University of Zurich^UZH

Periodic Pattern Mining for Moving Objects

Hoda Allahbakhshi

Benedikt Steger

Computational Movement Analysis FS2016

Table of Contents

1. Motivation

Periodicity is a kind of movement rule that naturally exists in moving objects. Moving objects always obey more or less the same route (spatial property) over regular time intervals (temporal property) [1].

Periodicity is a frequently happening phenomenon for moving objects. Finding periodic behaviors is essential to understanding object movements. However, periodic behaviors could be complicated, involving multiple interleaving periods, partial time span, and spatiotemporal noises and outliers. Such periodic behaviors provide an insightful and concise explanation over the long moving history [2]. For example, animal movements could be summarized using several daily and yearly periodic behaviors. Periodic behaviors are also useful for compressing movement data [3][4]. Since they are summarization of movements, we can use them to replace the original data to save space. Moreover, periodic behaviors are useful in future movement prediction [5]. At the same time, if an object fails to follow regular periodic behaviors, it could be a signal of abnormal environment change or an accident [2].

The discovery of hidden and valuable periodic patterns could reveal valuable information to the data analyst. Periodic patterns extracted from spatio-temporal trajectories of moving objects unveil regular movement behavior. For example, birds have yearly migration patterns, people travel on regular routes to work, and commercial airliners operate regular schedules from one place to another [1].

In summary, Periodic Pattern Mining (PPM) can be used for discovering the intrinsic behavior of moving objects, compressing movement data [6], predicting future movements of objects [5], and detecting abnormal events. Mining periodic behaviors can bridge the gap between raw data and semantic understanding of the data [2].

2. Research questions

On one hand, as it is mentioned before, periodic patterns extracted from spatio-temporal trajectories of moving objects unveil regular movement behavior. Since routine activity reflects both the temporal and the spatial regularities of people’s daily lives, we take it as the basis to detect periodic patterns.

On the other hand, movement disorder can be seen as one of the common indicators for elder people illnesses. So, finding periodic patterns in elder people can help us to predict their future movement and also detect irregular or abnormal events with their movement functionality.

It is worth mentioning that MOASIS project data provides us this opportunity to have access to the individualized everyday-life health data in older adults and motivates us to be curious about whether elder people follow a periodic pattern or not. So, that is why we went about investigating our below research questions:

1. How to extract routine activities from raw trajectories?

2. What are typical periodic patterns of elderly people?

3. Dataset

The small mobile sensor *uTrail* is used for the data collection, assuming no prior technical knowledge by the participants. uTrail, measures the *mobility* (spatial activity) with GPS, The sampling rate (Fig. 1) reflects the scale of analysis for the GPS. We have GPS data of a sample size 2 participants during a 2 week period.

	Sensor	**Variable**	**Sampling rate**
Spatial Mobility	GPS	Timestamp, lat, lon	1/sec

Figure 1. uTrail mobile sensor used in the MOASIS study

For users 11422 and 11431, there were 1'217'643/1'086'471 data entries of which 185'658/63'742 were valid GPS fixes where at least 5 satellites were seen. For both participants the dataset begins at the 25th of November 2015 and ends 15 days afterwards. Insights into the data are provided in the results section.

4. Methodology

To answer the research questions, we went through the approach of the paper titled “Mining user similarity based on routine activities” [7] in detail and tried to implement it using MOSIS data. What follows is the summary of the paper [7].

The overall flowchart of the methodology is depicted in figure 2.

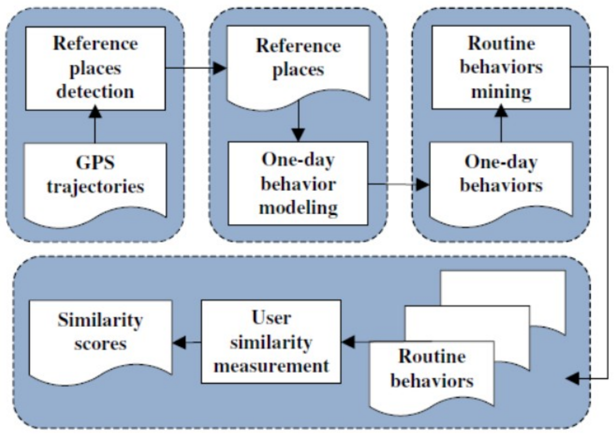


Figure 2. The system architecture [7]

In order to extract routine activities from raw trajectories, the system performs following steps for each individual user:

1. Extract reference places from their GPS trajectories by detecting visit points.
2. Transform the original GPS trajectories into 1-day activities as a daily style.
3. Extract patterns from 1-day activities based on the routine activity model.
4. Measure similarity between multiple users based on their routine activities.

4.1. Extract reference places from their GPS trajectories

To extract the reference places, firstly, visit points should be extracted. A visit point is a triple VP = (p, t_{in}, t_{out}), where p is a GPS point, t_{in} and t_{out} are timestamps, and the visit point stands for a location p around which the user stays for longer than a time threshold. A reference place is a collection of visit points. Reference places are the particular places the user regularly visits (e.g. home, work, etc.).

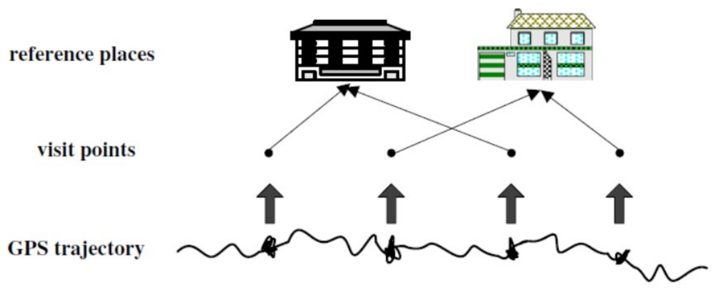
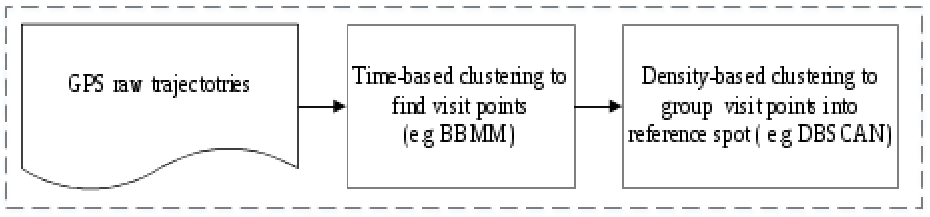


Figure 3. Reference places extraction based on hierarchical clustering [7]

The idea for extracting the reference places is to use a hierarchical clustering to benefit from the advantages of both time-based and density-based clustering approaches. To do this, we should take GPS trajectories as input and conduct a time-based clustering to identify visit points, and then adopt a density-based clustering to group these visit points into reference places.

We first decided to use Dynamic Brownian Bridge Movement Model (DBBMM) and the Density-based spatial clustering of applications with noise (DBSCAN) as a time-based and density-based clustering methods respectively. But the DBBMM has a very long runtime and does not handle the entrance and exit deviation problem [7], leading to the time-based clustering algorithm proposed in [7].



GPS raw trajectotries
Time-based clustering to find visit points (e.g BBMM)
Density-based clustering to group visit points into reference spot (e.g DBSCAN)

Figure 4. Using hierarchical clustering to extract reference places

4.2. Transform the original GPS trajectories into 1-day activities as a daily style

To transform the original GPS trajectories into 1-day activities as a daily style we need to divide a day into 24 time spans (i.e. one hour as a time span) as the columns of the place preference matrix. For each reference place visited in a particular day and each time span of that day, we check the visit points of the reference place to find how long the user stays in the reference place during the corresponding time span, and fill in the corresponding entry with the result. If the sum of staying time distributed in all reference places during a specific time span is less than an hour, we assign the remaining time to the ‘‘on the way’’ row of the corresponding time span (done in step4.php just before step4Output()). (Table 1.)

Table 1. Place preference matrices

Day 1 (~workday)	7:00-8:00	8:00-9:00	9:00-10:00	…
Reference spot 1	54 minutes	0 minutes	0 minutes	
Reference spot 2	5 minutes	55 minutes	60 minutes	
Reference spot 3	0 minutes	0 minutes	0 minutes	
…	…			
On the way	1 minute	5 minutes	0 minutes	

Day 2 (~weekend)	7:00-8:00	8:00-9:00	9:00-10:00	…
Reference spot 1	60 minutes	60 minutes	54 minutes	
Reference spot 2	0 minutes	0 minutes	0 minutes	
Reference spot 3	0	0 minutes	5 minutes	
…				
On the way	0	0 minutes	1 minute	

Day …	7:00-8:00	8:00-9:00	9:00-10:00	…
Reference spot 1	? minutes	...		
Reference spot 2	...			
Reference spot 3				
…				
On the way				

4.3. Extract patterns from 1-day activities based on the routine activity model

To extract patterns from 1-day activities based on routine activity model, first we should find groups of 1-day activities which are similar to each other.

A potential solution to find similar 1-day activities is to use clustering methods. In order to apply a clustering method, we need to define a similarity measure and choose an appropriate clustering algorithm. Since a 1-day activity is represented by a place preference matrix which can be further decomposed into 24 place preference vectors (shaded yellow in table 1), each of which stands for the distribution of time spent at each reference place during the specific time span of that day (Table 1.), we use the cosine coefficient to measure the similarity between each pair of place preference vectors (i.e. $X = (x_1, ..., x_D)$, $Y = (y_1, ..., y_D)$) as Eq. (1), in which D is the number of the total extracted reference places of the user, x_i and y_i are the time duration the user stayed at the ith reference place during the corresponding time span in two different days.

