ΠΕΙΡΑΜΑΤΙΚΗ ΜΕΛΕΤΗ ΑΛΓΟΡΙΘΜΩΝ ΧΡΩΜΑΤΙΣΜΟΥ ΣΕ ΤΕΛΕΙΑ ΓΡΑΦΗΜΑΤΑ

Η
ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ ΕΞΕΙΔΙΚΕΥΣΗΣ

Υποβάλλεται στην

ορισθείσα από την Γενική Συνέλευση Ειδικής Σύνθεσης
του Τμήματος Πληροφορικής
Εξεταστική Επιτροπή

από την

Σεβαστή Γαλάνη

ως μέρος των Υποχρεώσεων

για τη λήψη

του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ

ΜΕ ΕΞΕΙΔΙΚΕΥΣΗ ΣΤΗΝ ΘΕΩΡΙΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

Οκτώβριος 2006

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να απευθύνω τις ιδιαίτερες ευχαριστίες μου στον επιβλέποντα καθηγητή μου κ. Σταύρο Δ. Νικολόπουλο για τη βοήθεια και την υποστήριξή του, χωρίς την οποία η εργασία αυτή θα ήταν αδύνατο να υλοποιηθεί. Επίσης, θα ήθελα να τον ευχαριστήσω για την υπομονή που επέδειξε μέχρι την ολοκλήρωση της εργασίας αυτής. Τέλος θα ήθελα να ευχαριστήσω τους κ. Καθηγητές Λεωνίδα Παληό και Χρήστο Νομικό για τις παρατηρήσεις τους η συμβολή των οποίων ήταν πολύτιμη για ολοκλήρωση της εργασίας

Abstract

Graph Colouring is the procedure of colouring the nodes of a graph so that any two adjacent nodes do not have the same colour. In this thesis I will examine the behaviour of certain algorithms applied on perfect graphs, compared to the optimum colouring which is equivalent to the maximum graph clique. I will examine the behaviour of the following algorithms: MDsatur, greedy First Fit and optimum colour. In particular, the following graphs will be examined: permutation graphs, interval graphs, triangular graphs, cographs, complements of permutation graphs and complements of interval graphs. This thesis is structured as follows: initially I present the aforementioned graph classes, their properties and how to calculate the maximum clique of each graph. Next, I will examine the properties of the algorithms I have implemented. Following, I present the way I produce the graphs I used. Finally, experimental results are presented.

Περίληψη

Χρωματισμός ενός γραφήματος είναι η διαδικασία ανάθεσης χρώματος στους κόμβους του γραφήματος έτσι ώστε δυο γειτονικοί κόμβοι να μην έχουν το ίδιο χρώμα. Στην εργασία αυτή θα μελετηθεί η συμπεριφορά μερικών αλγορίθμων σε τέλεια γραφήματα κατά τον χρωματισμό αυτών συγκρίνοντάς τα αποτελέσματα με το βέλτιστο χρώμα που αντιστοιχεί στην μέγιστη κλίκα των γραφημάτων. Θα μελετηθεί η συμπεριφορά των γραφημάτων στον αλγόριθμο MDsatur, τον άπληστο First Fit και στο βέλτιστο χρώμα. Τα γραφήματα τα οποία θα παίρνουν μέρος στην εξέταση συμπεριφοράς αφορούν τα τέλεια γραφήματα και συγκεκριμένα, τα μεταθετικά γραφήματα, τα γραφήματα διαστημάτων, τα τριγωνικά, τα συμπληρωματικά παραγόμενα γραφήματα (cographs), τα συμπληρωματικά των μεταθετικών γραφημάτων και τέλος τα συμπληρωματικά των γραφημάτων διαστημάτων, όπου τα δυο τελευταία καλύπτουν μερικώς την κλάση των comparability γραφημάτων. Η εργασία διαρθρώνεται ως εξής: αρχικά θα δούμε τα γραφήματα αυτά και τις ιδιότητες τους, την μέγιστη κλίκα σε κάθε ένα, θα μελετήσουμε τους αντίστοιχους αλγορίθμους και την υλοποίησή τους. Ακολουθεί η παραγωγή των γραφημάτων και η πειραματική μελέτη της συμπεριφοράς των αλγορίθμων στα παραπάνω γραφήματα.

ΚΕΦΑΛΑΙΟ 1
ΙΔΙΟΤΗΤΕΣ ΓΡΑΦΗΜΑΤΩΝ

1.1 Τέλεια Γραφήματα

Στη θεωρία γραφημάτων ένα γράφημα είναι τέλειο όταν ο χρωματικός αριθμός του κάθε παραγόμενου υπογραφήματος είναι ίσος με τον αριθμό κλίκας αυτού του υπογραφήματος [wikipedia].

Για να μελετήσουμε τις ιδιότητες των τέλειων γραφημάτων παραθέτουμε κάποιες παραμέτρους ενός μη κατευθυνόμενου γραφήματος που θα χρησιμοποιηθούν:

- ω(G), ο αριθμός κλίκας του G: το μέγεθος του μεγαλύτερου πλήρους υπογραφήματος του G.
- χ(G), ο χρωματικός αριθμός του G: ο ελάχιστος αριθμός χρωμάτων που χρειάζεται για να χρωματιστούν κατάλληλα οι κόμβοι του G (δηλαδή οποιαδήποτε γειτονικοί κόμβοι να μην έχουν το ίδιο χρώμα), ή ο μικρότερος αριθμός ανεξάρτητων συνόλων που χρειάζονται για καλύψουν τους κόμβους του G.
- α(G), ο ευσταθής αριθμός του G: το μέγεθος του μεγαλύτερου σταθερού συνόλου του G (σταθερό σύνολο είναι ένα υποσύνολο X των κόμβων του γραφήματος που ανά δυο δεν είναι γείτονες , το λέμε και ανεξάρτητο σύνολο).
- k(G), ο αριθμός κλικών επικάλυψης του G: ο ελάχιστος αριθμός πλήρων υπογραφημάτων που χρειάζονται να καλύψουν τους κόμβους του G.

Η ένωση εκείνων των κόμβων που αποτελούν την κλίκα και του συνόλου του σταθερού αριθμού πρέπει να είναι τουλάχιστον ένας κόμβος.

Για ένα οποιοδήποτε γράφημα G ισχύει :

$$\omega(G) \leq \chi(G)$$

και $$\alpha(G) \leq k(G).$$

Οι παραπάνω ανισότητες είναι ισοδύναμες καθώς $\alpha(G) = \omega(\overline{G})$ και $k(G) = \chi(\overline{G})$.

Έστω G = (V,E) είναι ένα μη κατευθυνόμενο γράφημα. Εκείνα τα γραφήματα που ικανοποιούν τις ιδιότητες :

(P_1) $\omega(G_A) = \chi(G_A)$ (για όλα $A \subseteq V$)

(P_2) $\alpha(G_A) = k(G_A)$ (για όλα $A \subseteq V$)

λέγονται τέλεια [Golu80]. Είναι προφανές πως ένα γράφημα G ικανοποιεί την συνθήκη P_1 αν και μόνο αν το συμπληρωματικό του ικανοποιεί την P_2. Ο Berge [1961] τεκμηρίωσε ένα πιο ισχυρό αποτέλεσμα το οποίο μελετήθηκε και από τον Fulkerson [1969,1971,1972] και τελικά αποδείχτηκε από τον Lovász [1972a], ότι οι συνθήκες (P_1) και (P_2) είναι ισοδύναμες [Golu80].

Επιπρόσθετα ο Lovász [1972b] έδειξε πως στα τέλεια γραφήματα ισχύει και η :

(P3) $\omega(G_A)\, \alpha(G_A) \geq |A|$ (για κάθε $A \subseteq V$)

Οι παραπάνω συνθήκες P_1, P_2, P_3 είναι γνωστές και ως Θεώρημα Τέλειων Γραφημάτων (Lovász [1972b]) και είναι ισοδύναμες.

Επομένως, θα είναι αρκετό για ένα γράφημα να ικανοποιεί κάποια από τις (Pi) για να καταλήξουμε στο συμπέρασμα πως είναι τέλειο, και ένα τέλειο γράφημα θα ικανοποιεί την (P_1) και την (P_2) και την (P3) [Golu80].

Ένα γράφημα που ικανοποιεί την συνθήκη (P_1) λέγεται χ-τέλειο και αντίστοιχα α-τέλειο αν ικανοποιεί την συνθήκη (P_2). Το Θεώρημα Τέλειων Γραφήματων με βάση τα παραπάνω γίνεται ως εξής: ένα γράφημα είναι χ-τέλειο αν και μόνο αν είναι α-τέλειο.

Ένας άχορδος κύκλος περιττού μήκους ίσο τουλάχιστον με 5 λέγεται odd hole. Το συμπληρωματικό ενός odd hole γραφήματος λέγεται odd antihole. Ένα γράφημα που δεν περιέχει odd hole και odd antihole, ως επαγόμενο γράφημα, λέγεται γράφημα Berge. Από τα παραπάνω ένα τέλειο γράφημα είναι απαραίτητα ένα γράφημα Berge. Αυτό είναι και γνωστό ως ισχυρή εικασία τέλειων γραφημάτων [πηγή: wikipedia].

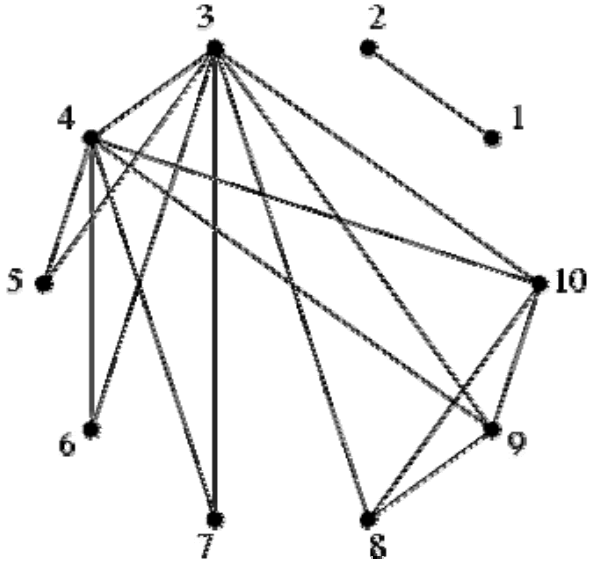
1.2 Μεταθετικά Γραφήματα

Μια διάταξη μετάθεση μιας ταξινομημένης λίστας S είναι μια αναδιάταξη των στοιχείων της S σε μια προς μια αντιστοιχία με τα στοιχεία της ίδιας της S. Ο αριθμός των μεταθέσεων ενός συνόλου n στοιχείων δίνεται από το n!

Ο ορισμός ενός μεταθετικού γραφήματος σύμφωνα με [N02] δίνεται παρακάτω. Έστω $\pi = (\pi_1,\pi_2, \dots ,\pi_n)$ μια διάταξη μετάθεσης πάνω στο σύνολο κόμβων $N_n = \{1,2,....,n\}$. Το σύνολο κόμβων V του γραφήματος $G[\pi] = (V,E)$ ορίζεται να είναι: το $V= N_n$ και 2 κόμβοι i , j είναι γειτονικοί αν και μόνο αν $(i - j)(\pi_i^{-1} - \pi_j^{-1}) < 0$ για όλα τα $i, j \in V$ όπου το π_i^{-1} είναι η θέση του στοιχείου i στην ακολουθία π.

Ένα γράφημα είναι γράφημα μετάθεσης αν υπάρχει μια μετάθεση π του N_n τέτοια ώστε το G γράφημα είναι ισομορφικό του $G[\pi]$. Το $G[\pi]$ λέγεται και αντεστραμμένο γράφημα της π (inversion graph).

Το παρακάτω γράφημα αντιστοιχεί στην μετάθεση π = [2, 1, 5, 6, 7, 10, 9, 4, 8, 3]:



Σχήμα 1.1: Permutation γράφημα

Σημειώνεται ότι σε μια τέτοια μετάθεση π η μέγιστη φθίνουσα ακολουθία της π είναι και η μέγιστη κλίκα του γραφήματος G[π] όπως θα δούμε και παρακάτω.

1.3 Τριγωνικά Γραφήματα

Μια από τις πρώτες κλάσεις που αναγνωρίστηκαν ως τέλεια γραφήματα είναι η κλάση των τριγωνικών γραφημάτων. Οι Hajnal και Suranyi [1958] έδειξαν ότι τα τριγωνικά γραφήματα ικανοποιούν την ιδιότητα P2 των τέλειων γραφημάτων (α-τέλεια) και ο Berge [1960] έδειξε ότι ικανοποιούν την ιδιότητα P1 (χ-τέλεια). Παρακάτω αναφέρουμε κάποιες από τις βασικές ιδιότητες των τριγωνικών γραφημάτων καθώς και τον ορισμό τους [Golu80].

Ένα μη κατευθυνόμενο γράφημα G λέγεται τριγωνικό εάν κάθε κύκλος μήκους αυστηρά μεγαλύτερου του 3 έχει χορδή, δηλαδή μια ακμή που ενώνει δυο μη

διαδοχικούς κόμβους του κύκλου. Ισοδύναμα το γράφημα G δεν περιέχει κανένα παραγόμενο υπογράφημα ισομορφικό του C_n (κύκλος μήκους n) με $n > 3$. Η τριγωνική ιδιότητα ενός γραφήματος G κληρονομείται σε όλα τα παραγόμενα υπογραφήματα του G. Επιπλέον τα γραφήματα διαστημάτων που θα εξεταστούν παρακάτω αποτελούν υποκλάση των τριγωνικών γραφημάτων [Golu80].

Στην βιβλιογραφία τα τριγωνικά γραφήματα ονομάζονται και χορδικά, rigid-circuit, μονότονα μεταβατικά και γραφήματα τέλειας απαλοιφής.

Ένας κόμβος x ενός γραφήματος G λέγεται μοναδικός (simplicial) αν κάθε γειτονικό του σύνολο Adj(x) παράγει ένα πλήρες υπογράφημα του G, το Adj(x) είναι κλίκα (όχι απαραίτητα μείζονα). Ο Dirac [1961] και αργότερα οι Lekkerkerker και Boland [1962] απέδειξαν ότι τα τριγωνικά γραφήματα έχουν πάντα μοναδικό κόμβο (στην πραγματικότητα τουλάχιστον 2 από αυτούς) και πάνω σε αυτό οι Fulkerson και Gross [1965] πρότειναν μια επαναληπτική διαδικασία για να αναγνωρίζουν τα τριγωνικά γραφήματα βασιζόμενοι στην κληρονομική ιδιότητα. Δηλαδή, εντοπίζεται ο μοναδικός κόμβος και απομονώνεται από το γράφημα. Η διαδικασία συνεχίζεται ώστε να μην μείνει κανένας κόμβος οπότε και καταλήγουμε στο συμπέρασμα πως το αρχικό γράφημα είναι τριγωνικό ή αν σε κάποιο στάδιο δε μείνει κανένας μοναδικός κόμβος το γράφημα δεν είναι τριγωνικό [Golu80].

Επιπρόσθετα, κάθε τριγωνικό γράφημα έχει τέλειο σχήμα απαλοιφής το οποίο δεν είναι μοναδικό. Σύμφωνα με τον [Golu80] εάν ένα γράφημα G είναι μη κατευθυνόμενο τότε τα παρακάτω είναι ισοδύναμα:

i. Το G είναι τριγωνικό

ii. Το G έχει ένα τέλειο σχήμα απαλοιφής και επιπλέον κάθε κόμβος μοναδικός μπορεί να ξεκινήσει ένα τέλειο σχήμα απαλοιφής

iii. Κάθε ελάχιστος διαχωριστής κόμβων παράγει ένα ολοκληρωμένο υπογράφημα του G.

Από το τέλειο σχήμα απαλοιφής του γραφήματος που προκύπτει θα υπολογίσουμε την μέγιστη κλίκα του γραφήματος.

1.4 Γραφήματα Διαστημάτων

Τα γραφήματα διαστημάτων σύμφωνα με τον [Golu80] είναι μια υποκλάση των τριγωνικών γραφημάτων. Αν το G είναι ένα μη κατευθυνόμενο γράφημα τότε τα παρακάτω είναι ισοδύναμα:

1. Το G είναι γράφημα διαστημάτων
2. Το G δεν περιέχει κύκλο μήκους μεγαλύτερου του 4 που να μην έχει χορδή και το συμπληρωματικό του G είναι comparability γράφημα.
3. Οι μείζονες κλίκες του G μπορούν γραμμικά να ταξινομηθούν έτσι ώστε για κάθε κόμβο x του G , οι μείζονες κλίκες περιέχουν το x.

Ακόμα ένα μη κατευθυνόμενο γράφημα G είναι γράφημα διαστημάτων αν και μόνο αν το G είναι τριγωνικό και το συμπληρωματικό του είναι comparability γράφημα.

Σύμφωνα με τον [Golu80] ένα γράφημα G είναι γράφημα διαστημάτων αν και μόνο αν ισχύει το παρακάτω:

1. το G είναι τριγωνικό και,
2. οποιοιδήποτε 3 κόμβοι του γραφήματος μπορούν να ταξινομηθούν με τέτοιο τρόπο έτσι ώστε κάθε μονοπάτι από τον πρώτο κόμβο στον τρίτο να περνά από τους γείτονες του δεύτερου.

Οι 3 κόμβοι που ικανοποιούν τη δεύτερη συνθήκη λέγονται astroidal triple.

Τα γραφήματα τομής (intersection γραφήματα) ανήκουν στην κλάση των γραφημάτων διαστημάτων. Εξ ορισμού, ένα γράφημα διαστημάτων αναπαρίσταται από ένα σύνολο διαστημάτων που βρίσκονται πάνω στην γραμμή των πραγματικών αριθμών.

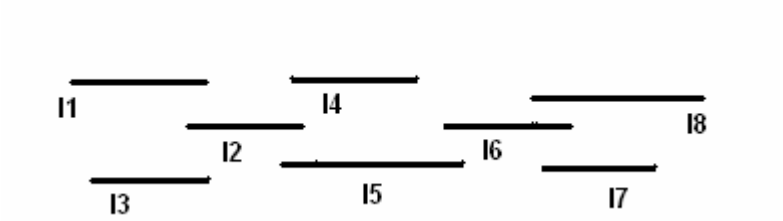
Κάθε κόμβος στο γράφημα αναπαρίσταται από ένα διάστημα και δυο κόμβοι συνδέονται αν τα αντίστοιχα διαστήματα επικαλύπτονται.

Πιο τυπικά, έστω: $I_1,I_2,\ldots,I_n \in R$ ένα σύνολο διαστημάτων. Το αντίστοιχο γράφημα G=(V,E) θα είναι αυτό για το οποίο:

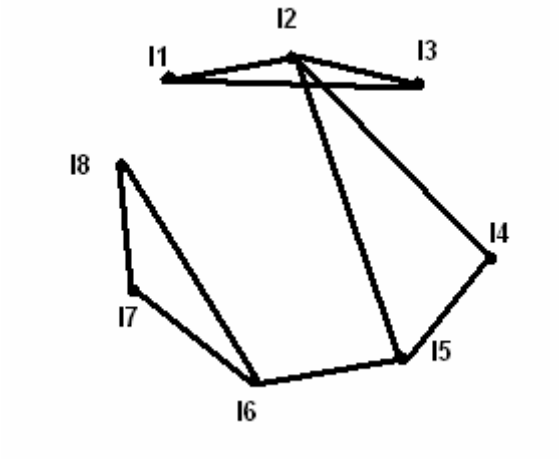
$$V = \{ I_1,I_2,\ldots,I_n \} \text{ και } (I_\alpha,I_\beta) \in E \Leftrightarrow I_\alpha \cap I_\beta \neq \varnothing.$$

Τα γραφήματα διαστημάτων είναι χορδικά γραφήματα άρα και τέλεια γραφήματα. Ακολουθεί μια αναπαράσταση γραφημάτων διαστημάτων.

Στο σχήμα 1.2 έχουμε την αναπαράσταση των διαστημάτων από $I_1,I_2,\ldots,I_8$ που αντιστοιχούν στους οχτώ κόμβους του γραφήματος και στο σχήμα 1.3 έχουμε το γράφημα που αντιστοιχεί στα διαστήματα αυτά.



Σχήμα 1.2: Αναπαράσταση Διαστημάτων, κάθε διάστημα αναπαραστά έναν κόμβο



Σχήμα 1.3: Το γράφημα διαστημάτων του σχήματος 1.2

Ελέγχουμε ένα διάστημα με όλα τα άλλα και ούτω καθ' εξής. Επομένως, κοιτάμε την αρχή και το τέλος του κάθε διαστήματος, εάν υπάρχει τομή μεταξύ των δύο αυτών κόμβων. Εκτενέστερη ανάλυση θα γίνει στο κεφάλαιο 3.

1.5 Συμπλήρωμα Παραγόμενων Γραφημάτων

Για τη μελέτη του συμπληρώματος παραγόμενων γραφημάτων θα κατασκευάσουμε πρώτα το αντίστοιχο δέντρο (cotree) και με βάση αυτό θα γίνει και η κατασκευή του αντίστοιχου γραφήματος.

