From Trajectories to Activities: A Spatio-Temporal Join Approach [*]

Kexin Xie
School of ITEE
the University of Queensland
Brisbane, Australia
kexin@itee.uq.edu.au

Ke Deng
School of ITEE
the University of Queensland
Brisbane, Australia
dengke@itee.uq.edu.au

Xiaofang Zhou
School of ITEE
the University of Queensland
Brisbane, Australia
zxf@itee.uq.edu.au

ABSTRACT

People's activity sequences such as eat at a restaurant after 2 hours of shopping, contains rich semantic information. This information can be explored for a broad range of applications and services. However, it is impractical to ask a large number of people to record their daily activities. As the increasing popularity of GPS-enabled mobile devices, a huge amount of trajectories which show people's movement behaviors have been acquiring. The natural link between activities and traveling motivates us to investigate a novel approach to automatically extract sequences of activities from large set of trajectory data. Intuitively, activities can only happen when trajectory is geographically near for a proper period of time for these activities, such as 30 minutes for dining in a restaurant. In this work, the concepts influence and influence duration are proposed to capture the intuition. We also propose two algorithms to join large set of trajectories with activities with duplication reuse techniques. We conduct comprehensive empirical studies to evaluate the two algorithms with synthetic data set generated from real world POIs and road networks.

Categories and Subject Descriptors

H.2.8 [**Database Applications:**]: Spatial databases and GIS

General Terms

Algorithms

Keywords

Spatial Databases, Spatial Join, Trajectories

1. INTRODUCTION

Everyday, people travel to different places for different activities. These activity sequences may include but not limited to "drinking in a bar after work", "eating at a restaurant after 2 hours of shopping", "walking in the park 30 minutes after dinner". These sequences of activities can be used by applying data mining techniques to allow innovative and useful applications and services. In general, those knowledge mined by various data mining techniques can be very useful in social network applications, targeted advertising, public security, town planning, and other related applications and services. So when the user moves to a new place, software can suggest similar places that the user may like to go.

Applying those data mining techniques requires sequential data. However, obtaining everyday activity sequences for large amount of users is not easy. It is impractical to ask all users for their daily activities. Thanks to the rapid development of location acquisition technologies (e.g. GPS enabled mobile devices), large set of spatio-temporal datasets can be collected. GPS points along with timestamps shows user's movement behaviour, called user's *trajectory*. The set of trajectories can be expanded to find user activities.

To find activity sequences based on trajectories, one must associate trajectories with the background geographic information, on which the objects are moving. This is because the geographical information carries much richer semantics about different activities. There are many types of background knowledge in geographical data. One of such background information about user activities are *point of interests*. A *Point of Interest* (POI) is a specific location that someone may find useful or interesting, such as restaurants, parks, and shopping centers. Most activities are carried on at some POIs. e.g. dining at restaurant, fuel the car at petrol stations, parking at car parks, exercising at gyms. We call such activities *user events*. In this paper, we study the *trajectory semantic join* problem, to find user activity sequences from a set of trajectories, which can then be used to apply data mining techniques, or other applications.

To associate trajectories with activities in different POIs is not easy. First of all, POI cannot be associated to trajectory by looking at their distances to the trajectory only. Consider the example shown in figure 1, a student studies at a university for 8 hours, then drives to a supermarket doing shopping for 45 minutes, at last went to a restaurant for diner for an hour. As shown in figure 1, both Petrol Shop, and Fast Food are very close to the trajectory, however, there is no activities carried on at those POIs. Second, user does not necessarily stop when activities carried on, e.g. walking in a park. Third, different activities have different

[*]This work is partially supported by Microsoft Research Asia Internet Services Theme Research Program.

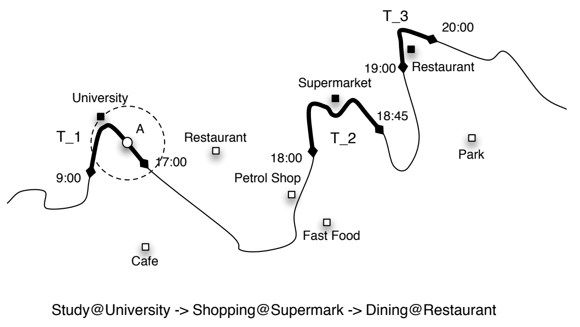


Figure 1: Trajectory and The User Event Sequence

durations, eating at fast food outlet may only take 15 minutes, while dining at a restaurant will took about an hour; At last, the activities carried on at the same POI may be different, e.g. dining at restaurant or working in the restaurant. To the best of our knowledge, there is little work that consider both trajectories and the background geographical information for user activities.

Recent works on querying and mining moving objects [6, 11, 15, 17] focused on trajectories only. [11] and [15] queries the database for people that travels together for a long time. [6] find location travel patterns for a set of trajectories, by dynamic discover regions of interests which are geographical areas that bypassed by many trajectories. These regions of interests are popular areas which intersects with many trajectories such as busy motorways, which do not necessarily have any user activities. Li et al. in [17] find similar trajectories with similar shapes and locations, they proposed a fixed disk and time based approach to replace trajectories with a sequence of feature points called stay points. These stay points have nothing to do with the background POIs with various activity duration times. Our work is similar to [1] in the sense of associate trajectories with semantic information. However, there are significant differences, [1] introduced a model with fixed stay length at predefined polygons for candidate stops. We propose method to associated activities happened at point data set (i.e. POIs), and conducted experiments with performance analysis.

