智能网络与优化实验室

Intelligent Network and Optimization Laboratory, Renmin University

中国人民大學

RENMIN UNIVERSITY OF CHINA

Hierarchical Core Maintenance on Large Dynamic Graphs

Zhe Lin, Fan Zhang, Xuemin Lin, Wenjie Zhang, Zhihong Tian

VLDB 2021

Xiaowei Lv

Background

Problems: Hierarchical core maintenance with edge insertion or deletion

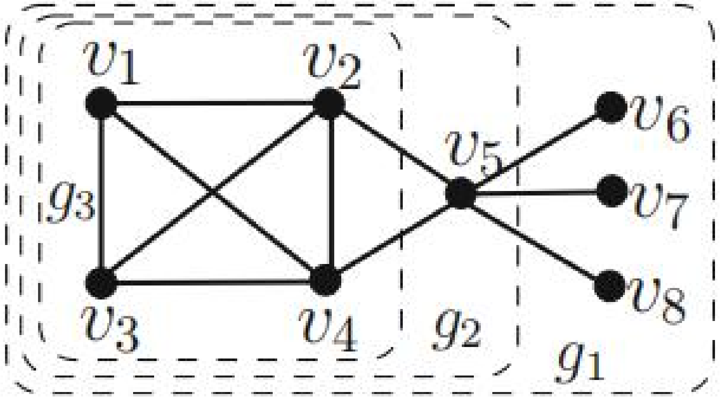


Fig. 3.1: An example graph and its *k*-cores

k-cores:

Maximal connected subgraph in which every vertex is connected to at least k other vertices in the same subgraph.

Background

Hierarchical structure (Core Hierarchy Tree) :

k-core-hierarchy: T(G)

k-core: C_i^k

Tree node: n_1 $V(n_1) = \{v | v \in C_i^k \wedge core(v) = k\}$

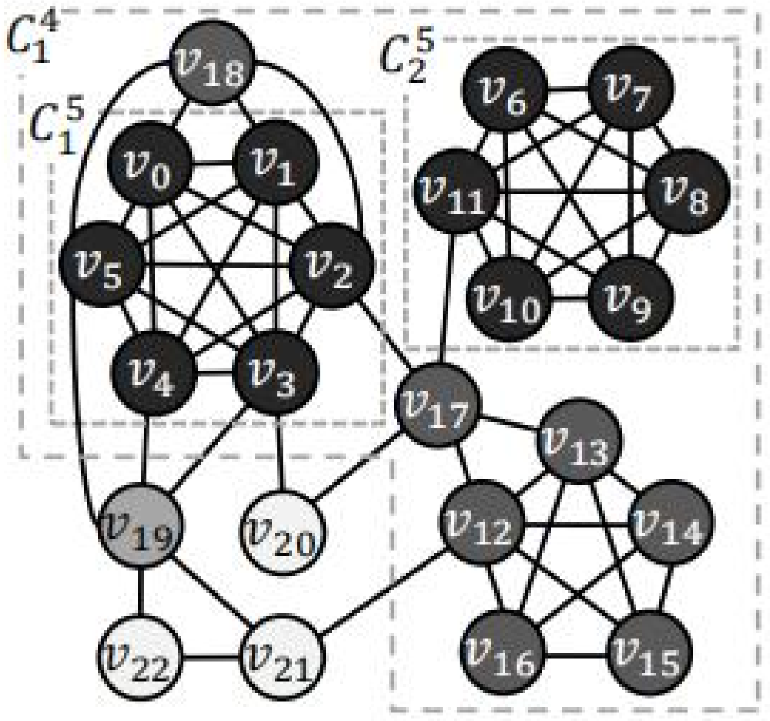
Tree edge: $n_1 - C_i^{k_1}$ $n_2 - C_j^{k_2}$

$$iff\ i)\ k_1 < k_2\ \ ii)\ C_j^{k_2} \subset C_i^{k_1}$$

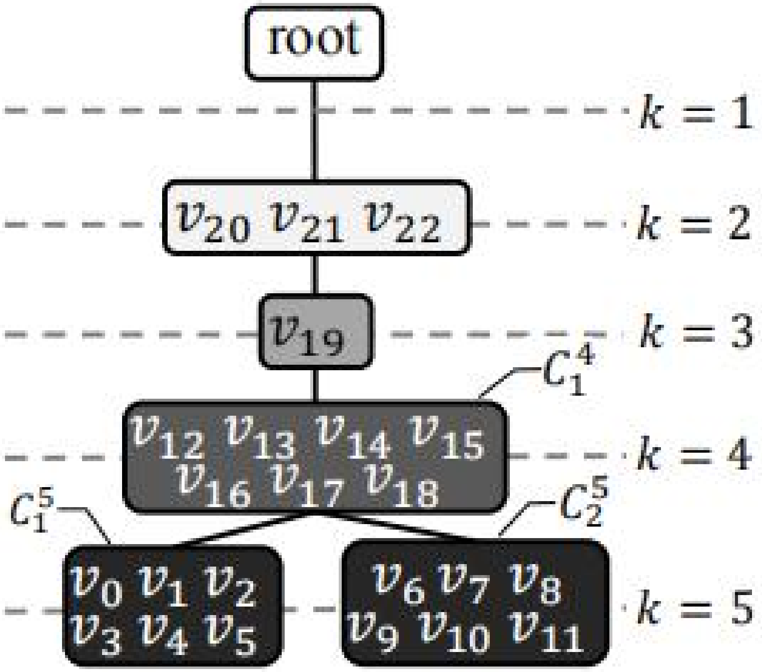
$iii)\ for\ any\ node\ k_1 <$ k' $< k_2$, the associate node is not the parent of n_2

