An Infrastructure to Store and Analyse Seismic Data as Suffix Trees

MSc Data Analytics - Project Proposal

Tom Taylor

April 2016

This report is substantially the result of my own work except where explicitly indicated in the text. I give my permission for it to be submitted to the JISC Plagiarism Detection Service. I have read and understood the sections on plagiarism in the programme Handbook and the College website.

Contents

1 Abstract

The purpose of this project is to develop an infrastructure and tool set for converting raw seismic time series data into a searchable string using SAX (**S**ymbolic **A**ggregate appro**X**imation) and then to store this data as a suffix tree for fast searching and analysis. An interface will then be developed to enable the searching of these suffix trees and provide visualisation of the data. This enable the searching for similar patterns over time or between stations after an event.

Potential applications of the software could be:

- Automatically finding the arrival times of events
- Correlating events across multiple stations
- Finding similar events at a given location at other times

The code should ideally be re-usable where possible in a project to receive live streaming data from many stations.

2 Background

2.1 Seismic Waves

Seismic waves take on two main forms, body waves and surface waves. Body waves are those that travel through the interior of the earth and are the fastest travelling. The body waves are comprised of **P** (primary) waves which are compressional waves, travel fastest and thus arrive first. **S** (secondary) waves are shear waves and travel more slowly, thus arrive later. The separation between the phases is related to the distance of the earthquake and the local velocity structure. The surface waves travel only along the earths crust and, as they are confined to shallow depths where seismic velocities are slow, will normally arrive much later than the body waves.

A seismic station records movement over three axis: vertical (**z**) alongside horizontal in terms of north-south (**n**) and east-west (**e**). Due to seismic velocities generally increasing with depth, P waves arrive at a seismic station close to the vertical axis. As a result, P waves can be measured as a simple metric of displacement along the Z axis. S waves follow similar ray paths, but have their particle motion perpendicular to the direction of propagation. As a result, they manifest on a seismogram as movement on both the n and e axis. The geometry of the fault tends to have a bearing on the orientation of the displacement so the two axis of movement cannot be easily combined in to a single metric for time series analysis.

2.2 Symbolic Aggregate Approximation (SAX)

SAX (**S**ymbolic **A**ggregate appro**X**imation) (Lin et al., 2003) is a technique where by a single dimension of a time series is reduced to a string of symbols for pattern matching. The technique involves first transforming the normalised time-series in to a Piecewise Aggregate Approximation (**PAA**) which is then represented by a fixed number of symbols.

For the PAA, the data is first divided into equal sized time frames, then the mean deviation from zero of each frame is calculated. An appropriate number of breakpoints symmetrical along the x-axis are created so that they follow a Gaussian distribution and a symbol assigned to each range between the breakpoints. Then for each frame, a symbol is assigned based on which range the mean falls in to. The symbols assigned to each frame are then concatenated in to a string and it is this string that gives the SAX representation of that data. The width of the time frame and the number of discrete regions would be two parameters passed to this process alongside the data.

`[ Diagram showing PAA -> SAX on a time-series ]`

2.3 Suffix Trees and the related infrastructure at Birkbeck

Suffix trees (Weiner, 1973) are a representation of a string designed for performing search related algorithms. They are based on a suffix trie of a given string **T**. The trie is constructed using all of the suffixes of **T** with a terminator character not used in the alphabet appended to the end of **T**. The terminator is deemed to be lexicographically of lower priority than all of the other characters to ensure that a prefix of a string would appear before that string (such as in *as* before *ash*). In the following example, the suffixes of *T: abaaba* would produce the following suffixes with **$** applied as a terminator.

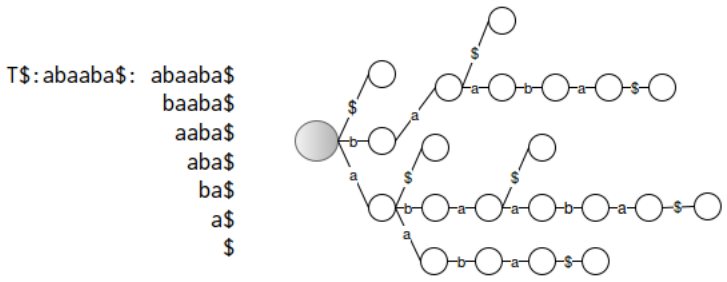

Figure 1: Example Suffix Trie

This allows for efficient searching and counting substrings of **T** by following the

trie from the root node. Each substring **S** of **T** is a prefix of a path starting at the root. It also allows for counting occurrences of the substring by finding the number of leaf nodes that can be reached from the end of the substring. This format is not particularly space efficient as it has an upper bound storage in the order of $O(n^2)$.

