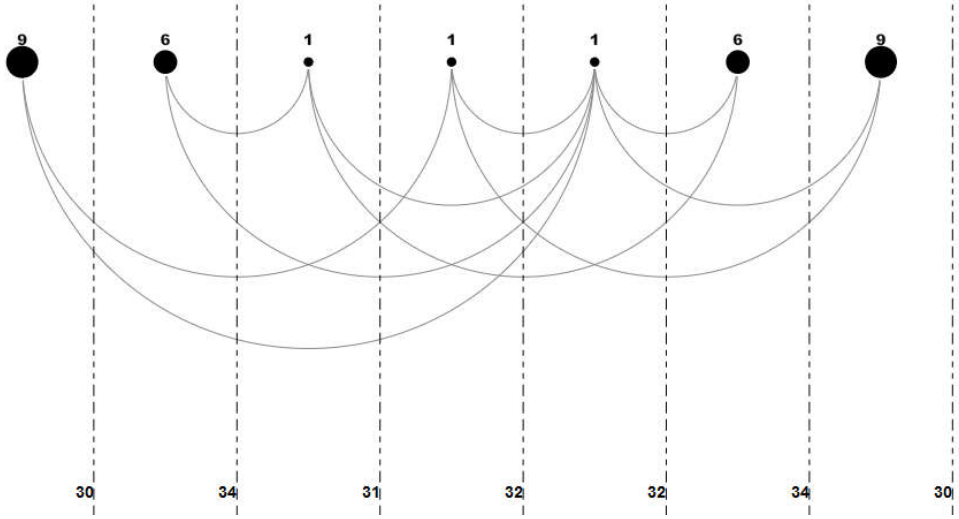
Εικόνα 22 Μια διατάξη με τα υπό-δέντρα συνεχόμενα, με πλάτος αποκοπής 27

Στο αντιπαράδειγμα αυτό δεν υπάρχει βέλτιστη διάταξη στην οποία το υπο-δέντρο (6,6,4,1) να εμφανίζεται σε συνεχόμενες θέσεις. Παρατηρούμε ότι η διάταξη που θα «θέλαμε» έχει πλάτος αποκοπής 27 ενώ υπάρχει βέλτιστη διάταξη με πλάτος αποκοπής 21. Μάλιστα το πλάτος αποκοπής 27 είναι το καλύτερο που έχουν όλες οι διατάξεις με τα υπό-δέντρα σε συνεχόμενες θέσεις. Αν μελετήσουμε το αντιπαράδειγμα αυτό θα δούμε ότι έχει κόμβους φύλλα που είναι *αυστηρά μεγαλύτερα* από την κλειστή γειτονιά τους, δηλαδή $|V_i| \geq N[V_i]$ (οι κόμβοι με μέγεθος $6 \geq N[6] = 2$). Επομένως η μεσαία κορυφή της κλειστής γειτονιάς τους θα είναι πάντα μέσα στους κόμβους αυτούς και άρα το Λήμμα 3.2 δεν είναι εφαρμόσιμο ώστε οι κόμβοι αυτοί να μπορούν να μετακινηθούν προς το υπόλοιπο υπό-δέντρο στο οποίο ανήκουν. Ταυτόχρονα όμως οι κόμβοι γονείς των φύλλων αυτών έχουν μοιρασμένοι την κλειστή γειτονιά τους αριστερά δεξιά(η μεσαία κορυφή της κλειστής γειτονιάς τους είναι μέσα στους ίδιους) και επομένως ούτε σε αυτούς είναι εφαρμόσιμο το Λήμμα 3.2 .

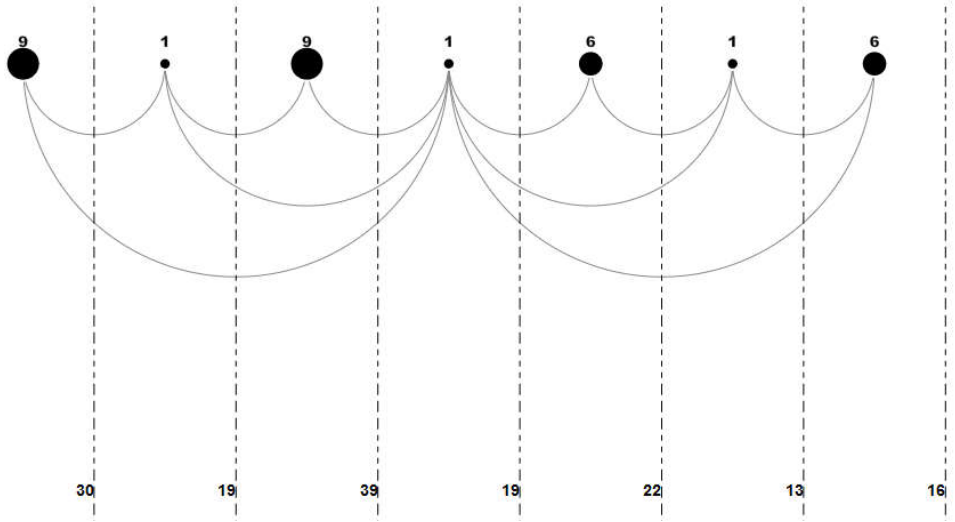
Επομένως ξέρουμε ότι τα υπο-δέντρα είναι πιθανόν να μην εμφανίζονται σε συνεχόμενες θέσεις. Λογικό όμως είναι να υποθέσουμε ότι το υποσύνολο των υπο-δέντρων που βρίσκονται αριστερά του καθολικού κόμβου εμφανίζεται σε συνεχόμενες θέσεις, και αντίστοιχα το δεξιό. Όμως και για αυτή την περίπτωση μπορούμε να βρούμε ένα κατάλληλο αντιπαράδειγμα όπως βλέπουμε στη παρακάτω εικόνα. Και πάλι βλέπουμε κόμβους φύλλα με μέγεθος μεγαλύτερο της κλειστής γειτονιάς τους, και κόμβους γονείς με μοιρασμένη την γειτονιά τους.



Εικόνα 23 Αντιπαράδειγμα όπου τα υπό-δέντρα δεν βρίσκονται σε διαδοχικές θέσεις αριστερά και δεξιά του καθολικού κόμβου



Εικόνα 24 Μια βέλτιστη διάταξη με πλάτος αποκοπής 34, τα υποσύνολα που βρίσκονται στην ίδια πλευρά του καθολικού κόμβου μπορεί να μη εμφανίζονται σε συνεχόμενες θέσεις