Root: record isolated vertices

Background

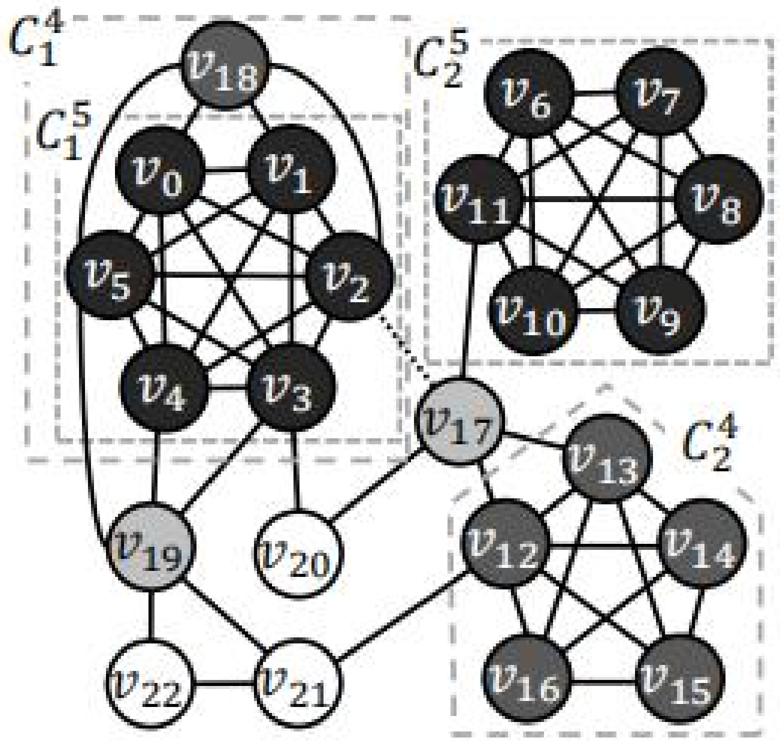


(a) graph

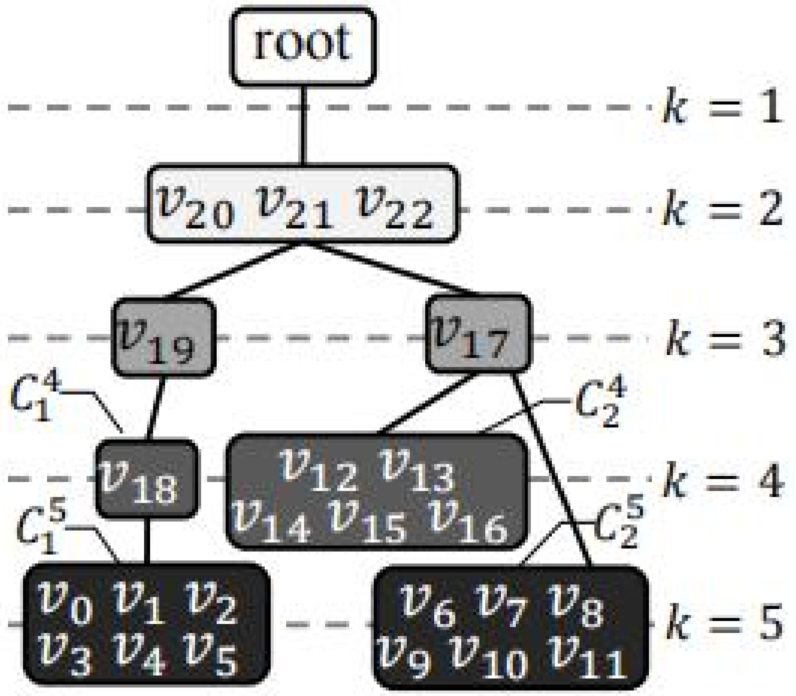


(b) k-core hierarchy

Background



(c) remove edge (v_2, v_{17})



(d) updated k-core hierarchy

Constructs k-core hierarchy:

Algorithm 3: CoreHierarchy: compute the core hierarchy tree of a graph

Input: A graph $G = (V, E)$, a degeneracy ordering `seq` of vertices, and the core numbers `core`$(\cdot)$ of vertices

Output: A core hierarchy tree `CoreHT` of G

```
Initialize an empty CoreHT, and a disjoint-set data structure 𝓕 for V;
for each vertex u ∈ V do
    Add a node r_u, with weight core(u) and containing vertex u, to CoreHT;
    Point u to r_u;
```

Fundamentals

for each *vertex u in* `seq` *in reverse order* **do**

  for each *neighbor v of u in G that appear later than u in* `seq` **do**

    Let r_v and r_u be the nodes of `CoreHT` pointed by the representatives of the sets containing v and u in $\mathscr{F}$, respectively;

    if $r_v \neq r_u$ **then**

      `/* Update the CoreHT */`

      if *the weight of r_v equals the weight of r_u* **then**

        Move the content (i.e., vertices and children) of r_v to r_u;

      else Assign r_u as the parent of r_v in the `CoreHT`;

      `/* Update the disjoint-set data structure` $\mathscr{F}$ `*/`

      Union u and v in $\mathscr{F}$, and point the updated representative of the set containing u to r_u;

return `CoreHT`;

Fundamentals

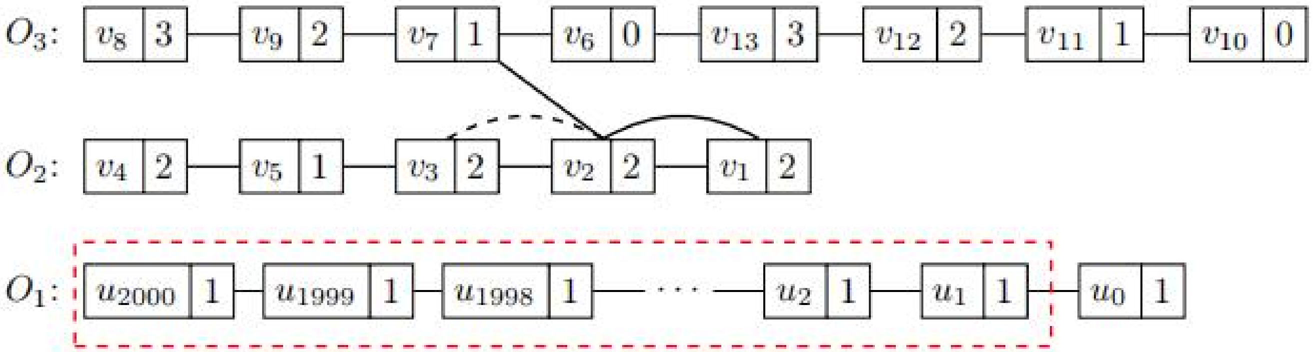
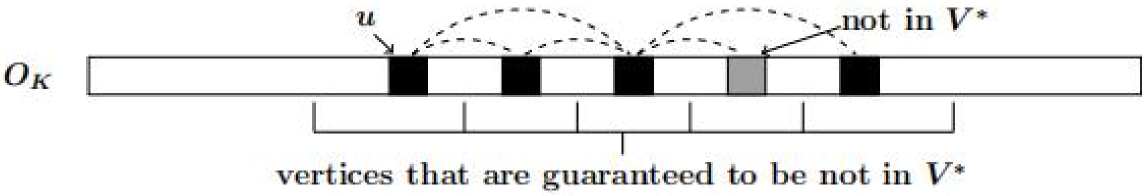
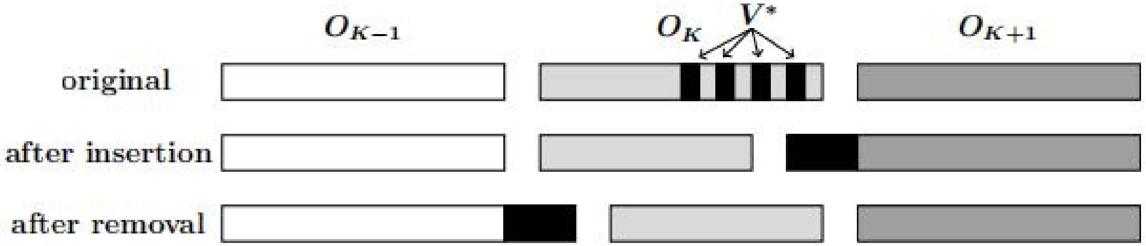


Fig. 6: The k-order for G in Fig. 3



(a) Identification of V^* in the Insertion Algorithm



(b) Maintenance of k-Orders

[1] Zhang Y, Yu J X, Zhang Y, et al. A fast order-based approach for core maintenance[C]//2017 IEEE 33rd International Conference on Data Engineering (ICDE). IEEE, 2017: 337-348.

Edge Insertion

Insert (x_1, x_2) $K = core(x_1, G_0) \leq core(x_2, G_0)$

Coreness Update:

- For each vertex $v \in V^*$, we have $core(v, G_0) = K$ and $core(v, G^*) = K + 1$.
- If $core(x_1, G_0) < core(x_2, G_0)$, we have $V^* \subseteq V(C(x_1))$, the subgraph induced by V^* on G_0 is connected, and $x_1 \in V^*$.
- If $core(x_1, G_0) = core(x_2, G_0)$, we have $V^* \subseteq \{V(C(x_1)) \cup V(C(x_2))\}$. The subgraph induced by V^* on G_0 either is connected, or consists of two connected components that one contains x_1 and the other contains x_2.
- The induced subgraph of V^* in G^* is connected.

Edge Insertion

Hierarchy Analysis:

(i) <u>$k > K + 1$.</u> For every vertex v with $core(v, G_0) > K + 1$, we have $core(v, G^*) = core(v, G_0)$. C_i^k keeps the same after the insertion, as C_i^k does not contain x_1, x_2, or any vertex in V^*.

(ii) <u>$k \leq K$.</u> (a) If C_i^k contains either x_1 or x_2. W.l.o.g, suppose we have $x_1 \in C_i^k$, the insertion of (x_1, x_2) will connect (merge) C_i^k and $C^k(x_2)$. (b) Besides, if $k = K$, the coreness of each vertex in V^* increases to $K + 1$ from K. C_i^k may lose some vertices(i.e., in V^*) and we will discuss this case in details later.

