International Journal of Computing and Optimization
Vol. 4, 2017, no. 1, 159 - 186
HIKARI Ltd, www.m-hikari.com
https://doi.org/10.12988/ijco.2017.758

Finite State Automaton Based Signal Acquisition with Bootstrap Learning

Stephen Winters-Hilt

Computer Science Department and Biology Department
Connecticut College
270 Mohegan Ave.
New London, CT 06320, USA
&
Meta Logos Inc.
124 White Birch Dr.
Guilford, CT 06437, USA

Abstract

It is commonplace to need to acquire a signal where the signal properties are not known, or the signal is only suspected and not discovered yet, or the signal properties are known but they may be too much trouble to enumerate. There is no common solution, however, to the acquisition task. For this reason the initial phases of acquisition methods unavoidably tend to be *ad hoc*. As with data dependency in non-evolutionary search metaheuristics, where there is no optimal search method that is guaranteed to always work well, here there is no optimal signal acquisition method known in advance. In this paper methods are described for bootstrap optimization in signal acquisition. The bootstrap algorithmic method involves repeated passes over the data sequence, with improved priors, and trained filters, among other things, to have improved signal acquisition on subsequent passes. The signal acquisition is guided by statistical measures to recognize anomalies. Informatics methods and information theory measures are central to the design of a good finite state automata acquisition method, and will be reviewed in the signal acquisition context. Code examples are given in Perl and C. Bootstrap acquisition methods may not automatically provide a common solution, but appear to offer a process whereby a solution can be improved to some desirable level.

Keywords: FSA, HMM, Gabor limit

1 Introduction

Signal acquisition often begins with the need to acquire a signal where the signal properties are not known, and the initial signal processing acquisition methods are, thus, unavoidably *ad hoc*. What is desirable in this situation is a method where signals and their attributes can be discovered or identified in some automated manner. *Ad hoc* signal acquisition refers to finding the solution for 'this' situation (whatever that is) without consideration of wider application. The solution is strongly data dependent in other words. Data dependent methodologies are, by definition, not defined at the outset, but must be invented as the data begins to be understood. As with data dependency in non-evolutionary search metaheuristics, where there is no optimal search method that is guaranteed to always work well, here there is no optimal signal acquisition method known in advance. This is simply restating a fundamental limit from non-evolutionary search metaheuristics in another form [1]. What can be done, however, is assemble the core tools and techniques from which a solution can be constructed and to perform a bootstrap algorithmic learning process with those tools (examples in what follows) to arrive at a functional signal acquisition on the data being analyzed. A universal, automated, bootstrap learning process may be possible using evolutionary learning algorithms (related to the co-evolutionary Free Lunch Theorem [2]), but that is not part of this paper.

"Bootstrap" refers to a method of problem solving when the problem is solved by seemingly paradoxical measures (the name references Baron von Munchausen who freed the horse he was riding from a bog by pulling himself, and the horse with him, up by his bootstraps [3]). Such algorithmic methods often involve repeated passes over the data sequence, with improved priors, or a trained filter, among other things, to have improved performance. The Background and Methods describe bootstrap FSA processes. The bootstrap amplifier from electrical engineering is an amplifier circuit where part of the output is used as input, particularly at start-up (known as bootstrapping), allowing proper self-initialization to a functional state (by amplifying ambient circuit noise in some cases) [4]. The bootstrap FSA proposed here is a meta-algorithmic method in that performance 'feedback' with learning is used in algorithmic refinements with iterated meta-algorithmic learning to arrive at a functional signal acquisition status.

Acquisition is often all that needed in a signal analysis problem, where a basic means to acquire the signals is sought, to be followed by a basic statistical analysis on those signals and their occurrences. Various methods for signal acquisition using Finite State Automaton (FSA) constructs are described in what follows that focus on statistical anomalies to identify the presence of signal and 'lock on' [5,6]. The signal acquisition is initially only guided by use of statistical measures to recognize anomalies. Informatics methods and information theory measures are central to the design of a good FSA acquisition method, and will be reviewed in the signal acquisition context [5], along with hidden Markov Models (HMMs).

HMMs offer a more sophisticated signal recognition process than FSAs, but with greater computational space requirements and computational time complexity. Like electrical engineering signal processing, HMMs usually involve pre-processing that assumes linear system properties or assumes observation is frequency band limited and not time limited, and thereby inherit the time-frequency uncertainty relations, Gabor limit, and Nyquist sampling relations. FSA methods, on the other hand, can be used to recover (or extract) signal features missed by HMM or classical electrical engineering signal processing. Even if the signal sought is well understood, and a purely HMM-based approach is possible, this is often needlessly computationally intensive (and slow), especially in areas where there is no signal. To address this there are numerous hybrid FSA/HMM approaches (such as BLAST [7]) that benefit from the O(L) complexity on length L signal with FSA processing, with more targeted processing, as needed, at $O(lN^2)$ complexity with HMM processing (where there are N states in the HMM model and l is the length of the local signal region).

Many signal features of interest are time limited and not band limited in the observational context of interest, such as noise ‘clicks’, ‘spikes’, or impulses. To acquire these signal features a time-domain FSA (tFSA) is often most appropriate. Human hearing, for example, is a non-linear system that thereby circumvents the restrictions of the Gabor limit to allow musicians with ‘perfect pitch’ whose time-frequency acuity surpasses what should be possible by linear signal processing alone [8], such as with Nyquist sampled linear response recording devices that are bound by the limits imposed by the Fourier uncertainty principle (or Benedick’s theorem) [9]. Thus, even when the powerful HMM feature extraction methods are utilized to full advantage, there is often a sector of the signal analysis that is only conveniently accessible to analysis by way of FSA’s, such that a parallel processing with both HMM and FSA methods is often needed (results demonstrating this in the context of channel current analysis [5] will be briefly discussed). The methods employed at the FSA processing stage relate to not only standard Fourier transform based signal processing approaches, but also to purely statistical methods such as with oversampling [10] (used in radar range oversampling [11,12]) and dithering [13] (used in device stabilization and to reduce quantization error [14,15]).

2 Background

In the Background section that follows an overview is given of information measures (Sec. 2.1) and of the uses of time-domain Finite State Automata (tFSAs) for signal acquisition (Sec. 2.2). Background on a specific tFSA implementation for ‘spike’ (impulse) event detection is given in Sec. 2.3.

2.1 Entropic measures, statistical linkage, and Mutual Information

The degree of randomness in a discrete probability distribution P is measured in terms of Shannon entropy [5]:

$$S(P) = \Sigma_k\, p_k \log(p_k). \text{ where P has outcome probabilities } \{p_k\}.$$

When comparing discrete probability distributions P and Q, both referring to the same N outcomes, the proper measure of their difference is measured in terms of their (often symmetrized) relative entropy [5] (a.k.a. Kullback-Leibler Divergence):

$$D(P||Q) = \Sigma_k\, p_k \log(p_k/q_k). \text{ where P and Q have outcome probabilities } \{p_k\} \text{ and } \{q_k\}.$$

In evaluating if there is a statistical linkage between two events X and Y we are essentially asking if the probability of those events are independent, e.g., does $P(X,Y) = P(X)P(Y)$? Since this reduces to measuring the difference between two probability distributions: $P(X,Y)$ and $Q(X,Y)=P(X)P(Y)$, the relative entropy between P and Q is sought, where $D(\, P(X,Y) \,||\, P(X)P(Y)\,)$ is the definition of ‘mutual information between $\{X,Y\}$: $MI(X,Y) = D(\, P(X,Y) \,||\, P(X)P(Y)\,)$.

Mutual information allows statistical linkages to be discovered that are not otherwise apparent (and requires no model assumptions, so is a core discovery method in machine learning). Consider the mutual information between nucleotides in genomic data when different gap sizes are considered between the nucleotides as shown in Fig. 1 Left. When the MI for different gap sizes is evaluated (see Fig. 1 Right), a highly anomalous long-range statistical linkage is seen, consistent with a three-element encoding scheme (the codon structure is thereby revealed) [16].

