Scalable and Accurate Density-Peaks Clustering on Fully Dynamic Data

Daichi Amagata
Osaka University
Osaka, Japan
amagata.daichi@ist.osaka-u.ac.jp

***Abstract*—Clustering is a primitive and important operator that analyzes a given dataset to discover its hidden patterns and features. Because datasets are usually updated dynamically (i.e., it accepts continuous insertions and arbitrary deletions), analyzing such dynamic data is also an important topic, and dynamic clustering effectively supports it, but is a challenging problem. In this paper, we consider the problem of density-peaks clustering (DPC) on dynamic data. DPC is one of the density-based clustering algorithms and attracts attention for many applications, due to its effectiveness. We investigate the hardness of this problem theoretically to measure the efficiencies of dynamic DPC algorithms. We prove that any exact solutions are costly, and propose an approximation algorithm to enable faster updates. We conduct experiments on real datasets, and the results confirm that our algorithm is much faster and more accurate than state-of-the-art.**

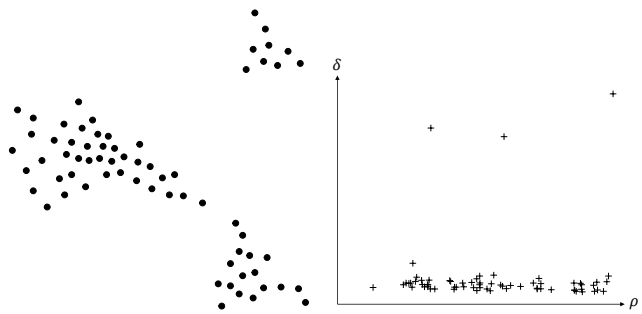
***Index Terms*—dynamic data, density-peaks clustering, metric space**

I. Introduction

Clustering is a primitive operator for data science, discovers patterns and events hidden in datasets, and supports data analysts in understanding the features of datasets. Therefore, clustering techniques in metric spaces have been studied in a wide range of fields, e.g., information retrieval [1], databases [2], data mining [3], artificial intelligence [4], and machine learning [5]. This paper considers density-peaks clustering (DPC) [6], one of the density-based clustering algorithms.

DPC forms clusters of a set P of objects based on *local density* and *dependent distance*. Given an object $p \in P$ and a user-specified cutoff threshold d_{cut}, its local density, ρ, is the number of other objects p' such that the distance between p and p' is no larger than d_{cut}. DPC obtains the dependent object of p, which is the nearest object to p with local density $> \rho$. The dependent distance of p, δ, is the distance between p and its dependent object. In DPC, a cluster center has the maximum local density (i.e., density-peak) among the objects in the cluster, so its dependent distance should be large. Informally, DPC considers objects with long dependent distances as cluster centers.

EXAMPLE 1. *Fig. 1(a) depicts a set of objects, and there are three clusters. Given d_{cut}, for each object, DPC computes its local density and dependent distance. The pair $\langle \rho, \delta \rangle$ of each object is mapped to a 2-dimensional space, as shown in Fig. 1(b), which is called a decision graph. The decision graph shows that three objects have much longer dependent distances than the others. This suggests that there are three clusters in the object set, as can be visually seen in Fig. 1(a).*



(a) Set of objects (b) Decision graph

Fig. 1. An example of mapping objects to a decision graph

Once cluster centers are determined, the other objects belong to the clusters of their dependent objects.

Motivation. DPC has several advantages: (i) It is unsupervised and does not require a priori knowledge to see the number of potential clusters, thanks to the decision graph (see Example 1). (ii) It can deal with arbitrary shaped clusters, whereas center-based clustering, such as k-center clustering [7], cannot. (iii) It can effectively divide a space with multiple density-peaks into different clusters even if there exist objects close to multiple clusters, while DBSCAN [8], which is also a famous density-based clustering, cannot cluster such datasets well [9], [10] (see Section VII for detail). Because of these advantages, DPC has been widely employed in data science applications. Concrete examples include graphics [11], neuroscience [12], time-series analysis [13], and visitor behavior analysis [14].

Because of the proliferation of IoT environments, in many applications, datasets are dynamically updated, i.e., new objects are inserted continuously and unnecessary objects are removed arbitrarily [15]–[18]. It is hence important to maintain clusters on dynamic data. Actually, the effectiveness of dynamic DPC in real-life applications has been already demonstrated. Gong et al. have shown that dynamic DPC effectively summarizes news topics online and is useful for news recommendation [19]. Ulanova et al. have tested DPC on three types of dynamic data: flying insect behaviors, human heartbeats, and mice ultrasonic vocalizations [20], and confirmed that dynamic DPC is more accurate than DBSCAN-based dynamic clustering. Their collaborators (domain experts) demonstrated

that the results of dynamic DPC provide meaningful knowledge. Zhang et al. have observed that dynamic DPC provides effective results on industrial IoT data [21].

The above works have demonstrated the importance of dynamic DPC, but they have issues regarding scalability. For example, [20] does not consider how to update dependent objects, and [21] incurs $O(n^2)$ time to update clusters for an object insertion, where n is the number of objects. EDMStream [19] reduces the number of processed objects with a sampling-like approach. However, because of this, as our experimental results confirm, its accuracy is low. Its efficiency, moreover, is dependent on a hyper-parameter, and its appropriate value is hard to specify.

Due to the dynamic nature of datasets generated in many real-life applications and the importance of dynamic DPC, its computational efficiency is worth addressing. We therefore address the problem of DPC on dynamic data.

Contributions. To the best of our knowledge, this is the first work to consider metric DPC in a general dynamic setting, which allows arbitrary object insertions and deletions. The main contributions of this paper are as follows:

(1) *Computational hardness of dynamic DPC in metric spaces* (Section IV). Because prior works have not considered the hardness of dynamic DPC in metric spaces, their optimality is not known. We prove that dynamic DPC in metric spaces requires $\Omega(n)$ time to solve exactly for a single update (i.e., object insertion/deletion) under a fixed dimensionality.

(2) *Approximation algorithm for fast update* (Section V). Because any exact solutions require $\Omega(n)$ time (i.e., an expensive update cost) even for a single update, we consider a new approximation algorithm that updates the dependency relationships with faster update time. First, we prove that heuristic approaches are required for approximate dynamic density-peaks clustering. We then devise a new graph-based data structure that takes approximately $O(\log^2 m+\hat{\rho})$ expected time for local density update, where m is the number of objects generated so far and $\hat{\rho}$ is an approximate local density of an object inserted/deleted. Because our new approximate local density computation yields highly accurate $\hat{\rho}$, density-peak spaces are preserved. For each object, by setting its close object with higher local density as its dependent object, we can efficiently update dependency relationships. For example, in an insertion case, our algorithm practically needs $\tilde{O}(\hat{\rho})$ time[1] to update dependency relationships, which is much faster than $\Omega(n)$ time.

(3) *Experiments on real-life dynamic data* (Section VI). We conduct experiments using real dynamic data. The experimental results show that our algorithm significantly improves update time while keeping a clustering result with high accuracy: it outperforms state-of-the-art w.r.t. both update time and accuracy.

In addition to the above contents, we formally define dynamic DPC in Section II and review related works in Section VII. This paper is concluded in Section VIII.

[1]The representation of $\tilde{O}(\cdot)$ ignores polylogarithmic factors.

II. Problem Definition

We assume a general dynamic setting, where new objects can be inserted into a system and arbitrary objects can be removed from the system. A dataset update corresponds to an object insertion or deletion, and it comes one by one [17], [19], [22]. Let P be a set of objects generated so far. In addition, let P_{active} be a set of objects generated so far and not removed (i.e., objects in $P-P_{active}$ are deleted objects), and $|P_{active}| = n$. We consider clustering objects in P_{active}. Density-peaks clustering (DPC) uses *local density* and *dependent distance* to form clusters [6]. We first define these.

