Appendix I

Algorithm 1 A graph privacy-preserving algorithm achieving k-anonymity by adding edges. Based on the idea of k-anonymity, we define the concept of the equivalence class of structural features. The equivalence class of structural feature, such as degree and hub fingerprint, refers to a combination of elements with the same feature. The elements can be nodes, node pairs and so on.

Input: A set comprising the equivalence classes of structural features: S; A set comprising the Δk of equivalence classes: ΔS; ▷ Δk is the difference between the target number of elements in the equivalence class and the actual number of elements in the equivalence class.

Output: A set of edges to be added: *edgeAddList*;

```
function GraphPrivacyPreserve(S, ΔS)
    totalNodeInfo = []
    for s in S do
        (pullCost, pullNodeInfo) = Pull(S, ΔS, s)  ▷ Make the current equivalence class satisfy k-anonymity by adding edges to the elements in other equivalence classes and transferring them into s
        (pushCost, pushNodeInfo) = Push(S, ΔS, s)  ▷ Make the number of elements in the current equivalence class to be 0, by adding edges to the elements in s and transferring them into other equivalence classes
    end for
    if pullCost = pushCost = ∞ then
        return None
    end if
    if pullCost <= pushCost then
        totalNodeInfo += pullNodeInfo
    else
        totalNodeInfo += pushNodeInfo
    end if
    edgeAddList = Convert(totalNodeInfo)
    return edgeAddList
end function

function Pull(S, ΔS, s)
    Δk_c = ΔS[s], pullCount = 0, pullCost = 0, pullNodeInfo = []
    pullSet = GetPullSet(s)        ▷ Get a set of equivalence classes that can transfer elements to s
    for s_j in pullSet do
        Δk_j = ΔS[s_j]
        if Δk_j < 0 then
            (edgeCost, priorityCost, nodeInfo) = Transfer(s_j, s, isLocked = false)     ▷ Transfer adequate unlocked idle elements in the s_j to s and calculate the number of added edges, the cost of priority of nodes and the node information that need to be added to the current solution
            pullCost += Combine(edgeCost, priorityCost)
            pullNodeInfo += nodeInfo
            if size(nodeInfo) <= Δk_c − pullCount then
                pullCount += size(nodeInfo)
            else
                pullCount = Δk_c
                break
            end if
        end if
    end for
    if pullCount != Δk_c then
```

```
        pullCost = ∞
    end if
    return (pullCost, pullNodeInfo)
end function

function PUSH(S, ΔS, s)
    Δk_c = ΔS[s], pushCount = 0, pushCost = 0, pushNodeInfo = []
    pushSet = GetPushSet(s)        ▷ Get a set of equivalence classes that can transfer elements of s to
    for s_j in pushSet do
        Δk_j = ΔS[s_j]
        if Δk_j > 0 then
            if Δk_j >= pushCount then
                transferCount = pushCount
            else
                transferCount = Δj
            end if
            [edgeCost, priorityCost, nodeInfo] = Transfer(s, s_j, transferCount)        ▷ Transfer transferCount elements in the current equivalence class to s_j
            pushCost += Combine(edgeCost, priorityCost)
            pushNodeInfo += nodeInfo
            pushCount -= transferCount
            if pushCount == 0 then
                break
            end if
        end if
    end for
    if pushCount != 0 then
        pushCost = ∞
    end if
    return (pushCost, pushNodeInfo)
end function
```