Information Sciences 690 (2025) 121526

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

NS-IDBSCAN: An efficient incremental clustering method for geospatial data in network space

Trang T.D. Nguyen [a,b], Loan T.T. Nguyen [c,d,*], Quang-Thinh Bui [e], Le Nhat Duy [a], Bay Vo [f]

[a] Faculty of Information Technology, Industrial University of Ho Chi Minh City, Ho Chi Minh City, Viet Nam
[b] Faculty of Information Technology, Nha Trang University, Nha Trang, Viet Nam
[c] School of Computer Science and Engineering, International University, Ho Chi Minh City, Viet Nam
[d] Vietnam National University, Ho Chi Minh City, Viet Nam
[e] Faculty of Education and Basic Sciences, Tien Giang University, Tien Giang Province, Viet Nam
[f] Faculty of Information Technology, HUTECH University, Ho Chi Minh City, Viet Nam

ARTICLE INFO

Keywords:
Spatial clustering
Incremental clustering
Network space (NS)
Geographic information system (GIS)

ABSTRACT

The exponential growth of big data presents a significant task for the incremental clustering problem. This paper proposes a density-based total clustering method in network space (NS-IDBSCAN) to cluster newly generated data. Instead of re-clustering the entire massive database, which would be much larger than the new datasets, we perform clustering with only the added data. Based on the current clustering results of the old data, the proposed algorithm checks the role of each newly added point and its neighbors in performing clustering. Depending on the density, an added point can be classified as noise, border, or core. This approach dramatically reduces the response time. Moreover, using a hash table with the key as the element's index corresponding to the point Id to store the cluster Id eliminates the need for searching by direct access, further increasing processing speed. Additionally, the proposed algorithm improves accuracy by reducing the intra-cluster distance when eliminating the phenomenon that the border points may belong to more than one cluster, as presented in the discussion. The proposed algorithm is compared with the spatial data clustering algorithm in network space. Test results on three data sources downloaded from OpenStreetMap, ESRI Open Data, and Inside Airbnb demonstrate that the proposed method significantly speeds up the processing time. The accuracy of the proposed algorithm is measured using eight indicators: Silhouette, BSS, WSS, WB, Davis-Bouldin, Dunn, Calinski-Harabasz, and NMI.

1. Introduction

The most significant feature of big data is not its capacity but instead its growth rate. With the proliferation of smart mobile devices with automatic data collection capabilities, such as Internet of Things (IoT) devices, sensors, and Global Positioning Systems, the growth of data has reached an exponential rate in real time. This explosive growth has posed many challenges with regard to extracting useful information from "existing" and "upcoming" data, as knowledge needs to be obtained promptly from this increasingly

* Corresponding author at: School of Computer Science and Engineering, International University, Ho Chi Minh City, Viet Nam.
E-mail addresses: 21142301.trang@student.iuh.edu.vn (T.T.D. Nguyen), nttloan@hcmiu.edu.vn (L.T.T. Nguyen), buiquangthinh@tgu.edu.vn (Q.-T. Bui), lenhatduy@iuh.edu.vn (L.N. Duy), vd.bay@hutech.edu.vn (B. Vo).

https://doi.org/10.1016/j.ins.2024.121526
Received 24 January 2024; Received in revised form 26 September 2024; Accepted 29 September 2024

Available online 2 October 2024

massive amount of data. Some requirements must be immediately analyzed to issue timely, urgent warnings to users. For example, real time analysis of customer order data determines the logical sequence of shipping addresses in a single shipment. A warning for cars to stop immediately is necessary when a bridge is on the verge of collapse, in a state of emergency, or the use case of traffic monitoring in sensor clustering [1].

Similarly, an emergency warning is necessary for people to evacuate their homes when a fire is coming. Continuous alert streams in real time make manual analysis impossible. The demand for dynamic data and the real time response requirements of more applications pose challenges to handling incremental data.

Clustering is an active research direction in the geospatial knowledge discovery field [2]. In this, *"objects are clustered or grouped based on maximizing the intraclass similarity and minimizing the interclass similarity"* [2]. Today, clustering techniques are used not only for text clustering, image clustering [3], and uncertain graphs clustering [4], [5], but also for location clustering [6], [7], [8]. With the proliferation of mobile devices with auto-location capabilities, location data has grown exponentially. The existing clustering methods require re-clustering all the data when new data is added, which is time-consuming. Therefore, we need to find incremental clustering solutions to group new data into the previous clustering results.

In recent years, there has been explosive growth in digitalization, marked by the widespread use of sensors, Internet of Things (IoT) devices, and available GPS devices integrated into mobile phones, capable of real-time automatic positioning. As a result, geospatial data is growing in considerable volumes in real time. Consequently, it becomes imperative to regularly update newly generated data, as changes in data render old clustering results inaccurate, potentially leading to erroneous decisions. While it is possible to update clustering results by repeatedly conducting clustering on existing data, including the newly added data, this approach requires significant computational resources and storage capacity, particularly in batch data processing scenarios. In such cases, incremental clustering emerges as an essential solution. Incremental clustering leverages historical information, thus saving time in the clustering process [9].

Although the proposed incremental clustering methods solve the problem of dynamic data, some things could still be improved. Total clustering methods based on K-means cannot identify clusters of any shape and cannot detect noise because all points are clustered. The well-known typical density-based incremental clustering method [10] uses DBSCAN to update clusters by averaging the distance between the core points of each cluster and the newly added, data and then assigning new data for the cluster with minimum average distance. This method must thus determine all the core points of the old data set clustered before, wasting time. It also takes time to calculate the distance between the newly generated data and all core points of each cluster. The enhanced version of this incremental clustering method [11] increases performance by limiting the clustering space instead of doing it on the entire dataset but then having to perform a merging of dense regions.

When new data is generated for network-constraint geospatial data, it is evident that it is only necessary to calculate the distance between the newly added point and the points whose paths are within the given radius threshold. Existing incremental clustering methods still need to do this and are designed explicitly for network-constrained spatial data. Furthermore, the demand for incremental clustering of geospatial data in the network space has been highlighted in numerous critical studies and applications [12], [13], and they will be applied more effectively if analyzed in network space. For instance, in scenarios like disaster and epidemic response [13], as well as the identification of tuberculosis hotspots to optimize travel costs for patient groups for effective management and care [12], or during emergency relief operations such as rescue and recovery after disasters [13] or epidemics.

Many objects in the real world are constrained by network space, such as transportation networks, power line networks, water pipe networks, etc. The distance between them is calculated based on the shortest path length rather than the usual Euclidean distance. However, existing clustering techniques mainly operate in the planar space using Euclidean distances [6], [14]. Therefore, existing incremental clustering methods primarily also utilize Euclidean space. The approach in network space with the shortest path distance measure makes it possible to group objects with network constraints that clustering algorithms with usual Euclidean measures cannot cluster, such as grouping to improve the effectiveness of emergency responses or urban planning, distribution of goods and services, the provision care such as disaster relief, timely responses to COVID-19 patients treated at home, and disease hotspot detection, enabling logistics costs - including travel costs in each patient group - to be as low as possible for effective management and care [12]. However, until now, according to our research, there have yet to be any incremental clustering algorithms for geospatial data with network constraints. Therefore, this paper proposes a total clustering method for geospatial data based on density in network space.

Building upon an efficient clustering algorithm for spatial data in network space [7], referred to as iNS-DBSCAN for brevity (which is an improved version of NS-DBSCAN [6]), we propose a clustering solution that accelerates the processing time for newly added data, named NS-IDBSCAN. This method follows a density-based approach [15]. The proposed NS-IDBSCAN algorithm has significantly improved processing time and, in some instances, can give better clustering results. In these cases, the data points are assigned to more than one cluster by the iNS-DBSCAN [7]. According to the earlier clustered data storage solution proposed in Section 3, each point is assigned to only one cluster. Moreover, by giving points belonging to many clusters to the cluster with fewer points, NS-IDBSCAN has improved cluster quality, as proved in the Discussion. The proposed algorithm has been tested using various cluster quality evaluation metrics, such as Silhouette, BSS, WSS, WB, Davis-Bouldin, Dunn, Calinski-Harabasz, and NMI (Normalized Mutual Information).

In summary, this paper has several significant contributions to the literature, including:

(1) Introducing NS-IDBSCAN, a novel clustering algorithm that effectively incorporates new points while significantly improving execution time. Unlike previous algorithms, NS-IDBSCAN eliminates the need for a complete database rescan when new data is added to existing data. This efficiency improvement is a significant advantage of the proposed algorithm.

(2) Addressing the limitations of the iNS-DBSCAN algorithm [7], NS-IDBSCAN successfully handles the clustering of border points that belong to multiple clusters. By overcoming this challenge, NS-IDBSCAN significantly enhances the quality of the clustering results compared to existing methods. This improvement is a crucial advance in cluster analysis by addressing the clustering of border points that belong to multiple clusters, which results in improved clustering quality.
(3) Proposing a cluster selection approach designed explicitly for border points likely to belong to multiple clusters. This approach further refines the quality of the clustering outcomes. By introducing this selection method, the paper offers an additional enhancement of the overall clustering process, resulting in more accurate and reliable results.

The remainder of this paper is organized as follows: Section 2 presents an overview of related research in data clustering in network space. Section 3 presents the technical details of our proposed NS-IDBSCAN algorithm. Section 4 offers an empirical evaluation of NS-IDBSCAN, which we use to demonstrate its effectiveness. Some discussions in Section 5 about measures can taken to make the algorithm more robust. Finally, in Section 6, we conclude the paper with a discussion of potential future directions for research in this field.

2. Related work

2.1. Clustering algorithms and incremental clustering algorithms

Clustering is one of the most fundamental tasks in data analysis, aiming to group similar entities into clusters, and data clustering is widely used in a variety of fields. For instance, marketing clustering is utilized to find groups of customers with similar behavior or to identify accident black spots to improve road traffic safety [16]. In biology, clustering finds plants or animals that share common characteristics. In search engines, clustering groups similar documents to facilitate user search and topic discovery. In network analysis, clustering analyzes and classifies network traffic [11]. In the area of human behavior, clustering can be used for profiling of people to categorize their behavior [17]. Several clustering algorithms have been researched, introduced, and effectively employed. The two most common types of clustering algorithms are centroid- and density-based. Centroid-based algorithms, such as the traditional partitioning algorithm K-means and its variants (batch K-means, incremental batch K-means, online K-means, and incremental online K-means [18]), are simple clustering algorithms that identify clusters of spherical shape with a predetermined number of clusters.

However, in practice a cluster can take any shape, and the number of clusters is often unknown in advance, prompting the proposal of many efficient clustering approaches that are continuously improved. These approaches include those based on fuzzy set theory [19], [20], ANN-based approach [21], and more. In particular, the well-known density-based clustering technique for spatial data has been widely used, especially for processing large datasets with noise, as it can identify clusters of irregular shapes. DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [15], proposed by Ester et al. in 1996, was the first density-based algorithm and had many real-world applications that have been discussed in numerous works, such as [2], [22]. In addition to its advantage of not requiring the number of clusters to be known in advance, DBSCAN can identify clusters of any shape, even in noisy datasets. It is well-known for its fast computation speed, as proven in theory and practice. In fact, in 2014, its strengths were recognized with the test-of-time award at the leading data mining conference, ACM SIGKDD [22].

Inspired by the ideas of DBSCAN [15], many density-based clustering methods have been proposed to improve its limitations [23], [24], to efficiently cluster data of varying densities [25], [26], and handle large datasets [27], [28]. In 2023, the work in [29] proposed a clustering method for incomplete data based on density. Based on the density-based method, in 2014, Alex Rodriguez and Alessandro Laio introduced a clustering method with the idea that cluster centers are characterized by higher density compared to neighboring points and a relatively large distance from points with higher density, known as Density Peaks Clustering. After this, several approaches were proposed to improve the performance of this method, such as [30], [31]. Moreover, improved density-based clustering methods for geospatial data with network constraints have alse been developed [6], [7], [8].

The majority of clustering techniques perform clustering on the entire existing dataset. However, as new data is added, using existing clustering methods becomes time-consuming. Moreover, in today's busy world, where time is a big issue, static clustering algorithms are unsuitable for rapidly changing data [32]. To address this issue, incremental clustering algorithms are preferred over traditional static clustering algorithms in dynamic information environments where the amount of information rapidly increases [11]. Incremental clustering algorithms aim to cluster only the new data instead of rerunning the clustering algorithm for the entire dataset. Some of the incremental clustering algorithms that have been proposed to deal with the dynamic nature of the data are presented in the following paragraphs.

Several incremental clustering algorithms, such as those presented in [33] and [32], have been derived from the foundational K-means algorithm to accommodate changing datasets. CFBA is order-insensitive and overcomes the limitations of the K-means algorithm with many additional features, a faster execution speed, and less complexity [33]. Several enhanced versions of CFBA have also been introduced. For instance, MCFA utilizes the Manhattan method to create clusters, and ICNBCFA leverages Naïve Bayes to calculate the threshold. At the same time, CBICA substitutes the probability-based calculation in CFBA with Pearson's correlation coefficient [32].

Based on the DBSCAN technique, several incremental clustering algorithms have been researched and introduced. Among them, the density-based incremental clustering algorithm (1998) [10] is one of the most popular and cited clustering algorithms [32]. It overcomes the limitations of incremental K-means clustering and can identify noisy data and clusters of arbitrary shape. This algorithm constructs clusters from the initial objects using a threshold value of the given radius (*eps*) and the minimum number of

Table 1
A comparison of relevant incremental clustering techniques.

