直接去匿名化算法 (DDA)

plaintext
复制代码
Algorithm 1: Direct De-anonymization (DDA) Algorithm
Input: Node sets VA and VB; link sets EA and EB.
Output: The optimal matching function $\pi 0 \in VA \times VB$.

1: Compute the distance matrices DA and DB.
2: l ← the maximal value in matrices DA and DB, as the collective level in the CDA algorithm.
3: π = CDA(DA, DB, l).
4: Compute the f1 score under π with Eqns. (2) and (3).
5: for each (uA, uB) $\in$ π do
6: Compute the f2 score under $\pi \setminus$ (uA, uB).
7: if f2 > f1 then
8: $\pi \leftarrow \pi \setminus$ (uA, uB); f1 ← f2;
9: return $\pi 0 \leftarrow \pi$.

解释

1. **输入**:
    1. VA 和 VB：两个网络的节点集合。
    2. EA 和 EB：两个网络的链接集合。
2. **输出**:

    1. 最优匹配函数 π0。

3. **步骤**:

    1. **计算距离矩阵 DA 和 DB**: 这些矩阵用于表示节点之间的距离（通常是最短路径距离）。
    2. **确定集合层次 l**: l 是距离矩阵中的最大值，表示集体去匿名化算法中的集合层次。
    3. **使用 CDA 算法计算初始匹配 π**: CDA 算法用于初步匹配节点。
    4. **计算初始匹配的 f1 得分**: f1 得分结合了链接精度和链接召回率。
    5. **优化匹配**: 遍历每个匹配对 (uA, uB)，计算在移除该对后的新的 f2 得分。如果 f2 得分大于初始 f1 得分，则移除该对。
    6. **返回最终匹配结果 π0**: 经过优化后的最优匹配结果。

基于渗透的去匿名化算法 (PDA)

plaintext

复制代码

```
Algorithm 2: Percolation De-anonymization (PDA) Algorithm
Input: Node sets VA and VB; link sets EA and EB.
Output: The optimal matching function π0 ∈ VA × VB.

1: Initialize: The degree threshold dT for seed discovery; The percolation
threshold r.
2: Compute the distance matrices D'A and D'B for nodes whose degrees are
higher than dT.
3: π = CDA(D'A, D'B, l).
4: Compute the f1 score under π for the sub-networks composed of
high-degree nodes.
5: for each (uA, uB) ∈ π do
6:   Compute the f2 score under π \ (uA, uB).
7:   if f2 > f1 then
8:     π ← π \ (uA, uB); f1 ← f2.
9: π = PGM(EA, EB, π, r);
10: return π0 ← π.
```

解释

1. **输入**:
    1. VA 和 VB：两个网络的节点集合。
    2. EA 和 EB：两个网络的链接集合。
2. **输出**:
    1. 最优匹配函数 π0。
3. **步骤**:
    1. **初始化**: 设置度阈值 dT 用于种子节点发现；设置渗透阈值 r。
    2. **计算高于度阈值的节点的距离矩阵 D'A 和 D'B**: 这些矩阵用于表示高于度阈值的节点之间的距离。
    3. **使用 CDA 算法计算高阶节点的初始匹配 π**: CDA 算法用于初步匹配高阶节点。
    4. **计算子网络的 f1 得分**: f1 得分结合了链接精度和链接召回率，仅针对高阶节点的子网络。
    5. **优化匹配**: 遍历每个匹配对 (uA, uB)，计算在移除该对后的新的 f2 得分。如果 f2 得分大于初始 f1 得分，则移除该对。
    6. **使用 PGM 算法进行全局匹配**: PGM（渗透图匹配）算法用于全局优化匹配结果。
    7. **返回最终匹配结果 π0**: 经过优化后的最优匹配结果。

