A General and Parallel Platform for Mining Co-Movement Patterns over Large-scale Trajectories

Qi Fan [#], Dongxiang Zhang [♮], Huayu Wu [▽], Kian-Lee Tan[#*]
# NUS Graduate School for Integrative Science and Technology, Singapore
♮ University of Electronic Science and Technology of China, China
▽ Institute for Infocomm Research (I^2R), Singapore
* School of Computing, NUS, Singapore
fan.qi@u.nus.edu, zhangdo@uestc.edu.cn, huwu@i2r.a-star.edu.sg, tankl@comp.nus.edu.sg

ABSTRACT

Discovering co-movement patterns from large-scale trajectory databases is an important mining task and has a wide spectrum of applications. Previous studies have identified several types of interesting co-movement patterns and showcased their usefulness. In this paper, we make two key contributions to this research field. First, we propose a more general co-movement pattern to unify those defined in the past literature. Second, we propose two types of parallel and scalable frameworks and deploy them on Apache Spark. To the best of our knowledge, this is the first work to mine co-movement patterns in real life trajectory databases with hundreds of millions of points. Experiments on three real life large-scale trajectory datasets have verified the efficiency and scalability of our proposed solutions.

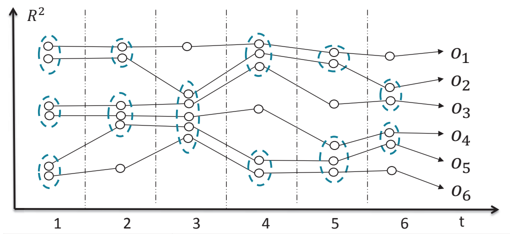
1. INTRODUCTION

The prevalence of positioning devices has drastically expanded the scale and spectrum of trajectory collection to an unprecedented level. Tremendous amounts of trajectories, in the form of sequenced spatial-temporal records, are continually being generated from animal telemetry chips, vehicle GPSs and wearable devices. Data analysis on large-scale trajectories benefits a wide range of applications and services, including traffic planning [25], animal analysis [17], location-aware advertising [7], and social recommendations [3], to name just a few.

A crucial task of data analysis on top of trajectories is to discover co-moving objects. A *co-movement* pattern [14, 24] refers to a group of objects traveling together for a certain period of time and the group is normally determined by their spatial proximity. A pattern is prominent if the size of the group exceeds M and the length of the duration exceeds K, where M and K are parameters specified by users. Rooted from such a basic definition and driven by different mining applications, there are many variants of co-movement patterns that have been developed with additional constraints.

Proceedings of the VLDB Endowment, Vol. 10, No. 4
Copyright 2016 VLDB Endowment 2150-8097/16/12.

Table 1 summarizes several popular co-movement patterns with different constraints with respect to clustering in spatial proximity, consecutiveness in temporal duration and computational complexity. In particular, the *flock* [6] and the *group* [20] patterns require all the objects in a group to be enclosed by a disk with radius r; whereas the *convoy* [8], the *swarm* [16] and the *platoon* [15] patterns resort to density-based spatial clustering. In the temporal dimension, the *flock* and the *convoy* require all the timestamps of each detected spatial group to be consecutive, which is referred to as *global consecutiveness*; whereas the *swarm* does not impose any restriction. The *group* and the *platoon* adopt a compromised approach by allowing arbitrary gaps between consecutive segments, which is called *local consecutiveness*. They introduce a parameter L to control the minimum length of each local consecutive segment.



Flock (M=2, K=3)	$\langle o_3, o_4: 1,2,3\rangle$ $\langle o_5, o_6: 3,4,5\rangle$
Convoy (M=2, K=3)	$\langle o_3, o_4: 1,2,3\rangle$ $\langle o_5, o_6: 3,4,5\rangle$
Group (M=2, L=2)	$\langle o_1, o_2: 1,2,4,5\rangle$ $\langle o_3, o_4, o_5: 2,3\rangle$ $\langle o_5, o_6: 3,4,5\rangle$
Platoon (M=2,K=3,L=2)	$\langle o_1, o_2: 1,2,4,5\rangle$ $\langle o_3, o_4: 1,2,3\rangle$ $\langle o_4, o_5: 2,3,5,6\rangle$ $\langle o_5, o_6: 3,4,5\rangle$
Swarm (M=2,K=3)	$\langle o_1, o_2: 1,2,4,5\rangle$ $\langle o_3, o_4: 1,2,3\rangle$ $\langle o_4, o_5: 2,3,5,6\rangle$ $\langle o_5, o_6: 1,3,4,5\rangle$ $\langle o_2, o_3: 3,4,6\rangle$

Figure 1: Trajectories and co-movement patterns. The example consists of six trajectories across six snapshots. Objects in spatial clusters are enclosed by dotted circles. M is the minimum cluster cardinality; K denotes the minimum number of snapshots for the occurrence of a spatial cluster; and L denotes the minimum length for local consecutiveness.

Figure 1 is an example to demonstrate the concepts of the various co-movement patterns. The trajectory database consists of six moving objects and the temporal dimension is discretized into six snapshots. In each snapshot, we treat the clustering method as a blackbox and assume that they generate the same clusters. Objects in proximity are grouped in the dotted circles. As aforementioned, there are three parameters to determine the co-movement patterns and the default settings in this example are $M = 2$, $K = 3$ and

Table 1: Constraints and complexities of co-movement patterns. The time complexity indicates the performance wrt. $|\mathbb{O}|$, $|\mathbb{T}|$ in the worst case, where $|\mathbb{O}|$ is the number of objects, and $|\mathbb{T}|$ is the number of discretized timestamps.

Pattern	**Proximity**	**Consecutiveness**	**Time Complexity**
flock [6]	disk	global	$O(\lvert\mathbb{O}\rvert\lvert\mathbb{T}\rvert\log(\lvert\mathbb{O}\rvert))$
convoy [8]	density	global	$O(\lvert\mathbb{O}\rvert^2 + \lvert\mathbb{O}\rvert\lvert\mathbb{T}\rvert)$
swarm [16]	density	-	$O(2^{\lvert\mathbb{O}\rvert}\lvert\mathbb{O}\rvert\lvert\mathbb{T}\rvert)$
group [20]	disk	local	$O(\lvert\mathbb{O}\rvert^2\lvert\mathbb{T}\rvert)$
platoon [15]	density	local	$O(2^{\lvert\mathbb{O}\rvert}\lvert\mathbb{O}\rvert\lvert\mathbb{T}\rvert)$

$L = 2$. Both the *flock* and the *convoy* require the spatial clusters to last for at least K consecutive timestamps. Hence,$\langle o_3, o_4 : 1, 2, 3\rangle$ and $\langle o_5, o_6 : 3, 4, 5\rangle$ are the only two candidates matching the patterns. The *swarm* relaxes the pattern matching by discarding the temporal consecutiveness constraint. Thus, it generates many more candidates than the *flock* and the *convoy*. The *group* and the *platoon* add another constraint on local consecutiveness to retain meaningful patterns. For instance, $\langle o_1, o_2 : 1, 2, 4, 5\rangle$ is a pattern matching local consecutiveness because timestamps $(1, 2)$ and $(4, 5)$ are two segments with length no smaller than $L = 2$. The difference between the *group* and the *platoon* is that the *platoon* has an additional parameter K to specify the minimum number of snapshots for the spatial clusters. This explains why $\langle o_3, o_4, o_5 : 2, 3\rangle$ is a *group* pattern but not a *platoon* pattern.

As can be seen, there are various co-movement patterns requested by different applications and it is cumbersome to design a tailored solution for each type. In addition, despite the generality of the *platoon* (i.e., it can be reduced to other types of patterns via proper parameter settings), it suffers from the so-called *loose-connection* anomaly. We use two objects o_1 and o_2 in Figure 2 to illustrate the scenario. These two objects form a *platoon* pattern in timestamps $(1, 2, 3, 102, 103, 104)$. However, the two consecutive segments are 98 timestamps apart, resulting in a false positive co-movement pattern. In reality, such an anomaly may be caused by the periodic movements of unrelated objects, such as vehicles stopping at the same petrol station or animals pausing at the same water source. Unfortunately, none of the existing patterns have directly addressed this anomaly.

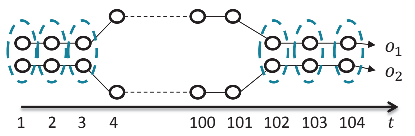
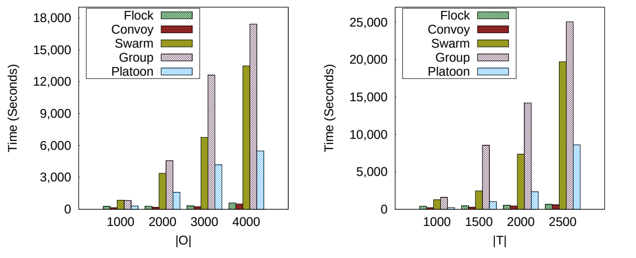


Figure 2: *Loose-connection* anomaly. Even though $\langle o_1, o_2\rangle$ is considered as a valid *platoon* pattern, it is highly probable that these two objects are not related as the two consecutive segments are 98 timestamps apart.

