FB*: A Compact Index for Efficient and Exact Density-based Clustering

Bide Zhao
bzha0953@uni.sydney.edu.au
The University of Sydney
Sydney, Australia

Zhiyi Wang
zwan9517@uni.sydney.edu.au
The University of Sydney
Sydney, Australia

Lijun Chang
lijun.chang@sydney.edu.au
The University of Sydney
Sydney, Australia

Xin Huang
xinhuang@comp.hkbu.edu.hk
Hong Kong Baptist University
Hong Kong, China

ABSTRACT

Density-based clustering is a fundamental technique for discovering arbitrarily shaped clusters and handling noise, without requiring the number of clusters to be specified in advance. However, existing methods often struggle with efficiency and accuracy across varying query parameters: distance threshold ε and size threshold μ. In this paper, we propose a novel index-based algorithm for efficient and exact cluster extraction. We introduce FB, the first linear-size index that supports exact clustering with running time linear in the output size for any query ε and a fixed μ, along with an empirically compact variant, FB*, for efficiently extracting density-based clusters. Due to the compactness of the index and the efficiency of the query algorithm, our index is well-suited for disk-based storage, enabling multiple versions of the index — one for each distinct μ — to support arbitrary (ε, μ) queries. We provide formal analyses of time and space complexity. Extensive experiments on 23 real-world datasets demonstrate that our method significantly outperforms existing approaches while guaranteeing exact clustering results.

PVLDB Reference Format:
Bide Zhao, Zhiyi Wang, Lijun Chang, and Xin Huang. FB*: A Compact Index for Efficient and Exact Density-based Clustering. PVLDB, 19(8): 1791 - 1803, 2026. doi:10.14778/3811243.3811252

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/Zhaobd121/MST.

1 INTRODUCTION

Clustering is the task of grouping objects into coherent subsets, or *clusters*, such that objects within the same cluster are more similar to each other than to those in different clusters. It is a fundamental technique in exploratory data analysis and underpins a wide range of applications. For instance, clustering has been used to detect spatial hotspots such as crime or disease outbreaks in geographic information systems (GIS) [11], and to support image segmentation and robust object tracking in computer vision [9]. It also plays a key role in trajectory mining, where it helps identify frequent routes and movement patterns from GPS data [36], and in outlier detection by isolating data points that deviate from dense regions [3].

Proceedings of the VLDB Endowment, Vol. 19, No. 8 ISSN 2150-8097.
doi:10.14778/3811243.3811252

Among various clustering paradigms, density-based clustering has emerged as a powerful approach, particularly valued for its ability to discover clusters of arbitrary shape, naturally handle noise, and adapt to varying data densities. It makes minimal assumptions about the underlying data distribution and does not require the number of clusters to be specified in advance, making it well-suited for real-world datasets with complex structures. The key intuition behind density-based clustering is that clusters correspond to dense regions in the data space, separated by areas of lower density. Given a dataset D with a symmetric distance function $d : D \times D \to \mathbb{R}_{\geq 0}$, a distance threshold $\varepsilon \in \mathbb{R}_{\geq 0}$ and a size threshold $\mu \in \mathbb{N}$, density-based clustering is defined based on the concepts of core object and (directly) density-reachable. An object o is a *core* object if there are at least μ objects whose distances to o are at most ε, and is a non-core object otherwise. An object o' is *directly* density-reachable from o if o is a core object and $d(o, o') \leq \varepsilon$, and o' is density-reachable from o if there exists a sequence of objects $(o_1 = o, o_2, \ldots, o_k = o')$ such that o_{i+1} is directly density-reachable from $o_i, \forall 1 \leq i < k$. A density-based cluster is formed around a core object and includes all other objects that are density-reachable from it. Non-core objects that are part of at least one cluster are called *border* objects, while those that do not belong to any cluster are considered *noise*. A border object may belong to multiple clusters.

Given query parameters ε and μ, the popular DBSCAN algorithm [11] identifies all density-based clusters by iteratively detecting unprocessed core objects and forming clusters consisting of all objects that are density-reachable from them. However, choosing appropriate values for these two parameters often requires trial and error, making the process time-consuming when DBSCAN must be run for multiple thresholds. Although various pruning techniques have been proposed to improve DBSCAN's efficiency [2, 12, 13, 15, 19], it still remains impractical for interactive clustering tasks involving multiple parameter settings. OPTICS [1] is the first index-based approach designed to make density-based clustering efficient. Given a fixed size threshold μ and a *maximally supported* distance threshold $\bar{\varepsilon}$, the OPTICS index produces a *total ordering of all objects* in the dataset, denoted by $O_{\bar{\varepsilon},\mu}$, along with two associated values for each object. For any query threshold $\varepsilon \leq \bar{\varepsilon}$, clusters are extracted as *disjoint, consecutive sub-sequences* of $O_{\bar{\varepsilon},\mu}$.

As observed in [27], core objects are correctly clustered by the OPTICS index. However, border objects may be mis-classified as noise. Furthermore, due to the nature of the cluster extraction algorithm, each border object is assigned to at most one cluster. In view of this, the concept of ***exact clustering*** was introduced in [27], which simplifies density-based clustering by allowing each border object to be assigned to exactly one cluster, specifically, any one of the clusters it would belong to under density-based clustering.

The authors of [27] also modified the OPTICS index construction algorithm by adjusting the positions of certain objects in the ordering, ensuring that the clusters extracted for $\varepsilon = \bar{\varepsilon}$ satisfy the exact clustering criteria. This revised ordering is referred to as the FINEX index, which retains the same query processing algorithm as OPTICS. In addition, they proposed a post-processing procedure to obtain exact clustering for any $\varepsilon < \bar{\varepsilon}$. However, this post-processing increases query time complexity and requires access to the original dataset. It is also worth noting that, like the OPTICS index, the FINEX index cannot be used to process queries with $\varepsilon > \bar{\varepsilon}$.

Our Approach. In this paper, we propose a novel index structure, the FB index, which enables the extraction of exact clustering results for any query $\varepsilon \in \mathbb{R}_{\geq 0}$ and a fixed μ in optimal time, i.e., in time linear to the result size. Unlike OPTICS and FINEX, which rely on a single module for cluster extraction, our FB index comprises two separate modules: the F_c module, dedicated to extracting clusters of core objects, and the B module, responsible for assigning border objects to the appropriate core clusters. Our main motivation stems from the observation that core and border objects exhibit fundamentally different behaviors. Specifically, F_c is implemented as a minimum spanning forest of an edge-weighted graph constructed from the dataset D, while B stores a single optimal assignment for each object such that the assignment can be reused to determine the cluster membership of border objects for any query ε. The FB index has a total space complexity of $O(|D|)$ where $|D|$ denotes the number of objects in D. For any query ε, our query processing algorithm only accesses the edges in F_c and the assignments in B whose weights are at most ε, the number of which is linear in the size of the resulting clusters.

Furthermore, we enhance our approach by replacing the B module in FB index with a more advanced variant, B^*, which allows for the exact extraction of density-based clusters directly from the index. We prove that the size of B^* is bounded above by $(\mu-2)|D|$, and our empirical results demonstrate that its size remains comparable to that of B and does not exceed $|D|$. Additionally, our experiments show that the combined size of F_c and B^* remains relatively stable across different values of μ. Together with the observation that our query processing algorithm is well-suited for disk-resident indices, these results suggest that storing multiple versions of the index on disk — one for each distinct μ — is feasible, thereby supporting queries with varying μ values.

Our main contributions are summarized as follows:

- We develop the first linear-size index FB that supports the extraction of exact clustering results for any query ε and a fixed μ, with running time linear in the size of the output.
- We design the first compact index FB* that enables the extraction of density-based clusters for any query ε.
- We propose efficient index construction algorithms and prove their time and space complexities.
- Given the compactness of our index for a fixed μ and the efficiency of our disk-friendly query algorithm, multiple versions of our index can be stored on disk to efficiently support cluster extraction for any query ε and μ.
- Extensive experiments on 23 large real-world datasets demonstrate the efficiency and effectiveness of our techniques.

Organization. The rest of the paper is organized as follows. Section 2 defines the problem. Section 3 introduces the two existing index structures, OPTICS and FINEX. We introduce our new index structures, FB, and FB*, in Section 4. Experimental results are presented in Section 5. Section 6 discusses related work, and Section 7 concludes the paper. Proofs are omitted due to space limitations and are available in the online Appendix [35].

2 PRELIMINARIES

In this paper, we study index structures for density-based clustering, whereas our techniques are generally applicable to any dataset D with a symmetric distance function $d : D \times D \to \mathbb{R}_{\geq 0}$ such that $d(o, o) = 0, \forall o \in D$. For concreteness, we will use two-dimensional points and Euclidean distance function in our examples.

Given a distance threshold $\varepsilon \in \mathbb{R}_{\geq 0}$ and a size threshold $\mu \in \mathbb{N}$, density-based clusters are defined based on the following concepts.

Definition 1 (ε-neighborhood)**.** The ε-neighborhood of an object $o \in D$ is defined as

$$N_\varepsilon(o) = \{o' \in D \mid d(o, o') \leq \varepsilon\} \tag{1}$$

The ε-neighborhood of an object contains itself, i.e., $o \in N_\varepsilon(o)$.

Definition 2 (Core object and non-core object)**.** An object $o \in D$ is a core object with respect to (abbreviated as w.r.t.) (ε, μ) if $|N_\varepsilon(o)| \geq \mu$, and is a non-core object otherwise.

Definition 3 (Directly density-reachable)**.** For objects $o, o' \in D$, o' is directly density-reachable from o w.r.t. (ε, μ) if $d(o, o') \leq \varepsilon$ and o is a core object w.r.t. (ε, μ).

The following definition naturally extends this concept.

Definition 4 (Density-reachable)**.** For objects $o, o' \in D$, o' is density-reachable from o w.r.t. (ε, μ) if there exists a sequence of objects $o_1, o_2, \ldots, o_k \in D$ such that

(1) $o_1 = o$ and $o_k = o'$,
(2) $\forall 1 \leq i < k$, o_{i+1} is directly density-reachable from o_i.

Note that an object can be (directly) density-reachable only from core objects, indicating that the relation is asymmetric. In addition, the relation of density-reachable is transitive. With these concepts, we are now ready to formally define density-based clusters.

Definition 5 (Density-based cluster)**.** A non-empty subset $K \subseteq D$ is a density-based cluster w.r.t. (ε, μ) if it satisfies the conditions:

(1) Connectivity: there exists a core object $o \in K$ such that all other objects of K are density-reachable from o w.r.t. (ε, μ).
(2) Maximality: $\forall o, o' \in D$, if $o \in K$ and o' is density-reachable from o w.r.t. (ε, μ), then o' should also be in K.

Density-based clustering aims to find all density-based clusters in D for a given query pair ε and μ. According to the definition, each core object belongs to exactly one cluster, while a non-core object may belong to multiple clusters. Non-core objects that belong to at least one cluster are referred to as ***border*** objects, while other objects that do not belong to any clusters are ***noise*** objects.

