

# Finding similar situations in sequences of events via random projections

*Heikki Mannila\** and *Jouni K. Seppänen†*

## 1 Introduction

Sequences of events occur in several applications, such as mobile services (the requests made by each user), telecommunication network alarm handling, user interface studies, etc. Such a sequence can be denoted  $(\langle e_i, t_i \rangle)$ , where  $i = 1, \dots, n$ , and for each  $i$ ,  $e_i \in E$  is an *event type* and  $t_i$  is the *occurrence time*.

As an example, consider the following excerpt from a sequence of 46662 events (alarms) from a telecommunications network. Here the first field is the alarm type, and the second is the occurrence time.

Type	Time
7279	881423
7277	881799
7277	881839
1940	881951
1940	882841
1568	883830
1585	883918
1940	883959
7404	883979
2535	884810

In several data analysis applications on sequences of events, we face the problem of finding “similar” situations. This is needed, e.g., for predicting the next

\*Nokia Research Center and Laboratory of Computer and Information Science, Helsinki University of Technology. Email: [Heikki.Mannila@nokia.com](mailto:Heikki.Mannila@nokia.com)

†Laboratory of Computer and Information Science, Helsinki University of Technology. Email: [Jouni.Seppanen@cis.hut.fi](mailto:Jouni.Seppanen@cis.hut.fi)

events, and for understanding the dynamics of the process producing the sequence.

More formally, the problem of finding similar situations can be defined as follows. Given a time  $t$  and a window width  $w$ , find another time  $s$  such that the windows of the sequence occurring in intervals  $(t - w, t)$  and  $(s - w, s)$  are similar. The similarity between two slices can be defined using an edit distance notion [16], i.e., the distance is defined as the cost of the cheapest possible sequence of operations that transforms one slice to another. The operations are insertion and deletion of an event and moving an event in time; each operation has an associated cost. The edit distance can be computed using a dynamic programming algorithm, but the computation is slow. Furthermore, assigning costs to the edit operations is quite problematic [17].

The uses of similarity search in event sequences are at least twofold. First, we are interested in showing to the human analyst previous situations that resemble the current one. That is, we want to be able to pinpoint certain cases in the past that might contain useful information for the human. Second, we would like to use the similarity criterion to predict the future events: if slices similar to the current one have in the past been followed by certain types of events, we might expect to see them again. In this paper we are mostly concerned with the first type of application. This means that an occasional false match is not really a problem, and that the method should be fast enough to be used even for searching long sequences.

In this paper we describe a simple and fast way of mapping a sequence of events into points in  $k$ -dimensional Euclidean space using a random function, and show how this mapping can be used as a preprocessing method for finding similar situations. We contrast the accuracy and performance of our method with the dynamic programming approach, but the real validation of the method is in the experimental results.

We close this introduction by considering related work. Similarity between objects is a fundamental notion whose definition is crucial to various data mining and information retrieval methods. In order to look for patterns or regularities in data, it is often necessary to be able to quantify how far from each other two data objects are. Once similarity has been defined, we can use, e.g., distance-based clustering or nearest neighbor techniques to search for interesting information from the data. Recently, there has been considerable interest in defining intuitive and easily computable measures of similarity between complex objects and in using abstract similarity notions in querying databases [13, 14, 22, 11, 15, 7].

Most of the above work has concentrated on similarity notions in unordered data. The sequential aspect of data is important also in the analysis of time series data occurring in, e.g., several financial and scientific applications, such as stock price indices, the volume of product sales, telecommunications data, one-dimensional medical signals, audio data, and environmental measurement sequences. While there is a vast body of statistical literature on time series, similarity notions appropriate for data mining applications such as described above have not been studied much. Time series similarity concepts have been studied in [1, 2, 9, 20].

An interesting recent solution for a somewhat similar problem is based on generative models instead of distance measures [8]. This paper considers real-valued time-series data and presents a way of constructing waveform models from the data.

Our work concerns sequences of events. They could be considered to be time-series data where the values are discrete events, and this means waveform models are not applicable. To apply the model-based approach, one would apparently need some other kind of prior assumptions on relationships in the data.

The rest of this paper is organized as follows. Section 2 describes the random mapping method somewhat abstractly, and Section 3 shows how the mapping can be used for similarity search in practice. Section 4 discusses why we expect the method to meet our goals in theory, and Section 5 describes how we have tested it with real data. Section 6 is a short conclusion.

## 2 Random mappings for event sequences

In this section we describe how sequences of events are mapped to real quantities. We use ideas stemming from the Johnson-Lindenstrauss lemma [12], which recently has attracted lots of interest: see, e.g., [6, 5, 19, 21, 3].

In our application, the events have occurrence times, and this has to be taken into account in the definition of the random projection. The projection is defined in two phases as follows.

For each event type  $e \in E$  and for each  $j = 1, \dots, k$  let  $\rho(e, j)$  be a normally distributed random variable with mean 0 and variance 1. In other words, we associate with each event type a random  $k$ -dimensional vector. This defines a random mapping from the set of event types into  $k$ -dimensional space.

