Πανεπιστήμιο Πατρών

Τμήμα Μηχανικών Η/Υ και Πληροφορικής

1ο Σύνολο Θεωρητικών – Προγραμματιστικών Ασκήσεων

Κωνσταντίνος Αναστασόπουλος

1093320

4ο έτος

Table of Contents

Θεωρητικές Ασκήσεις

Άσκηση 1

Έστω ότι η πιθανότητα επιλογής ενός κόμβου u να μπει στο M, όπου M το σύνολο των υποψήφιων να μπουν στο MIS κόμβων, είναι

$$p = \frac{1}{ld(u)}$$

Τότε όπως και στο GIAN Lecture Notes: Network Algorithms (Schmid), αντικαθιστώντας όπου 2, l προκύπτει

$$P[u \neg\in MIS | u \in M] = P[\exists w \in H(u), w \in M | v \in M]$$

$$= P[\exists w \in H(u), w \in M]$$

$$\leq \sum_{w \in H(v)} P[w \in M]$$

$$= \sum_{w \in H(v)} \frac{1}{ld(w)}$$

$$\leq \sum_{w \in H(v)} \frac{1}{ld(u)}$$

$$\leq \frac{d(u)}{ld(u)}$$

$$= \frac{1}{l}$$

Όπου H(u), οι γείτονες του u που έχουν μεγαλύτερο βαθμό, ή ίδιο βαθμό και μεγαλύτερο identifier.

Η συμπληρωματική πιθανότητα είναι

$$P[u \epsilon MIS | u \epsilon M] \geq 1 - \frac{1}{l} = \frac{l-1}{l}$$

Χρησιμοποιώντας τα παραπάνω υπολογίζεται

$$P[u \in MIS] = P[u \epsilon MIS | u \epsilon M] P[u \in M]$$

$$= \frac{l-1}{l} \frac{1}{ld(u)}$$

$$\geq \frac{l-1}{l^2 d(u)}$$

Στο GIAN Lecture Notes: Network Algorithms (Schmid) καλός κόμβος θεωρείται κάποιος κόμβος u για τον οποίο ισχύει

$$\sum_{w \in H(u)} \frac{1}{2d(u)} \geq \frac{1}{6}$$

Προφανώς ισχύει

$$\sum_{w \in H(u)} \frac{1}{2d(u)} \geq \frac{1}{3 \times 2}$$

Το 2 στο παραπάνω κλάσμα έχει το ρόλο του l στην ειδική περίπτωση.

Αυτό ισχύει επειδή ένας κόμβος είναι καλός αν έχει πολλούς μικρού βαθμού γείτονες. Από την παραπάνω ανισότητα θέλουμε η πιθανότητα να επιλεγεί γείτονας να είναι μεγαλύτερη ή ίση με ένα έκτο. Προφανώς για να είναι μεγάλη η πιθανότητα να ισχύει αυτό στη γενική περίπτωση, πρέπει η σταθερά που επιλέγουμε (στην παραπάνω περίπτωση, ένα έκτο) να εξαρτάται από τη σταθερά l, εφόσον αυτή επηρεάζει την πιθανότητα επιλογής.

Στη γενική περίπτωση λοιπόν, ένας κόμβος είναι καλός αν

$$\sum_{w \in H(u)} \frac{1}{ld(w)} \geq \frac{1}{3l}$$

Αυτός είναι ο γενικευμένος ορισμός του καλού κόμβου. Η επιλογή του l επηρεάζει τον βαθμό των γειτόνων του u για τους οποίους ισχύει τετριμμένα το λήμμα 8.5 του GIAN Lecture Notes: Network Algorithms (Schmid). Στην περίπτωση που l = 2, το λήμμα ισχύει τετριμμένα για τουλάχιστον ένα γείτονα βαθμού 1 ή 2. Επειδή το l επηρεάζει το βαθμό που είναι ακέραιος, η επιλογή του ως αριθμό με δεκαδικά ψηφία δεν έχει νόημα. Για l = 2,5 η απόδειξη είναι ίδια με αυτήν για l = 2. Έχει λοιπόν μόνο νόημα να ασχοληθούμε με ακέραια l μεγαλύτερα ή ίσα με 1. Επειδή με l = 1 ασχολούμαστε παρακάτω και με l = 2 η απόδειξη είναι στο GIAN Lecture Notes: Network Algorithms (Schmid), η παρακάτω απόδειξη αφορά l μεγαλύτερα ή ίσα του 3. Για τέτοιο l ισχύει τετριμμένα το γενικευμένο λήμμα 8.5 που αποδεικνύω παρακάτω για γείτονες βαθμού 1 και 2.

Για γείτονες του u (N(u)), βαθμού 3 και πάνω ισχύει

$$\forall w \in N(u) : \frac{1}{ld(w)} \leq \frac{1}{3l}$$

Εφόσον δείξαμε ότι για καλούς κόμβους ισχύει

$$\sum_{w \in H(u)} \frac{1}{ld(w)} \geq \frac{1}{3l}$$

Τότε υπάρχει S υποσύνολο του N(u) τέτοιο ώστε

$$\frac{1}{3l} \leq \sum_{w \in S} \frac{1}{ld(w)} \leq \frac{2}{3l}$$

Έστω γεγονός R : u αφαιρείται στο βήμα 3 του αλγορίθμου.

Τότε η απόδειξη ακολουθεί τη λογική της αντίστοιχης στο GIAN Lecture Notes: Network Algorithms (Schmid).

$$P[R] \geq P[\exists u \in S, v \in MIS]$$

$$\geq \sum_{u \in S} P[u \in MIS] \; - \sum_{u,w \in S, u \neq w} P[u \in MIS, w \in MIS]$$

$$\geq \sum_{u \in S} P[u \in MIS] \; - \sum_{u \in S} \sum_{w \in S} P[u \in M] P[w \in M]$$

$$\geq \sum_{u \in S} \frac{l-1}{l^2 d(u)} - \sum_{u \in S} \sum_{w \in S} \frac{1}{l^2 d(u) d(w)}$$

$$\geq \sum_{u \in S} \frac{1}{ld(u)} \left(\frac{l-1}{l} - \sum_{w \in S} \frac{1}{ld(w)}\right)$$

$$\geq \frac{1}{3l} \left(\frac{l-1}{l} - \frac{2}{3l}\right)$$

$$= \frac{3l-5}{9l^2}$$

Η τελευταία ανισότητα ισχύει επειδή

$$\frac{1}{3l} \leq \sum_{w \in S} \frac{1}{ld(w)} \leq \frac{2}{3l}$$

Το αποτέλεσμα αποτελεί το κάτω φράγμα της πιθανότητας ο κόμβος u να αφαιρεθεί και αυτό είναι η γενίκευση του λήμματος 8.5 του GIAN Lecture Notes: Network Algorithms (Schmid).

Όσον αφορά το λήμμα 8.6, ισχύει και στη γενική περίπτωση. Πάντα τουλάχιστον οι μισές ακμές είναι καλές, όπου καλή ακμή λέμε κάποια που συνδέεται με τουλάχιστον έναν καλό κόμβο. Για την απόδειξη κατασκευάζεται βοηθητικό γράφημα με ακμές προς τον κόμβο μεγαλύτερου βαθμού. Αν οι βαθμοί είναι ίδιοι τότε η ακμή κατευθύνεται προς τον κόμβο με το μεγαλύτερο identifier. Επίσης, χρησιμοποιείται το λήμμα 8.7 που στην περίπτωσή μας

γενικεύεται ως εξής: Ένας κακός κόμβος έχει έξω βαθμό τουλάχιστον δύο φορές μεγαλύτερο από τον έσω βαθμό.

Για την απόδειξη αυτού χρησιμοποιείται εις άτοπο απαγωγή. Έστω ένας κακός κόμβος u που δεν έχει έξω βαθμό τουλάχιστον διπλάσιο του έσω βαθμού. Δηλαδή, τουλάχιστον ένα τρίτο των γειτονικών κόμβων έχουν βαθμό το πολύ d(u). Τότε:

$$\sum_{w \in N(u)} \frac{1}{ld(w)} \geq \sum_{w \in S} \frac{1}{ld(w)} \geq \sum_{w \in S} \frac{1}{ld(u)} \geq \frac{d(u)}{3} \frac{1}{ld(u)} = \frac{1}{3l}$$

Άρα ο κόμβος είναι καλός. Άτοπο. Άρα αποδείχθηκε και το γενικευμένο λήμμα 8.7.

