```
%%% State transition matrix (STM) and state transition graph (STG) generation

% All possible states
numberNetworkNodes = 6; % Number of nodes

convertToBinary = 0:(2^numberNetworkNodes - 1); % Generate all integers from 0
to 64 - 1

binaryStatesChar = dec2bin(convertToBinary, numberNetworkNodes); % Convert
integers to binary

binaryStatesCell = cellstr(binaryStatesChar); % Convert binary states character
vector to cell

% All update order permutations
order = 123456; % Default update order

orderArray = arrayfun(@(x) str2double(x), num2str(order)); % Node values:
Convert number to string, then convert each string element to number

allOrders = perms(orderArray); % All permutations of update order

% Dictionary to map network-state to index
indices = 1:64; % Indices for states

numericStates = str2num(binaryStatesChar); % Numeric forms of network-states

numericStates = numericStates'; % Column vector to row vector for dictionary

statesDictionary = dictionary(numericStates, indices); % Create dictionary

% Function to generate state transition matrix and build the State Transition
Graph (STG)
function [T, G, source, target] = generateStmStg(allOrders, binaryStatesCell,
statesDictionary) % Define function

% Dictionary to map number of update order permutations to final state obtained
finalStates = []; % Dictionary keys

numberPermutations = []; % Dictionary values

permutationsDictionary = dictionary(finalStates, numberPermutations); %
Initialize dictionary

% Initialize source graph-nodes vector
source = [];

% Initialize target graph-nodes vector
target = [];

% Initialize array for STM
T = zeros(64);
```

```
% Nested loops to fill up T matrix
for initialState = 1:64 % Iterate through all initial states

    for inOrders = 1:length(allOrders) % Iterate through all update orders

        chosenOrder = allOrders(inOrders, :); % Extract update order

        sii = binaryStatesCell{initialState}; % Extract initial state

        si = sii; % Initialize and update transient state

        sj = si; % Initialize and update final state

        for inOrder = 1:6 % Iterate through all positions in chosen update order

            % Define network-node function
            if chosenOrder(inOrder) == 1
                % f1(v2, v5: T = 1) = NOT(v2) OR NOT(v5)
                condition_1 = si(2) == '0' || si(5) == '0';

                % Update final state
                if condition_1 == 1
                    sj(1) = '1';
                elseif condition_1 == 0
                    sj(1) = '0';
                end
                si = sj; % Update initial state


            % Define network-node function
            elseif chosenOrder(inOrder) == 2
                % f2(v3, v4: T = 1) = NOT(v3) OR NOT(v4)
                condition_2 = si(3) == '0' || si(4) == '0';

                % Update final state
                if condition_2 == 1
                    sj(2) = '1';
                elseif condition_2 == 0
                    sj(2) = '0';
                end
                si = sj; % Update initial state

            % Define network-node function
            elseif chosenOrder(inOrder) == 3
                % f3(v1, v6: T = 3) = v1 AND NOT(v6)
                condition_3 = si(1) == '1' && si(6) == '0';

                % Update final state
                if condition_3 == 1
                    sj(3) = '1';
                elseif condition_3 == 0
                    sj(3) = '0';
                end
                si = sj; % Update initial state
            % Define network-node function
            elseif chosenOrder(inOrder) == 4
                % f4(v3, v2, v5: T = 3) = v3 AND NOT(v2) OR NOT(v5)
                condition_4 = (si(3) == '1' && si(2) == '0') || si(5) == '0';

                % Update final state
                if condition_4 == 1
                    sj(4) = '1';
                elseif condition_4 == 0
                    sj(4) = '0';
                end
                si = sj; % Update initial state

            % Define network-node function
            elseif chosenOrder(inOrder) == 5
                % f5(v4: T = 2) = NOT(v4)
                condition_5 = si(4) == '0';

                % Update final state
                if condition_5 == 1
                    sj(5) = '1';
                elseif condition_5 == 0
                    sj(5) = '0';
                end
                si = sj; % Update initial state

            % Define network-node function
            elseif chosenOrder(inOrder) == 6
                % f6(v3, v4: T = 2) = NOT(v3) OR NOT(v4)
                condition_6 = si(3) == '0' || si(4) == '0';

                % Update final state
                if condition_6 == 1
                    sj(6) = '1';
                elseif condition_6 == 0
                    sj(6) = '0';
                end
                si = sj; % Update initial state

            end

        end

        sjNumeric = str2num(sj); % Numeric final state

        % Update permutations dictionary
        if isKey(permutationsDictionary, sjNumeric) % For final state is seen

            permutationsDictionary(sjNumeric) =
permutationsDictionary(sjNumeric) + 1; % Add to number of permutations

        else
```

```
            permutationsDictionary(sjNumeric) = 1; % Add new entry to dictionary

        end

        % Add values to STM
        indexSTM = statesDictionary(sjNumeric); % Extract index of current final
state from states dictionary

        T(initialState, indexSTM) = permutationsDictionary(sjNumeric)/
length(allOrders); % Add probability to T

        % Check if source-target nodes pair is already present in respective
vectors
        sourceNodeValue = statesDictionary(str2num(sii)); % Representative value
of initial state in states dictionary

        targetNodeValue = statesDictionary(sjNumeric); % Representative value of
final state in states dictionary

        if ~isempty(source) % Check source-target pair presence only if vectors
aren't empty

            isDuplicate = any(source == sourceNodeValue & target ==
targetNodeValue); % Logical true for both presence true

        else

            isDuplicate = false; % First pair is always unique

        end

        % Add graph-nodes to source and target vectors
        if ~isDuplicate % For no source-target pair found

            source = [source, sourceNodeValue]; % Update source vector

            target = [target, targetNodeValue]; % Update target vector

        end

    end

    % Calculate row sums in STM (sum of all row probabilities should be 1)
    if inOrders == 720 % For last iteration for current initial state

        rowSummation = T(initialState, :); % Extract row probabilities

        rowSums(initialState) = sum(rowSummation); % Add to row sums vector

    end

    % Reset permutations dictionary for next initial state
    numberPermutations = []; % Reset number of permutations
    finalStates = []; % Reset final states

    permutationsDictionary = dictionary(finalStates, numberPermutations); %
Rebuild dictionary

end

% Display sizes of source nodes and target nodes vectors to find number of edges
disp(['Size of source: ', int2str(size(source))]); % Size of source nodes array
disp(['Size of target: ', int2str(size(target))]); % Size of source nodes array

% Display size of STM for sanity check
disp(['Size of STM: ', int2str(size(T))]);

% Display row sums matrix for sanity check (all row probabilities sum to 1)
disp("Sum of rows:");
disp(rowSums);

% Build table with STM
colHeaders = binaryStatesCell; % Define column headers

rowHeaders = binaryStatesCell; % Define row headers

STMTable = array2table(T, 'VariableNames', colHeaders, 'RowNames', rowHeaders);
% Convert array to table with headers

% Display STM
disp("State Transition Matrix:");
disp(STMTable);

% Create STG
G = digraph(source, target);

% Display STG
disp("State Transition Graph:");

% Plot STG
graphPlot = plot(G, 'Layout', 'force');

end

% Generate STM and STG
[T, G, source, target] = generateStmStg(allOrders, binaryStatesCell,
statesDictionary);
```

