ΠΑΝΕΠΙΣΤΗΜΙΟ ΣΤΕΡΕΑΣ ΕΛΛΑΔΑΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗ ΒΙΟΙΑΤΡΙΚΗ

Κατανεμημένοι αλγόριθμοι ανακάλυψης σφαλμάτων σε συγχρονισμένους δακτύλιους

Χρήστος Παπαγεωργάκης

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ
Υπεύθυνος
Ευριπίδης Μάρκου
Λέκτορας

Λαμία, 2012

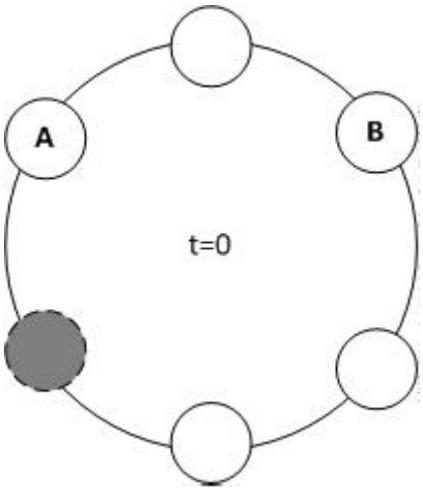
ΠΑΝΕΠΙΣΤΗΜΙΟ ΣΤΕΡΕΑΣ ΕΛΛΑΔΑΣ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗ ΒΙΟΪΑΤΡΙΚΗ

Κατανεμημένοι αλγόριθμοι ανακάλυψης σφαλμάτων σε συγχρονισμένους δακτυλίους

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ



ΧΡΗΣΤΟΣ ΠΑΠΑΓΕΩΡΓΑΚΗΣ

Επιβλέπων : ΕΥΡΙΠΙΔΗΣ ΜΑΡΚΟΥ, Λέκτορας

Λαμία, 2012

Πίνακας περιεχομένων

Πρόλογος

Μετά από τρία χρόνια φοίτησης στο Πανεπιστήμιο Στερεάς Ελλάδος, στο τμήμα Πληροφορικής με Εφαρμογές στη Βιοϊατρική και με αμείωτο το ενδιαφέρον για τον προγραμματισμό και την επίλυση μέσω αυτού, θεμάτων που απασχολούν τους γύρω μου αλλά και την επιστημονική κοινότητα, διαπίστωσα πως το θέμα της παρούσας πτυχιακής αποτελούσε για μένα μια μεγάλη πρόκληση.

Οι κατανεμημένοι αλγόριθμοι ανακάλυψης σφαλμάτων σε συγχρονισμένους δακτυλίους αποτελεί ένα θέμα που απασχολεί τους επιστήμονες – ερευνητές την τελευταία δεκαετία και με την ευκαιρία αυτή, λόγω της παρούσας πτυχιακής, μου δίνεται η δυνατότητα να συμβάλω με τον τρόπο μου, στην επίλυσή του.

Καθώς το θέμα ήταν ¨φλέγον¨, το ενδιαφέρον και η ενασχόλησή μου με αυτό ήταν αμείωτη και γεμάτη στόχους, που έπρεπε να κατακτηθούν. Αποτέλεσμα όλων των παραπάνω είναι η παρουσίαση νέων αλγορίθμων για την επίλυση ορισμένων πτυχών του γενικότερου προβλήματος.

Ευχαριστίες

Πρωτίστως, θα ήθελα να ευχαριστήσω θερμά τον λέκτορα κ. Ευριπίδη Μάρκου, για την εμπιστοσύνη που έδειξε στο πρόσωπό μου αναθέτοντάς μου την παρούσα πτυχιακή εργασία, την καθοδήγηση και τις γνώσεις που μου μεταλαμπάδευσε.

Δευτερευόντως, θα ήθελα να ευχαριστήσω τα μέλη της επιτροπής επίκουρο καθηγητή κ. Βασίλη Πλαγιανάκο και αναπληρωτή καθηγητή κ. Μιχάλη Βασιλακόπουλο.

Επίσης θα ήθελα να ευχαριστήσω το φίλο και συμφοιτητή Μωυσή Συμεωνίδη για τη συνεχή στήριξη, τις εύστοχες υποδείξεις, παρατηρήσεις και ιδέες.

Τέλος όλους τους συμφοιτητές και φίλους για τις εποικοδομητικές συζητήσεις και την ανιδιοτελή υποστήριξη.

Λαμία 2012 Χρήστος Παπαγεωργάκης

Περίληψη

Η ανακάλυψη σφαλμάτων σε δίκτυα δεδομένων είναι ένα σημαντικό πρόβλημα του οποίου η λύση μερικές φορές (όπως για παράδειγμα όταν στο δίκτυο εκτελούνται διεργασίες μη ανεκτικές σε σφάλματα) μπορεί να είναι άμεσα συνδεδεμένη με την προστασία αλλά και την ασφάλεια των δεδομένων που διακινούνται. Έτσι λοιπόν, ένας από τους στόχους της συντήρησης ενός δικτύου είναι η ανακάλυψη σφαλμάτων και η αναφορά των κόμβων στους οποίους συμβαίνουν. Ένα μεγάλο τμήμα της έρευνας στους κατανεμημένους υπολογισμούς, που ασχολείται με την εξερεύνηση δικτύων χρησιμοποιώντας ρομπότ (ή αλλιώς πράκτορες λογισμικού), αναφέρεται σε προβλήματα ανακάλυψης σφαλμάτων. Σε αυτή την εργασία χρησιμοποιείται η μοντελοποίηση των σφαλμάτων με μαύρες τρύπες. Μαύρη τρύπα ονομάζεται μια επιβλαβής διαδικασία (π.χ. ιός) που βρίσκεται σε έναν κόμβο ενός δικτύου υπολογιστών και καταστρέφει οποιονδήποτε πράκτορα λογισμικού επισκεφτεί αυτόν τον κόμβο χωρίς να αφήνει κάποιο ίχνος. Εξετάζουμε το ακόλουθο πρόβλημα: Πράκτορες λογισμικού αρχικά τοποθετημένοι σε διαφορετικούς, ασφαλείς κόμβους ενός συγχρονισμένου δικτύου τοπολογίας δακτυλίου, αναζητούν μαύρες τρύπες. Στόχος των πρακτόρων είναι να ανακαλύψουν τις θέσεις από όλες τις μαύρες τρύπες και να τις αναφέρουν σημειώνοντας τις συνδέσεις (links) που οδηγούν σε αυτές. Οι πράκτορες δεν μπορούν να επικοινωνήσουν αν βρίσκονται σε διαφορετικούς κόμβους, ενώ όλοι χρησιμοποιούν τον ίδιο ντετερμινιστικό αλγόριθμο. Στην παρούσα εργασία παρουσιάζονται νέοι αλγόριθμοι που λύνουν το πρόβλημα της ανακάλυψης τέτοιων σφαλμάτων σε ένα δίκτυο-δακτύλιο, χρησιμοποιώντας τον ελάχιστο αριθμό πρακτόρων. Οι αλγόριθμοι, που αναπτύσσονται λύνουν βέλτιστα το πρόβλημα της αναζήτησης, όταν υπάρχουν: μία ή δύο μαύρες τρύπες στο δακτύλιο. Αποδεικνύουμε την ορθότητα των αλγορίθμων μας, αναλύουμε την πολυπλοκότητά τους και δίνουμε σφιχτά κάτω όρια για τον ελάχιστο αριθμό πρακτόρων τα οποία καθιστούν τους αλγόριθμους μας βέλτιστους. Επίσης δίνουμε αλγορίθμους για μερικές παραλλαγές του βασικού μοντέλου που χρησιμοποιούμε καθώς και αλγορίθμους που κάτω από κάποιες επιπλέον προϋποθέσεις λειτουργούν για οποιαδήποτε γνωστό αριθμό μαύρων τρυπών. Παρ' όλο που το πρόβλημα της ανίχνευσης μαύρης τρύπας έχει μελετηθεί αρκετά στη διεθνή βιβλιογραφία σε συγχρονισμένα δίκτυα, από όσο γνωρίζουμε, αυτά είναι τα πρώτα αποτελέσματα για πράκτορες οι οποίοι είναι αρχικά τοποθετημένοι σε διαφορετικούς κόμβους και μπορούν να επικοινωνήσουν μόνο όταν συναντηθούν. Τέλος η εργασία, συμπεριλαμβάνει την υλοποίηση ενός από τους νέους αλγορίθμους που προτείνονται καθώς και ένα βασικό υπολογιστικό περιβάλλον το οποίο δέχεται παραμέτρους από το χρήστη και εκτελεί τον αλγόριθμο σε τοπολογίες δακτυλίου.

Λέξεις-κλειδιά: θεωρητική πληροφορική, κατανεμημένοι αλγόριθμοι, μαύρες τρύπες, κινητοί πράκτορες, εξερεύνηση δακτυλίου.

Abstract

Finding faults in data networks is an important problem whose solution (apart from necessary for the network operation) is often (e.g., when **non** fault-tolerant procedures are executed on the network) necessary for the data protection and is closely related to the network security. Therefore a usual task in network maintenances to report faulty nodes. Many researchers in Distributed Computing (as well as in other domains) study fault searching problems and especially using agents. In this work we model faults with black-holes. A black hole is a highly damaging procedure (e.g., a virus) which lies at a node n of the network and destroys any agent upon arrival at n without leaving any trace. More specifically we study the following problem. Software agents initially placed at different safe nodes of a synchronous ring network are searching for black hole in the network. Their task is to locate all black-holes and mark all incoming links. The agents execute the same deterministic algorithm and they can communicate only when they meet at nodes. We present new algorithms which solve this problem using a minimum number of agents. In particular, we optimally solve the problem when there are one or two black holes in the network. We prove correctness of our algorithms, we analyze their time complexity and we show tight lower bounds for the necessary number of agents (w.r.t. feasibility), thus proving optimality of our algorithms. We also give algorithms for variation of the basic studied model and algorithms which (under an additional condition) solve the problem for any given number of black holes. Although the problem has been previously studied for the case of synchronous networks, to the best of our knowledge, these are the first results for the case initially **scattered** agents capable of communicating each-other **only when they meet at a node**. We also include the implementation of one of the proposed new algorithms and an interface in which a user can define a few parameters and execute an algorithm in ring networks.

Keywords: Theoretical Computes Science, Distributed Algorithms, Black Holes, mobile agent, ring exploration.

Δομή εργασίας

Η παρούσα πτυχιακή εργασία αποτελείται από τα εξής κεφάλαια:

- Στο κεφάλαιο **1** παρουσιάζονται εισαγωγικά στοιχεία που αφορούν στις συνιστώσες του προβλήματος αναζήτησης μαύρων τρυπών, ειδικότερα στους κατανεμημένους αλγόριθμους για δίκτυα, στα βασικά χαρακτηριστικά των κινητών πρακτόρων, και αναφέρεται η προηγούμενη σχετική δουλειά.

- Στο κεφάλαιο **2** παρουσιάζονται τα βασικά χαρακτηριστικά, που διακρίνουν το μοντέλο που θα χρησιμοποιηθεί και στη συνέχεια δίνεται ένας τυπικός ορισμός αυτού.

- Στο κεφάλαιο **3** εξετάζεται το πρόβλημα της αναζήτησης όταν είναι γνωστό ότι το δίκτυο έχει ακριβώς μία ή το πολύ μία μαύρη τρύπα, παρουσιάζονται νέοι αλγόριθμοι επίλυσης του αντίστοιχου προβλήματος, επιπλέον υπολογίζονται οι πολυπλοκότητές τους και αναπτύσσονται οι αποδείξεις ορθότητάς τους.

- Το κεφάλαιο **4** ασχολείται με το πρόβλημα της αναζήτησης όταν είναι γνωστό ότι το δίκτυο έχει ακριβώς δύο μαύρες τρύπες, παρουσιάζονται νέοι αλγόριθμοι επίλυσης του αντίστοιχου προβλήματος, επιπλέον υπολογίζονται οι πολυπλοκότητές τους και αναπτύσσονται οι αποδείξεις ορθότητάς τους.

- Στο κεφάλαιο **5** ερευνάται το πρόβλημα της αναζήτησης όταν είναι γνωστό ότι το δίκτυο έχει ακριβώς Μ μαύρες τρύπες (όπου Μ είναι τμήμα της εισόδου μεγαλύτερο του δυο) και παρουσιάζονται τα βασικά χαρακτηριστικά και οι ιδιότητες μιας τέτοιας αναζήτησης.

- Στο παράρτημα, παραθέτουμε την υλοποίηση ενός από τους νέους αλγορίθμους που προτείνονται καθώς και ένα βασικό περιβάλλον το οποίο δέχεται παραμέτρους από το χρήστη και εκτελεί τον αλγόριθμο σε τοπολογίες δακτυλίου.

Κεφάλαιο 1 – Εισαγωγή

1.1 Εισαγωγή

Σε ένα κατανεμημένο σύστημα, διάφοροι υπολογιστές ή επεξεργαστές επικοινωνούν μεταξύ τους, με σκοπό να ανταλλάσουν πληροφορίες και να επιλύσουν κάποιο πρόβλημα. Η ανάλυση και η υλοποίηση τέτοιων συστημάτων, τόσο σε θεωρητικό όσο και σε πρακτικό επίπεδο, παρουσιάζει αρκετές δυσκολίες και προκλήσεις. Πολλές από τις εφαρμογές που υπάρχουν σήμερα χρησιμοποιούν κάποιο είδος κατανεμημένων υπολογισμών.

Οι κατανεμημένοι αλγόριθμοι χρησιμοποιούνται ευρέως για την επίλυση προβλημάτων, σε εφαρμογές όπως για παράδειγμα σε δίκτυα peer-to-peer, συστήματα τηλεφωνίας, αεροπορικές κρατήσεις, μετεωρολογικές προβλέψεις, συστήματα τραπεζών κ.λ.π.

Ένας τρόπος ανταλλαγής πληροφοριών μεταξύ των υπολογιστών ενός κατανεμημένου συστήματος είναι το σύστημα ανταλλαγής μηνυμάτων (Message Passing System). Ένα από τα μεγαλύτερα μειονεκτήματα αυτής της τεχνικής είναι ο μεγάλος όγκος δεδομένων που στέλνονται μέσω του δικτύου.

Αντίθετα ένα σύστημα κινητών πρακτόρων (mobile agent system) λειτουργεί διαφορετικά. Αντί να μετακινούνται πακέτα δεδομένων μεταξύ των υπολογιστών που βρίσκονται στους κόμβους του δικτύου, κινούνται πράκτορες, δηλαδή λογισμικό από κόμβο σε κόμβο.

Οι πράκτορες ουσιαστικά είναι προγράμματα τα οποία έχουν τη δυνατότητα να μετακινούνται στο δίκτυο και να εκτελούνται στους κόμβους. Κάθε κόμβος παρέχει στους πράκτορες την απαραίτητη υπολογιστική ισχύ, χώρο αποθήκευσης καθώς επίσης και άλλες υπηρεσίες, όπως για παράδειγμα την πληροφορία αν υπάρχει κάποιος άλλος πράκτορας στον ίδιο κόμβο.

1.2 Κινητοί πράκτορες

Οι κινητοί πράκτορες είναι λογισμικά που έχουν τη δυνατότητα κατά τη διάρκεια της εκτέλεσής τους να αντιγράφουν τον εαυτό τους και να συνεχίζουν την εκτέλεσή τους σε διαφορετικά σημεία του δικτύου.

Ένα από τα βασικότερα χαρακτηριστικά των πρακτόρων είναι ο μηχανισμός με τον οποίο επικοινωνούν. Όπως θα δούμε στη συνέχεια, σε κάποια μοντέλα οι

πράκτορες έχουν την ικανότητα να διαβάζουν την κατάσταση των άλλων πρακτόρων όταν βρίσκονται στον ίδιο κόμβο, ενώ σε άλλα μπορούν να επικοινωνούν μέσω κοινής (shared) μνήμης ή μέσω μηνυμάτων.

Οι πράκτορες μπορούν να είναι ή να μην είναι διακεκριμένοι (distinct). Αν έχουν διαφορετικές ταυτότητες μπορούν να τρέξουν διαφορετικό τμήμα κάποιου συγκεκριμένου αλγορίθμου. Στην περίπτωση όπου έχουν όλοι τις ίδιες ταυτότητες ονομάζονται ανώνυμοι (anonymous agents).

Τέλος ένα άλλο χαρακτηριστικό είναι η γνώση που έχουν οι πράκτορες για το περιβάλλον. Τέτοιες πληροφορίες είναι: η τοπολογία, το μέγεθος (π.χ. ο αριθμός των κόμβων) του δικτύου, ο αριθμός και οι ταυτότητες των πρακτόρων που βρίσκονται στο δίκτυο καθώς και οι θέσεις τους σε αυτό. Οι πληροφορίες αυτές, όταν είναι διαθέσιμες (μέρος της εισόδου του αλγορίθμου), μπορούν να αλλάξουν σημαντικά τον τρόπο επίλυσης και την αποτελεσματικότητα των πρακτόρων στα διάφορα προβλήματα.

1.2.1 Ιδιότητες πρακτόρων

Αυτονομία: Σε αντίθεση με τα συμβατικά μοντέλα, όπου υπάρχει μία γενική αρχή που κατευθύνει και δίνει οδηγίες στις οντότητες (π.χ. υπολογιστές ή πράκτορες), στους κατανεμημένους υπολογισμούς ο κάθε πράκτορας εκτελεί έναν αλγόριθμο και συνήθως δεν αναφέρει κάτι, ούτε παίρνει οδηγίες από μια γενική αρχή.

Κινητικότητα: οι κινητοί πράκτορες έχουν τη δυνατότητα να μετακινούνται από κόμβο σε κόμβο στο δίκτυο, να εκτελούν υπολογισμούς και πολλές φορές να έχουν πρόσβαση σε δεδομένα και υπηρεσίες.

Κάποιες άλλες ιδιότητες των πρακτόρων είναι:

Ο τρόπος με τον οποίο ο πράκτορας αντιδρά με άλλους πράκτορες ή με το περιβάλλον (για παράδειγμα να κάνει ερωτήσεις στους κόμβους και να αναφέρει τα ευρήματα του). Σε πολλές εφαρμογές χρησιμοποιούνται περισσότεροι από έναν πράκτορες και συνήθως χρησιμοποιούν κάποιον μηχανισμό επικοινωνίας. Υπάρχουν επίσης περιπτώσεις που οι πράκτορες έχουν ανταγωνιστική συμπεριφορά.

Η ικανότητα του πράκτορα να προσαρμόζεται σε νέες καταστάσεις δηλαδή να μαθαίνει από προηγούμενες εμπειρίες και να ερμηνεύει σωστά τις προθέσεις είτε του χρήστη που τον δημιούργησε είτε των πρακτόρων που συναντά.

1.2.2 Τα πλεονεκτήματα των κινητών πρακτόρων

Αποδοτικότητα: Η χρήση κινητών πρακτόρων μπορεί να μειώσει την κίνηση στο δίκτυο (latency) και το χρόνο επίλυσης ενός προβλήματος.

Ανεκτικότητα λαθών: Αν κάποιος κόμβος αποσυνδεθεί από το δίκτυο, ο πράκτορας μπορεί να κινηθεί σε άλλο κόμβο και να συνεχίζει την εκτέλεση του.

Εύκολη συντήρηση: Γενικά είναι εύκολο να προστεθούν νέες ιδιότητες στους πράκτορες χωρίς να χρειαστεί συντήρηση όλο το δίκτυο.

Ευκολία στη χρήση: σε πολλές περιπτώσεις είναι φυσιολογική για το χρήστη, αλλά και για τον προγραμματιστή, η χρήση πρακτόρων, καθώς μπορεί να βελτιώσει την εμπειρία που έχει ο χρήστης για την εφαρμογή και ίσως να κάνει το σχεδιασμό και την υλοποίηση του συστήματος, πιο εύκολα.

1.2.3 Ασφάλεια πρακτόρων

Η δυνατότητα των πρακτόρων να κινούνται δημιούργησε σημαντικές ανησυχίες για την ασφάλειά τους αλλά και την ασφάλεια του δικτύου (βλέπε [5]). Η ασφάλεια των πρακτόρων είναι εξίσου σημαντική με την ασφάλεια των κόμβων (hosts) ενός δικτύου. Οι πράκτορες πρέπει να είναι προστατευμένοι από τους κόμβους στους οποίους εκτελούνται και οι κόμβοι από τους πράκτορες που φιλοξενούν. Όταν ένας πράκτορας εκτελείται σε κάποιον εχθρικό κόμβο (malicious host), ο κόμβος αυτός μπορεί να προκαλέσει διάφορες ζημίες στον πράκτορα και στα δεδομένα του. Για παράδειγμα, αν ένας πράκτορας επισκεφτεί ένα μολυσμένο ή κακόβουλο κόμβο ενός δικτύου, ο κόμβος αυτός ίσως μπορέσει να ανακτήσει κάποιες ευαίσθητες πληροφορίες από τον πράκτορα, να αλλοιώσει τα δεδομένα ή τον κώδικά του, να αλλάξει τη συμπεριφορά του (για παράδειγμα τη διαδρομή του) ή ακόμη και να τον τερματίσει. Η λύση του προβλήματος της ανακάλυψης των εχθρικών κόμβων είναι αναγκαία αλλά αποτελεί και μεγάλη πρόκληση.

Μια λύση αποτελεί η χρήση αξιόπιστης αρχιτεκτονικής. Σε αυτή την προσέγγιση μία ανεξάρτητη και αξιόπιστη οντότητα, προμηθεύει τους κόμβους με την κατάλληλη αρχιτεκτονική, στόχος της οποίας είναι να παρέχει πλήρη ή μερική ασφάλεια για την εκτέλεση των λειτουργιών ενός πράκτορα. Η αρχιτεκτονική δεν πρέπει να εμποδίζει τις δυνατότητες των πρακτόρων (πχ επικοινωνία). Τα μειονεκτήματα αυτής της τεχνικής είναι ότι ο εξοπλισμός κοστίζει, περιορίζει τις δυνατότητες του συστήματος και πολλές φορές κάνει δύσκολη τη συντήρηση.

Μια διαφορετική προσέγγιση αποτελεί η αποκαλούμενη ασφάλεια του μαύρου κουτιού (Blackbox security). Ο συγγραφέας του [34] θεωρεί ως Blackbox έναν εκτελέσιμο (executable) πράκτορα ο κώδικας του οποίου, έχει κρυφτεί με τέτοιο τρόπο ώστε κανένας να μη μπορεί να καταλάβει τη συνάρτηση μετάβασης του. Έτσι ο πράκτορας αυτός θεωρητικά θα ήταν άτρωτος σε επιθέσεις που σκοπεύουν να

διαβάσουν ή να χειραγωγήσουν τον κώδικά του. Πρακτικά όμως δεν υπάρχει κάποιος γενικός αλγόριθμος για τη δημιουργία ενός τέτοιου πράκτορα.

Στη βιβλιογραφία έχουν προταθεί και άλλες τεχνικές (π.χ. βλέπε [5]) όπως η δημιουργία ενός αρχείου με πληροφορίες για τη λειτουργία του πράκτορα στους κόμβους που επισκέπτεται, χρήση υπογραφών (signatures) ή ψηφιακών πιστοποιητικών (digital certificates) για την ασφάλεια των πρακτόρων και την ανίχνευση επιθέσεων και αλλοιώσεων των πρακτόρων ή των δεδομένων τους. Όμως και αυτές οι τακτικές μειώνουν τις δυνατότητες των πρακτόρων ή πολλές φορές εισάγουν νέα προβλήματα ασφάλειας για την υλοποίηση τους.

Συνοψίζοντας ενώ οι πράκτορες που δεν κινούνται, μπορούν να χρησιμοποιήσουν τις ήδη υπάρχουσες τεχνικές ασφάλειας, αντίθετα οι κινητοί πράκτορες αντιμετωπίζουν προβλήματα ασφάλειας, που πολλές φορές είναι εξαιρετικά δύσκολο να ξεπεραστούν. Οι πράκτορες πρέπει να είναι σε θέση να δρουν και να παίρνουν αποφάσεις βασισμένοι σε διάφορες παραμέτρους ασφαλείας. Λύσεις για διάφορα προβλήματα ασφάλειας υπάρχουν, αλλά για να χρησιμοποιηθούν οι πράκτορες σε ένα μεγάλο εύρος εφαρμογών και να χρησιμοποιήσουν όλες τις δυνατότητες τους. ο τομέας της ασφάλειας χρειάζεται μεγαλύτερη ανάπτυξη και έρευνα.

1.2.4 Εφαρμογές κινητών πρακτόρων

Η εμφάνιση των κινητών πρακτόρων δημιούργησε πολλές νέες δυνατότητες και τράβηξε την προσοχή των ερευνητών. Ο σχεδιασμός πρακτόρων αναπτύχθηκε κυρίως στο πεδίο των κατανεμημένων συστημάτων, αν και εφαρμογές τους υπάρχουν και σε άλλες περιοχές της πληροφορικής όπως στην τεχνητή νοημοσύνη [βιβλίο 95], στη ρομποτική [βιβλίο 85],στην υπολογιστική οικονομία [βιβλίο 93] και στα δίκτυα [βιβλίο 92].

Μία από τις πιο συνηθισμένες χρήσεις των κινητών πρακτόρων είναι η αναζήτηση μιας πληροφορίας σε ένα δίκτυο. Πολλές φορές η αναζήτηση πληροφορίας γίνεται σε πολλές πηγές και οι ερωτήσεις που γίνονται πρέπει να αλλάζουν σύμφωνα με τις απαντήσεις (όπως συμβαίνει στον τομέα της εξόρυξης δεδομένων). Έτσι ένας πράκτορας, που μπορεί να επισκέπτεται τις πηγές και να αλλάζει τη συμπεριφορά του ανάλογα με τις απαντήσεις που δίνονται, είναι ιδιαίτερα χρήσιμος καθώς μπορεί να μειώσει σημαντικά τον όγκο των δεδομένων που θα χρειαζόταν να μετακινηθούν.

Συντήρηση δικτύου: σε ένα δίκτυο είναι απαραίτητο να παρέχονται στους κόμβους διάφορες υπηρεσίες πχ αναβάθμιση λογισμικού, έλεγχος ασφάλειας κτλ. Τέτοιου είδους υπηρεσίες μπορούν να τις αναλάβουν πράκτορες και να επισκέπτονται τους κόμβους για να κάνουν τη συντήρηση του δικτύου, που απαιτείται.

Ηλεκτρονικό εμπόριο: σε κάποιες περιπτώσεις μια συγκεκριμένη συναλλαγή μπορεί να απαιτεί την ταυτόχρονη επιτυχία πολλών συναλλαγών πχ όταν ετοιμάζουμε ένα ταξίδι ,πρέπει να κλείσουμε εισιτήρια, να κάνουμε κράτηση σε ξενοδοχείο και να προγραμματίσουμε τις συναντήσεις μας. Ένας κινητός πράκτορας μπορεί να κινείται μεταξύ των εφαρμογών, εξασφαλίζοντας ότι όλες οι συναλλαγές είναι σωστά προγραμματισμένες.

Εξερεύνηση από ρομπότ: σε ένα επικίνδυνο περιβάλλον έχει νόημα, τα ρομπότ να είναι τα πρώτα που θα το εξερευνήσουν. Μια απλή και φθηνή λύση είναι να χρησιμοποιηθεί μια ομάδα αυτόνομων ρομπότ, με περιορισμένες δυνατότητες, που συνεργάζονται για την εξερεύνηση της περιοχής.

1.3 Δίκτυο

Ένα δίκτυο μοντελοποιείται σαν ένα γράφημα οι κορυφές του οποίου αναπαριστούν τους υπολογιστές του δικτύου, ενώ οι ακμές αναπαριστούν τις συνδέσεις μεταξύ των υπολογιστών. Οι κόμβοι του δικτύου μπορεί να έχουν ή να μην έχουν ξεχωριστές ταυτότητες. Στην περίπτωση που οι κόμβοι έχουν όλοι τις ίδιες ταυτότητες, το δίκτυο ονομάζεται ανώνυμο και οι πράκτορες δεν μπορούν να τους ξεχωρίσουν από τις ετικέτες τους. Σε αυτή την περίπτωση, οι πράκτορες μπορούν ίσως να διαχωρίσουν τους κόμβους από κάποια δευτερεύοντα χαρακτηριστικά τους, όπως για παράδειγμα ο βαθμός του κόμβου, πολλές φορές όμως η τοπολογία είναι τέτοια που δεν το επιτρέπει.

Ένα σημαντικό χαρακτηριστικό του δικτύου είναι το κατά πόσο παρέχει πληροφορίες στους πράκτορες για να πλοηγηθούν από κόμβο σε κόμβο. Στην περίπτωση του τοπικού προσανατολισμού όλες οι εξερχόμενες ακμές από κάποιον κόμβο, έχουν διαφορετικές ετικέτες, οι ετικέτες αυτές όμως μπορεί να μην είναι ολικά συνεπείς (όπως φαίνεται στο παράδειγμα που ακολουθεί, σχήμα 1.1). Αν οι ετικέτες ικανοποιούν μια ιδιότητα ολικού προσανατολισμού, τότε το γεγονός αυτό ονομάζεται αίσθηση της κατεύθυνσης. Η αίσθηση της κατεύθυνσης έχει μεγάλη επιρροή στην επιλυσιμότητα μεγάλου αριθμού προβλημάτων, στους κατανεμημένους υπολογισμούς αλλά και στην αποδοτικότητα των μεθόδων λύσης.

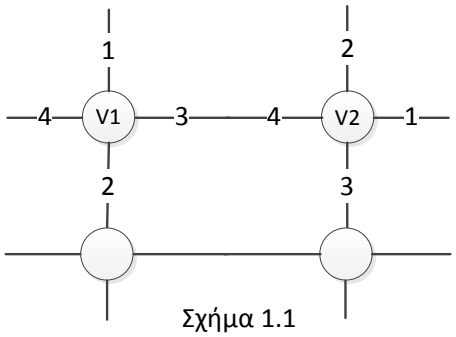
Τα δίκτυα επίσης κατηγοριοποιούνται σε σχέση με το χρονισμό. Σε ένα συγχρονισμένο δίκτυο υπάρχει ένα κεντρικό ρολόι που είναι διαθέσιμο σε όλους τους κόμβους. Συνήθως υποθέτουμε ότι σε ένα βήμα ένας πράκτορας φτάνει σε ένα κόμβο, εκτελεί κάποιους υπολογισμούς και τέλος φεύγει από τον κόμβο και όλοι οι πράκτορες εκτελούν αυτές τις διεργασίες συγχρονισμένα. Σε ένα ασύγχρονο δίκτυο το κεντρικό ρολόι δεν είναι διαθέσιμο. Σε αυτή την περίπτωση, ο χρόνος

μετακίνησης ενός πράκτορα από κόμβο σε κόμβο είναι πεπερασμένος, αλλά δεν είναι προκαθορισμένος.

Υποθέτουμε ότι οι πόροι και οι υπηρεσίες παρέχονται από τους κόμβους στους πράκτορες. Όλοι οι κόμβοι υποθέτουμε ότι παρέχουν αρκετό χώρο για τους πράκτορες και αρκετή υπολογιστική ισχύ για να διεκπεραιώσουν τις διεργασίες τους. Σε κάποιες περιπτώσεις μπορεί να υπάρχουν κακόβουλοι κόμβοι που παραπλανούν τους πράκτορες ή τους καταστρέφουν (μαύρες τρύπες). Επίσης θεωρούμε ότι ένας κόμβος μπορεί να μεταφέρει ένα πράκτορα σε κάποιον από τους γείτονες του κατόπιν αιτήματος.

Εκτός από αυτές τις βασικές υπηρεσίες, ένας κόμβος ίσως να μπορεί να προσφέρει χώρο που μπορεί να έχουν πρόσβαση όλοι οι πράκτορες (whiteboard). Μία ακόμα υπηρεσία που μπορεί να προσφέρει ένας κόμβος σε έναν πράκτορα είναι ένας μηχανισμός επικοινωνίας με άλλους πράκτορες.

Τέλος όταν εξετάζουμε την ανοχή του συστήματος σε σφάλματα, πρέπει να δούμε πώς τα στοιχεία ενός δικτύου μπορούν να αποτύχουν (crash failures, byzantine failures).



Σχήμα 1.1

Παράδειγμα 1.1

Ας υποθέσουμε πως έχουμε το παρακάτω πλέγμα (grid). Όπως φαίνεται στο σχήμα, κάθε κόμβος έχει τέσσερεις διαφορετικές ετικέτες για να προσδιορίσει την κατεύθυνση (Βοράς, Νότος, Ανατολή και Δύση) που οδηγεί κάθε ένα από τα εξερχόμενα ports. Ας υποθέσουμε επίσης πως στον κόμβο V1 υπάρχει ένας πράκτορας ο οποίος θέλει να εξερευνήσει το πλέγμα κινούμενος ανατολικά. Τότε ο πράκτορας θα φύγει από τον κόμβο V1 μέσω του port3 για να επισπευτεί τον κόμβο V2. Στη συνέχεια θα επέλεγε και πάλι το port3 για να συνεχίσει να κινείται ανατολικά, όμως αντί για ανατολικά θα βρεθεί νότια του κόμβου V2 λόγο της απουσίας της ολικής συνέπειας.

□

1.4 Έρευνα σχετική με τους κινητούς πράκτορες

Η εξερεύνηση ενός άγνωστου γραφήματος (δηλαδή ενός δικτύου) από έναν ή περισσοτέρους κινητούς πράκτορες (mobile agent) είναι ένα κλασικό πρόβλημα που έχει μελετηθεί εκτενώς (πχ βλέπε [4, 16, 31]). Ο πρώτος αλγόριθμος εξερεύνησης ενός γραφήματος από έναν πράκτορα, που μοντελοποιείται σαν πεπερασμένο αυτόματο, παρουσιάστηκε το 1951 από τον Shannon [40].

Ένα σχετικό πρόβλημα που τράβηξε το ενδιαφέρον των ερευνητών είναι το πρόβλημα της συνάντησης κινητών πρακτόρων σε ένα σημείο (Rendezvous Problem).

Έχουν μελετηθεί πολλές παραλλαγές αυτού του προβλήματος. Σε αυτές, πράκτορες ή αντίστοιχα ρομπότ κινούνται σε ένα γράφημα [2, 15, 17] στο επίπεδο ή στο χώρο [1, 3, 30]. Μερικές φορές οι πράκτορες έχουν διαφορετικές ετικέτες [14, 15, 38], ενώ σε άλλες είναι ανώνυμοι [1, 3, 30]. Πολλές φορές, οι πράκτορες χρησιμοποιούν σημάδια (tokens) [29] για την επίλυση του προβλήματος, τέλος, άλλοτε χρησιμοποιούν πιθανοτικούς [2] ή άλλοτε ντετερμινιστικούς αλγορίθμους [1, 3, 15, 29, 30, 38].

Τα τελευταία δέκα χρόνια το πρόβλημα της εξερεύνησης μελετήθηκε και σε μη ασφαλή δίκτυα, τα όποια περιέχουν εχθρικούς κόμβους, οι οποίοι αποκαλούνται μαύρες τρύπες (black holes). Μαύρη τρύπα είναι ένας κόμβος ο οποίος περιέχει μια στάσιμη διαδικασία η οποία καταστρέφει όλους τους κινητούς πράκτορες που επισκέπτονται αυτόν τον κόμβο, χωρίς να αφήνει κανένας ίχνος. Ένας τέτοιος κόμβος θα μπορούσε να είναι για παράδειγμα ένας υπολογιστής του δικτύου, ο οποίος έχει μολυνθεί με κάποιον ιό. Η διατύπωση του προβλήματος της αναζήτησης μίας μαύρης τρύπας, παρουσιάστηκε για πρώτη φορά στο [22]. Στο πρόβλημα αναζήτησης μαύρης τρύπας (Black Hole Search) στόχος των πρακτόρων είναι να εντοπίσουν τη μαύρη τρύπα σε πεπερασμένο χρόνο. Πιο συγκεκριμένα, τουλάχιστον ένας πράκτορας πρέπει να επιβιώσει γνωρίζοντας όλες τις ακμές που οδηγούν στη μαύρη τρύπα. Ο μόνος τρόπος για να εντοπιστεί μία μαύρη τρύπα, είναι τουλάχιστον ένας πράκτορας να την επισκεφτεί. Ωστόσο, αφού κάθε πράκτορας που επισκέπτεται μία μαύρη τρύπα χάνεται χωρίς να αφήνει κάποιο ίχνος, οι πράκτορες θα πρέπει να χρησιμοποιήσουν κάποιο είδος μηχανισμού επικοινωνίας έτσι ώστε να συμπεράνουν τη θέση της μαύρης τρύπας. Για την υλοποίηση τέτοιων μηχανισμών έχουν προταθεί στη βιβλιογραφία: α) το μοντέλο των πινάκων (whiteboards), β) το μοντέλο σήμανσης κόμβων και ακμών ('enhanced' token), γ) το μοντέλο σήμανσης κόμβων ('pure' token), και τέλος δ) ο μηχανισμός κατά τον οποίο οι πράκτορες (σε συγχρονισμένα δίκτυα) χρησιμοποιούν το καθολικό ρολόι για την επικοινωνία τους (time out mechanism).

α) Στο μοντέλο των πινάκων (whiteboards), σε κάθε κόμβο υπάρχει ένας χώρος μνήμης όπου οι πράκτορες που επισκέπτονται τον κόμβο μπορούν να γράψουν και

να διαβάσουν πληροφορίες. Η πρόσβαση των πρακτόρων σε κάποιο πίνακα πραγματοποιείται με αμοιβαίο αποκλεισμό (mutual exclusion), κατά τον οποίο μόνο ένας πράκτορας έχει πρόσβαση στον πίνακα κάποια συγκεκριμένη χρονική στιγμή.

Ένας μηχανισμός επικοινωνίας των πρακτόρων που χρησιμοποιεί τους πίνακες για την αναζήτηση μαύρης τρύπας, αναφέρεται στο [21] ως Cautious Walk και λειτουργεί ως εξής: ένας πράκτορας που βρίσκεται στον κόμβο ***u*** πριν εξερευνήσει κάποιον νέο μη ασφαλή κόμβο ***v***, σημειώνει στον πίνακα που βρίσκεται (δηλαδή στον u) ότι θα εξερευνήσει τον κόμβο v. Αν ο πράκτορας διαπιστώσει πως ο κόμβος v είναι ασφαλής, γυρίζει και ενημερώνει τον πίνακα στον κόμβο u. Η υλοποίηση αυτού του μοντέλου προϋποθέτει χώρο μνήμης σε κάθε κόμβο O (log n) bits, όπου n είναι ο αριθμός των κόμβων του δικτύου.

Σε αντίθεση με το μοντέλο των πινάκων, στα μοντέλα με σημάδια (tokens ή pebbles) απαιτείται λιγότερη μνήμη.

β) Στο μοντέλο σήμανσης κόμβων και ακμών ('enhanced' token), οι πράκτορες κουβαλούν πανομοιότυπα σημάδια τα οποία μπορούν να αφήσουν είτε σε ακμή είτε σε κόμβο και έχουν τη δυνατότητα να τα μετακινούν αν το επιθυμούν. Το μοντέλο για να υλοποιηθεί χρειάζεται O(log d)-bit πίνακα σε έναν κόμβο με βαθμό d (όπου d είναι ο αριθμός των ακμών που προσπίπτουν σε αυτόν τον κόμβο).

γ) Το μοντέλο σήμανσης κόμβων ('pure' token) είναι το πιο αδύναμο μοντέλο (και άρα πιο γενικό) μεταξύ αυτών που χρησιμοποιούν tokens. Σε αυτό το μοντέλο οι πράκτορες μπορούν να αφήσουν τα πανομοιότυπα σημάδια μόνο στους κόμβους. Αυτό το μοντέλο έχει χρησιμοποιηθεί στο [27], για πράκτορες που αρχικά βρίσκονται στον ίδιο κόμβο (co-located) σε ένα ασύγχρονο δίκτυο, με ένα σημάδι (pebble) για κάθε πράκτορα και στο [6] όπου οι πράκτορες ξεκινούν από διαφορετικούς κόμβους σε έναν συγχρονισμένο δακτύλιο και torus με έναν σταθερό αριθμό από tokens. Ο χώρος που απαιτείται για την υλοποίηση του μοντέλου είναι O(1)-bit.

Τέλος, **δ)** ένας ακόμα μηχανισμός είναι αυτός κατά τον οποίο οι πράκτορες χρησιμοποιούν ένα καθολικό ρολόι για την επικοινωνία τους (time out mechanism). Ο μηχανισμός αυτός είναι διαθέσιμος μόνο στα συγχρονισμένα δίκτυα. Για τη χρήση αυτού του μηχανισμού πρέπει δύο πράκτορες να βρίσκονται στον ίδιο κόμβο. Ο ένας από τους δύο εξερευνά κάποιον γειτονικό νέο μη ασφαλή κόμβο και στη συνέχεια θα πρέπει να επιστρέψει και να ενημερώσει τον πράκτορα που περιμένει. Αν ο νέος μη ασφαλής κόμβος είναι μια μαύρη τρύπα, τότε ο πράκτορας που περιμένει μπορεί να το συμπεράνει μετά από κάποιο σταθερό χρόνο.

