Characterising Seismic Data

Roel Bertens

Arno Siebes

Technical Report UU-CS-2014-002
January 2014

Department of Information and Computing Sciences
Utrecht University, Utrecht, The Netherlands
www.cs.uu.nl

ISSN: 0924-3275

Department of Information and Computing Sciences
Utrecht University
P.O. Box 80.089
3508 TB Utrecht
The Netherlands

Characterising Seismic Data *

Roel Bertens† Arno Siebes†

Abstract

When a seismologist analyses a new seismogram it is often useful to have access to a set of similar seismograms. For example if she tries to determine the event, if any, that caused the particular readings on her seismogram. So, the question is: when are two seismograms similar?

To define such a notion of similarity, we first preprocess the seismogram by a wavelet decomposition, followed by a discretisation of the wavelet coefficients. Next we introduce a new type of patterns on the resulting set of aligned symbolic time series. These patterns, called block patterns, satisfy an Apriori property and can thus be found with a levelwise search. Next we use MDL to define when a set of such patterns is characteristic for the data. We introduce the MuLTi-Krimp algorithm to find such *code sets*.

In experiments we show that these code sets are both good at distinguishing between dissimilar seismograms and good at recognising similar seismograms. Moreover, we show how such a code set can be used to generate a synthetic seismogram that shows what all seismograms in a cluster have in common.

Keywords: *Frequent Patterns, MDL, Seismogram*

1 Introduction

One of the goals of seismology is to detect and understand the sources and the causes – e.g., earthquakes, volcano eruptions or (man made) explosions – of seismic waves that travel through the earth. One of the main tools for this is the seismometer which produces recordings of earth motion at its location, as a function of time in a so-called seismogram. The type and the location of a (seismic) event is determined by analysing and combining seismograms from multiple seismometers. To a large extend, this identification is still manual labour by a seismologist.

There are large collections of explicitly and/or implicitly labelled seismograms, produced by seismometers all over the globe, of previous events that have been identified. So, it is natural to wonder whether or not the task of the seismologist can be simplified by giving her access to similar, identified, seismograms. That is, given a new, unidentified, seismogram, return a set of similar, identified, seismograms, that help identifying the new seismogram.

To explain what we mean by this we have to make precise what we mean by both "similar seismograms" and by "help identifying". Similar seismograms does *not* mean that their graphs should be (almost) identical. Firstly, because for our purposes seismograms are inherently noisy. That is, the graph is the weighted sum of all events that make the earth move at the location of the seismometer, from the intended event – such as an earthquake – to passing trucks and nearby road-works.

Secondly, even if we would have a clean signal, many of its characteristics depend on much more than just the event. For example, the amplitude measured depends not only on the size of the event, but also on the distance between the event and the location of the seismometer. In fact, many aspects of the graph depend, among others, on the composition of the earth's crust between the event and the seismometer.

The "noise" problem means that we have to somehow clean the seismogram, i.e, we have to filter the intended signal from the noisy graph. Fortunately, it is well known in seismology that the signal in the range from roughly 4 Hz to 1/4 Hz has a fairly good signal to noise ratio [1, 6]. That is, signals in that range are predominately of seismic origin. To decompose a seismogram, we use the discrete Haar wavelet and discard all components outside the 4 Hz to 1/4 range. Note that we use a wavelet decomposition rather than a Fourier decomposition because seismic events are limited in time.

This decomposition gives us a time series of multi-level detail coefficients. Since these coefficients still represent characteristics such as the amplitude of the original signal, we next discretise the wavelet coefficients. That is, we retain that the signal goes up or down, but we do not retain exactly by how much. Since the range of the wavelet coefficients will in general differ for the different frequency levels, we discretise each level separately. The discretisation is based on MDL histogram density estimation [5]; a method which finds the MDL-optimal bin count and cut point locations and is known to have good characteristics.

Preprocessing the data, the details of which are given in Section 2, transforms the original seismogram in a set of aligned categorical time series. Hence the question

*This is an extended and pre-print version of a paper accepted to SIAM International Conference on Data Mining, 24-26 April 2014, Philadelphia, USA.

†Department of Information and Computing Sciences, Utrecht University, {R.Bertens, A.P.J.M.Siebes}@uu.nl.

of when two seismograms are similar is transformed in the question when two such sets are similar. Our answer is simple: when they exhibit similar patterns. The question is then, of course, which patterns?

To answer this question recall that our goal is that the retrieved identified seismograms should help in identifying the new seismogram. When can seismogram x help in identifying seismogram y? Clearly, if knowing seismogram x helps in predicting what will happen next in seismogram y. The more accurate x helps us in predicting what happens next in y, the more similar and useful it is.

Hence, the patterns we are interested in should be such predictive patterns. For example, assume that we have a symbol a on level l at time t. To predict a, Physics tells us [6] we can use a "block" of symbols on all levels at all time points prior to t provided this block contains no holes and it is adjacent to a. Clearly these patterns satisfy an Apriori property and thus can be found with a standard levelwise algorithm.

As for (almost) any kind of pattern, the number of these patterns will quickly explode and most patterns will not be very descriptive of the (transformed) seismogram. To select a small set of descriptive patterns, we use the Minimum Description Length principle (MDL) [3]. That is, similar to the KRIMP algorithm [7], we encode the (transformed) seismogram with a set of patterns. The better a set of patterns compresses the data, the better they (collectively) describe the data. Finding an optimal set of patterns is again intractable and, hence, a heuristic algorithm called MULTI-KRIMP is used. Note that since the behaviour of the seismogram on different frequency levels can be different, *code sets* are computed for each level separately. The details of both the patterns, their discovery and the MULTI-KRIMP algorithm are given in Section 3.

Our claim is now that similar seismograms are seismograms that yield similar code sets. That is, if x and y are similar seismograms, then compressing x with the code set computed from y should be almost as good as compressing it with its own code set and vice versa, of course. Moreover, we claim that if the seismograms are similar – i.e., they compress each other well – their identification is similar – i.e., they indicate similar seismic events.

Since the classification of a seismic event by a seismogram isn't a mathematically defined property – in which case devising algorithms for the task would have been easy – we can not verify our claims formally. Hence, we use experiments to substantiate our claims. But before we describe these experiments, related work is first discussed in Section 4. Most notably *shapelets* [11] are discussed there. Not only because they are a well-known technique in time series analysis, but also because they are suitable for part of what we aim to achieve.

To substantiate our claims, we use data from different seismic events and seismometers at different locations. We show that our technique clusters the events correctly regardless of the location (and size) of the event and the location of the seismometers. Moreover, we illustrate visually that our technique captures the shape of a seismogram. For, given a code set, we can generate artificial seismograms. By plotting both the real seismogram and a generated one, one can see that they are similar. The details are given in Section 5. The discussion of these results is given in Section 6. Finally, the conclusions are formulated in Section 7.

