Fig. 1. Overview of the complete motif discovery scheme, which is composed of two major parts. In the first part, structural elements that appear in a maximal number of input RNAs are selected to initialize a model, and in the second part this model is improved using the EM algorithm.

1 GENERAL

This work suggests an efficient computational method, based on stochastic context free grammar (SCFG) representation of RNA motifs, for identifying short RNA motifs from a set of unaligned input RNAs (Fig. 1). This motif discovery scheme takes as input a set of unaligned RNAs, and tries to identify a common motif in these RNAs. For each input RNA, both the sequence and the secondary structure are specified. The motif prediction algorithm is composed of two main parts: in the first part, several heuristics are employed to identify good motif candidates, and in the second part, these candidates are refined using an EM algorithm. See detailed description of the motif discovery algorithm in the next section.
2 THE PROBABILISTIC MODEL

2.1 Stochastic Context Free Grammars (SCFGs)

Stochastic context free grammars (SCFGs) are a class of probabilistic models, proposed for modeling RNA sequence and structure, on which we base our model [14, 15].

Formal grammars are generative models consisting of a set of terminal symbols (the alphabet), a set of non-terminal symbols and a set of production rules. The production rules are used to produce different strings of alphabet symbols, by successive replacement of left side expression of a production rule with the right side expression. Using the production rules in a specific order to generate a specific string of alphabet symbols is called derivation of that string. The language of the grammar is defined as the set of all strings that can be derived by the grammar.

The expressive power of a grammar depends on the grammar’s production rules. There are various grammar types, each specify different set of restrictions for the production rules. In 1956 Noam Chomsky introduced a hierarchy of formal grammars that ranks grammar types by their expressive power [22].

Context Free Grammars (CFGs) are the second level of this hierarchy. The production rules of a CFG are of the form $X \rightarrow \alpha$ where $X$ is a non-terminal symbol, and $\alpha$ is a string of both terminal and non terminal symbols. The key feature of CFGs that makes them highly suitable for modeling RNAs is their ability to efficiently model nested, long-distance pairwise correlations in strings of symbols, which is required for modeling the long-range base-pairing interactions that occur in RNA secondary structures. However, to model pseudoknots within the RNA structure, a language of a higher level of the hierarchy is required.

Stochastic Context Free Grammars (SCFGs) are CFGs in which a specific probability is associated with each production rule of the grammar. As a result each derivation is also associated with a probability, which is the product of the probabilities of all the production rules used in the derivation. SCFGs can therefore be used as generative models that can generate both the structure and sequence of an RNA molecule and associate a probability with each such instance. SCFGs have been used for a variety of tasks, including RNA structure prediction [23], structure based database search [24], identification of non-coding RNAs [25] and simultaneous RNA folding and alignment [26].

There are grammars that can generate specific instances with more than a single derivation. Such grammars are called ambiguous grammars. It was shown that ambiguity can be problematic in some SCFG based algorithms and might lead to wrong results [27].
Therefore, it is better to design *unambiguous* grammars, for which each instance in the output space can be produced by a *single* derivation.

SCFGs can be used to describe features of both sequence and structure common to a set of RNAs, and by which produce a probabilistic model of a family of related RNAs. Using SCFGs to model RNAs is somewhat similar to the use of HMMs in modeling sets of DNA sequences [28]. Different methods for building such SCFGs from a set of input RNAs were developed. Covariance Models (CMs) are one particular convention for constructing SCFGs from an RNA multiple alignment [14]. These models were shown to have great utility in RNA similarity searching applications, such as identification of novel tRNA genes [29]. The Rfam database [21] describes various families of RNAs with common sequence and structure elements by modeling each family using a CM constructed from an alignment of known members of the family. These CMs are then used to identify new members of the family in large genome sequences.

After building a CM from a given alignment, the model's parameters can be optimized by using the EM algorithm. Iterating between an E-step that essentially counts the usage of each grammar rule in all possible derivations of the given sequence by the existing model (considering all possible secondary structures for the input sequences), and an M-step that estimate the model's parameters based on these counts, guarantees improvement of the model's parameters to a local maxima. Moreover, by finding the most probable derivation of each RNA by a given CM, a multiple alignment of RNAs which is based on both sequence and structure considerations can be constructed. This gives rise to algorithms that can learn a CM for a set of unaligned RNAs without using pre-defined multiple alignment, by iteratively building an alignment and using it to learn a CM [14].