Ο πιο ισχυρός μηχανισμός επικοινωνίας μεταξύ των πρακτόρων είναι ο μηχανισμός στον οποίο όλοι οι κόμβοι έχουν πίνακες. Επιπλέον αφού η πρόσβαση σε κάποιο πίνακα γίνεται με την τεχνική του αμοιβαίου αποκλεισμού (mutual exclusion), αυτό το μοντέλο παρέχει έναν μηχανισμό που σπάει τις συμμετρίες: αν όλοι οι πράκτορες

ξεκινούν από τον ίδιο κόμβο, μπορούν να αναθέσουν στον εαυτό τους διαφορετικές ταυτότητες. Στη συνέχεια οι (πλέον διαφορετικοί) πράκτορες μπορούν να αναθέσουν διαφορετικές ετικέτες σε όλους τους κόμβους. Συνεπώς σε αυτό το μοντέλο, αν οι πράκτορες αρχικά ξεκινούν από την ίδια θέση (co-located), τόσο οι πράκτορες όσο και οι κόμβοι μπορούμε να υποθέσουμε (χωρίς βλάβη της γενικότητας) ότι δεν είναι ανώνυμοι.

Σε ασύγχρονα (asynchronous) δίκτυα όπου θεωρούμε ότι όλοι οι πράκτορες ξεκινούν από τον ίδιο ασφαλή κόμβο, το πρόβλημα της αναζήτησης μαύρης τρύπας (BHS) έχει μελετηθεί όταν χρησιμοποιούνται πίνακες [7, 21-23], το μοντέλο σήμανσης κόμβων και ακμών (βιβλιογραφία [18, 24, 41]) και το μοντέλο σήμανσης κόμβων [27]. Έχει αποδειχτεί ότι αν οι πράκτορες ξεκινούν από τον ίδιο κόμβο, το πρόβλημα μπορεί να λυθεί σε πολυωνυμικό χρόνο χρησιμοποιώντας τον ελάχιστο αριθμό πρακτόρων. Στο σημείο αυτό είναι σημαντικό να αναφερθεί ότι στα ασύγχρονα δίκτυα οι πράκτορες πρέπει να γνωρίζουν τον αριθμό των κόμβων αλλιώς το πρόβλημα δεν λύνεται [22]. Αν η τοπολογία του γραφήματος είναι άγνωστη τότε χρειάζονται τουλάχιστον Δ+1 πράκτορες, όπου Δ είναι ο μέγιστος βαθμός κόμβου του γράφου [21]. Επιπλέον το δίκτυο πρέπει να είναι 2-συνδεδεμένο (2-connected). Επίσης είναι αδύνατον να αποφασιστεί αν υπάρχει ή όχι μαύρη τρύπα στο δίκτυο.

Σε ασύγχρονα δίκτυα όταν οι πράκτορες ξεκινούν από διαφορετικούς κόμβους (scattered), το πρόβλημα έχει μελετηθεί για την τοπολογία του δακτυλίου (ring) (βιβλιογραφία [20, 22]) και για γενικευμένες τοπολογίες [7, 28] με το μοντέλο των πινάκων (whiteboards) ενώ στο μοντέλο σήμανσης κόμβων και ακμών έχει μελετηθεί για δακτυλίους [25, 26] και για κάποια διασυνδεδεμένα δίκτυα (βιβλιογραφία [41]).

Στην περίπτωση του συγχρονισμένου δικτύου παρατηρούνται σημαντικές αλλαγές στο πρόβλημα. Σε αυτή την περίπτωση δύο διακεκριμένοι πράκτορες που βρίσκονται στον ίδιο κόμβο (co-located), μπορούν να εντοπίσουν μία μαύρη τρύπα σε οποιοδήποτε δίκτυο χρησιμοποιώντας τον μηχανισμό λήξης χρόνου (time out), χωρίς να χρειάζονται whiteboard ή tokens. Επιπρόσθετα το δίκτυο δεν χρειάζεται να είναι 2-connected πλέον, όπως στα ασύγχρονα δίκτυα, και επιπλέον τώρα είναι δυνατόν να αποφασίσουν αν υπάρχει ή όχι μαύρη τρύπα στο δίκτυο. Άρα, με διακεκριμένους πράκτορες που βρίσκονται στον ίδιο κόμβο, το θέμα δεν είναι η επιλυσιμότητα του προβλήματος της αναζήτησης της μαύρης τρύπας, αλλά η εύρεση του ταχύτερου αλγορίθμου. Το θέμα αυτό έχει μελετηθεί στα [8, 9, 12, 13, 35-37] Όταν οι πράκτορες ξεκινούν από διαφορετικές θέσεις, ο μηχανισμός λήξης χρόνου δεν επαρκεί. Στο [6] οι συγγραφείς λύνουν το πρόβλημα αναζήτησης μαύρης τρύπας, σε συγχρονισμένο δίκτυο, με διάσπαρτους πράκτορες που κουβαλούν σημάδια.

Το πρόβλημα της αναζήτησης μαύρης τρύπας έχει επίσης μελετηθεί για μία ομάδα πρακτόρων που ξεκινούν από την ίδια θέση (co-located) σε ασύγχρονα και σε συγχρονισμένα κατευθυνόμενα δίκτυα με πίνακες στα [10, 37]. Στο [9] μελετούν πώς να εντοπιστούν και να επισκευαστούν σφάλματα (πιο αδύναμα από μαύρες τρύπες) χρησιμοποιώντας πράκτορες που ξεκινούν από την ίδια θέση (co-located) σε γνωστό συγχρονισμένο δίκτυο με πίνακες και στο [32] μελετούν το πρόβλημα σε ασύγχρονο δίκτυο με πίνακες και πράκτορες που ξεκινούν από την ίδια θέση (co-located) χωρίς τη γνώση των εισερχόμενων συνδέσεων (incoming link). Μία διαφορετική επικίνδυνη συμπεριφορά έχει μελετηθεί για πράκτορες που ξεκινούν από την ίδια θέση στο [39], όπου ο συγγραφέας θεωρεί μια μαύρη τρύπα με Βυζαντινή (Byzantine) συμπεριφορά, στην οποία ένας πράκτορας που την επισκέπτεται δεν σκοτώνεται πάντα.

Ένα διαφορετικό είδος σφαλμάτων θεωρείται στο [33] όπου οι συγγραφείς θεωρούν ότι οι πράκτορες ρωτούν τους κόμβους που επισκέπτονται ώστε να τους κατευθύνουν προς ένα αντικείμενο που βρίσκεται σε κάποιον κόμβο. Οι κόμβοι μπορούν να απαντήσουν ορθά η επίτηδες εσφαλμένα.

Ακόμη, έχουν παρουσιαστεί έρευνες όπου συνδυάζουν το πρόβλημα αναζήτησης μαύρης τρύπας με το πρόβλημα rendezvous, όπου k πράκτορες είναι διάσπαρτοι στο δίκτυο και πρέπει να συναντηθούν σε έναν κόμβο παρά το γεγονός της ύπαρξης κάποιας μαύρης τρύπας, όπως για παράδειγμα στο [20].

Στην παρούσα εργασία μελετάμε το πρόβλημα της αναζήτησης μαύρων τρυπών για διασκορπισμένους (scattered) και διακεκριμένους (distinct) πράκτορες με μνήμη, σε ένα συγχρονισμένο ανώνυμο δίκτυο με βάση τον μηχανισμό κατά τον οποίο οι πράκτορες χρησιμοποιούν το καθολικό ρολόι για την επικοινωνία τους (time out mechanism).

Κεφάλαιο 2 – Περιγραφή του Μοντέλου

2.1 Εισαγωγικές έννοιες

Στο κεφάλαιο αυτό θα ασχοληθούμε με την περιγραφή του προβλήματος αναζήτησης μαύρων τρυπών. Θα αναλύσουμε τις βασικές έννοιες που χαρακτηρίζουν το πρόβλημα αναζήτησης και θα παρουσιάσουμε λεπτομερώς το μοντέλο που θα χρησιμοποιήσουμε.

Πράκτορας λογισμικού είναι ένα ανεξάρτητο πρόγραμμα το οποίο έχει την δυνατότητα να κινείται και να εκτελείται σε ένα δίκτυο υπολογιστών. Πιο συγκεκριμένα, ο πράκτορας είναι ένας ντετερμινιστικός αλγόριθμος ή πρωτόκολλο (protocol), ο οποίος μπορεί να εκτελεστεί μόνο στους κόμβους του δικτύου. Κάθε πράκτορας έχει την δυνατότητα να συνεχίσει την εκτέλεση του κώδικά του, στους κόμβους που επισκέπτεται κατά την κίνησή του στο δίκτυο. Οι πράκτορες που χρησιμοποιούμε είναι διακεκριμένοι, δηλαδή έχουν μοναδικές ταυτότητες (Ids) που τους διακρίνουν καθώς επίσης διαθέτουν μνήμη, η οποία τους επιτρέπει να 'θυμούνται' ένα σύνολο γεγονότων. Ένας πράκτορας μπορεί επίσης να αναπαρασταθεί σαν μία μηχανή Turing.

Μαύρη τρύπα (ΜΤ): Μαύρη τρύπα ονομάζεται μια επιβλαβής διαδικασία που βρίσκεται σε έναν κόμβο ενός δικτύου υπολογιστών και τερματίζει οποιονδήποτε πράκτορα λογισμικού την επισκεφτεί χωρίς να αφήνει κάποιο ίχνος. Οι πράκτορες γνωρίζουν την ύπαρξη της μαύρης τρύπας αλλά όχι την θέση της. Μια μαύρη τρύπα σε ένα δίκτυο αναπαριστά ένα εχθρικό λογισμικό (π.χ. έναν ιό που έχει μολύνει έναν υπολογιστή) ή κάποιο είδος βλάβης του υλικού ή του λογισμικού. Στην πραγματικότητα υπάρχουν πολλών ειδών σφάλματα η εχθρικά σημεία όπως για παράδειγμα κάποιο εμπόδιο που ανατρέπει ή εγκλωβίζει ένα ρομπότ ή ακόμα χειρότερα μία νάρκη η οποία καταστρέφει όποιο ρομπότ την επισκεφτεί. Είναι όμως φανερό ότι η μοντελοποίηση με μαύρες τρύπες είναι πιο γενική, με την έννοια ότι αν ένα πρόβλημα λύνεται στο μοντέλο με τις μαύρες τρύπες τότε λύνεται και στα υπόλοιπα.

Ένας δακτύλιος, είναι ένας μη κατευθυνόμενος γράφος οι κορυφές (κόμβοι) του οποίου έχουν ακριβώς δυο γείτονες. Ο δακτύλιος αναπαριστά το δίκτυο που θα χρησιμοποιηθεί.

Κάθε **Κόμβος** του δακτυλίου αναπαριστά έναν υπολογιστή συνδεδεμένο σε ένα δίκτυο υπολογιστών που φιλοξενεί προγράμματα (πράκτορες) προς εκτέλεση. Σύμφωνα με την τοπολογία του δικτύου, κάθε κόμβος έχει δύο συνδέσεις (link) προς δύο γειτονικούς κόμβους. Κάθε κόμβος ***v*** παρέχει κάποιες βασικές

πληροφορίες στους πράκτορες που τον επισκέπτονται. Οι πληροφορίες που παρέχονται είναι οι εξής: i)Προσανατολισμός, δηλαδή ποιος σύνδεσμος οδηγεί στον επόμενο κόμβο κατά την φορά των δεικτών του ρολογιού και ποιος προς την αντίθετη ii) οι ταυτότητες των πρακτόρων που βρίσκονται στον κόμβο ***v*** iii) το πρωτόκολλο επικοινωνία, το οποίο επιτρέπει στους πράκτορες του ίδιου κόμβου να επικοινωνούν. Τέλος στους κόμβους οι πράκτορες έχουν πρόσβαση στο καθολικό ρολόι το οποίο τους βοηθά στον συγχρονισμό τους. Καθώς κάθε πράκτορας καταλαμβάνει ένα μέρος της μνήμης του κόμβου που επισκέπτεται, θεωρείται ότι οι κόμβοι έχουν αρκετή μνήμη ώστε να δεχτούν έναν ή παραπάνω πράκτορες λογισμικού.

Ανώνυμο δίκτυο, είναι ένα δίκτυο υπολογιστών στο οποίο οι κόμβοι δεν έχουν (ή έχουν όλοι τις ίδιες) ετικέτες.

Η τοπολογία που χρησιμοποιείται είναι ένα **ανώνυμο δίκτυο**, τύπου δακτυλίου.

Οι πράκτορες είναι συγχρονισμένοι, δηλαδή ξεκινούν την ίδια χρονική στιγμή και αρχικά βρίσκονται στην ίδια κατάσταση ενώ η διάσχιση κάποιας ακμής διαρκεί μια χρονική μονάδα (στα συγχρονισμένα δίκτυα θεωρείται ότι υπάρχει ένα καθολικό ρολόι το οποίο βοηθά τους πράκτορες στον συγχρονισμό τους). Θεωρούμε πως, οτιδήποτε άλλο κάνει ο πράκτορας (υπολογισμούς, επικοινωνία με κάποιον άλλο πράκτορα) γίνεται στιγμιαία.

Επικοινωνία πρακτόρων. Δύο οι περισσότεροι πράκτορες μπορούν να επικοινωνήσουν μόνο όταν βρίσκονται στον ίδιο κόμβο. Οι πράκτορες μπορούν να ανταλλάξουν οποιαδήποτε πληροφορία έχουν στην μνήμη τους, καθώς επίσης έχουν την δυνατότητα να βλέπουν ο ένας την κατάσταση του άλλου.

Ασφαλής περιοχή (safe area): είναι το σύνολο των κόμβων για το οποίο κάθε πράκτορας γνωρίζει ότι είναι ασφαλές.

Σημειώνεται ότι αρχικά ο κάθε πράκτορας A θεωρεί σαν ασφαλή περιοχή S_A το σύνολο των κόμβων από τους οποίους ξεκινούν οι πράκτορες. Όταν δύο πράκτορες A, B συναντιούνται τη χρονική στιγμή t, ισχύει:

$S_A^t = S_B^t = S_A^{t-1} \cup S_B^{t-1}$

$S_i^0 = S$, όπου S το σύνολο των αρχικών θέσεων.

Επεξηγηματικά, ασφαλείς κόμβοι θεωρούνται οι αρχικές θέσεις των πρακτόρων, αφού αυτές δεν αποτελούν μαύρη τρύπα, καθώς και οι κόμβοι οι οποίοι έχουν εξερευνηθεί από κάποιον πράκτορα κατά τη διάρκεια της αναζήτησης.

Όπως θα δούμε στο λήμμα 3.4 του τρίτου κεφαλαίου, οι πράκτορες δεν μπορούν να συναντηθούν σε ανεξερεύνητο κόμβο.

Χάρτης δικτύου, είναι ένα στιγμιότυπο του δακτυλίου, στο οποίο είναι σημειωμένες οι αρχικές θέσεις και οι ταυτότητες όλων των πρακτόρων. Επιπρόσθετα μέσω του

χάρτη, προφανώς οι πράκτορες γνωρίζουν το μέγεθος (n) του δακτυλίου. Κατά τη διάρκεια της αναζήτησης οι πράκτορες σημειώνουν στον χάρτη που έχουν στην μνήμη τους ποιοί κόμβοι έχουν εξερευνηθεί (από τους ίδιους). Κάθε φορά που ένας πράκτορας συναντά κάποιον άλλο πράκτορα σε έναν κόμβο, ενημερώνει τη γνώση του για το δίκτυο.

2.2 Το πρόβλημα αναζήτησης μαύρων τρυπών

Θεωρούμε έναν δακτύλιο του οποίου κάποιοι κόμβοι είναι μαύρες τρύπες. Στον δακτύλιο υπάρχει ένας αριθμός από πράκτορες. Οι πράκτορες είναι αρχικά τοποθετημένοι έτσι ώστε σε κάθε κόμβο να υπάρχει το πολύ ένας πράκτορας. Θέλουμε να σχεδιάσουμε ένα ντετερμινιστικό αλγόριθμο που να οδηγεί τους πράκτορες να ανακαλύψουν όλες τις μαύρες τρύπες σε πεπερασμένο χρόνο.

2.3 Τυπική περιγραφή του μοντέλου

2.3.1 Το μοντέλο που θα χρησιμοποιήσουμε, περιληπτικά:

Δυνατότητες – χαρακτηριστικά που **υπάρχουν** στο μοντέλο:

- Τοπολογία δακτυλίου
- Ανώνυμο δίκτυο, n κόμβων
- Οι πράκτορες είναι συγχρονισμένοι
- Κ κινητοί πράκτορες
- Μ μαύρες τρύπες
- Οι πράκτορες ξεκινούν από διαφορετικές θέσεις
- Κάθε πράκτορας έχει μοναδική ταυτότητα (Id) μέσα από ένα ολικά διατεταγμένο σύνολο.
- Συμφωνία στον προσανατολισμό
- Χάρτης δικτύου με αρχικές θέσεις και ταυτότητες των πρακτόρων
- Επικοινωνία μεταξύ των πρακτόρων εντός του ίδιου κόμβου
- Μνήμη

Δυνατότητες – χαρακτηριστικά που **δεν υπάρχουν** στο μοντέλο:

- Αδύνατη επικοινωνία μεταξύ πρακτόρων που δεν βρίσκονται στον ίδιο κόμβο.
- Οι πράκτορες δεν μπορούν να γράψουν στους κόμβους
- Οι κόμβοι δεν έχουν ταυτότητες (labels).

Σε κάποιους από τους αλγόριθμους που θα παρουσιάσουμε στα κεφάλαια που ακολουθούν, το μοντέλο (πράκτορες – δίκτυο) μπορεί να είναι ελαφρώς διαφοροποιημένο.

2.3.2 Το μοντέλο αναλυτικά

Σε αυτό το μοντέλο υπάρχουν $K \geq 2$ πράκτορες. Οι πράκτορες αυτοί είναι στην ουσία πανομοιότυπα αυτόματα και διαφέρουν μόνο στο ότι έχουν διαφορετικές ταυτότητες. Αρχικά οι πράκτορες είναι τοποθετημένοι σε κόμβους ενός συγχρονισμένου δακτυλίου, έτσι ώστε σε κάθε κόμβο να βρίσκεται το πολύ ένας πράκτορας. Οι πράκτορες έχουν στη διάθεση τους έναν χάρτη του δακτυλίου, στον οποίο είναι σημειωμένες οι αρχικές θέσεις και οι ταυτότητες των πρακτόρων. Στο χάρτη επίσης φαίνεται ο αριθμός των κόμβων (n) του δακτυλίου. Οι πράκτορες έχουν όλοι συμφωνήσει για το ποια είναι η θετική φορά (ή αλλιώς η φορά σύμφωνα με τους δείκτες του ρολογιού) στο δακτύλιο. Στο δακτύλιο υπάρχουν επίσης $M \geq 1$ μαύρες τρύπες και το M είναι γνωστό σους πράκτορες. Οι πράκτορες μπορούν να επικοινωνήσουν μεταξύ τους, μόνο στην περίπτωση όπου βρίσκονται στον ίδιο κόμβο. Σε αυτή την περίπτωση κάθε πράκτορας μπορεί να διαβάσει την ταυτότητα την κατάσταση και τη μνήμη οποιουδήποτε από τα υπόλοιπα αυτόματα. Ο δακτύλιος είναι συγχρονισμένος και θεωρούμε ότι η διάσχιση οποιασδήποτε ακμής διαρκεί ακριβώς μία χρονική μονάδα. Τονίζουμε ότι δύο πράκτορες που διασχίζουν την ίδια ακμή ξεκινώντας από γειτονικούς κόμβους με διαφορετικές κατευθύνσεις δεν αντιλαμβάνονται ο ένας την παρουσία του άλλου.

Τη χρονική στιγμή **t** ένας πράκτορας **A** που βρίσκεται στον κόμβο **u** βλέπει ποιος άλλος πράκτορας βρίσκεται εκεί και μπορεί να διαβάσει την κατάστασή του καθενός. Με βάση αυτά τα δεδομένα και τον αλγόριθμό του, ο πράκτορας **A** μπορεί να μετακινηθεί σε κάποιον από τους γειτονικούς κόμβους του **u** ή να περιμένει εκεί. Ο πράκτορας έχει αρκετή μνήμη ώστε να «θυμάται» τις καταστάσεις του περιβάλλοντος (όπως για παράδειγμα να μετρά τους κόμβους που επισκέφτηκε). Όλοι οι πράκτορες ξεκινούν από την ίδια χρονική στιγμή, βρίσκονται στην ίδια χρονική κατάσταση και ακολουθούν τον ίδιο ντετερμινιστικό αλγόριθμο.

Πιο τυπικά, δίνεται ένα γράφημα τύπου δακτυλίου με **n** κόμβους. Σε αυτό το δακτύλιο υπάρχουν **K** πράκτορες τοποθετημένοι αρχικά το πολύ ένας σε κάθε κόμβο. Οι πράκτορες είναι συγχρονισμένοι και έχουν διαφορετικές και συγκρίσιμες μεταξύ τους ταυτότητες. Δίνεται ένας χάρτης του δακτυλίου όπου είναι σημειωμένες οι αρχικές θέσεις και οι ταυτότητες των πρακτόρων. Οι πράκτορες έχουν αρκετή μνήμη και μπορούν να επικοινωνούν όταν βρίσκονται στον ίδιο κόμβο. Στον δακτύλιο υπάρχουν M≥1 μαύρες τρύπες και ο αριθμός M είναι γνωστός στους πράκτορες. Οι πράκτορες έχουν συμφωνήσει στον προσανατολισμό του δακτυλίου και τρέχουν τον ίδιο ντετερμινιστικό αλγόριθμο. Ζητάμε να βρούμε

έναν ντετερμινιστικό αλγόριθμο που οδηγεί τους πράκτορες μετά από πεπερασμένο χρόνο στην ανακάλυψη των μαύρων τρυπών. Με τον τερματισμό του αλγορίθμου πρέπει να είναι γνωστές όλες οι ακμές που οδηγούν σε μαύρες τρύπες. Ανάλογα με τις ιδιότητες του αλγορίθμου, καθένας από τους πράκτορες πρέπει να γνωρίζει τις θέσεις των μαύρων τρυπών, τουλάχιστον ένας από τους πράκτορες πρέπει να γνωρίζει τις θέσεις των μαύρων τρυπών ή οι πράκτορες πρέπει να γνωρίζουν συλλογικά τις θέσεις των μαύρων τρυπών.

2.4 Αποτελέσματα

Αντίθετα με τις προηγούμενες μελέτες στην αναζήτηση μαύρων τρυπών, εμείς θεωρούμε το σενάριο διακεκριμένων και αρχικά διασκορπισμένων πρακτόρων με μνήμη που αναζητούν έναν αριθμό από μαύρες τρύπες σε ένα συγχρονισμένο και ανώνυμο δακτύλιο. Οι πράκτορες χρησιμοποιούν το καθολικό ρολόι για το συγχρονισμό τους (time out mechanism) και μπορούν να επικοινωνήσουν μόνο όταν βρίσκονται στον ίδιο κόμβο. Στόχος των πρακτόρων είναι να ανακαλύψουν τις θέσεις των μαύρων τρυπών, δηλαδή τις συνδέσεις (ή αλλιώς link) που οδηγούν σε αυτές.

Θεωρούμε τρείς διαφορετικές ενότητες αλγορίθμων ανάλογα με τον αριθμό των μαύρων τρυπών (μία, δύο ή ν μαύρες τρύπες). Παρουσιάζουμε βέλτιστους ντετερμινιστικούς αλγορίθμους (ως προς τον αριθμό τον αριθμό των πρακτόρων που χρησιμοποιούν) και στοχεύουμε στην σχεδίαση αλγορίθμων με μικρή χρονική πολυπλοκότητα. Παρουσιάζονται επίσης οι πολυπλοκότητες και οι αποδείξεις ορθότητας για κάθε έναν από αυτούς. Τέλος παρουσιάζεται μία προσομοίωση του μοντέλου που χρησιμοποιείται καθώς επίσης δοκιμάζουμε και έναν από τους αλγορίθμους μας.

Ο πίνακας που ακολουθεί δείχνει τις πολυπλοκότητες των αλγορίθμων, που παρουσιάζονται στα επόμενα κεφάλαια.

Πίνακας Πολυπλοκότητας Αλγορίθμων						
Όνομα		**Πολυπλοκότητα**	**ΜΤ**	**Πράκ/ρες**	**Χαρακ/στικά**	**Ιδιομορφίες**
Two_Agents		$O(3n-9)$	1	2	D≤2 *	Ακριβώς μια ΜΤ
Two_Agents**		$O(3n-6)$	≤1	2	D≤2 *	Το πολύ μία ΜΤ
Three_Agents	smallest_ID_starts	$O(4n-21)$	1	3	Χάρτη	Εύκολης κατανόησης
	smallest_ID_waits	$O(4n-31)$	1	3	Χάρτη	Καλύτερη Πολυπλοκότητας
	Without_Map	$O(4n-9)$	1	3	Κατεύθυνση Διάταξης Πρακ/ρων	Χωρίς Χάρτη
K agents only_direction		$O(n2^{\kappa})$	*1*	K	Μόνο κατεύθυνση	Χωρίς Χάρτη Χωρίς διάταξη πρακτόρων
BH2_MinID***		$O(n)$	≥2	K>6	Χάρτη	Κατηγορίας 1
BH2_MinID****		$O(n)$	≥2	K>5	Χάρτη	Κατηγορίας 2

* D είναι η ελάχιστη απόσταση μεταξύ των δύο πρακτόρων

** Με την χρήση της διαδικασίας Move_Together_maybe_1BH

*** Με την χρήση της διαδικασίας Move_Together_ver1

**** Με την χρήση της διαδικασίας Move_Together_ver2

2.5 Βασικές ιδιότητες των αλγορίθμων που ανακαλύπτουν Μαύρες Τρύπες.

Ένα σωστό σχήμα εύρεσης των μαύρων τρυπών θα πρέπει να τερματίζει βασισμένο σε μία συνθήκη τερματισμού. Στην εργασία μας όλοι οι πράκτορες που έχουν επιζήσει τερματίζουν, ενώ οι συνθήκες τερματισμού που χρησιμοποιούνται είναι οι εξής:

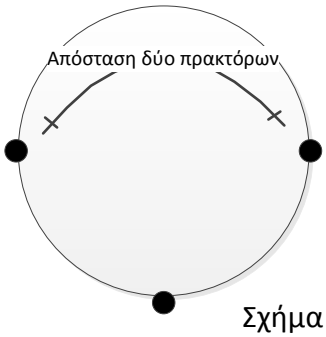
- Όλοι οι πράκτορες πριν τον τερματισμό γνωρίζουν όλες τις θέσεις των μαύρων τρυπών.
- Τουλάχιστον ένας από τους πράκτορες που τερματίζει γνωρίζει και τις δύο μαύρες τρύπες. (Σε αυτή τη περίπτωση θα υπάρχουν ακριβώς 2 μαύρες τρύπες)
- Οι πράκτορες που τερμάτισαν γνωρίζουν συλλογικά τις θέσεις των μαύρων τρυπών.

Στόχος των πρακτόρων είναι να ανακαλύψουν τις θέσεις των μαύρων τρυπών δηλαδή τις συνδέσεις (links) που οδηγούν σε αυτές. Υπάρχουν δύο τρόποι με τους οποίους μπορεί κάποιος πράκτορας να ανακαλύψει την θέση κάποιας μαύρης τρύπας. Κάποιος πράκτορας να χαθεί σε μία μαύρη τρύπα και ο άλλος να μάθει ότι ο κόμβος αυτός αποτελεί μαύρη τρύπα ή κάποιος πράκτορας να εξερευνήσει n-1

κόμβους και άρα να συμπεράνει ότι η μαύρη τρύπα βρίσκεται στον τελευταίο ανεξερεύνητο κόμβο.

Ακριβώς τη στιγμή t που ανακαλύπτει ένας πράκτορας ένα σύνδεσμο (link) που οδηγεί στη μαύρη τρύπα, μπορεί να ξέρει που ακριβώς είναι ο άλλος σύνδεσμος (αφού ξέρει το πλήθος των κόμβων του δικτύου). Συνεπώς, ακριβώς τη στιγμή t, ο πράκτορας έχει όλες τις πληροφορίες για τη θέση της μαύρης τρύπας (π.χ., η μαύρη τρύπα βρίσκεται έναν κόμβο μακριά κατά την φορά των δεικτών του ρολογιού και n-1 κατά την αντίθετη φορά). Στην περίπτωση όπου ένας πράκτορας ενημερώνει κάποιον άλλο για τη θέση μιας μαύρης τρύπας, ουσιαστικά τον ενημερώνει πόσα βήματα μακριά από αυτόν βρίσκεται η θέση της (αφού το δίκτυο είναι ανώνυμο).

Για το πρόβλημα της αναζήτησης στην παρούσα εργασία θεωρείται ότι απόσταση δύο πρακτόρων είναι το πλήθος των ακμών του ελάχιστου μονοπατιού που συνδέει τους δύο πράκτορες (χωρίς να παρεμβάλλονται άλλοι πράκτορες) και συνήθως συμβολίζεται ως d_i.



Σχήμα 2.1

2.6 Κριτήρια ποιότητας ενός αλγορίθμου

Σε ένα σύστημα κινητών πρακτόρων, υπάρχουν πολυάριθμες παράμετροι, οι οποίες επηρεάζουν την λύση ενός προβλήματος. Κάθε μεταβολή στις παραμέτρους του προβλήματος, είναι φυσικό να επηρεάζουν την δυνατότητα επίλυσης κα την πολυπλοκότητα του αντίστοιχου αλγορίθμου, που χρησιμοποιείται για την επίλυση αυτού του προβλήματος. Οι σημαντικότερες παράμετροι που επηρεάζουν την ποιότητα ενός αλγορίθμου που χρησιμοποιεί κινητούς πράκτορες είναι:

- Οι φυσικοί πόροι που χρειάζονται για την επίλυση του προβλήματος. Τέτοιοι πόροι είναι ο αριθμός των πρακτόρων και η μνήμη που χρειάζεται να έχει κάθε πράκτορας.
- Ο αριθμός των διασχίσεων που χρειάζονται οι πράκτορες.
- Ο αριθμός των σφαλμάτων που μπορεί να χειριστεί ένας αλγόριθμος, όπως για παράδειγμα ο αριθμός των μαύρων τρυπών σε ένα δίκτυο.

- Ο χρόνος που απαιτείται για την επίλυση του προβλήματος.

Ο χρόνος εκτέλεσης θα μπορούσε να διακριθεί στις ακόλουθες δύο κατηγορίες:

- Ο χρόνος που χρειάζεται κάποιος πράκτορας για να πάρει κάποιες αποφάσεις (ο χρόνος των υπολογισμών ή της «σκέψης»)
- Ο χρόνος που απαιτείται για την εκτέλεση των αποφάσεων (π.χ. για την διάσχιση μίας ακμής)

Συνήθως στα ασύγχρονα μοντέλα αυτό που ενδιαφέρει τους ερευνητές είναι ο χρόνος της δεύτερης κατηγορίας. Στο δικό μας μοντέλο, ενδιαφερόμαστε για το χρόνο της υλοποίησης των αποφάσεων και πιο συγκεκριμένα θεωρούμε ότι η επικοινωνία μεταξύ πρακτόρων, όταν είναι δυνατή, απαιτεί αμελητέο χρόνο. Επίσης οι υπολογισμοί στους αλγόριθμούς μας (όπως θα φανεί) χρειάζονται αμελητέο χρόνο. Συνεπώς στις πολυπλοκότητες των αλγορίθμων που δίνουμε μετράμε το μέγιστο αριθμό διασχίσεων (πολυπλοκότητα χειρότερης περίπτωσης) από την στιγμή που ξεκινά ο πρώτος πράκτορας έως την στιγμή τερματίσου (δηλαδή πόσες χρονικές στιγμές έχουν περάσει). Προφανώς δύο διασχίσεις από δύο πράκτορες που συμβαίνουν παράλληλα, τις μετράμε για μία.

Στην ανάλυση ενός αλγορίθμου είναι πολύ χρήσιμη η έννοια του αντιπάλου ο οποίος μπορεί να αλλάξει με όποιον τρόπο θέλει τις παραμέτρους του προβλήματος ώστε να οδηγήσει έναν αλγόριθμο σε αποτυχία ή σε όσο το δυνατόν μεγαλύτερη καθυστέρηση της επίλυσης του προβλήματος. Η ανταγωνιστικής φύσης αυτής της οντότητας, παρουσιάζεται στην συνέχεια.

2.7 Η έννοια του Ανταγωνιστή/Αντιπάλου (Adversary)

Πολλές φορές η μελέτη παρόμοιων προβλημάτων γίνεται με την εισαγωγή της έννοιας του ανταγωνιστή (adversary). Ο ανταγωνιστής μπορεί να θεωρηθεί ως μία οντότητα η οποία γνωρίζοντας όλες τις παραμέτρους του προβλήματος καθώς επίσης και τους αλγόριθμους που εκτελούν οι πράκτορες μπορεί να δώσει τιμές σε παραμέτρους που δεν είναι προκαθορισμένες (γνωστές στους πράκτορες) έτσι ώστε να κάνει την επίτευξη του στόχου τους αδύνατη ή όσο το δυνατόν πιο χρονοβόρα.

Ο ανταγωνιστής στο μοντέλο μας μπορεί κατά περίπτωση να:

- Επιλέξει τις αρχικές θέσεις των πρακτόρων μας,
- Επιλέξει τις θέσεις των μαύρων τρυπών,
- Μπορεί να επιλέξει το μέγεθος του δακτυλίου,

- Όταν δεν υπάρχει συμφωνία των πρακτόρων για τον προσανατολισμό ο ανταγωνιστής μπορεί να επιλέξει την αίσθηση του προσανατολισμού (sense of direction) που έχει κάθε πράκτορας και να
- Επιλέξει τον αριθμό των πρακτόρων.

Έτσι λοιπόν η μελέτη του προβλήματος θεωρώντας την έννοια του ανταγωνιστή καταλήγει στο να είναι ένας ανταγωνισμός μεταξύ εκείνου που σχεδιάζει την μέθοδο λύσης του προβλήματος και του ανταγωνιστή. Η έννοια του ανταγωνιστή είναι ιδιαίτερα χρήσιμη στις αποδείξεις ορθότητας και πολυπλοκότητας των αλγορίθμων.

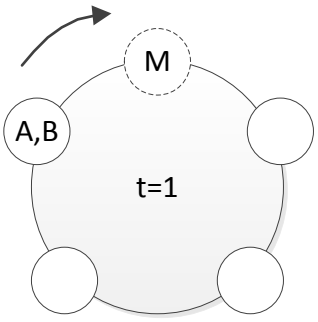
Γενικά μας ενδιαφέρει ο αλγόριθμος να βρίσκει τη σωστή λύση για οποιεσδήποτε επιλογές του ανταγωνιστή (ορθότητα).

Κεφάλαιο 3 - Αλγόριθμοι αναζήτησης μίας μαύρης τρύπας

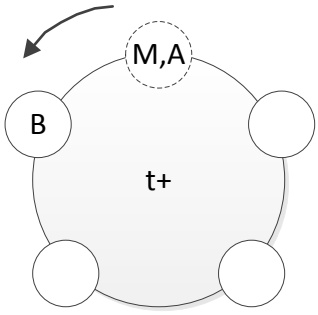
Σ’ αυτό το κεφάλαιο θα ασχοληθούμε με αλγόριθμους που αναζητούν μία μαύρη τρύπα σε ένα δακτύλιο. Επίσης θα παρουσιάσουμε την πολυπλοκότητα και τις αποδείξεις ορθότητας των αλγορίθμων.

Μία από τις βασικότερες διαδικασίες που χρησιμοποιούμε σε όλους τους αλγόριθμούς μας είναι η διαδικασία που επιτρέπει την ασφαλή εξερεύνηση ενός δικτύου όταν οι πράκτορες βρίσκονται στον ίδιο κόμβο. Η διαδικασία αυτή στη βιβλιογραφία ονομάζεται Cautious_Walk και πρωτοεμφανίστηκε στο [19] και έχει χρησιμοποιηθεί σε διάφορα μοντέλα (πχ synchronous [11], asynchronous [19]).
Βασική ιδέα του Cautious Walk:

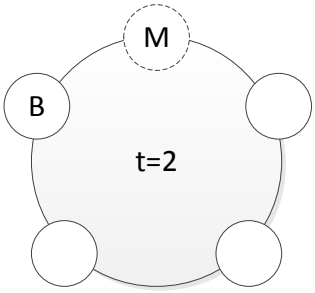
Η εξερεύνηση στο μοντέλο μας γίνεται ως εξής: Αρχικά δυο πράκτορες A,B βρίσκονται στον ίδιο κόμβο. Στη συνέχεια επιλέγεται ένας πράκτορας, έστω A, βάση ενός μοναδικού χαρακτηριστικού (συνήθως επιλέγεται ο πράκτορας με τη μικρότερη ταυτότητα) επισκέπτεται τον επόμενο κόμβο *v* και επιστρέφει πίσω. Ο άλλος πράκτορας, έστω B, περιμένει δύο χρονικές μονάδες. Αν ο πράκτορας A που κινήθηκε δεν επιστρέψει μέσα σε δυο χρονικές μονάδες, ο πράκτορας B μαθαίνει ότι η μαύρη τρύπα βρίσκεται στον κόμβο *v*. Στην περίπτωση όπου ο πράκτορας A επιστρέψει, ο κόμβος θεωρείται ασφαλής και όλοι οι πράκτορες μεταβαίνουν σε αυτόν. Στη συνέχεια επαναλαμβάνουν την ίδια διαδικασία.



Σχήμα 3.1



Σχήμα 3.2



Σχήμα 3.3

Παράδειγμα 3.1:

Έστω t=1 η χρονική στιγμή που δύο πράκτορες βρίσκονται στον ίδιο κόμβο και καλούν τη διαδικασία Cautious_Walk (Σχήμα 3.1). Ας υποθέσουμε επίσης ότι ο πράκτορας A του παραπάνω σχήματος έχει μικρότερο ID από τον πράκτορα B. Με βάση τον παρακάτω αλγόριθμο ο A θα κινηθεί όπως φαίνεται στο σχήμα την πρώτη χρονική μονάδα για να ανακαλύψει τον επόμενο κόμβο (στον οποίο έστω ότι βρίσκεται η μαύρη τρύπα με το σύμβολο M) όπου και θα χαθεί (Σχήμα 3.2). Στο

τέλος της δεύτερης χρονικής μονάδας ο Α δεν επιστρέφει, έτσι ο πράκτορας Β μαθαίνει τη θέση της μαύρης τρύπας.

□

Η διαδικασία καλείται με την εξής μορφή:

execute Procedure Cautious_Walk(Dir);

Όπου Dir είναι η κατεύθυνση που ακολουθούν οι πράκτορες για την εξερεύνηση. Δύο είναι οι πιθανές κατευθύνσεις σε έναν δακτύλιο: Η θετική κατεύθυνση και η αρνητική.

Procedure: Cautious_Walk (Dir)

```
//Procedure takes as input the direction of searching. Default direction is
clockwise.
//The following command means:
//move -> move one step
```

Let AgentA be the agent with the smalles id;
if (you are AgentA) **then**
    move at direction Dir;
    move at the opposite direction of Dir;
else `//if you are not the agent with smallest id`
    wait 2 time units;
endif
if (AgentA returned) **then**
    move at direction Dir;
else
    BH is at the next node at direction Dir;
endif

3.1 Αναζήτηση με δύο πράκτορες

Ο ελάχιστος αριθμός πρακτόρων για την αναζήτηση μίας μαύρης τρύπας, είναι δύο. Ο πρώτος αλγόριθμος που θα παρουσιάσουμε αφορά δύο πράκτορες. Οι πράκτορες ξεκινούν από διαφορετικές θέσεις, συνεπώς υπάρχει μεγάλη πιθανότητα ένας από τους δύο να χαθεί πριν οι δύο πράκτορες να συναντηθούν. Ο μόνος τρόπος όταν οι πράκτορες είναι δύο να ανακαλύψουν τη μαύρη τρύπα είναι να επέχουν το πολύ

δύο ακμές (όπως έχει αποδειχτεί στο Θεώρημα 3.1). Συνεπώς ο αλγόριθμος αυτός είναι βέλτιστος ως προς τον αριθμό των πρακτόρων που χρησιμοποιεί.

3.1.1 Δύο πράκτορες σε απόσταση $\mathbf{D \leq 2}$

Εξετάζεται το πρόβλημα της αναζήτησης **ακριβώς μιας μαύρης τρύπας** σε έναν **ανώνυμο δακτύλιο**. Η εξερεύνηση πραγματοποιείται από **δύο** πράκτορες οι οποίοι αρχικά βρίσκονται σε **διαφορετικούς κόμβους** και απέχουν απόσταση $\boldsymbol{D \leq 2}$ ακμών. Οι πράκτορες είναι συγχρονισμένοι, δηλαδή ξεκινούν την ίδια χρονική στιγμή και βρίσκονται στην ίδια αρχική κατάσταση ενώ η διάσχιση κάποιας ακμής διαρκεί μια χρονική μονάδα.

