Computational prediction of RNA editing sites

R. Bundschuh

Department of Physics, The Ohio State University, 174 West 18th Avenue, Columbus, OH 43210-1106, USA

Received on May 8, 2003; Revised and Accepted on June 25, 2004
Advance Access publication July 1, 2004

ABSTRACT

Motivation: Some organisms edit their messenger RNA resulting in differences between the genomic sequence for a gene and the corresponding messenger RNA sequence. This difference complicates experimental and computational attempts to find and study genes in organisms with RNA editing even if the full genomic sequence is known. Nevertheless, knowledge of these editing sites is crucial for understanding the editing machinery of these organisms.

Results: We present a computational technique that predicts the position of editing sites in the genomic sequence. It uses a statistical approach drawing on the protein sequences of related genes and general features of editing sites of the organism. We apply the method to the mitochondrion of the slime mold Physarum polycephalum. It correctly predicts over 90% of the amino acids and over 70% of the editing sites.

Availability: The source code is available upon request from the author.

Contact: bundschuh@mps.ohio-state.edu

INTRODUCTION

The central dogma of molecular biology states that the genetic information on the DNA is transcribed into messenger RNA (mRNA) which in turn is translated into proteins (Alberts et al., 1994). For most organisms the mRNA has the same sequence as the DNA (possibly after eliminating introns). However, there are organisms that edit their mRNA sequences, i.e., a dedicated biological machinery in these organisms, substitutes, inserts, or deletes single or dinucleotides of the mRNA (Smith et al., 1997; Brennicke et al., 1999; Gott, 2003; Gray, 2003). Although the editing machinery for some of these organisms is known, it remains a complete mystery in other organisms.

In order to understand the RNA editing machinery, it is desirable to identify as many examples of sites at which the mRNA is edited as possible. However, the very existence of RNA editing complicates the application of traditional techniques to such organisms. To amplify and sequence a given mRNA exactly complementary primers are required. Owing to RNA editing, knowledge of the genomic sequence of an organism does not imply knowledge of the mRNA sequence. Thus, such primers can only be constructed by laborious trial-and-error. Also, the computational identification of genes within the genome is hindered by RNA editing since the unedited sequence in the genome lacks the features, computational approaches to gene finding look for, as explained in more detail below.

As an example for an organism that performs RNA editing by a mechanism that is not yet understood, we will concentrate on the mitochondrion of the slime mold Physarum polycephalum. The complete genomic sequence of this mitochondrion is known (Takano et al., 2001). Experimentally, the editing sites of a handful of protein-coding genes have been determined (Mahendran et al., 1991; Gott et al., 1993; Miller et al., 1993; Wang et al., 1999; Takano et al., 2001). They show that the insertion of a single C is by far the most prevalent editing operation. Less often, other single nucleotides or dinucleotides are inserted. Computationally, some open reading frames have been found on the genome in spite of them being interspersed with RNA editing sites (Takano et al., 2001). However, the current state of annotation of this mitochondrial genome shows the complications insertional RNA editing introduces into gene finding. The length of the mitochondrial genome of Physarum is comparable to the length of the mitochondrial genome of the well-studied slime mold Dictostelium discoideum. However, while a good 40 genes are annotated on the genome of the latter, only 10 genes are known in Physarum. This means that as currently annotated long stretches of the genome do not contain any genes—a very unlikely scenario given that mitochondrial genomes are known to be under a strong evolutionary pressure to remain short. Also, some genes that are typically to be expected in a mitochondrial genome could not be found (Takano et al., 2001) using the standard tool blastx (Gish and States, 1993).