The other issue with existing methods is that they are built on top of centralized indexes which may not be scalable. Table 1 shows their theoretical complexities in the worst cases and the largest real dataset ever evaluated in previous studies is up to million-scale points collected from hundreds of moving objects. In practice, the dataset is of much higher scale and the scalability of existing methods is left unknown. Thus, we conduct an experimental evaluation with 4000 objects moving for 2500 timestamps to examine the scalability. Results in Figure 3 show that their performances degrade dramatically as the dataset scales up. For instance, the detection time of *group* drops twenty times as the number of objects grows from *1k* to *4k*. Similarly, the performance of *swarm* drops over fifteen times as the number of snapshots grows from *1k* to *2.5k*. These observations imply that existing methods are not scalable to support large-scale trajectory databases.



(a) Varying No. of objects (b) Varying No. of timestamps

Figure 3: Performance measures on existing co-movement patterns. A sampled GeoLife dataset is used with upto 2.4 million data points. Default parameters are $M = 15$, $K = 180$, $L = 30$.

In this paper, we close these two gaps by making the following contributions. First, we propose the *general co-movement pattern* (GCMP) which models various co-moment patterns in a unified way and can avoid the *loose-connection* anomaly. In GCMP, we introduce a new gap parameter G to pose a constraint on the temporal gap between two consecutive segments. By setting a feasible G, the loose-connection anomaly can be effectively controlled. In addition, our GCMP is also general. It can be reduced to any of the previous pattern by customizing its parameters.

Second, we investigate deploying our GCMP detector on the modern MapReduce platform (i.e., Apache Spark) to tackle the scalability issue. Our technical contributions are threefold. First, we design a baseline solution by replicating the snapshots to support effective parallel mining. Second, we devise a novel *Star Partitioning and ApRiori Enumerator* (SPARE) framework to resolve limitations of the baseline. SPARE achieves workload balance by partitioning objects into fine granular stars. For each partition, an Apriori Enumerator is adopted to mine the co-movement patterns. Third, we leverage the *temporal monotonicity* property of GCMP to design several optimization techniques including *sequence simplification*, *monotonicity pruning* and *forward closure check* to further reduce the number of candidates enumerated in SPARE.

We conduct a set of extensive experiments on three large-scale real datasets with hundreds of millions of temporal points. The results show that both our parallel schemes efficiently support GCMP mining in large datasets. In particular, with over 170 million trajectory points, SPARE achieves upto 112 times speedup using 162 cores as compared to the state-of-the-art centralized schemes. Moreover, SPARE further achieves almost linear scalability with upto 14 times efficiency as compared to the baseline algorithm.

The rest of our paper is organized as follows: Section 2 summarizes related work. Section 3 states the problem of general co-movement pattern mining. Section 4 provides a baseline solution. An advanced solution named *Star Partitioning and ApRiori Enumerator* (SPARE) is presented in Section 5. Section 6 reports our experimental evaluation. Finally, Section 7 concludes the paper.

2. RELATED WORK

Related work can be grouped into three categories: *co-movement patterns*, *dynamic movement patterns* and *trajectory mining frameworks.*

2.1 Co-Movement Patterns

2.1.1 Flock and Convoy

The difference between *flock* and *convoy* lies in the object clustering methods. In *flock*, objects are clustered based on their distances. Specifically, the objects in the same cluster need to have a pairwise distance less than *min_dist.* This essentially requires the objects to be within a disk-region of delimiter less than *min_dist.* In contrast, *convoy* clusters objects using density-based spatial clustering [5]. Technically, *flock* utilizes a m^{th}-order Voronoi diagram [12] to detect whether a subset of n ($n \geq m$) objects stay in a disk region. *Convoy* employs a trajectory simplification [4] technique to boost pairwise distance computations in the density-based clustering. After clustering, both *flock* and *convoy* use a sequential scanning method to examine each snapshot. During the scan, the object groups that do not appear in consecutive snapshots are pruned. However, such a method faces high complexity when supporting other patterns. For instance, in *swarm*, the candidate set during the sequential scanning grows exponentially, and many candidates can only be pruned after the entire dataset are scanned.

2.1.2 Group, Swarm and Platoon

Different from *flock* and *convoy*, all the *group*,*swarm* and *platoon* patterns have more relaxed constraints on the pattern duration. Therefore, their techniques of mining are of the same skeleton. The main idea of mining is to grow an object set from an empty set in a depth-first manner. During the growth, various pruning techniques are provided to prune unnecessary branches. *Group* pattern uses a VG-graph to guide the pruning of false candidates [20]. *Swarm* designs two more pruning rules called backward pruning and forward pruning [16]. *Platoon* [15] leverages a prefix table structure to steer the depth-first search, which shows efficiency as compared to the other two methods. However, the pruning rules adopted by the three patterns heavily rely on depth-first search which loses efficiency in a parallel scenario.

2.2 Other Related Trajectory Patterns

A closely related literature to co-movement patterns is the *dynamic movement* patterns. Instead of requiring the same set of object traveling together, *dynamic movement* patterns allow objects to temporally join or leave a group. Typical works include *moving clusters* [10], *evolving convoy* [2], *gathering* [23] etc. These works cannot model GCMP since they enforce global consecutiveness on the temporal domain.

2.3 Trajectory Mining Frameworks

Jinno et al. in [9] designed a MapReduce based algorithm to efficiently support T-pattern discovery, where a T-pattern is a set of objects visiting the same place at simliar time. Li et al. proposed a framework of processing online *evolving group* pattern [13], which focuses on supporting efficient updates of arriving objects. As these works essentially differ from co-movement pattern, their techniques cannot be directly applied to discover GCMPs.

3. PROBLEM STATEMENT

Let $\mathbb{O} = \{o_1, o_2, ..., o_n\}$ be the set of objects and $\mathbb{T} = (1, 2, ..., N)$ be the discretized temporal dimension. A time sequence T is defined as an ordered subset of $\mathbb{T}$. Given two time sequences T_1 and T_2, we define commonly-used operators in this paper in Table 2.

Table 2: Operators on time sequence.

Operator	**Definition**
$T[i]$	the i-th element in the sequence T
$\|T\|$	the number of elements in T
$\max(T)$	the maximum element in T
$\min(T)$	the minimum element in T
range(T)	the range of T, i.e., $\max(T) - \min(T) + 1$
$T[i : j]$	subsequence of T from $T[i]$ to $T[j]$ (inclusive)
$T_1 \subseteq T_2$	$\forall T_1[x] \in T_1$, we have $T_1[x] \in T_2$.
$T_3 = T_1 \cup T_2$	$\forall T_3[x] \in T_3$, we have $T_3[x] \in T_1$ or $T_3[x] \in T_2$
$T_3 = T_1 \cap T_2$	$\forall T_3[x] \in T_3$, we have $T_3[x] \in T_1$ and $T_3[x] \in T_2$

We say a sequence T is consecutive if $\forall 1 \leq i < |T|, T[i+1] = T[i] + 1$. We refer to each consecutive subsequence of T as a *segment.* It is obvious that any time sequence T can be decomposed into segments and we say T is *L-consecutive* [15] if the length of every segment is no smaller than L. As illustrated in Figure 2, patterns adopting the notion of L-consecutiveness (e.g., *platoon* and *group*) still suffer from the *loose-connection* anomaly. To avoid such an anomaly without losing generality, we introduce a parameter G to control the gaps between timestamps in a pattern. Formally, a G-connected time sequence is defined as follows:

Definition 1 (G-connected)**.** *A time sequence T is G-connected if the gap between any of its neighboring timestamps is no greater than G, i.e., $\forall 1 \leq i < |T|, T[i+1] - T[i] \leq G$.*

We take $T = (1, 2, 3, 5, 6)$ as an example. T can be decomposed into two segments $(1, 2, 3)$ and $(5, 6)$. T is not 3-consecutive since the length of $(5, 6)$ is 2. But it is safe to say either T is 1-consecutive or 2-consecutive. On the other hand, T is 2-connected since the maximum gap between its neighboring timestamps is 2. It is worth noting that T is not 1-connected because the gap between $T[3]$ and $T[4]$ is 2 (i.e., $5 - 3 = 2$).

Given a trajectory database that is discretized into snapshots, we can conduct a clustering method, either disk-based or density-based, to identify groups with spatial proximity. Let T be the set of timestamps in which a group of objects O are clustered. We are ready to define a more general co-movement pattern:

Definition 2 (General Co-Movement Pattern)**.** *A general co-movement pattern finds a set of objects O satisfying the following five constraints: (1)* ***closeness:*** *the objects in O belong to the same cluster in every timestamps of T; (2)* ***significance:*** *$|O| \geq M$; (3)* ***duration:*** *$|T| \geq K$; (4)* ***consecutiveness:*** *T is L-consecutive; and (5)* ***connection:*** *T is G-connected.*

There are four parameters in our general co-movement pattern, including object constraint M and temporal constraints K, L, G. By customizing these parameters, our pattern can express other patterns proposed in the literature, as illustrated in Table 3. In particular, by setting $G = |\mathbb{T}|$, we achieve the *platoon* pattern. By setting $G = |\mathbb{T}|, L = 1$, we reach the *swarm* pattern. By setting $G = |\mathbb{T}|$, $M = 2$, $K = 1$, we gain the *group* pattern. Finally by setting $G = 1$, we result in the *convoy* and *flock* patterns. In addition to the flexibility of representing other existing patterns, our GCMP is able to avoid the *loose-connection* anomaly by tuning the parameter G.

Table 3: Expressing other patterns using GCMP. · indicates a user specified value.