### 2.2 Data instance representation

The motif discovery algorithm takes as input a set of unaligned RNAs that are assumed to share a motif. For each input RNA, both the sequence and the secondary structure should be specified.

The literature describes various Context Free Grammars (CFGs) that generate the space of all legal RNAs (an RNA sequence and a matching secondary structure). The given input instances are obviously part of this space, and therefore can be derived by such CFG. Here we use a specific unambiguous CFG to represent the input instances:
given that $x \in \{a, u, c, g\}$ and $y_1y_2 \in \{au, ua, cg, gc, gu, ug\}$,

$$S_0 \rightarrow y_1P_{y_2} \mid xL \mid R x \mid B$$

$$L \rightarrow y_1P_{y_2} \mid xL \mid R x \mid E$$

$$P \rightarrow y_1P_{y_2} \mid xL \mid R x \mid E \mid B$$

$$R \rightarrow y_1P_{y_2} \mid R x$$

$$E \rightarrow \epsilon$$

$$B \rightarrow S_0S_1$$

$$S_1 \rightarrow y_1P_{y_2} \mid xL$$

The non terminals $P, L,$ and $R,$ correspond to a nucleotide pair, a single leftwise (5') nucleotide, or a single rightwise (3') nucleotide, respectively. The non terminals $S$ and $E$ correspond to the start and end of a stem in the structure, respectively. The non terminal $B$ represents bifurcation, a split in the underlying secondary structure into two separate stems.

For example, the data instance in Fig. 2b is produced by the (single) derivation:

$$S_0 \rightarrow B; B \rightarrow S_0S_1$$

$$S_0 \rightarrow aL; L \rightarrow Rg; R \rightarrow gPc; P \rightarrow uPa; P \rightarrow cL; L \rightarrow uL; L \rightarrow gL; L \rightarrow \epsilon; E \rightarrow \epsilon$$

$$S_1 \rightarrow cPg; P \rightarrow aL; L \rightarrow gL; L \rightarrow uL; L \rightarrow E; E \rightarrow \epsilon$$

Each input instance is represented by its (single) derivation by this grammar.

A derivation can be represented as an ordered tree $S,$ with nodes $S_0, \ldots, S_M,$ where each node $S_i$ is associated with a structure variable $h_i,$ and a sequence variable $o_i.$ Each structure variable $h_i \in \{P, L, R, S_0, S_1, E, B\}$ represents a single non-terminal symbol of the derivation, and corresponds to the secondary structure of the RNA. Each sequence variable $o_i \in \{a, u, c, g, au, ua, cg, gc, gu, ug, \epsilon\}$ represents all the terminal symbols produced from a specific grammar rule in the derivation, and corresponds to the sequence of the RNA. Because the grammar is unambiguous, both the tree structure and each variable’s value are determined uniquely by the input RNA. For example, the tree for the derivation of the data instance in Fig. 2b is shown in Fig. 2c. This tree representation is used to describe each input RNA in the algorithm.

An RNA is not represented by a single derivation tree, but by a list of such trees, each can be different and can only describe a specific part of the complete RNA sequence. This representation, allows to restrict the algorithm search space, yet not to a single secondary structure, but to a more representative group of structures. It also allows to specify a different number of structures for different parts of the molecule, as there are
Fig. 2. Example of a simple CM and an alignment of an RNA instance to it. (a) Graphical representation of a CM ordered tree, including structure (blue) and sequence variables (green). Transition and emission probabilities are described in tables, listing the probability of the child variable (column) given the parent variable (row). Cells colored in gray correspond to entries constrained to have zero probabilities. (b) Diagram of an RNA instance. (c) Graphical representation of the RNA instance ordered tree, including structure (pink) and sequence variables (gray). Each branch of the tree corresponds to a stem in the secondary structure. (d) Dynamic programming table for aligning the shown model and data instance. Rows correspond to model states and columns to data instance states (arrows show the parent-child relationship between the variables). Table filling order is indicated by black arrows, and red arrows show the best alignment path.

Parts in which the structure is more variable than in others. The motif search algorithm will attempt to find the best occurrence of a motif in one of the trees in this list.

