Supplement to: Exploiting Conserved Structure for Faster Annotation of Non-coding RNAs Without Loss of Accuracy

Zasha Weinberg* and Walter L. Ruzzo†

October 6, 2004

1 Introduction

This paper adds supplementary technical information to the paper "Exploiting Conserved Structure for Faster Annotation of Non-coding RNAs Without Loss of Accuracy" [3]. In particular, a fully automated scheme to design efficient rigorous filters is described, and some other technical issues avoided in the paper are discussed here.

This supplement assumes the reader has read that paper, and is familiar with CMs as they are traditionally presented in the literature, e.g. from [2, chpt. 9].

Issues considered are:

- Additional technical details on filters. Mentions details on the implementation of sub-CM filters.
- Conceptual issues with filter creation and selection. Describes issues that the automated method must deal with.
- The automated process for filter series creation & selection.
- How accurate are the approximations used in filter selection. A number of approximations and associated assumptions were used to efficiently select a series of filters; this sections considers how accurate these approximations are in practice.

2 Additional technical details on filters

Section 5.2 of the paper says that, in the augmented HMM with sub-CM rooted at state S_i, the Viterbi score to state $\overline{S}_i^R$ is the highest sum of sub-CM score ($\overline{S}_i^L$ to $\overline{S}_i^R$) plus HMM score for state $\overline{S}_i^L$. In fact, the sub-CM (in accordance with the standard CM/SCFG algorithm) finds Viterbi scores from $\overline{S}_i^L$ to $\overline{S}_i^R$ for a given length sequence. (The sub-CM score from $\overline{S}_i^L$ to $\overline{S}_i^R$ is equivalent to the score for the CM rooted at S_i.) We therefore must consider various lengths, up to W', in stitching together the HMM and sub-CM Viterbi paths.

*Dept. of Computer Science & Engineering, University of Washington, Seattle, WA, 98195, USA, zasha@cs.washington.edu

†Depts. of Computer Science & Engineering and Genome Sciences, University of Washington, Seattle, WA, 98195, USA, ruzzo@cs.washington.edu

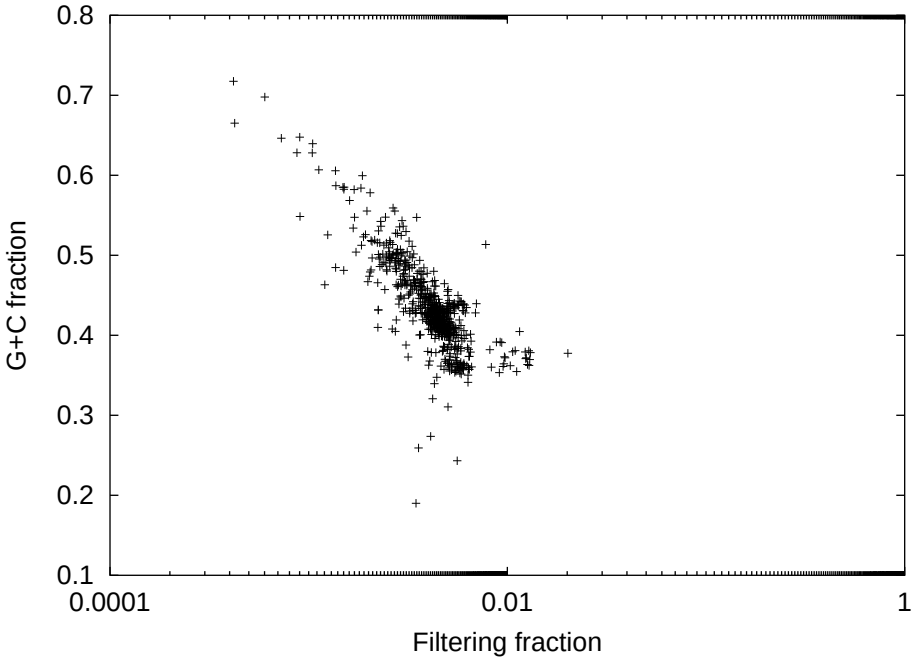

Figure 1: Filtering fraction correlates with G+C content.
The 8 Gbase RFAMSEQ genome database was divided into 5 Mbase chunks. For each chunk the G+C content was calculated as a fraction of nucleotides, and the filtering fraction was calculated for a profile HMM rigorous filter built from RF00029 in Rfam 4.1. The correlation coefficient is -0.64.

To find the augmented HMM's Viterbi score to state $\overline{\mathrm{S}}_i^R$ at nucleotide position j,

for $w = 0 \ldots W'$ **do**
score $\leftarrow$ (augmented HMM score to state $\overline{\mathrm{S}}_i^L$ at nucleotide position $j - w$) + (sub-CM score from state $\overline{\mathrm{S}}_i^L$ at position $j - w$ to state $\overline{\mathrm{S}}_i^R$ at position j).
best-score $\leftarrow$ max(best-score,score)
end for

To implement this algorithm efficiently, during the augmented HMM Viterbi algorithm, for each state $\overline{\mathrm{S}}_i^L$ with sub-CM rooted at state i, the Viterbi score to state $\overline{\mathrm{S}}_i^L$ is stored for the most recent W' positions.

