A OMITTED PROOFS

Proof of Lemma 1. Firstly, it is easy to see that for any two vertices in the same connected component, their corresponding core objects are density-reachable from each other and thus belong to the same cluster. Secondly, for any two core objects in the same cluster, they must be density-reachable from each other, and thus belong to the same connected component of G_ε. Hence, the lemma holds. □

Proof of Lemma 2. Firstly, for any object, if it is a core object w.r.t. ε_1, then it is also a core object w.r.t. ε_2 that is larger than ε_1; thus, $V(G_{\varepsilon_1}) \subseteq V(G_{\varepsilon_2})$. Secondly, it is obvious that $E(G_{\varepsilon_1}) \subseteq E(G_{\varepsilon_2})$ based on the fact that $\varepsilon_1 < \varepsilon_2$. Consequently, the lemma holds. □

Proof of Lemma 3. For any ε, G_ε consists of exactly the edges of G_c with weight no larger than ε as discussed above, and also some potentially isolated vertices. Then, following the property of the minimum spanning forest in an edge-weighted graph, the connected components of the subgraph of F_c that consists of all edges of weights no larger than ε are the same as the non-singleton connected components of G_ε. Hence, the lemma follows from Lemma 1. □

Proof of Lemma 4. Let's consider an arbitrary $\varepsilon' > \varepsilon$. Obviously, o' is still a core object w.r.t. ε'. For ε', if o is still a non-core object, then o is a border object and belongs to the same cluster as o' because of $d(o, o') \le \varepsilon < \varepsilon'$. Otherwise, o becomes a core object w.r.t. ε'; then o and o' also belong to the same cluster since $d(o, o') \le \varepsilon < \varepsilon'$. □

Proof of Theorem 1. Following the discussions in Section 4.1, it is easy to see that the core object clusters are correctly obtained. We prove in the following that the border objects are correctly assigned to clusters. Note that, it is easy to see that all the assignments of border objects to clusters in Lines 19–20 are correct. Let's consider a border object o w.r.t. the query ε. Firstly, if there exists a triplet $(o, o^*, w_b(o, o^*)) \in B$ such that $w_b(o, o^*) \le \varepsilon$. Then, o^* is a core object w.r.t. ε, and o is assigned to o^*'s cluster in Lines 19–20. Secondly, if o belongs to a cluster in the density-based clustering for ε, then there must exist a core object o' (i.e., $o'.C \le \varepsilon$) from which o is directly density-reachable (i.e., $d(o, o') \le \varepsilon$); as a result, there must exist a triplet $(o, o^*, w_b(o, o^*)) \in B$ with $w_b(o, o^*) \le \varepsilon$.

Regarding the time complexity, we use a forest of parent pointer trees to represent the disjoint-set data structure [10], and the representative object in a set is the one at the root of the tree. Let $\|\mathcal{K}\|$ denote the total number of distinct vertices in the clusters of $\mathcal{K}$. Then, both the number of processed edges of F_c (by Line 3) and the number of processed triplets of B (by Line 13) are bounded by $\|\mathcal{K}\|$. By adopting both the path compression optimization and the union by rank optimization in the disjoint-set data structure, the total time complexity of Lines 3–8 is $O(\|\mathcal{K}\| \cdot \alpha(\|\mathcal{K}\|))$, where $\alpha(\cdot)$ is the inverse Ackermann function. As $\alpha(\cdot)$ is an extremely slow-growing function, we omit this term in our time complexity analysis. By adopting the path compression optimization, Lines 10–20 take time $O(\|\mathcal{K}\|)$. Thus, the time complexity follows. □

Proof of Theorem 2. Firstly, computing $o.C$ for all objects in D (i.e., Lines 1–3) takes $O(T_{nei} + m \log \mu)$ time. Specifically, Lines 1–2 take time T_{nei}. For an object $o \in D$, running Line 3 takes $O(|N(o)| \log \mu)$ time, by using a priority queue of size μ to store the potential μ closest neighbors. Thus, running Line 3 for all objects of D takes $O(m \log \mu)$ time.

