Ατομική Διπλωματική Εργασία

ΔΡΟΜΟΛΟΓΗΣΗ ΜΕ ΒΑΣΗ ΤΑ ΜΥΡΜΗΓΚΙΑ ΣΕ ΚΙΝΗΤΑ AD HOC ΔΙΚΤΥΑ

(ANT-BASED ROUTING IN MOBILE AD HOC NETWORKS)

Αντωνία Τερζή

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ιούνιος 2009

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Δρομολόγηση με βάση τα μυρμήγκια σε Κινητά Ad Hoc Δίκτυα

(Ant-based Routing in Mobile Ad Hoc Networks)

Αντωνία Τερζή

Επιβλέπουσα Καθηγήτρια

Άννα Φιλίππου

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Ιούνιος 2009

Ευχαριστίες

Θέλω στο μέρος αυτό να ευχαριστήσω την επιβλέπουσα καθηγήτριά μου Δρ. Άννα Φιλίππου, Επίκουρη Καθηγήτρια του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου, για τη σημαντικότατη συμβολή της στην εκπόνηση της ατομικής διπλωματικής εργασίας μου. Έχοντας τη γενική εποπτεία της εργασίας μου, η Δρ. Άννα Φιλίππου μού παρείχε την ορθή καθοδήγηση και στήριξη που χρειαζόταν ώστε να μπορέσω να ολοκληρώσω την ατομική διπλωματική εργασία με επιτυχία και στα χρονικά όρια που επιβάλλονταν. Σαν έμπειρη ερευνήτρια στη θεωρία και πρακτική των θεωρημάτων της Πληροφορικής και ειδικότερα των αλγόριθμων, μου παρείχε την απαιτούμενη βοήθεια για σωστή μελέτη και έρευνα με σκοπό την απόκτηση γνώσεων και εμπειριών, στοιχεία απαραίτητα για την ολοκλήρωση της Ατομικής Διπλωματικής Εργασίας. Επίσης, μέσω των συχνών συναντήσεων και συζητήσεων, κατάφερα να εξοικειωθώ με το θέμα και το περιβάλλον του εργαλείου που αποφασίσαμε να χρησιμοποιήσουμε για την υλοποίηση, έτσι ώστε να πετύχουμε όσο το δυνατόν καλύτερα τους στόχους που θέσαμε από την αρχή.

Περίληψη

Στη Ατομική Διπλωματική Εργασία ασχοληθήκαμε με τη δρομολόγηση σε κινητά ad hoc δίκτυα έχοντας ως βάση αλγόριθμους που εμπνεύστηκαν από τη φύση και ειδικότερα, τα μυρμήγκια.

Αρχικά, μελετήσαμε τον τομέα της *Νοημοσύνης Σμηνών* ώστε να έχουμε μια γενική ιδέα για τη συμπεριφορά των οργανωμένων ομάδων της φύσης, όπως είναι τα ψάρια, οι μέλισσες και τα μυρμήγκια. Ακολούθως, μελετήσαμε τον τομέα της δρομολόγησης σε κινητά ad hoc δίκτυα, εντοπίζοντας τους κυριότερους αλγόριθμους. Στη συνέχεια, ασχοληθήκαμε με αλγόριθμους από τη φύση και συγκεκριμένα αλγόριθμους που έχουν ως βάση τη μίμηση της συμπεριφοράς των μυρμηγκιών στην προσπάθεια αναζήτησης τροφής. Εντοπίσαμε δύο από τους καλύτερους αλγόριθμους που ικανοποιούσαν κάποια σημαντικά χαρακτηριστικά ενός δικτύου, όπως είναι η αποδοτικότητα και η ανθεκτικότητα. Οι αλγόριθμοι αυτοί είναι οι AODV και AntHocNet για τους οποίους έγινε μια βαθύτερη μελέτη με σκοπό την υλοποίηση.

Μετέπειτα, αποφασίσαμε πως θα υλοποιήσουμε μια εκδοχή των δύο αλγορίθμων που βασίζεται στη συμπεριφορά των μυρμηγκιών, ώστε να μπορέσουμε να τους συγκρίνουμε να παρατηρήσουμε τη συμπεριφορά τους, καθώς και να επιβεβαιώσουμε τις διαπιστώσεων της βιβλιογραφίας που μελετήσαμε. Η υλοποίηση των αλγορίθμων έγινε στο εργαλείο MASON το οποίο δίνει τη δυνατότητα γραφικής απεικόνισης της συμπεριφοράς πρακτόρων που ορίζονται στο πρόγραμμα γραμμένο σε γλώσσα προγραμματισμού Java.

Τέλος, ολοκληρώνοντας τις εργασίες που είχαμε θέσει ως στόχο να εκτελέσουμε, εξάγαμε κάποια γενικά συμπεράσματα που αφορούσαν τους δύο αλγόριθμους, τη συμπεριφορά τους και πώς αυτή είναι πιθανόν να διαμορφωθεί σε υλοποίηση σε κινητά ad hoc δίκτυα.

Περιεχόμενα

Εικόνες

Πίνακες

ΚΕΦΑΛΑΙΟ 1:

ΕΙΣΑΓΩΓΗ

1.1 Περιγραφή θέματος Ατομικής Διπλωματικής Εργασίας

Πολλοί αλγόριθμοι έχουν προταθεί κατά καιρούς για δρομολόγηση σε κινητά δίκτυα, αφού η τεχνολογία αυτή είναι συνεχώς αναπτυσσόμενη και χρησιμοποιείται στην πλειονότητα των σημερινών δικτύων, όπως είναι κυρίως το διαδίκτυο. Πολλοί από αυτούς τους αλγόριθμους έχουν εμπνευστεί από τη συμπεριφορά διάφορων ζωντανών ομάδων εντόμων, ζώων και ψαριών. Στην ατομική διπλωματική εργασία αυτή, ασχοληθήκαμε με τη μελέτη της ευφυΐας και της συμπεριφοράς ομάδων μυρμηγκιών που συναντούμε στη φύση. Η συμπεριφορά των μυρμηγκιών έχει μοντελοποιηθεί από διάφορους αλγόριθμους που αναφέρονται στη συνέχεια του εγγράφου αυτού και οι οποίοι στηρίζονται στις ενέργειες των μυρμηγκιών ως ανεξάρτητους πράκτορες, καθώς και στο σύνολο των μυρμηγκιών ως ομάδα. Οι αλγόριθμοι αυτοί που έχουν εμπνευστεί από τη φύση, εφαρμόζονται σήμερα σε πολλές περιπτώσεις σε δίκτυα, αφού εμφανίζουν πολλά καλά χαρακτηριστικά που χρησιμεύουν στη δρομολόγηση πακέτων σε ένα τομέα,

όπως είναι τα δίκτυα, που είναι αναπόσπαστο κομμάτι της σύγχρονης επικοινωνίας μεταξύ των ανθρώπων. Πιο κάτω, αναλύουμε τις εργασίες που έγιναν στο πλαίσιο της Ατομικής Διπλωματικής Εργασίας σε κάθε της φάση.

Αρχικά, επικεντρωθήκαμε στην κατανόηση του τομέα της νοημοσύνης σμηνών, όπως αυτή περιγράφει τη συμπεριφορά οργανωμένων ομάδων της φύσης. Η νοημοσύνη σμηνών εξηγεί με λεπτομέρεια τον τρόπο επικοινωνίας μεταξύ πρακτόρων που ανήκουν στην ίδια ομάδα, όπως είναι τα ψάρια, οι μέλισσες και τα μυρμήγκια. Μέσα από αυτή τη μελέτη, κατανοήσαμε σε μεγάλο βαθμό τον τρόπο με τον οποίο επικοινωνούν οι πράκτορες αυτοί και πώς η επικοινωνία τους αυτή τους εξυπηρετεί στη διαδικασία εύρεσης του φαγητού.

Ακολούθως, συγκεντρώσαμε διάφορα άρθρα που ασχολούνται με το θέμα της δρομολόγησης σε κινητά ad hoc δίκτυα, ανεξαρτήτως αλγορίθμων και μεθοδολογιών, καθώς και άρθρα που περιλαμβάνουν τους σημαντικότερους αλγόριθμους δρομολόγησης σε τέτοια δίκτυα. Αφού κατανοήσαμε τη γενική ιδέα του τομέα της δρομολόγησης, προχωρήσαμε ένα στάδιο παρακάτω, στη μελέτη άρθρων που ασχολούνται ειδικότερα με τη δρομολόγηση σε κινητά ad hoc δίκτυα και έχουν ως βάση αλγόριθμους που εμπνεύστηκαν από τη φύση. Έτσι, καταλήξαμε στη μελέτη εγγράφων και άρθρων που ασχολούνταν αποκλειστικά με πρωτόκολλα και μεθοδολογίες δρομολόγησης με βάση τη νοημοσύνη των μυρμηγκιών.

Μέσα από τη μελέτη διάφορων πηγών που ασχολούνται με το θέμα αυτό, έγινε κατανοητή η μεγάλη σημασία της μελέτης και κατανόησης της συμπεριφοράς τέτοιων ομάδων, αφού σε αυτή στηρίζονται οι υλοποιήσεις αλγορίθμων δρομολόγησης σε κινητά ad hoc δίκτυα. Οι αλγόριθμοι αυτοί θεωρούν πως κάθε κόμβος ενός δικτύου ξεχωριστά αποτελεί έναν πράκτορα της ομάδας εντόμων και το δίκτυο αντιμετωπίζεται ως η ομάδα των εντόμων που διέπεται από τους κανόνες επικοινωνίας μεταξύ των πρακτόρων. Επομένως, είναι δυνατή η προσομοίωση της συμπεριφοράς και των ενεργειών τέτοιων

ομάδων, ώστε να επιτευχθεί όσο το δυνατόν καλύτερη επικοινωνία και ανταλλαγή μηνυμάτων και πακέτων μεταξύ των κόμβων ενός δικτύου.

Μέσα από τη μελέτη των άρθρων έγινε περισσότερο εμφανής η αποτελεσματικότητα των ιδεών των αλγορίθμων που βασίζονται στη φιλοσοφία συμπεριφοράς ομάδων της φύσης, μιας και υλοποιούν λειτουργίες που εγγυώνται ορθή λειτουργία αφού είναι ήδη δοκιμασμένες από ζωντανούς οργανισμούς. Συγκεκριμένα, μελετήσαμε τα άρθρα που παρουσιάζουν τους τρόπους που χρησιμοποιούν τα μυρμήγκια στην καθημερινή τους προσπάθεια να εντοπίσουν τα συντομότερα μονοπάτια από τη φωλιά προς το φαγητό τους και ταυτόχρονα να μεταδώσουν αυτή τη γνώση και προς τα υπόλοιπα μέλη της ομάδας τους. Οι αλγόριθμοι αυτοί προτείνουν τη χρήση πινάκων φερομονών (pheromones) που είναι το έμμεσο μέσο επικοινωνίας μεταξύ των μυρμηγκιών, ώστε να επιτευχθεί στο τέλος η χρήση του καλύτερου μονοπατιού από τη φωλιά στο φαγητό από όλα τα μέλη της ομάδας. Οι φερομόνες είναι νοητά «σημάδια» και συγκεκριμένα *αμετάβλητοι υδρογονάνθρακες*, που αφήνουν ορισμένα μυρμήγκια στο έδαφος και έτσι επιτρέπουν στα υπόλοιπα μυρμήγκια να τα εντοπίσουν και ακολουθώντας τα να βρουν το καλύτερο μονοπάτι που έχει ήδη ανακαλυφθεί από ένα άλλο μέλος της ομάδας.

Με βάση τη λογική αυτή, μελετήσαμε διάφορους αλγόριθμους δρομολόγησης που βασίζουν τη λειτουργία τους στη συμπεριφορά των μυρμηγκιών κατά την αναζήτηση φαγητού στο χώρο τους. Έτσι, επιλέξαμε να μελετήσουμε σε βάθος δύο από τους αλγόριθμους αυτούς, λόγω της ευρείας αποδοχής της οποίας τυγχάνουν. Οι αλγόριθμοι αυτοί είναι οι AODV [21] και AntHocNet [7,8], ένας αντιδραστικός και ένας υβριδικός αλγόριθμος αντίστοιχα.

Των όσων προαναφέρθηκαν ακολούθησε η υλοποίηση μιας εκδοχής των δύο αλγορίθμων, AODV και AntHocNet. Η υλοποίηση αυτή αφορούσε την προσομοίωση της συμπεριφοράς των μυρμηγκιών που στηρίζονται στις αρχές που διέπουν τους δύο αλγόριθμους, αφού η αυθεντική υλοποίηση τους αφορά δρομολόγηση σε κινητά δίκτυα.

Η υλοποίηση που προαναφέρουμε έγινε στο εργαλείο MASON που αναπαριστά γραφικά τη συμπεριφορά και κίνηση των μυρμηγκιών στο χώρο τους, ξεκινώντας από τη φωλιά και προχωρώντας προς το φαγητό σε παράλληλη αναζήτηση του βέλτιστου μονοπατιού. Εκτενέστερη αναφορά στο εργαλείο και τα γνωρίσματά του γίνεται στο υποκεφάλαιο *2.5* του εγγράφου αυτού. Με αυτό τον τρόπο, καταφέραμε να παρατηρήσουμε τη συμπεριφορά και ακριβή κίνηση των μυρμηγκιών, πλέον σε γραφική μορφή, κατά την κίνησή τους σε ένα συγκεκριμένο χώρο.

1.2 Στόχοι Ατομικής Διπλωματικής Εργασίας

Στο μέρος αυτό παρουσιάζουμε τους στόχους της Ατομικής Διπλωματικής Εργασίας:

1. Η εξοικείωση με το πεδίο της νοημοσύνης σμηνών μέσα από τη μελέτη άρθρων και ερευνών που ασχολούνται με αυτή.
2. Η μελέτη της περιοχής των κινητών ad hoc δικτύων με έμφαση στο θέμα της δρομολόγησης στα δίκτυα αυτά.
3. Η εμβάθυνση σε θέμα που βρίσκονται στην τομή των πεδίων «Νοημοσύνη Σμηνών» και «Δρομολόγηση σε ad hoc δίκτυα» και συγκεκριμένα της δρομολόγησης σε κινητά ad hoc δίκτυα με βάση αλγόριθμους μυρμηγκιών.
4. Η εξαγωγή συμπερασμάτων αναφορικά με τα οφέλη που προσφέρει η χρήση δρομολόγησης που βασίζεται σε αλγόριθμους μυρμηγκιών στα κινητά ad hoc δίκτυα.
5. Η λεπτομερής μελέτη και αξιολόγηση δύο αλγορίθμων δρομολόγησης που εμπνεύστηκαν από την περιοχή της νοημοσύνης σμηνών και συγκεκριμένα των μυρμηγκιών, μέσω υλοποίησής τους σε εργαλείο που επιτρέπει: (i) τη γραφική απεικόνιση και προσομοίωση, (ii) τη διεξαγωγή πειραμάτων μεταβάλλοντας τις παραμέτρους των αλγορίθμων και (iii) τη λήψη γραφικών παραστάσεων με τα στατιστικά κάθε προσομοίωσης

1.3 Περιγραφή μεθοδολογίας που χρησιμοποιήθηκε

Αρχικά, μελετήσαμε τον τομέα της *Νοημοσύνης Σμηνών* ώστε να έχουμε μια γενική ιδέα για τη συμπεριφορά των οργανωμένων ομάδων της φύσης, όπως είναι τα ψάρια, οι μέλισσες και τα μυρμήγκια. Ακολούθως, μελετήσαμε τον τομέα της δρομολόγησης σε κινητά ad hoc δίκτυα, εντοπίζοντας τους κυριότερους αλγόριθμους. Στη συνέχεια, ασχοληθήκαμε με αλγόριθμους από τη φύση και συγκεκριμένα αλγόριθμους που έχουν ως βάση τη μίμηση της συμπεριφοράς των μυρμηγκιών στην προσπάθεια αναζήτησης τροφής. Εντοπίσαμε δύο από τους καλύτερους αλγόριθμους που ικανοποιούσαν κάποια σημαντικά χαρακτηριστικά ενός δικτύου, όπως είναι η αποδοτικότητα και η ανθεκτικότητα. Οι αλγόριθμοι αυτοί είναι οι AODV και AntHocNet για τους οποίους έγινε μια βαθύτερη μελέτη με σκοπό την υλοποίηση.

Μετέπειτα, αποφασίσαμε πως θα υλοποιήσουμε μια εκδοχή των δύο αλγορίθμων που βασίζεται στη συμπεριφορά των μυρμηγκιών, ώστε να μπορέσουμε να τους συγκρίνουμε να παρατηρήσουμε τη συμπεριφορά τους, καθώς και να επιβεβαιώσουμε τις διαπιστώσεων της βιβλιογραφίας που μελετήσαμε. Η υλοποίηση των αλγορίθμων έγινε στο εργαλείο MASON το οποίο δίνει τη δυνατότητα γραφικής απεικόνισης της συμπεριφοράς πρακτόρων που ορίζονται στο πρόγραμμα γραμμένο σε γλώσσα προγραμματισμού Java.

Τέλος, ολοκληρώνοντας τις εργασίες που είχαμε θέσει ως στόχο να εκτελέσουμε, εξάγαμε κάποια γενικά συμπεράσματα που αφορούσαν τους δύο αλγόριθμους, τη συμπεριφορά τους και πώς αυτή είναι πιθανόν να διαμορφωθεί σε υλοποίηση σε κινητά ad hoc δίκτυα.

1.4 Περιγραφή δομής του εγγράφου

Το υπόλοιπο έγγραφο αποτελείται από άλλα πέντε κεφάλαια. Το δεύτερο κεφάλαιο ασχολείται με την παράθεση των πληροφοριών της υφιστάμενης γνώσης. Δηλαδή, δίνεται λεπτομερής επεξήγηση των πεδίων *Νοημοσύνη Σμηνών* στο υποκεφάλαιο *2.1*, *Κινητά Ad hoc Δίκτυα* στο υποκεφάλαιο *2.2*, *Αλγόριθμοι Δρομολόγησης* στο υποκεφάλαιο *2.3, Αλγόριθμοι Μυρμηγκιών* στο υποκεφάλαιο *2.4* και *Εργαλείο Υλοποίησης* στο υποκεφάλαιο *2.5.* Στη συνέχεια, παρατίθεται το τρίτο κεφάλαιο που περιγράφει με μεγαλύτερη λεπτομέρεια τους αλγόριθμους με τους οποίους ασχοληθήκαμε, δηλαδή τον AODV και τον AntHocNet. Το υποκεφάλαιο *3.1* δίνει μια λεπτομερή περιγραφή κάθε φάσης των δύο αλγορίθμων και το υποκεφάλαιο *3.2* παρουσιάζει τον ψευδο-κώδικα μιας επιλεγμένης των δύο αλγορίθμων που έχει προσαρμοστεί ειδικά στη συμπεριφορά των μυρμηγκιών. Στο τέταρτο κεφάλαιο γίνεται μια αναλυτική περιγραφή των μερών της υλοποίησης που έγινε για τις ανάγκες της Ατομικής Διπλωματικής Εργασίας, ενώ στο πέμπτο κεφάλαιο παρουσιάζονται τα αποτελέσματα κάποιων προσομοιώσεων που έγιναν με τη βοήθεια του εργαλείου υλοποίησης. Τέλος, στο έκτο κεφάλαιο εξάγονται κάποια συμπεράσματα για την εργασία που έγινε και δίνονται κάποιες κατευθυντήριες γραμμές για μια πιθανή μελλοντική εργασία που αφορά το θέμα αυτό.

ΚΕΦΑΛΑΙΟ 2:

ΥΦΙΣΤΑΜΕΝΗ ΓΝΩΣΗ

2.1 Νοημοσύνη Σμηνών - Swarm Intelligence

2.1.1 Γενική περιγραφή του όρου

Ο όρος *νοημοσύνη σμηνών* ή *swarm intelligence* (SI) [1] υποδεικνύει μια τεχνική τεχνητής νοημοσύνης που βασίζεται στη μελέτη συγκεντρωτικών συμπεριφορών αποκεντρωμένων και οργανωμένων από μόνων τους συστημάτων. Ο όρος SI παρουσιάστηκε για πρώτη φορά από τους Beni & Wang το 1989, στα πλαίσια δημιουργίας συστημάτων κυψελοειδών ρομποτικών.

Τα συστήματα SI αποτελούνται συνήθως από ένα πληθυσμό απλών πρακτόρων που αλληλεπιδρούν τοπικά μεταξύ τους, καθώς και με το περιβάλλον τους. Στις πλείστες περιπτώσεις δεν υπάρχει δομή κεντρικού ελέγχου που να προστάζει τον τρόπο με τον

οποίο πρέπει να συμπεριφέρονται οι ανεξάρτητοι πράκτορες, αλλά αντίθετα παρατηρείται τοπική επικοινωνία μεταξύ των πρακτόρων που συχνά οδηγεί στην εμφάνιση μιας καθολικής συμπεριφοράς. Παραδείγματα τέτοιων συστημάτων υπάρχουν στη φύση και συμπεριλαμβάνουν μεταξύ άλλων και τις αποικίες μυρμηγκιών, συγκεντρώσεις πουλιών, σμήνη μελισσών, βοσκή ζώων, σχηματισμό βακτηριδίων και εκπαίδευση ψαριών. Ακολουθεί μια περίληψη του τομέα της νοημοσύνης σμηνών και των χαρακτηριστικών του.

Η λογική της νοημοσύνης σμηνών εμφανίζεται σε βιολογικά σμήνη ορισμένων ειδών εντόμων και δίνει βάση σε πολύπλοκες και συχνά ευφυείς συμπεριφορές που παρατηρούνται λόγω της πολύπλοκης αλληλεπίδρασης μεταξύ χιλιάδων αυτόνομων μελών ενός σμήνους. Η αλληλεπίδραση βασίζεται σε πρωτόγονα ένστικτα και χαρακτηρίζεται από απουσία επίβλεψης. Το αποτέλεσμα αυτών των συμπεριφορών είναι η επίτευξη πολύ σύνθετων κοινωνικών συμπεριφορών και εκπλήρωση των στόχων σε προβλήματα βελτιστοποίησης [26].

2.1.2 Περιγραφή σημαντικών όρων και λειτουργιών του SI

Τα μυρμήγκια, οι τερμίτες και οι μέλισσες, ανάμεσα σε πολλά άλλα είδη της φύσης, είναι γνωστά για την πολύπλοκη κοινωνική δομή που κατέχουν. Η συμπεριφορά των σμηνών είναι μια από τις πολλές προκύπτουσες ιδιότητες των αποικιών που αποτελούνται από τα *κοινωνικά έντομα*. Ένα χαρακτηριστικό που παρουσιάζεται συνεχώς και παρατηρείται σε πολλές περιπτώσεις μελέτης των εντόμων είναι το *stigmergy* [24].

Ο όρος stigmergy εξηγεί την *έμμεση επικοινωνία* που φαίνεται να υποστηρίζει τη συνεργασία μεταξύ των κοινωνικών εντόμων. Το stigmergy προϋποθέτει την επικοινωνία μέσω «σημαδιών» που τοποθετούνται στο περιβάλλον από μια οντότητα, η οποία επηρεάζει τη συμπεριφορά των άλλων οντοτήτων που τα εντοπίζουν. Ένα παράδειγμα της χρήσης stigmergy αποτελεί η περίπτωση της απόθεσης φερομονών. Τα μυρμήγκια

αφήνουν φερομόνες καθώς προχωρούν στο μονοπάτι τους, βοηθώντας το μυρμήγκι που θα τις εντοπίσει να τις ακολουθήσει, αφού σχηματίζουν ένα δοκιμασμένο μονοπάτι.

Ένας χρήσιμος τρόπος για να κατανοήσουμε το stigmergy είναι να σκεφτόμαστε την επικοινωνία που βασίζεται σε αυτό σαν μια δομή, η οποία επηρεάζει τη συμπεριφορά των ανεξάρτητων οντοτήτων/πρακτόρων που τη «διαβάζουν». Οι οντότητες αυτές συνήθως προσθέτουν τις δικές τους πληροφορίες στη δομή αυτή. Τα στρατιωτικά μυρμήγκια βρίσκουν τις κατευθύνσεις που χρειάζονται για να ταξιδέψουν με βάση τα μονοπάτια φερομονών, αλλά προσθέτουν και αυτά τις δικές τους φερομόνες σε αυτά τα μονοπάτια. Η συμπεριφορά των τερμιτών προκαλείται από ορισμένα μοτίβα τα οποία βλέπουν τοπικά στο μερικώς χτισμένο ανάχωμα και ενεργούν με απλούς και συγκεκριμένους τρόπους ως αποτέλεσμα, προσθέτοντας τις πληροφορίες τους στην ίδια τη δομή [24]. Η παράγραφος που ακολουθεί περιγράφει την τυπική κίνηση των μυρμηγκιών στο χώρο στην προσπάθεια εύρεσης της τροφής. Η διαδικασία αυτή είναι παρόμοια για όλα τα είδη κοινωνικών εντόμων, αφού υπακούουν στους ίδιους κανόνες και στηρίζονται στην ίδια φιλοσοφία.

Η περιήγηση στο χώρο με σκοπό την εύρεση του φαγητού φαίνεται να εξαρτάται από την απόθεση φερομονών από τα ανεξάρτητα μυρμήγκια. Στο φυσικό περιβάλλον, η αρχική συμπεριφορά των μυρμηγκιών κατά την εξερεύνηση για το φαγητό είναι η τυχαία κίνηση των ανεξάρτητων πρακτόρων. Όταν ένα μυρμήγκι εντοπίσει το φαγητό, τότε επιστρέφει στη φωλιά του, ενώ καθόλη τη διάρκεια της εξερεύνησης τα μυρμήγκια άφηναν φερομόνες στο έδαφος, δημιουργώντας μονοπάτια φερομονών. Επομένως, τα μυρμήγκια που θα ξεκινήσουν την εξερεύνηση αργότερα, θα χρησιμοποιήσουν τις φερομόνες που υπάρχουν στο έδαφος για να φτάσουν με επιτυχία στο φαγητό. Με το πέρασμα του χρόνου φυσικά οι φερομόνες εξατμίζονται, αλλά παρόλα αυτά το γεγονός αυτό δεν επηρεάζει σημαντικά τα μυρμήγκια, κυρίως όταν η φωλιά και το φαγητό βρίσκονται σε κοντινή απόσταση.

Η νοημοσύνη σμηνών παρουσιάζει μια σειρά πλεονεκτημάτων λόγω της χρήσης κινητών πρακτόρων και της έννοιας του stigmergy. Τα πλεονεκτήματα αυτά είναι:

1. Ικανότητα εξέλιξης: Το σύνολο των πρακτόρων μπορεί να προσαρμοστεί ανάλογα με το μέγεθος του δικτύου. Η ικανότητα αυτή ενός σμήνους υπάρχει λόγω και των τοπικών και κατανεμημένων αλληλεπιδράσεων μεταξύ των πρακτόρων.
2. Ανοχή σφαλμάτων: Οι διαδικασίες που γίνονται σε ένα σμήνος δε βασίζονται σε ένα κεντρικό μηχανισμό ελέγχου. Έτσι, η απώλεια κάποιων κόμβων ή συνδέσεων δε συνεπάγεται μια καταστροφική αποτυχία, αλλά αντίθετα υποβαθμίζει τις απώλειες αυτές.
3. Προσαρμοστικότητα: Οι πράκτορες μπορούν να αλλάξουν, να πεθάνουν ή να αναπαραχθούν σύμφωνα με τις αλλαγές του δικτύου.
4. Ταχύτητα: Οι αλλαγές στο δίκτυο μπορούν να προωθηθούν πολύ γρήγορα, σε αντίθεση με τον αλγόριθμο Bellman-Ford.
5. Ικανότητα διαμόρφωσης: Οι πράκτορες ενεργούν αυτόνομα και ανεξάρτητα από τα άλλα επίπεδα του δικτύου.
6. Αυτονομία: Απαιτείται λίγη ή καθόλου ανθρώπινη επίβλεψη.
7. Παραλληλισμός: Οι ενέργειες των πρακτόρων είναι εγγενώς παράλληλες.

Οι ιδιότητες αυτές καθιστούν τη νοημοσύνη σμηνών πολύ ελκυστική για ad hoc δίκτυα [2]. Θεωρείται επίσης ότι η νοημοσύνη σμηνών είναι κατάλληλη για μια σειρά άλλων εφαρμογών, όπως η ρομποτική και βελτιστοποίηση που αναφέραμε πιο πάνω.

2.1.3 Νοημοσύνη σμηνών και δρομολόγηση σε δίκτυα

Η λογική της νοημοσύνης σμηνών χρησιμοποιείται σε μεγάλο βαθμό στα κινητά ad hoc δίκτυα. Αυτό γιατί, η νοημοσύνη σμηνών κατέχει πολλές δοκιμασμένες από τις ομάδες εντόμων λύσεις που χρησιμεύουν πολύ στη δρομολόγηση πακέτων στα δίκτυα. Συγκεκριμένα, αρκετοί αλγόριθμοι που έχουν υλοποιηθεί σε κινητά ad hoc δίκτυα με σκοπό την υποβοήθηση της δρομολόγησης, έχουν εμπνευστεί από τη συμπεριφορά

ομάδων σμηνών που εξηγούνται από τον όρο της νοημοσύνης σμηνών. Έτσι και εμείς, αποφασίσαμε να μελετήσουμε τόσο τη νοημοσύνη σμηνών όσο και τη δρομολόγηση στα δίκτυα μελετώντας κάποιους από τους αλγόριθμους που αναπτύχθηκαν και συνδυάζουν τους δύο αυτούς τομείς.

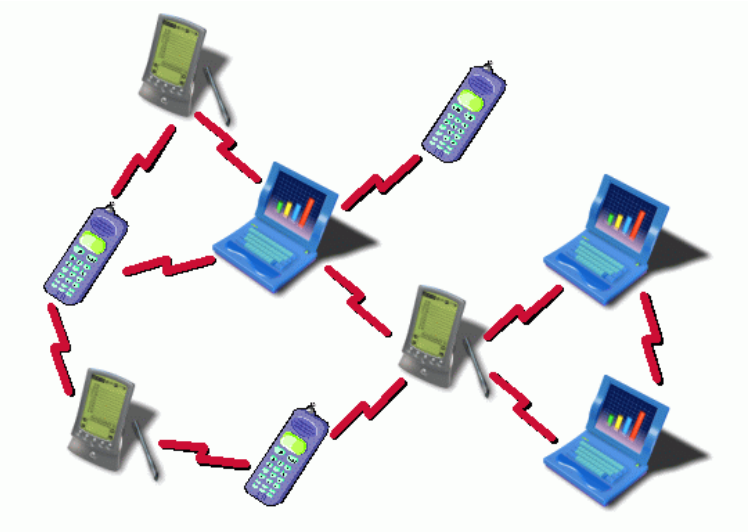
2.2 Κινητά Ad hoc Δίκτυα - Mobile Ad hoc Networks (MANETs)

Η συνεχώς αναπτυσσόμενη τεχνολογία των κινητών ad hoc δικτύων διευκολύνει την παροχή αλληλεπίδρασης και ηλεκτρονικής πρόσβασης σε υπηρεσίες, από χρήστες που διαθέτουν πρόσβαση εκτός του πλαισίου των κλασικών συνδεδεμένων δικτύων. Τα κινητά ad hoc δίκτυα έχουν σαν κύριο χαρακτηριστικό την παροχή πρόσβασης σε πληροφορίες, ανεξαρτήτως γεωγραφικών και ιδιοτήτων τοπολογίας ενός κόμβου. Σημειώνεται ότι τα δίκτυα MANET μπορούν να λειτουργούν τόσο αυτόνομα, όσο και συνδεδεμένα με το διαδίκτυο.

Ένα κινητό ad hoc δίκτυο (MANET) είναι ένας τύπος ad hoc δικτύου και αποτελεί ένα δίκτυο κινητών συσκευών που διαμορφώνεται από μόνο του, ανάλογα με την κίνηση των συσκευών αυτών. Κάθε τέτοια συσκευή σε ένα δίκτυο MANET αποτελεί ένα δρομολογητή, λόγω της απαίτησης για προώθηση της κίνησης που δεν αφορά μόνο τον ίδιο το δρομολογητή, αλλά και τους άλλους δρομολογητές του δικτύου. Κάθε συσκευή είναι ελεύθερη να κινηθεί οπουδήποτε, ανεξάρτητα και αυθαίρετα και επομένως κάθε συσκευή μπορεί ενδεχομένως να αλλάξει τις συνδέσεις της με άλλες συσκευές σε τακτά χρονικά διαστήματα. Η μεγάλη πρόκληση για δημιουργία ενός MANET είναι η διατήρηση των πληροφοριών που απαιτούνται για ορθή δρομολόγηση, εκ μέρους κάθε συσκευής [2].

Ο λατινικός όρος «ad hoc» υποδηλώνει τα συνεχώς τροποποιήσιμα χαρακτηριστικά ενός δικτύου MANET, δηλώνοντας ότι είναι σχεδιασμένο και αφοσιωμένο για να δέχεται τις απρόσμενες αλλαγές της τοπολογίας του δικτύου. Τα δίκτυα MANET αποτελούν τη

βασική υποδομή σε διάφορα συστήματα, κυρίως προσανατολισμένα σε στρατιωτικό και διαφημιστικό περιβάλλον. Οι ιδιότητες αυτό-διαμόρφωσης και κατοχής αυθαίρετης τοπολογίας του δικτύου ικανοποιούν τις απαιτήσεις τέτοιων συστημάτων, όπου είναι απαραίτητη η ανταλλαγή και η επεξεργασία δεδομένων πραγματικού χρόνου χωρίς να υπάρχει μέριμνα για γεωγραφικές αλλαγές στην τοπολογία του δικτύου. Παρόλο που τα δίκτυα MANET θεωρούνται δίκτυα με ανθεκτική και εξελικτική υποδομή, ιδανική για τέτοια συστήματα, αναμφισβήτητα εγείρουν σοβαρές ανησυχίες σε διάφορα πεδία, όπως η ασφάλεια, διαθεσιμότητα και αξιοπιστία. Τα δίκτυα MANET είναι σίγουρα ένα κρίσιμο ερευνητικό θέμα και απαιτεί μια τελείως διαφορετική προσέγγιση όσον αφορά την ανάλυση, αφού αυτή που ήδη υπάρχει για τα συνδεδεμένα δίκτυα δεν φαίνεται να είναι αρκετή.



Εικόνα 2.1: Παράδειγμα κινητού ad hoc δικτύου με σταθερές και κινητές συσκευές [9]

Όπως φαίνεται στην εικόνα *2.1,* ένα κινητό ad hoc δίκτυα μπορεί να αποτελείται από πολλές σταθερές και κινητές συσκευές, οι οποίες επικοινωνούν ασύρματα ή ενσύρματα. Όταν κάποια συσκευή σε ένα τέτοιο δίκτυο βρίσκεται εκτός του πεδίου προώθησης πληροφοριών, κάποιες άλλες συσκευές που βρίσκονται εντός του πεδίου αναλαμβάνουν να προωθήσουν τα μηνύματα προς αυτή. Η δυσκολία στο συντονισμό όλων αυτών των κόμβων έγκειται στη διατήρηση των πληροφοριών του δικτύου σε περιπτώσεις αλλαγών

στην τοπολογία του δικτύου όταν ένας ή περισσότεροι κόμβοι αποφασίσουν να κινηθούν. Σε αυτές τις περιπτώσεις, το δίκτυο πρέπει να κατέχει κάποιο μηχανισμό που να ανανεώνει τις πληροφορίες σύμφωνα με τις αλλαγές της τοποθεσίας του δικτύου.

Όπως όλα τα δίκτυα, τα δίκτυα MANET τυγχάνουν διαχείρισης και γίνονται λειτουργικά με τη χρήση πρωτοκόλλων δρομολόγησης (routing protocols). Ένα πρωτόκολλο δρομολόγησης ορίζεται ως ένα σύνολο κανόνων που είναι υπεύθυνοι για ανακάλυψη και καθιέρωση των πιο αξιόπιστων και σύντομων μονοπατιών μεταξύ των κόμβων. Ταυτόχρονα, ένα πρωτόκολλο δρομολόγησης είναι υπεύθυνο για διατήρηση και επισκευή (όπου χρειάζεται) οποιουδήποτε μονοπατιού. Τα πρωτόκολλα δρομολόγησης χωρίζονται σε τρεις κύριες κατηγορίες: *αντιδραστικά (reactive), δυναμικά (proactive) και υβριδικά (hybrid).*

Δυναμικό είναι ένα πρωτόκολλο που οδηγείται από τεχνικές που επικεντρώνονται κυρίως στη συντήρηση και ανανέωση των πληροφοριών στους πίνακες που διαχειρίζονται την κίνηση και την ορθότητα στις κατευθύνσεις των μονοπατιών. Παραδείγματα τέτοιων πρωτοκόλλων αποτελούν οι αλγόριθμοι DSDV [22], WRP [27], FSR [14] και OLSR [23]. Λεπτομερής περιγραφή των πρωτοκόλλων αυτών υπάρχει στο υποκεφάλαιο *2.3.3* του εγγράφου αυτού.

Τα *αντιδραστικά* πρωτόκολλα χρησιμοποιούν κάποιο αντιδραστικό αλγόριθμο και βρίσκουν μια διαδρομή κατόπιν απαίτησης, «πλημμυρίζοντας» το δίκτυο με πακέτα αιτημάτων για διαδρομές. Ωστόσο, η τεχνική αυτή έχει κάποια μειονεκτήματα σε θέματα καθυστέρησης στην ανακάλυψη διαδρομών και απόφραξης του δικτύου από τη συμφόρηση που προκαλείται λόγω της «πλημμύρας». Η πλημμύρα σε ένα δίκτυο αναφέρεται στο φαινόμενο όπου ένας δρομολογητής στέλνει ένα πακέτο από οποιοδήποτε κόμβο σε κάθε άλλο κόμβο του δικτύου που είναι συνδεδεμένος με αυτόν, με σκοπό τη διατήρηση των ανανεώσιμων πληροφοριών σε μεγάλα δίκτυα. Κάποιοι

αλγόριθμοι που χρησιμοποιούν την τεχνική αυτή είναι οι DSR [11], TORA [25] και AODV [21]. Τα πρωτόκολλα αυτά αναλύονται στο υποκεφάλαιο *2.3.2* του εγγράφου αυτού.

Ένα πρωτόκολλο αναφέρεται ως *υβριδικό* αν επιτρέπει τη συνεργασία μεταξύ αντιδραστικών και δυναμικών στοιχείων, ανεξαρτήτως του βασικού πρωτοκόλλου που αυτά χρησιμοποιούν. Τέτοια πρωτόκολλα είναι μεταξύ άλλων το AntHocNet [8] που στηρίζεται στον ACO αλγόριθμο [10], το ZRP [4] και το DDR [13]. Αναφερόμαστε εκτενώς στα πρωτόκολλα αυτά στο υποκεφάλαιο *2.3.4* του εγγράφου αυτού.

2.2.1 Χαρακτηριστικά κινητών ad hoc δικτύων

Λόγω της απρόβλεπτης τοπολογίας τους, τα δίκτυα MANET [18,20] συνθέτουν προκλήσεις σε διάφορα πεδία, όπου όλα έχουν άμεση σχέση με την απόδοση και τη γενική ποιότητα της υπηρεσίας (Quality of Service - QoS) που προσφέρεται στα δίκτυα αυτά. Τα δίκτυα MANET δεν προωθούν ένα κεντρικό μέσο ελέγχου για συντονισμό της κίνησης, όπως συμβαίνει στα συνδεδεμένα δίκτυα. Επίσης, οι κατανεμημένοι κόμβοι σε ένα δίκτυο MANET βασίζονται σε πηγές ενέργειας όπως οι μπαταρίες, αφού δεν κατέχουν μια σταθερή πηγή παροχής ενέργειας. Εξυπακούεται λοιπόν πως δημιουργείται ένα πρόβλημα όσον αφορά την κατανάλωση ενέργειας, το οποίο επηρεάζει τα μετρήσιμα μεγέθη της ποιότητας υπηρεσίας, όπως είναι η διαθεσιμότητα, βιωσιμότητα και αποδοτικότητα. Παράλληλα, συνθέτοντας όλα τα παραπάνω χαρακτηριστικά του δικτύου, εγείρεται ένα κρίσιμο θέμα που χρήζει αντιμετώπισης, το θέμα της ασφάλειας που αποτελεί μεγάλο θέμα επιστημονικών ερευνών μιας και η προστασία των δεδομένων και πληροφοριών ενός δικτύου είναι σημαντικό ζήτημα για ένα τέτοιο δίκτυο.

2.3 Αλγόριθμοι Δρομολόγησης σε κινητά ad hoc δίκτυα (MANETs)

2.3.1 Εισαγωγή

Η δρομολόγηση στα δίκτυα MANET αρχικά προσέγγιζε τη συμπεριφορά των συνδεδεμένων δικτύων, αλλά με την πάροδο του χρόνου υπήρξαν διάφορες άλλες προσεγγίσεις για τη δρομολόγηση σε κινητά ad hoc δίκτυα. Η πρώτη προσέγγιση αφορούσε την ανάπτυξη και χρήση αλγορίθμων που χρησιμοποιούσαν πίνακες και αποτελούσαν τις πρώτες τεχνικές *δυναμικής* δρομολόγησης, όπως η τεχνική DSDV [22]. Ακόμα και αν οι αλγόριθμοι αυτοί είναι απλοποιημένοι, η απρόσμενη και εκτεταμένη τοπολογία των δικτύων MANET δημιουργεί προβλήματα και θέματα που επηρεάζουν το δίκτυο αρνητικά. Η ανάγκη για εξέλιξη, η ανθεκτικότητα και η βελτιστοποίηση στη δρομολόγηση αποτελούν δύσκολους στόχους για επίτευξη και λόγω αυτού του γεγονότος οι ερευνητές μετά από μελέτες δημιούργησαν μια νέα ομάδα τεχνικών δρομολόγησης που ακολουθούν τις αρχές της δρομολόγησης μετά από απαίτηση, τις λεγόμενες *αντιδραστικές* τεχνικές δρομολόγησης. Εντούτοις, τα MANET είναι δίκτυα με ψηλές προσδοκίες σε ποιότητα υπηρεσίας και με σκοπό την εκπλήρωση των προσδοκιών αυτών, δημιουργήθηκε μια νέα ομάδα *υβριδικών* τεχνικών δρομολόγησης.

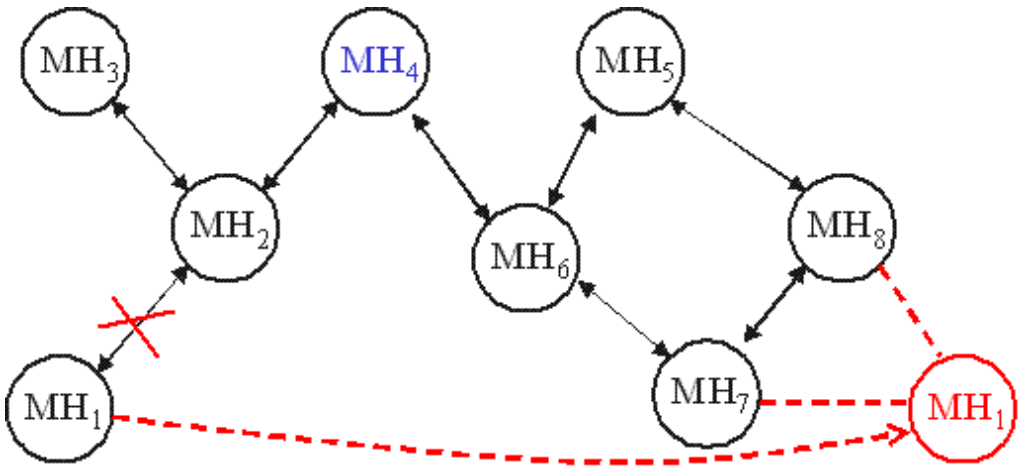
2.3.2 Δυναμικά Πρωτόκολλα (Proactive Protocols)

Η δυναμική δρομολόγησης υλοποιεί την ιδέα της υποδομής που οδηγείται από πίνακες, όπου κάθε κόμβος διατηρεί διάφορους πίνακες πάντα σύμφωνα με τις προδιαγραφές του δυναμικού αλγόριθμου. Ο προορισμός τοποθετείται στο δίκτυο σύμφωνα με τις περιοδικές ενημερώσεις που γίνονται σε κάθε κόμβο. Οι ενημερώσεις που γίνονται με σκοπό να αναβαθμιστούν οι πληροφορίες δρομολόγησης στους πίνακες σε κάθε κόμβο, εξαρτώνται από τις αλλαγές στην τοπολογία του δικτύου. Οι πληροφορίες δρομολόγησης τις πλείστες φορές αποθηκεύονται και διατηρούνται σε διαφορετικούς πίνακες, αλλά αυτό πιθανόν να διαφέρει από αλγόριθμο σε αλγόριθμο. Η κύρια διαφορά

μεταξύ διάφορων τέτοιων αλγορίθμων είναι ο τρόπος με τον οποίο γίνονται οι ενημερώσεις των πινάκων κάθε κόμβου. Στη συνέχεια, παρουσιάζουμε ορισμένα από τα πιο σημαντικά και ευρέως χρησιμοποιημένα πρωτόκολλα δρομολόγησης.

2.3.2.1. Πρωτόκολλο Destination-Sequenced Distance Vector (DSDV)

Το πρωτόκολλο DSDV [6,22] εκτελείται με βάση τις αρχές του κλασικού αλγορίθμου δρομολόγησης Bellman-Ford. Κάθε κόμβος διατηρεί έναν πίνακα δρομολόγησης και απαριθμεί όλους τους διαθέσιμους προορισμούς με χρήση των πληροφοριών που κρατά στους πίνακές του. Κάθε καταχώρηση στους πίνακες αυτούς χρησιμοποιεί ένα αύξοντα αριθμό, ο οποίος ταυτίζεται με τον προορισμό. Ακόμα, μια καταχώρηση περιέχει πληροφορίες για τη διεύθυνση και τον αριθμό των κόμβων που χρειάζεται να διαπεραστούν με στόχο την επιτυχή μεταφορά του πακέτου στον προορισμό. Παράλληλα, κάθε κόμβος στέλνει τις πληροφορίες που έχει σε όλους τους κόμβους του δικτύου (ή σε ορισμένες περιπτώσεις μόνο στους γείτονές του), χρησιμοποιώντας ένα πακέτο που περιέχει τον αύξοντα αριθμό του κόμβου που στέλνει το πακέτο, ώστε αυτός να αναγνωρίζεται από τους κόμβους που λαμβάνουν το πακέτο.



Εικόνα 2.2: Τοπολογία δικτύου με αποτυχία σύνδεσης (DSDV) [22]

Καθώς η τοπολογία του δικτύου αλλάζει, οι πληροφορίες δρομολόγησης που είναι αποθηκευμένες στους πίνακες κάθε κόμβου ανανεώνονται με ένα περιοδικό τρόπο, μοιράζοντας τις πληροφορίες αυτές στους υπόλοιπους κόμβους ανά καθορισμένες χρονικές περιόδους, ώστε να διατηρείται η συνέπεια. Όπως φαίνεται και στην εικόνα *2.2,* κάθε κόμβος μπορεί να μετακινηθεί όποτε θέλει σε μια άλλη θέση του δικτύου. Η μετακίνηση του αυτή πιθανό να προκαλέσει διαγραφή ορισμένων συνδέσεων με άλλους κόμβους και τη δημιουργία συνδέσεων με άλλους κόμβους του δικτύου. Η ανανέωση στους πίνακες δρομολόγησης είναι απαραίτητη κυρίως σε τέτοιες περιπτώσεις, αφού πρέπει κάθε κόμβος του δικτύου να γνωρίζει αν υπήρξε κάποια αλλαγή στην τοπολογία του δικτύου μιας και μπορεί να τον αφορά, είτε άμεσα είτε έμμεσα.

Ο DSDV θεωρείται αποδοτικός σε τοπολογίες μικρών δικτύων, αλλά μη αποδοτικός σε μετρήσεις μεγαλύτερων δικτύων που αφορούν εξέλιξη και ανθεκτικότητα σε αυτά. Σε τέτοια μεγαλύτερα δίκτυα παρατηρούνται διάφορα μειονεκτήματα, όπως η μεγάλη καθυστέρηση και υπερφόρτωση του δικτύου με μηνύματα ανανέωσης πληροφοριών, τα οποία ως επί το πλείστον εμφανίζονται λόγω των παραμέτρων που το πρωτόκολλο αναγκάζει τον κάθε κόμβο του δικτύου να γνωρίζει.

2.3.2.2. Πρωτόκολλο Wireless Routing (WRP)

Το πρωτόκολλο WRP [27] αποτελεί μια από τις πιο κύριες εφαρμογές που χρησιμοποιούν πίνακες δρομολόγησης στα δίκτυα MANET και βασίζεται στον αλγόριθμο εύρεσης μονοπατιού. Όπως και στο DSDV, ο κύριος στόχος του WRP είναι η συντήρηση των πληροφοριών δρομολόγησης που διατηρούν όλοι οι κόμβοι που συμμετέχουν στο δίκτυο. Το πρωτόκολλο αυτό ασχολείται περισσότερο με την ορθότητα των πληροφοριών δρομολόγησης που διανέμονται μεταξύ γειτονικών κόμβων και για το λόγο αυτό διαφέρει από άλλα δυναμικά πρωτόκολλα, όπως το DSDV.

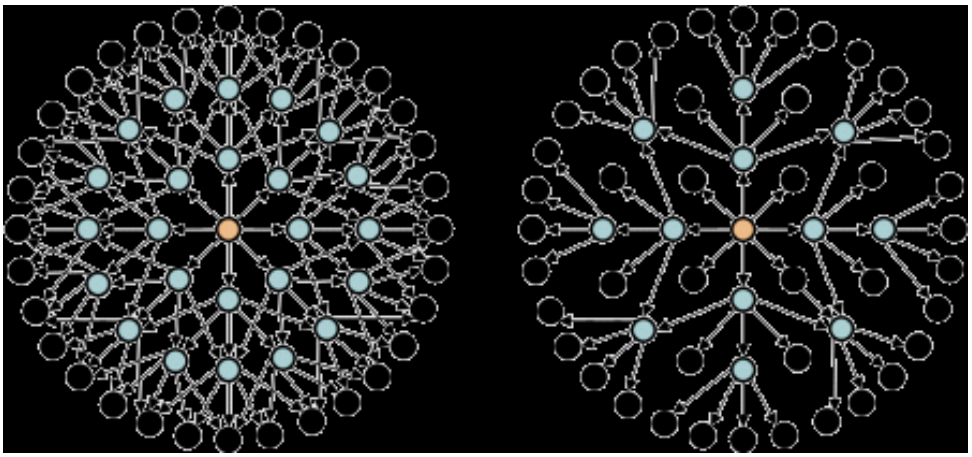
Σε αντίθεση με το DSDV, το WRP εισάγει τη χρήση τεσσάρων διαφορετικών πινάκων σε κάθε κόμβο και η κύρια λειτουργία του επιτυγχάνεται μέσω της χρήσης μηνυμάτων "hello" μεταξύ των κόμβων και τις παραλαβές μηνυμάτων επιβεβαίωσης. Ένα μήνυμα "hello" παρέχει τη δυνατότητα εξασφάλισης της συνδεσιμότητας ενός κόμβου του δικτύου σε περιπτώσεις που η αποστολή μηνυμάτων ενημέρωσης διακοπεί για κάποιο χρονικό διάστημα. Οι κόμβοι μαθαίνουν για την ύπαρξη άλλων κόμβων με τη χρήση αυτής της μεθόδου επιβεβαίωσης.