Intuitively, events can only happen when trajectory is geographically near a POI for long period of time. To solve the problems mentioned above, we make use of two measures. One is the distance between the POI and the trajectory to determine where user activities may happen; The other measure is the duration that determines which event may happen. We introduce the concept of *influence* and *influence duration* for associations among trajectories, POIs, and activities. *Influence* is a distance based measure, such that a trajectory T can only be associated with a POI p if there exists at least one point on T that is influenced by p. *Influence duration* is a temporal based measure to determine the duration of a trajectory influenced by a POI. We also introduced the concept of *POI-Activity Mapping Set* (PAMS), which provide the rules to map between different user activities and POIs based on various influence durations.

The rest of the paper is organized as follows: In section 2, we discuss previous works on trajectory pattern mining, spatial joins, and continuous nearest neighbor query. We propose measures to associate trajectories with user activities and formally define this problem in section 3. We introduce the basic method by modify an existing spatial join technique in section 4. In section 5, we propose optimizations that reuse duplicated entries to reduce the I/O cost and hence increase the efficiency of the algorithm. Section 6 reports the experimental performance comparisons. At last, we conclude in section 7.

2. RELATED WORK

In this section, we summarize some relevant research works related to the topic of this paper. First we briefly cover research works related to trajectory pattern mining, then we cover the studies on spatial joins and continuous nearest neighbor (CNN) queries.

2.1 Trajectory Pattern Mining

Mining trajectory patterns from spatio-temporal database, such as GPS logs or other location data sets, has attracted many attentions recently. Because of the continuous context of spatio-temporal sequences, such as the sequence of positions of a moving object. The traditional notion of frequent sequential patterns from transactional database pattern mining cannot usually be applied directly. Usually some kind of feature extraction and some kind of tolerance to small perturbation is needed. [4] considers patterns that are in the form of trajectory segments and searches approximate instances in the data. Kalnis et al. provides clustering-based perspective, and propose the notion of a *moving cluster* [12], which is a sequence of spatial clusters appearing during consecutive time points, such that the portion of common objects in any two consecutive clusters is not below a given threshold. Jeung et al. in [10, 11] introduces a new concept of *convoy queries* that discovers the group of objects, which travels together for a consecutive amount of time.

Li et al. in [17] proposed a framework to compute similarities of different trajectories with stay point detection and hierarchical-based measures. The stay point of a trajectory is computed with a fixed disk based algorithm which is a sequence of points that travels within a fixed disk within a certain amount of time. [1] introduces the concept of trajectory semantic, and models the trajectories as a series of stops and moves. [1], claims that one can use different size disks, which they call *application* blocks, to find those trajectory stops. A fixed global time constraint is used to extract those stop part of trajectories. A simple nested loop approach is proposed in [1], which is not suitable with large size of spatial data sets.

In [6], Giannotti et al. defines trajectory pattern as a sequence of *Region of Interests (ROIs)* with temporal annotations between consecutive ROIs from a trajectory database. ROIs can be static assigned or dynamic computed. The dynamically discovered ROIs are highly likely to be regions bypassed by many trajectories, e.g. busy motorways, which have little to do with people activities. The measure of determine ROI is primarily determined by number of visit.

2.2 Spatial Joins and CNN Queries

Given two sets of multi-dimensional objects in Euclidean space, *spatial join* is the operation to find all pairs of objects satisfying a given relation between the objects that involves the values of their spatial components, such as intersection [9]. The problem of perform spatial joins has been studied a lot in previous works. If both data set are indexed with hierarchical tree structure such as R-trees, [7, 3, 13, 8] proposed different method based on the *synchronized traversal* techniques, which traverse both indexes and compares their intermediate nodes with the on the same depth, if their intermediate node do not overlap, then their children node will not be examined in the later level comparisons.

In the case that neither of the data sets are indexed, Patel and Dewitt [19] proposed an algorithm called PBSM, which partitions the whole space using a uniform grid, which they call *tiles*. Each grid contains the spatial objects that are overlap with it. To overcome the issue of data screw, the grid cells are grouped into partitioning using a mapping function. The data is not physically partitioned into the grid cells, but only into the final partitions. Zhou et al. [24] uses a variation of this technique in which the data is physically partitioned into the grid cells. For partition based spatial joins, the more partitions there are, the greater the replication. However, the internal memory refinement techniques normally perform better with smaller partitions, the choice of number of partitions is unclear in the literature, and is still an open question. Koudas and Sevcik partition [14] introduced S^3J, a method without data replication by exploiting a variant of quad-trees. [5] modifies both PBSM and S^3J improves run-time costs by dealing with the impact of redundancy and duplicate detection on the performance. Our proposed algorithm also apply optimizations on duplicate entries as in [5]. The difference between our proposed optimization is that we reduce the I/O cost so that duplicated entries are not retrived from the hard disk, [5] focused eliminate the duplications when reporting the result. In fact, our algorithms can be used in conjunction with optimizations proposed in [5] to optimize the algorithm. Another approach called Scalable Sweeping-Based Join (SSSJ) [2] partition the data into strips so that they can take advantage of their modification to the plane-sweep algorithm. SSSJ is very efficient and is worst-case optimal on both I/O and CPU cost. A serious deficiency of SSSJ is when it is used in a DBMS, it requires both input relations to be sorted first before producing the first output tuple.