Εικόνα 25 Η διάταξη που θα θέλαμε με πλάτος αποκοπής 39

Ο αλγόριθμος ενός επιπέδου γνώριζε ότι μπορούσε να διατάξει τους κόμβους σε φθίνουσα σειρά ως προς το μέγεθος τους και αυτή θα ήταν η σωστή σειρά που θα εμφανιζόταν σε μια βέλτιστη διάταξη. Προφανής επέκταση αυτού του σκεπτικού σε γραφήματα πολλών επιπέδων θα ήταν να βρούμε μια αντίστοιχη διάταξη των κόμβων. Όμως η προσπάθεια αυτή «σκοντάφτει» στο γεγονός ότι κάθε κόμβος έχει διαφορετική γειτονιά, η οποία σε μια βέλτιστη διάταξη μπορεί να μοιρασμένη αριστερά-δεξιά και η διαφορά αυτή καθορίζει πότε ένας κόμβος μπορεί να αλλάξει θέση με έναν διπλανό του. Στα γραφήματα του ενός επιπέδου αυτό δεν ίσχυε αφού όλοι οι κόμβοι είχαν ακριβώς την ίδια γειτονιά, τον καθολικό κόμβο και μόνον αυτό. Άρα δεν μπορούμε να γνωρίζουμε εκ των προτέρων με πια σειρά θα εμφανιστούν οι κόμβοι σε μια βέλτιστη διάταξη. Για παράδειγμα έστω οι κόμβοι A_1, A_2, A_3, A_4 που είναι παιδιά ενός κόμβου U_1 ,έστω U ο καθολικός κόμβος και έστω μια βέλτιστη διάταξη $\sigma_1 = (A_1, A_2, U_1, A_3, A_4, U, \dots)$. Εύκολά μπορούμε να δείξουμε ότι οι κόμβοι

A_1, A_2 πρέπει να είναι σε φθίνουσα σειρά (ακριβώς όπως στο ένα επίπεδο), αλλά για τους κόμβους A_3, A_4 η σειρά τους εξαρτάται από την διαφορά $|U| \quad |U_1|$. Αν η διαφορά αυτή είναι θετική τότε πρέπει να είναι σε φθίνουσα σειρά αλλιώς σε αύξουσα. Όταν επεκτείνουμε την ιδέα αυτή από κόμβους σε υπό-δέντρα τότε οι συνθήκες βάση των οποίων θα πρέπει να τις κατατάξουμε γίνονται ακόμα πιο πολύπλοκες, και χωρίζονται σε ακόμα περισσότερες περιπτώσεις. Άρα φαίνεται ότι δεν μπορούμε να γνωρίζουμε μια εκ των προτέρων σειρά για τους κόμβους βάση της οποίας θα μπορούσαμε να χαρακτηρίσουμε μια βέλτιστη διάταξη και να σχεδιάσουμε έναν αλγόριθμο.

Παρατηρούμε λοιπόν ότι μια βέλτιστη διάταξη δεν έχει κάποιες προφανείς ιδιότητες ως προς την μορφή της. Επομένως μια λογική επέκταση του πολυωνυμικού αλγορίθμου του ενός επιπέδου σε πολλά επίπεδα δεν θα έχει ως αποτέλεσμα μια βέλτιστη λύση. Εξάλλου μια επέκταση του αλγορίθμου αυτού σε πολλά επίπεδα θα έδινε μια διάταξη με πλάτος αποκοπής που δεν συγκρίνεται με το βέλτιστο και επομένως δεν μπορούμε να μιλάμε ούτε για έναν προσεγγιστικό αλγόριθμο(αφού προϋποθέτει ότι γνωρίζουμε το *πόσες φόρες χειρότερες* λύση μπορεί να δώσει).

3.3.3. Δυναμικός αλγόριθμος διάταξης κόμβων

Βασιζόμενοι σε όλα τα παραπάνω επιχειρήματα η μόνη ιδιότητα που γνωρίζουμε για μια βέλτιστη διάταξη είναι αυτή του Λήμματος 3.3, ότι οι κορυφές των κόμβων εμφανίζονται σε διαδοχικές θέσεις και άρα μπορούμε να μιλάμε για *διατάξεις κόμβων* και *όχι κορυφών*. Άρα ήδη έχουμε μειώσει το πρόβλημα από το μέγεθος $n = |V|$ στον αριθμό των κόμβων της δεντρικής αναπαράστασης $N = |V_i| \in T(G) \leq |V|$. Αφού

μιλάμε για πρόβλημα διάταξης μπορούμε να δώσουμε έναν αλγόριθμο βασιζόμενοι στον δυναμικό αλγόριθμο που παρουσιάσαμε στο πρώτο κεφάλαιο των Bodlaender, Fomin, Koster, Kratsch και Thilikos (κεφάλαιο 1.7).

Ο αλγόριθμος αυτός σε κάθε βήμα εξέταζε τις κορυφές $w \in V$ του γραφήματος, θα πρέπει να τον τροποποιήσουμε κατάλληλα ώστε να εξετάζει τους κόμβους ενός σχεδόν κατωφλικού γραφήματος $\mathrm{V_i} \in T(G)$. Πρώτη απαίτηση του αλγορίθμου είναι η εύρεση της συνάρτησης f που υπολογίζει το τρέχον «κόστος» του προβλήματος(σε πολυωνυμικό χρόνο). Η συνάρτηση αυτή εξαρτάται από το γράφημα $G = (V, E)$, ένα σύνολο $S \subseteq V$, και μια κορυφή $u \in V \quad S$. Για την περίπτωση μας θα πρέπει να τροποποιήσουμε την συνάρτηση αυτή ώστε να δέχεται κόμβους αντί για κορυφές, άρα θα εξαρτάται από την δεντρική αναπαράσταση ενός σχεδόν κατωφλικού γραφήματος $T(G) = (V_1, V_2, ... V_N)$, ένα σύνολο $\mathrm{S} \subseteq \mathrm{T(G)}$, και έναν κόμβο $\mathrm{V_i} \in T(G) \quad S$.

Η συνάρτηση αυτή εκτελείται σε κάθε βήμα του αλγορίθμου και υπολογίζει την τιμή του προβλήματος μέσα στον κόμβο $\mathrm{V_i}$, σε μια διάταξη που οι κόμβοι του S εμφανίζονται πρώτα με άγνωστη σειρά, μετά ο κόμβος V_i και τέλος οι υπόλοιποι κόμβοι $T(G) \quad S \quad \{V_i\}$ επίσης με άγνωστη σειρά. Από τις ιδιότητες των σχεδόν κατωφλικών γραφημάτων γνωρίζουμε ότι κάθε κόμβος V_i επάγει μια κλίκα στο γράφημα, άρα κάθε κορυφή του κόμβου ενώνεται με όλες τις υπόλοιπες σε αυτόν και επομένως το πλάτος αποκοπής μέσα στον κόμβο και δεδομένων των παραπάνω συμβολισμών δίνεται από τον τύπο:

$$\boldsymbol{f(T(G), S, V_i)} = |W||X| : W \in S, X \in T(G) \quad S \quad \{V_i\} + \max\{ \quad i^2 + i(|V_i| + |R| \quad |L|) + |V_i||L|\}$$

Όπου $L = N(V_i) \cap S$ και $R = N(V_i) \cap T(G) \quad S \quad \{V_i\}$, δηλώνουν το μέρος της γειτονιάς του $\mathrm{V_i}$ βρίσκεται αριστερά του $\mathrm{V_i}$(επομένως ανήκει στο σύνολο S) και αντίστοιχα δεξιά του V_i (επομένως ανήκει στο $T(G) \quad S$).

Με βάση αυτή την συνάρτηση μπορούμε προφανώς να τροποποιήσουμε τον αλγόριθμο ώστε να αναζητά διατάξεις κόμβων, και όχι κορυφών, αντικαθιστώντας κατάλληλα τις κορυφές με κόμβους. Επομένως έχουμε έναν *εκθετικό αλγόριθμο για το πρόβλημα του πλάτους αποκοπής σε σχεδόν κατωφλικά γραφήματα που εκτελείται σε χρόνο* $O(2^N)$.

