Network Security and Artificial Intelligence

Laura Watkins, laura.watkins@gccaz.edu, Glendale Community College, AZ, AMATYC

Artificial Intelligence (AI)[1] can play a crucial role in improving network security. In today's digital environment where cyber threats are becoming increasingly sophisticated and pervasive, AI can enable faster and more accurate threat detection due to its ability to rapidly analyze and interpret large volumes of network data and identify unusual patterns. As a result, AI can identify potential threats before significant damage is caused and free up human resources to focus on more complex issues.

Graph Theory, a relatively new area of mathematics useful in nearly every branch of science, is important to both AI and network security. Graphs are a natural tool to represent computer networks where devices and connections can be represented. Some key activities related to network security in which AI's use of graph theory is essential include:

- **Anomaly Detection:** Graph-based representations are a tool used to identify unusual patterns which may indicate a security breach or a network attack.
- **Vulnerability Assessment:** Graph centrality measures are tools used to determine the importance of nodes within a graph and when considering the role of AI in network security these measures help to identify critical nodes or potential weak points in a network.
- **Attack Path Analysis:** Graph transversal techniques can be used by AI to simulate and predict potential attack paths through a network, which can aid in risk assessment and mitigation.

Graph theory offers a mathematical framework that translates complex network interactions into comprehensible structures, empowering AI algorithms to more precisely interpret and respond to security challenges, ultimately creating more intelligent and responsive defense mechanisms. In this module, you will be introduced to some basics of graph theory and its application in network security and AI algorithms and explore how graphs can represent networks and how AI algorithms can leverage graph structures to enhance security.

[1] Image generated by deepai.org.

Use a Graph to Represent a Network

What is a graph?

A graph is a fundamental mathematical structure used to represent relationships between objects and is composed of two main components: nodes (also called vertices) and edges. Nodes are points that represent individual entities and edges are lines connecting pairs of nodes indicating a relationship or connection between the two nodes. Graphs can be used to model a wide variety of real-world scenarios, e.g. social networks, transportation systems, supply chains, and computer systems. The edges in a graph can be one-way (directed) or two-way (undirected) and may have weights or costs associated with them. Because of this framework, graphs are a powerful tool for tackling intricate systems, optimizing solutions, and creating algorithms for a variety of applications including network traversal optimization.

An Example Network.

Let's start by considering a small computer network with 5 devices (A, B, C, D, E) or *nodes*. The connections between devices are *edges*. We assume that the connections between devices are undirected. Now suppose that for this network:

- Device A connects to Devices B, D, and E.
- Device B connects to Devices A, C, D, and E.
- Device C connects to Devices B and E.
- Device D connects to Devices A and B.
- Device E connects to Devices A, B, and C.

Next, we draw a diagram of the network nodes and their connections (see Figure 1).

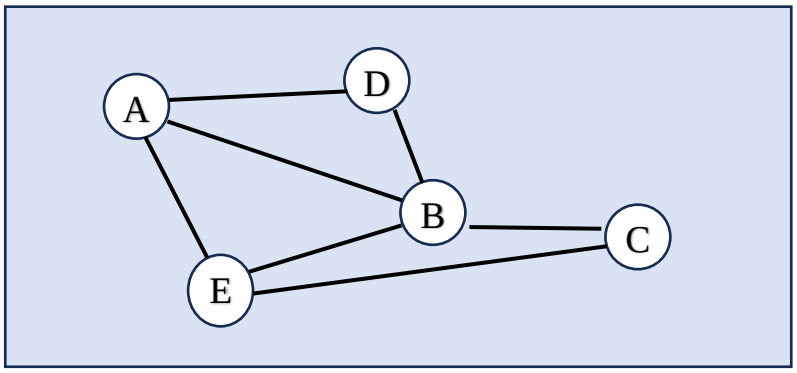


Figure 1. A graph representing a network connecting five devices.

Adjacency Matrix. We can use the graph of a network to create an ***Adjacency Matrix***, also called a connection matrix, which is a compact tool that can be used to represent a finite graph. An adjacency matrix is particularly useful in quickly determining whether two nodes (or vertices) are connected (or adjacent). This tool can facilitate activities such as efficient connectivity analysis, detection of anomalies, and visualization and analysis network segmentation. For a network containing n nodes we create an $n \times n$ matrix, where each node is assigned a row and a column (see Figure 2). We then enter a 0 or a 1 for each element of the matrix to indicate whether the nodes represented by the corresponding row and column are adjacent.

We want to create an adjacency matrix, Table 1, for the network in Figure 1. For each entry in the matrix, we consider the node represented by the row and the node represented by the column.

- If the two nodes are adjacent, meaning there is an edge, or connection, between the two nodes we enter a 1 into the matrix.
- If the two nodes are not adjacent, we enter a 0 into the matrix.