(iii) <u>$k = K + 1$.</u> The vertices in V^* may connect to C_i^k on G^*. We will discuss this case later too.

Edge Insertion

THEOREM 1. *For any tree node* $n_0 \in T_0$ *satisfying* $G_0[n_0] \cap \{C(x_1) \cup C(x_2)\} = \emptyset$, *we have* $T'(n_0)$ *keeps the same in* T^*.

PROOF. Let $k_0 = core(n_0)$. As $G_0[n_0] \cap \{C(x_1) \cup C(x_2)\} = \emptyset$ and $V^* \subseteq V(C(x_1) \cup C(x_2))$, in core decomposition of G^*, the vertices in all ancestors of n_0 will still be deleted when we compute the k_0-core set of G^*. Thus, $T'(n_0)$ keeps the same in T^*. □

Edge Insertion

Algorithm 1: **InsertOne**

Input : a graph G_0, the k-core hierarchy T_0, an edge $(x_1, x_2) \notin E(G_0)$

Output : T^*

$T \leftarrow T_0$; $G \leftarrow G_0$; $K \leftarrow core(x_1)$ (suppose $core(x_1) \leq core(x_2)$);

$V^* \leftarrow$ vertices with coreness changed by inserting (x_1, x_2) to G;

$n_1 \leftarrow node(x_1)$; $n_2 \leftarrow node(x_2)$;

while $n_1 \neq n_2$ **do**

    swap n_1 and n_2 **if** $core(n_1) > core(n_2)$;

    $p_1 \leftarrow P(n_1)$; $p_2 \leftarrow P(n_2)$;

    if $core(n_1) = core(n_2)$ **then**

        $n_0 \leftarrow$ merge n_1 and n_2 in T;

        $P(n_0) \leftarrow p_1$ or p_2 whose coreness is larger;

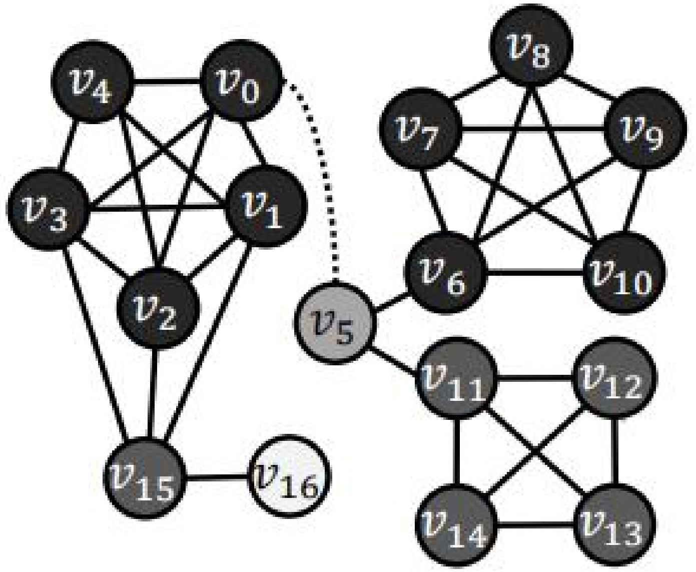
        $n_1 \leftarrow p_1$; $n_2 \leftarrow p_2$;

    else

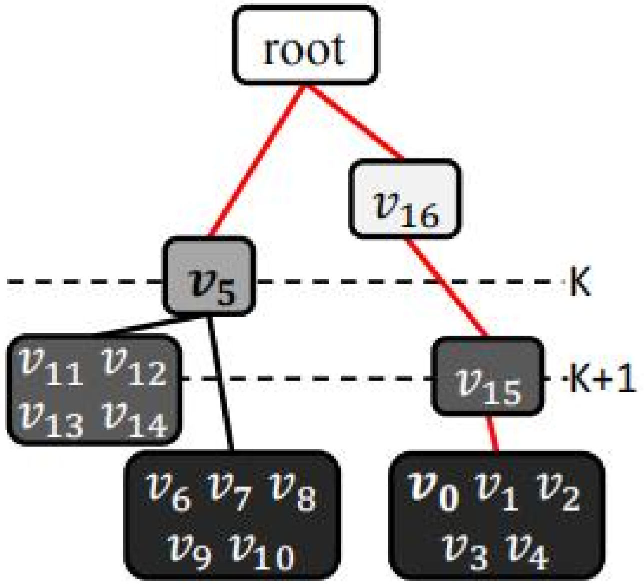
        $P(n_2) \leftarrow n_1$ **if** $core(n_1) > core(p_2)$;

        $n_2 \leftarrow p_2$;

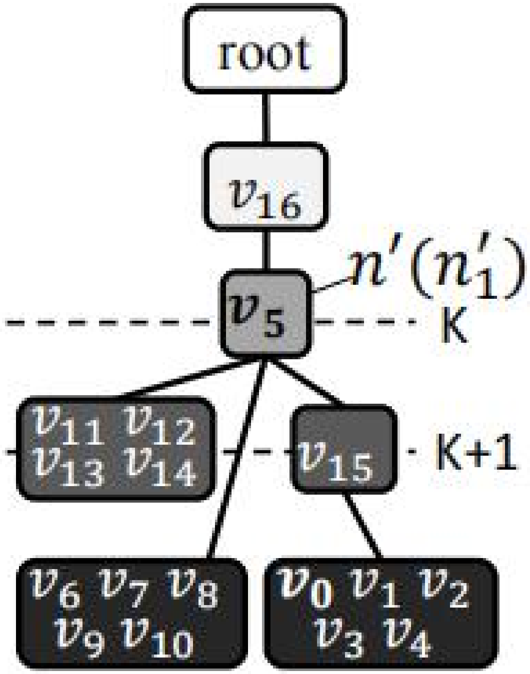
Edge Insertion



(a) $G_0 + (v_0, v_5)$



(b) T_0



(c) T_1

Edge Insertion

THEOREM 2. *(i) There is a node $n'_1 \in T_1$ satisfying $V^* \subseteq V(n'_1)$. (ii) For any node $n_0 \in T_1$ with $G^*[n_0] \cap G^*[n'_1] = \emptyset$, $T'(n_0)$ keeps the same in T^*.*

PROOF. (i) When $core(x_1, G_0) < core(x_2, G_0)$, we have $n'_1 = node(x_1, T_1)$ and $V^* \subseteq V(n'_1)$. When $core(x_1, G_0) = core(x_2, G_0)$, since $node(x_1, T_0)$ and $node(x_2, T_0)$ are merged in T_1, we have $V(n'_1) = V(node(x_1, T_0)) \cup V(node(x_2, T_0))$, and thus $V^* \subseteq V(n'_1)$. (ii) Similar to Theorem 1, for any node n_0 with $G^*[n_0] \cap G^*[n'_1] = \emptyset$, core decomposition on $G^*[n_0]$ is the same to that on $G[n_0]$. Thus, $T'(n_0)$ keeps the same in T^*. □

Edge Insertion

THEOREM 3. *There is a node $n^* \in T^*$ satisfying $V^* \subseteq V(n^*)$.*

Edge Insertion

$T_1 \leftarrow T$; $n' \leftarrow node(V^*)$ of T;

create a node n^+ on L_{K+1} in T as a child of n';

move v to $V(n^+)$ from $V(n')$ **for** each $v \in V^*$;

$NC = \{cn(n', u, T) \mid u \in N(V^*, G^*)\}$; $T_2 \leftarrow T$;

for each $n_c \in NC$ **do**

  if $core(n_c, G^*) = K + 1$ **then**

    merge n_c into n^+;

  else

    $P(n_c, T) \leftarrow n^+$;

if $V(n') = \emptyset$ **then**

  $P(n_0) \leftarrow P(n')$ for each child n_0 of n';

  remove n' from T;

return T (i.e., T^*)

