Distributed-Swarm: A Real-Time Pattern Detection Model Based on Density Clustering

TIAO QIAN, SHIMING SUN, XIN SHAN, XUEYUN WEI, CHUNLIANG TAI, AND CHAO LIU
NARI Group Corporation (State Grid Electric Power Research Institute), Nanjing 211106, China
NARI Technology Development Ltd. Company, Nanjing 211106, China
Corresponding author: Tiao Qian (1021216324@qq.com)

ABSTRACT

The advancement of power technology and the improvement of people’s living standards promote the expansion of the power grid scale and the sharp rise in electricity consumption. In the power system, due to the use of various sensors, we can collect a large number of power data (e.g., the spatial-temporal information of electric vehicle charging). Usually, such spatial-temporal data is generated in the form of a data stream. The analysis and mining of such data can be widely applied in power equipment condition monitoring and maintenance, user equipment anomaly warning, urban power grid analysis and other scenarios. Among them, the pattern detection of power data plays a key role in power data analysis. Since the power data such as the spatial-temporal information of electric vehicle charging is time-sensitive, it is crucial to perform real-time pattern mining in real-time monitoring systems. However, state-of-the-art pattern detection methods are built on batch mode. Extending such works directly to an online environment tends to result in (1) expensive network cost, (2) high processing latency, and (3) low accuracy results. In this paper, we propose a framework for frequent motion pattern detection of power data in the real-time distributed environment. Through the softmax differentiation function, the power data is filtered to reduce the workload and improve the performance of the framework. At the same time, we propose the concept of historical state matrix to solve the problem that the nodes of each physical partition in a distributed environment can not perceive each other. Extensive experiments are conducted on real dataset and the experimental results show that our pattern detection is about 70% faster than baseline methods, which proves the huge advantage of our approach over available solutions in the literature.

INDEX TERMS

Power data, real-time processing, pattern detection, distributed computing.

I. INTRODUCTION

With the rapid development of Internet technology and smart grid, massive and increasing volumes of power data are being generated [1], [2]. These data can monitor the status of power equipment, perform abnormal warnings on user equipment and analyze urban power grids, so as to achieve unified management of power data, improve resource management capabilities and power grid operation efficiency [3]. At the same time, due to the wide popularity of electric vehicles, there is no lack of spatial-temporal information of electric vehicle charging in these power data [4]. The longitude, latitude and other information extracted from these power data can be arranged according to the timestamp to get the charging trajectory of a vehicle, which can help us analyze the charging behavior. As one of the most fundamental problems in quantitative trajectory data analysis tasks, pattern mining aims to integrate and classify a large number of disordered and scattered data so as to discover deeper correlations between the data [5]. There are several existing studies on pattern mining. One typical area of study is to discover a group of objects that move together for a certain time period [6], called co-movement patterns detection. The discovery of such clusters has been facilitating in-depth study of animal behaviors, routes planning, vehicle control and power communication network [7]–[9].

Swarm [3] is one of the commonly-used definition of co-movement patterns. A swarm is a group of at least \( m \) objects that travel together for at least a total of \( k \) moments (possibly non-consecutive). Fig. 1 shows a swarm example...
with \( m = 2 \) and \( k = 2 \), where \( o^j_t \) represents the snapshot of \( o_i \) at \( t_j \). We observe that \( o^0_1 \) and \( o^2_2 \) are in the same cluster, while \( o^2_1 \) and \( o^2_2 \) are also in the same cluster, so it is not difficult to find a swarm composed of \( o_1 \) and \( o_4 \) gathering at \( t_0 \) and \( t_2 \). Swarm is more general than other pattern detection methods such as flock [10], convoy [11], group [12], and platoon [13] because of its simple parameters and loose timestamp restrictions. Furthermore, thanks to its fewer parameters and constraints, swarm pattern is instrumental in various current applications. For example, it can be used to detect people who move together during a specified period to predict the traffic congestion area and help drivers avoid in advance, or detect where large-scale activities will occur to remind the corresponding department to response immediately [14].

Due to the co-movement pattern detection is usually performed on data stream, this operation needs to process massive amount of spatial-temporal data in a real-time manner. Existing pattern detection methods [10], [11], [15] use static spatial-temporal data partitioning to discover the potential patterns, which mainly focused on the accuracy of their patterns. However, these methods may fails in real-time environment. In off-line processing, all data is available when the processing starts. In real-time manner, unbounded data arrives in real time, making pattern detection more difficult. For instance, the power grid monitoring methods without sufficient real-time process capacity cannot reflect the electricity consumption of the area in time. When the power consumption is overloaded, it can’t avoid the voltage dropping and even power outage [16]. Besides, in the case of pandemic control during the COVID-19, a real-time monitoring and alerts of high-risk areas can effectively reduce the chance of further transmission [17]. In summary, the challenges in designing real-time co-movement pattern detection can be summarized as: (1) Insufficient real-time performance. When we perform co-movement pattern detection, massive real-time data stream usually contain a lot of sparse data (unable to form clusters). If we process all data, it will cause large delay and waste of resources. Therefore, in order to ensure the efficiency of real-time pattern detection and remove useless loads, it is necessary to utilize a workload shedding mechanism. (2) State maintenance. In a distributed or parallel environment, share-nothing between tasks means that each partition cannot perceive the existence of the others after partitioning the data. At the same time, since the trajectory data is a series of chronologically ordered points, it may be divided into a several segments in a distributed environment. If a cluster located around the border of two partitions, it is most likely not detected due to the data being split, resulting in low accuracy.