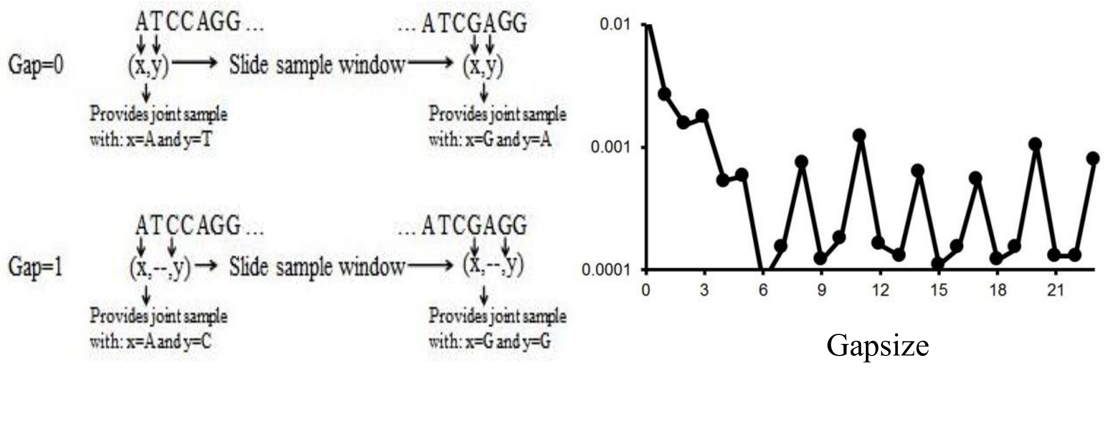

Fig. 1. Codon structure is revealed in the V. cholera genome by mutual information between nucleotides in the genomic sequence when evaluated for different gap sizes.

Once codon groupings are revealed, a frequency analysis on the different codons can be done, and the stop codons are thereby revealed (due to their anomalously low counts). Focusing on the stop codons it is easily found that the gaps between stop codons can be quite anomalous compared to the gaps between other codons when stepping across the gap in-frame (see Fig. 2). Codon framing refers to a series of codon windows, each three bases wide, that "steps" (non-overlapping three-base windows) across a nucleic acid sequence. ORFs are "open reading frames", where the reference to what is open is lack of encounter with a stop codon {(taa),(tag),(tga)} when traversing the genome with a particular codon framing , e.g., ORFs are regions devoid of in-frame stop codons when traversed with the codon framing choice of the ORF. In discussions of ORFs in most of the analysis we restrict to ORFs of length 300 bases or greater. The restriction to larger ORFs is due to their highly anomalous occurrences and likely biological encoding origin, e.g., the long ORFs give a strong indication of containing the coding region of a gene. By restricting to transcripts with ORFs >= 300 in length, we have a resulting pool of transcripts that are mostly *true* coding transcripts.

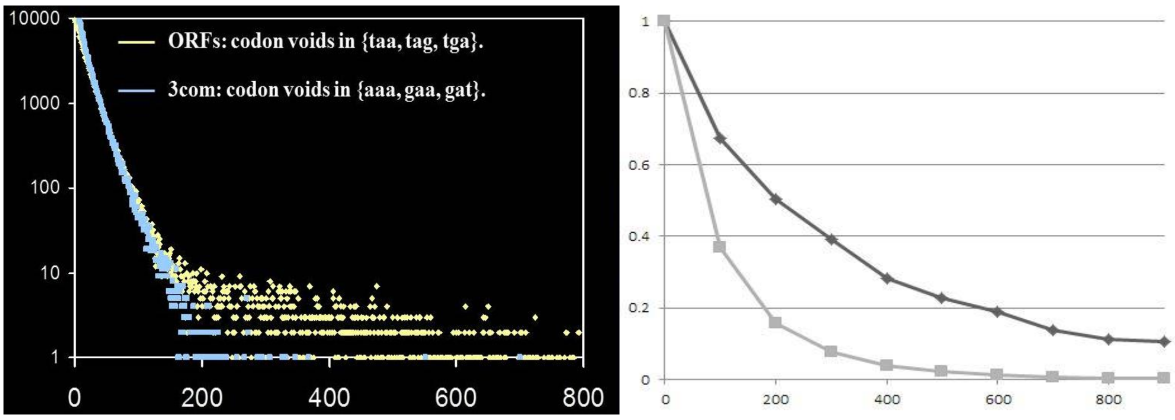

Fig. 2. ORF encoding structure is revealed in the V. cholera genome by gaps between stop codons in the genomic sequence. Left Panel: X-axis shows the size of the gap in codon count between reference codons (stops for conventional ORFs, or 3com set for comparisons in table), Y-axis shows the void counts (log scale). The Genome examined was *V. cholerae*. Reprinted with permission [16]. **Right panel:** X-axis shows the size of the gap in base count between the (tag) codons (dark curve with diamonds) and between (aaa) codons (gray curve with squares). The Y-axis shows the void counts (linear scale) normalized to the zero-void (adjacent) counts of 2115 for (tag) and 21256 for (aaa), obtained for the zero-frame pass on the forward strand of the *E. coli* genome (so approximately 1/6 of the number of such voids in the full genome).

The ORF discovery example shows a bootstrap FSA process on genomic data: first scan through the genomic data base-by-base and obtain counts on nucleotide pairs with different gap sizes between the nucleotides observed [5, 16].

This then allows a mutual information analysis on the nucleotide pairs taken at the different gap sizes. What is found for prokaryotic genomes (with their highly dense gene placement), is a clear signal indicating anomalous statistical linkages on bases three apart [5, 16]\. What is discovered thereby is the aforementioned codon structure, where the coding information comes in groups of three bases. Knowing this, a bootstrap analysis of the 64 possible 3-base groupings can then be done, at which point the anomalously low counts on 'stop' codons is then observed. Upon identification of the stop codons their placement (topology) in the genome can then be examined and it is found that their counts are anomalously low because there are large stretches of regions with no stop codon (e.g., there are stop codon 'voids', known as open reading frames, or 'ORF's). The codon void topologies are examined in a comparative genomic analysis in [5, 16]. The stop codons, which should occur every 21 codons on average if DNA sequence data was random, are sometimes not seen for stretches of several hundred codons (see Fig. 2). This is like flipping heads a hundred times in a row. For the genomic data what is found are the regions that contain the longer genes, whose anomalous, clearly non-random DNA sequence, is being maintained as such, and not randomized by mutation, as this would be selected against in the survival of the organism. This basic analysis can provide a gene-finder on prokaryotic genomes that comprises a one-page Perl script that can perform with 90 to 99% accuracy depending on the prokaryotic genome [16] (see Code Sample #1 in Sec. 3.1). A second page of Perl coding to introduce a 'filter', along the lines of the bootstrap learning process mentioned above, leads to an *ab initio* prokaryotic gene-predictor with 98.0 to 99.9% accuracy [5,16] (see Sec. 3.1 for code samples). In this bootstrap process all that is used is the raw genomic data (with its highly structured intrinsic statistics) and methods for identifying statistical anomalies and informatics structural anomalies: (i) anomalously high mutual information is identified (revealing codon structure); (ii) anomalously high (or low) statistics on an attribute or event is then identified (low stop codon counts, lengthy stop codon voids); then anomalously high sub-sequences (binding site motifs) are found in the neighborhood of the identified ORFs (used in the filter).

2.2. tFSA signal acquisition

All of the tFSA signal acquisition methods described in this paper are O(L) for data size 'L', i.e., they scan the data with a computational complexity no greater than that of simply seeing the data (via a 'read' or 'touch' command). Because the signal acquisition is only O(L) it is not significantly costly, computationally, to simply repeat the acquisition analysis multiple times with a more informed process with each iteration, to arrive at a 'bootstrap' signal acquisition process. In such a setting, initial rounds of signal acquisition are often done with bias to very high specificity (and sensitivity very poor), to get a 'gold standard' set of highly likely true signals that can be data mined for their attributes. With a filter stage thereby trained, later scan passes can pass suspected signals with very weak specificity (very high sensitivity now) with high specificity then recovered by use of the filter. This then allows a bootstrap process to a very high specificity (SP) and sensitivity (SN) at the tFSA acquisition stage on the signals of interest (see [5], and the references cited there, for more details).

The same procedure outlined for the above signal discovery and acquisition on genomic data can be applied to analysis of any set of stochastic sequential data. If the data is numerical, e.g., observational data on an electrical current, or stock price, or time series in general, then even more tools from statistics can be brought to bear (e.g., the expectation of a real number observational sequence can be calculated, while the same can't be done, directly, with the symbol-based {a,c,g,t} genomic data). In what follows a description of a time-domain FSA on real-number observational data (channel current readings) will be given. Further details will be given in the Methods, including practical implementation considerations such as working with raw integer encoded data (not floating point representation until absolutely needed). Working wither integer representation allows a 20-fold speedup for every processing loop working with integer multiplication instead of floating point multiplication.