Let  $f$  be a function from  $[0, w]$  to  $[0, 1]$ . This function will be used to extend the mapping defined above to slices of events. To do this, we first define some notation. Given an event sequence  $S = (\langle e_1, t_1 \rangle, \dots, \langle e_n, t_n \rangle)$  and a time  $t$ , denote by  $S(t, w)$  the subsequence of  $S$  consisting of those events that happen within the (half-open) interval  $(t - w, t]$ . We call  $S(t, w)$  a *slice* of  $S$ . Let this sequence consist of the events  $\langle d_1, s_1 \rangle, \dots, \langle d_m, s_m \rangle$ .

Now, the random mapping for slices maps  $S(t, w)$  to the  $k$ -dimensional vector  $r(t, w) = (y_1, \dots, y_k)$ , where

$$y_j = \sum_{i=1}^m f(t - s_i) \rho(d_i, j).$$

That is, for dimension  $j$  we sum the variables  $\rho(d_i, j)$  corresponding to the event types occurring in the slice, weighted by the function  $f$  on the distance of the event from the end point of the slice.

In our experiments, we have chosen  $f$  to be linear, i.e.,  $f(x) = x/w$ . In this case, there is a simple incremental algorithm to compute the  $k$ -dimensional representations  $r(t_i, w)$  of all the slices in the sequence. The algorithm slides a window of width  $w$  through the data and keeps the “current”  $k$ -vector in an array variable `current` and the unweighted sum of the vectors corresponding to events within the window in another variable `sum`. When the position of the window changes from  $t$  to  $t'$ , `current` can be updated by adding the value of `sum` scaled by  $f(t' - t)$ . When events enter and exit the window, `current` and `sum` are updated accordingly.

Another possibility would be to choose an exponential function  $f(x) = e^{-x}$ . The incremental algorithm would be even simpler: as the window position changes, it suffices to multiply `current` by  $f(t' - t)$ . If the incremental-computation property can be sacrificed, we might choose functions that give different weight to different parts of the slice.

Thus the mapping is quite easy to compute; in practice, the computation is very fast. For example, even a naive `awk` implementation that doesn't take advantage of the incremental nature of the task but computes the mapping separately for each window, took 19 minutes on an SGI O2 (195 MHz MIPS R10000 processor) to compute the vectors for the 46662-event alarm data used in our examples with  $w = 1000$  and  $k = 10$ . Thereafter, it took about 40 seconds per window to find the closest match for a window, i.e., to produce a single data point for Figure 1. An implementation of the incremental method in, say, C, would of course be orders of magnitude faster.

### 3 Using the mapping in searching for similar situations

Suppose we have accumulated a long sequence of events (or a set of sequences), and we are given a slice of events that have happened during the last  $w$  time units. Our task is to find the situations from the past which resemble the current situation as much as possible.

The trivial method using edit distance definitions would be to compute the edit distance between the query slice and all other slices. However, this will in practice be very slow. The edit distance computations take time  $O(nm)$  for slices of lengths  $n$  and  $m$ , with fairly large constant factors. An additional problem with the use of the edit distance framework is that, as mentioned above, we need to specify lots of parameters. These parameters are seldom available and hence the relatively intuitive fingerprinting approach can be more useful.

Suppose that we have computed for each slice  $S(t_i, w)$  the corresponding  $k$ -dimensional vector  $r(t_i, w)$ . Then we compute the  $k$ -dimensional vector for the query slice  $Q$ ; let this be  $q$ . Now we search (using normal linear search or more advanced data structures) for the vector  $r(t_i, w)$  such that it and  $q$  are close to each other, i.e.,  $\|r(t_i, w) - q\|$  is small. After we find such vectors  $r(t_i, w)$ , we verify the closeness of the corresponding slice  $S(t_i, w)$  to  $Q$  by using edit distance computation. That is, the mapped slices  $r$  are used as fingerprints of the original slices.

Thus our method avoids the full search in the space of slices of event sequences by doing first a search in  $k$ -dimensional Euclidean space. Such fingerprinting methods are, of course, quite often used; see, e.g., [18]. The complexity of the method is linear in the size of the data: in practice the method is very fast, even with naive implementation. The precomputed  $k$ -dimensional vectors  $r(t_i, w)$  could be stored in, e.g., an R-tree [10], to further speed up the search.

Here we just give two examples of what the method finds. In these examples, the parameter  $k$  was set to 9. Consider the fragment of sequence displayed in

Table 1(a). The events shown are all the events occurring at most 1000 seconds prior to the event 7260 573169 occurring as event number 1730 of the example sequence. We have included also the position in the sequence, and the time is relative to the end of the slice. The slice having the smallest distance to this query window occurred at position 29212 at time 2369304 and is shown in Table 1(b). Note the intuitive similarity between these two slices.

Another example is the sequence occurring at position 38000 of the original sequence. The query slice is shown in Table 2(a). The minimum distance to query window was 0.107156 at position 37988, see Table 2(b). It can be seen that the sequences have a strong resemblance to each other.

