Swarm: Mining Relaxed Temporal Moving Object Clusters

Zhenhui Li† Bolin Ding† Jiawei Han† Roland Kays‡
† University of Illinois at Urbana-Champaign, US
‡ New York State Museum, US
Email: {zli28, bding3, hanj}@uiuc.edu, rkays@mail.nysed.gov

ABSTRACT

Recent improvements in positioning technology make massive moving object data widely available. One important analysis is to find the moving objects that travel together. Existing methods put a strong constraint in defining moving object cluster, that they require the moving objects to stick together for *consecutive* timestamps. Our key observation is that the moving objects in a cluster may actually diverge temporarily and congregate at certain timestamps.

Motivated by this, we propose the concept of *swarm* which captures the moving objects that move within arbitrary shape of clusters for certain timestamps that are possibly non-consecutive. The goal of our paper is to find all discriminative swarms, namely *closed swarm*. While the search space for closed swarms is prohibitively huge, we design a method, ObjectGrowth, to efficiently retrieve the answer. In ObjectGrowth, two effective pruning strategies are proposed to greatly reduce the search space and a novel closure checking rule is developed to report closed swarms on-the-fly. Empirical studies on the real data as well as large synthetic data demonstrate the effectiveness and efficiency of our methods.

1. INTRODUCTION

Telemetry attached on wildlife, GPS set on cars, and mobile phones carried by people have enabled tracking of almost any kind of moving objects. Positioning technologies make it possible to accumulate a large amount of moving object data. Hence, analysis on such data to find interesting movement patterns draws increasing attention in animal studies, traffic analysis, and law enforcement applications.

A useful data analysis task in movement is to find moving object clusters, which is a loosely defined and general task to find a group of moving objects that are traveling together sporadically. The discovery of such clusters has been facilitating in-depth study of animal behaviors, routes planning, and vehicle control. A moving object cluster can be defined in both spatial and temporal dimensions: (1) a group of moving objects should be geometrically close to each other, and (2) they should be together for at least some minimum time duration.

Proceedings of the VLDB Endowment, Vol. 3, No. 1

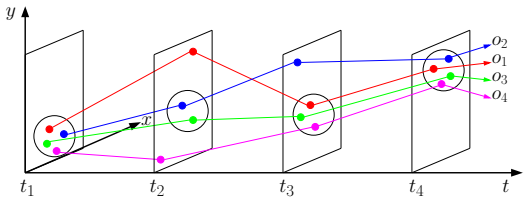


Figure 1: Loss of interesting moving object clusters in the definition of moving cluster, flock and convoy.

There have been many recent studies on mining moving object clusters. One line of study is to find moving object clusters including moving clusters [14], flocks [10, 9, 4], and convoys [13, 12]. The common part of such patterns is that they require the group of moving objects to be together for at least k *consecutive* timestamps, which might not be practical in the real cases. For example, if we set $k = 3$ in Figure 1, no moving object cluster can be found. But intuitively, these four objects travel together even though some objects temporarily leave the cluster at some snapshots. If we relax the *consecutive* time constraint and still set $k = 3$, o_1, o_3 and o_4 actually form a moving object cluster. In other words, enforcing the *consecutive* time constraint may result in the loss of interesting moving object clusters.

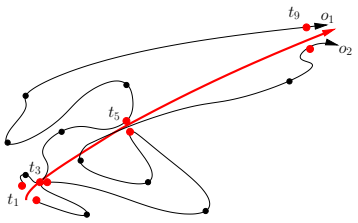


Figure 2: Loss of interesting moving object clusters in trajectory clustering.

Another line of study of moving object clustering is trajectory clustering [20, 6, 8, 17], which puts emphasis on geometric or spatial closeness of object trajectories. However, objects that are essentially moving together may not share similar geometric trajectories. As illustrated in Figure 2, from the geometric point of view, these two trajectories may be rather different. But if we pick the timestamps when they

are close, such as t_1, t_3, t_5 and t_9, the two objects should be considered as traveling together. In real life, there are often cases that a set of moving objects (*e.g.*, birds, flies, and mammals) hardly stick together all the time—they do travel together, but only gather together at some timestamps.

In this paper, we propose a new movement pattern, called *swarm*, which is a more general type of moving object clusters. More precisely, swarm is a group of moving objects containing at least min_o individuals who are in the same cluster for at least 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. Specially, the timestamps in T are not required to be consecutive, the detailed geometric trajectory of each object becomes unimportant, and clustering methods and/or measures can be flexible and application-dependent (*e.g.*, density-based clustering vs. Euclidean distance-based clustering). By definition, if we set $min_o = 2$ and $min_t = 3$, we can find swarm $(\{o_1, o_3, o_4\}, \{t_1, t_3, t_4\})$ in Figure 1 and swarm $(\{o_1, o_2\}, \{t_1, t_3, t_5, t_9\})$ in Figure 2. Such swarms discovered are interesting but cannot be captured by previous moving object cluster detection or (geometry-based) trajectory clustering methods. To avoid finding redundant swarms, we further propose the *closed swarm* concept (see Section 3). The basic idea is that if (O, T) is a swarm, it is unnecessary to output any subset $O' \subseteq O$ and $T' \subseteq T$ even if (O', T') may also satisfy swarm requirements. For example, in Figure 2, swarm $\{o_1, o_2\}$ at timestamps $\{t_1, t_3, t_9\}$ is actually redundant even though it satisfies swarm definition because there is a closed swarm: $\{o_1, o_2\}$ at timestamps $\{t_1, t_3, t_5, t_9\}$.

Efficient discovery of complete set of closed swarms in a large moving object database is a non-trivial task. First, the size of all the possible combinations is exponential (*i.e.*, $2^{|O_{DB}|} \times 2^{|O_{TB}|}$) whereas the discovery of moving clusters, flocks or convoys has polynomial solution due to stronger constraint posed by their definitions based on k *consecutive* timestamps. Second, although the problem is defined using the similar form of frequent pattern mining [1, 11], none of previous work [1, 11, 25, 23, 19, 21] solves exactly the same problem as finding swarms. Because in the typical frequent pattern mining problem, the input is a set of transactions and each transaction contains a set of items. However, the input of our problem is a sequence of timestamps and there is a collection of (overlapping) clusters at each timestamp (detailed in Section 3). Thus, the discovery of swarms poses a new problem that needs to be solved by specifically designed techniques.

Facing the huge potential search space, we propose an efficient method, ObjectGrowth. In ObjectGrowth, besides the *Apriori Pruning rule* which is commonly used, we design a novel *Backward Pruning rule* which uses a simple checking step to stop unnecessary further search. Such pruning rule could cover several redundant cases at the same time. After our pruning rules cut a great portion of unpromising candidates, the leftover number of candidate closed swarms could still be large. To avoid the time-consuming pairwise closure checking in the post-processing step, we present a nice *Forward Closure Checking* step that can report the closed swarms on-the-fly. Using this checking rule, no space is needed to store candidates and no extra time is spent on post-processing to check closure property.

In summary, the contributions of the paper are as follows.

- A new concept, *swarm*, and its associated concept *closed swarm* are introduced, which enable us to find relaxed temporal moving object clusters in the real world settings.
- ObjectGrowth is developed for efficient mining closed swarms. Two pruning rules are developed to efficiently reduce search space and a closure checking step is integrated in the search process to output closed swarms immediately.
- The effectiveness as well as efficiency of our methods are demonstrated on both real and synthetic moving object databases.

The remaining of the paper is organized as follows. Section 2 discusses the related work. The definitions of swarms and closed swarms are given in Section 3. We introduce the ObjectGrowth methods in Sections 4. Experiments testing effectiveness and efficiency are shown in Section 5. Finally, our study is concluded in Section 6. The proofs, pseudo code, and detailed discussions are stated in Appendix.

2. RELATED WORK

Related work on moving object clustering can be categorized into two lines of research: *moving object cluster discovery* and *trajectory clustering*. The former focuses on individual moving objects and tries to find clusters of objects with similar moving patterns or behaviors; whereas the latter is more from a geometric view to cluster trajectories. The related work, especially the ones for direct comparison, will be described in more details in Appendix C.

Flock is first introduced in [16] and further studied in [10, 9, 2]. Flock is defined as a group of moving objects moving in a disc of a fixed size for k *consecutive* timestamps. Another similar definition, *moving cluster* [14], tries to find a group of moving objects which have considerably portion of overlap at any two *consecutive* timestamps. A recent study by Jeung *et al.* [13, 12] propose *convoy*, an extension of flock, where spatial clustering is based on density. Comparing with all these definitions, swarm is a more general one that does not require k *consecutive* timestamps.

Group pattern, defined in [22], is the most similar pattern to swarm pattern. Group patterns are the moving objects that travel within a radius for certain timestamps that are possibly non-consecutive. Even though it considers relaxation of the time constraint, the group pattern definition restricts the size and shape of moving object clusters by specifying the disk radius. Moreover, redundant group patterns make the algorithm exponentially inefficient.