Κάθε συμπληρωματικό παραγόμενο γράφημα θεωρεί ένα μοναδικό δέντρο [NP05], ως ισομορφικό που λέγεται cotree. Το cotree ενός τέτοιου γραφήματος είναι ένα δέντρο με ρίζα, τέτοιο ώστε:

1. Κάθε εσωτερικός κόμβος εκτός ίσως από την ρίζα έχει τουλάχιστον δύο παιδιά.
2. Οι εσωτερικοί κόμβοι έχουν ετικέτες είτε 0 είτε 1 (0-κόμβοι και 1-κόμβοι) : τα παιδιά ενός 1-κόμβου (ή 0-κόμβου αντίστοιχα) είναι 0-κόμβοι (ή 1-κόμβοι αντίστοιχα) π.χ. το 0 και το 1 εναλλάσσονται σε κάθε μονοπάτι από την ρίζα προς κάθε κόμβο του δέντρου.
3. Τα φύλλα ενός cotree είναι σε μια προς μια αντιστοιχία με τους κόμβους του γραφήματος G και δύο κόμβοι u_i και u_j είναι γείτονες αν και μόνο αν ο ελάχιστος κοινός προκάτοχος των φύλλων που αντιστοιχούν στους κόμβους u_i και u_j είναι 1-κόμβος.

Συνήθως η ρίζα είναι πάντα 1-κόμβος.

Τα συμπληρωματικά παραγόμενα γραφήματα (complement reducible graphs) έχουν τις εξής ιδιότητες :

1. ένας μοναδικός κόμβος είναι συμπληρωματικό παραγόμενο γράφημα
2. η ένωση δύο συμπληρωματικών παραγόμενων γραφημάτων είναι συμπληρωματικό παραγόμενο γράφημα
3. το συμπλήρωμα ενός συμπληρωματικού παραγόμενου γραφήματος είναι συμπληρωματικό παραγόμενο γράφημα

Με βάση τα παραπάνω θα γίνει η μελέτη τους στα κεφάλαια που θα ακολουθήσουν.

1.6 Συμπλήρωμα Μεταθετικών Γραφημάτων

Στη μελέτη συμπεριφοράς χρωματισμού γραφημάτων χρησιμοποιήθηκαν και τα συμπληρωματικά των μεταθετικών γραφημάτων και τα συμπληρωματικά των γραφημάτων διαστημάτων.

Το συμπληρωματικό γράφημα μεταθετικού γραφήματος είναι επίσης μεταθετικό γράφημα.

Ένα μη κατευθυνόμενο γράφημα είναι μεταθετικο γράφημα αν και μόνο αν το G και το $\overline{G}$ είναι comparability.

Για την κατασκευή αυτών των γραφημάτων χρησιμοποιήθηκε η μεταθετική διάταξη όπως στα αρχικά γραφήματα μετάθεσης μόνο που είναι αντιστραμμένη δηλαδή αν μια μεταθετική διάταξη ήταν η:

$$\pi = [\ 2, 8, 6, 5, 1, 7, 4, 9, 10, 3]$$

τότε είναι:

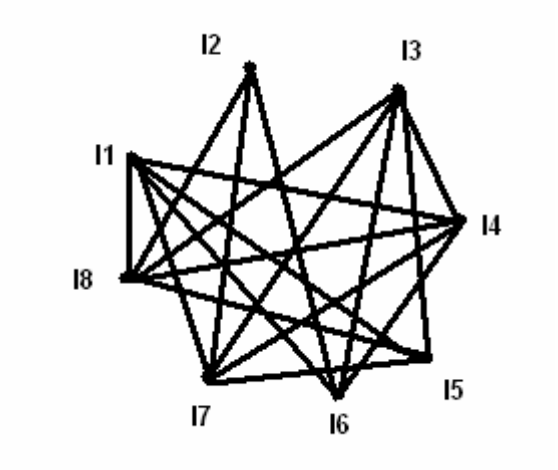
$$\overline{\pi} = [3, 10, 9, 4, 7, 1, 5, 6, 8, 2\ ]$$

Η μέγιστη φθίνουσα υπακολουθία και εδώ είναι η μέγιστη κλίκα, αντίστοιχα δηλαδή με τα μεταθετικά γραφήματα.

1.8 Συμπλήρωμα Γραφημάτων Διαστημάτων

Το συμπλήρωμα γραφήματος διαστημάτων G είναι comparability και το γράφημα G είναι τριγωνικό αν και μόνο αν το G είναι γράφημα διαστημάτων.

Ο υπολογισμός της μέγιστης κλίκας του γραφήματος γίνεται με την παραγωγή του αντίστοιχου κατευθυνόμενου γραφήματος και την συνάρτηση υπολογισμού ύψους.



Σχήμα 1.4: Το συμπληρωματικό του γραφήματος διαστημάτων του σχήματος 1.3

Στο σχήμα 1.4 αναπαρίσταται το συμπληρωματικό γράφημα διαστημάτων του σχήματος 1.3 για τα διαστήματα του σχήματος 1.2

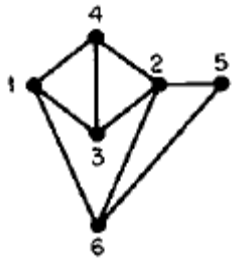
2.1 Μεταθετικά Γραφήματα

Σύμφωνα με τον [Golu80] εάν π είναι μια μετάθεση τότε τα παρακάτω είναι ισοδύναμα:

1. ο χρωματικός αριθμός του G[π]
2. ο ελάχιστος αριθμός των σειρών που απαιτούνται να ταξινομήσει την π
3. το μέγεθος της μέγιστης φθίνουσας υπακολουθίας της π

Η ισότητα μεταξύ του 1 και 3 έρχεται από το γεγονός ότι η μέγιστη φθίνουσα υπακολουθία σε γραφήματα μετάθεσης αντιστοιχεί στη μέγιστη κλίκα του γραφήματος και αφού τα γραφήματα μετάθεσης είναι τέλεια γραφήματα τότε η μέγιστη κλίκα συμπίπτει με τον χρωματικό αριθμό του γραφήματος. Ακολουθεί ένα παράδειγμα για τον υπολογισμό της μέγιστης κλίκας σε γράφημα μετάθεσης.

Έστω η παρακάτω μετάθεση π = [4, 3, 6, 1, 5, 2] και G το γράφημα που κατασκευάζεται από την π.



Σχήμα 2.1: Γράφημα της μετάθεσης π = [4, 3, 6, 1, 5, 2]

Στην παραπάνω ακολουθία η μέγιστη φθίνουσα υπακολουθία είναι η **4, 3, 2** ή και η **4, 3, 1** ή και η **6, 5, 2.** Παρατηρώντας το γράφημα βλέπουμε ότι αυτές οι υπακολουθίες αποτελούν και της μέγιστες κλίκες του γραφήματος μεγέθους 3.

Επομένως, ο υπολογισμός της μέγιστης κλίκας της παραπάνω κλάσης γραφημάτων είναι σχετικά απλός, αφού μπορούμε εύκολα να υπολογίσουμε τη μέγιστη φθίνουσα υπακολουθία σε ένα μονοδιάστατο πίνακα.

2.2 Τριγωνικά Γραφήματα και Γραφήματα Διαστημάτων

Σύμφωνα με τους Fulkerson και Gross [1965], ένα τριγωνικό γράφημα με n κόμβους έχει το πολύ n μείζονες κλίκες αν και μόνο αν το γράφημα δεν έχει ακμές.

Έστω ένα σύνολο S είναι ένας διαχωριστής κόμβων σε ένα συνδεδεμένο μη κατευθυνόμενο γράφημα G = (V,E) και G_{A1}, G_{A2}, … , G_{At} οι συνεκτικές συνιστώσες του G_{v-s} . Αν το S είναι κλίκα τότε σύμφωνα με τον [Golu80] :

$$\chi(G) = \max_i \chi(G_{S+A_i})$$

και

$$\omega(G) = \max_i \omega(G_{S+A_i}).$$

Το παραπάνω θεώρημα αποδεικνύει πως ένα τριγωνικό γράφημα είναι τέλειο.

Η μέγιστη κλίκα στα τριγωνικά γραφήματα υπολογίζεται από το τέλειο σχήμα απαλοιφής. Πιο συγκεκριμένα οι Leuker, Rose και Tarzan [1976] παρουσίασαν μια έκδοση αλγορίθμου που χρησιμοποιεί την λεξικογραφική αναζήτηση σε βάθος (lexicographic BFS) στην η οποία συνήθης ουρά των κόμβων της αναζήτησης σε βάθος (BFS) αντικαθιστάται από μία ουρά (μη ταξινομημένη) με υποσύνολα κόμβων η οποία κάποιες φορές μπορεί να βελτιωθεί αλλά ποτέ να ανασυνταχθεί [Golu80]. Ο αλγόριθμος έχει ως εξής:

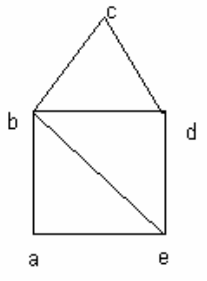
begin

1. assign the label ∅ to each vertex;
2. for i ← n to 1 step – 1 do
3. select: pick an unnumbered vertex u with largest label;
4. σ(i) ← u ; comment This assigns to u the number i
5. update: for each unnumbered vertex w∈Adj(u) do added i to label(w);

end

Αλγόριθμος 2.1 : Λεξικογραφική αναζήτηση σε βάθος.[Golu80]

Ακολουθεί η εφαρμογή του αλγορίθμου (2.1). Για τον υπολογισμό του τέλειου σχήματος απαλοιφής θα χρησιμοποιήσουμε το παρακάτω σχήμα:



Σχήμα 2.2: Τριγωνικό γράφημα για μελέτη τέλειου σχήματος απαλοιφής

Ξεκινάμε από τον κόμβο a. Το τέλειο σχήμα απαλοιφής αρχικά περιέχει τον κόμβο a, δηλαδή σ = [a]. Έπειτα επισκεπτόμαστε τον κόμβο που γειτνιάζει με τους περισσότερους κόμβους που βρίσκονται στο σ. Σε αυτό το βήμα, δύο κόμβοι είναι υποψήφιοι προς επιλογή: οι κόμβοι b και e. Τυχαία εδώ διαλέγουμε τον b. Επομένως, η ακολουθία σ γεμίζει διαδοχικά ως εξής:

$$\sigma = [a] , \sigma = [b,a], \ \sigma = [e,b,a], \ \sigma = [d,e,b,a], \ \sigma = [c,d,e,b,a]$$

Με βάση το τέλειο σχήμα απαλοιφής, μπορούμε να υπολογίσουμε την μέγιστη κλίκα του γραφήματος. Η διαδικασία είναι η ακόλουθη. Σαρώνουμε την ακολουθία από την αρχή εώς το τέλος και απαριθμούμε τον αριθμό των γειτόνων που έχει ο κάθε κόμβος στα δεξιά του. Δηλαδή στην σ =[c,d,e,b,a] για τον κόμβο c βρίσκουμε γείτονες τον b και d όπου το σύνολο όλων τον κόμβων μαζί και του c είναι 3. Όμοια για τον κόμβο d βρίσκω γείτονες τους b και e και πάλι σύνολο 3. Ο κόμβος e έχει τους b και a πάλι σύνολο 3 και τέλος ο b τον κόμβο a, σύνολο 2. Από τα παραπάνω προκύπτει πως η μέγιστη κλίκα του γραφήματος είναι το και ο μέγιστος αριθμός που βρίκαμε, στην προκειμένη περίπτωση 3.

Η μελέτη μέγιστης κλίκας στα γραφήματα διαστημάτων γίνεται ακριβώς με τον ίδιο τρόπο, αφού η κλάση αυτή των γραφημάτων υπάγεται στα τριγωνικά γραφήματα.

2.3 Συμπλήρωμα Συμπληρωματικών Παραγόμενων Γραφημάτων

Σύμφωνα με τους [GMR01] ένα συμπληρωματικό παραγόμενο γράφημα είναι γνωστό και ως P_4-free γράφημα . Με βάση τις ιδιότητες των γραφημάτων αυτών:

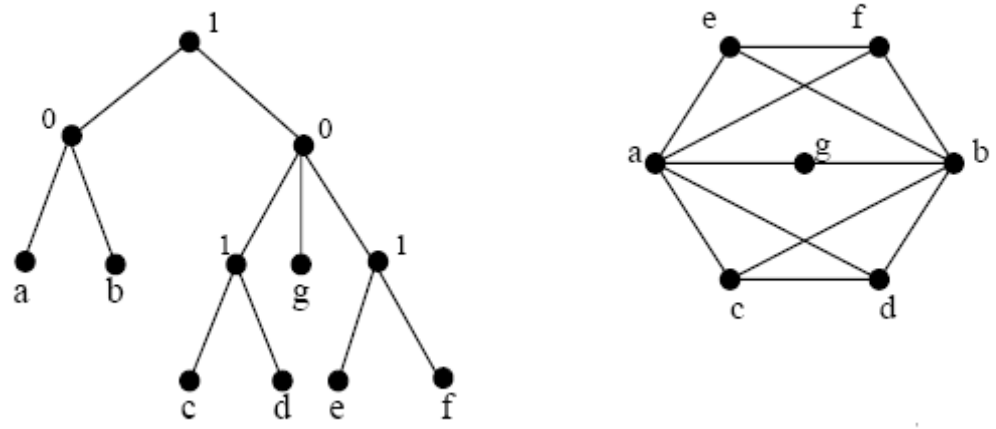
- Οποιοδήποτε υπογράφημα ενός συμπληρωματικού παραγόμενου γραφήματος είναι επίσης συμπληρωματικό παραγόμενο γράφημα
- Το συμπληρωματικό ενός συμπληρωματικού παραγόμενου γραφήματος είναι επίσης συμπληρωματικό παραγόμενο γράφημα
- Το συμπληρωματικό ενός συνδεδεμένου συμπληρωματικού παραγόμενου γραφήματος είναι μη συνδεδεμένο

Ένα τέτοιο γράφημα έχει μία μοναδική αναπαράσταση δέντρου που ονομάζεται cotree. Με βάση αυτό μπορούμε να παράγουμε και όλες τις μείζονες κλίκες. Υπενθυμίζεται παρακάτω η διαδικασία κατασκεύης ενός cotree.

Τα φύλλα του cotree είναι οι κόμβοι του αντίστοιχου συμπληρωματικού παραγόμενου γραφήματος. Κάθε εσωτερικός κόμβος εκτός πολύ πιθανόν της ρίζας έχει δυο ή περισσότερα παιδιά. Η ρίζα θα έχει μόνο ένα παιδί εάν το αντίστοιχο συμπληρωματικό παραγόμενο γράφημα είναι μη συνδεδεμένο.

Στους εσωτερικούς κόμβους ενός τέτοιου δέντρου βάζουμε ετικέτες ως εξής: Η ρίζα έχει ετικέτα 1. Τα παιδιά εσωτερικών κόμβων με τιμή 1, έχουν τιμή 0. Αντίστροφα, τα παιδιά εσωτερικών κόμβων με τιμή 0, έχουν τιμή 1. Δύο κόμβοι x και y ενώνονται στο γράφημα αν και μόνο αν ο ελάχιστος κοινός προκάτοχος στο αντίστοιχο δέντρο είναι 1.

Στο σχήμα 2.6 βλέπουμε δεξιά ένα συμπληρωματικό παραγόμενο γράφημα και αριστερά το αντίστοιχο δέντρο. Το γράφημα παρήχθει όπως περιγράφηκε προτύτερα. Στο τρίτο κεφάλαιο θα παρουσιαστεί η μέθοδος με την οποία παρήχθησαν τυχαία cotrees και πάνω σε αυτά κατασκευάστηκαν τα αντίστοιχα συμπληρωματικά παραγόμενα γραφήματα.

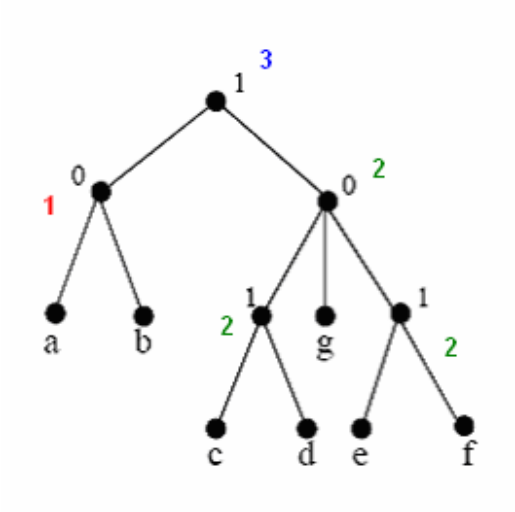


Σχήμα 2.3: Παράδειγμα Cotree και το αντίστοιχο συμπλήρωμα παραγόμενου γραφήματος (Cograph)

Σύμφωνα με τους [GMR01] στο cotree υπάρχει μια συνάρτηση με την οποία σχετίζεται. Η συνάρτηση αυτή είναι η $F = acd + aef + ag + bcd + bef + bg = (a+b)(cd + ef + g)$.

Για να υπολογιστεί η μέγιστη κλίκα του γραφήματος ακολουθείται η εξής διαδικασία: Ξεκινώντας από τα φύλλα και ανεβαίνοντας προς τα πάνω υπολογίζουμε σε κάθε 0-κόμβο την μέγιστη κλίκα των παιδιών του, ενώ σε κάθε 1-κόμβο προσθέτουμε τις κλίκες των παιδιών του. Επομένως, στο παραπάνω παράδειγμα αρχικά στο 0-κόμβο των φύλλων a και b θα ανέβει ο αριθμός 1. Στο δεξί υποδέντρο για τους κόμβους c και d θα ανέβει ο αριθμός 2 γιατί έχουμε 1-κόμβο όμοια και για τα φύλλα e και f . Στον ακριβώς από πάνω 0-κόμβο που ανήκουν τα φύλλα αυτά θα ανέβει η μέγιστη κλίκα των προηγουμένων αλλά και αυτή του αριθμού g που είναι άλλωστε 1. Επομένως ο δεξιά 0-κόμβος έχει κλίκα 2. Τελικά στην ρίζα θα ανεβεί το άθροισμα των κλίκων που έχουν οι 0-κόμβοι παιδιά του, δηλαδή σύνολο 3.

Στο παρακάτω σχήμα βλέπουμε σε κάθε κόμβο την τιμή της κλίκας με κόκκινο χρώμα



Σχήμα 2.4: Με κόκκινο είναι ο αριθμός κλίκας σε κάθε εσωτερικό κόμβο και στην ρίζα είναι μέγιστη κλίκα

Πραγματικά, παρατηρώντας το γράφημα του σχήματος 2.4 βλέπουμε πως η μέγιστη κλίκα που αντιστοιχεί σε αυτό είναι 3.

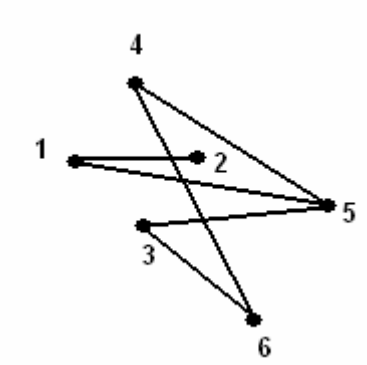
2.4 Συμπλήρωμα Μεταθετικών Γραφημάτων

Για τον υπολογισμό της μέγιστης κλίκας σε συμπληρωματικά μεταθετικά γραφήματα θα εξετάσουμε τι συμβαίνει αν αντιστρέψουμε την ακολουθία π. Κάθε ζευγάρι αριθμών που υπήρχε στην σωστή σειρά στην ακολουθία π τώρα θα είναι σε ανάποδη σειρά και το αντίστροφο. Οπότε το μεταθετικό γράφημα το οποίο προκύπτει είναι το συμπληρωματικό του G[π]. Επομένως εάν π^p είναι η μετάθεση που προκύπτει όταν αντιστρέφουμε την π τότε:

$$G[\pi^p] = \overline{G[\pi^p]}$$

Από το παραπάνω προκύπτει πως το συμπληρωματικό γράφημα ενός μεταθετικού γραφήματος είναι επίσης μεταθετικό γράφημα. [Golu80].

Αφού λοιπόν το συμπληρωματικό γράφημα ενός μεταθετικού γραφήματος είναι επίσης μεταθετικό γράφημα τότε ο υπολογισμός μέγιστης κλίκας γίνεται με ανάλογο τρόπο όπως γίνετε στην κανονική μεταθετική διάταξη.



Σχήμα 2.5: Γράφημα της $\overline{\pi}$ = [2, 5, 1, 6, 3, 4]

Σχηματικά, έστω η παρακάτω μετάθεση π = [4, 3, 6, 1, 5, 2] και G το γράφημα που κατασκευάζεται από την π. Η $\overline{\pi}$ θα είναι: $\overline{\pi}$ = [2, 5, 1, 6, 3, 4]. Στο σχήμα 2.8 αναπαρίσταται το γράφημα της ακολουθίας $\overline{\pi}$.

Από τις ιδιότητες των μεταθετικών γραφημάτων που αναφέρθηκαν πιο πάνω, ένα μη κατευθυνόμενο γράφημα είναι μεταθετικό αν και μόνο αν το G και το $\overline{G}$ είναι comparability. Άρα στην μελέτη παρακάτω έχουμε μια εικόνα για τα comparability γραφήματα και πιο συγκεκριμένα αυτά για τα οποία το συμπληρωματικό τους είναι μεταθετικό γράφημα.

