

1741 A PROOFS OF LEMMAS AND THEOREMS

1742 THEOREM 3.6 ICPIns and ICPDel are relatively unbounded algo-
 1743 rithms.

1744 PROOF. With the ICP-Index as auxiliary information to the IC
 1745 decomposition algorithm, we define AFF as the difference between
 1746 the ICP-Index of G and $G \oplus \Delta G$. For a given k , denote \mathcal{T}_k and \mathcal{T}'_k
 1747 as the ICP-Index of G_k and $(G \oplus \Delta G)_k$ respectively. Specifically,
 1748 AFF consists of vertices and edges in two types of tree nodes in the
 1749 ICP-Index: (1) a tree node u that occurs in \mathcal{T}_k but does not occur
 1750 in \mathcal{T}'_k (or vice versa), that is, $u \in (\mathcal{T}_k \setminus \mathcal{T}'_k) \cup (\mathcal{T}'_k \setminus \mathcal{T}_k)$. (2) a tree
 1751 node u , in which u has a child node c in \mathcal{T}_k with keynote v , but v 's
 1752 corresponding tree node in \mathcal{T}'_k is not a child of u (or vice versa).

1753 The graph G in Figure 1 provides an example to illustrate the
 1754 unboundedness of ICPIns and ICPDel. When edge (v_1, v_5) is in-
 1755 serted to (resp. deleted from) G_3 , $|\text{AFF}|$ contains the vertices and
 1756 their incident edges in the tree node containing v_1 , as shown in
 1757 3. That is, $|\text{AFF}| = O(|N(v_1, G_3 \oplus \Delta G)|)$. However, as ICPIns and
 1758 ICPDel involves recomputing \mathcal{T}_3 , their time complexity on main-
 1759 taining ICS for G_3 are both $O(|E_3| + |V_3| \log |V_3|)$. Thus, we have
 1760 $M(G, \Delta G) = O(|E_3| + |V_3| \log |V_3|)$, which cannot be expressed as
 1761 a polynomial of $|\text{AFF}|$ and $|Q|$. Therefore, both ICPIns and ICPDel
 1762 are relatively unbounded. \square

1763 LEMMA 4.2 Given a graph $G_k = (V_k, E_k)$ and its ICD-order O_k , a
 1764 vertex $u \in O_k$ is a keynote of G_k , if and only if, $d_{O_k}(u, G_k) \geq k$.

1765 PROOF. We first prove that if u is a keynote 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 keynote. 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. \square

1766 THEOREM 5.2 Given a graph G , a set of edges ΔG , and G 's ICD-
 1767 order O , Algorithm 2 can compute $G \oplus \Delta G$'s ICD-order O^+ correctly.

1768 PROOF. We prove that Ψ_t qualifies the ICD-order of G_k^+ by pro-
 1769 viding that it statifies the each condition in the PROPERTY 1 sepa-
 1770 rately:

1771 (1) We prove that all keynodes in Ψ_t must statify the **condition**
 1772 1 in PROPERTY 1. It is worth noting that for each $i \in [1, t]$, **Case**
 1773 **a** and **Case b(i)** are the **only cases** that result in vertices from
 1774 $\mathcal{R}_i \cup \mathcal{P}_i$ being included in Ψ_i as key nodes. For **Case a**, p^* is a new
 1775 keynote, we have $d_{\Psi_t}(p^*, G_k^+) = \Gamma(p^*) \geq k$. For **Case b(i)**, we have
 1776 $d_{\Psi_t}(\pi(\mathcal{P}_i), G_k^+) \geq d_{O_k}(\pi(\mathcal{P}_i), G_k)$.

1777 (2) For **condition 2**, consider a keynote $w \in \Psi_t$ and for each
 1778 vertex $v \in \Psi_t$ with $v \preceq w$, there are two cases for w : (1) w is not
 1779 appended to \mathcal{P}_i for any state $i \in [1, t]$; and (2) there exists a state
 1780 $i \in [1, t]$ such that $w \in \mathcal{P}_i$. For the first case, we need to consider
 1781 two sub-cases in the vertex sequence Ψ_t based on whether $v \in \mathcal{D}_k$.

- 1782 • (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
 1783 the vertex order in Ψ_t ;
- 1784 • (ii) $v \notin \mathcal{D}_k$, v is a new keynote 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 keynote in

1799 Ψ_t . This is because the new keynote v must be either before
 1800 $\mathcal{D}_{\Psi_t}[w]$ or after w in Ψ_t .