Continuous Nearest Neighbor (CNN) search has also gained much attention in the literature since it is first introduced in [20], which addressed this problem from the modeling and query language perspectives. Let $\mathcal{P}$ be a dataset of points in multi-dimensional space. A CNN query retrieves the nearest neighbor (NN) of every point in a line segment $q = [s, e]$. [21] proposed an algorithm using sampling techniques to compute the result, at each evaluation time, the query may get benefit from the previous result of the last evaluation. [22] uses R-tree based pruning strategies to reduce the data point to be accessed. Later works [18, 23] works on monitoring the CNN objects. In our work, we also used CNN as part of the joining conditions. However, we concentrate on the temporal duration of the CNN rather than the spatial attributes.

3. TRAJECTORIES AND SEMANTICS JOIN

The three key objects in our study are trajectories, POIs, and activities. Trajectories and POIs together shows where the user went, the time duration at user spent at each POI shows which activity took place at the POI. A trajectory is a sequence of time-stamped locations, representing the traces collected by some infrastructures, such as GPS traces recorded by mobile devices and submitted to a central server. The latitude-longitude pair from the GPS logs is then abstracted using ordinary Cartesian coordinates. In this paper, We model a *trajectory* as a *polyline*, which is a sequence of connected line segments defined by the GPS locations with each location annotated temporally. The formal definition of a trajectory is defined below:

Definition 1. (Trajectory) A *Trajectory* L is a poly-line that specified by a sequence of n timestamped points $\langle(v_0, t_0), (v_1, t_1), \ldots, (v_n, t_n)\rangle$, where $v_i (0 \le i \le n)$ is a point in 2-dimensional space, t_i is the timestamp of v_i, and $\forall_{0 \le i \le n, 0 \le j \le n}\ i < j \implies t_i < t_j$.

For each trajectory, the timestamps of every point on each line segment of the trajectory can be determined by a function $timestamp(v, T)$ given a spatial point v, and a trajectory T. The function will return the timestamp of v denoted as t_v according to v's location on the trajectory. If v is not on the trajectory, then the function will return -1.

$$timestamp(v, T) = \begin{cases} t_v, & \exists_{l \in T}.v \in l; \\ -1, & \text{otherwise.} \end{cases} \qquad (1)$$

The timestamp of a point *on* trajectory can be estimated by many ways depend on how the traveling velocity and acceleration is calculated for each line segment of the trajectory. In this study, we assume user moves in a constant speed between two end points on a trajectory line segment. The timestamp function can also be used to determine a timestamped point (v_k, t_k) is on a trajectory or not. If $timestamp(v_k, T) = t_k$, then (v_k, t_k) is *on* T denoted as $(v_k, t_k) \in T$, otherwise (v_k, t_k) is not on T.

A *subtrajectory* of a trajectory T is a segment of T. In figure 1, segment T_1, T_2, T_3 are subtrajectories of the trajectory. Subtrajectory is formally defined below:

Definition 2. (Subtrajectory) A trajectory of m vertices $T_s = \langle(v_{s_0}, t_{s_0}), \ldots, (v_{s_m}, t_{s_m})\rangle$ is a subtrajectory of a trajectory of n vertices $T = \langle(v_0, t_0), \ldots, (v_n, t_n)\rangle$ denoted as $T_s \sqsubseteq T$, if and only if $\forall_{v_k \in T_k}.v_k \in T$.

Trajectory itself does not carry any semantic meanings unless associated with geographical meta data. These meta data include the locations of businesses, offices, residential buildings, shops, university campuses and many other. These places are called *Point of Interests* (POIs). POIs are application dependent.

To join trajectories with user activities, the first step is to join trajectories with those POIs that user activities took place. To join line segments with points data such as POIs, we need a measure to associate both spatial objects together. POIs are an abstract point which gives an approximate location of some place with arbitrary shapes. e.g. shopping centers are the buildings of the shops and the surrounding car parks. Because of the nature of POIs, it does not make sense to use simple spatial measures such as intersections. We introduce a measure that associated those points with trajectories called *influence*. Influence uses the distance based measure that a point v on a trajectory is said to be influenced by a POI p, if the distance between v and p is smaller than any other POIs i.e. p is v's nearest neighbor. As illustrated in figure 1, point A on the trajectory is closer to University than any other POIs, hence it is less likely that this person at point A goes to other POIs but University.

We say in this situation that point A is *influenced* by POI University, same as other points on subtrajectory T_1.

To know which subtrajectory is influenced by which POI is not enough to identify user activities. To be able to find whether or which activity happened at a given POI influenced subtrajectory, we introduced a measure called *influence duration*. *Influence duration* is the total consecutive time a subtrajectory is influenced by some POI. As shown in figure 1, the influence duration of subtrajectory T_1 influenced by University is 8 hours. Influence duration is mainly used as the bridge between trajectory and different events through POIs. Now we formally define our measure to join trajectories with POIs called *influences* and *influence durations* to join with activities.

Definition 3. (Influence and influence duration) Given a set P of POIs, a sub-trajectory T_s of trajectory T is *influenced* by a POI $p \in P$ denoted as $influenced(T_s) = p$ if $\forall_{(v_k,t_k)\in T_s}.(\neg\exists_{p_i\in P}.dist(v_k,p_i) < dist(v_k,p))$, T_s is called p's influenced segment over T denoted as $T_s \in influence(p,T)$ if $influenced(T_s) = p$ and $\neg\exists_{T_m\sqsubseteq T} influenced(T_m) = p \wedge T_s \sqsubseteq T_m$. The *influence duration* of p over T_s denoted as $idr(p,T_s)$is the total time duration of T_s, given $T_s \in influence(p,T)$.

