Table 1: Summary of notation

Symbol	Description
$V/\{v\}$	set of all vertices except v
$V_i(v)$	set of i-hop neighbors of vertex v ($V_0(v) = \{v\}$)
$V_{\leq i}(v)$	set of vertices that are at most i hops away from vertex v (i.e., $V_{\leq i}(v) = \cup_{k=0}^{i} V_k(v)$)
$E_i(v)$	set of edges connecting two vertices in $V_i(v)$ ($E_0(v)=\emptyset$)
$E_{\leq i}(v)$	set of edges connecting two vertices in $V_{\leq i}(v)$

A Fast Algorithm for Nodal Betweenness Centraility

Abstract

In this section, we define terms that represent two types of egocentric networks, egocentric networks and multi-layered egocentric networks (Section ??), as well as the betweenness of a vertex in its ego and x-egocentric networks (Section 3). We then present several properties of the multi-layered egocentric networks (Section ??). Our betweenness computation algorithm in Section 6 takes advantage of these properties.

Keywords: Egocentric, Friendship Networks, Betweenness Centrality

1. Introduction

2. Multi-order Egocentric Friendship Networks and Their Betweenness

3. Betweenness in Multi-order Friendship Networks

4. Evaluation

5. EEGO

6. Computation

Algorithm 1 *nodal_betweenness*$(G(V, E), v)$

Input: a graph $G(V, E)$ and a vertex $v(\in V)$
Output: *betweenness of* v
create an $2D$ array $D[1, 2, ..., |V|][1, 2, ..., |V|]$
$sum \leftarrow 0$
for p : 1 to u **do**
    for q : $p + 1$ to u **do**
        $D[p][q] \leftarrow dependency1(p, q, N)$
        $sum \leftarrow sum + D[p][q]$
    end for
    for q : $u + 1$ to $u + w$ **do**
        $D[p][q] \leftarrow dependency2(p, q, N, D)$
        $sum \leftarrow sum + D[p][q]$
    end for
end for
for p : $u + 1$ to $u + w$ **do**
    for q : $p + 1$ to $u + w$ **do**
        $D[p][q] \leftarrow dependency2(p, q, N, D)$
        $sum \leftarrow sum + D[p][q]$
    end for
end for
return $\frac{2 \cdot sum}{(u+w)(u+w-1)}$

Algorithm 2 *dependency*$1(p, q, N)$

Input: p, q, $N[0, 1, ..., u + w]$
if $q \in N[p]$ **then**
    return 0 ▷ by Theorem 1
else
    return $1/|\ N[p] \cap N[q]\ |$ ▷ by Theorem 2
end if

Algorithm 3 *dependency2*(p, q, N, D)

```
Input: p, q, N[0, 1, ..., u + w], D[1, 2, ..., u + w][1, 2, ..., u + w]
C ← []
for each r ∈ N[q] do
    if p < r then
        τ = D[p][r]
    else
        τ = D[r][p]
    end if
    if τ == 0 then
        return 0                                  ▷ by Theorem 3
    else
        append τ to C
    end if
end for
return H̄(C)                                       ▷ by Theorem 4
```