Edge Insertion

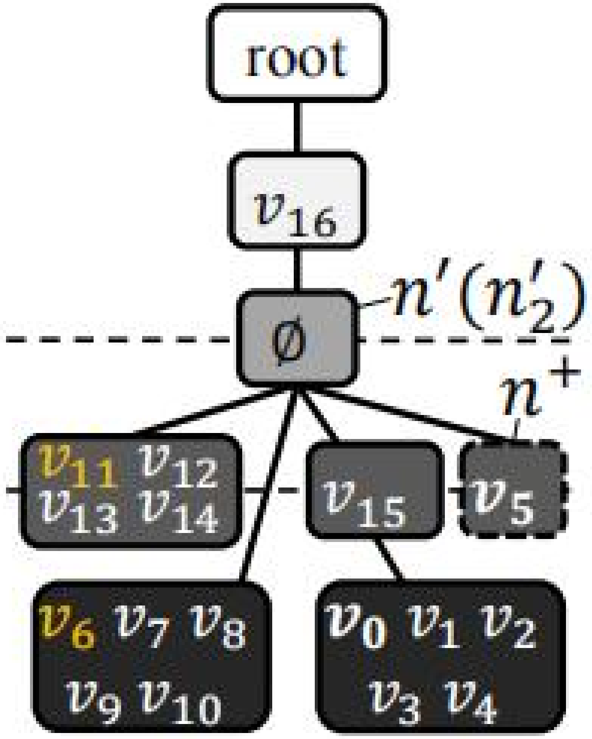
$cn(n_0, v_0) = \{n_c | P(n_c) = n_0 \text{ and } T'(n_c) \text{ contains } v_0\}$

$NC = \{cn(n_2', u) | u \in N(V^*, G^*)\}$ $N(V^*, G^*) = \cup_{v \in V^*} N(v, G^*)$

Theorem 4. *For each $n_c \in NC$, $G^*[n_c] \subseteq G^*[n^*]$ holds.*

$i)\ core(n_c) \geq core(n^+) = K + 1$

$ii)\ \exists v_1 \in n_c,\ v_2 \in n^+ \text{ satisfying } (v_1, v_2) \in E(G^*)$



(d) T_2

Edge Insertion

$T_1 \leftarrow T$; $n' \leftarrow node(V^*)$ of T;

create a node n^+ on L_{K+1} in T as a child of n';

move v to $V(n^+)$ from $V(n')$ **for** each $v \in V^*$;

$NC = \{cn(n', u, T) \mid u \in N(V^*, G^*)\}$; $T_2 \leftarrow T$;

for each $n_c \in NC$ **do**

    if $core(n_c, G^*) = K + 1$ **then**

        merge n_c into n^+;

    else

        $P(n_c, T) \leftarrow n^+$;

if $V(n') = \emptyset$ **then**

    $P(n_0) \leftarrow P(n')$ for each child n_0 of n';

    remove n' from T;

return T (i.e., T^*)

Edge Insertion

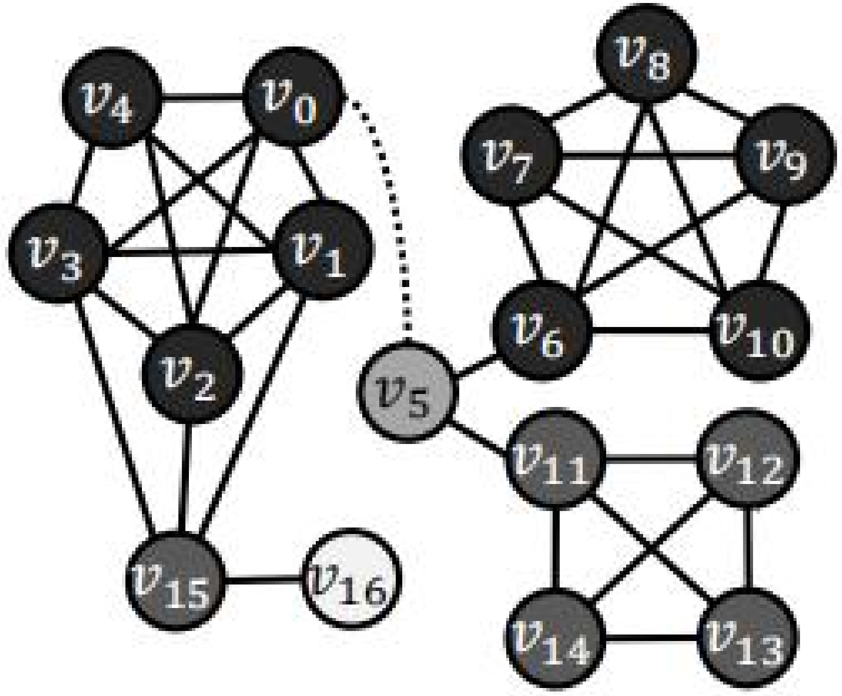
Algorithm 5: FindSubroot

Input : a node n_0, a vertex v_0

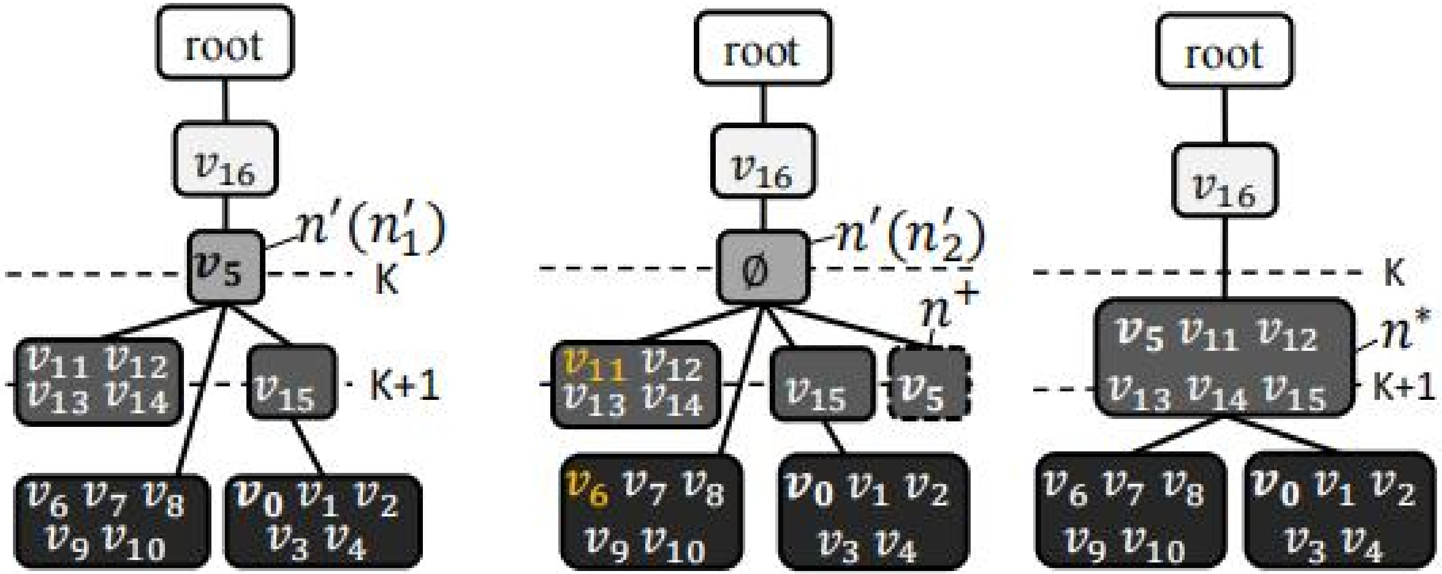
Output : the node n_c, i.e., $cn(n_0, v_0)$

```
A ← empty set; n_c ← n_1 ← node(v_0);
while n_1 ≠ n_0 do
    A ← A ∪ {n_1};
    n_c ← n_1; n_1 ← Jump(n_1);
Jump(n_2) ← n_c for each node n_2 ∈ {A \ n_c};
return n_c
```