3 Conceptual issues in filter creation & selection

3.1 G+C content

G+C content strongly relates to filtering fraction, as shown in Figure 1. For some families, filtering is easier with high G+C, while for others it is easier with low G+C. The difference between high and low G+C is dependent on the family. (Intuitively, it depends on the typical G+C content of family members.) For example, for Rfam 5.0 family RF00010, the filtering fraction on the *E. coli* genome is 0.749; on the *S. aureus* genome it is 0.069 (a difference of an order of magnitude). By contrast, for RF00015, the filtering fraction is 0.00148 for *E.*

coli and 0.00150 for *S. aureus* (no significant difference at all). It is generally reasonable to plan filtering based on sequence data corresponding to the the least favorable G+C content, because (1) what is optimal in the worst case is typically close to optimal in easier cases, and (2) since filtering fractions are much higher in the worst case, worst case performance tends to dominate the time taken for a large database scan. (We note that a more sophisticated scheme that selects different filter series depending on G+C content may improve performance somewhat, but can be risky: it is difficult to rule out the possibility that a sequence has small regions of difficult G+C content, and if a more aggressive filter series is selected for this region, it may be extremely inefficient.)

3.2 Appropriate test sequence sizes & "pre-filtering"

We now consider what is an appropriate sequence size to test filtering fraction. With a low filtering fraction and a small sequence, the estimated fraction may represent a small number of nucleotides whose upper bound scores exceeded the score threshold. Since these are rare events, our estimates are not very reliable. We must therefore use large sequences in order to obtain reasonable estimates when the filtering fraction is relatively small.

Unfortunately, larger test sequences require more CPU time to scan; to obtain a perfect estimate, we could scan the entire database, but this is clearly not practical if we wish to test many filters whose CPU time requirements may not be that much smaller than the CM's. We need to make intelligent trade-offs between the need for accuracy and the time required to test.

To solve this problem, we use *pre-filtering*. A large test sequence is used, and is pre-filtered once with a less selective filter. The pre-filter has a higher filtering fraction than the filter we wish to test. We then test the filter on the pre-filtered sequence. This pre-filtering strategy allows a more efficient test of highly selective filters, since it only tests them on the harder subsequences. Details on this scheme appear later.

3.3 Reliability of CPU timing

To estimate run time, the computer's clock is used to time the filtering process. Our logic assumes that this process will give consistent results.

We have found this assumption to be violated on multi-processor machines, where timing a subroutine can give radically different results, with times varying often by a factor of 2. On uniprocessor machines, we have found times to be consistent to within a fraction of a second, so we use these uniprocessor machines for filter creation & selection. We have not investigated the cause of the inconsistency on multi-processor machines.

4 The automated process for filter series creation & selection

At the beginning, we wish to rigorously filter some database sequence with a given CM. We are given:

- The database sequence, e.g. the approximately 8 Gigabase RFAMSEQ.
- The CM.
- A score threshold above which hits should be reported.

Our technique assumes that the above input data is correct; this supplement does not consider the question of how to create a CM or score threshold or how to select a genome database.

We are also given several parameters supplied by the user, which are described as they are used.

4.1 Selection of worst-case G+C content.

All filters are measured against sequences that have the worst-case G+C content. Also, the profile HMMs need to be optimized for a 0th-order Markov model (using the infinite-length forward algorithm); for this 0th-order model, we use the worst-case G+C content. The worst-case G+C content is determined in the following manner.

We consider three possible G+C contents, based on bacterial genomes: *Staphylococcus aureus* MW-2 (very low), *E. coli* K-12 (medium) and *Bordetella bronchiseptica* (very high). We will decide which of these bacterial genomes represents the hard case in the following steps:

1. Create a profile HMM filter based on the CM.
2. Test the profile HMM on each genome.

Create a profile HMM. The first step requires a profile HMM. The variable `hardGenomeHmmType` says what method to use to build the HMM. The main choices are to build a compact-type or an expanded-type profile HMM. In both cases, the profile HMM is optimized to a uniform 0th-order model.

Another option is to create a profile HMM using techniques designed for heuristics not rigorousness. The advantage of this technique is that the creation process is very fast, although its performance will not correlate as well as the rigorous profile HMM techniques. (The heuristic technique will be published soon.)

Test profile HMM on each genome. To test the profile HMM, we have two options, controlled by the parameter `hardGenomeScoreMethod`. The obvious technique is to run the profile HMM on a test sequence, and use the average Viterbi score as an indicator; the genome with the highest Viterbi score is presumed to have the hardest G+C content. The actual test sequence used can be smaller than the genome. To generate the test sequence, a 2nd-order Markov model is learned from each genome, and a sequence of a user-supplied size is generated. The size is specified in the parameter `hardGenomeViterbiSeqSize`.

The other possibility is to use the infinite-length forward algorithm result [4]. For each test genome, a 0th-order model is learned. Then the infinite-length forward algorithm is run on the given test profile HMM using this 0th-order model. The genome with the highest infinite-length forward algorithm result is presumed hardest.