这两个算法分别针对小规模和大规模网络，优化了去匿名化过程中的匹配精度和效率。

直接去匿名化算法 (DDA)

python
复制代码

```
import numpy as np
def compute_distance_matrix(network):
    # 使用 Floyd-Warshall 算法计算距离矩阵
    size = len(network)
    dist = np.full((size, size), np.inf)
    np.fill_diagonal(dist, 0)

    for i in range(size):
        for j in range(size):
            if network[i][j] != 0:
                dist[i][j] = network[i][j]

    for k in range(size):
        for i in range(size):
            for j in range(size):
                if dist[i][j] > dist[i][k] + dist[k][j]:
                    dist[i][j] = dist[i][k] + dist[k][j]

    return dist
def CDA(DA, DB, l):
    # 集体去匿名化算法的占位实现
    # 需要提供具体的实现细节
    # 这里简单返回一个随机匹配
    size = len(DA)
    return {i: i for i in range(size)}
def f_score(π, EA, EB):
    # 计算 f1 得分
    E_match = sum(1 for (uA, vA), (uB, vB) in zip(EA, EB) if π.get(uA)
== uB and π.get(vA) == vB)
    E_align = len(EA)
    E_true = len(EB)

    link_precision = E_match / E_align
    link_recall = E_match / E_true
```

```
    return 2 * (link_precision * link_recall) / (link_precision +
link_recall)
def DDA(VA, VB, EA, EB):
    DA = compute_distance_matrix(EA)
    DB = compute_distance_matrix(EB)
    l = max(np.max(DA), np.max(DB))

    π = CDA(DA, DB, l)
    f1 = f_score(π, EA, EB)

    for (uA, uB) in list(π.items()):
        π_temp = π.copy()
        del π_temp[uA]
        f2 = f_score(π_temp, EA, EB)

        if f2 > f1:
            π = π_temp
            f1 = f2

    return π
# 示例网络数据
VA = [0, 1, 2]
VB = [0, 1, 2]
EA = [[0, 1, 0], [1, 0, 1], [0, 1, 0]]
EB = [[0, 1, 0], [1, 0, 1], [0, 1, 0]]
# 执行 DDA 算法
π0 = DDA(VA, VB, EA, EB)print("Optimal Matching (π0):", π0)
```

基于渗透的去匿名化算法 (PDA)

```
python
复制代码
def PGM(EA, EB, π, r):
    # 渗透图匹配算法的占位实现
    # 需要提供具体的实现细节
    # 这里简单返回初始匹配
    return π
def PDA(VA, VB, EA, EB, dT, r):
    DA = compute_distance_matrix(EA)
    DB = compute_distance_matrix(EB)

    # 过滤出度高于阈值的节点
    high_degree_nodes_A = [i for i in range(len(VA)) if sum(EA[i]) > dT]
```

```
    high_degree_nodes_B = [i for i in range(len(VB)) if sum(EB[i]) > dT]

    D_A_high = DA[np.ix_(high_degree_nodes_A, high_degree_nodes_A)]
    D_B_high = DB[np.ix_(high_degree_nodes_B, high_degree_nodes_B)]

    l = max(np.max(D_A_high), np.max(D_B_high))
    π = CDA(D_A_high, D_B_high, l)

    f1 = f_score(π, EA, EB)

    for (uA, uB) in list(π.items()):
        π_temp = π.copy()
        del π_temp[uA]
        f2 = f_score(π_temp, EA, EB)

        if f2 > f1:
            π = π_temp
            f1 = f2

    π = PGM(EA, EB, π, r)
    return π
# 示例网络数据
VA = [0, 1, 2, 3, 4]
VB = [0, 1, 2, 3, 4]
EA = [[0, 1, 0, 0, 0], [1, 0, 1, 0, 0], [0, 1, 0, 1, 0], [0, 0, 1, 0, 1],
[0, 0, 0, 1, 0]]
EB = [[0, 1, 0, 0, 0], [1, 0, 1, 0, 0], [0, 1, 0, 1, 0], [0, 0, 1, 0, 1],
[0, 0, 0, 1, 0]]
# 执行 PDA 算法
π0 = PDA(VA, VB, EA, EB, dT=1, r=0.5)print("Optimal Matching (π0):",
π0)
```

解释

- **compute_distance_matrix**：使用 Floyd-Warshall 算法计算距离矩阵。
- **CDA**：占位函数，用于集体去匿名化算法，需要具体实现。
- **f_score**：计算结构 F-score，包括链接精度和链接召回率。
- **DDA**：直接去匿名化算法的实现，优化匹配结果。
- **PGM**：占位函数，用于渗透图匹配算法，需要具体实现。
- **PDA**：基于渗透的去匿名化算法的实现，适用于大规模网络，包含度阈值和渗透阈值的设置与优化。

上述代码中，一些函数（如 CDA 和 PGM）仅提供占位实现，具体细节需要根据论文中的算法进行填充。