Code Developed

Michael Friederich
École Polytechnique Fédérale de Lausanne
Transp-OR laboratory

Contents

Listings **2**

1 Statistics on the available data **3**

2 Building the dataset - data processing **5**
2.1 Identification of the trip purpose . . . 5
2.2 Missing train trips in the public transport alternative . . . 6
2.3 Trip detection . . . 9
2.4 Detection of the travel modes . . . 12
2.5 Determination of the chosen mode . . . 13
2.6 Detection of stops/activities . . . 15
2.7 Missing travel mode information for parts of the trips . . . 23
2.8 Attributes of unchosen alternatives . . . 24

List of Figures

1 Definition of the Lausanne agglomeration . . . 8

Listings

1 Total number of GPS records. . . . 3
2 Wi-Fi records and their position. . . . 3
3 Number of Wi-Fi records that are localized. . . . 3
4 GPS accuracy SQL code. . . . 4
5 Centroids and size of the clusters Home and Work. . . . 5
6 Origins and destinations traveled by bus or metro. . . . 7
7 Number of users having home and work locations inside the Lausanne agglomeration. 7
8 Wi-Fi records assigned with the meaning home and work. . . . 9
9 Home and Work trips. . . . 9
10 Cleaning of non consistent home to work trips. . . . 10
11 Map-matched trips inside meaningful time windows. . . . 13
12 Fixing ambiguities on sequences of modes. . . . 13
13 Determination of the chosen mode . . . 14
14 GPS and Wi-Fi records inside time windows. . . . 15
15 Trip/activity detection algorithm. . . . 15
16 Stops for all the trips. . . . 18
17 Number of stops and total duration for each stop. . . . 19
18 time recording in each state during stops. . . . 22
19 *ratio_mode_time* computed and added. . . . 23
20 Attributes of the public transport alternative . . . 24

In this paper we present the code developed for Friederich et al. (2014). Each code will be introduced in the corresponding section with respect to the outline of the technical report.

1 Statistics on the available data

The number of GPS records recorded during the campaign is obtained with the following SQL code:

```
SELECT count(*)
FROM GPS;
```

Listing 1: Total number of GPS records.

The total number of WLAN access points can be computed with the same SQL code on the *wlanrelation* table.

An important view for the project assigns to every record of the *wlanrelation* table its positional coordinates stored in the *wnetworks* table when they are available:

```
CREATE view mm_od_most_lilely_merged_wifi
as
SELECT wl.user_id
  ,wl.id
  ,wl.time_stamp
  ,wl.networkid
  ,wn.longitude
  ,wn.latitude
  ,wn.geom
  ,wn.geog
FROM wnetworks wn
  ,wlanrelation wl
WHERE wn.id = wl.networkid;
```

Listing 2: Wi-Fi records and their position.

Then it is possible from this table to see how many records are localized, which corresponds to the 34 millions of Wi-Fi records localized in the technical report (see Listing 3).

```
/* All the WIFI records that are localized */
SELECT count(*)
FROM mm_od_most_lilely_merged_wifi
WHERE longitude IS NOT NULL

/* The GPS table stores all the WLAN access points in a radius of 100m of at
    least one GPS point */
SELECT count(*)
FROM mm_od_most_lilely_merged_wifi
WHERE networkid IN (
    SELECT DISTINCT (nearest_wifi)
    FROM gps
    );
```

Listing 3: Number of Wi-Fi records that are localized.

For the histogram on GPS accuracy, as 14 millions records had to be split into four categories, we first used the following SQL code

```
SELECT cat
  ,count(*)
FROM (
  SELECT horizontal_accuracy
    ,(
      CASE
        WHEN horizontal_accuracy >= 0
          AND horizontal_accuracy < 100
          THEN 1
        WHEN horizontal_accuracy >= 100
          AND horizontal_accuracy < 200
          THEN 2
        WHEN horizontal_accuracy >= 200
          AND horizontal_accuracy < 300
          THEN 3
        WHEN horizontal_accuracy >= 300
          AND horizontal_accuracy < 400
          THEN 4
        WHEN horizontal_accuracy >= 400
          AND horizontal_accuracy < 500
          THEN 5
        WHEN horizontal_accuracy >= 500
          AND horizontal_accuracy < 600
          THEN 6
        WHEN horizontal_accuracy >= 600
          THEN 7
        END
      ) cat
  FROM gps
  ) t1
GROUP BY cat;
```

Listing 4: GPS accuracy SQL code.

Then we use Matlab to plot the histogram.

All the statistics on the socio-economics of the participants are easily inferred from the user_profiles table of the dataset.

2 Building the dataset - data processing

2.1 Identification of the trip purpose

In this section we are using the clusters found by Buisson (2013).

Then, a script was coded in Matlab to compute automatically the centroids and the size of the clusters with the k-means clustering (see Listing 5).

The distance between two points is computed with the Haversine formula:
For any two points on a sphere, the haversine is

$$\text{haversine}\left(\frac{d}{r}\right) = \text{haversin}(\phi_2 - \phi_1) + \cos(\phi_1)\cos(\phi_2)\,\text{haversin}(\lambda_2 - \lambda_1) \tag{1}$$

where d is the distance between the two points
r is the radius of the sphere ($radius_{earth} = 6371km$)
ϕ_1, ϕ_2 the latitude of point 1 and latitude of point 2
λ_1, λ_2 the longitude of point 1 and longitude of point 2
$\text{haversin}(\theta) = \sin^2\left(\frac{\theta}{2}\right) = \frac{1-\cos(\theta)}{2}$

and the distance is:

$$d = 2r\arctan\left(\sqrt{\sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right) \tag{2}$$