## 4 Properties of the random mapping

Given two identical slices  $S(t, w)$  and  $S(t', w)$ , the random vectors  $r(t, w)$  and  $r(t', w)$  are obviously identical. Also, if the slices  $S(t, w)$  and  $S(t', w)$  are quite close to each other in the sense of edit distance computed by dynamic programming, then the distance between  $r(t, w)$  and  $r(t', w)$  as vectors in  $k$ -dimensional space is small.

To see this, consider first the case  $S(t, w) = (\langle d_1, t_1 \rangle, \langle d_2, t_2 \rangle, \dots, \langle d_m, t_m \rangle)$  and  $S(t', w) = (\langle d_2, t_2 \rangle, \dots, \langle d_m, t_m \rangle)$ , i.e., the slices are identical except that the latter doesn't have the element  $\langle d_1, t_1 \rangle$ . For simplicity, assume that  $f(t - t_i) = 1$  for all  $t_i$ . Then, the squared distance between  $r(t, w)$  and  $r(t', w)$  is  $d^2 = \sum_{i=1}^k r(d_1, i)^2$ . Since the variables  $\rho(d_1, i)$  are i.i.d., the expected value of the squared distance is  $E(d^2) = kE(\rho(d_1, 1)^2) = k$ . The variance is  $E(d^4) - E(d^2)^2 = k(3 - 1^2) = 2k$ .

In contrast, consider the squared distance from the origin to a vector obtained by mapping a random  $m$ -event slice  $S(t, w) = (\langle d_1, t_1 \rangle, \dots, \langle d_m, t_m \rangle)$ . Every element of the vector  $r(t, w)$  is a sum of  $m$  random variables,  $r_i(t, w) = \sum_{j=1}^m f(t - t_j)\rho(d_j, i)$ . Assuming they are independent, i.e., that no event is repeated in the slice, the expected value of  $r_i(t, w)^2$  is  $E(r_i(t, w)^2) = \sum_{j=1}^m E(\rho(d_j, i)^2) = mE(\rho(d_1, i)^2) = m$ , so the expected squared distance from origin to the vector is  $km$ . For the variance, we get  $E(r_i(t, w)^4) - E(r_i(t, w)^2)^2 = \sum_{j=1}^m [E(\rho(d_j, i)^4) - E(\rho(d_j, i)^2)^2] = m[3 - 1^2] = 2m$ , so the variance of the squared distance is  $2km$ .

Thus, the editing operations of inserting and deleting one event have a small expected effect on the distance, compared to arbitrary vectors in the  $k$ -space. In the previous analysis, we assumed that all the events have equal weight. In practice, the effects of these editing operations will be even smaller for events towards the low-weight end of the slice. Also, assuming a continuous  $f$ , the editing operation of moving an event in time has an effect proportional to the length of the time change, just as in the notion of editing distance. Therefore, the approximated distance is small when a slice is subjected to a small number of editing operations.

An inverse relationship can also be shown. That is, if two slices are far from each other, then the corresponding random vectors are far from each other with high probability. Details are omitted in this version.

Event number	Alarm type	Relative time	Event number	Alarm type	Relative time
1730	7260	0	29212	7277	0
1729	7277	29	29211	7260	1
1728	1585	224	29210	1585	205
1727	1940	821	29209	1940	536
			29208	7403	646
			29207	1903	825
			29206	7711	915
			29205	7705	982

(a)

(b)

**Table 1.** *A query slice and the closest answer slice*

Event number	Alarm type	Relative time	Event number	Alarm type	Relative time
38000	1553	0	37988	1553	0
37999	1553	0	37987	1553	0
37998	1553	300	37986	7002	72
37997	1553	300	37985	1553	300
37996	7002	313	37984	1553	300
37995	7701	489	37983	1553	600
37994	7002	553	37982	1553	600
37993	1553	600	37981	7701	621
37992	1553	600	37980	7002	692
37991	1553	900	37979	1553	900
37990	1553	900	37978	1553	900

(a)

(b)

**Table 2.** *Another query slice and the closest answer slice*

## 5 Experiments

In this section we describe the experimental studies we have used to verify the operation of our method. Recall that our primary goal is to locate previously occurring situations that resemble the current one. To test how well this goal is met, we performed the following experiments.

We conducted some experiments on telecommunications alarm data (which is also the source of the examples above) and the Entree Chicago data from the UCI KDD Archive [4]. The alarm data consists of 46662 alarms over a period of a month. The time values seen on the  $x$ -axis of several figures are in seconds, and they range from about half a million to three million. The figures show only a subset of the full range for reasons of legibility. There are 180 types of alarms occurring in the data.

The Entree Chicago data comes from the log of an application which provides restaurant recommendations on the Web. We consider only the final recommendation made by the system for a single user. We converted the date/time information to seconds since the beginning of 1970 (a popular way of expressing time in some operating systems), so the time values vary from about 842 million to 924 million, and again the figures show a smaller range. The sequence contains 50672 events of 619 types.

