A Fast and More Accurate Seed-and-Extension Density-based Clustering Algorithm

Ming-Hao Tung, Yi-Ping Phoebe Chen *Senior Member, IEEE,*
Chen-Yu Liu and Chung-Shou Liao *Senior Member, IEEE*

Abstract—Clustering algorithms have been widely studied in many scientific areas, such as data mining, knowledge discovery, bioinformatics and machine learning. A density-based clustering algorithm, called density peaks (DP), which was proposed by Rodriguez and Laio, outperforms almost all other approaches. Although the DP algorithm performs well in many cases, there is still room for improvement in the precision of its output clusters as well as the quality of the selected centers. In this study, we propose a more accurate clustering algorithm, seed-and-extension-based density peaks (SDP). SDP selects the centers that hold the features of their clusters while building a spanning forest, and meanwhile, constructs the output clusters in a seed-and-extension manner. Experiment results demonstrate the effectiveness of SDP, especially when dealing with clusters with relatively high densities. Precisely, we show that SDP is more accurate than the DP algorithm as well as other state-of-the-art clustering approaches concerning the quality of both output clusters and cluster centers while maintaining similar running time of the DP algorithm, particularly for a variety of time-series data. Moreover, SDP outperforms DP in the dynamic model in which data point insertion and deletion are allowed. From a practical perspective, the proposed SDP algorithm is obviously helpful to many application problems.

Index Terms—Clustering, center selection, density peaks, seed-and-extension, spanning tree

1 Introduction

There are many types of clustering algorithms such as partitioning, hierarchical (i.e. distance-based clustering) and density-based clustering algorithms [18], [24], [13], [5], [11], [12], [10], [9], [28], [15]. Partitioning algorithms such as K-means [18], [24] and K-medoid [13] typically start with an initial partition and then iteratively move the cluster centers to optimize an objective function, which is usually the sum of the squared error. Density-based algorithms such as DBSCAN [5] and its variants [11], [10], [9], [12] distinguish data points as Core, Border and Noise based on the density of each point, and the algorithms group those that are sufficiently close. The clusters of density-based algorithms are defined by areas in which the density of the data points is high, and clusters are separated from each other by areas of low density. Readers can be referred to recent surveys [26], [17], [29].

The recently proposed density peaks algorithm (DP) presents a new clustering approach [27]. DP merges both characteristics of partitioning and density-based clustering algorithms. First, DP selects cluster centers in a similar way to K-medoids and finds the density peaks of an input dataset as centers. Next, DP identifies clusters by linking neighboring dense regions to form clusters like DBSCAN, which makes DP capable of finding the clusters with arbitrary shapes. Some recent work [4], [8] claimed that their improved clustering algorithms outperforms DP (e.g., using K-nearest neighbors under the framework of DP), but DP still has better performance in most cases. In addition, DP has other advantages: for instance, DP is a deterministic algorithm; i.e., DP always gets the same output clusters. However, there is room for improvement over the precision of the output clusters and the quality of the selected centers of DP. Precisely, datasets in which there are many nearby clusters with high densities may make DP fail. More details are presented later in Section 2.2.

In this study, we design a clustering algorithm, called SDP (seed-and-extension-based density peaks), which is built on a similar framework to the DP algorithm. As mentioned, there exist some cases in which DP fails to correctly find all clusters. We overcome the challenges using a seed-and-extension center selection approach. Briefly, similar to the idea of DP, we pick the centers as seeds that are capable of representing the property of input datasets, and we extend from the seeds to obtain their corresponding clusters. Note that both our SDP algorithm and DP perform through three major procedures, i.e. calculating two quantities, selecting cluster centers based on the above two quantities, and assigning the input data points to each cluster.

We first revisit the DP algorithm which, in addition to the all-pairs distance matrix, requires two input parameters: cutoff distance d_c and the number of clusters k. Next, for each data point i, we compute two quantities: local density ρ_i and the minimum distance between point i and any other points with a higher density, denoted by δ_i. Rodriguez and Laio [27] formally defined the two quantities as follows:

Definition 1. *The local density ρ_i of a data point i is defined to be the number of data points within a given cutoff distance d_c to the point i.*

$$\rho_i = |\{j \mid d_{ij} < d_c\}|$$

Definition 2. *Each data point i, except the one with the highest density, has its Closest data point with a Higher Density (CHD), where the distance between the point i and its CHD is denoted by δ_i.*

$$\delta_i = \begin{cases} \max\limits_{j}\{d_{ij}\}, & \text{if} \quad \rho_i = \max\limits_{k}\{\rho_k\}; \\ \min\limits_{j:\rho_j > \rho_i}\{d_{ij}\}, & \text{otherwise}. \end{cases}$$

Note that DP is performed based on two assumptions: (i) cluster centers are surrounded by points with a lower density, and (ii) centers are at a relatively large distance from each other. Using two quantities ρ and δ, DP can provide a two-dimensional graph, called a *decision graph*, to gain new insights into the data distribution and intuitively determine cluster centers [27].

TABLE 1
Three Subroutines of the DP and SDP algorithms

Subroutine 1: Calculate ρ and δ **Input:** $Data$, the input dataset; d_c, the cutoff distance **Output:** Two quantities ρ and δ
Subroutine 2: Select cluster centers **Input:** Number of clusters k; Two quantities ρ and δ **Output:** Cluster centers
Subroutine 3: Cluster assignment **Input:** Cluster centers; Closest data points with a higher density **Output:** All data points with their cluster assignment label

Briefly speaking, DP just compute the γ list of all possible combinations of ρ and δ, where $\gamma_i = \rho_i \times \delta_i$. Following the descending order of the γ list, the top k data points (i.e. the points with the highest k values) can be heuristically selected to be cluster centers (see Algorithm 1). In the procedure of assigning data points (Subroutine 3), each remaining data point i is assigned to the cluster in which i's closest data point with a higher density (denoted by CHD_i) lies. (More details of DP can be referred to Fig.S1 in the Supplementary.)

Algorithm 1: Cluster Center Selection of DP

Input: k, the number of clusters; ρ, a local density vector for each point in the dataset; δ, the distance between each point and its CHD point
Output: *centers*, a list of the points as cluster centers

```
array(centers) = ∅;
γ_i = ρ_i × δ_i for every point i;
γ_sorted = Sort(γ, descend);
for i = 1 to k do
    push γ_sorted(i) into array(centers);
end for
```

The key difference between DP and our SDP algorithm lies in the procedure of center selection (i.e. Subroutine 2). More precisely, we build a spanning forest step by step based on the two quantities ρ and δ (derived in Subroutine 1), and preserve the connectivity of each spanning tree as far as possible. Meanwhile, we select cluster centers as seeds along the tree edges in a top-down manner. Next, each cluster grows up from its seed using a seed-and-extension method. That is, we actually perform Subroutines 2 and 3 simultaneously while constructing the spanning trees, and repeat the steps until all data points are assigned. There are three advantages of the SDP clustering algorithm: first, the connectivity preservation (along tree edges) ensures the quality of the derived clusters. Second, the spanning forest structure speeds up update operations in the dynamic model. Third, the seed-and-extension approach which is running during the procedure of center selection can avoid too many centers with high density but close to each other. Later, we demonstrate SDP's superior performance against the DP algorithm, especially considering time-series input datasets. Moreover, in comparison with the cluster assignment procedure of DP (i.e. Subroutine 3), we apply two different types of assignment approaches: distance-based assignment and density-based assignment. The former, similar to K-medoids [13], assigns data points to their nearest centers, and the latter, similar to DP, assigns each data point to the cluster in which the point's closest neighbor with a higher density lies.

Based on the design of our SDP algorithm, the experiment results show that the cluster centers selected by SDP are of better quality than DP and other state-of-the-art clustering algorithms. Furthermore, the density-based assignment, just like DP, has the ability to cluster input datasets with non-spherical shapes. On the other hand, the distance-based assignment can assign input points according to their distances to the centers, which can form a *Voronoi diagram* in the *geometric* plane. Based on the above fact, we then choose one of the two assignment strategies with respect to the properties of input datasets.

A summary of the main contributions in this paper is as follows:

1) We design a new center selection strategy, i.e. the seed-and-extension-based density peaks (SDP) algorithm which performs more accurately while maintaining similar computational time.
2) We determine the condition under which the DP algorithm fails to find correct clusters.
3) We compare SDP against well-known clustering algorithms: K-means, Agglomerative Hierarchical, DBSCAN, DP and two state-of-the-art clustering algorithms: SPECTACL [9] and DENCLUE 2.0 [10], to demonstrate its superior performance, especially for a variety of input datasets.
4) We design the dynamic version of SDP and demonstrate its merits in the dynamic model from both theoretical and practical perspectives.

This paper is organized as follows: in Section 2, we introduce SDP, and then conduct numerical experiments and evaluate the performance of our algorithm in Section 3. Then we extend SDP to deal with time-series data in Section 4. We also present dynamic SDP and highlight its merits in Section 5. Finally, we conclude with some discussions.

2 SDP: Seed-and-extension Density Peaks

Recall the key idea of DP: cluster centers are the points that are surrounded by points with a lower density, and the distance between every two centers is relatively large. As mentioned, there are also three main subroutines in our SDP algorithm: calculating the quantities ρ and δ, center selection and cluster assignment. Briefly, the first subroutine is similar, and the second step is to find potential center candidates from the γ list (based on the two quantities ρ and δ) and locally select cluster centers along a spanning forest structure, where two nodes are connected in a tree if one node is the other's CHD. In the meantime, we extend from the centers as seeds to derive clusters. In the following, we introduce the details of our SDP algorithm and its difference from DP.

2.1 Algorithms

The major difference between DP and SDP is the procedure of center selection. DP picks the top k points from the sorted γ list (in descending order) as centers, where $\gamma_i = \rho_i \times \delta_i$ for each point i. In contrast, our SDP algorithm constructs a spanning forest for center selection as well as the *seed-and-extension* procedure as follows.

