EFFICIENT CORE MAINTENANCE IN LARGE DYNAMIC GRAPHS

RONG-HUA LI, JEFFREY XU YU, RUI MAO

Date

- Clique is a subset of veltices of an undirected graph such that every two distinct vertices in the clique are adjacent.

- An n-clan is an n-clique which has diameter less than or equal to n as an induced subgraph.

- An n-clique of an undirected graph is a maximal subgraph in which every pair of vertices is connected by a path of length n or less.

- A k-plex is a maximal subgraph with the following property: each vertex of the induced subgraph is connected to atleast n-k other vertizes, where n is the number of vertizes in the induced subgraph.

- Given a graph G, the k-truss of G is the largest subgraph of G in which every edge is contained in at least (k-2) triangles within the subgraph.

- Percolation is the process of a liquid slowly passing through a filter.

- lemma 1. For every node v of a graph G, we have $Y_v \le C_v \le X_v \le D_v$

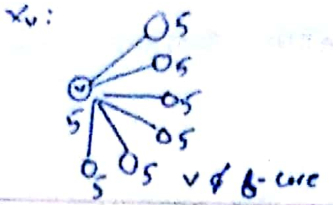
  C_v = core no. of node v.

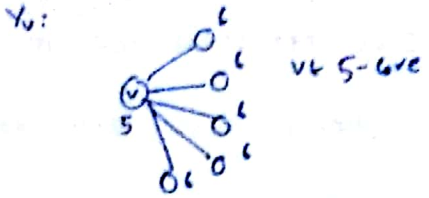
  $N(v)$ = the set of neighbor nodes of node v.

  D_v = degree of node v.

  Y_v = the no. of v's neighbors whose core number are strictly greater than C_v.

  X_v : the no. of v's neighbors whose core no. are greater than or qual to C_v.





- Definition 2. Given a graph $G = (V, E)$ and a node v, the induced core subgraph of node v, denoted as $G_v = (V_v, E_v)$, is a connected subgraph which contains node v. Moreover, the core numbers of all the nodes in G_v are equivalent to C_v.

- Definition 3. $G_{uvv} = (V_{uvv}, E_{uvv})$

$$V_{uvv} = V_v \cup V_u$$

$$E_{uvv} = \{ (v_i, v_j) \mid (v_i, v_j) \in E,\ v_i \in V_{uvv},\ v_j \in V_{uvv} \}$$

- Theorem 1 (k-core update theorem).
Given a graph $G = (V, E)$ and two nodes u & v. Then, after inserting or deleting an edge (u, v) in G, we have the following results.

- If $C_u > C_v$, only the core numbers of nodes in the induced core subgraph of node v, i.e., G_v, may need to be updated.

- If $C_u < C_v$, only the core numbers of nodes in the induced core subgraph of node u, i.e., G_u, may need to be updated.

- If $C_u = C_v$, only the core numbers of nodes in the union of two induced core subgraphs G_u and G_v, i.e., G_{uvv}, may need to be updated.

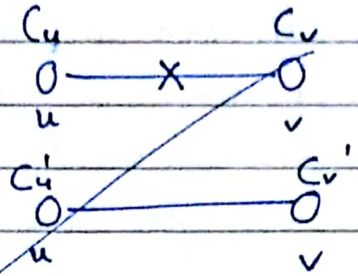
- brevity · concise and exact use of words in writing or speech.

Date

Date

- lemma 2. If we insert (delete) an edge (u,v) in a graph G, the core number of any node in G increases (decreases) by atmost 1.

Proof:



edge insertion
d_u - degree of node u in C_u - core.

If $v \notin C_u'$ - core then $C_u' = C_u$.

If $v \in C_u'$ - core then

if $d_u > C_u'$ & $d_v > C_v$ then $C_u' = C_u$, $C_v' = C_v$.

if $d_u = C_u$ then $C_u' \leq d_u + 1 = d_u' = C_u + 1$

if $d_v = C_v$ then $C_v' \leq d_v + 1 = d_v' = \cancel{d_v +} C_v + 1$

Similarly, for node v. Hence proved.