Note that the diagonal of this matrix is 0 because each node is not adjacent to itself. The first row of the matrix indicates that Node A is connected to Nodes B, D, and E.

Question 1. Finish entering the components of this adjacency matrix (Figure 2). What observations can you make about the network by considering the adjacency matrix alone?

	A	***B***	***C***	***D***	***E***
A	0	1	0	1	1
B		0			
C			0		
D				0	
E					0

Table 1. Adjacency matrix for network in Figure 1.

Cycles. A cycle in a graph is a path that starts and ends at the same node (or vertex). A cycle must pass through *at least one other node* without repeating any edges. In other words, a cycle is a closed loop within the graph where you can traverse a series of nodes (or vertices) without using any edge more than once. Cycles play an important role in graph theory and have significant implications for computer science and mathematics. Cycles within a network graph can, for example,

- Provide redundancy by providing alternative paths for data transmission.
- Ensure that a network remains connected in the event of a node or connection failure.
- Create potential security vulnerabilities by providing multiple attack paths.

Thus, understanding cycles in network graphs is important in designing and maintaining a robust, efficient, and secure network architecture.

Question 2. Find all cycles in the graph of the network in Figure 1. *For example: A – D – B – A.*

Shortest Path Algorithm

Now imagine an attacker is trying to find the quickest route through the network. This would be considered the "shortest path" based on "distance". One computational method that can determine the shortest path between nodes is the **Dijkstra algorithm** in which "distance" represents the cost of reaching from one node to another, rather than the physical distance between nodes.

To illustrate how this algorithm works, we consider our network from Figure 1 and assign weights to the edges (connections) representing the time it takes to traverse that connection. For this illustration, we will assign weight values between 1 and 10 to the edges of the graph (see Figure 2). To keep track of traversing the network we will use tables to capture progress towards the target node, and the path by which we arrive at a specific node along the path.

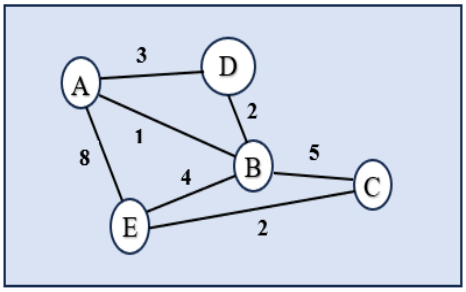


Figure 2. A weighted graph representing a network with five devices.

Suppose we wish to find the shortest path from Node A to Node C, that is the path with the least "cost". Starting at Node A, the connecting nodes D, B, and E with their corresponding costs are entered into Table 1.

Node	Total Cost	Path
D	3	From A
B	1	From A
E	8	From A

Table 2. Initial Table.

The nodes accessible from Node A are then sorted in terms of increasing cost so that the node with the least cost, Node B, rises to the top of the table (see Table 2). Since we haven't reached the destination, Node C, we continue to search.

Node	Total Cost	Path
B	1	From A
D	3	From A
E	8	From A

Table 3. Sorted Table.

Since Node B was the least cost, we now consider paths from Node B that lead to the goal, Node C. Node D is a neighbor of Node B, but since we already considered (or visited) that node so we drop it from consideration. Thus, we consider only Nodes C and E, and we include the cost of reaching Node B in calculating the new costs. To reach Node C the cost will be 1 + 5 = 6. To reach Node E the cost will be 1 + 4 = 5. Note that the cost of reaching Node E by passing through Node B is less than the path from Node A to Node E. So, we update the information in Table 3 and would no longer consider a path directly from Node A to Node C.

Node	Total Cost	Path
C	6	From B
E	5	From B

Table 4. Unvisited Neighbors from Node B

Note that our destination, Node C, is in the table but the total cost associated with Node E is less than the total cost to reach Node C through Node B (5 < 6). So, it is possible that a path through Node E could be as short or shorter than the path from Node B directly to Node C. Reading the graph in Figure 2, we see that to pass through Node E to Node C would add 2 to the total cost (see Table 4). The cost of passing through Node E is 7, which is greater than 6 the already calculated cost to arrive at Node C, and we remove Node E from the search.

Node	Total Cost	Path
C	6	From B
C	7	From E

Table 5. Total cost for Node C.

Recall that Node D had the second lowest cost so we could repeat this search process. From Node D we could reach Node B with a total cost of 3 + 2 = 5 (see Table 6). To reach Node C there would be an additional cost of 5 so the total cost of a pathway through Node D would be 10, which is greater than the cost passing through Node B.

Node	Total Cost	Path
B	5	From D

Table 6. Cost for starting an alternate path.

In our example, we reached Node C from Node B, Node B from Node A with the least cost. Therefore, the shortest route from Node A to Node C is: $A \rightarrow B \rightarrow C$.

Here is an outline of Dijkstra's algorithm as we implemented it using tables to keep track of the path and the corresponding cost[2].

