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].

[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.

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.

[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