Another line of research is to find trajectory clusters which reveal the common paths for a group of moving objects. The first and most difficult challenge for trajectory clustering is to give a good definition of similarity between two trajectories. Many methods have been proposed, such as Dynamic Time Warping (DTW) [24], Longest Common Subsequences (LCSS) [20], Edit Distance on Real Sequence (EFR) [6], and Edit distance with Real Penalty (ERP) [5]. Gaffney *et al.* [8] propose trajectory clustering methods based on probabilistic modeling of a set of trajectories. As pointed out in Lee *et al.* [17], distance measure established on *whole trajectories* may miss interesting common paths in *sub-trajectories*. To find clusters based on sub-trajectories, Lee *et al.* [17] proposed a partition-and-group framework. But this framework cannot find swarms because the real trajectories of the ob-

jects in a swarm may be complicated and different. Works on subspace clustering [15, 3] can be also applied to find sub-trajectory clusters. However, these works address the issue how to efficiently apply DBSCAN on high-dimensional space. Such clustering technique still cannot be directly applied to find swarm patterns.

3. PROBLEM DEFINITION

Let $O_{DB} = \{o_1, o_2, \ldots, o_n\}$ be the set of all moving objects and $T_{DB} = \{t_1, t_2, \ldots, t_m\}$ be the set of all timestamps in the database. A subset of O_{DB} is called an *objectset* O. A subset of T_{DB} is called a *timeset* T. The *size*, $|O|$ and $|T|$, is the number of objects and timestamps in O and T respectively.

Database of clusters. A database of clusters, $C_{DB} = \{C_{t_1}, C_{t_2}, \ldots, C_{t_m}\}$, is the collection of snapshots of the moving object clusters at timestamps $\{t_1, t_2, \ldots, t_m\}$. We use $C_{t_i}(o_j)$ to denote the set of clusters that object o_j is in at timestamp t_i. Note that an object could belong to several clusters at one timestamp. In addition, for a given objectset O, we write $C_{t_i}(O) = \bigcap_{o_j \in O} C_{t_i}(o_j)$ for short. To make our framework more general, we take clustering as a preprocessing step. The clustering methods could be different based on various scenarios. We leave the details of this step in Appendix D.1.

Swarm and Closed Swarm. A pair (O, T) is said to be a *swarm* if all objects in O are in the same cluster at any timestamp in T. Specifically, given two minimum thresholds min_o and min_t, for (O, T) to be a swarm, where $O = \{o_{i_1}, o_{i_2}, \ldots, o_{i_p}\} \subseteq O_{DB}$ and $T \subseteq T_{DB}$, it needs to satisfy three requirements:

(1) $|O| \geq min_o$: There should be at least min_o objects.

(2) $|T| \geq min_t$: Objects in O are in the same cluster for at least min_t timestamps.

(3) $C_{t_i}(o_{i_1}) \cap C_{t_i}(o_{i_2}) \cap \cdots \cap C_{t_i}(o_{i_p}) \neq \emptyset$ for any $t_i \in T$: there is at least one cluster containing all the objects in O at each timestamp in T.

To avoid mining redundant swarms, we further give the definition of *closed swarm*. A swarm (O, T) is *object-closed* if fixing T, O cannot be enlarged ($\nexists O'$ *s.t.* (O', T) is a swarm and $O \subsetneq O'$). Similarly, a swarm (O, T) is *time-closed* if fixing O, T cannot be enlarged ($\nexists T'$ *s.t.* (O, T') is a swarm and $T \subsetneq T'$). Finally, a swarm (O, T) is a *closed swarm* iff it is both object-closed and time-closed. Our goal is to find the complete set of closed swarms.

We use the following example as a **running example** in the remaining sections to give an intuitive explanation of our methods. We set $min_o = 2$ and $min_t = 2$ in this example.

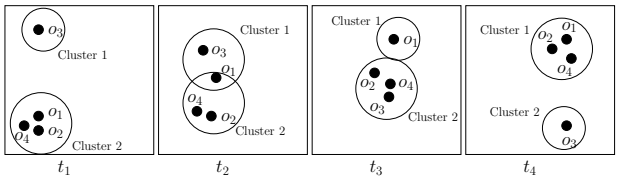


Figure 3: Snapshots of object clusters at t_1 to t_4.

EXAMPLE 1. *(Running Example) Figure 3 shows the input of our running example. There are 4 objects and 4 timestamps ($O_{DB} = \{o_1, o_2, o_3, o_4\}$, $T_{DB} = \{t_1, t_2, t_3, t_4\}$). Each sub-figure is a snapshot of object clusters at each timestamp. It is easy to see that o_1, o_2, and o_4 travel together for most of the time, and o_2 and o_4 form an even more stable swarm since they are close to each other in the whole time span. Given $min_o = 2$ and $min_t = 2$, there are totally 15 swarms: $(\{o_1, o_2\}, \{t_1, t_2\})$, $(\{o_1, o_4\}, \{t_1, t_2\})$, $(\{o_2, o_4\}, \{t_1, t_3, t_4\})$, and so on.*

But it is obviously redundant to output swarms like $(\{o_2, o_4\}, \{t_1, t_2\})$ and $(\{o_2, o_4\}, \{t_2, t_3, t_4\})$ (not time-closed) since both of them can be enlarged to form another swarm: $(\{o_2, o_4\}, \{t_1, t_2, t_3, t_4\})$. Similarly, $(\{o_1, o_2\}, \{t_1, t_2, t_4\})$ and $(\{o_2, o_4\}, \{t_1, t_2, t_4\})$ are redundant (not object-closed) since both of them can be enlarged as $(\{o_1, o_2, o_4\}, \{t_1, t_2, t_4\})$. There are only two closed swarms in this example: $(\{o_2, o_4\}, \{t_1, t_2, t_3, t_4\})$ and $(\{o_1, o_2, o_4\}, \{t_1, t_2, t_4\})$.

Transaction 1	$\{a, b, c\}$
Transaction 2	$\{a, c\}$
Transaction 3	$\{a, c, d\}$
Transaction 4	$\{b, d\}$

(a) FP mining problem

t_1	$\{\{o_1, o_2, o_4\}, \{o_3\}\}$
t_2	$\{\{o_1, o_3\}, \{o_1, o_2, o_4\}\}$
t_3	$\{\{o_1\}, \{o_2, o_3, o_4\}\}$
t_4	$\{\{o_3\}, \{o_1, o_2, o_4\}\}$

(b) Swarm mining problem

Figure 4: Difference between frequent pattern mining and swarm pattern mining

Note that even though our problem is defined in the similar form of frequent pattern mining [11], none of previous work in frequent pattern (FP) mining area can solve exactly our problem. As shown in Figure 4, FP mining problem takes transactions as input, swarms discovery takes clusters at each timestamp as input. If we treat each timestamp as one transaction, each "transaction" is a collection of "itemsets" rather than just one itemset. If we treat each cluster as one transaction, the support measure might be incorrectly counted. For example, if we do so for the example in Figure 4, the support of o_1 is wrongly counted as 5 because it is counted twice at t_2. Therefore, there is no trivial transformation of FP mining problem to swarm mining problem. The difference demands new techniques to specifically solve our problem.

4. DISCOVERING CLOSED SWARMS

The pattern we are interested in here, swarm, is a pair (O, T) of objectset O and timeset T. At the first glance, the number of different swarms could be $(2^{|O_{DB}|} \times 2^{|T_{DB}|})$, i.e., the size of the search space. However, for a *closed* swarm, the following Lemma shows that if the objectset is given, the corresponding maximal timeset can be uniquely determined.

LEMMA 1. *For any swarm (O, T), $O \neq \emptyset$, there is a unique time-closed swarm (O, T') s.t. $T \subseteq T'$.*

Its proof can be found in Appendix B. In the running example, if we set the objectset as $\{o_1, o_2\}$, its maximal corresponding timeset is $\{t_1, t_2, t_4\}$. Thus, we only need to search all subsets of O_{DB}. An alternative search direction based on timeset is discussed in Appendix E.1. In this way, the search space shrinks from $(2^{|O_{DB}|} \times 2^{|T_{DB}|})$ to $2^{|O_{DB}|}$.

Basic idea of our algorithm. From the analysis above we see that, to find closed swarms, it suffices to only search all the subsets O of moving objects O_{DB}. For the search space of O_{DB}, we perform depth-first search of all subsets of O_{DB}, which is illustrated as pre-order tree traversal in Figure 5:

tree nodes are labeled with numbers, denoting the depth-first search order (nodes without numbers are pruned).

Even though, the search space is still huge for enumerating the objectsets in O_{DB} ($2^{|O_{DB}|}$). So efficient pruning rules are demanding to speed up the search process. We design two efficient pruning rules to further shrink the search space. The first pruning rule, called *Apriori Pruning*, is to stop traversing the subtree when we find further traversal cannot satisfy min_t. The second pruning rule, called *Backward Pruning*, is to make use of the closure property. It checks whether there is a superset of the current objectset, which has the same maximal corresponding timeset as that of the current one. If so, the traversal of the subtree under the current objectset is meaningless. In previous studies [19, 25, 21] on closed frequent pattern mining, there are three pruning rules (*i.e.*, item-merging, sub-itemset pruning, and item skipping) to cover different redundant search cases (the details of these techniques are stated in Appendix C.3). We simply use one pruning rule to cover all these cases and we will prove that we only need to examine each superset with *one more* object of the current objectset. Armed with these two pruning rules, the size of the search space can be significantly reduced.