In [17, 18], a time-domain finite state automaton with eight states is used for signal identification and acquisition on the first 100 ms of channel current blockade signal (see Fig. 3 and the code description in the Methods). Two states, sequentially connected, were used for resetting and initializing the FSA. Transition between the two states, from reset-start to reset-ready, was accomplished upon measuring a short section of acceptable baseline current (200 μs). An abrupt drop in current to 70% residual current (determined by holistic tuning), or less, then triggered transition from the reset-ready state to the signal-active state. From the signal-active state, processing advanced to one of two states (good- and bad-end-level states) according to an end-of-signal profile. The profile rule simply required that the last end-level-range observations had to have current above minimum-end-level-value. Satisfying the rule led to the good-end-level state, otherwise the bad-end-level state was reached. If there was a normal return to baseline (good-end-level state), or a signal-blockade scan exited due to truncation (bad-end-level state), the signal complete state was reached, otherwise further scanning was performed. Further scanning involved transition through the internal active state, where local signal properties, observation less than maximum-cutoff and observation greater than minimum-cutoff, were used to decide whether to exit (to the reset-end state) or continue the blockade scan (return to the signal-active state). Similar to the local blockade signal properties that determined how to transition from the internal-active state, transition to the acquire-signal state from the signal-complete state was based on several global properties of the signal trace: maximum blockade sample less than maximum-cutoff and greater than min-max-internal, minimum blockade sample greater than minimum-cutoff and less than max-min-internal, and signal duration greater than or equal to minimum-duration.

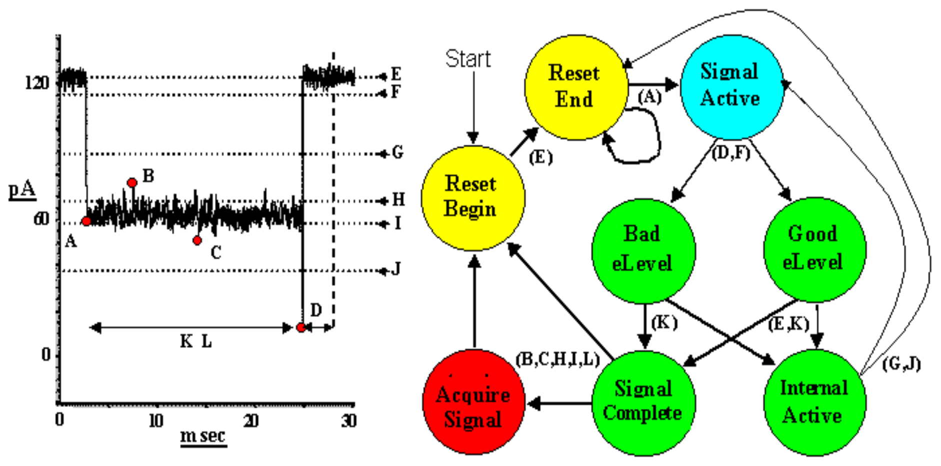

Fig. 3. Schematic for the finite state automaton used for acquisition of 6 base-pair DNA hairpin blockade signals observed in [17, 18], where a sample signal is shown to the left. The letters label various types of feature extraction parameters and their placement in the FSA diagram indicate where the decision-making or thresholding is dependent on those parameters. Reprinted with permission [18].

The FSA shown in Fig. 3 [17,18] was eventually tuned to operate such that it would rarely miss signal acquisitions (low false negatives) by allowing for large numbers of mistaken signal acquisitions (i.e., large false positives), followed by filtering to achieve high specificity. The acquisition bias was accomplished by imposing constraints on valid starts that were weak while maintaining constraints on valid interior and ends that were strong. The bias towards high sensitivity for *initiating* acquisition permitted tuning on FSA parameters with a simplified objective (part of the benefit of a multi-pass bootstrap tuning process). For the blockade signatures studied, the FSA parameters for maximal signal acquisition shared a broad, common range, allowing one set of FSA parameters (a single generic FSA) to acquire all signals. There aren't many examples of FSA acquisition code in the literature, so the code for the main loop underlying the tFSA in Fig. 3 is given in the Methods.

The FSA described in Fig. 3 enables acquisition of localizable channel current signals using 'holistic' tuning and 'emergent grammar' tuning. (Emergent grammar tuning, and use of wavelets, is described in [5, 18] and won't be discussed further here.) When attempting to tune the FSA it can be viewed as a "holistic engine" of a multiply connected (not independent) set of variables, states, and their interactions. For acquisition we seek minimal feature identification comprising identification of signal beginnings and ends (and thus durations as well). *Holistic tuning is mainly done by testing global features for anomalous changes, or 'phase transitions'*. One of the main global features of the acquisition process is the number of acquisitions itself, made under a particular set of tuning parameters. In Fig. 4 is shown the result of a holistic tuning process on the start_drop_value parameter. A critical requirement for holistic tuning is having a viable initial tuning state to initialize the process, e.g., multiple parameters must be within their 'lock range' on tuning parameters analogous to the PLL lock-range constraint [5,18]. The code description, with the core tuning parameters highlighted, is in the Methods.

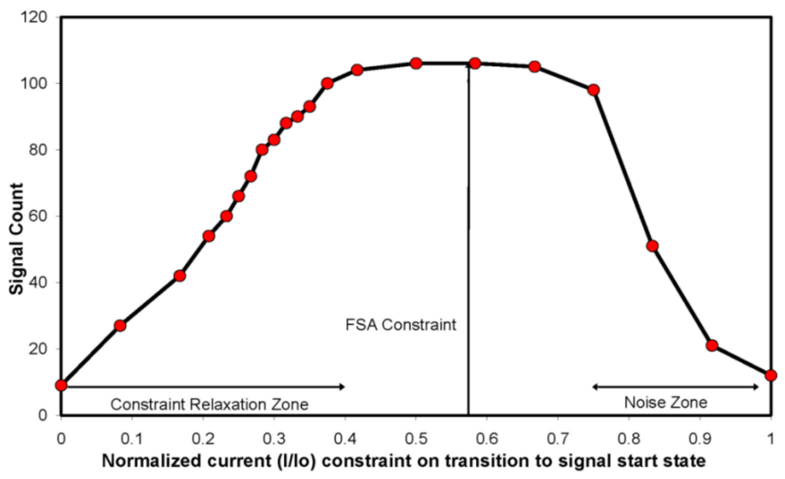

Fig.4. Tuning on 'start_drop_value' for a collection of blockade signals resulting from channel captures of DNA hairpins with 6 base-pair stem length. For baseline-normalized current constrained to drop to 0 channel current to trigger possible acquisition we see that very few acquisitions succeed (approximately 10 signal acquisitions shown). As we relax this start of acquisition constraint on possible signal acquisitions, we steadily see more signal counts until it plateaus starting at a baseline-normalized current of 0.4 to a baseline-normalized current of 0.7. The paradoxical seeming drop in signal acquisitions for the more hair-trigger acquisitions for baseline-normalized current drop, to only 0.8 or greater, is due to the FSA often triggering on noise, and eventually rejecting the indicated signal as invalid, but in doing so sometimes missing a valid signal start, resulting in fewer overall signal acquisitions. The holistic tuning process seeks the plateau region (that is not directly responsive to change in cutoff over a broad range) as an indication of a robust acquisition setting, with the 0.57 value chosen in the example shown. Reprinted with permission [18].

The O(L) time-complexity feature identification "scan" process can be employed for simultaneous feature extraction on various statistical moments, as mentioned previously. Identification of sharply localizable 'spike' behavior can also be done in the scan process (still with only O(L) time complexity) based on a nonparametric method that is described next.

2.3 tFSA spike detector

A channel current spike detector algorithm can be used to characterize the brief, very strong, blockade "spike" behavior observed for duplex DNA molecular termini that occasionally fray in the region exposed to the limiting aperture's strong electrophoretic force region. (See [5] for details, where nine base-pair hairpins were studied, and the spike events were attributed to a fray/extension event on the terminal base-pair.) A complication with the spike feature extraction is the blockade level from which the spike event occurs is not known, or too variable to use to identify the spike blockade event. To have a robust feature extraction a level-crossing heuristic was used, where for a fixed blockade level the number of signal crossings at that level are counted (such as from spikes). The test level used in the crossing analysis is then done at different blockade levels, with increasing crossing counts as the test-level gets near a signal blockade level. What results is linear increase in crossing count for actual spike features as the test level used in the crossing analysis is brought closer to the blockade signal's dominant blockade levels. In the case of the channel current analysis the various levels of blockade seen for a particular molecular blockade typically have Gaussian noise about the average of each level. Thus, as the line-crossing approaches the mean of the signal blockade 'level', it probes the tail of the Gaussian noise distribution about that signal blockade level, with an exponential increase in level crossings (see Fig. 5). Focusing on the linearly increasing count region, and extrapolating to the counts when at the signal blockade level from which the spike deflections are seen, a count on spike events (or a frequency on spike events) can then be robustly ascertained [5].