We first describe the experiments on the telecommunications alarm data, and then show some results on the Entree Chicago data.

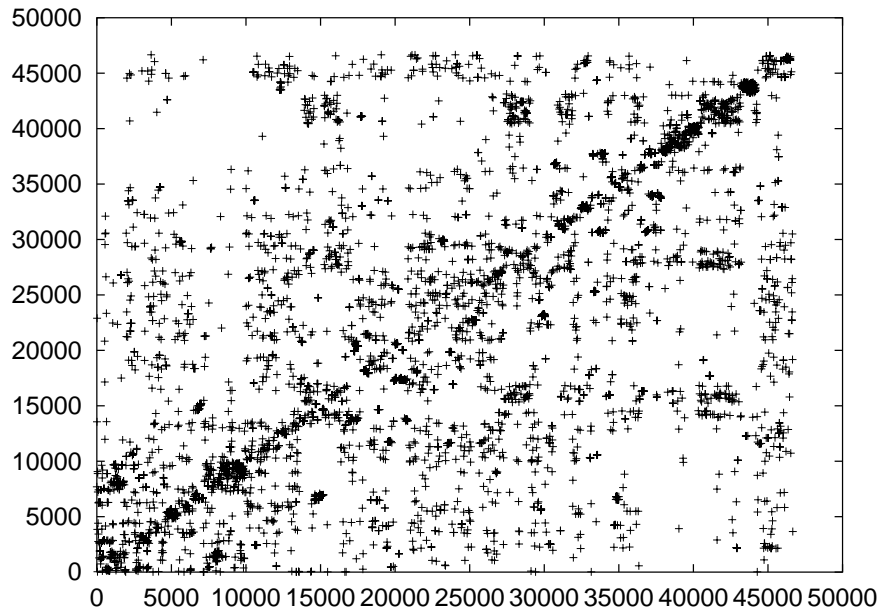
### 5.1 Alarm data

First, to get some feel for the data, we computed all closest slices for every 10th window with  $w = 1000$  using the random mapping approximation. Figure 1 shows the location for the closest match for each slice. Some temporal locality can be observed.

We then created an artificial query window of width 1000 consisting of 37 events of 20 types. None of the event types occurred in the original sequence. We pasted 50 copies of this query window onto the alarm data. The pasting was made transparently, i.e., the original events were left into the sequence. Thus the modified sequence contains some clear copies of the query window, but most of the copies contain also other events. The query window was constructed by copying a part of the sequence and renaming the events.

We ran our mapping algorithm with several values of the parameter  $k$ , and performed queries using the following rule: list all windows in the order of approximate distance to the query window, but when a window  $S$  is listed, forget about all subsequent windows that overlap  $S$ . So, if  $(9000, 10000]$  is listed as one of the resulting windows, none of the windows  $(9000 + i, 10000 + i]$  for  $i = -999, \dots, 999$  are listed. Otherwise, we would get several spurious results, since heavily overlapping windows are naturally similar.

Figure 2 shows the distances of some of the resulting windows for  $k = 5, 10, 15, 30$ . The  $x$ -axis gives the location in the sequence, and the  $y$ -axis gives the approximated distance, i.e., the Euclidean distance in  $k$ -space. The plots have



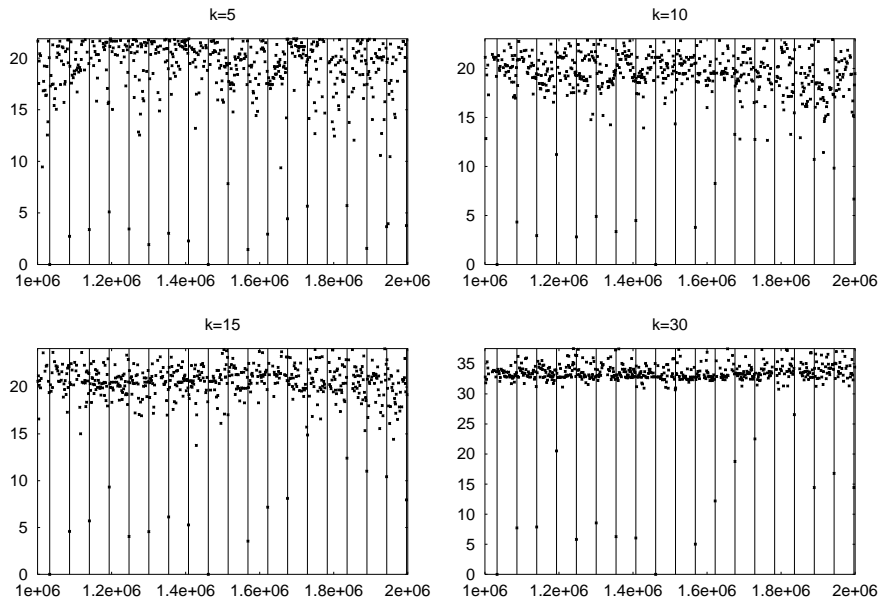
**Figure 1.** *The location (index to the sequence) of the closest slice to the query slice, for every 10th event in the sequence and for  $W=1000$  sec. Here,  $k = 10$ .*

been truncated at a point slightly above the median distance for scaling purposes, since some windows have extremely high distances. For legibility, the horizontal axis shows only a part of the full time scale. The vertical lines indicate the positions of the inserted windows.

