A PROOFS OF LEMMAS AND THEOREMS

Theorem 3.6 `ICPIns` *and* `ICPDel` *are not relatively unbounded algorithms.*

Proof. The graph G in Figure 1 provides an example to illustrate the unboundedness of `ICPIns` and `ICPDel`. When edge (v_8, v_{11}) is inserted to (*resp. deleted from*) G_2, the ICS decomposition takes $O(1)$ change in two of the influential communities $\{v_1, v_2\}$ and $\{v_8, v_9, v_{10}, v_{11}\}$, where v_8 move from one ICS to the other, as shown in Figure 2. That is, $|\mathsf{AFF}| = O(1)$, however, as `ICPIns` and `ICPDel` involves recomputing $\mathcal{T}_2$, their time complexity on maintaining ICS for G_2 are both $O(|E_2|)$. Thus, we have $M(G, \Delta G) = O(|E_2|)$, which cannot be expressed as a polynomial of $|\mathsf{AFF}|$ and $|\mathsf{Q}|$. Therefore, both `ICPIns` and `ICPDel` are relatively unbounded. □

Lemma 4.2 *Given a graph $G_k = (V_k, E_k)$ and its ICD-Order O_k, a vertex $u \in O_k$ is a keynode of G_k, if and only if, $d_{O_k}(u, G_k) \geq k$.*

Proof. We first prove that if u is a keynode of G_k, then $d_{O_k}(u, G_k) \geq k$. Consider a subgraph g of G_k induced by the vertex set $\{w \in V_k \mid u \preceq w\}$. Clearly, g is a k-IC with u as keynode. Therefore, $d_{O_k}(u, G_k) = |N(u, g)| \geq k$. On the other hand, if $d_{O_k}(u, G_k) \geq k$, we can always find a k-core g containing u in G_k, in which u is the vertex with the smallest order in g. This is because, after removing u from G_k, at least k of its neighbors remain, allowing u and these vertices to easily form a k-core. □

Theorem 5.2 *Given a graph G, a set of edges ΔG, and G's ICD-Order O, Algorithm 2 can compute $G \oplus \Delta G$'s ICD-order O^+ correctly.*

Proof. We prove that Ψ_t qualifies the ICD-Order of G_k^+ by providing that it statifies the each condition in the PROPERTY 1 separately:

(1) We prove that all keynodes in Ψ_t must statify the **condition 1** in PROPERTY 1. It is worth noting that for each $i \in [1, t]$, **Case a** and **Case b(i)** are the **only cases** that result in vertices from $\mathcal{R}_i \cup \mathcal{P}_i$ being included in Ψ_i as key nodes. For **Case a**, p^* is a new keynode, we have $d_{\Psi_t}(p^*, G_k^+) = \Gamma(p^*) \geq k$. For **Case b(i)**, we have $d_{\Psi_t}(\pi(\mathcal{P}_i), G_k^+) \geq d_{O_k}(\pi(\mathcal{P}_i), G_k)$.

(2) For **condition 2**, consider a keynode $w \in \Psi_t$ and for each vertex $v \in \Psi_t$ with $v \preceq w$, there are two cases for w: (1) w is not appended to $\mathcal{P}_i$ for any state $i \in [1, t]$; and (2) there exists a state $i \in [1, t]$ such that $w \in \mathcal{P}_i$. For the first case, we need to consider two sub-cases in the vertex sequence Ψ_t based on whether $v \in \mathcal{D}_k$.

- (i) $v \in \mathcal{D}_k$. we have $|N(w, G_k) \cap \{u \mid v \preceq u \wedge u \in \Psi_t\}| \geq |N(w, G_k) \cap \{u \mid v \preceq u \wedge u \in O_k\}| \geq k$, where $\preceq$ denotes the vertex order in Ψ_t;
- (ii) $v \notin \mathcal{D}_k$. v is a new keynode in G_k^+, and $|N(w, G_k) \cap \{u \mid v \preceq u \wedge u \in \Psi_t\}| \geq |N(w, G_k) \cap \{u \mid \mathcal{D}_{\Psi_t}[w] \preceq u \wedge u \in \Psi_t\}| \geq k$, where $\mathcal{D}_{\Psi_t}[w]$ is w's corresponding keynode in Ψ_t. This is because the new keynode v must be either before $\mathcal{D}_{\Psi_t}[w]$ or after w in Ψ_t.