Definition 1 (Local density). *Given a user-specified cut-off distance d_{cut}, the local density of an object $p_i \in P_{active}$, denoted by ρ_i, is*

$$\rho_i = |\{p_j \mid dist(p_i, p_j) \leq d_{cut}, p_j \in P_{active}\}|, \tag{1}$$

where $i \neq j$ and $dist(p_i, p_j)$ shows the distance between p_i and p_j.

We assume that the distance function satisfies that (i) $dist(p, p') = 0$ if $p = p'$, (ii) $dist(p, p') = dist(p', p)$, and (iii) triangle inequality. Here, let P_i^+ be a set of objects $p_j \in P_{active}$ such that $\rho_i < \rho_j$. To define dependent distance, we define *dependent object*:

Definition 2 (Dependent object). *The dependent object of $p_i \in P_{active}$ is*

$$q_i = \arg\min_{p_j \in P_i^+} dist(p_i, p_j). \tag{2}$$

Then, dependent distance follows:

Definition 3 (Dependent distance). *The dependent distance of $p_i \in P_{active}$, denoted by δ_i, is*

$$\delta_i = dist(p_i, q_i). \tag{3}$$

From Definition 2, the object p with the maximum local density in P_{active} cannot have its dependent object. We therefore set $\delta = \infty$.

DPC optionally determines noises from a user-specified threshold ρ_{min} to remove objects with small local densities.

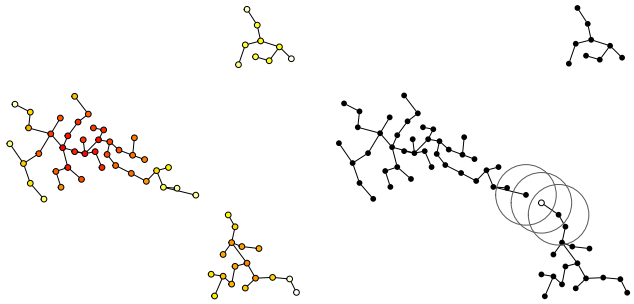
Definition 4 (Noise). *Iff $\rho_i \leq \rho_{min}$, $p_i \in P_{active}$ is a noise.*

Next, based on dependent distance and a user-specified threshold δ_{min}, we define cluster center.

Definition 5 (Cluster center). *Given δ_{min}, a non-noise object $p_i \in P_{active}$ is a cluster center iff $\delta_i \geq \delta_{min}$.*

Thanks to the decision graph, δ_{min} (and ρ_{min}) can be intuitively specified. For example, in Fig. 1(b), δ_{min} is specified so that the three objects with the largest dependent distance are selected as cluster centers.

For a cluster center p_i, we re-define its dependent object as itself. Consider non-noise objects p_j such that $q_j = p_i$. We see that p_j belongs to the cluster of p_i. Also, objects, whose dependent objects are p_j, belong to the cluster of p_i. In this sense, we say that an object p is reachable from a cluster



(a) Dependency relationships between objects. Darker (red) colors show denser local density.

(b) Dependency relationships after a new object insertion (white object)

Fig. 2. An example of Problem 2. The lines represent dependency relationships between objects.

center p_i if p_i can reach p by traversing dependent objects, so a cluster defined by DPC follows:

DEFINITION 6 (CLUSTER). *A cluster of P_{active} whose center is p_i is a set of non-noise objects in P_{active} that are reachable from p_i.*

Then, the problem of dynamic density-peaks clustering is:

PROBLEM 1 (DYNAMIC DENSITY-PEAKS CLUSTERING). *Given P_{active}, d_{cut}, ρ_{min}, and δ_{min}, the dynamic density-peaks clustering is to maintain clusters of P_{active} based on Definition 6.*

Because of object insertions and deletions, the local density and the dependent distance of each object are dynamic. It is hence desirable to specify ρ_{min}, and δ_{min} *on-demand*. This on-demand clustering query for dynamic data is common in the literature, e.g., [17], [20], [22], [23]. Once cluster centers are determined, cluster label propagation is done in $\Theta(n)$ time by traversing dependent objects from the cluster centers in a breadth-first search manner.

EXAMPLE 2. *In Fig. 2(a), the black lines represent the dependency relationships between objects in Fig. 1(a). Each object and line respectively can be considered as a vertex and an edge of a graph. The color of each node shows its local density, and darker (red) is denser. We see that cluster label propagation is done in a manner of breadth-first search.*

Now it is important to see that, for efficiently solving Problem 1, we have to address the following problem:

PROBLEM 2 (DEPENDENT OBJECT MONITORING). *Given P_{active} and d_{cut}, for each $p \in P_{active}$, this problem is to monitor (ρ and) q.*

EXAMPLE 3. *In Fig. 2(b), we have a new object (white object), which is inserted into P_{active}. The gray circles centered at the corresponding objects are balls with radius d_{cut}. Problem 2 computes the local density of the new object while updating the local densities of the corresponding objects. Then, the dependent object of the new object is computed while updating the dependent objects of the existing objects (in this example, they remain the same).*

In this paper, we address Problem 2, because Problem 1 is accordingly solved by solving Problem 2. Table I summarizes the symbols frequently used in this paper.

TABLE I
OVERVIEW OF SYMBOLS

Symbol	Description
P	a set of objects generated so far
P_{active}	a set of objects generated so far and not removed
p	an object
n	cardinality of P_{active}
m	cardinality of P
$dist(p, p')$	distance between p and p'
d_{cut}	cutoff distance (parameter for DPC)
ρ ($\hat{\rho}$)	(approximate) local density of p
q ($\hat{q}$)	(approximate) dependent object of p
p_{start}	the start node of our index
p_{ANN}	approximate nearest neighbor of p
$\hat{N}$	a set of objects p s.t. $\hat{\rho}$ is updated

III. BASELINE ALGORITHMS

Semi-static algorithm is an exact solution to Problem 2. Given a new object p, this algorithm computes its local density through a linear scan of P_{active} while updating the local densities of other objects p_i such that $dist(p, p_i) \leq d_{cut}$ [20]. (When p is removed from P_{active}, this algorithm updates ρ_i through a linear scan as well.) It can be seen that, because of the local density update, the dependent object of each object in P_{active} can be updated. Therefore, this algorithm computes q for each $p \in P_{active}$ from scratch[2].

This algorithm needs $O(n)$ time to update the local densities of all objects, and it incurs $O(n^2)$ time to update dependent objects. Clearly, this is not acceptable, even for a moderate n, and does not achieve real-time monitoring.

EDMStream [19] is a state-of-the-art approximate solution. The basic idea of EDMStream is to *ignore* a new object p_i if there is an existing *seed object* p_j such that $dist(p_i, p_j) \leq r$, where r is a hyper-parameter, and EDMStream sets $q_i = p_j$. EDMStream forms clusters based on a set of seeds. The local density of each seed is obtained from Equation (1), but its dependent object is the nearest seed with higher local density.

Given a new object p_i, it is absorbed to its nearest seed p_j if $dist(p_i, p_j) \leq r$. Otherwise, p_i becomes a seed. EDMStream then updates the local density of each seed, as with the semi-static algorithm. After that, EDMStream updates the dependent object of each seed p_j if necessary. For example, let p be the dependent object of p_j before p_i is inserted. If there exists a seed p' with $\rho' > \rho_j$ and $dist(p, p_j) > dist(p', p_j)$ because of the local density update, the dependent object of p_j has to be updated to p'. Object deletions can be handled similarly to object insertions. However, if a seed object is removed, EDMStream needs to select a new seed from a set of objects absorbed by the removed object and needs to compute its local density and update the dependency relationships.

