Τυχαιοκρατικοί Αλγόριθμοι

Άσκηση 1

Αμπλιανίτης Κωνσταντίνος

2017030014

Σκοπός Άσκησης: Η προσομοίωση του min-cut αλγορίθμου σε περιβάλλον Matlab.

Περιγραφή τεχνολογίας που χρησιμοποιήθηκε: Για την επίλυση της άσκησης χρησιμοποιήθηκε το εργαλείο Matlab (έκδοση 2016b) και το demo που μας δόθηκε μαζί με την εκφώνηση της εργασίας.

Περιγραφή της μεθοδολογίας επίλυσης των ασκήσεων:

Για την επίλυση της άσκησης σύμφωνα με το demo που μας δόθηκε, ζητήθηκε η υλοποίηση μίας συνάρτησης δημιουργίας ενός adjacency matrix (generate_adjacency_matrix), η εύρεση με τυχαίο τρόπο μίας ακμής του (find_edge_uniformly) και η κατάλληλη ενημέρωση του (πίνακα) ώστε να κάνει σωστά το merge των δύο κόμβων και να επιστρέφει έναν καινούργιο adjacency matrix στην επιθυμητή μορφή (update_adjucency_matrix). Τέλος, ζητήθηκε επίσης να υλοποιηθεί μία συνάρτηση που να επιστρέφει την νέα λίστα κόμβων που έχει προκύψει μετά τις αλλαγές στον adjacency matrix (συμπτυξη των κόμβων - update_nodes).

Συνάρτηση generate_adjacency_matrix(n,c):

Για να μπορέσουμε να γράψουμε αυτή την συνάρτηση πρέπει να μελετήσουμε τις ιδιότητες του adjacency Matrix. Αφού γνωρίζουμε ότι ο adjacency Matrix είναι πίνακας μορφής $n \times n$ (n ο αριθμός επιθυμητών κόμβων), δημιουργούμε έναν πίνακα μηδενικών $n \times n$ (εντολή $zeros(n)$) . Στην συνέχεια αφού σύμφωνα με την εκφώνηση ο κόμβος 1 συνδέεται με μία ακμή με τους κόμβους 2...c+1, δημιουργούμε μία for loop που να τρέχει από 2 (καθώς δεν θέλουμε να έχουμε self loop με το 1) μέχρι το c+1 και κάνουμε το εκάστοτε στοιχείο του adjacency matrix(1,i) ίσο με 1 (γιατί δεν έχουμε πειράξει τον πίνακα πριν ώστε να έχουμε

άλλες τιμές να εμπλέκονται). Επιπλέον η ίδια διαδικασία θα πρέπει να γίνει για τα στοιχεία adjacency matrix(i,1) αφού ο πίνακας μας είναι συμμετρικός. Για την σύνδεση των παραπάνω κόμβων μεταξύ τους (2... n) με αριθμό ακμών ομοιόμορφα κατανεμημένο στο διάστημα 1...10, αρχικά δημιουργήσαμε πάλι μία for που να τρέχει από το 2 μέχρι το n-1. Για κάθε επανάληψη i επιλέγεται τυχαία ο αριθμός ακμών. Στη συνέχεια ελέγχεται η περίπτωση να έχουμε ήδη συμπληρώσει το ποσό των επιτρεπτών γραμμών πράγμα που θα σήμαινε ότι δεν μπορούμε να προσθέσουμε ακμές και πρέπει να συνεχίσουμε στον επόμενο κόμβο. Σε περίπτωση που δεν έχουμε συμπληρώσει ακμές και ο αριθμός ακμών που επιλέχθηκε τυχαία είναι μεγαλύτερος από τον αριθμό ακμών που έχουμε ήδη, ενώνουμε με ακμές τον κόμβο που βρισκόμαστε με τυχαίους κόμβους προσέχοντας πάντα ο κόμβος με τον οποίο θα κάνουμε σύνδεση να μην έχει ξεπεράσει και αυτός τον μέγιστο αριθμό ακμών. Σημαντικό να σημειωθεί ότι οι κόμβοι ένωσης επιλέγονται από το διάστημα i+1 έως n με σκοπό να αποφύγουμε self loops. Τέλος φτάνουμε στον τελευταίο κόμβο και ελέγχουμε την περίπτωση οι κόμβοι από 2 έως n-1 να μην έχουν καμία ακμή ως προς αυτόν τον κόμβο (worst case scenario).Σε περίπτωση που ισχύει αυτό βάζουμε τον τελευταίο κόμβο να δημιουργήσει μία ακμή με έναν τυχαίο κόμβο μεταξύ των 2 έως n-1 που δεν έχει συμπληρώσει αριθμό ακμών με σκοπό να μην έχουμε κανέναν ασύνδετο κόμβο στο γράφο.