After pruning the invalid candidates, the remaining ones may or may not be closed swarms. A brute-force solution is to check every pair of the candidates to see if one makes the other violate the closed swarm definition. But the time spent on this post-processing step is the square of the number of candidates, which is costly. Our proposal, Forward Closure Checking, is to embed a checking step in the search process. This checking step *immediately* determines whether a swarm is closed after the subtree under the swarm is traversed, and takes little extra time (actually, $O(1)$ additional time for each swarm in the search space). Thus, closed swarms are discovered on-the-fly and no extra post-processing step is needed.

In the following subsections, we present the details of our ObjectGrowth algorithm. The proofs of lemmas and theorems are given in Appendix B.

4.1 The ObjectGrowth Method

The ObjectGrowth method is a depth-first-search (DFS) framework based on the *objectset search space* (*i.e.*, the collection of all subsets of O_{DB}). First, we introduce the definition of maximal timeset. Intuitively, for an objectset O, the *maximal timeset* $T_{max}(O)$ is the one such that $(O, T_{max}(O))$ is a time-closed swarm. For an objectset O, the maximal timeset $T_{max}(O)$ is well-defined, because Lemma 1 shows the uniqueness of $T_{max}(O)$.

DEFINITION 4.1. *(Maximal Timeset) Timeset $T = \{t_j\}$ is a* maximal timeset *of objectset $O = \{o_{i_1}, o_{i_2}, \ldots, o_{i_m}\}$ if:*
(1) $C_{t_j}(o_{i_1}) \cap C_{t_j}(o_{i_2}) \cap \cdots \cap C_{t_j}(o_{i_m}) \neq \emptyset$, $\forall t_j \in T$;
(2) $\nexists t_x \in T_{DB} \setminus T$, s.t. $C_{t_x}(o_{i_1}) \cap \cdots \cap C_{t_x}(o_{i_m}) \neq \emptyset$. We use $T_{max}(O)$ to denote the maximal timeset of objectset O.

In the running example, for $O = \{o_1, o_2\}$, $T_{max}(O) = \{t_1, t_2, t_4\}$ is the maximal timeset of O.

The objectset space is visited in a DFS order. When visiting each objectset O, we compute its maximal timeset. And three rules are further used to prune redundant search and detect the closed swarms on-the-fly.

4.1.1 Apriori Pruning Rule

The following lemma is from the definition of T_{max}.

LEMMA 2. *If $O \subseteq O'$, then $T_{max}(O') \subseteq T_{max}(O)$.*

This lemma is intuitive. When objectset grows bigger, the maximal timeset will shrink or at most keep the same. This further gives the following pruning rule.

RULE 1. ***(Apriori Pruning)** For an objectset O, if $|T_{max}(O)| < min_t$, then there is no strict superset O' of O ($O' \neq O$) s.t. $(O', T_{max}(O'))$ is a (closed) swarm.*

In Figure 5, the nodes with objectset $O = \{o_1, o_3\}$ and its subtree are pruned by Apriori Pruning , because $T_{max}(O) < min_t$, and all objectsets in the subtree are strict supersets of O. Similarly, for the objectsets $\{o_2, o_3\}$, $\{o_3, o_4\}$ and $\{o_1, o_2, o_3\}$, the nodes with these objectsets and their subtrees are also pruned by Apriori Pruning .

4.1.2 Backward Pruning Rule

By using Apriori Pruning, we prune objectsets O with $T_{max}(O) < min_t$. However, the pruned search space could still be extremely huge as shown in the following example.

Suppose there are 100 objects which are all in the same cluster for the whole time span. Given $min_o = 1$ and $min_t = 1$, we can hardly prune any node using Apriori Pruning . The number of objectsets we need to visit is 2^{100}! But it is easy to see that there is only one closed swarm: (O_{DB}, T_{DB}). We can get this closed swarm when we visit the objectset $O = O_{DB}$ in the DFS after 100 iterations. After that, we waste a lot of time searching objectsets which can never produce any closed swarms.

Since our goal is to mine only closed swarms, we can develop another stronger pruning rule to prune the subtrees which cannot produce closed swarms. Let us take some observations in the running example first.

In Figure 5, for the node with objectset $O = \{o_1, o_4\}$, we can insert o_2 into O and form a superset $O' = \{o_1, o_2, o_4\}$. O' has been visited and expanded before visiting O. And we can see that $T_{max}(O) = T_{max}(O') = \{t_1, t_2, t_4\}$. This indicates that for any timestamp when o_1 and o_4 are together, o_2 will also be in the same cluster as them. So for any superset of $\{o_1, o_4\}$ without o_2, it can never form a closed swarm. Meanwhile, o_2 will not be in O's subtree in the depth-first search order. Thus, the node with $\{o_1, o_4\}$ and its subtree can be pruned.

To formalize Backward Pruning rule, we first state the following lemma.

LEMMA 3. *Consider an objectset $O = \{o_{i_1}, o_{i_2}, \ldots, o_{i_m}\}$ ($i_1 < i_2 < \ldots < i_m$), if there exists an objectset O' such that O' is generated by adding an additional object $o_{i'}$ ($o_{i'} \notin O$ and $i' < i_m$) into O such that $C_{t_j}(O) \subseteq C_{t_j}(o_{i'})$, $\forall t_j \in T_{max}(O)$, then for any objectset O'' satisfying $O \subseteq O''$ but $O' \nsubseteq O''$, $(O'', T_{max}(O''))$ is not a closed swarm.*

Note that when overlapping is not allowed in the clusters, the condition $C_{t_j}(O) \subseteq C_{t_j}(o_{i'})$, $\forall t_j \in T_{max}(O)$ simply reduces to $T_{max}(O') = T_{max}(O)$. Armed with the above lemma, we have the following pruning rule.

RULE 2. ***(Backward Pruning)** Consider an objectset $O = \{o_{i_1}, o_{i_2}, \ldots, o_{i_m}\}$ ($i_1 < i_2 < \ldots < i_m$), if there exists an*

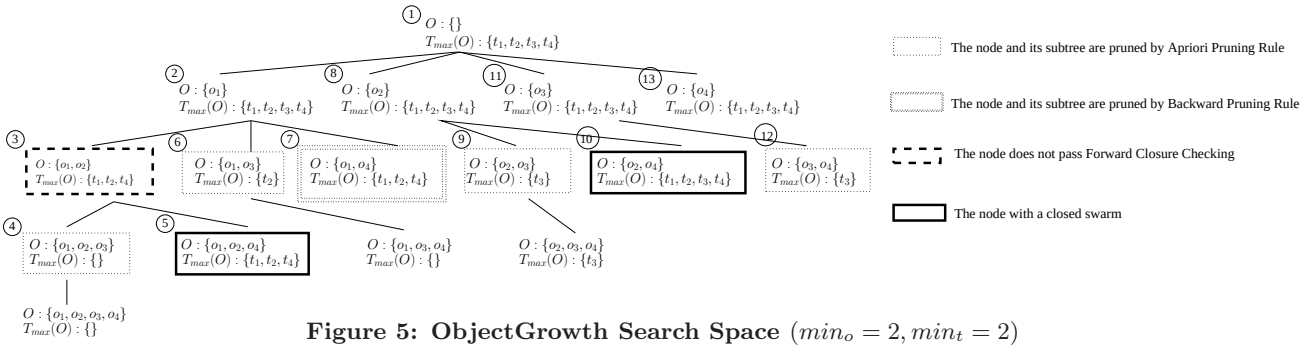


Figure 5: ObjectGrowth Search Space ($min_o = 2, min_t = 2$)

objectset O' such that O' is generated by adding an additional object $o_{i'}$ ($o_{i'} \notin O$ and $i' < i_m$) into O such that $C_{t_j}(O) \subseteq C_{t_j}(o_{i'})$, $\forall t_j \in T_{max}(O)$, then O can be pruned in the objectset search space (stop growing from O in the depth-first search).

Backward Pruning is efficient in the sense that it only needs to examine those supersets of O with one more object rather than all the supersets. This rule can prune a significant portion of the search space for mining closed swarms. Experimental results (see Figure 8) show that the speedup (compared with the algorithms for mining all swarms without this rule) is an exponential factor w.r.t. the dataset size.

4.1.3 Forward Closure Checking

To check whether a swarm $(O, T_{max}(O))$ is closed, from the definition of closed swarm, we need to check every superset O' of O and $T_{max}(O')$. But, actually, according to the following lemma, checking the superset O' of O with one more object suffices.

LEMMA 4. *Swarm $(O, T_{max}(O))$ is closed iff for any superset O' of O with exactly one more object, we have $|T_{max}(O')| < |T_{max}(O)|$.*

In Figure 5, the node with objectset $O = \{o_1, o_2\}$ is not pruned by any pruning rules. But it has a child node with objectset $\{o_1, o_2, o_4\}$ having same maximal timeset as $T_{max}(O)$. Thus ($\{o_1, o_2\}$, $\{t_1, t_2, t_4\}$) is not a closed swarm because of Lemma 4.

Consider a superset O' of objectset $O = \{o_{i_1}, \ldots, o_{i_m}\}$ s.t. $O' \setminus O = \{o_{i'}\}$. Rule 2 checks the case that $i' < i_m$. The following rule checks the case that $i' > i_m$.