When EDMStream needs to update the dependent object of a seed, it suffers from a large search space, typically rendering $O(n)$ time for each seed p that needs to update q. Besides, it

[2] [21] assumes that the number of cluster centers is known (and this is not practical). Without this, [21] also corresponds to this semi-static algorithm.

is difficult to specify an appropriate r, because its optimized value is highly dependent on data distributions. This object aggregation approach may not be able to accurately identify noises and borders between different clusters. Our empirical study observes that EDMStream does not yield high accuracy for our problem.

IV. Hardness of Dynamic DPC

Assume that dimensionality is fixed, i.e., $dist(\cdot,\cdot)$ is obtained in $O(1)$ time. The computational hardness of exact dynamic DPC in metric spaces is seen below.

THEOREM 1. *Given an update (object insertion or deletion), Problem 2 in metric spaces requires $\Omega(n)$ time to solve exactly.*

PROOF. Computing the dependent object of an object $p_i \in P_{active}$ corresponds to computing the nearest neighbor of p_i among P_i^+. In metric spaces, an exact nearest neighbor search requires $\Omega(n)$ time on a set of n objects [24] — ($\star$).

Here, assume that the local density of each object is distinct. This is true in practice, if we add a random value $\in (0,1)$ to ρ_i for each $p_i \in P_{active}$ [6]. For ease of presentation, let $\rho_i < \rho_{i+1}$ for each $i \in [1, n-1]$, to have distinct local density, and we have $|P_i^+| = n - i$. The amortized size of P_i^+ is $\frac{1}{n}\sum_{i=1}^{n}(n-i) = O(n)$. From this fact and ($\star$), this theorem holds. □

This theorem proves that any exact solutions need $\Omega(n)$ time for an update. Besides, the worst case is always $O(n^2)$ time. This is because we need $\Omega(n)$ time for each object that needs to update its dependent object and the number of such objects can be $O(n)$ for an update in the worst case. To conclude, any exact solutions are clearly expensive, and we need to design an approximate solution. Actually, many applications allow approximate results to achieve a fast update time [14], [17], [19], [21], [25]–[29]. To improve the update efficiency, we have to approximate the local density of each $p \in P_{active}$, because its exact computation incurs $\Omega(n)$ time. However, if we do this, we have the following result:

THEOREM 2. *Assume that the local density of each object in P_{active} is approximated and its exact local density is not known. Under this approximation, it is hard to guarantee the clustering quality in the DPC problem.*

PROOF. If ρ_i is approximated for each $p \in P_{active}$, the dependent objects of all objects in P_{active} may not be correct, because P^+ of each p is approximated. Therefore, the dependent distances of them are also approximated. We now see that an approximate dependent distance of an exact cluster center can be arbitrarily smaller than the exact one. This leads to no guarantee of returning the correct cluster centers. From this fact, Theorem 2 is clear. □

This theorem suggests that *faster update time than $\Omega(n)$ and approximation guarantee w.r.t. quality cannot be satisfied simultaneously*. The theoretical results of Theorems 1 and 2 suggest that dynamic DPC requires a heuristic approach providing fast update time and *empirically* accurate clustering result. Therefore, in this paper, we devise such an approximation algorithm for dynamic DPC.

V. Our Solution

Main ideas. Our idea for obtaining a clustering result with high accuracy is to *keep the local density distribution accurate*. The dependency relationships are formed toward density-peak spaces. If the local density of each object is approximated but its distribution is similar to the exact case, each p can still catch a space where p should depend on, and the density-peak space is also preserved.

To keep an accurate local density distribution, for each object, its approximate local density should be similar to those of other objects with small distances. Our novel approach to approximate dynamic DPC achieves this by exploiting a proximity graph. This is motivated by the following reasons. First, w.r.t. both computational efficiency and accuracy in the approximate k-NN search problem, recent studies have confirmed that graph-based approaches outperform the other approaches [30], [31]. The local density computation is essentially a range search—one kind of similarity search problem—so a graph-based approach potentially yields high efficiency and accuracy for approximate local density computation. Second, given a query object, a graph-based approach finds its similar objects through greedy graph traversal. A proximity graph has edges between objects with small distances, so similar query objects tend to traverse similar parts of the graph. This results in similar approximate local densities for objects with small distances.

We next need an approach that can quickly update dependency relationships. Our idea for achieving this is to employ an optimistic approach. This idea is derived from two facts. (1) An update does not affect dependency relationships much, as in Example 2. (2) For each object p, its close object with higher local density is sufficient as its approximate dependent object. From Fig. 2(a), it is easy to see that, if the distance between an object and its approximate dependent object is sufficiently small, the clustering quality does not degrade. Then, it can be seen that, if p_i keeps $\hat{\rho}_i < \hat{\rho}_j$, where $\hat{\rho}$ represents the approximate local density of p and p_j is an approximate dependent object of p_i, we do not need to update (check) an approximate dependent object of p_i. In other words, only when we have $\hat{\rho}_i \geq \hat{\rho}_j$ because of an object insertion or deletion, we update the approximate dependent object of p_i. In this case, we use the fact that d_{cut} is small, to quickly update dependency relationships.

Overall procedure. Given an update (object insertion or deletion), our algorithm first updates approximate local densities with our graph index (Section V-A), and then updates the index (Section V-B). After that, it updates approximate dependent objects (Section V-C).

A. Local Density Update

A similarity search on a graph index is essentially done by a greedy algorithm. Note that nodes in a graph are objects in P. Given a query object p and a start node of the graph p_{start}, a greedy algorithm traverses the graph from p_{start}. This algorithm finds a node p' such that $dist(p, p_{start}) >$

$dist(p, p')$ and $(p_{start}, p') \in E(p_{start})$, where $E(p_{start})$ is a set of edges of p_{start}. If there exists p', it next explores the edges of p'. This iteration is repeated until no objects can update the result.

In graph-based similarity search algorithms, there are two main graph types: a small world network model [32], [33] and a monotonic path model [30], [34]. The advantage of the small world network model is that the path length between two arbitrary nodes is $O(\log n)$. That is, this model avoids long hops for graph traversal. On the other hand, the advantage of the monotonic path model is that, given two nodes p and p', there exists at least one path $p \rightarrow p_i \rightarrow \cdots \rightarrow p_{i+j} \rightarrow p'$ such that $dist(p, p') \geq dist(p_i, p') \geq \cdots \geq dist(p_{i+j}, p')$ (i.e., the distance to the destination node monotonically decreases). As a greedy algorithm traverses nodes so that the distance to query is closer, the monotonic path is useful for keeping high accuracy.

Data structure. Our new graph index is designed so that it has both the advantages. Specifically, our graph has the following properties:

- It has a fixed start node, which is the object generated first.
- The degree of this graph is $O(\log m)$, where $m = |P|$.
- It has a monotonic path from the start node to an arbitrary node.
- The expected length of the above monotonic path is $O(\log m)$.
- Apart from edges, each node p_i has a set S_i of objects p_j such that $dist(p_i, p_j) \leq \frac{d_{cut}}{2}$ (p_j belongs to only one node).

It is important to note that the nodes of this graph do not have to be in P_{active}. We do not delete nodes, even if the corresponding objects are deleted, to avoid expensive graph updates. The last property also helps to reduce the number of distance computations, since if a new object p and a node p_i have $dist(p, p_i) \leq \frac{d_{cut}}{2}$, all objects $p_j \in S_i$ have $dist(p, p_j) \leq d_{cut}$ from triangle inequality. The above properties derive a new output-sensitive approximate range search algorithm (Theorem 3).

How to dynamically build (update) this graph is presented in Section V-B. This section focuses on how to update an approximate local density of each object on this graph when we have an insertion or a deletion, under an assumption that our graph index always keeps the above properties.