The Matlab script *lldistkm* that applies this formula is used for our function (Sohrabinia)

```
function POI= Centroids(X,poi_nb)

% Centroids:
% Computes the centroids and size of the clusters with the K-means clustering
%
% --Inputs:
%   X= ['user_id' 'cluster_meaning' 'latitude' 'longitude']
% cluster_meaning can be any number from 1 to poi_nb
% poi_nb= the number of different cluster_meaning for each user.
%
% --Outputs:
%   Poi= ['user_id' 'cluster_meaning' 'latitude' 'longitude' 'size']
% size is the radius of the cluster in meters.
%
% --Example: in our case for home and work locations poi_nb=2
%
% First version: 13 March 2014
% Updated: none
%
% Author: Michael Friederich, EPFL.
%-------------------------------------------------------------------------

Centroid=[];
for j=1:length(unique(X(:, 1) ))
    for i=1:poi_nb
        user=unique(X(:,1)) ;
```

```
        test= X(X(:,1)==user(j)& X(:,2)==i,[3,4]);
        if size (test,1)==1
            C(i,: )=test; Acc(i,1)=0;
        else
            [Idx,C(i,:)]=kmeans(test,1);
            for k=1 :size (test,1)
                Acc2(k,: )=lldistkm( test(k,:),C(i,:))*1000;
            end
            Acc(i,:)=max(Acc2);
            clearvars Acc2;
        end
        Point(i,:)=[user(j) i C(i, :) Acc(i) ];
    end
    Centroid=[Centroid; Point];
end
POI=Centroid
clearvars Acc C Idx i j k test user Centroid Point;

function d1km =lldistkm(latlon1,latlon2)

% lldistkm:
% d1km: computes the distance in km based on Haversine formula
% (Haversine: http://en.wikipedia.org/wiki/Haversine_formula)
%
%
% --Inputs:
%   latlon1: latlon of origin point [lat lon]
%   latlon2: latlon of destination point [lat lon]
%
% --Outputs:
%   d1km: distance calculated by Haversine formula
%
% First version: 15 Jan 2012
% Updated: 17 June 2012
%
% Author: M. Sohrabinia, University of canterbury.
%-------------------------------------------------------------------------

radius=6371;
lat1=latlon1(1)*pi/180;
lat2=latlon2(1)*pi/180;
lon1=latlon1(2)*pi/180;
lon2=latlon2(2)*pi/180;
deltaLat=lat2-lat1;
deltaLon=lon2-lon1;
a=sin((deltaLat)/2)^2 + cos(lat1)*cos(lat2) * sin(deltaLon/2)^2;
c=2*atan2(sqrt(a),sqrt(1-a));
d1km=radius*c;    %Haversine distance
```

Listing 5: Centroids and size of the clusters Home and Work.

The results of the Matlab script *Centroids* are stored in the PostgreSQL table: *mm_centroids_homeandwork_locations*

2.2 Missing train trips in the public transport alternative

To plot the origins and destinations for all the arcs traveled by bus or metro, the following SQL query was executed in QGIS Lisboa:

```
SELECT st_x(from_node_geom)
  ,st_y(from_node_geom)
  ,st_x(to_node_geom)
  ,st_y(to_node_geom)
FROM mm_seqs_mode
WHERE mode IN (
    'bus'
    ,'metro'
    )
```

Listing 6: Origins and destinations traveled by bus or metro.

To plot home and Work locations, centroids from the *mm_centroids_homeandwork_locations* were queried. Then spatial visualization of the results in QGIS Lisboa is used to find the number of full time workers that are both living and working in the same agglomeration. For the Lausanne agglomeration where there is 27 participants under these conditions, it is easier to use the following SQL code rather than the spatial visualization:

```
SELECT count(*)
FROM (
  SELECT user_id
    ,count(*)
  FROM mm_centroids_homeandwork_locations
  WHERE longitude < 6.8
    AND longitude > 6.5
    AND latitude < 46.6
    AND latitude > 46.45
  GROUP BY user_id
  ) t1
WHERE count >= 2
```

Listing 7: Number of users having home and work locations inside the Lausanne agglomeration.

With the boundaries imposed in the SQL code, the Lausanne agglomeration is defined as shown below:

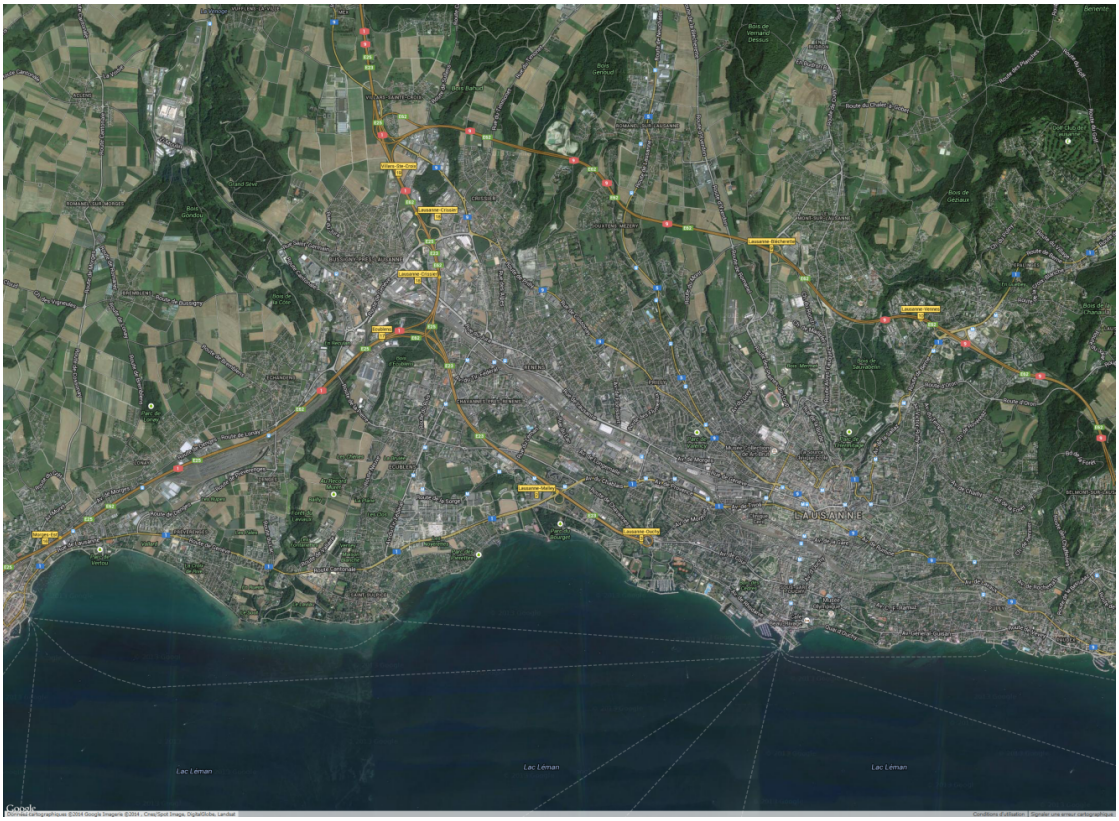

Figure 1: Definition of the Lausanne agglomeration

2.3 Trip detection

1. **Identify home and work records**:

```
/* This view contains all the WI-FI records localized for the 26 full
    time workers of our dataset on weekdays */

CREATE
  OR REPLACE VIEW PUBLIC.mm_wifi_poi_homeandwork_locations AS

SELECT w.user_id
  ,w.time_stamp
  ,w.longitude
  ,w.latitude
  ,st_distance(GEOGRAPHY (st_makepoint(c.longitude, c.latitude)), w.geog
      ) AS distance
  ,c.cluster_meaning
FROM mm_od_most_lilely_merged_wifi w
  ,mm_centroids_homeandwork_locations c
WHERE w.user_id = c.user_id
  AND w.latitude IS NOT NULL
  AND (c.user_id = ANY (ARRAY[5943,5944,5949,5950,5957,5959,5963,5974,59
      79,5980,6001,6002,6007,6010,6039,6040,6069,6075,6077,6090,6103,610
      4,6109,6170,6171,6198]))
  AND st_distance(GEOGRAPHY (st_makepoint(c.longitude, c.latitude)), w.
      geog) <= c.accuracy
  AND date_part('dow'::TEXT, w.time_stamp) >= 1::DOUBLE PRECISION
  AND date_part('dow'::TEXT, w.time_stamp) <= 5::DOUBLE PRECISION
ORDER BY w.user_id
  ,w.time_stamp;
```

Listing 8: Wi-Fi records assigned with the meaning home and work.

2. **Extract time windows**:

   The following SQL code is used to extract time windows of the trips:

```
CREATE
  OR REPLACE VIEW PUBLIC.mm_wifi_trips_homeandwork_locations AS

SELECT row_number() OVER (
    ORDER BY t4.user_id
      ,t4.departure_time
    ) AS trip_id
  ,t4.user_id
  ,t4.departure_time
  ,t4.arrival_time
  ,t4.arrival_time - t4.departure_time AS travel_time
  ,t4.from_location
  ,t4.to_location
FROM (
  SELECT t3.user_id
    ,t3.time_stamp AS departure_time
    ,t3.cluster_meaning AS from_location
    ,lead(t3.time_stamp) OVER (
      PARTITION BY t3.user_id ORDER BY t3.time_stamp
      ) AS arrival_time
    ,lead(t3.cluster_meaning) OVER (
      PARTITION BY t3.user_id ORDER BY t3.time_stamp
      ) AS to_location
```

```
    ,t3.is_start
  FROM (
    SELECT t2.user_id
      ,t2.time_stamp
      ,t2.cluster_meaning
      ,t2.previous_cluster_meaning
      ,t2.next_cluster_meaning
      ,t2.is_end
      ,t2.is_start
    FROM (
      SELECT t.user_id
        ,t.time_stamp
        ,t.cluster_meaning
        ,t.previous_cluster_meaning
        ,t.next_cluster_meaning
        ,CASE
          WHEN t.cluster_meaning != t.previous_cluster_meaning
            THEN 1
          ELSE 0
          END AS is_end
        ,CASE
          WHEN t.cluster_meaning != t.next_cluster_meaning
            THEN 1
          ELSE 0
          END AS is_start
      FROM (
        SELECT w.user_id
          ,w.time_stamp
          ,w.cluster_meaning
          ,lag(w.cluster_meaning) OVER (
            PARTITION BY w.user_id ORDER BY w.time_stamp
            ) AS previous_cluster_meaning
          ,lead(w.cluster_meaning) OVER (
            PARTITION BY w.user_id ORDER BY w.time_stamp
            ) AS next_cluster_meaning
        FROM mm_wifi_poi_homeandwork_locations w
        ORDER BY w.user_id
          ,w.time_stamp
        ) t
      ORDER BY t.user_id
        ,t.time_stamp
      ) t2
    WHERE t2.is_end = 1
      OR t2.is_start = 1
    ORDER BY t2.user_id
      ,t2.time_stamp
    ) t3
  ORDER BY t3.user_id
    ,t3.time_stamp
  ) t4
  AND t4.is_start = 1
ORDER BY t4.user_id
  ,t4.arrival_time - t4.departure_time DESC;
```

Listing 9: Home and Work trips.