Συνοπτικά το μοντέλο περιγράφεται ως εξής:

- Δύο πράκτορες
- Ακριβώς μία μαύρη τρύπα
- Τοπολογία: Δακτυλίου (Ring)
- Ανώνυμο δίκτυο με n κόμβους
- Απόσταση μεταξύ των πρακτόρων $D <= 2$

Επιπρόσθετα οι πράκτορες:

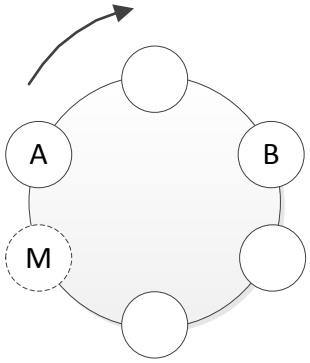
- Έχουν συμφωνήσει στον προσανατολισμό του δακτυλίου
- Κάθε πράκτορας έχει μοναδική ταυτότητα (Id) μέσα από ένα ολικά διατεταγμένο σύνολο
- Έχουν χάρτη του δικτύου με σημειωμένες τις αρχικές θέσεις των πρακτόρων
- Είναι συγχρονισμένοι
- Έχουν αρκετή μνήμη, δεν θα χρειαστεί περισσότερο από $O(n)$.

Παρατηρείται ότι στην περίπτωση όπου ο δακτύλιος έχει τρεις κόμβους η θέση της μαύρης τρύπας γίνεται αμέσως γνωστή. Συνεπώς μας ενδιαφέρουν οι περιπτώσεις όπου υπάρχουν τουλάχιστον τέσσερις κόμβοι στο δακτύλιο.

Βασική ιδέα αλγόριθμου:

Ο πράκτορας με το μικρότερο Id κινείται προς τον άλλο (ο οποίος μένει ακίνητος) ακολουθώντας το ελάχιστο μονοπάτι ή σε περίπτωση ισοπαλίας ακολουθώντας την κατεύθυνση που ορίζουν οι δείκτες του ρολογιού. Στην περίπτωση που ο πράκτορας που κινείται χαθεί στη μαύρη τρύπα, ο πράκτορας που είναι ακίνητος μπορεί να συμπεράνει τη θέση της. Εάν οι πράκτορες συναντηθούν, τότε εξερευνούν το δακτύλιο εκτελώντας τη διαδικασία Cautious_Walk.

Παράδειγμα 3.2:

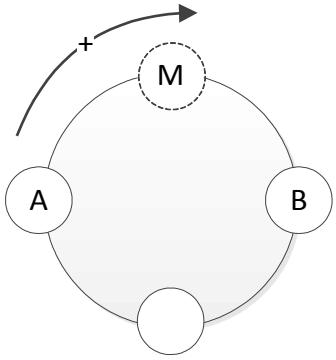


Σχήμα 3.4

Έστω A,B οι πράκτορες με Id(A)<Id(B) και M ο κόμβος που αποτελεί μαύρη τρύπα όπως στο διπλανό Σχήμα 3.4. Οι δύο πράκτορες απέχουν δυο ακμές. Με βάση τον παρακάτω αλγόριθμο, ο πράκτορας A θα ακολουθήσει τη συντομότερη εκ των δύο διαδρομών ώστε να συναντήσει τον πράκτορα B (δηλαδή την πορεία που δείχνει το βέλος του σχήματος). Όταν οι δύο πράκτορες θα συναντηθούν και θα χρησιμοποιήσουν τη τεχνική του Cautious Walk για να εντοπίσουν τη θέση της μαύρης τρύπας.

□

Παράδειγμα 3.3:



Σχήμα 3.5

Έστω το παράδειγμα του σχήματος 3.5. Τρέχοντας τον παρακάτω αλγόριθμο, ο πράκτορας A θα επιλέξει να κινηθεί προς τη μαύρη τρύπα όπου και θα χαθεί. Μετά από 2 χρονικές μονάδες ο B θα μάθει τη θέση της μαύρης τρύπας, δηλαδή στον κόμβο με το σύμβολο M (αφού ο πράκτορας A δεν τον συνάντησε).

□

Αλγόριθμος: Two_Agents

```
//The following commands means:
//move -> move one step
//wait -> wait one time unit
```

Let Agent1 be the agent with the smallest Id.

Let Agent2 be the agent with the biggest Id.

Let Dir be the direction from Agent1 to Agent2 along the shortest path or clockwise in case of ties;

```
repeat
        case 1) you are Agent1:
                move towards Dir;
        case 2) you are Agent2:
                wait;
        endcase
until ((co-located) OR (2 time units have passed))
if (not co-located) then
        BH is at the adjacent node on the reverse direction of Dir;
else
        execute Procedure Move_Together_2Agents (Dir);
endif
```

```
Procedure: Move_Together_2Agents (Dir)
repeat
  case 1) only two unexplored nodes remain:
 //in that case, a different cautious walk procedure should be followed
        if (you have the smallest Id) then
                move towards Dir;
                move at the opposite direction of Dir;
        else //you are the agent with the biggest Id
                wait 2 time units;
        endif
        if (agent with smallest id returned) then
                BH is at the last unexplored node; //BH two steps ahead
        else  //agent didn't returned
                BH is at the next node towards Dir;
        endif
  case 2) one unexplored node remains:
        BH is at that node;
  case 3) there are more than two unexplored nodes on the ring:
        execute Procedure Cautious_Walk(Dir);
  endcase
until (BH found)
```

Παρατηρείται ότι εάν η μαύρη τρύπα δεν βρίσκεται στους πρώτους $n-1$ κόμβους, τότε οι πράκτορες συμπεραίνουν τη θέση της χωρίς να χαθεί κάποιος από αυτούς.

Στο σημείο αυτό θα αποδείξουμε την ορθότητα του αλγόριθμου και την πολυπλοκότητά του.

Λήμμα 3.1
Έστω δυο πράκτορες που ξεκινούν σε απόσταση το πολύ 2 ακμών σε ένα δακτύλιο. Τρέχοντας τον παραπάνω αλγόριθμο, είτε ανακαλύπτουν τη μαύρη τρύπα μετά από το πολύ 2 χρονικές μονάδες είτε βρίσκονται στον ίδιο κόμβο.

Απόδειξη:

Έστω A, B οι δύο πράκτορες με Id(A) < Id(B). Αφού η απόσταση των πρακτόρων είναι το πολύ 2 ακμές, ο B θα περιμένει το πολύ 2 χρονικές μονάδες μέχρι να βγει από την επανάληψη του αλγορίθμου Two_Agents.

1η περίπτωση:

Ο A δεν συναντά μαύρη τρύπα. Τότε το πολύ σε 2 χρονικές μονάδες συναντά το B.

2η περίπτωση:

Ο A συναντά μαύρη τρύπα. Άρα δεν εμφανίζεται μετά από 2 χρονικές μονάδες και ο B μπορεί να συμπεράνει τη θέση της μαύρης τρύπας (είναι στο γειτονικό του κόμβο προς την αντίθετη κατεύθυνση από αυτή που κινείται ο A).

□

Λήμμα 3.2
Δυο πράκτορες μπορούν να ανακαλύψουν τη μαύρη τρύπα μέσα σε χρονική πολυπλοκότητα $3n - 9$ *χρονικές μονάδες = O(n).*

Απόδειξη:

Έστω n ο αριθμός των κόμβων του δακτυλίου και D είναι η ελάχιστη απόσταση που απέχουν οι δύο πράκτορες. Έχουμε δύο περιπτώσεις:

Περίπτωση D=2: Οι δύο πράκτορες χρειάζονται 2 χρονικές μονάδες για να συναντηθούν ή να ανακαλύψουν τη μαύρη τρύπα (Λήμμα 3.1). Στην περίπτωση που δεν ανακαλύψουν τη μαύρη τρύπα, τρέχουν τη διαδικασία Move_together η οποία καλεί τη διαδικασία Cautious Walk. Συνεπώς χρειάζονται $2 + 3(n - 5) + 2 = 3n - 11$ χρονικές μονάδες. Διότι για κάθε ανεξερεύνητο κόμβο τον οποίο οι πράκτορες εξερευνούν με τη διαδικασία Cautious_Walk (εκτός από τους δύο τελευταίους) δαπανούν τρεις χρονικές μονάδες. Όλοι οι ανεξερεύνητοι κόμβοι εκτός των δύο τελευταίων είναι $n - 5$. Για τον προτελευταίο κόμβο η εξερεύνηση κοστίζει δύο χρονικές μονάδες.

Περίπτωση D=1: Οι δύο πράκτορες χρειάζονται 1 χρονική μονάδα για να συναντηθούν (Λήμμα 3.1) και στη συνέχεια τρέχουν τη διαδικασία Move_together η οποία καλεί την Cautious Walk. Συνεπώς χρειάζονται $1 + 3(n - 4) + 2 = 3n - 9$ χρονικές μονάδες.

Η δεύτερη περίπτωση αποτελεί τη χειρότερη περίπτωση, συνεπώς η πολυπλοκότητα χειρότερης περίπτωσης είναι: **3n-9** χρονικές μονάδες, για n≥4.

□

Γενικός τύπος με χρήση της απόστασης D στη συνάρτηση πολυπλοκότητας:

$D + 3(n - D - 3) + 2$, όταν $D = \{1, 2\}$ και $n = \{5, 6, \dots\}$

Ενώ στις παρακάτω περιπτώσεις η πολυπλοκότητα είναι:

3 χρονικές μονάδες → Όταν $n = 4, D = 1$

2 χρονικές μονάδες → Όταν $n = 4, D = 2$

Συνεπώς από το λήμμα 3.1 και λήμμα 3.2 ισχύει:

Γενικό λήμμα 3.3

Δύο πράκτορες οι οποίοι απέχουν το πολύ δυο ακμές μπορούν να ανακαλύψουν ακριβώς μια μαύρη τρύπα μετά από το πολύ $3n - 9$ *χρονικές μονάδες.*

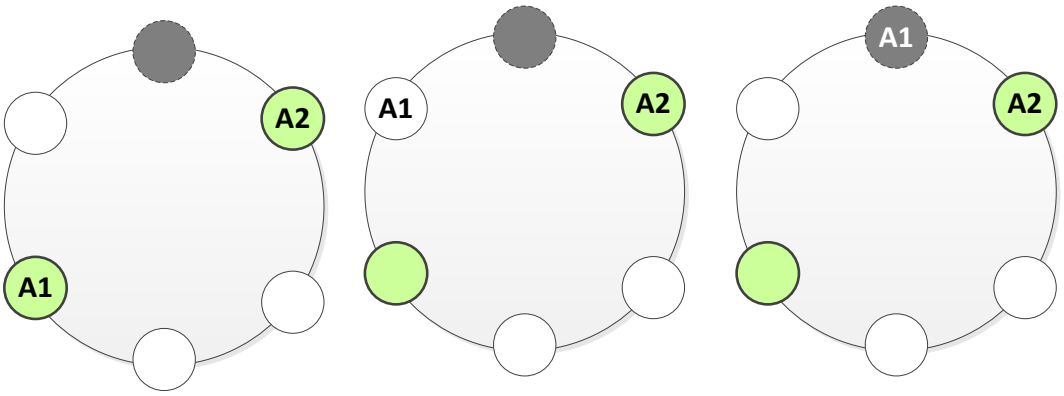
3.1.2 Δύο πράκτορες σε απόσταση $\mathbf{D > 2}$

Αν η ελάχιστη απόσταση μεταξύ των δύο πρακτόρων είναι μεγαλύτερη από δύο ακμές, η αναζήτηση της μαύρης τρύπας είναι αδύνατη για το μοντέλο που περιγράφτηκε προηγουμένως. Σε οποιονδήποτε σωστό αλγόριθμο πρέπει κάποιος από τους πράκτορες να κινηθεί για να ανακαλύψουν τη θέση της μαύρης τρύπας. Αν ο πράκτορας που πρόκειται να κινηθεί πρώτος χαθεί πριν προλάβει να συναντήσει το δεύτερο πράκτορα, μένουν περισσότερες από μία πιθανές θέσεις της μαύρης τρύπας να εξερευνήσει ο άλλος. Συνεπώς είναι αδύνατον ο πράκτορας που έμεινε να γνωρίζει την ακριβή θέση της μαύρης τρύπας χωρίς να επισκεφτεί μία από τις πιθανές θέσεις. Στην περίπτωση όπου η θέση που θα επιλέξει ο πράκτορας να εξερευνήσει αποτελεί μαύρη τρύπα τότε ο πράκτορας θα χαθεί.

Παράδειγμα 3.4:

Έστω ένας δακτύλιος με δύο πράκτορες όπως φαίνεται στο παρακάτω αριστερό σχήμα (Σχήμα 3.6). Αν κινηθεί ένας από τους δύο πράκτορες, έστω ο A1, θα χαθεί στη μαύρη τρύπα. Στη συνέχεια ο A2 γνωρίζει ότι ο A1 χάθηκε, όμως δεν γνωρίζει σε ποιόν από τους δύο κόμβους. Έτσι αν επιχειρήσει να εξερευνήσει τον κόμβο αριστερά του τότε θα χαθεί και αυτός.

□



Σχήμα 3.6

Μια έννοια που θα μας χρειαστεί στη συνέχεια είναι η παρακάτω.

Ασφαλής περιοχή (safe area): είναι το σύνολο των κόμβων για το οποίο κάθε πράκτορας γνωρίζει ότι είναι ασφαλές.

Σημειώνεται ότι αρχικά ο κάθε πράκτορας A θεωρεί σαν ασφαλή περιοχή S_A το σύνολο των κόμβων από τους οποίους ξεκινούν οι πράκτορες. Όταν δύο πράκτορες A, B συναντιούνται τη χρονική στιγμή t, ισχύει:

$S_A^t = S_B^t = S_A^{t-1} \cup S_B^{t-1}$

$S_i^0 = S$, όπου S το σύνολο των αρχικών θέσεων.

Όπως θα δούμε στο λήμμα 3.4 που ακολουθεί, οι πράκτορες δεν μπορούν να συναντηθούν σε ανεξερεύνητο κόμβο.

Λήμμα 3.4: Όμοιο με το Lemma 3.1 του [13].

Σε οποιοδήποτε αλγόριθμο επίλυσης του προβλήματος, δύο πράκτορες δεν μπορούν να εξερευνήσουν τον ίδιο μέχρι πριν ανεξερεύνητο κόμβο.

Απόδειξη:

Έστω ένας αλγόριθμος που επιλύει το πρόβλημα. Ας υποθέσουμε ότι εκτελώντας αυτόν τον αλγόριθμο οι δυο πράκτορες επιχειρούν να εξερευνήσουν τον ίδιο μέχρι πριν ανεξερεύνητο κόμβο. Αν ο κόμβος αυτός αποτελεί μαύρη τρύπα οι δύο πράκτορες θα χαθούν. Συνεπώς ένας τέτοιος αλγόριθμος δεν μπορεί να είναι σωστός.

Λήμμα 3.5:

Σε οποιοδήποτε αλγόριθμο επίλυσης του προβλήματος, πρέπει να ισχύει η εξής ιδιότητα. Μεταξύ δύο διαδοχικών επισκέψεων σε ανεξερεύνητους κόμβους από έναν πράκτορα πρέπει οπωσδήποτε να υπάρχει μία συνάντηση των πρακτόρων.

Απόδειξη:

Έστω ένας αλγόριθμος που επιλύει το πρόβλημα. Ας υποθέσουμε ότι ένας από τους πράκτορες, έστω Α, εξερευνεί τουλάχιστον δύο ανεξερεύνητους μέχρι πριν κόμβους χωρίς να συναντηθεί με τον άλλο πράκτορα, έστω Β. Τότε ο πράκτορας Α μπορεί να χαθεί σε έναν από αυτούς τους κόμβους με αποτέλεσμα ο πράκτορας Β να μην μπορεί να αποφασίσει ποιος από αυτούς τους κόμβους ήταν η μαύρη τρύπα.

Θεώρημα 3.1

Η εξερεύνηση ακριβώς μίας μαύρης τρύπας από δύο πράκτορες είναι αδύνατη αν οι πράκτορες απέχουν περισσότερο από δύο ακμές.

Απόδειξη:

Κάθε πράκτορας μπορεί να επισκεφθεί το πολύ έναν κόμβο εκτός της ασφαλούς περιοχής του. Μετά πρέπει οπωσδήποτε να συναντηθεί με τον άλλο πράκτορα σε ασφαλή κόμβο (Λήμμα 3.4), πριν επισκεφθεί οποιοδήποτε άλλο ανεξερεύνητο κόμβο (Λήμμα 3.5). Τέλος με βάση την τοπολογία ένας πράκτορας δεν μπορεί να συναντήσει τον άλλο χωρίς να επισκεφθεί διαδοχικά τουλάχιστον δύο ανεξερεύνητους κόμβους.

3.1.3 Η περίπτωση όπου υπάρχει το πολύ μία μαύρη τρύπα.

Πολλές φορές η γνώση αν υπάρχει ή όχι μαύρη τρύπα στο δίκτυο δεν είναι διαθέσιμη. Αυτή η παραλλαγή του προβλήματος σε ασύγχρονα μοντέλα κάνει την επίλυση αδύνατη (όπως είδαμε στην εισαγωγή). Στο συγχρονισμένο όμως μοντέλο που ακολουθούμε εμείς, μπορούμε να χρησιμοποιήσουμε τον μηχανισμό όπου οι πράκτορες χρησιμοποιούν το καθολικό ρολόι για την επικοινωνία τους, για την επίλυση του. Σε αυτή την περίπτωση όλοι οι κόμβοι του δικτύου πρέπει να ελεγχθούν.

Έτσι ο αλγόριθμος της προηγούμενης ενότητας μπορεί να τροποποιηθεί ώστε να ελέγχει όλους τους κόμβους. Η μόνη ουσιαστική αλλαγή στον νέο αλγόριθμο θα είναι στη διαδικασία Move_Together, η οποία πρέπει να αλλάξει ως εξής:

Procedure: Move_Together_maybe_1BH(Dir)

```
//The following command means:
//move -> move one step
```
repeat

case 1) one unexplored node remains:

if (you have the smallest Id) **then**

move towards Dir;

move at the opposite direction of Dir;

else `//you are the agent with the biggest Id`
wait 2 time units;
endif
if (agent with smallest id returned) **then**
The ring is clear; `//The program will exit`
else `//agent didn't returned`
BH is at the next node towards Dir;
endif
case 2) there are more than two unexplored nodes on the ring:
execute Procedure Cautious_Walk(Dir);
endcase
until (BH found)

Πολυπλοκότητα

Η πολυπλοκότητα του αλγορίθμου, θα αλλάξει ελαφρώς καθώς οι πράκτορες είναι αναγκασμένοι να εξερευνήσουν όλους τους κόμβους του δακτυλίου για να διαπιστώσουν αν υπάρχει μαύρη τρύπα στον δακτύλιο. Ουσιαστικά χρειάζονται τρία βήματα ακόμα. Συνεπώς η μόνη διαφορά της πολυπλοκότητας από το **Λήμμα 3.2**, θα είναι στις δύο περιπτώσεις:

Περίπτωση $D = 2$: Σε αυτή τη περίπτωση η πολυπλοκότητα θα είναι $2 + 3(n-4) + 2 = 3n - 8$ χρονικές μονάδες.

Περίπτωση $D = 1$: Σε αυτή τη περίπτωση η πολυπλοκότητα θα είναι $1 + 3(n-3) + 2 = 3n - 6$ χρονικές μονάδες.

Συνοψίζοντας η πολυπλοκότητα του αλγορίθμου θα είναι $\boldsymbol{3n - 6}$ χρονικές μονάδες.

3.2 Πρόβλημα αναζήτησης με τρεις πράκτορες

Όπως είδαμε στην προηγούμενη ενότητα (Θεώρημα 3.1) αν δύο πράκτορες έχουν απόσταση μεγαλύτερη από δύο ακμές, η εξερεύνηση είναι αδύνατη. Άρα ο ελάχιστος αριθμός πρακτόρων για την εξερεύνηση ενός δακτυλίου στη γενική περίπτωση, θα είναι τουλάχιστον τρεις πράκτορες. Συνεπώς οι αλγόριθμοι που θα παρουσιάσουμε είναι **βέλτιστοι** (optimal) ως προς τον αριθμό των πρακτόρων που χρησιμοποιούν για τη λύση του προβλήματος αναζήτησης μαύρης τρύπας και εφαρμόζονται όταν δεν υπάρχουν δύο πράκτορες οι οποίοι να απέχουν απόσταση το πολύ δύο ακμές (Στις περιπτώσεις όπου η απόσταση είναι μικρότερη από δυο ακμές εκτελείται ο αλγόριθμος Two_Agents του προηγούμενου κεφαλαίου).

Εξετάζουμε το πρόβλημα αναζήτησης **ακριβώς μίας** μαύρης τρύπας από **τρείς** πράκτορες σε έναν **ανώνυμο δακτύλιο**.

Συνοπτικά το μοντέλο περιγράφεται ως εξής:

- Τρείς πράκτορες
- Ακριβώς μία μαύρη τρύπα
- Τοπολογία: Δακτυλίου (Ring)
- Ανώνυμο δίκτυο με n κόμβους

Επιπρόσθετα οι πράκτορες:

- Κάθε πράκτορας έχει μοναδική ταυτότητα (Id) μέσα από ένα ολικά διατεταγμένο σύνολο
- Έχουν χάρτη του δικτύου με σημειωμένες τις ταυτότητες και τις αρχικές θέσεις των πρακτόρων
- Είναι συγχρονισμένοι
- Έχουν αρκετή μνήμη, δεν θα χρειαστεί περισσότερο από $O(n)$
- × **Δεν** έχουν συμφωνήσει στον προσανατολισμό του δακτυλίου

Η διαφορά του μοντέλου, με το προηγούμενο (Ενότητα 3.1.1) είναι ότι οι πράκτορες δεν έχουν συμφωνήσει στον προσανατολισμό του δακτυλίου. Όπως όμως θα δούμε, μπορούν με κάποιους υπολογισμούς να συμφωνήσουν ποία θα είναι η θετική κατεύθυνση για αυτούς.

Στον παρακάτω πίνακα, φαίνονται οι χρονικές πολυπλοκότητες των αλγορίθμων μας, που θα παρουσιάσουμε στη συνέχεια σε αυτό το κεφάλαιο.

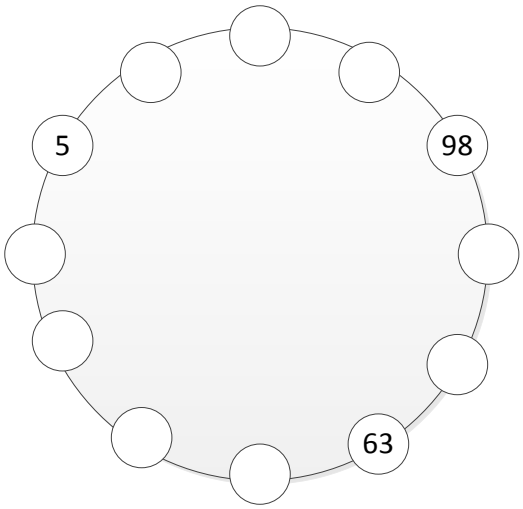
Πίνακας 3.1

Όνομα αλγορίθμου		**Πολυπλοκότητα**	**Χαρακτηριστικά**
Three_Agents	smallest ID starts	$O(\ 4n - 21\ )$	Εύκολος στην κατανόηση. Διαφορετική πολυπλοκότητα για ένα υποσύνολο περιπτώσεων (Με χάρτη)
	smallest ID waits	$O(\ 4n - \ 31\ )$	Καλύτερης πολυπλοκότητας από τον

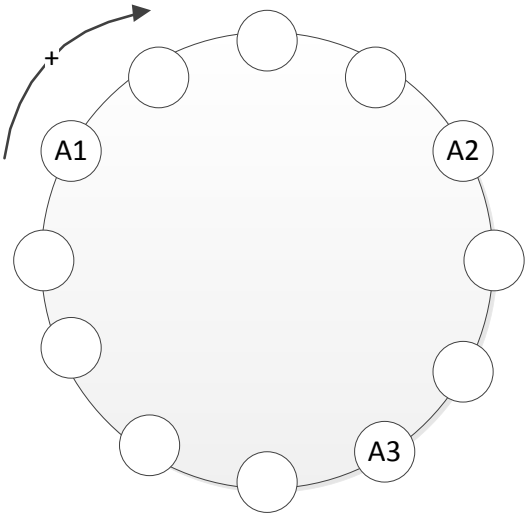
			παραπάνω (Με χάρτη)
	Without Map	$O(\ 4n - \ 9\ )$	Οι πράκτορες γνωρίζουν την διάταξη τους και την κατεύθυνση στον δακτύλιο (Χωρίς χάρτη)
	K agents only direction	$O(n2^{\kappa})$	Οι πράκτορες γνωρίζουν μόνο την κατεύθυνση του δακτυλίου (Χωρίς χάρτη)

Πριν διατυπώσουμε τους αλγορίθμους, δίνουμε ένα παράδειγμα που εξηγεί τον μηχανισμό με τον οποίο οι πράκτορες στους παραπάνω αλγορίθμους, εκτός του αλγόριθμου 'K agents only direction', χρησιμοποιούν τις μοναδικές τους ταυτότητες για να συμφωνήσουν στον προσανατολισμό του δακτυλίου.

Παράδειγμα 3.5:



Σχήμα 3.7



Σχήμα 3.8

Κάθε πράκτορας έχει στη διάθεσή του έναν χάρτη του δικτύου με τις αρχικές θέσεις όλων των πρακτόρων και τις ταυτότητες τους. Ας υποθέσουμε λοιπόν πως οι πράκτορες έχουν ως είσοδο τον παραπάνω χάρτη (Σχήμα 3.7). Οι πράκτορες, στους αλγόριθμους που παρουσιάζουμε (Three_Agents_smallest_ID_starts, Three_Agents_smallest_ID_waits και Three_Agents_Without_Map), θα μετατρέψουν τον αρχικό χάρτη (Σχήμα 3.7) στο χάρτη (Σχήμα 3.8) με βάση την παρακάτω λογική:

Αρχικά επιλέγεται ως A1 ο πράκτορας με τη μικρότερη ταυτότητα. Έτσι ο A1 θα είναι πλέον ο πράκτορας με τη ταυτότητα:5. Στη συνέχεια επιλέγεται ως A2 ο πράκτορας με τη μεγαλύτερη ταυτότητα, άρα A2 θα είναι ο πράκτορας με ταυτότητα:98. Τέλος αφού οι πράκτορες είναι μόνο τρεις, A3 θα είναι ο πράκτορας που απέμεινε, δηλαδή ο πράκτορας με την ταυτότητα: 63.

Στο σημείο αυτό, με βάση την παραπάνω 'ονομασία' οι πράκτορες μπορούν να συμφωνήσουν στον προσανατολισμό του δακτυλίου. Η εύρεσή της γίνεται ως εξής:

Η θετική κατεύθυνση είναι αυτή που ο A1 πρέπει να ακολουθήσει ώστε να φτάσει στον A2 χωρίς να περάσει από την αρχική θέση του A3. Εφόσον όλοι οι πράκτορες έχουν συμφωνήσει στις νέες ταυτότητες, η κατεύθυνση που περιγράψαμε παραπάνω βρίσκει σύμφωνους και με συνέπεια όλους τους πράκτορες.

□

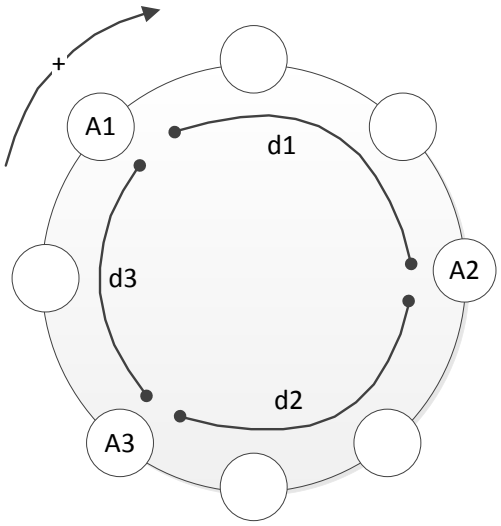
Παρατήρηση: Ο παραπάνω τρόπος επεκτείνεται για οποιονδήποτε αριθμό πρακτόρων ως εξής:

Έστω $(Id_1, \dots, Id_k)$ οι ταυτότητες των πρακτόρων σε αύξουσα διάταξη. Ανάθεσε τις ετικέτες $(A_1, \dots, A_\kappa)$ στους πράκτορες με $(Id_1, \dots, Id_k)$ αντίστοιχα. Όρισε σαν θετική κατεύθυνση την κατεύθυνση στην οποία πρέπει να κινηθεί ο A_1 για να πάει στον A_κ χωρίς να περάσει από τον A_2. Τέλος ανέθεσε τις νέες ταυτότητες $(A_1, \dots, A_\kappa)$ ξεκινώντας από τον προηγούμενο A_1 και ακολουθώντας τη θετική κατεύθυνση.

Έτσι από εδώ και στο εξής, θεωρούμε ότι οι πράκτορες έχουν συμφωνήσει στον προσανατολισμό του δακτυλίου.

Οι αλγόριθμοι μας λειτουργούν για δυο παραλλαγές του προβλήματος αναζήτησης μαύρης τρύπας. Η μία είναι εκείνη στην οποία όλοι οι πράκτορες πρέπει να ξέρουν τη θέση της μαύρης τρύπας και η δεύτερη είναι αυτή όπου κάποιοι πράκτορες έχουν ανακαλύψει τη θέση της μαύρης τρύπας χωρίς να έχουν ενημερώσει τους υπόλοιπους πράκτορες του δακτυλίου για τη θέση της. Γι' αυτό τον λόγο, στους αλγόριθμούς μας υπάρχει η παράμετρος "All_agents_should_know". Αν η τιμή της μεταβλητής είναι (false), τουλάχιστον ένας πράκτορας πρέπει να γνωρίζει τη θέση της μαύρης τρύπας. Αλλιώς αν η τιμή της μεταβλητής είναι (true) όλοι οι επιζήσαντες πράκτορες πρέπει να γνωρίζουν τη θέση της μαύρης τρύπας.

3.2.1 Κινείται ο πράκτορας με τη μικρότερη ταυτότητα



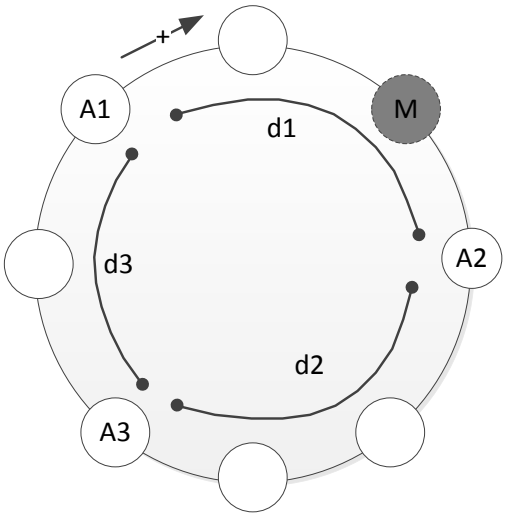
Σχήμα 3.9

Βασική Ιδέα:

Ο πράκτορας με τη μικρότερη ταυτότητα (A1) κινείται κατά τη θετική κατεύθυνση (έστω κατά τη φορά των δεικτών του ρολογιού), ενώ οι υπόλοιποι πράκτορες περιμένουν στις θέσεις τους. Αν οι πράκτορες A1 και A2 συναντηθούν, θα εξερευνήσουν το δακτύλιο καλώντας τη διαδικασία Cautious_Walk. Στην περίπτωση που ο A1 χαθεί στη μαύρη τρύπα, ο A2 θα συμπεράνει ότι η μαύρη τρύπα βρίσκεται ανάμεσα από αυτόν και τον A1. Έτσι θα κινηθεί κατά τη θετική κατεύθυνση μέχρι να συναντήσει τον A3, ο οποίος τον περιμένει. Τέλος οι δύο πράκτορες A2 και A3, θα μετακινηθούν στην κοντινότερη προς αυτούς αρχική θέση (δηλαδή την αρχική θέση του A1 ή του A2) και θα εξερευνήσουν τον τμήμα d1, όπου θα βρουν τη μαύρη τρύπα.

Παράδειγμα 3.6:

Έστω τρεις πράκτορες σε έναν δακτύλιο, με τις αρχικές θέσεις του διπλανού σχήματος (Σχήμα 3.10). Ο πράκτορας A1 (που έχει τη μικρότερη ταυτότητα) θα κινηθεί κατά τη θετική κατεύθυνση (όπως δείχνει το βέλος), ενώ οι υπόλοιποι πράκτορες θα παραμείνουν ακίνητοι. Μετά από δυο βήματα ο A2 θα χαθεί στη μαύρη τρύπα. Τη τρίτη χρονική μονάδα ο πράκτορας A2, θα διαπιστώσει ότι ο πράκτορας A1, θα έπρεπε να τον είχε συναντήσει και άρα θα συμπεράνει ότι χάθηκε στη μαύρη τρύπα. Συνεπώς θα κινηθεί κατά τη θετική κατεύθυνση για να συναντήσει τον A3. Όταν οι δύο πράκτορες συναντηθούν, θα υπολογίσουν ποια αρχική θέση είναι πιο κοντά σε αυτούς (μεταξύ των θέσεων του πράκτορα A1 και A2) και θα μεταβούν στην αρχική θέση του A1. Τέλος θα εξερευνήσουν μαζί το τμήμα d1 και θα ανακαλύψουν τη μαύρη τρύπα.



Σχήμα 3.10

□

Αλγόριθμος: **Three_Agents_smallest_ID_starts (all_agents_should_know)**

```
//The procedure takes as input:
// a flag if all agents must know or not the black hole position
//The following commands means:
//move -> move one step
//wait -> wait one time unit
```

Let *Agent1* be the agent with the smallest ID. Then *Agent2* will be the agent with the biggest ID. So *Agent3* will be the last agent.
Let *d1* be the distance between Agent1 and Agent2, *d2* be the distance between Agent2 and Agent3 and *d3* be the distance between Agent3 and Agent1.
Let **Dir** be the direction that agent1 should follow to reach Agent2 without passing through Agent 3.

if (you are Agent1) **then**
move towards Dir until you see an agent;
else `//if you are agent 2 or 3`
wait d1 time units;
endif
case 1) you see another agent: `//agent1 meets agent2`
execute Procedure Move_Together (Dir)(all_agents_should_know)(ring);
case 2) (d1 time units passed) **AND** (no agent showed up):
if (you are Agent2) **then** `//agent1 has lost into a black hole`
repeat
move towards Dir;
until (you see another agent)
Let d=min(d3,d2);
Move towards Dir to Agent1's start position or at the opposite direction of Dir to Agent2's start position depending on which is closer. In case of equality move towards Dir. Let that direction be Dir2;
execute Procedure Move_Together(Dir2)(all_agents_should_know)(d2);
endif
if (you are Agent3)
repeat
wait;
until (you see another agent) **OR** (d1+3*(d2-1)+1 time units passed)
case 1) an agent came to you, told you the BH position:
BH at that position;
`/* in a configuration that Agent 1 and 2 find the BH early enough, agent 2 meets Agent3 before d1+3*(d2-1)+1 time units */`
case 2) you see exactly one agent : `//Agent2 came to you`
Let d=min(d3,d2);
Move towards Dir to Agent1's start position or at the opposite direction of Dir to Agent2's start position depending on which is closer. In case of equality move towards Dir. Let that direction be Dir2;
execute Procedure Move_Together (Dir2)(all_agents_should_know)(d1);
case 3) all agents are at the same node:
`//at that time agent3 join Agent 2 and Agent3 who executes Move_Together`
execute Procedure Move_Together (at clockwise direction)(all_agents_should_know)(d3);
case 4) d1+3*(d2-1)+1 time units passed and no agent showed up:

```
    /* BH should be between Agent2 and Agent3 and the alive agent knows the
    BH position */
        If(all_agents_should_know)then
                wait until you see another agent;
        else
                BH found, Agent2 knows the BH position;
        endif
     endcase
   endif
endcase
```

Procedure Move_Together (Dir)(all_agents_should_know)(Segment)

```
//The procedure takes as input:
//i)   the direction that agents must follow to explore
//ii)  a flag if all agents must know or not the black hole position
//iii) the segment that they have to explore
//The following command means:
//move -> move one step
```

repeat

case 1) only two unexplored nodes remain at the Segment:

```
//in that case, a different cautious walk procedure should be followed
```

if (you have the smallest Id) **then** `//from the agents at that node`

move towards Dir;

move at the opposite direction of Dir;

else `//you are not the agent with the smallest Id`

wait 2 time units;

endif

if (agent with smallest id returned) **then**

BH is at the last unexplored node; `//BH two steps ahead`

else `//agent didn't returned`

BH is at the next node towards Dir;

endif

case 2) one unexplored node remains at the Segment:

BH is at that node;

case 3) there are more than two unexplored nodes on the ring:

execute Procedure Cautious_Walk(Dir);

case 4) next node is safe:

move towards Dir;

endcase

until (BH found)

if (you haven't met agent3) **AND** (all_agents_should_know)**then**
`//agent3 don't know the BH location`
move at the opposite direction of Dir to tell the BH location to the other agent;
endif

Ακολουθεί η απόδειξη της ορθότητας και ο υπολογισμός της πολυπλοκότητας του αλγορίθμου.

Πρόταση: ορθότητα διαδικασίας Move_together

Η διαδικασία Move_Together χρησιμοποιεί τον αλγόριθμο Cautious_Walk για την ανακάλυψη της μαύρης τρύπας σε ολόκληρο τον δακτύλιο ή σε ένα τμήμα του (όπως στο παράδειγμα 3.6 Σχήμα 3.11). Σκοπός της είναι να βελτιώσει την απόδοση της σχετικά χρονοβόρας διαδικασίας Cautious_Walk, σταματώντας την εξερεύνηση όπου δεν είναι αναγκαία. Περιπτώσεις κόμβων όπου η εξερεύνηση δεν είναι αναγκαία αποτελούν, οι αρχικές θέσεις των πρακτόρων, οι οποίες θεωρούνται ασφαλείς, και περιπτώσεις όπου τα συμπεράσματα είναι άμεσα. Για παράδειγμα, όταν υπάρχει μόνο ένας ανεξερεύνητος κόμβος και ακριβώς μία μαύρη τρύπα, οι πράκτορες μπορούν να συμπεράνουν με βεβαιότητα ότι η μαύρη τρύπα θα βρίσκεται σε αυτόν τον κόμβο.

□

Λήμμα 3.6
Τρέχοντας τον παραπάνω αλγόριθμο, μετά από d1 time units, ο Agent1 θα έχει χαθεί και ο Agent2 θα ξέρει ότι η μαύρη τρύπα βρίσκεται στο d1, εναλλακτικά, οι πράκτορες Agent1 και Agent2 θα έχουν συναντηθεί.
Απόδειξη:

Έστω ένας δακτύλιος με τρεις πράκτορες, οι οποίοι ξεκινούν από διαφορετικές αρχικές θέσεις. Το τμήμα που βρίσκεται ανάμεσα από τους πράκτορες Agent1 και Agent2 θα το ονομάζουμε d1.

Υπάρχουν δύο πιθανές περιπτώσεις ανάλογα με την θέση της μαύρης τρύπας:

Περίπτωση 1: Η μαύρη τρύπα να βρίσκεται κάπου στο τμήμα d1. Με βάση τον αλγόριθμο ο Agent1 θα κινηθεί προς την θετική κατεύθυνση και θα χαθεί στην μαύρη τρύπα. Ο Agent2 (που περίμενε) μετά από d1 χρονικές μονάδες θα διαπιστώσει ότι ο Agent1 δεν εμφανίστηκε και συνεπώς θα συμπεράνει ότι η μαύρη τρύπα βρίσκεται κάπου στο d1.

Περίπτωση 2: Η μαύρη τρύπα να βρίσκεται κάπου εκτός από το τμήμα d1. Σε αυτή την περίπτωση ο Agent1 θα κινηθεί κατά την θετική φορά και καθώς η μαύρη τρύπα δεν θα βρίσκεται στον δρόμο του, θα συναντήσει Agent2.

□

Λήμμα 3.7

Αν η θέση της μαύρης τρύπας βρίσκεται κάπου στο d1, εκτελώντας τον παραπάνω αλγόριθμο οι πράκτορες βρίσκουν την θέση της μαύρης τρύπας.
Απόδειξη:

Έστω t η χρονική στιγμή μετά από d1 time units. Με βάση το **Λήμμα 3.**, την χρονική στιγμή t, ο Agent1 θα έχει χαθεί στην μαύρη τρύπα. Τότε ο Agent2 θα κινηθεί κατά την θετική κατεύθυνση μέχρι να συναντήσει κάποιον πράκτορα, ενώ ο Agent3 θα περιμένει ακίνητος. Συνεπώς οι πράκτορες Agent2 και Agent3 θα συναντηθούν και θα μεταβούν στην κοντινότερη αρχική θέση μεταξύ των αρχικών θέσεων του Agent1 και του Agent2. Τέλος οι πράκτορες, θα εξερευνήσουν το τμήμα d1 (καλώντας την διαδικασία Move_Together) όπου θα ανακαλύψουν την μαύρη τρύπα.