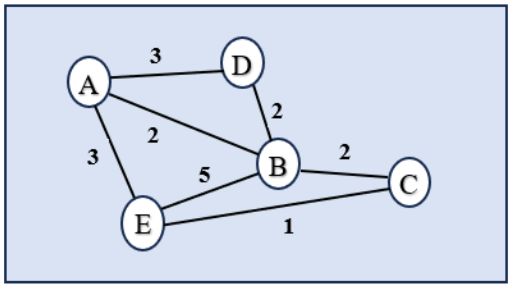
1. Start at the initial node.
2. From the initial node examine its neighboring nodes. Calculate their tentative distances from the initial node. Enter the nodes and their corresponding costs into a table, then sort the table from the least to the greatest cost.
3. Using the node with the least cost (current node), we now consider only its neighboring nodes that have not already been visited (considered) and calculate the tentative cost. Remember to include in the calculation the cost of arriving at the current node. If the current calculated distance for the neighboring nodes is less than what was previously recorded, update the distance (see Tables 3 & 4). Record this information along with the path in a table.

[2] For pseudocode for implementing this algorithm see: https://math.libretexts.org/Courses/SUNY_Schenectady_County_Community_College/Discrete_Structures/10%3A_Graph_Theory/10.07%3A_Weighted_Graphs_and_Dijkstra's_Algorithm

4. If the destination node is not in the table, sort the table from the least to the greatest.
5. Repeat Step 3 and Step 4 until the destination node is represented.
6. Check additional nodes that have not been removed from the search using the process in Steps 3 and 4 to determine whether a shorter path exists.

To summarize, Dijkstra's algorithm finds the shortest path between two nodes on a network by systematically exploring the closest unvisited nodes. It starts with a beginning node and gradually builds the shortest path by checking the neighboring nodes distances, always choosing the path with the smallest total distance. While the algorithm is running, it updates distances as it goes, which ensures that by the time it reaches the destination, it has found the most efficient path through the network.

Question 3. Implement Dijkstra's algorithm on the following network to find the shortest path from Node D to Node E.



Network Vulnerability Analysis

A critical aspect of cybersecurity is network vulnerability analysis. This involves identifying, quantifying, and prioritizing weaknesses in a computer network and is essential in maintaining the integrity and security of digital infrastructures. We can leverage concepts from graph theory to model networks which allow for a systematic examination that can identify potential vulnerabilities such as high-risk nodes, bottlenecks, and potential attack paths that could be exploited by malicious actors. Understanding the vulnerability analysis of a network is important in developing robust security strategies and proactive measures to protect against cyber threats.

Revisiting the network from Figure 1 (see Figure 3), we will consider the vulnerability of this network

Degree of a node. In graph theory the degree of a node (or vertex) refers to the number of edges connected to that node. For example, Node A in the network in Figure 3 has degree 3 because there are three edges connected to the node.

Question 4. Find the degree of the remaining nodes of the network in Figure 3. Which node has the highest degree? Why might this node be considered a security risk?

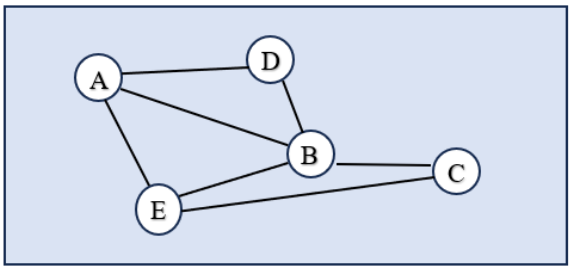


Figure 3. A graph representing a network connecting five devices (copy of Figure 1).

Question 5. Propose a change to the network in Figure 3 that could improve security. Redraw the graph with the change implemented. Briefly explain how the change enhances the security of the network.

Graph Centrality Measures

Graph centrality is a measure of the importance or influence of nodes within a graph. As such, graph centrality measures play a crucial role in network security and AI by identifying the most important nodes within a network which can help with prioritizing security measures. There are a variety of centrality measures that AI can leverage in exploring network security, for example, degree centrality, farness and closeness centrality, eigenvector centrality, and Katz centrality. These measures assist with pinpointing critical nodes or potential weak points in a network structure. Examples of nodes within a computer network with high centrality scores would be key servers, routers, or even workstations that

are vital to the functioning of the network, as well as potential targets for attacks. We look at two types of centrality: degree centrality and farness/closeness centrality.

Degree Centrality

Degree centrality is a measure of the number of connections a node has to other nodes in a network. Essentially, this measure is the degree of the node.

Question 6. Using the graph of the network in Figure 3, find the degree centrality for each of the nodes.