RULE 3. ***(Forward Closure Checking)*** *Consider an objectset $O = \{o_{i_1}, o_{i_2}, \ldots, o_{i_m}\}$ ($i_1 < i_2 < \ldots < i_m$), if there exists an objectset O' such that O' is generated by adding an additional object $o_{i'}$ ($o_{i'} \notin O$ and $i' > i_m$) into O, and $|T_{max}(O')| = |T_{max}(O)|$, then (O, T) is not a closed swarm.*

Note, unlike Rule 2, Rule 3 does not prune the objectset O in the DFS. In other words, we cannot stop DFS from O. But this rule is useful for detecting non-closed swarms.

4.1.4 The ObjectGrowth Algorithm

Figure 5 shows the complete ObjectGrowth algorithm for our running example. We traverse the search space in the DFS order. When visiting the node with $O = \{o_1, o_2, o_3\}$, it fails to pass the Apriori Pruning condition. So we stop growing from it, trace back and visit node $O = \{o_1, o_2, o_4\}$. O passes both pruning as well as Forward Closure Checking. By Theorem 1 that will be introduced immediately afterwards, O and its maximal timeset $T = \{t_1, t_2, t_4\}$ form a closed swarm. So we can output (O, T). When we trace back to node $\{o_1, o_2\}$, because its child contains a closed swarm with the same timeset as $\{o_1, o_2\}$'s maximal timeset, $\{o_1, o_2\}$ will not be a closed swarm by the Forward Closure Checking. We continue visiting the nodes until we finish the traversal of the objectset-based DFS tree.

THEOREM 1. ***(Identification of closed swarm in ObjectGrowth)*** *For a node with objectset O, $(O, T_{max}(O))$ is a closed swarm* ***if and only if*** *it passes the Apriori Pruning , Backward Pruning, Forward Closure Checking, and $|O| \geq min_o$.*

Theorem 1 makes the discovery of closed swarms well embedded in the search process so that closed swarms can be reported on-the-fly.

Algorithm 1 presents the pseudo code of ObjectGrowth. To find all closed swarms, we start with *ObjectGrowth*({}, T_{DB}, 0, min_o, min_t, $|O_{DB}|$, $|T_{DB}|$, C_{DB}).

When visiting the node with objectset O, we first check whether it can pass the Apriori Pruning (lines 2-3). Checking the size of T_{max} only takes $O(1)$.

Next, we check whether the current node can pass the Backward Pruning. In the subroutine *BackwardPruning* (lines 15-18), 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. This subroutine takes $O(|O_{DB}| \times |T_{DB}|)$ in the worst case.

After both pruning, we will visit all the child nodes in the DFS order (lines 6-12). (i) For a child node with objectset O', we generate its maximal timeset in the subroutine *GenerateMaxTimeset* (lines 19-22). When generating $T_{max}(O')$, we do not need to check every timestamp, instead we only need to check the timestamps in $T_{max}(O)$ because we know that $T_{max}(O') \subseteq T_{max}(O)$ by Lemma 2. So in each iteration, this subroutine takes $O(|T_{DB}|)$ time in the worst case. (ii) For Forward Closure Checking we use a variable $forward\_closure$ to record whether any child node with objectset O' has the same maximal timeset size as $T_{max}(O)$. This takes $O(1)$ times in each iteration. In sum, as a node will have at most $|O_{DB}|$ direct child nodes, this loop (lines 7-12) will repeat at most $|O_{DB}|$ times, and take $O(|O_{DB}| \times |T_{DB}|)$ time in the worst case.

Finally, after visiting the subtree under current node, if the node passes Forward Closure Checking and $|O| \geq min_o$, we can immediately output $(O, T_{max}(O))$ as a closed swarm (lines 13-14).

Algorithm 1 ObjectGrowth(O, T_{max}, o_{last}, min_o, min_t, $|O_{DB}|$, $|T_{DB}|$, C_{DB})

Input: O: current objectset; T_{max}: maximal timeset of O; o_{last}: latest object added into O; min_o and min_t: minimum threshold parameters; $|O_{DB}|$: number of objects in database; $|T_{DB}|$: number of timestamps in database; C_{DB}: clustering snapshots at each timestamp.
Output: (O, T_{max}) if it is a closed swarm.
Algorithm:

```
1: {Apriori Pruning}
2: if |T_max| < min_t then
3:    return;
4: {Backward Pruning}
5: if BackwardPruning(o_last, O, T_max, C_DB) then
6:    forward_closure ← true;
7:    for o ← o_last + 1 to |O_DB| do
8:       O' ← O ∪ {o};
9:       T'_max ← GenerateMaxTimeset(o, o_last, T_max,
         C_DB);
10:      if |T_max| = |T'_max| then
11:         forward_closure ← false; {Forward Closure
            Checking}
12:      ObjectGrowth(O_new, T_new, o, min_o, min_t, |O_DB|,
         |T_DB|, C_DB);
13:   if forward_closure and |O| ≥ min_o then
14:      output pair (O, T) as a closed swarm;

   Subroutine: BackwardPruning(o_last, O, T_max,
   C_DB)
15: for ∀o ∉ O and o < o_last do
16:    if C_t(o) ⊆ C_t(O), ∀t ∈ T_max then
17:       return false;
18: return true;

   Subroutine: GenerateMaxTimeset(o, o_last, T_max,
   C_DB)
19: for ∀t ∈ T_max do
20:    if C_t(o) ∩ C_t(o_last) ≠ ∅ then
21:       T'_max ← T'_max ∪ t;
22: return T'_max;
```

Therefore, it takes $O(|O_{DB}| \times |T_{DB}|)$ for each iteration in the depth-first-search. The memory usage for ObjectGrowth is $O(|T_{DB}| \times |O_{DB}|)$ in worst case.

5. EXPERIMENT

A comprehensive performance study has been conducted on both real and synthetic datasets. All the algorithms were implemented in C++, and all the experiments are carried out on a 2.8 GHz Intel Core 2 Duo system with 4GB memory. The system ran MAC OS X with version 10.5.5 and gcc 4.0.1.

The implementation of swarm mining is also integrated in our demonstration system [18]. The demo system is public online[1]. It is tested on a set of real animal data sets from MoveBank.org[2]. The data and results are visualized in Google Map[3] and Google Earth[4].

[1]http://dm.cs.uiuc.edu/movemine/
[2]http://www.movebank.org
[3]http://maps.google.com
[4]http://earth.google.com

5.1 Effectiveness

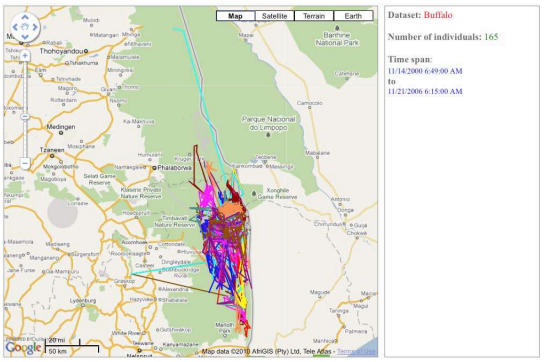


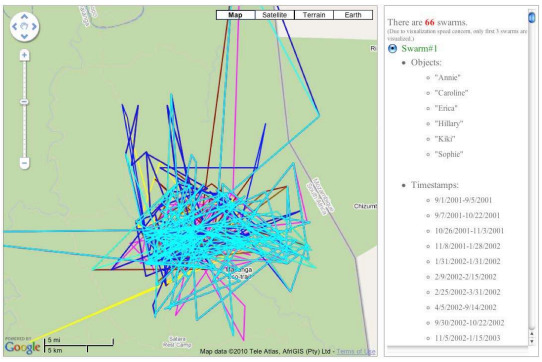
Figure 6: Raw buffalo data.

The effectiveness of swarm pattern can be demonstrated through our online demo system. Here, we use one dataset as an example to show the effectiveness. This data set contains 165 buffalo with tracking time from Year 2000 to Year 2006. The original data has 26610 reported locations. Figure 6 shows the raw data plotted in Google Map.

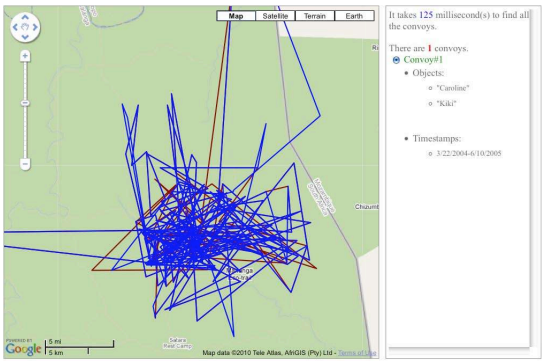
For each buffalo, the locations are reported about every 3 or 4 days. We first use linear interpolation to fill in the missing data with time gap as one "day". Note that the first/last tracking days for each buffalo could be different. The buffalo movement with longest tracking time contains 2023 days and the one with shortest tracking time contains only 1 day. On average, each buffalo contains 901 days. We do not interpolate the data to enforce the same first/last tracking day. Instead, we require the objects that form a swarm should be together for at least min_t *relative* timestamps over their *overlapping* tracking timestamps. For example, by setting $min_t = 0.5$, o_1 and o_2 form a swarm if they are close for at least half of their overlapping tracking timestamps. Then, DBSCAN [7] with parameter $MinPts = 5$ and $Eps = 0.001$ is applied to generate clusters at each timestamp (*i.e.*, C_{DB}). Note that, regarding to users' specific requirements, different clustering methods and parameter settings can be applied to pre-process the raw data.