□

Λήμμα 3.8

Αν η θέση της μαύρης τρύπας βρίσκεται κάπου στο d2, εκτελώντας τον παραπάνω αλγόριθμο οι πράκτορες ανακαλύπτουν την θέση της μαύρης τρύπας.
Απόδειξη

Έστω t η χρονική στιγμή μετά από d1 time units. Με βάση το **Λήμμα 3.**, την χρονική στιγμή t, οι πράκτορες Agent1 και Agent2 θα έχουν συναντηθεί, ενώ ο Agent3 θα περιμένει ακίνητος. Στη συνέχεια οι Agent1 και Agent2 θα ξεκινήσουν την εξερεύνηση του τμήματος d2 καλώντας τη διαδικασία Move_together. Κατά την διάρκεια αυτής της εξερεύνησης, ο Agent1 θα χαθεί κάπου στο d2, έτσι ο Agent2 θα μάθει την θέση της μαύρης τρύπας και θα κινηθεί κατά την αντίθετη κατεύθυνση για να ενημερώσει τον Agent3 για την θέση της.

□

Λήμμα 3.9

Αν η θέση της μαύρης τρύπας βρίσκεται κάπου στο d3 εκτελώντας τον παραπάνω αλγόριθμο οι πράκτορες βρίσκουν την θέση της μαύρης τρύπα μετά από πεπερασμένο αριθμό βημάτων.
Απόδειξη

Έστω t η χρονική στιγμή μετά από d1 time units. Με βάση το **Λήμμα 3.**, την χρονική στιγμή t ο Agent1 συναντά τον Agent2, ενώ ο Agent3 περιμένει. Στη συνέχεια οι

Agent1 και Agent2 θα εξερευνήσουν το τμήμα d2. Την χρονική στιγμή t1 οι Agent1 και Agent2 θα συναντήσουν τον Agent3, και όλοι μαζί θα εξερευνήσουν το τμήμα d3 καλώντας τη διαδικασία Move_together για την αναζήτηση της μαύρης τρύπας.

□

Γενικό Λήμμα 3.10

Ο παραπάνω αλγόριθμος εξασφαλίζει την ανακάλυψη ακριβώς μίας μαύρης τρύπας έτσι ώστε όλοι οι ζωντανοί πράκτορες να γνωρίζουν την θέση της.

Απόδειξη

Έστω πως έχουμε έναν δακτύλιο και μία μαύρη τρύπα. Η μαύρη τρύπα θα βρίσκεται κάπου μέσα σε ένα από τα τμήματα d1, d2 ή d3. Με βάση τα παραπάνω λήμματα (Λήμμα 3.7, Λήμμα 3.8, Λήμμα 3.9) όποια και αν είναι η θέση της μαύρης τρύπας σε οποιοδήποτε από αυτά ο αλγόριθμος εξασφαλίζει την ανακάλυψη της καθώς επίσης όλοι οι ζωντανοί πράκτορες ξέρουν την θέση της μαύρης τρύπας.

□

<u>Πολυπλοκότητα:</u>

Στο σημείο αυτό είναι σημαντικό να θυμίσουμε ότι ο αλγόριθμος παίρνει σαν όρισμα την παράμετρο all_agents_should_know, η οποία επηρεάζει την συνθήκη τερματισμού του αλγορίθμου. Συνεπώς επηρεάζει τον χρόνο εκτέλεσης του αλγορίθμου, άρα και την πολυπλοκότητά του. Αν και η πολυπλοκότητα χειρότερης περίπτωσης δεν επηρεάζεται σημαντικά, συμβαίνει το αντίθετο στην πολυπλοκότητα ενός συνόλου περιπτώσεων.

Για χάριν ευκολίας οι πράκτορες θα ονομάζονται A1, A2, A3 και οι αντίστοιχες αρχικές θέσεις 1,2,3.

Με βάση το σχήμα 3.9, υπάρχουν τρεις βασικές περιπτώσεις, οι οποίες επηρεάζουν την πολυπλοκότητα:

- **1η περίπτωση:** Η μαύρη τρύπα να βρίσκεται κάπου στο τμήμα d1, τότε ο τύπος της πολυπλοκότητας διαμορφώνεται ως εξής:
  $\boldsymbol{d_1 + d_2 + Min(d_2, d_3) + 3(d_1 - 3) + 2}$ χρονικές μονάδες.
  Όπου:

  $\boldsymbol{d_1}$ ο χρόνος ώστε ο A2 να διαπιστώσει ότι η μαύρη τρύπα βρίσκεται κάπου στο d1, αφού ο A1 δεν τον συνάντησε,
  $\boldsymbol{d_2}$ ο χρόνος που απαιτείται ώστε ο A2 να φτάσει στον A3 κινούμενος κατά την θετική κατεύθυνση,
  $\boldsymbol{Min(d_2, d_3)}$ ο χρόνος που απαιτείται ώστε οι πράκτορες A2,A3 να μεταβούν στην κοντινότερη αρχική θέση μεταξύ των αρχικών θέσεων του A2 και του A2 και

$\boldsymbol{3(d_1 - 3) + 2}$ ο χρόνος που απαιτείται ώστε οι A2 και A3 να εξερευνήσουν το τμήμα d1 και να βρουν την μαύρη τρύπα.
Μετά από απλοποιήσεις η σχέση παίρνει την μορφή:

$4d_1 + 2d_2 - 7$ ή $4d_1 + d_2 + d_3 - 7$

Για να βρούμε την πολυπλοκότητα, αρκεί να βρούμε το μέγιστο της παραπάνω συνάρτησης. Συνεπώς αρκεί να μεγιστοποιηθεί ο μεγαλύτερος όρος της $4d_1$ (και οι δύο εξισώσεις δίνουν το ίδιο αποτέλεσμα).

Άρα $4(n-6) + 2 * 3 - 7 = \boldsymbol{4n - 25}$, αφού ισχύει $d_i \geq 3$.

Σημείωση: Κατά τη χειρότερη περίπτωση η μαύρη τρύπα θα βρίσκετε είτε στον πρώτο είτε στον τελευταίο κόμβο του d_1, ανάλογα την αρχική θέση που θα επιλέξουν οι πράκτορες A2 και A3 να μεταβούν.

2η περίπτωση: Η μαύρη τρύπα να βρίσκετε κάπου στο d2, τότε ο τύπος της πολυπλοκότητας διαμορφώνεται ως εξής:

$\boldsymbol{d_1 + 3(d_2 - 2) + 2 + (d_2 - 2) + d_1 + d_3}$

Όπου:

$\boldsymbol{d_1}$ ο χρόνος που απαιτείται ώστε οι A1 και A2 να συναντηθούν,

$\boldsymbol{3(d_2 - 2) + 2}$ ο χρόνος που απαιτείται ώστε οι πράκτορας A1,A2 να εξερευνήσουν μαζί το τμήμα d2. Σε αυτό το στάδιο τη χειρότερη περίπτωση, αποτελεί το ενδεχόμενο η μαύρη τρύπα να βρίσκεται στον τελευταίο κόμβο του d2 κατά την θετική φορά, διότι η απόσταση που έχουν να εξερευνήσουν και στη συνέχεια να διασχίσει ο πράκτορας A2 είναι η μέγιστη. Αν η παράμετρος all_agents_should_know έχει τιμή 0 ο πράκτορας A2 μπορεί να τερματίσει.

$\boldsymbol{(d_2 - 2) + d_1 + d_3}$ είναι ο χρόνος που χρειάζεται ο A2 ώστε να μεταβεί στην αρχική του A3 κατά την αντίθετη φορά για να ενημερώσει τον A3 για την θέση της μαύρης τρύπας

Μετά από απλοποιήσεις η σχέση παίρνει την μορφή:

$2d_1 + 4d_2 + d_3 - 6 \Rightarrow n + d_1 + 3d2 - 6$ Αφού $d_1 + d_2 + d_3 = n$

Για να βρούμε την πολυπλοκότητα, αρκεί να βρούμε το μέγιστο της παραπάνω συνάρτησης. Συνεπώς αρκεί να μεγιστοποιηθεί ο μεγαλύτερος όρος της $3d_2$.

Άρα $n + 3 + 3(n-6) - 6 = \boldsymbol{4n - 21}$ αφού ισχύει $d_i > 2$.

Μετά από το πολύ $\boldsymbol{4n - 21}$ χρονικές μονάδες, και οι δύο πράκτορες θα έχουν τερματίσει.

Αν τουλάχιστον ένας πράκτορας πρέπει να γνωρίζει τη θέση της μαύρης τρύπας (δηλαδή η τιμή της παραμέτρου all_agents_should_know έχει την τιμή 1), $d_1 + 3(d_2 - 2) + 2$ χρονικές μονάδες αρκούν για την εξερεύνηση. Σε αυτή την περίπτωση, η πολυπλοκότητα όμοια θα είναι της μορφής: $\boldsymbol{3n - 15}$

3η περίπτωση: Η μαύρη τρύπα να βρίσκετε κάπου στο d3.

$\boldsymbol{d_1 + 3(d_2 - 1) + 1 + 3(d_3 - 3) + 2}$

Όπου:

$\boldsymbol{d_1}$ ο χρόνος που απαιτείται ώστε οι A1 και A2 να συναντηθούν,

$\boldsymbol{3(d_2 - 1) + 1}$ ο χρόνος που απαιτείται ώστε οι A1 και A2 να συναντήσουν τον A3 εξερευνώντας το τμήμα d2,

$\boldsymbol{3(d_3 - 3) + 2}$ ο χρόνος που απαιτείται ώστε οι A1,A2 και A3 να εξερευνήσουν μαζί το τμήμα d3.

Μετά από απλοποιήσεις η σχέση παίρνει την μορφή:

$d_1 + 3d_2 + 3d_3 - 9 => n + 2d_2 + 2d_3 - 9$ Αφού $d_1 + d_2 + d_3 = n$

Για να βρούμε την πολυπλοκότητα, αρκεί να βρούμε το μέγιστο της παραπάνω συνάρτησης. Συνεπώς αρκεί να μεγιστοποιηθεί ο μεγαλύτερος όρος της, $2d_2$ ή $2d_3$.

Άρα $n + 2(n - 6) + 2 * 3 - 9 = \boldsymbol{3n - 15}$ αφού ισχύει $d_i > 2$.

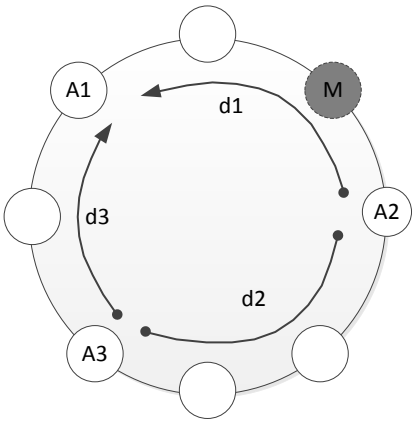
Σημείωση: Η χειρότερη περίπτωση θα είναι όταν η μαύρη τρύπα θα βρίσκεται στον τελευταίο κόμβο του d3 κατά την θετική φορά. Έτσι οι πράκτορες θα έχουν να εξερευνήσουν την μεγαλύτερη δυνατή απόσταση.

Συνοψίζοντας, η πολυπλοκότητα του αλγορίθμου είναι της τάξης: $\boldsymbol{4n - 21}$ χρονικές μονάδες .

3.2.2 Ο πράκτορας με τη μικρότερη ταυτότητα περιμένει

Βασική Ιδέα: Ο πράκτορας με την μικρότερη ταυτότητα (A1) περιμένει. Οι δύο άλλοι πράκτορες (A2 και A3) κινούνται με αντίθετες κατευθύνσεις προς τον πράκτορα A1, όπως για παράδειγμα φαίνεται στο σχήμα 3.11. Αφού υπάρχει ακριβώς μία μαύρη τρύπα, τουλάχιστον ένας πράκτορας θα συναντήσει τον A1. Ανάλογα ποιος ή πόσοι πράκτορες συναντήσουν τον A1, οι ζωντανοί πράκτορες θα μπορούν να συμπεράνουν σε ποιο από τα τμήματα d1, d2 ή d3 βρίσκεται η μαύρη τρύπα.

Παράδειγμα 3.7



Σχήμα 3.11

Έστω τρείς πράκτορες (A1, A2 και A3) με διαφορετικά και συγκρίσιμα Id και (M) μία μαύρη τρύπα. Ας υποθέσουμε ακόμη, ότι οι πράκτορες έχουν την διάταξη του διπλανού σχήματος 3.14. Τότε τρέχοντας τον παρακάτω αλγόριθμο οι πράκτορες A3 και A2 θα κινηθούν προς την κατεύθυνση που δείχνουν τα βέλη για να συναντήσουν τον πράκτορα A1. Έτσι μετά από μία χρονική μονάδα, ο πράκτορας A2 θα χαθεί στην μαύρη τρύπα. Όταν θα έχουν περάσει συνολικά 3 χρονικές μονάδες (από την αρχή) οι πράκτορες A1 και A3 θα έχουν συναντηθεί και θα γνωρίζουν ότι ο A2 χάθηκε. Οι πράκτορες έχοντας αυτή τη πληροφορία θα εξερευνήσουν μαζί το τμήμα d1 καλώντας τη διαδικασία Move_Together(Dir)(d1) με όρισμα την κατεύθυνση της αναζήτησης (την θετική κατεύθυνση) και το τμήμα που θα πρέπει να εξερευνηθεί (δηλαδή το d1).

□

Όπως αναφέρεται στο μοντέλο, κάθε πράκτορας βλέπει τις καταστάσεις της μετάβασης των πρακτόρων του βρίσκονται στον ίδιο κόμβο με αυτόν, συνεπώς και την κατεύθυνση της κίνησης τους. Έτσι οι πράκτορες χρησιμοποιούν αυτή την πληροφορία, όπως θα δούμε στον αλγόριθμο, για να συμφωνήσουν ως προς την κατεύθυνση της εξερεύνησης και την σωστή χρήση της διαδικασίας Move_Together.

Στον αλγόριθμο που ακολουθεί, οι πράκτορες δεν χρειάζονται την παράμετρο all_agents_should_know καθώς όπως γίνεται εύκολα αντιληπτό οι πράκτορες είναι μαζί κατά την διάρκεια της εξερεύνησης (άρα την στιγμή της ανακάλυψης της μαύρης τρύπας όλοι οι πράκτορες θα μάθουν την θέση της).

Αλγόριθμος: Three_agents_smallest_ID_waits

```
//The procedure takes as input:
// a flag if all agents must know or not the black hole position
//The following commands means:
//move -> move one step
//wait -> wait one time unit
```

Let *Agent1* be the agent with the smallest ID. Then *Agent2* will be the agent with the biggest ID. So *Agent3* will be the last agent.

Let *d1* be the distance between Agent1 and Agent2, *d2* be the distance between Agent2 and Agent3 and *d3* be the distance between Agent3 and Agent1.
Let **Dir** be the direction that agent1 should follow to reach Agent2 without passing through Agent 3.
Let d=MAX(d1,d3)

case 1) you are Agent1:
wait d time units;

```
//if d1!=d3 then Agent2 or Agent3 should wait for the other to came.
```

case 2) you are Agent2:
move d1 steps at opposite direction of Dir;
wait d-d1 time units;

case 3) you are Agent3:
move d3 steps toward Dir;
wait d-d3 time units;

if (you see all agents in that node) **then**
if (d2 has only one node) **then**

```
//if black hole found code stops here
```

BH at that node;
end
move d1 steps toward Dir OR d3 at the opposite direction of Dir depending on which one has the minimum distance. In case of equality choose that in direction Dir;
Let **Dir2** be the direction you followed before;
execute Procedure Move_Together(Dir2)(false)(d2)

```
//uses the direction chosen in previous step
//moves the agents at the nearest start position through the safe area
//if black hole found after Move_Together code stops here!
```

end
if (you are Agent1) **then**
case 1) agent 2 showed up:
execute Procedure Move_Together (for the opposite direction of Dir)(d3);

```
//explores the unexplored d3
```

case 2) agent 3 showed up:
execute Procedure Move_Together (Dir)(false)(d1);

```
//explore the unexplored d1
```

else `//you are agent 2 or agent3`
execute Procedure Move_Together(for the direction you followed before)(d3);
end

Στο σημείο αυτό θα αποδείξουμε την ορθότητα του αλγορίθμου και την πολυπλοκότητά του.

Απόδειξη Ορθότητας:

Λήμμα 3.11
Τρέχοντας *τον παραπάνω αλγόριθμο, μετά από d=MAX(d1,d3) χρονικές μονάδες, όλοι οι ζωντανοί πράκτορες βρίσκονται στην αρχική θέση του πράκτορα Agent1.*

Απόδειξη:

Έστω ένας δακτύλιος n κόμβων όπου υπάρχουν τρείς πράκτορες και μία μαύρη τρύπα. Τότε η μαύρη τρύπα θα βρίσκεται σε ένα από τα τρία τμήματα d1, d2 και d3. Αν η μαύρη τρύπα βρίσκεται στο τμήμα d1 τότε ο Agent2 θα χαθεί στην πορεία προς την αρχική θέση του Agent1, έτσι μόνο ο Agent1 και ο Agent3 θα συναντηθούν. Αν η θέση της μαύρης τρυπάς είναι κάπου στο d2 τρέχοντας τον παραπάνω αλγόριθμο, μετά από d χρονικές μονάδες όλοι οι πράκτορες θα βρίσκονται στην αρχική θέση του Agent1. Στην τελευταία περίπτωση όπου η μαύρη τρύπα βρίσκεται στο τμήμα d3, αντίστοιχα με την πρώτη περίπτωση χάνεται ο Agent3 και συναντιούνται ο Agent1 με τον Agent2.

□

Γενικό Λήμμα 3.12
*Ο αλγόριθμος **Three_agents_smallest_ID_waits** λύνει το πρόβλημα της αναζήτησης μαύρης τρύπας σε οποιοδήποτε δακτύλιο με τρείς πράκτορες.*

Απόδειξη:

Σύμφωνα με το **Λήμμα 3.11** μετά από d χρονικές μονάδες, όλοι οι ζωντανοί πράκτορες βρίσκονται στον ίδιο κόμβο. Αν όλοι οι πράκτορες έχουν επιβιώσει τότε η μαύρη τρύπα θα βρίσκεται σίγουρα στο τμήμα d2, οπότε οι τρείς πράκτορες σύμφωνα με τον αλγόριθμο θα κινηθούν στην κοντινότερη αρχική θέση μεταξύ των αρχικών θέσεων του Agent2 και Agent3. Στη συνέχεια θα καλέσουν τη διαδικασία Move_Together για την εξερεύνηση του τμήματος d2.

Στην περίπτωση όπου ένας από τους πράκτορες χάνεται, οι πράκτορες βλέπουν ποίος πράκτορας δεν φάνηκε στη συνάντηση που είχαν ορίσει στην αρχική θέση του Agent1. Στη συνέχεια οι πράκτορες θα καλέσουν τη διαδικασία Move_Together και θα εξερευνήσουν το τμήμα που βρίσκεται προς το μέρος του πράκτορα που δεν εμφανίστηκε στη συνάντηση. Συνεπώς θα ανακαλύψουν την μαύρη τρύπα.

Διαπιστώνεται ότι ο αλγόριθμος αυτός εξασφαλίζει την ανακάλυψη ακριβώς μίας μαύρης τρύπας, ανεξάρτητα από την θέση της στο δακτύλιο. Τέλος σε κάθε περίπτωση όλοι οι πράκτορες που έχουν επιβιώσει γνωρίζουν τη θέση της μαύρης τρύπας, καθώς με βάση το **Λήμμα 3.11** αρχικά όλοι οι επιζήσαντες πράκτορες έχουν

συναντηθεί και στη συνέχεια εξερευνούν τον δακτύλιο όλοι μαζί (άρα αν ανακαλύψουν την μαύρη τρύπα θα το μάθουν όλοι).

□

Πολυπλοκότητα:

Για χάριν ευκολίας οι πράκτορες θα ονομάζονται A1, A2, A3 και οι αντίστοιχες αρχικές θέσεις τους 1, 2, 3. Με βάση το παραπάνω σχήμα, υπάρχουν τρεις βασικές περιπτώσεις οι οποίες επηρεάζουν τον χρόνο εκτέλεσης.

- **1η περίπτωση**: Η μαύρη τρύπα να βρίσκεται κάπου στο τμήμα μήκους d1, τότε ο τύπος της πολυπλοκότητα διαμορφώνεται ως εξής:

  $\boldsymbol{Max(d1, d3) + 3(d1 - 3) + 2}$ **χρονικές μονάδες.**

  Οι πράκτορες A2 και A3 χρειάζονται $\boldsymbol{Max(d1, d3)}$ χρονικές μονάδες για να φτάσουν και οι δύο στην αρχική θέση του A1. Ο πράκτορας A2 θα χαθεί, έτσι οι πράκτορες A1 και A3 χρειάζονται ακόμη $\boldsymbol{3(d1 - 3) + 2}$ χρονικές μονάδες για να εξερευνήσουν το τμήμα μήκους d1. Κατά τη χειρότερη περίπτωση, η μαύρη τρύπα θα βρίσκεται έναν κόμβο αριστερά από την αρχική θέση του A2.

  Μετά από απλοποιήσεις η σχέση παίρνει την μορφή:

  $4d_1 - 7$ (αν $d_1 \geq d_3$) ή $3d_1 + d_2 - 7$ (αν $d_3 \geq d_1$)

  Για να βρούμε την πολυπλοκότητα, αρκεί να βρούμε το μέγιστο των παραπάνω συναρτήσεων. Συνεπώς αρκεί να μεγιστοποιηθεί ο μεγαλύτερος όρος της 4d1 της πρώτης εξίσωσης.

  Άρα $4(n - 6) - 7 = \boldsymbol{4n - 31}$ χρονικές μονάδες, αφού ισχύει $d_i \geq 3$.

- **2η περίπτωση:** Η μαύρη τρύπα να βρίσκεται κάπου στο τμήμα μήκους d2, τότε ο τύπος της πολυπλοκότητα διαμορφώνεται ως εξής:

  $\boldsymbol{Max(d_1, d_3) + Min(d_1, d_3) + 3(d_2 - 3) + 2}$ χρονικές μονάδες

  Μετά από $\boldsymbol{Max(d_1, d_3)}$ χρονικές μονάδες και οι τρείς πράκτορες θα έχουν συναντηθεί στην αρχική θέση του πράκτορα A1 γνωρίζοντας ότι η μαύρη τρύπα βρίσκεται κάπου στο τμήμα μήκους d2, έτσι θα διαλέξουν το συντομότερο μονοπάτι προς το d3 το οποίο και θα διανύσουν σε $\boldsymbol{Min(d_1, d_3)}$ χρονικές μονάδες. Τέλος θα ανακαλύψουν τη μαύρη τρύπα σε $\boldsymbol{3(d_2 - 3) + 2}$ χρονικές μονάδες η οποία θα βρίσκεται, κατά τη χειρότερη περίπτωση, είτε δεξιά από την αρχική θέση του A2 ή αριστερά από την αρχική θέση του A3 ανάλογα ποια διαδρομή ακολούθησαν.

  Μετά από απλοποιήσεις η σχέση παίρνει την μορφή:

  $d_1 + 3d_2 + d_3 - 7 => n + 2d_2 - 7$ *(αφού* $n = d_1 + d_2 + d_3$*)*

Παρατηρήστε ότι το άθροισμα $Max(d_1, d_3) + Min(d_1, d_3) = d_1 + d_3$.

Άρα $n + 2(n-6) - 7 = \mathbf{3n - 19}$ χρονικές μονάδες, αφού ισχύει $d_i \geq 3$.

- **3η περίπτωση**: Η μαύρη τρύπα να βρίσκεται κάπου στο τμήμα μήκους d3, τότε ο τύπος της πολυπλοκότητα διαμορφώνεται ως εξής:

  $\boldsymbol{Max(d_1, d_3) + 3(d_3 - 3) + 2}$

  Αντίστοιχα με την 1η περίπτωση, μετά από $\boldsymbol{Max(d_1, d_3)}$ χρονικές μονάδες θα έχει χαθεί ο πράκτορας Α3 ενώ οι υπόλοιποι δύο πράκτορες θα ξεκινήσουν την εξερεύνηση του τμήματος μήκους d3. Μετά από $\boldsymbol{3(d_3 - 3) + 2}$ χρονικές μονάδες οι δύο πράκτορες θα έχουν ανακαλύψει την θέση της μαύρης τρύπας, η οποία κατά την χειρότερη περίπτωση θα βρίσκεται μία θέση δεξιά από την αρχική θέση του πράκτορα Α3.

  Μετά από απλοποιήσεις η σχέση παίρνει την μορφή:

  $d_1 + 3d_3 - 7$ *(Αν* $d_1 > d_3$*)* ή $4d_3 - 7$*(Αν* $d_3 > d_1$*)*

  Για να βρούμε την πολυπλοκότητα, αρκεί να βρούμε το μέγιστο των παραπάνω συναρτήσεων. Συνεπώς αρκεί να μεγιστοποιηθεί ο μεγαλύτερος όρος της 4d3 της πρώτης εξίσωσης.

  Άρα $4(n-6) - 7 = \mathbf{4n - 31}$ χρονικές μονάδες, αφού ισχύει $d_i \geq 3$.

Συνοψίζοντας η πολυπλοκότητα χειρότερης περίπτωσης του αλγορίθμου θα είναι: $\mathbf{4n - 31}$ χρονικές μονάδες.

3.2.3 Κινείται ο πράκτορας με την μικρότερη απόσταση μπροστά του

Γνωρίζουμε λοιπόν ότι για τη αναζήτηση ακριβώς μίας μαύρης τρύπας, οι βέλτιστοι αλγόριθμοι ως προς τον αριθμό των πρακτόρων, χρησιμοποιούν τρείς πράκτορες. Ένα ενδιαφέρον ερώτημα είναι ποιος αλγόριθμος έχει τη μικρότερη πολυπλοκότητα.

Για παράδειγμα ένας ακόμα αλγόριθμος, που διαφέρει ελάχιστα από τον αλγόριθμο Three_Agents_smallest_ID_starts, θα ήταν ένας αλγόριθμος ο οποίος επιλέγει να κινηθεί ο πράκτορας που έχει να διανύσει την μικρότερη απόσταση προς τον επόμενο πράκτορα. Σε αυτή την περίπτωση ελέγχονται ένα υποσύνολο περιπτώσεων όπου τα αποτελέσματα εξάγονται άμεσα (όπως για παράδειγμα αν δύο πράκτορες Α, Β απέχουν μόνο δύο ακμές, αν ο πράκτορας Α κινούμενος προς τον Β χαθεί, ο πράκτορας Β θα μπορούσε να συμπεράνει με βεβαιότητα την θέση της μαύρης τρύπας ανάμεσά τους). Ένα άλλο επίσης ενδιαφέρον ερώτημα, είναι ποιος αλγόριθμος καταστρέφει τον μικρότερο αριθμό πρακτόρων.

Όταν ξεκινά ένα σχήμα αναζήτησης, οι πράκτορες ξεκινούν από διαφορετικές θέσεις. Ένας πράκτορας που κινείται σε ανεξερεύνητους κόμβους ενός δακτυλίου, έχει μεγάλη πιθανότητα να χαθεί μέσα σε μία μαύρη τρύπα. Μάλιστα όσο οι ανεξερεύνητοι κόμβοι λιγοστεύουν, το ενδεχόμενο ο πράκτορας να χαθεί, γίνεται όλο και πιο βέβαιο. Η λογική του παραπάνω αλγόριθμου που περιγράψαμε θα μπορούσε εύκολα να μειώσει τον αριθμό των πρακτόρων που χάνονται καθώς όσο λιγότερα βήματα κάνει ένας πράκτορας πριν συναντήσει κάποιον άλλο, τόσο μεγαλύτερη πιθανότητα έχει να επιβιώσει και συνεπώς να συμβάλει στην εξερεύνηση.

Η εύρεση των καλύτερων αλγορίθμων είναι ένα ενδιαφέρον ανοιχτό πρόβλημα.

3.3 Πρόβλημα αναζήτησης με λιγότερα χαρακτηριστικά

Σε αυτή την ενότητα θα ασχοληθούμε με αλγορίθμους, οι οποίοι λύνουν το πρόβλημα της αναζήτησης μίας μαύρης τρύπας, χρησιμοποιώντας λιγότερες πληροφορίες (για παράδειγμα χωρίς να γνωρίζουν τις αρχικές θέσεις των πρακτόρων). Συνεπώς οι αλγόριθμοι που θα χρησιμοποιηθούν είναι πιο ισχυροί συγκριτικά με τους προηγούμενους που αναφέρθηκαν.

3.3.1 Χωρίς χάρτη, μόνο με τη σειρά των πρακτόρων

Συνοπτικά το μοντέλο περιγράφεται ως εξής:

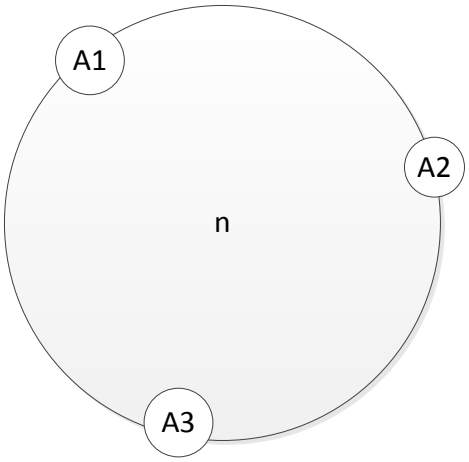
- Τρεις πράκτορες
- Ακριβώς μία μαύρη τρύπα
- Τοπολογία: Δακτυλίου (Ring)
- Ανώνυμο δίκτυο με n κόμβους

Επιπρόσθετα οι πράκτορες:

- Κάθε πράκτορας έχει μοναδική ταυτότητα (Id) μέσα από ένα ολικά διατεταγμένο σύνολο και γνωρίζει όλες τις ταυτότητες των πρακτόρων
- Γνωρίζουν τον αριθμό των κόμβων του δικτύου (n)
- Γνωρίζουν με ποια σειρά είναι διατεταγμένοι στον δακτύλιο.
- Είναι συγχρονισμένοι
- Έχουν αρκετή μνήμη, δεν θα χρειαστεί περισσότερο από $O(n)$.

× **Δεν** έχουν συμφωνήσει στον προσανατολισμό του δακτυλίου
× **Δεν** έχουν χάρτη του δικτύου με σημειωμένες τις αρχικές θέσεις των πρακτόρων

Παράδειγμα 3.7



Σχήμα 3.12

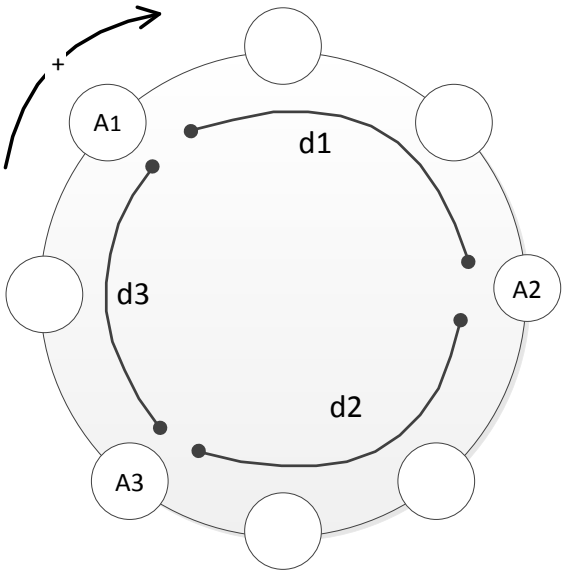
Οι πράκτορες βλέπουν το δίκτυο με την μορφή του διπλανού σχήματος, δηλαδή ξέρουν την σειρά με την οποία έχουν τοποθετηθεί στο δίκτυο και το μέγεθος του αλλά δεν γνωρίζουν τις ακριβείς θέσεις τους πάνω σε αυτό. Συνεπώς αφού δεν γνωρίζουν τις αποστάσεις μεταξύ τους, δεν μπορούν να υπολογίσουν τον ακριβές χρόνο που θα χρειαζόταν για μία συνάντηση. Βέβαια οι πράκτορες μπορούν να μετρούν τους κόμβους που επισκέφτηκαν ώστε να αντιλαμβάνονται την έννοια της απόστασης (για παράδειγμα, ώστε να μπορεί να ενημερώσει ο ένας τον άλλο για την θέση της μαύρης τρύπας).

□

Βασική ιδέα: Η βασική ιδέα του αλγορίθμου περιγράφεται στην παράγραφο που ακολουθεί. Αναλυτικότερα, ο πράκτορας με την μικρότερη ταυτότητα (A1) κινείται κατά την θετική κατεύθυνση με σκοπό να συναντήσει τον επόμενο πράκτορα (A2). Ο A2 εφόσον δεν γνωρίζει πόσο απέχει ο A1 απ' αυτόν, θα τον περιμένει σχεδόν n χρονικές μονάδες (τον μέγιστο δηλαδή χρόνο που θα χρειαζόταν ο A1 για να τον συναντήσει). Αντίστοιχα ο A3 θα περιμένει τους A1 και A2 για σχεδόν 4n.

Πιο συγκεκριμένα ο αλγόριθμος λειτουργεί με την εξής λογική:

Ο A1 κινείται κατά την θετική κατεύθυνση με σκοπό να συναντήσει τον A2, για τον οποίο γνωρίζει ότι είναι ο επόμενος πράκτορας κατ αυτή τη φορά.



Σχήμα 3.13

Οι πράκτορες A2 και ο A3 περιμένουν τον μέγιστο χρόνο που θα χρειαζόταν ένας πράκτορας για να τους συναντήσει. Αν ο A1 χαθεί πριν συναντήσει τον A2, ο A2 θα δει ότι ο χρόνος που χρειαζόταν πέρασε και κανένας δεν τον συνάντησε, έτσι θα κινηθεί προς τον A3 για να εξερευνήσουν μαζί τον δακτύλιο. Στην περίπτωση όπου ο A1 και ο A2 συναντηθούν εξερευνούν μαζί τον δακτύλιο. Σε αυτή την περίπτωση ανάλογα με την θέση της μαύρης τρύπας, είτε οι δύο πράκτορες συναντούν τον A3 κατά την διάρκεια της εξερεύνησης και στη συνέχεια συνεχίζουν όλοι μαζί τη εξερεύνηση (όταν η μαύρη τρύπα βρίσκεται στο d3) είτε χάνετε ο A1 και ο A2 κινείται αντίθετα για να ενημερώσει τον A3 για την θέση της μαύρης τρύπας που ανακάλυψαν (όταν η μαύρη τρύπα βρίσκεται στο d2).

Αλγόριθμος Three_agents_Without_Map (all_agents_should_know):

```
//The procedure takes as input:
// a flag if all agents must know or not the black hole position
//The following commands means:
//move -> move one step
//wait -> wait one time unit
```

Let *Agent1* be the agent with the smallest ID. Then *Agent2* will be the agent with the biggest ID. So *Agent3* will be the last agent.

Let **Dir** be the direction that agent1 should follow to reach Agent2 without passing through Agent 3.

case 1) you are Agent1:
repeat
move towards Dir;
until (you see an agent)

case 2) you are Agent2:
repeat
wait;
until (you see another agent) **OR** (n-3 time units passed)
`//n-3 is the maximum time may need A1 to meet A2`
`//n-3: (A)-(C)-(B)-(MT) or (A)-(C)-(MT)-(B)`

case 3) you are Agent3:
repeat
wait;
until (you see an agent) **OR** (4*n-11 time units passed)
`//-(MT)-(A3)-(A1)-(A2)-(...)-`

endcase

case 1) an agent told you the black hole position:
`/* agent2 moved at the negative direction and met agent3 */`
BH at that node;

case 2) (you are agent3) **AND** (you see another agent):
`//agent must synchronized with one agent or two`
wait;
case 1) agent2 returned alone: `//case were BH is at d1`
move towards Dir;
execute Procedure
Move_Together_noMap(Dir)(all_agents_should_know);
case 2) more than one agent came: `//case were BH is at d3`
execute Procedure
Move_Together_noMap(Dir)(all_agents_should_know);
case 3) agent2 didn't return:
`//case were BH is at d3 one node after A3`

BH at the next node towards Dir;

endcase

case 3) you see another agent: `//agent1 meet agent2`

execute Procedure Move_Together_noMap(Dir)(all_agents_should_know);

case 4) (you are Agent2) AND (no agent showed up):

repeat

move towards Dir;

until (you see an agent)

execute Procedure Move_Together_noMap(Dir)(all_agents_should_know);

endcase

Procedure Move_Together_noMap (Dir) (all_agents_should_know)

```
//The procedure takes as input:
// i) the direction that agents must follow to explore
// ii) a flag if all agents must know or not the black hole position. By default is true.
```

repeat

execute Procedure Cautious_Walk(Dir);

until (black hole found)

if (you haven't met agent3) **AND** (all agents should know) **then**

`//agent3 don't know the BH location`

move at the opposite direction of Dir to tell the BH location to the other agent;

end

Απόδειξη ορθότητας:

Λήμμα 3.13

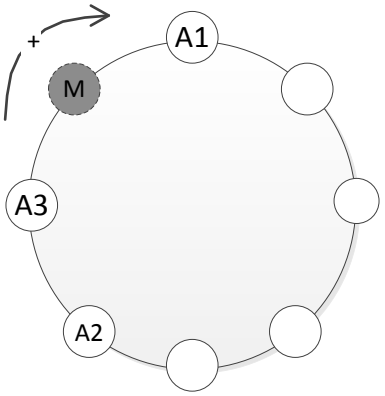
Τρέχοντας τον παραπάνω αλγόριθμο, τουλάχιστον δύο πράκτορες συναντιούνται και ανακαλύπτουν μία μαύρη τρύπα.

Απόδειξη

Υπάρχουν τρείς περιπτώσεις για τις πιθανές θέσεις της μαύρης τρύπας.

Περίπτωση 1: Η μαύρη τρύπα να βρίσκεται κάπου στο τμήμα μήκους d1. Σε αυτή τη περίπτωση ο Agent1 κινούμενος κατά τη θετική κατεύθυνση χάνεται πριν

συναντήσει τον Agent2. Ο Agent2 μετά από n-3 χρονικές μονάδες μπορεί να συμπεράνει ότι ο Agent1 χάθηκε στην μαύρη τρύπα, διότι αν ο Agent1 μπορούσε να συναντήσει τον Agent2, ο Agent1 αποκλείεται να συναντούσε πρώτα τον Agent3 (αφού ο Agent2 είναι ο επόμενος κατά τη θετική κατεύθυνση) και αποκλείεται ο Agent1 να χανόταν σε μαύρη τρύπα (αφού τότε δεν θα μπορούσε να τον συναντήσει). Συνολικά λοιπών θα χρειαζόταν το πολύ n-3 χρονικές μονάδες για να τον συναντήσει, όπως φαίνεται και στο σχήμα που ακολουθεί, το οποίο περιγράφει την μεγαλύτερη δυνατή απόσταση μεταξύ των πρακτόρων Agent1 και Agent2. Στη συνέχεια ο Agent2 θα κινηθεί κατά τη θετική κατεύθυνση για να συναντήσει τον Agent3. Όταν οι δύο πράκτορες συναντηθούν, θα εξερευνήσουν τον δακτύλιο καλώντας τη διαδικασία Move_Together.



Σχήμα 3.14

Περίπτωση 2: Η μαύρη τρύπα να βρίσκεται κάπου στο τμήμα μήκους d2. Σε αυτή τη περίπτωση ο Agent1 κινείται κατά τη θετική κατεύθυνση και συναντά τον Agent2. Οι δύο πράκτορες καλούν την διαδικασία Move_Together και πριν συναντήσουν τον Agent3 ανακαλύπτουν την μαύρη τρύπα. Έτσι ο Agent2 που έχει επιζήσει από την εξερεύνηση, γνωρίζοντας την θέση της μαύρης τρύπας, θα κινηθεί κατά την αντίθετη κατεύθυνση απ' ότι προηγουμένως για να ενημερώσει τον Agent3 για την θέση της μαύρης τρύπας.

Περίπτωση 3: Η μαύρη τρύπα να βρίσκεται κάπου στο τμήμα μήκους d3. Σε αυτή τη περίπτωση ο Agent1 κινείται κατά την φορά των δεικτών του ρολογιού και συναντά τον Agent2. Οι δύο πράκτορες καλούν την διαδικασία Move_Together και ξεκινούν την εξερεύνηση του δακτυλίου. Πριν οι δύο πράκτορες ανακαλύψουν την μαύρη τρύπα, θα συναντήσουν τον Agent3 και όλοι μαζί θα συνεχίσουν την εξερεύνηση και θα ανακαλύψουν την θέση της μαύρης τρύπας.