A far more efficient way to store and query the trie is to store it as a Suffix Tree. In order to produce a suffix tree from a trie, the non-branching nodes are firstly reduced (or coalesced) in to a single edge. The original string **T** is then stored along side the tree and the labels further reduced to a starting position and offset within **T**. The leaf nodes become labels to the offset of the suffix in the string. This reduces the upper bound storage to $O(n)$ and results in a tree for our example **T** that looks as follows:

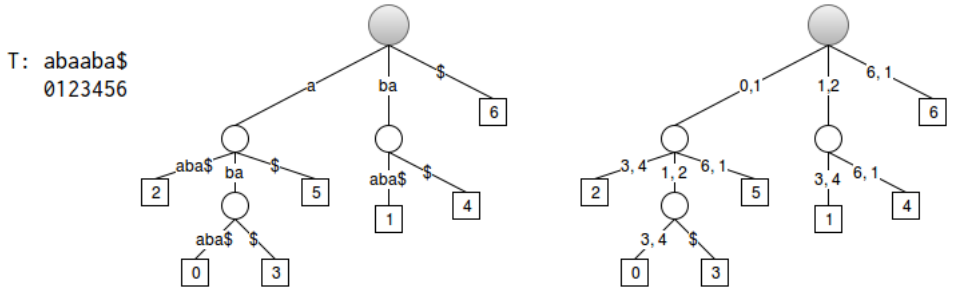

Figure 2: Example Suffix Tree (edges shown as text and as positions/offsets)

This technique of building the tree is considered naive though as it is again very inefficient (being computationally in the order of $O(n^3)$). A far more efficient way to build the tree is using the Ukkonens Algorithm (Ukkone, 1995). This technique allows for a time-linear *O(n)* construction of the tree. The algorithm is too long for inclusion but effectively starts with the implicit suffix tree of a string of length 1 and conditionally extends the tree on each iteration of adding a character.

At Birkbeck, an infrastructure has been developed to load and query Suffix Trees. There is current research in progress to utilise external memory (in this case SSDs) to back the loaded Suffix Trees. This would massively increase the maximum size of a tree to be queryable. The current libraries to utilise this are written in Python however there is currently a refactor happening to C due to Python not being particularly computationally efficient due to its interpreted nature.

3 Objectives

1. Interpretation of raw data from seismic stations (in either the SAC or miniseed format) and storage as time-series data in a time-series database.

2. Conversion of the data to SAX

3. Storage of the SAX representation as a loadable suffix tree.

4. An interface for searching and viewing the raw data.

4 Approach

As laid out in the objectives, the development will produce four separate components.

4.1 Interpretation and Storage as time-series data

The raw data received from the stations is accessed as files in the SAC (Seismic Analysis Code) or miniseed binary formats. These will be read using the ObsPy python library which can read both the headers and return the raw data as a Python object. Due to the complexities in analysing S waves (section 2.1), the scope of the project will be initially limited for searching for P waves on the Z axis only. However the interface will still have access to the raw data for all three axis and be able to show all three together for analysis by a seismologist.

This data will be batch processed and imported in to a time series database. Initially OpenTSDB is being considered for performance reasons but there may be scope for this to change depending on how difficult it is to implement this on the Universities hardware. Other options could be InfluxDB or Graphite which are considerably easier to implement but might suffer from scalability issues.

4.2 Conversion of data to SAX

Once the raw data has been loaded to the time series database, it will be re-read in batches and passed though an algorithm to produce a SAX string (as described in section 2.2). There are a few challenges in making this string usable. Firstly the seismic data will need to have noise removed and to be normalised across multiple stations. Secondly the parameters of alphabet length and the size of the time window most suitable will need to be empirically found.

4.3 Storage as a Suffix Tree

Once a suitable SAX string can be created across the dataset, it will be loaded in to a suffix tree using the toolset and infrastructure available at Birkbeck. The methodology

will ultimately be influenced by the exposed API once the aforementioned refactor is complete.

4.4 Interface to query the data

The interface to query and view the data will be web based as to ensure maximum compatibility with clients. Many open source graph renderers are in existence such as Grafana and Cubism.js and components of both are likely to be used for viewing the raw seismic data alongside the SAX data.

5 Plan

The various components (parsing, SAX, Suffix Trees, Interface, etc.) will all be written separately and act as micro services. They will each be callable via a REST HTTP API to reduce any compatibility issues between any technologies used and to increase portability for use in other projects.

Much of the coding is planned to be done in Python 3. This is because the working prototype for the Suffix Tree infrastructure at Birkbeck is written in Python. Also the Python library ObsPy is considered relatively mature sand has been shown to work with the raw sample data.

One potential limitation for using Python is the multi-threading components (for the cPython implementation at least) as it relies on the concept of a Global Interpreter Lock. The GIL effectively prevents multiple threads from running simultaneously to avoid potentially non-thread-safe operations from conflicting with each other. This will very likely have an impact on the speed of data analysis when working over large datasets so other languages may be considered. Also the maturity of various libraries will be considered when making language choices over individual components. Other languages that might come in to play to overcome this shortcoming are R and Scala.

Development will take the approach of Test Driven Development (TDD) where functional tests and class level unit tests will be written for each component and the test suite will be used within the build environment during development.

For training and initial validation of the usefulness of the software, some sample data based on research at the Nabro volcano in Eritrea will be provided by the Department of Earth and Planetary Sciences at Birkbeck. Key events have already been manually picked from this data. Further data will then be provided for test set validation.

Once a proof of concept is functioning, work will again be coordinated with the Department of Earth and Planetary Sciences to find suitable parameters for the normalisation of the seismic data and for the time frame and alphabet sizes for the SAX implementation.

References

Lin, J., E. Keogh, S. Lonardi, and B. Chiu
2003. A symbolic representation of time series, with implications for streaming algorithms. *Data Mining and Knowledge Discovery.*

Ukkone, E.
1995. On-line construction of suffix trees. *Algorithmica.*

Weiner, P.
1973. Linear pattern matching algorithms. *14th Annual IEEE Symposium on Switching and Automata Theory.*