A Comparison of Hardware and Software in Sequence Rule Evolution

Magnus Lie HETLAND[1] and Pål SÆTROM[2]
[1]*Norwegian University of Science and Technology, Dept. of Computer and Information Science, Sem Sælands vei 9, NO–7491 Trondheim, Norway*
magnus@hetland.org
[2]*Interagon AS, Medisinsk-teknisk senter, NO–7489 Trondheim, Norway*
paalsat@interagon.com

Abstract. Sequence rule mining is an important problem in the field of data mining. Many algorithms have been devised that are based on counting candidate rules and excluding those with low support. Recently, the techniques of heuristic search, and evolutionary algorithms in particular, have been applied to various data mining problems, including sequence mining. In our previous work we have used specialized hardware to make mining certain rule formats feasible. In this paper we compare the performance of this hardware with realistic software alternatives. We show that these software alternatives give acceptable, although significantly slower, running times for restricted rule formats. We also demonstrate that the increased expressiveness available with the hardware rule formats does not necessarily have a great impact on predictive power, and may be more useful as a way of tailoring rule formats to specific tasks.

1 Introduction

Sequence rule mining is an important problem in the field of data mining. The basic problem is that of discovering rules that describe the relationship between parts of the data and that have certain desirable qualities, such as confidence and support [1], or specialized measures of *interestingness* [2]. Many algorithms have been devised that are based on counting candidate rules and excluding those with low support. It is assumed that rules with low support (that is, those that occur infrequently) are less interesting.

Recently, the techniques of heuristic search, and evolutionary algorithms in particular, have been applied to various data mining problems [3], including sequence mining [4, 5]. These techniques make possible very flexible rule formats and quality measures, but they rely on efficient information retrieval techniques, because one or more queries on the full database must be performed for each evaluation of the heuristic objective function, and this function is typically calculated a very large number of times.

In our previous work we have shown that specialized pattern matching hardware can make it feasible to use evolutionary data mining to mine rules of a format that is a superset of those of most existing approaches, with many added features [4]. While this method has worked well, it may be of little use if one does not have access to the hardware. Therefore we wish to evaluate the feasibility of using a software solution instead.

In this paper we examine the role of the specialized hardware in our method, and compare its performance, both in terms of rule quality and execution speed, with realistic software alternatives. We focus on two software solutions: (1) similarity based time series retrieval with signature indexing [6] and (2) pattern based sequence retrieval with regular expressions [7,8].

2 Method

This section first describes the general principles of evolutionary sequence mining. Then follows a section about sequence retrieval, outlining the three main retrieval methods used in this paper.

2.1 Evolutionary Sequence Mining

There are many forms of heuristic search, including tabu search [9], simulated annealing [10], and simple hill-climbing. While we use evolutionary algorithms, other heuristic search methods would quite likely give similar results.

Evolutionary sequence mining is simply an evolutionary search for interesting patterns in sequence data. In practical terms:

– The individual solutions are patterns, or, in our case rules, and

– The fitness is related to the occurrences of the individual solutions in the data, found using some form of information retrieval.

To further elaborate, the patterns can be thought of as rules of the format: “If *antecedent* then *consequent* (within T time units). Here, the rule consequent is some given event that may occur at given times, possibly extrinsic to the series itself, while the antecedent (the condition) is a pattern in some pattern language. In the following experiments the consequent is part of the problem definition, while the antecedent is the solution. That is, we want to find rules for predicting the occurrence of certain events (more specifically, the upward movement of the time series) in a time series.

2.1.1 Evolution Setup

In our approach, the individuals are rule antecedents, expressed in a query language called IQL [11] or some subset. This query language is designed specifically to exploit the capabilities of the hardware we use, the PMC (see Section 2.2.3). The rules are represented as syntax trees, and are subject to the common crossover and mutation operations used in Genetic Programming (GP) [12].[1] The internal nodes in the trees represent the operators in the given language. The leaf nodes are symbols from the alphabet used for discretizing the data (see Section 3 for information about the discretization).

