ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ
ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

# Μελέτη και Σύγκριση Αλγορίθμων Ελέγχου Τοπολογίας για Ασύρματα Δίκτυα Ad Hoc και Δίκτυα Αισθητήρων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μαρία Ν. Μαρκάκη

**Επιβλέπων :** Συμεών Παπαβασιλείου
Επίκουρος Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2007

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ
ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

# Μελέτη και Σύγκριση Αλγορίθμων Ελέγχου Τοπολογίας για Ασύρματα Δίκτυα Ad Hoc και Δίκτυα Αισθητήρων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μαρία Ν. Μαρκάκη

**Επιβλέπων :** Συμεών Παπαβασιλείου
Επίκουρος Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 16η Οκτωβρίου 2007.

| | | |
|---|---|---|
| ..... | ..... | ..... |
| Συμεών Παπαβασιλείου | Ιάκωβος Βενιέρης | Δήμητρα Κακλαμάνη |
| Επίκουρος Καθηγητής Ε.Μ.Π. | Καθηγητής Ε.Μ.Π. | Αναπληρώτρια Καθηγήτρια Ε.Μ.Π. |

Αθήνα, Οκτώβριος 2007

.....
Μαρία Ν. Μαρκάκη

Διπλωματούχος Ηλεκτολόγων Μηχανικών και Μηχανικών Ηλεκτρονικών Υπολογιστών
Εθνικού Μετσόβιου Πολυτεχνείου

Copyright © Μαρία Ν. Μαρκάκη, 2007
Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου.

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή κ. Συμεών Παπαβασιλείου για την δυνατότητα που μου έδωσε να ασχοληθώ με αυτή την διπλωματική εργασία. Επίσης θα ήθελα να ευχαριστήσω ιδιαίτερα τον υποψήφιο διδάκτορα Βασίλη Καρυώτη για την πολύτιμη βοήθειά του, χωρίς την οποία θα ήταν αδύνατο να ολοκληρωθεί αυτή η διπλωματική εργασία. Τέλος θα ήθελα να ευχαριστήσω όλα τα αγαπημένα πρόσωπα που με υπομονή μου συμπαραστάθηκαν και με τον τρόπο τους συνέβαλλαν καθοριστικά στην ολοκλήρωση της εργασίας.

# Περίληψη

Σκοπός της συγκεκριμένης διπλωματικής εργασίας ήταν η μελέτη και η σύγκριση αλγορίθμων ελέγχου τοπολογίας για ασύρματα δίκτυα ad hoc και δίκτυα αισθητήρων. Μελετήθηκαν και υλοποιήθηκαν σε προγραμματιστικό περιβάλλον έξι αλγόριθμοι που ως στόχο είχαν τη βελτίωση της επίδοσης χαρακτηριστικών του δικτύου, όπως ο χρόνος ζωής του, μέσω του ελέγχου τοπολογίας.

Ο έλεγχος τοπολογίας συνίσταται στην επιλογή του επιπέδου της ισχύος εκπομπής για κάθε κόμβο στο ασύρματο δίκτυο, ώστε να επιτευχθεί η ελάχιστη δυνατή κατανάλωση ενέργειας του συνολικού δικτύου και να διασφαλιστεί η βιωσιμότητα και η αποδοτική του λειτουργία.

Οι αλγόριθμοι που μελετήθηκαν βασίζονται σε δύο διαφορετικές προσεγγίσεις που έχουν αναπτυχθεί για το έλεγχο της μεταφορικής ικανότητας του δικτύου και τον παράλληλο περιορισμό της κατανάλωσης ενέργειας του ασύρματου τερματικού, οι οποίες είναι οι γωνίες κώνων και οι περιοχές σκυτάλης. Στους πέντε πρώτους αλγορίθμους χρησιμοποιούνται οι γωνίες κώνων, δηλαδή οι εγγενείς γωνίες μεταξύ των ασύρματων συνδέσεων, ως κριτήριο για ελάττωση της ακτίνας μετάδοσης και στον τελευταίο οι περιοχές σκυτάλης, μέσω των οποίων επιδιώκεται ο προσδιορισμός ενός υποσυνόλου γειτόνων στους οποίους είναι πιο οικονομικό να προωθηθεί διαδικτυακή κίνηση, χρησιμοποιώντας καθαρά γεωμετρικά στοιχεία και ιδιότητες της ασύρματης διάδοσης κυμάτων.

Η υλοποίηση των αλγορίθμων έγινε σε γλώσσα προγραμματισμού C++ με τη βοήθεια του πακέτου ανάπτυξης εφαρμογών Microsoft Visual C++ 6.0 και τα μεγέθη μελέτης και σύγκρισης που χρησιμοποιήθηκαν ήταν η ακτίνα μετάδοσης, ο βαθμός κόμβου, οι διασπορές τους καθώς και η συνεκτικότητα του υποκείμενου γράφου δικτύου. Η σύγκριση των αλγορίθμων πραγματοποιήθηκε μέσω επαναλαμβανόμενων προσομοιώσεων που εστίασαν στα παραπάνω δικτυακά μεγέθη και ανέδειξαν τη συμπεριφορά των αλγορίθμων κατά τη μεταβολή συγκεκριμένων παραμέτρων. Τα συμπεράσματα παρουσιάστηκαν σε συγκριτικές γραφικές παραστάσεις με τη βοήθεια του MATLAB 6.5.

# Λέξεις κλειδιά

Ασύρματα δίκτυα, ad hoc, δίκτυα αισθητήρων, έλεγχος τοπολογίας, γωνίες κώνων, περιοχές σκυτάλης, ισχύς εκπομπής, ακτίνα μετάδοσης, βαθμός κόμβου.

# Abstract

The scope of this thesis was to study and compare various topology control algorithms for wireless ad hoc networks and sensor networks. Six algorithms have been analyzed and implemented through simulations, aiming at improving of the performance of certain characteristics of the network, such us lifetime, through topology control.

Topology control consists of choosing the level of the transmit power for every node in the wireless network, so that the minimum possible power consumption in the entire network is accomplished and the viability and efficiency of the network is ensured.

The algorithms that have been studied are mainly based on two different approaches, namely cone angles and relay regions. Both approaches have been developed for the control of the network-transfer ability and the simultaneous limitation of the wireless terminals' power consumption. In the first five algorithms cone angles, i.e. the inborn angles between the wireless connections, are being used as a criterion for the reduction of the coverage radius. The last algorithm, which is based on relay regions, determines a certain subset of neighbors to which is economically preferable to forward network traffic.

The implementation of the algorithms was actualized in the C++ programming language with the assistance of the applications development package Microsoft Visual C++ 6.0. The quantities which have been studied and compared, were the coverage radius, the node degree, their variances and the connectivity of the underlying graph of the network. The comparison of the algorithms was accomplished through repetitive simulations, which focused to the above network parameters and demonstrated the algorithms' behaviour during the variation of certain parameters. The conclusions were presented in comparative graphs with the assistance of MATLAB 6.5.



# Keywords

Wireless networks, ad hoc, sensor networks, topology control, cone angles, relay regions, transmit power, coverage radius, node degree.

**Πίνακας περιεχομένων**

**Πίνακας Σχημάτων** ..... **x**
**Πίνακας Πινάκων** ..... **xiii**
**Κεφάλαιο 1 - Εισαγωγή** ..... **1**
**Κεφάλαιο 2 – Σχετικές Εργασίες** ..... **4**
2.1 Πρωτόκολλο Span ..... 4
2.1.1 Σχεδιασμός του Span ..... 4
2.1.2 Ανακοίνωση συντονιστή κόμβου ..... 5
2.1.3 Αποχώρηση συντονιστή κόμβου ..... 5
2.2 Έλεγχος τοπολογίας και ομαδοποίηση ..... 6
2.2.1 Πρωτόκολλο CLUSTERPOW ..... 6
2.2.2 Πρωτόκολλο Tunnelled CLUSTERPOW ..... 8
2.2.3 Πρωτόκολλο MINPOW ..... 8
2.3 Έλεγχος τοπολογίας ασύρματων δικτύων χρησιμοποιώντας επαναπροσδιορισμό της εκπεμπόμενης ενέργειας. ..... 9
2.3.1 Στατικά δίκτυα–κεντροποιημένοι αλγόριθμοι ..... 10
2.3.2 Ασύρματα δίκτυα-κατανεμημένοι ευριστικοί αλγόριθμοι ..... 12
2.3.2.1 Local Information No Topology (LINT) ..... 13
2.3.2.2 Local Information Link-State Topology (LILT) ..... 13
**Κεφάλαιο 3 – Περιγραφή αλγορίθμων** ..... **15**
3.1 Γωνίες κώνων (cone angles) ..... 16
3.1.1 Ορισμός ..... 16
3.1.2 Αλγόριθμος ελέγχου τοπολογίας I (Algorithm I) ..... 17
3.1.3 Αλγόριθμος ελέγχου τοπολογίας II (Algorithm II) ..... 18
3.1.4 Αλγόριθμος ελέγχου τοπολογίας III (Algorithm III) ..... 18
3.1.5 Αλγόριθμος ελέγχου τοπολογίας IV (Algorithm IV) ..... 19
3.1.6 Αλγόριθμος ελέγχου τοπολογίας V (Algorithm V) ..... 19
3.2 Περιοχές σκυτάλης(relay regions) ..... 21
3.2.1 Ορισμός ..... 21
3.2.2 Αλγόριθμος RR (Algorithm RR) ..... 22
**Κεφάλαιο 4 – Υλοποίηση αλγορίθμων** ..... **27**
4.1 Αλγόριθμος I ..... 28
4.2 Αλγόριθμος II ..... 30
4.3 Αλγόριθμος III ..... 30
4.4 Αλγόριθμος IV ..... 30
4.5 Αλγόριθμος V ..... 30
4.6 Αλγόριθμος RR ..... 32
**Κεφάλαιο 5 – Παρουσίαση αποτελεσμάτων** ..... **34**
5.1 Αλγόριθμος I ..... 34
5.1.1 Αυξανόμενη πυκνότητα ..... 34
5.1.2 Σταθερή πυκνότητα ..... 36
5.2 Αλγόριθμος II ..... 39
5.2.1 Αυξανόμενη πυκνότητα ..... 39
5.2.2 Σταθερή πυκνότητα ..... 41
5.3 Συγκριτικά για αλγορίθμους I και II ..... 42
5.3.1 Αυξανόμενη πυκνότητα ( $dr = 5$ ) ..... 43
5.3.2 Σταθερή πυκνότητα ( $dr = 5$ ) ..... 44
5.4 Αλγόριθμος III ..... 45
5.4.1 Αυξανόμενη πυκνότητα ..... 45

5.4.2 Σταθερή πυκνότητα....47
5.5 Αλγόριθμος IV....49
5.5.1 δ=150....49
5.5.1.1 Αυξανόμενη πυκνότητα....49
5.5.1.2 Σταθερή πυκνότητα....51
5.5.2 δ=250....53
5.5.2.1 Αυξανόμενη πυκνότητα....53
5.5.2.2 Σταθερή πυκνότητα....55
5.6 Συγκριτικά των IV με διαφορετικά δ ( $dr = 5$ )....57
5.6.1 Αυξανόμενη πυκνότητα....57
5.6.2 Σταθερή πυκνότητα....58
5.7 Συγκριτικά για αλγορίθμους III και IV....59
5.7.1 Αυξανόμενη πυκνότητα....59
5.7.2 Σταθερή πυκνότητα....60
5.8 Αλγόριθμος V....61
5.8.1 Φάση 2 με α=120° ( $q = 2$ )....61
5.8.1.1 Αυξανόμενη πυκνότητα....61
5.8.1.2 Σταθερή πυκνότητα....64
5.8.2 Φάση 2 με α=90° ( $q = 2$ )....65
5.8.2.1 Αυξανόμενη πυκνότητα....65
5.8.2.2 Σταθερή πυκνότητα....68
5.8.3 Φάση 1 με α=120° ( $q = 2, dr = 5$ )....69
5.8.3.1 Αυξανόμενη πυκνότητα....70
5.8.3.2 Σταθερή πυκνότητα....72
5.8.4 Φάση 1 με α=90° ( $q = 2, dr = 5$ )....74
5.8.4.1 Αυξανόμενη πυκνότητα....74
5.8.4.2 Σταθερή πυκνότητα....76
5.8.5 Φάση 2 με α=120° ( $q = 1, dr = 5$ )....77
5.8.5.1 Αυξανόμενη πυκνότητα....78
5.8.6 Φάση 2 με α=90° ( $q = 1, dr = 5$ )....80
5.8.6.1 Αυξανόμενη πυκνότητα....80
5.9 Αλγόριθμος RR ( $dr = 5$ )....82
5.9.1 Αυξανόμενη πυκνότητα....82
5.9.2 Σταθερή πυκνότητα....84
5.9.3 Αυξανόμενη πυκνότητα ( $q = 1$ )....86
5.10 Συγκριτικά RR-V (2-90) - V (2-120), q=2....88
5.10.1 Αυξανόμενη πυκνότητα....88
5.10.2 Σταθερή πυκνότητα....90
5.11 Συγκριτικά RR-V (2-90) – V (2-120), q=2, L=1500, N=100 (όπου L η διάσταση της γεωγραφικής περιοχής και N ο αριθμός κόμβων)....91
5.12 Γενικά συγκριτικά όλων των αλγορίθμων....92
5.12.1 Αυξανόμενη πυκνότητα....92
5.12.2 Σταθερή πυκνότητα....94
**Κεφάλαιο 6 - Συμπεράσματα....95**
**Παράρτημα....97**
**Βιβλιογραφία....166**

**Πίνακας Σχημάτων**

Σχήμα 1. Δρομολόγηση μέσω του CLUSTERPOW σε ένα τυπικό μη ομογενές δίκτυο ..... 6
Σχήμα 2. Τροποποίηση του πρωτοκόλλου CLUSTERPOW ..... 7
Σχήμα 3. Παράδειγμα δημιουργίας άπειρου βρόχου ..... 7
Σχήμα 4. Επίλυση του προβλήματος άπειρου βρόχου μέσου του πρωτοκόλλου Tunnelled CLUSTERPOW ..... 8
Σχήμα 5. Αλγόριθμος `CONNECT` και διαδικασία `perNodeMinimalize` ..... 11
Σχήμα 6. Παρουσίαση παράπλευρων ζεύξεων ..... 11
Σχήμα 7. Αλγόριθμος BICONN-AUGMENT ..... 12
Σχήμα 8. Χρησιμοποιούμενο μοντέλο τοπολογίας ..... 16
Σχήμα 9. Γωνίες κώνου κόμβου με 6 εξερχόμενες ζεύξεις ..... 17
Σχήμα 10. Παράδειγμα επιλογής ακτίνας μετάδοσης με βάση τον αλγόριθμο I ..... 18
Σχήμα 11. Αποτελέσματα εφαρμογής αλγορίθμου I και II ..... 18
Σχήμα 12. Κώνος γωνίας α ..... 19
Σχήμα 13. Τυπική περιοχή σκυτάλης ..... 21
Σχήμα 14. Περιοχή εγκλεισμού κόμβου *i* ..... 22
Σχήμα 15. Αλγόριθμος αναζήτησης και συνάρτηση `FlipAllStatesDownChain` ..... 25
Σχήμα 16. Βοηθητικό σχήμα για την υλοποίηση της γωνίας κώνου ..... 28

**Πίνακας Διαγραμμάτων**

Διάγραμμα 1. Αλγόριθμος I-Αυξανόμενη πυκνότητα-Μέση ακτίνα ..... 34
Διάγραμμα 2. Αλγόριθμος I-Αυξανόμενη πυκνότητα-Διασπορά ακτίνας ..... 35
Διάγραμμα 3. Αλγόριθμος I-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων ..... 35
Διάγραμμα 4. Αλγόριθμος I-Αυξανόμενη πυκνότητα-Διασπορά αριθμού γειτόνων ..... 36
Διάγραμμα 5. Αλγόριθμος I-Σταθερή πυκνότητα-Μέση ακτίνα ..... 37
Διάγραμμα 6. Αλγόριθμος I-Σταθερή πυκνότητα-Διασπορά ακτίνας ..... 37
Διάγραμμα 7. Αλγόριθμος I-Σταθερή πυκνότητα-Μέσος αριθμός γειτόνων ..... 38
Διάγραμμα 8. Αλγόριθμος I-Σταθερή πυκνότητα-Διασπορά αριθμού γειτόνων ..... 38
Διάγραμμα 9. Αλγόριθμος II-Αυξανόμενη πυκνότητα-Μέση ακτίνα ..... 39
Διάγραμμα 10. Αλγόριθμος II-Αυξανόμενη πυκνότητα-Διασπορά ακτίνας ..... 39
Διάγραμμα 11. Αλγόριθμος II-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων ..... 40
Διάγραμμα 12. Αλγόριθμος II-Αυξανόμενη πυκνότητα-Διασπορά αριθμού γειτόνων ..... 40
Διάγραμμα 13. Αλγόριθμος II-Σταθερή πυκνότητα-Μέση ακτίνα ..... 41
Διάγραμμα 14. Αλγόριθμος II-Σταθερή πυκνότητα-Διασπορά ακτίνας ..... 41
Διάγραμμα 15. Αλγόριθμος II-Σταθερή πυκνότητα-Μέσος αριθμός γειτόνων ..... 42
Διάγραμμα 16. Αλγόριθμος II-Σταθερή πυκνότητα-Διασπορά αριθμού γειτόνων ..... 42
Διάγραμμα 17. Συγκριτικό I και II-Αυξανόμενη πυκνότητα-Μέση ακτίνα ..... 43
Διάγραμμα 18. Συγκριτικό I και II-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων ..... 44
Διάγραμμα 19. Συγκριτικό I και II-Σταθερή πυκνότητα-Μέση ακτίνα ..... 44
Διάγραμμα 20. Συγκριτικό I και II-Σταθερή πυκνότητα-Μέσος αριθμός γειτόνων ..... 44
Διάγραμμα 21. Αλγόριθμος III-Αυξανόμενη πυκνότητα-Μέση ακτίνα ..... 45

Διάγραμμα 22. Αλγόριθμος ΙΙΙ-Αυξανόμενη πυκνότητα-Διασπορά ακτίνας.....45
Διάγραμμα 23. Αλγόριθμος ΙΙΙ-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων...46
Διάγραμμα 24. Αλγόριθμος ΙΙΙ-Αυξανόμενη πυκνότητα-Διασπορά αριθμού γειτόνων .....46
Διάγραμμα 25. Αλγόριθμος ΙΙΙ-Σταθερή πυκνότητα-Μέση ακτίνα .....47
Διάγραμμα 26. Αλγόριθμος ΙΙΙ-Σταθερή πυκνότητα-Διασπορά ακτίνας .....47
Διάγραμμα 27. Αλγόριθμος ΙΙΙ-Σταθερή πυκνότητα-Μέσος αριθμός γειτόνων .....48
Διάγραμμα 28. Αλγόριθμος ΙΙΙ-Σταθερή πυκνότητα-Διασπορά αριθμού γειτόνων....48
Διάγραμμα 29. Αλγόριθμος IV(150)-Αυξανόμενη πυκνότητα-Μέση ακτίνα.....49
Διάγραμμα 30. Αλγόριθμος IV(150)-Αυξανόμενη πυκνότητα-Διασπορά ακτίνας.....50
Διάγραμμα 31. Αλγόριθμος IV(150)-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων .....50
Διάγραμμα 32. Αλγόριθμος IV(150)-Αυξανόμενη πυκνότητα-Διασπορά αριθμού γειτόνων .....51
Διάγραμμα 33. Αλγόριθμος IV(150)-Σταθερή πυκνότητα-Μέση ακτίνα.....51
Διάγραμμα 34. Αλγόριθμος IV(150)-Σταθερή πυκνότητα-Διασπορά ακτίνας .....52
Διάγραμμα 35. Αλγόριθμος IV(150)-Σταθερή πυκνότητα-Μέσος αριθμός γειτόνων.52
Διάγραμμα 36. Αλγόριθμος IV(150)-Σταθερή πυκνότητα-Διασπορά αριθμού γειτόνων .....53
Διάγραμμα 37. Αλγόριθμος IV(250)-Αυξανόμενη πυκνότητα-Μέση ακτίνα.....53
Διάγραμμα 38. Αλγόριθμος IV(250)-Αυξανόμενη πυκνότητα-Διασπορά ακτίνας.....54
Διάγραμμα 39. Αλγόριθμος IV(250)-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων .....54
Διάγραμμα 40. Αλγόριθμος IV(250)-Αυξανόμενη πυκνότητα-Διασπορά αριθμού γειτόνων .....55
Διάγραμμα 41. Αλγόριθμος IV(250)-Σταθερή πυκνότητα-Μέση ακτίνα.....55
Διάγραμμα 42. Αλγόριθμος IV(250)-Σταθερή πυκνότητα-Διασπορά ακτίνας .....56
Διάγραμμα 43. Αλγόριθμος IV(250)-Σταθερή πυκνότητα-Μέσος αριθμός γειτόνων.56
Διάγραμμα 44. Αλγόριθμος IV(250)-Σταθερή πυκνότητα-Διασπορά αριθμού γειτόνων .....57
Διάγραμμα 45. Συγκριτικό IV(150) και IV(250)-Αυξανόμενη πυκνότητα-Μέση ακτίνα.....57
Διάγραμμα 46. Συγκριτικό IV(150) και IV(250)-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων .....58
Διάγραμμα 47. Συγκριτικό IV(150) και IV(250)-Σταθερή πυκνότητα-Μέση ακτίνα.58
Διάγραμμα 48. Συγκριτικό IV(150) και IV(250)-Σταθερή πυκνότητα-Μέσος αριθμός γειτόνων .....59
Διάγραμμα 49. Συγκριτικό ΙΙΙ και IV-Αυξανόμενη πυκνότητα-Μέση ακτίνα.....59
Διάγραμμα 50. Συγκριτικό ΙΙΙ και IV-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων .....60
Διάγραμμα 51. Συγκριτικό ΙΙΙ και IV-Σταθερή πυκνότητα-Μέση ακτίνα .....60
Διάγραμμα 52. Συγκριτικό ΙΙΙ και IV-Σταθερή πυκνότητα-Μέσος αριθμός γειτόνων61
Διάγραμμα 53. Αλγόριθμος V(2-120)-Αυξανόμενη πυκνότητα-Μέση ακτίνα.....62
Διάγραμμα 54. Αλγόριθμος V(2-120)-Αυξανόμενη πυκνότητα-Διασπορά ακτίνας...62
Διάγραμμα 55. Αλγόριθμος V(2-120)-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων .....63
Διάγραμμα 56. Αλγόριθμος V(2-120)-Αυξανόμενη πυκνότητα-Διασπορά αριθμού γειτόνων .....63
Διάγραμμα 57. Αλγόριθμος V(2-120)-Σταθερή πυκνότητα-Μέση ακτίνα.....64
Διάγραμμα 58. Αλγόριθμος V(2-120)-Σταθερή πυκνότητα-Διασπορά ακτίνας .....64

Διάγραμμα 59. Αλγόριθμος V(2-120)-Σταθερή πυκνότητα-Μέσος αριθμός γειτόνων ..... 65
Διάγραμμα 60. Αλγόριθμος V(2-120)-Σταθερή πυκνότητα-Διασπορά αριθμού γειτόνων ..... 65
Διάγραμμα 61. Αλγόριθμος V(2-90)-Αυξανόμενη πυκνότητα-Μέση ακτίνα ..... 66
Διάγραμμα 62. Αλγόριθμος V(2-90)-Αυξανόμενη πυκνότητα-Διασπορά ακτίνας ..... 66
Διάγραμμα 63. Αλγόριθμος V(2-90)-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων ..... 67
Διάγραμμα 64. Αλγόριθμος V(2-90)-Αυξανόμενη πυκνότητα-Διασπορά αριθμού γειτόνων ..... 67
Διάγραμμα 65. Αλγόριθμος V(2-90)-Σταθερή πυκνότητα-Μέση ακτίνα ..... 68
Διάγραμμα 66. Αλγόριθμος V(2-90)-Σταθερή πυκνότητα-Διασπορά ακτίνας ..... 68
Διάγραμμα 67. Αλγόριθμος V(2-90)-Σταθερή πυκνότητα-Μέσος αριθμός γειτόνων. 69
Διάγραμμα 68. Αλγόριθμος V(2-90)-Σταθερή πυκνότητα-Διασπορά αριθμού γειτόνων ..... 69
Διάγραμμα 69. Αλγόριθμος V(1-120)-Αυξανόμενη πυκνότητα-Μέση ακτίνα ..... 70
Διάγραμμα 70. Αλγόριθμος V(1-120)-Αυξανόμενη πυκνότητα-Διασπορά ακτίνας ... 70
Διάγραμμα 71. Αλγόριθμος V(1-120)-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων ..... 71
Διάγραμμα 72. Αλγόριθμος V(1-120)-Αυξανόμενη πυκνότητα-Διασπορά αριθμού γειτόνων ..... 71
Διάγραμμα 73. Αλγόριθμος V(1-120)-Σταθερή πυκνότητα-Μέση ακτίνα ..... 72
Διάγραμμα 74. Αλγόριθμος V(1-120)-Σταθερή πυκνότητα-Διασπορά ακτίνας ..... 72
Διάγραμμα 75. Αλγόριθμος V(1-120)-Σταθερή πυκνότητα-Μέσος αριθμός γειτόνων ..... 73
Διάγραμμα 76. Αλγόριθμος V(1-120)-Σταθερή πυκνότητα-Διασπορά αριθμού γειτόνων ..... 73
Διάγραμμα 77. Αλγόριθμος V(1-90)-Αυξανόμενη πυκνότητα-Μέση ακτίνα ..... 74
Διάγραμμα 78. Αλγόριθμος V(1-90)-Αυξανόμενη πυκνότητα-Διασπορά ακτίνας ..... 74
Διάγραμμα 79. Αλγόριθμος V(1-90)-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων ..... 75
Διάγραμμα 80. Αλγόριθμος V(1-90)-Αυξανόμενη πυκνότητα-Διασπορά αριθμού γειτόνων ..... 75
Διάγραμμα 81. Αλγόριθμος V(1-90)-Σταθερή πυκνότητα-Μέση ακτίνα ..... 76
Διάγραμμα 82. Αλγόριθμος V(1-90)-Σταθερή πυκνότητα-Διασπορά ακτίνας ..... 76
Διάγραμμα 83. Αλγόριθμος V(1-90)-Σταθερή πυκνότητα-Μέσος αριθμός γειτόνων. 77
Διάγραμμα 84. Αλγόριθμος V(1-90)-Σταθερή πυκνότητα-Διασπορά αριθμού γειτόνων ..... 77
Διάγραμμα 85. Αλγόριθμος V(2-120)-q=1-Αυξανόμενη πυκνότητα-Μέση ακτίνα ... 78
Διάγραμμα 86. Αλγόριθμος V(2-120)-q=1-Αυξανόμενη πυκνότητα-Διασπορά ακτίνας ..... 78
Διάγραμμα 87. Αλγόριθμος V(2-120)-q=1-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων ..... 79
Διάγραμμα 88. Αλγόριθμος V(2-120)-q=1-Αυξανόμενη πυκνότητα-Διασπορά αριθμού γειτόνων ..... 79
Διάγραμμα 89. Αλγόριθμος V(2-90)-q=1-Αυξανόμενη πυκνότητα-Μέση ακτίνα ..... 80
Διάγραμμα 90. Αλγόριθμος V(2-90)-q=1-Αυξανόμενη πυκνότητα-Διασπορά ακτίνας ..... 80
Διάγραμμα 91. Αλγόριθμος V(2-90)-q=1-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων ..... 81

Διάγραμμα 92. Αλγόριθμος V(2-90)-q=1-Αυξανόμενη πυκνότητα-Διασπορά αριθμού γειτόνων ..... 81
Διάγραμμα 93. Αλγόριθμος RR-Αυξανόμενη πυκνότητα-Μέση ακτίνα ..... 82
Διάγραμμα 94. Αλγόριθμος RR-Αυξανόμενη πυκνότητα-Διασπορά ακτίνας ..... 82
Διάγραμμα 95. Αλγόριθμος RR-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων ..... 83
Διάγραμμα 96. Αλγόριθμος RR-Αυξανόμενη πυκνότητα-Διασπορά αριθμού γειτόνων ..... 83
Διάγραμμα 97. Αλγόριθμος RR-Σταθερή πυκνότητα-Μέση ακτίνα ..... 84
Διάγραμμα 98. Αλγόριθμος RR-Σταθερή πυκνότητα-Διασπορά ακτίνας ..... 84
Διάγραμμα 99. Αλγόριθμος RR-Σταθερή πυκνότητα-Μέσος αριθμός γειτόνων ..... 85
Διάγραμμα 100. Αλγόριθμος RR-Αυξανόμενη πυκνότητα-Διασπορά αριθμού γειτόνων ..... 85
Διάγραμμα 101. Αλγόριθμος RR(q=1)-Αυξανόμενη πυκνότητα-Μέση ακτίνα ..... 86
Διάγραμμα 102. Αλγόριθμος RR(q=1)-Σταθερή πυκνότητα-Διασπορά ακτίνας ..... 86
Διάγραμμα 103. Αλγόριθμος RR(q=1)-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων ..... 87
Διάγραμμα 104. Αλγόριθμος RR(q=1)-Αυξανόμενη πυκνότητα-Διασπορά αριθμού γειτόνων ..... 87
Διάγραμμα 105. Συγκριτικά V(2-120)-V(2-90)-6,q=2-Αυξανόμενη πυκνότητα-Μέση ακτίνα ..... 88
Διάγραμμα 106. Συγκριτικά V(2-120)-V(2-90)-6,q=2-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων ..... 88
Διάγραμμα 107. Συγκριτικά V(2-120)-V(2-90)-6,q=2-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων-Μόνο τελικά ..... 89
Διάγραμμα 108. Συγκριτικά V(2-120)-V(2-90)-6,q=2-Σταθερή πυκνότητα-Μέση ακτίνα ..... 90
Διάγραμμα 1Διάγραμμα 109. Συγκριτικά V(2-120)-V(2-90)-6,q=2-Σταθερή πυκνότητα-Μέσος αριθμός γειτόνων ..... 90
Διάγραμμα 110. Συγκριτικά V(2-120)-V(2-90)-6,q=2-Σταθερή πυκνότητα-Μέσος αριθμός γειτόνων-Μόνο τελικά ..... 91
Διάγραμμα 111. Συγκριτικά όλων των αλγορίθμων-Αυξανόμενη πυκνότητα-Μέση ακτίνα ..... 92
Διάγραμμα 112. Συγκριτικά όλων των αλγορίθμων-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων ..... 93
Διάγραμμα 113. Συγκριτικά όλων των αλγορίθμων-Σταθερή πυκνότητα-Μέση ακτίνα ..... 94
Διάγραμμα 114. Συγκριτικά όλων των αλγορίθμων-Σταθερή πυκνότητα-Μέσος αριθμός γειτόνων ..... 94

**Πίνακας Πινάκων**

Πίνακας 1. Σύγκριση μέσου αριθμού γειτόνων αλγορίθμων V(2-90),V(2-120) και RR ..... 91

# Κεφάλαιο 1 - Εισαγωγή

Ένα ad hoc δίκτυο ή δίκτυο αισθητήρων αποτελείται από ένα σύνολο ασύρματων κόμβων, οι οποίοι είναι διασκορπισμένοι σε μία πεπερασμένη γεωγραφική περιοχή. Κάθε κόμβος–αισθητήρας, εκτός της ασύρματης δυνατότητας επικοινωνίας, διαθέτει κάποιο επίπεδο υπολογιστικής ισχύος για επεξεργασία σήματος και την εκτέλεση μιας πληθώρας διεργασιών. Κάποια παραδείγματα ασύρματων ad hoc δικτύων ή αισθητήρων είναι τα ακόλουθα:

- Στρατιωτικά δίκτυα αισθητήρων για να ερευνώνται, να εντοπίζονται και να αποκτώνται όσο το δυνατόν περισσότερες πληροφορίες σχετικά με εχθρικές κινήσεις, εκρήξεις και άλλα φαινόμενα ενδιαφέροντος.
- Δίκτυα αισθητήρων για να ερευνώνται, να εντοπίζονται και να χαρακτηρίζονται χημικές, βιολογικές, ραδιολογικές, πυρηνικές, εκρηκτικές επιθέσεις και αντίστοιχα υλικά.
- Δίκτυα αισθητήρων για να ερευνώνται, να εντοπίζονται και να παρακολουθούνται πεδιάδες, δάση, ωκεανοί, κλπ.
- Ασύρματα δίκτυα αισθητήρων κίνησης για να παρακολουθείται η κυκλοφορία των οχημάτων σε εθνικές οδούς και σημεία συμφόρησης εντός των πόλεων.
- Ασύρματα δίκτυα αισθητήρων παρακολούθησης για την παροχή ασφάλειας σε εμπορικά κέντρα, χώρους στάθμευσης και άλλες εγκαταστάσεις.
- Ασύρματα δίκτυα αισθητήρων που χρησιμοποιούνται σε θέσεις στάθμευσης για να καθοριστούν οι ελεύθερες και δεσμευμένες θέσεις.
- Ad hoc δίκτυα που χρησιμοποιούνται για την παροχή πληροφοριών σε τουρίστες σε μουσεία, κλπ.
- Ad hoc δίκτυα για την εξυπηρέτηση των συνέδρων σε συνεδριακά κέντρα.

Η παραπάνω λίστα υποδηλώνει ότι τα ασύρματα ad hoc δίκτυα και δίκτυα αισθητήρων προσφέρουν συγκεκριμένες δυνατότητες σε ιδιωτικές αλλά και στρατιωτικές εφαρμογές.

Οι βασικοί στόχοι των δικτύων αυτού του τύπου εξαρτώνται από την εκάστοτε εφαρμογή, με πιο χαρακτηριστικούς τους ακόλουθους:

- Καθορισμός της τιμής συγκεκριμένων παραμέτρων σε δεδομένη τοποθεσία: σε ένα περιβαλλοντικό δίκτυο θα ήταν χρήσιμο να ήταν γνωστά τα επίπεδα ηλιοφάνειας, η θερμοκρασία του περιβάλλοντος, η ατμοσφαιρική πίεση, και η υγρασία σε έναν αριθμό τοποθεσιών. Το συγκεκριμένο παράδειγμα δείχνει πως ένας δεδομένος αισθητήρας-κόμβος μπορεί να συνδέεται με πολλά διαφορετικά είδη αισθητήρων, καθένας με διαφορετικό ρυθμό συλλογής δεδομένων και διαφορετικό εύρος επιτρεπόμενων τιμών.
- Εντοπισμός των περιστατικών ενδιαφέροντος και εκτίμηση παραμέτρων: στο ασύρματο δίκτυο αισθητήρων κίνησης θα ήταν επιθυμητό να εντοπιστεί ένα όχημα κατά την κίνησή του και να εκτιμηθεί η ταχύτητά και η κατεύθυνση του.
- Κατάταξη εντοπισμένου αντικειμένου: εάν για παράδειγμα το όχημα στο προηγούμενο παράδειγμα είναι αυτοκίνητο, φορτηγό, λεωφορείο, κλπ.

- Εντοπισμός αντικειμένου: σε στρατιωτικό δίκτυο εντοπισμός ενός τεθωρακισμένου καθώς κινείται σε δεδομένη γεωγραφική περιοχή.

Για τη σωστή λειτουργία πολλών ασύρματων ad hoc δικτύων και δικτύων αισθητήρων απαιτείται μεγάλος αριθμός από κόμβους (συνήθως στάσιμους). Εκτός από την διάταξη ασύρματων κόμβων στην επιφάνεια ωκεανών ή τη χρήση ασύρματων αισθητήρων ρομπότ σε στρατιωτικές εφαρμογές, οι περισσότεροι κόμβοι σε ένα έξυπνο δίκτυο αισθητήρων είναι σταθεροί. Δίκτυα 10.000 ή και 100.000 κόμβων μπορούν να γίνουν στο μέλλον πραγματικότητα και επομένως η δυνατότητα κλιμάκωσης του δικτύου είναι πολύ σημαντική.

Τα ασύρματα ad hoc δίκτυα και δίκτυα αισθητήρων χαρακτηρίζονται από τον αυτόνομο τρόπο λειτουργίας του κάθε κόμβου και την έλλειψη σταθερής υποδομής δικτύου κορμού. Ο αυτόνομος χαρακτήρας και ο χαμηλού–κόστους προσανατολισμός τέτοιων δικτύων, επιβάλλουν σημαντικούς περιορισμούς στην κατανάλωση ενέργειας των τερματικών σταθμών, περιορίζοντας αντιστοίχως τις λειτουργικές ικανότητες του κάθε ασύρματου κόμβου.

Δεδομένου του μεγάλου αριθμού των κόμβων και τη μελλοντική τοποθέτησή τους σε εχθρικά περιβάλλοντα απαραίτητο χαρακτηριστικό του δικτύου πρέπει να είναι η δυνατότητα αυτό-οργάνωσης. Επιπλέον κάποιοι κόμβοι, είτε λόγω έλλειψης ενέργειας, είτε λόγω τεχνικών προβλημάτων τίθενται εκτός λειτουργίας και είναι απαραίτητο νέοι κόμβοι να προστεθούν στο δίκτυο. Επομένως το δίκτυο πρέπει να είναι σε θέση περιοδικά να επαναπροσδιορίζεται έτσι ώστε να συνεχίσει να λειτουργεί. Μεμονωμένοι κόμβοι μπορούν να αποσυνδεθούν από το δίκτυο αλλά πάντα πρέπει να διατηρείται σε μεγάλο βαθμό η συνεκτικότητα του.

Σε συνδυασμό με αυτά που έχουν ήδη αναφερθεί μπορούν να παρουσιαστούν επιγραμματικά τα βασικά χαρακτηριστικά των ad hoc δικτύων και των δικτύων αισθητήρων. Η έλλειψη κεντρικής υποδομής αποτελεί βασικό στοιχείο, το οποίο καθορίζει αποφασιστικά τον τρόπο διάρθρωσης και λειτουργίας του δικτύου. Επίσης το ασύρματο περιβάλλον δημιουργεί απρόβλεπτους παράγοντες, οι οποίοι μπορούν να επηρεάσουν την αποδοτικότητα όπως για παράδειγμα οι καιρικές συνθήκες. Το βασικότερο όμως χαρακτηριστικό θεωρούνται οι περιορισμένοι πόροι, οι οποίοι έχουν άμεση επίπτωση στην ενέργεια που καταναλώνουν. Το γεγονός αυτό ενισχύεται από το δεδομένο ότι οι κόμβοι των δικτύων αυτών είναι πολλές φορές απαραίτητο να τοποθετηθούν σε δυσπρόσιτα μέρη, χωρίς τη δυνατότητα άμεσης αντικατάστασης. Επομένως η μείωση της κατανάλωσης ενέργειας οδηγεί σε αύξηση του χρόνου ζωής του κάθε κόμβου και άρα σπανιότερη ανάγκη αντικατάστασης λόγω εξάντλησης των ενεργειακών πόρων.

Η δημιουργία ενεργειακά αποδοτικών αλγορίθμων που θα διατηρούν τη συνεκτικότητα του δικτύου και θα βελτιώνουν την επίδοση χαρακτηριστικών όπως του χρόνου ζωής του δικτύου έχει απασχολήσει πολλές ερευνητικές ομάδες μέχρι σήμερα. Οι αλγόριθμοι αυτοί ασχολούνται με τον έλεγχο τοπολογίας. Η τοπολογία ενός ασύρματου δικτύου είναι το σύνολο των επικοινωνιακών ζεύξεων μεταξύ κόμβων. Η τοπολογία εξαρτάται από μη ελεγχόμενους παράγοντες όπως η κινητικότητα, ο καιρός, οι παρεμβολές, ο θόρυβος αλλά και από ελεγχόμενους παράγοντες όπως η ενέργεια εκπομπής και η κατεύθυνση της κεραίας. Ο έλεγχος της τοπολογίας είναι απαραίτητος. Σε περίπτωση χρησιμοποίησης λανθασμένης τοπολογίας μπορεί να μειωθεί σημαντικά η ικανότητα του δικτύου, να αυξηθεί η από άκρη σε άκρη καθυστέρηση πακέτου και επομένως να αυξηθούν οι αποτυχίες κόμβων ως προς την λειτουργίας τους. Για παράδειγμα στην περίπτωση που η τοπολογία είναι πολύ αραιή υπάρχει μεγαλύτερη πιθανότητα για χωρισμό του δικτύου σε διάφορα

μέρη και για υψηλές από άκρη σε άκρη καθυστερήσεις. Αντίθετα εάν η τοπολογία είναι πολύ πυκνή υπάρχει κίνδυνος μείωσης της χωρητικότητας του δικτύου.

Η παρούσα διπλωματική εργασία ασχολείται με την υλοποίηση και τη σύγκριση έξι αλγορίθμων ελέγχου τοπολογίας σε ασύρματα ad hoc δίκτυα και δίκτυα αισθητήρων. Συναφές είναι και το πρόβλημα ελέγχου της ενέργειας εκπομπής. Το πρόβλημα αυτό συνίσταται στο να επιλεχθεί το επίπεδο της ισχύος εκπομπής για κάθε κόμβο στο ασύρματο δίκτυο. Η επιλογή της ενέργειας εκπομπής, επομένως και της ακτίνας μετάδοσης, επηρεάζει την μεταφορική ικανότητα του δικτύου και αν ληφθεί υπόψιν ο περιορισμός της επιλογής μικρών "βημάτων" στο δίκτυο για αποφυγή παρεμβολών που θα υπήρχαν στην αντίθετη περίπτωση, είναι γενικά επιθυμητή η μείωση του επιπέδου της ενέργειας. Η επιλογή του επιπέδου ενέργειας επηρεάζει και το χρόνο ζωής της μπαταρίας όπως ήδη έχει αναφερθεί. Επιπλέον σημαντική επιρροή ασκείται και στη δρομολόγηση διότι η περιοχή κάλυψης των πομπών εξαρτάται από το επίπεδο ενέργειας που ορίζεται κάθε φορά.

Στο επόμενο κεφάλαιο γίνεται μία σύντομη αναφορά άλλων μεθόδων που χρησιμοποιήθηκαν για την επίλυση των προβλημάτων που αναφέρθηκαν. Στη συνέχεια ακολουθεί αναλυτική περιγραφή των έξι αλγορίθμων που υλοποιήθηκαν, έπειτα ο τρόπος υλοποίησής τους και τα προβλήματα που παρουσιάστηκαν. Τέλος, παρατίθενται τα αποτελέσματα των μετρήσεων που έγιναν με βάση τους αλγορίθμους, τα αποτελέσματα των μεταξύ τους συγκρίσεων και τα συμπεράσματα που προέκυψαν.

# Κεφάλαιο 2 – Σχετικές Εργασίες

Με στόχο την επίτευξη ελάχιστης κατανάλωσης ενέργειας και την ταυτόχρονη αύξηση της απόδοσης σε ένα ασύρματο ad hoc δίκτυο έχει πραγματοποιηθεί μεγάλη ερευνητική προσπάθεια και έχουν προταθεί διάφορα μοντέλα. Μερικά από αυτά θα αναφερθούν περιληπτικά στην συνέχεια και κάποια άλλα αποτέλεσαν τη βάση της συγκεκριμένης εργασίας. Αναλυτική περιγραφή και μελέτη των τελευταίων θα γίνει στο επόμενο κεφάλαιο.

## 2.1 Πρωτόκολλο Span

Στο [6] παρουσιάζεται μία τεχνική εξοικονόμησης ενέργειας για ασύρματα ad hoc δίκτυα που μειώνει την κατανάλωση ενέργειας χωρίς να ελαττώνει δραματικά την χωρητικότητα ή να επηρεάζει την συνεκτικότητα του δικτύου. Το Span στηρίζεται στην παρατήρηση ότι όταν μία περιοχή δικτύου μοιραζόμενων καναλιών έχει μία επαρκή πυκνότητα κόμβων, τότε μόνο ένας μικρός αριθμός αυτών πρέπει να είναι ενεργός κάθε στιγμή έτσι ώστε να προωθεί την κίνηση για τις ενεργές συνδέσεις.

Το Span είναι ένας καναμεμημένος, τυχαίος αλγόριθμος με βάση τον οποίο οι κόμβοι παίρνουν τοπικές αποφάσεις για το αν πρέπει να "κοιμηθούν" ή να συμμετάσχουν σε ένα δίκτυο προώθησης κορμού ως συντονιστές. Κάθε κόμβος βασίζει την απόφασή του σε έναν υπολογισμό για το πόσοι από τους γείτονές του θα επωφεληθούν από τον ίδιο αν μείνει "ξύπνιος" αλλά και το ποσό της ενέργειας που είναι διαθέσιμο για αυτόν. Η διαφύλαξη της χωρητικότητας του δικτύου απαιτεί από κάθε κόμβο να αποφασίσει εθελοντικά να οριστεί ως κόμβος-συντονιστής εάν ανακαλύψει ότι δύο από τους γείτονές του δεν είναι σε θέση να επικοινωνήσουν μεταξύ τους απευθείας ή μέσω ενός ήδη υπάρχοντα κόμβου-συντονιστή. Για να διατηρηθεί ο αριθμός των πλεοναζόντων συντονιστών χαμηλός και για να υπάρξει εξισορρόπηση του συγκεκριμένου ρόλου μεταξύ όλων των κόμβων, κάθε κόμβος καθυστερεί ανακοινώνοντας την θέλησή του με μία τυχαία καθυστέρηση που εξαρτάται από δύο παράγοντες: το ποσό της εναπομείνασας ενέργειας της μπαταρίας και από τον αριθμό των ζευγών των γειτόνων που μπορεί να συνδέσει μεταξύ τους. Αυτός ο συνδυασμός εξασφαλίζει, με μεγάλη πιθανότητα, ένα συνεκτικό υποδίκτυο που διατηρεί την χωρητικότητα σε κάθε χρονική στιγμή, όπου οι κόμβοι τείνουν να καταναλώνουν ενέργεια με τον ίδιο ρυθμό περίπου. Το Span καταφέρνει όλα τα παραπάνω χρησιμοποιώντας μόνο τοπικές πληροφορίες.

### 2.1.1 Σχεδιασμός του Span

Το Span εκλέγει συντονιστές από όλους τους κόμβους του δικτύου. Οι συντονιστές μένουν ξύπνιοι συνεχώς και βοηθούν στη δρομολόγηση των πακέτων στο δίκτυο, ενώ οι άλλοι κόμβοι παραμένουν σε κατάσταση εξοικονόμησης ενέργειας (idle) και περιοδικά ελέγχουν αν θα πρέπει να ξυπνήσουν και να γίνουν συντονιστές.

Το Span επιτυγχάνει τέσσερις στόχους. Αρχικά εξασφαλίζει ότι έχουν επιλεχθεί αρκετοί συντονιστές έτσι ώστε κάθε κόμβος να έχει στην περιοχή κάλυψής του τουλάχιστον έναν συντονιστή. Δεύτερον, εναλλάσσει τους συντονιστές με σκοπό να εξασφαλίσει ότι όλοι οι κόμβοι μοιράζονται ισάξια το βάρος της δρομολόγησης. Τρίτον, επιχειρεί να ελαχιστοποιήσει τον αριθμό των κόμβων που επιλέγονται ως συντονιστές επιμηκύνοντας έτσι το χρόνο ζωής του δικτύου, χωρίς όμως παράλληλα

να δημιουργείται μείωση της απόδοσης ή αύξηση της καθυστέρησης. Τέταρτον, επιλέγει συντονιστές χρησιμοποιώντας μόνο τοπικές πληροφορίες με έναν αποκεντρωμένο τρόπο.

Κάθε κόμβος περιοδικά εκπέμπει ένα HELLO μήνυμα το οποίο περιέχει την κατάστασή του, τους τρέχοντες συντονιστές και τους τρέχοντες γείτονές του. Αυτά τα μηνύματα και οι καταστάσεις που διατηρούνται από το Span, είναι παρόμοια με αυτά άλλων πρωτοκόλλων. Επιπλέον διατηρούνται και κάποια επιπρόσθετα στοιχεία, αυτά των συντονιστών και των συντονιστών των γειτόνων.

### 2.1.2 Ανακοίνωση συντονιστή κόμβου

Περιοδικά ένας μη συντονιστής-κόμβος καθορίζει εάν θα έπρεπε να γίνει συντονιστής ή όχι. Αυτό γίνεται με τη χρήση του εξής κανόνα: εάν δύο γείτονες ενός κόμβου που τη δεδομένη στιγμή είναι συντονιστές δεν μπορούν να προσεγγίσουν ο ένας τον άλλον είτε απευθείας είτε μέσω ενός ή δύο συντονιστών, τότε ο κόμβος πρέπει να γίνει συντονιστής.

Ενώ αυτός ο αλγόριθμος επιλογής δεν εξασφαλίζει τον ελάχιστο αριθμό συντονιστών που απαιτούνται για να διατηρηθεί η συνεκτικότητα, σχηματίζει ένα δίκτυο που χονδρικά περιλαμβάνει έναν συντονιστή σε κάθε περιοχή κάλυψης σε ολόκληρη την τοπολογία του δικτύου. Εφόσον τα πακέτα θα δρομολογηθούν μέσω συντονιστή αυτή η τοπολογία οφείλει να παρέχει καλή χωρητικότητα.

Όταν περισσότεροι από έναν κόμβοι ανακαλύπτουν την έλλειψη ενός συντονιστή την ίδια χρονική στιγμή και αποφασίζουν όλοι να γίνουν συντονιστές, παρατηρούνται παραπάνω από μία ανακοινώσεις. Το πρόβλημα αυτό επιλύεται με την καθυστέρηση της ανακοίνωσης μέσω μίας τυχαίας συνάρτησης καθυστέρησης επαναμετάδοσης. Κάθε κόμβος επιλέγει μία τιμή καθυστέρησης και καθυστερεί το μήνυμα HELLO που ανακοινώνει την εθελοντική πρόθεση του κόμβου να γίνει συντονιστής για αυτό το χρονικό διάστημα. Εάν στο τέλος της καθυστέρησης ο κόμβος δεν έχει δεχθεί άλλα μηνύματα HELLO από άλλους δυνητικούς συντονιστές, τότε στέλνει το δικό του μήνυμα. Αλλιώς επαναπροσδιορίζει μέσω του κανόνα που ήδη αναφέρθηκε την ανάγκη για να γίνει συντονιστής και επιχειρεί την ανακοίνωση μόνο εάν ο κανόνας ισχύει ακόμα.

### 2.1.3 Αποχώρηση συντονιστή κόμβου

Κάθε συντονιστής ελέγχει περιοδικά εάν πρέπει να αποσυρθεί ή όχι. Ένας κόμβος πρέπει να αποχωρήσει εάν σε κάθε ζευγάρι από τους γείτονές του οι κόμβοι μπορούν να προσεγγίσουν ο ένας τον άλλον είτε απευθείας, είτε μέσω κάποιων άλλων συντονιστών. Ωστόσο για να διασφαλιστεί η δικαιοσύνη αφού ένας κόμβος έχει υπάρξει συντονιστής για ένα χρονικό διάστημα, αποσύρεται αν σε κάθε ζευγάρι από τους γείτονές του οι κόμβοι μπορούν να προσεγγίσουν ο ένας τον άλλον μέσω κάποιων άλλων γειτόνων ακόμα και αν αυτοί οι γείτονες δεν είναι τη δεδομένη στιγμή συντονιστές. Αυτός ο κανόνας δίνει την δυνατότητα και σε άλλους γείτονες να γίνουν συντονιστές.

Για να αποφευχθεί προσωρινή απώλεια συνεκτικότητας μεταξύ του μηνύματος αποχώρησης ενός συντονιστή και της ανακοίνωσης ενός νέου, ο κόμβος συνεχίζει να λειτουργεί ως συντονιστής για ένα μικρό χρονικό διάστημα ακόμα και μετά την ανακοίνωση της αποχώρησής του. Αυτή η “περίοδος χάριτος” επιτρέπει στο πρωτόκολλο να χρησιμοποιεί τον παλιό συντονιστή μέχρι να αναλάβει ο καινούριος.

Συμπερασματικά, πρέπει να αναφερθεί ότι παρατηρήθηκε στο [6] ύστερα από εφαρμογή του πρωτοκόλλου, βελτίωση κατά ένα παράγοντα του δύο στο χρόνος ζωής του συστήματος. Αυτό παρατηρήθηκε σε μεγάλο εύρος πυκνοτήτων κόμβων και επιπρόσθετα διαπιστώθηκε ότι το ποσό της ενέργειας που εξοικονομείται αυξάνει ελαφρά όσο η πυκνότητα αυξάνει.

Παρόμοιο στη λειτουργία και τη λογική με το Span είναι το πρωτόκολλο GAF. Επιγραμματικά, κάθε κόμβος στο GAF χρησιμοποιεί τοπική πληροφορία βασισμένη στο GPS έτσι ώστε να συσχετίσει τον εαυτό του με ένα "εικονικό πλέγμα" και όλη η περιοχή να διαιρεθεί σε ένα πλέγμα τετράγωνα. Ο κόμβος με την υψηλότερη τοπική ενέργεια μέσα σε κάθε τετράγωνο του πλέγματος ορίζεται ως ο επικεφαλής. Άλλοι κόμβοι στο ίδιο τετράγωνο μπορούν να θεωρηθούν πλεονάζοντες για τη δρομολόγηση πακέτων και μπορούν να "κοιμηθούν" για κάποιο χρονικό διάστημα. Περισσότερες λεπτομέρειες για το πρωτόκολλο GAF μπορεί κάποιος να βρει στο [8].

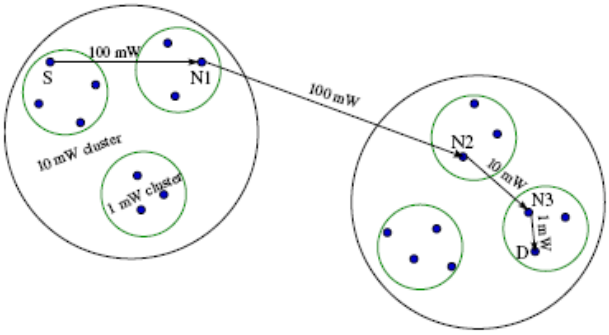
## 2.2 Έλεγχος τοπολογίας και ομαδοποίηση

Στο [4] εξετάζεται το πρόβλημα ελέγχου τοπολογίας όταν οι κόμβοι δεν είναι ομοιόμορφα κατανεμημένοι στο χώρο, δηλαδή συνυπάρχουν σε ομάδες. Σε αυτές τις περιπτώσεις επιδιώκεται να εφαρμοστεί ανά πακέτο έλεγχος τοπολογίας ανάλογα με την πηγή και τον προορισμό του πακέτου. Ο στόχος είναι να επιλεχθεί το επίπεδο της ενέργειας εκπομπής και έτσι ώστε οι περισσότερες από τις επικοινωνίες εντός της ομάδας να γίνονται με χαμηλά επίπεδα ενέργειας και ένα υψηλότερο επίπεδο να χρησιμοποιείται μόνο όταν χρειάζεται επικοινωνία με άλλη ομάδα. Προτείνονται τρεις λύσεις για την εφαρμογή της παραπάνω λογικής που αναλύονται στη συνέχεια.

### 2.2.1 Πρωτόκολλο CLUSTERPOW

Το πρωτόκολλο ελέγχου τοπολογίας CLUSTERPOW σχεδιάστηκε για έλεγχο τοπολογίας, ομαδοποίηση και δρομολόγηση σε δίκτυα των οποίων οι κόμβοι δεν είναι ομοιόμορφα κατανεμημένοι στο χώρο. Μία διαδρομή στο πρωτόκολλο αυτό περιλαμβάνει βήματα διαφορετικής ενέργειας εκπομπής, έτσι ώστε να διατηρείται και η δομή ομάδας του δικτύου. Ο αλγόριθμος περιλαμβάνει τη χρησιμοποίηση του ελάχιστου δυνατού επιπέδου ενέργειας $p$ ώστε ο προορισμός να προσεγγίζεται (με πολλαπλά βήματα) χωρίς τη χρήση επιπέδων ενέργειας μεγαλύτερων του $p$. Ο αλγόριθμος εκτελείται στην πηγή και σε κάθε ενδιάμεσο κόμβο κατά μήκος της διαδρομής από την πηγή στον προορισμό για κάθε πακέτο.

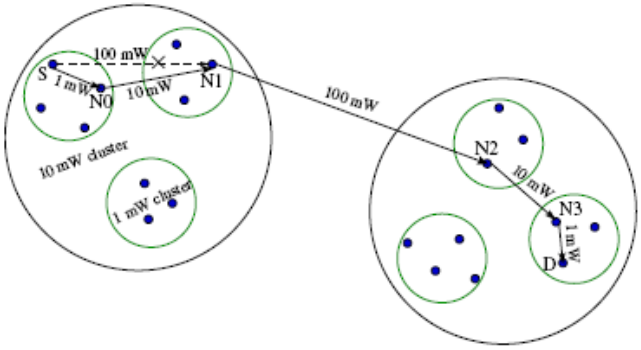
Η διαδρομή που προκύπτει τρέχοντας τον αλγόριθμο σε ένα τυπικό ομαδοποιημένο δίκτυο παρουσιάζεται στο παρακάτω Σχήμα 1.




**Σχήμα 1. Δρομολόγηση μέσω του CLUSTERPOW σε ένα τυπικό μη ομογενές δίκτυο**

Το δίκτυο έχει τρία επίπεδα ομαδοποίησης, τα οποία ανταποκρίνονται σε επίπεδα ενέργειας $1mW$, $10mW$ και $100mW$ και ολόκληρο το δίκτυο θεωρείται μία ομάδα των $100mW$. Για τη μετάβαση από την πηγή $S$ στον προορισμό $D$, χρησιμοποιείται ένα επίπεδο ενέργειας των $100mW$ σε κάθε βήμα μέχρι το πακέτο να φτάσει στην ομάδα των $10mW$, στο οποίο ανήκει ο προορισμός. Στη συνέχεια τα $10mW$ χρησιμοποιούνται σε κάθε βήμα μέχρι να προσεγγιστεί η ομάδα του $1mW$ στο οποίο ανήκει ο προορισμός και τελικά μία σειρά από βήματα του $1mW$ οδηγεί το πακέτο στον προορισμό. Πρέπει να σημειωθεί ότι ο έλεγχος ενέργειας εκπομπής οδηγεί σε αυτόματη ομαδοποίηση του δικτύου.

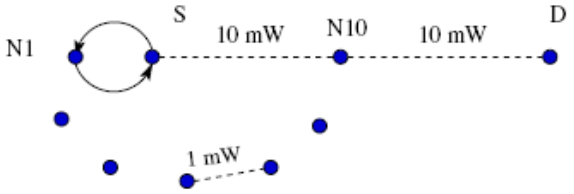
Περισσότερα βήματα χαμηλής ενέργειας είναι προτιμότερα από λιγότερα βήματα υψηλής ενέργειας. Σε σχέση με το προηγούμενο σχήμα μεγάλο πλεονέκτημα θα αποτελούσε η αντικατάσταση του πρώτου βήματος των $100mW$ από δύο μικρότερα βήματα του $1mW$ και $10mW$ αντίστοιχα, όπως φαίνεται στο Σχήμα 2.




**Σχήμα 2. Τροποποίηση του πρωτοκόλλου CLUSTERPOW**

Φαίνεται πιθανό να επιτευχθεί η παραπάνω εκδοχή μέσω της εξής προσέγγισης: περιοδικά επαναλαμβανόμενη αναζήτηση του επόμενου βήματος με στόχο χαμηλά επίπεδα ενέργειας, τα οποία είναι δυνατόν να προσεγγιστούν. Επομένως στο Σχήμα 2 το επόμενο βήμα $N1$ στον κόμβο $S$ αναζητείται με βάση την παραπάνω προσέγγιση και διαπιστώνεται ότι μπορεί να επιτευχθεί στα $10mW$ μέσω του $N0$, ο οποίος τελικά μπορεί να προσεγγιστεί με βήμα $1mW$. Άρα το πακέτο δίνεται στα $1mW$ στον $N0$. Ο ίδιος αλγόριθμος εκτελείται στον $N0$, όταν το πακέτο φτάσει εκεί, και σε κάθε επόμενο κόμβο στο μονοπάτι. Φαίνεται επομένως να επιτυγχάνεται με αυτόν τον τρόπο μία αποδοτικότερη υλοποίηση. Ωστόσο αυτό το σχήμα περιοδικά επαναλαμβανόμενης αναζήτησης μπορεί να οδηγήσει σε άπειρους βρόχους, όπως παρουσιάζεται στη συνέχεια.

Το αντιπαράδειγμα που επιβεβαιώνει τον παραπάνω ισχυρισμό παρουσιάζεται στο Σχήμα 3.



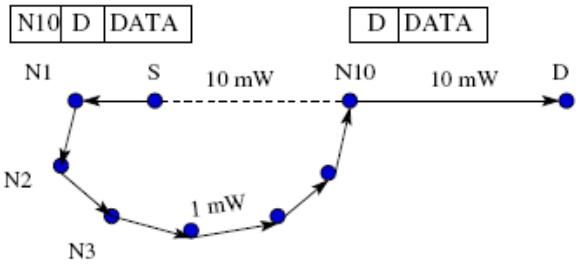

**Σχήμα 3. Παράδειγμα δημιουργίας άπειρου βρόχου**

Ο κόμβος $S$ πρέπει να στείλει ένα πακέτο στον κόμβο $D$. Φαίνεται ότι το επόμενο βήμα είναι ο κόμβος $N10$ στο επίπεδο ενέργειας των $10mW$. Μέσω περιοδικά επαναλαμβανόμενης αναζήτησης διαπιστώνεται ότι ο $N10$ είναι δυνατόν να

προσεγγιστεί με $1mW$ και το επόμενο βήμα προκύπτει ο $N1$. Επομένως ο $S$ προωθεί το πακέτο στον $N1$ στα $1mW$. Αφού το πακέτο προσεγγίσει τον $N1$, ο κόμβος τρέχει τον ίδιο αλγόριθμο. Βρίσκει ότι το χαμηλότερο επίπεδο ενέργειας με το οποίο μπορεί να προσεγγιστεί ο $D$ είναι τα $10mW$ και το επόμενο βήμα είναι ο $S$. Ο $S$ από τη μεριά του προσεγγίζεται στα $1mW$, με αποτέλεσμα το πακέτο να παραδίδεται πάλι στον $S$. Προκύπτει επομένως ένας άπειρος βρόχος.