2 Preprocessing the Data

Seismograms can be seen as functions from time to the real numbers, or more formally, $S : \mathbb{R}_{\geq 0} \to \mathbb{R}$. In reality, of course, the signal at a location is only sampled at some frequency rather than measured continuously. We come back to this point later.

In the Introduction we already explained why we cannot use S directly, it is noisy and characteristics such as the amplitude should be removed. Hence, S is first decomposed using a wavelet transform and subsequently the coefficients are further discretised. Both steps are discussed in detail in this Section.

2.1 Wavelet Decomposition

There are a number of techniques to decompose a function in frequency components, such as the Fourier decomposition and wavelet decompositions [2]. The advantage of a wavelet decomposition over the Fourier decomposition is that wavelets are localised and, thus, better at describing local behaviour in the signal. Since seismic events are by nature local events in the time series, we use a wavelet transform rather than Fourier analysis.

Formally, a wavelet transformation is the convolution, i.e., an inner product, of the function f with a scaled (parameter s) and translated (parameter b) wavelet ϕ:

$$Wf(s,b) = \langle f, \phi_{s,b} \rangle = \frac{1}{s} \int f(x) \phi \left(\frac{x-b}{s} \right) dx$$

A wavelet decomposition is computed by a set of wavelet transforms. Since we have sampled data, we use a discrete wavelet transform and in that case the decomposition is computed as follows. We assume that we have (a window of) data with 2^N data samples $f(t_1), \ldots f(t_{2^N})$ of f, (with time normalised such that $[t_1, t_{2^N}] = [0, 1]$). The discrete wavelet decomposition is now given by:

$$f = f^0 + \sum_{m=0}^{N} \sum_{l=0}^{2^m} \langle f, \phi_{m,l} \rangle \phi_{m,l}$$

where f^0 is the coarsest approximation of the time series.

This decomposition allows us to built a ladder of approximations to f given by:

$$f^{j-1} = f^j + \sum_{k=0}^{2^j} \langle f, \phi_{j,k} \rangle \phi_{j,k}.$$

This ladder gives us two sets of coefficients, the approximation coefficients (the f^j) and the detail coefficients ($f^{j-1} - f^j$). The detail coefficients encode the local behaviour of the time series at level j. Hence, these are the coefficients we will be using.

Each wavelet family has different properties, bringing out different aspects of the time series. Since we are interested in the shape of the graph, we use the Discrete Haar wavelet:

$$\phi(x) = \begin{cases} 1 & \text{for } 0 \leq x < 0.5 \\ -1 & \text{for } 0.5 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

Using the Haar wavelet the approximation coefficients are averages on progressively smaller subsets of (a window of) the time series, while the detail coefficients are the local deviations from those averages.

To be more concrete, consider the following toy example of a time series and its coefficients:

	detail coef.	approx. coef.	original signal
1st level:	1 -1 -1 1	8 4	9 7 3 5
2nd level:	2 -2	6	

The average of 9 and 7 is 8, and the deviation of 9 from this average is $9 - 8 = 1$. Similarly, the average of 3 and 5 is 4 and the deviation of 3 from this average is 3 - 4 = -1. Finally, the average of 8 and 4 is 6 and the deviation of 4 from this average is 4 - 6 = -2. Note that each original data point can be reconstructed using the coarsest approximation, the overall average, 6 and a sequence of detail coefficients, e.g., for the rightmost point we add the rightmost coefficients: $6 - 2 + 1 = 5$.

Further note that for technical reasons the actual coefficients at each level should be multiplied by $\sqrt{2}$, but that doesn't concern us here.

Given a seismogram S which may have any length, we compute the wavelet coefficients using a sliding window w, which has some dyadic (power of 2) length. Note that this implies that we do not compute coefficients for the first $|w| - 1$ data points.

Assume that w starts at position 1 in S and ends at $|w|$. We then compute the detail coefficients from the Haar decomposition of $S_1, \ldots, S_{|w|}$ as indicated above. We also noted above that it is the right most set of detail coefficients that tell something about the local behaviour of S at $S_{|w|}$. Hence we start our aligned (transformed) time series with these rightmost detail coefficients. Next we shift the window by a step (for us, always a step of size 1) and repeat the procedure to compute the next set of coefficients for $S_{|w|+1}$.

As an example, consider the following time series:

1 1 4 4 4 8 1 4 7 7 9

If we use a window size of 8 and a step size of 1, we have four windows on this signal. For real seismic data, we are only interested in the detail coefficients that represent the frequencies in or close to the interval 4 Hz to $1/4$ Hz, as discussed in the Introduction. Here, however, we simply retain all detail coefficients. Hence, our time series – which allows for four windows – is transformed to the following set of aligned time series.

window 1	*window 2*	*window 3*	*window 4*
-2.12	-2.12	0	-1.41
3.5	-1	-4.5	-2.5
-2.47	-2.47	0.35	-3.54

Computing all coefficients for all windows is wasteful, however optimising this is not the focus of this paper.

2.2 Discretisation

Together with the coarsest approximation coefficient the detail coefficients are sufficient to reconstruct the original time series. As already stated, we will not use all detail coefficients and neither will we use the coarsest approximation coefficient, but this observation means that the detail coefficients are too detailed for our purposes. We are interested in the general shape of the graph, not in its exact values. That is, we are interested in whether it is ascending or descending at various frequency levels. We may even be interested in whether it is ascending steeply or barely on a given frequency level, but we are not interested in how steeply it is rising exactly.

Hence, we discretise the detail coefficients we just computed as the next preprocessing step. Since the spread of the coefficients may very well be different for each of the levels, we discretise the levels separately. But, of course, we use the same discretisation for all seismograms.

There are many techniques to discretise data, each with its own strong and weak points. For this paper, we use one based on the Minimum Description Length (MDL) principle; we will briefly discuss MDL in the next section. More in particular, we use MDL histogram density estimation [5]. This method finds the MDL-optimal number of bins and cut point locations automatically.

For each level l we have a set of symbols – alphabet – A_l, which has one symbol for each bin the discretisation produces for l. Each value on level l of the aligned time series, produced by the wavelet transformation, is replaced by its corresponding element from A_l. In the end our original time series S is thus transformed in a set of aligned symbolic time series.

3 Patterns and Code Sets

Given the preprocessed data – the set of aligned symbolic time series – we now have to discover a small set of characteristic patterns for each level of the preprocessed data. The definition and discovery of the patterns is a standard pattern mining problem. For the second step, the discovery of a small set of *characteristic* patterns, we use MDL.

3.1 Patterns

As noted in the introduction, our predictive pattern occurrences should have no holes across either the time or the level axes. For our patterns that means the following.

A set of sequences $P = \{p_l, \ldots, p_m\}$ is a *block pattern* over the level alphabets $A_1, \ldots, A_n$ iff

- For each $j \in \{l, \ldots, m\}$, p_j is made from symbols in A_j.
- $\{l, \ldots, m\}$ is a consecutive subset of $\{1, \ldots, n\}$.