□

Πρόταση

Η διαδικασία Move_Together εξασφαλίζει την ανακάλυψη της μαύρης τρύπας καθώς χρησιμοποιεί την διαδικασία Cautious_Walk. Επίσης μετά το πέρας της διαδικασίας αυτής, οι 'επιζήσαντες' πράκτορες θα γνωρίζουν τη θέση της μαύρης τρύπας, καθώς στην περίπτωση όπου κάποιος πράκτορας δεν γνωρίζει τη θέση της οι πράκτορες που ανακάλυψαν την μαύρη τρύπα κινούνται με ασφάλεια αντίθετα από την κατεύθυνση της ανακάλυψης ώστε να ενημερώσουν και τον τελευταίο πράκτορα (αφού η μαύρη τρύπα θα βρίσκεται σίγουρα μετά από τον τελευταίο πράκτορα προς αυτή την κατεύθυνση).

Γενικό Λήμμα 3.14

Εκτελώντας τον παραπάνω αλγόριθμο οι πράκτορες μπορούν να ανακαλύψουν την μαύρη τρύπα και όλοι οι 'επιζώντες' πράκτορες να γνωρίζουν τη θέση της.

Απόδειξη

Με βάση το παραπάνω λήμμα και την παραπάνω πρόταση δύο πράκτορες καταφέρνουν πάντα να συναντηθούν και να καλέσουν την διαδικασία Move_Together η οποία όπως αποδείχτηκε μπορεί να ανακαλύψει την μαύρη τρύπα και να ενημερώσει όλους τους ζωντανούς πράκτορες.

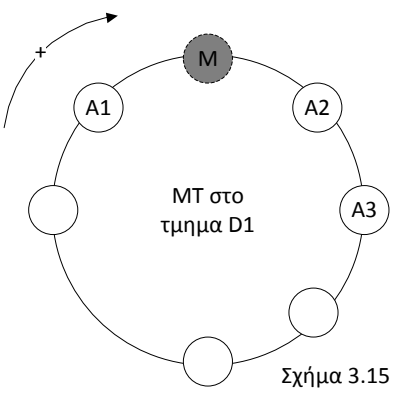
□

<u>Πολυπλοκότητα:</u>

<u>Σημείωση:</u> Ο adversary επιλέγει τις θέσεις των πρακτόρων και της μαύρης τρύπας έτσι ώστε οι πράκτορες να κάνουν όσο το δυνατόν περισσότερα βήματα, είτε εξερευνώντας το μεγαλύτερο δυνατό τμήμα με την χρονοβόρα διαδικασία Move Together η οποία χρειάζεται 3 βήματα για κάθε κόμβο είτε επηρεάζοντας την διαδρομή που πρέπει να ακολουθήσουν για να συναντηθούν.

- **$1^{η}$ περίπτωση:** Όταν η μαύρη τρύπα βρίσκεται **στο τμήμα d1**, τότε ο τύπος της πολυπλοκότητα διαμορφώνεται ως εξής: $\boldsymbol{n-3+1+3(n-3)+2}$ χρονικές μονάδες.

Κατά τη χειρότερη περίπτωση, η διάταξη των πρακτόρων θα είναι όμοια με αυτή του Σχήματος 3.15. Σε αυτή την περίπτωση απαιτούνται $\boldsymbol{n-3}$ χρονικές μονάδες ώστε ο A2 να καταλάβει ότι ο A1 χάθηκε κάπου κατά την αρνητική κατεύθυνση ως προς αυτόν. Στη συνέχεια ο A2 χρειάζεται $\boldsymbol{1}$ χρονική μονάδα ώστε να κινηθεί κατά τη θετική κατεύθυνση και να συναντήσει τον A3. Τέλος $\boldsymbol{3(n-3)+2}$



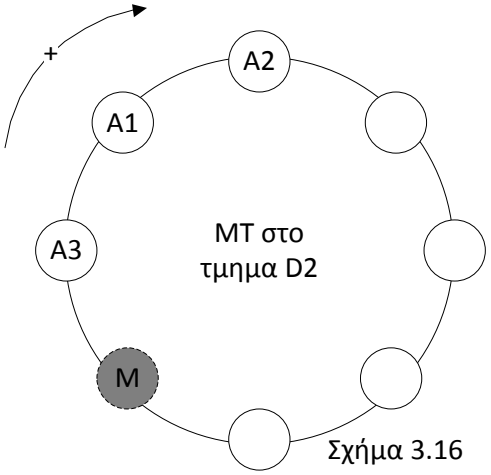
Σχήμα 3.15

χρειάζονται ώστε οι πράκτορες A2 και A3 να εξερευνήσουν τον δακτύλιο και να ανακαλύψουν τη θέση της μαύρης τρύπας.

Μετά από τις κατάλληλες πράξεις η σχέση θα είναι της μορφής: $\boldsymbol{4n-9}$.

- **2η περίπτωση:** Όταν η μαύρη τρύπα βρίσκεται **στο τμήμα d2**, τότε ο τύπος της πολυπλοκότητα διαμορφώνεται ως εξής: $\boldsymbol{1+3(n-4)+2+n-2}$ χρονικές μονάδες.

Σε αυτή την περίπτωση η χειρότερη διάταξη των πρακτόρων ως προς την πολυπλοκότητα θα ήταν η διάταξη του σχήματος .2, όπου θα χρειαζόταν $\boldsymbol{1}$ χρονική μονάδα ώστε ο A1 να επισκεφτεί τον A2 και $\boldsymbol{3(n-4)+2}$ ώστε να εξερευνήσουν τον δακτύλιο μέχρι την στιγμή που θα βρουν την μαύρη τρύπα. Τέλος θα χρειαζόταν ο επιπλέον χρόνος των $\boldsymbol{n-2}$ χρονικών μονάδων ώστε ο A2 να κινηθεί κατά την αντίθετη κατεύθυνση από αυτή που ακολουθούσε πριν, για να ενημερώσει τον A3 για την θέση της μαύρης τρύπας.



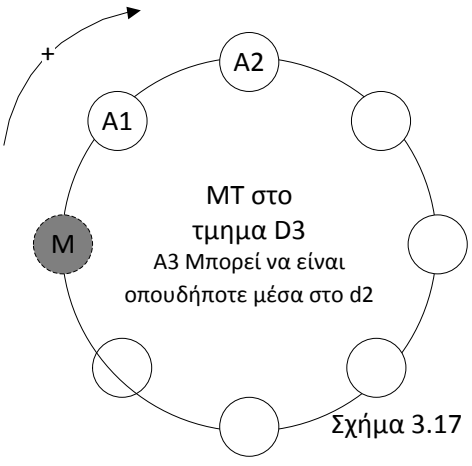
Σχήμα 3.16

Μετά από τις κατάλληλες πράξεις η σχέση θα είναι της μορφής: $\boldsymbol{4n-11}$.

Εναλλακτικά, αν δεν χρειάζεται όλοι οι πράκτορες να γνωρίζουν την θέση της μαύρης τρύπας (all_agents_should_know=false), η σχέση έχει την εξής μορφή: $\boldsymbol{3n-9}$.

- **3η Περίπτωση:** Όταν η μαύρη τρύπα βρίσκεται **στο τμήμα d3**, τότε ο τύπος της πολυπλοκότητα διαμορφώνεται ως εξής: $\boldsymbol{1+3(n-3)+2}$ χρονικές μονάδες.

Σε αυτή την περίπτωση η χειρότερη διάταξη των πρακτόρων ως προς την πολυπλοκότητα θα ήταν η διάταξη του σχήματος .3, όπου θα χρειαζόταν $\boldsymbol{1}$



Σχήμα 3.17

χρονική μονάδα ώστε ο Α1 να επισκεφτεί τον Α2 και $\boldsymbol{3(n-3)+2}$ ώστε να εξερευνήσουν τον δακτύλιο. Κάποια στιγμή οι πράκτορες Α1 και Α2 θα συναντήσουν τον πράκτορα Α3 ο οποίος μπορεί να βρίσκεται οπουδήποτε μετά την αρχική θέση του Α2 και πριν την μαύρη τρύπα. Η θέση του Α3 δεν επηρεάζει την πολυπλοκότητα σε αυτή τη περίπτωση.

Μετά από τις κατάλληλες πράξεις η σχέση θα είναι της μορφής: $\boldsymbol{3n-6}$.

Συνοψίζοντας η πολυπλοκότητα χειρότερης περίπτωσης του αλγορίθμου θα είναι $\boldsymbol{4n-9}$.

3.3.2 Αναζήτηση μίας μαύρης τρύπας από Κ πράκτορες

Η ενότητα αυτή ασχολείται με ένα διαφορετικό αλγόριθμο. Ο αλγόριθμος που ακολουθεί λύνει το πρόβλημα της αναζήτησης ακριβώς μιας μαύρης τρύπας χρησιμοποιώντας k πράκτορες λογισμικού. Βασική προϋπόθεση για τη σωστή λειτουργία του αλγορίθμου, είναι ο αριθμός των κόμβων του δακτυλίου (n) να μην υπερβαίνει τους 2^{k-1}κόμβους. Γίνεται άμεσα αντιληπτό, ότι ο αριθμός των κόμβων είναι αυτός που καθορίζει τον αριθμό των πρακτόρων και αντιστρόφως. Ωστόσο υπάρχουν υποπεριπτώσεις δακτυλίων με περισσότερους από 2^{k-1}κόμβους, όπου οι k πράκτορες αρκούν για την αναζήτηση των μαύρων τρυπών και αυτή η περίπτωση αντιμετωπίζεται από τον αλγόριθμο μας. Για το λόγο αυτό, ο αλγόριθμός μας είναι σχεδιασμένος να αντιλαμβάνεται πότε μία αναζήτηση είναι δυνατή και πότε όχι. Η αδυναμία της αναζήτηση έγκειται στο ότι οι πράκτορες που επέζησαν δεν μπορούν να ολοκληρώσουν την εξερεύνηση και έτσι ο αλγόριθμος τερματίζει.

Το σημαντικότερο πλεονέκτημα του αλγορίθμου που ακολουθεί συγκριτικά με όσους αναφέρθηκαν στις προηγούμενες ενότητες, είναι ότι οι πράκτορες δεν χρειάζονται πληροφορίες που να αφορούν στα στοιχεία του δακτυλίου (χάρτη του δακτυλίου με τις αρχικές θέσεις ή τη διάταξη των πρακτόρων στο δακτύλιο). Οι απαραίτητες πληροφορίες για τον αλγόριθμο είναι το μέγεθος του δακτυλίου (n) και η εξαρχής συμφωνία των πρακτόρων ως προς τον προσανατολισμό του δακτυλίου. Συνεπώς ο αλγόριθμος, είναι πιο ισχυρός καθώς έχει λιγότερες απαιτήσεις.

Από την άλλη όμως πλευρά απαιτείται ένας σημαντικός αριθμός πρακτόρων και αξιόλογος χρόνος εκτέλεσης λόγω της πολυπλοκότητάς του αλγορίθμου.

Συνοπτικά το μοντέλο περιγράφεται ως εξής:

- k πράκτορες, με k>2
- Ακριβώς μία μαύρη τρύπα
- Ανώνυμο δίκτυο με $n \leq 2^{k-1}$ κόμβους

Επιπρόσθετα οι πράκτορες:

- Έχουν συμφωνήσει ως προς τον προσανατολισμό του δακτυλίου
- Έχουν μοναδικές και συγκρίσιμες ταυτότητες (Ids), οι οποίες είναι γνωστές σε όλους
- Είναι συγχρονισμένοι
- Έχουν αρκετή μνήμη, δεν θα χρειαστεί περισσότερο από $O(n)$.

Οι πράκτορες:

- Δεν έχουν χάρτη με τις αρχικές θέσεις των πρακτόρων στο δακτύλιο
- Δεν ξέρουν τη σειρά των πρακτόρων στο δακτύλιο.

Βασική Ιδέα:

Η βασική ιδέα του αλγορίθμου περιγράφεται στην παράγραφο που ακολουθεί. Αναλυτικότερα, αν κάποιος από τους πράκτορες κινηθεί n βήματα κατά τη φορά της κίνησης των δεικτών του ρολογιού, θα συναντήσει τουλάχιστον έναν πράκτορα ή διαφορετικά θα χαθεί σε μία μαύρη τρύπα. Στην περίπτωση όπου ο πράκτορας χαθεί σε μία μαύρη τρύπα οι υπόλοιποι πράκτορες σίγουρα μπορούν να κινηθούν κατά ένα βήμα κατά τη φορά των δεικτών του ρολογιού με ασφάλεια. Καταληκτικά κάθε φορά που επαναλαμβάνεται η διαδικασία στην οποία χάνεται ένας πράκτορας, ο αριθμός των ασφαλών βημάτων που μπορούν να κάνουν οι υπόλοιποι πράκτορες αυξάνει.

Πιο συγκεκριμένα ο αλγόριθμος λειτουργεί με την εξής λογική:

Ένας από τους πράκτορες (έστω ο A1) επιλέγεται για να κινηθεί n βήματα κατά τη φορά κίνησης των δεικτών του ρολογιού. Συνεπώς ο πράκτορας A1 θα χαθεί ή θα συναντήσει κάποιον πράκτορα.

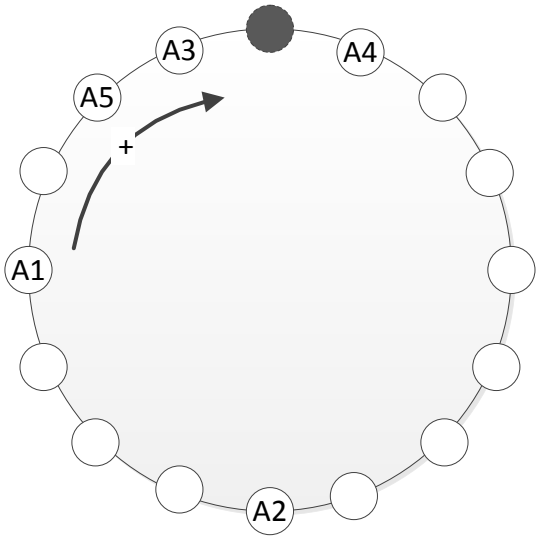
Αν ο πράκτορας χαθεί πριν συναντήσει κάποιον άλλο, οι επιζήσαντες πράκτορες μετά από n χρονικές μονάδες θα μάθουν ότι χάθηκε και θα επιχειρήσουν να συναντήσουν κάποιον άλλο ζωντανό πράκτορα, με τον εξής μηχανισμό: ένας-ένας οι πράκτορες θα κινηθούν ένα βήμα (δηλαδή όσο είναι τα ασφαλή βήματα τη συγκεκριμένη χρονική στιγμή) κατά τη φορά των δεικτών του ρολογιού και θα επιστρέψουν στην αρχική τους θέση.

Στη συνέχεια, με το πέρας του μηχανισμού αν τουλάχιστον δυο πράκτορες δεν έχουν συναντηθεί η παραπάνω διαδικασία επαναλαμβάνεται εξαρχής μέχρι να συναντηθούν τουλάχιστον δυο, ή μέχρι να απομείνουν μόνο δύο πράκτορες. Κάθε φορά που η παραπάνω διαδικασία επαναλαμβάνεται οι πράκτορες αυξάνουν τα ασφαλή βήματα με βάση τη συνάρτηση t=t+2^i, όπου αρχικά t,i =0.

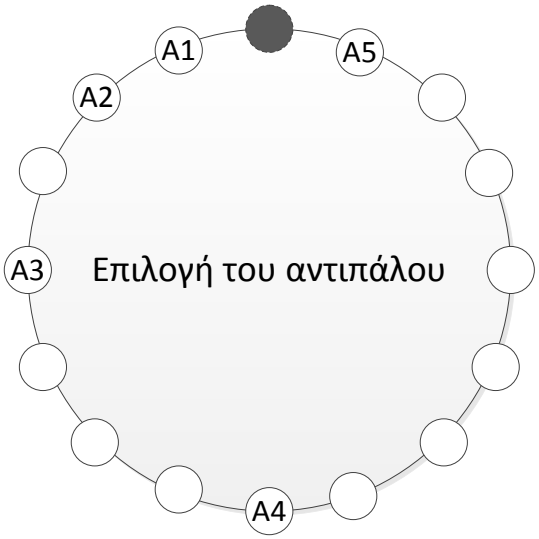
Στην περίπτωση όπου δύο ή περισσότεροι πράκτορες συναντηθούν, ξεκινούν την εξερεύνηση του δακτυλίου.

Η πληροφορία ότι δεν έχει πραγματοποιηθεί συνάντηση στο τέλος της κάθε επανάληψης της παραπάνω διαδικασίας, όπως θα δούμε στα παρακάτω λήμματα (Λήμμα 3.16, Λήμμα 3.18, Λήμμα 3.19), είναι αυτή που επιτρέπει στους πράκτορες να αυξήσουν την ασφαλή περιοχή.

Παράδειγμα 3.8



Σχήμα 3.18



Σχήμα 3.19

Στο παράδειγμα αυτό εξετάζεται η περίπτωση ενός δακτυλίου με 16 κόμβους και 5 πράκτορες. Η διάταξη των πρακτόρων φαίνεται παραπάνω και αποτελεί τη χειρότερη περίπτωση για το συγκεκριμένο αριθμό κόμβων.

Με βάση τον παρακάτω αλγόριθμο **K_agents_only_direction** επιλέγεται ένας πράκτορας για να κινηθεί n βήματα κατά τη φορά της κίνησης των δεικτών του ρολογιού. Έστω ότι ο πράκτορας που επιλέχθηκε είναι ο πράκτορας A1 (Σχήμα 3.16). Στην περίπτωση αυτή ο αντίπαλος θα επιλέξει ως θέσεις των πρακτόρων τις θέσεις που φαίνονται στο σχήμα (Σχήμα 3.18) ώστε ο πράκτορας A1 να χαθεί στη μαύρη τρύπα, για να μην προλάβει να συναντήσει κάποιον πράκτορα. Οι πράκτορες που απομένουν μόλις περάσουν n χρονικές μονάδες μαθαίνουν ότι κάποιος πράκτορας χάθηκε και μπορούν να κινηθούν με ασφάλεια ένα βήμα κατά τη φορά των δεικτών του ρολογιού. Έτσι ένας-ένας οι πράκτορες κινούνται ένα βήμα κατά τη φορά των δεικτών του ρολογιού και επιστρέφουν στην αρχική θέση τους. Με βάση την διάταξη του σχήματος του αντιπάλου, δεν θα πραγματοποιηθεί καμία συνάντηση.

Ο αντίπαλος θα επιλέξει να κινηθεί n βήματα άλλος ένας πράκτορας και να χαθεί στη μαύρη τρύπα πριν συναντήσει κάποιον άλλο. Συνεπώς, ο πράκτορας αυτός θα είναι ο A2 από το σχήμα του αντιπάλου. Μετά από n χρονικές μονάδες οι πράκτορες, αυτή τη φορά θα αυξήσουν τα ασφαλή βήματα από ένα σε τρία.

Η παραπάνω διαδικασία επαναλαμβάνεται μέχρι να συναντηθούν τουλάχιστον δυο πράκτορες ή μέχρι να απομείνουν μόνο δύο πράκτορες ζωντανοί. Κάθε φορά που επαναλαμβάνεται η παραπάνω διαδικασία, χάνεται ένας ακόμη πράκτορας και οι υπόλοιποι πράκτορες αυξάνουν τα ασφαλή βήματα με βάση την συνάρτηση t=t+2^i, όπου αρχικά t,i =0.

Στο συγκεκριμένο παράδειγμα φαίνεται ότι κάποια στιγμή ο πράκτορας A5 θα κάνει 7 βήματα κατά τη φορά των δεικτών του ρολογιού και θα συναντήσει τον A4, ο οποίος θα βρίσκεται στην αρχική του θέση. Έτσι οι δύο πράκτορες θα εξερευνήσουν μαζί τον δακτύλιο και θα εντοπίσουν την μαύρη τρύπα.

Αλγόριθμος: K_agents_only_direction

```
//The following commands means:
//move -> move one step
//wait -> wait one time unit
```

Agents={all IDs};
Short the Agents from the smallest id to the biggest;
clockTime=0; `//the number of time units passed from the beginning`
t=0; i=0;
repeat
ChosenAgent=first from Agents set;
if (you are the ChosenAgent) **then** `//moves n steps cw`
repeat
move clockwise;
until (you have moved n steps) **OR** (you see another agent)
else `//waits n steps`
repeat
wait;
until (n time units passed) **OR** (you see another agent)
clockTime= clockTime + n;
end
case 1) you see another agent
execute Procedure Move_Together_noMap(Dir)(false);
case 2) (n time units passed) **AND** (no agent showed up)
remove the first agent from the Agents set`//check may needed`
t=t+2^I;
repeat
ChosenAgent=next agent from the Agents set; `//check may needed`
if (you are the chosen agent) **then**
repeat
move at clockwise direction;

```
                    until (t time units have passed)OR(you see another
                    agent)
                    move t steps counterclockwise;
                else
                    repeat
                        wait;
                    until (2*t time units have passed) OR (you see another
                    agent)
                end
            until (the ChosenAgent is the biggest Id) OR (you see another agent)
            i++;
until (two agents left) OR (you see another agent)
if (you see another agent) then
    execute Procedure Move_Together_noMap(Dir)( false);
else
    BHS impossible;
end
```

Πολυπλοκότητα:

Η πολυπλοκότητα της χειρότερης περίπτωσης δίνεται από τον παρακάτω προσεγγιστικό τύπο:

$p = n(k-2) + \sum_{i=1}^{k-2} 2(\mathrm{k}-\mathrm{i}) \sum_{j=0}^{k-2} 2^{\mathrm{j}} \;=\; O(n2^{\kappa})$

Και προκύπτει από την εξής διαδικασία:

Όλοι οι πράκτορες εκτός από δύο κινούνται n βήματα κατά τη φορά των δεικτών του ρολογιού ενώ οι υπόλοιποι πράκτορες περιμένουν. Επίσης, (k-i) πράκτορες κινούνται $\sum_{i=0}^{k-2} 2^{\mathrm{i}}$ βήματα κατά τη φορά των δεικτών του ρολογιού ένας ένας και επιστρέφουν στην προηγούμενη θέση τους. Τέλος οι δύο πράκτορες που θα συναντηθούν, για να εξερευνήσουν το δακτύλιο χρειάζονται επιπλέον χρονικό διάστημα μικρότερο από 3n (που αντιστοιχεί στον χρόνο που απαιτεί η διαδικασία Move_Together).

Στο σημείο αυτό αποδεικνύεται η ορθότητα του αλγορίθμου με την παράθεση των παρακάτω λημμάτων.

Απόδειξη ορθότητα:

Λήμμα 3.15
Ένας πράκτορας που κινείται κατά τη φορά των δεικτών του ρολογιού για n βήματα, θα εξαφανιστεί σε μία μαύρη τρύπα ή θα συναντήσει κάποιον πράκτορα.

Απόδειξη

Έστω ένας δακτύλιος με μία μαύρη τρύπα και 2 ή περισσότερους πράκτορες. Αν ένας πράκτορας από αυτούς κινηθεί (έστω Α1) κατά τη φορά των δεικτών του ρολογιού και οι υπόλοιποι πράκτορες περιμένουν ακίνητοι τότε υπάρχουν δύο περιπτώσεις:

1. Μπροστά από τον πράκτορα Α1 κατά τη φορά των δεικτών του ρολογιού να βρίσκεται η μαύρη τρύπα, τότε ο πράκτορας Α1 θα χαθεί σε αυτήν.

2.Μπροστά από τον πράκτορα Α1 κατά την φορά των δεικτών του ρολογιού να βρίσκεται τουλάχιστον ένας πράκτορας πριν από την μαύρη τρύπα, οι δύο πράκτορες θα συναντηθούν.

□

Λήμμα 3.16

Αν ένας πράκτορας χαθεί κινούμενος n βήματα κατά τη φορά των δεικτών του ρολογιού πριν συναντήσει κάποιον πράκτορα, τότε οι υπόλοιποι πράκτορες μπορούν να κινηθούν με ασφάλεια τουλάχιστον 1 βήμα κατά τη φορά των δεικτών του ρολογιού.

Απόδειξη

Έστω ένας δακτύλιος n κόμβων με μία μαύρη τρύπα και 2 ή περισσότερους πράκτορες. Με βάση το λήμμα 3.15 για να χαθεί ένας πράκτορας πριν προλάβει να συναντηθεί με κάποιον άλλο, η μαύρη τρύπα θα πρέπει να βρίσκεται μπροστά του κατά την φορά των δεικτών του ρολογιού. Συνεπώς όλοι οι πράκτορες θα βρίσκονται τουλάχιστον μία ακμή μακριά του κατά την αντίθετη φορά, με αποτέλεσμα οποιοσδήποτε πράκτορας να χρειάζεται να διανύσει τουλάχιστον δύο ακμές πριν πέσει σε μαύρη τρύπα. Άρα οι πράκτορες μπορούν να κινηθούν τουλάχιστον 1 βήμα με ασφάλεια κατά τη φορά των δεικτών του ρολογιού.

□

Λήμμα 3.17

Αν τουλάχιστον ένας πράκτορας απέχει όσο τα ασφαλή βήματα ή λιγότερο από κάποιον άλλο 'ζωντανό' πράκτορα, τότε σίγουρα οι πράκτορες αυτοί θα συναντηθούν.

Απόδειξη

Έστω ένας δακτύλιος με μία μαύρη τρύπα και δύο ή περισσότερους πράκτορες.

Ας υποθέσουμε πως υπάρχει ένα ζευγάρι πρακτόρων του οποίου οι πράκτορες απέχουν όσο τα ασφαλή βήματα ή λιγότερο και γνωρίζουν τον αριθμό των ασφαλών βημάτων που μπορούν να κάνουν κατά τη δεξιόστροφη φορά.

Οι πράκτορες ακολουθούν την εξής τακτική: Επιλέγεται ένα από τους επιζήσατε πράκτορες, κινείται κατά τη φορά των δεικτών του ρολογιού όσα βήματα είναι η ασφαλής περιοχή και επιστρέφει στην προηγούμενη θέση του. Οι υπόλοιποι

πράκτορες περιμένουν ακίνητοι. Η διαδικασία επαναλαμβάνεται μέχρι όλοι οι πράκτορες να κινηθούν.

Συνεπώς κάποια χρονική στιγμή θα επιλεχθεί να κινηθεί ο πράκτορας που απέχει όσο τα ασφαλή βήματα ή λιγότερο από τον επόμενο πράκτορα κατά τη φορά των δεικτών του ρολογιού. Ο πράκτορας αυτός θα συναντήσει τον άλλο πράκτορα ο οποίος θα περιμένει ακίνητος.

□

Λήμμα 3.18

Αν ένας πράκτορας χαθεί κινούμενος n βήματα κατά τη φορά των δεικτών του ρολογιού πριν συναντήσει κάποιον πράκτορα και στη συνέχεια χαθεί ένας ακόμη χωρίς να έχει προηγηθεί συνάντηση, τότε οι υπόλοιποι πράκτορες μπορούν να κινηθούν με ασφάλεια τουλάχιστον 3 βήμα κατά τη φορά των δεικτών του ρολογιού.

Απόδειξη

Έστω ένας δακτύλιος n κόμβων με μία μαύρη τρύπα και 3 ή περισσότερους πράκτορες. Από λήμμα 3.16 συμπεραίνεται ότι αν χαθεί ο πρώτος τα ασφαλή βήματα που μπορούν να κάνουν οι πράκτορες είναι 1. Από λήμμα 3.17 συμπεραίνεται ότι αν οι πράκτορες απείχαν όσο τα ασφαλή βήματα ή λιγότερα θα είχαν συναντηθεί. Συνεπώς αν χαθεί ένας ακόμη πράκτορας κινούμενος n βήματα κατά τη φορά των δεικτών του ρολογιού οι πράκτορες πριν χαθούν θα χρειαζόταν:

- τουλάχιστον ένα βήμα για να περάσουν από την αρχική θέση του δεύτερου πράκτορα, καθώς όλοι οι πράκτορες θα βρισκόταν αντίθετα από τη φορά του ρολογιού από τη θέση του δεύτερου πράκτορα.
- τουλάχιστον ένα ακόμη βήμα ώστε να διανύσουν την προηγούμενη ασφαλή περιοχή που είχε μπροστά του ο δεύτερος πράκτορας
- Τέλος ένα βήμα ακόμη ώστε να διανύσουν την αρχική θέση του πρώτου πράκτορα που χάθηκε.

Συνολικά λοιπόν οι πράκτορες μπορούν να κινηθούν τρία βήματα με ασφάλεια.

□

Λήμμα 3.19

Μετά την εξαφάνιση του λ-οστού πράκτορα οι κ-λ υπόλοιποι πράκτορες μπορούν να κινηθούν με ασφάλεια για $f(k,\lambda,n) = 2^0 + 2^1 + \cdots 2^{\lambda-1}$ *βήματα.*

Απόδειξη

Με βάση το λήμμα 3.16 η παραπάνω εξίσωση ισχύει για ένα πράκτορα, και τα ασφαλή βήματα υπολογίζονται σε $2^0 = 1$.

Από το λήμμα 3.18 η παραπάνω εξίσωση ισχύει για και για δύο πράκτορα, και τα ασφαλή βήματα υπολογίζονται σε $2^0 + 2^1 = 3$.

Έστω ότι ισχύει για λ πράκτορες. Αν χαθεί ένας ακόμη πράκτορας πριν προλάβει να συναντηθεί με κάποιον άλλο, τότε οι επιζήσαντες πράκτορες πριν χαθούν σε μία μαύρη τρύπα κινούμενοι κατά τη φορά των δεικτών του ρολογιού θα πρέπει να κινηθούν 2^{λ} βήματα για να φτάσουν στην αρχική θέση του λ-οστού πράκτορα, αλλιώς με βάση το λήμμα 3.17 θα είχαν συναντηθεί και στη συνέχεια $2^0 + 2^1 + \cdots 2^{\lambda-1}$ βήματα τα οποία είναι ίσα με τα ασφαλή βήματα που μπορούσε να κάνει ο λ-οστός πράκτορας.

Άρα η σχέση ισχύει για λ+1, συνεπώς ισχύει για κάθε λ, όπου κ-λ<=2 διότι αν μείνουν λιγότεροι πράκτορες από δύο η αναζήτηση θα ήταν αδύνατη.

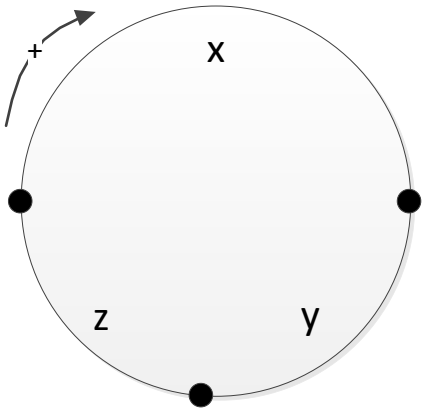
□

3.3.3 Απόδειξη αδυναμίας

Το παρακάτω λήμμα, αποδεικνύει ότι οι αλγόριθμοι που χρησιμοποιούν τρεις πράκτορες είναι βέλτιστοι ως προς τον αριθμό των πρακτόρων που χρησιμοποιούν.

Λήμμα 3.20
Σε ένα δακτύλιο με μία μαύρη τρύπα, η αναζήτηση της μαύρης τρύπας είναι αδύνατη όταν: υπάρχουν 3 πράκτορες που ξεκινούν από διαφορετικές θέσεις, οι πράκτορες έχουν μοναδικές και συγκρίσιμες ταυτότητες, έχουν συμφωνήσει ως προς τον προσανατολισμό του δακτυλίου, γνωρίζουν το μέγεθος του δακτυλίου και αρχικά απέχουν σημαντικά ο ένας από τον άλλο.

Απόδειξη:



Σχήμα 3.20

Θεωρείται ένας δακτύλιος στον όποιο οι πράκτορες έχουν αποστάσεις x,y,z για τις οποίες ισχύει x,y,z>6. Έστω ότι οι πράκτορες έχουν τη διάταξη του παραπάνω σχήματος και επίσης ισχύει ότι:

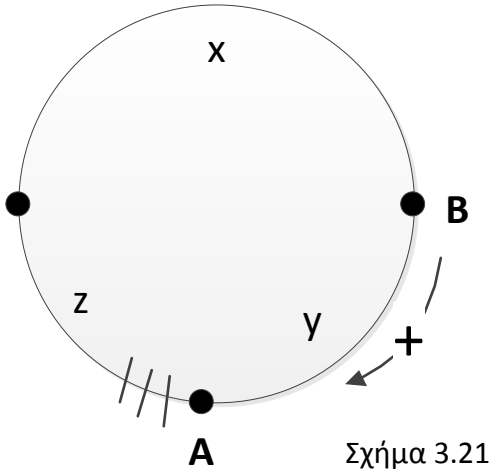
$x > 2(y + 6)$ (1)

$x > 2(z + 6)$ (2)

Είναι φανερό ότι αν ένας αλγόριθμος δεν κινήσει κανέναν πράκτορα δεν μπορεί να είναι ένας σωστός αλγόριθμος. Μάλιστα οποιοσδήποτε αλγόριθμος θα πρέπει να κινήσει τουλάχιστον έναν πράκτορα, έστω Α, έτσι ώστε ο Α να επισκεφθεί τρεις μη ασφαλείς κόμβους που βρίσκονται προς την ίδια κατεύθυνση ως προς τον Α (είτε κατά την φορά των δεικτών του ρολογιού είτε αντίθετα) διότι σε αυτή την περίπτωση υπάρχουν τουλάχιστον δύο κόμβοι που δεν τους επισκέπτεται κανένας πράκτορας. Έτσι λοιπόν κανείς απ' αυτούς τους αλγόριθμους δεν μπορεί να ανακαλύψει την μαύρη τρύπα που ο ανταγωνιστής (adversary) τοποθετεί σε έναν από αυτούς τους δύο κόμβους.

Έστω χωρίς βλάβη της γενικότητας ότι ο Α είναι ο πρώτος από τους πράκτορες που επισκέφτηκε 3 κόμβους προς την ίδια κατεύθυνση (εάν υπάρχουν περισσότεροι από 3 πράκτορες με την ίδια ιδιότητα έστω Α ένας από αυτούς) και ας υποθέσουμε επίσης ότι η κατεύθυνση που επέλεξε να κινηθεί είναι η θετική, όπως αυτή φαίνεται στο σχήμα 3.22. Ο ανταγωνιστής μπορεί να επιλέξει να τοποθετήσει τη μαύρη τρύπα σε έναν από αυτούς τους 3 κόμβους, έτσι ο Α καταστρέφεται. Από τους υπόλοιπους 2 πράκτορες, κάθε σωστός αλγόριθμος θα πρέπει να κουνήσει έναν από αυτούς, (έστω Β ο πράκτορας που κινείται) έτσι ώστε ο Β να επισκεφτεί $\lfloor\frac{\chi-6}{2}\rfloor$ κόμβους προς την ίδια κατεύθυνση γιατί αλλιώς οι εναπομείναντες πράκτορες δεν συναντώνται. Συνεπώς οποιοσδήποτε αλγόριθμος δεν μπορεί να συμπεράνει σε ποιον από τους κόμβους χάθηκε ο Α. Έστω χωρίς βλάβη της γενικότητας ότι Β είναι ο πράκτορας που κινείται πρώτος , έτσι ώστε να επισκεφτεί $\lfloor\frac{\chi-6}{2}\rfloor$ κόμβους προς την ίδια κατεύθυνση ως προς τον Β (εάν υπάρχουν περισσότεροι από 3 πράκτορες με την ίδια ιδιότητα έστω Β ένας από αυτούς).

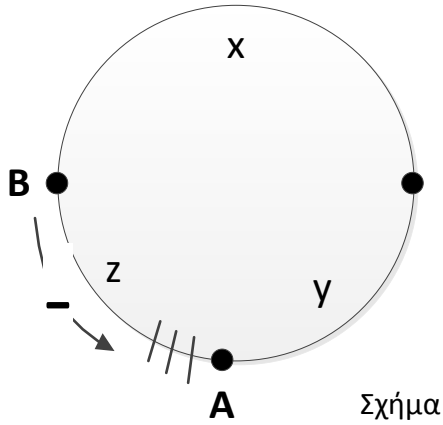
1η περίπτωση: Έστω ότι ο Β κινήθηκε ως προς τη θετική κατεύθυνση, τότε ο ανταγωνιστής επιλέγει τον Β να είναι ο πράκτορας όπως στο παρακάτω σχήμα.



Σχήμα 3.21

Η απόσταση που έχει ο πράκτορας Β από την αρχική θέση του Α είναι μεγαλύτερη ή ίση από y-3 και μικρότερη ή ίση από y+3. Όμως σε αυτή την περίπτωση λόγο της ανίσωσης (1) ο πράκτορας Β χάνεται στην περιοχή που χάθηκε ο Α.

$2^{η}$ περίπτωση: Ο Β κινείται μια απόσταση μεγαλύτερη από $\lfloor\frac{\chi-6}{2}\rfloor$ κατά την αρνητική κατεύθυνση. Τότε ο ανταγωνιστής επιλέγει τη θέση του Β όπως φαίνεται στο παρακάτω σχήμα.



Σχήμα 3.22

Η απόσταση που έχει ο πράκτορας Β από την αρχική θέση του Α αυτή τη φορά είναι μεγαλύτερη ή ίση από z-3 και μικρότερη ή ίση από z+3. Όμως στην περίπτωση αυτή λόγο της ανίσωσης (2) ο πράκτορας Β χάνεται στην περιοχή που χάθηκε ο Α.

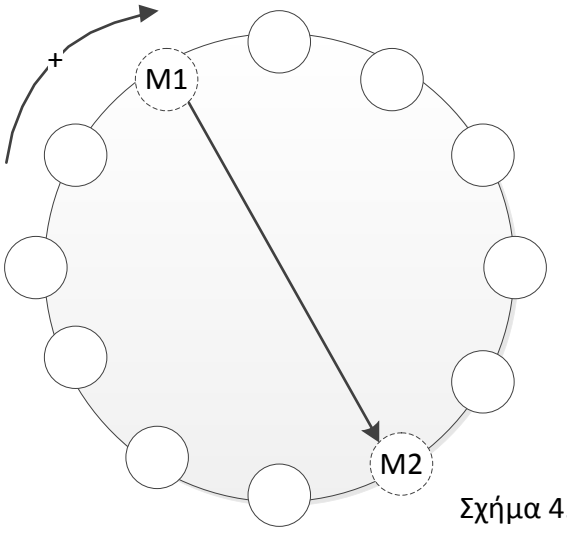
Σε οποιαδήποτε από τις παραπάνω δύο περιπτώσεις, απομένει μόνο ένας πράκτορας με περισσότερες από μία υποψήφιες θέσεις για τη μαύρη τρύπα.

□

Κεφάλαιο 4 - Αλγόριθμοι αναζήτησης δυο μαύρων τρυπών

Το κεφάλαιο αυτό ασχολείται με την περιγραφή νέων αλγορίθμων που αναζητούν ακριβώς δυο μαύρες τρύπες σε έναν δακτύλιο, στη συνέχεια αναλύεται η πολυπλοκότητά τους και παρουσιάζονται οι αποδείξεις της ορθότητάς τους.

Πιο συγκεκριμένα, εξετάζουμε το πρόβλημα της αναζήτησης **ακριβώς δύο** μαύρων τρυπών σε έναν ανώνυμο και συγχρονισμένο δακτύλιο n κόμβων. Η εξερεύνηση πραγματοποιείται από περισσότερους από τέσσερεις πράκτορες **ανάλογα με το πρόβλημα**. Οι πράκτορες έχουν μοναδικές και συγκρίσιμες ταυτότητες (Ids) οι οποίες είναι σημειωμένες στο χάρτη του δικτύου με τις αρχικές θέσεις των πρακτόρων. Οι πράκτορες δεν έχουν συμφωνήσει στον προσανατολισμό του δακτυλίου αλλά μπορούν εύκολα να συμφωνήσουν με βάση την παρατήρηση κάτω από το παράδειγμα 3.5 (στο κεφάλαιο 3). Τέλος θεωρείται ότι οι πράκτορες έχουν αρκετή μνήμη και δεν θα χρειαστεί περισσότερο από $O(n)$.



Σχήμα 4.1

Η βασική δομή του δακτυλίου που θα αναλυθεί στο παρόν κεφάλαιο, αποτελείται από έναν δακτύλιο με n κόμβους και δύο μαύρες τρύπες M1 και M2 όπως φαίνεται στο παραπάνω σχήμα (Σχήμα 4.1). Η ύπαρξη των δύο μαύρων τρυπών ορίζει τα τόξα M1-M2 και M2-M1 κατά τη φορά των δεικτών του ρολογιού.

Όπως γίνεται εύκολα αντιληπτό, ο ελάχιστος αριθμός πρακτόρων για την αναζήτηση των δύο μαύρων τρυπών είναι τουλάχιστον τρείς, αφού τουλάχιστον δύο πράκτορες πρέπει να χαθούν στις δύο μαύρες τρύπες.

Λήμμα 4.1
Είναι αδύνατον δύο πράκτορες να εξερευνήσουν δύο μαύρες τρύπες.

Όπως θα δούμε παρακάτω, πολλές φορές χρειάζονται περισσότεροι από τρεις πράκτορες, εξαιτίας του γεγονότος ότι οι πράκτορες δεν ξεκινούν από την ίδια θέση.