More specifically, in the regular expressions experiments the syntactical nodes used were *union*, *character sets*, and *concatenation*.

[1]The PCA rule format (see Section 2.2.1) is also amenable to simple fixed-length array genome representation and Genetic Algorithm operators [13].

$$
\begin{array}{ll}
(1) & R \rightarrow SS \\
(2) & S \rightarrow TT \\
(3) & T \rightarrow UU \\
(4) & U \rightarrow C\&C \\
(5) & U \rightarrow C \\
(6) & C \rightarrow \geq A \\
(7) & C \rightarrow \leq A \\
(8) & A \rightarrow a, \text{ for some } a \in \Sigma
\end{array}
$$

Figure 1: PCA rule grammar. The grammar is shown in Backus–Naur Form (BNF), with nonterminals represented by uppercase letters, terminals represented by the symbols & (Boolean conjunction), $\leq$ and $\geq$ (range constraints), and the lowercase a, which is any character from the data alphabet. Alternatives are represented as separate productions. Adjacent symbols are concatenated

Due to an interaction effect between union and closure when matching with the PMC, certain matches may be reported with a small offset. While this is not critical in most cases, it can be crucial in a predictive setting such as ours. It is possible to circumvent this problem, but for the sake of implementational convenience, we have opted to forego the use of closure. As this is done in both forms of pattern queries (the regular expressions and the larger IQL subset) this omission should not give an unfair advantage to either rule format. Also, preliminary experiments show no gain in predictive accuracy by adding closure to the available operators.

The PCA vector experiments (see Section 2.2.1) used *concatenation*, *conjunction*, and *ordinal comparisons* ($\leq$ and $\geq$). Also, in the PCA experiments, the syntactic structure was constrained as shown in Figure 1. The R, S and T nodes are used for generating a balanced tree of the height needed for matching a vector of (in our case) length 8. In principle, a grammar can be created for matching vectors of any given length. Also note that the grammar in Figure 1 corresponds to a subset of IQL, so the rules may be used with the PMC.

For all our experiments we used fifty generations. For the regular expression and full IQL experiments (see Section 3 for details) we used a population size of 100, while for the PCA experiments we used a population size of 1000.

2.1.2 Fitness

Several fitness measures are possible for rule mining. In [5], for example, we developed a fitness measure for unsupervised mining of interesting rules. In this paper we have chosen to use the correlation measure used in [4], because it makes objective quality comparisons based on predictive power easier.

The fitness measure is the *correlation* between the occurrences of a rule and the *desired* occurrences of the rule, as given by the event to be predicted. In our case, this event is the time series moving up in the next time step. Using the elements from the *confusion matrix* of a rule (true and false positives and negatives, with the obvious abbreviations) this correlation may be expressed as

$$\frac{\mathrm{TP}\cdot\mathrm{TN} - \mathrm{FP}\cdot\mathrm{FN}}{\sqrt{(\mathrm{TN}+\mathrm{FN})(\mathrm{TN}+\mathrm{FP})(\mathrm{TP}+\mathrm{FN})(\mathrm{TP}+\mathrm{FP})}},$$

where TP, FP, TN, and FN are the number of true and false positives and true and false negatives, respectively.

2.2 Sequence Retrieval and Rule Formats

The field of information retrieval is broad, and the number of methods for retrieving sequences is vast. In this paper we concentrate on two approaches: (1) similarity based retrieval, using sample sequences as queries, and (2) pattern based retrieval, where the query is a pattern in some query language, for example, a regular expression, a Boolean expression, or some combination.

We look at three specific retrieval methods: signature indexing, pattern matching with software (including pattern indexing), and pattern matching with hardware. These methods are described in the following three sections.

2.2.1 Signature Indexing

Signature indexing is a method for effectively performing similarity based sequence retrieval. The basic principles of the method are described in [6].

