Approaches for Avoiding RE-DBSCAN in Additional Graph Data

GwanUk Lee (20211216)

1 INTRODUCTION

Clustering is a foundational technique in network science and data mining, enabling the identification of communities and dense substructures in large graphs [4]. Among clustering algorithms, DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is widely used due to its ability to detect clusters of arbitrary shape and to distinguish noise points without requiring the number of clusters in advance [3]. However, classic DBSCAN is designed for static datasets and requires re-processing the entire dataset whenever new nodes or edges are added.

In this project, I focus on incremental, graph-based DBSCAN clustering, adapting density-based principles to edge-list representations of real-world network data. My approach is motivated by the need to efficiently update clusters in evolving graphs without re-running the full clustering algorithm when graph changes.

2 ALGORITHM

2.1 Overview

The proposed incremental clustering algorithm adapts DBSCAN principles to graph data by processing nodes one at a time and updating cluster assignments locally. The main procedure, `addPoint`, determines whether a new node should be labeled as noise, form a new cluster, join an existing cluster, or trigger the merging of clusters. The auxiliary procedure, `updateClusterID`, ensures consistency when clusters need to be merged.

2.2 updateClusterID(*mod_idx, target_idx*)

- **Cluster ID Validation:** The set of cluster IDs to be merged (`mod_idx`) is filtered to include only valid cluster IDs within the current cluster count.
- **Cluster Reassignment:** For each node in the graph, if its cluster ID is in the set of valid IDs, it is reassigned to the target cluster ID (`target_idx`). This effectively merges all specified clusters into one.
- **Cluster Count Update:** After merging, the algorithm recalculates the number of unique clusters and updates the cluster count accordingly.
- **Return Value:** The procedure returns `SUCCESS` upon completion.

2.3 addPoint(*node_ID*)

- **Node Insertion:** If the node with identifier `node_ID` does not already exist in the graph, it is added as a new node.
- **Neighborhood Query:** The algorithm retrieves the set of neighboring nodes (`clusterSeeds`) for the given node.
- **Core Point Check:** If the size of the neighborhood is less than the minimum required (`m_minPts`), the node is labeled as `NOISE` and the procedure terminates. This step adapts the DBSCAN core point criterion to the graph context.
- **Cluster Assignment:** Otherwise, the algorithm examines the cluster assignments of the neighbors:
  - If none of the neighbors belong to an existing cluster, a new cluster is created. The node and all its neighbors are assigned the new cluster ID.
  - If one or more neighbors already belong to clusters, the node and its neighbors are assigned the smallest cluster ID among them. If multiple distinct cluster IDs are present, these clusters are merged by invoking `updateClusterID`.

3 EXPERIMENTS

3.1 Setup and Motivation

The goal of this experiment is to evaluate the efficiency and similarity of incremental DBSCAN clustering on dynamic graphs, specifically to avoid the time expense of repeatedly running full DBSCAN when new data (edges) are added. I compared two main approaches:

- **Full+Partial DBSCAN:** Run partial DBSCAN on an initial subgraph, then, after new data arrives, re-run full DBSCAN on the updated graph.
- **Partial+Add Edges:** Run partial DBSCAN on an initial subgraph, then incrementally update the clustering as new edges are added, avoiding a full re-clustering.

The comparison is motivated by real-world scenarios where graph data grows over time, and computational efficiency is critical.

3.2 Experiment Design

I varied the percentage of edges used to initialize the partial cluster (~1%, 25%, 50%, 75%, ~99%) before incrementally adding the remaining edges. Experiments were conducted for both neighbor and non-neighbor update strategies, and for different `minPts` values (2, 5, 7) with $\epsilon = 1$. For each configuration, I measured:

- **Similarity (%):** The proportion of nodes for which the incremental DBSCAN result matches the full DBSCAN result.
- **Full DBSCAN time (ms):** Time to cluster the entire graph from scratch.
- **Partial DBSCAN time (ms):** Time to cluster the initial partial graph.
- **Adding edges time (ms):** Time to incrementally add the remaining edges and update clusters.