Edge Insertion



(a) $G_0 + (v_0, v_5)$



(c) T_1 (d) T_2 (e) T^*

Figure 2: Insert (v_0, v_5) to G_0

Edge Insertion

Algorithm 1: InsertOne

Input : a graph G_0, the k-core hierarchy T_0, an edge $(x_1, x_2) \notin E(G_0)$

Output : T^*

$T \leftarrow T_0; G \leftarrow G_0; K \leftarrow core(x_1)$ (suppose $core(x_1) \leq core(x_2)$);
$V^* \leftarrow$ vertices with coreness changed by inserting (x_1, x_2) to G;
$n_1 \leftarrow node(x_1); n_2 \leftarrow node(x_2)$;
while $n_1 \neq n_2$ **do**
    swap n_1 and n_2 **if** $core(n_1) > core(n_2)$;
    $p_1 \leftarrow P(n_1); p_2 \leftarrow P(n_2)$;
    if $core(n_1) = core(n_2)$ **then**
        $n_0 \leftarrow$ merge n_1 and n_2 in T;
        $P(n_0) \leftarrow p_1$ or p_2 whose coreness is larger;
        $n_1 \leftarrow p_1; n_2 \leftarrow p_2$;
    else
        $P(n_2) \leftarrow n_1$ **if** $core(n_1) > core(p_2)$;
        $n_2 \leftarrow p_2$;
$T_1 \leftarrow T; n' \leftarrow node(V^*)$ of T;
create a node n^+ on L_{K+1} in T as a child of n';
move v to $V(n^+)$ from $V(n')$ **for** each $v \in V^*$;
$NC = \{cn(n', u, T) \mid u \in N(V^*, G^*)\}; T_2 \leftarrow T$;
for each $n_c \in NC$ **do**
    if $core(n_c, G^*) = K + 1$ **then**
        merge n_c into n^+;
    else
        $P(n_c, T) \leftarrow n^+$;
if $V(n') = \emptyset$ **then**
    $P(n_0) \leftarrow P(n')$ for each child n_0 of n';
    remove n' from T;
return T (i.e., T^*)

Edge Insertion

Algorithm 1: InsertOne

Input : a graph G_0, the k-core hierarchy T_0, an edge $(x_1, x_2) \notin E(G_0)$

Output : T^*

$T \leftarrow T_0; G \leftarrow G_0; K \leftarrow core(x_1)$ (suppose $core(x_1) \le core(x_2)$);

$V^* \leftarrow$ vertices with coreness changed by inserting (x_1, x_2) to G; $\rightarrow$ $O(\log \max\{|O_K|, |O_{K+1}|\} \times \sum_{v \in V^+} |N(v, G^*)|)$

$n_1 \leftarrow node(x_1); n_2 \leftarrow node(x_2)$;

while $n_1 \neq n_2$ **do**

  swap n_1 and n_2 **if** $core(n_1) > core(n_2)$;

  $p_1 \leftarrow P(n_1); p_2 \leftarrow P(n_2)$;

  if $core(n_1) = core(n_2)$ **then** $\rightarrow$ $O(k_{max})$

    $n_0 \leftarrow$ merge n_1 and n_2 in T;

    $P(n_0) \leftarrow p_1$ or p_2 whose coreness is larger;

    $n_1 \leftarrow p_1; n_2 \leftarrow p_2$;

  else

    $P(n_2) \leftarrow n_1$ **if** $core(n_1) > core(p_2)$;

    $n_2 \leftarrow p_2$;

Edge Insertion

$$O(|O_K|)$$

$$O(|T'(node(x_1))| + |T'(node(x_2))|)$$

$T_1 \leftarrow T$; $n' \leftarrow node(V^*)$ of T;
create a node n^+ on L_{K+1} in T as a child of n';
move v to $V(n^+)$ from $V(n')$ **for** each $v \in V^*$;
$NC = \{cn(n', u, T) \mid u \in N(V^*, G^*)\}$; $T_2 \leftarrow T$;
for each $n_c \in NC$ **do**
    if $core(n_c, G^*) = K + 1$ **then**
        merge n_c into n^+;
    else
        $P(n_c, T) \leftarrow n^+$;
if $V(n') = \emptyset$ **then**
    $P(n_0) \leftarrow P(n')$ for each child n_0 of n';
    remove n' from T;
return T (i.e., T^*)

Edge Insertion

Algorithm 2: InsertX

Input : a graph G_0, the k-core hierarchy T_0, an edge set $E' \nsubseteq E(G_0)$

Output : T^*, i.e., the updated T_0

$V^* \leftarrow \emptyset; C \leftarrow \emptyset; G^* \leftarrow G_0; T \leftarrow T_0;$

for each $e \in E'$ **do**

  $V' \leftarrow$ vertices with coreness changed by inserting e to G^*;

  $\mathbb{N} \leftarrow$ the set of $node(v)$ in T **for** each $v \in V'$;

  $n' \leftarrow$ any node from $\mathbb{N}$;

  create n^* on $(core(n') + 1)^{th}$ layer in T as a child node of n';

  $C \leftarrow C \cup \{(n^*, n_0)\}$ **for** each $n_0 \in \mathbb{N}$;

  move each $v \in V'$ to $V(n^*)$; remove empty nodes in T;

  $G_0 \leftarrow G_0 + \{e\}; V^* \leftarrow V^* \cup V';$

$T_1 \leftarrow T;$

for each $(u, v) \in E'$ **do**

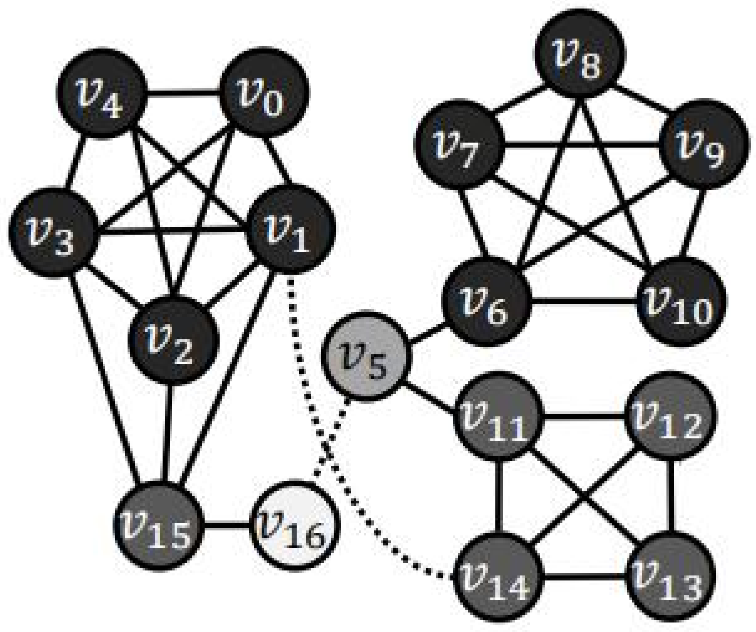
  $C \leftarrow C \cup (node(u, T), node(v, T));$

for each $v \in V^*$ **do**

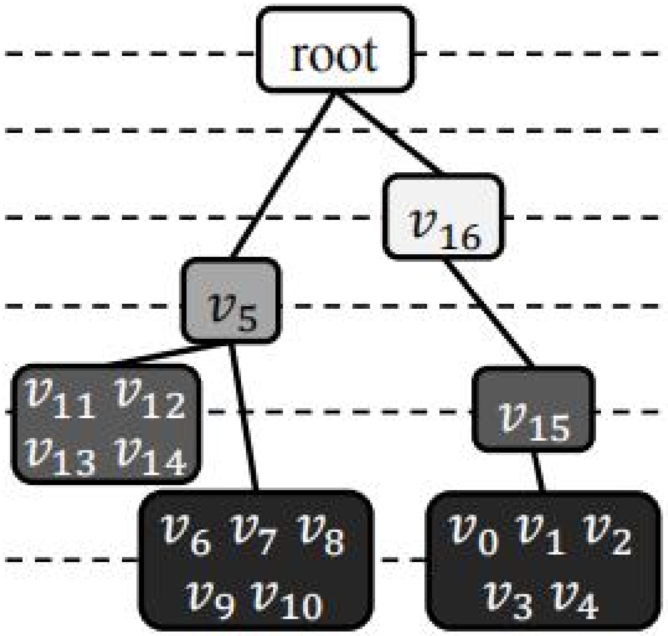
  for each $u \in N(v, G^*)$ with $core(u, G^*) > core(v)$ **do**