3. **Clean non consistent trips**: The following trips are removed from the dataset:

```
CREATE
  OR replace VIEW mm_wifi_trips_homeandwork_locations_clean
```

```
SELECT *
FROM mm_wifi_trips_homeandwork_locations w
/* Cleaning when departure_time and arrival_time are not the same day */
WHERE DATE (w.departure_time) = DATE (w.arrival_time)
  /* Cleaning when travel_time is longer than 2h */
  AND w.travel_time < '02:00:00'::interval
/* Cleaning when there are more than 6 home to work trips in a day (
    These trips were removed manually) */

EXCEPT

(
  (
    (
      (
        (
          (
            SELECT w.trip_id
              ,w.user_id
              ,w.departure_time
              ,w.arrival_time
              ,w.travel_time
              ,w.from_location
              ,w.to_location
            FROM mm_wifi_trips_homeandwork_locations w
            WHERE w.user_id = 5944
              AND DATE (w.departure_time) = '2010-06-14'::DATE

            UNION

            SELECT w.trip_id
              ,w.user_id
              ,w.departure_time
              ,w.arrival_time
              ,w.travel_time
              ,w.from_location
              ,w.to_location
            FROM mm_wifi_trips_homeandwork_locations w
            WHERE w.user_id = 5944
              AND DATE (w.departure_time) = '2010-06-21'::DATE
            )

          UNION

          SELECT w.trip_id
            ,w.user_id
            ,w.departure_time
            ,w.arrival_time
            ,w.travel_time
            ,w.from_location
            ,w.to_location
          FROM mm_wifi_trips_homeandwork_locations w
          WHERE w.user_id = 5944
            AND DATE (w.departure_time) = '2010-06-10'::DATE
          )

        UNION

        SELECT w.trip_id
          ,w.user_id
          ,w.departure_time
```

```
        ,w.arrival_time
        ,w.travel_time
        ,w.from_location
        ,w.to_location
      FROM mm_wifi_trips_homeandwork_locations w
      WHERE w.user_id = 6001
        AND DATE (w.departure_time) = '2010-01-18'::DATE
      )

    UNION

    SELECT w.trip_id
      ,w.user_id
      ,w.departure_time
      ,w.arrival_time
      ,w.travel_time
      ,w.from_location
      ,w.to_location
    FROM mm_wifi_trips_homeandwork_locations w
    WHERE w.user_id = 5944
      AND DATE (w.departure_time) = '2010-06-02'::DATE
    )

  UNION

  SELECT w.trip_id
    ,w.user_id
    ,w.departure_time
    ,w.arrival_time
    ,w.travel_time
    ,w.from_location
    ,w.to_location
  FROM mm_wifi_trips_homeandwork_locations w
  WHERE w.user_id = 5944
    AND DATE (w.departure_time) = '2010-06-22'::DATE
  )

UNION

SELECT w.trip_id
  ,w.user_id
  ,w.departure_time
  ,w.arrival_time
  ,w.travel_time
  ,w.from_location
  ,w.to_location
FROM mm_wifi_trips_homeandwork_locations w
WHERE w.user_id = 6001
  AND DATE (w.departure_time) = '2010-01-11'::DATE
);
```

Listing 10: Cleaning of non consistent home to work trips.

2.4 Detection of the travel modes

In listing 11, we include the map-matched trips inside the time windows inferred from section 2.3.

```
CREATE TABLE mm_wifi_trips_homeandwork_with_mm AS

SELECT (w.trip_id) AS trip_id
  ,(m.trip_id) AS mm_trip_id
  ,m.seq_id
  ,w.user_id
  ,w.from_location
  ,w.to_location
  ,w.departure_time
  ,m.begin_time
  ,m.mode
  ,m.end_time
  ,w.arrival_time
  ,round((abs(date_part('hour'::TEXT, m.end_time - m.begin_time)) * 60::
      DOUBLE PRECISION + abs(date_part('minute'::TEXT, m.end_time - m.
      begin_time)) / ((abs(date_part('hour'::TEXT, w.arrival_time - w.
      departure_time)) * 60::DOUBLE PRECISION + abs(date_part('minute'::
      TEXT, w.arrival_time - w.departure_time)))))::NUMERIC, 3) AS
      ratio_mode_time
  ,w.travel_time
  ,(m.from_node_geom) AS from_node
  ,(m.to_node_geom) AS to_node

FROM mm_wifi_trips_homeandwork_locations_clean_lessthan2h_table w
  ,mm_seqs_mode m
WHERE m.begin_time > w.departure_time
  AND m.end_time < w.arrival_time
  AND m.user_id = w.user_id
ORDER BY w.user_id
  ,w.trip_id
  ,m.trip_id
  ,m.seq_id
```

Listing 11: Map-matched trips inside meaningful time windows.

2.5 Determination of the chosen mode

The cleaning of the ambiguous sequences (e.g. car and bus changing over time) have been removed as shown in listing 12.

```
/* sequences of bus and car changing over the time*/
SELECT user_id
  ,trip_id
  ,count(*)
FROM mm_wifi_trips_homeandwork_locations_mode_shares
WHERE mode IN (
    'car'
    ,'bus'
    )
GROUP BY user_id
  ,trip_id
HAVING count(*) >= 2
ORDER BY user_id
```

Listing 12: Fixing ambiguities on sequences of modes.