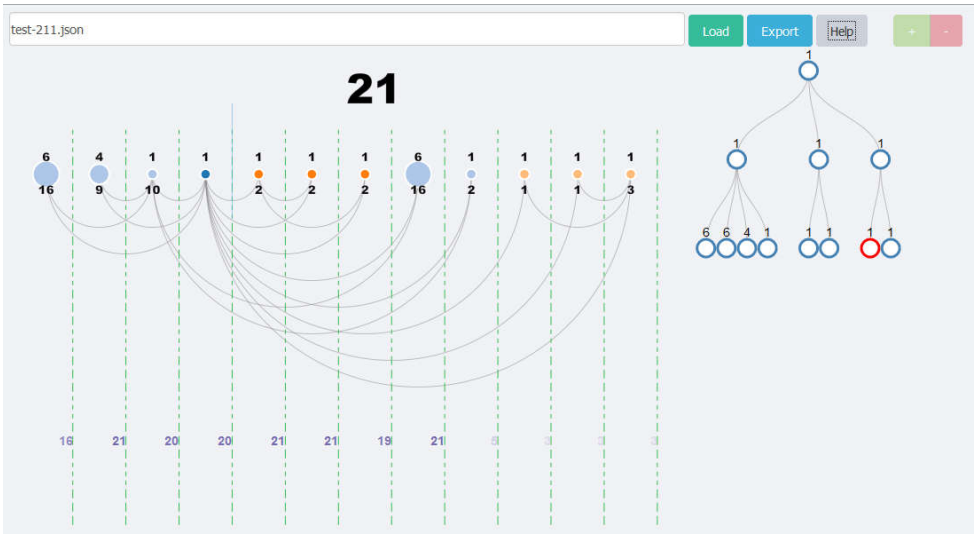
Αλγόριθμος 3: Αλγόριθμος δυναμικού προγραμματισμού, QT γράφημα $T(\mathrm{G}) = (\mathrm{V}_1, ... \mathrm{V}_\mathrm{N})$

Set $A(\ ) = \quad \infty$
for $i = 1$ *to* N:
for *all vertex sets* $S \subset T(G) : |S| = i :$
Set $A(S) = \min_{w \in S} \max\{f(G, S \quad \{w\}, w), A(S \quad \{w\})\}$
end for
end for
return $A(T(G))$

3.3.4. Υλοποιήσεις

Στα πλαίσια της έρευνας για την παρούσα εργασία χρειάστηκε να υλοποιήσουμε αρκετά χρήσιμα εργαλεία για να μας βοηθήσουν να εξετάσουμε το πρόβλημα πιο γρήγορα. Επιστέγασμα των προσπαθειών

αυτών είναι μια υλοποίηση ενός προγράμματος γραφικής αναπαράστασης γραφημάτων. Το πρόγραμμα αυτό έχει την δυνατότητα να εμφανίζει την δεντρική αναπαράσταση ενός σχεδόν κατωφλικού γραφήματος ενώ ταυτόχρονα εμφανίζει μια γραμμική διάταξη του όπως φαίνεται στην παρακάτω εικόνα.



Εικόνα 26 Γραφικό πρόγραμμα μελέτης σχεδόν κατωφλικών γραφημάτων, και των γραμμικών του διατάξεων

Για την τρέχουσα γραμμική διάταξη που εμφανίζεται στην οθόνη το πρόγραμμα υπολογίζει το πλάτος αποκοπής σε κάθε κόμβο εμφανίζει την τιμή του, ενώ το μέγιστο πλάτος αποκοπής εμφανίζεται σε κεντρικό σημείο. Ο χρήστης έχει την δυνατότητα να μετακινεί κόμβους με το ποντίκι στην διάταξη αυτή και ταυτόχρονα επανυπολογίζεται το πλάτος αποκοπής σε όλους τους κόμβους της διάταξης. Ο χρήστης έχει επίσης την δυνατότητα να κατασκευάζει ή να τροποποιεί το γράφημα χρησιμοποιώντας της δεντρική αναπαράσταση στα δεξιά. Πατώντας με το ποντίκι στην δεντρική αναπαράσταση ο χρήστης έχει την δυνατότητα να

προσθέτει και να αφαιρεί κόμβους παιδιά από το γράφημα, αλλά και να αυξομειώνει το μέγεθος του κάθε κόμβου.

Το πρόγραμμα έχει την δυνατότητα να διαβάζει γραφήματα αποθηκευμένα σε αρχεία ειδικής μορφής(json). Στα αρχεία αυτά είναι αποθηκευμένη η δεντρική αναπαράσταση ενός γραφήματος, καθώς και οι ακμές του γραφήματος. Ο χρήστης έχει την δυνατότητα εξαγωγής ενός τέτοιου αρχείου για ένα γράφημα που έχει κατασκευάσει χρησιμοποιώντας το πρόγραμμα αυτό. Επίσης στο αρχείο αυτό υπάρχει η δυνατότητα αποθήκευσης συγκεκριμένων διατάξεων ώστε ο χρήστης να μπορεί να τις δει με ευκολία και να μετακινηθεί από την μια στην άλλη.

Τέλος στο πρόγραμμα αυτό έχουν υλοποιηθεί οι αλγόριθμοι που παρουσιάστηκαν στην παρούσα εργασία, ο πολυωνυμικός αλγόριθμος ενός επιπέδου(που μπορεί να εκτελεστεί και για πολλά επίπεδα), ο άπληστος αλγόριθμος για τα κατωφλικά γραφήματα, και ο εκθετικός αλγόριθμος για διατάξεις κόμβων (που βρίσκει και την βέλτιστη διάταξη σε σχεδόν κατωφλικά γραφήματα). Μετά την εκτέλεση του κάθε αλγορίθμου ο χρήστης βλέπει άμεσα την βέλτιστη διάταξη που υπολογίστηκε καθώς και το πλάτος αποκοπής για αυτή. Το πρόγραμμα παρέχει και βοήθεια με όλες τις συντομεύσεις του πληκτρολογίου και του ποντικιού για όλες τις παραπάνω λειτουργίες.