Όπως προαναφέρθηκε, ένας κόμβος στο WRP συντηρεί τις πληροφορίες τεσσάρων πινάκων: του *πίνακα αποστάσεων*, του *πίνακα δρομολόγησης*, του *πίνακα με το κόστος κάθε σύνδεσης* και του *πίνακα με τη λίστα μηνυμάτων που ξαναστέλνονται* (message retransmission list - MRL). Ο τελευταίος αυτός πίνακας είναι πολύ σημαντικός, αφού στην πραγματικότητα έχει να κάνει με τη συντήρηση της λίστας μηνυμάτων. Σε συντομία, ο MRL είναι ένας πίνακας που αποθηκεύει ένα μήνυμα ενημέρωσης και αποφασίζει ποια ενημέρωση θα πρέπει να μεταδοθεί και ποιοι γείτονες πρέπει να γνωρίζουν για τη μετάδοση αυτή. Ένα από τα βασικότερα πλεονεκτήματα του πρωτοκόλλου WRP είναι η αποφυγή του προβλήματος *καταμέτρησης προς το άπειρο (count-to-infinity)*, μιας και αναγκάζει κάθε κόμβο να ελέγχει με συνέπεια τις πληροφορίες του μονοπατιού του. Το γνώρισμα αυτό βελτιώνει τη *σύγκλιση δρομολόγησης* στην περίπτωση αποτυχίας σύνδεσης (link failure).

Σημείωση: Το πρόβλημα της *καταμέτρησης προς το άπειρο* αναφέρεται στις περιπτώσεις δικτύων που όταν παρουσιαστεί αποτυχία σύνδεσης, δεν ενημερώνονται οι πίνακες δρομολόγησης όλων των κόμβων που δε συνδέονται άμεσα με τον κόμβο που έχει χαθεί λόγω της αποτυχίας. Λόγω της μη ενημέρωσης, ορισμένοι κόμβοι αγνοούν τη μη ύπαρξη του κόμβου που χάθηκε και εξακολουθούν να στέλνουν πακέτα προς αυτόν με αποτέλεσμα τα πακέτα αυτά να μην έχουν τελικό προορισμό, ταξιδεύοντας επ' άπειρον στο δίκτυο.

2.3.2.3. Πρωτόκολλο Optimized Link State Routing (OSLR)

Το πρωτόκολλο OLSR [23] θεωρείται μια δυναμική εφαρμογή που βασίζεται στον αλγόριθμο συνδέσεων-καταστάσεων (link-state) και ασχολείται με την ενημέρωση και τη διατήρηση της τοπολογίας του δικτύου σε κάθε κόμβο με τη χρήση των πινάκων. Ακόμα, το OLSR επιτρέπει την επικοινωνία και ανταλλαγή πληροφοριών δρομολόγησης και τοπολογίας, μέσω περιοδικών μηνυμάτων συνδέσεων-καταστάσεων που δημιουργούνται σε κάθε κόμβο. Το πρωτόκολλο OLSR σε αντίθεση με τα προηγούμενα πρωτόκολλα που περιγράφηκαν, εισάγει την έννοια μιας στρατηγικής που επιτυγχάνει τη μείωση του ποσού καθυστέρησης στα μεταδιδόμενα μηνύματα ελέγχου. Κάθε κόμβος σε δίκτυο που χρησιμοποιεί το OLSR επιλέγει ένα σύνολο γειτονικών κόμβων και αποστέλλει τα πακέτα του.



Εικόνα 2.3: Αποστολή μηνυμάτων στο πρωτόκολλο OLSR [23]

Η βασική διαδικασία επιλογής του συνόλου των γειτονικών κόμβων βασίζεται και πάλι στη χρήση περιοδικών μηνυμάτων “hello”, όπως φαίνεται και στην εικόνα *2.3 (δεξιά)*, τα οποία αποστέλλονται από τον κόμβο που επιθυμεί να γνωρίσει τους γείτονες του. Πρέπει επίσης να σημειώσουμε ότι η διαδικασία επιλογής του συνόλου των γειτονικών κόμβων δεν είναι η συνηθισμένη πλημμύρα του δικτύου με μηνύματα, όπως φαίνεται στην εικόνα *2.3 (αριστερά).* Τα μηνύματα που στέλνονται περιλαμβάνουν μια λίστα των

άμεσων γειτόνων του τρέχοντος κόμβου και οι κόμβοι που λαμβάνουν τη λίστα αυτή επιλέγουν ένα υποσύνολο της που καλύπτει τους άμεσους γείτονές τους, καθώς και τους γείτονες των γειτόνων τους. Κάθε κινητός κόμβος, καθορίζει τη βέλτιστη διαδρομή προς τους γνωστούς προορισμούς. Κάθε κόμβος του δικτύου που γνωρίζει την τοπολογία και τις πληροφορίες των πινάκων των γειτόνων του, καθορίζει τη βέλτιστη διαδρομή του πίνακα δρομολόγησης, δίνοντας το πλεονέκτημα της κατοχής διαδρομών προς κάθε επιθυμητό προορισμό.

2.3.2.4. Περίληψη Χαρακτηριστικών Δυναμικών Πρωτοκόλλων

Η δυναμική δρομολόγηση παρουσιάζει μια βασική προσέγγιση για δρομολόγηση, αλλά από την άλλη δεν είναι μια τεχνική που εκπληρώνει τις προσδοκίες των κινητών ad hoc δικτύων που απαιτούν ποιότητα υπηρεσίας. Τα πρωτόκολλα που έχουν βάση την τεχνική αυτή προκαλούν ένα μεγάλο ποσοστό καθυστέρησης κατά την ανανέωση των μεταδιδόμενων μηνυμάτων, γεγονός το οποίο στα μεγάλα δίκτυα με πολλούς κόμβους συνεπάγεται καθυστέρηση και σε κάποιες περιπτώσεις αποτυχία στη μετάδοση των μηνυμάτων. Συμπληρώνοντας τα προηγούμενα, οι διαδικασίες ανανέωσης, καθώς και γενικά η «βαριά» υλοποίηση των δυναμικών πρωτοκόλλων καταναλώνουν μεγάλο ποσοστό του εύρους ζώνης (bandwidth) του δικτύου. Σε κατανεμημένα περιβάλλοντα με ετερογενή δίκτυα και κόμβους, το γεγονός αυτό οδηγεί σε προβλήματα κατανάλωσης δύναμης και ενέργειας σε κάθε κόμβο.

2.3.3 Αντιδραστικά Πρωτόκολλα (Reactive Protocols)

Λόγω του προβλήματος καθυστέρησης στις διαδικασίες ενημέρωσης που παρατηρείται στα δυναμικά πρωτόκολλα, ήταν απαραίτητο να εισαχθούν νέες αρχιτεκτονικές δρομολόγησης που να μειώνουν την καθυστέρηση αυτή. Σε αντίθεση με τη δυναμική, η αντιδραστική δρομολόγηση ασχολείται μόνο με τις ενεργές διαδρομές και κάθε κόμβος συντηρεί μόνο τις πληροφορίες δρομολόγησης των ενεργών προορισμών. Επίσης, η

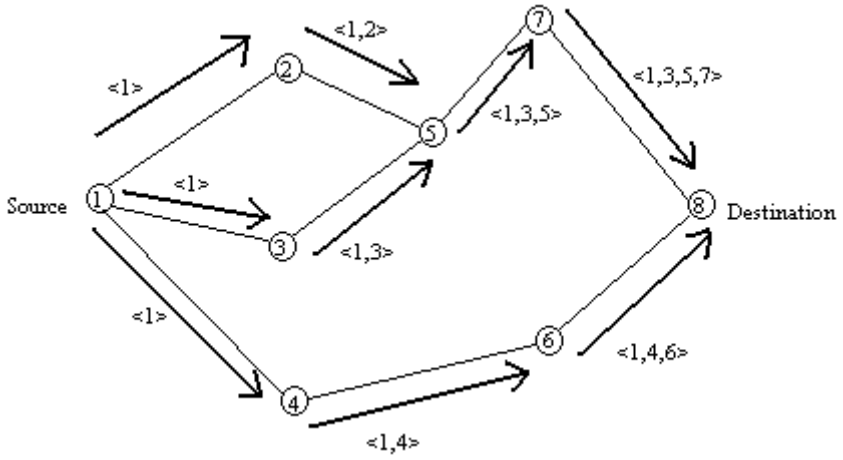
καινοτομία των αντιδραστικών πρωτοκόλλων είναι πως ένας συγκεκριμένος προορισμός γίνεται ενεργός λόγω ενός αιτήματος προερχόμενου από τον κόμβο-πηγή. Έτσι, είναι γνωστό πως οι αντιδραστικές εφαρμογές υλοποιούν δρομολογήσεις «μετά από απαίτηση». Συγκεκριμένα, ο κόμβος-πηγή ζητά να βρει μια διαδρομή προς τον κόμβο-προορισμό, κυρίως μέσω της μετάδοσης μηνυμάτων προς όλους τους κόμβους του δικτύου και αν ένας κόμβος κατέχει μια τέτοια διαδρομή προς τον προορισμό απαντά με ένα μήνυμα-απάντηση που περιέχει και τις πληροφορίες δρομολόγησης.

Η ομάδα των αντιδραστικών πρωτοκόλλων χωρίζεται σε δύο βασικές κατηγορίες που ακολουθούν την αρχή δρομολόγησης «μετά από απαίτηση», αλλά διαφέρουν στον τρόπο εύρεσης διαδρομών. Ένα πρωτόκολλο που ανήκει στην κατηγορία *δρομολόγησης πηγής* αναγκάζει τα μεταδιδόμενα πακέτα δεδομένων να μεταφέρουν όλες τις πληροφορίες της πηγής προς τον προορισμό και κάθε ενδιάμεσος κόμβος προωθεί τα πακέτα αυτά σύμφωνα με τις πληροφορίες που περιέχουν οι επικεφαλίδες τους. Αυτό κάνει πιο έντονο το πρόβλημα τοπικής αποθήκευσης σε κάθε ενδιάμεσο κόμβο, μειώνει όμως την καθυστέρηση στη διαδικασία ενημέρωσης. Επιπρόσθετα, επιτρέπει στους κόμβους αυτούς να μη διατηρούν τις ενημερώσεις των δρομολογίων και τις πληροφορίες των γειτόνων τους στους πίνακές τους. Από την άλλη, στα πρωτόκολλα που ανήκουν στην κατηγορία *δρομολόγησης κόμβος-προς-κόμβο*, ένα πακέτο δεδομένων περιέχει μόνο τον προορισμό και τη διεύθυνση του επόμενου κόμβου. Χρησιμοποιώντας αυτή την αρχή, κάθε ενδιάμεσος κόμβος υποχρεούται να διατηρήσει τις ενημερώσεις των γειτονικών του κόμβων και τις πληροφορίες δρομολόγησης που αφορούν τον επιθυμητό προορισμό. Ένας ενδιάμεσος κόμβος, προωθεί τα πακέτα αυτά σύμφωνα με τις πληροφορίες που τα ίδια τα πακέτα περιέχουν. Η υπακοή στις διαδικασίες της ακολουθίας αυτής χτίζει μια ανθεκτική αρχιτεκτονική που διαχειρίζεται την απρόσμενη τοπολογία ενός δικτύου MANET και βελτιώνει την προσαρμοστικότητα στη δρομολόγηση.

Τα ακόλουθα μέρη του υποκεφαλαίου αυτού περιγράφουν κάποια από τα αντιδραστικά πρωτόκολλα που χρησιμοποιούνται στα δίκτυα MANET.

2.3.3.1. Πρωτόκολλο Dynamic Source Routing (DSR)

Το πρωτόκολλο DSR [11] αποτελεί μια από τις παλαιότερες αντιδραστικές εφαρμογές και σχεδιάστηκε με τέτοιο τρόπο που να εξυπηρετεί δίκτυα με πολλούς κόμβους. Υιοθετεί χαρακτηριστικά όπως η αυτό-διαμόρφωση και προσαρμοστικότητα και με αυτό τον τρόπο ικανοποιεί τις αλλαγές που πιθανόν να πραγματοποιηθούν στην τοπολογία του δικτύου. Επίσης, τα κύρια αρχιτεκτονικά χαρακτηριστικά που διαθέτει το πρωτόκολλο αυτό και αφορούν την *εύρεση* και *συντήρηση διαδρομών*, βεβαιώνουν τη συνέπεια που παρουσιάζει το πρωτόκολλο στις δυναμικές αλλαγές της τοπολογίας. Η διαδικασία *εύρεσης διαδρομής* στο πρωτόκολλο DSR, όπως απεικονίζεται στην εικόνα *2.4,* βασίζεται στην εφαρμογή «μετά από απαίτηση», όπου η επικοινωνία ξεκινά μόνο όταν ένας κόμβος-πηγή ζητά να στείλει δεδομένα σε ένα κόμβο-προορισμό. Η διαδικασία αυτή επιτυγχάνεται με τη χρήση πακέτων *αίτησης διαδρομής* και *απάντησης διαδρομής*, τα οποία στέλνονται μέσω όλων των ενδιάμεσων κόμβων που γνωρίζουν τη συντομότερη διαδρομή προς τον κόμβο-προορισμό. Εντούτοις, η διαδικασία αυτή αναγκάζει τους αποστολείς να ελέγχουν τη διαδρομή τους και επιτρέπει τη χρήση πολλών διαδρομών που οδηγούν στον κόμβο-προορισμό.



Εικόνα 2.4: Διαδικασία εύρεσης διαδρομής στο πρωτόκολλο DSR [19]

Η *διατήρηση διαδρομής* αποτελεί το μηχανισμό που δίνει το δικαίωμα στον κόμβο-πηγή να αλλάξει τη διαδρομή προς τον κόμβο-προορισμό στην περίπτωση κρίσιμης αλλαγής στην τοπολογία, καθώς και στην περίπτωση που μια σύνδεση μεταξύ δύο κόμβων έχει χαθεί. Η αλλαγή στην τοπολογία αφορά τους κόμβους που σχετίζονται με την ενεργή διαδρομή προς τον κόμβο-προορισμό. Στην περίπτωση που δεν υπάρχουν νέες διαδρομές που θα αντικαταστήσουν τις προηγούμενες, ο κόμβος-πηγή ξεκινά τη διαδικασία *εύρεσης διαδρομής* και προσπαθεί να βρει μια νέα διαδρομή προς τον κόμβο-προορισμό. Η διαδικασία *συντήρησης διαδρομής* εκτελείται μόνο όταν η πραγματική μετάδοση των πακέτων δεδομένων μεταξύ πηγής και προορισμού είναι σε εξέλιξη. Τέλος, η *συντήρηση διαδρομής* επιτυγχάνεται με τη χρήση μηνυμάτων επιβεβαίωσης, καθώς και *μηχανισμού εύρεσης λαθών διαδρομής.*

Ακόμα και αν οι ενημερώσεις και αλλαγές στην τοπολογία λαμβάνονται υπόψη από το πρωτόκολλο, εντούτοις σε πραγματικά δεδομένα οι τοπολογίες υστερούν σε θέματα εξέλιξης. Υπάρχουν πολλά μειονεκτήματα λόγω των πακέτων *εύρεσης διαδρομής* και *συντήρησης διαδρομής* και των μεγάλων σε μέγεθος μηνυμάτων (των οποίων το μέγεθος μεγαλώνει εκθετικά με την αύξηση του μεγέθους του δικτύου) που «πλημμυρίζουν» το δίκτυο. Όλα τα μηνύματα που μεταδίδονται περιέχουν στις επικεφαλίδες τους τις πληροφορίες του κόμβου-πηγή και τις πληροφορίες του κόμβου-προορισμού όταν επιστρέφουν. Αυτό συντελεί στη δημιουργία επιπλέον καθυστέρησης και οδηγεί στην κατανάλωση μεγάλου μέρους τους εύρους ζώνης του δικτύου. Ακόμα, το γεγονός αυτό επιδρά αρνητικά στην ενδεδειγμένη λειτουργία ενός δικτύου MANET, μιας και δημιουργούνται περιορισμοί στο εύρος ζώνης του.

2.3.3.2. Πρωτόκολλο Ad hoc On Demand Distance Vector (AODV)

Το πρωτόκολλο AODV [21] είναι ικανό για χρήση τόσο της μεθόδου unicasting όσο και της μεθόδου multicasting και αποτελεί ένα πρότυπο των αντιδραστικών εφαρμογών δρομολόγησης σε δίκτυα MANET. Το πρωτόκολλο βασίζεται σε μια υβριδική διάσταση,

αφού προσαρμόζεται σε γνωρίσματα των πρωτοκόλλων DSR και DSDV. Συγκεκριμένα, κληρονομεί τα γνωρίσματα των πινάκων του δυναμικού πρωτοκόλλου DSDV και τις διαδικασίες ενημέρωσης κατά την εύρεση διαδρομών του αντιδραστικού πρωτοκόλλου DSR. Καινοτομία στον AODV αποτελεί το γεγονός ότι παρέχει μια προσαρμοστική και χωρίς επαναλήψεις συμπεριφορά, η οποία αντιμετωπίζει τις απρόσμενες τοπολογίες στα δίκτυα MANET, εξασφαλίζοντας εξέλιξη και ορθότητα όσον αφορά τη δρομολόγηση.

Περαιτέρω ανάλυση του πρωτοκόλλου αυτού υπάρχει στο υποκεφάλαιο *3.1.1* του εγγράφου αυτού, μιας και αποτελεί ένα εκ των δύο πρωτοκόλλων που μελετήθηκαν εις βάθος για τις ανάγκες της Ατομικής Διπλωματικής Εργασίας.

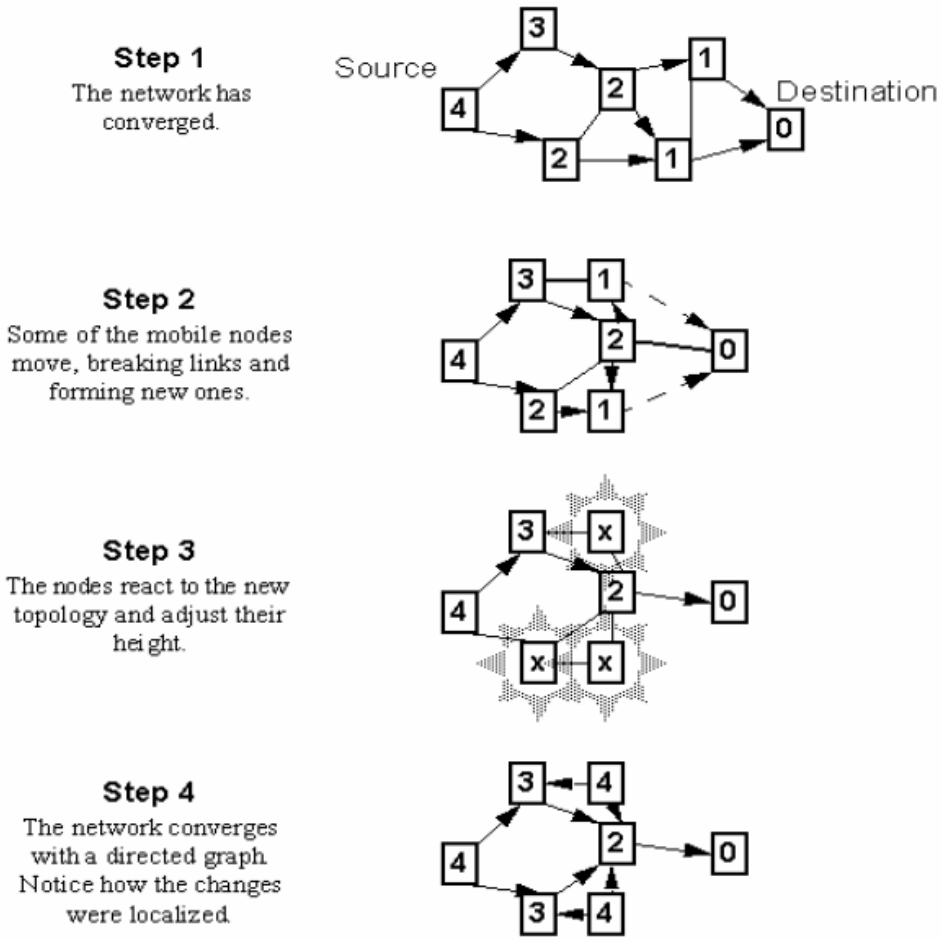
2.3.3.3. Πρωτόκολλο Temporarily-Ordered Routing Algorithm (TORA)

Το πρωτόκολλο TORA [25] αποτελεί μια προσαρμοστική, χωρίς επαναλήψεις εφαρμογή που βασίζεται στην αντιστροφή συνδέσεων και χρησιμοποιείται συχνά σε δυναμικά δίκτυα MANET. Το TORA είναι ένα αντιδραστικό πρωτόκολλο που παρέχει πολλές διαδρομές για κάθε επιθυμητό συνδυασμό πηγής-προορισμού σε περιβάλλον με πολλούς κόμβους.

Το πρωτόκολλο βασίζεται σε ένα μηχανισμό που υποστηρίζει τα τοπικά μηνύματα ελέγχου σε κάθε κόμβο ο οποίος επηρεάζεται από τις αλλαγές στην τοπολογία του δικτύου. Αυτό επιτυγχάνεται χρησιμοποιώντας κόμβους που κατέχουν πληροφορίες μόνο για τους άμεσους γείτονές τους, όπως εφαρμόζεται και στα πρωτόκολλα AODV και DSR.

Συγκεκριμένα, η λειτουργία του TORA χωρίζεται σε τρεις κύριες διαδικασίες: τη *δημιουργία διαδρομής,* τη *συντήρηση διαδρομής* και την *εξάλειψη διαδρομής.* Οι δύο πρώτες διαδικασίες βασίζονται στη συγχρονισμένη εκτίμηση της *μετρικής ύψους* που εισάγεται ως έννοια στο πρωτόκολλο αυτό. Η έννοια αυτή περιλαμβάνει μια εκτίμηση

της λογικής χρονικής περιόδου που παίρνει μια αποτυχία σύνδεσης, καθώς και άλλες πληροφορίες όπως αυτές της διεύθυνσης του κόμβου που σχετίζεται με την αποτυχία, του επιπέδου αναφοράς μετάδοσης δεδομένων που εξετάζει αν όλοι οι κόμβοι εκτελούν πράξεις κάτω από συγχρονισμένες περιόδους χρόνου. Σε σχέση με τη *μετρική ύψους* που βρίσκεται στο γειτονικό κόμβο, γίνεται μια ανάθεση κατευθύνσεων στις συνδέσεις, οι οποίες κατευθύνσεις είναι αυτές που δύναται να πάρει μια διαδρομή ώστε να φτάσει στον προορισμό. Η τεχνική αυτή, σε όρους του πρωτοκόλλου TORA, δίνεται ως Κατευθυνόμενος Μη-κυκλικός Γράφος (Directed Acyclic Graph - DAG). Ο γράφος αυτός αποτελεί ένα μηχανισμό με λειτουργία παρόμοια με αυτή της τεχνικής *αίτησης διαδρομής* και *απάντησης διαδρομής* που χρησιμοποιείται στα πρωτόκολλα AODV και DSR.



Εικόνα 2.5: Βήματα πρωτοκόλλου TORA για δημιουργία DAG [3]

Καθώς προκύπτουν αλλαγές στην τοπολογία του δικτύου, ένας κατευθυνόμενος μη-κυκλικός γράφος «σπάζει» και το γεγονός αυτό καθιστά αναγκαία την εκτέλεση της διαδικασίας *συντήρησης διαδρομής*, η οποία, όπως σε όλα τα άλλα αντιδραστικά πρωτόκολλα, δημιουργεί ένα νέο γράφο που αναπαριστά την πορεία προς τον επιθυμητό προορισμό. Η επαναδημιουργία αυτή του γράφου απαιτεί νέους υπολογισμούς του ύψους των σχετικών κόμβων που βρίσκονταν κοντά στην αποτυχία σύνδεσης που προέκυψε. Όλη η διαδικασία που περιγράφεται πιο πάνω, φαίνεται σχηματικά στην εικόνα *2.5*.

Το πρωτόκολλο TORA θεωρείται ένας πολύ προσαρμοστικός και αυτό-διαμορφώμενος σχεδιασμός που αντιμετωπίζει τις περισσότερες απαιτήσεις που έχουν τα δίκτυα MANET, κυρίως λόγω της παρουσίας του κατευθυνόμενου μη-κυκλικού γράφου. Από την άλλη, το TORA παρουσιάζει πολλά μειονεκτήματα, παρόμοια με αυτά των δυναμικών πρωτοκόλλων, όπως το DSDV. Αυτό παρατηρείται λόγω του εσωτερικού συντονισμού των κόμβων που γίνεται στο γράφο, ο οποίος φέρνει στην επιφάνεια το πρόβλημα *καταμέτρησης προς το άπειρο.* Το μεγάλο μέγεθος της τοπολογίας ενός δικτύου MANET αναγκάζει το TORA να έχει πολλούς κόμβους να προσπαθούν να εντοπίσουν αποτυχίες ταυτόχρονα, γεγονός το οποίο πιθανόν να οδηγήσει σε προβλήματα διατήρησης της ροής δεδομένων στο δίκτυο.

2.3.3.4. Περίληψη Χαρακτηριστικών Αντιδραστικών Πρωτοκόλλων

Τα πρωτόκολλα που ακολουθούν τον αντιδραστικό σχεδιασμό αλγορίθμων επιδεικνύουν ένα διαφορετικό τρόπο δρομολόγησης στα δίκτυα MANET, ο οποίος θεωρείται πολλά υποσχόμενος και σαφώς αποδοτικότερος από τον τρόπο δρομολόγησης των δυναμικών πρωτοκόλλων. Εντούτοις, η αντιδραστική δρομολόγηση, όπως και κάθε σχεδιασμός δρομολόγησης που αφορά τα δίκτυα MANET, έχει τα πλεονεκτήματα και τα μειονεκτήματά της. Τα αντιδραστικά πρωτόκολλα που συζητήθηκαν στα προηγούμενα υποκεφάλαια βελτιώνουν περιοχές όπως η προσαρμοστικότητα και η αυτό-διαμόρφωση,

αλλά δεν ικανοποιούν πλήρως τους περιορισμούς εύρους ζώνης του δικτύου, αφού προφανώς καταναλώνουν μεγάλα ποσά από αυτό για να διατηρήσουν την τοπολογία του δικτύου. Για παράδειγμα, στα πρωτόκολλα DSR, τα πακέτα *απάντησης διαδρομής* δημιουργούν μεγάλο πρόβλημα καθυστέρησης λόγω της «βαριάς» υλοποίησης που τα αναγκάζει να μεταφέρουν μεγάλα ποσά πληροφοριών.

Παράλληλα, το AODV προσπάθησε να εξαλείψει το πρόβλημα αυτό, χρησιμοποιώντας κατά τη μεταφορά των μηνυμάτων *απάντησης διαδρομής* μόνο κάποιες από τις πληροφορίες και αλλάζοντας ορισμένες από τις τοπικές ενημερώσεις της διαδικασίας εύρεσης διαδρομών σε κάθε κόμβο. Παρόλα αυτά, το AODV εξακολουθεί να παρουσιάζει κάποια προβλήματα καθυστέρησης λόγω της μετάδοσης μηνυμάτων *αίτησης διαδρομής* και των ανεξέλεγκτων *απαντήσεων διαδρομών*, παραμένει όμως αποδοτικό όσον αφορά την εξέλιξη του δικτύου.

Το μοντέλο TORA, με τη χρήση του μηχανισμού κατευθυνόμενου μη-κυκλικού γράφου, κατέχει μια ανθεκτική υποδομή που είναι απαραίτητη για την ελεγχόμενη *εύρεση* και *συντήρηση διαδρομής*, αλλά παρουσιάζει προβλήματα διατήρησης της ροής δεδομένων στο δίκτυο σε περιπτώσεις περιβάλλοντος δικτύου με πολλούς κόμβους. Επιπρόσθετα, το TORA, σε αντίθεση με τα υπόλοιπα αντιδραστικά πρωτόκολλα, φαίνεται να αποδίδει καλύτερα στη δρομολόγηση όταν προκύπτουν προβλήματα σε ψηλά επίπεδα μετάδοσης (multicast).

2.3.4 Υβριδικά Πρωτόκολλα (Hybrid Protocols)

Ένα από τα κυριότερα προβλήματα που προκαλούνται, τόσα στα αντιδραστικά όσο και στα δυναμικά πρωτόκολλα, είναι αυτά της καθυστέρησης στη διαδικασία εύρεσης διαδρομών. Λόγω των προβλημάτων αυτών, μια νέα γενιά προσεγγίσεων δρομολόγησης προτάθηκε, αυτή της υβριδικής δρομολόγησης που ενώνει με αποδοτικό τρόπο τα στοιχεία των αντιδραστικών και δυναμικών πρωτοκόλλων. Τα υβριδικά πρωτόκολλα

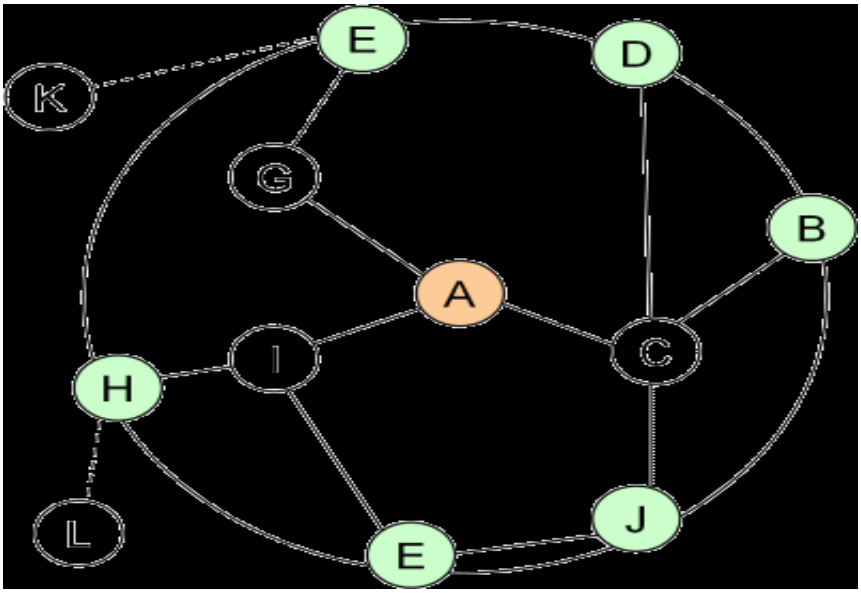
έχουν σχεδιαστεί με τέτοια μορφή που να βελτιώνουν την ιδιότητα εξέλιξης και να αναγκάζουν κοντινούς κόμβους να συνεργάζονται και να συντηρούν δυναμικές διαδρομές προς τον προορισμό και παράλληλα να καθορίζουν διαδρομές απομακρυσμένων κόμβων με τη χρήση μιας νέας στρατηγικής εύρεσης διαδρομών.

Υπάρχουν πολλές διαφορετικές προσεγγίσεις που αφορούν τον τρόπο συνεργασίας των κόμβων και τον καθορισμό διαδρομών για απομακρυσμένους κόμβους σε υβριδικά περιβάλλοντα ανάπτυξης. Κάποια πρωτόκολλα χρησιμοποιούν την ιδέα διαχωρισμού του δικτύου σε ζώνες κόμβων ή δημιουργία ομάδων κόμβων και κάποια άλλα πρωτόκολλα την ιδέα χρήσης δομών δέντρου στους κόμβους. Τα επόμενα υποκεφάλαια παρουσιάζουν ορισμένους από τους υβριδικούς σχεδιασμούς, καταθέτοντας τα πλεονεκτήματα και μειονεκτήματά τους.

2.3.4.1. Πρωτόκολλο Zone Routing (ZRP)

Το πρωτόκολλο ZRP [4] ενώνει τις αρχές αντιδραστικών και δυναμικών πρωτοκόλλων και θεωρείται ως μια προτεινόμενη λύση στο πρόβλημα ύπαρξης καθυστέρησης επικοινωνίας μεταξύ κατανεμημένων κινητών κόμβων.

Οι κόμβοι που λειτουργούν βάσει του ZRP συμπεριφέρονται ως αρχηγοί ομάδας σε ζώνες, όπου όλες οι ζώνες μαζί συνθέτουν την ενιαία ζώνη δρομολόγησης, η οποία χρησιμεύει στη συντήρηση των διαδρομών με μια δυναμική προσέγγιση, παρόμοια με αυτή στο πρωτόκολλο DSDV. Χρησιμοποιώντας το σχεδιασμό αυτό, υπάρχει μια άμεση και γρήγορη πρόσβαση στους επιθυμητούς προορισμούς στην καθορισμένη ζώνη δρομολόγησης. Για να δημιουργήσει τη ζώνη του, όπως φαίνεται και στην εικόνα *2*.6, αρχικά ο κόμβος χρειάζεται να καθορίσει τους άμεσους γείτονές του χρησιμοποιώντας είτε τα πρωτόκολλα *ελέγχου μέσης πρόσβασης* (Medium Access Control - MAC), είτε τα πρωτόκολλα *εύρεσης γειτόνων* (Neighbor Discovery Protocol - NDP), τα οποία βασίζονται στην παραδοσιακή μετάδοση μηνυμάτων “hello”.



Εικόνα 2.6: Καθορισμός ζώνης από τον αρχηγό της ομάδας A, όπου οι K,L (εκτός ορίων) τοποθετούνται από τους E,H αντίστοιχα [28]

Οι κόμβοι εκτός των ορίων της καθορισμένης ζώνης δρομολόγησης είναι ικανοί να ορίσουν μια διαδρομή προς ένα προορισμό χρησιμοποιώντας την «μετά από απαίτηση» προσέγγιση και για το λόγο αυτό να θέσουν σε λειτουργία οποιοδήποτε αντιδραστικό αλγόριθμο με σκοπό την επίτευξη του στόχου τους. Με βάση τον κανόνα αυτό, το ZRP υποστηρίζει την ευελιξία στη συμπεριφορά του και σύμφωνα με τις ανάγκες της τοπολογίας του μπορεί να χρησιμοποιεί διαφορετικές αντιδραστικές στρατηγικές με τρόπο ώστε να εκπληρώνει τις απαιτήσεις που αφορούν την αποδοτικότητα στο δίκτυο MANET. Από την άλλη, το ZRP εγείρει ορισμένα προβλήματα χρήσης της μνήμης λόγω της ιεραρχικής δρομολόγησης, όπου η επιλογή διαδρομής προς τον προορισμό δύναται να μην είναι βέλτιστη, απαιτώντας ψηλότερο επίπεδο πληροφοριών της τοπολογίας. Το γεγονός αυτό απαιτεί μεγάλο ποσοστό μνήμης για να συντονιστεί ένας κόμβος με κάθε άλλο κόμβο που έχει χαμηλότερο επίπεδο γνώσεων της τοπολογίας.

2.3.4.2. Πρωτόκολλο Distributed Dynamic Routing (DDR)

Το πρωτόκολλο DDR [13] αποτελεί έναν αλγόριθμο δρομολόγησης που εφαρμόζει την ιεραρχία δένδρου μεταξύ των κινητών κόμβων. Το DDR κατέχει έναν εντελώς υβριδικό χαρακτήρα και υιοθετεί χαρακτηριστικά από τα άλλα υβριδικά πρωτόκολλα, όπως το DST και ZRP. Επίσης, αποτελεί ένα πρωτόκολλο με επίπεδα, γνώρισμα που καθορίζεται από ένα οποιοδήποτε *Υβριδικό Ad hoc Πρωτόκολλο Δρομολόγησης* (Hybrid Ad hoc Routing Protocol - HARP). Τα στοιχεία αυτά καθιστούν το DDR ως ένα ευέλικτο, προσαρμοστικό και αυτό-διαμορφώμενο πρωτόκολλο, το οποίο μπορεί να αντιμετωπίσει τις περισσότερες από τις απαιτήσεις ενός δικτύου MANET σε θέματα κυρίως ποιότητας υπηρεσίας. Σύμφωνα με την επιλογή του HARP που θα υλοποιείται στο DDR, γίνεται σχετική ανάλυση των πλεονεκτημάτων και μειονεκτημάτων που μπορεί να αυτό να έχει όταν χρησιμοποιηθεί σε ένα δίκτυο MANET.

Οι κόμβοι που χρησιμοποιούν το DDR, αναγνωρίζουν τους πλησιέστερους τους γείτονες με μετάδοση και ανταλλαγή μηνυμάτων, δημιουργώντας με αυτό τον τρόπο δένδρα. Έτσι, οι κόμβοι που βρίσκονται στο ίδιο δένδρο έχουν στη διάθεσή τους πολλές διαδρομές προς τον προορισμό και είναι άμεσα προσβάσιμοι από τους κόμβους του ίδιου δένδρου.

Τα δένδρα συνδέονται μεταξύ τους μέσω κόμβων που χρησιμοποιούνται ειδικά για το σκοπό αυτό, δημιουργώντας ένα δάσος. Κάθε δένδρο που βρίσκεται σε ένα δάσος αποτελεί την ίδια τη ζώνη δρομολόγησής του και υπολογίζει την ταυτότητα της ζώνης αυτής με χρήση ενός αλγορίθμου, παρόμοιου με αυτόν που χρησιμοποιούσε το πρωτόκολλο ZRP. Με τη χρήση αυτού του αλγορίθμου γίνεται ανάθεση μιας συγκεκριμένης ζώνης στους κόμβους και το δίκτυο παίρνει τη μορφή πολλών επικαλυπτόμενων ζωνών.

Το DDR μπορεί να χωριστεί σε έξι φάσεις, οι οποίες εξαρτώνται από τα μηνύματα που μεταδίδονται μεταξύ των κόμβων. Με τη μετάδοση των μηνυμάτων αυτών επιτυγχάνεται η χρήση των λειτουργιών *επιλογής προτιμώμενου γείτονα, κατασκευής δάσους, ομαδοποίησης δένδρων, ομαδοποίησης κόμβων δένδρου, διαχωρισμού ζωνών* και *ονομασίας ζωνών.*

Η *επιλογή προτιμώμενου γείτονα* βασίζεται πλήρως στον κόμβο που κατέχει το μεγαλύτερο αριθμό γειτονικών κόμβων. Με βάση τον κόμβο αυτό, η κατασκευή του δάσους επιτυγχάνεται με την αναγνώριση του πλησιέστερου γείτονα από κάθε κόμβο και τη δημιουργία σύνδεσης με αυτόν. Ακολούθως, το νέο δάσος εκτελεί τον *αλγόριθμο ομαδοποίησης των δένδρων* του και καθορίζει τη δομή της ζώνης κάθε δένδρου, δημιουργώντας του πίνακες δρομολόγησης για κάθε ένα από τα δένδρα αυτά. Αμέσως μετά, κάθε ζώνη εκτελεί τον *αλγόριθμο ομαδοποίησης των κόμβων κάθε δένδρου* για να καθορίσει τη συνδεσιμότητα με τους υπόλοιπους γειτονικούς κόμβους. Η κατανομή των ταυτοτήτων στις ζώνες γίνεται με τη χρήση του *αλγορίθμου ονομασίας ζωνών*, αφού πρώτα γίνει ο *διαχωρισμός ζωνών*.

Τέλος, το DDR χρησιμοποιεί ένα πρωτόκολλο HARP για να καθορίσει ένα σταθερό μονοπάτι μεταξύ της πηγής και του προορισμού, αλλάζοντας αναλόγως τους πίνακες δρομολόγησης κάθε δένδρου που κατασκευάζονται με την αρχική εκτέλεση του πρωτοκόλλου DDR.

2.3.4.3. Πρωτόκολλο AntHocNet

Το πρωτόκολλο AntHocNet [8,16] αποτελεί έναν αλγόριθμο δρομολόγησης που λειτουργεί εντελώς υβριδικά και προτάθηκε από τους Di Caro G., Ducatelle F. και Gambardella L. το Σεπτέμβριο του 2004 [8]. Ανήκει στην ομάδα Ant Colony Optimization (ACO) [10], της οποίας οι περισσότεροι αλγόριθμοι βασίζονται στις αρχές της βιολογίας και συγκεκριμένα στη συμπεριφορά των μυρμηγκιών και τις δραστηριότητές τους.

Το AntHocNet έχει αντιδραστικά και δυναμικά χαρακτηριστικά που συνυπάρχουν παράλληλα, έχοντας πολλές ομοιότητες με αντιδραστικά πρωτόκολλα όπως το AODV και δυναμικά πρωτόκολλα όπως το DSDV. Παρόλα αυτά, καινοτομία του πρωτοκόλλου αυτού αποτελεί η εμφάνιση συμπεριφορών που βασίζονται σε πιθανότητες και αποθηκεύονται στους πίνακες φερομονών. Με βάση τους πίνακες αυτούς, η συμπεριφορά των μυρμηγκιών επιτυγχάνει να καταστήσει το AntHocNet ως ένα προσαρμοστικό και αυτό-διαμορφώμενο πρωτόκολλο που αντιμετωπίζει αποδοτικά τις απρόβλεπτες αλλαγές της τοπολογίας του δικτύου MANET.

Όπως και για το πρωτόκολλο AODV, περαιτέρω ανάλυση του AntHocNet υπάρχει στο υποκεφάλαιο *3.1.2* του εγγράφου αυτού, μιας και αποτελεί ένα εκ των δύο πρωτοκόλλων που μελετήθηκαν εις βάθος για τις ανάγκες της Ατομικής Διπλωματικής Εργασίας.

2.3.4.4. Περίληψη Χαρακτηριστικών Υβριδικών Πρωτοκόλλων

Στα υποκεφάλαια που προηγούνται παρουσιάστηκαν τρία διαφορετικά υβριδικά πρωτόκολλα. Βασισμένοι στα σημεία περιγραφής τους, μπορούμε να συμπεράνουμε πως τα υβριδικά πρωτόκολλα προτείνουν μια νέα προσέγγιση στη δρομολόγηση σε δίκτυα MANET, υιοθετώντας περισσότερα χαρακτηριστικά ανθεκτικότητας και εξέλιξης από αυτά των απλών αντιδραστικών ή δυναμικών πρωτοκόλλων.

Το ZRP, για παράδειγμα, υποστηρίζει την ευελιξία χρήσης οποιασδήποτε αντιδραστικής εφαρμογής, όταν ένας κόμβος βρίσκεται εκτός των ορίων της ζώνης και επομένως μπορεί να προσαρμοστεί στο δίκτυο, ανεξαρτήτως της τοπολογίας του. Ακόμα, η ιεραρχική δομή των κόμβων στο πρωτόκολλο ZRP διευκολύνει τη χρήση μιας ανθεκτικής και αξιόπιστης στρατηγικής που θα εφαρμοστεί στη διαδικασία ενημέρωσης των πινάκων δρομολόγησης. Επιπρόσθετα, αυτό μπορεί να επιτευχθεί με χρήση δομής δένδρων στο πρωτόκολλο DDR, όπου ο αλγόριθμος ομαδοποίησης των κόμβων των δένδρων συντηρεί

και ενημερώνει οποιεσδήποτε αλλαγές στην τοπολογία που αφορούν γειτονικά συνδεδεμένα δένδρα.

Εντούτοις, τα υβριδικά πρωτόκολλα έχουν κάποια μειονεκτήματα λόγω του ότι στηρίζονται σε αντιδραστικές ή δυναμικές εφαρμογές και μια πιθανή λανθασμένη επιλογή μπορεί να οδηγήσει σε αρνητικά αποτελέσματα στη δρομολόγηση. Έτσι, είναι απαραίτητη η επιλογή του καταλληλότερου πρωτοκόλλου HARP και η προσαρμογή στις ανάγκες του τρέχοντος δικτύου MANET στο οποίο εκτελείται, εκ μέρους κάθε υβριδικού πρωτοκόλλου.

2.4 Αλγόριθμοι Μυρμηγκιών για δρομολόγηση σε κινητά ad hoc δίκτυα

Πολλά από τα πρωτόκολλα που περιγράφηκαν πιο πάνω έχουν υλοποιηθεί χρησιμοποιώντας τις βασικές αρχές συμπεριφοράς των μυρμηγκιών ως μονάδες καθώς και ως ομάδες. Κάποια από τα πρωτόκολλα αυτά είναι τα AODV και AntHocNet που περιγράφουν με ακρίβεια και λεπτομέρεια διαφορετικές προσεγγίσεις για τη συμπεριφορά των μυρμηγκιών, υλοποιώντας τη συμπεριφορά τους σε πρωτόκολλα που χρησιμεύουν στη δρομολόγηση πακέτων σε κινητά ad hoc δίκτυα – MANET. Κάθε κόμβος σε τέτοια δίκτυα αναπαρίσταται ως ένα μυρμήγκι και το δίκτυο αποτελεί το σμήνος-ομάδα μυρμηγκιών. Στόχος των δικτύων αυτών είναι η δρομολόγηση πακέτων από μια πηγή σε ένα συγκεκριμένο προορισμό, όπως ακριβώς λειτουργούν τα μυρμήγκια στην προσπάθεια τους να εντοπίσουν την τροφή τους, κινούμενα από τη φωλιά προς το φαγητό.

Συγκεκριμένα, κάθε ομάδα μυρμηγκιών έχει ως στόχο την επιβίωση μέσω της συλλογής τροφής στη φωλιά. Έτσι, κάθε μυρμήγκι ξεκινά από τη φωλιά και τυχαία εξερευνά ένα συγκεκριμένο χώρο γύρω από αυτή, με σκοπό την εύρεση τροφής. Όταν ένα μυρμήγκι βρει την τροφή, παίρνει το δρόμο επιστροφής προς τη φωλιά, δημιουργώντας ένα μονοπάτι που θα ακολουθήσουν και τα άλλα μέλη της ομάδας, ώστε με το χρόνο να

καθοριστεί το βέλτιστο μονοπάτι από τη συγκεκριμένη τοποθεσία τροφής προς τη φωλιά. Στην προσπάθεια τους αυτή να καθορίσουν το βέλτιστο μονοπάτι, αφήνουν στο έδαφος τις φερομόνες, τις οποίες θα εντοπίσουν τα υπόλοιπα μυρμήγκια για να φτάσουν αρχικά προς την τροφή και ακολούθως να επιστρέψουν με τη γρηγορότερο δυνατό τρόπο που βρέθηκε μέχρι στιγμής στη φωλιά τους.

Με βάση αυτή τη γενική λογική έχουν αναπτυχθεί κάποια τα πρωτόκολλα δρομολόγησης στα κινητά ad hoc δίκτυα, εφαρμόζοντας το καθένα τις σχετικές παραλλαγές, ώστε να αναπτυχθεί το πρωτόκολλο που θα ικανοποιεί όλες τις απαιτήσεις των MANET δικτύων με τον καλύτερο δυνατό τρόπο.

2.4.1. Συσχέτιση εννοιών

Στο πλαίσια εκπόνησης της Ατομικής Διπλωματικής Εργασίας ασχοληθήκαμε με δύο πεδία, το πεδίο των καθαρών δικτύων και το πεδίο των αλγόριθμων μυρμηγκιών που εφαρμόζονται στα δίκτυα. Στους αλγόριθμους μυρμηγκιών εισάγονται οι έννοιες, όπως για παράδειγμα των φερομονών, που δεν αποτελούν γνώριμες έννοιες των δικτύων. Έτσι, στο υποκεφάλαιο αυτό του εγγράφου, θα κάνουμε μια προσπάθεια συσχέτισης των εννοιών των δύο αυτών πεδίων που συνδυάζονται για τις ανάγκες Ατομικής Διπλωματικής Εργασίας αυτής.

Τα *μυρμήγκια* που αναφέρονται στους αλγόριθμους μυρμηγκιών, ή αλλιώς οι πράκτορες σε αλγόριθμους σμηνών, αντιστοιχούν στα *μηνύματα αίτησης για εύρεση διαδρομής* που εμφανίζονται στα δίκτυα. Επιπλέον, είναι κατανοητό ότι η *φωλιά* των μυρμηγκιών αποτελεί το *δρομολογητή-πηγή* από τον οποίο ξεκινά το πακέτο και το *φαγητό* το *δρομολογητή-προορισμό* στον οποίο καταλήγει το πακέτο. Στη συνέχεια, αντιστοιχούμε το *κάθε σημείο στο χώρο* κίνησης των μυρμηγκιών με κάθε *δρομολογητή* που παρουσιάζεται ενεργός στο δίκτυο και συμμετέχει στη διαδικασία δρομολόγησης των πακέτων. Επίσης, οι *φερομόνες*, που αναφέρονται συχνά στους αλγόριθμους

μυρμηγκιών, αντιστοιχούν σε συγκεκριμένες *καταχωρήσεις των πινάκων δρομολόγησης* που χρησιμοποιούνται από τα πρωτόκολλα με σκοπό την ορθή μετάδοση του πράκτορα (πακέτου) από τη φωλιά (πηγή) στο φαγητό (προορισμός).

Στα ακόλουθα υποκεφάλαια αναφέρονται οι κυριότεροι αλγόριθμοι μυρμηγκιών που προτάθηκαν για δρομολόγηση σε κινητά ad hoc δίκτυα. Δύο από αυτούς, οι AODV και AntHocNet, μελετήθηκαν σε βάθος, ενώ ένας αλγόριθμος που δεν αναφέρθηκε σε προηγούμενα υποκεφάλαια και βασίζεται αποκλειστικά στη συμπεριφορά των μυρμηγκιών είναι ο AntNet που αναλύεται πιο κάτω:

2.4.2. Αλγόριθμος AntNet

Ο αλγόριθμος AntNet [5] στηρίζεται στις αρχές λειτουργίας ενός σμήνους μυρμηγκιών, όπως αυτές περιγράφηκαν προηγουμένως. Συγκεκριμένα, ο αλγόριθμος αυτός λειτουργεί με βάση δύο τύπους πρακτόρων (στην περίπτωση μας μυρμηγκιών). Τα μυρμήγκια *προώθησης (forward)*, τα οποία μαζεύουν πληροφορίες για την κατάσταση του δικτύου και τα μυρμήγκια *οπισθοχώρησης (backward),* τα οποία χρησιμοποιούν τη γνώση αυτή για να ανανεώσουν τις πληροφορίες που περιέχονται στους πίνακες δρομολόγησης που βρίσκονται στη διαδρομή τους. Ένας δρομολογητής περιέχει ειδικούς πίνακες δρομολόγησης όπου κάθε προορισμός συνδέεται με όλες τις διαπροσωπείες και κάθε μια από αυτές έχει μια συγκεκριμένη πιθανότητα. Η πιθανότητα αυτή ορίζει αν πρέπει να ακολουθηθεί η σύνδεση αυτή κάτω από αυτές τις περιστάσεις. Ο δρομολογητής περιέχει επίσης ένα στατιστικό μοντέλο για αποθήκευση των τιμών χρόνου ταξιδιού προς όλους τους προορισμούς, ώστε σε περίπτωση ανάγκης χρήσης ενός μονοπατιού να είναι γνωστός ο χρόνος ολοκλήρωσης της μετάδοσης και να επιλεχθεί έτσι το καλύτερο μονοπάτι.

Σε τακτικά διαστήματα, κάθε δρομολογητής στέλνει ένα μυρμήγκι προώθησης σε τυχαίο προορισμό στο δίκτυο. Η αποστολή του μυρμηγκιού αυτού είναι να συλλέξει

πληροφορίες για την κατάσταση του δικτύου. Περνώντας από κάθε δρομολογητή αφήνει το χρόνο που απομένει για να φτάσει στο τέλος και ακολούθως καθορίζεται ο επόμενος σταθμός του μυρμηγκιού. Κανονικά αυτό βασίζεται στις πιθανότητες του πίνακα δρομολόγησης. Υπάρχει όμως και μια μικρή πιθανότητα, αναλόγως της παραλλαγής του πρωτοκόλλου, ο επόμενος σταθμός να επιλέγεται τυχαία. Οι ενέργειες αυτές είναι απαραίτητες για την εξερεύνηση του δικτύου και για αντίδραση σε πιθανές αλλαγές στην τοπολογία του δικτύου λόγω αποτυχιών συνδέσεων ή συμφόρησης. Όταν το μυρμήγκι προώθησης φτάνει στον τελικό προορισμό, αφήνει και πάλι το χρόνο που απομένει για να φτάσει στο τέλος και το αναγνωριστικό του δρομολογητή αποθηκεύεται στη στοίβα του μυρμηγκιού. Το μυρμήγκι τότε μετατρέπεται σε μυρμήγκι οπισθοχώρησης, το οποίο ακολουθεί ακριβώς το ίδιο μονοπάτι με το μυρμήγκι προώθησης, αλλά στην αντίθετη κατεύθυνση.

