Social Network Analysis Dynamics

Joanna Biega

April 23, 2013

Social network

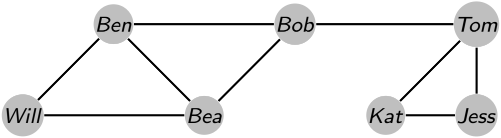
Social network

A graph $G(V, E)$ where V denotes entities in the social network and E denotes relationships between entities.

Social network

Social network

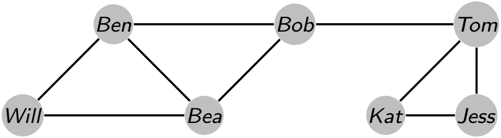
A graph $G(V, E)$ where V denotes entities in the social network and E denotes relationships between entities.



Social networks - community detection

Example task

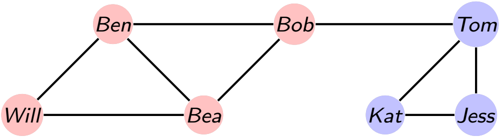
Cluster the aquaintance graph into communities, i.e. groups of nodes that are densely internally connected.



Social networks - community detection

Example task

Cluster the aquaintance graph into communities, i.e. groups of nodes that are densely internally connected.



K-Clique Clustering

K-Clique

A complete subgraph with k nodes.

Adjacent k-cliques

Two k-cliques are adjacent to each other iff they share $k-1$ nodes.

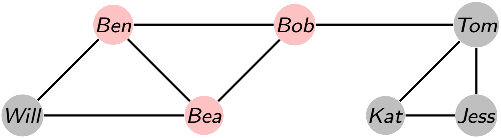
K-Clique Clustering

K-Clique

A complete subgraph with k nodes.

Adjacent k-cliques

Two k-cliques are adjacent to each other iff they share $k-1$ nodes.



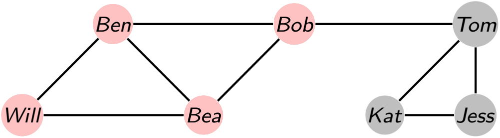
K-Clique Clustering

K-Clique

A complete subgraph with k nodes.

Adjacent k-cliques

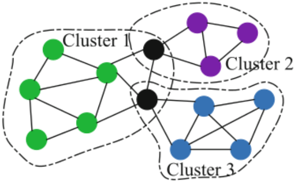
Two k-cliques are adjacent to each other iff they share $k - 1$ nodes.



K-Clustering - definitions

K-Clique Cluster

A union of k-cliques such that cliques can be reached from each other through a series of adjacent k-cliques.

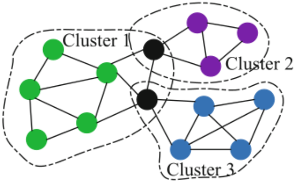


3-clique clusters

K-Clustering - definitions

K-Clique Cluster

A union of k-cliques such that cliques can be reached from each other through a series of adjacent k-cliques.

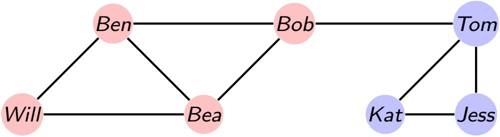


3-clique clusters

K-Clique Clustering

Compute all k-clique clusters in a graph G.

Intuition behind k-clique clustering.



Dynamic social networks

Capturing the evolution of a social network

- Snaphot graph model based methods
- Incremental methods
  - **Incremental k-clique clustering**

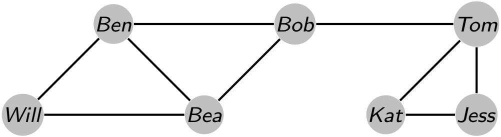
Dynamic social network

Initial graph G plus an infinite change stream $c_1, \ldots, c_\infty$.

c_i has one of the following types:

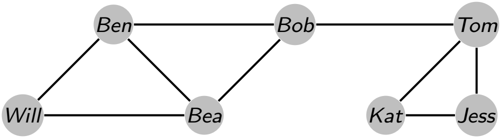
- node deletion: $u-$,
- edge deletion: $uv-$,
- node addition: $u+$,
- edge addition: $uv+$.