Pattern	M	K	L	G	**Clustering**
Group	2	1	2	$\|\mathbb{T}\|$	disk
Flock	·	·	K	1	disk
Convoy	·	·	K	1	density
Swarm	·	·	1	$\|\mathbb{T}\|$	density
Platoon	·	·	·	$\|\mathbb{T}\|$	density

Our definition of GCMP is independent of the clustering method. Users can apply different clustering methods to facilitate different application needs. We currently expose both disk-region based clustering and DBSCAN as options to the users. In summary, the goal of this paper is to present a parallel solution for discovering all the valid GCMPs from large-scale trajectory databases. Before we move on to the algorithmic part, we list the notations that are used in the following sections.

Table 4: Summary of notations.

Symbol	**Meaning**
S_t	snapshot of objects at time t
M	significance constraint
K	duration constraint
L	consecutiveness constraint
G	connection constraint
$P = \langle O : T \rangle$	pattern with object set O, time sequence T
S_t	set of clusters at snapshot t
η	replication factor in the TRPM framework
λ_t	partition with snapshots $S_t, .., S_{t+\eta-1}$
G_A	aggregated graph in SPARE framework
Sr_i	star partition for object (vertex) i

4. BASELINE: TEMPORAL REPLICATION AND PARALLEL MINING

In this section, we propose a baseline solution that resorts to MapReduce as a general, parallel and scalable paradigm for GCMP mining. The framework, named *temporal replication and parallel mining* (TRPM), is illustrated in Figure 4. There are two stages of mapreduce jobs connected in a pipeline manner. The first stage deals with spatial clustering of objects in each snapshot, which can be seen as a preprocessing step for the subsequent pattern mining phase. In particular, for the first stage, the timestamp is treated as the key in the map phase and objects within the same snapshot are clustered (DBSCAN or disk-based clustering) in the reduce phase. Finally, the reducers output clusters of objects in each snapshot, represented by a list of key-value pairs $\langle t, S_t \rangle$, where t is the timestamp and S_t is a set of clustered objects at snapshot t.

Our focus in this paper is on the second mapreduce stage of parallel mining, which essentially addresses two key challenges. The first is to ensure effective data partitioning such that the mining on each partition can be conducted independently; and the second is to efficiently mine the valid patterns within each partition.

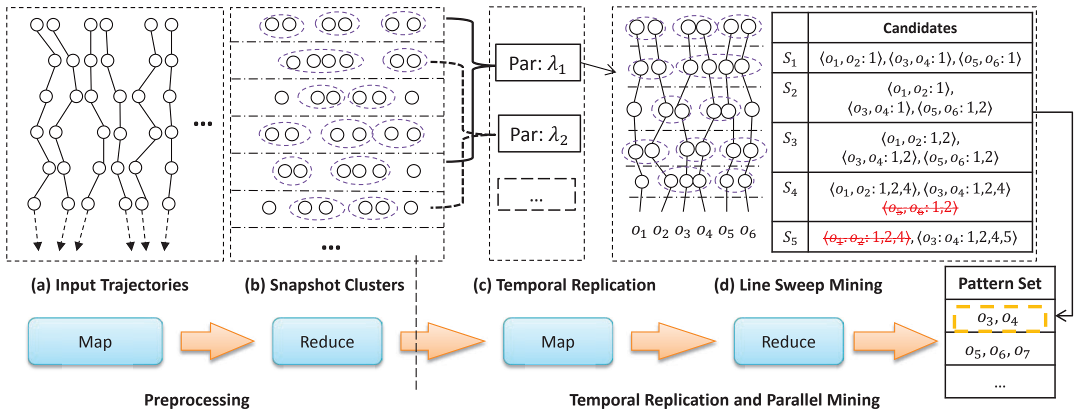
It is obvious that we cannot simply split the trajectory database into disjoint partitions because a GCMP requires L-consecutiveness and the corresponding segments may span multiple partitions. Our strategy is to use data replication to enable parallel mining. Each snapshot will replicate its clusters to $\eta - 1$ preceding snapshots. In other words, the partition for the snapshot S_t contains clusters in S_t, $S_{t+1} \ldots, S_{t+\eta-1}$. Determining a proper η is critical in ensuring the correctness and efficiency of TRPM. If η is too small, certain cross-partition patterns may be missed. If η is too large, expensive network communication and CPU processing costs would be incurred in the map and reduce phases respectively. Our objective is to find an η that is not large but can guarantee correctness.

In our implementation, we set $\eta = (\lceil \frac{K}{L} \rceil - 1) * (G-1) + K + L - 1$. Intuitively, with K timestamps, at most $\lceil \frac{K}{L} \rceil - 1$ gaps may be generated as the length of each L-consecutive segment is at least L. Since the gap size is at most $G - 1$, $(\lceil \frac{K}{L} \rceil - 1) * (G - 1)$ is the upper bound of timestamps allocated to gaps. The remaining part of the expression, $K + L - 1$, is used to capture the upper bound allocated for the L-consecutive segments. We formally prove that η can guarantee correctness.

Theorem 1. *$\eta = (\lceil \frac{K}{L} \rceil - 1) * (G-1) + K + L - 1$ guarantees that no valid pattern is missed.*

Proof. Given a valid pattern P, we can always find at least one valid subsequence of $P.T$ that is also valid. Let T' denote the valid subsequence of $P.T$ with the minimum length. In the worst case, $T' = P.T$. We define range$(T) = \max(T) - \min(T) + 1$ and prove the theorem by showing that range$(T') \leq \eta$. Since T' can be written as a sequence of L-consecutive segments interleaved by gaps: $l_1, g_1, \ldots, l_{n-1}$, g_{n-1}, l_n $(n \geq 1)$, where l_i is a segment and g_i is a gap. Then, range(T') is calculated as $\Sigma_{i=1}^{i=n}|l_i| + \Sigma_{i=1}^{i=n-1}|g_i|$. Since T' is valid, then $\Sigma_{i=1}^{i=n}|l_i| \geq K$. As T' is minimum, if we remove the last l_n, the resulting sequence should not be valid. Let $K' = \Sigma_{i=1}^{i=n-1}|l_i|$, which is the size of the first $(n-1)$ segments of T'. Then, $K' \leq K - 1$. Note that every $|l_i| \geq L$, thus $n \leq \lceil \frac{K'}{L} \rceil \leq \lceil \frac{K}{L} \rceil$. By using the fact that every $|g_i| \leq G - 1$, we achieve $\Sigma_{i=1}^{i=n-1}|g_i| \leq (n-1)(G-1) \leq (\lceil \frac{K}{L} \rceil - 1)(G-1)$. Next, we consider the difference between K and K', denoted by $\Delta = K - K'$. To ensure T''s validity, l_n must equal to $\min(L, \Delta)$. Then, $\Sigma_{i=1}^{i=n}|l_i| = K' + l_n = K - \Delta + \min(L, \Delta) \leq K - 1 + L$. We finish showing range$(T') \leq \eta$. Therefore, for any valid sequence T, there is at least one valid subsequence with range no greater than η and hence this pattern can be detected in a partition with η snapshots. $\square$

Based on the above theorem, under TRPM, every consecutive η snapshots form a partition. In other words, each snapshot S_t corresponds to a partition $\lambda_t = \{S_t, ..., S_{t+\eta-1}\}$. Next, we aim to design an efficient pattern mining strategy



	Candidates
S_1	$\langle o_1, o_2 : 1\rangle, \langle o_3, o_4 : 1\rangle, \langle o_5, o_6 : 1\rangle$
S_2	$\langle o_1, o_2 : 1\rangle$, $\langle o_3, o_4 : 1\rangle, \langle o_5, o_6 : 1,2\rangle$
S_3	$\langle o_1, o_2 : 1,2\rangle$, $\langle o_3, o_4 : 1,2\rangle, \langle o_5, o_6 : 1,2\rangle$
S_4	$\langle o_1, o_2 : 1,2,4\rangle, \langle o_3, o_4 : 1,2,4\rangle$ ~~$\langle o_5, o_6 : 1,2\rangle$~~
S_5	~~$\langle o_1, o_2 : 1,2,4\rangle$~~, $\langle o_3 : o_4 : 1,2,4,5\rangle$

Pattern Set
o_3, o_4
o_5, o_6, o_7
...

Figure 4: Workflow of Temporal Replication and Parallel Mining (TRPM). (a) and (b) correspond to the first mapreduce stage which clusters objects in each snapshot. (c) and (d) is the second mapreduce stage which uses TRPM to detect GCMPs.

within each partition. Our solution includes a line sweep algorithm to sequentially scan the η snapshots in a partition and an effective candidate pattern enumeration mechanism.

Algorithm 1 Line Sweep Mining

```
Require: $\lambda_t = \{S_t, ..., S_{t+\eta-1}\}$
 1: $C \leftarrow \{\}$                                    ▷ Candidate set
 2: for all clusters $s$ in snapshot $S_t$ do
 3:     if $|s| \geq M$ then
 4:         $C \leftarrow C \cup \{\langle s, t\rangle\}$
 5: for all $S_j \in \{S_{t+1}, \ldots, S_{t+\eta-1}\}$ do
 6:     $N \leftarrow \{\}$
 7:     for all $(c, s) \in C \times S_j$ do
 8:         $c' \leftarrow \langle c.O \cap s.O, c.T \cup \{j\}\rangle$
 9:         if $c'.T$ is valid then
10:             output $c'$
11:         else if $|c'.O| \geq M$ then
12:             $N \leftarrow N \cup \{c'\}$
13:     for all $c \in C$ do
14:         if $j - \max(c.T) \geq G$ then
15:             $C \leftarrow C - \{c\}$
16:             output $c$, if $c$ is a valid pattern
17:         if $c$'s first segment is less than $L$ then
18:             $C \leftarrow C - \{c\}$
19:     $C \leftarrow C \cup N$
20: output valid patterns in $C$
```

Details of the algorithm are presented in Algorithm 1. We keep a candidate set C (Line 1) during the sweeping. It is initialized using the clusters with size no smaller than M in the first snapshot. Then, we sequentially scan each snapshot (Lines 5-19) and generate new candidates by extending the original ones in C. Specifically, we join candidates in C with all the clusters in S_j to form new candidates (Lines 7-12).

