An algorithm for Trajectories Classification

Fabrizio Celli
28/08/2009

INDEX

ABSTRACT

Moving objects are objects like cars, persons or animals equipped with a GPS devise that have a geometry changing over the time (so the geometry is a function from a temporal domain to a range of spatial values): they produce trajectories, that is to say descriptions of the movements of those objects. We are interested in trajectories of people driving cars, that stop near some *points of interest* and reach them walking. We describe a conceptual model for those trajectories. We describe a probabilistic algorithm to *classify* a trajectory on the basis of the points of interest visited by a person during her trajectory.

APPLICATION SCENARIO

We collect data about the movements of persons driving cars in the city of Milan: cars are monitored by GPS devises. During her movement by a car, a person can stop driving in order to reach a place walking (for instance, she wants to go to take a coffee in a bar): the problem is that we don't know exactly where she is going to, because only the car is monitored by a GPS devise, so when a person is walking, we don't know her position. What we want to do is to try to understand which place the person is going to, in order to classify the person on the basis of her behavior: for example, we might be interested to see if the person is a tourist or a worker, a student or a housewife… To say the truth, we will not classify the physical person but her trajectory, as we will see next.

CONCEPTUAL MODEL

Starting from the paper "*A conceptual view of trajectories*" [S. Spaccapietra et al.], we can define a **trajectory** as a time-space function that records the changing of the position of an object moving in space during a given time interval. The definition of a trajectory is a user responsibility task: in fact, when we monitor an object (e.g. thanks to GPS devices), we obtain a lot of raw data (sequence of sample points, which are pairs <time,space>) and the user has to specify how to extract a trajectory from them. Thus, during its lifespan, a moving object can travel a lot of trajectories and the segmentation of its whole path into trajectories depends by the semantics that the application associates to trajectories.

The conceptual model developed in the mentioned paper describes a trajectory as a sequence of *stops* and *moves*. A *stop* is a part of a trajectory in which the moving object does not move, while a *move* is a part of a trajectory in which the moving object changes its position. Then, the trajectory

has a *begin* (that is the starting point) and an *end* (that is the end of the trajectory). All *stops* are temporally disjoint, while each *move* is delimited by two consecutive *stops*, or *begin* and the first *stop*, or the last *stop* and *end*, or *begin* and *end*. To simplify our discussion, we will represent a moving object as a point.

Now we can extend this conceptual model referring to our application, but first we must define some elements.

A **"point of interest"** is a physical location of which we know the geographical coordinates: for our application, it is a place that can be visited by a person (e.g. a bar, a museum…). A **"classification"** for a set of points of interest is a multilevel hierarchy that assigns categories to each point of interest: it is multilevel because in this way we can choose the granularity of the classification (e.g. first level will have less classes). A **"category"** is a label associated to a point of interest: the definition of categories is an application-dependent issue and it is related to the semantics that the application gives to a trajectory.

For our application, a trajectory is a sequence of *stops* and each *stop* must be associated to a set of *points of interest* reachable from that *stop*. The definition of the *stops* and their *points of interest* is an application responsibility task: in fact we are not interested in all physical stops of the car, for example we won't consider as *stop* those parts of trajectories in which the car does not move because of the traffic or a red traffic light. We have a *stop* only when a person is going to a place of interest walking.

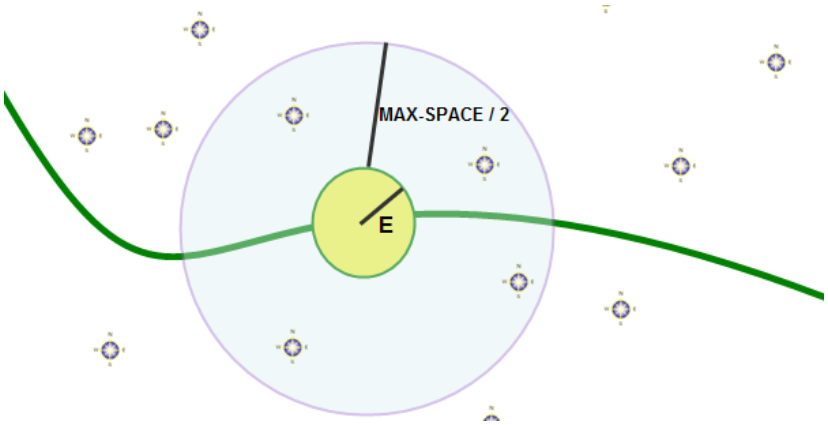
We assign to each *stop* a set of *points of interest* on the basis of:

- The **duration T** of the *stop.*
- The **average velocity V** of a person walking: this is a parameter of the application and it changes in different contexts. For example, if we are monitoring old persons, we can reduce V.
- The **measurement error E** of the input devise: in fact, when we consider the position of a *stop*, we must to be aware that the position is not accurate, because of the errors made by the input devise (and also other errors, such as the interpolation ones or the errors due to network delays). So we can think about a *stop* as a circle having radius equal to E: the area of that circle contains the exact position of the *stop*.
- The **average visit time AT** of each *point of interest*. This is also an application dependent parameter and represents the average time a person spends into a specific *point of interest*. We can extract this information by statistics.