Σύμφωνα με το παραπάνω λήμμα ο αριθμός των ακμών που κατευθύνονται σε κακούς κόμβους είναι το πολύ μισός του αριθμού των ακμών. Κατά συνέπεια, τουλάχιστον οι μισές ακμές θα κατευθύνονται προς καλούς κόμβους. Προφανώς αυτές οι ακμές είναι καλές. Αποδείχθηκε και το λήμμα 8.6, που είναι το ίδιο με το αντίστοιχο του GIAN Lecture Notes: Network Algorithms (Schmid).

Ακολουθεί απόδειξη γενίκευσης του Θεωρήματος 8.8, όπου υποστηρίζει πως ο παραπάνω αλγόριθμος τερματίζει σε χρόνο O(logn).

Από τα παραπάνω λήμματα, τουλάχιστον οι μισές ακμές θα αφαιρεθούν με πιθανότητα

$$\frac{3l-5}{18l^2}$$

Έστω R ο αριθμός των ακμών που θα αφαιρεθούν και m ο αριθμός των ακμών στην αρχή μιας φάσης. Χρησιμοποιώντας τη γραμμικότητα της μέσης τιμής

$$E[R] \geq \frac{(3l-5)m}{18l^2}$$

Έστω επίσης

$$p \coloneqq P[R \leq E[R/2]]$$

Τότε

$$E[R] = \sum_{r} P[R = r]r \leq \frac{pE[R]}{2} + (1-p)m$$

Λύνοντας για p παίρνουμε

$$p \leq \frac{m - E[R]}{m - \frac{E[R]}{2}} < \frac{m - E[R]}{m} \leq \frac{m - \frac{(3l-5)m}{18l^2}}{m} = 1 - \frac{3l-5}{18l^2}$$

Κατά συνέπεια, με πιθανότητα τουλάχιστον

$$\frac{3l-5}{18l^2}$$

Ο αριθμός κόμβων που θα διαγραφούν σε μια φάση είναι τουλάχιστον

$$\frac{3l-5}{36l^2}$$

Συνεπώς, αναμένεται σε O(logm) φάσεις ο αλγόριθμος να τερματίσει. Επειδή όμως ισχύει

$$m \leq n^2$$

Τελικά O(logm) = O(logn) και ο αλγόριθμος τερματίζει σε χρόνο O(logn).

Για την ειδική περίπτωση που l = 1 ισχύει

$$P[u \in M] = \frac{1}{d(u)}$$

Και

$$P[u \in MIS | u \in M] \geq \frac{l-1}{l} = 0$$

Το οποίο δε μας δίνει κάποια πληροφορία και η απόδειξη δεν μπορεί να συνεχίσει με αντίστοιχο τρόπο.

Άσκηση 2

Για να αποδείξουμε πως ο αλγόριθμος στη σελίδα 119 του GIAN Lecture Notes: Network Algorithms (Schmid) είναι self-stabilizing αρκεί να δείξουμε πως για αυτόν ισχύουν δύο κριτήρια, η κλειστότητα και η σύγκλιση.

Παρακάτω χρησιμοποιώ τον όρο κατάσταση και διαμόρφωση με την ίδια σημασία.

Κλειστότητα ισχύει γιατί από κάθε νόμιμη κατάσταση, ο αλγόριθμος μπορεί υπό φυσιολογικές συνθήκες να μεταβεί μόνο σε νόμιμη κατάσταση. Σε μία νόμιμη κατάσταση το σύνολο που έχει δημιουργηθεί είναι μη επεκτάσιμο και ανεξάρτητο. Αν παραβιαζόταν κάποιο από τα δύο κριτήρια, δηλαδή η ανεξαρτησία ή η μη επεκτασιμότητα, τότε θα μεταβαίναμε σε μη νόμιμη κατάσταση. Η ανεξαρτησία διατηρείται, αφού αν ένας κόμβος είναι στο MIS, κανένας γείτονας με υψηλότερο identifier δεν είναι στο MIS. Για να φύγει κόμβος από το MIS πρέπει να υπάρχει γείτονας με υψηλότερο identifier αλλά αυτό δεν ισχύει γιατί τότε ο κόμβος δε θα ήταν στο MIS. Συνεπώς, το πολύ ένας κόμβος ανά γειτονιά είναι στο MIS κάθε φορά, άρα η ανεξαρτησία διατηρείται. Η μη επεκτασιμότητα επίσης δεν παραβιάζεται αφού κόμβοι που δεν είναι στο MIS έχουν γείτονες σε αυτό. Οι γείτονες για να είναι στο MIS έχουν υψηλότερα identifiers. Συνεπώς, οι κόμβοι δεν μπορούν να μπουν στο MIS. Η μη επεκτασιμότητα, λοιπόν, διατηρείται. Αφού από κάθε νόμιμη κατάσταση ο αλγόριθμος δεν μπορεί να μεταβεί σε μη νόμιμη, σε μία δηλαδή για την οποία δεν ισχύει μη επεκτασιμότητα ή ανεξαρτησία, για τον αλγόριθμο ισχύει το κριτήριο της κλειστότητας.

Για την απόδειξη της σύγκλισης αρκεί να αποδειχθεί πως υπάρχει συνάρτηση δυναμικού VF(c), όπου c κάποια διαμόρφωση, όπου συγκλίνει σε κάθε βήμα του αλγορίθμου και κάτω ή πάνω από συγκεκριμένη σταθερά ο αλγόριθμος έχει φτάσει σε ασφαλή κατάσταση.

Έστω VF(c) ο αριθμός των κόμβων που θέτουν τον αλγόριθμο σε μη νόμιμη κατάσταση. Τέτοιοι είναι κόμβοι που είναι στο MIS, ενώ έχουν γείτονες με υψηλότερα identifiers ή κόμβοι που έχουν το υψηλότερο identifier στη γειτονιά τους αλλά δεν είναι στο MIS. Στην πρώτη περίπτωση, μέσω του βήματος 2 του αλγορίθμου, ο κόμβος θα βγει από το MIS και θα μειώσει την τιμή της συνάρτησης κατά 1. Στην δεύτερη περίπτωση, μέσω του βήματος 3 του αλγορίθμου, ο κόμβος θα μπει στο MIS και θα μειώσει την τιμή της συνάρτησης κατά 1. Σε κάθε βήμα η συνάρτηση θα φθίνει και όταν ο αλγόριθμος βρεθεί σε νόμιμη κατάσταση, η τιμή της συνάρτησης θα είναι 0. Αυτή είναι και η τιμή κατωφλίου, στην οποία ο αλγόριθμος είναι σε ασφαλή κατάσταση. Για τον αλγόριθμο λοιπόν ισχύει το κριτήριο της σύγκλισης.

Αφού ισχύουν τα κριτήρια της σύγκλισης και της κλειστότητας, ο αλγόριθμος είναι self-stabilizing.

Ο αλγόριθμος επίσης είναι silent, εφόσον είναι self-stabilizing και οι εφόσον φτάσει σε νόμιμη κατάσταση, λόγω της κλειστότητας δε θα φύγει από νόμιμη κατάσταση και τα nodeID, MIS or not MIS μηνύματα δεν θα ανανεώνονται.

