Resource Adaptive Periodicity Estimation of Streaming Data

Michail Vlachos, Deepak S. Turaga, and Philip S. Yu

IBM T.J. Watson Research Center
19 Skyline Dr, Hawthorne, NY

Abstract. Streaming environments typically dictate incomplete or approximate algorithm execution, in order to cope with sudden surges in the data rate. Such limitations are even more accentuated in mobile environments (such as sensor networks) where computational and memory resources are typically limited. This paper introduces the first "resource adaptive" algorithm for periodicity estimation on a continuous stream of data. Our formulation is based on the derivation of a closed-form incremental computation of the spectrum, augmented by an intelligent load-shedding scheme that can adapt to available CPU resources. Our experiments indicate that the proposed technique can be a viable and resource efficient solution for real-time spectrum estimation.

1 Introduction

Spectrum estimation, that is, analysis of the frequency content of a signal, is a core operation in numerous applications, such as data compression, medical data analysis (ECG data) [2], pitch detection of musical content [4], etc. Widely used estimators of the frequency content are the periodogram and the autocorrelation [5] of a sequence. For statically stored sequences, both methods have an $O(nlogn)$ complexity using the Fast Fourier Transform (FFT). For dynamically updated sequences (streaming case), the same estimators can be computed incrementally, by continuous update of the summation in the FFT computation, through the use of Momentary Fourier Transform [12, 9, 15].

However, in a high-rate, data streaming environment with multiple processes 'competing' over computational resources, there is no guarantee that each running process will be allotted sufficient processing time to fully complete its operation. Instead of blocking or abandoning the execution of processing threads that cannot fully complete, a desirable compromise would be for the system to make provisions for *adaptive* process computation. Under this processing model every analytic unit (e.g., in this case the 'periodogram estimation unit') can provide partial ('coarser') results under tight processing constraints.

Under the aforementioned processing model and given limited processing time, we are not seeking for results that are accurate or perfect, but only 'good-enough'. Since a typical streaming application will require fast, 'on-the-fly' decisions, we present an intelligent sampling procedure that can decide whether to

retain or discard an examined sample. Our technique is based on a lightweight linear predictor, which records a sample only if its value cannot be predicted by previously seen sequence values.

Due to the sampling process, the retained data samples (a subset of the examined data window) are not guaranteed to be equi-spaced. Hence, we also elaborate on a *closed-form* periodogram estimation given *unevenly spaced* samples. We should note that the proposed method for periodogram reconstruction based on irregularly spaced samples is significantly more lightweight than the widely used Lomb periodogram [13] (which incurs a very high computational burden).

Other recent work on periodicity estimation on data streams has appeared in [6], where the authors study sampling techniques for period estimation using sublinear space. [8] proposes sampling methods for retaining (with a given approximation error) the most significant Fourier coefficients. In [11] Papadimitriou, et al., adapt the use of wavelet coefficients for modeling a data stream, providing also a periodicity estimator using logarithmic space complexity. However, none of the above approaches address the issue of resource adaptation which is one of the main contributions of our work.

In the sections that follow we will illustrate the main concepts behind the adaptive computation of the spectrum. In section 3 we describe our intelligent 'on-the-fly' sampling, and in section 4 we elaborate on the closed-form incremental computation of the periodogram from unevenly spaced data samples. Finally, section 5 provides extensive experiments that depict the accuracy and effectiveness of the proposed scheme, under given complexity constraints.

2 Overview of our approach

Considering a data streaming scenario, our goal is to provide efficient mechanisms for estimating and updating the spectrum[1] within the current data window. We use the periodogram as an estimate of the spectrum. A schematic of our resource-adaptive methodology is provided in Fig. 1.

At any given time, there might not be enough processing capacity to provide a periodogram update using all the samples within the data window. The first step toward tackling this problem is the reduction of points using an 'on-the-fly' load-shedding scheme. Sub-sampling can lead to data aliasing and deteriorate the quality of the estimated periodogram. Therefore our sampling should not only be fast but also intelligent, mitigating the impact of the sub-sampling on the squared error of the estimated periodogram. Sampling is based on a linear predictor, which retains a sample only if its value cannot be predicted by its neighbors. An estimator unit is also employed, which changes over time the 'elasticity' of the linear predictor, for proper adaptation to the current CPU load.

[1] Note that during the course of the paper, we may use the terms periodicity estimation, spectrum estimation and periodogram estimation interchangeably

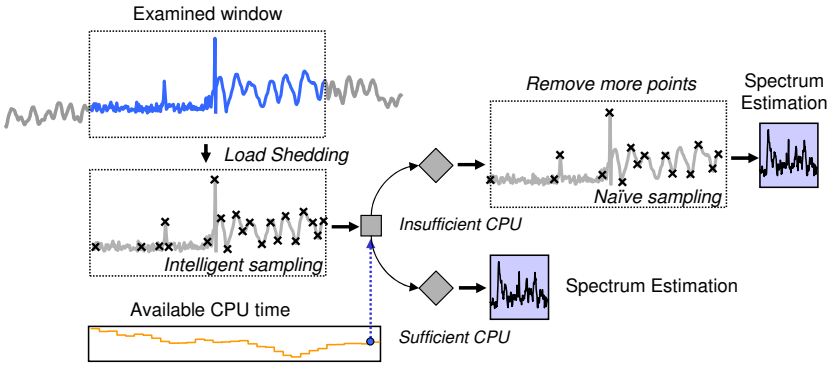

Fig. 1. Visual depiction of our methodology

If there is enough CPU time to process the final number of retained samples, the spectrum is computed. Otherwise, more samples are dropped randomly and the new estimate is computed on the remaining samples.

The computation of the approximate periodogram is based on a formulation of the DFT and the periodogram using unevenly spaced samples, a necessary step due to the sampling process. Under a sliding window model, some of the previously used samples are discarded, while new samples are added in the window. The proposed periodicity estimation algorithm possesses a very simple update structure, requiring only subtraction of contributions from discarded samples and addition of contributions due to the newly included samples.

The contributions of this paper are summarized below:

- We provide an abstraction of the resource adaptation problem for periodicity estimation.
- We propose an intelligent *load-shedding* scheme along with a *parameter estimator unit* that tunes the adaptation to the current CPU load.
- We present a *closed-form Fourier approximation* using unevenly spaced samples and we show how to update it incrementally.