The best 50 non-overlapping windows in the case  $k = 15$  are listed in Table 3 (p. 13). The first column gives the computed distance to the query window; in the first three cases, there were no extra events intermingled with the target, so the distance was zero. The second column indicates the position (i.e., time in seconds) of the window, and the third one gives the position of the closest target window, if there is one within the window width 1000. The fourth column is simply the difference of the second and third ones.

Of the 50 target windows, 22 are found exactly, and 20 more overlap one of the best 50 windows. Note how all the distances below 13 correspond to windows close to the targets, and distances up to about 5 are all exact hits (cf. Figure 2).

It seems from the figures, especially from the case  $k = 30$ , that there is no way the method can find all of the inserted windows. Indeed, even with  $k = 100$ , some of the inserts remain indistinguishable from other windows. This is because sometimes the target window is inserted at a place where there are lots of events already, and the resulting window is not very similar to the target.



**Figure 2.** *Distances to selected windows, alarm data*

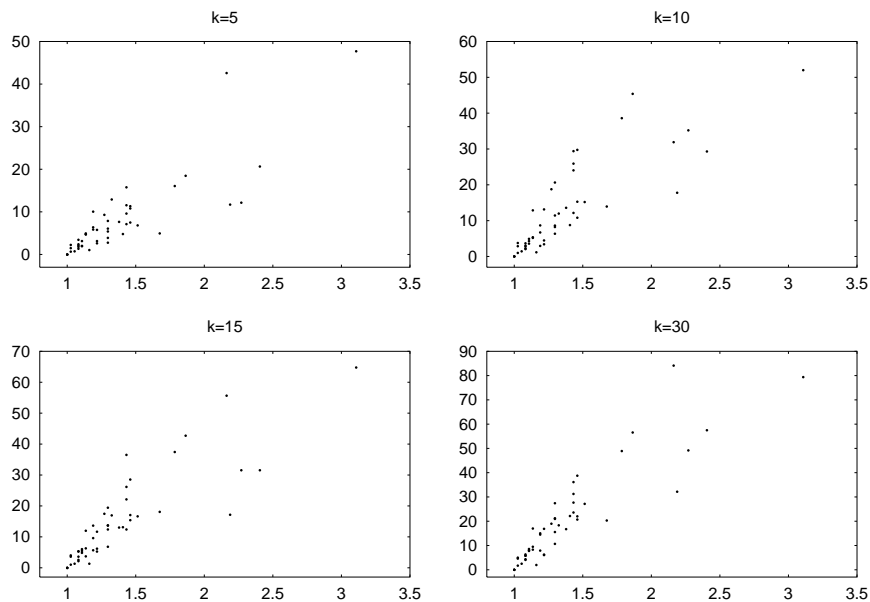
This is illustrated in Figure 3, where the approximated distances of the inserted target windows are plotted against the “density ratio”  $|I'|/|I|$ , where  $|I|$  is the number of inserted events and  $|I'|$  is the actual number of events within the 1000-width target window, counting both inserted events and those that were part of the original sequence.

For each target window  $W$ , either  $W$  appears or one or two windows overlapping  $W$  appear in the sorted list where overlaps have been omitted. We call the “rank” of  $W$  in this listing the rank of the first listed window  $W'$  for which  $W \cap W' \neq \emptyset$ . Ties are resolved arbitrarily to give each window a unique rank. Figure 4 shows the ranks of target windows against the density ratio defined earlier. Ranks greater than 100 are shown as 100.

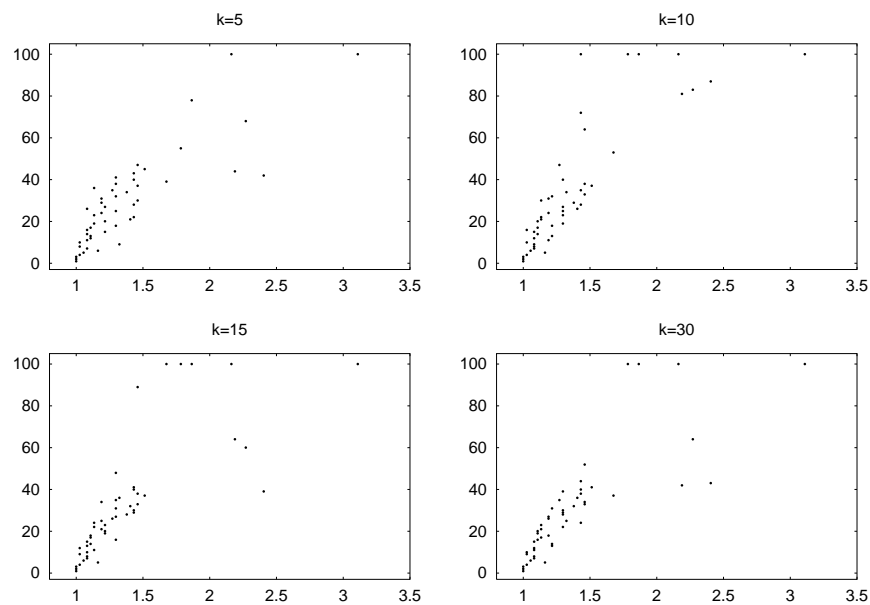
Note the effect of  $k$ : with a larger number of dimensions, the targets stand out better in Figure 2, but the number of targets within the best windows doesn’t increase very much. With this data, it seems that values of  $k$  around 5 or 10 are sufficient.