After sweeping all the snapshots, all the valid patterns are stored in C (Line 20). It is worth noting that C continues to grow during sweeping. We can use three pruning rules to remove false candidates early from C. Since there is a partition λ_t for each S_t, only patterns that start from timestamp t need to be discovered. Therefore, those patterns that do not appear in the S_t are false candidates. In particular, our three pruning rules are as follows: First, when sweeping snapshot S_j, new candidates with object set smaller than M are pruned (Line 12). Second, after joining with all clusters in S_j, candidates in C with the maximum timestamp no smaller than $j - G$ are pruned (Lines 14-16). Third, candidates in C with the size of the first segment smaller than L are pruned (Lines 17-18). With the three pruning rules, the size of C can be significantly reduced.

Algorithm 2 Temporal Replication and Parallel Mining

```
Require: list of $\langle t, S_t\rangle$ pairs
 1: $\eta \leftarrow (\lceil \frac{K}{L} \rceil - 1) * (G - 1) + K + L - 1$
 2: —Map Phase—
 3: for all snapshots $S_t$ do
 4:     for all $i \in 1...\eta - 1$ do
 5:         emit key-value pair $\langle \max(t - i, 0), S_t\rangle$
 6: —Partition and Shuffle Phase—
 7: for all key-value pairs $\langle t, S\rangle$ do
 8:     group-by $t$ and emit a key-value pair $\langle t, \lambda_t\rangle$, where
        $\lambda_t = \{S_t, S_{t+1}, ..S_{t+\eta-1}\}$
 9: —Reduce Phase—
10: for all key-value pairs $\langle t, \lambda_t\rangle$ do
11:     call line sweep mining for partition $\lambda_t$
```

The complete picture of TRPM is shown in Algorithm 2. We illustrate the workflow of TRPM using Figures 4 (c) and (d) with pattern parameters $M = 2, K = 3, L = 2, G = 2$. By Theorem 1, η is calculated as $(\lceil \frac{K}{L} \rceil - 1) * (G - 1) + K + L - 1 = 5$. Therefore, in Figure 4 (c), every 5 consecutive snapshots are combined into a partition in the map phase. In Figure 4 (d), the line sweep method is illustrated for partition λ_1. Let C_i be the candidate set when sweeping snapshot S_i. Initially, C_1 contains all object sets in S_1. At snapshot S_4, the candidate $\langle o_5, o_6\rangle$ is removed because the gap between its latest timestamp (i.e., 2) and the next sweeping timestamp (i.e., 5) is 3, which violates the G-connected constraint. Next, at snapshot S_5, the candidate $\langle o_1, o_2\rangle$ is removed because its local consecutive segment (4) has only 1 element, which violates the L-consecutive constraint. Finally, $\langle o_3, o_4\rangle$ is the valid pattern and is returned. Note that in this example, $\eta = 5$ is the minimum setting that can guarantee correctness. If η is set to be 4, the pattern $\langle o_3, o_4\rangle$ would be missed.

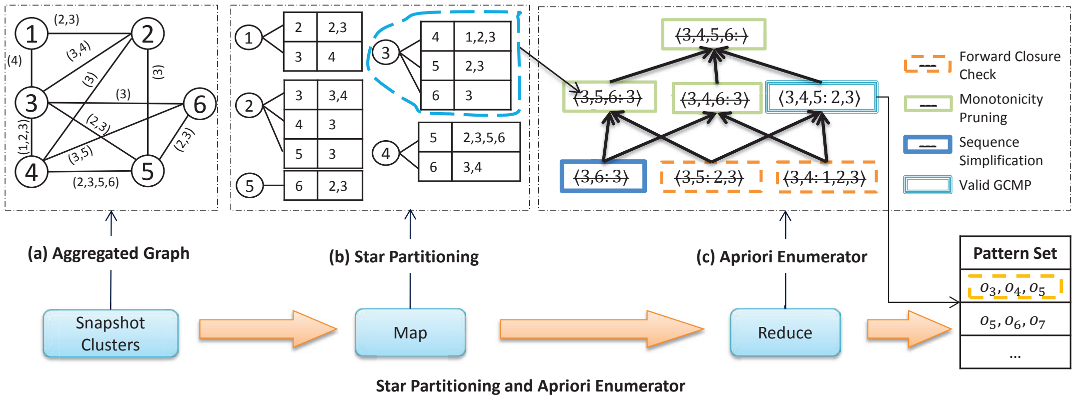


Figure 5: Star Partitioning and ApRiori Enumerator (SPARE). (a) Aggregated graph G_A generated from Figure 1. (b) Five star partitions are generated from G_A. Star IDs are circled, vertexes and inverted lists are in the connected tables. (c) Apriori Enumerator with various pruning techniques.

5. SPARE: STAR PARTITIONING AND APRIORI ENUMERATOR

The aforementioned TRPM scheme replicates snapshots based on the temporal dimension which suffers from two drawbacks. First, the replication factor η can be large. Second, the same valid pattern may be redundantly discovered from different partitions. To resolve these limitations, we propose a new Star Partitioning and ApRiori Enumerator, named SPARE, to replace the second stage of the mapreduce jobs in Figure 4. Our new parallel mining framework is shown in Figure 5. Its input is the set of clusters generated in each snapshot and the output contains all the valid GCMPs. In the following, we explain the two major components: star partitioning and apriori enumerator.

5.1 Star Partitioning

Let G_t be a graph for snapshot S_t, in which each node is a moving object and two objects are connected if they appear in the same cluster. It is obvious that G_t consists of a set of small cliques. Based on G_t, we define an aggregated graph G_A to summarize the cluster relationship among all the snapshots. In G_A, two objects form an edge if they are connected in any G_ts. Furthermore, we attach an inverted list for each edge, storing the associated timestamps in which the two objects are connected. An example of G_A, built on the trajectory database in Figure 1, is shown in Figure 5 (a). As long as two objects are clustered in any timestamps, they are connected in G_A. The object pair $\langle o_1, o_2\rangle$ appears in two clusters at timestamps 2 and 3 and is thus associated with an inverted list $(2, 3)$.

We use *star* [21] as the data structure to capture the pair relationships. To avoid duplication, as G_t is an undirected graph and an edge may appear in multiple stars, we enforce a global vertex ordering among the objects and propose a concept named *directed star*.

Definition 3 (Directed Star). *Given a vertex with global ID s, its directed star Sr_s is defined as the set of neighboring vertexes with global ID $t > s$. We call s the star ID.*

With the global vertex ordering, we can guarantee that each edge is contained in a unique star partition. Given the aggregated graph G_A in Figure 5 (a), we enumerate all the possible directed stars in Figure 5 (b). These stars are emitted from mappers to different reducers. The key is the star ID and the value is the neighbors in the star as well as the associated inverted lists. The reducer will then call the Apriori-based algorithm to enumerate all the valid GCMPs.

Before we introduce the Apriori Enumerator, we are interested to examine the issue of global vertex ordering. This is because assigning different IDs to the objects will produce different star partitioning results, which will eventually affect the workload balance among reducers. The job with the performance bottleneck is often known as a *straggler* [11]. In the context of star partitioning, a straggler refers to the job assigned with the maximum star partition. We use Γ to denote the size of such straggler partition and Γ is set to the number of edges in a directed star[1]. Clearly, a star partitioning with small Γ is preferred. For example, Figure 6 gives two star partitioning results under different vertex ordering on the same graph. The top one has $\Gamma = 5$ while the bottom one has $\Gamma = 3$. Obviously, the bottom one with smaller Γ is much more balanced.

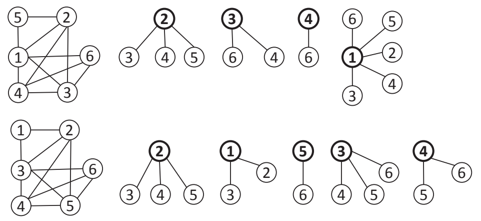


Figure 6: Star partitioning with different vertex orderings.

Although it is very challenging to find the optimal vertex ordering from the $n!$ possibilities, we observe that a random order can actually achieve satisfactory performance based on the following theorem.

Theorem 2. *Let Γ^* be the value derived from the optimal vertex ordering and Γ be the value derived from a random vertex ordering. With probability $1 - 1/n$, we have $\Gamma = \Gamma^* + O(\sqrt{n}\log n)$.*

Proof. See Appendix A. □

[1]A star is essentially a tree structure and the number of nodes equals the number of edges minus one.

If G_A is dense, we can get a tighter bound for $(\Gamma - \Gamma^*)$.

Theorem 3. *Let d be the average degree in G_A. If $d \geq \sqrt{12 \log n}$, with high probability $1-1/n$, $\Gamma = \Gamma^* + O(\sqrt{d \log n})$.*

Proof. See Appendix A. □

Hence, we can simply use object ID to determine the vertex ordering in our implementation.