Eq 1.

$$cosine\ similarity = \frac{\sum_{i=1}^{D} x_i y_i}{\sqrt{\sum_{i=1}^{D} x_i^2}\sqrt{\sum_{i=1}^{D} y_i^2}}$$

For example, the similarity between the place preference vector (0s,2300s,0s,0s,0s,1300s) which is the time span 13:00h-14:00h of the 26th November 2015 from user 11422 and the preference vector (0s,2482s,0s,0s,0s,1118s) (next day) is 99.58%. The implementation can be found in similarityBetweenPlacePreferenceVectors() in step5.php.

The similarity between two 1-day activities is the average value of the similarity of their decomposed place preference vectors with the same time span. The result is visible in Figure 6, the implementation in similarityBetween1DayActivities() in step5.php.

Eq 2. similarity between two 1-day activities = Sum(

Cosine similarity of the place pref. vectors 0:00-1:00 of the two days +

Cosine similarity of the place pref. vectors 1:00-2:00 of the two days + … +

Cosine similarity D

) / 24

The purpose of using clustering algorithm is to partition all the 1-day activities into K groups, the activities within which are similar to each other. However, K (i.e. the number of underlying routine activities) is always unknown, so we propose a bottom-up agglomerative clustering algorithm to group the 1-day activities while determining the optimal number of clusters at the same time.

The algorithm first treats each 1-day activity as a singleton cluster and constructs a similarity matrix, where the value of the ith row and jth column is the similarity score between the ith and jth singleton clusters (Table 3).

Table 3. similarity matrix (symmetric along the diagonal)

Similarity matrix	Day 1	Day 2	Day 3	…
Day 1	1	**similarity score between the 1-day activity of Day 1 and 1-day activity of Day 2**	similarity score between the 1-day activity of Day 1 and 1-day activity of Day 3	…
Day 2	**similarity score between the 1-day activity of Day 1 and 1-day activity of Day 2**	1		
Day 3	similarity score between the 1-day activity of Day 1 and 1-day activity of Day 3		1	
…	…			1

Then, the algorithm successively merges pairs of clusters until the clustering termination condition is satisfied. At each iteration, the algorithm merges the two clusters with maximum similarity and updates the similarity matrix accordingly. The similarity of two 1-day activity clusters is calculated by averaging the similarities between each pair of 1-day activities of the two clusters (similarityBetweenMultiple1DayActivities() in step5.php).

An ideal clustering result should maximize the intra-cluster similarity (i.e. the average similarity between pairs of 1-day activities in the same cluster) as well as minimize the inter-cluster similarity (i.e. the average similarity between pairs of 1-day activities of different clusters), thus we used a variant of the Dunn index to measure the clustering quality as Eq. (3), in which s_{intra} (Ck) stands for the intra-cluster similarity of cluster Ck, s_{inter}(Ci, Cj) denotes the inter-cluster similarity of cluster Ci and Cj. A higher value of Dunn index indicates a better clustering result. To determine whether to terminate the clustering algorithm, we monitor the change of Dunn index at each iteration of the clustering procedure, and stop the algorithm if the value of Dunn index decreases dramatically or all the 1-day activities have been merged into a single cluster. A dramatic decrease of Dunn index value indicates that the decreasing degree of the minimum intra-cluster similarity significantly exceeds that of the maximum inter-cluster similarity. It is a sign that the newly merged cluster may contain 1-day activities that belong to different routine activities, and the previous clustering result is likely to be of the optimal quality.

Eq 3. Dunn index= $min\left(S_{intra}(C_k)\right) - max\left(S_{inter}(C_i, C_j)\right),$ i,j,k=1…|CS|

When the algorithm terminates, we get a collection of clusters, each of which contains a set of 1-day activities {OA_1, . . . ,$OA_{|C|}$}. The representing routine activity for a cluster is characterized by a probability distribution matrix, and each entry p(x_j = i) of the matrix represents the probability that the user is at the ith reference place during the jth time span, which can be calculated based on Eq. 4.

Eq 4.
$$P(x_j = i) = \frac{\sum_{k=1}^{|C|} OA_k \cdot e_{ij}}{T \cdot |C|}$$

According to equation 4, each cell in the probability distribution matrix is the sum of seconds of all participating days in this time slot at the cluster divided by the number of clusters (T is for normalizing between 0 and 1, 1 hour=3600 seconds, divide by 3600). Two examples are shown in Figure 8, the implementation of this equation resides in step7.php ($value=...).

Implementation stops here and we don't compare user similarities. The next chapter is an outlook how the measurement of the similarity between multiple user would be calculated.

4.4. Measure similarity between multiple users based on their routine activities

If we want to find how periodic the user routine activity is, one way could be to find user similarity based on his routine activities. To do this, we hierarchically divided the problem into three sub-problems, i.e.

1. calculating the similarity score between two reference places,
2. calculating the similarity score between two routine activities,
3. calculating the similarity score between two users.

4.4.1. Reference place similarity calculation

Since a large fraction of reference places as referred in routine activities have personal meanings, we analyze the similarity of the reference places of different routine activities based on the visiting patterns to them instead of reverse geo-coding. To achieve this goal first define visiting pattern as follow:

A visiting pattern PV_{ik} to a reference place i of a routine activity k is a probability distribution vector, each element ej (1j $\leq$ T, $\leq$ T)is the number of the discrete time spans) of which is the probability that the user is at the reference place i during the jth time span following routine activity k. Each row of a routine activity is a visiting pattern.

We calculate the similarity of places based on their visiting patterns. Since visiting pattern is a probability distribution vector, we use the Kullback–Leibler divergence as the distance measure. Given two visiting patterns PV_1 and PV_2, their Kullback–Leibler divergence can be calculated based on Eq. (5).

Eq 5.
$$KL(PV_1 \| PV_2) = \sum_{j=1}^{T} PV_1 . e_j \times \log \frac{PV_1 . e_j}{PV_2 . e_j}$$

Directly applying Kullback–Leibler divergence here has two problems so we use a smoothing parameter $\lambda(0<\lambda<1)$ and Eq. (6). $e_j = (1-\lambda) e_j + \lambda e$

Eq 6.
$$S_{place} = \log \frac{1}{(KL(PV_1 \| PV_2) + KL(PV_2 \| PV_1)) / 2}$$

4.4.2. Routine activity similarity calculation

We propose the concept of Optimal Matching Sequence (OMS) to find the most matching place pairs for similarity calculation.

Given two routine activities A_1 and A_2, and their corresponding reference places sets P_{S1} = {P_{11}, . . . , P_{1m}} and P_{S2} = {P_{21}, . . . , P_{2n}}, the Optimal Matching Sequence is OMS(A_1, A_2) = } (OP_{11}, OP_{21}), . . . , (OP_{1s}, OP_{2s}){ ($OP_{1i} \in PS_1$, $OP_{2i} \in PS_2$, , $1 \leq i \leq s$ s = min(m, n)), which maximize the Eq. 7.

Eq 7.
$$\sum_{i=1}^{s} S_{place}(OP_{1i}, OP_{2i})$$

Finally, given two routine activities A_1 and A_2, and their corresponding reference place sets PS_1 and PS_2, their similarity can be calculated based on Eq. (8), where $P_{1i} \in PS_1$, $P_{2k} \in PS_2$, OMS is the Optimal Matching Sequence of A_1 and A_2, min(A_1.e_{ij}, A_2. e_{kj}) is the common probability of reference places i and k within the jth time span, T is the number of time spans.

Eq 8.
$$S_{routine} = \frac{\sum_{j=1}^{T} \sum (P_{1i}, P_{2k}) \in OMS^{S_{place}} (P_{1i}, P_{2k}) \times min\,(A_1 . e_{ij}, A_2 . e_{kj})}{T}$$

4.4.3. User similarity calculation

Since a user may have multiple routine activities, we consider the similarity between two users according to the similarities of all their routine activities. Let AS(U_1) = {A_{11}, . . . , A_{1m}} and AS(U_2) = {A_{21}, . . . , A_{2n}} be the routine activities of users U_1 and U_2, respectively. The similarity between U_1 and U_2 is calculated by Eq. (9).

Eq 9.

$$S_{user}=\frac{\sum_{I=1}^{m}\sum_{j=1}^{n} w\left(A_{1i}, A_{2j}\right)\times S_{routine}\left(A_{1i}, A_{2j}\right)}{\sum_{i=1}^{m}\sum_{j=1}^{n} w\left(A_{1i}, A_{2j}\right)}$$

where w(A_1, A_2) stands for the weight of $s_{routine}$(A_1, A_2). The reason to use a weighted value is that different routine activities may be of different importance to the user. Obviously, the higher support of a routine activity (i.e. how many times the user follows the routine activity), the more important the routine activity is. Thus, we define the weight w(A_1, A_2) as the geometric mean of the supports to the two routine activities:

Eq 10.

$$w\left(A_{1,}A_2\right)=\sqrt{support\left(A_1\right)\cdot support\left(A_2\right)}$$

5. Results, Problems and limitations

In the first step, the raw data was read in and cleaned. This involved removing duplicates, removing rows with missing data and requiring 5 visible satellites or more in order to decrease the dilution of precision.