Προγραμματιστικές Ασκήσεις

Άσκηση 1

Ο τρόπος υλοποίησης των MIS βασίζεται στις διαφάνειες του μαθήματος. (Τσίχλας, 2η Διάλεξη: Μη-Επεκτάσιμο Ανεξάρτητο Σύνολο)

Simple MIS

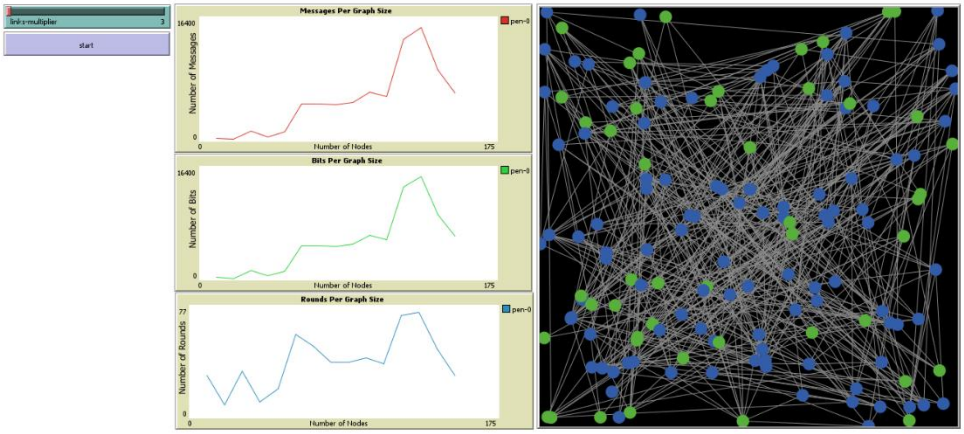
Κώδικας

```
; A series of assumptions have been made:
; 1. Each node sends a 1-bit message to all neighbors when it becomes a candidate or joins the
MIS
; 2. The message has a single type bit ("I'm a candidate" or "I joined")
; 3. To simulate sending the messages, we increment the message and bit counters
; 4. Ticks serve as rounds

; Global variables used throughout the simulation
globals [
  number-of-messages        ; Total number of messages sent in a run
  number-of-bits            ; Total number of bits sent in a run
  bits-per-message          ; Number of bits required to send a message (1)
  max-degree                ; Maximum degree (Δ)
  mis                       ; Nodes that are already part of the MIS
]

; Each turtle keeps track of its candidacy in the current round
turtles-own [
  candidate?                ; Whether the node is currently a candidate
]

; Main procedure to run the MIS algorithm for different graph sizes
to start
  clear-all                 ; Clear the world
  reset-ticks               ; Reset tick counter to 0

  ; Loop over different network sizes
  foreach [10 20 30 40 50 60 70 80 90 100 110 120 130 140 150] [ number-of-nodes ->
    set number-of-messages 0
    set number-of-bits 0

    run-algorithm number-of-nodes  ; Run the algorithm for n nodes

    ; Plot total messages used
    set-current-plot "Messages Per Graph Size"
    plotxy number-of-nodes number-of-messages

    ; Plot total bits used
    set-current-plot "Bits Per Graph Size"
    plotxy number-of-nodes number-of-bits

    ; Plot number of rounds (ticks)
    set-current-plot "Rounds Per Graph Size"
    plotxy number-of-nodes ticks
  ]
end
```

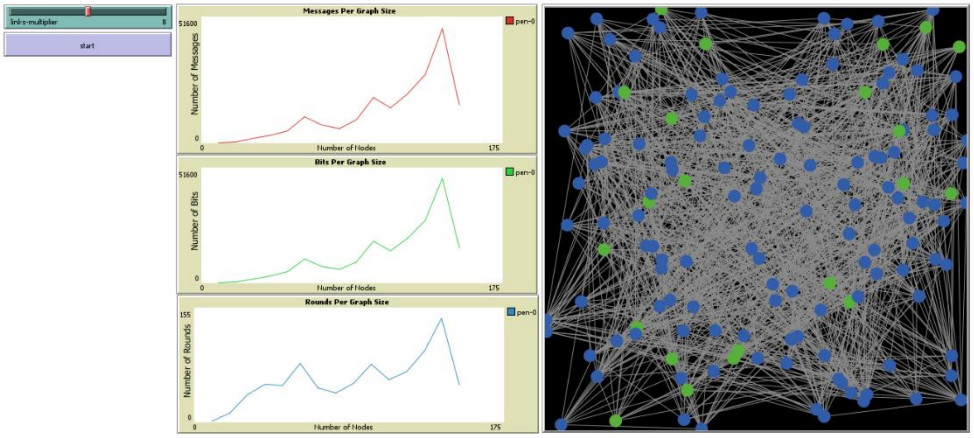
```
; Sets up a random graph and runs the MIS protocol on it
to run-algorithm [num-nodes]
  set bits-per-message 1         ; Only one bit per message now

  set mis []                     ; Initialize the MIS list
  set max-degree 0

  let max-links num-nodes * (num-nodes - 1) / 2
  let number-of-links links-multiplier * num-nodes

  clear-turtles
  clear-links

  ; Create turtles representing nodes
  set-default-shape turtles "circle"
  create-turtles num-nodes [
    set color blue
    set size max (list 1 (10 / sqrt num-nodes)) ; Visual size adjustment
    setxy random-xcor random-ycor
    set candidate? false                        ; Initialize candidacy
  ]

  ; Randomly add links until desired density
  while [ count links < min list number-of-links max-links ] [
    let a one-of turtles
    let b one-of other turtles with [not link-neighbor? a]
    if a != b [
      ask a [ create-link-with b ]
    ]
    set max-degree max [ count link-neighbors ] of turtles
  ]

  ; Run synchronous MIS rounds until termination
  reset-ticks
  while [ not check-termination ] [
    run-instance
    tick
  ]
end

; Executes one full round of the MIS algorithm
to run-instance
  select-candidates
  send-candidate-messages
  resolve-conflicts
  join-mis
  send-mis-join-messages
end

; Elect candidates with probability 1 / Δ, skip if a neighbor is already in MIS
to select-candidates
  ask turtles [
    set candidate? false                    ; Reset candidacy for this round
    if random-float 1 < 1 / max-degree [
      if not any? link-neighbors with [member? self mis] [
        set candidate? true
      ]
    ]
  ]
end

; Each candidate sends a 1-bit message to all its neighbors
to send-candidate-messages
  ask turtles with [candidate?] [
    let m count link-neighbors
    set number-of-messages number-of-messages + m
    set number-of-bits number-of-bits + m * bits-per-message
  ]
end
```

```
; Resolve conflicts – if two candidates are neighbors, both drop out
to resolve-conflicts
  ask turtles with [candidate?] [
    if any? link-neighbors with [candidate?] [
      set candidate? false
    ]
  ]
end

; Remaining candidates join the MIS
to join-mis
  ask turtles with [candidate?] [
    set color green                          ; Mark visually as MIS member
    set mis lput self mis                    ; Add to MIS list
    set candidate? false                     ; Reset candidacy
  ]
end

; MIS members broadcast their new status with a 1-bit message
to send-mis-join-messages
  ask turtles with [member? self mis] [
    let m count link-neighbors
    set number-of-messages number-of-messages + m
    set number-of-bits number-of-bits + m * bits-per-message
  ]
end

; Checks if all nodes are either in the MIS or have a neighbor in the MIS
to-report check-termination
  report all? turtles [
    member? self mis or any? link-neighbors with [member? self mis]
  ]
end
```

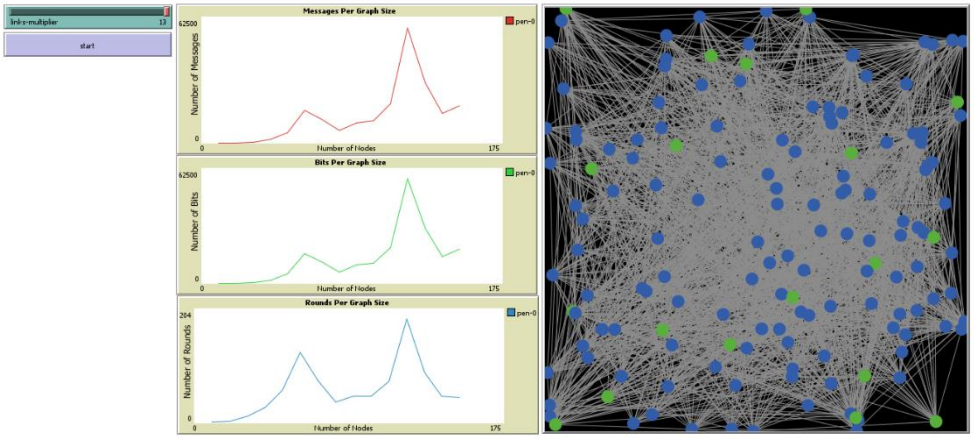
Διαγράμματα



Simple MIS for 3n links.



Simple MIS for 8n links.



Simple MIS for 15n links.