Study Work	Year	Algorithm	Core technology	NS	Characteristics
P. Mulay & Kulkarni [33]	2013	CFBA	Probabilistic CFBA	No	Overcoming the restrictions of K-means clustering, faster execution speed, and less complexity.
Mulay &Shinde [32]	2017	CBICA	Correlation-based Incremental Clustering	No	Enhanced versions of CFBA.
Martin Ester et al. [10]	1998	Density-based incremental clustering algorithm	Density-based	No	Can overcome the limitations of incremental K-means clustering and identify noisy data and arbitrary shape clusters.
Ahmad M. Bakr et al. [11]	2015	Incremental DBSCAN Algorithm	Density-based	No	Can limit the search space to a partition instead of traversing the entire data set.
Zhang et al. [35]	2017	ICFKM	Based on the CFS	No	Can solve two challenges: integrating new clusters into the previous cluster and updating cluster centers.
Xu et al. [9]	2019	MR-ISVM	SVM-based	No	Can reduce misclassification rates and runtime.
G. Shu-Juan [36]	2020	SBM	Spectral Clustering/Stochastic Block Model	No	Can avoid the costly computation of the eigenvectors by filtering random signals on the graph.
Chayut & Daniel [37]	2020	SOINN+	Probabilistic CFBA	No	Can detect clusters in noisy data streams.
Mai et al. [38]	2022	IncAnyDBC	Density-based	No	Can use an underlying cluster structure called an object node graph to update clustering results by parallel incremental data clustering.
Zhang et al. [39]	2023	DCS-AI-FCM	Dynamic conditional score model-based	No	Can solve the power load effective clustering problem by a proposed incremental clustering algorithm based on a statistical model.
Proposed work		NS-IDBSCAN	Density-based	Yes	Can solve the problem of rescanning the entire database to carry out clustering when new data is added for existing geospatial data.

points ($MinPts$). The resulting clusters and noise (if any) are obtained by this method. When new data is inserted, the algorithm uses DBSCAN to update the clusters by averaging the distance between the core points of each cluster and the latest data and then assigning the new data to the cluster with the minimum average distance. The new data will be classified as noises or outliers if no clusters are found. Noise points can form new clusters if they satisfy the threshold values of $Minpts$ and eps [34]. An enhanced version of the DBSCAN algorithm, proposed in 2015, improved the incremental clustering process by limiting the search space to partitions instead of performing it on the entire dataset. The algorithm's performance is significantly better than related incremental clustering algorithms, with test results showing that it is up to 3.2 times faster compared to the existing total clustering algorithms [11].

Some studies on relevant incremental clustering are compared in Table 1.

Although many incremental clustering algorithms have been developed, until now none has been designed explicitly for network-constrained geospatial data. This is because existing clustering techniques mainly operate in the planar space using Euclidean distances, which is rarely applied to network constraint events [6]. Many real-world applications must perform on network-constrained data, with shortest path distance instead of Euclidean distance. Therefore, this paper proposes an incremental clustering algorithm for geospatial data in the network space.

2.2. Algorithms for clustering geospatial data in network space

Existing spatial clustering algorithms typically employ Euclidean distance. To address this limitation, other distance metrics, such as Manhattan and Jaccard, are proposed in IncAnyDBC [38], or the standard deviation weighted distance is introduced in SFKNN-DPC [30]. However, they are rarely applied to objects with network constraints, despite many real-world phenomena being constrained by network space. Therefore, the demand arises to extend clustering algorithms to objects in network space, where objects have network constraints with path distance measurements, rather than just Euclidean space with Euclidean distance measurements. Some clustering algorithms for data with network constraints have been proposed, as outlined below.

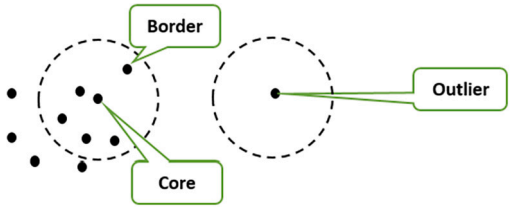

Fig. 1. Illustration of the types of density-based points, core points, border points, and outliers with $MinPts = 5$.

In 2004, Yu et al. introduced the first study on a spatial data clustering algorithm in network space [40]. This approach built upon existing clustering models and traversed the network using the Dijkstra algorithm, which is time-consuming. In 2018, a density-based clustering method, DBSCAN, was introduced for network space, utilizing NC_DT (network-constraint Delaunay triangulation) [14] to calculate network distances between points. Although this method is faster compared to the method in [40], it still requires a lot of time and memory to calculate the Delaunay triangulation. Then, in 2019, Wang et al. developed NS-DBSCAN, an extension of the DBSCAN algorithm, for spatial data clustering in network space [6]. The NS-DBSCAN algorithm significantly reduces execution time compared to the previous algorithm by determining the local shortest path to obtain density. To speed up the execution time for the NS-DBSCAN algorithm, in 2021, the work in [7] proposed an efficient spatial data clustering method in network space to enhance the efficiency of NS-DBSCAN (called iNS-DBSCAN).

The iNS-DBSCAN algorithm significantly reduces the processing time of NS-DBSCAN. However, the clustering result depends on two input parameters: the radius threshold (eps) and the minimum density threshold ($MinPts$). Defining an appropriate threshold value is inherently challenging [41], and most existing clustering algorithms become ineffective when provided with inappropriate parameters [42]. Almost all well-known clustering algorithms require input parameters that are difficult to determine but significantly influence the clustering result [32].

In 2016, the ACUTE clustering algorithm (called ACUTE_2016) [43] was introduced based on topological relationships. Apart from its ability to detect groups of various shapes and identify the noise, this method has a notable advantage in that it can reduce the number of parameters needed. The main idea of the ACUTE_2016 algorithm is to incrementally expand the data point sizes. If these points have a topological relationship that is overlaps or meets, they will be assigned to the same cluster.

ACUTE_2016 relies on fewer input parameters, but it operates in Euclidean space, which may not be suitable for data with network constraints. Inspired by this algorithm, a study [8] introduced a network space topological clustering algorithm (NS-TBC: Network Space Topological-Based Clustering) aimed at clustering objects in network space based on topology to reduce input parameters for the iNS-DBSCAN algorithm. NS-TBC enriches the research direction in topological-based clustering methods and geospatial data clustering techniques with network constraints.

In 2023, Alomari et al. [44] introduced an enhanced version of the ACUTE algorithm from 2016 [43] (referred to as ACUTE_2023), using clustering for objects with network constraints. However, this method required the computation of a distance matrix representing the shortest path between points, which is a time-consuming process.

This article's proposed incremental clustering method is developed based on the iNS-DNSCAN algorithm [7], which was (according to the best of our knowledge) the latest geospatial data clustering algorithm in network space when we began working on the project.

2.3. Some definitions

Definition 1. ([7]). Neighborhood eps of point p (eps-$neighbors$) is the set of points that are not more than a distance eps from point p. Notation: $N\_eps(p)$.

Definition 2. ([7]). The number of vertices in the set of (eps-$neighbors$) of p is defined as the density of vertex p. Notation: $Density(p) = |N\_eps(p)|$.

Definition 3. ([7]). Core point p with threshold $MinPts$ is a point with a density greater than or equal to the minimum threshold $MinPts$. Notation: p_core.

The set of its neighbor points within the distance of the eps is called the **border points**. Points of low density that are not border points are called **outliers** (or **noises**). The core points, border points, and outliers are illustrated visually in Fig. 1

Definition 4. ([15]). A point p is **directly densityreachable** from a point q with a radius threshold of eps and a density threshold of $MinPts$ if p is an $eps - neighborhood$ of q and the minimum density of q is $MinPts$, that is $p \in N\_eps(q)$ and $|N\_eps(q)| \geqslant MinPt$. (Fig. 2).

Definition 5. ([15]). If there is a chain of points $P_1, P_2, ..., P_{n-1}$ such that P_i is directly density-reachable from P_{i-1}, $i = 2..n$ then P_n is **density-reachable** from P_1 (Fig. 2).

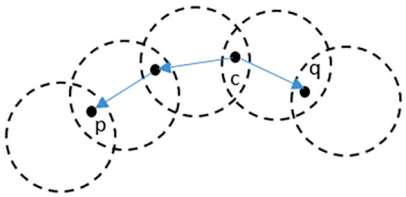

Fig. 2. Density-reachable and density-connected: q directly density-reachable from c, p density-reachable from c (c not density-reachable from p), p and q density-connected to each other by c.

It is considered that the density-reachable relation is transitive but not symmetric because the border point is density-reachable from the core point. The converse of the core being density-reachable by the border point is incorrect unless all the points are cores.

Definition 6. ([15]). If two points p and q are density-reachable from point c with thresholds eps and $MinPts$, they are considered **density-connected** with each other with the same thresholds (Fig. 2).

Definition 7. A $cluster$ with thresholds eps and $MinPts$ is a non-empty set that includes all density-reachable cores and neighbors of at least one core using the same.

All points must be connected in a cluster by $density-reachable$ or $density-connected$ relations. When a new point, x, is added to the dataset, it can cause points in the old data to change their role from $non-core$ points to core points. As a result, the dataset has two types of core points: old cores and new cores. The old cores cannot change the results of the existing clustering because they are already $density-reachable$ from their neighbors and have already been clustered. However, the new cores formed from the occurrence of x can affect the clustering results by clustering the neighboring points into the same cluster. The degree of change in the clustering results also depends on whether the new core is formed from the border or the noise. Therefore, the following defines core point types when x is added.

Definition 8. The **border-core point** p, denoted p_{bcore}, is a core point that satisfies the following two conditions before adding the point x:

1) Density(p) = MinPts - 1
2) p has been clustered (p is border).

Definition 9. The **noise-core point** p, denoted p_{ncore}, is a core point that satisfies the following two conditions before adding the point x:

1) Density(p) = MinPts - 1
2) p has not been clustered (p is a noise).

Theorem 1. *Two new types of core points are generated when adding a point x: border-core and noise-core points are neighbors of x.*

Proof. Assuming that a border-core/noise-core point is not a neighbor of x, we can infer that, due to the symmetric nature of the neighbor relationship, x is also not a neighbor of the border-core/noise-core point. Therefore, adding x will not affect the density of these points.

Since a border-core/noise-core point is a core point, before adding x, the density of these points must have been equal to $MinPts$. However, this contradicts condition 1) of the definitions of edge core and noisy core points. Thus, we can conclude that the edge core or noisy core point must be a neighbor of x.

This theorem plays an essential role in limiting the space in which points have the potential to significantly change the clustering result when a new point x is added. (Points changed their role from non-core to the core: border-core and noise-core.)

Given a weighted graph $G = (V, E, W)$, where V is the set of vertices, E is the set of edges, and W is the set of weight values. An edge between two vertices P and Q can be denoted as (P, Q). The weight of the edge (P, Q) is characterized by $W(P, Q)$. We have the following definitions:

Definition 10. [7]. The **path length** from point P_0 to point P_n passing through points $P_1, P_2, ..., P_{(n-1)} \in V$ has the value of the sum length of all path segments.

$d(P_0, P_n) = W(P_0, P_1) + W(P_1, P_2) + ... + W(P_{n-1}, P_n)$

Definition 11. [45]. The **shortest path** from point P_0 to point P_n is the path that has the minimum sum of weights among all possible paths from P_0 to P_n in the weighted graph $G = (V, E, W)$.

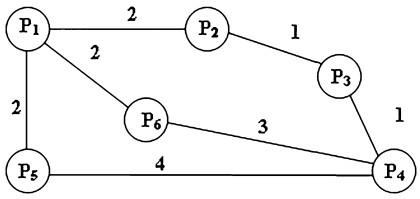

Fig. 3. Illustration of determining the shortest path between two points.

Definition 12. [46]. **Intra-cluster distance** is the distance between two objects or data points belonging to the same cluster or groups. It measures the similarity or dissimilarity between the objects within a cluster. In general, a lower intra-cluster distance indicates that the objects within a cluster are more similar to each other.

Definition 13. [46]. **Inter-cluster distance** is the distance between any two objects or data points that belong to different clusters or groups. It is a measure of the dissimilarity between different clusters. In general, a higher inter-cluster distance indicates that the objects belonging to different clusters are more dissimilar from each other.

The intra-cluster and inter-cluster distances can be used to evaluate the quality of the clustering results and to compare different clustering algorithms.

The problem of clustering geospatial data in the network space is identifying "hidden" clusters within the data. A cluster is a collection of data objects "close to each other" in the network space. The following are definitions of some concepts related to geospatial data in network space:

Definition 14. Given the set of road segments $R(road) = r_i/i = 1..|R|$ and the set of points of interest $P(point) = P_i/i = 1..|P|$ in the network space. The **distance** between two points P_1 and P_n is defined as the shortest path length between them. This distance is denoted as $dist(P_1, P_n)$ and can be calculated as follows:

$dist(P_1, P_n) = W(P_1, P_2) + W(P_2, P_3) + ... + W(P_{(n-1)}, P_n)$

Here, $W(P_i, P_{(i+1)})$ represents the weight of the edge (P_i, P_{i+1}) in the shortest path from P_1 to P_n.

Example:

The data illustrated in Fig. 3, from point P_1 to point P_4, has many paths, which are:

+ $P_1 \rightarrow P_6 \rightarrow P_4$ has a length of 5.
+ $P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4$ has a length of 4.
+ $P_1 \rightarrow P_5 \rightarrow P_4$ has a length of 6.

Then the shortest path between two points P_1 and P_4 is $P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4$ with a length of 4. Therefore, $dist(P_1, P_4) = 4$.

2.4. The iNS-DBSCAN algorithm

The proposed algorithm is built based on the iNS-DBSCAN algorithm. This algorithm, described in [7], consists of the following steps:

(1) Determining the *eps-neighbor* and density of each point with the threshold parameter radius *eps*, and building a density order table and density order graph.

(2) Using the density order graph to identify clusters of adjacent and dense points.

The *eps-neighbor* of a point p is determined by finding the points whose shortest path length to p does not exceed *eps*, using LSPD (local shortest-path distance) algorithm [7]. The details of the steps and algorithm evaluation are presented in the Appendix A.

3. The NS-IDBSCAN algorithm

When new data points are added to a dataset, a common approach is using an existing clustering algorithm to re-group the entire dataset. However, this approach can be time-consuming and inefficient since it does not use previously clustered data results. To overcome this limitation, this work proposes an incremental clustering algorithm called NS-IDBSCAN, which leverages the existing clustering results of the old data to group the newly added data.

3.1. Idea

NS-IDBSCAN is based on a density-based approach [15], [10]. In simple terms, its main idea is that the core point will "attract" its neighbors into the same group. Then, those neighbors, if they are the core, will continue to "attract" their neighbors into the same group.

