UPTEC X 02 005
MAR 2002

ISSN 1401-2138

PONTUS LARSSON

Development of application for predicting non-coding RNA genes

Master's degree project

Molecular Biotechnology Programme
Uppsala University School of Engineering

UPTEC X 02 005	**Date of issue 2002-03**
Author **Pontus Larsson**	
Title (English) **Development of application for predicting non-coding RNA genes**	
Title (Swedish)	
Abstract A strategy for predicting the occurrence of non-coding RNA genes was developed. The strategy is based on a combination of transcription signals, structural features and secondary and tertiary structure conservation. An application implementing this strategy was programmed using a combination of the Java and FORTRAN programming languages. The strategy and application was verified by predicting a certain class of tRNA genes in chromosomes 1-10 in *Sacharomyces cerevisiae*. 24 out of 24 predicted tRNAs were confirmed using a BLAST similarity search.	
Keywords Non-coding RNA, Hidden Markov model, Hairpin, RNA polymerase III, Application	
Supervisors **Anders Virtanen** **Department of Cell and Molecular Biology**	
Examiner **Leif Kirsebom** **Department of Cell and Molecular Biology**	
Project name	Sponsors
Language **English**	Security
ISSN 1401-2138	Classification
Supplementary bibliographical information	Pages **37**
Biology Education Centre Box 592 S-75124 Uppsala	Biomedical Center Husargatan 3 Uppsala Tel +46 (0)18 4710000 Fax +46 (0)18 555217

Development of application for predicting non-coding RNA genes

Pontus Larsson

Sammanfattning

Den traditionella synen på arvsmassa, DNA-molekylen, och hur en organism fungerar har varit att DNA innehåller ritningar för hur proteiner ska byggas och proteinerna i sin tur utför allt arbete i cellerna hos organismen. En nära släkting till DNA-molekylen är RNA-molekylen vilken också finns beskriven i DNA:t. Dess roll i cellen troddes länge vara begränsad till att fungera som en slags kod mellan DNA och proteiner. Tvärtemot denna uppfattning har RNA befunnits ha fler funktioner och under första halvan av 80-talet upptäckte man att RNA kunde fungera på samma sätt som proteiner. Sedan dess har många av dessa så kallade icke-kodande RNA-molekyler upptäckts och karaktäriserats. Emellertid är dessa molekyler svåra att upptäcka eftersom de inte har samma välkända och välstuderade kännetecken som de som kodar för proteiner. Målet med detta arbete har varit att utveckla en strategi för att upptäcka dessa gener och konstruera ett program som använder denna strategi. Genom att använda de numera kartlagda arvsmassorna hos vissa organismer, ska programmet kunna förutsäga förekomsten av dessa icke-kodande RNA-molekyler i arvsmassan.

Examensarbete 20 p i Molekylär bioteknikprogrammet

Uppsala universitet Januari 2002

Table of contents

Introduction

For a long time in molecular biology, the role of RNA in the cell was thought to be merely that of a molecule working as an intermediate between DNA and proteins. However, a growing awareness of the importance of RNA in various biological functions such as regulator of gene expression [1,2], protein secretion [3], tRNA processing [4] and splicing [5], has drastically changed the view on RNA and new functions have been discovered. Thus, in addition to genes coding for proteins there are also genes coding for functional RNA molecules that are never translated into proteins. These RNA molecules are small and have proven difficult to detect experimentally or by traditional computer based gene prediction [6].

The rapidly increasing amount of genomic sequences available calls for new fast and efficient means of identifying and characterizing genes and gene products. New gene finding software is continuously becoming available and thanks to the exponentially increasing performance of modern computers, identifying new genes can be a fast and relatively easy process. However, while great effort has been made to design software capable of detecting the traditional genes coding for proteins, less effort has been made to develop software capable of detecting genes coding for functional RNA molecules. Since these genes, in contrast to genes coding for proteins, lacks open reading frames and often are very short (<500 nt), finding characteristic and reliable traits that can be explored in a computational approach is difficult [6].

Recently, many previously unknown RNAs were discovered using a more heuristic approach based on transcription signals and genomic features present in these RNA molecules [7].

The aim of this project was to test a strategy for predicting the occurrence of genes encoding small, functional RNA molecules in genomic sequences and implement this strategy in an application.

The strategy

Whether the application is successful in detecting potential genes or not will depend upon the strategy on which it is based. If the search criteria are wisely chosen, hopefully the predictions will be accurate and conversely, if the criteria are less wisely chosen, the predictions will most likely be false and irrelevant.

A dilemma when trying to construct a model for this purpose is to infer the right amount of constraints on the potential genes. If an insufficient amount of constraints is used, the result will be a large number of possible hits so that the true genes are lost in the noise. On the other hand, if too many or too hard constraints are used, genes may fail to pass some criteria and too few hits may be the result. The important thing to acknowledge is that only the genes that fits the model can and will be found.

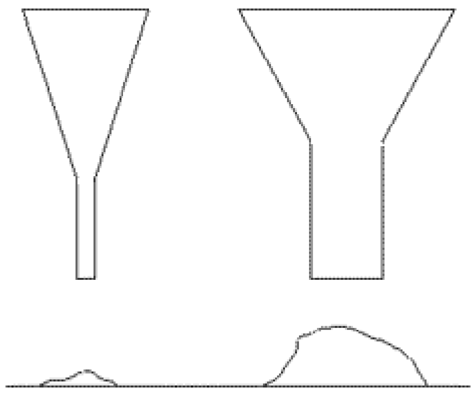


Fig 1. In the picture to the left, hard constraints allows only a small amount of information to fall through whereas, to the right, loose constraints allows much to pass and it can be hard to sort out the relevant information.

Since all genes must be transcribed from the blueprint in the DNA in order to become available to the cell, they must be associated with some kind of transcription signal. As a first criterion in the model, transcription signals are identified and sequences associated with these are passed on to the next stage. The second criterion concerns structure elements that may be present in the candidate genes. The interesting structure elements are primarily various loops that have stabilizing properties and possible functions in intermolecular interactions. Candidates that fail to meet these criteria are discarded. Third, a certain degree of conservation between species is assumed and the candidate genes are compared against other candidate genes from other species. The focal point of this comparison is secondary and tertiary rather than primary structure.

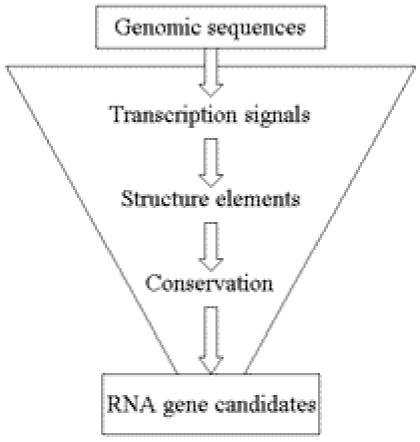


Fig 2. Illustration of the criteria of the model.

Having established this strategy for finding possible genes, one must realize that this is an approach that hopefully will identify one particular group of genes. Genes that do not meet these criteria are not found and sequences that meet the criteria may not be genes encoding functional RNA.

Hidden Markov models

Overview

Hidden Markov Models, or HMMs for short, have been widely used in a broad range of applications due to their excellent ability of signal processing [8]. The classic application for HMMs is speech recognition. In recent years, the use of HMMs to classify genomic data has grown in popularity and is today a widespread technique used in many multiple alignment- and gene finding software [9].

A HMM can be viewed as a machine that generates stochastic sequences of symbols. By choosing the parameters wisely it is possible to construct a model that generates sequences that are typical for *e.g.* a certain class of genes and the HMM can be said to model these genes. In order to find out if an unknown gene belongs to this class, a probability that the gene was generated by the HMM can be calculated. Based on this probability the unknown sequence can be classified as belonging or not belonging to the class of genes that the HMM represents.

HMM architecture

The HMM consists of **N** interconnected states, including a starting and a stopping state. Each state, or node, has a probability for emitting symbols and probabilities for transition to other states. Fig. 3 displays the general structure for a HMM with two states. $\mathbf{t_{ji}}$ denotes the probability of a transition from state $\mathbf{q_i}$ to state $\mathbf{q_j}$, $\mathbf{P(q_j|q_i)}$ and $\mathbf{e_{ix}}$ denotes the probability of emitting symbol **x** when in state $\mathbf{q_i}$, $\mathbf{P(x|q_i)}$.

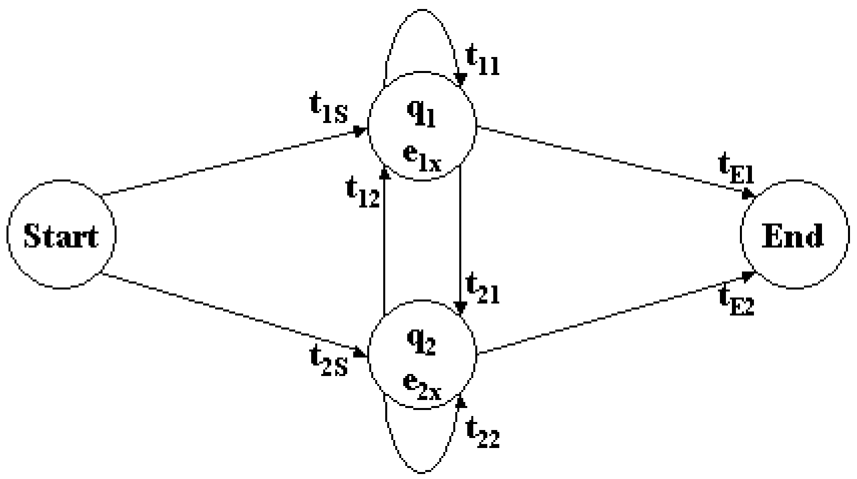


Fig 3. General structure for a HMM with two states, $\mathbf{q_1}$ and $\mathbf{q_2}$.

To illustrate how the HMM can be used, consider an example where you have two urns filled with a mixture of blue and red balls as in fig. 4. In the first urn, 2 out of 10 balls are red and 8 out of 10 are blue. This means that if you draw a ball from this urn the probability that it is red, **P(red | urn 1)**, is 0.2 and the probability that it is blue, **P(blue | urn 1)**, is 0.8. The corresponding probabilities for the second urn, are **P(red | urn 2)** = 0.4 and **P(blue | urn 2)** = 0.6.

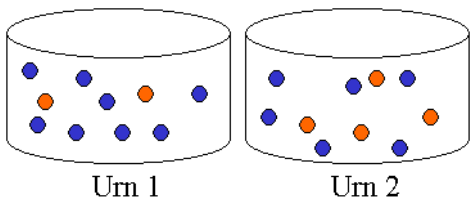


Fig 4. The ball and urn model

A person draws balls from these urns. To start with, the person has to decide which urn to draw the first ball from. The first urn will be chosen with some probability, **P(urn 1 | start)** and the second urn is chosen with another probability, **P(urn 2 | start)**. Next, a ball is drawn from the chosen urn and the colour is noted. The ball is then returned to the urn. The person now has three choices: a new ball can be drawn from the same urn, a new ball can be drawn from the other urn or the process can be ended. Depending on which urn the last ball was drawn from, the person will draw a ball from the same urn with probability **P(same urn | current urn)**, a ball will be drawn from the other urn with probability **P(other urn | current urn)** and the process will be ended with probability **P(end | current urn)**. The process is repeated until the person has chosen to stop. As a result you will have a sequence of coloured balls.

Now, imagine that you want to construct a model for the process of the person above drawing coloured balls from urns. You would then use two states as in fig. 4 representing the two urns. The emission probabilities for the two states represent the probabilities of drawing blue and red balls from each urn, hence $\mathbf{P(red|q_1) = 0.2}$, $\mathbf{P(blue|q_1) = 0.8}$, $\mathbf{P(red|q_2) = 0.4}$ and $\mathbf{P(blue|q_2) = 0.6}$. The transition probabilities will correspond to the person's likeliness to choose the different urns. When all the parameters have been determined you have a complete model and a complete set of parameters, λ.