Example 1. *Consider the two-dimensional points in Figure 1 with the Euclidean distance function, $\mu = 5$ and ε being the radius of the large circles. o_1, o_6, o_{11} and o_{12} are core objects. Both o_6 and o_{12} are*

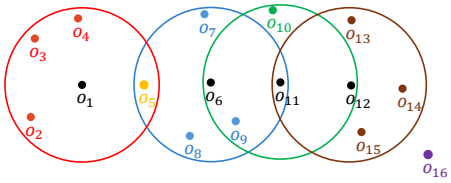


Figure 1: An example of density-based clusters

directly density-reachable from o_{11}, as their distances to o_{11} are within ε. Although o_{12} is not directly density-reachable from o_6, they are density-reachable from each other through o_{11}. There are two density-based clusters $K_1 = \{o_1, o_2, \ldots, o_5\}$ and $K_2 = \{o_5, o_6, \ldots, o_{15}\}$. o_{16} is a noise object, not belonging to any clusters. Among the border objects, o_5 belongs to both clusters, while each of the others belongs to one.

Recently, the concept of exact clustering was formulated in [27] to simplify the task of density-based clustering, by allowing each border object to be assigned to only one cluster, specifically, an arbitrary one to which it would belong in density-based clustering.

Definition 6 (Exact clustering). For a given dataset D and parameters ε and μ, let $K_1, \ldots, K_\ell$ be the density-based clusters. An exact clustering is a partitioning $K'_1, \ldots, K'_\ell$ of $\bigcup_{i=1}^{\ell} K_i$ such that

(1) $K'_i \subseteq K_i, \forall 1 \leq i \leq \ell$.
(2) All core objects of K_i are in K'_i, $\forall 1 \leq i \leq \ell$.
(3) Every border object is in exactly one partition that corresponds to the cluster it belongs to.

Note that, clusters in density-based clustering may overlap, while they are disjoint in exact clustering.

Example 2. *Reconsider Example 1, the sets $K'_1 = \{o_1, o_2, \ldots, o_4\}$ and K_2 form a valid exact clustering, as do K_1 and $K'_2 = \{o_6, \ldots, o_{15}\}$.*

Given a distance threshold $\varepsilon \in \mathbb{R}_{\geq 0}$ and a size threshold $\mu \in \mathbb{N}$, the classic DBSCAN algorithm [11] computes the results of density-based clustering (and thus also exact clustering) by iteratively detecting an unprocessed core object $o \in D$ and creating a new cluster consisting of all objects that are density-reachable from o w.r.t. (ε, μ). In practice, it is often necessary to extract density-based or exact clusters across multiple distance thresholds, as the appropriate threshold is typically determined through trial and error. Then, it will be extremely time-consuming to invoke DBSCAN for each possible distance threshold, in view of its high time complexity. Motivated by this, we in this paper study the following problem.

Problem Statement. Given a dataset D and a size threshold $\mu \in \mathbb{N}$, we in this paper study the problem of constructing an index that enables efficient and exact extraction of the results of either density-based or exact clustering for any distance threshold $\varepsilon \in \mathbb{R}_{\geq 0}$, thereby facilitating a more interactive clustering process.

Frequently used notations are summarized in Table 1. For ease of presentation, we define $\min \emptyset = \infty$ for an empty set $\emptyset$.

3 STATE-OF-THE-ART INDEX STRUCTURES

In this section, we review two state-of-the-art index structures, OPTICS [1] and FINEX [27], and identify their limitations.

Table 1: Frequently used notations

Notation	Description
D	A dataset
$o, o', o_1, o_2, \ldots$	Object of D
ε, μ	Distance threshold and size threshold
$d(o, o')$	The distance between o and o'
$N_\varepsilon(o)$	o's ε-neighborhood in D; note that $o \in N_\varepsilon(o)$
$O_{\bar{\varepsilon},\mu}, \widetilde{O}_{\bar{\varepsilon},\mu}$	OPTICS ordering and FINEX ordering
$o.C$	The minimum distance threshold ε such that o is still a core object w.r.t (ε, μ)
F_c, B, B^*	Three modules of our index structure

3.1 OPTICS

Given a size threshold μ and a ***maximally supported*** distance threshold $\bar{\varepsilon}$, the OPTICS index [1] consists of a total ordering of all objects in the dataset D, denoted by $O_{\bar{\varepsilon},\mu}$. The index $O_{\bar{\varepsilon},\mu}$ can be used for ***approximately*** extracting the results of any density-based or exact clustering with size threshold μ and distance threshold ε satisfying $\varepsilon \leq \bar{\varepsilon}$. This is why we refer to $\bar{\varepsilon}$ as the maximally supported distance threshold. That is, the index cannot be used for extracting clusters for distance thresholds larger than $\bar{\varepsilon}$.

For two distinct objects $o, o' \in O_{\bar{\varepsilon},\mu}$, we use $o \prec_{O_{\bar{\varepsilon},\mu}} o'$ to denote that o precedes o' in the total ordering $O_{\bar{\varepsilon},\mu}$. Besides the ordering $O_{\bar{\varepsilon},\mu}$, OPTICS also stores two values for each object o in $o.C$ and $o.R$, respectively. Firstly, it stores in $o.C$ the minimum distance threshold ε such that o is still a core object w.r.t (ε, μ), i.e.,

$$o.C = \min\{\varepsilon \mid \varepsilon \leq \bar{\varepsilon} \wedge |N_\varepsilon(o)| \geq \mu\} \tag{2}$$

Note that, if $|N_{\bar{\varepsilon}}(o)| < \mu$, then $\{\varepsilon \mid \varepsilon \leq \bar{\varepsilon} \wedge |N_\varepsilon(o)| \geq \mu\} = \emptyset$ and $o.C = \min \emptyset = \infty$. Consequently, for any $\varepsilon \leq \bar{\varepsilon}$, o is a core object w.r.t. (ε, μ) if and only if $o.C \leq \varepsilon$. Secondly, $o.R$ stores the minimum distance threshold ε such that o is still *directly* density-reachable from another object o' preceding it in the ordering $O_{\bar{\varepsilon},\mu}$. Note that for this to hold, o' must be a core object w.r.t. (ε, μ) and also the distance between o and o' must be no larger than ε. Thus,

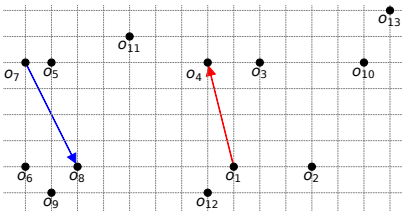
$$o.R = \min\{\varepsilon \mid \exists o' \prec_{O_{\bar{\varepsilon},\mu}} o \text{ s.t. } o'.C \leq \varepsilon \wedge d(o', o) \leq \varepsilon\} \tag{3}$$

If there does not exist any object $o' \prec_{O_{\bar{\varepsilon},\mu}} o$ such that $o'.C \leq \bar{\varepsilon}$, then $o.R = \min \emptyset = \infty$.

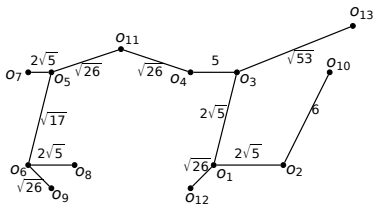
OPTICS computes the ordering $O_{\bar{\varepsilon},\mu}$ and $\{o.C, o.R\}_{o \in D}$ simultaneously, by iteratively and greedily choosing a vertex to be appended to the end of $O_{\bar{\varepsilon},\mu}$. Please refer to the original paper [1] for details.

Example 3. *Consider the 13 two-dimensional points in Figure 2(a) with $\mu = 5$ and $\bar{\varepsilon} = 8$. $o_7.C = d(o_7, o_8) = 2\sqrt{5}$ (indicated by the blue arrow), and $o_1.C = d(o_1, o_4) = \sqrt{17}$ (indicated by the red arrow). One valid OPTICS ordering is $O_{\bar{\varepsilon},\mu} = (o_1, o_2, o_3, o_4, o_{12}, o_{10}, o_{11}, o_{13}, o_5, o_6, o_7, o_8, o_9)$. Based on this ordering, $o_1.R = \infty$ since it is the first vertex in the ordering, $o_2.R = \max\{o_1.C, d(o_1, o_2)\} = \sqrt{17}$, and $o_3.R = \min\{\max\{o_1.C, d(o_1, o_3)\}, \max\{o_2.C, d(o_2, o_3)\}\} = \min\{\sqrt{17}, 2\sqrt{5}\} = \sqrt{17}$. The complete values for $o.C$ and $o.R$ are shown in Figure 2(c), where the first row shows the ordering $O_{\bar{\varepsilon},\mu}$.*

Given $O_{\bar{\varepsilon},\mu}$ and $\{o.C, o.R\}_{o \in D}$, the clusters for $\varepsilon \in (0, \bar{\varepsilon}]$ and μ are extracted as ***disjoint*** consecutive sub-sequences of $O_{\bar{\varepsilon},\mu}$. Specifically, each cluster starts from an object o that is a core object w.r.t. (ε, μ) (i.e., $o.C \leq \varepsilon$) and is not density-reachable from any preceding objects in $O_{\bar{\varepsilon},\mu}$ w.r.t. (ε, μ) (i.e., $o.R > \varepsilon$), and ends at (exclusive) the



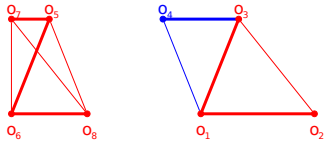
(a) A dataset of 13 data points



(b) A minimum spanning forest of the core graph

o	o_1	o_2	o_3	o_4	o_{12}	o_{10}	o_{11}	o_{13}	o_5	o_6	o_7	o_8	o_9
$o.C$	$\sqrt{17}$	$2\sqrt{5}$	$2\sqrt{5}$	5	$\sqrt{26}$	6	$\sqrt{26}$	$\sqrt{53}$	$\sqrt{17}$	$\sqrt{17}$	$2\sqrt{5}$	$2\sqrt{5}$	$\sqrt{26}$
$o.R$	∞	$\sqrt{17}$	$\sqrt{17}$	$\sqrt{17}$	$\sqrt{17}$	$2\sqrt{5}$	5	$\sqrt{29}$	$\sqrt{26}$	$\sqrt{17}$	$\sqrt{17}$	$\sqrt{17}$	$\sqrt{17}$

(c) OPTICS index for $\bar{\varepsilon} = 8$



(d) DDR graphs $G_{4.5}$ (consisting of red vertices and edges) and G_5 (the entire graph)

o	o_5	o_1	o_1	o_5	o_6	o_3	o_{12}	o_{11}	o_{11}	o_9	o_{10}	o_3
o'	o_6	o_2	o_3	o_7	o_8	o_4	o_1	o_5	o_4	o_6	o_2	o_{13}
w	$\sqrt{17}$	$2\sqrt{5}$	$2\sqrt{5}$	$2\sqrt{5}$	$2\sqrt{5}$	5	$\sqrt{26}$	$\sqrt{26}$	$\sqrt{26}$	$\sqrt{26}$	6	$\sqrt{53}$

(e) Module-I of our indices: F_c

o	o_2	o_3	o_4	o_7	o_8	o_9	o_{11}	o_{12}	o_{10}	o_{11}	o_{13}
o^*	o_1	o_1	o_1	o_5	o_6	o_6	o_5	o_1	o_3	o_4	o_3
w_b	$\sqrt{17}$	$\sqrt{17}$	$\sqrt{17}$	$\sqrt{17}$	$\sqrt{17}$	$\sqrt{17}$	$\sqrt{17}$	$\sqrt{17}$	$2\sqrt{5}$	5	$\sqrt{29}$

(f) Module-II of our indices: B^* (B is B^* excluding the shaded column)

Figure 2: An example comparing OPTICS with our FB and FB* indices for $\mu = 5$

first succeeding object o' that is not density-reachable from o w.r.t. (ε, μ) (i.e., $o'.R > \varepsilon$). The query time complexity is $\Theta(|D|)$, where $|D|$ denotes the number of objects in D.