Σε αυτή την ενότητα θα ασχοληθούμε με δύο κατηγορίες προβλημάτων:

Κατηγορία 1: Ο σκοπός είναι μετά από πεπερασμένο χρόνο, όλοι οι πράκτορες να σταματήσουν και ένας τουλάχιστον από αυτούς να γνωρίζει τις θέσεις των δύο μαύρων τρυπών.

Κατηγορία 2: Ο σκοπός είναι μετά από πεπερασμένο χρόνο όλοι οι πράκτορες που επέζησαν να σταματήσουν και να γνωρίζουν συλλογικά τις θέσεις των δύο μαύρων τρυπών.

Κατηγορία 3: Ο σκοπός είναι μετά από πεπερασμένο χρόνο, όλοι οι πράκτορες να σταματήσουν και κάθε πράκτορας να γνωρίζει τις θέσεις των δύο μαύρων τρυπών.

Αντίστοιχα στο κείμενο που ακολουθεί θα χρησιμοποιούμε τον όρο αλγόριθμος κατηγορίας i για τον αλγόριθμο που λύνει το πρόβλημα της κατηγορίας i. Στην παρούσα πτυχιακή εργασία δεν εξετάζονται αλγόριθμοι κατηγορίας 3.

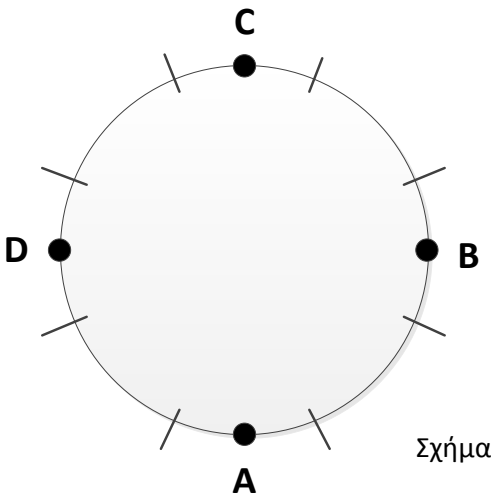
Προφανώς ένας αλγόριθμος ο οποίος λύνει το πρόβλημα κατηγορίας 1, λύνει και το πρόβλημα κατηγορίας 2, αλλά όχι το ανάποδο.

Το πρόβλημα που μελετούμε είναι η σχεδίαση ενός αλγορίθμου ο οποίος ανακαλύπτει τις μαύρες τρύπες μετά από ντετερμινιστικό χρόνο, για οποιεσδήποτε αρχικές θέσεις των πρακτόρων και των μαύρων τρυπών. Μάλιστα ενδιαφερόμαστε για τη σχεδίαση ενός βέλτιστου αλγορίθμου ως προς τον αριθμό των πρακτόρων.

Λήμμα 4.2

Δεν υπάρχει αλγόριθμος κατηγορίας 2 ή 1 που να λύνει το πρόβλημα με τέσσερις πράκτορες.

Απόδειξη:



Σχήμα 4.2

Έστω η διάταξη του παραπάνω σχήματος 4.2. Ας υποθέσουμε ότι υπάρχει ένας αλγόριθμος κατηγορίας 2, που λύνει το πρόβλημα για αυτή τη διάταξη των πρακτόρων (Σχήμα 4.2). Ξεκάθαρα ο αλγόριθμος πρέπει να κινεί κάποιον πράκτορα. Αν υποθέσουμε ότι αυτός ο αλγόριθμος δεν οδηγεί ποτέ έναν πράκτορα στο να επισκεφτεί δύο ανεξερεύνητους κόμβους, τότε είναι ξεκάθαρο ότι υπάρχουν τουλάχιστον δύο κόμβοι τους οποίους δεν θα επισκεφθεί κανείς πράκτορας. Άρα ένας τέτοιος αλγόριθμος δεν μπορεί να είναι σωστός. Συνεπώς σε έναν ορθό αλγόριθμο ένας πράκτορας πρέπει να επισκεφθεί τουλάχιστον δύο ανεξερεύνητους κόμβους. Έστω A ο πρώτος πράκτορας που επισκέπτεται δυο ανεξερεύνητους κόμβους u, v. Ο ανταγωνιστής μπορεί να τοποθετήσει τη μαύρη τρύπα σε έναν από αυτούς τους δύο κόμβους.

1η περίπτωση: Ένας από τους πράκτορες B,D επισκέπτεται κάποιον από τους κόμβους u, v. Τότε ο ανταγωνιστής επιλέγει τη θέση της μαύρης τρύπας έτσι ώστε και ο νέος πράκτορας (έστω B) να χαθεί. Από τους εναπομείναντες πράκτορες C, D πάλι τουλάχιστον ένας (έστω C) πρέπει να επισκεφθεί δύο ανεξερεύνητους κόμβους. Ο ανταγωνιστής μπορεί να τοποθετήσει τη δεύτερη μαύρη τρύπα σε έναν από αυτούς τους δύο κόμβους, σκοτώνοντας το C. Τότε ο πράκτορας D δεν μπορεί να ανακαλύψει τις δύο μαύρες τρύπες και να επιζήσει.

2η περίπτωση: Κανείς από τους πράκτορες B, D δεν επισκέπτεται τους κόμβους u, v. Και σε αυτή την περίπτωση ένας από τους B, C, D πρέπει να επισκεφθεί δύο ανεξερεύνητους κόμβους (διαφορετικούς από τους u,v). Έστω K εκείνος ο πράκτορας από τους B, C, D που επισκέπτεται δύο ανεξερεύνητους κόμβους u', v'. Τότε ο ανταγωνιστής τοποθετεί τη δεύτερη μαύρη τρύπα σε έναν από τους δύο κόμβους u', v'. Οι εναπομείναντες δυο πράκτορες ακόμη και αν καταφέρουν να συναντηθούν, δεν μπορούν να ανακαλύψουν τις ακριβείς θέσεις των δύο μαύρων τρυπών. Για να ανακαλύψουν τη θέση της μίας μαύρης τρύπας πρέπει ο ένας από τους δύο πράκτορες να χαθεί και συνεπώς ο τελευταίος πράκτορας δεν μπορεί να είναι σίγουρος για τη θέση της άλλης μαύρης τρύπας. Άρα δε μπορεί να υπάρχει αλγόριθμος κατηγορίας 2 (και κατά συνέπεια ούτε κατηγορίας 1) που λύνει το πρόβλημα.

□

Συμπερασματικά καταλήγουμε στο γεγονός ότι ένας σωστός αλγόριθμος κατηγορίας 2 ή κατηγορίας 1 θα χρησιμοποιεί τουλάχιστον πέντε πράκτορες για την αναζήτηση δύο μαύρων τρυπών σε ένα δακτύλιο. Όπως θα δούμε αργότερα υπάρχει αλγόριθμος κατηγορίας 2 που λύνει το πρόβλημα χρησιμοποιώντας πέντε πράκτορες. Πριν από αυτόν όμως, θα δώσουμε έναν αλγόριθμο κατηγορίας 1 που χρησιμοποιεί έξι πράκτορες.

4.1 Αλγόριθμος κατηγορίας 1 (BH2_MinID with Move_Together_ver1)

Ο αλγόριθμος που ακολουθεί είναι κατηγορίας 1 και λύνει το πρόβλημα της αναζήτησης δύο μαύρων τρυπών όταν υπάρχουν έξι πράκτορες στο δακτύλιο.

Συνοπτικά το μοντέλο περιγράφεται ως εξής:

- Έξι πράκτορες
- Ακριβώς δύο μαύρες τρύπες
- Τοπολογία: Δακτυλίου (Ring)
- Ανώνυμο δίκτυο με n κόμβους

Επιπρόσθετα οι πράκτορες:

- Κάθε πράκτορας έχει μοναδική ταυτότητα (Id) μέσα από ένα ολικά διατεταγμένο σύνολο
- Έχουν χάρτη του δικτύου με σημειωμένες τις αρχικές θέσεις και τις ταυτότητες των πρακτόρων
- Είναι συγχρονισμένοι
- Έχουν αρκετή μνήμη, δεν θα χρειαστεί περισσότερο από $O(n)$
- × **Δεν** έχουν συμφωνήσει στον προσανατολισμό του δακτυλίου

Βασική Ιδέα:

Η βασική ιδέα του αλγορίθμου BH2_MinID παρουσιάζεται στην παράγραφο που ακολουθεί. Αναλυτικότερα, ο πράκτορας με το μικρότερο ID (A1) κινείται κατά τη θετική κατεύθυνση, ενώ οι υπόλοιποι πράκτορες περιμένουν. Σκοπός είναι ο πράκτορας που κινείται πρώτος να συναντήσει κάποιον ακόμα πράκτορα και μαζί να εξερευνήσουν το δακτύλιο. Αν ο πράκτορας A1 χαθεί, ο επόμενος πράκτορας (A2) θα μάθει ότι χάθηκε (αφού δεν εμφανίστηκε μετά από συγκεκριμένο χρόνο) και θα κινηθεί και αυτός προς τη θετική κατεύθυνση ώστε να συναντήσει τον επόμενο πράκτορα.

Αν δύο ή περισσότεροι πράκτορες συναντηθούν, εξερευνούν το δακτύλιο μέχρι να ανακαλύψουν την πρώτη μαύρη τρύπα. Κάθε φορά που οι πράκτορες συναντούν έναν νέο πράκτορα, ο νέος πράκτορας συμμετέχει και αυτός στην αναζήτηση.

Όταν οι πράκτορες που εξερευνούν συναντήσουν την πρώτη μαύρη τρύπα οι επιζήσαντες πράκτορες θα εξερευνήσουν το δακτύλιο κατά την αντίθετη κατεύθυνση. Τέλος όσοι πράκτορες περιμένουν ακίνητοι και δεν τους επισκέπτεται άλλος πράκτορας (μετά από συγκεκριμένο χρόνο) κινούνται κατά τη θετική φορά.

Σημειώνεται ότι αρχικά ο κάθε πράκτορας Α θεωρεί σαν ασφαλή περιοχή S_A το σύνολο των κόμβων από τους οποίους ξεκινούν οι πράκτορες. Όταν δύο πράκτορες Α, Β συναντιούνται τη χρονική στιγμή t, ισχύει:

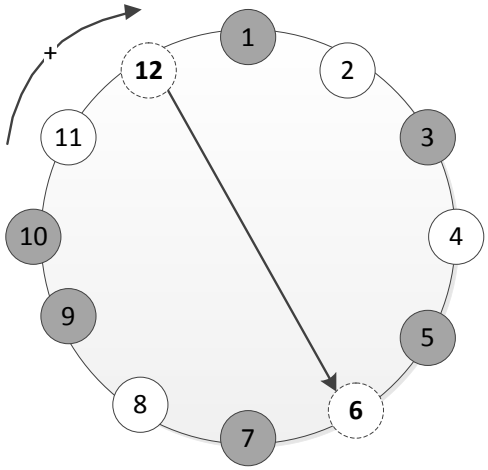
$S_A^t = S_B^t = S_A^{t-1} \cup S_B^{t-1}$

$S_i^0 = S$, όπου S το σύνολο των αρχικών θέσεων.

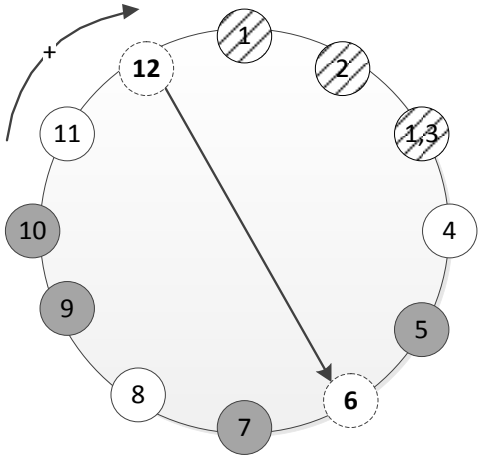
Υπενθυμίζουμε ότι, οι πράκτορες δεν μπορούν να συναντηθούν σε ανεξερεύνητο κόμβο.

Παράδειγμα 4.1: Έστω ο παρακάτω δακτύλιος (σχήμα 4.3), με έξι πράκτορες στους σκιασμένους κόμβους και δύο μαύρες τρύπες στους κόμβους με τις διακεκομμένες γραμμές (6, 12). Για χάρη της ανάλυσης, στο παράδειγμα του σχήματος 4.3 οι κόμβοι είναι αριθμημένοι (στην πραγματικότητα ο δακτύλιος είναι ανώνυμος).

Με την έναρξη του αλγορίθμου BH2_MinID, οι πράκτορες έχοντας το χάρτη του δικτύου με τις αρχικές θέσεις των πρακτόρων, «σημειώνουν» τους κόμβους που γνωρίζουν ότι είναι ασφαλείς. Στην περίπτωση του παραδείγματος, καταγράφουν τις αρχικές θέσεις των πρακτόρων, δηλαδή τους κόμβους 1, 3, 5, 7, 9, 10 καθώς αποκλείεται να αποτελούν μαύρη τρύπα. Στη συνέχεια κάθε πράκτορας ξεχωριστά «σημειώνει» τους κόμβους που έχει επισκεφτεί. Ας υποθέσουμε λοιπόν πως μετά από κάποιο χρόνο t ο πράκτορας από τον κόμβο 1 συναντά τον πράκτορα στον κόμβο 3 έχοντας εξερευνήσει τους γραμμοσκιασμένους κόμβους (1, 2, 3) του σχήματος 4.4, κινούμενος κατά τη θετική φορά. Την ίδια χρονική στιγμή ο πράκτορας 3 μαθαίνει ότι ο κόμβος 2 είναι ασφαλής.



Σχήμα 4.3



Σχήμα 4.4

Επισημαίνεται στον αλγόριθμο μας ότι όταν δύο ή περισσότεροι πράκτορες εξερευνούν μαζί, ελέγχουν αν ο επόμενος κόμβος αποτελεί αρχική θέση και κινούνται σε αυτόν με ασφάλεια, χωρίς να τον εξερευνήσουν. Έτσι όταν οι πράκτορες που εξερευνούσαν, συναντήσουν έναν νέο πράκτορα στην αρχική του θέση, ξεκινούν όλοι μαζί την εξερεύνηση των επόμενων κόμβων την ίδια χρονική στιγμή (συγχρονισμένα). Συνεπώς στον αλγόριθμό μας δεν χρειάζεται οι πράκτορες να ελέγχουν τις καταστάσεις της συνάρτησης μετάβασης των άλλων πρακτόρων που συμμετέχουν στην εξερεύνηση για τον συγχρονισμό τους.

Αλγόριθμος: BH2_MinID

```
//The following commands means:
//move -> move one step
//wait -> wait one time unit
//alone agent -> agent is alone at his node
```

```
/*The following variable is a flag representing whether the agents have
already found a BH or not*/
```
One_BH_found=false; `//by default`

Put the agent's ids to ascending order.

Let Agent1 be the agent with the first id from the ordered ids, Agent2 be the agent with the next id from the ordered ids, so on and the other agents until AgentK which will be the agent with the last id from the ordered ids.

Let **Dir** be the direction that Agent1 should follow to reach AgentK without passing through Agent 2.

Let now Agent1 be the agent with the smallest id. Agent2 be the next agent after Agent1 towards Dir, so on and the other agents until AgentK which will be the next agent towards Dir after AgentK-1.

Let *d1* be the distance between Agent1 and Agent2, *d2* be the distance between Agent2 and Agent3, so on and the other distances until *d6 which will* be the distance between AgentK and Agent1.

case 1) you are Agent1:

repeat

move towards Dir;

until (you see another agent)

execute Procedure Move_Together_ver1 (Dir)(one_BH_found);

case 2) you are Agent2:

repeat

wait;

until (d1 time units passed) **OR** (you see another agent)

if (you see another agent) **then**

execute Procedure Move_Together_ver1 (Dir)(one_BH_found);

else`//the time units needed have passed and no agent showed up`

repeat

move towards Dir;

until (you see another agent)

execute Procedure Move_Together_ver1 (Dir)(one_BH_found);

endif

case 3) you are any agent apart from Agent1 and Agent2:

Let d be the distance between you and Agent1 at the positive direction;

repeat

wait;

until (d1+3*(d-d1-1)+1 time units passed) **OR** (you see another agent)

if you see another agent **then** `//agents updates their one_BH_found flags`

execute Procedure Move_Together_ver1 (Dir)(one_BH_found);

else `//agent must be alone at that segment or the ring`

BHS is impossible for you, stop the execution;

end

endcase

Procedure: Move_Together_ver1(Dir)(one_BH_found)

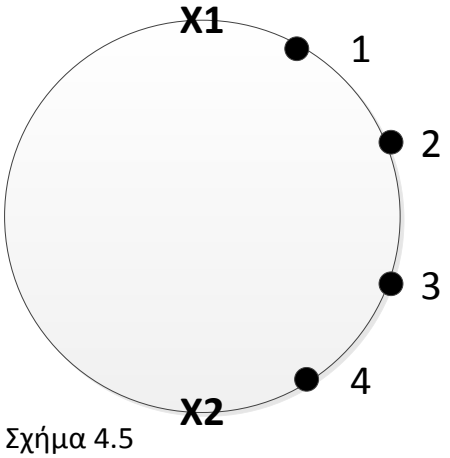
```
//The procedure takes as input:
//i)   Dir: the direction that agents must follow to explore
//ii)  a flag representing whether the agents have already found a BH or not
//The following commands means:
//move -> move one step
//wait -> wait one time unit
//alone agent -> agent is alone at his node
```

If (not(one_BH_found)) **then**
  repeat
    Look at your safe nodes set;
    case 1) next node towards Dir is safe:
      move towards Dir;
    case 2) next node towards Dir is unexplored:
      execute Procedure cautious walk(Dir);
    endcase
  until(black hole found)
one_BH_found=true;
endif
if (you are alone) **then**
    repeat
      move at the opposite direction of Dir;
    until(you see another agent)
endif
if (you are at least two agents at the same node) **then**
    repeat
      Look at your safe nodes set;
      case 1) next node at the opposite direction of Dir is safe:
        move at the opposite direction of Dir;
      case 2) next node at the opposite direction of Dir is unexplored:
        execute Procedure cautious walk(opposite direction of Dir);
      endcase
    until(All black holes have been found)
endif

Λήμμα 4.3

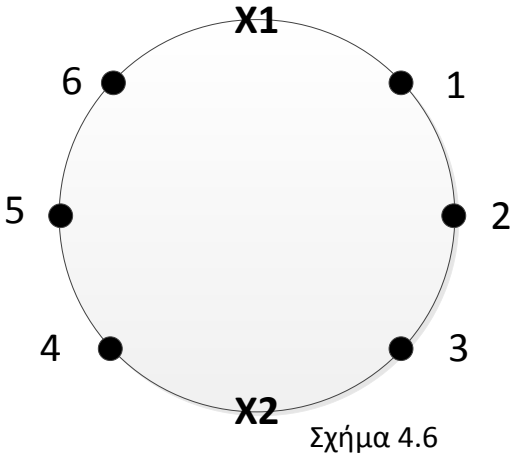
Ο αλγόριθμος BH2_MinID (κατηγορίας 1), επιλύει το πρόβλημα της αναζήτησης δύο μαύρων τρυπών, εάν στο δακτύλιο υπάρχουν τουλάχιστον έξι πράκτορες.

Απόδειξη:



Σχήμα 4.5

1η περίπτωση: Τουλάχιστον τέσσερεις πράκτορες βρίσκονται στην ίδια συνεκτική συνιστώσα. Φαίνεται εύκολα στον αλγόριθμο ότι οπωσδήποτε μετά από κάποιο χρονικό διάστημα, θα κινηθεί ακριβώς ένας από τους πράκτορες 1, 2, 3, ή 4 και μάλιστα προς τη θετική κατεύθυνση. Αν αυτός που θα κινηθεί είναι ένας από τους 1, 2 ή 3 τότε η ορθότητα του αλγορίθμου είναι προφανής (μαζί με τουλάχιστον άλλον ένα θα ανακαλύψουν τη μαύρη τρύπα X2 και στη συνέχεια επιστρέφουν και ανακαλύπτουν τη μαύρη τρύπα X1). Στην περίπτωση όπου ο πρώτος πράκτορας που θα κινηθεί είναι ο 4 τότε μετά από κάποιο συγκεκριμένο χρονικό διάστημα που ορίζεται στον αλγόριθμο, θα κινηθεί ο πράκτορας 1 προς την θετική κατεύθυνση και μετά τη συνάντησή του με τους 2 και 3 ανακαλύπτουν τις δύο μαύρες τρύπες.



Σχήμα 4.6

2η περίπτωση: Σε κάθε συνεκτική συνιστώσα υπάρχουν τρείς πράκτορες. Ακριβώς ένας πράκτορας θα επιλεγεί για να κινηθεί (κατά τη θετική φορά). Ας υποθέσουμε πως αυτός ο πράκτορας, χωρίς βλάβη της γενικότητας, ότι είναι ένας από τους 1, 2 και 3. Αν είναι ο 1 ή ο 2 τότε εύκολα φαίνεται ότι κάποιος από τους πράκτορες αυτής της συνεκτικής συνιστώσας θα ανακαλύψει τις θέσεις των δύο μαύρων τρυπών. Αν είναι ο πράκτορας 3 τότε μετά από συγκεκριμένο χρονικό διάστημα, που ορίζεται στον αλγόριθμο μας, ο πράκτορας 4 θα αρχίσει να κινείται κατά τη θετική φορά και συνεπώς το πρόβλημα λύνεται όπως παραπάνω (δηλαδή όπως όταν ξεκινούσε πρώτος ο πράκτορας 1).

□

Πολυπλοκότητα

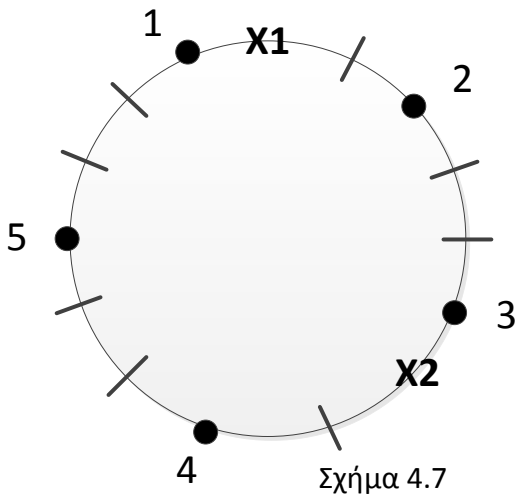
Είναι εύκολο να δει κανείς ότι η χρονική **πολυπλοκότητα** του αλγορίθμου BH2_MinID είναι $O(n)$, καθώς επίσης ότι όλοι οι πράκτορες τερματίζουν μετά από πεπερασμένο χρόνο.

□

Λήμμα 4.4

Δεν υπάρχει αλγόριθμος κατηγορίας ένα ο οποίος να λύνει το πρόβλημα της αναζήτησης δύο μαύρων τρυπών σε ένα δακτύλιο χρησιμοποιώντας ***λιγότερους από 6 πράκτορες.***

Απόδειξη:



Σχήμα 4.7

Έστω ένας αλγόριθμος κατηγορίας 1 ο οποίος λύνει το πρόβλημα με πέντε πράκτορες. Αυτός ο αλγόριθμος θα πρέπει να κινήσει τουλάχιστον έναν πράκτορα. Θεωρήστε ότι οι πράκτορες είναι τοποθετημένοι έτσι ώστε ανά δύο να έχουν απόσταση τουλάχιστον τρείς ακμές όπως φαίνεται στο σχήμα 4.7. Μάλιστα τουλάχιστον ένας από τους πράκτορες θα επιχειρήσει να επισκεφτεί τουλάχιστον δυο ανεξερεύνητους κόμβους (στην αντίθετη περίπτωση θα υπάρχουν τουλάχιστον δύο κόμβοι που δεν θα τους έχει επισκεφτεί κανένας πράκτορας). Έστω, χωρίς βλάβη της γενικότητας, ότι ο πράκτορας 1 είναι ο πρώτος (ή ένας από τους πρώτους) που κινείται με βάση τον αλγόριθμο και επιχειρεί να επισκεφτεί δύο ανεξερεύνητους κόμβους. Τότε ο ανταγωνιστής επιλέγει να τοποθετήσει τη μία μαύρη τρύπα σε έναν από τους δύο αυτούς κόμβους και την άλλη μαύρη τρύπα με τέτοιον τρόπο ώστε οι εναπομείναντες πράκτορες να μοιράζονται (ανά δύο) στις δύο συνεκτικές συνιστώσες. Μα τότε είναι προφανές ότι, σε οποιαδήποτε συνιστώσα ένας πράκτορας από τους δύο χάνεται για την ανακάλυψη της θέσης της πρώτης μαύρης τρύπας, ενώ ο άλλος πράκτορας δεν μπορεί να συμπεράνει μόνος του τη θέση της δεύτερης μαύρης τρύπας.

□

4.2 Αλγόριθμος κατηγορίας 2 (BH2_MinID with Move_Together_ver2)

Ο αλγόριθμος που ακολουθεί είναι κατηγορίας 2 και μπορεί να ανακαλύψει δύο μαύρες τρύπες σε έναν δακτύλιο χρησιμοποιώντας **πέντε** πράκτορες. Ο αλγόριθμος αυτός είναι βέλτιστος, ως προς τον αριθμό των πρακτόρων που χρησιμοποιεί, καθώς έχει αποδειχθεί ότι τέσσερις πράκτορες δεν είναι αρκετοί για να ολοκληρωθεί η αναζήτηση των δυο μαύρων τρυπών με επιτυχία.

Συνοπτικά το μοντέλο περιγράφεται ως εξής:

- Πέντε πράκτορες
- Ακριβώς δύο μαύρες τρύπες
- Τοπολογία: Δακτυλίου (Ring)
- Ανώνυμο δίκτυο με n κόμβους

Επιπρόσθετα οι πράκτορες:

- Κάθε πράκτορας έχει μοναδική ταυτότητα (Id) μέσα από ένα ολικά διατεταγμένο σύνολο
- Έχουν χάρτη του δικτύου με σημειωμένες τις αρχικές θέσεις και τις ταυτότητες των πρακτόρων
- Είναι συγχρονισμένοι
- Έχουν αρκετή μνήμη, δεν θα χρειαστεί περισσότερο από $O(n)$
- × **Δεν** έχουν συμφωνήσει στον προσανατολισμό του δακτυλίου

Βασική ιδέα:

Ο παρακάτω αλγόριθμος διαφέρει από τον προηγούμενο Αλγόριθμο (BH2_MinID) ως ακολούθως: Μετά την ανακάλυψη κάποιας μαύρης τρύπας από ακριβώς δύο πράκτορες (από τους οποίους έχει βέβαια επιζήσει ο ένας), ο πράκτορας που έχει επιζήσει αντί να κινηθεί κατά την αρνητική κατεύθυνση, περιμένει ένα συγκεκριμένο χρονικό διάστημα και εάν δεν εμφανιστεί κάποιος άλλος πράκτορας, τότε σταματά την εκτέλεση του αλγορίθμου, ανακοινώνοντας τη θέση της μαύρης τρύπας που ανακάλυψε.

Πιο συγκεκριμένα η βασική διαφορά του νέου αλγόριθμου από τον αλγόριθμο BH2_MinID που περιγράφτηκε παραπάνω βρίσκεται στη διαδικασία Move_Together, ενώ ο βασικός κώδικας είναι κοινός και για τους δύο.

Procedure: Move_Together_ver2(Dir)(one_BH_found)

```
//The procedure takes as input:
```

```
//i)   Dir: the direction that agents must follow to explore
//ii)   one_BH_found: a flag representing whether the agents have already
found a BH or not
//The following commands means:
//move -> move one step
//wait -> wait one time unit
```
If (not(one_BH_found)) **then**
  repeat
    Look at your safe nodes set;
    case 1) next node towards Dir is safe:
      move towards Dir;
    case 2) next node towards Dir is unexplored:
      execute Procedure cautious walk(Dir);
    end case
  until(black hole found)
one_BH_found=true;
endif
if (you are alone) **then**
    Let d be the distance between AgentK and agent1 towards Dir;
    Let d2 be the distance between your current position and AgentK towards Dir;
    Let t= d1+3*(d-d1-1)+1 + 3*(d2-1)+1 - time_units_already_passed;
    repeat
      wait;
    until(t time units passed)**AND** (no agent showed up)
    if(you see another agent)**then**
      move at the opposite direction of Dir;
    endif
    /*the agents will update their safe areas and will move towards the
    opposite direction of Dir */
endif
if (you are at least two agents at the same node) **then**
    repeat
      Look at your safe nodes set;
      case 1) next node at the opposite direction of Dir is safe:
        move at the opposite direction of Dir;
      case 2) next node at the opposite direction of Dir is unexplored:
        execute Procedure cautious walk(opposite direction of Dir);
      end case
    until(all black holes have been found)
endif

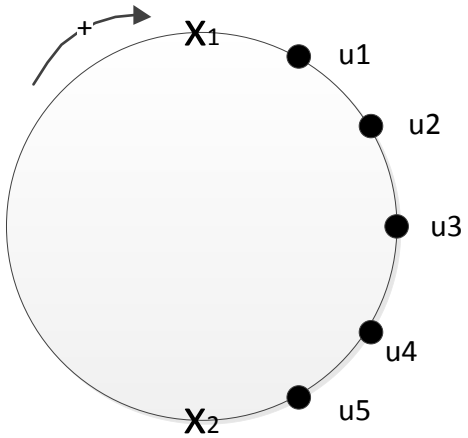
Λήμμα 4.5

Ο παραπάνω αλγόριθμος BH2_MinID with Move_Together_ver2 (κατηγορίας 2), επιλύει το πρόβλημα αναζήτησης δύο μαύρων τρυπών σε χρόνο $O(n)$ *χρησιμοποιώντας* ***πέντε*** *πράκτορες.*

Απόδειξη

Όπως φαίνεται στον αλγόριθμο, επιλέγεται ως Α1 ο πράκτορας με τη μικρότερη ταυτότητα, ως Α2 ο επόμενος πράκτορας μετά τον Α1 κατά τη φορά που έχουν συμφωνήσει οι πράκτορες ως θετική, ως Α3 ο επόμενος πράκτορας μετά τον Α2 κατά τη φορά που έχουν συμφωνήσει οι πράκτορες ως θετική και ούτω καθεξής. Σε κάθε έναν πράκτορα αντιστοιχίζεται ένας χρόνος $t(A_i)$. Ο αλγόριθμος δίνει οδηγίες σε κάθε πράκτορα να περιμένει $t(A_i)$ χρονικές μανάδες και μετά να αρχίσει να κινείται συνέχεια προς την κατεύθυνση που έχουν συμφωνήσει οι πράκτορες ότι είναι θετική, μέχρι να συναντήσει κάποιον πράκτορα. Ο αλγόριθμος αρχικά θέτει τις τιμές ως εξής: $t(A_1) = 0 < t(A_2) < t(A_3) \ldots < t(A_5)$. Έτσι λοιπόν ο πράκτορας Α1 ξεκινά να κινείται.

Οι συνεκτικές συνιστώσες στις οποίες χωρίζουν το δακτύλιο οι δύο μαύρες τρύπες φαίνονται στα σχήματα (4.8α, 4.8β, 4.8γ) όπου u1,u2,u3,u4,u5 αντιστοιχούν στις αρχικές θέσεις των πρακτόρων. Ας υποθέσουμε ότι, χωρίς βλάβη της γενικότητας, οι πράκτορες έχουν συμφωνήσει στην κατεύθυνση που δείχνει το βέλος στα σχήματα.

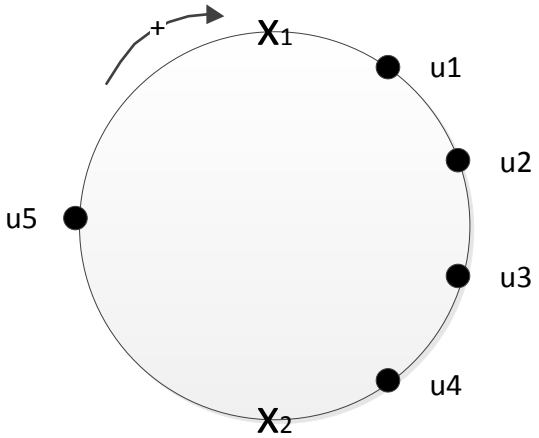


Σχήμα 4.8α

Περίπτωση 1α: Εάν ο Α1 είναι κάποιος από τους u1, u2 ή u3 του σχήματος 4.8α τότε καθώς κινείται θα συναντήσει τουλάχιστον τους πράκτορες Α4 και Α5 που βρίσκονται αντίστοιχα στα σημεία u4 και u5. Έτσι λοιπόν με βάση τον αλγόριθμο θα ανακαλυφθεί η μαύρη τρύπα Χ2, ενώ χάνεται ένας πράκτορας και οι τουλάχιστον δύο εναπομείναντες αρχίζουν να κινούνται προς την αρνητική κατεύθυνση, όπου ανακαλύπτουν τη δεύτερη μαύρη τρύπα Χ1.

Περίπτωση 2α: Εάν ο Α1 είναι ο u4 του σχήματος 4.8α τότε, ο πράκτορας αυτός κινείται κατά τη θετική κατεύθυνση, συναντά τον επόμενο πράκτορας (u5), ανακαλύπτουν τη μαύρη τρύπα Χ2 και ο πράκτορας που επέζησε περιμένει. Μετά από κατάλληλο χρόνο, αρχίζει να κινείται ο πράκτορας στη θέση u1 συναντά τους πράκτορες που βρίσκονται πριν από τον u4 (u2 και u3) και αφού συναντήσουν τον u4 κινούνται κατά την αρνητική κατεύθυνση γρήγορα μέχρι την αρχική θέση u1 και συνεχίζουν την εξερεύνηση όπου ανακαλύπτουν τη δεύτερη μαύρη τρύπα Χ1.

Περίπτωση 3α: Εάν ο Α1 είναι ο u5 του σχήματος 4.8α τότε, ο πράκτορας αυτός κινείται κατά τη θετική κατεύθυνση και χάνεται στη μαύρη τρύπα Χ2. Μετά από κατάλληλο χρόνο, αρχίζει να κινείται ο πράκτορας στη θέση u1 συναντά τους υπόλοιπους πράκτορες και μαζί ανακαλύπτουν τις δύο μαύρες τρύπες με όμοιο τρόπο όπως στην περίπτωση 1α.



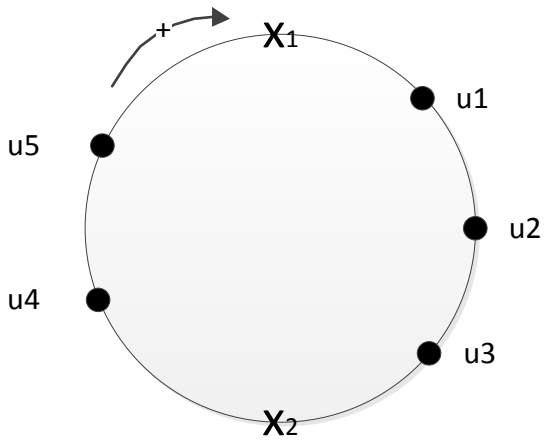
Σχήμα 4.8β

Περίπτωση 1β: Εάν ο Α1 είναι κάποιος από τους u1 ή u2 του σχήματος 4.8β τότε όμοια με την περίπτωση 1α οι τέσσερεις πράκτορες της μίας συνεκτικής συνιστώσας θα ανακαλύψουν τις δύο μαύρες τρύπες. Μετά από αρκετό χρόνο ο πράκτορας της θέσης u5 θα κινηθεί κατά τη θετική κατεύθυνση όπου και θα χαθεί στη μαύρη τρύπα Χ1.

Περίπτωση 2β: Εάν ο Α1 είναι ο u3 του σχήματος 4.8β τότε, όμοια με την περίπτωση 2α οι τέσσερεις πράκτορες της μίας συνεκτικής συνιστώσας θα ανακαλύψουν τις δύο μαύρες τρύπες ενώ ο u5 θα χαθεί στη μαύρη τρύπα Χ1 όπως και στην περίπτωση 1β.

Περίπτωση 3β: Εάν ο Α1 είναι ο u4 του σχήματος 4.8β τότε, όμοια με την περίπτωση 3α οι τέσσερεις πράκτορες της μίας συνεκτικής συνιστώσας θα ανακαλύψουν τις δύο μαύρες τρύπες ενώ ο u5 θα χαθεί στη μαύρη τρύπα Χ1 όπως και στην περίπτωση 1β.

Περίπτωση 4β: Εάν ο Α1 είναι ο u5 του σχήματος 4.8β τότε, ο πράκτορας αυτός θα κινηθεί κατά τη θετική κατεύθυνση και θα χαθεί στη μαύρη τρύπα Χ1. Μετά από κατάλληλο χρόνο θα κινηθεί ο πράκτορας της θέσης u1 και μαζί με τους υπόλοιπους πράκτορες θα ανακαλύψουν τις θέσεις των δύο μαύρων τρυπών όμοια με την περίπτωση 1α.



Σχήμα 4.8γ

Περίπτωση 1γ: Εάν ο Α1 είναι ο u1 του σχήματος 4.8γ τότε, όμοια με την περίπτωση 1α οι τρείς πράκτορες της μίας συνεκτικής συνιστώσας ανακαλύπτουν τις δύο μαύρες τρύπες. Μετά από κάποιο συγκεκριμένο χρόνο ο πράκτορας της θέσης u4 κινείται κατά τη θετική κατεύθυνση, συναντά τον u5 και ανακαλύπτουν τη μαύρη τρύπα Χ1. Τέλος ο πράκτορας που επέζησε (από τους u4 και u5) μετά από αρκετό χρόνο τερματίζει γνωρίζοντας τη θέση της μαύρης τρύπας Χ1.

Περίπτωση 2γ: Εάν ο Α1 είναι ο u2 του σχήματος 4.8γ τότε, όμοια με την περίπτωση 2α οι τρείς πράκτορες της μίας συνεκτικής συνιστώσας ανακαλύπτουν τις δύο μαύρες τρύπες, ενώ οι u4 και u5 ακολουθούν την ίδια τακτική της περίπτωσης 1γ.

Περίπτωση 3γ: Εάν ο Α1 είναι ο u3 του σχήματος 4.8γ τότε, ο πράκτορας αυτός κινείται κατά τη θετική κατεύθυνση και χάνεται στη μαύρη τρύπα Χ2. Μετά από κατάλληλο χρόνο, αρχίζει να κινείται ο πράκτορας στη θέση u4 συναντά τον πράκτορα u5 και όμοια με την περίπτωση 1γ ανακαλύπτουν την πρώτη μαύρη τρύπα Χ1. Μετά από αρκετό χρόνο κινείται ο πράκτορας της θέσης u1 συναντά τον u2 και κινούνται προς τη θετική κατεύθυνση, όπου ανακαλύπτουν τη δεύτερη μαύρη τρύπα Χ2.

Περίπτωση 4γ: Εάν ο Α1 είναι ο u4 του σχήματος 4.8γ τότε οι πράκτορες u4 και u5, όμοια με την περίπτωση 1γ ανακαλύπτουν τη θέση της μαύρης τρύπας Χ1. Μετά από κατάλληλο χρόνο ο πράκτορας στη θέση u1 ξεκινά να κινείται κατά τη θετική κατεύθυνση και αντίστοιχα με την περίπτωση 1α οι πράκτορες u1, u2 και u3 ανακαλύπτουν και τις δύο μαύρες τρύπες.

Περίπτωση 5γ: Εάν ο Α1 είναι ο u5 του σχήματος 4.8γ τότε, ο πράκτορας αυτός κινείται κατά τη θετική κατεύθυνση και χάνετε στη μαύρη τρύπα Χ1. Μετά από λίγο χρόνο ξεκινά κατά τη θετική κατεύθυνση ο πράκτορας στην θέση u1 και όμοια με την περίπτωση 1α οι πράκτορες αυτής της συνεκτικής συνιστώσας ανακαλύπτουν και τις δύο μαύρες τρύπες. Τέλος μετά από αρκετό χρόνο κινείται ο u4, ο οποίος θα χαθεί στη μαύρη τρύπα Χ1.

Είναι φανερό ότι σε όλες τις παραπάνω περιπτώσεις που εξετάσαμε για όλα τα σχήματα οι πράκτορες ανακαλύπτουν και τις δύο μαύρες τρύπες και μάλιστα όλοι οι πράκτορες τερματίζουν.

□

4.3 Κινείται ο πράκτορας με την μικρότερη απόσταση μπροστά του

Γνωρίζουμε λοιπόν ότι για την κατηγορία 1 οι βέλτιστοι αλγόριθμοι ως προς τον αριθμό των πρακτόρων, χρησιμοποιούν 5 πράκτορες. Ένα ενδιαφέρον ερώτημα είναι ποιος αλγόριθμος έχει τη μικρότερη πολυπλοκότητα.

