Week 6: Markov chains
Ranking of nodes in graphs
Solutions

A Markov chain model. The stochastic process $A_{\mathbb{N}}$ describing the agent's visits to nodes of the graph is an MC because it is memoryless—the probability of visiting any node depends only on the current node—and time-invariant—the probability of jumping from node i to node j does not depend on the time n. Both of these observations stem directly from the transition probability P_{ij} in (1).

As for the statements, I am giving an extensive explanation of each case below. This is for your benefit. It does *not* mean you are expected to delve into such details in your answers: a definition followed by an intuition is enough.

1. State i of this MC, i.e., visits of the agent to node i, is transient: since the MC is finite, the only way for a state i to be transient is if some node $j \neq i$ can be reached from i through a path of nonzero probabilities, but not vice versa. This could happen if the incoming neighborhood of node i is empty or if the incoming neighborhood of ALL nodes in its incoming neighborhood have empty incoming neighborhoods and so on.

   However, the collaboration matrix in this homework is symmetric, i.e., if student i collaborated with student j, then so did student j with student i. Hence, it induces an undirected graph. Therefore, every path is "bidirectional" and none of the states are transient.

2. All states of this MC are transient: In a finite MC, it is *impossible* for ALL states to be transient. We prove that by contradiction. Suppose all J states of a MC are transient. Now pick any node $i_0 \in \mathcal{V}$. Since i_0 is transient, there exist a path of nonzero probability between i_0 and another node $i_1 \in \mathcal{V}$ such that there is no path of nonzero probability from i_1 back to i_0. Moreover, since i_1 is also transient, there must exist a path of nonzero probability between i_1 and another node $i_2 \in \mathcal{V}$, but no such path from i_2 back to i_1. If we repeat this process J times, however, we will obtain a path containing $J+1$ nodes. Since the underlying graph of the MC only has J nodes, one of them must appear twice in the path. Call this node $i^\star$. We have therefore constructed a path of nonzero probability from $i^\star$ back to $i^\star$, which contradicts the assumption that it is transient.

3. All the states of this MC are recurrent: We know that some states of this MC must be recurrent. But for all states to be recurrent, there must exist a path of nonzero probability from any state $i \in \mathcal{V}$ to any state $j \in \mathcal{V}$. In graph-theoretic terms, every component (i.e., class of states) of the underlying graph must be a strongly connected component. For instance, this is the case when the underlying graph of the MC is undirected (as in the collaboration matrix).

4. All states of this MC are aperiodic: Recall that periodicity is a class property. In a directed graph, this property is related to the existence of cycles. For instance, if a class is formed

by a single cycle, then that class is periodic. If a class is formed by interconnected cycles of the same length (or more generally, if the lengths of the cycles are not co-primes), then it is again periodic.

In the case of the collaboration matrix, which is an undirected graph with no self-loops (unless the class consists of only one state), the number of steps it takes to return to a given state is necessarily a multiple of two (as in the random walk on a line). Therefore, suffices for a class to have a loop of odd length for that class to be aperiodic (why?).

5. All states are positive recurrent: Once again, note that positive recurrence is a class property. Do not confuse positive recurrence of all of the states with positive recurrence of the MC, which additionally requires irreducibility. In a finite MC, all recurrent classes are positive recurrent, i.e., the expected return time to any node is finite. Showing this from first principles is not easy. But we can argue through the limiting distribution $\boldsymbol{\pi}_{\mathcal{C}}$. Indeed, since the class is finite, we can write the eigenvalue problem $\boldsymbol{\pi}_{\mathcal{C}} = \boldsymbol{P}_{\mathcal{C}}^T \boldsymbol{\pi}_{\mathcal{C}}$ subject to $\boldsymbol{\pi}_{\mathcal{C}}^T \mathbb{1} = 1$, where $\boldsymbol{P}_{\mathcal{C}}$ is the transition matrix of class $\mathcal{C}$. Since $\boldsymbol{P}_{\mathcal{C}}$ is a finite matrix with finite entries, this problem has a solution finite solution. Since the class has a limiting distribution, it must be positive recurrent. Moreover, the stationary distribution is related to the expected time of return by

$$\pi_i = \frac{1}{\mathbb{E}[T_i]},$$

where π_i is the limit probability for node i and $\mathbb{E}[T_i]$ is the expected time of return (the proof is a cute application of the law of large numbers, you can email your TAs about it if you're interested). So if $\boldsymbol{\pi}_{\mathcal{C}}$ is finite, then so are the expected return time of all states in $\mathcal{C}$.