Example 4. *Consider the OPTICS index in Figure 2(c). For a query with $\varepsilon = 4.8$, the first cluster starts from o_1 (since $o_1.C \leq \varepsilon < o_1.R$) and ends at o_{11} (exclusive). The second cluster starts from o_5 and ends at o_9 (inclusive). Both o_{11} and o_{13} are classified as noise objects. However, it can be observed that $o_5.C < \varepsilon$ and $d(o_{11}, o_5) = \sqrt{10} < \varepsilon$. Thus, o_{11} is mis-classified and should belong to the second cluster.*

Limitations of OPTICS. OPTICS has three main limitations.

(1) An OPTICS ordering $O_{\bar{\varepsilon},\mu}$ can only be used for extracting clusters with distance threshold $\varepsilon \in [0, \bar{\varepsilon}]$.
(2) The clusters extracted from $O_{\bar{\varepsilon},\mu}$ for any $\varepsilon \in [0, \bar{\varepsilon}]$ are not guaranteed to be exact clusters or density-based clusters.
(3) It has a high query time complexity of $\Theta(|D|)$.

As observed in [27], for any $\varepsilon \leq \bar{\varepsilon}$, all core objects w.r.t. (ε, μ) are correctly clustered. The inaccuracy is that some border objects may be mis-classified as noise objects, as shown in the example above.

3.2 FINEX

FINEX [27] follows the same idea as OPTICS, and proposes a slightly different ordering, denoted $\widetilde{O}_{\bar{\varepsilon},\mu}$, from which an exact clustering for $\varepsilon = \bar{\varepsilon}$ can be extracted in $\Theta(|D|)$ time. FINEX uses the same query (i.e., cluster extraction) algorithm as OPTICS. Its index construction algorithm is also similar to that of OPTICS. The only difference is that a non-core object w.r.t. $(\bar{\varepsilon}, \mu)$ may be removed from $\widetilde{O}_{\bar{\varepsilon},\mu}$ and re-appended to the end again. In this way, a non-core object w.r.t. $(\bar{\varepsilon}, \mu)$ would be correctly clustered for $\varepsilon = \bar{\varepsilon}$.

Nevertheless, exact clustering for $\varepsilon < \bar{\varepsilon}$ still cannot be directly extracted from $\widetilde{O}_{\bar{\varepsilon},\mu}$. This is because a core object w.r.t. $(\bar{\varepsilon}, \mu)$ may become non-core w.r.t. (ε, μ) and then is mis-classified as noise. For example, *for $\bar{\varepsilon} = 8$ and $\mu = 5$, the FINEX index is exactly the same as the OPTICS index as illustrated in Figure 2(c).* Thus, the border object w.r.t. $\varepsilon = 4.8$ will still be mis-classified as a noise object. To resolve this issue, a post-processing procedure is proposed in [27] to obtain exact clustering for $\varepsilon < \bar{\varepsilon}$. However, this post-processing increases the query time complexity and also requires access to the original dataset—unlike OPTICS, whose query processing does not need to access the original dataset.

In summary, the main limitations of FINEX are as follows.

(1) A FINEX ordering $\widetilde{O}_{\bar{\varepsilon},\mu}$ can only be used for extracting clusters with distance threshold $\varepsilon \in [0, \bar{\varepsilon}]$.
(2) The clusters extracted from $\widetilde{O}_{\bar{\varepsilon},\mu}$ for any $\varepsilon \in [0, \bar{\varepsilon})$ are not guaranteed to be exact clusters or density-based clusters. This also has a high query time complexity of $\Theta(|D|)$.
(3) Although post-processing can produce exact clustering for $\varepsilon < \bar{\varepsilon}$, it increases query time complexity and requires access to the original dataset.

4 OUR NEW INDEX STRUCTURE

In the following, we first in Sections 4.1–4.3 propose a new index structure that enables us to extract the results of *exact clustering* for any $\varepsilon \in \mathbb{R}_{\geq 0}$ in optimal time, i.e., in time linear to the result size. We then in Section 4.4 extend our index structure to enable efficient extraction of *density-based clusters* for any $\varepsilon \in \mathbb{R}_{\geq 0}$. We omit the size threshold μ in Sections 4.1–4.4, as we assume a fixed μ in these subsections. We will discuss how to extend our index structure to support varying size thresholds in Section 4.5.

4.1 Index Structure

The state-of-the-art index structures OPTICS and FINEX consist of a single module (i.e., the total ordering of objects), which is used for both extracting clusters of core objects and assigning border objects to those clusters. However, we observe that, for a given distance threshold ε, core objects and border objects exhibit different properties. Specifically, for any three core objects o_1, o_2, and o_3, if o_1 and o_2 belong to the same cluster, and o_2 and o_3 also belong to the same cluster, then o_1, o_2, and o_3 must all be in the same cluster. However, this property does not hold when border objects are involved. That is, for three objects o_1, o_2, and o_3, which may be either core or border objects, it is possible that o_1 and o_2 appear together in one cluster, and o_2 and o_3 in another, while no cluster contains both o_1 and o_3. In view of these, we propose a new index structure comprising two modules: one dedicated to extracting clusters of core objects, and another to assigning border objects to the appropriate core clusters.

4.1.1 Module-I for Extracting Clusters of Core Objects. To illustrate our dedicated index module for core cluster extraction, we first formalize the transitivity property of core object clusters.

For a given ε, we define a ***DDR*** (short for directly density-reachable) graph for core objects, denoted G_ε. Each vertex of G_ε corresponds to a *core* object of D w.r.t. ε, and there is an undirected edge between two vertices in G_ε if the distance between the two corresponding objects, as measured by $d(\cdot,\cdot)$, is no larger than ε. For example, the red vertices and edges in Figure 2(d) form the DDR graph G_ε for the dataset in Figure 2(a) with $\varepsilon = 4.5$ and $\mu = 5$. There are seven core objects $\{o_1, o_2, o_3, o_5, o_6, o_7, o_8\}$. It is easy to see that for any two vertices that are connected by a path in G_ε, their corresponding core objects must belong to the same cluster.

Lemma 1. *For any given ε, the connected components of the DDR graph G_ε are exactly the clusters of core objects.*

Fortunately, to preserve/encode the connected components of a graph, it is not necessary to retain all edges of the graph; instead, a ***spanning forest***—comprising a spanning tree for each connected component—is sufficient. A spanning tree of an undirected and connected graph G is a subgraph of G that is a tree (i.e., is connected and contains no cycles) and includes all the vertices of G. Let $V(G)$ and $E(G)$, respectively, denote the vertex set and edge set of G. Any spanning forest of an arbitrary graph G (connected or disconnected) contains at most $|V(G)| - 1$ edges. For example, for the DDR graph $G_{4.5}$ consisting of red vertices and edges in Figure 2(d), the five edges highlighted in bold red form a spanning forest.

Since the index structure is required to support the extraction of core object clusters across all possible ε values, storing only the spanning forest of a single DDR graph is inadequate. Furthermore, maintaining a spanning forest for each distinct DDR graph is impractical due to space limitations. To overcome this challenge, we make the following observation.

Lemma 2. *For any two distance thresholds $\varepsilon_1 < \varepsilon_2$, the DDR graph G_{ε_1} is a subgraph of the DDR graph G_{ε_2}.*

Following Lemma 2, for any spanning forest F_1 of G_{ε_1}, we can expand it to a spanning forest F_2 of G_{ε_2}, i.e., F_1 is a subgraph of F_2. For example, continuing the above examples, the DDR graph G_5 extends $G_{4.5}$ by one vertex o_4 and two edges $\{(o_4, o_3), (o_4, o_1)\}$. Furthermore, extending the spanning forest of $G_{4.5}$ by the edge (o_4, o_3) obtains a spanning forest of G_5.

Consequently, it suffices to store only F_2, from which F_1 can be derived. To enable this, we transform F_2 into a weighted forest by assigning edge weights such that every edge in F_1 has a lower weight than any edge in $F_2 \setminus F_1$. Similarly, edges in G_{ε_1} are assigned lower weights than those in $G_{\varepsilon_2} \setminus G_{\varepsilon_1}$. This allows us to iteratively expand the spanning forest by incorporating those edges that are newly added into the DDR graphs G_ε in increasing order of ε. This process closely resembles Kruskal's algorithm for computing a minimum spanning forest in an edge-weighted graph. Motivated by this, we define a ***core graph*** G_c, which is an undirected, edge-weighted and complete graph with the edge weight

$$w(o, o') = \max\{o.C, o'.C, d(o, o')\} \tag{4}$$

where $o.C$ is defined the same as Equation (2) but with $\bar{\varepsilon} = \infty$. Thus, for any $\varepsilon \in \mathbb{R}_{\geq 0}$, there is an edge between o and o' in G_ε if and only if $w(o, o') \leq \varepsilon$. Consequently, the clusters of core objects are preserved in the minimum spanning forest F_c of G_c as follows.

Lemma 3. *For any ε, the connected components of the subgraph of F_c that consists of all edges of weights no larger than ε are exactly the core object clusters that contain at least two core objects.*

Thus, we store the minimum spanning forest F_c of the core graph G_c as our module for extracting clusters of core objects. For example, Figure 2(b) shows a minimum spanning forest F_c of the core graph G_c for $\mu = 5$. Edge weights are shown beside the edges.

4.1.2 Module-II for Assigning Border Objects to Clusters. In exact clustering, for a given ε, a border object is allowed to be assigned to only one cluster, specifically, an arbitrary one to which it would belong in density-based clustering. This naturally raises the question: *for each border object, is it possible to store a **single assignment** (of it to a core object cluster) such that the assignment can be reused to determine its cluster membership for an arbitrary ε?* We answer this question affirmatively, by choosing an optimal single assignment for each border object based on the following observation.

Lemma 4. *In density-based clustering, for a border object o w.r.t. ε, if o belongs to the cluster of core object o' (w.r.t. ε) with $d(o, o') \leq \varepsilon$, then o also belongs to the same cluster as o' for any $\varepsilon' > \varepsilon$.*

Following Lemma 4, for the purpose of retrieving the results of exact clustering, it is safe to assign o to the same cluster as object

$$o^* = \arg\min_{o' \in D} \max\{o'.C, d(o, o')\} \tag{5}$$

in the index. Let $\varepsilon^* = \max\{o^*.C, d(o, o^*)\}$. Then, this ε^* is the smallest distance threshold such that there exists an object $o' \in D$ satisfying that (1) o' is a core object w.r.t. ε^* and (2) o is directly density-reachable from o' w.r.t. ε^*. However, it is possible that $\varepsilon^* \geq o.C$, indicating that o is also a core object w.r.t. ε^* and that o will never be assigned to a cluster as a border object. Thus, we define

$$D_o = \{o' \in D \mid \max\{o'.C, d(o, o')\} < o.C\} \tag{6}$$

which contains all the objects o' such that o can be assigned as a border object to the cluster of core object o' based on directly density-reachability, in density-based clustering. Specifically, D_o is the set of objects o' for which there exists an ε (e.g., $\max\{o'.C, d(o, o')\}$ would be the smallest such ε if exists) such that

- o' is a core object w.r.t. ε, i.e., $o'.C \leq \varepsilon$,
- o is a non-core object w.r.t. ε, i.e., $o.C > \varepsilon$, and
- o is directly density-reachable from o', i.e., $d(o, o') \leq \varepsilon$.