To address the aforementioned challenges, in this paper, we propose a real-time pattern detection model based on density clustering, named **D-Swarm** (\( D \) stands for Distributed), to handle the cross-cluster undetectable in the distributed environment and redundant data in the massive data. By integrating D-Swarm into the existing mainstream DSPE (Distributed Stream Processing Engine), a real-time pattern detection model is enabled. We design a lightweight pruning strategy to filter out irrelevant data that are far from any swarm. To increase the processing speed of the swarm, ensure the correct rate of pattern mining and provide a low-latency effect, a suite of optimization techniques should be introduced. First, an excellent data pruning method is necessary to retain useful data. Secondly, a suitable state retention mechanism is essential to discover the existence of cross-cluster. In summary, the significant contributions of this work include:

- We design an accurate and efficient framework D-Swarm for mining co-movement patterns in a distributed environment.
- We propose a lightweight pruning strategy for redundant data filtering on large-scale dataset. According to the density state matrix calculated in real-time, the data is reasonably and timely pruned.
- We utilize the historical state transition matrix to save the state information of the last moment, which maintains the temporal continuity of the trajectory data to a certain extent, and avoids the reduction of the accuracy of pattern detection in the process of data pruning.
- We implement our algorithms on Flink [18], our experiment on real-world data validate the usefulness of our proposal and prove the huge advantage of our approach over other solutions in the literature.

The rest of the paper is organized as follows. We introduce related work in section II. In the new experiments section, we explain some problems and definitions of mining swarm. We specifically describe the D-Swarm framework and the two core parameters in section IV. Section V shows the experimental details and results of this article. Finally, we conclude this article in the brief section VI.

II. RELATED WORK

A. DISTRIBUTED STREAM PROCESSING

The processing of streaming data is gaining in importance, due to the steadily growing number of data sources and the increasing real-time requirements for data analysis [19]. In keeping with this, different distributed stream processing systems have been explored, proposed, including SPADE [20], Naiad [21], Microsoft StreamInsight\(^1\) and

\(^1\)https://blogs.msdn.microsoft.com/streaminsight/
IBM Streams. The systems are either simple prototype systems or closed-source systems, which renders them unsuited as an underlying platform for our work [22].

In recent years, several open-source distributed stream processing platforms have also been proposed. Storm, Spark Streaming and Flink are well-known streaming data processing frameworks and are widely used. From the strategy of streaming data processing, the above three frameworks can be divided into two categories. Both Flink and Storm adopt native streaming. When all data arrives, it will be processed immediately. Spark Streaming is implemented based on micro-batch strategy, that is, the data stream is divided into small batches, and then processed by the computing engine one by one.

B. CLUSTERING METHODS

The object we observe is mainly a clustering object, so we need to study and choose an excellent clustering method. Traditional data clustering can be divided into partitioning methods, hierarchical methods, density-based methods, grid-based methods, etc [23]. Grid-based clustering methods, such as STING [24], ignore the individual differences of the observed objects, and only cluster by density information in each partition of the data without repeated individual comparisons, which help to achieve an extremely fast speed.

In this article, we will use density-based clustering for data preprocessing to filter more useful data information. The high density data (within the region) belong to the same cluster is the basic idea of density-based clustering algorithm [6]. Thereby, clusters are considered as dense regions of objects and are separated by sparse regions with low density in the data space. Density-based clustering methods, such as DBSCAN [25], OPTICS [26] and PreDeCon [27], are able to discover arbitrary-shaped clusters and don’t require the predefined number of clusters. Although the performance of DBSCAN in processing high-dimensional data is not very good, it is more suitable for two-dimensional data, and its cluster’s shape is arbitrary [28] instead of the classic cluster produced by K-means method which is limited. The pattern composed of trajectory data is also an uncertain shape. Based on this, density-based clustering is very suitable for trajectory data [29].

C. PATTERN DETECTION METHODS

Today’s co-movement pattern detection has many recognition methods [15]. These methods generate different clusters by redefining the clusters. For these five patterns (flock [10], convoy [11], swarm [3], group [12], and platoon [13]) we can first divide them into two categories, global and local. According to the basic clustering method which is based on distance or density, each major category is further divided into two subcategories. It is not difficult to find that in real life, the same starting point and ending point often have multiple paths, that is, the user’s intermediate state is likely to be different, and their movement patterns are not always the same. From the perspective of clustering, their path points cannot always form a cluster.

Therefore, global clustering methods such as flock and convoy require global clustering to keep forming clusters, which is too restrictive and does not meet reality. For the actual trajectory data, we choose local clustering method, which does not require the continuous formation of clusters in the whole process.

As mentioned before, the clustering in trajectory data is irregular [30]. It is more appropriate to choose the density-based approach since the real cluster’s shape is arbitrary. In this way, we only need to choose from swarm and platoon. It is not difficult to find that the parameters of swarm are more streamlined and the constraints are smaller, which is more suitable for real-time trajectory data. Swarm is a destination-oriented pattern detection that focuses on the overall forward trend of moving objects and ignores local singularities [23]. Traditional pattern detection such as group, convoy can only detect regular, long, and continuous co-movement behavior, which is obviously contrary and not applicable to practical applications. What we know is no matter how many candidate paths a driver can choose, the destination is the same, which is very consistent with the destination-oriented swarm mode. We pay attention to the distribution of object final convergence points or clusters on all road networks. Swarm is simple and easy to implement and fits the actual situation, but its time complexity is particularly high, at an exponential level, because of the selection of its basic clustering method.