### 2.2.2 Πρωτόκολλο Tunnelled CLUSTERPOW

Το παραπάνω σχήμα περιοδικά επαναλαμβανόμενης αναζήτησης μπορεί να τροποποιηθεί έτσι ώστε να μην υπάρχει περίπτωση ο αλγόριθμος να παγιδευτεί σε ατέρμονα βρόχο. Αυτό επιτυγχάνεται μέσω tunneling του πακέτου στο επόμενο βήμα χρησιμοποιώντας χαμηλότερα επίπεδα ενέργειας, αντί να αποστέλλεται το πακέτο απευθείας. Ένας μηχανισμός για να επιτευχθεί αυτό είναι χρησιμοποιώντας τη μέθοδο ενθυλάκωσης $IP$. Επομένως όπως πραγματοποιείται η αναζήτηση του επόμενου βήματος, ενσωματώνεται με τον ίδιο τρόπο στο πακέτο η διεύθυνση του κόμβου για τον οποίο πραγματοποιήθηκε η αναζήτηση. Ανάλογα πραγματοποιείται και η αποθυλάκωση όταν το πακέτο φτάσει τον επόμενο κόμβο. Αυτό είναι το πρωτόκολλο tunneled CLUSTERPOW. Όπως φαίνεται στο Σχήμα 4 επιλύεται το πρόβλημα του άπειρου βρόχου που αναλύθηκε προηγουμένως.




**Σχήμα 4. Επίλυση του προβλήματος άπειρου βρόχου μέσου του πρωτοκόλλου Tunnelled CLUSTERPOW.**

Σε αυτήν την περίπτωση όταν ο κόμβος $S$ προωθεί το πακέτο στον $N1$, ενσωματώνει στο πακέτο τη διεύθυνση του κόμβου $N10$. Έτσι ο $N1$ εκτελεί μία αναζήτηση δρομολόγησης, όχι για τον προορισμό $D$, αλλά για τον κόμβο $N10$. Βρίσκει ότι ο $N10$ προσεγγίζεται στα $1mW$ μέσω του μονοπατιού $N2, N3, ...$ και προωθεί το πακέτο στον $N2$ στο $1mW$. Όταν το πακέτο φτάνει στον $N10$ αποθυλακώνει το πακέτο και το στέλνει στον $D$ στα $10mW$. Έτσι το πακέτο φτάνει τελικά στον προορισμό του. Αποδεικνύεται στο [4] ότι πάντα τα πακέτα στο συγκεκριμένο πρωτόκολλο φτάνουν στον προορισμό τους.

### 2.2.3 Πρωτόκολλο MINPOW

Τα πρωτόκολλα που έχουν παρουσιαστεί μέχρι τώρα εστιάζουν στη μεγιστοποίηση της απόδοσης του δικτύου. Επιθυμητή είναι επίσης η ελαχιστοποίηση της κατανάλωσης ενέργειας. Για την ταυτόχρονη επίτευξη των δύο παραπάνω στόχων προτείνεται ένα διαφορετικό πρωτόκολλο που ονομάζεται MINPOW, το οποίο ελαχιστοποιεί τη συνολική κατανάλωση ενέργειας σε μία διαδρομή. Πρόκειται για

τον αλγόριθμο Bellman-Ford με διαδοχικές ετικέτες και τη συνολική κατανάλωση ενέργειας ως κόστος αντί για τα ενδιάμεσα βημάτων που συνήθως χρησιμοποιείται.

Το MINPOW εφαρμόζεται κατευθείαν στο στρώμα δικτύου χρησιμοποιώντας μόνο πακέτα HELLO, χωρίς να απαιτεί υποστήριξη από το φυσικό στρώμα για την εκτίμηση του κόστους ενέργειας ανά ζεύξη.

Πρέπει να σημειωθεί ότι το συγκεκριμένο πρωτόκολλο υπολογίζει την ενέργεια που καταναλώνεται για επικοινωνίες, δηλαδή για εκπομπή και αποδοχή πακέτων. Δεν υπολογίζεται η ενέργεια που καταναλώνεται όταν οι κόμβοι δεν επικοινωνούν (idle energy). Εάν αυτού του είδους η κατανάλωση ενέργειας είναι υψηλή, τότε μία αποδοτική στρατηγική εξοικονόμησης ενέργειας θα είναι να οι κόμβοι να "κοιμηθούν". Επομένως το συγκεκριμένο πρωτόκολλο μπορεί να συνδυαστεί με το πρωτόκολλο Span που παρουσιάστηκε προηγουμένως.

Το κόστος ζεύξης, δηλαδή η κατανάλωση ενέργειας για την επικοινωνία, αποτελείται από τρεις παράγοντες: $P_{Rx_{elec}}$ είναι η ενέργεια που καταναλώνεται στα ηλεκτρονικά μέρη του δέκτη, $P_{Tx_{elec}}$ η ενέργεια που καταναλώνεται από τα ηλεκτρονικά μέρη του πομπού και $P_{Tx_{Rad}}(p)$ η ενέργεια που καταναλώνεται από τον ενισχυτή ισχύος έτσι ώστε να μεταφερθεί ένα πακέτο στο επίπεδο ενέργειας $p$, όπου $p$ η πραγματική ενέργεια που τοποθετείται στο περιβάλλον. Η $P_{Rx_{elec}}$ και η $P_{Tx_{elec}}$ είναι γνωστές τοπικά στον δέκτη και τον πομπό αντίστοιχα, σύμφωνα με τις προδιαγραφές του υλικού εξοπλισμού. Αυτό επιτρέπει στο πρωτόκολλο να λειτουργεί και με ετερογενές υλικό εξοπλισμό. Ο τρίτος παράγοντας μπορεί να υπολογιστεί εάν προσδιοριστεί η ελάχιστη εκπεμπόμενη ενέργεια $p$ που είναι απαραίτητη για την πραγματοποίηση της ζεύξης.

Ένας τρόπος για να εκτιμηθεί αυτή η ελάχιστη εκπεμπόμενη ενέργεια είναι μετρώντας την απόσταση μεταξύ δύο κόμβων στη ζεύξη και χρησιμοποιώντας ένα δεκαδικό μοντέλο, υποθέτοντας ότι οι απώλειες δρόμου στο μέσο ακολουθούν νόμο αντίστροφης δύναμης $\alpha$, με $a \geq 2$, δηλαδή η λαμβανόμενη ενέργεια σε απόσταση $\rho$ από τον πομπό χρησιμοποιώντας ένα επίπεδο ενέργειας $P_{trans}$ είναι $\frac{cP_{trans}}{\rho^{cx}}$, όπου $c$ μία σταθερά. Εάν υποτεθεί ότι για να γίνει αποδοχή πακέτου το επίπεδο ενέργειας πρέπει να είναι τουλάχιστον $\gamma$, δηλαδή $\frac{cP_{trans}}{\rho^{a}} \geq \gamma$. Επομένως το απαιτούμενο επίπεδο ενέργειας εκπομπής είναι τουλάχιστον $\frac{\gamma\rho^{a}}{c}$. Έτσι η $P_{trans}$ μπορεί να εκτιμηθεί δεδομένης της λαμβανόμενης ενέργειας και της απόστασης από τον πομπό.

## 2.3 Έλεγχος τοπολογίας ασύρματων δικτύων χρησιμοποιώντας επαναπροσδιορισμό της εκπεμπόμενης ενέργειας.

Στο [5] αντιμετωπίζεται το πρόβλημα της προσαρμογής της εκπεμπόμενης ενέργειας κόμβων σε ασύρματο ad hoc δίκτυο για τη δημιουργία επιθυμητής τοπολογίας. Για την επίλυση του προβλήματος παρουσιάζονται δύο κεντροποιημένοι αλγόριθμοι για στατικά δίκτυα και δύο κατανεμημένοι ευριστικοί αλγόριθμοι για ασύρματα δίκτυα που προσαρμόζουν την εκπεμπόμενη ενέργεια κόμβων σε σχέση με

αλλαγές που συμβαίνουν στην τοπολογία και επιχειρούν να τη διατηρήσουν συνδεδεμένη χρησιμοποιώντας ελάχιστη ενέργεια.

### 2.3.1 Στατικά δίκτυα–κεντροποιημένοι αλγόριθμοι

Ένα στατικό δίκτυο επιτρέπει τη χρησιμοποίηση ενός κεντροποιημένου ή ακόμα και ενός αλγορίθμου μη-πραγματικού χρόνου για να υπολογιστούν τα επίπεδα της εκπεμπόμενης ενέργειας. Οι θέσεις των κόμβων, όπως και η συνάρτηση ελάχιστης ενέργειας είναι διαθέσιμα ως είσοδος στον αλγόριθμο. Παρουσιάζονται δύο πολυωνυμικοί αλγόριθμοι, από τους οποίους ο ένας έχει ως αποτέλεσμα ένα συνδεδεμένο δίκτυο και ο άλλος ένα διπλά συνδεδεμένο δίκτυο. Ο αλγόριθμος CONNECT δίνεται στο

**Σχήμα 5** ακριβώς όπως διατυπώνεται στο [5].

```
Algorithm CONNECT

Input: (1) Multihop wireless network M=(N,L)
       (2) Least-power function λ
Output: Power levels p for each node that induces a
connected graph

begin
1.sort node pairs in non-decreasing order of mutual
distance
2.initialize |N| clusters, one per node
3.for each (u,v) in sorted order do
4.  if cluster(u) ≠ cluster(v)
5.      p(u)= p(υ)=distance(u,υ)
6.     merge cluster(u) with cluster(v)
7.     if number of clusters is 1
       then end
8.perNodeMinimalize(M,λ,p,1) end
```

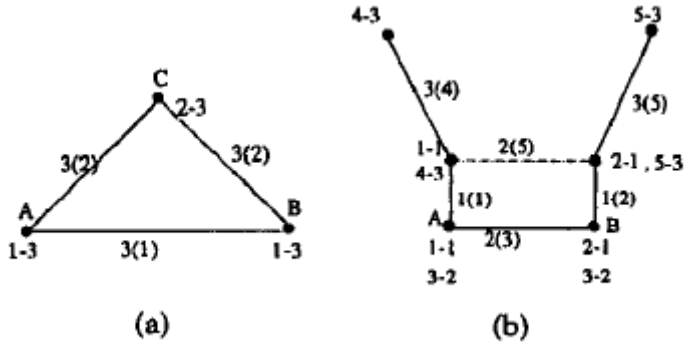
```
Procedure perNodeMinimalize(M,λ,p,k)

begin
1.let S=sorted node pair list
2.for each node u do
3.    T={(n1,n2)∈S : u=n1 or u=n2}
4.    sort T in non-increasing order of distance
5.    discard from T all (x,y) such that λ(d(x,y))> p(u)
6.    for (x,y)∈T using binary search do
7.       if graph with p(u)=λ(d(x,y)) is not k-connected,
         stop
8.       else p(u)=λ(d(x,y))
end
```

**Σχήμα 5. Αλγόριθμος `CONNECT` και διαδικασία `perNodeMinimalize`**

Είναι ένας "άπληστος" αλγόριθμος που συγχωνεύει επαναληπτικά συνεκτικά υποτμήματα μέχρι να προκύψει μόνο ένα. Αρχικά κάθε κόμβος αποτελεί ένα συνεκτικό υποτμήμα. Ζευγάρια κόμβων επιλέγονται σε αυξανόμενη σειρά της μεταξύ τους απόστασης. Εάν οι κόμβοι ανήκουν σε διαφορετικά συνεκτικά υποτμήματα, τότε η εκπεμπόμενη ενέργεια του καθένα αυξάνεται έτσι ώστε να προσεγγίσει τον άλλο. Αυτό γίνεται μέχρι το δίκτυο να γίνει συνδεδεμένο. Η περιγραφή υποθέτει, χάριν απλοποίησης, ότι η συνεκτικότητα του δικτύου μπορεί να επιτευχθεί χωρίς να ξεπεραστεί η μέγιστη δυνατή τιμή εκπεμπόμενης ενέργειας. Ωστόσο, ο αλγόριθμος μπορεί εύκολα να τροποποιηθεί για να επιστρέφει μήνυμα λάθους εάν αυτή η προϋπόθεση δεν ισχύει.

Αποδεικνύεται όμως ότι όταν επιτευχθεί η ελάχιστη τιμή της μέγιστης εκπεμπόμενης ενέργειας δεν επιτυγχάνεται απαραίτητα και η ελάχιστη τιμή ανά κόμβο. Αυτό ισχύει κυρίως διότι μία αύξηση της ενέργειας μπορεί να προσθέσει περισσότερες από μια ζεύξεις στο γράφο που σχηματίζεται. Τέτοιες πρόσθετες ζεύξεις, διαφορετικές από αυτές που υπάρχουν μεταξύ των επιλεγμένων ζευγαριών κόμβων, αποκαλούνται παράπλευρες ζεύξεις. Ένα παράδειγμα τέτοιου είδους ζεύξης φαίνεται στο Σχήμα 6. Μία παράπλευρη ζεύξη μπορεί να σχηματίσει βρόχο με άλλες ζεύξεις και μπορεί να επιτρέψει την ελάττωση των επιπέδων ενέργειας και την απαλοιφή κάποιων ζεύξεων που προστέθηκαν προηγουμένως. Ένα παράδειγμα φαίνεται στο (b) μέρος του σχήματος.




**Σχήμα 6. Παρουσίαση παράπλευρων ζεύξεων**

Στο Σχήμα 6 οι παράπλευρες ζεύξεις παριστάνονται με διακεκομμένες γραμμές. Η λεζάντα για τους κόμβους είναι της μορφής $s-p$, όπου $s$ το όνομα του βήματος και $p$ η ενέργεια που ανατίθεται κατά τη διάρκεια του βήματος. Η λεζάντα για τις ζεύξεις είναι της μορφής $d(s)$, όπου $d$ η απόσταση μεταξύ ανταποκρινόμενων κόμβων και $s$ το βήμα κατά το οποίο διαμορφώνεται αυτή η ζεύξη. Στο (a) μέρος του σχήματος φαίνεται το ελάχιστο ανά κόμβο αλλά στην εικόνα (b) οι ενέργειες των κόμβων $A$ και $B$ μπορούν να μειωθούν στο 1 με ταυτόχρονη διατήρηση της συνεκτικότητας του γράφου.

Μέσω της διαδικασίας `perNodeMinimalize` είναι πολύ εύκολο να απαλειφθούν οι παράπλευρες ζεύξεις και να γίνει η ανά κόμβο ανάθεση ελάχιστη. Η ιδέα βασίζεται στο να θεωρείται ένας κόμβος κάθε φορά και να μειώνεται η ενέργειά του όσο το δυνατόν περισσότερο έτσι ώστε να μην αποσυνδεθεί ο γράφος που δημιουργείται. Όμως θεωρητικά υπάρχουν άπειρα επίπεδα ενέργειας μεταξύ δύο συγκεκριμένων

επιπέδων και πρακτικά πρέπει να υπάρχει ένας μεγάλος αριθμός επιπέδων ενέργειας που εξαρτώνται από το βήμα της προσαρμογής ενέργειας που προσφέρει η ακτίνα.

Στη συγκεκριμένη προσέγγιση χρησιμοποιείται δυαδική αναζήτηση στο σύνολο των επιπέδων ενέργειας που έχουν σημασία, κάτι που δεν εξαρτάται από το βήμα της προσαρμογής ενέργειας. Αυτό το σύνολο καθορίζεται από μια ομάδα κόμβων που είναι μέσα στην περιοχή κάλυψης του εξεταζόμενου κόμβου με το τρέχων επίπεδο ενέργειας.

Η αναβάθμιση ενός συνδεδεμένου δικτύου σε ένα διπλά συνδεδεμένο πραγματοποιείται μέσω του αλγορίθμου BICONN-AUGMENT που δίνεται στο Σχήμα 7 ακριβώς όπως παρουσιάστηκε στο [5].

```
Algorithm BICONN-AUGMENT

Input: (1) Multihop wireless network M=(N,L)
       (2) Least-power function λ
       (3) Initial power assignment inducing connected network
Output: Power levels p for each node that induces a biconnected graph

begin
1.sort node pairs in non-decreasing order of distance
2.G =graph induced by (A,λ,p)
3.for each (u,υ) in sorted order do
4.  if biconn-comp(G,u) ≠ biconn-comp(G,v)
5.      q = λ (distance(u,v) )
6.      p(u) =max(q, p(u))
7.      p(υ) =max(q, p(υ))
8.      add (u,υ) to G
9.perNodeMinimalize(M,λ,p,2)
end
```

**Σχήμα 7. Αλγόριθμος BICONN-AUGMENT**

Και σε αυτή την περίπτωση περιγράφεται ένας "άπληστος" αλγόριθμος. Αρχικά αναγνωρίζονται τα διπλά συνδεδεμένα συνεκτικά υποτμήματα στο γράφο που δημιουργείται από τον προηγούμενο αλγόριθμο. Αυτό γίνεται από μία μέθοδο βασισμένη σε μία συγκεκριμένη μέθοδο αναζήτησης. Επομένως τα ζεύγη κόμβων επιλέγονται με αυξανόμενη σειρά σε σχέση με τη μεταξύ τους απόστασης και ενώνονται μόνο αν βρίσκονται σε διαφορετικά διπλά συνδεδεμένα συνεκτικά υποτμήματα. Αυτό συνεχίζεται μέχρι το δίκτυο να γίνει διπλά συνδεδεμένο.

Η ίδια διαδικασία που χρησιμοποιείται από τον προηγούμενο αλγόριθμο χρησιμοποιείται και σε αυτή την περίπτωση με τον ίδιο σκοπό. Σε αυτήν την περίπτωση το αποτέλεσμα μπορεί να μην είναι ελάχιστο ανά κόμβο ακόμα και υπό την απουσία παράπλευρων ζεύξεων.

### 2.3.2 Ασύρματα δίκτυα-κατανεμημένοι ευριστικοί αλγόριθμοι

Σε ένα ασύρματο κινητό δίκτυο η τοπολογία αλλάζει συνεχώς. Συνεπώς, οι ενέργειες εκπομπής των κόμβων θα πρέπει να επαναπροσδιορίζονται συνεχώς με σκοπό να διατηρηθεί η επιθυμητή τοπολογία. Επιπλέον,το πρωτόκολλο που θα υλοποιηθεί πρέπει να χρησιμοποιεί τοπικές ή ήδη διαθέσιμες πληροφορίες εφόσον η ανανέωση γενικής πληροφορίας, όπως οι θέσεις όλων των κόμβων, απαιτεί κατευθυνόμενο εξωτερικό έλεγχο. Επομένως τα προηγούμενα πρωτόκολλα δεν είναι κατάλληλα για αυτήν την περίπτωση και προτείνονται δύο νέοι αλγόριθμοι που ονομάζονται LINT και LILT αντίστοιχα και περιγράφονται στη συνέχεια.

### 2.3.2.1 Local Information No Topology (LINT)

Στον συγκεκριμένο αλγόριθμο ένας κόμβος προσδιορίζεται από τρεις παραμέτρους. Τον επιθυμητό βαθμό κόμβου $d_d$, ένα υψηλό κατώφλι στο βαθμό κόμβου $d_h$ και ένα χαμηλό κατώφλι $d_l$. Περιοδικά, ο κόμβος ελέγχει τον αριθμό των ενεργών γειτόνων (βαθμό) στον πίνακα γειτονίας του (που δημιουργείται από το μηχανισμό δρομολόγησης). Εάν ο βαθμός είναι μεγαλύτερος του $d_h$, ο κόμβος μειώνει την ενέργεια λειτουργίας του. Εάν ο βαθμός είναι μικρότερος από $d_l$, ο κόμβος αυξάνει την ενέργεια λειτουργίας του. Εάν κανένα από τα δύο δεν ισχύει, δεν εκτελείται καμία ενέργεια. Η αύξηση ή μείωση της ενέργειας περιορίζεται από την μέγιστη και την ελάχιστη δυνατή τιμή ενέργειας που καθορίζεται από τα χαρακτηριστικά της κεραίας.

Το μέγεθος της αλλαγής της τιμής της ενέργειας είναι συνάρτηση του επιθυμητού βαθμού κόμβου $d_d$ και του τρέχοντα βαθμού κόμβου $d$. Όσο μεγαλύτερη η απόσταση ανάμεσα στο $d_d$ και στο $d$ τόσο μεγαλύτερο το μέγεθος της αλλαγής.

Οι αλλαγές της ενέργειας γίνονται με ένα τυχαία-περιοδικό τρόπο. Αυτό συμβαίνει με σκοπό να απαλειφθούν πιθανά αδιέξοδα εκτέλεσης και οι παρεμβολές μεταξύ των πακέτων.

### 2.3.2.2 Local Information Link-State Topology (LILT)

Το LILT αποτελείται από δύο μέρη. Το πρωτόκολλο μείωσης γειτόνων (NRP) και το πρωτόκολλο αύξησης γειτόνων (NAP). Το NRP είναι βασικά ο μηχανισμός μέσω του οποίου επιχειρείται να διατηρηθεί ο βαθμός κόμβου γύρω από μία συγκεκριμένη τιμή. Το NAP ενεργοποιείται όταν συμβαίνει μία περιοδική ανανέωσης κατάστασης ζεύξης. Ο σκοπός του είναι να υπερπηδήσει το όριο του υψηλού κατωφλίου και να αυξήσει την ενέργεια εάν η αλλαγή της τοπολογίας που προωθείται από τη δρομολόγηση έχει ως αποτέλεσμα μία επιθυμητή τοπολογία. Η πρόκληση είναι να συντονιστούν τέτοιες αλλαγές ενέργειας με άλλους κόμβους, εφόσον δεν είναι επιθυμητό όλοι κόμβοι να συμμετάσχουν στην αλλαγή της τοπολογίας.

Αρχικά όλοι οι κόμβοι ξεκινούν με τη μέγιστη δυνατή ενέργεια . Με αυτόν τον τρόπο δημιουργείται ένα μέγιστα συνεκτικό δίκτυο. Στην συνέχεια ενεργοποιούνται τα δύο πρωτόκολλα που ήδη έχουν αναφερθεί.

Ένας κόμβος που λαμβάνει μια ανανέωση δρομολόγησης, πρώτα καθορίζει σε ποια από τις εξής τρεις καταστάσεις ανήκει η τοπολογία: συνδεδεμένη, διπλά συνδεδεμένη ή αποσυνδεδεμένη. Εάν είναι διπλά συνδεδεμένη δεν εκτελείται καμία ενέργεια. Εάν είναι αποσυνδεδεμένη ο κόμβος αυξάνει την ενέργειας εκπομπής του μέχρι τη μέγιστη δυνατή τιμή.

Εάν είναι συνδεδεμένη αλλά όχι διπλά συνδεδεμένη ο κόμβος κάνει μία προσπάθεια να επιτύχει το δεύτερο, όπως θα αναφερθεί ακολούθως. Ο πρώτος κόμβος βρίσκει την απόσταση από το κοντινότερο κρίσιμο σημείο συνεκτικότητας. Αυτό το σημείο είναι ένας κόμβος που η απαλοιφή του θα διαιρέσει το δίκτυο σε πολλά μέρη. Στην περίπτωση που το δίκτυο είναι συνδεδεμένο και όχι διπλά συνδεδεμένο υπάρχει πάντα τουλάχιστον ένα κρίσιμο σημείο συνεκτικότητας. Εφόσον βρεθούν αυτά τα σημεία τίθεται ένας χρονοδιακόπτη σε μία τιμή $t$ η οποία επιλέγεται τυχαία μέσω μιας εκθετικής κατανομής, η οποία είναι συνάρτηση της απόστασης από το κρίσιμο σημείο συνεκτικότητας. Εάν μετά από χρόνο $t$ το δίκτυο δεν είναι ακόμα διπλά συνδεδεμένο, τότε ο κόμβος αυξάνει την ενέργειά του στο μέγιστο δυνατό.

# Κεφάλαιο 3 – Περιγραφή αλγορίθμων

Δύο προσεγγίσεις για την βελτίωση της μεταφορικής ικανότητας του δικτύου και τον παράλληλο περιορισμό της κατανάλωσης ενέργειας του ασύρματου τερματικού είναι οι γωνίες κώνων (cone angles) και οι περιοχές σκυτάλης (relay regions). Σε αυτές τις δύο προσεγγίσεις βασίζονται οι αλγόριθμοι που υλοποιήθηκαν στην παρούσα διπλωματική εργασία και θα αναλυθούν παρακάτω. Πιο συγκεκριμένα, έχουν αναπτυχθεί πέντε αλγόριθμοι με βάση τις γωνίες κώνων και ένας με βάση τις περιοχές σκυτάλης.

Πριν την αναλυτική περιγραφή των αλγορίθμων δίνονται οι ορισμοί κάποιων απαραίτητων εννοιών όπως και η περιγραφή του μοντέλου διάδοσης που χρησιμοποιείται..

Το πρόβλημα ελέγχου τοπολογίας που επιχειρείται να επιλυθεί με διαφορετικό τρόπο από κάθε αλγόριθμο μπορεί επιγραμματικά να περιγραφεί ως εξής: θεωρείται ένα σύνολο $V$ από δυνατά ασύρματους κόμβους που βρίσκονται μέσα σε μία ορισμένη περιοχή. Κάθε κόμβος $u \in V$ καθορίζεται από τις συντεταγμένες του $(x(u), y(u))$ σε κάθε χρονική στιγμή. Κάθε κόμβος έχει μία συνάρτηση ενέργειας $p$, όπου $p(d)$ δίνει την ελάχιστη ενέργεια που απαιτείται για την δημιουργία επικοινωνιακής ζεύξης με έναν κόμβο $\upsilon$, ο οποίος βρίσκεται σε απόσταση $d$ από τον $u$. Θεωρείται επίσης ότι η μέγιστη ενέργεια εκπομπής $p_{\max}$ είναι ίδια για κάθε κόμβο και ότι η μέγιστη απόσταση μεταξύ δύο κόμβων για να μπορούν να επικοινωνήσουν απευθείας μεταξύ τους είναι ίση με $R$, δηλαδή $p(R) = p_{\max}$. Εάν κάθε κόμβος εκπέμπει με ενέργεια $p_{\max}$, τότε δημιουργείται γράφος $G_R = (V, E)$ , όπου με $V$ συμβολίζεται το πεπερασμένο σύνολο κορυφών (ασύρματων κόμβων) και $E = \{(u,\upsilon) | d(u,\upsilon) \le R\}$ (με $d(u,\upsilon)$ συμβολίζεται η Ευκλείδεια απόσταση μεταξύ $u$ και $\upsilon$).

Ο στόχος γενικά είναι να βρεθεί μέσω του ελέγχου τοπολογίας ένας υπογράφος $G$ του $G_R$, τέτοιος ώστε να περιλαμβάνει όλους τους κόμβους που περιλαμβάνονται στον $G_R$ αλλά με λιγότερες ζεύξεις. Επίσης αν δύο κόμβοι $u$ και $\upsilon$ συνδέονται στον $G_R$ θα πρέπει επίσης να συνδέονται και στον $G$ και τέλος ένας κόμβος $u$ μπορεί να εκπέμπει προς όλους τους γείτονές του στον $G$, χρησιμοποιώντας όμως λιγότερη ενέργεια από όση ήταν απαραίτητη για να εκπέμπει προς όλους τους γείτονές του στον $G_R$.

Το μέγεθος ενός γράφου χαρακτηρίζεται από τον αριθμό των κορυφών και των ακμών.

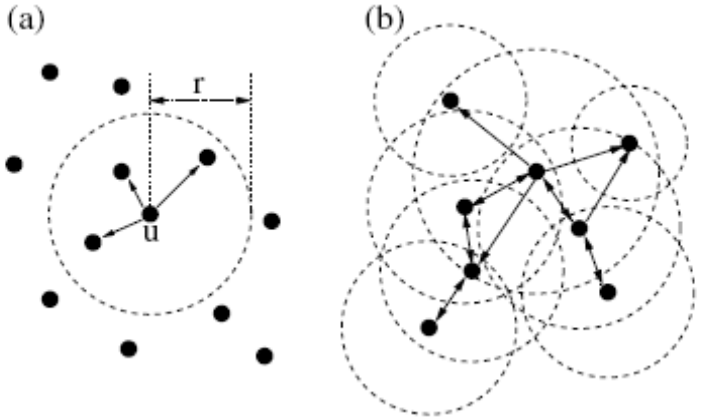
Βαθμός κόμβου είναι ο αριθμός των ζεύξεων που έχουν σχέση με αυτόν. Εάν οι ζεύξεις εισέρχονται στον κόμβο ονομάζεται έσω-βαθμός κόμβου, ενώ εάν εξέρχονται έξω βαθμός-κόμβου.

Ένας μη κατευθυνόμενος γράφος είναι συνδεδεμένος εάν κάποιος μπορεί να μεταβεί από οποιοδήποτε κόμβο σε οποιοδήποτε άλλο ακολουθώντας μια σειρά από ζεύξεις ή εάν όλοι οι κόμβοι ανά δύο συνδέονται μεταξύ τους με κάποιο μονοπάτι.

Ένας κατευθυνόμενος γράφος θεωρείται ισχυρά συνδεδεμένος εάν υπάρχει ένα κατευθυνόμενο μονοπάτι από ένα κόμβο σε οποιοδήποτε άλλο κόμβο.

Για την διατύπωση των αλγορίθμων χρησιμοποιείται το εξής απλοποιημένο μοντέλο: η περιοχή κάλυψης ενός κόμβου είναι ένας κυκλικός δίσκος. Εάν η ακτίνα μετάδοσης του κόμβου $u$ είναι $r$, τότε ο $u$ έχει κατευθυνόμενη ζεύξη προς κάθε

κόμβο του οποίου η Ευκλείδεια απόσταση από τον $u$ είναι μικρότερη ή ίση από το $r$. Ακολουθεί σχηματικό παράδειγμα για την πλήρη κατανόηση του μοντέλου.




**Σχήμα 8. Χρησιμοποιούμενο μοντέλο τοπολογίας**

Στο Σχήμα 8 (a) φαίνεται η ακτίνα μετάδοσης $r$, η περιοχή κάλυψης σε μορφή δίσκου που δημιουργείται και οι κατευθυνόμενες ζεύξεις από τον $u$ στους κόμβους που βρίσκονται εντός της ακτίνας μετάδοσής του. Στο Σχήμα 8 (b) αναπαρίστανται οι περιοχές κάλυψης των ασύρματων κόμβων και οι απευθείας ζεύξεις που δημιουργούνται.

## 3.1 Γωνίες κώνων (cone angles)

### 3.1.1 Ορισμός

Η προσέγγιση με βάση τις γωνίες κώνων βασίζεται στη χρησιμοποίηση των εγγενών γωνιών μεταξύ των ασύρματων συνδέσεων ενός κόμβου ως κριτήριο για ελάττωση της ακτίνας μετάδοσης.

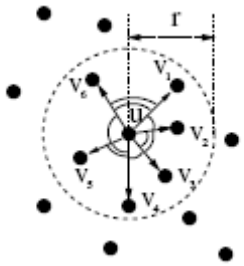
Στο [1] περιγράφονται οι τέσσερις πρώτοι αλγόριθμοι ελέγχου τοπολογίας που βασίζονται στην έννοια των γωνιών κώνων. Το πρόβλημα ελέγχου τοπολογίας ενός ασύρματου δικτύου έγκειται στο να επιλέξουν οι κόμβοι την ακτίνα μετάδοσής τους, η οποία τελικά και θα καθορίσει την τοπολογία του δικτύου δηλαδή τη θέση που θα έχει κάθε κόμβος στο δίκτυο. Η τοπολογία ενός ασύρματου δικτύου είναι βασική για την απόδοσή του. Σχεδόν όλα τα σημαντικά χαρακτηριστικά, όπως για παράδειγμα η συνεκτικότητα και η ικανότητα αποστολής δεδομένων, βασίζονται σε αυτήν.

Το πρόβλημα ελέγχου τοπολογίας που αναφέρθηκε παραπάνω επιδιώκεται μέσω των συγκεκριμένων αλγορίθμων να επιλυθεί τοπικά, δηλαδή κάθε κόμβος αποφασίζει για την ακτίνα μετάδοσής του χρησιμοποιώντας μόνο πληροφορίες από τους γειτονικούς του κόμβους. Με αυτόν τον τρόπο όχι μόνο μειώνεται η πολυπλοκότητα της διαδικασίας αλλά και στην περίπτωση που υπάρξουν αλλαγές στο δίκτυο (π.χ ένας ή περισσότεροι κόμβοι απομακρύνονται), οι κόμβοι μπορούν εύκολα να ανανεώσουν την ακτίνα μετάδοσής τους.

Οι αλγόριθμοι ελέγχου τοπολογίας που θα αναπτυχθούν στη συνέχεια μοιράζονται τα εξής κοινά χαρακτηριστικά: ανάμεσα σε κάθε ζεύγος από κόμβους πηγής και προορισμού υπάρχει ένα σχετικά σύντομο μονοπάτι το οποίο παρουσιάζει μονότονη πρόοδο. Κάθε κόμβος μπορεί να φτάσει άπειρους άλλους κόμβους μέσω κατευθυνόμενων μονοπατιών. Μέσω των αλγορίθμων διαφαίνεται πως τοπικές αποφάσεις σχετικά με την ακτίνα εκπομπής μπορούν να οδηγήσουν σε απότομη

αλλαγή της συμπεριφοράς προς το συμφέρον του δικτύου και τελικά σε αποδοτικές κατασκευές σε σχέση με τον έλεγχο τοπολογίας. Οι αλγόριθμοι χρησιμοποιούν τρία βασικά εργαλεία τα οποία κρίνεται απαραίτητο να αναφερθούν.

Το πρώτο και βασικότερο εργαλείο καλείται γωνία κώνου (cone-angle) το οποίο και προσδιορίζεται ακολούθως. Θεωρούμε ότι ένας κόμβος $u$ έχει $m$ εξερχόμενες ζεύξεις, έστω $v_1, v_2, ...v_m$. Υποθέτουμε επίσης ότι οι ζεύξεις αυτές έχουν ωρολογιακή τοποθέτηση γύρω από τον $u$, όπως φαίνεται και στο Σχήμα 9 (όπου $m = 6$). Σε αυτήν την περίπτωση οι γωνίες $\angle \upsilon_1 u \upsilon_2, \angle \upsilon_2 u \upsilon_3, ..., \angle \upsilon_m u \upsilon_1$ καλούνται γωνίες κώνων του $u$ (Είναι φανερό ότι $\angle \upsilon_1 u \upsilon_2, \angle \upsilon_2 u \upsilon_3, ..., \angle \upsilon_m u \upsilon_1 = 2\pi$). Η λογική των αλγορίθμων λοιπόν συνίσταται στην προσπάθεια κάθε κόμβου ξεχωριστά να καταφέρει να κάνει την ακτίνα του τη μέγιστη δυνατή έτσι ώστε οι γωνίες κώνων του να είναι μικρότερες από ένα συγκεκριμένο κάθε φορά κατώφλι.




**Σχήμα 9. Γωνίες κώνου κόμβου με 6 εξερχόμενες ζεύξεις**

Το δεύτερο εργαλείο είναι η δημιουργία ζεύξεων διπλής κατεύθυνσης μεταξύ των κόμβων. Αυτό σημαίνει ότι όταν υπάρχει κατευθυνόμενη ζεύξη από έναν κόμβο $u$ σε έναν κόμβο $v$, τότε ο $v$ αυξάνει την ακτίνα μετάδοσής του τόσο ώστε να δημιουργηθεί κατευθυνόμενη ζεύξη από τον $v$ στον $u$.

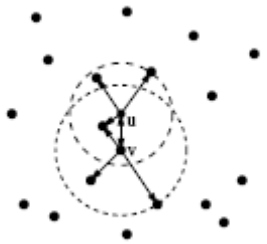
Τέλος το τρίτο εργαλείο θέτει ένα κατώτατο όριο στην ακτίνα μετάδοσης, δηλαδή απαιτείται οι ακτίνες όλων των κόμβων να μην είναι μικρότερες από ένα συγκεκριμένο κατώφλι.

Ουσιαστικά οι τέσσερις πρώτοι αλγόριθμοι αποτελούν ένα συνδυασμό των παραπάνω εργαλείων και προσαρμόζουν τις παραμέτρους τους με αντίστοιχη επίδραση και στα μεγέθη του ασύρματου δικτύου.

### 3.1.2 Αλγόριθμος ελέγχου τοπολογίας I (Algorithm I)

***Αλγόριθμος I:*** Κάθε κόμβος επιλέγει την ακτίνα μετάδοσής στην ελάχιστη δυνατή τιμή σύμφωνα με τον περιορισμό οι γωνίες κώνων του να είναι όλες μικρότερες από $\pi$.

Ένα παράδειγμα του πρώτου αλγορίθμου φαίνεται στο Σχήμα 10. Σε αυτό, παρατηρούμε πως δύο κόμβοι ο $u$ και ο $v$ επιλέγουν την ακτίνα μετάδοσής τους με βάση τον αλγόριθμο I. Οι ακτίνες των δύο κόμβων αναπαρίστανται σαν δύο διακεκομμένοι κύκλοι. Κανένας από τους δύο κόμβους δεν έχει τη δυνατότητα να μειώσει περαιτέρω την ακτίνα του, διότι σε αυτήν την περίπτωση οι γωνίες κώνων τους δεν θα μπορούσαν να είναι όλες μικρότερες από $\pi$.

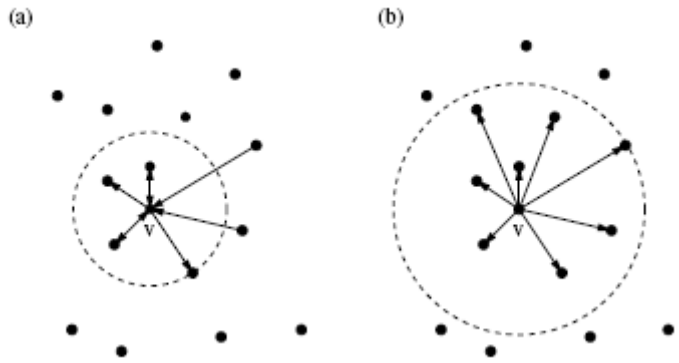



**Σχήμα 10**. **Παράδειγμα επιλογής ακτίνας μετάδοσης με βάση τον αλγόριθμο Ι**

### 3.1.3 Αλγόριθμος ελέγχου τοπολογίας ΙΙ (Algorithm ΙΙ)

***Αλγόριθμος ΙΙ:*** Ο συγκεκριμένος αλγόριθμος αποτελείται από δύο βήματα. Στο πρώτο κάθε κόμβος επιλέγει η ακτίνα μετάδοσής του να έχει την ελάχιστη δυνατή τιμή σύμφωνα με τον περιορισμό οι γωνίες κώνων του να είναι όλες μικρότερες από $\pi$. Στο δεύτερο βήμα για κάθε κόμβο, αν η ακτίνα μετάδοσης επιλέχθηκε να είναι $r$ και αν το μήκος της μακρύτερης εισερχόμενης ζεύξης ήταν $t$ στο πρώτο βήμα, τότε ο κόμβος επιλέγει ως τιμή της ακτίνας μετάδοσής του το μέγιστο μεταξύ $r$ και $t$.

Ένα παράδειγμα του δεύτερου αλγορίθμου φαίνεται στο Σχήμα 11.




**Σχήμα 11. Αποτελέσματα εφαρμογής αλγορίθμου Ι και ΙΙ**

Συγκεκριμένα παρατηρούμε ότι οι κόμβοι στο (a) και (b) έχουν τις ίδιες θέσεις. Εάν υποτεθεί ότι ο αλγόριθμος Ι χρησιμοποιείται για έλεγχο τοπολογίας, όλες οι ζεύξεις που ξεκινούν από τον $v$ και προκύπτουν μετά την εφαρμογή του αλγορίθμου, φαίνονται στο (a). Στην συνέχεια αν χρησιμοποιηθεί ο αλγόριθμος ΙΙ, προκύπτουν οι αντίστοιχες ζεύξεις στο (b).

### 3.1.4 Αλγόριθμος ελέγχου τοπολογίας ΙΙΙ (Algorithm ΙΙΙ)

***Αλγόριθμος ΙΙΙ:*** Σύμφωνα με τον αλγόριθμο ΙΙΙ το κατώφλι για τις γωνίες κώνων επιλέγεται να είναι μικρότερο από $\pi$. Πιο συγκεκριμένα κάθε κόμβος επιλέγει η ακτίνα κάλυψής του να έχει την ελάχιστη δυνατή τιμή σύμφωνα με τον περιορισμό οι γωνίες κώνων του να είναι όλες μικρότερες ή ίσες από παράμετρο $\theta$, για την οποία ισχύει $0 < \theta < 2\pi / 3$.

Η μείωση του κατωφλίου των γωνιών κώνων έχει ως αποτέλεσμα την αύξηση της συνεκτικότητας του δικτύου. Το γεγονός αυτό συνεπάγεται περισσότερες ζεύξεις για το δίκτυο, αλλά και μεγαλύτερες κατά μέσο όρο ακτίνες μετάδοσης..

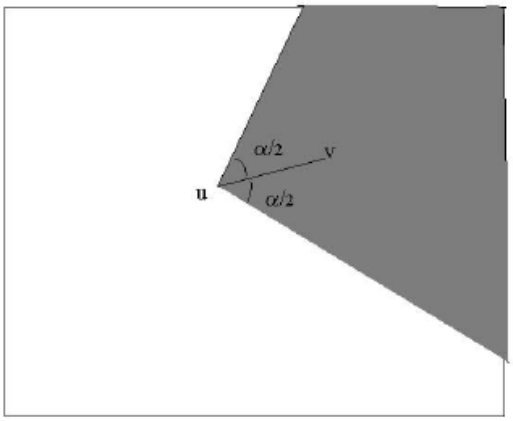
### 3.1.5 Αλγόριθμος ελέγχου τοπολογίας IV (Algorithm IV)

***Αλγόριθμος IV:*** Ο τέταρτος αλγόριθμος θέτει κάτω όριο $\delta$ για την ακτίνα κάλυψης και αποτελείται από δύο βήματα Στο πρώτο κάθε κόμβος επιλέγει η ακτίνα κάλυψής του να έχει την ελάχιστη δυνατή τιμή σύμφωνα με τον περιορισμό οι γωνίες κώνων του να είναι όλες μικρότερες ή ίσες από παράμετρο $\theta$, για την οποία ισχύει $0 < \theta < 2\pi / 3$. Στο δεύτερο βήμα, οι κόμβοι των οποίων η ακτίνα κάλυψης στο πρώτο βήμα είχε προκύψει μικρότερη από $\delta$, θέτουν τελικά την ακτίνα κάλυψής τους να είναι ίση με το κάτω όριο, δηλαδή το $\delta$ ($\delta > 0$).

### 3.1.6 Αλγόριθμος ελέγχου τοπολογίας V (Algorithm V)

Ο αλγόριθμος V όπως παρουσιάστηκε στο [2], βασίζεται και αυτός στην έννοια της γωνίας κώνου και αποτελείται από 2 φάσεις. Στην πρώτη φάση περιγράφεται ένα αποκεντρωμένο σχήμα το οποίο έχει ως αποτέλεσμα τη δημιουργία ενός συνδεδεμένου γράφου, ο οποίος αναφέρεται στο εξεταζόμενο δίκτυο κόμβων και προκύπτει ως εξής: κάθε κόμβος βρίσκει κοντινούς γείτονες κόμβους σε διαφορετικές κατευθύνσεις. Η δεύτερη φάση βελτιώνει την επίδοση με το να απομακρύνονται μη αποδοτικές ζεύξεις στο γράφο επικοινωνίας.

Πιο συγκεκριμένα η πρώτη φάση του αλγορίθμου έχει ως εξής: Κάθε κόμβος $u$ ερευνά με αυξανόμενη ενέργεια $p$. Αρχικά $p = \varepsilon$ ,όπου $\varepsilon$ ένα κατώφλι αρχικής ενέργειας, το οποίο είναι απαραίτητο να τεθεί για την αποφυγή άσκοπης έρευνας σε περιοχές πολύ μικρής ενέργειας. Εάν ο κόμβος $u$ ανακαλύψει έναν νέο γειτονικό κόμβο $v$, θα τον τοποθετήσει στο τοπικό σύνολο γειτόνων του $N(u)$. Ο κόμβος $u$ θα συνεχίσει να αυξάνει την εκπεμπόμενη ενέργειά του μέχρι το σύνολο των γειτόνων του $N(u)$ να είναι αρκετά μεγάλο έτσι ώστε, για κάθε κώνο με γωνία $\alpha$ να υπάρχει τουλάχιστον ένας γείτονας $v$ ο οποίος θα ανήκει στο $N(u)$, ή μέχρι ο κόμβος $u$ να φτάσει την μέγιστη τιμή $P$ που μπορεί να πάρει η ενέργεια εκπομπής. Το κριτήριο τερματισμού μπορεί εύκολα να προσδιοριστεί. Για δεδομένο κόμβο $u$, κάθε γείτονας $v$ που ανήκει στο $N(u)$ καλύπτει ένα κώνο, όπως φαίνεται και στο Σχήμα 12. Εάν οι ένωση των κώνων αυτών καλύπτει γωνία $2\pi$, ο κόμβος $u$ περνάει στην δεύτερη φάση.




**Σχήμα 12. Κώνος γωνίας α**

Για κάθε κόμβο $u$, έστω $p(u)$ η ελάχιστη ενέργεια με την οποία μπορεί βρεθεί ένας γείτονας σε κάθε κώνο με γωνία $\alpha$, ή $p(u) = P$. Σύμφωνα με το στόχο που είναι κάθε φορά επιθυμητός να επιτευχθεί μπορεί να προσδιοριστεί η γωνία $\alpha$ να είναι το πολύ $2\pi/3$ (ορθότητα μόνο) ή το πολύ $\pi/2$ (ανταγωνιστική επίδοση). Εάν ένας κόμβος $u$ με μέγιστη ενέργεια εκπομπής $P$ έχει κώνο $C = [\rho, \rho + \alpha]$ χωρίς κανένα κόμβο που να ανήκει στο $N(u)$, τότε ο κόμβος $u$ θα μειώσει την ενέργεια εκπομπής του στην ελάχιστη τιμή έτσι ώστε να μην υπάρχει κώνος χωρίς ένα γείτονα που έχει γείτονα όταν εκπέμπει με τη μέγιστη ενέργεια $P$.

Ο αλγόριθμος είναι συμμετρικός δηλαδή εάν ο κόμβος $u$ αποφασίσει ότι πρέπει να συμπεριλάβει τον κόμβο $v$ στο σύνολο γειτόνων του τότε και ο κόμβος $v$ πρέπει να συμπεριλάβει τον $u$ στο δικό του σύνολο γειτόνων.

Περαιτέρω βελτίωση στην επίδοση του αλγορίθμου είναι δυνατή και επιτυγχάνεται μέσω της δεύτερης φάσης η οποία έχει ως εξής: Εάν ο κόμβος $u$ έχει δύο γειτονικούς κόμβους $v, w$ οι οποίοι ανήκουν στο $N(u)$, και ισχύει ότι η ενέργεια που απαιτείται για αποστολή απευθείας από τον $u$ στον $w$ δεν είναι μικρότερη από τη συνολική ενέργεια για αποστολή μέσω του $v$, τότε μπορεί να απομακρυνθεί ο $w$ από το σύνολο γειτόνων του $u$. Πιο επίσημα εάν υπάρχουν δύο κόμβοι $v, w,$ οι οποίοι ανήκουν στο $N(u)$, με $w$ να ανήκει στο $N(v)$ και $p(u,v) + p(v,w) \le p(u,w)$ τότε απομακρύνεται ο $w$ από το $N(u)$.

Αυτή η βελτίωση έχει ως αποτέλεσμα να προκύπτουν λιγότεροι γείτονες με ταυτόχρονη διατήρηση των καλύτερων διαδρομών, κάτι που από πλευράς επίδοσης είναι και το πλέον επιθυμητό. Τα βέλτιστα αποτελέσματα επιτυγχάνονται όμως όταν ο αριθμός των γειτόνων περιοριστεί στο ελάχιστο δυνατό. Μία προσέγγιση για την επίτευξη αυτού του στόχου είναι η απομάκρυνση κόμβων από το σύνολο γειτόνων παρά το γεγονός ότι μία απευθείας μετάδοση χρησιμοποιεί λιγότερη ενέργεια από μια έμμεση. Ένας υποψήφιος κόμβος για απομάκρυνση θα ήταν ένας κόμβος $v$, ο οποίος είναι σε μεγάλη απόσταση από τον κόμβο $u$ που στέλνει τα δεδομένα. Ένας επιπλέον λόγος για μια τέτοια απομάκρυνση είναι το γεγονός ότι, όταν ο $u$ εκπέμπει προς τον απομακρυσμένο γείτονα κόμβο του $v$, πολλοί άλλοι κόμβοι υφίστανται παρεμβολές.

Τελικά η ολοκληρωμένη δεύτερη φάση του συγκεκριμένου αλγορίθμου έχει ως εξής: Εάν υπάρχουν δύο κόμβοι $v, w,$ οι οποίοι ανήκουν στο $N(u)$, εάν επίσης ο $w$ ανήκει στο $N(v)$, $p(u,v) \le p(v,w)$ και $p(u,v) + p(v,w) \le q \cdot p(u,w)$, τότε αφαιρείται ο $w$ από το $N(u)$ (και λόγω συμμετρίας επίσης ο $u$ από το $N(w)$). Εάν υπάρχουν περισσότεροι από έναν $v$ κόμβοι, οι οποίοι ικανοποιούν την ανισότητα για τον κόμβο $w$ τότε επιλέγεται τελικά ο κόμβος με το ελάχιστο $p(u,v)$. Μέσω διάσχισης των γειτονικών κόμβων με αυξανόμενη απόσταση από τον κεντρικό εξασφαλίζεται ότι η ζεύξη $(u,v)$ θα παραμείνει. Ουσιαστικά εάν η σταθερά $q$ ισούται 1 αφαιρούνται μόνο οι ζεύξεις που απαιτούν περισσότερη ενέργεια από ότι ένα έμμεσο μονοπάτι.

Γενικότερα ισχύουν οι παρακάτω παρατηρήσεις που αποδεικνύονται στο [2]:

- Εάν $\alpha \le 2\pi/3$, τότε ο γράφος που προκύπτει στο τελικό δίκτυο είναι συνδεδεμένος, εφόσον είναι συνδεδεμένος και όταν όλοι κόμβοι εκπέμπουν με μέγιστη ενέργεια $P$.
- Εάν $\alpha \le \pi/2$ ο τελικός γράφος επιτυγχάνει πολύ καλή κατανάλωση ενέργειας και μάλιστα σε επίπεδα που προσεγγίζεται η ιδανική τιμή της κατανάλωσης.
- Ο βαθμός του κάθε κόμβου μπορεί να περιοριστεί από μια σταθερά για $q \ge 2$.

## 3.2 Περιοχές σκυτάλης(relay regions)

### 3.2.1 Ορισμός

Μία δεύτερη προσέγγιση για την επίτευξη του διπλού στόχου που αναφέρθηκε προηγουμένως, επιδιώκει τον προσδιορισμό ενός υποσυνόλου γειτόνων, στους οποίους είναι πιο οικονομικό να προωθηθεί διαδικτυακή κίνηση, χρησιμοποιώντας καθαρά γεωμετρικά στοιχεία της τοπολογίας του δικτύου και ιδιότητες της ασύρματης διάδοσης κυμάτων. Η προσέγγιση αυτή βασίζεται στην έννοια των περιοχών σκυτάλης και περιγράφεται στο [3].

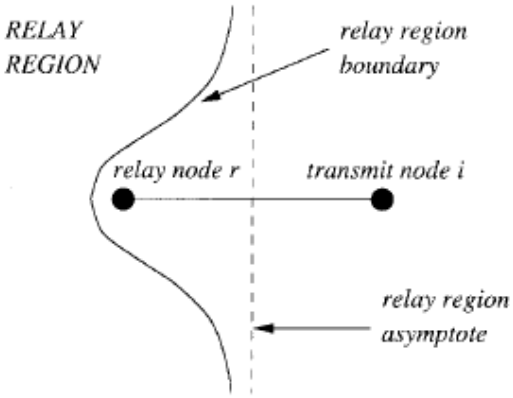
Για να μπορέσει να γίνει κατανοητή η λειτουργία του αλγορίθμου, μέσω του οποίου υλοποιείται η δεύτερη προσέγγιση, κρίνεται απαραίτητο να γίνει μια εισαγωγική αναφορά στις έννοιες και τα μεγέθη που χρησιμοποιούνται.

Στα πλαίσια της εισαγωγικής αυτής αναφοράς θεωρούνται τρεις κόμβοι σε χώρο δύο διαστάσεων, έστω $i, r$ και $j$. Ο κόμβος $i$ επιθυμεί να εκπέμψει πληροφορία στον κόμβο $j$ και χρησιμοποιεί τον κόμβο $r$ ως ενδιάμεσο κόμβο για την εκπομπή αυτή. Στόχος είναι να μεταδοθεί πληροφορία από τον $i$ στον $j$ με μικρότερη κατανάλωση ενέργειας σε σχέση με την απευθείας μετάδοση. Μεταβάλλοντας τη θέση του κόμβου $j$ είναι δυνατόν να εντοπιστεί κάτω από ποιες συνθήκες ο $i$ καταναλώνει λιγότερη ενέργεια για να εκπέμψει μέσω του $r$. Στην παρακάτω ανάλυση η θέση του κόμβου $j$ συμβολίζεται με $(x, y)$.

Περιοχή σκυτάλης (relay region) του ζευγαριού $(i, r)$, η οποία συμβολίζεται με $R_{i\to r}$, καθορίζεται η εξής :

$$R_{i\to r} \equiv \left\{ (x, y) \middle| P_{i\to r\to(x,y)} \langle P_{i\to(x,y)} \right\}$$

όπου με $P_{i\to r\to(x,y)}$ συμβολίζεται η ενέργεια που απαιτείται για την μετάδοση πληροφορίας από τον κόμβο $i$ στη θέση $(x, y)$ μέσω του κόμβου $r$ και με $P_{i\to(x,y)}$ συμβολίζεται η ενέργεια που απαιτείται για τη μετάδοση πληροφορίας από τον $i$ στην θέση $(x, y)$ απευθείας, χωρίς τη μεσολάβηση του $r$. Μία τυπική περιοχή σκυτάλης απεικονίζεται στο Σχήμα 13.




**Σχήμα 13. Τυπική περιοχή σκυτάλης**

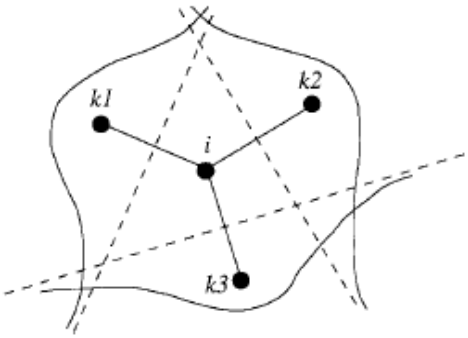
Εάν ο κόμβος $j$ ανήκει στην $R_{i\to r}$ τότε ισχύουν τα παρακάτω:

- $d_{jr} < d_{ij}$
- $d_{ij} > d_{ir}$

όπου με $d_{kl}$ συμβολίζεται η απόσταση μεταξύ $k$ και $l$.

Περιοχή δικτύου (deployment region) ενός συνόλου κόμβων $N$ ονομάζεται κάθε ορισμένο σύνολο σε χώρο δύο διαστάσεων, το οποίο περιέχει τη θέση των κόμβων στο $N$ ως υποσύνολο. Η αναγκαιότητα του ορισμού της περιοχής δικτύου έγκειται πρακτικά στο ότι υπάρχει μια πεπερασμένη περιοχή πέρα από την οποία κανένας κόμβος δεν θα έπρεπε να ερευνά για γείτονες. Επομένως τα όρια της περιοχής δικτύου μπορούν να θεωρηθούν δεδομένα και οι κόμβοι που είναι τοποθετημένοι κοντά στα άκρα και να μην ερευνούν άσκοπα πέρα από τα όρια αυτά.

Μια επίσης πολύ σημαντική για την κατανόηση του αλγορίθμου έννοια είναι η περιοχή εγκλεισμού (enclosure). Η έννοια αυτή μπορεί να περιγραφεί ευκολότερα με το Σχήμα 14 που ακολουθεί.




**Σχήμα 14. Περιοχή εγκλεισμού κόμβου** $i$

Όπως εύκολα μπορεί να γίνει αντιληπτό ο κεντρικός κόμβος $i$ μετά από έρευνα εντόπισε τρεις άλλους κόμβους $(k_1, k_2, k_3)$ πλησίον του. Μετά από υπολογισμό της relay region καθενός από τους τρεις αυτούς κόμβους και το σχεδιασμό τους γύρω από τον i προέκυψε μια ορισμένη περιοχή γύρω από το $i$ η οποία και ονομάζεται περιοχή εγκλεισμού του $i$.

Οι κόμβοι που βρίσκονται εντός της περιοχής εγκλεισμού του $i$ ονομάζονται γείτονες του και είναι οι μόνοι κόμβοι με τους οποίους ο $i$ διατηρεί επικοινωνιακές ζεύξεις έτσι ώστε να μπορέσει να επιτευχθεί ενεργειακά αποδοτική εκπομπή.

Ο γράφος εγκλεισμού ενός συνόλου κόμβων $N$ είναι ο γράφος του οποίου το σύνολο κόμβων είναι το $N$ και το σύνολο ακμών είναι το:

$$\bigcup_{i \in N} \bigcup_{k \in N(i)} l_{i \to k}$$

με $l_{i \to k}$ την απευθείας επικοινωνιακή ζεύξη μεταξύ του $i$ και $k$.

Έπειτα από αυτή την εισαγωγική αναφορά είναι δυνατόν να αναλυθεί περαιτέρω ο αλγόριθμος ο οποίος χρησιμοποιεί τις περιοχές σκυτάλης για την εύρεση του ελάχιστου δυνατού αριθμού γειτόνων.

### 3.2.2 Αλγόριθμος RR (Algorithm RR)

Κάθε κόμβος ερευνά αρχικά και βρίσκει τον γράφο εγκλεισμού και στη συνέχεια διενεργείται κατανομή κόστους από τον κόμβο-σταθμό σε όλους τους υπόλοιπους.

Κάθε κόμβος ενημερώνει τους κόμβους που βρίσκονται σε περιοχή που ορίζεται ως περιοχή αναζήτησης (search region) για τη θέση του. Η περιοχή αναζήτησης ορίζεται ως η περιοχή όπου το εκπεμπόμενο σήμα ενός κόμβου (και επομένως η θέση του) μπορεί να εντοπιστεί από κάθε κόμβο ο οποίος είναι τοποθετημένος σε αυτήν την περιοχή.

Κατά τη διαδικασία έρευνας για εντοπισμό γειτόνων κάθε κόμβος πρέπει να γνωρίζει εάν κάποιος κόμβος που βρέθηκε βρίσκεται στην περιοχή σκυτάλης κόμβων που έχουν ήδη εντοπιστεί κατά την έρευνα. Αυτό είναι δυνατόν να πραγματοποιηθεί με τη βοήθεια του γράφου προώθησης (relay graph).

Αν $A$ θεωρηθεί το σύνολο όλων των κόμβων που ο $i$ έχει εντοπίσει μέχρι κάποια δεδομένη χρονική στιγμή και υπάρχουν δύο κόμβοι $j$ και $k$, οι οποίοι ανήκουν στο Α, τότε όποτε ισχύει ότι ο $k$ ανήκει στην περιοχή σκυτάλης $R_{i\to j}$ $k \in R(j)$), τότε σχηματίζεται απευθείας ζεύξη από το $j$ στο $k$, η οποία συμβολίζεται ως $e_{j\to k}$. Ο γράφος προώθησης ενός κόμβου $i$ ορίζεται ως ο άμεσος γράφος του οποίου το σύνολο κόμβων είναι το $A$ και το σύνολο ακμών το:

$$\bigcup_{j\in A}\bigcup_{k\in R(j)} e_{j\to k}$$

Ο γράφος προώθησης του κόμβου i συμβολίζεται με $G(i)$.

Είναι σημαντικό να επισημανθεί ότι μέσω του $e_{j\to k}$ αναπαρίσταται η σχέση μεταξύ του $j$ και $k$ σε σχέση με τις θέσεις τους. Υποδηλώνεται ότι ο $k$ βρίσκεται στην περιοχή σκυτάλης $R_{i\to j}$. Σε καμία περίπτωση δεν αναπαριστά επικοινωνιακή ζεύξη μεταξύ του $j$ και $k$.

Στην συνέχεια θα γίνει αναλυτική περιγραφή του αλγορίθμου αναζήτησης ο οποίος εκτελείται από κάθε κόμβο και υπολογίζει το $N(i)$, δηλαδή το σύνολο γειτόνων του κόμβου $i$.

Αρχικά κάθε κόμβος εκπέμπει ένα σήμα που περιλαμβάνει πληροφορίες σχετικές με τη θέση του. Ο κάθε κόμβος εκτελεί ακριβώς τον ίδιο αλγόριθμο επομένως η περιγραφή μπορεί να περιοριστεί σε ένα μόνο κόμβο του δικτύου ο οποίος και θα αποκαλείται κόμβος πομπός. Ο κόμβος αυτός δέχεται επίσης σήματα από γειτονικούς κόμβους. Με τη λήψη και αποκωδικοποίηση των σημάτων αυτών ενημερώνεται για τις θέσεις των γειτονικών κόμβων και υπολογίζει τις περιοχές σκυτάλης τους. Όπως έχει ήδη αναλυθεί ο κόμβος πομπός πρέπει να διατηρήσει μόνο τους κόμβους που δεν βρίσκονται στις περιοχές σκυτάλης κόμβων που έχουν ήδη εξεταστεί. Επομένως κάθε φορά που νέοι κόμβοι ανευρίσκονται, πρέπει ο κόμβος πομπός να ανανεώνει τον γράφο προώθησής του.

Οι κόμβοι που έχουν βρεθεί μέχρι κάποιο στιγμιότυπο του αλγορίθμου μπορούν άρα να χωριστούν σε δύο κατηγορίες. Πιο συγκεκριμένα εάν ένας κόμβος που έχει βρεθεί (έστω $k$) βρίσκεται στην περιοχή σκυτάλης ενός άλλου κόμβου που επίσης έχει βρεθεί (έστω $j$), έχει ως αποτέλεσμα να δηλωθεί ο $k$ ως "νεκρός κόμβος" (dead). Θεωρείται δηλαδή ότι ο $j$ "μπλοκάρει" τον $k$. Με αυτόν τον τρόπο δηλώνεται η κατάσταση κάθε κόμβου. Εάν ένας κόμβος δεν μπλοκάρεται από κανέναν άλλο κόμβο που έχει βρεθεί μέχρι εκείνη τη στιγμή κατά την αναζήτηση, τότε ο κόμβος αυτός δηλώνεται ως "ζωντανός" (alive). Το σύνολο των ζωντανών

κόμβων που προκύπτουν μετά το τέλος της αναζήτησης συνιστά το σύνολο των γειτόνων του κόμβου πομπού $i$. Ως αποτέλεσμα, όταν τερματιστεί η αναζήτηση ο κόμβος πομπός έχει εγκλειστεί και οι κόμβοι που δημιουργούν αυτήν την περιοχή εγκλεισμού δεν βρίσκονται στην περιοχή σκυτάλης κανενός άλλου ήδη εξετασθέντα κόμβου.

