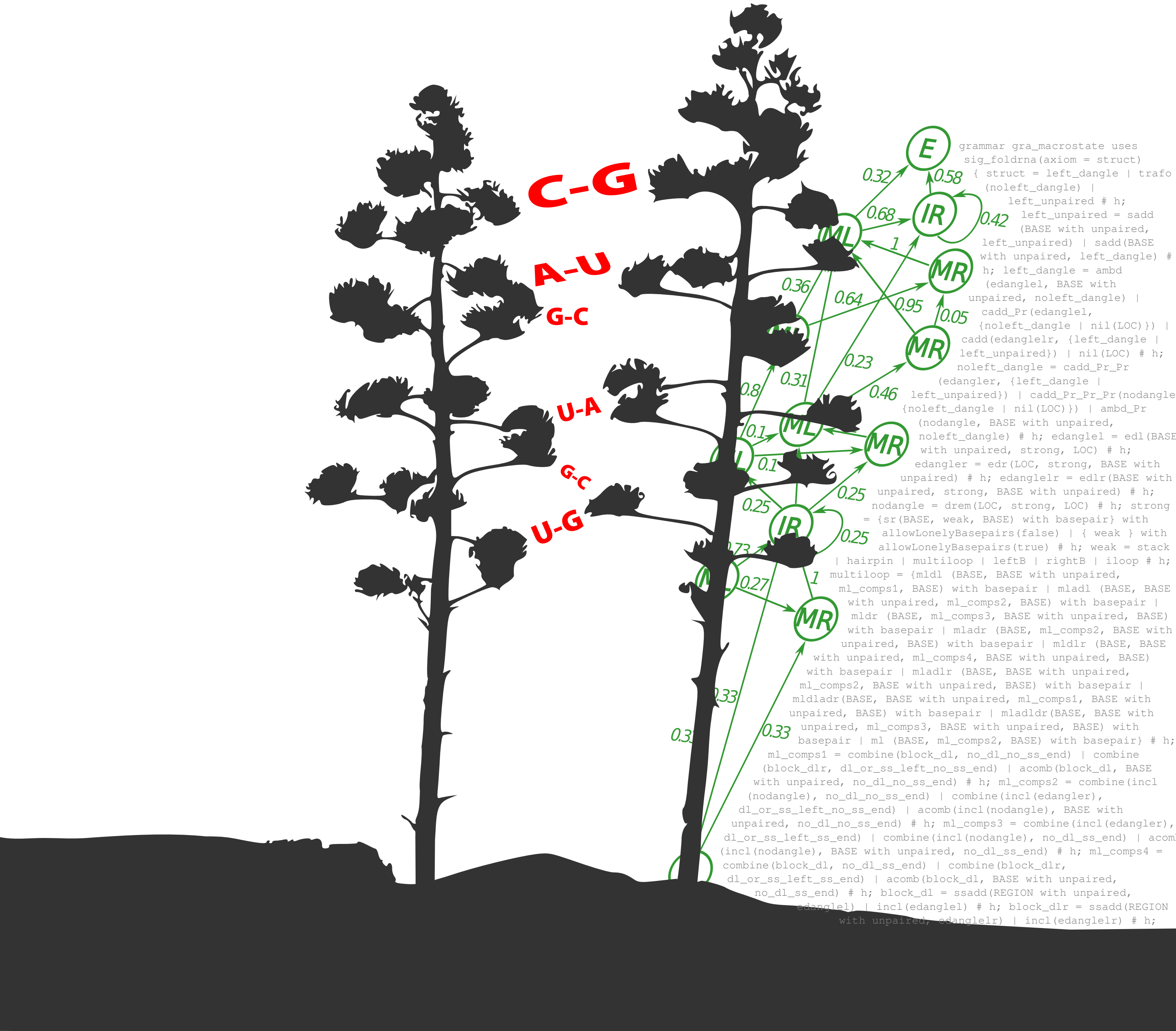
Dissertation zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften (Dr. rer. nat.)
der Technischen Fakultät der Universität Bielefeld

Kisses, ambivalent models and more: Contributions to the analysis of RNA secondary structure.

Stefan Janssen

April 8, 2014



```
grammar gra_macrostate uses
 sig_foldrna(axiom = struct)
  { struct = left_dangle | trafo
    (noleft_dangle) |
        left_unpaired # h;
          left_unpaired = sadd
          (BASE with unpaired,
     left_unpaired) | sadd(BASE
     with unpaired, left_dangle) #
     h; left_dangle = ambd
      (edanglel, BASE with
   unpaired, noleft_dangle) |
     cadd_Pr(edanglel,
     {noleft_dangle | nil(LOC)}) |
   cadd(edanglelr, {left_dangle |
   left_unpaired}) | nil(LOC) # h;
   noleft_dangle = cadd_Pr_Pr
  (edangler, {left_dangle |
 left_unpaired}) | cadd_Pr_Pr_Pr(nodangle
{noleft_dangle | nil(LOC)}) | ambd_Pr
  (nodangle, BASE with unpaired,
  noleft_dangle) # h; edanglel = edl(BASE
  with unpaired, strong, LOC) # h;
 edangler = edr(LOC, strong, BASE with
unpaired) # h; edanglelr = edlr(BASE with
unpaired, strong, BASE with unpaired) # h;
nodangle = drem(LOC, strong, LOC) # h; strong
= {sr(BASE, weak, BASE) with basepair} with
  allowLonelyBasepairs(false) | { weak } with
 allowLonelyBasepairs(true) # h; weak = stack
| hairpin | multiloop | leftB | rightB | iloop # h;
multiloop = {mldl (BASE, BASE with unpaired,
 ml_comps1, BASE) with basepair | mladl (BASE, BASE
  with unpaired, ml_comps2, BASE) with basepair |
   mldr (BASE, ml_comps3, BASE with unpaired, BASE)
  with basepair | mladr (BASE, ml_comps2, BASE with
 unpaired, BASE) with basepair | mldlr (BASE, BASE
with unpaired, ml_comps4, BASE with unpaired, BASE)
with basepair | mladlr (BASE, BASE with unpaired,
ml_comps2, BASE with unpaired, BASE) with basepair |
mldladr(BASE, BASE with unpaired, ml_comps1, BASE with
unpaired, BASE) with basepair | mladldr(BASE, BASE with
 unpaired, ml_comps3, BASE with unpaired, BASE) with
 basepair | ml (BASE, ml_comps2, BASE) with basepair} # h;
ml_comps1 = combine(block_dl, no_dl_no_ss_end) | combine
(block_dlr, dl_or_ss_left_no_ss_end) | acomb(block_dl, BASE
with unpaired, no_dl_no_ss_end) # h; ml_comps2 = combine(incl
(nodangle), no_dl_no_ss_end) | combine(incl(edangler),
dl_or_ss_left_no_ss_end) | acomb(incl(nodangle), BASE with
unpaired, no_dl_no_ss_end) # h; ml_comps3 = combine(incl(edangler),
dl_or_ss_left_ss_end) | combine(incl(nodangle), no_dl_ss_end) | acom
(incl(nodangle), BASE with unpaired, no_dl_ss_end) # h; ml_comps4 =
combine(block_dl, no_dl_ss_end) | combine(block_dlr,
dl_or_ss_left_ss_end) | acomb(block_dl, BASE with unpaired,
 no_dl_ss_end) # h; block_dl = ssadd(REGION with unpaired,
   edanglel) | incl(edanglel) # h; block_dlr = ssadd(REGION
      with unpaired, edanglelr) | incl(edanglelr) # h;
```

Gedruckt auf alterungsbeständigem Papier nach ISO 9706.

Acknowledgments

Scientists are crazy people. They devote their lives often with surpassing, sometimes even baneful, tenacity to explain the miracles of nature. I always wanted to become one of them.

I thank my parents for never bringing me to reason and full support to reach this level of craziness. I am deeply indebted to Robert Giegerich for many years of unobtrusively but priceless guidance to science and his faith in my skills.

Many thanks go to the father of Bellman’s GAP, Georg Sauthoff, who patiently bear all my special requirements. For past and ongoing RNA debates, I am especially grateful to Jan Reinkensmeier, Cédric Saule, Stefanie Schirmer and Jens Reeder. Apologies to Jan Krüger, Daniel Hagemeier, Thomas Gatter and Madis Rumming for all the extra work of bringing my software online via their BiBiServ project. I thank Andreas Bremges and Benedikt Loewes for reporting bugs and the whole AG-PI team for a very cooperative working atmosphere.

Further thanks go to Sebastian Loth, Sebastian Schipporeit, Steffi and Cedric for the pain of carefully proofreading this thesis. Anne Arend pushed me into the right direction, once again. Without her, I had certainly vasted much time by re-coding my software instead of writing this thesis. Last but not least, I thank my girlfriend for sharing her life with me. She made me not only explaining nature, but enjoying the beauty of it.

I love you, Iris!

Abstract

The full functional role of RNA in all domains of life is yet to be explored. Deep sequencing technologies generate massive data about RNA transcripts with functional potential. To decipher this information, bioinformatics methods for structural analysis are in demand. With this thesis at hand, we want to improve current secondary structure prediction in different respects.

The introductory chapter explains ADP with a focus on its comfortable, but atypical style of specifying algorithms. Then, we present five contributions to the analysis of RNA secondary structures.

1. It is the nature of models to abstract and simplify reality in order to master its complexity. Chapter 3 is an in depth analysis of four popular computational models of RNA secondary structure (Programs RNAshapes and RNAalishapes).

2. The secondary structure of RNA is too dynamic to be described by a single structure and in turn, there is no single optimal secondary structure. Thus, we compute the most likely abstract shape of a given RNA sequence. Improvements of the algorithms for computing the likelihood of abstract shapes are discussed in Chapter 4, specifically with regards to computational speed (Program RapidShapes).

3. For computational complexity reasons, models of RNA structures commonly exclude crossing base-pairs, the so-called “pseudoknots”, from the secondary structure. In Chapter 5, we introduce a heuristic for mastering a frequent type of pseudoknots: “kissing-hairpins” (Program pKiss).

4. In Chapter 6 we revisit the old algorithmic idea of outside-in computation for the new programming framework Bellman’s GAP. This broadens the arsenal of rapid prototyping algorithms for RNA and other sequential problems. It adds “outside” and “MEA” functionality to RNAshapes and RNAalishapes.

5. Covariance Models representing RNA families assume a single consensus secondary structure for a set of related RNAs and serve as statistical tools to search for additional members. In Chapter 7, we evaluate CM scorings that are more structure-specific than the standard sequence-to-model alignments. Furthermore, we introduce a technique to incorporate “ambivalent” consensus structures into covariance models (Program aCMs).

The results of this work are available at the Bielefeld Bioinformatic Server. The RNA Studio (`http://bibiserv.cebitec.uni-bielefeld.de/rna`) supports ready to use web-submissions, web-services and cloud computing for the programs developed in this thesis. debian packages foster a simple way to install our software on your local machine. Developers can benefit from our algorithmic analyses or use our sources for rapid prototyping as a primer for new implementations: `http://bibiserv.cebitec.uni-bielefeld.de/fold-grammars`.

Contents

1 Introduction

The RNA world hypothesis [35] puts the ribonucleic acid (RNA) in the center of the origin of life. RNA is a linear bio-polymer with a very repetitive structure. Every monomer (nucleotide) is composed of a ribose sugar, bound to a phospate, and carries at its 1' end one of the following bases: adenine, cytosine, guanine or uracil. A monomer is connected with its ribose 3' end to the phosphate of the next monomer, which is bound at its 5' end.

Simple as RNA is, it encodes genetic information by the sequence of nucleotides. Furthermore, it can catalyse chemical reactions by its 3D embodiment. Both properties in combination should be sufficient to create the first stable self-replicating system: life. During the evolution, RNA invented two new types of bio-molecules to specialize both capabilities: 1) error correcting double stranded DNA for more stable information storage and 2) more customizable proteins, composed out of about 20 different amino acids, for more efficient catalyses.

Due to this outsourcing, RNA was thought as only a rather passive messenger between DNA and proteins. Its importance has been obscured to scientists for many years. This neglect ended abruptly when it became clear that the human genome did not carry enough protein-coding genes to account for its metabolic and developmental complexity. Rapidly, several new RNA-based mechanisms were discovered, such as gene silencing in mammals and plants [24], and an immune system in bacteria [42]. Bioscience has just begun to understand the universal dependence of life on RNA. While determination of the sequence for an RNA molecule in a wet-lab is nowadays trivial, discovering the 3D structure and thereby clues about its function is a matter of years.

This situation raises the needs for computer predictions. Given the sequence, what is the most likely structure for this RNA molecule? Unfortunately, present days computer hardware is too slow to master this problem within reasonable time or resources. Luckily, chemists found that RNA structures form hierarchically when synthesised [89], i. e. early connections between nucleotides will most often remain in the final structure. A set of these pairs is called a *secondary structure*.

With this thesis at hand, we want to improve current secondary structure prediction in different directions.

- It is the nature of computer models to abstract and simplify reality to tackle its complexity. Chapter 3 is an in depth analysis about four popular variations to model secondary structures in computers.

- It is known that an RNA is too dynamic to describe it just by *one* optimal secondary structure. Instead, we prefer to predict the most probable secondary structure in terms of the shape in which the molecule stays most of its time. Chapter

4 reports about algorithmic improvements to significantly speed-up this kind of synoptic predictions.

- A common model cutback – purely for computational reasons – is to restrict a secondary structure to contain no two connections, which cross each other. However, these *pseudoknots* are frequently observed in nature. In Chapter 5, we introduce a biology inspired heuristics to conquer the sub-class of pseudoknots, which are named "kissing-hairpins".

- In Chapter 6 we re-invent an old algorithmic idea (outside-in computation [47]) for a new programming framework (Bellman's GAP). With the concrete example of McCaskill base-pair probabilities, we introduce a general approach to enable outside-in computations in Bellman's GAP, which is naturally not able to do so. This broadens the arsenal of rapid prototyping algorithms for RNA and other sequential problems.

- If we assume that related organisms evolved conjointly, their RNAs are expected to be similar but not identical. We can improve predictive power from these small differences compared to single sequences or identify important regions if they are conserved over time. Covariance Models (CMs) assume one consensus secondary structure for such a family of RNAs and serve as statistical tools to screen for further members. Despite the structural importance, current CMs are often very sequence centric. With Chapter 7, we evaluate more structure specific scorings for CMs and introduce a technique to incorporate ambivalent consensus structures.

Background information about the comfortable but atypical style to specify our algorithms is given in Chapter 2.1.

2 Background

2.1 Algebraic Dynamic Programming

Algebraic Dynamic Programming (ADP) [31] is a discipline to formulate algorithms for sequential problems. Its high level of abstraction allows for a clear separation of concerns. 1) A combinatorial search space is generated by a *regular tree grammar*. 2) Each candidate of the search space is evaluated by an *evaluation algebra*. 3) The "best" candidate is determined by an *objective function*. 4) Dynamic programming's characteristic tabulation of intermediate results – the *table design* [86] – can be tuned afterwards to trade speed for a lower memory footprint without editing existent components; just by annotations to the already existing regular tree grammar. Dynamic programming (DP) intermingles search space generation, evaluation and optimization for the – usually – exponential number of candidates in the search space by applying Bellman's Principle of Optimality [7], such that the best candidate can be determined within polynomial time and space. In contrast to traditional DP, only execution but not development of the different components are intermingled in ADP. Thus, components can be easily replaced or even combined [85] to tackle new challenges. Error prone subscripts, as known from traditional DP matrix recurrences, are not necessary any more.

Although highly optimized code will surely outperform ADP programs, development is significantly faster, programs are easier to debug due to the absence of indices and hence it is definitely more fun.

The formal concepts of ADP are defined in the following section. These are illustrated by examples from RNA secondary structure prediction. Still, ADP is not restricted to RNA problems, but can be applied to all kinds of problems over sequential data. Implementation details are postponed to Section 2.1.2, since ADP is more a style to organize and think about a DP program than a single framework.

2.1.1 ADP Formalism

Regular Tree Grammar

A regular tree grammar is formed by an alphabet, a set of non-terminal symbols and a set of production rules, whose functions follow a signature. In the following, we provide the general definitions as well as their specific form in RNA sequence analysis.

Definition 1. *An* alphabet $\mathcal{A}$ *is a finite set of symbols.*

In the context of RNA, the symbols are `A` for Adenine, `C` for Cytosine, `G` for Guanine and `U` for Uracil.

Table 2.1: Signature Σ_5 for RNA problems. The result of all algebra functions is of type $\mathcal{S}$ as defined in 2. $\mathcal{A}$ is a symbol from the alphabet, i. e. a single character from the input sequence.

$\mathrm{nil}()$
$\mathrm{pair}(\mathcal{A}, \mathcal{S}, \mathcal{A}, \mathcal{S})$
$\mathrm{open}(\mathcal{A}, \mathcal{S})$

Definition 2. *A* signature Σ *over* $\mathcal{A}$ *is a family of function declarations. Functions can have different arity. Their arguments are either of the type* $\mathcal{A}$ *or* $\mathcal{S}$ *– a placeholder for an unspecified data domain, called* sort*. The results are always of type* $\mathcal{S}$*.*

Signatures describe languages of *terms*. Terms are well-typed formulas constructed by functions of Σ and symbols of $\mathcal{A}$. The *term language* defined by Σ over $\mathcal{A}$ is denoted as T_Σ. Allowing variables from a set V in the terms result in a term language with variables $T_\Sigma(V)$.

The example signature Σ_5 for RNA related problems is given in Table 2.1. This can be regarded as a JAVA interface that is limited to defining the available functions and the abstract types of their arguments.

Definition 3. *A* regular tree grammar *over* Σ *is a four-tuple* $\mathcal{G} = (\mathcal{A}, V, Z, P)$*, where* V *is a finite set of non-terminal symbols,* $Z \in V$ *is a designated start symbol – called* axiom*, and* P *is a finite set of productions of the form* $v \to t$ *with* $v \in V$ *and* $t \in T_\Sigma(V)$*.*

A tree grammar $\mathcal{G}$ describes a language $\mathcal{L}$:

$$\mathcal{L}(\mathcal{G}) = \{t \mid t \in T_\Sigma,\ Z \Rightarrow^* t\}$$

This defines all valid terms $t \in T_\Sigma$ that can be formed by starting from the axiom Z and applying one or more productions ($\Rightarrow^*$) of $\mathcal{G}$. In contrast, using only a signature is not sufficient as it would allow any combination of functions. The grammar limits this multiplicity to only those combinations given by the productions, i. e. $\mathcal{L}(\mathcal{G}) \subseteq T_\Sigma$. In our RNA example, $\mathcal{L}(\mathcal{G})$ is the language of all structures for all RNA sequences.

Regular tree grammars are ADP's combinatorial machinery to systematically span the search space. So far, these definitions are independent of the input. This changes now:

Definition 4. *The* yield function y *has type* $T_\Sigma \to \mathcal{A}^*$ *and is defined by* $y(a) = a$ *for* $a \in \mathcal{A}$*, and* $y\left(C\left(x_1, \ldots, x_n\right)\right) = y(x_1) \ldots y(x_n)$ *for* $n \geq 0$ *and each* n*-ary function* C *of* Σ*.*

If we think of a term t as a tree, the yield function $y(t)$ concatenates all leaves of type $\mathcal{A}$ into a plain string through a preorder traversal.

Given a regular tree grammar $\mathcal{G}$, the yield-function y and a concrete input $x \in \mathcal{A}^*$, the search space spawned by x is

$$F_\mathcal{G}\left(x\right) = \{t \mid t \in \mathcal{L}(\mathcal{G}),\ y(t) = x\}.$$

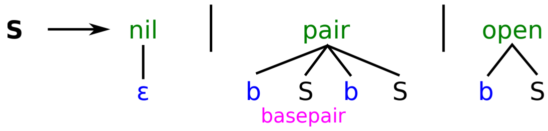


Figure 2.1: Regular tree grammar $\mathcal{G}_5$ for RNA secondary structure prediction. Single non-terminal and thus also axiom is S. Functions are given in green, terminal parsers are `blue`, syntactic filters are pink.

Figure 2.1 depicts the regular tree grammar $\mathcal{G}_5$ [20] for a model of RNA secondary structures, and Table 2.1 holds the respective signature Σ_5. Let the alphabet be $\mathcal{A} = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{U}\}$ and the only non-terminal be $V = \{S\}$. Then, axiom must be $Z = S$ which is given in bold face. Productions P with identical left hand sides are merged into one rule; alternative right hand sides are separated by vertical bars. Functions of Σ are colored in green. Terminal parser `b` takes exactly one *base* $\in \mathcal{A}$ from the input. The special terminal parser ε reads the "empty word", i. e. does not consume any character from the input. This is typically used to indicate the end of parsing for the current derivation. The annotation basepair at function pair means that left l and right r `b`ases must be able to form a valid base-pair, i. e. $(l, r) \in \{(\mathtt{A}, \mathtt{U}), (\mathtt{U}, \mathtt{A}), (\mathtt{C}, \mathtt{G}), (\mathtt{G}, \mathtt{C}), (\mathtt{G}, \mathtt{U}), (\mathtt{U}, \mathtt{G})\}$. Annotating a grammar with *syntactic filters*, e. g. basepair, is an abbreviation to modify the grammar. Otherwise, the explicit form for restricting the set of valid base-pairs would employ six different productions of pair. Consequently, we would use terminal parser for specific bases, e. g. pair$(\mathtt{A}, S, \mathtt{U}, S)$, pair$(\mathtt{U}, S, \mathtt{A}, S)$, ..., instead of the general terminal parser `b`.

The concept of $\mathcal{G}_5$ is as follows: Each term representing a secondary structure is derived from the axiom S. Any structure can be extended by adding a single unpaired base to its left (open). Alternatively, a structure grows by a base-pair (pair). This automatically introduces a potential structural bifurcation, as a result of the second non-terminal S of production pair. The bifurcation can be masked by immediately ending this derivation via a nil production. Also, the production nil must be used to terminate rows of unpaired bases or base-pairs with no inner sub-structure.

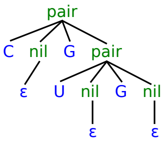
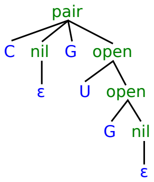
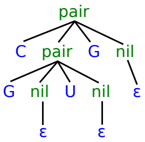
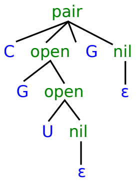
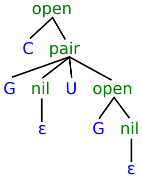
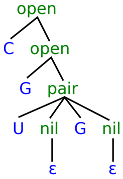
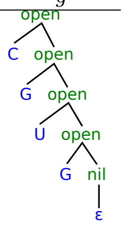
The complete search space $F_{\mathcal{G}_5}(\mathtt{CGUG})$ shown in Table 2.2 consists of seven terms which are depicted as little trees t_a to t_f.

A regular tree grammar has the same power as a *context free string grammar* (CFG), i. e. it generates the same language if we apply the yield function y to every term in the search space:

$$\mathcal{L}_y(\mathcal{G}) = \{y(t) \mid t \in T_\Sigma,\ Z \Rightarrow^* t\}$$

Looking at a candidate generated by a CFG there is no hint about the productions that have been used to parse the input. Our example input `CGUG` would be exactly re-established seven times, thus we cannot identify how they were produced. The great advantage of regular tree grammars is the use of algebra functions which are part of the terms. As with CFGs, terms do not capture by which productions they have been generated. However, the algebra functions retain an internal structure.

Table 2.2: Complete search space of $F_{\mathcal{G}_5}$(CGUG), holding seven terms (or candidates): t_a to t_g. Below the tree representations, the *yield-string* of the terms are shown in blue, their *Vienna-Dot-Bracket string* (see Section 2.1.1) in red, the number of base-pairs in **black** as well as their counting in regular font.

t_a	t_b	t_c	t_d	t_e	t_f	t_g
						
CGUG	CGUG	CGUG	CGUG	CGUG	CGUG	CGUG
()()	()..	(())	(..)	.().	..()	....
2	**1**	**2**	**1**	**1**	**1**	**0**
1	1	1	1	1	1	1

Evaluation Algebras

The algebra functions which are part of the candidates allow a conceptual separation of candidate generation, evaluation and selection for the "best" candidates. A Σ-*algebra* is a concrete implementation of a signature and is used to evaluate terms of the search space. Furthermore, it also specifies a concrete data type for the abstract sort $\mathcal{S}$.

Optimization problems typically pose the question: Given the scores of different candidates, which are the ones we are interested in? We identify the set of interesting candidates by an *objective function* h. Hence we define an *evaluation algebra* $\mathcal{I}$ as a Σ-algebra augmented with an objective function h.

Definition 5. *Evaluating terms from the search space with functions of a* Σ*-algebra results in a list of values which are of type* $\mathcal{S}$*:* $[\mathcal{S}]$*. An* objective function h *– often called* choice function *– is a mapping*

$$h \ : \ [\mathcal{S}] \rightarrow [\mathcal{S}] \, .$$

Using identity as objective function will simply preserve all candidate values. An objective function might withdraw all but the best k candidate values, or remove those values that exceed a specific threshold. If h is a function that naturally results in a single value (e. g. maximization or summation), it must be engulfed in a list! In the following, we use the terms "algebra" and "evaluation algebra" equivalently.

In Table 2.3 four different algebras are presented. All functions (leftmost column), defined by the signature, are implemented by concrete operations. In our case string concatenation ($\oplus$), summation ($+$) or multiplication ($\cdot$). Additionally, the abstract sorts $\mathcal{S}$ become concrete data types (string or int) and the optimization criteria – the objective functions – are determined (*id*entity, *max*imization or summation: Σ).

Table 2.3: Four evaluation algebras for terms of Σ_{RNA}. Algebra $\mathcal{I}_{yield}$ computes the yield function y for a candidate t. The result of $\mathcal{I}_{db}$ is a Vienna-Dot-Bracket string for t. Algebra $\mathcal{I}_{Nussinov}$ computes the number of base-pairs in t. $\mathcal{I}_{Count}$ with summation as objective function reports the size of the search space. Operator $\oplus$ denotes string concatenation, ε is the empty string. An algebra is the place where the abstract type $\mathcal{S}$ becomes concrete, thus we give these types as the penultimate row. In the last row, we define the objective function for the algebras, which is *id*entity for $\mathcal{I}_{yield}$ and $\mathcal{I}_{db}$, i. e. candidates are never ruled out, *max*imization in the case of counting base-pairs for $\mathcal{I}_{Nussinov}$ and summation (Σ) for search space counting via $\mathcal{I}_{Count}$.

algebra function	$\mathcal{I}_{yield}$	$\mathcal{I}_{db}$	$\mathcal{I}_{Nussinov}$	$\mathcal{I}_{Count}$
nil()	ε	ε	0	1
pair$(a, x, \hat{a}, y)$	$a \oplus x \oplus \hat{a} \oplus y$	$\texttt{(} \oplus x \oplus \texttt{)} \oplus y$	$1 + x + y$	$x \cdot y$
open(a, x)	$a \oplus x$	$\texttt{.} \oplus x$	x	x
$\mathcal{S}$	string	string	int	int
objective function	id	id	*max*	Σ

Yield string Given an arbitrary candidate t from the search space defined by $\mathcal{G}_5$ algebra $\mathcal{I}_{yield}$ recovers the original input sequence. This resembles the previously introduced behavior of the yield function y of Definition 4. Take candidate t_a of Table 2.2 as an example. Written as a function it looks like:

$$t_b = \text{pair}\left(\texttt{C}, \text{nil}\left(\varepsilon\right), \texttt{G}, \text{open}\left(\texttt{U}, \text{open}\left(\texttt{G}, \text{nil}\left(\varepsilon\right)\right)\right)\right)$$

We now systematically evaluate all abstract algebra functions by the concrete operations, defined by $\mathcal{I}_{yield}$:

$$(\texttt{C} \oplus \varepsilon \oplus \texttt{G} \oplus (\texttt{U} \oplus (\texttt{G} \oplus \varepsilon)))$$

Since string concatenation is associative, we can remove all parentheses:

$$\texttt{C} \oplus \varepsilon \oplus \texttt{G} \oplus \texttt{U} \oplus \texttt{G} \oplus \varepsilon$$

and finally, perform the concatenation:

$$\texttt{CGUG}$$

Please note that ε is the empty word, and thus vanishes if concatenated to a non-empty word. As expected from a yield function, the result of $y(t_b) \equiv \mathcal{I}_{yield}(t_b) = \texttt{CGUG}$ is equal to the original input sequence. Of course, this is also true for all other terms.

Now that we performed kind of a sanity check by applying a yield function algebra to a candidate to understand how it is evaluated, we can start to use other algebras for computing new information.

Vienna-Dot-Bracket string Vienna-Dot-Bracket strings are machine friendly representations for secondary structures. A dot '`.`' represents an unpaired base. A pair of parentheses '`()`' indicates an opening and closing position of a base-pair. The algebra $\mathcal{I}_{db}$ evaluates candidates into these string representations. With respect to concatenating elements, the algebras $\mathcal{I}_{db}$ and $\mathcal{I}_{yield}$ are highly similar. Crucially, $\mathcal{I}_{yield}$ operates on symbols of the input whereas $\mathcal{I}_{db}$ uses dots and brackets. For example, substituting algebra functions with operators () and function arguments with dots and brackets ($\mathcal{I}_{db}$, see column 3 in Table 2.3) the candidate t_b results in:

$$(\texttt{(} \oplus \varepsilon \oplus \texttt{)} \oplus (\texttt{.} \oplus (\texttt{.} \oplus \varepsilon))) \;=\; \texttt{()..}$$

Base-pair counting Ruth Nussinov's pioneering algorithm [67] for predicting a secondary structure computes the maximal number of base-pairs for a given RNA input sequence. The number of base-pairs in each term is counted by using the algebra $\mathcal{I}_{Nussinov}$ as defined in Table 2.3. Applying $\mathcal{I}_{Nussinov}$ for substituting algebra functions to our example candidate t_b results in:

$$(1 + 0 + ((0))) = 1.$$

Candidate counting The size of the search space is determined by counting each candidate using the algebra $\mathcal{I}_{Count}$ (as defined in Table 2.3). Every terminal parser (`b` or ε) returns 1 and every non-terminal parser simply reports the current value, without any changes.

An algebra function containing several terminal and non-terminal parsers returns the product of these numbers, i. e. $\mathrm{pair}(a, x, \hat{a}, y) = 1 \cdot x \cdot 1 \cdot y$ and $\mathrm{open}(a, x) = 1 \cdot x$. Our example candidate t_b is evaluated to

$$(1 \cdot 1 \cdot ((1))) = 1.$$

This reflects the expectation of evaluating *one* candidate.

Objective function

So far, we have evaluated candidates with algebras but we have not applied objective functions. This is the last step in solving an ADP problem.

Definition 6. *Given a regular tree grammar $\mathcal{G}$, an evaluation algebra $\mathcal{I}$, including an objective function h, and an input sequence x, an ADP problem is solved by computing:*

$$\mathcal{G}\,(\mathcal{I}, x) \;:=\; h\,[\mathcal{I}\,(t) \;\mid t \in \mathcal{L}(\mathcal{G}), y\,(t) = x]\,.$$

In words: all candidates, which are generated by the grammar $\mathcal{L}(\mathcal{G})$ and whose yield string equals the original input $y\,(t) = x$, are evaluated by the algebra functions defined in $\mathcal{I}$. From this list, the objective function h selects suitable values.

The search space $F_{\mathcal{G}_5}(\texttt{CGUG})$ consists of the candidates $[t_a, t_b, t_c, t_d, t_e, t_f, t_g]$. If we assume that the objective function of $\mathcal{I}_{Nussinov}$ is the identity function (as defined in Table 2.2), we would get

$$\mathcal{G}_5\left(\mathcal{I}_{Nussinov}, \texttt{CGUG}\right) = [2, 1, 2, 1, 1, 1, 0] .$$

The optimization problem of finding the maximal number of base-pairs for a given RNA sequence can be solved by switching the objective function to *max*imization:

$$\mathcal{G}_5\left(\mathcal{I}_{Nussinov}, \texttt{CGUG}\right) = [2] .$$

The evaluation of the candidates using $\mathcal{I}_{Count}$ and summation for h reduces the list of the results – $[1, 1, 1, 1, 1, 1, 1]$ – to the size of the search space:

$$\mathcal{G}_5\left(\mathcal{I}_{Count}, \texttt{CGUG}\right) = [7] .$$

Efficiency

The computational efficiency is irrelevant for abstract concepts, but this certainly differs for implementations of ADP! Specifically, two properties affect the efficiency of the computation: First, the conceptional question whether the problem can be tackled by Dynamic Programming at all, i. e. is Bellman's Principle of Optimality satisfied? Secondly, the rather technical side of deciding which non-terminals should be tabulated – the table design.

Table design Dynamic programming is fast because intermediate results are stored in tables and are re-used on demand instead of computing them from scratch for all (usually exponentially many) candidates of the search space. There is a trade-off between speed and memory footprint. One extreme option is to tabulate all non-terminals of a grammar. This reduces run-time to a minimum, but it also maximizes the memory usage. Often we want to trade the (in principle) infinite resource time for the naturally limited resource memory. Ideally with only little losses in speed, i. e. a constant slow-down is accepted as long as the algorithm preserves the original asymptotic class.