The next step after trajectories are joined with POIs is to find the activities took place for each POI and its influenced subtrajectories. It is very common that one POI may carry different activities for different users. For example, user may go to a restaurant to have dinner, or the user is the chef working in the kitchen. One of the differences between those different user activities at the same POI, is their elapsed time. Those different activity elapsed time can be compared with the influence durations to find which activity happened. In order to find the mappings between trajectories and user activities, we introduce a so-called *POI-Activity Mapping Set* (PAMS), which shows the elapsed time ranges for various activities at different POIs. These elapsed time ranges can then be compared with the influence durations to find possible activities took place.

Having a set $\mathcal{P}$ of POIs, a set $\mathcal{E}$ of user activities. A *POI-Activity Mapping Set* (PAMS) is a set of quadruple $M = \{(p, e, t_{min}, t_{max})\}$, where $p \in \mathcal{P}$, $e \in \mathcal{E}$, and t_{min} and t_{max} are the minimum and maximum elapsed time for user activity e happened at p. The set of possible events $\mathcal{E}_c$ that could happen given at POI p, given the influence duration d and an PAMS M, is determined by an event assignment function $\mathcal{E}_c = EA(p,d,M)$.

Figure 2 shows part of the PAMS with the different POIs carry different user activities with different elapsed time. Take the POI restaurant as an example, depending on the influence duration of the POI restaurant on the users trajectory, different activities may took place. If this person stays at restaurant for 20 minutes, the activity took place is "delivery", if the person stays for 4 hours, then the activity is "dinning", and if the person stays for 7 hours, then the activity is working. The PAMS allows us to determine the user activity took place given an POI and the influence duration over the trajectory. More discussions on how activities are assigned to subtrajectory-POI pairs is discussed in section 4.2. Here we assume the content of PAMS is provided by some domain experts and is out of the scope of this paper.

PAMS here provide the mappings between POIs and user activities, the influence and influence duration measure can be used with PAMS to determine the activities that might happen at subtrajectory-POI pairs. Now we formally define the *trajectory semantic join* problem as follows:

PAMS			
…			
…			
…			
Restaurant	Delivery	15 minutes	45 minutes
Restaurant	Dinning	30 minutes	2 hours
Restaurant	Working	3 hours	8 hours
University	Studying	2 hours	12 hours
University	Working	4 hours	12 hours
Petrol Shop	Fueling	10 minutes	30 minutes
Supermarket	Shopping	30 minutes	2 hours
Fast Food	Takeaway	10 minutes	45 minutes
Park	Playing	15 minutes	1 hour
…			
…			
…			

Figure 2: An PAMS Example

Definition 4. (Trajectory Semantic Join) Given a set $\mathcal{ST}$ of trajectories, a set $\mathcal{P}$ of POIs, and a PAMS M. *Trajectory semantic join* find all subtrajectory-activity set pairs (T_s, e), such that $\exists_{T\in\mathcal{ST}}.T_s \in T$, and $\exists_{p\in P}.E = EA(p, idr(p,T_s), M)$.

For example, given a set of trajectories that contains only one trajectory shown in figure 1, and a set of POI shown in figure 1, and a PAMS shown in figure 2. Trajectory semantic join returns the following result:

$$\{(T\_1, (Studying@University \vee Working@University)),$$

$$(T\_2, Shopping@Supermarket), (T\_3, Dining@Restaurant)\}$$

4. PARTITION BASED APPROACH

Because the trajectory and user activities are indirectly connected through POIs, the trajectory and semantic join consists of two phases. The first phase is to join trajectories with POIs using the influence measure to find the subtrajectory and POI pairs. The second phase is to assign user activities to those subtrajectory and POI pairs according to their influence durations. Note that many subtrajectory and POI pairs may be discarded because there is no activity to be associated to. In this section, we first present how to find subtrajectory and POI pairs using Voronoi diagram in subsection 4.1, then we discuss how activities can be assigned to those subtrajectory-POI pairs in subsection 4.2. At last in subsection 4.3, we introduce a basic algorithm called *Partition based Trajectory and Semantic Join* (PTSJ).

4.1 Join Trajectories with POIs

A *Voronoi diagram* of a set of points $\mathcal{P}$ is the subdivision of the plain into n cells, one for each site in $\mathcal{P}$, with the property that a point q lies in the cell corresponding to a site p_i if and only if $dist(q,p_i) < dist(q,p_j)$ for each $p_j \in \mathcal{P}$, with $j \neq i$. The Voronoi diagram suits in our definition of influence well in this case. If we treat each POI as a Voronoi site and construct a Voronoi diagram, each *Voronoi cell* forms the *influence region* of each POI, if the trajectory is within such Voronoi cell formed influence region of POI p, then then it is influenced by p. With the help of Voronoi diagram, the first phase of trajectory and semantic join is to compute the subtrajectories for each trajectory that are completely within the Voronoi cells, and the influence duration is simply the time duration of the subtrajectory. The result is a set of triples (T_s, p, d), where T_s is the subtrajectory, p is the POI that fully influences T_s, and d is the influence duration. We call this triple as *TPT*.

```
Algorithm 1: Trajectory_POI_Join
  Input: A set of trajectories $ST$, a set of POIs $P$, and its
         Voronoi diagram $V$.
  Output: A set of triples (subtrajectory, POI, actual
          influence duration)
  /* Step 1 - Compute line intersections                    */
1 Compute line intersection points of all trajectories in $ST$
  and all edges in $V$ using plain-sweep technique;
  /* Step 2 - Organize and Output                           */
2 Group the intersection points by trajectory and sort them
  by their timestamp;
3 foreach $T$ in $ST$ do
4     foreach consecutive intersection points $p_i$ and $p_j$ in $T$
      do
5         Find the subtrajectory $T_s$ of $T$ defined enclosed by
          $p_i$ and $p_j$;
6         Find common POI $p$ that influence both $p_i$ and $p_j$;
7         Find the time difference $d$ of $p_i$ and $p_j$;
8         Output the triple $(T_s, p, d)$;
```