2.3 RNA motif representation

An RNA motif is a formal way to describe a set of both structural and sequential local features common to a set of RNAs. Such an RNA motif is represented using a probabilistic model which is based directly on covariance models (CMs), yet introduces changes that make it better suited to the problem of discovering short RNA motifs in long unaligned RNAs. Similarly to CMs, the new representation uses SCFGs to define a joint probability distribution over the space of RNA sequence and structure.
A model can be viewed as a sequence of states, that each RNA produced by the model must pass through. At each state, there is a probability over the structure and sequence expected to be produced at that state. These states are nodes of an ordered tree $T$ with $N$ nodes $T_0, \ldots, T_N$. The tree structure defines the relationships between the model nodes. Each node $T_i$ (except the root node) has a single parent node and at most two child nodes. Each state node, $T_i$, is associated with two discrete-valued random variables: the structure variable $H_i$ with domain $\{P, L, R, S, E, B\}$, representing the part of the RNA structure produced by the state, and the sequence variable $O_i$ with domain $\{a, u, c, g, au, ua, cg, gc, gu, ug, \epsilon\}$, representing the part of the RNA sequence produced by the state. For example, the tree in Fig. 2a describes an RNA model.

Each random variable is associated with a probability distribution over its domain, which is given as conditional probabilities. The structure random variable probability is conditioned over the parent node’s structure variable $P(H_i \mid Pa(H_i))$, and is referred to as the state transition probability. The sequence random variable probability is conditioned over the node’s structure variable $P(O_i \mid H_i)$, and is referred to as the symbol emission probability.

These conditional probabilities are restricted based on both the model tree structure and the CFG given in section 2.2. The root of the model is deterministically assigned the value $H_0 = S, O_0 = \epsilon$. For the rest of the nodes, the probabilities over the random variables must agree with the CFG and with the tree structure. That is, $P(H_i = x \mid y) > 0$ if and only if the rule $y \rightarrow x$ exists in the grammar, and all rules of the form $x \rightarrow X$ agrees with the tree structure in the sense that the number of non terminals in $X$ must match the number of child nodes $T_i$ has in the tree.

These restrictions allow to associate such a tree structure with an SCFG, that contains a production rule for each pair of non-zero transition and emission probabilities in each node of the model tree. The probability of such a production rule is the product of these transition and the emission probabilities. For example, the following grammar describes the model in Fig. 2a:
Each of these production rules is associated with a probability $\text{Prob}(X_i \rightarrow Y_{i+1})$ where $X, Y \in \{P, L, R, S, E, B\}$, such that $\sum_{Y \in \{P, L, R, S, E, B\}} \text{Prob}(X_i \rightarrow Y_{i+1}) = 1$.

This grammar is unambiguous, as a legal pair of sequence and structure can be produced by a single derivation. Yet, this grammar does not produce the entire space of legal input RNAs, but a rather limited collection of RNAs. For example, the RNA in Fig. 2b cannot be derived from this grammar. In order to allow the model grammar to produce the entire space of legal input RNAs, structural deletions and insertions in the input RNA relative to the model must be modeled. This requires adding several more production rules to the model grammar.

For modeling deletes:

$$
P_i \rightarrow D_{i+1}; \quad L_i \rightarrow D_{i+1}; \quad R_i \rightarrow D_{i+1}; \quad S_i \rightarrow D_{i+1}
$$

$$
D_i \rightarrow y_1 P_{i+1} y_2 \mid x L_{i+1} \mid R_{i+1} y \mid E_{i+1} \mid B_{i+1} \mid D_{i+1} \mid D_{\text{right-child}(i)}$$
For modeling inserts:

\[ \mathcal{P}_i \rightarrow \mathcal{P}_{i+1}; \quad \mathcal{L}_i \rightarrow \mathcal{L}_{i+1}; \quad \mathcal{R}_i \rightarrow \mathcal{R}_{i+1}; \quad \mathcal{S}_i \rightarrow \mathcal{S}_{i+1} \]

\[ \mathcal{I}_i^X \rightarrow y_1 \mathcal{P}_i y_2 \mid x\mathcal{L}_i \mid \mathcal{R}_i x \mid \mathcal{E}_i \mid \mathcal{B}_i \]

\[ \mathcal{I}_{i,0}^S \rightarrow y_1 \mathcal{P}_i y_2 \mid x\mathcal{I}_i^L \mid \mathcal{I}_i^R x \mid \mathcal{I}_B \]

\[ \mathcal{I}_i^L \rightarrow y_1 \mathcal{P}_i y_2 \mid x\mathcal{I}_i^L \mid \mathcal{I}_i^R x \mid \mathcal{I}_E \]

\[ \mathcal{I}_i^R \rightarrow y_1 \mathcal{P}_i y_2 \mid \mathcal{I}_i^R x \]

\[ \mathcal{I}_E \rightarrow \epsilon \]

\[ \mathcal{I}_i^B \rightarrow \mathcal{I}_{i,0}^S \mid \mathcal{I}_{i,1}^S \]

\[ \mathcal{I}_{i,1}^S \rightarrow y_1 \mathcal{P}_i y_2 \mid x\mathcal{I}_i^L \]