1801 For the second case, assume that w is appended into \mathcal{P}_i at state
 1802 i and v is appended into Ψ_j at state j . Here, two sub-cases are
 1803 based on the relationship of i and j . (i) if $i > j$, **condition 2** is
 1804 clearly satisfied, and (ii) if $i < j$, consider the j -th state, vertices
 1805 are iteratively moved from the union of sets \mathcal{R}_j and \mathcal{P}_j into Ψ_j .
 1806 Now, considering the moment when v is removed from $\mathcal{R}_j \cup \mathcal{P}_j$,
 1807 we observe that

$$1808 |N(w, G_k^+) \cap \{u \mid v \preceq u \wedge u \in \Psi_t\}|$$

1809 is exactly the supremum degree of w . This is because, at that time,
 1810 v is the last vertex in Ψ_j , implying all other vertices in Ψ_j precede v
 1811 in Ψ_t . In addition, all the remaining vertices in \mathcal{R}_j or \mathcal{P}_j are succeed
 1812 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
 1813 the following equalititon:

$$1814 N(w, G_k^+) \cap \{u \mid v \prec u \wedge u \in \Psi_t\} = |N(w, G_k^+) \cap (\mathcal{R}_j \cup \mathcal{P}_j)| \\ 1815 = \Gamma(w).$$

1816 Therefore, $|N(w, G_k^+) \cap \{u \mid v \preceq u \wedge u \in O_k\}| \geq k$.

1817 (3) For **condition 3**, for a *cvs* vertex $v \in \Psi_t$, suppose that in
 1818 the i -th state, v is appended to Ψ_i , and there are also two cases
 1819 for v : (1) $v \in \mathcal{P}_i$, and (2) $v \in \mathcal{R}_i$. For the first case, clearly, we
 1820 have $d_{\Psi_t}(v, G_k^+) = \Gamma(v) < k$. For the second case, we prove this
 1821 by contraction. Assume that $\Gamma(v) \geq k > d_{O_k}(v, G_k)$, that is, there
 1822 must exist a state j ($j < i$), $v = \pi(\mathcal{P}_j)$, and $\omega(p^*) > \omega(v)$. Consider
 1823 in the j -th state, since $\Gamma(v) \geq k$, v is appended to \mathcal{P}_{j+1} (**Case b(iii)**)
 1824 and would not be in \mathcal{R}_i . Therefore, $d_{\Psi_t}(v, G_k^+) = \Gamma(v) < k$ must
 1825 hold. \square

1826 THEOREM 5.3 The time complexity of Algorithm 2 for inserting
 1827 an edge (u, v) is $O(\sum_{k=1}^{\psi} (\text{vol}(\text{diff}_k) + |N(\text{diff}_k, G_k)| \cdot \log |V_k|))$,
 1828 where $\psi = \min\{c(u), c(v)\} + 1$, V_k and diff_k denote the set of ver-
 1829 tices in G_k and the difference between ICD-orders of O_k and O_k^+ ,
 1830 respectively.

1831 PROOF. We start by analyzing the number of times entering the
 1832 while loop for a given G_k . We first aim to show: $(\bigcup_{i=0}^t \mathcal{P}_i) \subseteq \text{diff}_k$.

1833 In the i -th state, a vertex u would be added to \mathcal{P}_i , if and only if
 1834 **case Case b(iii)** occurs. In this case, u is not a keynote in G_k . Let
 1835 v be the first vertex that needs to be removed from \mathcal{P}_i after u is
 1836 added to \mathcal{P}_i . If $v \neq u$, then $u \preceq v$ and $v \preceq^+ u$, implying $u \in \text{diff}_k$.
 1837 Otherwise, $d_{O_k^+}(u, G_k^+) = \Gamma(u) \geq k$, implying $u \in \mathcal{D}_k^+$ and $u \in$
 1838 diff_k . In both cases, $u \in \text{diff}_k$, therefore $(\bigcup_{i=0}^t \mathcal{P}_i) \subseteq \text{diff}_k$.