In the second step, the visit point extraction algorithm was implemented. The visit point extraction is essentially a segmentation preprocessing step where pseudo-movement is stripped away [9]. There were three parameters that needed to be defined:

1. The tolerated distance: how much two centroid of clusters should be apart before they are summarized as a single visit point. Here, scale level issues come up [8]. We use a coarse level of study since we need to answer how to extract periodic pattern, and not how to deal with uncertainty problems. Furthermore, users could enter a building on the front side and leave it at the back, which introduces the need to set this distance to a size (100m) where entrance and exit deviation problems can be compensated for [7].
2. The cluster distance. This parameter defines the reach of a cluster centroid towards new points. During this time-based clustering step, near points are said to belong together. Again, we chose the value 100m because all the points inside a building should belong together since we are operating on a coarse scale.
3. The time gap accepted such that two spatially coherent clusters within the tolerated distance are still constituting a visit point. Therefore, the time gap is the pendant to the tolerated distance in the time dimension. If this value is very high, we would connect clusters along a path and gradually loose the clustering behavior of this algorithm. If this value is very low, we would not represent reality since humans do not change activities at a rate in the dimension of seconds. As a compromise, the value 180s was chosen.

The clustering resulted in 193/64 clusters for users 11422/11431. Step 3 then clustered the visit points spatially with DBSCAN. DBSCAN needs the size of the ε-neighborhood. This parameter was determined using the k-nearest neighbor distances in a matrix of points and looking for the sudden increase of distances (the “knee” in kNNdistplot{dbscan}). For the two users, eps equals to 754 and 757.

Furthermore, DBSCAN takes the argument of the minimal number of points within the eps region. This number influences the clustering greatly, at least in these cases, and can cause the algorithm to find different numbers of clusters. Values around 5 proved to match the visually inspected point density best. Subsequent parameter sensibility tests showed that this parameter does not change the final conclusion since clustering does not produce periodicity.

The reference spots were then translated into 1-day activities in step 4. The 1st of December 2015 shows such a 1-day activity for the user 11422 (in UTC):

in s	**0h**	**1h**	**2h**	**3h**	**4h**	**5h**	**6h**	**7h**	**8h**	**9h**	**10h**	**11h**	**12h**	**13h**	**14h**	**15h**	**16h**	**17h**	**18h**	**19h**	**20h**	**21h**	**22h**	**23h**
refSp1																								
refSp2								262	3524	2986	2730	2888		442	3323	3598	3236	3600	3600	3600	3600	3600	3600	3600
refSp3													2610	1570										
refSp4																								
refSp5																								
On the way	3600	3600	3600	3600	3600	3600	3600	3338	76	614	870	712	990	1588	277	2	364							

“On the way” is a bucket for missing values, outliers and data points that really are between two reference spots. This introduces the first limitation of this approach: here different behaviors are summed up, decreasing the validity of the subsequent clustering steps. It is clear that missing values arise from the need to charge the sensor, from the forgetfulness of elderly people (who leave the device at home) and from GPS signal receiving conditions. The visit point extraction algorithm is powerful in dealing with those problems, but they persist.

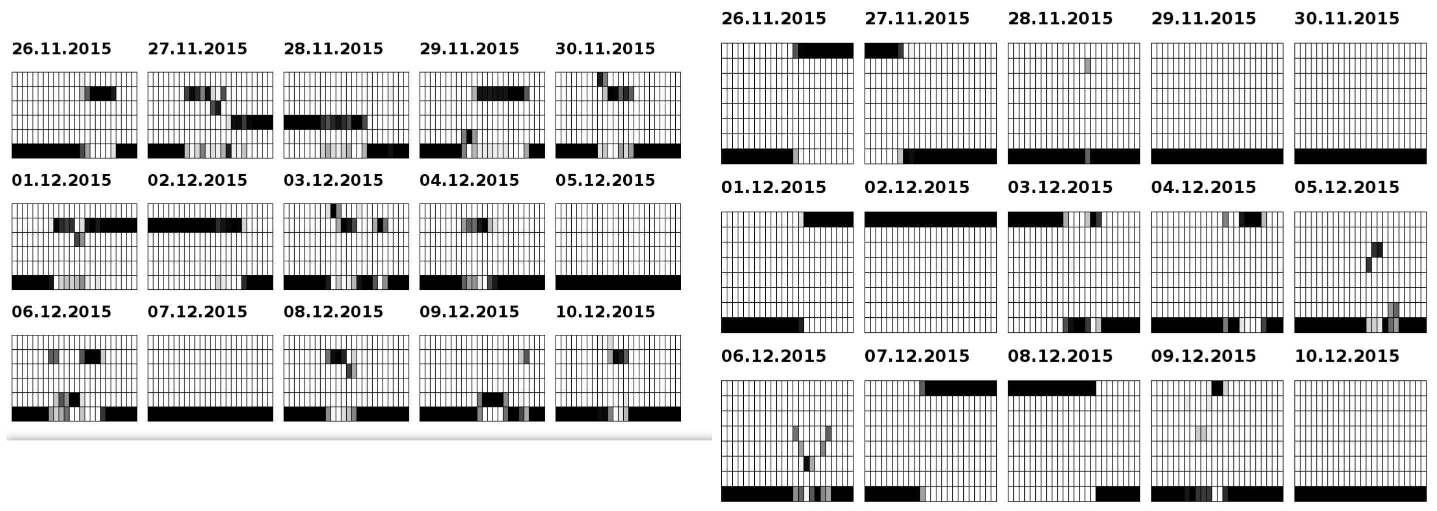


Figure 5. 1-day activities for users 11422 and 11431. Rows are reference spots, lowest row is “on the way”, columns are hours. A dark shade represents a large time span spent in the respective reference spot during the time slot.

in %	day_0	day_1	day_2	day_3	day_4	day_5	day_6	day_7	day_8	day_9	day_10	day_11	day_12	day_13	day_14
day_0	100	39.97	23.46	79.77	60.88	65.56	36.46	70.26	65.68	77.33	70.23	77.33	60.71	64.73	67.39
day_1	39.97	100	5	43.3	50.99	58.03	25.85	45.15	54.36	43.4	45.29	43.4	59.36	36.34	48.51
day_2	23.46	5	100	17.88	37.23	2.84	25.04	31.76	38.06	41.21	30.67	41.21	37.7	34.63	38.83
day_3	79.77	43.3	17.88	100	66.89	70.12	46.56	75.32	65.08	55.07	72.24	55.07	58.69	56.75	66.9
day_4	60.88	50.99	37.23	66.89	100	49.51	46.59	85.83	89.42	78.06	68.8	78.06	85.76	65.44	89.77
day_5	65.56	58.03	2.84	70.12	49.51	100	35.51	52.18	51.88	41.71	51.83	41.71	57.92	41.48	44.38
day_6	36.46	25.85	25.04	46.56	46.59	35.51	100	37.34	45.75	26.48	49.86	26.48	43.05	22.52	40.2
day_7	70.26	45.15	31.76	75.32	85.83	52.18	37.34	100	86.25	76.39	61.22	76.39	80.28	63.81	88.82
day_8	65.68	54.36	38.06	65.08	89.42	51.88	45.75	86.25	100	87.43	69.44	87.43	93.52	71.74	95.73
day_9	77.33	43.4	41.21	55.07	78.06	41.71	26.48	76.39	87.43	100	68.08	100	82.98	78.59	88.78
day_10	70.23	45.29	30.67	72.24	68.8	51.83	49.86	61.22	69.44	68.08	100	68.08	66.24	72.22	70.21
day_11	77.33	43.4	41.21	55.07	78.06	41.71	26.48	76.39	87.43	100	68.08	100	82.98	78.59	88.78
day_12	60.71	59.36	37.7	58.69	85.76	57.92	43.05	80.28	93.52	82.98	66.24	82.98	100	66.51	86.73
day_13	64.73	36.34	34.63	56.75	65.44	41.48	22.52	63.81	71.74	78.59	72.22	78.59	66.51	100	75.54
day_14	67.39	48.51	38.83	66.9	89.77	44.38	40.2	88.82	95.73	88.78	70.21	88.78	86.73	75.54	100

Figure 6. Step 5 produced the similarity matrix for user 11422. Day 0 is the 26th November 2015, day 1 the 27th etc.

in %	day_0	day_1	day_2	day_3	day_4	day_5	day_6	day_7	day_8	day_9	day_10	day_11	day_12	day_13	day_14
day_0	100	28.06	56.06	56.08	56.08	94.15	45.42	23.35	76.19	54.66	55.41	89.19	12.12	48.11	56.08
day_1	28.06	100	71.38	71.98	71.98	34.44	29.24	79.89	54.38	59.31	62.1	15.75	62.57	63.2	71.98
day_2	56.06	71.38	100	99.4	99.4	61.86	0	50.07	81.8	87.2	89.61	43.77	33.33	90.6	99.4
day_3	56.08	71.98	99.4	100	100	62.46	0	50.65	82.4	87.33	90.12	43.77	33.33	91.21	100
day_4	56.08	71.98	99.4	100	100	62.46	0	50.65	82.4	87.33	90.12	43.77	33.33	91.21	100
day_5	94.15	34.44	61.86	62.46	62.46	100	38.07	28.53	79.4	56.17	60.34	81.84	4.78	53.67	62.46
day_6	45.42	29.24	0	0	0	38.07	100	52.79	20.88	0	0	57.76	66.71	9.56	0
day_7	23.35	79.89	50.07	50.65	50.65	28.53	52.79	100	39.65	41.8	45.05	12.78	79.24	42.34	50.65
day_8	76.19	54.38	81.8	82.4	82.4	79.4	20.88	39.65	100	73.23	76.69	64.65	19.72	74.22	82.4
day_9	54.66	59.31	87.2	87.33	87.33	56.17	0	41.8	73.23	100	86.22	43.77	30.67	78.61	87.33
day_10	55.41	62.1	89.61	90.12	90.12	60.34	0	45.05	76.69	86.22	100	43.77	29.52	81.36	90.12
day_11	89.19	15.75	43.77	43.77	43.77	81.84	57.76	12.78	64.65	43.77	43.77	100	24.47	52.94	43.77
day_12	12.12	62.57	33.33	33.33	33.33	4.78	66.71	79.24	19.72	30.67	29.52	24.47	100	42.9	33.33
day_13	48.11	63.2	90.6	91.21	91.21	53.67	9.56	42.34	74.22	78.61	81.36	52.94	42.9	100	91.21
day_14	56.08	71.98	99.4	100	100	62.46	0	50.65	82.4	87.33	90.12	43.77	33.33	91.21	100

Figure 7. Step 5 produced the similarity matrix for user 11431.