Σχόλια

Όπως φαίνεται στα διαγράμματα, η καμπύλη των μηνυμάτων και αυτή των bits είναι πανομοιότυπες καθώς κάθε μήνυμα αποτελείται από ένα bit. Η καμπύλη των γύρων μοιάζει επίσης με τα άλλα δύο γραφήματα, που είναι λογικό, καθώς σε περισσότερους γύρους στέλνονται περισσότερα μηνύματα, χωρίς αυτό βέβαια να σημαίνει πως είναι γραμμικά εξαρτημένα τα δύο μεγέθη. Όπως είναι αναμενόμενο, περισσότεροι κόμβοι, αυξάνουν τον αριθμό των μηνυμάτων και των bits που στέλνονται γιατί αυξάνονται οι γείτονες και κατά συνέπεια και τη ανάγκη για επικοινωνία. Ο αριθμός των γύρων αυξάνεται αρκετά με την αύξηση του links multiplier από 3 σε 8, αλλά δεν παρουσιάζεται μεγάλη διαφορά με την αύξηση από 8 σε 15. Επίσης τα διαγράμματα με την αύξηση του αριθμού των κόμβων, φαίνεται να αυξάνουν σταδιακά και τις τρεις υπό μελέτη μετρικές, αλλά με μη μονότονο τρόπο. Τα δύο τελευταία φαινόμενα αποδίδονται στην τυχαιοκρατική συμπεριφορά του αλγορίθμου.

Luby's MIS

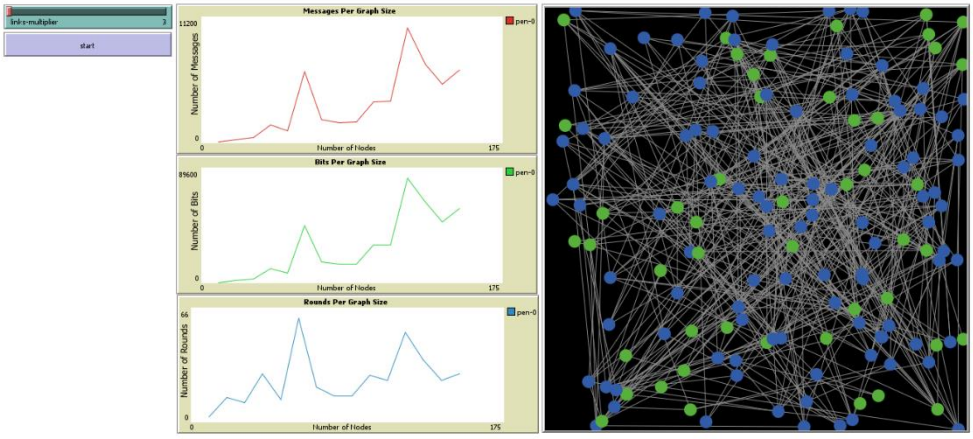
Κώδικας

```
; A series of assumptions have been made:
; 1. Each node sends a message to all neighbors when it becomes candidate or joins the MIS
; 2. The message has an id of ceil(log2(n)) bits and a message type flag of 1 bit
; 3. To simulate sending the messages, I just increment the message and bit counters
; 4. Ticks serve the purpose of rounds

; Global variables used throughout the simulation
globals [
  number-of-messages       ; Total number of messages sent in a run
  number-of-bits           ; Total number of bits sent in a run
  bits-per-message         ; Number of bits required to send a message
  mis                      ; Nodes that are already part of the MIS
]

; Turtle variables
turtles-own [
  degree                   ; Degree of each node
  candidate?               ; Whether the turtle is a candidate in the current round
]

; Main procedure to run the MIS algorithm for different graph sizes
to start
  clear-all                 ; Clear the world
  reset-ticks               ; Reset tick counter to 0

  ; Loop over different network sizes
  foreach [10 20 30 40 50 60 70 80 90 100 110 120 130 140 150] [ number-of-nodes ->
    set number-of-messages 0
    set number-of-bits 0

    run-algorithm number-of-nodes ; Run the algorithm for n nodes

    ; Plot the total messages used for this network size
    set-current-plot "Messages Per Graph Size"
    plotxy number-of-nodes number-of-messages

    ; Plot the total bits used for this network size
    set-current-plot "Bits Per Graph Size"
    plotxy number-of-nodes number-of-bits

    ; Plot the number of rounds (ticks) needed to complete MIS
    set-current-plot "Rounds Per Graph Size"
    plotxy number-of-nodes ticks
  ]
end

; Sets up a random graph and runs the MIS protocol on it
to run-algorithm [num-nodes]
  let id-bits ceiling log num-nodes 2
  set bits-per-message id-bits + 1  ; Each message carries one ID and a flag bit

  ; Initialize the MIS list
  set mis []

  let max-links num-nodes * (num-nodes - 1) / 2
  let number-of-links links-multiplier * num-nodes

  clear-turtles
  clear-links

  ; Create turtles representing nodes
  set-default-shape turtles "circle"
```

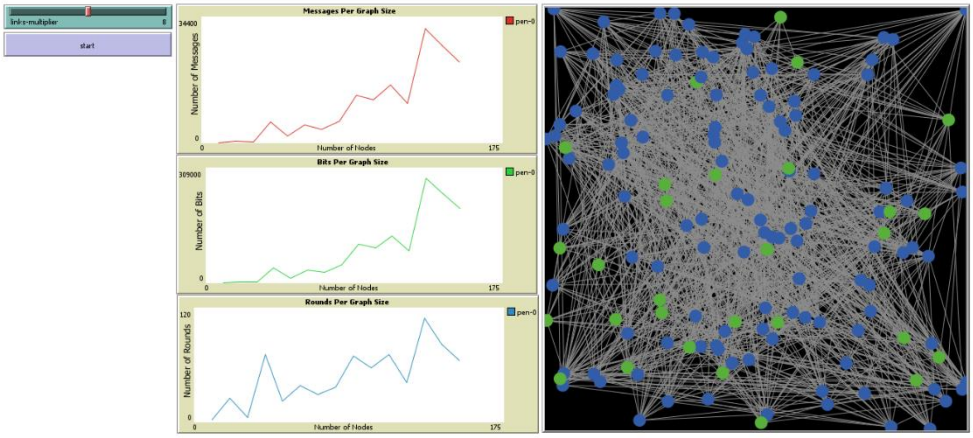
```
  create-turtles num-nodes [
    set color blue
    set size max (list 1 (10 / sqrt num-nodes)) ; Adjust size based on density
    setxy random-xcor random-ycor
    set degree 0
    set candidate? false                        ; Initialize candidacy state
  ]

  ; Randomly add links between turtles to form a graph of desired density
  while [ count links < min list number-of-links max-links ] [
    let a one-of turtles
    let b one-of other turtles with [not link-neighbor? a]
    if a != b [
      ask a [ create-link-with b ]
    ]
  ]

  ; Compute the degree of each turtle
  ask turtles [
    set degree count link-neighbors
  ]

  ; Run synchronous MIS rounds until termination
  reset-ticks
  while [ not check-termination ] [
    run-instance
    tick
  ]
end

; Executes one full round of the MIS algorithm
to run-instance
  select-candidates
  send-candidate-messages
  resolve-conflicts
  join-mis
  send-mis-join-messages
end

; Elect candidates with probability 1 / (2 * degree), skip if a neighbor is in MIS
to select-candidates
  ask turtles [
    set candidate? false  ; Reset candidacy each round
    if degree = 0 [
      set candidate? true
    ]
    if degree != 0 [
      if random-float 1 < 1 / (2 * degree) [
        if not any? link-neighbors with [ member? self mis ] [
          set candidate? true
        ]
      ]
    ]
  ]
end

; Each candidate sends a message to all its neighbors
to send-candidate-messages
  ask turtles with [ candidate? ] [
    let m count link-neighbors
    set number-of-messages number-of-messages + m
    set number-of-bits number-of-bits + m * bits-per-message
  ]
end

; Resolve conflicts – if two candidates are neighbors, one drops out
to resolve-conflicts
  while [ any? turtles with [ candidate? and any? link-neighbors with [ candidate? ] ] ] [
    let a one-of turtles with [ candidate? ]
```

```
    let b one-of other turtles with [ candidate? and link-neighbor? a ]
    if a != nobody and b != nobody [
      if [degree] of a > [degree] of b [
        ask b [ set candidate? false ]
      ]
      if [degree] of b > [degree] of a [
        ask a [ set candidate? false ]
      ]
      if [degree] of a = [degree] of b [ ; If the degree is the same, remove at random
        ifelse random-float 1 < 0.5 [
          ask a [ set candidate? false ]
        ] [
          ask b [ set candidate? false ]
        ]
      ]
    ]
  ]
end

; Remaining candidates join the MIS
to join-mis
  ask turtles with [ candidate? ] [
    set color green              ; Mark them visually
    set mis lput self mis        ; Add to MIS list
    set candidate? false         ; Clear candidate status
  ]
end

; MIS members broadcast their new status to neighbors
to send-mis-join-messages
  ask turtles with [ member? self mis ] [
    let m count link-neighbors
    set number-of-messages number-of-messages + m
    set number-of-bits number-of-bits + m * bits-per-message
  ]
end

; Checks if all nodes are either in the MIS or have a neighbor in the MIS
to-report check-termination
  report all? turtles [
    member? self mis or any? link-neighbors with [ member? self mis ]
  ]
end
```