To this end, we extract the main idea of STING which is just focus the number of points in each partition, and combine it with softmax we proposed in this article to perform a lightweight pruning strategy on the data in order to reduce the processing time cost from the perspective of cardinal. Then, through the historical state transition matrix proposed in this paper, the swarm is transplanted from a single machine to a distributed environment to speed up the swarm pattern enumeration further.

III. PRELIMINARIES

For co-movement pattern detection, it can be viewed from two perspectives of space and time clustering. The first is spatial clustering, we only perform clustering on the euclidean space from a specific time point. Then we cluster on the clustering results at this series of points, that is, pattern detection.

This section first introduces swarm’s density-based clustering method, and then introduce the swarm mining method. We list the notations to be used as well as their definitions in Table 1.

A. DENSITY-BASED CLUSTERING

In order to get closer to the characteristics of realistic trajectory data and find clusters of arbitrary shapes, we use
TABLE 1. Summary of notations.

| Notations | Definitions |
|-----------|-------------|
| DS        | Data source |
| \(C_i\)  | A moving object \(i\) |
| \(p_i\)  | The \(i\) th point in the trajectory data |
| \(T_{pi} = < p_1, p_2, ..., p_{n_i} >\) | The \(i\) th trajectory data set |
| \(|T_{pi}|\) | Number of points over trajectory data |
| \(S_i\)  | Grid id |
| \(SP_i = \{< O_{j}, p_k, \ldots, \ldots, p_{n_i} >\}\) | The snapshot taken at time \(i\) contains the GPS information of each object at the current time |
| \(C_i\)  | Cluster \(i\) |
| \(|C_i|\) | The number of points in cluster \(i\) |
| \(K\)     | Cluster formation in consecutive \(K\) times |
| \(K^*\)   | Clusters are formed in a total of \(K\) times, no continuous |
| \(L\)     | No cluster formation in the longest \(L\) time |
| \(\epsilon\) | Distance threshold between trajectory data |
| \(\text{minPts}\) | The minimum number of points required for clustering |
| \(N_{\epsilon}(o)\) | The \(\epsilon\)-neighborhood of an object \(o\) |

DBSCAN is a typical density-based clustering. DBSCAN algorithm requires the user to input two parameters: one is the radius \(\epsilon\), which represents the range of the neighbourhood centred on the given point \(p_i\). The other one is the minPts (minimum number of points) in the neighbourhood centred on point \(p_i\). If it is satisfied that the number of points in the neighbourhood with the point \(p_i\) as the centre and the radius \(\epsilon\) is not less than \(\text{minPts}\), the point \(p_i\) is called the core point.

In real-time processing, given a data set \(P = \{p_1, p_2, \ldots, p_{n_i}\}\), we calculate the distance between point \(p_i\) and all the other points in \(S_i\), and put points which the distance is not larger than \(\epsilon\) into the same set \(C_i = \{p_1, p_2, \ldots, p_{|C_i|}\}\). If \(|C_i| \geq \text{minPts}\), a cluster is formed. The clusters are expanded by recursively accessing the unvisited points in \(C_i\). Finally, points that are not visited after recursion are marked as noise point.

Algorithm 1 shows the algorithm DBSCAN. Initially, all points are unvisited. When the algorithm is completed, all points are assigned to the corresponding cluster or marked as noise. In line 6, the algorithm computes the \(\epsilon\)-neighborhood of each unvisited point. If the point is core point, the algorithm will continue to execute, otherwise the point is marked as noise (line 22). In lines 9~20, the algorithm computes the density-connected set of a point. In this process, lines 10~12 computes the directly density-reachable points and adds them to the current cluster (lines 13~16).

Algorithm 1 DBSCAN

Input: A dataset \(D\) containing \(n\) objects, radius \(\epsilon\), neighborhood density threshold \(\text{minPts}\).

Output: Clusters set \(C\).

1. Mark all objects in \(D\) as unvisited.
2. Initialize clusters set \(C\).
3. for \(o \in D\) do
   4. if \(o\) is unvisited then
      5. Mark \(o\) as visited.
      6. if \(|N_{\epsilon}(o)| \geq \text{minPts}\) then
         7. Initialize a cluster \(c\) and add \(o\) to \(c\).
         8. Insert objects in \(N_{\epsilon}(o)\) into the queue \(Q\).
         9. for \(p \in Q\) do
            10. if \(p\) is unvisited then
                11. Mark \(p\) as visited.
                12. if \(|N_{\epsilon}(p)| \geq \text{minPts}\) then
                   13. for \(X \in N_{\epsilon}(p)\) do
                       14. Insert \(X\) into \(Q\).
                       15. Add \(X\) to \(c\).
                16. end for
            17. end if
            18. Remove \(p\) from the queue \(Q\).
            19. end if
         20. end for
      21. else
         22. Mark \(o\) as noise.
      23. end if
   24. end if
   25. Add \(c\) to \(C\).
26. end for

return \(C\)

B. SWARM MINING

Swarm is a group of moving objects containing at least \(\text{min}_o\) individuals who are in the same cluster for at least \(\text{min}_t\) timestamp snapshots. If we denote this group of moving objects as \(O\) and the set of these timestamps as \(T\), a swarm is a pair \((O, T)\) that satisfies the above constraints [3]. Specially, the timestamps in \(T\) are not required to be consecutive.