We analyze the performance of our proposed approach under CPU constraints, and we measure the complexity abstractly, in terms of the number of multiplications, additions and divisions involved (making the analysis independent of the underlying processor architecture). Even though our model is very spartan in its memory utilization, we do not explicitly impose any memory constraints, since this work focuses primarily on CPU adaptation. However, inclusion of potential memory constraints is a straightforward addition to our model.

2.1 Notation

The Discrete Fourier Transform is used to analyse the frequency content in a discrete and evenly sampled signal. In particular for a discrete time signal $x[n]$

the DFT $X[m]$ is defined for all samples $0 \leq m, n \leq N-1$ as:

$$X[m] = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi n m}{N}} \tag{1}$$

The periodogram P of a signal corresponds to the energy of its DFT:

$$P[m] = ||X[m]||^2 \tag{2}$$

Consider now, a continuous signal $x(t)$ sampled unevenly at discrete time instants $\{t_0, t_1, \ldots, t_{N-1}\}$. We show an example of this in Fig. 2.

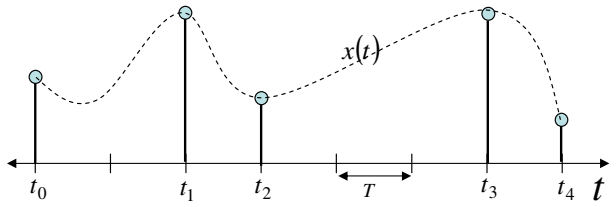

Fig. 2. Unevenly sampled signal.

We write this unevenly sampled signal using the discrete notation as $x[k_n]$ where $t_i = k_i T (k_i \epsilon \mathbb{Z}^+)$ and T corresponds to the sampling interval with all sampling instants as multiples. This is also shown in Fig. 2. In the remainder of this paper we will describe an adaptive load-shedding algorithm that retains unevenly spaced samples and we will also provide an incremental DFT estimation for such discrete signals.

We measure the complexity of all our algorithms in terms of the number of additions (subtractions), multiplications and divisions involved in the computations. Thus, we label the complexity of a single multiplication as ξ_{Mul}, of a division as ξ_{Div} and of a sum/subtraction as ξ_{Sub}.

3 Load-Shedding Scheme

We consider the typical problem of running spectral analysis where we slide a window across the temporal signal and incrementally update the signal's DFT (and the respective periodogram). We start with an evenly sampled signal, with sampling interval T. Consider that the window slides by a fixed amount $Width \times T$. As a result of this sliding we discard n_1 points from the beginning of the signal and add n_2 points to the end. However, if the available CPU cycles do not allow us to update the DFT using all the points, we can adaptively prune the set of added points using uneven sub-sampling to meet the CPU constraint while minimizing the impact on the accuracy of the updated DFT.

3.1 Intelligent sampling via a linear predictor

We now present an algorithm (with linear complexity) for the adaptive pruning of the newly added samples. In order to decide whether we can retain a particular sample, we determine whether it can be linearly[2] predicted from its neighbors. In particular, to make a decision for sample k_i we compare the interpolated value $x^{int}[k_i]$ with the actual value $x[k_i]$, where the interpolated value is computed as:

$$x^{int}[k_i] = \frac{x[k_{i-1}](k_{i+1} - k_i) + x[k_{i+1}](k_i - k_{i-1})}{k_{i+1} - k_{i-1}} \tag{3}$$

where sample k_{i-1} is the last retained sample before sample k_i and sample k_{i+1} is the immediately following sample. If $|x^{int}[k_i] - x[k_i]| \leq \frac{Thresh \times |x[k_i]|}{100}$ we can discard the sample k_i, otherwise we retain it. The parameter $Thresh$ is an adaptive threshold that determines the quality of the approximation. If the threshold is large, more samples are discarded, and similarly if the threshold is small fewer samples are discarded[3]. We show an example of this interpolation scheme in Fig. 3.

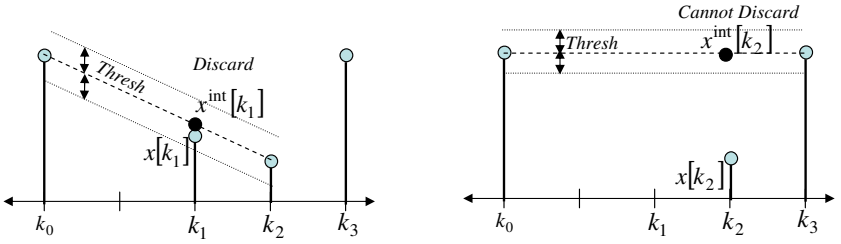

Fig. 3. Linear interpolation scheme for adaptive pruning of samples.

In Fig. 3, we show two steps of the algorithm. In the first step, we decide that we can discard sample k_1 as it can be interpolated by samples k_0 and k_2. In the next step, we decide that we cannot discard sample k_2, as it cannot be interpolated using samples k_0 and k_3, its neighbors. If we start out with n_2 samples that we need to prune, the complexity of this algorithm is:

$$\xi^{interp} = (2\xi_{Mul} + 4\xi_{Sub} + \xi_{Div})(n_2 - 2) \tag{4}$$

In Section 3.2 we discuss how to tune the threshold $Thresh$ in order to obtain the desired number of $\hat{n}_2$ samples, out of the n_2 samples added by the sliding window.

[2] Higher order predictors are also possible, but result in higher complexity

[3] Note that the squared approximation error due to this sub-sampling scheme cannot be bounded in general for all signals, however we select it for its computational simplicity. In particular, for the wide variety of signals we consider in our experiments, we do not observe squared error significantly larger than the absolute squared threshold value. Modification of this scheme to guarantee bounds on the approximation error is a direction for future research.