Node deletion

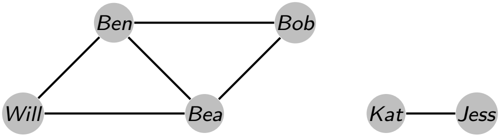


(Tom)-

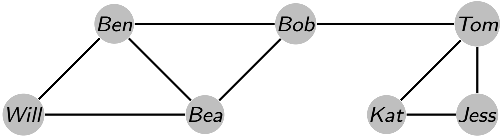
Node deletion



(Tom)-

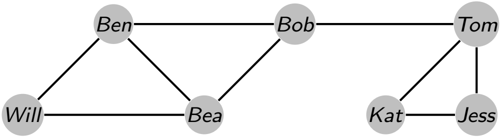


Edge deletion

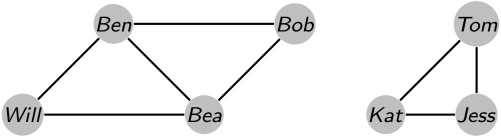


(Bob,Tom)-

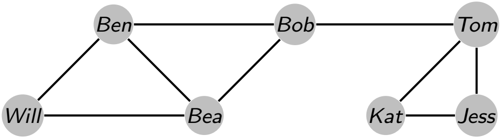
Edge deletion



(Bob,Tom)-

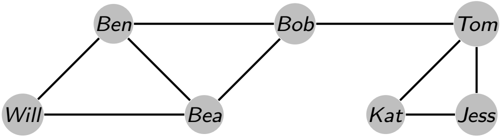


Node addition

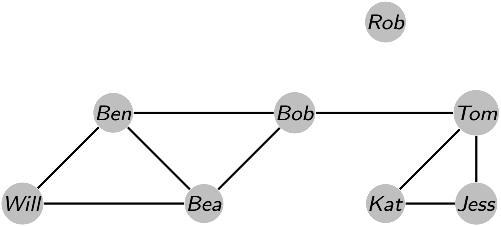


(Rob)+

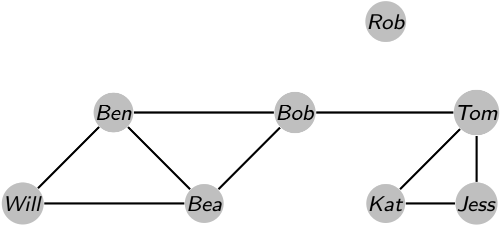
Node addition



(Rob)+

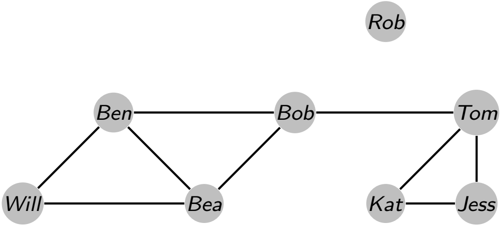


Edge addition

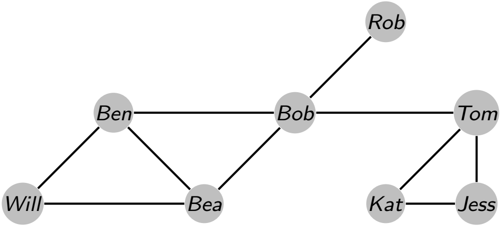


(Bob, Rob)+

Edge addition



(Bob, Rob)+



Incremental dynamic k-clique clustering - problem statement

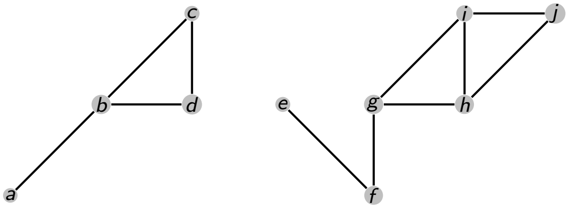
Given a network G, a set of clusters P, and a change c, locally update P so that it remains a valid set of k-clusters for the network $G' = G + c$.

$$(G, P) + c = (G + c, P')$$

2-Clique Clustering

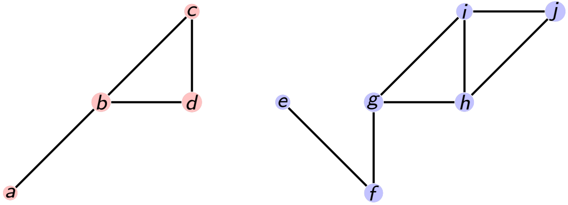
2-Clique clustering

What exactly are 2-clique clusters?