Output:

```
Size of source: 1   413

Size of target: 1   413

Size of STM: 64   64
```

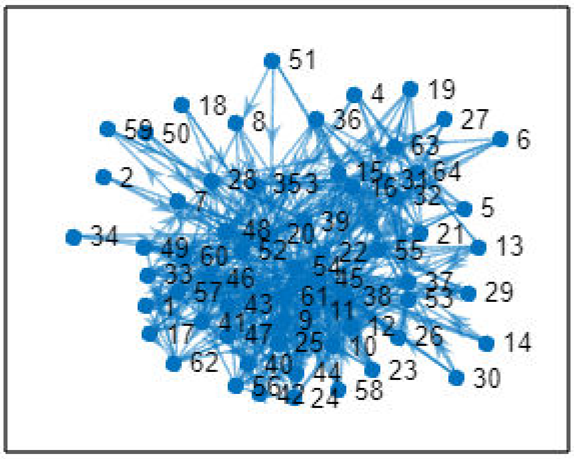
```
Sum of rows:

<1 x 64 double>

State Transition Matrix:

<64 x 64 table>

State Transition Graph:
```


51
4
19
18
8
36
27
59
50
6
63
64
15
2
28
353
16
31
32
7
5
48
20
39
21
34
49
60
52
22
55
13
33
46
54
45
37
57
43
61
38
53
29
1
41
9
11
12
17
47
25
10
26
14
62
40
44
23
30
56
42
24
58