Τα μυρμήγκια οπισθοχώρησης χρησιμοποιούν τις πληροφορίες που συλλέχθηκαν από τα μυρμήγκια προώθησης για να ανανεώσουν τις διάφορες δομές δεδομένων σε κάθε δρομολογητή που βρίσκεται στο μονοπάτι τους. Με βάση τη σύγκριση των πληροφοριών χρόνου της στοίβας και του μοντέλου του δρομολογητή, ανανεώνεται ο πίνακας δρομολόγησης που περιέχει τις πιθανότητες. Όταν το μυρμήγκι οπισθοχώρησης φτάσει στον αρχικό δρομολογητή, πεθαίνει. Τα μυρμήγκια οπισθοχώρησης έχουν μεγαλύτερη προτεραιότητα από τα πακέτα δεδομένων, έτσι τυγχάνουν επεξεργασίας το γρηγορότερο δυνατό, καθιστώντας τον αλγόριθμο πιο προσαρμοστικό. Τα μυρμήγκια προώθησης έχουν την ίδια προτεραιότητα με τα πακέτα δεδομένων έχοντας την ίδια καθυστέρηση, ώστε ο αλγόριθμος να μπορεί να αντιδράσει σε πιθανή συμφόρηση του δικτύου.

Σύμφωνα με το χρόνο ταξιδιού του μυρμηγκιού από την πηγή στον προορισμό, αλλάζουν και οι πιθανότητες στους πίνακες δρομολόγησης. Η μεταβολή στην τιμή των πιθανοτήτων αποτελεί μια ένδειξη της σταθερότητας του δικτύου. Συγκεκριμένα, μια μεγάλη μεταβολή στην τιμή αυτή δείχνει μια ασταθή κατάσταση του δικτύου, ενώ αντίθετα μια μικρή μεταβολή δείχνει μια σταθερή κατάσταση. Σε μια ασταθή

κατάσταση, οι επιδράσεις των πιθανοτήτων γίνονται πιο αδύνατες μιας και δε μπορεί να ειπωθεί με σιγουριά ότι ένας κακός χρόνος ταξιδιού του μυρμηγκιού δείχνει ότι το μονοπάτι που ακολούθησε το μυρμήγκι είναι μεγάλο.

2.4.3. Αλγόριθμος AODV – Αλγόριθμος AntHocNet

Όπως αναφέρθηκε και προηγουμένως, τα πρωτόκολλα *AODV* και *AntHocNet* περιγράφονται με λεπτομέρεια στα υποκεφάλαια *3.1.1* και *3.1.2* αντίστοιχα.

2.5 Αξιολόγηση αλγορίθμων SI για δρομολόγηση

Πολλές υλοποιήσεις σε κινητά ad hoc δίκτυα χρησιμοποιούν αλγόριθμους εμπνευσμένους από τη φύση, αφού οι αλγόριθμοι αυτοί περιγράφουν με ακρίβεια τη συμπεριφορά κάθε συστατικού μέρους του δικτύου. Αυτό συμβαίνει με την κατάλληλη συσχέτιση των μερών του δικτύου με τα αντίστοιχα μέρη της φύσης. Για παράδειγμα, ένας μέρος του δικτύου είναι ένα πακέτο που θα μεταδοθεί και αντιστοιχείται με την έννοια του πράκτορα (π.χ. μυρμήγκι) που κινείται στο χώρο. Άλλο ένα παράδειγμα είναι ο κόμβος-πηγή και ο κόμβος-προορισμός που συσχετίζονται με τη φωλιά και το φαγητό του σμήνους αντίστοιχα. Χρησιμοποιώντας αυτή τη συσχέτιση, επιτυγχάνεται μια υλοποίηση με λιγότερες πιθανότητες να αποτύχει αφού έχει ήδη δοκιμαστεί στον πραγματικό κόσμο, με τα σμήνη.

2.6 Εργαλείο Υλοποίησης: MASON

Το MASON είναι ένας πυρήνας βιβλιοθήκης γρήγορων διακριτών γεγονότων για πολυπρακτορικά συστήματα, με σκοπό την προσομοίωση και τρέχει χρησιμοποιώντας τη γλώσσα προγραμματισμού Java. Έχει σχεδιαστεί για να αποτελέσει τη βάση για μεγάλες εφαρμογές που εκτελούνται σε Java, καθώς και για να παρέχει πολλές λειτουργίες σε αρκετά πολύπλοκες ανάγκες προσομοίωσης κάποιας εφαρμογής. Το εργαλείο περιέχει

μοντέλα βιβλιοθήκης και προαιρετική ακολουθία δεδομένων που αποτελούνται από κάποια εργαλεία απεικόνισης σε δυσδιάστατη (2D) και τρισδιάστατη (3D) μορφή.

Το MASON αποτελεί μια κοινή προσπάθεια του εργαστηρίου Evolutionary Computation του πανεπιστημίου George Mason και του κέντρου GMU Center for Social Complexity [17].

Για τις ανάγκες της Ατομικής Διπλωματικής Εργασίας, χρησιμοποιήσαμε την έκδοση 13 του εργαλείου MASON. Ακολούθως, εισάγαμε τα αρχεία MASON στο περιβάλλον Eclipse [12] που διαχειρίζεται προγράμματα γραμμένα σε γλώσσα προγραμματισμού Java. Με τον τρόπο αυτό κατέστη δυνατή η επεξεργασία των αρχείων και η εκτέλεση προσομοιώσεων με βάση το περιβάλλον Eclipse και τα γνωρίσματα του εργαλείου MASON.

Η επιλογή του εργαλείου MASON έγινε μετά από σύσταση της επιβλέπουσας καθηγήτριας, λόγω των θετικών γνωρισμάτων που κατέχει σαν εφαρμογή προσομοίωσης σύνθετων και παράλληλων υπολογιστικών προβλημάτων. Το MASON έχει την ικανότητα να εκτελεί κάποιους αλγόριθμους απεικονίζοντας γραφικά την εκτέλεση, βήμα προς βήμα, δίνοντας έτσι καλύτερο νόημα στη θεωρία κάθε ενός από τους αλγορίθμους. Έτσι, χρησιμοποιήσαμε αυτή την ικανότητα του εργαλείου και υλοποιήσαμε ορισμένους αλγόριθμους σε γλώσσα προγραμματισμού Java, έχοντας πάντα ως βάση τα ήδη υπάρχοντα αρχεία του MASON. Με αυτό τον τρόπο, εκπληρώσαμε τον αρχικό μας στόχο, που ήταν η γραφική απεικόνιση των αλγορίθμων συμπεριφοράς των μυρμηγκιών, ώστε να εξάγουμε στο τέλος κάποια χρήσιμα συμπεράσματα για τη συμπεριφορά τους και να συγκρίνουμε όσο το δυνατόν πιο ολοκληρωμένα τους υλοποιημένους αλγορίθμους.

ΚΕΦΑΛΑΙΟ 3:

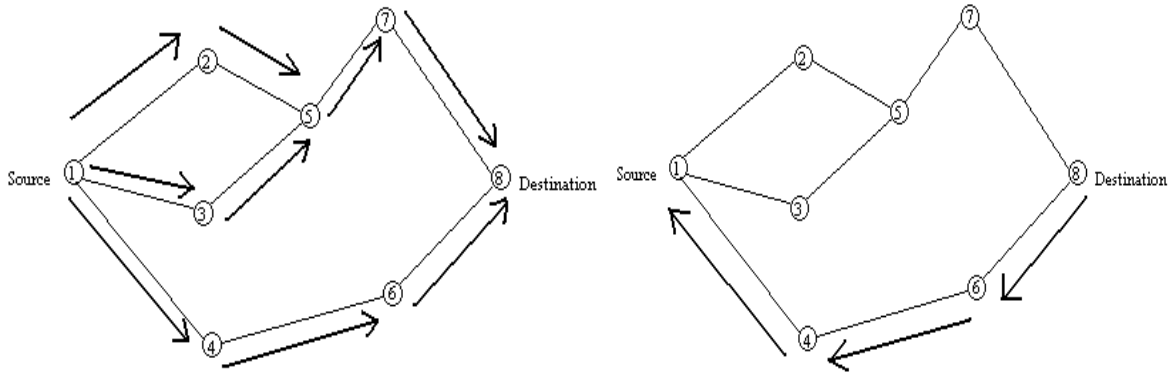
ΑΛΓΟΡΙΘΜΟΙ ΚΑΙ ΨΕΥΔΟ-ΚΩΔΙΚΑΣ

3.1. Αλγόριθμοι που μελετήθηκαν

Από το σύνολο των αλγορίθμων που μελετήσαμε στη διάρκεια της ανασκόπησης της βιβλιογραφίας που διεξήχθη στην πρωταρχική φάση της Ατομικής Διπλωματικής Εργασίας, αποφασίσαμε να μελετήσουμε σε περισσότερο βάθος δύο από αυτούς. Η επιλογή έγινε λόγω της μεγάλης χρήσης των αλγορίθμων αυτών, τόσο σε άρθρα και δημοσιεύσεις, όσο και σε υλοποιήσεις που δημοσιεύτηκαν στο διαδίκτυο. Οι δύο αλγόριθμοι που επιλέξαμε είναι οι AODV και AntHocNet, ένας αντιδραστικός και ένας υβριδικός αλγόριθμος αντίστοιχα. Για τις ανάγκες της υλοποίησης των δύο αλγορίθμων χρειάστηκε να μελετήσουμε βήμα προς βήμα, με ακρίβεια και λεπτομέρεια τη λειτουργία κάθε αλγορίθμου. Η περιγραφή των AODV και AntHocNet ακολουθεί στα επόμενα υποκεφάλαια.

3.1.1. AODV

Το AODV [21], ως ένα πρωτόκολλο που λειτουργεί καθαρά «μετά από απαίτηση», συντηρεί μόνο τις διαδρομές των ενεργών προορισμών, πάντα σύμφωνα με την απαίτηση που υπάρχει από την πηγή. Όταν ένας κόμβος-πηγή ζητά να στείλει δεδομένα σε έναν κόμβο-προορισμό, στέλνει αρχικά ένα μήνυμα *αίτησης διαδρομής* (Route Request - RREQ) και κάθε ενδιάμεσος κόμβος που δεν αποτελεί τον κόμβο-προορισμό και δεν περιέχει πληροφορίες δρομολόγησης για αυτόν, δημιουργεί μια νοητή σύνδεση με την πηγή και ξαναστέλνει το μήνυμα αίτησης διαδρομής προς το δίκτυο. Ένας κόμβος που έχει τη σωστή διαδρομή προς τον κόμβο-προορισμό ή αποτελεί ο ίδιος τον κόμβο-προορισμό, στέλνει ένα μήνυμα *απάντησης διαδρομής* (Route Reply - RREP), δίνοντας τις απαραίτητες πληροφορίες στον κόμβο-πηγή για τον αριθμό των κόμβων και τη διαδρομή που πρέπει να ακολουθήσει το πακέτο δεδομένων για να βρεθεί στον κόμβο-προορισμό. Η διαδικασία αυτή με τις απαιτήσεις και απαντήσεις φαίνεται και σχηματικά στην εικόνα *3.1.*



Εικόνα 3.7: Εύρεση διαδρομής με *RREQ* και *RREP* στο πρωτόκολλο AODV [21]

Στο πρωτόκολλο AODV, οι αποτυχίες συνδέσεων και οι συνδέσεις γειτόνων εντοπίζονται με το γνωστό μηχανισμό της μετάδοσης μηνυμάτων “hello”. Οι κόμβοι αναγνωρίζουν τους γείτονές τους με χρήση των μηνυμάτων αυτών και στην περίπτωση που ένας κόμβος

δε λάβει ένα μήνυμα από το γείτονά του, τότε δηλώνει αποτυχία σύνδεσης με το γείτονα αυτό. Παράλληλα, στην περίπτωση που τα δεδομένα προωθούνται προς τον προορισμό και εντοπιστεί μια αποτυχία σύνδεσης, τότε μεταδίδεται ένα μήνυμα *λάθους διαδρομής* (RERR) προς τα πίσω, μέχρι να φτάσει στην πηγή. Καθώς το μήνυμα αυτό μεταδίδεται, οι ενδιάμεσοι κόμβοι ενημερώνουν τις πληροφορίες δρομολόγησης που αφορούν το συγκεκριμένο προορισμό και μόλις το μήνυμα ληφθεί από την πηγή, πραγματοποιείται η ακύρωση της διαδρομής και μια νέα διαδικασία εύρεσης διαδρομής ξεκινά.

3.1.2. AntHocNet

Το AntHocNet [8] είναι παράλληλα ένα αντιδραστικό και δυναμικό πρωτόκολλο, έχοντας πολλές ομοιότητες με αντιδραστικά πρωτόκολλα όπως το AODV και δυναμικά πρωτόκολλα όπως το DSDV. Εντούτοις, καινούργιο γνώρισμα που χαρακτηρίζει το πρωτόκολλο αυτό είναι η συμπεριφορά που κατευθύνεται από πιθανότητες που αποθηκεύονται στους πίνακες φερομονών. Χρησιμοποιώντας τις πιθανότητες, τα πακέτα ελέγχου (στην περίπτωση μας μυρμήγκια) επιτυγχάνουν να καταστήσουν το AntHocNet προσαρμοστικό και αυτό-διαμορφώμενο σε σχέση με τις απρόσμενες αλλαγές στην τοπολογία ενός MANET δικτύου. Τα γνωρίσματα της προσαρμοστικότητας και αυτό-διαμόρφωσης του AntHocNet ικανοποιούν κάποιες από τις απαιτήσεις που αφορούν τη βέλτιστη υβριδική συμπεριφορά ενός δικτύου.

Ο αλγόριθμος εκτελεί δύο φάσεις, την αντιδραστική και τη δυναμική, αφού αποτελεί μια συνένωση αντιδραστικών και δυναμικών χαρακτηριστικών:

Αντιδραστική Φάση

Το αντιδραστικό στοιχείο του AntHocNet είναι υπεύθυνο για την *εύρεση διαδρομής* και *καθορισμό μονοπατιού* προς έναν άγνωστο προορισμό. Συνεπώς, όταν ο κόμβος-πηγή αρχίζει να επικοινωνεί με τον κόμβο-προορισμό και η διαδρομή προς τον κόμβο-προορισμό δεν είναι γνωστή, ο κόμβος-πηγή μεταδίδει ένα ορισμένο αριθμό

αντιδραστικών μυρμηγκιών προώθησης (ΑΜΠ), F_d^s, για να βρει ένα μονοπάτι. Στα πλαίσια αυτής της αρχικής μετάδοσης, κάθε γείτονας του *s* λαμβάνει μια απάντηση F_d^s(k). Για τις πληροφορίες που παρατίθενται πιο κάτω θεωρούμε κάθε σύνολο τέτοιων απαντήσεων που προέρχονται από το ίδιο μυρμήγκι σαν μια *γενιά μυρμηγκιών*. Το ζήτημα για κάθε μυρμήγκι F_d^s(k) είναι να βρει ένα μονοπάτι που να συνδέει το s (πηγή) με το d (προορισμός). Σε κάθε κόμβο, ένα μυρμήγκι μεταδίδει μια νέα γενιά μυρμηγκιών με βάση την ύπαρξη ή όχι πληροφοριών στον τρέχων κόμβο για τον προορισμό d. Οι πληροφορίες δρομολόγησης ενός κόμβου *i* αναπαρίστανται από τους πίνακες φερομονών T^i. Η είσοδος $T_{nd}^i \in R$ του πίνακα αυτού αποτελεί την τιμή φερομόνης που ορίζει την εκτίμηση *ποιότητας (goodness)* του μυρμηγκιού, i, που προχωρεί προς τον κόμβο-προορισμό, d, διαμέσου ενός γειτονικού κόμβου, n. Η εκτίμηση αυτή της ποιότητας σχετίζεται με το αντίστροφο του κόστους από έναν κόμβο σε έναν άλλον και μετριέται με τιμές χρόνου. Το κόστος υπολογίζεται χρησιμοποιώντας διάφορες μετρικές όπως η καθυστέρηση από την αρχή μέχρι το τέλος της διαδρομής που μπορεί να μετρηθεί χρησιμοποιώντας διάφορες τεχνικές [7]. Αν υπάρχουν οι πληροφορίες δρομολόγησης, το μυρμήγκια επιλέγουν τον επόμενο κόμβο που θα κινηθούν, n, με πιθανότητα P_{nd}:

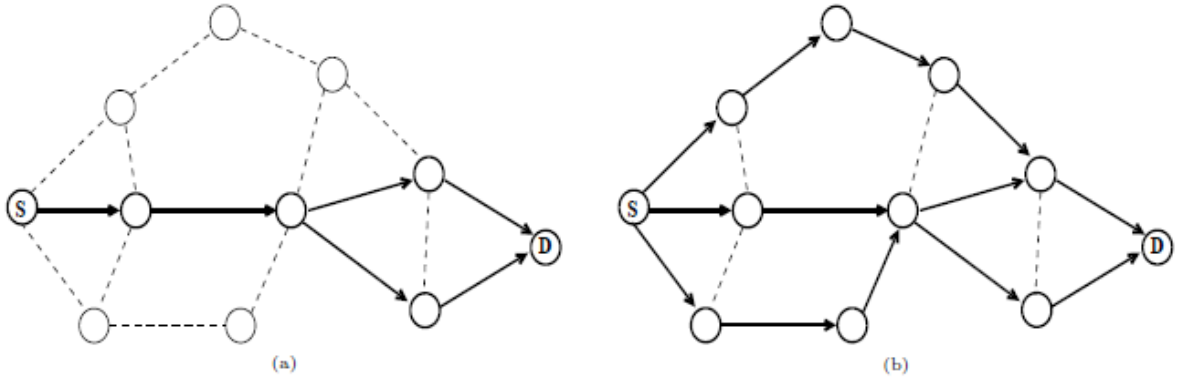
$$P_{nd} = \frac{(T_{nd}^i)^{\beta 1}}{\sum_{j \in N_d^i} (T_{nd}^i)^{\beta 1}}, \beta_1 \geq 1,$$

όπου N_d^i είναι το σύνολο των γειτόνων του i με το οποίο χτίζεται το μονοπάτι προς τον προορισμό d και β_1 είναι η παράμετρος που ορίζει το ποσοστό εξερεύνησης του χώρου εκ μέρους των μυρμηγκιών.

Αν δεν υπάρχουν διαθέσιμες πληροφορίες φερομονών, το μυρμήγκι εκτελεί μια νέα μετάδοση, δημιουργώντας μια άλλη γενιά μυρμηγκιών. Λόγω της μετάδοσης αυτής, τα μυρμήγκια μπορούν να πολλαπλασιαστούν γρήγορα, ακολουθώντας διαφορετικά μονοπάτια προς τον προορισμό. Όταν ένας κόμβος λάβει πολλά μυρμήγκια της ίδιας

γενιάς, συγκρίνει το μονοπάτι που διέγραψε κάθε ένα από αυτά με το μονοπάτι των μυρμηγκιών που έφτασαν και ανήκουν στην ίδια γενιά. Μόνο εάν τα νέα μυρμήγκια ικανοποιούν κάποια κριτήρια θα συνεχίσουν να εξερευνούν το χώρο. Χρησιμοποιώντας την πολιτική αυτή, η καθυστέρηση του δικτύου μειώνεται με τη διαγραφή των μυρμηγκιών που ακολουθούν κακά μονοπάτια, καθώς υπάρχει ακόμα η πιθανότητα για εύρεση νέων καλύτερων μονοπατιών.

Παρόλα αυτά, έχει μια αρνητική επίδραση μιας και το πρώτο μυρμήγκι που φτάνει περνά χωρίς ειδική αξιολόγηση, ενώ τα επόμενα μυρμήγκια πρέπει να ικανοποιούν ορισμένα κριτήρια για να συνεχίσουν, αλλιώς απορρίπτονται. Το γεγονός αυτό μπορεί να οδηγήσει σε μονοπάτια που μοιάζουν με *χαρταετό*, όπως φαίνεται στην εικόνα *3.2a.* Με στόχο τη δημιουργία ενός *πλέγματος* χωριστών μονοπατιών όπως φαίνεται στην εικόνα *3.2b,* το οποίο παρέχει πολύ καλύτερη προστασία σε περιπτώσεις αποτυχίας συνδέσεων, λαμβάνουμε υπόψη στη διαδικασία επιλογής τον πρώτο κόμβο που θέλει να κινηθεί το μυρμήγκι. Αν ο πρώτος κόμβος είναι διαφορετικός από όσους επιλέχθηκαν προηγουμένως από άλλα μυρμήγκια, τότε εφαρμόζονται λιγότερο περιοριστικά κριτήρια, με αποτέλεσμα τη δημιουργία νέων μονοπατιών από την πηγή στον προορισμό.



Εικόνα 3.8: (a) Μονοπάτια σε σχήμα χαρταετού, (b) Πλέγμα μονοπατιών [7]

Κάθε ΑΜΠ αποθηκεύει τη λίστα $\mathfrak{P}$ των κόμβων [1,…..,n] που επισκέφθηκε. Μόλις το ΑΜΠ φτάσει στον κόμβο-προορισμό μετατρέπεται σε *αντιδραστικό μυρμήγκι οπισθοχώρησης (ΑΜΟ)* και ακολουθεί αντίστροφα το ίδιο ακριβώς μονοπάτι [n,……..,1] για να επιστρέψει στον κόμβο-πηγή. Αυτό επιτυγχάνεται με τη χρήση της λίστας $\mathfrak{P}$ που κρατά όλους τους κόμβους που επισκέφθηκε το ΑΜΠ προτού φτάσει στον κόμβο-προορισμό. Στο στάδιο αυτό, η εκτίμηση του χρόνου, $\hat{T}_p$, που χρειάζεται ένα πακέτο για να ταξιδέψει προς τον προορισμό και χρησιμοποιείται για να ανανεωθούν οι πίνακες δρομολόγησης, υπολογίζεται από τα ΑΜΟ σε κάθε κόμβο που επισκέπτονται, αποθηκεύοντας τις φερομόνες που σχετίζονται με την ποιότητα μεταξύ των δύο κόμβων που μόλις επισκέφθηκε το μυρμήγκι. Το $\hat{T}_p$ αποτελεί το άθροισμα των τοπικών εκτιμήσεων $\widehat{T_{i+1}^{i}}$σε κάθε κόμβο i ∈ $\mathfrak{P}$ του χρόνου για να φτάσει το μυρμήγκι στον επόμενο κόμβο:

$$\hat{T}_p = \sum_{i=1}^{n-1} \widehat{T_{i+1}^{i}}$$

Οι πληροφορίες αυτές είναι σημαντικές στα μετέπειτα στάδια για τα υπόλοιπα ΑΜΠ που πρόκειται να μεταδοθούν και θα πάρουν τις πιθανότητες που χρειάζονται για να μάθουν την τοποθεσία του επόμενου κόμβου στην πορεία προς τον κόμβο-προορισμό. Το γεγονός αυτό οδηγεί τα μυρμήγκια στην εύρεση ενός συντομότερου μονοπατιού από αυτό που ήδη χρησιμοποιείται και την εισαγωγή των αλλαγών της τοπολογίας του δικτύου στην κίνησή τους. Τέλος, μια διαδικασία που εφαρμόζεται στα δίκτυα (και όχι στον πραγματικό κόσμο μυρμηγκιών) είναι η εξής: μόλις τα ΑΜΟ φτάσουν στον κόμβο-πηγή, ξεκινά η καθιερωμένη μετάδοση δεδομένων μεταξύ του κόμβου-πηγή και του κόμβου-προορισμού, οι οποίο ξεκινούν να ανταλλάσσουν πληροφορίες [8].

Δυναμική Φάση

Το δυναμικό μέρος του AntHocNet είναι υπεύθυνο για τη *συντήρηση μονοπατιού* και την *εξερεύνηση* με σκοπό την επίτευξη εναλλακτικών και καλύτερων διαδρομών προς τον κόμβο-προορισμό. Το σημαντικότερο στοιχείο για την επιτυχία του δυναμικού μηχανισμού είναι οι *πληροφορίες-δόλωμα* που υπολογίζονται ως επί το πλείστον με τη χρήση μηνυμάτων "hello" που μεταδίδονται περιοδικά μεταξύ γειτονικών κόμβων. Η κυριότερη διαφορά των μηνυμάτων "hello" στο AntHocNet με άλλους αλγορίθμους είναι ότι τα μηνύματα περιλαμβάνουν πληροφορίες δρομολόγησης, οι οποίες δίνουν μια εκτίμηση του κόστους που απαιτείται για να φτάσει ένα μυρμήγκι σε έναν αριθμό ενεργών προορισμών. Συγκεκριμένα, ένας κόμβος που δημιουργεί το δικό του μήνυμα "hello" ενημερώνεται από τους πίνακες δρομολόγησής του και επιλέγει έναν αριθμό προορισμών με τις καλύτερες εκτιμώμενες τιμές φερομονών που ανατίθενται σε κάθε έναν από αυτούς. Η καλύτερη φερομόνη χρησιμοποιείται από κάθε κόμβο για δημιουργία της φερομόνης-δόλωμα, η οποία θεωρείται ως η εναλλακτική φερομόνη που θα αντικαταστήσει την κανονική φερομόνη που δημιουργείται στην αντιδραστική φάση από τα AMO.

Η φερομόνη-δόλωμα ή αλλιώς *εικονική φερομόνη* αποθηκεύεται σε διαφορετικούς πίνακες που βρίσκονται σε κάθε κόμβο και ονομάζονται *πίνακες φερομονών.* Αυτό συμβαίνει με τέτοιο τρόπο που να αποφεύγεται η ανάμιξη κανονικών και τεχνητών φερομονών, που μπορεί να οδηγήσει σε ανεπιθύμητους βρόγχους στη διαδικασία δρομολόγησης. Η εικονική φερομόνη αποθηκεύεται με σκοπό την υποβοήθηση κάθε κόμβου να συγκρίνει αν η τρέχουσα κανονική φερομόνη είναι καλύτερη από την εικονική ή όχι. Στην περίπτωση που η εικονική φερομόνη είναι πολύ καλύτερη από την κανονική, η κανονική αλλάζει, παίρνοντας την τιμή της εικονικής. Επίσης, στην περίπτωση που ένας κόμβος δεν έχει στην κατοχή του πληροφορίες δρομολόγησης για τον κόμβο-προορισμό χρησιμοποιεί την εικονική φερομόνη και ενεργοποιεί μια νέα εναλλακτική διαδρομή προς τον κόμβο-προορισμό.

Όπως προαναφέρθηκε, η εικονική φερομόνη συγκρίνεται με την κανονική και αν η πρώτη είναι καλύτερη (τουλάχιστον 10%), τότε το νέο μονοπάτι που χαρακτηρίζει τη συγκεκριμένη τιμή της εικονικής φερομόνης ελέγχεται από τα *δυναμικά μυρμήγκια προώθησης (ΔΜΠ)* που γεννήθηκαν από το συγκεκριμένο κόμβο. Τα ΔΜΠ μεταβιβάζονται από κόμβο σε κόμβο με τον ίδιο τρόπο που το κάνουν τα ΑΜΠ, χρησιμοποιώντας δηλαδή πιθανότητες και όταν φτάσουν στον κόμβο όπου οι φερομόνες δεν είναι διαθέσιμες, απορρίπτονται. Στην περίπτωση που ένα ΔΜΠ φτάσει στον κόμβο-προορισμό, μετατρέπεται σε *δυναμικό μυρμήγκι οπισθοχώρησης (ΔΜΟ)* ακολουθώντας το ίδιο μοτίβο πρόσβασης με οπισθοχώρηση στον κόμβο-πηγή, όπως και τα ΑΜΟ στην αντιδραστική φάση. Μόλις ένα ΔΜΟ φτάνει στον κόμβο-πηγή, το μονοπάτι που ακολούθησε θεωρείται έγκυρο και η κανονική φερομόνη αντικαθίσταται από την εικονική φερομόνη, ενεργοποιώντας το νέο εναλλακτικό μονοπάτι. Έχοντας ως κρατούμενο ότι ένα δίκτυο MANET καθορίζεται από διάφορους κόμβους, ο μηχανισμός αυτός που περιγράψαμε δημιουργεί ένα πλέγμα (εικόνα *3.2b*) εναλλακτικών διαδρομών προς τον κόμβο-προορισμό. Η στρατηγική αυτή επιτυγχάνει εύρεση νέων και καλύτερων μονοπατιών και χρήση τους σε περιπτώσεις αποτυχίας σύνδεσης ή σημαντικής αλλαγής στην τοπολογία του δικτύου [8].

3.2. Ψευδο-κώδικας μιας εκδοχής των αλγορίθμων

Προτού ξεκινήσουμε την υλοποίηση των αλγορίθμων σε γλώσσα προγραμματισμού Java, επιλέξαμε μετά από εισήγηση της επιβλέπουσας καθηγήτριας, να γράψουμε ψευδο-κώδικα για τη συμπεριφορά των μυρμηγκιών σε κάθε έναν από τους αλγόριθμους. Η δημιουργία του ψευδο-κώδικα έγινε με σκοπό την κατανόηση του αλγορίθμου, καθώς και της ακριβούς συμπεριφοράς των μυρμηγκιών, ώστε αργότερα στη λεπτομερή κωδικοποίηση να μην αντιμετωπίσουμε σοβαρά προβλήματα που θα μας εμπόδιζαν στη συνέχιση της υλοποίησης μιας εκδοχής των δύο αλγορίθμων, AODV και AntHocNet.

Αναφέρουμε ότι η υλοποίηση που έγινε στα πλαίσια της Ατομικής Διπλωματικής Εργασίας αφορά τη μεταφορά της βασικής ιδέας στην οποία στηρίζονται οι αλγόριθμοι στο επίπεδο των μυρμηγκιών. Συγκεκριμένα, θεωρούμε ότι τα μυρμήγκια που κινούνται στο χώρο, ξεκινώντας από τη φωλιά και ψάχνοντας να βρουν το φαγητό, εφαρμόζουν τις ιδέες των δύο αλγορίθμων, AODV και AntHocNet, κατά την κίνησή τους στο χώρο.

Ακολουθεί ο ψευδο-κώδικας των σημαντικότερων κλάσεων κάθε αλγορίθμου, οι οποίες καθορίζουν και την ακριβή συμπεριφορά των μυρμηγκιών στο χώρο κίνησής τους. Οι κλάσεις που παραλείπονται δεν περιγράφουν τις ενέργειες των μυρμηγκιών, αλλά χρησιμοποιούνται για άλλους λόγους, γι' αυτό και ο πλήρης κώδικας αφήνεται για το *Παράρτημα Α*.

3.2.1. AODV

Το πρωτόκολλο AODV υλοποιεί μια αντιδραστική συμπεριφορά των μυρμηγκιών, που κινούνται προς την τροφή και επιστρέφουν στη φωλιά, αφήνοντας φερομόνες στο έδαφος. Ο ψευδο-κώδικας που ακολουθεί παρουσιάζει τις πιο κύριες λειτουργίες του αλγορίθμου με αφαιρετικό και ανεξάρτητο από την υλοποίηση τρόπο.

```
Ants-Forage
If hasReachedFood
        isBackward = TRUE
        Ant-Return-At-Home
Else
        Ant-Find-Food-Source
```

Ο ψευδο-κώδικας αυτός περιγράφει τη συμπεριφορά των μυρμηγκιών αν αυτό έχει φτάσει επιτυχώς στο φαγητό ή όχι. Δηλαδή, αν το μυρμήγκι έχει φτάσει στο φαγητό (hasReachedFood = TRUE), καθιστά το μέχρι τώρα μυρμήγκι προώθησης σε μυρμήγκι οπισθοχώρησης (isBackward = TRUE) και το κατευθύνει να κινηθεί προς τη φωλιά.

Αντίθετα, αν το μυρμήγκι δεν έχει φτάσει ακόμα στο φαγητό (hasReachedFood = False), εξακολουθεί να κινείται με σκοπό την εύρεση του φαγητού.

```
Ant-Return-At-Home
If ant is at food source
        Orientation = neighbour with maximum home pheromones
X = Select-Location(forward locations) – N, NE, NW
If X = NULL
        X = Select-Location(neighbour locations) – N, NE, NW, S, SW, SE, W, NW
If X != NULL
        Drop-Food-Pheromones
        Orientation = direction to X from current location
        Move to X
        If ant is at home
                Drop food item
                hasReachedFood = FALSE
                isBackward = FALSE
```

Το μέρος αυτό του ψευδο-κώδικα εξηγεί τον τρόπο με τον οποίο κινείται το μυρμήγκι όταν έχει φτάσει σε κάποια στιγμή στο φαγητό και θέλει να επιστρέψει στη φωλιά. Αν το μυρμήγκι βρίσκεται στο σημείο που βρήκε το φαγητό, τότε επιλέγει ως επόμενη του θέση το γείτονα που κατέχει τις περισσότερες φερομόνες. Όταν αναφερόμαστε σε γείτονα, εννοούμε κάποια θέση από τις οκτώ γειτονικές που έχει κάθε θέση στο χώρο: βόρεια, βορειοδυτικά, δυτικά, νοτιοδυτικά, νότια, νοτιοανατολικά, ανατολικά και βορειοανατολικά. Για να επιλέξει την επόμενη του θέση, το μυρμήγκι επιλέγει μεταξύ τριών κατευθύνσεων: βόρεια, βορειοανατολικά και βορειοδυτικά ώστε να επιταχύνει την κίνηση του στο χώρο, χρησιμοποιώντας μόνο κατευθύνσεις που το οδηγούν προς τα εμπρός. Στην περίπτωση που δεν καταφέρει να βρει την επόμενη του θέση μεταξύ των τριών κατευθύνσεων που μόλις αναφέραμε, ψάχνει να βρει τη θέση του ανάμεσα στους οκτώ γείτονές του. Όταν βρεθεί η θέση στην οποία θα κινηθεί το μυρμήγκι, τότε κινείται προς αυτήν και αφήνει τις φερομόνες φωλιάς που κατέχει. Με βάση την πορεία που χαράχτηκε με την επιλογή του γείτονα προηγουμένως, το μυρμήγκι συνεχίζει να κινείται

στις θέσεις γύρω του. Όταν το μυρμήγκι φτάσει στη φωλιά αφήνει νοητά το φαγητό που πήρε από τη θέση φαγητού και μετατρέπεται πάλι σε μυρμήγκι προώθησης (isBackward = FALSE) που προσπαθεί εκ νέου να βρει το φαγητό (hasReachedFood = False).

Σημείωση: Όταν το μυρμήγκι κινείται από τη φωλιά προς το φαγητό αφήνει στο έδαφος φερομόνες φωλιάς, ενώ όταν βρει το φαγητό και κινείται ανάποδα προς τη φωλιά αφήνει φερομόνες φαγητού. Η διαφοροποίηση αυτή του είδους των φερομονών βοηθά το μυρμήγκι στην επιλογή των σωστών φερομονών κατά την αναζήτηση του φαγητού και την επιστροφή στη φωλιά. Όταν δηλαδή προσπαθεί να βρει το φαγητό επιλέγει θέσεις στο έδαφος με μεγάλο αριθμό φερομονών φαγητού, ενώ αντίθετα όταν προσπαθεί να επιστρέψει στη φωλιά επιλέγει θέσεις στο έδαφος με μεγάλο αριθμό φερομονών φωλιάς.

```
Ant-Find-Food-Source
If ant is at home
        Orientation = neighbour with maximum food pheromones
X = Select-Location(forward locations)
If X = NULL
        X = Select-Location(neighbour locations)
If X != NULL
        Drop-Home-Pheromones
        Orientation = direction to X from current location
        Move to X
        If ant is at food source
                Pick up food item
                hasReachedFood = TRUE
                isBackward = TRUE
```

Το κομμάτι αυτό του ψευδο-κώδικα περιγράφει την κίνηση του μυρμηγκιού στην προσπάθεια εύρεσης του φαγητού. Αρχικά, το μυρμήγκι βρίσκεται στη φωλιά και επιλέγει το γείτονα (όπως τον εξηγήσαμε προηγουμένως) με τις περισσότερες φερομόνες φαγητού. Από τις θέσεις που εντοπίζει το μυρμήγκι επιλέγει μια εκ των τριών

κατευθύνσεων: βόρεια, βορειοανατολικά και βορειοδυτικά. Περιορίζουμε έτσι το μυρμήγκι να κινηθεί μόνο στις κατευθύνσεις προώθησης για να βρει πιο σύντομα το φαγητό. Αν όμως το μυρμήγκι αποτύχει να βρει μια θέση στις κατευθύνσεις που το περιορίσαμε, τότε ψάχνει μια άλλη θέση, έχοντας να επιλέξει μέσα από οκτώ πιθανές κατευθύνσεις (οι οκτώ κατευθύνσεις ορίζονται με βάση τους οκτώ γείτονες όπως περιγράφηκαν πιο πάνω). Όταν το μυρμήγκι βρει τη θέση που μπορεί να κινηθεί, κινείται προς αυτήν, αφήνει φερομόνες φωλιάς και εξακολουθεί να κινείται με τον ίδιο τρόπο μέχρι που να φτάσει στο φαγητό. Όταν αυτό γίνει, παίρνει νοητά το μέρος του φαγητού που πρέπει (hasReachedFood = TRUE) και μετατρέπεται σε μυρμήγκι οπισθοχώρησης (isBackward = TRUE) που προσπαθεί να επιστρέψει στη φωλιά.

```
Select-Location(orientations)
orientations = orientations – orientationsWithTooManyAnts
If orientations = NULL
        Return NULL
Else
        Select an orientation from the set with probability based on
                FoodPheromonesAtCurrentLocation
```

Το μέρος αυτό του ψευδο-κώδικα περιγράφει τη διαδικασία εύρεσης θέσης για να κινηθεί το μυρμήγκι. Συγκεκριμένα, η μέθοδος παίρνει σαν είσοδο την κατεύθυνση που επέλεξε προηγουμένως το μυρμήγκι για να κινηθεί και αφαιρεί όσες θέσεις έχουν πολλά μυρμήγκια την τρέχουσα στιγμή. Ακολούθως, αν δεν υπάρχουν άλλες κατευθύνσεις για να κινηθεί το μυρμήγκι επιστρέφει NULL, αλλιώς επιλέγει μια κατεύθυνση με βάση πιθανότητες που στηρίζονται στο ποσό των φερομονών που υπάρχουν στη θέση που βρίσκεται το μυρμήγκι (ο ακριβής υπολογισμός της πιθανότητας δίνεται στο υποκεφάλαιο *3.1.2* όπου περιγράφεται ο αλγόριθμος AntHocNet).

Drop-Home-Pheromones

```
If ant is at home
        Deposit a fixed amount of home pheromones
Else
        MAX = maximum amount of home pheromones of neighbours
        D = MAX – home pheromones at current location
        If D > 0
                Deposit D home pheromones at current location
```

Drop-Food-Pheromones

```
If ant is at food source
        Deposit a fixed amount of food pheromones
Else
        MAX = maximum amount of food pheromones of neighbours
        D = MAX – food pheromones at current location
        If D > 0
                Deposit D food pheromones at current location
```

Όπως εξηγήσαμε και προηγουμένως, οι φερομόνες χωρίζονται σε δύο είδη, τις φερομόνες φωλιάς και φαγητού. Στα κομμάτια αυτά του ψευδο-κώδικα δίνεται μια περιγραφή των ενεργειών που χρειάζεται να εκτελέσει το μυρμήγκι για να αφήσει τις φερομόνες του στο έδαφος. Αν το μυρμήγκι βρίσκεται στη φωλιά/φαγητό, «αφήνει» στο έδαφος ένα ορισμένο ποσό φερομονών φωλιάς/φαγητού. Αλλιώς, αν το μυρμήγκι βρίσκεται σε κάποια ενδιάμεση θέση, τότε υπολογίζεται το μέγιστο ποσό φερομονών των γειτόνων του μυρμηγκιού (MAX) και αφαιρείται από αυτό το ποσό φερομονών της θέσης που βρίσκεται το μυρμήγκι. Στη συνέχεια, ελέγχεται αν το νέο ποσό φερομονών που υπολογίζεται (D) είναι μεγαλύτερο του μηδέν και στην περίπτωση που αυτό ισχύει το μυρμήγκι αφήνει το ποσό φερομονών (D) φωλιάς/φαγητού στη θέση που βρίσκεται. Όλες αυτές οι πράξεις γίνονται ώστε να μην αφήνει κάθε μυρμήγκι το ποσό φερομονών που κατέχει στην περίπτωση που το ποσό αυτό είναι μικρότερο του ήδη υπάρχοντος ποσού φερομονών του εδάφους.

3.2.2. AntHocNet

Το πρωτόκολλο AntHocNet υλοποιεί μια υβριδική συμπεριφορά των μυρμηγκιών. Όταν λέμε υβριδική συμπεριφορά εννοούμε ότι κάποια από τα μυρμήγκια που κινούνται στο χώρο ψάχνουν απλά την τροφή και επιστρέφουν στη φωλιά όταν τη βρουν (αντιδραστικά μυρμήγκια), ενώ κάποια άλλα ψάχνουν για καλύτερα μονοπάτια ταυτόχρονα με την αναζήτηση της τροφής (δυναμικά μυρμήγκια). Τα αντιδραστικά μυρμήγκια κινούνται και στον AntHocNet με τον ίδιο τρόπο που κινούνται τα μυρμήγκια στον AODV, αφού ο AODV περιέχει μόνο αντιδραστικά μυρμήγκια. Στον ψευδο-κώδικα που ακολουθεί παρουσιάζουμε τη συμπεριφορά και κίνηση μόνο των δυναμικών μυρμηγκιών μιας και τα αντιδραστικά μυρμήγκια κινούνται με τον ίδιο ακριβώς τρόπο που περιγράφει ο ψευδο-κώδικας στο υποκεφάλαιο *3.2.1*.

Τα δυναμικά μυρμήγκια διαφέρουν με τα αντιδραστικά στο σκοπό κίνησής τους στο χώρο. Δηλαδή τα δυναμικά μυρμήγκια, αντίθετα με τα αντιδραστικά, δεν κινούνται μόνο για να βρουν το φαγητό ξεκινώντας από τη φωλιά, αλλά και για να βρουν νέα και καλύτερα μονοπάτια για να καλύψουν τη διαδρομή αυτή. Έτσι παραθέτουμε μόνο το σημείο του ψευδο-κώδικα όπου τα δυναμικά μυρμήγκια προσπαθούν, μέσω της μετάδοσης, να ανακαλύψουν βέλτιστα μονοπάτια που να οδηγούν όλα τα μυρμήγκια της ομάδας από τη φωλιά στο φαγητό και αντίστροφα.

```
Ant-Find-Food-Source
If ant is at home
        Orientation = neighbour with maximum food pheromones
Broadcast based on a probability
X = Select-Location(forward locations)
If X = NULL
        X = Select-Location(neighbour locations)
If X != NULL
        Drop-Home-Pheromones
        Orientation = direction to X from current location
```

```
Move to X
If ant has made more than two broadcasts
        Delete ant
Else
        If ant is at food source without a single broadcast
                Sample an existing path
                Pick up food item
                hasReachedFood = TRUE
                isBackward = TRUE
        Else if ant is at food source with at least one broadcast at any point
                Explore new paths
                Pick up food item
                hasReachedFood = TRUE
                isBackward = TRUE
```

Όπως και προηγουμένως, αν το μυρμήγκι βρίσκεται στη φωλιά επιλέγει την κατεύθυνση του σύμφωνα με το ποσό των φερομονών φαγητού κάθε ενός από τους οκτώ γείτονές του. Ακολούθως, εκτελεί μετάδοση νέων μυρμηγκιών (δημιουργεί δηλαδή μια νέα γενιά μυρμηγκιών) με βάση μια πιθανότητα [7]. Στη συνέχεια, όπως και τα αντιδραστικά μυρμήγκια, κινούνται προς την τοποθεσία που αποφασίζεται με βάση την κατεύθυνση που επέλεξαν και όταν φτάσουν στη νέα τους θέση, αφήνουν τις φερομόνες φωλιάς και εξακολουθούν να κινούνται με τον ίδιο τρόπο μέχρι που:

1. Το μυρμήγκι εκτελέσει δύο μεταδόσεις και δεν έχει φτάσει στη φωλιά. Στην περίπτωση αυτή το μυρμήγκι «πεθαίνει», δηλαδή διαγράφεται από την προσομοίωση.
2. Το μυρμήγκι έχει φτάσει στο φαγητό. Αν έχει φτάσει στο φαγητό χωρίς να έχει εκτελέσει μετάδοση, τότε θεωρεί ότι το καλύτερο μονοπάτι είναι αυτό που ήδη υπάρχει. Αν το μυρμήγκι έχει φτάσει στο φαγητό και έχει εκτελέσει έστω μια μετάδοση, τότε συνεχίζει την εξερεύνηση με σκοπό την εύρεση καλύτερου μονοπατιού. Και στις δύο περιπτώσεις, παίρνει το φαγητό που πρέπει (hasReachedFood = TRUE) και μετατρέπεται σε μυρμήγκι οπισθοχώρησης (isBackward = TRUE).

3.3. Προκαταρκτική σύγκριση AODV και AntHocNet

Θέλουμε στο σημείο αυτό να συγκρίνουμε τους δύο αλγόριθμους, λαμβάνοντας υπόψη ό,τι διαβάσαμε στη βιβλιογραφία και ξέρουμε για τους δύο αλγόριθμους και προτού βγάλουμε τα συμπεράσματά μας μέσω της υλοποίησης.

Όπως αναφέρεται στο άρθρο των Di Caro, Ducatelle και Gambardella [7], που πρότεινε για πρώτη φορά τον αλγόριθμο AntHocNet, οι δύο αλγόριθμοι χρησιμοποιούνται πολύ στα κινητά ad hoc δίκτυα λόγω των καλών χαρακτηριστικών που παρουσιάζουν. Έτσι, οι AODV και AntHocNet συγκρίνονται μέσα από πειράματα με σκοπό να αποφασιστεί ποιος αλγόριθμος είναι καλύτερος και σε ποιους τομείς. Τα πειράματα έγιναν βάσει τριών μεταβαλλόμενων παραμέτρων: το επίπεδο κίνησης των κόμβων, την πυκνότητα μεταξύ των κόμβων και το μέγεθος του δικτύου.

Επίπεδο κίνησης των κόμβων

Εξετάστηκαν τέσσερεις μετρήσεις για να εξαχθούν συμπεράσματα για τους δύο αλγόριθμους. Αρχικά, μελετήθηκε η *μέση καθυστέρηση μετάδοσης πακέτου*, όπου ο AntHocNet παρουσιάστηκε να έχει σημαντικά χαμηλότερη καθυστέρηση μετάδοσης. Ακολούθως, εξετάστηκε η *αναλογία μετάδοσης πακέτου*, όπου ο AODV φάνηκε να έχει ελαφρά καλύτερα αποτελέσματα. Στη συνέχεια, έγιναν πειράματα για να φανεί κατά πόσον ο AntHocNet είναι καλύτερος σε θέματα *μέσης καθυστέρησης jitter* και *καθυστέρησης δρομολόγησης.* Φάνηκε λοιπόν ότι σε μετρήσεις μέσης καθυστέρησης jitter καλύτερος είναι ο αλγόριθμος AntHocNet, ενώ αντίθετα με το πέρασμα του χρόνου ο AODV δείχνει χαμηλότερες μετρήσεις καθυστέρησης δρομολόγησης [7].

Πυκνότητα μεταξύ των κόμβων

Οι δύο αλγόριθμοι συγκρίθηκαν στο επίπεδο της πυκνότητας μεταξύ των κόμβων του δικτύου. Έτσι, σε κάθε πείραμα αυξανόταν η πυκνότητα μεταξύ των κόμβων ώστε να φανεί αν κάποιος αλγόριθμος υπερτερεί του άλλου σε ορισμένες μετρήσεις. Αρχικά,

μελετήθηκε η παράμετρος της *μέσης καθυστέρησης μετάδοσης πακέτου,* όπου ο AntHocNet παρουσιαζόταν πολύ καλύτερος από τον AODV. Ακολούθως, εξετάστηκε η *αναλογία μετάδοσης πακέτου*, όπου ο AODV φάνηκε λίγο καλύτερος από τον AntHocNet χωρίς όμως να έχουν μεγάλες διαφορές μεταξύ τους. Τέλος, μελετήθηκε η *καθυστέρηση δρομολόγησης,* όπου αν και ο AntHocNet ξεκινούσε έχοντας μεγαλύτερες τιμές καθυστέρησης, εντούτοις μετά από ελάχιστη αύξηση της πυκνότητας των κόμβων άρχισε να μειώνει την καθυστέρηση αυτή. Αντίθετα, ο AODV αν και ξεκινούσε έχοντας σχεδόν μηδενική καθυστέρηση, εντούτοις με την αύξηση της πυκνότητας αύξησε εκθετικά την καθυστέρηση δρομολόγησης [7].

Μέγεθος δικτύου

Η σύγκριση μεταξύ των δύο αλγορίθμων έγινε και πάλι βάσει των παραμέτρων της *μέσης καθυστέρησης μετάδοσης πακέτου,* της *αναλογίας μετάδοσης πακέτου* και της *καθυστέρησης δρομολόγησης*, αυτή τη φορά στο επίπεδο του μεγέθους του δικτύου. Έτσι, σε κάθε νέα μέτρηση αυξανόταν το μέγεθος του δικτύου ώστε να διαπιστωθεί ποιος αλγόριθμος παρουσιάζεται να λειτουργεί καλύτερα. Αρχικά, μελετήθηκε η μέση καθυστέρηση μετάδοσης πακέτου, όπου ο AntHocNet φάνηκε να είναι πολύ καλύτερος από τον AODV. Ακολούθως, εξετάστηκε η αναλογία μετάδοσης πακέτου, όπου ο AODV παρουσιάστηκε ελαφρώς καλύτερος από τον AntHocNet. Τέλος, μελετήθηκε η καθυστέρησης δρομολόγησης, όπου ο AntHocNet φάνηκε να έχει πολύ καλύτερα αποτελέσματα από τον AODV που με την αύξηση του δικτύου παρουσίαζε κάθε φορά μεγαλύτερη αύξηση της καθυστέρησης [7].

Συμπεραίνουμε λοιπόν, ότι σε γενικό βαθμό ο AntHocNet παρουσιάζει καλύτερα αποτελέσματα από τον AODV, μιας και όταν υπάρχουν μεγάλες διαφορές ο AntHocNet φαίνεται να λειτουργεί καλύτερα από τον AODV.

ΚΕΦΑΛΑΙΟ 4:

ΥΛΟΠΟΙΗΣΗ ΑΛΓΟΡΙΘΜΩΝ

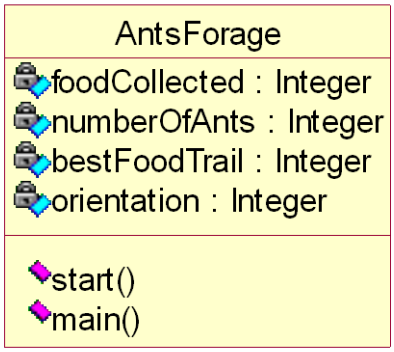
4.1. Επεξήγηση υλοποίησης αλγορίθμων

Με βάση τον ψευδο-κώδικα που αναπτύχθηκε και παρατίθεται στο προηγούμενο κεφάλαιο, ξεκινήσαμε την υλοποίηση των αλγορίθμων σε γλώσσα προγραμματισμού Java. Έτσι, αναπτύχθηκε λεπτομερής κώδικας που υλοποιούσε βήμα προς βήμα την ακριβή συμπεριφορά των μυρμηγκιών, σύμφωνα πάντα με τους δύο αλγόριθμους που θέλαμε να αναπτύξουμε.

Με σκοπό τη βαθύτερη κατανόηση της κωδικοποίησης των αλγορίθμων από τους αναγνώστες του εγγράφου αυτού, αποφασίσαμε να εισάγουμε στο κεφάλαιο αυτό τις περιγραφές των κλάσεων για κάθε αλγόριθμο. Συγκεκριμένα, παραθέσαμε την κάθε κλάση και τις κύριες μεθόδους και χαρακτηριστικά της. Ακολούθως, προσπαθήσαμε να δώσουμε μια επεξήγηση για των κλάσεων, δίνοντας μια περιγραφή των κυριότερων λειτουργιών των αλγορίθμων. Ο πλήρης κώδικας για κάθε κλάση των δύο αλγορίθμων αφήνεται στο Παράρτημα Α.