As illustrated in algorithm 1, the trajectory and POI join consists of two steps. Step 1 computes the intersections of trajectories and all edges in the Voronoi diagram of POI, the intersection points can be computed efficiently using plain-sweep technique. The next step organizes the those intersection points to find corresponding subtrajectory, POI and the influence duration. Since we are not interested in how the trajectory looks like within the influence region, but only the influence duration. It is sufficient to compute only the intersection points of trajectories and Voronoi edges, which shows when trajectory enters and exists Voronoi cells.

Voronoi diagram have the property that every edge within the Voronoi diagram is shared by exactly two Voronoi sites. This means that every point on the Voronoi edge is influenced by exactly two POIs.

Observation 1. *Given a trajectory intersects Voronoi edges on a sequence of points $\langle p_0, p_1, \ldots, p_n\rangle$, where $\forall_{0\leq i<n}.timestamp(p_i, T) \leq timestamp(p_{i+1}, T)$. All consecutive points p_i and $p_{i+1}(0 \leq i < n)$ have at least one common influenced POI P_c, and the subtrajectory T_i of T, defined by p_i and p_{i+1} is influenced by P_c.*

Observation 1 ensures temporally sorted intersection points encloses subtrajectories are fully influenced by some POI. As a result, the influencing POIs for each subtrajectories and their influence durations can be easily computed for each trajectory with the following steps.

1. Find the subtrajectory enclosed by consecutive intersection points.

2. Find the common influencing POI of those two consecutive intersection points.

3. Find the difference of the timestamps of the two consecutive intersection points.

Algorithm 1 illustrates how trajectory and POI join is performed to find the set of TPTs.

4.2 Assign User Events

After we found the subtrajectory, POI, and their influence duration triples (T_s, p, d), the next phase is to assign activities to the TPTs based on given PAMS M. There are two cases to consider when assigning the activities to the triples. The first one is there is no activity happened, this is the case if d does not fall into the duration range of any activity at p, which is $\neg\exists_{e\in M.p.allEvents} e.t_{min} \leq d \leq e.t_{max}$ then $EA(p, d, L) = null$. The other case is there are some events happened, in this case both "and" and "or" semantic can be used. For example, given the following PAMS:

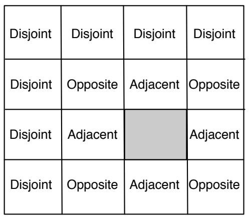
POI	Event	t_{min}	t_{max}
p_i	a	15	45
p_i	b	30	60
p_i	c	40	90

For a given TPT $(T_{s_i}, p_i, 50)$, the assigned activity should be $EA(p_i, 50, L) = (a \wedge b) \vee b \vee c$, where the event assignment function $EA()$ returns all combination of activities that d satisfies. More advanced event assignment functions can be designed by domain experts, by using other measures such as the sequence match. For example, if it is confirmed that the user goes to an restaurant 1 hour after he having dinner at another restaurant, it is unlikely this user is going to have dinner again. In this paper, we use the simple approach of event assignment function, which just assign all possible combinations of activities might happen given a TPT.

4.3 PTSJ

The steps covered in above subsections 4.1 and 4.2 to perform trajectory and semantic joins are internal memory based techniques, which requires all spatial objects to be loaded into internal memory for processing. Obviously this assumption is impractical if both POI and Trajectory sets are large. In this section, we cover our basic algorithm called *Partition based Trajectory and Semantic Join* (PTSJ). PTSJ is a modified version of PBMJ [19], which adopts a divide-and-conquer approach.

```
Algorithm 2: PTSJ
   Input: A set of trajectories $ST$, a set of POIs $P$ with their
          influence regions $VC$, and a PAMS $L$
   Output: A set of TPT and event pairs
 1 $p \leftarrow$ Estimate number of partitions, if not specified in the
   input;
 2 Partition table $G \leftarrow$ Partition the space into $p$ partitions;
 3 foreach $T \in ST$ do
 4     $G' \leftarrow$ The set of partitions that $T.mbr$ overlap with;
 5     Insert KPE of $T$ into partitions in $G'$;
 6 foreach $vc \in VC$ do
 7     $G' \leftarrow$ The set of partitions that $V.mbr$ overlap with;
 8     Insert KPE of $vc$ into partitions in $G'$;
 9 foreach partition $g \in G$ do
10     Load All Voronoi cells $VC'$ and Trajectories $ST'$ in $g$
       into internal memory;
11     $TPTS \leftarrow Trajectory\_POI\_Join(ST', VC')$;
12     Assign events for each $TPT$ in $TPTS$ and output the
       result;
```



Disjoint	Disjoint	Disjoint	Disjoint
Disjoint	Opposite	Adjacent	Opposite
Disjoint	Adjacent	▨	Adjacent
Disjoint	Opposite	Adjacent	Opposite