- lemma 3. Given a graph G and two nodes u & v such that $C_u = C_v$. If we insert an edge (u,v) in G, then either C_u and C_v increase by 1 or C_u & C_v do not change.

- Proof of lemma 2: (Note u can be any node in the graph).
Suppose the node ~~of~~ u's core number increased to $C_u + 2$ after addition of (u,v) edge. from C_u.
It means that (C_u+2)-core graph, every node has atleast C_u+2 degree in the subgraph.
Now, if we remove the edge (u,v), then the degree of each node in the ~~&~~ subgraph will have atleast C_u+2-1 degree in the subgraph. Which means that core number of original node ~~subgraph~~ is $C_u+2-1 = C_u+1$. Which is contradiction to our assumption.

Scanned by CamScanner

- Proof of Lemma 3:

Suppose node u & v have core number c.

If node u is in c+1-core subgraph after addition of node (u,v) then if node v is not in c+1-core subgraph then the degree in of node u in c-core & c+1-core are same. So, ~~essentially no change in c-core & c+1-core which is contradiction. Which means that degree of node u remains same and still the core number is increased~~ Which means that degree of all the nodes in c+1-core is exact same or less as in c-core which is contradiction.

Effect of the addition of edge only in case when both the nodes are either in c-core & or c+1-core because if one of the node is not in c+1-core then the effect of addition of edge is nothing to the other node.

- Lemma 4. Given a graph G and an egde (u,v). Suppose that G is updated by inserting or deleting an edge (u,v). Then, for any node w in G, if the core number of w (C_w) needs to be changed, such change only affects the core numbers of nodes in G_w. If C_w does not change, then it does not affect the core number of the nodes in G.

- Proof of Lemma 4:

If $C_u > C_v$ then by joinging edge with node of less core number doesnot increase core number of node u. It means that any core number increment of node v will have no effect on node u.

If $C_u < C_v$ then again any increment in node u core number will have no effect on node v core number.

If $C_u = C_v$ then it will follow Lemma 3.

- Theorem 2. Under the case of insertion of an edge (u,v), the core number of a node needs to be updated if and only if its color is 1 after Algorithm 3 terminates.

- Theorem 4. Given a graph G and an edge (u_0, v_0). After inserting an edge (u_0, v_0) in G, for a node $w \in V_c$ and $X_w < c+1$, we have the following pruning rules.

  - If $C_{u_0} > C_{v_0}$ (i.e., $V_c = V_{v_0}$), then for any node $u \in V_c$ that every path from v_0 to u in G_{v_0} must go through w can be pruned.

  - If $C_{u_0} < C_{v_0}$ (ie, $V_c = V_{u_0}$), then for any node $u \in V_c$ that every path from u_0 to u in G_{u_0} must go through w can be pruned.

  - If $C_{u_0} = C_{v_0}$ (i.e., $V_c = V_{u_0 \cup v_0}$), then for any node $u \in V_c$ that every path either from u_0 to u or from v_0 to u in $G_{u_0 \cup v_0}$ must go through w can be pruned.

- Theorem 3. Under the case of deletion of an edge (u,v), a node in V_c whose core number needs to update if and only if its color is 0 after Algorithm 6 terminates.

- Proof of lemma 4:

WLOG $C_u > C_v$ then by joining edge with node of less core number doesnot increase core number of node u. Suppose node v is connected to node w. If $C_w > C_v$ then increment of core number of node v will have no effect on ~~w~~ core number of w because corenumber of node v cannot be increased to $C_w + 1$. If $C_w < C_v$ then increment of core number of node v will have no effect on core number of w because node v is already present in the (C_w+1)-core.
For $C_u = C_v$, it follows lemma 3 follows.