2.1.1 Construct spanning trees for center selection

We derive the γ list in Subroutine 1, after calculating ρ_i and δ_i of each data point i. We also obtain CHD_i for each point i, where the distance between i and CHD_i is δ_i. Initially, we select the point with the highest density as the root, say O. Then, we link every point j to O if O is the CHD of j. Precisely, there is a directed edge from j to i in the tree if and only if i is the CHD of j. We thus construct a spanning tree level by level and start the *seed-and-extension* procedure from O, i.e. the first seed. Here the root O is at level 0 and every point whose CHD is O is O's child at level 1 in the tree. Then we link the remaining points in a breadth-first-search manner if their CHD are O's children, and repeat the procedure. Therefore, the cluster grows from O while constructing the tree (see Algorithm 2).

In addition, we devise two types of thresholds when performing the procedure. First, we construct the spanning tree within a given region, i.e. considering only the points under a threshold distance from the root O. We usually set the threshold to be the given *cutoff distance* or a smaller value sometimes. Next we design the second threshold to restrict the construction of a spanning tree. That is, when connecting a point i with its parent, i.e. CHD_i, we eliminate the directed edge if the edge distance, i.e. δ_i is larger than the threshold. Here we let the threshold to be a significantly larger value than the average distance between CHD_i and its children. Precisely, we usually set the threshold to be the mean of the distance between every point and its CHD plus the two standard deviations. (In this study, we use the usual setting of the two thresholds to conduct all the experiments.) The cluster thus keeps growing until all the incoming edges are larger than the second threshold or all the points within the given region have been already considered. Subsequently, we consider the remaining points in the γ list and pick the next center candidate in descending order. We repeat the *seed-and-extension* procedure from the center as a seed. Note that the whole process actually generates a spanning forest due to the setting of the two thresholds; that is, each component which forms a cluster is actually a tree in the spanning forest.

Based on the seed-and-extension procedure, our SDP algorithm performs more accurately than the DP algorithm. The reason why SDP can guarantee a better performance is because the algorithm considers not only the global property (i.e. ρ and δ in the decision graph) of input datasets but also the local connectivity between every point and its CHD. Each spanning tree we constructed preserves the connection between the points within the corresponding cluster. Moreover, the seed-and-extension approach which incorporates cluster assignment into the procedure of center selection can particularly avoid SDP to pick many nearby centers with high densities. The SDP algorithm thus outperforms DP as well as other well-known clustering algorithms, especially when an input instance contains many points with relatively high densities.

Algorithm 2: Cluster Center Selection of SDP

Input: *Data*, the input clustering dataset; k, the number of clusters; γ_{sorted}, the vector in descending order for center candidates; *CHD*, the closest data point with a higher density; δ, the distance to CHD

Output: *centers*, a list of the points as cluster centers; *Assignment*, cluster labels for each data point

1: **while** $cluster\_label \leq k$ and $j \leq$ size of *Data* **do**
2: **if** *Assignment(j)* is already done **then**
3: j++;
4: **continue**
5: **else**
6: *center(*$cluster\_label$*)* = $\gamma_{sorted}(j)$;
7: *withinnode* = find($d_{center(cluster\_label),:} \leq d_c$);
8: *withinnode* =
9: intersect(*withinnode*, find(*CHD=center*(i)));
10: **while** *withinnode* is not empty **do**
11: *withinnode* =
12: *withinnode* $\setminus\{$ Outlier of $\delta_{\text{withinnode}}\}$;
13: *Assignment(withinnode)* = $cluster\_label$;
14: *next* = find($d_{withinnode,:} \leq d_c$);
15: *next* = intersect(*next*, find(*CHD=withinnode*));
16: *withinnode* = *next*;
17: **end while**
18: $cluster\_label$++;
19: j++;
20: **end if**
21: **end while**

2.1.2 A concrete example

We use an instance, as shown in Figure 1, to illustrate the steps of Algorithm 2. In line 6, we first pick the next unassigned (or unselected) data point in the γ list in descending order as the center of the current cluster (as a seed), e.g., initially root O in Figure 1. In this example, suppose node O has the highest local density ρ_O, i.e. an *absolute density peak*. We thus let O be the root of a spanning tree and extend the cluster from the seed while building the spanning tree. Note that the spanning tree is constructed within a region, called *local zone*, bounded by the first threshold distance from the root (i.e. seed), as mentioned earlier. Here, we let the given cutoff distance, d_c be the threshold and consider only the points within d_c from O (see line 7).

Next, we use the breadth-first-search approach to construct the spanning tree level by level, as shown in lines 8-17. Precisely, assume the local density of points A, B, C and D are presented in descending order as $\rho_D > \rho_A > \rho_B > \rho_C$; besides, assume O is CHD of A, B, C and D. Hence, O is the root node at level 0, and squares A, B, C and D presents the tree nodes at the first level of the spanning tree, where the value of δ_i for these nodes are a, b, c and d, respectively. We repeat the construction, and the tree nodes at the second level are represented by triangles (i.e. nodes E, F, etc.).

Notice that in lines 11-12, we use the second threshold to restrict the construction of the spanning tree. That is, we ignore point i if δ_i is larger than the threshold

(i.e. i is an *outlier*), defined by either Interquartile Range (IQR) [14] or the mean plus two standard deviations. (Here both the definitions of an outlier actually perform well in our experiment.) In this example, we therefore remove the directed edge from C to O (i.e. C is an outlier), as well as the triangles E and F. Then, the seed-and-extension procedure continues until all the points within the *local zone* are considered. Subsequently, we select the next data point in the sorted γ list to be a center (i.e. seed) and repeat the above procedure to construct another spanning tree.

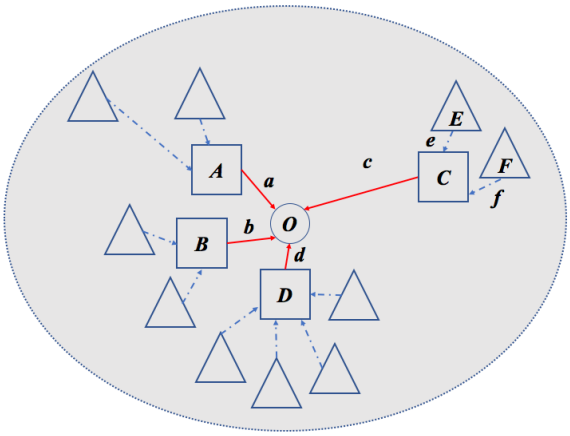


Fig. 1. An example illustrating how a spanning tree is constructed by the relationship between point i and its CHD_i, i.e. the seed-and-extension procedure, where the tree nodes in the first level are represented by squares and the nodes in the second level are represented by triangles

2.1.3 Cluster assignment

The rationale behind our SDP algorithm is a seed-and-extension strategy for center selection and cluster assignment. Furthermore, after performing the seed-and-extension procedure (i.e. linking a tree node with its children), we consider the remaining points that are not in the local zone of any cluster centers. We actually use two types of assignment approaches: density-based (denoted by SDP_{den}) assignment and distance-based (denoted by SDP_{dist}) assignment, according to the required property of output clusters. For the density-based assignment (SDP_{den}), as shown in Algorithm 3, each remaining data point is assigned to its CHD's cluster in a similar way to DP. That is, every tree edge is defined by the connection between point i and its CHD_i, as conducted by Algorithm 2. Therefore, the SDP_{den} algorithm also has the ability to cluster datasets with non-spherical shapes. Note that the procedure sometimes results in a *bad* cluster whose size might be too big; in other words, some points in the cluster might belong to other clusters. On the other hand, the distance-based assignment (SDP_{dist}), which greedily assigns each remaining point to its closest center (see lines 4-11 in Algorithm 4), sometimes becomes a better choice for finding clusters with similar properties. Note that, for the detection of outliers (i.e. the setting of the second threshold), we can simply use either the *cluster halo* provided by the DP algorithm [27] or a threshold distance to the cluster centers. We describe the details of the two assignment approaches in Algorithms 3 and 4, respectively.

Algorithm 3: SDP_{den}: Density-based Cluster Assignment

Input: *Data*, the input clustering dataset; *centers*, list of the points as cluster centers; ρ, local density vector for all points in dataset; δ, the distance to the CHD points; *CHD*s, the CHD points
Output: *Assignment*, cluster labels for each data point

```
1: [ρ_sorted, sortedIndex] = Sort(ρ, descend);
2: for i = 1 to the size of Data do
3:     if Assignment(sortedIndex(i)) is not done then
4:         Assign sortedIndex(i) to the cluster in which
5:         its CHD point lies;
6:     end if
7: end for
```

Algorithm 4: SDP_{dist}: Distance-based Cluster Assignment

Input: *Data*, the input clustering dataset; *centers*, a list of the points as cluster centers;
Output: : *Assignment*, cluster labels for each data point

```
1:
2: for i = 1 to the size of Data do
3:     if Data(i) is not assigned then
4:         bsf = inf;
5:         for j = 1 to the number of centers do
6:             if d_{i,centers(j)} < bsf then
7:                 bsf = d_{i,center(j)};
8:                 bsfcenter = center(j);
9:             end if
10:        end for
11:        Assignment(i) = Assignment(center(j));
12:    end if
13: end for
```

2.2 A worst-case instance for the DP algorithm

The center selection process of the DP algorithm considers only the descending order of the γ list, which might cause the distance between some pairs of the selected centers to be too small. Precisely, the worst scenario is illustrated as follows: let the number of clusters be k and the cluster centers be the top k points picked from the sorted γ list. Because every γ_i is the product of ρ_i and δ_i, the data point may be selected to be a center with a large value of ρ_i but a small value of δ_i. More intuitively, these centers chosen by DP may be too close to each other.