Secondly, computing the minimum spanning forest F_c (i.e., Lines 5–22) takes $O(T_{nei}+m+|D| \log |D|)$ time. Specifically, running Lines 14 for all objects takes T_{nei} time, the same as Lines 1–2. In addition, the Prim's algorithm has a time complexity of $O(m + n \log n)$ for a graph with n vertices and m edges, when using the Fibonacci heap.

Thirdly, running Lines 23–26 for all objects take $O(m)$ total time.

Thus, the time complexity holds. The space complexity is obvious as we do not materialize the core graph G_c. □

Proof of Lemma 5. Recall that ε_{o_1,o_2} is the smallest ε such that o_1 and o_2 belong to the same core object cluster. Following Lemma 3, it is the smallest ε such that o_1 and o_2 are in the same connected component of the subgraph of F_c that consists of all edges of weights no larger than ε. Consequently, it is the maximum edge weight in the unique path between o_1 and o_2 in F_c. □

Proof of Lemma 6. Recall that $D_o = \{o' \in D \mid \max\{o'.C, d(o, o')\} < o.C\}$, where $o.C$ is the μ-th smallest distance among the distances between o and its neighbors $N(o)$. Hence, at most $\mu - 1$ objects o' (including o itself) can satisfy $d(o, o') < o.C$, which implies $|D_o| < \mu - 1$. □

Proof of Theorem 3. Firstly, based on the definition of T_{nei}, the total time complexity of Lines 4–6 for all objects in D is T_{nei}. Secondly, for a specific object $o \in D$, Line 7 takes $O(|D_o| \log |D_o|)$ time, while Lines 8–15 take $O(|D_o| \cdot |D_o^*|)$ time. Thus, Algorithm 3 runs in $O(T_{nei} + \sum_{o \in D} |D_o| \cdot (\log |D_o| + |D_o^*|))$ time. As $|D_o| \le \mu - 2$, we have $\sum_{o \in D} |D_o| \cdot (\log |D_o| + |D_o^*|) \le \sum_{o \in D} (\mu \log \mu + \mu \cdot |D_o^*|)$. Hence, the theorem holds. □

Proof of Lemma 7. Consider an object $o' \in D_o$ for which there exists an object o^* preceding o' in the sorted order such that $d(o', o^*) \le o^*.C$. The weight $w(o', o^*)$, as defined by Equation (4), for the edge (o', o^*) in the core graph is

$$w(o', o^*) = \max\{o'.C, o^*.C, d(o', o^*)\} = \max\{o'.C, o^*.C\}.$$

Since ε_{o',o^*} is the maximum edge weight in the unique path between o' and o^* in the minimum spanning forest F_c of the core graph (see Lemma 5), we have

$$\varepsilon_{o',o^*} \le w(o', o^*) = \max\{o'.C, o^*.C\}.$$

We now prove that $\varepsilon_{o',o^*} \le w_b(o, o') = \max\{o', C, d(o, o')\}$ by considering two cases:

- If $o^*.C \le o'.C$, then $\varepsilon_{o',o^*} \le \max\{o'.C, o^*.C\} = o'.C \le \max\{o'.C, d(o, o')\} = w_b(o, o')$.
- If $o^*.C > o'.C$, then $\varepsilon_{o',o^*} \le \max\{o', C, o^*.C\} = o^*.C \le \max\{o^*.C, d(o, o^*)\} \le \max\{o'.C, d(o, o')\} = w_b(o, o')$ where the last inequality follows from the fact that o^* precedes o' in the sorted order of D_o.

Consequently, if o^* is not pruned (from D_o) by Algorithm 3, then o' must be pruned. Otherwise, o^* has already been pruned by Algorithm 3 before checking o'. Then, there must exists another object o^{**} preceding o^* in D_o that is not pruned such that $\varepsilon_{o^*,o^{**}} \le w_b(o, o^*) = \max\{o^*.C, d(o, o^*)\}$. In this case, we also have $\varepsilon_{o^*,o^{**}} \le \max\{o'.C, d(o, o')\}$ since $\max\{o^*.C, d(o, o^*)\} \le \max\{o'.C, d(o, o')\}$.