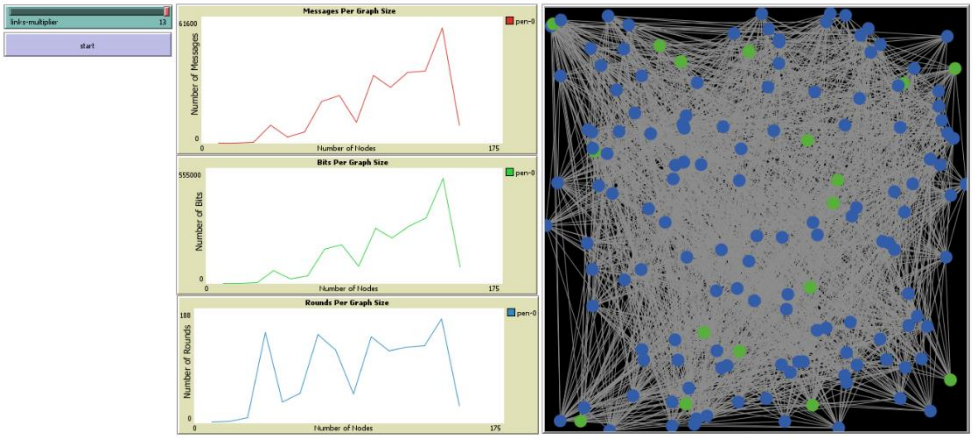
Διαγράμματα



Luby's MIS for 3n links.



Luby's MIS for 8n links.



Luby's MIS for 15n links.

Σχόλια

Για τον Luby's MIS ισχύουν τα ίδια με τον Simple MIS, αλλά τα bits αποτελούν πολλαπλάσιο των μηνυμάτων μιας και κουβαλούν τα IDs και το είδος μηνύματος. Έτσι οι γραφικές παραστάσεις είναι scaling η μία της άλλης. Περισσότερα links συνεπάγονται περισσότερα μηνύματα, bits και γύρους και όπως είναι αναμενόμενο, ο αριθμός των γύρων δε διαφέρει πολύ από τον Simple MIS καθώς είναι και οι δύο λογαριθμικού χρόνου. (Δlogn και logn)

Biologically Inspired MIS

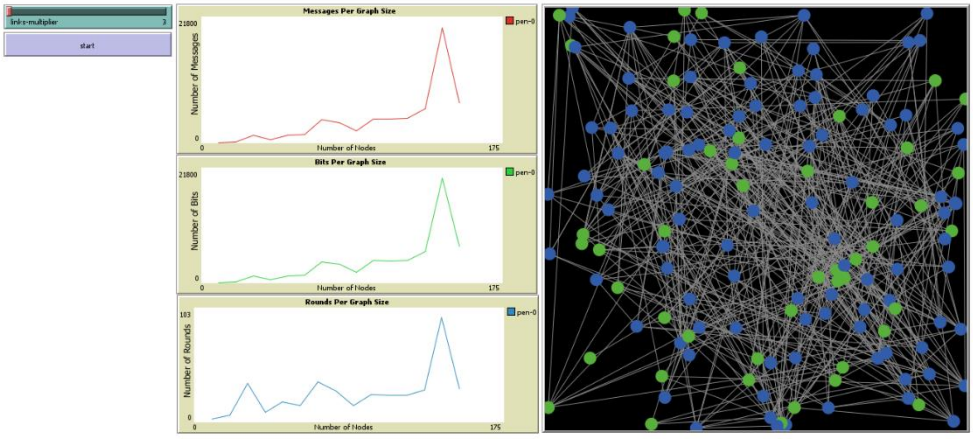
Κώδικας

```
; A series of assumptions have been made:
; 1. Each node sends a message to all neighbors when it becomes candidate or joins the MIS
; 2. The message has an id of ceil(log2(n)) bits and a message type flag of 1 bit
; 3. To simulate sending the messages, I just increment the message and bit counters
; 4. Ticks serve the purpose of rounds

; Global variables used throughout the simulation
globals [
  number-of-messages         ; Total number of messages sent in a run
  number-of-bits             ; Total number of bits sent in a run
  bits-per-message           ; Number of bits required to send a message
  mis                        ; Nodes that are already part of the MIS
]

; Turtle variables
turtles-own [
  degree                     ; Degree of each node
  candidate?                 ; Whether the turtle is a candidate in the current round
]

; Main procedure to run the MIS algorithm for different graph sizes
to start
  clear-all                  ; Clear the world
  reset-ticks                ; Reset tick counter to 0

  ; Loop over different network sizes
  foreach [10 20 30 40 50 60 70 80 90 100 110 120 130 140 150] [ number-of-nodes ->
    set number-of-messages 0
    set number-of-bits 0

    run-algorithm number-of-nodes        ; Run the algorithm for n nodes

    set-current-plot "Messages Per Graph Size"
    plotxy number-of-nodes number-of-messages   ; Plot the total messages used

    set-current-plot "Bits Per Graph Size"
    plotxy number-of-nodes number-of-bits       ; Plot the total bits used

    set-current-plot "Rounds Per Graph Size"
    plotxy number-of-nodes ticks                ; Plot the number of rounds (ticks)
  ]
end

; Sets up a random graph and runs the MIS protocol on it
to run-algorithm [num-nodes]
  let id-bits ceiling log num-nodes 2
  set bits-per-message id-bits + 1              ; Each message carries one ID and a flag bit

  set mis []                                    ; Initialize MIS list

  let max-links num-nodes * (num-nodes - 1) / 2
  let number-of-links links-multiplier * num-nodes

  clear-turtles
  clear-links

  ; Create turtles representing nodes
  set-default-shape turtles "circle"
  create-turtles num-nodes [
    set color blue
    set size max (list 1 (10 / sqrt num-nodes)) ; Adjust size based on density
    setxy random-xcor random-ycor
    set degree 0
```

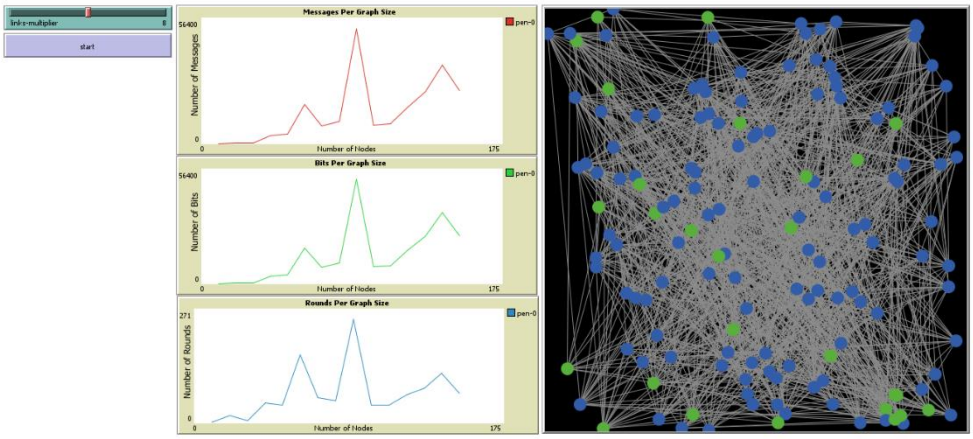
```
    set candidate? false                        ; Initialize candidacy state
  ]

  ; Randomly add links between turtles to form a graph of desired density
  while [ count links < min list number-of-links max-links ] [
    let a one-of turtles
    let b one-of other turtles with [not link-neighbor? a]
    if a != b [
      ask a [ create-link-with b ]
    ]
  ]

  ; Compute the degree of each turtle
  ask turtles [
    set degree count link-neighbors
  ]

  ; Run synchronous MIS rounds until termination
  reset-ticks
  while [ not check-termination ] [
    run-instance
    tick
  ]
end

; Executes one full round of the MIS algorithm
to run-instance
  select-candidates
  send-candidate-messages
  resolve-conflicts
  join-mis
  send-mis-join-messages
end

; Elect candidates with probability 1 / (2 * degree), skip if a neighbor is in MIS
to select-candidates
  ask turtles [
    set candidate? false                        ; Reset candidacy each round
    if degree = 0 [
      set candidate? true
    ]
    if degree != 0 [
      if random-float 1 < 1 / (2 * degree) [
        if not any? link-neighbors with [ member? self mis ] [
          set candidate? true
        ]
      ]
    ]
  ]
end

; Each candidate sends a message to all its neighbors
to send-candidate-messages
  ask turtles with [ candidate? ] [
    let m count link-neighbors
    set number-of-messages number-of-messages + m
    set number-of-bits number-of-bits + m * bits-per-message
  ]
end

; Resolve conflicts – if two candidates are neighbors, one drops out
to resolve-conflicts
  while [ any? turtles with [ candidate? and any? link-neighbors with [ candidate? ] ] ] [
    let a one-of turtles with [ candidate? ]
    let b one-of other turtles with [ candidate? and link-neighbor? a ]
    if a != nobody and b != nobody [
      if [degree] of a > [degree] of b [
        ask b [ set candidate? false ]
      ]
```

```
      if [degree] of b > [degree] of a [
        ask a [ set candidate? false ]
      ]
      if [degree] of a = [degree] of b [           ; If the degree is the same, remove at
random
        ifelse random-float 1 < 0.5 [
          ask a [ set candidate? false ]
        ] [
          ask b [ set candidate? false ]
        ]
      ]
    ]
  ]
end

; Remaining candidates join the MIS
to join-mis
  ask turtles with [ candidate? ] [
    set color green                              ; Mark them visually
    set mis lput self mis                        ; Add to MIS list
    set candidate? false                         ; Clear candidate status
  ]
end

; MIS members broadcast their new status to neighbors
to send-mis-join-messages
  ask turtles with [ member? self mis ] [
    let m count link-neighbors
    set number-of-messages number-of-messages + m
    set number-of-bits number-of-bits + m * bits-per-message
  ]
end

; Checks if all nodes are either in the MIS or have a neighbor in the MIS
to-report check-termination
  report all? turtles [
    member? self mis or any? link-neighbors with [ member? self mis ]
  ]
end
```