Note that these production rules impose some restrictions on which states can be deleted, and where inserts are allowed. By adding these rules to the grammar it can derive any legal input RNA, for example by first inserting all the input RNA and then deleting all the states of the model. Yet these new rules turns it into an ambiguous grammar, as it allows to produce the same RNA in different ways, for example by each time using the insert rules to produce other parts of it.

This ambiguity does not exist, however, if we view the grammar as producing in addition to an RNA sequence and structure, also its alignment to the model. This alignment is a vector, specifying for each unit of the structure and the sequence (a single nucleotide or a pair) whether it was produced by the node’s probabilities (\( \mathcal{M} \)) or was inserted according to background probabilities (\( \mathcal{I} \)). Nodes that were deleted are associated with the value \( \mathcal{D} \) in the alignment vector.

Going back to the tree representation of the model, modeling inserts and deletes means allowing tree nodes to be deleted or inserted. This is done by adding the values \( \mathcal{D}, \mathcal{I}^P, \mathcal{I}^L, \mathcal{I}^R, \mathcal{I}^S_0, \mathcal{I}^S_1, \mathcal{I}^E, \mathcal{I}^B \) to the domain of the structure random variables. The assignment \( H_i = \mathcal{D} \) means that state \( \mathcal{T}_i \) of the model does not produce any sequence or structure. The assignment \( H_i = \mathcal{I}^x \) allows producing more than a single sequence and structure value at state \( \mathcal{T}_i \) of the model, all of which, except the last, are produced not by the model probabilities but according to a predefined background probability. The following probabilities are added to the state transition probabilities of the model:

\[ P(H_i = \mathcal{D} \mid X) = P(\text{delete}) \]

\[ P(H_i = \mathcal{I}^X \mid Y) = P(\text{gap}_\text{open}) \cdot P_{BG}(X \mid Y) \]

\[ P(H_i = \mathcal{I}^x \mid \mathcal{I}^y) = P(\text{gap}_\text{extend}) \cdot P_{BG}(X \mid Y) \]
These probabilities will be positive if the grammar allows the transition (as was explained before). The gap open, gap extend and delete probabilities are pre-defined parameters of the model.

To allow alignment of short local motifs to longer data instances, inserts at the beginning of the model has no gap open and gap extend costs \( P(\text{gap open}) = P(\text{gap extend}) = 1 \). This way, most of the data instance states will be aligned as inserts at the beginning of the model (and will be ignored), and only a small portion of the instance tree will match the rest of the model states.

To summarize, an RNA motif is modeled as a pair \((T, \theta)\) of an ordered tree \(T\) and a set of parameters \(\theta\). Each node \(T_i\) of the tree \(T\) contains two random variables, a structure variable \(H_i\) and a sequence variable \(O_i\). The model’s parameters include a state transition probability, \(P(H_i \mid Pa(H_i))\), and a symbol emission probability, \(P(O_i \mid H_i)\), for each node \(i\) in \(T\). This tree representation can be easily converted into an SCFG. Except for pseudo-knots, such a tree representation is capable of capturing all the pairwise interactions of an RNA secondary structure.

### 2.4 Initializing a model

As in any application of probabilistic models, it is important to initialize a model with a good starting point.

As was stated before, the input for the motif discovery algorithm consists of a set of unaligned RNAs that are assumed to share a motif. For each input RNA, both the sequence and the secondary structure are specified. To reduce the search space in the initialization phase, the sequence of the input RNAs is ignored so the search is done only in the space of the given secondary structures.