Therefore, if the adding point x does not change the role of any point, then one of three simple cases (I) will occur: (1) x will be clustered with its core neighbor's cluster; (2) x will form a new cluster if it has $MinPts$ noisy neighbors, or (3) x will be considered noise.

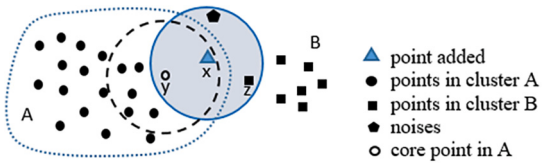

Fig. 4. The added point x has neighbor y is the old core: assign x to the same cluster with y ($MinPts = 5$).

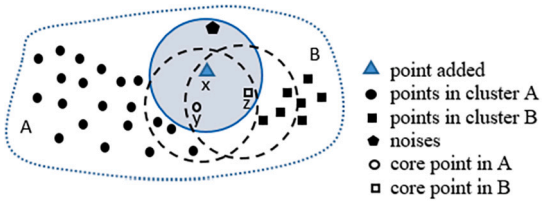

Fig. 5. The added point x is not core but has neighbors that are new cores that are not in the same group but are neighbors of each other ($MinPts = 5$): Merge clusters.

However, the points in the dataset are $density - reachable$ from the core point. Therefore, adding a new point x to the clustered dataset can change the clustering result because some points change its role from $non - core$ to core. According to Theorem 1, only points that are neighbors of x can be changed in such a way. When x is added, the points y that are "close" to x (in $N\_eps(x)$) may have their roles changed. This is because the presence of x causes $y's$ density to increase enough to reach the $MinPts$ threshold, making p becomes a new core ($border - core$ or $noise - core$) that could alter the clustering results: $Density(y) = MinPts, y \in N\_eps(x)$.

The new core point can come from either the border or the noise. If it comes from the border (border-core), it "attracts" its neighbors into the same group. It can create a new cluster if it comes from noise (noise-core). Therefore, when there is a change in the role of a point, the clustering result will also change. The change is not as simple as the three cases (I) mentioned earlier, but can be (II) as follows: (1) assigning x to the same cluster as its core neighbor (as shown in Fig. 4) and maybe merging the clusters (as shown in Fig. 6, Fig. 7); (2) creating a new cluster if x has enough $MinPts$ neighbors to be noisy; or (3) x is a noise. The detailed case of adding x with $eps = 1$ and $MinPts = 5$ is illustrated below.

- Case 1: Assigning x to an existing cluster and perhaps merging the clusters. When added point x has core neighbors.
  If $x's$ neighbor contains a core, that core will "attract" x to join the cluster. After that, the process of "attraction" and merging can continue as long as there are changes in the roles of the points. There are two possible merge cases: (1) merging through x as the core, acting as an intermediary to merge clusters, or (2) merging across borders that are neighbors of each other that have changed roles to become new cores. In this case, multiple clusters can be merged when new cores from different groups appear.
  If x has a core neighbor y that is already clustered: x is directly density-reachable from y then y will "attract" x to join the same cluster. As a result, x is assigned to the same cluster as y, as shown in Fig. 4).
  And continue to consider one of the following two merging cases (if any).
  - If x has a neighbor y that is a new core from the border: y will not only "attract" x but also all of $y's$ unclustered neighbors (i.e., noise) into the same cluster as y. Moreover, if a new core from the border z appears in the neighborhood of y that is not in the same cluster as y, then y will "attract" z and all points in the same cluster as z into the same cluster as y. This phenomenon is called merging clusters across each other's neighbors, where the border becomes the new core (i.e., the new core from the border) (Definition 8). Note that x is not the core in this case. So, if x has two neighbors that are new cores from borders and are not in the same cluster but are neighbors of each other, then they must be in the same cluster since they are density-reachable from each other. As such, we merge these two clusters. Generally, the same applies to up to $MinPts - 1$ $x's$ neighbor being a new core from the border. It must be less than $MinPts$ because if x has $MinPts$ neighbors, then x becomes the core, and all these neighbors are directly density-reachable from x. As a result, x will "attract" those points and its entire cluster into the same cluster as x as shown in Fig. 5.
  - Merging clusters through core x (Fig. 6): If the newly added point x is a core, then it will "attract" its neighbor points that do not border including core points (before adding x, it is already core), border-core point, noise-core points and noisy neighbors of x, into the same cluster as x. The merging is different for each type of core point, specifically as follows:
    * Core point: A core point and all points belonging to the same cluster with it are merged into the cluster of x.
    * Border-core/noise-core point: A border-core/noise-core point and its noisy neighbors, and all other points in its cluster are merged into the same cluster as x.

  Reasons why border points are not merged: A border point may already be part of another cluster that contains just enough $MinPts$ points. If the border is merged into the same cluster as point x, the cluster containing this border may not reach the minimum density.
  If x has no core neighbor, the following two cases remain:
- Case 2: Create a new cluster: Point x or one of its noisy neighbors becomes a core point and has $MinPts$ neighbors that have not been clustered, then a new cluster is created (Fig. 7).
- Case 3: Noise: The added point x does not generate any core points, then x is considered noise (Fig. 8).

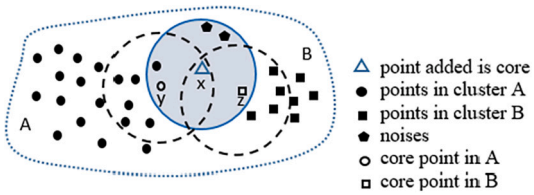

Fig. 6. Point x has neighbors as core, and x is also core with $MinPts = 5$: Merge clusters.

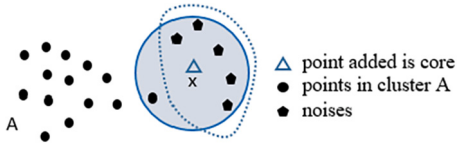

Fig. 7. Point x core whose $MinPts$ neighbors is a noise ($MinPts = 5$): Create new cluster.

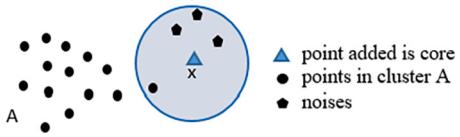

Fig. 8. Added point x does not generate core points with $MinPts = 5$): x is noise.

Table 2
Distance from each point to its nearest neighbor.

Point Id	Cluster Id
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
11	1
12	1
13	1
14	1

According to the proposed method, when a new point x appears, NS-IDBSCAN determines $x's$ neighbors. If the number of $x's$ neighbors does not reach the minimum threshold and x does not have any core neighbors, x is classified as noise. Therefore, NS-IDBSCAN can detect noise as soon as it appears, before adding it to the dataset.

3.2. Data storage solution

- Using a one-dimensional array to store the clustering results for n clustered points. The array index corresponds to the point Id, and the array value is stored in the cluster Id of the points. If a point is a noise, its value in the array is set to -1.
- The noise points must be saved because the newly added point can, together with these noise points, create a new cluster or a new core point whose neighbor is the noise point that will be clustered with that core.
- Table 2 shows an example clustering result file. Points with an Id of i belong to cluster i.

The resulting array would be [0, 0, 0, 0, 0, 0, 0, 0, 0, -1, -1, 1, 1, 1, 1]. Here, a value of 0 at index 2 indicates that the point of 2 belongs to cluster 0. A value of -1 for indices 9 and 10 indicates that the corresponding points are considered noise.

Using a one-dimensional array with the array index as the point's Id for storage eliminates the time-consuming operation of searching the entire clustered result array when determining whether a point is clustered and, if so, which cluster it belongs to. As a result, to identify the cluster to which point p belongs, access the element with an index of p directly. The value stored at this index represents the cluster Id with a value of -1, indicating that the point is a noise.

Convention: Point Ids are numbered sequentially, starting from 0.

3.3. Algorithm 1. Add one point

The proposed NS-IDBSCAN algorithm is designed to cluster newly added data more efficiently into a previously clustered dataset. Instead of having to perform clustering for the entire dataset by the existing clustering algorithm, this method only clusters for the newly added data. In this work, we build the clustering algorithm for an added point (Algorithm 1. Add one point) and call this algorithm multiple times to add a batch of multiple points. The pseudocode of the Add one point algorithm is presented in .

Algorithm 1: Add one point.

Input: Undirected planar graph, eps, $MinPts$, n points have been clustered, point x.
Output: The results are clustered $n+1$ points.

```
1  Calculate N_eps(x) by Algorithm A1 (in Appendix A)
2  foreach y ∈ N_eps(x) do
3      if y is core and y is clustered then
4          cluster(y) ← x
5          if x is core then
               /* Merge through core x                                   */
6              cluster(x) ← noise in N_eps(x)
7              merge core, borer-core and noise-core ∈ N_eps(x) and other points in its cluster into the same cluster as x
8          else if y is border-core or noise-core then
               /* merge through neighbor is new core                     */
9              cluster(y)← noise in N_eps(y)
10             merge border-core or noise-core ∈ N_eps(y) and other points in the same cluster with them to the same cluster as y
11     else if x or a neighbor of x is a core that has MinPts neighbors that are noise then
12         create a new cluster C containing it and its neighbor
13     else
14         x is a noise
```

The time complexity evaluation of the Algorithm 1 (Add one point) is described below.

Line 1 determines $x'eps-neighbors$ with a complexity of $O(nb*de)$ as evaluated in the Appendix A (Algorithm A1).

In line 3, the existence of a core point y is checked in time $O(nb*de)$. The storage solution NS-IDBSCAN uses a one-dimensional array with the array index as the point Id and the array value as the cluster Id to which the point belongs. This does not have to scan the entire array to check if a point is clustered. As mentioned above, we can directly access the element with index y, so reducing the operation time from $O(n)$ to $O(1)$.

The most time-consuming active operation is traversing the entire dataset of n points to search for points in the same cluster as the neighbors of x (also in the same cluster as y, of course) in line 7 (same for line 10). This active operation is performed in the loop $|N\_eps(x)|$ times in line 2, resulting in a runtime of $O(nb*n)$, where nb is $|N\_eps(x)|$.

Since the vertex degree (de) is too small compared to the number of vertices (n). We have $nb*de<nb*n$.

Therefore, the time complexity of Algorithm 1 (Add one point) is: $O(max(nb*de,nb*n))=O(nb*n)$.

According to the assessment in the Appendix A, the time complexity of the iNS-DBSCAN algorithm is: $O(n^2*nb^2*de)$.

Obviously: $O(nb*n)\ll O(n^2*nb^2*de)$.

So the time complexity of the Add one point algorithm is lower compared to that of the $iNS-DBSCAN$ algorithm.

Similar to the iNS-DBSCAN algorithm, the space complexity of the Add one point algorithm is $O((n+1)^2)\approx O(n^2)$ because the variable with the largest capacity in this algorithm is also the adjacency matrix containing the edge weights between n input points and point x.

The proposed NS-IDBSCAN algorithm thus improves time complexity while still preserving space complexity.

3.4. Illustration of Algorithm 1

To illustrate this algorithm, we use a test dataset consisting of 25 points labeled from P_0 to P_24 and a road network, as shown in Fig. 9.

This section illustrates the clustering process for the newly added data cases using simulation data consisting of 25 points labeled from P_0 to P_24 (with point Id from 0 to 24) and using values of $MinPts=2$ and $eps=1$.

- In case the added point$x$has a neighbor as a core point but x is not a core (do not merge clusters): Add a new point $x=15$ (Fig. 10, Fig. 11).
  - Number of points that have been clustered: 15
  - Clustering results of 15 points:
    $[0,0],[4,0],[1,0],[7,0],[2,0],[3,0],[6,0],[8,0],[5,0],[11,1],[12,1],[13,1],[14,1]$
    This result shows that two points 9, 10 are noise.
  - Convert the clustering result to an array:
    $[0,0,0,0,0,0,0,0,0,0,-1,-1,1,1,1,1,1]$

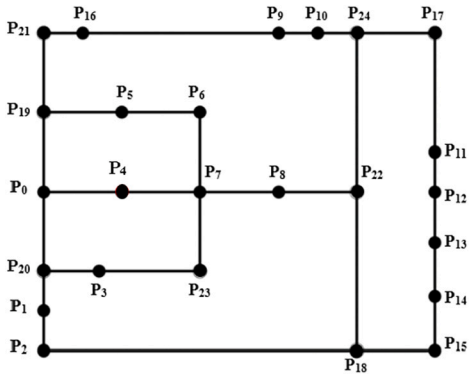

Fig. 9. The illustrative dataset (modified from [6]).

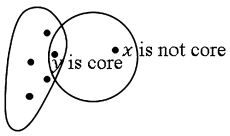

Fig. 10. The added point x is assigned to the same cluster with the core neighbor y (do not merge clusters).

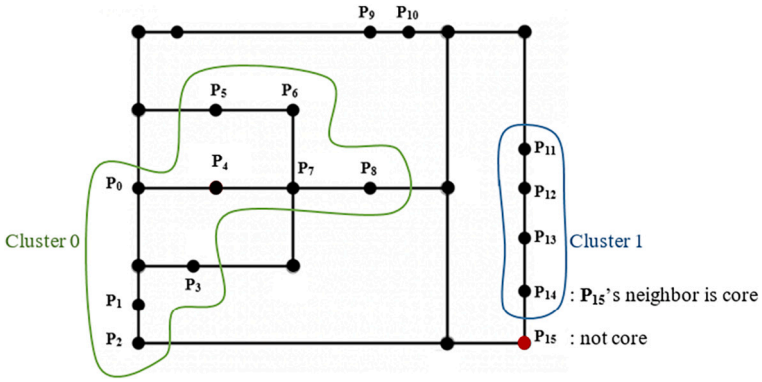

Fig. 11. Added P_{15} has the neighbor P_{14} as core: assign P_{15} to cluster 1.

