Unveil: An HMM-based Genefinder for Eukaryotic DNA

William H. Majoros[1]

[1] The Institute for Genomic Research, 9712 Medical Center Drive, Rockville, MD 20850, USA (bmajoros@tigr.org)

Abstract

This paper provides a detailed description of the implementation of Unveil, an *ab initio* gene-prediction program for Eukaryotic DNA. Unveil is implemented as a 283-state Hidden Markov Model (HMM). We describe the structure, training, utilization, and performance of this HMM on a real genomic annotation task. The program is available as open-source software and can be downloaded from the TIGR website, http://www.tigr.org/software. It is hoped that the availability of the source code together with this documentation will enable others to extend this work to produce better genome annotation and analysis tools.

Introduction

Gene Prediction

The problem of gene prediction is one of parsing: given a long DNA sequence, or *substrate*, we wish to identify the parts of that sequence which correspond to the exons and introns of the genes occurring in the substrate, if any. This entails identification of the signals – namely, splice sites and start/stop codons – which delimit those structures. Although 5' and 3' untranslated regions (UTRs) are valuable parts of the gene, they are generally not identified by current genefinders. Instead, the emphasis is still on the accurate identification of those open reading frames (ORFs) which are actually transcribed and translated into proteins – i.e., true coding segments, or CDSs.

An ORF is a sequence of codons (three nucleotides) beginning with a start codon (typically ATG) and ending with a stop codon (usually TAG, TGA, or TAA). In eukaryotes, this codon sequence can be interrupted by *introns*, noncoding regions which are spliced out by a spliceosome molecule after transcription into mRNA and before translation into a protein. An intron begins with a *donor site* (typically GT) and ends with an *acceptor site* (typically AG). Predicting a CDS therefore consists of identifying a start codon, followed by a sequence of zero or more donor and acceptor pairs, followed by an in-frame stop codon.

Unfortunately, many ORFs are not true CDSs, and in eukaryotic genomes there typically are vastly more ORFs than genes. In Arabidopsis, for example, there are on average approximately 350 million ORFs in every 900 base pairs of sequence. Thus, the problem is in determining which of the potentially many ORFs are true coding segments.

A genefinder typically performs this discrimination by applying a set of *signal sensors* and *content sensors* to identify likely components of a CDS and then assembling these components into well-formed gene models. A *signal sensor* is any model which identifies fixed-length signals, such as splice sites and start/stop codons. Such a sensor typically examines the bases surrounding the signal (as well as those comprising the signal itself) and scores them based on their identity and position relative to the signal. A content sensor is a model which scores variable-length subsequences for their conformation to a statistical model of a specific feature, such as an exon or intron. Content sensors for exons typically incorporate some measure of trinucleotide frequency, to account for the codon bias found in real CDSs.

Hidden Markov Models

A Hidden Markov Model (HMM) is a particular class of statistical model for sequences of discrete symbols. The model consists of a finite set of states, each of which can emit a symbol from a finite alphabet with a fixed probability distribution over those symbols, and a set of transitions between states, which allow the model to change state after each symbol is emitted. The transition and emission probabilities may differ between states. The model is conceptualized as starting in a designated *start state*, transitioning

stochastically from state to state for some variable number of time units, and then terminating when a designated *final state* is reached.

In order for an HMM to be employed in the task of gene prediction its transition and emission probabilities must be tuned to reflect the statistical characteristics of the genes of a particular species. This training phase can be performed in the following way: Given a set of training sequences (i.e., confirmed CDSs), we extract the exons and introns, as well as intergenic and untranslated regions surrounding the CDS, and then use these individual features to train the specific sets of states, or *submodels*, within the HMM which emit those particular features. Thus, for example, the states of the HMM which are intended to model the exon portions of genes are trained with real exon example sequences using some training procedure such as Expectation Maximization (EM) or maximum likelihood (both described below). This process will determine the emission probabilities for all of the states as well as the transition probabilities among states within each submodel. Transitions between submodels can then be trained by observing the frequencies of feature alternation within correct CDS parses.