HMM in practice

You might have a situation where a person gives you a sequence, **O**, of **T** coloured balls $\mathbf{\{O_1\ O_2\ \ldots\ O_T\}}$. Using the model above; you can now determine the probability that the person in the example drew the balls from the two urns and from which urn each ball was most likely drawn. In order to do so, there are two things that have to be calculated:

1. What is the probability that the model generated the sequence **O**, i.e. what is $\mathbf{P(O|\lambda)}$?
2. What path, π^*, through the model is the most probable given the sequence **O**, i.e. what is $\max_{\pi} \mathbf{P(\pi|O, \lambda)}$?

To start with, the first question is addressed. Since it cannot be determined from which urn each ball was drawn just by looking at the sequence of balls, the probabilities for all paths capable of generating **O** have to be calculated and to get the total probability they are summed together. One quickly realizes that this will yield a vast amount of calculations that needs to be carried out. Fortunately, the *forward-algorithm* can be used to systematically calculate all paths [10,11]. To do this, first the forward variable α is defined:

$$\boldsymbol{\alpha_t(i) = P(O_1\ O_2\ \ldots\ O_t,\ q_t = q_i \mid \lambda)} \qquad \textbf{(1)}$$

This is the probability for the observation of the partial sequence **{O_1 O_2 … O_t}** and that state $\mathbf{q_i}$ are visited at time **t** given the model parameters λ. With this definition α_T**(i)** can be calculated by a recursive procedure.

First the forward variable is initialised:

$$\alpha_1(\mathbf{i}) = t_{iS} e_{iO_1} \qquad \mathbf{1 \le i \le N} \qquad \mathbf{(2)}$$

This is the probability for making a transition from the starting state to state $\mathbf{q_i}$ and emitting the first symbol (**O_1**) in sequence **O** from state $\mathbf{q_i}$. The recursive step is calculated up to **t = T-1** according to:

$$\alpha_{t+1}(\mathbf{i}) = \left(\sum_{j=1}^{N} \left[\alpha_t(j) t_{ij} \right] \right) e_{iO_{t+1}} \qquad \mathbf{1 \le t \le T\text{-}1} \qquad \mathbf{(3)}$$

$$\mathbf{1 \le i \le N}$$

Then the transition to the end state has to be made and to get the total probability the probabilities for all states are summed:

$$\mathbf{P(O|\lambda)} = \sum_{i=1}^{N} \left[\alpha_T(i) t_{Ei} \right] \qquad \mathbf{(4)}$$

To understand the algorithm, consider equation 2 that gives the probability for the first symbol of the sequence being emitted by state $\mathbf{q_i}$, that is the probability for making a transition from the start node to state $\mathbf{q_i}$ multiplied by the probability for emitting the first symbol at that state. This is calculated for all states in the model and stored as the first column in a matrix α. Now, consider equation 3, the recursive step, when t = 1. Here the probability for being at state $\mathbf{q_i}$ and emitting the second symbol at time **t+1** = 2 is calculated. So far, the possible paths up to time **t** = 1 are stored in the first column of α**,** so if each element of this column is multiplied with the probability of making a transition from the state this element represents to state $\mathbf{q_i}$ and added together, the result is the probability for being at state $\mathbf{q_i}$ at this time, illustrated in fig. 5. By multiplying this with the probability for emitting the second symbol at this state, the result is the total probability so far. This is done for all states in the model and the results are stored in the second column of α.

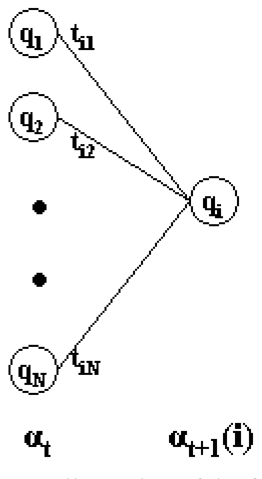


Fig 5. Illustration of the forward-algorithm. The total probability of reaching node $\mathbf{q_i}$ at time **t+1** is obtained by considering all paths that the node can be reached by.

This process is repeated for the whole sequence up to time **t** = T and the last column of α is summed to get the total probability that the sequence **O** was generated by the model, **P(O|λ)**.

In order to answer the second question, another algorithm and some new variables have to be introduced. As stated above, if every possible path through the model was to be examined, it would require a massive amount of calculations; instead an approach based on dynamical programming known as the *Viterbi-algorithm* is used [12,13]. The best state sequence, $\pi^* = \{\pi_1^* \pi_2^* \ldots \pi_T^*\}$, for an observed symbol sequence, $\mathbf{O} = \{O_1 O_2 \ldots O_T\}$, is desired. The variable δ is defined:

$$\delta_t(i) = \max_{\pi_1,\pi_2,\ldots,\pi_{t-1}} P(\pi_1 \pi_2 \ldots \pi_t = q_i, O_1 O_2 \ldots O_t \mid \lambda) \qquad (5)$$

This is the probability for the best state sequence up to time t-1 that ends in state $\mathbf{q_i}$ at time t. Again a recursive step:

$$\delta_{t+1}(i) = [\max_j \delta_t(j) t_{ij}]\, e_{iO_{t+1}} \qquad (6)$$

This step gives the probability for the emission of symbol $\mathbf{O_{t+1}}$ from state $\mathbf{q_i}$ when the most probable path so far has been used to get there. This algorithm will find the probability for the single most probable path through the model but the path needs to be remembered, *i.e.* which state **j** maximized eq. 6 for each **i** and **t**. For this purpose the matrix ψ is introduced. The algorithm now is as follows:

Initialise the variables:

$$\delta_1(i) = t_{iS} e_{iO_1} \qquad 1 \le i \le N \qquad (7)$$

$$\psi_1(i) = 0 \qquad 1 \le i \le N \qquad (8)$$

Recursive step:

$$\delta_t(i) = \max_{1\le j\le N}\left[\delta_{t-1}(j) t_{ij}\right] e_{iO_t} \qquad 2 \le t \le T, \quad 1 \le i \le N \qquad (9)$$

$$\psi_t(i) = \arg\max_{1\le j\le N}\left[\delta_{t-1}(j) t_{ij}\right] \qquad 2 \le t \le T, \quad 1 \le i \le N \qquad (10)$$

Termination:

$$P(\pi^*|O,\lambda) = \max_{\pi} P(\pi|O,\lambda) = \max_{1\le i\le N}\left[\delta_T(i) t_{Ei}\right] \qquad (11)$$

$$\pi_T^* = \arg\max_{1\le i\le N} \delta_T(i) \qquad (12)$$

Now all information needed in order to determine the optimal path is available. To do this, backtracking through the matrix ψ is performed and the optimal path π^* is noted:

$$\pi_t^* = \psi_{t+1}(\pi_{t+1}^*), \qquad t = T-1, T-2, \ldots, 1 \qquad (13)$$

Note the difference between the forward- and the Viterbi-algorithm. In the latter, *maximization* of the previous probabilities in order to perform the optimal action in every step is done instead of *summing* which is done to get the accumulated probability.

Hidden semi Markov models

Consider for a moment that you want to model a genomic sequence of a certain composition and length **L**. For simplicity, consider a spacer sequence that has an equal distribution of nucleotides. The model could be constructed as in fig. 6. It consists of only one node where the nucleotides A, C, G and T can be emitted, each with probability 1.0. Transitions can be made either back to the node itself or to the stop node.

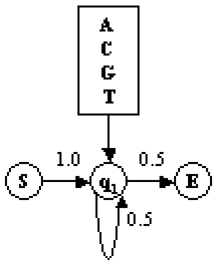


Fig 6. A hidden Markov model of spacer sequences. Emission probabilities are all 1.0.

After a little thought, one realizes that the probability for a spacer sequence of length **L** will be

$$\mathbf{P(O_1 \ldots O_L|\lambda)} = t_{1S} e_{1O_t} (t_{11} e_{1O_t})^{L-1} t_{E1} = \begin{bmatrix} t_{1S} = 1.0 \\ e_{1O_t} = 1.0 \\ t_{E1} = 1 - t_{11} \end{bmatrix} = t_{11}^{L-1}(1 - t_{11}) \quad \mathbf{(14)}$$

Plotting this probability against **l** and for some different $\mathbf{t_{11}}$ gives the curves shown in fig. 7.

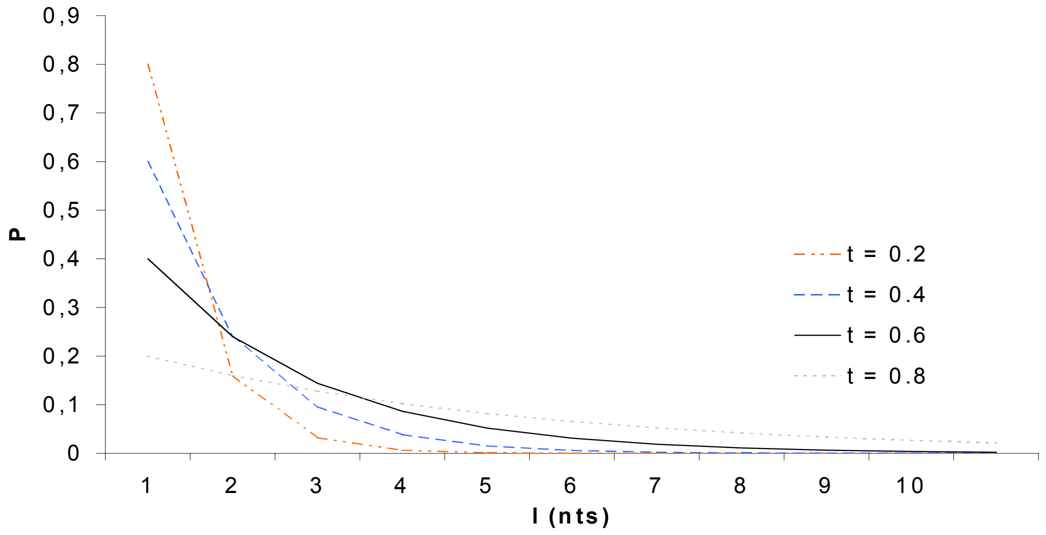


Fig 7. The probability of generating sequences of length **l** for different values of $\mathbf{t_{11}}$.

As can be seen in the plot, the probability for generating long sequences is decreasing exponentially, meaning that generating a short sequence will always be the most likely. Since spacers represent long, random sequences, a length distribution that has higher probability for long sequences would be desirable. Therefore a concept known as hidden *semi* Markov models (HSMMs) is introduced [14]. The basic idea is the same as for regular HMMs but in addition, each node is associated with a distribution of the probability for remaining in that node for some duration of time. This means that instead of being limited to the exponentially decreasing length distribution that arises spontaneously, one can have *e.g.* a gaussian or a rectangular distribution allowing for a much more flexible model.