Farness/Closeness Centrality

Farness centrality of a node is the sum of the shortest path distances between one node and all other nodes of the network, whereas the *closeness centrality* of a node is the reciprocal of the farness centrality. A lower farness centrality indicates that a node closer to other nodes in the network and therefore more centrally located. Since closeness centrality is the reciprocal of farness centrality, a node with a lower farness centrality will have a higher closeness centrality.

Referencing the network graph in Figure 4, we can determine farness and closeness centralities for each node. Since this graph is an unweighted graph, we assume that all edges have a distance or cost of 1. Let

f_N = the farness centrality of Node N,

d_{NM} = the distance from Node N to Node M, and

c_N = the closeness centrality of Node N.

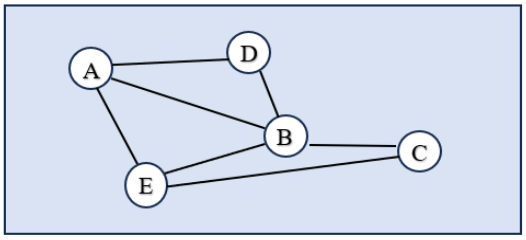


Figure 4. A graph representing a network connecting five devices (copy of Figure 1).

For the network in Figure 5,

farness centrality for Node A is $f_A = d_{AB} + d_{AC} + d_{AD} + d_{AE} = 1+2+1+1 = 5$ and

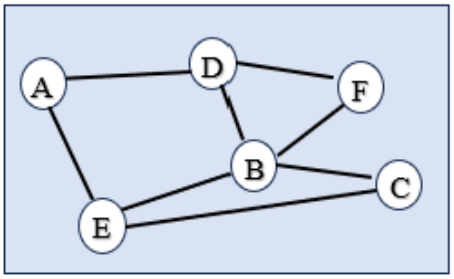
closeness centrality for Node A is $c_A = 1/f_A = 1/5 = 0.20$.

Question 7. Find the farness centrality and the closeness centrality for the remaining nodes of the network in Figure 5. Which node(s) are more central to the network? Why?

In closing, Graph Theory and Artificial intelligence form a powerful intersection of mathematics and technology, particularly when considering network security. Graph Theory provides a framework that can translate complex network interactions into comprehensible structures (nodes and edges). Using graph theory AI is becoming an essential tool for enhancing network security, especially since cyber threats are becoming more advanced. By representing computer networks as graphs, AI can analyze a network searching for anomalies, vulnerabilities, and potential attack paths, thereby identifying threats quickly and accurately by analyzing large amounts of network data. The future of network security lies in leveraging sophisticated mathematical approaches such as graph theory to anticipate, detect, and mitigate potential cyber risks more effectively than ever before.

Apply What You Learned

1. The graph of a computer network is provided. Answer the following questions about this network.



   a. Identify the degree of each node in the network.

   b. Create an adjacency matrix for this network graph.

   c. Identify any cycles in your graph.

d. Find the farness centrality and closeness centrality for each node on the unweighted graph. Which node(s) are more central to the network? Why?

e. Propose one change to the network that could improve security. Redraw the graph with the change implemented. Briefly explain how the change enhances the security of the network.

f. Assign weights, 1 through 10, to each edge in the graph. At least four edges should have a weight greater than 4. Implement Dijkstra's algorithm on this weighted graph to find the shortest path from Node A to Node F.

2. An adjacency matrix for a network is provided below. Answer the following questions about this network.

	A	***B***	***C***	***D***	***E***	***F***
A	0	1	0	1	1	1
B	1	0	1	0	1	0
C	0	1	0	1	0	1
D	1	0	1	0	1	0
E	1	1	0	1	0	1
F	1	0	1	0	1	0

   a. Identify the degree of each node in the network. Describe how the degree of a node can be determined from an adjacency matrix.

   b. Create the graph represented by the adjacency matrix.

   c. Identify any cycles in your graph.

d. Find the farness centrality and closeness centrality for each node on the unweighted graph. Which node(s) are more central to the network? Why?

e. Propose one change to the network that could improve security. Redraw the graph with the change implemented. Briefly explain how the change enhances the security of the network.

f. Assign weights, 1 through 10, to each edge in the graph. At least four edges should have a weight greater than 4. Implement Dijkstra's algorithm on this weighted graph to find the shortest path from Node A to Node C.

References:

Lacey, L. (Ed.) (n.d.) *Discrete Structures, Fifth Edition.* LibreTexts. https://math.libretexts.org/Courses/SUNY_Schenectady_County_Community_College/Discrete_Structures/10%3A_Graph_Theory/10.07%3A_Weighted_Graphs_and_Dijkstra's_Algorithm

Mittal, S. (2023, October 28). *Dijkstra Algorithm in Artificial Intelligence.* Medium https://medium.com/@shivansh20128/dijkstra-algorithm-in-artificial-intelligence-bb7c9ac9c3de#:~:text=To%20put%20it%20in%20short,parameter%20according%20to%20the%20application.