Στην συνέχεια παρατίθεται η βασική δομή του αλγορίθμου σε ψευδοκώδικα ο οποίος θα σχολιαστεί περαιτέρω.

$F = 0;$
$A = 0;$
$AliveNodes = 0;$
$DeadNodes = 0;$
$while(TRUE)\{$
$\quad S = SetSearchRegion(\ );$
$\quad F = F \cup S;$
$\quad M = \{n \mid (x_n, y_n) \in S, n \notin A, n \neq i\};$
$\quad A = A \cup M;$
$\quad foreach(m \in M)\{$
$\quad\quad MarkDead(m);$
$\quad\quad foreach(n \in A)\{$
$\quad\quad\quad if\ (n \in R(m))\ DrawEdge(m \to n);$
$\quad\quad\quad elseif\ (m \in R(n))\ DrawEdge(n \to m);$
$\quad\quad\quad \}$
$\quad \}$
$\quad foreach(m \in M)\{$
$\quad\quad FlipAllStatesDownChain(m);$
$\quad \}$
$\quad \eta = \bigcup_{k \in AliveNodes} R^c_{i \to k} \cap D_N;$
$\quad if\ (F \supset \eta)\ break;$
$\}$
$N(i) = AliveNodes;$
$\varepsilon_i = \eta;$

```
voidFlipAllStatesDownChain(m){
    if (m ∈ AliveNodes){
        MarkDead(m);
        foreach(k ∈ R(m)) FlipAllStatesDownChain(k);
    }
    elseif (m ∉ R(k) ∀k ∈ AliveNodes){
        MarkAlive(m);
        foreach(k ∈ R(m)) FlipAllStatesDownChain(k);
    }
}
```

**Σχήμα 15. Αλγόριθμος αναζήτησης και συνάρτηση `FlipAllStatesDownChain`**

Στο επίπεδο της υλοποίησης του αλγορίθμου είναι απαραίτητη η χρησιμοποίηση βοηθητικής συνάρτησης με το όνομα `FlipAllStatesDownChain` μέσω της οποίας θα ανανεώνεται ο γράφος προώθησης. Η συνάρτηση είναι αναγκαία για να αντιμετωπιστεί η ακόλουθη κατάσταση: έστω ότι σε κάποιο σημείο του αλγορίθμου ένας κόμβος, με την ετικέτα $k$, μπλοκάρεται μόνο από έναν κόμβο $j$. Στη συνέχεια στην επόμενη επανάληψη, υποθέτουμε ότι ένας νέος κόμβος $p$ μπλοκάρει τον $j$ αλλά όχι τον $k$. Σε αυτήν την περίπτωση ο $k$ θα πρέπει να θεωρηθεί ξανά ζωντανός

Στον παραπάνω αλγόριθμο με $D_N$ δηλώνεται η ορισμένη περιοχή δικτύου, με $A$ δηλώνονται όλοι οι κόμβοι που έχει βρει το $i$ μέχρι τώρα στην έρευνά του, με $M$ δηλώνονται οι καινούριοι κόμβοι που βρίσκονται στην τρέχουσα επανάληψη, με $S$ δηλώνεται η τρέχουσα περιοχή έρευνας και με $F$ δηλώνεται όλη η περιοχή που έχει ερευνηθεί μέχρι τώρα.

Η κατάσταση κάθε κόμβου μπορεί να πάρει μόνο τρεις τιμές: ο κόμβος μπορεί να είναι νεκρός, ζωντανός ή ακόμα "αμαρκάριστος" από τον αλγόριθμο. Τα σύνολα `AliveNodes` και `DeadNodes` δηλώνουν το σύνολο των ζωντανών κόμβων στο $A$ και το σύνολο των νεκρών κόμβων στο $A$ αντίστοιχα. Η συνάρτηση `MarkAlive(m)` δηλώνει την κατάσταση του κόμβου m ως ζωντανή ενώ η συνάρτηση `MarkDead(m)` δηλώνει την κατάστασή του ως νεκρή.

Η συνάρτηση `DrawEdge(m→n)` δημιουργεί τη ζεύξη $e_{m\to n}$ στον γράφο προώθησης του $i$. Τέλος η συνάρτηση `SetSearchRegion` ορίζει την περιοχή έρευνας σε κάθε επανάληψη η οποία εξαρτάται από τους κόμβους που έχουν εξεταστεί μέχρι τώρα και τους εναπομείναντες για να εξεταστούν. Ο τερματισμός του αλγορίθμου εξαρτάται από την επιλογή της περιοχής έρευνας. Είναι πάντα δυνατό να τερματιστεί ο αλγόριθμος θέτοντας $S = D_N$.

Στο δεύτερο μέρος του αλγορίθμου υπολογίζονται οι ιδανικές ζεύξεις με βάση τον υπολογισμό του ελάχιστου κόστους που απαιτείται για κάθε κόμβου έτσι ώστε να μπορέσει να στείλει το σήμα του στον κεντρικό σταθμό. Η παρούσα διπλωματική δεν ασχολείται με το δεύτερο μέρος του αλγορίθμου.

.

εφόσον δεν μπλοκάρεται πλέον από κανέναν κόμβο. Παρομοίως μπορεί να υπάρχει μία ολόκληρη αλυσίδα από κόμβους (π.χ ένα μονοπάτι στον γράφο προώθησης),

όπου ένας κόμβος μπλοκάρει τον επόμενο στην αλυσίδα. Εάν βρεθεί ένας καινούριος κόμβος, ο οποίος μπλοκάρει τον πρώτο ζωντανό κόμβο στην αλυσίδα, τότε πρέπει οι καταστάσεις όλων των κόμβων στην αλυσίδα να επανεξεταστούν. Αυτήν ακριβώς την κατάσταση διαχειρίζεται η συνάρτηση `FlipAllStatesDownChain`.

# Κεφάλαιο 4 – Υλοποίηση αλγορίθμων

Η υλοποίηση των αλγορίθμων έγινε σε γλώσσα προγραμματισμού C++ με τη βοήθεια του πακέτου ανάπτυξης εφαρμογών Microsoft Visual C++ 6.0. Για τη μεταφορά κάθε επιμέρους αλγορίθμου σε προγραμματιστικό περιβάλλον απαιτήθηκαν ειδικοί τρόποι αντιμετώπισης σε προβλήματα που ανέκυψαν αλλά και συγκεκριμένοι τρόποι προσέγγισης για να δημιουργηθεί το αντίστοιχο σε σχέση με κάθε αλγόριθμο τελικό δίκτυο χωρίς να υπάρξουν σημαντικές αποκλίσεις από την εκάστοτε θεωρητική διατύπωση.

Αρχικά θα γίνει μια αναφορά στις μεθόδους που χρησιμοποιήθηκαν στο σύνολο των αλγορίθμων, στη μορφή του δικτύου που δημιουργήθηκε αλλά και στα γενικότερα προβλήματα που εμφανίστηκαν.

Κρίθηκε απαραίτητη η δημιουργία μίας κλάσης με το όνομα `Tools` η οποία και περιέχει όλες τις απαραίτητες βοηθητικές συναρτήσεις για την υλοποίηση του προγράμματος. Οι πιο σημαντικές από αυτές τις βοηθητικές συναρτήσεις είναι η συνάρτηση που υπολογίζει τη μέγιστη γωνία κώνου ενός κόμβου και η συνάρτηση υπολογισμού του πίνακα γειτονίας. Οι δύο συναρτήσεις χρησιμοποιήθηκαν και στους πέντε πρώτους αλγορίθμους και η λειτουργία τους θα αναλυθεί στη συνέχεια. Κάποιες γενικότερες συναρτήσεις που υλοποιήθηκαν και θα μπορούσαν να επίσης αναφερθούν είναι οι εξής: συνάρτηση ταξινόμησης στοιχείων πίνακα κατά αύξουσα σειρά, συνάρτηση υπολογισμού απόστασης δύο κόμβων, συνάρτηση εύρεσης συνεκτικών υποτμημάτων (components) του δικτύου, συνάρτηση υπολογισμού μέγιστου και ελάχιστου στοιχείου πίνακα, συνάρτηση εύρεσης αριθμού γειτόνων, συνάρτηση εύρεσης μέσης ακτίνας, μέσου αριθμού γειτόνων, διασποράς ακτίνας και διασποράς γειτόνων. Ειδικότερα στον αλγόριθμο VI χρησιμοποιήθηκαν και άλλες συναρτήσεις που θα αναφερθούν όμως στη συνέχεια, στην αναλυτική περιγραφή της υλοποίησής του.

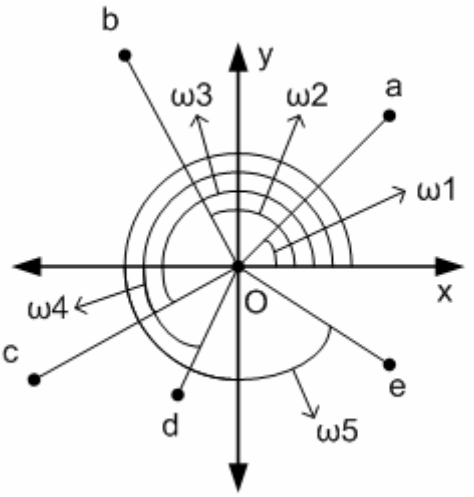
Σχετικά με τη μορφή του δικτύου, χρησιμοποιήθηκε η ίδια βασική δομή και στους έξι αλγορίθμους. Πιο συγκεκριμένα η θεωρητική διατύπωση των πέντε πρώτων αλγορίθμων αναφέρεται σε δίκτυο με άπειρες διαστάσεις κάτι που πρακτικά είναι αδύνατο να εφαρμοστεί. Προτιμήθηκε η δημιουργία μίας τετραγωνικής περιοχής δικτύου εντός του οποίου τοποθετήθηκαν κόμβοι με τυχαίες συντεταγμένες και στους οποίους εφαρμόστηκε τελικά ο αλγόριθμος. Αντίθετα ο αλγόριθμος VI περιγράφεται και θεωρητικά για πεπερασμένη περιοχή, οπότε η παραπάνω προσέγγιση ισχύει και σε αυτήν την περίπτωση.

Ένα σημαντικό πρακτικό πρόβλημα που προέκυψε σε όλους τους αλγορίθμους ήταν το γεγονός ότι δεν ήταν δυνατόν να τρέξουν από έναν αριθμό κόμβων και πάνω. Εντοπίστηκε τελικά ότι ήταν απαραίτητη η ρύθμιση μίας από τις παραμέτρους του μεταγλωττιστή (compiler), το λεγόμενο μέγεθος στοίβας (stack size). Η παράμετρος αυτή ορίζει το μέγιστο μέγεθος πινάκων που είναι δυνατόν να δημιουργηθούν από κάποιο πρόγραμμα. Στα προγράμματα που υλοποιήθηκαν, από κάποιον αριθμό κόμβων και άνω, το μέγεθος των πινάκων που έπρεπε να δημιουργηθούν ξεπερνούσε κατά πολύ την προκαθορισμένη από τον compiler τιμή μέγιστου μεγέθους πίνακα. Ήταν λοιπόν απαραίτητο η παράμετρος από το 1MByte που ήταν προκαθορισμένη να τεθεί στα 8 MBytes για να μην αντιμετωπιστεί πρόβλημα μεγέθους για όλες τις τιμές αριθμού κόμβων που έπρεπε να εξεταστούν.

Ακολουθεί αναλυτική αναφορά στις μεθόδους που χρησιμοποιήθηκαν σε κάθε αλγόριθμο ξεχωριστά. Ο πηγαίος κώδικας όλων των αλγορίθμων έχει τοποθετηθεί στο παράρτημα.

## 4.1 Αλγόριθμος Ι

Όπως ήδη έχει αναφερθεί, η βοηθητική συνάρτηση πάνω στην οποία βασίστηκε ουσιαστικά η υλοποίηση, ήταν η συνάρτηση υπολογισμού της γωνίας κώνου.




**Σχήμα 16. Βοηθητικό σχήμα για την υλοποίηση της γωνίας κώνου**

Έστω ένας κεντρικός κόμβος Ο με πέντε γειτονικούς κόμβους οι οποίοι είναι διατεταγμένοι γύρω του όπως φαίνεται στο Σχήμα 16. Οι γωνίες κώνου του κόμβου Ο είναι οι γωνίες $a\widehat{O}b, b\widehat{O}c, c\widehat{O}d, d\widehat{O}e$ όπως έχουν περιγραφεί στο προηγούμενο κεφάλαιο. Ακολουθείται η εξής τακτική: υπολογίζονται αρχικά οι γωνίες που σχηματίζονται ανάμεσα σε κάθε κόμβο γείτονα και τον x-άξονα, θεωρώντας ως αρχή των συντεταγμένων τον κόμβο Ο και υπολογίζοντας μέσω μετασχηματισμού συντεταγμένων τη θέση και τις συντεταγμένες των γειτόνων ως προς τον κεντρικό κόμβο Ο. Εφόσον υπολογιστούν οι γωνίες $\omega_1, \omega_2, ...\omega_5$ τοποθετούνται κατά αύξουσα σειρά και στη συνέχεια γίνεται αφαίρεση της κάθε γωνίας από την προηγούμενη της. Κατά αυτόν τον τρόπο προκύπτουν όλες οι γωνίες κώνων εκτός από αυτήν που σχηματίζεται στην συγκεκριμένη περίπτωση ανάμεσα στον τελευταίο και τον πρώτο γείτονα. Για την εύρεση της συγκεκριμένης γωνίας απαιτήθηκε ειδικός τύπος ο οποίος και δίνεται στο παράρτημα στον κώδικα του αλγορίθμου Ι.

Μια επίσης εξαιρετικά χρήσιμη βοηθητική συνάρτηση που υλοποιήθηκε ήταν η συνάρτηση εύρεσης του πίνακα γειτονίας. Ο πίνακας γειτονίας θεωρήθηκε ένας πίνακας [N,N], με N των αριθμό των κόμβων του δικτύου, μέσω του οποίου μπορεί να δηλωθεί ποιοι είναι οι γείτονες ενός κόμβου. Η συνάρτηση που υπολογίζει τον πίνακα κάνει ουσιαστικά τον εξής έλεγχο: με βάση τη δεδομένη ακτίνα που προκύπτει κάθε φορά για τον εκάστοτε κόμβο βρίσκει τους κόμβους, των οποίων η απόστασή τους από τον κεντρικό είναι μικρότερη ή ίση της ακτίνας του και όταν τους εντοπίσει τους ονομάζει γείτονες.

Η επιλογή πεπερασμένου χώρου για την δημιουργία του δικτύου ήταν επιβεβλημένη. Ωστόσο δημιούργησε ένα πολύ σημαντικό πρακτικό πρόβλημα το οποίο και έπρεπε να αντιμετωπιστεί. Συγκεκριμένα οι κόμβοι που προέκυψαν κοντά στα άκρα της περιοχής δεν ήταν δυνατόν να μπορέσουν να ικανοποιήσουν ποτέ τον αλγόριθμο διότι δεν υπήρχαν αρκετοί γείτονες, από όλες τις πλευρές διατεταγμένοι έτσι ώστε να προκύψουν κάποια στιγμή όλες οι γωνίες κώνων μικρότερες από $\pi$. Επομένως εάν ο αλγόριθμος εφαρμοζόταν σε όλους τους κόμβους, θα υπήρχε αέναη

προσπάθεια για την επαλήθευση της συνθήκης στην περίπτωση που εξεταζόταν κάποιος κόμβος στα άκρα. Στο πρόγραμμα οι κόμβοι αυτού του είδους αλλά και εκείνοι που λόγω της διάταξής τους δεν επέτρεπαν την ικανοποίηση της συνθήκης, ονομάστηκαν edge nodes και η μεταχείρισή τους είναι η εξής: Η προσπάθεια επαλήθευσης της συνθήκης του αλγορίθμου ξεκινά από μία μέγιστη τιμή ακτίνας στην οποία και υπολογίζεται η μέγιστη γωνία κώνου. Κανονικά εφόσον η ακτίνα έχει μία σχετικά μεγάλη τιμή συνεπάγεται αυτόματα ότι και οι γείτονες έχουν σχετικά μεγάλη τιμή και επομένως όλες οι γωνίες κόμβων πρέπει να είναι μικρότερες του $\pi$. Θεωρείται επομένως ότι εάν σε αυτή τη μεγάλη τιμή ακτίνας προκύψει ότι η μέγιστη γωνία κώνου ξεπερνά τις 180° πρόκειται για ένα κόμβο ο οποίος και δεν θα μπορέσει να ικανοποιήσει ποτέ τον αλγόριθμο. Σε αυτήν την περίπτωση δεν εφαρμόζεται ο αλγόριθμος, σημειώνεται ο κόμβος ως edge node και το πρόγραμμα εξετάζει τον επόμενο κόμβο.

Στην περίπτωση που δεν εξετάζεται edge node ξεκινά μια σταδιακή μείωση της ακτίνας με σκοπό να εντοπιστεί η ελάχιστη ακτίνα με την οποία μπορεί να εξασφαλιστεί ότι όλες οι γωνίες κώνων θα έχουν τιμή μικρότερη ή ίση του $\pi$. Με βάση κάθε τιμή ακτίνας που εξετάζεται υπολογίζεται ο πίνακας γειτονίας του κόμβου που θεωρείται τη δεδομένη στιγμή ως κεντρικός και στη συνέχεια με βάση αυτόν τον πίνακα η μέγιστη γωνία κώνου. Η τιμή αυτή ελέγχεται και αν βρεθεί μικρότερη του π η αναζήτηση συνεχίζεται. Η αναζήτηση και η περαιτέρω μείωση της ακτίνας σταματά μόνο όταν βρεθεί μέγιστη γωνία κώνου μεγαλύτερη του $\pi$ οπότε και η τιμή της ακτίνας επιστρέφει στην αμέσως προηγούμενη τιμή που είχε πριν το τελευταίο βήμα. Αυτό είναι απαραίτητο διότι με αυτόν τον τρόπο εντοπίζεται η ελάχιστη τιμή ακτίνας που μπορεί να ικανοποιήσει τη συνθήκη του αλγορίθμου.

Τα αποτελέσματα από την εφαρμογή του αλγορίθμου, τα χαρακτηριστικά δηλαδή του τελικού δικτύου, αποθηκεύονται σε αρχείο για περαιτέρω επεξεργασία. Τα χαρακτηριστικά αυτά είναι ο αύξων αριθμός κάθε κόμβου, οι συντεταγμένες του, η ακτίνα του, ο αριθμός των γειτόνων του, η μέση ακτίνα του δικτύου, ο μέσος αριθμός γειτόνων του δικτύου, η διασπορά της μέσης ακτίνας και η διασπορά του μέσου αριθμού γειτόνων. Αντίστοιχο αρχείο δημιουργείται και για το αρχικό δίκτυο, πριν την εφαρμογή του αλγορίθμου. Η διαδικασία της δημιουργίας των αρχείων στατιστικών δεδομένων ακολουθείται και στους υπόλοιπους αλγόριθμους. Για τον υπολογισμό της μέσης τιμής των εκάστοτε μεγεθών χρησιμοποιήθηκε ο τύπος:

$$\overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$$

και για τη διασπορά ο τύπος:

$$\sigma_{\overline{X}} = \sqrt{\frac{\sum_{i=1}^{n}(X_i - \overline{X})^2}{n(n-1)}}\ .$$

Οι παραπάνω τύποι παρουσιάζονται στο [11].

Στο τελικό δίκτυο οι κόμβοι που χαρακτηρίστηκαν τελικά ως edge nodes θεωρούνται ότι έχουν ακτίνα ίση με τη μέγιστη και επίσης ότι ο αριθμός γειτόνων τους είναι αυτός που αντιστοιχεί στη μέγιστη ακτίνα. Αυτή η διευκρίνιση είναι επιβεβλημένη διότι τελικά ο αριθμός των edge nodes επηρεάζει τις μέσες τιμές των

χαρακτηριστικών μεγεθών του δικτύου, όπως και τις τιμές της διασποράς. Αυτό θα γίνει κατανοητό στο επόμενο κεφάλαιο κατά την παράθεση των αποτελεσμάτων.

## 4.2 Αλγόριθμος II

Ο αλγόριθμος II, όπως έχει ήδη αναφερθεί σε προηγούμενο κεφάλαιο, αποτελείται από δύο μέρη. Στο πρώτο μέρος υλοποιείται ουσιαστικά ο αλγόριθμος I και στο δεύτερο επιτελείται διαδικασία επιπλέον σύγκρισης για τον υπολογισμό της τελικής ακτίνας. Πιο συγκεκριμένα δεδομένης της ακτίνας που έχει υπολογιστεί από τον αλγόριθμο I, γίνεται υπολογισμός του μήκους των ζεύξεων που κατευθύνονται προς τον κόμβο που εξετάζεται. Εντοπίζεται η μακρύτερη ζεύξη και τελικά ακτίνα του κόμβου ορίζεται το μέγιστο μεταξύ της αρχικής ακτίνας που υπολογίστηκε και της μακρύτερης ζεύξης που εντοπίστηκε.

Οι edge nodes αντιμετωπίστηκαν με τον ίδιο ακριβώς τρόπο και η μορφή των αρχείων με τα δεδομένα αρχικού και τελικού δικτύου ήταν ίδια με αυτή που προαναφέρθηκε.

## 4.3 Αλγόριθμος III

Σύμφωνα με τη θεωρητική διατύπωση του αλγορίθμου III η υλοποίηση έγκειται στην εύρεση της ελάχιστης δυνατής ακτίνας για την οποία όλες οι γωνίες κώνων είναι μικρότερες ή ίσες από κατώφλι $\theta$, με $0 < \theta < 2\pi/3$. Για τον προγραμματισμό του αλγορίθμου χρησιμοποιήθηκε η βασική δομή του αλγορίθμου I με τη διαφορά ότι επιλέχθηκε το κατώφλι ελέγχου να είναι το μέγιστο δυνατό, δηλαδή πρακτικά να υπάρχει μείωση της ακτίνας μέχρι να εντοπιστεί αυτή η τιμή ακτίνας για την οποία η μέγιστη γωνία κώνου είναι μεγαλύτερη ή ίση από $2\pi/3$. Σε αυτή την περίπτωση αυξάνεται η ακτίνα στην αμέσως προηγούμενη τιμή και ουσιαστικά ακολουθείται η ίδια διαδικασία με τον I.

Αντιστοίχως με τους δύο προηγούμενους αντιμετωπίζονται οι edge nodes και δημιουργούνται τα αρχεία αρχικού και τελικού δικτύου.

## 4.4 Αλγόριθμος IV

Ο αλγόριθμος IV αποτελείται όπως και ο II από δύο επιμέρους τμήματα. Στο πρώτο υλοποιείται ο αλγόριθμος III και στο δεύτερο συγκρίνεται η ακτίνα που έχει προκύψει από το πρώτο μέρος με κατώφλι $\delta$. Αν είναι μικρότερη του $\delta$, τότε η τιμή της γίνεται ίση με αυτήν του κατωφλίου. Στην ουσία ο αλγόριθμος δεν επιτρέπει την ύπαρξη ακτίνας μικρότερης από μία προκαθορισμένη τιμή. Προγραμματιστικά βασίζεται στον III, ο οποίος με την σειρά του βασίζεται στον I .

## 4.5 Αλγόριθμος V

Ο αλγόριθμος V ο οποίος περιγράφεται στο [2] αποτελείται από δύο ξεχωριστές φάσεις. Στην πρώτη φάση ο κάθε κόμβος αυξάνει συνεχώς την εκπεμπόμενη ενέργειά του, δηλαδή πρακτικά την ακτίνα του, μέχρι το σύνολο των γειτόνων που έχει εντοπίσει έως κάποια δεδομένη στιγμή να είναι αρκετά μεγάλο, έτσι ώστε για κάθε κώνο με γωνία $\alpha$ να υπάρχει τουλάχιστον ένας γείτονας ή μέχρι ο κόμβος να φτάσει την μέγιστη τιμή $P$ που μπορεί να πάρει η ενέργεια εκπομπής(δηλαδή τη μέγιστη ακτίνα).

Ξεκάθαρο είναι στο [2] ότι οι τιμές της γωνίας $\alpha$ που πρέπει να χρησιμοποιηθούν είναι 90° και 120°. Η προσέγγιση που ακολουθήθηκε ήταν η εξής: Δημιουργήθηκαν δύο αλγόριθμοι μέσω των οποίων υλοποιήθηκε η φάση 1 του αλγορίθμου V για γωνίες $\alpha$ των 120° και 90° αντίστοιχα. Στην συνέχεια δημιουργήθηκαν επιπλέον δύο αλγόριθμοι μέσω των οποίων υλοποιήθηκε ο ολοκληρωμένος αλγόριθμος V, συμπεριλαμβανομένης και της φάσης 2, πάλι για γωνίες $\alpha$ 120° και 90° αντίστοιχα.

Αρχικά θα γίνει μία περιγραφή της υλοποίησης της φάσης 1, η οποία ήταν ακριβώς η ίδια και για τις δύο εξεταζόμενες τιμές της γωνίας $\alpha$.

Μια πολύ σημαντική λεπτομέρεια σχετική με τον τρόπο αντιμετώπισης που ακολουθήθηκε είναι ότι επιλέχθηκε ο πιο αυστηρός περιορισμός για τον καθορισμό του τελικού δικτύου. Πιο συγκεκριμένα κάθε κόμβος ξεκινά από μία ελάχιστη τιμή ακτίνας, την οποία και αυξάνει με σταθερό βήμα. Για κάθε τιμή ακτίνας που προκύπτει υπολογίζεται ο πίνακας γειτονίας (ο ρόλος του είναι ο ίδιος με αυτόν που είχε και στους προηγούμενους αλγορίθμους) και στη συνέχεια η μέγιστη γωνία κώνου που προκύπτει σύμφωνα με αυτόν τον πίνακα. Όσο η μέγιστη γωνία κώνου ξεπερνά τις 90° (120° αντίστοιχα), η ακτίνα συνεχίζει να αυξάνεται. Όταν η τιμή της μέγιστης γωνίας κώνου γίνει ίση με 90° (120°) τότε αυτομάτως συνεπάγεται ότι στο δίκτυο που είχε προκύψει με την αμέσως προηγούμενη ακτίνα υπάρχει ένας γείτονας για κάθε κώνο με γωνία α αλλά όχι παραπάνω, όπως θα επέτρεπε ο αλγόριθμος. Δηλαδή τελικά στο δίκτυο που δημιουργείται οι κόμβοι έχουν τον ελάχιστο αριθμό γειτόνων που απαιτούνται για να ικανοποιηθεί η συνθήκη του αλγορίθμου.

Επομένως ο αλγόριθμος σταματά όταν η μέγιστη γωνία κώνου γίνει 90° (120°) αλλά και όταν η ακτίνα πάρει τη μέγιστη τιμή της, κάτι που δηλώνεται ρητά και από το [2]. Μετά τον αρχικό υπολογισμό των γειτόνων κάθε κόμβου γίνεται εφαρμογή και της συμμετρικότητας που αποτελεί βασικό χαρακτηριστικό του αλγορίθμου τόσο στη φάση 1 όσο και στη 2. Ως συνέπεια εάν ένας κόμβος $u$ έχει βρει κάποιους συγκεκριμένους κόμβους ως γείτονες, τότε καθορίζεται ότι και οι κόμβοι αυτοί ονομάζουν τον u ως γείτονά τους. Με αυτό τον τρόπο προκύπτει το τελικό δίκτυο από την πρώτη φάση του αλγορίθμου. Οι κόμβοι που βρίσκονται στα άκρα αντιμετωπίζονται όπως στους προηγούμενους αλγορίθμους.

Στη συνέχεια αναλύεται η υλοποίηση της δεύτερης φάσης η οποία συνοψίζεται στα εξής(σύμφωνα πάντα και με τη θεωρητική ανάλυση που έχει προηγηθεί): Εάν υπάρχουν δύο κόμβοι $v, w,$ οι οποίοι είναι γείτονες του εκάστοτε κόμβου $u$, εάν επίσης ο $w$ είναι γείτονας του $v$ και $d(u,v) \le d(v,w)$ και $d(u,v) + d(v,w) \le q \cdot d(u,w)$ τότε ο $w$ αφαιρείται από το σύνολο γειτόνων του $u$ (και λόγω συμμετρίας επίσης ο $u$ από το σύνολο γειτόνων του $w$). Στις παραπάνω ανισότητες με $d(x,y)$ συμβολίζεται η απόσταση μεταξύ των κόμβων $x$ και $y$ και με $q$ συμβολίζεται σταθερά μεγαλύτερη ή ίση του1. Εάν υπάρχουν περισσότεροι από έναν $v$ κόμβοι, οι οποίοι ικανοποιούν την ανισότητα για τον κόμβο $w$ τότε επιλέγεται τελικά ο κόμβος με το ελάχιστο $d(u,v)$. Οι υπόλοιποι που απορρίπτονται σε αυτήν την περίπτωση και αφαιρούνται στην ουσία από το σύνολο γειτόνων του $u$ υπακούουν επίσης στην αρχή της συμμετρικότητας. Βγάζουν επομένως και αυτοί τον $u$ από το δικό τους σύνολο γειτόνων. Έμφαση πρέπει να δοθεί στο γεγονός ότι οι γειτονικοί κόμβοι διασχίζονται με σειρά αυξανόμενης απόστασης από τον εκάστοτε κεντρικό τους έτσι ώστε να διασφαλιστεί ότι η ζεύξη $(u,v)$ θα παραμείνει.

Με συνδυασμό επομένως της φάσης 1 και 2 υλοποιείται ο συνολικός αλγόριθμος V και μάλιστα σε δύο εκδοχές: μια με γωνία α 120° και μία με γωνία $\alpha$ 90°. Στις δύο τελευταίες προσεγγίσεις είναι απαραίτητο να τονιστεί μία πολύ σημαντική διαφοροποίηση από όλες τις προηγούμενες σε σχέση με την αντιμετώπιση των edge

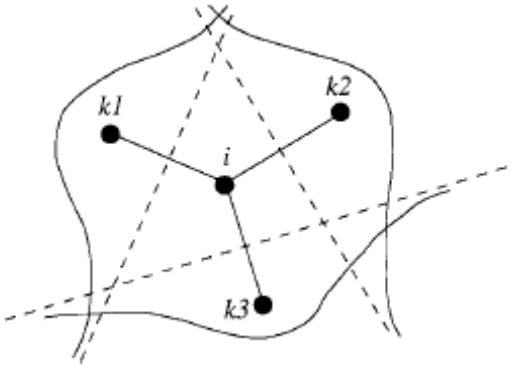
nodes. Οι edge nodes παρά το γεγονός ότι δεν έχουν ικανοποιήσει προφανώς τη φάση 1 υφίστανται τη διαδικασία της φάσης 2 και ο τελικός αριθμός γειτόνων τους καθορίζεται από τη φάση αυτή. Μειώνονται δηλαδή οι γείτονές τους στο ελάχιστο δυνατό, σύμφωνα με τις απαιτήσεις της φάσης 2, και στη συνέχεια ως ακτίνα τους τίθεται η μέγιστη.

## 4.6 Αλγόριθμος RR

Ο αλγόριθμος RR προγραμματίστηκε με βάση την θεωρητική προσέγγιση και τον ψευδοκώδικα που παρατίθεται στο [3] και αναλύθηκε διεξοδικά στο προηγούμενο κεφάλαιο.

Η πιο σημαντική απόφαση σχετικά με τη συγκεκριμένη υλοποίηση υπήρξε ο καθορισμός της περιοχής έρευνας, ο ορισμός της οποίας έχει ήδη δοθεί. Ως περιοχή έρευνας θεωρήθηκε τελικά η κυκλική περιοχή με κέντρο τον εκάστοτε κόμβο και ακτίνα τη μέγιστη τιμή ακτίνας που μπορεί να έχει ο κόμβος. Γίνεται μόνο μία επανάληψη για κάθε κόμβο και ως εκ τούτου ορίζεται μόνο μια περιοχή έρευνας, η μέγιστη δυνατή. Όλοι οι κόμβοι ξεκινούν με τη μέγιστη ακτίνα, εντοπίζουν τους υποψήφιους γείτονες που βρίσκονται εκτός αυτής και με την εφαρμογή των βημάτων του αλγορίθμου απορρίπτονται τελικά κάποιοι υποψήφιοι ως νεκροί και κάποιοι διατηρούνται ως ζωντανοί. Ο αλγόριθμος σταματά όταν σχηματιστεί μια περιοχή γύρω από τον κάθε κόμβο, η οποία και ονομάζεται περιοχή εγκλεισμού. Οι κόμβοι που παρέμειναν ζωντανοί μέχρι το τέλος αποτελούν τελικά και τους γείτονες του κόμβου που εξετάζεται. Ως ακτίνα του κόμβου αυτού ορίζεται η απόσταση του πιο απομακρυσμένου γείτονα από αυτόν.

Ένα δύσκολο προγραμματιστικά πρόβλημα αποδείχτηκε η υλοποίηση της συνθήκης τερματισμού που προαναφέρθηκε, δηλαδή η διαπίστωση της δημιουργίας της περιοχής εγκλεισμού γύρω από τον κάθε κόμβο. Το Σχήμα 14 το οποίο παρατέθηκε στο κεφάλαιο περιγραφής των αλγορίθμων και για λόγους ευκολίας παρατίθεται και παρακάτω, είναι πολύ κατατοπιστικό για τη λύση που τελικά βρέθηκε.




Παρατηρήθηκε η περιοχή εγκλεισμού ορίζεται από τα όρια των περιοχών σκυτάλης των γειτόνων του εξεταζόμενου κόμβου. Επομένως όταν από τα όρια των περιοχών αυτών σχηματίζεται μία κλειστή καμπύλη γύρω από τον κόμβο, αυτόματα καθορίζεται και η περιοχή εγκλεισμού του. Ο υπολογισμός των οριακών καμπυλών, ο έλεγχος των σημείων τομής τους και η διαπίστωση για το αν αυτές τελικά σχηματίζουν μια κλειστή περιοχή γύρω από τον κόμβο θα ήταν μια εξαιρετικά πολύπλοκη διαδικασία δεδομένης της φύσεως των καμπυλών που είναι αδύνατον να προσδιοριστούν από μια απλή εξίσωση. Είναι όμως εμφανές από το παραπάνω σχήμα

ότι εκτός από τις οριακές καμπύλες των περιοχών σκυτάλης, κλειστή περιοχή ορίζουν και οι ασύμπτωτες των καμπυλών αυτών. Οι ασύμπτωτες σε αντίθεση με τις καμπύλες μπορούν να χαρακτηριστούν από απλές εξισώσεις ευθειών και να εξυπηρετήσουν στους απαραίτητους υπολογισμούς. Ένα επιπλέον σημαντικό δεδομένο είναι το γεγονός ότι οι ευθείες αυτές είναι μεσοκάθετες στα ευθύγραμμα τμήματα που έχουν άκρα κάθε φορά τον κεντρικό κόμβο και τον εξεταζόμενο γείτονα. Επομένως η εξισώσεις των μεσοκαθέτων μπορούν ακόμα πιο εύκολα να προσδιοριστούν. Τελικά το αρχικό πολύπλοκο πρόβλημα ανάγεται στον προσδιορισμό των σημείων τομής των μεσοκαθέτων και στον έλεγχο της δημιουργίας από αυτές κλειστού πολυγώνου, δηλαδή κλειστής περιοχής γύρω από τον κεντρικό κόμβο. Για την υλοποίηση αυτού του ελέγχου χρησιμοποιήθηκε μια έτοιμη συνάρτηση η οποία βρέθηκε στο [9] και η οποία υπολογίζει εάν γύρω από ένα σημείο δημιουργείται κλειστό πολύγωνο, του οποίου οι κορυφές είναι γνωστές και στην συγκεκριμένη περίπτωση είναι οι τομές των μεσοκαθέτων.

Η εύρεση της περιοχής σκυτάλης η οποία ήταν απαραίτητη σε πολλά σημεία του αλγορίθμου έγινε μέσω της αξιοποίησης χαρακτηριστικών ιδιοτήτων της περιοχής αυτής και πιο συγκεκριμένα του γεγονότος ότι: Εάν ο κόμβος $j$ ανήκει στην $R_{i \to r}$ τότε ισχύουν τα παρακάτω:

- $d_{jr} < d_{ij}$
- $d_{ij} > d_{ir}$

όπου με $d_{kl}$ συμβολίζεται η απόσταση μεταξύ $k$ και $l$. Θεωρήθηκε επομένως ότι ισχύει και το αντίστροφο, ότι δηλαδή αν τα δύο παραπάνω δεδομένα ισχύουν τότε ο κόμβος $j$ ανήκει στην $R_{i \to r}$. Με αυτή την λογική έγιναν όλοι οι απαιτούμενοι από το πρόγραμμα έλεγχοι.

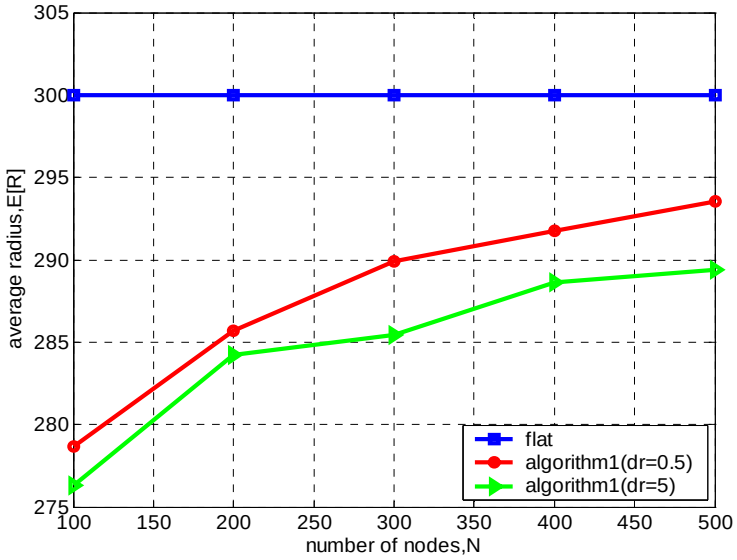
Τέλος πρέπει να αναφερθεί και σε αυτόν τον αλγόριθμο με ποιον τρόπο αντιμετωπίστηκαν οι edge nodes. Παρά το γεγονός ότι ήταν αδύνατον να δημιουργηθεί κλειστή καμπύλη γύρω από αυτούς, σύμφωνα με τα βήματα του αλγορίθμου υπήρξε δυνατότητα διαχωρισμού των γειτόνων σε νεκρούς και ζωντανούς. Οι ζωντανοί θεωρήθηκαν οι τελικοί γείτονες των κόμβων αυτών και η ακτίνα τους ορίστηκε η μέγιστη.

# Κεφάλαιο 5 – Παρουσίαση αποτελεσμάτων

Για κάθε αλγόριθμο πραγματοποιήθηκαν δύο σειρές μετρήσεων. Μια για αυξανόμενη πυκνότητα δικτύου και μια για σταθερή. Επίσης μετρήσεις έγιναν για δύο τιμές βήματος ακτίνας, $dr$, για να ελεγχθεί η συμβολή της παραμέτρου αυτής στο τελικό αποτέλεσμα. Τα μεγέθη που εξετάστηκαν ήταν η μεταβολή της μέσης ακτίνας σε σχέση με τον αριθμό των κόμβων, η μεταβολή του μέσου αριθμού γειτόνων σε σχέση με τον αριθμό των κόμβων και η διασπορά αυτών. Στην σειρά μετρήσεων της αυξανόμενης πυκνότητας το μέγεθος της γεωγραφικής περιοχής του δικτύου παρέμενε σταθερό στα *1000m* και ο αριθμός κόμβων πήρε τις τιμές *100*, *200*, *300*, *400*, *500*. Για σταθερή πυκνότητα το μέγεθος της γεωγραφικής περιοχής πήρε τις τιμές *1000*, *1400*, *1700*, *1900*, *2100* και ο αριθμός κόμβων διαμορφώθηκε αντίστοιχα στις τιμές *100*, *196*, *289*, *361*, *441* έτσι ώστε η πυκνότητα να διατηρηθεί σταθερή.

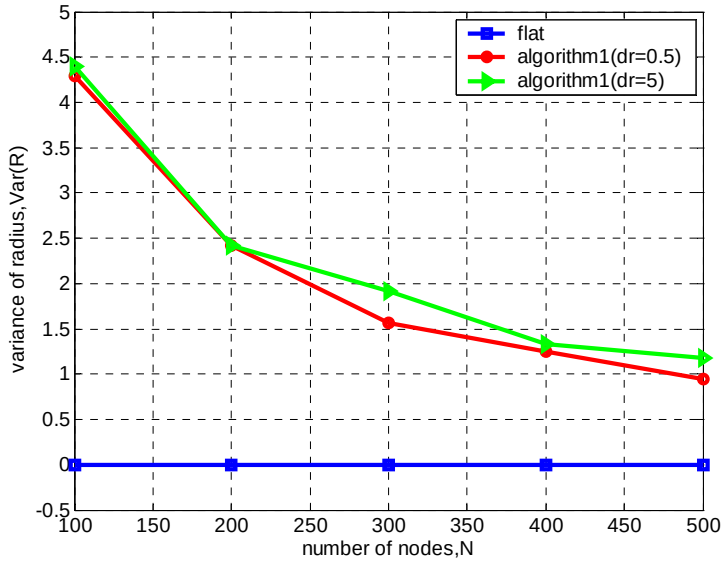
## 5.1 Αλγόριθμος I

### 5.1.1 Αυξανόμενη πυκνότητα



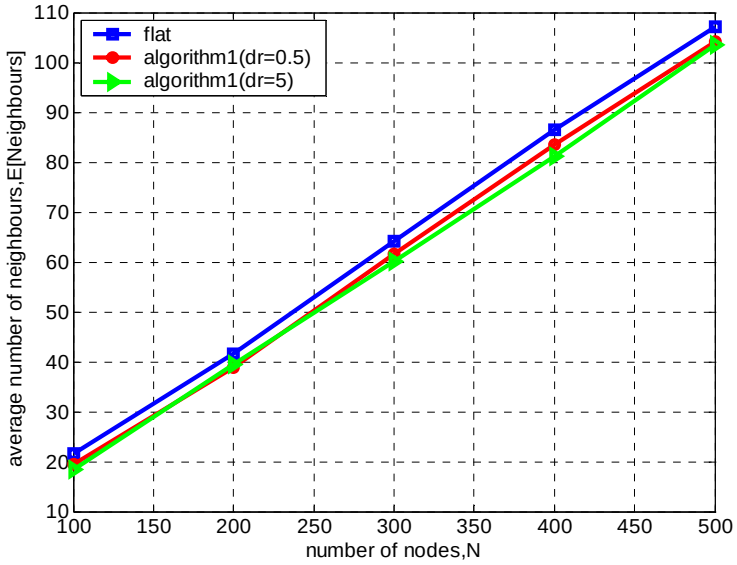

**Διάγραμμα 1. Αλγόριθμος I-Αυξανόμενη πυκνότητα-Μέση ακτίνα**

Όπως είναι εμφανές από το Διάγραμμα 1 η εφαρμογή του αλγορίθμου I έχει ως αποτέλεσμα τη μείωση της μέσης ακτίνας σε σχέση με τη μέγιστη τιμή που έχει στο αρχικό δίκτυο. Η συμπεριφορά αυτή ήταν αναμενόμενη διότι ο αλγόριθμος επιδιώκει την επίτευξη της ελάχιστης δυνατής ακτίνας με περιορισμό όλες οι γωνίες κώνων να είναι μεγαλύτερες από $\pi$. Η αύξηση της πυκνότητας του δικτύου έχει ως αποτέλεσμα την αύξηση της μέσης ακτίνας. Αύξηση της πυκνότητας οδηγεί σε αύξηση του αριθμού των γειτόνων κάθε κόμβου. Επομένως η ικανοποίηση του κριτηρίου που θέτει ο αλγόριθμος συμβαίνει όλο και πιο γρήγορα χωρίς να χρειαστεί να μειωθεί τόσο πολύ η ακτίνα του κόμβου. Τέλος παρατηρούμε ότι για τα δύο βήματα ακτίνας που επιλέχθηκαν η συμπεριφορά του δικτύου είναι παρόμοια. Αυτό συμβαίνει σχεδόν στο σύνολο των γραφικών παραστάσεων που προέκυψαν, επομένως στην συνέχεια ιδιαίτερη αναφορά στον παράγοντα αυτό θα γίνεται μόνο όταν υπάρχει κάποια διαφοροποίηση.




**Διάγραμμα 2. Αλγόριθμος I-Αυξανόμενη πυκνότητα-Διασπορά ακτίνας**

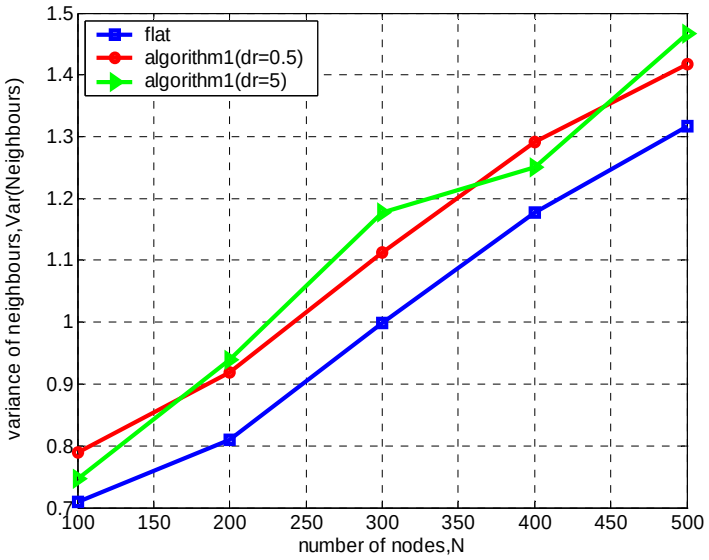
Η διασπορά της μέσης ακτίνας του αρχικού δικτύου προέκυψε μηδενική στο Διάγραμμα 2 διότι όλοι οι κόμβοι είχαν τη μέγιστη τιμή ακτίνας. Μετά την εφαρμογή του αλγορίθμου I παρατηρείται αύξηση της διασποράς σε σχέση με το αρχικό δίκτυο (εφόσον οι κόμβοι πλέον έχουν διαφορετικές ακτίνες). Επίσης με την αύξηση της πυκνότητας δικτύου η διασπορά μειώνεται, δηλαδή η τιμή της ακτίνας των κόμβων τείνει να προσεγγίσει τη μέση τιμή. Αυτό είναι ιδιαίτερα χρήσιμο, διότι εάν όλοι οι κόμβοι τελικά απαιτούνται να έχουν παραπλήσιες με τη μέση τιμή ακτίνες, μπορούν να κατασκευαστούν με τις ίδιες προδιαγραφές.




**Διάγραμμα 3. Αλγόριθμος I-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων**

Στο Διάγραμμα 3 παρατηρείται μείωση του μέσου αριθμού γειτόνων σε σχέση με αυτόν του αρχικού δικτύου. Αυτή ήταν και η επιδίωξη του αλγορίθμου. Διατηρούνται μόνο εκείνοι οι γείτονες που είναι απαραίτητοι για την ικανοποίηση του κριτηρίου.

Με την αύξηση της πυκνότητας ο μέσος αριθμός γειτόνων αυξάνεται. Αυτό είναι άμεσα συνυφασμένο με την ερμηνεία που δόθηκε παραπάνω για την αύξηση της μέσης ακτίνας στην ίδια περίπτωση.

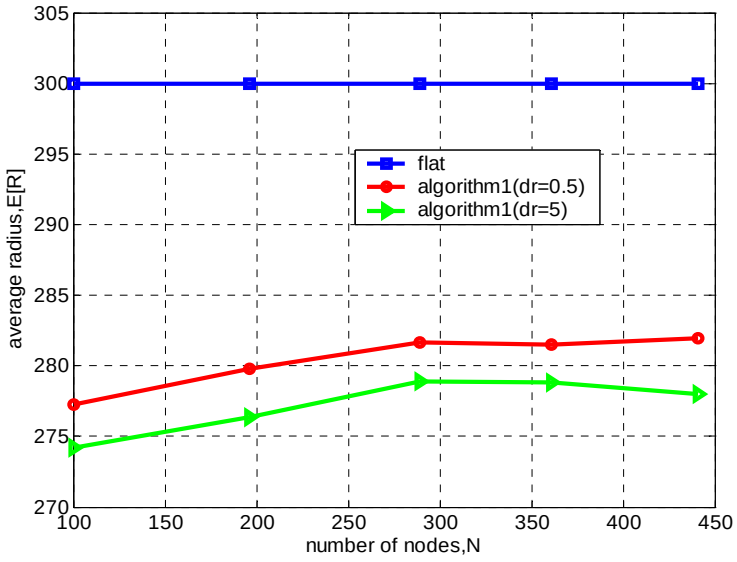



**Διάγραμμα 4. Αλγόριθμος Ι-Αυξανόμενη πυκνότητα-Διασπορά αριθμού γειτόνων**

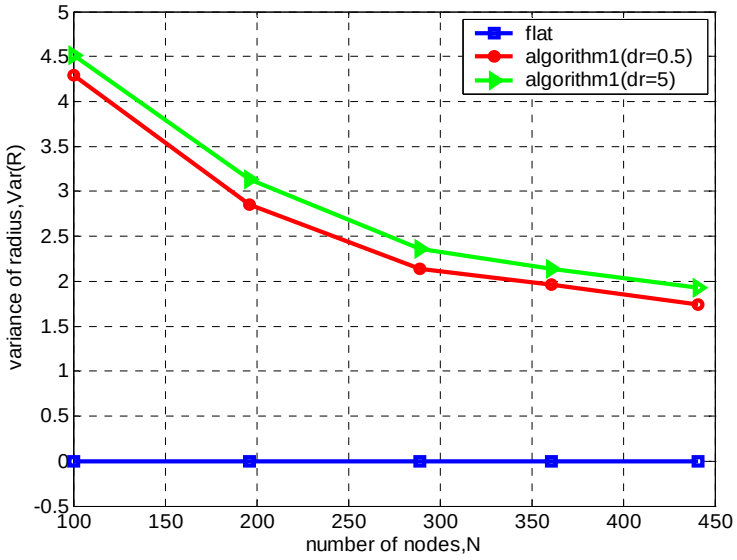
Η μέση διασπορά του αριθμού των γειτόνων εμφανίζεται στο Διάγραμμα 4 ελαφρώς αυξημένη σε σχέση με αυτήν του αρχικού δικτύου, αφού στο αρχικό δίκτυο οι κόμβοι είχαν όλοι την ίδια ακτίνα και είχαν παρόμοιους μέσους αριθμούς γειτόνων (ομοιόμορφα κατανεμημένοι). Με την εφαρμογή του αλγορίθμου υπήρξαν διαφοροποιήσεις, οι οποίες οφείλονται και σε ένα πολύ σημαντικό φαινόμενο, που αναλύθηκε στο προηγούμενο κεφάλαιο διεξοδικά, τους edge nodes. Αυτοί οι κόμβοι διατηρούν τους αρχικούς γείτονές τους και διαφέρουν στον αριθμό γειτόνων αρκετά από κόμβους στους οποίους εφαρμόστηκε πλήρως ο αλγόριθμος. Παρατηρούνται έτσι περισσότερες αποκλίσεις από τη μέση τιμή. Η διασπορά αυξάνεται με την αύξηση της πυκνότητας.

### 5.1.2 Σταθερή πυκνότητα

Στο Διάγραμμα 5 παρατηρείται μία τάση σταθεροποίησης της μέσης ακτίνας στο τελικό δίκτυο όταν η πυκνότητα παραμένει σταθερή. Η συμπεριφορά αυτή είναι απολύτως φυσιολογική και επιθυμητή εφόσον η σταθερή πυκνότητα εξασφαλίζει ότι η μέση ακτίνα κόμβων με την οποία επιτυγχάνεται η ικανοποίηση του κριτηρίου δεν θα μεταβάλλεται σημαντικά.

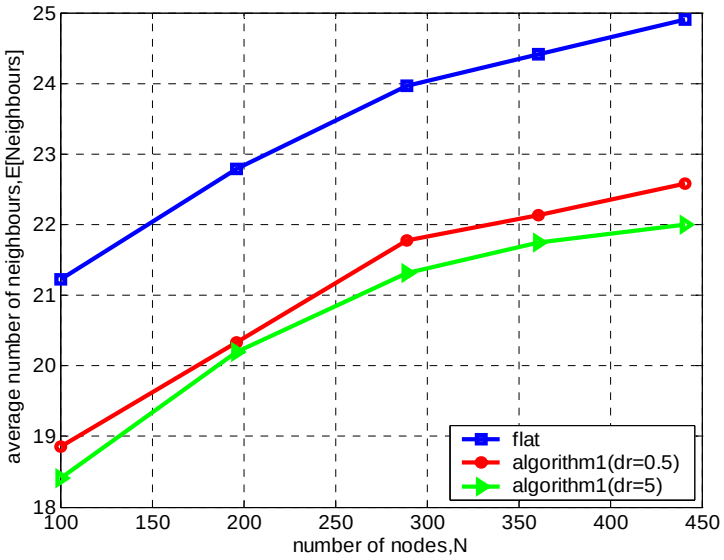



**Διάγραμμα 5. Αλγόριθμος Ι-Σταθερή πυκνότητα-Μέση ακτίνα**



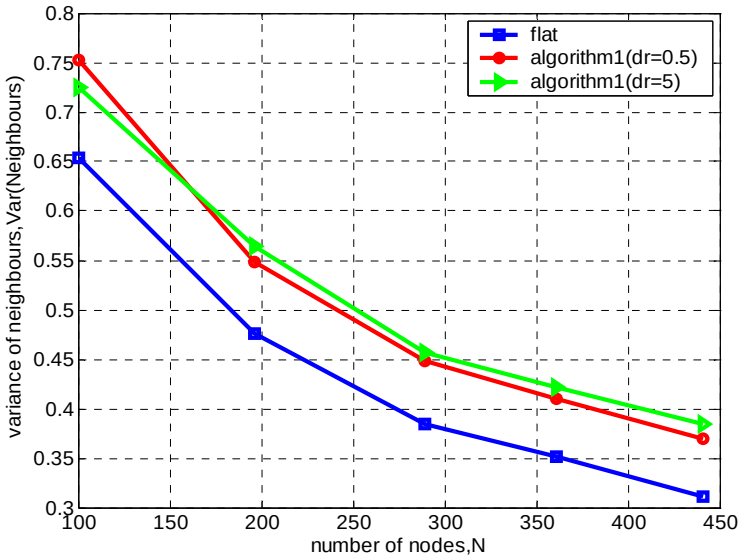

**Διάγραμμα 6. Αλγόριθμος Ι-Σταθερή πυκνότητα-Διασπορά ακτίνας**

Η διασπορά της ακτίνας συμπεριφέρεται στο Διάγραμμα 6 παρόμοια με την περίπτωση αυξανόμενης πυκνότητας.




**Διάγραμμα 7. Αλγόριθμος Ι-Σταθερή πυκνότητα-Μέσος αριθμός γειτόνων**

Ο μέσος αριθμός γειτόνων σταθεροποιείται επίσης σύμφωνα με το Διάγραμμα 7. Με μία προσεκτικότερη παρατήρηση του σχετικού εύρους τιμών, βλέπουμε ότι οι τιμές με σταθερή πυκνότητα κυμαίνονται (και στο αρχικό και στο τελικό δίκτυο) από 18 μέχρι 25 γείτονες ενώ στην περίπτωση της αυξανόμενης πυκνότητας η διακύμανση είναι από 20 μέχρι 110 γείτονες. Κάθε κόμβος δεν έχει μεγάλες μεταβολές στον αριθμό των γειτόνων του εφόσον η σταθερή πυκνότητα εγγυάται ότι θα τον πλαισιώνει περίπου ο ίδιος αριθμός κόμβων κάθε φορά. Η μείωση του εύρους τιμών που αναλύθηκε παραπάνω έχει ως αποτέλεσμα και την ευκρινέστερη παρατήρηση της μείωσης του μέσου αριθμού γειτόνων μετά την εφαρμογή του αλγορίθμου.

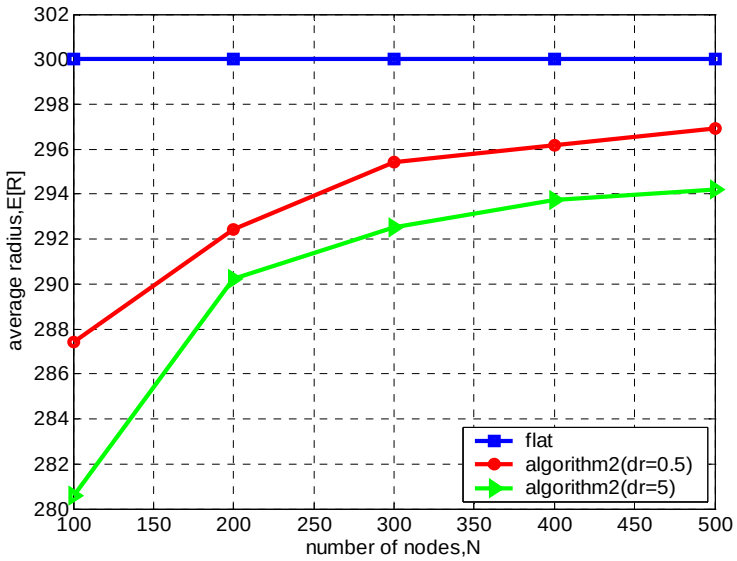



**Διάγραμμα 8. Αλγόριθμος Ι-Σταθερή πυκνότητα-Διασπορά αριθμού γειτόνων**

Αντίθετα με την αυξανόμενη πυκνότητα στο Διάγραμμα 8 παρατηρούμε ότι η διασπορά του μέσου αριθμού γειτόνων μειώνεται με την αύξηση του αριθμού των κόμβων, αν και αυξημένη σε σχέση με το αρχικό δίκτυο.

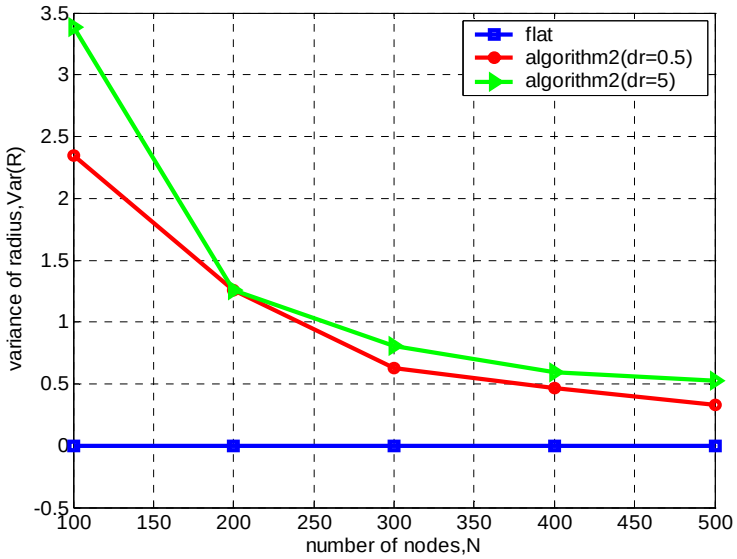
## 5.2 Αλγόριθμος ΙΙ

### 5.2.1 Αυξανόμενη πυκνότητα



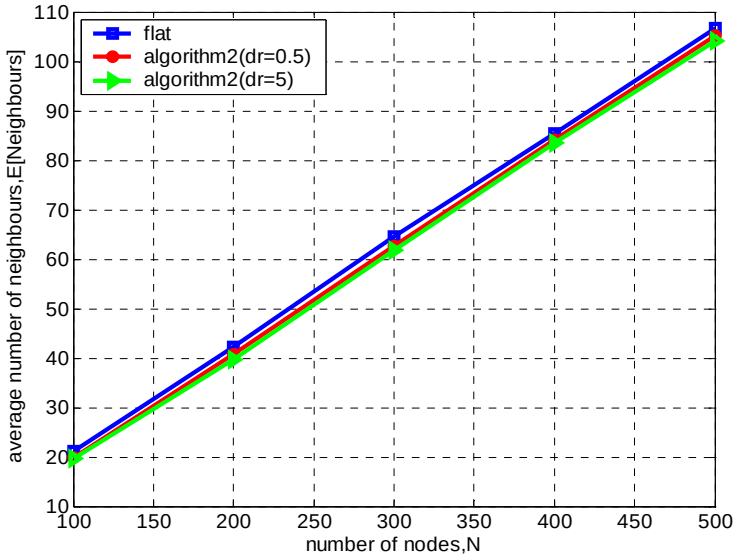

**Διάγραμμα 9. Αλγόριθμος ΙΙ-Αυξανόμενη πυκνότητα-Μέση ακτίνα**

Ο αλγόριθμος ΙΙ έχει την ίδια συμπεριφορά με τον Ι, όπως προκύπτει από το Διάγραμμα 9. Η μέση ακτίνα μειώνεται σε σχέση με το αρχικό δίκτυο και αυξάνεται με την αύξηση της πυκνότητας.



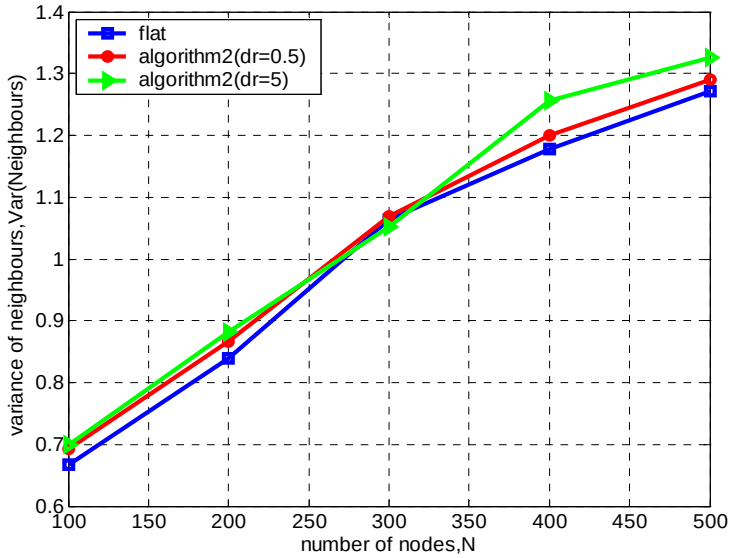

**Διάγραμμα 10. Αλγόριθμος ΙΙ-Αυξανόμενη πυκνότητα-Διασπορά ακτίνας**

Στο Διάγραμμα 10 εμφανές είναι ότι η διασπορά της ακτίνας μειώνεται με την αύξηση της πυκνότητας.