Το πρόγραμμα αυτό υλοποιήθηκε στην γλώσσα *Javascript*, σε μορφή δηλαδή σελίδας του διαδικτύου. Η γλώσσα αυτή επιλέχθηκε καταρχάς γιατί προσφέρει ένα εύκολο περιβάλλον εργασίας όπου μπορεί να εκτελεστεί σχεδόν σε κάθε υπολογιστή και σε κάθε λειτουργικό σύστημα, αφού εκτελείται μέσα στο περιβάλλον του browser για το διαδίκτυο. Άρα μπορεί να μεταφερθεί και να εκτελεστεί άμεσα από τον

οποιοδήποτε, ενώ θα μπορούσε να εκτελείται και μέσω του διαδικτύου σε μορφή σελίδας σε κάποιο server. Επίσης η γλώσσα *Javascript* προσφέρει ευκολία στην διαδραστικότητα με το χρήστη αφού ο λόγος κατασκευής της γλώσσας αυτής εξ' αρχής ήταν για να επιτρέψει την αλληλεπίδραση του χρήστη με μια σελίδα του διαδικτύου. Η αλληλεπίδραση του χρήστη ήταν βασική προδιαγραφή για να είναι χρήσιμο ένα τέτοιο εργαλείο. Τέλος η γλώσσα προσφέρει αρκετά πακέτα για την εμφάνιση γραφικών, γραφημάτων και διαγραμμάτων όπως το d3.js που χρησιμοποιήθηκε στο εργαλείο αυτό. Ο κώδικας του προγράμματος υπάρχει δημοσιευμένος στην ιστοσελίδα:

https://github.com/Antreasgr/CutwidthVisualization [41]

Επιπρόσθετα για την επαλήθευση των αποτελεσμάτων που παρουσιάστηκαν στην παρούσα χρειάστηκε η συγγραφή ενός προγράμματος εξαντλητικής αναζήτησης της βέλτιστης διάταξης ή διατάξεων με πολύ συγκεκριμένα χαρακτηριστικά. Για την βελτιστοποίηση του προγράμματος αυτού επιστρατεύθηκε η παράλληλη(και πολυνηματική) επεξεργασία, αφού όπως είναι προφανές η εξαντλητική αναζήτηση απαιτεί αρκετή ώρα ακόμα και σε μικρά σχετικά γραφήματα. Το πρόγραμμα αυτό είχε την δυνατότητα να εξάγει τα αποτελέσματα (τις διατάξεις που εντοπίζει) στην μορφή του πρώτου προγράμματος ώστε ο χρήστης να έχει μια «οπτική» ιδέα στα αποτελέσματα. Το πρόγραμμα αυτό υλοποιήθηκε στην γλώσσα *C#*.

Τέλος στην γλώσσα *Python* υλοποιήθηκε ένα πρόγραμμα που εκτελούσε αναζήτηση μια διάταξης με βάση τον αλγόριθμο Simulated Annealing(όπως στην εργασία [12]) και φυσικά δεν εξασφαλίζει ότι

επιστρέφει μια βέλτιστη διάταξη αλλά εκτελείται σε κλάσμα του χρόνου που απαιτούσε η εξαντλητική αναζήτηση.

ΚΕΦΑΛΑΙΟ 4

4. Σύνοψη – Επεκτάσεις

4.1. Συμπεράσματα

Η παρούσα εργασία μελετά το πρόβλημα του πλάτους αποκοπής στην κλάση των σχεδόν κατωφλικών γραφημάτων. Το πρόβλημα αυτό είναι ανοιχτό μέχρι στιγμής για την συγκεκριμένη κλάση γραφημάτων ερώτημα στο οποίο προσπαθήσαμε να δώσουμε απάντηση. Αν και η συγκεκριμένη κλάση γραφημάτων είναι υπερ-κλάση των κατωφλικών γραφημάτων για τα οποία είναι γνωστός γραμμικός αλγόριθμος για το συγκεκριμένο πρόβλημα, φαίνεται ότι το πρόβλημα είναι NP-Hard για αυτή. Για την υπερ-κλάση των interval γραφημάτων είναι επίσης άγνωστο σε ποια κλάση ανήκει το πρόβλημα.

Παρουσιάσαμε συγγενή προβλήματα διατάξεων με το πρόβλημα του πλάτους αποκοπής και γνωστά αποτελέσματα σε αρκετές κλάσεις γραφημάτων. Παρουσιάσαμε τους πιο γρήγορους μέχρι στιγμής αλγορίθμους σε γενικά γραφήματα που επιλύουν πολλά από τα προβλήματα διατάξεων σε χρόνο και χώρο $O(2^n)$. Αποδείξαμε την μορφή

που έχει μια βέλτιστη διάταξη για σχεδόν κατωφλικά γραφήματα ενός επίπεδου και δώσαμε έναν πολυωνυμικό αλγόριθμο δυναμικού προγραμματισμού που εκτελείται σε χρόνο $O(nN)$. Παρουσιάσαμε την μορφή που έχει μια βέλτιστη διάταξη για σχεδόν κατωφλικά γραφήματα και πως η δεντρική του αναπαράσταση είναι αρκετή για να λυθεί το πρόβλημα του πλάτους αποκοπής. Επίσης παρουσιάσαμε αντιπαραδείγματα για πολλές ιδιότητες που θα περιμέναμε να έχει μια βέλτιστη διάταξη ώστε να μπορούσαμε δώσουμε έναν πολυωνυμικό αλγόριθμο. Παρουσιάζουμε ένα εκθετικό αλγόριθμο σε χρόνο και χώρο $O(2^N)$.

Επίσης παρουσιάζουμε το λογισμικό που αναπτύχθηκε κατά την διάρκεια εκπόνησης της εργασίας για την διευκόλυνση στην έρευνα.

4.2. Ανοικτά ερωτήματα

Η παρούσα εργασία δεν μπόρεσε να δώσει καθοριστική απάντηση για το ανοιχτό πρόβλημα του πλάτους αποκοπής σε σχεδόν κατωφλικά γραφήματα. Προφανώς παραμένει ακόμα ανοιχτό το ερώτημα αν μπορεί να λυθεί σε πολυωνυμικό χρόνο ή αν μπορεί τελικά να βρεθεί αναγωγή σε κάποιο άλλο πρόβλημα της κλάσης NP-Hard.

- Ο αλγόριθμος δυναμικού προγραμματισμού που επιλύει το πρόβλημα σε χρόνο $O(2^N)$, δεν λαμβάνει υπόψιν καθόλου την μορφή των σχεδόν κατωφλικών γραφημάτων. Θα μπορούσε πιθανόν εύκολα να τροποποιηθεί κατάλληλα ώστε να λαμβάνει υπόψιν τον καθολικό κόμβο και τις ιδιότητες του καθώς και την προφανή συμμετρία που έχει μια γραμμική διάταξη αριστερά-

δεξιά του καθολικού κόμβου ώστε να δοθεί αλγόριθμος σε χρόνο $O(2^{\frac{(N-1)}{2}})$.

- Η υπό-κλάση των κατωφλικών γραφημάτων έχει δεντρικές αναπαραστάσεις που μοιάζουν με αυτές των σχεδόν κατωφλικών αν πάρουμε ένα μονοπάτι από την ρίζα σε ένα φύλλο και μειώσουμε κάθε άλλο κόμβο του γραφήματος σε κορυφή. Επομένως αξίζει μελέτης το αν θα μπορούσε ο άπληστος αλγόριθμος για τα κατωφλικά γραφήματα να τροποποιηθεί κατάλληλα ώστε να ομαδοποιεί κορυφές σε κόμβους και τελικά να δουλεύει με τους κόμβους αυτούς ώστε να πλησιάσει στη βέλτιστη διάταξη.
- Τα interval γραφήματα είναι υπερ-κλάση των σχεδόν-κατωφλικών γραφημάτων και έχουν μια δομή μονοπατιού στην οποία κάθε μονοπάτι P_3 αντιστοιχεί σε ένα σχεδόν κατωφλικό γράφημα. Αν υπήρχε πολυωνυμικός αλγόριθμος για το πρόβλημα του πλάτους αποκοπής σε σχεδόν κατωφλικά γραφήματα θα μπορούσε να μελετηθεί κατά πόσο αυτός μπορεί να επεκταθεί σε interval γραφήματα κρατώντας για παράδειγμα την δομή της βέλτιστης διατάξης για κάθε σχεδόν κατωφλικό υπό-γράφημα.
- Για τη σχεδίαση ενός πολυωνυμικού αλγορίθμου σε σχεδόν κατωφλικά γραφήματα θα πρέπει κανείς να εξετάσει και την περίπτωση που κάθε κόμβος-κλίκα του δέντρου αναπαράστασης αποτελείται από έναν μόνο κόμβο. Σε αυτή την περίπτωση έχουμε n= N και ο αλγόριθμος που δώσαμε καταλήγει να έχει την ίδια πολυπλοκότητα με τον εκθετικό (δυναμικό) αλγόριθμο. Αξίζει να σημειώσουμε ότι σε αυτή τη περίπτωση μπορεί κανείς