2.5 Συμπλήρωμα Γραφημάτων Διαστημάτων

Από τις ιδιότητες των γραφημάτων διαστημάτων, προκύπτει ότι τα γραφήματα αυτά είναι τριγωνικά. Ακολουθεί η περίπτωση των συμπληρωματικών γραφημάτων διαστημάτων.

Στην κλάση αυτή των γραφημάτων μπορούμε να υπολογίσουμε τη μέγιστη κλίκα με την συνάρτηση ύψους με την προϋπόθεση ότι το γράφημα έχει την μεταβατική ιδιότητα. Πιο συγκεκριμένα σύμφωνα με τον [Golu80] σε οποιοδήποτε άκυκλο προσανατολισμό F (όχι απαραίτητα μεταβατικό) ενός μη κατευθυνόμενου γραφήματος G = (V, E) μπορούμε να συσχετίσουμε μια αυστηρή τμηματική διάταξη των κόμβων. Συγκεκριμένα, x > y αν και μόνο αν υπάρχει μη τετριμμένο μονοπάτι στην F από τον κόμβο x στον y. Μια συνάρτηση ύψους h μπορεί να εφαρμοστεί στο σύνολο κόμβων V ως ακολούθως:

$$h(u) = \begin{cases} 0 \text{ εάν ο κόμβος u είναι sink} \\ 1 + \max\{h(w) \mid uw \in F\} \end{cases}$$

Επιπρόσθετα ο [Golu80] απέδειξε πως ότι η συνάρτηση ύψους μπορεί να υπολογιστεί σε γραμμικό χρόνο χρησιμοποιώντας τον αλγόριθμο DFS. Η συνάρτηση h δίνει πάντα ένα χρωματισμό των κόμβων του γραφήματος, χωρίς να είναι απαραίτητα το ελάχιστο χρώμα. Ο αριθμός των χρωμάτων που χρησιμοποιήθηκαν είναι ίσος με τον αριθμό των κόμβων στο μεγαλύτερο μονοπάτι της F. Αυτό είναι ίσο και με το 1 + max { h(u) }u ∈ V}εφόσον ξεκινάμε από χρώμα 0. Μια λάθος επιλογή της F μπορεί να δώσει πλήρη χρωματισμό. Τα αποτελέσματα είναι πολύ καλύτερα εάν η συνάρτηση F είναι μεταβατική [Golu80].

Έστω ότι το G γράφημα είναι ένα comparability γράφημα και F είναι ο μεταβατικός προσανατολισμός του γραφήματος. Σε αυτή την περίπτωση κάθε μονοπάτι στην F αντιστοιχεί σε μια κλίκα του γραφήματος, λόγω μεταβατικότητας. Έτσι η συνάρτηση ύψους θα δώσει χρωματισμό που θα χρησιμοποιεί ακριβώς τόσα χρώματα όσα είναι και τα χρώματα του ω(G), που είναι το βέλτιστο δυνατό. Επίσης,

καθώς το γράφημα G είναι comparability και έχει κληρονομική ιδιότητα, προκύπτει ότι $\omega(G_A) = \chi(G_A)$, για όλα τα υπογραφήματα G_A του G [Golu80]. Από το παραπάνω καταλήγουμε στο συμπέρασμα πως κάθε comparability γράφημα είναι και τέλειο γράφημα [Golu80].

Στα συμπληρωματικά γραφήματα διαστημάτων κάνουμε το γράφημα μεταβατικό και από τον πίνακα που προκύπτει θα βγάλουμε την συνάρτηση ύψους ως εξής:

Έστω για ένα μεταβατικό γράφημα G(V, E) προκύπτει ο πίνακας γειτνίασης της παρακάτω μορφής:

$$A=\begin{matrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{matrix}$$

Σε πρώτη φάση, ο αλγόριθμος για την εύρεση βέλτιστου χρώματος εξετάζει ποια γραμμή του πίνακα είναι μηδενική και μηδενίζει την αντίστοιχη στήλη. Στην προκειμένη περίπτωση η 2^{η} γραμμή είναι μηδέν άρα θα γίνει 0 και η 2^{η} στήλη οπότε και προκύπτει ο νέος πίνακας :

$$A=\begin{matrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{matrix}$$

Στον παραπάνω πίνακα και πάλι αναδρομικά βρίσκει την 3^{η} γραμμή μηδενική (τις παλαιότερες που μηδένισε τις αγνοεί) επομένως θα μηδενίσει και την 3^{η} στήλη άρα ο νέος πίνακας θα γίνει:

$$A=\begin{matrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{matrix}$$

Τελείως ανάλογα εξετάζει και βρίσκει πως κενή είναι η $1^{η}$ γραμμή, άρα θα μηδενίσει την $1^{η}$ στήλη, οπότε και προκύπτει ο μηδενικός πίνακας:

$$A = \begin{matrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{matrix}$$

Ως συνάρτηση ύψους ορίζεται ο αριθμός των βημάτων που απαιτούνται για να γίνει μηδενικός ο πίνακας γειτνίασης. Εδώ για παράδειγμα, σε 3 βήματα ο πίνακας γίνεται μηδενικός άρα η συνάρτηση ύψους είναι 3 και κατ' επέκταση η μέγιστη κλίκα και ο χρωματικός αριθμός είναι 3.

3.1 Ο Αλγόριθμος Dsatur

Για τον χρωματισμό των γραφημάτων των οποίων οι ιδιότητες αναφέρθηκαν στο προηγούμενο κεφάλαιο χρησιμοποιήθηκε ο αλγόριθμος MDsatur. Ο αλγόριθμος αυτός είναι εμπνευσμένος από τον αλγόριθμο Dsatur του Brelaz [Bre79].

Σύμφωνα με τον [Bre79] ο Dsatur χρωματίζει άπληστα τους κόμβους ενός γραφήματος με βάση το μέγιστο βαθμό κορεσμού. Ακολουθεί ο ορισμός του βαθμού κορεσμού.

Έστω G (V,E) ένα γράφημα του οποίου οι κόμβοι έχουν χρωματιστεί πλήρως ή έχουν χρωματιστεί τμηματικά. Ο βαθμός κορεσμού ενός μη χρωματισμένου κόμβου, έστω $u \in V$, είναι ο αριθμός των διαφορετικών χρωμάτων που έχουν ήδη δοθεί στους γείτονες και το συμβολίζουμε με ds(u) [ANS03].

Ο Dsatur λειτουργεί ως εξής: χρωματίζει τους κόμβους ενός γραφήματος G, έναν κάθε φορά δεδομένης μιας δυναμική σειρά Ο΄. Κάθε φορά σε κάθε επανάληψη (έστω i η επανάληψη) βρίσκει ποιος από τους μη χρωματισμένους κόμβους του G θα μπει στην θέση της αντίστοιχης επανάληψης (i) της δυναμικής σειράς Ο΄. Το στοιχείο στη θέση i της ακολουθίας Ο΄ συμβολίζεται με O_i' και το στοιχείο αυτό το χρωματίζει με το ελάχιστο διαθέσιμο χρώμα. Έστω ότι ένας κόμβος u είναι το i-στο στοιχείο της Ο΄. Τότε ο u είναι ένας μη χρωματισμένος κόμβος, από το σύνολο των κόμβων του G, του οποίο ο βαθμός κορεσμού είναι ο μέγιστος ανάμεσα από όλους τους βαθμούς κορεσμού των μη χρωματισμένων κόμβων [ANS03].

3.2 Ο Αλγόριθμος MDsatur

Ο αλγόριθμος MDsatur χρωματίζει τους κόμβους ενός δοθέντος γραφήματος πχ G_{in} με τον ίδιο τρόπο όπως και ο Dsatur. Χρησιμοποιεί μια δυναμική σειρά Ο, που περιέχει τους κόμβους του γραφήματος και τους χρωματίζει άπληστα διαλέγοντας κάθε φορά τον αμέσως επόμενο κόμβο με το ελάχιστο διαθέσιμο χρώμα. Ο αλγόριθμος MDsatur διαφέρει λιγάκι από τον Dsatur στον τρόπο με τον οποίο σχηματίζει τη σειρά Ο. Για να δούμε την διαφορά μεταξύ των δύο αλγορίθμων, έστω ότι θα συμβολίζουμε με Ο την ακολουθία που σχηματίζεται κατά την εκτέλεση του αλγορίθμου MDsatur και με Ο΄ την ακολουθία που θα σχηματιστεί από την εκτέλεση του αλγορίθμου Dsatur. Το στοιχείο στην i θέση της ακολουθίας Ο (δηλ. O_i) ικανοποιεί περισσότερες απαιτήσεις από ότι το αντίστοιχο στοιχείο στην ακολουθία Ο΄(το O_i'). Τις απαιτήσεις αυτές (R_2 και R_3)θα τις δούμε αναλυτικά παρακάτω στην παρουσίαση του αλγορίθμου [ANS03].

Ο αλγόριθμος MDsatur πηγάζει από την συμπεριφορά του Dsatur (Brelaz) και του αλγορίθμου NoChoice (Turner) που είναι μια περιορισμένη έκδοση του Dsatur. Ο τελευταίος χρωματίζει ένα τυχαίο γράφημα αποτελεσματικά με υψηλή πιθανότητα, ενώ ο NoChoice έχει ακόμα καλύτερη συμπεριφορά σε k-«χρωματίσιμο» τυχαίο γράφημα όταν το $k < \log n$ και η πιθανότητα ακμής ½ [ANS03].

Ο MDsatur χρωματίζει τους κόμβους ενός γραφήματος G βασισμένος σε μια σειρά Ο διαφορετική από αυτή που δίνει ο Dsatur, η οποία αυξάνεται με κόμβους που ικανοποιούν περισσότερους περιορισμούς. Πιο συγκεκριμένα έστω i είναι το τρέχον σημείο του MDsatur στην εφαρμογή του G, u ο τρέχον μη χρωματισμένος κόμβος που πρόκειται να γίνει ο κόμβος στην θέση i της σειράς Ο. Ο κόμβος u ικανοποιεί τις ακόλουθες απαιτήσεις στην σειρά:

- **R1:** Έχει την μέγιστη τιμή βαθμού κορεσμού στο σημείο αυτό του αλγορίθμου (όπως και ο Dsatur)

- **R2:** Έχει τον πιο παλιά χρωματισμένο κόμβο γείτονα (δηλ αθτόν στην αριστερότερη θέση της σειράς Ο ανάμεσα στους πιο παλιούς γείτονες από όλους τους μη χρωματισμένους κόμβους)

- **R3:** Είναι ο πιο κοντινός γείτονας κόμβος με αυτόν που χρωματίστηκε πιο πρίν και βρίσκεται στην σειρά στην θέση O_{i-1}

Ακολουθεί ο ψευδοκώδικας για τον αλγόριθμο MDsatur:

Algorithm: *MDsatur*
Input: a graph $G = (V, E)$. *Output*: a colouring of G.
Begin

1. (a) Set each vertex of G as uncoloured, and
   set a dynamic order on them, called O, as empty.
   (b) Select any vertex of G.
   Add it in the beginning of O (in the position 1 of O),
   and colour it with colour 1.
2. **For** $i = 2$ **to** $n = |V|$ **do**
   (a) Set X to be the set of the current uncoloured vertices
   which have the maximum value on their *degree of saturation*
   (Requirement R1).
   (b) **For** each vertex $v \in X$ **do**
   Find the first neighbour of v coloured
   (i.e. its oldest coloured neighbour) **(Requirement R2)**.
   (c) Select from X the vertex v which has the oldest coloured neighbour, denoted by ON, and also has the maximum number of common neighbours with the last coloured vertex (namely O_{i-1})
   (Requirement R3).
   If more than one of the vertices of X satisfy these requirements,
   then select at random one of them with equal probability.
   If none of the uncoloured neighbours of ON, which are in X, has common
   neighbours with vertex O_{i-1}, then select one of them at random.
   Set $O_i = v$.
   (d) Colour v with the smallest of its allowable colours, i.e. are not assigned to its neighbours.
   If none of the colours used is allowable for this vertex, insert a new colour.
3. Return the colour of each vertex.

End

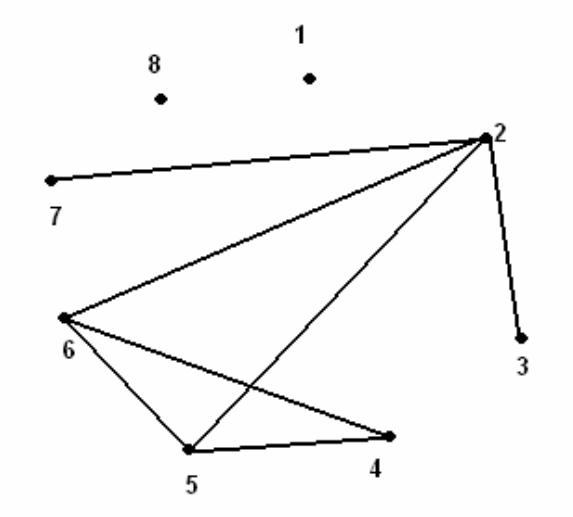
Σχήμα 3.1:Ψευδοκώδικας του αλγορίθμου MDsatur[ANS03].

Ο πιο κοντινός γείτονας του κόμβου u είναι εκείνος του οποίου το χρώμα καθορίζεται κυρίως από τα χρώματα των υπολοίπων γειτόνων του u. Όσον αφορά τα επίπεδα γραφήματα που μελετήθηκαν στην εργασία [ANS03], αυτός ο γείτονας του u βρίσκεται στο κοντινότερο ενσωματωμένο επίπεδο γράφημα του G. Σύμφωνα με την ίδια μελέτη, ο MDsatur χρωματίζει καλύτερα από τον Dsatur τα επίπεδα γραφήματα. Ο βασικός λόγος είναι πως ο MDsatur χρωματίζει πιο τοπικά και συμπαγώς το δοθέν γράφημα βασισμένο στον πιο παλιά χρωματισμένο κόμβο. Αυτή η ιδιότητα του MDsatur δικαιολογεί και την συμπεριφορά του στο χρωματισμό των τέλειων γραφημάτων που θα παρουσιαστεί στο κεφάλαιο 5.

Ειδικότερα, όσον αφορά τις απαιτήσεις οι οποίες είναι και οι μόνες που διαφοροποιούν τον αλγόριθμο από τον Dsatur, παρατηρούμε πως η απαίτηση R2 είναι αυτή που πρέπει να ικανοποιείται από τον επόμενο προς χρωματισμό κόμβο σχετικά με τον βαθμό κορεσμού του. Αντίστοιχα και η απαίτηση R3, είναι αυτή η οποία φροντίζει και διαλέγουμε κόμβο πιο κοντά στον πιο παλιό χρωματισμένο κόμβο. Αυτό σημαίνει πως πρώτα θα χρωματιστεί η κλίκα ενός κόμβου, ή διαφορετικά η γειτονιά του, και κατ΄ επέκταση η μέγιστη κλίκα του πιο παλιού κόμβου, αφού ο αλγόριθμος είναι αυτός που κινείται κοντά στους κόμβους με τον μέγιστο βαθμό κορεσμού λόγω της προηγούμενης απαίτησης.

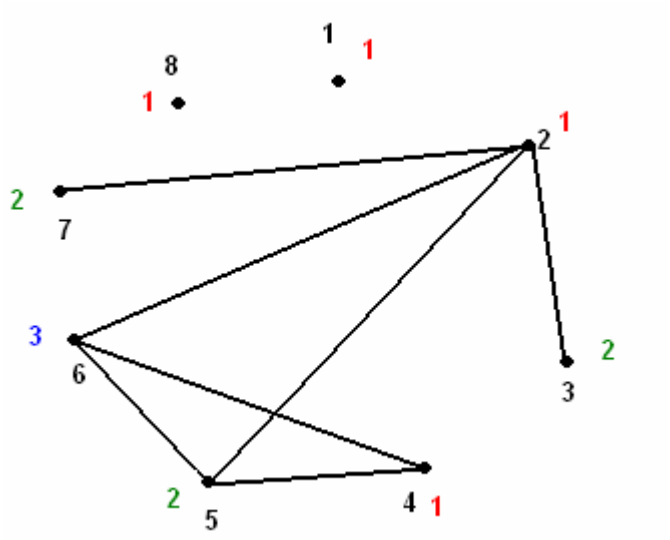
Στη συνέχεια της εργασία αυτής, θα δούμε πως η συμπεριφορά του MDsatur είναι επίσης πολύ καλή και μάλιστα πολύ καλύτερη από αυτή του άπληστου αλγορίθμου greedy. Αυτό συμβαίνει λόγω της ιδιότητας τοπικού χρωματισμού του γραφήματος που προαναφέρθηκε.

Για να δούμε πώς δουλεύει ο αλγόριθμος αυτός, θα κάνουμε μια εφαρμογή στο παρακάτω γράφημα που είναι ένα γράφημα μετάθεσης πάνω στην διάταξη: $\pi = \{1, 3, 6, 5, 4, 7, 2, 8\}$ οπότε και προκύπτει το γράφημα που του σχήματος 3.2.



Σχήμα 3.2: Μεταθετικό Γράφημα της ακολουθίας π = { 1, 3, 6, 5, 4, 7, 2, 8 }

Εφαρμόζοντας τον παραπάνω αλγόριθμο στο γράφημα αυτό, αρχικά ο κόμβος 1 θα μπει στην διάταξη O(1) = 1 και το χρώμα για τον κόμβο 1 θα είναι το 1. Επειδή ο κόμβος 1 δεν γειτνιάζει με κανέναν, πηγαίνουμε στον κόμβο 2 (σειριακά αυτόν με το χαμηλότερο index) με O(2)=2 και το χρώμα του 2 είναι 1. Στην ακολουθία X_1 (η ακολουθία X στην $1^{η}$ επανάληψη) μπαίνουν πλέον οι κόμβοι 3, 5, 6, 7 με βαθμό κορεσμού όλοι ίσο με 1. Από αυτούς, όλοι μπορεί να θεωρηθούν ΟΝ κόμβοι, επομένως διαλέγουμε αυτούς που έχουν περισσότερος κοινούς γείτονες με τον 2 που είναι ο 6 και ο 5 , και διαλέγουμε αυτόν με τον μικρότερο index που είναι ο 5 και παίρνει χρώμα 2, άρα O(3)=5. Η νέα διάταξη του είναι X_2= {6, 4, 3, 7} με τον 6 να έχει τον μέγιστο βαθμό από όλα άρα και είναι ο επόμενος που χρωματίζεται που έχει και τους περισσότερους κοινούς κόμβους με τον 2 που χρωματίστηκε πιο παλιά άρα O(4)=6 με χρώμα στον 6 να παίρνει το max των γειτόνων του +1 άρα 3. Η ακολουθία X_3={4, 3, 7} και ο 4 έχει τον μεγαλύτερο βαθμό κορεσμού οπότε και χρωματίζεται αμέσως μετά με χρώμα 1 το ελάχιστο διαθέσιμο, O(5) = 4 . Η νέα διάταξη X_4={3, 7} και οι δύο θεωρούνται κόμβοι ΟΝ και δεν έχουν κοινούς γείτονες με τον πιο παλιά χρωματισμένο τυχαία παίρνουμε το 3 και O(6)=3 και τέλος ο κόμβος 7 οπότε και η O(7) = 7 και χρώμα 2. Τελικά η διάταξη O = {1, 2, 5, 6, 4, 3, 7, 8} και χρώματα 1, 1, 2, 3, 1, 2, 2, 1 με μέγιστο χρώμα 3. Το γράφημα με τα χρώματα απεικονίζεται στο Σχήμα 3.3:



Σχήμα 3.3: Χρωματισμός γραφήματος σχήματος 3.2 με τον αλγόριθμο MDsatur

Η υλοποίηση του αλγορίθμου MDsatur έγινε με βάση τον ψευδοκώδικα του [ANS03]

.

Πιο συγκεκριμένα, στον κώδικα για την συνάρτηση του αλγορίθμου MDsatur χρησιμοποιήθηκε ένας πίνακας *(int *A)*ως πίνακας γειτνίασης, με δυναμική δέσμευση μνήμης. Οι βοηθητικοί πίνακες για την προσωρινή αποθήκευση των χρωμάτων καθώς και ένας τελικός πίνακας για τα τελικά χρώματα υλοποιούνται με δυναμική δέσμευση μνήμης. Οι έλεγχοι που προκύπτουν στον κώδικα ακολουθούν την δομή του ψευδοκώδικα του σχήματος 3.1.

3.3 Ο Αλγόριθμος First Fit

Ο αλγόριθμος First Fit είναι ένας online αλγόριθμος που παρέχει γρήγορο χρωματισμό αλλά δεν δίνει το βέλτιστο αποτέλεσμα. Ο συγκεκριμένος αλγόριθμος χρωματίζει άπληστα ένα γράφημα δίνοντας το κατάλληλο χρώμα στον τρέχοντα κόμβο. Για να χρωματίσει ένα γράφημα εκτελείται η ακόλουθη διαδικασία: ξεκινά δίνοντας στον πρώτο κόμβο το χρώμα 1 και συνεχίζει με τους υπόλοιπους δίνοντας το ελάχιστο διαθέσιμο χρώμα. Διαφορετικά δίνει το μέγιστο χρώμα των γειτόνων του συν ένα.