Step 6 and 7 ran the algorithm 2 of [11] over the similarity matrices. This imposed a huge challenge since it produced two different results, depending on the Dunn Index treatment:

1. It clustered all single 1-day clusters into one. This happened because the "significant decrease" of the Dunn Index as well as the under-specified "clustering termination condition" lead to an implementation where there was never a significant decrease of the Dunn Index, not even if the Dunn Index is used unmodified, where a division is calculated instead of the difference between the inter-cluster and intra-cluster similarity. The cache lookups are definitively not the problem, since the original values are always used to form the new similarity between clusters.

p in %	0h	1h	2h	3h	4h	5h	6h	7h	8h	9h	10h	11h	12h	13h	14h	15h	16h	17h	18h	19h	20h	21h	22h	23h
referenceSpot_1	0	0	0	0	0	0	0	0	7.04	10.39	3.98	0	0	0.42	0.57	0	0	0	0	0	0	0	0	0
referenceSpot_2	6.67	6.67	6.67	6.67	6.67	6.67	6.67	16.31	29.7	29.99	38.81	55.99	36.86	35.99	38.72	32.61	32.01	30.15	27.79	24.45	15.86	6.67	6.67	6.67
referenceSpot_3	0	0	0	0	0	0	0	0	0	0	0	0	14.52	11.22	0	0	0	0	0	0	0	0	0	0
referenceSpot_4	6.67	6.67	6.67	6.67	6.67	6.67	6.67	5.34	4.52	5.67	6.2	5.49	4.69	6.66	6.67	5.54	6.26	6.39	5.08	6.52	6.66	6.66	6.67	6.67
referenceSpot_5	0	0	0	0	0	0	0	0	4.86	11.16	5.58	10.16	13.33	8.06	6.67	6.67	2.97	0	0	0.34	0	0	0	0
on the way	86.66	86.66	86.66	86.66	86.66	86.66	86.66	78.35	53.88	42.79	45.43	28.36	30.6	37.65	47.37	55.18	58.76	63.46	67.13	68.69	77.48	86.67	86.66	86.66

p in %	0h	1h	2h	3h	4h	5h	6h	7h	8h	9h	10h	11h	12h	13h	14h	15h	16h	17h	18h	19h	20h	21h	22h	23h
referenceSpot_1	26.67	26.67	26.67	26.67	26.67	26.67	25.18	20	20.36	20	20.14	26.67	26.67	28.86	28.87	40	37.87	33.33	33.33	33.33	28.43	26.67	26.67	26.67
referenceSpot_2	0	0	0	0	0	0	0.61	0	0	0	0.54	0	0	0	2.52	0	0	0	0	0	0	0	0	0
referenceSpot_3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5.41	5.97	0	0	0	0	0	0	0	0
referenceSpot_4	0	0	0	0	0	0	0	0	1.34	1.51	0.46	0.36	0	8.9	0	0	0	0	0.39	4.2	0	0	0	0
referenceSpot_5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2.45	0	0	0	3.22	0	0	0	0	0
referenceSpot_6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.22	6.32	2.25	0	0	0	0	0	0	0
referenceSpot_7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2.8	4.01	0	0	0	0	0
on the way	73.33	73.33	73.33	73.33	73.33	73.33	74.21	80	78.3	78.49	78.86	72.97	73.33	62.24	60.53	47.71	59.88	63.87	59.05	62.47	71.57	73.33	73.33	73.33

Figure 8. Routine activities for users 11422/11431 using summarized clusters into one.

2. It clustered only day 9/11 of user 11422 and day 3/4/14 of user 11431, leaving the other 1-day activities as periodic patterns, which is not useful. This variant was achieved by introducing the activation of the termination condition whenever the Dunn Index increased. In such a case, a merge of two clusters actually decreases the intra-cluster similarity and increases the inter-cluster similarity, motivating the termination of the algorithm.

Subsequent analysis of a very similar foreign algorithm running with the precomputed similarity matrix computing the agglomerative hierachical clustering of the dataset (agnes{cluster}) showed that there are no clusters in the data we chose:

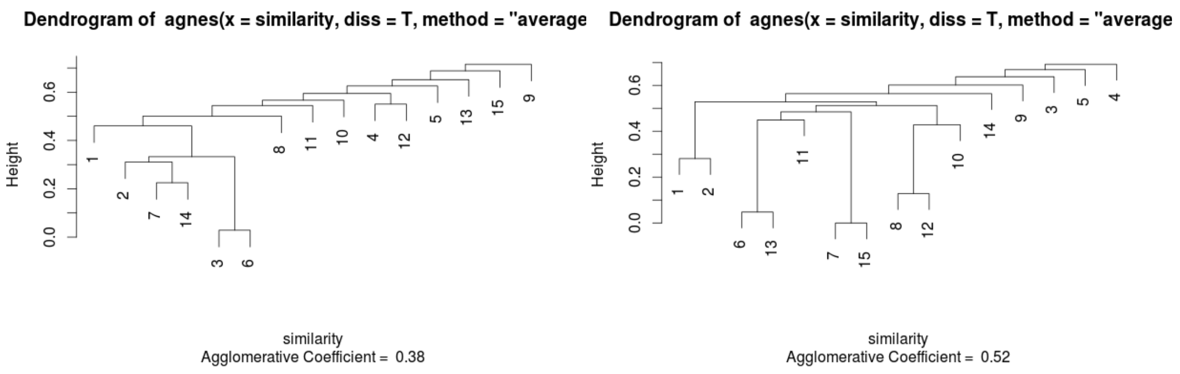


Figure 9. Analysis of the dendrograms reveals that clustering is neither possible nor sensible.

As a result, the devices of the two participants did not record a periodic pattern behaviour.

There are multiple reasons for such an outcome:

1. The devices did not record the behaviour of the elderly people. Reasons could include forgetting the device, forget to charge, charge the device during the day, living where GPS signal level is poor and intentionally producing wrong data by exchanging the devices.

2. The time span over which data is gathered is too short. On the other hand, it may very well represent periodic patterns occurring within days. Weekend behavior has little chance to be detected since other influences are greater.

3. Forcing the summary into days and hours may not overlap with the t_in and t_outs of the activities. The effect can best be compared to what Aliasing does to a representation of a perfect line rasterized into pixels: the value of the cell is valid over the whole area, and not over the area the value is derived from. Forcing the focus on whole days further misses periodic patterns. If shopping takes place every 3 days and sport every 5th, then the days are similar in an oscillating fashion.

4. In the life of these elderly people, no periodicity is available during the experiment. Context analysis shows that one was playing Samichlaus, moving around in a random fashion. It even seems user 11422 slept at different locations and was therefore not always in his home. Here, context information helped interpreting the result [10].

5. Apart from minimal variations of the minPts parameter from DBSCAN, a sensitivity analysis showed that the variables and subsequently the choice of scale has a very big impact on the whole analysis. It is believable that the reference places 2 and 4 of user

11422 are actually both in his home. It interesting to see how much the choice of parameters influences the result, raising suspicions that over-fitting can occur.

6. Further steps for the future

There are two further important steps:

1. Calculate the user similarity. Finding similar places with no geographic overlapping is very useful for animal tracking, since common movements can be detected. Because one does not need to reverse geo-code places – which is difficult for animals anyway – and only relies on visiting patterns and reference places, exploration of patterns in yet unknown areas becomes possible. This especially holds true for animals living in the outback.

2. Try to detect patterns based on the data, and not on an assumed daily behavior. Solutions could include a Fourier analysis where all frequencies can be detected, whatever periodicity they show.

7. Roles within the team

Hoda had the initial idea for the subject. Hoda and Benedikt then came up with the research question. Afterwards, Hoda organized the data before Benedikt created an initial algorithm based on [2]. We then moved over to [11], for which Hoda created a receipe for Benedikt what to implement. Benedikt subsequently implemented and included the code of the first three chapters while Hoda was writing the *Motivation*, the *Research Questions*, the *Dataset* and the *Methodology* part. We had no chance to meet after Benedikt finished the implementation. Having results in the hands, Benedikt wrote the *Results and Limitations* and *Further steps for the future* section.

8. References

[1] D. Zhang, K. Lee, and I. Lee, “Periodic Pattern Mining for Spatio-Temporal Trajectories: A Survey,” in *2015 10th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*, 2015, pp. 306–313.

[2] Z. Li, B. Ding, J. Han, R. Kays, and P. Nye, “Mining periodic behaviors for moving objects,” in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD ’10*, 2010, p. 1099.

[3] H. Cao, N. Mamoulis, and D. W. Cheung, “Discovery of periodic patterns in spatiotemporal sequences,” *IEEE Trans. Knowl. Data Eng.*, vol. 19, no. 4, pp. 453–467, 2007.