να εκμεταλλευτεί τον αλγόριθμο για την ελάχιστη αποκοπή σε απλά δέντρα με συγκεκριμένες και συστηματικές τροποποιήσεις λαμβάνοντας υπόψη τις ακμές που προσπίπτουν σε κάθε κόμβο του δέντρου και κάθε απόγονό του. Ωστόσο ο αλγόριθμος για τα απλά δέντρα είναι αρκετά τεχνικός και δεν βασίζεται σε κάποιο δομικό χαρακτηρισμό μιας βέλτιστης διάταξης

– Τέλος ανοιχτό παραμένει το ερώτημα για παρόμοια προβλήματα διατάξεων στα σχεδόν κατωφλικά γραφήματα όπως το πρόβλημα της ελάχιστης διάταξης(Minimum Linear Arrangement), το τροποποιημένο πλάτος αποκοπής(Modified Cutwidth), η απόσταση κορυφών(Vertex Seperation) και το άθροισμα αποκοπής(Sum Cut).

Βιβλιογραφία

[1] D. Adolphson and T.C. Hu, "Optimal linear ordering," *SIAM J. Appl.Math.*, pp. 403-423, 1973.

[2] G. Ausiello et al., "Complexity and Approximation:Combinatorial Optimization Problems and Their Approximability Properties," *Springer*, 1999.

[3] G. Blin, G. Fertin, D. Hermelin, and S.Viallete, "Fixed parameter algorithms for protein similarity search under mRNA structure constrains," *Discrete Algorithms*, pp. 618-626, 2008.

[4] H. Boadlaender, P. Heggernes, and D. Lokshtanov, "Graph modification problems(Dagstuhl Seminar 24071)," , 2014.

[5] Hans L. Bodlaender, Fedor V. Fomin, Arie M.C.A. Koster, Dieter Kratsch, and Dimitrios M. Thilikos, "A Note on Exact Algorithms for Vertex Ordering Problems on Graphs," 2011.

[6] R.A. Botafogo, Cluster analysis for hypertext systems.

[7] A. Brandstadt, V.B. Le, and J. Spinrad, "Graph Classes: a survey No. 3," Siam, 1999.

[8] Jianer Chen, Iyad A. Kanj, and Ge Xia, *Improved Parameterized Upper Bounds for Vertex Cover.*, 2006.

[9] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, *Introduction to Algorithms.*: The MIT Press, 1997.

[10] J. Diaz, A. Gibbons, M.S. Paterson, and J. Toran, "The Minsumcut problem," *Algorithms and Data Structures, F. Dehen, R. J. Sack, and N. Santoro, Eds. Lecture Notes in Computer Science, vol. 519*, pp. 65–79, 1991.

[11] J. Diaz, J. Petit, and M. Serna, "A Survey of Graph Layout Problems," 2002.

[12] R. Diekmann, R. Luling, B. Monien, and C. Spraner, "Combining helpful sets and parallel simulated annealing for the graph-partitioning problem," *Parallel Algorithms and Applications*, pp. 61-84, 1996.

[13] Rod G. Downey and Michael R. Fellows, *Parameterized Complexity.*: Springer, 1999.

[14] M. R. GAREY, D. S. JOHNSON, and L. STOCKMEYER, "Some simplified NP-complete graph," *Theoretical Computer Science 1*, pp. 237-267, 1976.

[15] F. Gavril, "Some NP-complete problems on graphs," , Baltimore, 1977.

[16] P. A. Golovach, "The total vertex separation number of a graph," *Diskretnaya Matematika*, pp. 86–91, 1997.

[17] M.C. Golumbic, "Trivially perfect graphs," *Discrete Mathematics*, pp. 105-107, 1978.

[18] P. Heggernes, P. Vant Hof, D. Lokshtanov, and J. Nederlof, "Computing the cutwidth of bipartite permutation graphs in linear time," *Graph theoretic Concepts in Computer Science*, pp. 75-87, 2010.

[19] P. Heggernes, D. Lokshtanov, D. Mihai, and C. Papadopoulos, "Cutwidth of split graphs, threshold graphs and proper interval graphs.," *In Graph-theoretic concepts in Computer Science*, vol. Springer, pp. 218-229, 2008.

[20] Clay Mathematics Institute. (2015) Clay Mathematics Institute millennium prize. [Online]. http://www.claymath.org/millenium

[21] M. Junguer, G. Reinelt, and G. Rinaldi, "The traveling salesman problem," *Handbook on Operations Research and Management Sciences*, vol. 7, pp. 225-330, 1995.

[22] D. R. Karger, "A randomized fully polynomial approximation scheme for the all-terminal network reliability problem," *Journal on Computing*, pp. 492-514, 1999.

[23] F.T. Leighton and S. Rao, "Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms," *Journal of the ACM*, pp. 787-832, 1999.

[24] T. Lengauer, "Black-white pebbles and graph separation," *Acta Informatica 16*, pp. 465–475, 1981.

[25] Simen Lilleeng and Pinar Heggernes, "A polynomial-time solvable case for the NP-hard problem Cutwidth," 2014.

[26] Simen Lilleen and Prof. Pinar Heggernes, "A polynomial-time solvable case for the NP-Hard problem Cutwidth," Bergen, 2014.

[27] Y. X. Lin and J. J. Yuan, "Profile minimization problem for matrices and graphs," *Acta Mathematicae Applicatae Sinica. English Series*, pp. 107–112, 1994b.

[28] F. Makedon and I.H. Sudborough, "On minimizing width in linear layouts," *Discrete Applied Mathematics*, pp. 243-265, 1989.

[29] B. Monien and I.H. Sudborough, "Min Cut is NP-complete for edge weighted trees," *Theoretical Computer Science*, pp. 209–229, 1988.

[30] P. Mutzel, "A polyhedral approach to planar augmentation and related problems," , ESA, 1995.

[31] S.D. Nikolopoulos, "Parallel algorithms for Hamiltonian problems on quasi-threshold graphs.," *Journal of Parallel and Distributed Computing*, pp. 48–67, 2004.

[32] C. Papadimitriou, "The NP-Completeness of the bandwidth minimization problem," *Computing*, pp. 263-270, 1976.

[33] C. Papadopoulos, *Θεωρία Γραφημάτων*. Ιωάννινα: Πανεπιστήμιο Ιωαννίνων Τμήμα Μαθηματικών, 2012.