The algorithm. We propose an approximate local density update algorithm that is based on a new approximate range search. This algorithm takes two phases: (1) ANN (approximate nearest neighbor) search phase and (2) Range search phase. In a nutshell, given an object p inserted/deleted, this algorithm first finds p_{ANN}, an ANN of p, by using the greedy algorithm in [32], then finds objects p' such that $dist(p, p') \leq d_{cut}$ in a breadth-first search (BFS) manner from p_{ANN}. During this, we compute approximate l-NN objects (nodes) of p among accessed objects (nodes) in an insertion case, where $l = O(\log m)$.

1. *ANN search phase*. This phase has two objectives: finding p_{ANN} of p to obtain a start node of BFS and obtaining a path from p_{start} to p_{ANN} to obtain a monotonic path to p. Let $E(p_i)$ be a set of edges held by p_i.

1) Given the start node $p_{start} = p_i$, we compute

$$p' = \underset{\{p_j \mid (p_i, p_j) \in E(p_i)\} \cup \{p_i\}}{\arg\min} dist(p, p_j). \quad (4)$$

2) If $p_i \neq p'$, we move to p', maintain a path (i.e., an ordered set $R = \{p_{start}, ...\}$), and repeat the same operation (by setting $p' = p_i$). Otherwise, $p_i = p_{ANN}$.

Note that, during this computation, if p_j, where $(p_i, p_j) \in E(p_i)$, has $dist(p, p_j) \leq 1.5 \cdot d_{cut}$, we insert p_j into a queue Q_{range}. Recall that each node p_x holds objects p_y such that $dist(p_x, p_y) \leq \frac{d_{cut}}{2}$. Therefore, for a node p_j where $dist(p, p_j) \leq 1.5 \cdot d_{cut}$, there may exist $p_y \in S_j$ such that $dist(p, p_y) \leq d_{cut}$, from triangle inequality.

2. *Range search phase*. If $dist(p, p_{ANN}) \leq 1.5 \cdot d_{cut}$, BFS is done from p_{ANN}.

1) We traverse our graph from p_{ANN} by using Q_{range}, and iff an accessed node p_i has $dist(p, p_i) \leq 1.5 \cdot d_{cut}$, p_i is inserted into Q_{range}. In addition, iff $p_i \in P_{active}$ and $dist(p, p_i) \leq d_{cut}$, $\hat{\rho}$ and $\hat{\rho}_i$ are updated. Also, each object $p_j \in S_i$ can have $dist(p, p_j) \leq d_{cut}$. We therefore insert $\langle p_i, dist(p, p_i) \rangle$ into a verification set V.
2) After the BFS terminates, we focus on each $p_j \in S_i$, where $\langle p_i, dist(p, p_i) \rangle \in V$. If $dist(p, p_i) \leq \frac{d_{cut}}{2}$, it is guaranteed that all $p_j \in S_i$ have $dist(p, p_j) \leq d_{cut}$. We hence update their approximate local densities without distance computation. Otherwise, for each $p_j \in S_i$, we compute $dist(p, p_j)$ and update $\hat{\rho}$ and $\hat{\rho}_j$ if necessary.

Note that, in an insertion case, we compute the object p_{max} with the highest approximate local density among the objects whose approximate local densities are updated.

Analysis. We analyze the time complexity of our approximate local density update algorithm.

THEOREM 3. *Let x be the exact local density of p with radius of $1.5 \cdot d_{cut}$. Furthermore, let h be the path length from p_{start} to p_{ANN}, i.e., $|R| = h$. Our local density update algorithm requires $O(h \log m + x)$ time.*

PROOF. The degree of our graph index is $O(\log m)$, so the ANN search phase needs $h \times O(\log m) = O(h \log m)$ time. The time of the range search phase is clearly $O(x)$, because it accesses objects p_i such that $dist(p, p_i) \leq 1.5 \cdot d_{cut}$. □

Because $dist(p, p_{ANN})$ is small, p tends to have the same answer in Equation (4) as p_{ANN}. This means that R is usually the monotonic path from p_{start} to p_{ANN}. Since, as we demonstrate later, our graph has the property that the expected length of the monotonic path is $O(\log m)$, the expected time of our local density update algorithm is $O(\log^2 m + x)$ in practice. In addition, because d_{cut} is also small in practice [6], we have $x \approx O(\hat{\rho})$. Hence, the performance of this algorithm empirically depends (almost) only on the approximate local density of a new or deleted object, and is much faster than $O(n)$ (since $x \ll n$).

B. Index Update

We present our index update algorithm that makes a monotonic path with an $O(\log m)$ length property.

The algorithm. Consider a deletion case of p. If p belongs to S_i, we simply remove p from S_i. If and only if p is a node, we do nothing, i.e., p is not deleted from the index. When we compute approximate local density, we check whether $p \in P_{active}$, thus this non-deletion is not an issue[3].

Next, consider an insertion case of p. If $dist(p, p_{ANN}) \leq \frac{d_{cut}}{2}$, p is inserted into S_{ANN}, and the index update is over. Otherwise, p becomes a node. Recall that our local density update algorithm obtains approximate l-NNs of p, where $l = O(\log m)$. We create undirected edges between p and its l-NNs. Next, we determine the level of p, denoted by $L(p)$. Given a random probability r,

$$L(p) = \min e \quad \text{s.t.} \quad \frac{1}{2^e} \leq r. \tag{5}$$

Our local density update algorithm also obtains the path $R = \{p_{start}, p_i, ..., p_{ANN}\}$. Let $p_j \in R$ be the nearest neighbor of p with one of the levels $\in [1, L(p)]$ in R. For each p_j, we create undirected edges between p and p_j.

Correctness and time complexity. We first present the monotonicity of R.

LEMMA 1. *The path $R = \{p_{start}, p_i, ..., p_{ANN}\}$ obtained by our local density update algorithm is a monotonic path.*

PROOF. Given a node p_i, Equation (4) proves that we take only a node p_j such that $dist(p, p_i) > dist(p, p_j)$, where p is a new object. It is thus ensured that $dist(p, p_{start}) > dist(p, p_i) > \cdots > dist(p, p_{ANN})$. □

As p_{ANN} is included in l-NNs of p, it is easy to see that p has a monotonic path from p_{start}. We next demonstrate that the expected length of this path is $O(\log m)$.

LEMMA 2. *The expected length of the monotonic path from p_{start} to p is $O(\log m)$.*

PROOF. Equation (5) and edge creation in R suggest that the monotonic path between the start node and a node constitutes a skip list [35]. Because its expected search complexity is $O(\log m)$ for a sequence of m objects, this lemma is clear. □

It is obvious that the index update time is at most $O(h)$, where $h = |R|$. As claimed before, $O(h) \approx O(\log m)$. In practice, we usually have $dist(p, p_{ANN}) \leq \frac{d_{cut}}{2}$, meaning that p does not become a node and belongs to S_{ANN}. Therefore, in many cases, this index update time is $O(1)$.

C. Dependent Object Update

Again, our idea for quick dependency relationships update is to employ an optimistic approach. Let $p_{i'}$ be an approximate dependent object of p_i before we have a new object p. Basically, if p_i still has $\hat{\rho}_i < \hat{\rho}_{i'}$ after the insertion of p, we do not update its approximate dependent object. Below, we present how to update dependency relationships, and use $\hat{q}_i$ to denote an approximate dependent object of p_i.

Insertion case. Assume that we have a new object p, and let $\hat{N}$ be a set of objects whose approximate local densities are updated due to the insertion of p. In this case, we update the approximate dependent object of $p_i \in \hat{N}$ such that $\hat{\rho}_i > \hat{\rho}_{i'}$. (The objects $\notin \hat{N}$ keep the same dependency relationships, because their local density do not change.) Recall that, we compute $p_{max} = \arg\max_{p \in \hat{N}_i} \hat{\rho}$ during local density update.