3.3 Results and Analysis

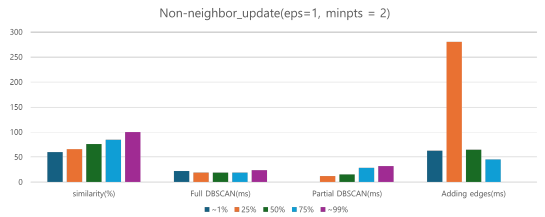

Figure 1: Non-neighbor_update ($\epsilon = 1$, minPts=5)

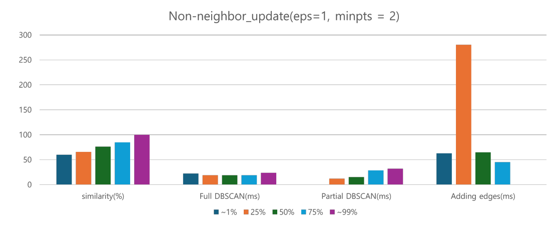

Figure 2: Non-neighbor_update (ϵ = 1, minPts=2)

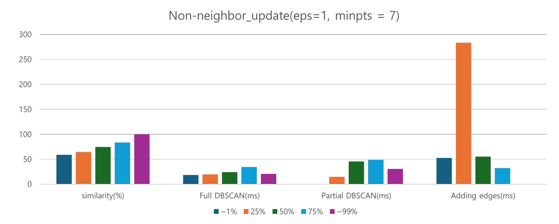

Figure 3: Non-neighbor_update (ϵ = 1, minPts=7)

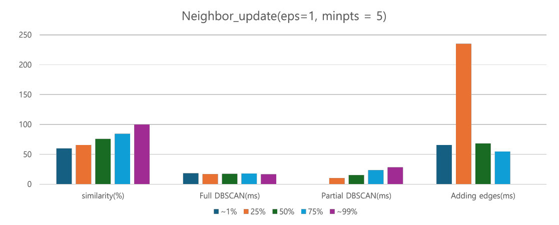

Figure 4: Neighbor_update (ϵ = 1, minPts=5)

Similarity Trends. Similarity increases steadily as the percentage of edges in the initial clustering rises. At ~1%, similarity is lowest (around 55–60%), indicating that incremental updates struggle to reconstruct the full clustering. As the initial edge percentage increases (25%, 50%, 75%), similarity improves, reaching nearly 100% at ~99%. This trend is robust across all tested `minPts` values and both update strategies.

Timing Trends. Full DBSCAN time remains stable and relatively low regardless of initialization. Partial DBSCAN time increases as more edges are included initially. The time to add edges incrementally is moderate for most percentages, but spikes dramatically at 25%, likely due to a critical change in graph connectivity or cluster structure.

Neighbor Updates. In incremental DBSCAN, a neighborhood update would update the clusters of all points in the vicinity of each newly added edge or point, in accordance with DBSCAN's density-reachability property. However, based on both the theoretical properties of DBSCAN and my experimental results, I chose **not to use the neighborhood update in the `addPoint` function** for the following reasons:

- **Time Efficiency:** Enabling neighborhood updates during incremental clustering significantly increases the computation time, as shown in the "Adding edges(ms)" bars of Figures 1–4. The time cost can spike dramatically, especially at certain update thresholds.
- **Similar similarity:** Despite the additional computation, the overall clustering similarity remains nearly unchanged whether or not neighborhood updates are used. As shown in the bar charts, the clustering result difference is negligible.
- **DBSCAN Locality:** Theoretical analysis shows that DBSCAN's clustering changes are typically localized to the immediate neighborhood of the update [1, 2]. Thus, full neighborhood updates are often unnecessary.
- **Practical Trade-off:** Given the lack of similarity improvement and the substantial time penalty, omitting neighborhood updates in `addPoint` yields much better scalability and responsiveness for dynamic graph clustering.

In summary, the neighborhood update is not used in the incremental `addPoint` function because it does not improve clustering quality but does increase computational cost.

Comparison of Approaches. The practical question is whether to re-run full DBSCAN after new data arrives or to incrementally update the clustering. Table 1 shows the measured execution times (in milliseconds) for both approaches at different initialization percentages.

Init %	Full+Partial (ms)	Partial+Add (ms)	Faster Approach
~1%	18.0	53.7	Full+Partial
25%	19.0	253.3	Full+Partial
50%	31.7	71.7	Full+Partial
75%	40.7	62.0	Full+Partial
~99%	44.7	28.0	Partial+Add

Table 1: Time cost comparison of two update strategies.