2-Clique clustering

What exactly are 2-clique clusters?



Connected components!

2-Clique clustering

How do we find connected components of a graph?

DFS

2-Clique clustering

How do we find connected components of a graph?

DFS

How do we represent the result?

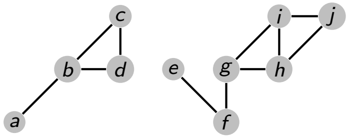
DFS forest

Input (2-clique clustering)

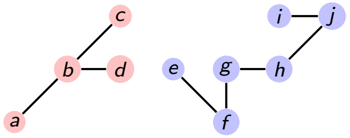
Input: G, F, c
Output: updated G and F

```
if c = (uv)− then
    call TED(G, F, u, v)
end if
if c = (u)− then
    for all neighbours v of u in G do
        call TED(G, F, u, v)
    end for
end if
if c = (uv)+ then
    call TEA(G, F, u, v)
end if
if c = (u)+ then
    add u into G and F
end if
```

Graph (social network) G



Forest F - the DFS representation of G



Change c

Input (2-clique clustering)

Input: G, F, c
Output: updated G and F

```
if c = (uv)− then
    call TED(G, F, u, v)
end if
if c = (u)− then
    for all neighbours v of u in G do
        call TED(G, F, u, v)
    end for
end if
if c = (uv)+ then
    call TEA(G, F, u, v)
end if
if c = (u)+ then
    add u into G and F
end if
```

TED(G, F, u, v)
two-clique edge deletion

Input (2-clique clustering)

Input: G, F, c
Output: updated G and F

```
if c = (uv)− then
    call TED(G, F, u, v)
end if
if c = (u)− then
    for all neighbours v of u in G do
        call TED(G, F, u, v)
    end for
end if
if c = (uv)+ then
    call TEA(G, F, u, v)
end if
if c = (u)+ then
    add u into G and F
end if
```

TED(G, F, u, v)
two-clique edge deletion

Input (2-clique clustering)

Input: G, F, c
Output: updated G and F

```
if c = (uv)− then
    call TED(G, F, u, v)
end if
if c = (u)− then
    for all neighbours v of u in G do
        call TED(G, F, u, v)
    end for
end if
if c = (uv)+ then
    call TEA(G, F, u, v)
end if
if c = (u)+ then
    add u into G and F
end if
```

TEA(G, F, u, v)
two-clique edge addition

Input: G, F, c
Output: updated G and F

```
if c = (uv)− then
    call TED(G, F, u, v)
end if
if c = (u)− then
    for all neighbours v of u in G do
        call TED(G, F, u, v)
    end for
end if
if c = (uv)+ then
    call TEA(G, F, u, v)
end if
if c = (u)+ then
    add u into G and F
end if
```

TEA(G, F, u, v)
two-clique edge addition

Edge deletion (2-clique clustering)

(u, v)—
Input: G, F, u, v
Output: updated G and F

Edge deletion (2-clique clustering)

(u, v)—
Input: G, F, u, v
Output: updated G and F

Case 1

(u, v) is a backward edge.

No changes in G, F.

Edge deletion (2-clique clustering)

(u, v)—
Input: G, F, u, v
Output: updated G and F

Case 2

- (u, v) is a forward edge
- subtree[v] becomes detached from the current tree.

Edge deletion (2-clique clustering)

(u, v)—
Input: G, F, u, v
Output: updated G and F

Case 3

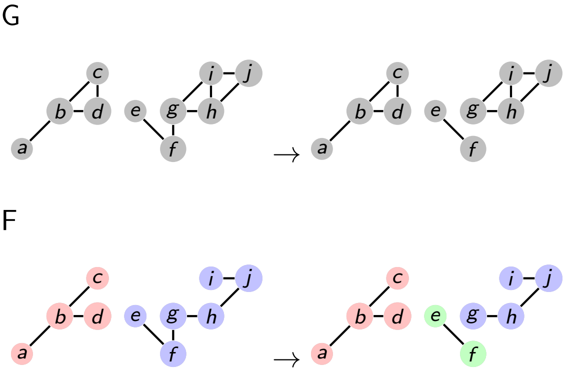
- (u, v) is a forward edge
- there exists a backward edge keeping subtree[v] attached to the current tree.