[4] N. Mamoulis, H. Cao, G. Kollios, M. Hadjieleftheriou, Y. Tao, and D. W. Cheung, “Mining, indexing, and querying historical spatiotemporal data,” *Proc. 2004 ACM SIGKDD Int. Conf. Knowl. Discov. data Min. KDD 04*, p. 236, 2004.

[5] H. Jeung, Q. Liu, H. T. Shen, and X. Zhou, “A hybrid prediction model for moving objects,” in *Proceedings - International Conference on Data Engineering*, 2008, vol. 00, pp. 70–79.

[6] R. Agrawal and R. Srikant, “Mining sequential patterns,” *Proc. 11th Int. Conf. Data Eng.*, pp. 3–14, 1995.

[7] M. Lv, L. Chen, and G. Chen, “Mining user similarity based on routine activities,” *Inf. Sci. (Ny).*, vol. 236, pp. 17–32, Jul. 2013.

[8] P. Laube, “T2 – Data Issues”, *GEO 880 Computational Movement Analysis (University of Zurich)*, Jul. 2016.

[9] P. Laube, “T3 – Trajectory operations”, *GEO 880 Computational Movement Analysis (University of Zurich)*, Jul. 2016.

[10] P. Laube, “T5 – Context-aware Movement Analysis”, *GEO 880 Computational Movement Analysis (University of Zurich)*, Jul. 2016.

9. Appendix

Step1.R (lines executed dependent on user ID):

```
#### STEP 1 ####
#Read in Data & convert to swiss CRS (R). The raw data input is read and points without NAs are saved to
data1-2_userID.csv

#user 11422
data1<-
read.csv("./data/11422/11422_2015_12_03_07_18_25_Location.csv",skipNul=T,na.strings="NA",strip.white=T,
blank.lines.skip=T,numerals="allow.loss",stringsAsFactors=T)
data2<-
read.csv("./data/11422/11422_2015_12_10_16_18_46_Location.csv",skipNul=T,na.strings="NA",strip.white=T,
blank.lines.skip=T,numerals="allow.loss",stringsAsFactors=T)

#user 11431
data1<-
read.csv("./data/11431/11431_2015_12_03_13_07_54_Location.csv",skipNul=T,na.strings="NA",strip.white=T,
blank.lines.skip=T,numerals="allow.loss",stringsAsFactors=T)
data2<-
read.csv("./data/11431/11431_2015_12_10_12_59_28_Location.csv",skipNul=T,na.strings="NA",strip.white=T,
blank.lines.skip=T,numerals="allow.loss",stringsAsFactors=T)

data<-merge(data1,data2,all=T,sort=F)

#omit all columns containing NA values
#user 11422: 1217643 > 223239
#user 11431: 1086471 > 79167
data<-na.omit(data)
#only select high quality GPS fixes
#user 11422: 223239 > 185662
#user 11431: 79167 > 63742
data<-subset(data,data$Satellites>4)
#Longitude is of type double ("Charging" was found in a column of type double!!!)
data$Longitude<-as.double(as.vector(data$Longitude))

#Date and Time to timestamp
data$timestamp <- as.POSIXct(paste(data$Date,data$Time),  format = "%d/%m/%Y %H:%M:%S", tz = "UTC")
#27/11/2015 15:00:59 is twice in the data, remove doubles (no control over which one)
#user 11422: 185662 > 185658
#user 11431: 63742 > 63742
data<-data[which(!duplicated(data$timestamp)),]

#reproject WGS84 => CH1903
source("./WGS84_CH1903.R")
```

```
data$X <- WGS.to.CH.y(data$Latitude, data$Longitude)
data$Y <- WGS.to.CH.x(data$Latitude, data$Longitude)

View(data)
#user 11422
write.csv(data,file="./data1-2_user11422.csv")
#user 11431
write.csv(data,file="/data1-2_user11431.csv")
```

Step2.php

```
<?php
#### STEP 2: Extract visit points (PHP) ####
//takes 120 seconds to calculate
$visitPoints=array();
//the size of a building
define("CLUSTER_DISTANCE",100);
//a visit point after an half hour
define("TIME",180);
//the size of a building
define("TOLERATED_DISTANCE",100);

//Extract visit points from a raw GPS-trajectory based on the algorithm 1 of Lv et al. (p. 22)
//This function is not running in linear time because of the centroid calculation.
//This conflicts with the statement the authors made in p. 22
function visitPointExtraction($trajectory) {
	global $visitPoints;
	//current cluster (point as row which is ...7=>timestamp, 8=>X, 9=>Y...
	$currentCluster=array();
	//centroid of the current cluster (point as array(X,Y,timestamp))
	$currentClusterCentroid=array();
	//last cluster
	$lastCluster=array();
	//centroid of the last cluster (point as array(X,Y,timestamp))
	$lastClusterCentroid=array();
	foreach($trajectory as $point) {
		//new cluster
		if(count($currentCluster)==0) {
			//insert current point and continue
			$currentCluster[]=array($point[8],$point[9],$point[7]);
			$currentClusterCentroid=array($point[8],$point[9]);
		} else {
			//cluster can not be empty now

			if(distance($currentClusterCentroid[0],$currentClusterCentroid[1],$point[8],
$point[9])<CLUSTER_DISTANCE) {
				//append point to CC
				$currentCluster[]=array($point[8],$point[9],$point[7]);
				//update centroids since they are needed
				$currentClusterCentroid=getCentroid($currentCluster);
				$lastClusterCentroid=getCentroid($lastCluster);
			} else {
				if(getDuration($currentCluster)>TIME) {
					//append CC to visit points
					$visitPoints[]=$currentCluster;
					$currentCluster=array();
					$currentClusterCentroid=array();
					$lastCluster=array();
```

```
                    $lastClusterCentroid=array();
                } else {
                    //update centroids since they are needed
                    $currentClusterCentroid=getCentroid($currentCluster);
                    $lastClusterCentroid=getCentroid($lastCluster);

                    if(count($lastCluster)>0 && interval($currentCluster,$lastCluster)>TIME &&
distance($currentClusterCentroid[0],$currentClusterCentroid[1],$lastClusterCentroid[0],
$lastClusterCentroid[1])<TOLERATED_DISTANCE) {
                        $currentCluster=array_merge($lastCluster,$currentCluster);
                        $visitPoints[]=$currentCluster;
                        $lastCluster=array();
                        $lastClusterCentroid=array();
                    } else {
                        $lastCluster=$currentCluster;
                        $lastClusterCentroid=getCentroid($lastCluster);
                    }
                    $currentCluster=array();
                    $currentClusterCentroid=getCentroid($currentCluster);
                }
            }
        }
    }
}

//in the timeline: ... [ cluster1 ] ... [ cluster2 ] ...
//The interval is the difference between the end of cluster1 and the beginning of cluster2, or how long it takes
until the next cluster starts.
//cluster1 and cluster2 must not overlap, order of arguments is irrelevant
function interval($cluster1,$cluster2) {
    //time extend of cluster 1
    $min1=0;
    $max1=0;
    $first=true;
    foreach($cluster1 as $point) {
        if($first) {
            $min1=$point[2];
            $max1=$point[2];
            $first=false;
        }
        if(!$first && $point[2]<$min1) {
            $min1=$point[2];
        }
        if(!$first && $point[2]>$max1) {
            $max1=$point[2];
        }
    }
    //time extend of cluster 2
    $min2=0;
    $max2=0;
    $first=true;
    foreach($cluster2 as $point) {
        if($first) {
            $min2=$point[2];
            $max2=$point[2];
            $first=false;
        }
        if(!$first && $point[2]<$min2) {
```

```
				$min2=$point[2];
			}
			if(!$first && $point[2]>$max2) {
				$max2=$point[2];
			}
		}

		if($max1<=$min2) {
			//cluster1 before cluster2
			return $min2-$max1;
		} elseif($max2<$min1) {
			//cluster2 before cluster1
			return $min1-$max2;
		} else {
			//Error, overlap!
			return false;
		}
}
//Test interval()
//var_dump(interval(array(array(0,0,9),array(0,0,19)),array(array(0,0,1),array(0,0,5))));//should be 4 => ok.

//Gets the geometric centroid of a cluster.
//$cluster: array of $data-rows
function getCentroid($cluster) {
	if(count($cluster)==0) {
		return array(0,0);
	}
	$sumX=0;
	$sumY=0;
	foreach($cluster as $row) {
		$sumX+=$row[0];
		$sumY+=$row[1];
	}
	return array($sumX/count($cluster),$sumY/count($cluster));
}

//Gets the dimension in time of a cluster (t_out-t_in)
//cluster: array of $data-rows
function getDuration($cluster) {
	$min=0;
	$max=0;
	$first=true;
	foreach($cluster as $point) {
		if($first) {
			$min=$point[2];
			$max=$point[2];
			$first=false;
		}
		if(!$first && $point[2]<$min) {
			$min=$point[2];
		}
		if(!$first && $point[2]>$max) {
			$max=$point[2];
		}
	}
	return $max-$min;
}
```

```php
//Get the triple centroid/time in/time out for a given cluster which is an array of $data-rows.
//Output: "650313.843,235000.123,2015-11-25 21:48:01,2015-11-25 23:48:10"
function getVisitPointStruct($cluster) {
	$centroid=getCentroid($cluster);
	$min=0;
	$max=0;
	$first=true;
	foreach($cluster as $point) {
		if($first) {
			$min=$point[2];
			$max=$point[2];
			$first=false;
		}
		if(!$first && $point[2]<$min) {
			$min=$point[2];
		}
		if(!$first && $point[2]>$max) {
			$max=$point[2];
		}
	}
	return $centroid[0].','.$centroid[1].','.date("Y-m-d H:i:s",$min).','.date("Y-m-d H:i:s",$max);
}
//Test getDuration()
//var_dump(getDuration(array(array(0,0,0),array(1,1,10))));
//10 => ok

//This function calculates the euclidean distance in meters between two points
function distance($p1X,$p1Y,$p2X,$p2Y) {
	return sqrt(pow($p1X-$p2X,2)+pow($p1Y-$p2Y,2));
}
//Test distance()
//var_dump(distance(0,0,1,100));//little less over 100 => ok

//MAIN PROGRAM
//get file contents of GPS fixes in CH1903
$data=file_get_contents('./data1-2_user'.intval($user).'.csv');
//split the lines into an array
$data=explode("\n",$data);
//remove first header line
array_shift($data);
//remove last empty line
array_pop($data);
//for each entry...
for($i=0;$i<count($data);++$i) {
	//...split the row into an array
	//!!! X coordinate is at $data[$i][8], Y coordinate at $data[$i][9]
	$data[$i]=explode(",",$data[$i]);
	//convert the time to an Unix-timestamp enabling time math (yes, it's UTC!)
	$data[$i][7]=strtotime($data[$i][7]);
}

//subset of 200 points
//$data=array_slice($data,0,200);

visitPointExtraction($data);//takes 78-120 seconds
//var_dump(count($visitPoints));//130

echo 'X,Y,tin,tout<br>';
```

```
foreach($visitPoints as $visitPoint) {
	echo getVisitPointStruct($visitPoint).'<br>';
}
//now: save copy-paste output to data2-3_userID.csv
?>
```