- Determine the neighborhood of $x = 15$:
  $N\_eps(15) = 15, 14$
  Point $x = 15$ has one neighbor as 14.
  $\rightarrow Density(15) = 1 < MinPts$: 15 is not a core.
- Check if there are any points in the neighbors of x that are cores or not.
  * Point 14:
    $N\_eps(14) = \{14, 13, 15, 12\}$
    $Density(14) = 3 > MinPts \rightarrow 14$ is core.
- In the neighbor set of 15 that contains point 14 as the core that has been clustered into cluster 1, so we include 15 into this cluster, that is, put 15 in the same cluster with point 14 as cluster 1.
- 15 is not a core, so it is not considered to merge the group.
- Clustering results after adding new points:
  $[0, 0, 0, 0, 0, 0, 0, 0, 0, -1, -1, 1, 1, 1, 1, 1]$
  That is, point 15 is assigned to the same cluster as point 14 (Cluster 1 includes points: 11, 12, 13, 14, 15).

• In case the added point is a noise: Add a new point $x = 16$ (Fig. 12, Fig. 13).
- Number of points that have been clustered: 16
- Clustering results of 16 points in array form:
  $[0, 0, 0, 0, 0, 0, 0, 0, 0, -1, -1, 1, 1, 1, 1, 1]$
- Determine the neighbor of the added point $x = 16$:
  $N\_eps(16) = \emptyset$
  Point 16 has no neighbors, so it is a noise.

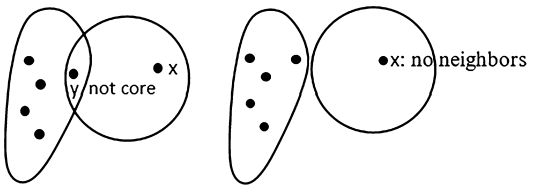

Fig. 12. The added point x is a noise.

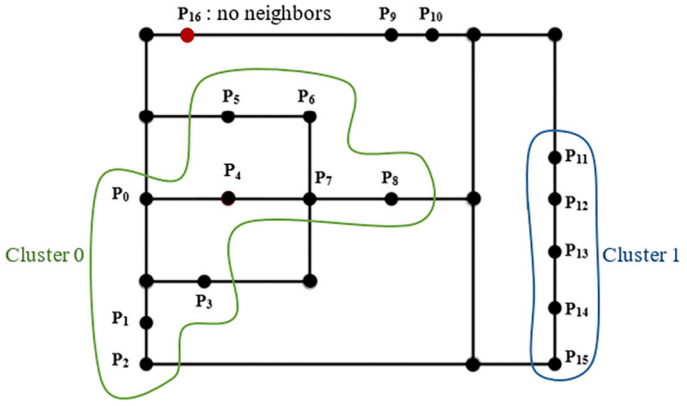

Fig. 13. P_{16} is added and has no neighbors: P_16 is a noise.

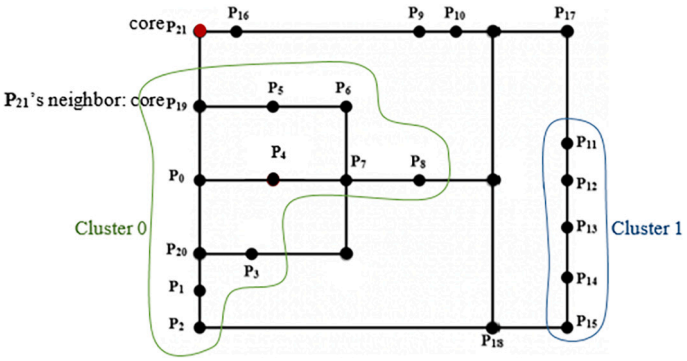

Fig. 14. The added P_21 is a core and its neighbor P_19 is also a core.

- In case the added point x has a neighbor as a core point and x is also a core (merge clusters): $x = 21$ (Fig. 14).
  - Number of points that have been clustered: 21
  - Clustering results of 21 points in array form:
    $[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, -1, 1, 1, 1, 1, 1, 1, -1, -1, -1, 0, 0]$.
  - Determine the neighbor of the added point $x = 21$:
    $N\_eps(21) = \{21, 16, 19\}$
    Point $x = 21$ has two neighbors, 16 and 19.
    $\rightarrow Density(21) = 2 (= MinPts)$: 21 is core.
  - Check if there are any points in the neighbors of x that are cores or not.
    * Point 16:
      $N\_eps(16) = \{16, 21\}$
      $Density(16) = 1 < MinPts \rightarrow 14$ is core.
    * Point 19: $N\_eps(19) = 19, 0, 5, 21$ $Density(19) = 3 > MinPts$
  - In the neighbor set of point 21 that contains point 19 as the core has been clustered into cluster 0, so we include 21 into this cluster, that is, put 21 in the same group with point 19 as group 0.
  - Point 21 is the core so all its neighbors must be in the same cluster as it, so we assign neighbor points that are not in the same cluster as this:
    Point 16 is not in the same cluster as 21, so assign 16 to cluster 0.
    Result: $[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, -1, 1, 1, 1, 1, 1, 1, 0, -1, -1, 0, 0, 0]$

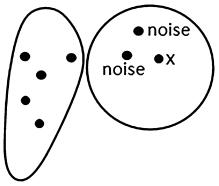

Fig. 15. Create a new cluster.

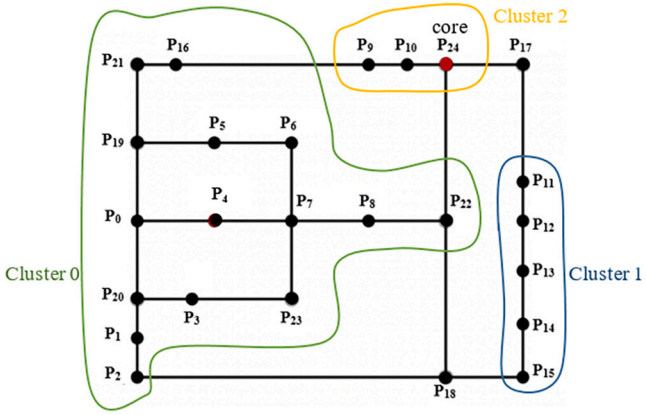

Fig. 16. The added P_{24} has two ($= MinPts$) neighbors that are noises: create a new cluster.

Table 3
Algorithm 2. Add m points.

Input	Undirected planar graph, eps, $MinPts$.
Output	The results are clustered $n + m$ points.
Method	- Read result clustered of n points to array.
	- Repeat m times
	+ Add one point algorithm (Algorithm 1).
	+ Update clustered result.

In this case, 16 is a noise, then assign 16 to the same cluster as 21. If 21 has a neighbor z that is not in the same cluster as 21, then we merge clusters: Join the cluster containing that neighbor z into the same cluster of 21 as 0.

- In case the added point has a number of neighbors that are noisy enough to form a new cluster ($\geq MinPts$) : $x = 24$ (Fig. 15 and Fig. 16).
  - Number of points that have been clustered: 24
  - Clustering results of 24 points in array form:
    $[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, -1, 1, 1, 1, 1, 1, 1, 0, -1, -1, 0, 0, 0, 0, 0]$
  - Determine the neighbor of the added point: $x = 24$.
    $N\_eps(24) = 24, 10, 9$
    Point $x = 24$ has two neighbors, 10 and 9.
    $\rightarrow Density(24) = 2(= MinPts)$.
  - Point $x = 24$ is core and 2 ($= MinPts$) neighbors (9 and 10) are noises, so create a new cluster containing 24 and these noise points.
  - Result: $[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 1, 1, 1, 1, 1, 1, 0, -1, -1, 0, 0, 0, 0, 0, 2]$

3.5. Add m points

This algorithm is used to add m new points to n points that have already been clustered. To do this, we first read the clustered results and then call Algorithm 1 (Add one point) m times. Algorithm Add m points is presented in Table 3.

According to the evaluation in Section 3.2, Algorithm 1 (Add one point algorithm) has a time complexity of $O(nb * n)$.
Because Algorithm 2 (Add m points algorithm) calls the Add one point algorithm m times, the time complexity of the Add m points algorithm is $O(m * nb * n)$.
According to the analysis in the Appendix A, the time complexity of the iNS-DBSCAN algorithm with $(n+m)$ input points is $O(n + m^2 * nb^2 * de)$.
Obviously, $O(m * nb * n) << O(n + m^2 * \mathrm{nb}^2 * de)$.

Therefore, the time complexity of the NS-IDBSCAN algorithm is much lower than that of iNS-DBSCAN.
Similar to evaluating the space complexity of the Add one point algorithm, we have the space complexity of the Add m point algorithm is $O(n + m^2)$. According to the initial assumption, the number of new points added (m) is small compared to the size of the old data (n), and we can consider m as a constant. The space complexity is thus approximately $O(n^2)$.
In short, the proposed NS-IDBSCAN algorithm enhances time complexity while still preserving space complexity in both the case of adding one point and adding batches of multiple points.

3.6. Illustration of Algorithm 2

Using the dataset consisting of 25 points in Fig. 9 to illustrate this algorithm. Suppose the old dataset has been clustered, consisting of 15 points that are P_0 to P_{14}. To add five points $P_{15}, P_{16}, P_{17}, P_{18}, P_{19}$, perform the following steps:

- Read clustered result of 15 points and convert the clustering result to an array: $arr = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, -1, 1, 1, 1, 1, 1]$.
- Repeat Algorithm 1 five times to add five points $P_{15}, P_{16}, P_{17}, P_{18}, P_{19}$ into the clustered result, as follows:
  - Implement the Add one point algorithm to add point P_{15} as illustrated step by step in Section 3.4.
  - Update result clustered to array arr: [0, 0, 0, 0, 0, 0, 0, 0, 0, -1, -1, 1, 1, 1, 1, 1].
  - Continue similarly to adding point P_{15} for each points $P_{16}, P_{17}, P_{18}, P_{19}$ and update the result clustered to the array *arr* after adding each point.

4. Experimental results

In this section, the effectiveness of the proposed algorithm NS-IDBSCAN is presented and evaluated by comparing its efficiency with the base algorithm iNS-DBSCAN [7] based on the experimental results. The two algorithms were developed in Python and ArcGIS, and executed on a computer with an Intel(R) Core(TM) CPU i5-8250U@1.60GHz (4 cores) and 12GB DDR4 SDRAM.

The Silhouette coefficient is utilized to assess the clustering quality, which measures the separation between clusters and the similarity within each cluster [2], [46]. The Silhouette of $s(x)$ ranges from -1 to +1, where a high value indicates that x is close to the points in the cluster that contains it, and distant from other clusters. Other clustering indices such as the WSS, BSS, WB-index, Davis-Bouldin index (DBI), Dunn and Calinski-Harabasz (CH) have also been used to evaluate the performance of the clustering algorithms. SSW is the sum of squares within the cluster, while BSS is the sum of squares between the clusters. A smaller value of SSW, a higher value of SSB, or a smaller value of WB indicate tighter clusters and better separation between them [47]. The lower the DB value, the better the similarity within the cluster and the higher the difference between clusters. The CH index measures how similar an object is to its own cluster compared to other clusters. So a higher value of the CH index indicates that the clusters are dense and well-separated.

The purpose of this work is to propose an incremental clustering algorithm for geospatial data in network space, NS-IDBSCAN, based on the iNS-DBSCAN algorithm aiming to speed up execution time. The experimental results in this section show that the proposed algorithm greatly accelerates processing time while still ensuring clustering results comparable to those of the base algorithm iNS-DBSCAN through seven indexes Silhouette, WSS, BSS, WB-index, DBI, Dunn and CH. In addition, this work also uses the external evaluation index NMI (Normalized Mutual Information) to evaluate the similarity in clustering results between the proposed algorithm and the baseline algorithm.

4.1. Experimental data

Experimental data were downloaded from three sources: Open Street Map (OSM), ESRI Open Data and Inside Airbnb. The first dataset was downloaded from a well-known geospatial data source, OSM, which is available at http://download.geofabrik.de/asia/vietnam.html. This study utilized the point layer of interest (POIs) and the line layer of road (roads). Fig. 17 shows the data for Binh Thuan, Dak Lak, Dak Nong, Lam Dong, and Ninh Thuan provinces in Vietnam. The data were downloaded on January 6, 2021 [8].

On March 2, 2022, the CenterFrontenac dataset[1] was obtained from ESRI Open Data, which includes the Frontenac Road Network line layer and the Civic Address Points View-SF CityView point layer. These layers represent the County of Frontenac Official Road Network and the civic address points. On February 28, 2022, the CenterBangKok dataset[2] was downloaded. It contains information on different types of lodging in Bangkok, Thailand, such as homes, apartments, private and shared rooms, and hotel rooms.

The datasets obtained from these sources are good quality and comparable to other high-quality datasets. Table 4 describes these three datasets after preprocessing.

4.2. Data preprocessing

In geospatial data sources and the real world, the point data on the road network, such as the OSM dataset, is often not accurately located on the road. For such dataset we perform preprocessing before carrying out experiments. We move these points onto the

[1] https://hub.arcgis.com/search.
[2] http://insideairbnb.com.

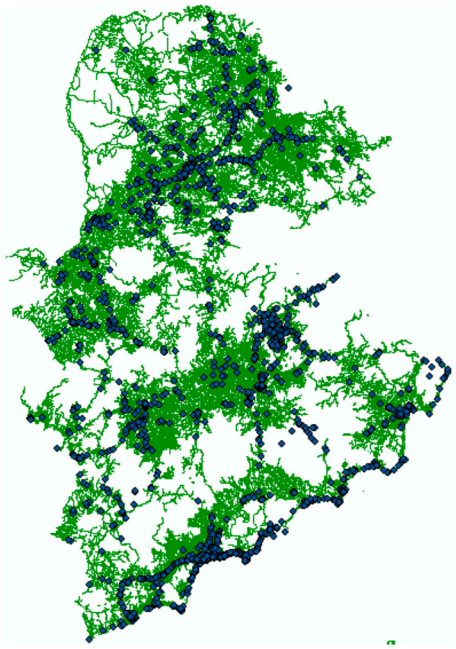

Fig. 17. The map of provinces in Vietnam downloaded from OSM.

Table 4
Illustration of the nearest neighbor of each point.