The second requirement goes a long way to ensure that we get no holes in occurrences, but we need one more requirements to ensure it properly. This is a requirement on what constitutes an occurrence of P.

Let S be a set of aligned time series over the level alphabets $A_1, \ldots, A_n$ and $P = \{p_l, \ldots, p_m\}$ a block pattern over those same alphabets. P occurs in S at time t iff for every $p_k \in P$ there is a consecutive subsequence of S at level k, $[S_k[t - |p_k|], \ldots, S_k[t]]$ such that:

$$\forall j \in \{1, \ldots, |p_k|\} : p_k[j] = S_k[t - |p_k| + j]$$

That is the pattern sequences should, of course, match their respective elements in S exactly and all the pattern sequences occurrences should end at the same time t.

To give an example, assume that in our example transformed time series from Section 2.1, the number -2.1 is replaced by the abstract symbol -2.1 and so on (in other words, assume for a moment that the numbers are labels). Then the patterns 1 and 2 below occur and the patterns 3 and 4 do not; their only potential occurrences exhibit holes.

pattern 1		*pattern 2*		*pattern 3*		*pattern 4*
-2.12	0	-1	-4.5	-1	-2.5	-1.41
	-4.5	-2.47	0.35			-3.54

With these definitions, the support of P in S is defined in the usual way, viz., its number of occurrences. Moreover it is clear that block patterns satisfy an Apriori principle: if P_1 is a sub-pattern of P_2, the support of P_1 will be at least as big as that of P_2. Hence, all frequent block patterns in S can be discovered with levelwise search [4].

3.2 Code Sets

As usual in pattern mining, if we set our threshold low, there are enormous numbers of frequent block patterns. How do we choose which ones are characteristic for a (preprocessed) seismogram? The first clue is that, as noted in the Introduction, our patterns should be predictive. That is, if a pattern occurs at time t in[1] S it should help us in predicting what happens at time $t+1$ in S.

Next observe that this gives us a way to encode S. Let P be some one level pattern, i.e., $P = \{p_l\}$. Furthermore, assume that we observe that if P occurs in S at a time t, $S_l[t+1]$ is either the symbol a or the symbol b with probability p_a and p_b respectively. To encode, or compress, S we could now replace a's and b's that occur right after P in S with a codeword of length $-\log(p_a)$ and $-\log(p_b)$ respectively, [3].

Clearly, it is slightly more complicated as there will be many patterns that are followed by an a and patterns will span multiple levels. But, the main idea that we can use patterns to encode the data is obviously valid. And that gives our second clue to how to choose: we can use the Minimum Description Length principle (MDL) [3].

MDL is a method for inductive inference that aims to find the best model to describe your data. It is based on the insight that any regularity in the data can be used to compress the data and the more we can compress, the more regularity we have found.

More formally, given a set of models $\mathcal{M}$, the best model $M \in \mathcal{M}$ is the one that minimises

$$L(M) + L(\mathcal{D} \mid M),$$

in which $L(M)$ is the length in bits of the description of M, and $L(\mathcal{D} \mid M)$ is the length of the description of the data when encoded with model M.

Given that each level S_l of S has its own alphabet A_l, we encode each level separately. The simplest way to encode S is by disregarding all patterns and simply use a (prefix) code that reflects how often each $a \in A_l$ occurs in S_l. That is, we give it a code of length $-\log p(a \mid S_l)$. This is what we call the standard encoding ST_l of S_l and by doing this on all levels we have the standard encoding ST of S.

Let $\mathcal{P}^S$ be the set of all frequent block patterns on S. A $P \in \mathcal{P}^S$ is said to be a level l pattern if one of the sequences in P is built from A_l. The set of all frequent level l patterns in S is denoted by $\mathcal{P}_l^S$. We will simply write $\mathcal{P}$ and $\mathcal{P}_l$ if S is clear from the context.

To simplify the remainder of this section, and indeed the rest of this paper, we augment each $\mathcal{P}_l$ with the special pattern $\emptyset$, which matches every element of S_l and even no element at all.

[1] We will use S both to denote the original and the preprocessed time series.

A *covering set* C_l for S_l is an ordered subset of $\mathcal{P}_l$ which contains $\emptyset$ as its last element. The *cover* of S by C_l is again a time series, denoted by $C_l(S_l)$, in which

> $C_l(S_l)[t]$ is the first element of C_l that occurs at time $t-1$ if $t > 1$, otherwise $C_l(S_l)[t] = \emptyset$.

To turn a cover of S_l into an encoding of S_l, we augment each element of a covering set C_l with a *code table* [7]. This code table consists of two columns. The first contains the elements of A_l in some order. The second contains a code word from some prefix code $\mathcal{C}_{C_l}$. Since we want to compress S_l, $\mathcal{C}_{C_l}$ has to be optimal for this compression. This is determined as follows.

For $a \in A_l$ and $P \in C_l$, define:

$$usage(a \mid P) = |\{t \mid S_l[t] = a \wedge C_l(S_l)[t] = P\}| + 1$$

which gives us:

$$Pr(a \mid P) = \frac{usage(a \mid P)}{\sum_{b \in A_l} usage(b \mid P)}$$

And thus, the code $\mathcal{C}_{C_l}$ should assign to a in the code table of $P \in C_l$ has length $-\log(Pr(a \mid P))$.

A *level code set* CS_l is a covering set C_l for S_l in which each pattern in C_l is augmented with a code table for A_l with the optimal codes as constructed above. A *code set* CS for S is simply a set of level code sets, one for each level S_l in S.

Coding S_l with CS_l is simple. First compute $C_l(S_l)$, and then replace $S_l[t]$ by its code as given in CS_l in the code table of $C_l(S_l)[t]$. Note that the standard encoding ST_l of S_l that we introduced above is simply the encoding induced by the covering set $C_l = \{\emptyset\}$ plus a Laplace correction. From now on, we use this Laplace corrected version as the standard encoding.

The encoded size of the data given CS_l, denoted by $L(S_l \mid CS_l)$ is now simply the sum of the code lengths of the codes in the encoded string.

To find the optimal encoding for S_l according to the MDL principle, we also have to determine the size of CS_l. This is determined as follows:

- For each of the codes in the code tables we have a length
- For each of the elements of A_l in those code tables we use the standard encoding ST_l
- Each of the patterns P is also encoded with the standard encoding. Each sequence in P is, of course, encoded by the standard encoding at the appropriate level.

By summing all these encoded lengths, we get the total encoded size of the model, denoted by $L(CS_l \mid S)$.

MDL now tells us that we need to find the code set CS_l such that

$$l(CS_l, S_l) = L(CS_l \mid S) + L(S_l \mid CS_l)$$

is minimised. Unfortunately, as in [7], this is an intractable problem. Firstly because of the order used in covering, every permutation will lead to another compressed size. Secondly because adding a pattern to the level code set may both increase and decrease the compressed size. In other words, there is no structure in the search space that allows us to prune large parts of it. Hence we need to resort to heuristics.