Date

- Algorithm 1 Insertion(G, u, v)
Input: Graph $G = (V, E)$ and an edge (u, v).
Output: the updated core numbers of the nodes.

Initialize visited$(w) \leftarrow 0$ for all node $w \in V$;
Initialize color$(w) \leftarrow 0$ for all node $w \in V$;
$V_c \leftarrow \phi$;
if $C_u > C_v$ then
    $c \leftarrow C_v$;
    Color(G, v, c);
    RecolorInsert(G, c);
    UpdateInsert(G, c);
else
    $c \leftarrow C_u$;
    Color(G, u, c);
    RecolorInsert(G, c);
    UpdateInsert(G, c);

- Algorithm 2 void Color(G, u, c);
Initialize a queue Q;
Q.enqueue(u); visited$(u) \leftarrow 1$;
while Q is not empty do
    $u \leftarrow Q$.dequeue();
    for each node $w \in N(u)$ do
        if visited$(w) = 0$ and $C_w = c$ then
            Q.enqueue(w); visited$(w) \leftarrow 1$;
    if color$(u) = 0$ then
        $V_c \leftarrow V_c \cup \{u\}$; color$(u) = 1$;

Date

- Algorithm 3 void RecolorInsert (G, c)

flag $\leftarrow 0$;
for each node $u \in V_c$ do
    if color$(u) = 1$ then
        $\tilde{X}_u \leftarrow 0$;
        for each node $w \in N(u)$ do
            if (color$(w) = 1$) or $((C_w > c)$ then
                $\tilde{X}_u \leftarrow \tilde{X}_u + 1$;
        if $\tilde{X}_u \leq c$ then
            color$(u) \leftarrow 0$;
            flag $\leftarrow 1$;
if flag $= 1$ then
    RecolorInsert (G, c);

- Algorithm 4 void UpdateInsert (G, c)

for each node $w \in V_c$ do
    if color$(w) = 1$ then
        $C_w \leftarrow c + 1$;

- Algorithm 5 Deletion (G, u, v)

Input: Graph $G = (V, E)$ and an edge (u, v).
Output: the updated core numbers of the nodes.

Initialize visited$(w) \leftarrow 0$ for all node $w \in V$;
Initialize color$(w) \leftarrow 0$ for all node $w \in V$;
$V_c \leftarrow \emptyset$;
if $C_u > C_v$ then
    $c \leftarrow C_v$;
    Color(G, v, c);
    Recolor~~Insert~~Delete (G, c);
    UpdateDelete (G, c);

```
if Cu < Cv then
        c ← Cu;
        Color (G, u, c);
        RecolorDelete ( G, c);
        UpdateDelete ( G, c);
if Cu = Cv then
        c ← Cu;
        Color (G, u, c);
        if color(v) = 0 then
                Initialize visited(w) ← 0 for all node w ∈ V;
                Color (G, v, c);
                RecolorDelete ( G, c);
                UpdateDelete (G, c);
        else
                RecolorDelete (G, c);
                UpdateDelete (G, c);
```

- Algorithm 6 void RecolorDelete (G, c)

```
flag ← 0;
for each node u ∈ Vc do
        if color(u) = 1 then
                x̃u ← 0;
                for each node w ∈ N(u) do
                        if (color(w) = 1) or (Cw > c) then
                                x̃u ← x̃u + 1 ;
                if x̃u < c then
                        color(u) ← 0;
                        flag ← 1;
if flag = 1 then
        RecolorDelete (G, c);
```

Algorithm 7 void UpdateDelete(G, c)

for each node $w \in V_c$ do
    if color$(w) = 0$ then
        $C_w \leftarrow c-1$;

Algorithm 8 void XPruneColor(G, u, c)