Considering that, in order to reach a *point of interest (POI)*, a person has to walk from the car to the POI and then she has to come back to the car, we can compute the radius of the circle (centered in the current *stop*) that contains the *POIs* associated to the current *stop*:

$$MAXSPACE = V * T \qquad \textit{(maximum distance walkable)}$$

$$RADIUS = R = E + \frac{MAXSPACE}{2}$$



We must observe that, if the duration of the *stop* is wide, the set of *POIs* associated to that *stop* will have a huge cardinality: so we can assume that a person doesn't want to walk for more than **X** meters and **Y** seconds. X and Y are application dependent variables. In this way, the radius of the circle containing the interesting *POIs* is:

$$MAXSPACE = V * (\min(T, Y))$$

$$RADIUS = R = \min\left(X, E + \frac{MAXSPACE}{2}\right)$$

Moreover, we will not associate to the current *stop* all the *POIs* that are in the circle, but only those that have an *average-visit-time* compatible with the duration of the *stop*. In order to reach the *POI* a person need the following time:

$$Tp = d/V$$

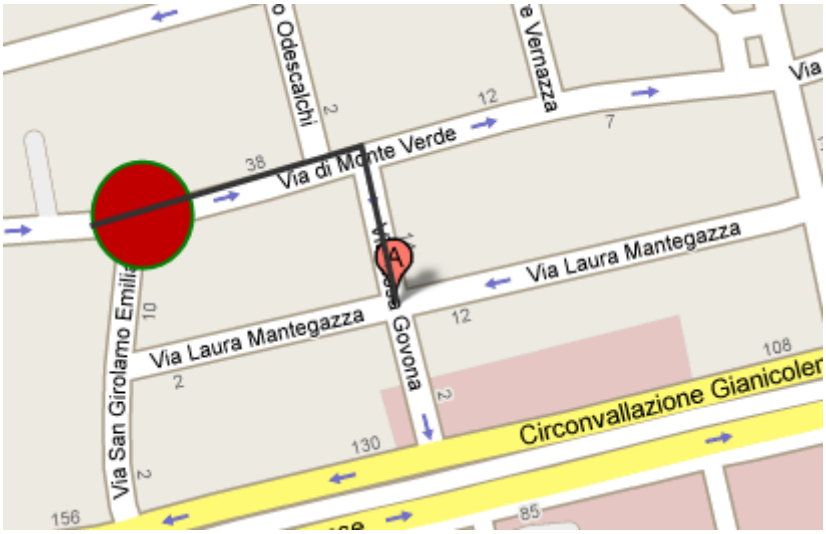
where **d** is the distance between the *POI* and the *stop*. Considering that the person has at least to go to the POI and come back to her car, the maximum time she has to visit the *POI* is:

$$TE = T - (2 \cdot Tp)$$

In this way we can state that we will consider only those *POIs* having an *average-visit-time* smaller than *TE*. So, for the *POI* **i***:*

$$if(TE_i > AT_i)$$
$$then\ TAKE(POI_i)$$

Another observation arises from the concept if the **distance d** of the *POI* from the *stop*: the person is walking, so she has to follow the geometry of the roads. Thus, **d** is not the aerial distance but the minimum distance walking over a road. In the work "Trajectory Data cleaning" [Hung, Zhixian, Spaccapietra] there is the description of an algorithm that maps a trajectory on a road map with an accuracy of 92% if the measurement error is at most of 8 meters. So we can map our *stops* and *POIs* over a road map and apply an algorithm that computes minimum distances, such as the *Dijkstra Algorithm*.



At last, the application could introduce an upper bound to the number of *POIs* that can be associated to a single *stop*.

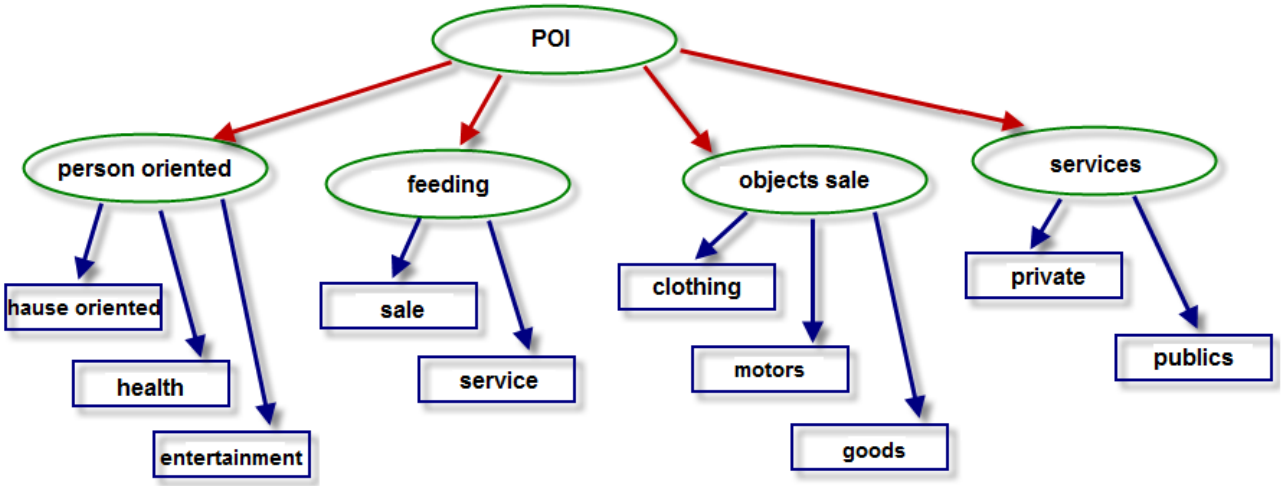
THE PROBLEM

"Given a trajectory described in terms of our conceptual model, we want to assign a probability to each POI to be the goal of the associated stop. In this way we can classify the trajectory, that is to say that we can assign a category to the trajectory on the basis of the POIs more probably visited by the moving person".