One may ask why it is so difficult to find and annotate genes in the Physarum mitochondrial genome. The simplest way to find genes in a genome is to look for long stretches without stop codons in a single frame. Since in the presence of insertional RNA editing the coding sequence constantly changes frames, there are plenty of stop codons in all three frames even within a coding sequence. They are simply not in the frame that is locally used. Other de novo gene finding methods in one
way or another search for a triplet pattern in coding regions of the genomic sequence. Again, since the frame changes on an average every 25 bases, such a triplet pattern does not exist in a *Physarum* gene. The other gene finding techniques rely on sequence similarity. A translation of the genome in all six frames (three forward and three backward) using the genetic code and a comparison of the translated sequences on the protein level as implemented, e.g. in blastx fails since each of these translations on an average contains 25/3 ∼ 8 correct amino acids followed by 16 amino acids that are not actually part of the protein. On the other hand, the sequences have diverged too far to allow for a comparison on the DNA level using a tool like blastn (Altschul et al., 1997) where the usual capabilities of sequence alignment algorithms to deal with insertions and deletions could handle the editing sites. Only for the cox1 gene DNA sequences from four very closely related species are known (Horton and Landweber, 2000). For this gene, most editing sites can indeed be inferred from the pairwise alignments on the DNA level. However, of the six genes with known editing sites only 33% are covered by sequence similarity hits on the DNA level and even these hits have large numbers of (long) gaps. Thus, at most one-third of the individual editing sites correctly. Predictions of the edited sites. Potential genes that have not yet been identified. This will greatly enlarge the number of known editing sites and provide a basis for a better understanding of the editing machinery. While the specific occurrence finds the editing sites in genes for which they can be used to find the editing sites in genes for which they have so far not been determined. This will greatly enlarge the number of known editing sites and provide a basis for a better understanding of the editing machinery. While the specific implementation of our method is optimized for *Physarum*, the general technique is applicable to any organism with RNA editing.

**METHODS**

Our technique relies on the hidden markov model (HMM) approach to sequence alignment (Thorne et al., 1991; Hughey and Krogh, 1996; Holmes and Durbin, 1998). Sequence alignment of a protein sequence to a query protein sequence can be formulated as the Viterbi algorithm applied to an HMM that generates protein sequences. We will start from such an HMM and turn it into an HMM that generates nucleotide sequences corresponding to the protein query. Then, we will modify the resulting HMM to allow RNA editing operations. Applying the Viterbi algorithm to the final HMM yields the optimal interpretation of the genomic sequence as coding for the query protein with RNA editing and thus specifically the optimal positions for the RNA editing sites.

The basis for comparing two protein sequences $a_1, \ldots, a_N$ and $b_1, \ldots, b_M$ is a scoring matrix that assigns a local similarity score $s_{a,b}$ to each pair $(a, b)$ of aligned amino acids. According to the Karlin–Altschul theory (Karlin and Altschul, 1990) these scoring matrices are given by the log odds scores

$$s_{a,b} = \lambda^{-1} \log \frac{p_{a,b}}{p_a p_b}$$

where the $p_{a,b}$ are the probabilities to find amino acid $a$ in the sequences and the $p_a, p_b$ are the probabilities to see a pair $(a, b)$ in a set of reference alignments. $\lambda$ is an arbitrary scaling parameter chosen in order to obtain convenient numbers in the scoring matrix.

For sequence alignment with gaps, the gap initiation cost $\delta$ and the gap extension cost $\epsilon$ have to be specified in addition to the scoring matrix. They are used to assign the affined gap cost $\delta + \epsilon n$ for each gap of length $n$ in an alignment of the two sequences. The local alignment score $\Sigma$ quantifying the overall similarity of the two sequences is then calculated by the Smith–Waterman algorithm (Smith and Waterman, 1981; Waterman, 1994).

$$S_{i,j}^S = \max \{ S_{i-1,j-1}^S + S_{i-1,j-1}^D, S_{i-1,j-1}^D + 1, 0 \} + s_{a,b}$$

$$S_{i,j}^D = \max \{ S_{i-1,j}^S - (\delta + \epsilon), S_{i-1,j}^D - (\delta + \epsilon) \}$$

$$S_{i,j}^I = \max \{ S_{i-1,j-1}^S - \epsilon, S_{i-1,j-1}^I - \epsilon, S_{i-1,j-1}^D - (\delta + \epsilon) \}$$

$$\Sigma = \max_{1 \leq i \leq N, 1 \leq j \leq M} \{ S_{i,j}^S, S_{i,j}^I, S_{i,j}^D \}$$