If $D_o = \emptyset$, then o will not be assigned to any cluster as a border object in density-based clustering. Otherwise, we store for o in our index structure the object

$$o^* = \arg\min_{o' \in D_o} \max\{o'.C, d(o, o')\} \tag{7}$$

Following the above discussions, this $\varepsilon^* = \max\{o^*.C, d(o, o^*)\}$ is the smallest distance threshold such that o is assigned to a cluster as a border object; that is, for any $\varepsilon < \varepsilon^*$, o will not be assigned to any cluster as a border object. Furthermore, by Lemma 4, this assignment can be reused for any distance threshold larger than ε^*.

Example 5. *Consider the dataset in Figure 2(a) and $\mu = 5$. $D_{o_1} = \emptyset$ and thus we do not store any object for o_1 in our index. $D_{o_3} = \{o_1\}$ and the object stored for o_3 is o_1. $D_{o_{11}} = \{o_4, o_5, o_7\}$ and the object stored for o_{11} is o_5 since $\max\{o_5.C, d(o_{11}, o_5)\} = \sqrt{17}$ achieves the minimum. Figure 2(f), excluding the shaded column, shows such information for all objects, where the object stored for o (row 1) is o^* (row 2).*

4.1.3 FB Index. In summary, our index structure consists of:

- **Module-I**: a minimum spanning forest F_c of the core graph.
- **Module-II**: the object $o^* = \arg\min_{o' \in D_o} \max\{o'.C, d(o, o')\}$ for each object $o \in D$.

To facilitate efficient extraction of the results of exact clustering, we store the minimum spanning forest F_c in Module-I as a list of edges (i.e., triplets $(o, o', w(o, o'))$ in non-decreasing order of edge weights $w(o, o')$, as shown in Figure 2(e). In addition, we also store the information of Module-II as a list of triplets $(o, o^*, w_b(o, o^*))$—where the weight $w_b(o, o^*)$ is defined as $\max\{o^*.C, d(o, o^*)\}$—in non-decreasing order of edge weights $w_b(o, o^*)$, as shown in Figure 2(f). We refer to the list of triplets of Module-II by B (stands for Border), and refer to our index structure as ***FB index***. Note that, each triplet (or edge) in F_c is symmetric (i.e., we can swap the order of o and o'), but it is asymmetric in B. Compared to FINEX/OPTICS, which store an ordering of objects (i.e., vertices), we store the ordering of two different subsets of object-object pairs (i.e., edges).

It is easy to see that the space complexity of our FB index is $O(|D|)$, which is the same as that of FINEX/OPTICS. More specifically, assuming that each object ID (i.e., o, o', and o^*) takes 4 bytes, and each numerical value (i.e., w and w_b) takes 8 bytes, then our index takes *at most* $32|D|$ bytes. In contrast, FINEX/OPTICS takes *exactly* $20|D|$ bytes. Thus, despite of decomposing our index into two modules, its space consumption is at most 1.6× that of FINEX/OPTICS. We remark that, our index sometimes even consumes smaller space than FINEX/OPTICS, as shown in our experiments; this is because the number of triplets in each of F_c and B could be significantly smaller than $|D|$.

4.2 Optimal Extraction of Exact Clustering

Given our FB index consisting of F_c and B, our cluster extraction algorithm for a given query distance threshold ε consists of two steps. Firstly, we extract the cluster of core objects (w.r.t. ε) as the connected components of the subgraph of F_c that consists of all edges of weight no larger than ε. Then, we assign border objects w.r.t. ε to the clusters by using B. The pseudocode of our algorithm is given in Algorithm 1, which does not require access to the original dataset D. The first step comprises Lines 1–12, while the second step comprises Lines 13–20.

We use a disjoint-set data structure $\mathcal{S}$ to dynamically grow the core object clusters by processing edges of F_c one-by-one. A disjoint-set data structure stores a collection of disjoint sets [10], typically in the form of trees, and supports two key operations:

- *Find*, which finds the set that contains a specific object.
- *Union*, which unions two sets into one set.

We incrementally create the disjoint-set data structure $\mathcal{S}$ which is initialized as empty (Line 2), and store the set of core objects that have been identified so far in a hash set C (Lines 1, 5, 6). For each edge $(o, o', w(o, o')) \in F_c$ with $w(o, o') \leq \varepsilon$, both o and o' are core objects (w.r.t. ε). Thus, we create a singleton set for o (resp. o') in $\mathcal{S}$ if o (resp. o') is not in C (and thus also not in $\mathcal{S}$), and then union the two sets containing o and o', respectively, into one set (Lines 5–8). Once all such edges have been processed, we create a cluster for each set $S \in \mathcal{S}$, and associate it with the representative object of the set (e.g., the object at the root of the tree representing the set). Specifically, each core object $o \in C$ is assigned to the cluster

Algorithm 1: ClusterExtraction(F_c, B, ε)

Input: Our FB index (F_c, B) and a query distance threshold ε
Output: The result $\mathcal{K}$ of exact clustering for ε

```
1  C ← ∅;        /* The set of core objects w.r.t. ε */;
2  Initialize an empty disjoint-set data structure S;
3  for each (o, o', w(o, o')) ∈ F_c in non-decreasing weight order do
4      if w(o, o') > ε then break;
5      if o ∉ C then Add o to C and also as a singleton set into S;
6      if o' ∉ C then Add o' to C and also as a singleton set into S;
7      if o and o' belong to different sets in S then
8          Union the sets of o and o' into one set in S;
9  Initialize the clusters to be empty;
10 for each object o ∈ C do
11     o_r ← the representative object of the set containing o in S;
12     K_{o_r} ← K_{o_r} ∪ {o};
13 for each (o, o*, w_b(o, o*)) ∈ B in non-decreasing weight order do
14     if w_b(o, o*) > ε then break;
15     if o* ∉ C then
16         Add o* to C and also as a singleton set into S;
17         K_{o*} ← {o*};
18     if o ∉ C then
19         o*_r ← representative object of the set containing o* in S;
20         K_{o*_r} ← K_{o*_r} ∪ {o};
21 return K as the collection of created clusters;
```

associated with the core object o_r, where o_r is the representative object of the set containing o in $\mathcal{S}$ (Lines 9–12).

After obtaining the core object clusters, we assign border objects (w.r.t. ε) to these clusters by processing all triplets of B with weight at most ε (Lines 13–14). For each such triplet $(o, o^*, w_b(o, o^*))$, o^* must be a core object as $o^*.C \leq w_b(o, o^*) \leq \varepsilon$. If o is not a core object (Line 18), we assign o to the cluster of o^* (Lines 19–20); otherwise, o should have already been put into the same cluster as o^* in Lines 10–12. Note that, it is possible that o^*, despite being a core object, has not been put into C in Lines 5–6; this happens when o^* is not directly density-reachable from any other core objects. [1] If this is the case (Line 15), we add o^* to C and also as a singleton set into $\mathcal{S}$ (Line 16), and initialize K_{o^*} as $\{o^*\}$ (Line 17).

Example 6. *Consider the FB index shown in Figures 2(e) and 2(f). For a query $\varepsilon = 5$, Lines 1–7 of Algorithm 1 process the first six edges of F_c (cf. Figure 2(e)), i.e., those with weights no greater than ε. As these edges are processed, objects are incrementally inserted into $\mathcal{S}$ and merged via union-find. Specifically, after processing (o_5, o_6) and (o_1, o_2), $\mathcal{S}$ contains two sets $\{o_5, o_6\}$ and $\{o_1, o_2\}$; processing (o_1, o_3) yields $\mathcal{S} = \big\{\{o_5, o_6\}, \{o_1, o_2, o_3\}\big\}$. After processing the fourth to sixth edges, $\mathcal{S}$ becomes $\big\{\{o_5, o_6, o_7, o_8\}, \{o_1, o_2, o_3, o_4\}\big\}$. As the seventh edge has weight $\sqrt{26} > \varepsilon$, all remaining edges are ignored. Subsequently, Lines 8–11 extract two core object clusters from $\mathcal{S}$: $K_{o_5} = \{o_5, o_6, o_7, o_8\}$ and $K_{o_1} = \{o_1, o_2, o_3, o_4\}$, with representatives*

[1] This also implies that Algorithm 1 may miss a cluster of size 1, when all the border objects that are directly density-reachable from the core object in the cluster are assigned to other clusters in the exact clustering. This issue can be easily rectified by also storing $\{(o, o.C) \mid o \in D\}$ in sorted order in our index, and scanning once the objects with $o.C \leq \varepsilon$ in query processing. For simplicity, we ignore such clusters of size 1, and thus do not need to store $o.C$ in our index.

o_5 *and* o_1*, respectively. Lines 12–19 then process the triplets in B (cf. Figure 2(f)), except the last two. The first five triplets involve only objects already in* $\mathcal{S}$ *and thus do not affect the clusters. Processing* (o_9, o_6) *and* (o_{11}, o_5) *adds* o_9 *and* o_{11} *to* K_{o_5}*, while processing* (o_{12}, o_1) *and* (o_{10}, o_3) *adds* o_{12} *and* o_{10} *to* K_{o_1}*. The final exact clusters are* $K_{o_5} = \{o_5, o_6, o_7, o_8, o_9, o_{11}\}$ *and* $K_{o_1} = \{o_1, o_2, o_3, o_4, o_{12}, o_{10}\}$.

Theorem 1. *Given* F_c*, B and* ε*, Algorithm 1 correctly extracts the result of exact clustering for* ε *in time linear to the result size.*

Remarks. We note that when the FB index is stored in main memory, query processing can also be conducted without using a disjoint-set data structure, e.g., by performing BFS or DFS traversals on the spanning forest F_c. Nonetheless, our Algorithm 1 offers the advantage of supporting query processing even when the index is stored on disk, which is particularly beneficial in Section 4.5.

As discussed in Section 3, both OPTICS and FINEX are capable of correctly identifying core object clusters. Thus, one might consider using the OPTICS or FINEX index for this purpose, instead of introducing the new module F_c. However, it is important to note that the running time for core cluster extraction using OPTICS and FINEX is $\Theta(|D|)$, which is less efficient than our F_c-based approach, whose running time is linear in the size of the resulting cluster. This optimal-time core cluster extraction becomes especially appealing when the index is stored on disk.

4.3 Index Construction

For completeness, we give the pseudocode of index construction in Algorithm 2. It consists of three parts. Firstly, it computes $o.C$ for each object o in the dataset (Lines 1–3), where $N(o)$ is the neighborhood of o (e.g., $N_\varepsilon(o)$ with $\varepsilon = \infty$); the construction of $N(o)$ depends on the dataset type, with details in the online Appendix. Although $o.C$ is not included in our index structure, it is still needed for the next steps of index construction. Secondly, it computes the minimum spanning forest F_c of the core graph G_c (Lines 5–22). This is achieved by running the classic Prim's algorithm [10] on G_c without actually materializing G_c; that is, the adjacent edges (or neighbors) of vertices are accessed one vertex at a time (Line 14). Thirdly, it computes the object o^* that should be stored for each object o' (Line 23–26); this is stored in Module B of our index.