3.3 MuLTi-Krimp

We adapt the heuristics used in [7], if only because they proved to work well. The first heuristic is that we define the order of the patterns in a level code set. These patterns are assumed to be ordered by the **Standard Cover Order**. Which is descending on cardinality first, descending on support second, ascending on height third, descending on top-level-length fourth, and last lexicographically ascending to make it a total order.

Secondly, we use a simple greedy algorithm to find good level code sets, called MuLTi-Krimp, the pseudocode is given in Algorithm 1. As input it takes the (preprocessed) time series S, a level l and the frequent block patterns P_l.

We start with the level code set containing only the empty set (1). We loop through all frequent patterns in **Standard Candidate Order** (like the Standard Cover Order, only sorted on support before cardinality) (2); we then add each pattern to our code set (3) and test if this new code set compresses our data better than before (4). If it does compress better, we keep the pattern and consider if other patterns in our code set can be pruned (5). After we have considered all patterns, we return the final level code set (8).

Algorithm 1 The MuLTi-Krimp Algorithm

Input: A preprocessed seismogram S, a level l, and a set of frequent patterns for level l, $\mathcal{P}_l$.
Output: A level CS_l

1: $CS_l \leftarrow \{\emptyset\}$
2: **for** $P \in \mathcal{P}_l$ in **Standard Candidate Order do**
3: $\quad CS_l^c \leftarrow (CS_l \oplus P)$ in **Standard Cover Order**
4: $\quad$ **if** $L(CS_l^c, S_l) < L(CS_l, S_l)$ **then**
5: $\quad\quad CS_l \leftarrow$ *post-prune*(CS_l^c)
6: $\quad$ **end if**
7: **end for**
8: **return** CS_l

4 Related Work

Our use of wavelets to decompose a time series is, of course, far from unique. In fact, we have used the discrete Haar wavelet ourselves before in [8]. An overview

of all possible ways to decompose time series data is far beyond the scope of this paper. For more on wavelets, we refer to [2]. Also for the discretisation of real valued data there are far too many techniques to even attempt an overview here. We chose MDL histogram density estimation [5] because it finds both the number of bins and the bins themselves automatically in a well founded manner. Moreover, it is known to work well in practice.

Frequent pattern mining in time series data is useful for tasks such as clustering, classification, prediction, and many more. All these fields have attracted much research from the data mining community. Our block patterns are somewhat unusual in that they span multiple frequency scales of a decomposed and discretised time series, but mining them is completely standard. For a brief overview of frequent pattern mining we refer to [4].

Techniques for clustering time series data fall mainly in three groups: those that work directly with the raw data, those that work with features extracted from the raw data, and those that work with models build from the data; an overview of time series clustering can be found in [9]. Our work fits in the intersection of the second and third category, because we cluster data samples based on their size when compressed with a range of code sets computed on discretised wavelet coefficients of the original data.

A good example of a time series clustering approach that works directly on the raw data is [11]. It uses *shapelets*, which are time series snippets characteristic for a certain cluster of data. Time series chunks are clustered using the distance to a shapelet, rather than the distance to the nearest neighbour chunk. This technique has proven to work very well in many different domains. Unfortunately, for our data – which can be very noisy and repetitive – it didn't always do very well; for more details see the experiments and the discussion thereof.

The way we use MDL to identify characteristic patterns is, of course, reminiscent of our earlier work on Krimp in [7] and many follow-up papers. While the high-level approach here is similar to that, many aspects are very different. Probably the biggest difference is that the current encoding is completely different. Here we compress for predictive capabilities rather than for descriptive ones. This means firstly, that the patterns we use to cover a value $S_l[t]$ do *not* contain $S_l[t]$ itself. Rather a pattern describes the behaviour of S just before $S[t]$. Secondly, it means that the code lengths are determined by conditional probabilities rather than by simple relative occurrence frequencies. Third and finally, it means that patterns cover exactly a single value $S_l[t]$ and never a larger subset of a preprocessed time series.

Application-wise, the Dynamic Bayesian Network (DBN) approach to seismic data classification in [6] is closely related to our research. They also decompose the signal using wavelets – albeit a different one: the Morlet wavelet – and then use a DBN to classify the incoming data as either "Earthquake" or "Noise". The most important difference is that we do not need predefined classes. Using MuLTi-Krimp we can both determine which clusters there are in the data and classify new data as belonging to any of these clusters or being something not seen before.

5 Experiments

To evaluate the code sets produced by MuLTi-Krimp experimentally, we perform three (sets of) experiments. Firstly to show their ability to distinguish between different types of seismograms. secondly to show that they can be used to identify similar seismograms. Finally we show that code sets can be used to generate synthetic seismograms and that these generated seismograms are visually similar to the seismogram the code set was computed from.

For reproducibility and to stimulate future research, both the code and the datasets are available through the first author.

5.1 Setup

For the wavelet transformation we use a step size of 1 and a window of 256 data points. Since all the seismograms used are sampled at 40 Hz, we use the levels 4 (corresponding to 2.5 Hz) to 8 (corresponding to 0.16 Hz). As noted before, the discretisation requires no parameters.

Since experiments show that larger patterns are hardly, if ever, used, we limit the patterns we mine to those that span across at most three frequency levels and two time points.

5.2 Datasets

In cooperation with a domain expert, we manually gathered seismograms from the publicly available ORFEUS POND data repository [10].

For the first experiments we collected seismograms of 2000 data points with varying frequencies and amplitudes. It serves to show the ability our method has to distinguish between only slightly different seismograms. It consists of seismograms at various moments and at different locations. We manually distinguished six different clusters, all containing four very similar seismograms, see Figure 1.

For the second experiment we gathered seismograms from two different events both measured at the same 12 stations. The first was a quake in the sea of Okhotsk of magnitude 6.4, the second was a quake in Pakistan of magnitude 7.7. We split each of these seismograms into

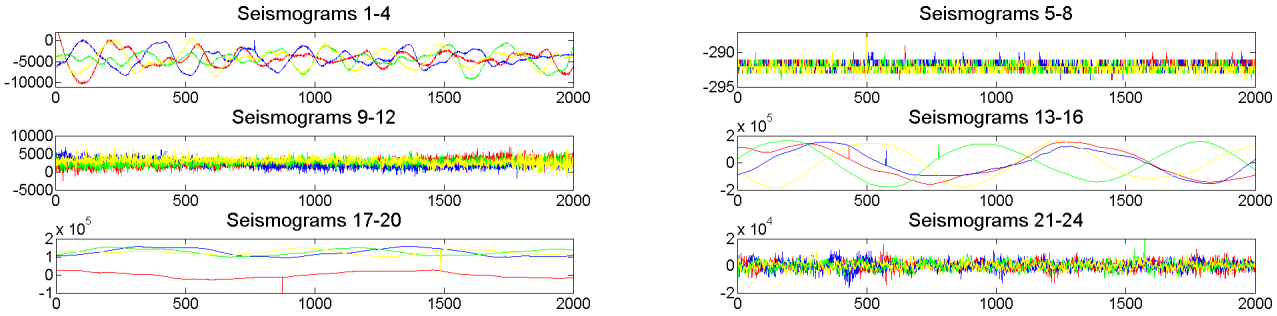

Figure 1: Six clusters, all containing four seismograms.