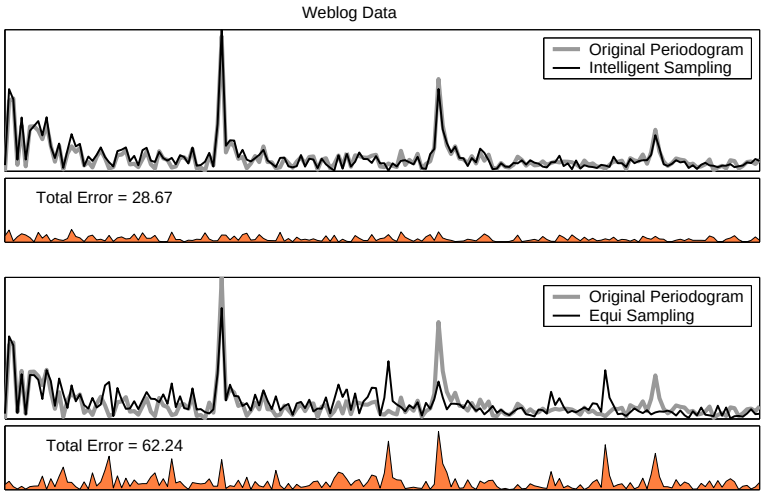

Fig. 4. Comparison of spectrum estimation errors for intelligent sampling and equi-sampling techniques.

In Fig. 4 we illustrate on a stream that measures web usage, a comparison of our intelligent sampling method against the equi-sampling technique, which samples data at a specified time interval. We execute our algorithm for a specific threshold and reduce the data points within a window from M down to N (unevenly spaced). We estimate the resulting periodogram (see section 4) as well the periodogram derived by equi-sampling every N/M points. It is apparent from the figure that intelligent sampling provides a much higher quality reconstruction of the periodogram, because it can retain important features of the data stream. Additional examples on more datasets are provided in Fig. 5.

3.2 Threshold Estimator

The load-shedding algorithm assumes the input of a threshold value, which directly affects the resulting number of retained points within the examined window. The desirable number of final points after the thresholding is dictated by the available CPU load. An optimal threshold value would lead to sampling exactly as many points as could be processed by the currently available CPU time. However, there is no way of predicting accurately the correct threshold without having seen the complete data, or without resorting to an expensive processing phase. In Figures 6 and 7 we provide various examples of the spectrum approximation for different parameters of the load-shedding threshold value.

We will provide a simple estimator of the threshold value with constant complexity, which is derived by training on previously seen portions of the data stream. The expectation is that the training will be performed on a data subset

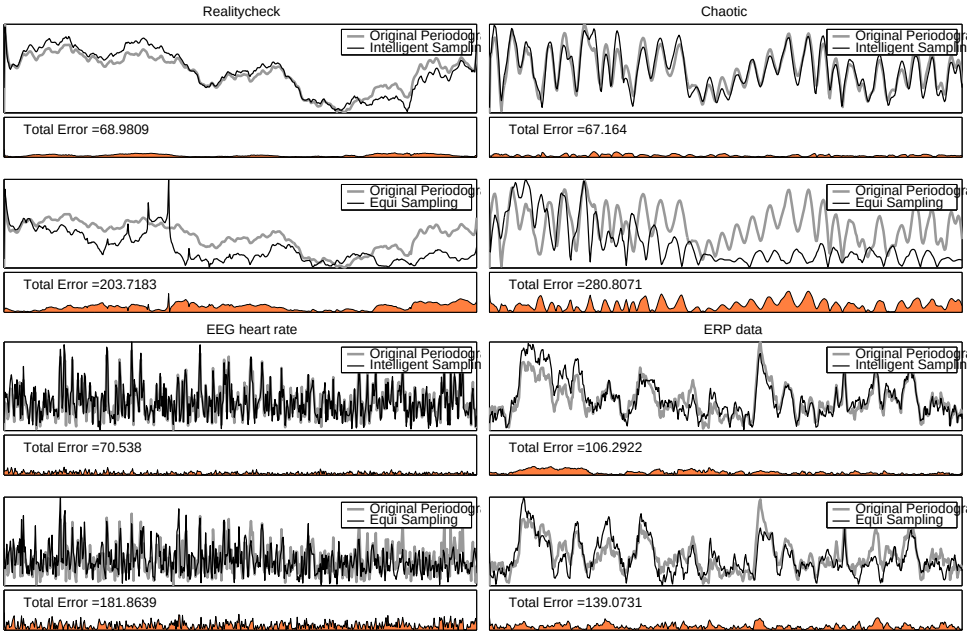

Fig. 5. Estimation comparisons for additional datasets

that captures a sufficient variation of the stream characteristics. The estimator will accept as input the desired number of final samples that should remain within the examined window, along with a small subset of the current data characteristics, which -in a way- describe its 'shape' or 'state' (e.g. a subset of the data moments, its fractal dimensionality, etc.). The output of the estimator is a threshold value that will lead (with high expectation) to the desirable number of window samples.

The estimator is not expected to have zero error, but it should lead *approximately* to the desired compression ratio. In the majority of cases the selected threshold will lead either to higher or lower compression ratio. Intuitively, higher compression (or *overestimated* threshold) is preferable. This is the case, because then one does not have to resort to the additional phase of dropping randomly some of the retained samples (a sampling that is 'blind' and might discard crucial points, such as important local minima or maxima). In the experiments, we empirically verify that this desirable feature is true for the threshold estimator that is presented in the following section.

3.3 Training phase

Assume that $\mathcal{F}$ is a set of features that capture certain desirable characteristics of the examined data window w, and $\mathcal{P} \in \{0, 1, \ldots, |w|\}$ describes how many