Case 1: $p_i \neq p_{max}$. We set $\hat{q}_i = p_{max}$. This is because, from triangle inequality, $dist(p_i, p_{max}) \leq 2 \cdot d_{cut}$, i.e., they are close (because d_{cut} is small). Also, if $p_j \in \hat{N}$ has $dist(p_j, p_{max}) < dist(p_j, p_{j'})$, we set $\hat{q}_j = p_{max}$.

Case 2: $p_i = p_{max}$ and $\hat{\rho}_i > \hat{\rho}_{i'}$. This case faces a computational challenge, because we do not know any objects that are close to p_{max} and have higher approximate local densities than $\hat{\rho}_{max}$. Observe that such p_{max} would have a higher approximate local density than objects existing around it. To find its approximate dependent object, it is desirable to access only objects with high approximate local densities.

To this end, for each insertion, we insert p_{max} having case 2 into a cache object set C (when p_{max} is deleted, it is removed from C). Notice that C contains objects with high approximate local densities in each space of P_{active}, and $|C| \leq n$ ($|C| \ll n$ in practice). If p_{max} has case 2, we compute the nearest neighbor object with higher local density than ρ_{max} in C. At the same time, if $p_j \in C$ has $\hat{\rho}_j < \rho_{max}$ and $dist(p_j, p_{max}) < dist(p_j, \hat{q}_j)$, we set $\hat{q}_j = p_{max}$.

THEOREM 4. *In an insertion case, our approximate dynamic DPC algorithm updates dependency relationships in $O(h \log m + x)$ time if p_{max} does not have case 2. Otherwise, it needs $O(n)$ time.*

PROOF. If p_{max} does not have case 2, we need $O(\hat{\rho})$ time to update dependency relationships, since $|\hat{N}| = \rho + 1$. Then, the time of this case is obvious from Theorem 3. If p_{max} has case 2, its dependent object update incurs $O(|C|)$ time. As $|C| \leq n$, this theorem holds. □

Deletion case. Assume that p is removed. In this deletion, we can have $p_i \notin \hat{N}$ and $p_{i'} \in \hat{N}$, where $p_{i'}$ is the previous approximate dependent object of p_i. In this case, denoted by case 3, we may have $\hat{\rho}_i \geq \hat{\rho}_{i'}$. If so, we have to update the dependent object of p_i. Otherwise, we have $\hat{\rho}_i < \hat{\rho}_{i'}$, so we do nothing.

LEMMA 3. *Except for case 3, we always have $\hat{\rho}_i < \hat{\rho}_{i'}$ in deletion cases.*

PROOF. Notice that we have $\hat{\rho}_i < \hat{\rho}_{i'}$ before p is removed. If $p_i \in \hat{N}$ and $p_{i'} \in \hat{N}$, both $\hat{\rho}_i$ and $\hat{\rho}_{i'}$ decrease due to the removal of p, so we still have $\hat{\rho}_i < \hat{\rho}_{i'}$. The case where $p_i \notin \hat{N}$ and $p_{i'} \notin \hat{N}$ is also the same result. Last, if $p_i \in \hat{N}$ and $p_{i'} \notin \hat{N}$, only $\hat{\rho}_i$ decreases, so we have $\hat{\rho}_i < \hat{\rho}_{i'}$. □

Case 3: $p_i \notin \hat{N}$ where $\hat{\rho}_i > \hat{\rho}_{i'}$. We retrieve an object p_j such that $\hat{\rho}_i < \hat{\rho}_j$ and $dist(p_i, p_j) \leq 2 \cdot d_{cut}$. Our approximate range search algorithm efficiently achieves this. The essential operations are the same as our local density update. The

[3] We experimentally confirmed that the number of cases of p being a node is $\ll n$ (e.g., less than 1% of million-scale updates).

differences are that (i) we do not update local densities, (ii) the radius is $2 \cdot d_{cut}$, and (iii) the search terminates when we find p_j.

THEOREM 5. *In a deletion case, our approximate dynamic DPC algorithm updates dependency relationships at most in $O((n-\hat{\rho})(h \log m + x'))$ time, where x' is the average number of objects returned by our approximate range search of p_i having case 3 with radius of $2 \cdot d_{cut}$.*

PROOF. Recall that c_3 is the number of objects having case 3 when p is deleted. From $c_3 \leq n - \hat{\rho}$ and Theorem 3, this theorem holds. □

Remark. Let c_3 be the number of objects having case 3 when p is deleted. It can be intuitively known that we practically have $c_3 \ll n - \hat{\rho}$. In addition, thanks to the early termination, the practical time of an approximate range search for p_i having case 3 is much less than $O(h \log m + x')$.

Recall Theorem 2, and any approximate dynamic DPC algorithms cannot provide theoretical guarantees w.r.t. clustering quality unfortunately. However, our algorithm yields a clustering result with high accuracy in practice, which is shown in the next section.

VI. EMPIRICAL STUDY

This section presents our experimental results. All experiments were conducted on a Ubuntu 18.04 LTS machine with 3.0GHz Intel Core i9-9900XE CPU and 128GB RAM.

Datasets. We used the following four real dynamic datasets[4].

- Gas: 4,208,261 18-D sensor-readings.
- Household: 2,049,280 7-D electric power consumption objects.
- Mirai: 764,137 115-D network traffic.
- PAMAP2: 2,844,868 51-D physical activity sensor objects.

Objects in the above datasets are sorted in generation order. We tested several distances, such as L_1 and L_2 distances, which are usually used in AI/ML applications [4], [14], [18], [36]. Because the results are similar, we report the results in the case of L_2 (Euclidean) distance, as with existing works considering metric spaces, e.g., [16], [36]–[40].

Algorithms. Our experiments evaluated

- Semi-static [20], [21]: A state-of-the-art exact algorithm introduced in Section III,
- EDMStream [19]: A state-of-the-art approximation algorithm (the inner parameter of EDMStream was set by following the original paper),
- AMD-DPC (Approximate, Metric, and Dynamic DPC): Our algorithm presented in Section V, and
- DISC [29]: A state-of-the-art density-based (DBSCAN-based) algorithm for dynamic data. Although, as shown in Fig. 6, the output of DISC is different from that of DPC, it is interesting to compare the scalability of AMD-DPC with that of this state-of-the-art. DISC was used for this comparison test.

The above algorithms were implemented in C++ and compiled by g++ 7.4.0 with -O3 flag (codes are available in a GitHub repository[5]). We do not consider the other density-based clustering algorithms, such as [23], [41]–[43], as competitors, because [19], [20] have already shown that DPC provides better clustering quality than them. Moreover, **these algorithms cannot deal with metric spaces**.

Workload. For each dataset, we generated a sequence of dataset update as a workload. Given an update, the probability that it is a deletion (insertion) is s $(1 - s)$. The insertion order follows the object generation order. When an update is a deletion, we deleted the oldest object in P_{active}. A workload terminates when all objects in a given dataset are inserted. By default, $s = 0.02$, since deletions are rare. The default values of d_{cut} for Gas, Household, Mirai, and PAMAP2 are respectively 100, 0.4, 5000, and 2.5, which were set according to the way in [6].

A. Comparison with State-of-the-art

Because **the peak memory of the evaluated algorithms was less than only 600MB**, we report their efficiency and accuracy.