Algorithm 2 shows the implementation code of Swarm. When visiting the object with objectset \(O\), we first check whether it can pass the data pruning (lines 2~3). Next, we check whether the current object can pass the backward pruning. In the function BackwardPruning (lines 19~23), we generate \(O'\) by adding a new object \(o\) (\(o < o_{last}\)) into \(O\). Then we check whether \(o\) is in the same cluster as other objects in \(O\). If so, current objectset \(O\) cannot pass the backward pruning. After both pruning, we will visit all the objects in the DFS order (lines 7~14). (1) For an object in objectset \(O'\), we generate its maximal timeset in the function GenerateMaxTime (lines 25~31). (2) For Forward Closure Checking we use a variable \(\text{forward} \_\text{closure}\) to record whether any object in objectset \(O'\) has the same maximal timeset size as \(T_{max}(O)\). Finally, if the object passes Forward Closure Checking and \(|O| \geq \text{min}_o\), we can immediately output \((O, T_{max}(O))\) as a closed swarm (lines 15~16).

In this paper, before we perform swarm pattern recognition, we need to find the valid data to reduce subsequent calculation Meanwhile maintaining the pattern mining's
accuracy rate. Drawing on the softmax [31] layer of deep learning, we designed a differentiation function to highlight the dense area. We set a judgment line parameter $J$ as a specific density judgment standard, replacing the traditional empirical method of setting the density threshold, to find out of the areas that are most likely to form a swarm. Since the trajectory data has not only spatial characteristics but also time characteristics, we take the context of time into consideration. The historical state transition matrix is introduced to save the density of each region at the previous time. The historical state is decided by the historical matrix coefficient. The historical matrix coefficient $\forall Y$ decide how much historical information affects the density area’s judgment of the next time through adopting a quantitative standard.

### IV. DISTRIBUTED-SWARM

The Distributed-swarm framework, implementing a real-time swarm pattern detection in a distributed environment, can be divided into two steps. As shown in Fig. 2, the first phase is responsible for picking valid data (VDM for short), includes congested grid detection and pruning operation, which judges the dense area and marks it for subsequent processing. The second phase is the refinement processing part, which is swarm mining, mainly for pattern detection on GPS track data generated by the dense area.

#### A. VALID DATA MINING

In order to improve the data hit rate, it is required to find out the data points that have a higher probability of becoming the co-movement behaviour’s points and delete the unrelated outlier data. We use softmax difference function to prune the data, and use the historical state transition matrix to save the previous state information.

We employ Apache Flink, a distributed real-time processing platform to pre-process the trajectory data. First, we divide the city map into a $N \times N$ grids and set up a CG matrix (Congested Grid) of size $N \times N$ to monitor the entire network. Each element $i$ of the CG matrix is the state monitoring of the area $g_i$. It can reflect the density area of the state. We hash the partial trajectory data to the corresponding grid $g_i$, the formula is:

$$i = \left\lceil \frac{\text{lon} - \text{lon}_0}{\text{gridWidth}} + 1 \right\rceil \cdot \left\lceil \frac{\text{lat} - \text{lat}_0}{\text{gridLength}} + 1 \right\rceil;$$

$$\text{gridWidth} = \frac{N}{\text{MapWidth}};$$

$$\text{gridLength} = \frac{\text{MapLength}}{N};$$

The trajectory data we received is limited to a certain range, and we find the longitude and latitude range that can contain these data. $\text{lat}_0, \text{lon}_0$ represent the minimum latitude and longitude of the range.

After we distributed the trajectory data to the grid map in real time, we try to evaluate the regional density.

#### Algorithm 2 Swarm

**Input:** A dataset $O$ containing $n$ objects, maximal timeset $T_{\text{max}}$ of $O$, latest object $o_{\text{last}}$ added into $O$, minimum threshold parameters $\text{min}_o$ and $\text{min}_t$, number of objects in database $|O_{\text{DB}}|$, number of timestamps in database $|T_{\text{DB}}|$, clustering snapshots at each timestamp $C_{\text{DB}}$.

**Output:** $(O, T_{\text{max}})$ if it is a closed swarm.

1: Data pruning.
2: if $|T_{\text{max}}| < \text{min}_t$ then
3: return;
4: end if
5: Backward pruning.
6: if BackwardPruning($o_{\text{last}}, O, T_{\text{max}}, C_{\text{DB}}$) then
7: forward_closure $\leftarrow$ true;
8: for $o \leftarrow o_{\text{last}} + 1$ to $|O_{\text{DB}}|$ do
9: $O' \leftarrow O \cup \{o\}$;
10: $T_{\text{max}}' \leftarrow \text{GenerateMaxTime}(o, o_{\text{last}}, T_{\text{max}}, C_{\text{DB}})$;
11: if $|T_{\text{max}}| = |T_{\text{max}}'|$ then
12: forward_closure $\leftarrow$ false;
13: end if
14: $\text{Swarm}(O_{\text{new}}, T_{\text{new}}, o, \text{min}_o, \text{min}_t, |O_{\text{DB}}|, |T_{\text{DB}}|, C_{\text{DB}})$;
15: if forward_closure $\&\& |O| \geq \text{min}_o$ then
16: return pair $(O, T)$ as a closed swarm;
17: end if
18: end for
19: function BackwardPruning($o_{\text{last}}, O, T_{\text{max}}, C_{\text{DB}}$)
20: for $o \notin O$ $\&\& o < o_{\text{last}}$ do
21: if $C_t(o) \subseteq C_t(o'), \forall t \in T_{\text{max}}$ then
22: return false;
23: end if
24: end for
25: function GenerateMaxTime($o, o_{\text{last}}, T_{\text{max}}, C_{\text{DB}}$)
26: for $t \in T_{\text{max}}$ do
27: if $C_t(o) \cap C_t(o_{\text{last}}) \neq \emptyset$ then
28: $T_{\text{max}}' \leftarrow T_{\text{max}}' \cup t$;
29: end if
30: end for
31: end function