Interpretation. As shown in Table 1, the Full+Partial DBSCAN approach is faster than the Partial+Add Edges approach for all initialization percentages except ~ 99%, where Partial + Add Edges become more efficient. This result demonstrates that, contrary to initial expectations, incremental DBSCAN does not always provide time savings, particularly when a significant portion of the graph is updated (e.g. 25% or 50%). The incremental approach only outperforms full re-clustering when nearly all edges are present in the initial clustering.

Practical Implications. These results suggest a hybrid strategy: use full re-clustering for moderate-to-large updates (up to 75% of edges) and reserve incremental updates for minor changes (when more than 99% of the graph is already clustered). This ensures both computational efficiency and clustering result similarity, as the similarity of the incremental approach approaches that of the full DBSCAN only at high initialization percentages.

3.4 Results on roadNet-PA

To further evaluate the scalability and effectiveness of the incremental DBSCAN approach, I conducted experiments on the **roadNet-PA** dataset using the non-neighbor update strategy with epsilon=2 and minPts=5. The results are summarized in Figure 5 and Table 2, and support the trends observed in the p2p-Gnutella08 experiments.

- **Similarity:** Similarity increases as the initial percentage of edges increases, reaching 100% near complete initialization

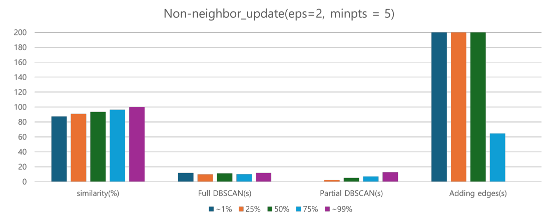

Figure 5: Incremental DBSCAN performance on roadNet-PA with non-neighbor update ($\epsilon = 2$, minPts=5).

Init %	Full+Partial (ms)	Partial+Add (ms)	Faster Approach
~1%	11,820	1,401,391	Full+Partial
25%	12,258	1,504,869	Full+Partial
50%	16,421	1,175,767	Full+Partial
75%	17,217	71,784	Full+Partial
~99%	24,515	12,726	Partial+Add

Table 2: Time cost comparison on roadNet-PA

(~ 99%). This indicates that incremental updates can reconstruct the full clustering result with high fidelity when most of the graph structure is already present.

- **Computation Time:** Full DBSCAN time remains stable. Adding edges incrementally is extremely costly when the initial subgraph is small (over 1400 seconds at 1% and 1500 seconds at 25%), but drops dramatically as the initial percentage increases, becoming negligible at 99%.
- **Trade-off:** For roadNet-PA, incremental DBSCAN is only competitive in terms of speed when the initial clustering covers the vast majority of edges ($\geq$ 75%). At lower initialization levels, the cost of incremental updates outweighs the savings from avoiding full re-clustering.

Analysis: These results reinforce the findings from the p2p-Gnutella08 experiments: incremental DBSCAN is highly accurate and efficient when the graph is nearly complete, but may incur significant overhead if too many edges are added incrementally. For large, sparse graphs like roadNet-PA, incremental updates are best suited for minor, frequent changes rather than bulk additions.

4 CONCLUSION

This project demonstrates that incremental DBSCAN clustering, where clusters are updated locally as new data are added, offers a practical and efficient alternative to full re-clustering in dynamic graph scenarios. Experimental results on real-world datasets show that the incremental approach (Partial DBSCAN + Adding Edges) is time-efficient strategy at certain update thresholds (notably ~ 99%). The findings confirm that avoiding full DBSCAN re-computation is both feasible and effective, enabling scalable clustering for applications where network data changes frequently.

Also, there remains some limitations and additional approaches such as deleting node/edge during clustering, making hybrid approaches based on my research results.

The test codes, data sets, and original DBSCAN code repositories can be checked at https://github.com/guinueng/Introduction_to_Data_Mining.git.

References

[1] Shamik Chakraborty. 2014. Analysis and Study of Incremental DBSCAN Clustering Algorithm. *arXiv preprint arXiv:1406.4754* (2014).
[2] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Michael Wimmer, and Xiaowei Xu. 1998. Incremental Clustering for Mining in a Data Warehousing Environment. In *Proceedings of the 24rd International Conference on Very Large Data Bases (VLDB).* 323–333.
[3] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD).* 226–231.
[4] Ulrike von Luxburg and Shai Ben-David. 2020. An Algorithmic Introduction to Clustering. *arXiv preprint arXiv:2006.04916* (2020).