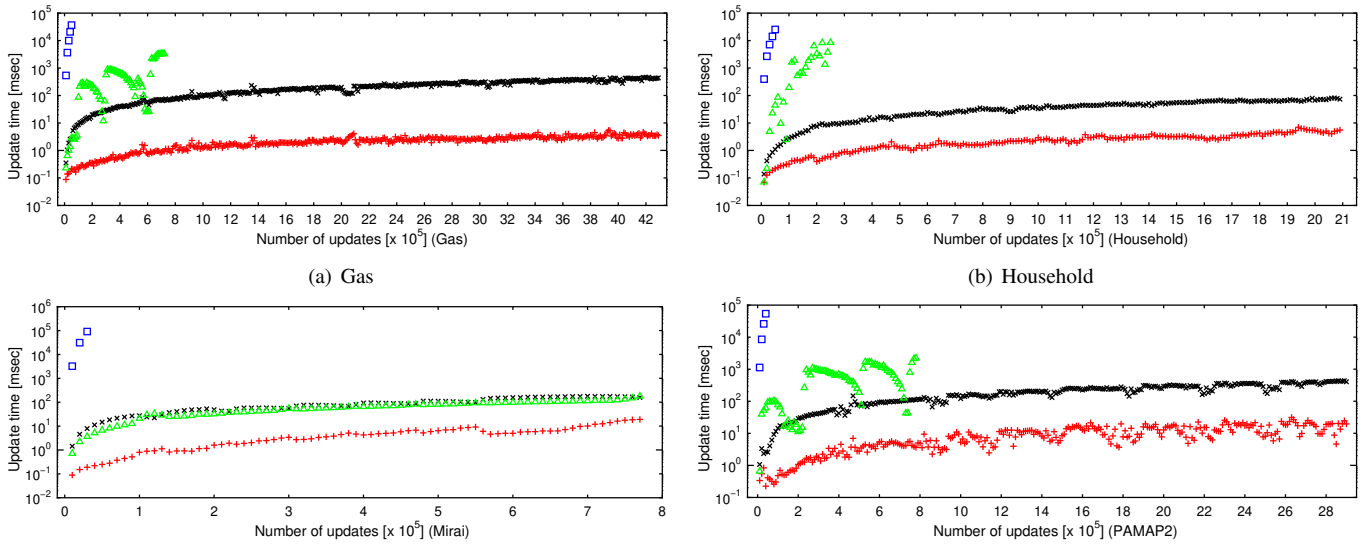
Efficiency. Fig. 3 plots the average update time of each algorithm every 10,000 updates. This figure shows how the update time of each algorithm varies as we have more updates, thus we can see the scalability of each algorithm.

The result shows that Semi-static is slow and cannot achieve real-time update even when n is small, as it needs $O(n^2)$ time for each update. (We stopped its experiments before it completed the workload, since its inefficiency is clear.) For example, on Gas, when we have 50,000 updates, the average update time of AMD-DPC is 169,201 times faster than that of Semi-static.

We turn our attention to approximate DPC algorithms (EDMStream and AMD-DPC). Fig. 3 confirms that AMD-DPC significantly outperforms EDMStream. For example, AMD-DPC completes the workload about 95, 16, 18, and 21 times faster than EDMStream on Gas, Household, Mirai, and PAMAP2, respectively. From Table II, which shows the decomposed time of EDMStream and AMD-DPC, we can see that the ρ comp. of AMD-DPC is much faster than that of EDMStream. Because x in Theorem 3 is usually small [6], this speed-up is supported theoretically. (The average index update times of AMD-DPC on Gas, Household, Mirai, and PAMAP2 are respectively 0.01, 0.01, 0.02, and 0.03 [msec], thus they are negligible.) On the other hand, EDMStream incurs $O(n)$ time to update local densities, and the cost of finding the nearest seed is not small. Table II clarifies that our optimistic approach to approximate dependent object update also yields high efficiency (while keeping high clustering accuracy, as shown later). As the workload progresses, the (approximate) local density of each object increases. The update time of AMD-DPC thereby increases when we have more updates, and its update time becomes comparatively long when we have

[4]https://archive.ics.uci.edu/ml/index.php

[5]https://github.com/9PSYAC3kjYB8/Dynamic-DPC



(a) Gas (b) Household (c) Mirai (d) PAMAP2

Fig. 3. Comparison with state-of-the-art. □, ×, +, and △ show Semi-static, EDMStream, AMD-DPC, and DISC, respectively

TABLE II
DECOMPOSED TIME [MSEC]

	Gas		Household		Mirai		PAMAP2	
Algorithm	ρ comp.	δ comp.	ρ comp.	δ comp.	ρ comp.	δ comp.	ρ comp.	δ comp.
EDMStream	104.32	105.61	12.98	24.96	78.16	12.27	141.15	59.30
AMD-DPC	0.40	1.79	0.78	1.66	1.16	3.89	0.50	9.00

TABLE III
RAND INDEX

	Gas	Household	Mirai	PAMAP2
EDMStream	0.88	0.52	0.65	0.69
AMD-DPC	0.90	0.97	0.99	0.94

TABLE IV
NMI

	Gas	Household	Mirai	PAMAP2
EDMStream	0.74	0.16	0.26	0.40
AMD-DPC	0.71	0.89	0.99	0.73

large local densities. This result suggests that its usual time complexity is $\tilde{O}(\hat{\rho})$.

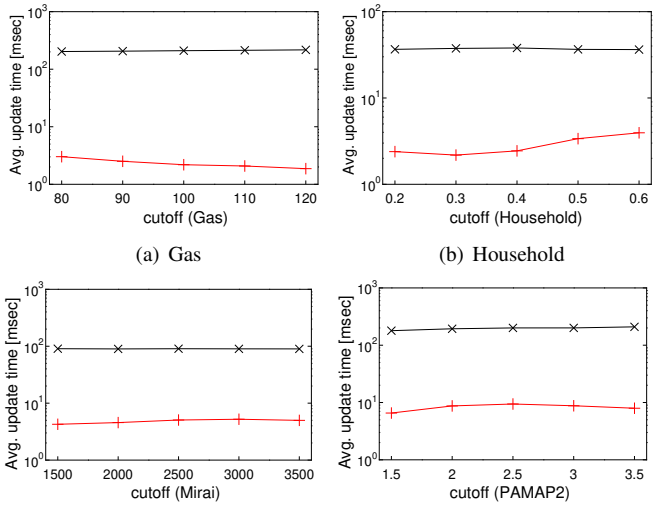
Last, we see that AMD-DPC is much more scalable than DISC, and DISC is too sensitive to the distribution of update points. (We terminated DISC on Gas, Household, and PAMAP2 before it completed the workload, because its scalability is clearly inferior to that of AMD-DPC.) In addition, DISC incurs $\Omega(n)$ time for a single update in metric spaces, so this result is reasonable. We do not consider DISC anymore, as it could not complete the workload for large datasets within a reasonable time.

Accuracy. We show Rand index and NMI of EDMStream and AMD-DPC in Tables III and IV, respectively. (As we early terminated DISC due to its inefficiency, we do not consider its accuracy.) The result clarifies that *AMD-DPC outperforms EDMStream in terms of both update time and accuracy* (except for NMI on Gas but still competitive). AMD-DPC yields a clustering result with high accuracy. This demonstrates that the optimistic approach in AMD-DPC effectively reduces the update time (see Table II) while avoiding clustering quality degradation. In addition, our new approximate range search also supports the high clustering accuracy of AMD-DPC. The average recall of approximate local density in AMD-DPC is 0.94, 0.97, 0.90, and 0.98 on Gas, Household, Mirai, and PAMAP2, respectively[6]. Our approximate range search provides high recall. The distribution of local densities is thus preserved, then the cluster centers of AMD-DPC are obtained in positions very near the exact ones.