An RNA structure can be represented in "bracket notation", in which for each paired bases, the first base is represented as an open bracket "("; and the second base as a closed bracket ")"; and unpaired bases are represented as dots ".". The RNA in Fig. 2b for instance, is represented in this notation as \.((...)).(...). Almost every such structure contains within it simpler sub-structures of varying lengths. For example, the RNA in Fig. 2b, contains within the following substructures:

\[
\{((...))(....), (...), .((...))., .((...)), ((...)), .((...))., \ldots, \ldots, .\}
\]

Each such sub-structure is referred to as a feature. Note that each input RNA may generate multiple features, and conversely each feature may be associated with one or more input RNAs.
At the first stage of the initialization procedure, an exhaustive extraction of all the 15 – 70 nucleotide long features in the input secondary structures is performed, resulting in a large collection of features. This feature set is reduced using two filtering criteria. First, simple features, that are likely to appear in any set of RNA sequences are removed, by comparing the number of times each feature appears in the input set to a background distribution specifying the expected number of appearances for a feature in a random set of structures. Next, note that when extracting all features in an input set, each feature usually contains within it other, smaller, features. A feature that always appears as part of the same, larger, feature, will also be removed from the feature set.

The features are ranked by the number of input RNAs they appear in, such that features that match a larger number of inputs have a higher rank. A certain flexibility in the match is allowed, by allowing to match loops and stems of close, but not identical, size. Each of the top ranking features is used to initialize a separate model.

Since a model is composed of a pair \((T, \theta)\), initialization should specify both a tree structure \(T\) and an initial assignment to the parameters \(\theta\). The structures of the selected feature is used to initialize the tree structure. The parameters for the model are then estimated using a single M-step (see details below) over all the input sequences that match the selected feature.

### 2.5 Learning the model parameters

The problem of learning the parameters of a model can be formulated as follows. We are given a dataset \(D = \{D_1, \ldots, D_K\}\) of \(K\) i.i.d data instances (represented as described above), and an initial model \(C = (T, \theta)\) built by the initialization procedure described above, and our goal is to find the parameters \(\theta\) of the model \(C\) that maximizes the log-likelihood of the input instances:

\[
\log \text{likelihood}(D) = \log P(D \mid (T, \theta)) = \sum_{i=1}^{K} \log P(D_i \mid (T, \theta))
\]

By simply replacing the sum in this equation with a maximization operator, the best alignment of an instance \(D_i\) to the model \(C\) can be calculated (the rest of the algorithm is unchanged).

Calculating \(P(D_i \mid (T, \theta))\) requires going over over all the productions that generate \(D_i\) and summing their probabilities. In most standard CM applications, each \(D_i\) represents only the sequence of an RNA, and therefore there is a much large number of productions that can generate it, as each might represent a different structure. Here, however, \(D_i\) consists of both sequence and structure, and therefore only productions that generate them both are considered. This restriction reduce the number of relevant productions
significantly, and results in a more efficient likelihood computation of $O(L^2)$ time (where $L$ is the length of the sequence of $D_i$), compared with the $O(L^3)$ time required for this computation in a standard CM application.

Unfortunately, as the model is ambiguous with respect to producing RNA sequence and structure pairs, there is usually more than one production that generates a specific data instance $D_i$. These productions, however, differ only in the underlying alignment vector, specifying for each item produced in the derivation, whether it was matched ($M$) or inserted ($I$) relative to the model, and which states of the model were deleted ($D$). As a result of this ambiguity, although the basic likelihood computation can be done efficiently, there is no closed form solution for finding the assignment to the model parameters $\theta$ that maximize the likelihood function, and we must resort to heuristics.

The EM algorithm is one standard heuristic for this task. Starting from an initial assignment to the parameters $\theta$, EM alternates between an expectation step (E-step), of computing the expected values for the hidden variables (in this case - the value of each of the model’s structure and sequence variables) given the current assignment of the parameters, and a Maximization step (M-step), of maximizing the model parameters relative to these expectations. Since each EM iteration improves the likelihood, the algorithm is guaranteed to converge to a local maximum of the likelihood function.

**Notation:** In order to describe the algorithm in detail, the following notation should be introduced:

1. $Pa(i)$ represents the parent node of node $i$ in the model or instance tree. Each tree node has a single parent, except the root node which has no parents. For most nodes $Pa(i) = i - 1$, except for the right child of a bifurcation node.

2. $Ch(i)$ represents the list of all children of the node $i$ in the model or instance tree. Most nodes in the tree have a single child in which case $Ch(i) = i + 1$, but there are specific nodes (bifurcation nodes) that have two children and nodes with no children at all (end nodes).

3. $Si(i)$ represents the list of all sibling nodes of node $i$. The list is non-empty only for nodes that immediately follow a bifurcation.