**Διάγραμμα 11. Αλγόριθμος ΙΙ-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων**

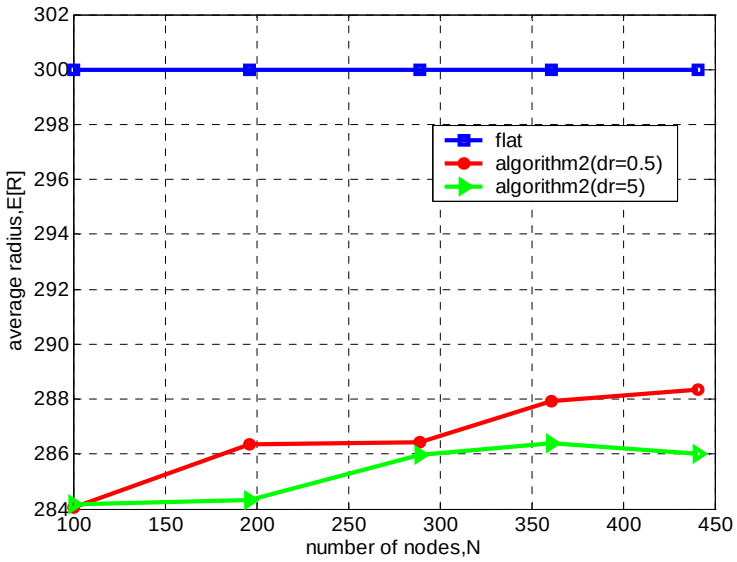
Ο μέσος αριθμός γειτόνων στο Διάγραμμα 11 εμφανίζεται ελαφρά μειωμένος σε σχέση με αυτόν του αρχικού δικτύου όπως αναμένεται. Αυξάνεται με την αύξηση του αριθμού των κόμβων και επομένως της πυκνότητας.




**Διάγραμμα 12. Αλγόριθμος ΙΙ-Αυξανόμενη πυκνότητα-Διασπορά αριθμού γειτόνων**

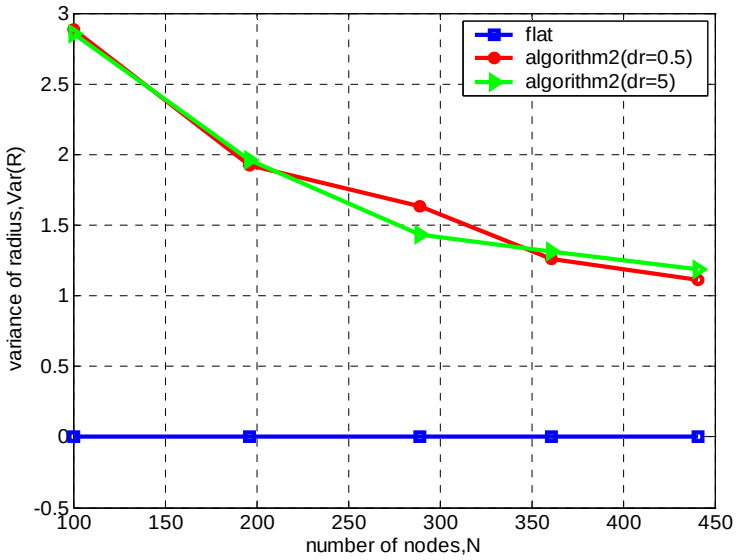
Από το Διάγραμμα 12 προκύπτει ότι η διασπορά του μέσου αριθμού γειτόνων στο τελικό δίκτυο είναι ελαφρώς αυξημένη σε σχέση με αυτήν του τελικού δικτύου και αυξάνεται με την αύξηση της πυκνότητας.

### 5.2.2 Σταθερή πυκνότητα



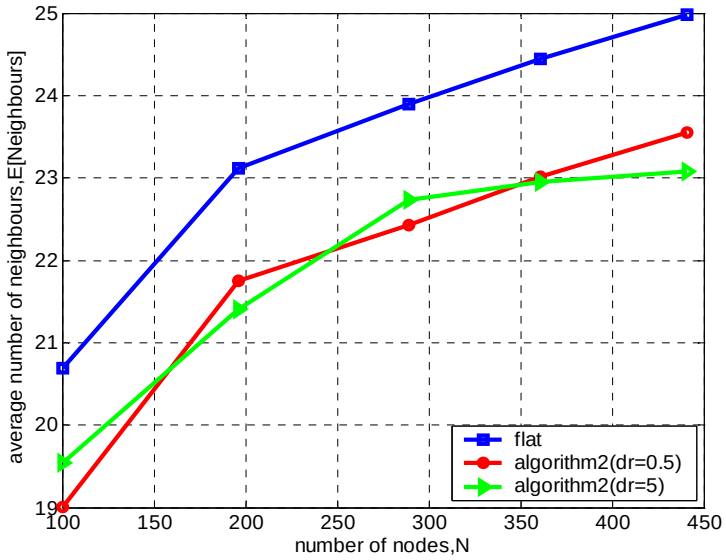

**Διάγραμμα 13. Αλγόριθμος ΙΙ-Σταθερή πυκνότητα-Μέση ακτίνα**

Όπως και στις μετρήσεις με αυξανόμενη πυκνότητα έτσι και στις μετρήσεις με σταθερή δεν υπάρχουν πολλές διαφοροποιήσεις σε σχέση με τον αλγόριθμο Ι. Η σταθεροποίηση της μέσης ακτίνας είναι εμφανής από το Διάγραμμα 13.



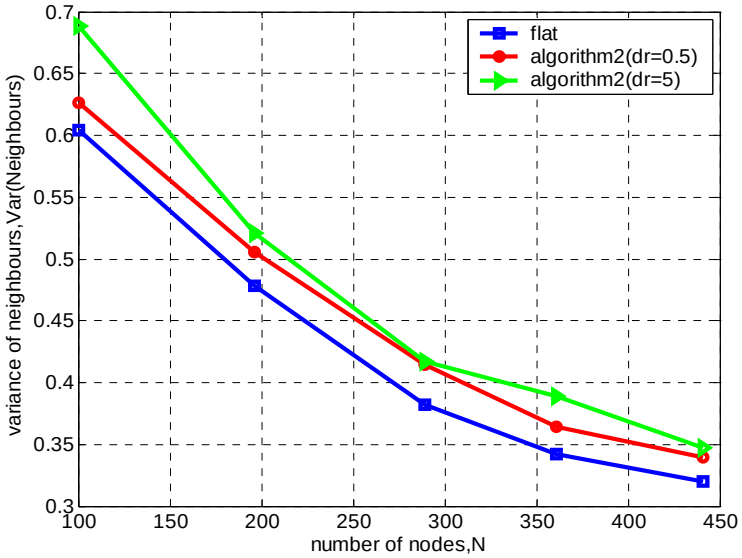

**Διάγραμμα 14. Αλγόριθμος ΙΙ-Σταθερή πυκνότητα-Διασπορά ακτίνας**

Στο Διάγραμμα 14 η διασπορά μειώνεται με την αύξηση του αριθμού των κόμβων.




**Διάγραμμα 15. Αλγόριθμος ΙΙ-Σταθερή πυκνότητα-Μέσος αριθμός γειτόνων**

Στο Διάγραμμα 15 ο μέσος αριθμός γειτόνων έχει μειωθεί σε σχέση με το αρχικό και εμφανίζει αυξητική τάση, κυμαινόμενος όμως σε πολύ μικρότερο εύρος τιμών από την περίπτωση της αυξανόμενης πυκνότητας. Ουσιαστικά τείνει να σταθεροποιηθεί.



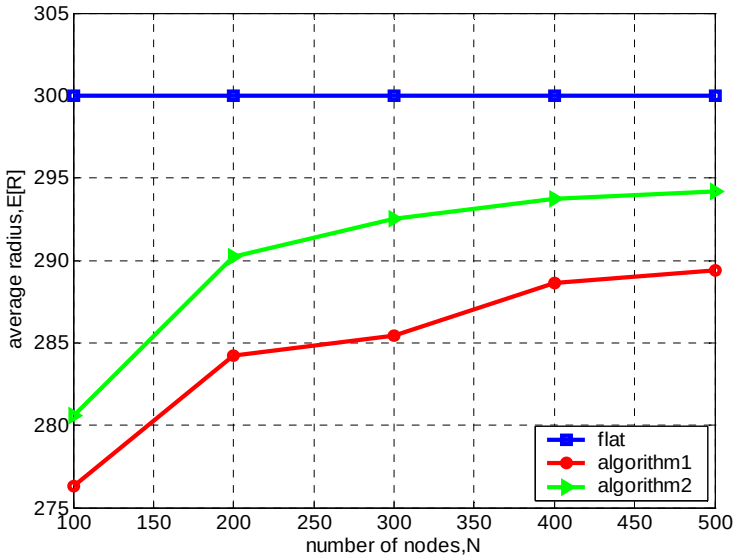

**Διάγραμμα 16. Αλγόριθμος ΙΙ-Σταθερή πυκνότητα-Διασπορά αριθμού γειτόνων**

Η διασπορά στο Διάγραμμα 16 μειώνεται με την αύξηση του μεγέθους του δικτύου.

## 5.3 Συγκριτικά για αλγορίθμους Ι και ΙΙ

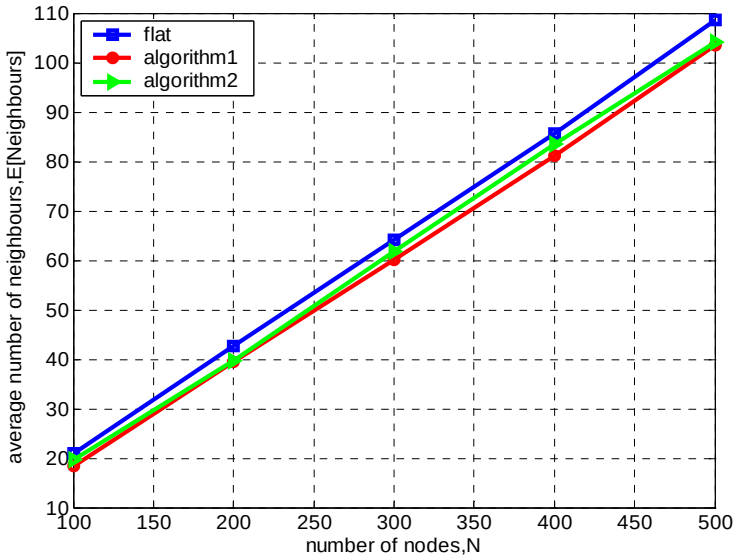
Για πιο ξεκάθαρη και ευκολότερη σύγκριση των αλγορίθμων επιλέχθηκαν να χρησιμοποιηθούν μόνο οι γραφικές παραστάσεις με βήμα ακτίνας 5. Αυτό εφαρμόστηκε σε όλα τα συγκριτικά διαγράμματα.

### 5.3.1 Αυξανόμενη πυκνότητα ( $dr = 5$ )




**Διάγραμμα 17. Συγκριτικό I και II-Αυξανόμενη πυκνότητα-Μέση ακτίνα**

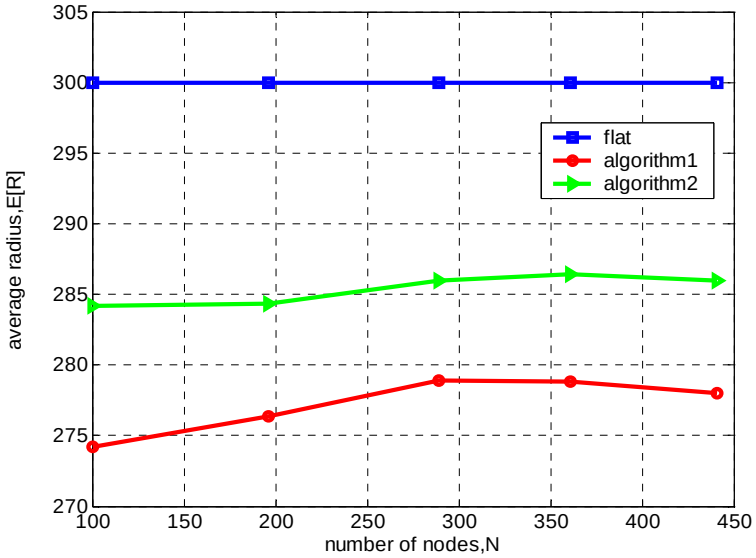
Παρατηρούμε στο Διάγραμμα 17 ότι ο αλγόριθμος II δίνει μεγαλύτερες τιμές μέσης ακτίνας από τον I. Αυτό οφείλεται το δεύτερο βήμα του αλγορίθμου, που είναι στην ουσία και η διαφοροποίησή του από τον I. Στο βήμα αυτό καθορίζεται ότι η ακτίνα που υπολογίζεται από το πρώτο βήμα συγκρίνεται με το μήκος της μακρύτερης ζεύξης που εισέρχεται στον κόμβο και τελικά επιλέγεται το μέγιστο των δύο. Δηλαδή στις περιπτώσεις που η μακρύτερη εισερχόμενη ζεύξη είναι μεγαλύτερη από την υπολογισμένη ακτίνα, η ακτίνα αυξάνει. Κάτι τέτοιο οδηγεί τελικά στην αύξηση του μέσου όρου.



**Διάγραμμα 18. Συγκριτικό I και II-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων**

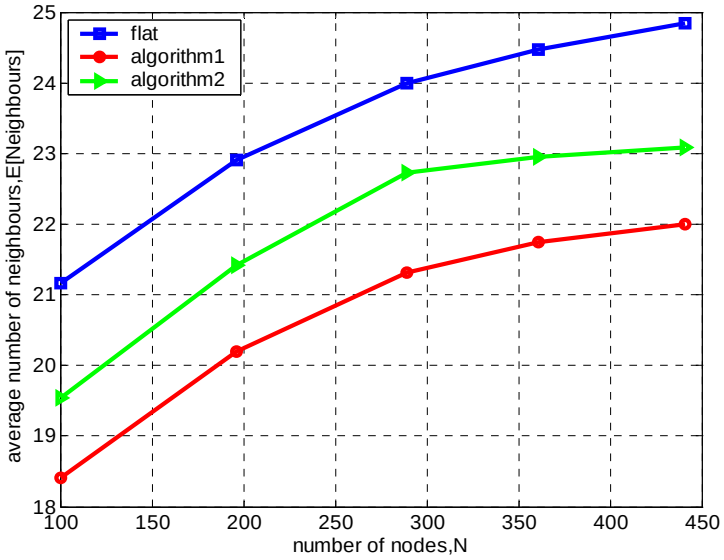
Διακρίνουμε στο Διάγραμμα 18 ότι ο μέσος αριθμός γειτόνων του αλγορίθμου II είναι ελαφρώς μεγαλύτερος από αυτόν του I. Αυτό έχει άμεση σχέση με την αύξηση της ακτίνας που εξηγήθηκε προηγουμένως.

### 5.3.2 Σταθερή πυκνότητα ($dr = 5$)




**Διάγραμμα 19. Συγκριτικό I και II-Σταθερή πυκνότητα-Μέση ακτίνα**

Εμφανής στο Διάγραμμα 19 είναι η σταθεροποίηση της μέσης ακτίνας και στους δύο αλγορίθμους. Η μέση ακτίνα του II έχει όπως και πριν μεγαλύτερες τιμές από αυτήν του I.

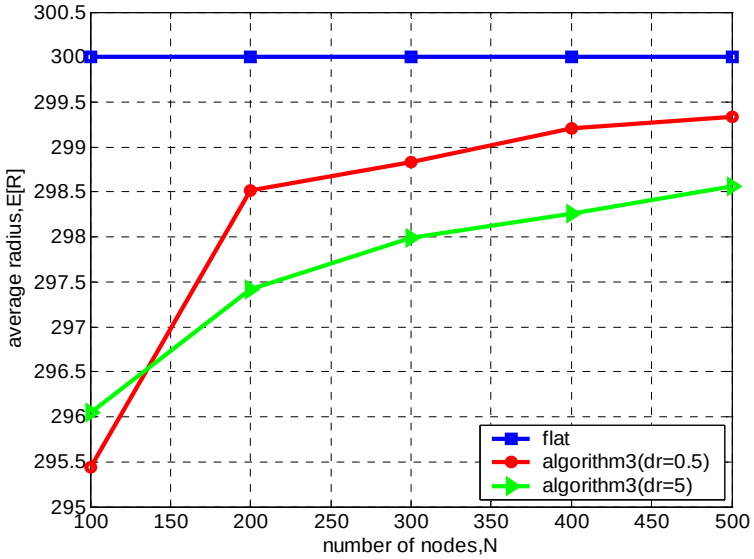



**Διάγραμμα 20. Συγκριτικό I και II-Σταθερή πυκνότητα-Μέσος αριθμός γειτόνων**

Ομοίως με την περίπτωση της αυξανόμενης πυκνότητας εξηγείται και το Διάγραμμα 20 με την επισήμανση ότι διακρίνεται ευκολότερα ο μεγαλύτερος μέσος αριθμός γειτόνων του ΙΙ σε σχέση με του αλγορίθμου Ι.

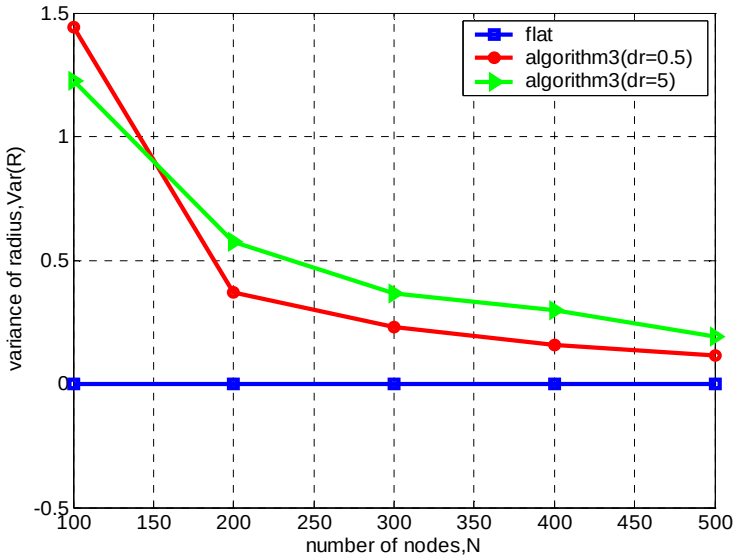
## 5.4 Αλγόριθμος ΙΙΙ

### 5.4.1 Αυξανόμενη πυκνότητα



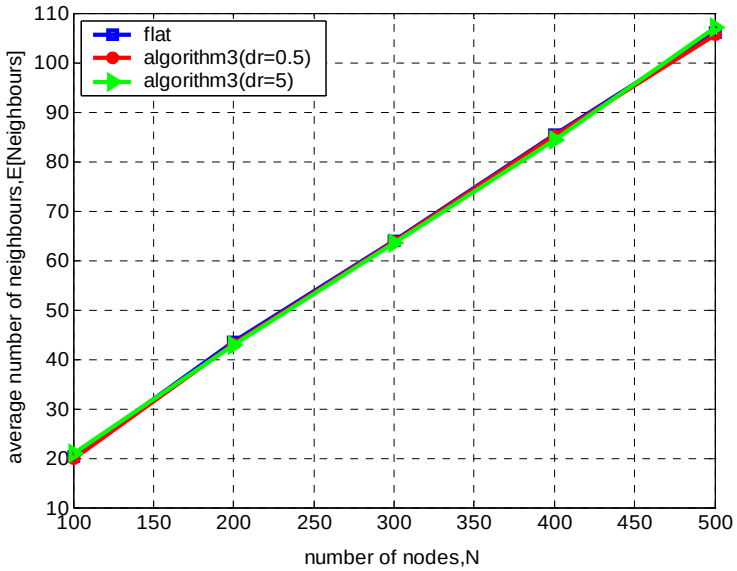

**Διάγραμμα 21. Αλγόριθμος ΙΙΙ-Αυξανόμενη πυκνότητα-Μέση ακτίνα**

Παρατηρούμε στο Διάγραμμα 21 ότι σε σχέση με το αρχικό δίκτυο μειώνεται η μέση ακτίνα με την εφαρμογή του αλγορίθμου ΙΙΙ. Στο τελικό δίκτυο παρατηρούμε επίσης ότι με την αύξηση του αριθμού των κόμβων αυξάνεται και η μέση ακτίνα.



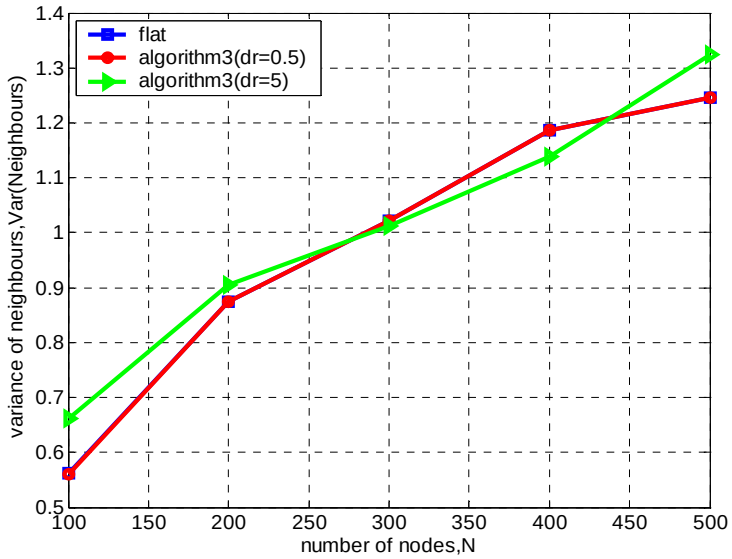

**Διάγραμμα 22. Αλγόριθμος ΙΙΙ-Αυξανόμενη πυκνότητα-Διασπορά ακτίνας**

Η διασπορά της μέσης ακτίνας μειώνεται με την αύξηση της πυκνότητας στο Διάγραμμα 22. Ακριβώς όπως παρατηρήσαμε και στους αλγορίθμους I και II.




**Διάγραμμα 23. Αλγόριθμος III-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων**

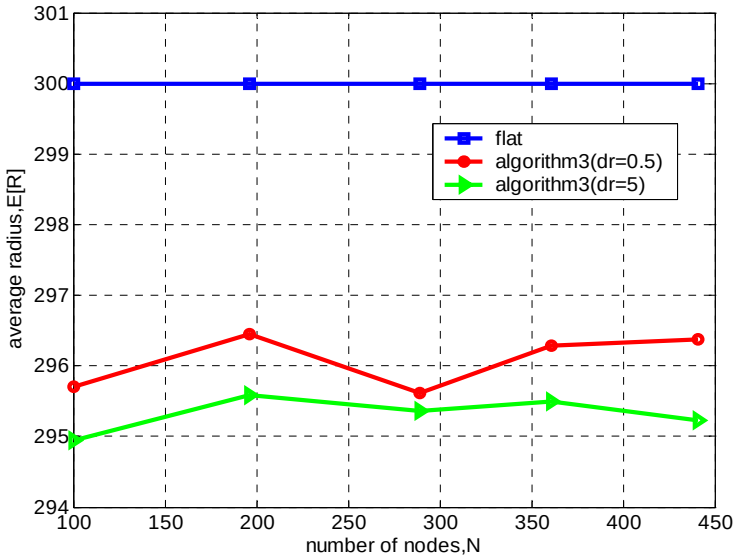
Στην αυξανόμενη πυκνότητα δεν μπορούμε να διακρίνουμε μεγάλες διαφοροποιήσεις στο μέσο αριθμό γειτόνων, όπως είναι εμφανές από το Διάγραμμα 23. Τα αποτελέσματα θα είναι πιο ξεκάθαρα στην σταθερή πυκνότητα.




**Διάγραμμα 24. Αλγόριθμος III-Αυξανόμενη πυκνότητα-Διασπορά αριθμού γειτόνων**

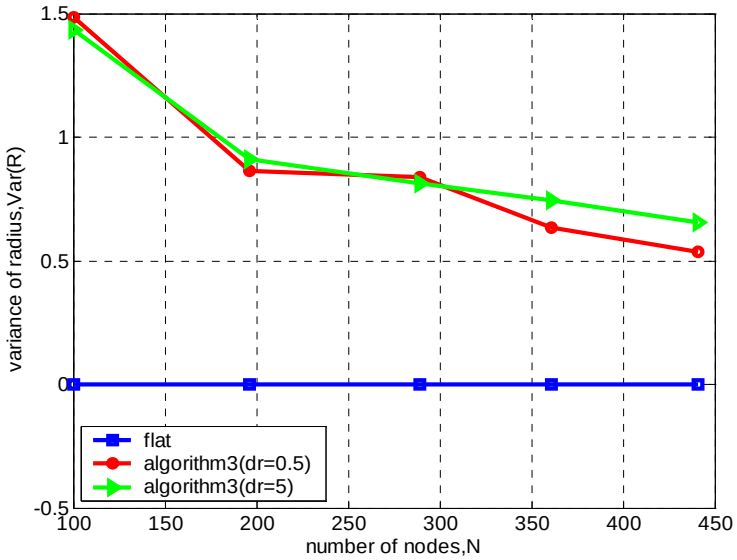
Στο Διάγραμμα 24 η διασπορά του αριθμού γειτόνων αυξάνεται με την αύξηση των κόμβων του δικτύου όπως και στους προηγούμενους αλγορίθμους. Σε αυτήν την περίπτωση δεν διαφοροποιείται από τη διασπορά του αρχικού δικτύου. Η διαφοροποίηση που παρατηρείται στη γραφική με βήμα ακτίνας 5 οφείλεται στο γεγονός ότι για αρχικό δίκτυο χρησιμοποιήθηκε το αρχικό δίκτυο που προέκυψε με βήμα 0.5. Δεν κρίθηκε απαραίτητο να παρατεθούν και τα δύο αρχικά δίκτυα καθώς οι τιμές που προέκυπταν από τις μετρήσεις ήταν πολύ κοντινές.

## 5.4.2 Σταθερή πυκνότητα



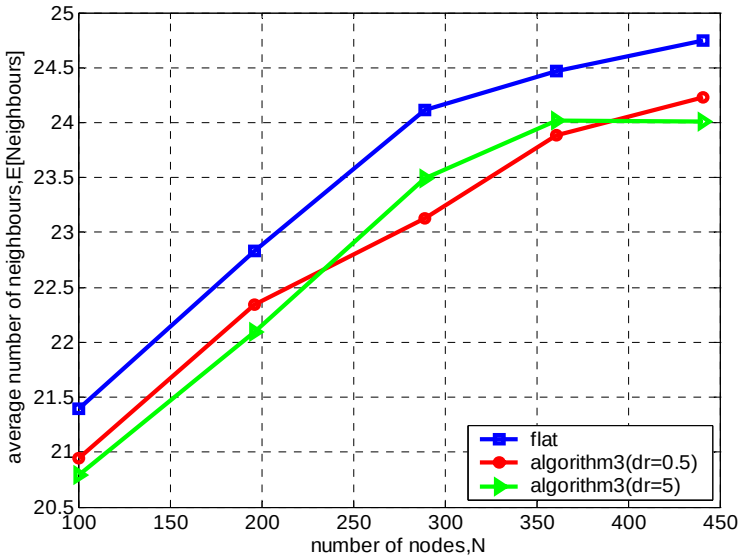

**Διάγραμμα 25. Αλγόριθμος ΙΙΙ-Σταθερή πυκνότητα-Μέση ακτίνα**

Στο Διάγραμμα 25 παρατηρούμε την σταθεροποίηση της μέσης ακτίνας σε αρκετά μικρότερες τιμές από αυτές που παίρνει στο αρχικό δίκτυο.



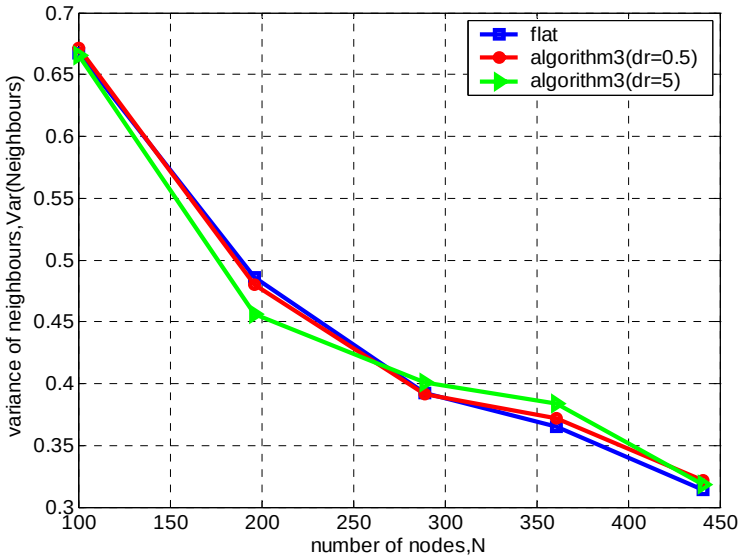

**Διάγραμμα 26. Αλγόριθμος ΙΙΙ-Σταθερή πυκνότητα-Διασπορά ακτίνας**

Στο Διάγραμμα 26 η διασπορά της μέσης ακτίνας μειώνεται με την αύξηση του μεγέθους του δικτύου.




**Διάγραμμα 27. Αλγόριθμος ΙΙΙ-Σταθερή πυκνότητα-Μέσος αριθμός γειτόνων**

Στο Διάγραμμα 27 φαίνεται ξεκάθαρα η μείωση που επιφέρει ο αλγόριθμος στον μέσο αριθμό γειτόνων. Αυτή η παρατήρηση δεν μπόρεσε να γίνει από την γραφική παράσταση της αυξανόμενης πυκνότητας, όπως ήδη επισημάνθηκε. Ο μέσος αριθμός γειτόνων αυξάνεται με την αύξηση του μεγέθους του δικτύου όπως είναι λογικό αλλά κυμαίνεται σε πολύ μικρότερο εύρος τιμών σε σχέση με την αυξανόμενη πυκνότητα.




**Διάγραμμα 28. Αλγόριθμος ΙΙΙ-Σταθερή πυκνότητα-Διασπορά αριθμού γειτόνων**

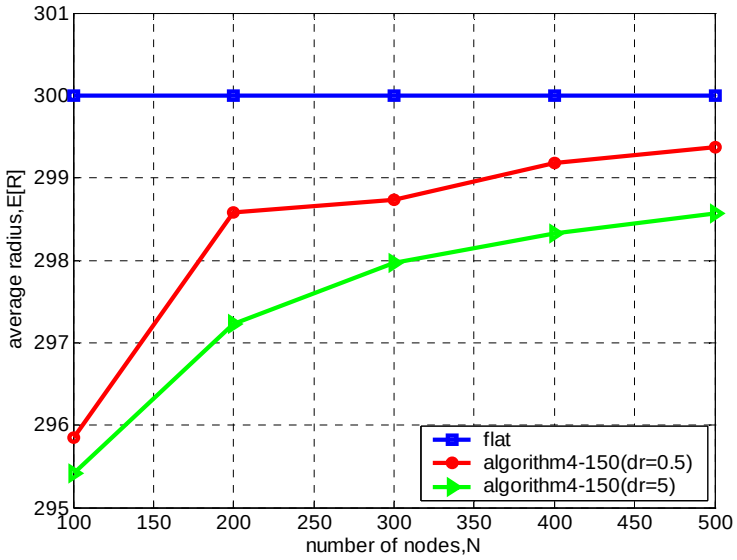
Η διασπορά έχει ακριβώς την ίδια συμπεριφορά στο Διάγραμμα 28 όπως και στην αυξανόμενη πυκνότητα.

## 5.5 Αλγόριθμος IV

Σύμφωνα με τη θεωρητική ανάλυση που προηγήθηκε στα προηγούμενα κεφάλαια, προκύπτει ότι η πρώτη φάση του αλγορίθμου είναι στην ουσία ο αλγόριθμος III και η δεύτερη φάση θέτει ένα κατώφλι, κάτω από το οποίο η ακτίνα δεν μπορεί να πάρει τιμές. Στις μετρήσεις που έγιναν χρησιμοποιηθήκαν δύο διαφορετικές τιμές κατωφλίου με σκοπό να μελετηθεί η επίδραση που έχει στο δίκτυο η επιλογή του κατωφλίου. Πιο συγκεκριμένα χρησιμοποιήθηκαν οι τιμές κατωφλίου 150 και 250.

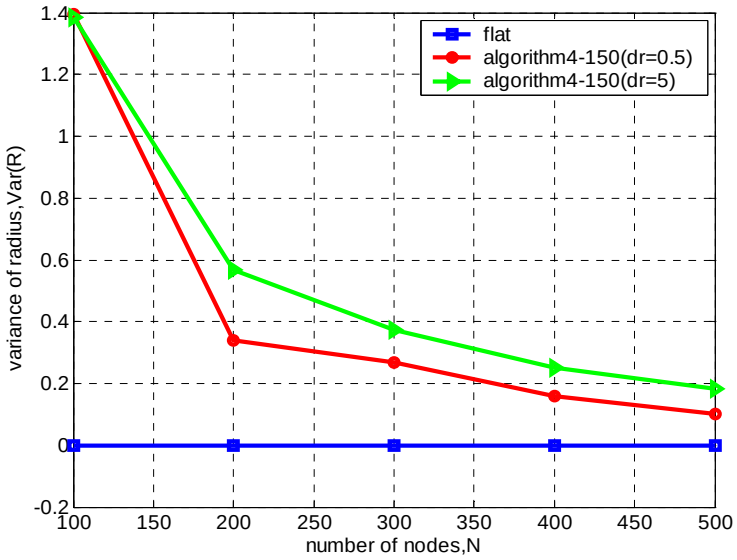
### 5.5.1 δ=150

#### 5.5.1.1 Αυξανόμενη πυκνότητα



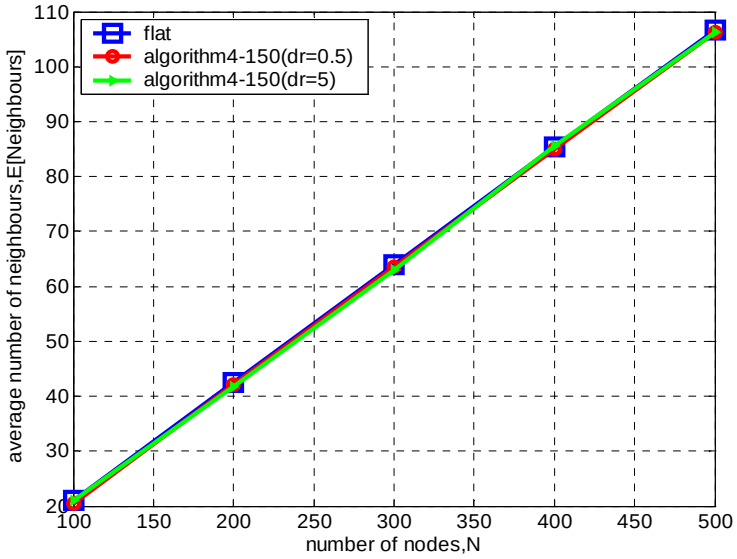

**Διάγραμμα 29. Αλγόριθμος IV(150)-Αυξανόμενη πυκνότητα-Μέση ακτίνα**

Και σε αυτόν τον αλγόριθμο παρατηρείται από το Διάγραμμα 29 μείωση της μέσης ακτίνας στο τελικό δίκτυο σε σχέση με το αρχικό αλλά και παράλληλη αύξηση της μέσης ακτίνας με αύξηση του αριθμού των κόμβων.



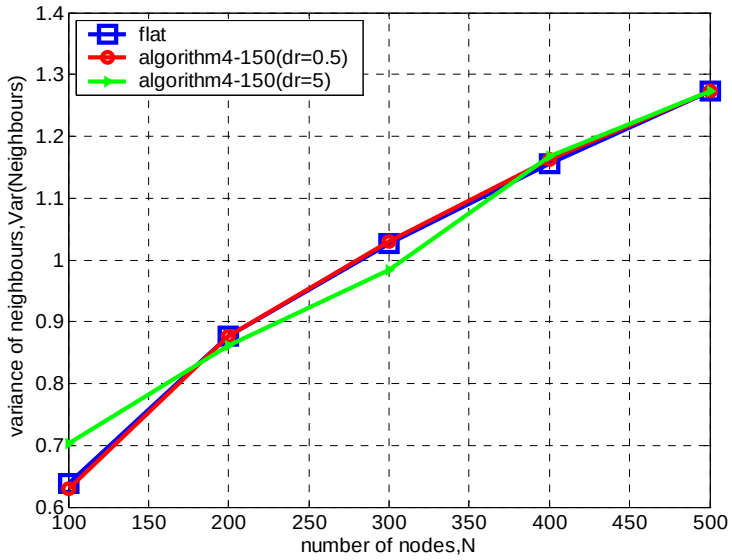

**Διάγραμμα 30. Αλγόριθμος IV(150)-Αυξανόμενη πυκνότητα-Διασπορά ακτίνας**

Μείωση της διασποράς της μέσης ακτίνας με την αύξηση του αριθμού των κόμβων παρατηρούμε στο Διάγραμμα 30.




**Διάγραμμα 31. Αλγόριθμος IV(150)-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων**

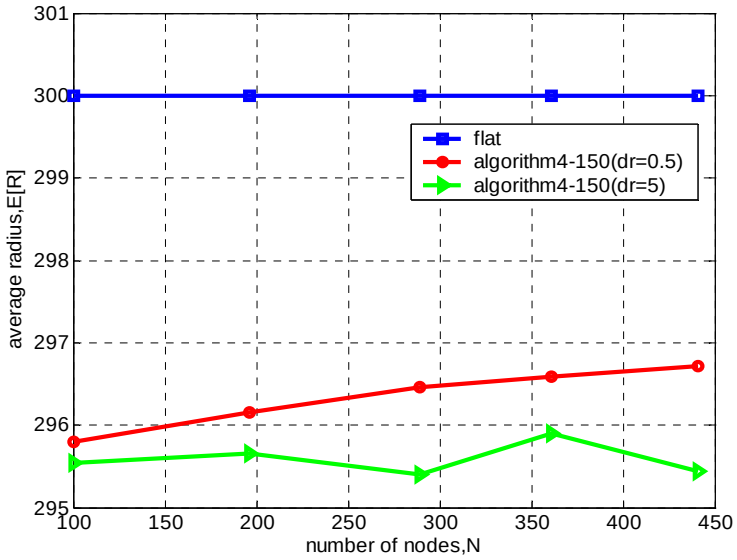
Μείωση του μέσου αριθμού γειτόνων που λόγω του εύρους τιμών δεν είναι δυνατόν να αναπαρασταθεί στην αυξανόμενη πυκνότητα, συμπεραίνουμε από το Διάγραμμα 31. Επίσης αύξηση του μέσου αριθμού γειτόνων με την αύξηση του αριθμού των κόμβων.




**Διάγραμμα 32. Αλγόριθμος IV(150)-Αυξανόμενη πυκνότητα-Διασπορά αριθμού γειτόνων**

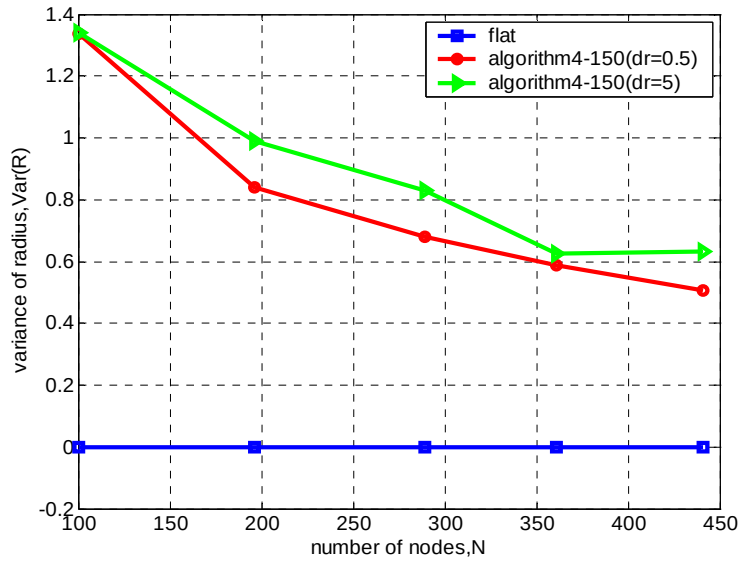
Η διασπορά στο Διάγραμμα 32 αυξάνεται όσο αυξάνεται το μέγεθος του δικτύου. Δεν υπάρχει μεταβολή από το αρχικό δίκτυο.

### 5.5.1.2 Σταθερή πυκνότητα



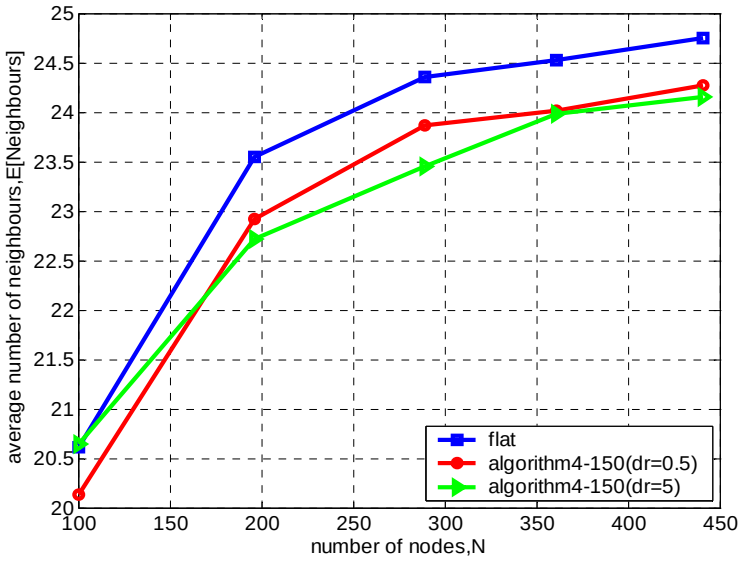

**Διάγραμμα 33. Αλγόριθμος IV(150)-Σταθερή πυκνότητα-Μέση ακτίνα**

Η τιμή της ακτίνας σταθεροποιείται στο Διάγραμμα 33. Παρατηρούμε ότι για το βήμα 5 υπάρχει μία διακύμανση. Οι μετρήσεις επαναλήφθηκαν και η διακύμανση παρέμεινε. Αυτό υποδηλώνει ότι στην ουσία η μέση τιμή ακτίνας όντως μπορεί να θεωρηθεί ότι είναι σταθερή διότι η διακύμανση που παρουσιάζει δεν ξεπερνά τη μία μονάδα ενώ το βήμα είναι 5.



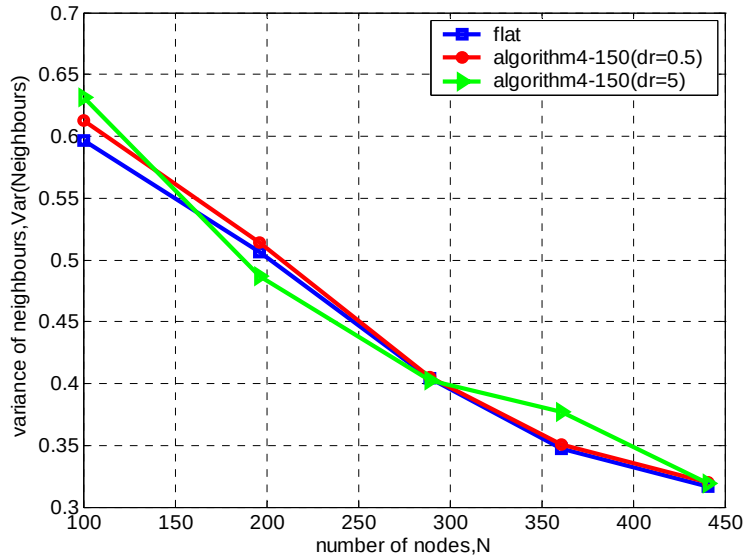

**Διάγραμμα 34. Αλγόριθμος IV(150)-Σταθερή πυκνότητα-Διασπορά ακτίνας**

Η διασπορά μειώνεται στο Διάγραμμα 34 όπως και στους υπόλοιπους αλγορίθμους στο αντίστοιχο διάγραμμα.




**Διάγραμμα 35. Αλγόριθμος IV(150)-Σταθερή πυκνότητα-Μέσος αριθμός γειτόνων**

Στο Διάγραμμα 35 διακρίνεται πλέον η ελάττωση του μέσου αριθμού γειτόνων που επιδιώκει ο αλγόριθμος.



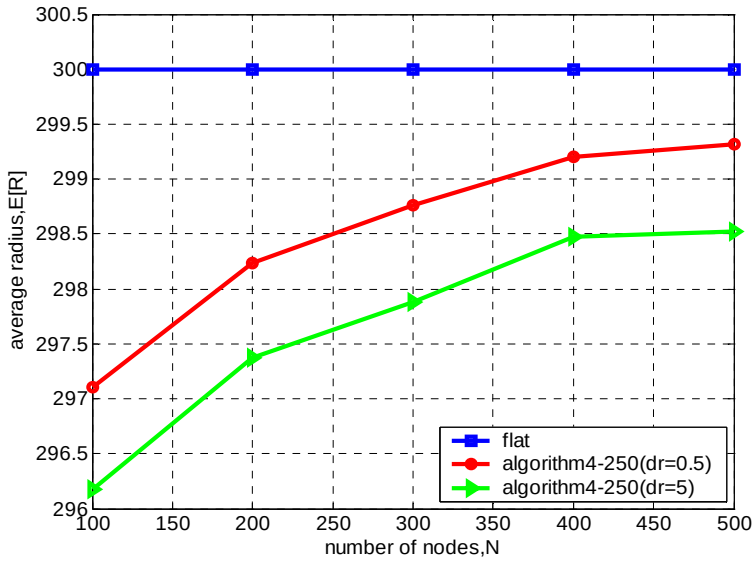

**Διάγραμμα 36. Αλγόριθμος IV(150)-Σταθερή πυκνότητα-Διασπορά αριθμού γειτόνων**

Η διασπορά του μέσου αριθμού γειτόνων ακολουθεί τη μορφή που είχε στο αρχικό δίκτυο στο Διάγραμμα 36.

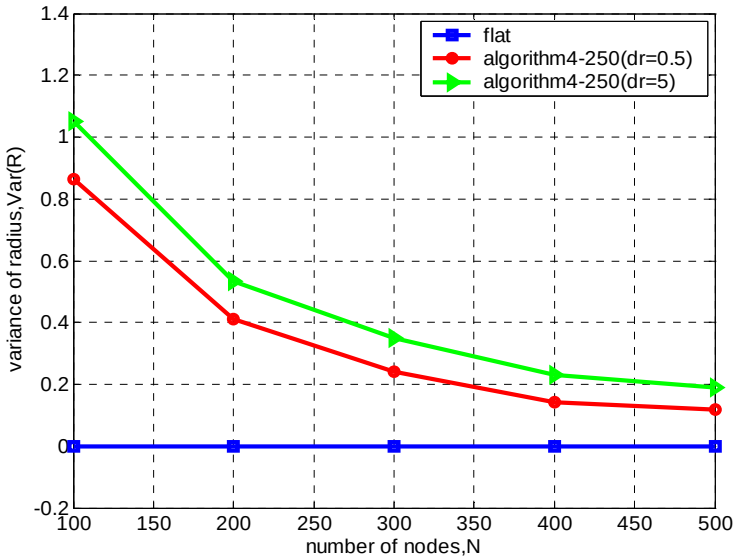
## 5.5.2 δ=250

Τα διαγράμματα 37-44 που ακολουθούν παρατίθενται απλά χωρίς σχολιασμό διότι έχουν ίδια μορφή με αυτά που προέκυψαν για δ=150.

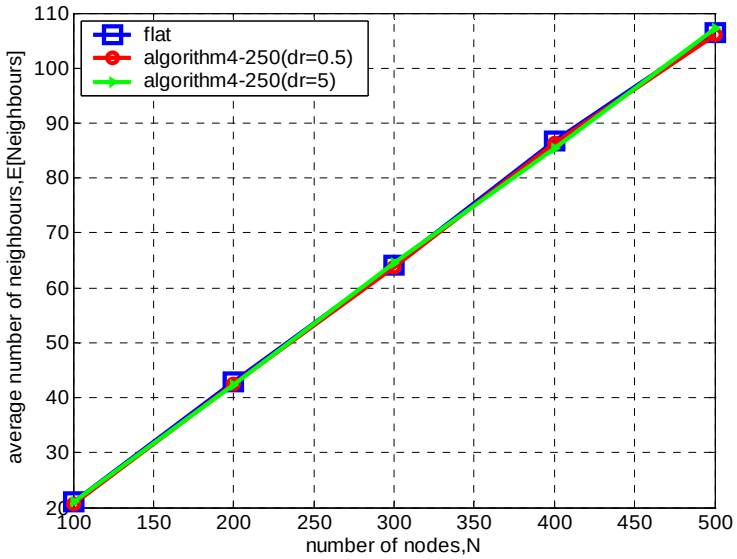
### 5.5.2.1 Αυξανόμενη πυκνότητα



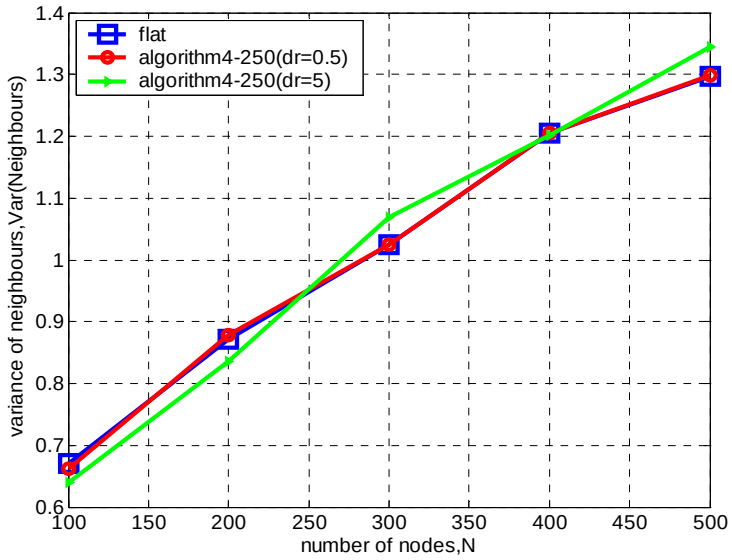

**Διάγραμμα 37. Αλγόριθμος IV(250)-Αυξανόμενη πυκνότητα-Μέση ακτίνα**




**Διάγραμμα 38. Αλγόριθμος IV(250)-Αυξανόμενη πυκνότητα-Διασπορά ακτίνας**

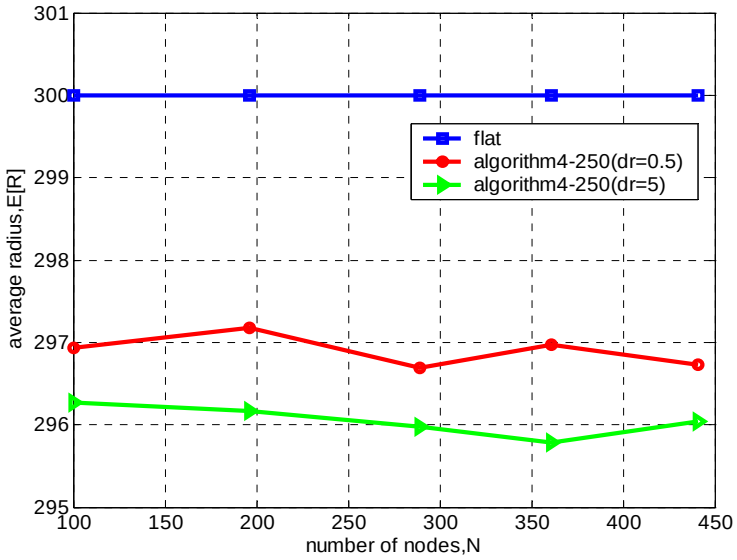



**Διάγραμμα 39. Αλγόριθμος IV(250)-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων**

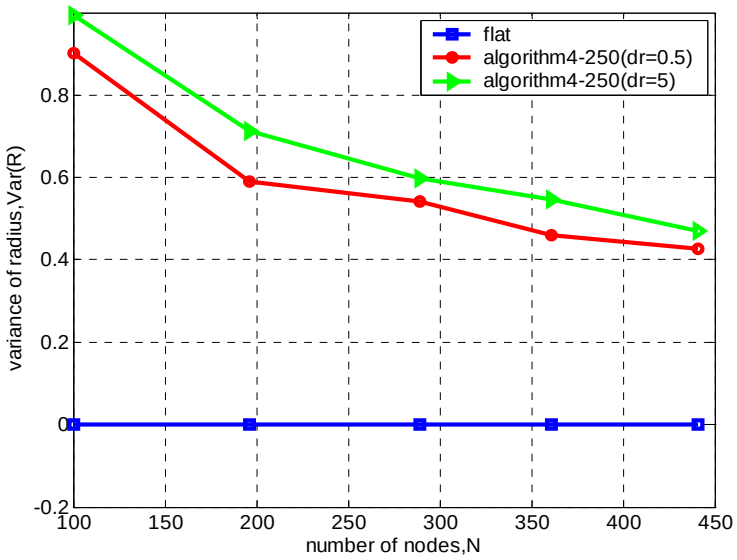



**Διάγραμμα 40. Αλγόριθμος IV(250)-Αυξανόμενη πυκνότητα-Διασπορά αριθμού γειτόνων**

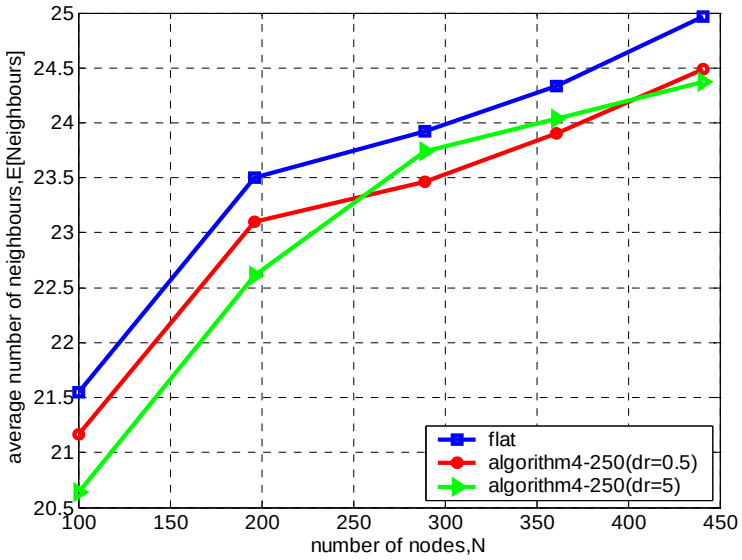
### 5.5.2.2 Σταθερή πυκνότητα



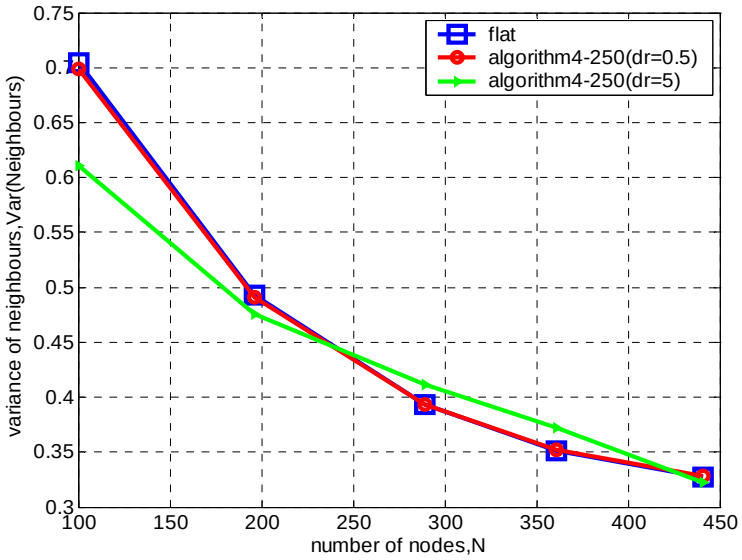

**Διάγραμμα 41. Αλγόριθμος IV(250)-Σταθερή πυκνότητα-Μέση ακτίνα**




**Διάγραμμα 42. Αλγόριθμος IV(250)-Σταθερή πυκνότητα-Διασπορά ακτίνας**



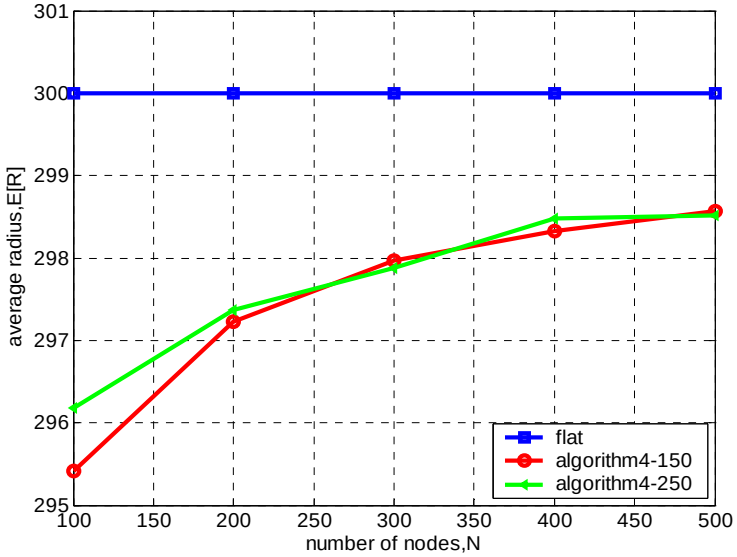

**Διάγραμμα 43. Αλγόριθμος IV(250)-Σταθερή πυκνότητα-Μέσος αριθμός γειτόνων**




**Διάγραμμα 44. Αλγόριθμος IV(250)-Σταθερή πυκνότητα-Διασπορά αριθμού γειτόνων**

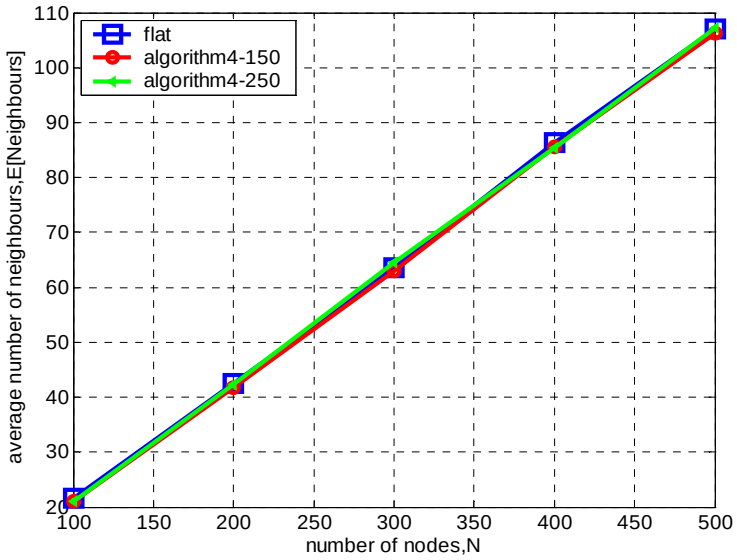
## 5.6 Συγκριτικά των IV με διαφορετικά δ ( $dr = 5$ )

### 5.6.1 Αυξανόμενη πυκνότητα




**Διάγραμμα 45. Συγκριτικό IV(150) και IV(250)-Αυξανόμενη πυκνότητα-Μέση ακτίνα**

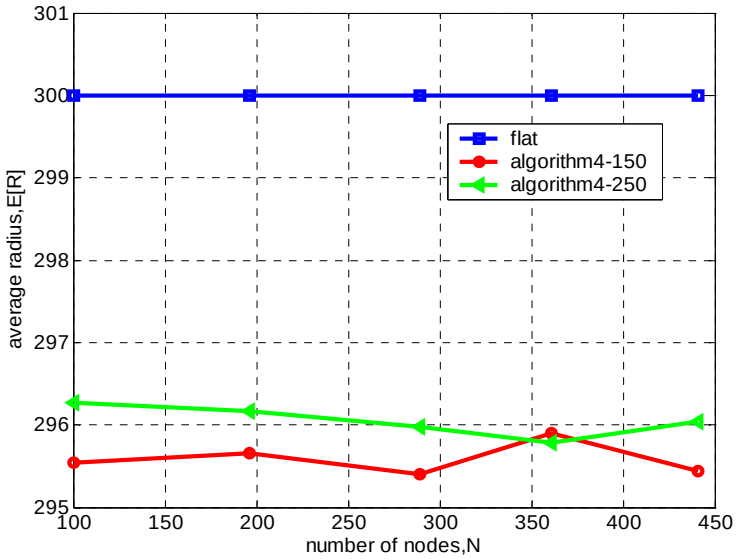
Στο Διάγραμμα 45 βλέπουμε ότι η μέση ακτίνα που προκύπτει με κατώφλι 250 είναι μεγαλύτερη από τη μέση ακτίνα για κατώφλι 150 για όλες τις τιμές αριθμού κόμβων. Αυτό ήταν αναμενόμενο διότι όταν το κατώφλι είναι μεγαλύτερο όλες οι ακτίνες που πρέπει να τεθούν στο κατώφλι λόγω του αλγορίθμου αυξάνουν τελικά το μέσο όρο.




**Διάγραμμα 46. Συγκριτικό IV(150) και IV(250)-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων**

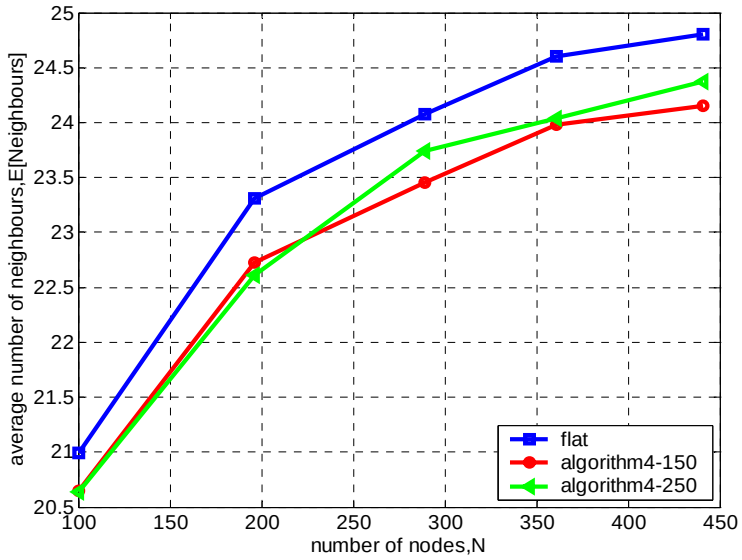
Οι διαφορές μεταξύ των δύο αλγορίθμων στο μέσο αριθμό γειτόνων δεν είναι ευδιάκριτες στο Διάγραμμα 46. Θα μελετηθούν πιο αναλυτικά στην σταθερή πυκνότητα.