Finally, as explained in the technical report, the chosen mode was identified for all the trips as shown in Listing 13.

```
CREATE TABLE mm_wifi_trips_homeandwork_locations_main_mode AS

SELECT share.user_id
  ,share.trip_id
  ,(share.mode) AS main_mode
  ,agg.length
  ,agg.ratio_mode_time
  ,agg.from_location
  ,agg.to_location
  ,agg.departure_time
  ,agg.arrival_time
  ,agg.travel_time
FROM (
  SELECT *
  FROM (
    SELECT *
      ,row_number() OVER (
        PARTITION BY user_id
        ,trip_id ORDER BY user_id
          ,trip_id
        ) AS row
    FROM mm_wifi_trips_homeandwork_locations_mode_shares
    ) t
  WHERE row = 1
  ) share
  ,(
    SELECT agg.*
      ,m.from_location
      ,m.to_location
      ,m.departure_time
      ,m.arrival_time
      ,m.travel_time
    FROM (
      SELECT user_id
        ,trip_id
        ,sum(length) AS length
        ,sum(DISTINCT (ratio_mode_time)) AS ratio_mode_time
      FROM mm_wifi_trips_homeandwork_locations_clean_lessthan2h_with_mm
          m
      GROUP BY user_id
        ,trip_id
      ORDER BY user_id
        ,trip_id
      ) agg
      ,mm_wifi_trips_homeandwork_locations_clean_lessthan2h_with_mm m
    WHERE agg.trip_id = m.trip_id
    ORDER BY agg.user_id
      ,agg.trip_id
    ) agg
WHERE share.trip_id = agg.trip_id
GROUP BY share.user_id
  ,share.trip_id
  ,share.mode
  ,agg.length
  ,agg.ratio_mode_time
  ,agg.from_location
  ,agg.to_location
  ,agg.departure_time
  ,agg.arrival_time
  ,agg.travel_time
```

Listing 13: Determination of the chosen mode

2.6 Detection of stops/activities

The code used as well as the coding language employed is explained step by step according to the processing description of the technical report:

(a) **Extract available records**: All GPS and Wi-Fi records available inside the time windows of our 700 trips are extracted with the SQL code shown in Listing 14.

```
CREATE TABLE mm_gpsandwifi_home2work_records AS

SELECT *
FROM (
  SELECT w.user_id
    ,m.trip_id
    ,w.time_stamp
    ,w.latitude
    ,w.longitude
  FROM gps w
    ,mm_wifi_trips_homeandwork_locations_main_mode m
  WHERE w.user_id = m.user_id
    AND w.time_stamp <= m.arrival_time
    AND w.time_stamp >= m.departure_time
    AND latitude notnull

  UNION ALL

  SELECT w.user_id
    ,m.trip_id
    ,w.time_stamp
    ,w.latitude
    ,w.longitude
  FROM mm_od_most_lilely_merged_wifi w
    ,mm_wifi_trips_homeandwork_locations_main_mode m
  WHERE w.user_id = m.user_id
    AND w.time_stamp <= m.arrival_time
    AND w.time_stamp >= m.departure_time
    AND latitude notnull
  ) t
ORDER BY trip_id
  ,time_stamp
```

Listing 14: GPS and Wi-Fi records inside time windows.

(b) **Apply the activity/stop detection algorithm**: The following algorithm, based on the Density-Based spatial clustering proposed by Ester et al. (1996) and proposed in Matlab code by Daszykowski et al. (2001), has been modified to take into account the time dimension in the clustering according to the code in Listing 15. The results of this final script are stored in the PostgreSQL server as *mm_gpsandwifi_stopsfound_from_matlab*.

```
function [class,type]=dbscan_with_time(x,k,Eps,delta)

% Function: [class,type]=dbscan(x,k,Eps,delta)
```

```
%
% Aim:
% Clustering the data with a modified Density-Based Scan Algorithm
    with
% Noise (DBSCAN) which takes into account both space and time
    parameters
%
% Input:
% x - data set (m,n); m-objects, n-variables
% k - number of objects in a neighborhood of an object
% (minimal number of objects considered as a cluster).
% Eps - neighborhood radius (in meters).
% Delta - neighborhood time (in minutes).
%
% Output:
% class - vector specifying assignment of the i-th object to certain
% cluster (m,1)
% type - vector specifying type of the i-th object
% (core: 1, border: 0, outlier: -1)
%
% Original DB Scan clustering written by Michal Daszykowski
% Department of Chemometrics, Institute of Chemistry,
% The University of Silesia
% December 2004
%
% Modified by Michael Friederich
% Civil Engineering department, Transp OR laboratory,
% Last version: 12 april 2014
%
    ------------------------------------------------------------------------



xa=x(:,3);
x=x(:,4:5);
[m,n]=size(x);

x=[[1:m]' x];
[m,n]=size(x);
type=zeros(1,m);
no=1;
touched=zeros(m,1);

for i=1:m
    if touched(i)==0;
        ob=x(i,:);
        ob2=xa(i,:);
        D=dist(ob(2:n),x(:,2:n));
        A=time(ob2,xa);
        ind=find(A>=delta & D<=Eps);
        if isempty(ind)
            ind=1;
        end

        if length(ind)>1 & length(ind)<k+1
            type(i)=0;
            class(i)=0;
        end
        if length(ind)==1
            type(i)=-1;
            class(i)=-1;
            touched(i)=1;
        end
```

```
        if length(ind)>=k+1;
            type(i)=1;
            class(ind)=ones(length(ind),1)*max(no);

            while ~isempty(ind)
                ob=x(ind(1),:);
                ob2=xa(ind(1),:);
                touched(ind(1))=1;
                ind(1)=[];
                D=dist(ob(2:n),x(:,2:n));
                A=time(ob2,xa);
                i1=find(A>=delta & D<=Eps);
                if isempty(i1)
                    i1=1;
                end


                if length(i1)>1
                    class(i1)=no;
                    if length(i1)>=k+1;
                        type(ob(1))=1;
                    else
                        type(ob(1))=0;
                    end

                    for i=1:length(i1)
                        if touched(i1(i))==0
                            touched(i1(i))=1;
                            ind=[ind i1(i)];
                            class(i1(i))=no;
                        end
                    end
                end
            end
            no=no+1;
        end
    end
end

if length(type)~=length(class),
    class(1,m)=0;
end
i1=find(class==0);
class(i1)=-1;
type(i1)=-1;


function [D]=dist(i,x)

% function: [D]=dist(i,x)
%
% Aim:
% Calculates the distance (in m) based on Haversine formula
%between the i-th object and all objects in x
%
% Input:
% i - an object (1,n)
% x - data matrix (m,n); m-objects, 1-variables
% x=[latitude longitude]
%
% Output:
```

```
% D - haversine distance(m,1)
%
    ------------------------------------------------------------------------


[m,n]=size(x);
D=zeros(1,m);
for j=1:m
    D(1,j)=lldistkm(i,x(j,1:2))*1000;
end

function [A]=time(i,xa)

% function: [A]=time(i,xa)
%
% Aim:
% Calculates the time (in min) between the i-th object and all
    objects in x
%
% Input:
% i - an object (1,n)
% xa - data matrix (m,n); m-objects, 1-variables
% xa=[{time_stamp}]
%
% Output:
% D - duration (m,1)
%
    ------------------------------------------------------------------------


[m,n]=size(xa);
A=zeros(1,m);
for j=1:m
    A(1,j)= (abs(datenum(i)-datenum(xa(j,1))))/(1.15741277113557e-05))
        /60;
end
```