By setting $min_o = 2$ and $min_t = 0.5$ (*i.e.*, half of the overlapping time span), we can find 66 closed swarms. Figure 7(a) shows one swarm. Each color represents the raw trajectory of a buffalo. This swarm contains 5 buffalo. And the timestamps that these buffalo are in the same cluster are non-consecutive. Looking at the raw trajectory data in Figure 6, people can hardly detect interesting patterns manually. The discovery of the swarms provides useful information for biologists to further examine the relationship and habits of these buffalo.

For comparison, we test convoy pattern mining on the same data set. Note that there are two parameters in convoy definition, m (number of objects) and k (threshold of consecutive timestamps). So m actually equals to min_o and k is the same as min_t. (For the details of convoy definition and algorithm, please refer to Section C.1.) We first use the same parameters (*i.e.*, $min_o = 2$ and $min_t = 0.5$) to mine convoys. However, no convoy is discovered. This is because there is no group of buffalo that move together for consecutively half of the whole time span. By lowering the parameter min_t from 0.5 to 0.2, there is one convoy



(a) One of the seven swarms discovered with $min_o = 2$ and $min_t = 0.5$



(b) One convoy discovered with $min_o = m = 2$ and $min_t = k = 0.2$

Figure 7: Effectiveness comparison between swarm and convoy.

discovered as shown in Figure 7(b). But this convoy, containing 2 buffalo, is just a subset of one swarm pattern. The rigid definition of convoy makes it not practical to find potentially interesting patterns. The comparison shows that the concept of (closed) swarms are especially meaningful in revealing relaxed temporal moving object clusters.

5.2 Efficiency

To show the efficiency of our algorithms, we generate larger synthetic dataset using Brinkhoff's network-based generator of moving objects[5]. We generate 500 objects ($|O_{DB}| = 500$) for 10^5 timestamps ($|T_{DB}| = 10^5$) using the generator's default map and parameter setting. There are $5 \cdot 10^7$ points in total. DBSCAN ($MinPts$= 3, Eps = 300) is applied to get clusters at each snapshot.

In the efficiency comparison, we include a new algorithm ObjectGrowth+, which is an extension of ObjectGrowth to handle probablistic data. Due to space limit, the details of ObjectGrowth+ are presented in Appendix A. Here, we briefly describe the idea of ObjectGrowth+. ObjectGrowth+ is designed to handle an important issue in raw data—asynchronous data collection. Specifically, each object usually reports their locations at asynchronous timestamps. However, since we assume there is a location for each object at every timestamp, some interpolation method should be used to fill in the missing data first. But such interpolation is just an estimation on the real locations. So each point is associated with a probability showing the confidence of its estimation. While ObjectGrowth assumes every point is certain, ObjectGrowth+ is a more general version of ObjectGrowth that can handle the probabilistic data.

We will compare our algorithms with VG-Growth [22], which is the only previous work addressing the non-consecutive timestamps issue. To make fair comparison, we adapt VG-Growth to take the same input as ours but its time complexity will remain the same. This transformation will be descried in Section C.2. We further create a probabilistic database by randomly samping 1% points and assigning a random probability to these points. ObjectGrowth+ takes this additional probabilistic database as input. The algorithms are compared with respect to two parameters (*i.e.*, min_o and min_t) and the database size (*i.e.*, O_{DB} and T_{DB}). By default, $|O_{DB}| = 500$, $|T_{DB}| = 10^5$, $\frac{min_o}{|O_{DB}|} = 0.01$, $\frac{min_t}{|T_{DB}|} = 0.01$, and $\theta = 0.9$ (for ObjectGrowth+). We carry out four experiments by varying one variable with the other three fixed. Note that in the following experiment part, we use min_o to denote the ratio of min_o over O_{DB} and min_t to denote the ratio of min_t over T_{DB}.

Efficiency w.r.t. min_o and min_t. Figure 8(a) shows the running time w.r.t. min_o. It is obvious that VG-Growth takes much longer time than ObjectGrowth. VG-Growth cannot even produce results within 5 hours when $min_o = 0.018$ in Figure 8(a). The reason is that VG-Growth tries to find all the swarms rather than *closed* swarms, and the number of swarms is exponentially larger than that of closed swarms as shown in Figure 9(a) and Figure 9(b). Besides, we can see that ObjectGrowth+ is slower than ObjectGrowth because Backward pruning rule is weaker in ObjectGrowth+ due to the strong constraint posed by probabilistic database.

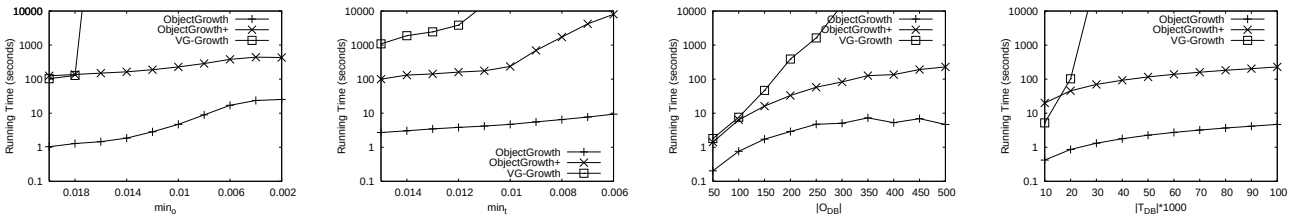
Efficiency w.r.t. $|O_{DB}|$ and $|T_{DB}|$. Figure 8(c) and Figure 8(d) depict the running time when varying $|O_{DB}|$ and $|T_{DB}|$ respectively. In both figures, VG-Growth is much slower than ObjectGrowth and ObjectGrowth+. Furthmore, ObjectGrowth+ is usually 10 times slower than ObjectGrowth. Comparing Figure 8(c) and Figure 8(d), we can see that ObjectGrowth is more sensitive to the change of O_{DB}. This is because its search space is enlarged with larger O_{DB} whereas the change of T_{DB} does not directly affect the running time of ObjectGrowth.

In summary, ObjectGrowth and ObjectGrowth+ greatly outperforms VG-Growth since the number of swarms is exponential to the number of closed swarms. ObjectGrowth+ is slower than ObjectGrowth because it considers probabilistic data and thus its pruning rule is weaker. Besides, both ObjectGrowth and ObjectGrowth+ are more sensitive to the size of O_{DB} rather than that of T_{DB} since the search space is based on the objectset.

6. CONCLUSIONS

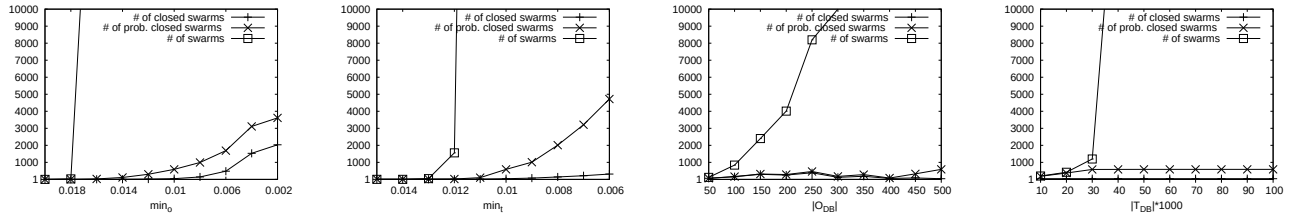
We propose the concepts of swarm and closed swarm. These concepts are different from that in the previous work and they enable the discovery of interesting moving object clusters with relaxed temporal constraint. A new method, ObjectGrowth, with two strong pruning rules and one closure checking, is proposed to efficiently discover closed swarms. The effectiveness is demonstrated using real data and efficiency is tested on large synthetic data.

[5]http://www.fh-oow.de/institute/iapg/personen/brinkhoff/generator/



(a) Running time w.r.t. min_o (b) Running time w.r.t. min_t (c) Running time w.r.t. $|O_{DB}|$ (d) Running time w.r.t. $|T_{DB}|$

Figure 8: Running Time on Synthetic Dataset



(a) Number of (closed) swarms w.r.t. min_o (b) Number of (closed) swarms w.r.t. min_t (c) Number of (closed) swarms w.r.t. $|O_{DB}|$ (d) Number of (closed) swarms w.r.t. $|T_{DB}|$

Figure 9: Number of (closed) swarms in Synthetic Dataset

7. REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *VLDB*, 1994.

[2] G. Al-Naymat, S. Chawla, and J. Gudmundsson. Dimensionality reduction for long duration and complex spatio-temporal queries. In *SAC*, 2007.

[3] I. Assent, R. Krieger, E. Müller, and T. Seidl. Inscy: Indexing subspace clusters with in-process-removal of redundancy. In *ICDM*, 2008.

[4] M. Benkert, J. Gudmundsson, F. Hubner, and T. Wolle. Reporting flock patterns. In *COMGEO*, 2008.

