

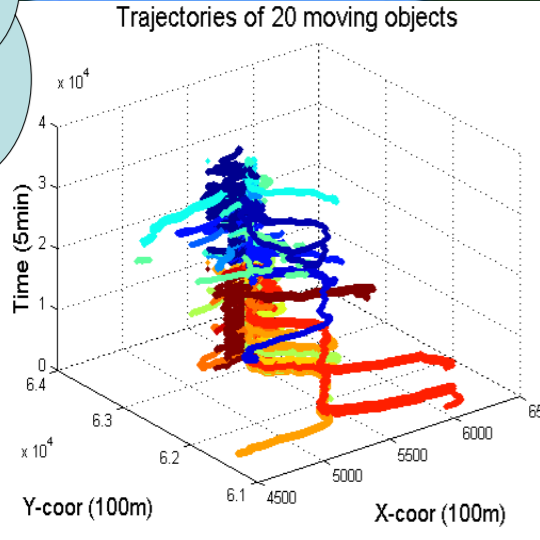
# Mining Long Sharable Patterns in Trajectories of Moving Objects

Győző Gidofalvi and Torben Bach Pedersen

Arrrrgggg, all this spatio-temporal data from moving objects!! What to do?! What to do...?!

People are predictable animals...

How can I extract the regularities / patterns in the trajectories?



Trajectories of 20 moving objects

Time (5min)  $\times 10^4$

Y-coor (100m)  $\times 10^4$

X-coor (100m)

But why do I even bother? What is in it for me?

If I knew the patterns..., I could:

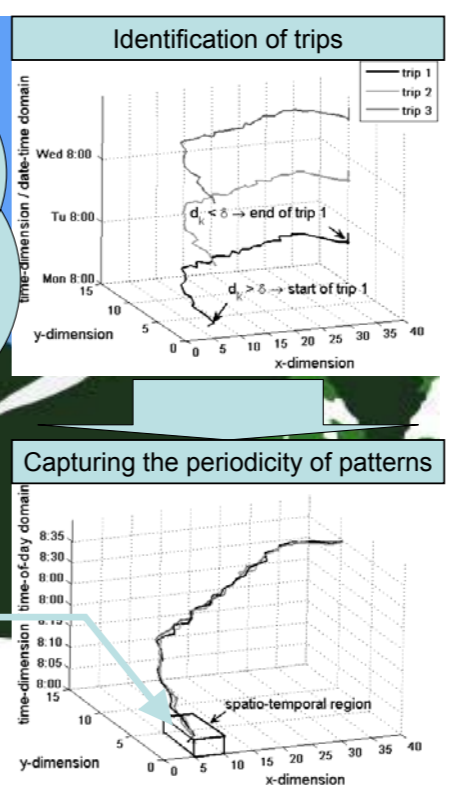
- aid the management, storage, and retrieval of trajectories
- improve tracking of moving objects
- provide customized Location-Based Services (LBS) and Location-Based Advertising (LBA)

If the patterns were **long** and **shared by multiple users / objects**, maybe I could get them to carpool, and reduce traffic!

To find patterns I need to:

- identify trips
  - a trip ends when the total displacement in the last  $k$  GPS readings is less than  $\delta$
- capture periodicity of patterns
  - Map date-time domain to time-of-day domain
- eliminate the problem of noisy GPS readings
  - Substitute readings with spatio-temporal regions