Once an HMM has been trained it can be used to predict gene structures on a given substrate by identifying the most probable path π through the HMM which would emit that substrate; i.e. maximizing $P(x,\pi|M)$ over all paths π for sequence x and model M. This can be efficiently computed using the Viterbi algorithm (described below).

Unveil employs a 283-state HMM, grouped into substates as shown in Figure 1. Separate programs are provided for retraining the HMM for new organisms, and for predicting genes on novel sequences. The remainder of this document describe the specific algorithms and techniques used in those programs.

Design and Implementation

Model Structure

The overall structure of Unveil's HMM is shown in Figure 1. This structure is very similar to that described in [Henderson, *et al*., 1997] for the program VEIL. Shown in the figure are the eight submodels constituting the forward strand, plus the intergenic model, which is strandless.

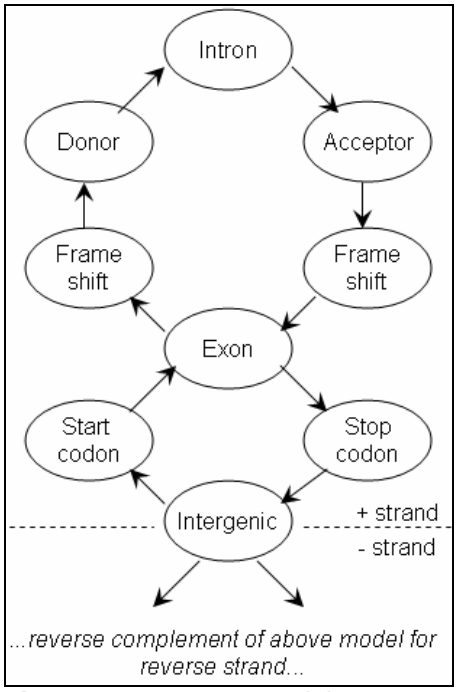

Figure 1. HMM metamodel structure.

The overall structure is referred to as the *metamodel*, and the individual states in the metamodel constitute *submodels*. The metamodel and submodels are loaded dynamically at run-time, and the metamodel is used to direct the merging of all of the submodels into a single composite HMM which can then be subjected to the standard HMM algorithms.

Merging is performed as follows. Let $P_M(x \rightarrow y)$ denote the probability of a transition from state x to state y in submodel M with states S(M), and let $\emptyset_M$ denote the silent start/stop state in M. Then the merged model $M_1 \triangleright M_2$ is defined by combining submodels M_1 and M_2 according to the following:

$$S(M_1 \triangleright M_2) = S(M_1) \cup S(M_2) \cup \{\emptyset_{M1 \triangleright M2}\} - \{\emptyset_{M1}, \emptyset_{M2}\}$$

$$\forall_{a \in S(M1),\, b \in S(M2)}\ P_{M1 \triangleright M2}(a \rightarrow b) = P_{M1}(a \rightarrow \emptyset_{M1}) \cdot P_{M2}(\emptyset_{M2} \rightarrow b) \cdot P_{metamodel}(M_1 \rightarrow M_2)$$

$$\forall_{a,b \in S(M1)-\{\emptyset M1\}}\ P_{M1 \triangleright M2}(a \rightarrow b) = P_{M1}(a \rightarrow b)$$

$$\forall_{a,b \in S(M2)-\{\emptyset M2\}}\ P_{M1 \triangleright M2}(a \rightarrow b) = P_{M2}(a \rightarrow b)$$

$$\forall_{a \in S(M1)}\ P_{M1 \triangleright M2}(\emptyset_{M1 \Rightarrow M2} \rightarrow a) = P_{M1}(\emptyset_{M1} \rightarrow a)$$

$$\forall_{b \in S(M2)}\ P_{M1 \triangleright M2}(b \rightarrow \emptyset_{M1 \Rightarrow M2}) = P_{M2}(b \rightarrow \emptyset_{M2})$$

The composite HMM is formed through a succession of such binary merges, as directed by the metamodel

(i.e., entailing $M_1 \triangleright M_2$ whenever $P_{metamodel}(M_1 \rightarrow M_2) > 0$), until only one submodel remains. The resulting model is then merged with its reverse-complementary model to produce an HMM capable of predicting genes on either strand.