[5] L. Chen and R. T. Ng. On the marriage of lp-norms and edit distance. In *VLDB*, 2004.

[6] L. Chen, M. T. Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *SIGMOD*, 2005.

[7] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases. In *KDD*, 1996.

[8] S. Gaffney, A. Robertson, P. Smyth, S. Camargo, and M. Ghil. Probabilistic clustering of extratropical cyclones using regression mixture models. In *Technical Report UCI-ICS 06-02*, 2006.

[9] J. Gudmundsson and M. van Kreveld. Computing longest duration flocks in trajectory data. In *GIS*, 2006.

[10] J. Gudmundsson, M. J. van Kreveld, and B. Speckmann. Efficient detection of motion patterns in spatio-temporal data sets. In *GIS*, 2004.

[11] J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *DMKD*, 2004.

[12] H. Jeung, H. T. Shen, and X. Zhou. Convoy queries in spatio-temporal databases. In *ICDE*, 2008.

[13] H. Jeung, M. L. Yiu, X. Zhou, C. S. Jensen, and H. T. Shen. Discovery of convoys in trajectory databases. In *PVLDB*, 2008.

[14] P. Kalnis, N. Mamoulis, and S. Bakiras. On discovering moving clusters in spatio-temporal data. In *SSTD*, 2005.

[15] P. Kröger, H.-P. Kriegel, and K. Kailing. Density-connected subspace clustering for high-dimensional data. In *SDM*, 2004.

[16] P. Laube and S. Imfeld. Analyzing relative motion within groups of trackable moving point objects. In *GIS*, 2002.

[17] J.-G. Lee, J. Han, and K.-Y. Whang. Trajectory clustering: A partition-and-group framework. In *SIGMOD*, 2007.

[18] Z. Li, M. Ji, J.-G. Lee, L. Tang, Y. Yu, J. Han, and R. Kays. Movemine: Mining moving object databases, to appear. In *SIGMOD*, 2010.

[19] J. Pei, J. Han, and R. Mao. Closet: An efficient algorithm for mining frequent closed itemsets. In *DMKD*, 2000.

[20] M. Vlachos, D. Gunopulos, and G. Kollios. Discovering similar multidimensional trajectories. In *ICDE*, 2002.

[21] J. Wang, J. Han, and J. Pei. Closet+: searching for the best strategies for mining frequent closed itemsets. In *KDD*, 2003.

[22] Y. Wang, E.-P. Lim, and S.-Y. Hwang. Efficient mining of group patterns from user movement data. In *DKE*, 2006.

[23] X. Yan, J. Han, and R. Afshar. CloSpan: Mining closed sequential patterns in large datasets. In *SDM*, 2003.

[24] B.-K. Yi, H. V. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. In *ICDE*, 1998.

[25] M. J. Zaki and C. J. Hsiao. Charm: An efficient algorithm for closed itemset mining. In *SDM*, 2002.

APPENDIX

A. HANDLING ASYNCHRONOUS DATA

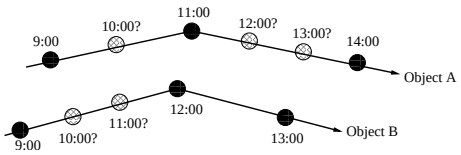


Figure 10: Asynchronous raw data

When mining swarms, we assume that each moving object has a reported location at each timestamp. However, in most real cases, the raw data collected is not as ideal as we expected. First, the data for a moving object could be *sparse*. When tracking animals, it is quite possible that we only get one reported point every several hours or even every several days. When tracking vehicles or people, there could be a period of missing data when people turn off the tracking devices (*e.g.*, GPS or cell phones). Second, the sampling timestamps for different moving objects are usually *not synchronized*. As shown in Figure 10, the recorded times of objects A and B are different. The recorded time points of A are 9:00, 11:00, and 14:00; and that of B are 9:00, 12:00, and 13:00. The locations at other timestamps are an estimation of the real locations.

The raw data is usually preprocessed using linear interpolation. But the moving objects may not necessarily follow the linear model. Such interpolation could be wrong. Even though more complicated interpolation methods could be used to fill in the missing data with higher precision, any interpolation is only a guessing of real positions. For some missing points, we could have higher confidence in guessing its location whereas for the others, the confidence could be lower. So each interpolated point is associated with a probability showing the confidence of guessing.

To find real swarms, it is important to differentiate between reported locations and interpolated locations. For example, if a swarm (O, T) with most objects in O having low confidence in interpolated positions at times in T, those objects in O may not actually move together because the interpolated points have high probability to be wrong. On the other hand, if we can first find those swarms with high confidence, we can use these swarms to better adjust the missing points and further iteratively refine the swarms. In this section, we focus on how to generalize our ObjectGrowth method to handle probabilistic data. In Appendix D.2, we discuss how to obtain the probability and leave the iterative refinement framework as an interesting future work.

A.1 Closed Swarms with Probability

We first define closed swarms in the context of probabilistic data. A probabilistic database is derived from original trajectory data. $P_{t_i}(o_j)$ is used to denote the probability of o_j at time point t_i. If there is a recorded location of o_j at t_i, $P_{t_i}(o_j) = 1$. If not, some interpolation method is used to guess this location and the probability is calculated based on the confidence of interpolation of o_j at t_i.

Given a probabilistic database $\{P_{t_i}(o_j)\}$, we define the *confidence* of a pair (O, T) as:

$$f(O,T) = \sum_{t_i \in T} \prod_{o_j \in O} P_{t_i}(o_j).$$

Recall that in our definition, a swarm is a pair (O, T) which satisfies three additional constraints as listed in Section 3. To accommodate the probabilistic issue, we propose to simply replace the second constraint, $|T| \geq min_t$, with the following *generalized minimum timestamp constraint*:

(2') $f(O, T) \geq \theta \times min_t$: (O, T) needs to satisfy the confidence threshold θ.

It is easy to see that if $P_{t_i}(o_j) = 1$, $\forall t_i \in T$ and $\forall o_j \in O$, condition (2') becomes $f(O, T) = |T| \geq min_t$ when $\theta = 1$. Meanwhile, the more uncertainty in the original data, the more difficult this requirement can be satisfied. In other words, the objects having more reported locations at timestamps in T are preferred. Note that the definition of *closed swarms* remains the same.

A.2 The ObjectGrowth+ method

The ObjectGrowth+ method is derived from the ObjectGrowth method to accommodate the probabilistic data. The general philosophy of ObjectGrowth+ is the same to that of ObjectGrowth. We search on the objectset space in the DFS fashion using the Apriori and Backward rules to prune the redundant search space and use Closure Checking to report the closed swarms on-the-fly. Such rules are modified accordingly. In this section, we will formulate the lemmas and informally describe the rules. The proofs are deferred in Appendix B.

LEMMA 5. *If $O \subseteq O'$, then $f(O', T_{max}(O')) \leq f(O, T_{max}(O))$.*

Apriori pruning rule can be naturally derived from Lemma 5. That is, when visiting node $(O, T_{max}(O))$, if $f(O, T_{max}(O)) < \theta \times min_t$, we can stop searching deeper from this node. Because all the objectsets in its children nodes are supersets of O and thus all the children nodes will also violate this requirement for closed swarms.

LEMMA 6. *Consider an objectset $O = \{o_{i_1}, o_{i_2}, \ldots, o_{i_m}\}$ $(i_1 < i_2 < \ldots < i_m)$, if there exists an object O' generated by adding an additional object $o_{i'}$ ($o_{i'} \notin O$ and $i' < i_m$) into O such that $C_{t_j}(O) \subseteq C_{t_j}(o_{i'})$ and $P_{t_j}(o_{i'}) = 1$, $\forall t_j \in T_{max}(O)$, then for any objectset O'' satisfying $O \subseteq O''$ but $O' \nsubseteq O''$, $(O'', T_{max}(O''))$ is not a closed swarm.*

Comparing Lemma 6 with Lemma 3, we can see that there is an additional constraint in Lemma 6. Besides checking whether $C_{t_j}(O) \subseteq C_{t_j}(o_{i'})$, we need to further check whether $P_{t_j}(o_{i'}) = 1, \forall t_j \in T_{max}(O)$. As the constraints are harder to be satisfied, the Backward pruning rule becomes weaker in ObjectGrowth+. For example, let $O = \{o_2\}$ and $O' = \{o_1, o_2\}$. We cannot prune node with $(O, T_{max}(O))$ because there could exist a child node of O, $O'' = \{o_2, o_3\}$, such that $f(O'', T_{max}(O'')) \geq \theta \times min_t$ whereas the child node of O', $O''' = \{o_1, o_2, o_3\}$, does not satisfy $f(O''', T_{max}(O''')) \geq \theta \times min_t$. This is the major reason why ObjectGrowth+ takes longer time to discover swarms than ObjectGrowth as shown in experiments in Section 5.