Ο αλγόριθμος First Fit έχει χρησιμοποιηθεί στην πειραματική μελέτη σε όλες τις κλάσεις των γραφημάτων χρησιμοποιώντας μια τυχαία ακολουθία κόμβων. Ωστόσο, στην κλάση των μεταθετικών γραφημάτων και στο συμπλήρωμα των μεταθετικών γραφημάτων έχει ακολουθηθεί διαφορετική διαδικασία η οποία και περιγράφεται στο κεφάλαιο 4.

4.1 Γραφήματα Μετάθεσης

Η κατασκευή μιας μετάθεσης γίνεται με παραγωγή τυχαίων αριθμών με τη χρήση της συνάρτησης srand ως εξής:

*srand((unsigned int)time((time_t *)NULL));*

και με την κατάλληλη συνθήκη έτσι ώστε να σχηματιστεί μια μεταθετική διάταξη. Αμέσως μετά και με βάση τη θεωρία για τα μεταθετικά γραφήματα κατασκευάζουμε τον πίνακα γειτνίασης του γραφήματος.

Έστω η παρακάτω μετάθεση:

2	3	8	9	6	7	1	4	5

Με βάση τον τρόπο κατασκευής των μεταθετικών γραφημάτων θα προκύψει ο εξής πίνακας γειτνίασης:

$$A = \begin{matrix} 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \end{matrix}$$

Αυτό τον πίνακα τον χρησιμοποιούμε και στην συνάρτηση του αλγορίθμου MDsatur. Με βάση τον πίνακα και την ακολουθία βρίσκουμε την μέγιστη φθίνουσα ακολουθία του γραφήματος που είναι και η κλίκα του G.

Για την υλοποίηση του αλγορίθμου First Fit ακολουθούμε την εξής διαδικασία:

Έστω Α μία μετάθεση Ν αριθμών:

A

2	3	8	9	6	7	1	4	5

Έπειτα και με το ίδιο Ν κατασκευάζουμε μία δεύτερη ακολουθία:

B

7	6	9	3	2	1	4	5	8

Δηλαδή προκύπτει η αρχική ακολουθία με διαφορετική σειρά. Ο χρωματισμός θα γίνει στην ακολουθία Β, δηλαδή ελέγχουμε έναν έναν αριθμό στην Β ψάχνοντας την θέση του στην ακολουθία Α.

- Οπότε στην παραπάνω ακολουθία ο αριθμός **7** έχει χρώμα **1**
- Ελέγχουμε την ακολουθία Β και ο επόμενος κόμβος είναι ο **6.** Τον ψάχνουμε στην Α και βλέπουμε πως δεν έχουμε χρωματίσει κανέναν γείτονά του μέχρι στιγμής οπότε και παίρνει τον αριθμό **1**.
- Ο κόμβος **9** έχει 2 γείτονες τους 6, 7 με χρώμα 1, οπότε το χρώμα του είναι το **2**
- Ο κόμβος **3** στην Α δεν έχει κανένα γείτονα, επομένως παίρνει το διαθέσιμο χρώμα που είναι το **1**(ή και 2).
- Ο κόμβος **2** δεν έχει πάλι κανέναν χρωματισμένο κόμβο στην Α οπότε θα πάρει το χρώμα **1.**
- Ο **1** έχει 2 γείτονες χρωματισμένους, τους 2, 3, 6, 7 με χρώματα 1 και 2 οπότε παίρνει το μέγιστο των γειτόνων του συν 1 άρα **3.**
- Ο κόμβος **4** έχει γείτονες το 9, 6 και 7 με χρώματα 2, 1, 1 αντίστοιχα οπότε το χρώμα που παίρνει είναι το **3.**
- Ο κόμβος **5** έχει γείτονες το 9, 6 και 7 με χρώματα 2, 1, 1 αντίστοιχα οπότε το χρώμα που παίρνει είναι επίσης το **3.**

- Τέλος ο **8** έχει γείτονες 6, 7, 1, 4, 5 με χρώματα 1, 1, 3, 3, 3 οπότε και παίρνει ένα από τα διαθέσιμα, το **2.**

Από τα παραπάνω προκύπτει ότι το μέγιστο χρώμα είναι το 3 που είναι και ο χρωματικός αριθμός του γραφήματος.

4.2 Τριγωνικά Γραφήματα

Για την κατασκευή ενός τέτοιου γραφήματος ακολουθήθηκε η παρακάτω διαδικασία:

Παίρνουμε μια τυχαία ακολουθία αριθμών όσοι και οι κόμβοι του γραφήματος. Για παράδειγμα για ένα γράφημα με 7 κόμβους έστω η τυχαία ακολουθία:

$$A = \{3,\ 7,\ 2,\ 1,\ 6,\ 5,\ 4\}$$

και για κάθε ένα από τους κόμβους επιλέγουμε τυχαίους γείτονες από αυτούς που βρίσκονται στα δεξιά τους και έστω προκύπτουν τα παρακάτω:

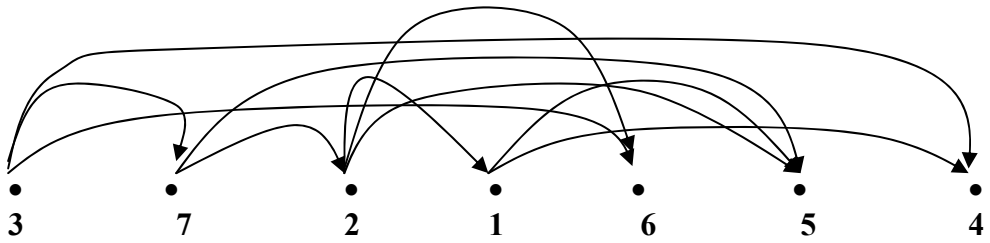
Κόμβος	***Γείτονες***
3	4
	6
	7

Κόμβος	***Γείτονες***
7	2
	5

Κόμβος	***Γείτονες***
2	5
	6
	1

Κόμβος	***Γείτονες***
1	5
	4

Ο κόμβος 6 δεν έχει κανένα γείτονα. Όμοια ο 5 και ο 4.



Σχήμα 4.1:Ακολουθία για την κατασκευή τριγωνικού γραφήματος

Αρχικά ενώνουμε τους γείτονες του καθενός όπως προέκυψαν από την τυχαία παραγωγή, δηλαδή τον κόμβο 3 με τους 4, 6, 7, τον 7 με τους 2, 5, τον 2 με τους 5, 6,

1, τον 1 με τους 5, 4 και τους 6, 5, 4 με κανέναν. Η αρχική ακολουθία που προκύπτει είναι αυτή του σχήματος 4.1.

Σαρώνουμε την ακολουθία από τα αριστερά προς τα δεξιά και για κάθε κόμβο ελέγχουμε ποιους επιπλέον γείτονες έχει προς τα δεξιά του. Παρατηρούμε ότι ο 7 ενώνεται εκτός από τον 2 και τον 5 και με τον 4 τον 6 και τον 3. Επομένως όλοι αυτοί ως γείτονες θα πρέπει να ενωθούν και με αυτούς που βρίσκονται προς τα δεξιά δηλαδή και τον 7 με τον 4 και τον 6 αλλά και με όλους τους δυνατούς συνδιασμούς των 2,5 με τους 4,6. Δεν έχουμε ένωση αυτών των κόμβων με τον 3 γιατί είναι στα αριστερά του. Έτσι κατασκευάζεται ο αντίστοιχος πίνακας γειτνίασης.

Στα τριγωνικά γραφήματα ο υπολογισμός της μέγιστης κλίκας έχει γίνει με τον υπολογισμό ενός τέλειου σχήματος απαλοιφής (Perfect Elimination Ordering).

4.3 Γραφήματα Διαστημάτων

Αρχικά, σε ένα διάστημα από το 0 έως το 200 παράγουμε N τυχαία διαστήματα (α, β) με οποιοδήποτε εύρος. Υπάρχει ο περιορισμός όπου ο αριθμός α είναι μικρότερος από τον β. Συγκρίνουμε το κάθε ένα διάστημα με το επόμενο και αν καλύπτονται τότε η συγκεκριμένη συνάρτηση επιστρέφει 1 διαφορετικά 0. Έτσι συμπληρώνεται και ο πίνακας γειτνίασης τον οποίο και χρησιμοποιούμε στον αλγόριθμο MDsatur και First Fit.

4.4 Συμπληρωματικά Παραγόμενα Γραφήματα

Για να κατασκευάσουμε ένα συμπληρωματικό παραγόμενο γράφημα κατασκευάζουμε πρώτα το δέντρο που του αντιστοιχεί και μετά με βάση τις ιδιότητες που έχουν αναφερθεί προκύπτει το γράφημα. Για την κατασκευή του αντίστοιχου δέντρου κατασκευάστηκαν αρχικά τα φύλλα, τα οποία σύμφωνα με τον ορισμό είναι όσα και οι κόμβοι του γραφήματος. Από εκεί και έπειτα με μια τυχαία παραγωγή διαλέγονται τα φύλλα από το $1^{ο}$ μέχρι και αυτό με ετικέτα «1+αριθμός παιδιών που έμειναν» / 2. Αυτοί είναι οι κόμβοι που θα ενωθούν. Έπειτα διαλέγονται τυχαία από τους υπόλοιπους κόμβους, αλλά με μικρότερο εύρος, αυτοί που θα παραμείνουν

όπως είναι σε αυτή την φάση και θα ενωθούν στην επόμενη φάση. Αυτοί που πρόκειται να ενωθούν στην $1^{η}$ φάση ενώνονται με έναν κόμβο – πατέρα. Έτσι οι κόμβοι που πρόκειται να ενωθούν στην επόμενη φάση είναι οι «πατέρες» των κόμβων της πρώτης φάσης και οι κόμβοι φύλλα που έμειναν στην πρώτη φάση χωρίς να ενωθούν. Η διαδικασία συνεχίζεται αναδρομικά προς τα πάνω μέχρι που στο τέλος να ενωθούν οι εναπομείναντες κόμβοι με τον κόμβο-ρίζα.

Μετά την κατασκευή του δέντρου θα πρέπει να δοθούν ετικέτες στους εσωτερικούς κόμβους. Ξεκινώντας από την ρίζα που έχει την ετικέτα 1 κατεβαίνουμε προς τα φύλλα και ονοματίζουμε εναλλάσσοντας το 0 και το 1 σε κάθε εσωτερικό κόμβο.

Η κατασκευή του γραφήματος σύμφωνα με τον ορισμό γίνεται ως εξής: δύο κόμβοι στο γράφημα ενώνονται αν και μόνο αν ο ελάχιστος κοινός προκάτοχος των φύλλων αυτών στο δέντρο είναι 1. Οπότε προκύπτει και ο πίνακας γειτνίασης που χρησιμοποιείται στον MDsatur και First Fit. Όσον αφορά την μέγιστη κλίκα σε συμπληρωματικά παραγόμενα γραφήματα, ακολουθήθηκε ο αλγόριθμος που περιγράφηκε στο προηγούμενο κεφάλαιο.

4.5 Συμπλήρωμα Μεταθετικών Γραφημάτων

Στην περίπτωση των συμπληρωματικών μεταθετικών γραφημάτων κατασκευάζουμε μια μεταθετική διάταξη π και την αντιστρέφουμε σε $\overline{\pi}$. Πάνω σε αυτήν εφαρμόζουμε ακριβώς ότι και στα μεταθετικά γραφήματα στην κατασκευή του πίνακα γειτνίασης, τον υπολογισμό μέγιστης κλίκας, τον αλγόριθμο First Fit καθώς και τον MDsatur.

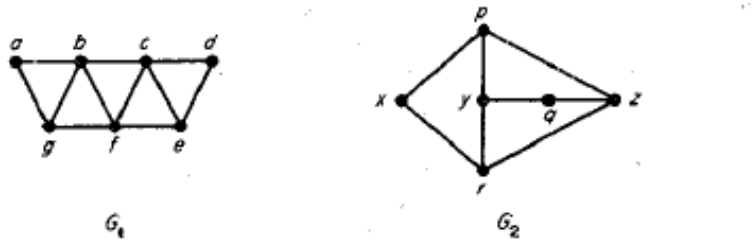
4.6 Συμπλήρωμα Γραφημάτων Διαστημάτων

Τέλος, κατά την κατασκευή των συμπληρωματικών των γραφημάτων διαστημάτων κάνουμε και πάλι αντίστροφη διαδικασία από αυτή που έγινε στα

κατασκευή των απλών γραφημάτων διαστημάτων. Αφού κατασκευάσουμε τα διαστήματα, ελέγχουμε ανά δυο τα γραφήματα και όπου καλύπτονται βάζουμε 0 στον αντίστοιχο πίνακα γειτνίασης και 1 αν δεν καλύπτονται. Στην περίπτωση αυτής της κλάσης γραφημάτων, τα διαστήματα σχηματίζονται ακολουθιακά στον άξονα των τιμών και αυτό γίνεται με σκοπό τη δημιουργία γραφήματος με μεταβατική ιδιότητα. Για τον υπολογισμό της μέγιστης κλίκας χρησιμοποιούμε την συνάρτηση ύψους από όπως αύτη παρουσιάστηκε στο κεφάλαιο 3.

5.1 Τριγωνικά Γραφήματα

Στην κλάση των τέλειων γραφημάτων ανήκουν και τα τριγωνικά γραφήματα. Όπως αναφέρθηκε στο κεφάλαιο 3, τα τριγωνικά γραφήματα έχουν την τριγωνική ιδιότητα, δηλαδή, κάθε κύκλος μήκους μεγαλύτερος του 3 έχει χορδή. Στο σχήμα 4.1 παρουσιάζονται δυο γραφήματα: το G_1 που είναι τριγωνικό, και το μη τριγωνικό G_2.



Σχήμα 4.1: G_1 :Τριγωνικό Γράφημα. G_2 :Μη ΤριγωνικόΓράφημα.

Στο παραπάνω σχήμα το γράφημα G1 έχει την τριγωνική ιδιότητα οπότε είναι και τριγωνικό σε αντίθεση με το γράφημα G2 το οποίο έχει κύκλο μήκους 4 (r, y, q, z, r) και δεν είναι τριγωνικό.

Με βάση τις ιδιότητες των τριγωνικών γραφημάτων, πάνω σε ένα τέτοιο γράφημα μπορούμε να υπολογίσουμε ένα τέλειο σχήμα απαλοιφής το οποίο όμως δεν είναι μοναδικό. Από το τέλειο σχήμα απαλοιφής και τον πίνακα γειτνίασης του κάθε κόμβου σύμφωνα με το [Golu80] μπορούμε υπολογίσουμε τον αριθμό της μέγιστης κλίκας καθώς και τους κόμβους που την αποτελούν.

Στη συνέχεια παρουσιάζονται τα αποτελέσματα από την εκτέλεση των αλγορίθμων MDsatur και First Fit για γραφήματα με 20, 50, 100 και 1000 κόμβους. Ακόμη παρατίθεται και το βέλτιστο χρώμα των γραφημάτων που αντιστοιχεί στην μέγιστη κλίκα.

20 Επαναλήψεις	N	ω(G)	MDsatur	First Fit
	20	5,05	5,05	5,3
	50	9,5	9,5	10,1
	100	13,85	13,85	14,5
	500	59,75	59,75	60,75
	1000	119,4	119,4	121,55
	2000	227,1	22,7	229,4
	4000	473,7	473,7	478,65

Πίνακας 5.1 : Αποτελέσματα τριγωνικών γραφημάτων για εκτέλεση 20 φορές

50 Επαναλήψεις	N	ω(G)	MDsatur	First Fit
	20	5,4	5,4	5,8
	50	9,8	9,8	10,3
	100	13,6	13,6	14
	500	59,6	59,6	60,6
	1000	117,4	117,4	119
	2000	241,4	241,4	244,4
	4000	474,2	474,2	479,5

Πίνακας 5.2 : Αποτελέσματα τριγωνικών γραφημάτων για εκτέλεση 50 φορές

100 Επαναλήψεις	N	ω(G)	MDsatur	First Fit
	20	5,5	5,5	5,7
	50	8,5	8,5	8,9
	100	13,3	13,3	13,84
	500	61,3	61,3	62,3
	1000	119	119	120,7
	2000	239	239	242
	4000	475,42	475,42	480,5

Πίνακας 5.3: Αποτελέσματα τριγωνικών γραφημάτων για εκτέλεση 100 φορές

1000 Επαναλήψεις	N	ω(G)	MDsatur	First Fit
	20	5,3	5,3	5,68
	50	8,9	8,9	9,43
	100	14,5	14,5	43,25
	500	60,2	60,2	61,4
	1000	119,2	119,2	121,1
	2000	237,8	237,8	240,5
	4000	475,2	475,2	480,2

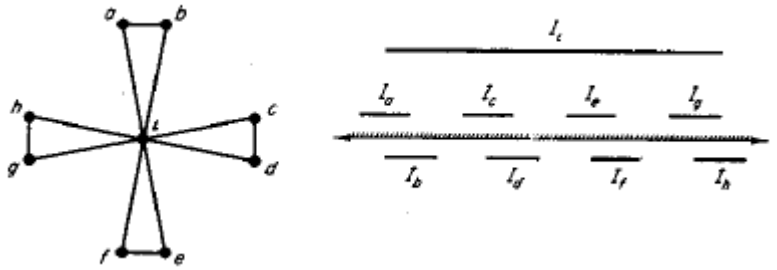
Πίνακας 5.4: Αποτελέσματα τριγωνικών γραφημάτων για εκτέλεση 1000 φορές

Βασιζόμενοι στα παραπάνω αποτελέσματα παρατηρούμε πως ο αλγόριθμος MDsatur δίνει το βέλτιστο αποτέλεσμα.Υπάρχει ταύτιση δηλαδή με το χρωματικό αριθμό των γραφημάτων.

Στην περιγραφή του αλγορίθμου στο κεφάλαιο 3 αναφέρθηκε πως ο αλγόριθμος αυτός βασίζεται στον άπληστο αλγόριθμο Dsatur, που χρωματίζει τους κόμβους ανάλογα με τον μέγιστο βαθμό κορεσμού, με την διαφορά πως στον MDsatur ο επόμενος προς χρωματισμό κόμβος διαλέγεται με βάση τους κοινούς κόμβους που έχει με τον πιο παλιά χρωματισμένο κόμβο. Αντίστοιχα και το τέλειο σχήμα απαλοιφής για να σχηματιστεί, κινείται στην γειτονιά του κόμβου που έχει τους περισσότερους επισκεπτόμενους κόμβους. Δηλαδή στη μεν μία περίπτωση για να χρωματιστεί ένας κόμβος κινούμαστε κοντά σε αυτόν που χρωματίστηκε πιο παλιά και έχει διαλεχτεί με βάση το βαθμό κορεσμού. Στη δε άλλη περίπτωση, κινούμαστε σε αυτόν που έχει γείτονες που έχουμε επισκεφτεί πιο πολύ. Και οι δύο αλγόριθμοι χαρακτηρίζονται από τοπικότητα, για αυτό και δίνουν ίδια αποτελέσματα. Ο First Fit δίνει διαφορετικά αποτελέσματα. Όχι βέλτιστα ούτε όμως και με μεγάλη απόκλιση.

5.2 Γραφήματα Διαστημάτων

Σύμφωνα με τον ορισμό, ένα γράφημα διαστημάτων είναι το intersection γράφημα των διαστημάτων πάνω σε μια ευθεία και δυο κόμβοι ενώνονται αν τα αντίστοιχα διαστήματα υπερκαλύπτονται.



Σχήμα 5.2: Ένα γράφημα διαστημάτων και η αναπαράστασή του

Για τον χρωματισμό αυτών των διαστημάτων χρησιμοποιήσαμε τον αλγόριθμο MDsatur, τον άπληστο αλγόριθμο First Fit που παίρνει τυχαία κόμβους και τους χρωματίζει. Τα αποτελέσματα τα συγκρίνουμε με αυτά του χρωματικού αριθμού.

Από τις ιδιότητες των γραφημάτων διαστημάτων που αναφέρθηκαν γνωρίζουμε πως ένα τέτοιο γράφημα έχει την τριγωνική ιδιότητα, δηλαδή δεν περιέχει κύκλο μεγαλύτερου του 3 (κάθε κύκλος μεγέθους 3 και πάνω έχει χορδή).

Για τον υπολογισμό της μέγιστης κλίκας σε γραφήματα διαστημάτων χρησιμοποιήθηκε τέλειο σχήμα απαλοιφής, όπως δόθηκε στο κεφάλαιο 3 [Golu80 σελ98]. Τα γραφήματα διαστημάτων είναι και τέλεια γραφήματα, δηλαδή ο χρωματικός αριθμός τους είναι ίσος με την μέγιστη κλίκα του γραφήματος.

Παρακάτω θα δούμε πως τα αποτελέσματα ενός First Fit αλγορίθμου σε γραφήματα διαστημάτων δεν είναι βέλτιστα. Ωστόσο, ο χρωματικός αριθμός που επιστρέφει ο αλγόριθμος MDsatur είναι ίδιος με αυτόν της μέγιστης κλίκας του γραφήματος.