The submodels currently in use are shown in Figures 2–9. Circles depict states and arrows depict all nonzero transition probabilities. States labeled with a nucleotide indicate the *a priori* expected consensus for that state, and are initialized to those consensus bases prior to execution of the training algorithm, though the consensus is not otherwise (i.e., artificially) enforced.

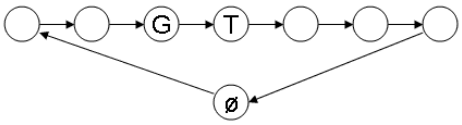

Figure 2. Donor site submodel.

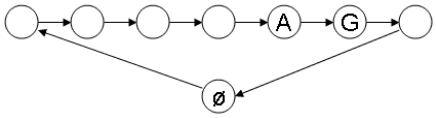

Figure 3. Acceptor site submodel.

The donor and acceptor site submodels work effectively as a positional weight matrix by scoring bases immediately left and right of the consensus positions.

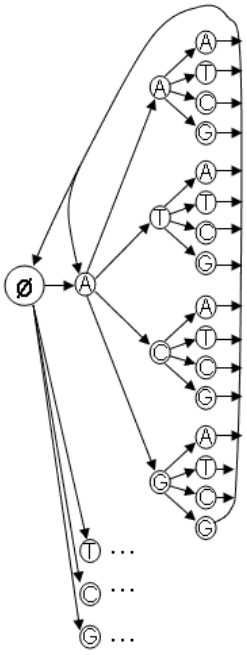

Figure 4. Exon submodel.

The exon submodel, copied directly from [Henderson, *et al*., 1997], explicitly represents all 64 trinucleotide combinations, effectively performing as a measure of codon bias. Note that this model could be significantly reduced in size by eliminating the third (rightmost) tier and replacing it with a simpler version in which every group of four {A,T,C,G} states is instead represented as a single state emitting all four bases. Such a variant submodel would perform identically to the original, since the emission probabilities of the new states would simply reflect the transition probabilities of their predecessors in the middle tier. This modification cannot be generalized to all three tiers without changing the performance of the model, however.

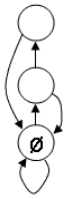

Figure 5. Frameshift submodel.

The frameshift submodel occurs both before and after the exon submodel in order to accommodate out-of phase introns. When such an intron occurs, one or two bases at the beginning or end of the exon will by matched by the frameshift submodel instead of the main exon submodel.

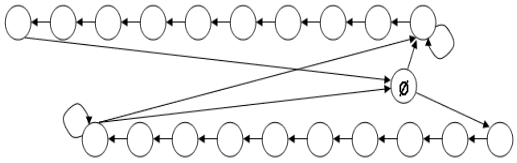

Figure 6. Intergenic submodel.

The intergenic submodel consists of a 5’ UTR submodel and a 3’ UTR submodel, each a linear sequence of nine states, joined in the middle with a pair of states with self transitions allowing arbitrarily large intergenic regions. During training we assume that the ten bases immediately upstream of the start codon or immediately downstream of the stop codon are UTR, although this will in many cases not be the case. This simple heuristic avoids the problem of identifying high-confidence UTR sequence for training while presumably capturing some of the signal present in such UTRs when they do occur (as well as noise where they do not occur). Note that the 5’ UTR and 3’ UTR components of this submodel are trained separately.

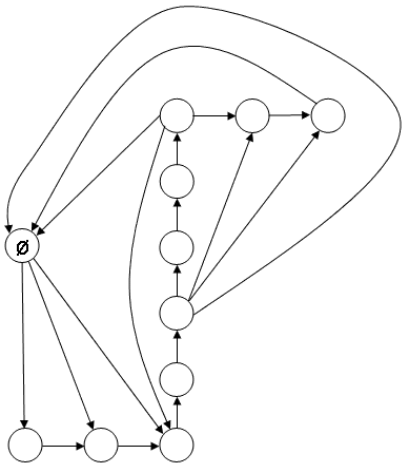

Figure 7. Intron submodel.

