Linear Predictive Coding and Cepstrum coefficients for mining time variant information from software repositories

Giuliano Antoniol
RCOST- University Of Sannio
Via Traiano 1
82100, Benevento (BN), ITALY
+390824305526
antoniol@ieee.org

Vincenzo Fabio Rollo
RCOST- University Of Sannio
Via Traiano 1
82100, Benevento (BN), ITALY
+390824305526
f.rollo@unisannio.it

Gabriele Venturi
RCOST- University Of Sannio
Via Traiano 1
82100, Benevento (BN), ITALY
+390824305526
venturi@unisannio.it

ABSTRACT

This paper presents an approach to recover time variant information from software repositories. It is widely accepted that software evolves due to factors such as defect removal, market opportunity or adding new features. Software evolution details are stored in software repositories which often contain the changes history. On the other hand there is a lack of approaches, technologies and methods to efficiently extract and represent time dependent information. Disciplines such as signal and image processing or speech recognition adopt frequency domain representations to mitigate differences of signals evolving in time. Inspired by time-frequency duality, this paper proposes the use of Linear Predictive Coding (LPC) and Cepstrum coefficients to model time varying software artifact histories. LPC or Cepstrum allow obtaining very compact representations with linear complexity. These representations can be used to highlight components and artifacts evolved in the same way or with very similar evolution patterns. To assess the proposed approach we applied LPC and Cepstral analysis to 211 Linux kernel releases (i.e., from 1.0 to 1.3.100), to identify files with very similar size histories. The approach, the preliminary results and the lesson learned are presented in this paper.

Keywords

Software evolution, data mining.

1. INTRODUCTION

An intrinsic property of software is malleability: Software systems change and evolve at each and every level of abstraction and implementation during their entire life span from inception to phase out. This fact, calls for approaches, methods, and technologies to study evolution of software characteristics during the system life.

The evolution of a software system is observable as changes in structural information (e.g. modular decomposition and relation between modules), behavioral information (e.g. functionalities, or bugs), and project information (e.g., maintenance effort). As these changes happen in time, software evolution can be modelled and studied as time series. A time series is a collection of measures recorded over time. Time series and time series based approaches haves been successfully applied to many disciplines such as speech processing, computer vision, or stock market forecasting. Common to these disciplines is the need to detect the occurrence of similar phenomena evolutions over time. Therefore models and technologies developed to study time series and time dependant phenomena or signals can be applied to software engineering.

In applying time dependant models to software artifacts evolution our goal is the definition of a criterion to establish similarity or dissimilarity of artifact histories. Indeed, similarity is quite a crucial issue: there are several software engineering areas such as software evolution and maintenance, software analysis, software testing, or automatic Web Services composition where the ability to effectively compute a similarity between artifact histories can greatly help researchers and practitioners.

On the other hand, similarity computation is a difficult problem. Often, similarity discovering is hampered by the presence of some distortion in one dimension of data (e.g., time). This distortion can cause dissimilar instances seem similar and the opposite as well.

As an example, effort prediction in software development or maintenance requires both effort prediction and effort distribution forecasting (i.e, schedule) [9]. Traditional approaches focus on effort prediction assuming a relation, often linear [14], between metrics related to complexity and/or size and the effort [1] [2] [3] [10]. Often a simple figure quantifying the effort doesn’t suffice. Effort distribution over time is a key issue for project planning and staffing, therefore is an important cost driver and a cause of organizational disruption.

Unfortunately, effort distribution forecasting is more difficult than effort prediction because discovering similarities between past projects effort distributions is hampered by several factors causing ‘distortion’ in the data if represented as evolving in a linear time.

While the overall effort in past maintenance projects is mainly related to high level software metrics [6][14], the effort distribution is determined by internal system dependencies and organizational issues. Internal system dependencies can easily induce ripple effects imposing constraints between activities, a component must be changed *after* some other has undergone maintenance. Organizational issues like holydays, staffing decisions, reorganizations, and so on, can cause postponing of activities and impact on the effort distribution in an unpredictable way. Therefore, analysing past effort distribution to determine similarities among time histories can be a difficult task, since similarities among activities are hidden because of these factors, while spurious similarities can emerge for the same reason. In other words, automating similarity computation between artifact histories is a challenging and difficult task. Similar difficulties are present in other software engineering activities such as log file or user behaviour analysis.

The above example outlines the usefulness of robust similarity detection approaches, robust when the original data are distorted in time.