The details of Part two (i.e., Prim's algorithm) are as follows. It processes each unprocessed object $o \in D$ (i.e., not already in F_c) in an arbitrary order (Lines 5–6). For each such object o, it initializes $pre(o)$ as **nil** (Line 7) and initializes a min-priority queue Q with entry $(o, 0)$ (Line 8); the second value of each entry is the ranking key in Q. Specifically, Q maintains the set of unprocessed objects that have at least one neighbor in F_c. For each such object o', its ranking key in Q is defined as the smallest edge weight among all edges connecting o' to its neighbors in F_c; the corresponding neighbor that yields this minimum edge weight is stored in $pre(o')$. Then, it iteratively pops the highest priority object o' from Q (Line 10), and adds the edge $(pre(o'), o')$ to F_c if $pre(o')$ exists (Lines 11–12). At the same time, Q is also updated by o''s neighbors (Lines 14–22).

Theorem 2. *Let m be the total (i.e., sum of) number of neighbors for objects in D, and* T_{nei} *be the total time complexity of getting the neighbors and their distances for all objects in D (i.e., the time complexity of Lines 1–2). Algorithm 2's time complexity is* $O(T_{nei} + m \log \mu + |D| \log |D|)$*, and its space complexity is* $O(|D|)$.

Algorithm 2: FB-Construction(D, μ)

Input: A dataset D and a size threshold $\mu \in \mathbb{N}$
Output: Our FB index (F_c, B)

1 **for each** *object* $o \in D$ **do**
2 Compute the distances between o and all its neighbors $N(o)$;
3 $o.C \leftarrow$ the μ-th distance in non-decreasing order;
4 Initialize F_c and B as empty;
5 **for each** *object* $o \in D$ **do**
6 **if** $o \in F_c$ **then continue**;
7 $pre(o) \leftarrow$ **nil**;
8 Initialize a min-priority queue Q with entry $(o, 0)$;
9 **while** $Q \neq \emptyset$ **do**
10 $o' \leftarrow$ pop the top object from Q;
11 **if** $pre(o') \neq$ **nil then**
12 Add edge $(pre(o'), o', w(pre(o'), o'))$ to F_c;
13 $o^* \leftarrow$ **nil**;
14 **for each** *object* $o'' \in N(o')$ **do**
15 **if** $o'' \notin F_c$ **then**
16 $w(o', o'') \leftarrow \max\{o'.C, o''.C, d(o', o'')\}$;
17 **if** $o'' \notin Q$ **then**
18 $pre(o'') \leftarrow o'$;
19 Insert entry $(o'', w(o', o''))$ into Q;
20 **else if** $w(o', o'') <$ *o'''s weight in* Q **then**
21 $pre(o'') \leftarrow o'$;
22 Update o'''s weight as $w(o', o'')$ and adjust its position in Q;
23 **if** $o^* =$ **nil or** $\max\{o''.C, d(o', o'')\} < \max\{o^*.C, d(o', o^*)\}$ **then**
24 $o^* \leftarrow o''$;
25 **if** $o^* \neq$ **nil and** $\max\{o^*.C, d(o', o^*)\} < o'.C$ **then**
26 Add entry $(o', o^*, \max\{o^*.C, d(o', o^*)\})$ to B;
27 Sort edges of F_c and entries of B into non-decreasing weight order;
28 **return** (F_c, B);

In the worst case, m can be as large as $|D|^2$, i.e., $N(o) = D, \forall o \in D$. Assuming that the distance between two objects can be computed in constant time, T_{nei} may also grow to $|D|^2$. Consequently, the worst-case time complexity of Algorithm 2 is essentially $O(|D|^2 \log \mu)$. This scenario arises for vector data with Euclidean distance.

In Section 5, we evaluate both set and vector data. For set data with Jaccard distance, however, m is typically much smaller than $|D|^2$. Let $I(e)$ denote the inverted list of element e, i.e., $I(e) = \{o' \in D \mid e \in o'\}$. In this case, $T_{nei} = \sum_{e \in S} |I(e)|^2$, where S is the set of all distinct elements in D. Details on computing the neighborhood $N(o)$ for both data types are provided in the online Appendix [35].

Potential Ways to Improve Index Construction Efficiency. As discussed above, m and T_{nei} in the time complexity of Algorithm 2 could reach $|D|^2$ and become the bottleneck, e.g., for vector data; note that, Algorithm 2's space complexity is $O(|D|)$ and will not be a bottleneck. One possible way to improve the efficiency of Algorithm 2 is parallel computing. Recall that, our index construction algorithm consists of three parts. Part one (Lines 1–3) and Part three (Lines 23–26) are trivially parallelizable, while Part two (Lines 5–22) involves computing a minimum spanning forest—a task for which

parallel algorithms are available (e.g., [22]). Another possible way is imposing a maximally supported distance threshold $\bar{\varepsilon}$, replacing $N(o)$ in Algorithm 2 with $N_{\bar{\varepsilon}}(o)$, and adopt existing proximity-based index structures (e.g., [16, 20]) to organize the dataset D, enabling efficient retrieval of $N_{\bar{\varepsilon}}(o)$. We leave these extensions to our future work, as they are orthogonal to our contributions.

4.4 Density-based Clustering

In this subsection, we consider density-based clustering and propose a different Module-II for assigning border objects to clusters; note that, Module-I of our index structure (i.e., a minimum spanning forest F_c of the core graph) remains intact. It is obvious that the Module-II introduced in Section 4.1 is insufficient for density-based clustering, since it stores only a single assignment per object, whereas a border object may belong to multiple clusters. On the other hand, storing D_o (defined in Equation (6)) for each object $o \in D$ would enable the correct extraction of density-based clusters. However, this may result in a significantly increased index size and thus also query processing time. We in the following show that not all objects in D_o need to be stored for o in the index.

Recall from Section 4.1 that D_o consists of the set of objects o' in D satisfying $\max\{o'.C, d(o,o')\} < o.C$. Let o^* be the object that attains the smallest $\max\{o^*.C, d(o,o^*)\}$ among all object of D_o. We stored o^* for o in the Module-II (i.e., B) of our FB index for the task of exact clustering in Section 4.1. Let's consider another object $o' \in D_o \setminus \{o^*\}$, and *let $\varepsilon_{o^*,o'}$ be the smallest ε such that o^* and o' belong to the same core object cluster.* If $\varepsilon_{o^*,o'} \le \max\{o'.C, d(o,o')\}$, then for every ε such that o is directly density-reachable from o' w.r.t. ε (and thus belongs to the same cluster as o'), we have $\varepsilon \ge \max\{o'.C, d(o,o')\} \ge \varepsilon_{o^*,o'}$, and thus o^* and o' have already been assigned to the same core object cluster. Consequently, the object o' does not provide any additional information (regarding cluster assignments of o) than o^*, and can be discarded from D_o. Based on this insight, we could remove from D_o all objects o' that satisfies $\varepsilon_{o^*,o'} \le \max\{o'.C, d(o,o')\}$. After this pruning, let o^{**} be the object that attains the next smallest $\max\{o^{**}.C, d(o,o^{**})\}$ in the resulting D_o. Then, o^{**} needs to be kept in our index, and we can prune D_o again based on o^{**}. We can continue this process, until no object can be pruned from D_o. *Let D_o^* be the final pruned version of D_o. We store D_o^* in Module-II of our index, denoted B^*.*

Example 7. *Reconsider Example 5. $D_{o_{11}} = \{o_4, o_5, o_7\}$ and $o^* = o_5$. Based on o^*, o_7 can be pruned since $\varepsilon_{o^*,o_7} = 2\sqrt{5} = \max\{o_7.C, d(o_{11}, o_7)\}$, o_4 cannot be pruned since $\varepsilon_{o^*,o_4} = \sqrt{26} > \max\{o_4.C, d(o_{11}, o_4)\} = 5$. After removing o_7, no more objects can be pruned. Thus, the final $D^*_{o_{11}}$ is $\{o_5, o_4\}$. The complete B^* is illustrated in Figure 2(f).*

One critical operation in the above pruning process is obtaining ε_{o_1,o_2} for any two objects. We observe that this can be obtained from Module-I of our index structure.

Lemma 5. *Given a minimum spanning forest F_c of the core graph, ε_{o_1,o_2} is equal to the maximum edge weight in the unique path between o_1 and o_2 in F_c. It is defined as ∞, if o_1 and o_2 are disconnected in F_c.*

Following the above discussions, the pseudocode of our algorithm for computing B^*, Module-II of our index structure for density-based clustering, is shown in Algorithm 3. For each object $o \in D$ (Line 2), we first get D_o (Lines 4–6) and sort entries in D_o into non-decreasing order regarding $w_b(\cdot,\cdot)$ (Line 7). Then, for each $(o', w_b(o,o')) \in D_o$ in non-decreasing order regarding $w_b(\cdot,\cdot)$ (Line 8), we loop through entries of D_o^* (Line 10) to check whether o' can be pruned based on an entry of D_o^* (Lines 11–12). If o' is not pruned, it is added to D_o^* (Line 13). After going through all entries of D_o, entries of D_o^* are added to B^* (Lines 14–15). Lastly, entries of B^* are sorted (Line 16), to support efficient query processing.

Algorithm 3: B^*-Construction(D, μ, F_c)

Output: Module B^* of our index for density-based clustering

```
1  B* ← ∅;
2  for each object o ∈ D do
3      D_o, D_o^* ← ∅;
4      for each object o' ∈ N(o) do
5          w_b(o,o') ← max{o'.C, d(o,o')};
6          if w_b(o,o') < o.C then D_o ← D_o ∪ {(o', w_b(o,o'))};
7      Sort entries in D_o in non-decreasing order regarding w_b(·,·);
8      for each (o', w_b(o,o')) ∈ D_o in the sorted order do
9          pruned ← false;
10         for each (o*, w_b(o,o*)) ∈ D_o^* do
11             w_max ← the maximum edge weight on the unique
                   path between o* and o' in F_c;
12             if w_max ≤ w_b(o,o') then pruned ← true; break;
13         if not pruned then D_o^* ← D_o^* ∪ {(o', w_b(o,o'))};
14     for each (o*, w_b(o,o*)) ∈ D_o^* do
15         B* ← B* ∪ {(o, o*, w_b(o,o*))};
16 Sort entries of B* into non-decreasing weight order;
17 return B*;
```

One of the bottlenecks of Algorithm 3 regarding running time lies in the frequent queries for the maximum edge weight on the unique path between two vertices in F_c (Line 11). A naive implementation would take $O(|D|)$ time per query. We note that this can be optimized to constant time per query through a preprocessing technique based on the *Kruskal Reconstruction Tree* (KRT) studied in [5] that takes $O(|D| \log |D|)$ time. Please refer to the online Appendix [35] for details of the preprocessing.

To analyze the time complexity of Algorithm 3, we first prove that $|D_o| \le \mu - 2$ holds for all $o \in D$.

Lemma 6. *For any $o \in D$, it holds that $|D_o| \le \mu - 2$.*

Theorem 3. *The time complexity of Algorithm 3 is $O(T_{nei} + \sum_{o\in D} |D_o| \cdot (\log |D_o| + |D_o^*|)) \subseteq O(T_{nei} + |D| \cdot \mu \log \mu + \mu \sum_{o\in D} |D_o^*|)$.*

As $D_o^* \subseteq D_o$, we have $\sum_{o\in D} |D_o^*| \le (\mu - 2)|D|$. Moreover, our empirical results in Section 5 show that $\sum_{o\in D} |D_o^*|$ is close to $|D|$ in practice. This can be partially explained by the lemma below.