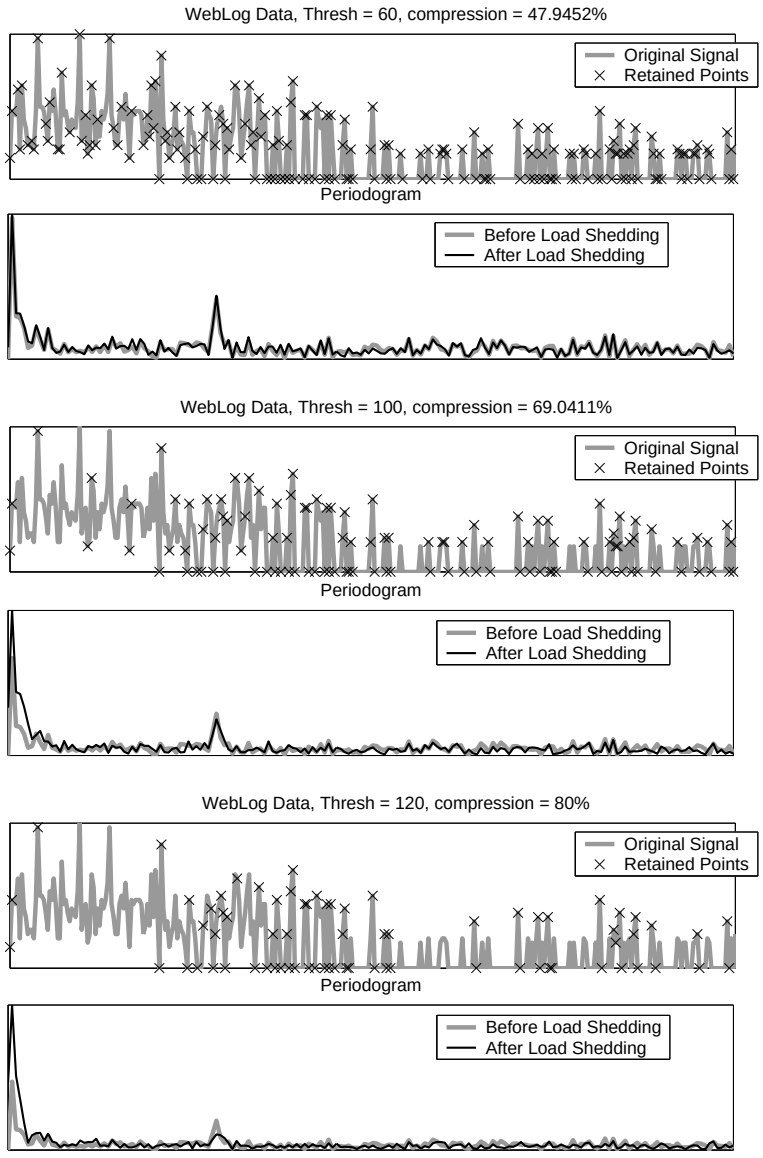

Fig. 6. [Weblog Data]: Spectrum approximation for different threshold values

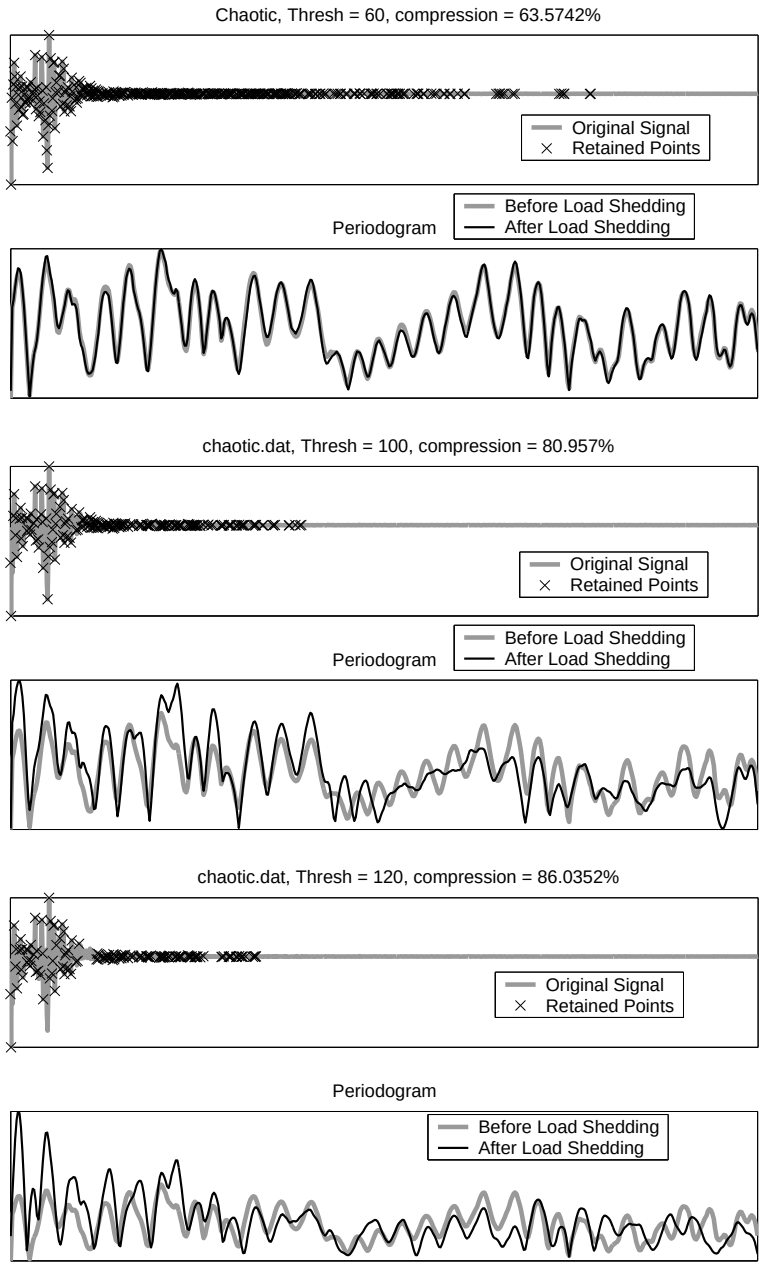

Fig. 7. [Chaotic Data]: Spectrum approximation for different threshold values

points can be processed at any given time. The threshold estimator will provide a mapping $\mathcal{F} \times \mathcal{P} \mapsto \mathcal{T}$, where $\mathcal{T}$ is a set of threshold values.

It is not difficult to imagine, that data whose values change only slightly (or depict small variance of values) do not require a large threshold value. The reverse situation exists for sequences that are 'busy', or exhibit large variance of values. With this observation in mind, we will use the *variance* within the examined window as a descriptor of the window state. Higher order moments of the data could also be used in conjunction with the variance for improving the accuracy of the predictor. However, for simplicity and for keeping the computational cost as low as possible, we select to use just the variance in our current prototype implementation.

The training phase proceeds as follows; given the training data we run a sliding window on them. For each data window we compute the variance and we execute the load-shedding algorithm for different threshold values (typically, $20, 40, \ldots, 100, 120$). After the algorithm execution the remaining number of data points is recorded. This process is repeated for all the extracted data windows. The result of this algorithm will be a set of triplets: `[threshold, variance, number of points]`. Given this, we can construct the estimator as a mapping $f(numPoints, variance) \mapsto Thresh$, where the actual estimator is essentially stored as a 2-dimensional array for constant retrieval time. An example of this mapping is shown in Fig. 8.