Edge deletion (2-clique clustering)

Case 2

Deleting a forward edge (f, g)—(with a real subtree detachment)

1: detach *subtree*$[v]$ from *tree*$[v]$
2: add *subtree*$[v]$ to F

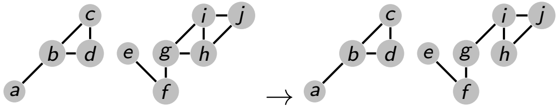


Edge deletion (2-clique clustering)

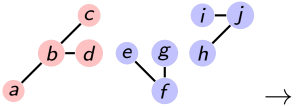
Case 3

Deleting a forward edge (g, h)—(without a real subtree detachment)

G



F

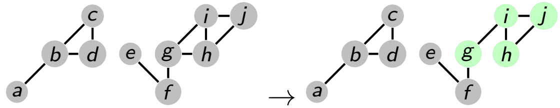


Edge deletion (2-clique clustering)

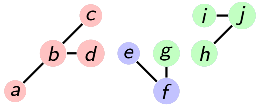
Case 3

Deleting a forward edge (g, h)— (without a real subtree detachment)

G



F



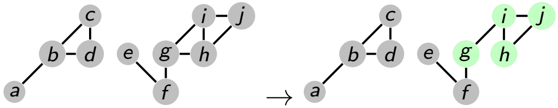
$\longrightarrow DFS_G(g,h,i,j)$

Edge deletion (2-clique clustering)

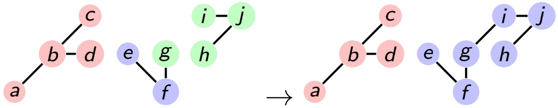
Case 3

Deleting a forward edge (g, h)— (without a real subtree detachment)

G



F



Edge deletion (2-clique clustering) - complexity

2-clique edge deletion $(u, v)-$

- Best case: $O(1)$ (backward edge)
- Worst case: $O(log(N) + |subtree[v]|)$
  - find the closest ancestor
  - run the DFS procedure

Edge addition (2-clique clustering)

$(u, v)+$
Input: G, F, u, v
Output: updated G and F

$(u, v)+$
Input: G, F, u, v
Output: updated G and F

Case 1

- (u, v) is an edge between two nodes in one DFS tree
- the DFS order does not get violated

$\rightarrow$ No changes.

Edge addition (2-clique clustering)

$(u, v)+$
Input: G, F, u, v
Output: updated G and F

Case 2

- (u, v) is an edge between two nodes in one DFS tree
- the DFS order gets violated.

Edge addition (2-clique clustering)

$(u, v)+$
Input: G, F, u, v
Output: updated G and F

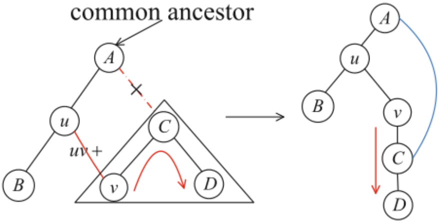
Case 3

(u, v) is a an edge crossing two DFS trees in F.

Edge addition (2-clique clustering)

Case 2

Adding an edge $(u, v)+$ violating the DFS order.

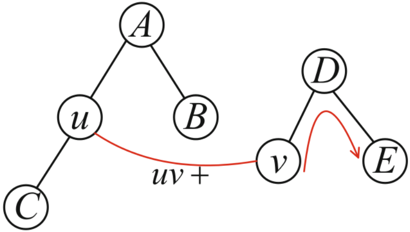


DFS on the tree to which the "shortcut" is added.

Edge addition (2-clique clustering)

Case 3

Adding an edge $(u, v)+$ crossing two DFS trees.



DFS on the smaller tree.