Finding an optimal table design for an arbitrary grammar is a NP-complete problem [86]. However, the ADP implementation Bellman's GAP typically provides good results by employing an heuristic [79]. The heuristic takes into account that it is sometimes faster to re-compute than to access a value from the relatively slow main memory and thus, not every non-terminal should be stored. Using a yield-size analysis, the heuristic computes the optimal number of dimensions of the tables – not every non-terminal needs a quadratic table – and estimates constants for the asymptotic run-time.

Bellman's Principle of Optimality Working with tabulated solutions for sub-problems will only yield correct results, if the algebra satisfies Bellman's Principle of Optimality, i. e. if the overall solution can be combined of solutions for sub-problems. In Bellman's own words:

> An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision [7].

We re-formalize this principle for ADP:

Definition 7. *An evaluation algebra $\mathcal{I}$ satisfies* Bellman's Principle of Optimality, *iff for each k-nary algebra function f in $\mathcal{I}$ and all answer lists $Z_1, \ldots, Z_k$, the objective function h satisfies:*

1. $h([f(x_1, \ldots, x_k) \mid x_1 \leftarrow Z_1, \ldots, x_k \leftarrow Z_k]) =$ $h([f(x_1, \ldots, x_k) \mid x_1 \leftarrow h(Z_1), \ldots, x_k \leftarrow h(Z_k)])$
2. $h([]) = []$
3. $h(Z_i \oplus Z_j) = h(h(Z_i) \oplus h(Z_j)) \;\; \forall\, 1 \leq i, j \leq k$

Operator $\oplus$ is list concatenation: $[a_1, \ldots, a_n] \oplus [b_1, \cdots, b_m] = [a_1, \ldots, a_n, b_1, \cdots, b_m]$, $[]$ is the empty list and $\leftarrow$ is list membership.

The practical interpretation of the optimality principle is that we may push the application of the objective function inside the computation of sub-problems, thus preventing combinatorial explosion.

Algebra products

ADP is very adaptable. An existing ADP program can be turned into an algorithm for solving a different problem simply by replacing the algebra, i. e. by adjusting a single component. For example, $\mathcal{G}_5\left(\mathcal{I}_{Nussinov}, x\right)$ determines the maximal number of base-pairs for x, while $\mathcal{G}_5\left(\mathcal{I}_{Count}, x\right)$ identifies the size of the search space. We do not have to re-think the search space, the signature or the table design. Yet, ADP goes one step further with the product of algebras [85].

Definition 8. *Let $\mathcal{I}_M$ and $\mathcal{I}_N$ be two evaluation algebras over signature Σ. Their* lexicographic algebra product $\mathcal{I}_M * \mathcal{I}_N$ *is an evaluation algebra over Σ and has the functions*

$$f_{\mathcal{I}_M * \mathcal{I}_N}\left(\left[\left(m_1, n_1\right), \ldots, \left(m_k, n_k\right)\right]\right) = \left(f_{\mathcal{I}_M}\left(\left[m_1, \ldots, m_k\right]\right), f_{\mathcal{I}_N}\left(\left[n_1, \ldots, n_k\right]\right)\right)$$

for each k-nary algebra function f in Σ and the objective function

$$\begin{aligned} h_{\mathcal{I}_M * \mathcal{I}_N}\left(\left[\left(m_1, n_1\right), \ldots, \left(m_k, n_k\right)\right]\right) = [(l, r) \mid \\ & l \in L, \\ & r \leftarrow h_{\mathcal{I}_N}\left(\left[r' \mid \left(l', r'\right) \leftarrow \left[\left(m_1, n_1\right), \ldots, \left(m_k, n_k\right)\right], l' = L\right]\right)\,] \\ & \textit{where } L = h_{\mathcal{I}_M}\left(\left[m_1, \ldots, m_k\right]\right) \end{aligned}$$

In the above, $\in$ denotes set membership and hence ignores duplicates. In contrast, $\leftarrow$ denotes list membership and respects duplicates.

Intuition For an intuitive understanding, we take the perspective of successive phases again: The first step is the evaluation of all k candidates by $\mathcal{I}_M$ and $\mathcal{I}_N$. Each algebra evaluates the candidates independently. The tuple (m_i, n_i) stores the results of algebra $\mathcal{I}_M$ and $\mathcal{I}_N$ for the candidate i. In turn, evaluating a list of candidates results in a list of tuples $[(m_1, n_1), \ldots, (m_k, n_k)]$.

The second phase, the candidate selection, takes place in two steps: The first step is to compile a unique *set* of answers L regarding only the left algebra $\mathcal{I}_M$. This is done in the "normal" ADP way as described by Definition 6. Duplicates might have to be removed from the result list, if the objective function $h_{\mathcal{I}_M}$ has not already done so. In the second step, $[(m_1, n_1), \ldots, (m_k, n_k)]$ is pruned such that only tuples are left, whose left component is element of L. All right components from the remaining tuples are collected in a list processed by the second objective function $h_{\mathcal{I}_N}$ thus compiling answer list $[r]$. In contrast to L, r can have duplicates. A list of tuples with a unique left element is generated by combining all elements of L with all elements of r (in process similar to the Cartesian product). This list is the desired result for our problem: the application of a algebra product to a search space.

Application examples Given some standard algebras and the algebra product operation, ADP can perform a remarkable variety of applications: optimizations under multiple objective functions, alternative solutions and backtracing, holistic search space analysis, ambiguity checking, and more. Each of these applications can be implemented without additional programming effort or the need for debugging.

The single instance $\mathcal{G}_5\,(\mathcal{I}_{Nussinov}, \texttt{CGUG})$ reports the numeric value of the maximal number of base-pairs which the sequence `CGUG` can fold $[2]$ and $\mathcal{G}_5\,(\mathcal{I}_{db}, \texttt{CGUG})$ shows Vienna-Dot-Bracket strings for all seven candidates $([\texttt{()()}, \texttt{()..}, \texttt{(())}, \texttt{(..)}, \texttt{.().}, \texttt{..()}, \texttt{....}])$. Importantly, both algebras can be combined by using the algebra product into a more complex instance $\mathcal{G}_5\,(\mathcal{I}_{Nussinov} * \mathcal{I}_{db}, \texttt{CGUG})$. This instance resembles a backtracing mechanism and returns only those Vienna-Dot-Bracket strings, that belong to those candidates which have maximal number of base-pairs, together with the number of base-pairs: $[(2, \texttt{()()}), (2, \texttt{(())})]$. Such an algorithm would otherwise be tedious to implement, specifically for co-optimal or sub-optimal enumeration. This manual work can be avoided elegantly by using the built-in algebra products of ADP.

We can count the number of co-optimal solutions by applying the algebra product $\mathcal{G}_5\,(\mathcal{I}_{Nussinov} * \mathcal{I}_{Count}, \texttt{CGUG})$, which is $[(2, 2)]$. Left 2 of the tuple is the maximal number of base-pairs, right 2 is the number of co-optimal candidates.

We can also combine algebras to meaningless products like $\mathcal{G}_5\,(\mathcal{I}_{Count} * \mathcal{I}_{Nussinov}, \texttt{CGUG})$ $= []$. The answer list is empty in this case because in the first phase of selection, L is $[7]$, the size of the search space. However, $\mathcal{I}_{Count}$ operated on each candidate individually without applying its objective function. Thus, all left components were evaluated as 1. Consequently, the second phase cannot find any candidate whose left component of the answer tuple is 7.

There is no additional programming effort needed to use algebra products, since they are a generic operation in ADP. But it is our obligation to prove whether a product of algebras still satisfies Bellman's Principle, which is not necessarily the case even if both single algebras fulfill it.

2.1.2 History of ADP implementations

During the past 15 years, the concepts of ADP were implemented four times. It started with *Haskell-ADP* [27, 31], a prototypical embedding of the concepts in the functional programming language Haskell.

Secondly, aiming for a smaller memory footprint and a gain of speed, the compiler *ADP-C* was developed for translating Haskell-ADP into native C code [33, 84]. Essentially, this requires that Haskell is still used for programming ADP. However, the ADP principles are not accessible to the Haskell interpreter or compiler and thus, they reported incomprehensible errors messages, mostly convoluted type errors.

These problems led to the development of Bellman's GAP [32, 80, 79, 81]. This package consists of a new programming language (GAP-L) and a dedicated compiler (GAP-C) for generating C++ code. The name reflects the most important concepts: Bellman's Principle of Optimality, Grammars, Algebras and Products. GAP-C can comprehensively analyse ADP components written in GAP-L code. This allows the GAP-C to produce better error messages. Also, GAP-C introduces significant improvements in terms of developer support including (but not limited to): Non-productive or unreachable non-terminals in the grammar are reported, the burden of deciding on a good table design is taken from the developer by providing a feasible heuristic, asymptotic run-time analysis is provided and yield size analysis is performed automatically to avoid unnecessary moving boundaries while partitioning the input.

Due to recent improvement to the *Glasgow Haskell Compiler* (GHC), Christian Höner carries ADP back to a highly optimized Haskell version: *ADP-fusion* [41]. ADP-fusion's run-times are comparable to native C code, by sacrificing code readability for speed. Unfortunately, ADP-fusion works only in Christian's hands – for now.

In this work, we will focus only on Bellman's GAP.

Terminal parsers

Terminal parsers in Bellman's GAP do not hold symbols from the input, but start- and stop-positions of the parsed sub-sequence. Given the input and start- and stop-positions, the symbols can easily be recovered. Additionally, storing a region with only two numbers instead of all its symbols is often more efficient when working with longer sub-sequences. Not the symbols but their boundaries are addressed by indices, an example input might look like $_0\mathtt{G}_1\mathtt{A}_2\mathtt{G}_3\mathtt{A}_4\mathtt{A}_5\mathtt{A}_6\mathtt{C}_7\mathtt{C}_8$. The three successive As are addressed by the sub-sequence $(3,6) = {}_3\mathtt{A}_4\mathtt{A}_5\mathtt{A}_6$. The use of a fictitious 0-position at the start of the input avoids a lot of fiddling with ±1, since a sub-word (i,j) has length $j-i$ and splits seamlessly into sub-words (i,k) and (k,j) for $i \leq k \leq j$.
We will use four different terminal parsers:

1. The terminal parser b takes exactly one *base* $\in \mathcal{A}$ from the input. The resulting

tuple for the boundaries is of the form (i, j), with $j = i + 1$. In GAP-L, this parser is written as `BASE`.

2. The terminal parser `r` consumes a `r`egion from the input. A region is a successive row of bases written as $\mathcal{A}^*$ in a signature and `REGION` in GAP-L. The boundaries are depicted as (i, j) with $i < j$. An *empty* region constitutes a special case and is covered by `r0`, where $i \leq j$. In GAP-L, this empty region is written as `REGION0`.

3. Terminal parser ε does not consume any character from the input and is applicable at every position, but returns nothing. There is no boundary tuple. This parser is written as `EMPTY` in GAP-L.

4. A second non-consuming terminal parser is `l`, which can also be placed at every position. In contrast to ε, it reports its `l`ocation, i. e. there is a boundary tuple of the form (i, j), with $j = i$. GAP-L's writing is `LOC`. In signatures, we will write it as $\mathcal{A}^0$.

RNA energy library

In the usual approximation model, the free energy of an individual RNA secondary structure s is the sum of the energetic contributions of all structural elements of s [45]:

$$\Delta G^0_{T,s} = \sum_{\text{helices } j} \Delta G^0_{T,j} + \sum_{\text{loops } k} \Delta G^0_{T,k}$$

with energy of an individual helix:

$$\Delta G^0_{T,\text{helix}} = \sum_{\substack{\text{base-pair} \\ \text{stacks } m}} \Delta G^0_{T,m} \ .$$

That is, the energy of a helix depends only on its type of base-pairs $\{(\mathtt{A},\mathtt{U}),(\mathtt{U},\mathtt{A}),(\mathtt{C},\mathtt{G}),(\mathtt{G},\mathtt{C}),(\mathtt{G},\mathtt{U}),(\mathtt{U},\mathtt{G})\}$ stacking on its neighboring base-pair [10]. The minimum length of a helix is two base-pairs (one base-pair stack). Single (lonely) pairs should not exist. The energy of a loop depends on its type, the sequence(s) of loop nucleotides, and type of closing base-pair(s). A hairpin loop is closed by a helix, internal and bulge loops are closed by two helices and multiloops or junctions are closed by more than two helices. Thus, the free energy of any given secondary structure s can be computed by decomposing it into its basic structural elements. The elementary energy function E (see Table 2.4) is called repeatedly for each basic element. Summing up these values yields the free energy of s.

For Bellman's GAP, we include some of the original Vienna-Package [50] source files to address the concrete energy parameters. We can load the same parameter files and use the same temperature re-scaling as used by software from the Vienna-Package, which facilitate comparison of prediction results. The Vienna-Package is written in `C` and directly addresses the characters of the input. That is different from Bellman's GAP, where *boundaries* are addressed, see Section 2.1.2. See functions E of Table 2.4 as interfaces between both programming languages.

Energy functions typically operate on the RNA input, specifically the input positions. Their output depends on the RNA input. The function $\mathrm{E}_{\mathrm{sbase}}$, E_{ml} and E_{ul} are exceptions and point to constants. Let us assume `(.(...))` is a secondary structure with a hairpin loop and a left bulge for the input sequence ${}_0\mathtt{G}_1\mathtt{A}_2\mathtt{G}_3\mathtt{A}_4\mathtt{A}_5\mathtt{A}_6\mathtt{C}_7\mathtt{C}_8$. At some point, we have to look up the energy contribution for the left bulge, by accessing $\mathrm{E}_{bl}({}_1\mathtt{A}_2, {}_7\mathtt{C}_8)$. However, the energy value cannot be determined by taking into account only the pair $({}_1\mathtt{A}_2,\ {}_7\mathtt{C}_8)$. Additional information is required. First, the outer base-pair $({}_0\mathtt{G}_1, {}_7\mathtt{C}_8)$. Secondly, the inner base-pair $({}_2\mathtt{G}_3, {}_6\mathtt{C}_7)$ of the hairpin-loop, onto which the outer base-pair stacks. Thirdly, the size of the bulged out region, here the single ${}_1\mathtt{A}_2$. That means that we make assumptions about the sub-structure of a candidate by using E_{bl}. Thus, this imposes restrictions on the design of the grammar.

The call $\mathrm{E}_{\mathrm{bl}}(r, \hat{a})$, provides not only the symbols of r and $\hat{a}$ implicitly, but also their positions within the input. We may write that as $\mathrm{E}_{\mathrm{bl}}({}_ir_j, {}_k\hat{a}_l)$. All three energy contributions can be looked-up via these indices: 1) outer base-pair embraces the sub-word $({}_{i-1}r, \hat{a}_l)$, since we know that its left partner must be directly left of the unpaired region r. 2) inner base-pair covers the sub-word $(r_j, {}_k\hat{a})$, which could also be written as $(r_j, \hat{a}_{l-1})$, because $k+1 = l$. Here, we use the structural assumption that the left partner must be the right neighbor of the last unpaired base of r and right partner is directly left of $\hat{a}$. At last, 3) the size of the bulge loop is $j - i$.

Table 2.5 lists special energy functions – or better constants, used only in the context of pseudoknot folding algorithms. These values have not been determined in a wet lab, but show reasonable behavior in various computational experiments. However, the most important two – namely $\mathrm{E}_{\mathrm{init}_{pk}}$ and $\mathrm{E}_{\mathrm{init}_{pkiss}}$ – will be adjustable for the user when integrated into Bellman's GAP programs. We discuss the details about pseudoknot energies in Chapter 5.

Precision of the energy parameters in the Vienna-Package is limited to two decimal places, used data type is int instead of a floating point number. Thus, the parameters were shifted and in turn, the results have to be divided by 100 to get kcal/mol. For more details on the energy functions inspect the elaborate documentation in the source files `rtlib/rna.hh` and `librna/rnalib.c` of Bellman's GAP.

Table 2.4: Basic energy functions. The table lists the energy functions, their names in actual Bellman's GAP code, their inherent assumptions about the structure of sub-candidates and their meanings. Parameters a and $\hat{a}$ are pairing bases, r is a non-empty region and l is a non-consuming location.

Function	**GAP-C name**	**Structural assumptions**	**Meaning**
$\mathrm{E}_{\mathrm{sr}}(a, \hat{a})$	`sr_energy`	bases a and $\hat{a}$ build an outer base-pair and bases right of a and left of $\hat{a}$ form an inner base-pair	The most important source for stabilizing an RNA secondary structure is stacking of two (or more) base-pairs.
$\mathrm{E}_{\mathrm{termau}}(a, \hat{a})$	`termau_energy`	bases a and $\hat{a}$ form a base-pair	Base -pair different from (`G`,`C`) or (`C`,`G`) at the terminal end of a stacking region adds less stabilizing energy than *within* a stacking region.
$\mathrm{E}_{\mathrm{hl}}(r)$	`hl_energy`	bases left and right of r form a base-pair	Stabilizing contribution for the loop-closing base -pair stack plus destabilizing contribution for the hairpin loop region plus bonus energy for special loop sequence (e. g. extra-stable tetra loops).
$\mathrm{E}_{\mathrm{bl}}(r, \hat{a})$	`bl_energy`	base left of r builds an outer base-pair with $\hat{a}$ and base right of r and left of $\hat{a}$ form an inner base-pair	Analog to E_{hl}, but for a destabilizing loop region bulged out to the left.
$\mathrm{E}_{\mathrm{br}}(a, r)$	`br_energy`	base right of r builds an outer base-pair with a and base left of r and right of a form an inner base-pair	Symmetric case to E_{bl}.
$\mathrm{E}_{\mathrm{il}}(r_l, r_r)$	`il_energy`	bases left of r_l and right of r_r form an outer base-pair and bases right of r_l and left of r_r form an inner base-pair	Analog to E_{hl}, but with two destabilizing loop regions.
$\mathrm{E}_{\mathrm{ml}}()$	`ml_energy`	–	Since a multiloop of x stems is less stable than x adjacent stems, it gets a penalty.
$\mathrm{E}_{\mathrm{ul}}()$	`ul_energy`	–	Each stem in a multiloop gets an initial penalty.
$\mathrm{E}_{\mathrm{ss}}(r)$	`ss_energy`	–	Regions of unpaired bases could get penalized, but typically this value is set to zero.
$\mathrm{E}_{\mathrm{sbase}}()$	`sbase_energy`	–	Same as E_{ss}, but for a single unpaired base.
$\mathrm{E}_{\mathrm{dl}}(a, l)$	`dl_energy`	bases a and l build a base-pair, on which the base left of a dangles	A single base left of a closed sub-structure can dangle onto this stack and thus might further stabilize it.
$\mathrm{E}_{\mathrm{dr}}(l, \hat{a})$	`dr_energy`	bases l and $\hat{a}$ build a base-pair, on which the base right of $\hat{a}$ dangles	Symmetric case to E_{dl}.
$\mathrm{E}_{\mathrm{ext\_mm}}(a, \hat{a})$	`ext_mismatch_energy`	bases a and $\hat{a}$ build a base-pair, on which the bases left of a and right of $\hat{a}$ dangle	Two bases left and right of a stack, which do not form a base-pair (they mismatch), can dangle from both sides to the stack.
$\mathrm{E}_{\mathrm{dli}}(a, \hat{a})$	`dli_energy`	bases a and $\hat{a}$ build a base-pair, on which the base left of a dangles from the inner	A multiloop is closed by one stack. A single base at the inside of the multiloop and directly next to the closing stack might dangle from left onto this stack. The energy values are the same as E_{dr}, but for a reversed sub-sequence.
$\mathrm{E}_{\mathrm{dri}}(a, \hat{a})$	`dri_energy`	bases a and $\hat{a}$ build a base-pair, on which the base right of $\hat{a}$ dangles from the inner	Symmetric case to $\mathrm{E}_{\mathrm{dl}i}$.
$\mathrm{E}_{\mathrm{ml\_mm}}(a, \hat{a})$	`ml_mismatch_energy`	bases a and $\hat{a}$ build a base-pair, on which the bases right of a and left of $\hat{a}$ dangle from the inner	Two bases on both inner sides of a multiloop closing stack may dangle from the inside onto this stack, but do not form a base-pair (mismatch).

Table 2.5: Special pseudoknot energy functions.

Function	GAP-C name	Structural assumptions	Meaning
$\mathrm{E}_{\mathrm{sr\_pk}}((a,\hat{a}),(b,\hat{b}))$	`sr_pk_energy`	bases a and $\hat{a}$, as well as b and $\hat{b}$ build base-pairs, where the first one stacks onto the second.	Models the stabilizing contribution of coaxial stacking, observed in H-type pseudoknots.
$\mathrm{E}_{\mathrm{init}_{pkml}}()$	`pkmlinit`	–	Destabilizing energy for integrating a pseudoknot into a multiloop. Set to +6.0 kcal/mol.
$\mathrm{E}_{\mathrm{init}_{pk}}()$	`pkinit`	–	Energetic penalty for opening an H-type pseudoknot. Default is +9.0 kcal/mol.
$\mathrm{E}_{\mathrm{init}_{pkiss}}()$	`pkissinit`	–	Energetic penalty for opening an K-type pseudoknot. Default is +12.0 kcal/mol.
$\mathrm{E}_{\mathrm{npp}}()$	`npp`	–	Energetic penalty for a single unpaired base inside a pseudoknot. Set to +0.1 kcal/mol.

2.1.3 Ambiguity

A grammar is called *ambiguous*, if the same input has two or more different derivations [70]. That likely causes problems, if you want to design a grammar to parse e.g. a programming language or an XML document, but it is a key ingredient for optimization problems. Here, we exploit the ambiguity to systematically explore the space of different derivations and seek for the "best" one. For example, all possible alignments of two fixed protein sequences form a defined search space such that the best one can be selected by measuring their similarity. Similarly, the stablest RNA secondary structures for a fixed RNA sequence can be determined. Ambiguity is "good" to choose from many solutions, but it may turn "bad" if some solutions are enumerated several times.

Imagine you arrive in a hotel after a long flight. Unfortunately, you forgot the correct combination of digits for the lock of your suitcase. Maybe you are lucky with some arbitrary combinations, but in the worst case you have to test all the 10^n sequences. A systematic approach such as grammar $\mathcal{G}_{l1}$ (see Figure 2.2) would be a better solution:

Every **comb**ination ends with a last *digit*. We can also create a multi-digit combination, by adding two sub-combinations next to each other. A *digit* parses one `c`haracter of the input sequence in ten different ways (0 to 9), thus creating the wanted "good" ambiguity. Opposite to the mentioned examples above, here the sequence just gives the *number* of digits for the lock, not a combination itself. For example, the input `xxxx` encodes a 4-digit problem. The very same problem could be encoded as `xyxy` since the input `c`haracters do not matter, only their number (here four). Grammar $\mathcal{G}_{l1}$ is syntactically ambiguous, since the input can be parsed in many different ways.

All combinations can now be enumerated in a systematical fashion with the help of $\mathcal{G}_{l1}$. For example, the combination `1409` is produced by the derivation l_a of Table 2.6, if you read the numbers in a depth-first ordering. Interpreting a derivation as a combination can be formally expressed by an algebra $\mathcal{I}_c$, given in Table 2.7. The good news is, that you will recover the unlocking combination for sure, i.e. the optimization problem can be solved correctly. Unfortunately, you are spending to much time with the lock, instead of enjoying your holidays, because every combination is enumerated several times! In

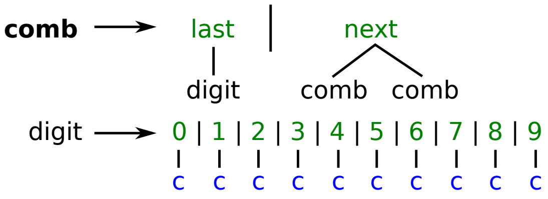
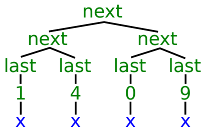
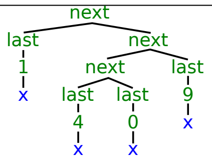
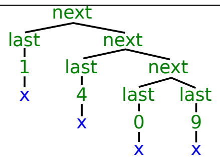
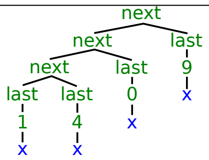
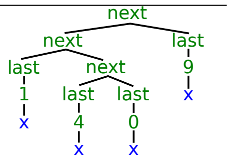


Figure 2.2: Grammar $\mathcal{G}_{l1}$ to enumerate all **comb**inations for a *n-digit* lock. Axiom is **comb**.

Table 2.6: All five derivations, whose canonical representation is the combination `1409`. See Algebra $\mathcal{I}_c$ of Table 2.7 for the mapping from a derivation to a canonical representation. Input sequence is `xxxx` for a 4-digit lock. Applying $\mathcal{I}_c$ to $\mathcal{G}_{l1}$ reveals the grammars semantic ambiguity regarding *combinations*, since different derivations map to the same combination.

l_a	l_b	l_c	l_d	l_e
				

fact, all five derivation l_a to l_e of Table 2.6 point to the very same combination `1409`, just with different internal structures, which does not impress the lock.

While a combination is the meaning of a real world object, the derivation has – in principle – no interpretation. We call the meaning the *semantics*, and the result of mapping a derivation to its meaning a *canonical representation*. Let us refine our definition to distinguish "good" (syntactic) and "bad" (semantic) ambiguity:

Definition 9. *A grammar is called* syntactically ambiguous *if there is more than one derivation for a given input.*

Definition 10. *A grammar is* semantically ambiguous *regarding a given semantics, if two derivations have the same canonical representation, i. e. the semantic mapping is not injective.*

When solving optimization problems, semantic ambiguity often does not hurt. The optimal value may be computed by several derivations but can still be correctly determined. Problems arise, if we want to e. g.

- report a meaningful list of sub-optimal solutions. This list will be polluted by many duplicates.
- say something about the number of different meanings, e. g. counting them. The problem is, that the search space is larger than the space of different meanings.

Table 2.7: Algebra $\mathcal{I}_c$ to formally define the semantics of a *combination*. Operator $\oplus$ is string concatenation.

algebra function	$\mathcal{I}_c$
last(d)	d
next(x, y)	$x \oplus y$
0(c)	0
⋮	⋮
9(c)	9
$\mathcal{S}$	string
objective function	id

- compute a most likely meaning. A meaning might be composed of several derivations. In this case its probability is not identical with the probability of the most likely derivation. We will not be able to observe this situation, but a Viterbi algorithm must fail. That might be manageable for the lock example, where every combination has the same number of different derivations. But what about grammars where different meanings have different numbers of derivations? In general, semantic ambiguity most likely ruins all forms of stochastical analysis! We will face this situation, e. g. for computing RNA shape probabilities (see Chapter 3.2.5) or matching an RNA sequence against an RNA family model (see Chapter 7.3).

In conclusion, it is better to avoid semantic ambiguity in the first place. Unfortunately, checking a given grammar, together with a formally defined semantics, for being semantically ambiguous is – in general – an undecidable problem [70].

Coming back to our example, the ambiguity of $\mathcal{G}_{l1}$ is easily avoided by replacing next(comb, comb) by e. g. adddigit(digit, comb), because now a further digit can only be added to a combination in a single fashion.

3 Lost in folding space?

This chapter follows our BMC Bioinformatics publication [45]. The test data have been replaced with longer sequences, algorithms are described more elaborately and additional influencing factors like energy parameter set and lonely base-pairs are investigated.

3.1 Background

3.1.1 Motivation

A wide variety of bioinformatics tools exist, which help to analyze RNA secondary structure based on an experimentally supported thermodynamic model of RNA folding [61]. Typical tasks performed by such tools are

- prediction of a single, "optimal" structure of minimal free energy,
- computation of near-optimal structures, either by complete enumeration up to a certain energy threshold, or by sampling from the folding space,
- computation of base-pair probabilities and dot plots,
- computation of representative structures of different abstract shapes,
- computation of Boltzmann probabilities, either of individual structures or accumulated over all structures of the same abstract shape.

From a macroscopic point of view, all these approaches are based on the same thermodynamic model, but when checking in detail, this does not hold. Algorithms for different tasks make certain assumptions about the folding space, where little is known to which extent these assumptions influence the outcome of the analysis.

The present study is designed to fill this gap. We explicate the details of four different models of the RNA folding space, named NoDangle, OverDangle, MicroState and MacroState. They capture four different models of the folding space, as they are implemented in the programs RNAFOLD [40], RNASHAPES [34], and RNASUBOPT [95]. [1] We compare the outcome of predictions from the different models, and evaluate them against a widely trusted data set derived from experimentally proven structures.

[1] Our observations may pertain also to other popular programs such as MFOLD [97], UNAFOLD [55] and RNASTRUCTURE [75], but their folding space implementations have not been re-modeled here.

3.1.2 Goals of the evaluation

The goal of this study is not to define a “correct” or “best” way of modeling the RNA folding space. Different definitions may retain their merits in the light of different computational constraints. We want to explicate the differences in the results which are due to the choice of a particular model. Aside being interesting in its own right, this allows future algorithm designers to make a well-founded choice of the model they base their work on.

How to compare the performance of different models? A first idea would be to evaluate them with respect to prediction of the structure of minimum free energy (MFE; for details see below), using a reference set of trusted structures. This has been done occasionally [20, 61], and we will include such an evaluation here for the sake of completeness. However, MFE structure prediction is notorious in the sense that a slight offset in energy can lead to a radically different structure. This is a consequence of the underlying thermodynamic model, and not due to its inadequate implementation. For a more robust evaluation, we need a measure which constitutes a more comprehensive characteristic of the overall folding space of an RNA molecule, including evidence for competing near-optimal structures of significant structural variation.

Abstract shapes of RNA [34, 91] provide such a measure. This approach provides two essential types of analysis: (1) to compute a handsome set of representative, near-optimal structures, which are different enough to be of interest, and (2) to compute shape probabilities, which accumulate individual Boltzmann probabilities over all structures of the same shape. The shape probability is a robust measure of structural well-definedness, and in contrast to folding energy, it is independent of base composition and meaningful for comparing foldings of different sequences with similar length.

Types (1) and (2) of abstract shape analysis are achieved by different algorithms, using different models of the folding space in the program RNAshapes. A similar situation prevails within the Vienna-RNA-Package, where different models of the folding space are used with various functions of RNAfold and RNAsubopt under different parameter settings.

For our evaluation, we implement probabilistic shape analysis in four different ways, three of which closely correspond to the folding space models implemented for MFE prediction in RNAfold[2], and two of which correspond to the algorithms used in RNAshapes. This set of programs will allow us to derive observations about the underlying folding space models.

3.2 Methods

In this section, we recall the definitions underlying the thermodynamic model of RNA folding, and then proceed to specify four different implementations of this model.

[2] One may view our re-engineering as adding shape probability functionality to the Vienna-RNA-Package from outside.

3.2.1 Free energy and partition function

Structure formation of a single-stranded nucleic acid sequence x—from an unfolded, random coil structure c into the folded structure s—is a standard equilibrium reaction with temperature-dependent free energy ΔG^0_T and equilibrium constant K_T:

$$\begin{aligned} c &\rightleftharpoons s \\ K_T &= \frac{[s]}{[c]} \\ \Delta G^0_T &= -RT \ln K_T \ . \end{aligned}$$

The number of possible secondary structures of a single sequence, i. e. the folding space $F(x)$ of x, grows exponentially with the sequence length n [93, 66]. These possible structures s_i of a single sequence coexist in solution with concentrations dependent on their free energies $\Delta G^0(s_i)$; that is, each structure is present as a fraction p_{s_i} according to its Boltzmann probability

$$p_{s_i} = \exp\left(-\frac{\Delta G^0_T(s_i)}{RT}\right) / Q$$

given by its molar Boltzmann weight $\exp(-\Delta G^0_T(s_i)/(RT))$ and the partition function Q for the ensemble of all possible structures

$$Q = \sum_{\text{all structures } s_i \in F(x)} \exp\left(-\frac{\Delta G^0_T(s_i)}{RT}\right) \ .$$

The structure of lowest free energy is called the (thermodynamically) optimal structure or structure of minimum free energy (MFE). Variable R is the universal gas constant (0.00198717 kcal/K) and T is the temperature in Kelvin.

3.2.2 Implementing the energy model

The free energy of a given secondary structure s is obtained by decomposition of s into its structural elements and summation of values obtained by respective calls of the elementary energy functions of these elements as listed in Table 2.4 on page 25. With the example shown in Figure 3.1, this would be three calls to E_{sr} for the three base-pair stacks ($\substack{^{5'}\mathsf{AC}^{3'} \\ _{3'}\mathsf{UG}_{5'}}$, $\substack{^{5'}\mathsf{CC}^{3'} \\ _{3'}\mathsf{GG}_{5'}}$, and $\substack{^{5'}\mathsf{XY}^{3'} \\ _{3'}\mathsf{YX}_{5'}}$), a call to $\mathrm{E}_{\mathrm{termau}}$ for the terminal $\substack{^{5'}\mathsf{A} \\ _{3'}\mathsf{U}}$ pair, and a call to E_{bl} for a bulge loop with sequence $^{5'}\mathsf{N\text{---}N}^{3'}$ and closing pairs $\substack{^{5'}\mathsf{C} \\ _{3'}\mathsf{G}}$ and $\substack{^{5'}\mathsf{Y} \\ _{3'}\mathsf{X}}$.