It is clear that the training phase is not performed in real-time. However it happens only once (or periodically) and it allows for a very fast prediction step.

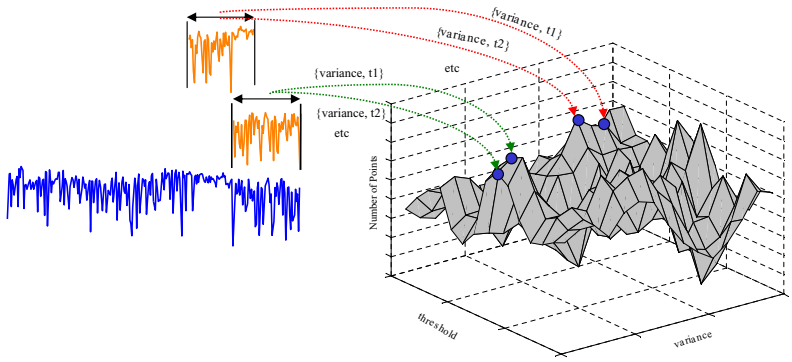

Fig. 8. Training phase for the threshold estimator

3.4 Additional notes

There are a couple of points that we would like to bring to the attention of the reader:

1. Even though we assume that the training data will provide 'sufficient' clues on the data stream characteristics, the estimator might come upon an input of `[variance, numPoints]` that has not encountered during the training

phase. In this case, we can simply provide the closest match, e.g. the entry that has the closest distance (in the Euclidean sense) to the given variance and number of points. Alternatively, we could provide an extrapolation of the values, in other words, explicitly learn the mapping function. This can be achieved by constructing an RBF network [1] based on the training triplets. Since this approach is significantly more expensive and could present overfitting problems, in our experiments we follow the former alternative.

2. Over the period of time, the stream characteristics may gradually change, and finally differ completely from the training data, hence leading to inconsistent predictions. We can compensate for this by 'readjusting' the predictor, by also recording the observed threshold error during the algorithm execution. This will result in a more extended maintenance phase of the estimator, but this cost is bound to pay off in the long run for datasets that exhibit frequent 'concept drifts' [10, 7]. We do not elaborate more on this extension, but we note it as potential addition for a more complex version of the threshold estimator.

4 Incremental Spectrum Estimation for Unevenly Sampled Signals

Consider a signal $x[k_i], 0 \leq i \leq N-1$, as shown in Fig. 2. Since the DFT is defined only for evenly sampled signals, we implicitly recreate an evenly sampled signal before computing the DFT. For this, we again use a linear interpolator (that matches our sub-sampling algorithm), thereby reconstructing a piece-wise linear evenly sampled signal. The DFT of this evenly sampled signal may be computed in terms of the sum of contributions of each of the individual line segments that constitute it. Due to the nature of the linear interpolator the contribution of each line segment to the DFT may be analytically derived in terms of only the endpoints of the segment (i.e. samples in the original unevenly sampled signal) and the distance between them. This means that we do not actually need to interpolate the unevenly sampled signal but can derive a closed form expression for the DFT under the assumption of a linear interpolation scheme. Similar approaches to ours have also been followed in [2]. Note that while the time domain signal consists of only N (uneven) samples, in order to compute the Discrete Fourier Transform (DFT) of this signal, we need to sample the DFT at least $M = k_{N-1} - k_0$ times to avoid time domain aliasing. If we denote by $X_n[m]$ the contributions to the Fourier Transform from each of the $N-1$ line segments that make up the implicitly recreated evenly sampled signal, then the DFT of the whole signal can be written as:

$$X[m] = \sum_{n=1}^{N-1} X_n[m] \tag{5}$$

where for $m = 1, \ldots, M-1$

$$X_n[m] = \frac{1}{(k_n - k_{n-1})(\frac{2\pi m}{M})^2}[(x[k_{n-1}] - x[k_n])(e^{-j\frac{2\pi m k_{n-1}}{M}} - e^{-j\frac{2\pi m k_n}{M}}) + j\frac{2\pi m}{M}(x[k_n]e^{-j\frac{2\pi m k_n}{M}} - x[k_{n-1}]e^{-j\frac{2\pi m k_{n-1}}{M}})] \tag{6}$$

and for $m = 0$

$$X_n[0] = \frac{1}{2}(x[k_{n-1}] + x[k_n])(k_n - k_{n-1}) \tag{7}$$

A significant benefit that equation (5) brings is that the DFT for such unevenly sampled signals can be evaluated incrementally. Hence, if we shift the window by a fixed width such that the first n_1 points are discarded, and n_2 points are added at the end, then the DFT of the signal may be updated as follows:

$$X^{new}[m] = X^{old}[m] - \sum_{n=1}^{n_1} X_n[m] + \sum_{n=N}^{N+n_2-1} X_n[m] \tag{8}$$

We now consider the complexity of computing this update. As with several papers that analyze the complexity of the FFT, we assume that the complex exponentials $e^{\frac{j2\pi m k_n}{M}}$ (and the intermediate value $\frac{2\pi m k_n}{M}$) are considered precomputed for all m and n. Using our labels for complexity as defined in the notation, the complexity of computing one single update coefficient $X_n[m]$ for $m = 1, \ldots, M-1$ may be represented as:

$$\hat{\xi} = 6\xi_{Mul} + 5\xi_{Sub} + \xi_{Div} \tag{9}$$

and for $m = 0$ as

$$\hat{\xi} = 2\xi_{Mul} + 2\xi_{Sub} \tag{10}$$

Finally, the complexity of updating all the M DFT coefficients in this scenario is:

$$\xi^{update}(M, n_1, n_2) = (n_1 + n_2)[(M-1)(6\xi_{Mul} + 5\xi_{Sub} + \xi_{Div}) + (2\xi_{Mul} + 2\xi_{Sub}) + M\xi_{Sub}] + 2M\xi_{Sub} \tag{11}$$

4.1 Benefit of Sub-sampling Algorithm

Using our sub-sampling algorithm we can reduce the number of samples that need to be used to update the DFT. Consider that as a result of the pruning, we can reduce n_2 samples into a set of $\hat{n}_2$ samples ($\hat{n}_2 \leq n_2$). While the reduction in the number of samples directly translates to a reduction in the complexity of the update, we also need to factor in the additional cost of the sub-sampling algorithm. Comparing equations (11) and (4) we realize that the overall complexity of the update (including the sub-sampling) is reduced when:

$$\xi^{update}(M, n_1, n_2) \geq \xi^{update}(M, n_1, \hat{n}_2) + \xi^{interp} \tag{12}$$