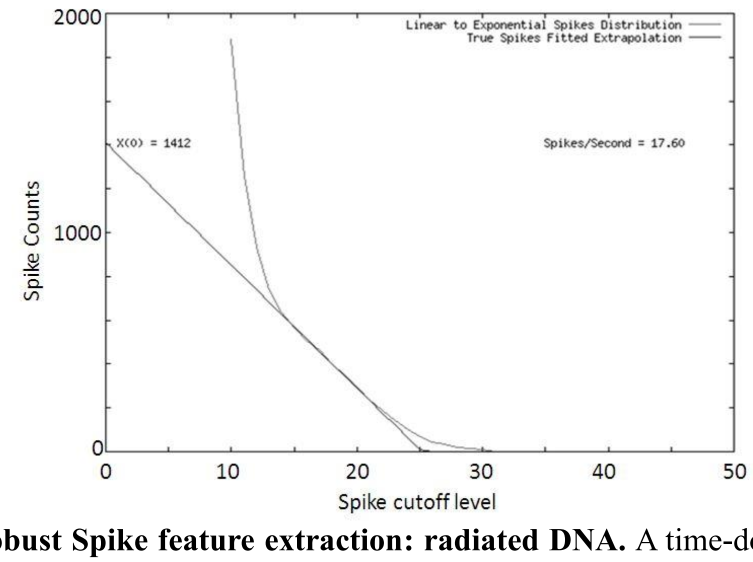

Fig. 5. Robust Spike feature extraction: radiated DNA. A time-domain FSA is used to extract fast time-domain features, such as "spike" blockade events. Automatically generated "spike" profiles are created in this process. One such plot is shown here for a radiated 9 base-pair hairpin, with a fraying rate indicated by the spike events per second (from the lower level sub-blockade). Results: the radiated molecule has more "spikes" which are associated with more frequent "fraying" of the hairpin terminus--the radiated molecules were observed with 17.6 spike events per second resident in the lower sub-level blockade. Reprinted with permission [5].

The spike detector software is designed to count “anomalous” spikes, i.e., spike noise not attributable to the Gaussian fluctuations about the mean of the dominant blockade-level. The extrapolations provide an estimate of “true” anomalous spike counts. Together, the formulation of HMM-EM, FSAs and Spike Detector provide a robust method for analysis of channel current data [5]. In Fig. 5 the plot is automatically generated for spike characteristics for blockade data for the DNA hairpins examined: one with cross-linking radiation damage and one without damage. The plots are also automatically fit with extrapolations of their linear phases (shown). By this method, the non-radiated DNA exhibited a full-blockade “spike” from its lower-level blockade with a frequency of 3.58 spikes per second (indicating a fraying of the blunt ended terminus of the molecule at that rate). For the radiated molecule the frequency of spikes was 17.6 spikes per second, indicating a much greater fraying rate (and associated dissociation of the terminal base-pair), consistent with that duplex DNA molecule being weakened by radiation such that its terminal base-pair frays more frequently.

The additional “spike” frequency feature is found to improve classification accuracy between two species of DNA hairpins by approximately 5% in the hairpin discrimination SVM tuning that is scored for various kernel parameters in Fig. 6. This is an example of how non band limited signal features can be extracted without the limitations of a HMM state quantization pre-processing (or Fourier transform method feature extraction from electrical engineering signal processing) to arrive at a more informed process than seems possible given the usual constraint of the Gabor limit, as mentioned in the Introduction.

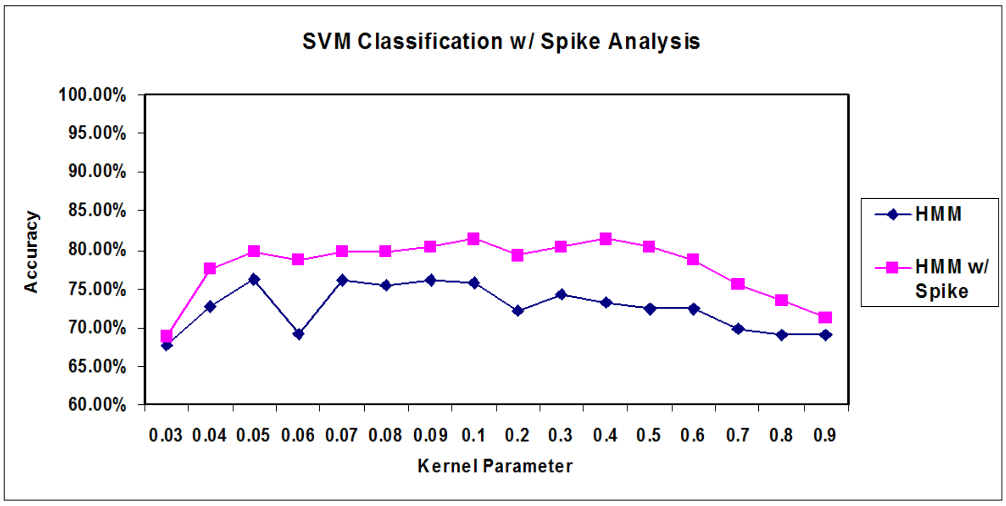

Figure 6. SVM classification results with and without spike analysis. Adding a spike feature significantly improves classification accuracy, by approximately 5%, over a wide range of kernel parameters. Reprinted with permission [5].

Once the lifetimes of the various levels are obtained, information about a variety of other kinetic properties is accessible. If the experiment is repeated over a range of temperatures, a full set of kinetic data is obtained (including the aforementioned "spike" feature frequency analysis). This data may be used to calculate k_{on} and k_{off} rates for binding events, as well as indirectly calculate forces by means of the van't Hoff Arrhenius equation [5].

3 Methods and Results via bootstrapped algorithmic refinements

An overview of FSA-based prokaryotic gene-finder code is given in Sec. 3.1, including a description of the bootstrap learning for the FSA algorithm. Variations on the FSA-based ORF-finder code for prokaryotic genomes can be applied to both prokaryotic and eukaryotic transcriptomes, but these are described and used elsewhere [19] and, [20], so won't be discussed further here. In Sec. 3.2 tFSA methods are given for channel current cheminformatics (CCC) when there is stable channel baseline reference signal. In Sec. 3.3 are tFSA methods for CCC with unstable baseline. In Sec. 3.4 are efficient implementations of the FSA and core statistical tools.

3.1 FSA-based gene-finder code for prokaryotic genomes

The code that follows is an example of a bootstrap learning process, beginning with a subroutine that finds ORFs longer than a specified cutoff, and moves in-frame from the left to the first 'atg' codon, for which a tentative coding region is identified, as well as the upstream 'cis' region and downstream 'trans' regions. Extracts of the coding, cis, and trans regions are taken. The crude `longORFfirstATG_geneFinder` indicated in Code Sample #1 will catch many true gene regions, but the 'first atg' heuristic for the start of the coding region will fail 1-10% of the time depending on the genome. This is where a filter is obtained by looking for anomalous high count motif structures in the cis region. Even if 10% of the genes so indicated are incorrect, the strongly recurring motifs tallied from the 90% correctly delineated genes will provide a clear indication of the cis regulatory motifs found in valid cis regions. This then provides a filter test for a second-pass (bootstrap) genefinder that has a validation test on the cis region. This boosts the genefinder accuracy to 99% for many prokaryotes genomes (if not sufficiently accurate for a genome of interest, similar refinements could be made for validation on trans motifs and on codon usage).

If we examine the gap size distribution from start of ORF (from the left) to the first codon of some type we find an anomalous peak in the occurrence of (atg) codons (e.g., the distribution does not fall off exponentially with gap size as would be expected if random (see Fig. 7).

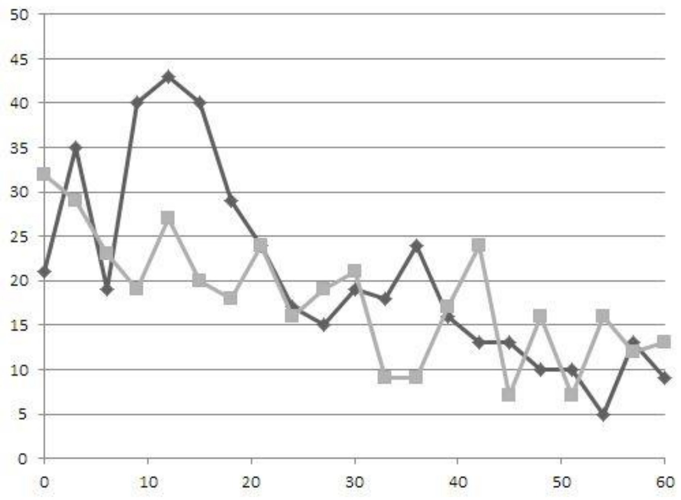

Figure 7. Counts on gap sizes from left ORF boundary to first occurrence of various codons. The dark curve with diamonds shows the number of occurrences (y-axis) of the various gap sizes in bases (x-axis) to the first ‘atg’, not the anomolos peak at gap size 12. The gray curve shows the more normal exponential fall-off on gap size counts with increasing gap size.