Simply put, similarity based retrieval is based around the notion of *distance functions* (or, more precisely, *dissimilarity* functions). A query takes the form of a sample object (or possibly a sample sub-object, such as a subsequence). Given such a query q and a distance function $d(\cdot,\cdot)$, the retrieval system must find all objects in the data base D that fall within a certain distance d_{max}, that is,

$$R = \{r \in D \mid d(q,r) \leq d_{max}\}, \tag{1}$$

where R is the set of returned objects.

It is typically difficult to create index structures that fit this form of retrieval directly for a given object type, such as a time series. However, methods exist for indexing simple objects such as real vectors of fixed dimensionality (so-called spatial access methods).

To use such indexing methods, we must define a map s, which creates a *signature* vector for an object, and a corresponding distance measure d_s, such that

$$\forall x,y \quad d_s(\tilde{x},\tilde{y}) \leq d(x,y), \tag{2}$$

where $\tilde{x} = s(x)$ and $\tilde{y} = s(y)$. Then, intuitively, if we use $s(q)$ as our query and d_s as our distance function, things seem closer than they really are (unless $s(x) = x$ and $d_s = d$). This requirement guarantees that using d_s will not result in any false dismissals. (It will, however, most likely result in several incorrectly returned objects that may be filtered out at a later stage.) For it to be possible for an index to work with d_s, it must usually obey certain other restrictions. We will not go into that here.

This form of signature indexing for similarity retrieval has received much attention in recent years (see [6] for a survey). This makes this form of sequence database a natural candidate for data mining. In the following, we describe one specific form of signature, and how it may be used in evolutionary data mining. Similar techniques will be possible with other signature forms.

Figure 2 demonstrates a signature form introduced independently by Yi et al. [14] and Keogh et al. [15], the Piecewise Constant Approximation, or PCA (also called the Piecewise Aggregate Approximation, PAA). The signature is calculated from a time series by dividing the sequence into k equal-sized non-overlapping segments, and calculating the average value in each segment. The resulting k dimensional vector is the signature. This signature type can

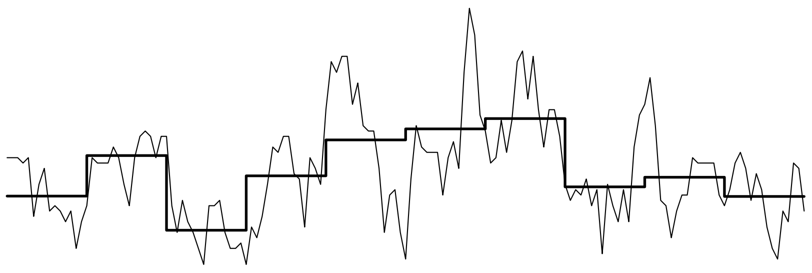

Figure 2: A time series and its PCA signature

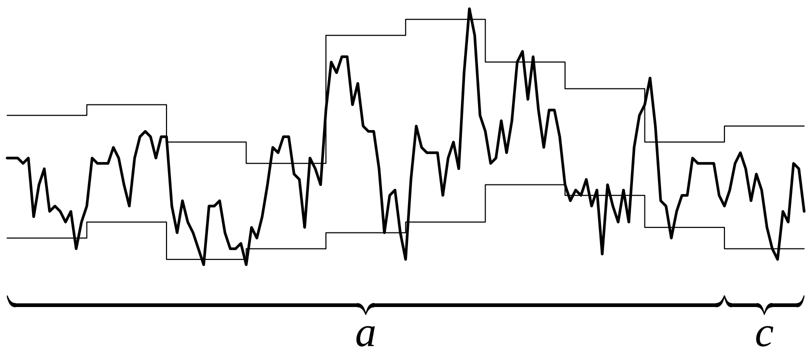

Figure 3: Intuitive interpretation of a bounding hyperrectangle as a rule with antecedent *a* and (optional) consequent *c*

