ZESZYTY NAUKOWE POLITECHNIKI BIAŁOSTOCKIEJ 2008
Informatyka – Zeszyt 3

Tomasz Rybak[1]

USING TEMPORAL PROCESS MODEL TO RECOVER LOST DATA

Abstract: Article describes data gathered during 23rd Chaos Communication Congress held in Berlin in December 2006. It presents characteristics of data set describing movements of participants in conference venue and errors present in it. The main part of article is description of attempts of recovering lost data, problems with it, and how different information present in data set can help with restoring lost parts. To recover lost data spatial dependencies and temporal model of Sputnik system were used.

Keywords: temporal data, spatial data, data mining, data recovery

1. Introduction to Sputnik system

Analysing human interactions is in center of interest of social sciences. It is usually done by employing questionnaires ([1]), whether filled in on paper or by using computers. This requires, however, that participants remember details of their behaviours, which is not always possible with required details.

Sputnik is Radio Frequency Identifier (RFID) system intended to trace people in small areas and buildings. Each person is wearing tag that transmits its identifier in regular time intervals to allow to store this persons position at those precise moments. System was used during 23rd Chaos Communication Conference (23C3) held in December 2006 in Berlin. This article describes analysis performed on data from 23C3.

Unlike ordinary RFID systems, Sputnik uses active tags; they are equipped in battery and transmit data whatever there is reader listening to it or not. Tag range in buildings is up to the 10m even through dry walls. Concrete walls tend to block the signal. Because transmission occurs at 2.4GHz, human body decreases power of signal by about 50%; transmissions from WiFi and Bluetooth devices using this frequency range can disturb occurring communication.

Rag has control over transmission power and can send signals varying in strength because of having independent power source. This allows for estimating distance

[1] Faculty of Computer Science, Bialystok Technical University, Bialystok

from reader. During conference 25 readers were placed in conference center in such a way that in most cases more than one reader saw tag. This, because of possibility of estimating distance from reader, allows for estimation of position of tag.

Sputnik system is not the only one that is intended for tracing and tracking people. Because of reasons mentioned earlier and development in electronics similar systems are created and used. Vassili Kostakos placed Bluetooth readers in city of Leeds in late 2006, and recorded all identifiers of pedestrians ([6]). Nathan Eagle equipped 100 cellular phones in special software which recorded cell towers and nearby Bluetooth devices seen by those phones ([3]). Josua Smith et. al. ([9]) were using ordinary passive RFID to get activities of human in house. Their setup required putting RFID tag on each of items in home, and giving reader to each human in form of wrist bracelet.

Setup similar to described in article was presented in [5], where museum items was equipped in tags, and visitors could get readers which took role of tour guides.

Privacy concerns present when human movements is traced were described by Miako Okhubo et. al. in [8].

2. Data gathered during conference

Data gathered during 23C3 was made available as both XML and binary files.

XML file contains very small portion of gathered data; it has only 357974 entries, while full data set contains 11.1 million of observations. It does not contain information about readers used to calculate positions of tags. This omission is important, as about 1/3rd of observations has no meaningful position calculated, probably because in those cases there was not enough data to calculate them. Also XML file contains data from only few hours for each day of Congress.

XML file consists of “observation” tags with attributes

id ID of tag
time time of receiving radio transmission
position position of tag; (0, 0, 0) if unknown

XML file was not used in analysis because it lacked sequence numbers and identifiers of reading stations used to calculate positions of tags.

Binary file contained all data packets received from tags; it consisted of 1144232 values. Each packet contained time of reception, IP address of reading station, strength of signal and sequence number. Because of error in server software, identifiers of tags were not saved. It made analysis of behaviour impossible, so first concern was to recover sequences of packets produced by individual tags.

Entire data set was stored in relational database so reading and parsing file with data is needed only once, as those operations take long time. SQL offered by database allows for writing analysis algorithms in much simpler way than it would be written when parsing would be required. Basic table used to store received packets consists of columns storing identifier of tag (initially empty column), time of reception, sequence value, strength of received signal, station that received it and additional information (pressed button, etc.).

Created database can be seen as temporal, and when looking at XML data containing position of tags also as spatial one. Such databases store information about presence of phenomenas in space and time. This database stores information about presence of tags (and probably persons wearing them) at the place at the moment. Activities performed using tags, like pressing button, are also stored in it. Additional spatial data, like geometry of building and rooms where events were held, and temporal data (schedule of Congress) can be used later for more sophisticated analysis.