## 5.6.2 Σταθερή πυκνότητα




**Διάγραμμα 47. Συγκριτικό IV(150) και IV(250)-Σταθερή πυκνότητα-Μέση ακτίνα**

Φαίνεται και από το Διάγραμμα 47 ότι η μέση ακτίνα για κατώφλι 250 είναι μεγαλύτερη από αυτήν για κατώφλι 150 αλλά παίρνει πιο σταθερές τιμές καθώς ο αριθμός των κόμβων αυξάνεται.

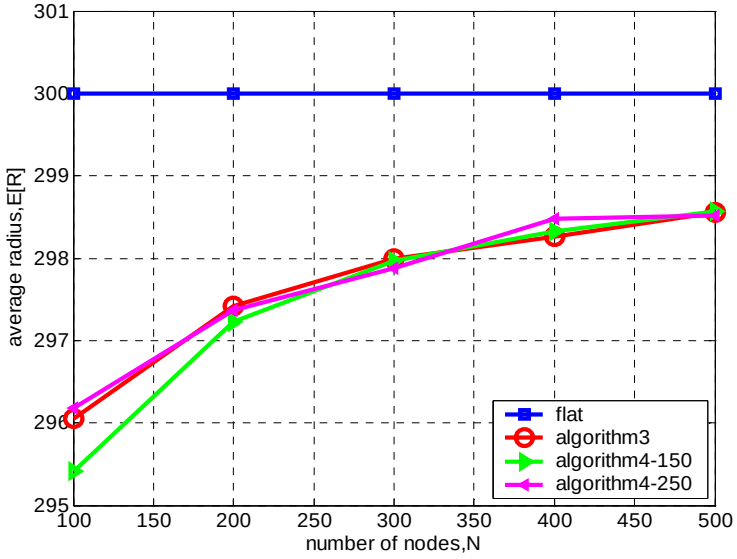



**Διάγραμμα 48. Συγκριτικό IV(150) και IV(250)-Σταθερή πυκνότητα-Μέσος αριθμός γειτόνων**

Ο μέσος αριθμός γειτόνων στο Διάγραμμα 48 είναι οριακά μεγαλύτερος για κατώφλι 250.

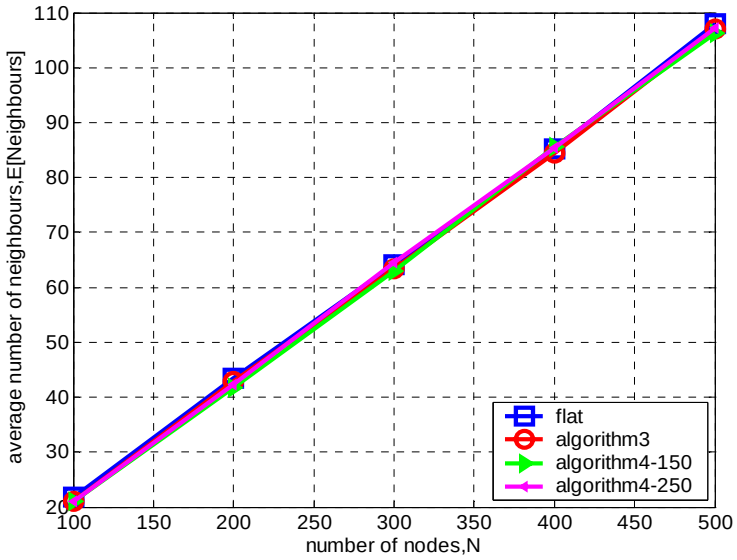
## 5.7 Συγκριτικά για αλγορίθμους III και IV

### 5.7.1 Αυξανόμενη πυκνότητα




**Διάγραμμα 49. Συγκριτικό III και IV-Αυξανόμενη πυκνότητα-Μέση ακτίνα**

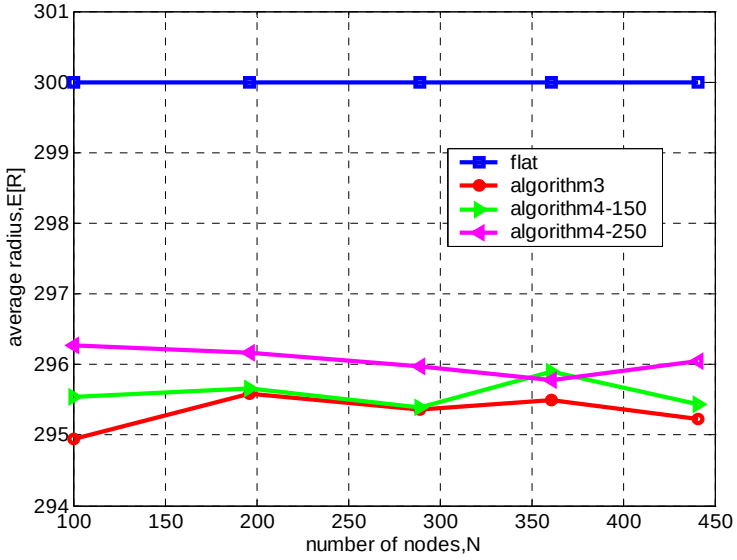
Οι τιμές των μέσων ακτινών και για τους τρεις αλγορίθμους φαίνεται στο Διάγραμμα 49 να κυμαίνονται στα ίδια επίπεδα. Πιο ξεκάθαρος διαχωρισμός θα γίνει στη σταθερή πυκνότητα.




**Διάγραμμα 50. Συγκριτικό III και IV-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων**

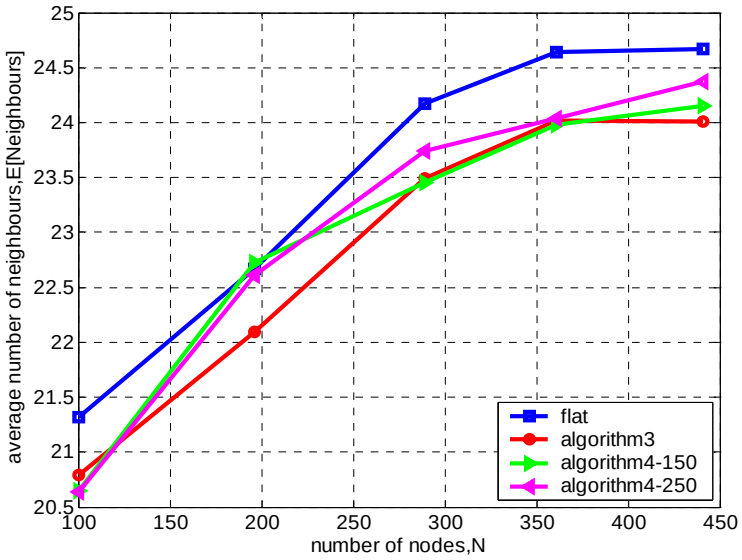
Το ίδιο ακριβώς συμβαίνει και στο Διάγραμμα 50.

## 5.7.2 Σταθερή πυκνότητα




**Διάγραμμα 51. Συγκριτικό III και IV-Σταθερή πυκνότητα-Μέση ακτίνα**

Στο Διάγραμμα 51 διακρίνεται πλέον ότι η μικρότερη μέση ακτίνα επιτυγχάνεται από τον αλγόριθμο III. Η αμέσως καλύτερη τιμή επιτυγχάνεται από τον αλγόριθμο IV με κατώφλι 150. Ο χειρότερος αποδεικνύεται ο IV με κατώφλι 250.




**Διάγραμμα 52. Συγκριτικό III και IV-Σταθερή πυκνότητα-Μέσος αριθμός γειτόνων**

Τα ίδια ακριβώς παρατηρούμε και στο Διάγραμμα 52 σχετικά με το μέσο όρο αριθμού γειτόνων. Καλύτερος λοιπόν σχετικά και με τα δύο μεγέθη αποδεικνύεται ο III από τον IV.

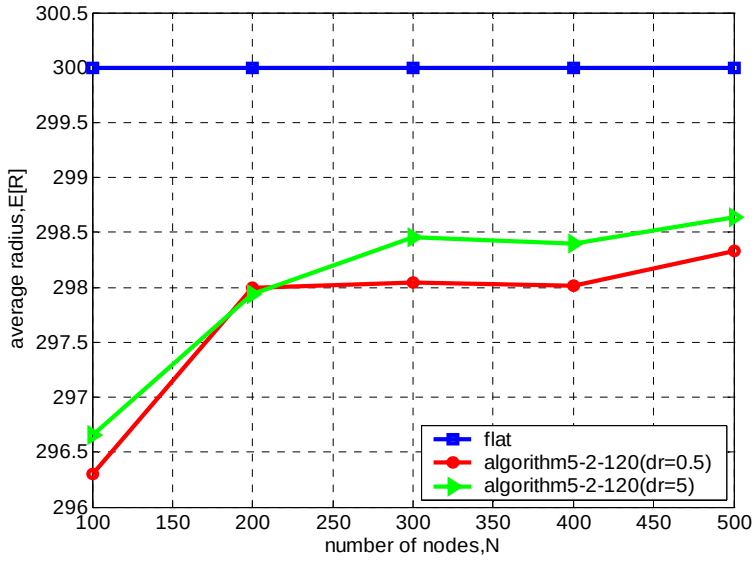
## 5.8 Αλγόριθμος V

Έγιναν μετρήσεις για δύο γωνίες $\alpha$. Για $a = 120^\circ$ και $a = 90^\circ$. Για κάθε γωνία έγιναν μετρήσεις μόνο για τη φάση 1 του αλγορίθμου και για φάση 1 και 2 μαζί. Θεωρήθηκε απαραίτητο να γίνουν μετρήσεις για δύο τιμές της παραμέτρου $q$. Για $q = 1$ και $q = 2$. Τέλος επιχειρήθηκε να γίνει και μία αντιστοίχηση των αποτελεσμάτων που προέκυψαν με τα αποτελέσματα που παρουσιάστηκαν στο [2].

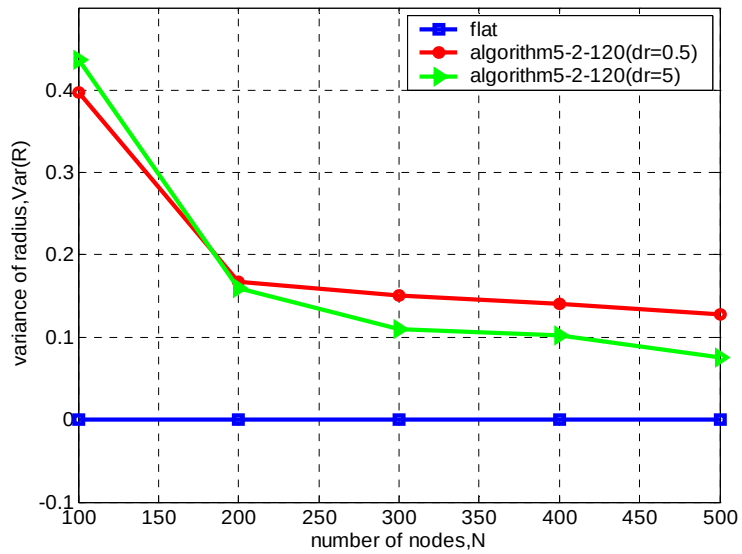
### 5.8.1 Φάση 2 με α=120° ($q = 2$)

#### 5.8.1.1 Αυξανόμενη πυκνότητα

Η μέση ακτίνα στο Διάγραμμα 53 είναι αισθητά μειωμένη σε σχέση με το αρχικό δίκτυο. Επίσης αυξάνεται με την αύξηση του αριθμού των κόμβων. Οι λόγοι έχουν αναφερθεί στα προηγούμενα διαγράμματα.

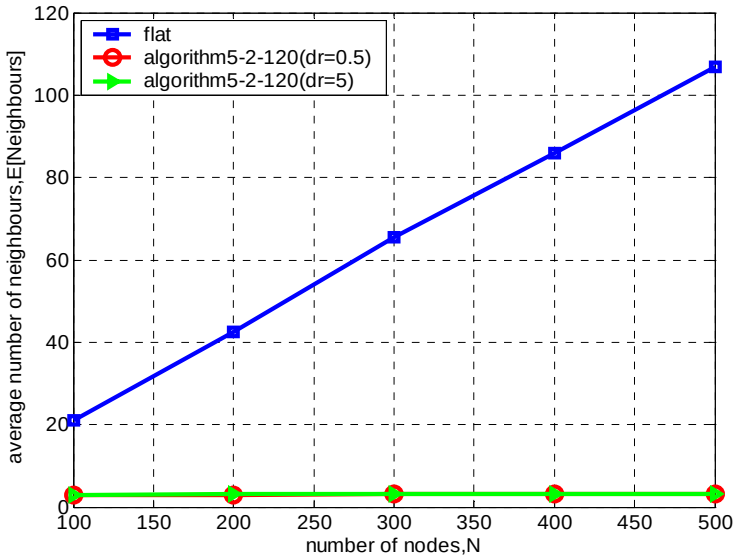



**Διάγραμμα 53. Αλγόριθμος V(2-120)-Αυξανόμενη πυκνότητα-Μέση ακτίνα**



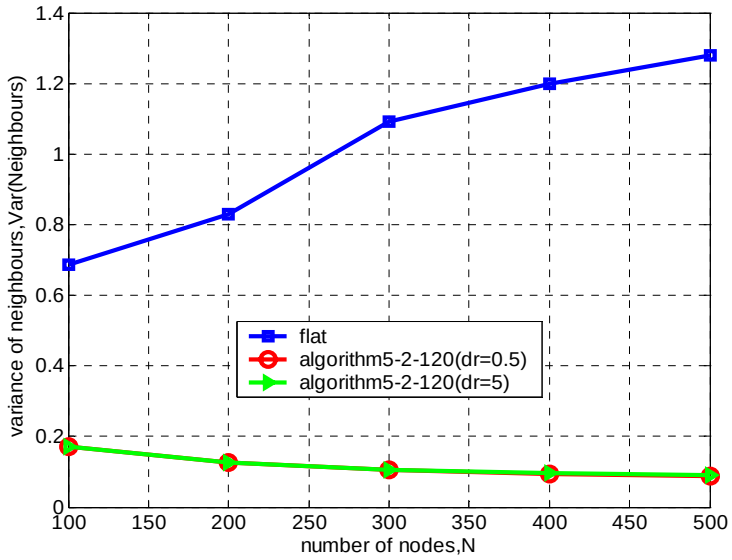

**Διάγραμμα 54. Αλγόριθμος V(2-120)-Αυξανόμενη πυκνότητα-Διασπορά ακτίνας**

Στο Διάγραμμα 54 η διασπορά της μέσης ακτίνας μειώνεται με την αύξηση της πυκνότητας.




**Διάγραμμα 55. Αλγόριθμος V(2-120)-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων**

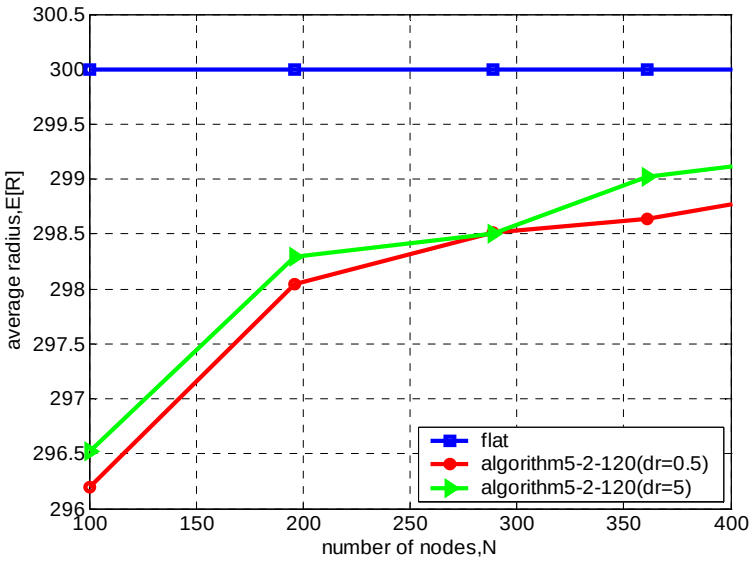
Στο Διάγραμμα 55 βλέπουμε μία σημαντική διαφοροποίηση από τα αντίστοιχα προηγούμενα Ο μέσος αριθμός γειτόνων μετά την εφαρμογή του αλγορίθμου έχει μειωθεί δραματικά. Σε αυτό συνέβαλε η θεώρηση να εφαρμοστεί η φάση 2 του αλγορίθμου και στους edge nodes. Αυτό είχε ως αποτέλεσμα να μειωθεί πολύ και ο αριθμός γειτόνων των edge node και επομένως και ο γενικός μέσος όρος.




**Διάγραμμα 56. Αλγόριθμος V(2-120)-Αυξανόμενη πυκνότητα-Διασπορά αριθμού γειτόνων**

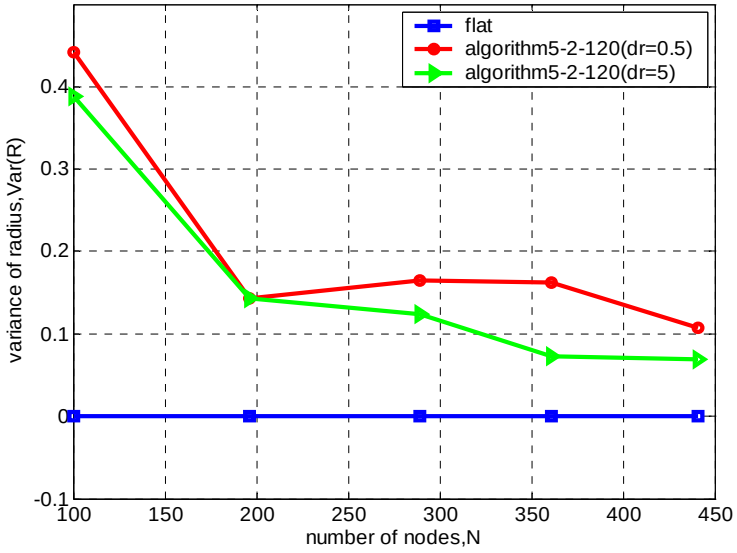
Η διασπορά μέσου αριθμού γειτόνων στο Διάγραμμα 56 μειώνεται με την αύξηση του αριθμού κόμβων και είναι πολύ μικρότερη από τη διασπορά του αρχικού δικτύου. Αυτό έχει σχέση με το γεγονός ότι στα αποτελέσματα παρατηρήθηκε μικρή διαφοροποίηση μεταξύ των τιμών που προέκυπταν για τους γείτονες κάθε κόμβου. Επομένως και οι τιμές αυτές απείχαν πολύ λίγο από τη μέση τιμή και η διασπορά προέκυψε πολύ μικρή.

### 5.8.1.2 Σταθερή πυκνότητα



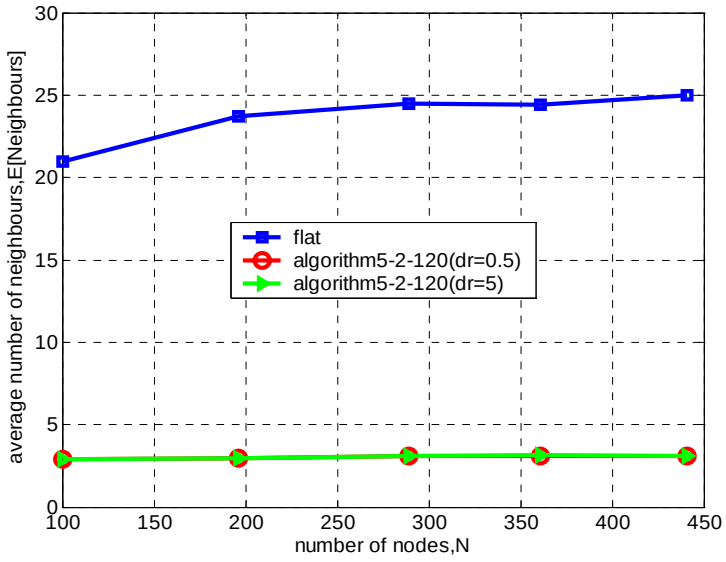

**Διάγραμμα 57. Αλγόριθμος V(2-120)-Σταθερή πυκνότητα-Μέση ακτίνα**

Στο Διάγραμμα 57 η μέση ακτίνα αυξάνεται με την αύξηση του μεγέθους του δικτύου. Είναι μειωμένη σε σχέση με τη μέση ακτίνα στο αρχικό δίκτυο.



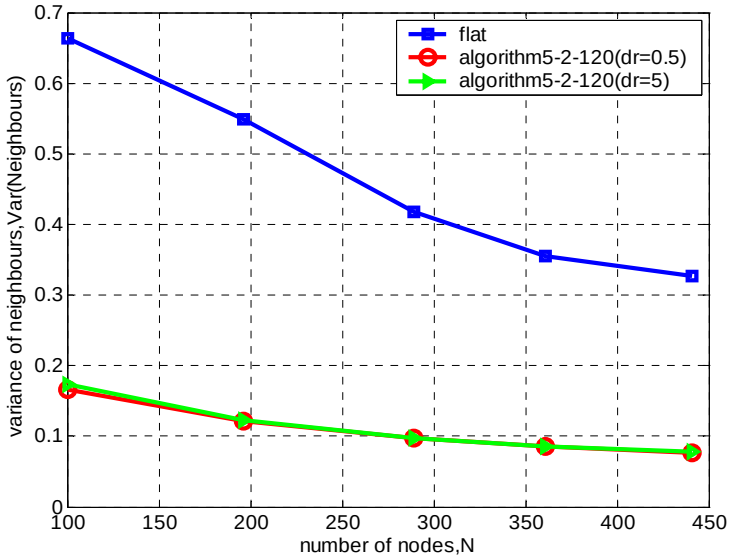

**Διάγραμμα 58. Αλγόριθμος V(2-120)-Σταθερή πυκνότητα-Διασπορά ακτίνας**

Στο Διάγραμμα 58 η διασπορά μειώνεται με την αύξηση του μεγέθους του δικτύου.




**Διάγραμμα 59. Αλγόριθμος V(2-120)-Σταθερή πυκνότητα-Μέσος αριθμός γειτόνων**

Ο μέσος αριθμός γειτόνων στο Διάγραμμα 59 παραμένει σε πολύ χαμηλά επίπεδα και με σταθερές τιμές, ανεξάρτητα από την αύξηση του αριθμού των κόμβων.

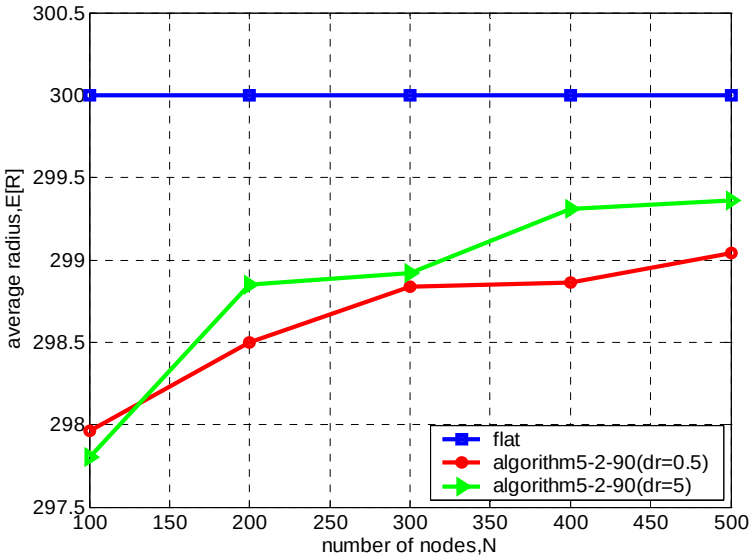



**Διάγραμμα 60. Αλγόριθμος V(2-120)-Σταθερή πυκνότητα-Διασπορά αριθμού γειτόνων**

Η διασπορά στο Διάγραμμα 60 είναι πολύ μικρή και με την αύξηση του μεγέθους του δικτύου μειώνεται.

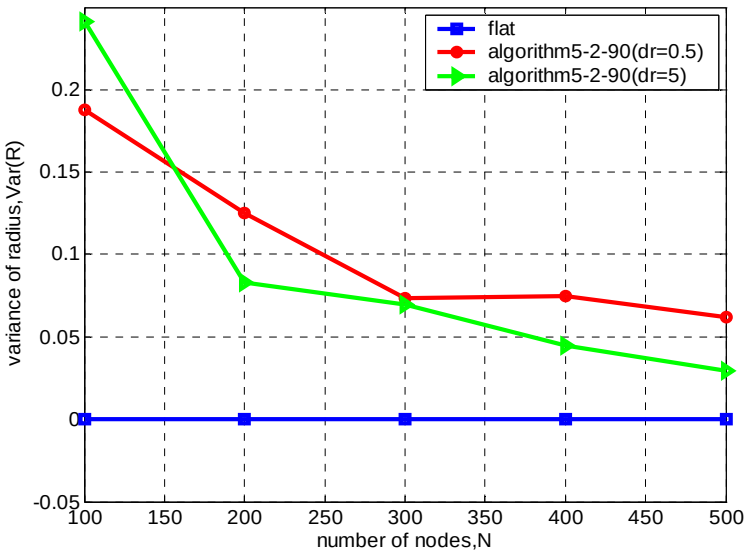
## 5.8.2 Φάση 2 με α=90° ($q = 2$)

### 5.8.2.1 Αυξανόμενη πυκνότητα



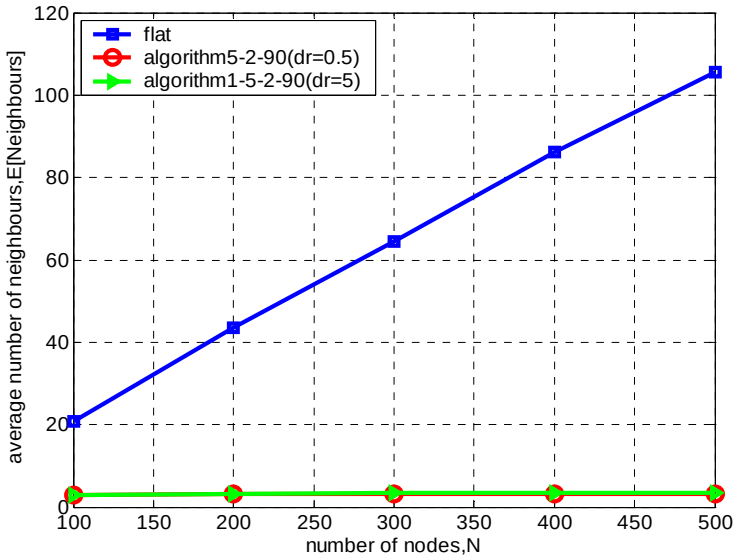

**Διάγραμμα 61. Αλγόριθμος V(2-90)-Αυξανόμενη πυκνότητα-Μέση ακτίνα**

Παρατηρούμε στο Διάγραμμα 61 αύξηση της μέσης ακτίνας με την αύξηση του αριθμού των κόμβων στο τελικό δίκτυο. Η εφαρμογή του αλγορίθμου έχει ως αποτέλεσμα τη μείωση της μέσης ακτίνας, η οποία στο αρχικό δίκτυο είχε τη μέγιστη τιμή.



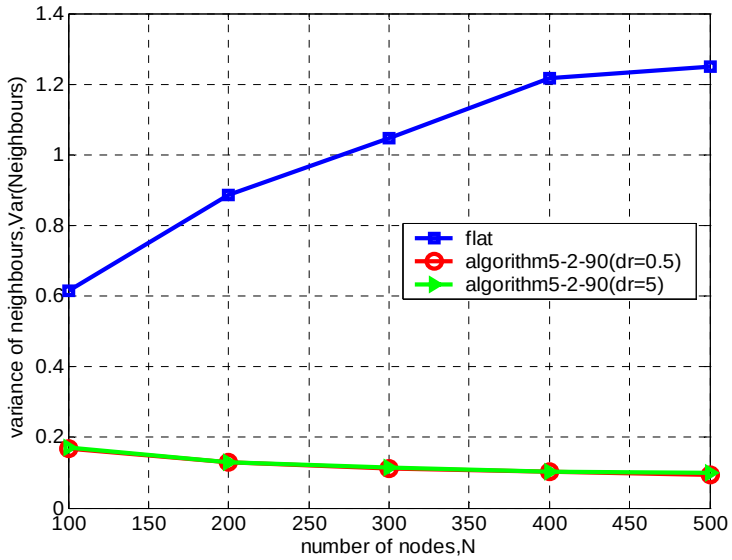

**Διάγραμμα 62. Αλγόριθμος V(2-90)-Αυξανόμενη πυκνότητα-Διασπορά ακτίνας**

Η διασπορά της μέσης ακτίνας μειώνεται με την αύξηση του αριθμού των κόμβων στο Διάγραμμα 62 όπως και σε όλους τους προηγούμενους αλγορίθμους.




**Διάγραμμα 63. Αλγόριθμος V(2-90)-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων**

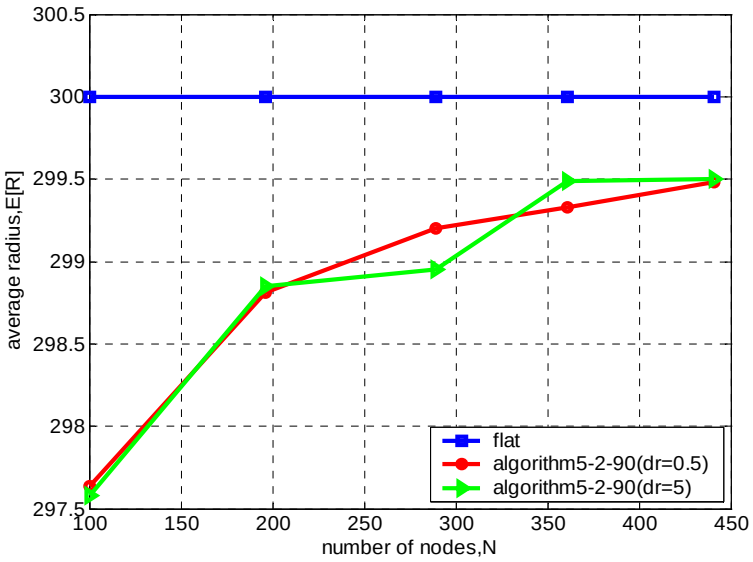
Ο μέσος αριθμός γειτόνων στο Διάγραμμα 63 εμφανίζεται δραστικά μειωμένος όπως συνέβη και για $a = 120^{\circ}$.




**Διάγραμμα 64. Αλγόριθμος V(2-90)-Αυξανόμενη πυκνότητα-Διασπορά αριθμού γειτόνων**

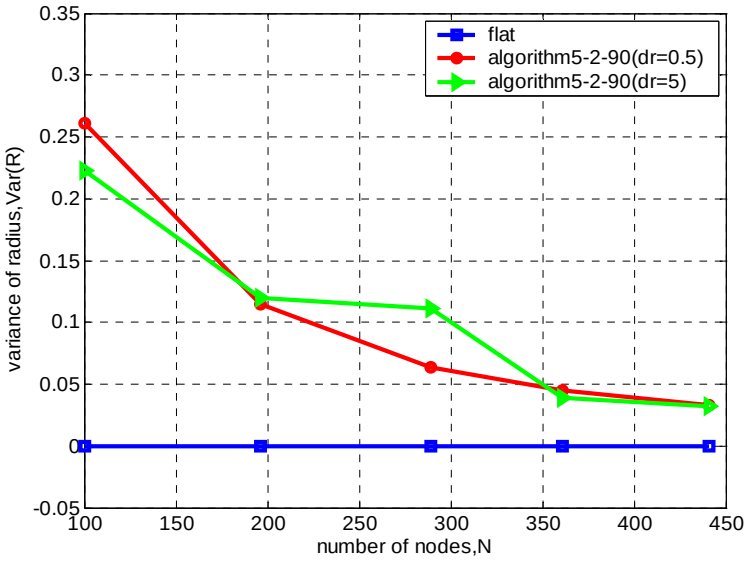
Η διασπορά του μέσου αριθμού γειτόνων παρουσιάζει στο Διάγραμμα 64 επίσης πολύ μεγάλη μείωση σε σχέση με αυτήν του αρχικού δικτύου.

### 5.8.2.2 Σταθερή πυκνότητα



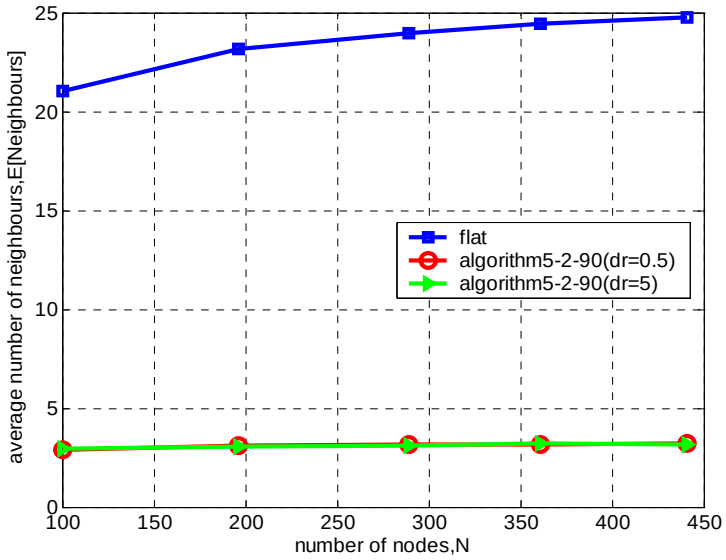

**Διάγραμμα 65. Αλγόριθμος V(2-90)-Σταθερή πυκνότητα-Μέση ακτίνα**

Η μέση ακτίνα έχει αυξητικές τάσεις στο Διάγραμμα 65 καθώς το μέγεθος του δικτύου μεγαλώνει.



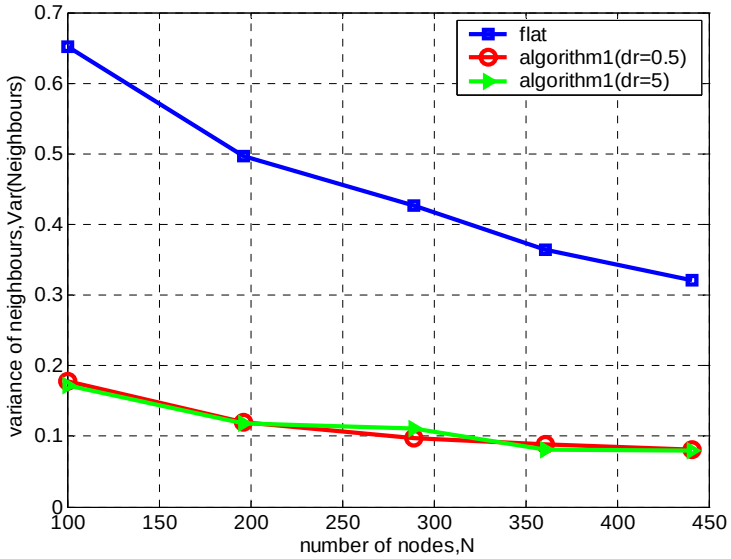

**Διάγραμμα 66. Αλγόριθμος V(2-90)-Σταθερή πυκνότητα-Διασπορά ακτίνας**

Η διασπορά μέσης ακτίνας μειώνεται στο Διάγραμμα 66 με την αύξηση του μεγέθους του δικτύου.




**Διάγραμμα 67. Αλγόριθμος V(2-90)-Σταθερή πυκνότητα-Μέσος αριθμός γειτόνων**

Ο μέσος αριθμός γειτόνων συμπεριφέρεται στο Διάγραμμα 67 όπως στην αυξανόμενη πυκνότητα με τη διαφορά ότι στο διάγραμμα μπορούμε να διακρίνουμε ευκολότερα ότι κυμαίνεται γύρω στο 3.




**Διάγραμμα 68. Αλγόριθμος V(2-90)-Σταθερή πυκνότητα-Διασπορά αριθμού γειτόνων**

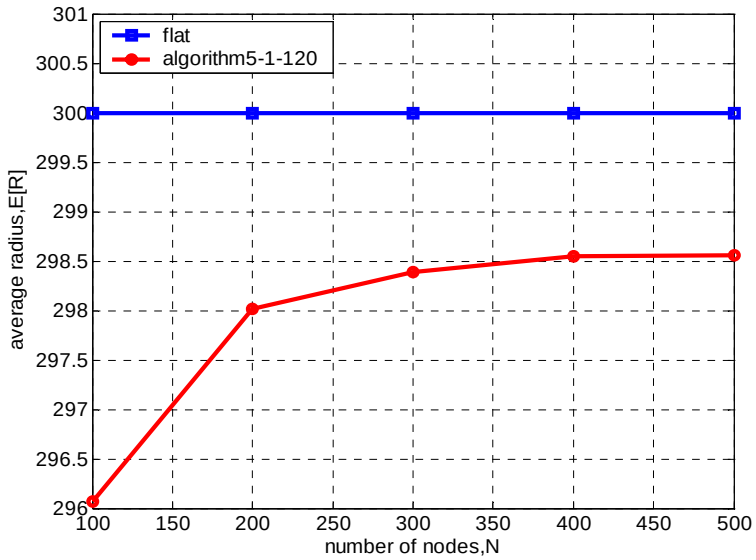
Στο Διάγραμμα 68 η διασπορά δεν διαφέρει επίσης από την αυξανόμενη.

### 5.8.3 Φάση 1 με α=120° ($q = 2, dr = 5$)

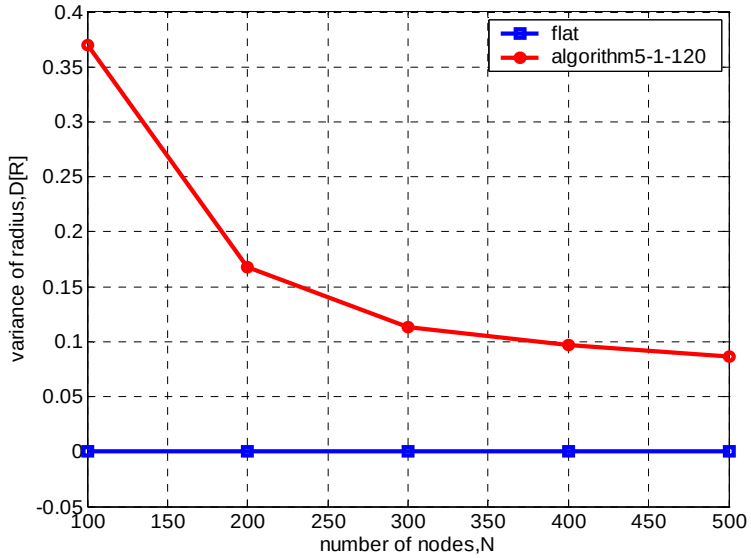
Οι μετρήσεις που έγιναν για τη φάση 1 του αλγορίθμου V, τόσο για $a = 120^\circ$ όσο και για $a = 90^\circ$, είχαν ως αποτέλεσμα να προκύψουν γραφικές παραστάσεις με μορφή

όμοια με αυτή των τεσσάρων πρώτων αλγορίθμων. Η μεγάλη διαφορά σε σχέση με τη φάση δύο του αλγορίθμου V παρατηρείται στο μέσο αριθμό γειτόνων και τη διασπορά του μέσου αριθμού γειτόνων όπου πλέον δεν παρατηρούνται οι πολύ μικρές τιμές που είδαμε στις αμέσως προηγούμενες γραφικές. Δηλαδή τελικά η φάση δύο είναι καθοριστική για την ελάττωση του αριθμού των γειτόνων. Παρατίθενται στη συνέχεια οι γραφικές παραστάσεις για τις δύο τιμές του $a$, για αυξανόμενη και σταθερή πυκνότητα μόνο για τη φάση ένα, χωρίς περαιτέρω σχολιασμό για τα διαγράμματα 69-84 που ακολουθούν.

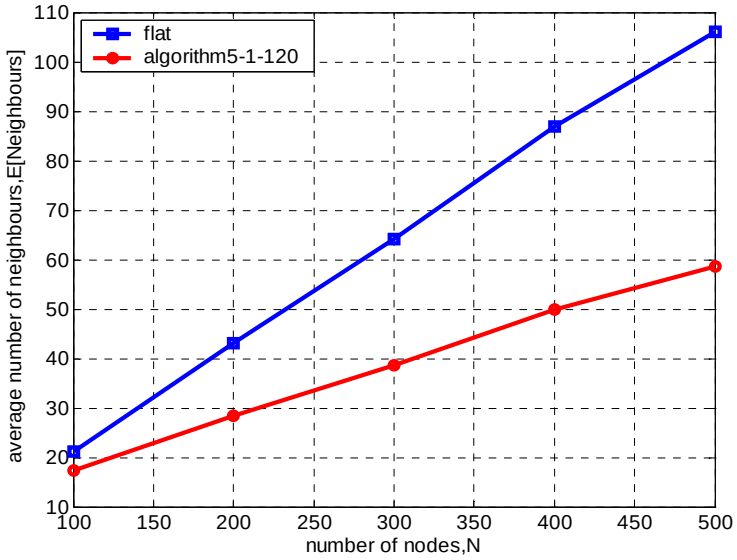
### 5.8.3.1 Αυξανόμενη πυκνότητα



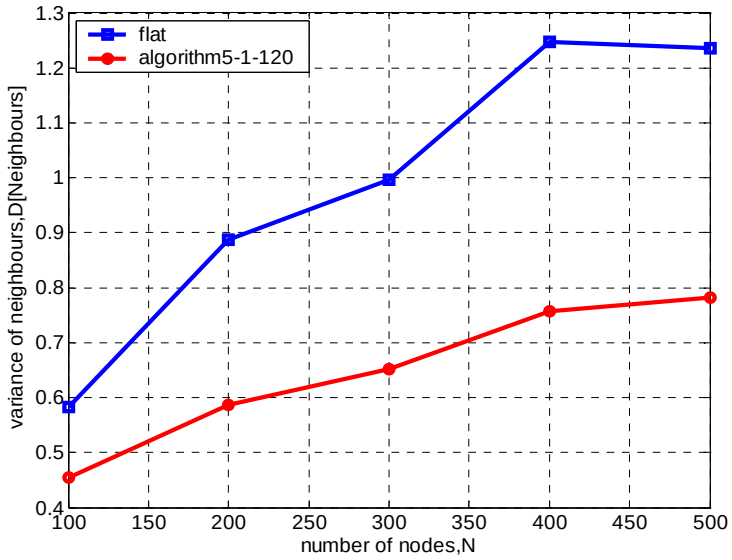

**Διάγραμμα 69. Αλγόριθμος V(1-120)-Αυξανόμενη πυκνότητα-Μέση ακτίνα**




**Διάγραμμα 70. Αλγόριθμος V(1-120)-Αυξανόμενη πυκνότητα-Διασπορά ακτίνας**

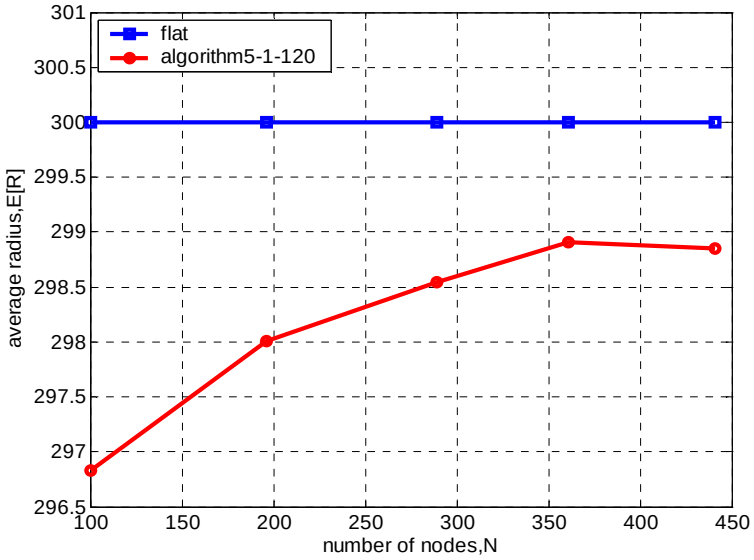



**Διάγραμμα 71. Αλγόριθμος V(1-120)-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων**

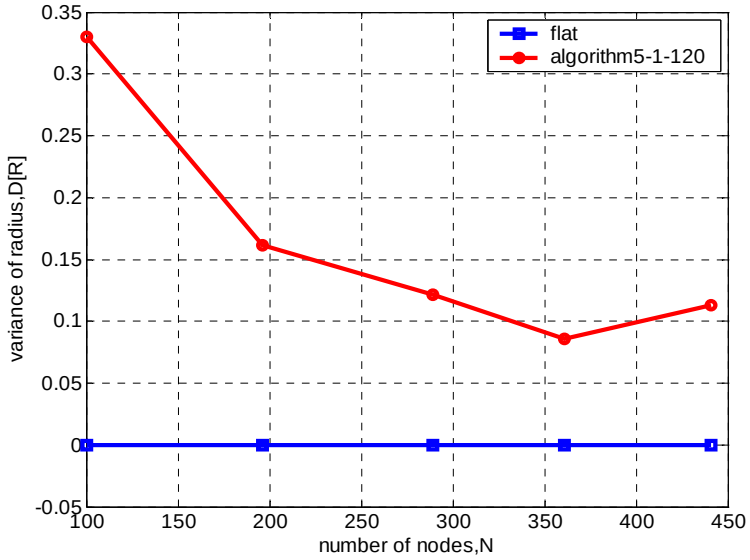



**Διάγραμμα 72. Αλγόριθμος V(1-120)-Αυξανόμενη πυκνότητα-Διασπορά αριθμού γειτόνων**

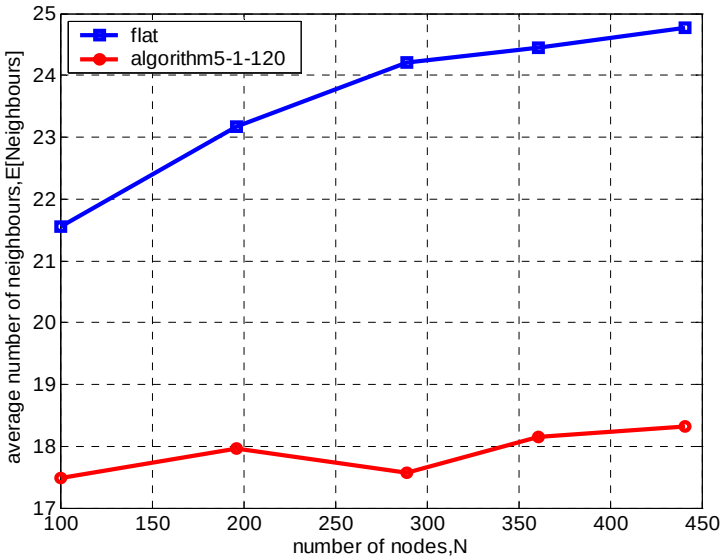
### 5.8.3.2 Σταθερή πυκνότητα



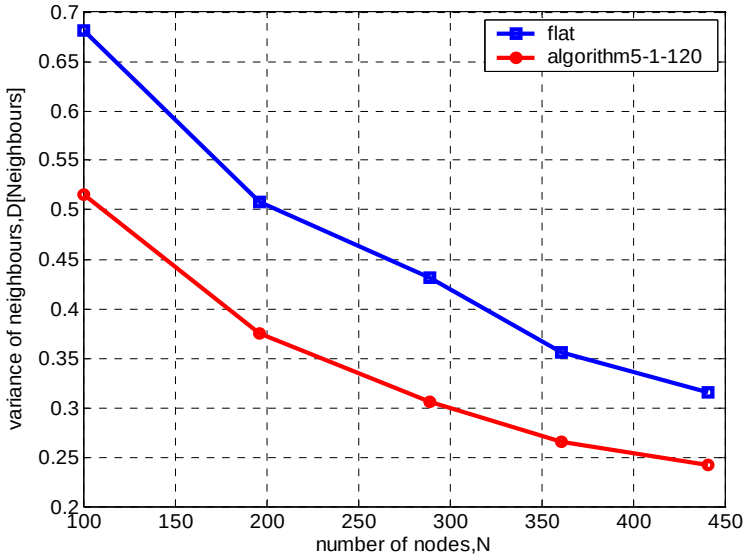

**Διάγραμμα 73. Αλγόριθμος V(1-120)-Σταθερή πυκνότητα-Μέση ακτίνα**




**Διάγραμμα 74. Αλγόριθμος V(1-120)-Σταθερή πυκνότητα-Διασπορά ακτίνας**



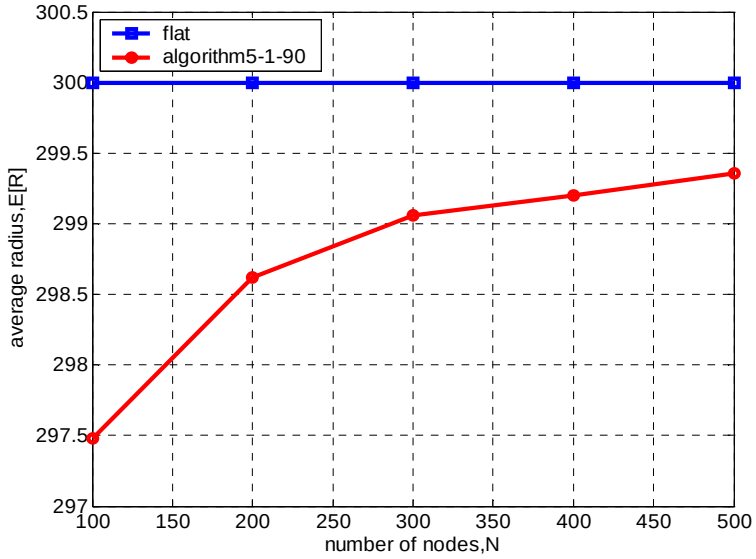

**Διάγραμμα 75. Αλγόριθμος V(1-120)-Σταθερή πυκνότητα-Μέσος αριθμός γειτόνων**



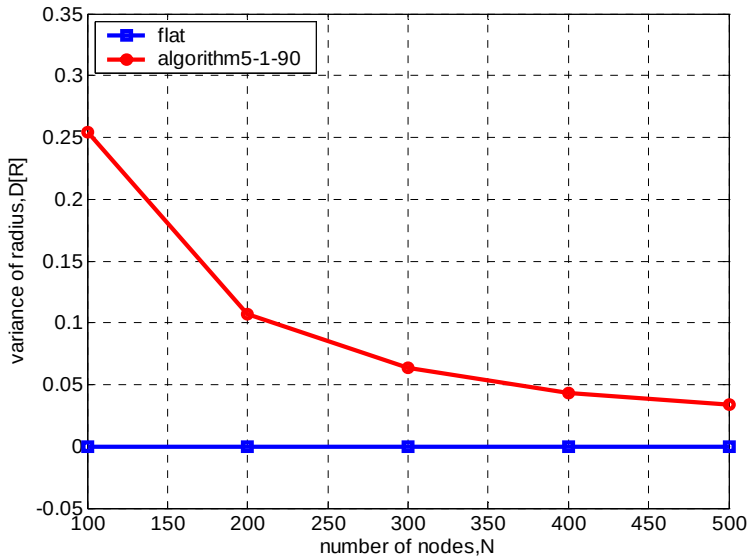

**Διάγραμμα 76. Αλγόριθμος V(1-120)-Σταθερή πυκνότητα-Διασπορά αριθμού γειτόνων**

### 5.8.4 Φάση 1 με α=90° ($q = 2, dr = 5$)

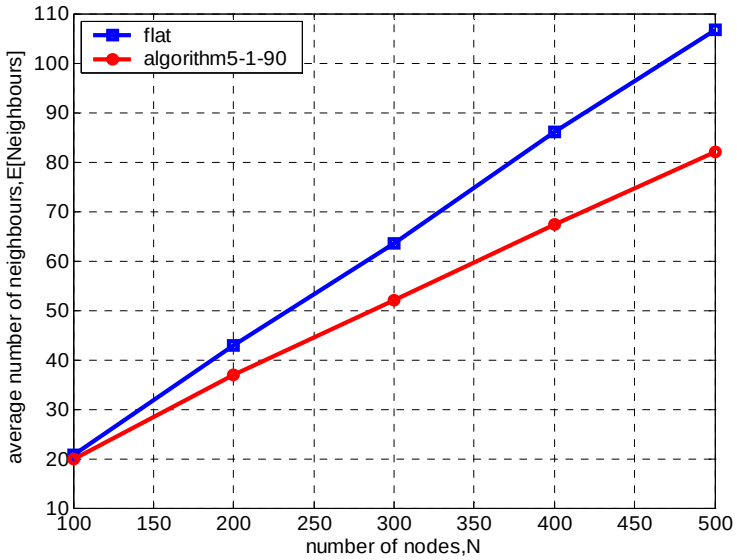
#### 5.8.4.1 Αυξανόμενη πυκνότητα



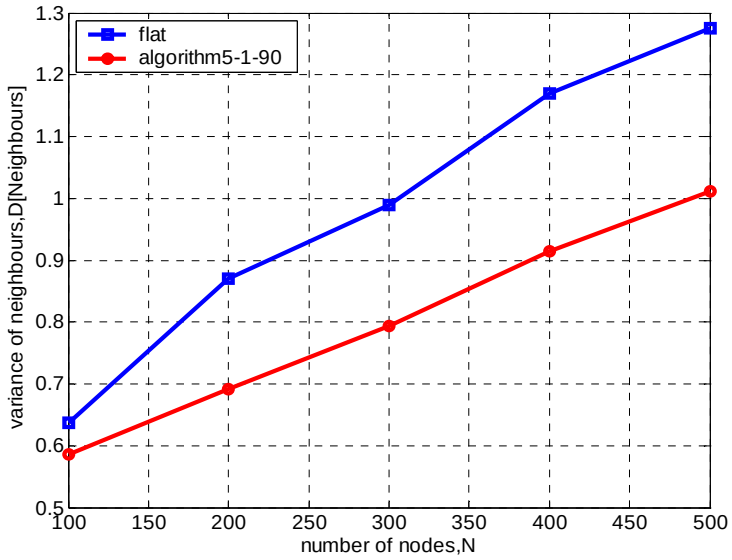

**Διάγραμμα 77. Αλγόριθμος V(1-90)-Αυξανόμενη πυκνότητα-Μέση ακτίνα**




**Διάγραμμα 78. Αλγόριθμος V(1-90)-Αυξανόμενη πυκνότητα-Διασπορά ακτίνας**

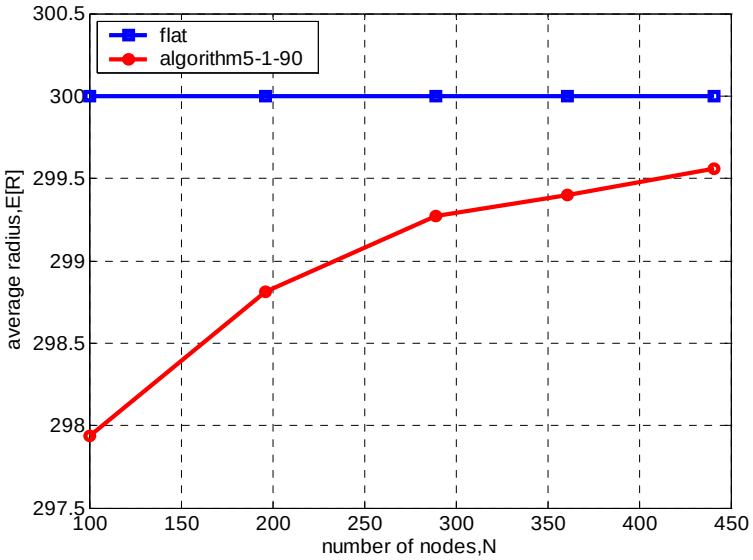



**Διάγραμμα 79. Αλγόριθμος V(1-90)-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων**

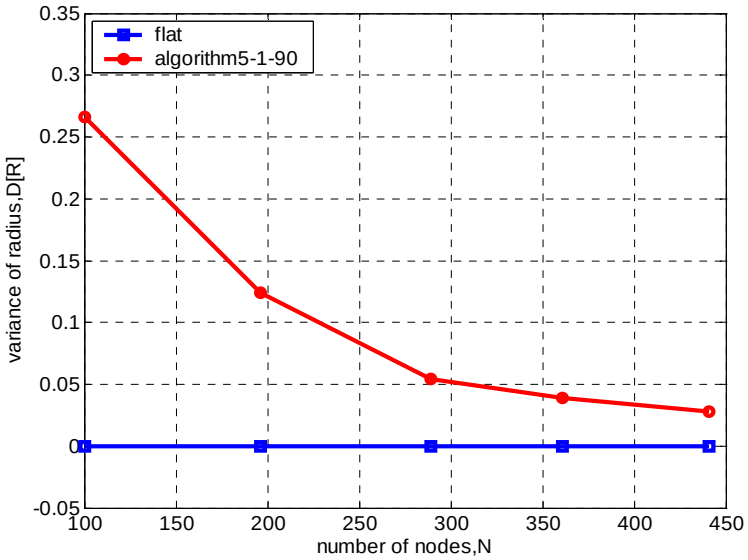



**Διάγραμμα 80. Αλγόριθμος V(1-90)-Αυξανόμενη πυκνότητα-Διασπορά αριθμού γειτόνων**

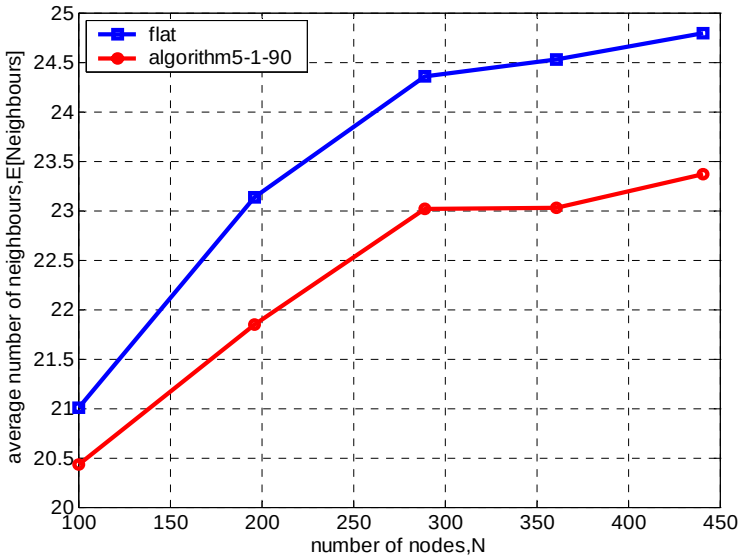
### 5.8.4.2 Σταθερή πυκνότητα



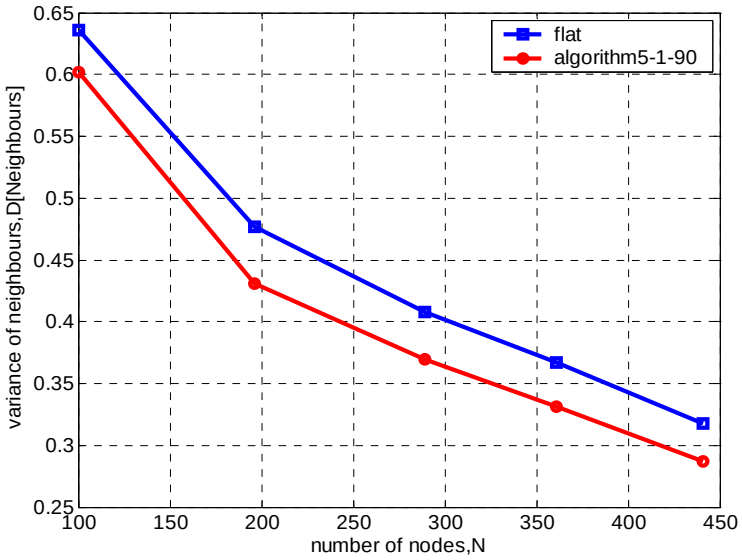

**Διάγραμμα 81. Αλγόριθμος V(1-90)-Σταθερή πυκνότητα-Μέση ακτίνα**




**Διάγραμμα 82. Αλγόριθμος V(1-90)-Σταθερή πυκνότητα-Διασπορά ακτίνας**




**Διάγραμμα 83. Αλγόριθμος V(1-90)-Σταθερή πυκνότητα-Μέσος αριθμός γειτόνων**



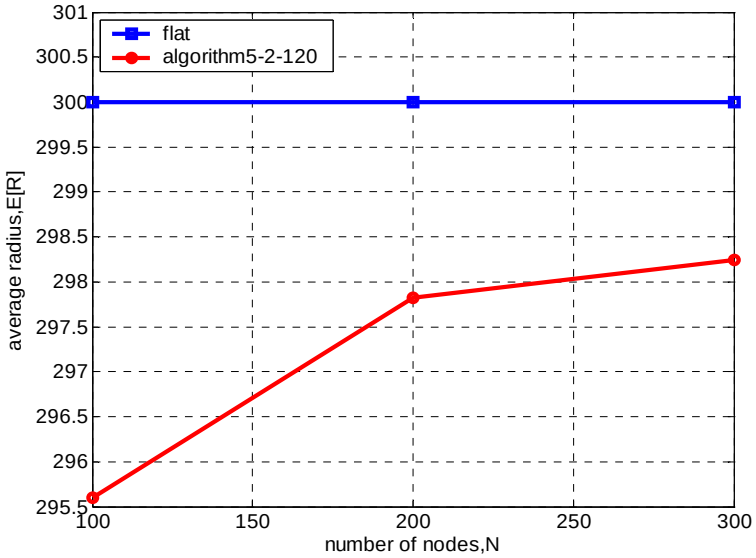

**Διάγραμμα 84. Αλγόριθμος V(1-90)-Σταθερή πυκνότητα-Διασπορά αριθμού γειτόνων**

### 5.8.5 Φάση 2 με α=120° ($q=1, dr=5$)

Κρίθηκε απαραίτητο να γίνουν μετρήσεις και για $q=1$ ώστε να διαπιστωθεί αν προκύπτουν εξίσου καλά αποτελέσματα όπως με $q=2$. Στις πρώτες απόπειρες που έγιναν διαπιστώθηκε ότι για τις πυκνότητες δικτύου που εξετάζονταν στους προηγούμενους αλγόριθμους δεν ήταν δυνατό με $q=1$ να ικανοποιηθεί η συνθήκη που ήταν απαραίτητη για να εκτελέσει ο αλγόριθμος τη φάση 2. Ουσιαστικά δηλαδή

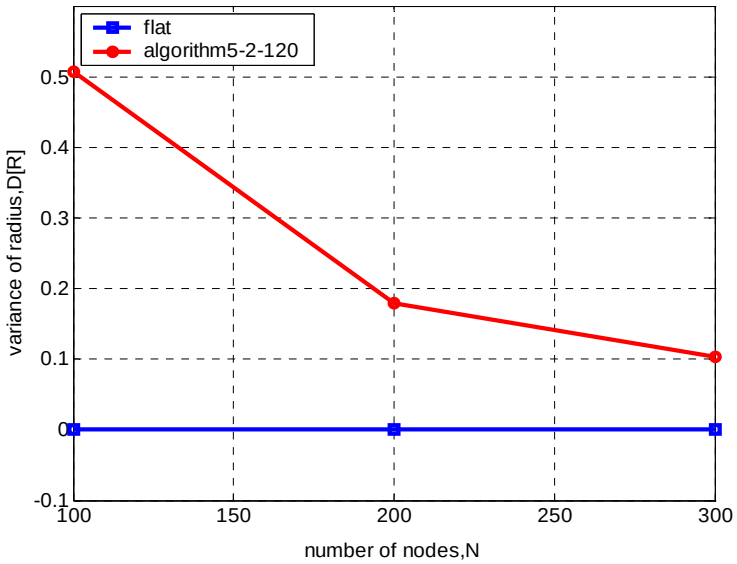
η ανισότητα $d(u,v)+d(v,w)\le q\cdot d(u,w)$ δεν ικανοποιούνταν ποτέ και ο αλγόριθμος δεν έβρισκε επιπλέον γείτονες για να αφαιρέσει. Έγιναν διάφορες δοκιμαστικές προσπάθειες για τον εντοπισμό των πυκνοτήτων εκείνων στις οποίες η ανισότητα ικανοποιείται και τελικά μπορεί να πραγματοποιηθεί αφαίρεση ορισμένου αριθμού γειτόνων. Τελικά επιλέχθηκαν τα μεγέθη *500*, *900* και *1200* για τη γεωγραφική περιοχή και τα μεγέθη *100*, *200* και *300* για τον αριθμό των κόμβων. Είναι εμφανές ότι ήταν απαραίτητο να αυξηθεί αρκετά η πυκνότητα του δικτύου για να μπορέσει να ικανοποιηθεί η ανισότητα.