Edge addition (2-clique clustering) - complexity

2-clique edge addition $(u, v)+$

- Best case: $O(1)$ (backward edge)
- Worst case: $O(log(N) + |tree[v]|)$
  - find the closest ancestor
  - run the DFS procedure

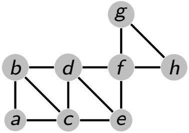
Incremental K-Clique Clustering

Input (k-clique clustering)

Input: G, C, H, F, c
Output: updated G, C, H, F

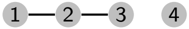
```
if c = (uv)− then
    call KED(G, C, H, F, u, v)
end if
if c = (u)− then
    for all neighbours v of u in G do
        call KED(G, C, H, F, u, v)
    end for
end if
if c = (uv)+ then
    call KEA(G, C, H, F, u, v)
end if
if c = (u)+ then
    add u into G
end if
```

- Graph (social network) G



- Set C of maximal cliques in G of size $\geq k$:
$C_1 = (a, c, b, d)$, $C_2 = (c, d, e)$,
$C3 = (d, e, f)$, $C4 = (f, g, h)$

- Graph H = (C, E) where
$(i, j) \in E \iff C_i$ and C_j have $\geq k - 1$ common nodes.



(k-clique clusters!)

- Forest F - DFS(H)
- Change c

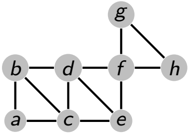
Input (k-clique clustering)

Input: G, C, H, F, c
Output: updated G, C, H, F

```
if c = (uv)− then
    call KED(G, C, H, F, u, v)
end if
if c = (u)− then
    for all neighbours v of u in G do
        call KED(G, C, H, F, u, v)
    end for
end if
if c = (uv)+ then
    call KEA(G, C, H, F, u, v)
end if
if c = (u)+ then
    add u into G
end if
```

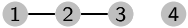
Problem?

- Graph (social network) G



- Set C of maximal cliques in G of size $\geq k$:
$C_1 = (a, c, b, d)$, $C_2 = (c, d, e)$,
$C3 = (d, e, f)$, $C4 = (f, g, h)$

- Graph H = (C, E) where
$(i, j) \in E \iff C_i$ and C_j have $\geq k - 1$ common nodes.



(k-clique clusters!)

- Forest F - DFS(H)
- Change c

Input (k-clique clustering)

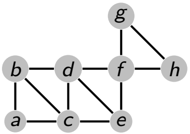
Input: G, C, H, F, c
Output: updated G, C, H, F

```
if c = (uv)− then
    call KED(G, C, H, F, u, v)
end if
if c = (u)− then
    for all neighbours v of u in G do
        call KED(G, C, H, F, u, v)
    end for
end if
if c = (uv)+ then
    call KEA(G, C, H, F, u, v)
end if
if c = (u)+ then
    add u into G
end if
```

Problem?

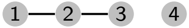
Maximal clique discovery is a NP-Complete problem.

- Graph (social network) G



- Set C of maximal cliques in G of size $\geq k$:
$C_1 = (a, c, b, d)$, $C_2 = (c, d, e)$,
$C3 = (d, e, f)$, $C4 = (f, g, h)$

- Graph H = (C, E) where
$(i, j) \in E \iff C_i$ and C_j have $\geq k - 1$ common nodes.



(k-clique clusters!)

- Forest F - DFS(H)
- Change c

Input (k-clique clustering)

Input: G, C, H, F, c
Output: updated G, C, H, F

if c = $(uv)-$ **then**
    call KED(G, C, H, F, u, v)
end if
if c = $(u)-$ **then**
    for all neighbours v of u in G **do**
        call KED(G, C, H, F, u, v)
    end for
end if
if c = $(uv)+$ **then**
    call KEA(G, C, H, F, u, v)
end if
if c = $(u)+$ **then**
    add u into G
end if

KED(G, C, H, F, u, v)
k-clique edge deletion

Input (k-clique clustering)

Input: G, C, H, F, c
Output: updated G, C, H, F

```
if c = (uv)− then
    call KED(G, C, H, F, u, v)
end if
if c = (u)− then
    for all neighbours v of u in G do
        call KED(G, C, H, F, u, v)
    end for
end if
if c = (uv)+ then
    call KEA(G, C, H, F, u, v)
end if
if c = (u)+ then
    add u into G
end if
```