Let's proceed under the assumption that the gene starts with the first ‘atg’ in the ORF and try to find associated structures that will offer a means to validate the first 'atg' as the start, so begin effort by capturing the 'first atg genes' in the long-ORFs, along with their pre (cis) and post (trans) regions.

Code Sample #1: `longORFfirstATG_geneFinder`

```
sub longORFfirstATG_geneFinder {
    my ($ref,$frame,$start,$length,$ciswindow) = @_;
    my @dna = @{$ref};
    my $seq_length = scalar(@dna);
    my %count;
    my $index;
    my $gene_output_fh = new FileHandle ">genefile";
    my $cisgene_output_fh = new FileHandle ">cisfile";
    my $transgene_output_fh = new FileHandle ">transfile";
    my $oldindex=0;
    for $index (0..$seq_length-1-2) {
        if ($index%3 != $frame) { next; }
        my $codon = "$dna[$index]$dna[$index+1]$dna[$index+2]";
        if ($codon eq 'taa' || $codon eq 'tag' || $codon eq 'tga') {
            my $gap = $index - $oldindex;
            if ($gap>$length) {
                my $i;
FOR2:           for $i ($oldindex+3..$index-3) {
                    if ($i%3 != $frame) { next FOR2; }
                    my $cdn = "$dna[$i]$dna[$i+1]$dna[$i+2]";
                    if ($cdn eq $start) {
                        my @gene = @dna[$i..$index+2];
                        my $geneseq = join('',@gene);
                        my @cisgene = @dna[$i-$ciswindow..$i-1];
                        my $cisgeneseq = join('',@cisgene);
                        my @transgene = @dna[$index+3..$index+$ciswindow];
                        my $transgeneseq = join('',@transgene);
                        $gene_output_fh->print("$geneseq\n");
                        $cisgene_output_fh->print("$cisgeneseq\n");
                        $transgene_output_fh->print("$transgeneseq\n");
                        last FOR2;  # taking only first start
                    }
                }
            }
            $oldindex=$index;
        }
    }
}
```

By running longORFfirstATG_geneFinder we generate output files for the hypothesized gene region and the preceding (cis) region and the following (trans) region. We now want to look through the collection of cis regions to see if there are any anomalously occurring DNA substrings (motifs) that may associate with the hypothesized start of gene region. The subroutine to do this, `line_oligo_counter`, is shown in Code Sample #2.

Code Sample #2: `line_oligo_counter`

```
longORFfirstATG_geneFinder(\@dna,0,'atg',500,30);

sub line_oligo_counter {
    my ($filename,$order,$anom_mult) = @_;
    if (!$anom_mult) { $anom_mult = 5; }
    my $data_input_fh = new FileHandle "$filename";
    my $types = 4**$order; #types of $order 'mers'
    my $samples=0;
    my $lines=0;
    my $label;
    my %count;
    while (<$data_input_fh>) {
        chomp;
        s/\s//g;
        s/\d//g;
        if (s/^>//) {
            $label = $_;
        }
        else {
            tr/ACGT/acgt/;
            my @dna = split //, $_;
            my $seq_length = scalar(@dna);
            $lines++;
            $samples += $seq_length-$order+1;
            my $index;
            for $index (0..$seq_length-1-$order+1) {
                my $xmer;
                my $i;
                for $i (0..$order-1) {
                    $xmer .= $dna[$index+$i];
                }
                $count{$xmer}++;
            }
        }
    }

    my @seen_xmer_types = keys %count;
    my $expected_average_count = $samples/$types;
    my $anomalous_count_cutoff = $expected_average_count*$anom_mult;
    print "acc=$anomalous_count_cutoff\n";
    print "number of dna segments = $lines\n";
    print "anom_mult=$anom_mult\n";

    my $above_acc_count=0;
    my %motifs;
    my $xmer;
    foreach $xmer (@seen_xmer_types) {
        if ($count{$xmer}>$anomalous_count_cutoff) {
            $above_acc_count++;
            print "count{$xmer}=$count{$xmer}\n";
            $motifs{$xmer}=$count{$xmer};
        }
    }
    print "above_acc_count=$above_acc_count\n";
    return %motifs;
}
```

In the above while loop, that scans thru each cis region, we are getting counts on the oligomers ("xmers" in code) of the specified order. In what follows we then estimate the number of occurrences of a particular type of xmer when the data is random (uniform probability distribution) – this is referred to as the expected_average_count. A cutoff is then introduced for when the number of occurrences of a particular motif is anomalous by use of a multiplier, anom_mult. All motifs with counts above the indicated cutoff are deemed anomalous and thus potentially of interest as a signaling motif that is paired with the 'atg' start of coding.

There is found to be a lengthy list of anomalous hexamers occurring in the window 15 bases prior to the start codon (see Table 1). Doing a bootstrap algorithmic refinement going forward, we can simply look for an occurrence of one of these hexamers to validate any hypothesized start codon. Failure to validate on the first 'atg' would then lead to looking for the next, in-frame, 'atg' as a possible start. It turns out that 'atg' is used for start only 90 to 99% of the time, the other main start codon being 'gtg'. The percentage of genes starting with 'atg' is genome specific. For some strains of *E. Coli*, about 99% of the genes start with 'atg', for the most common strain of *Vibrio Cholerae*, only about 93% of the genes start with 'atg', 6.9% starting with 'gtg'. 0.1% starting with 'ttg', and very rarely, some genes starting with 'ctg'. To handle this, if we don't have validation via an occurrence of the Shine-Dalgarno motif (aaggaa) or other cis motifs in the window 15 based prior to the hypothesized start, then we want to continue to step codon-by-codon (in-frame) into the ORF (from left to right) until the next 'atg' or 'gtg' is encounted, where the validation is attempted again. This is then repeated until either validation achieved, or the remaining length of the ORF drops below our ORF length cutoff. In the code thus far we have focused on ORFs>300 in length, to ensure the very likely collection of true gene regions. Now that we have a validation test, we can relax the length-anomaly filter to not be quite so stringent, and repeat the datarun with filter refinements. By using the aforementioned subroutines, followed by the subroutine `geneFinder`, shown in Code Sample #3, a gene finder with start validation is achieved (with further refinements as necessary in further iterations).

aaaaggaa	9	aggaggaa	21	gagaaa	10	ggggaa	10
aaaagg	14	aggagc	10	gagaag	9	gtggag	9
aaagga	20	aggagt	20	gagcag	9	taaagg	10
aaaggt	10	aggata	9	gaggaa	12	taagga	29
aacagg	10	aggatg	9	gaggat	9	tcagga	12
aaggaa	20	agggaa	9	gaggtt	10	tgagga	

```
                                my @gene = @dna[$i..$index+2];
                                my $gene_length = scalar(@gene);
                                if ($gene_length<90) {
                                    print "exiting ORF search at $i\n";
                                    last FOR2;
                                } #length cutoff
                                my $end = $index+2;
                                print "gene: start=$i, end=$end\n";
                                my $geneseq = join('',@gene);
                                $gene_output_fh->print("$geneseq\n");
                                $cisgene_output_fh->print("$cisgeneseq\n");
                                $transgene_output_fh->print("$transgeneseq\n");
                                last FOR2;  # ORF extraction complete
                            }
                        }
                        # if here no motif match, shift to next 'atg' OR 'gtg'
                        $alt_start = 'gtg';
                        print "doing repeat start pass at index=$i\n";
                        next FOR2;  # taking next possible start
                    }
                }
            }
            $oldindex=$index;
        }
    }
}
geneFinder(\@dna,0,'atg',240,$motifref);
```