### 5.8.5.1 Αυξανόμενη πυκνότητα



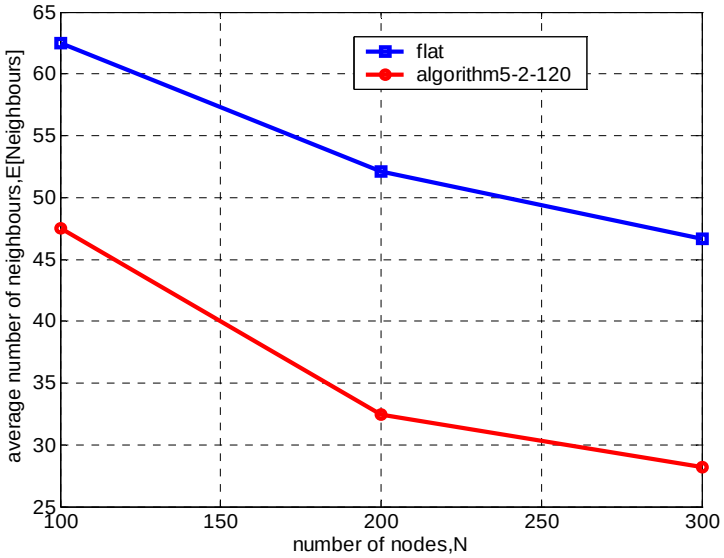

**Διάγραμμα 85. Αλγόριθμος V(2-120)-q=1-Αυξανόμενη πυκνότητα-Μέση ακτίνα**

Στο Διάγραμμα 85 η μέση τιμή ακτίνας αυξάνεται με την αύξηση του αριθμού των κόμβων.



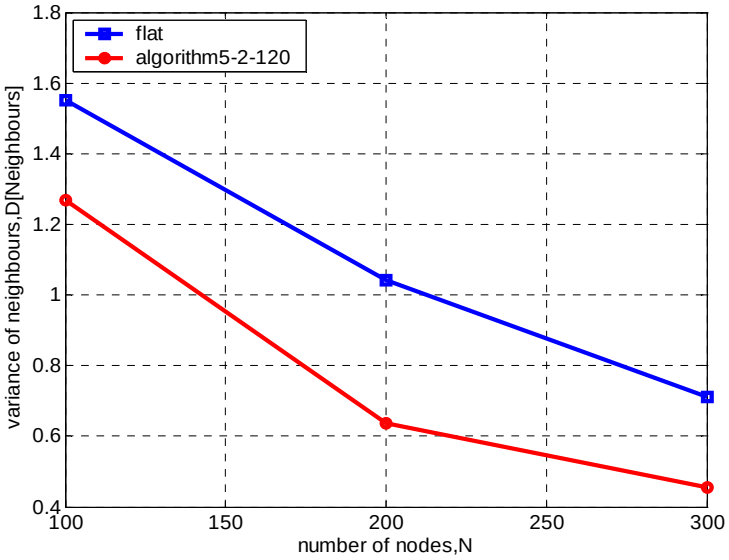

**Διάγραμμα 86. Αλγόριθμος V(2-120)-q=1-Αυξανόμενη πυκνότητα-Διασπορά ακτίνας**

Η διασπορά της μέσης ακτίνας μειώνεται με την αύξηση του αριθμού των κόμβων στο Διάγραμμα 86.




**Διάγραμμα 87. Αλγόριθμος V(2-120)-q=1-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων**

Παρατηρούμε στο Διάγραμμα 87 ότι ο μέσος αριθμός γειτόνων μειώνεται με την αύξηση του αριθμού των κόμβων. Αυτό είναι τελείως φαινομενικό διότι οι τιμές της πυκνότητας δικτύου επιλέχθηκαν τυχαία με μόνο γνώμονα το γεγονός ότι σε αυτές μπορούσε να ικανοποιηθεί η ανισότητα. Το σημαντικό συμπέρασμα που μπορεί να προκύψει είναι ότι για τις δεδομένες πυκνότητες δεν μειώνεται δραματικά ο αριθμός των γειτόνων, δηλαδή ακόμα και για τις δεδομένες σχετικά μεγάλες πυκνότητες με $q = 1$ ο αλγόριθμος δεν είναι τόσο αποτελεσματικός.

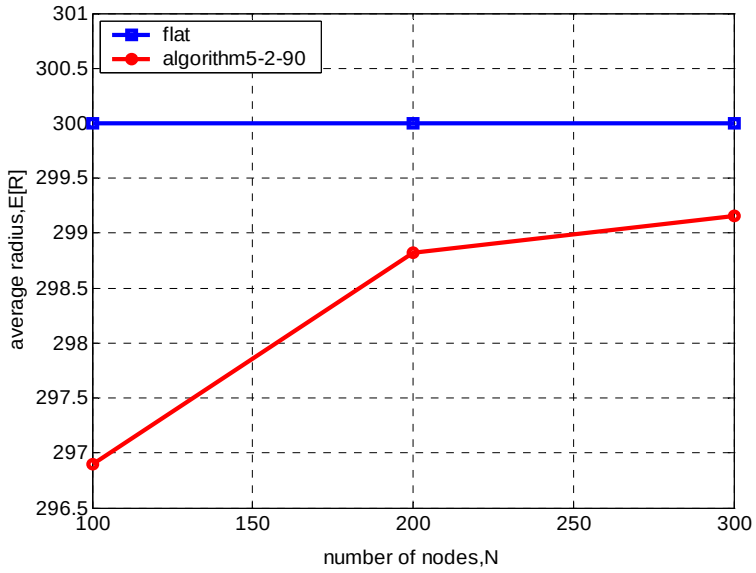



**Διάγραμμα 88. Αλγόριθμος V(2-120)-q=1-Αυξανόμενη πυκνότητα-Διασπορά αριθμού γειτόνων**

Η διασπορά μέσου αριθμού γειτόνων μειώνεται επίσης στο Διάγραμμα 88.

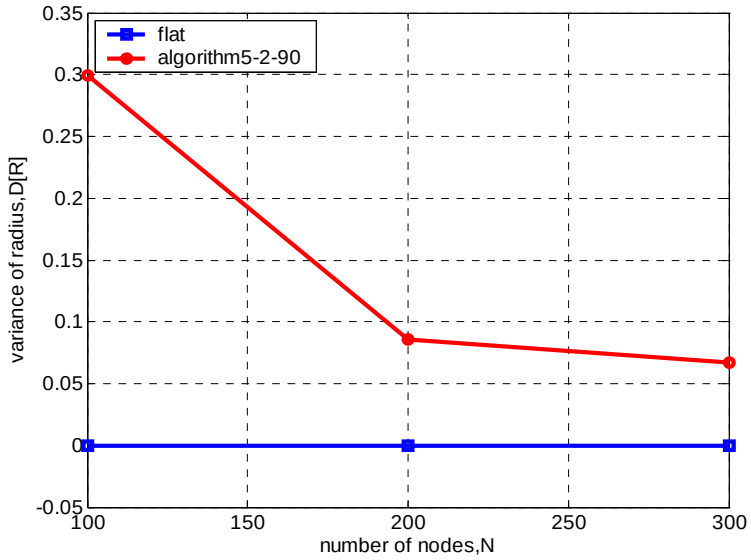
### 5.8.6 Φάση 2 με α=90° ($q=1, dr=5$)

#### 5.8.6.1 Αυξανόμενη πυκνότητα



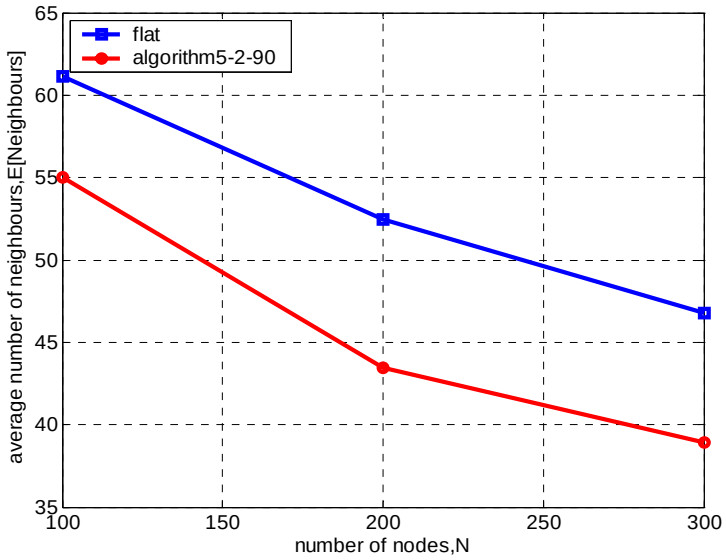

**Διάγραμμα 89. Αλγόριθμος V(2-90)-q=1-Αυξανόμενη πυκνότητα-Μέση ακτίνα**

Η μέση τιμή ακτίνας στο Διάγραμμα 89 αυξάνεται με την αύξηση του αριθμού των κόμβων όπως και πριν.



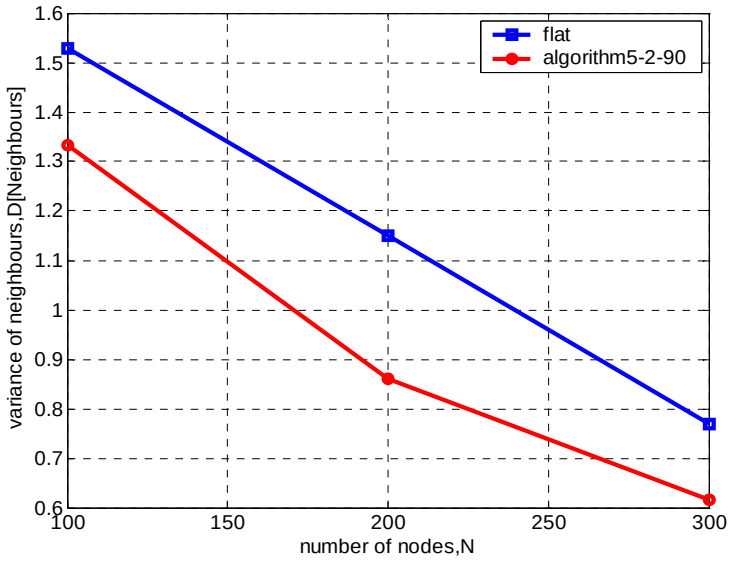

**Διάγραμμα 90. Αλγόριθμος V(2-90)-q=1-Αυξανόμενη πυκνότητα-Διασπορά ακτίνας**

Στο Διάγραμμα 90 η διασπορά της μέσης ακτίνας μειώνεται με την αύξηση του αριθμού των κόμβων.




**Διάγραμμα 91. Αλγόριθμος V(2-90)-q=1-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων**

Ισχύουν στο Διάγραμμα 91 οι παρατηρήσεις που έγιναν για το μέσο αριθμό γειτόνων για $a = 120^{\circ}$.

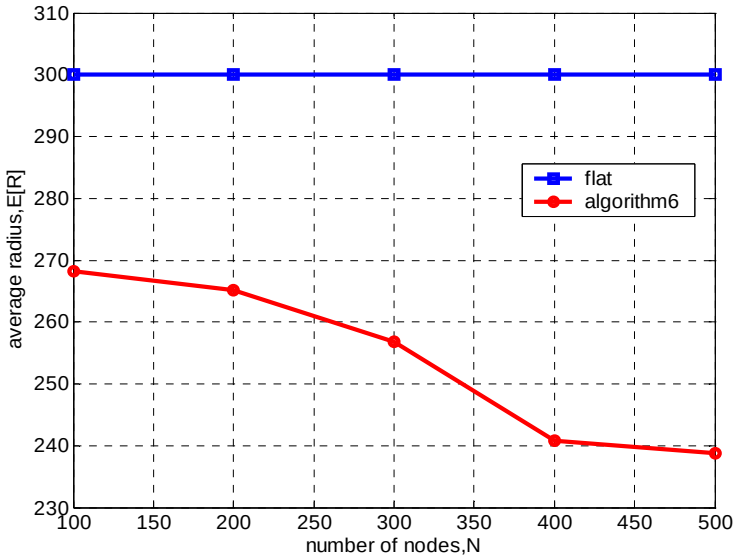



**Διάγραμμα 92. Αλγόριθμος V(2-90)-q=1-Αυξανόμενη πυκνότητα-Διασπορά αριθμού γειτόνων**

Το ίδιο και στο Διάγραμμα 92 για τη διασπορά του.

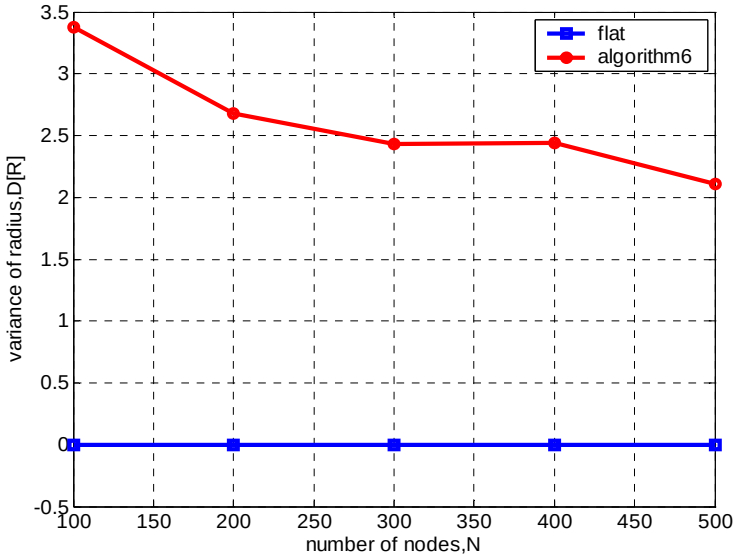
## 5.9 Αλγόριθμος RR ( $dr = 5$ )

### 5.9.1 Αυξανόμενη πυκνότητα



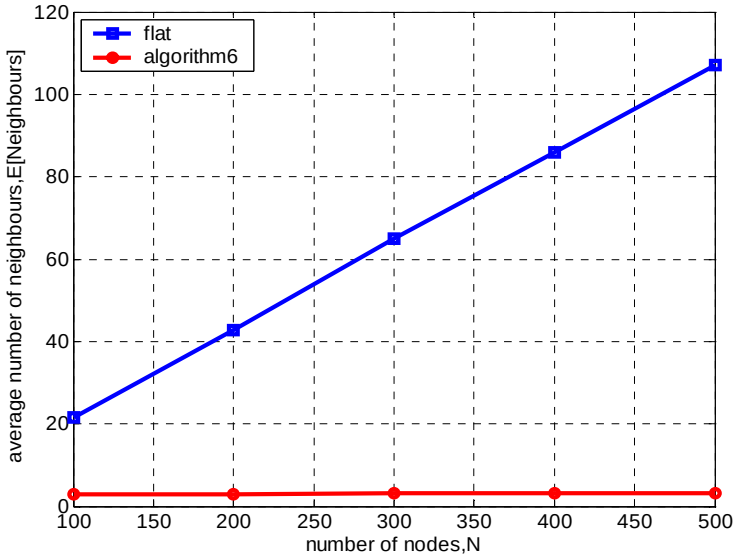

**Διάγραμμα 93. Αλγόριθμος RR-Αυξανόμενη πυκνότητα-Μέση ακτίνα**

Στο Διάγραμμα 93 η μέση ακτίνα μειώνεται με την αύξηση της πυκνότητας.



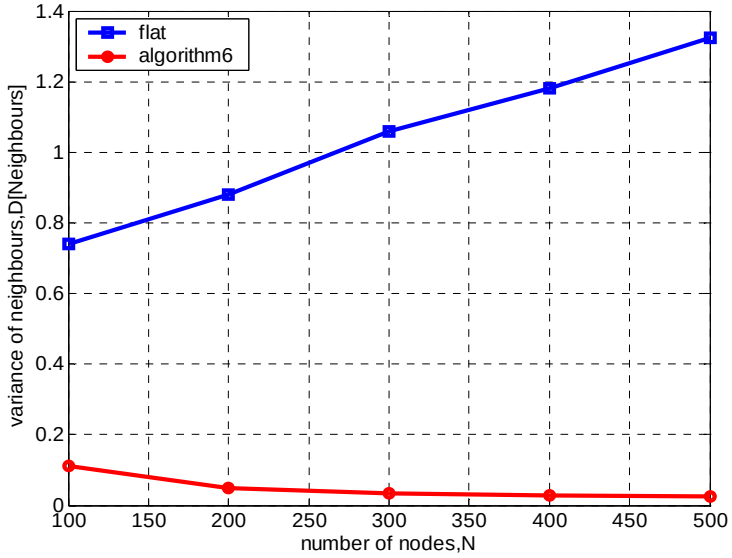

**Διάγραμμα 94. Αλγόριθμος RR-Αυξανόμενη πυκνότητα-Διασπορά ακτίνας**

Η διασπορά μέσης ακτίνας μειώνεται με την αύξηση του αριθμού των κόμβων στο Διάγραμμα 94.




**Διάγραμμα 95. Αλγόριθμος RR-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων**

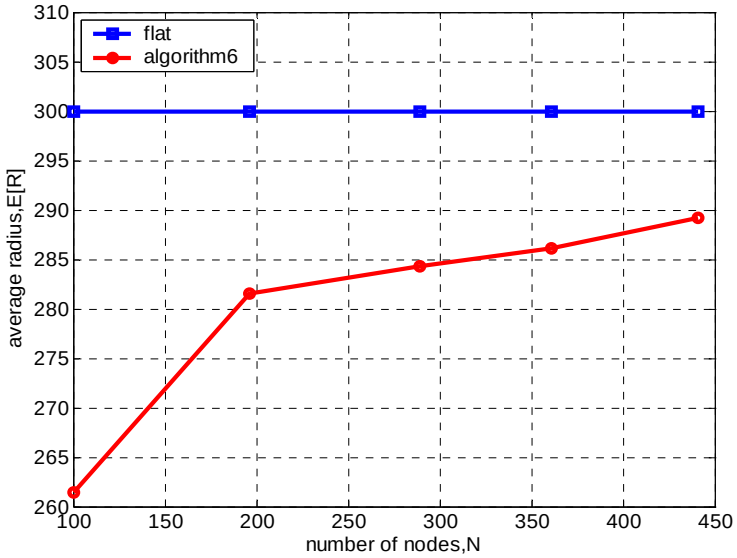
Το Διάγραμμα 95 δείχνει την αποτελεσματικότητα του αλγορίθμου RR στη μείωση του μέσου αριθμού κόμβων σε επίπεδα αντίστοιχα με του αλγορίθμου V για $q = 2$.




**Διάγραμμα 96. Αλγόριθμος RR-Αυξανόμενη πυκνότητα-Διασπορά αριθμού γειτόνων**

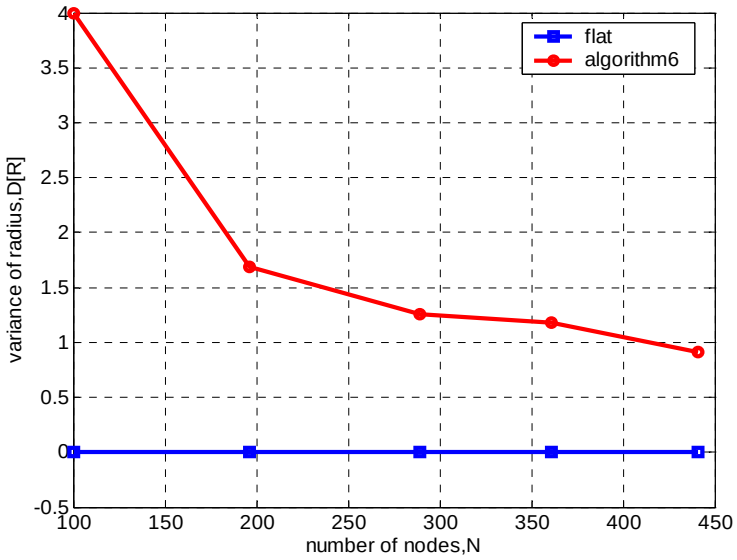
Η διασπορά μέσου αριθμού γειτόνων εμφανίζεται στο Διάγραμμα 96 επίσης εξαιρετικά μειωμένη.

## 5.9.2 Σταθερή πυκνότητα



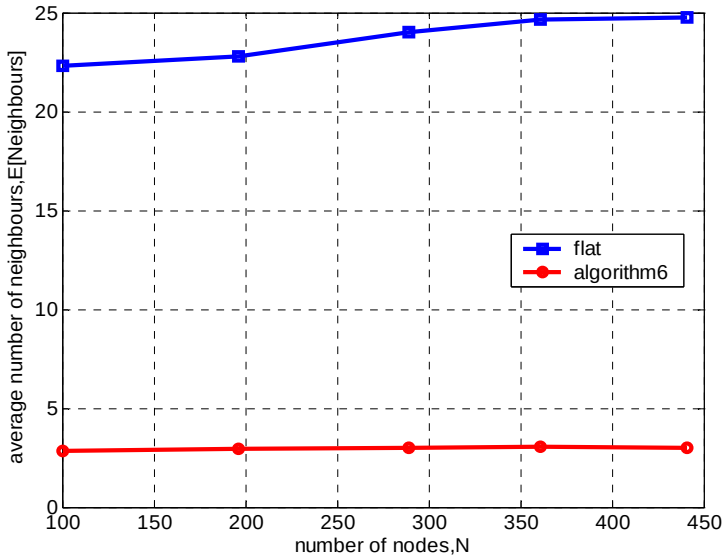

**Διάγραμμα 97. Αλγόριθμος RR-Σταθερή πυκνότητα-Μέση ακτίνα**

Στο Διάγραμμα 97 η μέση ακτίνα αυξάνεται με την αύξηση του μεγέθους του δικτύου.



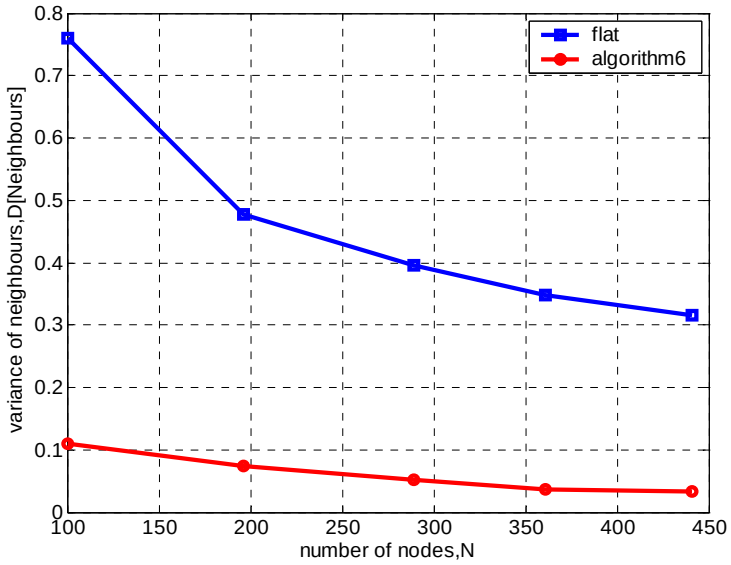

**Διάγραμμα 98. Αλγόριθμος RR-Σταθερή πυκνότητα-Διασπορά ακτίνας**

Στο Διάγραμμα 98 η διασπορά της μέσης ακτίνας μειώνεται με την αύξηση του μεγέθους του δικτύου.




**Διάγραμμα 99. Αλγόριθμος RR-Σταθερή πυκνότητα-Μέσος αριθμός γειτόνων**

Η μεγάλη μείωση του μέσου αριθμού γειτόνων είναι και πάλι εμφανής στο Διάγραμμα 99. Όπως παρατηρούμε έχει ως αποτέλεσμα να αντιστοιχούν γύρω στους 3 γείτονες ανά κόμβο.

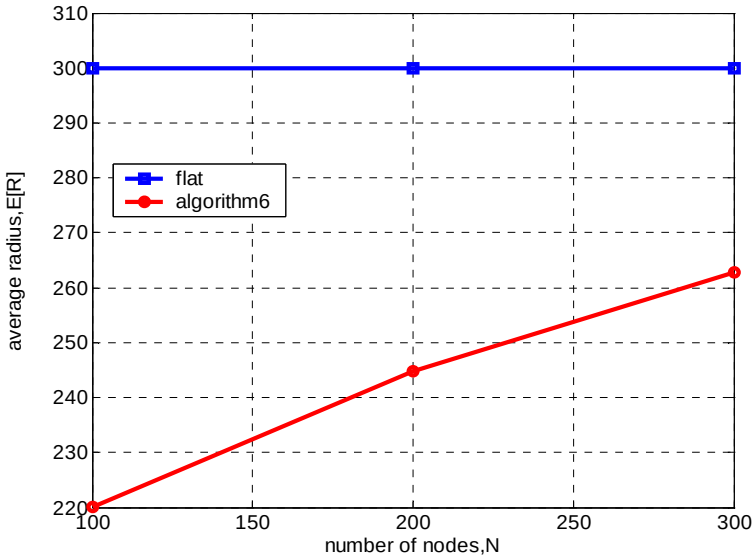



**Διάγραμμα 100. Αλγόριθμος RR-Αυξανόμενη πυκνότητα-Διασπορά αριθμού γειτόνων**

Η διασπορά στο Διάγραμμα 100 είναι αντίστοιχα μειωμένη.

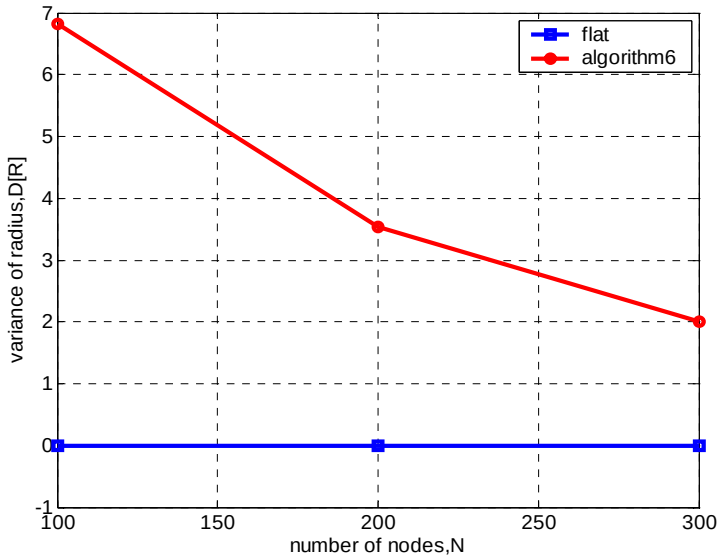
### 5.9.3 Αυξανόμενη πυκνότητα ($q=1$)

Χρησιμοποιήθηκαν οι τιμές πυκνότητας που επιλέχθηκαν στα αντίστοιχα τρεξίματα του αλγορίθμου V για $q=1$.



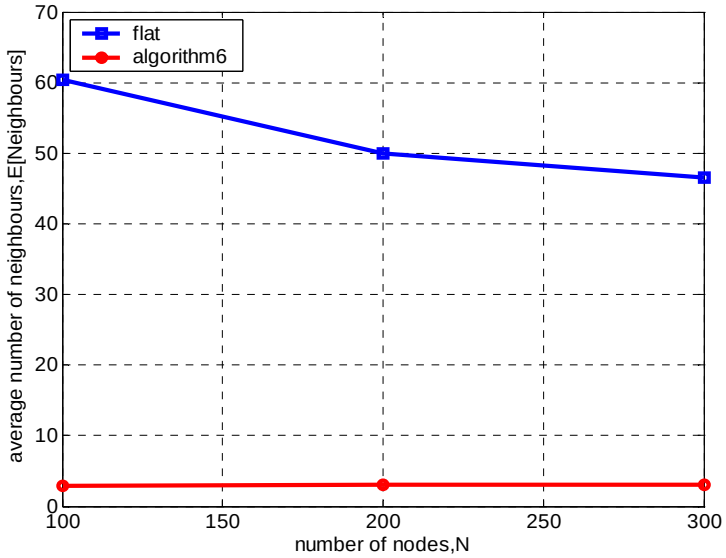

**Διάγραμμα 101. Αλγόριθμος RR(q=1)-Αυξανόμενη πυκνότητα-Μέση ακτίνα**

Η μέση ακτίνα στο τελικό δίκτυο αυξάνεται με την αύξηση της πυκνότητας του δικτύου, όπως φαίνεται από το Διάγραμμα 101.



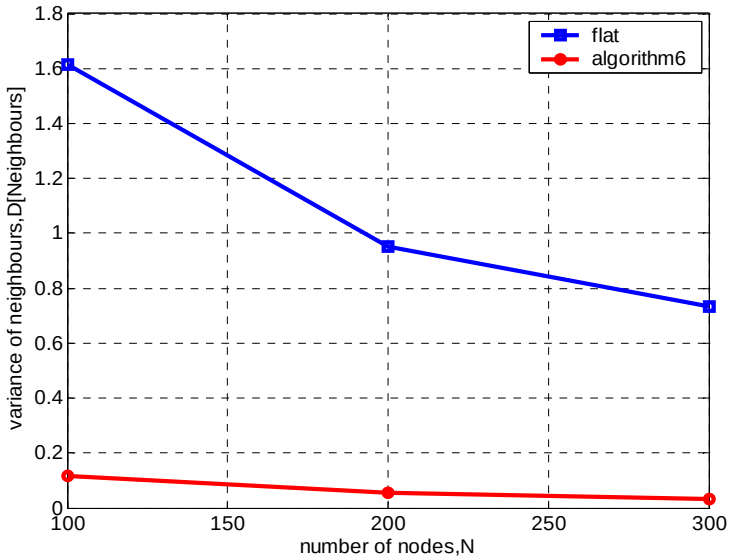

**Διάγραμμα 102. Αλγόριθμος RR(q=1)-Σταθερή πυκνότητα-Διασπορά ακτίνας**

Βλέπουμε στο Διάγραμμα 102 ότι η διασπορά της μέσης ακτίνας μειώνεται.




**Διάγραμμα 103. Αλγόριθμος RR(q=1)-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων**

Η μείωση του μέσου αριθμού γειτόνων παραμένει εξαιρετικά υψηλή στο Διάγραμμα 103. Δεν επηρεάζεται το αποτέλεσμα από την αλλαγή της πυκνότητας.

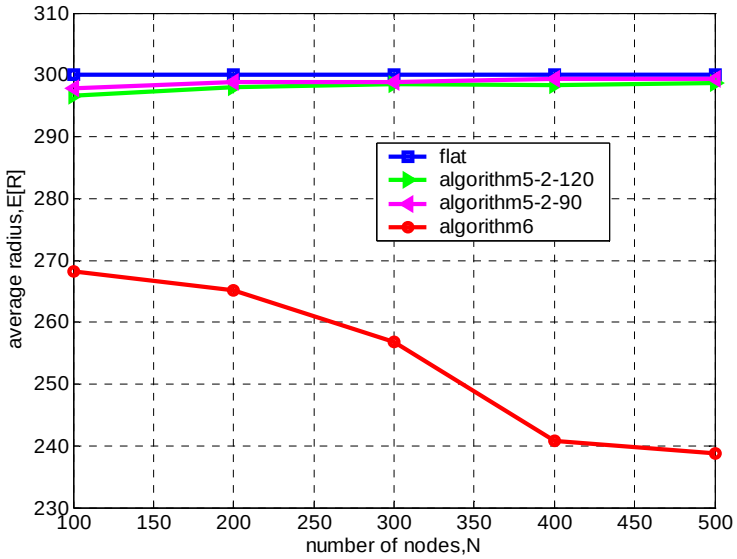



**Διάγραμμα 104. Αλγόριθμος RR(q=1)-Αυξανόμενη πυκνότητα-Διασπορά αριθμού γειτόνων**

Ομοίως στο Διάγραμμα 104 μειώνεται και η διασπορά.

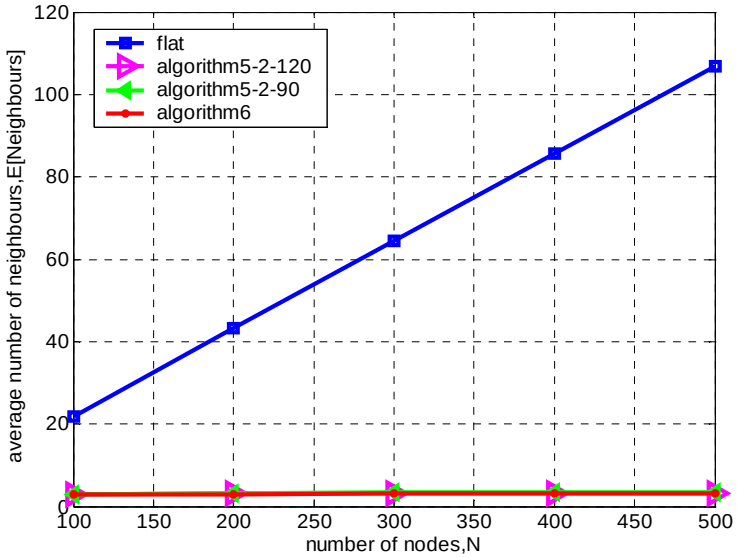
## 5.10 Συγκριτικά RR-V (2-90) - V (2-120), q=2

### 5.10.1 Αυξανόμενη πυκνότητα



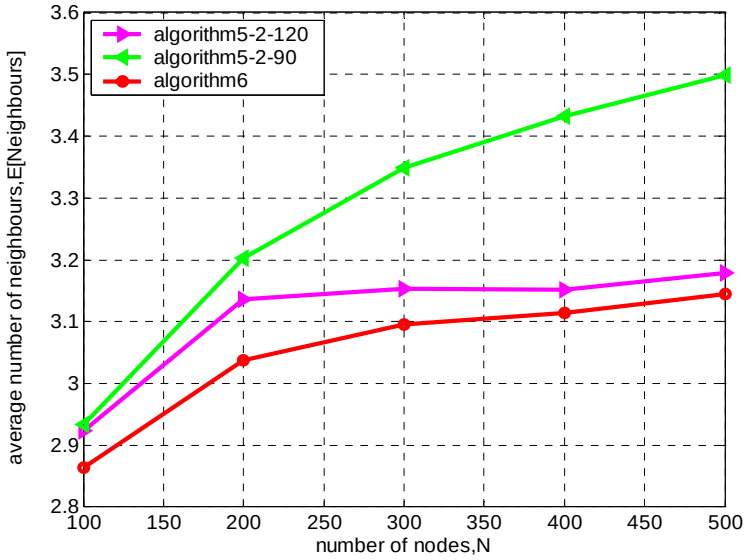

**Διάγραμμα 105. Συγκριτικά V(2-120)-V(2-90)-6,q=2-Αυξανόμενη πυκνότητα-Μέση ακτίνα**

Στο Διάγραμμα 105 βλέπουμε ότι η μέση ακτίνα στον αλγόριθμο RR είναι αρκετά μειωμένη σε σχέση με αυτήν του αλγορίθμου V. Αυτό μπορεί να εξηγηθεί μέσω της ακόλουθης παρατήρησης: ο αριθμός των κόμβων που χαρακτηρίστηκαν ως edge nodes μετά την εφαρμογή του αλγορίθμου RR ήταν πολύ μικρότερος από τον αριθμό των edge nodes του αλγορίθμου V. Αυτό είχε ως άμεση συνέπεια την μείωση της μέσης ακτίνας του RR εφόσον κάθε edge node θεωρείται ότι διατηρεί τη μέγιστη ακτίνα. Ανάμεσα στις δύο εκδοχές του αλγόριθμου V μικρότερη μέση ακτίνα δίνει η εκδοχή με $a = 120^{\circ}$. Αυτό είναι λογικό διότι όσο μεγαλύτερη η γωνία κώνου, τόσο λιγότεροι γείτονες και άρα μικρότερη ακτίνα για κάθε κόμβο.




**Διάγραμμα 106. Συγκριτικά V(2-120)-V(2-90)-6,q=2-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων**

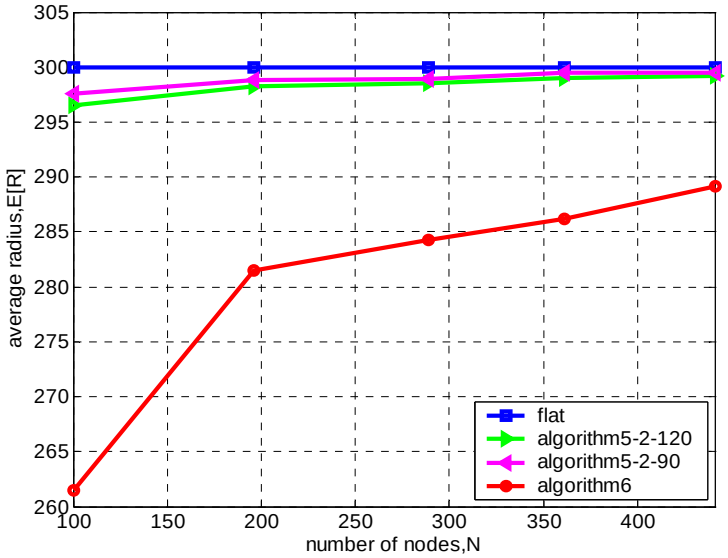
Δεν διακρίνεται ξεκάθαρα από το Διάγραμμα 106 η διαφορά μεταξύ των αλγορίθμων. Αυτό μπορεί να επιτευχθεί αν παραλείψουμε τη γραφική του αρχικού δικτύου για να μειωθεί το εύρος τιμών που είναι απαραίτητο για το διάγραμμα και να μπορέσουν να συγκριθούν οι αλγόριθμοι.




**Διάγραμμα 107. Συγκριτικά V(2-120)-V(2-90)-6,q=2-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων-Μόνο τελικά**

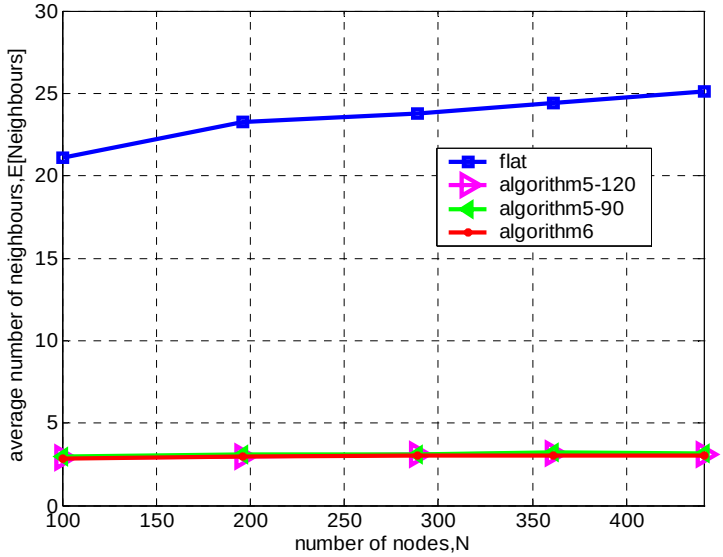
Βλέπουμε στο Διάγραμμα 107 ότι ο αλγόριθμος RR δίνει τον χαμηλότερο μέσο αριθμό γειτόνων. Ακολουθεί ο αλγόριθμος V(2-120) και επιβεβαιώνεται με αυτόν τον τρόπο ότι η μεγαλύτερη γωνία κώνου έχει ως αποτέλεσμα μικρότερο μέσο αριθμό γειτόνων. Τα αποτελέσματα αυτά έρχονται σε αντίθεση με τα αποτελέσματα που προέκυψαν στο [2], όπου είχε αποδειχτεί ότι ο V είναι καλύτερος από τον RR. Οι μετρήσεις στο [2] όμως έγιναν για συγκεκριμένο μέγεθος δικτύου και αριθμού κόμβων. Συγκεκριμένα επιλέχθηκε διάσταση της γεωγραφικής περιοχής το *1500m* και αριθμός κόμβων το *100*. Αυτός ο συνδυασμός δεν καλύπτεται από τις παραπάνω μετρήσεις. Γι' αυτό το λόγο πραγματοποιήθηκαν στη συνέχεια ειδικά μετρήσεις για τον έλεγχο της συμπεριφοράς των αλγορίθμων σε δίκτυο με τα παραπάνω χαρακτηριστικά.

## 5.10.2 Σταθερή πυκνότητα



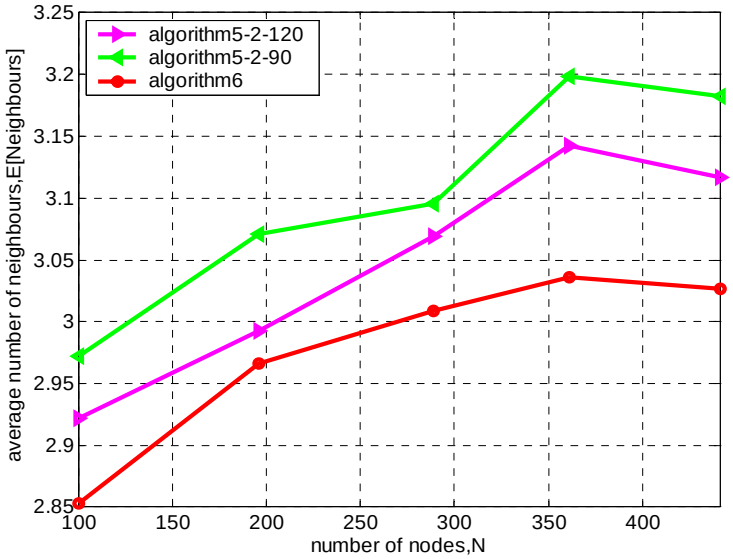

**Διάγραμμα 108. Συγκριτικά V(2-120)-V(2-90)-6,q=2-Σταθερή πυκνότητα-Μέση ακτίνα**

Οι παρατηρήσεις που είχαν γίνει στην αυξανόμενη πυκνότητα ισχύουν και στην σταθερή για το Διάγραμμα 108. Η μέση ακτίνα του αλγορίθμου RR είναι με διαφορά μικρότερη από τη μέση ακτίνα του V.




**Διάγραμμα 1Διάγραμμα 109. Συγκριτικά V(2-120)-V(2-90)-6,q=2-Σταθερή πυκνότητα-Μέσος αριθμός γειτόνων**

Πάλι δεν διακρίνουμε τις διαφορές στο Διάγραμμα 1 και επομένως ακολουθεί γραφική χωρίς το αρχικό.




**Διάγραμμα 110. Συγκριτικά V(2-120)-V(2-90)-6,q=2-Σταθερή πυκνότητα-Μέσος αριθμός γειτόνων-Μόνο τελικά**

Ο αλγόριθμος RR παραμένει καλύτερος και στις μετρήσεις σταθερής πυκνότητας αν και οι τρεις κινούνται σε πολύ χαμηλά επίπεδα τιμών αριθμού γειτόνων, όπως φαίνεται από το Διάγραμμα 110.

## 5.11 Συγκριτικά RR-V (2-90) – V (2-120), q=2, L=1500, N=100 (όπου L η διάσταση της γεωγραφικής περιοχής και N ο αριθμός κόμβων)

Παρατίθενται στον παρακάτω πίνακα τα αποτελέσματα που προέκυψαν από το συγκεκριμένο τρέξιμο, που πραγματοποιήθηκε για να ελεγχθούν και αν δυνατόν να επιβεβαιωθούν τα αποτελέσματα του [2]. Πραγματοποιείται σύγκριση μόνο του μέσου αριθμού γειτόνων.

| | **Αλγόριθμος V-φάση 2** | | **Αλγόριθμος RR** | **Αρχικό δίκτυο** |
|---|---|---|---|---|
| **Μέσος αριθμός γειτόνων** | $a = 2\pi/3$ | $a = \pi/2$ | | |
| | 2.63 | 2.66 | 2.77 | 10.43 |

**Πίνακας 1. Σύγκριση μέσου αριθμού γειτόνων αλγορίθμων V(2-90),V(2-120) και RR**

Παρατηρούμε ότι στη συγκεκριμένη μέτρηση, με τις δεδομένες διαστάσεις και αριθμό κόμβων ο αλγόριθμος V, και για τις δύο τιμές του $a$, προέκυψε οριακά καλύτερος από τον RR όσο αφορά το μέσο αριθμό γειτόνων. Τα αποτελέσματα επιβεβαιώνουν επομένως την εργασία που έγινε στο [2]. Παρατηρούμε επίσης ότι και οι δύο αλγόριθμοι περιορίζουν σημαντικά το μέσο αριθμό γειτόνων του αρχικού δικτύου.

Δεν θεωρήθηκε απαραίτητο να πραγματοποιηθεί επιπλέον μέτρηση, υπό τις δεδομένες τιμές παραμέτρων, για την φάση 1 του αλγορίθμου V. Όπως έχει δειχθεί στο αντίστοιχο υποκεφάλαιο η φάση 1 μειώνει τον μέσο αριθμό γειτόνων του αρχικού δικτύου αλλά όχι στο σημαντικό βαθμό που το επιτυγχάνει η φάση 2. Αυτό

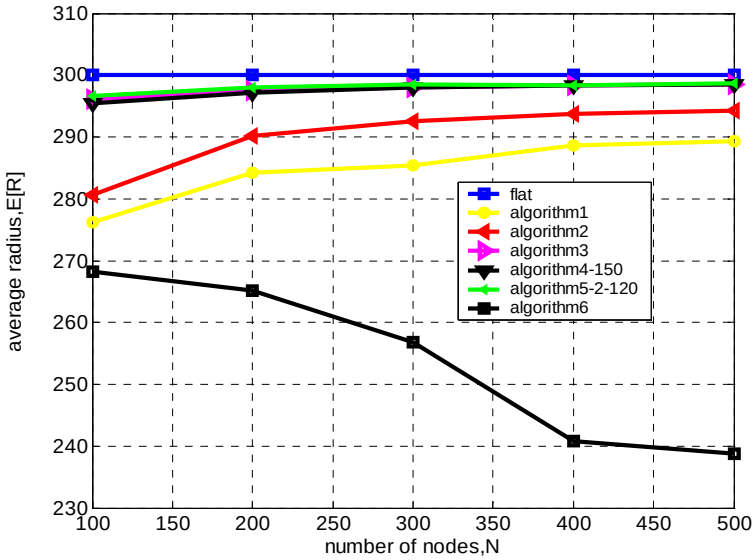
το αποτέλεσμα είχε προκύψει και με τις προηγούμενες τιμές μετρήσεων, συμφωνεί με τα αποτελέσματα του [2] και δεν χρειάστηκε συμπληρωματική μέτρηση για την επιβεβαίωση του

Σε αυτό το σημείο είναι απαραίτητη να αναφερθεί μία σημαντική διαφοροποίηση των συγκεκριμένων μετρήσεων με αυτές του [2]. Οι μετρήσεις του [2] έγιναν για $q=1$, ενώ οι συγκεκριμένες για $q=2$. Η επιλογή της τιμής 2 για το $q$ ήταν επιβεβλημένη διότι όπως ήδη αναφέρθηκε για τη δεδομένη πυκνότητα με $q=1$ δεν μπορούσε να εφαρμοστεί η φάση 2. Το γεγονός αυτό συνδέεται με την παρατήρηση που είχε γίνει στην περιγραφή της υλοποίησης του αλγορίθμου V, ότι δηλαδή επιλέχθηκε να υπάρχει ένας γείτονας για κάθε κώνο με γωνία α αλλά όχι παραπάνω, όπως θα επέτρεπε ο αλγόριθμος. Αυτό έχει ως αποτέλεσμα στο δίκτυο που δημιουργείται μετά τη φάση 1 οι κόμβοι να έχουν τον ελάχιστο αριθμό γειτόνων που απαιτούνται για να ικανοποιηθεί η συνθήκη του αλγορίθμου. Άρα η φάση 2 σε αυτήν την περίπτωση αφαιρεί λιγότερους γείτονες από όσους θα μπορούσε. Άρα σε κάποιες τιμές πυκνοτήτων έχει δημιουργηθεί τόσο αυστηρά το δίκτυο μετά τη φάση 1 που η φάση 2 δεν έχει να αφαιρέσει κάποιο γείτονα και ως εκ τούτου η ανισότητα δεν μπορεί να ικανοποιηθεί. Αν η κατασκευή του δικτύου ήταν διαφορετική και οι γείτονες μετά τη φάση 1 ήταν περισσότεροι από τους ελάχιστους τότε στις ίδιες τιμές πυκνοτήτων η φάση 2 θα μπορούσε να λειτουργήσει.

## 5.12 Γενικά συγκριτικά όλων των αλγορίθμων

Στα συγκριτικά αυτά χρησιμοποιήθηκαν ο αλγόριθμος I, ο αλγόριθμος II, ο αλγόριθμος III, ο αλγόριθμος IV-150 (που αποδείχτηκε καλύτερος από τον IV-250), ο αλγόριθμος V-120 (που αποδείχτηκε καλύτερος από τον V-90) και ο αλγόριθμος RR.

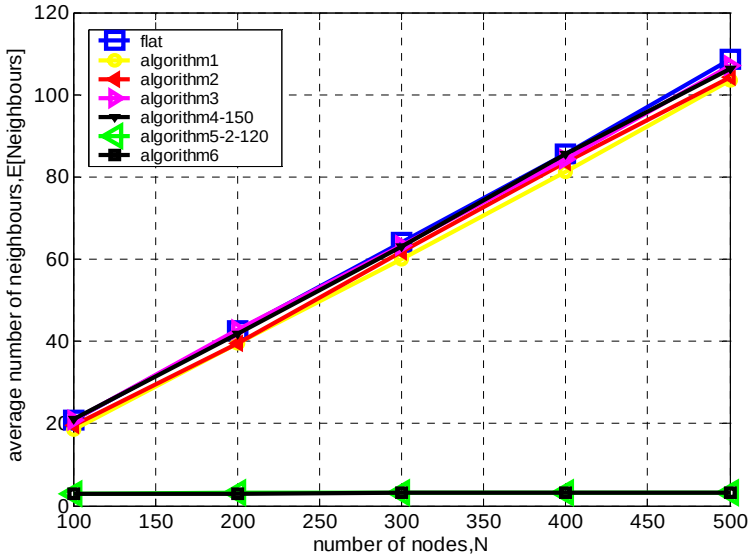
### 5.12.1 Αυξανόμενη πυκνότητα




**Διάγραμμα 111. Συγκριτικά όλων των αλγορίθμων-Αυξανόμενη πυκνότητα-Μέση ακτίνα**

Στο Διάγραμμα 111 βλέπουμε ότι τη μικρότερη μέση ακτίνα την παρουσιάζει ο αλγόριθμος RR. Αυτό είχε διαπιστωθεί και νωρίτερα όταν επισημάνθηκε ότι ο μειωμένος αριθμός edge nodes κατεβάζει αρκετά το μέσο όρο. Ακολουθεί ο αλγόριθμος I. Αυτό είναι φυσιολογικό διότι ο αλγόριθμος αυτός χρησιμοποιεί την

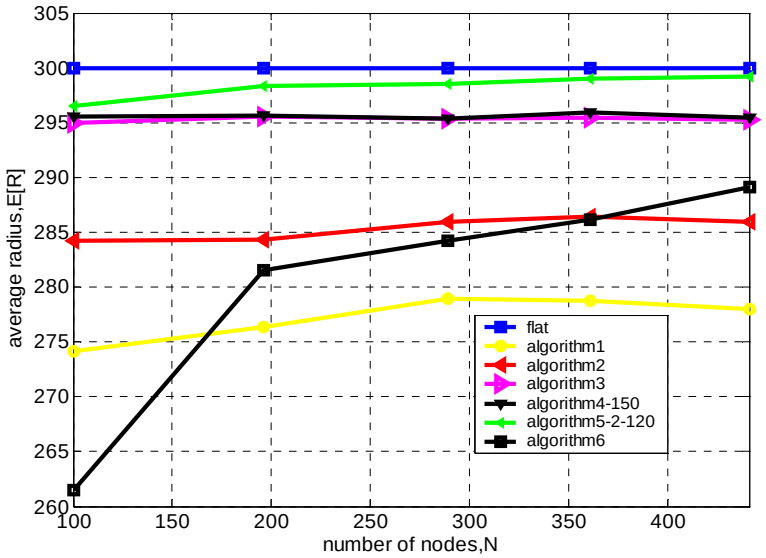
μεγαλύτερη γωνία κώνου από όλους τους αλγορίθμους του είδους του που εξετάζονται σε αυτήν την εργασία. Μεγάλη γωνία κώνου έχει ως αποτέλεσμα μικρότερο αριθμό γειτόνων άρα και μικρότερη ακτίνα. Με την ίδια λογική αμέσως καλύτερος είναι ο αλγόριθμος ΙΙ, που βασίζεται στο Ι και έχει ένα δεύτερο βήμα που του διευρύνει λίγο όπως περιγράφηκε προηγουμένως την μέση ακτίνα του. Ο ΙΙΙ και IV-150 βρίσκονται στα ίδια επίπεδα και πιο ψηλά από τους Ι και ΙΙ, εφόσον είναι σχεδόν όμοιοι, το κατώφλι του IV είναι σχετικά χαμηλό και η γωνία κώνου που χρησιμοποιήθηκε μικρότερη από αυτήν που χρησιμοποίησαν ο Ι και ο ΙΙ. Τέλος ο V-120 εμφανίζεται κι αυτός να έχει τα ίδια επίπεδα μέσης ακτίνας με τους προηγούμενους δύο ή οριακά μεγαλύτερη.




**Διάγραμμα 112. Συγκριτικά όλων των αλγορίθμων-Αυξανόμενη πυκνότητα-Μέσος αριθμός γειτόνων**

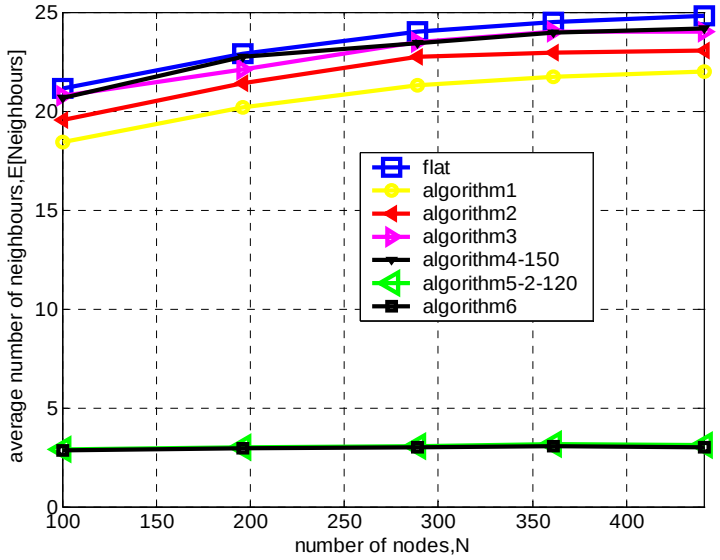
Στο Διάγραμμα 112 της αυξανόμενης πυκνότητας δεν μπορεί να γίνει εύκολα σύγκριση απλά διακρίνουμε την μεγάλη διαφορά σε μέσο όρο γειτόνων που δίνουν ο αλγόριθμος V-120 και ο αλγόριθμος RR από τους υπόλοιπους

## 5.12.2 Σταθερή πυκνότητα




**Διάγραμμα 113. Συγκριτικά όλων των αλγορίθμων-Σταθερή πυκνότητα-Μέση ακτίνα**

Ως προς τη σύγκριση των αλγορίθμων μεταξύ τους ισχύουν οι ίδιες παρατηρήσεις που έγιναν και πριν με μόνη διαφορά ότι στο Διάγραμμα 113 βλέπουμε λίγο πιο σταθεροποιημένες τις τιμές σε κάθε αλγόριθμο εκτός από τον RR.




**Διάγραμμα 114. Συγκριτικά όλων των αλγορίθμων-Σταθερή πυκνότητα-Μέσος αριθμός γειτόνων**

Στο Διάγραμμα 114 μπορούμε λίγο πιο ξεκάθαρα να συγκρίνουμε τους αλγορίθμους. Αντίστοιχα με τις παρατηρήσεις που έγιναν πριν για τη μέση ακτίνα της αυξανόμενης πυκνότητας, φαίνονται τα αποτελέσματα και στο μέσο αριθμό γειτόνων και ανάλογα ερμηνεύονται..

# Κεφάλαιο 6 - Συμπεράσματα

Σε αυτήν την εργασία έγινε μελέτη, υλοποίηση και σύγκριση έξι αλγορίθμων ελέγχου τοπολογίας για ασύρματα δίκτυα ad hoc και δίκτυα αισθητήρων. Οι αλγόριθμοι που μελετήθηκαν επιδιώκουν μέσω του ελέγχου τοπολογίας να επιτύχουν μείωση της συνολικής κατανάλωσης ενέργειας του ασύρματου δικτύου.

Ο έλεγχος τοπολογίας συνίσταται στο έλεγχο του επιπέδου ισχύος εκπομπής κάθε κόμβου και επομένως τον καθορισμό της ακτίνας μετάδοσης, με σκοπό κάθε κόμβος να περιορίσει τους γείτονές του σε έναν όσο το δυνατόν μικρότερο αριθμό με ταυτόχρονη διατήρηση της συνεκτικότητας του υποκείμενου γράφου δικτύου. Η μείωση του επιπέδου ισχύος εκπομπής κάθε κόμβου έχει ως αποτέλεσμα, την αύξηση του χρόνου ζωής του εκάστοτε τερματικού αλλά και του συνολικού δικτύου.

Μέσω των συγκεκριμένων αλγορίθμων ουσιαστικά συγκρίθηκαν και μελετήθηκαν δύο διαφορετικές προσεγγίσεις που έχουν προταθεί για αποτελεσματικό έλεγχο τοπολογίας, οι γωνίες κώνων και οι περιοχές σκυτάλης. Οι πέντε πρώτοι αλγόριθμοι βασίζονται στις γωνίες κώνων και ουσιαστικά επιδιώκουν μείωση της ακτίνας μετάδοσης, άρα και του αριθμού γειτόνων, μέσω του καθορισμού διαφορετικών κάθε φορά κριτηρίων για τις εγγενείς γωνίες μεταξύ των ζεύξεων επικοινωνίας. Ο έκτος αλγόριθμος βασίζεται στις περιοχές σκυτάλης και μέσω του καθορισμού αυτών επιδιώκει τον προσδιορισμό αυτού του συνόλου γειτόνων κάθε κόμβου, μέσω των οποίων θα είναι πιο οικονομική η προώθηση πληροφορίας.

Οι αλγόριθμοι υλοποιήθηκαν σε C++ με τη βοήθεια του πακέτου ανάπτυξης εφαρμογών Microsoft Visual C++ 6.0. Οι πέντε πρώτοι αλγόριθμοι είχαν περιγραφεί θεωρητικά για άπειρη γεωγραφική περιοχή και το σημαντικότερο πρόβλημα που προέκυψε είχε σχέση με την προσπάθειά υλοποίησης τους για πεπερασμένη περιοχή. Διαπιστώθηκε πολύ γρήγορα ότι υπήρχαν κόμβοι, οι οποίοι λόγω αυτού του πρακτικού περιορισμού δεν ήταν δυνατόν να ικανοποιήσουν τους αλγορίθμους. Οι κόμβοι αυτοί ονομάστηκαν edge nodes και σε κάθε αλγόριθμο αντιμετωπίστηκαν με συγκεκριμένο τρόπο. Στον τελευταίο αλγόριθμο, αν και αυτός αναφερόταν θεωρητικά σε συγκεκριμένο, πεπερασμένο χώρο, προέκυψαν επίσης edge nodes λόγω της γεωμετρικής φύσεως του κριτηρίου τερματισμού του αλγορίθμου.

Έπειτα από την υλοποίηση των αλγορίθμων πραγματοποιήθηκε ικανός αριθμός μετρήσεων για να προκύψουν αξιόπιστα αποτελέσματα. Τα δικτυακά μεγέθη που μετρήθηκαν σε κάθε αλγόριθμο ήταν η μέση ακτίνα κάλυψης, ο μέσος αριθμός γειτόνων και οι διασπορές τους. Παρατηρήθηκε επίσης ο αριθμός των edge nodes που προέκυπτε κάθε φορά και ελέγχθηκε η διατήρηση της συνεκτικότητας του δικτύου μετά από κάθε εφαρμογή αλγορίθμου. Η σύγκριση των αποτελεσμάτων έγινε με τη βοήθεια γραφικών παραστάσεων, οι οποίες για την επίτευξη μεγαλύτερης ευκρίνειας υλοποιήθηκαν στο MATLAB 6.5.