The formulas introduced above need to be extended to account for this new type of model. The probability for remaining in a state for a duration of time, **d**, needs to be accounted for. **(3)** is modified in the following fashion:

$$\alpha_t(\mathbf{i}) = \sum_{j=1}^{N}\sum_{d=1}^{d_i}\left[\alpha_{t-d}(j)t_{ij}p_i(d)P(O_{t-d+1}...O_t \mid q_i)\right] \qquad \textbf{(15)}$$

As in **(3)**, the probabilities for transitions from all nodes leading up to the current node is summed, but the duration of time, **d**, spent in the current node is also taken into account. By the same logic as above, the initialisation step is obtained by multiplying **(2)** by the probability of a duration of 1 in state **i**.

$$\alpha_1(\mathbf{i}) = t_{iS}e_{iO_1}p_i(1) \qquad \mathbf{1 \le i \le N} \qquad \textbf{(16)}$$

The termination step is identical as above (**4**)

$$\mathbf{P(O|\lambda)} = \sum_{i=1}^{N}\left[\alpha_T(i)t_{Ei}\right] \qquad \textbf{(17)}$$

Now, **P(O|λ)** has been calculated but the best path needs to be found as well. To do this, **(7)-(13)** are modified in the same fashion as above and a new matrix μ is introduced. μ contains information about the optimal duration of time to remain in a state and the algorithm now looks like this:

Initiation:

$$\delta_0(\mathbf{i}) = \sigma_i \qquad \mathbf{1 \le i \le N} \qquad \textbf{(18)}$$

$$\psi_0(\mathbf{i}) = \mathbf{0} \qquad \mathbf{1 \le i \le N} \qquad \textbf{(19)}$$

$$\mu_0(\mathbf{i}) = \mathbf{0} \qquad \mathbf{1 \le i \le N} \qquad \textbf{(20)}$$

Recursive step:

$$\delta_{t+1}(\mathbf{i}) = \max_{d_{\min} \le d \le d_{\max}}\left\{\left[\max_{1 \le j \le N}\left(\delta_{t-d}(j)t_{ij}\right)\right]p_i(d)P(O_{t-d+1}...O_{t+1} \mid q_i)\right\} \qquad \mathbf{1 \le i \le N} \qquad \mathbf{0 \le t \le T\text{-}1} \qquad \textbf{(21)}$$

$$\boldsymbol{\mu_{t+1}(i)} = \underset{d_{\min}\le d\le d_{\max}}{\arg\max}\left\{\left[\max_{1\le j\le N}\left(\delta_{t-d}(j)t_{ij}\right)\right]p_i(d)P(O_{t-d+1}...O_{t+1}\mid q_i)\right\} \quad \mathbf{1 \le i \le N}$$

$$\mathbf{0 \le t \le T\text{-}1} \qquad \mathbf{(22)}$$

$$\boldsymbol{\psi_{t+1}(i)} = \underset{1\le j\le N}{\arg\max}\left\lfloor\delta_{t-\mu_{t+1}}(j)t_{ij}\right\rfloor \quad \mathbf{1 \le i \le N}$$

$$\mathbf{0 \le t \le T\text{-}1} \qquad \mathbf{(23)}$$

Termination:

$$\mathbf{P(\pi^*|O,\lambda)} = \max_{\pi}\mathbf{P(\pi|O,\lambda)} = \max_{1\le i\le N}\delta_T(i) \qquad \mathbf{(24)}$$

$$\boldsymbol{\pi_T}^* = \underset{1\le i\le N}{\arg\max}\,\delta_T(i) \qquad \mathbf{(25)}$$

As above, the matrix ψ contains information about the states that maximized the probability for a certain state at a certain time. Again the backtracking starts at the ending state $\boldsymbol{\pi_T}^*$, but rather than stepping backwards one step at a time, the duration of time to remain in this state is considered. This information is found in $\boldsymbol{\mu_T(\pi_T^*) = \mu_T^*}$. $\boldsymbol{\mu_T}^*$ steps are backtracked and the most probable state $\pi^*_{T-\mu^*_T}$ are determined. This process is repeated until **t** = **0** is reached. By backtracking the best path is found and in the process the sequence the states were visited in are noted along with the time each state was visited.

A problem often encountered when dealing with hidden Markov models and long sequences is that $\mathbf{P(O_t|\lambda)}$ will show an exponential decrease with increasing **t**. This results in very small numbers and sooner or later, the machine epsilon of the computer will be reached, causing the computer to consider the probability as zero. To solve this, some sort of scaling has to be introduced. The simplest way to achieve this is to calculate $\mathbf{\log P(O|\lambda)}$ instead of $\mathbf{P(O|\lambda)}$ [15]. This will not change the result since only the *relative* probabilities are interesting. The logarithm has the property:

$$\mathbf{\log (x) > \log (y)} \qquad \mathbf{x > y > 0} \qquad \mathbf{(26)}$$

This guarantees that the relative order of the probabilities will be conserved and now an exponential decrease in magnitude is transformed into a linear decrease. Formulas **(7), (9)** and **(11)** have to be modified in a straightforward way. **(7)**, the initiation step, will be

$$\boldsymbol{\phi_1(i)} = \log(t_{iS}) + \log(e_{iO_1}) \qquad \mathbf{1 \le i \le N} \qquad \mathbf{(27)}$$

Formula **(9)**, the recursive step:

$$\boldsymbol{\phi_t(i)} = \max_{1\le j\le N}\left\lfloor\log(\phi_{t-1}(j) + \log(t_{ij})\right\rfloor + \log(e_{iO_t}) \quad \mathbf{1 \le i \le N} \qquad \mathbf{(28)}$$

Finally, **(11)**, the termination step:

$$\mathbf{\log P(O|\lambda)} = \max_{1\le i\le N}\left[\phi_T(i)\right] \qquad \mathbf{(29)}$$

Implementing a hidden Markov model for gene prediction

The goal of this project was to predict RNA-genes from genetic sequences. Focus was set on genes transcribed by *RNA polymerase III*, whose transcription signals have been well characterized [16]. The promoter directs the RNA polymerase III transcription machinery to the correct site where transcription is initiated. The promoter is *internal*, *i.e.* it is located within the transcribed part of the gene, downstream of the start point and it consists mainly of three conserved structural elements. These appear in two conformations. As illustrated in fig. 8, it is either the box A sequence separated from the box B sequence or box A separated from the box C sequence. A third type of polymerase III promoter exists as well but contrary the promoter described above, it lies upstream of the transcribed part, it is not common and it is not very well characterized. No attention is directed toward this type of promoter.

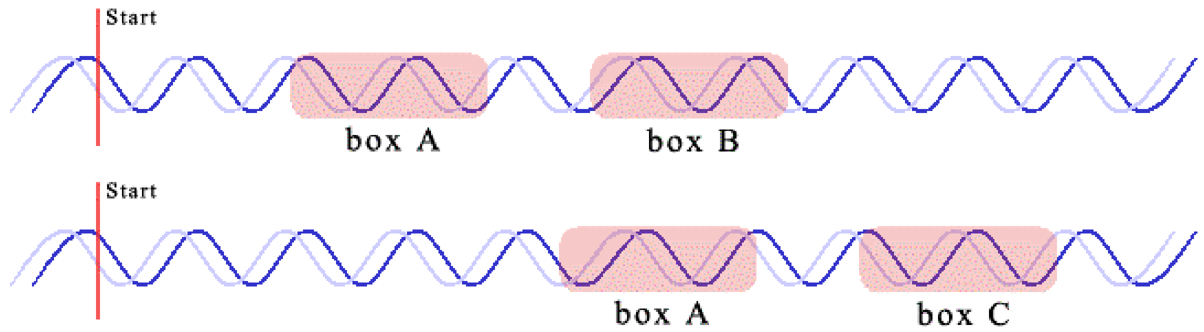
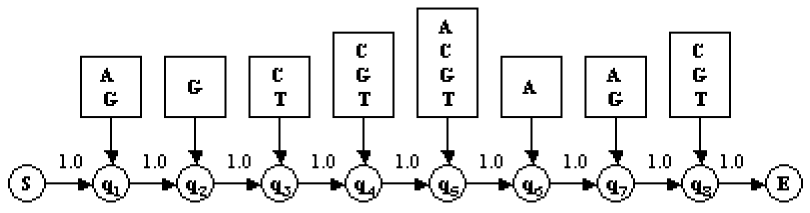
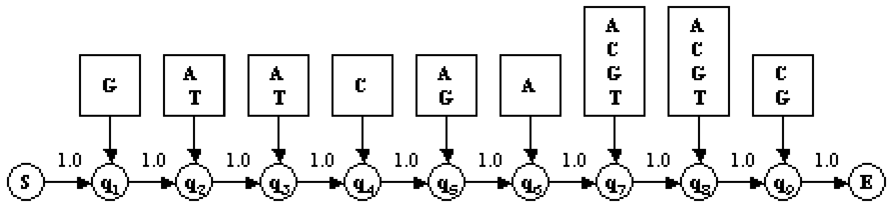


Fig 8. RNA polymerase III promoter elements downstream from the starting point of transcription

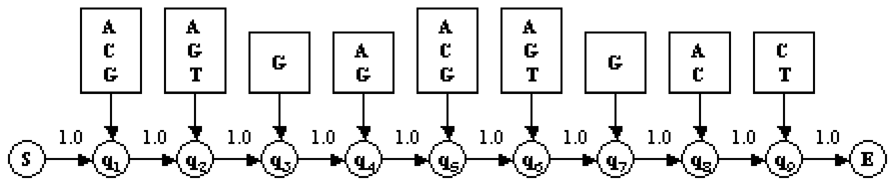
Statistic data on the promoter elements was gathered by aligning sequences of various non-coding RNAs, downloaded from the IMB-JENA website [17], with ClustalW [18]. In another project, this data was combined with literature studies to determine consensus sequences for the promoter elements [19]. These consensus sequences were utilized to design HMMs that model RNA polymerase III promoter boxes. Schematic descriptions of these models are shown in fig. 9.



Hidden Markov model of the box A sequence



Hidden Markov model of the box B sequence



Hidden Markov model of the box C sequence

Fig 9. Schematics of the HMMs modelling the box A-, box B- and box C sequences. All emissions occur with probability 1.0.

In order to process genomic sequences, the models have to be placed in an appropriate context. As mentioned earlier, the promoter consists of box A and either box B or box C. This behaviour is modelled by linking together the HMMs for these sequence motifs together with HMMs that model spacer elements, shown in fig. 6. The resulting model for the RNA polymerase III promoter is shown in fig. 10.

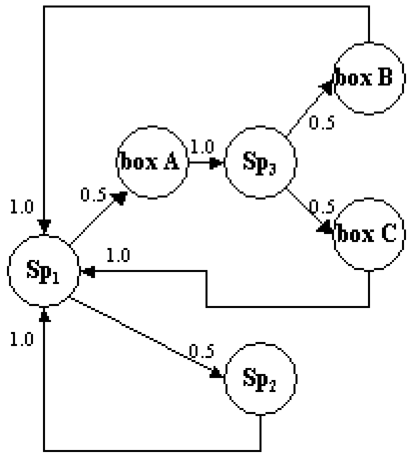


Fig 10. Model for the RNA polymerase III promoter. Sp_1, Sp_2 and Sp_3 are spacer sequences

The structure shown in fig.10 has some important properties. It consists of 6 individual HMMs linked together and rather than emitting one symbol at a time partial sequences are generated from each HMM. Three of the HMMs model the transcription elements and are shown in fig. 9. The other three HMMs are spacer sequences, each with an associated probability distribution hence they are actually HSMMs. As can be seen, the HMMs make up two closed loops, an upper and a lower. The upper path models an RNA polymerase III promoter and the lower represents everything that isn't a promoter. Since all loops are closed, this model will be capable of modelling sequences of arbitrary length and composition and by choosing the parameters of the HMMs wisely, the model will detect all sequences corresponding to a polymerase III promoter. In fact, the parameters are chosen so that the path through the upper model, the promoter, will always be the most likely. However, if the sequence doesn't match the consensuses, the model is forced to take the lower path through the spacer sequence. This way, whenever the model can take the upper path, corresponding to a promoter, it will do so but otherwise it is forced to pass through the spacer model.