The intron submodel consists of a linear sequence of six states, to reflect hexanucleotide frequencies, flanked on either end by a built-in frameshift mechanism to allow for the proper concordance of the central six-state path with any hexanucleotide signal that may be present.

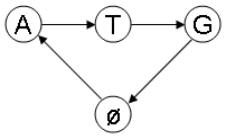

Figure 8. Start codon submodel.

The start codon and stop codon submodels encode the common consensus sequences for those signals while still allowing noncanonical consensus signals to be encountered and appropriately incorporated during training.

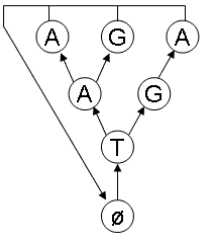

Figure 9. Stop codon submodel.

Training Submodels

Unveil provides two methods for the training of its submodels. The first method, which utilizes a simple maximum likelihood approach, is applicable only to those submodels for which a single, unambiguous path exists through the submodel for every sequence emitted by that submodel. In this case, it is a simple matter to align the training sequences with the path's state sequence and compute the relative transition and emission frequencies for each state pair and each state × symbol pair, respectively. In the model currently used by Unveil, this training procedure is applicable only to exon, splice junction, and start/stop codon submodels.

For the remaining features, the Baum-Welch algorithm is used to perform Expectation Maximization (EM) training, which is described at length in the following sections. The treatment follows [Durbin, *et al*., 1998], but differs on some crucial points, due to problems with numerical underflow which were encountered during development of Unveil. Durbin, *et al*. prescribe a scaling approach to avoid such underflow problems, after [Rabiner, 1989], but two problems were found with that formulation, namely (1) that the formulas given for scaling in the forward algorithm were not entirely adequate to avoid underflow, and (2) that the suggestion, originally made by Rabiner, that identical scaling factors could be used in the forward and backward algorithms was found empirically to be false for a set of *Drosophila melanogaster* genes used to train Unveil. Note that Rabiner's original paper addressed the use of HMMs for natural language processing, not sequence analysis, which likely explains the discrepancy. We use notation similar to that of [Durbin, *et al*., 1998].

1. Notation and Overview of Computation

a. Basic Notation

M=(Σ,Q,e,a) represents a hidden Markov model, where

Q is the set of *states*,

Σ={A,T,C,G,N} is the *alphabet* of symbols which are emitted by the states of the model,

$e_{k,s}$ = P[M emits symbol s | M is in state k] gives emission probabilities, and

$a_{i,j}$ = P[M transitions to state j | M is in state i] gives transition probabilities.

Note that state 0 is the silent start/stop state Ø referred to earlier. In the context of processing a given sequence, L will be used to denote the length of that sequence, and x(i) will denote the i^{th} symbol in the sequence, where i=1 corresponds to the very first

symbol. |S| is used to denote set cardinality, and ln(x) denotes the natural logarithm (i.e., $\log_e$).

b. Overview of the EM algorithm

The Baum-Welch algorithm is an iterative procedure which consists of repeated applications of the *forward* and *backward algorithms* to produce progressively better estimates of the transition and emission probabilities of the HMM. It can be formally shown that during the operation of this procedure, the likelihood of the training data given the current parameterization of the model increases monotonically, so that one can get arbitrarily close to a local maximum for this likelihood by simply extending the computation out indefinitely. Proof of this assertion is given in [Durbin, *et al*., 1998, p324-325].

The forward algorithm is embodied in the computation of a multidimensional variable, $f_{k,i}$. Formally, $f_{k,i}$ denotes the probability P[a model M in the initial state will emit the sequence $x_1..x_i$ and reside in state k when emitting x_i at time i]. Similarly, the backward algorithm is embodied in $b_{k,i}$ = P[M will next emit the string $x_{i+1}..x_L$ and then terminate | M is currently in state k]. The forward and backward variables are computed using a dynamic programming approach, as described in [Durbin, *et al*., 1998, p58-59].

The modification of the forward and backward algorithms to incorporate scaling will make use of two additional variables, $s_{f,i}$ and $s_{b,i}$ (respectively). To distinguish between the scaled and unscaled versions of the forward and backward variables, we will denote by $\tilde{f}_{k,i}$ the scaled version of $f_{k,i}$, and by $\tilde{b}_{k,i}$ the scaled version of $b_{k,i}$.