1839 Then, we can conclude that the number of iterations of the while
 1840 loop is bounded by $O(|N(\text{diff}_k, G_k^+)|)$. This is because, in each
 1841 iteration, either $\pi(\mathcal{P}_i)$ is removed from \mathcal{R}_i , or p^* is removed from
 1842 \mathcal{P}_i . The number of iterations of the while loop does not exceed
 1843 $|\bigcup_{i=0}^t (\{\pi(\mathcal{P}_i)\} \cup \{p^*\})|$. Note that $\pi(\mathcal{P}_i)$ is a neighbor of some
 1844 vertex in \mathcal{P}_i , which implies that

$$1845 \bigcup_{i=0}^t (\{\pi(\mathcal{P}_i)\} \cup \{p^*\}) \subseteq N(\bigcup_{i=0}^t \mathcal{P}_i, G_k^+) \subseteq N(\text{diff}_k, G_k^+).$$

1846 Therefore, the total number of iterations is at most $O(|N(\text{diff}_k, G_k^+)|)$.

1847 Next, for each iteration of the while loop, the computation costs
 1848 can be divided into two parts, DeleteVertex and others. For the
 1849 second part, the time complexity of each iteration of the while loop
 1850 can be bounded by $O(\log |V_k|)$. The involved operations includes:

(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 . These operations are done by extracting and moving verticies in batches. In each iteration of the while loop, there are $O(1)$ batches, and each batch can be processed using a heap or a balanced BST in $O(\log |V_k|)$ time, with their sizes not exceeding $|V_k|$. Thus, in each iteration of the while loop, the above process takes $O(\log |V_k|)$ time, and the total time of complexity of this part is $O(|N(\text{diff}_k, G_k)| \cdot \log |V_k|)$.

The total time complexity of DeleteVertex for all iterations is bounded by $O(\text{vol}(\text{diff}_k))$, 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 OrdIns is $O\left(\sum_{k=1}^{\psi} (\text{vol}(\text{diff}_k) + |N(\text{diff}_k, G_k)| \cdot \log |V_k|)\right)$. \square

THEOREM 5.4 Given a graph G , its ICD-order 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, as discussed in Section 4, we have $|\text{AFF}| = \sum_{k=1}^{\psi} \text{vol}(\text{diff}_k) + |N(\text{diff}_k, G_k)|$ w.r.t. the ICD algorithm. Thus, for an incremental algorithm M to maintain the new ICD-order O^+ according to G and O , if the time cost of M is a polynomial of $\sum_{k=1}^{\psi} \text{vol}(\text{diff}_k) + |N(\text{diff}_k, G_k)|$, it is relatively bounded to the IC Decomposition algorithm.

Based on the definition of AFF, we know the time complexity of Algorithm 2, $O\left(\sum_{k=1}^{\psi} (\text{vol}(\text{diff}_k) + |N(\text{diff}_k, G_k)| \cdot \log |V_k|)\right)$ is polynomial of $\sum_{k=1}^{\psi} \text{vol}(\text{diff}_k) + |N(\text{diff}_k, G_k)|$. Then, according to Theorem 5.3, Algorithm 2 is a relatively bounded algorithm. \square

THEOREM 5.6 Given a graph G , a set of edges ΔG , and G 's ICD-order O , Algorithm 3 can compute $G \ominus \Delta G$'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 first show that all keynodes in Ψ_t satisfy 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 $N_{O_k}(u, G_k^-[\mathcal{R}_i]) \cup N(u, G_k^-[\mathcal{P}_i \cup \mathcal{S}_i \cup \Psi_i])$ 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 $\{u \mid \pi(\mathcal{P}_i) \preceq u \wedge u \in \Psi_t\} \subseteq \mathcal{U}_i$. 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* vertex $v \in \Psi_t$, 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 $\beta(u) = d_{O_k}(u, G_k) < k$ because:

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

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. \square

THEOREM 5.7 The time complexity of Algorithm 3 for deleting an edge (u, v) is $O\left(\sum_{k=1}^{\psi} (\text{vol}(\overline{\text{diff}}_k) + |N(\overline{\text{diff}}_k, G_k)| \cdot \log |V_k|)\right)$, V_k and $\overline{\text{diff}}_k$ 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: $(\cup_{i=0}^t \mathcal{P}_i) \subseteq \overline{\text{diff}}_k$.