    $C \leftarrow C \cup (node(u, T), node(v, T));$

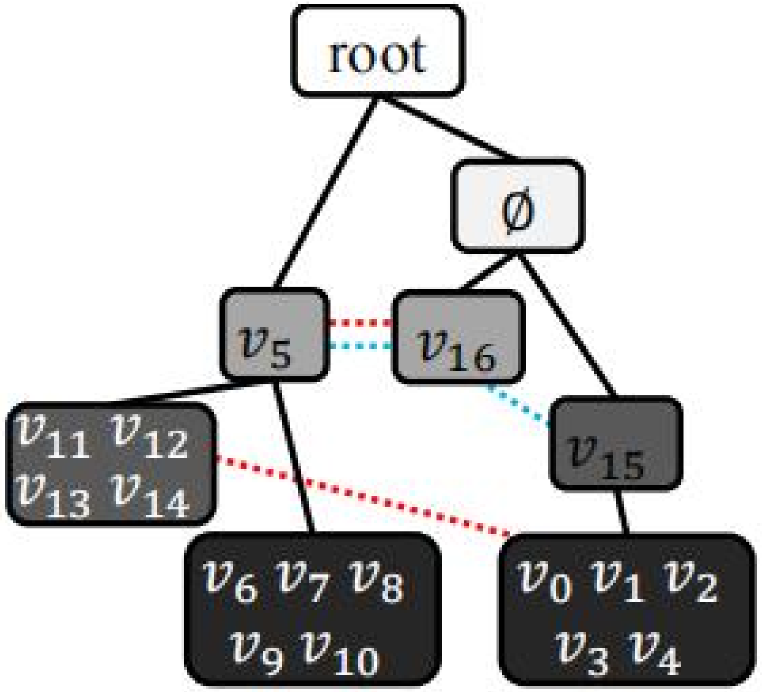
Edge Insertion



(a) $G_0 + (v_1, v_{14}) + (v_5, v_{16})$



(b) T_0



(c) T_1

Edge Insertion

for each integer K from k_{max} to 0 **do**

    $n_0 \leftarrow$ an unvisited node in a node pair of C with $core(n_0) = K$;

    $\mathbb{N}_1 \leftarrow \{n_0\}$; $\mathbb{N}_2 \leftarrow \emptyset$;

    while there is an unvisited node n_1 in $\mathbb{N}_1$ **do**

        $\mathbb{N}_2 \leftarrow \mathbb{N}_2 \cup \{P(n_1)\}$; $n_1 \leftarrow$ visited;

        for each node n_2 with $(n_1, n_2) \in C$ **do**

            if $core(n_2) = K$ **then**

                $\mathbb{N}_1 \leftarrow \mathbb{N}_1 \cup \{n_2\}$;

            else

                $\mathbb{N}_2 \leftarrow \mathbb{N}_2 \cup \{n_2\}$;

    $n' \leftarrow$ a node in $\mathbb{N}_2$ with the largest coreness;

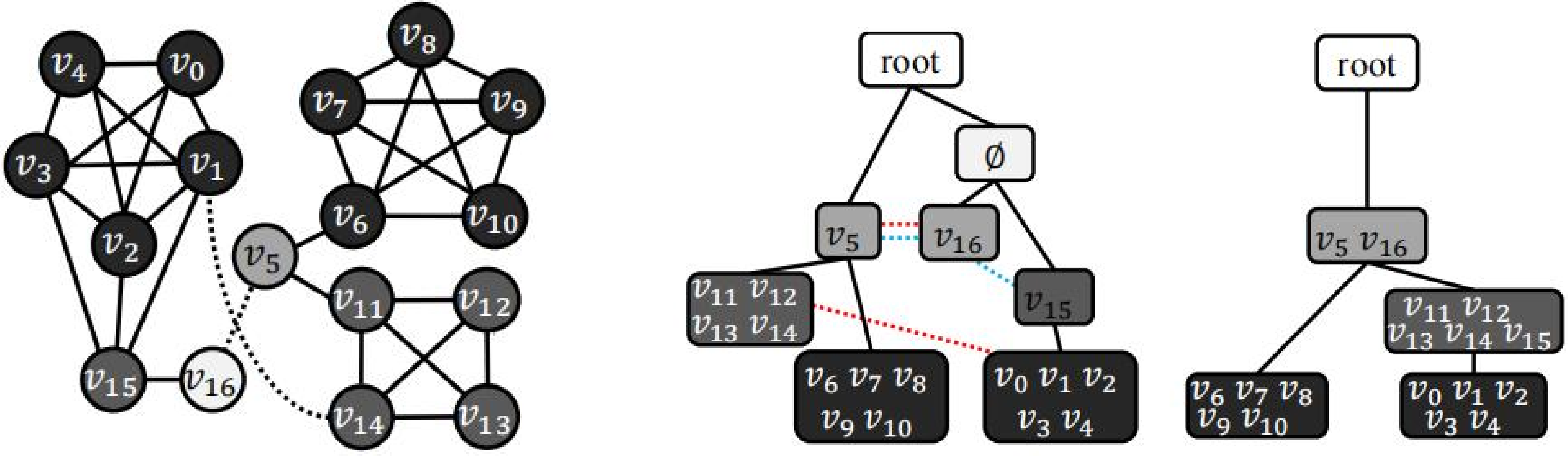
    $C \leftarrow C \cup (n', n_2)$ **for** each $n_2 \in \mathbb{N}_2$;

    merge n_1 into n_0 **for** each $n_1 \in \mathbb{N}_1$;

    $P(n_0) \leftarrow n'$;

return T, i.e., T^*;

Edge Insertion



(a) $G_0 + (v_1, v_{14}) + (v_5, v_{16})$ (c) T_1 (d) T^*

Complexity: $O(\sum |V^*| + x.(k_{max} + |T_0|))$

Edge Removal

Coreness Update:

- For every vertex $v \in V^*$, we have $core(v, G_0) = K$ and $core(v, G^*) = K - 1$.
- We have $V^* \subseteq V(C(x_1, G_0))$, and the induced subgraph of V^* in G_0 is connected.

Edge Removal

k-core hierarchy Update:

(i) The k-cores with $k > K$. For every vertex v with $core(v) > K$, we have $C(v, G^*) = C(v, G_0)$, because $(x_1, x_2) \notin C(v, G_0)$. Thus, the hierarchy of k-cores (the subtrees rooted on L_k) with $k > K$ keeps the same in G_0 and G^*.

(ii) The k-cores with $k \leq K$. For every vertex v with $core(v) < K$, we have $core(v, G^*) = core(v, G_0)$. The removal of (x_1, x_2) will move the vertices in V^* to L_{K-1} from L_K. Besides, the ancestors of $node(x_1)$ or $node(x_2)$ may split, because some k-cores become disconnected by the removal of (x_1, x_2) and the move of the vertices in V^*.