2. The forward algorithm

As a scaling factor for the forward algorithm we use

$$s_{f,i} = \sum_{\ell=1}^{|Q|-1} \left(e_{\ell,x(i)} \sum_{k=0}^{|Q|-1} \tilde{f}_{k,i-1} a_{k,\ell} \right) \tag{2.1}$$

as suggested by [Durbin, *et al*., 1998, p78]. Because this formula makes use of $\tilde{f}_{k,i-1}$, computation of $s_{f,i}$ and $\tilde{f}_{k,i}$ occur in tandem.

Initialization of the forward variable is unaffected by scaling:

$$\forall_{1 \le k < |Q|} \tilde{f}_{k,0} = 0 \tag{2.2}$$

$$\forall_{1 \le i \le L} \tilde{f}_{0,i} = 0 \tag{2.3}$$

$$\tilde{f}_{0,0} = 1 \tag{2.4}$$

The recursion portion of the algorithm now incorporates the scaling term in the denominator:

$$\forall_{1 \le i \le L} \forall_{1 \le \ell < |Q|} \tilde{f}_{\ell,i} = \frac{e_{\ell,x(i)}}{s_{f,i}} \sum_{k=0}^{|Q|-1} \tilde{f}_{k,i-1} a_{k,\ell} \tag{2.5}$$

again, as given in [Durbin, *et al*., 1998, p78]. Finally, the scaled probability $\tilde{P}_j$ of a training sequence j is given by:

$$\tilde{P}_j = \sum_{k=1}^{|Q|-1} \tilde{f}_{k,L} a_{k,0} \tag{2.6}$$

3. The backward algorithm

A scaling factor for the backward algorithm can be computed as follows:

$$s_{b,i} = \sum_{k=0}^{|Q|-1} \sum_{\ell=1}^{|Q|-1} a_{k,\ell} e_{\ell,x(i)} \tilde{b}_{\ell,i} \tag{3.1}$$

Note again that while [Durbin, et al., 1998, p78] state that $s_{a,i}$ *must* be used as the scaling factor for $\tilde{b}_{k,i}$ and [Rabiner, 1989] states that it *may* be used for such, we have found that in practice doing so fails to protect $\tilde{b}_{k,i}$ from underflow, and so we use (3.1) instead.

Similarly to (2.5), this scaling factor can be incorporated into the denominator of the recursion for $\tilde{b}_{k,i}$ as follows:

$$\forall_{0\le i<L}\ \forall_{0\le k<|Q|}\ \tilde{b}_{k,i} = \frac{1}{s_{b,i+1}} \sum_{\ell=1}^{|Q|-1} a_{k,\ell} e_{\ell,x(i+1)} \tilde{b}_{\ell,i+1} \qquad (3.2)$$

As in the forward algorithm, no scaling is necessary for the initialization step of the backward algorithm:

$$\forall_{0\le k<|Q|}\ \tilde{b}_{k,L} = a_{k,0} \qquad (3.3)$$

4. The Baum-Welch algorithm

a. The scaling ratio

Scaling the forward and backward algorithms as described above invalidates the invariants which ensure convergence of the Baum-Welch algorithm. For this reason, the scaling factors $s_{f,i}$ and $s_{b,i}$ must be explicitly cancelled out when $\tilde{f}_{k,i}$ and $\tilde{b}_{k,i}$ are later combined to re-estimate the model parameters. Failure to do this properly will typically result in failure to converge, giving rise to poorly trained models, and can even result in underflow or overflow errors. The following scaling ratio provides most of the necessary cancellation:

$$\forall_{1\le i\le L}\ r_i = \ln\left(\frac{1}{s_{f,L}} \prod_{h=i}^{L-1} \frac{s_{b,h+1}}{s_{f,h}} \right) \qquad (4.1)$$

The remaining terms that are not cancelled by (4.1) are handled individually below.