Eliminating the problem of noisy GPS readings



Identification of trips

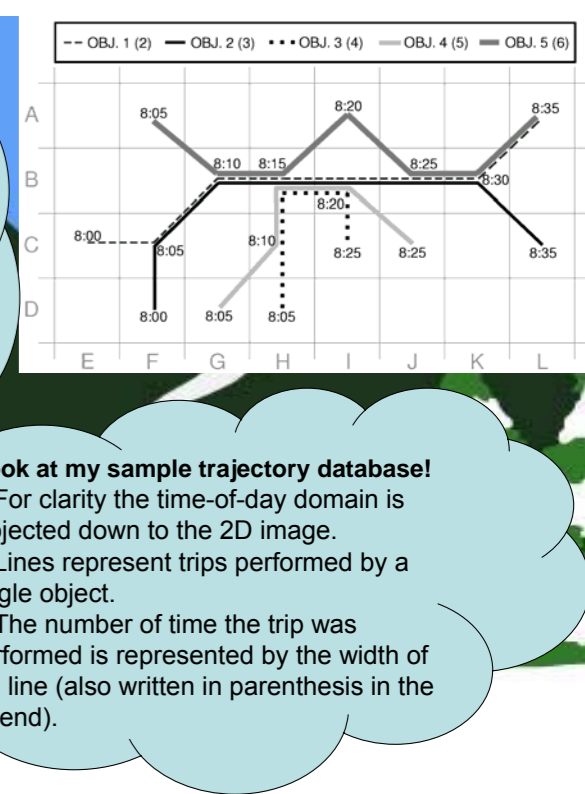
Capturing the periodicity of patterns

If I place for each trip, the unique spatio-temporal region identifiers inside a basket, I can analyze the set of baskets using a **Frequent Itemset Mining (FIM) algorithm** [AGR94].

Frequent itemsets will represent frequent trajectories.

Look at my sample trajectory database!

- For clarity the time-of-day domain is projected down to the 2D image.
- Lines represent trips performed by a single object.
- The number of time the trip was performed is represented by the width of the line (also written in parenthesis in the legend).



Frequent itemset mining, but for carpooling...

- a frequent itemset is only interesting if it is part of trips of multiple (say  $n$ ) cars
- a frequent itemset is only interesting if it is long
- subsets of frequent itemsets are only interesting if they are part of more trips than the superset
- interesting frequent itemsets may be part of a relatively few trips, and in that case traditional FIM algorithms run slow

How do I modify a traditional FIM algorithm to do all this efficiently?

Wait! All the data is already in my database in a relational format  $T = \langle tid, oid, item \rangle$ , where the columns contain trip-, object-, and spatio-temporal region identifiers.

So..., how do I write this algorithm in my favourite language, SQL?

STEP 1: Filter infrequent items!

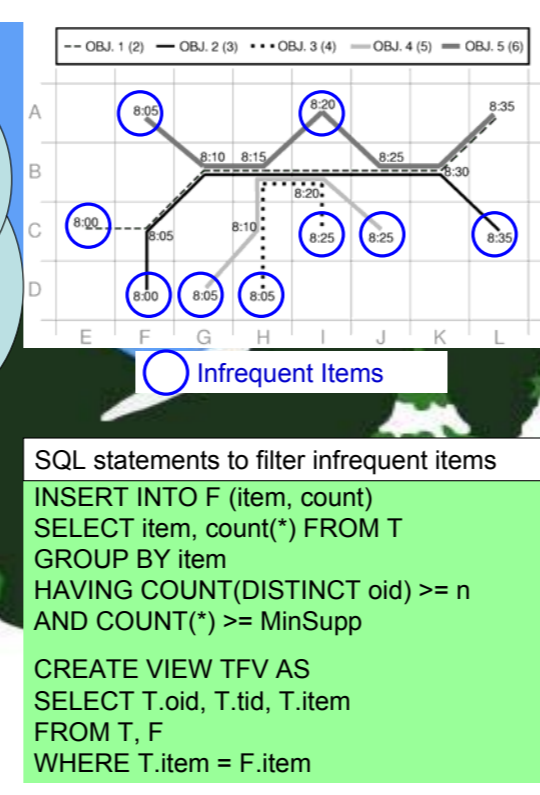
DEF: An item (ST-region) is frequent if the number of transactions (trips) that contain that item  $\geq$  MinSupp (for example 4), and the number of unique objects associated with those transactions  $\geq n$  (for example 2).