Edge Removal

Algorithm 3: RemoveOne

Input : a graph G_0, the k-core hierarchy T_0, an edge $(x_1, x_2) \in E(G_0)$

Output : T^*, i.e., the updated T_0

$T \leftarrow T_0$;
$V^* \leftarrow$ vertices with coreness changed by removing (x_1, x_2) from G_0;
$n' \leftarrow$ node(x_1) in T (suppose $core(x_1) \leq core(x_2)$);
$n^* \leftarrow P(n')$;
if $V^* \neq \emptyset$ **then**
    if $core(P(n')) \neq K - 1$ **then**
        create n_0 on L_{K-1} as a child node of n^* ;
        $n^* \leftarrow n_0$;
        $P(n') \leftarrow n^*$;
    move each vertex in V^* from n' to n^*;

Edge Removal

$T_1 \leftarrow T$;

$T_2 \leftarrow$ **SplitNode**(n', T_1);

Algorithm 4: SplitNode

Input : a subtree rooted at n_r to split, the k-core hierarchy T

Output : the updated T

$n_r^* \leftarrow P(n_r)$; $V_r \leftarrow V(n_r)$; $K = core(n_r)$;

for each vertex $u \in V(n_r)$ **do**

  create an empty node n_c on L_K as a child node of n_r;

  move u to n_c from n_r;

for each node $n_c \in n_r$.children **do**

  for each node $n_d \in T'(n_c)$ **do**

    $cn(n_r, n_d) \leftarrow n_c$;

for each vertex $u \in V_r$ **do**

  for each vertex $v \in N(u, G^*)$ **do**

    if $core(v, G^*) = K$ **then**

      merge $node(u)$ and $node(v)$;

    else if $core(v, G^*) > K$ **then**

      $n_c \leftarrow cn(n_r, v)$;   /* FindSubroot(n', v) */

      if $P(n_c) = n_r$ **then**

        $P(n_c) \leftarrow node(u)$;

      else

        merge $node(u)$ and $P(n_c)$;

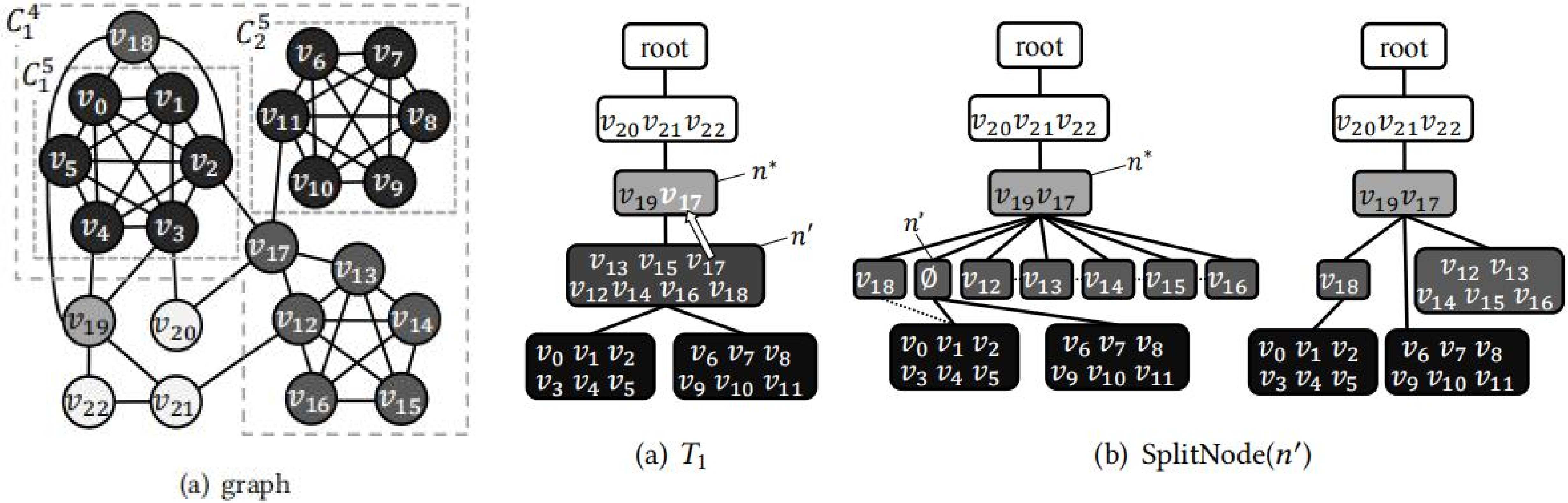
for each node $n_c \in n_r$.children **do**

  $P(n_c) \leftarrow n_r^*$;

remove n_r from T;

return T, i.e., updated T

Edge Removal



$$delete(v_2, v_{17})$$

Edge Removal

$T_1 \leftarrow T$;

$T_2 \leftarrow$ **SplitNode**(n', T_1);

flag $\leftarrow$ true;

$i \leftarrow 2$;

while flag = true **do**

    $i \leftarrow i + 1$; $n_i^* = P(n_{i-1}^*)$;

    $T_i \leftarrow$ **SplitNode**(n_i^*, T_{i-1});

    flag $\leftarrow (T_{i-1} \neq T_i)$;

$T^* \leftarrow T_i$;

return T^*

Edge Removal

Algorithm 3: RemoveOne

Input : a graph G_0, the k-core hierarchy T_0, an edge $(x_1, x_2) \in E(G_0)$

Output : T^*, i.e., the updated T_0

$T \leftarrow T_0$;
$V^* \leftarrow$ vertices with coreness changed by removing (x_1, x_2) from G_0;
$n' \leftarrow$ node(x_1) in T (suppose $core(x_1) \leq core(x_2)$);
$n^* \leftarrow P(n')$;
if $V^* \neq \emptyset$ **then**
    if $core(P(n')) \neq K - 1$ **then**
        create n_0 on L_{K-1} as a child node of n^* ;
        $n^* \leftarrow n_0$;
        $P(n') \leftarrow n^*$;
    move each vertex in V^* from n' to n^*;
$T_1 \leftarrow T$;
$T_2 \leftarrow$ **SplitNode**(n', T_1);
flag $\leftarrow$ true;
$i \leftarrow 2$;
while flag = true **do**
    $i \leftarrow i + 1$; $n_i^* = P(n_{i-1}^*)$;
    $T_i \leftarrow$ **SplitNode**(n_i^*, T_{i-1});
    flag $\leftarrow (T_{i-1} \neq T_i)$;
$T^* \leftarrow T_i$;
return T^*

Edge Removal

Algorithm 6: RemoveX

Input : a graph G_0, the k-core hierarchy T_0, an edge set $E' \subseteq E(G_0)$

Output : T^*, i.e., the updated T_0

$T \leftarrow T_0$; $G \leftarrow G_0$; $C \leftarrow \emptyset$;

for each $(u, v) \in E'$ **do**

- $V^* \leftarrow$ vertices with coreness changed by removing (u, v) from G;
- $G \leftarrow G - (u, v)$;
- $node' \leftarrow node(u, T_i)$ (suppose $K = core(u, G) \leq core(v, G)$);
- **if** $core(P(node')) = K - 1$ **then**
  - $node^* \leftarrow P(node')$;
- **else**
  - create an empty node $node^*$ on L_{K-1} as a child of $P(node')$;
  - $P(node') \leftarrow node^*$;
- move each vertex $v \in V^*$ from $node'$ to $node^*$;
- $C \leftarrow C \cup \{node', node^*\}$;

$T_1 \leftarrow T$; $G^* \leftarrow G$; $i = 1$;

for each $n' \in C$ in descending order of coreness **do**

- $i \leftarrow i + 1$;
- $T_i \leftarrow$ **SplitNode**(n', T_{i-1});
- $C \leftarrow C \cup \{P(n')\}$ **if** $T_{i-1} \neq T_i$;