Structural elements

As mentioned earlier, structural elements are identified in the potentially interesting RNA molecules. Focus has been set on various forms of hairpin loops occurring in the molecule. A general hairpin loop can be seen in fig. 11. It consists of a loop of various lengths, ranging from 4 nucleotides and upwards and a stem. The stem is double stranded and base pairing occurs but mismatches in the stem can cause patches of unpaired bases called bulges to appear. These bulges can stabilize the stem by relaxing the helix and they may also function as recognition sites for proteins or other RNA molecules [20,21,23]. A common motif is the tetraloop that has 4 nucleotides in the loop and is one of the most fundamental building blocks when it comes to RNA tertiary interactions. Tetraloops with loop consensuses of UNCG is important due to their stabilizing probabilities and loops with a consensus of GNRA has proven to be crucial for intermolecular interactions [20,22]. Since these characteristics are presumably crucial to an RNA molecule with some sort of function, structural or catalytic, only sequences that contain at least one GNRA loop or one UNCG loop are considered possible gene candidates.

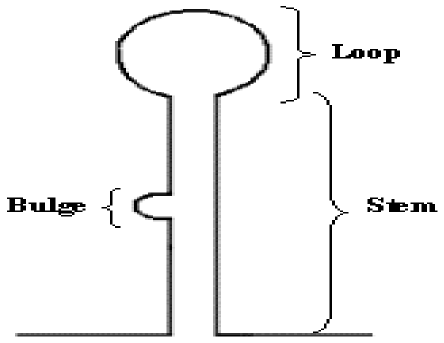


Fig 11. A general hairpin loop.

Structure homology

If an RNA molecule has managed to acquire a structural or catalytic function, it seems reasonable that its function and structure will be under selection pressure during the course of evolution. Therefore, a certain amount of conservation between species is expected. This homology may not necessarily be a conserved primary structure but rather a conserved tertiary structure with important secondary motifs conserved as well [6], the important thing being that these secondary motifs occupy the same positions relative to each other in the folded structure. A simple approach to determine if this is the case would be to measure the internal distance between the important secondary motifs and compare this distance with distances between similar structures in other species.

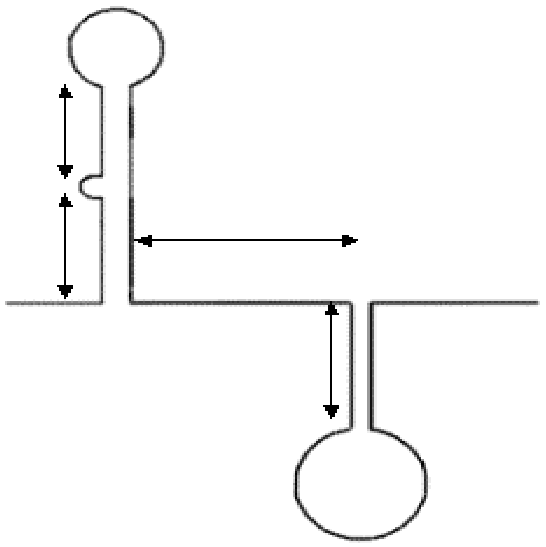


Fig 12. The distance between important secondary motifs as well as the internal distance within a motif should be conserved to give a conserved tertiary structure.

Programming the application

Computer languages

When choosing computer language for an application, there are many aspects that must be taken into account. Issues such as performance, interface and compatibility should be considered. The different languages available all have their own pros and cons and there is no such thing as the ultimate programming language. Traditionally, C and also C++ are very popular languages to use when writing applications. Most operating systems are written in C and there are few limitations to the possibilities when it comes to applications. Java is a pretty new language and apart from being a fully object oriented language its main advantage is that it is not platform dependant. This means that it does not matter if you compile your code on PC, UNIX or Macintosh, it will run on all platforms. This is because there is an underlying interpreter that translates the so-called Java byte code into machine code for the current platform. Another nice advantage of Java is that it is very easy to design graphical interfaces using native Java classes instead of various commercial tools or very complicated programming. Unfortunately, the process of interpreting the Java byte code makes Java a slow language in comparison; hence it is not suitable for applications demanding heavy calculations [24]. One of the earliest so-called high-level languages is FORTRAN, introduced in 1954. While being a language specifically designed to be easy to understand and write programs in (in contrast to machine code or assembler language), optimisation has always been stressed which makes it a popular language when it comes to scientific calculations. The language has been revised several times (FORTRAN 66, FORTRAN 77, FORTRAN 90, FORTRAN 95) in order to keep it up to date. However, a significant drawback of FORTRAN is its lack of convenient means of handling input and output [25].

Program structure

Since there is no language that combines the simplicity of Java with the efficiency of FORTRAN, the application was implemented with different modules programmed in different languages. An interface that handles user input and output was programmed in Java and a module that implements the HSMM described above was programmed in FORTRAN 90. This module communicates with the Java interface only, not with the user directly. Input parameters to the HSMM are the genomic sequence and the HSMM parameters. Output is the optimal path through the model.

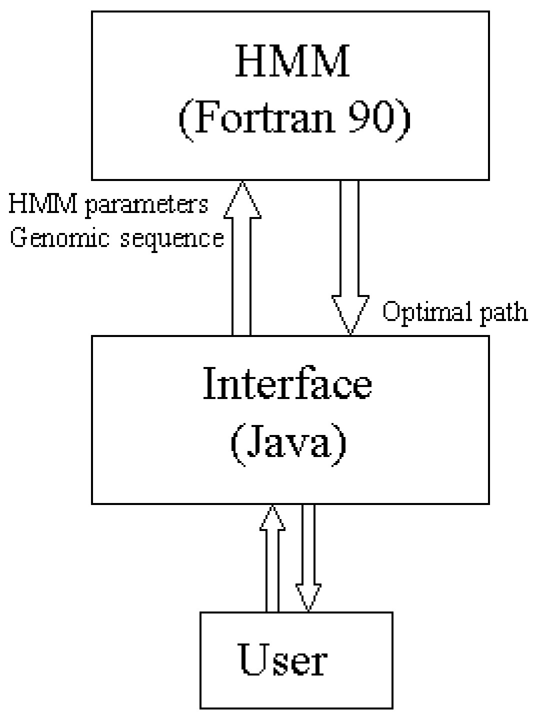


Fig 13. The flow of information between program modules

The Java module parses the output from the HSMM and presents the result to the user. The user can then apply filters and further process the data in order to identify candidate genes.

Verification of HMM and filters

In order to evaluate and verify the functionality of the RNA polymerase III model and the various filters of the application, genomic files were scanned with the purpose of identifying genes coding for transfer RNA, tRNA. tRNA is a functional RNA molecule transcribed by RNA polymerase III and it has a promoter structure consisting of a box A and a box B [26]. In order to identify tRNA genes, the genome of baker's yeast, *Saccharomyces cerevisiae*, was run through the RNA polymerase III-model. A filter isolating the parts of the genome that had passed through the path of the model corresponding to the promoter was applied. The resulting sequences were scanned for the occurrence of known and well-defined hairpin loop structures in tRNA, see fig. 14.

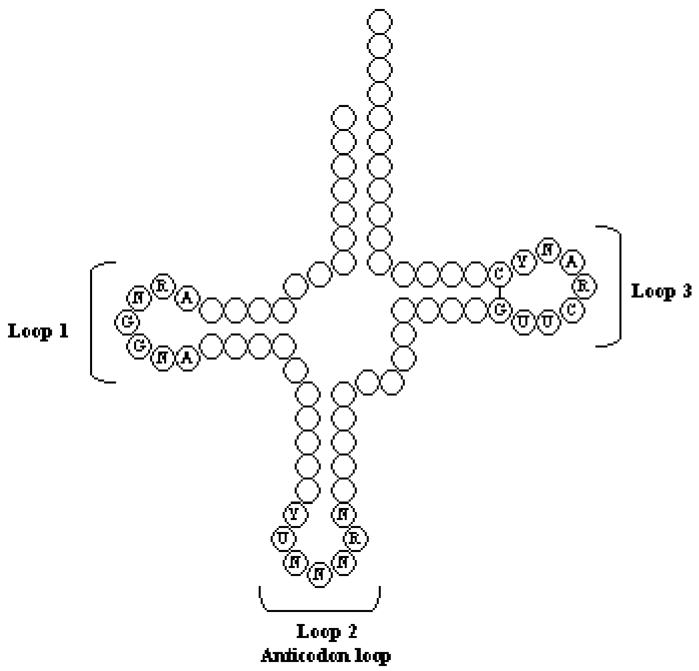


Fig 14. Structure of tRNA. There are three characteristic hairpin loops, one of which is the loop containing the anticodon. Loop 1 may have up to four extra nucleotides but in this verification, only the loop in the figure was considered.

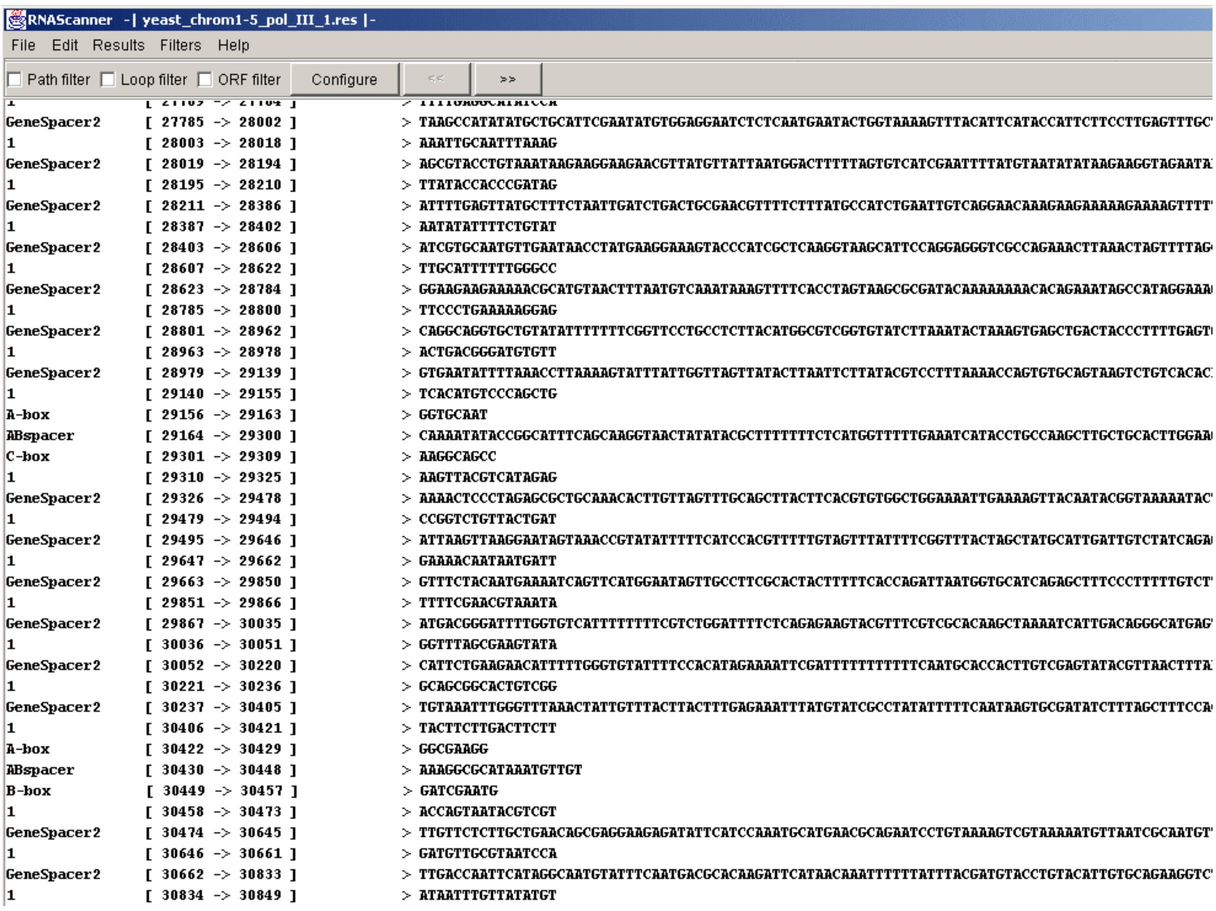
There are three such structures: the first one is a hairpin loop with a four base pair stem and a loop consensus of **ARNGGNA**. There are some slight variations on the length of this loop but only loops with the consensus above were considered. The second structure is the anticodon loop and it has a stem of five base pairs and a **YTNNNRN** loop consensus. The three middle nucleotides are in fact the anticodon. The third hairpin has a five base pair stem and a **TTCRANY** loop consensus. In addition, the base pair closing the loop is a GC pair [27]. Scanning the sequences and demanding the presence of all three loops resulted in 24 candidates from the 10 first chromosomes of the genome. The candidate tRNAs were fed into **BLAST** [28,29] and all of them were confirmed to contain actual tRNA genes.