For the second case, assume that w is appended into $\mathcal{P}_i$ at state i and v is appended into Ψ_j at state j. Here, two sub-cases are based on the relationship of i and j. (i) if $i > j$, **condition 2** is clearly satisfied, and (ii) if $i < j$, consider the j-th state, vertices are iteratively moved from the union of sets $\mathcal{R}_j$ and $\mathcal{P}_j$ into Ψ_j. Now, considering the moment when v is removed from $\mathcal{R}_j \cup \mathcal{P}_j$, we observe that

$$\left|N(w, G_k^+) \cap \{u \mid v \preceq u \wedge u \in \Psi_t\}\right|$$

is exactly the supremum degree of w. This is because, at that time, v is the last vertex in Ψ_j, implying all other vertices in Ψ_j precede v in Ψ_t. In addition, all the remaining vertices in $\mathcal{R}_j$ or $\mathcal{P}_j$ are succeed v in the Ψ_t, i.e., $\forall u \in (\mathcal{R}_j \cup \mathcal{P}_j)$, we have $v \prec u$. That is, we include the following equalition:

$$\begin{aligned} N(w, G_k^+) \cap \{u \mid v \prec u \wedge u \in \Psi_t\}| &= |N(w, G_k^+) \cap (\mathcal{R}_j \cup \mathcal{P}_j)| \\ &= \Gamma(w). \end{aligned}$$

Therefore, $|N(w, G_k^+) \cap \{u \mid v \preceq u \wedge u \in O_k\}| \geq k$.

(3) For **condition 3**, for a *cvs* $v \in \Psi_t$, suppose that in the i-th state, v is appended to Ψ_i, and there are also two cases for v: (1) $v \in \mathcal{P}_i$, and (2) $v \in \mathcal{R}_i$. For the first case, clearly, we have $d_{\Psi_t}(v, G_k^+) = \Gamma(v) < k$. For the second case, we prove this by contraction. Assume that $\Gamma(v) \geq k > d_{O_k}(v, G_k)$, that is, there must exist a state j ($j < i$), v=$\pi(\mathcal{P}_j)$, and $\omega(p^*) > \omega(v)$. Consider in the j-th state, since $\Gamma(v) \geq k$, v is appended to $\mathcal{P}_{j+1}$ (**Case b(iii)**) and would not be in $\mathcal{R}_i$. Therefore, $d_{\Psi_i}(v, G_k^+) = \Gamma(v) < k$ must hold. □

Theorem 5.3 *The time complexity of Algorithm 2 for inserting an edge (u, v) is $O(\sum_{k=1}^{\widetilde{c}} \mathrm{vol}(|\mathtt{diff}_k|) \log |V_k|)$, where $\widetilde{c} = \min\{c(u), c(v)\}$ +1, V_k and $\mathtt{diff}_k$ denote the set of vertices in G_k and the difference between ICD-Orders of O_k and O_k^+, respectively.*

Proof. We start by analyzing the number of times entering the while loop for a given G_k. We first aim to show: $\left(\bigcup_{i=0}^{t} \mathcal{P}_i\right) \subseteq \mathtt{diff}_k$.

In the i-th state, a vertex u would be added to $\mathcal{P}_i$, if and only if case **Case b(iii)** occurs. In this case, u is not a keynode in G_k. Let v be the first vertex that needs to be removed from $\mathcal{P}_i$ after u is added to $\mathcal{P}_i$. If $v \neq u$, then $u \preceq v$ and $v \preceq^+ u$, implying $u \in \mathtt{diff}_k$. Otherweise, $d_{O_k^+}(u, G_k^+) = \Gamma(u) \geq k$, implying $u \in \mathcal{D}_k^+$ and $u \in \mathtt{diff}_k$. In both cases, $u \in \mathtt{diff}_k$, therefore $\left(\bigcup_{i=0}^{t} \mathcal{P}_i\right) \subseteq \mathtt{diff}_k$.

Then, we can conclude that the number of iterations of the while loop is bounded by $O(\mathrm{vol}(\mathtt{diff}_k))$. This is because, in each iteration, either $\pi(\mathcal{P}_i)$ is removed from $\mathcal{R}_i$, or p^* is removed from $\mathcal{P}_i$. Note that $\pi(\mathcal{P}_i)$ is a neighbor of some vertex in $\mathcal{P}_i$, which implies that the total degree:

$$\sum_{u \in \left(\cup_{j=0}^{t} \mathcal{P}_j\right)} |N(u, G_k^+[\mathcal{R}_i])|.$$

decreases by at least one in each iteration. Since this quantity is initially bounded by $\mathrm{vol}(\mathtt{diff}_k)$, the total number of iterations is at most $O(\mathrm{vol}(\mathtt{diff}_k))$.