To determine when this happens, consider a simple case when $\hat{n}_2 = n_2 - 1$, i.e. the sub-sampling leads to a reduction of one sample. The increase in complexity for the sub-sampling is $(2\xi_{Mul} + 4\xi_{Sub} + \xi_{Div})(n_2 - 2)$ while the corresponding decrease in the update complexity is $(M-1)(6\xi_{Mul} + 5\xi_{Sub} + \xi_{Div}) + (2\xi_{Mul} + 2\xi_{Sub}) + M\xi_{Sub}$ (from equation (11)). Clearly, since $\hat{n}_2 < n_2 \leq M$, one can easily realize that the reduction in complexity far outweighs the increase due to the sub-sampling algorithm. In general, equation (12) is always true when the sub-sampling algorithm reduces the number of samples (i.e., when $\hat{n}_2 < n_2$).

If, at a certain time, the CPU is busy, thereby imposing a computation constraint of ξ^{limit}, we need to perform our DFT update within this constraint. If $\xi^{update}(M, n_1, n_2) > \xi^{limit}$ we cannot use all the samples n_2 for the update, and hence we need to determine the optimal number of samples to retain $\hat{n}_2$, such that $\xi^{update}(M, n_1, \hat{n}_2) + \xi^{interp} \leq \xi^{limit}$. Specifically, we may compute this as:

$$\hat{n}_2 \leq \frac{\xi^{limit} - \xi^{interp} - 2M\xi_{Sub}}{(M-1)(6\xi_{Mul} + 5\xi_{Sub} + \xi_{Div}) + (2\xi_{Mul} + 2\xi_{Sub}) + M\xi_{Sub}} - n_1 \quad (13)$$

Finally, we can achieve this by tuning the sub-sampling threshold $Thresh$ based on the algorithm described in Section 3.2.

5 Experiments

The usefulness of the proposed resource-adaptive periodicity estimation depends on two factors:

- The accuracy of the approach, which is indicated by the quality of the DFT approximation and its respective periodogram. If the periodogram after the load-shedding closely resembles the original one, then the provided estimate is meaningful.
- The adaptiveness of the proposed methodology, which is highly dependent on the quality of the threshold estimator. An accurate estimator will lead to sampling rates that closely adapt to the current CPU loads.

We examine separately those two factors in order to provide a more thorough and clear evaluation.

5.1 Quality of DFT estimation

The quality of the approximated Fourier coefficients is measured on a variety of periodic datasets obtained from the time-series archive at UC Riverside [14]. These datasets only have a length of 1024, therefore it is difficult to provide a meaningful evaluation on the streaming version of the algorithm. However, by providing the whole sequence as input to the periodicity estimation unit we can evaluate the effectiveness of the load-shedding scheme in conjunction with the closed-form DFT computation on the unevenly spaced samples. We compute the accuracy by comparing the estimated periodogram against the actual one

(had we not discarded any point from the examined data window). We run the above experiment on different threshold values $Thresh = 20 \ldots 120$. For example, a value of $Thresh = 20$ signifies that the predicted value (using the linear predictor) does not differ more than 20% from the actual sequence value.

Note that the original periodogram is evaluated on a window of M points ($M = 1024$), while the one based on uneven sampling uses only the N remaining samples ($N \leq M$). In order to provide a meaningful comparison between them we evaluate the latter periodogram on all $M/2$ frequencies -see equation 6-, even though this is not necessary on an actual deployment of the algorithm.

We compare the accuracy of our methodology against a naive approach that uses equi-sampling every N/M points (i.e., leading again to N remaining points within the examined window). This approach is bound to introduce aliasing and distort more the original periodogram, because (unlike the intelligent load-shedding) it does not adapt according to the signal characteristics.

Figures 9, 10 indicate the periodogram error introduced by the intelligent and the equi-sampling techniques. On top of each bar we also portray the compression achieved using the specific threshold $Thresh$, computed as $100 * (1 - N/1024)$.

The results suggest that the load-shedding scheme employed by our technique can lead to spectrum estimates of much higher quality than competing methods. In two cases (Fig. 9, Reality Check) the equi-sampling performs better than the linear interpolator, but this occurs only for minute compression ratios (i.e., when the threshold discards less than 10 samples per 1024 points). In general the observed reduction in the estimation error compared to equi-sampling, can range from 10% to more than 90% on the 14 datasets examined in this paper.

5.2 Threshold Estimator Accuracy

For testing the accuracy of the threshold estimator we need longer datasets, which could be used for simulating a sliding window model execution and additionally provide a training subset. We utilize real datasets provided by the automotive industry. These are diagnostic measurements that monitor the evolution of variables of interest during the operation of a vehicle. Examples of such measurements could be the engine pressure, the torque, vibration patterns, instantaneous fuel economy, engine load at current speed, etc.

Periodic analysis is an indispensable tool in automotive industry, because predictive maintenance can be possible by monitoring the changes in the spectrum of the various rotating parts. Therefore, a change in the periodic structure of the various engine measurements can be a good indicator of machine wear and/or of an incipient failure.

The measurements that we use have length of 50000 points and represent monitoring of a variable over an extended period of time[4]. On this data we use a sliding window of 1024 points. We generate a synthetic CPU load, which is provided as input to the periodicity estimation unit. Based on the synthetic CPU

[4] We have not provided the name of the specific engine measurement, because it is provided to us unlabeled by our automotive partner.