4.1.1 Κοινές Κλάσεις AODV και AntHocNet

4.1.1.1 Κλάση AntsForage



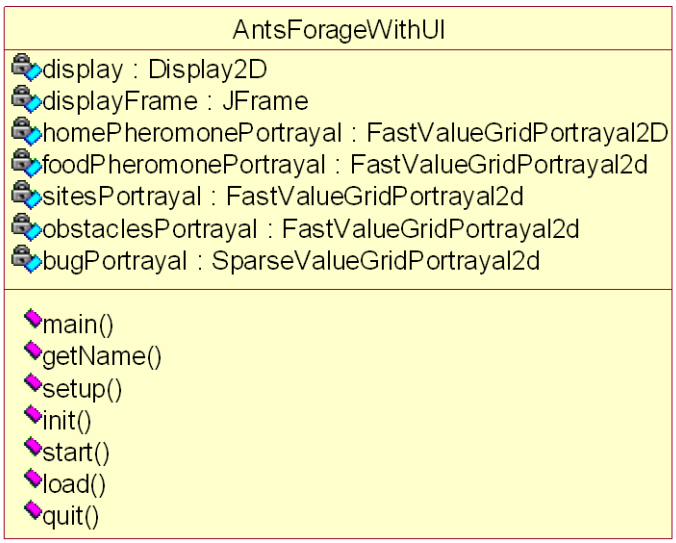
Η κλάση αυτή περιγράφει με ακρίβεια τη διαδικασία που ακολουθείται για συντονισμό της κίνησης όλων των μυρμηγκιών στο χώρο. Υλοποιεί δύο μόνο μεθόδους: τη start() που περιγράφει την αρχικοποίηση της ομάδας των μυρμηγκιών και τη main() που δίνει το σήμα για εκκίνηση της διαδικασίας προσομοίωσης. Για τις ανάγκες αυτού του αλγόριθμου χρειάζεται να δημιουργήσουμε έναν ορισμένο αριθμό αντιδραστικών μυρμηγκιών και στην περίπτωση του AntHocNet έναν ορισμένο αριθμό δυναμικών μυρμηγκιών.

Οι μεταβλητές που χρησιμοποιούνται καθορίζουν χαρακτηριστικά της ομάδας των μυρμηγκιών που δημιουργούνται. Συγκεκριμένα, ορίζεται το συνολικό ποσό φαγητού που συνέλεξαν τα μυρμήγκια σε κάθε προσομοίωση (foodCollected), ο συνολικός αριθμός των μυρμηγκιών που δημιουργούνται σε κάθε προσομοίωση (numberOfAnts), το μήκος του καλύτερου μονοπατιού που εντοπίστηκε μέχρι τώρα (bestFoodTrail), καθώς και η κατεύθυνση του μυρμηγκιού (orientation).

Κατά την αρχικοποίηση του περιβάλλοντος όπου θα κινηθούν τα μυρμήγκια, δίνεται η επιλογή μέσω του προγράμματος για εισαγωγή εμποδίων στο χώρο (παράμετρος OBSTACLES). Η εισαγωγή αυτή γίνεται κατά την αρχικοποίηση των μερών του χώρου (με διαστάσεις GRID_HEIGHT και GRID_WIDTH) στον οποίο θα κινούνται τα μυρμήγκια κατά τη διάρκεια της προσομοίωσης. Τα εμπόδια αυτά τοποθετούνται στο χώρο και

αποτελούν σημεία στα οποία είναι αδύνατο να κινηθούν τα μυρμήγκια. Επίσης, κατά την αρχικοποίηση του χώρου τοποθετούνται τα σημεία φωλιάς και φαγητού (συντεταγμένες HOME και FOOD), τα οποία θα αποτελούν σημεία αναφοράς της συμπεριφοράς των μυρμηγκιών. Στη συνέχεια, καθορίζουμε το μέγιστο αριθμό μυρμηγκιών που μπορούν να λαμβάνουν μέρος στην προσομοίωση (MAX_ANTS), το μέγιστο αριθμό μυρμηγκιών που μπορούν να βρίσκονται σε μια θέση του χώρου (MAX_ANTS_PER_LOCATION), το χρόνο ύπαρξης ενός μυρμηγκιού στο χώρο της προσομοίωσης (TIME_TO_LIVE), τα νέα μυρμήγκια που δημιουργούνται κάθε σε βήμα (NEW_ANTS_PER_TIME_STEP) ανεξαρτήτως των μεταδόσεων που δύναται να εκτελέσει κάθε μυρμήγκι, τον αριθμό των αρχικών μυρμηγκιών που ξεκινούν στην προσομοίωση (INITIALANTS), το μήκος του χειρότερου μονοπατιού από τη φωλιά στο φαγητό (WORSE_FOOD_TRAIL) που ορίζεται σε ένα αριθμό μεγαλύτερο κάθε μονοπατιού και τέλος ορισμένες παραμέτροι που αφορούν το ποσό των φερομονών.

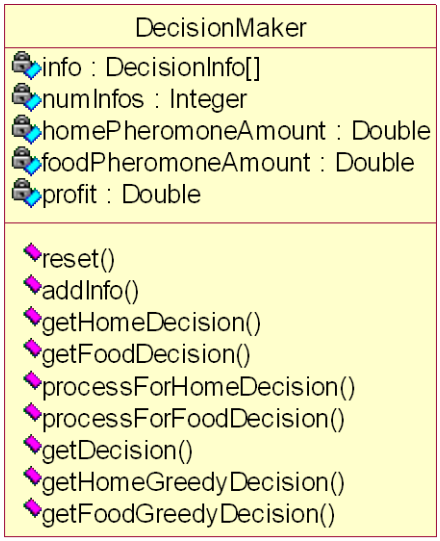
4.1.1.2 Κλάση AntsForageWithUI



Η κλάση AntsForageWithUI χρησιμοποιείται ώστε να αρχικοποιηθούν τα χρώματα και χαρακτηριστικά του χώρου της προσομοίωσης. Η μέθοδος main() καλεί όλες τις άλλες μεθόδους με σκοπό τον καθορισμό των γνωρισμάτων του χώρου, ενώ η getName()

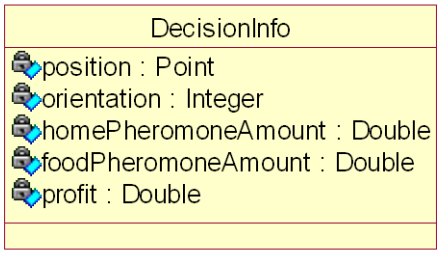
χρησιμοποιεί το όνομα που παίρνει για να το εισάγει στο παράθυρο της προσομοίωσης. Επίσης, η μέθοδος setupPortrayals() χρησιμοποιείται για να καθοριστούν οι χρωματισμοί κάθε ξεχωριστού αντικειμένου που μπορεί να εμφανιστεί στο χώρο της προσομοίωσης. Τέλος, οι μέθοδοι start(), load(), init() και quit() χρησιμοποιούνται με σκοπό την ορθή ανταπόκριση στα καλέσματα των κουμπιών του παραθύρου της προσομοίωσης.

4.1.1.3 Κλάση DecisionMaker



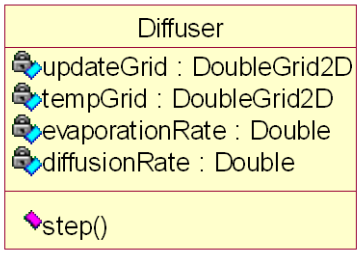
Η κλάση DecisionMaker είναι υπεύθυνη για τη διαμόρφωση των αποφάσεων που έχουν να κάνουν με το μυρμήγκι και τη συμπεριφορά του στο χώρο. Συγκεκριμένα, η μέθοδος reset() διαγράφει κάθε πληροφορία που υπάρχει για τις γνώσεις του μυρμηγκιού, ενώ η μέθοδος addInfo() προσθέτει τις πληροφορίες που κατέχει το μυρμήγκι στη μεταβλητή info, ώστε να μπορούν να τύχουν επεξεργασίας στη συνέχεια. Επίσης, οι μέθοδοι getHomeDecision(), getFoodDecision, getDecision(), getHomeGreedyDecision() και getFoodGreedyDecision() χρησιμοποιούν τις άλλες δύο μεθόδους processForHomeDecision() και processForFoodDecision() με σκοπό τον καθορισμό συγκεκριμένων τιμών σε κάποια από τα πεδία της μεταβλητής info που περιέχει όλες τις πληροφορίες που αφορούν το μυρμήγκι.

4.1.1.4 Κλάση DecisionInfo



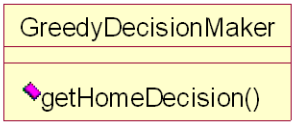
Η κλάση αυτή καλείται για δημιουργία ενός νέου σημείου (position) στο χώρο, ενώ οι υπόλοιπες μεταβλητές χρησιμοποιούνται για να καθοριστούν τα χαρακτηριστικά του νέου αυτού σημείου.

4.1.1.5 Κλάση Diffuser



Η κλάση Diffuser χρησιμοποιεί τη μέθοδο step() ώστε να κατανέμει με ορθό τρόπο το ποσό των φερομονών σε κάθε θέση του χώρου, με τη βοήθεια των μεταβλητών updateGrid, tempGrid, evaporationRate και diffusionRate που περιέχουν τις μετρικές και όρια που επιτρέπονται στην προσομοίωση.

4.1.1.6 Κλάση GreedyDecisionMaker



Η κλάση αυτή χρησιμοποιείται για να διαμορφωθούν οι αποφάσεις που επηρεάζουν την «άπληστη» συμπεριφορά των μυρμηγκιών. Τα μυρμήγκια που συμπεριφέρονται με απληστία καθορίζονται από την αρχή της προσομοίωσης μέσω παραμέτρων και

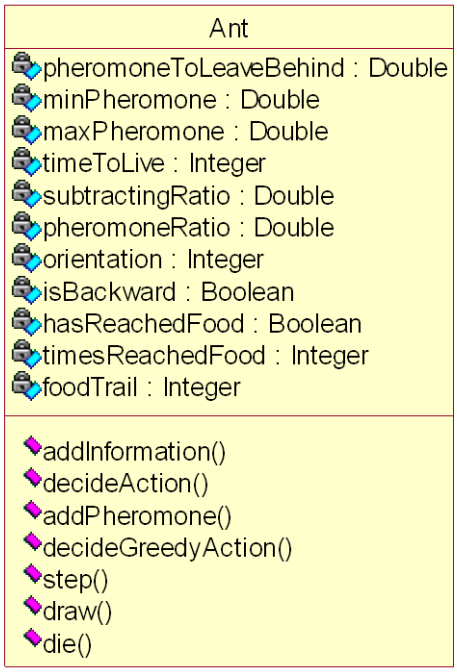
περιγράφουν μια συμπεριφορά που χαρακτηρίζεται από συνεχή διερεύνηση του χώρου, έστω και αν έχει ήδη βρεθεί το φαγητό μια φορά.

4.1.2 Ειδικές κλάσεις AODV και AntHocNet

Στο υποκεφάλαιο αυτό θα αναλύσουμε τις κλάσεις που διαφοροποιούνται στους δύο αλγόριθμους, αφού οι κλάσεις που περιγράφηκαν προηγουμένως αποτελούν κλάσεις με κοινές λειτουργίες στους δύο αλγόριθμους, AODV και AntHocNet.

4.1.2.1 AODV

4.1.2.1.1 Κλάση Ant



Η κλάση αυτή περιγράφει την ακριβή κίνηση ενός μυρμηγκιού του αλγορίθμου AODV. Στην αρχική φάση της κίνησης του μυρμηγκιού τοποθετούνται οι αρχικές ποσότητες, τόσο φερομονών φωλιάς, όσο και φερομονών φαγητού, καθώς και οι αρχικές συντεταγμένες του μυρμηγκιού στο χώρο. Κάθε μυρμήγκι παίρνει τις τιμές που του

αναθέτονται ώστε να μπορεί να κινηθεί στο χώρο, pheromoneToLeaveBehind, minPheromone, maxPheromone, timeToLive και orientation. Επίσης, κάθε μυρμήγκι στην αρχή της προσομοίωσης θεωρείται μυρμήγκι προώθησης (isBackward = FALSE) και σημειώνεται ότι δεν έχει βρει ακόμα το φαγητό (hasReachedFood = FALSE). Έτσι, ο αριθμός των φορών που το μυρμήγκι έχει βρει το φαγητό ορίζεται στο μηδέν (timesReachedFood = 0) και αρχικά το καλύτερο μονοπάτι που έχει βρεθεί είναι το χειρότερο που καθορίζεται από την κλάση AntsForage. Η ανάθεση αυτή των τιμών γίνεται από τη μέθοδο addInformation().

Αφού καθοριστούν τα χαρακτηριστικά κάθε μυρμηγκιού, αυτό ξεκινά την κίνησή του στο χώρο με σκοπό την εύρεση του φαγητού και επιστροφή στη φωλιά. Το κάθε μυρμήγκι έχει οκτώ επιλογές κίνησης στο χώρο, όπως περιγράφηκαν σε προηγούμενα υποκεφάλαια, βρισκόμενο αρχικά σε κάποια από τις θέσεις του χώρου. Σύμφωνα με την επιλογή που γίνεται, η μέθοδος decideAction() καθορίζει την ακριβή κίνηση του μυρμηγκιού στο χώρο.

Με βάση και πάλι τη θέση που βρίσκεται το μυρμήγκι σε κάποια στιγμή της προσομοίωσης, επιλέγεται μια μέθοδος υπολογισμού του ποσού των φερομονών. Συγκεκριμένα, σύμφωνα με τη θέση του, το μυρμήγκι υπολογίζει το ποσό φερομονών που πρέπει να αφήσει στη θέση όπου βρίσκεται, με βάση τις μεταβλητές subtractingRatio και pheromoneRatio και ακολούθως αφήνει τη φερομόνη αυτή (η μέθοδος αυτή παραμένει η ίδια ανεξαρτήτως του είδους της φερομόνης, αν είναι δηλαδή φερομόνη φωλιάς ή φερομόνη φαγητού). Για τη διαδικασία αυτή είναι υπεύθυνη η μέθοδος addPheromone().

Το μυρμήγκι εντοπίζει όλους τους γείτονές του, ώστε να γνωρίζει τις επιλογές του για κίνηση στο χώρο. Αυτό είναι απαραίτητο, αφού αν κάποια γειτονική του θέση είναι κατειλημμένη από ένα ορισμένο αριθμό μυρμηγκιών, τότε απαγορεύεται στο μυρμήγκι να κινηθεί προς την κατεύθυνση αυτή. Στην περίπτωση αυτή το μυρμήγκι είναι

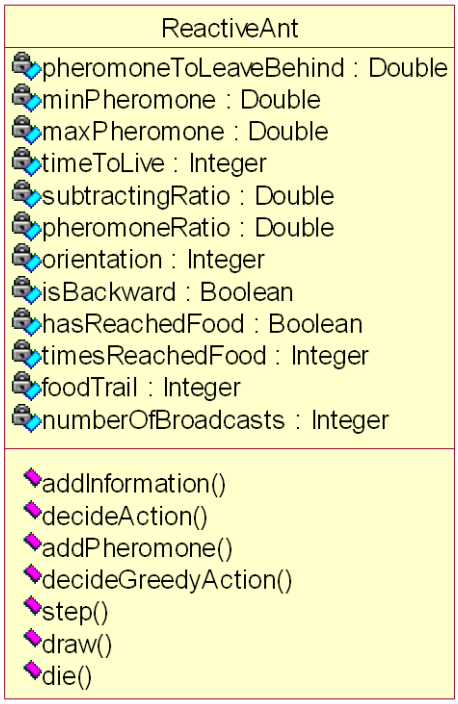
υποχρεωμένο να επιλέξει μια άλλη κατεύθυνση από τις γειτονικές κατευθύνσεις που του παρέχονται.

Η μέθοδος step() αποτελεί τη σημαντικότερη διαδικασία της κίνησης ενός μυρμηγκιού. Περιγράφει με ακρίβεια κάθε κίνηση του μυρμηγκιού και αποφασίζει αν αυτό μπορεί να συνεχίσει την κίνησή του ή αν έχει φτάσει η ώρα να «πεθάνει», δηλαδή να μην αποτελεί πια μέρος της προσομοίωσης. Αναλυτικότερα, η διαδικασία αυτή καθορίζει την ακριβώς επόμενη θέση του μυρμηγκιού με βάση την τρέχουσα θέση και κατάσταση του μυρμηγκιού. Εφόσον το μυρμήγκι μπορεί να κινηθεί στη θέση που αποφασίστηκε από τους κανόνες κίνησής του, τότε το μυρμήγκι κινείται κανονικά προς τη θέση που του επιβλήθηκε να κινηθεί. Αν όμως, για λόγους που προαναφέρθηκαν, δεν μπορεί να εκτελέσει την κίνηση που αποφασίστηκε, τότε επιλέγεται μια άλλη θέση που είναι επιτρεπτή για το μυρμήγκι. Για κάθε θέση που αποφασίζεται να κινηθεί το μυρμήγκι, γίνεται και ο ανάλογος υπολογισμός ποσότητας φερομονών που πρέπει να αφεθούν στο χώρο, ώστε να συνεχίζεται κανονικά η κίνηση όλων των μυρμηγκιών στο χώρο, στην προσπάθεια εύρεσης μονοπατιών που να οδηγούν τα μυρμήγκια από τη φωλιά στο φαγητό και αντίστροφα. Τέλος, η κίνηση των μυρμηγκιών καθορίζεται με βάση ένα σημαντικό κανόνα, αν αυτά είναι μυρμήγκια προώθησης ή οπισθοχώρησης, αν δηλαδή δεν έχουν φτάσει στο φαγητό ή αν έχουν φτάσει στο φαγητό και επιστρέφουν στη φωλιά αντίστοιχα.

Επίσης, η μέθοδος die() χρησιμοποιείται ώστε ένα μυρμήγκι να διαγράφεται από την προσομοίωση σε περιπτώσεις που ένα μυρμήγκι δεν πληρεί τα κριτήρια κίνησης στο χώρο ή έχει λήξει ο επιτρεπτός χρόνος παραμονής του στην προσομοίωση. Τέλος, η μέθοδος draw() χρησιμοποιείται ώστε κάθε φορά που ένα μυρμήγκι κινείται, να επανασχεδιάζεται στο χώρο της προσομοίωσης.

4.1.2.2 AntHocNet

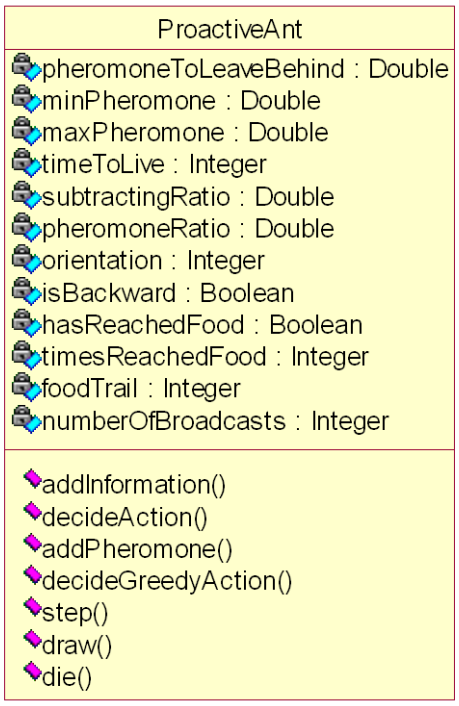
4.1.2.2.1 ReactiveAnt



Η κλάση αυτή περιγράφει την ακριβή κίνηση ενός αντιδραστικού μυρμηγκιού. Η κίνηση των αντιδραστικών μυρμηγκιών δε διαφέρει κατά πολύ από την κίνηση των μυρμηγκιών στο πρωτόκολλο AODV, αφού και αυτό χρησιμοποιεί τη φιλοσοφία των αντιδραστικών μυρμηγκιών. Η μόνη διαφορά μεταξύ των δύο ειδών μυρμηγκιών έγκειται στην εκτέλεση μεταδόσεων που εκτελούν τα μυρμήγκια στο AntHocNet. Συγκεκριμένα, ένα αντιδραστικό μυρμήγκι που χρησιμοποιεί το πρωτόκολλο AntHocNet έχει δύο επιλογές για να κινηθεί στο χώρο, ανάλογα με το αν υπάρχουν ή όχι πληροφορίες για τις φερομόνες στη θέση που βρίσκεται που θα το οδηγήσουν στον προορισμό. Αν υπάρχουν πληροφορίες τότε το μυρμήγκι επιλέγει την επόμενη του θέση με βάση μια πιθανότητα (που χρησιμοποιεί τον παράγοντα b_1), η εξίσωση της οποίας δίνεται στο υποκεφάλαιο *3.1.2* όπου περιγράφεται αναλυτικά ο αλγόριθμος AntHocNet. Από την άλλη, αν δεν υπάρχουν πληροφορίες για τις φερομόνες, τότε το μυρμήγκι εκτελεί μετάδοση δημιουργώντας μια νέα γενιά μυρμηγκιών που θα εξερευνήσουν το χώρο με σκοπό την

εύρεση του φαγητού. Ένα αντιδραστικό μυρμήγκι δημιουργεί μια νέα ομάδα μυρμηγκιών (NEW_ANTS_PER_BROADCAST) που θα ξεκινήσουν με τη σειρά τους την εξερεύνηση του χώρου για εύρεση του φαγητού. Ωστόσο, κάθε μυρμήγκι μπορεί να εκτελέσει μόνο ένα ορισμένο αριθμό μεταδόσεων (MAX_BROADCASTS) ώστε να μην παρατηρείται συμφόρηση και σύγκρουση μυρμηγκιών στο χώρο της προσομοίωσης. Αυτό μπορεί να μεταφερθεί και στα δίκτυα, όπου δεν μπορούμε να δημιουργούμε συνεχώς νέα πακέτα αφού σε κάποια στιγμή θα «φορτώσουμε» το δίκτυο με περισσότερες εργασίες από αυτές που μπορεί να εκτελέσει. Οι ενέργειες που περιγράφηκαν πιο πάνω υλοποιούνται στη μέθοδο decideAction(). Οι υπόλοιπες λειτουργίες της κλάσης περιγράφονται στο υποκεφάλαιο *4.1.2.1.1* όπου καλύπτεται η λεπτομερής λειτουργία της κλάσης *Ant* που αφορά το πρωτόκολλο AODV.

4.1.2.2.2 ProactiveAnt



Η κλάση αυτή περιγράφει την ακριβή κίνηση ενός δυναμικού μυρμηγκιού, που χρησιμοποιείται στον αλγόριθμο με σκοπό την εύρεση νέων μονοπατιών από τη φωλιά

στο φαγητό και αντίστροφα. Τα μυρμήγκια αυτά έχουν τη δυνατότητα μετάδοσης νέων μυρμηγκιών στην προσπάθεια τους να βρουν νέα και καλύτερα μονοπάτια από τα ήδη υπάρχοντα.

Στην αρχική φάση της κίνησης του μυρμηγκιού, τοποθετούνται οι αρχικές ποσότητες, τόσο φερομονών φωλιάς, όσο και φερομονών φαγητού, καθώς και οι αρχικές συντεταγμένες του μυρμηγκιού στο χώρο. Όπως και προηγουμένως, κάθε μυρμήγκι παίρνει τις τιμές που του αναθέτονται ώστε να μπορεί να κινηθεί στο χώρο, pheromoneToLeaveBehind, minPheromone, maxPheromone, timeToLive και orientation. Επίσης, κάθε μυρμήγκι στην αρχή της προσομοίωσης θεωρείται μυρμήγκι προώθησης (isBackward = FALSE) και σημειώνεται ότι δεν έχει βρει ακόμα το φαγητό (hasReachedFood = FALSE). Έτσι, ο αριθμός των φορών που το μυρμήγκι έχει βρει το φαγητό ορίζεται στο μηδέν (timesReachedFood = 0) και αρχικά το καλύτερο μονοπάτι που έχει βρεθεί είναι το χειρότερο που καθορίζεται από την κλάση AntsForage. Η ανάθεση αυτή των τιμών γίνεται από τη μέθοδο addInformation().

Με βάση μια πιθανότητα που καθορίζεται σαν παράμετρος (BROADCAST_PROBABILITY), κάθε μυρμήγκι ορίζει μια μετάδοση μυρμηγκιών (NEW_ANTS_PER_BROADCAST) που θα αναζητήσουν νέα μονοπάτια. Εντούτοις, κάθε μυρμήγκι δικαιούται να εκτελέσει μόνο ένα ορισμένο αριθμό μεταδόσεων (MAX_BROADCASTS), ένας αριθμός που καθορίζεται σαν παράμετρος, ώστε να μην υπάρξει σε καμιά στιγμή της προσομοίωσης ένας ανεξέλεγκτος αριθμός μυρμηγκιών που θα κινούνται στο χώρο.

Το κάθε μυρμήγκι έχει οκτώ επιλογές κίνησης στο χώρο, βρισκόμενο σε κάποια από τις θέσεις του. Σύμφωνα με την επιλογή που έγινε προηγουμένως, η συνάρτηση αυτή καθορίζει την ακριβή κίνηση του μυρμηγκιού στο χώρο.

Με βάση και πάλι τη θέση που βρίσκεται το μυρμήγκι τη στιγμή της προσομοίωσης, επιλέγεται μια μέθοδος εμφάνισης των φερομονών. Συγκεκριμένα, σύμφωνα με τη θέση

του, το μυρμήγκι υπολογίζει το ποσό φερομονών που πρέπει να αφήσει στη θέση όπου βρίσκεται και ακολούθως αφήνει τη φερομόνη αυτή (η διαδικασία αυτή παραμένει η ίδια ανεξαρτήτως του είδους της φερομόνης, αν είναι δηλαδή φερομόνη φωλιάς ή φερομόνη φαγητού).

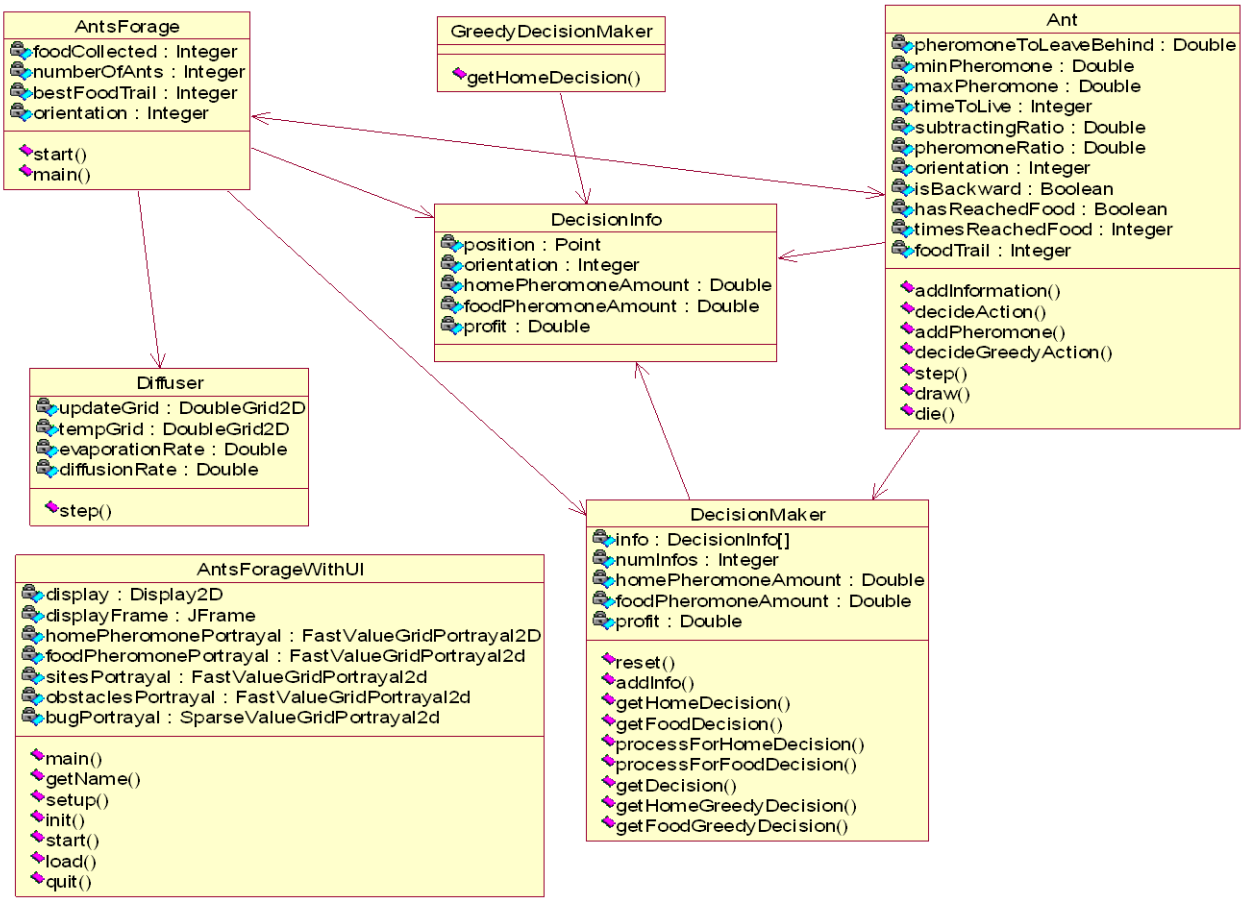
Το μυρμήγκι εντοπίζει όλους τους γείτονές του, ώστε να γνωρίζει τις επιλογές του για κίνηση στο χώρο. Αυτό είναι απαραίτητο, αφού αν κάποια γειτονική του θέση είναι κατειλημμένη από ένα ορισμένο αριθμό μυρμηγκιών, τότε απαγορεύεται στο μυρμήγκι να κινηθεί προς την κατεύθυνση αυτή. Στην περίπτωση αυτή το μυρμήγκια είναι υποχρεωμένο να επιλέξει μια άλλη κατεύθυνση από τις γειτονικές κατευθύνσεις που του παρέχονται. Για να επιτευχθούν τα παραπάνω γίνεται χρήση της μεθόδου decideAction().

Η μέθοδος step() αποτελεί τη σημαντικότερη διαδικασία της κίνησης ενός μυρμηγκιού. Περιγράφει με ακρίβεια κάθε κίνηση του μυρμηγκιού και αποφασίζει αν αυτό μπορεί να συνεχίσει την κίνησή του ή αν έχει φτάσει η ώρα να «πεθάνει» (μέθοδος die()), δηλαδή να μην αποτελεί πια μέρος της προσομοίωσης. Επιπλέον, η διαδικασία αυτή αποφασίζει αν πρέπει να εκτελεστούν οι μεταδόσεις που επιτρέπεται να εκτελέσει ένα δυναμικό μυρμήγκι, στην προσπάθεια εύρεσης νέων και καλύτερων μονοπατιών από τη φωλιά προς το φαγητό και αντίστροφα. Έτσι, κάθε δυναμικό μυρμήγκι, με βάση μια πιθανότητα, μπορεί να εκτελέσει μετάδοση νέων μυρμηγκιών που θα εξερευνήσουν το χώρο στην προσπάθεια τους να χτίσουν νέα και καλύτερα μονοπάτια. Η ενέργεια αυτή αποτελεί και τη διαφορά μεταξύ δυναμικών και αντιδραστικών μυρμηγκιών, αφού τα δυναμικά μυρμήγκια έχουν σε γενικό βαθμό την ίδια λειτουργία με τα αντιδραστικά μυρμήγκια.

Σημείωση: Οι μέθοδοι και μεταβλητές που δεν αναφέρονται στο παρόν υποκεφάλαιο εκτελούν τις ίδιες λειτουργίες με την κλάση *Ant* και εξηγούνται στο υποκεφάλαιο *4.1.2.1.1*.

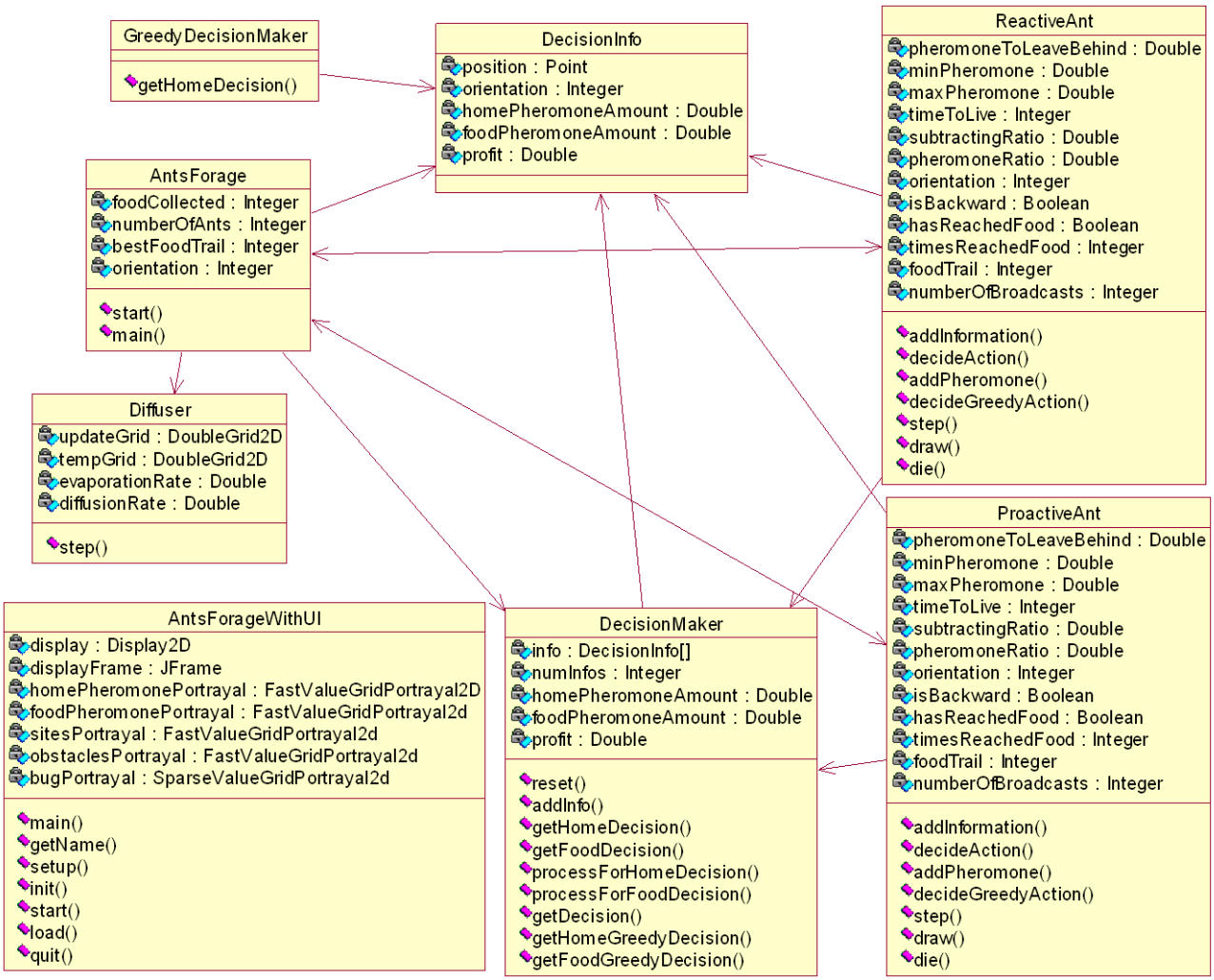
4.1.3 Συσχέτιση κλάσεων AODV και AntHocNet

4.1.3.1 AODV



Στο σχήμα αυτό βλέπουμε τον τρόπο με τον οποίο επικοινωνούν οι κλάσεις μεταξύ τους. Συγκεκριμένα, η κλάση AntsForage δημιουργεί αντικείμενα τύπου DecisionMaker, DeicisionInfo, Ant και Diffuser. Η κλάση Diffuser δεν επικοινωνεί με καμιά άλλη κλάση από τις 7 κλάσεις του προγράμματος. Η κλάση DecisionInfo καλείται από άλλες κλάσεις αλλά η ίδια δεν καλεί καμιά άλλη. Η κλάση DecisionMaker δημιουργεί αντικείμενα τύπου DecisionInfo. Η κλάση Ant δημιουργεί αντικείμενα τύπου AntsForage, DecisionInfo και DecisionMaker. Επίσης, η κλάση GreedyDecisionMaker δημιουργεί αντικείμενα τύπου DecisionInfo αφού μόνο με αυτή την κλάση επικοινωνεί. Τέλος, η κλάση AntsForageWithUI αποτελεί την κύρια κλάση του προγράμματος, αφού από αυτήν ξεκινά η προσομοίωση και σε αυτή γίνονται οι απαραίτητες αρχικοποιήσεις του χώρου.

4.1.3.2 AntHocNet



Στο σχήμα αυτό βλέπουμε τον τρόπο με τον οποίο επικοινωνούν οι κλάσεις μεταξύ τους. Συγκεκριμένα, η κλάση AntsForage δημιουργεί αντικείμενα τύπου DecisionMaker, DeicisionInfo, Ant και Diffuser. Η κλάση Diffuser δεν επικοινωνεί με καμιά άλλη κλάση από τις 7 κλάσεις του προγράμματος. Η κλάση DecisionInfo καλείται από άλλες κλάσεις αλλά η ίδια δεν καλεί καμιά άλλη. Η κλάση DecisionMaker δημιουργεί αντικείμενα τύπου DecisionInfo. Οι κλάσεις ReactiveAnt και ProactiveAnt δημιουργούν αντικείμενα τύπου AntsForage, DecisionInfo και DecisionMaker. Η κλάση GreedyDecisionMaker δημιουργεί μόνο αντικείμενα τύπου DecisionInfo. Τέλος, η κλάση AntsForageWithUI αποτελεί την κύρια κλάση του προγράμματος, αφού από αυτήν ξεκινά η προσομοίωση και σε αυτή γίνονται οι απαραίτητες αρχικοποιήσεις του χώρου.

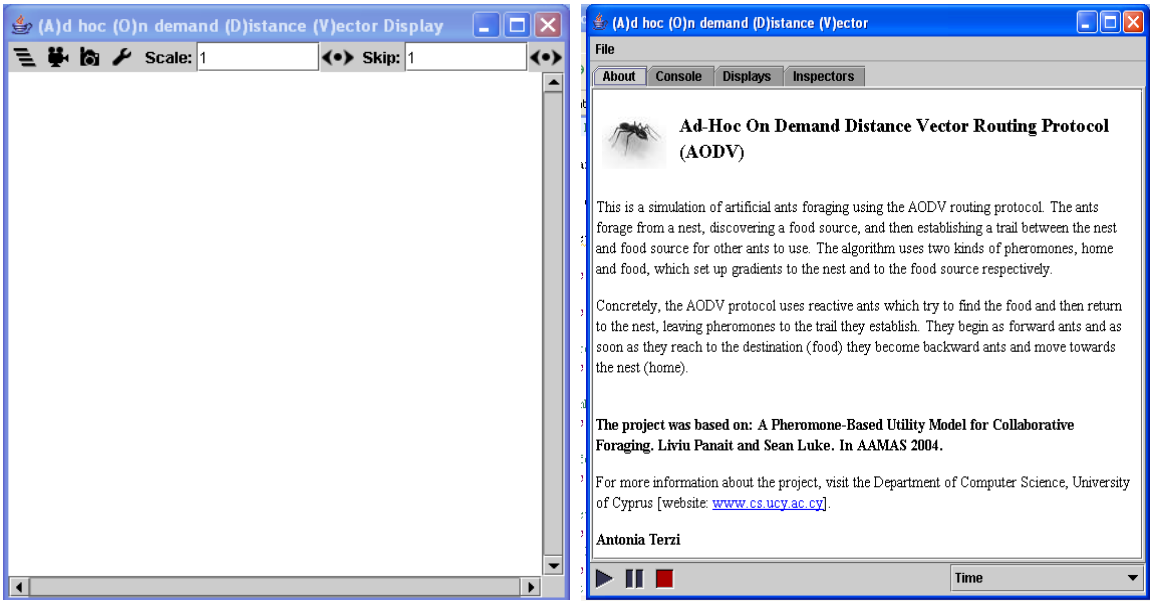
ΚΕΦΑΛΑΙΟ 5:

ΑΠΟΤΕΛΕΣΜΑΤΑ ΠΡΟΣΟΜΟΙΩΣΕΩΝ

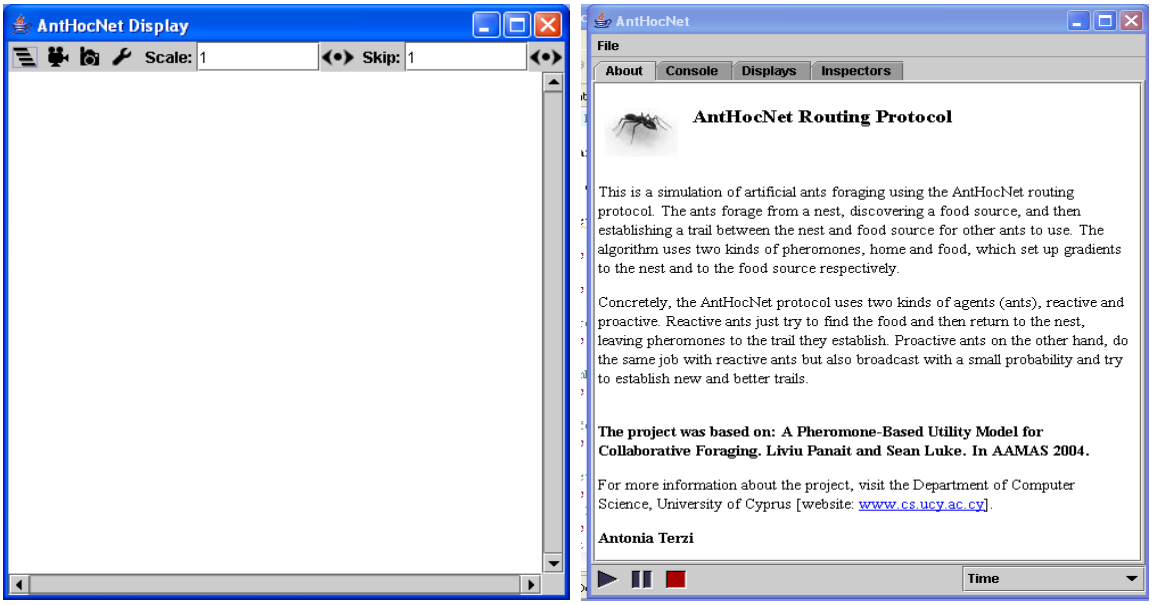
Στο κεφάλαιο αυτό παραθέτουμε κάποιες προσομοιώσεις που πιστεύουμε ότι αντιπροσωπεύουν τη βασικότερη λειτουργία των δύο αλγορίθμων, τόσο για τον καθέναν ξεχωριστά όσο και συγκριτικά μεταξύ τους. Στο υποκεφάλαιο *5.*1 δείχνουμε σύντομα το περιβάλλον της προσομοίωσης που δημιουργείται κατά την εκτέλεση του προγράμματος στο εργαλείο MASON. Ακολούθως, παρουσιάζουμε τα αποτελέσματα κάποιων προσομοιώσεων με μεταβαλλόμενες παραμέτρους οι οποίες φαίνεται να επηρεάζουν την όλη λειτουργία κάθε αλγορίθμου. Η συμπεριφορά του AODV φαίνεται στο υποκεφάλαιο *5.2*, ενώ του AntHocNet στο υποκεφάλαιο *5.3*. Έπειτα, μεταβάλλουμε κάποιες παραμέτρους ώστε να συγκρίνουμε τους δύο αλγόριθμους, AODV και AntHocNet, μεταξύ τους. Τα αποτελέσματα της σύγκρισης αυτής παρατίθενται στο υποκεφάλαιο *5.4*.

5.1 Περιβάλλον προσομοίωσης στο MASON

Σε κάθε περίπτωση εκτέλεσης αλγορίθμου προσφέρεται η δυνατότητα επιλογής για αποθήκευση ενός στιγμιότυπου της προσομοίωσης για οποιαδήποτε χρονικής στιγμή. Οι οθόνη για κάθε εκτέλεση των δύο αλγορίθμων είναι όπως φαίνεται πιο κάτω:



Το περιβάλλον αυτό εμφανίζεται όταν εκτελείται ο αλγόριθμος AODV.



Το περιβάλλον αυτό εμφανίζεται όταν εκτελείται ο αλγόριθμος AntHocNet.

5.2 Παραδείγματα προσομοιώσεων αλγορίθμου AODV

Σημειώνουμε ότι για κάθε προσομοίωση, με γαλάζιο χρώμα φαίνεται το αρχικό φόντο και με πράσινο χρώμα το φόντο που δημιουργείται όταν τα μυρμήγκια κινούνται στο χώρο και αφήνουν φερομόνες. Επίσης, με μπλε χρώμα σχηματίζεται το μονοπάτι που ακολουθεί ένα μυρμήγκι όταν φτάσει στο φαγητό και επιστρέφει στη φωλιά. Το κόκκινο σημείο υποδηλώνει το χώρο εκκίνησης των μυρμηγκιών, επομένως θεωρούμε ότι είναι η φωλιά, ενώ το καφέ σημείο δείχνει το χώρο συγκέντρωσης του φαγητού. Τέλος, τα μυρμήγκια προώθησης απεικονίζονται με μαύρο χρώμα, ενώ τα μυρμήγκια οπισθοχώρησης με άσπρο χρώμα. Για πρακτικούς λόγους, δεν παρουσιάζουμε τις εικόνες για όλες τις εκτελέσεις κάθε εκδοχής, αλλά δείχνουμε κάποιες αντιπροσωπευτικές εκτελέσεις.

5.2.1 Σενάριο προσομοίωσης αλγορίθμου AODV

Σταθερές Παράμετροι	
HOME_XMIN	25
HOME_XMAX	26
HOME_YMIN	25
HOME_YMAX	26
FOOD_XMIN	5
FOOD_XMAX	6
FOOD_YMIN	43
FOOD_YMAX	44
MIN_PHEROMONE	0
MAX_PHEROMONE	1000
PHEROMONE_TO_LEAVE_BEHIND	1

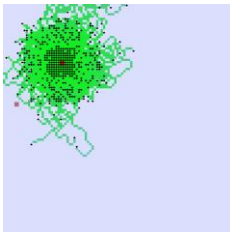
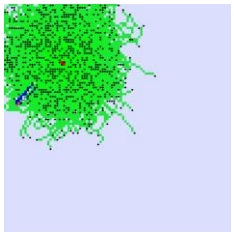
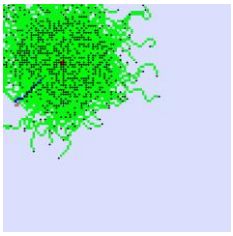
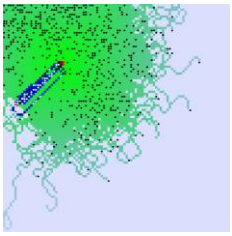
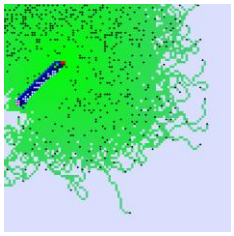
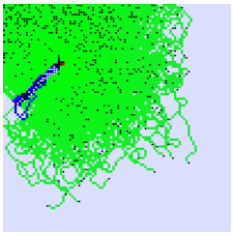
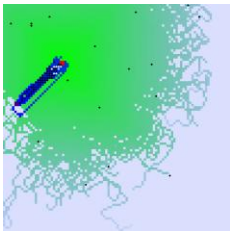
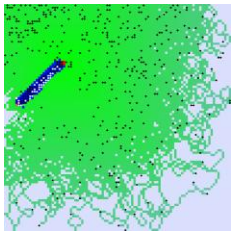
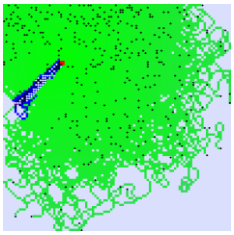
Πίνακας 5.1: Σταθερές Παράμετροι αλγορίθμου AODV

Στην προσομοίωση αυτή τοποθετήσαμε τη φωλιά δεξιότερα και πάνω από το φαγητό που τοποθετήθηκε αριστερά και πιο κάτω. Κάθε μυρμήγκι επιτρέπεται να αφήσει ένα ποσό φερομονών από 0 μέχρι 1000, αφήνοντας 2 κάθε φορά που κινείται σε νέα θέση. Στη συνέχεια, ακολουθούν 3 εκδοχές του σεναρίου στις οποίες μεταβάλλουμε τις παραμέτρους ώστε να παρατηρήσουμε τη συμπεριφορά των μυρμηγκιών του αλγορίθμου σε κάθε διαφοροποίηση των παραμέτρων.

Εκδοχή Α

Μεταβλητές Παράμετροι					
Όνομα Παραμέτρου	**1η εκτέλεση**	**2η εκτέλεση**	**3η εκτέλεση**	**4η εκτέλεση**	**5η εκτέλεση**
TIME_TO_LIVE	*100*	*150*	*200*	*250*	*500*
NEW_ANTS_PER_BROADCAST	2	2	2	2	2
MAX_ANTS	1000	1000	1000	1000	1000

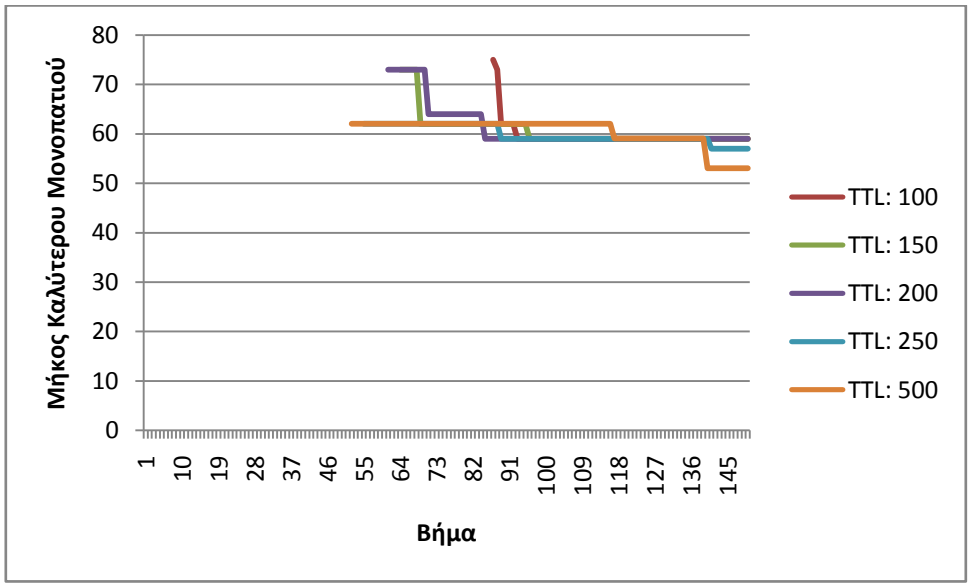
Πίνακας 5.2: Μεταβλητές Παράμετροι Εκδοχής Α

	1η εκτέλεση	3η εκτέλεση	5η εκτέλεση
Βήμα 50			
Βήμα 100			
Βήμα 150			

Στην εκδοχή αυτή του σεναρίου, αποφασίσαμε να κρατήσουμε σταθερά τα νέα μυρμήγκια σε κάθε μετάδοση, καθώς και το μέγιστο αριθμό μυρμηγκιών που δύναται να έχει η προσομοίωση και να μεταβάλλουμε το «χρόνο ζωής» των μυρμηγκιών. Αρχικά ορίζουμε τα μυρμήγκια να ζουν για 70 βήματα και ακολούθως για 100, 150, 200 και 150 βήματα σε κάθε αντίστοιχη εκτέλεση της προσομοίωσης. Αυτό γίνεται ώστε να παρατηρήσουμε τον τρόπο με τον οποίο θα λειτουργήσουν τα μυρμήγκια όταν μεταβάλλεται ο χρόνος ζωής τους. Είδαμε λοιπόν ότι αυξάνοντας κάθε φορά το χρόνο παραμονής των μυρμηγκιών στην προσομοίωση, αυξάνουμε και την πιθανότητα εύρεσης του φαγητού από μεγαλύτερο αριθμό μυρμηγκιών, αφού αφήνουμε τα μυρμήγκια που εξερευνούν το χώρο για πολλή ώρα να συνεχίσουν να ψάχνουν. Αντίθετα, στην πρώτη

εκτέλεση της προσομοίωσης, τα μυρμήγκια εξαφανίζονται μετά από 100 βήματα, μειώνοντας έτσι τον αριθμό των μυρμηγκιών που ψάχνουν ανεπιτυχώς στο χώρο.