B. Parameter Sensitivity

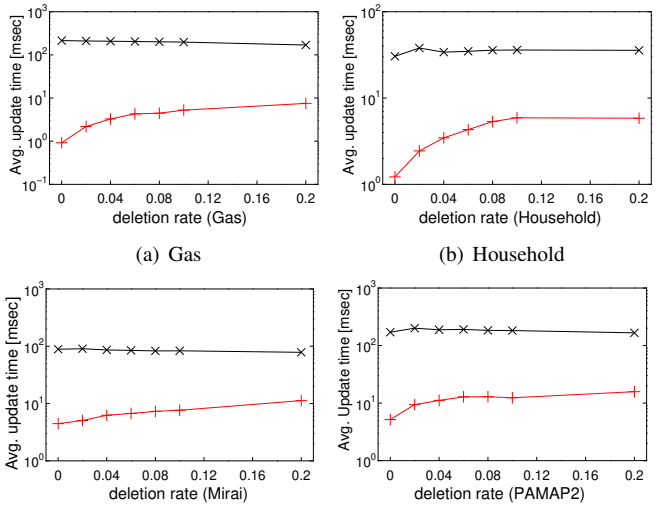
Varying d_{cut}. Fig. 4 reports how the cutoff distance d_{cut} affects the update time. (Due to space limitation, we omit accuracy, and it is consistent with the result in Tables III and IV.) Notice that the update time is the sum of ρ comp. and δ comp. times. (The update time of AMD-DPC includes its index time, but, as mentioned before, it is negligible.) When d_{cut} is larger, $\hat{\rho}$ of an object p becomes larger while the number of nodes becomes smaller, because p tends to find $dist(p, p_{ANN}) \leq \frac{d_{cut}}{2}$. Hence, we observed that the ρ comp. of AMD-DPC has a small influence of d_{cut}. The δ comp. time of AMD-DPC mainly depends on c_3, and we observed that a larger d_{cut} provides less (larger) c_3 on Gas (Household). However, the influence of d_{cut} is still small, and we see that AMD-DPC is robust to d_{cut}.

[6]EDMStream computes local densities only for seed (sampled) objects, so we cannot compute recall for EDMStream.



(a) Gas (b) Household (c) Mirai (d) PAMAP2

Fig. 4. Average update time vs. d_{cut}. $\times$ and $+$ respectively show EDMStream and AMD-DPC.



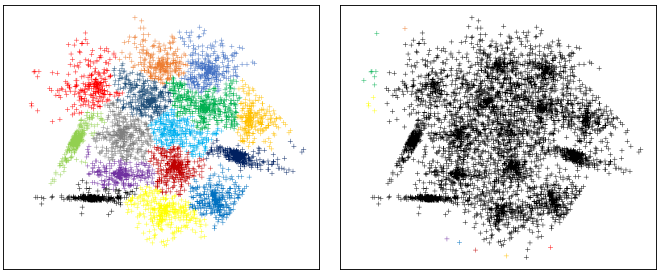
(a) Gas (b) Household (c) Mirai (d) PAMAP2

Fig. 5. Average update time vs. deletion rate. $\times$ and $+$ respectively show EDMStream and AMD-DPC.

Varying deletion rate. Finally, we investigate the impact of deletion rate s, which is presented in Fig. 5. The update time of AMD-DPC increases, as s becomes larger. The reason for this result is simple. When s is larger, AMD-DPC has more c_3, so its update time becomes longer. However, in practice, an update is essentially an insertion [15], [22] (e.g., streaming data). The performance of AMD-DPC matches such a practical case. Notice that, even if deletion rate is comparatively large, e.g., 20%, the update time of AMD-DPC is still small and much faster than that of EDMStream.

VII. Related Work

Density-peaks clustering. Section III has already introduced the state-of-the-art *dynamic* DPC algorithms, so we review other existing techniques for DPC. The DPC algorithm was developed in [6], and it has found many applications, as introduced in Section I. In addition, some works, e.g., [1], [14], [44], [45], proposed variants of DPC, but we follow the original definition for the dynamic DPC.



(a) DPC (b) DBSCAN

Fig. 6. Clustering result on S4. DPC successfully obtains the clusters, while DBSCAN does not work well.

Because [6] has not provided an algorithm to efficiently obtain the clustering result, some works have devised efficient algorithms. Zhang et al. [28] developed an LSH-based MapReduce algorithm. Rasool et al. [46] devised some heuristic pruning techniques that work on static datasets. [9] proposed some sub-quadratic algorithms for static DPC. These algorithms assume only the Euclidean space. Bai et al. proposed a pruning technique for efficiently computing local densities and dependent objects [47]. This technique, however, incurs $O(n^2)$ time.

Other density-based clustering. There are other density-based clustering algorithms for dynamic data [48]. For example, DenStream [41], D-Stream [42], MR-Stream [43], and DBSTREAM [23] are DBSCAN-based algorithms with ad-hoc definitions. They essentially assume (low-dimensional) Euclidean spaces, but our algorithm does not have this limitation. In addition, it has been confirmed in [19] that *EDMStream is faster than these algorithms*. DISC [29] can deal with metric spaces, but we demonstrated that AMD-DPC scales much better than DISC.

One main difference between DPC and DBSCAN is that DPC can identify a space having density-peak as one cluster. Focus on Fig. 1(a). DPC can catch two clusters (left and right bottom ones) although there are close objects between the clusters. This is useful for applications, because the two clusters are obviously different observations. On the other hand, DBSCAN essentially connects such close objects, thus may consider these two clusters as one cluster. To clarify this more, we use dataset S4 [49], which has 15 Gaussian clusters existing close each other. Fig. 6 depicts the clustering results of DPC and DBSCAN (we set their parameters so that we have 15 clusters via OPTICS [50]). DPC clearly identifies 15 clusters, whereas DBSCAN merges all clusters, i.e., fails to cluster S4 well because of its drawback described above.

VIII. Conclusion

Mining features and events from dynamically generated data is important. The effectiveness of density-peaks clustering (DPC) for this operation has been becoming well known. Motivated by this, this paper addressed the problem of DPC on dynamic data.

We first proved that the hardness of this problem: any exact solutions incur $\Omega(n)$ time for a single update (insertion or deletion). We next demonstrated that faster update time with a theoretical clustering quality guarantee cannot be achieved. Then we proposed an approximation algorithm that provides a faster update time and an empirically accurate clustering result. Our experiments on real datasets confirm that our algorithm is accurate and much faster than state-of-the-art.

REFERENCES

[1] S. Yang, X. Shen, and M. Chi, "Streamline density peak clustering for practical adoptions," in *CIKM*, 2019, pp. 49–58.

[2] J. Gan and Y. Tao, "Dbscan revisited: Mis-claim, un-fixability, and approximation," in *SIGMOD*, 2015, pp. 519–530.

[3] S. Alipour, "Approximation algorithms for probabilistic k-center clustering," in *ICDM*, 2020, pp. 1–11.

[4] A. Zubaroğlu and V. Atalay, "Data stream clustering: a review," *Artificial Intelligence Review*, vol. 54, no. 2, pp. 1201–1236, 2021.

[5] J. Jang and H. Jiang, "Dbscan++: Towards fast and scalable density clustering," in *ICML*, 2019, pp. 3019–3029.

[6] A. Rodriguez and A. Laio, "Clustering by fast search and find of density peaks," *Science*, vol. 344, no. 6191, pp. 1492–1496, 2014.

[7] T. H. Chan, A. Guerqin, and M. Sozio, "Fully dynamic k-center clustering," in *Web Conference*, 2018, pp. 579–587.

[8] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *KDD*, 1996, pp. 226–231.

[9] D. Amagata and T. Hara, "Fast density-peaks clustering: Multicore-based parallelization approach," in *SIGMOD*, 2021, pp. 49–61.

[10] H.-P. Kriegel, P. Kröger, J. Sander, and A. Zimek, "Density-based clustering," *Data Mining and Knowledge Discovery*, vol. 1, no. 3, pp. 231–240, 2011.