Table containing data from 23C3 occupies about 700MB on hard drive. Data types used to store sequence and time values occupy 8 bytes each; index for each of those columns takes 250MB. Sequence identifier is stored as 4 byte integer and its index takes about 130MB. Creation of indexes is necessary for performance reasons, because size of database is larger than available RAM and analysis speed depends on disk performance.

3. Analysis of data

Data was stored and worked upon by using relational database. Article [2] describes basic algorithms and techniques used for data mining in relational databases, like regression (including linear) and decision trees.

To understand further operations one needs to understand how tags work. In each transmission tag sends its ID and strength of signal it uses to transmit. Each transmission is encrypted using XXTEA. To avoid replay attacks it is necessary that each send packet differs from previous ones — so simple ever-increasing counter was added to the tag. Base station discards all packages with counter numbers less than the one seen previously. To avoid problems with reset of tag (i.e. removing battery) which sets counter value to 0, counter was split into two parts. Higher word (16 bits) is saved during reset, and lower (also 16 bits) is not. After reset tag increases higher word and lower is set to 0 so counter value always grows. This feature means that gaps may occur in counter values sequences when tag battery is removed. To avoid problems with transmitting packets from two tags at the same time each tag transmits and sleeps for random time, from 2 to 4 seconds.

Figure 1 shows number of packets seen in entire system in each minute. It can be seen that during day there is high activity, and during night hours activity is very low, because most of attendees left conference venue.

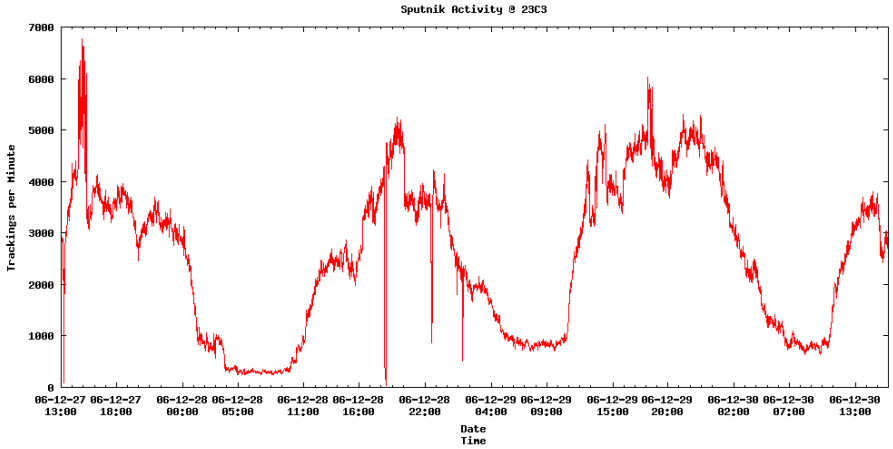

Fig. 1. Number of packets read during one minute

Table shows number of packets with particular strength of signal. Most of received signals were the strongest ones; only about 1.5% were from the weakest packets.

Strength	count
0	182874
85	568413
170	1167287
255	9225658

3.1 Rebuilding sequences

To be able to analyse data and gain some knowledge from it, sequences need to be restored. It requires joining single packets into sequences and then attaching unique number into each found sequence. Unfortunately original tag identifiers are lost and it is impossible to recover them; but even without them restoring sequences will allow for analysis of data.

Technique used for testing data mining algorithms, where part of data with known parameters is used to check correctness of proposed program, cannot be used

here, as no part of main data set contains identifiers. XML data set cannot be used to test because it contains neither reader identifier nor sequence number, which does not allow for joining those two sets.

There is not single parameter that decides whether sequence is correctly rebuild. But knowledge coming from observations of participants during Congress and analysis of source code used in tags can help with creation of heuristics and numerical parameters (like time periods) which should decide whether results are correct or not. Slope of created lines must lie in correct range and observations made using data must be correct according to physical reality. It means that one tag cannot be seen in two places at the same time, nor it may move faster than walking (running) human — move in very short time for few dozens of meters. Sequences also should be long, because tags were working all the time and most of places in BCC had coverage of readers.

3.2 Local algorithms

Local search for short sequences was used initially because global searching requires large amounts of processing time, memory and disk resources.