in $O(MN)$ time using three auxiliary arrays $S_{i,j}^S, S_{i,j}^I$ and $S_{i,j}^D$. At this point, we shall break the symmetry between the two sequences to be compared and fix $a_1, \ldots, a_N$ as the query sequence while treating $b_1, \ldots, b_M$ as the variable subject sequence. In order to reflect this in our notation, we replace the local similarity score $s_{a,b}$ by $s_1(b_j)$ and the auxiliary variables $S_{i,j}^S, S_{i,j}^I$ and $S_{i,j}^D$ by $S_{j,i}^S, S_{j,i}^I$ and $S_{j,i}^D$, respectively. In this formulation, the $s_1(b_j)$ do not even have to be determined by a query sequence $a_1, \ldots, a_N$ and a scoring matrix any more but can be any position-specific scoring matrix that assigns to each index $i$ a set of 20 scores, one for each amino acid.

This form of the Smith–Waterman algorithm is formally a special case of the Viterbi algorithm applied to the HMM shown in Figure 1 except for the zero in Equation (2).
In the following, we shall take this into account by drawing HMM diagrams but assigning to the transitions in these diagrams instead of the ‘probabilities’ $p^{X,Y}$ immediately their logarithms $s^{X,Y} = \log p^{X,Y}$, i.e. the scores.

The zero in the Smith–Waterman algorithm ensures that also sub-HMMs using any of the nodes $S_j$ as their starting nodes are considered in the maximization of the probability. This allows local matches to be found while the algorithm without the zero also known as the Needleman–Wunsch algorithm (Needleman and Wunsch, 1970) only finds global matches between the model and the subject sequence. In order to reintroduce this zero into our algorithm, we shall specify for each node of the model if it is a possible starting node or not. Then, the general algorithm to find the best match of a subject sequence $b_1, \ldots, b_M$ to a model reads

$$S_j^X = \max_{Y \rightarrow X} \left\{ s_j^{Y \rightarrow X} \right\} + s_Y(b_j)$$

for the model depicted in Figure 1b this is exactly the Smith–Waterman algorithm Equations (2)–(5). A standard backtracking procedure can reconstruct the sequence of nodes passed by the optimal parse of the model leading to the score $\Sigma$. In the following we shall modify the HMM to take into account RNA editing. However, we will not write down recursion equations any more but simply draw the new HMMs with the understanding that we apply the algorithm described in Equations (7) and (8) to these new HMMs.

**ALGORITHM**

RNA editing changes the sequence on the nucleotide level. Thus, we have to reformulate the HMM shown in Figure 1 in terms of an HMM that emits nucleotides instead of protein sequences. To this end, we shall replace the different node types of the protein HMM by more complex assemblies of nodes. The resulting HMM will have only two types of nodes: silent nodes that do not emit any letter denoted by diamonds in our figures and nodes that emit one of the four letters A, T, G or C, which we denote by circles with the understanding that we apply the algorithm described in Equations (7) and (8) to these new HMMs.

This HMM has the states $I_0, I_1, \ldots, I_M, D_1, \ldots, D_M$ and $S_1, \ldots, S_M$ and the start state $O$. The states $I_i$ as well as the states $S_j$ emit a single amino acid $b$ with probabilities $p^b_{I_i}$ and $p^b_{S_j}$, respectively, while the states $D_j$ and $O$ are silent. The transitions between the different states are given by transition probabilities $p^{S,D}, p^{S,I}, p^{I,D}$, $p^{S,I}$, $p^{D,I}$, $p^{D,D}$, $p^{S,S}$, $p^{D,D}$ and $p^{D,D}$ as indicated in the figure.