Εφόσον τα γραφήματα διαστημάτων είναι υποκλάση των τριγωνικών γραφημάτων, τα αποτελέσματα που προκύπτουν από τους δυο αλγόριθμους σε σχέση με αυτά του χρωματικού αριθμού είναι ανάλογα με αυτά που προέκυψαν στα τριγωνικά γραφήματα. Επομένως, λόγω τοπικότητας του αλγορίθμου και της διαδικασίας εύρεσης μέγιστης κλίκας όπως αναφέρθηκε στην ενότητα 4.1, τα αποτελέσματα του χρωματικού αριθμού συμπίπτουν με αυτά του MDsatur

Τα παρακάτω αποτελέσματα προέκυψαν μετά από εκτέλεση των αλγορίθμων MDsatur και First Fit μετά από εκτέλεση για 20, 50, 100, 1000 φορές για γραφήματα με 20, 50, 100, 500, 1000, 2000, 4000 κόμβους αντίστοιχα. Υπάρχει και η στήλη με τα αποτελέσματα του χρωματικού αριθμού για σύγκριση

20 Επαναλήψεις

N	ω(G)	MDsatur	First Fit
20	9,5	9,5	9,75
50	21	21	21,85
100	40,7	40,7	40,8
500	193,19	193,19	198,69
1000	383,25	383,25	393
2000	763,9	763,9	779,79
4000	1515,3	1515,3	1539,25

Πίνακας 5.5 : Αποτελέσματα γραφημάτων διαστημάτων για εκτέλεση 20 φορές

50 Επαναλήψεις

N	ω(G)	MDsatur	First Fit
20	9,74	9,74	10,1
50	21,84	81,84	22,68
100	41,82	41,82	43,43
500	194,96	194,96	200,139
1000	385,51	385,51	395
2000	764,14	764,14	778,4
4000	1516,2	1516,2	1540

Πίνακας 5.6 : Αποτελέσματα γραφημάτων διαστημάτων για εκτέλεση 50 φορές

100 Επαναλήψεις	N	ω(G)	MDsatur	First Fit
	20	9,67	9,67	9,89
	50	21,6	21,6	22,34
	100	41,74	41,74	43,16
	500	197,25	197,25	202,85
	1000	383,39	383,39	392,89
	2000	760	760	776,4
	4000	1526,5	1526,5	1548,96

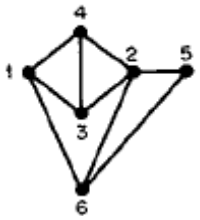
Πίνακας 5.7: Αποτελέσματα γραφημάτων διαστημάτων για εκτέλεση 100 φορές

1000 Επαναλήψεις	N	ω(G)	MDsatur	First Fit
	20	9,6	9,6	9,95
	50	21,7	21,7	22,47
	100	41,74	41,74	43,25
	500	202	202	207,25
	1000	385	385	394,519
	2000	763,34	763,34	778,69
	4000	1554,3	1554,3	1573,5

Πίνακας 5.8: Αποτελέσματα γραφημάτων διαστημάτων για εκτέλεση 1000 φορές

5.3 Αποτελέσματα αλγορίθμων για μεταθετικά γραφήματα

Ένα μεταθετικό γράφημα, όπως περιγράφηκε και στο δεύτερο κεφάλαιο, είναι ένα γράφημα G=(V,E) αν και μόνο αν υπάρχει μια διάταξη $\pi = (\pi_1,\pi_2, \dots ,\pi_n)$ πάνω σε ένα σύνολο κόμβων V={1,2,3,…,n}τέτοιο ώστε (i ,j)∈ E αν και μόνο αν $(i - j)(\pi_i^{-1} - \pi_j^{-1}) < 0$ για όλα τα i, j ∈ V, όπου το π_i^{-1} είναι ο δείκτης του στοιχείου i στην ακολουθία π. [NP00] . Μια αναπαράσταση ενός μεταθετικού γραφήματος φαίνεται στο γράφημα που ακολουθεί, για την διάταξη: G = {4,3,6,1,5,2 }



Σχήμα 5.4: Μεταθετικό Γράφημα

Ακολουθούν τα αποτελέσματα για την εκτέλεση των αλγορίθμων MDsatur και Fist Fit καθώς και ο χρωματικός αριθμός για γραφήματα 20, 50, 100, 500, 1000, 2000, 4000 κόμβων.

20 Επαναλήψεις

N	ω(G)	MDsatur	First Fit
20	6	6,1	6,45
50	10,2	10,25	11,5
100	14,9	15,15	18,2
500	34,7	36,79	48,049
1000	50,65	55,75	72,08
2000	72,75	81,09	109,3
4000	98,8	112,5	160,3

Πίνακας 5.9 : Αποτελέσματα γραφημάτων μετάθεσης για εκτέλεση 20 φορές

50 Επαναλήψεις	N	ω(G)	MDsatur	First Fit
	20	6,1	6,2	6,68
	50	10,66	10,72	12,36
	100	14,8	15,2	18,059
	500	35,24	37,7	48,36
	1000	49,82	54,32	72,1997
	2000	89,18	97,4	112,94
	4000	110,72	124,16	166,539

Πίνακας 5.10 : Αποτελέσματα γραφημάτων μετάθεσης για εκτέλεση 50 φορές

100 Επαναλήψεις	N	ω(G)	MDsatur	First Fit
	20	6,08	6,24	6,67
	50	10,27	10,67	12,25
	100	15,07	15,44	18,23
	500	35,07	37,7	47,84
	1000	49,95	54,369	71,59
	2000	70,98	79,33	108,05
	4000	104,4	117,76	162,88

Πίνακας 5.11: Αποτελέσματα γραφημάτων μετάθεσης για εκτέλεση 100 φορές

1000 Επαναλήψεις	N	ω(G)	MDsatur	First Fit
	20	6,1	6,209	6,641
	50	10,12	10,154	11,522
	100	14,857	15,172	17,957
	500	34,896	37,36	47,59
	1000	49,98	54,539	72,124
	2000	71,3	79,47	108,3
	4000	110,2	117,2	161,9

Πίνακας 5.12: Αποτελέσματα γραφημάτων μετάθεσης για εκτέλεση 1000 φορές

Σύμφωνα με το [Golu80] σε ένα μεταθετικό γράφημα G επί μίας μετάθεσης π τα παρακάτω είναι όλα ισοδύναμα:

1. ο χρωματικός αριθμός του G[π]
2. το μήκος της μέγιστης φθίνουσας υποακολουθίας π

3. ο ελάχιστος αριθμός των ουρών που χρειάζονται για να ταξινομηθεί η π.

Από τα αποτελέσματα των παραπάνω πειραμάτων προκύπτει πως ο αλγόριθμος MDsatur χρωματίζει καλύτερα ένα γράφημα από ότι ο First Fit. Τα αποτελέσματα για την μέγιστη κλίκα είναι παρόμοια με αυτά των συμπληρωματικών μεταθετικών γραφημάτων και αυτό αναλύεται πιο κάτω. Τα καλά αποτελέσματα που δίνει ο MDsatur συγκρίνοντάς τα με αυτά της μέγιστης κλίκας είναι αναμενόμενα λόγω της τοπικότητας του πρώτου.

5.4 Συμπληρωματικά Παραγόμενα Γραφήματα

Όπως έχει αναφερθεί και στις ιδιότητες τους, τα Συμπληρωματικά Παραγόμενα Γραφήματα μπορούν να αναπαρασταθούν μοναδικά από ένα δέντρο το οποίο ονομάζεται cotree. Βάσει αυτού έχει γίνει και η κατασκευή των γραφημάτων .

Ένα συμπληρωματικό παραγόμενο γράφημα ανήκει στην κλάση των τέλειων γραφημάτων, επομένως έχει όλες εκείνες τις ιδιότητες που διέπουν αυτή την κλάση γραφημάτων [BoCe].

Η μέγιστη κλίκα του cotree, και κατ' επέκταση του συμπληρωματικού παραγόμενου γραφήματος, έχει υπολογιστεί στο κεφάλαιο 2. Για κάθε εσωτερικό κόμβο 0 κρατάμε την μέγιστη κλίκα των παιδιών του ενώ στους εσωτερικούς κόμβους που είναι 1 προσθέτουμε τις κλίκες των παιδιών του.

Τα συμπληρωματικά παραγόμενα γραφήματα ονομάζονται και P_4–free γραφήματα, δηλαδή δεν περιέχουν άχορδο μονοπάτι 4 κόμβων. Η κλάση αυτή των γραφημάτων είναι ειδική περίπτωση των γραφημάτων μετάθεσης.

Παρατηρούμε στα παρακάτω πειράματα, ότι τα αποτελέσματα που επιστρέφει ο MDsatur και η μέγιστη κλίκα των συμπληρωματικών παραγόμενων γραφημάτων ταυτίζονται. Ακόμα παρατηρούμε πως ο αλγόριθμος First Fit χρωματίζει βέλτιστα τα γραφήματα αυτά.

Η παραπάνω διαπίστωση είναι αναμενόμενη αφού σύμφωνα με τους [BoCe] ο First Fit αλγόριθμος χρωματίζει βέλτιστα ένα συμπληρωματικό παραγόμενο γράφημα ανεξάρτητα με την σειρά των κόμβων που έρχονται για χρωματισμό.

Τα αποτελέσματα που ακολουθούν είναι για την εκτέλεση των συμπληρωματικών παραγόμενων γραφημάτων για 20, 50, 100, 500, 1000, 2000, 4000 κόμβους και η εκτέλεση είναι επίσης για 20, 50, 100 και 1000 φορές για τον κάθε αλγόριθμο. Αντίστοιχα και ο χρωματικός αριθμός.

20 Επαναλήψεις

N	ω(G)	MDsatur	First Fit
20	7,45	7,45	7,45
50	15,9	15,9	15,9
100	20,85	20,85	20,85
500	125,15	125,15	125,15
1000	226,5	226,5	226,5
2000	475,9	475,9	475,9
4000	1005,65	1005,65	1005,65

Πίνακας 5.13 : Αποτελέσματα γραφημάτων διαστημάτων για εκτέλεση 20 φορές

50 Επαναλήψεις

N	ω(G)	MDsatur	First Fit
20	6,86	6,86	6,86
50	13,7	13,7	13,7
100	17,13	17,13	17,13
500	123,4	123,4	123,4
1000	245,1	245,1	245,1
2000	373,83	373,83	373,83
4000	953,2	953,2	953,2

Πίνακας 5.14 : Αποτελέσματα γραφημάτων διαστημάτων για εκτέλεση 50 φορές

100 Επαναλήψεις

N	ω(G)	MDsatur	First Fit
20	7,2	7,2	7,2
50	13,27	13,27	13,27
100	27,61	27,61	27,61
500	117,48	117,48	117,48
1000	241,11	241,11	241,11
2000	500,04	500,04	500,04
4000	952,9	952,9	952,9

Πίνακας 5.15: Αποτελέσματα γραφημάτων διαστημάτων για εκτέλεση 100 φορές

1000 Επαναλήψεις

N	ω(G)	MDsatur	First Fit
20	7,1	7,1	7,1
50	13,4	13,4	13,4
100	24,3	24,3	24,3
500	117,65	117,65	117,65
1000	241	241	241
2000	486,3	486,3	486,3
4000	1003,7	1003,7	1003,7

Πίνακας 5.16: Αποτελέσματα γραφημάτων διαστημάτων για εκτέλεση 1000 φορές

Από τα αποτελέσματα που προκύπτουν παρατηρούμε πως και ο αλγόριθμος MDsatur χρωματίζει βέλτιστα σε αυτήν την περίπτωση. Αυτό συμβαίνει γιατί όπως έχει αναφερθεί, ο συγκεκριμένος αλγόριθμος προκύπτει από τον άπληστο αλγόριθμο Dsatur με δύο επιπλέον ιδιότητες οι οποίες εδώ δεν μας επηρεάζουν κατά τον χρωματισμό, αφού για να συνδεθούν δυο κόμβοι στο γράφημα πρέπει στο αντίστοιχο cotree να έχουν ελάχιστο κοινό προκάτοχο 1 (οι εσωτερικοί κόμβοι εναλλάσσονται σε 0 και 1 ξεκινώντας από την ρίζα που είναι 1).

Πιο συγκεκριμένα ενώ στον Dsatur που είναι άπληστος και χρωματίζει βάσει του βαθμού κορεσμού έχουμε το βέλτιστο χρώμα, τα πράγματα δεν αλλάζουν καθόλου στον MDsatur. Αυτό συμβαίνει καθώς για να χρωματιστεί ο επόμενος κόμβος, ο αλγόριθμος αυτός κινείται γύρω από τον πιο παλιά χρωματισμένο κόμβο που είναι ουσιαστικά ο ελάχιστος κοινός προκάτοχος των κόμβων που είναι προς χρωματισμό.

5.5 Συμπλήρωμα Μεταθετικών Γραφημάτων

Σε αυτή την παράγραφο θα ασχοληθούμε με τα συμπληρωματικά των μεταθετικών γραφημάτων. Από τις ιδιότητες που αναφέρθηκαν στο πρώτο κεφάλαιο των μεταθετικών γραφημάτων, γνωρίζουμε πως το συμπληρωματικό ενός μεταθετικού γραφήματος είναι επίσης μεταθετικό γράφημα [Golu80].

Όμοια και εδώ, η μέγιστη φθίνουσα υποακαλουθία δίνει την μέγιστη κλίκα των γραφημάτων.

Και πάλι από τις ιδιότητες των γραφημάτων [Golu80], ένα μη κατευθυνόμενο γράφημα G είναι μεταθετικό αν και μόνο αν τα G και $\overline{G}$ είναι comparability γραφήματα. Δηλαδή αφού το G γράφημα είναι μεταθετικό τότε το $\overline{G}$ είναι και μεταθετικό αλλά και comparability γράφημα. Τα comparability γραφήματα είναι τέλεια γραφήματα δηλαδή και σε αυτά ο χρωματικός αριθμός τους είναι ο αριθμός της κλίκας τους [Golu80].

Παρακάτω ακολουθούν τα αποτελέσματα των πειραμάτων για γραφήματα με 20, 50, 100, 500, 1000, 2000, 4000 κόμβους καθώς ο αντίστοιχος χρωματικός αριθμός:

20 Επαναλήψεις

N	ω(G)	MDsatur	First Fit
20	6,5	6,55	7,2
50	10,7	10,9	12,8
100	17,1	17,54	20,35
500	39,45	42,1	53
1000	57,4	63,09	79,34
2000	83,4	93,34	121,94
4000	118,65	134,8	180,05

Πίνακας 5.17 : Αποτελέσματα αλγορίθμων για εκτέλεση 20 φορές

50 Επαναλήψεις

N	ω(G)	MDsatur	First Fit
20	6,5	6,55	7,2

50	10,86	11,04	12,7
100	17,02	17,54	20,3
500	39,98	42,8	53,27
1000	58,11	63,77	80,82
2000	83,68	92,86	121,7
4000	117,6	133,7	179,8

Πίνακας 5.18: Αποτελέσματα αλγορίθμων για εκτέλεση 50 φορές

100 Επαναλήψεις	**N**	**ω(G)**	**MDsatur**	**First Fit**
	20	6,58	6,6	7,11
	50	11,04	11,21	12,67
	100	17,04	17,55	20,25
	500	40,02	43,18	53,48
	1000	57,7	63,32	80,43
	2000	83,7	92,73	121,34
	4000	118,3	134,2	180,1

Πίνακας 5.19: Αποτελέσματα αλγορίθμων για εκτέλεση 100 φορές

1000 Επαναλήψεις	**N**	**ω(G)**	**MDsatur**	**First Fit**
	20	6,48	6,51	7,05
	50	11,09	11,27	12,67
	100	16,67	17,157	19,87
	500	40,18	43,11	53,33
	1000	58,05	63,41	80,717
	2000	83,4	92,5	121,18
	4000	118,6	134,6	179,7

Πίνακας 5.20: Αποτελέσματα αλγορίθμων για εκτέλεση 1000 φορές

Παρατηρούμε αρχικά τις τιμές που δίνει ο χρωματικός αριθμός των συμπληρωματικών μεταθετικών γραφημάτων και τα αποτελέσματα που δίνει ο χρωματικός αριθμός στα μεταθετικά γραφήματα. Πιο συγκεκριμένα, το αποτέλεσμα που δίνει ο χρωματικός αριθμός για 20 κόμβους στα μεταθετικά γραφήματα για 1000 εκτελέσεις είναι 6,1 και 6,48 το αποτέλεσμα για 20 κόμβους και 1000

εκτελέσεις στα συμπληρωματικά μεταθετικά γραφήματα. Το αποτέλεσμα είναι αναμενόμενο και αυτό γιατί: έστω σε μια τυχαία ακολουθία πχ π = {4, 3, 6, 1, 5, 2} που τη διατρέχουμε από τα αριστερά προς δεξιά, βρίσκουμε πως η μέγιστη φθίνουσα ακολουθία είναι μήκους n. Διατρέχωντας την ακολουθία αντίστροφα, όπως συμβαίνει στα συμπληρωματικά γραφήματα, η πιθανότητα η μέγιστη φθίνουσα ακολουθία να είναι πάλι n είναι κοντά στο 1. Επομένως τα αποτελέσματα είναι αναμενόμενα όμοια τα αποτελέσματα στον MDsatur και στον First Fit. Το γεγονός πως τα αποτελέσματα δεν είναι ακριβώς τα ίδια οφείλεται στο ότι δεν εξετάζονται ακριβώς οι ίδιες τυχαίες μεταθέσεις στα μεταθετικά γραφήματα με τα συμπληρωματικά αυτών.

5.6 Συμπλήρωμα Γραφημάτων Διαστημάτων

Στο τελευταίο τμήμα της μελέτης θα ασχοληθούμε με τα συμπληρωματικά γραφήματα των γραφημάτων διαστημάτων. Η κατασκευή τους έχει γίνει ανάλογα με την κατασκευή των γραφημάτων διαστημάτων μόνο που όταν δυο διαστήματα καλύπτονται, ο αντίστοιχος πίνακα γειτνίασης έχει την τιμή 0 ενώ στην αντίθετη περίπτωση έχει 1. Ο υπολογισμός της μέγιστης κλίκας έγινε με τον υπολογισμό συνάρτησης ύψους όπως έχει περιγραφεί στο κεφάλαιο 3.

Από τις ιδιότητες των γραφημάτων διαστημάτων [Golu80] ένα μη κατευθυνόμενο γράφημα G είναι γράφημα διαστημάτων αν και μόνο αν το G είναι τριγωνικό και το $\overline{G}$ είναι comparability γράφημα. Αυτά τα γραφήματα ανήκουν στην κλάση των τέλειων γραφημάτων οπότε και ο αριθμός της μέγιστης κλίκας δίνει το καλύτερο αποτέλεσμα χρωματισμού, δηλαδή τον χρωματικό αριθμό των γραφημάτων.

Τα αποτελέσματα του αλγορίθμου MDsatur θα δούμε πως είναι ίδια με αυτά του χρωματικού αριθμού, ενώ τα αποτελέσματα του First Fit είναι πολύ κοντά σε αυτά του χρωματικού αριθμού

Ακολουθούν τα πειραματικά αποτελέσματα των αλγορίθμων για την εκτέλεση γραφημάτων για κόμβους 20, 50, 100, 500, 1000, 2000, 4000 και αυτά για 20, 50, 100 και 1000 φορές επαναλήψεις. Ακόμα, υπάρχει και η στήλη με τα αποτελέσματα του χρωματικού αριθμού.