Step3.R

```
#### STEP 3 ####
#Cluster visit points to find reference spots (R). The output is the same as in step 2, but with the cluster ID.
data<-read.csv("./data2-
3_user11422.csv",header=T,skipNul=T,na.strings="NA",strip.white=T,blank.lines.skip=T,numerals="allow.loss",s
tringsAsFactors=T)

data<-read.csv("./data2-
3_user11431.csv",header=T,skipNul=T,na.strings="NA",strip.white=T,blank.lines.skip=T,numerals="allow.loss",s
tringsAsFactors=T)

dataDBSCAN<-cbind(x<-data$X,y<-data$Y,tin<-data$tin,tout<-data$tout)
#dataDBSCAN<-as.matrix(dataDBSCAN[,1:2])

#detecting the knee in raw values (for finding a suitable value for the eps-neighborhood)
#user 11422
sort(kNNdist(dataDBSCAN,k=5))
#user 11431
sort(kNNdist(dataDBSCAN,k=4))
#detecting the knee in plot
#user 11422
plot(sort(kNNdist(dataDBSCAN,k=5)))
#user 11431
plot(sort(kNNdist(dataDBSCAN,k=4)))
#=> user 11422: knee around 754 (index 216)
#=> user 11431: knee around 757 (index 193)
kNNdistplot(dataDBSCAN)
#user 11422
dbscanResult<-dbscan(dataDBSCAN,eps=754,minPts=5)
#user 11431
dbscanResult<-dbscan(dataDBSCAN,eps=757,minPts=4)
dbscanResult$cluster
#user 11422
#DBSCAN clustering for 193 objects. Parameters: eps = 754, minPts = 5
#The clustering contains 5 cluster(s). 0 indicates noise points
#user 11431
#DBSCAN clustering for 64 objects. Parameters: eps = 757, minPts = 4
#The clustering contains 7 cluster(s). 0 indicates noise points

data$clusterID<-dbscanResult$cluster

data$tin <- as.POSIXct(data$tin,  format = "%Y-%m-%d %H:%M:%S", tz = "UTC")
data$tout <- as.POSIXct(data$tout,  format = "%Y-%m-%d %H:%M:%S", tz = "UTC")

dataPoints<-SpatialPoints(cbind(data$X,data$Y))
#coordinates(data)<-c("X","Y")#promote to SpatialPointsDataFrame
#spplot(data)
#spplot(dataPoints,value=as.vector(data$clusterID))
ggplot(data,aes(data$X,data$Y,colour=data$clusterID))+geom_point()

write.csv(data,file="/home/bs/data3-4_user11422.csv")
write.csv(data,file="/home/bs/data3-4_user11431.csv")

View(data)
```

Step4.php

```
<?php
#### Step 4: Transform the visit points into 1-day activities as a daily style. ####
//takes 2 seconds to calculate
//get visit points
$num_referenceSpots=0;
//get file contents of visit points
$visit_points=file_get_contents('./data3-4_user'.intval($user).'.csv');
//split the lines into an array
$visit_points=explode("\n",$visit_points);
//remove first header line
array_shift($visit_points);
//remove last empty line
array_pop($visit_points);
//for each entry...
for($i=0;$i<count($visit_points);++$i) {
	//...split the row into an array
	//!!! X coordinate is at $visit_points[$i][1], Y coordinate at $visit_points[$i][2], referencePlaceID
(clusterID) at $visit_points[$i][5]
	$visit_points[$i]=explode(",",$visit_points[$i]);

	//convert the times to an Unix-timestamp enabling time math (yes, it's UTC!)
	$visit_points[$i][3]=strtotime($visit_points[$i][3]);
	$visit_points[$i][4]=strtotime($visit_points[$i][4]);

	if($visit_points[$i][5]>$num_referenceSpots) {
		//find out the highest clusterID which is at the same time the number of reference spots
		$num_referenceSpots=$visit_points[$i][5];
	}
}
//reference spot ids 1, 2, 3, 4, 5
//reference spot id == 0  are outliers (not important for periodicity measurements) and therefore ignored.

//1-day activities as a daily style
//access: $days[daynumber][referencePlaceID][hour_start]
//return: number of seconds

//initialize $days: 15 days for 5 reference places
$days=array();
$anEmptyDay=array(0=>0,1=>0,2=>0,3=>0,4=>0,5=>0,6=>0,7=>0,8=>0,9=>0,10=>0,11=>0,12=>0,13=>0,14=>0
,15=>0,16=>0,17=>0,18=>0,19=>0,20=>0,21=>0,21=>0,22=>0,23=>0);
for($i=0;$i<15;++$i) {
	//add the day with counters for 1. on the way 2. referenceSpot_1 3. referenceSpot_2 4. referenceSpot_3
5. referenceSpot_4 6. referenceSpot_5
	$addMe=array($anEmptyDay);
	//add a stub for each reference spot
	for($j=0;$j<$num_referenceSpots;++$j) {
		$addMe[]=$anEmptyDay;
	}
	$days[]=$addMe;
}

//The beginning of the wall clock, day0 = $days[0] = 26. November 2015
```

```
$epoch=strtotime('2015-11-26 00:00:00');

//Following foreach loop is the implementation of following sentence in the paper:
//For each reference place visited in a specific day and each time span of that day, we check the visit points of
the reference place to find how long the user stay[s] in the reference place during the corresponding time
span, and fill in the corresponding entry with the result.
//I have done it in inverse: I go through all visit points since I will traverse the visit spots less (visit points define
the table, so why ignore the data and loop through the time slots instead? - makes only difference in speed).

//Go through every visit point
foreach($visit_points as $visit_point) {
	//Assign the number of seconds to the corresponding cell in $days
	//=go through every second and add it to the corresponding cell in $days
	for($i=$visit_point[3];$i<$visit_point[4];++$i) {
		//For adding the second to the cell, we need to know:

		//1. the day ID = number of times 24*60*60 seconds passed from the epoch to the current second
		$dayID=floor(($i-$epoch)/86400);

		//2. the referencePlaceID = $visit_point[5]

		//3. the hour.
		$hour=intval(date("H",$i));

		//Add the second
		$days[$dayID][$visit_point[5]][$hour]++;
	}
}

//Following for loop is the implementation of following sentence in the paper:
//If the sum of staying time distributed in all reference places during a specific time span is less than an hour,
we assign the remaining time to the ‘‘on the way’’ row of the corresponding time span.

//Update "on the way"-values
for($i=0;$i<count($days);++$i) {
	for($hour=0;$hour<24;++$hour) {
		$timeSpentPerTimeSlot=0;
		for($j=1;$j<$num_referenceSpots+1;++$j) {
			//add contribution of each reference spot
			$timeSpentPerTimeSlot+=$days[$i][$j][$hour];
		}

		if($timeSpentPerTimeSlot<3600) {
			//[0] is the "on the way" reference spot.
			$days[$i][0][$hour]=3600-$timeSpentPerTimeSlot;
		}
	}
}

function step4Output() {
	global $days,$num_referenceSpots,$epoch,$user;
	$out='';
	//dump $days visually (saved in step4.html)
	$out.='<b>Warning:</b> Every time reference is in UTC. For analysis, keep in mind CET is +1 hour in
winter. User '.intval($user);
	$out.='<style>table,tr,td,th{border:1px solid #000;border-collapse:collapse}</style>';
	$dayID=0;
	foreach($days as $day) {
		//Title
		$out.='<h2>'.date('d.m.Y',$epoch+($dayID*86400)).'</h2>';
		$out.='<table><tr><td></td>';
		//0h, 1h, 2h, ...
		for($hour=0;$hour<24;++$hour) {
			$out.='<th>'.$hour.'h</th>';
		}
		$out.='</tr>';
		//for each reference spot...
		for($i=1;$i<$num_referenceSpots+1;++$i) {
			$out.='<tr><th>referenceSpot_'.$i.'</th>';
			//...dump every hour
			for($hour=0;$hour<24;++$hour) {
				//Here I shade the background for a better visual comprehension. The numbers represent
seconds.
				$out.='<td style="color:#000;text-shadow:0px 0px 1px #fff;background-
color:rgb('.round(255-($day[$i][$hour]/3600*255)).','.round(255-($day[$i][$hour]/3600*255)).','.round(255-
($day[$i][$hour]/3600*255)).');">'.$day[$i][$hour].'</td>';
			}
			$out.='</tr>';
		}
		//on the way
		$out.='<tr><th>on the way</th>';
		for($hour=0;$hour<24;++$hour) {
			$out.='<td style="color:#000;text-shadow:0px 0px 1px #fff;background-color:rgb('.round(255-
($day[0][$hour]/3600*255)).','.round(255-($day[0][$hour]/3600*255)).','.round(255-($day[0]
[$hour]/3600*255)).');">'.$day[0][$hour].'</td>';
		}
		$out.='</tr>';
		$out.='</table>';
		$dayID++;
	}
	return $out;
}
?>
```