Which method should be used? In practice, all of these techniques tend to give highly similar results. Moreover, when they disagree (e.g., one method indicates that *E. coli* was hardest, while another indicates *Staphylococcus*), the difference in filtering fractions between the genomes was usually small, so the discrepancy is insignificant.

The advantage of the heuristic profile HMM and the infinite-length forward algorithm is that these are quick to evaluate, whereas other techniques take more time. However, a lot of work follows the selection of the hard genome, so spending more time estimating the

hard G+C content will not dominate overall time. Moreover, an error in determining the hard G+C content can result in a poor choice of filters. Therefore, we recommend using the expanded-type rigorous profile HMM and the average Viterbi score.

For the eukaryotic tRNAscan-SE model, the use of the heuristic profile HMM causes *Bordetella* to be considered the hard genome, whereas in fact *S. aureus* is the hard genome. This is a serious difference since the two genomes are at opposite extremes of G+C content, and *Bordetella* is very easy for this model. So the resulting filters are too agressive; on genomes like *S. aureus*, these agressive filters are 25% slower. On *Tetrahymena thermophila*-like sequences, they are a factor of 2 slower. (*Tetrahymena* has an even lower G+C content than *S. aureus*.)

4.2 Creation of test sequences

Test sequences, on which we measure the filtering fraction and run time of filters, are generated from a 2nd-order Markov model. This 2nd-order model is learned from the genome with hardest G+C content, e.g., if it is concluded that *Staphylococcus* has the hardest G+C content, a maximum-likelihood 2nd-order model of that genome is learned, and that model is used to generate sequences of arbitrary size. This is necessary, since some test sequences may be larger than the genome.

Test sequences are of various sizes, depending on the filtering fraction of the filter. Sequences of any size can be generated from the 2nd-order model.

4.3 Determining test sequence size

As described previously, the test sequence size must be large enough to give statistically reliable results, e.g., filters with low filtering fractions must be tested on larger sequences. Initially the filter is tested on a short sequence. This short initial test sequence size is given by the parameter `initialTestSeqSize`. The recommended value is 10000, which is dictated by the amount of run time it takes a profile HMM to process that size sequence; this run time must be large enough to get a reasonably accurate reading from the computer's clock.

We then determine if the size was sufficient. Intuitively, if the filtering fraction is the result of a small number of events where nucleotides were above the threshold, then the estimate is unreliable; if it is the result of a relatively large number of events, the estimate is reasonably reliable. Each event where a nucleotide's score is above the threshold results in a window of size *window length* being added to the numerator of the filtering fraction, or less if windows overlap. We estimate the number of events as $\frac{s \times f}{W}$, where s is the sequence size, f is the estimated filtering fraction on this sequence size, and W is the window length.

To determine if the number of events is too low, it is compared to the parameter `minEstAboveThresholdEvents`. The recommended value is 10, which seems to give an accepable level of robustness in the estimates.

If this test indicates that the sequence is too small, the sequence size is increased by a factor of 10 (hardcoded parameter), and the test is repeated with this larger size. Tests are repeated until a size is found to be sufficient, or until the maximum sequence size is reached (see below).

Some filters can be extremely selective, having genuinely very low filtering fractions. For these filters, we would need a very large test sequence, which requires a prohibitive amount of CPU time, even with pre-filters. Moreover, once the filtering fraction is low enough, running the CM immediately after it will not affect overall scan time significantly; therefore, it is not

important to accurately determine the filtering fraction in this low range. To avoid wasting CPU time, we impose a maximal test sequence size, and accept the filtering fraction estimate on this maximum size even if the earlier test indicates that it is not robust. The maximum test sequence size is given by the parameter `maxTestSeqSize`, with recommended value 10,000,000.

4.4 Prefilters

As discussed previously, much time can be spent testing selective & slow filters on large sequences (e.g. 1 or 10 Mbases). We therefore pre-filter these sequences once with less selective filters. For example, suppose a highly selective filter must be tested on a 10 Mbase sequence, and call this filter the *test filter*. The 10 Mbase sequence is generated from the 2nd-order model, as above. Then a pre-filter is selected. This pre-filter will be a filter that has already been tested (so its filtering fraction has already been estimated). The 10 Mbase sequence is then scanned using the pre-filter, and the remainder is stored as a list of intervals within the original sequence. This remainder sequence is then scanned with the test filter, and the filtering fraction relative to the original sequence is used (i.e. the test filter's filtering fraction is computed as if the sequence were never pre-filtered). The estimated run time is the number of CPU seconds taken for the scan, divided by the size of the remainder sequence (i.e. the size of the sequence we actually ran the filter on). If another filter must be tested on the 10 Mbase sequence, the remainder sequence after the pre-filter is re-used. (It is somewhat important to re-use test sequences, not only for efficiency, but also because it makes the filtering fractions and run time statistics more comparable, since there is no statistical noise resulting from variations in the test sequence.)

To further reduce CPU time, pre-filters can be chained. Thus, suppose a 100 Kbase sequence is scanned, and filter 1 is used as a pre-filter. Then, a 1 Mbase sequence must be tested. First this 1 Mbase sequence is pre-filtered with filter 1, and then an additional pre-filter, filter 2, is selected and applied. In some cases, no acceptable pre-filter can be found; in this event, we simply do not use a pre-filter.