20 Επαναλήψεις

N	ω(G)	MDsatur	First Fit
20	2,5	2,5	2,85
50	4,75	4,75	5,1
100	6,35	6,35	7,25
500	14,35	14,35	16,54
1000	21,35	21,35	24,25
2000	29,55	29,55	33,55
4000	43,4	43,4	49,04

Πίνακας 5.21 : Αποτελέσματα γραφημάτων διαστημάτων για εκτέλεση 20 φορές

50 Επαναλήψεις	N	ω(G)	MDsatur	First Fit
	20	2,74	2,74	2,9
	50	4,62	4,62	4,96
	100	6,28	6,28	6,8
	500	14,78	14,78	16,78
	1000	21,14	21,14	23,78
	2000	29,74	29,74	33,61
	4000	42,5	42,5	48,34

Πίνακας 5.22 : Αποτελέσματα γραφημάτων διαστημάτων για εκτέλεση 50 φορές

100 Επαναλήψεις	N	ω(G)	MDsatur	First Fit
	20	2,72	2,72	2,93
	50	4,38	4,38	4,81
	100	6,41	6,41	7,08
	500	15	15	16,89
	1000	21,03	21,03	23,89
	2000	29,66	29,66	33,83
	4000	42,87	42,87	48,6

Πίνακας 5.23: Αποτελέσματα γραφημάτων διαστημάτων για εκτέλεση 100 φορές

1000 Επαναλήψεις	N	ω(G)	MDsatur	First Fit
	20	2,78	2,78	2,97
	50	4,45	4,45	4,89
	100	6,34	6,34	7,064
	500	14,56	14,56	16,48
	1000	20,72	20,72	23,43
	2000	29,7	29,7	33,79
	4000	42,5	42,5	48,27

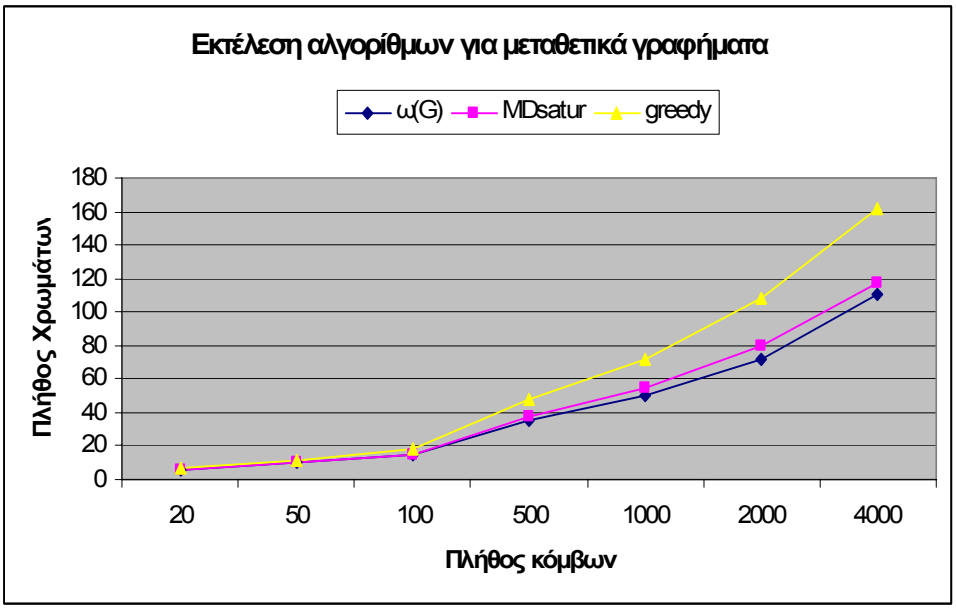
Πίνακας 5.24: Αποτελέσματα γραφημάτων διαστημάτων για εκτέλεση 1000 φορές

Τα αποτελέσματα με τα χρώματα που προκύπτουν είναι πολύ πιο μικρά λόγω της κατασκευής των γραφημάτων για αυτό και στους 20 κόμβους το αποτέλεσμα είναι 2,78 στον χρωματικό αριθμό. Η ύπαρξη καλύτερου αποτελέσματος στον

Mdsatur και όχι στον First Fit εξηγείται από το γεγονός ότι ο πρώτος αλγόριθμος, λόγω των δυο επιπλέον απαιτήσεων, μπορεί και χρωματίζει κόμβους κοντά στους πιο παλιά χρωματισμένους και όχι αυτόν με τον μέγιστο βαθμό κορεσμού όπως λειτουργεί ο Dsatur που είναι ένας άπληστος αλγόριθμος.

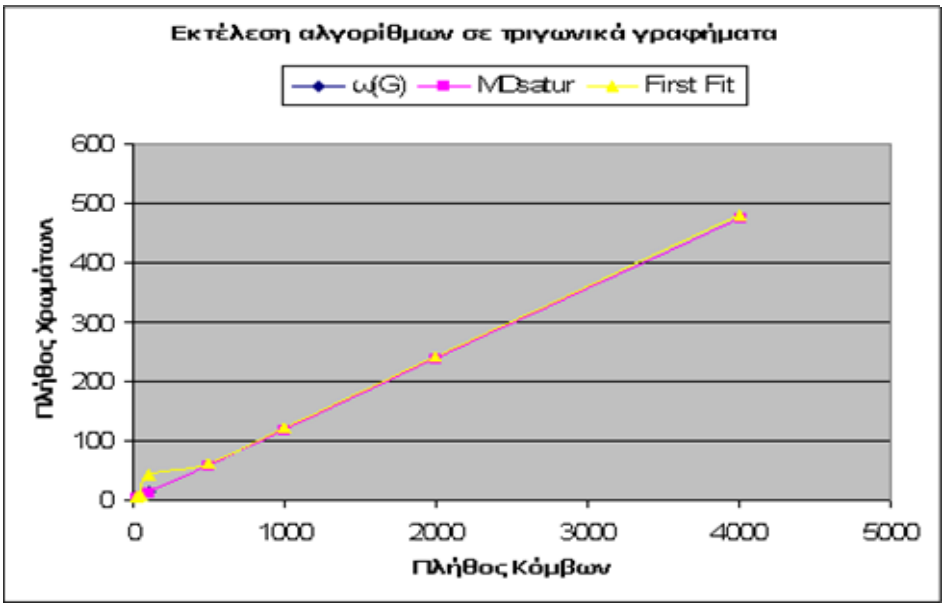
5.7 Γραφικές Παραστάσεις

Ακολουθούν οι γραφικές παραστάσεις των αποτελεσμάτων των αλγορίθμων MDsatur και First Fit σε όλες τις κλάσεις των γραφημάτων. Επίσης αναπαρίσταται και ο χρωματικός αριθμός κάθε κλάσης.



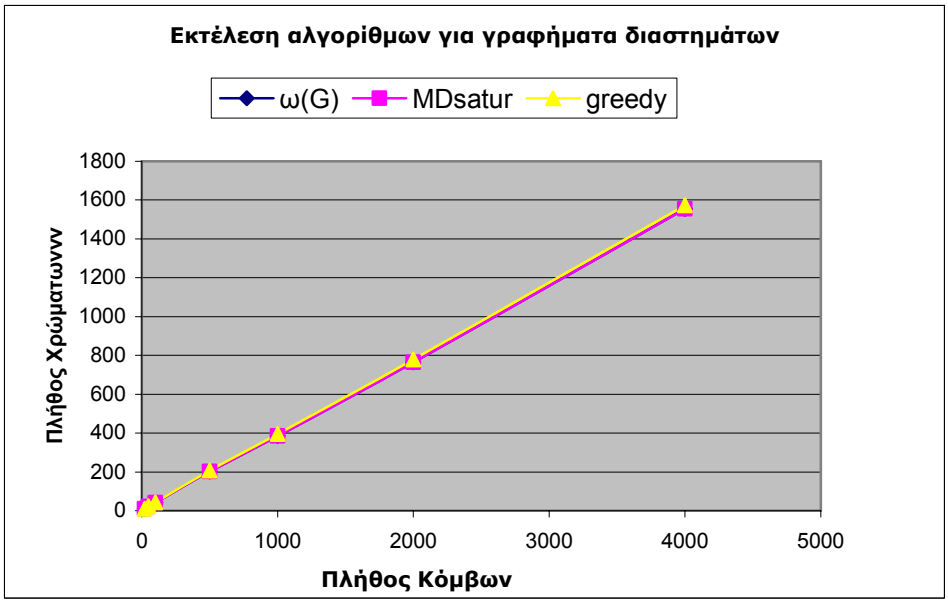
Σχήμα 5.5: Γραφική αναπαράσταση αποτελεσμάτων σε μεταθετικά γραφήματα

Στην κλάση των μεταθετικών γραφημάτων παρατηρούμε πως η καμπύλη του MDsatur είναι πολύ κοντά σε αυτή του χρωματικού αριθμού σε σχέση με την καμπύλη του αλγορίθμου First Fit.



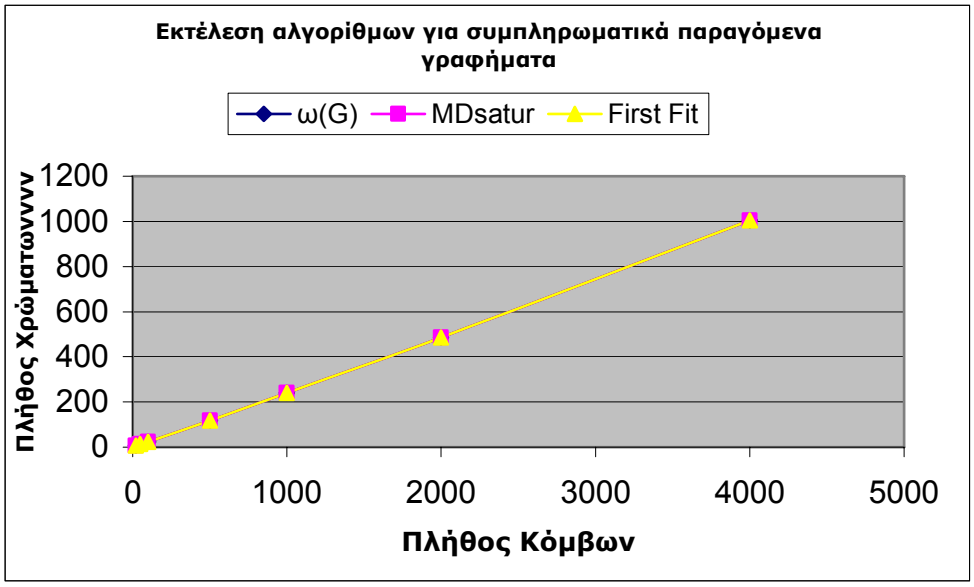
Σχήμα 5.6: Γραφική αναπαράσταση αποτελεσμάτων σε τριγωνικά γραφήματα

Στην κλάση των τριγωνικών γραφημάτων τα αποτελέσματα του αλγορίθμου MDsatur ταυτίζονται με την καμπύλη του χρωματικού αριθμού ενώ η καμπύλη με τα αποτελέσματα του First Fit αποκλίνει ελάχιστα.



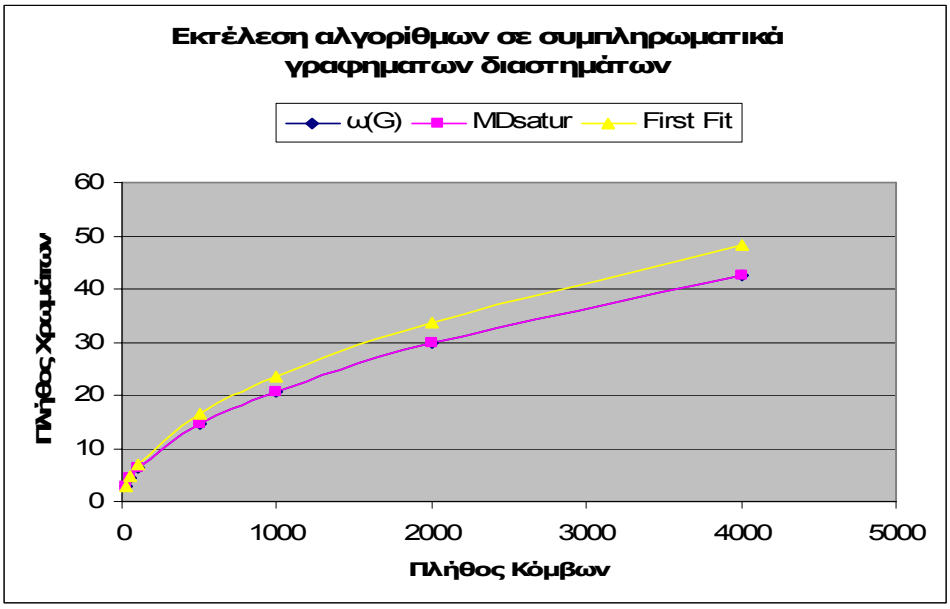
Σχήμα 5.7: Γραφική αναπαράσταση αποτελεσμάτων σε γραφήματα διαστημάτων

Στα γραφήματα διαστημάτων τα αποτελέσματα είναι παρόμοια με αυτά των τριγωνικών. Κάτι τέτοιο είναι αναμενόμενο, αφού τα γραφήματα διαστημάτων είναι υποκλάση των τριγωνικών.



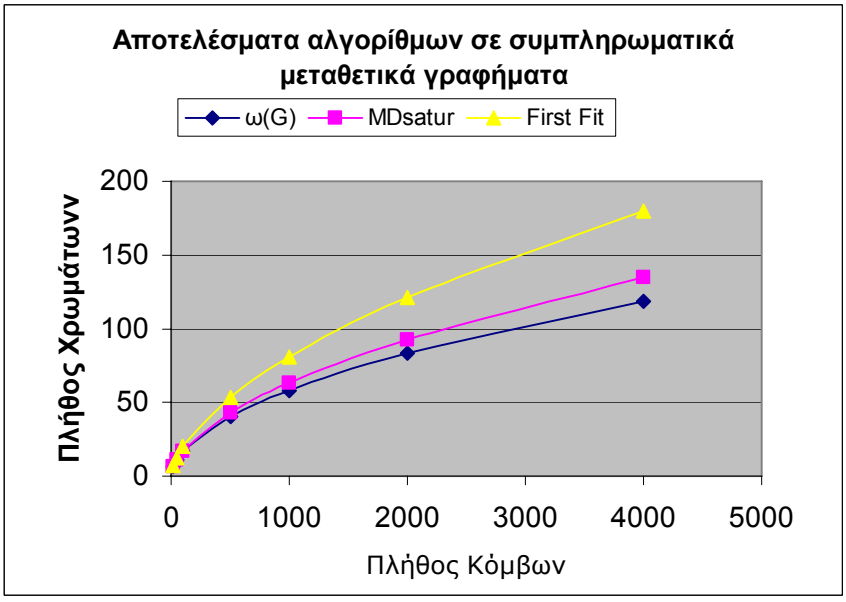
Σχήμα 5.8: Γραφική αναπαράσταση αποτελεσμάτων σε συμπληρωματικά παραγόμενα γραφήματα

Στα συμπληρωματικά μεταθετικά γραφήματα τα αποτελέσματα των αλγορίθμων First Fit και MDsatur ταυτίζονται με την καμπύλη του χρωματικού αριθμού.



Σχήμα 5.9: Γραφική αναπαράσταση αποτελεσμάτων σε συμπληρωματικά των γραφημάτων διαστημάτων

Στην κλάση των συμπληρωματικών των interval γραφημάτων τα αποτελέσματα ταυτίζονται σε όλους τους κόμβους για τον MDsatur και τη μέγιστη κλίκα ενώ και ο άπληστος αλγόριθμος δίνει χειρότερα αποτελέσματα.



Σχήμα 5.10: Γραφική αναπαράσταση αποτελεσμάτων σε συμπληρωματικά των μεταθετικών γραφήματα

Στην κλάση των συμπληρωματικών των μεταθετικών γραφημάτων τα αποτελέσματα του MDsatur είναι κοντά στον χρωματικό αριθμό ενώ στον First Fit διαφέρουν αρκετά όπως συμβαίνει και με τα μεταθετικά γραφήματα Γενικά μπορούμε να πούμε ότι για την κλάση των comparability γραφημάτων, για αυτά που εξετάσαμε, ο αλγόριθμος First Fit δεν δίνει βέλτιστα αποτελέσματα ενώ ο MDsatur δίνει αρκετά καλά αποτελέσματα.

Κεφάλαιο 6
Συμπεράσματα

Με βάση τα πειράματα της μελέτης και τις ιδιότητες των αλγορίθμων μελετήσαμε την συμπεριφορά των αλγορίθμων MDsatur και First Fit σε κάποιες κλάσεις γραφημάτων. Η κλάση των comparability γραφημάτων δεν μελετήθηκε ολόκληρη παρά μόνο αυτά που είναι συμπληρωματικά των μεταθετικών και συμπληρωματικά των γραφημάτων διαστημάτων. Επομένως με βάση τα αποτελέσματα που προέκυψαν, ο αλγόριθμος MDsatur δίνει αποτελέσματα πολύ κοντά στον χρωματικό αριθμό των γραφημάτων. Πιο συγκεκριμένα, στα τριγωνικά γραφήματα, τα γραφήματα διαστημάτων, τα συμπληρωματικά αυτών καθώς και τα συμπληρωματικά μεταθετικών γραφημάτων τα αποτελέσματα του MDsatur ταυτίζονται με τον χρωματικό αριθμό. Ωστόσο ο αλγόριθμος First Fit, δίνει αποτελέσματα που δεν είναι τόσο κοντά στο χρωματικό αριθμό εκτός από τα συμπληρωματικά μεταθετικών γραφημάτων που ταυτίζονται.

Δηλαδή ο MDsatur χρωματίζει πολύ καλά ένα γράφημα και σίγουρα καλύτερα από τον First Fit. Καθώς ο αλγόριθμος αυτός χρωματίζει τοπικά, όπως έχει αναφερθεί, τα αποτελέσματά του είναι ίδια με αυτά του βέλτιστου χρώματος.

Στα συμπληρωματικά μεταθετικών γραφημάτων έχει δειχθεί [BoCe] πως ένας άπληστος αλγόριθμος χρωματίζει βέλτιστα ένα τέτοιο γράφημα επομένως είναι πολύ λογικό το γεγονός πως και ο MDsatur δίνει αποτέλεσμα ίδιο με τον χρωματικό αριθμό αφού στηρίζεται στον άπληστο Dsatur αλγόριθμο με επιπλέον απαιτήσεις που όπως φαίνεται δεν μειώνουν την απόδοσή του.

Στα τριγωνικά γραφήματα και γραφήματα διαστημάτων η μέγιστη κλίκα υπολογίζεται με βάση το τέλειο σχήμα απαλοιφής που επίσης έχει χαρακτήρα τοπικό για την συμπλήρωσή του. Επομένως τα αποτελέσματα είναι αναμενόμενα.

Όσον αφορά τα συμπληρωματικά των γραφημάτων διαστημάτων ο MDsatur και πάλι δίνει βέλτιστα αποτελέσματα, ίδια με αυτά που δίνει η συνάρτηση ύψους.

Τέλος, στα μεταθετικά γραφήματα, η συμπεριφορά του MDsatur είναι καλύτερη από αυτή του First Fit.

Όσον αφορά την κλάση των comparability γραφημάτων γενικά δεν μπορούμε να δούμε κάποιο συγκεκριμένο αποτέλεσμα. Με βάση όμως την κλάση των συμπληρωματικών τόσο στα γραφήματα διαστημάτων όσο και στα μεταθετικά έχουμε μια ένδειξη πως ένας άπληστος αλγόριθμος δίνει αποτελέσματα που δεν είναι πολύ κοντά σε αυτά του χρωματικού αριθμού σε αντίθεση με τον MDsatur που τα αποτελέσματά του είναι πιο κοντά σε αυτά του χρωματικού αριθμού.

ΑΝΑΦΟΡΕΣ

[N02] S.D. Nikolopoulos, Coloring Permutation Graphs in Parallel, *Discrete Applied Mathematics*, Vol.120, pp.165 – 195, 2002.

[Golu80] M.C. Golumbic, Algorithmic Graph Theory and Perfect Graphs, Academic Press, Inc., New York, 1980

[NP05] S.D. Nikolopoulos and L.Palios , Efficient parallel recognition of cographs, , *Discrete Applied Mathematics* ,Vol.150, pp.182 – 215, 2005.

[ANS03] M.I. Andreou , S. E. Nikoletseas, and P. G. Spirakis, Algorithms and Experiments on Colouring Squares of Planar Graphs, *TR2003/03* Computer Technology Institute, Greece, 2003.

[GMR01] M.C. Golumbic, A.Mintz, Factoring and Recognition of Read-Once Functions using Cographs and Normality, *DAC 2001,* June 18-22, 2001, Las Vegas, Nevada, USA.

[NP00] S.D. Nikolopoulos and C.Papadopoulos , On the performance of the first-fit coloring algorithm on permutation graphs, *Inform. Proc. Lett.,* Vol 65, pp. 183 – 188, 1998

[BoCe] F. Bonomo and M. Cecowski, Between coloring and list-coloring: μ-coloring, *Electronic Notes in Discrete Mathematics,*Vol.19, pp. 117 – 123, 2005.