We present an approach to detect similarities between artifacts histories. In particular we aim at devising an approach to detect similarities in evolutions starting from past maintenance and activities effects, notwithstanding their temporal distortions. Theories and technologies to detect similarities in phenomena evolving in time, in a manner that the time rate can change among instances and also during a single instance are present in literature. In this work we applied one of these, namely LPC/Cepstrum, to mine from a repository of Linux kernel modules, files evolved in the same or very similar ways.

The remainder of this paper is organized as follow: first we present the background of the used approach, **Case study and results** section illustrate the application of the approach to the Linux kernel evolution data, and in **Discussion and future works** we debate about our results and indicate our future work guidelines

.

2. TACKLING TIME RATE CHANGES

Automatic speech recognition and speech synthesis researchers have a long history of wrestling with time distortion. Human beings change the rate of speech when talking (prosody), but humans recognize words also in presence of dramatic changes in pronunciation speed or accent during locution. When machines come into play, it is quite obvious expecting from them at least a similar ability in comprehension. Therefore, a speech recognition system must be robust with respect to time distortion as well as to disturbance (noise).

Among the speech recognition approaches the family based on Linear Predictive Coefficient and Cepstrum (LPC/Cepstrum) is prominent for its performances and its relative simplicity. LPC/Cepstrum, first proposed in [7] and subsequently in [12] and [13], models a time evolving signal as an ordered set of coefficients representing the signal spectral envelope. That is a curve passing close to the peaks of the original signal spectrum. To obtain the LPC/Cepstrum representation the first step is to compute Linear Predictive Coding (LPC) coefficients. These are the coefficients of an auto-regressive model minimizing the difference between linear predictions and actual values in the given time window.

The LPC analysis uses the autocorrelation method of order p.

In matrix form, we have

$$Ra = r$$

where

$$r = [r(1)r(2)..r(p)]^T$$

is the autocorrelation vector,

$$a = [a_1 a_2 \ldots a_p]^T$$

is the filter coefficients vector and R is the p*p Toeplitz autocorrelation matrix, which is nonsingular and gives the solution

$$a = R^{-1} r.$$

Once LPC have been obtained it is possible to compute cepstra from them. Cepstra are the coefficients of the inverse Fourier transform representation of the *log* magnitude of the spectrum. The cepstra series represents a progressive approximation of the 'envelope' of the signal: as for LPC, the more are the cepstra considered the more the envelope adheres to the original spectrum.

Starting from ***a*** and ***r***, we have as $\boldsymbol{c_m}$ coefficients (for order p):

$$c_0 = r(0),$$

$$c_m = a_m + \sum_{k=1}^{m-1} \frac{k}{m} c_k a_{m-k},$$

for 1 < m < p, and

$$c_m = \sum_{k=m-p}^{m-1} \frac{k}{m} c_k a_{m-k},$$

where m > p.

In speech recognition LPC/Cepstrum has been proven capturing most of the relevant information contained in the original series. For a sequence of 30-300 points a number of 8-30 coefficients suffice for most application. Therefore, LPC/Cepstrum allows to obtain a very synthetic representation of a time evolving phenomenon. This compact representations can be used to efficiently compare signals, once a suitable distance measure has been defined between LPC or Cepstrum coefficients. Most approaches aiming to assess similarity between time series use the Euclidean distance among the LPC/Cepstrum representations as an indirect similarity measure. Although distance and similarity are different concepts, cepstral distance can be used to assess series similarity: If two cepstra series are "close", the original signals have a similar evolution in time. As an alternative to Cepstrum and Euclidean distance, it is possible to use the Itakura distance (a.k.a. Log Likelihood Ratio LLR) [4] that can be computed directly from LPC.

LPC/Cepstrum has been used also in computer vision and in other research fields [11]. For examples in [15] LPC/Cepstrum is applied to online signatures verification and Euclidean distance between LPC/Cepstrum has been used as dissimilarity measure to cluster ARIMA series modeling electrocardiogram signals [5].

3. CASE STUDY AND RESULTS

We tested the application of LPC/Cepstrum to the evolution of a real world software system: the Linux kernel. Our goal was to verify if LPC/Cepstrum can be a starting point to produce compact representations of software modules evolution while preserving essential characteristics of the phenomena under study. In other words, if the spectral based representations could be applied to identify artifacts having very similar maintenance evolution histories. Being interested in mining the effect of maintenance on artefact but also in effort we selected a metric that is quite commonly recognized as strongly related to maintenance effort: size measure in LOC. Therefore our initial dataset was composed by the LOC histories, 211 releases, of 1788 files composing the Linux kernel from version 1.1.0 to 1.3.100 for 211 releases.