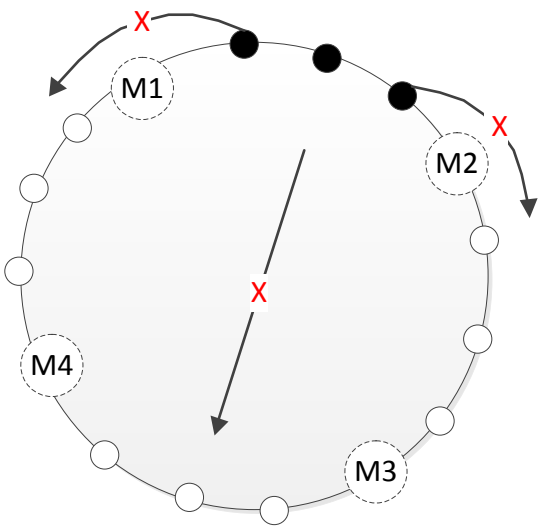
Για παράδειγμα ένας ακόμα αλγόριθμος κατηγορίας 1, που διαφέρει ελάχιστα από τον αλγόριθμο BH2_MinID, θα ήταν ένας αλγόριθμος ο οποίος επιλέγει να κινηθεί ο πράκτορας που έχει να διανύσει τη μικρότερη απόσταση προς τον επόμενο πράκτορα. Σε αυτή την περίπτωση ελέγχονται ένα υποσύνολο περιπτώσεων όπου τα αποτελέσματα εξάγονται άμεσα (όπως για παράδειγμα αν δύο πράκτορες Α, Β απέχουν μόνο δύο ακμές, αν ο πράκτορας Α κινούμενος προς το Β χαθεί, ο πράκτορας Β θα μπορούσε να συμπεράνει με βεβαιότητα τη θέση της μαύρης τρύπας ανάμεσά τους).

Η εύρεση των καλύτερων αλγορίθμων των δύο κατηγοριών είναι ένα ενδιαφέρον ανοιχτό πρόβλημα.

Κεφάλαιο 5 - Αλγόριθμοι αναζήτησης M>2 μαύρων τρυπών

Στο κεφάλαιο αυτό θα ασχοληθούμε με M μαύρες τρύπες, όπου M>2. Θα αναφερθούμε στα βασικά χαρακτηριστικά ενός δακτυλίου με M μαύρες τρύπες καθώς επίσης θα αποδείξουμε ποιος είναι ο αναγκαίος και ικανός αριθμός πρακτόρων για αυτό το πρόβλημα.

Θεωρήστε ένα δακτύλιο στον οποίο υπάρχουν περισσότερες από δύο μαύρες τρύπες, όπως στο παράδειγμα του διπλανού σχήματος (Σχήμα 5.1), στο οποίο οι μαύρες τρύπες είναι σημειωμένες με M1, M2, M3, M4. Οι μαύρες τρύπες αυτές αποσυνδέουν το δακτύλιο δημιουργώντας δύο συνεκτικές συνιστώσες. Προφανώς ένας οποιοσδήποτε πράκτορας που βρίσκεται σε μία από αυτές τις συνεκτικές συνιστώσες δεν μπορεί να επισκεφτεί κάποιον κόμβο ο οποίος ανήκει σε κάποια άλλη συνεκτική συνιστώσα. Άρα για να είναι δυνατή η ανακάλυψη όλων των μαύρων τρυπών πρέπει όλες οι μαύρες τρύπες να είναι προσβάσιμες από κάποιον πράκτορα. Για παράδειγμα, αν υπάρχουν τέσσερις μαύρες τρύπες, όπως φαίνεται στο σχήμα 5.1 και οι πράκτορες βρίσκονται στα τμήματα M1M2 και M3M4, τότε η επιτυχής αναζήτηση όλων των μαύρων τρυπών είναι πιθανή, ενώ αν οι πράκτορες βρίσκονται στα τμήματα M1M2 και M2M3 η αναζήτηση και των τεσσάρων μαύρων τρυπών είναι αδύνατη.



Σχήμα 5.1

Αρχικά θα ασχοληθούμε με τις βασικές ιδιότητες και τα χαρακτηριστικά της αναζήτησης μαύρων τρυπών σε κάποια συνεκτική συνιστώσα ενός δακτυλίου.

5.1 Βασικές ιδιότητες

Καθώς δεν υπάρχει πράκτορας που να έχει πρόσβαση σε περισσότερες από δύο μαύρες τρύπες σε ένα δακτύλιο, πλέον δεν ορίζεται αλγόριθμος κατηγορίας 1, στον οποίο μετά από πεπερασμένο χρόνο ένας τουλάχιστον από τους πράκτορες σταματά και γνωρίζει τις θέσεις όλων των μαύρων τρυπών. Συνεπώς στο σημείο αυτό

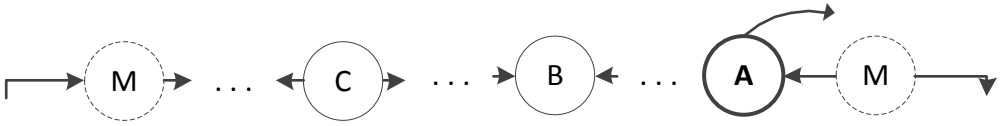
μπορούμε να ορίσουμε ως αλγόριθμους κατηγορίας 4, τους αλγορίθμους στους οποίους μετά από πεπερασμένο χρόνο ένας τουλάχιστον από τους πράκτορες σταματά και γνωρίζει τις θέσεις των δύο μαύρων τρυπών που μπορεί να προσεγγίσει.

Προφανώς δύο πράκτορες ανεξάρτητα αν ξεκινούν στον ίδιο κόμβο ή όχι δεν μπορούν να ανακαλύψουν δύο μαύρες τρύπες και τουλάχιστον ένας από αυτούς να επιζήσει. Δηλαδή για οποιοδήποτε αλγόριθμο που χρησιμοποιεί δύο μόνο πράκτορες υπάρχει τουλάχιστον μία περίπτωση τοποθέτησης των μαύρων τρυπών έτσι ώστε ο αλγόριθμος να μη δουλέψει.

Αν τρείς πράκτορες ξεκινούν από τον ίδιο κόμβο τότε προφανώς οι πράκτορες μπορούν να ανακαλύψουν τις δυο μαύρες τρύπες που μπορούν να προσεγγίσουν. Εάν σε κάποια συνεκτική συνιστώσα έχουμε τρεις πράκτορες και δύο από αυτούς ξεκινούν από τον ίδιο κόμβο τότε, ένας απλός αλγόριθμος θα έλεγε στους πράκτορες που είναι μαζί, να κινηθούν χρησιμοποιώντας τη διαδικασία Cautious Walk. Συνεπώς οι δύο πράκτορες θα ανακαλύψουν πρώτα τη μία μαύρη τρύπα και στη συνέχεια αφού συναντηθούν θα ανακαλύψουν και τη δεύτερη μαύρη τρύπα ή αφού και οι τρείς πράκτορες συναντηθούν θα ανακαλύψουν τις δύο μαύρες τρύπες. Όπως θα δούμε στο λήμμα που ακολουθεί, όταν οι πράκτορες ξεκινούν από διαφορετικές θέσεις, τρείς πράκτορες δεν αρκούν για την εξερεύνηση των μαύρων τρυπών της ίδια συνεκτικής συνιστώσας.

Λήμμα 5.1:

Δεν υπάρχει αλγόριθμος κατηγορίας 4 ο οποίος λύνει το πρόβλημα της αναζήτησης δύο μαύρων σε μία συνεκτική συνιστώσα χρησιμοποιώντας τρεις πράκτορες.



Σχήμα 5.2

Απόδειξη:

Ας υποθέσουμε πως έχουμε 3 πράκτορες αρχικά τοποθετημένοι σε διαφορετικές θέσεις στην ίδια συνεκτική συνιστώσα ενός δακτυλίου. Ας υποθέσουμε επίσης πως οι πράκτορες απέχουν μεταξύ τους τουλάχιστον τρεις ακμές. Σε αυτή την περίπτωση οποιοσδήποτε σωστός αλγόριθμος πρέπει να κινήσει τουλάχιστον έναν πράκτορα ώστε να επισκεφθεί τουλάχιστον δύο ανεξερεύνητους κόμβους (διότι αν όλοι οι

πράκτορες επισκεφτούν το πολύ έναν κόμβο τότε υπάρχουν κόμβοι που δεν τους επισκέπτεται κανείς).

Θεωρήστε τον πρώτο από τους πράκτορες (έστω Α) ή έναν από τους πρώτους που επισκέπτεται 2 ανεξερεύνητους κόμβους. Ο αντίπαλος βάζει τη μαύρη τρύπα σε έναν από αυτούς τους δύο κόμβους. Συνεπώς ο πράκτορας Α χάνεται. Οι υπόλοιποι δύο πράκτορες, μπορούν να συμπεράνουν με κάποιον μηχανισμό (αφού ο πράκτορας Α δεν φάνηκε) ότι η μαύρη τρύπα βρίσκεται σε έναν από τους δύο κόμβους που επισκέφτηκε ο πράκτορας Α, αλλά δεν μπορούν να συμπεράνουν ακριβώς σε ποίον από τους δύο βρίσκεται. Τελικά, έχουν μείνει δύο πράκτορες για να ανακαλύψουν τις θέσεις δύο μαύρων τρυπών, συνεπώς η αναζήτηση είναι αδύνατη.

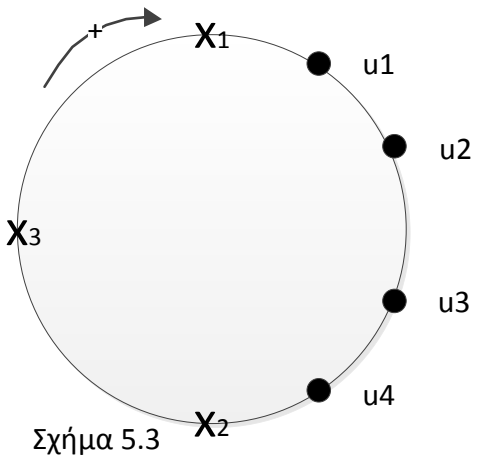
□

Βέβαια, υπάρχουν κάποιες υποπεριπτώσεις για τις οποίες η εξερεύνηση με 3 πράκτορες σε ένα τμήμα είναι δυνατή. Μία προφανής περίπτωση, θα ήταν αν οι πράκτορες μπορούσαν να συναντηθούν πριν χαθεί κάποιος από αυτούς σε μία μαύρη τρύπα. Εναλλακτικά, στην περίπτωση όπου για παράδειγμα οι πράκτορες απέχουν το πολύ 2 ακμές μεταξύ τους, αν χαθεί κάποιος πράκτορας Α, πριν προλάβει να συναντήσει τον επόμενο πράκτορα Β, ο πράκτορας Β μπορεί να συμπεράνει ότι ο πράκτορας Α χάθηκε ανάμεσα τους (διότι υπάρχει μόνο μια πιθανή θέση της μαύρης τρύπας, αλλιώς οι δύο πράκτορες θα είχαν συναντηθεί) και άρα να συμπεράνει τη θέση της μαύρης τρύπας.

Αν ο αριθμός των πρακτόρων του ίδιου τμήματος είναι μεγαλύτερος από τρεις, η εξερεύνηση είναι δυνατή. Η απόδειξη της παραπάνω πρότασης παρουσιάζεται στο λήμμα που ακολουθεί.

Λήμμα 5.2:

Τέσσερις πράκτορες που βρίσκονται στην ίδια συνεκτική συνιστώσα ενός δακτυλίου μπορούν να εντοπίσουν τις θέσεις των δύο μαύρων τρυπών που την ορίζουν.
Απόδειξη:



Σχήμα 5.3

Έστω ένας δακτύλιος με τέσσερις πράκτορες που ξεκινούν από διαφορετικές θέσεις. Οι θέσεις των δύο μαύρων τρυπών επιλέγονται έτσι ώστε όλοι οι πράκτορες να βρίσκονται ανάμεσα τους σε ένα από τα δύο τμήματα που αυτές ορίζουν (όπως για παράδειγμα φαίνεται στο παραπάνω (Σχήμα 5.3), συνεπώς και οι τέσσερεις πράκτορες θα βρίσκονται στην ίδια συνεκτική συνιστώσα.

Με βάση τον αλγόριθμο **BH2_MinID** κάποιος από τους πράκτορες (έστω A1) θα επιλεγεί να κινηθεί κατά τη θετική φορά.

Περίπτωσης 1η: Έστω ότι επιλέγεται A1 να είναι ο πράκτορας στην θέση u1. Τότε ο πράκτορας κινείται κατά τη θετική κατεύθυνση (όπως φαίνεται στο σχήμα με το βέλος) και συναντά τους πράκτορες στις θέσεις u2, u3 και u4. Τότε οι πράκτορες θα ανακαλύψουν τη θέση της πρώτης μαύρης τρύπας X2 και στη συνέχεια αφού κινηθούν κατά την αντίθετη κατεύθυνση θα ανακαλύψουν και τη δεύτερη μαύρη τρύπα X1.

Περίπτωση 2η: Έστω ότι επιλέγεται A1 να είναι ο πράκτορας στη θέση u2. Τότε ο πράκτορας κινείται κατά τη θετική κατεύθυνση συναντά τους πράκτορες στις θέσεις u3 και u4, με τους οποίους θα ανακαλύψει τη θέση της μαύρης X2. Στη συνέχεια δύο πλέον πράκτορες (αφού ένας χάθηκε στη μαύρη τρύπα για να την ανακαλύψουν) κινούνται κατά την αντίθετη κατεύθυνση με πριν και αφού συναντήσουν τον πράκτορα στη θέση u1 ανακαλύπτουν και τη δεύτερη μαύρη τρύπα (X1) της συνεκτικής τους συνιστώσας.

Περίπτωση 3η: Έστω ότι επιλέγεται A1 να είναι ο πράκτορας στη θέση u3. Τότε ο πράκτορας κινείται κατά τη θετική κατεύθυνση συναντά και αφού συναντήσει τον πράκτορα στη θέση u4 ανακαλύπτουν την πρώτη μαύρη τρύπα (X2). Ο πράκτορας που επέζησε από την εξερεύνηση, κινείται κατά την αντίθετη κατεύθυνση με πριν και αφού συναντήσει τους πράκτορες των αρχικών θέσεων u2 και u1 ανακαλύπτουν και τη δεύτερη μαύρη τρύπα της συνεκτικής τους συνιστώσας.

Περίπτωση 4η: Έστω ότι επιλέγεται Α1 να είναι ο πράκτορας στη θέση u4. Τότε ο πράκτορας κινείται κατά τη θετική κατεύθυνση και χάνεται στη μαύρη τρύπα Χ2. Μετά από συγκεκριμένο χρόνο (ο οποίος ορίζεται στον αλγόριθμο), ο πράκτορας της θέσης u1 κινείται κατά τη θετική κατεύθυνση συναντά τους πράκτορες των θέσεων u2 και u3 και μαζί τους ανακαλύπτει τη θέση της μαύρης τρύπας Χ2. Οι πράκτορες που επέζησαν κινούνται κατά την αντίθετη κατεύθυνση με πριν και ανακαλύπτουν και τη δεύτερη μαύρη τρύπα Χ1.

Συνεπώς τέσσερεις πράκτορες που βρίσκονται στην ίδια συνεκτική συνιστώσα, μπορούν να ανακαλύψουν τις θέσεις των μαύρων τρυπών που έχουν πρόσβαση.

□

Στο σημείο αυτό, θα επικεντρωθούμε στις περιπτώσεις όπου η αναζήτηση M>2 μαύρων τρυπών είναι πραγματοποιήσιμη.

- Αν για κάθε μαύρη τρύπα υπάρχει τουλάχιστον μία συνεκτική συνιστώσα στην οποία υπάρχουν τουλάχιστον τέσσερεις πράκτορες, τότε σε κάθε συνεκτικής συνιστώσας θα υπάρχει τουλάχιστον ένας πράκτορας ο οποίος να γνωρίζει τις θέσεις των δύο μαύρων τρυπών που έχει πρόσβαση και προφανώς η αναζήτηση M>2 μαύρων τρυπών είναι δυνατή.
- Αν σε κάθε συνεκτική συνιστώσα υπάρχουν τρεις πράκτορες, οι πράκτορες αυτοί μπορούν να γνωρίζουν συλλογικά τις θέσεις των M μαύρων τρυπών.

Σε αυτό το κεφάλαιο παρουσιάσαμε κάποιες βασικές ιδιότητες της αναζήτησης M>2 μαύρων τρυπών και ποιο συγκεκριμένα επικεντρωθήκαμε στην αναζήτηση M μαύρων τρυπών ανάλογα με τον αριθμό των πρακτόρων στις συνεκτικές συνιστώσες που ορίζουν οι μαύρες τρύπες σε έναν δακτύλιο.

Ένα ενδιαφέρον ανοιχτό πρόβλημα είναι, η εύρεση του ελάχιστου αριθμού πρακτόρων για την αναζήτηση M>2 μαύρων τρυπών σε ένα δακτύλιο και ποια θα είναι η διάταξη των πρακτόρων σε αυτή την περίπτωση.

Κεφάλαιο 6 - Συμπεράσματα

Δώσαμε έναν αλγόριθμο ο οποίος λύνει το πρόβλημα της αναζήτησης μίας μαύρης τρύπας με τρεις πράκτορες, ενώ παράλληλα αποδείξαμε ότι με δύο πράκτορες η αναζήτηση είναι αδύνατη (για οποιοδήποτε διάταξη των πρακτόρων στο δακτύλιο). Επιπλέον δώσαμε έναν αλγόριθμο ο οποίος λύνει το πρόβλημα της αναζήτησης ακριβώς μίας μαύρης τρύπας όταν οι πράκτορες γνωρίζουν μόνο τον προσανατολισμό του δακτυλίου (χωρίς χάρτη ή τη γνώση για τη διάταξη των πρακτόρων), χρησιμοποιώντας $K(n)$ πράκτορες. Δημιουργήσαμε δύο αλγόριθμους που λύνουν το πρόβλημα αναζήτησης δύο μαύρων τρυπών χρησιμοποιώντας τον ελάχιστο αριθμό πρακτόρων για διάφορες περιπτώσεις. Αποδείξαμε την ορθότητα όλων των αλγορίθμων και υπολογίσαμε τη χρονική τους πολυπλοκότητα. Στη συνέχεια παρουσιάσαμε κάποιες βασικές ιδιότητες και κάποια πρώτα αποτελέσματα όταν υπάρχουν περισσότερες από δύο μαύρες τρύπες. Τέλος σχεδιάσαμε και υλοποιήσαμε σε κώδικα Java έναν από τους αλγόριθμούς μας.

Αυτή η πτυχιακή αποτελεί το πρώτο ουσιαστικό βήμα στη σχεδίαση βέλτιστων αλγορίθμων για την ανακάλυψη σφαλμάτων σε δίκτυα. Πιστεύουμε ότι κάποιοι από τους μηχανισμούς που προτάθηκαν και αναλύθηκαν εδώ θα μπορούσαν να φανούν πολύ χρήσιμοι για την ανάλυση του προβλήματος σε άλλες τοπολογίες και σε πιο αδύναμα μοντέλα (π.χ. όταν οι πράκτορες δεν έχουν ταυτότητες, όταν το δίκτυο δεν είναι συγχρονισμένο, όταν η μνήμη των πρακτόρων είναι πολύ περιορισμένη κτλ). Σημειώνουμε εδώ ότι, όπως έχουμε επαναλάβει στην εισαγωγή, αν και υπάρχουν πολλές λύσεις στη διεθνή βιβλιογραφία για το πρόβλημα της αναζήτησης μαύρης τρύπας, σε διάφορα μοντέλα, στην πτυχιακή αυτή εργασία εξετάζεται για πρώτη φορά, απ’ όσο γνωρίζουμε, το πρόβλημα σε συγχρονισμένο δίκτυο, όταν οι πράκτορες ξεκινούν από διαφορετικές θέσεις και δεν έχουν τη δυνατότητα να αφήσουν οποιοδήποτε είδους σημάδι στους κόμβους του δικτύου. Τα ενδιαφέροντα ανοιχτά προβλήματα είναι πολλά, με ίσως κεντρικότερο, το εξής: Ποια είναι τα όρια του συγχρονισμένου μοντέλου και πώς αυτά μπορούν να αλλάξουν, εάν αλλάξουμε κάποιες παραμέτρους του; Δηλαδή, σε ποιες τοπολογίες δικτύου μπορεί να εφαρμοστεί και ποιος είναι ο μέγιστος αριθμός μαύρων τρυπών που μπορεί να ανακαλύψει, χωρίς την εισαγωγή πιο δυνατού μοντέλου επικοινωνίας. Επιπλέον ποια είναι η ελάχιστη μνήμη που πρέπει να έχει κάθε πράκτορας; Ποιος είναι ο ελάχιστος αριθμός πρακτόρων; Και τέλος πώς αυτός συσχετίζεται με το μέγιστο αριθμό μαύρων τρυπών που μπορεί να ανακαλυφθεί σε κάθε τοπολογία δικτύου;

Σε αυτές τις ερωτήσεις και στην περεταίρω έρευνα του προβλήματος πιστεύουμε ότι θα μπορούσε να φανεί πολύ χρήσιμο το υλοποιημένο πρόγραμμα που συνοδεύει την παρούσα εργασία, καθώς με ελάχιστες μετατροπές, θα μπορούσε να χρησιμεύσει στο μελλοντικό ερευνητή για τον πειραματισμό με νέους αλγόριθμους

και νέα μοντέλα στο τομέα της ανακάλυψης βέλτιστων αλγορίθμων για όσο το δυνατόν πιο γενικά μοντέλα.

6.1 Ανοιχτά προβλήματα

Παρακάτω δίνουμε έναν κατάλογο με ανοιχτά προβλήματα των οποίων η λύση πιστεύουμε ότι θα βελτίωνε σημαντικά τα ερευνητικά αποτελέσματα πάνω στο συγκεκριμένο πρόβλημα.

Πώς αλλάζουν τα αποτελέσματα στο δακτύλιο όταν:

- Οι πράκτορες δεν έχουν στη διάθεση τους χάρτη.
- Δεν έχουν συμφωνήσει στον προσανατολισμό του δακτυλίου.
- Δεν έχουν διαφορετικές ταυτότητες.
- Έχουν μεγάλους περιορισμούς στη μνήμη.

Ποια είναι τα αποτελέσματα σε άλλες τοπολογίες δικτυού;

Σημειώνουμε εδώ, ότι με κάποιες από τις περιπτώσεις που αναφέρθηκαν παραπάνω ασχοληθήκαμε και εμείς στην παρούσα εργασία, δίνοντας κάποια πρώτα αποτελέσματα (π.χ. απουσία χάρτη και ασυμφωνία στον προσανατολισμό του δακτυλίου). Επίσης σημαντική βελτίωση των αποτελεσμάτων της παρούσας εργασίας θα αποτελούσε η εύρεση βέλτιστων αλγορίθμων, όχι μόνο ως προς τον αριθμό των πρακτόρων, αλλά ως προς τη χρονική πολυπλοκότητα, και ως προς τη μνήμη των πρακτόρων κτλ.

Επιπρόσθετα, πολύ σημαντικά θα ήταν τα αποτελέσματα που αφορούν την αντιστάθμιση (trade offs) μεταξύ αυτών των παραμέτρων.

Επίσης το πρόγραμμα που δίνουμε έχει πολλά περιθώρια βελτίωσης έτσι ώστε να καταλήξει σε ένα φιλικό περιβάλλον που θα μπορούσε να διευκολύνει πολύ την απόδειξη νέων αποτελεσμάτων*.

Τέλος επειδή όπως τονίζεται και στην εισαγωγή, το πρόβλημα σχεδίαση βέλτιστων αλγορίθμων για την ανακάλυψη σφαλμάτων σε δίκτυα είναι πολύ σημαντικό, έχει ιδιαίτερο ενδιαφέρον (και πολλά ανοιχτά προβλήματα) η έρευνα σε άλλα αρκετά πιο διαφορετικά μοντέλα από αυτό που εξετάσαμε στην εργασία μας (π.χ. ασύγχρονα δίκτυα, διαφορετικά πρωτόκολλα επικοινωνίας μεταξύ των πρακτόρων κτλ).

* Στην πραγματικότητα ετοιμάζεται η επόμενη έκδοση του λογισμικού αισθητά βελτιωμένη ως προς τις περιπτώσεις που μπορεί να χειριστεί.

Η ανακάλυψη σφαλμάτων σε δίκτυα είναι ένα σημαντικό πρόβλημα του οποίου η λύση μερικές φορές (π.χ. όταν το δίκτυο δεν είναι ανεκτικό σε σφάλματα) είναι απαραίτητη για την απρόσκοπτη ροή αλλά και την ασφάλεια των δεδομένων που διακινούνται στο δίκτυο. Ένας από τους στόχους της συντήρησης ενός δικτύου (ιδιαίτερα κάποιου δικτύου όχι ανεκτικού σε σφάλματα) είναι η ανακάλυψη σφαλμάτων και η αναφορά των κόμβων στους οποίους συμβαίνουν.

Η παρούσα πτυχιακή εργασία ασχολήθηκε με την αναζήτηση μαύρων τρυπών σε ένα δίκτυο-δακτύλιο χρησιμοποιώντας τους ελάχιστους δυνατούς πόρους. Αποδείξαμε την ορθότητά τους και υπολογίσαμε τις πολυπλοκότητες τους.

Επιπρόσθετα, αποδείξαμε ότι σε κάποιες περιπτώσεις, η επίλυση του προβλήματος με τη χρήση λιγότερων πρακτόρων ή χαρακτηριστικών από αυτούς που χρησιμοποιούν οι αλγόριθμοί μας, είναι αδύνατη και συνεπώς σε αυτές τις περιπτώσεις οι αλγόριθμοι που προτείνουμε είναι βέλτιστοι ως προς τον αριθμό των πρακτόρων ή των απαιτήσεων τους. Καταληκτικά, συμπεριλαμβάνεται μία προσομοίωση του μοντέλου που αναλύθηκε στην παρούσα πτυχιακή και η υλοποίηση ενός από τους αλγορίθμους που αναπτύχθηκαν.

6.2 Μελλοντικές επεκτάσεις

Στην παρούσα μελέτη ασχοληθήκαμε με την απλή τοπολογία ενός ανώνυμου και συγχρονισμένου δακτυλίου στον οποίο βρίσκονται διακεκριμένοι πράκτορες, για την αναζήτηση ενός αριθμού από μαύρες τρύπες.

Ως μελλοντική έρευνα πάνω στην ήδη υπάρχουσα πτυχιακή εργασία, προτείνεται η βελτίωση των αλγορίθμων που χρησιμοποιήθηκαν, ως προς τις απαιτήσεις τους σε δεδομένα εισόδου (χάρτης του δικτύου, αριθμός πρακτόρων) όπου αυτό είναι δυνατό.

Ενδεικτική θα ήταν η μελέτη του προβλήματος αναζήτησης μαύρων τρυπών και σε άλλες τοπολογίες (όπως δένδρα και torus).

Επίσης σημαντικό ενδιαφέρον παρουσιάζει και η επιλογή ενός διαφορετικού μοντέλου για την επίλυση του ίδιου προβλήματος.

Ένα άλλο θέμα που θα απασχολούσε για λόγους βελτίωσης και σύγκρισης είναι η μελέτη του υπολογισμού ενός κάτω ορίου της πολυπλοκότητας για κάθε αλγόριθμο, με σκοπό την σύγκριση του κάτω αυτού ορίου με την πολυπλοκότητα που υπολογίστηκε στην παρούσα πτυχιακή εργασία.

Βιβλιογραφία

[1] N. Agmon, D. Peleg, Fault-Tolerant Gathering Algorithms for Autonomous Mobile Robots, SIAM Journal on Computing, 36 (2006) 56-82.

[2] S. Alpern, S. Gal, The Theory of Search Games and Rendezvous, Kluwer Academic Publishers, Norwell, Massachusetts, 2003.

[3] H. Ando, Y. Oasa, I. Suzuki, M. Yamashita, A Distributed Memoryless Point Convergence Algorithm for Mobile Robots with Limited Visibility, I.E.E.E. Transactions on Robotics and Animation, 15 (1999) 818--828.

[4] M.A. Bender, The power of team exploration: two robots can learn unlabeled directed graphs, in: D.K. Slonim (Ed.), 1994, pp. 75-85.

[5] N. Borselius, Mobile Agent Security, Electronics and Communication Engineering Journal, 14 (2002) 211--218.

[6] J. Chalopin, S. Das, A. Labourel, E. Markou, Tight Bounds for Scattered Black Hole Search in a Ring, in: Proceedings of 18th International Colloquium on Structural Information and Communication Complexity, 2011.

[7] J. Chalopin, S. Das, N. Santoro, Rendezvous of Mobile Agents in Unknown Graphs with Faulty Links, in: A. Pelc (Ed.) Distributed Computing, Springer Berlin / Heidelberg, 2007, pp. 108-122-122.

[8] C. Cooper, R. Klasing, T. Radzik, Searching for Black-Hole Faults in a Network Using Multiple Agents, in: M. Shvartsman (Ed.) Principles of Distributed Systems, Springer Berlin / Heidelberg, 2006, pp. 320-332-332.

[9] C. Cooper, R. Klasing, T. Radzik, Locating and Repairing Faults in a Network with Mobile Agents, in: Proceedings of International Colloquium on Structural Information and Communication Complexity, 2008, pp. 20--32.

[10] J. Czyzowicz, S. Dobrev, R. Kralovic, S. Miklik, D. Pardubska, Black hole search in directed graphs, in: Proceedings of 16th International Colloquium on Structural Information and Communication Complexity, 2009, pp. 182-194.

[11] J. Czyzowicz, D. Kowalski, E. Markou, A. Pelc, Searching for a Black Hole in Tree Networks

Principles of Distributed Systems, 3544 (2005) 900-900.

[12] J. Czyzowicz, D. Kowalski, E. Markou, A. Pelc, Complexity of Searching for a Black Hole, Fundamenta Informaticae, 71 (2006) 229--242.

[13] J. Czyzowicz, D. Kowalski, E. Markou, A. Pelc, Searching for a Black Hole in Synchronous Tree Networks, Combinatorics, Probability \& Computing, 16 (2007) 595--619.

[14] G. De~Marco, L. Gargano, E. Kranakis, D. Krizanc, A. Pelc, U. Vacaro, Asynchronous Deterministic Rendezvous in Graphs, in: Proceedings of 30th International Symposium on Mathematical Foundations of Computer Science, 2005, pp. 271--282.

[15] G. De~Marco, L. Gargano, E. Kranakis, D. Krizanc, A. Pelc, U. Vacaro, Asynchronous Deterministic Rendezvous in Graphs, Theoretical Computer Science, 355 (2006) 315--326.

[16] X. Deng, C.H. Papadimitriou, Exploring an Unknown Graph, Journal of Graph Theory, 32 (1999) 265--297.

[17] A. Dessmark, P. Fraigniaud, D.R. Kowalski, A. Pelc, Deterministic rendezvous in graphs, Algorithmica, 46 (2006) 69--96.

[18] S. Dobrev, P. Flocchini, R. Kralovic, N. Santoro, Exploring a Dangerous Unknown Graph Using Tokens, in: Proceedings of 5th IFIP International Conference on Theoretical Computer Science, 2006, pp. 131--150.

[19] S. Dobrev, P. Flocchini, G. Prencipe, N. Santoro, Mobile Search for a Black Hole in an Anonymous Ring

Distributed Computing, 2180 (2001) 166-179.

[20] S. Dobrev, P. Flocchini, G. Prencipe, N. Santoro, Multiple Agents Rendezvous in a Ring in Spite of a Black Hole, in: Proceedings of 6th International Conference on Principles of Distributed Systems, 2003, pp. 34--46.

[21] S. Dobrev, P. Flocchini, G. Prencipe, N. Santoro, Searching for a Black Hole in Arbitrary Networks: Optimal Mobile Agents Protocols, Distributed Computing, 19 (2006) 1--19.

[22] S. Dobrev, P. Flocchini, G. Prencipe, N. Santoro, Mobile Search for a Black Hole in an Anonymous Ring, Algorithmica, 48 (2007) 67--90.

[23] S. Dobrev, P. Flocchini, N. Santoro, Improved Bounds for Optimal Black Hole Search in a Network with a Map, in: Proceedings of 10th International Colloquium on Structural Information and Communication Complexity, 2004, pp. 111--122.

[24] S. Dobrev, R. Kralovic, N. Santoro, W. Shi, Black Hole Search in Asynchronous Rings Using Tokens, in: Proceedings of 6th Conference on Algorithms and Complexity, 2006, pp. 139--150.

[25] S. Dobrev, N. Santoro, W. Shi, Scattered Black Hole Search in an Oriented Ring Using Tokens, in: Proceedings of IEEE International Parallel and Distributed Processing Symposium, 2007, pp. 1--8.

[26] S. Dobrev, N. Santoro, W. Shi, Using Scattered Mobile Agents to Locate a Black Hole in an Un-Oriened Ring with Tokens, International Journal of Foundations of Computer Science, 19 (2008) 1355--1372.

[27] P. Flocchini, D. Ilcinkas, N. Santoro, Ping Pong in Dangerous Graphs: Optimal Black Hole Search with Pure Tokens, in: G. Taubenfeld (Ed.) Distributed Computing, Springer Berlin / Heidelberg, 2008, pp. 227-241-241.

[28] P. Flocchini, M. Kellett, P. Mason, N. Santoro, Map construction and exploration by mobile agents scattered in a dangerous network, in, 2009, pp. 1-10.

[29] P. Flocchini, E. Kranakis, D. Krizanc, N. Santoro, C. Sawchuk, Multiple Mobile Agent Rendezvous in the Ring, in: Proceedings of Latin American Theoretical Informatics Symposium, 2004, pp. 599--608.

[30] P. Flocchini, G. Prencipe, N. Santoro, P. Widmayer, Gathering of Asynchronous Robots with Limited Visibility, Theoretical Computer Science, 337 (2005) 147-168.

[31] P. Fraigniaud, L. Gasieniec, D. Kowalski, A. Pelc, Collective Tree Exploration, Networks, 48 (2006) 166--177.

[32] P. Glaus, Locating a Black Hole without the Knowledge of Incoming Link, in, 2009, pp. 128-138.

[33] N. Hanusse, D. Kavvadias, E. Kranakis, D. Krizanc, Memoryless Search Algorithms in a Network with Faulty Advice, in: Proceedings of IFIP International Conference on Theoretical Computer Science, 2002, pp. 206--216.

[34] F. Hohl, Time Limited Blackbox Security: Protecting Mobile Agents from Malicious Hosts, in: Proceedings of Conference on Mobile Agent Security, 1998, pp. 92--113.

[35] R. Klasing, E. Markou, T. Radzik, F. Sarracco, Hardness and Approximation Results for Black Hole Search in Arbitrary Graphs, Theoretical Computer Science, 384 (2007) 201--221.

[36] R. Klasing, E. Markou, T. Radzik, F. Sarracco, Approximation Bounds for Black Hole Search Problems, Networks, 52 (2008) 216--226.

[37] A. Kosowski, A. Navarra, C.M. Pinotti, Synchronization Helps Robots to Detect Black Holes in Directed Graphs, in: Proceedings of 13th International Conference on Principles of Distributed Systems, 2009, pp. 86-98.

[38] D. Kowalski, A. Pelc, Polynomial Deterministic Rendezvous in Arbitrary Graphs, in: Proceedings of 15th

International Symposium on Algorithms and Computation, 2004, pp. 644--656.

[39] R. Kr\`alovic, S. Mikl\`ik, Periodic data retrieval problem in rings containing a malicious host, in: Proceedings of 17th International Colloquium on Structural Information and Communication Complexity, 2010, pp. 156-167.

[40] C.E. Shannon, Presentation of a Maze-Solving Machine, in: Proceedings of 8th Conference of the Josiah Macy Jr. Found. (Cybernetics), 1951, pp. 173--180.

[41] W. Shi, Black hole search with tokens in interconnected networks, in: Proceedings of 11th International Symposium on Stabilization, Safety, and Security of Distributed Systems, 2009, pp. 670-682.

Παράρτημα - Προσομοίωση αλγορίθμου

Στο κεφάλαιο αυτό θα ασχοληθούμε με την προσομοίωση του αλγορίθμου **Three_agents_smallest_ID_waits** (που είδαμε στο κεφάλαιο 3.2.2) που αναζητά μία μαύρη τρύπα σε ένα ανώνυμο και συγχρονισμένο δακτύλιο, χρησιμοποιώντας τρεις πράκτορες. Το μοντέλο που αντιστοιχεί στον παραπάνω αλγόριθμο παρουσιάστηκε αναλυτικά στο κεφάλαιο 3.2

Η προσομοίωση παρέχει στον χρήστη τις παρακάτω δυνατότητες:

- ✓ Οπτικοποίηση των αποτελεσμάτων των αλγορίθμων αναζήτησης, γεγονός που εξυπηρετεί την κατανόηση του αλγορίθμου τόσο από το δημιουργό του, όσο και από τους χρήστες.
- ✓ Δυνατότητα επαλήθευσης των αλγορίθμων, δυνατότητα που προσδίδει τον άμεσο έλεγχο στο χρήστη.
- ✓ Υπολογισμό της πολυπλοκότητας, προσεγγίζει τη τάξη του αντιστοίχου μεγέθους ακόμα και σε περιπτώσει όπου ο υπολογισμός της αποτελεί κοπιαστική και χρονοβόρα διαδικασία.
- ✓ Δυνατότητα πειραματισμών, ελεύθερη τροποποίηση των ιδιοτήτων του αλγορίθμου ή του μοντέλου.
- ✓ Δεν απαιτείται η ύπαρξη κάποιου φυσικού δικτύου, καθώς είναι δυνατή η εφαρμογή του αλγορίθμου σε ένα εικονικό δίκτυο μηδενικού κόστους.

Μειονεκτήματα της μεθόδου προσομοίωσης.

- × Σημαντική, μα και αναγκαία, είναι η τμηματοποίηση του αλγορίθμου σε φάσεις εκτέλεσης για την υλοποίηση του συγχρονισμένου μοντέλου, καθώς είναι αναγκαία και η εκτέλεση συγκεκριμένου τμήματος του αλγορίθμου κάθε δεδομένη χρονική στιγμή. Η διαδικασία της τμηματοποίησης απαιτεί χρόνο για την μετατροπή του αλγορίθμου σε συμβατή μορφή.

Το πρόγραμμα της προσομοίωσης έχει υλοποιηθεί με βάση την ελεύθερη γλώσσα προγραμματισμού Java, της Sun. Επομένως η εκτέλεση της προσομοίωσης δεν αποτελεί επιπλέον κόστος για τον χρήστη.

Ο τρόπος λειτουργίας της προσομοίωσης είναι ο εξής:

Κάθε πράκτορας τρέχει τον ίδιο ντετερμινιστικό αλγόριθμο καλώντας τη συνάρτηση **AgentAlgorithm**, η οποία αποτελεί το βασικό αλγόριθμο. Το σύνολο του κώδικα που τρέχουν οι πράκτορες, ολοκληρώνεται με τις δύο βοηθητικές διαδικασίες, MoveTogether και cautiousWalk οι οποίες είναι υπεύθυνες για τη σωστή εξερεύνηση ενός δακτυλίου από τη στιγμή που δύο ή περισσότεροι πράκτορες

συναντηθούν. Οι υπόλοιπες συναρτήσεις που εμπεριέχονται στην προσομοίωση χωρίζονται σε τρεις βασικές κατηγορίες: **i)** στην πρώτη κατηγορία ανήκουν οι βασικές συναρτήσεις (Basic functions) οι οποίες διαχειρίζονται τις δομές δεδομένων τις οποίες χρησιμοποιεί η προσομοίωση (αντιγραφή, αρχικοποίηση πινάκων, κ.τ.λ.). **ii)** στη δεύτερη κατηγορία εντάσσονται οι συναρτήσεις που είναι υπεύθυνες για την υλοποίηση του μοντέλου (Simulation functions), τέτοιες είναι οι συναρτήσεις μέσω των οποίων οι κόμβοι παρέχουν πληροφορίες στους πράκτορες και άλλες δυνατότητες όπως η κίνηση των πρακτόρων στο δίκτυο. **iii)** τέλος στη τρίτη κατηγορία ανήκουν οι συναρτήσεις που είναι υπεύθυνες για την εκτύπωση και οπτικοποίηση των αποτελεσμάτων (Output functions), όπως για παράδειγμα η συνάρτηση που εμφανίζει το δακτύλιο με τις θέσεις των πρακτόρων. Συγκεντρωτικά όλες οι παραπάνω συναρτήσεις καλούνται από τη συνάρτηση της προσομοίωσης (simulationAll), που είναι υπεύθυνη για τη συγχρονισμένη εκτέλεση τους.