Figure 3: Partition Relationships

Algorithm 2 shows the steps of PTSJ. The first step is to divide the space into equal size partitions, and insert the Voronoi cells and the trajectories into those partitions. If the MBR of the spatial object o overlap with the partition g, then its key-pointer element (KPE), which is the (MBR, OID) pair, is inserted to g. Since both MBR of the spatial object and the partition is rectangle, whether they overlap can be determined very efficiently. The next step, we apply algorithm 1 to the spatial objects of each partition and then assign events to each TPT to report the trajectory semantic join result. The number of partition pp can be determined using the following formula if not provided as given in [19]:

$$pp = \lceil \frac{(||T|| + ||V||) \times sizeof(KPE)}{M} \rceil \tag{2}$$

Note that the number of partition is computed such that all spatial object can be fit into internal memory. In this case, pp does not guarantee that every partition can fit into the internal memory, so repartition of some over sized partition may be needed. In fact, the partition step does not only reduce the memory required to perform intersection computation, but also reduces the number of active spatial objects for the plain-sweep algorithm. So it also act as a filter step to reduce the cost for the refinement step. As a result, the number of partition p should be greater than pp calculated from the above formula. However, more partitions means that more duplicated entries, which increases the cost.

5. PTSJ WITH DUPLICATION REUSE

One of the problem with using partition and spatial approximations (i.e. MBR) in PTSJ is duplicated KPE entries. These spatial objects with duplicated KPE entries in different partitions will be loaded into main memory multiple times when perform the refinements step for each partition. We need a way to perform the refinement step for each partition in an order such that they share many duplicated KPE entries, so that the spatial objects with duplicated KPE can be reused rather than loaded from the disk multiple times. To describe how KPEs are shared by different partitions, we define relationships between the partitions. Given a partition g_a, if another partition g_b shares a side with partition g_a, then we call g_b is *adjacent* to g_a; If partition g_b shares a vertex with g_a, we call g_b is *opposite* to g_a; If partition g_b do not share side or vertex with g_a, we call g_b is *disjoint* with g_a. Figure 3 shows the relationships between the gray partition and the white partitions.

OBSERVATION 2. *Given a KPE of a spatial object, all partitions that contains this KPE are adjacent or opposite directly or indirectly to each other. The number of duplicated KPEs is either 1 or even numbers.*

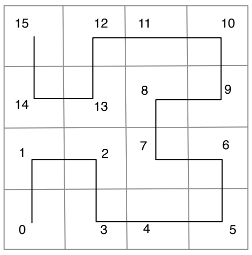


Figure 4: Hillbert order

Observation 2 shows how KPEs are replicated in different partitions. The number of duplicated KPEs is either 1 or even number is caused by using spatial approximations, which is always rectangles rather than random shapes. Here the order of each partition is retrieved for refinement computations plays an important role of how many spatial objects can be avoided to retrieve multiple times.

First of all, we examine the number of shared KPEs between a spatial partition with its adjacent partitions and then the shared KEPs with its opposite partitions.

LEMMA 1. *Let $adjacent(g_i)$ be the set of all adjacent partitions of g_i, $opposite(g_i)$ be the set of all opposite partitions of g_i, and $sk(g_i, g_j)$ be the number of shared KPEs between partition g_i and g_j. We have*

$$\neg\exists_{g_i \in adjacent(g_a), g_j \in opposite(g_a)}.sk(g_a, g_i) < sk(g_a, g_j)$$

Lemma 1 shows the number of shared KPEs between a partition and any of its adjacent partitions is always greater or equal to the number of shared KPEs the partition and any of its opposite partitions.

LEMMA 2. *Let $adjacent(g_i)$ be the set of all adjacent partitions of g_i, $disjoint(g_i)$ be the set of all disjoint partitions of g_i, and $sk(g_i, g_j)$ be the number of shared KPEs between partition g_i and g_j. We have*

$$\neg\exists_{g_i \in adjacent(g_a), g_j \in disjoint(g_a)}.sk(g_a, g_i) < sk(g_a, g_j)$$

Lemma 2 shows the number of shared KPEs between a partition and any of its adjacent partitions is always greater or equal to the number of shared KPEs the partition and any of its disjoint partitions. As a result, adjacent partitions have the greatest number of shared KPEs, and should be traversed in an order such that adjacent partitions have highest priority. We use a space filling curve to order the order of partitions to be traversed. There are many different space filling curves, in which Hillbert order as shown in figure 4 always traverse adjacent spatial partitions next.

Instead of loading spatial objects for the refinement computation for each partition in random order.The partitions are firstly ordered using Hillbert order, then only spatial objects that are not already loaded in the previous partition need to be loaded to perform the refinement computation. Algorithm 3 summarizes the PTSJ+ algorithm that uses duplication reuse technique to reduce the I/O cost.

6. EXPERIMENTS

In this section, we evaluate the performance of the two algorithms PTSJ, and PTSJ+. We first cover the experimatal settings in section 6.1, then we show how the performance

Algorithm 3: PTSJ+

```
Input: A set of trajectories $ST$, a set of POIs $P$ with their
       influence regions $VC$, and a PAMS $L$
Output: A set of TPT and event pairs
1  $p \leftarrow$ Estimate number of partitions if not specified in input;
2  Partition table $G \leftarrow$ Partition the space into $p$ partitions;
3  foreach $T \in ST$ do
4      $G' \leftarrow$ The set of partitions that $T.mbr$ overlap with;
5      Insert KPE of $T$ into partitions in $G'$;
6      Annotate each KPE with all the partitions they are
       inserted to;
7  foreach $vc \in VC$ do
8      $G' \leftarrow$ The set of partitions that $V.mbr$ overlap with;
9      Insert KPE of $vc$ into partitions in $G'$;
10     Annotate each KPE with all the partitions they are
       inserted to;
11 Sort the partitions in $G$ using Hillbert order;
12 Shared Voronoi cells $SSV \leftarrow \emptyset$;
13 Shared Trajectories $SST \leftarrow \emptyset$;
14 foreach partition $g \in G$ do
15     $VC' \leftarrow SSV \cup$ load Voronoi Cells in $g$ but not in $SSV$;
16     $SSV \leftarrow$ Voronoi cells in both $g.Contents$ and
       $g.next.Contents$;
17     $ST' \leftarrow \emptyset$;
18     foreach KPE $k \in g.TrajectoryContents$ do
19         if $k.id \in SST.ids$ then
20             $ST' \leftarrow ST' \cup \{SST.get(k.id)\}$;
21         else
22             $ST' \leftarrow ST' \cup \{loadFromDisk(k.id)\}$;
23     $TPTS \leftarrow Trajectory\_POI\_Join(ST', VC')$;
24     Assign events for each $TPT$ in $TPTS$ and output result;
```