Name	Nodes	Edges
OSM	5,649	15,692
CenterFrontenac	5,430	6,156
CenterBangKok	1,785	8,719

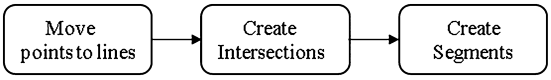

Fig. 18. Geospatial data preprocessing steps.

road to determine the path between them. After relocating the set of points not on the road onto it, we perform splitting lines at the intersections of these points and lines. Therefore, before experimenting, we preprocessed the data from the above sources. This involved extracting the area of interest, moving points not on the lines to their nearest segment, and splitting the line segments.

The details of the main steps are shown in Fig. 18

- Step 1: Move points to the lines: Move points to the nearest line (called *near_point*) to create segments in Step 3.
- Step 2: Create Intersections: Identify the intersection of the *near_point* points with the lines and also the lines with the lines (called *intersections_point*).
- Step 3: Create Segments: Split lines at the *intersections_point* points.

This paper uses mainly the spatial analysis tools in ArcMap, including “Near,” “Intersect,” and “Split line at point” to perform the above operations.

For instance, after preprocessing, the OSM map of some provinces in Vietnam downloaded from OSM in Fig. 17 is shown in Fig. 19.

The CenterFrontenac and CenterBangKok datasets contain accurately positioned points on roads, so this preprocessing is unnecessary.

4.3. Experimental results

The experiments are conducted on the same datasets for both algorithms, NS-IDBSCAN and iNS-DBSCAN [7].

The purpose of this work is to propose an incremental clustering algorithm for geospatial data in network space, NS-IDBSCAN, based on the iNS-DBSCAN algorithm aiming to speed up execution time. The experimental results in this section show that the

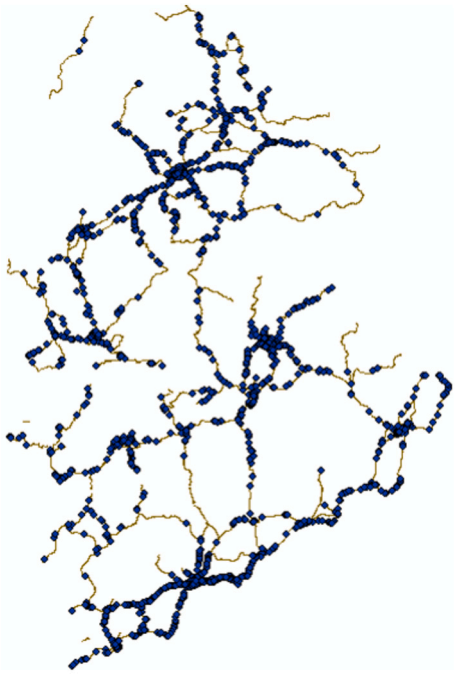

Fig. 19. The map of some provinces in Vietnam downloaded from OSM after preprocessing.

proposed algorithm greatly accelerates the processing time while still ensuring clustering results compared to the base algorithm iNS-DBSCAN. In addition to internal evaluation metrics, the evaluation results based on the external evaluation index NMI also indicate that the proposed algorithm NS-IDBSCAN produces clustering results similar to the baseline algorithm.

The results presented in Table 5 compare the quality of the output clustering results of the NS-IDBSCAN algorithm with the baseline algorithm iNS-DBSCAN on the OSM dataset when adding one point. The NS-IDBSCAN algorithm gives similar or better clustering results to the iNS-DBSCAN algorithm, as evaluated by the internal evaluation index Silhouette and the external evaluation index NMI.

The values shown in bold in the third column of Table 5 indicate that the measures in the corresponding row of the proposed NS-IDBSCAN are comparable or superior to those of the iNS-DBSCAN.

The results in Table 5a show that with 1000 clustered points and parameters $eps = 200$, $MinPts = 10$, a better Silhouette value is achieved compared to the iNS-DBSCAN algorithm. A total of 1000 points have been clustered with parameters $eps = 800$, $MinPts = 10$, the NS-IDBSCAN algorithm gives equivalent results to the iNS-DBSCAN algorithm (Table 5b). With 3000 clustered points with parameters $eps = 200$, $MinPts = 10$ and $MinPts = 20$ (Table 5c, Table 5d), the NS-IDBSCAN algorithm gives a better Silhouette value than the iNS-DBSCAN algorithm.

The NMI indicator evaluates the similarity between two output clustering results, so adding just one point has almost no effect on this index. Therefore, the value of the NMI index is 1.0 in the case of adding a new point to the dataset, which has a much larger size compared to 1, as shown in the fourth column of Table 5.

Thus, the proposed algorithm maintains or enhances the clustering results when adding a new point to the OSM dataset.

When a batch of multiple points is added, the NS-IDBSCAN algorithm not only preserves the cluster quality but also significantly speeds up the runtime as shown in the sixth column in bold in Table 6.

The test results on the OSM (DaNang) dataset of 2000 clustered points with parameters $eps = 800$, $MinPts = 20$ show that the NS-IDBSCAN algorithm provides equivalent or better clustering quality. For instance, the algorithm shows better results when adding 500 new points (the Silhouette value is $0.82330714 > 0.82314934$), and when adding 1000 new points, the Silhouette value is 0.80996587, significantly higher than 0.79603826 (column 3, Table 6). Similarly, adding 1500, 2000, 2500 and 3000 points also results in better clustering, with Silhouette values increasing from 0.73995307, 0.78568781, 0.78956240 and 0.78540792 to 0.78852850, 0.82199125, 0.82275510 and 0.81778061, respectively.

The NMI metric also show that NS-IDBSCAN has a similar cluster quality to iNS-DBSCAN (column 4, Table 6).
The cases of adding 1, 10, 50 and 100 points do not change the clustering results compared to the baseline iNS-DBSCAN algorithm, shown by the value of NMI being 1, and they are completely identical. In cases of adding 500 points or more, the NMI value is not equal to 1, meaning the grouping results are not absolutely identical. However, they are still at a high level of similarity (with NMI values ranging from 0.9 to 1). It is worth noting that the change in clustering results in these cases is better because the proposed algorithm not only improves execution time but also enhances cluster quality when the old clustered data contains points belonging to many clusters.

Table 5
Comparison of the clustering quality between NS-IDBSCAN and iNS-DBSCAN.

(a) 1000 points clustered $Eps = 200$, $MinPts = 10$.

Point Id added (1)	iNS-DBSCAN with (1) points	NS-IDBSCAN add point (1)	NMI
1000	0.799025654	0.812454691	1.0
1001	0.799025654	0.812454691	1.0
1002	0.799025654	0.812454691	1.0
1003	0.799025654	0.812454691	1.0
1004	0.799025654	0.812454691	1.0
1005	0.799025654	0.812454691	1.0
1006	0.799025654	0.812454691	1.0
1007	0.799025654	0.812454691	1.0
1008	0.799025654	0.812454691	1.0
1009	0.799025654	0.812454691	1.0
1010	0.799025654	0.812454691	1.0
1011	0.799025654	0.812454691	1.0
1012	0.799025654	0.812454691	1.0
1013	0.799025654	0.812454691	1.0
1014	0.799025654	0.812454691	1.0
1015	0.799025654	0.812454691	1.0
1016	0.799025654	0.812454691	1.0
1017	0.799025654	0.812454691	1.0
1018	0.799025654	0.812454691	1.0
1019	0.799025654	0.812454691	1.0

(b) 1000 points clustered $Eps = 800$, $MinPts = 10$.

Point Id added (1)	iNS-DBSCAN with (1) points	NS-IDBSCAN add point (1)	NMI
1000	0.835423414	0.835423414	1.0
1001	0.835423414	0.835423414	1.0
1002	0.835423414	0.835423414	1.0
1003	0.835423414	0.835423414	1.0
1004	0.835423414	0.835423414	1.0
1005	0.835423414	0.835423414	1.0
1006	0.835423414	0.835423414	1.0
1007	0.835423414	0.835423414	1.0
1008	0.835423414	0.835423414	1.0
1009	0.835423414	0.835423414	1.0
1010	0.835423414	0.835423414	1.0
1011	0.835423414	0.835423414	1.0
1012	0.835423414	0.835423414	1.0
1013	0.835423414	0.835423414	1.0
1014	0.835423414	0.835423414	1.0
1015	0.835423414	0.835423414	1.0
1016	0.835423414	0.835423414	1.0
1017	0.835423414	0.835423414	1.0
1018	0.835423414	0.835423414	1.0
1019	0.835423414	0.835423414	1.0

(c) 3000 points clustered $Eps = 200$, $MinPts = 10$.

Point Id added (1)	iNS-DBSCAN with (1) points	NS-IDBSCAN add point (1)	NMI
3000	0.739093410	0.763701240	1.0
3001	0.739110410	0.763718240	1.0
3002	0.739125676	0.763733507	1.0
3003	0.739142099	0.763749929	1.0
3004	0.739142099	0.763749929	1.0
3005	0.739142099	0.763749929	1.0
3006	0.739142099	0.763749929	1.0
3007	0.739142099	0.763749929	1.0
3008	0.739142099	0.763749929	1.0
3009	0.738993374	0.763662511	1.0
3010	0.738679856	0.763423933	1.0
3011	0.738584523	0.763328601	1.0
3012	0.738584523	0.763328601	1.0
3013	0.738584523	0.763328601	1.0
3014	0.738622207	0.763365620	1.0
3015	0.738622207	0.763365620	1.0
3016	0.738668449	0.763411259	1.0
3017	0.738664454	0.763407264	1.0
3018	0.738664454	0.763407264	1.0
3019	0.738697570	0.763440380	1.0

Table 5 (*continued*)

(d) 3000 points clustered $Eps = 800$, $MinPts = 20$.

Point Id added (1)	iNS-DBSCAN with (1) points	NS-IDBSCAN add point (1)	NMI
3000	0.796069685	0.796557386	1.0
3001	0.796090857	0.796578557	1.0
3002	0.796108215	0.796595916	1.0
3003	0.796129562	0.796617263	1.0
3004	0.796129562	0.796617263	1.0
3005	0.796129562	0.796617263	1.0
3006	0.796129562	0.796617263	1.0
3007	0.796129562	0.796617263	1.0
3008	0.796129562	0.796617263	1.0
3009	0.795903706	0.796391406	1.0
3010	0.795381049	0.795868750	1.0
3011	0.795169941	0.795657866	1.0
3012	0.795169941	0.795657866	1.0
3013	0.795169941	0.795657866	1.0
3014	0.795172203	0.795657866	1.0
3015	0.795172203	0.795660128	1.0
3016	0.795174024	0.795661949	1.0
3017	0.795101600	0.795646235	1.0
3018	0.795101600	0.795589525	1.0
3019	0.795159525	0.795647654	1.0

Table 6
Comparison of the quality and clustering time of the NS-IDBSCAN and iNS-DBSCAN algorithms based on the number of newly added points in the OSM dataset.

No points added	Silhouette (↑)		NMI	Runtime (seconds)	
	iNS-DBSCAN	NS-IDBSCAN		iNS-DBSCAN	NS-IDBSCAN
1	0.80317254	0.80317254	1.0	207.1815045	**3.90629101**
10	0.80389774	0.80389774	1.0	202.2329257	**5.52249885**
50	0.80460328	0.80460328	1.0	214.1958005	**9.02689433**
100	0.80468155	0.80468155	1.0	212.0919223	**13.96714735**
500	0.82314934	**0.82330714**	0.984904018	241.9440634	**62.35520244**
1000	0.79603826	**0.80996587**	0.969726844	286.5911825	**119.8200779**
1500	0.73995307	**0.78852850**	0.944498524	315.5306916	**163.9314392**
2000	0.78568781	**0.82199125**	0.956690833	336.2904525	**216.7360053**
2500	0.78956240	**0.82275510**	0.964881326	364.0885999	**251.6463432**
3000	0.78540792	**0.81778061**	0.973468215	392.3537817	**293.4305968**

Table 7
Comparison of the average values of clustering quality evaluation indices between algorithms iNS-DBSCAN and NS-IDBSCAN in the OSM dataset.

Algorithm	Sil (↑)	BSS (↑)	WSS (↓)	WB(↓)	DB(↓)	Dunn(↑)	CH(↑)
iNS-DBSCAN	0.793615	2.147E+09	1173.5338	6.15E-06	17.31202648	1.31E-05	5.86E+08
NS-IDBSCAN	**0.810068**	2.147E+09	**1173.2142**	**5.92E-06**	**14.85533858**	1.31E-05	**6.01E+08**

The arrow signs ↑ in Table 6 mean that the increased value of the Silhouette index in that column gives better cluster quality. Similarly, in Table 7, 8, 9, and 10, 11 the signs ↑/↓ mean that the value of the index in the column with that arrow increases/decreases for a better assessment of the resulting cluster quality.

In addition, Table 7 shows that the BSS, WSS, WB, DB, Dunn, and CH indices produce comparable results.

In terms of execution time, the experimental results, highlighted in bold in column 6 of Table 6, show that the proposed NS-IDBSCAN algorithm significantly reduces the processing time. Specifically, when one new data point is added, the clustering time of the proposed algorithm is only 2% of the original time.

Moreover, when adding a batch of multiple points, the execution time is also significantly reduced. Specifically, the proposed algorithm's clustering time is only 3%, 4%, 7%, 26% and 42% of the original time when adding 10 points, 50 points, 100 points, 500 points and 1000 points, respectively. Even when adding batches larger than the clustered data size of 2000 points, such as 2000, 2500 and 3000 new points, the processing time is still reduced to 64%, 75% and 75% of the original time, respectively.

The illustrated results demonstrate that the time taken to add n points is significantly smaller than the time to add one point multiplied by n. For instance, adding 1000 points takes 119.82007790 seconds, much less than adding one point in 3.90629101 seconds multiplied by 1000, which equals 3906.29101 seconds. This result is because when adding one point we have to read the existing clustering result from the file. However, when adding 1000 points, we read the file only once, not 1000 times.