There are a number of issues that this approach raises. A key assumption being made is that the pre-filter is reasonably correlated with the test filter, in other words, that if a pre-filter is able to eliminate a subsequence rigorously, then the test filter will also eliminate this subsequence. If this is not the case, then the test filter will obtain an unfair advantage from the pre-filter. In the main, this is not a serious problem in practice since (1) filters are created with related techniques and will all use the same underlying profile HMM, so they are highly correlated in this sense, (2) the pre-filter used will be less selective than the test filter, so it is unlikely to be able to eliminate subsequences that the test filter cannot, and (3) in practice filters will be run in series, and a filter like the pre-filter is likely to be run before the test filter, so the testing scenario is actually a realistic one.

To reduce the effects of the pre-filter interfering with the estimate of the test filter's filtering fraction estimate, we require that the two filtering fractions are not too close. As stated previously, we used already-tested filters as pre-filters; since the candidates for pre-filter have already been tested, we already have an estimate of their filtering fractions. We do not know the test filter's fraction a priori. However, we can compute the maximum filtering fraction of a filter tested on a sequence of a particular size. For example, if we are testing a 1 Mbase sequence, the test filter's fraction cannot be above some threshold fraction, since otherwise it would have been sufficient to have estimated based on the 100 Kbase sequence. By considering in reverse our logic for determining the test sequence size, it is easy to solve

for the maximal filtering fraction.

We require that the pre-filter's filtering fraction is not within some constant factor of the maximal possible filtering fraction of the test filter. If this condition is violated for some candidate pre-filter, we reject it as a pre-filter. The constant factor is given by the parameter `minRatioPrefilterToEstTestFilterFraction`, which has recommended value 4. (This value is simply an educated guess as to the ideal value. It is also a typical difference between adjacent filters in series selected by Dijkstra's algorithm.)

In a similar vein, we can apply pre-filters one after another. The candidate new pre-filter's filtering fraction cannot be within some constant factor of the previously applied pre-filter's fraction. If it is, the candidate is rejected. If no pre-filter has been applied for the test sequence, we imagine that there is a pre-filter with fraction equal to 1. The constant factor is given by the parameter `minRatioPrefilterToPrevPrefilterFraction`, which has recommended value 3 (also an educated guess).

Our use of pre-filters may also distort the estimate of the run time. This is particularly an issue for sub-CM filters. For sub-CMs, recall that the window length is a factor in the run time. When a sequence has been pre-filtered, it may contain many intervals whose lengths are equal to the window length or slightly more. This will result in faster scans, because many cells in the CM/sub-CM Viterbi algorithm's dynamic programming table will be out of the bounds of the short interval.

This effect does indeed distort the run time estimates for sub-CM filters, but this distortion is desirable. A pre-filter will only be used for sub-CM filters that are highly selective (low filtering fraction). Therefore, if the sub-CM filter is selected as part of the series of filters, it will be preceded by some other filter. In this context, it will again run fast as a result of having relatively small intervals to scan. So, it is more realistic to estimate the run time of these sub-CM filters after a pre-filter has been applied.

(Alternately, a more realistic performance model could perhaps be designed. We are using filtering fraction to measure the effect of running a filter, but this is not a good model of sub-CM or CM performance, because CMs' efficiency depends on the length of intervals due to the extra dimension in CMs' dynamic programming table. In place of filtering fraction, we could probably store some kind of two-dimensional filtering fraction, or simply store the distribution of interval lengths. Then, a somewhat more realistic model of sub-CM performance would use this statistic instead of filtering fraction to compute the edge weights in the shortest path graph problem used to select an optimal series of filters. However, our relatively simplistic use of filtering fraction seems to provide good estimates, and therefore the extra work for a more realistic model seems unwarranted.)

Finally, we have noted that the pre-filters considered as candidates are those that have already been tested. Therefore, in order to have good candidate pre-filters on hand, it is important to test the filters in order from least selective (highest filtering fraction) to most selective (lowest fraction). Suppose, to the contrary, that we started with the most selective filters first. These filters would require large test sequences (e.g. 10 Mbases), but we would not have any pre-filters to apply at this point, since no filters were tested. Therefore, we would have to test the filters directly on these large sequences, wasting CPU time. To avoid this problem, we are careful to try to test filters from least to most selective.

We can predict which filters will be less or more selective in both the store-pair and sub-CM filter techniques. Both techniques estimate a filter's selectivity in terms of predicted average reduction in Viterbi score; assuming these predictions are accurate, we can easily order the filters from least to most selective before actually testing their filtering fractions.

For straight profile HMM filters (compact or expanded type), we predict that they are less selective than any sub-CM or store-pair filter.

4.5 Generation of profile HMM filters

The least selective and fastest filters are the profile HMM-based filters. Once the hard G+C content has been determined, the next step is to create compact- and expanded-type profile HMM filters. These profile HMMs are optimized for the 0th-order model given by the hard G+C content. The profile HMM filters' fraction and run time are estimated as described above.