The **goal** of a *stop* is the *POI* visited by the moving person. This means that we will use an algorithm that assigns a probability to all the *POIs* of each *stop*: considering that each *POI* belongs to a category, the whole trajectory can be classified on the basis of the categories of the *POI* visited. Thus, for instance, we can classify a trajectory as a *touristic* trajectory or as a trajectory of a *worker*.

Then we can use these results as input for applications that try to extract behavior patterns from trajectories, used by sellers, touristic agencies or the Government to reach their aims.

Here is an example of classification of *Milan points of interest*: the main categories are "person oriented", "feeding", "objects sale" and "services", that reflect a classification of the types of *POIs*. For the moment we assume that each *POI* has only a category, that is to say that he can be only into one branch of the categories tree.



THE ALGORITHM

INPUT

- A set of *Trajectories*, defined as a sequence of *Stops* S
- A set of *POI* having a *position <x,y>,* an *average-visit-time* ***AT***, a category ***C***
- Each *Stop* S of each *Trajectory* must have a *duration* ***T***, a *position <x,y>,* a set of *POI* associated according to our conceptual model
- The average velocity **V** of a walking person (application dependent parameter)

OUTPUT

- A probability for each *POI* of each *Stop* to be the *POI* visited in that *Stop*
- A classification all *Trajectories*: for each *Trajectory* we will have the most probable category with an associated probability

This is an algorithm that computes probabilities and this computation is characterized by successive refinements.

1. First of all, considering the **generic trajectory J**, we try to find a *stop* for which is simple to understand the correct goal *POI*: in fact, if for a *stop* we have only one *POI*, we can assume that the *POI* is 100% the goal of the *stop.* So we can order the *stops* of the trajectory J for growing number of *POIs* associated.
2. Then we assign a probability to all *POIs* of all *stops* of J on the basis of their distances from the associated *stop* and of their *average-visit-time*: thus, the nearest *POI* to the *stop* will have more probability to be its goal and this probability is refined by the comparison between the *average-visit-time* of the *POI* and the *duration* of the *stop*. The last statement means that, if we suppose that for each *stop* we have only one *POI* visited, we prefer a *POI* whose *average-visit-time* is closer to the *duration* of the *stop*: then, if we assume that we can have more than one *POI* visited for each *stop*, this assumption falls and has to be worked out again, for example by considering different weights for the *distance* criterion and the *average-visit-time* one.
3. Now we have a probability for each *POI* of each *stop*. We can refine these probabilities by considering the category of the *POIs*: this means that, if for a lot of *stops* of the trajectory J we have identified as goals *POIs* of a category *X*, more probable also for the current *stop* the goal *POI* will belong to the category X. In fact, we are trying to classify a trajectory

according to predefined categories, so that if it is a *feeding* trajectory, it is allowable to think that the person has visited a lot of *POIs* of "feeding" category. We can use this criterion only when, from the step 2, we haven't identify a *POI* with a probability that exceeds a given *threshold T1* (that is an application dependent parameter): so we can analyze all the past history of the current trajectory, i.e. the *stops* already considered, and update the probabilities of the *POIs* of the current *stop* on the basis of their categories.

4. If for the current *stop* we obtain a probability for a *POI* **Z** exceeding a *threshold T2* (that is an application dependent parameter), we can perform a backtracking in order to update probabilities of all *POIs* of all *stops* already considered for the trajectory J, on the basis of the category of Z. Thus we are saying that, if for a *POI* we are sure that it is the goal of the current *stop* (where "sure" is connected to the value of the *threshold T2*), we can use the *category criterion* to refine probabilities of the *POIs* already considered. The meaning of backtracking is the following: with step 1 we find stop having an high level of certainty about the goal *POI*; then, if an uncertain stop happens at the end of a trajectory, we can increase the level of certainty using past history; but if an uncertain stop happens at the beginning of the trajectory, backtracking is the only way to increase its level of certainty.
5. At last, we can assign a probability to the whole trajectory J on the basis of the probabilities assigned to each *POI* of each *stop.* So, for each existing category, we compute the probability for it to be the category of the trajectory: at the end we will have a probability for each category and we will take the biggest one.

DETAILS

In this section we want to analyze which formulas are used to compute the probabilities for each *POI* of each *stop*.

We start from the **second step** of the algorithm, in which we assign probabilities on the basis of *POI* distance from the *stop* and *POI average-visit-time.* We assume the following symbol:

- $\boldsymbol{S_Z}$ is the current *stop*
- $\boldsymbol{T_Z}$ is *stop* duration
- $\boldsymbol{V}$ is the average velocity of a walking person
- $\boldsymbol{T_{Mi}}$ is the *average- visit-time* of the *POI* "**i**"
- $\boldsymbol{d_i}$ is the distance between the *POI* "**i**" and the *stop* S_Z
- $\sum_j \ldots \sum_K \ldots$ are a sum over all *POIs* "**j**" or "**K**" associated to S_Z

- T_{Ei} is the maximum time that a person has to visit the *POI* "**i**", that is to say the difference between the duration of the stop and the time needed to reach "i" and to come back to the stop.

The formula that computes the probability related to the distance between the *POI* "**i**" and its associated *stop* is:

$$SP_i = \frac{\frac{\sum_j d_j}{d_i}}{\sum_K \frac{\sum_j d_j}{d_K}}$$

The more the *POI* is closed to the *stop*, the bigger is the value returned by this formula.