In addition to the basic energy model described above, unpaired bases at the end of a helix can stabilize the helix by stacking on the terminal base-pair [12, 68, 49] [3]. Introducing dangling bases effectively refines our notion of structure. Any secondary structure,

[3] Similarly, stacking of helices [92, 96, 61] can further contribute free energy. This aspect is not considered here.

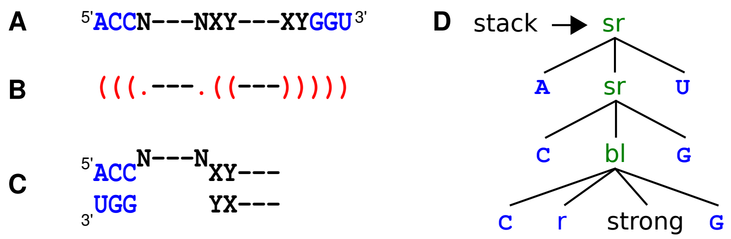


Figure 3.1: Example on structure representations. A sequence, shown in A), folds into a structure that is represented by the three equivalent illustrations in B–D). The structure consists of a helix with three base-pairs (ACC paired with GGU), a bulge loop (N---N; N meaning aNy nucleotide), and a helix with two base-pairs formed by any complementary nucleotides. The dashes designate omitted sequence stretches. The structure in B) is in Vienna-Dot-Bracket notation. The structure in C) is the usual squiggly representation. D) is the tree representation of the same structure: a stacked region (sr) is formed by an A:U pair stacked on top of a second stacked region C:G on-top of a bulge loop (bl) including a stacking pair (C:G) and a loop region with one or more residues (r) on the left (5') side. The helix continues with a "strong" structural element (which is defined as any sub-structure starting with a base stack).

as defined solely by its set of base-pairs, can now have several variants according to different choices of dangling bases. Such a refinement can be reflected in our structure representation by replacing certain dot symbols by d, indicating a base dangling onto a helix to its left, and b for a base dangling onto a helix to its right. For example, a structure like

((..((...)).((...)).))

now has dangle variants such as

((d.((...))b((...))b))

((.b((...))b((...))b))

((db((...))b((...))b))

((..((...))d((...))b))

((..((...))b((...)).))

and 31 more. Each end of a helix can have dangling bases, except an end which leads to the hairpin loop. In this case, energy contributions from dangling bases are already incorporated in the energy parameters (E_{hl}) for the loops.

Given a concrete secondary structure, it is no problem to consider all possible dangles and compute the optimal energy for this structure. The program RNAEval from the Vienna-Package can be used for this purpose. However, for structure prediction from a primary RNA sequence, dangle means trouble, as we shall see shortly.

Grammars and their relation to established structure prediction programs

All approaches using the thermodynamic model are implemented via dynamic programming. Recursively, structures are composed from smaller sub-structures. Such a dynamic programming algorithm always has an underlying grammar, which describes all the candidates in the folding space of a given RNA sequence. Hence, by extracting the grammars behind different algorithms, we can analyze the differences in their respective folding space in a precise way, and without obscuring implementation detail.

We will present four grammars, $\mathcal{G}_{\text{NoDangle}}$, $\mathcal{G}_{\text{OverDangle}}$, $\mathcal{G}_{\text{MicroState}}$ and $\mathcal{G}_{\text{MacroState}}$. The first three implement the folding space of RNAFOLD used with options -d0, -d2, and -d1, respectively. The grammars $\mathcal{G}_{\text{MicroState}}$ and $\mathcal{G}_{\text{MacroState}}$ implement the folding space of RNASHAPES in its two functions. All four grammars will then be empowered with algebras for MFE prediction ($\mathcal{I}_{mfe}$), dot-bracket representation ($\mathcal{I}_{db}$), shape abstraction ($\mathcal{I}_{\pi_i}$) for five different levels i, Boltzmann weighted energies ($\mathcal{I}_{bwe}$), and are used in our evaluation for computing shape probabilities (via the algebra product $\mathcal{I}_{\pi_i} * \mathcal{I}_{bwe}$) under the different models.

3.2.3 Model NoDangle

Grammar $\mathcal{G}_{\text{NoDangle}}$ is our incorporation of the elementary energy model, without considering dangling bases at all. It corresponds to the model underlying RNAFOLD when used with option -noLP -d0 [4]. It is also used in RNASUBOPT. We give a narrative explanation of how this grammar of Figure 3.2 works.

Each term, which represents a secondary structure, is a **struct**, i. e. it is derived from the axiom of $\mathcal{G}_{\text{NoDangle}}$. It might have leading unpaired bases (sadd), hold one or more closed sub-structures (cadd), or just ends with the empty word (nil). A *dangle* is a closed sub-structure whose directly neighbored bases might dangle onto the stack of base-pairs. We keep the name *dangle* for consistency with the other grammars, but no dangle energies are considered in $\mathcal{G}_{\text{NoDangle}}$; A stack must have *strong* stabilizing energies, if we forbid lonely base-pairs, i. e. at least two directly nested base-pairs must stack on top of each other. Otherwise, *strong* is just a forwarding to *weak*, a substructure enclosed by a base-pair, which eventually leads to one of five structural motifs: hairpin loop (hl), bulge to the left (bl), bulge to the right (br), internal loop (iloop) or multiloop (ml). The multiloop is a concatenation (*ml_comps* and *ml_comps*1) of two or more sub-structures, embraced by one closing stack. A helix initiated by *weak* can be elongated by sr.

Filter ≥ 3 requires the region of the hairpin-loop (hl) to be at least 3 bases long. Similar is the effect of ≤ 30 to regions. Annotations basepair beneath non-terminals mean that leftmost l and rightmost r bases of all right hand sides must be able to form a valid base-pair, i. e. $(l, r) \in \{(\texttt{A}, \texttt{U}), (\texttt{U}, \texttt{A}), (\texttt{C}, \texttt{G}), (\texttt{G}, \texttt{C}), (\texttt{G}, \texttt{U}), (\texttt{U}, \texttt{G})\}$. Filters $\neq$ LP and $=$ LP control presents of *L*onely base-*P*airs in the terms. A base-pair at positions (i, j) is *lonely* iff neither $(i-1, j+1)$ nor $(i+1, j-1)$ form another base-pair. Shall we want to model terms with lonely base-pairs, we set variable LP = **true**, hence production

[4] RNAFOLD-manual: "-d or -d0 ignores dangling ends altogether (mostly for debugging)."

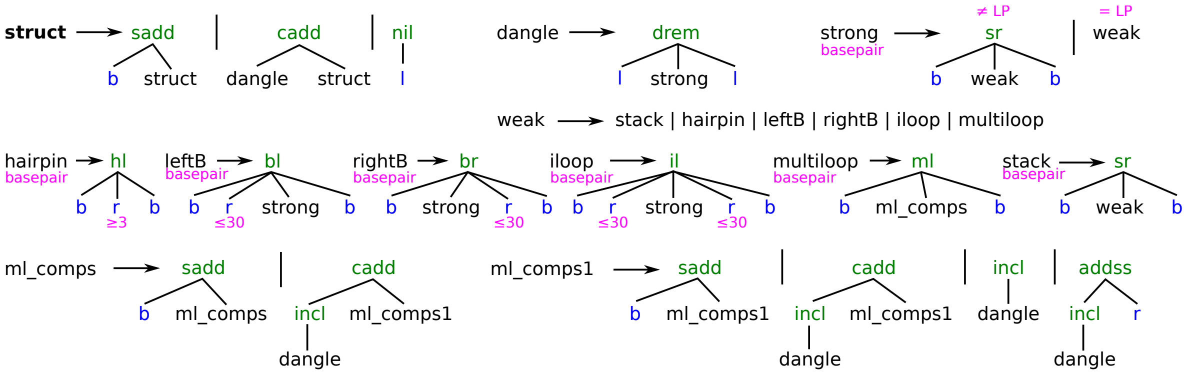


Figure 3.2: Regular tree grammar $\mathcal{G}_{\text{NoDangle}}$ for RNA secondary structure prediction. Axiom is **struct**. Functions are given in green, terminal parsers are `blue`, syntactic filters are pink. Search space $\mathcal{L}(\mathcal{G}_{\text{NoDangle}})$ is identical to RNAFOLD (with option `-d 0`), except both `r`egions of internal loops (il) *together* may not exceed 30 bases in RNAFOLD.

"*strong* → *weak*" can be applied, but not production "*strong* → sr(`b`, *weak*, `b`)". *LP* set to **false**, on the other hand, makes formation of lonely base-pairs impossible; which should be the default.

Automated table design of BELLMAN'S GAP for $\mathcal{G}_{\text{NoDangle}}$, results in the tabulation of non-terminals $\{struct, dangle, strong, weak, iloop, ml\_comps, ml\_comps1\}$. The other 5 non-terminals $\{hairpin, leftB, rightB, multiloop, stack\}$ are not tabulated, yielding a run-time[5] of $6 \cdot n^3 + 4{,}478 \cdot n^2 + 8 \cdot n + 6$.

3.2.4 Model OverDangle

Grammar $\mathcal{G}_{\text{OverDangle}}$ considers dangling base energies in a simplified form. It corresponds to RNAFOLD called with options -noLP -d2 [6]. The grammar itself is identical to $\mathcal{G}_{\text{NoDangle}}$ (cf. Figure 3.2). It computes the same folding space, but evaluates energies differently. It assumes an energy contribution from dangling bases on every side of a helix, even if a base is not available for dangling, for example because it is itself engaged in another helix, or already dangling there. The algebra functions drem and ml control the dangling behavior, which is the only difference between the models NoDangle and OverDangle. In the OverDangle model drem and ml always adds dangling energies for left and right dangles. This is why the production using drem uses two `l` terminal parser: `l` recognizes the empty word, and returns its position in the sequence. These positions are used by drem to look at the two bases to the left and right of the *strong* sub-structure.

[5] determined by the tool `multi_rt_approx` from the Bellman's GAP suite

[6] RNAFOLD-manual: "With -d2 this check is ignored, dangling energies will be added for the bases adjacent to a helix on both sides in any case; this is the default for partition function folding (-p)."

This "overdangling" model is used because a correct treatment of dangles is much more complicated, as we shall see below. As a plausibility argument in favor of this heuristic, one may say that when a base is overdangled, for example between two adjacent helices, as with the midpoint in `((...)).((...))`, this can be seen as a bonus for co-axial stacking of the two helices. Including full co-axial stacking could be considered as a further refinement of the folding space beyond the $\mathcal{G}_{\text{MicroState}}$ model, which will be described below. Still, due to overdangling, the MFE energy value computed may be smaller than actually assigned by the thermodynamic model to the underlying structure. Partition function computations in RNAFOLD use the OverDangle approach, and so does RNASUBOPT with option -d2 (and even -d1, but see below).

Would we use both NoDangle and OverDangle to produce a list of all structures in the folding space, sorted by free energy, these lists would hold the same structures, but in a different order. The true MFE structure (under the full model with correct dangles) will be near the front of each list, but it is not guaranteed to come out on first place. Our next two grammars are designed to achieve this goal.

3.2.5 Model MicroState

Grammar $\mathcal{G}_{\text{MicroState}}$ is a grammar which refines our model of a secondary structure. It corresponds to `RNAfold -noLP -d1` [7] and is used in the 2004 release of RNASHAPES [34] for the computation of representative structures of different shape.

$\mathcal{G}_{\text{MicroState}}$ has separate rules for a helix end with two bases, one base or no base dangling onto it (see Figure 3.3). These four cases compete with each other for minimum free energy. If surrounding bases are already base-paired, only the **drem** case applies (no dangles). If it is decided (say) that the left neighboring base dangles onto the helix, then this base is not available for also dangling on another helix. In this way, grammar $\mathcal{G}_{\text{MicroState}}$ correctly finds the structure of minimal free energy, and could, in principle, also explicitly report the optimal dangles, as in `..b((...))d((...))....`

All variants of the same secondary structure, augmented with different dangles, are now separate members of the folding space. Grammar $\mathcal{G}_{\text{MicroState}}$ is semantically ambiguous with respect to the Vienna-Dot-Bracket notation (see Section 2.1.3). In contrast to the classical model, accounting only for base-pairs, we call them "microstates". Let us derive a rough estimate of this folding space enlargement. The size of the folding space for a sequence of length n grows asymptotically with $a \cdot b^n \cdot n^{-3/2}$, with $b = 1.44358$ and $a = 3.45373$ [66]. A structure has, on average, $k(n)$ helices, where k grows with n. Each helix end has up to four ways to play with the dangles, but helix ends in hairpin loops do not count. Directly adjacent helices further reduce the number of dangling alternatives.

Let us, for simplicity, assume that an helix has 4 dangle variants on average. Then, the above formula changes for the number of microstates to $a \cdot 4^{k(n)} \cdot b^n \cdot n^{-3/2}$. An empirical measurement is shown in Figure 3.4. From the measurements, and for their particular data sequences and lengths, we can estimate $k(n) \approx \frac{n}{15}$. For a sequence of length 100, for

[7] RNAFOLD-manual: "With -d1 only unpaired bases can participate in at most one dangling end, this is the default for MFE folding but unsupported for the partition function folding."

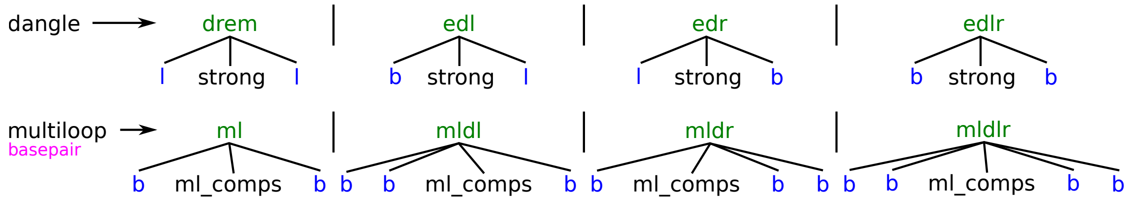


Figure 3.3: Grammar $\mathcal{G}_{\text{MicroState}}$ extends the rules of grammars $\mathcal{G}_{\text{NoDangle}}$ or $\mathcal{G}_{\text{OverDangle}}$ (Figure 3.2) for the non-terminal symbols "dangle" and "multiloop". Instead of just one way, we now have four alternatives to dangle bases onto a closed sub-structure: Both neighboring bases do not dangle (drem and ml), only the left neighbored base dangles onto the stack (edl and mldl), only the right one (edr and mldr), or both ones (edlr and mldlr).

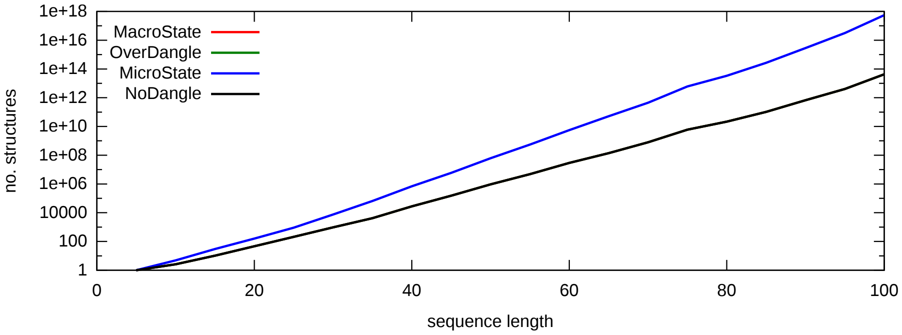


Figure 3.4: Growth of folding spaces for all four grammars. We used uniformly distributed random sequences, with step-size 5 bp. The number of secondary structures heavily depends on sequence composition, thus we took the average over 100 sequences per data point. Curves for $\mathcal{G}_{\text{MacroState}}$ and $\mathcal{G}_{\text{OverDangle}}$ are not visible, because they are perfectly overlayed by $\mathcal{G}_{\text{NoDangle}}$, i. e. all three folding spaces have exactly the same size.

example, we see an increase by a factor of 10^4. Clearly, this is a substantial enlargement of the folding space, and different structures are affected to a different extent. (For example, the open structure (no base-pairs) gives rise to only one microstate.)

This enlargement of the search space is not a problem for MFE structure prediction. The dynamic programming algorithm derived from $\mathcal{G}_{\text{MicroState}}$ only does a constant amount of extra work compared to $\mathcal{G}_{\text{NoDangle}}$ and $\mathcal{G}_{\text{OverDangle}}$. But a severe problem arises with the desire to investigate near-optimal structures. The roughly 4^k microstates of an optimal structure with k helices crowd the near-optimal folding space, while representing the same structure in the non-dangling sense. Enumerating sub-optimals returns a tremendous amount of useless information. RNAsubopt therefore uses $\mathcal{G}_{\text{OverDangle}}$ for enumeration, even when option -d1 is specified. Afterwards, it re-evaluates the energy of predicted structures using correct dangling. Hence, the ranking of structures may change. Occasionally, we observe that the energy of the true MFE structure is so much above the energy of other, overdangled structures that it falls above the energy threshold for enumeration and is not returned at all. [8]

The second problem arises with computations that are based on Boltzmann statistics. The partition function Q sums up the Boltzmann-weighted energies of all members in the folding space. Each secondary structure contributes to the partition function as many times as it has microstates, hence the result would be skewed towards structures with many microstates. The significance of this bias is hard to judge [9], and up to this study, it could not be evaluated empirically. For this reason, RNAFOLD does not support partition function computation with the MicroState model (option -d1).

Fortunately, the partition function with correct dangles, avoiding overdangling as well as explosion of the folding space, can also be computed. To keep the folding space simple, we need a more sophisticated grammar: $\mathcal{G}_{\text{MacroState}}$.

3.2.6 Model MacroState

Grammar $\mathcal{G}_{\text{MacroState}}$ (see Figure 3.2.6) follows the overall pattern of the other grammars, but is much more refined. This grammar was designed originally with the 2006 release of RNASHAPES [91] to compute complete probabilistic shape analysis. Its rules are written to record and distinguish the situation where a helix (1) ends with a base-pair, (2) already has a single unpaired base to its right or left, or (3) has several unpaired bases on either side. No dangle energies are added in cases (1) and (3), and in case (2), all possible dangle variants (up to four microstates) are evaluated and minimized over while considering the corresponding macrostate. This leads to a much larger number of non-terminal symbols and functions in the grammar. $\mathcal{G}_{\text{MacroState}}$ has 26 non-terminal symbols and 32 algebra functions, compared to $\mathcal{G}_{\text{NoDangle}}$ with 12 non-terminals and 12 algebra functions.

The important feature of $\mathcal{G}_{\text{MacroState}}$ is that for any sequence, it defines the identical folding space as $\mathcal{G}_{\text{NoDangle}}$. This is hard to believe when just looking at the grammar, but has been shown in [91], and is further demonstrated by the measurements shown in Figure 3.4. The size of the folding space, as defined by $\mathcal{G}_{\text{MacroState}}$, agrees with that of $\mathcal{G}_{\text{NoDangle}}$ and $\mathcal{G}_{\text{OverDangle}}$ not only on average, but also on each individual sequence.

What is the effect of using either $\mathcal{G}_{\text{MicroState}}$ or $\mathcal{G}_{\text{MacroState}}$? Does it really matter? Here is an extreme example of how the choice of the state space affects the computed probabilities:

`GACCAAAGCCUUUGUCCCACAAAUUGCGAUCGCGUCGCGGAGC`

MacroState prob.	MicroState prob.	shape class
58.44%	32.58%	`[][]`
29.32%	63.43%	`[[][]]`
12.24%	03.99%	`[]`

[8] A larger threshold will always help. However, one cannot tell whether this situation has occurred.

[9] Whether or not it is adequate in partition function computations to split a secondary structure into several microstates is an unresolved dispute among experts (M. Zuker, personal communication).

In this example, 40% of the probability mass is shifted by switching models, causing the order of the two top-ranking shapes to be reversed. To find out whether this situation is the exception or the rule is a main motivation of this study.

MacroState's conception

The basic structure of $\mathcal{G}_{\text{MacroState}}$ (Figure 3.2.6) is inherited from the previous three grammars, but it has a more complex distinction of cases for dangling bases.

$\mathcal{G}_{\text{MacroState}}$ has to consider all the different dangling situations as in $\mathcal{G}_{\text{MicroState}}$, but its search space is restricted to the $k(n)$-times smaller folding space of the input sequence. To achieve these contradicting goals, dangling alternatives do not exist as search space candidates but are implicitly examined within the evaluation algebra.

Grammar $\mathcal{G}_{\text{MacroState}}$ has to ensure that a sub-structure is of a defined dangling type whenever its energy or partition function value is used in an algebra evaluation function. We know that any helix derived from *noleft_dangle* has no unpaired bases to its left or right, while helices from *edanglel*, *edangler* or *edanglelr* have exactly one unpaired base dangling from left, right or exactly two unpaired bases dangling from both sides, respectively. In all four cases, there is no unpaired base left for a further dangling. Care must be taken, where we can not be sure if e. g. the leftmost unpaired base of a *block_dl* derivation is free to dangle to some helix to its left. The unpaired base would be available for a dangling if we use ssadd, but is occupied in incl situations. This uncertainty is passed to every calling function, but with a clever grammar design we can at least ensure that its type does not change. For example every *ml_comps1* or *dl_or_ss_left_no_ss_end* derivation contains one or more helices with one or more unpaired bases at its 5' end and definitely no unpaired base at its 3' end. Furthermore *ml_comps2* and *no_dl_no_ss_end* always have no unpaired bases to both sides, *ml_comps3* or *no_dl_ss_end* have one or more unpaired bases only at its 3' end and finally *ml_comps4* or *dl_or_ss_left_ss_end* are known to have one or more unpaired bases to both ends. The benefit of these distinctions can be demonstrated with the multiloop functions mldl and mladl. The important base is the one that is directly left to the *ml_comps1* or *ml_comps2* sub-structure. In principle, it can either dangle to the left, that is the closing stem of the multiloop, or the right, that is the leftmost helix within the multiloop. Actually, for mldl our base of interest can only dangle to the left, because every *ml_comps1* derivation already has at least one further base in front of the first inner helix. For mladl we truly have an **a**mbiguous situation, where the base of interest could dangle to one of both sides. Please note that mldl and mladl correspond to two different dot-bracket structures. mldl handles macrostates of the type `((...` including microstates `((...` and `((d..`, whereas mladl handles macrostates of type `((.((...` and includes the microstates `((.((...`, `((d((...`, and `((b((...`. The MFE algebra function locally chooses the variant with the better free energy, even if a global analysis would reveal that the locally worse structure would become MFE in the end. This constitutes a rare case where the MFE structure may be missed. Our partition function algebra correctly keeps track of these situations.

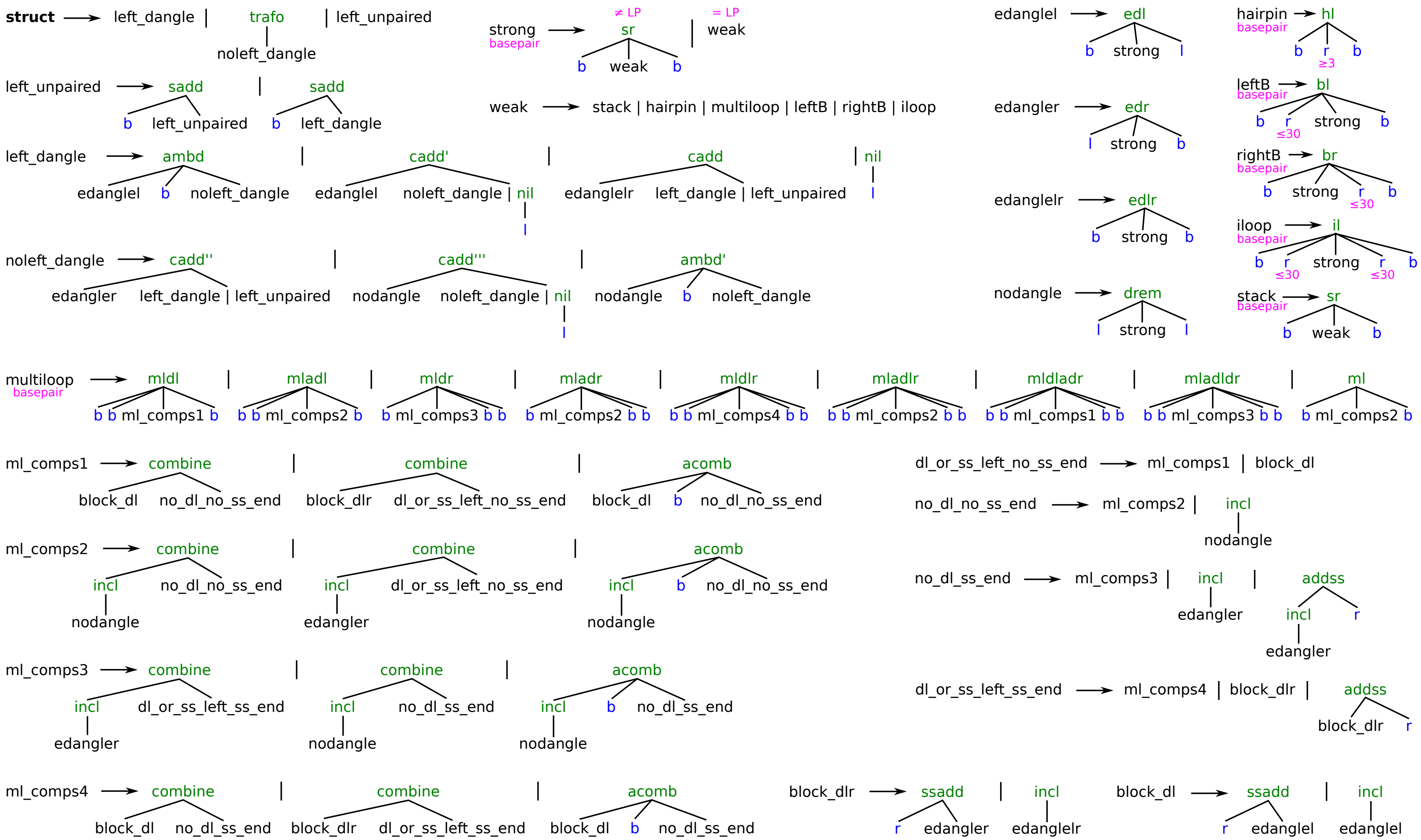

struct → left_dangle | trafo (noleft_dangle) | left_unpaired
left_unpaired → sadd (b, left_unpaired) | sadd (b, left_dangle)
left_dangle → ambd (edanglel, b, noleft_dangle) | cadd' (edanglel, noleft_dangle | nil (l)) | cadd (edanglelr, left_dangle | left_unpaired) | nil (l)
noleft_dangle → cadd'' (edangler, left_dangle | left_unpaired) | cadd''' (nodangle, noleft_dangle | nil (l)) | ambd' (nodangle, b, noleft_dangle)
strong basepair → ≠ LP sr (b, weak, b) | = LP weak
weak → stack | hairpin | multiloop | leftB | rightB | iloop
edanglel → edl (b, strong, l)
edangler → edr (l, strong, b)
edanglelr → edlr (b, strong, b)
nodangle → drem (l, strong, l)
hairpin basepair → hl (b, r ≥3, b)
leftB basepair → bl (b, r ≤30, strong, b)
rightB basepair → br (b, strong, r ≤30, b)
iloop basepair → il (b, r ≤30, strong, r ≤30, b)
stack basepair → sr (b, weak, b)
multiloop basepair → mldl (b b ml_comps1 b) | mladl (b b ml_comps2 b) | mldr (b ml_comps3 b b) | mladr (b ml_comps2 b b) | mldlr (b b ml_comps4 b b) | mladlr (b b ml_comps2 b b) | mldladr (b b ml_comps1 b b) | mladldr (b b ml_comps3 b b) | ml (b ml_comps2 b)
ml_comps1 → combine (block_dl, no_dl_no_ss_end) | combine (block_dlr, dl_or_ss_left_no_ss_end) | acomb (block_dl, b, no_dl_no_ss_end)
ml_comps2 → combine (incl (nodangle), no_dl_no_ss_end) | combine (incl (edangler), dl_or_ss_left_no_ss_end) | acomb (incl (nodangle), b, no_dl_no_ss_end)
ml_comps3 → combine (incl (edangler), dl_or_ss_left_ss_end) | combine (incl (nodangle), no_dl_ss_end) | acomb (incl (nodangle), b, no_dl_ss_end)
ml_comps4 → combine (block_dl, no_dl_ss_end) | combine (block_dlr, dl_or_ss_left_ss_end) | acomb (block_dl, b, no_dl_ss_end)
dl_or_ss_left_no_ss_end → ml_comps1 | block_dl
no_dl_no_ss_end → ml_comps2 | incl (nodangle)
no_dl_ss_end → ml_comps3 | incl (edangler) | addss (incl (edangler), r)
dl_or_ss_left_ss_end → ml_comps4 | block_dlr | addss (block_dlr, r)
block_dlr → ssadd (r, edangler) | incl (edanglelr)
block_dl → ssadd (r, edanglel) | incl (edanglel)

Figure 3.5: Grammar $\mathcal{G}_{\text{MacroState}}$ unambiguously enumerates the folding space with respect to the Vienna-Dot-Bracket notation and treats dangling bases in a nearly correct fashion as does $\mathcal{G}_{\text{MicroState}}$. It is correct for partition function computation, but it violates Bellman's Principle of Optimality (cf. Section 2.1.1) in the case of MFE prediction.

Table 3.1: Signature Σ_{RNA} for RNA problems. Result type of all algebra functions is sort $\mathcal{S}$. $\mathcal{A}$ is a symbol from the alphabet, i. e. a single character from the input sequence. $\mathcal{A}^*$ stands for a non-empty sub-sequence from the input and $\mathcal{A}^0$ shall denote reading the position within the input, but not consuming any symbol.

sadd($\mathcal{A}, \mathcal{S}$)
cadd($\mathcal{S}, \mathcal{S}$)
nil($\mathcal{A}^0$)
drem($\mathcal{A}^0, \mathcal{S}, \mathcal{A}^0$)
edl($\mathcal{A}, \mathcal{S}, \mathcal{A}^0$)
edr($\mathcal{A}^0, \mathcal{S}, \mathcal{A}$)
edlr($\mathcal{A}, \mathcal{S}, \mathcal{A}$)
sr($\mathcal{A}, \mathcal{S}, \mathcal{A}$)
hl($\mathcal{A}, \mathcal{A}^*, \mathcal{A}$)
bl($\mathcal{A}, \mathcal{A}^*, \mathcal{S}, \mathcal{A}$)
br($\mathcal{A}, \mathcal{S}, \mathcal{A}^*, \mathcal{A}$)
il($\mathcal{A}, \mathcal{A}^*, \mathcal{S}, \mathcal{A}^*, \mathcal{A}$)
ml($\mathcal{A}, \mathcal{S}, \mathcal{A}$)
mldl($\mathcal{A}, \mathcal{A}, \mathcal{S}, \mathcal{A}$)
mldr($\mathcal{A}, \mathcal{S}, \mathcal{A}, \mathcal{A}$)
mldlr($\mathcal{A}, \mathcal{A}, \mathcal{S}, \mathcal{A}, \mathcal{A}$)
addss($\mathcal{S}, \mathcal{A}^*$)
incl($\mathcal{S}$)
cadd’($\mathcal{S}, \mathcal{S}$)
cadd”($\mathcal{S}, \mathcal{S}$)
cadd”’($\mathcal{S}, \mathcal{S}$)
ambd($\mathcal{S}, \mathcal{A}, \mathcal{S}$)
ambd’($\mathcal{S}, \mathcal{A}, \mathcal{S}$)
mladl($\mathcal{A}, \mathcal{A}, \mathcal{S}, \mathcal{A}$)
mladr($\mathcal{A}, \mathcal{S}, \mathcal{A}, \mathcal{A}$)
mladlr($\mathcal{A}, \mathcal{A}, \mathcal{S}, \mathcal{A}, \mathcal{A}$)
mldladr($\mathcal{A}, \mathcal{A}, \mathcal{S}, \mathcal{A}, \mathcal{A}$)
mladldr($\mathcal{A}, \mathcal{A}, \mathcal{S}, \mathcal{A}, \mathcal{A}$)
ssadd($\mathcal{A}^*, \mathcal{S}$)
trafo($\mathcal{S}$)
combine($\mathcal{S}, \mathcal{S}$)
acomb($\mathcal{S}, \mathcal{A}, \mathcal{S}$)

3.2.7 Signature and evaluation algebras

In the previous sections, we developed four regular tree grammars ($\mathcal{G}_{\text{NoDangle}}$, $\mathcal{G}_{\text{OverDangle}}$, $\mathcal{G}_{\text{MicroState}}$ and $\mathcal{G}_{\text{MacroState}}$) to enumerate all secondary structures for a given RNA input sequence. Missing components for an executable Bellman’s GAP program are signature and evaluation algebras, which will be presented in the following.