According to Sputnik behaviour model, one should be able to take first packet and then be able to find next one, that has next value of counter, and is 1 or 2 seconds from previous one. This ideal situation does not take into consideration gaps in sequences because of person leaving conference venue, or because one is not in the range of any readers, or when tag is transmitting too weak signal to be received by any of readers. However this is idea of finding local sequences; all functions described in this subsection are using this approach and add code dealing with gaps and choosing one packet that can be added to sequence when there is more than one.

The basic idea of algorithm for searching local sequences is to assume that packets are send once per 1.5s. Program needs to take all points from chosen period of few dozens seconds. Starting from the lowest counter value it tries to find the next value. In case of very close values of counter, difference of time is 1 or 2 seconds. In case of longer time distances, difference should be closer to 1.5s for every packet.

When more than one packet can be chosen to extend sequence conflict occurs and this problem must be resolved. Conflict may arise because either at the same time there are two different counter values, or the same value occurs at different moments. In case of either conflict we must choose only one packet to include in sequence, and discard another one. It needs to be noticed that conflict occurs when more than one packet can be added; it means that more than two packets may be involved in that conflict. The general case is presence of more than one sub-sequence (consisting of

one or more packets) that can extend existing sequence. Only one of them must be chosen, as adding all sub-sequences will destroy existing sequence by introducing decreases in either time or counter values. Sub-sequence may be chosen by taking into consideration length or resemblance to already existing sequence.

To be able to recreate sequences it is necessary to create all alternatives and then choose the best ones. All values of time and sequence counter are read in increasing order, and all points are treated as potential extension of sequences. If considered point can be added to sequence, it is. If not, conflict is detected. Previous value is removed from sequence, and both points are added to special list of alternatives. In such case each subsequent point is treated as extension not of main sequence, but alternative sub-sequences. If it can be added to all of them, alternatives are stored, and this point is added to main sequence. If it can be added to only some of sub-sequences, conflict still remains. If it cannot be added to any of sub-sequences, it is added as another alternative sub-sequence.

All found alternative sub-sequences are used to build optimal sequence. Each step of algorithm tries to build the longest and the smoothest sequence, with the smallest number of missing points. Slope of line is used to choose the best possible sub-sequence to add. Line with slope closest to 1.5 is chosen by using minimal square difference.

First program used to find sequences had time cost of $O(N^3)$, where N is number of packets. For each point it was finding whether any of other points can be added to the sequence by checking if equation $\Delta s = a\Delta t, 1.0 \leq a \leq 2.0$ was met. After finding all possible points it was generating all possible alternatives from all those chosen points. Because for every point from given interval it was checking all other points, time cost of this operation was $O(N^2)$. If any sequence was found, points belonging to it were removed from data set, and entire process was started from the beginning, giving time cost $O(N^3)$.

Improving speed of this algorithm came from observation that the longest sequences are made when starting from the lowest time and lowest counter values. Query was changed to return sorted result. Algorithm was changed to take first tuple, and try to find all other packets that can make sequence with the first one. If sequence was found, it was removed from data set; if not, only the first tuple was removed. Each packet was considered once as start of the sequence and all other points were used to build sequence with it; this gives time cost $O(N^2)$.

Local algorithms were not able to find long enough sequences. Although few found sequences were rather long (up to 20 packets for 1 minute), the most found were only consisting of only 2 or 3 packets. Attempts of joining sequences coming

from different time intervals (consecutive minutes) were not successful because of large gaps between end of one sequence and beginning of the next one.

3.3 Global algorithms

Because local searching did not lead to useful sequences, global algorithms were used. All calculations were done inside blocks of counter values of size 65536 to avoid problems with gaps caused by counter reset.

Starting from the point with lowest values of time and sequence counter program draws lines through all other points in range. Line that includes the most points is chosen; all those points are put into one sequence, and removed from the original set of points. In each case algorithm chooses line consisting of the largest number of points; this means that it is greedy algorithm.

Histogram of all slopes with bucket of size 0.1 is used to choose the best line coefficient. The largest count of points belonging to one class (bucket) reveals coefficient of line that, when drawn, will include the most points. This line should be chosen as the next sequence. This approach can be seen as using "naive Bayes" classification, as the most populated class is used — but on the other hand not all points are selected for this sequence. To be sure that no point is left without sequence because of rounding errors, range of slopes is used instead of simple comparison: all points that are on lines with slopes differing less than ± 0.3 from chosen slope are included into created sequence.