6. All states are ergodic: Again, ergodicity is a class property: do not confuse ergodicity of all of the states with ergodicity of the MC, which requires irreducibility. A state is ergodic if and only if it is aperiodic and positive recurrent. Suffices then to put together items 3–5. For the undirected graph case, suffices for a class to be composed of a single state or have an odd-length loop to be ergodic.

7. The MC is irreducible: To be irreducible the underlying graph of the MC must be strongly connected (form a single strongly connected component). In the case of the collaboration matrix, this means that there must not be singletons (students alone) or disjoint classes of friendship.

B Random walk implementation. You'll find the MATLAB script that implements the random walk method of rank computation for the reducible collaboration graph below.

```
% Delete all variables and close figures
clear all
close all

% Load collaboration matrix
collaboration_matrix;

J = size(graph,1);  % Number of states
D = sum(graph);     % Degree of each node of the graph
N = 1e6;            % Length of simulation
A0 = 1;             % Initial state
% A0 = 6;             % Initial state

% Random walk
```

```
number_visits = zeros(J,1);
current_state = A0;
for n = 1:N
    number_visits(current_state) = number_visits(current_state) + 1;
    neighborhood = find(graph(current_state,:));
    current_state = neighborhood( randi(D(current_state)) );
end

% Compute ranks
ranks = number_visits/N;

% Display ranks
figure();
bar(1:J, ranks);
xlabel('Node');
ylabel('Rank');
xlim([0.5 J+0.5]);
grid;


%%% Export figure %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
set(gcf,'Color','w');
export_fig -q101 -pdf HW6_B1-1.pdf
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Results are depicted in Figure 1. Notice that we can investigate the disjoint classes by inspecting Figure 1. Indeed, states that receive zero rank (explicitly, states 6, 11, 16, 28, and 33), i.e., states that were never visited, belong to a different class than state $A_0 = 1$. If we change the initial state to $A_0 = 6$, for instance, we obtain Figure 2. Immediately, we can see that nodes 6, 16, and 33 form a component. Performing the same for states 11 and 28 reveal that they are singletons.

To include the "professor" (fully connected node), we simply augment the graph as in the following script and repeat the random walk experiment.

```
% Delete all variables and close figures
clear all
close all

% Load collaboration matrix
collaboration_matrix;

% Include fully connected node to graph (node "J+1")
graph = [graph ones(size(graph,1),1) ; ones(1,size(graph,1)) 0];

J = size(graph,1);  % Number of states
D = sum(graph);     % Degree of each node of the graph
N = 1e6;            % Length of simulation
A0 = 1;             % Initial state

% Random walk
number_visits = zeros(J,1);
current_state = A0;
for n = 1:N
    number_visits(current_state) = number_visits(current_state) + 1;
    neighborhood = find(graph(current_state,:));
    current_state = neighborhood( randi(D(current_state)) );
end

% Compute ranks
ranks = number_visits/N;

% Display ranks
figure();
```

```
bar(1:J, ranks);
xlabel('Node');
ylabel('Rank');
xlim([0.5 J-1+0.5]);
grid;


%%% Export figure %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
set(gcf,'Color','w');
export_fig -q101 -pdf HW6_B2.pdf
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

The results are shown in Figure 3. Notice that we only show the ranks of the original nodes and ignore the fully connected node. Indeed, since this node is connected to all nodes it has an artificially high rank that distorts the plot. Moreover, observe that since the graph is now irreducible, there is only one class and all nodes have non-zero rank.