Listing 15: Trip/activity detection algorithm.

In order to launch the algorithm for each trip, another script was written in Matlab as shown in Listing 16.

```
function Activities =stops(X,k,Eps,delta)

% Function: Activities =stops(X,k,Eps,delta)
%
% Aim:
% This algorithm launches the Dbscan_with_time algorithm for each
    trip
% and stores the results in the Activities Matrix.
%
% Input:
% x - Records data set (m,n) matrix format ;
%    [user_id tip_id time_stamp latitude longitude]
% k - number of objects in a neighborhood of an object
% (minimal number of objects considered as a cluster)
% Eps - neighborhood radius (in meters).
% Delta - neighborhood time (in minutes).
%
% Output:
% Activities: All the stops for the trips inputed.
```

```
% [user_id stop_nb latitude longitude radius trip_id duration]
% radius is in meters
% duration estimated is in minutes
%
% Author: Michael Friederich
% Civil Engineering department, Transp OR laboratory,
% Last version: 15 april 2014
%
    ____________________________________________________________________


Activities=[];
trip=unique(X(:,2));
[m,n]=size(trip);

for j=1:m
    obj=X(X(:,2)==trip(j),:);
    [class,type]=dbscan_with_time(obj,k,Eps,delta);
    if isempty(find(type==1)),
        Activities=Activities;
    else
        nb_stops= length(unique(class(find(type==1))));
        obj2=[];

        for l=1:nb_stops
            obj3=obj(type~=-1 & class==l,:);
            obj3(:,2)=l;
            if ~isempty(obj3),
                time(l,1)=(abs((obj3(end,3)-obj3(1,3)))/
                (1.15741277113557e-05))/60;
                else time=-1;
            end
            obj3(:,3)=[];
            obj2=[obj2;obj3];
        end

        centroid=Centroids(obj2,nb_stops);
        centroid(:,6)=trip(j);
        centroid(:,7)=time(:,1);
        clearvars time;
        Activities=[Activities; centroid];

    end
end
```

Listing 16: Stops for all the trips.

(c) **Centroid and radius of the stops**: The centroids are computed within the script *stops* as shown in Listing 16.

(d) **Duration of the stops**: The same method that was used in section 2.3 is applied for the stops. With this methodology, it is possible to see when a stop is seen more than one time. The SQL code presented in Listing 17 was used. Finally we only keep the stops longer than 3 minutes.

```
/* All the records inside a stop cluster are extracted*/

CREATE TABLE mm_gpsandwifi_records_with_stopsfound AS

SELECT s.user_id
  ,s.trip_id
```

```
  ,s.stop_nb
  ,r.time_stamp
  ,s.latitude
  ,s.longitude
  ,s.accuracy
  ,st_distance(GEOGRAPHY (st_makepoint(s.longitude, s.latitude)),
      GEOGRAPHY (st_makepoint(r.longitude, r.latitude))) AS distance
FROM mm_gpsandwifi_stopsfound_from_matlab s
  ,mm_gpsandwifi_home2work_records r
WHERE st_distance(GEOGRAPHY (st_makepoint(s.longitude, s.latitude)),
    GEOGRAPHY (st_makepoint(r.longitude, r.latitude))) <= s.accuracy
  AND s.user_id = r.user_id
  AND s.trip_id = r.trip_id
ORDER BY user_id
  ,trip_id

/* Time windows of the stops are extracted.*/

SELECT user_id
  ,trip_id
  ,stop_nb
  ,stop_begins
  ,stop_ends
  ,(stop_ends - stop_begins) AS duration
  ,latitude
  ,longitude
FROM (
  SELECT *
    ,row_number() OVER (
      PARTITION BY user_id
      ,trip_id
      ,stop_nb ORDER BY stop_begins
      ) AS row_nb
  FROM (
    SELECT user_id
      ,trip_id
      ,stop_nb
      ,time_stamp AS stop_begins
      ,lead(time_stamp) OVER (
        PARTITION BY user_id
        ,trip_id ORDER BY time_stamp
        ) AS stop_ends
      ,latitude
      ,longitude
    FROM (
      SELECT *
      FROM (
        SELECT user_id
          ,trip_id
          ,stop_nb
          ,time_stamp
          ,latitude
          ,longitude
          ,lag(stop_nb) OVER (
            PARTITION BY user_id
            ,trip_id ORDER BY time_stamp
            ) AS previous_stop_nb
          ,lead(stop_nb) OVER (
            PARTITION BY user_id
            ,trip_id ORDER BY time_stamp
            ) AS next_stop_nb
        FROM mm_gpsandwifi_records_with_stopsfound
```

```
        WHERE accuracy != 0
        ) t
      WHERE previous_stop_nb IS NULL
        OR next_stop_nb IS NULL
        OR stop_nb != previous_stop_nb
        OR stop_nb != next_stop_nb

      EXCEPT

      (
        SELECT *
        FROM (
          SELECT user_id
            ,trip_id
            ,stop_nb
            ,time_stamp
            ,latitude
            ,longitude
            ,lag(stop_nb) OVER (
              PARTITION BY user_id
              ,trip_id ORDER BY time_stamp
              ) AS previous_stop_nb
            ,lead(stop_nb) OVER (
              PARTITION BY user_id
              ,trip_id ORDER BY time_stamp
              ) AS next_stop_nb
          FROM mm_gpsandwifi_records_with_stopsfound
          WHERE accuracy != 0
          ) t
        WHERE previous_stop_nb = next_stop_nb
        )
      ORDER BY user_id
        ,trip_id
        ,time_stamp
      ) t2
    ) t3
  ) t4
WHERE row_nb IN (1,3,6)
  ORDER BY user_id
  ,trip_id
  ,stop_begins

/* For each home to work trip, we compute the number of stops, the
    total duration of the stops and we eliminate all the stops
    shorter than 3 minutes which we consider are not real stops .*/

CREATE TABLE
    mm_wifi_trips_homeandwork_locations_main_mode_with_stops_150m

SELECT m.*
  ,agg.total_duration AS stop_duration
  ,agg.nb_stops
FROM (
  SELECT s.user_id
    ,s.trip_id
    ,sum(s.duration) AS total_duration
    ,count(*) AS nb_stops
  FROM mm_gpsandwifi_all_stops s
  WHERE s.duration >= '00:03:00'
  GROUP BY s.user_id
    ,s.trip_id
  ) agg
```

```
FULL JOIN mm_wifi_trips_homeandwork_locations_main_mode m ON agg.
    trip_id = m.trip_id
ORDER BY m.user_id
  ,m.trip_id
  ,m.departure_time
```