#### 1) SOFTMAX MATRIX

Softmax is a commonly used and essential function in deep learning. It is used to find the object with the highest probability and mainly used in classification problems. Furthermore, we can also regard the judgment of dense grid area as a classification problem.

When it comes to city congestion, it is not always crowded in places with a lot of cars. For example, some small streets also have a small number of users, but they do cause severe
We can use the parameter $J$ and this formula to solve a key question, namely how many objects in a region can become a dense region. Through the formula, we can normalize all regions to a very small 0-1 range, and through $J$, we can quantitatively demarcate a region as a dense region, without having to consider the specific quantity. The quantity metric of the area may change with time, place, and situation, and depends on the surrounding conditions. Using this formula, the density area can be quantitatively determined in conjunction with the global road network.

2) HISTORICAL MATRIX

It is inevitable for us to get faster processing speed by deleting data. However, the speed increase brought by the ambiguous deletion of data will result in a decrease in the pattern hit rate. It is wrong to filter the data only by the density of the grid without considering the specific conditions of the trajectory data. In order to consider the time characteristics, we need to consider the historical movement states of each object on the road network, so we introduced the historical state transition coefficient.

In addition, the framework of this paper is running under distributed conditions. There is a pronounced problem with the data after partitioning, that is, the state among the nodes is share nothing, and it is difficult to perceive the data of each other between the nodes. In the detection stage of dense grids, we only count the number of sampling points of each grid $g_i$ to judge whether the grid is dense. And we do not save specific state information, and the data information of the surrounding grids is not perceived. This is bad for clusters across grids. For example, the two clusters formed by $g_1$, $g_2$, $g_5$, $g_6$ and $g_{11}$, $g_{15}$ in Fig. 3, they can form a high-density area as a whole, but it is difficult to reach the density threshold when allocated to their respective grids.

We consider that the trajectory data is fluid and non-transitional, which means that a cluster (such as the cluster...
formed by \( g_{11}, g_{15} \) in Fig. 3) is difficult to jump out of several grids at once or disappear directly. We inherited the information of the clusters across the grid to the next moment, and added additional information for it. Such additional information will make the CG matrix discover that there is a cross-grid matrix for pattern recognition. All in all, the current clustering across grid will definitely have an effect on future clustering. All we need to do is to apply the current state to the next.

The judgment of whether a grid is a dense grid cannot be limited to the current time, but also needs to consider the historical time. Here we set a historical state transition matrix \( H(\text{History}) \) and set a weight value \( W \). The definition is as follows:

\[
\begin{align*}
H(j) &= G[i - 1]; \\
G[i] &= (1 - W) \cdot f(g_i) + W \cdot H(j);
\end{align*}
\]

We have also considered expanding the neighbouring grids around the top, bottom, left, and right of the current dense grid \( g_c \). Experience has shown that a clustering across grid is difficult to exceed one grid, so we only need to expand the structure by one layer. We mark the dense grid as follows:

\[
\begin{align*}
G[(c - N) > 0?(c - n) : c] &= 1; \\
G[c + N > N^2 - 1?c : c + N] &= 1; \\
G[c \mod N == 0?c : c - 1] &= 1; \\
G[(c + 1) \mod N == 0?c : c + 1] &= 1; \\
G[c] &= 1;
\end{align*}
\]

In fact, we found through experiments that the latter way of expanding the neighbour grid greatly increases the amount of data processing. This method requires saving four copies of data except itself. There is much redundant information since it is a cross shape. It is easy to overlap with other grids, and the detection to swarm is far from efficient than using the history matrix.

B. REAL-TIME SWARM MINING

The Distributed-swarm will firstly make statistics on the data, that is, the congested grids detection part of Fig. 4. If pattern recognition is performed on the statistical data at this time, it will cause two problems. The first one is that the real-time characteristics of the data are reduced because it must be statistics first and then do further process. The second one is the waste of system resources caused by secondary processing of data.

Through the analysis of the actual trajectory data, in fact, even if we repeat the processing of the data, we can only perform secondary processing on the data in a small part of the dense area, most of the data will be discarded. Although all input data is processed at the first processing, we only count the quantity information without doing any other calculations. The calculation resources used in this step are extremely low, and in the case of experimental testing, the speed is 60% faster than the direct processing and analysis of all data.

We have solved the problem of high latency caused by secondary processing of data through the pruning method mentioned in section IV-A. The sampling of trajectory data is a kind of high-frequency sampling which the sampling interval is very short. As for the actual taxi operation mode, we don’t need its data every moment to judge its movement mode. The trajectory data is continuous and long-term data. Even if we are missing a part of the data, we can still see that it is roughly but accurate.