C Probability update. Following slides 19 and 20, in ranking_nodes_in_graphs, we can write the probability $p_i(n+1)$ as

$$p_i(n+1) = \sum_{j \in \mathcal{N}^{-1}(i)} \mathbb{P}\left[A_{n+1} = i \mid A_n = j\right] \mathbb{P}\left[A_n = j\right] = \sum_{j \in \mathcal{N}^{-1}(i)} P_{ji} p_j(n).$$

In matrix form this becomes

$$\boldsymbol{p}(n+1) = \boldsymbol{P}^T \boldsymbol{p}(n),$$

where $\boldsymbol{P}$ is the transition probability matrix and $\boldsymbol{p}(n)$ is the vector that collects the probabilities of being in each state.

D Probability update implementation. The limit probabilities $\lim_{n\to\infty} p_i(n)$ exist when the MC is positive recurrent and aperiodic (if it is periodic the probability oscillates). Thus, this MC has a limit distribution (slides 77 to 82 of markov_chains). Moreover, even if the MC is reducible, it is always ergodic *within* each of its classes. Thus, the limit distribution within class $\mathcal{C}$ describes the fraction of time spent on the states of $\mathcal{C}$ conditioned on the fact that the initial state is in $\mathcal{C}$. Therefore, the limit probabilities can be used in this case to compute the rank $r_i(A_0)$ using the following MATLAB script:

```
% Delete all variables and close figures
clear all
close all

% Load collaboration matrix
collaboration_matrix;

P = random_walk;
J = size(graph,1);  % Number of states
D = sum(graph);     % Degree of each node of the graph
N = 100;            % Length of simulation
A0 = 1;             % Initial state

% Probability update
p = zeros(J,1);
p(A0) = 1;

% Also known as p = (P')^N*p...
for n = 1:N
    p = P'*p;
```

```
end

% Compute ranks
ranks = p;

% Display ranks
figure();
bar(1:J, ranks);
xlabel('Node');
ylabel('Rank');
xlim([0.5 J+0.5]);
grid;


%%% Export figure %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
set(gcf,'Color','w');
export_fig -q101 -pdf HW6_D1.pdf
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

The results can be found in Figure 4.

Now, for the limit probabilities to be independent of initial distribution, the MC must additionally be irreducible (and hence an ergodic MC, refer to slide 60 of markov_chains). Recall that adding the fully connected node makes this MC irreducible and we can compute the modified ranks using the MATLAB script below.

```
% Delete all variables and close figures
clear all
close all

% Load collaboration matrix
collaboration_matrix;

% Include fully connected node to graph (node "J+1")
graph = [graph ones(size(graph,1),1) ; ones(1,size(graph,1)) 0];

J = size(graph,1);  % Number of states
D = sum(graph);     % Node degrees
N = 100;            % Length of simulation

P = zeros(J,J);
for n = 1:J
    if D(n) > 0
        P(n,:) = graph(n,:) / D(n);
    end
end

% Probability update
p = ones(J,1)/J;

% Also known as p = (P')^N*p...
for n = 1:N
    p = P'*p;
end

% Compute ranks
ranks = p;

% Display ranks
figure();
bar(1:J, ranks);
xlabel('Node');
ylabel('Rank');
xlim([0.5 J-1+0.5]);
```

```
grid;


%%% Export figure %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
set(gcf,'Color','w');
export_fig -q101 -pdf HW6_D2.pdf
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Results are shown in Figure 5.

E Recast as a system of linear equations. Focusing on the modified, irreducible graph, we can compute the node ranks by finding the limit distribution of the MC. To do so, we can follow the derivation on slide 64 of markov_chains and solve the following system of linear equations:

$$\begin{bmatrix} \boldsymbol{\pi} \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{P}^T \\ \mathbb{1}^T \end{bmatrix} \boldsymbol{\pi} \tag{1}$$