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

Then, we can conclude that the number of times entering the while loop is bounded by $O(|N(\overline{\text{diff}}_k, G_k)|)$. 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:

$$\cup_{i=0}^t (\{\pi(\mathcal{P}_i)\} \cup \{p^*\}) \subseteq N(\cup_{i=0}^t \mathcal{P}_i, G_k^+) \subseteq N(\overline{\text{diff}}_k, G_k^+).$$

Therefore, the total number of iterations is at most $O(|N(\overline{\text{diff}}_k, G_k)|)$.

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

1973 main steps in CheckKeynode, first, increase the infimum degrees
 1974 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 traverse all its
 1975 neighbors in \mathcal{S}_i to update its infimum degree.
 1976

1977 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
 1978 6-8), there are two cases for $v' \in \mathcal{S}$, for this case, the time
 1979 complexity of the whole while-loop is $\sum_{u \in \mathcal{W}_i} |N(u, G_k^-[\mathcal{S}_i \cup \{u\}])|$;
 1980 (ii) $v' \in \mathcal{P}$, for this case, we observe that before updating its infimum
 1981 degree at line 2, we have $\beta(v') < k$. This is because, in the i -th
 1982 state, the vertices in \mathcal{P}_i are either from ΔV or from \mathcal{W}_{i-1} . For
 1983 vertices from ΔV , these vertices are removed from G_k , so their
 1984 infimum degrees are zero. For vertices from \mathcal{W}_{i-1} , their infimum
 1985 degree must be less than k (see line 11 in CheckKeynode). Now, we
 1986 consider how many times a vertex $v' \in \mathcal{P}_i$ can be visited in line 6.
 1987 We assume that a vertex $v' \in \mathcal{P}_i$ can be visited c times in line 6
 1988 of CheckKeynode, and denote $\beta(v')$ as its infimum degree before
 1989 updated in line 2. In fact, v' can only be visited if it is not removed
 1990 from \mathcal{P}_i . Then, we have $\beta(v') + |N(v', G_k^-[\mathcal{S}_i \cup \{v'\}])| - c \geq k - 1$,
 1991 and $c \leq \beta(v') + |N(v', G_k^-[\mathcal{S}_i \cup \{v'\}])| - (k - 1)$. The upper bound of
 1992 $\beta(v') = k - 1$, so we can claim that $c \leq |N(v', G_k^-[\mathcal{S}_i \cup \{v'\}])|$. Putting
 1993 them together, for each round of the while loop, Algorithm 3 takes
 1994 $\sum_{u \in \mathcal{P}_i \cup \mathcal{W}_i} |N(u, G_k^-[\mathcal{S}_i \cup \{u\}])|$ time. Hence, for all states, the
 1995 CheckKeynode takes $O(\sum_{i=1}^t \sum_{u \in \mathcal{P}_i \cup \mathcal{W}_i} |N(u, G_k^-[\mathcal{S}_i \cup \{u\}])|) \leq$
 1996 $O(\text{vol}(\overline{\text{diff}}_k))$.
 1997

1998 Regrading of DeleteVertex, it is invoked only when the type
 1999 of $\pi(\mathcal{P}_i)$ changes, or when its corresponding cvs vertices is not
 2000 the same as its new cvs vertices in the updated order. Therefore,
 2001 when DeleteVertex is called at the i -th state, we have $(\mathcal{S}_i \cup \mathcal{P}_i) \subseteq$
 2002 $\overline{\text{diff}}_k$, and each vertex is traversed at most once in DeleteVertex.
 2003 Moreover, each time a vertex is removed, we need to enumerate all
 2004 its neighbors. Hence, the total time complexity of DeleteVertex
 2005 is bounded by $O(\text{vol}(\overline{\text{diff}}_k))$.

2006 Lastly, for the remaining operations in each iteration of the
 2007 while-loop, we use a heap and a BST to maintain $\pi(\mathcal{P}_i)$ among all
 2008 neighbors of \mathcal{P}_i , and move vertices in \mathcal{U}_i to Ψ_i , which is similar to
 2009 the process stated in the proof of Theorem 5.3. For each iteration,
 2010 the above process takes $O(\log |\mathcal{V}_k|)$ time. Therefore, the time com-
 2011 plexity of the Algorithm 3 is $O(\sum_{k=1}^{\psi} (\text{vol}(\overline{\text{diff}}_k) + |N(\overline{\text{diff}}_k, G_k)| \cdot$
 2012 $\log |\mathcal{V}_k|))$. \square