Ο κώδικας της συνάρτησης συνολικά:

```
function A = generate_adjacency_matrix(n, min_cut_weight)


adjacencyM = zeros(n);
%% creation of c edges betweeen 1 and 2...c+1
for i=1:min_cut_weight+1
    adjacencyM(1,i) = 1;
    adjacencyM(i,1) = 1;
end


%% rest of the nodes
for j=2:n-1
```

```
% number of edges that will have the ith vetrice
numofedges = randi(10);
% find how many edges i have now so that i can compare
%with the
% rand number .... case more edges have came from other
% vetrices
numofedgesnow = sum(adjacencyM(j,:));
% if the i is in that range it has already an edge
if( j < min_cut_weight+1)
    while (numofedgesnow < numofedges +1)
          newE = randi([j+1 n ]);
          % i have to check if this vetrice has
          % more than the legal values
          if( newE <= min_cut_weight+1)
              newEline = sum(adjacencyM(newE,:));
              if (newEline < 11)
                adjacencyM(j,newE) = adjacencyM(j,newE) + 1;
                adjacencyM(newE,j) = adjacencyM(newE,j) +1;
                numofedgesnow = numofedgesnow +1;
              end
          else
              newEline = sum(adjacencyM(newE,:));
              if (newEline < 10)
              adjacencyM(j,newE) = adjacencyM(j,newE) + 1;
              adjacencyM(newE,j) = adjacencyM(newE,j) +1;
              numofedgesnow = numofedgesnow +1;
          end
    end
end
else
    while (numofedgesnow < numofedges)
          newE = randi([j+1 n ]);
```

```
                % i have to check if this vetrice has more than
                % the legal values
                if( newE <= min_cut_weight+1)
                    newEline = sum(adjacencyM(newE,:));
                    if (newEline < 11)
                        adjacencyM(j,newEdge) = adjacencyM(j,newEdge) + 1;
                        adjacencyM(newEdge,j) = adjacencyM(newEdge,j) +1;
                        numofedgesnow = numofedgesnow +1;
                      end
                else
                    newEline = sum(adjacencyM(newE,:));
                    if (newEline < 10)
                        adjacencyM(j,newE) = adjacencyM(j,newE) + 1;
                        adjacencyM(newE,j) = adjacencyM(newE,j) +1;
                        numofedgesnow = numofedgesnow +1;
                    end
              end
          end
      end
end
% case last element of the table has no edges
worstcasescenario = sum(adjacencyM(n,:));
while worstcasescenario == 0
      edgetcmplt = randi(n-1);
      if( edgetcmplt <= min_cut_weight+1)
          sumoff = sum(adjacencyM(edgetcmplt,:));
          if ( sumoff < 11)
            adjacencyM(n,edgetcmplt) = adjacencyM(n,edgetcmplt) + 1;
            adjacencyM(edgetcmplt,j) = adjacencyM(edgetcmplt,n) +1;
            worstcasescenario = 1;
          end
      else
```

```
            if( sumoff < 10)
              adjacencyM(n,edgetcmplt) = adjacencyM(n,edgetcmplt) + 1;
              adjacencyM(edgetcmplt,j) = adjacencyM(edgetcmplt,n) +1;
              worstcasescenario = 1;
            end
        end
end
A = adjacencyM;
end
```

Συνάρτηση find_edge_uniformly(AA):

Για να βρω τυχαία μία ακμή χρησιμοποιώ την ενολή find για τον πίνακα AA η οποία επιστρέφει τον συνδιασμό γραμμών στηλών που δεν έχουν μηδενικά στοιχεία. Έπειτα παίρνω τυχαία έναν αριθμό στο διάστημα 1 έως το μήκος του πίνακα γραμμών που βρέθηκε από την εντολή find(). Τέλος επιστρέφω ως u,v το περιεχόμενο των πινάκων row και col στην θέση του τυχαίου αριθμού που πήραμε από πριν.