5.2 Apriori Enumerator

Intuitively, given a GCMP with an object set $\{o_1, \ldots, o_m\}$, all the pairs of $\langle o_i, o_j \rangle$ with $1 \leq i < j \leq m$ must be connected in the associated temporal graphs $\{G_t\}$. This inspires us to leverage the classic Apriori algorithm [1] to enumerate all the valid GCMPs starting from pairs of objects. However, we observe that the monotonicity property does not hold between an object set and its supersets.

Example 1. *In this example, we show that if an object set is not a valid pattern, we cannot prune all its super sets. Consider two candidates $P_1 = \langle o_1, o_2 : 1, 2, 3, 6 \rangle$ and $P_2 = \langle o_1, o_3 : 1, 2, 3, 7 \rangle$. Let $L = 2, K = 3$ and $G = 2$. Both candidates are not valid patterns because the constraint on L is not satisfied. However, when considering their object superset $\langle o_1, o_2, o_3 \rangle$, we can infer that their co-clustering timestamps are in $(1, 2, 3)$. This is a valid pattern conforming to the constraints of L, K, G. Thus, we need a new type of monotonicity to facilitate pruning.*

5.2.1 Monotonicity

To ensure monotonicity, we first introduce a procedure named *sequence simplification*, to reduce the number of edges as well as unnecessary timestamps in the inverted lists. For instance, if the size of the inverted list for an edge e is smaller than K, then the edge can be safely removed because the number of timestamps in which its supersets are clustered must also be smaller than K. To generalize the idea, we propose three concepts: *maximal G-connected subsequence*, *decomposable sequence* and *sequence simplification.*

Definition 4 (Maximal G-connected Subsequence)**.** *A sequence T' is said to be a maximal G-connected subsequence of T if (1) T' is the subsequence of T, (2) T' is G-connected, and (3) there exists no other subsequence T'' of T such that T' is the subsequence of T'' and T'' is G-connected.*

Example 2. *Suppose $G = 2$ and consider two sequences $T_1 = (1, 2, 4, 5, 6, 9, 10, 11, 13)$ and $T_2 = (1, 2, 4, 5, 6, 8, 9)$. T_1 has two maximal 2-connected subsequences:$T_1^A = (1, 2, 4, 5, 6)$ and $T_1^B = (9, 10, 11, 13)$. This is because the gap between T_1^A and T_1^B is 3 and it is impossible for the timestamps from T_1^A and T_1^B to form a new subsequence with $G \leq 2$. Since T_2 is 2-connected, T_2 has only one maximal 2-connected subsequence which is itself.*

The maximal G-connected subsequence has the following two properties:

Lemma 1. *Suppose $\{T_1, T_2, \ldots, T_m\}$ is the set of all maximal G-connected subsequences of T, we have (1) $T_i \cap T_j = \emptyset$ for $i \neq j$ and (2) $T_1 \cup T_2 \cup \ldots \cup T_m = T$.*

Proof. We assume $T_i \cap T_j \neq \emptyset$. Let $T_i = (T_i[1], T_i[2], \ldots, T_i[p])$ and $T_j = (T_j[1], T_j[2], \ldots, T_j[n])$. Suppose $T[x]$ is a timestamp occurring in both T_i and T_j. Let $T[y] = \min\{T_i[1], T_j[1]\}$, i.e., the minimum timestamp of $T_i[1]$ and $T_j[1]$ occurs at the y-th position of sequence T. Similarly, we assume $T[z] = \max\{T_i[p], T_j[n]\}$. Apparently, the two subsequences $T[y : x]$ and $T[x : z]$ are G-connected because T_i and T_j are both G-connected. Then, sequence $(T_y, \ldots, T_x, \ldots, T_z)$, the superset of T_i and T_j, is also G-connected. This contradicts with the assumptions that T_i and T_j are maximal G-connected subsequences.

To prove (2), we assume $\cup_{i=1}^{i=m} T_i$ does not cover all the timestamps in T. Then, we can find a subsequence $T' = T[x : x+t]$ such that $T[x-1] \in T_a$ $(1 \leq a \leq m)$, $T[x+t+1] \in T_b$ $(1 \leq b \leq m)$ and all the timestamps in T' is not included in any T_i. Let $g' = \min\{T[x] - T[x-1], T[x+t+1] - T[x+t]\}$. If $g' \leq G$, then it is easy to infer that T_a or T_b is not a maximal G-connected subsequence because we can combine it with $T[x]$ or $T[x+t]$ to form a superset which is also G-connected. If $g' > G$, T' itself is a maximal G-connected subsequence which is missed in $\cup_{i=1}^{i=m} T_i$. Both cases lead to contradictions. □

Lemma 2. *If $T_1 \subseteq T_2$, then for any maximal G-connected subsequence T_1' of T_1, we can find a maximal G-connected subsequence T_2' of T_2 such that $T_1' \subseteq T_2'$.*

Proof. Since $T_1' \subseteq T_1 \subseteq T_2$, we know T_1' is a G-connected subsequence of T_2. Based on Lemma 1, we can find a maximal G-connected subsequence of T_2, denoted by T_2', such that $T_1' \cap T_2' \neq \emptyset$. If there exists a timestamp $T_1'[x]$ such that $T_1'[x] \notin T_2'$, similar to the proof of case (1) in Lemma 1, we can obtain a contradiction. Thus, all the timestamps in T_1' must occur in T_2'. □

Definition 5 (Decomposable Sequence)**.** *T is decomposable if for any of its maximal G-connected subsequence T', we have (1) T' is L-consecutive; and (2) $|T'| \geq K$.*

Example 3. *Let $L = 2, K = 4$ and we follow the above example. T_1 is not a decomposable sequence because one of its maximal 2-connected subsequence (i.e., T_1^B) is not 2-consecutive. In contrast, T_2 is a decomposable sequence because the sequence itself is the maximal 2-connected subsequence, which is also 2-consecutive and with size ≥ 4.*

Definition 6 (Sequence Simplification)**.** *Given a sequence T, the simplification procedure $\texttt{sim}(T) = g_{G,K} \cdot f_L(T)$ can be seen as a composite function with two steps:*

1. *f-step: remove segments of T that are not L-consecutive;*
2. *g-step: among the maximal G-connected subsequences of $f_L(T)$, remove those with size smaller than K.*

Example 4. *Take $T = (1, 2, 4, 5, 6, 9, 10, 11, 13)$ as an example for sequence simplification. Let $L = 2, K = 4$ and $G = 2$. In the f-step, T is reduced to $f_2(T) = (1, 2, 4, 5, 6, 9, 10, 11)$. The segment (13) is removed due to the constraint of $L = 2$. $f_2(T)$ has two maximal 2-consecutive subsequences: $(1, 2, 4, 5, 6)$ and $(9, 10, 11)$. Since $K = 4$, we will remove $(9, 10, 11)$ in the g-step. Finally, the output is $\texttt{sim}(T) = (1, 2, 4, 5, 6)$.*

It is possible that the simplified sequence $\texttt{sim}(T) = \emptyset$. For example, Let $T = (1, 2, 5, 6)$ and $L = 3$. All the segments will be removed in the f-step and the output is $\emptyset$. We define $\emptyset$ to be not decomposable. We then link *sequence simplification* and *decomposable sequence* in the following lemma:

Lemma 3. *If sequence T is a superset of any decomposable sequence, then* $\texttt{sim}(T) \neq \emptyset$.

Proof. It is obvious that $\texttt{sim}(T)$ is a one-to-one function. Given an input sequence T, there is a unique $\texttt{sim}(T)$. Let T_p be a decomposable subset of T and we prove the lemma by showing that $\texttt{sim}(T)$ is a superset of T_p.

Suppose T_p can be decomposed into a set of maximal G-connected subsequences $T_p^1, \ldots, T_p^m$ ($m \geq 1$). Since T_p is a subset of T, all the T_p^i are also subsets of T. By definition, each T_p^i is L-consecutive. Thus, in the f-step of $\texttt{sim}(T)$, none of T_p^i will be removed. In the g-step, based on Lemma 2, we know that each T_p^i has a superset in the maximal G-connected subsequences of $f_L(T)$. Since $|T_p^i| \geq K$, none of T_p^i will be removed in the g-step. Therefore, all the T_p^i will be retained after the simplification process and $\texttt{sim}(T) \neq \emptyset$. □

With Lemma 3, we are ready to define the *monotonicity* concept based on the simplified sequences to facilitate the pruning in the Apriori algorithm.

Theorem 4 (Monotonicity)**.** *Given a candidate pattern $P = \{O : T\}$, if $\texttt{sim}(P.T) = \emptyset$, then any pattern candidate P' with $P.O \subseteq P'.O$ can be pruned.*

Proof. We prove by contradiction. Suppose there exists a valid pattern P_2 such that $P_2.O \supseteq P.O$. It is obvious that $P_2.T \subseteq P.T$. Based on Definition 2, the following conditions hold: (1) $P_2.T$ is G-connected. (2) $|P_2.T| \geq K$ and (3) $P_2.T$ is L-consecutive. Note that the entire $P_2.T$ is G-connected. Thus, $P_2.T$ itself is the only maximal G-connected subsequence. Based on conditions (1),(2),(3) and Definition 6, $P_2.T$ is decomposable. Then, based on Lemma 3, we know $\texttt{sim}(T) \neq \emptyset$ because $P_2.T \subseteq P.T$ and $P_2.T$ is decomposable. This leads to a contradiction with $\texttt{sim}(P.T) = \emptyset$. □

5.2.2 Apriori Enumerator

We design an Apriori based enumeration algorithm to efficiently discover all the valid patterns in a star partition. The principle of the Apriori algorithm is to construct a lattice structure and enumerate all the possible candidate sets in a bottom-up manner. Its merit lies in the monotonic property such that if a candidate set is not valid, then all its supersets can be pruned. Thus, it works well in practice in spite of the exponential search space.