Διαδικασία εύρεσης μονοπατιού

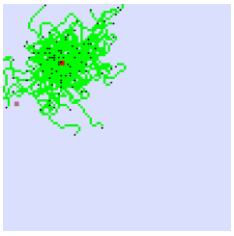
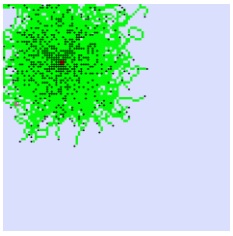
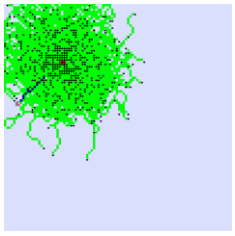
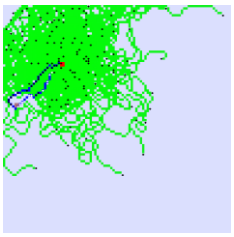
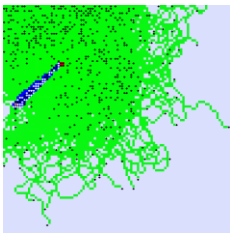
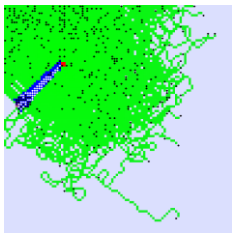
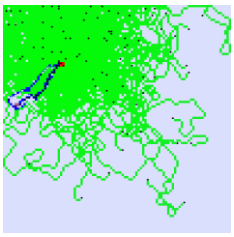
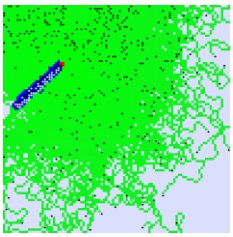
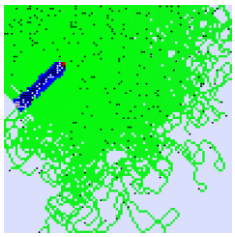


Παρατηρούμε πως όσο αυξάνουμε τον αριθμό των βημάτων παραμονής ενός μυρμηγκιού στην προσομοίωση, μειώνονται ο χρόνος εύρεσης ενός μονοπατιού και το μήκος του καλύτερου μονοπατιού. Αυτό συμβαίνει γιατί αφήνουμε τα μυρμήγκια να εξερευνήσουν το χώρο για περισσότερο αριθμό βημάτων, μεγαλώνοντας έτσι την πιθανότητα εύρεσης ενός καλύτερου μονοπατιού σε λιγότερο χρόνο.

Εκδοχή Β

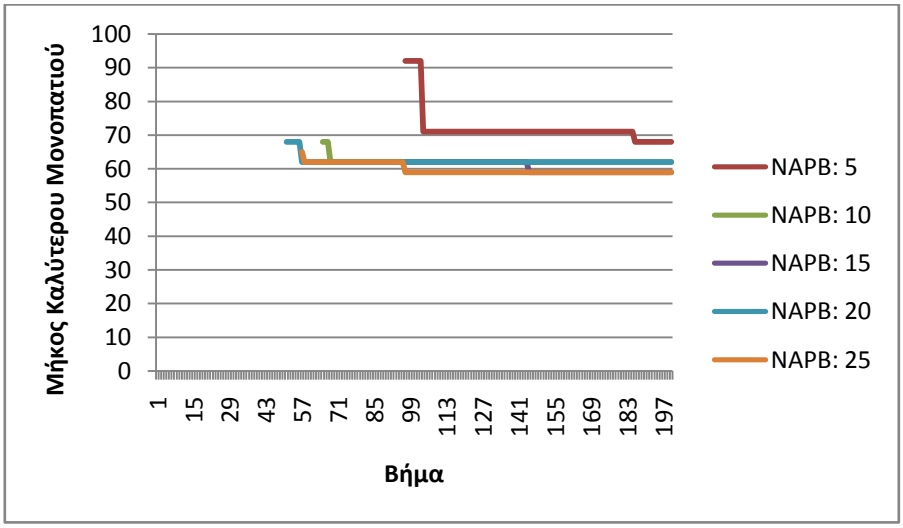
Μεταβλητές Παράμετροι					
Όνομα Παραμέτρου	**1η εκτέλεση**	**2η εκτέλεση**	**3η εκτέλεση**	**4η εκτέλεση**	**5η εκτέλεση**
TIME_TO_LIVE	500	500	500	500	500
NEW_ANTS_PER_BROADCAST	*2*	*3*	*5*	*8*	*10*
MAX_ANTS	1000	1000	1000	1000	1000

Πίνακας 5.3: Μεταβλητές Παράμετροι Εκδοχής Β

	1η εκτέλεση	3η εκτέλεση	5η εκτέλεση
Βήμα 50			
Βήμα 100			
Βήμα 150			

Στην δεύτερη εκδοχή του σεναρίου, αποφασίσαμε να κρατήσουμε σταθερά το χρόνο παραμονής των μυρμηγκιών στην προσομοίωση, καθώς και το μέγιστο αριθμό μυρμηγκιών που δύναται να έχει η προσομοίωση και να μεταβάλλουμε τον αριθμό των νέων μυρμηγκιών σε κάθε μετάδοση. Αρχικά ορίζουμε τον αριθμό αυτό στο 2 και στην τελευταία εκδοχή στο 10 ώστε να παρατηρήσουμε αν η σταδιακή δημιουργία περισσότερων μυρμηγκιών θα οδηγήσει σε νέα αποτελέσματα. Είδαμε λοιπόν ότι τα μυρμήγκια στην πρώτη εκτέλεση βρίσκουν το φαγητό σε κάποια στιγμή, αλλά αυτή η στιγμή είναι πολύ αργότερα από αυτήν τόσο στην εκδοχή με 5, όσο και στην εκδοχή με 10 νέα μυρμήγκια. Αυτό συμβαίνει γιατί δημιουργούμε κάθε φορά περισσότερα μυρμήγκια σε κάθε νέα μετάδοση, οπότε οι φερομόνες που βρίσκονται στο έδαφος γίνονται όλο και περισσότερες, βοηθώντας όλα τα μέλη του σμήνους να βρουν γρηγορότερα το φαγητό και να επιστρέψουν στη φωλιά.

Διαδικασία εύρεσης μονοπατιού

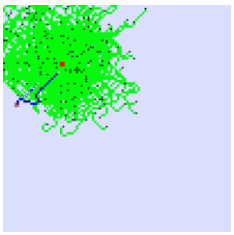
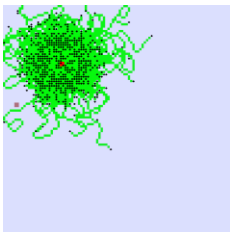
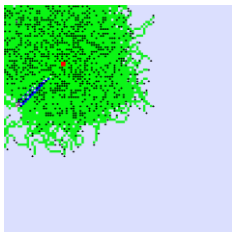
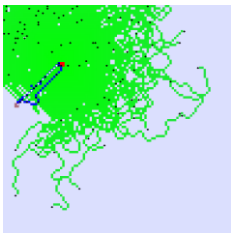
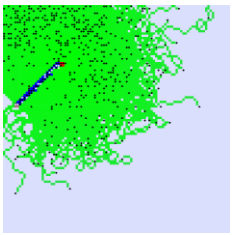
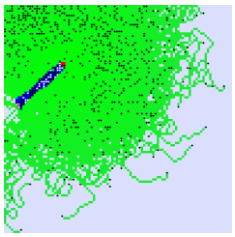
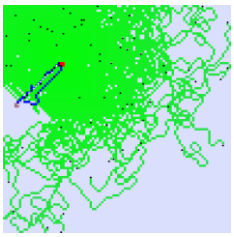
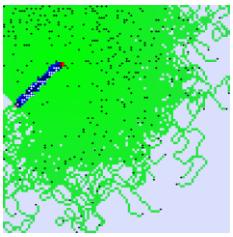
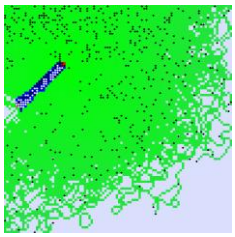


Βλέπουμε πως αυξανόμενου του αριθμού των νέων μυρμηγκιών σε κάθε μετάδοση, αυξάνεται και η επιτυχία των μυρμηγκιών. Συγκεκριμένα, βλέπουμε πως όταν ο αριθμός των νέων μυρμηγκιών είναι 5, τα μυρμήγκια βρίσκουν το μονοπάτι στο βήμα 96, όταν ο αριθμός τους είναι 10 στο βήμα 64, όταν είναι 15 στο βήμα 68, όταν είναι 20 στο βήμα 50 και όταν είναι 25 στο βήμα 56. Επίσης, βλέπουμε πως το καλύτερο μονοπάτι το έχουν βρει τα μυρμήγκια που αυξάνονται κατά 25, όπως αναμέναμε.

Εκδοχή Γ

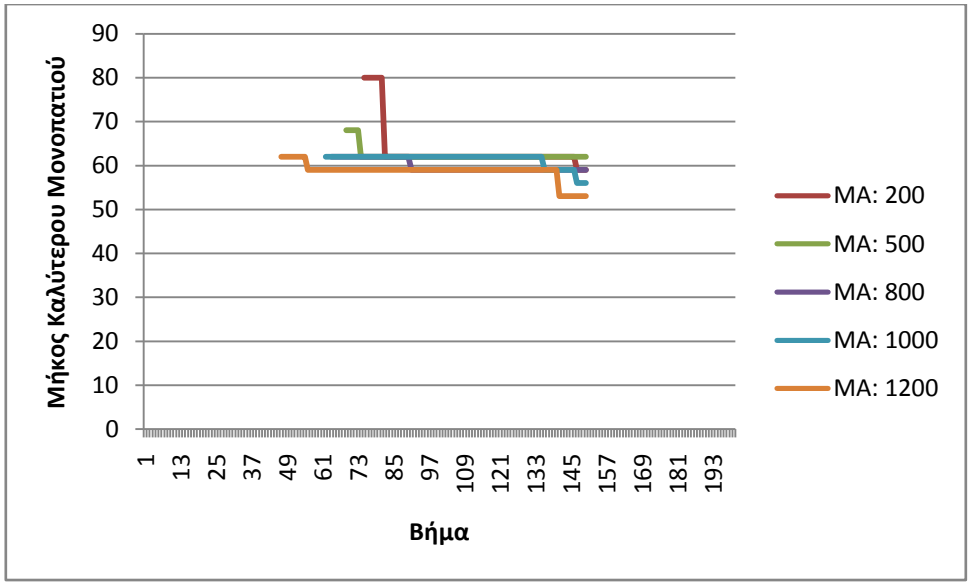
Μεταβλητές Παράμετροι					
Όνομα Παραμέτρου	**1η εκτέλεση**	**2η εκτέλεση**	**3η εκτέλεση**	**4η εκτέλεση**	**5η εκτέλεση**
TIME_TO_LIVE	500	500	500	500	500
NEW_ANTS_PER_BROADCAST	2	2	2	2	2
MAX_ANTS	*200*	*500*	*800*	*1000*	*1200*

Πίνακας 5.4: Μεταβλητές Παράμετροι Εκδοχής Γ

	1η εκτέλεση	3η εκτέλεση	5η εκτέλεση
Βήμα 50			
Βήμα 100			
Βήμα 150			

Σε αυτή την τρίτη εκδοχή του σεναρίου, κρατούμε σταθερά το χρόνο παραμονής των μυρμηγκιών στην προσομοίωση, καθώς και τον αριθμό των νέων μυρμηγκιών σε κάθε μετάδοση και μεταβάλλουμε το μέγιστο αριθμό μυρμηγκιών που δύναται να έχει η προσομοίωση. Αρχικά ορίζουμε τον αριθμό αυτό στο 200 και ακολούθως στο 500, 800, 1000 και 1200 ώστε να παρατηρήσουμε αν η δημιουργία περισσότερων μυρμηγκιών κάθε φορά θα οδηγήσει σε διαφορετικά αποτελέσματα. Τα μυρμήγκια στην πρώτη εκτέλεση σε κάποιο σημείο της προσομοίωσης σταματούν να πολλαπλασιάζονται, ενώ στο ίδιο σημείο για την τελευταία εκτέλεση συνεχίζουν επειδή ορίζουμε το σύνολό τους να είναι μεγαλύτερο. Έτσι, με τη δημιουργία περισσότερων μυρμηγκιών είναι αναμενόμενο να υπάρχει μεγαλύτερη πιθανότητα να βρεθεί το φαγητό κάποια στιγμή και η στιγμή αυτή να έρθει γρηγορότερα.

Διαδικασία εύρεσης μονοπατιού



Παρατηρούμε πως, όπως αναμενόταν, αυξανόμενου του αριθμού των μυρμηγκιών που επιτρέπουμε να λαμβάνουν μέρος στην προσομοίωση, μειώνονται τόσο ο χρόνος που χρειάζονται τα μυρμήγκια για να βρουν ένα μονοπάτια, καθώς και το μήκος του καλύτερου μονοπατιού που βρίσκουν.

5.3 Παραδείγματα προσομοιώσεων αλγορίθμου AntHocNet

Σημειώνουμε ότι για κάθε προσομοίωση, με γαλάζιο χρώμα φαίνεται το αρχικό φόντο και με πράσινο χρώμα το φόντο που δημιουργείται όταν τα μυρμήγκια κινούνται στο χώρο και αφήνουν φερομόνες. Με μπλε χρώμα σχηματίζεται το μονοπάτι που ακολουθεί ένα μυρμήγκι όταν φτάσει στο φαγητό και επιστρέφει στη φωλιά. Το κόκκινο σημείο είναι η φωλιά, ενώ το καφέ σημείο απεικονίζει το φαγητό. Τέλος, τα αντιδραστικά μυρμήγκια προώθησης απεικονίζονται με μαύρο χρώμα, τα αντιδραστικά μυρμήγκια οπισθοχώρησης με άσπρο χρώμα, τα δυναμικά μυρμήγκια προώθησης απεικονίζονται με κόκκινο χρώμα και τα δυναμικά μυρμήγκια οπισθοχώρησης με γαλάζιο χρώμα.

5.3.1 Σενάριο προσομοίωσης αλγορίθμου AntHocNet

Σταθερές Παράμετροι	
AntsForage	
HOME_XMIN	25
HOME_XMAX	26
HOME_YMIN	25
HOME_YMAX	26
FOOD_XMIN	55
FOOD_XMAX	56
FOOD_YMIN	63
FOOD_YMAX	64
MIN_PHEROMONE	0
MAX_PHEROMONE	1000
PHEROMONE_TO_LEAVE_BEHIND	1
TIME_TO_LIVE	300

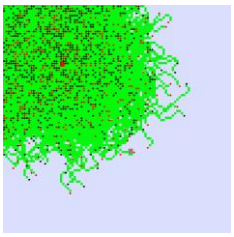
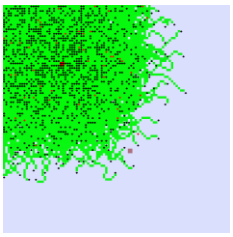
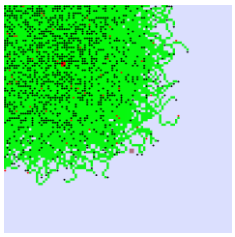
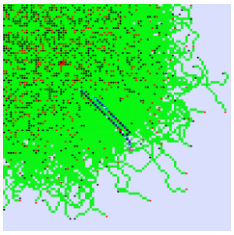
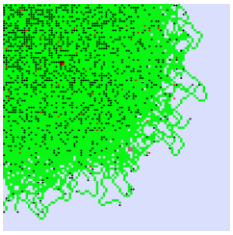
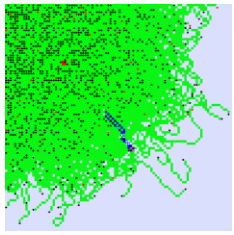
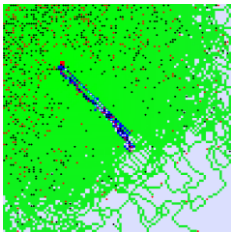
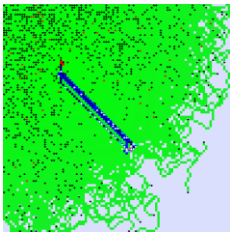
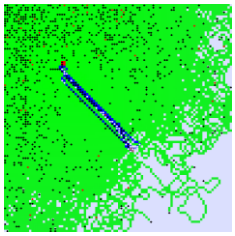
Πίνακας 5.5: Σταθερές Παράμετροι αλγορίθμου AntHocNet

Στην προσομοίωση αυτή τοποθετήσαμε τη φωλιά πάνω και αριστερότερα από το φαγητό που τοποθετήθηκε περίπου κάτω από το κέντρο του χώρου και δεξιά. Κάθε μυρμήγκι «ζει» για 300 βήματα. Κάθε μυρμήγκι επιτρέπεται να αφήσει ένα ποσό φερομονών από 0 μέχρι 1000, αφήνοντας 1 κάθε φορά που κινείται σε νέα θέση.

Εκδοχή Α

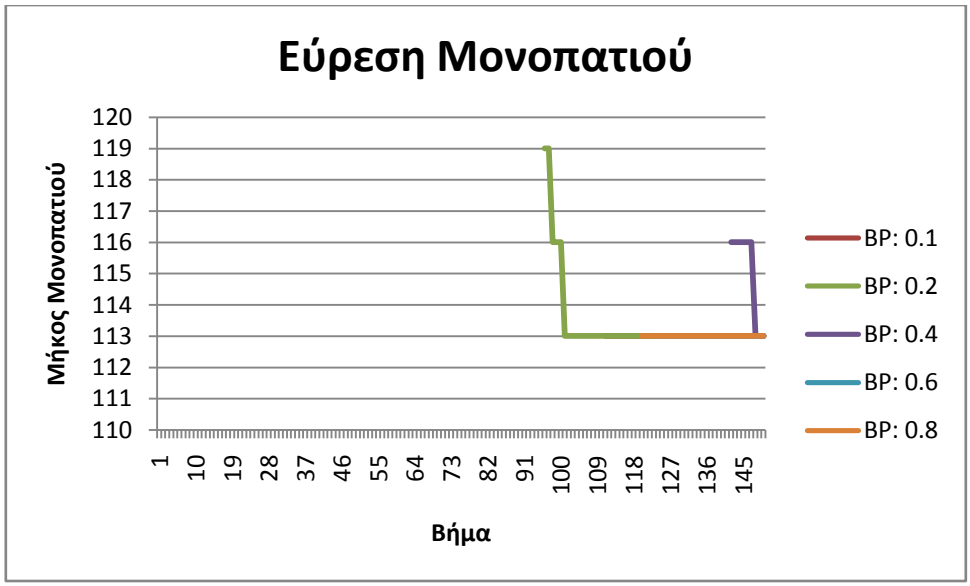
Μεταβλητές Παράμετροι					
Όνομα Παραμέτρου	**1η εκτέλεση**	**2η εκτέλεση**	**3η εκτέλεση**	**4η εκτέλεση**	**5η εκτέλεση**
AntsForage					
MAX_ANTS	1500	1500	1500	1500	1500
ProactiveAnt					
BROADCAST_PROBABILITY	*0.1*	*0.2*	*0.4*	*0.6*	*0.8*
NEW_ANTS_PER_BROADCAST	5	5	5	5	5
MAX_BROADCASTS	2	2	2	2	2
ReactiveAnt					
NEW_ANTS_PER_BROADCAST	5	5	5	5	5
MAX_BROADCASTS	2	2	2	2	2
b1	4	4	4	4	4

Πίνακας 5.6: Μεταβλητές Παράμετροι Εκδοχής Α

	1η εκτέλεση	3η εκτέλεση	5η εκτέλεση
Βήμα 70			
Βήμα 100			
Βήμα 150			

Στην εκδοχή αυτή του σεναρίου, μεταβάλλουμε την πιθανότητα που έχει κάθε δυναμικό μυρμήγκι για να εκτελέσει μετάδοση. Αρχικά του δίνουμε 90% πιθανότητα για να εκτελέσει, ενώ στο τέλος περιορίζουμε την πιθανότητα αυτή στο 20%. Με τη μεταβολή αυτή της πιθανότητας βλέπουμε κάποιες αλλαγές όσον αφορά τη χρονική στιγμή που τα μυρμήγκια εντοπίζουν το πρώτο μονοπάτι. Παρατηρούμε επίσης ότι δεν υπάρχουν μεγάλες διαφορές λόγω του ότι αν και περιορίζουμε τα δυναμικά μυρμήγκια, εντούτοις τα αντιδραστικά μυρμήγκια συνεχίζουν να ψάχνουν χωρίς κανένα επιπλέον περιορισμό.

Διαδικασία εύρεσης μονοπατιού

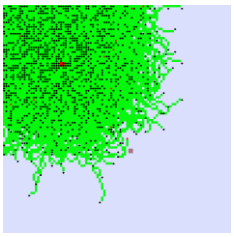
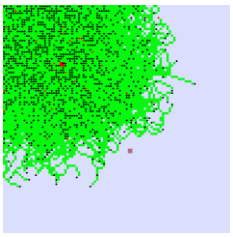
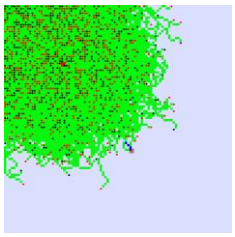
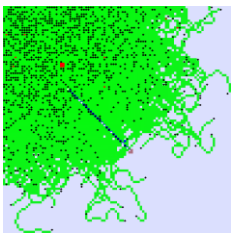
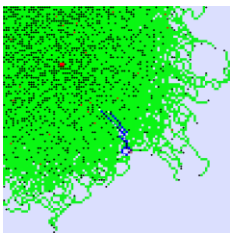
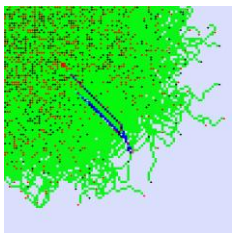
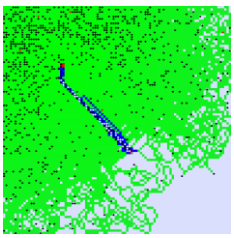
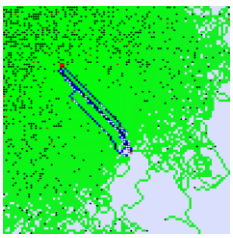
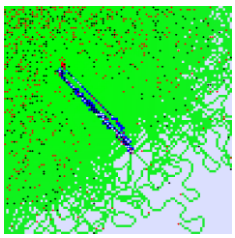


Όπως βλέπουμε, πιο γρήγορα εντοπίζεται το μονοπάτι όταν η πιθανότητα για μετάδοση των δυναμικών μυρμηγκιών είναι 80% (BP: 0.2). Δεν παρατηρείται ανάλογη μεταβολή της πιθανότητας με το μήκος μονοπατιού και το χρόνο για το λόγο που αναφέραμε προηγουμένως, ότι δηλαδή η αλλαγή στην πιθανότητα επηρεάζει μόνο τα δυναμικά μυρμήγκια, ενώ τα αντιδραστικά μυρμήγκια συνεχίζουν κανονικά να εξερευνούν το χώρο.

Εκδοχή Β

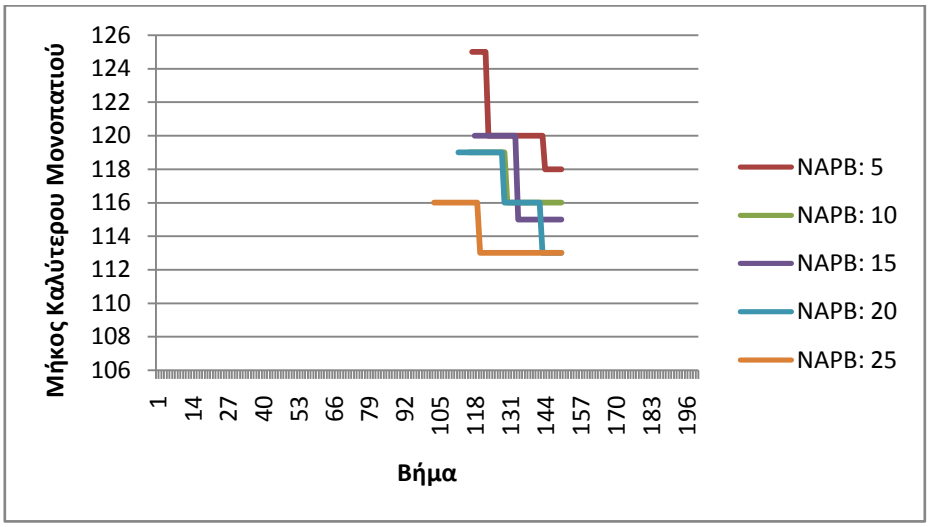
Μεταβλητές Παράμετροι					
Όνομα Παραμέτρου	**1η εκτέλεση**	**2η εκτέλεση**	**3η εκτέλεση**	**4η εκτέλεση**	**5η εκτέλεση**
AntsForage					
MAX_ANTS	1500	1500	1500	1500	1500
ProactiveAnt					
BROADCAST_PROBABILITY	0.5	0.5	0.5	0.5	0.5
NEW_ANTS_PER_BROADCAST	*5*	*10*	*15*	*20*	*25*
MAX_BROADCASTS	2	2	2	2	2
ReactiveAnt					
NEW_ANTS_PER_BROADCAST	5	5	5	5	5
MAX_BROADCASTS	2	2	2	2	2
b1	4	4	4	4	4

Πίνακας 5.7: Μεταβλητές Παράμετροι Εκδοχής Β

	1η εκτέλεση	3η εκτέλεση	5η εκτέλεση
Βήμα 70			
Βήμα 100			
Βήμα 150			

Στην εκδοχή αυτή του σεναρίου, μεταβάλλουμε το νέο αριθμό δυναμικών μυρμηγκιών σε κάθε μετάδοση που εκτελεί ένα τέτοιο μυρμήγκι. Αρχικά ορίζουμε τον αριθμό αυτό στο 5 και σταδιακά αυξάνουμε τον αριθμό σε 10, 15, 20 και 25. Με τη μεταβολή αυτή του αριθμού νέων δυναμικών μυρμηγκιών βλέπουμε κάποιες αλλαγές όσον αφορά τη χρονική στιγμή που τα μυρμήγκια εντοπίζουν το πρώτο μονοπάτι. Παρατηρούμε επίσης ότι είναι πιθανότερο τα μυρμήγκια της προσομοίωσης να βρουν ένα καλύτερο μονοπάτι όταν σε κάθε μετάδοση δημιουργούμε περισσότερα μυρμήγκια.

Διαδικασία εύρεσης μονοπατιού

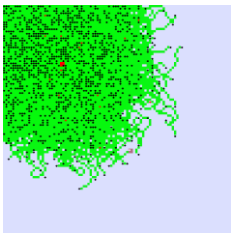
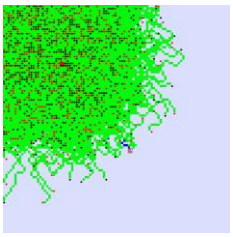
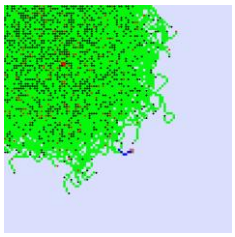
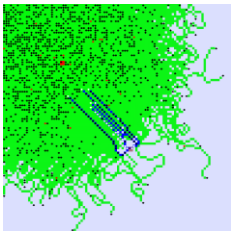
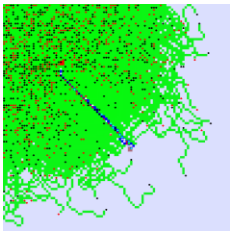
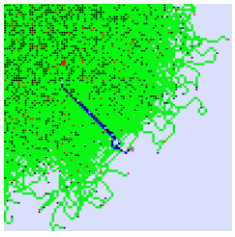
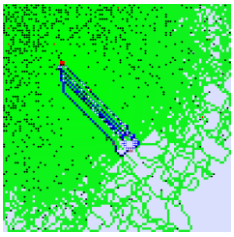
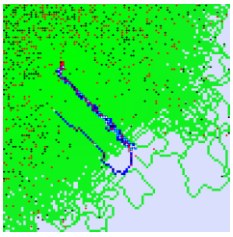
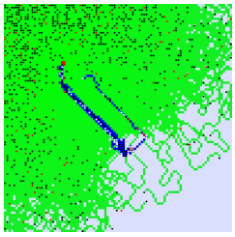


Παρατηρούμε πως με την αύξηση του αριθμού των νέων δυναμικών μυρμηγκιών σε κάθε μετάδοση, μειώνονται και ο χρόνος εύρεσης μονοπατιού, καθώς και το μήκος του καλύτερου μονοπατιού. Συγκεκριμένα, όταν δημιουργούμε 5 νέα δυναμικά μυρμήγκια, το πρώτο μονοπάτι εντοπίζεται στο βήμα 116, όταν δημιουργούμε 10 μυρμήγκια στο βήμα 115, όταν δημιουργούμε 15 μυρμήγκια στο βήμα 117, όταν δημιουργούμε 20 νέα μυρμήγκια στο βήμα 111 και τέλος όταν δημιουργούμε τα περισσότερα μυρμήγκια, δηλαδή 25, το μονοπάτι εντοπίζεται στο βήμα 102. Βλέπουμε ακόμα ότι το καλύτερο μονοπάτι εντοπίζεται από την προσομοίωση με 25 νέα μυρμήγκια σε κάθε μετάδοση, ενώ όσο μειώνεται ο αριθμός των νέων μυρμηγκιών, αυξάνεται και το μήκος του καλύτερου μονοπατιού. Υπάρχουν βέβαια κάποιες μικροδιαφορές ανάμεσα σε δύο συνεχόμενες εκτελέσεις, αλλά αυτό δεν καθορίζει το τελικό συμπέρασμα.

Εκδοχή Γ

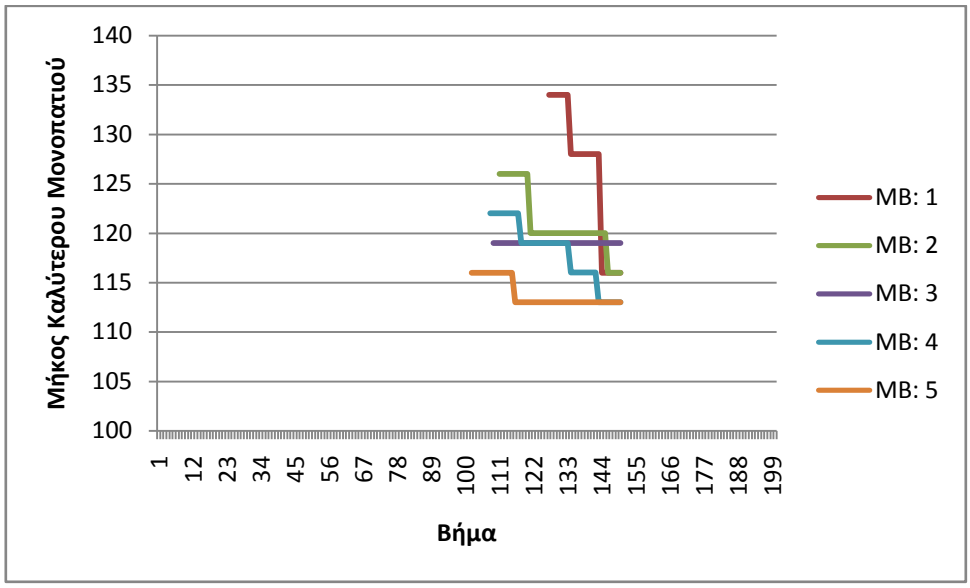
Μεταβλητές Παράμετροι					
Όνομα Παραμέτρου	**1η εκτέλεση**	**2η εκτέλεση**	**3η εκτέλεση**	**4η εκτέλεση**	**5η εκτέλεση**
AntsForage					
MAX_ANTS	1500	1500	1500	1500	1500
ProactiveAnt					
BROADCAST_PROBABILITY	0.5	0.5	0.5	0.5	0.5
NEW_ANTS_PER_BROADCAST	10	10	10	10	10
MAX_BROADCASTS	*1*	*2*	*3*	*4*	*5*
ReactiveAnt					
NEW_ANTS_PER_BROADCAST	5	5	5	5	5
MAX_BROADCASTS	2	2	2	2	2
b1	4	4	4	4	4

Πίνακας 5.8: Μεταβλητές Παράμετροι Εκδοχής Γ

	1η εκτέλεση	3η εκτέλεση	5η εκτέλεση
Βήμα 70			
Βήμα 100			
Βήμα 150			

Στην εκδοχή αυτή του σεναρίου, μεταβάλλουμε τον αριθμό των μεταδόσεων που δικαιούται να εκτελέσει κάθε δυναμικό μυρμήγκι. Αρχικά το περιορίζουμε να εκτελέσει μόνο μια μετάδοση, ενώ στη συνέχεια του επιτρέπουμε να εκτελέσει 2, 3, 4 και 5 μεταδόσεις. Με τη μεταβολή αυτή του αριθμού των μεταδόσεων βλέπουμε κάποιες αλλαγές που αφορούν τη χρονική στιγμή που εντοπίζεται ένα μονοπάτι, καθώς και το μήκος του καλύτερου μονοπατιού της προσομοίωσης.

Διαδικασία εύρεσης μονοπατιού

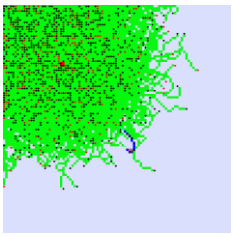
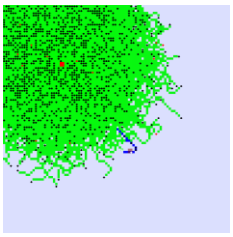
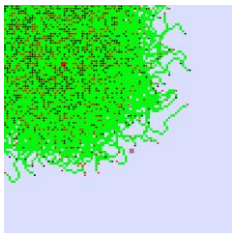
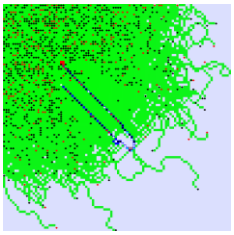
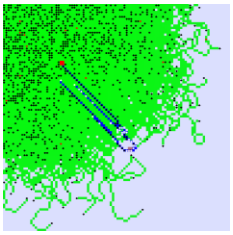
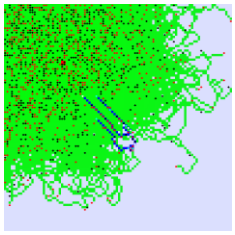
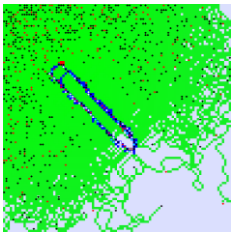
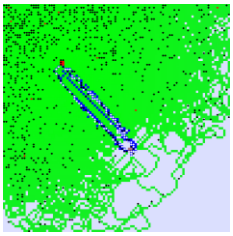
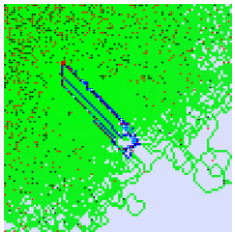


Παρατηρούμε πως όσο αυξάνουμε τον αριθμό των μεταδόσεων που μπορεί να εκτελέσει ένα δυναμικό μυρμήγκι, αυξάνουμε και την πιθανότητα να βρεθεί γρηγορότερα το φαγητό μέσω ενός καλύτερου μονοπατιού. Αυτό φαίνεται και στη γραφική παράσταση, όπου το μονοπάτι εντοπίζεται γρηγορότερα (βήμα 101) όταν οι μεταδόσεις είναι 5 και πολύ αργότερα (βήμα 126) όταν η μετάδοση είναι 1. Η αλλαγή αυτή του αριθμού των μεταδόσεων επηρεάζει και το μήκος του καλύτερου μονοπατιού, αφού όταν οι μεταδόσεις αυξάνονται, αρχίζει να μειώνεται και το μήκος του καλύτερου μονοπατιού, λόγω του ότι τα μυρμήγκια που δημιουργούνται είναι πλέον περισσότερα και μπαίνουν και αυτά στη «μάχη» εύρεσης ενός καλύτερου μονοπατιού.

Εκδοχή Δ

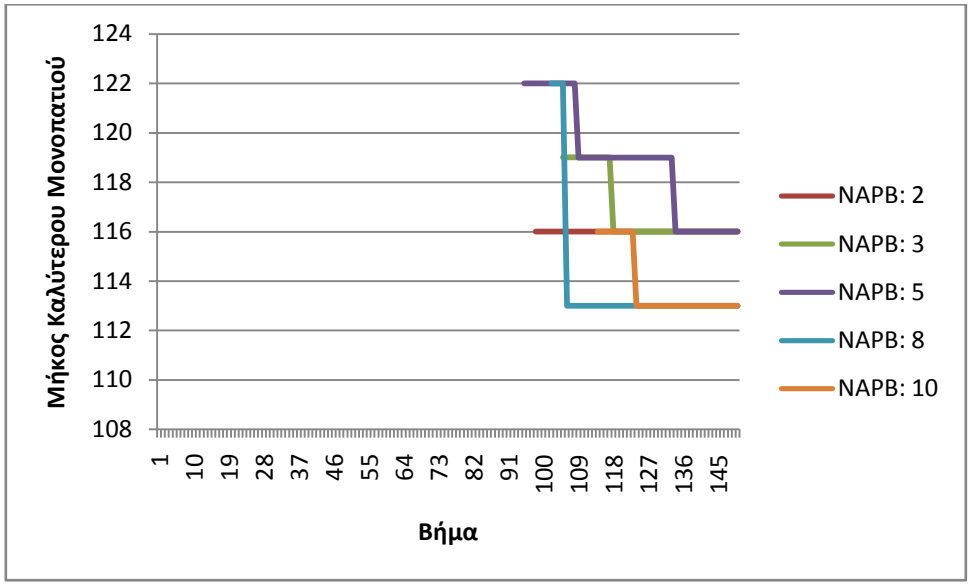
Μεταβλητές Παράμετροι					
Όνομα Παραμέτρου	**1η εκτέλεση**	**2η εκτέλεση**	**3η εκτέλεση**	**4η εκτέλεση**	**5η εκτέλεση**
AntsForage					
MAX_ANTS	1500	1500	1500	1500	1500
ProactiveAnt					
BROADCAST_PROBABILITY	0.5	0.5	0.5	0.5	0.5
NEW_ANTS_PER_BROADCAST	10	10	10	10	10
MAX_BROADCASTS	2	2	2	2	2
ReactiveAnt					
NEW_ANTS_PER_BROADCAST	*2*	*3*	*5*	*8*	*10*
MAX_BROADCASTS	2	2	2	2	2
b1	4	4	4	4	4

Πίνακας 9: Μεταβλητές Παράμετροι Εκδοχής Δ

	1η εκτέλεση	3η εκτέλεση	5η εκτέλεση
Βήμα 70			
Βήμα 100			
Βήμα 150			

Στην εκδοχή αυτή του σεναρίου, μεταβάλλουμε τον αριθμό των νέων αντιδραστικών μυρμηγκιών σε κάθε μετάδοση που εκτελεί ένα αντιδραστικό μυρμήγκι. Αρχικά ορίζουμε τον αριθμό αυτό στο 2 και ακολούθως στο 3, 5. 8 και 10. Με τη μεταβολή αυτή του αριθμού βλέπουμε κάποιες αλλαγές που αφορούν το μήκος του καλύτερου μονοπατιού που εντοπίζουν τα μυρμήγκια.

Διαδικασία εύρεσης μονοπατιού

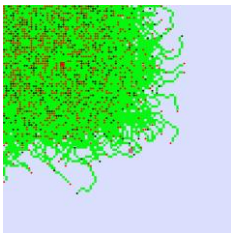
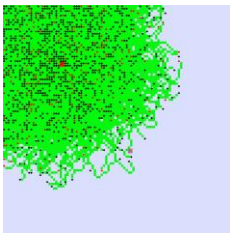
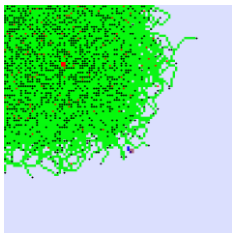
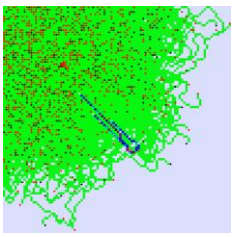
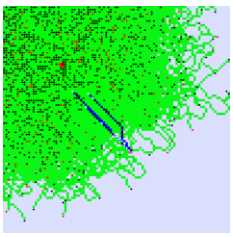
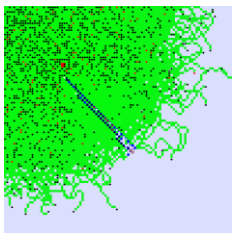
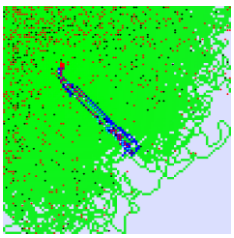
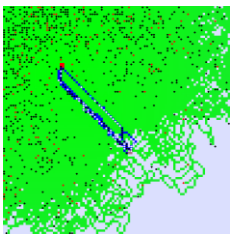
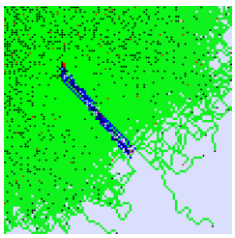


Παρατηρούμε πως αυξανόμενου του αριθμού των νέων αντιδραστικών μυρμηγκιών σε κάθε μετάδοση τέτοιου μυρμηγκιού, φαίνεται να μειώνεται κυρίως το μήκος του καλύτερου μονοπατιού και όχι τόσο σημαντικά ο χρόνος μεταξύ των διάφορων τιμών που μεταβάλλουμε. Αυτό γιατί τα δυναμικά μυρμήγκια εξακολουθούν να κάνουν τη δουλειά τους χωρίς καμιά αλλαγή και βοηθούν τα υπόλοιπα μέλη της ομάδας να εντοπίσουν ένα οποιοδήποτε μονοπάτι και γρήγορα.

Εκδοχή Ε

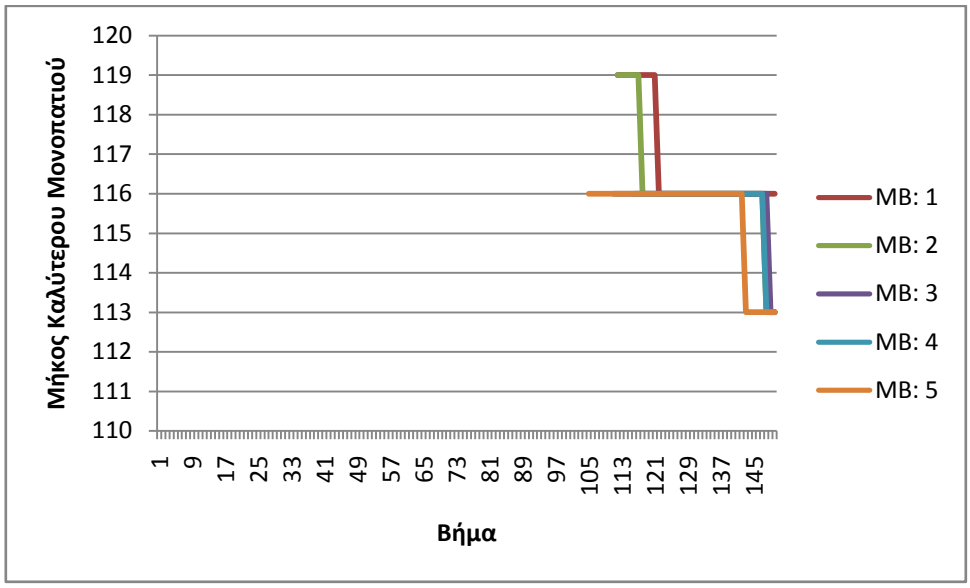
Μεταβλητές Παράμετροι					
Όνομα Παραμέτρου	**1η εκτέλεση**	**2η εκτέλεση**	**3η εκτέλεση**	**4η εκτέλεση**	**5η εκτέλεση**
AntsForage					
MAX_ANTS	1500	1500	1500	1500	1500
ProactiveAnt					
BROADCAST_PROBABILITY	0.5	0.5	0.5	0.5	0.5
NEW_ANTS_PER_BROADCAST	10	10	10	10	10
MAX_BROADCASTS	2	2	2	2	2
ReactiveAnt					
NEW_ANTS_PER_BROADCAST	5	5	5	5	5
MAX_BROADCASTS	*1*	*2*	*3*	*4*	*5*
b1	4	4	4	4	4

Πίνακας 10: Μεταβλητές Παράμετροι Εκδοχής Ε

	1η εκτέλεση	3η εκτέλεση	5η εκτέλεση
Βήμα 70			
Βήμα 100			
Βήμα 150			

Στην εκδοχή αυτή του σεναρίου, μεταβάλλουμε τον αριθμό των μεταδόσεων που δικαιούται να εκτελέσει κάθε δυναμικό μυρμήγκι. Αρχικά το περιορίζουμε να εκτελέσει μόνο μια μετάδοση, ενώ στη συνέχεια του επιτρέπουμε να εκτελέσει 2, 3, 4 και 5 μεταδόσεις. Με τη μεταβολή αυτή του αριθμού των μεταδόσεων βλέπουμε κάποιες αλλαγές που αφορούν τη χρονική στιγμή που εντοπίζεται ένα μονοπάτι, καθώς και το μήκος του καλύτερου μονοπατιού της προσομοίωσης.

<u>Διαδικασία εύρεσης μονοπατιού</u>

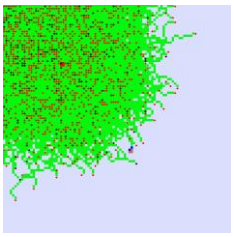
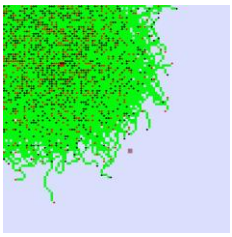
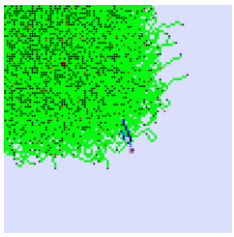
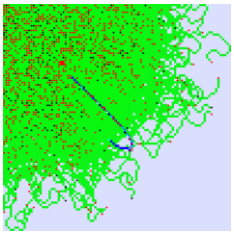
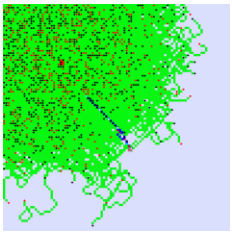
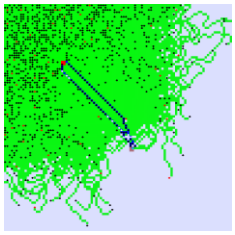
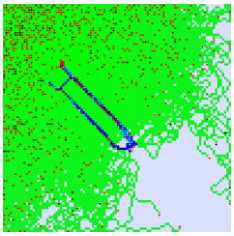
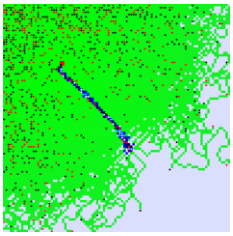
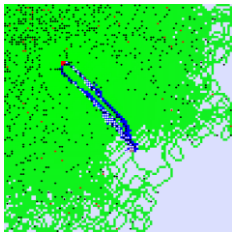


Παρατηρούμε πως υπάρχει ουσιαστική διαφορά στη διαδικασία μείωσης του μήκους του καλύτερου μονοπατιού. Αν και ο χρόνος δεν παίζει τόσο ρόλο στην εύρεση ενός οποιουδήποτε μονοπατιού, εντούτοις είναι φανερό πως αυξανόμενου του αριθμού των μεταδόσεων που μπορεί να εκτελέσει ένα αντιδραστικό μυρμήγκι, μειώνεται και το μήκος του καλύτερου μονοπατιού που εντοπίζεται σε όλη την προσομοίωση.

Εκδοχή Z

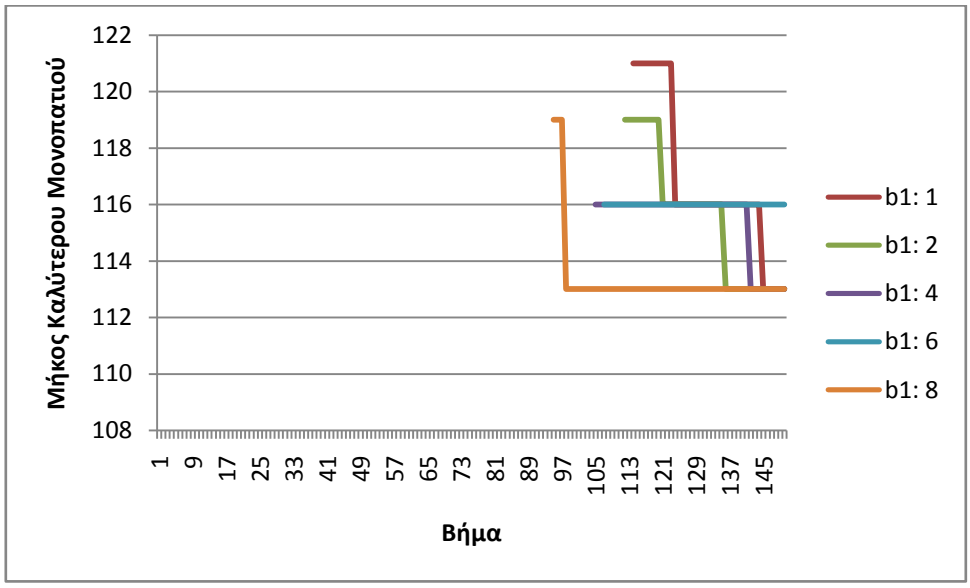
Μεταβλητές Παράμετροι					
Όνομα Παραμέτρου	**1η εκτέλεση**	**2η εκτέλεση**	**3η εκτέλεση**	**4η εκτέλεση**	**5η εκτέλεση**
AntsForage					
MAX_ANTS	1500	1500	1500	1500	1500
ProactiveAnt					
BROADCAST_PROBABILITY	0.5	0.5	0.5	0.5	0.5
NEW_ANTS_PER_BROADCAST	10	10	10	10	10
MAX_BROADCASTS	2	2	2	2	2
ReactiveAnt					
NEW_ANTS_PER_BROADCAST	5	5	5	5	5
MAX_BROADCASTS	2	2	2	2	2
b1	*1*	*2*	*4*	*6*	*8*

Πίνακας 11: Μεταβλητές Παράμετροι Εκδοχής Z

	1η εκτέλεση	3η εκτέλεση	5η εκτέλεση
Βήμα 70			
Βήμα 100			
Βήμα 150			

Στην εκδοχή αυτή του σεναρίου, μεταβάλλουμε τον παράγοντα που καθορίζει την εξερευνητική συμπεριφορά των αντιδραστικών μυρμηγκιών. Αρχικά του δίνουμε την τιμή 1 και ακολούθως 2, 4, 6 και 8 ώστε να παρατηρήσουμε αν με την αύξηση του παράγοντα αυτού θα αλλάξουν τα αποτελέσματα της προσομοίωσης. Παρατηρούμε λοιπόν ότι με την αύξηση του παράγοντα b1 αλλάζει η συμπεριφορά των αντιδραστικών μυρμηγκιών, χωρίς όμως αυτό να επηρεάζει αισθητά την όλη λειτουργία της ομάδας μιας και υπάρχουν άλλοι παράγοντες που παίζουν σημαντικότερο ρόλο στον αλγόριθμο.