Ο κώδικας της συνάρτησης συνολικά:

```
function [u, v] = find_edge_uniformly(AA)
    %find all the elements that have non zero values
    [row, col]= find(AA);

    lenrow = length(row);

    randa = randi([1 lenrow]);

    uu = row(randa);
    vv = col(randa);

    u = uu;
    v = vv;
end
```

Συνάρτηση update_adjacency_matrix(adjacencyM, u, v):

Για την υλοποίηση της συνάρτησης επέλεξα να κάνω το merge ως προς τον κόμβο u και να αφαιρώ τον κόμβο v από τον πίνακα ώστε ο adjacency matrix μου κάθε φορά να γίνεται μικρότερος σε στήλες-γραμμές κατά 1 μέχρι να φτάσει να έχει μορφή πίνακα 2×2. Για να γίνει αυτό, πρέπει να διασχίσω τις στήλες πίνακα με μία εντολή for για την γραμμή v. Αν κάποιο στοιχείο i της γραμμής δεν είναι μηδενικό και το i δεν είναι το u (καθώς αν είναι το u έτσι και αλλιώς θα διαγραφεί για να μην προκύψει self loop μετά το merge) τότε προσθέτω τον αριθμό αυτό (αντιπροσωπεύει τις ακμές μεταξύ του κόμβου v με τον κόμβο i) στο στοιχείο που υπάρχει στον πίνακα στην θέση (u,i). Επειδή ο πίνακας όμως είναι συμμετρικός και πρέπει να διατηρήσω την συμμετρία του προσθέτω το ίδιο στοιχείο στο στοιχείο του πίνακα στην θέση (i,u). Τέλος διαγράφω τόσο την γραμμή όσο και την στήλη v.

Ο κώδικας της συνάρτησης:

```
function AA = update_adjacency_matrix(adjacencyM, u, v)
    for i=1:length(adjacencyM)
        if( adjacencyM(v,i) > 0 && i~= u)
            adjacencyM(u,i) = adjacencyM(u,i) + adjacencyM(v,i);
            adjacencyM(i,u) = adjacencyM(i,u) + adjacencyM(i,v);
        else
            continue
        end
    end
    %delete the row and the collumn
    adjacencyM(v,:) = [];
    adjacencyM(:,v) = [];

    AA = adjacencyM;
end
```

Συνάρτηση update_nodes(nodes, u, v):

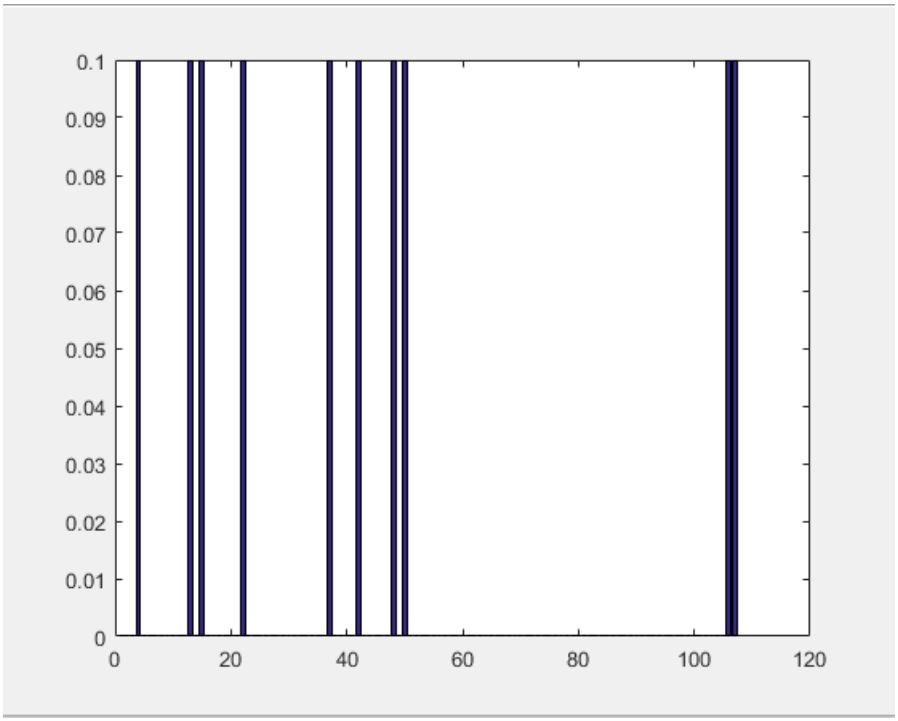
Για να γίνουν update οι κόμβοι (η λίστα nodes έχει δημιουργηθεί στο main body της άσκησης), αφού έχει αποφασιστεί ότι θα γίνεται merge ως προς τον κόμβο u, πρέπει να βρίσκω κάθε φορά το μήκος του πίνακα που υπάρχει μέσα το nodesu (αφού το nodes είναι τύπου cell) και στην συνέχεια να βρω το μήκος του nodesv. Τέλος, για να ολοκληρώσω την διαδικασία, μέσω μίας δομής επανάληψης (for) προσθέτω στο (μήκος πίνακα nodesu+1, 1) το στοιχείο nodesv(i,1) και διαγράφω από την λίστα των nodes τον πίκανα nodesv.