Over this dataset we performed LPC/Cepstrum analysis where the modules evolution in size was thought of as signals evolving in time. Once obtained LPC/cepstum coefficients we computed the distance between each pair of module (that is about one million of module pairs). A method to be effective must efficiently produce results, our approach for the 1788 histories requires less than 5 seconds on a Pentium 4 machine at 1.6 GHz. The tools used in each phase are summarized in Table 1. These are all open source software integrated together allowing an almost fully automated analysis.

Table 1: Test case technologies and instruments

Phase	Instruments
Extraction of size modules evolution from CVS repository	Perl scripts
LPC computation	C program
Cepstra computation	C program
Euclidean distance computation	C program
Results classification and graph plotting	Perl script and GNUPlot

To produce useful results, a similarity assessment based on an abstraction and on a distance measure must respond to three minimal requirements:

a) It has to discriminate among similar histories, allowing to identify some as similar and some as dissimilar by applying a threshold (such as the more restrictive the threshold the less the pairs deemed similar). The possibility to vary the threshold is important because similarity research often starts with a blur similarity definition gaining sharpness in late phases. Therefore it must be possible to customize similarity detection on the fly.

b) It has to be sensible to the relative richness of the information supplied. With less information most items seem similar, increasing the information used we expect fine grain dissimilarities to emerge. This is important because allows researchers to decide the best abstraction level for the case at hand.

c) It has to respond to some intuitive and meaningful notion of similarity. Because similarity is not a value in themselves: similarities discovery is ancillary to other purposes for which a clear understanding of a similarity judgment is fundamental.

To address items a) and b) we calculate the sets of files with indistinguishable time series applying three distance thresholds (Euclidean distances less that $1*10^{-3}$, $1*10^{-4}$, and $1*10^{-5}$) and four cepstra series lengths (12, 16, 20, 32). Since the more cepstra are used the more the envelope representation adhere to the original data, with less cepstra we expect to find more similar pairs and the opposite as well. The size of 8, 12, and 16, for both the LPC coefficients and the subsequent cepstra series, is a rule of thumb in speech coding. However, as this is the first application to software engineering of LPC/Cepstrum spectral representation, we decided to try the sizes from 12 to 32 to allow a richer signal representation. It should be noted that our thresholds are quite thight because the computed distances among software modules were far smaller than the ones among words in speech recognition.

By applying the above defined parameters we obtained Table 2, in which the number of files pairs deemed indistinguishable over a given threshold is shown for each combination of threshold value and cepstra series length.

Table 2. Number of pairs beating the thresholds for cepstra cardinality.

Threshold	Cepstra series cardinality			
	12	16	20	32
1*10-3	6045	4049	2897	1605
1*10-4	858	607	440	312
1*10-5	194	163	144	129

Table 2 responds to a) and b): the cardinality of the pairs considered undistinguishable is sensible to both threshold value and cepstra series length. These effects can be better appreciated in **Figure 1** and 2 reporting the impacts of thresholds and cepstra series length, respectively. Notice that both tables have a logarithmic Y axis thus quite different results are obtained with different configurations.

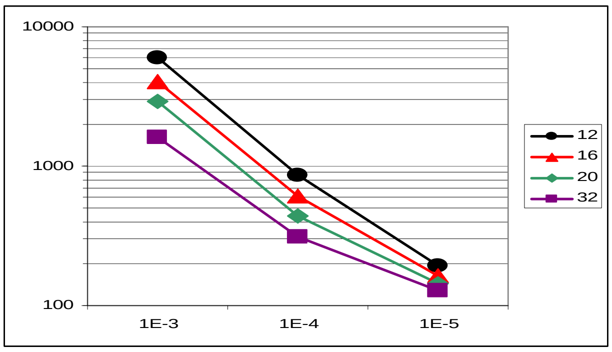

Figure 1. Impact of the threshold over the number of pairs deemed similar (logaritmic).