[34] Charis Papadopoulos, Pinar Heggernes, Daniel Lokshtanov, and Rodica Mihai, "Cutwidth of Split Graphs, Threshold Graphs, and Proper Interval Graphs," 2008.

[35] D.M. Thilikos, M.J. Serna, and H.L. Bodlaender, "Cutwidth I: A linear time fixed parameter algorithm," *Journal of Algorithms*, pp. 1-14, 2005.

[36] D.M. Thilikos, M.J. Serna, and H.L. Bodlaender, "Cutwidth II: Algorithms for partial w-trees of bounded degree," *Journal of Algorithms*, pp. 24-49, 2005.

[37] E.S. Wolk, "The comparability graph of a tree," *Proc. Amer. Math Soc*, pp. 789-795, 1962.

[38] J-H Yan, J-J Chen, and G.J. Chang, "Quasi-threshold graphs," *Discrete App. Math.*, pp. 247-255, 1996.

[39] M. Yannakakis, "A polynomial algorithm for the min cut linear arrangement of trees," *Journal of the ACM*, pp. 950-988, 1985.

[40] J. Yuan and S. Zhou, "Optimal labelling of unit interval graphs," *Appl. Math. J. Chinese Univ. Ser.B(English edition)*, p. 1995, 337-344.

[41] Γκριμπαβιώτης Ανδρέας. (2015) Cutwidth Visualization Software. [Online]. https://github.com/Antreasgr/CutwidthVisualization

Παράρτημα

Στο παράρτημα αυτό παραθέτουμε τους πηγαίους κώδικες από τα κυριότερα προγράμματα που υλοποιήθηκαν στα πλαίσια της έρευνας για την παρούσα εργασία.

Για το πρώτο πρόγραμμα διαδραστικής αναπαράστασης γραφημάτων ο πηγαίος κώδικας παραλείπετε λόγω μεγέθους, αλλά υπάρχει στο διαδίκτυο στην σελίδα

Πηγαίος Κώδικας 1 Διαδραστική αναπαράσταση γραφημάτων

---

https://github.com/Antreasgr/CutwidthVisualization [41]

Το πρόγραμμα εξαντλητικής αναζήτησης βέλτιστων διατάξεων κόμβων σε σχεδόν κατωφλικά γραφήματα με παράλληλη επεξεργασία σε γλώσσα *C#*:

Πηγαίος Κώδικας 2 Εξαντλητική αναζήτηση σε σχεδόν κατωφλικά γραφήματα

```
// <copyright file="QuasiThresholdEnumeration.cs" company="Antreas">
// Copyright (c) Antreas. All rights reserved.
// </copyright>

namespace CliqueGraphsCsharp
{
    using System;
    using System.Collections.Generic;
    using System.Diagnostics;
    using System.Threading.Tasks;
    using Combinatorics.Collections;
    using Newtonsoft.Json;

    /// <summary>
    /// Represents a Node in the tree
    /// </summary>
    [JsonObject(MemberSerialization.OptIn)]
    public class TreeNode
    {
        /// <summary>
        /// Initializes a new instance of the <see cref="TreeNode"/>
class.
        /// </summary>
        /// <param name="size">The size of the node</param>
        public TreeNode(int size)
        {
            this.Size = size;
            this.Children = new List<TreeNode>();
            this.Parent = null;
        }

        /// <summary>
        /// Gets or sets a list of children
        /// </summary>
        public List<TreeNode> Children { get; set; }

        /// <summary>
```

```
        /// Gets or set the size of this node
        /// </summary>
        [JsonProperty("size")]
        public int Size { get; set; }

        /// <summary>
        /// Gets or sets the name of this node
        /// </summary>
        [JsonProperty]
        public string Name { get; set; }

        /// <summary>
        /// Gets or sets the parent of this node
        /// </summary>
        [JsonProperty]
        public string Parent { get; set; }

        /// <summary>
        /// Converts this node to a string
        /// </summary>
        /// <returns>A string</returns>
        public override string ToString()
        {
            return string.Format("{0}", this.Size);
        }
    }

    /// <summary>
    /// Represents a link in the tree
    /// </summary>
    public class Link
    {
        /// <summary>
        /// Gets or sets the source node
        /// </summary>
        [JsonProperty("source")]
        public int Source { get; set; }

        /// <summary>
        /// Gets or sets the target node
        /// </summary>
        [JsonProperty("target")]
        public int Target { get; set; }

        /// <summary>
        /// Initializes a new instance of the <see cref="Link"/>
class.
```

```
        /// </summary>
        /// <param name="source">the source node</param>
        /// <param name="target">the target node</param>
        public Link(int source, int target)
        {
            this.Source = source;
            this.Target = target;
        }

        /// <summary>
        /// Converts this link to a string
        /// </summary>
        /// <returns>A string</returns>
        public override string ToString()
        {
            return string.Format("Source: {0}, Target: {1}",
this.Source, this.Target);
        }
    }

    internal class Program
    {
        /// <summary>
        /// Calculates the cutwidth of a given order of node
        /// </summary>
        /// <param name="permutation">The order of the nodes</param>
        /// <param name="nodes">The node list</param>
        /// <param name="links">the link list</param>
        /// <returns>the cutwidth</returns>
        public static int CalculateCutWidth(IList<int> permutation,
List<TreeNode> nodes, List<Link> links)
        {
            // We now suppose that node[i] has order of perm[i]
            int i = 0;
            int cut = 0;
            foreach (var node in nodes)
            {
                int iCut = 0;
                int left = 0;
                int right = 0;
                int j = permutation[i];
                foreach (var link in links)
                {
                    var source = nodes[link.Source];
                    var sourceOrder = permutation[link.Source];
                    var target = nodes[link.Target];
                    var targetOrder = permutation[link.Target];
```

```
                    if ((sourceOrder > j && targetOrder < j) ||
(sourceOrder < j && targetOrder > j))
                    {
                        iCut += source.Size * target.Size;
                    }
                    else if (sourceOrder == j)
                    {
                        if (targetOrder < j)
                        {
                            left += target.Size;
                        }
                        else
                        {
                            right += target.Size;
                        }
                    }
                    else if (targetOrder == j)
                    {
                        if (sourceOrder < j)
                        {
                            left += source.Size;
                        }
                        else
                        {
                            right += source.Size;
                        }
                    }
                }

                for (int x = 0; x <= node.Size; x++)
                {
                    cut = Math.Max(cut, iCut + (x * (node.Size - x)) +
(x * right) + ((node.Size - x) * left));
                }

                i++;
            }

            return cut;
        }

        /// <summary>
        /// Traverses a tree with BFS
        /// </summary>
        /// <param name="n">the root node</param>
        /// <param name="nodes">a list of nodes</param>
        public static void Bfs(TreeNode n, List<TreeNode> nodes)
```

```
        {
            n.Name = nodes.Count.ToString();
            nodes.Add(n);
            foreach (var nn in n.Children)
            {
                nn.Parent = n.Name;
                Bfs(nn, nodes);
            }
        }

        /// <summary>
        /// Creates a tree
        /// </summary>
        /// <returns>the root node</returns>
        private static TreeNode InitTree()
        {
            var root = new TreeNode(1)
            {
                Children =
                {
                    new TreeNode(1)
                    {
                        Children =
                        {
                            new TreeNode(9),
                            new TreeNode(9)
                        }
                    },
                    new TreeNode(1)
                    {
                        Children =
                        {
                            new TreeNode(6),
                            new TreeNode(6)
                        }
                    }
                }
            };

            return root;
        }

        /// <summary>
        /// Helper to report progress
        /// </summary>
        /// <param name="current">the current permutation
number</param>
```

```
        /// <param name="max">the number of possible
permuations</param>
        /// <param name="elapsedMs">miliseconds from previous
call</param>
        private static void ReportProgress(int current, long max,
TimeSpan elapsedMs)
        {
            var remain = new TimeSpan((long)(max - current) *
(elapsedMs.Ticks / current));
            Console.Out.Write(string.Format("\r    {0}      {1}
{2}", Convert.ToInt32(current * 100.0 / max),
elapsedMs.ToString(@"hh\:mm\:ss"), remain.ToString(@"hh\:mm\:ss")));
        }

        private static void Main(string[] args)
        {
            List<TreeNode> nodes = new List<TreeNode>();
            List<Link> links = new List<Link>();
            Bfs(InitTree(), nodes);

            var integers = new List<int>();

            // Init nodes and links
            int i = 0;
            foreach (var node in nodes)
            {
                // make a list of integers of nodes.Count size for the
permutations
                integers.Add(i);

                var childs = new List<TreeNode>(node.Children);
                while (childs.Count > 0)
                {
                    var c = childs[childs.Count - 1];
                    childs.RemoveAt(childs.Count - 1);
                    if (c.Children.Count > 0)
                    {
                        childs.AddRange(c.Children);
                    }

                    links.Add(new Link(i, nodes.IndexOf(c)));
                }

                i++;
            }

            // Start the permutations loop
```

```
            var permutations = new Permutations<int>(integers,
GenerateOption.WithoutRepetition);
            long repeats = permutations.Count;
            int minCW = int.MaxValue;
            List<IList<int>> minOrders = new List<IList<int>>();
            Console.Out.WriteLine(" Percent   Elapsed    Remaining");
            i = 0;
            var watch = Stopwatch.StartNew();

            Parallel.ForEach(permutations, perm =>
            {
                i++;
                if (i % (1 * repeats / 100) == 0 || i == 1000000)
                {
                    ReportProgress(i, repeats, watch.Elapsed);
                }
                var cut = CalculateCutWidth(perm, nodes, links);
                if (cut < minCW)
                {
                    minCW = cut;
                    minOrders = new List<IList<int>>();
                    minOrders.Add(perm);
                }
                else if (cut == minCW)
                {
                    minOrders.Add(perm);
                }
            }
            );
            watch.Stop();

            Console.Out.WriteLine();
            Console.Out.WriteLine(string.Format("Min Cutwidth: {0},
#of Minimums: {1}", minCW, minOrders.Count));

            var json = JsonConvert.SerializeObject(new { nodes =
nodes, links = links, order = minOrders }, Formatting.Indented);
            System.IO.File.WriteAllText(@"test-21.json", json);

            return;
        }
    }
}
```