Διαδικασία εύρεσης μονοπατιού

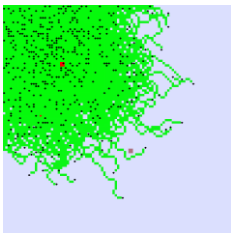
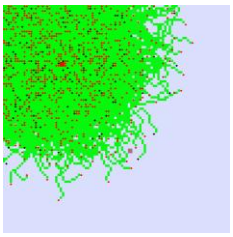
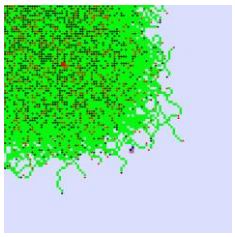
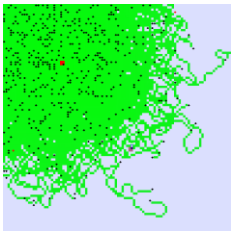
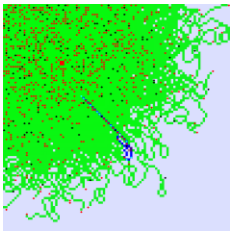
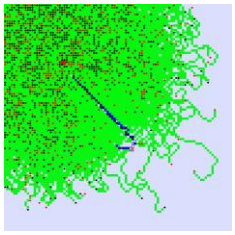
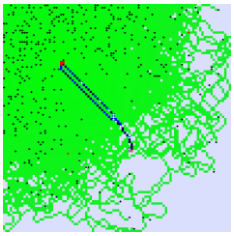
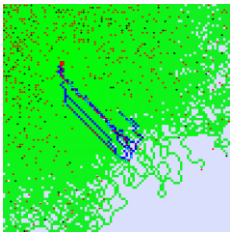
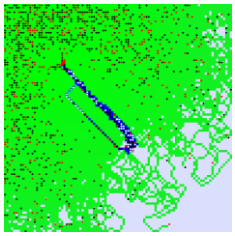


Όπως βλέπουμε, η μεγάλη διαφορά φαίνεται όταν το b1 καθορίζεται σε μεγάλη τιμή (b1: 8), όπου το πρώτο μονοπάτι εντοπίζεται στο βήμα 94 και αν και δεν έχει το καλύτερο μήκος, εντούτοις στη συνέχεια βελτιώνεται αισθητά. Όσον αφορά τις υπόλοιπες τιμές του παράγοντα b1, όσο μειώνεται η τιμή του φαίνεται να υπάρχουν διαφορές ως προς τη χρονική στιγμή της εύρεσης ενός μονοπατιού, αλλά οι διαφορές του μήκους για το καλύτερο μονοπάτι δεν είναι μεγάλες.

Εκδοχή Η

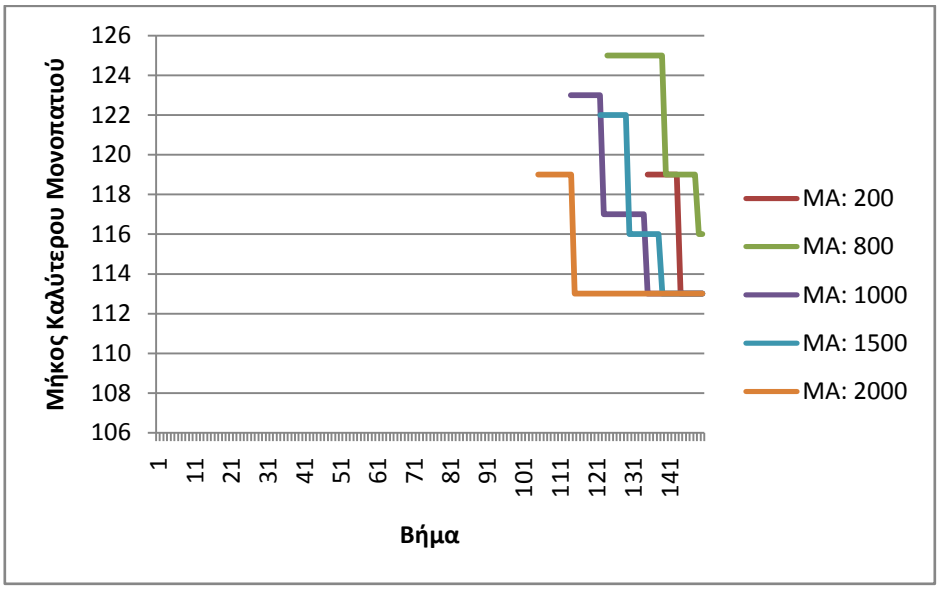
Μεταβλητές Παράμετροι					
Όνομα Παραμέτρου	**1η εκτέλεση**	**2η εκτέλεση**	**3η εκτέλεση**	**4η εκτέλεση**	**5η εκτέλεση**
AntsForage					
MAX_ANTS	*500*	*800*	*1000*	*1500*	*2000*
ProactiveAnt					
BROADCAST_PROBABILITY	0.5	0.5	0.5	0.5	0.5
NEW_ANTS_PER_BROADCAST	10	10	10	10	10
MAX_BROADCASTS	2	2	2	2	2
ReactiveAnt					
NEW_ANTS_PER_BROADCAST	5	5	5	5	5
MAX_BROADCASTS	2	2	2	2	2
b1	4	4	4	4	4

Πίνακας 12: Μεταβλητές Παράμετροι Εκδοχής Η

	1η εκτέλεση	3η εκτέλεση	5η εκτέλεση
Βήμα 70			
Βήμα 100			
Βήμα 150			

Στην εκδοχή αυτή του σεναρίου, μεταβάλλουμε το μέγιστο αριθμό μυρμηγκιών που δικαιούνται να υπάρχουν στην προσομοίωση. Αρχικά ορίζουμε τον αριθμό αυτό στο 200 και ακολούθως στο 800, 1000, 1500 και 2000. Με τη μεταβολή αυτή του μέγιστου αριθμού μυρμηγκιών θέλουμε να δείξουμε πώς αλλάζει η συμπεριφορά των μυρμηγκιών και η διαδικασία εύρεσης μονοπατιού όταν αυξάνονται τα μυρμήγκια που μπορούν να δημιουργηθούν.

<u>Διαδικασία εύρεσης μονοπατιού</u>



Όπως αναμέναμε, αυξάνοντας τον αριθμό των μυρμηγκιών που μπορούν να δημιουργηθούν στην προσομοίωση, αυξάνεται και η πιθανότητα εύρεσης ενός καλύτερου μονοπατιού σε συντομότερο χρονικό διάστημα. Όπως φαίνεται και στη γραφική παράσταση, όταν ο μέγιστος αριθμός μυρμηγκιών είναι 2000 τα μυρμήγκια βρίσκουν το πρώτο τους μονοπάτι στο βήμα 104, εντοπίζοντας το βέλτιστο με μήκος 113 στο βήμα 114. Όταν μεταβάλουμε την τιμή του μέγιστου αριθμού μυρμηγκιών σε 1500 και 1000, παρατηρούμε ότι τα μυρμήγκια βρίσκουν το καλύτερο μονοπάτι (μήκους 113) κάποια στιγμή αργότερα στην προσομοίωση. Αντίθετα, όταν ο μέγιστος αριθμός μυρμηγκιών στην προσομοίωση είναι 200, τα μυρμήγκια βρίσκουν ένα μονοπάτι στο βήμα 134, αλλά δεν μπορούν να φτάσουν το καλύτερο μήκος του μονοπατιού που εντοπίζεται για άλλες τιμές μέγιστους αριθμού μυρμηγκιών.

5.4 Σύγκριση προσομοιώσεων στους αλγόριθμους AODV και AntHocNet

Σημειώνουμε ότι ισχύουν τα ίδια χαρακτηριστικά και για τους δύο αλγόριθμους, όπως αυτά περιγράφηκαν για τον αλγόριθμο AODV στο υποκεφάλαιο *5.1* και για τον αλγόριθμο AntHocNet στο υποκεφάλαιο *5.2.* Επισημαίνουμε επίσης ότι για το σενάριο σύγκρισης των δύο αλγορίθμων επιλέγηκαν οι σημαντικότερες παραμέτροι για κάθε αλγόριθμο που θα συνδυαστούν, γιατί ο αριθμός του συνδυασμού όλων των παραμέτρων ήταν τόσο μεγάλος.

5.4.1 Σενάριο Σύγκρισης AODV και AntHocNet

Σταθερές Παράμετροι	
AntsForage	
HOME_XMIN	25
HOME_XMAX	26
HOME_YMIN	25
HOME_YMAX	26
FOOD_XMIN	55
FOOD_XMAX	56
FOOD_YMIN	63
FOOD_YMAX	64
MIN_PHEROMONE	0
MAX_PHEROMONE	1000
PHEROMONE_TO_LEAVE_BEHIND	1
TIME_TO_LIVE	1000

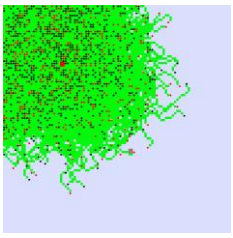
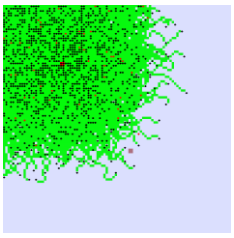
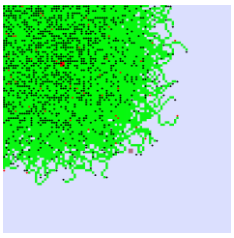
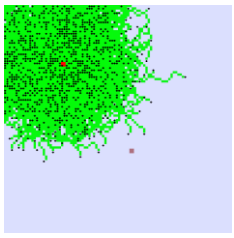
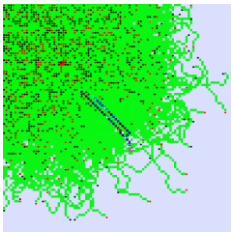
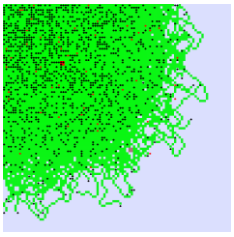
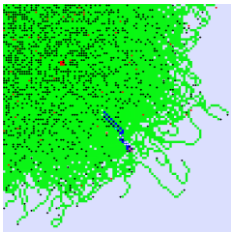
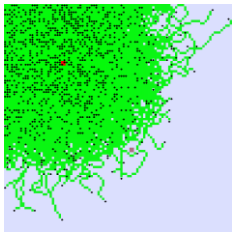
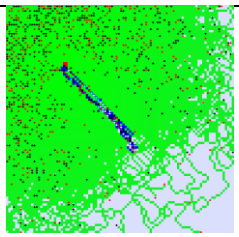
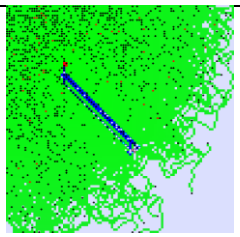
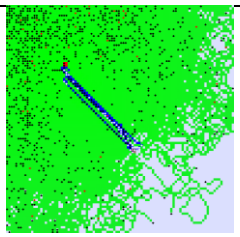
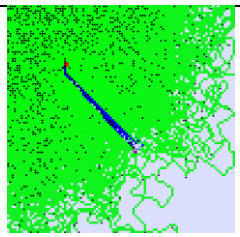
Πίνακας 13: Σταθερές Παράμετροι αλγορίθμων AODV και AntHocNet

Στην προσομοίωση αυτή τοποθετήσαμε τη φωλιά αρκετά κάτω και στο κέντρο, ενώ αφήσαμε το φαγητό σε μακρινή απόσταση πάνω αριστερά. Κάθε μυρμήγκι «ζει» για 1000 βήματα. Κάθε μυρμήγκι επιτρέπεται να αφήσει ένα ποσό φερομονών από 0 μέχρι 1000, αφήνοντας 1 κάθε φορά που κινείται σε νέα θέση.

Εκδοχή Α

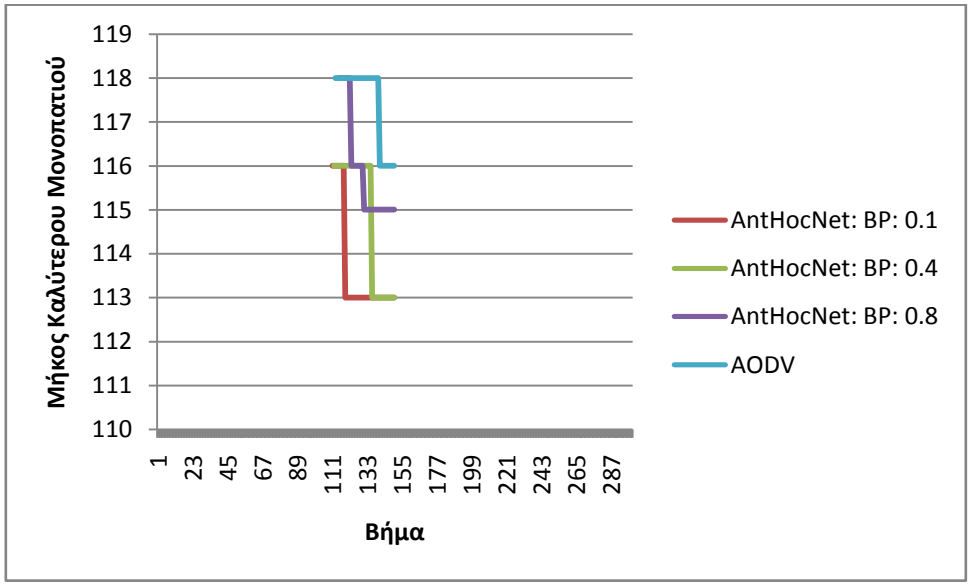
Μεταβλητές Παράμετροι				
	AntHocNet			**AODV**
Όνομα Παραμέτρου	**1η εκτέλεση**	**2η εκτέλεση**	**3η εκτέλεση**	
AntsForage				
MAX_ANTS	1500	1500	1500	1500
ProactiveAnt				
BROADCAST_PROBABILITY	*0.1*	*0.4*	*0.8*	-
NEW_ANTS_PER_BROADCAST	5	5	5	-
MAX_BROADCASTS	2	2	2	-
ReactiveAnt (Ant)				
NEW_ANTS_PER_BROADCAST	5	5	5	10
MAX_BROADCASTS	2	2	2	-
b1	4	4	4	-

Πίνακας 14: Μεταβλητές Παράμετροι Εκδοχής Α

	AntHocNet			AODV
	1η εκτέλεση	**2η εκτέλεση**	**3η εκτέλεση**	
Βήμα 70				
Βήμα 100				
Βήμα 150				

Όπως αναμέναμε, ο AntHocNet βρίσκει γρηγορότερα το μονοπάτι που οδηγεί από το φαγητό στη φωλιά. Ο AODV βρίσκει ένα μονοπάτι αλλά πολύ αργότερα από τον AntHocNet, έστω και αν έχουμε ορίσει ότι τα μυρμήγκια που δημιουργούνται και στους δύο αλγόριθμους κατά τις μεταδόσεις είναι περίπου τα ίδια.

Διαδικασία εύρεσης μονοπατιού

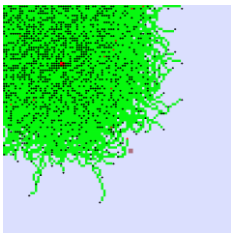
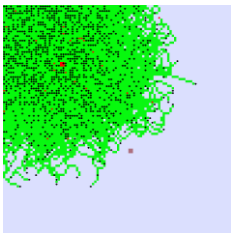
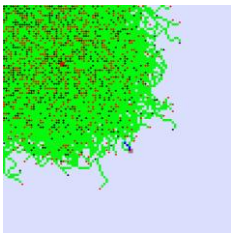
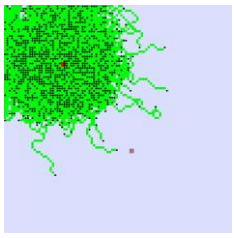
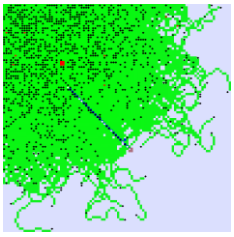
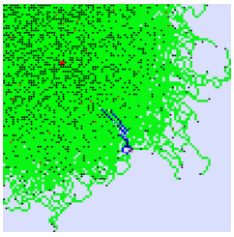
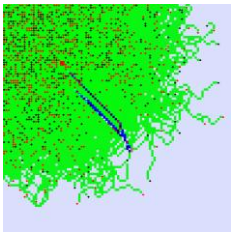
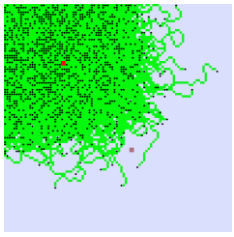
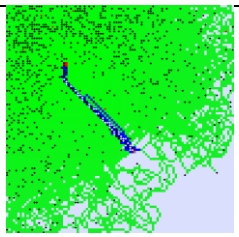
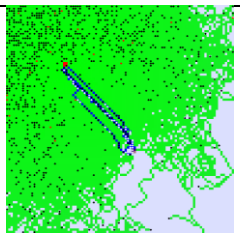
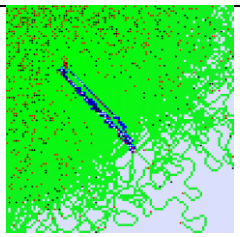
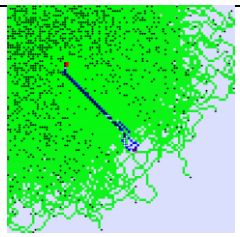


Βλέπουμε ότι οι 2 εκτελέσεις του AntHocNet βρίσκουν ένα μονοπάτι με περίπου το ίδιο μήκος και αυτό γιατί αλλάζουμε μόνο την πιθανότητα για τα δυναμικά μυρμήγκια, χωρίς να επηρεάζουμε την κίνηση των αντιδραστικών μυρμηγκιών. Το μονοπάτι που εντοπίζουν τα μυρμήγκια στον AODV είναι κοντά στο μήκος του μονοπατιού που εντοπίζεται από την πρώτη εκτέλεση του AntHocNet. Το γεγονός αυτό ήταν αναμενόμενο, αφού ο AntHocNet έχει σχεδιαστεί με τέτοιο υβριδικό τρόπο, που να καταφέρνει να βρίσκει γρηγορότερα τα καλύτερα μονοπάτια.

Εκδοχή Β

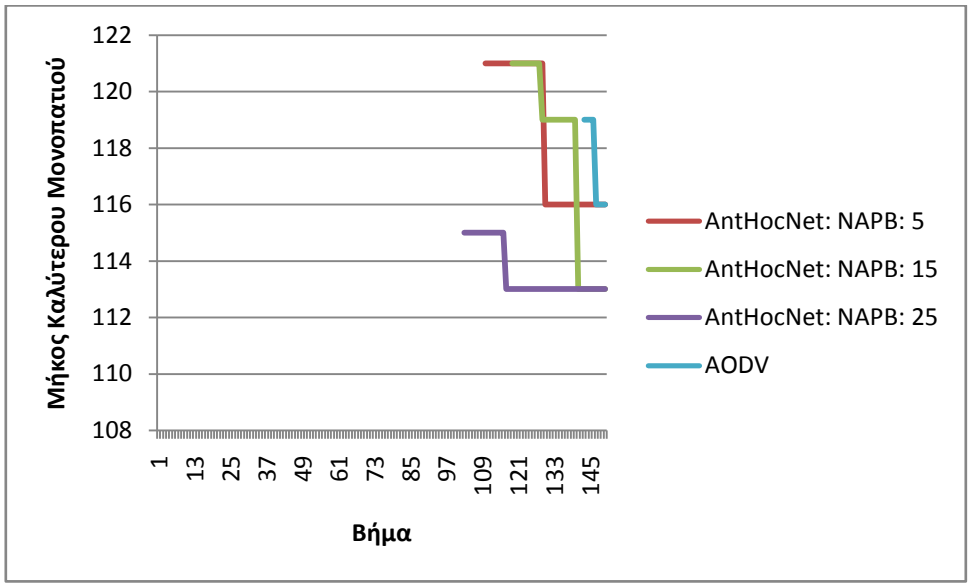
Μεταβλητές Παράμετροι				
	AntHocNet			**AODV**
Όνομα Παραμέτρου	**1η εκτέλεση**	**2η εκτέλεση**	**3η εκτέλεση**	
AntsForage				
MAX_ANTS	1500	1500	1500	1500
ProactiveAnt				
BROADCAST_PROBABILITY	0.5	0.5	0.5	-
NEW_ANTS_PER_BROADCAST	*5*	*15*	*25*	-
MAX_BROADCASTS	2	2	2	-
ReactiveAnt (Ant)				
NEW_ANTS_PER_BROADCAST	5	5	5	30
MAX_BROADCASTS	2	2	2	-
b1	4	4	4	-

Πίνακας 15: Μεταβλητές Παράμετροι Εκδοχής Β

	AntHocNet			AODV
	1η εκτέλεση	**2η εκτέλεση**	**3η εκτέλεση**	
Βήμα 70				
Βήμα 100				
Βήμα 150				

Βλέπουμε ότι όσο περισσότερα μυρμήγκια δημιουργούμε σε κάθε μετάδοση ενός δυναμικού μυρμηγκιού στον AntHocNet, τόσο πιο εύκολο είναι για τα μυρμήγκια να βρουν ένα μονοπάτι και αυτό το μονοπάτι να έχει μεγαλύτερες πιθανότητες να είναι το καλύτερο. Στον αλγόριθμο AODV, δημιουργούμε σταθερά πολλά μυρμήγκια αφήνοντας έτσι τον αλγόριθμο να βρει ένα πολύ καλό μονοπάτι. Με τον τρόπο αυτό ισορροπούμε την κατάσταση μεταξύ των δύο αλγορίθμων, ενισχύοντας κατά κάποιο τρόπο τον AODV που φαίνεται όμως να εξακολουθεί να υστερεί έναντι του AntHocNet.

Διαδικασία εύρεσης μονοπατιού

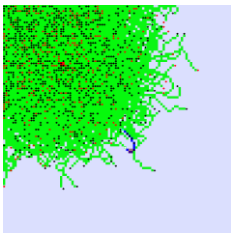
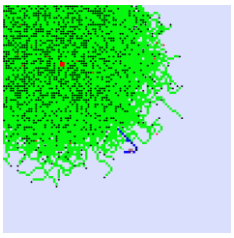
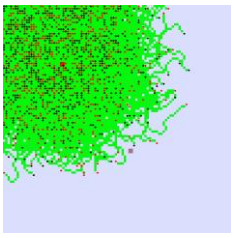
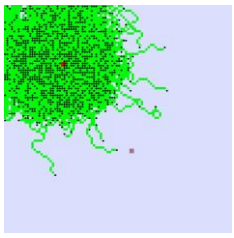
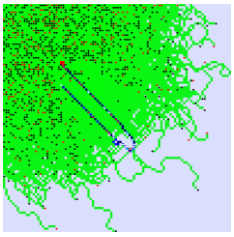
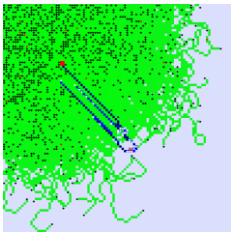
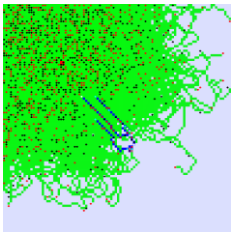
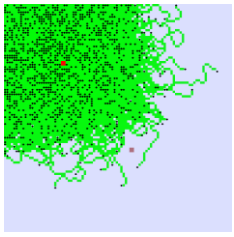
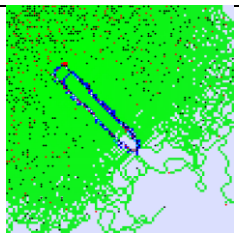
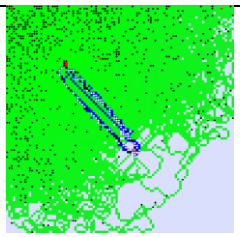
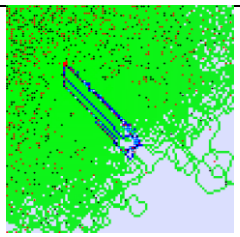
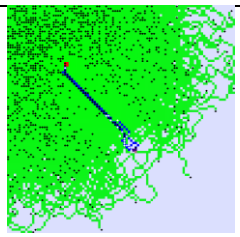


Παρατηρούμε πως όταν δημιουργούμε 25 δυναμικά μυρμήγκια σε κάθε μετάδοση του AntHocNet, το μονοπάτι εντοπίζεται μόλις στο βήμα 102, ενώ αντίθετα με τη δημιουργία 30 μυρμηγκιών σε κάθε μετάδοση του αλγόριθμου AODV το μονοπάτι εντοπίζεται στο βήμα 142 και είναι χειρότερο από αυτό του AntHocNet (113 και 119 αντίστοιχα). Οι ενδιάμεσες τιμές για τον αριθμό των νέων μυρμηγκιών στον AntHocNet, παρουσιάζουν καλά αποτελέσματα όσον αφορά το μήκος του μονοπατιού μετά την πάροδο ενός χρονικού διαστήματος, ενώ φαίνεται πως βρίσκουν ένα μονοπάτι πολύ πιο γρήγορα από τον AODV.

Εκδοχή Γ

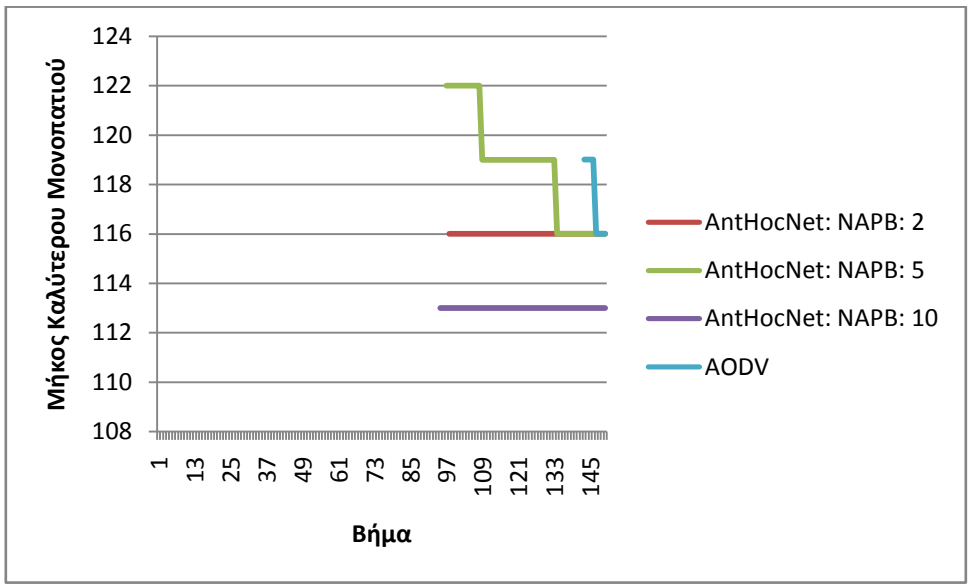
Μεταβλητές Παράμετροι				
	AntHocNet			**AODV**
Όνομα Παραμέτρου	**1η εκτέλεση**	**2η εκτέλεση**	**3η εκτέλεση**	
AntsForage				
MAX_ANTS	1500	1500	1500	1500
ProactiveAnt				
BROADCAST_PROBABILITY	0.5	0.5	0.5	-
NEW_ANTS_PER_BROADCAST	5	5	5	-
MAX_BROADCASTS	2	2	2	-
ReactiveAnt (Ant)				
NEW_ANTS_PER_BROADCAST	*2*	*5*	*10*	*30*
MAX_BROADCASTS	2	2	2	-
b1	4	4	4	-

Πίνακας 16: Μεταβλητές Παράμετροι Εκδοχής Γ

	AntHocNet			AODV
	1η εκτέλεση	**2η εκτέλεση**	**3η εκτέλεση**	
Βήμα 70				
Βήμα 100				
Βήμα 150				

Παραθέτοντας διάφορες εκτελέσεις για τον AntHocNet και μια για τον AODV, βλέπουμε πως αυξανόμενου του αριθμού των νέων μυρμηγκιών σε κάθε μετάδοση ενός αντιδραστικού μυρμηγκιού, μειώνεται και ο χρόνος που χρειάζονται τα μυρμήγκια της ομάδας για να εντοπίσουν ένα μονοπάτι. Το γεγονός αυτό όμως δεν είναι απαραίτητο να επηρεάζει το μήκος του καλύτερου μονοπατιού της προσομοίωσης, αφού η δουλειά των αντιδραστικών μυρμηγκιών είναι ο εντοπισμός ενός μονοπατιού και όχι ο εντοπισμός του καλύτερου μονοπατιού. Παρόλα αυτά, ο AntHocNet παραμένει καλύτερος στον εντοπισμό του καλύτερου μονοπατιού σε λιγότερο χρόνο, ανεξάρτητα της τιμής των νέων μυρμηγκιών σε κάθε μετάδοση.

Διαδικασία εύρεσης μονοπατιού

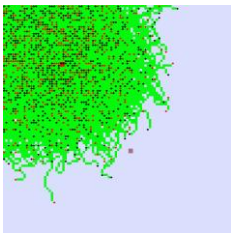
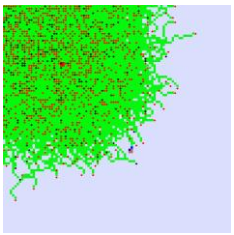
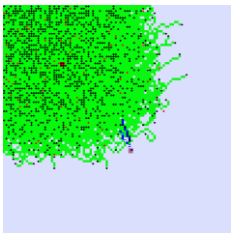
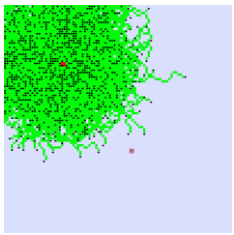
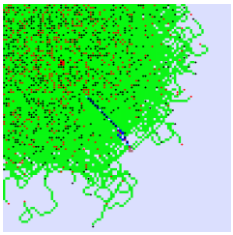
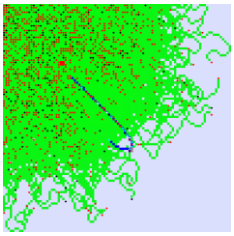
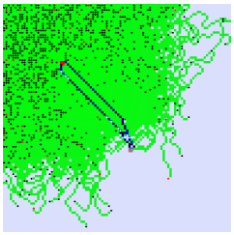
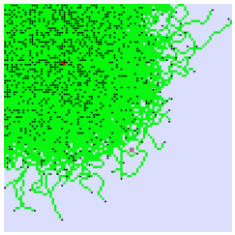
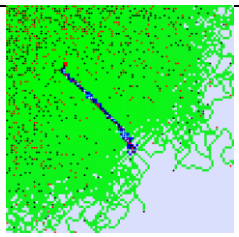
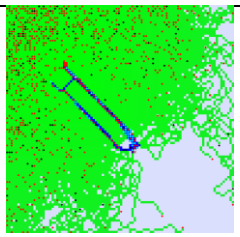
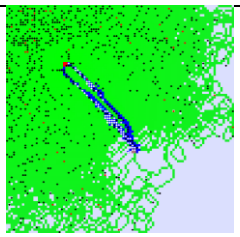
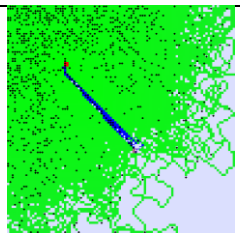


Βλέπουμε πως το καλύτερο μονοπάτι από όλες τις προσομοιώσεις που εκτελέσαμε για την εκδοχή αυτή του σεναρίου, παρατηρείται όταν τα νέα αντιδραστικά μυρμήγκια σε κάθε μετάδοση είναι 10. Παρατηρούμε επίσης ότι ο AODV βρίσκει ένα καλό μονοπάτι στο τέλος της προσομοίωσης (σε σύγκριση με τις άλλες δύο εκτελέσεις του AntHocNet), εντούτοις ο εντοπισμός αυτός του μονοπατιού γίνεται πολύ αργά σε σχέση με τις 3 εκτελέσεις του αλγόριθμου AntHocNet.

Εκδοχή Δ

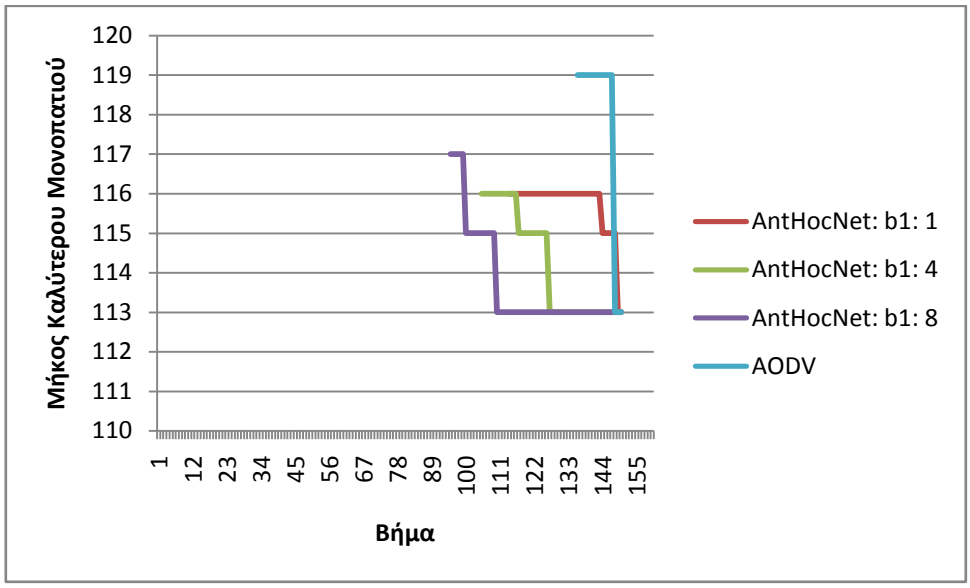
Μεταβλητές Παράμετροι				
	AntHocNet			AODV
Όνομα Παραμέτρου	**1η εκτέλεση**	**2η εκτέλεση**	**3η εκτέλεση**	
AntsForage				
MAX_ANTS	1500	1500	1500	1500
ProactiveAnt				
BROADCAST_PROBABILITY	0.2	0.2	0.2	-
NEW_ANTS_PER_BROADCAST	5	5	5	-
MAX_BROADCASTS	2	2	2	-
ReactiveAnt (Ant)				
NEW_ANTS_PER_BROADCAST	5	5	5	10
MAX_BROADCASTS	2	2	2	-
b1	*1*	*4*	*8*	-

Πίνακας 17: Μεταβλητές Παράμετροι Εκδοχής Δ

	AntHocNet			AODV
	1η εκτέλεση	**2η εκτέλεση**	**3η εκτέλεση**	
Βήμα 70				
Βήμα 100				
Βήμα 150				

Αυξάνοντας τον παράγοντα που καθορίζει την εξερευνητική συμπεριφορά των μυρμηγκιών στον AntHocNet, αναμένουμε πως τα μυρμήγκια θα βρουν γρηγορότερα κάποιο μονοπάτι. Έτσι γίνεται, αφού βλέπουμε πως τα μυρμήγκια κάθε φορά εντοπίζουν γρηγορότερα το πρώτο μονοπάτι. Όσον αφορά τον AODV, φαίνεται και πάλι πως αργεί να εντοπίσει ένα οποιοδήποτε μονοπάτι, ανεξάρτητα από το αν το μονοπάτι αυτό θα είναι καλύτερο ή χειρότερο από αυτά που εντοπίζουν τα μυρμήγκια στον AntHocNet.

Διαδικασία εύρεσης μονοπατιού

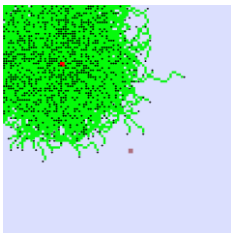
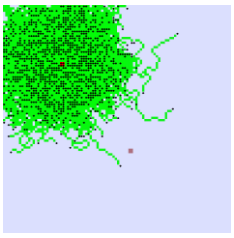
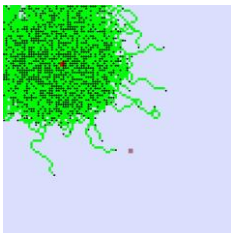
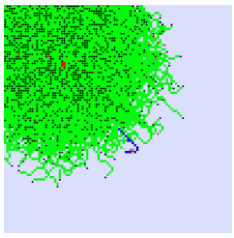
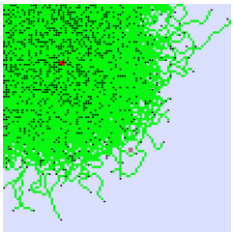
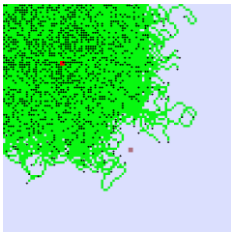
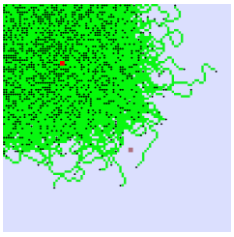
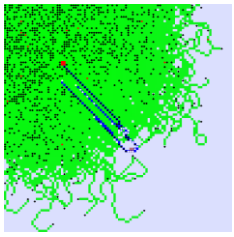
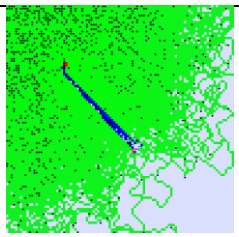
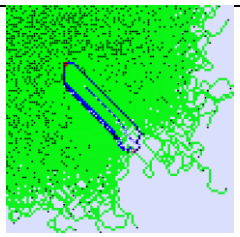
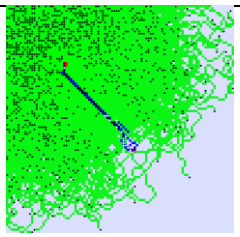
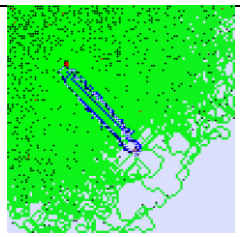


Βλέπουμε πως αυξανόμενης της τιμής του παράγοντα b1 μειώνεται ο χρόνος εύρεσης ενός μονοπατιού στον αλγόριθμο AntHocNet. Δεν ισχύει το ίδιο για το μήκος του καλύτερου μονοπατιού, αφού αργά ή γρήγορα τα μυρμήγκια εντοπίζουν το καλύτερο δυνατό μονοπάτι. Αντίθετα, στον αλγόριθμο AODV τα μυρμήγκια, αν και στο τέλος βρίσκουν το καλύτερο δυνατό μονοπάτι, εντούτοις αργούν πολύ να εντοπίσουν το πρώτο τους μονοπάτι.

Εκδοχή Ε

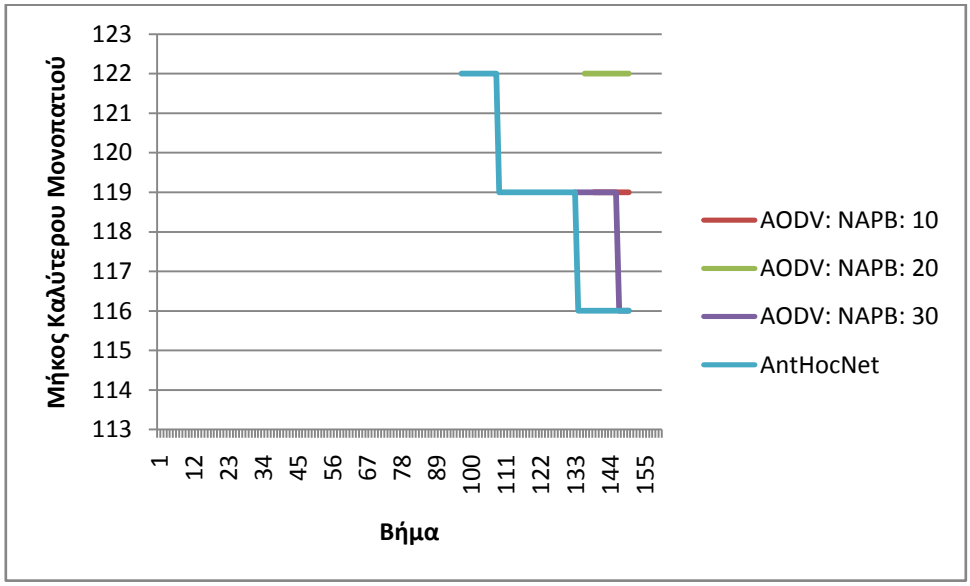
Μεταβλητές Παράμετροι				
	AODV			AntHocNet
Όνομα Παραμέτρου	**1η εκτέλεση**	**2η εκτέλεση**	**3η εκτέλεση**	
AntsForage				
MAX_ANTS	1500	1500	1500	1500
ProactiveAnt				
BROADCAST_PROBABILITY	-	-	-	0.5
NEW_ANTS_PER_BROADCAST	-	-	-	5
MAX_BROADCASTS	-	-	-	2
ReactiveAnt (Ant)				
NEW_ANTS_PER_BROADCAST	*10*	*20*	*30*	*5*
MAX_BROADCASTS	-	-	-	2
b1	-	-	-	4

Πίνακας 18: Μεταβλητές Παράμετροι Εκδοχής Ε

	AODV			AntHocNet
	1η εκτέλεση	**2η εκτέλεση**	**3η εκτέλεση**	
Βήμα 70				
Βήμα 100				
Βήμα 150				

Αυτή τη φορά μεταβάλλουμε μια παράμετρο που αφορά τον αλγόριθμο AODV. Είναι εμφανές ότι ακόμα και όταν αυξάνουμε κατά πολύ τον αριθμό των νέων μυρμηγκιών σε κάθε μετάδοση του αλγορίθμου AODV, τα μυρμήγκια του δεν μπορούν να εντοπίσουν το πρώτο τους μονοπάτι τόσο γρήγορα όσο τα μυρμήγκια στον AntHocNet.

Διαδικασία εύρεσης μονοπατιού

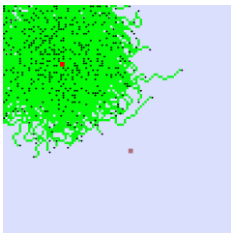
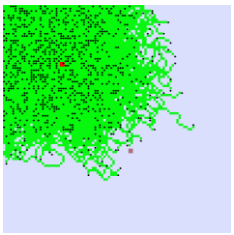
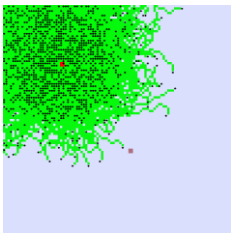
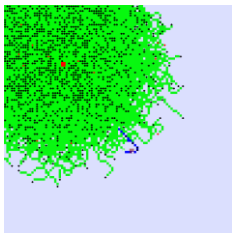
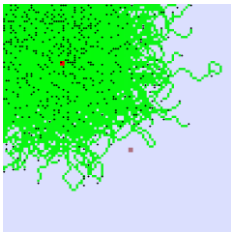
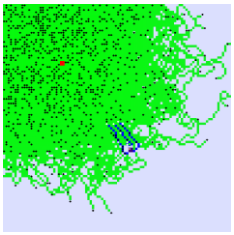
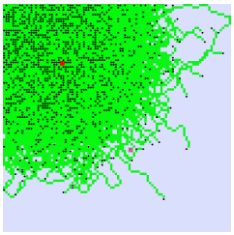
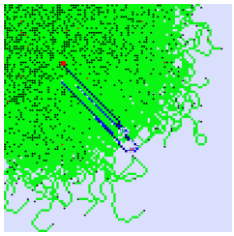
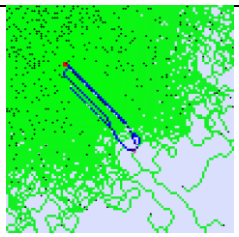
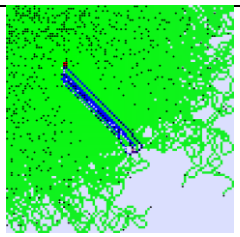
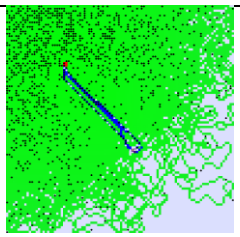
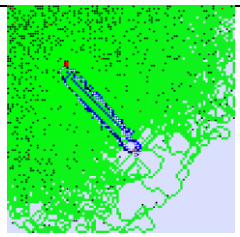


Παρατηρούμε πως σε μια μόνο εκτέλεση ο AntHocNet καταφέρνει να βρει γρηγορότερα ένα μονοπάτι και τελικά να έχει εντοπίσει το καλύτερο μονοπάτι από όλες τις εκτελέσεις του AODV. Αντίθετα, ο AODV φαίνεται να καλυτερεύει τα αποτελέσματά του όταν αυξάνονται τα αντιδραστικά μυρμήγκια που δημιουργούνται σε κάθε νέα μετάδοση. Δεν μπορούμε όμως να θεωρήσουμε τον AODV καλύτερο για νέο αριθμό μυρμηγκιών ίσο με 30 (NAPB: 30) έστω και αν παρουσιάζει παρόμοια αποτελέσματα σε θέματα εύρεσης του καλύτερου μονοπατιού. Αυτό γιατί ο αριθμός αυτός των νέων μυρμηγκιών ορίζεται για λόγους εξέτασης της συμπεριφοράς των δύο αλγορίθμων και δεν μπορεί να θεωρηθεί ως δεδομένος. Όταν λοιπόν ορίσουμε τους δύο αλγόριθμους να δημιουργούν περίπου τον ίδιο αριθμό μυρμηγκιών σε κάθε νέα μετάδοση, τότε ο AntHocNet είναι εμφανώς καλύτερος τόσο στην εύρεση του καλύτερου μονοπατιού, όσο και στο χρόνο που βρίσκει το πρώτο μονοπάτι.

Εκδοχή Z

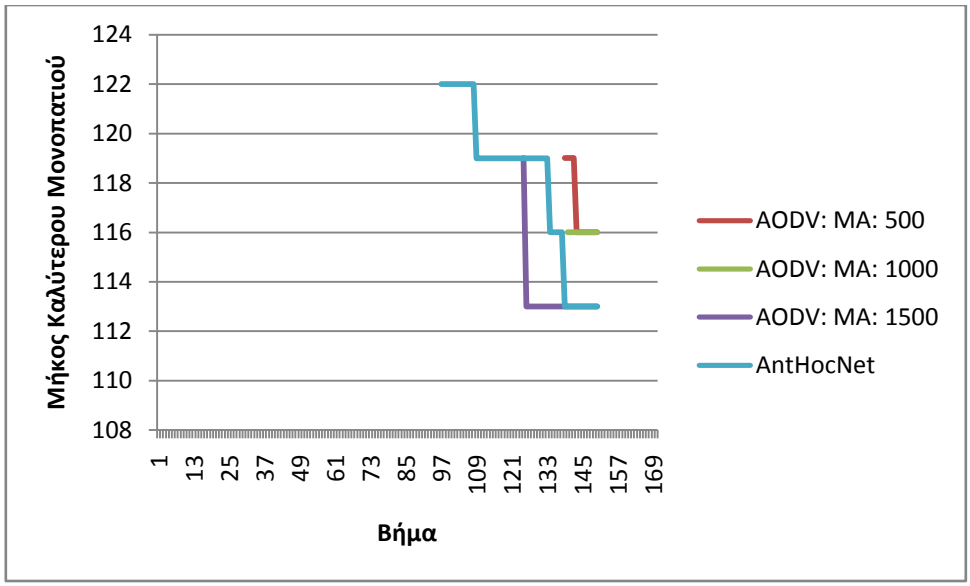
Μεταβλητές Παράμετροι				
	AODV			AntHocNet
Όνομα Παραμέτρου	**1η εκτέλεση**	**2η εκτέλεση**	**3η εκτέλεση**	
AntsForage				
MAX_ANTS	*500*	*1000*	*1500*	*1500*
ProactiveAnt				
BROADCAST_PROBABILITY	-	-	-	0.5
NEW_ANTS_PER_BROADCAST	-	-	-	5
MAX_BROADCASTS	-	-	-	2
ReactiveAnt				
NEW_ANTS_PER_BROADCAST	15	15	15	5
MAX_BROADCASTS	-	-	-	2
b1	-	-	-	4

Πίνακας 19: Μεταβλητές Παράμετροι Εκδοχής Z

	AODV			AntHocNet
	1η εκτέλεση	**2η εκτέλεση**	**3η εκτέλεση**	
Βήμα 70				
Βήμα 100				
Βήμα 150				

Στην εκδοχή αυτή του σεναρίου, μεταβάλλουμε το μέγιστο αριθμό μυρμηγκιών της προσομοίωσης. Βλέπουμε ότι όσο αυξάνεται ο αριθμός αυτός στον αλγόριθμο AODV, το πρώτο μονοπάτι εντοπίζεται σε γρηγορότερα, με μικρές διαφορές στους χρόνους μεταξύ 2 εκτελέσεων. Εντούτοις, ο αλγόριθμος AntHocNet, φαίνεται και πάλι να έχει καλύτερα αποτελέσματα όσον αφορά το χρόνο εύρεσης του πρώτου μονοπατιού, παρά τη λιγότερα «ενίσχυση» που του προσφέρουμε. Συγκεκριμένα, αφήνουμε τον αλγόριθμο AODV να δημιουργεί περισσότερα μυρμήγκια σε σταθερή βάση ώστε να δούμε αν το γεγονός αυτό θα το βοηθήσει να υπερτερήσει έναντι του AntHocNet.

Διαδικασία εύρεσης μονοπατιού



Παρατηρούμε πως αυξανόμενου του μέγιστου αριθμού μυρμηγκιών που μπορούν να λαμβάνουν μέρος στην προσομοίωση, τα αποτελέσματα του AODV γίνονται καλύτερα. Δηλαδή, μειώνεται ο χρόνος εύρεσης του πρώτου μονοπατιού και για μέγιστο αριθμό μυρμηγκιών ίσο με 1500 (MA: 1500) το καλύτερο μονοπάτι συμπίπτει με αυτό του AntHocNet. Εντούτοις, πρέπει να σημειώσουμε ότι ορίσαμε τον αριθμό των νέων μυρμηγκιών στον AODV να είναι μεγαλύτερος από αυτόν στον AntHocNet. Επομένως, θα περιμέναμε να δούμε καλύτερα αποτελέσματα στον AODV. Αυτό όμως δε συμβαίνει λόγω της υβριδικής ιδιότητας που χαρακτηρίζει τον αλγόριθμο AntHocNet.

ΚΕΦΑΛΑΙΟ 6:

ΣΥΜΠΕΡΑΣΜΑΤΑ

6.1 Συμπεράσματα

Στο υποκεφάλαιο αυτό θα συνοψίσουμε όλη τη γνώση και δουλειά που έγινε για τις ανάγκες της ατομικής διπλωματικής εργασίας. Συγκεκριμένα, θα κάνουμε μια αναδρομή στις εργασίες που έγιναν από την αρχή και τελικά θα εξάγουμε κάποια συμπεράσματα που αφορούν τη σύνδεση που έγινε μεταξύ των δύο πεδίων, δρομολόγηση σε κινητά ad hoc δίκτυα και δρομολόγηση στον πραγματικό κόσμο με βάση τα μυρμήγκια.

Αρχικά, μελετήσαμε κάποιες πηγές που αναφέρονταν σε δρομολόγηση σε κινητά ad hoc δίκτυα. Αφού κατανοήσαμε την ιδέα λειτουργίας τους και κάποιους από τους αλγόριθμους που εφαρμόζονται σε αυτά, προχωρήσαμε ένα βήμα παραπάνω, δηλαδή στη μελέτη αλγορίθμων που σχετίζονται πιο συγκεκριμένα με μυρμήγκια. Κάνοντας την κατάλληλη σύνδεση μεταξύ δρομολόγησης σε δίκτυα και δρομολόγησης με χρήση αλγορίθμων προερχόμενων από τη φύση – και συγκεκριμένα τα μυρμήγκια – μελετήσαμε σε περισσότερο βάθος δύο από τους αλγόριθμους δρομολόγησης σε κινητά ad hoc δίκτυα, τον AODV και AntHocNet. Στη συνέχεια, εντοπίσαμε τα κοινά στοιχεία μεταξύ των δύο αλγορίθμων, καθώς και τα στοιχεία που διαφοροποιούν τον ένα με τον άλλο με

σκοπό να τους συγκρίνουμε και να διαμορφώσουμε μια άποψη για τις ομοιότητες και διαφορές τους σε θεωρητικό επίπεδο. Η διαδικασία αυτή μας βοήθησε να επαληθεύσουμε τα αποτελέσματα που πήραμε ακολούθως και μετά την υλοποίηση μιας εκδοχής των δύο αλγορίθμων.

Φτάνοντας σε ένα προχωρημένο στάδιο μελέτης της θεωρίας που αφορούσε το θέμα της ατομικής διπλωματικής εργασίας, μεταβήκαμε σε ένα άλλο επίπεδο, αυτό της πρακτικής εφαρμογής των όσων μελετήθηκαν. Έτσι, προχωρήσαμε στην υλοποίηση μιας εκδοχής των αλγορίθμων AODV και AntHocNet που αφορούσε τη γραφική αναπαράσταση της συμπεριφοράς των μυρμηγκιών στον πραγματικό κόσμο, όπως αυτή υιοθετείται από τους αλγόριθμους δρομολόγησης σε κινητά ad hoc δίκτυα.