Below we will generate sub-CM and store-pair filters, both of which augment a profile HMM. In all cases, we use the expanded-type profile HMM generated as a candidate filter as a basis for the sub-CM and store-pair filters.

4.6 Are the profile HMM filters sufficient?

Generation of profile HMM filters is significantly faster than the full process of generating a series of filters using the sub-CM and store-pair techniques. Moreover, for many families, profile HMM filters lead to a sufficiently fast scan speed. Therefore, after testing profile HMMs, we consider whether these profile HMMs are good enough.

The expanded-type profile HMM will always filter more selectively than the compact-type profile HMM. So, if the expanded-type HMM filters well, there is no reason for the more sophisticated techniques, and we would not be able to do better with augmented filters anyway.

To estimate whether it is possible to improve on the profile HMM as a filter, we use the following observation. For the 139 non-local families in Rfam 5.0, we estimated the CPU time required by the expanded-type profile HMM and the CM to run on 1 Mbase of sequence. The CM's time divided by the HMM's time divided by the window length of the CM is on average 1.09 (smallest observed = 0.76, largest = 2.34). Thus, if the filtering fraction is significantly less than the recipricol of the window length, the total scan time is dominated by the profile HMM, and improving the filtering fraction will not significantly reduce overall scan time. In this case, therefore, there is no reason to consider more sophisticated techniques. If the filtering fraction is not this low, then more sophisticated techniques may yet improve overall performance.

We therefore introduce a constant parameter `ratioOfWindowLenRecipricolToEpsilonFilteringFraction`, which we abbreviate k. If the estimated filtering fraction of a profile HMM is less than $1/(kW)$, where W is the window length of the CM, then we conclude that the profile HMM is sufficient, and the augmented filters are not considered. In this case, we skip to testing the run time of the CM (see below). We recommend $k = 4$.

4.7 Store-pair filters

We then consider filters using the store-pair technique. Store-pair filters are created as described in the main paper. The paper mentions that we use the logarithm of the infinite-length forward algorithm instead of computing the average Viterbi score. The infinite-length forward algorithm is computed relative to the 0th-order model given by the hard G+C content. (In some cases it may be better to use the average Viterbi score, since (1) with careful selection of

the test sequence size, computing the average Viterbi score may not inflate overall filter creation & selection time too much, and (2) subsequent to submitting the paper, we have found some cases where the average Viterbi score can be more accurate. We have not explored this alternate scheme in depth.)

The paper describes a constant $c = 250$, where, if the HMM has n states, only store-pair HMMs of up to cn states should be considered. The rationale is that eventually the store-pair HMM filters will be slower than the CM, and therefore useless. As described previously, the profile HMMs are faster than the CM by a factor of approximately the CM's window length. Also, recall that the run time of a store-pair HMM is roughly proportional to its number of states. Therefore, as a rule of thumb, if the number of states in a store-pair HMM is greater than the original profile HMM by a factor of the window length, it is too slow. We therefore introduce a parameter `maxTotalStatesMultipleRatioToWindowLen`, which we temporarily abbreviate as k. Let W be the window length. Then we compute the just-described constant c as $c = kW$. A conservatively high value of k would be $k = 1$, but we recommend $k = 0.3$; when the store-pair run time gets this close to the run time of the CM, it's not very useful, and these slower filters are time-consuming to test.

After creating store-pair HMMs using the dynamic programming algorithm described in the paper, we scan through the table looking for filters that give an estimated increase in the logarithm of the infinite-length forward algorithm of at least 0.5. This number is given by the parameter `minStorePairScoreIncreaseFromPrevious`, for which we now recommend the value 0.3. The lower number is mainly motivated by the fact that the estimates of which filters should be considered are only approximate, and so trying more filters makes it more likely that a good filter will be found.

Once the set of candidate store-pair filters is generated, their filtering fraction and run time is estimated on the test sequences.

4.8 Sub-CM filters

The paper described a largely manual process for creating sub-CM filters. We have since fully automated this process, using strategies that resemble those used for the store-pair HMMs. The automated algorithm is as follows.

Hairpins. First, we find the hairpins in the CM by inspecting the structure of the CM itself. Within each hairpin, we find all CM nodes corresponding to base-pair positions, at which a sub-CM could be rooted. (There is little reason to root a sub-CM at an unpaired position, since that position has no secondary structure for the CM to analyze.)

A hairpin is defined as a CM node whose descendants contain no bifurcation nodes. The number of hairpins in a CM is thus equal to the number of bifurcation nodes plus 1. (It would be possible to consider sub-CMs that are rooted on top of a bifurcation node, but this complicates the scanning algorithm because there are multiple end nodes; we have not implemented this solution, so these cases are not considered. We expect that using sub-CMs on multiple hairpins is in practice roughly equivalent to rooting a single sub-CM on top of these hairpins.) Within a hairpin, it is a simple matter to enumerate all base-paired positions.

Simple sub-CMs. A simple sub-CM filter is one that contains exactly one sub-CM rooted at one CM node. The next step is to (1) enumerate these simple sub-CMs, (2) find their optimal window length (an issue discussed in the paper) and (3) estimate their average reduction

in Viterbi score and run time, which will be used to choose good candidate sub-CM filters for further testing.