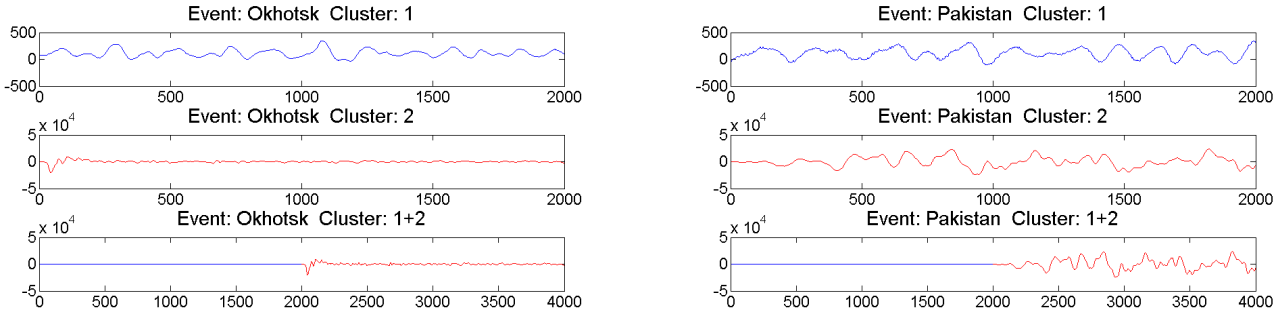

Figure 2: Seismograms from the Brant station for both events.

2000 data points before the quake and 2000 beginning at the start of the quake. That is, for each event we have two clusters of seismograms, Cluster 1: no event, Cluster 2: quake.

Data from both events are plotted for one station in Figure 2. Due to space limitations, the data from all other stations can be found in Appendix A. The seismograms from the event at Okhotsk in cluster 1 are numbered 1-12 (Figure A9) and those in cluster 2 are numbered 13-24 (Figure A10). For the event in Pakistan the seismograms 25-36 are in cluster 1 (Figure A11) and 37-48 in cluster 2 (Figure A12). In Figure 3 the locations of the two events and of all stations involved are given.

Figure 3: The location of the events and the stations.

5.3 The Power to Distinguish

Using MuLTi-Krimp we build a code set for each of the seismograms from the first dataset. Then each seismogram was encoded with the code set of every other seismogram. This gave a table in which a row indicates a seismogram and a column a code set of one of the seismograms. In this table, each cell contained the size of the corresponding seismogram compressed with the corresponding code set. This table was used to hierarchically cluster the seismograms using single linkage and the Spearman metric. The resulting dendrogram is shown in Figure 4. The numbers on the x-axis represent seismograms, where every four subsequent numbers (1-4, 5-8, 9-12, 13-16, 17-20, 21-24) represent seismograms that we manually identified to be very similar, see also Figure 1. Note that all seismograms are clustered as we would expect, the distance between seismograms which are alike is relatively small compared to the distance to other seismograms. Clustering the seismograms using shapelets and k-means clustering gave rather less convincing results.

To further substantiate our claim, we also did a leave-one out validation. That is for each time series, we clustered the remaining 23 clusters manually, computed a code set for each of the clusters and determined the "right cluster" for the left out time series by choosing the cluster whose code set compressed it most. Each of the seismograms was assigned to its own cluster. This confirms our claim that code sets are good in distin-

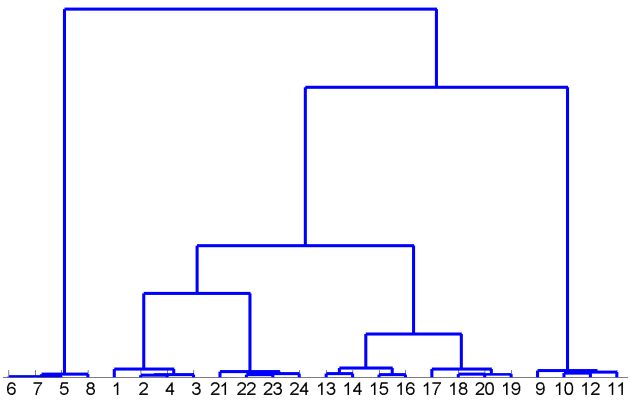

Figure 4: A dendrogram clearly representing the six clusters.

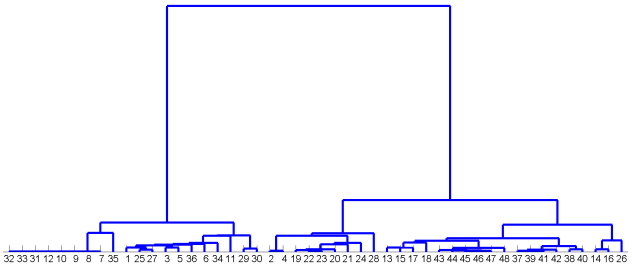

Figure 5: A dendrogram for all seismograms from two clusters of two events.

guishing between different seismograms.

5.4 Identifying Similar Seismograms

With the second dataset – the seismograms related to the two quakes – we first redid the first experiment. That is we again build a table of the respective compressed sizes and performed a clustering based on that table, see Figure 5. All seismograms from cluster 1 are easily distinguished from cluster 2. However, there are four mistakes; the seismograms 2, 4, 26 and 28 from cluster 1 are clustered with the data from cluster 2. This can be explained by the fact that these seismograms show a sizeable amount of movement, as can be seen in (Figures A9 and A11). Experiments using shapelets gave comparable results.

Next we assigned all seismograms manually to three clusters, no event, quake in the sea of Okhotsk and quake in Pakistan respectively. Then we performed again a leave one out validation. This gave near perfect results. Only one non-event, seismogram 26 (which shows significant movement) was assigned to the event in Okhotsk. Moreover, two seismograms from the event at Okhotsk are clustered with the event in Pakistan, to which they apparently are more similar. Note that from the point of view of identification only the mistake with seismogram 26 is a real mistake.

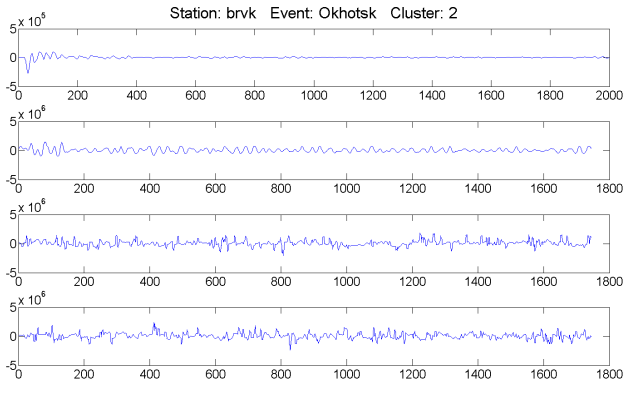

Figure 6: Visualisation of the code set for a seismogram.