Το πρόγραμμα αναζήτησης βέλτιστων διατάξεων με χρήση του αλγορίθμου Simulated Annealing σε γλώσσα Python:

Πηγαίος Κώδικας 3 Αναζήτηση με Simulated Annealing (Anneal.py)

```
from __future__ import division
from __future__ import print_function
from __future__ import absolute_import
from __future__ import unicode_literals
import copy
import math
import sys
import time
import random


def round_figures(x, n):
    """Returns x rounded to n significant figures."""
    return round(x, int(n - math.ceil(math.log10(abs(x)))))


def time_string(seconds):
    """Returns time in seconds as a string formatted HHHH:MM:SS."""
    s = int(round(seconds))  # round to nearest second
    h, s = divmod(s, 3600)   # get hours and remainder
    m, s = divmod(s, 60)     # split remainder into minutes and
seconds
    return '%4i:%02i:%02i' % (h, m, s)


class Annealer(object):

    """Performs simulated annealing by calling functions to calculate
    energy and make moves on a state.  The temperature schedule for
    annealing may be provided manually or estimated automatically.
    """

    Tmax = 25000.0
    Tmin = 2.5
    steps = 50000
    updates = 100
    copy_strategy = 'deepcopy'
```

```
    def __init__(self, initial_state):
        self.initial_state = initial_state
        self.state = self.copy_state(initial_state)

    def set_schedule(self, schedule):
        """Takes the output from `auto` and sets the attributes
        """
        self.Tmax = schedule['tmax']
        self.Tmin = schedule['tmin']
        self.steps = int(schedule['steps'])

    def copy_state(self, state):
        """Returns an exact copy of the provided state
        Implemented according to self.copy_strategy, one of

        * deepcopy : use copy.deepcopy (slow but reliable)
        * slice: use list slices (faster but only works if state is
list-like)
        * method: use the state's copy() method
        """
        if self.copy_strategy == 'deepcopy':
            return copy.deepcopy(state)
        elif self.copy_strategy == 'slice':
            return state[:]
        elif self.copy_strategy == 'method':
            return state.copy()


    def update(self, step, T, E, acceptance, improvement):
        """Prints the current temperature, energy, acceptance rate,
        improvement rate, elapsed time, and remaining time.

        The acceptance rate indicates the percentage of moves since
the last
        update that were accepted by the Metropolis algorithm.  It
includes
        moves that decreased the energy, moves that left the energy
        unchanged, and moves that increased the energy yet were
reached by
        thermal excitation.

        The improvement rate indicates the percentage of moves since
the
        last update that strictly decreased the energy.  At high
        temperatures it will include both moves that improved the
overall
```

```
        state and moves that simply undid previously accepted moves
that
        increased the energy by thermal excititation.  At low
temperatures
        it will tend toward zero as the moves that can decrease the
energy
        are exhausted and moves that would increase the energy are no
longer
        thermally accessible."""

        elapsed = time.time() - self.start
        if step == 0:
            print(' Temperature        Energy    Accept   Improve
Elapsed   Remaining')
            sys.stdout.write('\r%12.2f  %12.2f                      %s
' % \
                (T, E, time_string(elapsed)))
            sys.stdout.flush()
        else:
            remain = (self.steps - step) * (elapsed / step)
            sys.stdout.write('\r%12.2f  %12.2f  %7.2f%%  %7.2f%%  %s
%s' % \
                (T, E, 100.0 * acceptance, 100.0 * improvement,
                    time_string(elapsed), time_string(remain))),
            sys.stdout.flush()

    def anneal(self):
        """Minimizes the energy of a system by simulated annealing.

        Parameters
        state : an initial arrangement of the system

        Returns
        (state, energy): the best state and energy found.
        """
        step = 0
        self.start = time.time()

        # Precompute factor for exponential cooling from Tmax to Tmin
        if self.Tmin <= 0.0:
            raise Exception('Exponential cooling requires a minimum "\
                "temperature greater than zero.')
        Tfactor = -math.log(self.Tmax / self.Tmin)

        # Note initial state
        T = self.Tmax
        E = self.energy()
```

```
        prevState = self.copy_state(self.state)
        prevEnergy = E
        bestState = self.copy_state(self.state)
        bestEnergy = E
        trials, accepts, improves = 0, 0, 0
        if self.updates > 0:
            updateWavelength = self.steps / self.updates
            self.update(step, T, E, None, None)

        # Attempt moves to new states
        while step < self.steps:
            step += 1
            T = self.Tmax * math.exp(Tfactor * step / self.steps)
            self.move()
            E = self.energy()
            dE = E - prevEnergy
            trials += 1
            if dE > 0.0 and math.exp(-dE / T) < random.random():
                # Restore previous state
                self.state = self.copy_state(prevState)
                E = prevEnergy
            else:
                # Accept new state and compare to best state
                accepts += 1
                if dE < 0.0:
                    improves += 1
                prevState = self.copy_state(self.state)
                prevEnergy = E
                if E < bestEnergy:
                    bestState = self.copy_state(self.state)
                    bestEnergy = E
            if self.updates > 1:
                if step // updateWavelength > (step - 1) //
updateWavelength:
                    self.update(
                        step, T, E, accepts / trials, improves /
trials)
                    trials, accepts, improves = 0, 0, 0

        # line break after progress output
        print('')

        # Return best state and energy
        return bestState, bestEnergy

    def auto(self, minutes, steps=2000):
```

```
        """Minimizes the energy of a system by simulated annealing
with
        automatic selection of the temperature schedule.

        Keyword arguments:
        state -- an initial arrangement of the system
        minutes -- time to spend annealing (after exploring
temperatures)
        steps -- number of steps to spend on each stage of exploration

        Returns the best state and energy found."""

        def run(T, steps):
            """Anneals a system at constant temperature and returns
the state,
            energy, rate of acceptance, and rate of improvement."""
            E = self.energy()
            prevState = self.copy_state(self.state)
            prevEnergy = E
            accepts, improves = 0, 0
            for step in range(steps):
                self.move()
                E = self.energy()
                dE = E - prevEnergy
                if dE > 0.0 and math.exp(-dE / T) < random.random():
                    self.state = self.copy_state(prevState)
                    E = prevEnergy
                else:
                    accepts += 1
                    if dE < 0.0:
                        improves += 1
                    prevState = self.copy_state(self.state)
                    prevEnergy = E
            return E, float(accepts) / steps, float(improves) / steps

        step = 0
        self.start = time.time()

        # Attempting automatic simulated anneal...
        # Find an initial guess for temperature
        T = 0.0
        E = self.energy()
        self.update(step, T, E, None, None)
        while T == 0.0:
            step += 1
            self.move()
            T = abs(self.energy() - E)
```

```
        # Search for Tmax - a temperature that gives 98% acceptance
        E, acceptance, improvement = run(T, steps)

        step += steps
        while acceptance > 0.98:
            T = round_figures(T / 1.5, 2)
            E, acceptance, improvement = run(T, steps)
            step += steps
            self.update(step, T, E, acceptance, improvement)
        while acceptance < 0.98:
            T = round_figures(T * 1.5, 2)
            E, acceptance, improvement = run(T, steps)
            step += steps
            self.update(step, T, E, acceptance, improvement)
        Tmax = T

        # Search for Tmin - a temperature that gives 0% improvement
        while improvement > 0.0:
            T = round_figures(T / 1.5, 2)
            E, acceptance, improvement = run(T, steps)
            step += steps
            self.update(step, T, E, acceptance, improvement)
        Tmin = T

        # Calculate anneal duration
        elapsed = time.time() - self.start
        duration = round_figures(int(60.0 * minutes * step / elapsed),
2)

        # Don't perform anneal, just return params
        return {'tmax': Tmax, 'tmin': Tmin, 'steps': duration}
```