V. EXPERIMENT

A. SETUP

1) EXPERIMENT ENVIRONMENT

In this section, we evaluate the real-time performance, hit rate, and pruning performance of our proposed framework and method. All the experiments are carried out on 12 computers with 1GB DDR4 RAM and 500GB SSD.
The framework runs on Ubuntu with version 16.04. All algorithms are implemented in Java and run on Flink 1.9.

2) DATASET
In order to verify the performance of our framework, we randomly extract one hour of trajectory data of one day for real-time detection. This dataset contains 2310697 pieces of trajectory data of Didi in Chengdu on November 1, 2016. The data organization format is "<vehicalID, orderID, timestamp, lon, lat >". VehicalID is linked with the driving vehicle, orderID is used to confirm which order was generated by this trajectory, timestamp is the charging time, and <lon, lat> is the charging coordinate of the electric vehicle.

3) PARAMETER SETTINGS & COMPARISON METHOD
The experiment mainly includes the adjustment of two parameters, judgmentline (J) and historical state coefficient (W).

We use the method of controlling variables. When the value of J or W is unchanged, we adjust another parameter and check the influence of the parameter on the experiment by the evaluations mentioned in the next part. It is worth noting that, for the parameter J, we also debug the parameter J from two granularity (0.001 and 0.0001) to see its specific impact on the experimental results.

At the end of each experiment, we set up a native swarm as a comparison group of experiments to highlight the superiority of our framework.

4) EVALUATION METRICS
Our fundamental requirement is to improve the accuracy of mining for swarm patterns while ensuring real-time performance. There are two core contents here, one is real-time and the other is accuracy. Our job is to adjust the two parameters J and W in order to examine the changes in the corresponding three indicators. Specifically, the first thing to examine is the data pruning result. Here we define the data loss rate θ and quantify the pruning ability of D-swarm:

\[ \theta = (1 - \frac{\text{Total pruned data}}{\text{Total data}}) \times 100\% \]; (2)

We have defined the pattern hit rate η for whether the data after pruning can accurately reflect the actual motion pattern;

\[ \eta = \frac{\text{Total patterns number by } D - \text{swarm}}{\text{Total patterns number by swarm}} \times 100\% \]; (3)

And the ratio of the processing speed of this framework to the native swarm \( \nu \),

\[ \nu = \frac{\text{Processing time by } D - \text{swarm}}{\text{Processing time by swarm}} \times 100\% \]; (4)

The effect we want to achieve is that \( \theta \) and \( \eta \) are as large as possible and \( \nu \) is as small as possible, so as to achieve real-time performance without losing accuracy. The following Table 2 is our summary of the evaluation metrics.

**TABLE 2. The notations.**

| Evaluation Metric         | Notation | Definition                                      |
|---------------------------|----------|-------------------------------------------------|
| Data loss rate            | \( \theta \) | Proportion of pruned data, used to indicate pruning performance. |
| Hit rate                  | \( \eta \) | The degree of matching between the detected pattern and the actual pattern. |
| Ratio of Processing speed | \( \nu \) | Demonstrate the superiority of the processing speed of our mode. |

B. RESULTS

1) NATIVE SWARM PERFORMANCE
The native swarm took 300538ms to process all 2310697 trajectory data when running on the local standalone and finally found 255 swarm patterns. We will use these data as reference data for subsequent experiments in this framework.

2) DATA PRUNING PERFORMANCE
The experiment found that the D-swarm framework performed a pruning operation of about 60% on the data, and finally got a hit rate of more than 76% and a speed increase of 70% after debugging the parameters J and W. The pruning effect achieved by using 6-core and 12-thread, and the reference experiment is a swarm mode detection program that is run by a single machine and single thread.

3) EFFECT OF J
In order to show the influence of parameter J on the mining of swarm mode, we keep the parameter W unchanged (\( W = 0.5 \)), and adjust the parameter J with two granularities of size. Here, the two particle sizes refer to the variation range of J, one is 0.001 to 0.005, and the other is 0.0001 to 0.0009.

Fig. 6 shows the corresponding data processing volume, number of pattern hits, processing speed changes, and native swarm data processing volume when adjusting the parameter J from the two granularities. We take native swarm 's number of output patterns and processing speed as reference lines.

In Fig. 6, we increase the filtering ability of the data by increasing J to examine the influence of the parameter J on pattern recognition. The larger the value of J, the more data will be filtered, which will undoubtedly promote the processing speed of pattern recognition fast. However, discarding too much data will result in a decrease in the pattern hit rate.

According to Fig. 6 (\( \text{Precision@J} = 0.001 \)), we analyze from a coarse-grained perspective that parameter J should take a smaller value, so as to ensure a faster processing speed and a high hit rate. Specifically, when J is greater than 0.002, each curve tends to be flat, so blindly increasing the parameter J is meaningless. Similarly, it is not difficult to find that when J takes the minimum value of 0.001, it seems that the hit rate is very high, but the speed is much slower than J = 0.002, which is close to 20% slower. We have rejected to make J as large as possible, so what about trying to make it as small as possible?
To this end, we study the impact on the hit rate and speed from fine-grained in Fig. 6 (Precision@ $\mathcal{J} = 0.0001$) when $\mathcal{J}$ is as small as possible. The results show that when $\mathcal{J}$ is as close to zero as possible, there will be no more prominent advantages, and the curve is still relatively flat. Furthermore, we found that when $\mathcal{J}$ is as small as possible, its ability to prune data does not weaken with the decrease of $\mathcal{J}$. And the overall pruning rate of data is always close to 30%, which is due to the previous operation of the softmax matrix.