LEMMA 7. *Swarm $(O, T_{max}(O))$ is closed iff for any superset O' of O with exactly one more object, we have $|T_{max}(O')| < |T_{max}(O)|$ or $f(O', T_{max}(O')) < \theta \times min_t$.*

When visiting node with O, different from the forward closure checking rule in ObjectGrowth, we need to check both "forward" and "backward" supersets of O. If $O = \{o_{i_1},$

$o_{i_2}, \ldots, o_{i_m}\}$, we need to test each superset O' by adding $o_{i'} \notin O$. Once $|T_{max}(O')| < |T_{max}(O)|$ or $f(O', T_{max}(O')) < \theta \times min_t$, current node with O will pass the closure checking.

Finally, we output the set of nodes $(O, T_{max}(O))$ passing all the conditions and $|O| \geq min_o$.

B. PROOFS OF LEMMAS AND THEOREMS

B.1 Proof of Lemma 1

The existence is trivial, since we can always let $T' = T$. We only need to prove the uniqueness. For the purpose of contradiction, suppose there are two time-closed swarms (O, T_1) and (O, T_2), $T_1 \neq T_2$ s.t. $T \subseteq T_1$ and $T \subseteq T_2$. Letting $T'' = T_1 \cup T_2$, from the definition, (O, T'') is also a swarm. Because $T_1 \neq T_2$, we have $T_1 \subsetneq T''$ and $T_2 \subsetneq T''$. However, this contradicts with the fact that (O, T_1) and (O, T_2) are time-closed. So there is a unique time-closed swarm (O, T') s.t. $T \subseteq T'$. □

B.2 Proof of Lemma 2

$\forall t \in T_{max}(O')$, by the definition of maximal timeset, we have $C_t(o'_i) \cap C_t(o'_j) \neq \emptyset$, $\forall o'_i, o'_j \in O'$. This implies $C_t(o_i) \cap C_t(o_j) \neq \emptyset$, $\forall o_i, o_j \in O$ since $O \subseteq O'$. Therefore, $t \in T_{max}(O)$. □

B.3 Proof of Lemma 3

We show that $(O'' \cup \{o_{i'}\}, T_{max}(O''))$ is a swarm, hence $(O'', T_{max}(O''))$ cannot be a closed swarm. By construction, it satisfies conditions (1) and (2) in the definition of a swarm trivially, so we only need to check condition (3). To see that (3) holds for $(O'' \cup \{o_{i'}\}, T_{max}(O''))$ as well, note that $C_{t_j}(O) \subseteq C_{t_j}(o_{i'})$, $\forall t_j \in T_{max}(O)$. Since $T_{max}(O'') \subseteq T_{max}(O)$, we have

$$C_{t_j}(O'') \subseteq C_{t_j}(O) \subseteq C_{t_j}(o_{i'}), \forall t_j \in T_{max}(O''),$$

hence,

$$C_{t_j}(O'') \cap C_{t_j}(o_{i'}) = C_{t_j}(O'') \neq \emptyset, \forall t_j \in T_{max}(O'').$$

Therefore, $(O'' \cup \{o_{i'}\}, T_{max}(O''))$ is a swarm and we are done. □

B.4 Proof of Lemma 4

The proof of $(\Rightarrow)$ is trivial by the definition of closed swarm. For $(\Leftarrow)$, first note that by using $T_{max}(O)$, it automatically satisfies the time-closed condition. Second, $\forall O''$, *s.t.* $O \subsetneq O''$, choose $o'' \in O'' \setminus O$. Consider the set $O^+ = O \cup \{o''\}$. Since $T_{max}(O^+) \subseteq T_{max}(O)$ and $|T_{max}(O^+)| < |T_{max}(O)|$ by assumption, we get $T_{max}(O^+) \subsetneq T_{max}(O)$. By Lemma 2, $T_{max}(O'') \subsetneq T_{max}(O)$. Thus, $(O'', T_{max}(O))$ is not a swarm. Hence $(O, T_{max}(O))$ is object-closed and therefore closed. □

B.5 Proof of Theorem 1

Clearly, every closed swarm is derived by Apriori Pruning , Backward Pruning, Forward Closure Checking, and $|O| \geq min_o$. Now suppose that a node $(O, T_{max}(O))$ passes all the conditions. First, Apriori Pruning ensures that $|T_{max}(O)| \geq min_t$. Also, we explicitly require $|O| \geq min_o$, so $(O, T_{max}(O))$ satisfies the swarm requirement. Next, by definition, $(O, T_{max}(O))$ is guaranteed to be time-closed. Finally, if $(O, T_{max}(O))$ were not object-closed, it fails the conditions of Backward Pruning or Forward Closure Checking (Lemma 4). Therefore, $(O, T_{max}(O))$ is a closed swarm. □

B.6 Proof of Lemma 5

By Lemma 2, we have $T_{max}(O') \subseteq T_{max}(O)$. Therefore,

$$\begin{aligned} f(O', T_{max}(O')) &= \sum_{t_i \in T_{max}(O')} \prod_{o_j \in O'} P_{t_i}(o_j) \\ &\leq \sum_{t_i \in T_{max}(O)} \prod_{o_j \in O'} P_{t_i}(o_j) \\ &\leq \sum_{t_i \in T_{max}(O)} \prod_{o_j \in O} P_{t_i}(o_j) \\ &= f(O, T_{max}(O)), \end{aligned}$$

where we use the fact that for any t_i and o_j, $0 < P_{t_i}(o_j) \leq 1$. □

B.7 Proof of Lemma 6

Note that since $0 < P_{t_i}(o_j) \leq 1$ for any t_i and o_j, the conditions $C_{t_j}(O) \subseteq C_{t_j}(o_{i'})$ and $P_{t_j}(o_{i'}) = 1$ imply:

$$f(O', T_{max}(O')) = f(O, T_{max}(O)).$$

We demonstrate that $(O'' \cup \{o_{i'}\}, T_{max}(O''))$ is a swarm, hence $(O'', T_{max}(O''))$ cannot be a closed swarm. First note that $(O'' \cup \{o_{i'}\}, T_{max}(O''))$ satisfies condition (1) trivially. For (2'), observe that $P_{t_j}(o_{i'}) = 1$, $\forall t_j \in T_{max}(O'')$ because $T_{max}(O'') \subseteq T_{max}(O)$, therefore

$$\begin{aligned} & f(O'' \cup \{o_{i'}\}, T_{max}(O'')) \\ = & \sum_{t_i \in T_{max}(O'')} \prod_{o_j \in O'' \cup \{o_{i'}\}} P_{t_i}(o_j) \\ = & \sum_{t_i \in T_{max}(O'')} \prod_{o_j \in O''} P_{t_i}(o_j) \\ = & f(O'', T_{max}(O'')). \end{aligned}$$

Finally, since $T_{max}(O'') \subseteq T_{max}(O)$, we have

$$C_{t_j}(O'') \subseteq C_{t_j}(O) \subseteq C_{t_j}(o_{i'}), \forall t_j \in T_{max}(O''),$$

hence,

$$C_{t_j}(O'') \cap C_{t_j}(o_{i'}) = C_{t_j}(O'') \neq \emptyset, \forall t_j \in T_{max}(O'').$$

Therefore condition (3) also holds. □

B.8 Proof of Lemma 7

The proof of $(\Rightarrow)$ is trivial by definition. For $(\Leftarrow)$, first note that by using $T_{max}(O)$, it automatically satisfies the time-closed condition. Second, $\forall O''$, *s.t.* $O \subsetneq O''$, choose $o'' \in O'' \setminus O$. Consider the set $O^+ = O \cup \{o''\}$. One one hand, if $|T_{max}(O^+)| < |T_{max}(O)|$ we get $T_{max}(O^+) \subsetneq T_{max}(O)$. Therefore,

$$T_{max}(O'') \subsetneq T_{max}(O^+) \subsetneq T_{max}(O)$$

by Lemma 2. One the other hand, if $T_{max}(O'') = T_{max}(O)$ and $f(O', T_{max}(O')) < \theta \times min_t$, then we get

$$\begin{aligned} f(O'', T_{max}(O)) &= f(O'', T_{max}(O'')) \\ &\leq f(O', T_{max}(O')) \\ &< \theta \times min_t \end{aligned}$$

by Lemma 5. So we conclude that $(O'', T_{max}(O))$ is not a swarm and $(O, T_{max}(O))$ is object-closed. □

C. RELATED WORKS FOR COMPARISON

C.1 Moving cluster, flock and convoy

Kalnis *et al.* propose the notion of *moving cluster* [14], which is a sequence of spatial clusters appearing during consecutive timestamps, such that the portion of common objects in any two consecutive clusters is not below a given threshold parameter θ, *i.e.*, $\frac{c_t \cap c_{t+1}}{c_t \cup c_{t+1}} \geq \theta$, where c_t denotes a cluster at time t. A *flock* [10, 9, 2, 4] is a group of at least m objects that move together within a circular region of radius r during a specific time interval of at least k timestamps. While flock could be sensitive to user-specified disc size and the circular shape might not always be appropriate, Jeung *et al.* further propose the concept of *convoy* [13, 12]. A *convoy* is a group of objects, containing at least m objects and these objects are density-connected with respect to distance e during k consecutive time points. Comparing with moving cluster and flock, convoy is a more flexible definition to find moving object clusters but it is still confined to strong constraint on consecutive time. So we compare our effectiveness with convoy in Section 5.