The algorithm to find the optimal window lengths at various base pair positions is improved slightly from the paper. Before describing this new algorithm, we first describe in more detail the method to find the window length for one CM node. The paper suggests a binary search for values in the range 1 through W. We now refine this idea. First, we need a test to see if a proposed window length is too small (for the binary search). The paper suggests comparing the filtering fraction at some $W' < W$ with the filtering fraction at W. This is not robust, since sometimes the filtering fraction can be 1 in both cases (i.e., when a simple sub-CM is not sufficient to do any filtering, but combining simple sub-CMs might be). We therefore compare average Viterbi score instead.

Let $\overline{V}(x_i, W')$ be the average Viterbi score for a simple sub-CM filter with sub-CM rooted at CM node x_i, using window length W' for the sub-CM. To estimate the average Viterbi score, we use a test sequence (generated from the 2nd-order model) whose length is the parameter `avgScoreEstSeqSize`, with recommended size 10000. This number needs to be large enough that any timing inaccuracy is insignificant, or else we could not estimate the sub-CM filter's run time accurately. However, statistical reliability is not as much of a challenge as with estimating filtering fraction, because each nucleotide makes a contribution to the average Viterbi score, whereas, in the case of highly selective filters, most nucleotides are too easy to filter and make little contribution to a filtering fraction estimate.

We also need to compare average Viterbi scores to see if the window length is too small; if $\overline{V}(x_i, W') = \overline{V}(x_i, W)$ with $W' < W$, then W' is an acceptably large window length, since the filter is producing the same results as with the larger W.

We allow a bit of flexibility in raising the average Viterbi score slightly, by some small constant; sometimes, this allows a significantly smaller window length W', speeding searches, and does not affect the filtering fraction significantly. We use a parameter `acceptableViterbiLossForWindowLenReduction`, with recommended value 0.001. We will abbreviate this parameter below as V^{ε}.

We now discuss the binary search algorithm to find $\widehat{W}'(x_i)$, i.e. the optimal sub-CM window length W' at one specific base pair position x_i. Initially, we assume $L \leq \widehat{W}'(x_i) \leq H$. The initial values of the low and high bounds, L and H, will be specified later; for now, we assume they are given. It is not important to know $\widehat{W}'(x_i)$ to the exact nucleotide. Moreover as its value is higher, the acceptable margin of error is higher. We assume that the acceptable accuracy is proportional to the value, with ratio given by the parameter `acceptableProportionalWindowLenResolution`, for which we recommend 0.2 as a reasonable value. We abbreviate this parameter as r, for resolution, in the following.

Here is the algorithm:

1: Compute $\overline{V}(x_i, W)$ {This is the average Viterbi score with the largest window length we could use.}
2: **while** $H - L > rL$ **do** {Continue until we know the optimal window length to within an acceptably small range.}
3: $W' \leftarrow \frac{L+H}{2}$
4: **if** $\overline{V}(x_i, W') - \overline{V}(x_i, W) \leq V^{\varepsilon}$ **then**
5: $H \leftarrow W'$
6: **else**
7: $L \leftarrow W'$
8: **end if**

9: **end while**
10: $\widehat{W'}(x_i) \leftarrow H$ {The upper bound, H, has definitely passed the test as being just as good as W}

A problem with the method described in the paper is as follows. Suppose we have a hairpin with one long helix, and suppose all positions in this helix require a long window length (e.g., there is an insert state within the hairpin that has a high self-loop probability, so it allows very long sequences with relatively high probability). We will then do a binary search to estimate the window length for the first helix position, then do a separate binary search for the next helix position, and so on. Each step is relatively expensive, especially with a large window length, so this process is wasteful.

Observe the following: if x_1 and x_2 are two base pair nodes in a CM and x_1 is an ancestor of x_2, then the ideal window length of x_1 must be at least as big as the ideal window length of x_2. This follows because sequences generated from x_1 contain a subsequence generated from x_2, so the typical sequence size from x_1 must be at least as big as that of x_2. So, if a given window length was too small for x_2, leading to a poor filter, it must also be too small for x_1.

We can exploit this fact in a kind of double binary search. Let $x_1, \ldots, x_n$ be base pair positions within some helix, with x_i an ancestor of x_{i+1}. Let the window length of the full CM be W, and let the optimal window length of position x_i be $\widehat{W'}(x_i)$. From the above, we know that $\widehat{W'}(x_1) \leq W$, $\widehat{W'}(x_n) \geq 1$ and $\widehat{W'}(x_i) \geq \widehat{W'}(x_{i+1})$, and these inequalities suggest a more efficient algorithm. The algorithm is as follows, and uses the preceding algorithm for $\widehat{W'}(x_i)$ as a subroutine.

1: Compute $\widehat{W'}(x_1)$ with $L, H = 1, W$.
2: Execute W-Recurse$(1, n)$

Procedure W-Recurse(i, j) is now defined. The parameters i and j define the current range in the hairpin, x_i through x_j.