We can take the smallest $\mathcal{J}$ as possible as we can, but this will sacrifice some of the speed advantages. Moreover, our primary purpose is to dig the correct pattern while maintaining real-time. Real-time is our first demand. Therefore, we define to take a sufficiently small $\mathcal{J}$ (sufficiently means small enough, not infinitely small. $\mathcal{J}$ which is too small will cause a reduction in processing speed). For this experiment, we take $\mathcal{J} = 0.002$, which can still achieve 80% hit rate and 70% speed increase.

4) EFFECT OF $\mathcal{W}$

D-swarm has two main parameters, we have already discussed about $\mathcal{J}$. Now keep $\mathcal{J} = 0.002$, and observe the effect of parameter $\mathcal{W}$ on the experimental results. It is easy to find that when $\mathcal{W}$ is in the range of 0.6 to 0.8, while the speed advantage is the largest, the hit rate is also stable above 85% from Fig. 7.

For the range of 0.1–0.5, the effect has a fluctuation of about 10%. In the 0.8–0.9 high-value range, there will be a steep curve, the speed advantage greatly reduced about 20%. But only 5% of the increase in hit rate. Therefore, we should choose a sufficiently large value for $\mathcal{W}$, but it should not be too large.

Given a very small $\mathcal{W}$, cross-cluster will be difficult to find as mentioned in section IV-A. If $\mathcal{W}$ is too small, the retention effect on historical state information will be reduced a lot. However, if $\mathcal{W}$ is too large, historical information occupies almost all the state information. It will make the framework
process more data at the last moment, which caused a delay and the rate slowed down accordingly.

VI. CONCLUSION AND FUTURE WORK

Swarm is an effective pattern recognition method with clear structure, but the existing methods still have some limitations in distributed environment and real-time processing. In this paper, we propose a real-time pattern detection model based on density clustering, named D-swarm. We introduce Softmax to calculate the density in each grid to find dense grids and prune irrelevant data, so as to improve the processing speed of the framework. We design a historical state transition matrix to retain the historical information of the previous stage and improve the accuracy of pattern detection. We implement this framework on Apache Flink. Experiments on a large number of real data sets show that our framework has good real-time performance and accuracy in real-time pattern detection.

In the future, how to achieve more efficient and stable power analysis is still a problem worth studying. We will try different pattern detection methods and use the detected patterns for behavior prediction.