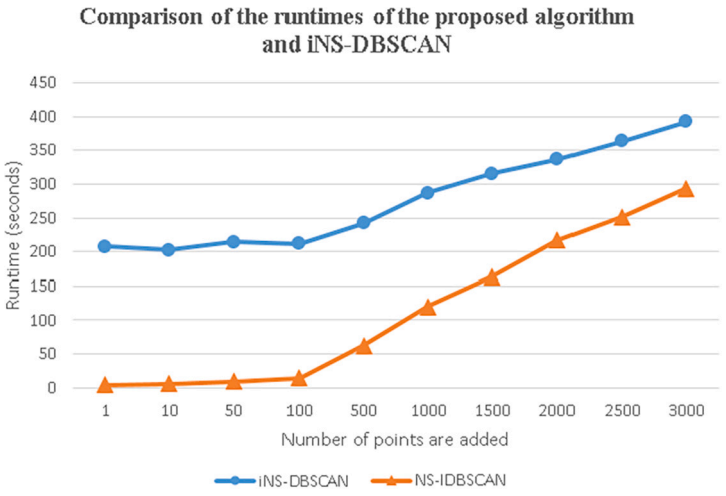

Fig. 20. Comparison of the runtimes of the proposed algorithm and iNS-DBSCAN.

Table 8
Comparison of the quality and runtime between the NS-IDBSCAN and iNS-DBSCAN based on the number of clustered points in the OSM dataset.

No. points clustered	Algorithm	Sil (↑)	BSS (↑)	WSS (↓)	WB(↓)	DB(↓)	Dunn(↑)	CH(↑)	NMI	Runtime (seconds)
100	iNS-DBSCAN	9.97E-01	2.15E+09	5.48E+02	5.10E-07	1.01E-02	8.60E+01	5.80E+08	1.0	25.88131
	NS-IDBSCAN	9.97E-01	2.15E+09	5.48E+02	5.10E-07	1.01E-02	8.60E+01	5.80E+08		37.67542
500	iNS-DBSCAN	9.97E-01	2.15E+09	5.48E+02	5.10E-07	1.01E-02	8.60E+01	5.80E+08	1.0	53.65607
	NS-IDBSCAN	9.97E-01	2.15E+09	5.48E+02	5.10E-07	1.01E-02	8.60E+01	5.80E+08		18.08395
1000	iNS-DBSCAN	8.37E-01	2.15E+09	5.86E+02	1.64E-06	1.64E+00	3.71E-06	7.65E+08	0.964479784	124.44676
	NS-IDBSCAN	8.38E-01	2.15E+09	5.82E+02	1.63E-06	1.57E+00	4.39E-06	7.71E+08		16.51371
1500	iNS-DBSCAN	8.21E-01	2.15E+09	7.60E+02	2.83E-06	7.79E+00	2.03E-06	6.22E+08	0.98991263	165.0228
	NS-IDBSCAN	8.26E-01	2.15E+09	7.47E+02	2.78E-06	7.57E+00	2.03E-06	6.33E+08		10.00372
2000	iNS-DBSCAN	8.05E-01	2.15E+09	1.09E+03	3.56E-06	3.73E+00	2.41E-06	6.70E+08	1.0	213.71785
	NS-IDBSCAN	8.05E-01	2.15E+09	1.09E+03	3.56E-06	3.73E+00	2.41E-06	6.70E+08		9.45461
2500	iNS-DBSCAN	8.15E-01	2.15E+09	1.16E+03	4.32E-06	4.39E+00	3.59E-05	6.72E+08	0.998689809	259.72903
	NS-IDBSCAN	8.17E-01	2.15E+09	1.16E+03	4.32E-06	4.39E+00	3.59E-05	6.72E+08		11.34687
3000	iNS-DBSCAN	7.46E-01	2.15E+09	1.21E+03	6.22E-06	2.26E+01	2.28E-06	5.55E+08	0.988193332	297.51241
	NS-IDBSCAN	7.48E-01	2.15E+09	1.21E+03	6.21E-06	2.26E+01	2.28E-06	5.56E+08		9.2053
3500	iNS-DBSCAN	7.58E-01	2.15E+09	1.30E+03	7.87E-06	3.36E+01	3.01E-06	4.87E+08	0.997938302	311.21835
	NS-IDBSCAN	7.61E-01	2.15E+09	1.30E+03	7.84E-06	3.33E+01	1.66E-05	4.89E+08		10.16967
4000	iNS-DBSCAN	7.82E-01	2.15E+09	1.31E+03	9.16E-06	3.23E+01	4.61E-05	4.72E+08	0.997920533	346.44113
	NS-IDBSCAN	7.88E-01	2.15E+09	1.31E+03	9.14E-06	3.17E+01	4.61E-05	4.73E+08		6.63922
4500	iNS-DBSCAN	7.90E-01	2.15E+09	1.32E+03	9.86E-06	3.26E+01	4.64E-05	4.91E+08	1.0	369.99048
	NS-IDBSCAN	7.92E-01	2.15E+09	1.32E+03	9.83E-06	3.23E+01	4.64E-05	4.92E+08		8.7819
5000	iNS-DBSCAN	7.85E-01	2.15E+09	1.32E+03	1.11E-05	3.84E+01	1.10E-05	4.82E+08	0.998475529	405.27895
	NS-IDBSCAN	7.90E-01	2.15E+09	1.31E+03	1.10E-05	3.83E+01	1.10E-05	4.83E+08		7.83486

Fig. 20 visually depicts the experimental results comparing the processing time of the proposed NS-IDBSCAN algorithm with that of the iNS-DBSCAN algorithm, according to the number of added points. As the size of the newly added data batch increases, the processing timelines of both algorithms tend to get closer. It is obvious that when the amount of newly added data is too large for the size of the clustered data, the existing clustering algorithm should be used to aggregate all the data, which is consistent with the assumption made in the Introduction section that the clustered data is much larger than the newly added data. However, the experimental data in Table 6 shows that the proposed algorithm is still effective when adding a new batch of data larger than the existing dataset, as assessed above.

It is encouraging that the rate of acceleration increases along with the size of the clustered dataset, as detailed in Table 8 and visually depicted in Fig. 21. Moreover, the proposed algorithm maintains the quality of the clusters, as demonstrated by eight cluster quality assessment indicators.

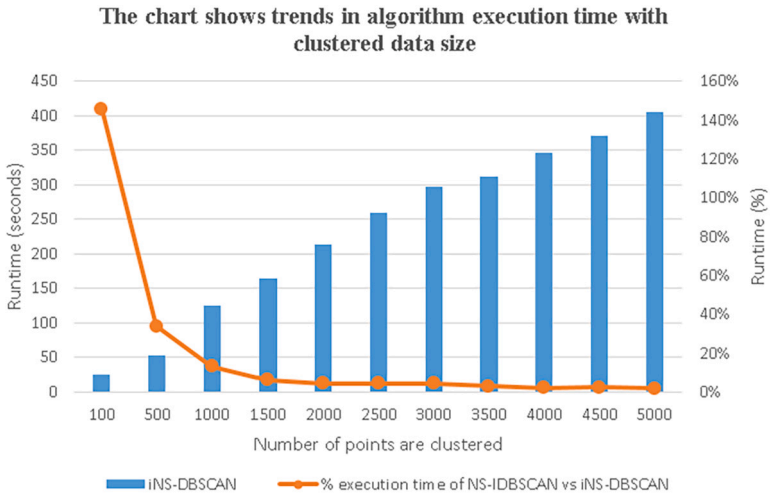

Fig. 21. The ratio of the NS-IDBSCAN proposed algorithm's execution time compared to that of the iNS-DBSCAN decreases with the increase of clustered points.

Table 9
The results of comparing the clustering quality indicators and runtime of algorithms iNS-DBSCAN and NS-IDBSCAN for data sources from ESRI Open Data and Inside Airbnb.

Datasets	Algorithms	Sil ($\bar{s}$)	BSS	WSS	WB	DB	Dunn	CH	NMI	Runtime (seconds)
Center Frontenac (5000, 50, 800, 20)(*)	iNS-DBSCAN	7.56E-01	2.96E+07	1.80E+03	2.56E-03	3.34E+01	1.31E-01	2.00E+06	0.9995	311.4
	NS-IDBSCAN	7.60E-01	2.98E+07	1.80E+03	2.54E-03	3.31E+01	1.31E-01	2.02E+06		10.5
Center Bangkok (1500, 50, 1000, 5)(*)	iNS-DBSCAN	7.56E-01	2.96E+07	1.80E+03	2.56E-03	3.34E+01	1.31E-01	2.00E+06	0.9988	101.8
	NS-IDBSCAN	7.60E-01	2.98E+07	1.80E+03	2.54E-03	3.31E+01	1.31E-01	2.02E+06		6.0

(*) is corresponding to: Number of points clustered, number of points added, radius threshold, and density threshold.

In terms of execution time, the experimental results show that when 50 new data points are added to the clustered dataset of 500, 1000, and 1500 points, the execution time is only 34%, 13%, and 6%, respectively, compared to the iNS-DBSCAN execution time. The runtime is reduced to only 4% of the iNS-DBSCAN execution time when 2000, 2500, and 3000 points are clustered and further reduced to 2% when 4000 points were clustered. Therefore, the experimental data shows that the larger the size of the clustered data, the greater the rate of acceleration can be.

However, if a batch of 50 points is added to a too-small dataset, as in the first case with 100 points, the proposed incremental algorithm takes 146% more time. In this case, using the iNS-DBSCAN algorithm to cluster the entire dataset is recommended. In contrast, for larger datasets, the proposed algorithm takes less time to execute.

As depicted in Fig. 21, the runtime of the iNS-DBSCAN increases with the number of clustered points. In contrast, the execution time ratio of the proposed NS-DBSCAN algorithm tends to decrease.

The ESRI and Inside Airbnb data sources also obtain similar results to those with the OSM data. The test results are highlighted in bold in rows 3 and 5 of Table 9, indicating that the proposed algorithm gives better results. The greater speed is significant, at 3.4% (=10.5/311.4) and 5.9% (=6.08/101.8) of the original time needed, respectively, for the ESRI and Inside Airbnb dataset.

Measurement results according to the NMI index in column 11, Table 9 also show that the NS-IDBSCAN algorithm achieves similar results compared to the original algorithm.

Furthermore, the proposed algorithm was experimentally compared with the latest geospatial data clustering algorithm ACUTE (ACUTE_2023) [60] and also utilized the dataset in network space that ACUTE_2023 had used as Chicago.[3] The results of this experiment with the number of clustered points, added points, radius threshold, and density threshold set as (150, 20, 0.001, 5) are shown in Table 10.

Because the ACUTE_2023 clustering algorithm is based on topological relations by gradually expanding the radius of data points, the clustering result depends on the value r of each expansion. Therefore, ACUTE_2023 is performed on different step values of r (r_step). The experimental results in Table 10 show that the proposed NS-IDBSCAN incremental algorithm's execution time is lower while the quality is better than ACUTE_2023 according to the four evaluation indexes Silhouette, WSS, Dunn, and CH.

Algorithm ACUTE_2023 runs with many step values of r (r_step). All values of r_step need more time than with NS-IDBSCAN. Regarding cluster quality, for $r\_step = 0.00005$, five (of seven) index values are not as good as NS-IDBSCAN (WB and DB are better), $r\_step = 0.0001$ and $r\_step = 0.001$ have six (of seven) worse values, and $r\_step = 0.0005$, and all quality indicators are not as good as

[3] http://cs.uef.fi/mopsi/routes/network.

Table 10
The results of comparing the clustering quality indicators and time of algorithms NS-IDBSCAN and ACUTE_2023 for the Chicago dataset.

Algorithm	Sil (↑)	BSS (↑)	WSS (↓)	WB (↓)	DB (↓)	Dunn(↑)	CH (↑)	Runtime (seconds)
ACUTE_2023	No points clustered = 150, No points added = 20, $r = 0.001$, $MintPts = 5$							
0.00005	7.72E-01	1.29E-02	1.85E-03	4.30E-01	5.87E-01	9.63E-01	5.23E+01	0.33607
0.0001	8.04E-01	1.35E-02	1.75E-03	5.17E-01	7.45E-01	9.63E-01	5.41E+01	0.11672
0.0005	6.69E-01	1.57E-02	1.72E-03	6.58E-01	1.47E+00	9.63E-01	6.20E+01	0.03993
0.001	6.52E-01	1.71E-02	1.77E-03	8.31E-01	2.95E+00	5.17E-01	7.01E+01	0.02394
NS-IDBSCAN	8.40E-01	1.65E-02	1.46E-03	4.43E-01	9.38E-01	1.15E+00	4.66E+02	0.02294

Table 11
The results of comparing the clustering quality indicators and runtime of the most recent network-constrained geospatial data clustering algorithms with NS-IDBSCAN on some other datasets from OSM and Chicago.

Datasets	No points clustered	No points added	eps	MinPts	r_step (ACUTE_2023)	Algorithm	Sil (↑)	Runtime (seconds)	%Sil vs Proposed	% Runtime vs Proposed
CanTho	2000	10	800	20	20	NS-DBSCAN	0.8038	908.0704	89%	4231%
						iNS-DBSCAN	0.8038	814.0822	89%	3793%
						NS-TBC	0.8765	543.2967	97%	2532%
						ACUTE_2023	0.8870	23.8475	98%	111%
						NS-IDBSCAN	0.9038	21.4600		
DongNai	2000	10	400	20	20	NS-DBSCAN	0.0733	209.6113	10%	5378%
						iNS-DBSCAN	0.7032	197.1193	95%	5057%
						NS-TBC	0.7086	128.0886	96%	3286%
						ACUTE_2023	0.7706	26.2542	104%	674%
						NS-IDBSCAN	0.7407	3.8979		
Group(*)	4000	50	800	20	100	NS-DBSCAN	0.8058	1,645.7649	98%	6812%
						iNS-DBSCAN	0.8058	1,249.3389	98%	5171%
						NS-TBC	0.8700	625.4747	106%	2589%
						ACUTE_2023	0.7864	41.1715	96%	170%
						NS-IDBSCAN	0.8208	24.1586		
Chicago	170	10	0.001	5	0.001	NS-DBSCAN	0.5588	0.6159	66%	2684%
						iNS-DBSCAN	0.5588	0.6351	66%	2768%
						NS-TBC	0.5594	0.2454	67%	1070%
						ACUTE_2023	0.5513	0.0878	66%	383%
						NS-IDBSCAN	0.5635	0.0459		

(*) Group includes the provinces of Binh Dinh, Kon Tum, Gia Lai, Phu Yen, Quang Ngai, and Quang Nam in Vietnam from OSM.
This dataset is Group 1 in the paper [8].

with NS-IDBSCAN. Thus, it is necessary to experiment through many jump values (with each value taking more time) to get a better case with only one or two (of seven) of the evaluation indexes.