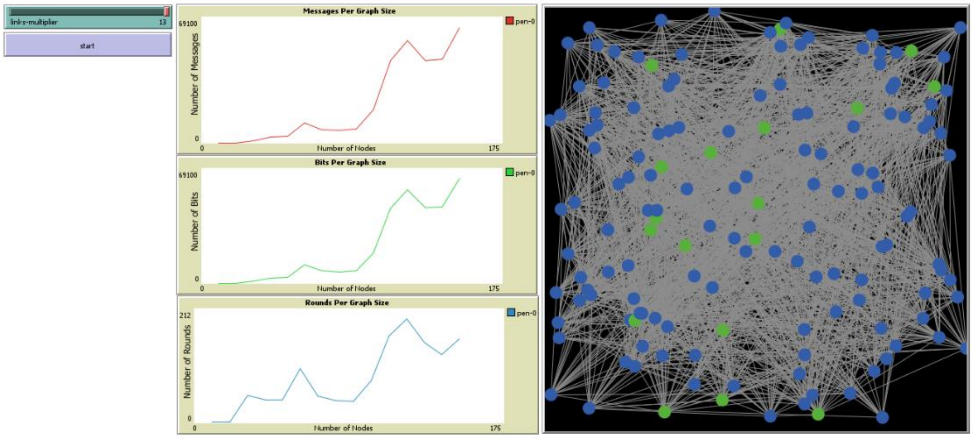
Διαγράμματα



Biologically Inspired MIS for 3n links.



Biologically Inspired MIS for 8n links.



Biologically Inspired MIS for 15n links.

Σχόλια

Ο Biologically Inspired MIS, όπως φαίνεται από τα διαγράμματα, στέλνει τόσα μηνύματα, όσα bits και οι γύροι όπως και τα μηνύματα αυξάνονται ανά αριθμό κόμβων, με ανώμαλο τρόπο βέβαια, λόγω τυχαιότητας του γραφήματος κάθε φορά. Όπως αναμένεται, περισσότερα links, συνεπάγονται περισσότερα μηνύματα αλλά και γύρους. Μιας και τρέχει σε $log^2 n$, κάνει περισσότερους γύρους από τους προηγούμενους αλγόριθμους που έτρεχαν σε λογαριθμικό χρόνο.

Improved Biologically Inspired MIS

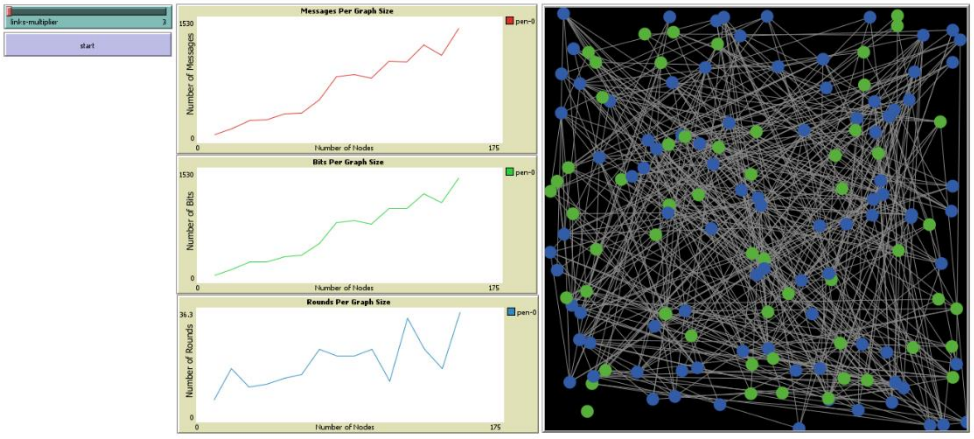
Κώδικας

```
; A series of assumptions have been made:
; 1. Each node sends a 1-bit message to all neighbors when it becomes candidate
; 2. The message has no type; it's just a signal ("I'm signaling")
; 3. To simulate sending the messages, we increment the message and bit counters
; 4. Ticks serve the purpose of rounds

globals [
  number-of-messages         ; Total number of messages sent in a run
  number-of-bits             ; Total number of bits sent in a run
  bits-per-message           ; Number of bits used per message (1 in this case)
  mis                        ; List of nodes that joined the MIS
]

turtles-own [
  v                         ; Whether this node is signaling (1) or not (0)
  inactive?                 ; Whether this node has exited the algorithm
  pv                        ; Local signaling probability
  received-signal?          ; Whether this node received a signal from a neighbor
  join-mis?                 ; Whether this node joined the MIS in the current round
]

; Main driver procedure for running MIS over different network sizes
to start
  clear-all                 ; Clear the world
  reset-ticks               ; Reset tick counter to 0

  foreach [10 20 30 40 50 60 70 80 90 100 110 120 130 140 150] [ num ->
    set number-of-messages 0
    set number-of-bits 0

    run-algorithm num       ; Run the algorithm for num nodes

    set-current-plot "Messages Per Graph Size"
    plotxy num number-of-messages   ; Plot message count for this graph size

    set-current-plot "Bits Per Graph Size"
    plotxy num number-of-bits       ; Plot total bits used for this graph size

    set-current-plot "Rounds Per Graph Size"
    plotxy num ticks                ; Plot number of rounds taken
  ]
end

; Runs the algorithm on a random graph with num-nodes
to run-algorithm [num-nodes]
  set bits-per-message 1           ; Each signal is 1 bit

  set mis []
  clear-turtles
  clear-links
  set-default-shape turtles "circle"
  create-turtles num-nodes [
    set color blue
    set size max (list 1 (10 / sqrt num-nodes)) ; Adjust turtle size based on density
    setxy random-xcor random-ycor
    set inactive? false
    set v 0
    set pv 0.5
    set received-signal? false
    set join-mis? false
  ]

  ; Generate random links to form a connected graph
```