Instead, the formula that takes into account the *average- visit-time* of the *POI* "**i**" is the following:

$$TP_i = \frac{T_{Mi}/T_{Ei}}{\sum_j (T_{Mj}/T_{Ej})}$$

that, as we have just explained, prefers that *POI* whose *average-visit-time* is closer to the *duration* of the *stop*.

We can connect this two formulas to compute the total probability:

$$P_{Ti} = \frac{TP_i + \alpha \times SP_i}{\alpha + 1}$$

where α is a weight used to give more importance to the distance-criterion.

Now we can analyze the **third step** of the algorithm, the one in which we consider the past history of the trajectory. Here a good threshold can be:

$$T1 = 50 + \frac{50}{number\ of\ POIs}$$

that means that we perform a past analysis only if we haven't a *POI* with a probability bigger than 50%, that is to say that the level of uncertainty is very high.

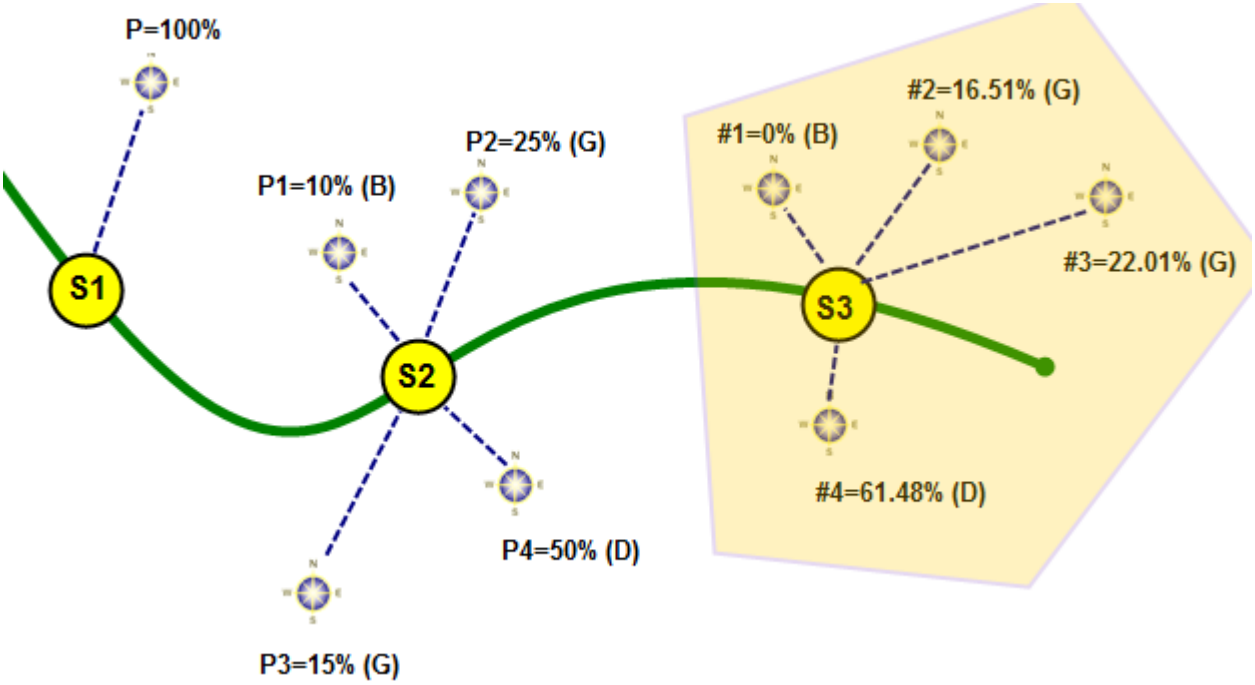
So, if we don't reach the threshold T1, we have to perform a past analysis to update probabilities on the basis of categories of the *POIs*. We can show this analysis with an example.

Considering that we are at the third stop and that we have only three categories: B, D and G. For the first stop we have only one *POI* associated, of category D, with a probability of 100%. For the second stop the situation is the following:

POI	CATEGORY	PROBABILITY
P1	B	10%
P2	G	25%
P3	G	15%
P4	D	50%

that is 10% for category B, 40% for category G and 50% for category D. Note that P2 joins the category D with a ratio of 25/40, while P3 of 15/40.

For the current stop we have four *POIs* associated, where #2 and #3 belong to the category G. The following figure summarizes the situation:



So at the third stop, for the category G we have an aggregate probability of $16.51 + 22.01 = 38.52$.

We can update probabilities of current *categories* on the basis of the categories of the past stops:

$$P(G3) = \frac{0+40+38.52}{3} = 26.18$$

$$P(D3) = \frac{100+50+61.48}{3} = 70{,}49$$

$$P(B3) = \frac{0+10+0}{3} = 3.33$$

So the probabilities for current *POIs* will be:

$$P_{T2} = \frac{16.51}{38.52} * 26.18 = 11.22\% \qquad P_{T3} = \frac{22.01}{38.52} * 26.18 = 19.96\%$$

$$\boldsymbol{P_{T4} = 70.49\%} \qquad P_{T1} = 3.33\%$$

About the **fourth step** of the algorithm we can say that here the threshold T2 means the certainty that we have about a *POI* to be the goal of the stop. So T2 must be closed to 100%, for example:

$$T2 = 90 + \frac{10}{number\ of\ POIs}$$

In order to assign a type to the whole trajectory, for each category defined we compute the probability. That is to say that, for each category, we sum all the probabilities of all *POIs* of that category and we divide this number for the total sum of probabilities.