[11] R. Hu, W. Li, O. V. Kaick, H. Huang, M. Averkiou, D. Cohen-Or, and H. Zhang, "Co-locating style-defining elements on 3d shapes," *ACM Transactions on Graphics*, vol. 36, no. 3, p. 33, 2017.

[12] R. Mehmood, S. El-Ashram, R. Bie, H. Dawood, and A. Kos, "Clustering by fast search and merge of local density peaks for gene expression microarray data," *Scientific Reports*, vol. 7, p. 45602, 2017.

[13] N. Begum, L. Ulanova, J. Wang, and E. Keogh, "Accelerating dynamic time warping clustering with a novel admissible pruning strategy," in *KDD*, 2015, pp. 49–58.

[14] G. Y.-Y. Chan, F. Du, R. A. Rossi, A. B. Rao, E. Koh, C. T. Silva, and J. Freire, "Real-time clustering for large sparse online visitor data," in *Web Conference*, 2020, pp. 1049–1059.

[15] D. Amagata, T. Hara, and C. Xiao, "Dynamic set knn self-join," in *ICDE*, 2019, pp. 818–829.

[16] V. Cohen-Addad, N. O. D. Hjuler, N. Parotsidis, D. Saulpic, and C. Schwiegelshohn, "Fully dynamic consistent facility location," *NeurIPS*, vol. 32, pp. 3255–3265, 2019.

[17] J. Gan and Y. Tao, "Dynamic density based clustering," in *SIGMOD*, 2017, pp. 1493–1507.

[18] S. Mai, J. Jacobsen, S. Amer-Yahia, I. Spence, P. Tran, I. Assent, and Q. V. H. Nguyen, "Incremental density-based clustering on multicore processors," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[19] S. Gong, Y. Zhang, and G. Yu, "Clustering stream data by exploring the evolution of density mountain," *PVLDB*, vol. 11, no. 4, pp. 393–405, 2017.

[20] L. Ulanova, N. Begum, M. Shokoohi-Yekta, and E. Keogh, "Clustering in the face of fast changing streams," in *SDM*, 2016, pp. 1–9.

[21] Q. Zhang, C. Zhu, L. T. Yang, Z. Chen, L. Zhao, and P. Li, "An incremental cfs algorithm for clustering large data in industrial internet of things," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 3, pp. 1193–1201, 2017.

[22] Y. Zhang, K. Tangwongsan, and S. Tirthapura, "Streaming k-means clustering with fast queries," in *ICDE*, 2017, pp. 449–460.

[23] M. Hahsler and M. Bolaños, "Clustering data streams based on shared density between micro-clusters," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 6, pp. 1449–1461, 2016.

[24] R. Krauthgamer and J. R. Lee, "Navigating nets: Simple algorithms for proximity search," in *SODA*, 2004, pp. 798–807.

[25] J. Gan and Y. Tao, "Fast euclidean optics with bounded precision in low dimensional space," in *SIGMOD*, 2018, pp. 1067–1082.

[26] A. Lulli, M. Dell'Amico, P. Michiardi, and L. Ricci, "Ng-dbscan: Scalable density-based clustering for arbitrary data," *PVLDB*, vol. 10, no. 3, pp. 157–168, 2016.

[27] Z. Wang, R. Zhang, J. Qi, and B. Yuan, "Dbsvec: Density-based clustering using support vector expansion," in *ICDE*, 2019, pp. 280–291.

[28] Y. Zhang, S. Chen, and G. Yu, "Efficient distributed density peaks for clustering large data sets in mapreduce," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 12, pp. 3218–3230, 2016.

[29] B. Kim, K. Koo, J. Kim, and B. Moon, "Disc: Density-based incremental clustering by striding over streaming data," in *ICDE*, 2021, pp. 828–839.

[30] C. Fu, C. Xiang, C. Wang, and D. Cai, "Fast approximate nearest neighbor search with the navigating spreading-out graph," *PVLDB*, vol. 12, no. 5, pp. 461–474, 2019.

[31] W. Li, Y. Zhang, Y. Sun, W. Wang, M. Li, W. Zhang, and X. Lin, "Approximate nearest neighbor search on high dimensional data-experiments, analyses, and improvement," *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[32] Y. Malkov, A. Ponomarenko, A. Logvinov, and V. Krylov, "Approximate nearest neighbor algorithm based on navigable small world graphs," *Information Systems*, vol. 45, pp. 61–68, 2014.

[33] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 4, pp. 824–836, 2020.

[34] B. Harwood and T. Drummond, "Fanng: Fast approximate nearest neighbour graphs," in *CVPR*, 2016, pp. 5713–5722.

[35] W. Pugh, "Skip lists: A probabilistic alternative to balanced trees," *Communications of the ACM*, vol. 33, no. 6, pp. 668–676, 1990.

[36] H. Ding, F. Yang, and M. Wang, "On metric DBSCAN with low doubling dimension," in *IJCAI*, 2020, pp. 3080–3086.

[37] M. Borassi, A. Epasto, S. Lattanzi, S. Vassilvitskii, and M. Zadimoghaddam, "Sliding window algorithms for k-clustering problems," in *NeurIPS*, 2020.

[38] M. Ceccarello, A. Pietracaprina, and G. Pucci, "Solving k-center clustering (with outliers) in mapreduce and streaming, almost as accurately as sequentially," *PVLDB*, vol. 12, no. 7, pp. 766–778, 2019.

[39] S. Song, F. Gao, R. Huang, and Y. Wang, "On saving outliers for better clustering over noisy data," in *SIGMOD*, 2021, pp. 1692–1704.

[40] Y. Zeng, Y. Tong, and L. Chen, "Hst+: An efficient index for embedding arbitrary metric spaces," in *ICDE*, 2021, pp. 648–659.

[41] F. Cao, M. Estert, W. Qian, and A. Zhou, "Density-based clustering over an evolving data stream with noise," in *SDM*, 2006, pp. 328–339.

[42] Y. Chen and L. Tu, "Density-based clustering for real-time stream data," in *KDD*, 2007, pp. 133–142.

[43] L. Wan, W. K. Ng, X. H. Dang, P. S. Yu, and K. Zhang, "Density-based clustering of data streams at multiple resolutions," *ACM Transactions on Knowledge Discovery from Data*, vol. 3, no. 3, pp. 1–28, 2009.

[44] Y. Chen, X. Hu, W. Fan, L. Shen, Z. Zhang, X. Liu, J. Du, H. Li, Y. Chen, and H. Li, "Fast density peak clustering for large scale data based on knn," *Knowledge-Based Systems*, vol. 187, p. 104824, 2020.

[45] G. Wang and Q. Song, "Automatic clustering via outward statistical testing on density metrics," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 8, pp. 1971–1985, 2016.

[46] Z. Rasool, R. Zhou, L. Chen, C. Liu, and J. Xu, "Index-based solutions for efficient density peak clustering," *TKDE*, vol. 34, no. 5, pp. 2212–2226, 2022.

[47] L. Bai, X. Cheng, J. Liang, H. Shen, and Y. Guo, "Fast density clustering strategies based on the k-means algorithm," *Pattern Recognition*, vol. 71, pp. 375–386, 2017.

[48] J. A. Silva, E. R. Faria, R. C. Barros, E. R. Hruschka, A. C. d. Carvalho, and J. Gama, "Data stream clustering: A survey," *ACM Computing Surveys*, vol. 46, no. 1, pp. 1–31, 2013.

[49] P. Fränti and O. Virmajoki, "Iterative shrinking method for clustering problems," *Pattern Recognition*, vol. 39, no. 5, pp. 761–765, 2006.

[50] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "Optics: Ordering points to identify the clustering structure," in *SIGMOD Record*, vol. 28, no. 2, 1999, pp. 49–60.