Figure 2 shows a worst-case numerical example, where the 50 Words dataset, retrieved from [2], was used to test DP and SDP. The 50 Words dataset is time-series data, which can reveal a significant difference between the two algorithms. We thus exploited the time series version of the DP algorithm, called TADPole [1], for the test dataset, where the input similarity distance of every pair of points is calculated by dynamic time warping (DTW) [1]. Figure 2 shows the partial clustering result of the 50 Words dataset. There are 18 big pink diamonds (left) and only three big blue squares (right), each of which represents a cluster center, obtained by DP and our SDP algorithm, respectively. Here the diamonds and the squares correspond to the *correct* points with the Class 1 label provided by the UCR Archive[2]. As shown in Figure 2 (left), the diamonds are split into many different clusters; in contrast, Figure 2 (right) shows the clustering

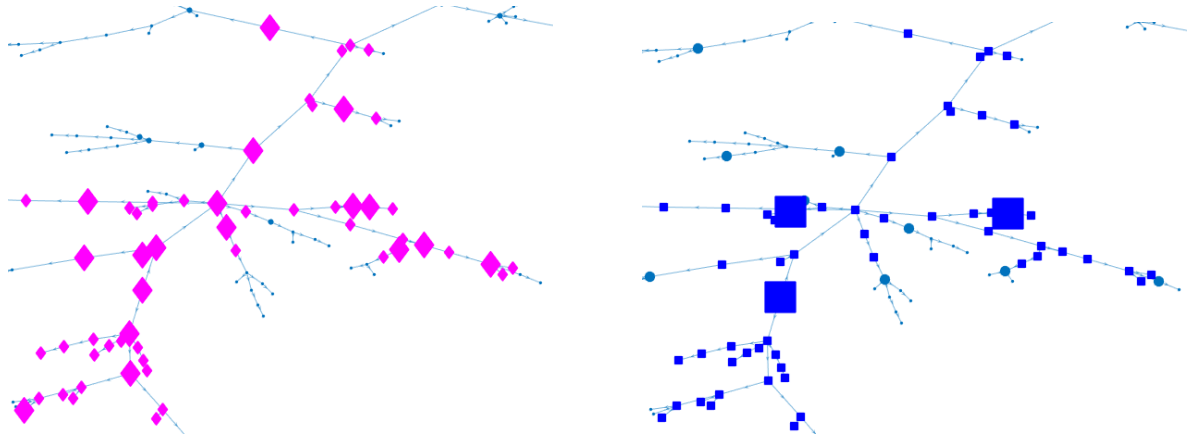


Fig. 2. (Left) The partial clustering result of the 50 Words dataset derived by the density peaks (DP) algorithm, where the 18 big nodes (diamonds in pink) represent the cluster centers; (Right) The result obtained by SDP, where there are only three big nodes (squares in blue), representing the cluster centers

result derived by SDP, which accurately partitions the points with the Class 1 label into only three clusters.

The key reason why our seed-and-extension strategy performs more accurately than DP is that while we hold similar assumptions about DP by applying the *local zone* to each cluster, i.e., using the first threshold distance (e.g., the given cutoff distance), which can preserve the connectivity within each cluster, we also ensure that the center of each cluster is not too close to each other. Precisely, the distance between the selected centers is at least larger than the second threshold distance. That is, we consider the points with an extremely high density to be noises and skip them if they are not at a relatively large distance.

3 Experimental Evaluation

In this section, we first demonstrate the effectiveness of our new center selection strategy regarding cluster performance and *center quality*. Here we provide a new performance measure, called *center quality* whose details are introduced in Section 3.3. Moreover, we show that SDP overcomes the weakness of DP when dealing with datasets that have significantly different densities and similarity distances between each data point in a non-metric space (e.g. DTW for time series data). In Section 3.1, we describe the datasets, the experiment settings and cluster performance measures. We compare two versions of our SDP algorithm: distance-based (SDP_{dist}) and density-based (SDP_{den}) assignment with density peaks (DP) and other state-of-the-art algorithms in Section 3.2. Note that we attempted to apply kernel density estimation using Guassian kernel [10] to define the local density of a point instead of the traditional counting-based approach (Definition 1), but it does not perform better. Finally, we introduce the new measure, center quality, and have more observations in Section 3.3.

3.1 Experiment setup

Machine Configuration. The experiments were conducted on a machine with Intel Core i5-6500 3.20GHz and 32GB memory. All algorithms used in our experiments were implemented in Python (version 3.6)[23].

Datasets. For each dataset, we computed the similarity matrix as an input for all algorithms. In the initial experiment, the similarity matrices are calculated in a metric space. Later, we carry out the other experiments for time-series data in the next section. Table 2 presents the properties of each dataset.

TABLE 2
Dataset properties

Dataset	# of Instances	# of Features	# of Clusters
aggregation	788	2	7
flame	240	2	2
s3	5000	2	15
wine	178	13	3
wdbc	569	30	2
ecoli	336	7	8
dermatology	358	34	6

Performance Measurement. We evaluated the output of the algorithms by considering pairs of the data points [31]. *Precision* is calculated as the fraction of pairs that are correctly assigned to the same cluster; *recall* is the fraction of actual pairs that are identified by the algorithms. Precision and recall provide different perspectives to evaluate the performance. In addition, *F-measure* is the harmonic mean of precision and recall and shows the overall performance of a predicted result. We give the definition of F-measure as follows:

$$F\text{-}measure = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Moreover, when the datasets have a large number of classes which usually have more pairs of True Negatives, we can consider another measure: *Rand index*. Rand index measures the fraction of pairs that are correct. Formally, Rand index is defined as follows:

$$Rand\ Index = \frac{True\ Positive + True\ Negative}{\binom{n}{2}},$$

where n is the number of data points. We also consider another widely-used measure for evaluating clustering ap-

proaches, Normalized Mutual Information (NMI) [12]. Here is the formal definition:

$$\text{NMI}(Y, C) = \frac{2 \times I(Y;C)}{H(Y) + H(C)},$$

where $H(Y) = -\sum_k P(y_k) \log P(y_k)$ denotes the entropy of class labels and $P(y_k)$ denotes the probability that points belong to the class y_k. The mutual information $I(Y;C) = H(Y) - H(Y|C)$. The higher NMI means the better quality of clustering. NMI can be used to compare two clusterings that have different number of clusters since it is normalized. Here we set $minPts = 10$ and $\epsilon = 0.03$ for DBSCAN. For SPECTACL, we set $k = 10$, suggested by [9]. We give the best setting of the smoothing parameter for DENCLUE 2.0 (e.g., $h = 0.08$ for aggregation and $h = 0.28$ for wine).

As shown in Tables 3 and 4, SDP outperforms all the clustering algorithms including DP regarding F-measure, Rand index and NMI. In particular, SDP_{dist} performs better than SDP_{den} in almost all the test datasets, except the two-dimensional datasets. Besides, notice that the performance of SDP_{den} and DP are almost the same except for the two high dimensional datasets: ecoli and dermatology which have overlapping borders. Next we discuss the details of the comparison results as well as the evaluation of the cluster center quality over the metric datasets.

TABLE 3
Performance comparison between SDP and other clustering algorithms over two-dimensional datasets

Algorithm	Precision	Recall	F-Measure	Rand Index	NMI
aggregation					
Hierarchical	0.9510	0.8574	0.9018	0.9596	0.89
K-Means	0.9146	0.7324	0.8134	0.9273	0.86
DBSCAN	0.9984	0.9584	0.9780	0.9906	0.97
SPECTACL	0.9572	0.8000	0.8715	0.9489	0.92
DENCLUE	0.8591	0.9358	0.8958	0.9529	0.91
DP	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.00**
SDP_{den}	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.00**
SDP_{dist}	0.9794	0.9128	0.9449	0.9770	0.93
flame					
Hierarchical	0.5225	0.7429	0.6135	0.4985	0.07
K-Means	0.7579	0.7046	0.7303	0.7211	0.40
DBSCAN	0.9844	0.9632	0.9737	0.9721	0.88
SPECTACL	0.9556	0.9685	0.9620	0.9590	0.85
DENCLUE	0.9748	0.9790	0.9769	0.9752	0.90
DP	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.00**
SDP_{den}	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.00**
SDP_{dist}	0.9693	0.9693	0.9693	0.9671	0.87
s3					
Hierarchical	0.4898	0.6192	0.5469	0.9317	0.70
K-Means	0.7317	0.7350	0.7334	0.9644	0.79
DBSCAN	0.2334	0.6541	0.3440	0.8339	0.54
SPECTACL	0.5298	0.5679	0.5481	0.9376	0.69
DENCLUE	0.7449	0.7527	0.7488	0.9664	**0.80**
DP	0.7464	0.7532	0.7497	0.9665	**0.80**
SDP_{den}	0.7464	0.7532	0.7497	0.9665	**0.80**
SDP_{dist}	**0.7522**	**0.7558**	**0.7540**	**0.9672**	**0.80**

3.2 Comparison result

We initially test the datasets in which the similarity distance between each pair of points is measured in the metric space. Tables 3 and 4 shows the performance measurement of each algorithm. Obviously, SDP outperforms the other algorithms regarding each measure over all the datasets (except recall over the wdbc dataset). More precisely, SDP_{dist} performs best excluding the aggregation and flame datasets, and SDP_{den} and DP achieve the exact same result, except for the ecoli and dermatology datasets. In the next section, we compare the performance of these algorithms over time-series data, which reveals SDP's significant improvement against DP.