When executing ACUTE_2023 with multiple step values of radius r, each hop of r takes longer than with the NS-IDBSCAN algorithm. In terms of group quality, for $r\_step = 0.00005$, five (of seven) of evaluation index values are not as good as NS-IDBSCAN, $r_{step} = 0.0001$ and $r\_step = 0.001$ have six (of seven) of worse indexes, and $r\_step = 0.0005$, and all quality indicators are not as good as with NS-IDBSCAN. It thus needs to experiment through many step values of r (each step value taking more time than NS-IDBSCAN) to achieve a better case than only one or two (of seven) of the evaluation indexes, while the processing time for each case is longer.

Table 11 shows the results of comparing the proposed algorithm with some of the most recent network-bound geospatial data clustering algorithms, including NS-DBSCAN [6], iNS-DBSCAN [7], and NS-TBC [8] and on other data sets from OSM and Chicago.

The results show that the proposed NS-IDBSCAN algorithm greatly speeds up the execution time while enhancing the quality of the resulting cluster compared to the recent clustering algorithms in network space, namely NS-DBSCAN (2018, [6]), iNS-DBSCAN (2019, [7]), NS-TBC (2022, [8]) and ACUTE_2023 (2023, [44]).

Although many incremental clustering algorithms have been developed, they have not been specifically designed for network-constrained geospatial data.

In conclusion, the experimental findings indicate that the proposed NS-IDBSCAN has increased the clustering speed while maintaining the same or effectively improving cluster quality in many cases. The experimental results from comparing three data sources show that the proposed algorithm significantly reduces clustering time. Specifically, NS-IDBSCAN saves a lot of time for newly added data, and the time decreases as the size of the clustered data increases. The longer the algorithm takes to cluster once for the entire dataset, the greater the reduction in overall time.

5. Discussion

To discuss evaluating cluster quality, we present the following proposition.

Proposition 1. *Reducing one border point from a cluster will decrease the intra-cluster distance of the clustering result.*

Proof. The average distance among points within cluster C to its center C_P, $(x_i/i = 1..k)$ is calculated by the formula (1).

$$\frac{1}{k}\sum_{i=1}^{k}\|x_i - C_P\| \tag{1}$$

We have:

$$\frac{1}{k}\sum_{i=1}^{k-1}\|x_i - C_P\| = \frac{1}{k}(\|x_1 - C_P\| + \|x_2 - C_P\| + \cdots + \|x_{k-1} - C_P\| + \|x_b - C_P\|) \tag{2}$$

Where x_b is the border point. Assuming x_b is removed from C, the distance between the remaining $k-1$ points in the cluster C to its center C_P is:

$$\frac{1}{k-1}\sum_{i=1}^{k-1}\|x_i - C_P\| = \frac{1}{k-1}(\|x_1 - C_P\| + \|x_2 - C_P\| + \cdots + \|x_{k-1} - C_P\|) \tag{3}$$

In a geospatial dataset with network constraints, it can be assumed that the border point is a point of low density that is farther away from the cluster center. This is because the cluster center is typically a location of high density. Since x_b is a border point, so:

$$\|x_b - C_P\| > \|x_i - C_P\|, \quad (i = 1..k, i \neq b) \tag{4}$$

From (2), (3) and (4), we have (3) $<$ (2). Therefore, the intra-cluster distance of the clustering result includes m clusters $C_i/i = 1..m$ according to formula (5) will decrease.

$$SSW = \frac{1}{n}\sum_{i=1}^{m}\sum_{j\in C_i}\|x_j - C_{P(j)}\| \tag{5}$$

It is also certain that the clustering results with the reduced intra-cluster distance while the inter-cluster distance is equal or increased, which will give better cluster results in quality.

In principle, the proposed NS-IDBSCAN algorithm aims to reduce the experimental execution time, while maintaining the same accuracy as iNS-DBSCAN. The test results have also shown that not only does the NS-IDBSCAN achieve the expected acceleration, but it also demonstrated an improved clustering accuracy.

In the following, we will explain why there are instances where the cluster quality indexes of the NS-IDBSCAN algorithm show better results compared to iNS-DBSCAN.

- iNS-DBSCAN has cases where border points can belong to multiple clusters.
- On the other hand, the NS-IDBSCAN algorithm ensures that each point belongs to only one cluster.

The NS-IDBSCAN provides a better evaluation index because the iNS-DBSCAN algorithm has cases where the border points can belong to multiple clusters. On the other hand, the NS-IDBSCAN algorithm assigns only one cluster to each border point. Therefore, removing border points that appear in multiple clusters and retaining only one cluster reduces the value of the intra-cluster distance (Proposition 1). The intra-cluster distance decreases while the inter-cluster distance is not changed, thus improving overall cluster quality. The cluster quality is evaluated by the WB-index, defined as $WB = m * SSW/SSB$, where m is the number of clusters, SSW is intra-cluster distance, and SSB is the inter-cluster distance. When the inter-cluster distance stays the same or increases while the intra-cluster distance remains unchanged or decreases, the value of the WB-index becomes smaller. Yuvaraj et al. also concluded that a smaller WB-index means the detected clusters are both tighter and better separated [47].

Moreover, when border points belong to multiple clusters, classifying them into clusters with fewer points results in a better WB index. This is because moving a point to a cluster with a smaller diameter decreases the distance within the cluster.

What is presented above is proof of the theoretical statements, and in order to increase the reliability we experimented with evaluating the proposed approach according to the Silhouette index [46], which gave similar results.

The data from Can Tho [8] downloaded from OSM is used to illustrate this:

- Add point 2000 ($MinPts = 10, eps = 200$), there are 11 points belonging to two clusters.
  - iNS-DBSCAN: Point 1384 belongs to two clusters 0 and 2:
    Sil = 0.6113321287943133
  - NS-IDBSCAN:
    If assigning point 1384 to cluster 0 (1175 points):
    Sil = 0.6749213598213641.
    If assigning point 1384 to cluster 2 (10 points):
    Sil = 0.683172125493212.

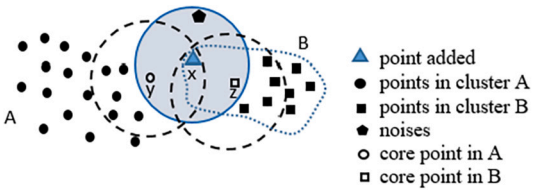

Fig. 22. Point x is density-reachable to y (belongs to cluster A) and z (belongs to cluster B). Assign to the x cluster with the lower number of points (cluster B) to reduce the imbalance in the number of points between clusters ($MinPts = 5$).

We have:
0.683172125493212 > 0.6749213598213641 > 0.6113321287943133.
(The value of Silhouette is higher indicating that x is close to the points in its containing cluster and far from other clusters [46]).
Illustrating the data of Group 3 [8] downloaded from OSM:

- Add point 3019 ($MinPts = 10$,$eps = 800$), there are seven points belonging to two clusters.
  - iNS-DBSCAN: Sil = 0.7386975697495171
  - NS-IDBSCAN: Sil = 0.7634403797526667 > 0.7386975697495171.

The technique is to ensure each point belongs to only one cluster in the NS-IDBSCAN algorithm by storing the clustering results in a one-dimensional array. The i^{th} element of the array contains the cluster Id to which point i belongs, and each element of the array has only one value and is the cluster's Id with fewer points.

Another solution to improve cluster quality when adding a new point x is to consider the number of core neighbor in $x's$ neighbors set. If x has many core neighbors, it should be added to the cluster with the fewest points, as shown in Fig. 22. This approach helps reduce the imbalance in the number of points between clusters. This is an important factor in improving clustering results, and contributes to solving the group size imbalance problem, an essential task in improving the quality of the results. However, this solution requires iteratively checking all the core neighbors of x, which takes a lot of time. Since this work aims to speed up the clustering time for the rapidly growing data problem, we present this idea as a proposal for further work focused on improving the clustering quality rather than reducing processing time.

Refining the algorithm's data structures by using other types of structures, such as edge lists or weighted adjacency lists instead of the weighted matrix, will save significant amounts of memory, especially when the graph is large, in order to manage varying dataset sizes and dimensions. Because the proposed method has to determine neighbors within the eps distance threshold of a point, it must traverse adjacent vertices. The weighted adjacency list allows efficient access to adjacent vertices of each vertex, along with their corresponding edge weights, making it suitable for this purpose.

Furthermore, loading into primary memory only the data related to the newly added point will greatly reduce storage capacity. For example, when adding a new point x, load only x, points on the same edge as x, and their edge weights into memory instead of reading all of points in the dataset. During algorithm processing, continue to load only the points adjacent (the same edge) to those that need to be processed into primary memory. This solution may take more time because it involves reading from secondary memory during algorithm processing, but it helps the algorithm to handle large-scale datasets as the dataset size increases and exceed the limits of primary memory.

Additionally, parallel processing on multicore processors or implementation on distributed systems is also an approach that is able to enhance scalability.

6. Conclusion and future works

Overall, the NS-IDBSCAN algorithm proposed in this work provides a promising incremental clustering solution for geospatial data in network space. When new data is added, instead of re-clustering the entire dataset our proposed approach only clusters the newly added data elements. This addresses the problem of wasting time aggregating the entire dataset, which speeds up the processing of the clustering algorithm and leads to faster clustering results.

The conversion from the two-dimensional structure of the clustered results (which includes the point Id and the cluster Id) to a one-dimensional array with the main array index as the point Id, and the array value as the clustering index of the corresponding points, provides the most efficient way to check if a point is noisy or already clustered. This solution contributes to the impressive acceleration of the proposed NS-IDBSCAN algorithm. Moreover, since time is proportional to effort, a decrease in algorithm execution time leads to a decrease in effort and cost. The NS-IDBSCAN can thus enrich spatial analysis tasks for information systems such as geographic information systems.

This work and the proposed algorithm have the following main contributions: the work proposes a clustering algorithm for newly added data that improves clustering quality and execution time. Secondly, NS-IDBSCAN solves the problem of rescanning the entire database when new data is added to existing data. Thirdly, the proposed algorithm overcomes the limitations of the iNS-DBSCAN for the phenomenon that border points belong to two or more clusters. Moreover, the study proposes a proposition concerning cluster selection for border points likely to belong to multiple clusters.

Future research should explore parallelization, distribution, or stream processing to meet real-time needs. Other incremental clustering methods should also be considered to solve clustering problems for network-constrained data. Additionally, the incremental

clustering of time series should be studied. A move towards practical application in real-world scenarios will be pursued in the next phase of our research.

CRediT authorship contribution statement

Trang T.D. Nguyen: Writing – original draft, Software, Data curation. **Loan T.T. Nguyen:** Writing – review & editing, Validation, Supervision, Methodology. **Quang-Thinh Bui:** Writing – review & editing, Validation. **Le Nhat Duy:** Writing – review & editing, Validation, Supervision. **Bay Vo:** Writing – review & editing, Validation, Investigation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgement

This research is funded by the Vietnam National Foundation for Science and Technology Development (NAFOSTED) under grant number 102.05-2021.08.

Appendix A. Pseudocode along with time complexity and space complexity evaluation of the iNS-DBSCAN algorithm

The total number of input points is n, and for brevity, the distance from cp to p is denoted as $d(p)$.

Algorithm A1: Determines $eps-neighbor$ of a point (Local shortest-path distance - LSPD) [8].

Input: Undirected planar graph $G(V, E, W)$, cp, eps.
Output: $N\_eps(cp)$

```
1  d(cp) = 0, d(cp') = ∞, ∀cp' ∈ N, cp' ≠ cp, Q = ∅, N_eps(cp) = ∅
2  Q ← cp
3  while Q ≠ ∅ do
4      p ← Q
5      if p ∉ N_eps(cp) then
6          N_eps(cp) ← p
7      for each q that shares the same edge with p do
8          if d(p) < eps then
9              a = d(p) + W(p, q)
10             if a < d(q) and a ≤ eps then
11                 d(q) = d(p) + W(p, q)
12             if q ∉ Q then
13                 Q ← q
```

Assessing the complexity of Algorithm A1:

To obtain $N\_eps(cp)$, Algorithm A1 must traverse all points where the shortest path from cp is less than or equal to eps.

The loop at line 7 then traverses each point q adjacent to cp within the radius eps. The runtime of this operation is O(de), where de is the degree of vertex p. If q is within a radius less than eps, the algorithm continues from point q as shown in the central command lines 10, 13, and 3. The number of iterations in line 3 equals the number of eps-neighbors of vertex cp, denoted by nb. Therefore, the time complexity of Algorithm A.1 is O(nb*de), where nb refers to the count of p's neighbors (line 13 and line 3), and de indicates the degree of p's neighbors (line 7).

The running time of the LSPD algorithm to find neighbors within an eps radius for a vertex depends on the number of edges adjacent to the vertex (vertex's degree: de) and the eps radius to get the number of neighbor points (nb). Since nb depends on eps, it is not too large. Furthermore, the degree of each vertex is also a small number compared to the size of the dataset. So, the product of $nb * de$ is not large relative to the data size because both nb and de are limited by radius eps.

Therefore, time complexity of Algorithm A1 (LSPD) is $O(nb * de)$.

The most significant space complexity of Algorithm A1 comes from the weight matrix containing the edge weights between n input points, which is $O(n * n) = O(n^2)$.