At this point we have a complete genefinder (see Code Sample #3), with cis motif discovery code and code for use of those motifs for start-of-gene validation. What is missing however are the repeated passes over the genomic data for the different frames (referred to as frame 0, 1, and 2 in what follows). Since there are three possible framings that can occur on the genomic data when you have a three-element encoding scheme. The other framings can be obtained by repeating the analysis just done with the first base removed (for the frame ‘1’ case), or the first two bases removed (for the frame ‘2’ case). Having obtained the gene predictions for the three possible framings there is a further subtlety that must be dealt with having to do with the fact that the genomic sequence information provided is for only ‘half’ of the genome. This is because the sequence information is typically obtained from double stranded DNA genomes, such that we need to repeat the entire analysis for the other strand. The other strand is fully specified by the first, however, since the second strand pairs with the first strand according to the Watson-Crick base-pairing scheme, where you have A pairing with T (and vice-versa) and C pairing with G. The read direction of the other strand is in the opposite direction to the read direction of the first (an explanation of the biochemistry is in [5]). This means that the gene count is actually approximately six times that indicated from the frame 0 ‘positive’ strand analysis done thus far. If we use the motif results from just the frame 0 positive strand above, to keep things simple, we can repeat the analysis directly for the other strands to obtain a fully functional prokaryotic gene-finder. Further details of this are in [5, 16, 19, 20].

```
gene: start=336, end=2798
performing repeat start pass at index=7365
gene: start=12162, end=14078
performing repeat start pass at index=15444
performing repeat start pass at index=15606
performing repeat start pass at index=15666
performing repeat start pass at index=15840
performing repeat start pass at index=16062
gene: start=16074, end=16556
gene: start=18714, end=19619
performing repeat start pass at index=20697
performing repeat start pass at index=20727
performing repeat start pass at index=20853
gene: start=20874, end=21062
performing repeat start pass at index=25206
gene: start=25251, end=25700
gene: start=30816, end=34037
gene: start=34299, end=34694
gene: start=35445, end=35903
………
```

Fig. 9. **Code Sample #3 Results**. The partial results shown above show how some genes are successfully validated at the first ‘atg’ (those entries not preceded by attempts at a “repeat start pass”), while other genes require one or more repeat start searches due to validation failures on the start codon.

3.2 tFSA-based channel signal acquisition methods with stable baseline

The tFSA program shown in Fig. 3 begins with State=“Reset Begin” (see comments in Code Sample #4 below) with a loop to self a minimum of 10 times on the sample data being scanned, where the data was sampled at 20us [17, 18], thus minimum time to advance from the “Reset Begin” state is 0.2 ms in the application shown (via baseline_to_reset=10 in the code shown below). In order to only do loop 10 times, the observed blockade value must exceed the open_channel_avg value on each sample of the 10 observations. Until 10 such observations exceeding the open_channel_avg value are tallied, whether consecutively or not, the loop will not advance from “Reset Begin” to “Reset End”. The baseline_to_reset parameter is reset to 10 after each possible signal acquisition is resolved (with acquisition or rejection). The value of 10 is itself chosen by the ‘holistic’ tuning process mentioned in Sec. 2.

Once at the State “Reset End”, the blockade sample values are checked (shown as self-loop in Fig. 3) to see if they’ve dropped significantly from the reset condition (e.g., dropped below baseline). This will be the first of a series of instances where weak conditions are used on initiating signal acquisitions, while much stricter conditions must be satisfied later (when better informed about the signal) in order to complete, and fully acquire, the signal. A blockade sample observation is deemed to have “dropped significantly” from its reset condition if it drops below a cutoff named the “start_drop_value” in Fig. 3 and the code below, to arrive at the next state, “signal active”, for initiating signal acquisition. Sometimes a blockade sample drops right through the floor, however, to large negative values, etc., due to noise or a shock, etc. These falsely triggered signals are excluded by excluding start drops that go below the "start_drop_limit" value. (Again, all parameters are tuned.) Once at the State "Signal Active", each subsequent blockade sample is read into an array, for possible signal acquisition and recording, and for use by O(1) data analysis algorithms (keeping the overall FSA operation O(L) on L observation samples). Such algorithms are used to calculate simple statistical properties of the blockade region (in an O(1) process), such as the maximum, minimum, duration, and (running) average of the blockade signal, and the (running) standard deviation of the blockade signal. The notion of a 'running' statistical evaluation is that the initialization of the statistical parameter may be O(N), for N length observation in the signal scan window, but that as the windowing on data used in the scan operation is slid along the data observation sequence, further updates on that sliding-window statistical parameter is only O(1). This sliding-window, or 'running', evaluation, then allows higher order statistical moments to be computed at O(L) on the full observation sequence under study (code implementations for this will be shown in detail in Sec. 3.4 for the first two statistical moments: mean and variance).

Code Sample #4: The `tFSA code, written in C, that goes with the FSA in Fig. 3.`

```
while (index<length) {
    fread(&d,2,1,datafile);
    rescale = (double) d/scale;
    if (baseline_to_reset>0) {
        if (rescale>open_channel_avg) { baseline_to_reset--; }
        index++; continue;
    }
    if (start_active<1 && rescale<start_drop_value && rescale>start_drop_limit) {
        signal_start[j] = index;
        signal_max[j] = rescale;
        signal_min[j] = rescale;
        sigindex=0;
        sigdata[sigindex] = rescale;
        start_active = 1;
        get_base_lead = 0;
        index++; continue;
    }
    if (start_active<1) { index++; continue; }
    else {
        sigindex++;
        sigdata[sigindex] = rescale;
        bad_end_level=0;
        for (i=0;i<end_level_range;i++) {
            if (data[i]<end_level_value) { bad_end_level=1; i=end_level_range;}
        }
        signal_end[j] = index-1-end_level_range;
        signal_length = signal_end[j] - signal_start[j] + 1;
        if (signal_length>max_length) {
            signal_end[j] += 1+end_level_range;
            signal_length += 1+end_level_range;
        }
        if ((bad_end_level<1 && rescale>open_channel_avg) ||
                                      (signal_length > max_length)) {
            if (signal_length>min_length
                && signal_min[j]<max_min_internal) {
                total_pts = sigindex-end_level_range;
            do_simple_profile(sigfile,signal_start,signal_end,
                              signal_max,signal_min,total_pts,
                              sigdata,j);
            printf("signal %d processing complete\n",j);
            sigindex=0; //resets signal info
            j++;
        }
        baseline_to_reset=10;
        start_active=0; //resets
        get_base_lead = 1;
        index++; continue;
    }
    else if (((index-signal_start[j]>end_level_range)&&
                  (data[end_level_range-1]>max_internal))
             || rescale<min_internal) {
        start_active=0; //resets
        get_base_lead = 1;
        index++; continue;
    }
    else if (mod_index>0 && ((index%mod_index<mod_index_range) ||
               (index%mod_index>mod_index-mod_index_range))) {
        start_active=0; //resets
        get_base_lead = 1;
        index++; continue;
    }
    else {
        if ((index-signal_start[j]>end_level_range)&&
               (data[end_level_range-1]>signal_max[j])) {
            signal_max[j]=data[end_level_range-1];
        }
        if (rescale<signal_min[j]) { signal_min[j]=rescale; }
        index++; continue;
    }
}
index++;
}
```

As a preliminary step for each new sample acquisition, to minimize bad-acquisition blocking on good signal that might immediately follow, exit conditions are tested for channel blockade completion (i.e., a return to the baseline, open channel, current readings). In Fig. 3, the exit condition is obtained either by a return to baseline (bad_end_level=0, or State=”Good eLevel”), or due to acquisition truncation (case with State=”Bad eLevel”, due to truncation, even though good for acquisition). Once an exit condition is reached, a proper signal acquisition has occurred (with data already loaded into the acquisition array), and we now arrive at the State ”Signal Complete”. A collection of signal conditions are then tested on the total signal data for the final acceptance or rejection decision.

In what follows a distributional profile of the blockade levels, with attention to those most frequented, e.g., the statistical modes, will be used in trying to lock onto signal features of interest. Sample code for a modal scan is given in Code Sample #5, where use is also made of integer-based variables for 20-fold speedup on multiplications with integer variables instead of floating point variables (as mentioned in the Background). Working with integer-valued data is not as lossy as it might seem at first since the data encoding schemes used by third-party DAQs and amplifier developers, and by their efficient binary datafile encodings (if saved to file), are themselves integer-based with shift and float multiplication operations to bring the data back to the floating value originally observed, with the resolution (number of significant figures) used in the recording or quantization process.