Αμέσως μετά την υλοποίηση και την προσομοίωση αρκετών σεναρίων, εισάγαμε στο έγγραφο αυτό κάποια αντιπροσωπευτικά παραδείγματα με συγκεκριμένους συνδυασμούς παραμέτρων. Η διαδικασία αυτή ήταν χρήσιμη για την εξαγωγή κάποιων συμπερασμάτων που αφορούν τη σύγκριση των δύο αλγορίθμων, των οποίων οι εκδοχές υλοποιήθηκαν. Πιο κάτω αναφέρουμε με περισσότερη λεπτομέρεια τα αποτελέσματα της θεωρητικής μελέτης για τους δύο αλγόριθμους, καθώς και τα συμπεράσματα που εξήχθηκαν μετά την προσομοίωση των εκδοχών των αλγορίθμων που υλοποιήθηκαν.

Μελετώντας τα άρθρα κάποιων ερευνητών, καθώς και τα άρθρα των καθηγητών που πρότειναν για πρώτη φορά τους αλγόριθμους που επιλέξαμε να μελετήσουμε σε βάθος, παρατηρήσαμε πως οι δύο αλγόριθμοι, AODV και AntHocNet, συγκρίνονταν σε μεγάλο βαθμό λόγω της αξιοπιστίας και της χρησιμότητάς τους σε κινητά ad hoc δίκτυα. Είδαμε λοιπόν πως ο αλγόριθμος AntHocNet μέσα από πολλές προσομοιώσεις οριζόταν ως ο καλύτερος και πιο αποδοτικός, λόγω της υβριδικής του φύσης. Δηλαδή, ο AntHocNet παρουσιαζόταν να έχει γενικά καλύτερα αποτελέσματα σε κινητά ad hoc δίκτυα από αυτά του AODV που παρουσίαζε καλύτερα αποτελέσματα μόνο σε θέματα καθυστέρησης μετάδοσης πακέτων. Η διαφορά αυτή των δύο αλγορίθμων οφείλεται στο ότι ο

AntHocNet υλοποιεί τόσο αντιδραστικά όσο και δυναμικά χαρακτηριστικά με σκοπό τη βελτίωση των αποτελεσμάτων δρομολόγησης, ενώ το AODV αποτελεί ένα καθαρά αντιδραστικό πρωτόκολλο (αν και θεωρείται το καλύτερο μεταξύ των αντιδραστικών πρωτοκόλλων).

Μετά την υλοποίηση συγκεκριμένων εκδοχών των δύο αλγορίθμων που αφορούσαν καθαρά την προσομοίωση της συμπεριφοράς των μυρμηγκιών στον πραγματικό κόσμο, εκτελέσαμε αρκετές προσομοιώσεις με σκοπό και πάλι τη σύγκριση των δύο αλγορίθμων. Χρησιμοποιήσαμε διάφορες τιμές για τις παραμέτρους του προγράμματος ώστε να δούμε τη συμπεριφορά των μυρμηγκιών κάτω από διαφορετικές συνθήκες. Γενικά, παρατηρήσαμε ότι ο AntHocNet επιβεβαίωσε την προηγούμενη μελέτη των ερευνητών αφού παρουσιάστηκε ως επί το πλείστον καλύτερος από τον AODV. Όταν λέμε καλύτερος εννοούμε ότι με τις ίδιες τιμές παραμέτρων και τον ίδιο αριθμό βημάτων, ο AntHocNet παρουσίαζε πιο ολοκληρωμένα αποτελέσματα.

Συγκεκριμένα, τα μυρμήγκια στον AntHocNet – είτε αντιδραστικά είτε δυναμικά – έφταναν γρηγορότερα στον προορισμό τους σχηματίζοντας καλύτερα και συντομότερα μονοπάτια από αυτά που εντόπιζαν τα μυρμήγκια στον αλγόριθμο AODV. Έτσι, για έναν ορισμένο αριθμό βημάτων παρατηρήσαμε ότι τα περισσότερα μυρμήγκια της ομάδας, στον αλγόριθμο AntHocNet, βρίσκονταν ήδη στο καλύτερο μονοπάτι, λόγω των φερομονών που άφηναν στο έδαφος τα μυρμήγκια που κατάφεραν λίγη ώρα νωρίτερα να εντοπίσουν με επιτυχία το μονοπάτι που τα οδηγεί από τη φωλιά στο φαγητό και αντίστροφα. Από την άλλη, για τον ίδιο αριθμό βημάτων, λιγότερα μυρμήγκια του αλγορίθμου AODV βρίσκονταν στο μονοπάτι που βρέθηκε (το οποίο ήταν ελαφρά χειρότερο από αυτό της εκτέλεσης στον AntHocNet). Επίσης, παρατηρήσαμε πως τα μυρμήγκια στον AntHocNet έτειναν να βρίσκουν τουλάχιστον ένα μονοπάτι σε λιγότερο χρόνο από αυτόν που χρειάζονταν τα μυρμήγκια του AODV για να εντοπίσουν ένα οποιοδήποτε μονοπάτι.

Καταληκτικά, σημειώνουμε πως τα αποτελέσματα της υλοποίησης μιας εκδοχής των δύο αλγορίθμων επαλήθευσαν τις προσδοκίες μας, δηλαδή ταυτίστηκαν με τα συμπεράσματα που εξήχθηκαν από έρευνες και άρθρα που μελέτησαν και σύγκριναν τους δύο αλγόριθμους, AODV και AntHocNet, τόσο σε θεωρητικό όσο και σε πρακτικό επίπεδο.

6.2 Μελλοντική εργασία

Για σκοπούς επιπρόσθετης αξιολόγησης της αποδοτικότητας των δύο αλγορίθμων, AODV και AntHocNet, θα μπορούσαν να υλοποιηθούν και άλλες εκδοχές αλγορίθμων δρομολόγησης δικτύων. Με τον τρόπο αυτό θα είναι δυνατή η σύγκριση αποτελεσμάτων μεταξύ περισσότερων αλγόριθμων, ώστε να είναι δυνατή η εξαγωγή πιο γενικών συμπερασμάτων που να αφορούν αλγόριθμους δρομολόγησης δικτύων που εμπνεύστηκαν από τη συμπεριφορά των μυρμηγκιών.

Είναι εφικτό να υλοποιηθούν ορισμένοι από τους αλγόριθμους που αναφέραμε στο Κεφάλαιο 2 ώστε να συγκριθούν με τις εκδοχές των αλγορίθμων που υλοποιήσαμε στα πλαίσια της ατομικής διπλωματικής εργασίας αυτής. Συγκεκριμένα, μια ολοκληρωμένη έρευνα που θα ασχολείτο ειδικά με τους αλγόριθμους δρομολόγησης, θα μπορούσε να μελετήσει τα χαρακτηριστικά των 3 ομάδων αλγορίθμων (αντιδραστικοί, δυναμικοί, υβριδικοί) και να υλοποιήσει κάποιες εκδοχές των αντιπροσωπευτικότερων αλγορίθμων προσαρμοσμένες στη συμπεριφορά των μυρμηγκιών. Ακολούθως, θα ήταν χρήσιμο να επιλέξει τους πιο αποδοτικούς από κάθε ομάδα και να τους συγκρίνει μεταξύ τους, ώστε να εξαχθούν ορισμένα αποτελέσματα που να οδηγούν κάποιο δημιουργό ενός δικτύου στην επιλογή ενός κατάλληλου πρωτοκόλλου. Φυσικά, κάθε αλγόριθμος θεωρείται αποδοτικός όταν εκτελείται κάτω από συγκεκριμένες συνθήκες. Δεν μπορούμε δηλαδή να αποφασίσουμε την καταλληλότητα κάποιου αλγόριθμου αν δε γνωρίζουμε τις παραμέτρους που χρησιμοποιεί ένα δίκτυα. Έτσι, η υλοποίηση αυτή των διάφορων

αλγορίθμων θα ήταν πιο χρήσιμο να έδινε κάποια μετρήσιμα αποτελέσματα αποδοτικότητας με χρήση συγκεκριμένων παραμέτρων.

Επίσης, σαν χρήστης του εργαλείου Mason θα πρότεινα σε όποιον επιχειρήσει να συνεχίσει τη δουλειά της ατομικής διπλωματικής εργασίας αυτής, να λάβει σοβαρά υπόψη τα πλεονεκτήματα της χρήστης του εργαλείου αυτού. Το εργαλείο Mason προσφέρει πολλές ευκολίες στο χρήστη του, καθώς και πολλή βοήθεια αν αυτός αντιμετωπίσει οποιαδήποτε προβλήματα. Η άποψη μου λοιπόν είναι πως για την υλοποίηση αλγορίθμων που αφορούν γραφική απεικόνιση της συμπεριφοράς των μυρμηγκιών, το εργαλείο Mason αποτελεί ίσως την καλύτερη δυνατή επιλογή.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Anandamoy S., *"Swarm Intelligence based optimization of MANET cluster formation"*, M.Sc. Thesis, The University of Arizona, 2006.

[2] Arabshahi P., Gray A., Kassabalidis I., Das A.K., Narayanan S., El-Sharkawi M.A., Marks II R.J., "Adaptive Routing in Wireless Communication Networks using Swarm Intelligence", *In Proceedings of 19th AIAA International Communications Satellite Systems Conference,* 2001.

[3] B. Cameron Lesiuk, "Routing in Ad-Hoc Networks of Mobile Hosts", *Mech590 Report*, University of Victoria, British Columbia, 1998.

[4] Beijar N., *"Zone Routing Protocol (ZRP)"*, website at: www.netlab.tkk.fi/opetus/s38030/k02/Papers/08-Nicklas.pdf (τελευταία επίσκεψη στις 05/04/2009).

[5] Coppent J., Vincent V., Strobbe M., Breusegem E.V., Pickavet M., Demeester P., "AntNet: ACO routing algorithm in practice", *In Proceedings of the 8e INFORMS Telecommunications Conference*, 2006.

[6] *Destination Sequenced Distance Vector*, website at: http://wiki.uni.lu/secan-lab/graphics/exampledsdv.GIF (τελευταία επίσκεψη στις 25/04/2009).

[7] Di Caro G., Ducatelle F. and Gambardella L.M., "AntHocNet: An adaptive nature-inspired algorithm for routing in mobile ad hoc networks", *European Transactions on Telecommunications*, 2005.

[8] Di Caro G., Ducatelle F. and Gambardella L.M., "AntHocNet: An Ant-Based Hybrid Routing Algorithm for Mobile Ad Hoc Networks", *In Proceedings of PPSN VIII – Eight International Conference on Parallel Problem Solving from Nature*, 2004, pp: 461-470.

[9] Dongsoo S.K., Seok K.H., "Markov Model of Link Connectivity in Mobile Ad Hoc Networks", *Special Issue of Telecommunication Systems Journal*, 2006.

[10] Dorigo M., Di Caro G., Gambardella L.M., *"Ant Algorithms for discrete optimization"*, Vol.5, N. 2, Artificial Life, 1999, pp: 137-172.

[11] *Dynamic Source Routing,* website at: http://wiki.uni.lu/secan-lab/Dynamic+Source+Routing.html (τελευταία επίσκεψη στις 20/03/2009).

[12] Eclipse website at: http://www.eclipse.org (τελευταία επίσκεψη στις 29/04/2009).

[13] Erciyes K., Pinar D., Akile S., "A Distributed Dynamic Routing Protocol for Arbitrary Networks", *Technical Report*, Department of Mathematics Izmir, 1998.

[14] Gerla M., Pei G., Tsu-Tsu-Wei C., "Fisheye State Routing in Mobile Ad Hoc Networks", *Proceedings of Workshop on Wireless Networks and Mobile Computing*, 2000.

[15] Janson S., Merkle D. and Middendofr M., "A decentralization approach for swarm intelligence algorithms in networks applied to multi swarm PSO", *International Journal of Intelligent Computing and Cybernetics*, pp: 24-45, 2008.

[16] Marnerides A., *"Working with the GridKit Overlay Framework, The Secure-AntHocNet Overlay"*, M.Sc. Thesis, Faculty of Science and Technology, Lancaster University, 2007.

[17] Mason Multiagent Simulation Toolkit website at:
http://www.cs.gmu.edu/~eclab/projects/mason/
(τελευταία επίσκεψη στις 12/05/2009).

[18] Mobile Ad-hoc Network, Wikipedia website at:
http://en.wikipedia.org/wiki/MANET (τελευταία επίσκεψη στις 05/03/2009).

[19] Mobile Ad Hoc Networks, website at:
http://nraresearch.blogspot.com/ (τελευταία επίσκεψη στις 11/05/2009).

[20] Mobile ad-hoc networks (MANETs) Chapter, website at:
http://www.ietf.org/html.charters/manet-charter.html
(τελευταία επίσκεψη στις 18/04/2009).

[21] Perkins C.E. and E.M. Royer, "Ad-hoc on-demand distance vector routing", *In Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, 1999.

[22] Perkins C., Bhagwat P., "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for mobile computers", *ACM SIGCOMM Symposium on Communication, Architecture and Protocols,* 1994, pp: 233-244.

[23] Philippe J., Mühlethaler P., Clausen Th., Laouiti A., Qayyum A. and Viennot L., "Optimized Link State Routing protocol for ad hoc networks", *In Proceedings of the IEEE International Multitopic Conference (INMIC)*, 2001.

[24] Raynolds A. and Corne D., *"Swarm Intelligence: A tutorial account. Bulletin of the European Association for Theoretical Computer Science, Columns: Natural Computing"*, 2008.

[25] Temporarily-Ordered Routing Algorithm, website at:
http://wiki.uni.lu/secan-lab/Temporally-Ordered+Routing+Algorithm.html
(τελευταία επίσκεψη στις 25/03/2009).

[26] Theraulaz G., Dorigo M. and Bonabeau E., "Swarm Intelligence: From Natural to Artificial Systems", *Oxford University Press,* 1999.

[27] Wireless Routing Protocol, website at:
http://en.wikipedia.org/wiki/Wireless_Routing_Protocol
(τελευταία επίσκεψη στις 23/02/2009).

[28] Zygmunt J. Haas and Marc R. Pearlman, "A New Routing Protocol for the Reconfigurable Wireless Networks", *In Proceedings of the 6th IEEE International Conference on Universal Personal Communications,* 1997.

ΠΑΡΑΡΤΗΜΑ Α:

ΟΛΟΚΛΗΡΩΜΕΝΟΣ ΚΩΔΙΚΑΣ ΚΟΙΝΩΝ ΚΛΑΣΕΩΝ ΑΛΓΟΡΙΘΜΩΝ

Στο παράρτημα αυτό παραθέτουμε τις κοινές κλάσεις που χρησιμοποιούν οι δύο αλγόριθμοι, AODV και AntHocNet, κατά την υλοποίησή τους.

Κλάση DecisionInfo

```
/*
  Copyright 2006 by Sean Luke and George Mason University
  Licensed under the Academic Free License version 3.0
  See the file "LICENSE" for more information
*/

package sim.app.antsforage;
import java.awt.*;

public class DecisionInfo implements java.io.Serializable
    {

    public Point position;
    public int orientation;
    public double homePheromoneAmount;
    public double foodPheromoneAmount;

    // to be computed from homePheromoneAmount and foodPheromoneAmount
based on goal (go to home or to food)
    public double profit;

    public DecisionInfo() { position = new Point(); }

    }
```

Κλάση DecisionMaker

```
/*
  Copyright 2006 by Sean Luke and George Mason University
  Licensed under the Academic Free License version 3.0
  See the file "LICENSE" for more information
*/

package sim.app.antsforage;

import sim.engine.SimState;
```

```
public /*strictfp*/ class DecisionMaker implements java.io.Serializable
    {

    public DecisionInfo[] info = null;
    public int numInfos = 0;

    public DecisionMaker()
        {
        info = new DecisionInfo[8];
        for( int i = 0 ; i < info.length ; i++ )
            info[i] = new DecisionInfo();
        }

    public void reset() { numInfos = 0; }

    public void addInfo( DecisionInfo di )
        {
        info[numInfos].position.x = di.position.x;
        info[numInfos].position.y = di.position.y;
        info[numInfos].orientation = di.orientation;
        info[numInfos].homePheromoneAmount = di.homePheromoneAmount;
        info[numInfos].foodPheromoneAmount = di.foodPheromoneAmount;
        numInfos++;
        }

    public DecisionInfo getHomeDecision( final SimState state )
        {
        for( int i = 0 ; i < numInfos ; i++ )
            {
            processForHomeDecision( info[i] );
            }
        return getDecision( state );
        }

    public DecisionInfo getFoodDecision( final SimState state )
        {
        for( int i = 0 ; i < numInfos ; i++ )
            {
            processForFoodDecision( info[i] );
            }
        return getDecision( state );
        }

    protected void processForHomeDecision( final DecisionInfo info )
        {
        info.profit = info.homePheromoneAmount;
        }

    protected void processForFoodDecision( final DecisionInfo info )
        {
        info.profit = info.foodPheromoneAmount;
        }

    public DecisionInfo getDecision( final SimState state )
        {
```

```
        int x, index;

        if( numInfos == 0 )
            {
            return null;
            }

        // first normalize
        double sum=0.0;
        for(x=0;x<numInfos;x++)
            {
            if (info[x].profit<0.0)
                throw new ArithmeticException("Distribution has negative
probabilities");
            sum += info[x].profit;
            }
        if (sum==0.0) throw new ArithmeticException("Distribution has all
0 probabilities");
        for(x=0;x<numInfos;x++)
            info[x].profit /= sum;

        // now sum
        sum=0.0;
        for(x=0;x<numInfos;x++)
            {
            sum += info[x].profit;
            info[x].profit = sum;
            }

        // now we need to work backwards setting 0 values
        for(x=numInfos-1; x > 0; x--)
            if (info[x].profit==info[x-1].profit)  // we're 0.0
                info[x].profit = 1.0;
            else
                break;
        info[x].profit = 1.0;

        //
        // make the decision (pick randomly)
        //
        double prob = state.random.nextDouble();
        if (numInfos==1) // quick
            return info[0];
        // simple linear scan
        for(x=0;x<numInfos-1;x++)
            if (info[x].profit>prob)
                {
                index = x;
                if (info[index].profit==0.0) // I need to scan forward
because I'm in a left-trail
                    while(index < numInfos-1 && info[index].profit==0.0)
                        index++;
                else
                    while(index > 0 && info[index].profit==info[index-
1].profit)
                        index--;
```

```
                return info[index];
                }
        index = numInfos-1;
        if (info[index].profit==0.0) // I need to scan forward because
I'm in a left-trail
            while(index < numInfos-1 && info[index].profit==0.0)
                index++;
        else
            while(index > 0 && info[index].profit==info[index-1].profit)
                index--;
        return info[index];
        }

    public DecisionInfo getHomeGreedyDecision( final SimState state )
        {

        int index;

        if( numInfos == 0 )
            {
            return null;
            }

        for( int i = 0 ; i < numInfos ; i++ )
            {
            processForHomeDecision( info[i] );
            }

        // compute the maximum value
        index = 0;
        for( int i = 0 ; i < numInfos ; i++ )
            if( info[i].profit > info[index].profit )
                index = i;

        int howMany = 0;
        for( int i = 0 ; i < numInfos ; i++ )
            if( info[i].profit == info[index].profit )
                howMany++;

        int x = state.random.nextInt( howMany );
        for( int i = 0 ; i < numInfos ; i++ )
            if( info[i].profit == info[index].profit )
                if( x == 0 )
                    return info[i];
                else
                    x--;
        return null;

        }

    public DecisionInfo getFoodGreedyDecision( final SimState state )
        {

        int index;

        if( numInfos == 0 )
```

```
            {
            return null;
            }

        for( int i = 0 ; i < numInfos ; i++ )
            {
            processForFoodDecision( info[i] );
            }

        // compute the maximum value
        index = 0;
        for( int i = 0 ; i < numInfos ; i++ )
            if( info[i].profit > info[index].profit )
                index = i;

        int howMany = 0;
        for( int i = 0 ; i < numInfos ; i++ )
            if( info[i].profit == info[index].profit )
                howMany++;

        int x = state.random.nextInt( howMany );
        for( int i = 0 ; i < numInfos ; i++ )
            if( info[i].profit == info[index].profit )
                if( x == 0 )
                    return info[i];
                else
                    x--;
        return null;

        }

    }
```

Κλάση Diffuser

```
/*
  Copyright 2006 by Sean Luke and George Mason University
  Licensed under the Academic Free License version 3.0
  See the file "LICENSE" for more information
*/

package sim.app.antsforage;

import sim.engine.*;
import sim.field.grid.*;

public /*strictfp*/ class Diffuser implements Steppable
    {

    DoubleGrid2D updateGrid;
    DoubleGrid2D tempGrid;
    double evaporationRate;
```

```
    double diffusionRate;

    public Diffuser( final DoubleGrid2D updateGrid,
                     final DoubleGrid2D tempGrid,
                     final double evaporationRate,
                     final double diffusionRate )
        {
        this.updateGrid = updateGrid;
        this.tempGrid = tempGrid;
        this.evaporationRate = evaporationRate;
        this.diffusionRate = diffusionRate;
        }

    public void step(SimState state)
        {
        // stolen from HeatBugs and modified for our own purposes
        // locals are faster than instance variables
        final DoubleGrid2D _valgrid = updateGrid;
        final double[][] _valgrid_field = updateGrid.field;
        final double[][] _valgrid2_field = tempGrid.field;
        final int _gridWidth = _valgrid.getWidth();
        final int _gridHeight = _valgrid.getHeight();
        final double _evaporationRate = evaporationRate;
        final double _diffusionRate = diffusionRate;

        double average;

        double[] _past = _valgrid_field[_valgrid.stx(-1)];
        double[] _current = _valgrid_field[0];
        double[] _next;
        double[] _put;

        int yminus1;
        int yplus1;

        // for each x and y position
        for(int x=0;x< _gridWidth;x++)
            {
            _next = _valgrid_field[_valgrid.stx(x+1)];
             put =  valgrid2 field[ valgrid.stx(x)];

            yminus1 = _valgrid.sty(-1);     // initialized
            for(int y=0;y< _gridHeight;y++)
                {
                // for each neighbor of that position
                // go across top
                yplus1 = _valgrid.sty(y+1);
                average = (_past[yminus1] + _past[y] + _past[yplus1] +
                           _current[yminus1] + _current[y] +
_current[yplus1] +
                           _next[yminus1] + _next[y] + _next[yplus1]) /
9.0;

                // load the new value into HeatBugs.this.valgrid2
                _put[y] = (1.0-_evaporationRate) *
                    (_current[y] + _diffusionRate *
```

```
                    (average - _current[y]));

                // set y-1 to what y was "last time around"
                yminus1 = y;
                }

            // swap elements
            _past = _current;
            _current = _next;
            }

        // now finally copy HeatBugs.this.valgrid2 to
HeatBugs.this.valgrid, and we're done
        _valgrid.setTo(tempGrid);
        }
    }
```

Κλάση GreedyDecisionMaker

```
/*
  Copyright 2006 by Sean Luke and George Mason University
  Licensed under the Academic Free License version 3.0
  See the file "LICENSE" for more information
*/

package sim.app.antsforage;

import sim.engine.SimState;

public /*strictfp*/ class GreedyDecisionMaker extends DecisionMaker
    {
    public DecisionInfo getHomeDecision( final SimState state )
        {

        int index;

        if( numInfos == 0 )
            {
            return null;
            }

        for( int i = 0 ; i < numInfos ; i++ )
            {
            processForHomeDecision( info[i] );
            }

        // compute the maximum value
        index = 0;
        for( int i = 0 ; i < numInfos ; i++ )
            if( info[i].profit > info[index].profit )
                index = i;
```

```
        int howMany = 0;
        for( int i = 0 ; i < numInfos ; i++ )
            if( info[i].profit == info[index].profit )
                howMany++;

        int x = state.random.nextInt( howMany );
        for( int i = 0 ; i < numInfos ; i++ )
            if( info[i].profit == info[index].profit )
                if( x == 0 )
                    return info[i];
                else
                    x--;
        return null;

        }

    }
```

ΠΑΡΑΡΤΗΜΑ Β:

ΟΛΟΚΛΗΡΩΜΕΝΟΣ ΚΩΔΙΚΑΣ ΕΙΔΙΚΩΝ ΚΛΑΣΕΩΝ ΑΛΓΟΡΙΘΜΟΥ AODV

Στο παράρτημα αυτό παραθέτουμε τις ειδικές κλάσεις που χρησιμοποιούνται για την υλοποίηση αποκλειστικά του αλγόριθμου AODV, όπως περιγράφηκε πιο πάνω στο έγγραφο αυτό.

Κλάση AntsForage

```
package sim.app.antsforage;

import sim.engine.*;
import sim.field.grid.*;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;


public class AntsForage extends SimState
{

	//Home location
	public static final int HOME_XMIN = 25;
	public static final int HOME_XMAX = 26;
	public static final int HOME_YMIN = 25;
	public static final int HOME_YMAX = 26;

	//Food location
	public static final int FOOD_XMIN = 55;
	public static final int FOOD_XMAX = 56;
	public static final int FOOD_YMIN = 63;
	public static final int FOOD_YMAX = 64;

	//Constants related to the simulation
	public static final int MAX_ANTS_PER_LOCATION = 10;
	public static final int TIME_TO_LIVE = 100;
	public static final int MAX_ANTS = 1000;
	public static final int INITIALANTS = 50;
	public static final int WORSE_FOOD_TRAIL = 500;
	public static final double MIN_PHEROMONE = 0.0;
	public static final double MAX_PHEROMONE = 1000.0;
	public static final double PHEROMONE_TO_LEAVE_BEHIND = 1;

	public static final double EVAPORATE_CONSTANT = 0.0001;
	public static final double DIFFUSION_CONSTANT = 0.0001;

	//Grid dimensions
```

```
      public static final int GRID_HEIGHT = 100;
      public static final int GRID_WIDTH = 100;

      //Existence of obstacles
      public static final int NO_OBSTACLES = 0;
      public static final int ONE_OBSTACLE = 1;
      public static final int TWO_OBSTACLES = 2;

      public static final int OBSTACLES = NO_OBSTACLES;

      public DoubleGrid2D sites = new DoubleGrid2D(GRID_WIDTH,
GRID_HEIGHT,0);
      public DoubleGrid2D toFoodGrid = new DoubleGrid2D(GRID_WIDTH,
GRID_HEIGHT,0);
      public DoubleGrid2D toHomeGrid = new DoubleGrid2D(GRID_WIDTH,
GRID_HEIGHT,0);
      public DoubleGrid2D valgrid2 = new DoubleGrid2D(GRID_WIDTH,
GRID_HEIGHT, 0);
      public SparseGrid2D buggrid = new SparseGrid2D(GRID_WIDTH,
GRID_HEIGHT);
      public DoubleGrid2D obstacles = new DoubleGrid2D(GRID_WIDTH,
GRID_HEIGHT,0);

      // a couple of objects to be shared by all ants in the simulation
      DecisionMaker decisionMaker = new DecisionMaker();
      DecisionInfo decisionInfo = new DecisionInfo();

      public AntsForage(long seed)
      {
            super(seed);
      }

      public int foodCollected = 0;

      public void start()
      {
            super.start();  // clear out the schedule

            // make new grids
            sites = new DoubleGrid2D(GRID_WIDTH, GRID_HEIGHT,0);
            toFoodGrid = new DoubleGrid2D(GRID_WIDTH, GRID_HEIGHT,0);
            toHomeGrid = new DoubleGrid2D(GRID_WIDTH, GRID_HEIGHT,0);
            valgrid2 = new DoubleGrid2D(GRID_WIDTH, GRID_HEIGHT, 0);
            buggrid = new SparseGrid2D(GRID_WIDTH, GRID_HEIGHT);
            obstacles = new DoubleGrid2D(GRID_WIDTH, GRID_HEIGHT, 0);

            //Food collected by the nest
            foodCollected = 0;

            //Draw the obstacles, if any
            switch( OBSTACLES )
            {
            case NO_OBSTACLES:
                  break;
            case ONE_OBSTACLE:
                  for( int x = 0 ; x < GRID_WIDTH ; x++ )
```

```
                        for( int y = 0 ; y < GRID_HEIGHT ; y++ )
                        {
                              obstacles.field[x][y] = 0.0;
                              if( ((x-55)*0.707+(y-35)*0.707)*((x-
55)*0.707+(y-35)*0.707)/36+
                                          ((x-55)*0.707-(y-
35)*0.707)*((x-55)*0.707-(y-35)*0.707)/1024 <= 1 )
                                    obstacles.field[x][y] = 1.0;
                        }
                  break;
            case TWO_OBSTACLES:
                  for( int x = 0 ; x < GRID_WIDTH ; x++ )
                        for( int y = 0 ; y < GRID_HEIGHT ; y++ )
                        {
                              obstacles.field[x][y] = 0.0;
                              if( ((x-45)*0.707+(y-25)*0.707)*((x-
45)*0.707+(y-25)*0.707)/36+
                                          ((x-45)*0.707-(y-
25)*0.707)*((x-45)*0.707-(y-25)*0.707)/1024 <= 1 )
                                    obstacles.field[x][y] = 1.0;
                              if( ((x-35)*0.707+(y-70)*0.707)*((x-
35)*0.707+(y-70)*0.707)/36+
                                          ((x-35)*0.707-(y-
70)*0.707)*((x-35)*0.707-(y-70)*0.707)/1024 <= 1 )
                                    obstacles.field[x][y] = 1.0;
                        }
                  break;
            }

            //Place home and food positions
            for( int x = HOME_XMIN ; x <= HOME_XMAX ; x++ )
                  for( int y = HOME_YMIN ; y <= HOME_YMAX ; y++ )
                        sites.field[x][y] = 1.0;
            for( int x = FOOD_XMIN ; x <= FOOD_XMAX ; x++ )
                  for( int y = FOOD_YMIN ; y <= FOOD_YMAX ; y++ )
                        sites.field[x][y] = 0.5;

            //Create the swarm of ants
            Steppable antFarm = new Steppable()
            {
                  public void step(SimState state)
                  {
                        try
                        {
                              //Open a text file to write the results
                              BufferedWriter ResultsFile = new
BufferedWriter(new FileWriter("Results.txt",true));

                              //Write the length of the best food trail
and each time step of the simulation
                              System.out.println("Best food trail at time
step: " + (int)state.schedule.getTime() + "\t"
                                          + bestFoodTrail);


      ResultsFile.write((int)state.schedule.getTime() + "\t"
```

```
                                                + bestFoodTrail + "\n");

                                    ResultsFile.flush();
                              }
                              catch ( IOException e )
                              {
                                    System.err.println("Error handling I/O
exceptions");
                                    return;
                              }
                        }
                  };

                  numberOfAnts = 0;

                  //Create the initial ants of the nest
                  for(int x=0;x<INITIALANTS;x++)
                  {
                        Ant bug = new Ant(random.nextInt(8),
                                    PHEROMONE_TO_LEAVE_BEHIND,
                                    MIN_PHEROMONE,
                                    MAX_PHEROMONE,
                                    TIME_TO_LIVE);

      buggrid.setObjectLocation(bug,(HOME_XMAX+HOME_XMIN)/2,(HOME_YMAX+HO
ME_YMIN)/2);
                        bug.toDiePointer = schedule.scheduleRepeating(bug);
                        numberOfAnts++;
                  }

                  // Schedule the decreaser to happen after the AntsForage
                  schedule.scheduleRepeating(Schedule.EPOCH,1,new
Diffuser(toHomeGrid,valgrid2,EVAPORATE_CONSTANT,DIFFUSION_CONSTANT),1);
                  schedule.scheduleRepeating(Schedule.EPOCH,1,new
Diffuser(toFoodGrid,valgrid2,EVAPORATE CONSTANT,DIFFUSION CONSTANT),1);
                  // Schedule the ant farm to happen after the AntsForage
                  schedule.scheduleRepeating(Schedule.EPOCH,1,antFarm,1);
            }

            //Total number of ants created for the simulation
            public int numberOfAnts = 0;
            //Number of ants that managed to find the location of food
            public int numberOfSuccessfulAnts = 0;
            //Number of hops needed to build the best food trail so far
            public int bestFoodTrail = WORSE_FOOD_TRAIL;

            ////////////////////////////////////////////////////////////////
            ////////////////////////// M A I N /////////////////////////////
            ////////////////////////////////////////////////////////////////
            public static void main(String[] args)
            {
                  doLoop(AntsForage.class, args);
                  System.exit(0);
            }
}
```

Κλάση AntsForageWithUI

```
package sim.app.antsforage;

import sim.engine.*;
import sim.display.*;
import sim.portrayal.grid.*;
import java.awt.*;

import javax.swing.*;

public class AntsForageWithUI extends GUIState
    {
    public Display2D display;
    public JFrame displayFrame;

    FastValueGridPortrayal2D homePheromonePortrayal = new
FastValueGridPortrayal2D("Home Pheromone");
    FastValueGridPortrayal2D foodPheromonePortrayal = new
FastValueGridPortrayal2D("Food Pheromone");
    FastValueGridPortrayal2D sitesPortrayal = new
FastValueGridPortrayal2D("Site", true);  // immutable
    FastValueGridPortrayal2D obstaclesPortrayal = new
FastValueGridPortrayal2D("Obstacle", true);  // immutable
    SparseGridPortrayal2D bugPortrayal = new SparseGridPortrayal2D();

    public static void main(String[] args)
        {
        AntsForageWithUI antsForage = new AntsForageWithUI();
        Console c = new Console(antsForage);
        c.setVisible(true);
        }

    public AntsForageWithUI() { super(new
AntsForage(System.currentTimeMillis())); }
    public AntsForageWithUI(SimState state) { super(state); }

    public static String getName() { return "(A)d hoc (O)n demand
(D)istance (V)ector"; }

    public void setupPortrayals()
        {
        AntsForage af = (AntsForage)state;

        //Grid settings and colors
        homePheromonePortrayal.setField(af.toHomeGrid);
        homePheromonePortrayal.setMap(new sim.util.gui.SimpleColorMap(
                                        AntsForage.MIN_PHEROMONE,
                                        AntsForage.MAX_PHEROMONE,
                                        new Color(26,50,250,40),
                                        new Color(0,255,0,255) ));
        foodPheromonePortrayal.setField(af.toFoodGrid);

        //Path color
        foodPheromonePortrayal.setMap(new sim.util.gui.SimpleColorMap(
```

```
                                        AntsForage.MIN_PHEROMONE,
                                        AntsForage.MAX_PHEROMONE,
                                        new Color(0,0,255,0),
                                        new Color(0,0,255,255) ));
        sitesPortrayal.setField(af.sites);

        sitesPortrayal.setMap(new sim.util.gui.SimpleColorMap(
                                0,
                                1,
                                new Color(0,0,0,0),
                                new Color(255,0,0,255) ));
        obstaclesPortrayal.setField(af.obstacles);

        obstaclesPortrayal.setMap(new sim.util.gui.SimpleColorMap(
                                    0,
                                    1,
                                    new Color(0,0,0,0),
                                    new Color(128,64,64,255) ));
        bugPortrayal.setField(af.buggrid);

        // reschedule the displayer
        display.reset();

        // redraw the display
        display.repaint();
        }

    public void start()
        {
        super.start();  // set up everything but replacing the display
        // set up our portrayals
        setupPortrayals();
        }

    public void load(SimState state)
        {
        super.load(state);
        // we now have new grids.  Set up the portrayals to reflect that
        setupPortrayals();
        }

    public void init(Controller c)
        {
        super.init(c);

        // Make the Display2D.  We'll have it display stuff later.
        display = new Display2D(400,400,this,1); // at 400x400, we've got
4x4 per array position
        displayFrame = display.createFrame();
        c.registerFrame(displayFrame);
        displayFrame.setVisible(true);

            // Settings about the "Display" menu
        display.attach(homePheromonePortrayal,"Pheromones To Home");
        display.attach(foodPheromonePortrayal,"Pheromones To Food");
        display.attach(sitesPortrayal,"Site Locations");
```

```
        display.attach(obstaclesPortrayal,"Obstacles");
        display.attach(bugPortrayal,"Agents");

        // specify the backdrop color
        display.setBackdrop(Color.white);
        }

    public void quit()
        {
        super.quit();

        if (displayFrame!=null) displayFrame.dispose();
        displayFrame = null;  // let gc
        display = null;       // let gc
        }

    }
```

Κλάση Ant

```
package sim.app.antsforage;

import sim.field.grid.*;
import sim.portrayal.*;
import sim.portrayal.simple.*;
import sim.util.*;
import sim.engine.*;
import java.awt.*;
import java.util.Random;

public class Ant extends OvalPortrayal2D implements Steppable
{
      public static final boolean GREEDY_REPOSITIONING = true;

      public static final boolean GREEDY_EXPLORATION = true;

      //Probability for the ant to broadcast
      public static final double BROADCAST_PROBABILITY = 0.8;

      //New ants created after a broadcast
      public static final int NEW_ANTS_PER_BROADCAST = 2;

      public static final int TIME_TO_LIVE = 100;

      //North Direction
      public static final int N = 0;
      //North-East Direction
      public static final int NE = 1;
      //East Direction
      public static final int E = 2;
      //South-East Direction
      public static final int SE = 3;
```

```
	//South Direction
	public static final int S = 4;
	//South-West Direction
	public static final int SW = 5;
	//West Direction
	public static final int W = 6;
	//North-West Direction
	public static final int NW = 7;

	public double pheromoneToLeaveBehind;
	public double minPheromone;
	public double maxPheromone;
	public int timeToLive;

	double subtractingRatio;
	double pheromoneRatio;

	int orientation;

	//GET/SET isBackward
	public boolean getIfIsBackward() { return isBackward; }
	public void setIfIsBackward(boolean val) { isBackward = val; }
	//GET/SET hasReachedFood
	public boolean gethasReachedFood() { return hasReachedFood; }
	public void setHasReachedFood(boolean val) { hasReachedFood = val;
}

	public boolean isBackward;
	public boolean hasReachedFood;
	public int timesReachedFood;
	public int foodTrail;

	////////////////////////////////////////////////////////////
	////////////////////////// A N T ///////////////////////////
	////////////////////////////////////////////////////////////
	public Ant( int orientation,
			double pheromoneToLeaveBehind,
			double minPheromone,
			double maxPheromone,
			int timeToLive)
	{
		//Initialise the parameters based on the values sent from the
nest
		this.orientation = orientation;
		this.pheromoneToLeaveBehind = pheromoneToLeaveBehind;
		this.minPheromone = minPheromone;
		this.maxPheromone = maxPheromone;
		this.timeToLive = timeToLive;

		subtractingRatio = ( 1.0 / timeToLive ) * maxPheromone;
		pheromoneRatio = 1.0 * maxPheromone;

		//At start, the ant has never reached food, so it's still a
forward ant
		isBackward = false;
		hasReachedFood = false;
```

```
            timesReachedFood = 0;
      }

      ////////////////////////////////////////////////////////////////////
/////////////////
      ////////////////// ADD INFORMATION FOR EVERY PLACE OF THE ANT (x,y)
/////////////////
      ////////////////////////////////////////////////////////////////////
/////////////////
      protected void addInformation( final SimState state, int x, int y,
final int orientation )
      {
            final AntsForage af = (AntsForage)state;
            final DecisionInfo di = af.decisionInfo;
            final DecisionMaker decisionMaker = af.decisionMaker;

            if( x < 0 || x >= AntsForage.GRID_WIDTH || y < 0 || y >=
AntsForage.GRID_HEIGHT )
                  return;

            if( ( af.buggrid.getObjectsAtLocation(x,y) == null ||
                        af.buggrid.getObjectsAtLocation(x,y).numObjs <
AntsForage.MAX_ANTS_PER_LOCATION ) &&
                        af.obstacles.field[x][y] <= 0.5 )
            {
                  //set coordinates
                  di.position.x = x;
                  di.position.y = y;
                  di.orientation = orientation;

                  //Add pheromones to current location of ant
                  di.homePheromoneAmount = 0.001 +
af.toHomeGrid.field[di.position.x][di.position.y];
                  di.foodPheromoneAmount = 0.001 +
af.toFoodGrid.field[di.position.x][di.position.y];
                  decisionMaker.addInfo( di );
            }
      }

      ////////////////////////////////////////////////////////////////////
/////////////////
      ////////////// DEPENDING ON THE PLACE OF THE ANT (x,y), DECIDE NEXT
ACTION /////////
      ////////////////////////////////////////////////////////////////////
/////////////////
      public DecisionInfo decideAction( final SimState state, final int
myx, final int myy, final int orientation )
      {

            final AntsForage af = (AntsForage)state;
            final DecisionMaker decisionMaker = af.decisionMaker;

            decisionMaker.reset();

            Random random = new Random();
```

```
            //Depending on a probability, we decide if the ant is going
to execute a broadcast
            if ( random.nextDouble() >= BROADCAST_PROBABILITY )
            {
                  for(int x=0 ; x<NEW_ANTS_PER_BROADCAST &&
af.numberOfAnts < af.MAX_ANTS; x++)
                  {
                        Ant bug = new Ant( random.nextInt(8),
                                    pheromoneToLeaveBehind,
                                    minPheromone,
                                    maxPheromone,
                                    TIME_TO_LIVE );


      af.buggrid.setObjectLocation(bug,(af.HOME_XMAX+af.HOME_XMIN)/2,(af.
HOME_YMAX+af.HOME_YMIN)/2);
                        bug.toDiePointer =
state.schedule.scheduleRepeating(bug);
                        af.numberOfAnts++;
                  }
            }

            switch( orientation )
            {
            case N: addInformation( state, myx-1, myy+1,
(orientation+7)%8 );  // forward-left
            addInformation( state, myx,   myy+1, orientation );        //
forward
            addInformation( state, myx+1, myy+1, (orientation+1)%8 );  //
forward-right
            break;
            case NE: addInformation( state, myx,   myy+1,
(orientation+7)%8 );  // forward-left
            addInformation( state, myx+1, myy+1, orientation );        //
forward
            addInformation( state, myx+1, myy,   (orientation+1)%8 );  //
forward-right
            break;
            case E: addInformation( state, myx+1, myy+1,
(orientation+7)%8 );  // forward-left
            addInformation( state, myx+1, myy,   orientation );        //
forward
            addInformation( state, myx+1, myy-1, (orientation+1)%8 );  //
forward-right
            break;
            case SE: addInformation( state, myx+1, myy,
(orientation+7)%8 );  // forward-left
            addInformation( state, myx+1, myy-1, orientation );        //
forward
            addInformation( state, myx,   myy-1, (orientation+1)%8 );  //
forward-right
            break;
            case S: addInformation( state, myx+1, myy-1,
(orientation+7)%8 );  // forward-left
            addInformation( state, myx,   myy-1, orientation );        //
forward
```

```
            addInformation( state, myx-1, myy-1, (orientation+1)%8 );  //
forward-right
            break;
            case SW: addInformation( state, myx,   myy-1,
(orientation+7)%8 );  // forward-left
            addInformation( state, myx-1, myy-1, orientation );        //
forward
            addInformation( state, myx-1, myy,   (orientation+1)%8 );  //
forward-right
            break;
            case W: addInformation( state, myx-1, myy-1,
(orientation+7)%8 );  // forward-left
            addInformation( state, myx-1, myy,   orientation );        //
forward
            addInformation( state, myx-1, myy+1, (orientation+1)%8 );  //
forward-right
            break;
            case NW: addInformation( state, myx-1, myy,
(orientation+7)%8 );  // forward-left
            addInformation( state, myx-1, myy+1, orientation );        //
forward
            addInformation( state, myx,   myy+1, (orientation+1)%8 );  //
forward-right
            break;
            }

            if( isBackward && hasReachedFood )
                  return decisionMaker.getHomeGreedyDecision( state );
            else
            {
                  if( GREEDY_EXPLORATION )
                        return decisionMaker.getFoodGreedyDecision( state
);
                  else
                        return decisionMaker.getFoodDecision( state );
            }
      }

      ////////////////////////////////////////////////////////////////////
////////////////
      ////////////////////// ADD PHEROMONES TO THE GRID AT (x,y)
//////////////////////
      ////////////////////////////////////////////////////////////////////
////////////////
      public void addPheromone(DoubleGrid2D grid, int x, int y, double
pheromone)
      {
            double amount = Math.max( grid.field[x][y], pheromone );

            //Depending on the location of ant decide the amount of
pheromone to be left at location x,y
            if( x > 0 && y > 0 )
                  amount = Math.max( Math.max( grid.field[x-1][y-1]-
subtractingRatio, minPheromone ), amount );

            if( x > 0)
```

```
                  amount = Math.max( Math.max( grid.field[x-1][y]-
subtractingRatio, minPheromone ), amount );

            if( x > 0 && y < grid.field[x].length-1 )
                  amount = Math.max( Math.max( grid.field[x-1][y+1]-
subtractingRatio, minPheromone ), amount );

            if( y > 0 )
                  amount = Math.max( Math.max( grid.field[x][y-1]-
subtractingRatio, minPheromone ), amount );

            if( y < grid.field.length-1 )
                  amount = Math.max( Math.max( grid.field[x][y+1]-
subtractingRatio, minPheromone ), amount );

            if( x < grid.field.length-1 && y > 0 )
                  amount = Math.max( Math.max( grid.field[x+1][y-1]-
subtractingRatio, minPheromone ), amount );

            if( x < grid.field.length-1 )
                  amount = Math.max( Math.max( grid.field[x+1][y]-
subtractingRatio, minPheromone ), amount );

            if( x < grid.field.length-1 && y < grid.field[x].length-1 )
                  amount = Math.max( Math.max( grid.field[x+1][y+1]-
subtractingRatio, minPheromone ), amount );

            //Deposit pheromone amount at x,y of the grid
            grid.field[x][y] = amount;
            pheromoneRatio = amount;
      }

      ////////////////////////////////////////////////////////////////////
/////////////////
      ////////////// DEPENDING ON THE STATUS OF THE ANT, DECIDE NEXT
ACTION ////////////
      ////////////////////////////////////////////////////////////////////
/////////////////
      public DecisionInfo decideGreedyAction( final SimState state, final
int myx, final int myy, final int orientation )
      {

            final AntsForage af = (AntsForage)state;
            final DecisionMaker decisionMaker = af.decisionMaker;

            decisionMaker.reset();

            //Locate all 8 new neighbours
            addInformation( state, myx,   myy+1, N );  //move North
            addInformation( state, myx+1, myy+1, NE ); //move North-East
            addInformation( state, myx+1, myy,   E );  //move East
            addInformation( state, myx+1, myy-1, SE ); //move South-East
            addInformation( state, myx,   myy-1, S );  //move South
            addInformation( state, myx-1, myy-1, SW ); //move South-East
            addInformation( state, myx-1, myy,   W );  //move West
            addInformation( state, myx-1, myy+1, NW ); //move North-West
```

```
            if( isBackward && hasReachedFood )
                  return decisionMaker.getHomeGreedyDecision( state );
            else
                  return decisionMaker.getFoodGreedyDecision( state );

      }

      ////////////////////////////////////////////////
      ////////////// NORMAL MOVE OF THE ANT //////////
      ////////////////////////////////////////////////
      public void step( final SimState state )
      {
            final AntsForage af = (AntsForage)state;
            final DecisionMaker decisionMaker = af.decisionMaker;

            Int2D location = af.buggrid.getObjectLocation(this);
            int myx = location.x;
            int myy = location.y;

            int bestx, besty, besto;

            DecisionInfo movingDecision = null;

            //If has found food more than once
            if( isBackward && hasReachedFood )
            {
                  movingDecision = decideGreedyAction( state, myx, myy,
orientation );
                  ++foodTrail;
            }
            else
            {
                  decisionMaker.reset();
                  addInformation( state, myx,   myy+1, 0 );
                  addInformation( state, myx+1, myy+1, 1 );
                  addInformation( state, myx+1, myy,   2 );
                  addInformation( state, myx+1, myy-1, 3 );
                  addInformation( state, myx,   myy-1, 4 );
                  addInformation( state, myx-1, myy-1, 5 );
                  addInformation( state, myx-1, myy,   6 );
                  addInformation( state, myx-1, myy+1, 7 );

                  int max = 0;
                  int howMany = 1;

                  for( int i = 1 ; i < decisionMaker.numInfos ; i++ )
                        if( decisionMaker.info[max].foodPheromoneAmount
==
                              decisionMaker.info[i].foodPheromoneAmount )
                        {
                              howMany++;
                        }
                        else if(
decisionMaker.info[max].foodPheromoneAmount <
```

```
      decisionMaker.info[i].foodPheromoneAmount )
                        {
                              max = i;
                              howMany = 1;
                        }
                  if( howMany == 1 )
                        movingDecision = decideGreedyAction( state, myx,
myy, orientation );
                  else
                        movingDecision = decideAction( state, myx, myy,
orientation );

                  if (isBackward)
                        ++foodTrail;
            }
            if( movingDecision == null )
            {
                  movingDecision = decideGreedyAction( state, myx, myy,
orientation );
                  if( movingDecision == null )
                  {
                        bestx = myx;
                        besty = myy;
                        besto = orientation;
                  }
                  else
                  {
                        bestx = movingDecision.position.x;
                        besty = movingDecision.position.y;
                        besto = movingDecision.orientation;
                  }
                  if (isBackward)
                        ++foodTrail;
            }
            else
            {
                  bestx = movingDecision.position.x;
                  besty = movingDecision.position.y;
                  besto = movingDecision.orientation;

                  if (isBackward)
                        ++foodTrail;
            }

            //Add some pheromones
            if( ( bestx != myx || besty != myy ))
            {
                  //FOOD PHEROMONES
                  if( isBackward && hasReachedFood ) {

      addPheromone(af.toFoodGrid,myx,myy,(pheromoneRatio));
                        ++foodTrail;
                  }

                  //HOME PHEROMONES
                  else
```

```
      addPheromone(af.toHomeGrid,myx,myy,(pheromoneRatio));

                  if( bestx != myx && besty != myy )
                        pheromoneRatio -= subtractingRatio*1.4142;
                  else
                        pheromoneRatio -= subtractingRatio;

                  if( pheromoneRatio < 0 )
                  {
                        die( state );
                        return;
                  }

                  //Place the ant and deposit food and home pheromones
                  af.buggrid.setObjectLocation(this,bestx,besty);
                  orientation = besto;

                  if( ( besty >= AntsForage.HOME_YMIN ) && ( besty <=
AntsForage.HOME_YMAX ) &&
                              ( bestx >= AntsForage.HOME_XMIN ) && (
bestx <= AntsForage.HOME_XMAX ) )
                  {
                        //If the ant has found the food more than once
                        if( isBackward && hasReachedFood )
                        {
                              af.foodCollected++;
                              isBackward = false;
                              hasReachedFood = false;
                              pheromoneRatio = 1.0 * maxPheromone;

                              if (foodTrail < af.bestFoodTrail)
                                    af.bestFoodTrail = foodTrail;

                              if( GREEDY_REPOSITIONING )
                              {
                                    DecisionInfo temp =
decideGreedyAction( state, myx, myy, orientation );
                                    if( temp != null )
                                          orientation = temp.orientation;
                                    else
                                          orientation =
(orientation+4)%8;
                              }
                              else
                                    orientation = (orientation+4)%8;
                        }
                  }
                  else if(  ( besty >= AntsForage.FOOD_YMIN ) && ( besty
<= AntsForage.FOOD_YMAX ) &&
                              ( bestx >= AntsForage.FOOD_XMIN ) && (
bestx <= AntsForage.FOOD_XMAX ) )
                  {
                        //If the ant has never found the food
                        if( !isBackward && !hasReachedFood )
                        {
```

```
                              foodTrail = 0;
                              isBackward = true;
                              hasReachedFood = true;
                              timesReachedFood++;

                              ++foodTrail;

                              //Count each ant only once
                              if ( timesReachedFood == 1 )
                                    //Increase the number of ants that
found the food
                                    af.numberOfSuccessfulAnts++;

                              pheromoneRatio = 1.0 * maxPheromone;

                              if( GREEDY_REPOSITIONING )
                              {
                                    DecisionInfo temp =
decideGreedyAction( state, myx, myy, orientation );
                                    if( temp != null )
                                          orientation = temp.orientation;
                                    else
                                          orientation =
(orientation+4)%8;
                                    if (isBackward)
                                          ++foodTrail;
                              }
                              else
                                    orientation = (orientation+4)%8; //
rotate 180
                        }
                  }

                  timeToLive--;
                  if( timeToLive <= 0 )
                  {
                        die( state );
                        return;
                  }
            }
      }

      //Ants that haven't ever reached the food and are in the path
      private Color noFoodColor = Color.black;
      //Ants that have reached the food sometime and are in the path
      private Color foodColor = Color.white;
      public final void draw(Object object, Graphics2D graphics,
DrawInfo2D info)
      {
            if( isBackward && hasReachedFood )
                  graphics.setColor( foodColor );
            else
                  graphics.setColor( noFoodColor );

            int x = (int)(info.draw.x - info.draw.width / 2.0);
            int y = (int)(info.draw.y - info.draw.height / 2.0);
```

```
            int width = (int)(info.draw.width);
            int height = (int)(info.draw.height);
            graphics.fillOval(x,y,width, height);
      }

      public Stoppable toDiePointer = null;
      public void die( final SimState state )
      {
            AntsForage antsforage = (AntsForage)state;
            antsforage.numberOfAnts--;
            antsforage.buggrid.remove( this );
            if(toDiePointer!=null) toDiePointer.stop();
      }

}
```

ΠΑΡΑΡΤΗΜΑ Γ:

ΟΛΟΚΛΗΡΩΜΕΝΟΣ ΚΩΔΙΚΑΣ ΑΛΓΟΡΙΘΜΟΥ AntHocNet

Στο παράρτημα αυτό παραθέτουμε τις ειδικές κλάσεις που χρησιμοποιούνται για την υλοποίηση αποκλειστικά του αλγόριθμου AntHocNet, όπως περιγράφηκε πιο πάνω στο έγγραφο αυτό.