4. $Br(i)$ represents the list of all nodes in the sub-tree rooted at the sibling node of node $i$. If node $i$ has no siblings this list is empty.

5. $D_k(x, y)$ is the list of all positions $j$ in $D_k$ tree representation such that $h_j = x$, and $h_{Pa(j)} = y$. 
6. Probabilities will be noted as:

\[
P(T_i = \langle x, o, \mathcal{M} \rangle \mid y) = P(H_i = x \mid y) \cdot P(O_i = o \mid x)
\]

\[
P(T_i = \langle x, o, \mathcal{T} \rangle \mid y) = P(H_i = \mathcal{T}^x \mid y)P(O_i = o \mid \mathcal{T}^x)
\]

\[
P(T_i = \langle x, o, \mathcal{D} \rangle \mid y) = P(T_i = \langle \mathcal{D} \rangle \mid y) = P(H_i = \mathcal{D} \mid y)
\]

for all \( x \in \{ \mathcal{P}, \mathcal{L}, \mathcal{R}, \mathcal{S}, \mathcal{E}, \mathcal{B} \} \), \( o \in \{ a, u, c, g, au, ua, cg, gc, gu, ug \} \) and \( X \in \{ \mathcal{M}, \mathcal{T}, \mathcal{D} \} \).

### 2.5.1 The E-step

At the E-step, the current setting of the model parameters are used to compute the expectation over the assignments for the structure variables \( H_i \) and sequence variables \( O_i \) of the model in the production of an instance.

Specifically, for every state \( i \) of the model tree we wish to compute the expected counts, across all input instances, of assigning a specific value to the structure variables \( E[H_i = x, H_{pa(i)} = y] \) and the sequence variables \( E[O_i = z, H_i = x] \). To compute these expected counts, we sum the probabilities over each data instance separately:

\[
E[H_i = x, H_{pa(i)} = y] = \sum_{k=1}^{K} P(H_i = x, H_{pa(i)} = y \mid D_k)
\]

\[
E[O_i = z, H_i = x] = \sum_{k=1}^{K} P(O_i = z, H_i = x \mid D_k)
\]

Calculating the probability terms in these formulas can be done efficiently, using a dynamic programming algorithm originally proposed by Sakakibara et al. [30], with similar structure to the more familiar "forward-backward" algorithm for learning hidden Markov models (HMMs) parameters from a set of DNA sequences [28] and "inside-outside" algorithm for learning CM parameters from a set of RNA sequences [14]. All these algorithms essentially consider all possible derivations of the input data instance by the model, and sum their probabilities. Yet, unlike the "inside-outside" algorithm, this algorithm learns the model parameters from a set of RNA instances of both sequence and structure, and is therefore more efficient, running in \( O(L^2) \) in the input sequence length rather then \( O(L^3) \) for the "inside-outside" algorithm. As stated above, the reason for the reduced running time is the need to consider less possible derivation because of the pre-defined secondary structure. The original algorithm of Sakakibara et al. had to be modified to allow for learning small local motifs from long input RNAs, and to allow deletions and insertions relative to the model. The new version of the algorithm is referred to as the "upwards-downwards" algorithm.

The intuition for the "upwards-downwards" algorithm comes from observing that the probability \( P(H_i = x, H_{pa(i)} = y \mid D_k) \) is the sum of the probabilities of all the model productions that generate \( D_k \), in which the structure variable in state \( i \) of the model takes
the value $x$ and the structure variable in the parent state takes the value $y$. Each such alignment can be decomposed into three components:

$$P(H_i = x, H_{Pa(i)} = y \mid D_k) = \sum_{j \in D_k(x,y)} U[i,j] \cdot P(H_i = x \mid y) \cdot D[Pa(i), Pa(j)]$$

the production up to model state $i$ ($U[i,j]$), the transition from the value $x$ in state $i$ to the value $y$ in its parent state ($P(H_i = x \mid y)$), and the production from the parent of state $i$ to the root ($D[i,j]$).

The only remaining question is how to compute the two types of alignments, $U[i,j]$ and $D[i,j]$. Intuitively, values for $U[i,j]$ can be computed from the leaves of the tree up towards the root, and therefore this calculation is called the "Upwards" pass. Values for $D[i,j]$ are computed from the root and down to the leaves of the tree, and thus is called the "Downwards" pass.