Infrequent Items

SQL statements to filter infrequent items

```
INSERT INTO F (item, count)
SELECT item, count(*) FROM T
GROUP BY item
HAVING COUNT(DISTINCT oid) >= n
AND COUNT(*) >= MinSupp

CREATE VIEW TFV AS
SELECT T.oid, T.tid, T.item
FROM T, F
WHERE T.item = F.item
```



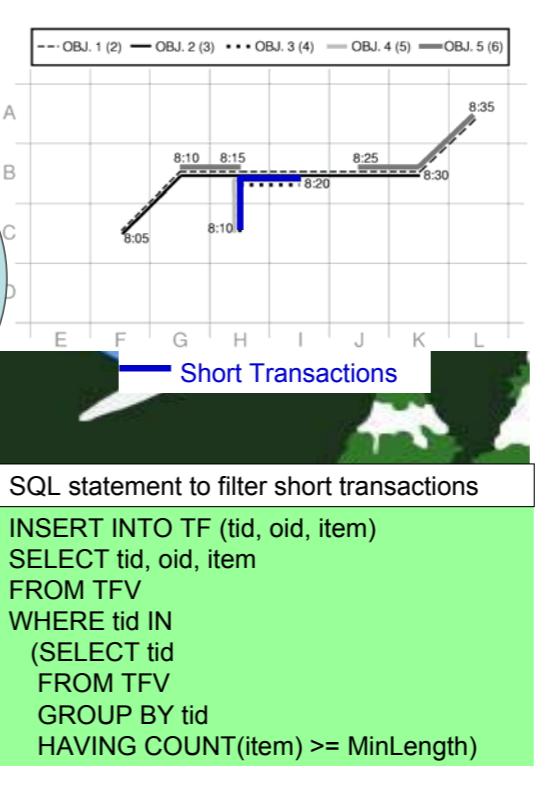
STEP 2: Filter short transactions!

DEF: A transaction is short if the number of items in it  $\geq$  MinLength (for example 4)

Short Transactions

SQL statement to filter short transactions

```
INSERT INTO TF (tid, oid, item)
SELECT tid, oid, item
FROM TFV
WHERE tid IN
(SELECT tid
FROM TFV
GROUP BY tid
HAVING COUNT(item) >= MinLength)
```



STEP 3 & 4: Project DB and find the single most frequent itemset!

DEF: An item-projected DB,  $T_i$ , contains all the items from the transactions containing item  $i$ .

There is a single most frequent itemset in any  $T_i$ , and its items all have maximal support in  $T_i$ .

single most frequent itemset in item-projected DB

support in predecessor DB

support in item-projected DB

projecting item

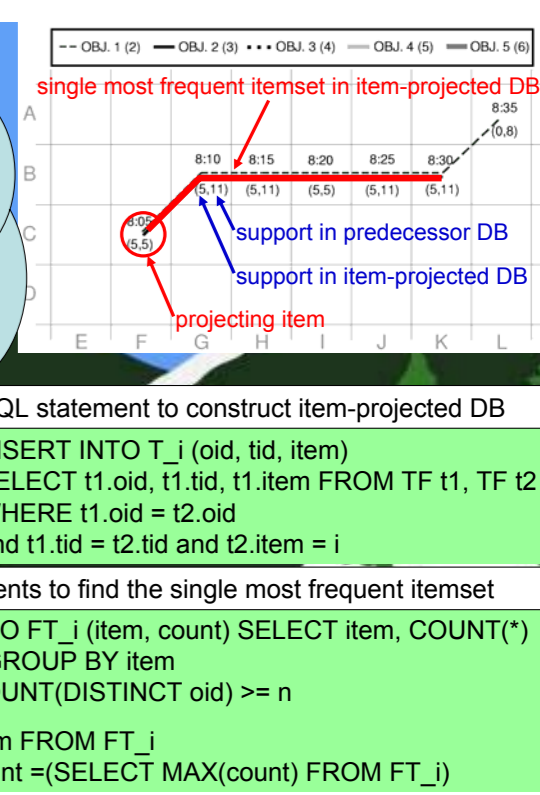
SQL statement to construct item-projected DB

```
INSERT INTO T_i (oid, tid, item)
SELECT t1.oid, t1.tid, t1.item FROM TF t1, TF t2
WHERE t1.oid = t2.oid
and t1.tid = t2.tid and t2.item = i
```