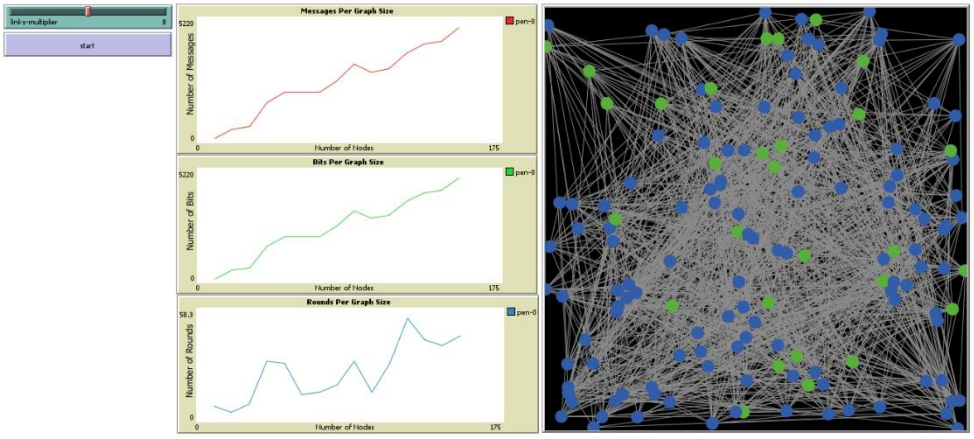
```
  let max-links num-nodes * (num-nodes - 1) / 2
  let target-links links-multiplier * num-nodes
  while [ count links < min list target-links max-links ] [
    let a one-of turtles
    let b one-of other turtles with [ not link-neighbor? a ]
    if a != b [ ask a [ create-link-with b ] ]
  ]

  reset-ticks
  run-instance                ; Begin the MIS protocol
end

; Executes the MIS rounds
to run-instance
  while [not all? turtles [inactive?]] [
    exchange-1
    exchange-2
    tick
  ]
end

; First half of the round: signaling and contention resolution
to exchange-1
  ; Reset per-round state
  ask turtles [
    set v 0
    set received-signal? false
    set join-mis? false
  ]

  ; Nodes signal with probability p_v
  ask turtles with [not inactive?] [
    if random-float 1 < pv [
      set v 1
      send-signal
    ]
  ]

  ; Neighbors of signaling nodes register that they received a signal
  ask turtles with [v = 1] [
    ask link-neighbors with [not inactive?] [
      set received-signal? true
    ]
  ]

  ; Update local probabilities based on contention
  ask turtles with [not inactive? and v = 1] [
    if received-signal? [
      set v 0
      set pv pv / 2                          ; Back off if collision
    ]
    if not received-signal? [
      set pv min (list (2 * pv) 0.5)         ; Increase p_v if no contention
    ]
  ]
end

; Second half of the round: MIS decision and deactivation
to exchange-2
  ; Successful signalers join the MIS
  ask turtles with [v = 1 and not inactive?] [
    set color green
    set join-mis? true
    set inactive? true
    set mis lput self mis
  ]

  ; Neighbors of MIS members deactivate
  ask turtles with [not inactive?] [
```

```
    if any? link-neighbors with [join-mis?] [
      set inactive? true
    ]
  ]
end

; Simulates sending a 1-bit message to all neighbors
to send-signal
  let m count link-neighbors
  set number-of-messages number-of-messages + m
  set number-of-bits number-of-bits + m * bits-per-message
end
```

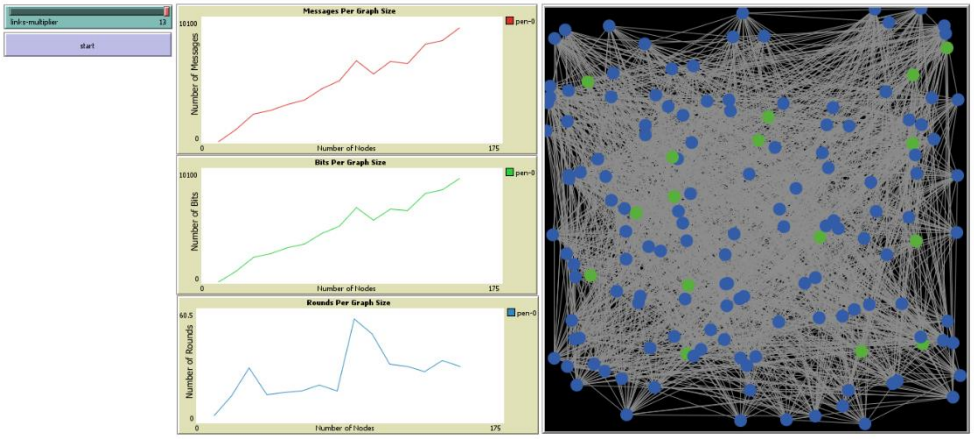
Διαγράμματα



Improved Biologically Inspired MIS for 3n links.



Improved Biologically Inspired MIS for 8n links.



Improved Biologically Inspired MIS for 15n links.

Σχόλια

Ο Improved Biologically Inspired MIS, όπως και ο απλός, έχει ίδιο αριθμό μηνυμάτων με bits. Οι γύροι, όπως και τα μηνύματα, αυξάνονται με την αύξηση του αριθμού των κόμβων. Μάλιστα αυξάνονται με λιγότερο ανώμαλο τρόπο από τους άλλους τρεις αλγόριθμους. Περισσότερα links συνεπάγονται περισσότερα μηνύματα, όπως και γύρους. Επίσης, είναι λογαριθμικού χρόνου και τρέχει γρηγορότερα από τον προηγούμενο, που είναι τετραγωνικού λογαριθμικού. Όπως δείχνουν τα γραφήματα, τρέχει επίσης πιο γρήγορα από τους άλλους 2 αρχικούς αλγόριθμους.