The signature defines the set of functions that can be used in the production rules of a grammar (see Definition 2). A grammar does not need to incorporate *all* these functions. Thus, we can specify just one common signature Σ_{RNA} (see Table 3.1) for all grammars, where e. g. $\mathcal{G}_{\text{NoDangle}}$ will use only 12 functions, while the most complex grammar $\mathcal{G}_{\text{MacroState}}$ needs all 32 functions.

Using a common signature comes with the advantage that we have to implement some of the evaluation algebras also just once. The Vienna-Dot-Bracket algebra $\mathcal{I}_{db}$ (cf. Section 2.1.1) and the five shape abstraction algebras $\mathcal{I}_{\pi_i}$ $\forall\ 5 \geq i \geq 1$ will work for all four grammars (see Table 3.2).

Abstract shapes

Many of the possible secondary structures for an RNA sequence differ from each other by only tiny structural rearrangements like addition or removal of a base-pair, or a slight shift in position of a small bulge loop. Structures can be pooled according to their abstract shape. Generally, an abstract shape gives information about the arrangement of structural elements such as helices, but no concrete base-pairs [34, 91]. The MFE structure within each shape class is called “shrep”, which is short for shape representative structure. The partition function Q_p for the ensemble of all structures of shape p is

$$Q_p = \sum_{\text{all structures } s_i \in p} \exp\left(-\frac{\Delta G_T^0(s_i)}{RT}\right) .$$

Table 3.2: Algebras for terms of Σ_{RNA}. Result of $\mathcal{I}_{db}$ is a Vienna-Dot-Bracket string. The other five algebras $\mathcal{I}_{\pi_5}$ to $\mathcal{I}_{\pi_1}$ realize the abstract shape concept, by reducing a candidate t to a shape-string. For the sake of readability, we omit the string concatenation operator symbol $\oplus$ between every two parts of the result. ε is the empty string and $\dots_{|r|}$ means one ‘.’ for each base in r. In the case of the shape algebras $\mathcal{I}_{\pi_i}$, the concatenation must be implemented such that _ $\oplus$ _ $=$ _ is satisfied, i. e. adjacent symbols for unpaired regions are fused.

algebra function	$\mathcal{I}_{db}$	$\mathcal{I}_{\pi_5}$	$\mathcal{I}_{\pi_4}$	$\mathcal{I}_{\pi_3}$	$\mathcal{I}_{\pi_2}$	$\mathcal{I}_{\pi_1}$
sadd(a, x)	. x	_ x	_ x	_ x	_ x	_ x
cadd(x, y)	x y	x y	x y	x y	x y	x y
nil(l)	ε	ε	ε	ε	ε	ε
drem(l_1, x, l_2)	x	x	x	x	x	x
edl(a, x, l)	. x	x	x	x	x	_ x
edr(l, x, b)	x .	x	x	x	x	x _
edlr(a, x, b)	. x .	x	x	x	x	_ x _
sr$(a, x, \hat{a})$	(x)	x	x	x	x	x
hl$(a, r, \hat{a})$	($\dots_{\vert r\vert}$)	[ ]	[ ]	[ ]	[ ]	[ ]
bl$(a, r, x, \hat{a})$	($\dots_{\vert r\vert}$ x)	x	x	[x]	[_ x]	[_ x]
br$(a, x, r, \hat{a})$	(x $\dots_{\vert r\vert}$)	x	x	[x]	[x _]	[x _]
il$(a, r_1, x, r_2, \hat{a})$	($\dots_{\vert r_1\vert}$ x $\dots_{\vert r_2\vert}$)	x	[x]	[x]	[_ x _]	[_ x _]
ml$(a, x, \hat{a})$	(x)	[x]	[x]	[x]	[x]	[x]
mldl$(a, b, x, \hat{a})$	(. x)	[x]	[x]	[x]	[x]	[_ x]
mldr$(a, x, b, \hat{a})$	(x .)	[x]	[x]	[x]	[x]	[x _]
mldlr$(a, b, x, c, \hat{a})$	(. x .)	[x]	[x]	[x]	[x]	[_ x _]
addss(x, r)	x $\dots_{\vert r\vert}$	x	x	x	x	x _
incl(x)	x	x	x	x	x	x
cadd’(x, y)	x y	x y	x y	x y	x y	x y
cadd”(x, y)	x y	x y	x y	x y	x y	x y
cadd”’(x, y)	x y	x y	x y	x y	x y	x y
ambd(x, b, y)	x . y	x y	x y	x y	x y	x _ y
ambd’(x, b, y)	x . y	x y	x y	x y	x y	x _ y
mladl$(a, b, x, \hat{a})$	(. x)	[x]	[x]	[x]	[x]	[_ x]
mladr$(a, x, c, \hat{a})$	(x .)	[x]	[x]	[x]	[x]	[x _]
mladlr$(a, b, x, c, \hat{a})$	(. x .)	[x]	[x]	[x]	[x]	[_ x _]
mldladr$(a, b, x, c, \hat{a})$	(. x .)	[x]	[x]	[x]	[x]	[_ x _]
mladldr$(a, b, x, c, \hat{a})$	(. x .)	[x]	[x]	[x]	[x]	[_ x _]
ssadd(r, x)	$\dots_{\vert r\vert}$ x	x	x	x	x	_ x
trafo(x)	x	x	x	x	x	x
combine(x, y)	x y	x y	x y	x y	x y	x y
acomb(x, b, y)	x . y	x y	x y	x y	x y	x _ y
$\mathcal{S}$	string	string	string	string	string	string
choice function	id	unique	unique	unique	unique	unique

Of course, the structures from all shape classes sum up to the ensemble of all structures:

$$Q = \sum_{\text{all shape classes } p} Q_p$$

and the probability of shape p is

$$Prob(p) = Q_p / Q \ .$$

Shape abstraction can be defined in various ways. RNAShapes provides shape abstraction algebras $\mathcal{I}_{\pi_1}$ to $\mathcal{I}_{\pi_5}$ which implement different levels of abstraction, with $\mathcal{I}_{\pi_5}$ being the most abstract. Shapes can be represented as strings, similar to structure representations, where a single pair of square brackets `[ ]` marks a helix (of any length), and an underscore `_` marks a stretch of unpaired bases, also of any length. Levels of abstraction differ in the amount of information they retain about unpaired regions. The RNA structure `((.((((..(((...))).....((.((.....))...)).))))))` is mapped to shape strings on abstraction levels 2 and 5 as follows:

$$\begin{aligned} \mathcal{I}_{\pi_2} : &\quad \texttt{[\_[][\_[]\_]]]} \\ \mathcal{I}_{\pi_5} : &\quad \texttt{[[][]]} \end{aligned}$$

Both shapes indicate that the structure is a so-called Y-shape, a multiloop with a two-way branch. This most abstract view is conveyed by abstraction level 5. The less abstract level 2 shape indicates, in addition, that the outer stem is interrupted by a bulge on the 5' side, and that the 3' branch inside the multiloop is interrupted by an internal loop. For a detailed definition of shape abstraction levels, see Table 3.2.

MFE algebras

Differences between the four models of this study stem on the one hand from different grammars and on the other hand from different MFE algebras. We still use the common signature Σ_{RNA}, to re-use as many functions as possible, but we will need three different versions for the MFE algebras. A starting point is algebra $\mathcal{I}_{mfe}$ of Table 3.3. It suits for models NoDangle and MicroState. Since these grammars do not use all functions defined in Σ_{RNA}, we omit the unused ones here for the sake of space. For Bellman's GAP, they are of course implemented – in a dummy fashion: they simply return 0.

We have now all components at our disposal to create an RNA secondary structure prediction program for the NoDangle model: To resemble RNAfold with option `-d 0` for an input sequence x, we compile the Bellman's GAP instance

$$NoDangle: \quad \mathcal{G}_{\text{NoDangle}}(\mathcal{I}_{mfe} * \mathcal{I}_{db}, x).$$

The change to model MicroState – equivalent to RNAfold `-d 1` – is fairly easy, just replace the grammar

$$MicroState: \quad \mathcal{G}_{\text{MicroState}}(\mathcal{I}_{mfe} * \mathcal{I}_{db}, x).$$

For model OverDangle we need the variant $\mathcal{I}_{mfe}^{OverDangle}$ (Table 3.4) of the MFE algebra. Positions where a stem is initialized (drem and ml), additionally account dangling energies for both adjacent bases – available or not. All other algebra functions remain untouched. Thus, RNAFOLD `-d 2` agrees with

$$OverDangle: \quad \mathcal{G}_{\text{OverDangle}}(\mathcal{I}_{mfe}^{OverDangle} * \mathcal{I}_{db}, x).$$

The MFE algebra variant for model MacroState $\mathcal{I}_{mfe}^{MacroState}$ (Table 3.5) is not only complicated, but also violates Bellman's Principle of Optimality. We want to consider all dangling variants as in $\mathcal{G}_{\text{MicroState}}$, but have only the smaller search space of $\mathcal{G}_{\text{NoDangle}}$. Thus, within one macrostate, several microstates have to be explored, for example when combining one closed component x to a sequence of closed components y, with one unpaired base b in-between: $\text{ambd}(x, b, y)$.

Base b can either dangle from right onto the stem of x (E_{dr}) or from left onto the first stem of y (E_{dl}). By grammar construction, base left of b must be part of the outermost base-pair of the stem in x; as well as the base right of b is part of the outermost base-pair of the first stem in y. Since x might have leading unpaired bases in front of the stem, we cannot know the left partner of the crucial base-pair. The same holds for y, which can hold several adjacent stems. The base-pair partner will be hidden somewhere in y. The only chance to keep track of the important base-pairs is to use a special answer type. Instead of one integer value for the free energy, we switch to a five-tuple of integers. The five components of an answer x stand for x_1: free energy, x_2: left border of first stem, x_3: right border of first stem, x_4: left border of last stem and finally x_5: right border of last stem. With this additional information, we can determine the correct dangling energies: $\text{E}_{\text{dr}}(x_4, x_5)$ or $\text{E}_{\text{dl}}(y_2, y_3)$.

Minimizing within an algebra function instead of creating real candidates can lead to wrong results, as we will see in the following example: Let our input sequence be xay = `XXZZZZZYYaGGGGAAACUCUC`. The left part up to the lower case `a`, which is the unpaired base between x and y, shall be consumed by x in the form of a simple hairpin-loop `((.....))` with free energy of – say -3.0 $kcal/mol$. The following two alternative secondary structures must have been enumerated when determining the optimal solution(s) for y: i) `(((.....))).` with 1.3 $kcal/mol$ and ii) `(((......)))` with 1.6 $kcal/mol$. Of course, only i) would have been kept. Let us assume that `a` achieves better free energy if it dangles onto the right stem, i. e. the base-pair (`G`, `U`) with -0.3 $kcal/mol$, instead of dangling onto x. Then, the total free energy is $-3.0 + -0.3 + 1.3 = -2.0$ $kcal/mol$ for the secondary structure `((.....)).(((.....))).`

But what if dangling base `a` onto a base-pair (`G`, `C`) would give -0.7 $kcal/mol$, like in alternative ii)? Total energy would be $-3.0 + -0.7 + 1.6 = -2.1$ $kcal/mol$ for the structure `((.....)).(((......)))`, which is better than the above one. This is a clear violation of the third rule of Bellman's Principle (see Section 2.1.1), since we cannot move application of choice function h down.

Dangling contributions have not too huge energy values, thus this effect is not often seen for real RNA inputs. Thus, we include the MacroState model into our MFE

Table 3.3: MFE evaluation algebra $\mathcal{I}_{mfe}$ for terms of Σ_{RNA}. Algebra functions cadd' to acomb are not shown, since they are not part of Grammars $\mathcal{G}_{\text{NoDangle}}$ and $\mathcal{G}_{\text{MicroState}}$. Detailed information about the energy functions E are given in Section 2.1.2. Terminals are represented with single letters, like a. In fact, they are tuples of starting and ending borders of the parsed sub-word (see Section 2.1.2). Thus, a_{+1} means shifting both borders one position to the right; a_{-1} is a shift to the left.

algebra function	$\mathcal{I}_{mfe}$
sadd(a, x)	$x + \text{E}_{\text{sbase}}$
cadd(x, y)	$x + y$
nil(l)	0
drem(l_1, x, l_2)	$x + \text{E}_{\text{termau}}(l_1, l_2)$
edl(a, x, l)	$x + \text{E}_{\text{termau}}(a_{+1}, l) + \text{E}_{\text{dl}}(a_{+1}, l)$
edr(l, x, b)	$x + \text{E}_{\text{termau}}(l, b_{-1}) + \text{E}_{\text{dr}}(l, b_{-1})$
edlr(a, x, b)	$x + \text{E}_{\text{termau}}(a_{+1}, b_{-1}) + \text{E}_{\text{ext\_mm}}(a_{+1}, b_{-1})$
sr$(a, x, \hat{a})$	$x + \text{E}_{\text{sr}}(a, \hat{a})$
hl$(a, r, \hat{a})$	$\text{E}_{\text{hl}}(r)$
bl$(a, r, x, \hat{a})$	$x + \text{E}_{\text{bl}}(r, \hat{a})$
br$(a, x, r, \hat{a})$	$x + \text{E}_{\text{br}}(a, r)$
il$(a, r_1, x, r_2, \hat{a})$	$x + \text{E}_{\text{il}}(r_1, r_2)$
ml$(a, x, \hat{a})$	$x + \text{E}_{\text{ml}} + \text{E}_{\text{ul}} + \text{E}_{\text{termau}}(a, \hat{a})$
mldl$(a, b, x, \hat{a})$	$x + \text{E}_{\text{ml}} + \text{E}_{\text{ul}} + \text{E}_{\text{termau}}(a, \hat{a}) + \text{E}_{\text{dli}}(a, \hat{a})$
mldr$(a, x, b, \hat{a})$	$x + \text{E}_{\text{ml}} + \text{E}_{\text{ul}} + \text{E}_{\text{termau}}(a, \hat{a}) + \text{E}_{\text{dri}}(a, \hat{a})$
mldlr$(a, b, x, c, \hat{a})$	$x + \text{E}_{\text{ml}} + \text{E}_{\text{ul}} + \text{E}_{\text{termau}}(a, \hat{a}) + \text{E}_{\text{ml\_mm}}(a, \hat{a})$
addss(x, r)	$x + \text{E}_{\text{ss}}(r)$
incl(x)	$x + \text{E}_{\text{ul}}$
$\mathcal{S}$	int
choice function	min.

evaluation, via the Bellman's GAP instance

$$MacroState: \quad \mathcal{G}_{\text{MacroState}}(\mathcal{I}_{mfe}^{MacroState} * \mathcal{I}_{db}, x).$$

Boltzmann weighted energy algebras

The second part of the evaluation is based on probabilities, or more precisely on shape class probabilities. Probability p_{s_i} of a single secondary structure s_i is its free energy $\Delta G_T^0(s_i)$, weighted according to Boltzmann's formula pf and divided by the sum of all Boltzmann weights, i. e. the "partition function" Q (see Section 3.2.1). Function $pf(x)$ is defined as $e^{\frac{-x/100}{R \cdot T}}$, where the division by 100 stems from the fact that energy parameters in the Vienna-Package are encoded times 100. Thus, **b**oltzmann **w**eighted

Table 3.4: MFE evaluation algebra $\mathcal{I}_{mfe}^{OverDangle}$ for the OverDangle model. It is identical to $\mathcal{I}_{mfe}$ of Table 3.3, except for the algebra functions drem and ml. Differences are marked with bold face.

algebra function	$\mathcal{I}_{mfe}^{OverDangle} \rightharpoonup\!\!\triangleright \mathcal{I}_{mfe}$
$\text{drem}(l_1, x, l_2)$	$x + \mathrm{E}_{\text{termau}}(l_1, l_2) \boldsymbol{+ \mathbf{E}_{\textbf{ext\_mm}}(l_1, l_2)}$
$\text{ml}(a, x, \hat{a})$	$x + \mathrm{E}_{\text{ml}} + \mathrm{E}_{\text{ul}} + \mathrm{E}_{\text{termau}}(a, \hat{a}) \boldsymbol{+ \mathbf{E}_{\textbf{ml\_mm}}(a, \hat{a})}$

energy algebras $\mathcal{I}_{bwe}$ are very similar to MFE algebras, but energy values are scaled via pf, additions are replaced by multiplications and the choice function is summation instead of minimization. Due to technical reasons, values are down scaled by $sc(y) = \left(e^{\frac{0.1843}{R \cdot T}}\right)^{-y}$, according to sub-word length y to avoid numerical overflows.

Probability of a shape class is its individual partition function Q_p, divided by the overall partition function Q: $Prob(p) = Q_p/Q$. Both partition functions can be computed via two Bellman's GAP instances. First instance returns Q_p for all possible p, here for the NoDangle model:

$$\mathcal{G}_{\text{NoDangle}}(\mathcal{I}_{\pi_i} * \mathcal{I}_{bwe}, x).$$

Second instance directly gives Q:

$$\mathcal{G}_{\text{NoDangle}}(\mathcal{I}_{bwe}, x).$$

Alternatively, Q could be deduced from a run of the first instance, simply as $\sum_p Q_p$. Algebra $\mathcal{I}_{bwe}$ is given in Table 3.6.

For MicroState results, we just have to replace the grammar: $\mathcal{G}_{\text{MicroState}}(\mathcal{I}_{\pi_i} * \mathcal{I}_{bwe}, x)$. Transition to the OverDangle model is achieved by using a special bwe algebra $\mathcal{I}_{bwe}^{OverDangle}$ (Table 3.7), which only differs in the functions drem and ml.
The according Bellman's GAP instance is $\mathcal{G}_{\text{OverDangle}}(\mathcal{I}_{\pi_i} * \mathcal{I}_{bwe}^{OverDangle}, x)$.

The correct MacroState algebra $\mathcal{I}_{bwe}^{MacroState}$ has a very complicated internal structure. Thus, we forbear to present it here in full details and point to the source code. However, we give one example to sketch the intuition of the algebra:

Let us first reconsider the "meaning" of the answer type x in a simple function like $\text{sr}(a, x, \hat{a})$. It represents the Boltzmann weighted energies of all possible sub-candidates that can be parsed by the sub-sequence for x. By grammar construction, we know that whatever x might be, outermost bases form a base-pair. Thus, we can conceptually extend all these candidates by one base-pair, practically by one multiplication of x with the according Boltzmann weighted energy $pf\,(\mathrm{E}_{\text{sr}}(a, \hat{a}))$.

Function $\text{acomb}(x, b, y)$ connects one closed component x to a sequence of one or more closed components y with an unpaired base b between them in the context of a multiloop. Conceptually, the list of candidates for the complete word xby is the Cartesian product of x and y with optimal dangling of b for each combination. The problem is that we have no access to these lists of candidates, since x and y are just two numbers. In the

Table 3.5: MFE evaluation algebra $\mathcal{I}_{mfe}^{MacroState}$ for the MacroState model. Here, the answer data-type is a five int-tuple. First component is the free energy. Second and third component store left and right border of the leftmost stem in the candidate. Left border of a terminal parser a is addressed as ${}_ia$, right border as a_j. The last two components store borders for the rightmost stem.

algebra function	$\mathcal{I}_{mfe}^{MacroState}$
sadd(a, x)	$(x_1 + \mathrm{E}_{\mathrm{sbase}}, x_2, x_3, x_4, x_5)$
cadd(x, y)	$(x_1 + y_1, x_2, x_3, y_4, y_5)$
nil(l)	$(0, 0, 0, 0, 0)$
drem(l_1, x, l_2)	$(x_1 + \mathrm{E}_{\mathrm{termau}}(l_1, l_2), x_2, x_3, x_4, x_5)$
edl(a, x, l)	$(x_1 + \mathrm{E}_{\mathrm{termau}}(a_{+1}, l) + \mathrm{E}_{\mathrm{dl}}(a_{+1}, l), x_2, x_3, x_4, x_5)$
edr(l, x, b)	$(x_1 + \mathrm{E}_{\mathrm{termau}}(l, b_{-1}) + \mathrm{E}_{\mathrm{dr}}(l, b_{-1}), x_2, x_3, x_4, x_5)$
edlr(a, x, b)	$(x_1 + \mathrm{E}_{\mathrm{termau}}(a_{+1}, b_{-1}) + \mathrm{E}_{\mathrm{ext\_mm}}(a_{+1}, b_{-1}), x_2, x_3, x_4, x_5)$
sr$(a, x, \hat{a})$	$(x_1 + \mathrm{E}_{\mathrm{sr}}(a, \hat{a}), x_2, x_3, x_4, x_5)$
hl$(a, r, \hat{a})$	$(\mathrm{E}_{\mathrm{hl}}(r), {}_ia, \hat{a}_j, {}_ia, \hat{a}_j)$
bl$(a, r, x, \hat{a})$	$(x_1 + \mathrm{E}_{\mathrm{bl}}(r, \hat{a}), a_i, \hat{a}_j, a_i, \hat{a}_j)$
br$(a, x, r, \hat{a})$	$(x_1 + \mathrm{E}_{\mathrm{br}}(a, r), {}_ia, \hat{a}_j, {}_ia, \hat{a}_j)$
il$(a, r_1, x, r_2, \hat{a})$	$(x_1 + \mathrm{E}_{\mathrm{il}}(r_1, r_2), {}_ia, \hat{a}_j, {}_ia, \hat{a}_j)$
ml$(a, x, \hat{a})$	$(x_1 + \mathrm{E}_{\mathrm{ml}} + \mathrm{E}_{\mathrm{ul}} + \mathrm{E}_{\mathrm{termau}}(a, \hat{a}), {}_ia, \hat{a}_j, {}_ia, \hat{a}_j)$
mldl$(a, b, x, \hat{a})$	$(x_1 + \mathrm{E}_{\mathrm{ml}} + \mathrm{E}_{\mathrm{ul}} + \mathrm{E}_{\mathrm{termau}}(a, \hat{a}) + \mathrm{E}_{\mathrm{dli}}(a, \hat{a}), {}_ia, \hat{a}_j, {}_ia, \hat{a}_j)$
mldr$(a, x, b, \hat{a})$	$(x_1 + \mathrm{E}_{\mathrm{ml}} + \mathrm{E}_{\mathrm{ul}} + \mathrm{E}_{\mathrm{termau}}(a, \hat{a}) + \mathrm{E}_{\mathrm{dri}}(a, \hat{a}), {}_ia, \hat{a}_j, {}_ia, \hat{a}_j)$
mldlr$(a, b, x, c, \hat{a})$	$(x_1 + \mathrm{E}_{\mathrm{ml}} + \mathrm{E}_{\mathrm{ul}} + \mathrm{E}_{\mathrm{termau}}(a, \hat{a}) + \mathrm{E}_{\mathrm{ml\_mm}}(a, \hat{a}), {}_ia, \hat{a}_j, {}_ia, \hat{a}_j)$
addss(x, r)	$(x_1 + \mathrm{E}_{\mathrm{ss}}(r), x_2, x_3, x_4, x_5)$
incl(x)	$(x_1 + \mathrm{E}_{\mathrm{ul}}, x_2, x_3, x_4, x_5)$
cadd’(x, y)	$(x_1 + y_1, x_2, x_3, y_4, y_5)$
cadd”(x, y)	$(x_1 + y_1, x_2, x_3, y_4, y_5)$
cadd”’(x, y)	$(x_1 + y_1, x_2, x_3, y_4, y_5)$
ambd(x, b, y)	$(x_1 + y_1 + \mathbf{min}(\mathrm{E}_{\mathrm{dr}}(x_4, x_5), \mathrm{E}_{\mathrm{dl}}(y_2, y_3)), x_2, x_3, y_4, y_5)$
ambd’(x, b, y)	$(x_1 + y_1 + \mathbf{min}(\mathrm{E}_{\mathrm{dr}}(x_4, x_5), \mathrm{E}_{\mathrm{dl}}(y_2, y_3)), x_2, x_3, y_4, y_5)$
mladl$(a, b, x, \hat{a})$	$(x_1 + \mathrm{E}_{\mathrm{ml}} + \mathrm{E}_{\mathrm{ul}} + \mathrm{E}_{\mathrm{termau}}(a, \hat{a}) + \mathbf{min}(\mathrm{E}_{\mathrm{dli}}(a, \hat{a}), \mathrm{E}_{\mathrm{dl}}(x_2, x_3)), {}_ia, \hat{a}_j, {}_ia, \hat{a}_j)$
mladr$(a, x, c, \hat{a})$	$(x_1 + \mathrm{E}_{\mathrm{ml}} + \mathrm{E}_{\mathrm{ul}} + \mathrm{E}_{\mathrm{termau}}(a, \hat{a}) + \mathbf{min}(\mathrm{E}_{\mathrm{dri}}(a, \hat{a}), \mathrm{E}_{\mathrm{dr}}(x_4, x_5)), {}_ia, \hat{a}_j, {}_ia, \hat{a}_j)$
mladlr$(a, b, x, c, \hat{a})$	$(x_1 + \mathrm{E}_{\mathrm{ml}} + \mathrm{E}_{\mathrm{ul}} + \mathrm{E}_{\mathrm{termau}}(a, \hat{a})$ $+\mathbf{min}(\mathrm{E}_{\mathrm{dli}}(a, \hat{a}), \mathrm{E}_{\mathrm{dl}}(x_2, x_3)) + \mathbf{min}(\mathrm{E}_{\mathrm{dri}}(a, \hat{a}), \mathrm{E}_{\mathrm{dr}}(x_4, x_5)), {}_ia, \hat{a}_j, {}_ia, \hat{a}_j)$
mldladr$(a, b, x, c, \hat{a})$	$(x_1 + \mathrm{E}_{\mathrm{ml}} + \mathrm{E}_{\mathrm{ul}} + \mathrm{E}_{\mathrm{termau}}(a, \hat{a})$ $+\mathrm{E}_{\mathrm{dli}}(a, \hat{a}) + \mathbf{min}(\mathrm{E}_{\mathrm{dri}}(a, \hat{a}), \mathrm{E}_{\mathrm{dr}}(x_4, x_5)), {}_ia, \hat{a}_j, {}_ia, \hat{a}_j)$
mladldr$(a, b, x, c, \hat{a})$	$(x_1 + \mathrm{E}_{\mathrm{ml}} + \mathrm{E}_{\mathrm{ul}} + \mathrm{E}_{\mathrm{termau}}(a, \hat{a})$ $+\mathrm{E}_{\mathrm{dri}}(a, \hat{a}) + \mathbf{min}(\mathrm{E}_{\mathrm{dli}}(a, \hat{a}), \mathrm{E}_{\mathrm{dl}}(x_2, x_3)), {}_ia, \hat{a}_j, {}_ia, \hat{a}_j)$
ssadd(r, x)	$(x_1 + \mathrm{E}_{\mathrm{ss}}(r), x_2, x_3, x_4, x_5)$
trafo(x)	x
combine(x, y)	$(x_1 + y_1, x_2, x_3, y_4, y_5)$
acomb(x, b, y)	$(x_1 + y_1 + \mathbf{min}(\mathrm{E}_{\mathrm{dr}}(x_4, x_5), \mathrm{E}_{\mathrm{dl}}(y_2, y_3)), x_2, x_3, y_4, y_5)$
$\mathcal{S}$	(int,int,int,int,int)
choice function	min.

Table 3.6: Boltzmann weighted energy evaluation algebra $\mathcal{I}_{bwe}$. Unused algebra functions in models NoDangle and MicroState are omitted for the sake of readability.

algebra function	$\mathcal{I}_{bwe}$
sadd(a, x)	$x \cdot pf\ (\mathrm{E}_{\mathrm{sbase}}) \cdot sc(1)$
cadd(x, y)	$x \cdot y$
nil(l)	1.0
drem(l_1, x, l_2)	$x \cdot pf\ (\mathrm{E}_{\mathrm{termau}}(l_1, l_2))$
edl(a, x, l)	$x \cdot pf\ (\mathrm{E}_{\mathrm{termau}}(a_{+1}, l) + \mathrm{E}_{\mathrm{dl}}(a_{+1}, l)) \cdot sc(1)$
edr(l, x, b)	$x \cdot pf\ (\mathrm{E}_{\mathrm{termau}}(l, b_{-1}) + \mathrm{E}_{\mathrm{dr}}(l, b_{-1})) \cdot sc(1)$
edlr(a, x, b)	$x \cdot pf\ (\mathrm{E}_{\mathrm{termau}}(a_{+1}, b_{-1}) + \mathrm{E}_{\mathrm{ext\_mm}}(a_{+1}, b_{-1})) \cdot sc(2)$
sr$(a, x, \hat{a})$	$x \cdot pf\ (\mathrm{E}_{\mathrm{sr}}(a, \hat{a})) \cdot sc(2)$
hl$(a, r, \hat{a})$	$pf\ (\mathrm{E}_{\mathrm{hl}}(r)) \cdot sc(2 + \lvert r \rvert)$
bl$(a, r, x, \hat{a})$	$x \cdot pf\ (\mathrm{E}_{\mathrm{bl}}(r, \hat{a})) \cdot sc(2 + \lvert r \rvert)$
br$(a, x, r, \hat{a})$	$x \cdot pf\ (\mathrm{E}_{\mathrm{br}}(a, r)) \cdot sc(2 + \lvert r \rvert)$
il$(a, r_1, x, r_2, \hat{a})$	$x \cdot pf\ (\mathrm{E}_{\mathrm{il}}(r_1, r_2)) \cdot sc(2 + \lvert r_1 \rvert + \lvert r_2 \rvert)$
ml$(a, x, \hat{a})$	$x \cdot pf\ (\mathrm{E}_{\mathrm{ml}} + \mathrm{E}_{\mathrm{ul}} + \mathrm{E}_{\mathrm{termau}}(a, \hat{a})) \cdot sc(2)$
mldl$(a, b, x, \hat{a})$	$x \cdot pf\ (\mathrm{E}_{\mathrm{ml}} + \mathrm{E}_{\mathrm{ul}} + \mathrm{E}_{\mathrm{termau}}(a, \hat{a}) + \mathrm{E}_{\mathrm{dli}}(a, \hat{a})) \cdot sc(3)$
mldr$(a, x, b, \hat{a})$	$x \cdot pf\ (\mathrm{E}_{\mathrm{ml}} + \mathrm{E}_{\mathrm{ul}} + \mathrm{E}_{\mathrm{termau}}(a, \hat{a}) + \mathrm{E}_{\mathrm{dri}}(a, \hat{a})) \cdot sc(3)$
mldlr$(a, b, x, c, \hat{a})$	$x \cdot pf\ (\mathrm{E}_{\mathrm{ml}} + \mathrm{E}_{\mathrm{ul}} + \mathrm{E}_{\mathrm{termau}}(a, \hat{a}) + \mathrm{E}_{\mathrm{ml\_mm}}(a, \hat{a})) \cdot sc(4)$
addss(x, r)	$x \cdot pf\ (\mathrm{E}_{\mathrm{ss}}(r)) \cdot sc(\lvert r \rvert)$
incl(x)	$x \cdot pf\ (\mathrm{E}_{\mathrm{ul}})$
$\mathcal{S}$	float
choice function	Σ

Table 3.7: Boltzmann weighted energy evaluation algebra $\mathcal{I}_{bwe}^{OverDangle}$ for the OverDangle model.

algebra function	$\mathcal{I}_{bwe}^{OverDangle} -\!\!\rhd \mathcal{I}_{bwe}$
drem(l_1, x, l_2)	$x \cdot pf\ (\mathrm{E}_{\mathrm{termau}}(l_1, l_2) + \boldsymbol{\mathrm{E}_{\mathrm{ext\_mm}}(l_1, l_2)})$
ml$(a, x, \hat{a})$	$x \cdot pf\ (\mathrm{E}_{\mathrm{ml}} + \mathrm{E}_{\mathrm{ul}} + \mathrm{E}_{\mathrm{termau}}(a, \hat{a})\ + \boldsymbol{\mathrm{E}_{\mathrm{ml\_mm}}(a, \hat{a})}) \cdot sc(2)$

MFE example, we saw that representing y with just one candidate led to wrong results. How many classes of candidates do we have to store, to handle this situation correctly? In fact, it is just two for each sub-component! Why?

Grammar design for y tells us, that b can only dangle onto a base-pair, whose left partner is directly adjacent to b, but there can be different right partners. Not only the position of b is fixed, but of cause also the according nucleotide; say it is an `A`. Wherever the right partner might be located, it must be an `U`. For the Boltzmann weighted energy contribution the partnering *position* is of no interest, what counts is the nucleotide. Thus, multiplying the position independent amount $pf\,(\mathrm{E}_{\mathrm{dl'}}(b, (\mathtt{A}, \mathtt{U})))$ with the result will implicitly handle all candidates correctly.