To solve this system in MATLAB, notice that (1) can also be written as the homogeneous system

$$\begin{aligned} \left(\boldsymbol{I} - \boldsymbol{P}^T\right) \boldsymbol{\pi} &= 0 \\ \mathbb{1}^T \boldsymbol{\pi} &= 1 \end{aligned}$$

which implies that $\boldsymbol{\pi}$ is in the null space of $\left(\boldsymbol{I} - \boldsymbol{P}^T\right)$. We can use this property to find a solution of (1) using the following MATLAB script:

```
% Delete all variables and close figures
clear all
close all

% Load collaboration matrix
collaboration_matrix;

% Include fully connected node to graph (node "J+1")
graph = [graph ones(size(graph,1),1) ; ones(1,size(graph,1)) 0];

J = size(graph,1);  % Number of states
D = sum(graph);     % Node degrees

P = zeros(J,J);
for n = 1:J
    if D(n) > 0
        P(n,:) = graph(n,:) / D(n);
    end
end

% Compute ranks
null_vector = null(eye(J) - P');
pi = null_vector/sum(null_vector);

% Display ranks
figure();
bar(1:J, pi);
xlabel('Node');
ylabel('Limit probability');
xlim([0.5 J-1+0.5]);
grid;


```

```
%%% Export figure %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
set(gcf,'Color','w');
export_fig -q101 -pdf HW6_E.pdf
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Results are shown in Figure 6.

F Recast as an eigenvalue problem. Notice that the top part of (1) reads

$$\boldsymbol{\pi} = \boldsymbol{P}^T \boldsymbol{\pi}. \tag{2}$$

In other words, $\boldsymbol{\pi}$ is related to the eigenvector of $\boldsymbol{P}^T$ corresponding to the eigenvalue one (refer to slide 18 of ranking_nodes_in_graphs). This eigenvector can be computed in MATLAB using the following script. Be careful: MATLAB returns eigenvectors normalized to have unit norm. Hence, to obtain $\boldsymbol{\pi}$ you must normalize the returned eigenvector to be a probability distribution.

```
% Delete all variables and close figures
clear all
close all

% Load collaboration matrix
collaboration_matrix;

% Include fully connected node to graph (node "J+1")
graph = [graph ones(size(graph,1),1) ; ones(1,size(graph,1)) 0];

J = size(graph,1);  % Number of states
D = sum(graph);     % Node degrees

P = zeros(J,J);
for n = 1:J
    if D(n) > 0
        P(n,:) = graph(n,:) / D(n);
    end
end

% Compute ranks
[V,D] = eig(P');
eigenvector = V(:,1);
pi = eigenvector/sum(eigenvector);

% Display ranks
figure();
bar(1:J, pi);
xlabel('Node');
ylabel('Limit probability');
xlim([0.5 J-1+0.5]);
grid;


%%% Export figure %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
set(gcf,'Color','w');
export_fig -q101 -pdf HW6_F.pdf
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Results are depicted in Figure 7.

G Compare implementations. (Refer to slides 18, 22–28 of ranking_nodes_in_graphs.)

- *Random walk*: the main advantage of this approach is that it evaluates the ranks in a distributed fashion, i.e., each node can determine its own rank without the need for a central unit to aggregate all the information. This method is therefore "secure," as almost no information is shared between nodes. For any node to compute its rank it needs only know how many neighbors it has and how many time steps have passed (which can be easily done by token passing).

- *Probability update*: using probability propagation has similar advantages to the random walk method in that implementation can be distributed and there can be limited sharing of information between nodes. However, its main advantage is that it converges to the true ranks significantly faster than the random walk implementation (how fast it converges actually depends on the graph). Indeed, observe that the ranks of all states are updated at each iteration, unlike the random walk approach where only one node rank is updated at each step. Moreover, the updates can be halted at any iteration to provide an approximation of the ranks of the nodes, in contrast to the system of linear equations or eigenvector approaches[1]. This is particularly important for large MCs. Also notice that the algorithm reduces to a simple matrix-vector multiplication, for which there exist very fast algorithms, both centralized and distributed (especially if the matrix is *sparse*, i.e., has many zero elements, which is typically the case with graphs). In fact, this is MATLAB's specialty: it is designed to perform matrix computations exceptionally fast.

- *System of linear equations*: this approach mitigates the slow convergence issue since it does not (necessarily) depends on iteration. Nevertheless, all ranks must be computed centrally, which compromises the privacy of the nodes. Moreover, for large networks, it may be impossible to directly compute all ranks since even the matrix $\boldsymbol{P}$ alone may be too large to even fit in memory. Again, methods do exist to address some of these issues (this is a whole area of numerical methods), but they are beyond the scope of this course.