Next, for each iteration of the while loop, the computation costs can be divided into two parts, `DeleteVertex` and others. For the second part, the time complexity of each iteration of the while loop can be bounded by $O(\log |V_k|)$. This is because, the involved operations, including: (1) maintaining p^* in $\mathcal{P}_i$, (2) maintaining $\pi(\mathcal{P}_i)$ among all neighbors of $\mathcal{P}_i$, and (3) calculating $\mathcal{U}_i$, kn and maintaining the vertices' movement between $\mathcal{R}_i$, $\mathcal{U}_i$ and Ψ_i. Those operations can be implemented using a heap and a balanced BST, with their sizes not exceeding $|V_k|$. Thus, In each iteration of the while loop, the above process takes $O(\log|V_k|)$ time. The time complexity of DeleteVertex for all iterations is bounded by $O(\mathrm{vol}(\mathtt{diff}_k))$, since each time a vertex is removed, all of its neighbors need to be enumerated.

Therefore, given an interger k and G_k, the total time complexity of OrderIns is $O(\sum_{k=1}^{\widetilde{c}} \mathrm{vol}(\mathtt{diff}_k)\log|V_k|)$. □

Theorem 5.4 *Given a graph G, its ICD-Order $\mathcal{O}$, and a set of edges ΔG to be inserted into G, Algorithm 2 is relatively bounded with respect to the ICD maintenance algorithm.*

Proof. To analyze the relative boundedness of Algorithm 2, according to the [77], we have $|\mathsf{AFF}| = \sum_{k=1}^{\widetilde{c}} \mathrm{vol}(\mathtt{diff}_k)$ w.r.t. the ICD maintenance algorithms under edge insertion. Thus, for an incremental algorithm M to maintain the new ICD-Order $\mathcal{O}^+$ according to G and $\mathcal{O}$. If the time cost of M is a polynomial of $\sum_{k=1}^{\widetilde{c}} \mathrm{vol}(\mathtt{diff}(\mathcal{O}_k, \mathcal{O}_k^+))$, it is relatively bounded to the ICS Decomposition algorithm.

Based on the definition of $|\mathsf{AFF}|$, we know the time complexity of Algorithm 2 is polynomial of $|\mathsf{AFF}| = \sum_{k=1}^{\widetilde{c}} \mathrm{vol}(\mathtt{diff}(\mathcal{O}_k, \mathcal{O}_k^+))$. Then, according to Theorem 5.3, Algorithm 2 is a relatively bounded algorithm. □

Theorem 6.2 *Given a graph G, a set of edges ΔG, and G's ICD-Order $\mathcal{O}$, Algorithm 3 can compute $G \ominus \Delta G$'s ICD-order $\mathcal{O}'$ correctly.*

Proof. We prove that Ψ_t qualifies the ICD-Order of G_k^- by providing that it statifies the each condition in the Property 1 separately:

(1) We first show that all keynodes in Ψ_t statify the **condition 1** in the Property 1. At the i-th state, if a keynode from $\mathcal{U}_i$ is appended to Ψ_i, it clearly satisfies the condition. Besides, if $\pi(\mathcal{P}_i)$ is a keynode in the new order, it can be appended to Ψ_i at the i-th state if and only if $\pi(\mathcal{P}_i) \in \mathcal{S}_i$ and $\beta(\pi(\mathcal{P}_i)) \geq k$. We observe that $N_{\mathcal{O}_k}(u, G_k^-[\mathcal{R}_i]) \cup N(u, G_k^-[\mathcal{P}_i \cup \mathcal{S}_i \cup \Psi_i])$ is exactly the set of vertices having a larger order than $\pi(\mathcal{P}_i)$ in Ψ_t. Therefore, $d_{\Psi_t}(\pi(\mathcal{P}_i), G_k^-) = \beta(\pi(\mathcal{P}_i)) \geq k$.