varies on different number of trajectories in section 6.2, at last, we compares the algorithms on trajectories with different lengths in section 6.3.

6.1 Experimental Settings

All algorithms are implemented in Java, running on a Macbook with 2GHz Intel Core 2 Duo processor and 2 GB memory. We use California Point of Interests and road networks [16]. The POI file consists of 105725 POIs. Instead of using all POIs, we uses a subset that consists of 6487 POIs from a POI dense area shown in figure 5.

We implemented a trajectory simulator which generates trajectories on road networks using a modified version of the random way-point model. In order to simulate real world trajectory data, the trajectory simulator generates trajectories that can toggle with 5 modes, which are driving, walking, short stop, stop, and long stop. Each have different speed and difference possibility to change to other modes. The velocity of moving at each mode is computed with an Gaussian distribution function that make sure the velocity is not constant in one travelling mode but with similar average velocity. The exact configuration is shown in figure 6. As it is illustrated in figure 6, each mode can only switch to Walk mode, and Walk mode can switch to all other modes. The starting point of each trajectories are randomly placed on each road, so it will have the same distribution with the road network. Each point between each other on the same trajectory have a 1 minute timestamp difference.

We use the trajectory simulator described above with the California road networks to generate trajectory data set. PAMS is also randomly generated using seeded value with Gaussian distribution, those seeded values are 30, 60, 120, . . . minutes.

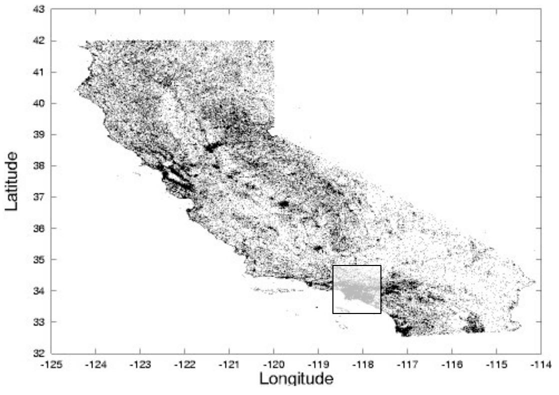


Figure 5: California POI

Since our optimization is on the I/O cost. We conduct our expriments measuring the I/O cost, which are the time used to retrieve spatial objects from the hard drive.

	Drive	Walk	Stop$_s$	Stop$_m$	Stop$_l$
Average Speed (Km/Hour)	50	5	0.3	0.3	0.3
Switching Mode (probability)	$\frac{1}{50}$	$\frac{1}{15}$	$\frac{1}{20}$	$\frac{1}{120}$	$\frac{1}{240}$
Can Switch To (Mode)	Walk	All	Walk	Walk	Walk

Figure 6: Trajectory Simulator Configuration

6.2 Comparison with Various Number of Trajectories

To do comparisons on different number of trajectories. We use a fixed number of POIs, the POIs are obtained by crop one POI dense area from the California POI data set as shown in figure 5. We perform four experiments on different number of trajectories, which are 4000, 8000, 6000, and 12000. All trajectories are simulated with equal length, which is 8 hours. We use a fixed 10×10 partitions. Here we evaluate the I/O cost of the algorithm.

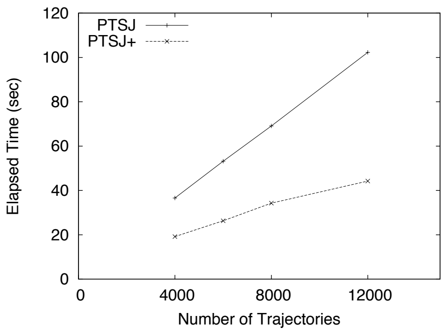


Figure 7: IO Cost vs Number of Trajectories

Figure 7 shows the I/O cost of the trajectory semantic join with various size of the trajectory data set. The result shows that the I/O cost of both algorithms PTSJ and

PTSJ+ increase linealy with the number of trajectories in the database. However, PTSJ+ uses less time to retrieve spatial objects from database. This shows that our duplication reuse technique is very effective, which avoids retrieve the spatial objects with duplicated entries.

6.3 Comparison with Various Length of Trajectory

In this subsection, we again use POIs cropped from California POI set as shown in figure 5. We use our trajectory simulator to simulate 4 sets of trajectories. Each contains 6000 trajectories. Each set contains trajectories with length 4 hours, 6 hours, 8 hours and 12 hours respectively. The objective of this experiment setting is to evaluate the effect of the length of trajectory with the performance.