5.5 Generating Seismograms

Given that the patterns in a code set predict what is going to happen next, we can use a code set CS to generate a seismogram. Recall that $\emptyset$ matches anything – including nothing – so, for $t = 1$ we choose random symbols drawn with probabilities according to its (i.e., $\emptyset$) code table on each level. For each next step, on each level, we use its level code set to determine which pattern matches the preceding time series and draw a new symbol according to the code table of that pattern.

This gives a series of aligned symbolic time series. Next we translate each symbol into a number by replacing it with the middle-value of its associated bin. Finally, we sum the values at all levels at the same time point and smooth this single level signal by combining each five consecutive values as compensation for using only the middle-value of each discretisation bin. This gives us a synthetic time series, say T.

Now CS is, of course, computed from a seismogram, say S. If CS is characteristic of S, T should resemble S. Well, it should resemble S, when all the details of the frequency levels not in $\{4, 5, 6, 7, 8\}$ are filtered out of S.

To test this we did this experiment for a few of the seismograms. One of these tests is pictured in Figure 6. The top plot is the original seismogram. The second one is the signal as present on the frequency levels 4 to 8, i.e., 2.5 Hz to 0.16 Hz. The next two plots are generated using the procedure above.

6 Discussion

The whole procedure of wavelet decomposition, discretisation, and building a code set using MuLTiKrimp serves one purpose: to characterise a seismogram. The experiments substantiate that this is indeed the case.

The experiments on the first dataset show that our code sets perform well in distinguishing between visu-

ally different seismograms. In fact, as noted in the previous section, they perform better than the state of the art method [11] based on shapelets.

The experiments on the second set of data show that the code sets perform also well in clustering similar events together, while simultaneously distinguishing between different events. In this case it performed on par with the shapelet based method from [11]. An important aspect of this experiment is that it shows that, at least in this case, the characterisation is independent of, the event size, the location of the event, and of the location of the seismometers. This is important since as we already discussed in the Introduction many of the characteristics of a seismic signal *are* dependent on these three aspects.

Hence, code sets are both good in the recognition of similar seismograms and in the distinction between similar seismograms. In other words, code sets are characteristic for seismograms.

This is further corroborated by our third experiment. Here we see that the artificial seismograms one can generate from a code set visually resemble the seismograms from which the code set itself was computed. This is important to show to the seismologist what exactly are the characteristics of a given cluster of seismograms. Clearly, one can always present the seismologist with a center of a cluster. The strong point of a generated seismogram is, however, that it shows what all the seismograms have in common. That is something that is hard to infer from a representative seismogram.

7 Conclusion

The goal of this paper is to find a method to characterise seismograms, such that the identification of an event in a new seismogram (is it an earthquake, a passing truck, or something else) can be facilitated by finding similar seismograms that have already been identified.

To achieve this goal, we devised a method that first preprocesses a seismogram using a wavelet decomposition and discretisation and then uses a new algorithm called MuLTi-Krimp, to discover a code set that contains characteristic patterns for that seismogram.

The experiments show that these resulting code sets are indeed characteristic of a seismogram. They are good in both the recognition of similar seismograms and in the distinction between similar seismograms. Moreover, they can be used to generate a synthetic seismogram that shows what all seismograms in a cluster have in common.

The final conclusion is two seismograms S_1 and S_2 are similar if we have for their MuLTi-Krimp's code sets CS_1 and CS_2 that $CS_1(S_1) \approx CS_2(S_1)$ and $CS_1(S_2) \approx CS_2(S_2)$.

Acknowledgements

This publication was supported by the Dutch national program COMMIT.

References

[1] K. Aki and P.G. Richards. Quantitative Seismology, 2nd ed., *University Science Books*, 2009

[2] I. Daubechies. Ten Lectures on Wavelets. *SIAM*, 1992.

[3] P. Grünwald. The Minimum Description Length Principle. *MIT Press*, 2007.

[4] J. Han, H. Cheng, D. Xin, and X. Yan. Frequent pattern mining: current status and future directions. In *Data Min. Knowl. Disc.* 15(1):5586, 2007.

[5] P. Kontkanen and P. Myllymäki. MDL Histogram Density Estimation. In *Proc. AISTATS*, San Juan, Puerto Rico, USA, March 2007.

[6] C. Riggelsen, M. Ohrnberger, and F. Scherbaum. Dynamic Bayesian Networks for Real-Time Classification of Seismic Signals. In *Know. Disc. in Databases: PKDD 2007*, pp 565-572, 2007.

[7] A. Siebes, J. Vreeken, and M. van Leeuwen. Itemsets that compress. In *Proc. SDM*, pp 393-404, 2006.

[8] Z. Struzik and A. Siebes. The Haar Wavelet Transform in the Time Series Similarity Paradigm. In *Proc. PKDD*, pp 12-22, 1992

[9] T. Warren Liao. Clustering of time series data a survey. In *Pattern Recognition*, Volume 38, Issue 11, pp 1857-1874, 2005.

[10] *ftp://www.orfeus-eu.org/pub/data/POND/*

[11] J. Zakaria, A. Mueen, and E. Keogh. Clustering Time Series using Unsupervised-Shapelets. In *Proc. IEEE ICDM*, pp 785-794, 2012.

Appendix A Seismograms

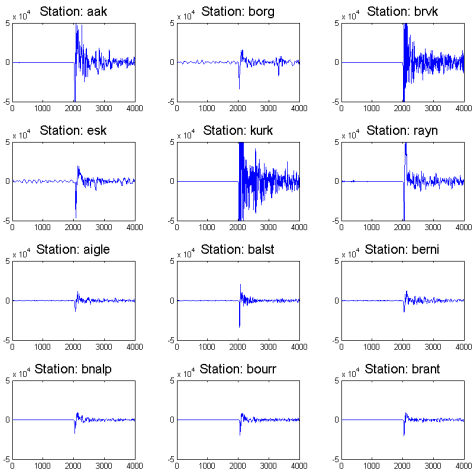

Figure A7: Seismograms from the event at Okhotsk.

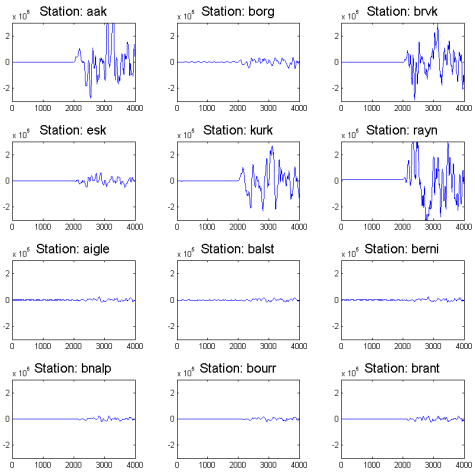

Figure A8: Seismograms from the event at Pakistan.