TABLE 4
Performance comparison between SDP and other clustering algorithms over multi-dimensional datasets

Algorithm	Precision	Recall	F-Measure	Rand Index	NMI
wine					
Hierarchical	0.8670	0.8830	0.8749	0.9147	0.78
K-Means	0.8743	0.8599	0.8670	0.9109	0.78
DBSCAN	0.5352	0.6591	0.5907	0.6914	0.50
SPECTACL	0.7874	0.8473	0.8162	0.8711	0.72
DENCLUE	0.8128	0.7429	0.7763	0.8553	0.72
DP	0.8549	0.8411	0.8479	0.8981	0.77
SDP_{den}	0.8549	0.8411	0.8479	0.8981	0.77
SDP_{dist}	**0.9144**	**0.9031**	**0.9087**	**0.9387**	**0.84**
wdbc					
Hierarchical	0.6474	0.8700	0.7424	0.6790	0.36
K-Means	0.7422	0.8417	0.7889	0.7605	0.43
DBSCAN	0.5961	0.6203	0.6080	0.5747	0.26
SPECTACL	0.7826	0.7475	0.7647	0.7554	0.41
DENCLUE	0.5322	**0.9975**	0.6941	0.5326	0.01
DP	0.8780	0.9155	0.8964	0.8874	0.68
SDP_{den}	0.8780	0.9155	0.8964	0.8874	0.68
SDP_{dist}	**0.8925**	0.9230	**0.9075**	**0.8999**	**0.70**
ecoli					
Hierarchical	0.6704	0.5919	0.6287	0.8112	0.63
K-Means	0.7473	0.4382	0.5525	0.8082	0.56
DBSCAN	0.5478	0.8409	0.6635	0.7696	0.50
SPECTACL	0.7457	0.5163	0.6102	0.8218	0.55
DENCLUE	0.4766	**0.9419**	0.6330	0.7050	0.53
DP	0.7047	0.4340	0.5372	0.7980	0.58
SDP_{den}	0.7232	0.8197	**0.7684**	0.8665	**0.69**
SDP_{dist}	**0.7592**	0.7679	0.7635	**0.8715**	**0.69**
dermatology					
Hierarchical	0.5114	0.7229	0.5991	0.8061	0.73
K-Means	0.7711	0.7306	0.7503	0.9025	0.84
DBSCAN	0.5052	0.7245	0.5953	0.8026	0.68
SPECTACL	0.7345	0.7393	0.7369	0.8942	0.80
DENCLUE	0.5571	0.8525	0.6739	0.8346	0.72
DP	0.8585	0.8503	0.8544	0.9419	0.81
SDP_{den}	0.7572	0.8498	0.8008	0.9153	0.84
SDP_{dist}	**0.8875**	**0.9271**	**0.9069**	**0.9452**	**0.89**

In the following, we look into the performance result according to the different properties of the datasets. As shown in Tables 3 and 4, the distance-based clustering algorithm, K-Means, did not perform well in recall over the aggregation and flame (and ecoli) datasets. In contrast, based on the center selection strategy, SDP and DP derived a completely correct clustering result over aggregation and flame. On the other hand, K-means performs better for the remaining datasets. However, the pure density-based algorithm, DBSCAN which has a very good performance over aggregation and flame with shaped clusters performed poorly in precision over the other five datasets: s3, wine, wdbc, ecoli and dermatology. One can observe that SPECTACL which incorporates the spectrum of an input similarity matrix into DBSCAN, can compensate the flaw of DBSCAN for these datasets in which clusters may have overlapping borders. However, for time-series data, the performance of SPECTACL is surprisingly worse than DBSCAN, as shown in the next section. The experiment results reveal that these clustering algorithms may have

significantly different performance according to the dataset properties, although SDP still performed better than all the others over every dataset.

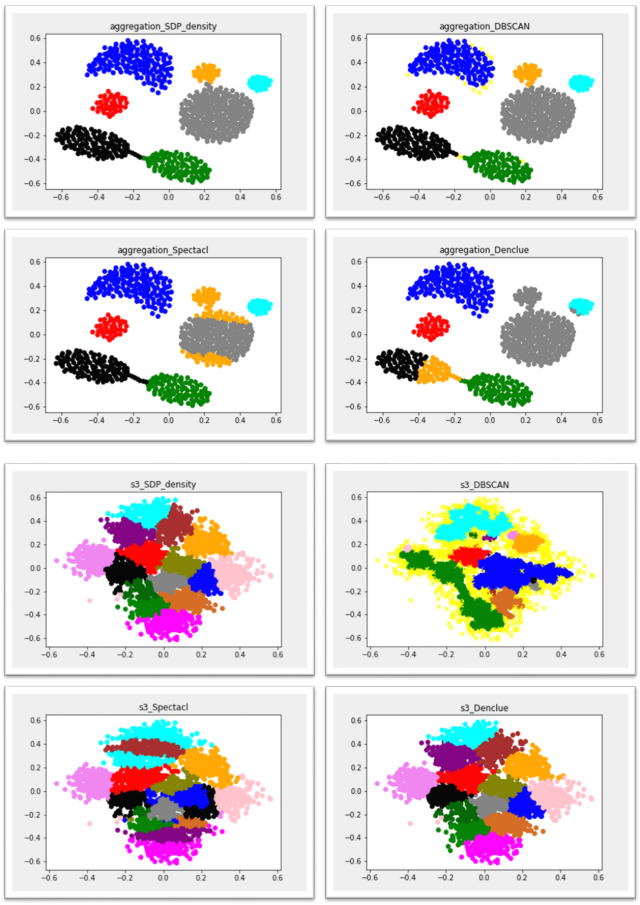


Fig. 3. Clustering results of SDP, DBSCAN, SPECTACL, DENCLUE over two-dimensional datasets: (top) aggregation (bottom) s3

Figure 3 (Figure 4) illustrates the key difference between the clustering results of SDP_{den}, DBSCAN, SPECTACL and DENCLUE 2.0 over the datasets: aggregation and s3 (wine and ecoli, respectively). Although DBSCAN performs well in aggregation with shaped clusters, the setting of $minPts$ and ϵ obviously affects the performance. For s3, wine and ecoli, DBSCAN missed some points and considered them to be outliers (yellow points). SPECTACL improves the performance of DBSCAN but still cannot *extend* their clusters to derive a correct clustering even for aggregation and wine. On the other hand, DENCLUE 2.0 exploits a (new) hill climbing approach for identifying the local maxima of a density estimation function (and then assigns points to the local maxima), which is somehow similar to the center selection procedure in DP. It coped with most of the metric datasets well, epecially two-dimensional data; however, it derived a poor clustering over wdbc, ecoli and dermatology (especially for wdbc, which can be indicated by the measure of NMI (0.01), as shown in Figure S7 in the supplementary). Note that DENCLUE 2.0 actually derived four clusters (one more than the correct solution) for wine, when achieving its best clustering result.

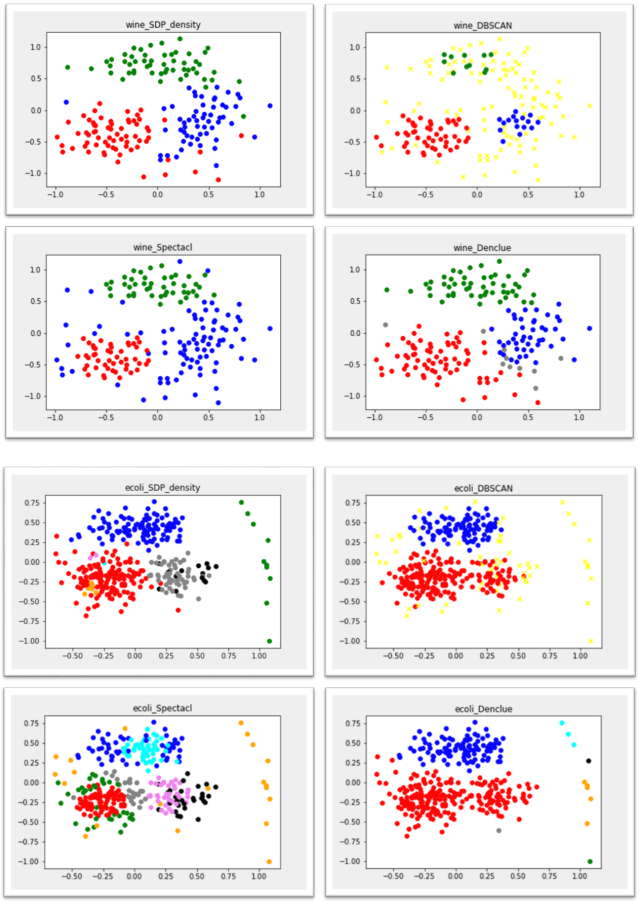


Fig. 4. Clustering results of SDP, DBSCAN, SPECTACL, DENCLUE over multi-dimensional datasets: (top) wine (bottom) ecoli

3.3 Evaluation of cluster centers

In this section, we introduce another performance measure: *center quality*. That is, we define the measure by calculating the percentage of the data points that have the same label as their cluster centers, i.e. the ratio of correct cluster assignment. We give an example to illustrate the importance of the new measure in Figure 5. Let the squares and triangles represent the data points with Class 1 and Class 2, respectively, and the output is classified into two clusters. The Rand index of the output is 0.69. If one algorithm selects A and C as cluster centers, the *center quality* of the output is 0.8182. However, if B is selected as the center of the left cluster instead of A, the center quality drops to 0.5909, even though the value of the Rand index remains the same.

TABLE 5
Cluster center quality of SDP and DP

Dataset	SDP_{dist}	SDP_{den}	Density Peaks
aggregation	0.9975	**1.0000**	**1.0000**
flame	**1.0000**	**1.0000**	**1.0000**
s3	**0.8628**	0.8598	0.8598
wine	**0.9607**	0.9382	0.9382
wdbc	**0.9525**	0.9402	0.9402
ecoli	0.7708	**0.7827**	0.5863
dermatology	**0.8496**	0.8073	0.8492

As shown in Table 5, SDP still performs well with respect to the center quality measure. Similarly, SDP_{dist} performs slightly better than SDP_{den} and DP over all the datasets except aggregation and ecoli, while SDP_{den} and DP have

TABLE 6
Time Series Datasets Properties

Dataset	# of Instances	# of Clusters	Time Series Length	Warping Windows(%)
50 Words	455	50	270	6
WordSynonyms	638	25	270	8
MedicalImages	760	10	99	20

exactly the same performance in the first five datasets. In the next section, we show the improvement of SDP against DP for time-series datasets.

