DECOY PATH-BASED OPTIMIZATION ALGORITHM

We provide notations used in the optimization algorithm in the following and then present the algorithm in Algorithm 1.

- m_i: A node in the network for $i \in \{1, ..., n\}$ where n is the total number of nodes
- M: A set of nodes in the network, denoted by $M = \{m_1, ..., m_n\}$
- M_d: A set of decoy nodes in the network, denoted by $M_d \subset M$
- M_r: A set of real nodes in the network, denoted by $M_r \subset M$
- M_t: A set of real nodes considered as targets, denoted by $M_t \subset M$
- M_{dt}: A set of decoy nodes considered as decoy targets, denoted by $M_{dt} \subset M_t$
- e_{m_i,m_j}: An edge (connection) from m_i to m_j
- $m_i.con$: A list of out-degree connections of m_i
- getDecoys(M, ζ): A function that randomly selects a certain number of decoy IoT nodes from all decoy IoT nodes based on a probability ζ where each real IoT node is connected to at least half of the decoy nodes; a returned list is stored in M_d
- checkCon(m_i, m_j): Return true for $e_{m_i,m_j} > 0$; return false otherwise
- c: A cost associated with shuffling edges which is incremented by 1 upon any edge changed (i.e., removal or addition)
- DP_{m_i}: A set of paths toward decoy targets where node m_i is an entry point
- traverseNet(M_r): A function that takes a set of nodes in the network and returns Dic_{M_r} with node m_i as the key and $|DP_{m_i}|$ as the item where $m_i \in M_r$
- Max(Dic_M): A function that returns a list of nodes with maximum $|DP_{m_i}|$
- Min(Dic_M): A function that returns a list of nodes with minimum $|DP_{m_i}|$
- $T_{out-degree}$: A threshold value that constraints the number of outgoing edges to other real nodes of a real node in the network
- getCon(m_i): A function that returns the number of outgoing edges to other real nodes of m_i where $m_i \in M_r$
- removeConWithMinDP(m_i): A function that removes an outgoing edge of m_i where a connected real IoT node m_j has minimum $|DP_{m_j}|$ compared with other connected real IoT nodes
- H_{m_i}: A maximum number of hops from node m_i to the real target
- H_{max}: A threshold value that constraints the maximum number of hops from a node to the real target, considered as maximum path length

In Algorithm 1, we explain what each line does as follows:

line 2: Go through each node in the network.
line 3: Check two conditions: (i) whether node m_i is a real node or not; and (ii) the node is a target or not.
line 4: If node m_i is a real node and not a target (i.e., a real IoT node), we randomly select a certain number of decoy IoT nodes and store them in M_d.
line 5: Go through each node in the network.
line 6: Check two conditions: (i) whether node m_j is a decoy or not; and (ii) whether the node is in the out-degree connection list of m_i or not.
lines 7-8: If node m_j is a decoy and not connected with m_i, we add an edge (connection) from m_i to m_j and increment the cost by 1.
line 11: Go through each node in $mi.con$.
line 12: Check two conditions: (i) whether node m_k is a decoy or not; and (ii) whether the node is a decoy target or not.
lines 13-14: If node m_k is a decoy and not a decoy target, we remove the existing connection from m_i to m_k and increment the cost by 1.

Algorithm 1 Decoy Path-based Optimization Algorithm

1: **procedure** Decoy-Path-Optimization
2:     **for** $m_i \in M$ **do**
3:         **if** $m_i \in M_r \wedge m_i \notin M_t$ **then**
4:             M_d = getDecoys(M, ζ)
5:             **for** $m_j \in M$ **do**
6:                 **if** $m_j \in M_d \wedge$ checkCon(m_i, m_j) == false **then**
7:                     $e_{m_i,m_j} = 1$ ▷ Add an edge
8:                     $c = c + 1$
9:                 **end if**
10:             **end for**
11:             **for** $m_k \in m_i.con$ **do**
12:                 **if** $m_k \in M_d \wedge m_k \notin M_{dt}$ **then**
13:                     $e_{m_i,m_k} = 0$ ▷ Remove an edge
14:                     $c = c + 1$
15:                 **end if**
16:             **end for**
17:         **end if**
18:     **end for**
19:     Dic_{M_r} = traverseNet(M_r)
20:     **for** $m_i \in M_r$ **do**
21:         **if** $m_i \in$ Min(Dic_{M_r}) $\wedge m_i \notin M_t$ **then**
22:             **for** $m_j \in M_r$ **do**
23:                 **if** $m_j \in$ Max(Dic_{M_r}) $\wedge m_j \notin m_i.con \wedge m_j \notin M_t$ **then**
24:                     **if** $getCon(m_i) > T_{out-degree}$ **then**
25:                         removeConWithMinDP(m_i) ▷ Remove an edge
26:                         $c = c + 1$
27:                     **end if**
28:                     **if** $getCon(m_i) <= T_{out-degree} \wedge H_{m_j} <= H_{max}$ **then**
29:                         $e_{m_i,m_j} = 1$ ▷ Add an edge
30:                         $c = c + 1$
31:                     **end if**
32:                 **end if**
33:             **end for**
34:         **end if**
35:     **end for**
36: **end procedure**

line 19: In the second step, we first compute a dictionary with each real IoT node as the key and number of decoy paths as the item.
line 20: Go through each real node in the network.
line 21: Check two conditions: (i) whether node m_i has the minimum number of decoy paths or not; and (ii) whether the node is a real target or not.
line 22: Go through each real node in the network.
line 23: Check three conditions: (i) whether node m_j has the maximum number of decoy paths or not; and (ii) whether the node is in out-degree connection list of m_i or not; (iii) whether the node is a real target or not.

line 24: Check whether node m_i exceeds the threshold value of outgoing edges to other real nodes.
line 25-26: Remove an outgoing edge (connection) of m_i and increment the cost by 1.
line 28: Check two conditions: (i) whether node m_i exceeds the threshold value of outgoing edges to other real nodes; and (ii) whether the maximum hop count of node m_j exceeds the threshold value.
lines 29-30: Add an edge (connection) from m_i to m_j and increment the cost by 1.