Άσκηση 2

Κώδικας

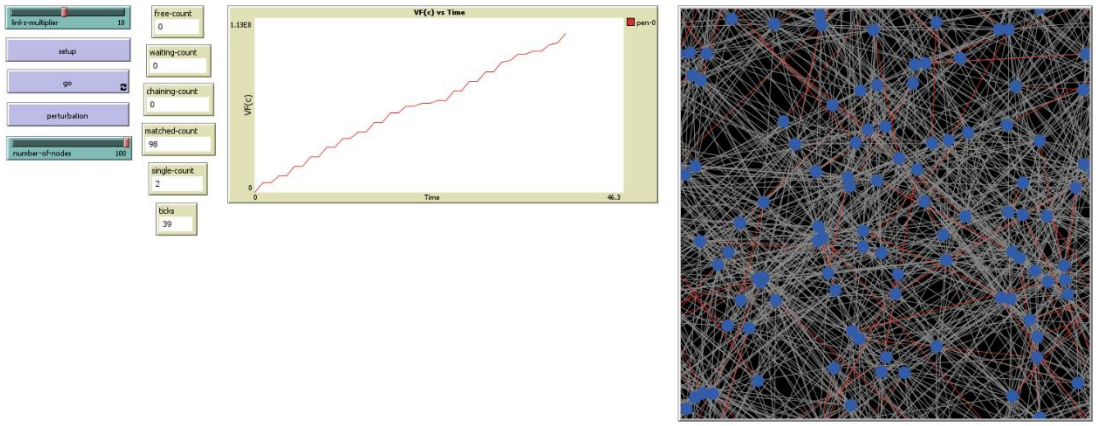
```
globals [
  matched-count         ;; Keeps track of the number of matched turtles
  single-count          ;; Number of turtles that are "single"
  waiting-count         ;; Number of turtles that are "waiting"
  free-count            ;; Number of turtles that are "free"
  chaining-count        ;; Number of turtles that are "chaining"
  spacing-constant      ;; A constant that influences the calculations and
behavior
  proposals             ;; Holds the list of proposals made by free turtles
]

turtles-own [
  pointer               ;; Pointer that stores the partner each turtle
points to
  state                 ;; Current state of the turtle (e.g., matched, free,
waiting)
]

;; The setup procedure is used to initialize the environment and create
turtles
to setup
  clear-all
  ;; Set the spacing constant based on the number of nodes
  set spacing-constant number-of-nodes + 1

  ;; Calculate the maximum number of links (edges) that can exist between
turtles
  let max-links number-of-nodes * (number-of-nodes - 1) / 2

  ;; The actual number of links is controlled by the links-multiplier
  let number-of-links links-multiplier * number-of-nodes

  ;; Set the default shape of the turtles to a circle
  set-default-shape turtles "circle"

  ;; Create the turtles with an initial color and random position
  create-turtles number-of-nodes [
    set color blue  ;; Set turtle color to blue
    set size max list 1 10 / sqrt number-of-nodes  ;; Scale the size of
turtles based on the number of nodes
    setxy random-xcor random-ycor  ;; Place the turtles randomly in the
world
    set pointer nobody  ;; Initially, turtles have no pointer (not matched)
  ]

  ;; Create links (edges) between turtles until the target number of links
is reached
  while [ count links < min list number-of-links max-links ] [
    let a one-of turtles
```

```
    let b one-of other turtles with [not link-neighbor? a]  ;; Pick two
turtles that aren't already linked
    if a != b [
      ask a [ create-link-with b ]  ;; Create a link between these two
turtles
    ]
  ]

  reset-ticks  ;; Reset the tick counter to start the simulation
end

;; The main procedure that runs during each tick of the simulation
to go
  ;; Reset counts for each run
  set matched-count 0
  set waiting-count 0
  set free-count    0
  set single-count  0
  set chaining-count 0
  set proposals []  ;; Clear the list of proposals

  ;; Clear the state of all turtles before the classification phase
  ask turtles [ set state "" ]

  ;; Classification Phase: Determine the state of each turtle
  ask turtles [
    if pointer != nobody and [pointer] of pointer = self [  ;; If a turtle
points to itself, it's "matched"
      set state "matched"
      set matched-count matched-count + 1
    ]
  ]

  ;; Waiting turtles: If a turtle is not yet assigned a pointer but has
neighbors, it's in the "waiting" state
  ask turtles with [state = ""] [
    if pointer = nobody and any? link-neighbors with [pointer = self] [
      set state "waiting"
      set waiting-count waiting-count + 1
    ]
  ]

  ;; Free turtles: If a turtle has no pointer and is not yet waiting, it is
considered "free"
  ask turtles with [state = ""] [
    if pointer = nobody and not any? link-neighbors with [pointer = self]
       and any? link-neighbors with [pointer = nobody] [
      set state "free"
      set free-count free-count + 1
    ]
  ]

  ;; Single turtles: If a turtle is not matched but all its neighbors are
already pointing to others, it's "single"
  ask turtles with [state = ""] [
```

```
    if pointer = nobody and all? link-neighbors [pointer != nobody] [
      set state "single"
      set single-count single-count + 1
    ]
  ]

  ;; Chaining turtles: If a turtle is pointing to another turtle that is
not pointing back, it enters the "chaining" state
  ask turtles with [state = ""] [
    if pointer != nobody and [pointer] of pointer != self [
      set state "chaining"
      set chaining-count chaining-count + 1
    ]
  ]

  ;; Free turtles propose to neighbors (deterministically)
  ask turtles with [state = "free"] [
    let target min-one-of link-neighbors with [pointer = nobody] [who]
    if target != nobody [
      set proposals lput (list self target) proposals  ;; Add the proposal
to the list
    ]
  ]

  ;; Apply the proposals by updating pointers
  (foreach proposals [
    pair ->
      let from-item item 0 pair
      let to-item item 1 pair
      ask from-item [ set pointer to-item ]  ;; Turtle "from-item" points
to "to-item"
  ])

  ;; Waiting turtles: Accept the lowest-numbered turtle as their suitor
  ask turtles with [state = "waiting" and pointer = nobody] [
    let suitor min-one-of link-neighbors with [pointer = self] [who]
    if suitor != nobody [
      set pointer suitor  ;; Accept the suitor's pointer
    ]
  ]

  ;; Chaining turtles: Clear their pointer
  ask turtles with [state = "chaining"] [
    set pointer nobody  ;; Clear the pointer as they were in a chain
  ]

  ;; Visualize matched pairs by coloring the links red
  ask turtles with [state = "matched"] [
    let partner pointer
    if partner != nobody [
      ask link-with partner [ set color red ]  ;; Set the link color to red
for matched turtles
    ]
  ]
```

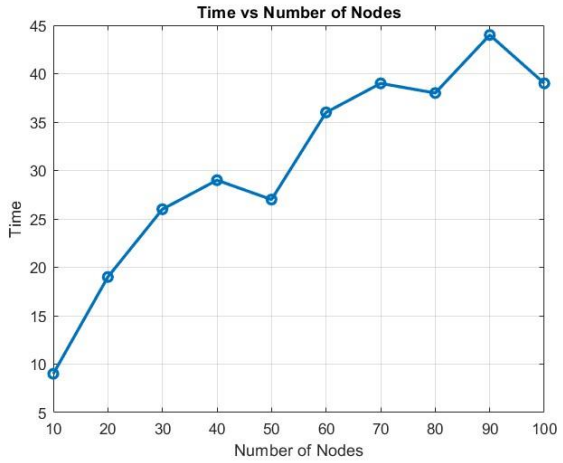
```
  ;; Plot VF(c) over time (VF is a function we define below)
  set-current-plot "VF(c) vs Time"
  plotxy ticks vf

  ;; Stop the simulation once the configuration is safe
  if safe-configuration vf [ stop ]
  tick  ;; Advance the simulation by one tick
end

;; This function calculates the VF(c) value based on the current counts of
states
to-report vf
  report spacing-constant ^ 3 * (matched-count + single-count)
         + spacing-constant ^ 2 * waiting-count
         + spacing-constant * free-count
         + chaining-count
end

;; This function checks if the configuration has reached the "safe" state
to-report safe-configuration [func-value]
  report func-value = number-of-nodes * (spacing-constant ^ 3)
end

;; Perturbation procedure randomly changes pointers of some turtles to
simulate disruptions
to perturbation
  let num-to-perturb floor (number-of-nodes * 0.1)  ;; 10% of turtles will
be perturbed
  let targets n-of num-to-perturb turtles  ;; Choose the turtles to perturb

  ask targets [
    let non-neighbors other turtles with [not link-neighbor? myself and
self != myself]
    if any? non-neighbors [
      set pointer one-of non-neighbors  ;; Assign a random non-neighbor as
a new pointer
    ]
  ]
end
```

Σχόλια



Στιγμιότυπο τρεξίματος από το interface του NetLogo.

Όπως φαίνεται στη διεπαφή, υπάρχει κουμπί setup που δημιουργεί τυχαίο γράφημα, και go που τρέχει τον αλγόριθμο. Στον κώδικα έχουν υλοποιηθεί οι ομώνυμες συναρτήσεις και ακόμη τρεις. Μία που ελέγχει τον τερματισμό, μία που υλοποιεί τη συνάρτηση δυναμικού και μία που υλοποιεί τη λειτουργία του κουμπιού perturbation, που ουσιαστικά εισάγει μια διατάραξη στο γράφημα. Η σύμβαση που έγινε είναι πως κατά τη διαταραχή, επιλέγεται τυχαία το 10% των κόμβων και θέτουν τους δείκτες τους σε άλλους τυχαίους, μη γειτονικούς κόμβους. Η αυτοσταθεροποίηση που χαρακτηρίζει τον αλγόριθμο του επιτρέπει να διορθώσει τη διαταραχή αυτή. Επίσης, στη διεπαφή υπάρχουν και δύο sliders που ελέγχουν τον αριθμό των κόμβων, όπως και τον πολλαπλασιαστή των ακμών. Υπάρχουν επιπρόσθετα monitors για την παρακολούθηση των ticks, όπως και του αριθμού των κόμβων που ανήκουν σε κάθε κατηγορία. Στο τέλος κάθε τρεξίματος, οι αριθμοί στα monitors των matched-count και single-count αθροίζουν στον αριθμό των συνολικών κόμβων. Τέλος, όπως φαίνεται στο παρακάτω γράφημα, η αύξηση του αριθμού των κόμβων αυξάνει και το χρόνο εκτέλεσης.



Εργαλείο: MATLAB

Βιβλιογραφία

Schmid, Stefan. GIAN Lecture Notes: Network Algorithms. n.d., https://schmiste.github.io/GIAN-Lecture-Notes-NetAlg.pdf

Τσίχλας, Κωνσταντίνος, "2η Διάλεξη: Μη-Επεκτάσιμο Ανεξάρτητο Σύνολο" CEID_NE589, 20 Μαρτίου 2025, Πανεπιστήμιο Πατρών. Παρουσίαση Διάλεξης. https://eclass.upatras.gr/modules/document/file.php/CEID1220/02-MIS.pdf

Τσίχλας, Κωνσταντίνος, "3η Διάλεξη: Αυτοσταθεροποίηση (Self-Stabilization)" CEID_NE589, 20 Μαρτίου 2025, Πανεπιστήμιο Πατρών. Παρουσίαση Διάλεξης. https://eclass.upatras.gr/modules/document/file.php/CEID1220/03-Self_Stabilization.pdf