Lemma 7. *Consider D_o for an object $o \in D$, where the entries of D_o are sorted in non-decreasing order regarding $w_b(\cdot,\cdot)$, see Line 7 of Algorithm 3. An object $o' \in D_o$ is pruned by Algorithm 3 if there exists an object o^* that precedes o' in this order such that $d(o', o^*) \le o^*.C$.*

A consequence of Lemma 7 is that, for any (pruned or unpruned) object $o^* \in D_o$, all subsequent objects of D_o in the sorted order that lie within the $o^*.C$-neighborhood of o^* are pruned from D_o when computing D_o^*. As $|D_o| \le \mu - 2$ and there are μ objects within the $o^*.C$-neighborhood of o^*, many objects are expected to be pruned, resulting in a small value of $\sum_{o\in D} |D_o^*|$.

The query processing algorithm for density-based clustering is almost the same as Algorithm 1, but takes F_c and B^* as inputs. The only other difference is that a border object may be added to the same cluster multiple times. As a result, post-processing may be needed to eliminate duplicate objects in a cluster; nevertheless, this post-processing is cheap.

4.5 Varying Size Threshold

Our discussions in the previous sections assume a fixed μ, and build an index specifically for that setting. To support varying query size thresholds, we propose to simply construct multiple versions of our index, each corresponding to a different μ value, leveraging two distinctive properties of our index.

- **Compactness:** For a fixed μ, our indices are space-efficient. The size of FB is theoretically bounded by $O(|D|)$, while FB*, although only bounded by $O((\mu-2)|D|)$, is empirically observed to scale linearly with $|D|$ in practice.
- **Sequential access pattern:** As the query processing algorithm described in Section 4.2 performs only one sequential access to prefixes of F_c and B (or B^*), it is well-suited for disk-based query processing.

Consequently, all index versions for different μ values can be stored on disk, and the appropriate portion corresponding to the query's μ can be efficiently accessed during query processing. Moreover, prior studies have shown that μ is typically chosen from a small set of candidate values [1], making our approach both efficient and practical in real-world scenarios.

We remark that although FINEX [27] also proposes techniques to process queries with $\varepsilon = \bar{\varepsilon}$ and a different size threshold $\mu' > \mu$ using a single index $\widetilde{O}_{\bar{\varepsilon},\mu}$, this is achieved through a post-processing step. The procedure is conceptually similar to the one discussed in Section 3.2, but it operates on the result of a query with parameters $\varepsilon = \bar{\varepsilon}$ and μ. This approach, however, has several limitations. (1) It increases query time complexity and requires access to the original dataset. (2) The query threshold μ' must be no less than the indexed μ. (3) The query ε must match $\bar{\varepsilon}$ exactly.

5 EXPERIMENTS

In this section, we empirically evaluate the performance of our FB index against the existing indices, FINEX and OPTICS, for the task of exact clustering. Additionally, we assess the effectiveness of our extended FB* index for the task of density-based clustering.

Compared Approaches. We compare the following approaches.

- Our indices: FB and FB*.
- The existing indices: FINEX and OPTICS.
- The existing index-free algorithm: DBSCAN.

All algorithms are implemented in C++ and compiled with GNU GCC with the -O3 optimization flag. All experiments are conducted on a machine with an Intel Core i9-12900KF CPU and 128GB main memory running Linux. All programs are run as single-thread programs. Disk I/O time is excluded from all experiments, except when evaluating our disk-based query processing algorithms.

Datasets. We evaluate the performance of all algorithms on two types of data: set data and vector data. Among the 19 set datasets, two (i.e., Kosarak and Enron) are from [28], while the remaining 17 are available on KONECT [2]. Each object is a set of elements. Statistics of these datasets are summarized in Table 2, where columns 2 and 3 show the number of sets and total set size, respectively. Datasets are ordered in ascending order regarding column 3.

Table 2: Dataset statistics and index sizes for set data (smaller index size between FB* and FINEX is highlighted in bold)

	#Sets	Set Size	Data Size	FB*	FINEX
YT (YouTube)	94,238	293,360	1.96 MB	**1.74 MB**	1.89 MB
GH (GitHub)	56,519	440,237	2.65 MB	**0.94 MB**	1.13 MB
LX (Linux)	42,045	599,858	4.20 MB	**0.64 MB**	0.84 MB
BS (Bibsonomy)	767,447	801,784	9.07 MB	**11.68 MB**	15.35 MB
BC (BookCross)	105,278	1,149,739	7.67 MB	**2.05 MB**	2.11 MB
AM (ActorMovie)	127,823	1,470,404	9.90 MB	3.61 MB	**2.56 MB**
WU (WebUni)	6,202	1,948,004	10.08 MB	0.19 MB	**0.12 MB**
CU (CiteULike)	731,769	2,338,554	17.23 MB	**14.06 MB**	14.64 MB
TV (TVTropes)	64,415	3,232,134	16.84 MB	1.57 MB	**1.29 MB**
IM (IMDB)	303,617	3,782,463	28.06 MB	8.06 MB	**6.07 MB**
DI (Discogs)	1,754,823	5,302,276	43.48 MB	**31.61 MB**	35.10 MB
AZ (Amazon)	2,146,057	5,743,258	53.73 MB	**34.45 MB**	42.93 MB
KO (Kosarak)	990,001	8,018,988	37.99 MB	20.22 MB	**19.57 MB**
FL (Flickr)	395,979	8,545,307	46.29 MB	9.13 MB	**7.92 MB**
DB (DBLP)	1,953,085	12,282,059	109.82 MB	**35.99 MB**	39.07 MB
EN (Enron)	517,431	69,114,932	332.16 MB	11.25 MB	**10.35 MB**
NY (NYTimes)	299,752	69,679,426	348.38 MB	8.76 MB	**6.01 MB**
DE (Delicious)	833,081	101,798,957	835.37 MB	22.89 MB	**16.66 MB**
OR (Orkut)	2,724,230	117,184,899	926.35 MB	81.31 MB	**54.49 MB**

Table 3: Dataset statistics and index sizes for vector data

	#Vectors	#Dimensions	Data Size	FB*	FINEX
DG (DatabaseGas)	416,153	9	24.5 MB	8.45 MB	**8.34 MB**
3D (3DSpatial)	434,874	4	23.3 MB	9.57 MB	**8.71 MB**
XY (XYRGB)	723,551	6	37.2 MB	19.16 MB	**14.49 MB**
HT (HTSensor)	928,991	8	75.2 MB	25.17 MB	**18.58 MB**

We use four vector datasets from the UCI archives [3]. Each object is a vector. Statistics of these datasets are summarized in Table 3, where columns 2–3 show the number and dimensions of the vectors.

5.1 Results on Set Data

In this subsection, we present the results on set data. The parameter μ is varied over $\{3, 6, 9, 12, 15, 18, 72, 288, 1152\}$, with $\mu = 6$ as the default. The query parameter ε is varied from 0.1 to 0.9 in increments of 0.2, with a default value of 0.5. When varying one parameter, the other is fixed at its default. For the OPTICS index, we set $\bar{\varepsilon} = 0.99$. For the FINEX index, we use two values of $\bar{\varepsilon}$: 0.99 (default) and 0.65.

5.1.1 Space Efficiency of Our Index. We first evaluate the space efficiency of our indices FB and FB*.

Exp-1: Index Size of FB* Across All Datasets. Recall that our FB index consists of F_c and B, while our FB* index comprises F_c and B^*. As $|B| \leq |B^*|$ always holds, we report the index size (in MB) of FB* in the fifth column of Table 2, using the default $\mu = 6$. The last column shows the index size of FINEX, which is bounded by $O(|D|)$. Note that OPTICS has the same index size as FINEX, since they differ only in the object ordering used; thus, OPTICS results are

[2] http://konect.cc/networks/
[3] https://archive.ics.uci.edu/datasets/

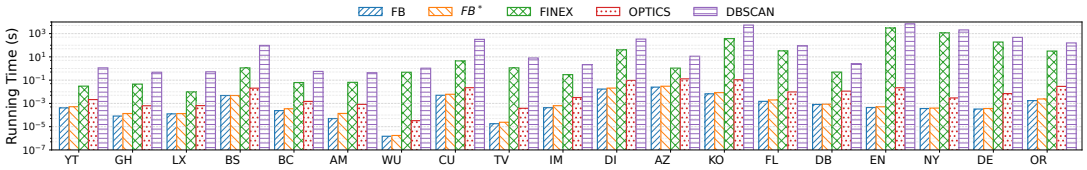


Figure 3: Query processing time across all set datasets ($\mu = 6$ and $\varepsilon = 0.5$)

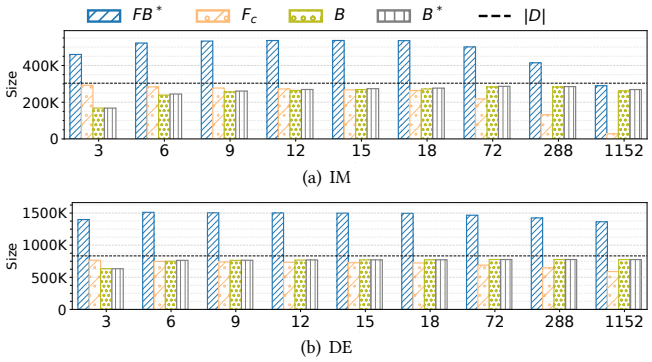


Figure 4: Individual component sizes of our index (vary μ)

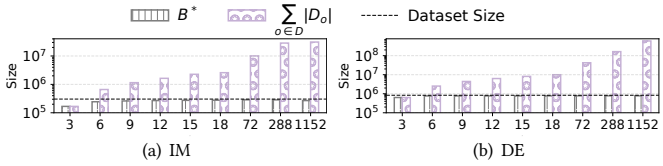


Figure 5: Comparison between $|B^*|$ and $\sum_{o \in D} |D_o|$ (vary μ)

omitted. As shown in the table, our index remains relatively small — typically much smaller than the dataset itself (fourth column). For example, for the DE dataset, storing the raw data requires 835 MB, whereas our index occupies only about 23 MB. Furthermore, the index size of FB* is comparable to that of FINEX. These results highlight the empirical compactness of our FB* index.

Exp-2: Individual Component Sizes of Our Index for Different μ. We also report the sizes of individual components of our index, *measured by the number of stored triplets*, under varying values of μ. The results are reported in Figure 4, which also shows $|D|$. We omit FINEX and OPTICS, as their indices have size identical to $|D|$. As shown, the sizes of both the F_c and B components are no larger than $|D|$, consistent with our theoretical analysis. Although the size of B^* is not theoretically bounded by $|D|$, empirical results show that it remains close in size to B and does not exceed $|D|$.

Moreover, the size of FB* remains relatively stable across different values of μ and may decrease as μ increases (e.g., on IM) due to fewer core objects, demonstrating robust space efficiency. Combined with the observation that the index size for each individual μ is small, this suggests that it is feasible to store multiple versions of the index on disk, enabling support for queries with varying μ values, as discussed in Section 4.5. Specifically, storing the index for all $3 \leq \mu \leq 20$ and all datasets takes only ≈ 4.95 GB space.