**The Upwards pass.** At the Upwards pass, for each state $i$ of the model tree and each state $j$ of the instance tree, we compute $U_X[i,j]$ which is the probability of all productions of the model sub-tree rooted at node $T_i$ that generate the instance sub-tree rooted at node $S_j$, given that $T_i = \langle h_j, o_j, X \rangle$, where $X \in \{M, I, D\}$.

This computation can be done efficiently using dynamic programming, by utilizing previous computations for the descendants of nodes $i$ and $j$. Thus, we use a two-dimensional matrix whose rows represent the model nodes $T_0, \ldots, T_N$, and whose columns represent the instance nodes $S_0, \ldots, S_M$ (see the table at Fig. 2d). At each cell $(i,j)$, we store a vector of $U_X[i,j]$ for all three possible $X$ values. To utilize the computations of the descendants, we fill the table starting from the bottom-right cell, moving up along each column, from the rightmost to the leftmost column, finally ending at the top-left cell. This ensures that when we reach a cell in the table, its descendants have already been computed.

The recursion equations for updating the Upwards matrix are:

**Initialization:**

\[ \forall l \in \text{leaves}(T) \forall X \in \{M, I, D\} \quad U[l,j] = \begin{cases} 1 & h_j = E \\ 0 & \text{otherwise} \end{cases} \]

\[ \forall l \in \text{leaves}(T) \quad U[M][l,j] = 0 \]

\[ \forall l \in \text{leaves}(T) \quad U[I][l,j] = 0 \]

\[ \forall l \in \text{leaves}(T) \quad U[D][l,j] = 0 \]
Induction:

\[
U_M[i, j] = \prod_{c_i, c_j \in Ch(i), Ch(j)} U_M[c_i, c_j] \cdot P(T_{c_i} = \langle h_{c_j}, o_{c_j}, M \rangle \mid h_j) +
U_I[i + 1, j + 1] \cdot P(T_{i+1} = \langle h_{j+1}, o_{j+1}, I \rangle \mid h_j) +
U_D[i + 1, j + 1] \cdot P(T_{i+1} = \langle D \mid h_j \rangle)
\]

\[
U_I[i, j] = \sum_{c_j \in Ch(j)} U_I[i, c_j] \cdot P(T_i = \langle h_{c_j}, o_{c_j}, I \rangle \mid T^{h_j}) \cdot \prod_{k \in Br(j)} P(T_i = \langle h_k, o_k, I \rangle \mid T^{h_{pa(k)}}) +
U_M[i, j + 1] \cdot P(T_i = \langle h_{j+1}, o_{j+1}, M \rangle \mid T^{h_j}) +
U_D[i, j + 1] \cdot P(T_i = \langle D \mid T^{h_j} \rangle)
\]

\[
U_D[i, j] = \sum_{c_i \in Ch(i)} U_D[c_i, j] \cdot P(T_{c_i} = \langle D \mid D \rangle) \cdot \prod_{k \in Br(i)} P(T_k = \langle D \mid D \rangle) +
U_M[i + 1, j] \cdot P(T_{i+1} = \langle h_j, o_j, M \rangle \mid D) +
U_I[i + 1, j] \cdot P(T_{i+1} = \langle h_j, o_j, I \rangle \mid D)
\]

Note that because a bifurcation node must be followed by two start nodes, the case of two children only apply when the same operator \([M, D, I]\) operate on both the node and its children.

The Downwards pass. At the Downwards pass, for each state \(i\) of the model and state \(j\) of the instance we compute \(D_X[i, j]\), the probability of all productions of the model tree excluding the descending sub-trees of \(i\) that produce the instance tree excluding the descending sub-trees of \(j\), in which \(T_i = \langle h_j, o_j, X \rangle\), where \(X \in \{M, I, D\}\).

As it turns out, the Downwards pass is not symmetrical to the Upwards pass, and thus cannot be computed independently and requires intermediate computations from the Upwards pass (unlike the symmetric forward and backward computations done for hidden Markov models). The reason for the asymmetry is that going from the leaves of the tree up (as in the Upwards pass) each node has a single parent, but going from the root down, there are nodes with more than a single child. For nodes that immediately follow a bifurcation, the Downwards pass requires the probability of both its parent node and its sibling node, but at that stage of the Downwards pass, only the probability of its parent has been computed. The sibling probability, however, was computed in the Upwards pass, and using it will allow us to perform the Downwards pass efficiently.

The same matrix as in the Upwards pass is used here too. This time, however, it is filled from the top left cell, going down in each column and finally ending at the bottom-right cell. This way, when the recursion reaches a cell, all the states upstream to it in the tree were already computed.