Our customized Apriori Enumerator is presented in Algorithm 3. Initially, the edges (pairs of objects) in the star constitute the bottom level (Lines 2-4) and invalid candidates are excluded (Line 4). An indicator *level* is used to control the result size for candidate joins. During each iteration (Lines 6-19), only candidates with object size equals to *level* are generated (Line 8). When two candidates c_1 and c_2 are joined, the new candidate becomes $c' = \langle c_1.O \cup c_2.O, c_1.T \cap c_2.T\rangle$ (Line 9). To check the validity of the candidate, we calculate $\texttt{sim}(c'.T)$. If its simplified sequence is empty, c' is excluded from the next level (Line 10). This ensures that all the candidates with $P.O \supseteq c'.O$ are pruned. If a candidate cannot generate any new candidate, then it is directly reported (Lines 12-14). To further improve the performance, we adopt the idea of *forward closure* [19, 18] and aggressively check if the union of all the current candidates form a valid pattern (Lines 15-18). If yes, we can terminate the algorithm early and output the results.

Algorithm 3 Apriori Enumerator

```
Require: $Sr_s$
 1: $C \leftarrow \emptyset$
 2: for all edges $c = \langle o_i \cup o_j, T_{o_i} \cap T_{o_j}\rangle$ in $Sr_s$ do
 3:     if $\texttt{sim}(T_{o_i} \cap T_{o_j}) \neq \emptyset$ then
 4:         $C \leftarrow C \cup \{c\}$
 5: level $\leftarrow 2$
 6: while $C \neq \emptyset$ do
 7:     for all $c_1 \in C$ do
 8:         for all $c_2 \in C$ and $|c_2.O \cup c_2.O| =$ level do
 9:             $c' \leftarrow \langle c_1.O \cup c_2.O : (c_1.T \cap c_2.T)\rangle$
10:             if $\texttt{sim}(c'.T) \neq \emptyset$ then
11:                 $C' \leftarrow C' \cup \{c'\}$
12:         if no $c'$ is added to $C'$ then
13:             if $c_1$ is a valid pattern then
14:                 output $c_1$
15:     $O_u \leftarrow$ union of $c.O$ in $C$
16:     $T_u \leftarrow$ intersection of $c.T$ in $C$
17:     if $\langle O_u, T_u\rangle$ is a valid pattern then
18:         output $\langle O_u, T_u\rangle$, break
19:     $C \leftarrow C'; C' \leftarrow \emptyset;$ level $\leftarrow$ level $+ 1$
20: output $C$
```

Example 5. *As shown in Figure 5 (c), in the bottom level of the lattice structure, candidate $\langle 3, 6 : 3\rangle$ is pruned because its simplified sequence is empty. Thus, all the object sets containing $\langle 3, 6\rangle$ can be pruned. The remaining two candidates (i.e., $\langle 3, 4 : 1, 2, 3\rangle$ and $\langle 3, 5 : 2, 3\rangle$) derive a new $\langle 3, 4, 5 : 2, 3\rangle$ which is valid. By the forward closure checking, the algorithm can terminate and output $\langle 3, 4, 5 : 2, 3\rangle$ as the final pattern.*

5.3 Put Everything Together

We summarize the workflow of SPARE in Figure 5 as follows. After the parallel clustering in each snapshot, for ease of presentation, we use an aggregated graph G_A to capture the clustering relationship. However, in the implementation of the map phase, there is no need to create G_A in advance. Instead, we simply need to emit the edges within a star to the same reducer. Each reducer is an Apriori Enumerator. When receiving a star Sr_i, the reducer creates initial candidate patterns. Specifically, for each $o \in Sr_i$, a candidate pattern $\langle o, i : e(o, i)\rangle$ is created. Then it enumerates all the valid patterns from the candidate patterns. The pseudocode of SPARE is presented in Algorithm 4. In our implementation of SPARE on Spark [22], we take advantage of Spark features to achieve better workload balance. In particular, we utilize Spark DAG execution engine to inject a planning phase between map and reduce phases. By knowing all map results (i.e., star sizes), a simple best-fit strategy is adopted which assigns the most costly unallocated star to the most lightly loaded reducer, where the edges in a star are used as cost estimations. We also leverage Spark in-memory cache to avoid recomputing all stars after the planning phase.

Compared with TRPM, the SPARE framework does not rely on snapshot replication to guarantee correctness. In addition, we can show that the patterns derived from a star partition are unique and there would not be duplicate patterns mined from different star partitions.

Algorithm 4 Star Partitioning and ApRiori Enumerator

```
Require: list of $\langle t, S_t \rangle$ pairs
 1: —Map phase—
 2: for all $C \in S_t$ do
 3:     for all $o_1 \in C, o_2 \in C, o_1 < o_2$ do
 4:         emit a $\langle o_1, o_2, \{t\} \rangle$ triplet
 5: —Partition and Shuffle phase—
 6: for all $\langle o_1, o_2, \{t\} \rangle$ triplets do
 7:     group-by $o_1$, emit $\langle o_1, Sr_{o_1} \rangle$
 8: —Reduce phase—
 9: for all $\langle o, Sr_o \rangle$ do
10:     call Apriori Enumerator for star $Sr_o$
```

Theorem 5 (Pattern Uniqueness). *Let Sr_i and Sr_j ($i \neq j$) be two star partitions. Let P_i (resp. P_j) be the patterns discovered from Sr_i (resp. Sr_j). Then, $\forall p_i \in P_i, \forall p_j \in P_j$, we have $p_i.O \neq p_j.O$.*

Proof. We prove by contradiction. Suppose there exist $p_i \in P_i$ and $p_j \in P_j$ with the same object set. Note that the center vertex of the star is associated with the minimum id. Let o_i and o_j be the center vertexes of the two partitions and we have $o_i = o_j$. However, P_i and P_j are from different stars, meaning their center vertexes are different (i.e., $o_i \neq o_j$), leading to a contradiction. □

Theorem 5 implies that no mining efforts are wasted in discovering redundant patterns in the SPARE framework, which is superior to the TRPM baseline. Finally, we show the correctness of the SPARE framework.

Theorem 6. *The SPARE framework guarantees completeness and soundness.*

Proof. See Appendix B. □

6. EXPERIMENTAL STUDY

In this section, we evaluate the efficiency and scalability of our proposed parallel GCMP detectors on real trajectory datasets. All the experiments are carried out in a cluster with 12 nodes, each equipped with four quad-core 2.2GHz Intel processors, 32GB memory and Gigabit Ethernet.

Environment Setup: We use Yarn[2] to manage our cluster. We pick one machine as Yarn's master node, and for each of the remaining machines, we reserve one core and 2GB memory for Yarn processes. We deploy our GCMP detector on Apache Spark 1.5.2[3] with the remaining 11 nodes as the computing nodes. To fully utilize the computing resources, we configure each node to run five executors, each taking three cores and 5GB memory. In Spark, one of the 55 executors is taken as the Application Master for coordination, therefore our setting results in 54 executors. We set the number of partitions to be 486 to fully utilize the multi-threading feature of every core. All our implementations as well as cluster setups are publicly available[4].

Table 5: Statistics of datasets.

Attributes	Shopping	GeoLife	Taxi
# objects	13,183	18,670	15,054
# data points	41,052,242	54,594,696	296,075,837
# snapshots	16,931	10,699	44,364
# clusters	211,403	206,704	536,804
avg. cluster size	171	223	484

Datasets: We use three real trajectory datasets that are collected from different applications:

- Shopping[5]: The dataset contains trajectories of visitors in the ATC shopping center in Osaka. To better capture the indoor activities, the visitor locations are sampled every half second, resulting in 13,183 long trajectories.
- GeoLife[6]: The dataset essentially keeps all the travel records of 182 users for a period of over three years, including multiple kinds of transportation modes (walking, driving and taking public transportation). For each user, the GPS information is collected periodically and 91 percent of the trajectories are sampled every 1 to 5 seconds.
- Taxi[7]: The dataset tracks the trajectories of 15,054 taxies in Singapore. For each taxi, the GPS information are continually collected for one entire month with the sampling rate around 30 seconds.

Preprocessing: We replace timestamps with global sequences (starting from 1) for each dataset. We set a fixed sampling rate for each dataset (i.e., GeoLife = 5 seconds, Shopping=0.5 seconds, Taxi = 30 seconds) and use linear interpolation to fill missing values. For the clustering method, we use DBSCAN [5] and customize its two parameters ϵ (proximity threshold) and $minPt$ (the minimum number of points required to form a dense region). We set $\epsilon = 5$, $minPt = 10$ for GeoLife and Shopping datasets; and $\epsilon = 20$, $minPt = 10$ for Taxi dataset. After preprocessing, the statistics of the three datasets are listed in Table 5.

Table 6: Parameters and their default values.

Param.	Meaning	Values
M	min objects	5, 10, **15**, 20, 25
K	min duration	120, 150, **180**, 210, 240
L	min local duration	10, 20, **30**, 40,50
G	max gap	10, 15, **20**, 25, 30
O_r	ratio of objects	20%,40%,60%,80%,**100%**
T_r	ratio of snapshots	20%,40%,60%,80%,**100%**
N	number of machines	1, 3, 5, 7, 9, **11**

Parameters: To systematically study the performance of our algorithms, we conduct experiments on various parameter settings. The parameters to be evaluated are listed in Table 6, with default settings in bold.