Exp-3: B^* vs. $\sum_{o \in D} |D_o|$. As discussed in Section 4.4, directly storing D_o for each object o would allow for correct extraction of density-based clusters. However, as illustrated in Figure 5, $\sum_{o \in D} |D_o|$ could be significantly larger than $|D|$ and $|B^*|$, particularly as μ increases; recall that, $\sum_{o \in D} |D_o| \leq (\mu - 2)|D|$. This highlights the effectiveness of shrinking D_o to D_o^* stored in B^* in Section 4.4.

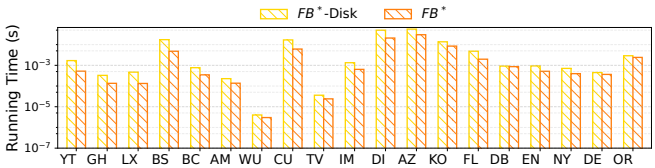


Figure 6: Disk-based query processing time ($\mu = 6$ and $\varepsilon = 0.5$)

5.1.2 Query Efficiency of Our Index. We now evaluate the efficiency of query processing using our indices.

Exp-4: Query Efficiency Across All Datasets. Figure 3 presents the query processing time for all datasets under the default query parameters, comparing different indices — FB, FB*, FINEX and OPTICS — as well as the index-free approach DBSCAN. FB and FINEX produce exact results for exact clustering, while OPTICS returns approximate clusters. On the other hand, FB* and DBSCAN yield exact results for density-based clustering. FB consistently achieves the best query performance across all datasets, while DBSCAN runs the slowest (e.g., 1.5 hours and 1.88 hours on datasets KO and EN, respectively), as expected. FINEX is significantly slower than OPTICS due to the additional post-processing required to produce exact clustering results; recall that $\varepsilon = 0.5 < \bar{\varepsilon} = 0.99$. FB* is slightly slower than FB, as its index size $|B^*|$ is slightly larger than $|B|$, as demonstrated in earlier experiments. Nonetheless, both FB and FB* outperform OPTICS, whose query time complexity is $\Theta(|D|)$. These results demonstrate that our FB and FB*-based query processing algorithm runs in sub-linear time in practice.

Exp-5: Efficiency of Disk-based Query Processing. In this experiment, we evaluate the efficiency of our query processing algorithms when the index is stored on SSD. This variant of FB* is referred to as FB*-Disk. Since the triplets in F_c and B^* are pre-sorted by weight, it is unnecessary to load the entire index into memory during query processing. Instead, only those triplets with weights less than or equal to the query threshold ε are retrieved. The results are shown in Figure 6. Note that, the reported time for FB*-Disk include I/O overhead, whereas that for FB* does not. The results show that our query processing algorithms remain efficient even when the index is stored on disk, with I/O overhead introducing only a negligible additional cost.

Exp-6: Query Processing Time by Varying ε. Query processing time of different approaches by varying ε is shown in Figure 8. We also include results for the FINEX index constructed with $\bar{\varepsilon} = 0.65$, referred to as FINEX65. The overall trend is similar to that observed in Figure 3, with FB and FB* consistently achieving the best performance. As ε increases, both FB and FB* require more

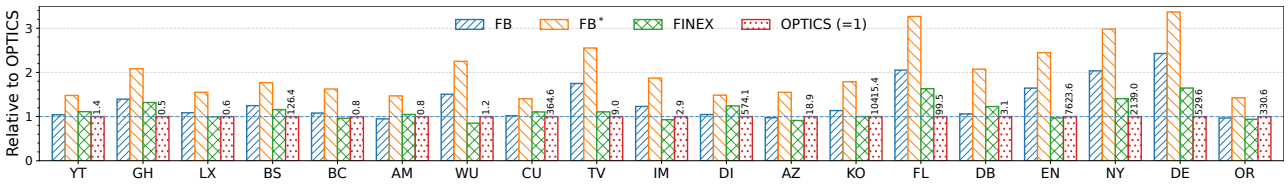
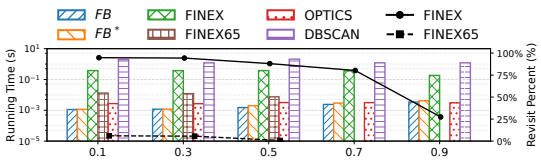


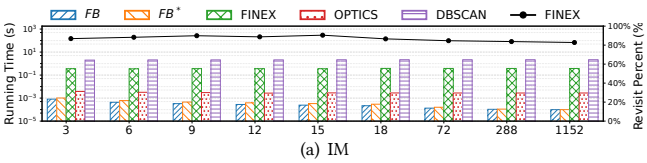
Figure 7: Index construction time relative to OPTICS on all set datasets ($\mu = 6$)



(a) IM

(b) DE

Figure 8: Query processing time by varying ε ($\mu = 6$)



(a) IM

(b) DE

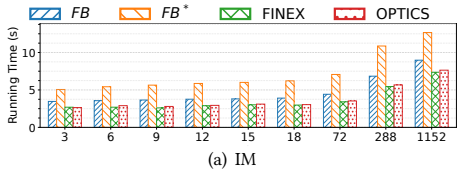
Figure 9: Query processing time by varying μ ($\varepsilon = 0.5$)

time due to larger result size, while OPTICS maintains a steady running time due to its $\Theta(|D|)$ query complexity. In contrast, FINEX and FINEX65 become faster as ε approaches their indexed $\bar{\varepsilon}$ value, with FINEX65 outperforming FINEX, due to requiring less post-processing. This is further supported by the line charts, which show a lower percentage of revisited vertices in post-procesing. Note that, FINEX65 does not produce results for $\varepsilon \in \{0.7, 0.9\}$, as it cannot support queries with ε values greater than its indexed $\bar{\varepsilon}$.

Exp-7: Query Processing time by Varying μ. Query processing time of different approaches by varying μ is shown in Figure 9. The overall trend is similar to that observed in Figure 3, with FB and FB* consistently performing the best. The running time of FINEX, OPTICS, and DBSCAN remain largely unaffected by changes in μ, while FB and FB* run faster due to smaller result size, for larger μ.

5.1.3 Index Construction Efficiency. We now evaluate the efficiency of our index construction algorithms.

Exp-8: Index Construction Time Across All Datasets. Figure 7 shows the construction time of different indices relative to OPTICS, using the default $\mu = 6$; the actual indexing time of OPTICS is also shown above its corresponding Bars. Overall, there is no significant difference in construction time among the different indexing methods, as our algorithm shares a similar time complexity with OPTICS



(a) IM

(b) DE

Figure 10: Index construction time by varying μ

and FINEX. The construction time of FB and FB* is at most 2.5× and 3.5× that of OPTICS, respectively. FB* requires slightly more time to construct than FB because it additionally executes Algorithm 3 to build B^*. Nevertheless, the construction time of FB* never exceeds twice that of FB. This efficiency stems from two factors: (1) the total size $\sum_{o \in D} |D^*_o|$, which determines the time complexity of constructing B*, remains close to $|D|$, as demonstrated in **Exp-3**; and (2) the preprocessing technique of F_c, described in Section 4.4, allows constant-time execution of Line 11 in Algorithm 3.

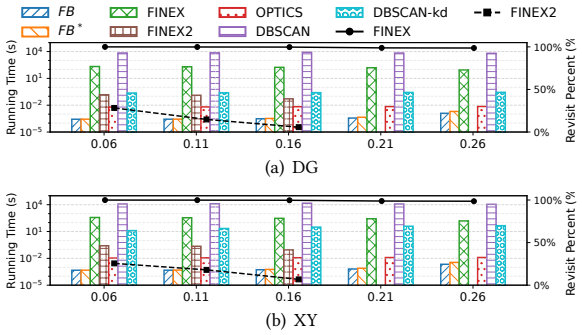
Exp-9: Index Construction Time by Varying μ. Figure 10 shows the construction time of different indices with varying μ values. We observe that as μ increases, the construction time of FB increases slightly on IM and remains relatively stable on DE. This is mainly because Lines 1–3 (in particular, Line 3) of Algorithm 2 take longer for larger μ due to the $m \log \mu$ factor in the time complexity of Algorithm 2, whereas Lines 5–26 tend to run slightly faster as μ increases due to fewer core objects. On IM, Lines 1–3 dominates the overall running time, while on DE the two components contribute comparably. Nevertheless, the variation in construction time is minor, confirming that index construction remains efficient across different μ values. Other indices exhibit a similar trend.

5.2 Results on Vector Data

We now evaluate the algorithms on vector data. We set $\mu = 64$ and vary the query parameter ε from 0.06 to 0.26 in increments of 0.05, with a default value of 0.11. For the OPTICS index, we set $\bar{\varepsilon} = 999$. For the FINEX index, we use two values of $\bar{\varepsilon}$: 999 (default) and 0.2.

The index sizes of FB* and FINEX, along with the raw data size (in MB), are reported in Table 3. Once again, the size of FB* is comparable to that of FINEX, as observed in Table 2. However, in this case, they are not significantly smaller than the raw data size. This is mainly because the number of dimensions is small, and therefore the raw data size is only a few times larger than $|D|$.

Figure 11 shows the query processing time of different approaches by varying ε, which also includes the results of FINEX constructed



(a) DG

(b) XY

Figure 11: Querying time by varying ε for vector datasets

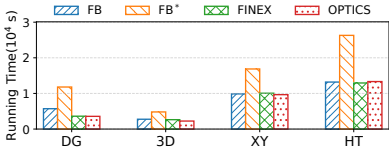
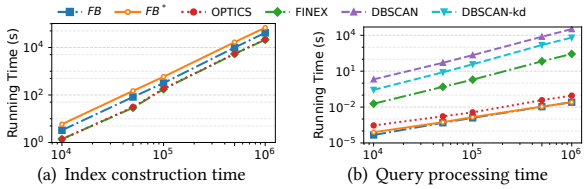


Figure 12: Index construction time for vector data ($\mu = 64$)



(a) Index construction time

(b) Query processing time

Figure 13: Index construction time and query processing time by varying $|D|$ ($\varepsilon = 0.11$ and $\mu = 64$)

with $\bar{\varepsilon} = 0.2$, referred to as FINEX2, and the results of an accelerated version of DBSCAN that leverages kd-tree to speed up the ε-neighborhood $N_\varepsilon(\cdot)$ computation [16], referred to as DBSCAN-kd. The overall trend is similar to that observed for the set dataset in Figure 8, with FB and FB* consistently outperforming others. We also observe that, with the help of a kd-tree, DBSCAN-kd runs significantly faster than DBSCAN. Nevertheless, it remains several orders of magnitude slower than FB*; for example, on the XY dataset, DBSCAN-kd is on average 17,150× slower. We remark that our constructed indices FB and FB* support arbitrary query values of ε, even when ε is extremely large. If an upper limit on the query ε is imposed (e.g., $\bar{\varepsilon}$), our index construction can be further accelerated by leveraging a kd-tree, as discussed at the end of Section 4.3.

Figure 12 shows the construction time of different indices for $\mu = 64$. The indexing times of the various indices are similar to one another, consistent with the results on set data in Figure 7.