Code Sample 5:

```
sub Scan_Data_Modes { # used to do binary data read and identify stat modes
    my ($self,$input_file,$type,$mode) = @_;
    my $Data_fh = new FileHandle "$input_file";
    binmode($Data_fh);
    my $header_file="header_file";
    my $Header_out_fh = new FileHandle "$header_file";
    binmode($Header_out_fh);
    my $abf_file="$input_file" . ".abf";
    my $Data_out_fh;
    while (read($Data_fh,$buffer,4)) {
        $index++;
        my ($value,$vvalue) = unpack 'ss', $buffer;
        $data_index = $index-$header_skip;
        $abf_data[$data_index]=$value;
        my $binned_sample_value = $abf_data[$data_index];
        $mode_count{$binned_sample_value}++;
    }
    close($Data_fh);
    # …..output
}
```

Analysis is usually done on data with reference to a past window of data in the data sequence. Suppose you use a reference window of data with the 1000 samples preceding the current data instance in the sequence. To determine if the current data instance is a significant departure from “the norm” we first define the norm to be what can be ascertained from analysis of the 1000-sample window, such as its mean (standard deviation test), variance (standard deviation test), modes (mode tests), and modal transitions (monotonicity tests). For the standard deviation test you determine the mean and standard deviation of the reference window, then choose a cutoff (common is any data instance above the mean plus three standard deviations; or below mean minus three standard deviations). For the mode test you begin by identifying the modes. In channel current analysis one dominant mode for much of the time, when nothing is present at the channel, is the ‘baseline’ signal, referred to as the baseline mode in what follows. If there is one type of blockade signal, then there will typically be one other significant mode in the modal analysis, the blockade mode. Using the Std. Dev. Test and the automated discovery of the baseline and blockade modes, it is possible to have a signal acquisition process based on the identified modal information alone, but the ‘edge discovery’ can be weak for the leading edge and ending edge of the signal.

For improved edge detection consider a second reference window centered on the current data instance being analyzed and ask if the values are all above or all below (no crossing) or all above transitioning to all below. The latter describes the sought after channel blockade event edge detection situation, where going from a sequence of observations all above the current value to a sequence all below, is catching the position of the ‘falling edge’ (with the reverse for the ‘rising edge’ detection). For one ‘line crossing’ on a horizontal ‘test line’ overlaid on the data at the data instance value position, i.e., where the line segments connecting the observed data values result in curve that crosses the horizontal reference line only once, then the edge is identified and passes the monotonicity test. Sometimes small window averaging is needed to smooth the edge a little to pass the monotonicity test (this only works if there is really an edge there), this is usually identified as a filter condition and handled earlier (eventually implemented at the hardware level). Once there is strong, validated, edge detection on the beginning and ending of signals, more sophisticated statistical moment analysis can be done on the blockade modes themselves, to arrive at tests to validate signal as potentially good (or not obviously bad) during the initial acquisition process itself. Additional signal characteristics can be used to acquire signals if there are stable modal references, particularly in the example of channel current cheminformatics [5]. Having a stable baseline, however, is sometimes precisely the problem, where a stable baseline must typically be ‘recovered’ to then access the stable-baseline acquisition methodologies. Numerous results involving the nanopore data acquisition are given elsewhere [5], so aren’t shown here.

3.3 tFSA-based channel signal acquisition methods without stable baseline

In channel current blockade analysis, and electrical signal analysis in general, the tFSA signal acquisition is much more difficult if the reference baseline is not stable. For electrical signals one type of unstable baseline that typically results is from capacitive effects giving rise to an exponential rise (or fall) in the baseline current when fast event sampling is done before system relaxation can occur (to steady baseline). If the exponential rise/fall in the channel current signal was the same this could simply be factored out, but typically the charge/discharge reset is incomplete, and the capacitive properties themselves are variable under load, giving rise to (effectively) different exponential rise/fall baseline references with every device reset. Even this extreme case can be handled with a properly designed tFSA (one making use of filters analogous to notch filters from electrical engineering, for example).

Using the start_drop_value parameter introduced already one seeks to identify onset of blockade events by their deviation, typically expressed by some multiple of standard deviations, from the baseline mean. As mentioned previously, a ‘three-sigma’ rule is often used, i.e., event acquisition onset is triggered when a channel current reduction by more than three standard deviations of baseline noise, from the baseline mean, is observed. The falling edge of the blockade onset can then be precisely fixed (down to a specific sample observation in many cases) by performing a monotonicity test on the falling edge, which can be done with an O(L) line-crosser analysis like that used in the spike analysis [1]. Fixing the start of blockade then depends on the data processing conventions adopted (often one chooses whatever convention yields the best classification/clustering at later (SVM) processing stages, if in use). Whatever the convention, the core, ‘stable’, information that guides the blockade level identification is a modal analysis on the different blockade levels seen. What is revealed by this is a mode identifying the baseline level (at least in the region just prior to the triggered signal acquisition) and the blockade level. If there is stable baseline, or even stable capacitive discharge baseline (i.e., a single exponential profile occurring with each sampling reset), then the modal analysis will directly reveal that baseline, or identification of the asymptotic, 'relaxed', exponential baseline level. Once a suspected blockade signal onset has occurred, passing the standard deviation, monotonicity, and mode tests, signal acquisition commences on further streaming signal, with ongoing 'running' O(L) measurement of statistical moments and {max, min} sample values, with rejection if the 'internal' signal statistics does not fall within the desired range of possibilities. Identification of end-of-blockade, i.e., identification of the rising edge back to baseline current with no blockade, is then much the same as the falling edge identification, with use of standard deviation, monotonicity and modal tests. Typically identification of return to baseline requires additional measurement of a minimum number of baseline samples after return to baseline to avoid premature truncation on signal acquisitions that are sufficiently noisy that their internal blockade fluctuations occasionally 'spike' back to baseline level.

If the channel current blockade signals have an unstable baseline, then a window-based FSA, and elaborations on it, can be critical to locking onto the signal, but may not be optimal at calling the edges, thus multiple passes may be needed (bootstrap acquisition), with early passes involving the window-based tFSA to get a preliminary lock so that the unstable base-line moving average can be subtracted, shifting to a simpler acquisition where a sample-based tFSA can then take over for the final signal acquisition with the most precise edge recognition possible. Results for the nanopore data acquisition with unstable baseline are particularly relevant with the laser-tweezer driven modulations [21-23] and situations where bilayer permeability becomes highly variable [21-23], but are beyond the scope of this paper and will not be discussed further.

3.4 Efficient implementations for statistical tools (O(L))

Working with the native integer encoded binary representation of the data is faster on multiple levels. This would not be of much benefit, however, if the subroutines for the statistical moments (mean, standard deviation, etc.) could not operate at the integer variable level for most of their evaluation. An implementation of statistical methods for evaluating the mean and standard deviation at integer-variable level is shown in Code Sample 6. A *window-based* tFSA implementation using these methods (instead of single-sample based tFSA implementation like that shown in Code Sample 4) is discussed further in [5]. The window-based implementation is more robust with variable, unstable, baseline, but is less precise at identifying falling edges and other sharp transitions. Since the window based method often involves sums over the window, it is sometimes called an integration (or calculus-based) tFSA [5, 6].

Code Sample 6: non-lossy statistical moment evaluations that are integer-based at leading order in time complexity

```
my $Get_Lowpass = sub {
    my ($abf_ref,$sample_index,$offset,$window_size,$factor) = @_;
    my @abf_data;
    if ($abf_ref) {
        @abf_data = @{$abf_ref}[$sample_index+$offset-
                          $window_size+1..$sample_index+$offset];
    }
    else {
        print "error in passing raw data array\n";
    }
    my $i;
    my $window_sum=0;
    for $i (0..$window_size-1) {
        $window_sum += $abf_data[$i];
    }
    my $mean = $factor*$window_sum/$window_size;
    return $mean;
};

my $Get_Std_Dev = sub {
    my ($abf_ref,$sample_index,$offset,$window_size,$factor) = @_;
    my @abf_data;
    if ($abf_ref) {
        @abf_data = @{$abf_ref}[$sample_index+$offset-
                           $window_size+1..$sample_index+$offset];
    }
    my $mean = $Get_Lowpass->
                ($abf_ref,$sample_index,$offset,$window_size,$factor);

    my $i;
    my $sum_squared_central_moment=0;
    my $factorlessmean = int($mean/$factor+0.5);
    for $i (0..$window_size-1) {
         my $diff = $abf_data[$i]-$factorlessmean;
        $sum_squared_central_moment += $diff*$diff;
    }
    my $variance=$sum_squared_central_moment/$window_size;
    my $std_dev = $factor*sqrt($variance);
    return $std_dev;
};
```

4 Discussion

Sometimes the more sophisticated window-based tFSA methods can be avoided entirely by use of a notch filter (a form of lowpass filter) as a preprocessing stage, code for which is shown in Code Sample 7. In the worst case scenario, all of these methods need to be used. The process that might be undertaken on a challenging signal acquisition, thus, might go as follows:
(1) scan for asymptotic baseline statistics
(2) do a preliminary window-based FSA scan to get a handle on the baseline
(3) estimate the baseline signal
(4) perform a sample-based tFSA scan on the baseline-subtracted signal
(5) perform repeated tFSA scans (since fast) with different biases to lock onto all signal regions
(6) perform notch filter on raw signal in indicated signal regions with identified baseline attributes used to determine the optimal notch filter.