[2]http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

[3]We have experimented with a query-based TRPM using Spark-SQL 2.0.0 window function. We find that Spark-SQL fails to execute the query-based TRPM in parallel, which results in a 120x performance slowdown compared to mapreduce-based TRPM. Thus we only report the performance of mapreduce-based TRPM in this paper.

[4]https://github.com/fanqi1909/TrajectoryMining/.

[5]http://www.irc.atr.jp/crest2010_HRI/ATC_dataset/

[6]http://research.microsoft.com/en-us/projects/geolife/

[7]Taxi is our proprietary dataset

6.1 Performance Evaluation

Varying M: Figures 7 (a),(g),(m) present the performance with increasing M. The SPARE framework demonstrates a clear superiority over the TRPM framework, with a performance gain by a factor of 2.7 times in Shopping, 3.1 times in GeoLife and 7 times in Taxi. As M increases, the running time of both frameworks slightly improve because the number of clusters in each snapshot drops, generating fewer valid candidates.

Varying K: The performance with increasing K is shown in Figures 7 (b),(h),(n). SPARE tends to run faster, whereas the performance of TRPM degrades dramatically. This is caused by the *sequence simplification* procedure in SPARE, which can prune many candidates with large K. However, the line sweep algorithm in TRPM does not utilize such property for pruning. It takes longer time because more replicated data has to be handled in each partition.

Varying L: Figures 7 (c),(i),(o) present the performances with increasing L. When $L = 10$, SPARE can outperform TRPM by around 10 times. We also observe that there is a significant performance improvement for TPRM when L increases from 10 to 20 and later the running time drops smoothly. This is because η is proportional to $O(K * G/L + L)$. When L is small (i.e., from 10 to 20), η decreases drastically. As L increases, η varies less significantly.

Varying G: Figures 7 (d),(j),(p) present the performances with increasing G. TRPM is rather sensitive to G. When G is relaxed to larger values, more valid patterns would be generated. TPRM has to set a higher replication factor and its running time degrades drastically when G increases from 20 to 30. In contrast, with much more effective pruning strategy, SPARE scales well with G. Particularly, SPARE is 14 times faster than TRPM when $G = 20$ in GeoLife dataset.

Varying O_r:Figures 7 (e),(k),(q) present the performances with increasing number of moving objects. Both TRPM and SPARE take longer time to find patterns in a larger database. We can see that the performance gap between SPARE and TRPM is widened as more objects are involved, which shows SPARE is more scalable.

Varying T_r: Figures 7 (f),(l),(r) present the performances with increasing number of snapshots. As T_r increases, SPARE scales much better than TRPM due to its effective pruning in the temporal dimension.

Resources: Table 7 lists the system resources taken by TRPM and SPARE under the default setting. Both TRPM and SPARE are resource efficient as they only occupy less than 20% of the available memory (i.e., 270GB) . Again, SPARE outperforms TRPM in both the execution time and the memory usage.

Table 7: Resources taken for TRPM and SPARE. Vcore-seconds is the aggregate of time spent in each core. Memory is the actual size (in MB) of RDDs.

Dataset	**Method**	**Vcore-seconds**	**Memory**
Shopping	TRPM	90,859	10,019
	SPARE	33,638	8,613
Geolife	TRPM	106,428	18,454
	SPARE	35,343	14,369
Taxi	TRPM	503,460	51,691
	SPARE	68,580	35,912

6.2 Analysis of SPARE framework

In this part, we extensively evaluate SPARE from three aspects: (1) the advantages brought by the sequence simplification, (2) the effectiveness of load balance, and (3) the scalability with increasing computing resources.

6.2.1 *Power of sequence simplification*

To study the power of *Sequence Simplification* (SS), we collect two types of statistics: (1) the number of pairs that are shuffled to the reducers and (2) the number of pairs that are fed to the Apirori Enumerator. Their difference is the number of size-2 candidates pruned by SS. The results in Table 8 show that SS is very powerful and eliminates nearly 90 percent of the object pairs, which significantly reduces the overhead of the Apriori enumerator. In fact, without SS Apriori cannot finish in five hours.

Table 8: Pruning power of SPARE.

Dataset	**Shopping**	**GeoLife**	**Taxi**
Before pruning	878,309	1,134,228	2,210,101
After pruning	76,672	123,410	270,921
Prune ratio	91.2%	89.1%	87.7%

6.2.2 *Load balance*

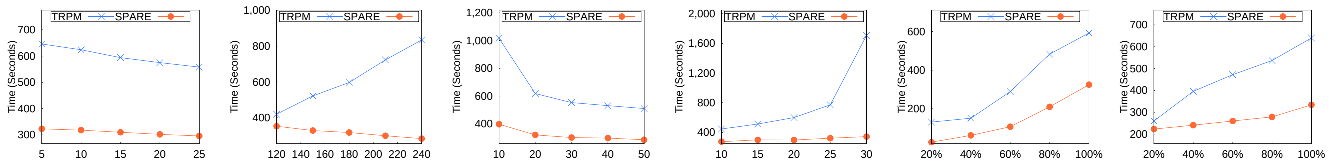
To study the effect of load balance in the SPARE framework, we use random task allocation (the default setting of Spark) as a baseline, denoted by SPARE-RD, and compare it with our best-fit method. In best-fit, the largest unassigned star is allocated to the currently most lightly loaded reducer. Figure 8 shows the breakdown of the costs in the mapreduce stages for SPARE and SPARE-RD. We observe that the map and shuffle time of SPARE and SPARE-RD are identical. The difference is that SPARE incurs an additional overhead to generate an allocation plan for load balance (around 4% of the total cost), resulting in significant savings in the reduce stage (around 20% of the total cost). Meanwhile, both SPARE and SPARE-RD outperform TRPM in each phase. This shows the efficiency of the star partition and apriori enumeration. We also report the cost of the longest job (Max) and the standard deviation (Std. Dev.) for all jobs in Table 9, whose results clearly verify the effectiveness of our allocation strategy for load balance.

Table 9: Statistics of execution time (seconds) on all jobs.

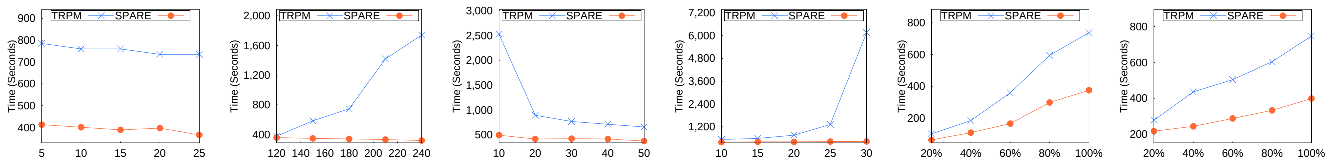
Dataset	**SPARE-RD**		**SPARE**	
	Max	Std. Dev.	Max	Std. Dev.
Shopping	295	41	237	21
GeoLife	484	108	341	56
Taxi	681	147	580	96

6.2.3 *Scalability*

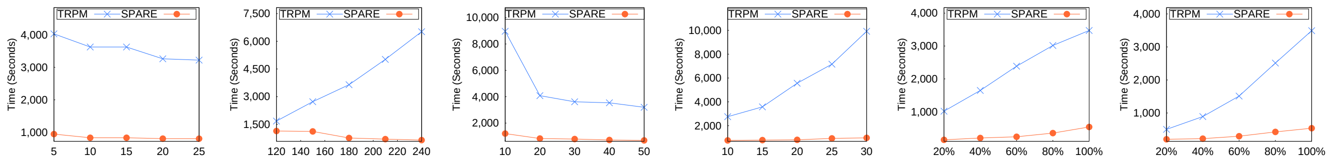
When examining SPARE with increasing computing resources (number of machines), we also compare SPARE with the state-of-the-art solutions for *swarm* and *platoon* in the single-node setting. Since the original *swarm* and *platoon* detectors cannot handle very large-scale datasets, we only use 60% of each dataset for evaluation. For a fair comparisons, we customize two variants of SPARE to mine *swarms*



(a) Shopping vary M (b) Shopping vary K (c) Shopping vary L (d) Shopping vary G (e) Shopping vary O_r (f) Shopping vary T_r



(g) GeoLife vary M (h) GeoLife vary K (i) GeoLife vary L (j) GeoLife vary G (k) GeoLife vary O_r (l) GeoLife vary T_r



(m) Taxi vary M (n) Taxi vary K (o) Taxi vary L (p) Taxi vary G (q) Taxi vary O_r (r) Taxi vary T_r

Figure 7: Performance of SPARE and TRPM on real datasets under different pattern parameters.

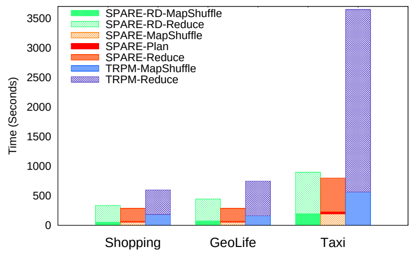
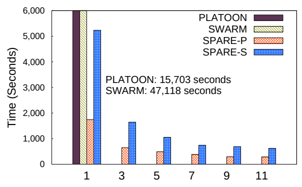
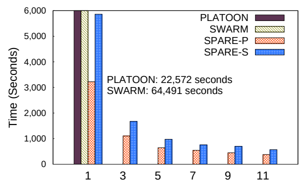


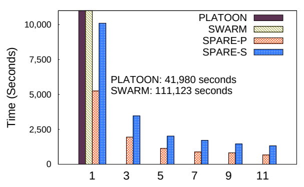
Figure 8: Cost breakdown of TRPM, SPARE-RD and SPARE.