For example, considering a trajectory of five stops in which the aggregate probabilities for each category are described in the following table:

STOP 1	**STOP 2**	**STOP 3**	**STOP 4**	**STOP 5**	
B1=70%	G2=60%	B3=40%	D4=50%	B5=70%	TOT: 290 punti
G1=30%	B2=30%	G3=39%	B4=40%	G5=20%	TOT: 159 punti
D1=0%	D2=10%	D3=21%	G4=10%	D5=10%	TOT: 51 punti

For each category we compute the probability:

$$\boldsymbol{B = \frac{180 + (30 + 40) + 0}{290 + 159 + 51} * 100 = 50.00\%}$$

$$G = \frac{60 + (30 + 39 + 20) + 10}{290 + 159 + 51} * 100 = 31{,}8\%$$

$$D = \frac{50 + (10 + 21 + 10) + 0}{290 + 51 + 159} * 100 = 18.2\%$$

So the current trajectory belongs to the category B, with a certainty of 50%.

THE ALGORITHM PSEUDO-CODE

INPUT

- A set of *Trajectories*, defined as a sequence of *Stops* S
- A set of *POI* having a *position <x,y>,* an *average-visit-time* ***AT***, a category ***C***
- Each *Stop* S of each *Trajectory* must have a *duration* ***T***, a *position <x,y>,* a set of *POI* associated according to our conceptual model
- The average velocity **V** of a walking person (application dependent parameter)

OUTPUT

- A probability for each *POI* of each *Stop* to be the *POI* visited in that *Stop*
- A classification all *Trajectories*: for each *Trajectory* we will have the most probable category with an associated probability

```
1: LIST<STOP> all = SORT(tutti gli Stop per numero crescente di POI)
2: LIST<STOP> past = null     //stop already visited
3: FOR EACH stop S OF all DO

4:      assignProbabilityUsingDistanceAndAverageTime(all)
5:      double threshold5 = 50+(50/size(L))
6:      double threshold7 = 90+(10/size(L))
7:      POI MP = getPoiHavingMaxProbability(L)

8:      IF(NOT(probability(MP)> threshold5) && size(past)!=0)
9:           THEN updateProbabilityUsingPastHistory(S, past)

10:     POI MP = getPoiHavingMaxProbability(L)
11:     IF(probability(MP)> threshold7)
12:               THEN backtracking(S,past)

13:     past.add(S)
14: END FOR
15: assignProbabilityToTrajectory(all)
```

Let us now investigate the main procedure of this algorithm. The instruction 4 implements the second step of the algorithm, that is to say that we assign a probability to each *POI* of each *stop* on

the basis of the distance of the *POI* from the *stop* and the *POI average-visit-time*. The procedure is the following:

INPUT: a list **L** of all *POIs* to consider

OUTPUT: a probability for each *POI* to be the goal of the associated *stop*

```
1:    double totalDistance = 0
2:    double totalTimeRelation = 0
3:    double totaldistanceRelation = 0
```

//computes total distance

```
3:    FOR EACH poi P OF L DO
4:          totalDistance += P.getDistanceFromStop()
5:    ENDFOR
```

//computes time and distance relations

```
6:    FOR EACH poi P OF L DO
7:          double timeRelation = P.getAverageTime/P.getEffectiveTime
8:          P.setTimeRelation(timeRelation)
9:          double distanceRel = totalDistance/ P.getDistanceFromStop()
10:         P.setDistanceRelation(distanceRel)
11:         totalTimeRelation += timeRelation
12:         totaldistanceRelation += distanceRel
13:   ENDFOR
```

//computes probability

```
14:   FOR EACH poi P OF L DO
15:         double TP = P.getTimeRelation()/totalTimeRelation
16:         double SP = P.getDistanceRelation()/totaldistanceRelation
17:         P.setProbability((TP+4*SP)/5)
18:   ENDFOR
```

RESULTS

Now we can give the results reached by the implementation of the algorithm, showing some statistic details.

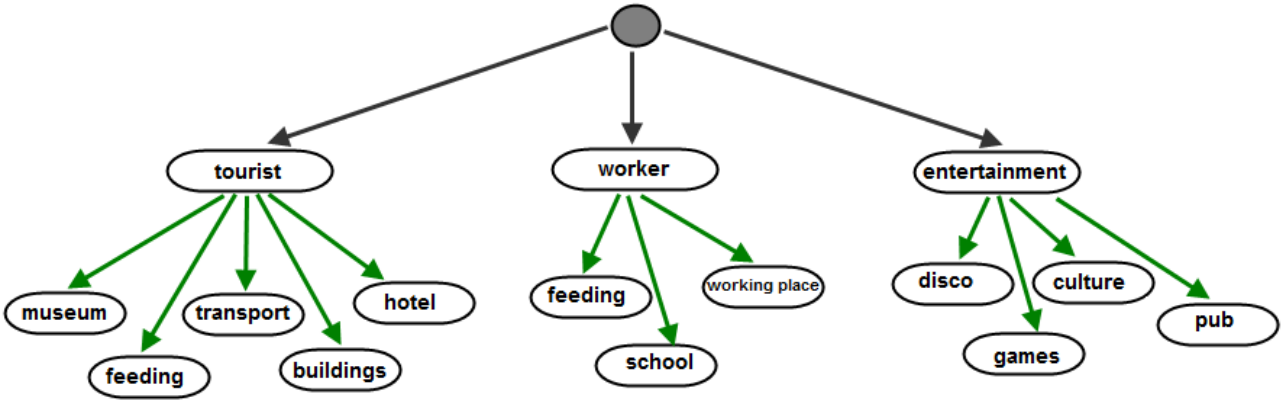
- The algorithm has been implemented in JAVA: it needs only few seconds to run, compute trajectories classification and update the DB (**6 seconds for 734 stops and 121 trajectories**)
- Our POIs refers to Milano's center POIs and has been classified according to the following main categories:
    - SERVICES (4339 POIs)
    - FOOD (7036 POIs)
    - PERSON ORIENTED (15371 POIs)
    - ITEM SALE (12510 POIs)
- The test environment consist of:
    - 1436 stops
    - 734 stops without POIs associated. There are two reasons for this situation: the first is that a stop is too short, so the person hasn't enough time to visit a POI; the second is that a stop happens in the suburbs of Milan, while we have information about POIs in the center of Milan.
    - 702 stops having a number of associated POIs included in the interval [1,8]
    - 121 trajectories having a number of stops included in the interval [4,15]
    - 5.8 stops on average for each trajectory
- The actual values for application parameters are:
    - Velocity of a walking person = 1.3 m/s = 4.68 Km/h
    - Maximum distance that a person wants to walk = 1000 meters
- The output of the algorithm is a classification of all 121 trajectories:
    - 59 ITEM SALE trajectories
    - 9 FOOD trajectories
    - 2 SERVICES trajectories
    - 51 PERSON ORIENTED trajectories