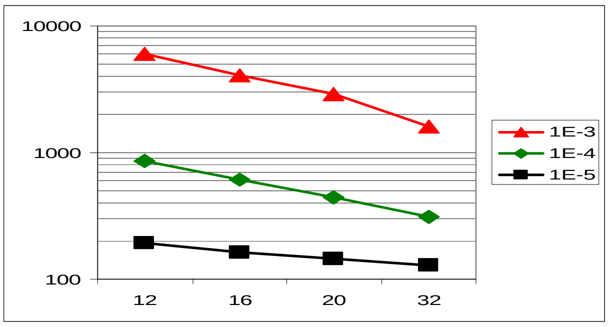

Figure 2. Impact of the cepstra series cardinality over the number of pairs demmed similar (logaritmic).

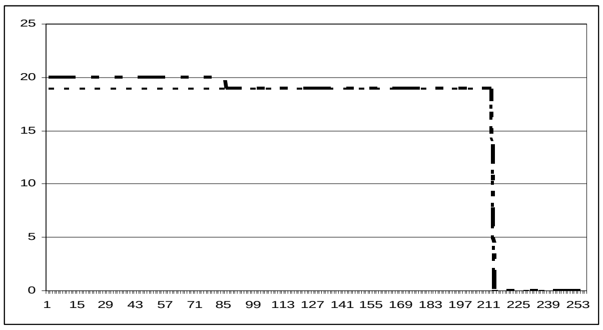

Figure 3. Less similar pair selected with 32 cepstra and a threshold of 10^{-5}.

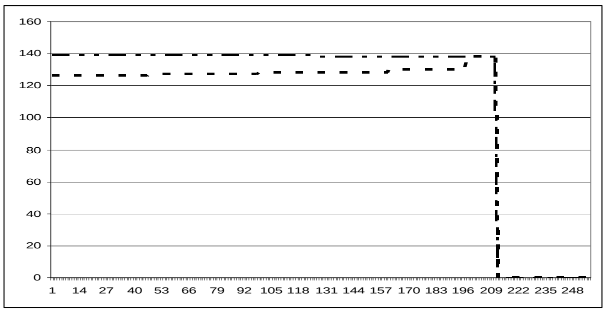

Figure 4. Less similar pair selected with 16 cepstra and a threshold of 10^{-5}.

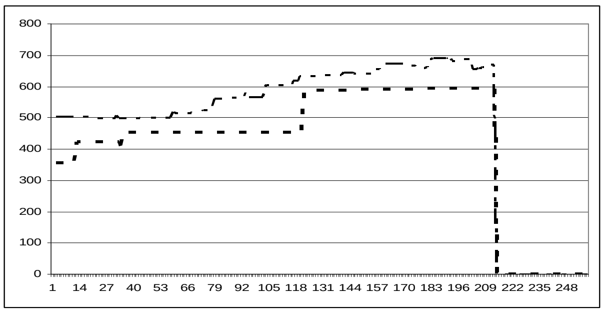

Figure 5. Most similar pair selected with 12 cepstra and a threshold of 10^{-2} that is discarded by more restrictive criteria.

To qualitatively assess whether the results of the automated analysis responds to some intuitive notion of distance and similarity (item c) we plotted the graphs of pairs classified as indistinguishable. Here we report three examples chosen to give an insight of how different configurations impact on distance and similarity. Figure 3, 4, and 5 report plots of the less similar pair selected with 32 cepstra and a threshold of 10^{-5}; the less similar pair selected with 16 cepstra and a threshold of 10^{-5}; and the most similar pair selected with 12 cepstra and a threshold of 10^{-3} not included in the sets selected by the other criteria. Notice that the Y axis aren't of the same scale.

The graphs show an appreciable progressive relaxation of the similarity as far as the cepstra series size is reduced and a less stringent threshold is applied.

4. DISCUSSION AND FUTURE WORKS

This work presents a case study assessing the suitability of LPC/Cepstrum to compare software artifacts evolutions. LPC/Cepstrum allows to obtain a compact representation of signals with linear complexity and to perform a robust comparison with respect to signal distortion. Computational efficiency, output compactness, and robustness are appealing characteristics for tools supporting software engineering activities. However, since the approach stems from a different research field, there is the need to assess its suitability. We conducted a first case study comparing evolution histories of 1788 Linux files at LOC level. We also defined three success criteria for the case study. To be deemed interesting for further explorations, the approach must: allow defining similarity thresholds, be sensible to the quantity of information used, and produce results responding to an intuitively understandable notion of similarity.