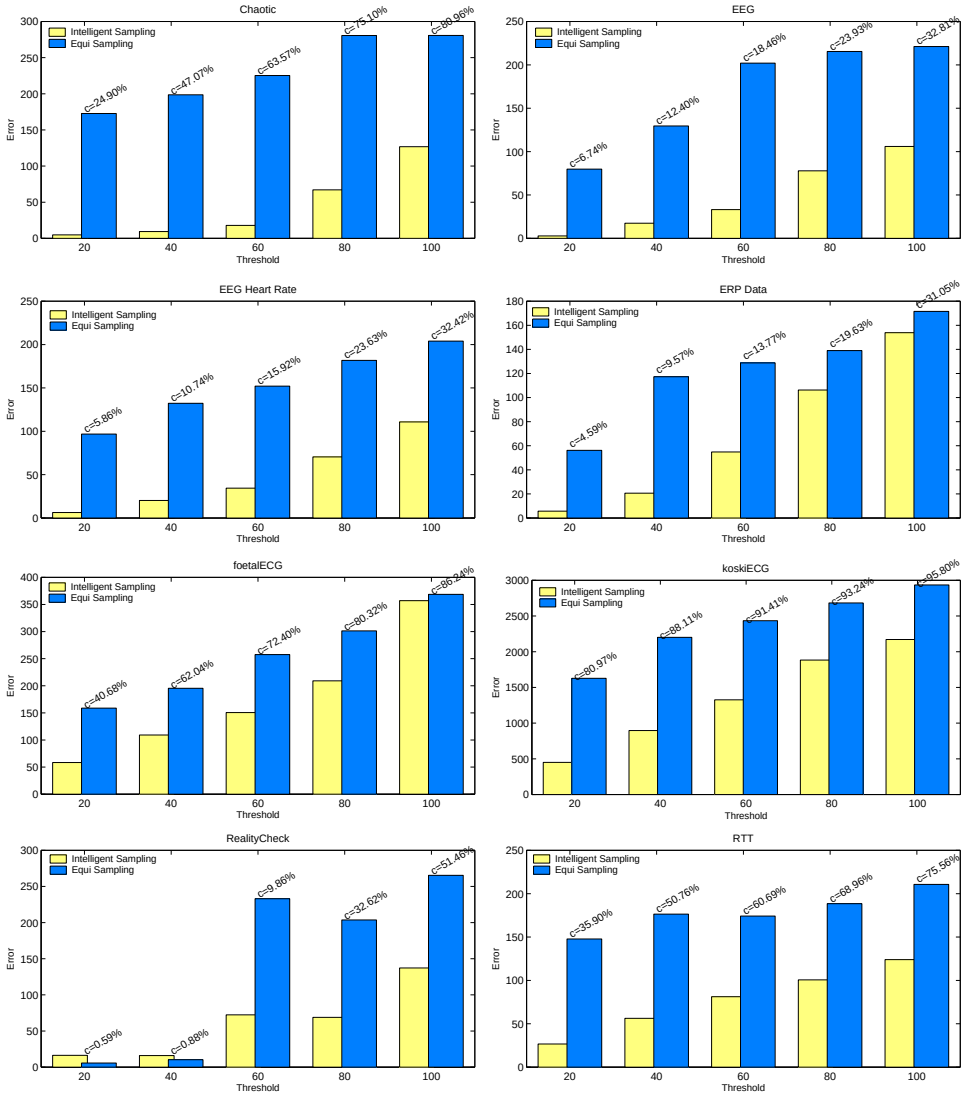

Fig. 9. Spectrum estimation comparison for various compression rates. The proposed intelligent sampling provides spectrum reconstruction of higher quality given the same number of samples.

trace, at any given point in time the periodicity unit is given adequate time for processing a set of points with cardinality within the range of 50 to 1024 (1024 being the length of the window). In Fig. 11 we show two instances of the approximated spectrum under limited CPU resources. On the first instance the indicated available CPU of 12.41% means that only 12.41% of the total window points should remain after the load-shedding, given the available processing time.

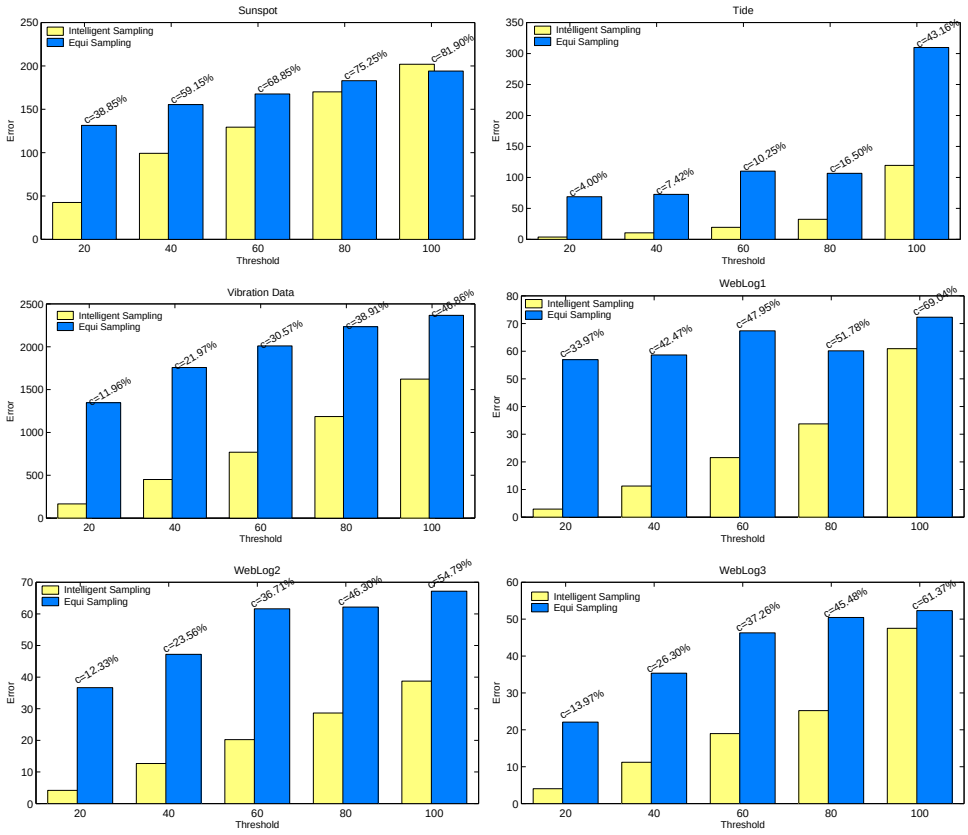

Fig. 10. Again the intelligent sampling outperforms equi-sampling techniques for the same compression rates.