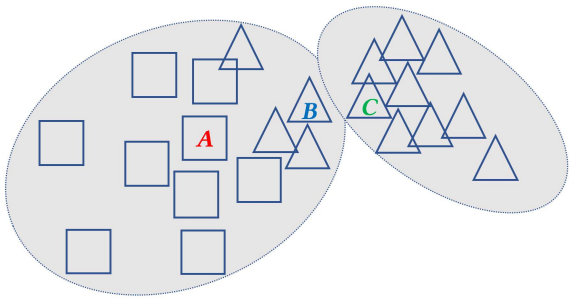


Fig. 5. An example to illustrate the *center quality* measure; There is no difference in the Rand index of the output clusters when we select different centers in the left cluster. However, the center quality is different when we choose A or B as the center.

4 Time Series Data

In this section, we consider time-series data (in non-metric spaces) and compare SDP with other clustering algorithms. The past studies [3], [25], [30] pointed out that for time series data, DTW is better exploited than Euclidean Distance, especially when dealing with clustering problems. We detail the settings as follows.

Datasets. For each dataset, we computed the similarity matrix as an input for all algorithms. In the experiment for time-series data, the similarity matrices are calculated using DTW. Each dataset has a different warping window of DTW as shown in Table 6 [25].

4.1 Comparison result

Table 7 compares the clustering performance of SDP against other clustering algorithms. The results demonstrate that SDP outperforms the others in precision, F-measure, Rand index and NMI except for the MedicalImages dataset in which DBSCAN has better performance. First of all, we look into the performance difference between SDP and DP for the time-series datasets. SDP derived better precision, recall, F-measure, Rand index and NMI than DP for all the instances. Moreover, SDP obtains a significant improvement against DP in precision (that is, 0.3186 for 50 Words, 0.2481 for WordSynonyms and 0.1712 for MedicalImages). We also notice that both SDP_{dist} and SDP_{den} perform better than all the other four approaches in each of precision, F-measure, Rand index and NMI (except DBSCAN for the MedicalImages dataset). DBSCAN actually has the best performance in recall (with the setting of $minPts = 4$ and $\epsilon = 0.05$). We remark that we did not compare SDP with K-Means over time-series datasets because K-Means is usually performed in metric spaces. We also ignore DENCLUE 2.0 because it obtained a poor clustering for each time-series dataset, after trying all possible settings of its smoothing parameter.

One can observe that SDP works better especially for the datasets containing clusters with significantly different sizes. For example, let us take a close look at the 50 Words dataset with 50 clusters. The five largest clusters of 50 Words contain 57, 42, 35, 34 and 28 data points, respectively; however, the median cluster size of 50 Words is only five. Therefore, density-based algorithms, such as DBSCAN, SPECTACL and DP cannot work well when extending or growing their clusters with a fixed setting of ϵ. Moreover, the center selection strategy of DP may choose a center with a large local density but a small distance to its CHD. Figure 6 explains the reason why SDP derives better precision, but DBSCAN, SPECTACL and DP considered the points in the lower right corner to be outliers or a couple of clusters of points. Similarly, Figure 7 also illustrates the superior performance of SDP against other algorithms.

TABLE 7
Performance comparison between SDP and other clustering algorithms over time-series datasets

Algorithm	Precision	Recall	F-Measure	Rand Index	NMI
50 Words					
Hierarchical	0.9204	0.8850	0.9024	0.9913	0.96
DBSCAN	0.6397	**0.9660**	0.7697	0.9737	0.90
SPECTACL	0.3495	0.3878	0.3676	0.9392	0.81
DP	0.6453	0.5540	0.5962	0.9658	0.85
SDP_{den}	**0.9768**	0.9212	**0.9482**	**0.9954**	**0.97**
SDP_{dist}	0.9639	0.8963	0.9289	0.9937	0.96
WordSynonyms					
Hierarchical	0.8114	0.6163	0.7005	0.9534	0.86
DBSCAN	0.6382	**0.9600**	0.7667	0.9484	0.85
SPECTACL	0.4724	0.3976	0.4318	0.9075	0.79
DP	0.6338	0.4982	0.5579	0.9302	0.83
SDP_{den}	0.8599	0.8485	0.8542	0.9744	**0.90**
SDP_{dist}	**0.8819**	0.8983	**0.8900**	**0.9804**	0.89
MedicalImages					
Hierarchical	0.9367	0.6214	0.7471	0.8757	0.86
DBSCAN	0.9558	**0.8212**	**0.8834**	**0.9359**	0.86
SPECTACL	0.5479	0.3956	0.4594	0.7250	0.65
DP	0.8041	0.5272	0.6369	0.8224	0.75
SDP_{den}	0.9753	0.6217	0.7594	0.8836	**0.88**
SDP_{dist}	**0.9774**	0.6045	0.7470	0.8790	0.86

4.2 Other evaluations

Next we consider the accuracy of each cluster center, i.e. *cluster center quality*. As shown in Table 8, the center selection strategy of SDP also outperforms DP in terms of center quality. SDP obtains an improvement of center quality against DP on the three datasets, precisely, by 24% for 50 Words, 14.85% for WordSynonyms and 3.67% for MedicalImages.

As previously mentioned, we also refer to [8] and discuss the parameter sensitivity of SDP concerning the setting of cutoff distance. Here we used one metric dataset and one time series dataset: wine and MedicalImages, respectively

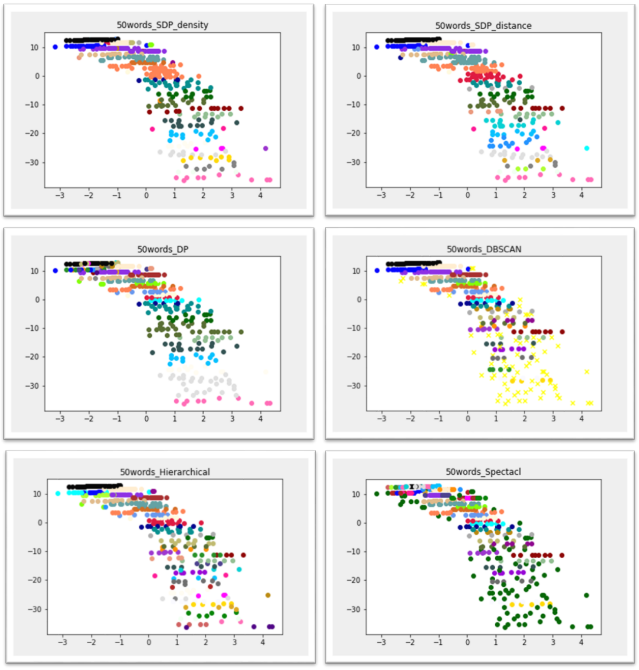


Fig. 6. Clustering results of SDP_{den}, SDP_{dist}, DP, DBSCAN, Hierarchical and SPECTACL over the 50 Words dataset

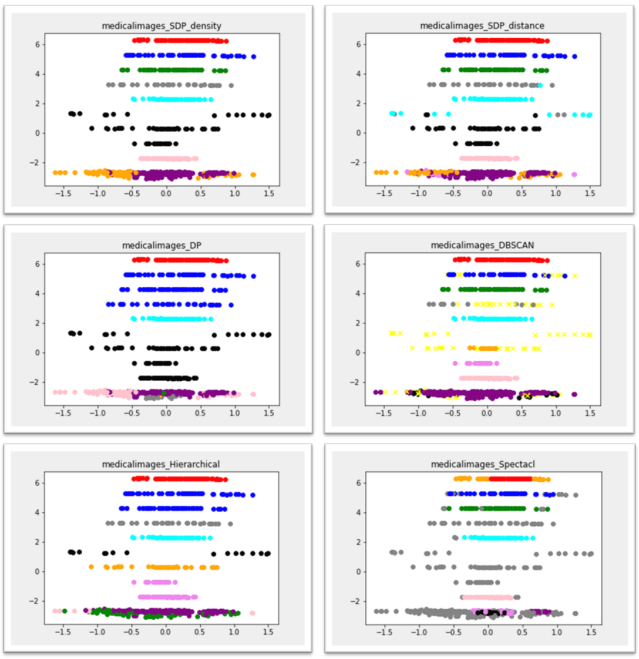
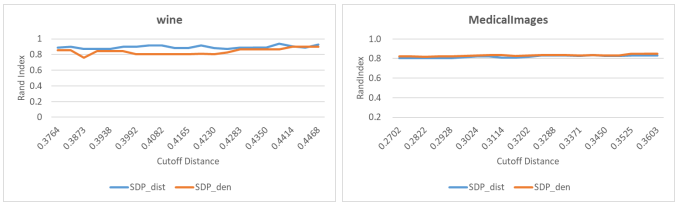


Fig. 7. Clustering results of SDP_{den}, SDP_{dist}, DP, DBSCAN, Hierarchical and SPECTACL over the MedicalImages dataset

and examined the range of cutoff distances, letting the average local density lie between 1% to 3% of the total number of data points in the dataset. As shown in Figure 8, our SDP algorithm is capable of selecting an appropriate value of cutoff distance. Thus, the result demonstrates the robustness of SDP as well as its stable performance of the output clustering. Precisely, the standard deviations of Rand index for SDP_{dist} and SDP_{den} are very low, only 0.0191 and 0.0375, respectively for wine (and 0.0112 and 0.0093, respectively for MedicalImages).

TABLE 8
Cluster Center Quality of SDP and DP

Dataset	$\mathbf{SDP_{dist}}$	$\mathbf{SDP_{den}}$	Density Peaks
50 Words	**0.7231**	0.7187	0.5495
WordSynonyms	0.5799	**0.6129**	0.5219
MedicalImages	0.5013	**0.5395**	0.5197



(a) (b)

Fig. 8. The robustness of our SDP algorithm for (a) wine; and (b)MedicalImages

5 Dynamic Model

In this section, we refer to [7] which studied the dynamic version of DBSCAN using a grid graph approach, and particularly consider the dynamic model of SDP from the theoretical perspective. We mainly discuss two operations: data point addition and data point deletion. Any dynamic operation may change the local connectivity of a dataset, so clustering results may vary by operations [19]. To avoid re-computing the entire dataset, we attempt to design the dynamic version of SDP, called the dynamic seed-and-extension-based density peaks (DSDP) to reduce the dynamic effects. We also devise the dynamic operations of DP, called dynamic density peaks (DDP) as well. Numerical comparisons between DSDP and DDP are provided later.