Η υλοποίηση του δακτυλίου βασίζεται σε έναν υποτιθέμενο μονοδιάστατο πίνακα, κάθε κελί του οποίου αποτελεί έναν από τους κόμβους που θα είχε το δίκτυο. Στην πραγματικότητα δεν αποθηκεύεται ολόκληρος ο πίνακας, παρά μόνο οι θέσεις που θα είχαν οι πράκτορες και η μαύρη τρύπα στον υποτιθέμενο αυτό πίνακα. Έτσι μειώνεται σημαντικά το μέγεθος των δεδομένων που πρέπει να αποθηκευτούν. Για παράδειγμα όπου το προς εξέταση δίκτυο είχε εκατό κόμβους, τότε θα έπρεπε να δεσμευθεί χώρος για έναν πίνακα εκατό θέσεων. Στην περίπτωση της προσομοίωσης αρκεί ένας πίνακας τεσσάρων θέσεων (τρεις θέσεις για τους τρεις πράκτορες και μία θέση για τη μαύρη τρύπα). Στο μοντέλο υποθέτουμε πως σε κάθε χρονική μονάδα έναν πράκτορας μπορεί να μετακινηθεί από τον κόμβου που βρίσκεται, μόνο σε έναν από τους δύο γείτονές του, συνεπώς οι πράκτορες στην εξομοίωση μπορούν να κινηθούν από ένα κελί του πίνακα, στο επόμενο ή στο προηγούμενο.

Όταν ένας πράκτορας επισκεφτεί μία μαύρη τρύπα, δεν μπορεί να φύγει από αυτή και στη συνέχεια η προσομοίωση δεν εκτελεί κώδικα που αφορά τον πράκτορα αυτό.

Για κάθε πράκτορα υπάρχει ένα σύνολο δεδομένων, στο οποίο αποθηκεύονται πληροφορίες όπως ο χάρτης του δικτύου, η ταυτότητα του πράκτορα, η αποστάσεις από τους άλλους πράκτορες και ένα πλήθος από σηματοδότες (flags) που αφορούν την κατάσταση στην οποία βρίσκεται κάθε πράκτορας. Οι σηματοδότες αυτοί αποθηκεύουν:

- τη φάση που βρίσκεται η εκτέλεση του αλγόριθμου, ώστε να συνεχίσει η εκτέλεση την επόμενη χρονική στιγμή
- τη φάση της διαδικασίας Move Together και Cautious Walk
- τα βήματα που πρέπει να κάνει ένας πράκτορας ή το χρόνο που πρέπει να περιμένει, με βάση τον αλγόριθμο που εκτελεί.

Για να επιτευχθεί ο συγχρονισμός των πρακτόρων υπάρχει μία παγκόσμια (global) μεταβλητή, με όνομα Clock, η οποία αποθηκεύει τους χτύπους του ρολογιού από την στιγμή της εκκίνησης της εξομοίωσης και είναι διαθέσιμη σε όλους τους πράκτορες. Άλλες πληροφορίες που παρέχονται γενικά, σε όλους τους πράκτορες, είναι το μέγεθος του δικτύου (n2), ο αριθμός των πρακτόρων (k) και ο αριθμός των μαύρων τρυπών (M).

Ο αλγόριθμος της προσομοίωσης επιτρέπει στο χρήστη να επιλέξει τα παρακάτω, τροποποιώντας τις μεταβλητές εισόδου:

- ✓ Να τροποποιήσει το μέγεθος του δακτυλίου μη αυτόματα (αλλάζοντας την τιμή της καθολικής μεταβλητής **n2**) ή αυτόματα με βάση μία τυχαία τιμή (αναθέτοντας τη τιμή 1 στην καθολική μεταβλητή **RandomRing**).

- ✓ Να επιλέξει τις αρχικές θέσεις των πρακτόρων και της μαύρης τρύπας μη αυτόματα (αλλάζοντας τις τιμές στον πίνακα **positions** που βρίσκεται στη συνάρτηση main), ή να επιλέξει την αυτόματη μέθοδο **RandAgents** που επιλέγει τυχαία τις θέσεις της μαύρης τρύπας και των πρακτόρων (αναθέτοντας την τιμή 1 στην καθολική μεταβλητή **RandomAgentsPositions**). Σε αυτή την περίπτωση οι πράκτορες συμφωνούν ως προς την κατεύθυνση του δακτυλίου και αναθέτουν τις κατάλληλες ταυτότητες στον εαυτούς με βάση τη μέθοδο **AgentLabeling**.

- ✓ Να διαλέξει μία από τις δύο υλοποιημένες μεθόδους εκτύπωσης των αποτελεσμάτων, οι οποίες εμφανίζουν τον δακτύλιο ως μονοδιάστατο πίνακα (**printRing1** επιλέγοντας τη τιμή 0 για την καθολική μεταβλητή **PrintMethod**) ή ως ελλειπτικό δακτύλιο (**printRing** επιλέγοντας τη τιμή 1 για την καθολική μεταβλητή **PrintMethod**).

- ✓ Να επιλέξει κατά τον χρόνο εκτέλεσης της προσομοίωσης, i) την εμφάνιση των στιγμιότυπων του δακτυλίου βήμα βήμα, ii) τη διακοπή της εκτέλεσης της προσομοίωσης iii) την αυτόματη εκτέλεση της προσομοίωσης και ταυτόχρονα την εμφάνιση όλων των στιγμιότυπων του δακτυλίου μέχρι το πέρας της εκτέλεσης του αλγορίθμου της αναζήτησης (μέσω των αντίστοιχων επιλογών 0, 1 και 2).

Ο χρήστης μπορεί επιπρόσθετα να επιλέξει να επιλέξει τα παρακάτω τροποποιώντας τον κώδικα της προσομοίωσης:

- Τον αριθμό των πρακτόρων

- Τον αριθμό των μαύρων τρυπών

- Τη ταχύτητα του κάθε πράκτορα για κάθε χτύπο του ρολογιού (για παράδειγμα να κινείται δύο βήματα αντί για ένα).

Το πρόγραμμα της εξομοίωσης δέχεται ως δεδομένα εισόδου και επιστρέφει ως έξοδο τα παρακάτω.

Είσοδος:

Το μέγεθος του δακτυλίου (n), τον αριθμό των πρακτόρων που συμμετέχουν στην αναζήτηση, τον αριθμό των μαύρων τρυπών (στην περίπτωσή μας μία), τις θέσεις των πρακτόρων (αν ο χρήστης επιλέξει να μη δημιουργηθούν αυτόματα)

Οι υπόλοιπές απαραίτητες πληροφορίες (όπως για παράδειγμα ο χάρτης με τις αρχικές θέσεις των πρακτόρων) δημιουργούνται από την προσομοίωσε με βάση τα παραπάνω δεδομένα εισόδου.

Έξοδος:

i) Αρχικά εμφανίζονται κάποιες βασικές πληροφορίες όπως οι θέσεις των πρακτόρων στο δακτύλιο και η κατεύθυνση στην οποία συμφώνησαν οι πράκτορες ως θετική (αν εμφανίζεται το σύμβολο + η κατεύθυνση είναι ίδια με το βέλος που εμφανίζεται πάνω από κάθε σχήμα του δακτυλίου) καθώς επίσης οι τυχαίες θέσεις των πρακτόρων και το τυχαίο μέγεθος του δακτυλίου, αν έχουν επιλεχθεί από το χρήστη οι αυτόματες διαδικασίες.

ii) Για κάθε χρονική μονάδα, εμφανίζεται ένα στιγμιότυπο του δακτυλίου με τις θέσεις των πρακτόρων και της μαύρης τρύπα.

iii) Κατά το τερματισμό του αλγορίθμου, οι επιζήσαντες πράκτορες αναφέρουν τη θέση της μαύρης τρύπας, εμφανίζοντας τον αριθμό του κόμβου που αυτή βρίσκεται (με βάση μία εικονική αρίθμηση των κόμβων), για την ευκολία του αναγνώστη.

Πιο αναλυτικά:

Αρχικά στον κώδικα της προσομοιώσεις δηλώνονται οι καθολικές μεταβλητές στις οποίες αποθηκεύονται τα δεδομένα εισόδου και οι μεταβλητές που πρέπει να έχουν πρόσβαση όλοι οι πράκτορες και όλες οι διαδικασίες.

Στη συνέχεια, στη μέθοδο main αρχικοποιούνται οι πίνακες των θέσεων (positions, tempPos) και των δεδομένων των πρακτόρων (AgData). Καθώς οι πράκτορες ουσιαστικά δεν τρέχουν παράλληλα, για να επιτύχουμε το σωστό συγχρονισμό των πρακτόρων και οι πράκτορες να έχουν τα δεδομένα της προηγούμενης χρονικής μονάδας αναλλοίωτα για τους υπολογισμούς τους, οι πράκτορες αποθηκεύουν τις αλλαγές τους στο προσωρινό πίνακα tempPos. Στο τέλος κάθε χρονικής μονάδας, ο πίνακας αυτός αντικαθιστά τα δεδομένα του πίνακα positions.

Σειρά έχει η συνάρτηση ελέγχου για λάθη *checkPositions*, κυρίως στις θέσεις των πρακτόρων και της μαύρης τρύπας.

Ακολουθούν οι διαδικασίες που παρέχουν τυχαίο μέγεθος δακτυλίου (*RandRingSize*) και τυχαίες τιμές στις θέσεις των πρακτόρων, της μαύρης τρύπας και των ταυτοτήτων των πρακτόρων (*RandAgents*).

Στη συνέχεια καλείται η συνάρτηση *findDirection* με την οποία οι πράκτορες συμφωνούν στον προσανατολισμό του δακτυλίου και εμφανίζεται η κατεύθυνση στην οποία συμφώνησαν οι πράκτορες.

Η εκτέλεση της main τελειώνει με τον έλεγχο ξανά στις θέσεις των πρακτόρων και της μαύρης τρύπας και αν όλα είναι σωστά καλείται η συνάρτηση της προσομοίωσης.

Στη συνάρτηση της προσομοίωσης ορίζεται ένα κατώφλι, το οποίο αποτρέπει την εκτέλεση της προσομοίωσης επ' άπειρον (κυρίως αν υπάρχει κάποιο λάθος στον αλγόριθμο ή στην συνθήκη τερματισμού). Στη συνέχεια οι πράκτορες ξεκινούν με την ίδια αρχική κατάσταση, ώστε να ξεκινήσουν συγχρονισμένα. Ακολουθούν, η δημιουργία του χάρτη με τις αρχικές θέσεις, ο υπολογισμός των αποστάσεων των πρακτόρων και ο υπολογισμός της μέγιστης απόστασης μεταξύ των d1 και d3 (χρήσιμο για τον αλγόριθμο των πρακτόρων). Εμφανίζονται κάποιες πληροφορίες στον χρήστη που περιγράφτηκαν και στο τμήμα που αναφέρεται στα δεδομένα εξόδου του αλγορίθμου.

Καταληκτικά είναι σημαντικό να αναφέρουμε τον ρόλο της συνάρτησης deadAgent η οποία ελέγχει αν ένας πράκτορας χάθηκε σε μία μαύρη τρύπα. Στην περίπτωση όπου κάποιος πράκτορας επισκέφτηκε μία μαύρη τρύπα (και άρα χάθηκε σε αυτήν) η διαδικασία αυτή αποτρέπει την εκτέλεση του αλγορίθμου του συγκεκριμένου πράκτορα.

Συνοψίζοντας, η εξομοίωση είναι μια γρήγορη, εύκολη και άμεση λύση για την υλοποίηση σωστών αλγορίθμων αναζήτησης μαύρων τρυπών σε ένα συγχρονισμένο και ανώνυμο δίκτυο. Δεν χρειάζεται κάποιο εξειδικευμένο υλικό ή λογισμικό, παρά μόνο η γλώσσα προγραμματισμού Java, η οποία είναι ελεύθερη.

Κώδικας προσομοίωσης:

```
/**
 * @(#)Simulation_3Agents.java
 *
 * @author chris papageorgakis
 * @version 1.00 2011/10/15
 */
import java.util.*;

public class Simulation_3Agents {

 public static int n2=8;        //number of nodes (n) at the ring
 public static int k=3;         //number of Agents
 public static int M=1;         //number of Black Holes

 public static int RandomRing=1;                  //If value is 1 the ring takes a random size, 0 ring size is the default.
 public static int RandomAgentsPositions=1; //If value is 1 the agents has different and random positions, 0 agents positions are the
default.
 public static int PrintMethod=1;           //Chooses the print Method, 1 prints as ring and 2 prints as array.

 public static long clock=0;    //time units, clock
```

```
public static int BHfound=0; //flag for stopping the execution
public static int flag=0;
public static int BH=0;              //positions at the positionArray.  Used at printRing
public static int A1=1;
public static int A2=2;
public static int A3=3;

public static int map[]=new int[k];   //all agent has the same map with agents ID as [x] and position as map[x]
public static int d[]=new int[k+1];   //distances: d[1] is d1... d[3] is d3, d[0] null but used later
public static int mapDirection=1;     //is the direction all agents agree

public static void main (String[] args) {
      int positions[]=new int[k+M];              //0 BH, 1 agent1, 2 agent2, 3 agent3
      int tempPos[]=new int[k+M];        //same as positions Array. Used for better synchronous model
      int AgData[][]=new int[k+M][9];    //x-> 0 null 1 Agent1 2 Agent2....
      InitArray(AgData);                 //y-> 0 phase of CWalk 1 MINid  2 flag for MINid 3 flag for wait, move or MoveTogether
                                         //    4 delay       5 phase      6 dir        7 MoveT phase       8 BHfound per agent
      InitArray(positions);
      //Use it for manual input of agents position
      positions[0]=1;              //BH    position
      positions[1]=4;              //Agent1 position
      positions[2]=0;              //Agent2 position
      positions[3]=5;              //Agent3 position
      //Warning: positions must be smaller than n
```

```
        if(checkPositions(positions)==-1){System.err.println("Error at start positions, n<4"); return;}
        if(RandomRing==1){ RandRingSize();  }
        if(RandomAgentsPositions==1){        RandAgents(positions);          }

        findDirection(positions);
        if(mapDirection>0){System.out.println("Direction is: +\n");}
        else{System.out.println("Direction is: -\n");}

        UpdatePos(positions, tempPos);              //inits the tempPos with positions Array
        if(checkPositions(positions)==0){
                simulationAll(positions, AgData, tempPos);  }
         else{
                System.err.println("Error at start positions");}
        }//end of main


//-------------------------Basic functions--------------------------------------------------------------------
public static void InitArray(int A[]){   //Initialazation of an array
        int i;
                for(i=0;i<A.length; i++ ){A[i]=0;}
}//end of InitArray

public static void InitArray(int A[][]){ //Initialazation of an 2D array
        int i,j;
                for(i=0;i<A.length; i++ ){
```

```
                    for(j=0;j<A[0].length; j++){
                        A[i][j]=0;
                    }
                }
}//end of InitArray2D

    public static void PrintArray(int A[]){
        int i;
        for(i=0;i<A.length;i++){
            System.out.printf("%d \t",A[i]);}
            System.out.printf("\n \n");
    } //end of PrintArray

    public static void swap(int A[], int x, int y){//swap the values A[x],A[y]
        int t=0;
        t=A[x];
        A[x]=A[y];
        A[y]=t;
    }

    public static int Uinput(){ //read users input from the ketboard
        int f=-1;
        Scanner s=new Scanner(System.in);
        try{
```

```
            f=s.nextInt();
        }catch( Exception e){}
        if(f==-1){System.err.println("Uinput returned -1");}
        return f;
}//end of Uinput

public static void UpdatePos(int A1[], int A2[]){    //copies the A1 Array elements to the A2 Array
//For most cases A1 will be the tempPos Array and A2 will be the positions Array
        int i;
        for(i=0;i<A1.length; i++ ){A2[i]=A1[i];}
}//end of UpdatePos

public static int Min(int A[],int start){
        //returns where is the min value of an array A
        //start is the start position of this function at the array(1 if we want to skip BHs position).
        int i,min,th;
        min=A[start];
        th=start;
        for(i=start+1;i<A.length; i++){
            if(A[i]<min){    min=A[i]; th=i; }
        }
        return  th;
}//end of Min

public static int Max(int A[],int start){
```

```
        //returns where is the max value of an array A
        //start is the start position of this function at the array.
        int i,max,th;
        max=A[start];
        th=start;
        for(i=start+1;i<A.length; i++){
                if(A[i]>max){    max=A[i]; th=i;}
        }
        return th;
  }//end of Max

  public static int MAX(int a, int b){//returns the maximum detween 2 integers
        if(a>b){return a;}
        else{return b;}
  }//end if MAX


//-------------------------Simulation functions-------------------------------------------------------------------

  public static int checkPositions(int P[]){//Checks if positions are smaller than the nubber of nodes.
  //If agents positions are right returns 0, if not returns 1. If n<4 return -1.
        if(n2<4){return -1;}
        int i;
                for(i=0;i<P.length; i++ ){
```

```
                        if(P[i]>=n2){
                                System.err.println("Position bigger than size of ring");
                                return 1;
                        }
                }
        return 0;
}//end of checkPositions

public static void RandRingSize(){
        Random diceRoller = new Random();
        int roll = diceRoller.nextInt(25);
        n2=roll+4;
        System.out.println("Size of ring (n) is:" + n2);
}//end of RandRingSize

public static void uniqueValues(int P[], int v, int k){ //Makes unique and random values. for example: IDs, Positions.
//input: The array to be filled with unique values, values between 0 and v, k size of array
//Uses the global n2. k is overriden.
        Random diceRoller = new Random();
        int roll=-1,i=0,j,flag=0;
        int A[]=new int[k+1];
        while(i<=k){
                if(i==0){A[i]=diceRoller.nextInt(v); i++;}
                else{j=0; flag=0;
```

```
                roll = diceRoller.nextInt(v);
                while(j<i){             //ensures that all agent will have different posiotions
                        if(roll==A[j]){flag=1; break;}
                        else{j++;}
                }
                if(flag==0){A[i]=roll; i++;}
            }
        }
        for(i=0;i<=k;i++){      P[i]=A[i];      }
}//end of uniqueValues

public static void AgentLabeling(int P[], int IDs[]){//label agents from random IDs to A1,...,An
        int i,th=-1,temp=-1;
        i=1;
                th=Min(IDs,i-1); //Min Agent ID
                swap( IDs,i-1,th );
                swap( P, i, th+1 );
        i=2;
                th=Max(IDs,i-1); //Min Agent ID
                swap( IDs, i-1 ,th );
                swap( P, i,th+1 );
}//end of AgentLabeling

public static void findDirection(int P[]){//Agents use this function to agree at the direction of ring
```

```
	//input is the map
	int i,fl=0;
	i=P[1];
	while(fl==0){
		if(i==P[2]){ mapDirection=1; fl=1; break; }
		if(i==P[3]){ mapDirection=-1; fl=1; break; }

		if(i==n2-1)	{ i=0;	}
		else		{ i=i+1;}
	}
}//end of findDirection

public static void RandAgents(int P[]){//This functions returns random agent positions and Ids
	int IDs[]=new int[k],i;
	System.out.println("Random positions:");
	uniqueValues(P, n2, k);			//makes unique and random positions. [0] is BH position!!
	for(i=0;i<=k;i++){
	 if(i==0){	System.out.printf("BH:%d \t",P[i]);			}
	 else{		System.out.printf("A%d:%d \t",i,P[i]);			}
	}
	System.out.println("\n");

	System.out.printf("Random IDs:\n \t\t");
	uniqueValues(IDs, 10000, k-1);  //makes unique and random IDs  [0] is first ID P[1] is the first agent position
```

```
        for(i=0;i<=k-1;i++){      System.out.printf("A%d:%d \t",i+1,P[i]);       }
        System.out.println("\n");

        AgentLabeling(P, IDs);
}//end of RandAgents

public static int distance(int Agent, int dir){//Finds the distance in a ring between one agent to the other agent CW
        //input is the agent starting measuring the distance
        //map direction
        int Agent1=Agent-1,Agent2=-1,d=-1,th;

        if(Agent>k-1){ Agent2=0;       }
        else{    Agent2=Agent;          }
        th=map[Agent1];
        d=0;
        while(map[Agent2]!=th){
                d=d+1;
                if(dir==1){
                        if(th>=n2-1){th=0;}
                        else{th=th+1;}
                }
                if(dir==-1){
                        if(th<=0){th=n2-1;}
                        else{th=th-1;}
```

```
                }
        }
        return d;
}//end of distance

public static void makeMap(int P[]){//Put the right values to Map Array
        //Each agent has in [Agent][0] his position and the other CW
         //it is a snapshot of the initial (position) configuration
        InitArray(map);
        int i=0;
        for(i=1;i<=k;i++){ //for all agents
                        map[i-1]=P[i];
        }
}//end of makeMap

public static void makeD(){//Find the distances between the agents: d1:d[1], d2:d[2] ,d3.. ,d[0] null used later
        //it the initial distance configuration
        InitArray(d);
        int i=-1;
        for(i=1;i<=k;i++){
                d[i]=distance(i,mapDirection);
        }
}//end of makeD

public static int deadAgent(int P[], int Agent){//return 1 if agents is dead, 0 otherwise
```

```
  //input is the agents row of positions Array
        if(P[BH]==P[Agent]){
                return 1;}
        return 0;
  }//end of deadAgent

  public static void moveCW( int P[],int pos){ //input array index of agent px 1 for A1
        if(P[pos]!=P[BH]){
                if(P[pos]==n2-1){       P[pos]=0;       }
                else{    P[pos]=P[pos]+1;       }
        }
  }//end of moveCW

public static void moveCCW( int P[],int pos){
        if(P[pos]!=P[BH]){
                if(P[pos]==0){  P[pos]=n2-1;   }
                else{    P[pos]=P[pos]-1;       }
        }
  }//end of moveCCW

  public static void move( int P[],int pos, int dir){
        //uses the above 2 functions, dir 1 is CW -1 CCW
        if(dir>0){moveCW(P,pos);}
        if(dir<0){moveCCW(P,pos);}
```

```
        if(dir!=1 && dir!=-1){System.out.println("Error:move called with dir="+dir);}
  }

public static int seeAnotherAgent( int P[],int pos){ //for n agents in the position array
        //return the number of different Agents at the given node(pos).
        //if all k agents are at the same node will return k-1
        //0 if the agent is alone
        int i=0,agents=0;
        while(i<P.length){
                if(i!=pos){
                        if(P[i]==P[pos]){agents=agents+1;}
                }
        i++;
        }
        return agents;
}//end of seeAnotherAgent

 public static int minID( int P[],int ID){//is the given ID the minimum ID of that node? 1 if yes 0 otherwise
        if(seeAnotherAgent( P,ID)==1 ){
                int i=0;
                while(i<P.length){
                        if(ID!=i){
                                if(P[ID]==P[i]){
                                        if(ID>i){return 0;}
```

```
                                }
                        }
                i++;
                }
                return 1;
        }
        return 0;
}//end of minID

public static int smallestID( int P[],int ID){//the min ID at a node, if there are more than one agent
  //return the min of the node for the givens agent node
  //-1 if agent is alone
  if(seeAnotherAgent( P,ID)>0){
        int i=0;
        int min=ID;
        while(i<P.length){
                if(ID!=i){
                        if(P[ID]==P[i]){
                                if(min>i){min=i;}
                        }
                }
        i++;
        }
        return min;
```

```
    }
  return -1;
  }//end of smallestID

  public static int isAgentWithYou( int P[],int posYou, int posAgent){
        //input: number of the agent asking, and the number of the agent looking for
        //1 if agents are together at the same node
        //0 if not
        int i=0;
                if(P[posYou]==P[posAgent]){return 1;}
                else{return 0;}
  }//end of isAgentWithYou


//-------------------------Output functions-------------------------------------------------------------------
//those two functions needed for the printRing, to display agents symbols
public static int printSymbol(int P[], int start, int end, String s, String s2){
        int flg=0,c=0, f=0;
        for(int i=start;i<=end; i++){
                if(P[BH]==i){System.out.print("M");
                System.out.print(s);}
                else{
                        int j=A1;
                        while(j<P.length){
                                if(P[j]==i){ System.out.printf("%c",'A'+j-1); flg=1; c=c+1;}
```

```
                        j++;
                        }
                        if(flg==0){System.out.print(" ");}
                        if(c>1){f=1;}
                        if( (end-start+1)>1 ){ System.out.print(s2); }
                }
                c=0;
        }
        return f;
}
public static void printRevSymbol(int P[], int start, int end, String s){
        for(int i=start;i>end; i-- ){
                if(P[BH]==i){System.out.print("M" + s);}
                else{
                        int j=A1;
                        while(j<P.length){
                                if(P[j]==i){ System.out.printf("%c",'A'+j-1); }
                        j++;
                        }
                        if( (start-end)>1 ){ System.out.print(s); }
                }
        }
}
```

```
public static void printRing1( int P[]){
        int i;
        System.out.print("Clock :" + clock +"   -- + --> \n");
        System.out.print("Node :\t");
        for(i=0;i<n2; i++ ){System.out.print(i+"\t\t");}
        System.out.print("\n");
        System.out.print("Agent:\t");
        for(i=0;i<n2; i++ ){
                if(P[BH]==i){System.out.print("M \t\t");}
                else{
                        int j=A1;
                        while(j<P.length){
                                if(P[j]==i){ System.out.printf("%c",'A'+j-1); }
                        j++;
                        }
                        System.out.print("\t\t");
                }
        }
        System.out.print("\n\n");
}//end of printRing

public static void printRing( int P[]){
        int i,n=n2,f=0;
        System.out.print("Clock:" + clock +"   -- + --> \n");
```

```
if(n2%2==0){
//1st line, node numbers
        System.out.print("\t--\t");
        for(i=1;i<=(n-2)/2; i++ ){System.out.print(i+"\t--\t");}

//2nd line Agent symbols at the ring (if any)
        System.out.print("\n /\t\t");
        if(n2==4){printSymbol(P, 1, (n-2)/2 ,"","\t\t");}
        else{printSymbol(P, 1, (n-2)/2 ,"\t\t","\t\t");}
        if(n2==4){System.out.print("\t\t");}
        System.out.print("\\ \n");

//3rd line
        System.out.print(0+" "); //prints 0  ...
        f=printSymbol(P, 0, 0 ,"","\t\t");
        for(i=1;i<=(n-2)/2; i++ ){
                System.out.print("\t\t");
        }
        if(f==0){
                System.out.print("\t  "); // prin to A or B.. or M and after 5
                printSymbol(P, ((n-2)/2)+1, ((n-2)/2)+1, "" ,"\t\t");
                System.out.print("  ");
                System.out.print( ((n-2)/2)+1 + "\n");  //prints   ...    5
        }else{
```

```
            System.out.print("  "); // prin to A or B.. or M and after 5
            printSymbol(P, ((n-2)/2)+1, ((n-2)/2)+1, "" ,"\t\t");
            System.out.print("  ");
            System.out.print( ((n-2)/2)+1 + "\n");  //prints   ...    5
}
}else{
//1st line, node numbers
            System.out.print("\t--\t");
            for(i=1;i<=(n-2)/2+1; i++ ){System.out.print(i+"\t--\t");}

//2nd line Agent symbols at the ring (if any)
            System.out.print("\n /\t\t");
            printSymbol(P, 1, (n-2)/2+1 ,"\t\t","\t\t");
            System.out.print("\\ \n");

//3rd line
            System.out.print(0+" ");//prints 0  ...
            printSymbol(P, 0, 0 ,"","\t\t");

            System.out.print("\n");  //prints   ...    half
}

//4th line
            System.out.print(" \\ \t\t");
```

```
                printRevSymbol(P, n-1, ((n-2)/2)+1,"\t\t");
                if(n2==4){System.out.print("\t\t");}
                System.out.print("/\n");

            //5th line
            System.out.print("\t--\t");
            for(i=n-1;i>((n-2)/2)+1; i-- ){System.out.print(i+"\t--\t");}  //prints n n-1 n-2
            System.out.print("\n\t\t");
            System.out.println("\n- - - - - - - - - - - - - - - - - - - - - - - - - -\n\n");
        }//end of printRing


//********************** Algorithm, Procedures and Simulation ************************************************
  public static void cautiousWalk(int P[],int Agent,int dir,int AgData[][]){
                switch (AgData[Agent][0]) {
   case 1:
        if(AgData[Agent][2]==0){//min ID initialization
            AgData[Agent][1]=smallestID(P,Agent); AgData[Agent][2]=1;
        }
        if(Agent==AgData[Agent][1]){ //min ID
            move( P,Agent, AgData[Agent][6]); //move to next node
        }
        else {/*Wait*/          }
        AgData[Agent][0]=2;
```

```
        break;
case 2:
        if(Agent==AgData[Agent][1]){ //min ID
                move( P,Agent, AgData[Agent][6]*-1); //return, opposite direction
        }
        else {/*Wait*/          }
        AgData[Agent][0]=3;
        break;
case 3:
        if(Agent==AgData[Agent][1]){ //min ID
                move( P,Agent, AgData[Agent][6]); //move to next node
        }
        else {
                if(AgData[Agent][1]!=-1 && isAgentWithYou(P,Agent,AgData[Agent][1])==1 ){
                        move( P,Agent, AgData[Agent][6]); //move to next node
                }
                else{
                        int tt=P[Agent]+dir; if(tt>=n2){tt=0;} if(tt<0){tt=n2-1;}
                        if(dir>0){System.out.println("Agent:"+Agent+"------>BH CW at node:"+ tt +"\n"); AgData[Agent][8]=1;}
                        else{System.out.println("Agent:"+Agent+"------>BH CCW at node:"+ tt +"\n"); AgData[Agent][8]=1;}}
        }
        AgData[Agent][2]=0;
        AgData[Agent][0]=1;
        break;
```

```
        }
}//end of cautiousWalk

 public static void MoveTogether(int P[], int Agent, int AgData[][], int segment){
        //P must be tempPos            segment is one of d1,d2 or d3 for exploration
                int tt;
                segment=segment-1;//distance converted to nodes!
                if(AgData[Agent][7]==0){//phase=0
                        if(segment==2){
                                AgData[Agent][7]=1;
                                AgData[Agent][0]=1;  //<----
                        }
                        if(segment==1){
                                /*BHfound=1;*/ AgData[Agent][8]=1;
                                tt=P[Agent]+AgData[Agent][6]; if(tt>=n2){tt=0;} if(tt<0){tt=n2-1;}
                                if(AgData[Agent][6]>0){System.out.println("Agent:"+Agent+"  BH at last node:"+ tt);}
                                if(AgData[Agent][6]<0){System.out.println("Agent:"+Agent+"  BH at last node:"+ tt);}
                        }
                        if(segment>2){
                                if(AgData[Agent][0]==3){
                                        AgData[Agent][4]=AgData[Agent][4]-1;
                                }
                                cautiousWalk( P,Agent ,AgData[Agent][6] , AgData);
                        }
```

```
        }
if(AgData[Agent][7]!=0){
        switch (AgData[Agent][0]){
        case 1:
                if(AgData[Agent][2]==0){ //min ID initialization
                        AgData[Agent][1]=smallestID(P,Agent); AgData[Agent][2]=1;
                }
                if(Agent==AgData[Agent][1]){ //min ID
                        move( P,Agent, AgData[Agent][6]); //move to next node
                }
                else {/*Wait*/          }
                AgData[Agent][0]=2;
                break;
                case 2:
                        if(Agent==AgData[Agent][1]){ //min ID
                move( P,Agent, AgData[Agent][6]*-1); //return, oposite direction
                }
                else {/*Wait*/          }
                AgData[Agent][0]=3;
                break;
        case 3:
                if(Agent==AgData[Agent][1]){ //min ID
                        System.out.println("-->The other agent will be updated\n");
                int e;
```

```
for(e=1;e<=k;e++){AgData[e][8]=1; //all agents know that the BH has been found (not the location of the BH)
        if(deadAgent(P,e)==0){
        tt=P[Agent]+AgData[Agent][6]+AgData[Agent][6]; //for right output
        if(tt>=n2){ if(tt==n2){tt=0;} if(tt>n2){tt=1;}    }
        if(tt<=-1){ if(tt==-1){tt=n2-1;} if(tt<-1){tt=n2-2;} }
        if(AgData[Agent][6]>0){System.out.println("Agent:"+e+"---------->BH CW at:"+ tt); AgData[Agent][8]=1;}
        else{     System.out.println("Agent:"+e+"---------->BH CCW at:"+ tt ); AgData[Agent][8]=1;}
        }
}
BHfound=1;

}
else {

if(AgData[Agent][1]!=-1 && isAgentWithYou(P,Agent,AgData[Agent][1])==1 ){
        move( P,Agent, AgData[Agent][6]); //move to next node
}
else{
        tt=P[Agent]+AgData[Agent][6]; if(tt>=n2){tt=0;} if(tt<0){tt=n2-1;}
        if(AgData[Agent][6]>0){System.out.println("Agent:"+Agent+"------>BH CW at node:"+ tt +"\n");
         System.out.println();
         AgData[Agent][8]=1;}
        else{     System.out.println("Agent:"+Agent+"------>BH CCW at node:"+ tt +"\n"); AgData[Agent][8]=1;}}
}
```

```
                    AgData[Agent][2]=0;
                    AgData[Agent][0]=1;
                    break;
                }
            }
        }//MoveTogether

public static void AgentAlgorithm(int P[],int map[], int d[],int Agent, long phase ,int AgData[][], int tempP[], int mapDir){
        //input: the current positions of agents for the changes of one agent at a time
        //map the initial positions of agents, AgentID the agent calls the algorithm
        //y-> 0 phase of CWalk 1 MINid        2 flag for MINid 3 flag for wait,move or MoveTogether
        //    4 delay                 5 Algphase    6 dir                       7 MoveT phase
        //delay: steps for wait, move or MoveTogether
        //clock will be the phase!
  if(AgData[Agent][3]==0){//no waiting flag exists
                if(AgData[Agent][5]<2){//phase<2
                switch (Agent){
                case 1:  //you are agent1:
                                if(AgData[Agent][5]<2){
                                        if(d[0]>0){
                                        AgData[Agent][3]=1;  //flag, wait d time units;
                                        AgData[Agent][4]=d[0]; //delay time d
                                        }
                                        AgData[Agent][5]=2;
```

```
                    }
                    break;
                case 2:  //you are agent2:
/*ph=0*/            if(AgData[Agent][5]==0){
                        if(d[1]>0){//move d1 steps at counterclockwise direction;
                            AgData[Agent][3]=2; //flag for move
                            AgData[Agent][4]=d[1];  //steps - delay
                            AgData[Agent][6]=-1*mapDir;   //direction
                            //AgData[Agent][6]=-1;   //direction
                            AgData[Agent][5]=1;   //phase
                        }
                    break;
                    }
/*ph=1*/            if(AgData[Agent][5]==1){
                        if(d[0]-d[1]>0){
                            AgData[Agent][3]=1;  //flag, wait d-d1 time units;
                            AgData[Agent][4]=d[0]-d[1];   //delay wait d-d1 time units;
                    }else{AgData[Agent][3]=0;}    //phase}  <---changed
                     AgData[Agent][5]=2;  //phase
                    }
                    break;
                case 3: // you are agent3
/*ph=0*/            if(AgData[Agent][5]==0){
                        if(d[3]>0){//move d3 steps at clockwise direction;
```

```
                                        AgData[Agent][3]=2; //flag for move
                                        AgData[Agent][4]=d[3];  //steps - delay
                                        AgData[Agent][6]=1*mapDir;   //direction
                                        //AgData[Agent][6]=1;   //direction
                                        AgData[Agent][5]=1;   //phase
                                }
                                break;
                        }
/*ph=1*/                if(AgData[Agent][5]==1){
                                if(d[0]-d[3]>0){//wait d-d3 time units;
                                        AgData[Agent][3]=1;  //flag, wait d-d1 time units;
                                        AgData[Agent][4]=d[0]-d[3];   //delay wait d-d1 time units;
                                }
                        AgData[Agent][5]=2;   //phase
                        }
                        break;
                }//end of switch1
                }
        if(AgData[Agent][5]==2 && AgData[Agent][3]==0){ //2<=phase<4
                if( seeAnotherAgent(P,Agent)==k-1 ){
                        if(d[2]-1==1){ System.out.println("Agent:" +Agent+ " BH at the one node of d2:"+ (map[A2-1]+1));
AgData[Agent][8]=1;; }
                        if(AgData[Agent][5]==2){ //phase is 2
                                if(d[1]<=d[3]){//move d1 steps CW
```

```
                                AgData[Agent][3]=2; //flag for move
                                        AgData[Agent][4]=d[1];  //steps - delay
                                        AgData[Agent][6]=1*mapDir;   //direction
                                        //AgData[Agent][6]=1;   //direction
                                        AgData[Agent][5]=3;   //phase
                                }else{//move d3 steps CCW
                                        AgData[Agent][3]=2; //flag for move
                                                AgData[Agent][4]=d[3];  //steps - delay
                                                AgData[Agent][6]=-1*mapDir;   //direction
                                                //AgData[Agent][6]=-1;   //direction
                                                AgData[Agent][5]=3;   //phase
                                }
                        }
                 }else{ AgData[Agent][5]=4; } //next phase, phase = 4
}
if(AgData[Agent][5]==3 && AgData[Agent][3]==0){//phase will be 3   <--
                        //call moveT with d2
                        AgData[Agent][3]=3; //flag for MoveT
                                AgData[Agent][4]=d[2];  //distance for exploration
                                AgData[Agent][5]=10;//phase 10 for catching errors
}
if(AgData[Agent][5]>=4 && AgData[Agent][3]==0 && AgData[Agent][5]<10){//phase will be 4   <--
                if(Agent==A1){
                        if( isAgentWithYou(P,Agent,A2)==1 ){
```

```
                AgData[Agent][6]=-1*mapDir;
                //AgData[Agent][6]=-1;
                //call moveT (with d3)
                AgData[Agent][3]=3; //flag for MoveT
                        AgData[Agent][4]=d[3];  //distance for exploration
                        AgData[Agent][5]=10;//phase 10 for catching errors
        }
        if( isAgentWithYou(P,Agent,A3)==1 ){
                //call moveT (with d1)
                AgData[Agent][6]=1*mapDir;
                //AgData[Agent][6]=1;
                AgData[Agent][3]=3; //flag for MoveT
                        AgData[Agent][4]=d[1];  //distance for exploration
                        AgData[Agent][5]=10;//phase 10 for catching errors
        }
}else{
                if(Agent==A2){//call moveT (with d3)
                        AgData[Agent][3]=3; //flag for MoveT
                                AgData[Agent][4]=d[3];  //distance for exploration
                                AgData[Agent][5]=10;//phase 10 for catching errors
                }
                if(Agent==A3){//call moveT (with d1)
                        AgData[Agent][3]=3; //flag for MoveT
                                AgData[Agent][4]=d[1];  //distance for exploration
```

```
                                        AgData[Agent][5]=10;//phase 10 for catching errors
                                }
                        }
                if(AgData[Agent][5]==10 && AgData[Agent][3]==0){ System.out.println("ERROR should has stopped");  }
        }
}//END of flag 0
        if(AgData[Agent][3]==1){//flag will be 1, wait for the delay
                if(AgData[Agent][4]>0) {  AgData[Agent][4]=AgData[Agent][4]-1; }
                if(AgData[Agent][4]==0){  AgData[Agent][3]=0;                           }
}//end of flag 1

if(AgData[Agent][3]==2){//flaf will be 2, move for the delay-steps
                if(AgData[Agent][4]>0){
                        AgData[Agent][4]=AgData[Agent][4]-1;
                        move(tempP,Agent,AgData[Agent][6]);
                }
                if(AgData[Agent][4]==0){ AgData[Agent][3]=0; }
        }//end of flag 2

        if(AgData[Agent][3]==3){//flag will be 3, call d times MoveTogether
                        if(AgData[Agent][4]>0){
                                //MoveT will decrease d!!
                                MoveTogether( tempP, Agent, AgData, AgData[Agent][4]);
                        }
```

```
                    if(AgData[Agent][4]==0){ AgData[Agent][3]=0;  }
        }//end of flag 3

}//end of Algorithm

 public static void simulationAll(int P[],int AgData[][], int tempPos[]){
        int i,t;
        int threshold=400;   //threshold is the maximum number of clock cycle: prevent simulation running for infinity
                for(t=1;t<k+1;t++){AgData[t][0]=1;} //for all agents the init state is the same: phse=1
                makeMap(P);
                makeD();
                d[0]=MAX(d[1],d[3]);

                //-----Simulation output data:-----------------------------------------------------
                System.out.println("Map distances: {Max(d1,d3), A->B, B->C, C->A }");
                PrintArray(d);
                System.out.println("Agents positions at the map: {A1 A2 A3}");
                PrintArray(map);

                System.out.println("Initial positions");
                if(PrintMethod==1){printRing(P);}else{printRing1(P);}
                //--------------------------------------------------------------------------------

        int contFlag=0;
```

```
while(BHfound==0 && contFlag!=1 && clock<threshold){//stops after #threshold steps!!!
	i=0;
	for(i=k;i>0;){
		 if(deadAgent(P,i)==0){
			AgentAlgorithm(P, map, d, i    , clock ,AgData, tempPos, mapDirection);
		 }
		 i--;
	}
	BHfound=1;
	for(i=1;i<=k;i++){
		if(deadAgent(P,i)==0){
			if(AgData[i][8]==0){ BHfound=0; }
		}
	}
	if(BHfound!=1){
		UpdatePos(tempPos, P);
		clock=clock+1;
		if(PrintMethod==1){printRing(P);}else{printRing1(P);}
	}
	if(contFlag==0 ){
		System.out.printf("continue?? ( Yes(0)/No(1)/Auto(2)  )");
		contFlag=Uinput();
	}
}
```

```
    }//end of simulationAll
}//end of Simulation_3Agents
```