Indeed, we found that Euclidean distances computed among LPC/Cepstrum representations can be used to assess similarity in a way that is sensible to the richness of the representation and allows to define effective similarity thresholds. By inspecting histories we also verified that the sets of similar pairs selected with our approach respond to an intuitive notion of similar evolution in size. Therefore, the case study results show that LPC/Cepstrum is worth of further exploration by software engineering researchers.

An important theoretical issue is left aside from this case study. Distance measures are often seen and used as indirect similarity measures under the assumption that closeness between items is related to their similarity. This is a keystone of spectral representations use in speech recognition. In this case study we followed this approach as well. Nevertheless is must be pointed out that similarity and closeness remain two different concepts. From a theoretical perspective we believe that a better clarification of similarity between software artifact histories can be of great help in software evolution research.

A first further research step will be to apply the same framework to other software systems. In doing so we aim to gain knowledge about what are the cepstra containing the most relevant information, what are the threshold values most suitable for the various tasks and how the approach performs when metrics other than size are used. This should allow a broader understand of LPC/Cepstrum characteristics when applied in software engineering.

Spectral based representations support also comparisons with metrics other than Eucliedean distance (e.g. the Itakura distance), ad allow for further abstracting from data distortion in time (e.g. by means of time warping [8]). Exploring these alternatives it is possible to increase the robustness of the approach with respect to distortion and its flexibility with respect to the distance used.

Finally it is remarkable that LPC/Cepstrum has been successfully used also in situations in which data are distorted in dimensions other than time. This suggests the application to software engineering situation in which data are distorted in other dimensions as well (e.g. size or effort).

5. REFERENCES

[1] Boehm, B.W. *Software Engineering Echonomics*. Prentice-Hall Inc., Englewood Cliffs, N.J., 1981.

[2] Boehm, B., Clark, B., Horowitz, E., Westland, C., Madachy, R., and Selby, R. Cost Models for Future Software Life Cycle Processes: COCOMO 2.0.*Annals of Software Engineering*.vol. 1, 1987, 57-94.

[3] Hastings, T.E., and Sajeev, A.S.M. A Vector-Based Approach to Software Size Measurement and Effort Estimation. *IEEE Transactions on Software Enginnering* , vol. 27, no. 4, 2001, 337-350.

[4] Itakura F.,Minimum prediction residual principle applied to speech recognition, *IEEE Trans. Acoustics, Speech, and Signal Processing* . vol.23,pp.67- 72,Feb. 1975

[5] Kalpakis K., Gada D., and Puttagunta V., "Distance Measures for Effective Clustering of ARIMA Time-Series". In *Proc. of the 2001 IEEE International Conference on Data Mining (ICDM'01)*, San Jose, CA, November 29-December 2, 2001, pp. 273-280.

[6] Lindvall, M. Monitoring and Measuring the Change-Prediction Process at Different Granularity Levels: An Empirical Study. *Software Process Improvement and Practice*, no. 4, 1998, 3-10.

[7] Markel, J.D. and Gray Jr, A.H. *Linear Prediction of Speech*. Springer-Verlag, New York, 1976.

[8] Myers C.S. and Rabiner L.R. A comparative study of several dynamic time-warping algorithms for connected word recognition. *The Bell System Technical Journal*, 60(7):1389-1409, September 1981

[9] Mockus A., Weiss D.M., Zhang P. Understanding and Predicting effort In Software Projects. *Proc. of the 25th International Conference On Software Engineering*, 2003, 274 - 284

[10] Nesi, P. Managing Object Oriented Projects Better, *IEEE Software*, vol. 15, no.4. 1998, 50-60.

[11] Oppenheim A.V and Schafer R.W, "From Frequency to Quefrency: A History of the Cepstrum", *IEEE Signal Processing Magazine*, September 2004.

[12] Papamichalis, P.E. *Practical Approaches to Speech Coding*. Prentice Hall, Englewood Cliffs, NJ, 1987

[13] Rabiner, L.R. and Juang B.H. *Fundamentals of Speech Recognition*. Prentice Hall, Englewood Cliffs, NJ, 1993

[14] Ramil, J.F. Algorithmic Cost Estimation Software Evolution. *Proceding of Int.Conference on Software Engineeringr*, Limerick, Ireland, IEEE CS Press, 2000, 701-703.

[15] Wu, Q.Z., Jou, I.C., Lee, S.Y., Online Signature Verification Using LPC Cepstrum and Neural Networks, *IEEE Transactions on Systems, Man, and Cybernetics (27)*, No. 1, February 1997, pp. 148-153.