The RNAScanner

The RNAScanner offers a computational approach to finding non-coding RNAs in genomic sequences through a graphical and highly interactive process. A screenshot of the GUI can be seen in fig. 15. Through the menubar and the toolbar the user can access various functions and the output is presented on the screen. Due to the massive amount of data typically processed in the program, measures have to be taken to avoid running out of memory. As a first step, the user should be sure to allocate as much working memory as possible to the application. This can be done by specifying the amount of RAM the application is allowed to use at the command prompt. In order to launch the application, the user gives the following command at the command prompt. The Ns are substituted for the amount of memory in bytes that is to be allocated.

java –XmxNNNNNNNN RNAScanner

The user may also launch the application by double-clicking on the enclosed batch file, **RNAScanner.bat**. This file allocates 500MB of RAM. As a second step to avoid memory failure, the program cannot show more than a limited amount of output on the screen. Instead, the results are divided onto many pages that the user can browse.



```
GeneSpacer2     [ 27785 -> 28002 ]          > TAAGCCATATATGCTGCATTCGAATATGTGGAGGAATCTCTCAATGAATACTGGTAAAAGTTTACATTCATACCATTCTTCCTTGAGTTTGC
1               [ 28003 -> 28018 ]          > AAATTGCAATTTAAAG
GeneSpacer2     [ 28019 -> 28194 ]          > AGCGTACCTGTAAATAAGAAGGAAGAACGTTATGTTATTAATGGACTTTTTAGTGTCATCGAATTTTATGTAATATATAAGAAGGTAGAATA
1               [ 28195 -> 28210 ]          > TTATACCACCCGATAG
GeneSpacer2     [ 28211 -> 28386 ]          > ATTTTGAGTTATGCTTTCTAATTGATCTGACTGCGAACGTTTTCTTTATGCCATCTGAATTGTCAGGAACAAAGAAGAAAAAGAAAAGTTTT
1               [ 28387 -> 28402 ]          > AATATATTTTCTGTAT
GeneSpacer2     [ 28403 -> 28606 ]          > ATCGTGCAATGTTGAATAACCTATGAAGGAAAGTACCCATCGCTCAAGGTAAGCATTCCAGGAGGGTCGCCAGAAACTTAAACTAGTTTTAG
1               [ 28607 -> 28622 ]          > TTGCATTTTTTGGGCC
GeneSpacer2     [ 28623 -> 28784 ]          > GGAAGAAGAAAAACGCATGTAACTTTAATGTCAAATAAAGTTTTCACCTAGTAAGCGCGATACAAAAAAAACACAGAAATAGCCATAGGAAA
1               [ 28785 -> 28800 ]          > TTCCCTGAAAAAGGAG
GeneSpacer2     [ 28801 -> 28962 ]          > CAGGCAGGTGCTGTATATTTTTTTCGGTTCCTGCCTCTTACATGGCGTCGGTGTATCTTAAATACTAAAGTGAGCTGACTACCCTTTTGAGT
1               [ 28963 -> 28978 ]          > ACTGACGGGATGTGTT
GeneSpacer2     [ 28979 -> 29139 ]          > GTGAATATTTTAAACCTTAAAAGTATTTATTGGTTAGTTATACTTAATTCTTATACGTCCTTTAAAACCAGTGTGCAGTAAGTCTGTCACAC
1               [ 29140 -> 29155 ]          > TCACATGTCCCAGCTG
A-box           [ 29156 -> 29163 ]          > GGTGCAAT
ABspacer        [ 29164 -> 29300 ]          > CAAAATATACCGGCATTTCAGCAAGGTAACTATATACGCTTTTTTTCTCATGGTTTTTGAAATCATACCTGCCAAGCTTGCTGCACTTGGAA
C-box           [ 29301 -> 29309 ]          > AAGGCAGCC
1               [ 29310 -> 29325 ]          > AAGTTACGTCATAGAG
GeneSpacer2     [ 29326 -> 29478 ]          > AAAACTCCCTAGAGCGCTGCAAACACTTGTTAGTTTGCAGCTTACTTCACGTGTGGCTGGAAAATTGAAAAGTTACAATACGGTAAAAATAC
1               [ 29479 -> 29494 ]          > CCGGTCTGTTACTGAT
GeneSpacer2     [ 29495 -> 29646 ]          > ATTAAGTTAAGGAATAGTAAACCGTATATTTTTCATCCACGTTTTTGTAGTTTATTTTCGGTTTACTAGCTATGCATTGATTGTCTATCAGA
1               [ 29647 -> 29662 ]          > GAAAACAATAATGATT
GeneSpacer2     [ 29663 -> 29850 ]          > GTTTCTACAATGAAAATCAGTTCATGGAATAGTTGCCTTCGCACTACTTTTTCACCAGATTAATGGTGCATCAGAGCTTTCCCTTTTTGTCT
1               [ 29851 -> 29866 ]          > TTTTCGAACGTAAATA
GeneSpacer2     [ 29867 -> 30035 ]          > ATGACGGGATTTTGGTGTCATTTTTTTTCGTCTGGATTTTCTCAGAGAAGTACGTTTCGTCGCACAAGCTAAAATCATTGACAGGGCATGAG
1               [ 30036 -> 30051 ]          > GGTTTAGCGAAGTATA
GeneSpacer2     [ 30052 -> 30220 ]          > CATTCTGAAGAACATTTTTGGGTGTATTTTCCACATAGAAAATTCGATTTTTTTTTTTCAATGCACCACTTGTCGAGTATACGTTAACTTTA
1               [ 30221 -> 30236 ]          > GCAGCGGCACTGTCGG
GeneSpacer2     [ 30237 -> 30405 ]          > TGTAAATTTGGGTTTAAACTATTGTTTACTTACTTTGAGAAATTTATGTATCGCCTATATTTTTCAATAAGTGCGATATCTTTAGCTTTCCA
1               [ 30406 -> 30421 ]          > TACTTCTTGACTTCTT
A-box           [ 30422 -> 30429 ]          > GGCGAAGG
ABspacer        [ 30430 -> 30448 ]          > AAAGGCGCATAAATGTTGT
B-box           [ 30449 -> 30457 ]          > GATCGAATG
1               [ 30458 -> 30473 ]          > ACCAGTAATACGTCGT
GeneSpacer2     [ 30474 -> 30645 ]          > TTGTTCTCTTGCTGAACAGCGAGGAAGAGATATTCATCCAAATGCATGAACGCAGAATCCTGTAAAAGTCGTAAAAATGTTAATCGCAATGT
1               [ 30646 -> 30661 ]          > GATGTTGCGTAATCCA
GeneSpacer2     [ 30662 -> 30833 ]          > TTGACCAATTCATAGGCAATGTATTTCAATGACGCACAAGATTCATAACAAATTTTTTATTTACGATGTACCTGTACATTGTGCAGAAGGTC
1               [ 30834 -> 30849 ]          > ATAATTTGTTATATGT
```

Fig 15. Partial screenshot of the RNAScanner. Results are shown as the complete path through the model.

When the application launches for the first time, it will create all necessary subfolders it needs. There will be a directory named "Fortran", this is the directory where files that the FORTRAN module uses and outputs. A directory labelled "Models" where files containing HMM and model parameters are stored and a directory labelled “Results” where results will be stored are also created. In the “Results”-directory, a subdirectory “Data” will be created where saved data will be stored. Finally, a directory named “Sequences” is created and that is where the program will look for files containing genomic sequences. In addition, temporary files and directories may be created during the execution of the program but these will be deleted automatically.

Functions are accessed through the menus; the filters can be controlled through the toolbar as well.

File menu

RNA scan - Starts a HMM scan of a genomic sequence. A dialog appears where the user specifies a file defining the model, a so-called linker file, and a sequence file. The sequence file must be in FASTA [30] or raw format and have a .seq file extension. If several genomic files are to be scanned using the same linker file, multiple files can be specified and will then be scanned in a consecutive order. The results will be automatically stored in a file that is named according to the rule: [sequence file name]_[linker file name].res. When the scan is done the complete path through the model will be presented on the screen for further processing. Note: the scan may be very time consuming depending of the size of the genomic file(s) and the complexity of the HMM, it may run for several days or weeks. Since it is a heavy computational process, the computer will be virtually inaccessible during the run.
Reset – Clears the screen and releases allocated memory.
Exit - Exits the RNAScanner.

Edit menu

Edit linker – Opens the Link editor. See separate section.
Edit HMM – Opens the HMM editor. See separate section.
Invert strand – Starting from a strand in 5’ $\rightarrow$ 3’ direction, this option creates a file containing the complementary strand, also in 5’ $\rightarrow$ 3’ direction. A dialog appears where the user can specify the original file. The created file is named according to [original filename]_comp.seq.

Results menu

Save data – The user can save the results currently showing on the screen.
Load data – Loads a saved data file.
Load results – Loads a previously stored result file. A dialog appears and the user is asked to specify the result file and the link file used to produce the results.
Secondary structure homology – Allows for a comparison of secondary and tertiary structure between multiple data files.
Show partial sequence – Lets the user specify an interval of the genomic sequence that will be shown.

Filters menu

Path filter – Applies a filter that allows the user to view only the nodes corresponding to a piece of the path. See separate section.
Loop filter – Applies a filter that searches for loop structures as defined by the user. See separate section.
ORF filter – Removes potential genes that contain an open reading frame. See separate section.

Help menu

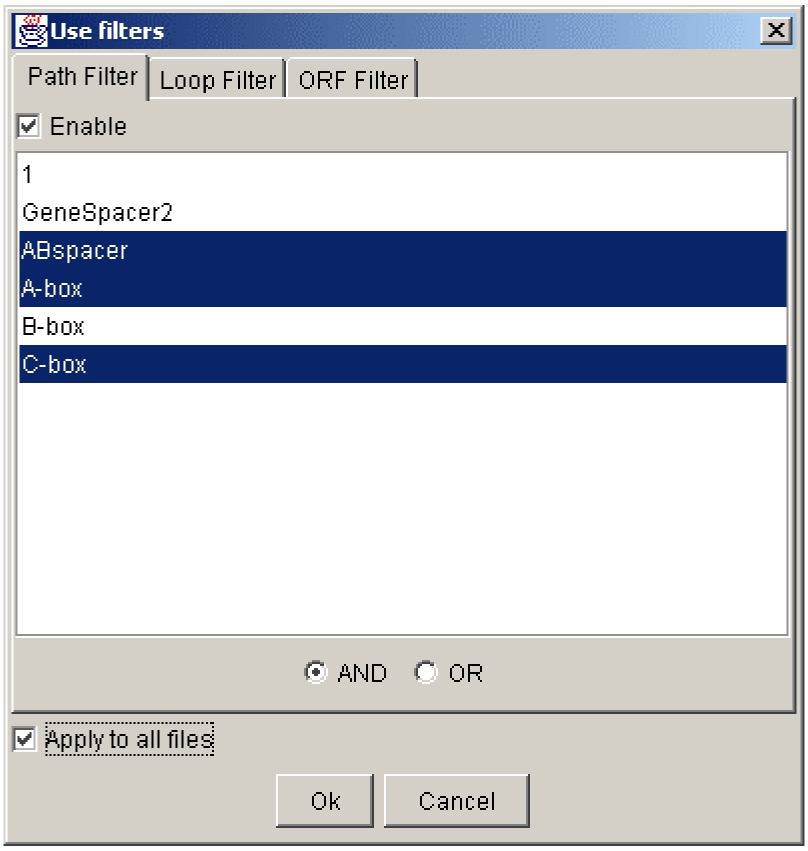
Help – Displays a help file describing the various functions.