[Bre79] Daniel Brelaz, New Methods to Color the Vertices of a Graph , *Communications of the Assoc. of Comput. Machinery* , Vol.22, pp. 251 – 256, 1979

[HaPa04] M.Habib, C.Paul, A simple linear time algorithm for cograph recognition, *Discrete Applied Mathematics*, Vol. 145, n° 2, pp. 183-197 ,2005.

http://en.wikipedia.org/wiki/cographs

http://en.wikipedia.org/wiki/Perfect_graph

http://mathworld.wolfram.com/Permutation.html

ΠΑΡΑΡΤΗΜΑ

Μεταθετικά Γραφήματα

Permutation.c

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#define N 4001
#define M 4000
#define n N
#define TIMES 1000

int *produce_permutation();
int *graph_constraction(int *);
int fmax2(int *);


int main(){

int i,k,l,ma;
int *w,*g,*s,*per,j,maximum=0,gre=0;
long double sum_xrwmatikos=0,sum_klikas=0,sum_greedy=0;
float average_xrwmatikos=0,average_klikas=0,average_greedy=0;

  w=(int *)malloc(N*sizeof(int));
  s=(int *)malloc(M*sizeof(int));
  per=(int *)malloc(M*sizeof(int));
  g=malloc(N*N*sizeof(int));


        for(k=0;k<TIMES;k++){
           w=produce_permutation();
                gre=greedy(w);
                j=0;
                for(i=1;i<N;i++){

                per[j]=w[i];
                j++;
                }
                maximum=fmax2(per);
                g=graph_constraction(w);

                ma=mdsatur(g);
```

```
                printf("tipota:%d\n",ma);
                printf("kai h megisth klika einai:%d\n",maximum);
                printf("kai o greedy einai:%d\n",gre);

                sum_xrwmatikos=sum_xrwmatikos+ma;
                sum_klikas=sum_klikas+maximum;
                sum_greedy=sum_greedy+gre;

                sleep(2);
printf("_____________________________________%i_____________________\n",k);

                free(g);
                //free(per);
                //free(w);
        }


        average_xrwmatikos=(sum_xrwmatikos/TIMES);
        average_klikas=(sum_klikas/TIMES);
        average_greedy=(sum_greedy/TIMES);

printf("O mesos oros tou xrwmatikou arithmou einai: %f\n",average_xrwmatikos);
        printf("O mesos oros ths klikas einai: %f\n",average_klikas);
        printf("O mesos oros tou algorithmou greedy einai: %f\n",average_greedy);

}

 int *produce_permutation(){
  int temp,vtemp,i,*v;
    v=(int *)malloc(N*sizeof(int));
          for(i=1;i<N;i++)
            v[i] = i;

            for(i=2;i<N;i++)

             {
             temp = randint(1,i);
             vtemp = v[temp];
             v[temp] = v[i];
             v[i] = vtemp;
          }
      return v;
 }

 int randint(int x,int y)
 {
  int zz;
  srand((unsigned int)time((time_t *)NULL));
```

```
  zz = rand();
  if((zz>=x)&(zz<=y))
    return(zz);
  else
  {
    zz = ((zz%(y-x)) + x);
    return(zz);
  }
}


int greedy(int *p)
{
int temp,vtemp,i,*v,j,*per,*w,*ptr,max_num=0,k;
   v=(int *)malloc(N*sizeof(int));
   per=(int *)malloc(N*sizeof(int));
   w=(int *)malloc(N*sizeof(int));
   ptr=(int *)malloc(N*sizeof(int));

          for(i=1;i<N;i++)
            v[i] = i;

            for(i=2;i<N;i++)

             {
              temp = randint(1,i);
              vtemp = v[temp];
              v[temp] = v[i];
              v[i] = vtemp;
          }

          j=0;
          for(i=1;i<N;i++){
          per[j]=v[i];
          j++;
          }

          for(i=0;i<M;i++){
          ptr[i]=p[per[i]];
          }
          max_num=fmax2(ptr);
          return(max_num);

}


int *graph_constraction(int *p){
 int i,j,k,l; //g[N][N],
 int *g;
```

```
 g=(int *)malloc((N)*(N)*sizeof(int));
  for(i=0;i<N;i++)
   for(j=0;j<N;j++)
    *(g+i*(N)+j)=0;

     for(i=1;i<N;i++){
     for(j=i+1;j<N;j++){

      if(p[i]>p[j]){
       k=p[i];
          l=p[j];

          *(g+k*N+l)=1;
          *(g+l*N+k)=1;

       }
     }
   }
  return(g);
  }

int mdsatur(int *A)
{

       int c, cc, sum, ind, fin, max, ma,  COL[N], difcol[N][N], IS[N][2];
          int i, maxNEIG, indtmp, indNEIG, count, comNEIG,j;

          for(cc=1;cc<n+1;cc++)
          for(c=0;c<n+1;c++)
                    difcol[cc][c] = 0;

          /* COL[cc] = colour of vertex cc*/
          for(cc=1;cc<n+1;cc++)
          COL[cc] = 0;


          /*IS[cc] has the vertices of colour cc*/
          for(cc=1;cc<n+1;cc++)
          IS[cc][0] = 0;

          fin = 0;
          ind = 0;

          /*While there are uncoloured vertices do the following */
          while (fin < n+1)
          {
                    max = 0;
                    indtmp = 0;
                    indNEIG = 0;
```

```
maxNEIG = 0;
comNEIG =0;

/* find the degree of saturation of each
currently uncoloured  vertex */
for(cc=1;cc<n;cc++)
{
        if(COL[cc] == 0)
        {
        for(c=1; c<n+1; c++)
        {
                //if(A[cc][c]==1)
                        if((*(A+cc*N+c))==1)
                {
                        if(COL[c] != 0)
                        {
                        if(difcol[cc][COL[c]] ==0)
                        {
                                difcol[cc][COL[c]] = 1;
                                difcol[cc][0]++;
                        }
                        }
                };
        };


                        /* find the vertex that is 'closer'
                        to the last coloured vertex (ind)*/
                //if(A[ind][cc] == 1)
                if((*(A+ind*N+cc))==1)
                {
                        if(maxNEIG <= difcol[cc][0])
                        {
                                count= 0;
                                for(i = 1; i<=n; i++)
                         if((*(A+ind*N+i))==1 &&(*(A+cc*N+i))==1)
                                        count++;
                                if(count > comNEIG)
                                {
                                        comNEIG = count;
                                maxNEIG = difcol[cc][0];
                                indNEIG = cc;
                                }
                        }
                }

                if(max <= difcol[cc][0])
                {
                max = difcol[cc][0];
```

```
                                indtmp = cc;
                                }

                        }
                }


                /* Find the next vertex that will
                be coloured */
                if((maxNEIG == max) && (max != 0))
                        ind = indNEIG;
                else
                        ind = indtmp;


                /* colour the last vertex with the
                smallest allowable colour */

        for(c = 1;c<n+1; c++)
        {
                if(difcol[ind][c] == 0)
                {
                        COL[ind] = c;
                        c = n +1;
                }
                }
                fin++;
        }

        ma = 0;
        IS[1][0] = 0 ;

        for(c=1;c<=n;c++)
        {
        IS[COL[c]][0]++;
        if(ma < COL[c])
                ma = COL[c];
        }

        sum = 0;
        for(c=1;c<=ma;c++)
        {
                sum += IS[c][0];

        }

printf("\n\n The running algorithm is the ModDsatur");
printf("\n #colours = %d   #vertices = %d\n\n\n", ma, n);
return(ma);
```

```
}


int fmax2(int *p)
{
  int length = M;
  int *h,i,j,max = 0;
  h=(int *)malloc(M*sizeof(int));
  for(i=0;i<length;i++)
     h[i]=0;

  for(i=length-1;i>=0;i--)
  {
     max = 0;
     for(j=i;j<length;j++)
     {
        if(p[i]>p[j] && max<h[j])
           max = h[j];
     }
     h[i]=max+1;
  }
  return(max_num(h));
}



int max_num(int *a){
int max=0,i;

 max=a[0];

 for(i=0;i<M;i++){
          if(max<a[i])
           max=a[i];
}
return max;

}
```

File : help.h //θα χρησιμοποιηθεί παρακάτω σαν header file σε όλα τα υπολοιπα αρχεία//

```
#include <stdio.h>
#include <stdlib.h>
// Διάσταση του πίνακα που αρχίζει από το 0
#define SIZE 4000
// Διάσταση του πίνακα που αρχίζει από 1
#define BIG_SIZE (SIZE+1)
#define BIG_MATRIX_SIZE BIG_SIZE*BIG_SIZE
#define arrayBIG(matrix,i,j) *(matrix+i*BIG_SIZE+j)
#define arraySIZE(matrix,i,j) *(matrix+i*SIZE+j)
#define array(p,i) *(p+i)
 int randint(int x,int y);
 int fmax_reverse(int *p,int *g);
 int find_min_avail(int a[], int size, int max);
 int max_num(int *a);
int find_max_counter(int a[])
{
   int i,max=0,index=-1;
   for(i=0;i<SIZE;i++)
      if(a[i]>max)
      {
         max=a[i];
         index=i;
      }
   return(index);
}
// Elegxei an enas ari8mos brisketai mesa se ena array.
int is_in_set(int forcheck, int set[], int limit)
{
   int i;
   for(i=0; i<limit; i++){
      if(forcheck==set[i])
         return(1);
   }
   return(0);
}

void print_r(int a[],int l)
{
   int i;
   for(i=0;i<=l;i++)
      printf("%d ",a[i]);
   printf("\n");
}
```

```
void syn1(int *input_mat, int *result_mat)
{
   int i,j,current,max_counter,in_set=0;
         int *counters;
         counters = malloc(SIZE*sizeof(int));

   //To prwto stoixeio pou mpainei ston pinaka
         array(result_mat,0)=0;
         for (i=0; i<SIZE-1; i++)
   {
                  for (j=0; j<SIZE; j++)
                           *(counters+j)=0;

       // Elegxw ton ka8e kombo pou den einai mesa so result na dw me posous
geitoneyei
       for(j=0;j<SIZE;j++)
       {
         // Psaxnw kombo pou den einai mesa sto result.
                           if (is_in_set(j,result_mat,SIZE)==0)
         {
            int k;
            //elegxw me posous kombous apo aytous pou einai
            //mesa sto result_mat synoreyei o j
//             printf("j=%d is not in set\n",j);

                                    for(k=0;k<i+1;k++)
            {
                                             int myi;
                                             myi = array(result_mat,k);
              if(arraySIZE(input_mat,myi,j)==1)
                                             {
                                                      *(counters+j)+=1;
                                             }
            }
         }
       }
                  max_counter = find_max_counter(counters);
       if(max_counter !=-1)
                  {
                                             ++in_set;
                                             current = *(result_mat+in_set) = max_counter;
                  }
                  else
                  {
                           int a;
                           for(a=0;a<SIZE;a++)
```

```
                        {
                                // Psaxnw kombo pou den einai mesa sto result.
                                if (is_in_set(a,result_mat,SIZE)==0)
                                {
                                        ++in_set;
                                        current = *(result_mat+in_set) = a;
                                        break;
                                }
                        }
                }
  }
}

void invert(int *in, int *out)
{
   int i,j=0;
   for(i=0;i<SIZE;i++)
      *(out+i)=-1;
   for(i=0;i<SIZE;i++)
   {
      if(*(in+(SIZE-i-1))>=0)
                {
         *(out+j)=*(in+(SIZE-i-1));
                        j++;
                }
   }

}

int find_max_klik(int *ar,int *adj)
{
        int i,j,max=-1;
        for(i=0;i<SIZE;i++)
        {
                int counter=1;
                //printf("For %d Neighbours:\n",*(ar+i));
                for(j=i+1;j<SIZE;j++)
                {
                        int x=*(ar+i),y=*(ar+j);
                        //printf("Checking %d\n",y);

                        if(*(adj+x*SIZE+y)==1)
                        {
                                counter++;
                        }
                }
                if(counter>max)
                        max=counter;
   }
```

```
        return (max);
}

void makeme(int *input,int *out)
{
        int i,j;
        for(i=1;i<BIG_SIZE;i++)
                for(j=1;j<BIG_SIZE;j++)
                {
                        *(out+(i-1)*SIZE+(j-1))=*(input+i*BIG_SIZE+j);
                }

}

int *produce_permutation(){
  int temp,vtemp,i,*v;
    v=(int *)malloc(BIG_SIZE*sizeof(int));
        for(i=1;i<BIG_SIZE;i++)
          v[i] = i;

          for(i=2;i<BIG_SIZE;i++)

           {
           temp = randint(1,i);
           vtemp = v[temp];
           v[temp] = v[i];
           v[i] = vtemp;
        }
    return v;
 }

 int randint(int x,int y)
 {
   int zz;
   zz = rand();
   if((zz>=x)&(zz<=y))
     return(zz);
   else
   {
     zz = ((zz%(y-x)) + x);
     return(zz);
   }
 }

int greedy(int *g)
 {
 int i,*v,j,*per,max_num=0;
     per=(int *)malloc(BIG_SIZE*sizeof(int));
```

```
        for(i=1;i<=5000000;i++);


      v=produce_permutation();

         j=0;
         for(i=1;i<BIG_SIZE;i++){
                  per[j]=v[i];
                  j++;
         }
         max_num=fmax_reverse(per,g);
    return(max_num);

}



int fmax_reverse(int *p,int *g)//kanei to anapodo sto permutation
{
  int length = SIZE;
  int used_colors[SIZE];
  int neighb_num;
  int *h,i,j,k,max = 0,temp=0,min;
  h=(int *)malloc(SIZE*sizeof(int));
  for(i=0;i<length;i++)
     h[i]=1;
  for(i=0;i<length;i++)
  {
      neighb_num=0;
      for(k=0;k<length;k++)
         used_colors[k]=0;
      for(j=0;j<i;j++)
      {
         // Elegxos an einai geitonas
         if((*(g+(p[j]*BIG_SIZE)+p[i])==1)  /*&&  max<h[j]*/)
         {
//         printf("brhka geitona:%d\n",p[j]);
           //printf("min num:%d\n",min);
//         printf("xrhsimopoiei xrwma:%d\n",h[j]);
           used_colors[neighb_num]=h[j];
           // ayxanetai o ari8mos geitwnwn toy trexontos
           neighb_num++;
              }
      }
      // Eyresh toy mikroteroy dia8esimoy xrwmatos
      min = find_min_avail(used_colors,neighb_num,SIZE);
      h[i]= min;

  }
```

```
  return(max_num(h));
}

int max_num(int *a){
int max=0,i;

 max=a[0];

 for(i=0;i<SIZE;i++){
          if(max<a[i])
           max=a[i];
}
return max;

}

int find_min_avail(int a[], int size, int max)
{
   int i,j,found=0;
   for(i=1;i<=max;i++)
   {
      found=0;
      for(j=0;j<size;j++)
      {
         if(a[j]==i)
         {
            found = 1;
            break;
         }
      }
      if(found==0)
         return(i);
   }
         return(-500000);
}
```

triangular.c

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include "help.h"
#define N 4001
#define n 4001
#define TIMES  1000
#define DIV (800)


void reset_A(int*A)
{
	int i,j;
	for(i=0;i<N;i++){
		for(j=0;j<N;j++)
		{
			*(A+i*(N)+j)=0;
		}
	}
}

void produce_sequence(int seq[])
{
	int i;
	for(i=0;i<N-1;i++)
		seq[i]=-1;
	for(i=0;i<N-1;i++)
	{
		// 3ekinaei apo aso. Bazw i+1 gia na parei timh ish kai me N+1
		// to seq ta apouhkeyei stis 8eseis 0..N-1
		int pos=(int)(rand()%(N-1));
		while(seq[pos]!=-1)
		{
			pos=(pos+1)%(N-1);
		}
		seq[pos]=i;
	}
}


// pinakas, posoi komboi yparxun, poia seira koitaw, poion ari8mo psaxnw
int check_if_in(int*A, int limit, int row, int check)
{
	int b;
	int result = 0;
```

```
	for(b=0;b<limit;b++)
	{
		if(*(A+row*N+b)==check)
		{
			result = 1;
			break;
		}
		//printf("%d ",*(A+row*N+b));
	}
	return(result);
}

void mat_mult(int *A, int *B, int*res)
{
	int i,j,k;

	for(i=0;i<N;i++){
		for(j=0;j<N;j++){
		*(res+i*(N)+j)=0;
		}
	}

	for(i=0;i<N;i++)
	{
		for(j=0;j<N;j++)
		{
			for(k=0;k<N;k++)
			{
				*(res+i*N+j)=*(res+i*N+j) + (*(A+i*N+k) *
*(B+k*N+j));
			}
		}
	}
}
// sequence, 8esh apo thn opoia arxizw na psaxnw, ayto pou psaxnw
int check_right(int seq[], int pos, int comp)
{
	int i;
	for(i=pos+1; i<N-1; i++)
	{
	//	printf("Checking position %d to find %d \n",i,comp);
		if(seq[i]==comp)
			return (1);
	}
	return(0);
}

int* triangular(int*A)
```

```
{
	int i,j,k,a,b,c1,c2,c3,c4,sequence[N-1];
	int *all_geitones = (int *)malloc((N)*(N)*sizeof(int));
	int *all_num_geitones = (int *)malloc((N)*sizeof(int));
	int sum_all_num_geitones = 0;
	reset_A(A);
	produce_sequence(sequence);
	for(i=0;i<N-1;i++)
	{
		// Ariumos Geitonwn
		int num_geitones = (int)(rand()%((N-1-i)))/DIV;
		int *the_geitones = malloc(num_geitones*sizeof(int));
		// De to bazw na koitaw gia dipla. Mporei na balei 2 fores ton idio
ari8mo
		for(j=0;j<num_geitones;j++)
		{
			int geitonas =  i+1+(int)(rand()%(N-i-2));
			*(the_geitones+j)=sequence[geitonas];

	*(A+(1+sequence[geitonas])*N+(1+sequence[i]))=*(A+(1+sequence[i])*N+(
1+sequence[geitonas]))=1;
		}

		for(j=0;j<num_geitones;j++)
		   for(k=j+1;k<num_geitones;k++)
	{
				if(*(the_geitones+j)!=*(the_geitones+k))
				{

	*(A+(1+(*(the_geitones+j)))*N+(1+(*(the_geitones+k))))=*(A+(1+(*(the_ge
itones+k)))*N+(1+(*(the_geitones+j))))=1;

				}
		}
		// Dhmiourgia tou pinaka twn extra syndesewn
		*(all_num_geitones+i)= num_geitones;
		for(j=0;j<num_geitones;j++)
		{
			*(all_geitones+i*N+j) = *(the_geitones+j);
		}
		free(the_geitones);
	}

	for(a=0;a<N-1;a++)
		sum_all_num_geitones += *(all_num_geitones+a);

	//// H symplhrwsh twn  epomenwn
	for(c1=0;c1<N-1;c1++)
	{
```

```
                int *connecting = (int *)malloc((N)*sizeof(int));
                int index=0;
                //printf("Koitaw to %d. \n",sequence[c1]);
                for(c2=1;c2<N;c2++)
                {
                        if(*(A+(1+sequence[c1])*N+c2)==1 &&
check_right(sequence, c1, (c2-1)))
                        {
//                              printf("Koitaw to %d. Ayto sta de3ia exei to %d kai
einai 1 ston A\n",sequence[c1],c2-1);
                                *(connecting+index) = c2-1;
                                index++;
                        }
                }
                for(c2=0;c2<index;c2++)
                {
                        for(c3=0;c3<index;c3++)
                        {
                                if(*(connecting+c3)!=*(connecting+c2))
                                {

        *(A+(*(connecting+c2)+1)*N+(*(connecting+c3)+1))=*(A+(*(connecting+c
3)+1)*N+(*(connecting+c2)+1))=1;
//                                      printf("Kanw 1 to zeygari
(%d,%d)\n",*(connecting+c3),*(connecting+c2));
                                }
                        }
                }
                free(connecting);
        }


        free(all_geitones);
        free(all_num_geitones);

        return (A);
}

int mdsatur(int *A)
{
/////// η ίδια συνάρτηση όπως και στα permutatation//////
}


main()
{
        int i,j,ma=0,k,klika=0,gre=0;
        int *g;
        int *input,*res,*fin;
```

```
	floatsum_xrwmatikos=0,average_xrwmatikos=0,sum_klika=0,average_klika=
0,sum_greedy=0,average_greedy=0;
	input=(int *)malloc((SIZE)*(SIZE)*sizeof(int));
	res = malloc(SIZE*sizeof(int));
	fin = malloc(SIZE*sizeof(int));
	srand((unsigned int)time(NULL));

	for(k=0;k<TIMES;k++){

g=(int *)malloc((N)*(N)*sizeof(int));
	for(i=0;i<SIZE;i++){
		*(res+i)=0;
		*(fin+i)=0;
		*(input+i)=0;
		for(j=0;j<SIZE;j++)
		{
			*(input+i*SIZE+j)=0;

		}
	}

	g=triangular(g);
	ma=mdsatur(g);
	gre=greedy(g);
	makeme(g,input);
	syn1(input, res);
	invert(res,fin);

	klika=find_max_klik(fin,input);
	printf("mdSatur Result:%d ",ma);
	printf(" Best color is:%d ",klika);
	printf(" Greedy Result:%d\n",gre);
	sum_xrwmatikos=sum_xrwmatikos+ma;
	sum_klika=sum_klika+klika;
	sum_greedy=sum_greedy+gre;
	free(g);
	printf("_____%i_____
___\n",k);

	}
	average_xrwmatikos=(sum_xrwmatikos/TIMES);
	average_klika=(sum_klika/TIMES);
	average_greedy=(sum_greedy/TIMES);
	printf("O mesos oros tou xrwmatikou arithmou einai:
%f\n",average_xrwmatikos);
	printf("O mesos oros ths klikas                    einai: %f\n",average_klika);
	printf("O mesos oros tou al/mou greedy      einai: %f\n",average_greedy);
	printf("_____");
}
```