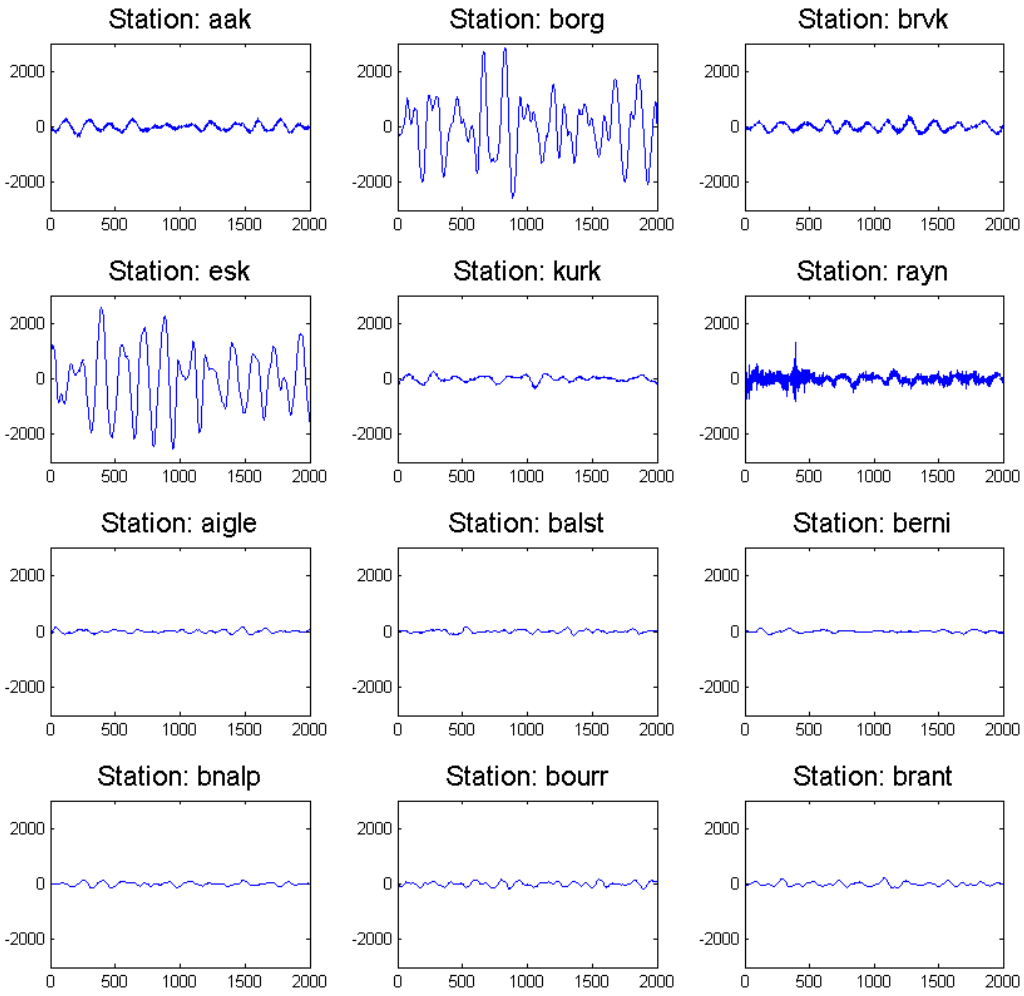

Figure A9: Seismograms from the event at Okhotsk from cluster 1. Numbered 1-12 from left to right and top to bottom respectively.

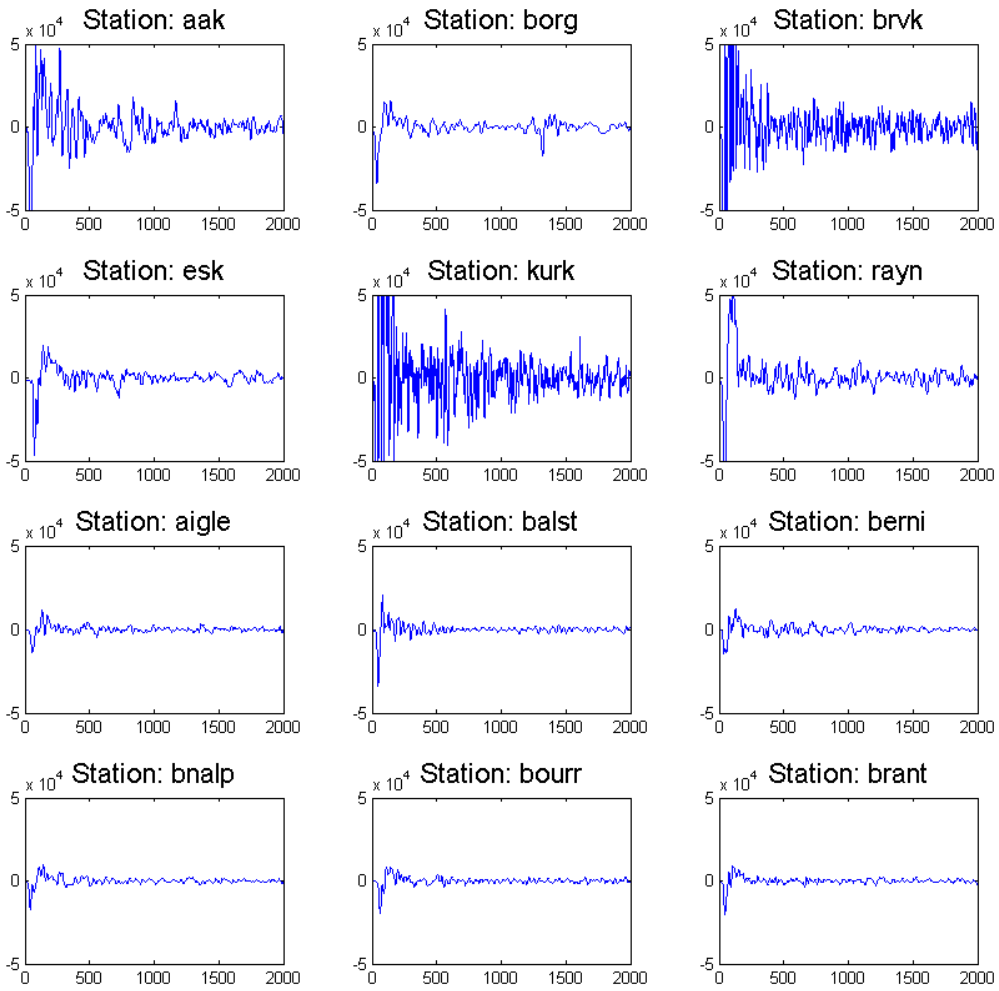

Figure A10: Seismograms from the event at Okhotsk from cluster 2. Numbered 13-24 from left to right and top to bottom respectively.