The recursion equations for updating the Downwards matrix are therefore:
Initialization:

\[
D_M[0, j] = \begin{cases} 
1 & j = 0 \\
0 & \text{otherwise}
\end{cases}
\]

\[D_I[0, j] = 0\]

\[D_D[0, j] = 0\]

Induction:

\[
D_M[i, j] = D_M[Pa(i), Pa(j)] \cdot P(T_i = \langle h_j, o_j, M \rangle \mid h_{Pa(j)}) \cdot \prod_{b_i, b_j \in Si(i), Si(j)} U_M[b_i, b_j] + \\
D_I[i, Pa(j)] \cdot P(T_i = \langle h_j, o_j, M \rangle \mid T^{h_{Pa(j)}}) + \\
D_D[Pa(i), Pa(j)] \cdot P(T_i = \langle h_j, o_j, M \rangle \mid D) + \\
D_M[Pa(i), Pa(j)] \cdot P(T_i = \langle h_j, o_j, Ĵ \rangle \mid h_{Pa(j)}) + \\
D_D[Pa(i), Pa(j)] \cdot P(T_i = \langle h_j, o_j, Ĵ \rangle \mid D) + \\
D_M[Pa(i), j] \cdot P(T_i = \langle D \rangle \mid D) \cdot \prod_{b_i \in Br(i)} P(T_{b_i} = \langle D \rangle \mid D) + \\
D_M[Pa(i), j] \cdot P(T_i = \langle D \rangle \mid h_j) + \\
D_I[i, j] \cdot P(T_i = \langle D \rangle \mid h_j)
\]

Similarly to the Upwards pass, because a bifurcation node must be followed by two start nodes, the case of a sibling node only apply when the same operator \(X \in \{M, D, Ĵ\}\) operate on both the node and its parent.

2.5.2 The M-step The M-step of EM optimizes the parameters relative to the expected counts computed in the E-step. Since the transition and emission probability distributions are multinomials, this optimization step is standard and has a closed form. A Dirichlet prior is used to smooth the parameter settings. The maximum aposteriori settings to the parameters, given the prior and the expected counts computed in the E-step, are calculated using the following parameter update formulas:

\[
\theta_{Hi=x|Pa(Hi)=y} = \frac{E[Hi = x, Pa(Hi) = y] + \alpha(Hi = x, Pa(Hi) = y)}{\sum_x E[Hi = x, Pa(Hi) = y] + \alpha(Hi = x, Pa(Hi) = y)}
\]

\[
\theta_{Oi=x|Hi=y} = \frac{E[Oi = x, Hi = y] + \alpha(Oi = x, Hi = y)}{\sum_x E[Oi = x, Hi = y] + \alpha(Oi = x, Hi = y)}
\]

where \(\alpha(Hi = x, Pa(Hi) = y)\) and \(\alpha(Oi = x, Hi = y)\) represent the pseudo-counts in the Dirichlet prior for entries in the transition and emission probability distributions, respectively.
Fig. 3. Examples of the graphical representation of the probabilistic RNA model. (a) a simple stem structure (b) a motif composed of two stems (c) The probability scale. For each example both the tree representation of the model and the graphical representation are shown. The probability of the structure is indicated by the grey level of the structural elements (base pair or single nucleotide), and the sequence probability is indicated by the nucleotide color.

2.6 Graphical representation of an RNA motif

The result of the learning process is a probabilistic model, specifying two probability distributions for each model state. In order to allow a more schematic view of the large number of parameters learned by the model, which will allow also a non-professional user to understand the resulting model, we provide a simplified graphical representation of a model. This representation is based on the familiar two dimensional drawing of RNA secondary structures, yet also includes some information about probabilities (Fig. 3).

The basic shape is that of the the most probable structure according to the model probabilities. This structure includes two elements: single nucleotides (marked by a dot) and base pairs (marked by two dots connected by a line), each is produced from a specific state of the model. The color of such a structural element is darker as the probability assigned to it by the model is higher. Sequence elements with high probability are also indicated in this representation, with a color scale that represents their probability (green for low probability and red for high). Looking at this representation, one must bear in mind that although the structure shown is the most probable one, other structures, sometimes quite different, can also have a relatively high probability. Two examples of such representations are shown in Fig. 3, along with the tree representation of the model they represent.
REFERENCES