We let p be the point added into the input dataset or removed from the dataset during dynamic operations. Assume there are n data points in total after point insertion or deletion. Note that there are three main procedures of SDP in which the last two procedures are performed based on the first procedure of calculating the quantities ρ and δ. Here we divide the dynamic algorithm into two parts: δ, ρ-updates and cluster updates.

5.1 δ, ρ-Updates

There is no obvious difference between DSDP and DDP regarding δ, ρ-updates. In the following, we discuss the dynamic updates of ρ and δ, respectively.

5.1.1 *Update of the local density*

The local density ρ_i of a point i is defined as the number of points within the cutoff distance, d_c, of point i. For *dynamic* point p, we denote the set of points in the *local zone* of p by L; that is, only the *local points* in L may change their ρ due to the dynamic operation of p.

Lemma 1. *Given a point p, it takes $O(n)$ time to update every local density ρ_i, $1 \leq i \leq n$, if needed, for point insertion of p, and it takes $O(|L|)$ time for point deletion of p, where $|L|$ is the number of points in the local zone of p.*

Proof. Obviously, the δ, ρ-updates can be done if we go through all the n data points. For point insertion, when p is inserted, we first identify the points that are in the *local zone* of p by checking the distance between p and all the other points. Here, for each point, we use a linked list to store the points in its *local zone*. It takes $O(n^2)$ preprocessing time and $O(n^2)$ space. To check the distance between $n-1$ pairs of points and update the lists, it takes $O(n)$ time. For point deletion, we construct an inverse array to store every position in the linked list of the *local zone* of each point. When p is deleted, we can check the inverse arrays and update the lists of *local points* for every point in the *local zone* of p as well as their local density in constant time. Hence, the local density ρ of the *local points* of p can be updated in $O(|L|)$ time.

We remark that the process of checking if any other points lie in the local zone of p can be done more efficiently, e.g., using *range tree*, but the update of the densities of these points still takes at least $O(|L|)$ time. □

5.1.2 Update of the CHD

Next, we discuss the change of the other quantity δ. Let $\rho_{L_{max}}$ denote the highest density of *local points* in L. Note that if ρ_i changes, δ_i may also change due to the dynamic operations. The following lemma shows the necessary condition of updates of δ.

Lemma 2. *Given a dynamic point p with the set of its local points L, a point i may change its δ_i and* CHD$_i$ *only if $\rho_i < \rho_{L_{max}}$.*

Proof. Consider a point i with $\rho_i \geq \rho_{L_{max}}$. By Definition 2, δ_i is defined to be the minimum distance between point i and any other point with a higher density. Since the dynamic operation changes only the density of *local points* in L, the *local points* cannot become candidates of CHD$_i$ if $\rho_i \geq \rho_{L_{max}}$. Thus, δ_i and CHD$_i$ of point i do not change, and the proof is complete. □

Based on Lemma 2, we consider only the points in the *local zone* of p as well as the points with their density lower than $\rho_{L_{max}}$. First, we sort all data points by their local density in descending order and build a sorted list. We also spend $O(n^2)$ preprocessing time and $O(n^2)$ space collecting a sub-list of the points with higher density than ρ_i from the sorted list, for each point $1 \leq i \leq n$. Then, for each point i, we construct a minimum heap for each sub-list, where the value of each heap node j in the heap structure represents its distance from i, i.e. d_{ij}.

Lemma 3. *For each point i, δ_i and* CHD$_i$ *can be updated, if needed, in $O(\log n)$ time.*

Proof. When quantity ρ is updated due to point insertion or deletion, the sorted list as well as all the sub-lists can be updated in $O(n + |L|)$ time. For each point i, the heap structure can return the minimum value in constant time and can be updated in $O(\log n)$ time. Therefore, the total update time is bounded by $O(n \log n)$. Note that we consider only the points with lower density than $\rho_{L_{max}}$, which can significantly reduce the update time in practice using an index-based approach. □

More precisely, we look into the points that will change their CHD and will then be assigned to different clusters during the seed-and-extension procedure. First, a local point $i \in L$ may change its CHD due to the insertion or deletion of p; that is, i may be assigned to a different cluster because the value of ρ_i increases or decreases by one so that CHD$_i$ changes. Next, we consider the *non-local* points in the neighboring levels of a local point i in the tree structure, i.e. i's parents and i's children. Obviously, i's parents will not change its CHD if it is not in L. Thus, i's parents remain in its original cluster. On the other hand, if $j \notin L$ is a child of i in the tree structure, j may change its CHD since the deletion of a dynamic point p may violate CHD$_j = i$ by definition. The following lemma specifies the points that may change their CHD in the tree structure. Note that this lemma would be helpful to the updates of cluster assignment labels.

Lemma 4. *During the seed-and-extension procedure, a point may change its* CHD *in the tree structure due to dynamic operations if and only if it is a local point in L or if it is the child of a local point.*

Proof. As mentioned, a local point as well as its child may change their CHD due to dynamic operations, even if the child is not in L. Consider a point i which is neither in L nor a child of any local point in the tree structure. We are aware that i's parents (more precisely, i's ancestors) do not change its CHD if it is not in L. Moreover, let j be i's descendant and based on the assumption, neither j nor j's parents are in L. It is straightforward to see that j does not change its CHD because both ρ_j and the value of ρ of j's parents remain unchanged. □

5.2 Cluster updates

The next step is to find new cluster centers and update the assignments for all the points, if needed, based on the dynamic updates of quantities. As mentioned in Section 3, we use the seed-and-extension procedure to select centers in SDP; in contrast, DP simply picks centers from the sorted γ list heuristically. Here, we follow Algorithm 2 for the cluster center selection in DSDP (as shown in Algorithm 5). Note that the sorted γ list can also be maintained by a minimum heap structure, which takes $O(m \log n)$ time, where m denotes the number of points that have a lower density than $\rho_{L_{max}}$.

Algorithm 5: Cluster Center Selection of DSDP

Input: *Data*, the input clustering dataset; k, the number of clusters; *centers*, a list of points as the former cluster centers; γ_{sorted}, the vector in descending order for center candidates; δ, the distance to the CHD point; *CHD*, the closest data point with a higher density; L^*, a set of local points and their children for checking cluster updates;

Output: : *new centers*, a list of the points as present cluster centers; *assignment*, cluster labels for each data point; *checking points*, the points that have to be checked after cluster center selection;

1: *new centers = Algorithm 2(Data, k, γ_{sorted}, δ, CHD)*;
2: L^* = intersect(L^*, difference(*centers*, *new centers*));
3: L^* = intersect(L^*, *CHD* = difference(*new centers*, *centers*));

The cluster assignment updates of DDP are the same as the third procedure of DP; that is, reassigning all the data points according to the updates of quantities (as shown in Algorithm 6). Precisely, each unassigned data point is assigned to its CHD's cluster (similar to Algorithm 3). Hence, DDP takes $O(n \log n)$ time for the whole procedure of cluster updates.

Algorithm 6: DDP: Dynamic Density Peaks Cluster Assignment

Input: *Data*, the input clustering dataset; *CHD*, the closest data point with a higher density; ρ, local density vector for all points in the dataset; δ, the distance to the CHD points;
Output: *Assignment*, cluster labels for each data point

```
[ρ_sorted, sortedIndex] = Sort(ρ, descend);
for i = 1 to the size of Data do
    if Assignment(sortedIndex(i)) is not done then
        Assign sortedIndex(i) to the cluster in which its
        CHD point lies;
    end if
end for
```

On the other hand, the dynamic updates of the cluster assignment labels in DSDP can be done more efficiently using the merits of the seed-and-extension approach. More precisely, in order to avoid reassigning all the data points that have to be updated, we modify the clusters, if necessary, according to dynamic updates of quantities by adjusting the structure of the spanning forest built by SDP. Based on Lemma 4, we consider only the set of points that may change their CHD. For ease of convenience, we denote the set by L^*. Notice that a cluster center may belong to another different cluster from its CHD, due to dynamic updates. Therefore, if there is any change on a cluster center, we need to put the center point into L^* even its CHD remains the same.

Here we describe the steps of updating the cluster assignment in DSDP. The idea of Algorithm 7 is as follows: The cluster centers and L^* are updated by Algorithm 5 due to dynamic operations. Then we check if each point in L^* has the same cluster as its CHD. If not, its cluster assignment is updated level by level and so are its descendants. Because each point should have the same cluster as its CHD except center points, when reassigning a point, we move the whole subtree rooted at the point in the tree structure. Figure 9 illustrates an example of updating the tree update. The following lemma thus shows that DSDP can update the cluster assignment in $O(|L^*|)$ time, where $|L^*|$ is the number of points we have to check.