- *Eigenvalue problem*: this approach is very similar to the system of linear equations one and therefore has similar advantages and disadvantages.

[1]This is not completely true. There exist iterative linear systems and eigenvalue problem solvers (e.g., Gauss-Jacobi iterations and Lanczos algorithm), some of which MATLAB actually uses (although it makes it look like they aren't iterative by just spitting out an answer).

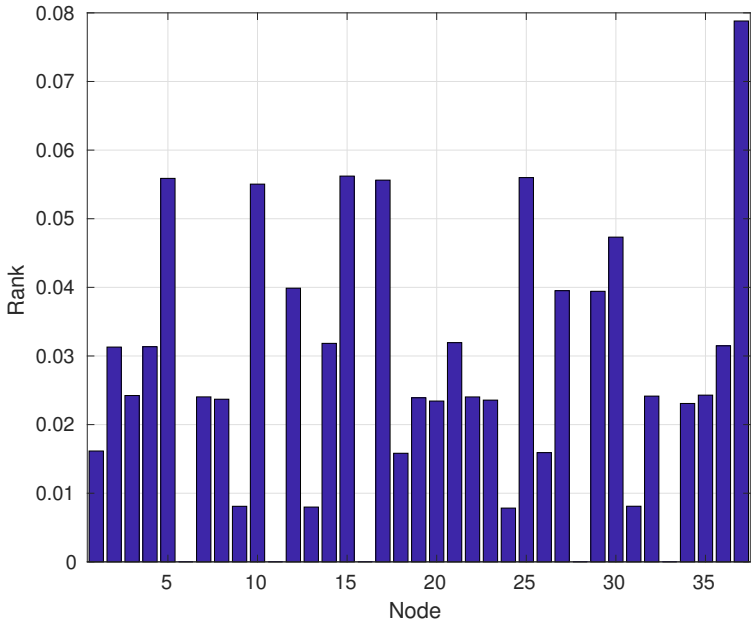

Figure 1: Ranks evaluated using the random walk method for $N = 10^6$ iterations on the reducible graph with initial state $A_0 = 1$ (Part B).

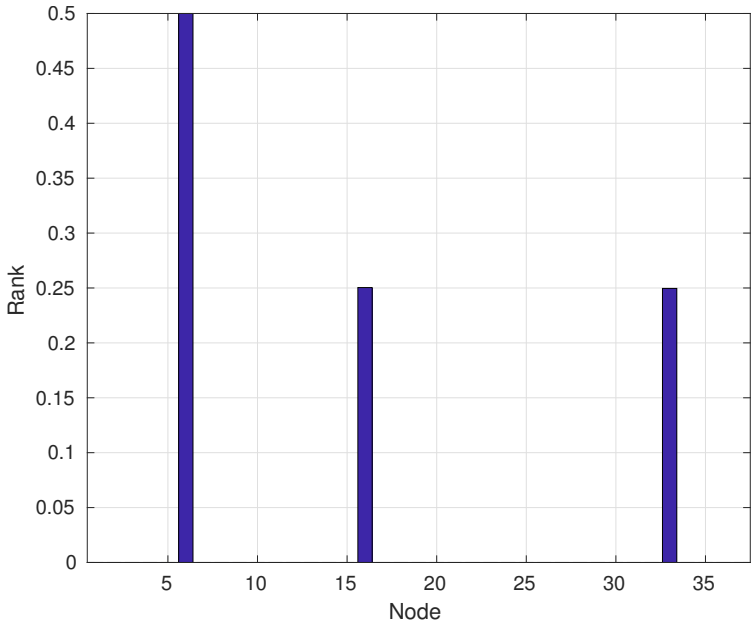

Figure 2: Ranks evaluated using the random walk method for $N = 10^6$ iterations on the reducible graph with initial state $A_0 = 6$ (Part B).