Because for each point all other points are used to calculate slopes and then all points that are in right coefficient range are chosen, time cost of this algorithm is $O(N^2)$.

Program was running for about 72h on AMD Duron 1.3GHz with 768MB RAM and single HDD IDE 7200RPM. It was disk-constraint, probably because of database size larger than available RAM; processor was not much used. Update of packets to include them in the sequence was done by one SQL query for one sequence. This required loading large part of table, but on the other hand allowed for database to optimise access to this table.

Many long sequences were found; only 4000 points (out of 11.1 million) were not included in sequence. However generated lines had rather strange coefficients; besides ordinary 1.4 or 2.5, one could find values 0.1, 0.4, 0.5, 9.9, 10.0, 8.1, ... Amongst sequences generated by this algorithm some were the proper ones, but many other were wrong. Those incorrect ones had line coefficient that could not be generated by firmware installed in tags, or they had points that came from two differ-

ent lines, similarly to Figure 6 or Figure 7; the most noticeable example is shown in Figure 2.

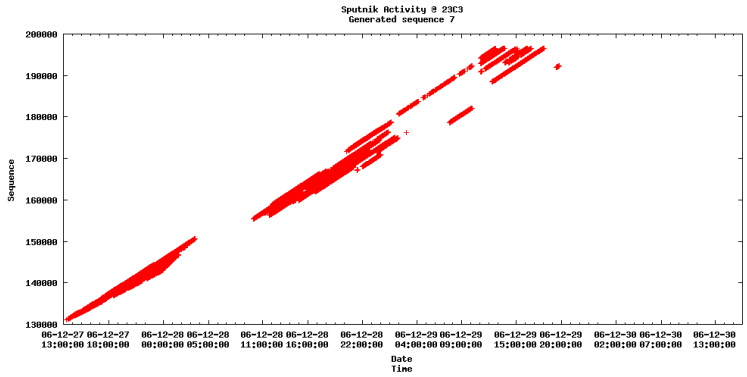

Fig. 2. Generated sequence; first set, number 7

Figure 2 shows sequence that looks like collage of many sequences. This shows the main problem of algorithm: range of allowed coefficients is too wide so too many points are added to sequence. The farther away from the first point, the more obvious it is — Figure 3 shows sequence that in the beginning is correct and gets incorrect in the end. The first part of this sequence should be preserved, and after it sequence should end; remaining points should be used to create another sequences.

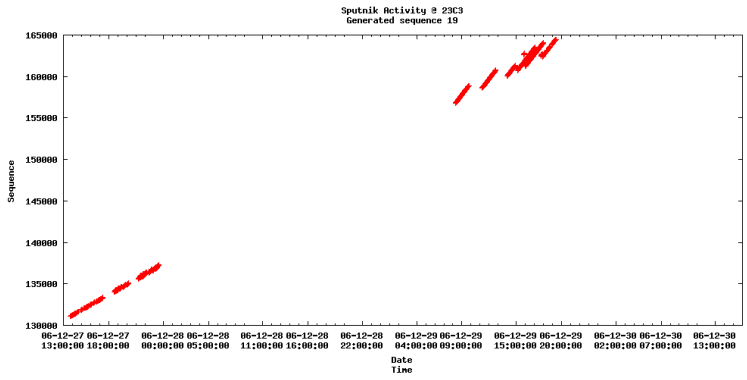

Fig. 3. Generated sequence; first set, number 19

Sequences that contain point that should belong to many different sequences unveil the main problem with algorithm, which is caused by to wide range of possible coefficient values and incorrect slopes of generated lines.

Two histograms of line coefficients were generated; one with buckets of size of 0.1 (Figure 4), and another with buckets of size of 0.001 (Figure 5). As can be seen, first histogram presents false situation; number of points in many lines that consist of small number of points but have close coefficient values is able to outnumber one line with high number of points. This leads to situation where instead of long line the short one is chosen.