Lemma 5. *The cluster assignment can be dynamically updated in $O(|L^*|)$ time.*

Proof. Algorithm 7 checks the points in L^* level by level in a top-down manner in the tree structure. As mentioned, if a point is reassigned to a new center, then its descendants follow according to the seed-and-extension procedure of SDP. The cluster assignment can be simply done by moving the whole subtree rooted at the point, and so too its descendants. Hence the assignment can be updated in $O(|L^*|)$ time. □

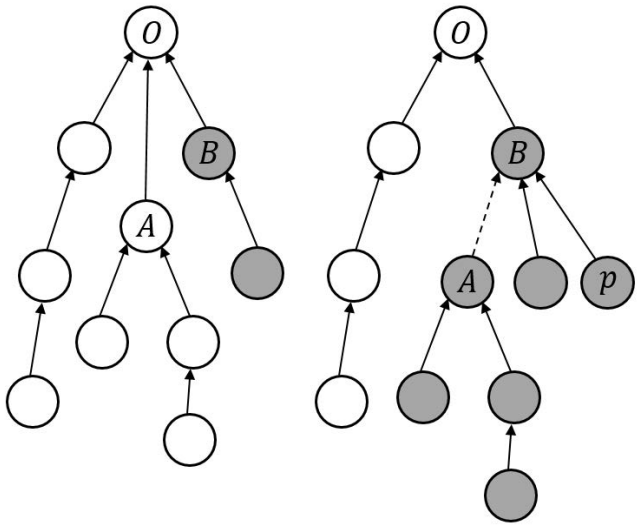


Fig. 9. An example illustrating a simple tree update operation due to a dynamic insertion of p. (Left) Assume point A and point B belong to different clusters but have the same local density, i.e. $\rho_A = \rho_B$, where point O and point B are the centers of their clusters, respectively, and $\rho_O > \rho_A + 1$. We also assume $d_{AB} < d_{AO}$. (Right) Suppose $B \in L$ and thus $\rho_B := \rho_B + 1$ due to the insertion of p. Then CHD_A is changed from O to B, and the subtree rooted at A is moved so that A is a child of B.

Algorithm 7 : DSDP_{den}: Dynamic Density-based Cluster Assignment

Input: *Data*, the input clustering dataset; *CHD*, the closest data point with a higher density; *new centers*, a list of the points as present cluster centers; *checking points*, a list of points as check points for updating clusters;
Output: *Assignment*, cluster labels for each data point

```
[C_sorted, sortedIndex] = Sort(checking points, descend)
for i = 1 to the size of checking points do
    if Assignment(sortedIndex(i))≠
    Assignment(CHD(sortedIndex(i))) then
        update list = sortedIndex(i)
        cluster label = Assignment(CHD(sortedIndex(i)))
        while update list is not empty do
            update list = update list delete new centers
            Assignment(update list) = cluster label
            next = find(CHD = update list))
            update list = next
        end while
    end if
end for
```

We recall Algorithms 3 and 4 in SDP, which assign the remaining points that are not in the local zone of any selected cluster centers, using the density-based assignment and distance-based assignment approaches, respectively. The reassignment of SDP_{den} can be dynamically performed in a similar way as above, i.e. using the merit of the tree structure. However, the distance-based assignment approach makes no difference between SDP_{dist} and DP regarding cluster re-assignment. We need to check each point outside the local zone of all the cluster centers and update them one by one. We thus suggest SDP_{den} be applied in the dynamic model.

We conclude that, in the dynamic model, the δ, ρ-updates take $O(n \log n)$ time for both DSDP and DDP by Lemma 1 and Lemma 3. Furthermore, DSDP can reduce the update time in practice by Lemma 2 using an index-based approach. Concerning the cluster updates, DSDP considers only the points in L^*, and efficiently updates the tree structure, if needed, while DDP has to traverse all n data points due to dynamic operations. This difference reveals the advantage of using DSDP, especially for a large-scale dataset. In particular, DSDP performs faster than DDP regarding the total update time when choosing an appropriate setting of the cutoff distance d_c. The experiment results compare the performance of DSDP and DDP in Section 5.3.

5.3 Evaluation of dynamic model

SDP not only performs well in the static model, but also takes less time than DP for dynamic updates. In the previous subsection, we discuss the algorithm design of DSDP (the dynamic version of SDP) as well as the theoretical analysis of DSDP and DDP (the dynamic version of DP). Here, we carry out numerical comparisons between DSDP, DDP and SDP to demonstrate the superior performance of DSDP regarding running time. Note that the clustering results of DSDP and SDP are exactly the same, so we only consider the execution time issue in the dynamic model. We thus tested the algorithms mainly on large-scale datasets, and the experiments were conducted on another machine with Intel Xeon Gold 6154 3.70GHz and 360GB memory.

TABLE 9
Dataset properties

Dataset	# of Instances	# of Clusters
MNIST	60000	10
shuttle	58000	7
avila	20867	12
HTRU2	17898	2
ESR	11500	5
s3	5000	15

The dataset properties are shown in Table 9. For each of the six datasets, we perform fifty random dynamic operations to simulate dynamic data inputs. The two key procedures of each dynamic operation of DSDP are δ, ρ-updates and cluster updates, as discussed earlier. We evaluate the average time cost of every procedure for each experiment. Note that we consider the center selection time independently for only DSDP and SDP (i.e. Algorithm 5 and Algorithm 2, respectively) as shown in Figure 10(bottom), because DDP and DP immediately determine their cluster centers once the δ, ρ-updates have been done. The figure shows that DSDP and SDP have almost equal time cost over the six datasets, since the key difference in the cluster updates occurs in cluster assignment.

Figure 10(top) and (middle) show that DSDP is clearly faster than its static version (SDP) as well as the dynamic version of DP (DDP), which reveals the advantage of the design of the dynamic algorithm. Precisely, DSDP takes at least 9% less time than DDP concerning the δ, ρ-update time cost (17.18% for MNIST, 23.61% for shuttle, 19.63% for avila, 22.84% for HTRU2, 9.09% for ESR and 17.65% for s3), as shown in Figure 10(top). Moreover, the cluster

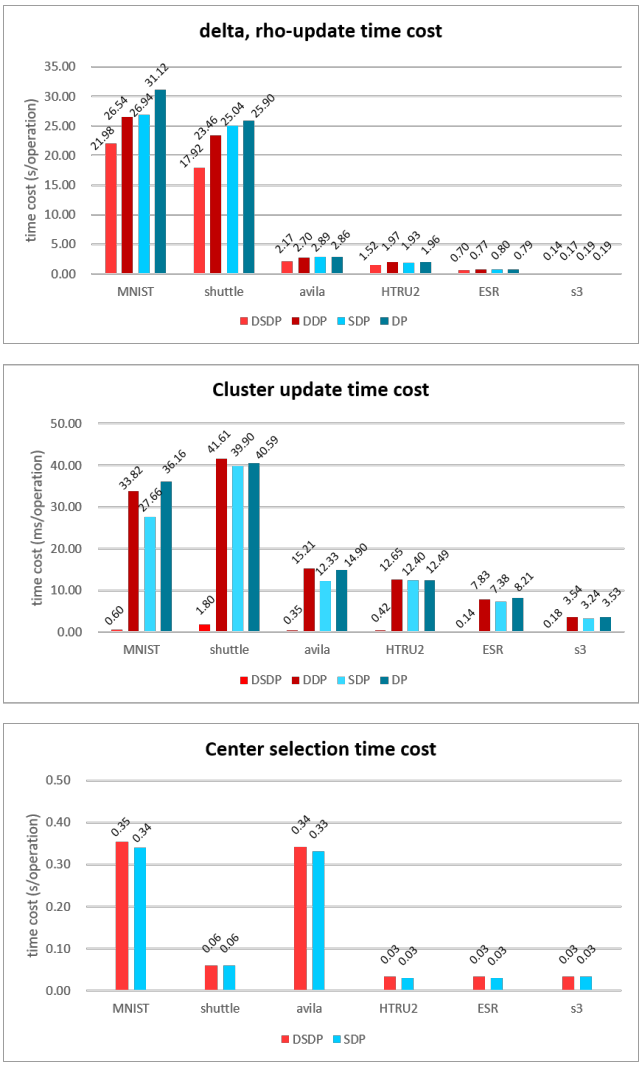


Fig. 10. Comparison results between DSDP, DDP, SDP and DP for each dataset: (top) δ, ρ-update time (middle) cluster update time (bottom) center selection update time

update procedure of DSDP takes under 2 milliseconds even on the two largest datasets (MINST and shuttle) with sizes close to 60,000 points, while the other three (DDP, SDP and DP) take about 30 milliseconds or more. In particular, DSDP performs almost 20 times faster than the other three algorithms on each dataset, as shown in Figure 10(middle). We remark that the static version, DP has a slightly quicker cluster update time than DDP in some experiments (see Figure 10(middle)) because both of them take $O(n \log n)$ time for the cluster updates and sometimes the hidden constant factors vary due to random dynamic operations, especially under such a small time cost. In addition, another interesting observation is that SDP and DP perform slightly faster than DDP in the HTRU2 dataset regarding δ, ρ-update time (see Figure 10(top)). We believe that the reason comes from the small number of clusters in HTRU2, which leads to a very close running time in practice. We also tested the algorithms on another larger dataset, Dota2 with 102,944 points but the benefit of DSDP and DDP is again not obvious because of the same reason. (Dota2 has only two clusters.) Finally, we remark that the standard deviation is low for

every test. For example, in avila, the standard deviations of δ, ρ-update time are 0.6736, 0.0161, 0.0575 and 0.0051 (s) for DSDP, DDP, SDP and DP, respectively; for cluster update time, 0.06, 0.36, 0.31 and 0.33 (ms) for DSDP, DDP, SDP and DP, respectively; and for center selection time, 0.047 and 0.015 (s) for DSDP and SDP, respectively.

Finally, DSDP achieves a significant improvement over DDP as well as their static versions (SDP and DP) concerning the total update time. As shown in Figure 11, DSDP outperforms DDP on every dataset; precisely, 15.96% less time for MNIST, 23.52% less time for shuttle, 7.01% less time for avila, 21.32% less time for HTRU2, and 5.19% less time for ESR. Obviously, DSDP performs much better than DDP especially on a large dataset with more than 50,000 points. One can observe that the performance in avila is not as good as the other two larger datasets (only 7 percent improvement) because DSDP takes more time on center selection. We remark that DSDP can perform better by choosing a more appropriate value of the cutoff distance, i.e. d_c, as we have already shown the robustness of SDP concerning the setting of the cutoff distance in Section 4.2.