The Viterbi algorithm calculates the highest probability $P^X_j$ of all ways that the HMM could have emitted the sequence $b_1, \ldots, b_j$ ending in node $X$ with $i$ such that $b_i$ is emitted by node $S_1$ or $I_1$ with the help of the recursion formula

$$P^X_j = \max_{Y \rightarrow X} \left\{ p^Y_{j-1} p^{Y,X} \right\} p^X(b_j) \quad \text{X emits} \quad P^X_j$$

$$\max_{Y \rightarrow X} \left\{ p^Y_{j-1} p^{Y,X} \right\} \quad \text{X silent}. \quad (6)$$

It becomes equivalent to the Smith–Waterman algorithm if we identify $P^X_j = e^{\lambda ST}$, $p^{S,T} = p^{S,D} = p^{D,D} = p^{D} = e^{-\lambda (d+1)}$, $p^{I,T} = p^{D,D} = e^{-\lambda}, p^{I}(b) = 1, p^{S}(b) = e^{\lambda b}$ and $p^{S,S} = p^{I,S} = p^{S,D} = p^{D} = p^{D} = 1$. We note that this equivalence is only formal, since the ‘probabilities’ assigned above do not add up to one at each node\(^1\).

\(^1\) See Yu and Hwa (2001) for a full and more detailed discussion.
Computational prediction of RNA editing sites

Fig. 2. Replacement operations that transform the HMM in Figure 1 emitting amino acid sequences into an HMM emitting nucleotide sequences. The diamonds represent silent nodes and the circles with a nucleotide letter represent nodes that emit only this letter. (a) Shows the replacement for a substitution node and (b) Shows the replacement for an insertion node of the original HMM.

A block of nodes containing 64 tracks. Each of these tracks emits the three letters of a codon. The emission scores of the original models are absorbed into the 64 transition scores \( r_{xyz} \) and \( t_{xyz} \) for the substitution and insertion nodes, respectively. Thus, we have to assign transition scores equal to the emission score of the amino acid each codon corresponds to, i.e.

\[
\begin{align*}
    r_{xyz} &= \begin{cases}
    s_i(b) & \text{xyz codes for } b \\
    -\infty & \text{xyz stop codon}.
    \end{cases} \\
    t_{xyz} &= \begin{cases}
    0 & \text{xyz codes for } b \\
    -\infty & \text{xyz stop codon}.
    \end{cases}
\end{align*}
\]

Applying the algorithm defined by Equations (7) and (8) to this HMM is equivalent to translating the nucleotide sequence in all three frames, running the protein HMM from Figure 1 against all three translations, and at the end choosing the best match among the three translations.

Once we have set up an HMM emitting nucleotides, it is a relatively easy task to include RNA editing in this model. Here, we will only include insertions of single Cs since this is the most frequent editing operation in \( P. \) polycephalum. However, the technique presented here can easily be modified to include other insertion events in order to predict RNA editing sites in a different organism with a different prevalent editing operation. One could also include nucleotide deletion events if such events were to be expected in a given organism (note that although the algorithm presented here includes deletion nodes, these deletion nodes correspond to amino acid deletions and thus not to editing events that are deletions of individual bases). It is even possible to include the possibility for more than one editing operation at the same time in order to not only model the most prevalent but also more rarely occurring editing events.

In choosing which editing events to include in the model a subtle balance has to be kept: the downside of restricting ourselves to the most prevalent editing event, e.g. C insertion, is that the algorithm will interpret other kinds of insertions also as C insertions. Thus, it produces wrong predictions in the neighborhood of insertion sites that are not of the most prevalent type. On the other hand, the algorithm becomes more specific if a smaller number of editing events are allowed since there are less possible interpretations of the same sequence. This improves the quality of the predictions.

In the case of \( P. \) polycephalum studied here we tried to include insertions of single Us as the second most prevalent insertion event. However, it turned out that the predictions of additional spurious U insertions overwhelmed the few real U insertions that the algorithm that does not allow for U insertions misses. Since the overall quality of the predictions went down by including U insertions, below we present the algorithm that only allows C insertions. However, although this algorithm does not take into account insertions of other nucleotides or dinucleotides, it is not confused by such other insertions: such other insertion sites are merely emulated by the algorithm by C insertions such that the reading frame remains consistent with the protein model supplied. In the case of dinucleotides such an event can be easily spotted by visual inspection of the prediction since it leads to two predicted C insertions closer together than the biologically found minimal distance of nine nucleotides between editing sites.