be used with any L_p norm, and Keogh has shown that it can be used with more complex distance functions, such as Dynamic Time Warping [16]. It is robust, and simple to implement. Assume that an index with PCA vectors is available, that the original time series was of length n and that the signatures have been extracted from all (overlapping) windows of length m. Also assume that we have associated a boolean value p with each signature, which indicates whether the given window is immediately followed by an upward movement in the time series. Given this database, our challenge is finding a rule format that may be used in forming queries. We could use a single signature (as shown in Figure 2), a distance measure such as $d = L_2$ (Euclidean distance), and a radius d_{max}, which would be equivalent to a hypersphere in k dimensional space. However, an even simpler alternative is possible, which is also easier to visualize: an axis parallel hyperrectangle.

An axis parallel hyperrectangle simply gives an upper and lower limit to the permitted values in each dimension. All common spatial access methods are able to retrieve objects found within such hyperrectangles. (This is commonly known as a multidimensional range search.) As Figure 3 shows, axis parallel hyperrectangles are (unlike hyperspheres) easy to visualize, which is important when the results are presented to a human expert. The figure also shows how such rules may be used in unsupervised rule mining (as in [5]), by letting the first $k-1$ segments function as antecedent (marked a), and the last segment as consequent (marked c). We will not be using the rules in this manner here.

In our experiments we use the Hybrid Tree [17] to index the PCA signatures.

2.2.2 Pattern Matching and Indexing

Pattern matching is a well-known problem in the field of information retrieval [18]. A *pattern* is some form of specification, often in the form of a grammar or a predicate, which characterizes the requested objects. Well-known examples are *regular expressions* (the kind of grammars that characterize regular languages) and *Boolean* or *set expressions*, using set operations such as union, intersection, and negation. Such set operations assume that some other form of query has been executed to generate result sets that may be combined. These subqueries may be plain substring searches (for example, for words in text) or other pattern queries, such as regular expressions.

For simple query types such as Boolean combinations of single word occurrences, effective and simple index forms exist (such as, for example, inverted files [18]). There are index methods for more complex patterns such as regular expressions, but much less research has been done in this area.

One theoretically attractive indexing method is that described in [19]. The basic idea is to use the finite automaton derived from a regular expression to traverse a suffix tree over the data. For a certain class of regular expressions, this gives a logarithmic running time, and for regular expressions in general, it gives a sublinear running time. Whereas [19] theoretically analyzes the properties of the method in great detail, the method has not (to our knowledge) yet been implemented.

Another indexing scheme is that of [20], which relies on *k*-gram indexing. The software system described in [20] is not, however, publicly available.

In this paper we use two well-known tools for efficiently matching regular expressions:

– The index structure Glimpse [7], which uses agrep for its regular expression matching. Agrep has long been regarded as the state of the art for regular expression searching, both exact and approximate.

– A more recent tool, NR-grep [8], which in many cases is known to be faster than agrep.

Note that, unlike the suffix tree method in [19], neither of these actually index the regular expressions (the index structure of Glimpse is designed for roughly locating words in large amounts of data consisting of several files).

2.2.3 Pattern Matching Hardware

In this paper, as well as in our previous work, we use the Interagon Pattern Matching Chip (PMC) [21] to localize rule occurrences in the data. This gives us access to the full Interagon Query Language (IQL) [11] as rule format.

IQL is a rich query language supporting functionality such as regular expressions, positional offsets, and generalized Boolean expressions. The PMC is able to efficiently parallelize IQL queries, and can process data at about 100 MB/s.