Toolbar

Path filter – Applies a filter that allows the user to view only the nodes corresponding to a piece of the path. See separate section.
Loop filter – Applies a filter that searches for loop structures as defined by the user. See separate section.
ORF filter – Removes potential genes that contain an open reading frame. See separate section.
Configure – Configures the applied filters allowing for some tweaking of the applied filters or application of new filters.
Backward / Forward – Used for browsing through the pages of results.

The path filter

The path filter lets the user show only partial sequences corresponding to a piece of the path through the HMM used to obtain the results.



[✓] Enable

1
GeneSpacer2
ABspacer
A-box
B-box
C-box

[● AND] ○ OR

[✓] Apply to all files

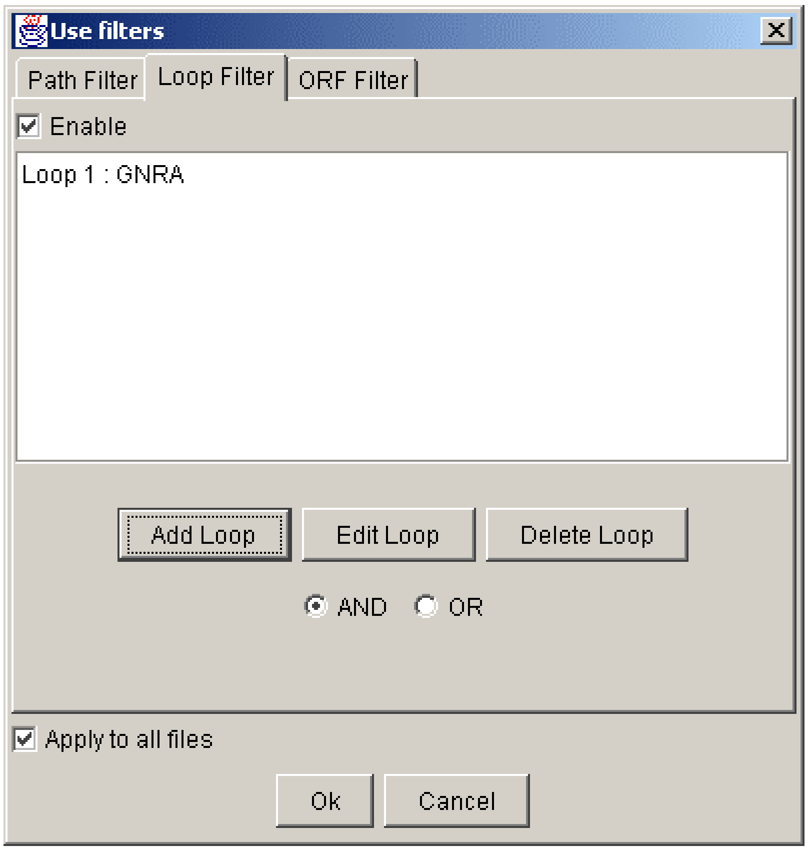
Fig 16. Configuring the path filter. In the figure, nodes corresponding to the path through the box A, box C and the spacer between them are selected, making up a typical RNA polymerase III promoter.

Features

Enable – Enables or disables the filter.
Node selection – The nodes corresponding to a piece of the path can be selected. Multiple selections are allowed.
AND button – If the AND option is selected, only the paths where the selected nodes occurs in succession are shown.
OR button – If the OR option is selected, all paths through any of the selected nodes are shown.
Apply to all files – If this box is checked, the selected filters will be applied to all pages of results. If the box is unchecked, the selected filters will only be applied to the result page currently showing.

The loop filter

The loop filter scans sequences for the possibility of forming various hairpin loop structures that can be defined by the user. Features such as presence or absence of a bulge in the stem, consensus of the loop, length of the stem and loop, constraints on the base pair closing the loop and more can be specified.



[✓] Enable

Loop 1 : GNRA

[✓ AND] OR

[✓] Apply to all files

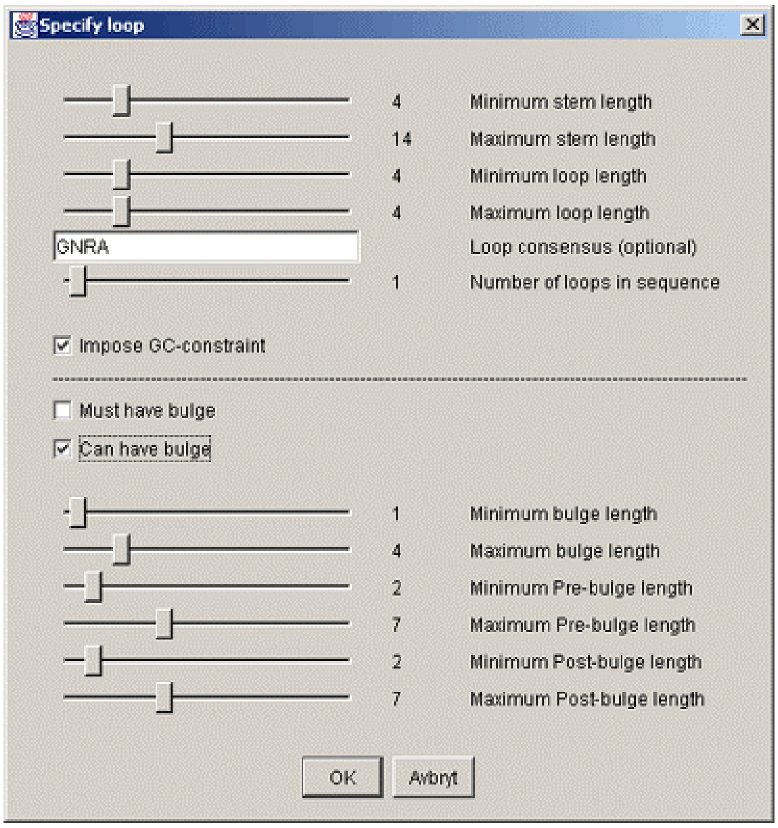
Fig 17. Configuring the loop filter. As can be seen, a loop with GNRA-consensus has been added to the filter.

Features

Enable – Enables or disables the filter.
Added loops – Displays the added loops. If specified, the consensus of the loop is shown. Added loops can be deleted or edited by selecting the loop and clicking the appropriate button.
Add loop – Adds a new loop to the filter. Brings up a new dialog where features of the loop can be specified.
Edit loop – Lets the user modify the selected loop.
Delete loop – Removes the selected loop from the filter.
AND button – If the AND option is selected, all added loops must be present in the sequence for it to pass the filter.
OR button – If the OR option is selected, it is enough that one of the added loops is present in the sequence for it to pass the filter.
Apply to all files – If this box is checked, the selected filters will be applied to all pages of results. If the box is unchecked, the selected filters will only be applied to the result page currently showing.

Configuring a loop

When adding a new loop to the filter, a dialog appears asking the user to specify what the features of the loop should be. The settings have preset values that may be used or they can be changed using the sliders.

a) 

Specify loop

4	Minimum stem length
14	Maximum stem length
4	Minimum loop length
4	Maximum loop length
GNRA	Loop consensus (optional)
1	Number of loops in sequence

☑ Impose GC-constraint

☐ Must have bulge

☑ Can have bulge

1	Minimum bulge length
4	Maximum bulge length
2	Minimum Pre-bulge length
7	Maximum Pre-bulge length
2	Minimum Post-bulge length
7	Maximum Post-bulge length

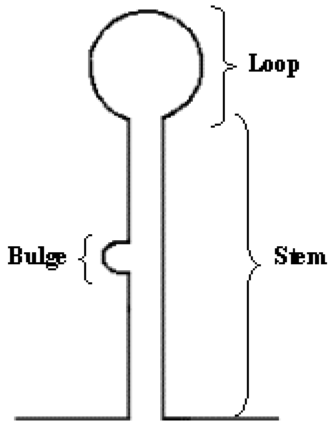
b) 

Fig 18. **a)** The dialog lets the user specify what a loop should look like. **b)** The hairpin structure.

Min stem length – Specifies the minimum length of the stem.
Max stem length - Specifies the maximum length of the stem.
Min loop length – Specifies the minimum length of the loop.
Max loop length - Specifies the maximum length of the loop.
Loop consensus – Lets the user specify a consensus for the loop. This is an optional specification. Note: If a consensus is specified, the loop lengths are overridden.
Number of loops – Specifies the number of loops of this type that must be present in the sequence for it to pass.
Impose GC-constraint – If this box is checked, the base pair in the stem closing the loop must be a GC base pair.
Must have bulge – If this box is checked, an A-bulge must be present in the stem.
Can have bulge – If this box is checked, an A-bulge may or may not be present in the stem.
Note: If none of the above boxes is checked, no A-bulge may be present in the stem.
Min bulge length – Specifies the minimum length of a bulge.
Max bulge length – Specifies the maximum length of a bulge.

Min pre-bulge length – Specifies the minimum number of base pairs between the bulge and the loop.
Max pre-bulge length – Specifies the maximum number of base pairs between the bulge and the loop.
Min. post-bulge length – Specifies the minimum number of base pairs after the bulge.
Max. post-bulge length – Specifies the maximum number of base pairs after the bulge.

The ORF filter

Since a functional RNA molecule does not contain any open reading frames, the ORF filter removes the sequences that contain a start codon and a termination codon.

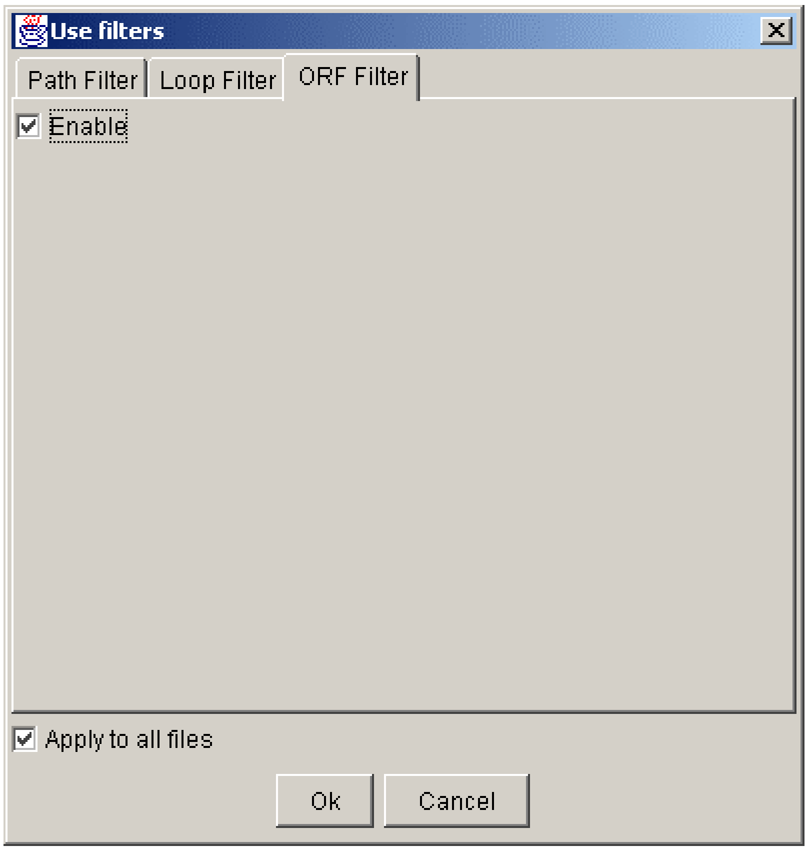


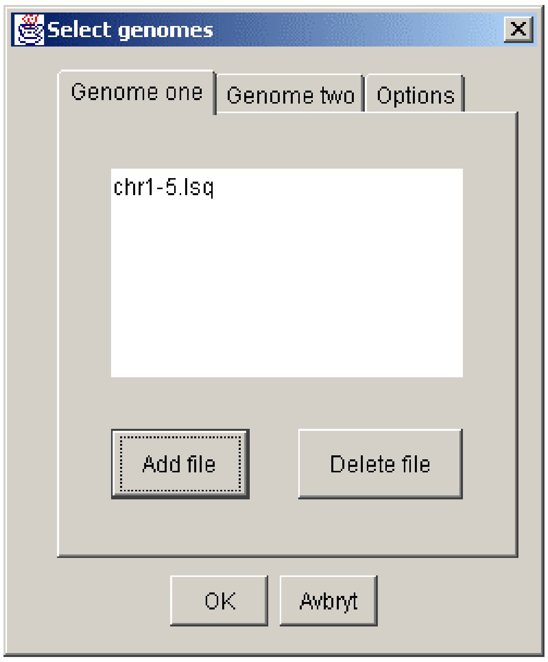
Fig 19. The ORF filter can be enabled.