b. Updating the transition counts

Before the transition probabilities can be re-estimated, the expected number of uses of each transition during a hypothetical generation of the training sequences by the current model must be tabulated. Initialization steps for this procedure are given by (4.2) and (4.3):

$$A_{0,0} = 0 \qquad (4.2)$$

$$\forall_{1\le k<|Q|}\ A_{k,0} = \sum_{\text{strings j}} \frac{e^{r_L} s_{f,L} \tilde{f}_{k,L} a_{k,0}}{\tilde{P}_j} \qquad (4.3)$$

Equation (4.3) incorporates the scaling ratio r_L and an additional term to be cancelled, $s_{f,L}$. Cancellation in the recursion step is achieved fully by the scaling ratio r_{i+1}:

$$\forall_{0\le k<|Q|}\ \forall_{1\le \ell<|Q|}\ A_{k,\ell} = \sum_{\text{strings j}} \sum_{i=0}^{L-1} \frac{e^{r_{i+1}} \tilde{f}_{k,i} a_{k,\ell} e_{\ell,x(i+1)} \tilde{b}_{\ell,i+1}}{\tilde{P}_j} \qquad (4.4)$$

c. Updating the emission counts

As with the expected transition counts, the expected emission counts must be modified to cancel the extraneous scaling terms:

$$\forall_{1\le k<|Q|}\ \forall_{s\in\Sigma}\ E_{k,s} = \sum_{\text{strings j}} \sum_{\substack{1\le i\le L,\\ x(i)=s}} \frac{e^{r_i} \tilde{f}_{k,i} \tilde{b}_{k,i} s_{f,i}}{\tilde{P}_j} \qquad (4.5)$$

d. Updating the transition and emission probabilities

Given the foregoing modifications, re-estimation of the transition and emission probabilities may proceed exactly as given in [Durbin, *et al*., 1998]:

$$\forall_{0\le k<|Q|}\ \forall_{0\le \ell<|Q|}\ a_{k,\ell} = \frac{A_{k,\ell}}{\sum_{\ell'=0}^{|Q|-1} A_{k,\ell'}} \qquad (4.6)$$

$$\forall_{1\le k<|Q|}\ \forall_{s\in\Sigma}\ e_{k,s} = \frac{E_{k,s}}{\sum_{s'\in\Sigma} E_{k,s'}} \qquad (4.7)$$

e. Computing the log-likelihood

Although the log-likelihood of the model need not be explicitly computed in order to perform training, it is useful to do so during development in order to verify that an implementation is correct. Except for extremely small fluctuations due to rounding errors in floating point numbers, the log-likelihood of the current model should increase monotonically throughout the training process. Any but the smallest of decreases in this value are indications of an error in the implementation.

The log-likelihood may be computed as:

$$\lambda = \sum_{\text{strings j}} \ln(\tilde{P}_j) + \sum_{i=1}^{L} \ln(s_{f,i}) \qquad (4.8)$$

5. Verification

Although scaling is necessary during execution of the forward and backward algorithms in order to avoid numerical underflow, all scaling factors applied during those computations must be subsequently cancelled before the updating of the expected transition and emission counts. Failure to do this will invalidate the proof, given in [Durbin, *et al*., 1998, p323-325] that the likelihood of the model increases monotonically – that is, it will fail to ensure continual progress during the learning process. This was remedied above by incorporating additional terms into equations (4.2) through (4.5).

That these modifications do achieve the cancellation without otherwise modifying the basic mathematical structure can be seen by applying simple algebra. Expressing all scaled quantities as closed-form (i.e., not recurrence) expressions and then substituting these back into the equations from section 4 will show that all scaling terms cancel, leaving the original, unmodified equations from [Durbin, *et al*., 1998].

Equations (5.1)-(5.3) give the closed-form equations. The algebra is left as an exercise for the enthusiastic reader.

$$\tilde{P}_x = \frac{1}{\prod_{h=1}^{L} s_{f,h}} \sum_{k=1}^{|Q|-1} f_{k,L} a_{k,0} \qquad (5.1)$$

$$\forall_{1 \le \ell < |Q|} \forall_{1 \le i \le L} \tilde{f}_{\ell,i} = \frac{f_{\ell,i}}{\prod_{h=1}^{i} s_{f,h}} \qquad (5.2)$$

$$\forall_{0 \le k < |Q|} \forall_{0 \le i < L} \tilde{b}_{k,i} = \frac{b_{k,i}}{\prod_{h=i}^{L-1} s_{b,h+1}} \qquad (5.3)$$

Quantifying the Adaptation Process

To see that the scaled implementation of the Baum-Welch algorithm in Unveil behaves as intended it is instructive to print out the log-likelihood of the model during training and verify that it increases monotonically.