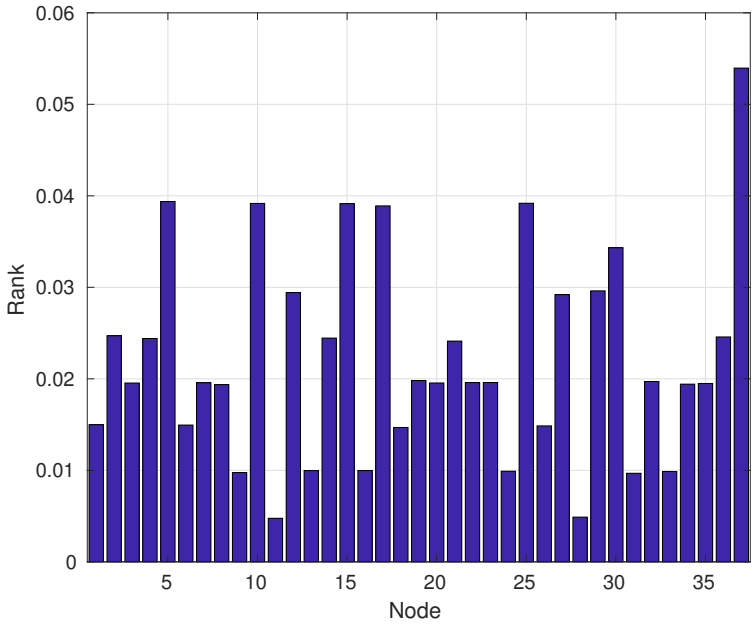

Figure 3: Ranks evaluated using the random walk method for $N = 10^6$ iterations on the irreducible graph with initial state $A_0 = 1$ (Part B).

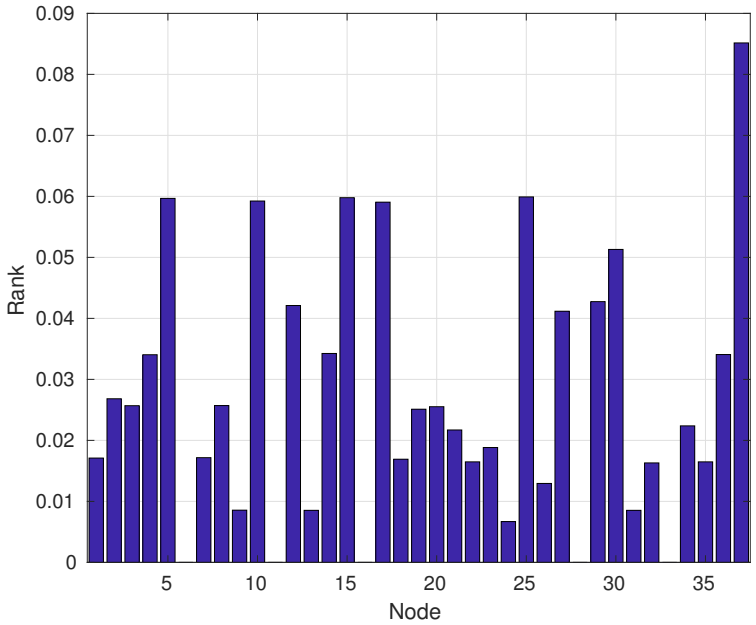

Figure 4: Ranks evaluated using the probability update method for $N = 100$ iterations on the reducible graph where the initial probability vector $\boldsymbol{p}(0)$ has all elements equal to zero except for the first one, which is equal to one (Part D).