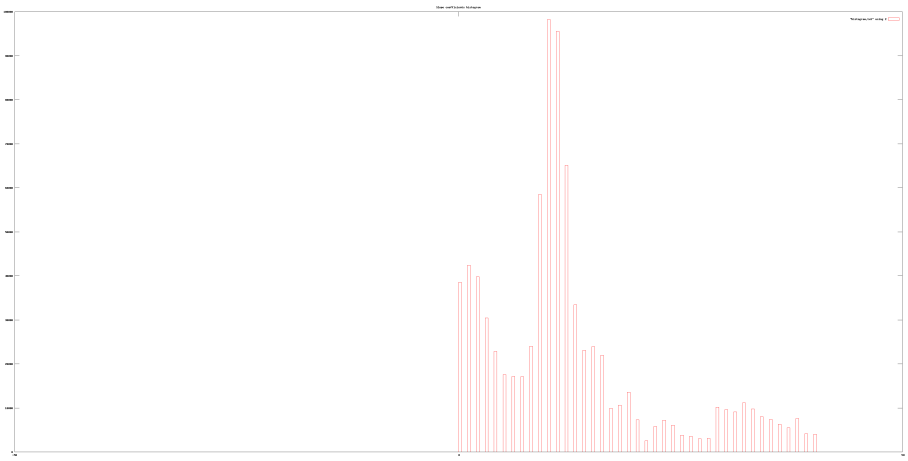

Fig. 4. Coefficients histogram for 10 buckets

Figure 4 and Figure 5 show that experiments are needed to find proper values of parameters, as incorrect values lead to wrong results.

Improvements of algorithm were necessary to obtain better results. Size of histogram class was changed to 0.001 to avoid problems with many lines joining into one. Histogram was calculated for slopes from range 1.0 to 5.0, instead of from 0.0 to 10.0. Range of coefficients of points that were used to build lines was changed from ± 0.3 to ± 0.001. This however caused gap at the beginning of each sequence, as rounding errors in the first few minutes of sequence caused slope of those initial points to be not close enough to the ideal to be included in chosen range.

SQL aggregate function was used to choose point which was included in sequence in case of presence of more than one counter value at the same time. It re-

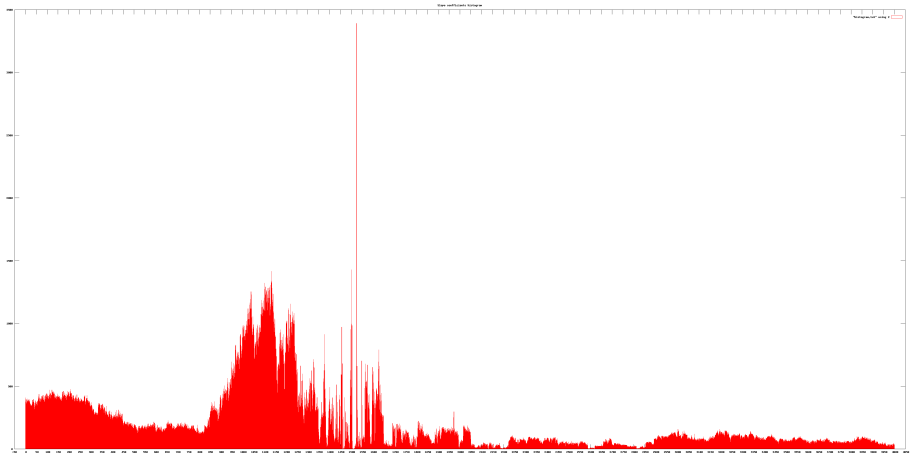

Fig. 5. Coefficients histogram for 1000 buckets

quired grouping by time in SQL query. Point which distance from the chosen slope was the smallest was chosen.

Program was running on the same machine, but it was very slow. It was running for about one week before it was stopped. It could not finish calculating sequences belonging to the first large data set ($counter \in < 2 * 65536; 3 * 65536 >$) so it was stopped and run for later counter values. It did not leave the next counter values block for another week. Its usage of disk subsystem and processor time was more balanced that for previous one. Its long working time may come from performing more calculations, using custom aggregate function, and updating information about sequences as many individual queries instead of one bulk query.

First few generated sequences were big, but later ones were getting smaller and smaller, down to dozen points. Sequences had some points missing; probably some points that should be included into those sequences were not added because of rounding errors and too narrow range of allowed coefficients.

Algorithm was joining sequences in spite of aggregate function which was used to guard against it. Data analysis shown that some sequences had errors (Figure 6, Figure 7), but they were more subtle and could not easily be seen on the graphs.

Figure 6 shows two distinct sequences that are joined. Their points are in allowed slope range, and they are interlaced, so even aggregate function can not remove them.

Figure 7 shows three distinct sequences joined into one. They have similar slope and their points lie in allowed range, so they are joined together, even though that points should create distinct lines.