Figure 10 shows the result of performing such an exercise. The training procedure was performed on a training set of 100 sequences, each 5000 nucleotides long. The log-likelihood is shown on the Y-axis and iterations on the X-axis. As the adaptation proceeds, the log-likelihood of the model improves according to an S-curve, rising slowly at first, then making rapid gains, and then finally leveling off as a local optimum is approached.

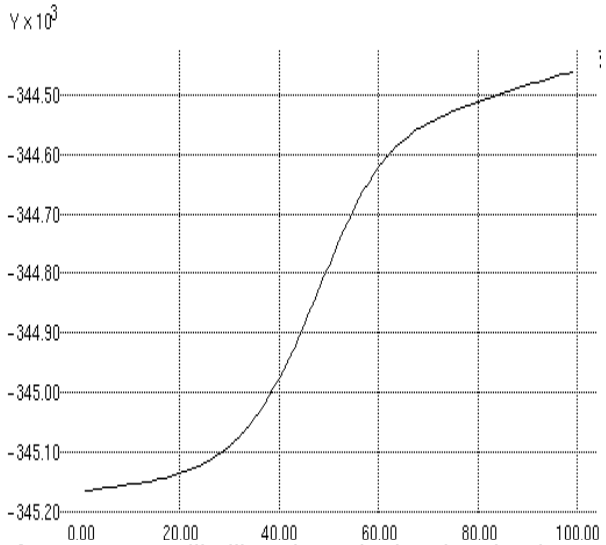

Figure 10. Log-likelihood graph showing that log-likelihood (Y) monotonically increases as EM iterations (X) proceed. The training set consisted of 100 sequences, each 5000 nucleotides long.

Viterbi Decoding

In the Baum-Welch training program for Unveil we used a multiplicative scaling approach, because logarithmic scaling is not easily performed in the forward and backward algorithms due to the addition (vs. multiplication) of probabilities. However, in the Viterbi algorithm, which is used to find the most likely path through the HMM for a given sequence (thereby choosing a gene model), probabilities are only multiplied, making the log transformation simple. The dynamic programming implementation in Unveil follows [Durbin, *et al*., 1998, p78] exactly.

Performance Evaluation

To see that Unveil performs gene prediction with accuracy rivaling other genefinders, a comparison on 300 full-length cDNAs from *Arabidopsis thaliana* was performed. Table 1 shows the results.

program	nucleotide accuracy	exon specificity	exon sensitivity	percentage of exact genes
Unveil	94%	75%	74%	46%
Exonomy	95%	63%	61%	42%
GlimmerM	93%	71%	71%	44%
Genscan	94%	80%	75%	27%

Table 1. Gene prediction accuracy on a set of 300 cDNAs from *A. thaliana*.

As can be seen from the table, Unveil scores very highly in terms of nucleotide accuracy, exon accuracy, and whole-gene accuracy.

Acknowledgements

The development of Unveil was supported by the National Science Foundation under grant MCB-0114792.

References and Suggested Reading

[Durbin, *et al*., 1998] Durbin, R., Eddy, S., Krogh, A. & Mitchison, G. *Biological Sequence Analysis*. Cambridge University Press. (1998).

[Henderson, *et al*., 1997] Henderson, J., Salzberg, S. & Fasman, K. Finding Genes in Human DNA with a Hidden Markov Model. *Journal of Computational Biology* 4:127-141 (1997).

[Rabiner, 1989] Rabiner, L.R. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257-286 (1989).

[Rabiner and Juang, 1986] Rabiner, L.R. & Juang, B.-H. An introduction to hidden Markov models. *IEEE Transactions on Acoustics Speech, Signal Processing*, 3(1):4-16 (1986).