SQL statements to find the single most frequent itemset

```
INSERT INTO FT_i (item, count) SELECT item, COUNT(*)
FROM T_i GROUP BY item
HAVING COUNT(DISTINCT oid) >= n

SELECT item FROM FT_i
WHERE count = (SELECT MAX(count) FROM FT_i)
```



STEP 5: Delete unnecessary items from predecessor DB!

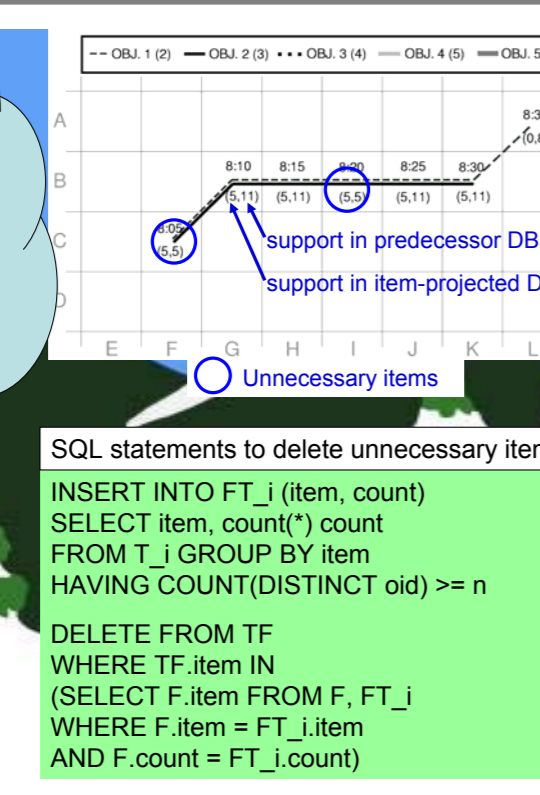
An item that has the same support in the projected DB as in the predecessor DB, can be deleted from the later.

Unnecessary items

SQL statements to delete unnecessary items

```
INSERT INTO FT_i (item, count)
SELECT item, count(*) count
FROM T_i GROUP BY item
HAVING COUNT(DISTINCT oid) >= n

DELETE FROM TF
WHERE TF.item IN
(SELECT F.item FROM F, FT_i
WHERE F.item = FT_i.item
AND F.count = FT_i.count)
```

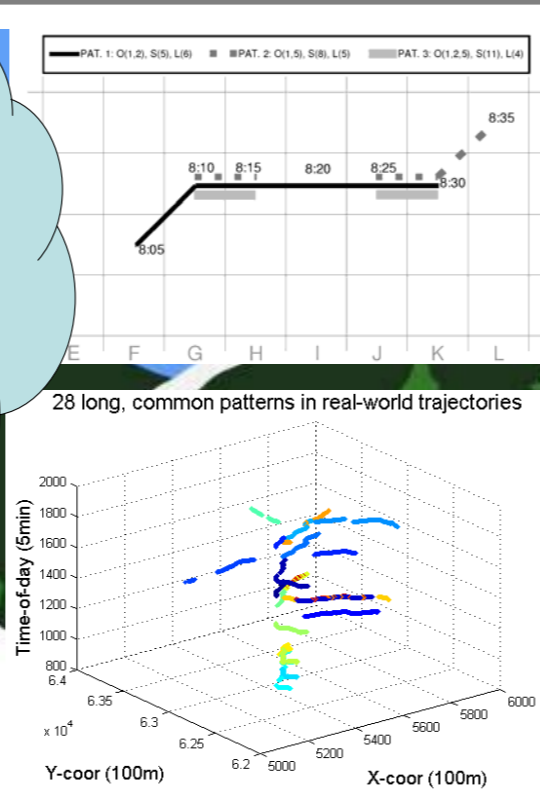


Project, find, delete, project, find, delete, ... until I find:

3 patterns in the sample trajectory DB, AND

28 long, common patterns in real-world trajectories

28 long, common patterns in real-world trajectories

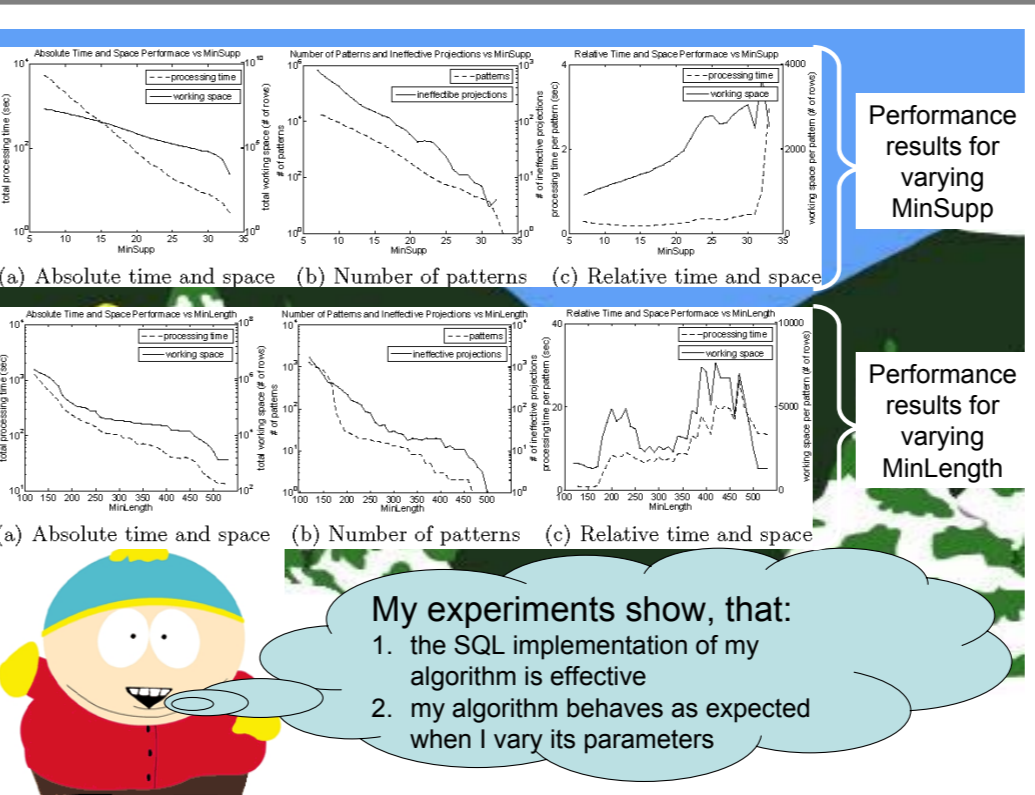


Performance results for varying MinSupp

Performance results for varying MinLength

My experiments show, that:

- the SQL implementation of my algorithm is effective
- my algorithm behaves as expected when I vary its parameters



I learned that such an algorithm has many uses in moving object DB management, Location-Based Services, and Location-Based Advertising.

We all learned something today, ... I learned that Cartman is not such a d...s after all, and can create something **simple, effective, and useful**.

Alright! That's it! I am goin' HOME!

AMM, I MMMM EHMM MMM, SQL AHMM, LBS, MMMMMMM CARPOOLING...?