Step5.php

```
<?php
#### Step 5: Define necessary function to calculate the similarity between place reference vectors, two 1-day
activities and two clusters of 1-day activities. The similarity matrix is dumped. ####
//Continue with the table generated in previously executed step 4 and produce the similarity matrix.

//The following function is the implementation of following sentence in the paper:
//Since a 1-day activity is represented by a place preference matrix which can be further decomposed into 24
place preference vectors, each of which stands for the distribution of time spent at each reference place
during the specific time span of that day, we use cosine coefficient to measure the similarity between each
pair of place preference vectors (i.e. X = (x1 , ... , xD), Y = (y1, ... , yD )) as Eq. (1), in which D is the number of
the total extracted reference places of the user, xi and yi are the time duration the user stayed at the ith
reference place during the corresponding time span in two different days.

//A filled $days-variable must be computed already (in step4.php)
function similarityBetweenPlacePreferenceVectors($day1,$hour1,$day2,$hour2) {
	global $days;
	//We need to access the array by columns, not rows.

	//$aPlacePreferenceVector is array(ref1=>1800s,ref2=>1000s,onTheWay=>800s) with onTheWay==0,
ref1==1, ...
```

```
	$aPlacePreferenceVector=array();
	$anotherPlacePreferenceVector=array();

	//iterate over all reference places (should be 5 + 1 on the way)
	$numOfPlaces=count($days[$day1]);
	for($i=0;$i<$numOfPlaces;$i++) {
		//add the number of seconds a user stays at a reference place in a given hour on a given day.
		$aPlacePreferenceVector[]=$days[$day1][$i][$hour1];
		$anotherPlacePreferenceVector[]=$days[$day2][$i][$hour2];
	}

	//Place preference vectors are ready now.
	//compute the cosine similarity

	$upperSum=0;
	//safe to initialize to 0 because there will always be a value that is not 0 (on the way)
	$x_iSquared=0;
	$y_iSquared=0;
	for($i=0;$i<$numOfPlaces;++$i) {
		//x_i*y_i
		$upperSum+=($aPlacePreferenceVector[$i]*$anotherPlacePreferenceVector[$i]);
		//x_i^2
		$x_iSquared+=pow($aPlacePreferenceVector[$i],2);
		//y_i^2
		$y_iSquared+=pow($anotherPlacePreferenceVector[$i],2);
	}
	return $upperSum/(sqrt($x_iSquared)*sqrt($y_iSquared));
}
//Testing
//var_dump(similarityBetweenPlacePreferenceVectors(0,0,1,20));

//This function is the implementation of the following sentence from the paper:
//The similarity between two 1-day activities is the average value of the similarity of their decomposed place
preference vectors with the same time span.

//$day1 is an integer referring to the dayID
function similarityBetween1DayActivities($day1,$day2) {
	$cosineSimilaritySum=0;
	//Attention: according to the paper, place preference vectors are compared. This conflicts with the table
on page 4. Adhering to paper...
	//This means that I compute the average over 24 values, not D which is the number of reference places.
	for($i=0;$i<24;++$i) {
		//...with the same time span = $i for both $hour1 and $hour2 parameter of
similarityBetweenPlacePreferenceVectors().
		$cosineSimilaritySum+=similarityBetweenPlacePreferenceVectors($day1,$i,$day2,$i);
	}
	return floatval($cosineSimilaritySum/24);
}
//Testing
//var_dump(similarityBetween1DayActivities(9,11));//9,11 should give 1

//Build a global similarity cache-matrix one can use to look up similarities. (takes 1 second)
$similarities=array();
for($i=0;$i<count($days);++$i) {
	//initialize similarity matrix with -1
	$similarities[]=array_fill(0,count($days),-1);
	//same day is equal to itself
	$similarities[$i][$i]=1;
```

```
}
//15 elements=120 links (15+14+13+12...)

//for every element...
for($i=0;$i<count($days);++$i) {
	for($j=0;$j<$i;++$j) {
		//...compare against every other once. (building triangle)
		$value=floatval(similarityBetween1DayActivities($i,$j));
		//assign to both sides along the diagonal
		$similarities[$i][$j]=floatval($value);
		$similarities[$j][$i]=floatval($value);
	}
}

function dumpSimilarityMatrixHTML() {
	global $similarities,$days,$user;
	$out="<style>
	table,tr,th,td{border:1px solid #000;border-collapse:collapse;}
	tr:hover{background-color:#fcc;}</style><h2>Similarity matrix user
".intval($user)."</h2><table><tr><th>in %</th>";
	for($i=0;$i<count($days);++$i) {
		$out.="<th>day_".$i."</th>";
	}
	$out.="</tr>";
	for($i=0;$i<count($days);++$i) {
		$out.="<tr><th>day_".$i."</th>";
		for($j=0;$j<count($days);++$j) {
			$out.="<td>".round($similarities[$i][$j]*100,2)."</td>";
		}
		$out.="</tr>";
	}
	$out.="</table>";
	return $out;
}

function dumpSimilarityMatrixR() {
	global $similarities,$days;
	$out="rm(similarity)\nsimilarity<-matrix(nrow=15,byrow=T,data=c(";
	for($i=0;$i<count($days);++$i) {
		for($j=0;$j<count($days);++$j) {
			$out.=$similarities[$i][$j].',';
		}
	}
	//remove last comma
	$out=substr($out,0,-1);
	$out.="))";
	return $out;
}

//This function can calculate the similarity between two clusters because the two clusters get merged into a
$setOfDays. In this function an implementation of following sentence in the paper follows:
//The similarities of two 1-day activity clusters is calculated by averaging the similarities between each pair of
1-day activities of the two clusters.

//$setOfDays is array(with list of $dayIDs)
function similarityBetweenMultiple1DayActivities($setOfDays) {
	global $similarities;
	//Require $setOfDays to be a set.
	$setOfDays=array_unique($setOfDays);
	//avg(each pair of day-day similarity)
	//=>look up similarities in $similarities
	if(count($setOfDays)==1) {
		//identity
		return 1;
	}
	$gatheredSimilaritiesSum=floatval(0);
	$gatheredSimilaritiesCounter=0;
	//for every element...
	for($i=0;$i<count($setOfDays);++$i) {
		for($j=0;$j<$i;++$j) {
			//...compare against every other once. (building triangle)
			$gatheredSimilaritiesSum+=floatval($similarities[$setOfDays[$i]][$setOfDays[$j]]);
			$gatheredSimilaritiesCounter++;
		}
	}
	return floatval($gatheredSimilaritiesSum/$gatheredSimilaritiesCounter);
}
//Test, overall similarity is 58.521667%
//var_dump(similarityBetweenMultiple1DayActivities(array(0,1,2)));//should be 22.807..., returns 22.807...
?>
```