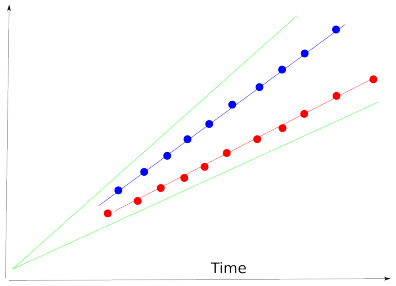

Fig. 6. Interlaced sequences

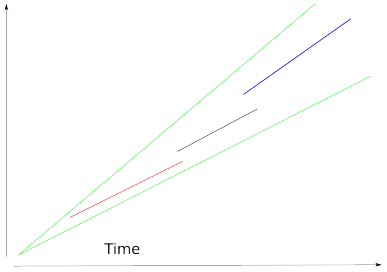

Fig. 7. Collinear sequences

Sequences that are joined but have different slopes can be detected by calculating difference of slopes between consecutive points, similarly to differentiating. The long sequence of differences of the same sign followed by long sequence of differences of another sign may suggest coupling of different sequences. To avoid splitting one sequence into many short ones number of points that have the same sign of difference between slopes and absolute difference between those slopes can be used. If both of those parameters are small, all points belong to one sequence.

This is similar approach to shown in [4], where authors were trying to find event that separates two lines. Unlike in [4] here splitting event comes not from experiment, but is interpreted by us as coupling of two lines. So "event" is just point where two lines that are joined and need to be separated.

Basing on experiments, correct values of parameters were found. While value 0.001 as size of histogram classes was found to be correct, the same value was to small as border of line coefficient. Experiments shown that value 0.01 was giving much better results. Solution to problem of points belonging to other sequences that might be included by larger border will be described later.

As mentioned earlier, because of rounding errors at the beginning of sequence coefficients do not have the same values as coefficients for further points. It is necessary to have wider allowed range of slopes in the beginning and more narrow near the end. This can be accomplished by sigmoid function. Function $0.01 + \frac{0.09}{1+e^{(x-500)/100}}$ was used in program. At the distance 0 it generates border of 0.1; its value was getting smaller to reach 0.01 for argument of 1000. Because of very large exponential values, mathematical exception is generated for arguments greater than about 70000.

Using only time and counter values gives us choice of either having too wide range and having joined sequences, or having too narrow range and leaving some points out, without guarantee that appropriate points are included in sequence. Additional data must be used in searching for good sequences.

First choice of additional parameter is strength of signal. Each tag changes strength of sent signal in sequence of values 0x00, 0xff, 0x55, 0xff, 0xaa, 0xff, 0xff, 0xff. First problem is that 5 out of 8 values are the same 0xff, so it is difficult to determine where in sequence of signal strengths particular point is. However analysing of source code and Sputnik data revealed that strength of signal is not distinctive between tags. Each tag starts at the same strength sequence point, so there is no variability between sequences. If more than one point has the same counter value, they also have the same strength of signal. It can not be used to distinguish between different sequences.

Because strength of signal could not be used to help to generate sequences, stations that received signal from tag were used. The main assumption was that set of

seen readers did not change from one point to another if those points were close in time. To keep algorithm simple only list of seen stations was considered, not their distribution in space. Similarity was defined as number of stations in both sets, divided by size of joined sets; it is known as Jaccard coefficient and defined as $c(A,B) = \frac{|A \cap B|}{|A \cup B|}$.

To avoid errors shown in Figure 6 algorithm was changed to retrieve all potential points that could be added to generated sequence and choose the best one. This approach is return to the idea of generating alternative sub-sequences used in local algorithm. Points that are part of one sequence have condition $(T_1 > T_0 \wedge S_1 > S_0)$ met. Program creates all possible sub-sequence from point that have not met this condition and then chooses the best one. To choose the best it locally compares lengths, slopes of sub-sequences and reading stations seen by all sub-sequences and chooses one that is the most similar to main sequence (pseudo-code is shown in Figure 8).

Last version of global algorithm differs from previous ones in more than only numerical parameters. Instead of using constant range, sigmoid function is used to include more points in the beginning of sequence. All points are read from database, and program builds alternative sequences from them. Instead of using custom aggregate function to choose only one point, standard function aggregating all seen readers into array is used. This array is later used to choose the best points to include into sequence. The last change is breaking line if it is discovered that created line has high probability of being two different lines.