Features

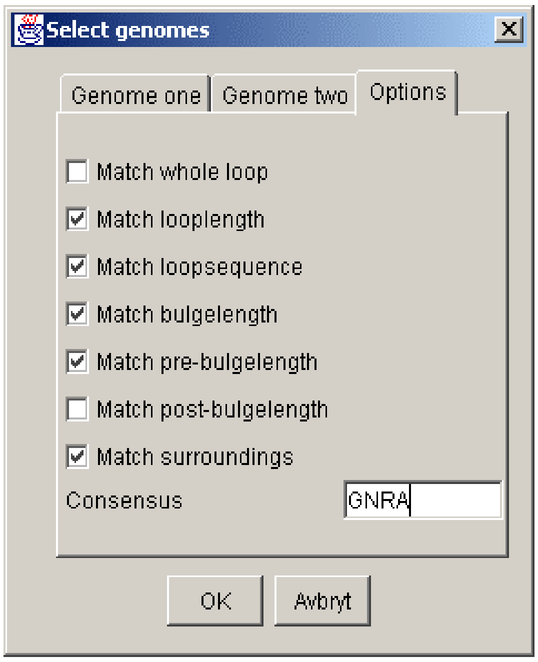
Enable – Enables or disables the filter.
Apply to all files – If this box is checked, the selected filters will be applied to all pages of results. If the box is unchecked, the selected filters will only be applied to the result page currently showing.

Structure homology

The option for structure homology lets the user compare hairpin loops from one file against loops from another file. The intention is that these files contain important structural motifs such as GNRA or UNCG tetraloops that will be compared in order to look for structural conservation in different species. The user can specify a number of constraints and options.

a)


chr1-5.lsq

b)


- ☐ Match whole loop
- ☑ Match looplength
- ☑ Match loopsequence
- ☑ Match bulgelength
- ☑ Match pre-bulgelength
- ☐ Match post-bulgelength
- ☑ Match surroundings

Consensus: GNRA

Fig 20. **a)** Added genomes are compared against each other. **b)** Options for the structure comparison can be specified.

Under the first tab, labelled **Genome one**, multiple files can be added or deleted. These files will be compared to those added under the second tab, **Genome two**. None of the files added under the same tab will be compared against each other. Under the last tab, **Options**, a number of alternatives for the comparison can be specified.

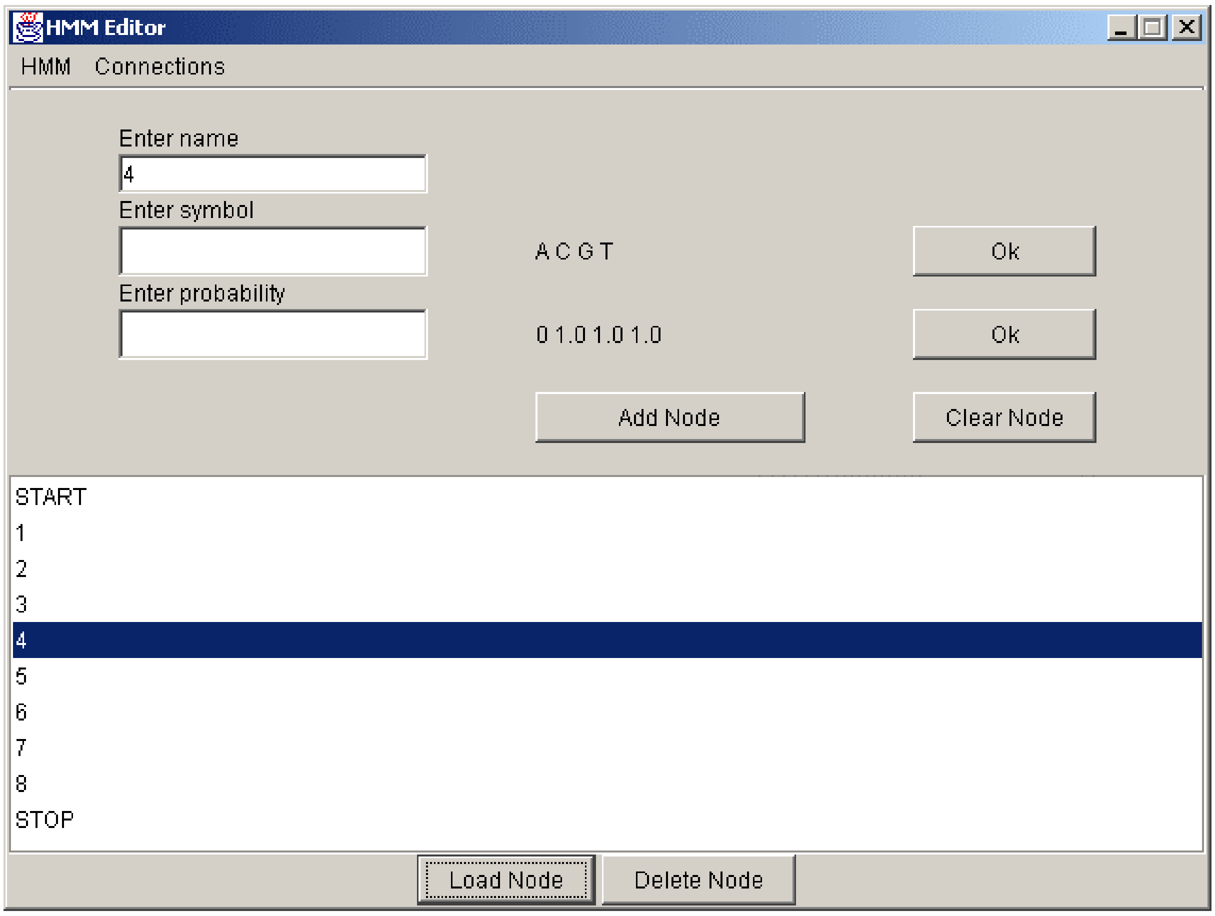
Match whole loop – If checked, the complete hairpin structures must be identical.
Match loop length – If checked, the lengths of the loops must be identical.
Match loop sequence – If checked, the consensuses of the loops must be identical.
Match bulge length – If checked, the length of a bulge in the stem must be identical.
Match pre-bulge length – If checked, the number of base pairs between a bulge and the loop must be identical.
Match post-bulge length – If checked, the number of base pairs after a bulge must be identical.
Match surroundings – If checked, at least one more loop in the first sequence must match another loop in the other sequence and the distance between the two loops must be approximately the same.
Consensus – If a consensus is specified, it must match the consensus of the loop in the first sequence.

Results will be presented with a header that indicates what loops in the first file have matched loops in other files. Below the header follows the sequences that contain the matching loops and all loops contained within them are displayed.

In order to simplify the design of hidden Markov models for the program, some editors have been programmed that enables the user to create models through a graphical interface. The output from the editors is parameter files that are on the correct format for the program to read.

The HMM editor

The HMM editor lets the user define the nodes used in the model and set the connections between them. A screenshot of the editor can be seen in fig. 21.



4

A C G T

0 1.0 1.0 1.0

START
1
2
3
4
5
6
7
8
STOP

Fig 21. Screenshot of the HMM Editor

A hidden Markov model is constructed by first defining the nodes and then connecting them. Each node has to have a unique name and there has to be a start and a stop node. The user has to make sure that there exists a path from the start node to the stop node, otherwise the model cannot be used.

Main window

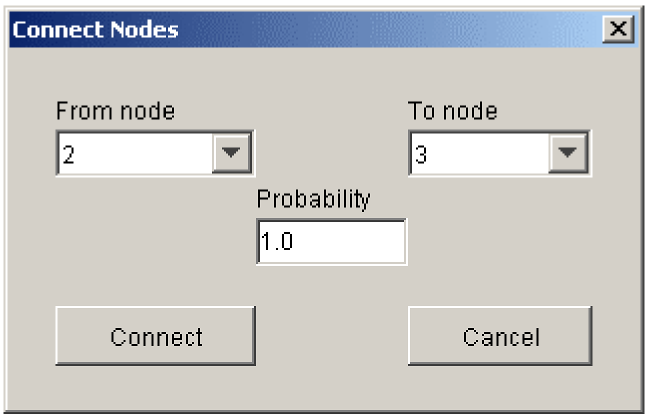
Name field – Text field to specify a name for the node
Symbol field – Text field for entering the symbols to be emitted from the node.
Probability field – Text field for entering the probabilities linked to the symbols.
Add node – Adds the current node to the HMM and clears the fields.
Clear node - Clears all fields in the current node.
Added nodes – Here the nodes that have been added to the model are displayed.
Load node – Loads the parameters from the selected node for viewing or editing.
Delete node – Deletes the selected node from the HMM, removing all connections to this node.

HMM menu

Load HMM - Opens a dialog that lets the user specify a file with a previously defined HMM that can be edited.
Save HMM - Opens a dialog where the user can specify a file to save the HMM in. Note: all changes will be lost unless saved to a file.
Reset - Clears all nodes from the current HMM.
Exit – Exits the editor.

Connections menu

Set connections - Opens up a new window that lets the user define the connections between states, see fig. 22.