	TRAJ_ID	TRAJ_TYPE	TRAJ_TYPE_PROBABILITY
1	24756100	items_sale	0,8268083956868176
2	7305900	items_sale	0,5224954806175307
3	12758800	items_sale	0,7699782336921265
4	30482500	items_sale	0,7391142714611428
5	32902800	items_sale	0,7933920623329205
6	589200	items_sale	0,888701061711125
7	27398700	items_sale	0,8729717936665202
8	22463500	items_sale	0,9424722041360427
9	32834300	items_sale	0,7347639699577474
10	32041800	items_sale	1
11	31663800	person oriented	0,6965128763217244
12	15107900	items_sale	0,9270557313797789
13	7047900	items_sale	0,9258616138097218
14	27177900	items_sale	0,720377772969764
15	32405600	items_sale	0,871095886850189
16	31191100	feeding	0,5052895184976667
17	30300100	items_sale	0,9870433287381777
18	17343200	items_sale	0,8600368769928447

CLASSIFICATION PATTERNS

Once we have yield an algorithm for trajectories classification, we can try to improve it by adding semantic information. We may think of a decision process, that is to say that, during classification, we can take care about some rules (or *patterns*) that help the classification of a trajectory. These **rules** are annotations that refer to the categories we are using for the current application and that say what can or cannot happen in a trajectory of a given category or in a stop with certain parameters. In this way, during the process of trajectory classification, we can exclude those situations that don't satisfy our patterns: patterns can be used to bind the classification of a single stop or of a group of stops, as we will see next.

Suppose we are using the following categories, belonging to a two-level hierarchy:



The meaning of the picture is the following: if, for example, a POI belongs to the second-level category "hotel", it also belongs to the main category "tourist". We assume that **a POI can belong to more than one category**, which means that it can belong to different categories of the same hierarchal level, for example it can be a "feeding" for "tourist" and a "feeding" for "worker".

Now that we have defined an hierarchy, we can write some rules. First of all, we can observe that there are two kinds of rules:

- (**A-rules**) Rules that bind the category of a single stop, independently from categories already assign to the other stops of the same trajectory. This kind of rules are used to exclude some possibilities from our decision process, looking only at the current stop, and need only information about the current stop and its associated POIs.

- (**B-rules**) Rules that foresee the category of the current stop on the basis of the categories assigned to past stops. This kind of rules also influence the classification of the whole trajectory.

Rules are annotations associated to categories and they represent procedures that take in input some parameters and return a Boolean value. The first issue we have to deal with is the language of these annotations: we may think to use a system-generic language and then to translate it in a system-specific one; for example, we can use a pseudo-code syntax that will be translated in Java by a preprocessor of the algorithm.
Let's see some example of A-rules, that take in input only the current stop and one of its POIs. The language used is a Java-like system-generic language:

```
IF(poi_category_level2="FEEDING" AND hours_of_stop=[22:00;07:00])
      THEN RETURN FALSE;
```

this rule means that a person can't go to a restaurant (or another "feeding" place) in the hours included between 22:00 and 07:00. We can observe that the only information needed by this kind of rule regards the current stop and its associated *POIs*.
Other A-rules can be:

```
IF(poi_category_level2="MUSEUM" AND hours_of_stop=[19:00;09:00])
      THEN RETURN FALSE;

IF(poi_category_level2="WORKING PLACE" AND duration_of_stop<2H)
      THEN RETURN FALSE;
```

As far as B-rules concerns, they represent procedures that take in input the current stop, one of its POIs and the stops of the same trajectory already classified. For example:

```
IF(poi_category_level1="TOURIST" AND poi_category_level2="FEEDING"
            AND last_stop_category_level2="FEEDING")
      THEN RETURN FALSE;
```

this rule means that, if we are analyzing for the current stop a *POI* belonging to the main category "tourist", if it is a place for feeding but last stop has already been associated to a place for feeding, this POI have to be excluded, i.e. it cannot be the goal of the current stop. The meaning of this rule is that maybe a tourist will not go to two "feeding place" consecutively, but the rule can be refined for example by considering the duration of the previous stop.