The line that has the most points laying on it is chosen using histogram. Then all points that are on lines that differ less than border range are read from database; range is not constant, but sigmoid function is used to get it narrower in the end of block. All those points are used to build sequence; alternative sub-sequences are generated if needed. To find which sub-sequence should be added to main sequence, similarity of slopes is used. Special function was created to return similarity of sets of seen stations; it returns number from range $< 0.0; 1.0 >$:

1. If strength of signals are the same for two points, similarity is calculated by dividing number of stations seen by both points by number of all seen stations (Jaccard coefficient)
2. If first of points has more power than second, similarity is calculated as number of stations seen by latter by number of all readers seen by former (stronger)

Sub-sequence which is more similar to preceding point is chosen.

The very important part of function choosing sequences is condition allowing only sub-sequences which time and counter values are greater than already existing in sequence to be considered as alternatives. This protects from the problem of having improper sequence in case when one alternative proceeds another.

The final difference is new function `Break` which determines whether generated sequence should end. This protects from situation seen in Figures 3, and 7. This function (deciding whether to break one sequence into two) takes into consideration six parameters:

1. Similarity of sets of seen stations for both lines
2. Slope of first line
3. Slope of second line
4. Whether they differ too much to avoid stair-step-case line; from experiments 10 times difference is too much
5. Whether there is much difference in time of consecutive points
6. Whether slope of any of lines differs too much from global scope of line we are currently building

If more than 50% of those conditions is met, line is split.

```
for s in "SELECT DISTINCT sequence FROM sputnik WHERE id IS NULL":
        for t in "SELECT DISTINCT time FROM sputnik WHERE id IS NULL AND sequence = i":
                Calculate histogram of slopes of all lines starting at s, t
                slope = max(histogram)
                points = find all x, y that slope-delta <= (y-t)/(x-s) <= slope+delta
                lines = CreateAllPossibleLines(points)
                ResultLine = []
# Choose the best line:
                for l0, l1 in lines:
                        if similarity(l0, ResultLine) >  similarity(l1, ResultLine):
                                ResultLine.append(l0)
                        else:
                                ResultLine.append(l1)
                for point in ResultLine:
                        if ShouldBreak(ResultLine[0]-point, point-ResultLine[-1]):
                                Split ResultLine into two at point
```

Fig. 8. Final algorithm for finding lines

Program was run on different machine than previous ones. It was running for about 100h on 64 bit AMD 3400+ with 1GB of RAM and one SATA HDD 7200RPM. It was stopped by FPU error in sigmoid function for large values of counter. 10.6 million points was used in generated sequences. Over 1600 sequences were made from more than 1000 points.

Because some of generated sequences were short, the next step should be joining them. One solution is to try to join existing sequences, another could be trying to extend sequences by points not belonging to any sequence. But problem with joining is choosing which sequence to join with each another. No experiments with joining sequences were performed.

4. Final remarks

Regenerating sequences was started with assumption that each tag sends packet every 1.5s. This lead to setting coefficient range from 1.0 to 2.0. Because this was not giving good results in local algorithms, and by observing scatter plots, global algorithms with ragne 0.0 to 10.0 was used; and later, basing on analysing source code of Sputnik firmware, range of coefficients was changed to 1.0 to 5.0. Source code of firmware contains two calls of sleep function. One sleeps for 2s, and another for random period from 0s to 2s. This gives range of line slopes from 2s to 4s. But because second sleep function parameter is random value, there should be no straight line! However scatter plots reveals many lines. So either Sputnik data contains so many points that one can draw any line, or function rand() returns numbers that are not random. Basing on analysing packets generated by single tag, second possibility is believed to be true.

No physical (or geometrical) model was taken into consideration during generating sequences. No distance between stations or speed of movement was analysed. This could give better results in sequences, by limiting point to only those that are in range to reach from previous point. On the other hand this approach would require calculating position of each tag in every moment.

XML data set proves that it is possible to calculate position of tag. Tags send packets with different signal strength to allow for estimation of distance from reader. This estimation bases on negative knowledge. If reader is unable to read signal with small strength it means that tag is far away from it. So having few packets it is possible to calculate minimal and maximal distance tag is from reader. Power of signal was set so next level of power increases radius two times. This gives two spheres with small and large radius; person is between them. When data from few readers is known, it is possible to calculate common fragment of space where all those spheres intersect — this is position of tag. This approach requires knowing exact positions of readers.