The PMC consists of three functional units, as illustrated in Figure 4: A data distribution tree (top), a set of processing elements, or PEs (middle), and a result gathering tree (bottom). The PEs monitor the data flowing through the chip. They receive the data from the data distribution tree, which can be configured so that single PEs and groups of PEs receive data either sequentially or in parallel. Each PE is configured with one byte (character) and a comparison operator, which it uses to look for bytes in the data stream. Matches are reported to the result

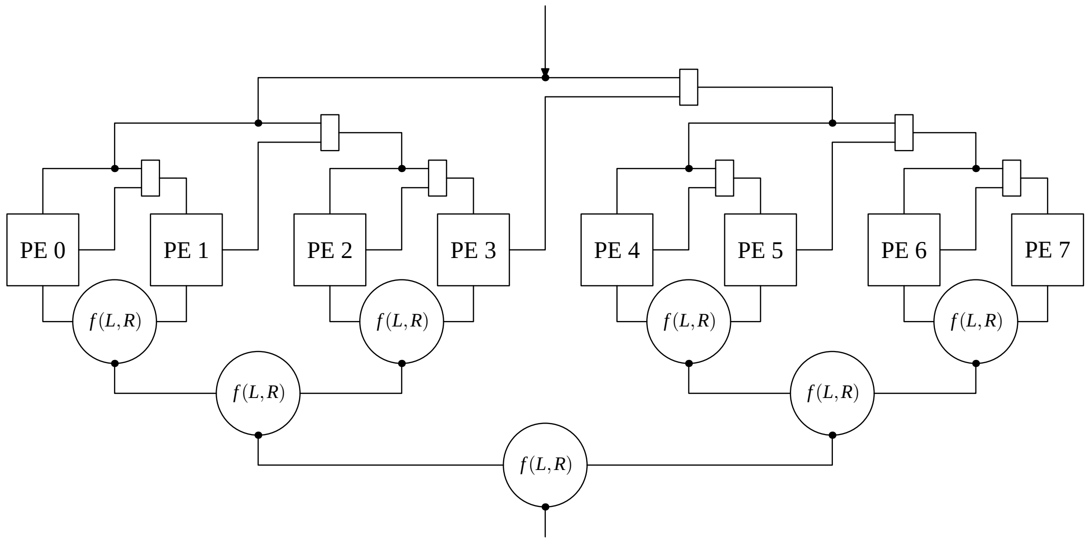

Figure 4: A data distribution tree with eight leaf nodes (PEs), and the corresponding result gathering tree with $f(L,R)$ calculated from the above left (L) and right (R) results.

gathering tree, which combines them and produces the final result, a Boolean value representing a hit or miss for the entire pattern. If the result is a hit, the hit location is reported back to the host computer.

3 Experiments

In this paper we have restricted ourselves to using a single data set, the ECG data used in [22] (taken from the UCR time series data mining archive [23]). This data set is highly regular and periodic, and should yield good results for any viable data mining method. In addition, it is one of the largest data sets in the URC archive.

For the pattern rules (regular expressions and IQL), we use the discretization process described in [22], using the best parameters found there (alphabet size 2 and window size 2). For the PCA experiments, the data were handled differently for the spatial index and the PMC. As a preprocessing step to the use of Glimpse and NR-grep, the discretized data were broken into records of fixed length, using a sliding window. Thus, the data size for Glimpse and NR-grep was increased by a factor of 64 (the size of the sliding window) compared to that used with the PMC. This was necessary to get exact positions from the software, which is inherently record oriented. For a regular expression to match, it would have to match the end of the record. Each record thus represented one position in the sequence.

For the spatial index, a sliding window was used to generate a set of vectors of fixed dimensionality (eight dimensions were used, for ease of representation in the PMC; see Section 2.2). Each vector then had its contents shifted and scaled (by subtracting the minimum value and dividing by the value span) to have its values fall in the range $[0,200]$. (The number 200 was chosen to parallel that used in the PMC experiments; see below.)