Algorithm 4: Neighborhood-SetData(o, D)

```
// Inverted lists are built from elements to objects for D
1 Initialize an empty map M for storing the number of common
    elements between o and other objects;
2 for each element e ∈ o do
3     for each object o' s.t. e ∈ o' do
4         if o' ∉ M then Add o' to M and set M[o'] ← 0;
5         M[o'] ← M[o'] + 1;
6 N(o) ← ∅;
7 for each object o' ∈ M do
8     N(o) ← N(o) ∪ {o'};
9     d(o, o') ← 1 − M[o'] / (|o|+|o'|−M[o']);
```

As a result, we have

$$\varepsilon_{o',o^{**}} \leq \max\{\varepsilon_{o',o^*}, \varepsilon_{o^*,o^{**}}\} \leq \max\{o'.C, d(o,o')\} = w_b(o,o').$$

Consequently, o' will be pruned from D_o. □

B NEIGHBORHOOD COMPUTATION

Now, we discuss how to get the neighbors $N(o)$ of an object o, which is required at Lines 2 and 14 of Algorithm 2. This operation differs for different dataset types. We focus on set data and vector data which are used in our experiments.

Set Data. Set datasets, where each object is a set of elements, are highly prevalent in real-world applications (e.g., business process data, sales data, and user interactions), and have been used in existing studies [17, 27]. The distance between two objects is measured by the Jaccard distance. Specifically, let o and o' be sets of elements, and $o \cap o'$ and $o \cup o'$ be their intersection and union, respectively. The Jaccard distance between o and o' is defined as

$$d(o,o') = 1 - \frac{|o \cap o'|}{|o \cup o'|} \tag{8}$$

Note that, the Jaccard distance between any two objects is between 0 and 1. A distance of 1 means that the two sets do not share any common elements, and intuitively they should not be assigned to the same cluster. As a result, the neighbors of o is defined as

$$N(o) = \{o' \in D \mid d(o,o') < 1\} \tag{9}$$

The pseudocode of obtaining the neighbors of an object o and also their distances to o for set data is shown in Algorithm 4. Lines 2–3 enumerate all the objects that share at least one common element with o. Line 3 can be efficiently conducted as we have built inverted lists from element to objects, prior to the algorithm. After Lines 2–5, $\mathcal{M}[o']$ stores the number of common elements between o' and o. Thus, the Jaccard distance between o' and o can be computed as $1 - \frac{\mathcal{M}[o']}{|o|+|o'|-\mathcal{M}[o']}$, by noting that $|o \cup o'| = |o| + |o'| - |o \cap o'|$.

Let $I(e)$ denote the inverted list of element e, i.e., $I(e) = \{o' \in D \mid e \in o'\}$. The time complexity of Algorithm 4 is $\sum_{e \in o} |I(e)|$. Thus, the time complexity of Lines 1–2 of Algorithm 2, i.e., T_{nei}, is $\sum_{e \in S} |I(e)|^2$, where S is the set of all distinct elements in D.

Vector Data. Vector (i.e., multi-dimensional) datasets, where each object is a multi-dimensional data point, are also prevalent in real-world applications. The distance between two objects is measured by the Euclidean distance, as illustrated in the previous examples. In this case, the neighborhood of any object is the entire dataset, i.e., $N(o) = D, \forall o \in D$. As a result, for the time complexity in Theorem 2, $m = T_{nei} = |D|^2$, assuming a constant number of dimensions.

C KRUSKAL RECONSTRUCTION TREE

The Kruskal Reconstruction Tree (KRT) constructed for F_c is a vertex-weighted tree (or forest) with approximately twice as many vertices as F_c, and is derived from Kruskal's algorithm. Specifically, KRT is initialized by the objects/vertices of F_c without any edges. Then, it processes all edges of F_c in non-decreasing order regarding their weights. When processing an edge, say between o and o', it identifies the roots of o and o' in the current KRT, creates a new internal vertex as their parent, and assigns to it the weight $w(o,o')$ of the edge. In this way, the resulting KRT is a binary tree (or forest), where objects of F_c correspond to leaf vertices. Moreover, the maximum edge weight between any two objects in F_c is equal to the weight of their lowest common ancestor (LCA) in the KRT. LCA queries can be answered in constant time by applying an additional preprocessing step to the KRT, using an Euler tour combined with a sparse table. Please refer to [5] for more details. The overall time complexity of the construction of KRT is $O(|D| \log |D|)$.