REFERENCES

[1] P. Kalnis, N. Mamoulis, and S. Bakiras, “On discovering moving clusters in spatio-temporal data,” in Proc. Int. Symp. Spatial Temporal Databases, Springer, 2005, pp. 364–381.
[2] P. Laube and S. Imfeld, “Analyzing relative motion within groups of trackable moving point objects,” in Proc. Int. Conf. Geograph. Inf. Sci. Springer, 2002, pp. 132–144.
[3] Z. Li, B. Ding, J. Han, and R. Kays, “Swarm: Mining relaxed temporal moving object clusters,” Dept. Comput. Sci., Univ. Illinois Urbana-Champaign, Champaign, IL, USA, Tech. Rep., 2010.
[4] A. Contreras-Valdes, J. P. Amezquita-Sanchez, D. Granados-Lieberman, and M. Valtierra-Rodriguez, “Predictive data mining techniques for fault diagnosis of electric equipment: A review,” Appl. Sci., vol. 10, no. 3, p. 950, Feb. 2020.
[5] F. Giannotti, M. Nanni, F. Pinelli, and D. Pedreschi, “Trajectory pattern mining,” in Proc. 13th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, 2007, pp. 330–339.
[6] Z. Yu, “Trajectory data mining: An overview,” ACM Trans. Intell. Syst. Technol., vol. 6, no. 3, pp. 1–41, 2015.
[7] Y. Lv, Y. Duan, W. Kang, Z. Li, and F.-Y. Wang, “Traffic flow prediction with big data: A deep learning approach,” IEEE Trans. Intell. Transp. Syst., vol. 16, no. 2, pp. 865–873, Apr. 2015.
[8] S. Ishak and H. Al-Deek, “Performance evaluation of short-term time-series traffic prediction model,” J. Transp. Eng., vol. 128, no. 6, pp. 490–492, 2002.
[9] M. Butenhof, F. Burkert, F. Schmidt, S. Hinz, D. Hartmann, A. Knießl, A. Borrmann, and B. Sirmacek, “Integrating pedestrian simulation, tracking and event detection for crowd analysis,” in Proc. IEEE Int. Conf. Comput. Vis. Workshops (ICCV Workshops), Nov. 2011, pp. 150–157.
[10] J. Gudmundsson and M. van Kreveld, “Computing longest duration flocks in trajectory data,” in Proc. 14th Annu. ACM Int. Symp. Adv. Geograph. Inf. Syst., 2006, pp. 35–42.
[11] H. Jeung, M. L. Yu, X. Zhou, C. S. Jensen, and H. T. Shen, “Discovery of convoys in trajectory databases,” 2010, arXiv:1002.0963.
[12] Y. Wang, E.-P. Lim, and S.-Y. Hwang, “Efficient mining of group patterns from user movement data,” Data Knowl. Eng., vol. 57, no. 3, pp. 240–282, Jun. 2006.
[13] Y. Li, J. Bailey, and L. Kulik, “Efficient mining of platoon patterns in trajectory databases,” Data Knowl. Eng., vol. 100, pp. 167–187, Nov. 2015.
[14] Y. Zheng, Y. Liu, J. Yuan, and X. Xie, “Urban computing with taxicabs,” in Proc. 13th Int. Conf. Ubiquitous Comput., 2011, pp. 89–98.
[15] Q. Fan, D. Zhang, H. Wu, and K.-L. Tan, “A general and parallel platform for mining co-movement patterns over large-scale trajectories,” Proc. VLDB Endowment, vol. 10, no. 4, pp. 313–324, Nov. 2016.
[16] S. Park and Y. B. Park, “Photovoltaic power data analysis using hierarchical clustering,” in Proc. Int. Conf. Web Inf. Syst. Eng. Springer, 2021, pp. 619–635.
[17] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tsoumas, “Apache flink: Stream and batch processing in a single engine,” Bull. IEEE Comput. Soc. Tech. Committee Data Eng., 2015, vol. 36, no. 4.
[18] D. Yang, J. Guo, Z.-J. Wang, Y. Wang, J. Zhang, L. Hu, J. Yin, and J. Cao, “FastPM: An approach to pattern matching via distributed stream processing,” Inf. Sci., vol. 453, pp. 263–280, Jul. 2018.
[19] B. Gedik, H. Andrade, K.-L. Wu, P. S. Yu, and M. Doo, “SAFE: The systems declarative stream processing engine,” in Proc. ACM SIGMOD Int. Conf. Manage. Data, 2008, pp. 1123–1134.
[20] D. G. Murray, F. McSherry, R. Isaacs, M. Isard, P. Barham, and M. Abadi, “Naiad: A timely dataflow system,” in Proc. 24th ACM Symp. Operating Syst. Princ., Nov. 2013, pp. 439–455.
[21] L. Chen, Y. Gao, Z. Fang, X. Miao, C. S. Jensen, and C. Guo, “Real-time distributed co-movement pattern detection on streaming trajectories,” Proc. VLDB Endowment, vol. 12, no. 10, pp. 1208–1220, Jun. 2019.
[22] H. L. Nguyen, Y.-K. Woon, and W.-K. Ng, “A survey on data stream clustering and classification,” Knowl. Inf. Syst., vol. 45, no. 3, pp. 535–569, 2015.
[23] W. Wang, J. Yang, and R. Muntz, “Sting: A statistical information grid approach to spatial data mining,” in Proc. VLDB, vol. 97, Princeton, NJ, USA: Citeseer, 1997, pp. 186–195.
[24] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, “A density-based algorithm approach to spatial data mining,” in Proc. KDD, vol. 96, no. 34, 1996, pp. 226–231.
[25] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, “OPTICS: Ordering points to identify the clustering structure,” ACM SIGMOD. Rec., vol. 28, no. 2, pp. 49–60, Jun. 1999.
[26] C. Bohm, K. Kailing, H.-P. Kriegel, and P. Kroger, “Density connected clustering and classification,” in Proc. VLDB Endowment, vol. 4, no. 2, pp. 196–207, Jun. 2011.
[27] J. Sander, M. Ester, H.-P. Kriegel, and X. Xu, “Density-based clustering in spatial databases: The algorithm GDBSCAN and its applications,” in Proc. 4th IEEE Int. Conf. Data Mining (ICDM), Nov. 2004, pp. 27–34.
[28] J. Sander, M. Ester, H.-P. Kriegel, and X. Xu, “Density-based clustering in spatial databases: The algorithm DBSCAN and its applications,” Data Mining Knowl. Discovery, vol. 2, no. 2, pp. 169–194, Jun. 1998.
[29] Y. Chen and L. Tu, “Density-based clustering for real-time stream data,” in Proc. 13th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, 2007, pp. 133–142.
[30] J.-G. Lee, J. Han, and K.-Y. Whang, “Trajectory clustering: A partition-and-group framework,” in Proc. ACM SIGMOD Int. Conf. Manage. Data, 2007, pp. 593–604.
[31] A. Joulin, M. Cissé, D. Grangier, and H. Jégou, “Efficient softmax approximation for GPUs,” in Proc. Int. Conf. Mach. Learn., 2017, pp. 1302–1310.

TIAO QIAN received the bachelor’s degree from Nanjing University, China, in 2009. He has been working in the smart grid field, since 2009. His research interests include software architecture design and information visualization on power dispatching automation systems.

SHIMING SUN received the B.Eng. degree in computer technology from the Nanjing Institute of Technology, Nanjing, China, in 2003, and the M.S. degree in computer software engineering from Hohai University, Nanjing, in 2006. He is currently a Senior Engineer with the Department of Basic Research and Development, NARI Group Corporation (State Grid Electric Power Research Institute), Nanjing. His research interests include electric power automation systems, recommender systems, and big data analysis.

XIN SHAN received the bachelor’s degree. His main research interests include the development and engineering of wide-area measurement systems and integrated alarm applications.

XUEYUN WEI received the M.S. degree from the University of Science and Technology of China, China, in 2013. He is working in the smart grid field. His research interests include anti-maloperation strategy and remote control technique on power dispatching automation systems.

CHUNLIANG TAI received the M.S. degree from the Nanjing University of Posts and Telecommunications, China, in 2017. He is working in the smart grid field. His research interests include software architecture design on power dispatching automation systems and analysis of software applications.

CHAO LIU received the bachelor’s degree from the Nanjing University of Posts and Telecommunications, in 2010. He is working in the smart grid field. His research interests include software architecture design of recommendation systems and power dispatching automation systems.