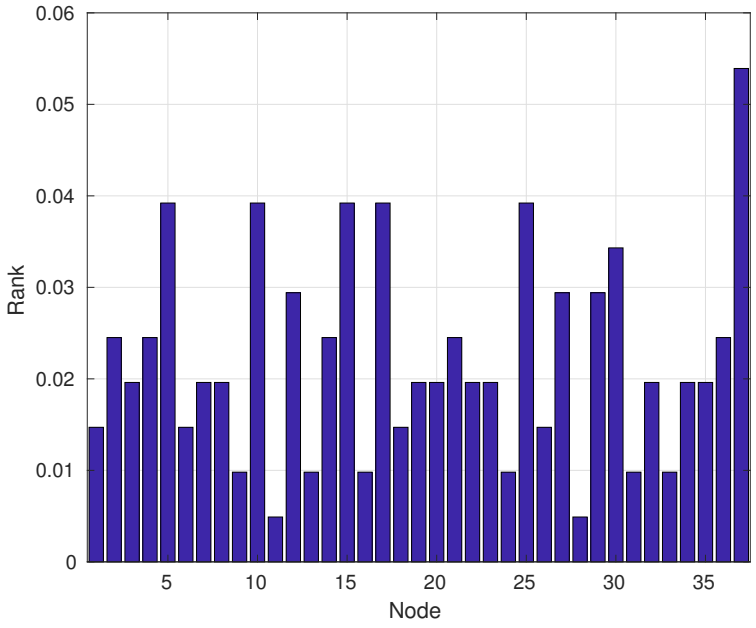

Figure 5: Ranks evaluated using the probability update method for $N = 100$ iterations on the irreducible graph where the initial probability vector $\boldsymbol{p}(0) = \mathbb{1}/J$ (Part D).

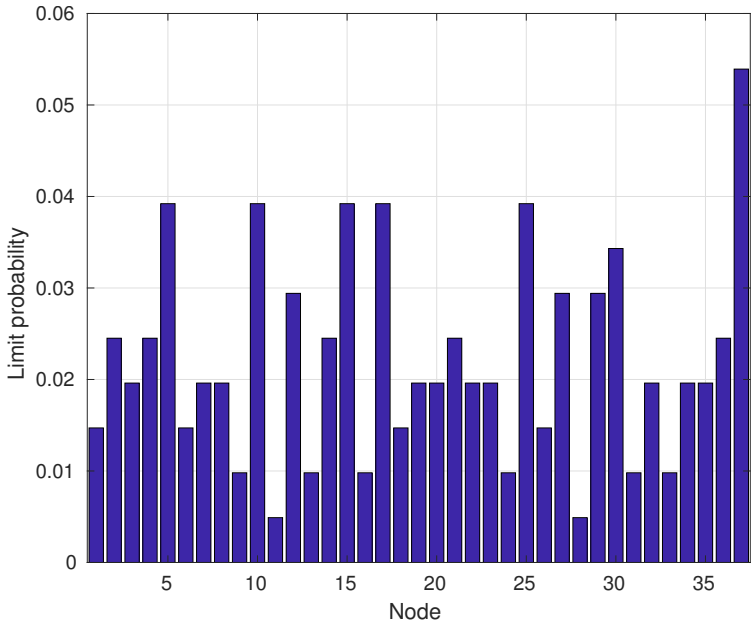

Figure 6: Ranks evaluated using the linear system of equations method on the irreducible graph (Part E).

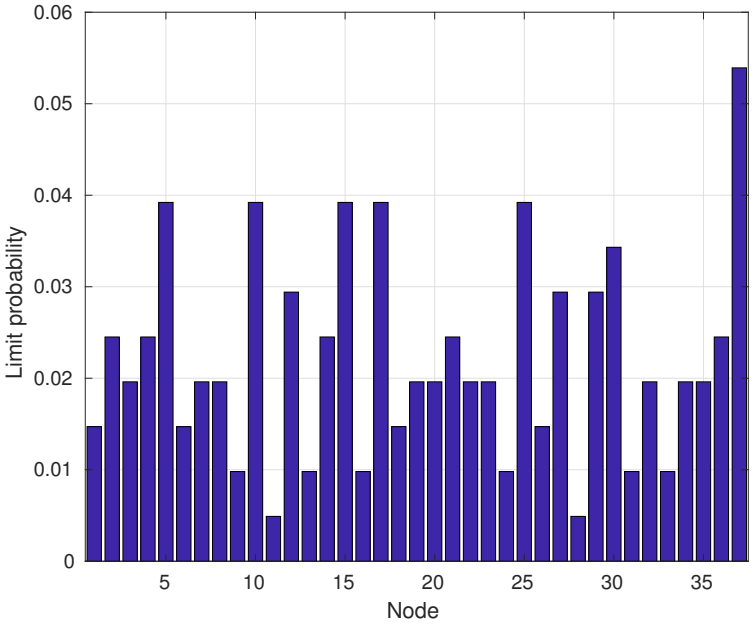

Figure 7: Ranks evaluated using the eigenvalue method on the irreducible graph (Part F).