B-rules can also be procedures that return a category instead of a Boolean value. For example:

```
IF(poi_category_level1=”TOURIST” AND last_stop_category_level2=”HOTEL”
            AND period_of_the_day=”MORNING” AND weather=”sunny”)
      THEN RETURN poi_category_level12(”MUSEUM”);
```

that means that if a tourist has been in a hotel in the morning and it is sunny, then he will go to a museum (these kinds of association can be extracted by statistics or by experts). Note as in the last rule there are also information about the ***context***, such as the weather and the period of the day.
Moreover, we could introduce **C-rules** as those rules similar to B-rules but that return also a probability: for example, last B-rule asserts that the tourist certainly will go to a museum; instead a C-rule asserts the probability the tourist will go to a museum:

```
IF(poi_category_level1=”TOURIST” AND last_stop_category_level2=”HOTEL”
            AND period_of_the_day=”MORNING” AND weather=”sunny”)
      THEN RETURN poi_category_level12(”MUSEUM”, 80%);
```

and this probability can be used by the algorithm to update the probability of the current *POI*.

Now we can analyze how to use these patterns/rules inside the algorithm. We have to remember that the definition of the categories and of the parameters used to associate *POIs* to *stops* is a user responsibility task: in this way, also the definition of the rules is a user responsibility task and it affects the output of the algorithm. In fact, using these rules, when the algorithm tries to assign a probability to the *POIs* of a stop, it can filter in this way:

- A-rules exclude those *POIs* whose characteristics don’t match with stop’s parameters. For example, if the opening time of a *POI* is outside the time interval of the *stop*, that *POI* will have certainly a probability of 0% to be the goal of the stop and won’t be considered anymore by the algorithm.
- B-rules (and C-rules) help to make foresights on the more probable category of a stop on the basis of the more probable categories associated to past stops of the same trajectory. In fact we can define a sequence of *POIs* to be visited in a certain trajectory, so if in the past a group of *POIs* belonging to this sequence have been visited, we can foresee the future *POIs*.

Moreover, this is a very simple mechanism: in fact, it is sufficient to annotate a category with a list of rules and then the algorithm will take care about these annotations during its computation. In this way we have added semantics to categories.

MORE COMPLEX PATTERNS

C-rules can be extended by introducing a probability function: we call **D-rules** those rules that are similar to C-rules but that don't return a single value of probability, however a user-defined function used to compute a value of probability.

```
IF(poi_category_level1="TOURIST" AND last_N_stop_category_level2="HOTEL"
        AND period_of_the_day="MORNING" AND weather="sunny")
    THEN RETURN poi_category_level12("MUSEUM", [80-((N-1)*10)] %);
```

This rule means that if a tourist has been in a hotel in the morning, there is an high probability that he will go to a museum if it is sunny, but this probability decreases with the increase of the number of stops after the hotel. So the rule says that is more probable that the tourist will go to a museum immediately after he has been in a hotel.

D-rules are patterns that can be used in place of complex systems such as the ones based on "Markov Model".

At the moment we have:

- A-rules that consider only the single stop
- B-rules, C-rules and D-rules that consider the whole trajectory

Another kind of rules can consider groups of trajectories, that is to say that they compare different trajectories of the same person. We call these rules as **E-rules**.

For example:

```
IF(poi_category_level1="TOURIST" AND
      (last_N_PAST_trajectory contains "LOUVRE MUSEUM") AND
      poi_category_level3=" LOUVRE MUSEUM")
      THEN RETURN MAX(50, [10+((N-1)*10)] % );
```

In this rule we have extended the categories' hierarchy by considering a more deep level: the third level is a specialization of the second one, so "LOUVRE MUSEUM" is a "MUSEUM" at the second level. The rule says that if a tourist has already visited the "Louvre" in a recent past trajectory, it is unlikely that he will visit it again. Note that the return value of probability increases with the "old age" of the past trajectory considered.

We can extend E-rules by considering the context in their conditions and also by returning more complex results, such as the prediction of a category for the current stop or the current trajectory.

RELATED WORKS

In literature "uncertainty" has been defined as the measure of the difference between the actual content of a database and the content that the current user would have created by direct observation of reality. There are a lot of kinds of uncertainty that can be analyzed. We can find uncertainty in the acquisition process of the raw components of a model (the extraction of the sampling points): though progress of technologies (such as GPS) and improvements in storage techniques have greatly improved the quality of spatial data, it is simply not possible to eliminate errors in data acquisition. This kind of uncertainty does not refer only to the spatial dimension, but also to time and speed of the moving object. This is called **measurement uncertainty.** In addition to the errors made when recording the sample points, to obtain the curve representing the trajectory we have to apply an interpolation method to that sample points (usually a linear interpolation), so the resulting curve will only be an approximation of the real trajectory: In literature this is called **interpolation uncertainty**.

In this article we have chosen a representation model for measurement and interpolation uncertainty: in fact, as we have already explained at the beginning of this document, we model a *stop* as a circle, which radius is the known maximum measurement error of the input devise. In this way, the area of the circle is the place where our *stop* is contained.

This approach is similar to the one proposed in [3]: that work introduces a threshold **r** that denotes the maximum distance of the point to the assumed location of the trajectory. Thus, each point *(x,y,t)* of the trajectory becomes the center of an horizontal disc of radius equals to the threshold and the trajectory is modeled as a sheared **cylindrical volume** in 3D space around the given trajectory polyline. This is shown in figure 1.

A more complex solution to the problem of measurement and interpolation uncertainty has been proposed in [5] and it has been called "*bead model*". Here the uncertainty associated with the location of an object travelling between two endpoints of a line segment can be an ellipse with foci at the endpoints. The problem is that this model works under the assumption that we must know an upper bound for the object's speed between sample points in order to calculate 3D cones centered in each sample point: the intersection of two cones is an ellipse. Bead model reduces the uncertainty by a factor 3 (a cone as 1/3 volume of its minimal bounding cylinder) but

is not easy to handle and to query; moreover it is no easy to find an upper bound for velocity. Figure 2 shows this model.