C.2 Group pattern

Wang *et al.* [22] further propose to mine group patterns. The definition of group pattern is similar to that of the swarm, which also addresses time relaxation issue. Group pattern is a set of moving objects that stay within a disc with max_dis radius for min_wei period and each consecutive time segment is no less than min_dur. [22] develops VG-Growth method whose general idea is depth-first search based on conditional VG-graph. Although the idea of group pattern is well-motivated, the problem is not well defined. First, the "closeness" of moving objects is confined to be within a max_dis disk. A fixed max_dis for all group patterns could not produce natural cluster shapes. Second, since it does not consider the closure property of group patterns, it will produce an exponential number of redundant patterns that severely hinders efficiency. All these problems can be solved in our work by using density-based clustering to define "closeness" flexibly and introducing closed swarm definition.

To make fair comparison on efficiency in Section 5, we adapt VG-Growth to accommodate clusters as input. We set $min\_dur = 1$ and $min\_wei = min_t$. Since the search space of VG-Growth is the same as our methods to produce swarms, it is equivalent to compare the latter ones with our proposed closed swarm methods. To produce swarms, we can simply omit the Backward Pruning rule and Forward Closure Checking in ObjectGrowth. So VG-Growth is essentially searching on objectset and using Apriori pruning rule only.

C.3 Closed frequent pattern mining

In overview of our algorithms in Section 4, we mention there are three major pruning techniques for closed itemset mining in previous works [19, 25, 21]. (1) *Item merging:* Let X be a frequent itemset. If every transaction containing itemset X also contains itemset Y but not any proper superset of Y, then $X \cup Y$ forms a frequent closed itemset and there is no need to search any itemset containing X but no Y. (2) *Sub-itemset pruning*: Let X be the frequent itemset currently under consideration. If X is a proper subset of an already found frequent closed itemset Y and support of X is equal to that of Y, then X and all of X's descendants in the set enumeration tree cannot be frequent closed itemsets and thus can be pruned. (3) *Item skipping:* If a local frequent item has the same support in several header tables at different levels, one can safely prune it from the header tables at higher levels. It is easy to see that all these three pruning strategies are all covered by our one simple Backward Pruning rule. Thus, we consider our Backward Pruning rule is a novel pruning strategy that is able to detect several redundant cases at the same time.

D. PRE-PROCESSING

D.1 Obtaining clusters

The clustering method is not fixed in our framework. One can cluster cars along highways using a density-based method, or cluster birds in 3-D space using the k-means algorithm. Clustering methods that generate overlapping clusters are also applicable, such as EM algorithm or using ϵ-disk to define a cluster. Also, clustering parameters are decided by users' requirements or can be indirectly controlled by users' expectation on the number of clusters at each timestamp.

Usually, most of clustering methods can be done in polynomial time. In our experiment, we used DBSCAN [7], which takes $O(|O_{DB}| \times \log|O_{DB}| \times |T_{DB}|)$ in total to do clustering at every timestamp. Comparing with exponential search space of swarms, such polynomial time in preprocessing step is acceptable. To speed it up, there are also many incremental clustering methods for moving object. Instead computing clusters from scratch at each timestamp, clusters can be incrementally updated from last timestamp.

D.2 Estimation of missing points

For a moving object, there could be many interpolation methods to fill in the missing points based on its own movement history. Among all, linear interpolation is the most commonly used method. Here, we propose a method based on linear interpolation to obtain the probability on the estimation of missing points.

For a missing point, we only consider its immediate last recorded point and immediate next recorded point. Given two reported locations (x_0, y_0) at time t_0 and (x_1, y_1) at time t_1, we need to fill in the points for any timestamp between t_0 and t_1. We assume the moving object follows the linear model. The intuition to obtain the probability is that for the timestamp that is closer to t_0 or t_1, the linearly interpolated points have higher probabilities to be correct. Therefore, the probability at $t(t_0 \leq t \leq t_1)$ can be calculated as $e^{-\lambda \times \min\{t-t_0, t_1-t\}}$, where $\lambda > 0$ is used to control the degree of sharpness in the probability function. In the extreme cases, when $t = t_0$ or $t = t_1$, the probability equals to 1.0.

After we obtain an initial estimation of the missing points, ObjectGrowth+ method can be applied to mine swarms. In turn, discovered swarms can be further used to adjust the missing points. Since the initial linear interpolation is a rough estimation of real locations based on one's own movement history, swarms can help better estimate the missing points from other similar movements. The general idea is that, if we find an object is in a swarm with objectset O and the position of o at timestamp t is estimated, then this position can be adjusted towards those reported locations of O at t and the probability is updated accordingly. We consider such iterative framework to refine the swarms and missing points as a promising future work. ‘

E. EXTENSION

E.1 Search based on timeset

As ObjectGrowth is based on objectset search space, similarly, the search could be conducted on timeset space. Apriori Pruning and Backward Pruning in ObjectGrowth can be easily adapted to prune the unnecessary timeset search space and Forward Closure Checking can also used to discover closed swarms during the search space.

The major difference between two search directions is that if we fix one timeset, there could be more than one maximal corresponding objectset. For example, in the running example, if we fix the timeset as $\{t_1\}$, there are two maximal corresponding objectsets: $\{o_3\}$ and $\{o_1, o_2, o_4\}$. However, when the objectset is fixed, there is only one maximal corresponding timeset. So on the node of DFS tree based on timeset, we need to maintain a timeset T and set of corresponding objectsets. Accordingly, the rules need to be modified. For Apriori Pruning rule, once there is one corresponding objectset has less than min_o objects, it can be deleted. And for a node with timeset T, there is no more remaining corresponding objectset, it can be pruned. For Backward Pruning rule, we add one more timestamp $t_{i'}$ ($i' < i_m$ and $t_{i'} \notin T$) in $T = \{t_{i_1}, t_{i_2}, \ldots, t_{i_m}\}$. If every maximal corresponding objectset remains unchanged, this node can be pruned. Similarly, for Forward Closure Checking, if we add $t_{i'}$ ($i' > i_m$) into T, and every maximal corresponding objectset remains unchanged, this node is *not* closed. Finally, for any node passed all the rules and $|T| \geq min_t$, it is a closed swarm.

Comparing two different search methods, ObjectGrowth is suitable for mining the group of moving objects that travel together for considerably *long* time, whereas search based on timeset is more efficient at finding the *large* group of objects moving together. In most real applications, we usually tack a certain set of moving objects over long time, for example, tracking 100 moving objects over a year. Therefore, it is usually the case that we have $|T_{DB}| \gg |O_{DB}|$. So search based on timeset often is less efficient than the one based on timeset because its search space is much larger. Thus, we introduce ObjectGrowth as our major algorithm.

E.2 Enforcing gap constraint

In the case that min_t is much smaller than $|T_{DB}|$, there could be two kinds of swarm discovered with rather different meanings. For example, if the whole time span is 365 days, min_t is set to be 30 (days), a swarm could be a group of objects that move together for only one month but keep far away for the rest 11 months or a set of objects gather in each month during the whole year.

For the first case, we can specify a range of time period and then discover swarms. For the latter, it requires more strategies. One solution could be, for a swarm (O, T), we enforce gap constraint on the time dimension T. For example, if we set $min\_gap = 7(days)$ and suppose there are two objects being together for 14 consecutive days, these 10 days can contribute at most 2 to min_t because we require there should be at least a gap with length 7 between any two timestamps in T.

To further embed such min_gap in our ObjectGrowth method, we can compute an upper bound for $T_{max}(O)$ at each node of the search tree with objectset O. The upper bound can be computed using greedy algorithm as shown Algorithm 2. Accordingly, the size of $T_{max}(O)$ is no longer simply measure by $|T_{max}(O)|$. Instead, it should be measured by its upper bound. And the pruning rules are all affected accordingly.

Algorithm 2 Calculate upper bound of $T_{max}(O)$

Input: $T_{max}(O) = \{t_1, t_2, \ldots, t_m\}(t_1 < t_2 < \cdots < t_m)$ and min_gap.
Output: upper bound of $T_{max}(O)$.

```
1: upper_bound ← 1;
2: last_t ← t_m;
3: for i ← m − 1 to 1 do
4:    if last_t − t_i > min_gap then
5:       upper_bound ← upper_bound + 1;
6:       last_t ← t_i;
7: Return upper_bound;
```

E.3 Sampling

The size of the trajectory dataset has two factors $|O_{DB}|$ and $|T_{DB}|$. In animal movements, $|O_{DB}|$ is usually relatively small because it is expensive to track animals so the number of animals being tracked seldom goes up to hundreds. When tracking vehicles, the number could be as large as thousands. For both cases, $|T_{DB}|$ is usually large. When $|T_{DB}|$ is very large, we may use sampling as a pre-processing step to reduce the data size. For example, if the location sampling rate is every second, $|T_{DB}|$ will be 86400 for one day. We can first sample one location in every minute to reduce the size to 1500. This is based on the assumption that one moving object will not travel too far away within a considerable short time.

Acknowledgment

We would like to thank all the reviewers for their valuable comments to improve this work.

The work was supported in part by the Boeing company, NSF BDI-07-Movebank, NSF CNS-0931975, U.S. Air Force Office of Scientific Research MURI award FA9550-08-1-0265, and by the U.S. Army Research Laboratory under Cooperative Agreement Number W911NF-09-2-0053 (NS-CTA). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.