## 5.2 Entree Chicago data

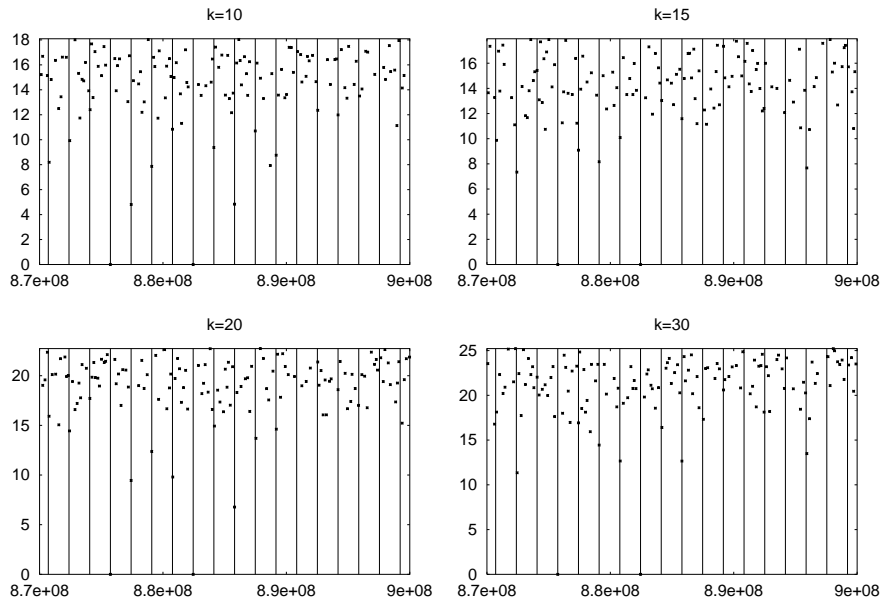
In the case of the Entree Chicago data, we again inserted 50 windows in the sequence. Since the time scale is different, we used 10000 seconds as the window width. In contrast to the previous experiments, the inserts contain mostly event types that also appear in the original sequence: of the 24 events in the insert, 10 are



**Figure 3.** Distances of target windows against the density ratio, alarm data



**Figure 4.** Ranks of target windows against the density ratio, alarm data



**Figure 5.** *Distances to selected windows, Entree Chicago data*

artificial (ones that don't appear in the unmodified sequence). This makes the targets a bit harder to recognize, but they should still be quite far away from other windows. As in the previous case, the insert was obtained by taking a part of the sequence and renaming some events.

Figure 5 shows the distances of windows selected as in the case of alarm data. The values of  $k$  shown are 10, 15, 20 and 30. The plots indicate that several of the targets can be found in this case as well. With this data and this insert, the density ratio of the targets was higher than in the previous case. This is nicely shown in Figures 6 and 7, plotted for the cases  $k = 10$  and  $k = 30$ .

## 6 Concluding Remarks

We have described a simple method for similarity search in sequences of events. The method is based on the use of random projections. We have shown some simple theoretical properties of the method and evaluated its performance on real data. The experiments show that the method is quite good in recognizing similar situations. There are cases where it fails to find a pasted occurrence of the query window, but then there are so many other events mixed within the window that the pasted window would not be considered to be close to the original in the edit distance sense either, assuming some reasonable assignment of costs to the editing operation of inserting events.

Some open problems remain. The theoretical properties of the random mapping applied to sequences of events are largely unexplored. The choice of a linear weighting function  $f$  in this paper is somewhat arbitrary, since also other functions admit an incremental algorithm. Another line of research that we are currently pursuing is the use of the similarity metric in the prediction of future events.