Executing our algorithm on the complete data stream, we monitor the accuracy of the threshold estimator. The estimator is fed with the current CPU load and provides a threshold estimate $Thresh_{est}$ that will lead with high probability to $\hat{P}$ remaining points (so that they could be sufficiently processed given the available CPU load). Suppose that the actual remaining points after the application of the threshold $Thresh_{est}$ are P. An indicator of the estimator accuracy is provided by contrasting the estimated number of points $\hat{P}$ against the actual remaining ones P (error $= |\hat{P} - P|$).

The experimental results are very encouraging and indicate an average error on the estimated number of points in the range of 5% of the data window. For this experiment, if the predicted number of points for a certain threshold is 250 points, the actual value of remaining points could be (for example) 200 points. This is the case of an *overestimated threshold* which compressed more the flowing data stream. As mentioned before, this case is more desirable (than an underestimated threshold), because no additional points need to be subse-

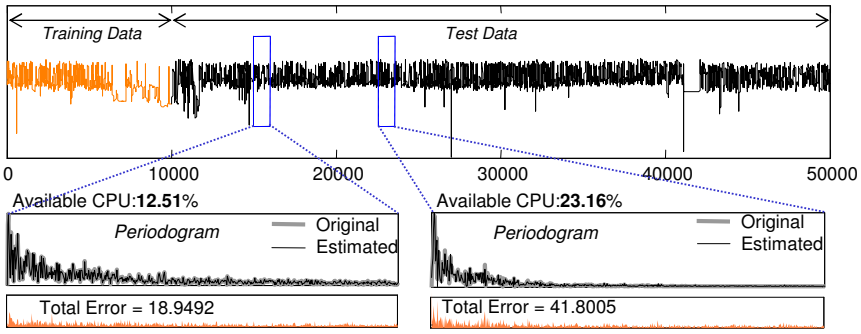

Fig. 11. A deployment of our algorithm on streaming automotive measurements. We constrast the estimated spectrum with the original one at two instances of the sliding window.

quently dropped from the current data window (which is not bound to introduce additional aliasing problems).

A histogram of the estimator approximation error is given on the left part of Fig. 12. We observe that for the majority of data windows the estimation error is small, while fewer instances of the algorithm execution report a large error in the threshold estimation. On the right part of Fig. 12 we also provide how many cases of overestimated thresholds we have and how many underestimated. The overestimated ones (more desirable) are higher than the underestimated, which again indicates many of the attractive properties of the proposed threshold predictor.

6 Conclusion

We have presented the first resource-adaptive method for periodicity estimation. The key aspects of the proposed method are: (1) An intelligent load-shedding scheme that can adapt to the CPU load using a lightweight predictor. (2) A DFT estimation that utilizes unevenly spaced samples, provided by the previous phase. We have shown the quality of the approximated DFT and we also demonstrated that our scheme can adapt closely to the available CPU resources. We compare our intelligent load-shedding scheme against equi-sampling and we show improvements in the periodogram estimation ranging from 10% to 90%. As part of future work, we plan to examine whether it is possible to reduce even further the computational cost. This could be achieved by investigating the possibility of a 'butterfly' structure [3] in the incremental spectrum computation. We also plan to modify the sub-sampling algorithm in order to support provable bounds on the periodogram approximation error.

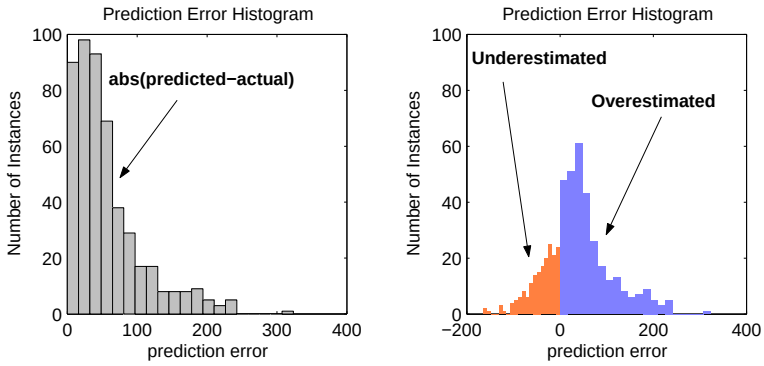

Fig. 12. *Left*: Histogram of the threshold estimator error. *Right*: Cases of *overestimated threshold* (fewer remaining samples -more desirable) are more frequent than instances of underestimated threshold.

References

1. D. Broomhead and D. Lowe. Multivariate functional interpolation and adaptive networks. In *Complex Systems, 2: 321:355*, 1988.
2. P. Castiglioni, M. Rienzo, and H. Yosh. A Computationally Efficient Algorithm for Online Spectral Analysis of Beat-to-Beat Signals. In *Computers in Cardiology:29, 417:420*, 2002.
3. J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex Fourier series. In *Math. Comput. 19, 297:301*, 1965.
4. P. Cuadra, A. Master, and C. Sapp. Efficient Pitch Detection Techniques for Interactive Music. In *International Computer Music Conference*, 2001.
5. M. G. Elfeky, W. G. Aref, and A. K. Elmagarmid. Using Convolution to Mine Obscure Periodic Patterns in One Pass. In *EDBT*, 2004.
6. F. Ergün, S. Muthukrishnan, and S. C. Sahinalp. Sublinear methods for detecting periodic trends in data streams. In *LATIN*, 2004.
7. W. Fan. StreamMiner: A Classifier Ensemble-based Engine to Mine Concept-drifting Data Streams. In *Proc. of VLDB*, pages 1257–1260, 2004.
8. A. C. Gilbert, S. Guha, P. Indyk, S. Muthukrishnan, and M. Strauss. Near-optimal Sparse Fourier Representations via Sampling. In *STOC*, pages 152–161, 2002.
9. M. Kontaki and A. Papadopoulos. Efficient similarity search in streaming time sequences. In *SSDBM*, 2004.
10. M. Lazarescu, S. Venkatesh, and H. H. Bui. Using Multiple Windows to Track Concept Drift. In *Intelligent Data Analysis Journal, Vol 8(1)*, 2004.
11. S. Papadimitriou, A. Brockwell, and C. Faloutsos. Awsom: Adaptive, hands-off stream mining. In *VLDB*, pages 560–571, 2003.
12. A. Papoulis. *Signal Analysis*. McGraw-Hill, 1977.
13. W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 1992.
14. Time-Series Data Mining Archive. http://www.cs.ucr.edu/ eamonn/TSDMA/.
15. Y. Zhu and D. Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. In *VLDB*, 2002.