Due to the *wobbling* base-pairs, the right partner is not unambiguously defined, e. g. partner for an `G` can either be `C` or `U`. Since some bases have two alternatives, we need to split the partition function value for y into two parts. The same thoughts apply to x. In the end, we are able to treat all ambiguous situations of $\mathcal{G}_{\text{MacroState}}$ correctly by locally splitting the partition function values in at most 4 parts. Thus, the answer type is an eight tuple, with four components for the partition function value and four components for tracing first and last stem borders, as in $\mathcal{I}_{mfe}^{MacroState}$. Computing shape probabilities for MacroState is realized with the instance

$$\mathcal{G}_{\text{MacroState}}(\mathcal{I}_{\pi_i} * \mathcal{I}_{bwe}^{MacroState}, x).$$

3.3 Results & Discussion

3.3.1 Data set

Compiling a list of experimentally verified RNA secondary structures is a very tedious and error prone task. It is unclear to what extend e. g. the crystallization for X-ray diffraction – or other techniques – changes the native structure of an RNA molecule. Furthermore, only one of all possible structures of the ensemble is captured. Is it the most likely or thermodynamically most stable one?

We used a very conservative set of reference structures in our original publication. The side effect was a relatively short mean sequence length of only ≈ 46bp. To see if our observations pertain for longer sequences too, we here switch to the widely used set of Andronescu et al. “S-full”[4]. The set “S-full”contains 3,245 sequence / structure pairs. The mean sequence length is ≈ 270bp. A length histogram is given in Figure 3.6. Andronescu et al. applied processing steps to obtain structures that can be predicted by the features of the Turner model and to reduce the uncertainty in the data. These steps include shortening the structures to be at most 700 nucleotides in length and the removal of the minimum number of base-pairs that close pseudoknots, of non-canonical base-pairs, of structures with unknown nucleotides, and of overly large loops. We consider the “S-full”set of structures as the set of “reference” structures. They constitute our standard of truth, but we are reluctant to call them “true” structures, since structures

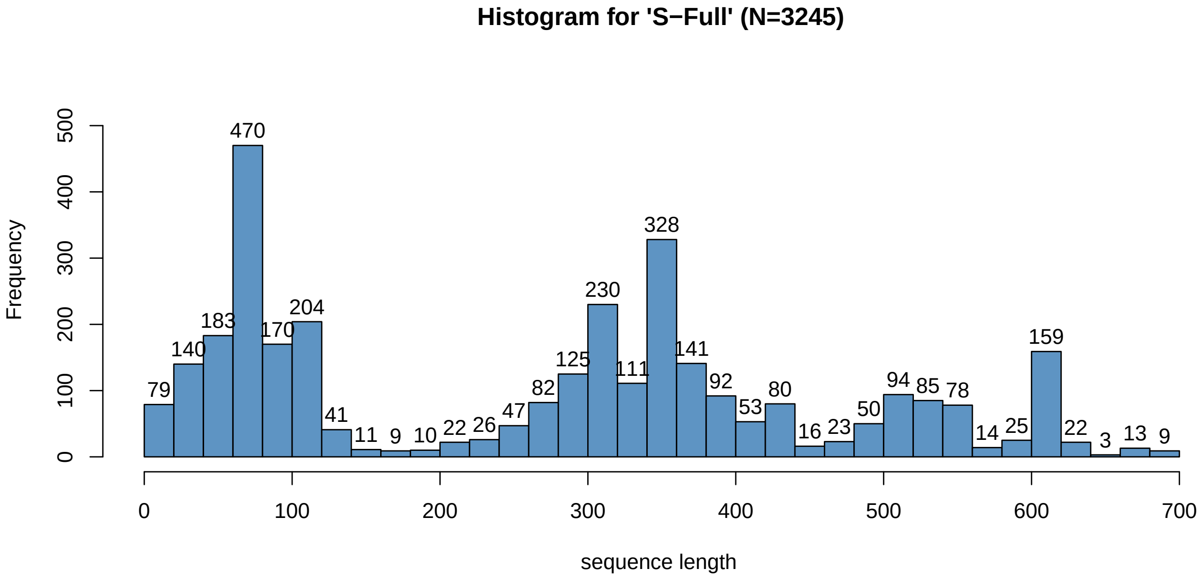


Figure 3.6: Sequence length histogram for the data set “S-full”. It contains 3,245 RNA sequences with known secondary structures from the RNA STRAND v2.0 data-base [2].

in cristallo may be different from structures *in vivo* [10].

3.3.2 Technical Environment

Measurements for the evaluation have been performed on a 48 core machine, powered by a *Red Hat 4.7.2-8* operating system. It is equipped with 4 *AMD Opteron(tm) 6168* processors, running at 1.9 *GHz* and 128 *GB* main memory. Jobs were scheduled via the *Oracle Grid Engine* (OGE), version *6.2u7*. Maximal memory consumption for each job has been restricted to 4 *GB* via the `-l virtual_free` and `-l h_vmem` OGE parameters.

We used the tool *GNU time 1.7* to log run-time and memory consumption. The latter in the form of the maximum resident set size (RSS) of the process during its lifetime.

3.3.3 Evaluation of models for MFE structure prediction

While our main interest is in the effect of the chosen model on the partition function based computations, we here evaluate the four grammars with respect to prediction of a single MFE structure.

Evaluation setup

In evaluating models with respect to MFE structure prediction, we include not only our programs NoDangle and OverDangle, MicroState and MacroState, but also the folding

[10] This can be evaluated by experimental techniques [26], but sufficient data are not yet available.

programs UNAFOLD and RNAFOLD, which our readers are rightfully curious about because of their practical importance. *Turner'2004* parameters [60] were used throughout[11]. These parameters are derived from melting experiments, with a few exceptions. Multiloop parameters such as E_{ml} in *Turner'2004* are not derived from experiment, but are optimized from structure data to be used in conjunction with the MicroState model. Out of competition, we also include CENTROIDFOLD, which goes beyond strict energy minimization by producing a near-optimal ensemble of structures and choosing the eventual, single-structure prediction based on this sample.

Relative performance of programs of different origin is, however, not our main interest here. Mainly, the evaluation should support that our four grammars faithfully reproduce the behavior of the models underlying RNAFOLD with options -d0, -d1, and -d2, as postulated at the outset of this study.

Evaluation results are summarized in Figure 3.7. We use an asymmetric base-pair distance for comparison, where one structure (row entry) is treated as the prediction, the other as the reference (column entry): One base-pair set, i. e. secondary structure, is the reference (R: table columns), the other one is the prediction (P: table rows). Traditional base-pair distance is defined as $|R \setminus P| + |P \setminus R|$. Following [25], we decide to allow additional base-pairs in the prediction, as long as they are compatible with the reference, i. e. both bases are unpaired and the additional base-pair does not introduce a pseudoknot in the reference. The set of compatible base-pairs is $P^{-c} = P \setminus \{(a,b)|(a,b) \notin R \wedge (a,b) \text{ compatible to } R\}$. Then, our asymmetric base-pair distance is: $|R \setminus P| + |P^{-c} \setminus R|$. Table values are the sums of base-pair distances for all 3,245 sequences. In the case of co-optimal results, the one with the smallest distance to the reference is chosen.

Our distance function is rather strict and does not allow base-pair slippage. If a reference base-pair (i,j) is mispredicted as $(i+1,j)$, this contributes a distance of 2.

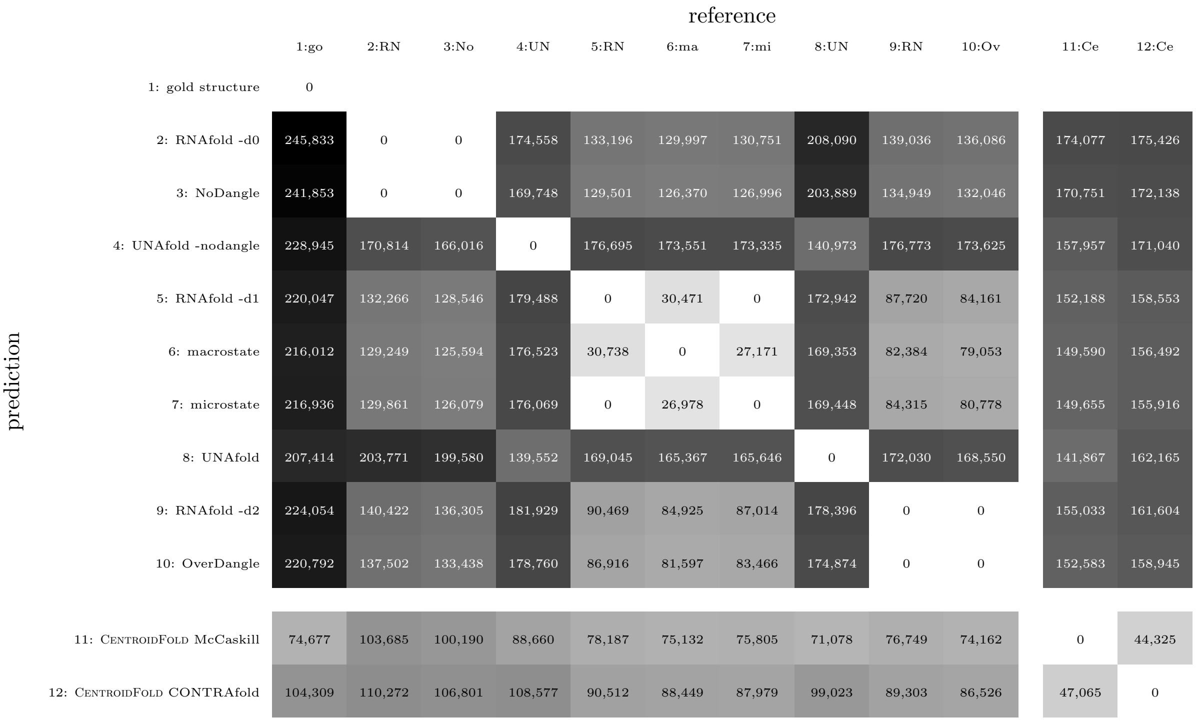
For each RNA sequence of our test-set we called the programs with the following command line options:

- **RNAfold** (version 2.1.3): `echo sequence | RNAfold --noPS --noLP -dX`, where `X` is 0, 1 or 2.

- **UNAfold** (version 3.8): `hybrid-ss-min --suffix=DAT --mfold --NA=RNA --tmin=37 --tinc=1 --tmax=37 --sodium=1 --magnesium=0 --noisolate --nodangle tmpseqfile > /dev/null && ct2b.pl tmpseqfile.ct`, with and without the `--nodangle` , where "`tmpseqfile`" is a fasta file containing the sequence and "`ct2b.pl`" is a small PERL script from the Vienna-Package, which converts RNA structures from "connect" to "dot-bracket" format.

- **CentroidFold** (version v0.0.9): `centroid_fold --engine=X tmpseqfile`, where `X` is the source of base-pair probabilities and is either computed by RNAFOLD (McCaskill) or by CONTRAFOLD.

[11] Except for CENTROIDFOLD, since its most current release is not aware of these new parameters.


reference
prediction

	1:go	2:RN	3:No	4:UN	5:RN	6:ma	7:mi	8:UN	9:RN	10:Ov	11:Ce	12:Ce
1: gold structure	0											
2: RNAfold -d0	245,833	0	0	174,558	133,196	129,997	130,751	208,090	139,036	136,086	174,077	175,426
3: NoDangle	241,853	0	0	169,748	129,501	126,370	126,996	203,889	134,949	132,046	170,751	172,138
4: UNAfold -nodangle	228,945	170,814	166,016	0	176,695	173,551	173,335	140,973	176,773	173,625	157,957	171,040
5: RNAfold -d1	220,047	132,266	128,546	179,488	0	30,471	0	172,942	87,720	84,161	152,188	158,553
6: macrostate	216,012	129,249	125,594	176,523	30,738	0	27,171	169,353	82,384	79,053	149,590	156,492
7: microstate	216,936	129,861	126,079	176,069	0	26,978	0	169,448	84,315	80,778	149,655	155,916
8: UNAfold	207,414	203,771	199,580	139,552	169,045	165,367	165,646	0	172,030	168,550	141,867	162,165
9: RNAfold -d2	224,054	140,422	136,305	181,929	90,469	84,925	87,014	178,396	0	0	155,033	161,604
10: OverDangle	220,792	137,502	133,438	178,760	86,916	81,597	83,466	174,874	0	0	152,583	158,945
11: CentroidFold McCaskill	74,677	103,685	100,190	88,660	78,187	75,132	75,805	71,078	76,749	74,162	0	44,325
12: CentroidFold CONTRAfold	104,309	110,272	106,801	108,577	90,512	88,449	87,979	99,023	89,303	86,526	47,065	0

Figure 3.7: Base-pair distances of different MFE prediction programs with “S-full”. Together, all “reference” structures contain 235,654 base-pairs.

- Our ADP implementation of the four grammars “NoDangle”, “OverDangle”, “MicroState” and “MacroState” get the sequence as their sole input. The binaries can be built with the source code provided by the fold-grammars repository `http://bibiserv.cebitec.uni-bielefeld.de/fold-grammars/` and the Bellman’s GAP compiler.

Observations from MFE prediction experiment

Consistency of implementations Naturally, comparing the results from the same tool leads to entries of zero base-pair distance in the diagonal of Figure 3.7. The off-diagonal zero entries, however, are quite remarkable. When two different algorithms perfectly agree in their MFE predictions on the complete data set, this provides strong evidence that they both faithfully implement the same thermodynamic model of the folding space in each of its variants. In particular, this shows that our grammars $\mathcal{G}_{\text{NoDangle}}/\mathcal{G}_{\text{OverDangle}}$ and $\mathcal{G}_{\text{MicroState}}$ indeed capture the analysis computed by RNAFold with options -d0/d2 and -d1. The perfect zeroes might even make our reader suspicious! Occasionally, there must be two (or more) co-optimal structures of minimal free energy, and it is not formally defined which one a program should return in this situation. Hence, it is accidental whether or not two different programs, implemented by different programmers, make the same choice. We therefore have designed our new programs to report all co-optimal solutions in such a situation, and then choose the structure closest to the RNAfold prediction. This always delivered a perfect match. Additionally, we had to obey the Vienna-Package rule, that the combined size of both unpaired regions of an internal loop (il) cannot exceed 30 base-pairs, which is a minor difference from the presented grammars in Figures 3.2, 3.3 and 3.2.6. Otherwise, 22 of the 3,245 sequences would not perfectly agree. Differences between MacroState and MicroState/RNAfold -d1 illustrate cases where MacroState's violation against Bellman's Principle of Optimality leads to wrong results.

We apply the same technique of safe-guarding against co-optimals when comparing to a database structure. Note that in practice, when predicting structure for a novel RNA, the users of a structure prediction program have no reference structure to resort to. In this case, reporting all co-optimal structures makes them aware of the ambiguity of the situation, and leaves them with the choice to make. This is somewhat preferable to quietly reporting a single MFE structure, selected from several by implementation peculiarities.

The agreement of MacroState with the MFE prediction of RNAfold -d1 as well as the perfect agreement with MicroState demonstrates that MacroState in fact computes the energy model of the other two programs, while avoiding (as explained above) their explosion of the state space. Taken together, these consistency results shows that we have correct programs set up for our second experiment, where we will evaluate the effect of the chosen energy and state space model on partition function calculations.

Results for UNAfold are not in perfect agreement with MicroState as well as UNAfold `-nodangle` does not totally agree with NoDangle. Nevertheless, these pairs fit best. Discrepancies can be explained by the fact that UNAfold basically implements the same folding space as the other programs, but also models some additional aspects. Unfortunately, these extra aspects violates Bellman's Principle of Optimality (cf. Section 2.1.1).

UNAfold violates Bellman's principle of optimality UNAfold's manual [12] says

> Finally, there are some special hairpin loop rules derived from experiments that will be defined explicitly here. A hairpin loop closed by r_i and r_j $(i < j)$ called a "GGG" loop if $r_{i-2} = r_{i-1} = r_i =$ G and $r_j =$ U . Such a loop receives a free energy bonus that is stored in the `miscloop.dg` or `miscloop.TC` file, which contains a variety of miscellaneous, or extra free energy parameters.

Assume UNAfold would use $\mathcal{G}_{\text{NoDangle}}$ to span the search space, use $\mathcal{I}_{mfe}$ to evaluate candidates and $\mathcal{I}_{db}$ to represent them. Actually, UNAfolds recurrences are not defined by a grammar and uses only three "non-terminals", but the following considerations still hold. Two of the candidates of $F_{\mathcal{G}_{\text{NoDangle}}}$(GGGAAAUUU) are given in Figure 3.8.

Furthermore, assume that the special energy bonus is -2.2 kcal/mol as defined in file `miscloop.DAT` of the UNAfold package [13]. Respecting this bonus, energy of candi-

[12] `http://mfold.rna.albany.edu/download/mfold-3.0-manual.pdf.gz`, seen on 26.06.2013
[13] shipped with program version 3.8

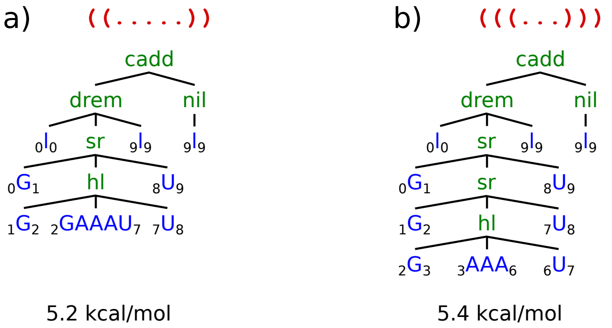


Figure 3.8: Two of several candidates from the search space $F_{\mathcal{G}_{\text{NoDangle}}}(\texttt{GGGAAAUUU})$, to illustrate situations where UNAFOLD violates Bellman's principle of optimality, regarding an energy evaluation algebra.

date t_b would promote to 3.2 kcal/mol and clearly overtake former energetically better candidate t_a. This promotion can only be made at $\text{sr}({}_0\text{G}_1, x, {}_8\text{U}_9)$, where x holds the energy for the *best* sub-structure for sub-word $(1, 8)$, which is that of t_a. Thus, we would compute the false result of 3.0 kcal/mol, which is even better than the optimal value of 3.2 kcal/mol!

Here are some thoughts about possible solutions:

1. Similar to the energy functions, we might be able to "look" into the sub-word $(1, 8)$ at sr to recognise the GGG ↔ UYY pattern, but we would not know, if it is really modelled as the right secondary structure, because sr can precede all kinds of closed motives. We even cannot get this right if we do not apply choice function *min* to x beforehand. Although x would turn from a single energy value into a list of energy values, we would still miss the associated structural information. Besides, we would loose a significant amount of run-time speed-up from dynamic programming.

2. Another dangerous route to integrate this bonus could be to add a special production to $\mathcal{G}_{\text{NoDangle}}$, like $\textit{hairpin} = \text{hl}_{\text{GGG}}(\text{sr}(\texttt{G}, \text{sr}(\texttt{G}, \text{hl}(\texttt{G}, \texttt{r}, \texttt{U}), \texttt{Y}), \texttt{Y}))$. Due to this production, it is ensured that hl_{GGG} captures both, the right sequencial pattern and the right sub-structure. Thus, the energy bonus could specifically be applied. Predicting just one MFE structure will be fine. But identical secondary structures like (((...))) are modeled by two candidates (semantic ambiguity, cf. Section 2.1.3), destroying all stochastic properties and thus renders the grammar useless regarding shape-class or base-pair probabilities and other purposes.

3. In principle, it is possible to design a special grammar, that can precisely differentiate between *normal* hairpins with three base-pairs and the special one. But it would require enormous overhead and a significant slow-down in run-time is to be expected, although leaving the asymptotics untouched. Since there are no traces of this effort in the UNAFOLD code and the default parameter for the bonus is set to zero, this route does not seem to be promising, as well.

Quality of MFE predictions Overall, the quality of MFE predictions compared to "reference" structures is moderate when measured on the individual base-pair level, with errors [14] ranging from 44% to 52% for the reference structures. This is expected and well-known. It is the reason why researchers have developed more advanced techniques, such as structure sampling, complete enumeration, or shape abstraction. Experimentally resolved structures contain base-pairs which by definition are not predicted – non-standard pairs, 3D interactions, pseudoknots, and lonely pairs.

The reference structures are best predicted by UNAFOLD (distance 207,414), due to its additional rules. However, these rules – and thus UNAFOLD – cannot be considered in the prediction of shape probabilities, because of the the problems stated in Section 3.3.3.

Performance of different dangling models Comparing the full dangling model (MicroState, MacroState) to its upper and lower approximations NoDangle and OverDangle, we find that its proper implementation pays off. It reduces the accumulated distance by about 11% over NoDangle, and by 2% over OverDangle. Similar percentages apply for RNAFOLD option -d1 versus -d0 and -d2. This also shows that OverDangle approximates the correct model better than NoDangle and justifies its use as a substitute for the full model in partition function calculations with RNAFOLD and RNASUBOPT, where $\mathcal{G}_{MacroState}$ is not available.

Looking deeper into the near-optimal folding space We included CENTROIDFOLD [37] as a representative of methods which, in contrast to the above programs, look deeper into the Boltzmann ensemble of near-optimal structures. Our evaluation shows that the extra effort is well spent. CENTROIDFOLD comes closest to the gold structures, and with respect to the single structure predictors, it corresponds on average best with OverDangle.

3.3.4 Evaluating models for partition function and related computations

Evaluation Criteria

In this section, we apply probabilistic shape analysis to our data set. In fact, we have to fall back to heuristics, to compute with the long sequences of "S-full"(see "Technical feasibility"). We are interested in the difference of performance of the four models NoDangle, OverDangle, MicroState and MacroState. For simplicity, we call the abstract shape of the reference structure the "reference shape", and refer to the most likely predicted shape as the "dominant shape", although its actual dominance within the Boltzmann ensemble will not be strong if there is another shape with similar probability.

[14] It is not obvious how to convert our absolute distances into error rates. Remember that a mispredicted base-pair can contribute a distance of 2 (cf. Section 3.3.3). Assuming that predictions hold about the same number of base-pairs as the reference structures (235,654), the interval of possible distance scores is [0,471308], from which the above percentages are derived.

The shape string of the reference shape of sequence s is obtained by a call to `RNAshapes --mode=abstract --shapelevel=l "s"`, where `l` is one of the five shape abstraction levels.

We ask the following questions:

- What are the differences in the shape probabilities computed with each of the four models?
- How is the difference affected by the shape abstraction level considered?

Since we do observe significant differences in model behavior, we also ask which model comes closer to the truth:

- To what extend does the dominant shape agree with the reference shape?
- What is the median (or the 20%, 35% and 65% quantile) of the reference shape among the predicted shapes?

And we consider

- What are the run-time or memory trade-offs for computing with different models?

Evaluation method Shape probabilities do not make a structure prediction per se. They provide holistic information by assigning probabilities to all shapes in the folding space of a sequence x. It is our responsibility how we interpret these data. The hope is, of course, to find the biologically functional structure among the high-probability shapes, to find two high probability shapes for a riboswitch, to use lack of any shape with high probability as an indicator of absence of a well-defined structure, and so on. Such analysis goes beyond shape probabilities, and takes into account the concrete shreps returned for each shape.

Independent of what the shape probabilities will be used for, we want to focus on the agreement between the four grammars. To measure this, we use the *shape probability shift* (SPS), first introduced and later published by Andreas Bremges [36]. For a given sequence x, all grammars will report the same shape classes, but with different probabilities. Let $P(x)$ be the *shape space*, i. e. the set of all shape classes for x, and $Prob_{\mathcal{G}}(p)$ the shape probability of p under grammar $\mathcal{G}$. The shape probability shift for x and grammars A and B is defined as:

$$SPS_{A,B}(x) = \frac{1}{2} \cdot \sum_{p \in P(x)} \left| Prob_A(p) - Prob_B(p) \right| \tag{3.1}$$

Note that $0 \leq SPS(x) \leq 1$, where the extreme case of 1 would only be achieved when all shapes with positive probability by grammar A have zero probability by grammar B and vice versa. The SPS can be interpreted as the overall probability mass that moves between shapes.

We chose the SPS measure because of this nice interpretation. We also evaluated two alternative measures. The squared distance of base-pair probability matrices is

Table 3.8: Relative memory. Averaged RSS for sequences 1 to 70 of *Random set 1* relative to the reference NoDangle in level 1, which equals 40 MB of RSS.

memory	5	4	3	2	1	avg.
NoDangle	1.0	1.0	1.0	1.2	2.6	1.4
OverDangle	1.0	1.0	1.0	1.2	2.6	1.4
MicroState	1.0	1.0	1.0	1.3	3.0	1.5
MacroState	1.1	1.1	1.2	1.9	5.5	2.2
avg.	1.0	1.1	1.1	1.4	3.4	

Table 3.9: Relative run-time. Averaged run-time for sequences 1 to 70 of *Random set 1* relative to the reference NoDangle in level 1, which equals 1.02 seconds.

run-time	5	4	3	2	1	avg.
NoDangle	1.0	1.3	1.5	4.7	22.2	6.2
OverDangle	0.9	1.2	1.5	4.7	22.4	6.2
MicroState	1.2	1.9	2.3	8.0	39.8	10.7
MacroState	1.8	2.7	3.3	9.9	36.7	10.9
avg.	1.2	1.8	2.2	6.8	30.3	

correlated with the SPS by a factor around 0.83 at shape level 5 and not much lower on less abstract shape levels. The Kullback-Leibler divergence turned out to be unsuitable for the purpose, as it is not symmetric and both versions (KL(x,y) versus KL(y,x)) show the poorest correlation among all methods tested. Details of this investigation of alternatives are given in additional file 1 of [45].

Relative run-time and memory consumption Running probabilistic shape analyses with random sequences for all models in all shape levels reveals that MacroState in level 1 is the most memory consumptive (see Table 3.8) one. *Random set 1* contains 1,000 random sequences, which have an equally distributed nucleotide composition and are ascendingly ordered by their sequence lengths. Each length is covered exactly once. We subsequently executed probabilistic shape analysis in all four models and for all five shape abstraction levels sequence by sequence. Relative run-times are given in Table 3.9. Memory has been initially exceeded for a random sequence with 71 nucleotides.

MacroState is the most sophisticated grammar and hence the most expensive to compute with in terms of memory as well as in run-time. This is clear, since all algorithms are implemented via dynamic programming, where a difference in the number of tables to be filled (with MacroState needing the most) directly maps to the difference in run-time as well as in space requirements.

Grammars NoDangle, OverDangle and MicroState all use the same number of DP tables (7), while only MacroState uses 20. Both, search space extension in MicroState and algebra enhancement of MacroState cause similar slowdowns compared to NoDangle/OverDangle on average. Automated table design (`multi_rt_all` of GAP-C) derives asymptotic run-times of $6n^3 + 4478n^2 + 8n + 6$ for NoDangle and OverDangle, $6n^3 + 4506n^2 + 8n + 6$ for MicroState and $36n^3 + 4571n^2 + 18n + 4$ for MacroState. Overall, the selected shape abstraction level makes more difference with resource requirements than the chosen model. For example, NoDangle (the most efficient) used with abstraction level 2 uses more time and space than MacroState (the least efficient) with abstraction levels 5, or 4.

Technical feasibility The number of shape classes grows exponentially with sequence length, as stated previously. A lower abstraction level cause a larger exponential factor. Nebel et al. estimates this factor to be ≈ 1.5 for shape level 1 [66]. As a practical consequence, shape probability computations often exceed available memory, even for short sequences $\ll 100$ *bp*, as observed in the last section.

There are two types of heuristics available for probabilistic shape analysis to handle sequences of common length, like in "S-full".

- **low probability filter (probs)**: during the search space exploration sub-shape classes less likely than a given threshold α will be excluded, thus reducing the exponential number of shape classes. A priori, it is impossible to know if an initially unlikely class cannot collect sufficient probability mass during computation to overshoot threshold α. That fact renders this approach into a heuristics. In the worst case, it might even miss the dominant shape class.

- **stochastical backtrace (sample)**: similar to SFOLD [13, 18], a given number of random structures r are drawn from a forward filled DP matrix. For a concrete RNA input, the summed Boltzmann weights for sub-structures at alternatives of a non-terminal in the grammar (i. e. other cells of the DP matrix) transform it into a *stochastic* context free grammar. The stochastical backtrace process randomly picks one alternative at each non-terminal, according to its Boltzmann weight. All sampled structures are abstracted to their shape classes and their frequency in the sample serves as an estimate for the shape class probability. The larger the number of sampled structures, the better the estimates. A (small) discrepancy between estimate and true probabilities will always remain. Thus, stochastical backtrace is a heuristics as well.

Both presented heuristics allow to trade speed-up for accuracy, either via the filter threshold α or the sample size r.

Figure 3.9 illustrates the consequences of different parameter values. All measurements are for MacroState in shape level 1, since it is the most resource demanding computation (cf. Tables 3.8 and 3.9). Exact shape probabilities could only be computed for the first and shortest 70 sequences of *Random set 1*. All heuristic runs have been called with these 70 sequences. Their results are compared to the exact probabilities in terms of

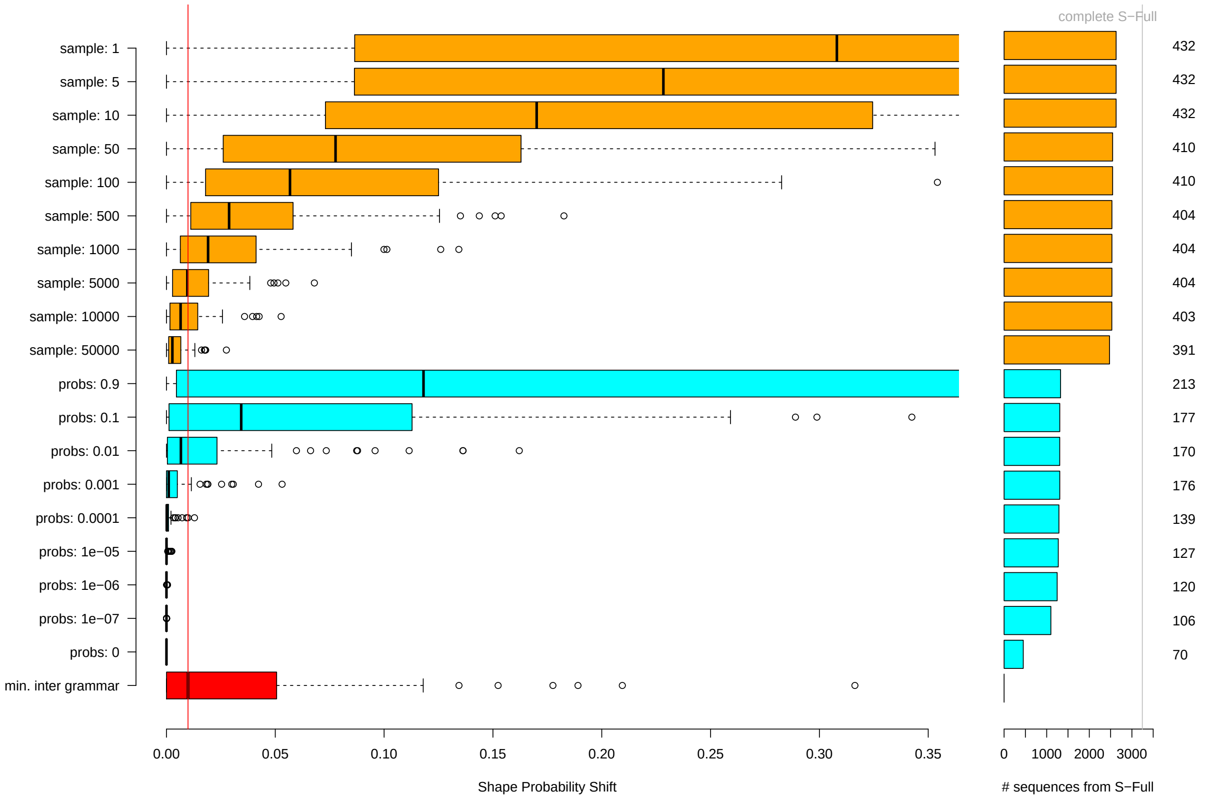


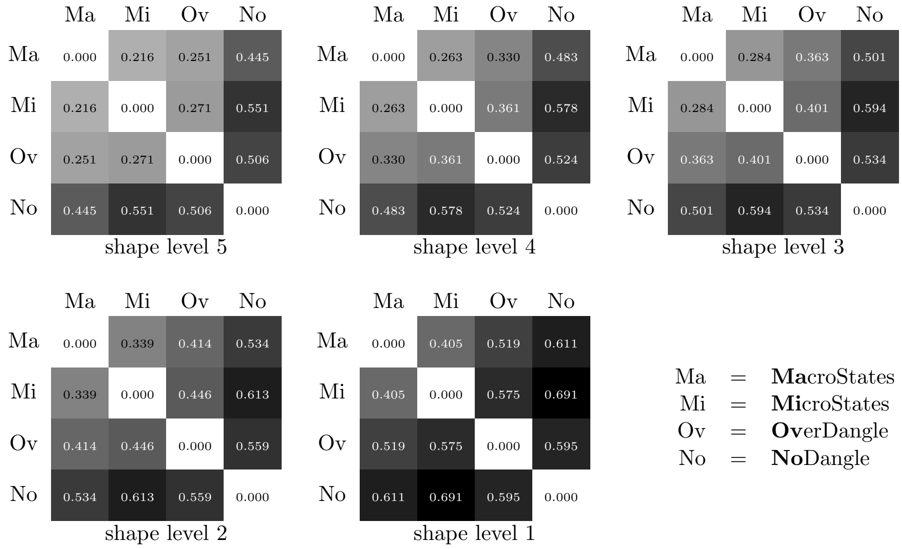
Figure 3.9: Influence of different parameter values for stochastical backtracing (orange) and low probability filter (blue) heuristics. The last row is to relate the error rate to the effects, i. e. model differences, that shall be analysed.