1: **if** $i = j$ **then**
2: Stop — nothing to do, since we've already computed $\widehat{W'}(x_i)$.
3: **end if**
4: $k \leftarrow \frac{i+j}{2}$. {Rounding up if $i + j$ is odd.}
5: Compute $\widehat{W'}(x_k)$ with $L, H = \widehat{W'}(x_i), \widehat{W'}(x_j)$
6: Execute W-Recurse(i, k)
7: Execute W-Recurse(k, j)

Selection of candidate sub-CMs filters. We now consider how to estimate which sub-CM filters make the best filtering fraction vs. run time trade-offs, including compound sub-CM filters, i.e. those containing more than one sub-CM. The algorithm uses a similar strategy to the algorithm to generate candidate store-pair filters.

First, for each simple sub-CM, we compute its average reduction in Viterbi score relative to the profile HMM. Since we have already computed its average Viterbi score, the reduction is computed by simply subtracting the profile HMM's average Viterbi score. Also, while exploring optimal window lengths for the simple sub-CMs, we also measure their increase in run time versus the profile HMM; we use this increase in run time to estimate run times of filters. (In the store-pair technique we were able to assume that the run time is proportional to number of states, but in the case of sub-CMs, we have no such convenient rule of thumb—although, of course, it is probably tractable to create a run time model for the CM Viterbi algorithm, which has a fairly regular structure.)

We will build up all possible compound sub-CM filters one hairpin at a time. Let S be the set of known sub-CM filters.

1: $S \leftarrow \emptyset$ {Initially no sub-CM filters are known}
2: **for** each hairpin H **do**
3: Let S_H be the simple sub-CMs in hairpin H, which were previously enumerated
4: $S \leftarrow S \cup S_H \cup S \times S_H$ {$S \times S_H$ is the cross product of S and S_H}
5: **end for**

If we were to run the preceding algorithm, the size of S is exponential in the number of hairpins. This might be prohibitive for large ncRNAs with many hairpins, so we wish to discard redundant members of S as the algorithm progresses. Suppose two sub-CM filters in S have average Viterbi scores that are very close to each other. In this case, we expect that these filters will also have similar filtering fractions. Therefore, we should discard whichever takes more run time. To quantify whether average Viterbi scores are close to one another, we say that average Viterbi scores are close if they are within some constant given by the parameter `subcmCombiningScoreResolution`, with recommended value 0.05. After adding a sub-CM filter to S, we see if it has a close average Viterbi score to some existing member of S, and if so discard a filter. (We note that for many families, the full exponential number of sub-CM filters is not so prohibitive; these sub-CM filters could probably be stored in memory. This is not the case for store-pair filters, since with store-pair the base and exponent of the exponential set are much larger than for sub-CM.)

While combining sub-CMs for each hairpin, we keep track of the estimated average reduction in Viterbi score, as suggested above. We also keep track of the increase in run time. If compound sub-CM filter s is formed by combining sub-CM filters s_1 and s_2, then the estimated average reduction in Viterbi score of s is the sum of s_1 and s_2. The estimated increase in run time is also the sum of that of s_1 and s_2.

Once we have computed the set S, we run through the filters looking for the next filter that increases the estimate average reduction in Viterbi score by at least some constant. This is exactly the same approach as for the store-pair technique, except the parameter can be different. The parameter is called `minCombinedSubcmScoreIncreaseFromPrevious`, and we use 0.5.

Once this set of candidate sub-CM filters is generated, their filtering fraction and run time is estimated on the test sequences.

4.9 Stopping tests of store-pair or sub-CM filters

As we test either store-pair or sub-CM filters in order from predicted least to most selective, we may find that some filter's fraction is good enough that it is not worthwhile to explore more filters. The logic here is similar to the test of whether the profile HMMs are good enough.

In testing both candidate store-pair and candidate sub-CM filters, we determine if the filtering fraction is below some threshold, and stop testing further filters if it is. The test used is exactly the same as that for profile HMMs, except that we create an additional parameter `ratioOfWindowLenRecipricolToEpsilonFilteringFractionForAugmenteds`, on the theory that we may wish to have a different value.

This early-stopping option is important particularly for families that are only slightly too hard for a pure profile HMM to work. In this case, relatively large test sequences must be used early on due to the filters' lower filtering fractions (since the family is not so hard), and

it is difficult to find good pre-filters if the profile HMMs themselves are not acceptable as pre-filters, so tests can be quite time-consuming. Fortunately, it is acceptable to stop early for these families, so time is saved.

4.10 Estimating CM's run time.

To find the optimal series of filters, it is necessary to estimate the CM's run time. For the same reasons as the sub-CM, the CM's run time should be estimated after some pre-filters, since, in the actual scan, it will be run after some filters.

The CM does not truly have a filtering fraction, so we need to decide what size test sequence to use. As a heuristic, we use the largest test sequence size that has been generated; this largest sequence is bound to have pre-filters already set up.

4.11 Selection of a series of filters.

At this point we have a set of filters (profile HMM, sub-CM and store-pair) each with estimated filtering fraction and run time, and we have estimated the run time of the CM. We therefore apply Dijkstra's algorithm as described in the paper to select a series of filters.