(2) For **condition 2**, we first assume that a vertex $w \in \Psi_t$ is appended to Ψ_i at the i-th state. Clearly, all keynodes added to Ψ_t after state i have a higher order than w. On the other hand, $\pi(\mathcal{P}_i)$ is the last keynode in Ψ_t whose order is smaller than that of w. Thus, we have:

$$\begin{aligned}&|N(w, G_k^-) \cap \{u \mid v \preceq u \wedge u \in \Psi_t\}| \geq \\ &|N(w, G_k^-) \cap \{u \mid \pi(\mathcal{P}_i) \preceq u \wedge u \in \Psi_t\}|,\end{aligned}$$

on the other hand, we only need to prove $|N(w, G_k^-) \cap \{u \mid \pi(\mathcal{P}_i) \preceq u \wedge u \in \Psi_t\}| \geq k$. Following (1), we have $\{u \mid \pi(\mathcal{P}_i) \preceq u \wedge u \in \Psi_t\} = \mathcal{U}_i \cup \mathcal{S}_i \cup \mathcal{P}_i \cup \Psi_i$. Then we can show that:

$$\begin{aligned}&|N(w, G_k^-) \cap \{u \mid \pi(\mathcal{P}_i) \preceq u \wedge u \in \Psi_t\}| \\ &= |N(w, G_k^-) \cap (\mathcal{U}_i \cup \mathcal{S}_i \cup \mathcal{P}_i \cup \Psi_i)\,| \\ &= \beta(w) \geq k,\end{aligned}$$

Thus, **condition 2** holds.

(3) For **condition 3**, consider a *cvs* $v \in \Psi_t$, suppose that in the i-th state, v is appended to Ψ_i, and there are two cases for v: (i) $v \in \mathcal{U}_i$, and (ii) $v \in \mathcal{S}_i \cup \mathcal{P}_i$. The former case is trivial. For the latter, since all vertices with infimum degree less than k are removed and added to the candidate vertex set of $\pi(\mathcal{P}_i)$ during DeleteVertex, it remains to prove that the set $\mathcal{S}_i \cup \mathcal{P}_i$ is empty after DeleteVertex finishes. We first show that all vertices in $\mathcal{S}_i$ must be removed after DeleteVertex finishes. Consider the vertex $u \in \mathcal{S}_i$ with the smallest order in $\mathcal{S}_i$. We have $\beta(u) = d_{\mathcal{O}_k}(u, G_k) < k$ because:

$$\begin{aligned}&N_{\mathcal{O}_k}\left(u, G_k^-[\mathcal{R}_i]\right) \cup N\left(u, G_k^-[\mathcal{P}_i \cup \mathcal{S}_i \cup \Psi_i]\right) \\ &= N_{\mathcal{O}_k}\left(u, G_k^-[\mathcal{R}_i \cup \mathcal{P}_i \cup \mathcal{S}_i \cup \Psi_i]\right) \\ &= d_{\mathcal{O}_k}(u, G_k^-).\end{aligned}$$

Therefore, the vertex with the smallest order in $\mathcal{S}_i$ must be removed. By induction, all vertices in $\mathcal{S}_i$ are eventually removed. Now the remaining set is reduced to the vertices in $\mathcal{P}_i$. If $i = 0$, i.e., the initial state, then all vertices in $\mathcal{P}_i$ have already been removed in CheckKeynode. Otherwise, for any vertex $u \in \mathcal{P}_i$, it must have satisfied $\beta(u) < k$ in the previous state $i-1$, which led to its inclusion in $\mathcal{P}_i$ at state i. After all vertices in $\mathcal{S}_i$ are removed via DeleteVertex, the infimum degree of u remains bound by its value in the previous state, that is, $\beta(u) < k$. Therefore, all vertices in $\mathcal{P}_i$ are eventually removed as well. □

Theorem 6.3 *The time complexity of Algorithm 3 for deleting an edge (u, v) is $O(\sum_{k=1}^{c} \mathrm{vol}(|\overline{\mathtt{diff}}_k|)\log|V_k|)$, where $c = \min\{c(u), c(v)\}$, V_k and $\overline{\mathtt{diff}}_k$ denote the set of vertices in G_k and the difference between ICD-Orders of $\mathcal{O}_k$ and $\mathcal{O}_k^-$, respectively.*

Proof. We start by analyzing the number of times entering the while loop for a given G_k. We first aim to show: $\left(\cup_{i=0}^{t}\mathcal{P}_i\right) \subseteq \overline{\mathtt{diff}}_k$.