KED(G, C, H, F, u, v)
k-clique edge deletion

Input (k-clique clustering)

Input: G, C, H, F, c
Output: updated G, C, H, F

```
if c = (uv)− then
    call KED(G, C, H, F, u, v)
end if
if c = (u)− then
    for all neighbours v of u in G do
        call KED(G, C, H, F, u, v)
    end for
end if
if c = (uv)+ then
    call KEA(G, C, H, F, u, v)
end if
if c = (u)+ then
    add u into G
end if
```

KEA(G, C, H, F, u, v)
k-clique edge addition

Input (k-clique clustering)

Input: G, C, H, F, c
Output: updated G, C, H, F

```
if c = (uv)− then
    call KED(G, C, H, F, u, v)
end if
if c = (u)− then
    for all neighbours v of u in G do
        call KED(G, C, H, F, u, v)
    end for
end if
if c = (uv)+ then
    call KEA(G, C, H, F, u, v)
end if
if c = (u)+ then
    add u into G
end if
```

Edge deletion (k-clique clustering)

Input: G, C, H, F, u, v
Output: updated G, C, H, F

```
for C_i in C containing u and v do
    if size(C_i) > k then
        n ← size(C) + 1
        C_n ← C_i − u
        C_i ← C_i − v
        add node n into H and F
        for all neighbour j of i in H do
            if |C_i ∩ C_j| < k − 1 then
                call TED(H, F, i, j)
            end if
            if |C_n ∩ C_j| ≥ k − 1 then
                call TEA(H, F, n, j)
            end if
        end for
        call TEA(H, F, i, n)
    else
        delete C_i from C
        delete i from H and F
    end if
end for
```

Case 1

No clique in C containins both u and v: C, H unchanged.

Edge deletion (k-clique clustering)

Input: G, C, H, F, u, v
Output: updated G, C, H, F

```
for C_i in C containing u and v do
    if size(C_i) > k then
        n ← size(C) + 1
        C_n ← C_i − u
        C_i ← C_i − v
        add node n into H and F
        for all neighbour j of i in H do
            if |C_i ∩ C_j| < k − 1 then
                call TED(H, F, i, j)
            end if
            if |C_n ∩ C_j| ≥ k − 1 then
                call TEA(H, F, n, j)
            end if
        end for
        call TEA(H, F, i, n)
    else
        delete C_i from C
        delete i from H and F
    end if
end for
```

Edge deletion (k-clique clustering)

Input: G, C, H, F, u, v
Output: updated G, C, H, F

```
for C_i in C containing u and v do
    if size(C_i) > k then
        n ← size(C) + 1
        C_n ← C_i − u
        C_i ← C_i − v
        add node n into H and F
        for all neighbour j of i in H do
            if |C_i ∩ C_j| < k − 1 then
                call TED(H, F, i, j)
            end if
            if |C_n ∩ C_j| ≥ k − 1 then
                call TEA(H, F, n, j)
            end if
        end for
        call TEA(H, F, i, n)
    else
        delete C_i from C
        delete i from H and F
    end if
end for
```

Case 2

$size(C_i) = k$.

Edge deletion (k-clique clustering)

Input: G, C, H, F, u, v
Output: updated G, C, H, F

```
for C_i in C containing u and v do
    if size(C_i) > k then
        n ← size(C) + 1
        C_n ← C_i − u
        C_i ← C_i − v
        add node n into H and F
        for all neighbour j of i in H do
            if |C_i ∩ C_j| < k − 1 then
                call TED(H, F, i, j)
            end if
            if |C_n ∩ C_j| ≥ k − 1 then
                call TEA(H, F, n, j)
            end if
        end for
        call TEA(H, F, i, n)
    else
        delete C_i from C
        delete i from H and F
    end if
end for
```

Case 3

$size(C_i) > k$.

Edge deletion (k-clique clustering)

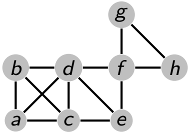
Case 3

$size(C_i) > k$.