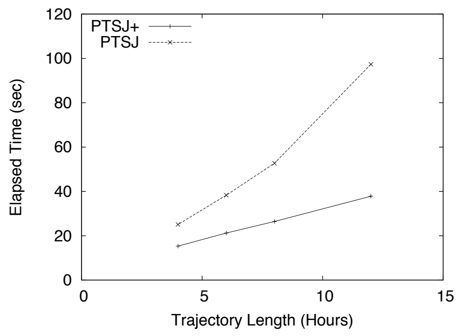


Figure 8: IO Cost vs Trajectory Length

Figure 8 shows the result of our experiments. As we could see from the experiment, the the length of trajectory have similar effect on both algorithms. PTSJ+ out performs PTSJ significantly, and the I/O cost of PTSJ+ grows slower than PTSJ, i.e. only 1/3 of PTSJ's cost whend dealing with 12-hour long trajectories.

7. CONCLUSION

Discover sequences of activities from trajectories is a challenging, yet useful problem. There is little existing work to associate trajectories with activities. This study formally defines the *trajectory semantic join* problem. We introduce two measures to find activities from trajectories. *Influence* measure associates trajectory with POIs, *influence duration* is used with PAMS to determine which activities took place at each part of trajectory given its influencing POIs. Two algorithms (PTSJ and PTSJ+) are proposed to perform trajectory semantic joins. Our experiment results show that PTSJ+ performs better than PTSJ by employ duplication reuse techniques that significantly reduce the I/O cost.

8. REFERENCES

[1] L. Alvares, V. Bogorny, B. Kuijpers, J. de Macedo, B. Moelans, and A. Vaisman. A model for enriching trajectories with semantic geographical information. In *ACM GIS*. ACM New York, NY, USA, 2007.

[2] L. Arge, O. Procopiuc, S. Ramaswamy, T. Suel, and J. Vitter. Scalable sweeping-based spatial join. In *VLDB*, pages 570–581. IEEE, 1998.

[3] T. Brinkhoff, H. Kriegel, and B. Seeger. Efficient processing of spatial joins using R-trees. In *SIGMOD*, pages 237–246. ACM New York, NY, USA, 1993.

[4] H. Cao, N. Mamoulis, and D. Cheung. Mining frequent spatio-temporal sequential patterns. In *ICDM*, page 8, 2005.

[5] J. Dittrich and B. Seeger. Data redundancy and duplicate detection in spatial join processing. In *ICDE*, pages 535–546, 2000.

[6] F. Giannotti, M. Nanni, F. Pinelli, and D. Pedreschi. Trajectory pattern mining. In *SIGKDD*, pages 330–339. ACM Press New York, NY, USA, 2007.

[7] O. Gunther and F. Ulm. Efficient computation of spatial joins. In *ICDE*, pages 50–59, 1993.

[8] Y. Huang, N. Jing, and E. Rundensteiner. Spatial joins using R-trees: Breadth-first traversal with global optimizations. In *VLDB*, pages 396–405. IEEE, 1997.

[9] E. H. Jacox and H. Samet. Spatial join techniques. *TODS*, 32(1):7, 2007.

[10] H. Jeung, H. T. Shen, and X. Zhou. Convoy queries in spatio-temporal databases. In *ICDE*, pages 1457–1459, April 2008.

[11] H. Jeung, M. L. Yiu, X. Zhou, C. S. Jensen, and H. T. Shen. Discovery of convoys in trajectory databases. *Proc. VLDB Endow.*, 1(1):1068–1080, 2008.

[12] P. Kalnis, N. Mamoulis, and S. Bakiras. On Discovering Moving Clusters in Spatio-temporal Data. In *SSTD*, page 364. Springer Verlag, 2005.

[13] S.-W. Kim, W.-S. Cho, M.-J. Lee, and K.-Y. Whang. A new algorithm for processing joins using the multilevel grid file. In *DASFAA*, pages 115–123. World Scientific Press, 1995.

[14] N. Koudas and K. Sevcik. Size separation spatial join. In *SIGMOD*, pages 324–335. ACM New York, NY, USA, 1997.

[15] J. Lee, J. Han, and K. Whang. Trajectory clustering: A partition-and-group framework. In *SIGMOD*, pages 593–604. ACM New York, NY, USA, 2007.

[16] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, and S. hua Teng. On trip planning queries in spatial databases. In *In SSTD*, 2005.

[17] Q. Li, Y. Zheng, X. Xie, Y. Chen, W. Liu, and W. Ma. Mining user similarity based on location history. In *ACM SIGSPATIAL*. ACM New York, NY, USA, 2008.

[18] K. Mouratidis, D. Papadias, and M. Hadjieleftheriou. Conceptual partitioning: An efficient method for continuous nearest neighbor monitoring. In *SIGMOD*, pages 634–645. ACM New York, NY, USA, 2005.

[19] J. Patel and D. DeWitt. Partition based spatial-merge join. *ACM SIGMOD Record*, 25(2):259–270, 1996.

[20] A. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and querying moving objects. In *ICDE*, 1997.

[21] Z. Song and N. Roussopoulos. K-Nearest Neighbor Search for Moving Query Point. *In SSTD*, 2001.

[22] Y. Tao, D. Papadias, and Q. Shen. Continuous nearest neighbor search. In *VLDB*, pages 287–298. VLDB Endowment, 2002.

[23] X. Yu, K. Pu, and N. Koudas. Monitoring k-nearest neighbor queries over moving objects. In *ICDE*, pages 631–642, 2005.

[24] X. Zhou, D. Abel, and D. Truffet. Data Partitioning for Parallel Spatial Join Processing. In *SSD*, pages 178–196. Springer-Verlag London, UK, 1997.