Considering the i-th state, there are two cases for $\pi(\mathcal{P}_i)$. (1) $\pi(\mathcal{P}_i)$ is not a keynode in $\mathcal{O}_k^-$, all vertices in $(\pi(\mathcal{P}_i) \cup cvs_k[\pi(\mathcal{P}_i)])$ are appended into $\mathcal{P}_i$, and (2) $\pi(\mathcal{P}_i)$ is a keynode in $\mathcal{O}_k^-$, all vertices in $\left(cvs_k[\pi(\mathcal{P}_i)] \setminus cvs_k^-[\pi(\mathcal{P}_i)]\right)$ are appended to $\mathcal{P}_i$. In both cases, the vertices appended to $\mathcal{P}_i$ are in $\overline{\mathtt{diff}}_k$. Therefore, we have $\left(\cup_{i=0}^{t}\mathcal{P}_i\right) \subseteq \overline{\mathtt{diff}}_k$.

Then, we can conclude that the number of times entering the while loop is bounded by $O(\mathrm{vol}(\overline{\mathtt{diff}}_k))$. This is because in each iteration, $\pi(\mathcal{P}_i)$ is moved from $\mathcal{R}_i$ to either $\mathcal{P}_{i+1}$ or Ψ_i, that is, in each round, at least one of $\mathcal{P}_i$'s neighbors is removed from $\mathcal{R}_i$, which implies that the total degree:

$$\sum_{u \in \left(\cup_{j=0}^{t}\mathcal{P}_j\right)} |N(u, G_k^+[\mathcal{R}_i])|.$$

decreases by at least one in each iteration. Since this quantity is initially bounded by $O(\mathrm{vol}(\overline{\mathtt{diff}}_k))$, the total number of iterations is at most $O(\mathrm{vol}(\overline{\mathtt{diff}}_k))$.

Next, for each iteration of the while loop, the computation costs can be divided into three parts, CheckKeynode, DeleteVertex, and others. Regarding of CheckKeynode, for each state i, the time complexity is $O(\sum_{u \in \mathcal{P}_i \cup \mathcal{W}_i} |v \in N(u, G_k^-[\mathcal{S}_i \cup \{u\}])|)$. There are two main steps in CheckKeynode, first, increase the infimum degrees of all vertices in $\mathcal{P}_i$ (line 2), which takes $O(\sum_{u \in \mathcal{P}_i} |N(u, G_k^-[\mathcal{S}_i \cup \{u\}])|)$ time, since for each vertex in $\mathcal{P}_i$, we need to travse all its neighbors in $\mathcal{S}_i$ to update its infimum degree.

The second step is the while-loop (lines 4-11) in CheckKeynode, clearly, we have $|Q| = |\mathcal{W}|$. Now consider the inner for-loop (lines

6-8), there are two cases for v': (i) $v' \in \mathcal{S}$, for this case, the time complexity of the whole while-loop is $\sum_{u \in \mathcal{W}_i} |N(u, G_k^-[\mathcal{S}_i \cup \{u\}])|$;

(ii) $v' \in \mathcal{P}$, for this case, we observe that before updating its infimum degree at line 2, we have $\beta(v') < k$. This is because, in the i-th state, the vertices in $\mathcal{P}_i$ are either from ΔV or from $\mathcal{W}_{i-1}$. For vertices from ΔV, these vertices are removed from G_k, so their infimum degrees are zero. For vertices from $\mathcal{W}_{i-1}$, their infimum degree must be less than k (see line 11 in `CheckKeynode`). Now, we consider how many times a vertex $v' \in \mathcal{P}_i$ can be visited in line 6. We assume that a vertex $v' \in \mathcal{P}_i$ can be visited c times in line 6 of `CheckKeynode`, and denote $\beta(v')'$ as its infimum degree before updated in line 2. In fact, v' can only be visited if it is not removed from $\mathcal{P}_i$. Then, we have $\beta(v')' + |N(v', G_k^-[\mathcal{S}_i \cup \{v'\}])| - c \geq k-1$, and $c \leq \beta(v')' + |N(v', G_k^-[\mathcal{S}_i \cup \{v'\}])| - (k-1)$. The upper bound of $\beta(v')' = k-1$, so we can claim that $c \leq |N(v', G_k^-[\mathcal{S} \cup \{v'\}])|$. Putting them together, for each round of the while loop, Algorithm 3 takes $\sum_{u \in \mathcal{P}_i \cup \mathcal{W}_i} |N(u, G_k^-[\mathcal{S}_i \cup \{u\}])|$ time. Hence, for all states, the `CheckKeynode` takes $O(\sum_{i=1}^{t} \sum_{u \in \mathcal{P}_i \cup \mathcal{W}_i} |N(u, G_k^-[\mathcal{S}_i \cup \{u\}])|) \leq O(\mathrm{vol}(\overline{\mathtt{diff}_k}))$.