In the nucleotide HMM the insertion of a C corresponds to replacing a node that originally emits a C by a silent node. Thus, we can transform the nucleotide HMM into a nucleotide HMM with RNA editing awareness by replacing each node emitting a C in the original model has to be replaced by a pair of nodes one of which is silent and corresponds to an editing event. The transition scores into the silent node are identical to the original scores with an additional RNA editing penalty \( \gamma \).

Fig. 3. Replacement operations that transforms a nucleotide HMM into an HMM that includes single nucleotide C insertions. Each node emitting a C in the original model has to be replaced by a pair of nodes one of which is silent and corresponds to an editing event. The transition scores into the silent node are identical to the original scores with an additional RNA editing penalty \( \gamma \).
identical to the transition scores to the original node emitting C minus an ‘RNA editing cost’ $\gamma$. This cost discourages the algorithm from including too many editing sites into a proposed parse of the nucleotide sequence. In addition, this cost can be used to incorporate a priori statistical information on the sequence neighborhood of RNA editing sites by making it dependent on the identities of the neighboring nucleotides. For example, we observe for *P. polycephalum* that C insertions preferentially occur after a combination of a purine and a pyrimidine. In order to take this into account, we assign a smaller editing cost to sites that follow a purine–pyrimidine combination by setting

$$
\gamma(b_{j-1}, b_j) = \begin{cases} 
\gamma_0 - \gamma_1 & b_{j-1} \in \{G, A\} \text{ and } b_j \in \{C, T\} \\
\gamma_0 & \text{otherwise.}
\end{cases}
$$

### RESULTS

The downside of *P. polycephalum* being such an interesting target to determine new editing sites is that the number of known editing sites the algorithm can be tuned and tested on is rather small. In the mitochondrion of *Physarum* there are only six genes with known RNA editing sites. No editing in the nuclear genome of *Physarum* is known. In order to be able to test the performance of our algorithm we use all six genes in a leave-one-out approach.

For each of the six genes, nad7, cox1, cox3, cytb, atp and pl, we build the HMM starting from a sequence for the same gene in a different organism. Table 1 lists the accession numbers of the reference sequences used for each of the six genes. Then, we use the iterative database search tool PSI-BLAST (Altschul *et al.*, 1997) with its default parameter settings in order to find other sequences for the same gene from the non-redundant database. We iterate the PSI-BLAST search until convergence. Then, we extract the position-specific scoring matrix (PSSM) of the PSI-BLAST model. This matrix contains for each position $i$ in the original query sequence probabilities $p_i(b)$ to find each of the 20 amino acids at position $i$. These probabilities reflect the letter composition of all sequences included in the PSI-BLAST model during the iterations at the given query position. 2. We use these probabilities and the Robinson and Robinson (1991) background frequencies $p_b$ to find amino acid $b$ in any position in a protein to calculate the local scores $s_i(b) = \lambda^{-1} \log (p_i(b)/p_b)$. For the scaling factor $\lambda$, we use $\lambda = 0.318$ as appropriate for the usual form of the BLOSUM62 scoring matrix (Karlin and Altschul, 1990), the default scoring matrix of BLAST and PSI-BLAST.

The use of PSI-BLAST to construct the local scores $s_i(b)$ has two advantages over choosing a query sequence $a_1, \ldots, a_N$ and a scoring matrix $s_{a,b}$ and simply setting $s_i(b) = s_{a_i,b}$. First, the PSSM constructed by PSI-BLAST integrates information from many sequences for the gene in question. Thus, it can distinguish between very strongly conserved and rather variable positions in the sequence even if these have the same amino acid in the original query. Second, this approach makes it much less important which query sequence we start with. The set of sequences found by PSI-BLAST and included in the PSSM is more or less independent of the organism from which we choose the original query sequence. We indeed verified that our results do not change significantly if we start from different sequences than the ones shown in Table 1.