2013

2014

2015

2016

B OMITTED ALGORITHMS

2017 Algorithm 4 illustrates how to find all cvs vertices of a vertex u
 2018 within the vertex set \mathcal{S} in the updated graph G'_k . Here, \mathcal{T} denotes
 2019 the set to which u belongs (i.e., \mathcal{P}_i or \mathcal{R}_i) in the current state i . We
 2020 use a queue Q to store all cvs vertices we find. Initially, $cvs = Q = \emptyset$,
 2021 and u is removed from \mathcal{T} (line 1). All vertices in \mathcal{S} with supremum
 2022 degrees less than k are appended into Q (line 2). Afterwards, we
 2023 use a while loop to find the cvs vertices of u (lines 3-8). Inside the
 2024 loop, we first pop a vertex v from Q (line 4). Then, we examine
 2025 all neighbors of u in the subgraph $G'_k[\mathcal{S}]$. All vertices in $G'_k[\mathcal{S}]$
 2026 with supremum degrees equal to k first are added to Q , and then
 2027 decrease their supremum degrees by one (lines 5-7). Next, we delete
 2028 v from \mathcal{S} and insert it into cvs (line 8). When the loop ends, we
 2029 return cvs as the result (line 12).

2030

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

```

input : A vertex  $u$ , two vertex sets  $\mathcal{S}$  and  $\mathcal{T}$ , and an integer  $k$ 
1  $cvs \leftarrow \emptyset; Q \leftarrow \emptyset;$  remove  $u$  from  $\mathcal{T}$ ;
2  $Q \leftarrow$  the vertices in  $\mathcal{S}$  with supremum degrees less than  $k$ ;
3 while  $Q \neq \emptyset$  do
4    $v \leftarrow Q.\text{poll}();$ 
5   foreach  $v' \in N(v, G'_k[\mathcal{S}])$  do
6     if  $\Gamma(v') = k$  then  $Q.\text{add}(v');$ 
7      $\Gamma(v') \leftarrow \Gamma(v') - 1;$ 
8   delete  $v$  from  $\mathcal{S}$ ; append  $v$  to the end of  $cvs$ ;
9 return  $cvs$ ;
```

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

```

input : Two vertex sets  $\mathcal{S}$ , and  $\mathcal{P}$ , a graph  $G_k^-$ , a positive integer  $k$ 
1  $\mathcal{W} \leftarrow \emptyset; Q \leftarrow \emptyset;$ 
2 increase the infimum degrees of all vertices in  $\mathcal{P}$ ;
3  $Q \leftarrow$  the vertices in  $\mathcal{S} \cup \mathcal{P}$  with infimum degrees less than  $k$ ;
4 while  $Q \neq \emptyset$  do
5    $v \leftarrow Q.\text{poll}();$ 
6   foreach  $v' \in N(v, G_k^-[\mathcal{S} \cup \mathcal{P}])$  do
7     if  $\beta(v') = k$  then  $Q.\text{add}(v');$ 
8      $\beta(v') \leftarrow \beta(v') - 1;$ 
9   if  $v \in \mathcal{S}$  then delete  $v$  from  $\mathcal{S}$ ;
10  else delete  $v$  from  $\mathcal{P}$ ;
11   $\mathcal{W}.\text{add}(v);$ 
12 return  $\mathcal{S}, \mathcal{P}, \mathcal{W}$ ;
```

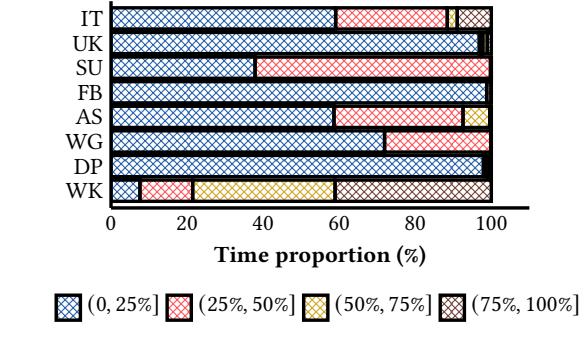


Figure 15: Proportion of the time cost of different k in OrdDel.

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 5.3. 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 v 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).