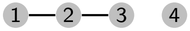
```
if size(C_i) ≥ k then
    n ← size(C) + 1
    C_n ← C_i − u
    C_i ← C_i − v
    add node n into H and F
    for all neighbour j of i in H do
        if |C_i ∩ C_j| < k − 1 then
            call TED(H, F, i, j)
        end if
        if |C_n ∩ C_j| ≥ k − 1 then
            call TEA(H, F, n, j)
        end if
    end for
end if
```

$$(a, b)-,\ i = 1$$
$$C_1 = (a, c, b, d), C_2 = (c, d, e),$$
$$C3 = (d, e, f), C4 = (f, g, h)$$

G



H



Edge deletion (k-clique clustering)

Case 3

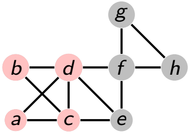
$size(C_i) > k$.

```
if size(C_i) ≥ k then
    n ← size(C) + 1
    C_n ← C_i − u
    C_i ← C_i − v
    add node n into H and F
    for all neighbour j of i in H do
        if |C_i ∩ C_j| < k − 1 then
            call TED(H, F, i, j)
        end if
        if |C_n ∩ C_j| ≥ k − 1 then
            call TEA(H, F, n, j)
        end if
    end for
end if
```

$$(a, b)-,\ i = 1$$

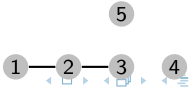
$$C_1 = (a, c, b, d), C_2 = (c, d, e), C3 = (d, e, f), C4 = (f, g, h)$$

G



$$C_1 = (a, c, d),\ C_5 = (b, c, d)$$

H



Edge deletion (k-clique clustering)

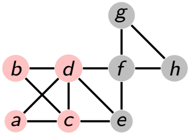
Case 3

$size(C_i) > k$.

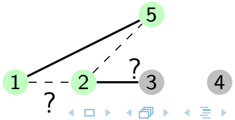
```
if size(C_i) ≥ k then
    n ← size(C) + 1
    C_n ← C_i − u
    C_i ← C_i − v
    add node n into H and F
    for all neighbour j of i in H do
        if |C_i ∩ C_j| < k − 1 then
            call TED(H, F, i, j)
        end if
        if |C_n ∩ C_j| ≥ k − 1 then
            call TEA(H, F, n, j)
        end if
    end for
    call TEA(H, F, i, n)
end if
```

$$(a, b)-, \ i = 1$$
$$C_1 = (a, c, d), C_2 = (c, d, e),$$
$$C_3 = (d, e, f), C_4 = (f, g, h)$$
$$C_5 = (b, c, d)$$

G



H



Edge deletion (k-clique clustering)

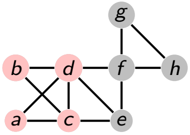
Case 3

$size(C_i) > k$.

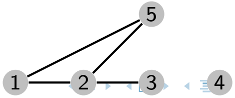
```
if size(C_i) ≥ k then
    n ← size(C) + 1
    C_n ← C_i − u
    C_i ← C_i − v
    add node n into H and F
    for all neighbour j of i in H do
        if |C_i ∩ C_j| < k − 1 then
            call TED(H, F, i, j)
        end if
        if |C_n ∩ C_j| ≥ k − 1 then
            call TEA(H, F, n, j)
        end if
    end for
    call TEA(H, F, i, n)
end if
```

$$(a, b)-, \; i = 1$$
$$C_1 = (a, c, d), C_2 = (c, d, e),$$
$$C_3 = (d, e, f), C_4 = (f, g, h)$$
$$C_5 = (b, c, d)$$

G



H



Edge deletion (k-clique clustering) - complexity

k-clique edge deletion $(u, v)-$

- Best case: $O(1)$ (backward edge)
- Worst case: $O(ndx)$
  - n - number of cliques of size $> k$ containing u and v
  - d - average degree of nodes in H
  - x - complexity of '2-clique' local updating

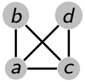
Edge addition (k-clique clustering)