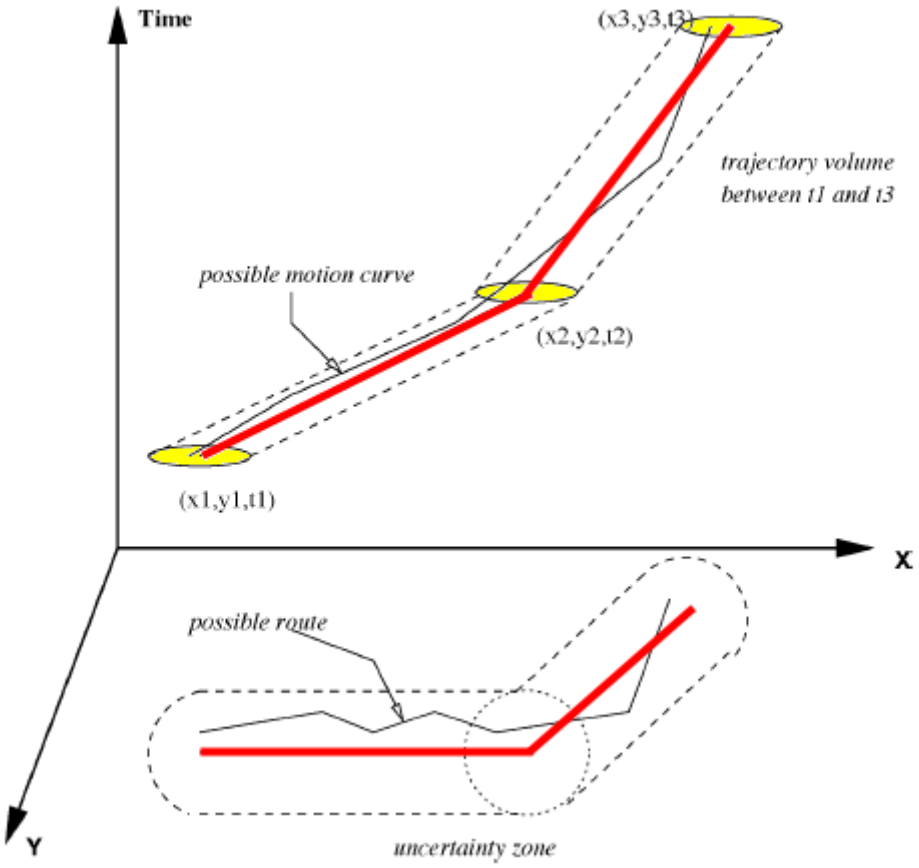


Figure 1: solution proposed in [3]

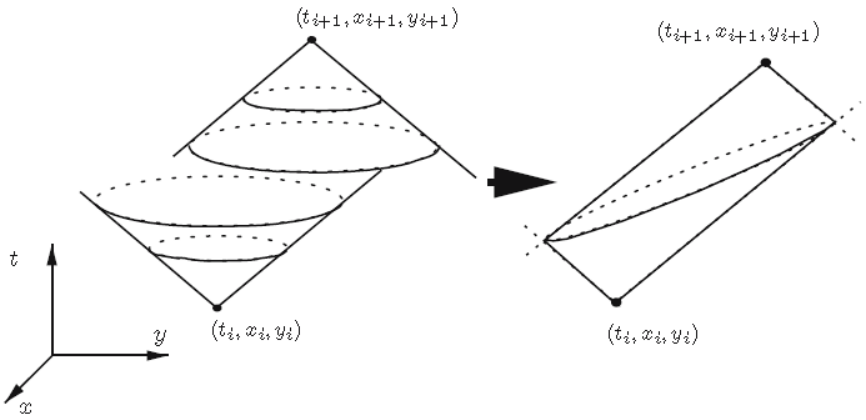


Figure 2: the bead model

After we have analyzed the uncertainty, we can talk about **trajectory classification**. In literature, “trajectory classification” is defined as the process of identifying the class of a moving object on the basis of its trajectories. The applications of this process go from pattern recognition, to bioengineering, to video surveillance[10]. In this work, we are using trajectory classification to understand the behavior patterns of persons that produce trajectory visiting a number of points of interest. A lot of trajectory classification methods have been proposed, but the bulk refers to the Hidden Markov Model [9]: to do the classification, HMM refers to the shape of the whole trajectory and creates a mathematical function that models the trajectory. The basic formulation of the problem is given by maximization of a conditional probability that tries to assign a class to the whole trajectory: the problem is that we lose the facets of the single parts of a trajectory. Instead, in our model we consider a segmented trajectory, where each segment has two stops as endpoints: the total probability for the class to assign to the trajectory is given by the maximization of the sum of probabilities of all segment of the trajectory.

A framework for trajectory classification has been proposed in [8]: “*TraClass*” is a framework used for those application in which discriminative features of classes are not relevant to the shapes of trajectories. An example of application is the “vessel-trajectory classification”. In [8] there is the concept of “discriminative feature” that appear not only as common movement patterns, but also as regions and subtrajectories: thus, after trajectory partitioning, the framework creates hierarchical region-based and trajectory-based clustering, to discover regions and subtrajectories that indicate common movement patters of each class we are considering for the classification. This technique is very sophisticated and can be used after the application of our algorithm to discover behavior patterns from trajectories.