Κλάση AntsForage

```
package sim.app.antsforage;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

import sim.engine.*;
import sim.field.grid.*;
import sim.util.Int2D;


public class AntsForage extends SimState
{

	//Home location
	public static final int HOME_XMIN = 25;
	public static final int HOME_XMAX = 26;
	public static final int HOME_YMIN = 25;
	public static final int HOME_YMAX = 26;

	//Food location
	public static final int FOOD_XMIN = 55;
	public static final int FOOD_XMAX = 56;
	public static final int FOOD_YMIN = 63;
	public static final int FOOD_YMAX = 64;

	//Constants related to the simulation
	public static final int MAX_ANTS_PER_LOCATION = 10;
	public static final int TIME_TO_LIVE = 1000;
	public static final int MAX_ANTS = 1500;
	public static final int INITIALANTS = 2;
	public static final int WORSE_FOOD_TRAIL = 500;
	public static final double MIN_PHEROMONE = 0.0;
	public static final double MAX_PHEROMONE = 1000.0;
	public static final double PHEROMONE_TO_LEAVE_BEHIND = 1;

	public static final double EVAPORATE_CONSTANT = 0.0001;
	public static final double DIFFUSION_CONSTANT = 0.0001;
```

```
      //Grid dimensions
      public static final int GRID_HEIGHT = 100;
      public static final int GRID_WIDTH = 100;

      //Existence of obstacles
      public static final int NO_OBSTACLES = 0;
      public static final int ONE_OBSTACLE = 1;
      public static final int TWO_OBSTACLES = 2;

      public static final int OBSTACLES = NO_OBSTACLES;

      public DoubleGrid2D sites = new DoubleGrid2D(GRID_WIDTH,
GRID_HEIGHT,0);
      public DoubleGrid2D toFoodGrid = new DoubleGrid2D(GRID_WIDTH,
GRID_HEIGHT,0);
      public DoubleGrid2D toHomeGrid = new DoubleGrid2D(GRID_WIDTH,
GRID_HEIGHT,0);
      public DoubleGrid2D valgrid2 = new DoubleGrid2D(GRID_WIDTH,
GRID_HEIGHT, 0);
      public SparseGrid2D buggrid = new SparseGrid2D(GRID_WIDTH,
GRID_HEIGHT);
      public DoubleGrid2D obstacles = new DoubleGrid2D(GRID_WIDTH,
GRID_HEIGHT,0);

      // a couple of objects to be shared by all ants in the simulation
      DecisionMaker decisionMaker = new DecisionMaker();
      DecisionInfo decisionInfo = new DecisionInfo();

      public AntsForage(long seed)
      {
            super(seed);
      }

      public int foodCollected = 0;

      public void start()
      {
            super.start();  // clear out the schedule

            // make new grids
            sites = new DoubleGrid2D(GRID_WIDTH, GRID_HEIGHT,0);
            toFoodGrid = new DoubleGrid2D(GRID_WIDTH, GRID_HEIGHT,0);
            toHomeGrid = new DoubleGrid2D(GRID_WIDTH, GRID_HEIGHT,0);
            valgrid2 = new DoubleGrid2D(GRID_WIDTH, GRID_HEIGHT, 0);
            buggrid = new SparseGrid2D(GRID_WIDTH, GRID_HEIGHT);
            obstacles = new DoubleGrid2D(GRID_WIDTH, GRID_HEIGHT, 0);

            //Food collected by the nest
            foodCollected = 0;

            //Draw the obstacles, if any
            switch( OBSTACLES )
            {
            case NO_OBSTACLES:
                  break;
```

```
            case ONE_OBSTACLE:
                  for( int x = 0 ; x < GRID_WIDTH ; x++ )
                        for( int y = 0 ; y < GRID_HEIGHT ; y++ )
                        {
                              obstacles.field[x][y] = 0.0;
                              if( ((x-55)*0.707+(y-35)*0.707)*((x-
55)*0.707+(y-35)*0.707)/36+
                                          ((x-55)*0.707-(y-
35)*0.707)*((x-55)*0.707-(y-35)*0.707)/1024 <= 1 )
                                    obstacles.field[x][y] = 1.0;
                        }
                  break;
            case TWO_OBSTACLES:
                  for( int x = 0 ; x < GRID_WIDTH ; x++ )
                        for( int y = 0 ; y < GRID_HEIGHT ; y++ )
                        {
                              obstacles.field[x][y] = 0.0;
                              if( ((x-45)*0.707+(y-25)*0.707)*((x-
45)*0.707+(y-25)*0.707)/36+
                                          ((x-45)*0.707-(y-
25)*0.707)*((x-45)*0.707-(y-25)*0.707)/1024 <= 1 )
                                    obstacles.field[x][y] = 1.0;
                              if( ((x-35)*0.707+(y-70)*0.707)*((x-
35)*0.707+(y-70)*0.707)/36+
                                          ((x-35)*0.707-(y-
70)*0.707)*((x-35)*0.707-(y-70)*0.707)/1024 <= 1 )
                                    obstacles.field[x][y] = 1.0;
                        }
                  break;
            }

            //Place home and food positions
            for( int x = HOME_XMIN ; x <= HOME_XMAX ; x++ )
                  for( int y = HOME_YMIN ; y <= HOME_YMAX ; y++ )
                        sites.field[x][y] = 1.0;
            for( int x = FOOD_XMIN ; x <= FOOD_XMAX ; x++ )
                  for( int y = FOOD_YMIN ; y <= FOOD_YMAX ; y++ )
                        sites.field[x][y] = 0.5;

            //Create the swarm of ants
            Steppable antFarm = new Steppable()
            {
                  public void step(SimState state)
                  {
                        try
                        {
                              //Open a text file to write the results
                              BufferedWriter ResultsFile = new
BufferedWriter(new FileWriter("Results.txt",true));

                              //Write the length of the best food trail
and each time step of the simulation
                              System.out.println("Best food trail at time
step: " + (int)state.schedule.getTime() + "\t"
                                          + bestFoodTrail);
```

```
      ResultsFile.write((int)state.schedule.getTime() + "\t"
                                                + bestFoodTrail + "\n");

                              ResultsFile.flush();
                        }
                        catch ( IOException e )
                        {
                              System.err.println("Error handling I/O
exceptions");
                              return;
                        }
                  }
            };

            //Create the initial reactive and proactive ants of the nest
            for(int x=0;x<INITIALANTS;x++)
            {
                  ReactiveAnt bug = new ReactiveAnt(random.nextInt(8),
                              PHEROMONE_TO_LEAVE_BEHIND,
                              MIN_PHEROMONE,
                              MAX_PHEROMONE,
                              TIME_TO_LIVE);


      buggrid.setObjectLocation(bug,(HOME_XMAX+HOME_XMIN)/2,(HOME_YMAX+HO
ME_YMIN)/2);
                  bug.toDiePointer = schedule.scheduleRepeating(bug);
                  numberOfAnts++;
            }

            for(int x=0;x<INITIALANTS;x++)
            {
                  ProactiveAnt bug = new ProactiveAnt(random.nextInt(8),
                              PHEROMONE TO LEAVE BEHIND,
                              MIN_PHEROMONE,
                              MAX_PHEROMONE,
                              TIME_TO_LIVE);


      buggrid.setObjectLocation(bug,(HOME_XMAX+HOME_XMIN)/2,(HOME_YMAX+HO
ME_YMIN)/2);
                  bug.toDiePointer = schedule.scheduleRepeating(bug);
                  numberOfAnts++;
            }

            // Schedule the decreaser to happen after the AntsForage
            schedule.scheduleRepeating(Schedule.EPOCH,1,new
Diffuser(toHomeGrid,valgrid2,EVAPORATE_CONSTANT,DIFFUSION_CONSTANT),1);
            schedule.scheduleRepeating(Schedule.EPOCH,1,new
Diffuser(toFoodGrid,valgrid2,EVAPORATE_CONSTANT,DIFFUSION_CONSTANT),1);
            // Schedule the ant farm to happen after the AntsForage
            schedule.scheduleRepeating(Schedule.EPOCH,1,antFarm,1);
      }

      //Total number of ants created for the simulation
```

```
	public int numberOfAnts = 0;
	//Number of ants that managed to find the location of food
	public int numberOfSuccessfulAnts = 0;
	//Number of hops needed to build the best food trail so far
	public int bestFoodTrail = WORSE_FOOD_TRAIL;

	////////////////////////////////////////////////////////////////
	////////////////////////// M A I N /////////////////////////////
	////////////////////////////////////////////////////////////////
	public static void main(String[] args)
	{
		doLoop(AntsForage.class, args);
		System.exit(0);
	}
}
```

Κλάση AntsForageWithUI

```
package sim.app.antsforage;

import sim.engine.*;
import sim.display.*;
import sim.portrayal.grid.*;
import java.awt.*;

import javax.swing.*;

public class AntsForageWithUI extends GUIState
{
	public Display2D display;
	public JFrame displayFrame;

	FastValueGridPortrayal2D homePheromonePortrayal = new
FastValueGridPortrayal2D("Home Pheromone");
	FastValueGridPortrayal2D foodPheromonePortrayal = new
FastValueGridPortrayal2D("Food Pheromone");
	FastValueGridPortrayal2D sitesPortrayal = new
FastValueGridPortrayal2D("Site", true);  // immutable
	FastValueGridPortrayal2D obstaclesPortrayal = new
FastValueGridPortrayal2D("Obstacle", true);  // immutable
	SparseGridPortrayal2D bugPortrayal = new SparseGridPortrayal2D();

	public static void main(String[] args)
	{
		AntsForageWithUI antsForage = new AntsForageWithUI();
		Console c = new Console(antsForage);
		c.setVisible(true);
	}

	public AntsForageWithUI() { super(new
AntsForage(System.currentTimeMillis())); }
	public AntsForageWithUI(SimState state) { super(state); }
```

```
      public static String getName() { return "AntHocNet"; }

      public void setupPortrayals()
      {
            AntsForage af = (AntsForage)state;

            homePheromonePortrayal.setField(af.toHomeGrid);
            homePheromonePortrayal.setMap(new
sim.util.gui.SimpleColorMap(
                        AntsForage.MIN_PHEROMONE,
                        AntsForage.MAX_PHEROMONE,
                        //Color of the pheromones
                        //new Color(26,50,33,45),
                        //lightBlue
                        new Color(26,50,250,40),
                        new Color(0,255,0,255) ));
            foodPheromonePortrayal.setField(af.toFoodGrid);

            //Path color
            foodPheromonePortrayal.setMap(new
sim.util.gui.SimpleColorMap(
                        AntsForage.MIN_PHEROMONE,
                        AntsForage.MAX_PHEROMONE,
                        new Color(0,0,255,0),
                        new Color(0,0,255,255) ));
            sitesPortrayal.setField(af.sites);

            sitesPortrayal.setMap(new sim.util.gui.SimpleColorMap(
                        0,
                        1,
                        new Color(0,0,0,0),
                        new Color(255,0,0,255) ));
            obstaclesPortrayal.setField(af.obstacles);

            obstaclesPortrayal.setMap(new sim.util.gui.SimpleColorMap(
                        0,
                        1,
                        new Color(0,0,0,0),
                        new Color(128,64,64,255) ));
            bugPortrayal.setField(af.buggrid);

            // reschedule the displayer
            display.reset();

            // redraw the display
            display.repaint();
      }

      public void start()
      {
            super.start();  // set up everything but replacing the
display
            // set up our portrayals
            setupPortrayals();
      }
```

```
	public void load(SimState state)
	{
		super.load(state);
		// we now have new grids.  Set up the portrayals to reflect
that
		setupPortrayals();
	}

	public void init(Controller c)
	{
		super.init(c);

		// Make the Display2D.  We'll have it display stuff later.
		display = new Display2D(400,400,this,1); // at 400x400, we've
got 4x4 per array position
		displayFrame = display.createFrame();
		c.registerFrame(displayFrame);
		displayFrame.setVisible(true);

		// Settings about the "Display" menu
		display.attach(homePheromonePortrayal,"Pheromones To Home");
		display.attach(foodPheromonePortrayal,"Pheromones To Food");
		display.attach(sitesPortrayal,"Site Locations");
		display.attach(obstaclesPortrayal,"Obstacles");
		display.attach(bugPortrayal,"Agents");

		// specify the backdrop color
		display.setBackdrop(Color.white);
	}

	public void quit()
	{
		super.quit();

		if (displayFrame!=null) displayFrame.dispose();
		displayFrame = null;  // let gc
		display = null;       // let gc
	}

}
```

Κλάση ProactiveAnt

```
package sim.app.antsforage;

import java.awt.Color;
import java.awt.Graphics2D;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Random;
```

```
import sim.field.grid.*;
import sim.portrayal.*;
import sim.portrayal.simple.*;
import sim.util.*;
import sim.engine.*;

public class ProactiveAnt extends OvalPortrayal2D implements Steppable
{
      public static final boolean GREEDY_REPOSITIONING = true;

      public static final boolean GREEDY_EXPLORATION = true;

      //Probability for the ant to broadcast
      public static final double BROADCAST_PROBABILITY = 0.5;

      //New ants created after a broadcast
      public static final int NEW_ANTS_PER_BROADCAST = 10;

      //Maximum number of broadcasts allowed for every ant
      public static final int MAX_BROADCASTS = 2;

      //North Direction
      public static final int N = 0;
      //North-East Direction
      public static final int NE = 1;
      //East Direction
      public static final int E = 2;
      //South-East Direction
      public static final int SE = 3;
      //South Direction
      public static final int S = 4;
      //South-West Direction
      public static final int SW = 5;
      //West Direction
      public static final int W = 6;
      //North-West Direction
      public static final int NW = 7;

      public double pheromoneToLeaveBehind;
      public double minPheromone;
      public double maxPheromone;
      public int timeToLive;

      double subtractingRatio;
      double broadcastProbability;
      double pheromoneRatio;

      int orientation;

      //GET/SET isBackward
      public boolean getIfIsBackward() { return isBackward; }
      public void setIfIsBackward(boolean val) { isBackward = val; }

      //GET/SET hasReachedFood
      public boolean getHasReachedFood() { return hasReachedFood; }
```

```
      public void setHasReachedFood(boolean val) { hasReachedFood = val;
}

      public boolean isBackward;
      public boolean hasReachedFood;
      public int timesReachedFood;
      public int numberOfBroadcasts;
      public int foodTrail;

      ////////////////////////////////////////////////////////////////////
/
      ////////////////////// P R O A C T I V E   A N T
//////////////////
      ////////////////////////////////////////////////////////////////////
/
      public ProactiveAnt( int orientation,
                  double pheromoneToLeaveBehind,
                  double minPheromone,
                  double maxPheromone,
                  int timeToLive )
      {
            //Initialise the parameters based on the values sent from the
nest
            this.orientation = orientation;
            this.pheromoneToLeaveBehind = pheromoneToLeaveBehind;
            this.minPheromone = minPheromone;
            this.maxPheromone = maxPheromone;
            this.timeToLive = timeToLive;

            subtractingRatio = ( 1.0 / timeToLive ) * maxPheromone;
            pheromoneRatio = 1.0 * maxPheromone;

            //At start, the ant has never reached food, so it's still a
forward ant
            isBackward = false;
            hasReachedFood = false;
            timesReachedFood = 0;

            //At start, the ant has never broadcasted
            numberOfBroadcasts = 0;
      }

      ////////////////////////////////////////////////////////////////////
/////////////////
      ////////////////// ADD INFORMATION FOR EVERY PLACE OF THE ANT (x,y)
/////////////////
      ////////////////////////////////////////////////////////////////////
/////////////////
      protected void addInformation( final SimState state, int x, int y,
final int orientation )
      {
            final AntsForage af = (AntsForage)state;
            final DecisionInfo di = af.decisionInfo;
            final DecisionMaker decisionMaker = af.decisionMaker;

            if( x < 0 || x >= AntsForage.GRID_WIDTH || y < 0 || y >=
```

```
AntsForage.GRID_HEIGHT )
                  return;

            if( ( af.buggrid.getObjectsAtLocation(x,y) == null ||
                        af.buggrid.getObjectsAtLocation(x,y).numObjs <
AntsForage.MAX_ANTS_PER_LOCATION ) &&
                        af.obstacles.field[x][y] <= 0.5 )
            {
                  //set coordinates
                  di.position.x = x;
                  di.position.y = y;
                  di.orientation = orientation;

                  //Add pheromones to current location of ant
                  di.homePheromoneAmount = 0.001 +
af.toHomeGrid.field[di.position.x][di.position.y];
                  di.foodPheromoneAmount = 0.001 +
af.toFoodGrid.field[di.position.x][di.position.y];
                  decisionMaker.addInfo( di );
            }
      }

      ////////////////////////////////////////////////////////////////////
////////////////
      ////////////// DEPENDING ON THE PLACE OF THE ANT (x,y), DECIDE NEXT
ACTION /////////
      ////////////////////////////////////////////////////////////////////
////////////////
      public DecisionInfo decideAction( final SimState state, final int
myx, final int myy, final int orientation )
      {
            final AntsForage af = (AntsForage)state;
            final DecisionMaker decisionMaker = af.decisionMaker;

            decisionMaker.reset();

            Random random = new Random();

            //Depending on a probability, we decide if the ant is going
to execute a broadcast
            if ( random.nextDouble() >= BROADCAST_PROBABILITY &&
numberOfBroadcasts < MAX_BROADCASTS )
            {
                  ++numberOfBroadcasts;
                  for(int x=0 ; x<NEW_ANTS_PER_BROADCAST &&
af.numberOfAnts < af.MAX_ANTS; x++)
                  {
                        ProactiveAnt bug = new ProactiveAnt(
random.nextInt(8),
                                    pheromoneToLeaveBehind,
                                    minPheromone,
                                    maxPheromone,
                                    timeToLive );


      af.buggrid.setObjectLocation(bug,(af.HOME_XMAX+af.HOME_XMIN)/2,(af.
```

```
HOME_YMAX+af.HOME_YMIN)/2);
                        bug.toDiePointer =
state.schedule.scheduleRepeating(bug);
                        af.numberOfAnts++;
                    }
            }

            switch( orientation )
            {
            case N: addInformation( state, myx-1, myy+1,
(orientation+7)%8 );  // forward-left
            addInformation( state, myx,   myy+1, orientation );        //
forward
            addInformation( state, myx+1, myy+1, (orientation+1)%8 );  //
forward-right
            break;
            case NE: addInformation( state, myx,   myy+1,
(orientation+7)%8 );  // forward-left
            addInformation( state, myx+1, myy+1, orientation );        //
forward
            addInformation( state, myx+1, myy,   (orientation+1)%8 );  //
forward-right
            break;
            case E: addInformation( state, myx+1, myy+1,
(orientation+7)%8 );  // forward-left
            addInformation( state, myx+1, myy,   orientation );        //
forward
            addInformation( state, myx+1, myy-1, (orientation+1)%8 );  //
forward-right
            break;
            case SE: addInformation( state, myx+1, myy,
(orientation+7)%8 );  // forward-left
            addInformation( state, myx+1, myy-1, orientation );        //
forward
            addInformation( state, myx,   myy-1, (orientation+1)%8 );  //
forward-right
            break;
            case S: addInformation( state, myx+1, myy-1,
(orientation+7)%8 );  // forward-left
            addInformation( state, myx,   myy-1, orientation );        //
forward
            addInformation( state, myx-1, myy-1, (orientation+1)%8 );  //
forward-right
            break;
            case SW: addInformation( state, myx,   myy-1,
(orientation+7)%8 );  // forward-left
            addInformation( state, myx-1, myy-1, orientation );        //
forward
            addInformation( state, myx-1, myy,   (orientation+1)%8 );  //
forward-right
            break;
            case W: addInformation( state, myx-1, myy-1,
(orientation+7)%8 );  // forward-left
            addInformation( state, myx-1, myy,   orientation );        //
forward
            addInformation( state, myx-1, myy+1, (orientation+1)%8 );  //
```

```
forward-right
            break;
            case NW: addInformation( state, myx-1, myy,
(orientation+7)%8 );  // forward-left
            addInformation( state, myx-1, myy+1, orientation );        //
forward
            addInformation( state, myx,   myy+1, (orientation+1)%8 );  //
forward-right
            break;
            }

            if( isBackward && hasReachedFood )
                  return decisionMaker.getHomeGreedyDecision( state );
            else
            {
                  if( GREEDY_EXPLORATION )
                        return decisionMaker.getFoodGreedyDecision( state
);
                  else
                        return decisionMaker.getFoodDecision( state );
            }
      }

      ////////////////////////////////////////////////////////////////////
////////////////
      ////////////////////// ADD PHEROMONES TO THE GRID AT (x,y)
//////////////////////
      ////////////////////////////////////////////////////////////////////
////////////////
      public void addPheromone(DoubleGrid2D grid, int x, int y, double
pheromone)
      {
            double amount = Math.max( grid.field[x][y], pheromone );

            //Depending on the location of ant decide the amount of
pheromone to be left at location x,y
            if( x > 0 && y > 0 )
                  amount = Math.max( Math.max( grid.field[x-1][y-1]-
subtractingRatio, minPheromone ), amount );

            if( x > 0)
                  amount = Math.max( Math.max( grid.field[x-1][y]-
subtractingRatio, minPheromone ), amount );

            if( x > 0 && y < grid.field[x].length-1 )
                  amount = Math.max( Math.max( grid.field[x-1][y+1]-
subtractingRatio, minPheromone ), amount );

            if( y > 0 )
                  amount = Math.max( Math.max( grid.field[x][y-1]-
subtractingRatio, minPheromone ), amount );

            if( y < grid.field.length-1 )
                  amount = Math.max( Math.max( grid.field[x][y+1]-
subtractingRatio, minPheromone ), amount );
```

```
            if( x < grid.field.length-1 && y > 0 )
                  amount = Math.max( Math.max( grid.field[x+1][y-1]-
subtractingRatio, minPheromone ), amount );

            if( x < grid.field.length-1 )
                  amount = Math.max( Math.max( grid.field[x+1][y]-
subtractingRatio, minPheromone ), amount );

            if( x < grid.field.length-1 && y < grid.field[x].length-1 )
                  amount = Math.max( Math.max( grid.field[x+1][y+1]-
subtractingRatio, minPheromone ), amount );

            //Deposit pheromone amount at x,y of the grid
            grid.field[x][y] = amount;
            pheromoneRatio = amount;

      }

      ////////////////////////////////////////////////////////////////
////////////////
      ////////////// DEPENDING ON THE STATUS OF THE ANT, DECIDE NEXT
ACTION /////////////
      ////////////////////////////////////////////////////////////////
////////////////
      public DecisionInfo decideGreedyAction( final SimState state, final
int myx, final int myy, final int orientation )
      {

            final AntsForage af = (AntsForage)state;
            final DecisionMaker decisionMaker = af.decisionMaker;

            decisionMaker.reset();

            //Locate all 8 new neighbours
            addInformation( state, myx,   myy+1, N );  //move North
            addInformation( state, myx+1, myy+1, NE ); //move North-East
            addInformation( state, myx+1, myy,   E );  //move East
            addInformation( state, myx+1, myy-1, SE ); //move South-East
            addInformation( state, myx,   myy-1, S );  //move South
            addInformation( state, myx-1, myy-1, SW ); //move South-East
            addInformation( state, myx-1, myy,   W );  //move West
            addInformation( state, myx-1, myy+1, NW ); //move North-West

            if( isBackward && hasReachedFood )
                  return decisionMaker.getHomeGreedyDecision( state );
            else
                  return decisionMaker.getFoodGreedyDecision( state );

      }

      ////////////////////////////////////////////////
      ////////////// NORMAL MOVE OF THE ANT /////////////
      ////////////////////////////////////////////////
      public void step( final SimState state )
      {
            final AntsForage af = (AntsForage)state;
```

```
            final DecisionMaker decisionMaker = af.decisionMaker;

            Int2D location = af.buggrid.getObjectLocation(this);
            int myx = location.x;
            int myy = location.y;

            int bestx, besty, besto;

            DecisionInfo movingDecision = null;

            //If has found food more than once
            if( isBackward && hasReachedFood )
            {
                  movingDecision = decideGreedyAction( state, myx, myy,
orientation );
                  ++foodTrail;
            }
            else
            {
                  decisionMaker.reset();
                  addInformation( state, myx,   myy+1, 0 );
                  addInformation( state, myx+1, myy+1, 1 );
                  addInformation( state, myx+1, myy,   2 );
                  addInformation( state, myx+1, myy-1, 3 );
                  addInformation( state, myx,   myy-1, 4 );
                  addInformation( state, myx-1, myy-1, 5 );
                  addInformation( state, myx-1, myy,   6 );
                  addInformation( state, myx-1, myy+1, 7 );

                  int max = 0;
                  int howMany = 1;

                  for( int i = 1 ; i < decisionMaker.numInfos ; i++ )
                        if( decisionMaker.info[max].foodPheromoneAmount
==
                              decisionMaker.info[i].foodPheromoneAmount )
                        {
                              howMany++;
                        }
                        else if(
decisionMaker.info[max].foodPheromoneAmount <

      decisionMaker.info[i].foodPheromoneAmount )
                        {
                              max = i;
                              howMany = 1;
                        }

                  if( howMany == 1 )
                        movingDecision = decideGreedyAction( state, myx,
myy, orientation );
                  else
                        movingDecision = decideAction( state, myx, myy,
orientation );

                  if (isBackward)
```

```
                        ++foodTrail;
            }

            if( movingDecision == null )
            {
                  movingDecision = decideGreedyAction( state, myx, myy,
orientation );
                  if( movingDecision == null )
                  {
                        bestx = myx;
                        besty = myy;
                        besto = orientation;
                  }
                  else
                  {
                        bestx = movingDecision.position.x;
                        besty = movingDecision.position.y;
                        besto = movingDecision.orientation;
                  }
                  if (isBackward)
                        ++foodTrail;
            }
            else
            {
                  bestx = movingDecision.position.x;
                  besty = movingDecision.position.y;
                  besto = movingDecision.orientation;

                  if (isBackward)
                        ++foodTrail;
            }

            //Add some pheromones
            if( ( bestx != myx || besty != myy ))
            {
                  //FOOD PHEROMONES
                  if( isBackward && hasReachedFood && numberOfBroadcasts
>= 1 )
                  {

      addPheromone(af.toFoodGrid,myx,myy,pheromoneRatio);
                        ++foodTrail;
                  }

                  //HOME PHEROMONES
                  else if ( !isBackward && ! hasReachedFood &&
numberOfBroadcasts >= 1 )

      addPheromone(af.toHomeGrid,myx,myy,pheromoneRatio);

                  if( bestx != myx && besty != myy )
                        pheromoneRatio -= subtractingRatio*1.4142;
                  else
                        pheromoneRatio -= subtractingRatio;

                  if( pheromoneRatio < 0 )
```

```
                  {
                        die( state );
                        return;
                  }

                  //Place the ant and deposit food and home pheromones
                  af.buggrid.setObjectLocation(this,bestx,besty);
                  orientation = besto;

                  if( ( besty >= AntsForage.HOME_YMIN ) && ( besty <=
AntsForage.HOME_YMAX ) &&
                              ( bestx >= AntsForage.HOME_XMIN ) && (
bestx <= AntsForage.HOME_XMAX ) )
                  {
                        //If the ant has found the food more than once
                        if( isBackward && hasReachedFood )
                        {
                              if ( numberOfBroadcasts < 1 )
                                    af.foodCollected++;

                              isBackward = false;
                              hasReachedFood = false;
                              pheromoneRatio = 1.0 * maxPheromone;

                              if (foodTrail < af.bestFoodTrail)
                                    af.bestFoodTrail = foodTrail;

                              if( GREEDY_REPOSITIONING )
                              {
                                    DecisionInfo temp =
decideGreedyAction( state, myx, myy, orientation );
                                    if( temp != null )
                                          orientation = temp.orientation;
                                    else
                                          orientation =
(orientation+4)%8;
                              }
                              else
                                    orientation = (orientation+4)%8; //
rotate 180
                        }
                  }
                  else if(  ( besty >= AntsForage.FOOD_YMIN ) && ( besty
<= AntsForage.FOOD_YMAX ) &&
                              ( bestx >= AntsForage.FOOD_XMIN ) && (
bestx <= AntsForage.FOOD_XMAX ) )
                  {
                        //If the ant has never found the food
                        if( !isBackward && !hasReachedFood )
                        {
                              //The ant is allowed to do only a certain
amount of broadcasts
                              if ( numberOfBroadcasts > MAX_BROADCASTS )
                              {
                                    numberOfBroadcasts = 0;
                                    die(state);
```

```
                                    return;
                              }

                              isBackward = true;
                              hasReachedFood = true;
                              timesReachedFood++;

                              ++foodTrail;

                              //Count each ant only once
                              if ( timesReachedFood == 1 )
                                    //Increase the number of ants that
found the food
                                    af.numberOfSuccessfulAnts++;

                              pheromoneRatio = 1.0 * maxPheromone;


                              if ( GREEDY_REPOSITIONING )
                              {
                                    DecisionInfo temp =
decideGreedyAction( state, myx, myy, orientation );
                                    if ( temp != null )
                                          orientation = temp.orientation;
                                    else
                                          orientation =
(orientation+4)%8;
                                    if (isBackward)
                                          ++foodTrail;
                              }
                              else
                                    orientation = (orientation+4)%8; //
rotate 180
                        }
                  }

                  timeToLive--;
                  if ( timeToLive <= 0 )
                  {
                        die( state );
                        return;
                  }
            }
      }

      //Ants that haven't ever reached the food and are in the path
      private Color noFoodColor = Color.red;
      //Ants that have reached the food sometime and are in the path
      private Color foodColor = Color.cyan;
      public final void draw(Object object, Graphics2D graphics,
DrawInfo2D info)
      {
            if( isBackward && hasReachedFood )
                  graphics.setColor( foodColor );
            else
                  graphics.setColor( noFoodColor );
```

```
            int x = (int)(info.draw.x - info.draw.width / 2.0);
            int y = (int)(info.draw.y - info.draw.height / 2.0);
            int width = (int)(info.draw.width);
            int height = (int)(info.draw.height);
            graphics.fillOval(x,y,width, height);
      }

      public Stoppable toDiePointer = null;
      public void die( final SimState state )
      {
            AntsForage antsforage = (AntsForage)state;
            antsforage.numberOfAnts--;
            antsforage.buggrid.remove( this );
            if(toDiePointer!=null) toDiePointer.stop();
      }

}
```

Κλάση ReactiveAnt

```
package sim.app.antsforage;

import sim.field.grid.*;
import sim.portrayal.*;
import sim.portrayal.simple.*;
import sim.util.*;
import sim.engine.*;
import java.awt.*;
import java.io.*;
import java.lang.*;
import java.util.Random;

public class ReactiveAnt extends OvalPortrayal2D implements Steppable
{
      public static final boolean GREEDY_REPOSITIONING = true;

      public static final boolean GREEDY_EXPLORATION = true;

      //New ants created after a broadcast
      public static final int NEW_ANTS_PER_BROADCAST = 5;

      //Maximum number of broadcasts allowed for every ant
      public static final int MAX_BROADCASTS = 2;

      //b1 factor for probability calculation
      public static final double b1 = 4;

      //North Direction
      public static final int N = 0;
      //North-East Direction
      public static final int NE = 1;
```

```
	//East Direction
	public static final int E = 2;
	//South-East Direction
	public static final int SE = 3;
	//South Direction
	public static final int S = 4;
	//South-West Direction
	public static final int SW = 5;
	//West Direction
	public static final int W = 6;
	//North-West Direction
	public static final int NW = 7;

	public double pheromoneToLeaveBehind;
	public double minPheromone;
	public double maxPheromone;
	public int timeToLive;

	double subtractingRatio;
	double pheromoneRatio;

	int orientation;

	//GET/SET isBackward
	public boolean getIfIsBackward() { return isBackward; }
	public void setIfIsBackward(boolean val) { isBackward = val; }

	//GET/SET hasReachedFood
	public boolean getHasReachedFood() { return hasReachedFood; }
	public void setHasReachedFood(boolean val) { hasReachedFood = val;
}

	public boolean isBackward;
	public boolean hasReachedFood;
	public int timesReachedFood;
	public int numberOfBroadcasts;
	public int foodTrail;

	////////////////////////////////////////////////////////////////
	////////////////////// R E A C T I V E   A N T /////////////////
	////////////////////////////////////////////////////////////////
	public ReactiveAnt( int orientation,
			double pheromoneToLeaveBehind,
			double minPheromone,
			double maxPheromone,
			int timeToLive )
	{
		//Initialise the parameters based on the values sent from the
nest
		this.orientation = orientation;
		this.pheromoneToLeaveBehind = pheromoneToLeaveBehind;
		this.minPheromone = minPheromone;
		this.maxPheromone = maxPheromone;
		this.timeToLive = timeToLive;

		subtractingRatio = ( 1.0 / timeToLive ) * maxPheromone;
```

```
            pheromoneRatio = 1.0 * maxPheromone;

            //At start, the ant has never reached food, so it's still a
forward ant
            isBackward = false;
            hasReachedFood = false;
            timesReachedFood = 0;

            //At start, the ant has never broadcasted
            numberOfBroadcasts = 0;
      }

      ////////////////////////////////////////////////////////////////
////////////////
      ////////////////// ADD INFORMATION FOR EVERY PLACE OF THE ANT (x,y)
////////////////
      ////////////////////////////////////////////////////////////////
////////////////
      protected void addInformation( final SimState state, int x, int y,
final int orientation )
      {
            final AntsForage af = (AntsForage)state;
            final DecisionInfo di = af.decisionInfo;
            final DecisionMaker decisionMaker = af.decisionMaker;

            if( x < 0 || x >= AntsForage.GRID_WIDTH || y < 0 || y >=
AntsForage.GRID_HEIGHT )
                  return;

            if( ( af.buggrid.getObjectsAtLocation(x,y) == null ||
                        af.buggrid.getObjectsAtLocation(x,y).numObjs <
AntsForage.MAX_ANTS_PER_LOCATION ) &&
                        af.obstacles.field[x][y] <= 0.5 )
            {
                  //set coordinates
                  di.position.x = x;
                  di.position.y = y;
                  di.orientation = orientation;

                  //Add pheromones to current location of ant
                  di.homePheromoneAmount = 0.001 +
af.toHomeGrid.field[di.position.x][di.position.y];
                  di.foodPheromoneAmount = 0.001 +
af.toFoodGrid.field[di.position.x][di.position.y];
                  decisionMaker.addInfo( di );
            }
      }

      ////////////////////////////////////////////////////////////////
////////////////
      ////////////// DEPENDING ON THE PLACE OF THE ANT (x,y), DECIDE NEXT
ACTION ////////
      ////////////////////////////////////////////////////////////////
////////////////
      public DecisionInfo decideAction( final SimState state, final int
myx, final int myy, final int orientation )
```

```
      {

            final AntsForage af = (AntsForage)state;
            final DecisionMaker decisionMaker = af.decisionMaker;
            int newOrientation = orientation;
            double Pnd = 0;

            decisionMaker.reset();

            Random random = new Random();

            //No pheromone information available
            if ( af.toFoodGrid.field[myx][myy] == 0.0 ||
af.toHomeGrid.field[myx][myy] == 0.0 )
            {
                  //Depending on the number of broadcasts already
executed by the ant,
                  //we decide if the ant is going to execute a new
broadcast
                  if ( numberOfBroadcasts < MAX_BROADCASTS )
                  {
                        ++numberOfBroadcasts;
                        for(int x=0 ; x<NEW_ANTS_PER_BROADCAST &&
af.numberOfAnts < af.MAX_ANTS; x++)
                        {
                              ReactiveAnt bug = new ReactiveAnt(
random.nextInt(8),
                                          pheromoneToLeaveBehind,
                                          minPheromone,
                                          maxPheromone,
                                          timeToLive );


      af.buggrid.setObjectLocation(bug,(af.HOME_XMAX+af.HOME_XMIN)/2,(af.
HOME_YMAX+af.HOME_YMIN)/2);
                              bug.toDiePointer =
state.schedule.scheduleRepeating(bug);
                              af.numberOfAnts++;
                        }
                        newOrientation = orientation;
                  }
            }
            //Pheromone information available
            else
            {
                  double sum = 0;

                  //Calculate the sum of pheromones for every neighbour
                  if ( myx < af.GRID_HEIGHT-1 && myx > 0 && myx <
af.GRID_WIDTH-1 && myx > 0
                              && myy < af.GRID_HEIGHT-1 && myy > 0 && myy
< af.GRID_WIDTH-1 && myy > 0 )
                  {
                        //Neighbour 1 - N
                        sum += Math.pow(af.toFoodGrid.field[myx][myy+1],
b1);
```

```
                        //Neighbour 2 - NE
                        sum +=
Math.pow(af.toFoodGrid.field[myx+1][myy+1], b1);
                        //Neighbour 3 - E
                        sum += Math.pow(af.toFoodGrid.field[myx+1][myy],
b1);
                        //Neighbour 4 - SE
                        sum += Math.pow(af.toFoodGrid.field[myx+1][myy-
1], b1);
                        //Neighbour 5 - S
                        sum += Math.pow(af.toFoodGrid.field[myx][myy-1],
b1);
                        //Neighbour 6 - SW
                        sum += Math.pow(af.toFoodGrid.field[myx-1][myy-
1], b1);
                        //Neighbour 7 - W
                        sum += Math.pow(af.toFoodGrid.field[myx-1][myy],
b1);
                        //Neighbour 8 - NW
                        sum += Math.pow(af.toFoodGrid.field[myx-
1][myy+1], b1);

                        //Choose the next position with the probability
Pnd
                        Pnd = Math.pow(af.toFoodGrid.field[myx][myy], b1)
/ sum;

                        if ( Pnd <= 0.125 )
                              newOrientation = N;
                        else if ( Pnd > 0.125 && Pnd <= 0.25 )
                              newOrientation = NE;
                        else if ( Pnd > 0.25 && Pnd <= 0.375 )
                              newOrientation = E;
                        else if ( Pnd > 0.375 && Pnd <= 0.5 )
                              newOrientation = SE;
                        else if ( Pnd > 0.5 && Pnd <= 0.625 )
                              newOrientation = S;
                        else if ( Pnd > 0.625 && Pnd <= 0.75 )
                              newOrientation = SW;
                        else if ( Pnd > 0.75 && Pnd <= 0.875 )
                              newOrientation = W;
                        else if ( Pnd > 0.875 )
                              newOrientation = NW;

                        newOrientation = orientation;
                  }
            }

            switch( newOrientation )
            {
            case N: addInformation( state, myx-1, myy+1,
(orientation+7)%8 );  // forward-left
            addInformation( state, myx,   myy+1, orientation );        //
forward
            addInformation( state, myx+1, myy+1, (orientation+1)%8 );  //
forward-right
```

```
            break;
            case NE: addInformation( state, myx,   myy+1,
(orientation+7)%8 );  // forward-left
            addInformation( state, myx+1, myy+1, orientation );        //
forward
            addInformation( state, myx+1, myy,   (orientation+1)%8 );  //
forward-right
            break;
            case E: addInformation( state, myx+1, myy+1,
(orientation+7)%8 );  // forward-left
            addInformation( state, myx+1, myy,   orientation );        //
forward
            addInformation( state, myx+1, myy-1, (orientation+1)%8 );  //
forward-right
            break;
            case SE: addInformation( state, myx+1, myy,
(orientation+7)%8 );  // forward-left
            addInformation( state, myx+1, myy-1, orientation );        //
forward
            addInformation( state, myx,   myy-1, (orientation+1)%8 );  //
forward-right
            break;
            case S: addInformation( state, myx+1, myy-1,
(orientation+7)%8 );  // forward-left
            addInformation( state, myx,   myy-1, orientation );        //
forward
            addInformation( state, myx-1, myy-1, (orientation+1)%8 );  //
forward-right
            break;
            case SW: addInformation( state, myx,   myy-1,
(orientation+7)%8 );  // forward-left
            addInformation( state, myx-1, myy-1, orientation );        //
forward
            addInformation( state, myx-1, myy,   (orientation+1)%8 );  //
forward-right
            break;
            case W: addInformation( state, myx-1, myy-1,
(orientation+7)%8 );  // forward-left
            addInformation( state, myx-1, myy,   orientation );        //
forward
            addInformation( state, myx-1, myy+1, (orientation+1)%8 );  //
forward-right
            break;
            case NW: addInformation( state, myx-1, myy,
(orientation+7)%8 );  // forward-left
            addInformation( state, myx-1, myy+1, orientation );        //
forward
            addInformation( state, myx,   myy+1, (orientation+1)%8 );  //
forward-right
            break;
            }

            if( isBackward && hasReachedFood )
                  return decisionMaker.getHomeGreedyDecision( state );
            else
            {
```

```
                  if( GREEDY_EXPLORATION )
                        return decisionMaker.getFoodGreedyDecision( state
);
                  else
                        return decisionMaker.getFoodDecision( state );
            }
      }

      ////////////////////////////////////////////////////////////////////
////////////////
      ////////////////////// ADD PHEROMONES TO THE GRID AT (x,y)
//////////////////////
      ////////////////////////////////////////////////////////////////////
////////////////
      public void addPheromone(DoubleGrid2D grid, int x, int y, double
pheromone)
      {
            double amount = Math.max( grid.field[x][y], pheromone );

            //Depending on the location of ant decide the amount of
pheromone to be left at location x,y
            if( x > 0 && y > 0 )
                  amount = Math.max( Math.max( grid.field[x-1][y-1]-
subtractingRatio, minPheromone ), amount );

            if( x > 0)
                  amount = Math.max( Math.max( grid.field[x-1][y]-
subtractingRatio, minPheromone ), amount );

            if( x > 0 && y < grid.field[x].length-1 )
                  amount = Math.max( Math.max( grid.field[x-1][y+1]-
subtractingRatio, minPheromone ), amount );

            if( y > 0 )
                  amount = Math.max( Math.max( grid.field[x][y-1]-
subtractingRatio, minPheromone ), amount );

            if( y < grid.field.length-1 )
                  amount = Math.max( Math.max( grid.field[x][y+1]-
subtractingRatio, minPheromone ), amount );

            if( x < grid.field.length-1 && y > 0 )
                  amount = Math.max( Math.max( grid.field[x+1][y-1]-
subtractingRatio, minPheromone ), amount );

            if( x < grid.field.length-1 )
                  amount = Math.max( Math.max( grid.field[x+1][y]-
subtractingRatio, minPheromone ), amount );

            if( x < grid.field.length-1 && y < grid.field[x].length-1 )
                  amount = Math.max( Math.max( grid.field[x+1][y+1]-
subtractingRatio, minPheromone ), amount );

            //Deposit pheromone amount at x,y of the grid
            grid.field[x][y] = amount;
            pheromoneRatio = amount;
```

```
      }

      ////////////////////////////////////////////////////////////////////
////////////////
      ////////////// DEPENDING ON THE STATUS OF THE ANT, DECIDE NEXT
ACTION /////////////
      ////////////////////////////////////////////////////////////////////
////////////////
      public DecisionInfo decideGreedyAction( final SimState state, final
int myx, final int myy, final int orientation )
      {

            final AntsForage af = (AntsForage)state;
            final DecisionMaker decisionMaker = af.decisionMaker;

            decisionMaker.reset();

            //Locate all 8 new neighbours
            addInformation( state, myx,   myy+1, N );  //move North
            addInformation( state, myx+1, myy+1, NE ); //move North-East
            addInformation( state, myx+1, myy,   E );  //move East
            addInformation( state, myx+1, myy-1, SE ); //move South-East
            addInformation( state, myx,   myy-1, S );  //move South
            addInformation( state, myx-1, myy-1, SW ); //move South-East
            addInformation( state, myx-1, myy,   W );  //move West
            addInformation( state, myx-1, myy+1, NW ); //move North-West

            if( isBackward && hasReachedFood )
                  return decisionMaker.getHomeGreedyDecision( state );
            else
                  return decisionMaker.getFoodGreedyDecision( state );

      }

      ////////////////////////////////////////////////
      ////////////// NORMAL MOVE OF THE ANT /////////////
      ////////////////////////////////////////////////
      public void step( final SimState state )
      {
            final AntsForage af = (AntsForage)state;
            final DecisionMaker decisionMaker = af.decisionMaker;

            Int2D location = af.buggrid.getObjectLocation(this);
            int myx = location.x;
            int myy = location.y;

            int bestx, besty, besto;

            DecisionInfo movingDecision = null;

            //If has found food more than once
            if( isBackward && hasReachedFood )
            {
                  movingDecision = decideGreedyAction( state, myx, myy,
orientation );
                  ++foodTrail;
```

```
            }
            else
            {
                  decisionMaker.reset();
                  addInformation( state, myx,   myy+1, 0 );
                  addInformation( state, myx+1, myy+1, 1 );
                  addInformation( state, myx+1, myy,   2 );
                  addInformation( state, myx+1, myy-1, 3 );
                  addInformation( state, myx,   myy-1, 4 );
                  addInformation( state, myx-1, myy-1, 5 );
                  addInformation( state, myx-1, myy,   6 );
                  addInformation( state, myx-1, myy+1, 7 );

                  int max = 0;
                  int howMany = 1;

                  for( int i = 1 ; i < decisionMaker.numInfos ; i++ )
                        if( decisionMaker.info[max].foodPheromoneAmount
==
                              decisionMaker.info[i].foodPheromoneAmount )
                        {
                              howMany++;
                        }
                        else if(
decisionMaker.info[max].foodPheromoneAmount <

      decisionMaker.info[i].foodPheromoneAmount )
                        {
                              max = i;
                              howMany = 1;
                        }

                  if( howMany == 1 )
                        movingDecision = decideGreedyAction( state, myx,
myy, orientation );
                  else
                        movingDecision = decideAction( state, myx, myy,
orientation );

                  if ( isBackward )
                        ++foodTrail;
            }

            if( movingDecision == null )
            {
                  movingDecision = decideGreedyAction( state, myx, myy,
orientation );
                  if( movingDecision == null )
                  {
                        bestx = myx;
                        besty = myy;
                        besto = orientation;
                  }
                  else
                  {
                        bestx = movingDecision.position.x;
```

```
                        besty = movingDecision.position.y;
                        besto = movingDecision.orientation;
                  }
                  if ( isBackward )
                        ++foodTrail;
            }
            else
            {
                  bestx = movingDecision.position.x;
                  besty = movingDecision.position.y;
                  besto = movingDecision.orientation;

                  if ( isBackward )
                        ++foodTrail;
            }

            //Add some pheromones
            if( ( bestx != myx || besty != myy ))
            {
                  //FOOD PHEROMONES
                  if( isBackward && hasReachedFood )
                  {

      addPheromone(af.toFoodGrid,myx,myy,pheromoneRatio);
                        ++foodTrail;
                  }

                  //HOME PHEROMONES
                  else

      addPheromone(af.toHomeGrid,myx,myy,pheromoneRatio);

                  if( bestx != myx && besty != myy )
                        pheromoneRatio -= subtractingRatio*1.4142;
                  else
                        pheromoneRatio -= subtractingRatio;
                  if( pheromoneRatio < 0 )
                  {
                        die( state );
                        return;
                  }

                  //Place the ant and deposit food and home pheromones
                  af.buggrid.setObjectLocation(this,bestx,besty);
                  orientation = besto;

                  if( ( besty >= AntsForage.HOME_YMIN ) && ( besty <=
AntsForage.HOME_YMAX ) &&
                              ( bestx >= AntsForage.HOME_XMIN ) && (
bestx <= AntsForage.HOME_XMAX ) )
                  {
                        //If the ant has found the food more than once
                        if( isBackward && hasReachedFood )
                        {
                              af.foodCollected++;
                              isBackward = false;
```

```
                              hasReachedFood = false;
                              pheromoneRatio = 1.0 * maxPheromone;

                              if ( foodTrail < af.bestFoodTrail )
                                    af.bestFoodTrail = foodTrail;

                              if( GREEDY_REPOSITIONING )
                              {
                                    DecisionInfo temp =
decideGreedyAction( state, myx, myy, orientation );
                                    if( temp != null )
                                          orientation = temp.orientation;
                                    else
                                          orientation =
(orientation+4)%8;
                              }
                              else
                                    orientation = (orientation+4)%8; //
rotate 180
                        }
                  }
                  else if(  ( besty >= AntsForage.FOOD_YMIN ) && ( besty
<= AntsForage.FOOD_YMAX ) &&
                              ( bestx >= AntsForage.FOOD_XMIN ) && (
bestx <= AntsForage.FOOD_XMAX ) )
                  {
                        //If the ant has never found the food
                        if ( !isBackward && !hasReachedFood )
                        {
                              //The ant is allowed to do only a certain
amount of broadcasts
                              if ( numberOfBroadcasts > MAX_BROADCASTS )
                              {
                                    numberOfBroadcasts = 0;
                                    die(state);
                                    return;
                              }

                              isBackward = true;
                              hasReachedFood = true;
                              timesReachedFood++;

                              ++foodTrail;

                              //Count each ant only once
                              if ( timesReachedFood == 1 )
                                    //Increase the number of ants that
found the food
                                    af.numberOfSuccessfulAnts++;

                              pheromoneRatio = 1.0 * maxPheromone;

                              if( GREEDY_REPOSITIONING )
                              {
                                    DecisionInfo temp =
decideGreedyAction( state, myx, myy, orientation );
```

```
                                    if( temp != null )
                                          orientation = temp.orientation;
                                    else
                                          orientation =
(orientation+4)%8;

                                    if ( isBackward )
                                          ++foodTrail;
                              }
                              else
                                    orientation = (orientation+4)%8; //
rotate 180
                        }
                  }

                  timeToLive--;
                  if( timeToLive <= 0 )
                  {
                        die( state );
                        return;
                  }
            }
      }

      //Ants that haven't ever reached the food and are in the path
      private Color noFoodColor = Color.black;
      //Ants that have reached the food sometime and are in the path
      private Color foodColor = Color.white;
      public final void draw(Object object, Graphics2D graphics,
DrawInfo2D info)
      {
            if( isBackward && hasReachedFood )
                  graphics.setColor( foodColor );
            else
                  graphics.setColor( noFoodColor );

            int x = (int)(info.draw.x - info.draw.width / 2.0);
            int y = (int)(info.draw.y - info.draw.height / 2.0);
            int width = (int)(info.draw.width);
            int height = (int)(info.draw.height);
            graphics.fillOval(x,y,width, height);
      }

      public Stoppable toDiePointer = null;
      public void die( final SimState state )
      {
            AntsForage antsforage = (AntsForage)state;
            antsforage.numberOfAnts--;
            antsforage.buggrid.remove( this );
            if(toDiePointer!=null) toDiePointer.stop();
      }

}
```