(7) perform merge on signal acquisitions indicated at steps (5) and (6)

Code Sample 7:

```
sub Lowpass_Notch_Filter {
    my ($self,$notch_window,$buffer_window) = @_;
    my @abf_data = @{$self->{abf_data_ref}};
    my $total_sample_count=$self->{total_sample_count};
    if (!$notch_window) { $notch_window=1000; }
    if (!$buffer_window) { $buffer_window=100; }

    print "starting notch filter evaluation\n";
    my $start_time = time;

    my @notch_filter_data;
    my @lowpass_filter_data;
    my @notch_window_array;
    my $sample_index;
    my $notch_window_sum;
    my $notch_window_avg;
    for $sample_index (0..$total_sample_count-1) {
        my $sample_value = $abf_data[$sample_index];
        if ($sample_index < $notch_window) {
            my $pop = $notch_window_array[0];
            my $i;
            for $i (0..$notch_window-2) {
                $notch_window_array[$i] = $notch_window_array[$i+1];
            }
            $notch_window_array[$notch_window-1] = $sample_value;
            $notch_window_sum += ($sample_value-$pop);
            $notch_window_avg = $notch_window_sum/($sample_index+1);

            $lowpass_filter_data[$sample_index]=$notch_window_avg;
        }
        else {
            my $pop = $abf_data[$sample_index-$notch_window];
            my $push = $abf_data[$sample_index];

$lowpass_filter_data[$sample_index]=$lowpass_filter_data[$sample_index-1]
                                                +($push-$pop)/$notch_window;
        }
    }

    for $sample_index (0..$total_sample_count-1) {
        my $sample_value = $abf_data[$sample_index];
        if ($sample_index < $notch_window+$buffer_window) {
            $notch_filter_data[$sample_index]=$abf_data[$sample_index]-
            $lowpass_filter_data[$sample_index+2*$notch_window+$buffer_window];
        }
        else {
            $notch_filter_data[$sample_index]=$abf_data[$sample_index]-
            0.5*($lowpass_filter_data[$sample_index-$notch_window-$buffer_window]
            + $lowpass_filter_data[$sample_index+2*$notch_window+$buffer_window]);
        }
    }

    $self->{notch_filter_data}=\@notch_filter_data;

    my $end_time = time;
    my $timediff = int($end_time-$start_time);
    print "notch filter scan done in $timediff seconds.\n";
}
```

5 Conclusion

Bootstrap acquisition methods provide a common methodology for performing signal acquisition in a variety of settings. In this paper finite state automata (FSA) methods are described with bootstrap optimization for signal acquisition, where repeated passes over the data sequence are done with improved priors and trained filters to have improved signal acquisition. Use of informatics measures, such as Shannon entropy, relative entropy, and mutual information, are found to be useful in quantifying relations and discovering statistical linkages (shown for discovering the codon encoding scheme, among other things). The FSA methods shown are also described in terms of implementations maintaining their O(L) computational complexity on length L sequence, and of having operability at the level of integer operations (predominantly) for fastest processing speed.

Acknowledgements. The author would like to thank NIH, NSF, NASA, and the Louisiana Board of Regents for research support. The author would also like to thank Meta Logos Inc., for research support and a research license. (Meta Logos Inc. was co-founded by the author in 2009; and Meta Logos Systems was founded by the author in 1997.)

References

[1] D.H. Wolpert and W.G. Macready, No Free Lunch Theorems for Optimization, *IEEE Transactions on Evolutionary Computation*, **1** (1997), 67-82. https://doi.org/10.1109/4235.585893

[2] D.H. Wolpert and W.G. Macready, Coevolutionary free lunches, *IEEE Transactions on Evolutionary Computation*, **9** (2005), no. 6, 721-735. https://doi.org/10.1109/tevc.2005.856205

[3] R.E. Raspe, *Baron Munchhausen's Narrative of his Marvellous Travels and Campaigns in Russia*, London 1785.

[4] IEEE Standard *100-2000 The Authoritative Dictionary of IEEE Standards Terms*, Seventh Edition, IEEE Press, 2000, 123. https://doi.org/10.1109/ieeestd.2000.322230

[5] S. Winters-Hilt, *Machine-Learning Based Sequence Analysis, Bioinformatics & Nanopore Transduction Detection*, Lulu, 2011.

[6] S. Winters-Hilt, Method and System for Stochastic Carrier Wave Communications, Radio-Noise Embedded Steganography, and Robust Self-Tuning Signal Discovery and Data-Mining, Pat. Pend. June, 2015, 62/186827.

[7] S. Altschul, Gish, W. Miller, E. W. Myers, D. Lipman, Basic local alignment search tool, *Journal of Molecular Biology*, **215** (1990), no. 3, 403–410. https://doi.org/10.1016/s0022-2836(05)80360-2

[8] J.N. Oppenheim and M.O. Magnasco, Human Time-Frequency Acuity Beats the Fourier Uncertainty Principle. Phys Rev Lett. **110** (2013), no. 4. https://doi.org/10.1103/physrevlett.110.044301

[9] M. Benedicks, On Fourier transforms of functions supported on sets of finite Lebesgue measure, *J. Math. Anal. Appl.*, **106** (1985), no. 1, 180–183. https://doi.org/10.1016/0022-247x(85)90140-4

[10] Silicon Laboratories Inc., Improving ADC Resolution by Oversampling and Averaging, Retrieved 17 January 2015.

[11] S.M. Torres and D.S. Zrnic, Whitening in Range to Improve Weather Radar Spectral Moment Estimates. Part I: Formulation and Simulation, *J. Atmospheric and Oceanic Tech.*, **20** (2003), 1433-1448. https://doi.org/10.1175/1520-0426(2003)020<1433:wirtiw>2.0.co;2

[12] S.M. Torres and D.S. Zrnic, Whitening of Signals in Range to Improve Estimates of Polarimetric Variables, *J. Atmospheric and Oceanic Tech.*, **20** (2003), 1776-1789. https://doi.org/10.1175/1520-0426(2003)020<1776:wosirt>2.0.co;2

[13] K.C. Pohlmann, *Principles of Digital Audio*, McGraw-Hill Professional, 2005.

[14] L. Schuchman, Dither Signals and Their Effect on Quantization Noise, *IEEE Trans. Communications*, **12** (1964), no. 4, 162–165. https://doi.org/10.1109/tcom.1964.1088973

[15] Analog Devices: A Technical Tutorial on Digital Signal Synthesis, 1999. http://www.analog.com/static/imported-files/tutorials/450968421DDS_Tutorial_rev12-2-99.pdf.

[16] S. Winters-Hilt, Hidden Markov Model Variants and their Application, *BMC Bioinformatics*, **7** (2006), no. S2, S14. https://doi.org/10.1186/1471-2105-7-s2-s14

[17] S. Winters-Hilt, W. Vercoutere, V. S. DeGuzman, D. Deamer, M. Akeson and D. Haussler, Highly Accurate Classification of Watson-Crick Base-Pairs on Termini of Single DNA Molecules, *Biophys. J.*, **84** (2003), 967-976. https://doi.org/10.1016/s0006-3495(03)74913-3

[18] S. Winters-Hilt, *Machine Learning Methods for Channel Current Cheminformatics, Biophysical Analysis, and Bioinformatics*, PhD Dissertation, UCSC, University of California, 2003.

[19] S. Winters-Hilt and J. Evanilla, Characterization of fish stock diversity via EST analysis, Submitted May 2017.

[20] S. Winters-Hilt, RNA-dependent RNA polymerase encoding artifacts in eukaryotic transcriptomes, Submitted March 2017.

[21] S. Winters-Hilt, Nanopore Transducer Engineering and Design, *Int. J. MolBiol. Med.*, **2** (2017), no. 1,

[22] S. Winters-Hilt, Biological System Analysis Using a Nanopore Transduction Detector: from miRNA Validation, to Viral Monitoring, to Gene Circuit Feedback Studies, *Advanced Studies in Medical Sciences*, **5** (2017), no. 1, 13 – 53. https://doi.org/10.12988/asms.2017.722

[23] S. Winters-Hilt, Isomer-Specific Trace-Level Biosensing Using a Nanopore Transduction Detector, *Clinical and Experimental Medical Sciences*, **5** (2017), no. 1, 35-66. https://doi.org/10.12988/cems.2017.722

Received: May 21, 2017; Published: July 18, 2017