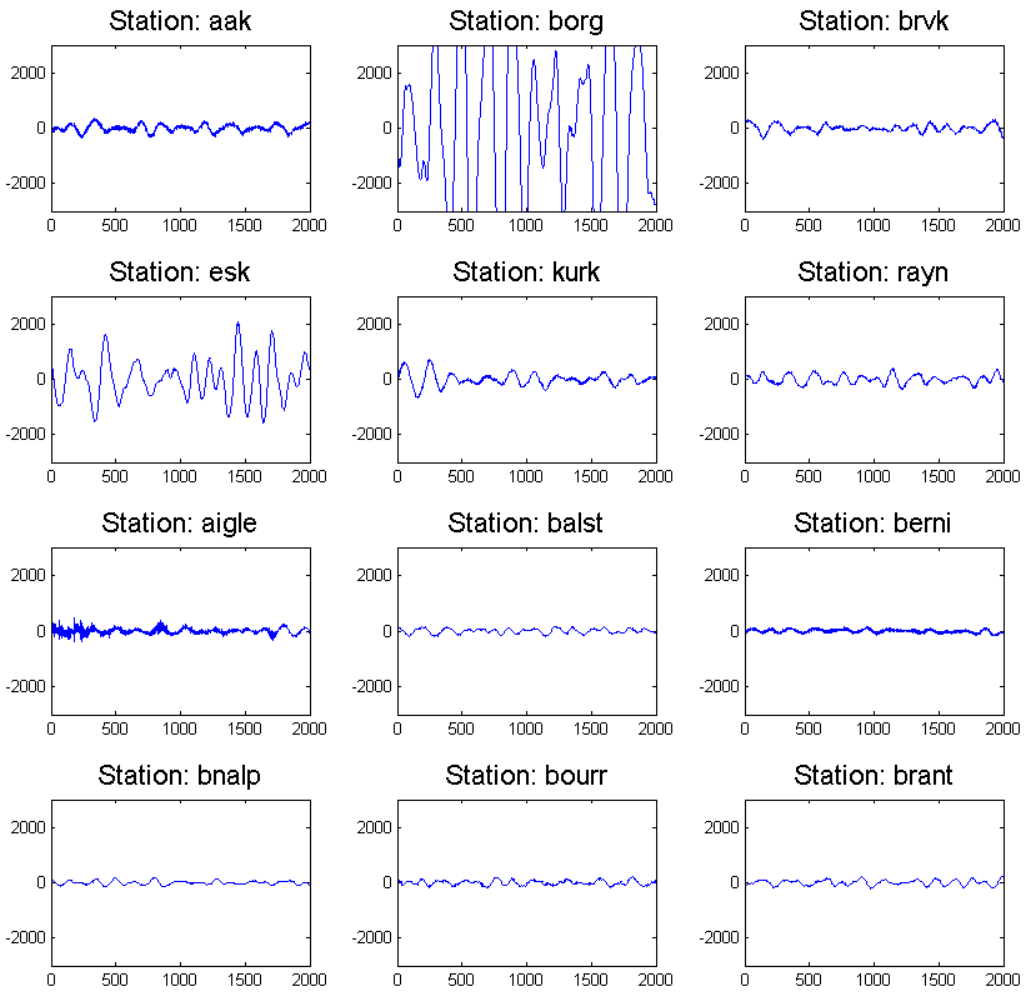

Figure A11: Seismograms from the event at Pakistan from cluster 1. Numbered 25-36 from left to right and top to bottom respectively.

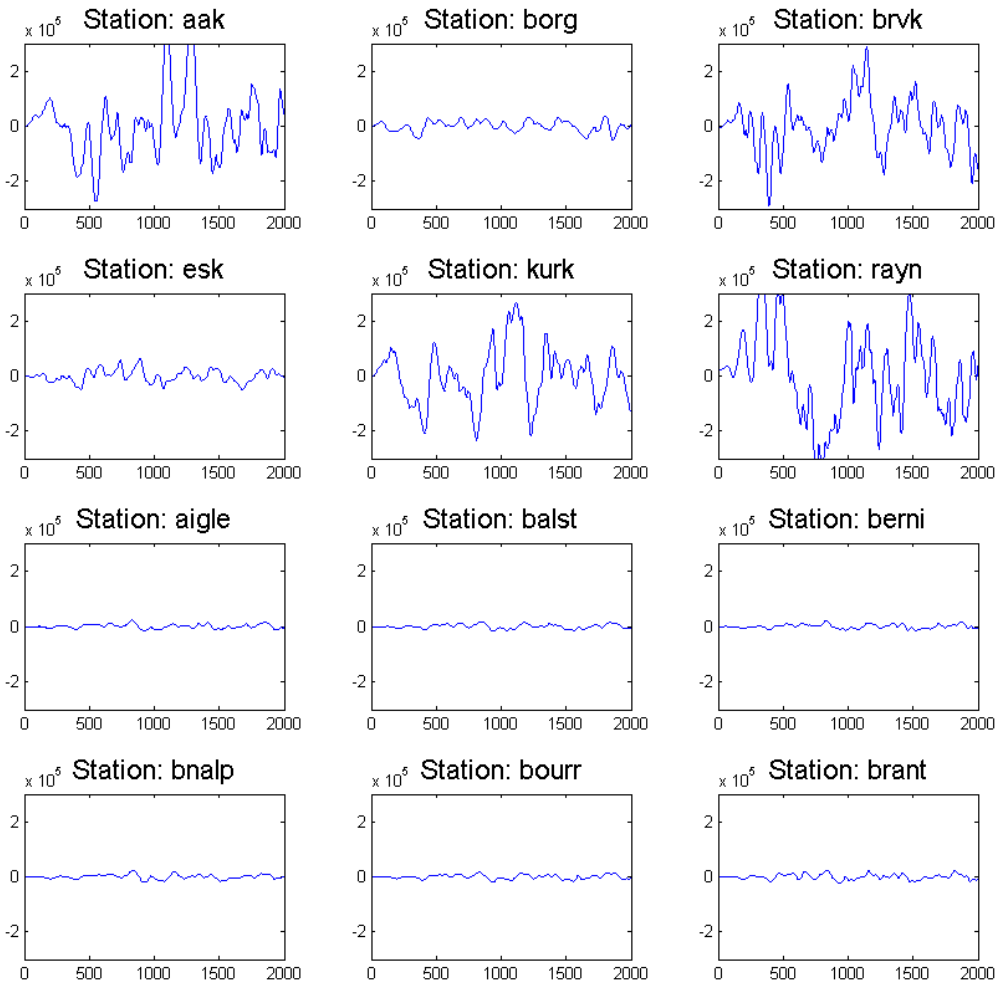

Figure A12: Seismograms from the event at Pakistan from cluster 2. Numbered 37-48 from left to right and top to bottom respectively.