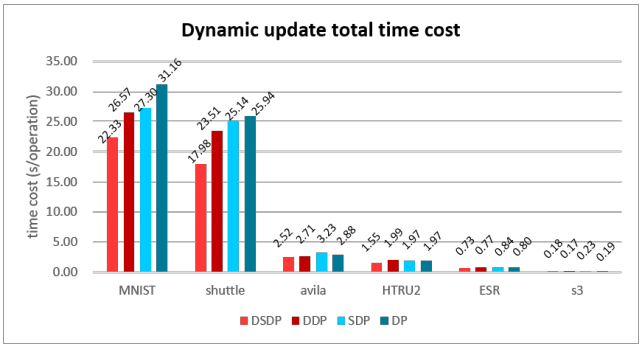


Fig. 11. Comparison of total update time between DSDP, DDP, SDP and DP for each dataset

6 Conclusion

We have proposed the seed-and-extension-based density peaks (SDP) algorithm which can find a more accurate clustering while keeping the property of the output clusters. Moreover, we have given the option to the algorithm that can form the clusters based on the data itself. Our SDP algorithm has better performance than the state-of-the-art clustering algorithms in the literature especially for a variety of datasets. SDP also performs well when dealing with a large number of clusters and even clusters with significantly different densities. We conclude by suggesting several research directions and open problems. For clustering approaches, it is always a challenge to calculate the distance matrix of an input dataset, i.e. $O(n^2)$ complexity for both time and space issues. Some previous studies [6], [20], [22], [21], [16] investigated parallel incremental clustering algorithms for overcoming the challenge. It would be worthwhile to incorporate parallel computing techniques to solve the problem. We also believe that it would be interesting to consider learning-based clustering algorithms for datasets with different properties, especially for classification problems or database indexing frameworks. That is, it would be of independent interest to determine the number of clusters or decide the settings of a clustering algorithm in a more efficient way.

Acknowledgments

We wish to thank the anonymous reviewers for helpful comments, and Yi-Fang Yang for conducting partial computational experiments in the supplementary. We also thank the support from MOST Taiwan (under Grants 110-2221-E-007-106-MY3 and 110-2622-8-007-017-SB), National Center for High-performance Computing (NCHC) and the Brain Research Center under the Higher Education Sprout Project. Our software is available freely for non-commercial purposes from: http://acolab.ie.nthu.edu.tw/sdp/

References

[1] N. Begum, L. Ulanova, J. Wang, and E. Keogh. Accelerating dynamic time warping clustering with a novel admissible pruning strategy. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 49–58, New York, NY, USA, 2015. ACM.

[2] Y. Chen, E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen, and G. Batista. The ucr time series classification archive, July 2015.

[3] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh. Querying and mining of time series data: Experimental comparison of representations and distance measures. *Proc. VLDB Endow.*, 1(2):1542–1552, August 2008.

[4] M. Du, S. Ding, and H. Jia. Study on density peaks clustering based on k-nearest neighbors and principal component analysis. *Knowledge-Based Systems*, 99(1):135–145, 2016.

[5] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pages 226–231. AAAI Press, 1996.

[6] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Michael Wimmer, and Xiaowei Xu. Incremental clustering for mining in a data warehousing environment. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB '98, page 323–333, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[7] Junhao Gan and Yufei Tao. Dynamic density based clustering. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, page 1493–1507, New York, NY, USA, 2017. Association for Computing Machinery.

[8] J. Gao, L. Zhao, Z. Chen, P. Li, H. Xu, and Y. Hu. Icfs: An improved fast search and find of density peaks clustering algorithm. In *Proceedings of the IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress*, pages 537–543. IEEE, 2016.

[9] Duivesteijn W. Honysz P. Morik K. Hess, S. The spectacl of nonconvex clustering: a spectral approach to density-based clustering. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*, AAAI 2019, pages 3788–3795, 2019.

[10] Alexander Hinneburg and Hans-Henning Gabriel. Denclue 2.0: Fast clustering based on kernel density estimation. In *IDA*, LNCS 4723, pages 70–80. Springer, 2007.

[11] Alexander Hinneburg and Daniel A. Keim. An efficient approach to clustering in large multimedia databases with noise. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, KDD'98, page 58–65. AAAI Press, 1998.

[12] Jian Hou, Chengcong Lv, Aihua Zhang, and E Xu. Merging dbscan and density peak for robust clustering. In *International Conference on Artificial Neural Networks*, pages 595–610. Springer, 2019.

[13] L. Kaufman and P.J. Rousseeuw. *Finding Groups in Data: an introduction to cluster analysis*. Wiley, 1990.

[14] S. Kokoska and D. Zwillinger. *CRC Standard Probability and Statistics Tables and Formulae*. Chapman and Hall CRC., 2000.

[15] H.-P. Kriegel, P. Kröger, J. Sander, and A. Zimek. Density-based clustering. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(3):231–240, 2011.

[16] Hans-Peter Kriegel, Peer Kröoger, and Irina Gotlibovich. Incremental optics: Efficient computation of updates in a hierarchical cluster ordering. In *International Conference on Data Warehousing and Knowledge Discovery*, pages 224–233. Springer, 2003.

[17] T Warren Liao. Clustering of time series data—a survey. *Pattern recognition*, 38(11):1857–1874, 2005.

[18] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, pages 281–297, Berkeley, Calif., 1967. University of California Press.

[19] S. T. Mai, S. Amer-Yahia, I. Assent, M. S. Birk, M. S. Dieu, J. Jacobsen, and J. M. Kristensen. Scalable interactive dynamic graph clustering on multicore cpus. *IEEE Transactions on Knowledge and Data Engineering*, 31(7):1239–1252, 2019.

[20] Son Mai, Jon Jacobsen, Sihem Amer-Yahia, Ivor Spence, Phuong Tran, Ira Assent, and Quoc Viet Hung Nguyen. Incremental density-based clustering on multicore processors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[21] Son T Mai, Ira Assent, and Martin Storgaard. 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*, pages 1025–1034, 2016.

[22] Son T Mai, Xiao He, Jing Feng, Claudia Plant, and Christian Böhm. Anytime density-based clustering of complex data. *Knowledge and Information Systems*, 45(2):319–355, 2015.

[23] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[24] Dan Pelleg and Andrew W. Moore. X-means: Extending k-means with efficient estimation of the number of clusters. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, page 727–734, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[25] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 262–270, New York, NY, USA, 2012. ACM.

[26] Sangeeta Rani and Geeta Sikka. Recent techniques of clustering of time series data: a survey. *International Journal of Computer Applications*, 52(15), 2012.

[27] A. Rodriguez and A. Laio. Clustering by fast search and find of density peaks. *Science*, 344(6191):1492–1496, 2014.

[28] R. Sibson. Slink: An optimally efficient algorithm for the single-link cluster method. *The Computer Journal*, 16(1):30–34, 1973.

[29] Dongkuan Xu and Yingjie Tian. A comprehensive survey of clustering algorithms. *Annals of Data Science*, 2(2):165–193, 2015.

[30] Y. Yuan, Y.-P. P. Chen, S. Ni, A.G. Xu, L. Tang, M. Vingron, M. Somel, and P. Khaitovich. Development and application of a modified dynamic time warping algorithm (dtw-s) to analyse the primate brain expression time series. *BMC Bioinformatics*, 12(1):347, Aug 2011.

[31] J. Zhang, Y. Pei, G. Fletcher, and M. Pechenizkiy. Evaluation of the sample clustering process on graphs. *IEEE Transactions on Knowledge and Data Engineering*, 32(7):1317–1332, 2020.

Ming-Hao Tung received his BSc degree in Life Science from National Chung Cheng University, Taiwan in 2014 and MEng degree in Industrial Engineering from National Tsing Hua University, Taiwan in 2017. He is currently working in Micron Technology. His research interests include data mining and large-scale clustering. He started conducting this research work while he was visiting La Trobe University, Melbourne, Australia.

Yi-Ping Phoebe Chen (M'04-SM'07) received the BInfTech and the Ph.D. degrees in computer science from the University of Queensland, Brisbane, Australia. She is a Professor and Chair of the Department of Computer Science and Information Technology, La Trobe University, Melbourne, Australia. She is also a chief investigator in the ARC Center of Excellence in Bioinformatics. She has published over 240 research papers, many of them appeared in top journals and conferences. She has been editorial board for IEEE Transactions on Neural Networks and Learning Systems, IEEE Transactions on Multimedia, Gene and Current Bioinformatics. Phoebe has recently appointed as a member of the College of Experts of the Australian Research Council. She is the Steering Committee Chair of the Asia-Pacific Bioinformatics Conference (founder) and the International Conference on Multimedia Modelling. She has been involved in research on bioinformatics, artificial intelligence and multimedia. More information about her can be found at http://homepage.cs.latrobe.edu.au/ypchen/index.htm .

Chen-Yu Liu received his BEng and MEng degrees in Industrial Engineering from National Tsing Hua University, Taiwan in 2017 and 2019, respectively. He is currently working in Greatek Electronics. His research interests include data mining and dynamic clustering.

Chung-Shou Liao (M'14-SM'20)Chung-Shou Liao joined the faculty of Department of Industrial Engineering and Engineering Management, National Tsing Hua University (NTHU) in February 2010. He has served as a full professor since August 2018. Before joining NTHU, he had worked in Algorithms and Computation Laboratory at Institute of Information Science, Academia Sinica for eight years. His research mainly focuses on designing efficient algorithms that can be used to solve difficult combinatorial optimization problems from real applications. His lab has developed approximation algorithms with theoretical analysis for well-known hard problems such as online shortest path, facility location, domination, and scheduling and packing problems. Dr. Liao has also extended his study to systems biology. He has designed graph-theoretic algorithms for global alignment between multiple biological networks and conducted comparative analysis across species.

Dr. Liao received Outstanding Young Researcher award of IICM (ACM Taiwan) in 2014. He is also a Fulbright Senior Research Scholar for 2018 and 2019. Dr. Liao served as Program Committee Chair of AAAC 2016 and AAAC 2021 (Annual Meeting of Asian Association for Algorithms and Computation) and ISAAC 2018 (the 29th International Symposium on Algorithms and Computation). He is Board Member of AAAC, Steering Committee Member of ISAAC, and Associate Editor of Journal of Combinatorial Optimization. Dr. Liao is Senior Member of ACM and IEEE.