Step6.php

```
<?php
#### Step 6: One-day activity clustering (algorithm 2): generate updated similarity matrices, calculate the Dunn index and merge clusters. ####
//Overview of the cluster structure:
//	cluster list at the beginning: (day0), (day1), (day2), (day3), (day4), (day5), (day6), (day7)
//	cluster list at the termination: (day0,day3,day5), (day2,day7,day4), (day1)
//	What we need are the similarities between the clusters. How to calculate this? For days, see step5.php, for clusters see similarityMatrix().

//similarity matrix between clusters
//$clusters is an array(array(0,1,4,5),array(2,6,7),...)
function similarityMatrix($clusters) {
	$matrix=array();
	for($i=0;$i<count($clusters);++$i) {
		//initialize similarity matrix with -1
		$matrix[]=array_fill(0,count($clusters),-1);
		//same cluster is equal to itself
		$matrix[$i][$i]=floatval(1);
		//for debugging only
		//$matrix[$i][$i]=similarityBetweenMultiple1DayActivities($clusters[$i]);
	}

	//for every cluster...
	for($i=0;$i<count($clusters);++$i) {
		for($j=0;$j<$i;++$j) {
			//...compare against every other cluster once. (building triangle)
			$value=similarityBetweenMultiple1DayActivities(array_merge($clusters[$i],$clusters[$j]));
			//assign to both sides along the diagonal
			$matrix[$i][$j]=floatval($value);
			$matrix[$j][$i]=floatval($value);
		}
	}
	return $matrix;
}
//Test
//var_dump(similarityBetweenMultiple1DayActivities(array(0,1)));//float(0.39968715687805)
//var_dump(similarityBetweenMultiple1DayActivities(array(2,3)));//float(0.17880993827685)
//var_dump(similarityBetweenMultiple1DayActivities(array(0,1,2,3)));//float(0.34896607862758)
//var_dump(similarityMatrix(array(array(0,1),array(2,3))));//correct 0.3489 in matrix!

//This function returns the place where the maximal value in the matrix resides, ignoring the identity diagonal.
//Found in the paper in algorithm 2, function call argmax_i,j(SM[i][j])

//$matrix from similarityMatrix()
function coordsOfMaxInMatrix($matrix) {
	$maxFound=-1;
	$coords=array();
	for($i=0;$i<count($matrix);++$i) {
		for($j=0;$j<$i;++$j) {
			if($matrix[$i][$j]>$maxFound) {
				$maxFound=floatval($matrix[$i][$j]);
				//Add maxFound to result set for convenient access.
				$coords=array($i,$j,floatval($maxFound));
			}
		}
	}
	return $coords;
}
//Test
//var_dump(coordsOfMaxInMatrix(similarityMatrix(array(array(0,1),array(2,3)))));//is correct $matrix[1]
[0]==0.34896607862758!
//Also correct for single element cluster matrix ($similarities).

//$clusters is array(array(day0,day4,day5),array(day1,day2,day3),array(day6))
function minIntraClusterSimilarity($clusters) {
	$minFound=floatval(1);
	foreach($clusters as $cluster) {
		//$value: avoid double computation because $value will probably be assigned afterwards
		$value=similarityBetweenMultiple1DayActivities($cluster);
		//echo implode('_',$cluster).' > '.$value.'<br>';
		if($value<$minFound) {
			$minFound=floatval($value);
		}
	}
	return floatval($minFound);
}
//Test
//var_dump(similarityBetweenMultiple1DayActivities(array(0,4,5)));//float(0.58651555180373)
//var_dump(similarityBetweenMultiple1DayActivities(array(1,2,3)));//float(0.22059600086679)
//var_dump(minIntraClusterSimilarity(array(array(0,4,5),array(1,2,3))));//correct 0.22...

//Observe that i!=j condition is ensured (equation 2, page 24) because of $j<$i in coordsOfMaxInMatrix(). For
the same reason, I had to initialize the diagonal with 1 (ones), because the loop cannot reach the diagonal.

function maxInterClusterSimilarity($matrix) {

	$maxValue=coordsOfMaxInMatrix($matrix);
	return floatval($maxValue[2]);
}
//Test
//var_dump(maxInterClusterSimilarity($similarities));//1, correct
```

```
//$clusters is array(array(day0,day4,day5),array(day1,day2,day3))
function dunnIndex($clusters,$matrix) {
	//Debug: uncomment
	//echo 'intra: '.minIntraClusterSimilarity($clusters).' und inter: '.maxInterClusterSimilarity($matrix).' = 
'.floatval(minIntraClusterSimilarity($clusters)-maxInterClusterSimilarity($matrix)).'<br>';
	return floatval(minIntraClusterSimilarity($clusters)-maxInterClusterSimilarity($matrix));
}
//Test
//var_dump(dunnIndex(array(array(0,4,5),array(1,2,3)),$similarities));//very bad value -0.77940399913321, 
seems correct

function dumpClusterMatrix($matrix,$clusters) {
	$out="<style>
	table,tr,th,td{border:1px solid #000;border-collapse:collapse;}
	tr:hover{background-color:#fcc;}</style><h2>Cluster Similarity matrix</h2><table><tr><th>in %</th>";
	for($i=0;$i<count($matrix);++$i) {
		$out.="<th>".implode('_',$clusters[$i])."</th>";
	}
	$out.="</tr>";
	for($i=0;$i<count($matrix);++$i) {
		$out.="<tr><th>".implode('_',$clusters[$i])."</th>";
		for($j=0;$j<count($matrix);++$j) {
			$out.="<td>".round($matrix[$i][$j]*100,2)."</td>";
		}
		$out.="</tr>";
	}
	$out.="</table>";
	return $out;
}

//initialize with single element-clusters
$clusters=array();
for($i=0;$i<count($days);++$i) {
	//Add an array with just one element
	$clusters[]=array($i);
}

//This matrix has not the day IDs as columns and rows, but the cluster ID which results in array(dayID,dayID,...) 
through $clusters[clusterID].
$similarityMatrix=$similarities;

//I calculate the maxCoords before re-entering the while loop such that I have the last cluster configuration in 
the variable $similarityMatrix BEFORE the Dunn Index drops significantly.
$maxInMatrixCoords=coordsOfMaxInMatrix($similarityMatrix);

$finalClusters=array();
$terminalCondition=true;
$dunnIndex=1000;
$previousDunnIndex=$dunnIndex;
$counter=0;
while($terminalCondition) {
	$counter++;

	//merge 2 most similar clusters
	//==merge the clusters at the specified row and column specified by $maxInMatrixCoords
	$newClusters=array();
	$newClusters[]=array_merge($clusters[$maxInMatrixCoords[0]],$clusters[$maxInMatrixCoords[1]]);
	for($i=0;$i<count($clusters);++$i) {
		//if: Remove AC_x and AC_y from CS
		if($i!=$maxInMatrixCoords[0] && $i!=$maxInMatrixCoords[1]) {
			//the merged cluster is already inserted (ordering is irrelevant)
			$newClusters[]=$clusters[$i];
		}
	}

	if($counter<count($days)-1) {
		//var_dump($newClusters);
		//get the NEW similarity matrix
		$newSimilarityMatrix=similarityMatrix($newClusters);
		//Debug: uncomment
		//echo dumpClusterMatrix($newSimilarityMatrix,$newClusters);

		//calculate the Dunn Index for the NEW cluster
		$newDunnIndex=dunnIndex($newClusters,$newSimilarityMatrix);

	}

	if($newDunnIndex<0
		|| ($newDunnIndex!=1 && abs($newDunnIndex)>0.1)
		//|| $dunnIndex<$newDunnIndex //if merging of clusters would increase intra-similarities more than
decrease inter-similarities
		|| $counter>=count($days)-1 //count(days): ...or all the 1-day activities have been merged into a
single cluster. (p. 24)
	) {
		//Break and do not update $similarityMatrix and friends
		$terminalCondition=false;
	} else {
		//update $dunnIndex, $clusters, $similarityMatrix and $maxInMatrixCoords
		$dunnIndex=$newDunnIndex;
		$clusters=$newClusters;
		$similarityMatrix=$newSimilarityMatrix;
		$maxInMatrixCoords=coordsOfMaxInMatrix($similarityMatrix);
	}
}

//Our clusters are now in $clusters.
//var_dump($clusters);
//Uncomment for a single cluster
//$clusters=array(array(0,1,2,3,4,5,6,7,8,9,10,11,12,13,14));
?>
```

Step6.R

```
#### STEP 6 ####
#Cluster into K groups. It is observed that the similarity matrix doesn't really exhibit clusters.
#user 11422
rm(similarity)
similarity<-... (generated by calling echo dumpSimilarityMatrixR(); after invoking step5.php)
#user 11431
rm(similarity)
similarity<-... (generated by calling echo dumpSimilarityMatrixR(); after invoking step5.php)
require(cluster)
clusterResult<-agnes(similarity,diss=T,method="average")
plot(clusterResult,which.plots=c(2))
```

Step7.php

```
<?php
#### Step 7: Dump the representing routine activities. ####
//we have $days and $clusters. Produce the table like it is found in the paper Fig 2.b).
echo '<b>Note:</b> Day 0 is 26. November 2015, Day 1 27. November 2015 ...<br>';

foreach($clusters as $cluster) {
	echo '<h2>Representing routine activity for user '.intval($user).' cluster days '.implode('/', 
$cluster).'</h2>';
	echo "<style>
	table,tr,th,td{border:1px solid #000;border-collapse:collapse;}
	tr:hover{background-color:#fcc;}</style>";
	echo '<table><tr><td>p in %</td>';
	//0h, 1h, 2h, ...
	for($hour=0;$hour<24;++$hour) {
		echo '<th>'.$hour.'h</th>';
	}
	echo '</tr>';

	$onTheWayValues=array_fill(0,24,0);

	//for each reference spot...
	for($i=1;$i<$num_referenceSpots+1;++$i) {
		echo '<tr><th>referenceSpot_'.$i.'</th>';
		//...dump every hour
		for($hour=0;$hour<24;++$hour) {//$j in paper
			$value=0;
			//sum of seconds of all participating days in this time slot
			$sum=0;
			for($k=0;$k<count($cluster);++$k) {
				$sum+=$days[$cluster[$k]][$i][$hour];
			}
			$value=floatval(round(($sum*100)/(3600*count($cluster)),2));
			$onTheWayValues[$hour]+=$value;

			//Here I shade the background for a better visual comprehension. The numbers represent 
probabilities.
			echo '<td style="color:#000;text-shadow:0px 0px 1px #fff;background-color:rgb('.round(255-
($value/100*255)).','.round(255-($value/100*255)).','.round(255-($value/100*255)).');">'.$value.'</td>';
		}
		echo '</tr>';
	}
	//on the way
	echo '<tr><th>on the way</th>';
	for($hour=0;$hour<24;++$hour) {
		$value=100-$onTheWayValues[$hour];
		echo '<td style="color:#000;text-shadow:0px 0px 1px #fff;background-color:rgb('.round(255-
($value/100*255)).','.round(255-($value/100*255)).','.round(255-($value/100*255)).');">'.$value.'</td>';
	}
	echo '</tr>';
	echo '</table>';
}
?>
```