Regrading of `DeleteVertex`, it is invoked only when the type of $\pi(\mathcal{P}_i)$ changes, or when its corresponding *cvs* is not the same as its new *cvs* in the updated order. Therefore, when DeleteVertex is called at the i-th state, we have $(\mathcal{S}_i \cup \mathcal{P}_i) \subseteq \overline{\mathtt{diff}_k}$, and each vertex is traversed at most once in `DeleteVertex`. Moreover, each time a vertex is removed, we need to enumerate all its neighbors. Hence, the total time complexity of `DeleteVertex` is bounded by $O(\mathrm{vol}(\overline{\mathtt{diff}_k}) \cdot \log |V_k|)$.

Lastly, for the remaining operations in each iteration of the while-loop, we use a heap and a BST to maintain $\pi(\mathcal{P}_i)$ among all neighbors of $\mathcal{P}_i$, and move vertices in $\mathcal{U}_i$ to Ψ_i. For each iteration, the above process takes $O(\log |V_k|)$ time. Therefore, the time complexity of the Algorithm 3 is $O(\sum_{k=1}^{c} \mathrm{vol}(|\overline{\mathtt{diff}_k}|) \log |V_k|)$. □

B OMITTED ALGORITHMS

Algorithm 4 illustrates how to find all *cvs* of a vertex u within the vertex set $\mathcal{S}$ in the updated graph G'_k. Here, $\mathcal{T}$ denotes the set to which u belongs (i.e., $\mathcal{P}_i$ or $\mathcal{R}_i$) in the current state i. We use a queue Q to store all *cvs* we find. Initially, $cvs = Q = \emptyset$, and u is removed from $\mathcal{T}$ (line 1). All vertices in $\mathcal{S}$ with supremum degrees less than k are appended into Q (line 2). Afterwards, we use a while loop to find the *cvs* of u (lines 3-8). Inside the loop, we first pop a vertex v from Q (line 4). Then, we examine all neighbors of u in the subgraph $G'_k[\mathcal{S}]$. All vertices in $G'_k[\mathcal{S}]$ with supremum degrees equal to k first are added to Q, and then decrease their supremum degrees by one (lines 5-7). Next, we delete v from $\mathcal{S}$ and insert it into *cvs* (line 8). When the loop ends, we return *cvs* as the result (line 12).

Algorithm 5 aims to check whether a vertex v is still a keynode after the edge deletion during the state transition process, and returns $\mathcal{S}, \mathcal{P}, \mathcal{W}$ as mentioned in Section 6. At first, we set $\mathcal{W} = Q = \emptyset$, and increase the infimum degrees of vertices in $\mathcal{P}$ (lines 1-2). Then, all vertices in $\mathcal{S} \cup \mathcal{P}$ with infimum degrees less than k are added to Q (line 3). Afterwards, we use a while loop to update the three vertex sets (lines 4-11). Inside the loop, we first pop a vertex v from Q (line 5). Next, we examine all neighbors of u in the subgraph $G_k^-[\mathcal{S} \cup \mathcal{P}]$. All vertices with infimum degrees equal to k are first appended into Q, and then decrease their infimum degrees by one (lines 6-8). Then, we delete v from $\mathcal{S}$ or $\mathcal{P}$, and insert it into $\mathcal{W}$ (lines 9-11). When the loop ends, we return $\mathcal{S}, \mathcal{P}, \mathcal{W}$ as the result (line 12).