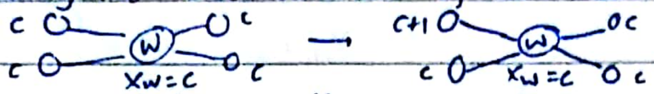
Initialize a queue Q;
Q.enqueue(u); visited$(u) \leftarrow 1$;
while Q is not empty do
    $u \leftarrow Q$.dequeue();
    Compute X_u;
    if $X_u > c$ then
        for each node $w \in N(u)$ do
            if visited$(w) = 0$ and $C_w = c$ then
                Q.enqueue(w); visited$(w) \leftarrow 1$;
    if color$(u) = 0$ then
        $V_c \leftarrow V_c \cup \{u\}$; color$(u) = 1$;

Algorithm 9 void YPruneColor(G, u, c)

Initialize a queue Q;
Q.enqueue(u); visited$(u) \leftarrow 1$;
while Q is not empty do
    $u \leftarrow Q$.dequeue();
    Compute Y_u;
    if $Y_u < c$ then
        for each node $w \in N(u)$ do
            if visited$(w) = 0$ & $C_w = c$ then
                Q.enqueue(w); visited$(w) \leftarrow 1$;
tab
    if color$(u) = 0$ then
        $V_c \leftarrow V_c \cup \{u\}$; color$(u) = 1$;

Scanned by CamScanner

- Theorem 4 : X-Pruning Proof : (insertion case)
Idea is that if a node w has $X_w = c$ after inserting (u_0, v_0) then after updating the core number of all nodes, $X_w' = X_w$.



Because if a node N(w) gets its core number increased then also it doesnt affect X_w' because it was already counted before.

- X-Pruning Proof : (deletion case). (u_0, v_0)

if $C_{u_0} < C_{v_0}$,
if after deleting (u_0, v_0), $X_{u_0} \geq c$ then core number of u_0 will not change. because u_0 still have more than equal to c neighbours whose core number is greater than equal to c.

- Theorem 5 : Given a graph G and an edge (u_0, v_0). After inserting/deleting an edge (u_0, v_0) in G, for a node $w \in V_c$, if $Y_w = c$, then we have the following pruning rules.

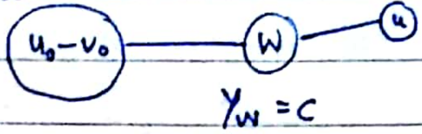
• If $C_{u_0} > C_{v_0}$ (i.e. $V_c = V_{v_0}$), then for any node $u \in V_c$ and $u \neq w$ that every path from v_0 to u in G_{v_0} must go through w can be pruned.

• If $C_{u_0} < C_{v_0}$ (i.e. $V_c = V_{u_0}$), then for any node $u \in V_c$ and $u \neq w$ that every path from u_0 to u in G_{u_0} must go through w can be pruned.

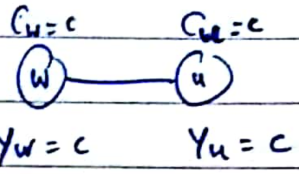
• If $C_{u_0} = C_{v_0}$ (i.e. $V_c = V_{u_0 \cup v_0}$), then for any node $u \in V_c$, and $u \neq w$ that every path either from u_0 to u or from v_0 to u in $G_{u_0 \cup v_0}$ must go through w can be pruned.

- Theorem 5: Y-Pruning Proof:

Case 1: Insertion:



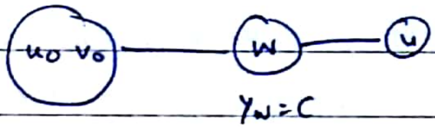
If $Y_w = c$ then $Y_u < c$ because if not then $Y_u = c$, so



Then no. of ~~Xxx~~ nodes w & u are connected to whose core no. is greater than c is c & since both are also connected to each other, ~~&~~ which should make the core number of w & u to be c+1. Hence, contradiction.

Case 2: Deletion:

~~Za~~ :



In case of deletion, Y_w will still remain c because those node which contributes to Y_w are not in $G_{u_0 v_0}$ because their core number is greater than c.

Hence, core number of w will also not change.

Scanned by CamScanner