For the PMC, the values were transformed in a similar manner: Each number was transformed by subtracting the minimum of the last $k = 8$ numbers (the ones leading up to, and including, the number in question), and then dividing by the value span of these k numbers. Finally, these numbers were multiplied by 200 and rounded to get integers in the range $[0,200]$. This was done to accommodate the need for byte-oriented data in the PMC.

Table 1: Speed comparison

	PCA	Regex	IQL
Hybrid tree	27.16 s	–	–
Glimpse	–	509.65 s	–
NR-grep	–	542.44 s	–
PMC	0.96 s	0.28 s	2.38 s

The rule formats used in our experiments are described in Section 2.2. We have compared the three for predictive accuracy. In addition, for the PCA rules and the simple pattern rules, we have compared the running time when using an index structure to that of using the PMC.

All the experimental results (times and accuracies) are the average over ten separate runs (and, for the accuracies, using separate test sets).

3.1 Mining Speed

In the following, the time used by the GP system is not considered, as it is not dependent on the retrieval mechanism used in the fitness computation. It may be interesting, though, to know that the time used by the GP system ranged from about 22 s to about 243 s.

The software retrieval experiments (Glimpse, NR-grep and the Hybrid Tree) were performed on a 1 GHz Pentium III with 256 Mb of memory, running Gentoo Linux.

As can be seen in Table 1, there are great differences in running times. For regular expression rules, the time taken by the PMC is less than 0.05% of that taken by the software (Glimpse and NR-grep). For the PCA data, the differences are less pronounced but still quite clear, with the PMC taking only 3.5% of the time used by the Hybrid Tree.

Please note that these figures should not be taken to indicate that Glimpse is in general faster than NR-grep, or indeed as a thorough empirical test of the speed of the given software solutions. We have modified the software to run multiple queries to avoid the overhead in starting the programs, but may still not have used the software in an optimal manner. Also, since our main goal was to demonstrate the feasibility of the software solution, rather than to do a comprehensive benchmark, we have not tested the behavior across many different kinds of data and queries. The specific data set was chosen because of its size and its clearly discernible features, as mentioned in Section 3.

3.2 Rule Quality

For comparing rule quality, we use predictive accuracy, that is, the (maximum likelyhood-estimate of the) probability that a rule will give a correct prediction at a random position in the test data. The results are shown in Table 2.

Again, using other data sets would most likely produce different results (see [22] for the application of IQL to other data sets), but the given data still demonstrate that the different rule forms are comparable in predictive power. Even though the differences are statistically significant ($p \ll 0.01$ with Fisher's exact test), they may well be problem dependent, and may not by themselves be large enough to justify using one method over the others. Other factors,

Table 2: Quality comparison

Rule form	Accuracy
PCA	69.0%
Regex	71.5%
IQL	72.2%

such as rule comprehensibility or ease of implementation, may also influence the choice of rule format.

4 Discussion

In this paper we have compared the use of specialized hardware and existing solutions for information retrieval in evolutionary sequence mining. In addition to rule formats used in our previous work, we have also introduced the PCA rule format, which is suited for spatial indexing. We have shown that the software solutions are substantially slower than the PMC, but the running times are still acceptable, indicating that evolutionary sequence mining may well be performed with existing software. By using distributed evolution (that is, parallelization) the running times could be reduced even further. Even so, the superior running time of the PMC indicates that it could be used to mine huge data sets directly, whereas the software solutions most likely would have to be used on random data samples.

The other main advantage of the PMC is the expressiveness of its rule language. Even though this is clearly useful in that it allows the user to specify a domain-specific language, depending on the type of rules desired, it may not be crucial for predictive power, as indicated by the results in this paper. It should be noted, though, that the data chosen for these experiments is not particularly complex, and that for more complex data sets, the expressiveness of full IQL might impact the results.

Acknowledgements

We would like to thank Prof. Ricardo Baeza-Yates for his kind assistance. Also, we would like to thank Kaushik Chakrabarti, Gonzalo Navarro, and the Arizona Board of Regents for making the source code available for the Hybrid Tree, NR-grep, and Glimpse, respectively.