From node: 2
To node: 3
Probability: 1.0

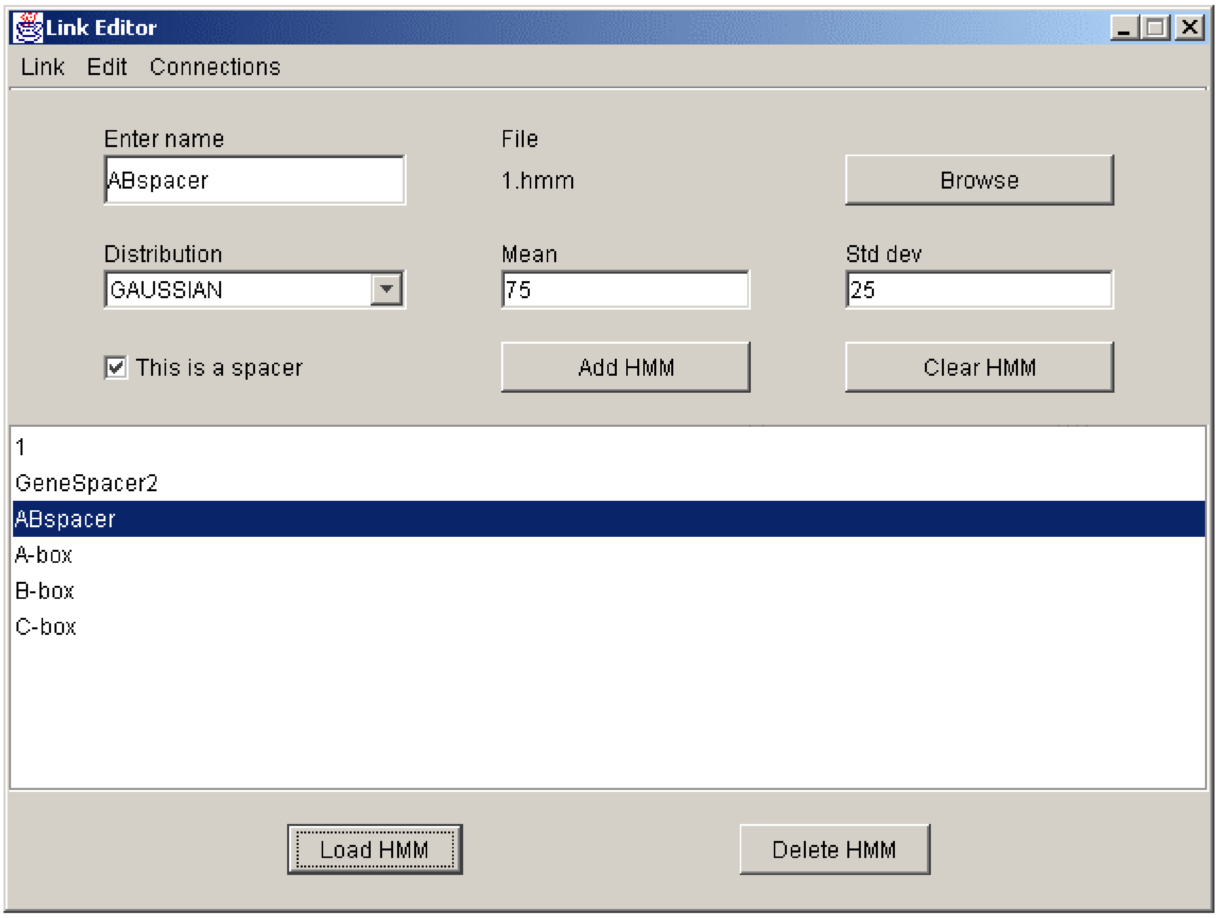
Fig 22. Dialog box where the connections between nodes can be defined. The drop-down menus display the connection to be set and the text field is for entering the transition probability.

The drop-down list to the left lets the user select the node to make a transition from and the list to the right selects the node to make a connection to. In the probability field the probability for the transition is entered. Note that the button **connect** must be pressed in order to make the connection.

Remember to save the HMM before exiting the editor. The file should be saved in the “Models”-directory and have an .hmm extension.

The link editor

In order to construct a model for a whole genome, many individual hidden Markov models can be connected in closed loops. An editor that facilitates this was programmed. Connections between the HMMs and additional parameters have to be specified, in addition, length distributions for the HMMs can be specified. The output from this editor is parameter files where this linkage of models is defined. A screenshot of the editor is shown in fig. 23.



Enter name: ABspacer
File: 1.hmm
Distribution: GAUSSIAN
Mean: 75
Std dev: 25
[✓] This is a spacer

1
GeneSpacer2
ABspacer
A-box
B-box
C-box

Fig 23. Screenshot of the link editor.

There are many similarities to the HMM editor but also some important differences. As in the case with the nodes in the hidden Markov model, each model has to have a unique name. However, the same file defining the model can be used for multiple HMMs. There is also a checkbox stating whether the model is for a spacer or not. It is strongly recommended to check this box if the model is a spacer since it will save a lot of calculations.

Main window

Name field – Text field where the name for the current HMM must be specified.
Browse - Brings up a dialog where the file defining the HMM to be used is selected.
Distribution - Sets the type of length distribution to be associated with this HMM.
Mean - Sets the mean value for the distribution.

Standard deviation - Sets the standard deviation for the distribution. Lengths possible for this HMM will span three standard deviations up and down from the mean value.
Spacer checkbox - If the HMM models a spacer sequence, this box should be checked in order to boost the performance of the program.
Add HMM - Adds the current HMM to the model and clears all fields.
Clear HMM – Clears all fields in the current HMM.
Added HMMs – The HMMs that have been added to the model are displayed here.
Load HMM - Loads the selected HMM's parameters for viewing or editing.
Delete HMM - Deletes the selected HMM from the linker, removing all references to it.

Link menu

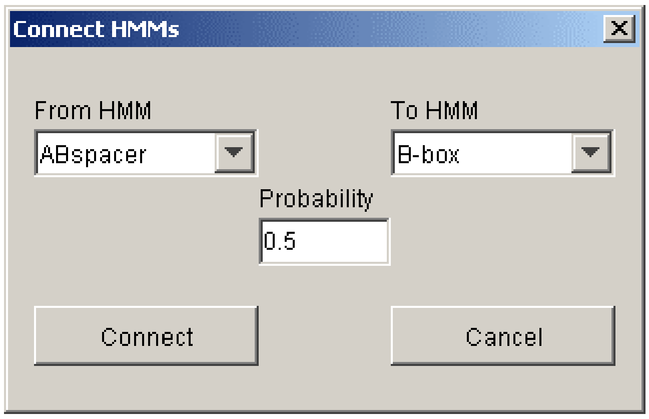
Load link - Brings up a dialog where a previously saved linker can be loaded.
Save link - Brings up a dialog where the user can specify the location to save the linker.
Reset- Clears the current linker.
Exit - Exits the editor.

Edit menu

Edit HMM – Opens the HMM Editor.

Connections menu

Set connections - Opens up a new window that lets the user define the connections between the HMMs, see fig. 24.



From HMM: ABspacer
To HMM: B-box
Probability: 0.5

Fig 24. Screenshot of the connections dialog

The drop-down list to the left lets the user select the HMM to make a transition from and the list to the right selects the HMM to make a connection to. In the probability field the probability for the transition is entered. Note that the button **connect** must be pressed in order to make the connection.

Do not forget to save the model before exiting the editor. The file should be saved in the “Models”-directory and have an .lnk extension.

Future work

The program will be further enhanced and primarily the loop filter will be further developed. The intention is to increase the number of features possible to scan for in the stem. Some subroutines currently implemented in Java will be rewritten in C++ and compiled into a dynamic link library (dll) in an attempt to boost performance of the filters. Searches will be performed on various genomes in order to obtain candidate genes and these will be experimentally confirmed.

References

[1] M. Hildebrandt and W. Nellen, (1992) "Differential antisense transcription from the Dictyostelium EB4 gene locus: implications on antisense-mediated regulation of mRNA stability," *Cell*, **69**, 197-204.

[2] T. V. Sharp, M. Schwemmle, I. Jeffrey, K. Laing, H. Mellor, C. G. Proud, K. Hilse, M. J. Clemens, (1993) "Comparative analysis of the regulation of the interferon-inducible protein kinase PKR by Epstein-Barr virus RNAs EBER-1 and EBER-2 and adenovirus VAI RNA," *Nucl. Acids Res.*, **21**, 4483-90.

[3] J. Luirink and B. Dobberstein, (1994) "Mammalian and Escherichia coli signal recognition particles," *Mol. Microbiol.* **11**, 9-13.

[4] L. A. Kirsebom, (1995) "Rnase P — a 'Scarlet Pimpernel'," *Mol. Microbiol.* **17**, 411-20.

[5] H. D. Madhani and C. Guthrie, (1994) "Dynamic RNA-RNA interactions in the spliceosome," *Annu. Rev. Genet.* **28**, 1-26

[6] S. R. Eddy, (1999), "Noncoding RNA genes," *Curr. Opin. Genet. Dev.* **9**, 695-699.

[7] L. Argaman, R. Hershberg, J. Vogel, G. Bejerano, E. G. Wagner, H. Margalit and S. Altuvia, (2001) "Novel small RNA-encoding genes in the intergenic regions of Escherichia coli," *Curr. Biol.* **26,** 941-50.

[8] L. R. Rabiner, (1989) "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," *Proc. IEEE*, **77**, 257-86.

[9] D. W. Mount, (2001) "Bioinformatics: sequence and genome analysis," New York: *Cold Spring Harbor Laboratory Press*.

[10] L. E. Baum and J. A. Egon, (1967) "An inequality with applications to statistical estimation for probabilistic functions of a Markov process and to a model for ecology," *Bull. Amer. Meteorol. Soc.*, **73**, 360-63.

[11] L. E. Baum and G. R. Sell, (1968) "Growth functions for transformations of manifolds," *Pac. J. Math.*, **27**, 211-27.

[12] A. J. Viterbi, (1967) "Error bounds for convolutional codes and an asymptotically optimal decoding algorithm," *IEEE Trans. Informat. Theory*, **IT-13**, 260-69.

[13] G. D. Forney, (1973) "The Viterbi algorithm," *Proc. IEEE*, **61**, 268-78.

[14] C. B. Burge and S. Karlin, (1998) "Finding the genes in genomic DNA," *Curr. Opin. Struct. Biol.*, **8**, 346-54.

[15] S. E. Levinson, L. R. Rabiner and M. M. Sondhi, (1983) "An introduction to the application of the theory of probibalistic functions of a Markov process to automatic speech recognition," *Bell Syst. Tech. J.*, **62**, 1035-74.

[16] E. P. Geiduschek and G. A. Kassavetis, (2001) "The RNA Polymerase III Transcription Apparatus," *J. Mol. Biol.*, **310**, 1-26.

[17] The RNA World Website, http://www.imb-jena.de/RNA.html (2002)

[18] J. D. Thompson, D. G. Higgins and T. J. Gibson, (1994) "CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice," *Nucl. Acids Res.*, **22**, 4673-80.

[19] A. L. Larsson, (2002) "Development of model for finding of functional polymerase III transcribed RNA-products," *Master thesis, Uppsala University*.

[20] T. Hermann and D. J. Patel, (1999) "Stitching Together RNA Tertiary Architectures," *J. Mol. Biol.*, **294**, 829-49.

[21] T. Hermann and D.J. Patel, (2000) "RNA bulges as architectural and recognition motifs," *Structure*, **8**, R47-R54.

[22] H. A. Heus and A. Pardi, (1991) "Structural Features That Give Rise to the Unusual Stability of RNA Hairpins Containing GNRA Loops," *Science*, **253**, 191-93.

[23] P. Nissen, J. A. Ippolito, N. Ban, P. B. Moore and T. A. Steitz, (2001) "RNA tertiary interactions in the large ribosomal subunit: The A-minor motif," *PNAS*, **98**, 4899-4903.

[24] J. Skansholm, (1999) "Java direkt," Lund: *Studentlitteratur*.

[25] J. Skansholm, (1996) "C++ direkt," Lund: *Studentlitteratur*.

[26] M. R. Paule and R. J. White, (2000) "Transcription by RNA polymerases I and III," *Nucl. Acids Res.*, **28**, 1283-98.

[27] B. Lewin, (2000) "Genes VII," New York: *Oxford University Press and Cell Press*.

[28] S. F. Altschul, W. Gish, W. Miller, E. W. Myers and D. J. Lipman, (1990) "Basic local alignment search tool," *J. Mol. Biol.*, **215**, 403-10.

[29] NCBI BLAST Home Page, http://www.ncbi.nlm.nih.gov/blast/ (2002)

[30] W. R. Pearson and D. J. Lipman, (1988) "Improved tools for biological sequence comparison," *Proc. Natl. Acad. Sci. U S A*, **85**, 2444-48.