Συμπερασματικά παρατηρήθηκε ότι όσο μεγαλύτερο ήταν το κατώφλι για τη γωνία κώνου τόσο μικρότερος προέκυπτε ο μέσος αριθμός γειτόνων και η μέση ακτίνα. Επομένως ανάμεσα στους τέσσερις πρώτους, καλύτερος ως προς τα παραπάνω μεγέθη αναδείχτηκε ο πρώτος. Ο πέμπτος αλγόριθμος, παρά το γεγονός ότι βασιζόταν και αυτός στην έννοια της γωνίας κώνου, περιείχε και μία δεύτερη φάση εκτέλεσης, η οποία αποδείχτηκε ότι κατάφερε να μειώσει σημαντικά το μέσο αριθμό γειτόνων. Αυτή η διαπίστωση τον κατέταξε στη πρώτη θέση τελικά ανάμεσα στους αλγορίθμους της πρώτης προσέγγισης. Ο τελευταίος αλγόριθμος, ο οποίος ήταν και ο

μόνος εκπρόσωπος της δεύτερης προσέγγισης, είχε αντίστοιχα καλά αποτελέσματα με τον πέμπτο. Απαραίτητη επομένως κρίθηκε και μία επιμέρους σύγκριση των δύο τελευταίων. Αυτή η σύγκριση είχε ήδη πραγματοποιηθεί στο [2] με συγκεκριμένες τιμές παραμέτρων δικτύου γι' αυτό και επιχειρήθηκε και επαλήθευση των αποτελεσμάτων με τις συγκεκριμένες παραμέτρους. Τελικά διαπιστώθηκε ότι για τις δεδομένες τιμές ο πέμπτος αλγόριθμος έδινε οριακά καλύτερα αποτελέσματα από τον έκτο, κάτι που επαλήθευσε και τις μετρήσεις του [2]. Έγινε όμως και μία προσπάθεια σύγκρισης της συμπεριφοράς των αλγορίθμων κατά τη μεταβολή της πυκνότητας και του μεγέθους δικτύου και προέκυψε ότι για το καινούριο σύνολο τιμών που εξετάσθηκαν ο έκτος ήταν οριακά καλύτερος από τον πέμπτο. Τελικά οι δύο τελευταίοι αλγόριθμοι αποδείχθηκε ότι δίνουν συγκρίσιμα αποτελέσματα, τα οποία απέχουν κατά πολύ από τους υπόλοιπους τέσσερις.

Τα συμπεράσματα της συγκεκριμένης εργασίας μπορούν να χρησιμοποιηθούν μελλοντικά σε περαιτέρω έρευνα για την επίτευξη αποδοτικότερου ελέγχου τοπολογίας, μιας και το συγκεκριμένο πεδίο είναι από τα κρισιμότερα στο σχεδιασμό ασύρματων ad hoc δικτύων και δικτύων αισθητήρων. Η περαιτέρω μελέτη θα μπορούσε ενδεχομένως να περιλαμβάνει έλεγχο και σύγκριση της συμπεριφοράς των αλγορίθμων σε σχέση με την χωρητικότητα δικτύου που επιτυγχάνεται κάθε φορά αλλά και σε σχέση με τις παρεμβολές που προκύπτουν. Δεδομένου του γεγονότος ότι η συγκεκριμένη εργασία περιορίστηκε στην εφαρμογή των αλγορίθμων σε στατικό δίκτυο, εξαιρετικό ενδιαφέρον θα παρουσίαζε επίσης η επέκταση της έρευνας στον έλεγχο της ανθεκτικότητας των αλγορίθμων στην περίπτωση κίνησης των κόμβων ή σε πιθανά σφάλματα.

# Παράρτημα

## Αλγόριθμος Ι

```
#include <iostream.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <fstream.h>
#include <iomanip.h>

#define N 100
#define RMAX 300
#define L 1000
#define STEP 0.5

/*klasi tools pou periexei tis genikes sinartiseis tou programmatos*/
class Tools
{
public:

  float Distance(float x1,float y1,float x2,float y2);
  void CreateMatrixOfNeighbourhood(int matrix[N][N],float
node[3][N],int n);

  void qsort_auxil(float a[],int lower,int upper);
  void qsort(float a[],int n);

  float FindConeAngles(int matrix[N][N],int i,float node[3][N]);
  int FindNumberOfNeighbours(int matrix[N][N],int i);
  int FindMaxOfMatrix(int matrix[N]);
  int FindMinOfMatrix(int matrix[N]);
  int FindNumberOfComponents(int matrix[N][N]);
  float FindAverageRadius(float node[3][N],int n);
  float FindAverageNumberOfNeighbours(int matrix[N][N],int n);
  float FindVarianceOfAverageRadius(float node[3][N],int n);
  float FindVarianceOfAverageNeighbours(int matrix[N][N],int n);

private:
   int matrix[N][N];
   float node[3][N];
};

/*sinartisi Distance pou ipologizei tin apostasi metaksi
dio simeiwn A(x1,y1) kai B(x2,y2)*/
float Tools::Distance(float x1,float y1,float x2,float y2)
{
 return sqrt(((x2-x1)*(x2-x1))+((y2-y1)*(y2-y1)));
}

/*sinartisi CreateMatrixOfNeighbourhood pou ipologizei
ton pinaka geitonias pairnontas ws parametrous ton arithmo
twn kombwn kai ton pinaka pou periexei tis sintetagmenes,
to id kai tin aktina tous*/
void Tools::CreateMatrixOfNeighbourhood(int matrix[N][N],float
node[3][N],int n)
{
```

```
  float x1,y1,x2,y2;
  float d,r;
  int i,j;

  for(i=0;i<N;i++)
   for(j=0;j<N;j++)
       matrix[i][j]=0;


  for (i=0;i<n;i++)
  {
   x1=node[0][i];
   y1=node[1][i];
   r=node[2][i];

   for (j=0;j<n;j++)           //diatrexontai oloi oi komboi kai
simfwna me tin aktina poy
   {                           //exei o ekastote kombos elegxetai gia
olous tous ipoloipous
         if(j!=i)
      {
       x2=node[0][j];
       y2=node[1][j];
       d=Distance(x1,y1,x2,y2);

       if ( d <= r )           //an i apostasi tous apo ton kombo
einai mikroteri tis aktinas
            matrix[i][j]=1;    //tou diladi an einai mesa stin
embeleia tou
         }                       //an nai topotheteitai to 1 sto
antistoixo stoixeio tou pinaka geitonias
        else matrix[i][j]=0;     //alliws to 0
     }
  }
}

/*sinartisi qsort_auxil pou xrisimopoiitai apo tin qsort
gia tin diataksi twn stoixeiwn enos pinaka se ayksousa seira*/
void Tools::qsort_auxil(float a[],int lower,int upper)
{
  if (lower<upper)
    {
     float x=a[(lower+upper)/2];
     int i,j;

     for (i=lower,j=upper;i<=j;i++,j--)
       {
          while (a[i]<x) i++;
          while (a[j]>x) j--;

          if (i<=j)
            {
           float temp=a[i];

           a[i]=a[j];
           a[j]=temp;
            }
       }
     qsort_auxil(a,lower,j);
     qsort_auxil(a,i,upper);
    }
```

```
}

void Tools::qsort(float a[],int n)
{
  qsort_auxil(a,0,n-1);
}


/*sinartisi FindConeAngles pou briskei tis cone angles enos kombou
kai epistrefei ti megisti*/
float Tools::FindConeAngles(int matrix[N][N],int i,float node[3][N])
 {
 int j,k,l;
 float xnew,ynew,biggest;
 float cosf,f,g;
 float matrix_angles[N];


 for(l=0;l<N;l++)
 {
   matrix_angles[l]=0.0;
 }


 k=0;

 for (j=0;j<N;j++)
   {

    if (matrix[i][j]==1)             //ean to stoixeio tou pinaka
geitonias einai 1
     {
       xnew=node[0][j]-node[0][i];   //metasximatismos twn
sintetagmenwn tou kathe geitonikou
       ynew=node[1][j]-node[1][i];   //kombou se sxesi me ton
kentriko pou orizetai arxi twn axonwn

       if((xnew==0.0)&&(ynew>0.0))   //ean o geitonas einai stin x=0
kai y>0
         {
            f=90.0;                     //tote i gwnia prokiptei 90
            matrix_angles[k]=f;
           }
       else
         {
            if ((xnew==0.0)&&(ynew<0.0)) //ean o geitonas einai stin
x=0 kai y<0
              {
             f=270.0;                     //tote i gwnia prokiptei 270
             matrix_angles[k]=f;
              }
            else
              {
             if ((ynew==0.0)&&(xnew>0.0)) //ean o geitonas einai stin
y=0 kai x>0
                 {
               f=0.0;                      //tote i gwnia prokiptei 0
               matrix_angles[k]=f;
               }
             else
               {
```

```
              if ((ynew==0.0)&&(xnew<0.0)) //ean o geitonas einai
stin y=0 kai x<0
               {
                  f=180.0;                   //tote i gwnia einai 180
                matrix_angles[k]=f;
                  }
              else                         //ean den anikei se mia
apo tis 4 eidikes periptwseis
              {
                  if
(((xnew>0.0)&&(ynew>0.0))||((xnew<0.0)&&(ynew>0.0))) //ean o kombos
brisketai sto 1 i sto 2 tetartimorio
                  {

cosf=(1*xnew+1*ynew)/((sqrt(2))*Distance(0.0,0.0,xnew,ynew));
                   f=acos(cosf)*180/3.141592654;

                     matrix_angles[k]=f;     //ipologizetai i gwnia
toy kathe geitona me to thetiko tmima tou orizontiou axona
                }
              else
                 {
                     if
(((xnew<0.0)&&(ynew<0.0))||((xnew>0.0)&&(ynew<0.0))) //ean einai sto
3 i sto 4
                   {

cosf=(1*xnew+1*ynew)/((sqrt(2))*Distance(0.0,0.0,xnew,ynew));
                          f=acos(cosf)*180/3.141592654;

                          matrix_angles[k]=360.0-f; //afairesi me to
360 dioti o tipos dinei panta ti mikri gwnia
                     }
                   }
                }
            }
         }
        }
   k=k+1;
       }
     }

 if (k==1)
   return 360.0;


 qsort(matrix_angles,k);   //diataxi tou pinaka me tis gwnies se
ayksoysa seira

 biggest=-12000.0;

 for(i=0;i<k-1;i++)
 {
   g=matrix_angles[i+1]-matrix_angles[i];  //afairesi kathe stoixeiou
apo to epomeno gia na prokipsoun telika oi cone angles

   if (g>biggest)
       biggest=g;
 }
```

```
 g=360.0-matrix_angles[k-1]+matrix_angles[0];//ipologismos tis
teleytaias gwnias me eidiko typo
                                            //dioti to apotelesma
den tha itan swsto me afairesi
 if (g>biggest)
   return g;

 return biggest;
}

/*sinartisi FindNumberOfNeighbours pou briskei ton arithmo
 twn geitonwn enos kombou*/
int Tools::FindNumberOfNeighbours(int matrix[N][N],int i)
{
  int j,k;

  k=0;

  for(j=0;j<N;j++)
  {
       if (matrix[i][j]==1)
             k=k+1;
  }

  return k;
}

/*sinartisi FindMaxOfMatrix pou briskei to megisto stoixeio enos
pinaka*/
int Tools::FindMaxOfMatrix(int matrix[N])
{
 int k,i;

 k=matrix[0];

 for(i=0;i<N;i++)
 {
       if (matrix[i]>k)
             k=matrix[i];
 }

 return k;
}

/*sinartisi FindMinOfMatrix pou briskei to elaxisto stoixeio enos
pinaka*/
int Tools::FindMinOfMatrix(int matrix[N])
{
 int k,i;

 k=matrix[0];

 for(i=0;i<N;i++)
 {
       if (matrix[i]<k)
             k=matrix[i];
 }

 return k;
}
```

```
/*sinartisi FindNumberOfComponents pou briskei ton arithmo
 twn components enos diktioy*/
int Tools::FindNumberOfComponents(int matrix[N][N])
{
 int components;
 int k,m,n,b;
 int nodeVisited[N];
 int nodeStack[N];

 components=0;

 for(k=0;k<N;k++)
 {
   nodeVisited[k]=0;
   nodeStack[k]=0;
 }

 m=0;
 nodeVisited[m]=1;

 while ((FindMinOfMatrix(nodeVisited))==0) //oso iparxoun komboi poy
den toys exoume episkeftei
 {
   nodeVisited[m]=1;

   for(n=m+1;n<N;n++)
   {
        if ((matrix[m][n])==1)              //an o kombos n einai
geitonas toy m
        {
             nodeVisited[n]=1;                //markaretai o n ws
kombos poy ton episkeftikame
             nodeStack[n]=1;                  //markarw sti stoiba
ton n kanontas to stoixeio 1
        }
   }

   while ((FindMaxOfMatrix(nodeStack))==1)
   {
       k=0;

       while ((nodeStack[k])==0)
       {
       k=k+1;
       }

       nodeStack[k]=0;                     //briskei to prwto mi
mideniko stoixeio kai to midenizei

       for(n=0;n<N;n++)
       {
         if ((matrix[k][n])==1)            //an yparxei geitonas
aytoy toy stoixeioy
         {
         if (nodeVisited[n]==0)
         {
               nodeVisited[n]=1;            //markaretai o n san
kombos poy episkeftikame
               nodeStack[n]=1;
             }
       }
```

```
        }
    }

    components=components+1;

    b=0;
    m=0;

    while (b<N-1)
    {
      b=b+1;
        if (nodeVisited[b]==0)
        {
          m=b;    //stin oysia briskei ton prwto kombo poy den
enwnetai me toys proigoymenoys
          b=N-1;  //kai an ton brei termatizei ayto to while kai
trexei apo tin arxi to ekswteriko while
        }
    }
 } //to ekswteriko while termatizetai otan exei ginei episkepsi se
oloys toys komboys

return components;
}

/*sinartisi pou ipologizei ti mesi aktina*/
float Tools::FindAverageRadius(float node[3][N],int n)
{
  int i;
  float sum=0;

  for(i=0;i<n;i++)
      sum=sum+node[2][i];

  sum=sum/n;
  return sum;
}

/*sinartisi pou ipologizei to meso arithmo geitonwn*/
float Tools::FindAverageNumberOfNeighbours(int matrix[N][N],int n)
{
  int i;
  float sum=0.0;

  for(i=0;i<n;i++)
    sum=sum+FindNumberOfNeighbours(matrix,i);
  sum=sum/n;
  return sum;
}

/*sinartisi FindVarianceOfAverageRadius pou ipologizei ti variance
tis aktinas*/
float Tools::FindVarianceOfAverageRadius(float node[3][N],int n)
{
  int i;
  float rm;
  float sum=0.0;
  float d;

  rm=FindAverageRadius(node,n);
```

```
  for(i=0;i<n;i++)
  {
        sum=sum+((node[2][i]-rm)*(node[2][i]-rm));
  }

  d=sqrt(sum/(n*(n-1)));

  return d;
}

/*sinartisi FindVarianceOfAverageNeighbours pou ipologizei ti
variance twn geitonwn*/
float Tools::FindVarianceOfAverageNeighbours(int matrix[N][N],int n)
{
  int i,xm;
  float sum=0.0;
  float d;

  xm=FindAverageNumberOfNeighbours(matrix,n);

  for(i=0;i<n;i++)
  {
        sum=sum+((FindNumberOfNeighbours(matrix,i)-
xm)*(FindNumberOfNeighbours(matrix,i)-xm));
  }

  d=sqrt(sum/(n*(n-1)));

  return d;
}

/*kyriws programma*/
void main()
{
 float x,y,r;
 float f[N];
 int i,j,components;
 int NumberOfEdgeNodes=0;

 float node[3][N];

 for ( i=0;i<3;i++)
   for (j=0;j<N;j++)
     node[i][j]=0;


 int matrix_neighbour[N][N];
 Tools myTool;

 for ( i=0;i<N;i++)
   for (j=0;j<N;j++)
     matrix_neighbour[i][j]=0;

 for(i=0;i<N;i++)
       f[i]=0.0;

 srand(time(NULL));

 for(i=0;i<N;i++)
 {
```

```
   x=((rand()%32767)/32767.0)*L;

   y=((rand()%32767)/32767.0)*L;

   node[0][i]=x;        //dinontai oi sintetagmenes
   node[1][i]=y;        //stoys komboys


  }


 for(i=0;i<N;i++)
    node[2][i]=RMAX;    //dinetai arxika i timi tis megistis aktinas
se oloys toys komboys

 myTool.CreateMatrixOfNeighbourhood(matrix_neighbour,node,N);
//ipologismos twn geitonwn sto arxiko diktio


 for(i=0;i<N;i++)
   f[i]=myTool.FindConeAngles(matrix_neighbour,i,node); //eysresi
megistis gwnias kwnoy gia kathe kombo

 components=myTool.FindNumberOfComponents(matrix_neighbour);

 char filename1[80];
 char buffer1[1000];
 cout<<"File name 1:";
 cin>> filename1;

 ofstream fout1(filename1);             //anoigma arxeiou gia arxiko
diktio
   if(!fout1)                           //elegxos an anoikse swsta
   {
    cout<<"Unable to open"<<filename1<<"for appending\n";
   }

 fout1<<"components : "<<components<<"\n\n";

 fout1<<"id\tx\ty\tR\tneighbours\n";

 for(i=0;i<N;i++)
 {

fout1<<setprecision(4)<<i<<"\t"<<node[0][i]<<"\t"<<node[1][i]<<"\t"<<
RMAX<<"\t"<<myTool.FindNumberOfNeighbours(matrix_neighbour,i)<<"\n";
 }

 fout1<<setprecision(4)<<"\narxiki mesi aktina:
"<<myTool.FindAverageRadius(node,N)<<"\n";
 fout1<<setprecision(4)<<"arxikos mesos arithmos geitonwn:
"<<myTool.FindAverageNumberOfNeighbours(matrix_neighbour,N)<<"\n";
 fout1<<setprecision(4)<<"arxiki variance aktinas:
"<<myTool.FindVarianceOfAverageRadius(node,N)<<"\n";
 fout1<<setprecision(4)<<"arxiki variance geitonwn:
"<<myTool.FindVarianceOfAverageNeighbours(matrix_neighbour,N)<<"\n";
 fout1.close();                        //kleisimo prwtou arxeiou


 for(i=0;i<N;i++)
 {
```

```
  if (f[i]>=180.0)
  {
     cout<<"\nError-max radius with f["<<i<<"]>=180\n";
       NumberOfEdgeNodes+=1;
  }
  else
  {
   do
   {
    r=node[2][i];
    myTool.CreateMatrixOfNeighbourhood(matrix_neighbour,node,N);
    cout<<"\n";

    f[i]=myTool.FindConeAngles(matrix_neighbour,i,node);

    r=r-STEP; //stadiaki meiwsi tis aktinas mexri na kataliksoyme
stin epithimiti

    node[2][i]=r;
   } while((f[i]<180.0)&&(r>0)); //kataligoyme se aktina otan broyme
megisti gwnia kwnoy >180 i otan i aktina parei arnitiki timi

   node[2][i]=r+STEP; //ayksisi tis aktinas giati sto teleytaio bima
eixe meiwthei mia fora parapanw

   cout<<"\ni aktina tou kombou "<<i<<" einai "<<node[2][i];
  }

 }

 cout<<"\nNumber Of Edge Nodes: "<<NumberOfEdgeNodes;
 components=myTool.FindNumberOfComponents(matrix_neighbour);

 char filename2[80];
 char buffer2[1000];
 cout<<"\nFile name 2:";
 cin>> filename2;

 ofstream fout2(filename2);             //anoigma arxeiou gia teliko
diktio
   if(!fout2)                           //elegxos an anoikse swsta
   {
    cout<<"Unable to open"<<filename2<<"for appending\n";
   }

 fout2<<"components : "<<components<<"\n\n";

 fout2<<"id\tx\ty\tR\tneighbours\n";

 for(i=0;i<N;i++)
 {

fout2<<setprecision(4)<<i<<"\t"<<node[0][i]<<"\t"<<node[1][i]<<"\t"<<
node[2][i]<<"\t"<<myTool.FindNumberOfNeighbours(matrix_neighbour,i)<<
"\n";
 }

 fout2<<setprecision(4)<<"\nteliki mesi aktina:
"<<myTool.FindAverageRadius(node,N)<<"\n";
 fout2<<setprecision(4)<<"telikos mesos arithmos geitonwn:
"<<myTool.FindAverageNumberOfNeighbours(matrix_neighbour,N)<<"\n";
```

```
 fout2<<setprecision(4)<<"teliki variance aktinas:
"<<myTool.FindVarianceOfAverageRadius(node,N)<<"\n";
 fout2<<setprecision(4)<<"teliki variance geitonwn:
"<<myTool.FindVarianceOfAverageNeighbours(matrix_neighbour,N)<<"\n";
 fout2.close();                          //kleisimo deuterou arxeiou


}
```

## Αλγόριθμος ΙΙ

```
#include <iostream.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <fstream.h>
#include <iomanip.h>

#define N 441
#define RMAX 300
#define L 2100
#define STEP 0.5

/*klasi tools pou periexei tis genikes sinartiseis tou programmatos*/
class Tools
{
public:

  float Distance(float x1,float y1,float x2,float y2);
  void CreateMatrixOfNeighbourhood(int matrix[N][N],float
node[3][N],int n);

  void qsort_auxil(float a[],int lower,int upper);
  void qsort(float a[],int n);

  float FindConeAngles(int matrix[N][N],int i,float node[3][N]);
  int FindNumberOfNeighbours(int matrix[N][N],int i);
  int FindMaxOfMatrix(int matrix[N]);
  int FindMinOfMatrix(int matrix[N]);
  int FindNumberOfComponents(int matrix[N][N]);
  float FindAverageRadius(float node[3][N],int n);
  float FindAverageNumberOfNeighbours(int matrix[N][N],int n);
  float FindVarianceOfAverageRadius(float node[3][N],int n);
  float FindVarianceOfAverageNeighbours(int matrix[N][N],int n);

private:
   int matrix[N][N];
   float node[3][N];
};

/*sinartisi Distance pou ipologizei tin apostasi metaksi
dio simeiwn A(x1,y1) kai B(x2,y2)*/
float Tools::Distance(float x1,float y1,float x2,float y2)
{
 return sqrt(((x2-x1)*(x2-x1))+((y2-y1)*(y2-y1)));
}

/*sinartisi CreateMatrixOfNeighbourhood pou ipologizei
ton pinaka geitonias pairnontas ws parametrous ton arithmo
twn kombwn kai ton pinaka pou periexei tis sintetagmenes,
```

```
to id kai tin aktina tous*/
void Tools::CreateMatrixOfNeighbourhood(int matrix[N][N],float
node[3][N],int n)
{
  float x1,y1,x2,y2;
  float d,r;
  int i,j;

  for(i=0;i<N;i++)
   for(j=0;j<N;j++)
       matrix[i][j]=0;


  for (i=0;i<n;i++)
  {
   x1=node[0][i];
   y1=node[1][i];
   r=node[2][i];

   for (j=0;j<n;j++)           //diatrexontai oloi oi komboi kai
simfwna me tin aktina poy
   {                           //exei o ekastote kombos elegxetai gia
olous tous ipoloipous
      if(j!=i)
      {
       x2=node[0][j];
       y2=node[1][j];
       d=Distance(x1,y1,x2,y2);

       if ( d <= r )           //an i apostasi tous apo ton kombo
einai mikroteri tis aktinas
            matrix[i][j]=1;    //tou diladi an einai mesa stin
embeleia tou
         }                       //an nai topotheteitai to 1 sto
antistoixo stoixeio tou pinaka geitonias
        else matrix[i][j]=0;     //alliws to 0
     }
  }
}

/*sinartisi qsort_auxil pou xrisimopoiitai apo tin qsort
gia tin diataksi twn stoixeiwn enos pinaka se ayksousa seira*/
void Tools::qsort_auxil(float a[],int lower,int upper)
{
  if (lower<upper)
    {
     float x=a[(lower+upper)/2];
     int i,j;

     for (i=lower,j=upper;i<=j;i++,j--)
       {
          while (a[i]<x) i++;
          while (a[j]>x) j--;

          if (i<=j)
            {
           float temp=a[i];

           a[i]=a[j];
           a[j]=temp;
            }
```

```
       }
     qsort_auxil(a,lower,j);
     qsort_auxil(a,i,upper);
    }
}

void Tools::qsort(float a[],int n)
{
  qsort_auxil(a,0,n-1);
}


/*sinartisi FindConeAngles pou briskei tis cone angles enos kombou
kai epistrefei ti megisti*/
float Tools::FindConeAngles(int matrix[N][N],int i,float node[3][N])
 {
 int j,k,l;
 float xnew,ynew,biggest;
 float cosf,f,g;
 float matrix_angles[N];


 for(l=0;l<N;l++)
 {
   matrix_angles[l]=0.0;
 }

 k=0;

 for (j=0;j<N;j++)
   {

    if (matrix[i][j]==1)             //ean to stoixeio tou pinaka
geitonias einai 1
     {
       xnew=node[0][j]-node[0][i];   //metasximatismos twn
sintetagmenwn tou kathe geitonikou
       ynew=node[1][j]-node[1][i];   //kombou se sxesi me ton
kentriko pou orizetai arxi twn axonwn

       if((xnew==0.0)&&(ynew>0.0))   //ean o geitonas einai stin x=0
kai y>0
         {
            f=90.0;                     //tote i gwnia prokiptei 90
            matrix_angles[k]=f;
           }
       else
         {
            if ((xnew==0.0)&&(ynew<0.0)) //ean o geitonas einai stin
x=0 kai y<0
              {
             f=270.0;                     //tote i gwnia prokiptei 270
             matrix_angles[k]=f;
              }
            else
              {
             if ((ynew==0.0)&&(xnew>0.0)) //ean o geitonas einai stin
y=0 kai x>0
                 {
               f=0.0;                      //tote i gwnia prokiptei 0
               matrix_angles[k]=f;
```

```
              }
            else
              {
             if ((ynew==0.0)&&(xnew<0.0)) //ean o geitonas einai
stin y=0 kai x<0
              {
                f=180.0;                   //tote i gwnia einai 180
               matrix_angles[k]=f;
                }
             else                        //ean den anikei se mia
apo tis 4 eidikes periptwseis
             {
                if
(((xnew>0.0)&&(ynew>0.0))||((xnew<0.0)&&(ynew>0.0))) //ean o kombos
brisketai sto 1 i sto 2 tetartimorio
                {

cosf=(1*xnew+1*ynew)/((sqrt(2))*Distance(0.0,0.0,xnew,ynew));
                 f=acos(cosf)*180/3.141592654;

                   matrix_angles[k]=f;     //ipologizetai i gwnia
toy kathe geitona me to thetiko tmima tou orizontiou axona
               }
             else
                {
                   if
(((xnew<0.0)&&(ynew<0.0))||((xnew>0.0)&&(ynew<0.0))) //ean einai sto
3 i sto 4
                 {

cosf=(1*xnew+1*ynew)/((sqrt(2))*Distance(0.0,0.0,xnew,ynew));
                         f=acos(cosf)*180/3.141592654;

                         matrix_angles[k]=360.0-f; //afairesi me to
360 dioti o tipos dinei panta ti mikri gwnia
                   }
                 }
               }
           }
         }
        }
   k=k+1;
       }
     }

 if (k==1)
   return 360.0;


  qsort(matrix_angles,k);   //diataxi tou pinaka me tis gwnies se
ayksoysa seira

 biggest=-12000.0;

 for(i=0;i<k-1;i++)
 {
   g=matrix_angles[i+1]-matrix_angles[i];  //afairesi kathe stoixeiou
apo to epomeno gia na prokipsoun telika oi cone angles

   if (g>biggest)
       biggest=g;
```

```
 }

 g=360.0-matrix_angles[k-1]+matrix_angles[0];//ipologismos tis
teleytaias gwnias me eidiko typo
                                                //dioti to apotelesma
den tha itan swsto me afairesi
 if (g>biggest)
   return g;

 return biggest;
}

/*sinartisi FindNumberOfNeighbours pou briskei ton arithmo
 twn geitonwn enos kombou*/
int Tools::FindNumberOfNeighbours(int matrix[N][N],int i)
{
  int j,k;

  k=0;

  for(j=0;j<N;j++)
  {
       if (matrix[i][j]==1)
             k=k+1;
  }

  return k;
}

/*sinartisi FindMaxOfMatrix pou briskei to megisto stoixeio enos
pinaka*/
int Tools::FindMaxOfMatrix(int matrix[N])
{
 int k,i;

 k=matrix[0];

 for(i=0;i<N;i++)
 {
       if (matrix[i]>k)
             k=matrix[i];
 }

 return k;
}

/*sinartisi FindMinOfMatrix pou briskei to elaxisto stoixeio enos
pinaka*/
int Tools::FindMinOfMatrix(int matrix[N])
{
 int k,i;

 k=matrix[0];

 for(i=0;i<N;i++)
 {
       if (matrix[i]<k)
             k=matrix[i];
 }

 return k;
```

```
}

/*sinartisi FindNumberOfComponents pou briskei ton arithmo
 twn components enos diktioy*/
int Tools::FindNumberOfComponents(int matrix[N][N])
{
 int components;
 int k,m,n,b;
 int nodeVisited[N];
 int nodeStack[N];

 components=0;

 for(k=0;k<N;k++)
 {
   nodeVisited[k]=0;
   nodeStack[k]=0;
 }

 m=0;
 nodeVisited[m]=1;

 while ((FindMinOfMatrix(nodeVisited))==0) //oso iparxoun komboi poy
den toys exoume episkeftei
 {
   nodeVisited[m]=1;

   for(n=m+1;n<N;n++)
   {
        if ((matrix[m][n])==1)              //an o kombos n einai
geitonas toy m
        {
             nodeVisited[n]=1;                //markaretai o n ws
kombos poy ton episkeftikame
             nodeStack[n]=1;                  //markarw sti stoiba
ton n kanontas to stoixeio 1
        }
   }

   while ((FindMaxOfMatrix(nodeStack))==1)
   {
       k=0;

       while ((nodeStack[k])==0)
       {
       k=k+1;
       }

       nodeStack[k]=0;                     //briskei to prwto mi
mideniko stoixeio kai to midenizei

       for(n=0;n<N;n++)
       {
         if ((matrix[k][n])==1)            //an yparxei geitonas
aytoy toy stoixeioy
         {
         if (nodeVisited[n]==0)
         {
               nodeVisited[n]=1;            //markaretai o n san
kombos poy episkeftikame
               nodeStack[n]=1;
```

```
            }
        }
        }
    }

    components=components+1;

    b=0;
    m=0;

    while (b<N-1)
    {
      b=b+1;
        if (nodeVisited[b]==0)
        {
          m=b;    //stin oysia briskei ton prwto kombo poy den
enwnetai me toys proigoymenoys
          b=N-1;  //kai an ton brei termatizei ayto to while kai
trexei apo tin arxi to ekswteriko while
        }
    }
 } //to ekswteriko while termatizetai otan exei ginei episkepsi se
oloys toys komboys

return components;
}

/*sinartisi pou ipologizei ti mesi aktina*/
float Tools::FindAverageRadius(float node[3][N],int n)
{
  int i;
  float sum=0;

  for(i=0;i<n;i++)
      sum=sum+node[2][i];

  sum=sum/n;
  return sum;
}

/*sinartisi pou ipologizei to meso arithmo geitonwn*/
float Tools::FindAverageNumberOfNeighbours(int matrix[N][N],int n)
{
  int i;
  float sum=0;

  for(i=0;i<n;i++)
    sum=sum+FindNumberOfNeighbours(matrix,i);
  sum=sum/n;
  return sum;
}

/*sinartisi FindVarianceOfAverageRadius pou ipologizei ti variance
tis aktinas*/
float Tools::FindVarianceOfAverageRadius(float node[3][N],int n)
{
  int i;
  float rm;
  float sum=0.0;
  float d;
```

```
  rm=FindAverageRadius(node,n);

  for(i=0;i<n;i++)
  {
        sum=sum+((node[2][i]-rm)*(node[2][i]-rm));
  }

  d=sqrt(sum/(n*(n-1)));

  return d;
}

/*sinartisi FindVarianceOfAverageNeighbours pou ipologizei ti
variance twn geitonwn*/
float Tools::FindVarianceOfAverageNeighbours(int matrix[N][N],int n)
{
  int i,xm;
  float sum=0.0;
  float d;

  xm=FindAverageNumberOfNeighbours(matrix,n);

  for(i=0;i<n;i++)
  {
        sum=sum+((FindNumberOfNeighbours(matrix,i)-
xm)*(FindNumberOfNeighbours(matrix,i)-xm));
  }

  d=sqrt(sum/(n*(n-1)));

  return d;
}

/*kyriws programma*/
void main()
{
 float x,y,r,maxdistance,distanceBetweenNodes;
 float f[N];
 int i,j,components;

 float node[3][N];

 for ( i=0;i<3;i++)
   for (j=0;j<N;j++)
     node[i][j]=0;


 int matrix_neighbour[N][N];
 Tools myTool;

 for ( i=0;i<N;i++)
   for (j=0;j<N;j++)
     matrix_neighbour[i][j]=0;

 for(i=0;i<N;i++)
       f[i]=0.0;

 srand(time(NULL));

 for(i=0;i<N;i++)
 {
```

```
   x=((rand()%32767)/32767.0)*L;

   y=((rand()%32767)/32767.0)*L;

   node[0][i]=x;        //dinontai oi sintetagmenes
   node[1][i]=y;        //stoys komboys


  }


 for(i=0;i<N;i++)
    node[2][i]=RMAX;    //dinetai arxika i timi tis megistis aktinas
se oloys toys komboys

 myTool.CreateMatrixOfNeighbourhood(matrix_neighbour,node,N);
//ipologismos twn geitonwn sto arxiko diktio


 for(i=0;i<N;i++)
   f[i]=myTool.FindConeAngles(matrix_neighbour,i,node); //eysresi
megistis gwnias kwnoy gia kathe kombo

 components=myTool.FindNumberOfComponents(matrix_neighbour);

 char filename1[80];
 char buffer1[1000];
 cout<<"File name 1:";
 cin>> filename1;

 ofstream fout1(filename1);             //anoigma arxeiou gia arxiko
diktio
   if(!fout1)                           //elegxos an anoikse swsta
   {
    cout<<"Unable to open"<<filename1<<"for appending\n";
   }

 fout1<<"components : "<<components<<"\n\n";

 fout1<<"id\tx\ty\tR\tneighbours\n";

 for(i=0;i<N;i++)
 {

fout1<<setprecision(4)<<i<<"\t"<<node[0][i]<<"\t"<<node[1][i]<<"\t"<<
RMAX<<"\t"<<myTool.FindNumberOfNeighbours(matrix_neighbour,i)<<"\n";
 }

 fout1<<setprecision(4)<<"\narxiki mesi aktina:
"<<myTool.FindAverageRadius(node,N)<<"\n";
 fout1<<setprecision(4)<<"arxikos mesos arithmos geitonwn:
"<<myTool.FindAverageNumberOfNeighbours(matrix_neighbour,N)<<"\n";
 fout1<<setprecision(4)<<"arxiki variance aktinas:
"<<myTool.FindVarianceOfAverageRadius(node,N)<<"\n";
 fout1<<setprecision(4)<<"arxiki variance geitonwn:
"<<myTool.FindVarianceOfAverageNeighbours(matrix_neighbour,N)<<"\n";
 fout1.close();                           //kleisimo prwtou arxeiou

 for(i=0;i<N;i++)
 {
```

```
  if (f[i]>=180.0)
     cout<<"\nError-max radius with f["<<i<<"]>=180\n";
  else
  {
   do
   {
    r=node[2][i];
    myTool.CreateMatrixOfNeighbourhood(matrix_neighbour,node,N);
    cout<<"\n";

    f[i]=myTool.FindConeAngles(matrix_neighbour,i,node);

    r=r-STEP; //stadiaki meiwsi tis aktinas mexri na kataliksoyme
stin epithimiti

    node[2][i]=r;
   } while((f[i]<180.0)&&(r>0)); //kataligoyme se aktina otan broyme
megisti gwnia kwnoy >180 i otan i aktina parei arnitiki timi

   node[2][i]=r+STEP; //ayksisi tis aktinas giati sto teleytaio bima
eixe meiwthei mia fora parapanw

   cout<<"\ni aktina tou kombou prin"<<i<<" einai "<<node[2][i];

   /* meros tou algorithmou 2 opou diaforopoieitai apo ton ena*/
   maxdistance=0;

   for(j=0;j<N;j++)
   {
         if ((matrix_neighbour[j][i]==1)&&(node[2][j]!=RMAX))
//elegxos se poious kombous einai geitonas o i
                                                                //an
einai geitonas se edge_node
         {

distanceBetweenNodes=myTool.Distance(node[0][i],node[1][i],node[0][j]
,node[1][j]);

               if (distanceBetweenNodes>maxdistance)
                     maxdistance=distanceBetweenNodes; //euresi tis
megistis apostasis apo kapoion geitona kombo pros ton i
         }

   }

   if (node[2][i]<maxdistance) //sigkrisi me tin aktina pou brethike
apo to prwto kommati
         node[2][i]=maxdistance; //epilogi tou megaliterou


   cout<<"\ni aktina tou kombou "<<i<<" einai "<<node[2][i];
  }

 }

 components=myTool.FindNumberOfComponents(matrix_neighbour);

 char filename2[80];
 char buffer2[1000];
 cout<<"\nFile name 2:";
 cin>> filename2;
```

```
 ofstream fout2(filename2);             //anoigma arxeiou gia teliko
diktio
   if(!fout2)                           //elegxos an anoikse swsta
   {
    cout<<"Unable to open"<<filename2<<"for appending\n";
   }

 fout2<<"components : "<<components<<"\n\n";

 fout2<<"id\tx\ty\tR\tneighbours\n";

 for(i=0;i<N;i++)
 {

fout2<<setprecision(4)<<i<<"\t"<<node[0][i]<<"\t"<<node[1][i]<<"\t"<<
node[2][i]<<"\t"<<myTool.FindNumberOfNeighbours(matrix_neighbour,i)<<
"\n";
 }

 fout2<<setprecision(4)<<"\nteliki mesi aktina:
"<<myTool.FindAverageRadius(node,N)<<"\n";
 fout2<<setprecision(4)<<"telikos mesos arithmos geitonwn:
"<<myTool.FindAverageNumberOfNeighbours(matrix_neighbour,N)<<"\n";
 fout2<<setprecision(4)<<"teliki variance aktinas:
"<<myTool.FindVarianceOfAverageRadius(node,N)<<"\n";
 fout2<<setprecision(4)<<"teliki variance geitonwn:
"<<myTool.FindVarianceOfAverageNeighbours(matrix_neighbour,N)<<"\n";
 fout2.close();                         //kleisimo deuterou arxeiou
                        //kleisimo deuterou arxeiou


}
```

## Αλγόριθμος III

```
#include <iostream.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <fstream.h>
#include <iomanip.h>

#define N 196
#define RMAX 300
#define L 1400
#define STEP 0.5
#define F 120

/*klasi tools pou periexei tis genikes sinartiseis tou programmatos*/
class Tools
{
public:

  float Distance(float x1,float y1,float x2,float y2);
  void CreateMatrixOfNeighbourhood(int matrix[N][N],float
node[3][N],int n);

  void qsort_auxil(float a[],int lower,int upper);
  void qsort(float a[],int n);
```

```
  float FindConeAngles(int matrix[N][N],int i,float node[3][N]);
  int FindNumberOfNeighbours(int matrix[N][N],int i);
  int FindMaxOfMatrix(int matrix[N]);
  int FindMinOfMatrix(int matrix[N]);
  int FindNumberOfComponents(int matrix[N][N]);
  float FindAverageRadius(float node[3][N],int n);
  float FindAverageNumberOfNeighbours(int matrix[N][N],int n);
  float FindVarianceOfAverageRadius(float node[3][N],int n);
  float FindVarianceOfAverageNeighbours(int matrix[N][N],int n);

private:
   int matrix[N][N];
   float node[3][N];
};

/*sinartisi Distance pou ipologizei tin apostasi metaksi
dio simeiwn A(x1,y1) kai B(x2,y2)*/
float Tools::Distance(float x1,float y1,float x2,float y2)
{
 return sqrt(((x2-x1)*(x2-x1))+((y2-y1)*(y2-y1)));
}

/*sinartisi CreateMatrixOfNeighbourhood pou ipologizei
ton pinaka geitonias pairnontas ws parametrous ton arithmo
twn kombwn kai ton pinaka pou periexei tis sintetagmenes,
to id kai tin aktina tous*/
void Tools::CreateMatrixOfNeighbourhood(int matrix[N][N],float
node[3][N],int n)
{
  float x1,y1,x2,y2;
  float d,r;
  int i,j;

  for(i=0;i<N;i++)
   for(j=0;j<N;j++)
       matrix[i][j]=0;


  for (i=0;i<n;i++)
  {
   x1=node[0][i];
   y1=node[1][i];
   r=node[2][i];

   for (j=0;j<n;j++)           //diatrexontai oloi oi komboi kai
simfwna me tin aktina poy
   {                           //exei o ekastote kombos elegxetai gia
olous tous ipoloipous
         if(j!=i)
      {
       x2=node[0][j];
       y2=node[1][j];
       d=Distance(x1,y1,x2,y2);

       if ( d <= r )           //an i apostasi tous apo ton kombo
einai mikroteri tis aktinas
            matrix[i][j]=1;    //tou diladi an einai mesa stin
embeleia tou
         }                       //an nai topotheteitai to 1 sto
antistoixo stoixeio tou pinaka geitonias
```

```
        else matrix[i][j]=0;     //alliws to 0
      }
  }
}

/*sinartisi qsort_auxil pou xrisimopoiitai apo tin qsort
gia tin diataksi twn stoixeiwn enos pinaka se ayksousa seira*/
void Tools::qsort_auxil(float a[],int lower,int upper)
{
  if (lower<upper)
    {
     float x=a[(lower+upper)/2];
     int i,j;

     for (i=lower,j=upper;i<=j;i++,j--)
       {
          while (a[i]<x) i++;
          while (a[j]>x) j--;

          if (i<=j)
            {
           float temp=a[i];

           a[i]=a[j];
           a[j]=temp;
            }
       }
     qsort_auxil(a,lower,j);
     qsort_auxil(a,i,upper);
    }
}

void Tools::qsort(float a[],int n)
{
  qsort_auxil(a,0,n-1);
}


/*sinartisi FindConeAngles pou briskei tis cone angles enos kombou
kai epistrefei ti megisti*/
float Tools::FindConeAngles(int matrix[N][N],int i,float node[3][N])
 {
 int j,k,l;
 float xnew,ynew,biggest;
 float cosf,f,g;
 float matrix_angles[N];


 for(l=0;l<N;l++)
 {
   matrix_angles[l]=0.0;
 }


 k=0;

 for (j=0;j<N;j++)
   {

    if (matrix[i][j]==1)             //ean to stoixeio tou pinaka
geitonias einai 1
```

```
     {
       xnew=node[0][j]-node[0][i];   //metasximatismos twn
sintetagmenwn tou kathe geitonikou
       ynew=node[1][j]-node[1][i];   //kombou se sxesi me ton
kentriko pou orizetai arxi twn axonwn

       if((xnew==0.0)&&(ynew>0.0))   //ean o geitonas einai stin x=0
kai y>0
         {
            f=90.0;                     //tote i gwnia prokiptei 90
            matrix_angles[k]=f;
           }
       else
         {
            if ((xnew==0.0)&&(ynew<0.0)) //ean o geitonas einai stin
x=0 kai y<0
              {
             f=270.0;                     //tote i gwnia prokiptei 270
             matrix_angles[k]=f;
              }
            else
              {
             if ((ynew==0.0)&&(xnew>0.0)) //ean o geitonas einai stin
y=0 kai x>0
                  {
               f=0.0;                      //tote i gwnia prokiptei 0
               matrix_angles[k]=f;
               }
             else
               {
              if ((ynew==0.0)&&(xnew<0.0)) //ean o geitonas einai
stin y=0 kai x<0
               {
                  f=180.0;                   //tote i gwnia einai 180
                matrix_angles[k]=f;
                  }
              else                          //ean den anikei se mia
apo tis 4 eidikes periptwseis
              {
                  if
(((xnew>0.0)&&(ynew>0.0))||((xnew<0.0)&&(ynew>0.0))) //ean o kombos
brisketai sto 1 i sto 2 tetartimorio
                  {

cosf=(1*xnew+1*ynew)/((sqrt(2))*Distance(0.0,0.0,xnew,ynew));
                   f=acos(cosf)*180/3.141592654;

                     matrix_angles[k]=f;     //ipologizetai i gwnia
toy kathe geitona me to thetiko tmima tou orizontiou axona
                }
              else
                 {
                     if
(((xnew<0.0)&&(ynew<0.0))||((xnew>0.0)&&(ynew<0.0))) //ean einai sto
3 i sto 4
                   {

cosf=(1*xnew+1*ynew)/((sqrt(2))*Distance(0.0,0.0,xnew,ynew));
                          f=acos(cosf)*180/3.141592654;
```

```
                          matrix_angles[k]=360.0-f; //afairesi me to
360 dioti o tipos dinei panta ti mikri gwnia
                      }
                    }
                  }
              }
           }
          }
   k=k+1;
       }
     }

 if (k==1)
   return 360.0;


 qsort(matrix_angles,k);   //diataxi tou pinaka me tis gwnies se
ayksoysa seira

 biggest=-12000.0;

 for(i=0;i<k-1;i++)
 {
   g=matrix_angles[i+1]-matrix_angles[i];  //afairesi kathe stoixeiou
apo to epomeno gia na prokipsoun telika oi cone angles

   if (g>biggest)
       biggest=g;
 }

 g=360.0-matrix_angles[k-1]+matrix_angles[0];//ipologismos tis
teleytaias gwnias me eidiko typo
                                             //dioti to apotelesma
den tha itan swsto me afairesi
 if (g>biggest)
   return g;

 return biggest;
}

/*sinartisi FindNumberOfNeighbours pou briskei ton arithmo
 twn geitonwn enos kombou*/
int Tools::FindNumberOfNeighbours(int matrix[N][N],int i)
{
  int j,k;

  k=0;

  for(j=0;j<N;j++)
  {
       if (matrix[i][j]==1)
             k=k+1;
  }

  return k;
}

/*sinartisi FindMaxOfMatrix pou briskei to megisto stoixeio enos
pinaka*/
int Tools::FindMaxOfMatrix(int matrix[N])
{
```

```
 int k,i;

 k=matrix[0];

 for(i=0;i<N;i++)
 {
       if (matrix[i]>k)
             k=matrix[i];
 }

 return k;
}

/*sinartisi FindMinOfMatrix pou briskei to elaxisto stoixeio enos
pinaka*/
int Tools::FindMinOfMatrix(int matrix[N])
{
 int k,i;

 k=matrix[0];

 for(i=0;i<N;i++)
 {
       if (matrix[i]<k)
             k=matrix[i];
 }

 return k;
}

/*sinartisi FindNumberOfComponents pou briskei ton arithmo
 twn components enos diktioy*/
int Tools::FindNumberOfComponents(int matrix[N][N])
{
 int components;
 int k,m,n,b;
 int nodeVisited[N];
 int nodeStack[N];

 components=0;

 for(k=0;k<N;k++)
 {
   nodeVisited[k]=0;
   nodeStack[k]=0;
 }

 m=0;
 nodeVisited[m]=1;

 while ((FindMinOfMatrix(nodeVisited))==0) //oso iparxoun komboi poy
den toys exoume episkeftei
 {
   nodeVisited[m]=1;

   for(n=m+1;n<N;n++)
   {
        if ((matrix[m][n])==1)              //an o kombos n einai
geitonas toy m
        {
```

```
            nodeVisited[n]=1;                //markaretai o n ws
kombos poy ton episkeftikame
            nodeStack[n]=1;                  //markarw sti stoiba
ton n kanontas to stoixeio 1
        }
   }

   while ((FindMaxOfMatrix(nodeStack))==1)
   {
       k=0;

       while ((nodeStack[k])==0)
       {
       k=k+1;
       }

       nodeStack[k]=0;                     //briskei to prwto mi
mideniko stoixeio kai to midenizei

       for(n=0;n<N;n++)
       {
         if ((matrix[k][n])==1)            //an yparxei geitonas
aytoy toy stoixeioy
         {
         if (nodeVisited[n]==0)
         {
               nodeVisited[n]=1;            //markaretai o n san
kombos poy episkeftikame
               nodeStack[n]=1;
             }
       }
       }
   }

   components=components+1;

   b=0;
   m=0;

   while (b<N-1)
   {
     b=b+1;
       if (nodeVisited[b]==0)
       {
         m=b;    //stin oysia briskei ton prwto kombo poy den
enwnetai me toys proigoymenoys
         b=N-1;  //kai an ton brei termatizei ayto to while kai
trexei apo tin arxi to ekswteriko while
       }
   }
 } //to ekswteriko while termatizetai otan exei ginei episkepsi se
oloys toys komboys

return components;
}

/*sinartisi pou ipologizei ti mesi aktina*/
float Tools::FindAverageRadius(float node[3][N],int n)
{
  int i;
  float sum=0;
```

```
  for(i=0;i<n;i++)
      sum=sum+node[2][i];

  sum=sum/n;
  return sum;
}

/*sinartisi pou ipologizei to meso arithmo geitonwn*/
float Tools::FindAverageNumberOfNeighbours(int matrix[N][N],int n)
{
  int i;
  float sum=0;

  for(i=0;i<n;i++)
    sum=sum+FindNumberOfNeighbours(matrix,i);
  sum=sum/n;
  return sum;
}

/*sinartisi FindVarianceOfAverageRadius pou ipologizei ti variance
tis aktinas*/
float Tools::FindVarianceOfAverageRadius(float node[3][N],int n)
{
  int i;
  float rm;
  float sum=0.0;
  float d;

  rm=FindAverageRadius(node,n);

  for(i=0;i<n;i++)
  {
        sum=sum+((node[2][i]-rm)*(node[2][i]-rm));
  }

  d=sqrt(sum/(n*(n-1)));

  return d;
}

/*sinartisi FindVarianceOfAverageNeighbours pou ipologizei ti
variance twn geitonwn*/
float Tools::FindVarianceOfAverageNeighbours(int matrix[N][N],int n)
{
  int i,xm;
  float sum=0.0;
  float d;

  xm=FindAverageNumberOfNeighbours(matrix,n);

  for(i=0;i<n;i++)
  {
        sum=sum+((FindNumberOfNeighbours(matrix,i)-
xm)*(FindNumberOfNeighbours(matrix,i)-xm));
  }

  d=sqrt(sum/(n*(n-1)));

  return d;
}
```

```
/*kyriws programma*/
void main()
{
 float x,y,r;
 float f[N];
 int i,j,components;
 int NumberOfEdgeNodes=0;

 float node[3][N];

 for ( i=0;i<3;i++)
   for (j=0;j<N;j++)
     node[i][j]=0;


 int matrix_neighbour[N][N];
 Tools myTool;

 for ( i=0;i<N;i++)
   for (j=0;j<N;j++)
     matrix_neighbour[i][j]=0;

 for(i=0;i<N;i++)
       f[i]=0.0;

 srand(time(NULL));

 for(i=0;i<N;i++)
 {

   x=((rand()%32767)/32767.0)*L;

   y=((rand()%32767)/32767.0)*L;

   node[0][i]=x;        //dinontai oi sintetagmenes
   node[1][i]=y;        //stoys komboys


  }


 for(i=0;i<N;i++)
    node[2][i]=RMAX;    //dinetai arxika i timi tis megistis aktinas
se oloys toys komboys

 myTool.CreateMatrixOfNeighbourhood(matrix_neighbour,node,N);
//ipologismos twn geitonwn sto arxiko diktio


 for(i=0;i<N;i++)
   f[i]=myTool.FindConeAngles(matrix_neighbour,i,node); //eysresi
megistis gwnias kwnoy gia kathe kombo

 components=myTool.FindNumberOfComponents(matrix_neighbour);

 char filename1[80];
 char buffer1[1000];
 cout<<"File name 1:";
 cin>> filename1;
```

```
 ofstream fout1(filename1);             //anoigma arxeiou gia arxiko
diktio
   if(!fout1)                           //elegxos an anoikse swsta
   {
    cout<<"Unable to open"<<filename1<<"for appending\n";
   }

 fout1<<"components : "<<components<<"\n\n";

 fout1<<"id\tx\ty\tR\tneighbours\n";

 for(i=0;i<N;i++)
 {

fout1<<setprecision(4)<<i<<"\t"<<node[0][i]<<"\t"<<node[1][i]<<"\t"<<
RMAX<<"\t"<<myTool.FindNumberOfNeighbours(matrix_neighbour,i)<<"\n";
 }

 fout1<<setprecision(4)<<"\narxiki mesi aktina:
"<<myTool.FindAverageRadius(node,N)<<"\n";
 fout1<<setprecision(4)<<"arxikos mesos arithmos geitonwn:
"<<myTool.FindAverageNumberOfNeighbours(matrix_neighbour,N)<<"\n";
 fout1<<setprecision(4)<<"arxiki variance aktinas:
"<<myTool.FindVarianceOfAverageRadius(node,N)<<"\n";
 fout1<<setprecision(4)<<"arxiki variance geitonwn:
"<<myTool.FindVarianceOfAverageNeighbours(matrix_neighbour,N)<<"\n";
 fout1.close();                        //kleisimo prwtou arxeiou

 for(i=0;i<N;i++)
 {
  if (f[i]>=F)
  {
     cout<<"\nError-max radius with f["<<i<<"]>="<<F<<"\n";
       NumberOfEdgeNodes+=1;
  }
  else
  {
   do
   {
    r=node[2][i];
    myTool.CreateMatrixOfNeighbourhood(matrix_neighbour,node,N);
    cout<<"\n";

    f[i]=myTool.FindConeAngles(matrix_neighbour,i,node);

     r=r-STEP; //stadiaki meiwsi tis aktinas mexri na kataliksoyme
stin epithimiti

    node[2][i]=r;
   } while((f[i]<F)&&(r>0)); //kataligoyme se aktina otan broyme
megisti gwnia kwnoy >F i otan i aktina parei arnitiki timi

   node[2][i]=r+STEP; //ayksisi tis aktinas giati sto teleytaio bima
eixe meiwthei mia fora parapanw

   cout<<"\ni aktina tou kombou "<<i<<" einai "<<node[2][i];
  }

 }

 cout<<"\nNumber Of Edge Nodes: "<<NumberOfEdgeNodes;
```

```
 components=myTool.FindNumberOfComponents(matrix_neighbour);

 char filename2[80];
 char buffer2[1000];
 cout<<"\nFile name 2:";
 cin>> filename2;

 ofstream fout2(filename2);             //anoigma arxeiou gia teliko
diktio
   if(!fout2)                           //elegxos an anoikse swsta
   {
    cout<<"Unable to open"<<filename2<<"for appending\n";
   }

 fout2<<"components : "<<components<<"\n\n";

 fout2<<"id\tx\ty\tR\tneighbours\n";

 for(i=0;i<N;i++)
 {

fout2<<setprecision(4)<<i<<"\t"<<node[0][i]<<"\t"<<node[1][i]<<"\t"<<
node[2][i]<<"\t"<<myTool.FindNumberOfNeighbours(matrix_neighbour,i)<<
"\n";
 }

 fout2<<setprecision(4)<<"\nteliki mesi aktina:
"<<myTool.FindAverageRadius(node,N)<<"\n";
 fout2<<setprecision(4)<<"telikos mesos arithmos geitonwn:
"<<myTool.FindAverageNumberOfNeighbours(matrix_neighbour,N)<<"\n";
 fout2<<setprecision(4)<<"teliki variance aktinas:
"<<myTool.FindVarianceOfAverageRadius(node,N)<<"\n";
 fout2<<setprecision(4)<<"teliki variance geitonwn:
"<<myTool.FindVarianceOfAverageNeighbours(matrix_neighbour,N)<<"\n";
 fout2.close();                         //kleisimo deuterou arxeiou


}
```