Input: G, C, H, F, u, v
Output: updated G, C, H, F

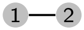
```
CN ← common neighbours of u and v
C' ← the maximal cliques in G(CN)
for all C'_i in C' do
    C'_i ← C'_i + {u, v}
end for
for all C'_i in C' do
    for all C_j in C do
        if C'_i ⊇ C_j then
            delete C_j from C
            for all neighbour l of j in H do
                call TED(H, F, l, j)
            end for
            delete j from H and F
        end if
    end for
    n ← size(C) + 1
    C_n ← C'_i
    for all C_j in C do
        if |C_n ∩ C_j| ≥ k − 1 then
            call TEA(H, F, n, j)
```

$$(b, d)+$$
$$C_1 = (a, c, b), C_2 = (a, c, d),$$

G



H



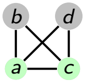
Edge addition (k-clique clustering)

Input: G, C, H, F, u, v
Output: updated G, C, H, F

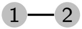
```
CN ← common neighbours of u and v
C' ← the maximal cliques in G(CN)
for all C'_i in C' do
    C'_i ← C'_i + {u, v}
end for
for all C'_i in C' do
    for all C_j in C do
        if C'_i ⊇ C_j then
            delete C_j from C
            for all neighbour l of j in H do
                call TED(H, F, l, j)
            end for
            delete j from H and F
        end if
    end for
    n ← size(C) + 1
    C_n ← C'_i
    for all C_j in C do
        if |C_n ∩ C_j| ≥ k − 1 then
            call TEA(H, F, n, j)
        end if
```

$$(b, d)+$$
$$C_1 = (a, c, b), C_2 = (a, c, d),$$

G



H



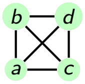
Edge addition (k-clique clustering)

Input: G, C, H, F, u, v
Output: updated G, C, H, F

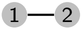
$CN \leftarrow$ common neighbours of u and v
$C' \leftarrow$ the maximal cliques in G(CN)
for all C'_i in C' **do**
    $C'_i \leftarrow C'_i + \{u, v\}$
end for
for all C'_i in C' **do**
    for all C_j in C **do**
        if $C'_i \supseteq C_j$ **then**
            delete C_j from C
            for all neighbour l of j in H **do**
                call TED(H, F, l, j)
            end for
            delete j from H and F
        end if
    end for
    $n \leftarrow size(C) + 1$
    $C_n \leftarrow C'_i$
    for all C_j in C **do**
        if $|C_n \cap C_j| \geq k - 1$ **then**
            call TEA(H, F, n, j)
        end if

$$(b, d)+$$
$$C_1 = (a, c, b), C_2 = (a, c, d),$$

G



H



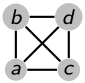
Input: G, C, H, F, u, v
Output: updated G, C, H, F

```
CN ← common neighbours of u and v
C' ← the maximal cliques in G(CN)
for all C'_i in C' do
    C'_i ← C'_i + {u, v}
end for
for all C'_i in C' do
    for all C_j in C do
        if C'_i ⊇ C_j then
            delete C_j from C
            for all neighbour l of j in H do
                call TED(H, F, l, j)
            end for
            delete j from H and F
        end if
    end for
    n ← size(C) + 1
    C_n ← C'_i
    for all C_j in C do
        if |C_n ∩ C_j| ≥ k − 1 then
            call TEA(H, F, n, j)
        end if
```

Fix the maximality of cliques

$$C = \{C_1 = (a, c, b), C_2 = (a, c, d)\}$$
$$\rightarrow C = \{C_3 = (a, b, c, d)\}$$

G



H

Edge addition (k-clique clustering) - complexity

k-clique edge addition $(u, v)+$

- Best case: $O(|CN|3^{|CN|/3})$
- Worst case: $O(|CN|3^{|CN|/3} + c)$
    - $O(|CN|3^{|CN|/3})$ - maximal clique discovery (Etsuji Tomita and Tanaka, 2004)
    - c - complexity of TEA

K-clique clustering - summary

- Models the dynamic social network using **change streams**.
- Translates **network changes** into elegant operations on a **DFS forest structure**.
- Guarantees **accuracy** of the clustering.
- Allows **avoiding the problem of granularity** of network snaphots.
- Does **not lose** the historical evolution **information**.

K-clique clustering - summary

Success?

K-clique clustering - summary

Success?

Not suitable for large networks with frequent updates.

References

Duan, D. et al.
Incremental K-Clique Clustering in Dynamic Social Networks.
Artificial Intelligence Review, 38(2):129-147, 2011.

The end

Thank you for your attention!