Using the local similarity scores $s_i(b)$ obtained as just described, the gap costs $\delta$ and $\epsilon$, the overall editing site cost $\gamma_0$ and the purine–pyrimidine editing site gain $\gamma_1$, we construct

<table>
<thead>
<tr>
<th>Gene</th>
<th>gi</th>
<th>Comp.gi</th>
<th>$\gamma_0/\gamma_1$</th>
<th>Amino acids</th>
<th>Missing amino acids</th>
<th>C insertions</th>
<th>Off by</th>
</tr>
</thead>
<tbody>
<tr>
<td>nad7</td>
<td>9757432</td>
<td>15676168</td>
<td>8/4</td>
<td>1548/1678 = 92%</td>
<td>183</td>
<td>116/171 = 68%</td>
<td>9</td>
</tr>
<tr>
<td>cox1</td>
<td>290967</td>
<td>16357068</td>
<td>8/5</td>
<td>1478/1591 = 93%</td>
<td>77</td>
<td>112/159 = 70%</td>
<td>8</td>
</tr>
<tr>
<td>cox3</td>
<td>15451796</td>
<td>10/7</td>
<td>1712/1894 = 91%</td>
<td>112</td>
<td>134/181 = 74%</td>
<td>9</td>
<td>14</td>
</tr>
<tr>
<td>cytb</td>
<td>3719264</td>
<td>15355/1694 = 93%</td>
<td>221</td>
<td>118/182 = 68%</td>
<td>11</td>
<td>11</td>
<td>6</td>
</tr>
<tr>
<td>atp</td>
<td>208876</td>
<td>8/7</td>
<td>153/1566 = 93%</td>
<td>168</td>
<td>106/152 = 70%</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>pl</td>
<td>456820</td>
<td>14/4</td>
<td>184/12008 = 93%</td>
<td>171</td>
<td>144/199 = 72%</td>
<td>10</td>
<td>18</td>
</tr>
</tbody>
</table>

Average

| Best | 1598/1731 = 92% | 155 | 122/172 = 71% | 9 | 13 | 7 | 30 |

For each gene, the access code for the mRNA sequence in *P. polycephalum* and the access code of the protein sequence used to build the position-specific scoring matrix are given. Then, the number and percentage of correctly predicted amino acids, the number of amino acids missed at the ends of the gene, the number and percentage of correctly predicted C insertions, and the number of C insertions that are predicted 1, 2, 3 or more than 4 nt away from their correct position are given. The latter numbers refer to predictions in all genes evaluated on all six genes.

---

5 We note that since the protein sequences of the six genes in *P. polycephalum* themselves are in the non-redundant database, they are also included in their respective models. However, this should not play any role, since there are several hundred to a thousand examples of each mitochondrial gene in the non-redundant database. Thus, each individual sequence has only minute influence on the position-specific scoring matrix constructed.
RNA editing HMMs for each gene as described in the previous sections. We apply this HMM with the local Viterbi algorithm to a region of the mitochondrial genome of *P. polycephalum* that is known to contain the respective gene. Backtracking the parse that gives the optimal score Σ through the HMM yields a prediction of the protein sequence of the gene in *Physarum* (the sequence of codon tracks traversed by the optimal parse) and of the position of editing sites (the position of silent editing site nodes traversed by the optimal parse.) We compare this prediction to the known protein sequences and the known editing site positions in *Physarum* for these genes. We quantify the performance of the algorithm by the fraction of amino acids and by the fraction of C insertion sites that it predicts correctly. Like all sequence alignment algorithms, the RNA editing site prediction algorithm has difficulties determining the beginning and end of a gene. Thus, we use only the fraction of correctly predicted amino acids and editing sites on the piece of the gene recognized by the algorithm. In cases where the C insertion occurs next to a C in the genomic sequence the editing site is not uniquely defined. This happens at roughly a quarter of all editing sites. In most such cases it is biologically not known which of the Cs is inserted. Thus, we count the prediction as correct if it identifies any of the equivalent positions as the editing site. This is consistent with our goal of primer design since for this purpose it is only important to predict the mRNA sequence. Note, that in a fully probabilistic framework we would have to normalize the HMM such that it considers only one of each set of equivalent editing sites in order to avoid that the total probability for the editing site is ‘smeared out’ across several possibilities. In the Viterbi framework considered here, this is not necessary. All equivalent parses of the sequence obtain the same score which is identical to the score of a uniquely defined editing site. Thus, the presence of several equivalent parses does not change the score associated with an editing event and there is no difference in score or the prediction between a normalized and this unnormalized model.