SPS to see the decline in accuracy. All SPS values are plotted as horizontal *boxplots*. Completely switching off the low probability filter (row "probs: 0" in Figure 3.9) by setting $\alpha = 0$ yields the exact values. Consequently, the SPS is zero for all sequences. As expected, with growing α or declining r the error rate, i. e. SPS, increases.

The numbers in the rightmost column of Figure 3.9 give the maximal sequence length for which the program did not exceed the given resource limitations when processing *Random set 1*. Only considering the sequence lengths, the mid part of Figure 3.9 indicates the ratio of computable sequences of "S-full". A clear anti-correlation between SPS and computable sequence length is observable.

In order to find a good trade-off between error rate and possible sequence length we must consider that we want to make statements about differences between the four grammars. Last row in Figure 3.9 "min. inter grammar" represents 70 SPS values for those two grammars that have smallest SPS deviations on average. The two closest grammars for the 70 sequences are MacroState and OverDangle. The vertical red line marks the median of those 70 SPS values.

Due to the results of Figure 3.9, we decide to use stochastical backtrace with a sample size of $r = 10,000$ for all subsequent shape class probability analyses.



	Ma	Mi	Ov	No
Ma	0.000	0.216	0.251	0.445
Mi	0.216	0.000	0.271	0.551
Ov	0.251	0.271	0.000	0.506
No	0.445	0.551	0.506	0.000

shape level 5

	Ma	Mi	Ov	No
Ma	0.000	0.263	0.330	0.483
Mi	0.263	0.000	0.361	0.578
Ov	0.330	0.361	0.000	0.524
No	0.483	0.578	0.524	0.000

shape level 4

	Ma	Mi	Ov	No
Ma	0.000	0.284	0.363	0.501
Mi	0.284	0.000	0.401	0.594
Ov	0.363	0.401	0.000	0.534
No	0.501	0.594	0.534	0.000

shape level 3

	Ma	Mi	Ov	No
Ma	0.000	0.339	0.414	0.534
Mi	0.339	0.000	0.446	0.613
Ov	0.414	0.446	0.000	0.559
No	0.534	0.613	0.559	0.000

shape level 2

	Ma	Mi	Ov	No
Ma	0.000	0.405	0.519	0.611
Mi	0.405	0.000	0.575	0.691
Ov	0.519	0.575	0.000	0.595
No	0.611	0.691	0.595	0.000

shape level 1

Ma	=	**Ma**croStates
Mi	=	**Mi**croStates
Ov	=	**Ov**erDangle
No	=	**No**Dangle

Figure 3.10: Model similarity: shape probability shift.

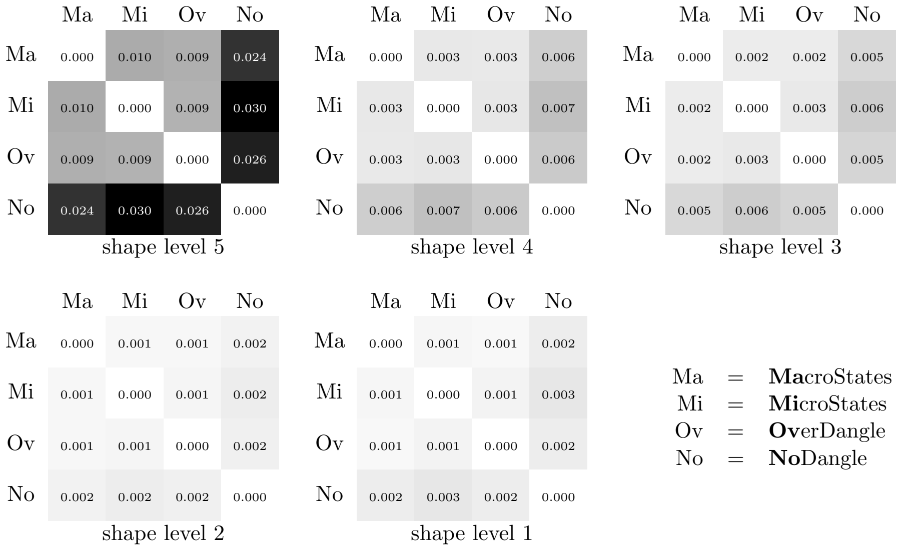
Observations The values in Figure 3.10 are average SPS over the 2,522 computable sequences of “S-full”.

First, consider shape abstraction level 5. We find that models MacroState and MicroState show the most agreement, where the SPS is around 21.6%. MacroState shows a significant SPS against the others, strongest against NoDangle (44.5%) but also against OverDangle (25.1%). A SPS in this range means that while in many cases, the predicted dominant shape will be the same for all models, this need not hold in general. This justifies the question which of the model finds the reference shape as the dominant shape more often (see below). By the way: the dominant shape and the shape of the MFE structure agree for MacroState in 2,027 out of 2,522 cases, for MicroState in 1,716, for OverDangle in 2,019 and for NoDangle in 2,110 cases.

Let us turn from level 5 to decreasing levels of abstraction. Moving to abstraction levels 4, 3, 2, and 1, the number of shapes increases with each step, while each shape class holds a smaller number of structures. The overall relationship between the models on levels 4 through 1 is consistent with what we observe for level 5. Overall, the SPS values increase. A closer inspection of the raw data shows that SPS values actually decrease for each individual shape, but due to the larger number of (smaller) shifts, their sum increases. Evidence is provided in Figure 3.11.

Dominant shape is reference shape? The values in Table 3.10 show the ratios of correct shape predictions vs. the size of the test-set, which is 2,522 for “S-full”. We observe the following:

The best ratio of agreement of dominant shape and reference shape is 40.0%. The fact that this value is not higher is the reason which makes investigators look into several high-probability shapes and their shreps in practice. Comparing the models, we find



	Ma	Mi	Ov	No
Ma	0.000	0.010	0.009	0.024
Mi	0.010	0.000	0.009	0.030
Ov	0.009	0.009	0.000	0.026
No	0.024	0.030	0.026	0.000

shape level 5

	Ma	Mi	Ov	No
Ma	0.000	0.003	0.003	0.006
Mi	0.003	0.000	0.003	0.007
Ov	0.003	0.003	0.000	0.006
No	0.006	0.007	0.006	0.000

shape level 4

	Ma	Mi	Ov	No
Ma	0.000	0.002	0.002	0.005
Mi	0.002	0.000	0.003	0.006
Ov	0.002	0.003	0.000	0.005
No	0.005	0.006	0.005	0.000

shape level 3

	Ma	Mi	Ov	No
Ma	0.000	0.001	0.001	0.002
Mi	0.001	0.000	0.001	0.002
Ov	0.001	0.001	0.000	0.002
No	0.002	0.002	0.002	0.000

shape level 2

	Ma	Mi	Ov	No
Ma	0.000	0.001	0.001	0.002
Mi	0.001	0.000	0.001	0.003
Ov	0.001	0.001	0.000	0.002
No	0.002	0.003	0.002	0.000

shape level 1

Ma	=	**Ma**croStates
Mi	=	**Mi**croStates
Ov	=	**Ov**erDangle
No	=	**No**Dangle

Figure 3.11: Model similarity: average shape probability shift per shape.

Table 3.10: Ratio of agreement between dominant shape and reference shape for the different grammars (columns) and different shape abstraction levels (rows).

Level	**MacroStates**	**MicroStates**	**OverDangle**	**NoDangle**
5	0.383	0.400	0.395	0.312
4	0.231	0.243	0.239	0.180
3	0.213	0.220	0.209	0.162
2	0.192	0.202	0.191	0.141
1	0.125	0.123	0.137	0.107

that there is no clear winner, with a margin of only 6.0% between the best and the worst performer. Here, MicroState finds agreement most often, with a 1.0% margin over MacroState and 5.7% margin over NoDangle. NoDangle performs worst (31.2%), but not hopeless when we consider that one will look at a number of top-ranking shapes anyway.

Thus, the more interesting question is how the reference shape is placed among the predicted shapes – cf. Table 3.11. We investigate this aspect by compiling a list of $rank(p^{\text{reference}})$ for all 2,522 test-sequences, sorting this list ascendingly and report the median (50%), the 20%, the 35% and the 65% quantile of the list. For example, the value 3 for MacroState in shape abstraction level 5 in the 50% column means that, if we decide to take only the top three shapes for closer study, the reference shape is among them in 50% of the cases. Four top shapes suffice to reach this coverage with NoDangle. Overall, the advantage of OverDangle appears marginal over the other grammars on level 5, and appears somewhat randomized for weaker abstraction levels.

Table 3.11: Positions of correct shapes. The $-$ indicates that the reference shape class was not among the $10,000$ sampled structures.

	MacroStates				**MicroStates**				**OverDangle**				**NoDangle**			
Level	20%	35%	50%	65%	20%	35%	50%	65%	20%	35%	50%	65%	20%	35%	50%	65%
5	1	1	3	14	1	1	3	19	1	1	3	12	1	2	4	20
4	1	4	145	-	1	4	125	-	1	4	64	-	2	7	-	-
3	1	5	-	-	1	4	-	-	1	5	400	-	2	8	-	-
2	2	17	-	-	1	15	-	-	2	16	-	-	4	43	-	-
1	3	57	-	-	3	58	-	-	3	71	-	-	10	266	-	-

Considering further folding space parameters The folding space of an RNA sequence is not only ruled by the used grammars, but also depends on other variables. We want to shed some light on the two factors "lonely base-pairs" and the "energy parameter set" in a very coarse grained fashion.

Stabilizing energy basically comes from stacking base-pairs. Thus, lonely base-pairs should in principle not occur; also energy parameters are not available for those situations. However, algorithmically, inclusion of lonely base-pairs is not a problem. Programs like RNAFOLD allows them on default. Inclusion of lonely base-pairs for our grammars is controlled by the syntactic filter = LP. Figure 3.12 illustrates the impact of lonely base-pairs.

We applied the sampling heuristics – as done before – to all "S-full"sequences twice, to produce lists of shape classes ranked by their probability. The reference run is without lonely base-pairs, the second run allows for lonely base-pairs. We identify the *rank* of the reference shape in the reference shape class list and compare it to the rank of the same shape in the second list. This difference is the *rank shift*; an asymmetric measure. Since we expect to observe similar tendencies throughout all four grammars and all five shape levels we merge these results. Figure 3.12 is basically a histogram over all $2,522 \cdot 4 \cdot 5$ rank shifts between lonely base-pairs (blue) and no lonely base-pairs (red) in the folding space(s). If the reference shape is not among the samples structures the rank shift is $\pm\infty$ (outermost vertical bars). To break down the results to one single number, the mean rank shift is also provided.

In conclusion, lonely base-pairs do not only raise computational demands, they also worsen the predictions.

A second factor on prediction performance is the used set of energy parameters. Even if the folding space, i. e. the grammar, remains unchanged, results can change to a great degree if the same candidates are scored differently. Figure 3.13 is constructed in the same way as Figure 3.12, but 7 non-standard parameter sets are related to the default "rna_turner2004.par". Confirmatively, both DNA sets ("dna_mathews1999.par" and "dna_mathews2004.par" from the Vienna-Package), score worst. The update from "rna_turner1999.par" to "rna_turner2004.par" (both from Vienna-Package) seems worth the effort, since predictions improve. The four *andronescu* sets [4] can improve predic-

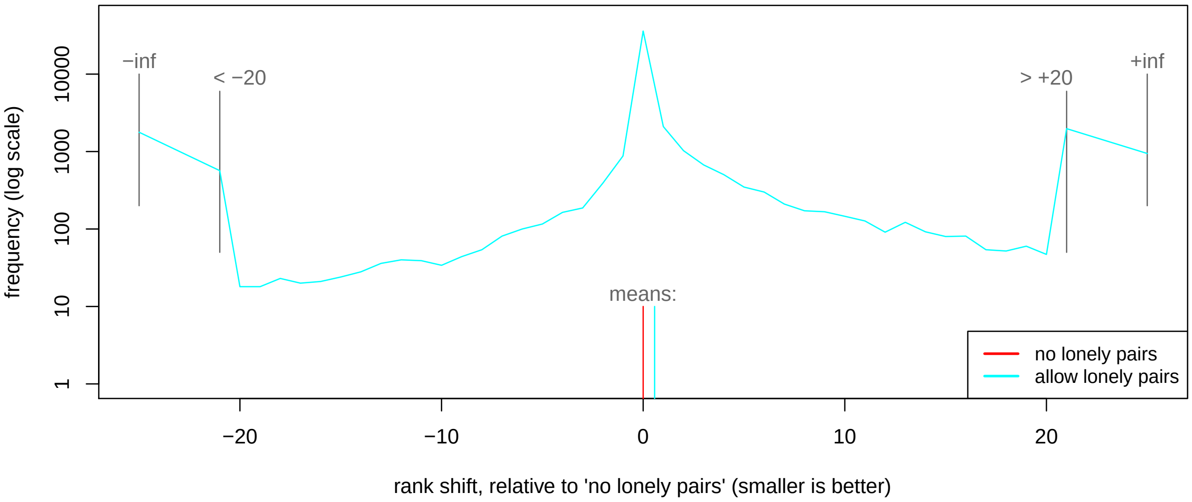


Figure 3.12: Influence of the presents of lonely base-pairs on prediction results.

tions even beyond the “rna_turner2004.par” set – at least for “S-full”sequences. But this impression can be misleading. These parameter sets have been trained with exactly the “S-full”set itself, thus we might observe some over-fitting effects.

3.4 Conclusion

3.4.1 Model comparison

Summing up our observations from model comparison and model performance evaluation, we conclude the following:

Conclusion 1 *For prediction of a single structure, there is no better alternative (among the models considered) than* UNAFOLD*, possibly augmented to report ALL structures with the optimal MFE value as in MicroState, when several exist.*

However, with such augmentation, a filter must be provided to safeguard against co-optimal microstates of the same optimal macrostate being reported.

Conclusion 2 *The distortion of shape probabilities caused by state space explosion (MacroState versus MicroState) is smaller than the one caused by over- or underestimating energies (MacroState and MicroState versus NoDangle or OverDangle).*

Models being so similar leads us to the question of run-time effort.

Conclusion 3 *Since results between models MacroState and MicroState differ only marginally, MicroState may be used for probability calculation. The higher computational effort of MacroState is not justified.*

In the light of the previous conclusions we find:

Conclusion 4 *On longer sequences, the only remaining virtue of MacroState appears to be its ability to enumerate sub-optimal structures with correct energies, and without redundancy.*

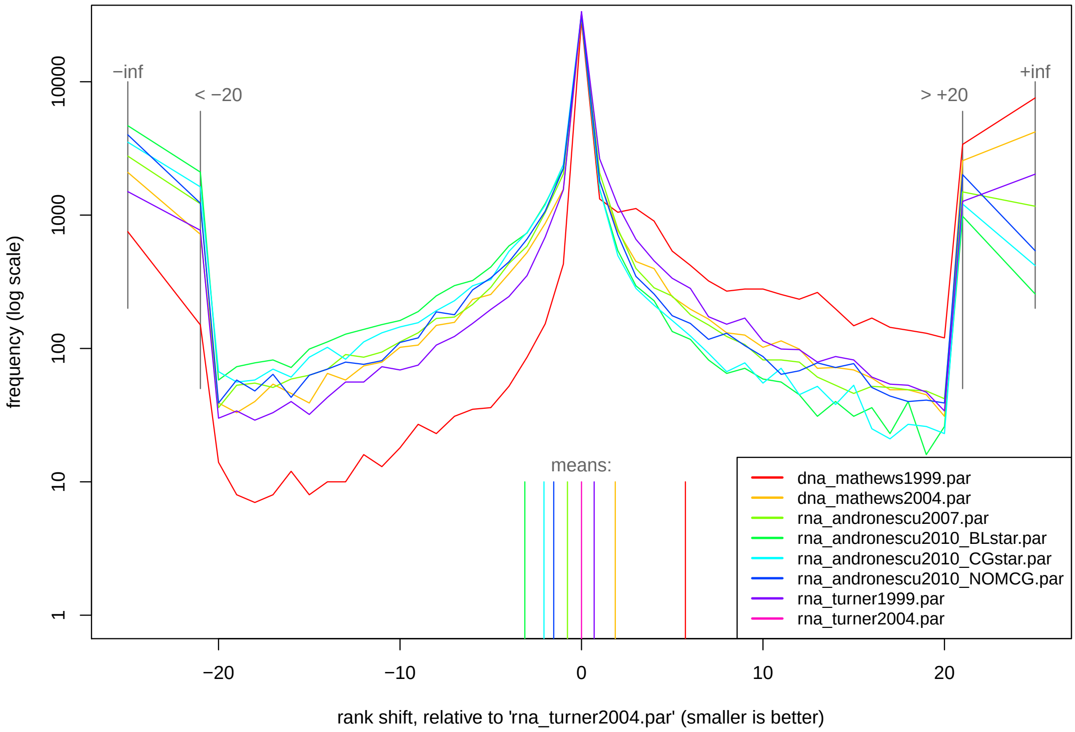


Figure 3.13: Influence of energy parameter sets on prediction results.

Furthermore, we make the

Conclusion 5 *Best default energy parameters are "rna_turner2004.par" while disallowing lonely base-pairs.* Although Andronescu parameters from various machine learning approaches could achieve better results for "S-full", we suspect to see over-fitting effects.

This answers the questions raised at the outset of this study.

3.4.2 Evaluation of further models

Our evaluation was focused on the models underlying the three programs RNAFold, RNAShapes, and RNAsubopt. There are many other folding programs out there. If these implementations adhere to the abstract models we present here in the form of tree grammars, our evaluation pertains to them as well. More likely, each implementation has its own peculiarities. In fact, one may think of extending our evaluation to models that are not based on thermodynamics at all, but are derived via machine learning techniques [19, 4]. These programs could be evaluated in the setting of this study in one of two ways. Either, the program source code is extended by the computation of abstract shapes and their shape probabilities (a useful feature anyway), and applied to our data sets directly. Or, the model behind the program is extracted as a tree grammar, coded in Bellman's GAP, and combined with existing modules for shape abstraction and partition function computations. Depending on the model differences, extracting the grammar behind the code may come down to a few minor changes to the four models provided here.

Generally, the four models MacroState, MicroState, OverDangle and NoDangle are available as a starting point for future research into on thermodynamic RNA folding. Implemented in the Bellman's GAP language, these programs are especially easy to modify or extend, while the Bellman's GAP compiler provides automatic translation into efficient and correct dynamic programming algorithms. The complete source code of our four models is available at `http://bibiserv.cebitec.uni-bielefeld.de/fold-grammars/`.

3.4.3 A new strategy for level-2 shape probabilities?

Our observations about the performance of shape level 2 gives rise to the investigation of a new strategy. Recall that level 2 gives much stronger information than levels 5 or even 3. Level 2 records not only the overall arrangement of helices, but also reports and distinguishes internal loops, 5' and 3' bulges.

Over all sequences of “S-full”, consideration of (only) the fifteen most likely level-2 shapes (using MicroState) reports the reference shape in 35% of the cases, while for just 8% the reference shape is on a deeper rank and it is not worth to inspect the shape list for the remaining 57%, because their reference shape class it not among the sampled structures (data not shown). However, the cost of level-2 shape analysis becomes prohibitive for longer sequences. Our data show a slowdown factor of 6.8 (on average for all grammars) over level-5 analysis, which should become even worse for longer sequences. Therefore, we conclude

Conclusion 6 *A strategy to efficiently compute level-2 shapes for long sequences is desirable*

Let us sketch a strategy how this speed-up can be achieved, borrowing ideas from the RapidShapes method [43]. Directly accessing the complete level-2 shape space of a long sequence appears infeasible. But we can compute a level-5 analysis at 90% or 100% coverage quickly, by reporting a small number top-ranking level-5 shapes. For these shapes, we can generate a thermodynamic matcher [43] to perform a separate level-2 analysis within each of the reported level-5 shape classes. Generating such a matcher as a tree grammar, encoded in Bellman's GAP, plus its subsequent compilation has negligible run-time. This should reduce the computational effort (which results from the number of shapes) considerably. While this is not mathematically guaranteed to yield the most likely level-2 shape, the idea appears promising.

3.4.4 A word on longer sequences

In comparison to our publication [45], we used a much larger test set with longer sequences, which required heuristic shape probability computations. Overall, we see the same trends but with small differences:

1. Best MFE prediction is now achieved by UNAfold, instead of MicroState/-MacroState (Figure 3.7).

2. Reference shape matches most often the dominant shape for MicroState, not for MacroState (Table 3.10).

3. Run-time (Table 3.9) and memory consumption (Table 3.8) are more homogeneous and better fit to our theoretical arguments, probably because the impact of the overhead for starting a process compared to pure computation time is reduced due to longer sequences.

4. MFE prediction results, as well as shape probability analysis are much worse. This is to be expected, because of the sheer length and thus size of the folding spaces for the longer sequences.

3.5 Acknowledgments

Thanks go to Michael Zuker for comments on the energy model and the UNAfold program, to Georg Sauthoff for support with the Bellman's GAP system and to Andreas Bremges for the SPS measure. We also thank the anonymous reviewers for helpful comments on the manuscript.

4 RapidShapes

The chapter at hand follows our Bioinformatics publication [43]. Since probabilistic shape analysis has been introduced and motivated in Chapter 3, we do not repeat much of these arguments and basic explanations here. In short, shape probabilities have been used to assign significance levels to predicted miRNA precursors in large scale studies [8, 53]. Consensus shapes for a set of sequences can be determined quickly and have been used as the basis of consensus structure prediction [72]. Finally, it has been determined that RFAM families can be indexed by their shape spectra, which leads to a significant speed-up of Rfam searches [44].

We will not retell ADP concepts here, since they have been introduced in Chapter 2.1. The original article is about shape level 5 for $\mathcal{G}_{\text{MacroState}}$. Considering results of Chapter 3, we now pass over to $\mathcal{G}_{\text{OverDangle}}$. We give detailed descriptions of $\mathcal{G}_{\text{OverDangle}}$ in level 5 in the following section and move those details for the other grammars and shape levels to the appendix section 9.1 for clarity.

4.1 Introduction

4.1.1 Computational cost of probabilistic shape analysis

While standard MFE folding algorithms have the asymptotic run-time of $\mathcal{O}(n^3)$, where n is the sequence length, probabilistic shape analysis has a run-time of $\mathcal{O}(rn^3)$. The value r is the (expected) number of *all* shapes encountered in the folding space of the analysed RNA sequence. The asymptotic number of *all* shapes of *all* sequences of length n has been determined to be $1.20^n \cdot 5.13 \cdot n^{\frac{-3}{2}}$ [51, 66], but the expected number of shapes encountered for an individual sequence is not known. Rudimentary measurements in [91] indicate a value of $r \approx 1.1^n$. As a consequence of this exponential factor, exact probabilistic shape analysis for the random sequence of length 125bp from the "S-full"test-set requires about 2.7 hours and 2.7 GB memory on the evaluation hardware (see Section 3.3.2). All longer sequences cannot be computed within 8 GB of memory.

It is clear that when computing the probabilities of *all* shapes, a factor related to the number of shapes cannot be avoided. But in practice, we are interested only in a handful of top-ranking shapes. We can compute the top k shapes ranked by shrep energy in $\mathcal{O}(kn^3)$ time – so it may seem surprising that this should not be possible for shapes ranked by probability. The explanation is that shape probabilities add up from many small contributions, and there is no way to determine early which sub-shapes may later contribute to the top ranking ones. In other words, Bellman's Principle of Optimality (see Section 2.1.1), the prerequisite of dynamic programming algorithms, does not apply for shape probability computation.

There may be a different approach to compute shape probabilities in polynomial time, but the chances do not look good: Problems of this type are closely related to the path labeling problem for hidden Markov models, shown to be NP-hard in [11].

For this reason of computational expense, the RNAShapes program also provides two heuristics: One is a *low probability filter* that excludes sub-shapes of very low probability, e. g. less than 10^{-6}. This implies that the overall partition function appears smaller than it is. The other heuristic is a *sampling* mode, akin to Sfold, but defining clusters *a priori* as shapes. Still, sampling gives up on computing exact probabilities, and also becomes expensive for longer sequences when a large number of samples must be drawn. Therefore, we set out here to provide a run-time heuristic to compute the shapes of highest probability. A *run-time* heuristic means that we still compute the exact probabilities, efficiently in many cases, but with no guarantee for polynomial run-time in general.

4.1.2 Outline of ideas

Let s be the sequence under consideration, and $length(s) = n$. The partition function $Q(s)$ of the complete folding space $F(s)$ can be computed efficiently in $\mathcal{O}(n^3)$ time. Now consider a specific shape p, such as `[][]`. All the structures of shape p constitute a sub-space $F_p(s) \subset F(s)$. The probability of p is

$$Prob(p, s) = Q_p(s)/Q(s), \text{ where } Q_p(s) = \sum_{x \in F_p(s)} e^{\frac{-E_x}{RT}} \tag{4.1}$$

This means that $Q_p(s)$ is itself the partition function of $F_p(s)$, and with a special program that folds s exactly and only into the structures of $F_p(s)$, we can compute $Q_p(s)$ efficiently in $\mathcal{O}(n^3)$ time.

A program folding an RNA sequence into a restricted set of structures, using the standard energy model, is called a thermodynamic matcher (TDM) [72]. Such TDMs are produced, for example, via the tool Locomotif, where a user composes pictures of annotated RNA structures, which are then compiled into programs for RNA motif search [69]. We use a similar TDM generator which, given an abstract shape p, generates the TDM for p, which computes the partition function.

The overall idea is as follows:

1) We compute $Q(s) = \mathcal{G}_{\text{OverDangle}}(\mathcal{I}_{bwe}^{OverDangle}, s)$ in $\mathcal{O}(n^3)$.

2) We enumerate (heuristically) a series of promising shapes $p_1, p_2, \ldots$

3) For each p_i, we generate TDM_i as a program coded in ADP style.

4) We compile TDM_i and execute it to compute $Q_{p_i}(s) = \mathcal{G}_{\text{TDM}_i}(\mathcal{I}_{bwe}^{OverDangle}, s)$ in $\mathcal{O}(n^3)$, and obtain $Prob(p, s)$ according to Eq. 4.1.

5) We continue until a specified portion of the shape probabilities is exhausted.

Since the time for TDM generation and compilation is small compared to their execution time, overall run-time of this heuristics is $\mathcal{O}(tn^3)$ where t is the number of TDMs (shapes) used. We call this approach RapidShapes.

4.2 A method for faster shape probability computation

4.2.1 Basic problem: Shapes with a least T% probability

Here we define the standard problem we want to solve efficiently; later we shall discuss variations of it. Let us set up a probability threshold T for the shapes of interest, say 0.9 or 0.6 as used in [8], or maybe as low as 0.1. Given T, only a constant number of shapes (1 with the first two settings, 9 for the third, or $\lceil (1/T) - 1 \rceil$ in general) can meet the threshold – independent of the sequence length n. Note that there may be no shapes at all meeting the threshold. On the other hand, there is a large population of shapes living in sub-thresholdia, their number growing exponentially with n.

Problem Definition

Given an RNA sequence s of length n and a threshold $0 < T \leq 1$, compute all shapes p of s with $Prob(p) \geq T$, together with their shape representative structures and free energy.

This definition permits that some shapes with sub-threshold probability will also be computed, but the goal is, of course, to minimize our efforts spent on those. We have to solve two sub-problems, namely the analysis of sub-spaces $F_p(s)$ and the generation of a good list of "promising" shapes.

4.2.2 Analysis of the folding space partitioned by shape

A thermodynamic matcher (TDM) folds a sequence only to a restricted set of structures. For RapidShapes, such a restricted set of structures comprises all structures of a particular shape (and no other). A TDM can compute the partition function value $Q_p(s)$ of its restricted folding space $F_p(s)$ in $\mathcal{O}(n^3)$ time, just as the unrestricted RNA folding program does for the complete folding space $F(s)$. Since $Q_p(s)$ is the sum of *all* structures in $F_p(s)$, and $F_p(s)$ is a precise subset of $F(s)$, we have to ensure that a TDM folds exactly those structures constituting the shape class p. Our strategy is to generate such programs on demand.

Generating grammars from shapes

Given shape p, we generate the specialized grammar $\mathcal{G}_p$. We do so using another tree grammar $\mathcal{G}_{\mathrm{TDM}_5}$ (see Figure 4.1), which parses a shape representation as its input in a syntactically non-ambiguous way. The rules of $\mathcal{G}_{\mathrm{TDM}_5}$ follow the intention of the shape

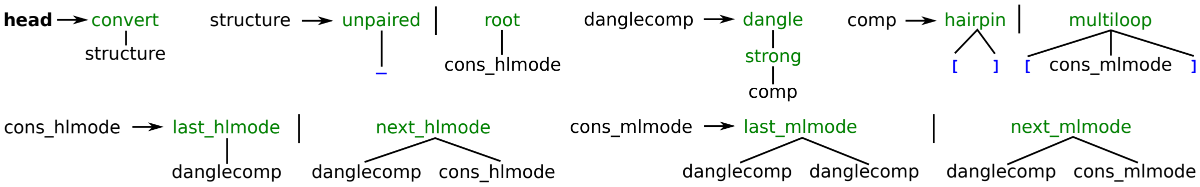


Figure 4.1: Grammar $\mathcal{G}_{\mathrm{TDM}_5}$ to generate specialized tree grammars. Input for this grammar is a shape string like `[][]`.

abstraction in its respective shape level: Recognizable structural sub-*comp*onents in level 5 are either hairpins or multiloops. A component has either some others *next* to it or is the *last* in an – otherwise maybe empty – row of components. This must hold for a row of *cons*ecutive components (*cons_hlmode*) as well as for a list of components enclosed by a multiloop (*cons_mlmode*). Rows within a multiloop must have at least two components, cf. last_mlmode. Aspects like base-dangling and lonely base-pairs are treated by the algebra functions dangle and strong, respectively. Every structure starts at the root or is completely unpaired. Some general transformations will be done in the function convert.

For our example $p =$ `[][]` (see left column of Figure 4.2), $t_{\texttt{[][]}}$ is the unique parse of `[][]` with $\mathcal{G}_{\mathrm{TDM}_5}$. The restricted tree grammar $\mathcal{G}_{\texttt{[][]}}$ for folding exactly those structures of shape class `[][]` is constructed by applying a grammar-generating evaluation scheme $\mathcal{I}_{tdm\_OverDangle\_5}$ (see Table 4.1) to $t_{\texttt{[][]}}$.

The result is shown in the right column of Figure 4.2. The important difference to $\mathcal{G}_{OverDangle}$ is that the non-terminals are now position specific - or better helix specific - by the addition of magenta coloured subscripts to their names. These subscripts reflect the sub-shape of its non-terminals. Furthermore, the alternatives of $\mathcal{G}_{OverDangle}$ are reduced for some positions. For example, the rule $struct^{\texttt{[][]}}$ lacks the nil alternative, while $struct^{\_}$ lacks cadd.

The parse-tree $t_{\texttt{[][]}}$ has the repetitive sub-tree dangle(strong(hairpin(`[`, `]`))). Thus, our generator would create those grammar rules twice, leading to a waste of run-time and memory consumption. Since a grammar is a *set* of rules, and non-terminals are helix specific, multiply generated rules will collapse to just one instance, saving re-computation of whole RNA sub-structures by using dynamic programming.

On the technical side, this is realized by using a two level `C++` hash data-structure – called *rules* – as the answer type of $\mathcal{I}_{tdm\_OverDangle\_5}$. The first level key is the helix specific non-terminal name, e. g. $stack^{\texttt{[]}}$; second level key is the right hand side of the production, e. g. sr(`b`, weak$^{\texttt{[]}}$, `b`). The actual hash *value* is always empty, because we just utilize the hash's uniqueness property for keys to store our grammar rules. In Table 4.1 the → symbol separates first and second key. To be concise, `C++` data type *rules* also keeps track of the current shape string, which is used to specialize the non-terminals to their helix contexts (magenta subscripts in Table 4.1).