Intervals .c : Για help4000.h χρησιμοποιούμε το help.h

```
#include <stdio.h>
 #include <string.h>
 #include <math.h>
 #include <stdlib.h>
 #include <time.h>
 #include "help4000.h"
 #define N 4001
 #define n 4001
 #define TIMES 1000

typedef struct p{
        int a;
        int b;
}pair;

pair produce_random_pair()
{
        int temp;
        pair mypair;
        mypair.a=(int)(rand()%200);
        temp=1+(int)(rand()%(200-mypair.a));
        mypair.b=temp+mypair.a;
        return(mypair);
}

int check_pairs(pair one, pair two)
{
        if(one.a > two.b || one.b <two.a)
        {
//              printf("Not covered\n");
                return(0);
        }
//      printf("Covered\n");
        return(1);
}

int *adjastency_matrix(pair pairs[],int no_pairs)
{
      int i,j;
      int *g;
      g=(int *)malloc((N)*(N)*sizeof(int));

      for(i=1;i<N;i++)
         for(j=0;j<N;j++)
           *(g+i*(N)+j)=0;
```

```
        for(i=1;i<no_pairs;i++)
                for(j=i+1;j<no_pairs;j++)
                *(g+j*N+i)=*(g+i*N+j)= check_pairs(pairs[i], pairs[j]);
        return(g);
}
int mdsatur(int *A)
{
 /////// η ίδια συνάρτηση όπως και στα permutatation//////
}
main()
{
        int i,j,*g,ma=0,k,klika=0,gre=0;
        int *input,*res,*fin;
        pair mypair[N];
        float
sum_xrwmatikos=0,average_xrwmatikos=0,sum_klika=0,average_klika=0,sum_gree
dy=0,average_greedy=0;
        g=(int *)malloc((N)*(N)*sizeof(int));
        input=(int *)malloc((SIZE)*(SIZE)*sizeof(int));
        res = malloc(SIZE*sizeof(int));
        fin = malloc(SIZE*sizeof(int));
        srand((unsigned int)time(NULL));

        for(k=0;k<TIMES;k++){

        for(i=0;i<SIZE;i++){
                *(res+i)=0;
                *(fin+i)=0;
                *(input+i)=0;
                for(j=0;j<SIZE;j++)
                {
                        *(input+i*SIZE+j)=0;
                }
        }


   for(i=1;i<N;i++)
        {
                mypair[i]=produce_random_pair();
                //printf(":%d %d\n",mypair[i].a, mypair[i].b);
        }
        g=adjastency_matrix(mypair,N);
        ma=mdsatur(g);
        gre=greedy(g);
        // Extra Begin
         makeme(g,input);
         syn1(input, res);
         invert(res,fin);
```

```
            klika=find_max_klik(fin,input);
            printf("mdSatur Result:%d\n",ma);
           printf("\nBest color is:%d\n",klika);
           printf("\nGreedy Result:%d\n",gre);
           // Extra End

           sum_xrwmatikos=sum_xrwmatikos+ma;
           sum_klika=sum_klika+klika;
           sum_greedy=sum_greedy+gre;
           free(g);
           printf("_____%i_____
___\n",k);
           }
           average_xrwmatikos=(sum_xrwmatikos/TIMES);
           average_klika=(sum_klika/TIMES);
           average_greedy=(sum_greedy/TIMES);

           printf("O mesos oros tou xrwmatikou arithmou einai:
%f\n",average_xrwmatikos);
           printf("O mesos oros ths klikas                    einai: %f\n",average_klika);
           printf("O mesos oros tou al/mou greedy       einai: %f\n",average_greedy);

}
```

cographlib.h

```
struct node
{
	int value;
	int data;
	int no_children;
	struct node *parent;
	struct node *children[SIZE];
};

int find_tree_klik(struct node* root)
{
	int i;
	int sum_klik = 0;
	int max_ret = -1;
	for(i=0;i<root->no_children;i++)
	{

		int ret_klik;

		if(root->children[i]!=NULL)
		{
			ret_klik = find_tree_klik(root->children[i]);
		}
		else
		{
			ret_klik = 1;
		}
		if(ret_klik>max_ret)
				max_ret = ret_klik;
		sum_klik+= ret_klik;
	}
	if(root->no_children==0)
	{
		return(1);
	}
	if(root->data==0)
	{
		return(max_ret);
	}
	else
	{
		return(sum_klik);
	}
}
```

```
int fill_A(int *A, struct node* root, int descendants[])
{
	int i,j=0,num_all=0,num=-1;
	int c1,c2,c3,c4;
	int *temp =(int *)malloc((SIZE)*sizeof(int));
	int *temp_ar = (int *)malloc((SIZE)*(root->no_children)*sizeof(int));
	int *num_ar =(int *)malloc((root->no_children)*sizeof(int));
	for(i=0;i<root->no_children;i++)
	{
		*(num_ar+i)=0; //Arxikopoihsh
		if(root->children[i]!=NULL)
		{
			num=fill_A(A,root->children[i],descendants);

			*(num_ar+i)=num;

			if(root->data==1)
			{
				for(c1=0;c1<num;c1++)
				{
					*(temp_ar+i*SIZE+c1) = descendants[c1];
				}
			}
			for(j=0;j<num;j++)
			{
				*(temp+num_all) = descendants[j];
				num_all++;
			}
		}
	}

	if(root->data==1)
	{
		for(c1=0;c1<root->no_children;c1++)
		{
			for(c2=0;c2<root->no_children;c2++)
			{
				if(c1!=c2)
				for(c3=0;c3<*(num_ar+c1);c3++)
				{
					for(c4=0;c4<*(num_ar+c2);c4++)
					{

	*(A+(*(temp_ar+c1*SIZE+c3))*(SIZE+1)+(*(temp_ar+c2*SIZE+c4))) =
*(A+(*(temp_ar+c2*SIZE+c4))*(SIZE+1)+(*(temp_ar+c1*SIZE+c3))) = 1;
					}
				}
			}
```

```
                }
        }

        if(root->no_children==0)
        {
                *(temp+num_all)= root->value;
                num_all++;
        }
        for(i=0; i<num_all; i++)
        {
                descendants[i] = *(temp+i);
        }
        free(temp);
        free(temp_ar);
        free(num_ar);
        return num_all;
}


void set1_0(struct node* root, int tab_no)
{
        int i;
        tab_no++;
        root->data = tab_no%2;
        for(i=0;i<root->no_children;i++)
        {
                if(root->children[i]!=NULL)
                set1_0(root->children[i],tab_no);
                else return;
        }
}

void free_tree(struct node* root)
{
        int i;
        for(i=0;i<root->no_children;i++)
        {
                if(root->children[i]!=NULL)
                {
                        free_tree(root->children[i]);
                        free(root->children[i]);
                }
                else return;
        }
}

struct node* make_tree(struct node *root)
{
        int i;
```

```
	int lr=1; // o mikroteros tyxaios ari8mos
	int ur=1+(root->no_children)/2; // o megalyteros tyxaios ari8mos
	int remain=root->no_children;  // posa paidia den exw syndesei akoma

	struct node *temp = malloc(sizeof(struct node)); // proswrinos kombos. Sto telos ua ginei riza
	// Arxikopoihsh tou temp
	temp->data=-1;
	temp->no_children=0;
	temp->parent=NULL;
	temp->value=-1;

	while (remain>0)
	{
		int no_connect = lr + rand()%ur;
Connect=%d\n",lr,ur,no_connect);

		if(no_connect==1) // syndeetai me ton pappou katey8eian (edw to afhnw sydedemeno me th riza)
		{
			temp->children[temp->no_children] = root->children[root->no_children-remain];
			root->children[root->no_children-remain]->parent=temp;
			temp->no_children++;
		}
		else // 8a dhmioyrghsw patera me ton opoio 8a kanw th syndesh
		{
			int j=0;
			// Dhmioyrgw kombo. Aytos einai paidi tou temp (riza) kai pateras twn fyllwn

			temp->children[temp->no_children]=malloc(sizeof(struct node));
			temp->children[temp->no_children]->data=-10;
			temp->children[temp->no_children]->no_children=no_connect;
			temp->children[temp->no_children]->parent=temp;
			temp->children[temp->no_children]->value=55;

			// se ayto to for dhmioyrgw goneis kai kanw tis syndeseis
			for(i=root->no_children-remain;i<(root->no_children-remain)+no_connect;i++)
			{
				temp->children[temp->no_children]->children[j++]=root->children[i];
				root->children[i]->parent=temp->children[temp->no_children];

			}
```

```
                        temp->no_children++;
                }


                remain -= no_connect;
                ur = 1+remain/2;
        }
        root=temp;

        if(root->no_children>1)
                make_tree(root);
        else
                return root;
}

struct node* init_tree(struct node *leaves[],struct node *root)
{
        int i,j;
        root->data=-1;
        root->no_children=SIZE;
        root->parent=NULL;
        root->value=-1;
        // Arxikopoihsh twn paidiwn
        for(i=0;i<SIZE;i++)
        {
                root->children[i]=leaves[i];
                leaves[i]->value=i+1;
                leaves[i]->data=0;
                leaves[i]->no_children=0;
                leaves[i]->parent=root;
                for(j=0;j<SIZE;j++)
                {
                        leaves[i]->children[j]=NULL;
                }
        }

        return (root);
}
```

Cographs.c : Για το help.h χρησιμοποιούμε το αρχικό

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include "help.h"
#include "cographlib.h"
#define N 4001
#define n 4001
#define TIMES 1000
#define DIV (200)


void reset_A(int*A)
{
	int i,j;
	for(i=0;i<N;i++){
		for(j=0;j<N;j++)
		{
			*(A+i*(N)+j)=0;
		}
	}
}

void produce_sequence(int *seq)
{
	int i;
	for(i=0;i<N-1;i++)
		*(seq+i)=-1;
	for(i=0;i<N-1;i++)
	{
		// 3ekinaei apo aso. Bazw i+1 gia na parei timh ish kai me 7
		// to seq ta apouhkeyei stis 8eseis 0..N-1
		int pos=(int)(rand()%(N-1));
		while(*(seq+pos)!=-1)
		{
			pos=(pos+1)%(N-1);
		}
		*(seq+pos)=i;
	}
}


// pinakas, posoi komboi yparxun, poia seira koitaw, poion ari8mo psaxnw
int check_if_in(int*A, int limit, int row, int check)
{
	int b;
```

```
	int result = 0;

	for(b=0;b<limit;b++)
	{
		if(*(A+row*N+b)==check)
		{
			result = 1;
			break;
		}
	}
	return(result);
}


void mat_mult(int *A, int *B, int*res)
{
	int i,j,k;

	for(i=0;i<N;i++){
		for(j=0;j<N;j++){
		*(res+i*(N)+j)=0;
		}
	}

	for(i=0;i<N;i++)
	{
		for(j=0;j<N;j++)
		{
			for(k=0;k<N;k++)
			{
				*(res+i*N+j)=*(res+i*N+j) + (*(A+i*N+k) *
*(B+k*N+j));
			}
		}
	}
}

// sequence, 8esh apo thn opoia arxizw na psaxnw, ayto pou psaxnw
int check_right(int *seq, int pos, int comp)
{
	int i;
	for(i=pos+1; i<N-1; i++)
	{
		if(seq[i]==comp)
			return (1);
	}
	return(0);
}
```

```
int mdsatur(int *A)
{
/// Η ίδια συνάρτηση όπως στα permutation γραφήματα
}
main()
{
	int i,j,ma=0,k,klika=0,gre=0;
	int descendants[SIZE];
	int *g;
	int *input,*res,*fin;
	float
sum_xrwmatikos=0,average_xrwmatikos=0,sum_klika=0,average_klika=0,sum_gree
dy=0,average_greedy=0;
	struct node *leaves[SIZE],*root;
	input=(int *)malloc((SIZE)*(SIZE)*sizeof(int));
	res = malloc(SIZE*sizeof(int));
	fin = malloc(SIZE*sizeof(int));
	srand((unsigned int)time(NULL));

	for(k=0;k<TIMES;k++){
	g=(int *)malloc((N)*(N)*sizeof(int));
	for(i=0;i<SIZE;i++){

		*(res+i)=0;
		*(fin+i)=0;
		*(input+i)=0;

		for(j=0;j<SIZE;j++)
		{
			*(input+i*SIZE+j)=0;

		}
	}
	reset_A(g);
/***********************************/
	for(i=0;i<SIZE;i++)
	{
		leaves[i] = malloc(sizeof(struct node));
		descendants[i]=0;
	}
	root = malloc(sizeof(struct node));
	root = init_tree(leaves,root);
	root = make_tree(root);
	root = root->children[0];
	root->parent=NULL;
	set1_0(root,0);
	//print_tree(root,0);
	fill_A(g,root, descendants);
	//printf("================================\n");
```

```
        klika = find_tree_klik(root);
        free_tree(root);
        printf("\nKlik returned:%d\n",klika);
/*      for(i=0;i<n;i++)
        {
                for(j=0;j<n;j++)
                {
                        printf("%d ",*(A+i*n+j));
                }
                printf("\n");
        }
        */
/**********************************/


     ma=mdsatur(g);
        gre=greedy(g);
        // Extra Begin

        makeme(g,input);
        syn1(input, res);
        invert(res,fin);
        printf("mdSatur Result:%d ",ma);
        printf(" Best color is:%d ",klika);
        printf(" Greedy Result:%d\n",gre);
        // Extra End

        sum_xrwmatikos=sum_xrwmatikos+ma;
        sum_klika=sum_klika+klika;
        sum_greedy=sum_greedy+gre;

        free(g);
        printf("_____%i_____
___\n",k);
        }
        average_xrwmatikos=(sum_xrwmatikos/TIMES);
        average_klika=(sum_klika/TIMES);
        average_greedy=(sum_greedy/TIMES);
        printf("O mesos oros tou xrwmatikou arithmou einai:
%f\n",average_xrwmatikos);
        printf("O mesos oros ths klikas                 einai: %f\n",average_klika);
        printf("O mesos oros tou al/mou greedy      einai: %f\n",average_greedy);
        printf("_____");


}
```

Σαν help.h χρησιμοποιήθηκε η αρχική

Complement_perm.c

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include "help.h"
#define N 4001
#define M 4000
#define n N
#define TIMES 100

int *produce_permutation();
int *graph_constraction(int *);

void invert2(int *, int *);
int mdsatur(int *A);
int max_num(int *a);


int fmax2(int *p)
{
  int length = M;
  int *h,i,j,max = 0;
  h=(int *)malloc(M*sizeof(int));
  for(i=0;i<length;i++)
     h[i]=0;

  for(i=length-1;i>=0;i--)
  {
     max = 0;
     for(j=i;j<length;j++)
     {
        if(p[i]>p[j] && max<h[j])
           max = h[j];
     }
     h[i]=max+1;
  }
  return(max_num(h));
}


int main(){

int i,k,ma;
```

```
 int
*w,*g,*s,*per,color=0,max_com=0,q=0,j,maximum=0,gre=0,*g_com,*com,*per_co
m,gre_com=0;
 float
sum_xrwmatikos=0,sum_klikas=0,sum_greedy=0,s_c_klik=0,s_c_color=0,s_c_greed
y=0;
 float
average_xrwmatikos=0,average_klikas=0,average_greedy=0,a_c_klik=0,a_c_color=0
,a_c_greedy=0;


   w=(int *)malloc(N*sizeof(int));
   s=(int *)malloc(M*sizeof(int));
   per=(int *)malloc(M*sizeof(int));
   g=malloc(N*N*sizeof(int));
   g_com=malloc(N*N*sizeof(int));
   com=(int *)malloc(N*sizeof(int));
   per_com=(int *)malloc(M*sizeof(int));

         for(k=0;k<TIMES;k++){
                  for(i=0;i<N;i++){
                           *(w+i)=0;
                           *(com+i)=0;
                  }
                  w=produce_permutation();
                  invert2(w,com);
                  j=0;
                  for(i=1;i<N;i++){
                  per[j]=w[i];
                  per_com[j]=com[i];
                  j++;
                  }
                  max_com=fmax2(per_com);
                  g_com=graph_constraction(com);
                  color=mdsatur(g_com);
                  gre_com=greedy(g_com);
                  s_c_klik = s_c_klik+max_com;
                  s_c_color = s_c_color + color;
                  s_c_greedy= s_c_greedy + gre_com;
                  for(q=0;q<50000000;q++){}

         printf("_____%i_____
___\n",k);

                  free(g_com);
         }
         a_c_klik=( s_c_klik/TIMES);
         a_c_color=(s_c_color/TIMES);
         a_c_greedy=(s_c_greedy/TIMES);
```

```
        printf("\n");
        printf("O mesos oros klikas=%f \n tou MDsatur=%f\n kai greedy einai: %f\n",
        a_c_klik,a_c_color,a_c_greedy);

  }
void invert2(int *in, int *out)
{
   int i,j=1;
   for(i=1;i<N;i++)
      *(out+i)=-1;
   for(i=0;i<N-1;i++)
   {

        *(out+j)=*(in+(N-i-1));
                        j++;
   }
}

 int *graph_constraction(int *p){
  int i,j,k,l; //g[N][N],
  int *g;
  g=(int *)malloc((N)*(N)*sizeof(int));
   for(i=1;i<N;i++)
   for(i=0;i<N;i++)
    for(j=0;j<N;j++)
     *(g+i*(N)+j)=0;
    for(i=1;i<N;i++){
      for(j=i+1;j<N;j++){
       if(p[i]>p[j]){
        k=p[i];
          l=p[j];
          *(g+k*N+l)=1;
          *(g+l*N+k)=1;
      }
     }
   }
  return(g);
  }

int mdsatur(int *A)
{
//// η ίδια όπως στο permutation//////////
}
```

Σαν help.h χρησιμοποιήθηκε η αρχική

Complement_inter.c

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include "help.h"
#define N 4000
#define n 4000
#define TIMES 1000

typedef struct p{
        int a;
        int b;
}pair;

pair produce_random_pair()
{
        int temp;
        pair mypair;
        mypair.a=(int)(rand()%200);
        temp=1+(int)(rand()%(200-mypair.a));
        mypair.b=temp+mypair.a;
        return(mypair);
}

int check_pairs(pair one, pair two)
{
        if(one.a > two.b || one.b <two.a)
        {
        //covered
                return(1);
        }
       // not Covered
                return(0);
}

int *adjastency_matrix(pair pairs[],int no_pairs)
{
      int i,j;
      int *g;
      g=(int *)malloc((N)*(N)*sizeof(int));

      for(i=1;i<N;i++)
```

```
        for(j=0;j<N;j++)
          *(g+i*(N)+j)=0;

        for(i=1;i<no_pairs;i++)
                for(j=i+1;j<no_pairs;j++)
                *(g+i*N+j)= check_pairs(pairs[i], pairs[j]);
    for(i=1;i<N;i++)
    {
        for(j=1;j<N;j++)
              printf("%d ",*(g+i*N+j));
        printf("\n");
    }
    return(g);
}


int compl_klik(int *A)
{
        char c;
        int *columns;
        int i,j,x,y,zero_col_counter=0,times=0;
        columns = malloc(SIZE*sizeof(int));

        while(zero_col_counter!=SIZE)
        {
                times++;

                for(x=1;x<N;x++)
                {
                        for(y=1;y<N;j++)
                        {
                                printf("%d ",*(A+x*N+y));
                        }
                        printf("\n");
                }
                printf("Times=%d,zero cols=%d",times,zero_col_counter);
                zero_col_counter=0;
                // Elegxos sthlwn
                for(i=1;i<N;i++)
                {
                        int t_c=0; // posa mhdenika?
                        for(j=1;j<N;j++)
                        {
                                if(*(A+j*N+i)==0)
                                {
                                        t_c++;
                                }
                                else
                                        break;
```

```
                        }

                        if(t_c==SIZE)//olh h sthlh einai mhdenika
                        {
                                zero_col_counter++;
                                // Kanw th grammh i 0
                                for(j=1;j<N;j++)
                                {
                                        *(A+i*N+j)=0;
                                }
                        }
                }
        }
        return(times);
}



int mdsatur(int *A)
{

 }


main()
{
        int i,j,*g,ma=0,k;
        int *input,*res,*fin;
        pair mypair[N];
        float sum_xrwmatikos=0,average_xrwmatikos=0;
        g=(int *)malloc((N)*(N)*sizeof(int));
        input=(int *)malloc((SIZE)*(SIZE)*sizeof(int));

        res = malloc(SIZE*sizeof(int));
        fin = malloc(SIZE*sizeof(int));
        srand((unsigned int)time(NULL));

        for(k=0;k<TIMES;k++){

        for(i=0;i<SIZE;i++){
                *(res+i)=0;
                *(fin+i)=0;
                for(j=0;j<SIZE;j++)
                {
                        *(input+i*SIZE+j)=0;
                }
        }
   for(i=1;i<N;i++)
        {
```

```
                mypair[i]=produce_random_pair();
        }
        g=adjastency_matrix(mypair,N);
        compl_klik(g);
        ma=mdsatur(g);
        // Extra Begin
        makeme(g,input);
         gre=greedy(g);
        syn1(input, res);
        invert(res,fin);
        printf("\nBest color is:%d\n",find_max_klik(fin,input));
        printf("mdSatur Result:%d\n",ma);
        // Extra End
        sum_xrwmatikos=sum_xrwmatikos+ma;
        sum_klika=sum_klika+klika;
        sum_greedy=sum_greedy+gre;
        free(g);
        printf("_____%i_____
___\n",k);
                }
        average_xrwmatikos=(sum_xrwmatikos/TIMES);
        average_klika=(sum_klika/TIMES);
        average_greedy=(sum_greedy/TIMES);
        printf("O mesos oros tou xrwmatikou arithmou einai:
%f\n",average_xrwmatikos);
        printf("O mesos oros ths klikas                 einai: %f\n",average_klika);
        printf("O mesos oros tou al/mou greedy      einai: %f\n",average_greedy);
        printf("_____");
}
```