We further evaluate scalability on synthetic datasets by varying the number of vectors, $|D| \in \{10^4, 5 \times 10^4, 10^5, 5 \times 10^5, 10^6\}$. The datasets are generated using a mixture of 9-dimensional Gaussian clusters following [15]. Index construction times are shown in Figure 13(a). All indices exhibit similar scaling behavior, with FB construction running in $O(|D|^2 \log \mu)$ time as discussed in Section 4.3. FB* incurs a modest additional cost due to constructing B^*. Query processing times are shown in Figure 13(b). FB- and FB*-based methods scale linearly with $|D|$ and significantly outperform the other methods, with the gap widening as $|D|$ increases. The improvement of DBSCAN-kd over DBSCAN is less pronounced than in Figure 11, due to more similar vector pairs in the synthetically generated data. Overall, these results demonstrate that the proposed index remains efficient and scalable for large-scale datasets.

In conclusion, our indices perform equally well on both set and vector data when compared with existing indices, FINEX and OPTICS, as well as the index-free approach DBSCAN.

6 RELATED WORK

Density-based Clustering. The concept of density-based clustering was introduced by Ester et al. [11] in 1996, along with the DBSCAN algorithm. Since then, a large body of work has focused on improving DBSCAN's efficiency, including pruning redundant pairwise distance computations [2, 14, 15, 18, 19, 29], divide-and-conquer strategies that partition the dataset into subsets [8, 17, 26, 32], hardware acceleration approaches [21, 23, 31], and approximation algorithms [12, 33]. Complementary to these directions, there are also works accelerating DBSCAN by leveraging spatial indexes, such as kd-tree and R-tree, to speed up ε-neighborhood queries [6, 7, 24]. In Section 5.2, we included an accelerated version of DBSCAN that leverages kd-tree, following [7], for comparison. These acceleration techniques are largely orthogonal to our work and could be incorporated to improve the efficiency of our index construction; we leave such integration as future work.

OPTICS [1] and FINEX [27] introduced index structures to support efficient online querying of density-based clusters. In this work, we follow this line of research by proposing a new compact index structure that enables efficient extraction of exact density-based clusters. Experimental results demonstrate that our index-based approach significantly outperforms OPTICS and FINEX in query efficiency while maintaining exactness.

Structural Graph Clustering. Density-based clustering has also been extended to graph-structured data, with the goal of grouping *vertices* into clusters; this is often referred to as structural graph clustering. Efficient index-free query processing algorithms have been proposed in [4, 25, 34], and index structures have also been developed in [30] to accelerate query processing. However, in structural graph clustering, two vertices are considered similar (i.e., have non-zero similarity or finite distance) only if they are connected by an edge; this constraint greatly reduces the number of candidate pairs and simplifies algorithm design. As a result, the indexing techniques developed for structural graph clustering do not directly generalize to general, non-graph data.

7 CONCLUSION

In this paper, we introduced FB, the first linear-size index that supports exact clustering with query time linear in the output size for any query ε. We also presented an empirically compact variant, FB*, designed for efficient extraction of density-based clusters for any query ε. Extensive experiments on 23 real-world datasets demonstrate that our method significantly outperforms existing methods while guaranteeing exact clustering results.

ACKNOWLEDGMENTS

Lijun Chang is supported by the Australian Research Council Fundings of DP220103731. Xin Huang is supported by the Hong Kong RGC Project No. 12200424.

REFERENCES

[1] Mihael Ankerst, Markus M Breunig, Hans-Peter Kriegel, and Jörg Sander. 1999. OPTICS: Ordering points to identify the clustering structure. *ACM Sigmod record* 28, 2 (1999), 49–60.

[2] Bhogeswar Borah and Dhruba K Bhattacharyya. 2004. An improved sampling-based DBSCAN for large spatial databases. In *International conference on intelligent sensing and information processing, 2004. proceedings of*. IEEE, 92–96.

[3] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. 2000. LOF: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. 93–104.

[4] Lijun Chang, Wei Li, Lu Qin, Wenjie Zhang, and Shiyu Yang. 2017. pSCAN: fast and exact structural graph clustering. *IEEE Transactions on Knowledge and Data Engineering* 29, 2 (2017), 387–401.

[5] Lijun Chang, Xuemin Lin, Lu Qin, Jeffrey Xu Yu, and Wenjie Zhang. 2015. Index-based Optimal Algorithms for Computing Steiner Components with Maximum Connectivity. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*. ACM, 459–474. https://doi.org/10.1145/2723372.2746486

[6] Min Chen et al. 2010. Parallel DBSCAN with Priority R-tree. In *Proceedings of the 2nd IEEE International Conference on Information Management and Engineering (ICIME)*. https://doi.org/10.1109/ICIME.2010.5477926

[7] Difei Cheng, Ruihang Xu, Bo Zhang, and Ruinan Jin. 2023. Fast Density Estimation for Density-based Clustering Methods. *Neurocomputing* 532 (2023), 170–182. https://doi.org/10.1016/j.neucom.2023.02.035

[8] Dongdong Cheng, Cheng Zhang, Ya Li, Shuyin Xia, and Guoyin Wang. 2024. GB-DBSCAN: A fast granular-ball based DBSCAN clustering algorithm. *Information Sciences* (2024). https://doi.org/10.1016/j.ins.2024.120731

[9] Dorin Comaniciu and Peter Meer. 2002. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on pattern analysis and machine intelligence* 24, 5 (2002), 603–619.

[10] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms, 3rd Edition*. MIT Press. http://mitpress.mit.edu/books/introduction-algorithms

[11] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, Vol. 96. 226–231.

[12] Junhao Gan and Yufei Tao. 2015. DBSCAN revisited: Mis-claim, un-fixability, and approximation. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*. 519–530.

[13] Junhao Gan and Yufei Tao. 2017. Dynamic density based clustering. In *Proceedings of the 2017 ACM international conference on management of data*. 1493–1507.

[14] Xiaogang Huang, Tiefeng Ma, Conan Liu, and Shuangzhe Liu. 2023. GriT-DBSCAN: a spatial clustering algorithm for very large databases. *Pattern Recognition* 142 (2023), 109658. https://doi.org/10.1016/j.patcog.2023.109658

[15] Jennifer Jang and Heinrich Jiang. 2019. DBSCAN++: Towards fast and scalable density clustering. In *International conference on machine learning*. PMLR, 3019–3029.

[16] Matthew B Kennel. 2004. KDTREE 2: Fortran 95 and C++ software to efficiently search for near neighbors in a multi-dimensional Euclidean space. *arXiv preprint physics/0408067* (2004).

[17] Daniel Kocher, Nikolaus Augsten, and Willi Mann. 2021. Scaling Density-Based Clustering to Large Collections of Sets. In *Proc. of EDBT'21*. OpenProceedings.org, 109–120.

[18] K. Mahesh Kumar and A. Rama Mohan Reddy. 2016. A fast DBSCAN clustering algorithm by accelerating neighbor searching using Groups method. *Pattern Recognition* 58 (2016), 39–48. https://doi.org/10.1016/j.patcog.2016.03.008

[19] Son T Mai, Ira Assent, and Martin Storgaard. 2016. AnyDBC: An efficient anytime density-based clustering algorithm for very large complex datasets. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 1025–1034.

[20] Willi Mann, Nikolaus Augsten, and Panagiotis Bouros. 2016. An empirical evaluation of set similarity join techniques. *Proceedings of the VLDB Endowment* 9, 9 (2016), 636–647.

[21] Lesia Mochurad, Andrii Sydor, and Oleh Ratinskiy. 2023. A fast parallelized DBSCAN algorithm based on OpenMp for detection of criminals on streaming services. *Frontiers in big Data* 6 (2023), 1292923.

[22] Vitaly Osipov, Peter Sanders, and Johannes Singler. 2009. The Filter-Kruskal Minimum Spanning Tree Algorithm. In *Proceedings of the Eleventh Workshop on Algorithm Engineering and Experiments, ALENEX 2009, New York, New York, USA, January 3, 2009*. SIAM, 52–61. https://doi.org/10.1137/1.9781611972894.5

[23] Andrey Prokopenko, Damien Lebrun-Grandie, and Daniel Arndt. 2023. Fast tree-based algorithms for DBSCAN for low-dimensional data on GPUs. In *Proceedings of the 52nd International Conference on Parallel Processing*. 503–512.

[24] Erich Schubert, Jörg Sander, Martin Ester, Hans-Peter Kriegel, and Xiaowei Xu. 2017. DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN. *ACM Transactions on Database Systems* 42, 3, Article 19 (2017). https://doi.org/10.1145/3068335

[25] Hiroaki Shiokawa, Yasuhiro Fujiwara, and Makoto Onizuka. 2015. Scan++ efficient algorithm for finding clusters, hubs and outliers on large-scale graphs. *Proceedings of the VLDB Endowment* 8, 11 (2015), 1178–1189.

[26] Dawid Suchy et al. 2023. GrDBSCAN: A Granular Density-Based Clustering Algorithm. *International Journal of Applied Mathematics and Computer Science* 33, 2 (2023), 297–312. https://doi.org/10.34768/amcs-2023-0022

[27] Konstantin Emil Thiel, Daniel Kocher, Nikolaus Augsten, Thomas Hütter, Willi Mann, and Daniel Schmitt. 2023. FINEX: A Fast Index for Exact & Flexible Density-Based Clustering. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–25.

[28] Xubo Wang, Lu Qin, Xuemin Lin, Ying Zhang, and Lijun Chang. 2019. Leveraging set relations in exact and dynamic set similarity join. *The VLDB Journal* 28 (2019), 267–292.

[29] Ziqing Wang, Zhirong Ye, Yuyang Du, Yi Mao, Yanying Liu, Ziling Wu, and Jun Wang. 2022. AMD-DBSCAN: An Adaptive Multi-density DBSCAN for Datasets of Extremely Variable Density. *CoRR* abs/2210.08162 (2022). https://arxiv.org/abs/2210.08162

[30] Dong Wen, Lu Qin, Ying Zhang, Lijun Chang, and Xuemin Lin. 2019. Efficient structural graph clustering: an index-based approach. *The VLDB Journal* 28, 3 (2019), 377–399.

[31] Guoqing Wu, Liqiang Cao, Hongyun Tian, and Wei Wang. 2022. HY-DBSCAN: A hybrid parallel DBSCAN clustering algorithm scalable on distributed-memory computers. *J. Parallel and Distrib. Comput.* 168 (2022), 57–69.

[32] Haochuan Xu and Ninh Pham. 2024. Scalable Density-based Clustering with Random Projections. In *Proceedings of NeurIPS 2024*. https://arxiv.org/abs/2402.15679

[33] Xiaowei Xu, Martin Ester, H-P Kriegel, and Jörg Sander. 1998. A distribution-based clustering algorithm for mining in large spatial databases. In *Proceedings 14th International Conference on Data Engineering*. IEEE, 324–331.

[34] Xiaowei Xu, Nurcan Yuruk, Zhidan Feng, and Thomas AJ Schweiger. 2007. Scan: a structural clustering algorithm for networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. 824–833.

[35] Bide Zhao, Zhiyi Wang, Lijun Chang, and Xin Huang. 2026. Online Appendix. https://github.com/Zhaobd121/MST/blob/main/appendix.pdf

[36] Yu Zheng, Lizhu Zhang, Xing Xie, and Wei-Ying Ma. 2009. Mining interesting locations and travel sequences from GPS trajectories. In *Proceedings of the 18th international conference on World wide web*. 791–800.