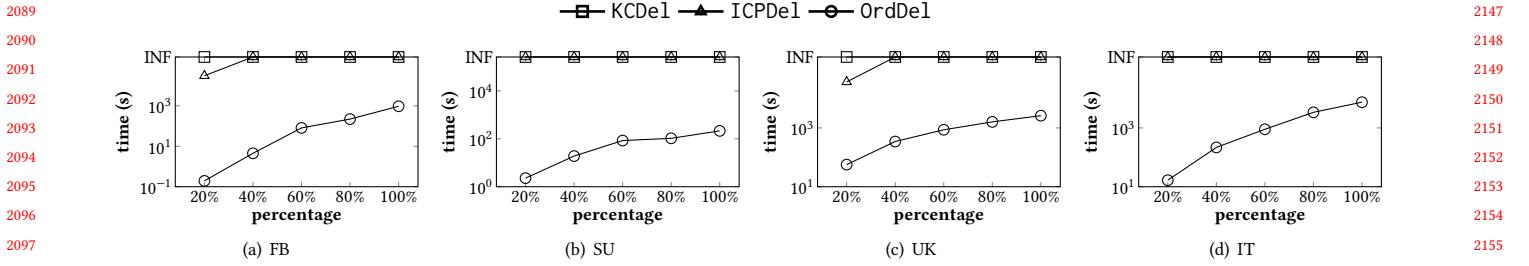


Figure 16: The scalability test for handling edge deletion.

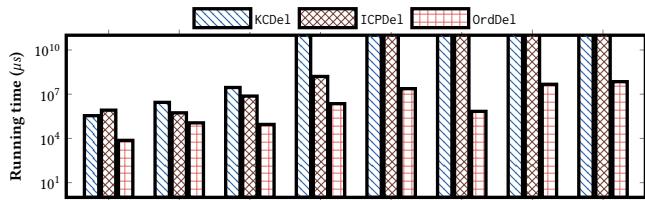


Figure 17: Efficiency of power-law-distributed edge deletions on all datasets.

C ADDITIONAL 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 16 since the trends are similar on all other datasets. The running time of these three algorithms increases with the number of edges, our algorithm performs better than Recompute and ICPDe1 in all cases, and the growth rate of the curve of OrdDe1 is smaller. Therefore, our proposed edge deletion algorithm scales well on large graphs in practice.

► **Exp.5. Time proportion of different k .** As OrdDe1 requires maintaining ICD-order for all $k \in [1, \delta]$, we analyze the computational overhead distribution through progressive k subdivisions. Figure 15 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.

► **Exp.6. Efficiency on power-law distributions.** We generate edge deletions following a power-law distribution, since many real-world graphs exhibit power-law degree characteristics. Specifically, for an edge $e = \{u, v\}$, let $P(e)$ denote the probability that e is selected for deletion. We define $P(e) \propto (d(u, G) \cdot d(v, G))^\beta$, where $d(u, G)$ represents the degree of vertex u in the current graph G , and β controls the skewness of the distribution. In our experiments, we set $\beta = 5.0$ to emphasize the preference toward high-degree

vertices. Figures 17 demonstrate the average time for each edge deletion, respectively, where the 6K edges are selected for each dataset. On these graphs, OrdDe1 still achieves speedups of up to three orders of magnitude compared to the two baselines.

Table 9: Comparison of |AFF| on different edge deletions.

| Datasets | AFF | | Time (μ s) | |
|----------|-------------|-------------|--------------------|--------------------|
| | Original | Power Law | Original | Power Law |
| WK | 2,822,270 | 9,313,491 | 1.85×10^3 | 7.38×10^3 |
| DP | 22,080,290 | 272,663,097 | 1.42×10^4 | 1.14×10^5 |
| WG | 68,643,111 | 122,452,445 | 4.04×10^4 | 8.88×10^4 |
| AS | 191,561,450 | 971,406,589 | 3.79×10^5 | 2.27×10^6 |

► **Exp.7. Efficiency on adversarial updates.** In this experiment, we explore the stability of OrdDe1 by constructing large a AFF. Intuitively, the larger the degrees of the endpoints of the inserted or deleted edge, the larger the resulting |AFF| will be. That is, by deleting edges whose endpoints have high degrees, we can maximize |AFF| with a high probability. To examine this effect, we conduct an additional experiment that compares different edge deletion strategies. Table 9 illustrates the values under two update strategies: the strategy used in Exp. 1 (random and chronological; denoted as Original) and the power-law-based updates (denoted as Power Law). For details on how edges are selected for deletion under the power-law distribution, please refer to Exp. 6. Figure 9 illustrate the average time for OrdIns to handle each update on both strategies.