Human body decreases strength of signal. This decreases precision of estimating position of tag, but could be used to calculate direction person has, assuming that tag is worn in the front. Range would not be sphere, but two hemispheres, larger in the front and smaller in the back. This would require performing more calculations (two times for each reader), but as there is no situation when all readers see one tag,

it would not be impossible. Direction could be proven when person moves in this direction, again with assumption that person walks forwards, not backwards.

The most interesting analysis is looking for connections and similarities between attendees. This can be done by looking for people that attended similar talks. Those people may not even know each other but have common interests.

Another research area is looking for friends. Friends can be defined as people that stay together; they tend to be together not only during talks, but also and especially during breaks. If two people are close during most breaks, they are close friends. If they are close for some times, and not close for other moments, they may be colleagues. Or they may just stay in the same queue for pizza. However here the most important is relative position (distance between people), not exact position of tags.

Article by Kumar et al. ([7]) describes analysis done on data gathered from online messaging system. Authors describe finding friends and fellows in online communities, and process of creation of groups and cliques. In my opinion it is desirable to use Sputnik in similar way.

The other possibility is to try to join internet and real life friends. But this assumes that it is possible to gather online data; companies do not make it available, as it is source of advertisement revenue.

Although observing movements of people in real time is interesting, it also raises many privacy concerns. Article shows that issues raised in [8] are real; it shows that there is even more risks to privacy than previously were thought. Authors of [8] do not even raise question of restoring identity of tags worn by people. Presented here approach is similar to calculating identity of person from "cloud" described by Kostakos ([6]), where we can compute identity of person basing on some (changing) set of tags (RFID of watch, computer, suitcase). While Kostakos presents is as theoretical danger, this article shows that this is real concern and must be taken into consideration.

References

[1] P.S. Bearman, J. Moody, and K. Stovel. Chains of affection: The structure of adolescent romantic and sexual networks. *American Journal of Sociology*, 110(1):44–91, 2004.

[2] Sašo Džeroski. Multi-relational data mining: an introduction. *SIGKDD Explorations Newsletter*, 5(1):1–16, 2003.

[3] Nathan Eagle and Alex (Sandy) Pentland. Reality mining: sensing complex social systems. *Personal Ubiquitous Comput.*, 10(4):255–268, 2006.

[4] Valery Guralnik and Jaideep Srivastava. Event detection from time series data. In *KDD '99: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 33–42, New York, NY, USA, 1999. ACM.

[5] Sherry Hsi and Holly Fait. Rfid enhances visitors' museum experience at the exploratorium. *Communications of ACM*, 48(9):60–65, 2005.

[6] Vassilis Kostakos. The privacy implications of bluetooth.

[7] Ravi Kumar, Jasmine Novak, and Andrew Tomkins. Structure and evolution of online social networks. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 611–617, New York, NY, USA, 2006. ACM Press.

[8] Miyako Ohkubo, Koutarou Suzuki, and Shingo Kinoshita. Rfid privacy issues and technical challenges. *Communications of ACM*, 48(9):66–71, 2005.

[9] Joshua R. Smith, Kenneth P. Fishkin, Bing Jiang, Alexander Mamishev, Matthai Philipose, Adam D. Rea, Sumit Roy, and Kishore Sundara-Rajan. Rfid-based techniques for human-activity detection. *Communications of ACM*, 48(9):39–44, 2005.

UZYCIE MODELU TEMPORALNEGO W CELU ODZYSKANIA UTRACONYCH DANYCH

Streszczenie: Artykuł przedstawia dane zgromadzone podczas konferencji Chaos Communication Congress która odbyła się w grudniu 2006r. w Berlinie. Dane pochodzą z systemu Sputnik który monitorował ruch uczestników konferencji. Pierwszy rozdział to któtka prezentacja danych oraz obecnych w nich błędów. Główną część artykułu stanowi opis prób odzyskania danych które zostały utracone na skutek błędu w oprogramowaniu systemu Sputnik. Opisane są sposoby odzyskiwania tych danych oraz ich rezultaty. Do odzyskania został użyty temporalny model działania systemu oraz zależności przestrzenne obecne w danych.

Słowa kluczowe: dane temporalne, dane przestrzenne, analiza danych, odzyskiwanie danych

Artykul zrealizowano w ramach pracy badawczej S/WI/2/03.