The first thing we note when applying the algorithm for different settings of the four parameters is that the gap costs δ and ε are hardly relevant for the prediction. We thus use the BLAST default values δ = 11 and ε = 1 from here on. In order to choose good values for the two parameters γ₀ and γ₁, we systematically try different combinations of these two parameters and assess the quality of the prediction for each parameter setting as described above. We identify for each gene the parameter choice that results in the highest percentage of correctly predicted C insertion sites. If there is more than one such parameter setting we choose the one with the highest percentage of correctly predicted amino acids among the editing-site-wise equally well performing parameter settings. Finally, we apply the algorithm for each of the six parameter choices to all five genes that have not been used to choose the parameters. The results are shown in Table 1. Most importantly, we see that over 90% of the amino acids and over 70% of the editing sites are predicted correctly. If we allow errors of up to three bases in the position of a predicted editing site, 87% of the editing sites are predicted correctly.

At this point we can judge how important the inclusion of biological information, i.e. the purine–pyrimidine reward is. The last two lines of Table 1 show the performance of the algorithm on all six genes. The last but first line shows the performance on the optimal parameter setting while the last line shows the performance on the optimal parameter setting without purine–pyrimidine reward, i.e. at γ₁ = 0. It can be clearly seen that the purine–pyrimidine reward hardly has an influence on the prediction of the amino acid sequence. However, it has a very big effect on the prediction of the editing sites themselves. Thus, we conclude that the purine–pyrimidine reward is mainly used to break degeneracies where the precise position of the editing site does not or hardly change the amino acid sequence and thus cannot be inferred from comparison to the protein family model alone.

Although there are no known genes without RNA editing in the mitochondrion of *Physarum*, we estimated the false positive rate of the predictions by running the algorithm at its optimal parameter setting (γ₀ = 12, γ₁ = 6) against the true mRNA sequences of the six genes instead of against their genomic counterparts. This resulted in a prediction of eight spurious editing sites in total for all six genes. In relation to the total of 7240 bases in all these six genes combined this is a very low number. All of these spuriously predicted editing sites were close to the ends of the proteins where the protein models tend to deteriorate in specificity.

At least, as far as this can be tested on the limited data available the performance of the algorithm transfers well to other organisms. The genomic DNA and the mRNA of the cox1 gene of four organisms with RNA editing have been sequenced (Horton and Landweber, 2000). Without any further tuning the algorithm predicts correctly four out of four U insertions in *Arcyria cinerea* and four out of the five U insertions in *Clastoderma debaryanum* (these organisms do not show C insertions in cox1) as well as 65% and 68% of the C insertions in *Stemonitis flavogenita* and *Didymium nigripes*, respectively.

CONCLUSION
We have presented a computational method that predicts RNA editing sites in mitochondrial genes of the slime mold *P. polycephalum*. The method uses information on homologs of the gene in other organisms and statistical information on editing sites specific for *Physarum*. We have shown that the method achieves a predictive performance of over 90% on the amino acid level and of over 70% on the editing site level. Although the implementation of the method discussed here is specific for *Physarum* and its close relatives the general approach can be easily applied to organisms with different
RNA editing features. This method will provide biologists explicit guidance for choosing primers for the experimental verification of new editing sites. In the future we would further improve the accuracy of the prediction by taking newly determined editing sites into account. We would also use the method to scan the genomes of organisms with RNA editing for regions that score unusually high under the model presented here. This might identify the presence or verify the absence of genes that have eluded other gene finding techniques due to a high degree of RNA editing.

ACKNOWLEDGEMENTS

I acknowledge fruitful discussions with Jonatha Gott that pointed me toward the problem of RNA editing and were very helpful all throughout the development of the method.

REFERENCES