4.12 Caveat on automation

Although the entire process is automated, it is clear that the scheme involves several heuristics for finding a good filtering series while not taking too much CPU time. It is possible that as-yet-unknown ncRNA families may defeat these heuristics, and for example take unnecessarily long to find filters for. Although not necessary, it is therefore helpful for a human to inspect the intermediate results of the algorithms to see if the heuristics may not be working well, particularly if the filter selection process has been taking unusually long.

5 How accurate are the approximations used in filter selection

In the above automated process to select a series of filters, we used a number of approximations (simplifications) to reality, for example: (1) we assumed that running a filter f_1 before running a more selective filter f_2 will not affect the filtering fraction of f_2, (2) we assumed that filtering fraction and run time estimates obtained on one test sequence (generated from a 2nd-order model) are accurate predictions for real genomic data that may contain a wide variety of sequences, and (3) we assumed that the total CPU time used by a filter or by the CM could be estimated as the fraction of the database on which it is run, multiplied by its run time (s/Kb) (although in this case, since the assumption is noticeably violated by sub-CM filters and CMs, we used pre-filters to model the effects).

We have not extensively tested these assumptions, but they appear to be reasonable approximations. We report some findings on 3 ncRNA families derived by a recent report on riboswitch-like motifs[1], for which we performed rigorous scans on the RFAMSEQ for Rfam 6.0 (i.e. EMBL release 78). (A more extensive report on these scans will be published elsewhere; here the focus is simply on how well the filtering predictions matched the actual results.) We do not report the statistics on the Rfam scans reported in the main paper, since those scans did not use the automated process above (and we were sometimes inconsistent in the at-the-time-manual process of deciding an appropriate test sequence size).

		Estimated	**Actual**
gcvT, model 1 (5 filters in series)	Filter 1 fraction	0.433	0.377
	Filter 2 fraction	0.229	0.194
	Filter 3 fraction	0.0824	0.0858
	Filter 4 fraction	0.00565	0.00727
	Filter 5 fraction	0.000314	0.000436
	Overall run time (sec/Kb)	0.204	0.183
gcvT, model 2 (6 filters in series)	Filter 1 fraction	0.621	0.574
	Filter 2 fraction	0.3678	0.430
	Filter 3 fraction	0.0706	0.0913
	Filter 4 fraction	0.00254	0.00540
	Filter 5 fraction	0.000687	0.00139
	Filter 6 fraction	0.00005	0.000171
	Overall run time (sec/Kb)	0.294	0.287
glmS (8 filters in series)	Filter 1 fraction	0.157	0.115
	Filter 2 fraction	0.0390	0.0266
	Filter 3 fraction	0.0183	0.0148
	Filter 4 fraction	0.00774	0.00447
	Filter 5 fraction	0.000858	0.00497
	Filter 6 fraction	0.000196	0.000117
	Filter 7 fraction	0.000015	0.0000125
	Filter 8 fraction	0	0.00000435
	Overall run time (sec/Kb)	0.0310	0.0347

Table 1: Estimated vs. actual statistics on rigorous filter series scans
For these tests, we used 3 ncRNA families derived from the original glmS and gcvT families previously published[1]. Rigorous filter series were created for each family using the automated procedure described in this supplement. Each filter series was then used to scan the RFAMSEQ subset of EMBL release 78. Each of the filters used has an estimated filtering fraction (estimated on a test sequence). This estimated filtering fraction is given in the table, along with the actual filtering fraction measured on the scan of RFAMSEQ. When a series of filters is selected using Dijkstra's algorithm, the total cost of the shortest path is the predicted run time of the series of filters plus the CM, in secs/Kb. This estimate is given, along with the actual amount of CPU time used per nucleotide in the scan of RFAMSEQ. (Note that the least accurate estimates are for filters with low fractions, which may be explained by the fact that we give up on statistical reliability of the filtering fraction when a reliable estimate would require too large a test sequence, e.g. greater than 10 Mbases.)

The results are in Table 1. Qualitatively, the estimates appear generally quite close to the actual values. Moreover, the estimates may actually be better in practical terms than these results suggest; if the estimates are off by a roughly consistent amount, the approximately best filters will likely still be chosen.

References

[1] J. E. Barrick, K. Corbino, W. Winkler, A. Nahvi, M. Mandal, J. Collins, M. Lee, A. Roth, N. Sudarsan, I. Jona, J. Wickiser, and R. R. Breaker. New RNA motifs suggest an expanded scope for riboswitches in bacterial genetic control. *Proc Natl Acad Sci USA*, 101(17):6421–6426, 2004.

[2] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis*. Cambridge U. Press, Cambridge, UK, 1998.

[3] Z. Weinberg and W. L. Ruzzo. Exploiting conserved structure for faster annotation of non-coding RNAs without loss of accuracy. *Bioinformatics*, 20(suppl. 1):i334–i340, 2004.

[4] Z. Weinberg and W. L. Ruzzo. Faster genome annotation of non-coding RNA families without loss of accuracy. In *Proc. Eighth Annual Inter. Conf. on Computational Molecular Biology (RECOMB)*, pages 243–251. ACM Press, 2004.