Algorithm 4: `DeleteVertex`$(u, \mathcal{S}, \mathcal{T}, G'_k, k)$

```
input :A vertex u, two vertex sets S and T, and an integer k
1 cvs ← ∅; Q ← ∅; remove u from T;
2 Q ← the vertices in S with supremum degrees less than k;
3 while Q ≠ ∅ do
4     v ← Q.poll() ;
5     foreach v' ∈ N(v, G'_k[S]) do
6         if Γ(v') = k then Q.add(v') ;
7         Γ(v') ← Γ(v') − 1;
8     delete v from S; append v to the end of cvs;
9 return cvs;
```

Algorithm 5: `CheckKeynode`$(\mathcal{S}, \mathcal{P}, G_k^-, k)$

```
input :Two vertex sets S, and P, a graph G_k^-, a positive integer k
1 W ← ∅; Q ← ∅;
2 increase the infimum degrees of all vertices in P;
3 Q ← the vertices in S ∪ P with infimum degrees less than k;
4 while Q ≠ ∅ do
5     v ← Q.poll() ;
6     foreach v' ∈ N(v, G_k^-[S ∪ P]) do
7         if β(v') = k then Q.add(v') ;
8         β(v') ← β(v') − 1;
9     if v ∈ S then delete v from S;
10    else delete v from P;
11    W.add(v);
12 return S, P, W;
```

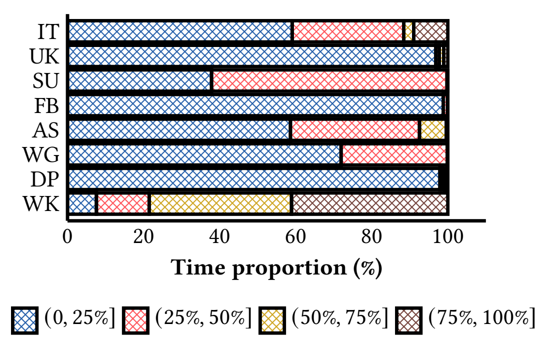
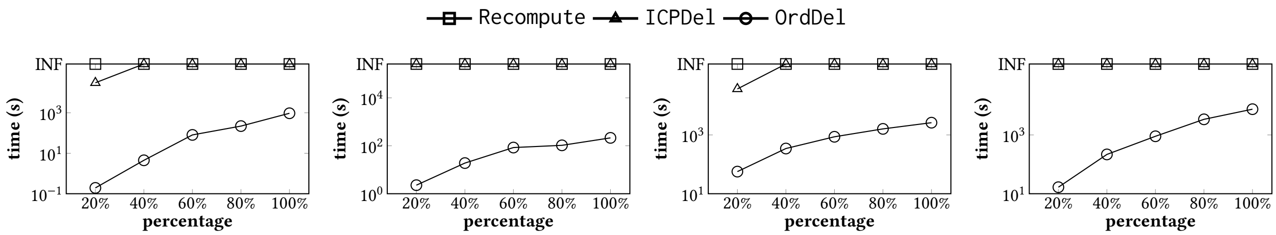


Figure 13: Proportion of the time cost of different k in `OrdDel`.

C ADDIONTIAL RESULTS ON EDGE DELETIONS

▶ **Exp.4. Scalability test.** To test the scalability, we randomly selected 20%, 40%, 60%, 80%, and 100% of edges from each graph, and then obtained five induced subgraphs by these edges. We show the results on all datasets when deleting 6,000 edges in Figure 14 since the trends are similar on all other datasets. The running time



(a) FB (b) SU (c) UK (d) IT

Figure 14: The scalability test for handling edge deletion.

of these three algorithms increases with the number of edges, our algorithm performs better than `Recompute` and `ICPDel` in all cases, and the growth rate of the curve of `OrdDel` is smaller. Therefore, our proposed edge deletion algorithm scales well on large graphs in practice.

▶ **Exp.5. Time proportion of different** k. As `OrdDel` requires maintaining ICD-Order for all $k \in [1, \delta]$, we analyze the computational overhead distribution through progressive k subdivisions. Figure 13 demonstrates the temporal heterogeneity when processing 6,000 edge deletions (post-memory initialization) across eight datasets. Notably, the algorithm exhibits distinct phase characteristics: The first quartile consumes nearly or over 60% of total computation time in six datasets. This phenomenon correlates with the scale of G_k subgraphs - smaller k values correspond to denser graph structures where edge modifications trigger cascading updates across broader vertex neighborhoods. Conversely, the last quartile demonstrates superior efficiency with time consumption below 10% in seven datasets. The contracted subgraph dimensions at higher k thresholds substantially reduce the propagation range of structural updates, resulting in localized computational adjustments.