(a) Shopping vary N



(b) GeoLife vary N



(c) Taxi vary N

Figure 9: Comparisons among TRMP, SPARE, PLATOON and SWARM.

and *platoons*, which are denoted as SPARE-S and SPARE-P respectively. The customization is according to the settings in Table 3 and the results are reported in Figure 9. First, the centralized schemes are not suitable to discover patterns in large-scale trajectory databases. It takes nearly 30 hours to detect *swarms* and 11 hours to detect *platoons* in the Taxi dataset in a single machine. In contrast, when utilizing the multi-core (i.e., a single node with four executors) environment, SPARE-P achieves 7 times speedup and SPARE-S achieves 10 times speedup. Second, we see that SPARE schemes demonstrate promising scalability in terms of the number of machines available. The running times decrease almost inversely as more machines are used. When all the 11 nodes (162 cores) are available, SPARE-P is upto 65 times and SPARE-S is upto 112 times better than the state-of-the-art centralized schemes.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a generalized co-movement pattern to unify those proposed in the past literature. We also devised two types of parallel frameworks in Spark that can scale to support pattern detection in trajectory databases with hundreds of millions of points. The efficiency and scalability were verified by extensive experiments on three real datasets. In the future, we intend to examine co-movement pattern detection in streaming data for real-time monitoring. We are also interested in extending the current parallel frameworks to support other types of advanced patterns.

Acknowledgment: The authors would like to thank the anonymous reviewers for their responsible feedback.

8. REFERENCES

[1] R. Agrawal, R. Srikant, et al. Fast algorithms for mining association rules. In *VLDB*, pages 487–499, 1994.

[2] H. H. Aung and K.-L. Tan. Discovery of evolving convoys. In *SSDM*, pages 196–213, 2010.

[3] J. Bao, Y. Zheng, D. Wilkie, and M. F. Mokbel. A survey on recommendations in location based social networks. In *Geoinformatica*, pages 525–543, 2015.

[4] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. In *Cartographica*, pages 112–122, 1973.

[5] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *SIGKDD*, pages 226–231, 1996.

[6] J. Gudmundsson and M. van Kreveld. Computing longest duration flocks in trajectory data. In *GIS*, pages 35–42, 2006.

[7] L. Guo, D. Zhang, G. Cong, W. Wu, and K.-L. Tan. Influence maximization in trajectory databases. In *TKDE*, page 1, 2016.

[8] H. Jeung, M. L. Yiu, X. Zhou, C. S. Jensen, and H. T. Shen. Discovery of convoys in trajectory databases. In *VLDB*, pages 1068–1080, 2008.

[9] R. Jinno, K. Seki, and K. Uehara. Parallel distributed trajectory pattern mining using mapreduce. In *CloudCom*, pages 269–273, 2012.

[10] P. Kalnis, N. Mamoulis, and S. Bakiras. On discovering moving clusters in spatio-temporal data. In *SSTD*, pages 364–381, 2005.

[11] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia. Skewtune: mitigating skew in mapreduce applications. In *SIGMOD*, pages 25–36, 2012.

[12] P. Laube, M. van Kreveld, and S. Imfeld. Finding remodetecting relative motion patterns in geospatial lifelines. In *DSDH*, pages 201–215. 2005.

[13] X. Li, V. Ceikute, C. S. Jensen, and K.-L. Tan. Effective online group discovery in trajectory databases. In *TKDE*, pages 2752–2766, 2013.

[14] X. Li, V. Ceikute, S. Jensen, Christian, and K.-L. Tan. Effective online group discovery in trajectory databases. 2013.

[15] Y. Li, J. Bailey, and L. Kulik. Efficient mining of platoon patterns in trajectory databases. In *DKE*, pages 167–187, 2015.

[16] Z. Li, B. Ding, J. Han, and R. Kays. Swarm: Mining relaxed temporal moving object clusters. In *VLDB*, pages 723–734, 2010.

[17] Z. Li, B. Ding, J. Han, R. Kays, and P. Nye. Mining periodic behaviors for moving objects. In *SIGKDD*, pages 1099–1108, 2010.

[18] J. Pei, J. Han, R. Mao, et al. Closet: An efficient algorithm for mining frequent closed itemsets. In *DMKD*, pages 21–30, 2000.

[19] J. Wang, J. Han, and J. Pei. Closet+: Searching for the best strategies for mining frequent closed itemsets. In *SIGKDD*, pages 236–245, 2003.

[20] Y. Wang, E.-P. Lim, and S.-Y. Hwang. Efficient mining of group patterns from user movement data. In *DKE*, pages 240–282, 2006.

[21] J. S. Yoo and S. Shekhar. A joinless approach for mining spatial colocation patterns. In *TKDE*, pages 1323–1337, 2006.

[22] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *USENIX*, pages 15–28, 2012.

[23] K. Zheng, Y. Zheng, N. J. Yuan, and S. Shang. On discovery of gathering patterns from trajectories. In *ICDE*, pages 242–253, 2013.

[24] Y. Zheng. Trajectory data mining: an overview. In *TIST*, pages 1–41, 2015.

[25] Y. Zheng, Y. Liu, J. Yuan, and X. Xie. Urban computing with taxicabs. In *UbiComp*, pages 89–98, 2011.

APPENDIX

A. PROOFS OF THEOREM 2 AND 3

Proof. Γ can be formalized using linear algebra: Let J be the adjacent matrix of G_A. A vertex order on G_A can be represented as PJP^T where P is a *permutation matrix* [8]. Consider an assignment matrix B of star partitioning (i.e., $b_{i,j} = 1$ if vertex j is in star Sr_i), $B = \text{triu}(PJP^T)$ [9] holds. Let vector $\vec{b}$ be the *one*[10] vector with size n. Let $\vec{c} = B\vec{b}$, then each c_i denotes the number of edges in star Sr_i. Thus, Γ can be represented as the infinity norm of $B\vec{b}$. Let Γ^* be the minimum Γ among all vertex orderings, that is:

$$\Gamma^* = \min_{P \in \mathbb{P}} ||B\vec{b}||_\infty \text{ ,where } ||B\vec{b}||_\infty = \max_{1 \le j \le n} (c_j) \tag{1}$$

Let B^* be the optimal assignment matrix. It follows that $\Gamma^* = ||B^*\vec{b}||_\infty \ge d/2$. Next, let $e_{i,j}$ be an entry in PJP^T, $e_{i,j}$s are independent. Further, $E[\Sigma_{1\le j \le n} e_{i,j}] = d$ where d is the average degree of G_A. On the other hand, $b_{i,j} = e_{i,j}$ for $i > j$. Since $i > j$ and $e_{i,j}$s are independent. $E[b_{i,j}] = E[e_{i,j}]E[i > j] = E[e_{i,j}]/2$. By linearity of expectations, we get: $E[c_i] = E[\Sigma_{1\le j\le n} b_{i,j}] = E[\Sigma_{1\le j\le n} e_{i,j}]/2 = d/2$. Let $\mu = E[c_i] = d/2$, $t = \sqrt{n \log n}$, by Hoeffding Bound, the following holds:

$$Pr(c_i \ge \mu + t) \le \exp(\frac{-2t^2}{n}) = \exp(-2\log n) = n^{-2}$$

Next, the event $(\max_{1\le j\le n}(c_j) \ge \mu + t)$ can be viewed as $\cup_{c_i}(c_i \ge \mu + t)$, by Union Bound, the following holds:

$$Pr(\Gamma \ge \mu + t) = Pr(\max_{1\le j\le n}(c_j) \ge \mu + t) = Pr(\cup_{c_i}(c_i \ge \mu + t))$$
$$\le \Sigma_{1\le i\le n} Pr(c_i \ge \mu + t) = n^{-1} = 1/n$$

This indicates the probability of $(\Gamma - d/2) \le O(\sqrt{n \log n})$ is $(1 - 1/n)$. Since $\Gamma^* \ge d/2$, Theorem 2 holds. When the aggregated graph is *dense* (i.e., $d \ge \sqrt{12 \log n}$), the Chernoff Bound can be used to derive a tighter bound of $O(\sqrt{d \log n})$ following a similar reasoning. □

B. PROOF OF THEOREM 6

Proof. For soundness, let P be a pattern enumerated by SPARE. For any two objects $o_1, o_2 \in P.O$, the edge $e(o_1, o_2)$ is a superset of $P.T$. As $P.T$ is a valid sequence, by the definition of GCMP, P is a true pattern. For completeness, let P be a true pattern. Let s be the object with the smallest ID in $P.O$. We prove that P must be outputted by Algorithm 3 from Sr_s. First, based on the definition of star, every object in $P.O$ appears in Sr_s. Since $P.T$ is decomposable, then by Lemma 3 the time sequence of any subset would not be eliminated by any `sim` operations. Next, we prove at every iteration *level* $\le |P.O|$, $P.O \subset O_u$, where O_u is the forward closure. We prove by induction. *level* $= 2$ trivially holds. If $P.O \subset O_u$ at *level* i, then any subsets of $P.O$ with size i are in the candidate set. This suggests that no subsets are removed by Lines 12-20. Then, $P.O \subset U_{i+1}$ holds. Since $P.O$ does not pruned by simplification, monotonicity and forward closure, P must be returned by SPARE. □

[8]An identity matrix with rows shuffled

[9]`triu` is the upper triangle part of a matrix

[10]Every element in $\vec{b}$ is 1