References

[1] David Hand, Heikki Mannila, and Padhraic Smyth. *Principles of Data mining.* The MIT Press, 2001.

[2] R. J. Hilderman and H. J. Hamilton. Knowledge discovery and interestingness measures: A survey. Technical Report CS 99–04, Department of Computer Science, University of Regina, Saskatchewan, Canada, October 1999.

[3] Alex A. Freitas. *Data Mining and Knowledge Discovery with Evolutionary Algorithms.* Springer-Verlag, 2002.

[4] Magnus Lie Hetland and Pål Sætrom. Temporal rule discovery using genetic programming and specialized hardware. In *Proc. of the 4th Int. Conf. on Recent Advances in Soft Computing (RASC),* 2002.

[5] Pål Sætrom and Magnus Lie Hetland. Unsupervised temporal rule mining with genetic programming and specialized hardware. In *Proc. 2003 Int. Conf. on Machine Learning and Applications, ICMLA*, 2003.

[6] Magnus Lie Hetland. A survey of recent methods for efficient retrieval of similar time sequences. In *Data Mining in Time Series Databases*. World Scientific, 2003.

[7] U. Manber and S. Wu. GLIMPSE: A tool to search through entire file systems. In *Proceedings of the USENIX Winter 1994 Technical Conference*, pages 23–32, San Fransisco, CA, USA, 17–21 1994.

[8] Gonzalo Navarro. NR-grep: a fast and flexible pattern-matching tool. *Software Practice and Experience*, 31(13):1265–1312, 2001.

[9] Fred W. Glover and Manuel Laguna. *Tabu Search*. Kluwer Academic Publishers, 1998.

[10] P. J. M. Van Laarhoven and E. H. L. Aarts. *Simulated Annealing: Theory and Applications*. D Reidel Publishing Company, 1987.

[11] Interagon AS. The Interagon query language : a reference guide. `http://www.interagon.com/pub/whitepapers/IQL.reference-latest.pdf`, sep 2002.

[12] J. R. Koza. *Genetic Programming: On the programming of Computers by Means of Natural Selection*. The MIT Press, Cambridge, MA, 1992.

[13] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison–Wesley, 1989.

[14] B. Yi and C. Faloutsos. Fast time sequence indexing for arbitrary L_p norms. *The VLDB Journal*, pages 385–594, 2000.

[15] E. J. Keogh, K. Chakrabarti, M. J. Pazzani, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *Journal of Knowledge and Information Systems*, 3(3):263–286, 2001.

[16] E. J. Keogh. Exact indexing of dynamic time warping. In *Proc. 28th Int. Conf. on Very Large Data Bases, VLDB*, pages 406–417, 2002.

[17] Kaushik Chakrabarti and Sharad Mehrotra. The hybrid tree: An index structure for high dimensional feature spaces. In *Proc. Int. Conf. on Data Engineering, ICDE*, pages 440–447, 1999.

[18] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press/Addison–Wesley Longman Limited, 1999.

[19] Richardo A. Baeza-Yates and Gaston H. Gonnet. Fast text searching for regular expressions or automaton searching on tries. *Journal of the ACM*, 43(6):915–936, 1996.

[20] Junghoo Cho and Sridhar Rajagopalan. A fast regular expression indexing engine. In *Proc. 18th International Conference on Data Engineering, ICDE*, 2002.

[21] Interagon AS. Digital processing device. PCT/NO99/00308, Apr 2000.

[22] Magnus Lie Hetland and Pål Sætrom. The role of discretization parameters in sequence rule evolution. In *Proc. 7th Int. Conf. on Knowledge-Based Intelligent Information & Engineering Systems, KES*, 2003.

[23] E. Keogh and T. Folias. The UCR time series data mining archive. `http://www.cs.ucr.edu/~eamonn/TSDMA`, sep 2002.