## Αλγόριθμος IV

```
#include <iostream.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <fstream.h>
#include <iomanip.h>

#define N 441
#define RMAX 300
#define L 2100
#define D 150
#define STEP 0.5
#define F 120

/*klasi tools pou periexei tis genikes sinartiseis tou programmatos*/
class Tools
{
public:
```

```
  float Distance(float x1,float y1,float x2,float y2);
  void CreateMatrixOfNeighbourhood(int matrix[N][N],float
node[3][N],int n);

  void qsort_auxil(float a[],int lower,int upper);
  void qsort(float a[],int n);

  float FindConeAngles(int matrix[N][N],int i,float node[3][N]);
  int FindNumberOfNeighbours(int matrix[N][N],int i);
  int FindMaxOfMatrix(int matrix[N]);
  int FindMinOfMatrix(int matrix[N]);
  int FindNumberOfComponents(int matrix[N][N]);
  float FindAverageRadius(float node[3][N],int n);
  float FindAverageNumberOfNeighbours(int matrix[N][N],int n);
  float FindVarianceOfAverageRadius(float node[3][N],int n);
  float FindVarianceOfAverageNeighbours(int matrix[N][N],int n);


private:
   int matrix[N][N];
   float node[3][N];
};

/*sinartisi Distance pou ipologizei tin apostasi metaksi
dio simeiwn A(x1,y1) kai B(x2,y2)*/
float Tools::Distance(float x1,float y1,float x2,float y2)
{
 return sqrt(((x2-x1)*(x2-x1))+((y2-y1)*(y2-y1)));
}

/*sinartisi CreateMatrixOfNeighbourhood pou ipologizei
ton pinaka geitonias pairnontas ws parametrous ton arithmo
twn kombwn kai ton pinaka pou periexei tis sintetagmenes,
to id kai tin aktina tous*/
void Tools::CreateMatrixOfNeighbourhood(int matrix[N][N],float
node[3][N],int n)
{
  float x1,y1,x2,y2;
  float d,r;
  int i,j;

  for(i=0;i<N;i++)
   for(j=0;j<N;j++)
       matrix[i][j]=0;


  for (i=0;i<n;i++)
  {
   x1=node[0][i];
   y1=node[1][i];
   r=node[2][i];

   for (j=0;j<n;j++)           //diatrexontai oloi oi komboi kai
simfwna me tin aktina poy
   {                           //exei o ekastote kombos elegxetai gia
olous tous ipoloipous
         if(j!=i)
      {
       x2=node[0][j];
       y2=node[1][j];
       d=Distance(x1,y1,x2,y2);
```

```
       if ( d <= r )           //an i apostasi tous apo ton kombo
einai mikroteri tis aktinas
            matrix[i][j]=1;    //tou diladi an einai mesa stin
embeleia tou
         }                       //an nai topotheteitai to 1 sto
antistoixo stoixeio tou pinaka geitonias
        else matrix[i][j]=0;     //alliws to 0
     }
  }
}

/*sinartisi qsort_auxil pou xrisimopoiitai apo tin qsort
gia tin diataksi twn stoixeiwn enos pinaka se ayksousa seira*/
void Tools::qsort_auxil(float a[],int lower,int upper)
{
  if (lower<upper)
    {
     float x=a[(lower+upper)/2];
     int i,j;

     for (i=lower,j=upper;i<=j;i++,j--)
       {
          while (a[i]<x) i++;
          while (a[j]>x) j--;

          if (i<=j)
            {
           float temp=a[i];

           a[i]=a[j];
           a[j]=temp;
            }
       }
     qsort_auxil(a,lower,j);
     qsort_auxil(a,i,upper);
    }
}

void Tools::qsort(float a[],int n)
{
  qsort_auxil(a,0,n-1);
}


/*sinartisi FindConeAngles pou briskei tis cone angles enos kombou
kai epistrefei ti megisti*/
float Tools::FindConeAngles(int matrix[N][N],int i,float node[3][N])
 {
 int j,k,l;
 float xnew,ynew,biggest;
 float cosf,f,g;
 float matrix_angles[N];


 for(l=0;l<N;l++)
 {
   matrix_angles[l]=0.0;
 }
```

```
 k=0;

 for (j=0;j<N;j++)
   {

    if (matrix[i][j]==1)             //ean to stoixeio tou pinaka
geitonias einai 1
     {
       xnew=node[0][j]-node[0][i];   //metasximatismos twn
sintetagmenwn tou kathe geitonikou
       ynew=node[1][j]-node[1][i];   //kombou se sxesi me ton
kentriko pou orizetai arxi twn axonwn

       if((xnew==0.0)&&(ynew>0.0))   //ean o geitonas einai stin x=0
kai y>0
         {
            f=90.0;                     //tote i gwnia prokiptei 90
            matrix_angles[k]=f;
           }
       else
         {
            if ((xnew==0.0)&&(ynew<0.0)) //ean o geitonas einai stin
x=0 kai y<0
              {
             f=270.0;                     //tote i gwnia prokiptei 270
             matrix_angles[k]=f;
              }
            else
              {
             if ((ynew==0.0)&&(xnew>0.0)) //ean o geitonas einai stin
y=0 kai x>0
                 {
               f=0.0;                      //tote i gwnia prokiptei 0
               matrix_angles[k]=f;
               }
             else
               {
              if ((ynew==0.0)&&(xnew<0.0)) //ean o geitonas einai
stin y=0 kai x<0
               {
                  f=180.0;                    //tote i gwnia einai 180
                matrix_angles[k]=f;
                  }
              else                         //ean den anikei se mia
apo tis 4 eidikes periptwseis
              {
                  if
(((xnew>0.0)&&(ynew>0.0))||((xnew<0.0)&&(ynew>0.0))) //ean o kombos
brisketai sto 1 i sto 2 tetartimorio
                  {

cosf=(1*xnew+1*ynew)/((sqrt(2))*Distance(0.0,0.0,xnew,ynew));
                   f=acos(cosf)*180/3.141592654;

                     matrix_angles[k]=f;     //ipologizetai i gwnia
toy kathe geitona me to thetiko tmima tou orizontiou axona
                 }
               else
                  {
```

```
                      if
(((xnew<0.0)&&(ynew<0.0))||((xnew>0.0)&&(ynew<0.0))) //ean einai sto
3 i sto 4
                   {

cosf=(1*xnew+1*ynew)/((sqrt(2))*Distance(0.0,0.0,xnew,ynew));
                           f=acos(cosf)*180/3.141592654;

                           matrix_angles[k]=360.0-f; //afairesi me to
360 dioti o tipos dinei panta ti mikri gwnia
                      }
                   }
                }
            }
         }
        }
   k=k+1;
       }
     }

 if (k==1)
   return 360.0;


  qsort(matrix_angles,k);   //diataxi tou pinaka me tis gwnies se
ayksoysa seira

 biggest=-12000.0;

 for(i=0;i<k-1;i++)
 {
   g=matrix_angles[i+1]-matrix_angles[i];  //afairesi kathe stoixeiou
apo to epomeno gia na prokipsoun telika oi cone angles

   if (g>biggest)
       biggest=g;
 }

 g=360.0-matrix_angles[k-1]+matrix_angles[0];//ipologismos tis
teleytaias gwnias me eidiko typo
                                             //dioti to apotelesma
den tha itan swsto me afairesi
 if (g>biggest)
   return g;

 return biggest;
}

/*sinartisi FindNumberOfNeighbours pou briskei ton arithmo
 twn geitonwn enos kombou*/
int Tools::FindNumberOfNeighbours(int matrix[N][N],int i)
{
  int j,k;

  k=0;

  for(j=0;j<N;j++)
  {
       if (matrix[i][j]==1)
             k=k+1;
  }
```

```
  return k;
}

/*sinartisi FindMaxOfMatrix pou briskei to megisto stoixeio enos
pinaka*/
int Tools::FindMaxOfMatrix(int matrix[N])
{
 int k,i;

 k=matrix[0];

 for(i=0;i<N;i++)
 {
       if (matrix[i]>k)
             k=matrix[i];
 }

 return k;
}

/*sinartisi FindMinOfMatrix pou briskei to elaxisto stoixeio enos
pinaka*/
int Tools::FindMinOfMatrix(int matrix[N])
{
 int k,i;

 k=matrix[0];

 for(i=0;i<N;i++)
 {
       if (matrix[i]<k)
             k=matrix[i];
 }

 return k;
}

/*sinartisi FindNumberOfComponents pou briskei ton arithmo
 twn components enos diktioy*/
int Tools::FindNumberOfComponents(int matrix[N][N])
{
 int components;
 int k,m,n,b;
 int nodeVisited[N];
 int nodeStack[N];

 components=0;

 for(k=0;k<N;k++)
 {
   nodeVisited[k]=0;
   nodeStack[k]=0;
 }

 m=0;
 nodeVisited[m]=1;

 while ((FindMinOfMatrix(nodeVisited))==0) //oso iparxoun komboi poy
den toys exoume episkeftei
 {
```

```
   nodeVisited[m]=1;

   for(n=m+1;n<N;n++)
   {
        if ((matrix[m][n])==1)              //an o kombos n einai
geitonas toy m
        {
             nodeVisited[n]=1;                //markaretai o n ws
kombos poy ton episkeftikame
             nodeStack[n]=1;                  //markarw sti stoiba
ton n kanontas to stoixeio 1
        }
   }

   while ((FindMaxOfMatrix(nodeStack))==1)
   {
       k=0;

       while ((nodeStack[k])==0)
       {
       k=k+1;
       }

       nodeStack[k]=0;                     //briskei to prwto mi
mideniko stoixeio kai to midenizei

       for(n=0;n<N;n++)
       {
         if ((matrix[k][n])==1)            //an yparxei geitonas
aytoy toy stoixeioy
         {
         if (nodeVisited[n]==0)
         {
               nodeVisited[n]=1;            //markaretai o n san
kombos poy episkeftikame
               nodeStack[n]=1;
             }
       }
       }
   }

   components=components+1;

   b=0;
   m=0;

   while (b<N-1)
   {
     b=b+1;
       if (nodeVisited[b]==0)
       {
         m=b;    //stin oysia briskei ton prwto kombo poy den
enwnetai me toys proigoymenoys
         b=N-1;  //kai an ton brei termatizei ayto to while kai
trexei apo tin arxi to ekswteriko while
       }
   }
 } //to ekswteriko while termatizetai otan exei ginei episkepsi se
oloys toys komboys

return components;
```

```
}

/*sinartisi pou ipologizei ti mesi aktina*/
float Tools::FindAverageRadius(float node[3][N],int n)
{
  int i;
  float sum=0;

  for(i=0;i<n;i++)
      sum=sum+node[2][i];

  sum=sum/n;
  return sum;
}

/*sinartisi pou ipologizei to meso arithmo geitonwn*/
float Tools::FindAverageNumberOfNeighbours(int matrix[N][N],int n)
{
  int i;
  float sum=0;

  for(i=0;i<n;i++)
    sum=sum+FindNumberOfNeighbours(matrix,i);
  sum=sum/n;
  return sum;
}

/*sinartisi FindVarianceOfAverageRadius pou ipologizei ti variance
tis aktinas*/
float Tools::FindVarianceOfAverageRadius(float node[3][N],int n)
{
  int i;
  float rm;
  float sum=0.0;
  float d;

  rm=FindAverageRadius(node,n);

  for(i=0;i<n;i++)
  {
        sum=sum+((node[2][i]-rm)*(node[2][i]-rm));
  }

  d=sqrt(sum/(n*(n-1)));

  return d;
}

/*sinartisi FindVarianceOfAverageNeighbours pou ipologizei ti
variance twn geitonwn*/
float Tools::FindVarianceOfAverageNeighbours(int matrix[N][N],int n)
{
  int i,xm;
  float sum=0.0;
  float d;

  xm=FindAverageNumberOfNeighbours(matrix,n);

  for(i=0;i<n;i++)
  {
```

```
        sum=sum+((FindNumberOfNeighbours(matrix,i)-
xm)*(FindNumberOfNeighbours(matrix,i)-xm));
  }

  d=sqrt(sum/(n*(n-1)));

  return d;
}

/*kyriws programma*/
void main()
{
 float x,y,r;
 float f[N];
 int i,j,components;
 int NumberOfEdgeNodes=0;

 float node[3][N];

 for ( i=0;i<3;i++)
   for (j=0;j<N;j++)
     node[i][j]=0;


 int matrix_neighbour[N][N];
 Tools myTool;

 for ( i=0;i<N;i++)
   for (j=0;j<N;j++)
     matrix_neighbour[i][j]=0;

 for(i=0;i<N;i++)
       f[i]=0.0;

 srand(time(NULL));

 for(i=0;i<N;i++)
 {

   x=((rand()%32767)/32767.0)*L;

   y=((rand()%32767)/32767.0)*L;

   node[0][i]=x;        //dinontai oi sintetagmenes
   node[1][i]=y;        //stoys komboys


  }


for(i=0;i<N;i++)
    node[2][i]=RMAX;    //dinetai arxika i timi tis megistis aktinas
se oloys toys komboys

 myTool.CreateMatrixOfNeighbourhood(matrix_neighbour,node,N);
//ipologismos twn geitonwn sto arxiko diktio


 for(i=0;i<N;i++)
   f[i]=myTool.FindConeAngles(matrix_neighbour,i,node); //eysresi
megistis gwnias kwnoy gia kathe kombo
```

```
 components=myTool.FindNumberOfComponents(matrix_neighbour);

 char filename1[80];
 char buffer1[1000];
 cout<<"File name 1:";
 cin>> filename1;

 ofstream fout1(filename1);             //anoigma arxeiou gia arxiko
diktio
   if(!fout1)                           //elegxos an anoikse swsta
   {
    cout<<"Unable to open"<<filename1<<"for appending\n";
   }

 fout1<<"components : "<<components<<"\n\n";

 fout1<<"id\tx\ty\tR\tneighbours\n";

 for(i=0;i<N;i++)
 {

fout1<<setprecision(4)<<i<<"\t"<<node[0][i]<<"\t"<<node[1][i]<<"\t"<<
RMAX<<"\t"<<myTool.FindNumberOfNeighbours(matrix_neighbour,i)<<"\n";
 }

 fout1<<setprecision(4)<<"\narxiki mesi aktina:
"<<myTool.FindAverageRadius(node,N)<<"\n";
 fout1<<setprecision(4)<<"arxikos mesos arithmos geitonwn:
"<<myTool.FindAverageNumberOfNeighbours(matrix_neighbour,N)<<"\n";
 fout1<<setprecision(4)<<"arxiki variance aktinas:
"<<myTool.FindVarianceOfAverageRadius(node,N)<<"\n";
 fout1<<setprecision(4)<<"arxiki variance geitonwn:
"<<myTool.FindVarianceOfAverageNeighbours(matrix_neighbour,N)<<"\n";
 fout1.close();                        //kleisimo prwtou arxeiou

 for(i=0;i<N;i++)
 {
  if (f[i]>=F)
  {
     cout<<"\nError-max radius with f["<<i<<"]>="<<F<<"\n";
       NumberOfEdgeNodes+=1;
  }
  else
  {
   do
   {
    r=node[2][i];
    myTool.CreateMatrixOfNeighbourhood(matrix_neighbour,node,N);
    cout<<"\n";

    f[i]=myTool.FindConeAngles(matrix_neighbour,i,node);

      r=r-STEP; //stadiaki meiwsi tis aktinas mexri na kataliksoyme
stin epithimiti

    node[2][i]=r;
   } while((f[i]<F)&&(r>0)); //kataligoyme se aktina otan broyme
megisti gwnia kwnoy >F i otan i aktina parei arnitiki timi
```

```
   node[2][i]=r+STEP; //ayksisi tis aktinas giati sto teleytaio bima
eixe meiwthei mia fora parapanw


   if (node[2][i]<D)
         node[2][i]=D;

   cout<<"\ni aktina tou kombou "<<i<<" einai "<<node[2][i];
  }
 }

 myTool.CreateMatrixOfNeighbourhood(matrix_neighbour,node,N);

 cout<<"\nNumber Of Edge Nodes: "<<NumberOfEdgeNodes;
 components=myTool.FindNumberOfComponents(matrix_neighbour);

 char filename2[80];
 char buffer2[1000];
 cout<<"\nFile name 2:";
 cin>> filename2;

 ofstream fout2(filename2);             //anoigma arxeiou gia teliko
diktio
   if(!fout2)                           //elegxos an anoikse swsta
   {
    cout<<"Unable to open"<<filename2<<"for appending\n";
   }

 fout2<<"components : "<<components<<"\n\n";

 fout2<<"id\tx\ty\tR\tneighbours\n";

 for(i=0;i<N;i++)
 {

fout2<<setprecision(4)<<i<<"\t"<<node[0][i]<<"\t"<<node[1][i]<<"\t"<<
node[2][i]<<"\t"<<myTool.FindNumberOfNeighbours(matrix_neighbour,i)<<
"\n";
 }

 fout2<<setprecision(4)<<"\nteliki mesi aktina:
"<<myTool.FindAverageRadius(node,N)<<"\n";
 fout2<<setprecision(4)<<"telikos mesos arithmos geitonwn:
"<<myTool.FindAverageNumberOfNeighbours(matrix_neighbour,N)<<"\n";
 fout2<<setprecision(4)<<"teliki variance aktinas:
"<<myTool.FindVarianceOfAverageRadius(node,N)<<"\n";
 fout2<<setprecision(4)<<"teliki variance geitonwn:
"<<myTool.FindVarianceOfAverageNeighbours(matrix_neighbour,N)<<"\n";
 fout2.close();                         //kleisimo deuterou arxeiou


}
```

## Αλγόριθμος V

Για λόγους οικονομίας χώρου παρατίθεται μόνο ο πηγαίος κώδικας που υλοποίησε τον αλγόριθμο V, με τη φάση δύο, για γωνία $a=120°$. Για γωνία 90° η υλοποίηση είναι ακριβώς η ίδια με αντικατάσταση της τιμής των 120° από τις 90° όπου συναντάται. Επίσης η υλοποίηση μόνο της φάσης ένα περιλαμβάνεται στον

παρακάτω κώδικα και είναι εμφανές πως μπορεί να χρησιμοποιηθεί σε ξεχωριστό αρχείο.

```
#include <iostream.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <fstream.h>
#include <iomanip.h>

#define N 100
#define RMIN 150
#define RMAX 300
#define L 1000
#define Q 2
#define STEP 5

/*klasi tools pou periexei tis genikes sinartiseis tou programmatos*/
class Tools
{
public:

  float Distance(float x1,float y1,float x2,float y2);
  void CreateMatrixOfNeighbourhood(int matrix[N][N],float
node[3][N],int n);

  void qsort_auxil(float a[],int lower,int upper);
  void qsort(float a[],int n);

  float FindConeAngles(int matrix[N][N],int i,float node[3][N]);
  int FindNumberOfNeighbours(int matrix[N][N],int i);
  int FindMaxOfMatrix(int matrix[N]);
  float FindMaxOfMatrixFloat(float matrix[N]);
  int FindMinOfMatrix(int matrix[N]);
  int FindNumberOfComponents(int matrix[N][N]);
  float FindAverageRadius(float node[3][N],int n);
  float FindAverageNumberOfNeighbours(int matrix[N][N],int n);
  float FindVarianceOfAverageRadius(float node[3][N],int n);
  float FindVarianceOfAverageNeighbours(int matrix[N][N],int n);

private:
   int matrix[N][N];
   float node[3][N];
};

/*sinartisi Distance pou ipologizei tin apostasi metaksi
dio simeiwn A(x1,y1) kai B(x2,y2)*/
float Tools::Distance(float x1,float y1,float x2,float y2)
{
 return sqrt(((x2-x1)*(x2-x1))+((y2-y1)*(y2-y1)));
}

/*sinartisi CreateMatrixOfNeighbourhood pou ipologizei
ton pinaka geitonias pairnontas ws parametrous ton arithmo
twn kombwn kai ton pinaka pou periexei tis sintetagmenes,
to id kai tin aktina tous*/
void Tools::CreateMatrixOfNeighbourhood(int matrix[N][N],float
node[3][N],int n)
{
  float x1,y1,x2,y2;
```

```
  float d,r;
  int i,j;

  for(i=0;i<N;i++)
   for(j=0;j<N;j++)
       matrix[i][j]=0;


  for (i=0;i<n;i++)
  {
   x1=node[0][i];
   y1=node[1][i];
   r=node[2][i];

   for (j=0;j<n;j++)           //diatrexontai oloi oi komboi kai
simfwna me tin aktina poy
   {                           //exei o ekastote kombos elegxetai gia
olous tous ipoloipous
         if(j!=i)
      {
       x2=node[0][j];
       y2=node[1][j];
       d=Distance(x1,y1,x2,y2);

       if ( d <= r )           //an i apostasi tous apo ton kombo
einai mikroteri tis aktinas
            matrix[i][j]=1;    //tou diladi an einai mesa stin
embeleia tou
         }                       //an nai topotheteitai to 1 sto
antistoixo stoixeio tou pinaka geitonias
        else matrix[i][j]=0;     //alliws to 0
     }
  }
}

/*sinartisi qsort_auxil pou xrisimopoiitai apo tin qsort
gia tin diataksi twn stoixeiwn enos pinaka se ayksousa seira*/
void Tools::qsort_auxil(float a[],int lower,int upper)
{
  if (lower<upper)
    {
     float x=a[(lower+upper)/2];
     int i,j;

     for (i=lower,j=upper;i<=j;i++,j--)
       {
          while (a[i]<x) i++;
          while (a[j]>x) j--;

          if (i<=j)
            {
           float temp=a[i];

           a[i]=a[j];
           a[j]=temp;
            }
       }
     qsort_auxil(a,lower,j);
     qsort_auxil(a,i,upper);
    }
}
```

```
void Tools::qsort(float a[],int n)
{
  qsort_auxil(a,0,n-1);
}


/*sinartisi FindConeAngles pou briskei tis cone angles enos kombou
kai epistrefei ti megisti*/
float Tools::FindConeAngles(int matrix[N][N],int i,float node[3][N])
 {
 int j,k,l;
 float xnew,ynew,biggest;
 float cosf,f,g;
 float matrix_angles[N];


 for(l=0;l<N;l++)
 {
   matrix_angles[l]=0.0;
 }


 k=0;

 for (j=0;j<N;j++)
   {

    if (matrix[i][j]==1)             //ean to stoixeio tou pinaka
geitonias einai 1
     {
       xnew=node[0][j]-node[0][i];   //metasximatismos twn
sintetagmenwn tou kathe geitonikou
       ynew=node[1][j]-node[1][i];   //kombou se sxesi me ton
kentriko pou orizetai arxi twn axonwn

       if((xnew==0.0)&&(ynew>0.0))   //ean o geitonas einai stin x=0
kai y>0
         {
            f=90.0;                     //tote i gwnia prokiptei 90
            matrix_angles[k]=f;
           }
       else
         {
            if ((xnew==0.0)&&(ynew<0.0)) //ean o geitonas einai stin
x=0 kai y<0
              {
             f=270.0;                     //tote i gwnia prokiptei 270
             matrix_angles[k]=f;
              }
            else
              {
             if ((ynew==0.0)&&(xnew>0.0)) //ean o geitonas einai stin
y=0 kai x>0
                 {
               f=0.0;                      //tote i gwnia prokiptei 0
               matrix_angles[k]=f;
               }
             else
               {
```

```
              if ((ynew==0.0)&&(xnew<0.0)) //ean o geitonas einai
stin y=0 kai x<0
               {
                  f=180.0;                   //tote i gwnia einai 180
                matrix_angles[k]=f;
                  }
              else                         //ean den anikei se mia
apo tis 4 eidikes periptwseis
              {
                  if
(((xnew>0.0)&&(ynew>0.0))||((xnew<0.0)&&(ynew>0.0))) //ean o kombos
brisketai sto 1 i sto 2 tetartimorio
                  {

cosf=(1*xnew+1*ynew)/((sqrt(2))*Distance(0.0,0.0,xnew,ynew));
                   f=acos(cosf)*180/3.141592654;

                     matrix_angles[k]=f;     //ipologizetai i gwnia
toy kathe geitona me to thetiko tmima tou orizontiou axona
                }
              else
                 {
                     if
(((xnew<0.0)&&(ynew<0.0))||((xnew>0.0)&&(ynew<0.0))) //ean einai sto
3 i sto 4
                   {

cosf=(1*xnew+1*ynew)/((sqrt(2))*Distance(0.0,0.0,xnew,ynew));
                          f=acos(cosf)*180/3.141592654;

                          matrix_angles[k]=360.0-f; //afairesi me to
360 dioti o tipos dinei panta ti mikri gwnia
                     }
                   }
                }
            }
         }
        }
   k=k+1;
       }
     }
 if (k==0)
   return -1.0; //ayto tha epistrefei mexri na megalwsei arketa i
aktina kai na brei geitona
 if (k==1)
   return 360.0;


 qsort(matrix_angles,k);   //diataxi tou pinaka me tis gwnies se
ayksoysa seira

 biggest=-12000.0;

 for(i=0;i<k-1;i++)
 {
   g=matrix_angles[i+1]-matrix_angles[i];  //afairesi kathe stoixeiou
apo to epomeno gia na prokipsoun telika oi cone angles

   if (g>biggest)
       biggest=g;
 }
```

```
 g=360.0-matrix_angles[k-1]+matrix_angles[0];//ipologismos tis
teleytaias gwnias me eidiko typo
                                            //dioti to apotelesma
den tha itan swsto me afairesi
 if (g>biggest)
   return g;

 return biggest;
}

/*sinartisi FindNumberOfNeighbours pou briskei ton arithmo
 twn geitonwn enos kombou*/
int Tools::FindNumberOfNeighbours(int matrix[N][N],int i)
{
  int j,k;

  k=0;

  for(j=0;j<N;j++)
  {
       if (matrix[i][j]==1)
             k=k+1;
  }

  return k;
}

/*sinartisi FindMaxOfMatrix pou briskei to megisto stoixeio enos
pinaka*/
int Tools::FindMaxOfMatrix(int matrix[N])
{
 int k,i;

 k=matrix[0];

 for(i=0;i<N;i++)
 {
       if (matrix[i]>k)
             k=matrix[i];
 }

 return k;
}

/*sinartisi FindMaxOfMatrixFloat pou briskei to megisto stoixeio enos
pinaka me float stoixeia*/
float Tools::FindMaxOfMatrixFloat(float matrix[N])
{
 int i;
 float k;

 k=matrix[0];

 for(i=0;i<N;i++)
 {
       if (matrix[i]>k)
             k=matrix[i];
 }

 return k;
```

```
}

/*sinartisi FindMinOfMatrix pou briskei to elaxisto stoixeio enos
pinaka*/
int Tools::FindMinOfMatrix(int matrix[N])
{
 int k,i;

 k=matrix[0];

 for(i=0;i<N;i++)
 {
       if (matrix[i]<k)
             k=matrix[i];
 }

 return k;
}

/*sinartisi FindNumberOfComponents pou briskei ton arithmo
 twn components enos diktioy*/
int Tools::FindNumberOfComponents(int matrix[N][N])
{
 int components;
 int k,m,n,b;
 int nodeVisited[N];
 int nodeStack[N];

 components=0;

 for(k=0;k<N;k++)
 {
   nodeVisited[k]=0;
   nodeStack[k]=0;
 }

 m=0;
 nodeVisited[m]=1;

 while ((FindMinOfMatrix(nodeVisited))==0) //oso iparxoun komboi poy
den toys exoume episkeftei
 {
   nodeVisited[m]=1;

   for(n=m+1;n<N;n++)
   {
        if ((matrix[m][n])==1)              //an o kombos n einai
geitonas toy m
        {
             nodeVisited[n]=1;                //markaretai o n ws
kombos poy ton episkeftikame
             nodeStack[n]=1;                  //markarw sti stoiba
ton n kanontas to stoixeio 1
        }
   }

   while ((FindMaxOfMatrix(nodeStack))==1)
   {
       k=0;

       while ((nodeStack[k])==0)
```

```
       {
       k=k+1;
       }

       nodeStack[k]=0;                     //briskei to prwto mi
mideniko stoixeio kai to midenizei

       for(n=0;n<N;n++)
       {
         if ((matrix[k][n])==1)            //an yparxei geitonas
aytoy toy stoixeioy
         {
         if (nodeVisited[n]==0)
         {
               nodeVisited[n]=1;            //markaretai o n san
kombos poy episkeftikame
               nodeStack[n]=1;
             }
       }
       }
   }

   components=components+1;

   b=0;
   m=0;

   while (b<N-1)
   {
     b=b+1;
       if (nodeVisited[b]==0)
       {
         m=b;    //stin oysia briskei ton prwto kombo poy den
enwnetai me toys proigoymenoys
         b=N-1;  //kai an ton brei termatizei ayto to while kai
trexei apo tin arxi to ekswteriko while
       }
   }
 } //to ekswteriko while termatizetai otan exei ginei episkepsi se
oloys toys komboys

return components;
}

/*sinartisi pou ipologizei ti mesi aktina*/
float Tools::FindAverageRadius(float node[3][N],int n)
{
  int i;
  float sum=0;

  for(i=0;i<n;i++)
      sum=sum+node[2][i];

  sum=sum/n;
  return sum;
}

/*sinartisi pou ipologizei to meso arithmo geitonwn*/
float Tools::FindAverageNumberOfNeighbours(int matrix[N][N],int n)
{
  int i;
```

```
  float sum=0;

  for(i=0;i<n;i++)
    sum=sum+FindNumberOfNeighbours(matrix,i);
  sum=sum/n;
  return sum;
}

/*sinartisi FindVarianceOfAverageRadius pou ipologizei ti variance
tis aktinas*/
float Tools::FindVarianceOfAverageRadius(float node[3][N],int n)
{
  int i;
  float rm;
  float sum=0.0;
  float d;

  rm=FindAverageRadius(node,n);

  for(i=0;i<n;i++)
  {
        sum=sum+((node[2][i]-rm)*(node[2][i]-rm));
  }

  d=sqrt(sum/(n*(n-1)));

  return d;
}

/*sinartisi FindVarianceOfAverageNeighbours pou ipologizei ti
variance twn geitonwn*/
float Tools::FindVarianceOfAverageNeighbours(int matrix[N][N],int n)
{
  int i,xm;
  float sum=0.0;
  float d;

  xm=FindAverageNumberOfNeighbours(matrix,n);

  for(i=0;i<n;i++)
  {
        sum=sum+((FindNumberOfNeighbours(matrix,i)-
xm)*(FindNumberOfNeighbours(matrix,i)-xm));
  }

  d=sqrt(sum/(n*(n-1)));

  return d;
}

/*kyriws programma*/
void main()
{
 float x,y,r,d,d1,d2,dmin,s;
 float f[N],one_matrix[N],increased_matrix[N];
 float sum=0.0;
 int i,j,w,components,t,l,q,m,k;
 int NumberOfEdgeNodes=0;
 float matrixd[N];
 float node[3][N];
 float dr[N];
```

```
 for ( i=0;i<3;i++)
   for (j=0;j<N;j++)
     node[i][j]=0;

 for(i=0;i<N;i++)
       matrixd[i]=0;

 for(i=0;i<N;i++)
       dr[i]=0.0;

 int matrix_neighbour[N][N];
 int matrix_neighbour_first[N][N];
 Tools myTool;

 for ( i=0;i<N;i++)
   for (j=0;j<N;j++)
     matrix_neighbour[i][j]=0;

for ( i=0;i<N;i++)
   for (j=0;j<N;j++)
     matrix_neighbour_first[i][j]=0;


 for(i=0;i<N;i++)
       f[i]=0.0;

 for(i=0;i<N;i++)
       one_matrix[i]=0.0;

 for(i=0;i<N;i++)
       increased_matrix[i]=0.0;

 srand(time(NULL));

 for(i=0;i<N;i++)
 {

   x=((rand()%32767)/32767.0)*L;

   y=((rand()%32767)/32767.0)*L;

   node[0][i]=x;        //dinontai oi sintetagmenes
   node[1][i]=y;        //stoys komboys


  }

for(i=0;i<N;i++)
    node[2][i]=RMAX;    //dinetai arxika i timi tis megistis aktinas
se oloys toys komboys

 myTool.CreateMatrixOfNeighbourhood(matrix_neighbour,node,N);
//ipologismos twn geitonwn sto arxiko diktio


 for(i=0;i<N;i++)
   f[i]=myTool.FindConeAngles(matrix_neighbour,i,node); //eysresi
megistis gwnias kwnoy gia kathe kombo

 components=myTool.FindNumberOfComponents(matrix_neighbour);
```

```
 char filename1[80];
 char buffer1[1000];
 cout<<"File name 1:";
 cin>> filename1;

 ofstream fout1(filename1);             //anoigma arxeiou gia arxiko
diktio
   if(!fout1)                           //elegxos an anoikse swsta
   {
    cout<<"Unable to open"<<filename1<<"for appending\n";
   }

 fout1<<"components : "<<components<<"\n\n";

 fout1<<"id\tx\ty\tR\tneighbours\n";

 for(i=0;i<N;i++)
 {

fout1<<setprecision(4)<<i<<"\t"<<node[0][i]<<"\t"<<node[1][i]<<"\t"<<
RMAX<<"\t"<<myTool.FindNumberOfNeighbours(matrix_neighbour,i)<<"\n";
 }

 fout1<<setprecision(4)<<"\narxiki mesi aktina:
"<<myTool.FindAverageRadius(node,N)<<"\n";
 fout1<<setprecision(4)<<"arxikos mesos arithmos geitonwn:
"<<myTool.FindAverageNumberOfNeighbours(matrix_neighbour,N)<<"\n";
 fout1<<setprecision(4)<<"arxiki variance aktinas:
"<<myTool.FindVarianceOfAverageRadius(node,N)<<"\n";
 fout1<<setprecision(4)<<"arxiki variance geitonwn:
"<<myTool.FindVarianceOfAverageNeighbours(matrix_neighbour,N)<<"\n";
 fout1.close();                        //kleisimo prwtou arxeiou


for(i=0;i<N;i++)
    node[2][i]=RMIN;


s=0.0;

for(i=0;i<N;i++)
{

   do
   {
    r=node[2][i];

    myTool.CreateMatrixOfNeighbourhood(matrix_neighbour,node,N);

    f[i]=myTool.FindConeAngles(matrix_neighbour,i,node);

    r=r+STEP; //stadiaki ayksisi tis aktinas mexri na broyme tin
epithimiti timi

    node[2][i]=r;
    } while(((f[i]>120.0)||(f[i]<0.0))&&(r<RMAX));

    if (r>=RMAX)
      {
```

```
            cout<<"\nno cone angles<120 "<<i<<"
"<<myTool.FindNumberOfNeighbours(matrix_neighbour,i);
        NumberOfEdgeNodes+=1;

      }
      else
      {
       node[2][i]=r-STEP;
      }

}

for(i=0;i<N;i++)
  for(j=0;j<N;j++)
         {
               if(i!=j)
               {
                     if(matrix_neighbour[i][j]==1)
                         matrix_neighbour[j][i]=1;
               }
         }


for (l=0;l<N;l++)
      for(j=0;j<N;j++)
            matrix_neighbour_first[l][j]=matrix_neighbour[l][j];

for(i=0;i<N;i++)
{
  if(node[2][i]!=RMAX)
  {
   for(j=0;j<N;j++)
         {
         if ((i!=j)&&(matrix_neighbour[i][j]==1))
             {

dr[j]=myTool.Distance(node[0][i],node[1][i],node[0][j],node[1][j]);
             }
          node[2][i]=myTool.FindMaxOfMatrixFloat(dr);//dinetai ston i
telika i aktina pou antistoixei stin apostasi
  }
  }                                                    //tou pio
apomakrismenou geitona tou
}

for(i=0;i<N;i++)
       dr[i]=0.0;

/*phasi 2  5ou algorithmou*/

for(i=0;i<N;i++)//gia kathe kombo
{
           for(j=0;j<N;j++)
             {
          if (matrix_neighbour[i][j]==1)//an o j einai geitonas tou
              {

one_matrix[j]=myTool.Distance(node[0][i],node[1][i],node[0][j],node[1
][j]);//topothetise ston one_matrix
              }
//tin apostasi tou i apo ton geitona tou
```

```
              else//an den einai geitones
              {
               one_matrix[j]=-1;//kane to antistoixo stoixeio ston
one_matrix -1
              }
             }

             for(t=0;t<N;t++)
                 increased_matrix[t]=one_matrix[t];   //antegrapse
ton one_matrix ston increased_matrix

             myTool.qsort(increased_matrix,N);//taksinomise ton
increased_matrix kata ayksousa seira

         for(j=0;j<N;j++)//dietrekse me ti seira ola ta stoixeia tou
increased_matrix apo to mikrotero sto megalitero
             {
                   if(increased_matrix[j]!=-1)//ean i timi tou
stoixeioy den einai -1,diladi ousiastika
                   {                          //ean prokeitai gia
geitona

                        for(q=0;q<N;q++)
                          {

if(increased_matrix[j]!=one_matrix[q])//ousiastika entopise to q
diladi gia
                           {;}
//poion geitona milame
                           else
                             break; //otan to entopiseis bges
                          }

                      w=q;

                      for(m=0;m<N;m++)
                        {
                         if((increased_matrix[m]!=-1)&&(m!=j))//an
prokeitai gia allo geitona diaforetiko apo ton proigoumeno
                         {
                           for(k=0;k<N;k++)
                          {

if(increased_matrix[m]!=one_matrix[k])//bres poios einai
                           {;}
                           else
                             break;
                          }

                           t=k;

                              if ((t!=i)&&(t!=w))
                              {

if((matrix_neighbour[t][w]==1)&&(matrix_neighbour[i][w]==1)&&(matrix_
neighbour[i][t]==1))
                                   {

d=myTool.Distance(node[0][i],node[1][i],node[0][t],node[1][t]);//ipol
ogise tin apostasi metaksi tou i kai tou t
```

```
d1=myTool.Distance(node[0][i],node[1][i],node[0][w],node[1][w]);//ipo
logise tin apostasi metaksi tou i kai tou w

d2=myTool.Distance(node[0][t],node[1][t],node[0][w],node[1][w]);//ipo
logise tin apostasi metaksi tou w kai tou t


                                    if(((d+d2)<=(Q*d1))&&(d<=d1))//an
isxuei i anisotita
                                    {

                                 matrixd[t]=d;//topothtise tin d sto
matrixd
                                    }
                                 }
                            }
                         }//stin epomeni epanalipsi tha eksetasei ton
amesws makrinotero geitona kai etsi
                         //ikanopoieitai i apaitisi na elegxoun kata
ayksousa apostasi apo ton kentriko
                       }

                       for(l=0;l<N;l++)
                           sum=sum+matrixd[l];

                       if (sum!=0.0)//an exei ikanopoiithei i
anisotita estw mia fora
                       {
                       //cout<<"mpike";
                  matrix_neighbour[i][w]=0;//afairese ton w apo
geitona tou i
                         matrix_neighbour[w][i]=0;//kai to antitheto

                  for(l=0;l<N;l++)
                         {
                             if (matrixd[l]==0)
                                    matrixd[l]=10000;//thetw poli
megali timi sta midenika stoixeia etsi wste meta na bgei to swsto
elaxisto
                         }

                    s=matrixd[0];//eyresi elaxistoy stoixeioy toy
matrixd

                  for(l=0;l<N;l++)
                         {
                     if(matrixd[l]<s)
                           s=matrixd[l];
                         }

                         dmin=s;
                  cout<<dmin;

                         for(l=0;l<N;l++)
                               {

if((matrixd[l]!=dmin)&&(matrixd[l]!=10000))//an gia to idio w
ikanopoiithike i anisotita gia perissotera t
                                 {
//kai den milame gia to t me elaxisti apostasi apo to i
```

```
                                        matrix_neighbour[i][l]=0;//bgale
apo geitones tou i ta ipoloipa
                                      matrix_neighbour[l][i]=0;//kai to
antitheto
                                       }
                                  }

                        for(l=0;l<N;l++)
                          matrixd[l]=0.0;//arxikopoiisi tou pinaka
matrix gia ton elegxo tou epomenou j;

                        sum=0;
                        }

                    }
             }

      cout<<"\ngeitones
2:("<<i<<")"<<myTool.FindNumberOfNeighbours(matrix_neighbour,i);

  if(node[2][i]!=RMAX)
  {
   for(j=0;j<N;j++)
        {
        if ((i!=j)&&(matrix_neighbour[i][j]==1))
            {

dr[j]=myTool.Distance(node[0][i],node[1][i],node[0][j],node[1][j]);
            }
         node[2][i]=myTool.FindMaxOfMatrixFloat(dr);//dinetai ston i
telika i aktina pou antistoixei stin apostasi
  }
  }

}


 cout<<"\nNumber Of Edge Nodes: "<<NumberOfEdgeNodes;
 components=myTool.FindNumberOfComponents(matrix_neighbour);

 char filename2[80];
 char buffer2[1000];
 cout<<"\nFile name 2:";
 cin>> filename2;

 ofstream fout2(filename2);             //anoigma arxeiou gia teliko
diktio
   if(!fout2)                           //elegxos an anoikse swsta
   {
    cout<<"Unable to open"<<filename2<<"for appending\n";
   }

 fout2<<"components : "<<components<<"\n\n";

 fout2<<"id\tx\ty\tR\tneighbours\n";

 for(i=0;i<N;i++)
 {

fout2<<setprecision(4)<<i<<"\t"<<node[0][i]<<"\t"<<node[1][i]<<"\t"<<
```

```
node[2][i]<<"\t"<<myTool.FindNumberOfNeighbours(matrix_neighbour,i)<<
"\n";
 }

 fout2<<setprecision(4)<<"\nteliki mesi aktina:
"<<myTool.FindAverageRadius(node,N)<<"\n";
 fout2<<setprecision(4)<<"telikos mesos arithmos geitonwn:
"<<myTool.FindAverageNumberOfNeighbours(matrix_neighbour,N)<<"\n";
 fout2<<setprecision(4)<<"teliki variance aktinas:
"<<myTool.FindVarianceOfAverageRadius(node,N)<<"\n";
 fout2<<setprecision(4)<<"teliki variance geitonwn:
"<<myTool.FindVarianceOfAverageNeighbours(matrix_neighbour,N)<<"\n";
 fout2.close();                          //kleisimo telikou arxeiou


}
```

## Αλγόριθμος RR

```
#include <iostream.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <fstream.h>
#include <iomanip.h>

#define N 100
#define RMAX 300
#define RMIN 0
#define L 1200

/*klasi tools pou periexei tis genikes sinartiseis tou programmatos*/
class Tools
{
public:

  float Distance(float x1,float y1,float x2,float y2);
  void CreateMatrixOfNeighbourhood(int matrix[N][N],float
node[4][N],int n);

  void qsort_auxil(float a[],int lower,int upper);
  void qsort(float a[],int n);

  int FindNumberOfNeighbours(int matrix[N][N],int i);
  int FindMaxOfMatrix(int matrix[N]);
  float FindMaxOfMatrixFloat(float matrix[N]);
  int FindMinOfMatrix(int matrix[N]);
  int FindNumberOfComponents(int matrix[N][N]);
  float FindAverageRadius(float node[4][N],int n);
  float FindAverageNumberOfNeighbours(int matrix[N][N],int n);
  float FindVarianceOfAverageRadius(float node[3][N],int n);
  float FindVarianceOfAverageNeighbours(int matrix[N][N],int n);
  int NBelongsToRM(int i,int n,int m,float node[4][N]);
  void FlipAllStatesDownChain(float node[4][N],int m,int i);
  int isLeft(float node[4][N],float MeetPoint[2][N],int i,int j,int
k);
  int wn_PnPoly(float node[4][N],int i,float MeetPoint[2][N],int n);
  float ComputeOfa(float x1,float y1,float x2,float y2);
  float ComputeOfb(float x1,float y1,float x2,float y2);
  float ComputeCommonX(float a1,float b1,float a2,float b2);
```

```
  float ComputeCommonY(float a1,float b1,float a2,float b2);

private:
   int matrix[N][N];
   float node[4][N];
   int RelayMatrix[N][N];
};

/*sinartisi Distance pou ipologizei tin apostasi metaksi
dio simeiwn A(x1,y1) kai B(x2,y2)*/
float Tools::Distance(float x1,float y1,float x2,float y2)
{
 return sqrt(((x2-x1)*(x2-x1))+((y2-y1)*(y2-y1)));
}

/*sinartisi CreateMatrixOfNeighbourhood pou ipologizei
ton pinaka geitonias pairnontas ws parametrous ton arithmo
twn kombwn kai ton pinaka pou periexei tis sintetagmenes,
to id kai tin aktina tous*/
void Tools::CreateMatrixOfNeighbourhood(int matrix[N][N],float
node[4][N],int n)
{
  float x1,y1,x2,y2;
  float d,r;
  int i,j;

  for(i=0;i<N;i++)
   for(j=0;j<N;j++)
       matrix[i][j]=0;


  for (i=0;i<n;i++)
  {
   x1=node[0][i];
   y1=node[1][i];
   r=node[2][i];

   for (j=0;j<n;j++)           //diatrexontai oloi oi komboi kai
simfwna me tin aktina poy
   {                           //exei o ekastote kombos elegxetai gia
olous tous ipoloipous
         if(j!=i)
      {
       x2=node[0][j];
       y2=node[1][j];
       d=Distance(x1,y1,x2,y2);

       if ( d <= r )           //an i apostasi tous apo ton kombo
einai mikroteri tis aktinas
            matrix[i][j]=1;    //tou diladi an einai mesa stin
embeleia tou
         }                       //an nai topotheteitai to 1 sto
antistoixo stoixeio tou pinaka geitonias
        else matrix[i][j]=0;     //alliws to 0
     }
  }
}

/*sinartisi qsort_auxil pou xrisimopoiitai apo tin qsort
gia tin diataksi twn stoixeiwn enos pinaka se ayksousa seira*/
void Tools::qsort_auxil(float a[],int lower,int upper)
```

```
{
  if (lower<upper)
    {
     float x=a[(lower+upper)/2];
     int i,j;

     for (i=lower,j=upper;i<=j;i++,j--)
       {
          while (a[i]<x) i++;
          while (a[j]>x) j--;

          if (i<=j)
            {
           float temp=a[i];

           a[i]=a[j];
           a[j]=temp;
            }
       }
     qsort_auxil(a,lower,j);
     qsort_auxil(a,i,upper);
    }
}

void Tools::qsort(float a[],int n)
{
  qsort_auxil(a,0,n-1);
}



/*sinartisi FindNumberOfNeighbours pou briskei ton arithmo
 twn geitonwn enos kombou*/
int Tools::FindNumberOfNeighbours(int matrix[N][N],int i)
{
  int j,k;

  k=0;

  for(j=0;j<N;j++)
  {
       if (matrix[i][j]==1)
             k=k+1;
  }

  return k;
}

/*sinartisi FindMaxOfMatrix pou briskei to megisto stoixeio enos
pinaka*/
int Tools::FindMaxOfMatrix(int matrix[N])
{
 int k,i;

 k=matrix[0];

 for(i=0;i<N;i++)
 {
       if (matrix[i]>k)
             k=matrix[i];
 }
```

```
 return k;
}

/*sinartisi FindMaxOfMatrixFloat pou briskei to megisto stoixeio enos
pinaka me float stoixeia*/
float Tools::FindMaxOfMatrixFloat(float matrix[N])
{
 int i;
 float k;

 k=matrix[0];

 for(i=0;i<N;i++)
 {
       if (matrix[i]>k)
             k=matrix[i];
 }

 return k;
}

/*sinartisi FindMinOfMatrix pou briskei to elaxisto stoixeio enos
pinaka*/
int Tools::FindMinOfMatrix(int matrix[N])
{
 int k,i;

 k=matrix[0];

 for(i=0;i<N;i++)
 {
       if (matrix[i]<k)
             k=matrix[i];
 }

 return k;
}

/*sinartisi FindNumberOfComponents pou briskei ton arithmo
 twn components enos diktioy*/
int Tools::FindNumberOfComponents(int matrix[N][N])
{
 int components;
 int k,m,n,b;
 int nodeVisited[N];
 int nodeStack[N];

 components=0;

 for(k=0;k<N;k++)
 {
   nodeVisited[k]=0;
   nodeStack[k]=0;
 }

 m=0;
 nodeVisited[m]=1;

 while ((FindMinOfMatrix(nodeVisited))==0) //oso iparxoun komboi poy
den toys exoume episkeftei
```

```
 {
   nodeVisited[m]=1;

   for(n=m+1;n<N;n++)
   {
        if ((matrix[m][n])==1)              //an o kombos n einai
geitonas toy m
        {
             nodeVisited[n]=1;                //markaretai o n ws
kombos poy ton episkeftikame
             nodeStack[n]=1;                  //markarw sti stoiba
ton n kanontas to stoixeio 1
        }
   }

   while ((FindMaxOfMatrix(nodeStack))==1)
   {
       k=0;

       while ((nodeStack[k])==0)
       {
       k=k+1;
       }

       nodeStack[k]=0;                     //briskei to prwto mi
mideniko stoixeio kai to midenizei

       for(n=0;n<N;n++)
       {
         if ((matrix[k][n])==1)            //an yparxei geitonas
aytoy toy stoixeioy
         {
         if (nodeVisited[n]==0)
         {
               nodeVisited[n]=1;            //markaretai o n san
kombos poy episkeftikame
               nodeStack[n]=1;
             }
       }
       }
   }

   components=components+1;

   b=0;
   m=0;

   while (b<N-1)
   {
     b=b+1;
       if (nodeVisited[b]==0)
       {
         m=b;    //stin oysia briskei ton prwto kombo poy den
enwnetai me toys proigoymenoys
         b=N-1;  //kai an ton brei termatizei ayto to while kai
trexei apo tin arxi to ekswteriko while
       }
   }
 } //to ekswteriko while termatizetai otan exei ginei episkepsi se
oloys toys komboys
```

```
return components;
}

/*sinartisi pou ipologizei ti mesi aktina*/
float Tools::FindAverageRadius(float node[4][N],int n)
{
  int i;
  float sum=0;

  for(i=0;i<n;i++)
      sum=sum+node[2][i];

  sum=sum/n;
  return sum;
}

/*sinartisi pou ipologizei to meso arithmo geitonwn*/
float Tools::FindAverageNumberOfNeighbours(int matrix[N][N],int n)
{
  int i;
  float sum=0.0;

  for(i=0;i<n;i++)
    sum=sum+FindNumberOfNeighbours(matrix,i);
  sum=sum/n;
  return sum;
}

/*sinartisi FindVarianceOfAverageRadius pou ipologizei ti variance
tis aktinas*/
float Tools::FindVarianceOfAverageRadius(float node[3][N],int n)
{
  int i;
  float rm;
  float sum=0.0;
  float d;

  rm=FindAverageRadius(node,n);

  for(i=0;i<n;i++)
  {
        sum=sum+((node[2][i]-rm)*(node[2][i]-rm));
  }

  d=sqrt(sum/(n*(n-1)));

  return d;
}

/*sinartisi FindVarianceOfAverageNeighbours pou ipologizei ti
variance twn geitonwn*/
float Tools::FindVarianceOfAverageNeighbours(int matrix[N][N],int n)
{
  int i,xm;
  float sum=0.0;
  float d;

  xm=FindAverageNumberOfNeighbours(matrix,n);

  for(i=0;i<n;i++)
  {
```

```
        sum=sum+((FindNumberOfNeighbours(matrix,i)-
xm)*(FindNumberOfNeighbours(matrix,i)-xm));
  }

  d=sqrt(sum/(n*(n-1)));

  return d;
}

/*sinartisi NBelongsToRM pou ipologizei an o kombos n anikei stin
Relay Region toy m*/
int Tools::NBelongsToRM(int i,int n,int m,float node[4][N])
{
      float dnm,din,dim;
      dnm=0.0;
      din=0.0;
      dim=0.0;

      dnm=Distance(node[0][n],node[1][n],node[0][m],node[1][m]);
      din=Distance(node[0][i],node[1][i],node[0][n],node[1][n]);
      dim=Distance(node[0][i],node[1][i],node[0][m],node[1][m]);

      if ((dnm<din)&&(din>dim))
            return 1;
      else
            return 0;
}

/*sinartisi FlipAllStatesDownChain poy ipologizei tin teliki
katastasi twn kombwn poy periballoyn ton kentriko*/
void Tools::FlipAllStatesDownChain(float node[4][N],int m,int i)
{

      int k=0;
      int alive=0,notRelay=0;

      if (node[3][m]==1)  //an o m einai zwntanos
      {
            node[3][m]=0;   //markare ton san nekro
            for(k=0;k<N;k++)
            {
                  if (NBelongsToRM(i,k,m,node)==1) //an o k anikei
stin relay region toy m
                        FlipAllStatesDownChain(node,k,i);//efarmose
pali ti sinartisi anadromika
        }
      }
      else
      {
            for(k=0;k<N;k++)
            {
                  if (node[3][k]==1) //an o k einai zwntanos
                  {
                        alive=alive+1; //metrise ton ws zwntano
                if (NBelongsToRM(i,m,k,node)==0) //an o m den anikei
stin relay region toy k
                              notRelay=notRelay+1;         //metrise
ton
            }
        }
```

```
            if (alive==notRelay) //an o m den anikei stin relay
region kanenos zwntanoy komboy
            {
                  node[3][m]=1;    //markare ton ws zwntano
            for(k=0;k<N;k++)
                  {
                  if (NBelongsToRM(i,k,m,node)==1) //an o k anikei
stin relay region toy m
                        FlipAllStatesDownChain(node,k,i);//efarmose
pali ti sinartisi anadromika
                  }
            }
      }
}

// Copyright 2001, softSurfer (www.softsurfer.com)
// This code may be freely used and modified for any purpose
// providing that this copyright notice is included with it.
// SoftSurfer makes no warranty for this code, and cannot be held
// liable for any real or imagined damage resulting from its use.
// Users of this code must verify correctness for their application.


/*sinartisi isLeft pou epistrefei:
   >0 an o kombos k einai aristera tis grammis poy sximatizoyn o i
kai j
   =0 an o kombos k einai panw sti grammi poy sximatizoyn o i kai j
   <0 an o kombos k einai deksia tis grammis poy sximatizoyn o i kai
j*/
int Tools::isLeft(float node[4][N],float MeetPoint[2][N],int i,int
j,int k)
{
    return ((MeetPoint[0][j]-MeetPoint[0][i]) * (node[1][k]-
MeetPoint[1][i])
            - (node[0][k]-MeetPoint[0][i]) * (MeetPoint[1][j]-
MeetPoint[1][i]));
}

/*sinartisi wn_PnPoly poy epistrefei ton arithmo twn forwn poy ena
kombos
  perikleietai apo ena poligwno
  An den perikleietai tote i sinartisi epistrefei 0*/
int Tools::wn_PnPoly(float node[4][N],int i,float MeetPoint[2][N],int
n)
{
    int    wn = 0;    // the winding number counter

    if (n<3) //an ta simeia tomis einai ligotera apo 3 den mporei na
sximatistei poligwno
            return -2;
      else
      {
       // diatrexw olo to poligwno me seira komnwn ti seira twn
stoixeiwn toy pinaka
     for (int k=0; k<n; k++) {
        if (MeetPoint[1][k] <= node[1][i]) {  //an i sintetagmeni y
toy i < tis y toy simeioy tomis
            if (MeetPoint[1][k+1] > node[1][i]) //tote diatrexw pros
ta panw
                if (isLeft(node,MeetPoint,k,k+1,i) > 0)  //an i
grammi anamesa sto k kai k+1 einai deksia toy i
```

```
                    ++wn;            //tote ayksanw ton aritmo forwn
poy o i perikleietai
        }
        else {                       //an i sintetagmeni y toy i >=
tis y toy simeioy tomis
            if (MeetPoint[1][k+1]<= node[1][i])    //tote diatrexw
pros ta katw
                if (isLeft(node,MeetPoint,k,k+1,i) < 0)  //an i
grammi anamesa sto k kai k+1 einai aristera toy i
                    --wn;            // tote meiwnw ton aritmo forwn
poy o i perikleietai
        }
     }
     return wn;
      }
}

/*sinartisi ComputeOfa poy ipologizei tin parametro a tis eytheias
y=ax+b*/
float Tools::ComputeOfa(float x1,float y1,float x2,float y2)
{
      return ((x1-x2)/(y2-y1));
}

/*sinartisi ComputeOfb poy ipologizei tin parametro b tis eytheias
y=ax+b*/
float Tools::ComputeOfb(float x1,float y1,float x2,float y2)
{
      return ((y2*y2+x2*x2-y1*y1-x1*x1)/(2*y2-2*y1));
}

/*sinartisi ComputeCommonX poy briskei ti sintetagmeni x toy simeioy
tomis dyo mesokathetwn*/
float Tools::ComputeCommonX(float a1,float b1,float a2,float b2)
{
      return((b2-b1)/(a1-a2));
}

/*sinartisi ComputeCommonX poy briskei ti sintetagmeni y toy simeioy
tomis dyo mesokathetwn*/
float Tools::ComputeCommonY(float a1,float b1,float a2,float b2)
{
      return((a2*(b2-b1)/(a1-a2))+b2);
}

/*kyriws programma*/
void main()
{
 float x,y,r;
 float d[N];
 int AlreadySearched[N];
 float LineChar[2][N];
 float MeetPoint[2][N];
 int NowSearching[N];
 int i,j,l,k,components,included;
 int nodessearch;
 int NumberOfEdgeNodes=0;
 float node[4][N];

 for ( i=0;i<4;i++)
   for (j=0;j<N;j++)
```

```
      node[i][j]=0;

 for ( i=0;i<2;i++)
   for (j=0;j<N;j++)
     LineChar[i][j]=0.0;

 for ( i=0;i<2;i++)
   for (j=0;j<N;j++)
     MeetPoint[i][j]=0.0;


 int matrix_neighbour[N][N];
 int RelayMatrix[N][N];

 Tools myTool;

 for ( i=0;i<N;i++)
   for (j=0;j<N;j++)
     matrix_neighbour[i][j]=0;

 for ( i=0;i<N;i++)
   for (j=0;j<N;j++)
     RelayMatrix[i][j]=0;


 for(i=0;i<N;i++)
       d[i]=0.0;

 for(i=0;i<N;i++)
       AlreadySearched[i]=0;

 for(i=0;i<N;i++)
       NowSearching[i]=0;


 srand(time(NULL));

 for(i=0;i<N;i++)
 {

   x=((rand()%32767)/32767.0)*L;

   y=((rand()%32767)/32767.0)*L;

   node[0][i]=x;  //topothetisi kombwn se sigkekrimeni thesi sto
epipedo
   node[1][i]=y;  //me kathorismo tis x kai y sintetagmenis tous


  }


 for(i=0;i<N;i++)
    node[2][i]=RMAX; //arxikos kathorismos tis aktinas ws megistis
gia arxiko diktio

 myTool.CreateMatrixOfNeighbourhood(matrix_neighbour,node,N);

 components=myTool.FindNumberOfComponents(matrix_neighbour);

 char filename1[80];
```

```
 char buffer1[1000];
 cout<<"File name 1:";
 cin>> filename1;

 ofstream fout1(filename1);             //anoigma arxeiou gia arxiko
diktio
   if(!fout1)                           //elegxos an anoikse swsta
   {
    cout<<"Unable to open"<<filename1<<"for appending\n";
   }

 fout1<<"components : "<<components<<"\n\n";

 fout1<<"id\tx\ty\tR\tneighbours\n";

 for(i=0;i<N;i++)
 {

fout1<<setprecision(4)<<i<<"\t"<<node[0][i]<<"\t"<<node[1][i]<<"\t"<<
RMAX<<"\t"<<myTool.FindNumberOfNeighbours(matrix_neighbour,i)<<"\n";
 }

 fout1<<setprecision(4)<<"\narxiki mesi aktina:
"<<myTool.FindAverageRadius(node,N)<<"\n";
 fout1<<setprecision(4)<<"arxikos mesos arithmos geitonwn:
"<<myTool.FindAverageNumberOfNeighbours(matrix_neighbour,N)<<"\n";
 fout1<<setprecision(4)<<"arxiki variance aktinas:
"<<myTool.FindVarianceOfAverageRadius(node,N)<<"\n";
 fout1<<setprecision(4)<<"arxiki variance geitonwn:
"<<myTool.FindVarianceOfAverageNeighbours(matrix_neighbour,N)<<"\n";
 fout1.close();                        //kleisimo prwtou arxeiou

 for ( i=0;i<N;i++)
   for (j=0;j<N;j++)
     matrix_neighbour[i][j]=0;

 for(i=0;i<N;i++) //gia kathe kombo
 {
       l=0;
       r=node[2][i];
         for(j=0;j<N;j++)
         {
               if((j!=i)&&(AlreadySearched[j]==0)) //gia kathe kombo
ektos toy kentrikoy kai efoson aytos den exei elegxthei
               {

if((myTool.Distance(node[0][i],node[1][i],node[0][j],node[1][j]))<=r)
//efoson brisketai mesa sto eyros tis aktinas
                     {
                   NowSearching[j]=1;         //markare ton ws
psaxnomeno
                        AlreadySearched[j]=1;      //markare ton kai
ws psaxthenta
                     }
               }
         }


        for(j=0;j<N;j++)
         {
```

```
                if (NowSearching[j]==1) //an o kombos einai
psaxnomenos
                {
                      node[3][j]=0;       //markare ton ws nekro
                      for(k=0;k<N;k++)
                      {
                            if(AlreadySearched[k]==1) //an o kombos
einai psaxthentas
                            {


if((myTool.NBelongsToRM(i,k,j,node))==1) //kai an o k anikei stin
relay region toy j
                                  {
                                       RelayMatrix[k][j]=1; //ftiakse
antistoixa to relay graph

                                  }
                                  else
if((myTool.NBelongsToRM(i,j,k,node))==1) //alliws an isxyei to
anapodo
                                  {
                                        RelayMatrix[j][k]=1; //ftiakse
antistoixa to relay graph

                                  }
                        }
                          }
                }
              }

            for(j=0;j<N;j++)
          {
                if (NowSearching[j]==1)
               myTool.FlipAllStatesDownChain(node,j,i);

              }

          for(j=0;j<N;j++)
              {
                if((AlreadySearched[j]==1)&&(node[3][j]==1)) //pairnw
enan enan toys komboys poy exw brei mexri twra kai ean einai zwntanoi
                {

LineChar[0][j]=myTool.ComputeOfa(node[0][i],node[1][i],node[0][j],nod
e[1][j]);//briskw toys sintelestes a kai b

LineChar[1][j]=myTool.ComputeOfb(node[0][i],node[1][i],node[0][j],nod
e[1][j]);//twn eksiswsewn twn mesokathetwn
                }
          }



          for (j=0;j<N;j++)
              {
                for(k=j+1;k<N;k++)
                {
                      if
((node[3][j]==1)&&(node[3][k]==1)&&(LineChar[0][j]!=LineChar[0][k]))/
/an yparxei antistoixi mesokathetos kai an i kliseis twn eytheiwn
```

```
                    {
//den einai ises

MeetPoint[0][l]=myTool.ComputeCommonX(LineChar[0][j],LineChar[1][j],L
ineChar[0][k],LineChar[1][k]);

      MeetPoint[1][l]=myTool.ComputeCommonY(LineChar[0][j],LineChar[1
][j],LineChar[0][k],LineChar[1][k]);
                l=l+1;
                     }
          }
         }
         MeetPoint[0][l]=MeetPoint[0][0]; //kanw to prwto stoixeio
toy pinaka iso me to
             MeetPoint[1][l]=MeetPoint[1][0]; //teleytaio gia na
treksei swsta i wn_PnPoly

           included=myTool.wn_PnPoly(node,i,MeetPoint,l);
      //ipologizw an dimioyrgithike kleisti kampyli gyrw apo ton i


             if (included==-1)
                   included=1;

           for(j=0;j<N;j++)
          NowSearching[j]=0;

      if (included!=1)
        {
            cout<<"\nno relay region";
        NumberOfEdgeNodes+=1;
            for(j=0;j<N;j++)
            {
         if (i!=j)
             {
              if ((AlreadySearched[j]==1)&&(node[3][j]==1))
              {
                   matrix_neighbour[i][j]=1; //eyresi telikwn
geitonwn edge kombou
              }
             }
            }
      }
     else
     {
      for(j=0;j<N;j++)
        {
         if (i!=j)
             {
              if ((AlreadySearched[j]==1)&&(node[3][j]==1))
              {
                   matrix_neighbour[i][j]=1; //eyresi telikwn
geitonwn kombou

d[j]=myTool.Distance(node[0][i],node[1][i],node[0][j],node[1][j]);
              }
             }
      }
      node[2][i]=myTool.FindMaxOfMatrixFloat(d);
     }
```

```
       /*arxikopoisi gia treksimo se neo kentriko kombo*/

     for ( j=0;j<2;j++)
      for (k=0;k<N;k++)
        LineChar[j][k]=0.0;

     for ( j=0;j<2;j++)
      for (k=0;k<N;k++)
        MeetPoint[j][k]=0.0;

       for(j=0;j<N;j++)
       AlreadySearched[j]=0;

     for (j=0;j<N;j++)
       node[3][j]=0;

 }

 cout<<"\nNumber Of Edge Nodes: "<<NumberOfEdgeNodes;
 components=myTool.FindNumberOfComponents(matrix_neighbour);



 char filename2[80];
 char buffer2[1000];
 cout<<"\nFile name 2:";
 cin>> filename2;

 ofstream fout2(filename2);             //anoigma arxeiou gia teliko
diktio
   if(!fout2)                           //elegxos an anoikse swsta
   {
    cout<<"Unable to open"<<filename2<<"for appending\n";
   }

 fout2<<"components : "<<components<<"\n\n";

 fout2<<"id\tx\ty\tR\tneighbours\n";

 for(i=0;i<N;i++)
 {

fout2<<setprecision(4)<<i<<"\t"<<node[0][i]<<"\t"<<node[1][i]<<"\t"<<
node[2][i]<<"\t"<<myTool.FindNumberOfNeighbours(matrix_neighbour,i)<<
"\n";
 }

 fout2<<setprecision(4)<<"\nteliki mesi aktina:
"<<myTool.FindAverageRadius(node,N)<<"\n";
 fout2<<setprecision(4)<<"telikos mesos arithmos geitonwn:
"<<myTool.FindAverageNumberOfNeighbours(matrix_neighbour,N)<<"\n";
 fout2<<setprecision(4)<<"teliki variance aktinas:
"<<myTool.FindVarianceOfAverageRadius(node,N)<<"\n";
 fout2<<setprecision(4)<<"teliki variance geitonwn:
"<<myTool.FindVarianceOfAverageNeighbours(matrix_neighbour,N)<<"\n";
 fout2.close();                         //kleisimo deuterou arxeiou
```

# Βιβλιογραφία


[1] Jiang and J. Bruck, "Monotone Percolation and The Topology Control of Wireless Networks", *In Proceedings of IEEE INFOCOM*, March 2005.

[2] R. Wattenhofer, P. Bahl, L. Li and Y. M. Wang, "Distributed Topology Control for Power Efficient Operation in Multihop Wireless Ad Hoc Networks", *In Proceedings of IEEE INFOCOM*, April 2001.

[3] V. Rodoplu and T. H. Meng, "Minimum Energy Mobile Wireless Networks", *IEEE Journal on Selected Areas in Communications*, vol. 17, no.8, pp. 1333-1344, August 1999.

[4] V. Kawadia and P. R. Kumar, "Power Control and Clustering in Ad hoc Networks", *In Proceedings of IEEE INFOCOM*, April 2003.

[5] R. Ramanathan and R. Rosales-Hain, "Topology Control of Multihop Wireless Networks using Transmit Power Adjustment", *In Proceedings of IEEE INFOCOM*, March 2000.

[6] B. Chen, K. Jamieson, H. Balakrishnan and R. Morris, "Span: An Energy-Efficient Coordination Algorithm for Topology Maintenance in Ad Hoc Wireless Networks", *ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom),* July 2001.

[7] L. Li, J. Y. Halpern, P. Bahl, Y. M. Wang and R. Wattenhofer, "A Cone-Based Distributed Topology-Control Algorithm for Wireless Multi-Hop Networks", *ACM Symposium on Principle of Distributed Computing (PODC),* 2001.

[8] Y. Xu, J. Heidemann and D. Estrin, "Geography-Informed Energy Conservation for Ad Hoc Routing", *Proc. Mobicom*, pp. 108-116, July 2001.

[9] www.softsurfer.com, Dan Sunday, "Fast Winding Number Inclusion of a Point in a Polygon".

[10] V. A. Karyotis, "A Base Case Study of the Worm Propagation Problem in the Framework of Topology Control in Wireless Ad-Hoc Networks. *Master's Thesis, School of Engineering & Applied Sciences, University of Pennsylvania, Philadelphia, PA,* USA, August 2005.

[11] Εθνικό Μετσόβιο Πολυτεχνείο, Τομέας Φυσικής, Γενικό τμήμα, "Εργαστηριακές ασκήσεις φυσικής - Τόμος Ι ", Εκδόσεις Συμμετρία, Αθήνα 1999.

[12] Jesse Liberty, "Εγχειρίδιο C++", Εκδόσεις Μ. Γκοιύρδας, Αθήνα 1999.

[13] www.antd.nist.gov, National Institute of Standards and Technology, "Wireless Ad Hoc Sensor Networks".

[14] www.netmode.ntua.gr, Συμεών Παπαβασιλείου, Διαφάνειες μεταπτυχιακού μαθήματος "Αυτοοργανούμενα (Ad-hoc) Δίκτυα ".