Ο κώδικας της συνάρτησης:

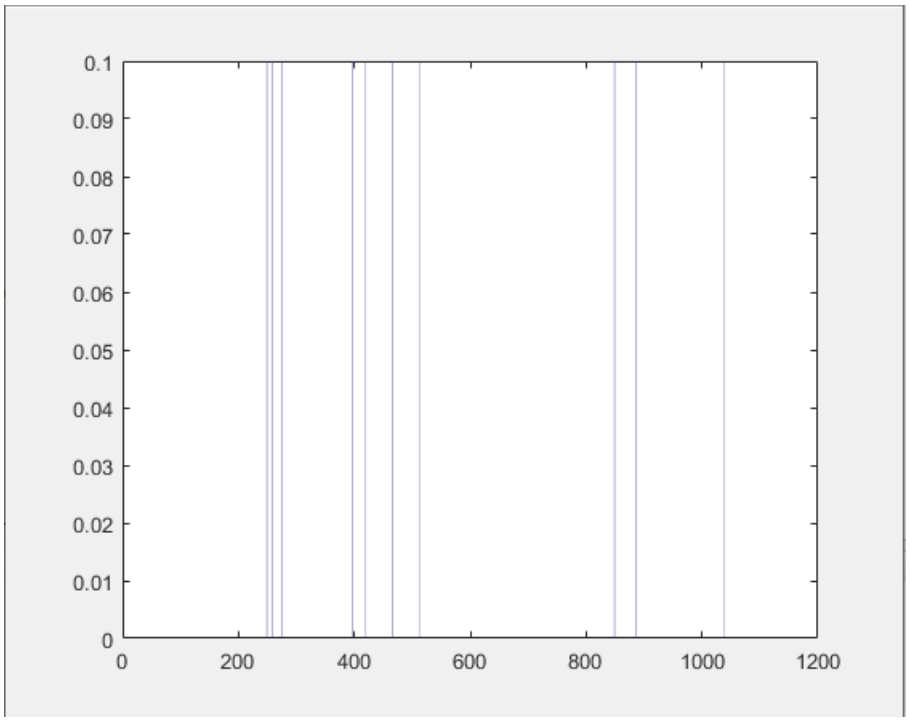
```
function nodef = update_nodes(nodes, u, v)
    a= length(nodes{v});
    for i=1:a
        b = length(nodes{u});
        nodes{u}(b+1) = nodes{v}(i);
    end
    nodes(v)=[];
    nodef = nodes;
end
```

Αποτελέσματα:

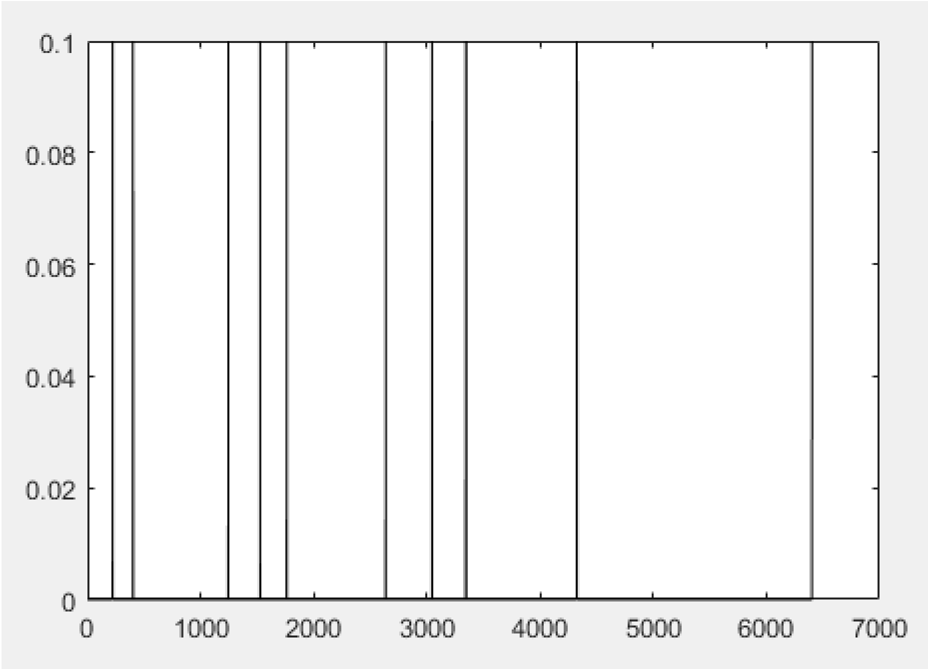
Για λόγους που αφορούν το σύστημα μου (για 500 κόμβους που ήταν αυτό που δόθηκε είχα προβλήματα με το matlab-το έκλεινε ο υπολογιστής λόγω μνήμης) ο κώδικας δοκιμάστηκε για 50, 100 ,200 κόμβους με min cut 10,20,60 αντίστοιχα. Τα histograms που προκύπτουν είναι τα εξής:



Σχήμα 1: Histogram για 50 κόμβους



Σχήμα 2: Histogram για 100 κόμβους



Σχήμα 3: Histogram για 200 κόμβους

Παρατηρείται ότι όσο αυξάνουμε τον αριθμό των κόμβων βλέπουμε αύξηση και των προσπαθειών που γίνονται μέχρι να εκτελεστεί ο αλγόριθμος με επιτυχία. Παρ' όλα αυτά βλέπουμε ότι η αύξηση των προσπαθειών είναι μέχρι και τάξης ανώτερη της αύξησης των κόμβων. Κάτι τέτοιο είναι λογικό καθώς με την αύξηση των κόμβων δημιουργούνται πολλά καινούργια μονοπάτια και έτσι ο αλγόριθμός μας έχει περισσότερες επιλογές οι οποίες μπορεί να καταλήξουν σε λάθος αποτέλεσμα μιας και οι επιλογή των κόμβων γίνεται τυχαία.

Ο κώδικας του main body της εργασίας:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% M file for testing the randomized min-cut algorithm
%
%
%
% A. P. Liavas, Feb. 25, 2015, March 1, 2021
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
clear, clc

% Number of nodes
n = 200;
min_cut_weight = 60;

A = generate_adjacency_matrix(n, min_cut_weight);



for iter=1:10 % Repeat the whole process

    iter

    % Graph partitions - the algorithm returns two nodes
    opt1 = [];
    opt2 = [];

    inner_iter = 0;
    found = 0;
    while (found == 0)

        inner_iter = inner_iter + 1;

        % Randomized min-cut algorithm
        % Initialization
        AA = A;
        nodes = cell(n,1);
        for jj=1:n
          nodes{jj} = jj;
        end
```

```
        % Main body
        for ii=n:-1:3     % perform n-2 iterations
            [u, v] = find_edge_uniformly(AA);
            AA = update_adjacency_matrix(AA, u, v);
            nodes = update_nodes(nodes, u, v);
        end



        % Check weight of output
        if ( AA(1,2) == min_cut_weight )
            found = 1;
            fprintf('\nMin-cut found... %d\n', iter);
            opt1 = sort(nodes{1}(:));
            opt2 = sort(nodes{2}(:));
        else
            fprintf('\nMin-cut failed...%d\n', iter);
        end

    end
    iters(iter) = inner_iter;
end

a = hist(iters, [1:max(iters)]);
bar(a/iter)
```