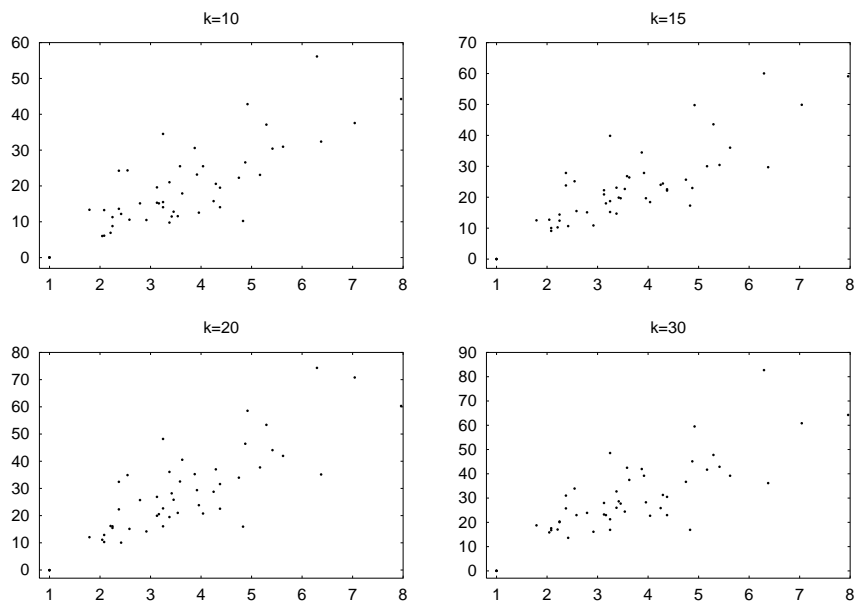
## **Acknowledgement**

The Entree Chicago data was taken from the UCI KDD Archive [4].

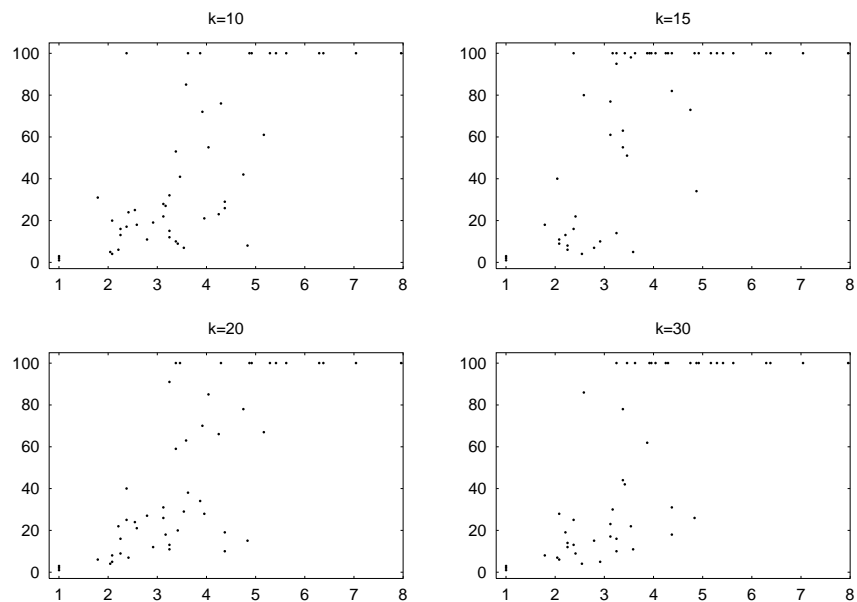
Dist	Window	Closest	Offset	Dist	Window	Closest	Offset
0.00	1461230	1461230	exact	8.11	1675060	1675440	-380
0.00	2157420	2157420	exact	8.41	3014113	3014260	-147
0.00	1032800	1032800	exact	8.57	2799882	2800050	-168
1.02	2210970	2210970	exact	8.58	979102	979249	-147
1.26	497272	497272	exact	9.32	1193557	1193460	97
1.32	711484	711484	exact	10.18	818800	818590	210
2.10	872143	872143	exact	10.26	2639124	2639390	-266
2.59	3067820	3067820	exact	10.42	1942820	1943200	-380
3.55	1568330	1568330	exact	10.49	2853333	2853600	-267
3.61	2425180	2425180	exact	11.02	2478783	2478730	53
3.68	604378	604378	exact	11.02	1889383	1889650	-267
4.00	657931	657931	exact	11.33	2103614	2103860	-246
4.04	1247010	1247010	exact	12.17	2692793	2692940	-147
4.55	1300570	1300570	exact	12.41	1835763	1836100	-337
4.57	925696	925696	exact	12.91	2906893	2907160	-267
4.57	3121370	3121370	exact	13.14	2264140	2264520	-380
4.58	1086360	1086360	exact	13.75	3059438		missed
4.79	2532290	2532290	exact	13.77	1428734		missed
5.27	1407670	1407670	exact	14.08	2959387		missed
5.64	2371432	2371630	-198	14.17	755127		missed
5.70	1139910	1139910	exact	14.43	1961635		missed
5.82	2585840	2585840	exact	14.59	2053796		missed
6.13	1354120	1354120	exact	14.88	1729345	1728990	355
7.17	1621733	1621880	-147	15.01	1116290		missed
7.95	1996493	1996760	-267	15.26	2087183		missed