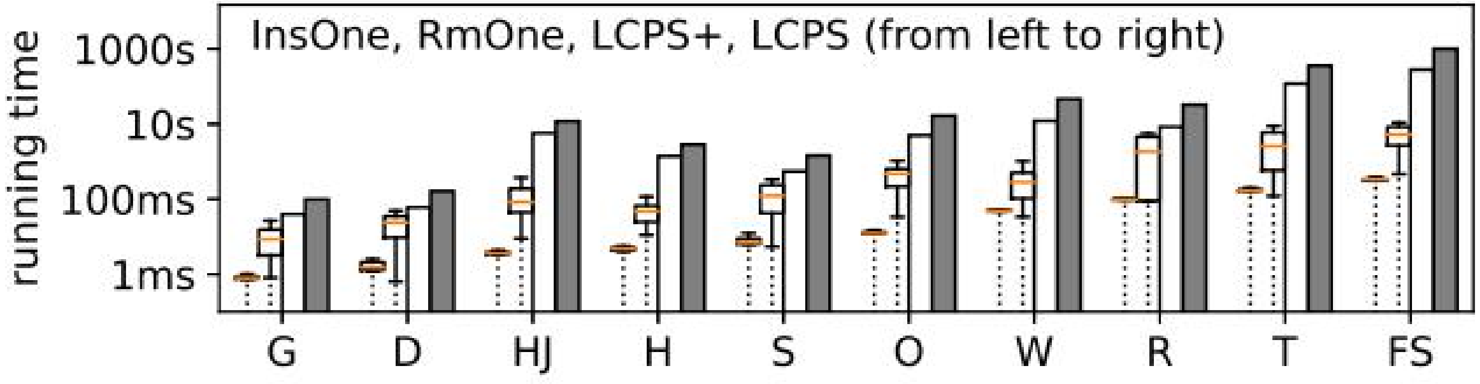
return T, i.e., T^*

Datasets

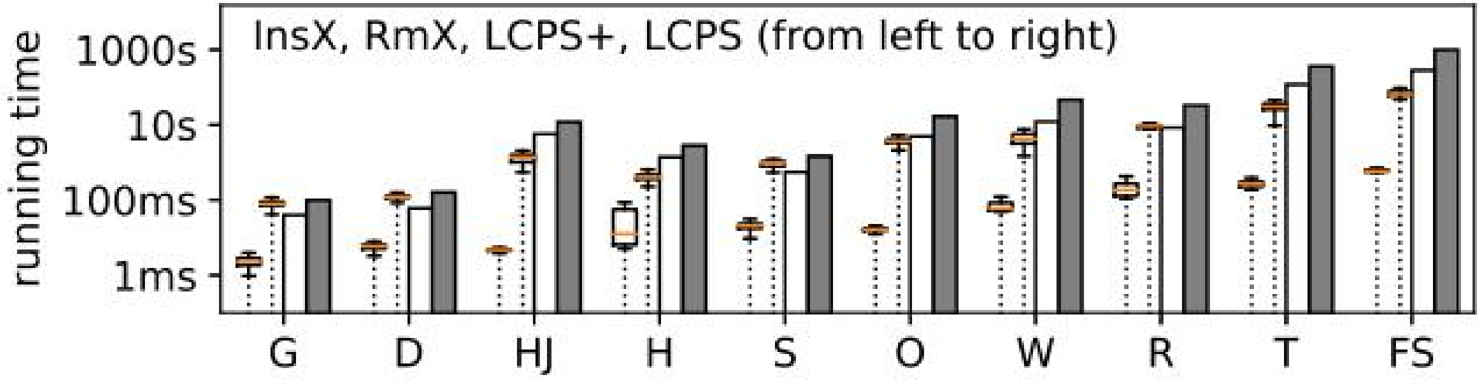
Table 2: Statistics of Datasets

Dataset	$\|V\|$	$\|E\|$	d_{avg}	k_{max}	$\|T\|$
Gowalla	196,591	950,327	9.7	51	75
DBLP	317,080	1,049,866	6.6	113	767
Human-Jung	784,262	267,844,669	683.1	1200	4088
Hollywood	1,069,126	56,306,653	105.3	2208	679
Skitter	1,696,415	11,095,298	13.1	131	903
Orkut	3,072,441	117,185,083	76.3	253	254
Wiki	12,150,976	378,142,420	62.2	1122	5049
Rgg	16,777,216	132,557,200	15.8	20	117422
Twitter	41,652,230	1,468,365,182	8.8	2488	3049
FriendSter	65,608,366	1,806,067,135	55.1	304	451

Performance



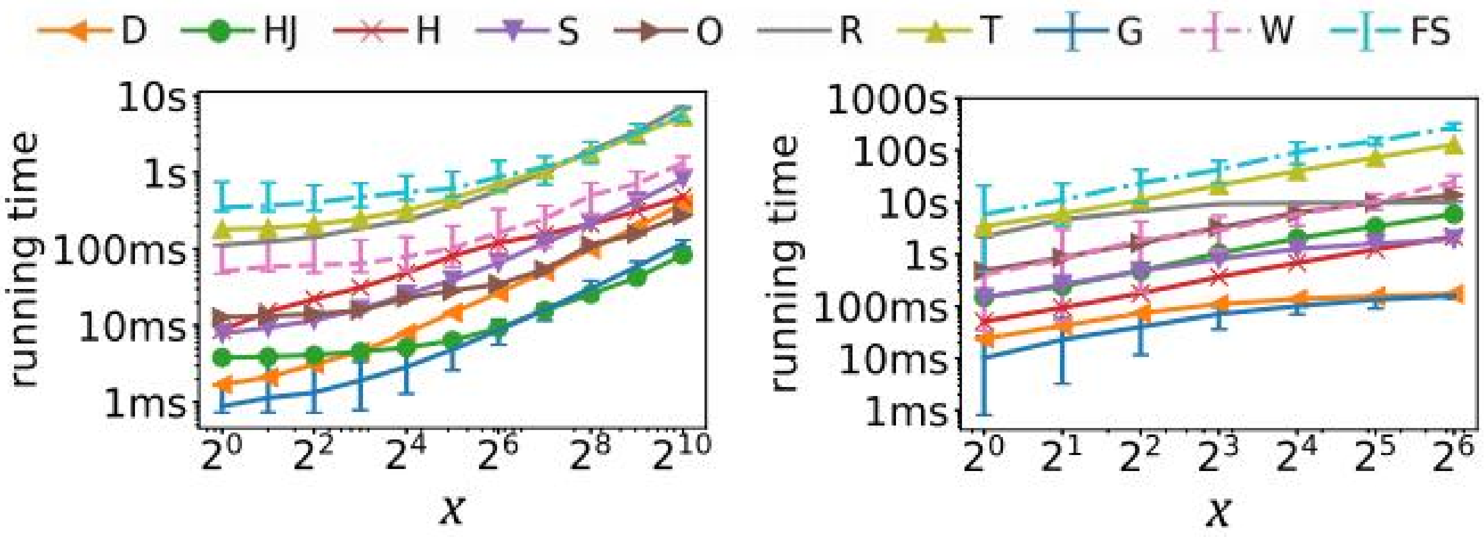
(a) single edge



(b) 10 edges

Figure 5: Performance on All the Datasets

Performance



(a) Insertion (InsX) (b) Removal (RmX)

Figure 6: Performance on Inserting/Removing *x* Edges

Performance

Table 3: The engagement of users in node n_1, compared with the parent node of n_1 (T-edge), or the nodes with smaller sub-trees (T-size), on DBLP from Year 19-20 (Win Percent)

k	1	2	3	4	5	6	7	8	total
T-edge(%)	-	100	99.7	98.9	100	100	100	100	99.44
T-size(%)	80.9	78.6	86.2	93.4	80.6	44.1	100	100	84.58

Conclusion and Future Work

Problem: **Maintaining the k-core hierarchy on dynamic graphs**

- We propose effective local update techniques.
- Our algorithms for updating the *k*-core hierarchy largely outperform the baselines for one or a small batch of updated edge(s).
- Our approach may be adapted to other decompositions if they hold the same hierarchical structure
- Besides, the framework of our algorithms may inspire a sound solution for parallel maintenance of k-core hierarchy.

THANK YOU

Xiaowei Lv