The only purpose of the convert algebra function, which prefixes every candidate of $\mathcal{G}_{\mathrm{TDM}_5}$, is to convert the *rules* data-type into a plain ASCII string of Bellman's GAP

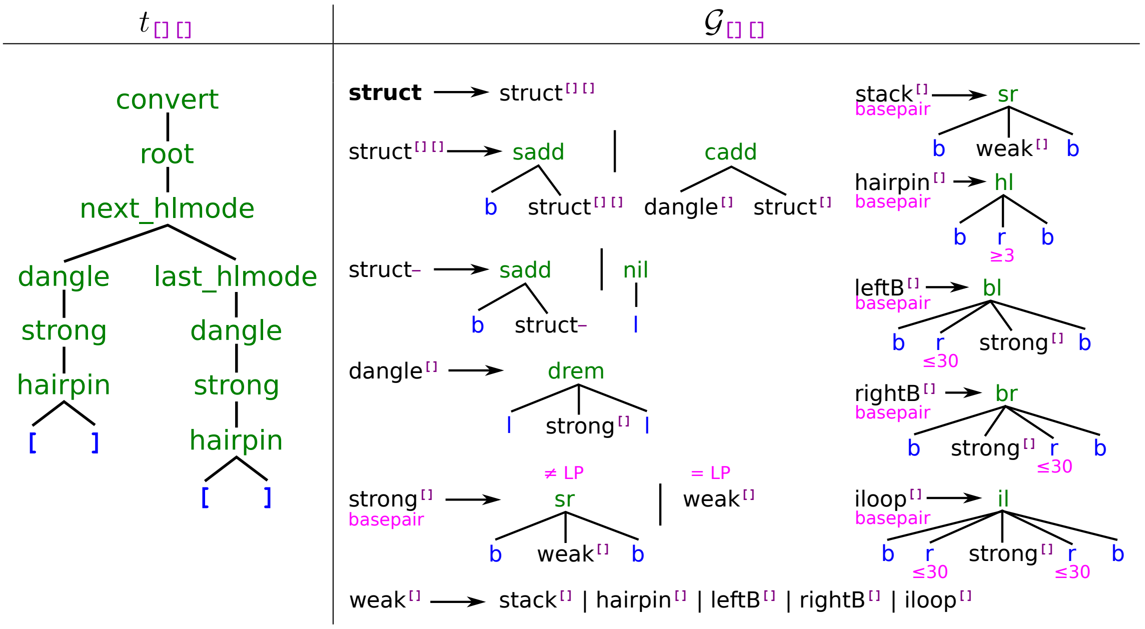


Figure 4.2: Left: the unique parse tree for the input shape string [] []. Right: the generated Grammar $\mathcal{G}_{[][]}$, which is specialized to enumerate all but no other secondary structures than those whose shape string is [] [].

source code and to encapsulate the pure grammar source with an appropriate name in Bellman's GAP syntax. Thus, $\mathcal{I}_{tdm\_OverDangle\_5}$ in fact uses two different *sorts*; default is *rules* but the result of convert is of type *code*, cf. Signature Σ_{TDM} in Table 4.2. To avoid defining two choice functions for the two *sorts*, we simply omit application of a choice function to the non-terminal *head*.

Different shape levels deal with different types of shape strings, e. g. the _ character is only present in levels 2 and 1 if we ignore the degenerated case of the completely unpaired structure. Thus, we need different shape string parsing grammars $\mathcal{G}_{\mathrm{TDM}_i}$ for different shape levels i. Details are given in the appendix Section 9.1. In short, grammars for lower levels require more complex rules and thus more different algebra functions. However, we just want to deal with one common signature Σ_{TDM}, given in Table 4.2, fitting all $\mathcal{G}_{\mathrm{TDM}_i}$ grammars. Since $\mathcal{G}_{\mathrm{TDM}_5}$ only uses the first 10 and the convert algebra functions, defined in Σ_{TDM}, we decided to move definition of the remaining 72 algebra functions into Table 9.1 of the appendix.

Generation of a specialized grammar $\mathcal{G}_p$ for shape string p of level 5, following the OverDangle architecture, is achieved by executing the Bellman's GAP instance

$$\mathcal{G}_{\mathrm{TDM}_5}(\mathcal{I}_{tdm\_OverDangle\_5}, p).$$

Using the standard energy model Although $Q(s)$ and $Q_p(s)$ are computed by two independent programs, they have to fold the same structures *and* evaluate them to the same energy values, i. e. Boltzmann weighted energies. We can assure this by using the

Table 4.1: TDM generating evaluation algebra for grammars following the OverDangle concept in shape level 5. Algebra functions in the left column stem from parsing the shape string with $\mathcal{G}_{\mathrm{TDM}_5}$, while the green algebra functions on the right column will be part of the generated, specialized grammar.

algebra function	$\mathcal{I}_{tdm\_OverDangle\_5}$
$\mathrm{convert}(x)$	$bgap(x)$
$\mathrm{root}(x)$	$\mathrm{struct} \rightarrow \mathrm{struct}^{\texttt{x}}$
	$\mathrm{struct}^{\texttt{-}} \rightarrow \mathrm{sadd}(\texttt{b}, \mathrm{struct}^{\texttt{-}})$
	$\mathrm{struct}^{\texttt{-}} \rightarrow \mathrm{nil}(\texttt{l})$
$\mathrm{dangle}(x)$	$\mathrm{dangle}^{\texttt{x}} \rightarrow \mathrm{drem}(\texttt{l}, \mathrm{strong}^{\texttt{x}}, \texttt{l})$
$\mathrm{next\_hlmode}(x, y)$	$\mathrm{struct}^{\texttt{xy}} \rightarrow \mathrm{sadd}(\texttt{b}, \mathrm{struct}^{\texttt{xy}})$
	$\mathrm{struct}^{\texttt{xy}} \rightarrow \mathrm{cadd}(\mathrm{dangle}^{\texttt{x}}, \mathrm{struct}^{\texttt{y}})$
$\mathrm{last\_hlmode}(x)$	$\mathrm{struct}^{\texttt{x}} \rightarrow \mathrm{sadd}(\texttt{b}, \mathrm{struct}^{\texttt{x}})$
	$\mathrm{struct}^{\texttt{x}} \rightarrow \mathrm{cadd}(\mathrm{dangle}^{\texttt{x}}, \mathrm{struct}^{\texttt{-}})$
$\mathrm{unpaired}(a)$	$\mathrm{struct}^{\texttt{-}} \rightarrow \mathrm{struct}^{\texttt{-}}$
	$\mathrm{struct}^{\texttt{-}} \rightarrow \mathrm{sadd}(\texttt{b}, \mathrm{struct}^{\texttt{-}})$
	$\mathrm{struct}^{\texttt{-}} \rightarrow \mathrm{nil}(\texttt{l})$
$\mathrm{strong}(x)$	$\mathrm{strong}^{\texttt{x}} \rightarrow \mathrm{sr}(\texttt{b}, \mathrm{weak}^{\texttt{x}}, \texttt{b})_{bp, \neq LP}$
	$\mathrm{strong}^{\texttt{x}} \rightarrow \mathrm{weak}^{\texttt{x}}_{= LP}$
	$\mathrm{weak}^{\texttt{x}} \rightarrow \mathrm{stack}^{\texttt{x}}$
	$\mathrm{weak}^{\texttt{x}} \rightarrow \mathrm{leftB}^{\texttt{x}}$
	$\mathrm{weak}^{\texttt{x}} \rightarrow \mathrm{rightB}^{\texttt{x}}$
	$\mathrm{weak}^{\texttt{x}} \rightarrow \mathrm{iloop}^{\texttt{x}}$
	$\mathrm{stack}^{\texttt{x}} \rightarrow \mathrm{sr}(\texttt{b}, \mathrm{weak}^{\texttt{x}}, \texttt{b})_{bp}$
	$\mathrm{leftB}^{\texttt{x}} \rightarrow \mathrm{bl}(\texttt{b}, \texttt{r}_{\leq 30}, \mathrm{strong}^{\texttt{x}}, \texttt{b})_{bp}$
	$\mathrm{rightB}^{\texttt{x}} \rightarrow \mathrm{br}(\texttt{b}, \mathrm{strong}^{\texttt{x}}, \texttt{r}_{\leq 30}, \texttt{b})_{bp}$
	$\mathrm{iloop}^{\texttt{x}} \rightarrow \mathrm{il}(\texttt{b}, \texttt{r}_{\leq 30}, \mathrm{strong}^{\texttt{x}}, \texttt{r}_{\leq 30}, \texttt{b})_{bp}$
$\mathrm{hairpin}(a, b)$	$\mathrm{weak}^{\texttt{[]}} \rightarrow \mathrm{hairpin}^{\texttt{[]}}$
	$\mathrm{hairpin}^{\texttt{[]}} \rightarrow \mathrm{hl}(\texttt{b}, \texttt{r}_{\geq 3}, \texttt{b})_{bp}$
$\mathrm{multiloop}(a, x, b)$	$\mathrm{weak}^{\texttt{[x]}} \rightarrow \mathrm{multiloop}^{\texttt{[x]}}$
	$\mathrm{multiloop}^{\texttt{[x]}} \rightarrow \mathrm{ml}(\texttt{b}, \mathrm{ml\_comps}^{\texttt{x}}, \texttt{b})_{bp}$
$\mathrm{next\_mlmode}(x, y)$	$\mathrm{ml\_comps}^{\texttt{xy}} \rightarrow \mathrm{sadd}(\texttt{b}, \mathrm{ml\_comps}^{\texttt{xy}})$
	$\mathrm{ml\_comps}^{\texttt{xy}} \rightarrow \mathrm{cadd}(\mathrm{incl}(\mathrm{dangle}^{\texttt{x}}), \mathrm{ml\_comps}^{\texttt{y}})$
$\mathrm{last\_mlmode}(x, y)$	$\mathrm{ml\_comps}^{\texttt{y}} \rightarrow \mathrm{sadd}(\texttt{b}, \mathrm{ml\_comps}^{\texttt{y}})$
	$\mathrm{ml\_comps}^{\texttt{y}} \rightarrow \mathrm{incl}(\mathrm{dangle}^{\texttt{y}})$
	$\mathrm{ml\_comps}^{\texttt{y}} \rightarrow \mathrm{addss}(\mathrm{incl}(\mathrm{dangle}^{\texttt{y}}), \texttt{r})$
	$\mathrm{ml\_comps}^{\texttt{xy}} \rightarrow \mathrm{sadd}(\texttt{b}, \mathrm{ml\_comps}^{\texttt{xy}})$
	$\mathrm{ml\_comps}^{\texttt{xy}} \rightarrow \mathrm{cadd}(\mathrm{incl}(\mathrm{dangle}^{\texttt{x}}), \mathrm{ml\_comps}^{\texttt{y}})$
$\mathcal{S}$	rules
objective function	id

Table 4.2: First part of the common Signature Σ_{TDM} used for generating TDMs. Shown are only the 11 algebra functions used by $\mathcal{G}_{\text{TDM}_5}$. We moved definition of the remaining 72 algebra functions to Table 9.1 of appendix Section 9.1.1.

$\text{root}(\mathcal{S})$
$\text{dangle}(\mathcal{S})$
$\text{next\_hlmode}(\mathcal{S}, \mathcal{S})$
$\text{last\_hlmode}(\mathcal{S})$
$\text{unpaired}(\mathcal{A})$
$\text{strong}(\mathcal{S})$
$\text{hairpin}(\mathcal{A}, \mathcal{A})$
$\text{multiloop}(\mathcal{A}, \mathcal{S}, \mathcal{A})$
$\text{next\_mlmode}(\mathcal{S}, \mathcal{S})$
$\text{last\_mlmode}(\mathcal{S}, \mathcal{S})$
$code = \text{convert}(\mathcal{S})$

identical evaluation algebra $\mathcal{I}_{bwe}^{OverDangle}$ (see Table 3.7 on page 47) for both programs. This comes for free, because the algebra functions in $\mathcal{G}_{OverDangle}$ and $\mathcal{G}_p$ are always the same.

Theorem 1. *$\mathcal{G}_{TDM_5}(\mathcal{I}_{tdm\_OverDangle\_5}, p)$ generates a TDM_p which correctly computes $Q_p(s)$.*

Proof. By construction, TDM_p recognizes unambiguously all structures of shape p. When applied to s, it exactly constructs $F_p(s)$ and hence computes $Q_p(s)$. That our implementation actually satisfies this mathematical property, can be systematically tested by checking $\sum_p Q_p(s) = Q(s)$. □

4.2.3 Heuristic Shape selection

To run RapidShapes for a given sequence s, we construct a list $L(s)$ of "promising" shapes. For each shape $p \in L(s)$, we construct TDMs and compute $Prob(p, s)$ in $\mathcal{O}(n^3)$ time. Ideally, $L(s)$ would only contain the shapes above the threshold, but this is exactly the problem to be solved.

Selection by shrep energies ($\mathbf{L_{energy}}$) The simple shape analysis for a given sequence s computes the top k shapes ranked by the energy of their shreps (cf. Section 3.1.2). The Bellman's GAP instance

$$\mathcal{G}_{OverDangle}\left(\left(\mathcal{I}_{\pi_5} * \mathcal{I}_{mfe}^{OverDangle}\right)^{\text{subopt}}, s\right)$$

without the *semantic* filter subopt would return the complete shape space with its according shrep energies. The filter subopt reduces computation to only that part of the shape space, which is top ranked according to shrep energies. Although shape ranks by shrep energy and shape ranks by probability do not agree, there is a positive correlation between shrep energies and shape probabilities. We start with a small k to compute $L(s) = [p_1, \ldots, p_k]$ by simple shape analysis. If $\sum_{i=1}^{k} Prob(p_i, s) < 1 - T$, we repeat the simple shape analysis with larger k to extend $L(s)$.

Selection by sampled frequencies ($\mathbf{L_{sampling}}$) Sampling of structures (via stochastical backtrace, see Section 3.3.4) to estimate shape probabilities can be done very fast, but the results are not exact, cf. Figure 3.9. However, the reported shapes may be the most likely ones, in spite of their probabilities being incorrect. To combine both advantages, we use the quickly calculated shapes from the sampling as members for $L(s)$ and precisely determine their probabilities via TDMs. Sampling requires a bound on the number of samples drawn, which was set to 10,000 for this study. This might overlook shapes with probability $\geq T$, but this chance can be decreased by drawing more structures.

4.2.4 Asymptotics

To satisfy the problem definition, given in chapter 4.2.1, RapidShapes must calculate the probability, the shrep structure and its free energy for all k shapes with $Prob(p) \geq T$. All three values can be computed for one shape by using specialized evaluation schemes with the TDM, where $Prob(p)$ is $Q_p(s)$ divided by $Q(s)$, which must be calculated just once. For the sake of speed, we separate the computation of shape probabilities for all shapes in $L(s)$ from the computation of shrep structures and free energy values for all k shapes, with $Prob(p) \geq T$, because usually $|L(s)| >> k$. This leads to an asymptotic run-time of $\mathcal{O}(n^3 + |L(s)| \cdot n^3 + k \cdot n^3 + l)$.

4.3 Evaluation

The technical environment for all measurements is described in Section 3.3.2. For the following evaluations we increased the main memory limit to $8GB$.

4.3.1 Evaluation setup

Our evaluation, summarized in Figure 4.3, uses two kinds of RNA sequences. On the one hand, the set “random” is compiled from every fifth sequence of *Random set 1* (Section 3.3.4), thus providing 200 sequences of lengths 5 up to $1,000$ nucleotides. On the other hand, we use the “real” set “S-full”, see Section 3.3.1.

We tested three different methods to fill the list of promising shapes $L(s)$ for a threshold of $T = 0.1$ for *Random set 1*:

- The green colored $L_{oracle}(0.1, s)$ is the optimal choice of shapes for RapidShapes.
- “Selection by shrep energies” $L_{energy}(0.1, s)$ is colored in red
- and “Selection by sampled frequencies” $L_{sampling}(0.1, 10000, s)$ for 10,000 samples per sequence is colored in blue.

To gain the measurements, particularly for the oracle, we ran RapidShapes with a very low threshold of $T = 0.01$ and a combined guessing mode. Shape list $L(s)$ was first filled by “Selection by sampled frequencies” with 10,000 structures. Should the combined shape probability mass not match 99%, we iteratively ran “Selection by shrep energies” with growing k.

By omitting the artificial oracle method, we could use normal versions of our program RapidShapes with $T = 0.1$ (black) and $T = 0.5$ (**gray**) for *Random set 1*. The remaining two methods “Selection by sampled frequencies” and “Selection by shrep energies” are visualized in Figure 4.3 with continuous and dashed lines, respectively.

The RapidShapes processes for the different sequences of both sets where allowed to compute TDMs in a round-robin fashion on our 33 nodes compute cluster. After consuming ≈ 17.4 CPU years, we stopped the evaluation. Thus, RapidShapes could not reach the required probability mass for all test sequences in the given time span. As a consequence, the 23% sequences of “S-full”, larger than 396 nucleotides, are not included in the evaluation. In the case of *Random set 1*, the longest computable sequence with at least 90% combined probability mass has a length of 570 nucleotides. Moreover, RapidShapes failed to discover the required 99% combined probability mass for sequences larger than 425 nucleotides. This means, that the number of used TDMs for $L_{oracle}(0.1, s)$ might be slightly overestimated, should RapidShapes have missed shape classes with low energy but high probability.

The shape-space $F(s)$, i. e. number of different shape classes for input sequence s, is determined by counting classes of $\mathcal{G}_{OverDangle}(\mathcal{I}_{\pi 5} * \mathcal{I}_{bwe}^{OverDangle}, s)$ with low probability filter set to 10^{-6}, colored in **magenta**, and without filtering, colored in **cyan** for sequences of *Random set 1*. Largest sequences, whose shape-space enumeration could fit within $8GB$ of memory, are 125 and 395 nucleotides long, without and with low probability filter, respectively.

Pure run-time of the stochastical backtracing process, aka “sampling” (cf. Section 3.3.4), with 10,000 drawn structures is given as the orange curve in part B) of Figure 4.3.

Oracle

To mark the theoretical maximum speed-up for RapidShapes, we assume an *oracle* which a priori denominates the shapes for $L(s)$, ordered by their shape probabilities. This would allow to use the minimum number of TDMs for any choice of T. Of course, such an oracle does not exist, but we can determine what it would have returned by dropping sub-threshold shapes from $L(s)$ after their evaluation.

4.3.2 Results on random data

The performance evaluation of RapidShapes has two aspects: The effective number of shapes which have to be evaluated, and the absolute gain in run-time.

Required number of TDMs

Part A) of Figure 4.3 is a comparison of the growth of $F(s)$ and $L(s)$ for different methods to fill this list with promising shapes.

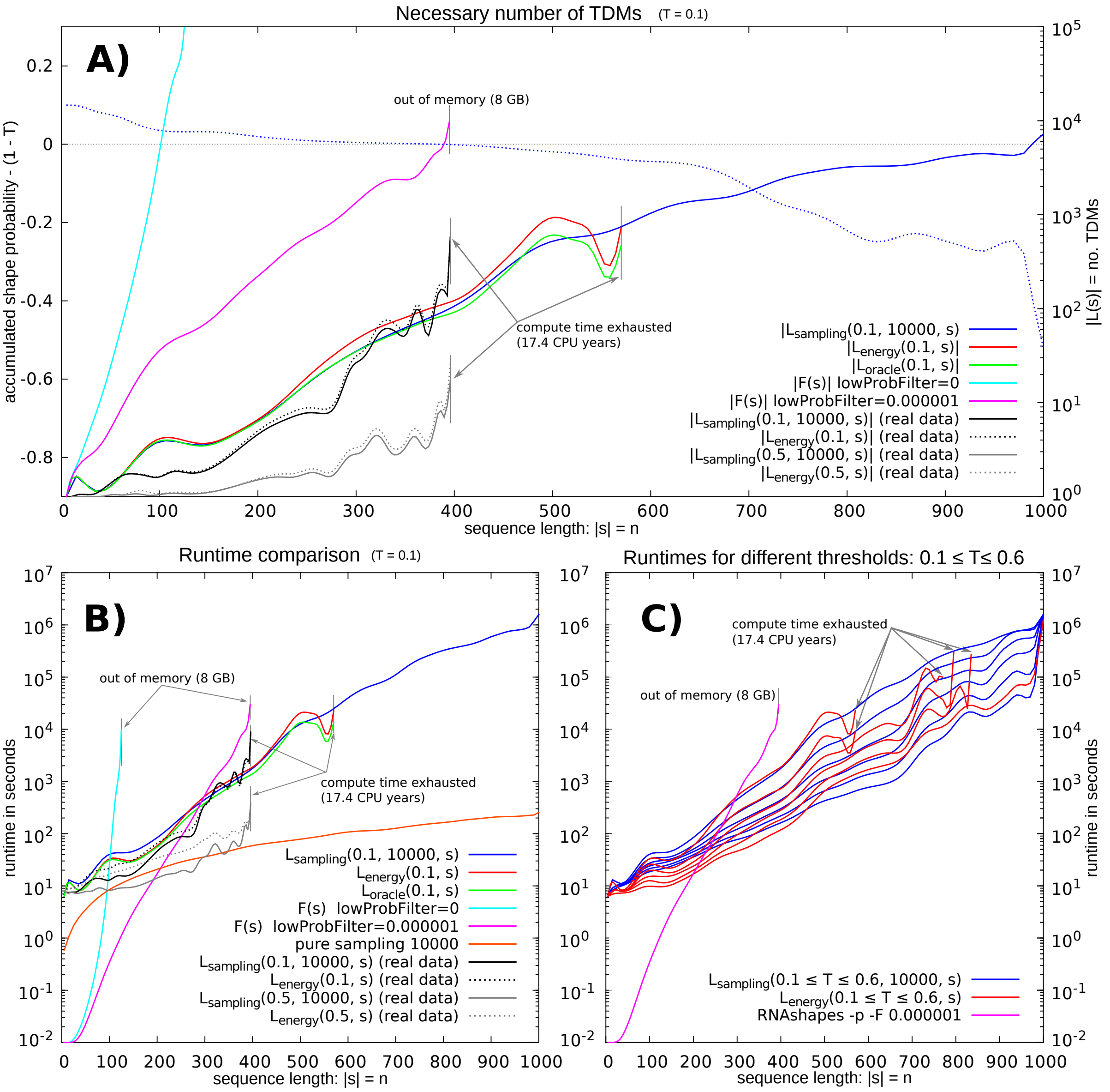


Figure 4.3: **A) Necessary number of TDMs** compares the growing numbers of shapes in $F(s)$ and $|L(s)|$. The x-axis is sequence length, the logarithmic scaled right y-axis is $|L(s)|$. Shape guessing methods "oracle", "energy" and "sample" for $T = 0.1$ for *Random set 1* as well as "sample" with continuous and "energy" with dashed lines for "S-full" for $T = 0.1$ (black) and $T = 0.5$ (gray) have been tested. See main text for details. The left y-axis depicts the amount of unaccounted folding space, i. e. the accumulated probability of all evaluated shapes minus $(1 - T)$. The dotted blue curve corresponds to $L_{sampling}(0.1, 10000, s)$.
B) Run-time comparison illustrates actual run-times instead of counting shapes. The log-scale y-axis is here run-time in seconds. Run-time for sampling is given in orange.
C) Run-times for different thresholds shows the influence of selecting different values for the threshold T.
All curves are smoothed via the GNUPLOT option "smooth bezier".

As expected, just a very small number of all existing shape classes in $F(s)$ (cyan colored curve) seems to account for a major part of all structure probabilities. The number of TDMs, which must be generated, compiled and executed for RapidShapes, is strikingly smaller than $|F(s)|$, regardless of the method for filling $L(s)$.

The *Selection by shrep energies* (red curve) is not perfect, observable by the somehow below green curve of the omniscient oracle, but the distance is not too wide and it follows the trend of the oracle.

Selection by sampled frequencies (blue curve) runs the risk of not creating enough shapes, due to the limited sample size. Sampling $10,000$ structures can result in at most $10,000$ different shapes, and normally much less. For larger sequences the accumulated probability of these shapes may not be sufficient to cover $1-T$. Where this strategy needs even less TDMs than the oracle, it comes with an increasing proportion of unexplored parts of $F(s)$. The amount of unaccounted folding space for the *Selection by sampled frequencies* method is indicated by the blue dotted curve in part A) of Figure 4.3.

It is interesting that the sampling strategy becomes faster than the energy based strategy exactly where it starts missing shapes above the threshold. We conclude that there lies no real advantage in the sampling strategy when computing *all* exact shape probabilities above the threshold is required.

Run-time speed-up

Different shape strings result in different TDMs, i. e. different grammar sizes. The larger the grammar, the higher is the run-time for generation, compilation and execution, independent of the input sequence. To include these effects into the evaluation results, part B) of Figure 4.3 shows our analysis of empirically measured run-times.

For short sequences, the overhead of constructing TDMs prevents RapidShapes from being useful, but with input sequences larger than 300 nucleotides there is a growing speed-up, compared to the Bellman's GAP instance $\mathcal{G}_{OverDangle}(\mathcal{I}_{\pi_5} * \mathcal{I}_{bwe}^{OverDangle}, s)$. Furthermore, the heuristic makes it possible to compute shape probabilities for sequences larger than 395 nucleotides, where $\mathcal{G}_{OverDangle}(\mathcal{I}_{\pi_5} * \mathcal{I}_{bwe}^{OverDangle}, s)$ runs out of memory.

The achieved speed-up depends on the choice of threshold T. Our previous measurements were made for a rather low threshold of $T = 0.1$. In practice, a threshold of 0.6 or even 0.9 makes sense when checking for the existence of a dominant shape. The larger T, the faster is RapidShapes– this is demonstrated by the different red and blue curves in part C) of Figure 4.3 for “selection by shrep energies” and “selection by sampled frequencies”, respectively.

4.3.3 Results on real data

Natural RNAs, whether functional or not, are not random sequences. Functional non-coding RNAs are known to be optimized for good folding energy, although this signal is not strong enough to discern functional from non-functional RNA. All natural RNA has a bias towards lower folding energy than random sequences. Cf. [16] and the long debate summarized therein. We can expect this bias to favor a small number of shapes with a high probability over a more even distribution in real RNAs. This fact must lead to RapidShapes requiring a smaller number of TDMs and hence becoming faster. This result is confirmed on the "real" data testset "S-full". Returning to part A) of Figure 4.3, consider the black and the grey line, computed with $T = 0.5$ and $T = 0.1$, respectively. The smaller number of required TDMs also results in a moderate speed-up.

872 out of these 3,245 sequences are longer than $300nt$, the break-even point (for $T = 0.1$) where RapidShapes starts to become faster than RNAshapes and smaller than $396nt$ to not crash RNAshapes with 8 GB memory. While RNAshapes with activated filtering consumes $693.6h$ to calculate shape probabilities, RapidShapes takes only $227.0h$ of run-time. This speed-up factor is expected to significantly grow for larger sequences.

4.4 Discussion

4.4.1 Speed-ups and brake-even points achieved

Abstract shape probabilities, as computed by RNAshapes, provide useful information beyond minimum free energy folding. Due to the large computational cost, previously, a sampling heuristic had to be used for sequences longer than about 300 bases. This heuristic, however, has the disadvantages that (1) it does not return exact probabilities, (2) does not account for the part of the folding space not covered by the sampling, and (3) does not return shape representative structures. The average shape probability shift (SPS) between sampled and exact shape probabilities for all testsequences is 0.025. In a case where a particular shape clearly dominates all others, these disadvantages do not really matter, as this shape will be sampled many times (dwarfing the probabilities of other shapes), and the shape representative structure has a good chance to be in among the sample. However, one cannot know in beforehand whether this situation applies.

The approach RapidShapes presented here overcomes these limitations and enables the computation of shape probabilities to a much wider sequence range. It is a *run-time* heuristic – i. e. the computed probabilities are exact, we obtain the shape representative structures, and we know about the uncovered amount of probability in the folding space. What cannot be guaranteed is polynomial run-time in a strict asymptotic sense, but the evaluation shows that RapidShapes performs well in practice.

The speed-up achieved by RapidShapes depends on the threshold and on the sequence length. Using a very relaxed threshold $T = 0.1$, RapidShapes becomes faster than the traditional method (using its low probability filter) at sequence length 300. At sequence length 395, RapidShapes is faster by a factor of 3.6. Taking a more stringent threshold at $T = 0.6$, RapidShapes becomes faster at sequence length 265, and at sequence length 395, the speed-up factor is about 45.6. Independent of threshold, RapidShapes is the only practical method to compute exact shape probabilities for sequences longer than $395nt$.

Expected numbers of shapes above a probability threshold

The combinatorics of shapes has found considerable interest recently, but the expected number of shapes of a sequence of length n is still unknown. Our large-scale evaluation has produced some empirical data in this respect. Assuming a simple exponential growth pattern of $\mathcal{O}(\alpha^n)$, we can estimate α. Our data (part A) of Figure 4.3) suggest that $\alpha = 1.09495^n$ for the number of all shapes of a sequence of length n (cyan curve), $\alpha = 1.02331$ for all shapes with a probability larger than 10^{-6} (magenta curve), and $\alpha = 1.01156^n$ for all shapes with probability larger than $T = 0.1$ (green oracle). RapidShapes computes $\alpha = 1.01156^n$ shapes (red curve) and hence is so close to optimal, that it cannot be distinguished at this level. For $T = 0.5$, the oracle value is $\alpha = 1.00584$ (data not shown). All estimations have been computed via gnuplot's fitting capability. The function to fit is $f(x^\alpha)$ with an initial α-value of 1.1.

4.4.2 Problem variants

k-best shapes

As explained initially, we can efficiently compute the k best shapes of sequence s ranked by shrep energy, but not ranked by probability. In order to compute the k best shapes ranked by probability, we compute shape probabilities according to the L_{energy} strategy. Assuming $K \geq k$ shapes have been computed, and $p_1, \ldots, p_k$ are the best k shapes seen so far, we can stop computation as soon as $1 - \sum_{i=1}^{K} Prob(p_i) \leq Prob(p_k)$.

Best shape only

If we ask for the best shape no matter how small its probability is, we can do no better than to apply the above strategy for $k = 1$. However, if we are interested in the existence of a *dominant shape*, loosely defined here as one which is more likely than all the rest together, we just solve the standard problem with threshold $T = 0.5$.

4.4.3 Implementation alternatives

Algorithm parameterization versus generation

From an algorithmic point of view, or method of generating, compiling and running algorithms for sub-problems on the fly appears somewhat unusual. As an alternative, one could think of modifying the code of the traditional method to accommodate a target shape p as an extra parameter, and restrict the folding to structures which match p running through list L_{energy}. This would turn the general method into the equivalent of a TDM for p. However, this would slow down the inner loop of an $\mathcal{O}(n^3)$ algorithm, whereas generating and compiling takes $\mathcal{O}(n)$ time (empirically: less than 7,76% of the overall process) to yield an $\mathcal{O}(n^3)$ TDM algorithm without such a slowdown. And besides, generating TDMs from shapes has other applications, e. g. in RNA motif search.

Other shape enumeration heuristics

We have experimented with other ideas of enumerating promising shapes, such as (1) using a precompiled library of frequently encountered shapes, or (2) always computing low complexity shapes (such as `[]`) first. Neither of these ideas has provided an improvement. When computing shape probabilities for abstraction levels $i < 5$, a good strategy may be to first compute best shapes of level $i + 1$ and then computing their sub-shapes at level i. This is possible since shape abstraction levels form a hierarchy. However, this idea has not been further explored yet.

4.4.4 Open problems

Given a particular TDM, it is easy to generate a scanning version to find high probability instances of its shape in a longer sequence. An adaptive window size, subject to a reasonable upper bound, also seems feasible. However, thinking of RNA gene prediction based on dominant shapes, we would need a scanning version which dynamically changes the shape as it moves along the sequence. This presents a challenge for future research.

Our technique does not depend on the concrete information which is accumulated for each shape. Recent approaches such as CONTRAfold [19] and CG [3] replace the classical thermodynamic model by stochastic models, trained from structural data via machine learning techniques. To benefit from our approach, these methods need to be augmented to support shape abstraction. Technically, they are based on different grammars than RNAshapes. For these grammars, shape abstraction functions need to be defined and implemented. Then, our TDM generator and the strategies described here should carry over without change.

4.5 Acknowledgments

The authors thank Jens Reeder for providing the TDM generator from Locomotif, Georg Sauthoff for the ADP to C code generator (Bellman's GAP), Robert Homann for his program randomseq and Stefanie Schirmer for careful proofreading and discussions. Special thanks go to Jana Raupach and Philipp Schubert for taking RapidShapes online: `http://bibiserv.cebitec.uni-bielefeld.de/rapidshapes/`.

5 pKiss

The following chapter is based on our publication "Prediction of RNA secondary structure including kissing hairpin motifs" [87], which we extend here by details about the Bellman's GAP implementation and a thorough evaluation to compare run-time and accuracy to a set of different pseudoknot prediction programs.

5.1 Introduction

5.1.1 Biological relevance of pseudoknots in RNA structure

RNA is a chain molecule, the activated form of genetic information in all living organisms. Folding back onto itself, RNA forms secondary structure via base-pairing of complementary nucleotides. Stacks of base-pairs form helices, akin to the Watson-Crick helix of DNA, but with base-pairs (A, U), (G, C), (G, U), and occasionally some non-standard pairs. Ultimately, a tertiary (spatial) structure forms which is essential for biological function. *Pseudoknots* are structural motifs also defined via base-pairing patterns, but, as they form late in the folding process, are generally considered as elements of tertiary structure.