Listing 17: Number of stops and total duration for each stop.

In order to obtain the pies with the shares of time recording in each state, we first computed the pie of the situation (ii) (e.g. during stops) with the SQL code provided in Listing 18.

Following the same procedure, we obtained the recording durations for the states during trips (travel periods and stops confounded), we denote this pie (iii). Finally, we deduced the pie of the situation (i) (e.g. during travel periods) by subtracting the durations of pies (iii) and (i) in order to capture the recording durations during travel periods.

```
SELECT STATE
  ,sum(duration_state) AS duration_state
FROM (
  SELECT user_id
    ,trip_id
    ,STATE
    ,time_stamp
    ,(
      CASE
        WHEN next_state IS NULL
          THEN arrival_time - time_stamp
        WHEN previous_state IS NULL
          THEN lead(time_stamp) OVER (
              PARTITION BY user_id
              ,trip_id ORDER BY time_stamp ASC
              ) - departure_time
        ELSE (
            lead(time_stamp) OVER (
              PARTITION BY user_id
              ,trip_id ORDER BY time_stamp ASC
              )
            ) - time_stamp
        END
      ) AS duration_state
  FROM (
    SELECT user_id
      ,trip_id
      ,STATE
      ,time_stamp
      ,departure_time
      ,arrival_time
      ,lead(STATE) OVER (
        PARTITION BY user_id
        ,trip_id ORDER BY time_stamp ASC
        ) AS next_state
      ,lag(STATE) OVER (
        PARTITION BY user_id
        ,trip_id ORDER BY time_stamp ASC
        ) AS previous_state
    FROM (
      SELECT m.user_id
        ,m.trip_id
        ,s.time_stamp
        ,s.STATE
```

```
        ,s.reason
        ,m.begin_stop AS departure_time
        ,m.end_stop AS arrival_time
      FROM mm_state_records s
        ,mm_gpsandwifi_stopsfound_from_matlab_150m
      WHERE s.user_id = m.user_id
        AND s.time_stamp > m.begin_stop - interval '1' minute
        AND s.time_stamp < m.end_stop
      ) t1
    ORDER BY user_id
      ,trip_id
      ,time_stamp ASC
    ) t2
  ) t3
GROUP BY STATE
ORDER BY sum(duration_state) DESC
```

Listing 18: time recording in each state during stops.

2.7 Missing travel mode information for parts of the trips

The *ratio_mode_time* defined in the technical report is added to the table *mm_wifi_trips_homeandwork_locations_main_mode_with_stops*
The SQL code is shown in Listing 19.

```
/*create a new column with the ratio_mode_time*/
ALTER TABLE mm_wifi_trips_homeandwork_locations_main_mode_with_stops_150
    m ADD COLUMN ratio_mode_time DOUBLE PRECISION;

/* compute the ratio_mode_time for the trips without stops */
UPDATE mm_wifi_trips_homeandwork_locations_main_mode_with_stops_150m w
SET ratio_mode_time =

SELECT trip_id
  ,round((abs(date_part('hour'::TEXT, sum_mm_time)) * 60::DOUBLE
      PRECISION + abs(date_part('minute'::TEXT, sum_mm_time)) / ((abs(
      date_part('hour'::TEXT, travel_time)) * 60::DOUBLE PRECISION + abs
      (date_part('minute'::TEXT, travel_time)))))::NUMERIC, 3)
FROM (
  SELECT trip_id
    ,min(travel_time) AS travel_time
    ,sum(sum_mm_time) AS sum_mm_time
  FROM (
    SELECT trip_id
      ,mm_trip_id
      ,max(travel_time) AS travel_time
      ,(max(end_time) - max(begin_time)) AS sum_mm_time
    FROM (
      SELECT w.*
        ,m.mm_trip_id
        ,m.begin_time
        ,m.end_time
      FROM mm_wifi_trips_homeandwork_locations_main_mode_with_stops_150m
          w
        ,mm_seqs_mode m
      WHERE m.begin_time > w.departure_time
        AND m.end_time < w.arrival_time
        AND m.user_id = w.user_id
```

```
        AND nb_stops = 0
      ) t1
    GROUP BY trip_id
      ,mm_trip_id
    ) t2
  GROUP BY trip_id
  ) t3
WHERE t2.trip_id = w.trip_id
  AND nb_stops IS NULL
```

Listing 19: *ratio_mode_time* computed and added.