Πηγαίος Κώδικας 3 Αναζήτηση με Simulated Annealing (CliqueAnnealer.py)

```
from __future__ import print_function
import math
import random
from anneal import Annealer
from CliqueGraphs import CliqueGraph, CliqueTree

class CliqueTreeProblem(Annealer):
    # pass extra data into the constructor
    def __init__(self, state, node_list, links_list):
        self.node_list = node_list
        self.links_list = links_list
        super(CliqueTreeProblem, self).__init__(state) # important!

    def move(self):
        """Swaps two nodes in the tree."""
        a = random.randint(0, len(self.state) - 1)
        b = random.randint(0, len(self.state) - 1)
        self.state[a], self.state[b] = self.state[b], self.state[a]

    def energy(self):
        """Calculates the cutwidth of the ordering."""
        cuts = []
        for j in xrange(len(self.node_list)):
            independentCut = 0
            left = 0
            right = 0
            for link in self.links_list:
                if ((self.state.index(link["source"]) > j and
self.state.index(link["target"]) < j) or
(self.state.index(link["source"]) < j and
self.state.index(link["target"]) > j)):
                    independentCut += next(x["size"] for x in
self.node_list if x["name"] == link["source"]) * next(x["size"] for x
in self.node_list if x["name"] == link["target"])
                elif (self.state.index(link["source"]) == j):
                    if (self.state.index(link["target"]) < j):
                        left += next(x["size"] for x in self.node_list
if  x["name"] == link["target"])
                    else:
                        right += next(x["size"] for x in
self.node_list if  x["name"] == link["target"])
                elif (self.state.index(link["target"]) == j):
```

```
                    if (self.state.index(link["source"]) < j):
                        left += next(x["size"] for x in self.node_list
if  x["name"] == link["source"])
                    else:
                        right += next(x["size"] for x in
self.node_list if  x["name"] == link["source"])

            cut = 0
            nodeSize = next(x["size"] for x in self.node_list if
x["name"] == self.state[j])
            for pp in xrange(nodeSize + 1):
                cut = max(cut, pp * (nodeSize - pp) + pp * right +
(nodeSize - pp) * left)
            cuts.append(independentCut + cut)

        return max(cuts)


    # state is list of node names
    # data {
    #   node { name: string, size: integer }
    #   links { source: string , target: string }
    # }
def main():
    nodes = [{ "name": "n00", "size": 4 } ,
              { "name": "n01", "size": 10 } ,
              { "name": "n02", "size": 8 } ,
              { "name": "n03", "size": 5 } ,
              { "name": "n04", "size": 3 }]

    links = [{"source": "n00" , "target": "n01" },
             {"source": "n00" , "target": "n02" },
             {"source": "n00" , "target": "n03" },
             {"source": "n00" , "target": "n04" },]

    initial_state = ["n00", "n01" , "n02", "n03", "n04"]

    trees = [CliqueGraph([10, 8, 5], 3)]
    trees.append(CliqueGraph([10, 5, 3, 1], 2))
    trees.append(CliqueGraph([4, 3, 2, 1, 1], 4))

    ct = CliqueTree([CliqueGraph([],10), CliqueGraph([8, 8, 7, 5, 4,
1],1)], 1)

    nodes, links = ct.toAnnealer()
    initial_state = [x["name"] for x in nodes]
    print(nodes)
```

```
    print(links)
    print(initial_state)


    p = CliqueTreeProblem(initial_state, nodes,links)

    p.copy_strategy = "slice"
    state, e = p.anneal()

    print(state, e)

if __name__ == '__main__':
    main()
```