Assessing the complexity of Algorithm A2:

The most significant time complexity comes from the active operation in lines 2, 5, 9 and 11 of the algorithm. The algorithm traverses the entire dataset of n vertices (p) (line 2). For each vertex (p), the algorithm traverses the set Q can reach n point (q) (line

Algorithm A2: Generating Density Ordering Algorithm.

```
1  Input: Undirected planar graph G(V, E, W), eps.
2  Output: Density Ordering Table: T.
3  T ← ∅
4  for each p ∈ V do
5      if p ∉ T then
6          Q ← p
7          Calculate N_eps(p) by Algorithm A1
8      while Q ≠ ∅ do
9          q ← first vertex Q
10         if |N_eps(q)| ≥ ln(n) then
11             T ← |N_eps(q)| and N_eps(q)
12             for each s ∈ N_eps(q) do
13                 if |N_eps(s)| has not been calculated then
14                     Calculate |N_eps(s)| by using Algorithm A1
15                 if q ∉ T and q ∉ Q then
16                     Q ← q /* High-to-low density order                    */
17 Create a graph according to Table T.
```

5) and for each vertex q, the algorithm traverses nb neighbors (s) of q to determines s's neighbors using Algorithm A.1 (LSPD), which has a time complexity of $O(nb * de)$ (line 11) (Evaluated in algorithm A.1).

So, time complexity of Algorithm A2 is $O(n * n * nb * nb * de) = O(n^2 * nb^2 * de)$.

Similar to the evaluation of Algorithm A1, the space complexity of Algorithm A2 is also $O(n^2)$.

Algorithm A3: Forming Clusters Algorithm.

```
1  Input: Density Ordering Table T, MinPts
2  Output: Result clusters
3  for each p in T do
4      if there does not exist any cluster that contains p then
5          if |N_eps(p)| ≥ MinPts then
6              create cluster C, C ← p
7          for each q ∈ C do
8              if |N_eps(q)| ≥ MinPts then
9                  for each s ∈ N_eps(q) do
10                     if s ∉ C then
11                         C ← s
```

Assessing the complexity of Algorithm A3:

The algorithm considers the set of vertices in T (line 1) to verify a condition and add to the clustering result, which runs in $O(n)$ time. For each point in the clustering result (line 6), the algorithm checks if it is a core point and then examines its set of neighbors (line 8) to include them in the clustering result. The computational complexity of this process is $O(nc * nb)$, where nc is the number of points in a group and nb is the number of neighbors of a point in the group. Since both nc and nb are also small compared to the data size, the overall computational complexity of Algorithm 3 is O(n*nc*nb).

The algorithm iNS-DBSCAN executes algorithms A.2, which has a time complexity of $O(n^2 * nb^2 * de)$, and Algorithms A3 in sequence. So the complexity is determined by the addition rule to get the maximum value.

We have: $n * nc * nb < n^2 * nb^2 * de$.

Therefore, the time complexity of the iNS-DBSCAN algorithm is:

$O(max(n^2 * nb^2 * de, n * nc * nb)) = O(n^2 * nb^2 * de)$

The space complexity of the iNS-DBSCAN algorithm is $O(n^2)$.

References

[1] L.-Y. Wei, W.-C. Peng, An incremental algorithm for clustering spatial data streams: exploring temporal locality, Knowl. Inf. Syst. 37 (2) (2013) 453–483, https://doi.org/10.1007/s10115-013-0636-8.

[2] J. Han, M. Kamber, J. Pei, Data Mining: Concepts and Techniques: Concepts and Techniques, 3rd edition, Elsevier Science, 2012.

[3] L. Li, F. Zhang, J. Zhang, Q. Hua, C. Dong, C. Lim, A novel image clustering algorithm based on supported nearest neighbors, Comput. Sci. (2023) 1–20, https://doi.org/10.1142/S0129054122460017.

[4] Z. Halim, M. Waqas, A. Baig, A. Rashid, Efficient clustering of large uncertain graphs using neighborhood information, Int. J. Approx. Reason. 90 (2017) 274–291, https://doi.org/10.1016/j.ijar.2017.07.013.

[5] L. Xu, Q. Hu, E. Hung, B. Chen, X. Tan, C. Liao, Large margin clustering on uncertain data by considering probability distribution similarity, Neurocomputing 158 (2015) 81–89.
[6] T. Wang, C. Ren, Y. Luo, J. Tian, Ns-dbscan: a density-based clustering algorithm in network space, ISPRS Int.l J. Geo-Inf. 8 (5) (2019) 5, https://doi.org/10.3390/ijgi8050218.
[7] T. Nguyen, L. Nguyen, A. Nguyen, U. Yun, B. Vo, A method for efficient clustering of spatial data in network space, J. Intell. Fuzzy Syst. 40 (2021) 11653–11670, https://doi.org/10.3233/JIFS-202806.
[8] T. Nguyen, L. Nguyen, Q. Bui, U. Yun, B. Vo, An efficient topological-based clustering method on spatial data in network space, Expert Syst. Appl. 215 (2023) 119395, https://doi.org/10.1016/j.eswa.2022.119395.
[9] J. Xu, C. Xu, B. Zou, Y. Tang, J. Peng, X. You, New incremental learning algorithm with support vector machines, IEEE Trans. Syst. Man Cybern. Syst. 49 (11) (2019) 2230–2241, https://doi.org/10.1109/TSMC.2018.2791511.
[10] M. Ester, H.-P. Kriegel, J. Sander, M. Wimmer, X. Xu, Incremental clustering for mining in a data warehousing environment, in: Proceedings of the 24rd International Conference on Very Large Data Bases, VLDB'98, Morgan Kaufmann Publishers Inc., 1998, pp. 323–333.
[11] A. Bakr, N. Ghanem, M. Ismail, Efficient incremental density-based algorithm for clustering large datasets, Alex. Eng. J. 54 (4) (2015) 1147–1154, https://doi.org/10.1016/j.aej.2015.08.009.
[12] D. Shaweno, J. Trauer, T. Doan, J. Denholm, E. McBryde, Geospatial clustering and modelling provide policy guidance to distribute funding for active tb case finding in Ethiopia, Epidemics 36 (2021) 100470, https://doi.org/10.1016/j.epidem.2021.100470.
[13] Y. Wu, A. Pal, J. Wang, K. Kant, Incremental spatial clustering for spatial big crowd data in evolving disaster scenario, in: 2019 16th IEEE Annual Consumer Communications & Networking Conference (CCNC), IEEE, 2019, pp. 1–8.
[14] M. Deng, X. Yang, Y. Shi, J. Gong, Y. Liu, H. Liu, A density-based approach for detecting network-constrained clusters in spatial point events, Int. J. Geogr. Inf. Sci. 33 (3) (2019) 466–488, https://doi.org/10.1080/13658816.2018.1541177.
[15] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, 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'96, AAAI Press, Portland, Oregon, 1996, pp. 226–231.
[16] D. Wang, Y. Huang, Z. Cai, A two-phase clustering approach for traffic accident black spots identification: integrated gis-based processing and hdbscan model, Int. J. Injury Control Safety Prom. (2023).
[17] Z. Halim, M. Atif, A. Rashid, C. Edwin, Profiling players using real-world datasets: clustering the data and correlating the results with the big-five personality traits, IEEE Trans. Affect. Comput. 10 (4) (2019) 568–584, https://doi.org/10.1109/TAFFC.2017.2751602.
[18] A. Abernathy, M. Celebi, The incremental online k-means clustering algorithm and its application to color quantization, Expert Syst. Appl. 207 (2022) 117927.
[19] Q. Bui, B. Vo, H. Do, N. Hung, V. Snasel, F-mapper: a fuzzy mapper clustering algorithm, Knowl.-Based Syst. 189 (2020) 105107, https://doi.org/10.1016/j.knosys.2019.105107.
[20] Q. Bui, et al., Sfcm: a fuzzy clustering algorithm of extracting the shape information of data, IEEE Trans. Fuzzy Syst. 29 (1) (2021) 75–89, https://doi.org/10.1109/TFUZZ.2020.3014662.
[21] R. Tkachenko, I. Izonin, Model and principles for the implementation of neural-like structures based on geometric data transformations, in: Z. Hu, S. Petoukhov, I. Dychka, M. He (Eds.), Advances in Computer Science for Engineering and Education, Springer International Publishing, Cham, 2019, pp. 578–587.
[22] E. Schubert, J. Sander, M. Ester, H. Kriegel, X. Xu, Dbscan revisited, revisited: why and how you should (still) use dbscan, ACM Trans. Database Syst. 42 (3) (2017) 1–21, https://doi.org/10.1145/3068335.
[23] F. Ros, S. Guillaume, R. Riad, M. El Hajji, Detection of natural clusters via s-dbscan a self-tuning version of dbscan, Knowl.-Based Syst. 241 (2022) 108288, https://doi.org/10.1016/j.knosys.2022.108288.
[24] Y. Sun, G. Pan, Y. Li, Y. Yang, Differential evolution with nearest density clustering for multimodal optimization problems, Inf. Sci. 637 (2023) 118957, https://doi.org/10.1016/j.ins.2023.118957.
[25] M. Ankerst, M. Breunig, H.-P. Kriegel, J. Sander, Optics: ordering points to identify the clustering structure, in: Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data, SIGMOD'99, Association for Computing Machinery, New York, NY, USA, 1999, pp. 49–60.
[26] S. Mai, I. Assent, A. Le, Anytime optics: an efficient approach for hierarchical density-based clustering, in: S. Navathe, W. Wu, S. Shekhar, X. Du, X. Wang, H. Xiong (Eds.), Database Systems for Advanced Applications, in: Lecture Notes in Computer Science, Springer International Publishing, Cham, 2016, pp. 164–179.
[27] I.d.M. Ventorim, D. Luchi, A. Rodrigues, F. Varejão, Birchscan: a sampling method for applying dbscan to large datasets, Expert Syst. Appl. 184 (2021) 115518, https://doi.org/10.1016/j.eswa.2021.115518.
[28] Z. Halim, J. Khattak, Density-based clustering of big probabilistic graphs, Evolv. Syst. 10 (3) (2019) 333–350, https://doi.org/10.1007/s12530-018-9223-2.
[29] R. Leibrandt, S. Günnemann, Generalized density attractor clustering for incomplete data, Data Min. Knowl. Discov. 37 (2) (2023) 970–1009, https://doi.org/10.1007/s10618-022-00904-6.
[30] J. Xie, X. Liu, M. Wang, Sfknn-dpc: standard deviation weighted distance based density peak clustering algorithm, Inf. Sci. 653 (2024) 119788, https://doi.org/10.1016/j.ins.2023.119788.
[31] C. Li, S. Ding, X. Xu, H. Hou, L. Ding, Fast density peaks clustering algorithm based on improved mutual k-nearest-neighbor and sub-cluster merging, Inf. Sci. 647 (2023) 119470, https://doi.org/10.1016/j.ins.2023.119470.
[32] S. Chakraborty, S. Islam, D. Samanta, Research intention towards incremental clustering, in: Data Classification and Incremental Clustering in Data Mining and Machine Learning, Springer International Publishing, Cham, 2022, pp. 101–127.
[33] P. Kulkarni, P. Mulay, Evolve systems using incremental clustering approach, Evolv. Syst. 4 (2) (2013) 71–85, https://doi.org/10.1007/s12530-012-9068-z.
[34] S. Chakraborty, N. Nagwani, Analysis and study of incremental DBSCAN clustering algorithm, CoRR abs/1406.4, 2014.
[35] Q. Zhang, C. Zhu, L. Yang, Z. Chen, L. Zhao, P. Li, An incremental cfs algorithm for clustering large data in industrial internet of things, IEEE Trans. Ind. Inform. 13 (3) (2017) 1193–1201, https://doi.org/10.1109/TII.2017.2684807.
[36] S.-J. Gu, Fast incremental spectral clustering in titanate application via graph Fourier transform, IEEE Access 8 (2020) 57252–57259, https://doi.org/10.1109/ACCESS.2020.2982439.
[37] C. Wiwatcharakoses, D. Berrar, Soinn+, a self-organizing incremental neural network for unsupervised learning from noisy data streams, Expert Syst. Appl. 143 (2020) 113069, https://doi.org/10.1016/j.eswa.2019.113069.
[38] S. Mai, et al., Incremental density-based clustering on multicore processors, IEEE Trans. Pattern Anal. Mach. Intell. 44 (3) (2022) 1338–1356, https://doi.org/10.1109/TPAMI.2020.3023125.
[39] Y. Zhang, X. Li, L. Wang, S. Fan, L. Zhu, S. Jiang, An autocorrelation incremental fuzzy clustering framework based on dynamic conditional scoring model, Inf. Sci. 648 (2023) 119567, https://doi.org/10.1016/j.ins.2023.119567.
[40] M. Yiu, N. Mamoulis, Clustering objects on a spatial network, in: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, SIGMOD'04, Association for Computing Machinery, New York, NY, USA, 2004, pp. 443–454.
[41] R. Campello, P. Kröger, J. Sander, A. Zimek, Density-based clustering, WIREs Data Min. Knowl. Discov. 10 (2) (2020) e1343, https://doi.org/10.1002/widm.1343.
[42] G. Mishra, S. Mohanty, A fast hybrid clustering technique based on local nearest neighbor using minimum spanning tree, Expert Syst. Appl. 132 (2019) 28–43, https://doi.org/10.1016/j.eswa.2019.04.048.
[43] H. Alomari, A. Al-Badarneh, A topological-based spatial data clustering, in: D. Casasent, M. Alam (Eds.), Optical Pattern Recognition XXVII, SPIE, 2016, pp. 221–229.
[44] H. Alomari, A. Al-Badarneh, A. Al-Alaj, S. Khamaiseh, Enhanced approach for agglomerative clustering using topological relations, IEEE Access 11 (2023) 21945–21967, https://doi.org/10.1109/ACCESS.2023.3252374.

[45] K. Flores, S. Garza, Shortest paths, Knowl.-Based Syst. 206 (2020) 106350, https://doi.org/10.1016/j.knosys.2020.106350.
[46] P. Rousseeuw, Silhouettes: a graphical aid to the interpretation and validation of cluster analysis, J. Comput. Appl. Math. 20 (1987) 53–65, https://doi.org/10.1016/0377-0427(87)90125-7.
[47] M. Yuvaraj, A. Dey, V. Lyubchich, Y. Gel, H. Poor, Topological clustering of multilayer networks, Proc. Natl. Acad. Sci. 118 (21) (2021), https://doi.org/10.1073/PNAS.2019994118.