2.8 Attributes of unchosen alternatives

The attributes of the unchosen alternative are manually imputed with Google directions site. For the soft mode and car alternative, the site gives the distance and travel time for the origin and destination wished. However for the public transport alternative, the site gives the total time but does not estimate the total distance that includes the distance walking (e.g. distance to reach the public transport station, distance for transfer, etc.) plus the distance with the public transport modes. Hence, we propose a Javascript code based on Google directions API to impute time and distance as shown in Listing 20. We use Google chrome as an interface to extract the time and distance computed.

```
<!DOCTYPE html>
<html>
<head>
   <meta http-equiv="content-type" content="text/html; charset=UTF-8"/>
   <title>Google Maps API v3 Directions Example</title>
   <script type="text/javascript"
           src="https://maps.google.com/maps/api/js?sensor=false"></
               script>
</head>
<body style="font-family: Arial; font-size: 12px;">
<div id="panel" style="float:middle;width:30%;height 100%">
<div style="float:middle;margin-middle: 20px;width: 158px;text-align:
    middle;"><h3 id="duration"></h3></div>
<div style="float:middle;width: 158px;text-align: middle;"><h3 id="total
    "></h3></div>
<div style="float:middle;width: 158px;text-align: middle;"><h3 id="mode
    "></h3></div>



   <script type="text/javascript">


     var directionsService = new google.maps.DirectionsService();
     var directionsDisplay = new google.maps.DirectionsRenderer();;

  function initialize() {

     directionsDisplay.setPanel(document.getElementById('panel'));
   google.maps.event.addListener(directionsDisplay, 'directions_changed'
       , function() {
     computeTotalDistance(directionsDisplay.directions);

  });
```

```
30
  calcRoute();
}
function calcRoute() {

35  // This is where we input the origin and destination. This is for
       example for user 5974.
    var request = {
      origin: new google.maps.LatLng(46.51544761, 6.631547754)
    ,
      destination: new google.maps.LatLng(46.51970477, 6.567332755)
40 ,
      travelMode: google.maps.DirectionsTravelMode.TRANSIT,
  // This is where we input the date and time (01/02/2010 at 09am)
    transitOptions: {departureTime: new Date('2010', '02' - 1, '01', '
        09', '00', '00')},
    unitSystem: google.maps.DirectionsUnitSystem.METRIC,
45  provideRouteAlternatives: false

    };

    directionsService.route(request, function(response, status) {
50    if (status == google.maps.DirectionsStatus.OK) {
        directionsDisplay.setDirections(response);

      }
    });
55 }

   // This is the function propose to estimate the total distance. The
       function also computes the total time.
function computeTotalDistance(result) {

60  var total = 0;
  var time= 0;
  var walk=0;
  var stops=0;
  var myroute = result.routes[0];
65


   for (var i = 0; i < myroute.legs.length; i++) {
    total += myroute.legs[i].distance.value;
    time +=myroute.legs[i].duration.value;
70    }
    for (var i = 0; i < myroute.legs[0].steps.length; i++) {

    if (myroute.legs[0].steps[i].travel_mode=="WALKING")
     {
75  walk+=myroute.legs[0].steps[i].duration.value;
    }
    else if (myroute.legs[0].steps[i].travel_mode=="TRANSIT")
      {
      stops+= myroute.legs[0].steps[i].transit.num_stops.value;
80    }

       }

  time = time / 60.
85  total = total / 1000.
  walk = walk / 60.

 document.getElementById('duration').innerHTML = "total time : " + Math.
```

```
      round(time)+" min";
    document.getElementById('total').innerHTML ="total distance : " + Math
        .round(total)+ " km" ;
    document.getElementById('duration').innerHTML ="total walk : " + Math.
        round(walk)+ " min" ;
    document.getElementById('num_stops').innerHTML ="total stop : " + Math
        .round(stops)+ " min" ;

}

google.maps.event.addDomListener(window, 'load', initialize);

</Script>

</body>
</html>
```

Listing 20: Attributes of the public transport alternative

.

References

[1] A. Buisson. Identify user's locations of interest from smartphone wifi data. Technical report, Polytechnic School of Lausanne (EPFL), 2013.

[2] M Daszykowski, B Walczak, and D.L Massart. Looking for natural patterns in data: Part 1. density-based approach. *Chemometrics and Intelligent Laboratory Systems*, 56(2):83 – 92, 2001. ISSN 0169-7439. doi: http://dx.doi.org/10.1016/S0169-7439(01)00111-3. URL http://www.sciencedirect.com/science/article/pii/S0169743901001113.

[3] M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In Evangelos Simoudis, Jiawei Han, and Usama M. Fayyad, editors, *KDD*, pages 226–231. AAAI Press, 1996. ISBN 1-57735-004-9. URL http://dblp.uni-trier.de/db/conf/kdd/kdd96.html#EsterKSX96.

[4] M. Friederich, M. Nikolic, E. Kazagli, A.A. Fernandéz, and M. Bierlaire. Mode choice models with smartphone data. Technical report, École Polytechnique Fédérale de Lausanne, 2014.

[5] M. Sohrabinia. URL http://www.mathworks.com/matlabcentral/fileexchange/38812-latlon-distance/content/lldistkm.m.