Dist=Distance to query window  
Window=Position (time) of window found  
Closest=Position of closest target window, if closer than 1000  
Offset=Difference of this window and closest target

**Table 3.** Distances of fifty closest selected windows,  $k = 15$ , alarm data



**Figure 6.** Distances of target windows against density ratio, Entree Chicago data



**Figure 7.** Ranks of target windows against the density ratio, Entree Chicago data

# Bibliography

- [1] R. AGRAWAL, C. FALOUTSOS, AND A. SWAMI, *Efficient similarity search in sequence databases*, Proc. Fourth Intl. Conf. on Foundations of Data Organization and Algorithms (FODO), 1993.
- [2] R. AGRAWAL, K.-I. LIN, H. S. SAWHNEY, AND K. SHIM, *Fast similarity search in the presence of noise, scaling and translation in time-series databases*, Proc. 21st Intl. Conf. on Very Large Data Bases (VLDB), 1995, pp. 490–501.
- [3] R. I. ARRIAGA AND S. VEMPALA, *An algorithmic theory of learning: Robust concepts and random projection*, Proc. 40th Foundations of Computer Science, New York, 1999.
- [4] C. L. BLAKE AND C. J. MERZ, *UCI repository of machine learning databases, 1998*, <http://www.ics.uci.edu/~mllearn/MLRepository.html>, University of California, Irvine, Dept. of Information and Computer Sciences.
- [5] S. DASGUPTA, *Learning mixtures of Gaussians*, IEEE Symp. on Foundations of Computer Science, New York, 1999, pp. 634–644.
- [6] S. DASGUPTA AND A. GUPTA, *An elementary proof of the Johnson-Lindenstrauss lemma*, Technical Report TR-99-006, International Computer Science Institute, Berkeley, CA, 1999.
- [7] G. DAS, H. MANNILA, AND P. RONKAINEN, *Similarity of attributes by external probes*, Proc. 4th Intl. Conf. on Knowledge Discovery and Data Mining (KDD), 1998, pp. 23–29.
- [8] X. GE AND P. SMYTH, *Deformable Markov model templates for time-series pattern matching*, Proc. 6th Intl. Conf. on Knowledge Discovery and Data Mining (KDD), 2000, pp. 81–90.
- [9] D. Q. GOLDIN AND P. KANELLAKIS, *On similarity queries for time-series data: Constraint specification and implementation*, Intl. Conf. on Principles and Practices of Constraint Programming, 1995.
- [10] A. GUTTMAN, *R-trees: A dynamic index structure for spatial searching*, SIGMOD'84, Proceedings of Annual Meeting, ACM Press, Boston, MA, 1984, pp. 47–57.

- [11] E.-H. HAN, G. KARYPIS, V. KUMAR, AND B. MOBASHER, *Clustering based on association rule hypergraphs*, Workshop on Research Issues on Data Mining and Knowledge Discovery, 1997.
- [12] W. JOHNSON AND J. LINDENSTRAUSS, *Extensions of Lipschitz maps into a Hilbert space*, Contemporary Mathematics, 26 (1983), American Mathematical Society, pp. 189–206.
- [13] H. V. JAGADISH, A. O. MENDELZON, AND T. MILO, *Similarity-based queries*, Proc. 14th Symp. on Principles of Database Systems (PODS), 1995, pp. 36–45.
- [14] A. J. KNOBBE AND P. W. ADRIAANS, *Analyzing binary associations*, Proc. 2nd Intl. Conf. on Knowledge Discovery and Data Mining (KDD), 1996, pp. 311–314.
- [15] Y. KAROV AND S. EDELMAN, *Similarity-based word sense disambiguation*, Computational Linguistics, 24 (1998), pp. 41–59.
- [16] H. MANNILA AND P. MOEN, *Similarity between event types in sequences*, Proc. First Intl. Conf. on Data Warehousing and Knowledge Discovery (DaWaK'99), Florence, Italy, 1999, pp. 271–280.
- [17] P. MOEN, *Attribute, Event Sequence, and Event Type Similarity Notions for Data Mining*, Ph.D. thesis, Department of Computer Science, University of Helsinki, Finland, 2000.
- [18] R. MOTWANI AND P. RAGHAVAN, *Randomized Algorithms*, Cambridge University Press, United Kingdom, 1995.
- [19] C. H. PAPADIMITRIOU, P. RAGHAVAN, H. TAMAKI, AND S. VEMPALA, *Latent semantic indexing: A probabilistic analysis*, Proc. 17th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, ACM Press, 1998.
- [20] D. RAFIEI AND A. MENDELZON, *Similarity-based queries for time series data*, SIGMOD Record, 26 (1997), pp. 13–25.
- [21] S. VEMPALA, *Random projection: A new approach to VLSI layout*, Proc. 39th Foundations of Computer Science, Palo Alto, CA, 1998.
- [22] D. A. WHITE AND R. JAIN, *Algorithms and strategies for similarity retrieval*, Technical Report VCL-96-101, Visual Computing Laboratory, UC Davis, 1996.