Kissing hairpins are a common RNA folding motif belonging to the class of pseudoknots. The unpaired bases of a secondary structure build crossing base-pairs by loop-loop interactions (the "kiss") and form a stable tertiary structure motif. Although these motifs have been known for over fifteen years, our understanding of kissing hairpins is still small. Especially viral genomes have been investigated for kissing hairpins, but also bacterial and eukaryotic ones. Researchers showed that kissing hairpins have important duties in a wide variety of RNA mediated processes. For example, they contribute extensively in stabilizing the structure and also play a role in viral plasmid DNA replication [14] or RNA synthesis [63]. Li et al. investigated in 2006 the mechanical unfolding of a minimal kissing complex [48]. They discovered that the loop-loop interaction is exceptionally stable.

5.1.2 Folding pseudoknots

Structures with pseudoknots are much more difficult to predict than nested structures. Even under energy models much simpler than what we use in practice, prediction of the optimal pseudo-knotted structure has been shown to be NP-hard [54, 1]. This has generated considerable interest in algorithms that solve the problem in polynomial time for restricted topologies of pseudoknots – see the review by Condon and Jabbari [17].

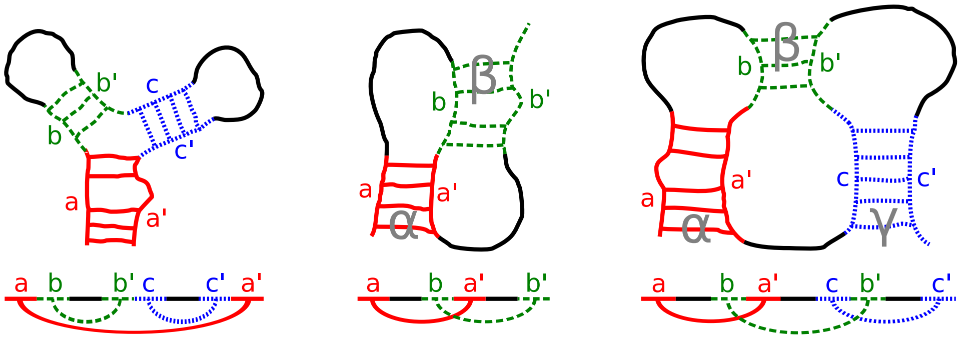


Figure 5.1: Schematic representation of a nested structure (the Y shape), a simple pseudoknot, and a kissing hairpin motif. The bottom line shows the arrangement of helix parts mapped to the primary sequence, with arbitrary sequence in between.

Many of these results are mainly of theoretical interest and have not led to practical tools. In an investigation of pseudoknot topologies [77], Rødland argues that the full topological complexity of pseudoknots is probably not needed in practical applications. For reasons of space, in the sequel we focus on those approaches which have resulted in realistic programs.

Pseudoknot folding using the established energy model was pioneered by Rivas and Eddy [76]. They presented an $O(n^6)$ time, $O(n^4)$ space algorithm for a fairly general class of pseudoknots, which we will refer to with the name PKNOTSSE. The high effort allows to fold only rather short sequences, and hence, the generality of the algorithm cannot really be exploited. A pragmatic approach was chosen by Reeder and Giegerich with the program PKNOTSRG [71]. They restricted the analysis to the class of *canonical simple recursive* pseudoknots, achieving $O(n^4)$ time, $O(n^2)$ space, and leading to a program widely used [1] today. The program HOTKNOTS [74] uses a heuristics to assemble pseudoknots from low-energy helices.

Quite recently, a new algorithm has been published in [15], but at the point of this writing, an implementation was not yet available. Our new approach presented here is an extension of the ideas used with PKNOTSRG, which we will review in necessary detail in Section 5.2.1.

5.1.3 Typology of structures

Notation

Dynamic programming over sequences leads to a decomposition of the given sequence into sub-words, typically in all possible ways. Let us recall, that we address sub-words as their pair of boundaries. For example, sub-word $(0, n)$ is S and sub-word $(2, 4)$ is

[1] Counting over 200 downloads and over 4,000 submissions per year according to http://bibiserv.techfak.uni-bielefeld.de/statistics/

${}_2s_3s_4$.

We write $s = xyz$ to indicate that s is split into sub-words x, y, z. The notation $s = {}_ix_ky_lz_j$ indicates, more concretely, that s is itself a sub-word of the overall input sequence S with boundaries i and j, and k, l denote the sub-word boundaries between x, y, z. If all boundaries are independent, a dynamic programming algorithm investigating all possible decompositions of this type has at least $O(n^4)$ steps, iterating over all $0 \leq i \leq k \leq l \leq j \leq n$.

Nested structures, simple pseudoknots, and kissing hairpins

We use the notation axa' to indicate that sub-word a' is a reverse complement (under RNA rules) of a, and hence the two can form a helix. Using these conventions, Figure 5.1 sketches three types of RNA structures, together with their associated sequence decomposition. The first is a nested structure, the so-called Y-shape, the second a simple pseudoknot (sometimes called H-type), and the third is a kissing hairpin structure, which is our specific concern here. We shall reserve the word “pseudoknot” for simple pseudoknots here, to distinguish them from kissing hairpins. When we allude to pseudoknots with a more complex topology than these two classes, we shall explicitly say so.

To evaluate the folding energy of a kissing hairpin motif on sub-word s, we need to split $s = aubva'wcxb'yc'$. The sub-words named u, v, w, x, y can attain arbitrary (sub-)structures, so kissing hairpins (as well as pseudoknots) may be embedded within each other.

5.2 Three strategies for kissing hairpin prediction

5.2.1 The combined power of canonization rules and non-ambiguous dynamic programming

Canonization

The algorithm of pknotsRG reduces computational complexity by imposing three canonization rules on the pseudoknots it considers:

Rule 1: In a helix $s = aua'$, a and a' are perfect helices.
Rule 2: In a helix $s = aua'$, a and a' extend towards each other maximally according to the rules of base-pairing, except the following case:
Rule 3: With crossing helices as in $aubva'wb'$, Rule 2 might imply a negative length of v. We set $v = \varepsilon$ and both helices meet at an arbitrary position.

Note that these rules are imposed on pseudoknots only, the search space of nested structures remains untouched. The beneficial effect of these rules is that maximal helices of form ${}_iaza'_j$ can be precomputed, and a canonical split into a pseudoknot of form $s = aubva'wb'$ is uniquely characterized by four moving boundaries only, more precisely as $s = {}_iau_kbva'_lwb'_j$. This is the key to achieve $O(n^4)$ time, $O(n^2)$ space efficiency. For details, we refer to [71]. There, it is shown that while an optimal, pseudoknotted structure P may not satisfy the canonicity constraints, there is a near-optimal pseudoknot P_{can} which does. However, minimum free energy folding might deliver an unknotted structure U with free energy such that $E(P) \leq E(U) \leq E(P_{can})$. U will be returned without a hint to P_{can}, and hence to the potential existence of P. At this point, computing with canonical pseudoknots seems but another heuristic approach.

Semantic non-ambiguity

A dynamic programming algorithm is called *semantically ambiguous* [28, 30], if it examines an object of interest in its search space more than once (cf. Section 2.1.3). This typically leads to exponential explosion of redundant solution candidates. For finding a single, optimal solution in a dynamic programming algorithm, such redundancy does not matter, but it renders the algorithm useless for producing near-optimals. The pknotsRG program is implemented in a semantically non-ambiguous way, according to the inclusion of pseudoknots.

Combining canonicity with a non-ambiguous algorithm allows the program to return sub-optimals. In particular, we can ask for the best canonical pseudoknot from the near-optimal search space, even when the minimum free energy structure comes out unknotted. The best canonical pseudoknot P_{can} may be checked for potential extension to a non-canonical structure P of even lower energy. In this sense, the heuristic constraint of canonization appears tolerable. Our algorithms presented here adhere to the same idea. All considered structures are canonical, and there will be only one situation where a structure is considered twice.

5.2.2 Decomposition alternatives of the kissing hairpin motif

An elementary decomposition of a kissing hairpin leads to three helices ($a-a', b-b', c-c'$) with intervening sequences u, v, w, x, y, folded in arbitrary ways, with the overall arrangement $aubva'wcxb'yc'$. See Figure 5.2 for an illustration. Such a decomposition, in full generality, leads to 12 moving boundaries, and makes us resort to canonization. Rule 2 of our canonization constraints eliminates six moving boundaries – the inner endpoints of three helices, which are now fixed by the helix maximality rule. The remaining boundaries are the outer endpoints of the three helices. Iterating over these six boundaries would lead to an $O(n^6)$ time, $O(n^2)$ space strategy. We implemented this slow but exhaustive variant as strategy “D” for evaluation purposes only, but our goal is to do better than this.

Our key idea is the view of the kissing hairpin motif as an overlay of two simple pseudoknots (Figure 5.2). Given that we already know how to compute optimal simple pseudoknots for the overlapping sub-words $aubva'zb'$ and $btcxb'yc'$, can we find their optimal overlay such that $z = wcx$ and $t = va'w$, thus defining the overall optimal decomposition into $aubva'wcxb'yc'$? Can we find its optimal energy as the sum from its two constituents?

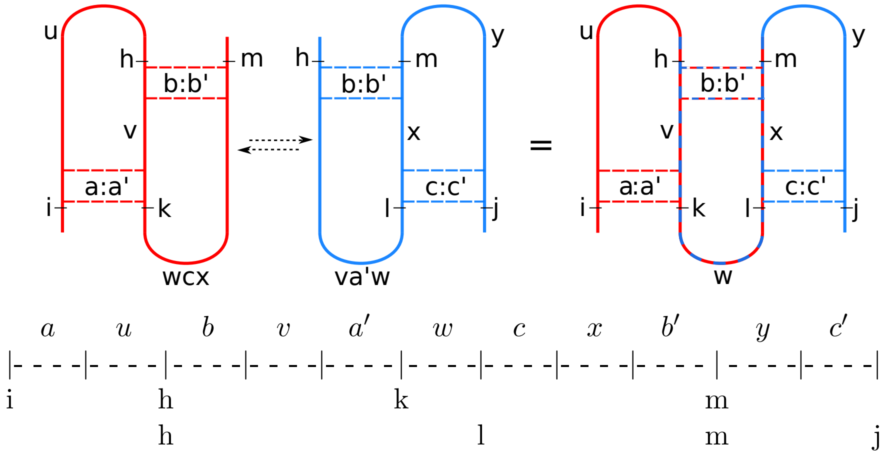


Figure 5.2: The composition of two pseudoknots leading to a kissing hairpin motif with the overlay of parts of the sequence and the moving boundaries i, h, k, l, m, and j on top. The linear form of the sequence below shows 12 moving boundaries (vertical lines). With the canonization rules, only six boundaries (labeled lines) remain.

Simple as it seems, there is a problem. First, if $w = \varepsilon$, the optimal choice of a' (with respect to a and b') may conflict with the optimal choice of c (with respect to b and c'). Moreover, in the overlay, the energy contribution of the middle helix ($b - b'$) and the structure for v, w, and x embedded within both pseudoknots are accounted for twice, and must be subtracted from the energy sum of both parts. This violates the monotonicity requirement for dynamic programming known as Bellman's Principle (cf. Section 2.1.1): for the overlay, the energy function is non-monotonic, and as a consequence, an optimal kissing hairpin motif may arise as an overlay of sub-optimal pseudoknots.

We will present three, increasingly complex strategies A, B, and C, plus an evaluation strategy D, such that their search spaces are properly included in the form $Searchspace_A \subseteq Searchspace_B \subseteq Searchspace_C \subseteq Searchspace_D \subset Searchspace_{KH}$. This relation will allow us to evaluate whether the expense for a more general strategy pays off in practice, but we will not be able to relate our results to an evaluation of the complete search space $Searchspace_{KH}$ of all (non-canonical) structures.

5.2.3 Strategy A – an O(n^4) time, O(n^2) space algorithm

Strategy A (see Figure 5.3) makes the optimistic assumption that at least one of the pseudoknots is the optimal structure for its underlying sub-word. This fixed, we choose the rest of the motif in the best possible way.

(1) For all sub-words p, find the optimal pseudoknot such that $p = aubva'zb'$. Store results in a table of size $O(n^2)$.

(2) For all sub-words s, split in all ways $s = pt$ and look up the optimal decomposition $p = aubva'zb'$.

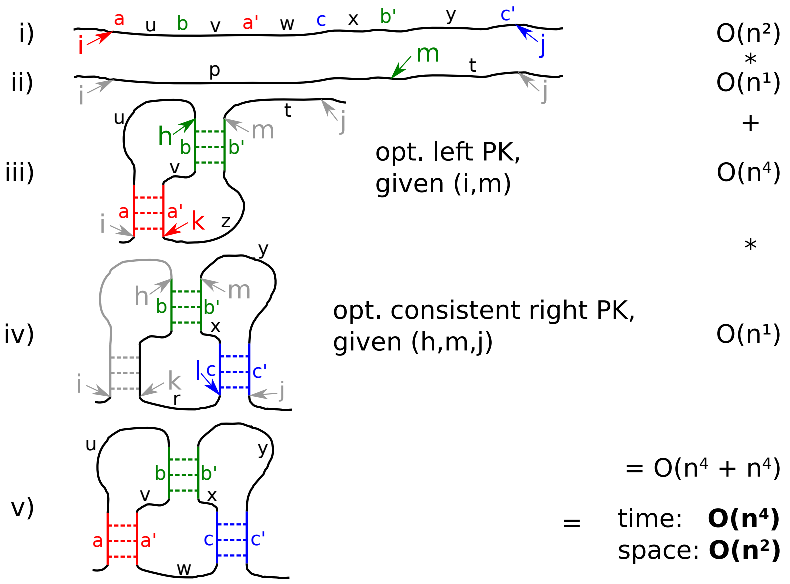


Figure 5.3: **Strategy A** makes the optimistic assumption that an optimal pseudoknot for the first half of the input sequence can be taken over to the kissing hairpin. The missing stem is adopted by an optimal, consistent pseudoknot for the second half.

(3) For all s of Step 2, use $s = auq$ and find the pseudoknot decomposition such that $q = brcxb'yc'$ and $r = va'w$, to complete the kissing hairpin decomposition $s = aubva'wcxb'yc'$. This pseudoknot must be chosen such that c lies strictly to the right of a', hence this is not, in general, the optimal pseudoknot over its underlying sub-word q. Record the decomposition of lowest free energy.

(4 - 6) Apply symmetric steps starting from an optimal choice for the right pseudoknot in the overlay.

(7) Choose lower energy value from (3) and (6); store it in a table of size $O(n^2)$.

The symmetry of (1-3) and (4-6) leads to the only case of ambiguity in our approach: If the two locally optimal pseudoknots make a perfect overlay as a kissing hairpin, this (optimal) structure will be found twice.

Efficiency: (1) takes $O(n^4)$ steps as with pknotsRG. (2) takes $O(n^3)$ steps, as the decomposition of p is already computed. (3) takes also $O(n^4)$, because it inherits $O(n^3)$ from Step 2 for all splits of s, which determine au and hence, the split auq. (Only) one extra factor of n arises from the split rc, which in turn determines the inner endpoints of helix $(c - c')$ due to the maximality rule, and hence implies the split yc'. (4-6) take $O(n^4)$ steps for symmetry reasons. (7) takes $O(n^2)$ steps. Postponing implementation details, we see that this yields an algorithm with $O(n^4)$ time, $O(n^2)$ space requirements. Note that Strategy A does some redundant work – the right pseudoknot determined in Step 3 has already been considered as a (generally sub-optimal) pseudoknot in Step 1.

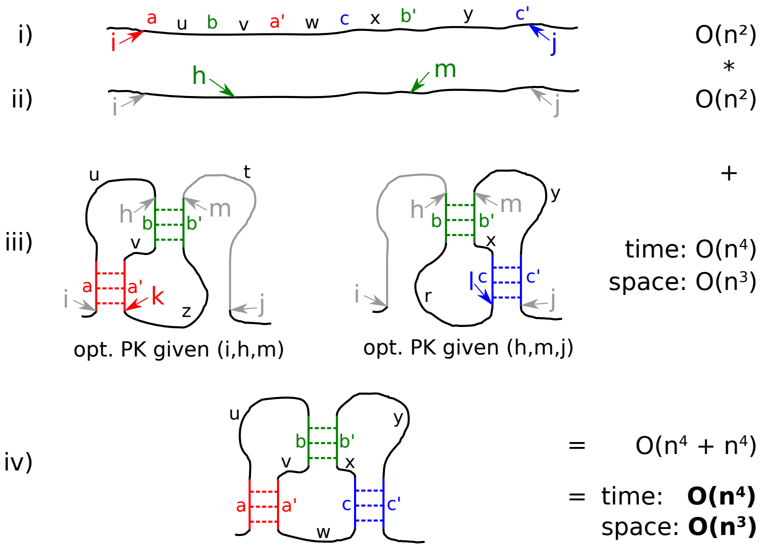


Figure 5.4: **Strategy B**: The overlay of two optimal pseudoknots must not necessarily yield an optimal kissing hairpin, since the overlay idea violates Bellman's principle of optimality. Thus the combination of two sub-optimal pseudoknots might result in an energetically better kissing hairpin. This knowledge is the basis for Strategy B. This modification leads to higher memory consumption to store certain sub-optimal pseudoknots.

5.2.4 Strategy B – an O(n^4) time, O(n^3) space algorithm

Strategy B (see Figure 5.4) avoids the redundant work of Strategy A, and also enlarges the search space. We spend extra space in Step 1 to store results about sub-optimal pseudoknots.

(1) For $p = aubva'zb'$, and for each choice of b therein, we record the optimal choice of a'. Conversely, for each choice of a', we store the optimal choice of b. This requires two tables of size $O(n^3)$.

(2) For the kissing hairpin motif, we first choose a, b, b', and c', which costs $O(n^4)$, and use the stored information to optimally determine the other bounds for a' and c by look-up with $O(1)$.

(3) Unfortunately, the stored information may suggest that with an optimal choice, a' and c would overlap (and w have negative length). We correct this by a heuristic decision – selecting an a' further to the left and a c further to the right. This decision will also be based on precomputed information in order to retain a run-time of $O(n^4)$.

(4) We minimize over all cases considered.

The overall efficiency is $O(n^4)$ time and $O(n^3)$ space. Note that the search space here is more general than with strategy A, as neither pseudoknot needs to be optimal with respect to its underlying sub-word. This generalization lies with Step 1. In Strategy A, only the optimal choice of b within p is considered for overlay, while here, all possible choices of b are tried.

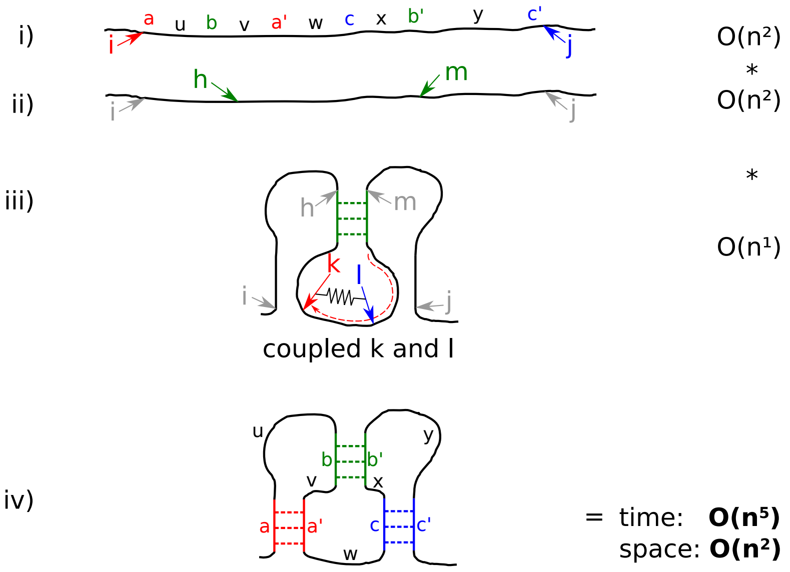


Figure 5.5: **Strategy C**: Since larger memory is often a harder problem than longer run-time, we alter Strategy B to trade memory for run-time. Strategy C avoids the extra storage required by Strategy B by re-computing the necessary information on demand. Coupling k and l reduces the run-time by one dimension.

5.2.5 Strategy C – an O(n^5) time, O(n^2) space algorithm

Strategy C (see Figure 5.5) avoids the extra storage required by Strategy B. The necessary information is re-computed on demand, after choosing a, b, b' and c'. This increases run-time, but also allows us to avoid the heuristic decision when a' and c would overlap. For each choice of a', we compute the best choice of c strictly to its right. This threatens to raise time complexity to $O(n^6)$, but with a clever arrangement of computations and an extra table of size $O(n)$, we can keep it at $O(n^5)$.

The optimal choice of l with respect to (h, j) as a pseudoknot is a heuristics with respect to (i, j) as a kissing hairpin (see Figure 5.6). It assumes that $va'w$ can fold optimally. For the kiss, however, v and w can only fold individually, as they are separated by a', which is the partner of a. Thus, l need not be optimal for (i, j) as a kissing hairpin.

5.3 Algorithms

5.3.1 Algorithmic subtleties

Annotated energies

When computing minimum free energies from pseudoknots, we will need to also record the internal boundaries of the given sub-word which achieved optimal energy. These

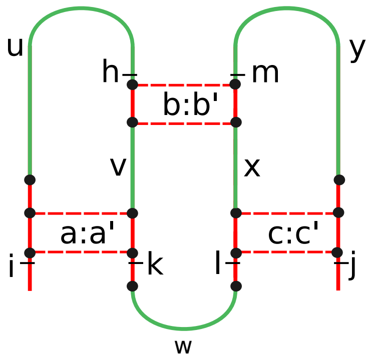


Figure 5.6: The graphic shows the mandatory bases (black dots) of a kissing hairpin and the indices i, h, k, l, m, and j determining the start and end points of the helices (black tics). Green regions u, v, w, x, and y can fold in an arbitrary way.

will be data of the form (e, σ, τ). When we minimize over these tuples, we do this with a lexicographic ordering. This is consistent with minimizing over energies alone. When two structures have the same energy, then the choice is arbitrary and remains unspecified.

Exact sub-word boundaries in the input decomposition

Sub-structures have certain minimal sizes. For example, we forbid lonely pairs, i. e. helices of length 1. Therefore, in ${}_ia_kza'_j$, we do not iterate k over $i \leq k \leq j$, but only over $i+2 \leq k \leq j-2$. This does not affect the asymptotics, but saves substantial time in practice. The minimal sub-word sizes used are two base-pairs for each helix, loop u and y have one unpaired base. Loop w has two single bases $(k+2 \leq l)$. The size of loop v and x is ≥ 0, because we want to keep the possibility of coaxially stacking of the helices. With that, we get a minimal sequence length of 16 bases to form a kissing hairpin (see Figure 5.6).

To be concrete in the following recurrences, we use the precise boundaries consistent with our implementation. But for understanding the essentials of the algorithms, the reader may choose to ignore them.

5.3.2 Pseudoknot-recurrence of pknotsRG – csrPK

Due to the canonization of PKNOTSRG, the calculation of a canonical simple recursive pseudoknot (csrPK) for a given sub-word needs two boundaries in addition to (i, j): h, the start position of the $b - b'$ helix, and k, the end position of the $a - a'$ helix. The recurrence of a csrPK for a sub-word (i, j) is:

$$\text{csrPK}\,(i,j) = \underset{\substack{i+3 \,\leq\, h \,\leq\, j-8 \\ h+4 \,\leq\, k \,\leq\, j-4}}{min} E_{\text{csrPK}}\left({}_iau_hbva'_krb'_j\right)$$

The energy function E_{csrPK} makes use of a precomputed DP-table (`stacklen`) to determine the inner endpoints of the helices in a unique, maximal and non-overlapping fashion. With these boundaries fixed, the energy value is the sum of stabilizing energies of both helices + energy contributions of the arbitrary folded regions u, v and w + contributions from bases which dangle onto the helices from inside the csrPK + penalties for explicitly unpaired bases in front of u and b', see Table 5.3 for details. For later use, we adapt E_{csrPK} to additionally store $\sigma = h$ and $\tau = k$, which can be retrieved by the functions boundary$_{\text{left}}$ and boundary$_{\text{right}}$.

5.3.3 Recurrences of Strategy A – csrKH$_{\text{A}}$

For Strategy A we make two strong assumptions. (1) Helices $a - a'$ and $b - b'$ of an optimal csrPK, starting at i and ending at m, can be adopted for the overall csrKH and thus determine the boundaries h and k. We can look up these values via the table csrPK. (2) The remaining boundary l, the starting point for the $c - c'$ helix, can be determined by using the energy of a second csrPK as an objective function. This second csrPK must start at h, end at j and have its end position of the left helix $b : b'$ at m, thus overlaying a part of the first csrPK:

$$\begin{aligned}
\text{left}\,(i,j) = &\min_{i+13\ \le\ m\ \le\ j-3} E_{\text{csrKH}}\left({}_i a u_h b v a'_k w_l c x b'_m y c'_j\right), \quad \text{where} \\
&h = \text{boundary}_{\text{left}}\,(\text{csrPK}\,(i,m)), \\
&k = \text{boundary}_{\text{right}}\,(\text{csrPK}\,(i,m)), \\
&l = \text{boundary}_{\text{left}}\left(\min_{k+2\ \le\ d\ \le\ m-4} E_{\text{csrPK}}\left({}_h b v a' w_d c x b'_m y c'_j\right)\right)
\end{aligned}$$

A csrKH may alternatively arise from the opposite direction, i. e. an optimal csrPK on its right half overlaying a sub-optimal csrPK at its left:

$$\begin{aligned}
\text{right}\,(i,j) = &\min_{i+3\ \le\ h\ \le\ j-13} E_{\text{csrKH}}\left({}_i a u_h b v a'_k w_l c x b'_m y c'_j\right), \quad \text{where} \\
&l = \text{boundary}_{\text{left}}\,(\text{csrPK}\,(h,j)), \\
&m = \text{boundary}_{\text{right}}\,(\text{csrPK}\,(h,j)), \\
&k = \text{boundary}_{\text{right}}\left(\min_{h+4\ \le\ d\ \le\ l-2} E_{\text{csrPK}}\left({}_i a u_h b v a'_d w c x b'_m\right)\right)
\end{aligned}$$

The optimal csrKH with Strategy A is:

$$\text{csrKH}_A\,(i,j) = \min\,(\text{left}\,(i,j)\,, \text{right}\,(i,j))$$

5.3.4 Recurrences of Strategy B – csrKH$_{\text{B}}$

Since Strategy B has to store the optimal choice of a' for every given b for csrPKs on the left side and the optimal b for every given a' for csrPKs on the right side of the csrKH, we have to replace the function csrPK with lpk and rpk. A csrPK for a sub-word (i,j) can now be determined by minimizing over $lpk\,(i,h,j)$ and $rpk\,(i,k,j)$:

$$lpk\,(i,h,j) = \min_{h+4\ \le\ k\ \le\ j-4} E_{\text{csrPK}}\left({}_iau_hbva'_krb'_j\right)$$
$$rpk\,(i,k,j) = \min_{i+3\ \le\ h\ \le\ k-4} E_{\text{csrPK}}\left({}_iau_hbva'_krb'_j\right)$$

An overlay of csrPKs from lpk and rkp might overlap in region w of the csrKH, when building it. We can overcome this obstacle in a heuristic way by introducing an artificial border ξ:

$$lpk_{\text{heuristic}}\,(i,h,j) = \min_{h+4\ \le\ k\ \le\ \xi} E_{\text{csrPK}}\left({}_iau_hbva'_krb'_j\right)$$
$$rpk_{\text{heuristic}}\,(i,k,j) = \min_{\xi\ \le\ h\ \le\ k-4} E_{\text{csrPK}}\left({}_iau_hbva'_krb'_j\right)$$

Thus we can construct a csrKH with Strategy B by first iterating over the outer endpoints of helix $b-b'$, namely m and h. Second, we choose the energetically optimal combination of k and l by overlaying all csrPKs from $lpk\,(i,h,m)$ and $rpk\,(h,m,j)$, as well as their heuristic counterparts $lpk_{\text{heuristic}}\,(i,h,m)$ and $rpk_{\text{heuristic}}\,(h,m,j)$ to guarantee at least one feasible overlay:

$$\text{csrKH}_B(i,j) = \min_{\substack{i+13\ \le\ m\ \le\ j-3\\ i+3\ \le\ h\ \le\ m-10}} E_{\text{csrKH}}\left({}_iau_hbva'_kw_lcxb'_myc'_j\right), \quad \text{where}$$
$$k \in \text{boundary}_{\text{right}}\left\{lpk\,(i,h,m)\,,\ lpk_{\text{heuristic}}\,(i,h,m)\right\}$$
$$l \in \text{boundary}_{\text{left}}\left\{rpk\,(h,m,j)\,,\ rpk_{\text{heuristic}}\,(h,m,j)\right\}$$

5.3.5 Recurrences of Strategy C – csrKH$_C$

We start with Strategy C identical to Strategy B, by iterating over m and h. But instead of retrieving k and l from precomputed csrPK tables, we now also iterate k to determine a' and look up the optimal choice for l depending on k in a one dimensional table rpk:

$$\text{csrKH}_C(i,j) = \min_{\substack{i+13\ \le\ m\ \le\ j-3\\ i+3\ \le\ h\ \le\ m-10\\ h+4\ \le\ k\ \le\ m-6\\ l\ =\ \text{boundary}_{\text{left}}(rpk(k))}} E_{\text{csrKH}}\left({}_iau_hbva'_kw_lcxb'_myc'_j\right)$$

When iterating over k, we go from right to left. Thus we have a growing sub-word (k,m). While shifting k one position to the left, the function $rpk(k)$ also determines the optimal csrPK that begins at h, ends at j, has its b' at m and its c somewhere in the sub-word (k,m). Since we temporarily store the results for $rpk(k)$, it can be calculated in $O(1)$ time. We just compare the existing result for the one letter shorter sub-word $rpk(k+1)$ with one new csrPK, whose boundaries are at $h, k+2, m, j$:

$$rpk(k) = \min\left(E_{\text{csrPK}}\left({}_hbva'w_{k+2}cxb'_myc'_j\right), rpk(k+1)\right)$$

5.3.6 Recurrences of Strategy D – csrKH$_\mathrm{D}$

Just for completeness, we give recurrences for the evaluation Strategy D, where all six boundaries have to be iterated independently:

$$\mathrm{csrKH}_D(i,j) = \mathop{\mathit{min}}_{\substack{i+13 \;\leq\; m \;\leq\; j-3 \\ i+3 \;\leq\; h \;\leq\; m-10 \\ h+4 \;\leq\; k \;\leq\; m-6 \\ k+2 \;\leq\; l \;\leq\; m-4}} E_{\mathrm{csrKH}}\left({}_i a u_h b v a'_k w_l c x b'_m y c'_j\right)$$

5.4 Implementation via Bellman's GAP

Alike pKnotsRG, pKiss is implemented with ADP. This makes it easy to add and combine different types of analysis. Our main goal is computation of thermodynamic optimal and sub-optimal structures, plus integration of possible co-axial stacking of pseudoknot stems, which is fairly similar to dangling bases. Thus, we incorporate pseudoknot dependent rules into $\mathcal{G}_{\mathrm{MicroState}}$ (see Figure 3.3). In the following, we list technical details, how we integrated context sensitive non-terminals into the principally context free framework Bellman's GAP, focusing on MFE prediction. However, our implementation (`bibiserv.cebitec.uni-bielefeld.de/pkiss`) provides a whole set of further functionalities, e. g. abstract and probabilistic shape analysis, enforced and local pseudoknot folding, energetic evaluation of given pseudoknotted structures and comparative structure prediction.

5.4.1 Signature

We treat a pseudoknot (may it be a csrPK or a csrKH) like a closed sub-structure, which might be target of dangling bases. Different from "normal" closed sub-structures, where the partners of the base-pair onto which neighboring bases might dangle are always the outermost symbols of the according sub-string, one partner of a pseudoknot stem is always "hidden" somewhere in the middle of the sub-string. In order to calculate the correct energy contribution, it is necessary to record these positions (σ and τ), together with the current energy value of the pseudoknot. Thus, for pseudoknot dependent algebra functions, we need a second *sort* $\mathcal{K}$ and a second objective function h_{knot}, which chooses from a list of $\mathcal{K}$. Furthermore, these information must sometimes be fed into other algebra functions, e. g. emptymid, with the help of additional parameters.

Table 5.1 lists all algebra functions of Signature Σ_{knot}. Since we need two sorts, we give the result type of each algebra function as the right hand side of the arrows.

5.4.2 Grammar

Grammar $\mathcal{G}_{\mathrm{pKiss}}$ of Figure 5.7 embeds pseudoknots into the MicroState Grammar. The idea is, that a pseudoknot is another closed sub-structure, onto which neighboring bases

Table 5.1: Signature Σ_{knot} for RNA problems. Result type of an algebra functions is either sort $\mathcal{S}$ or sort $\mathcal{K}$. $\mathcal{A}$ is a symbol from the alphabet, i. e. a single character from the input sequence. $\mathcal{A}^*$ stands for a non-empty sub-sequence from the input and $\mathcal{A}^0$ shall denote reading the position within the input, but not consuming any symbol. Additional *int* parameters are used to identify the correct pairing partners of base-pairs, onto which other bases might dangle.

sadd$(\mathcal{A}, \mathcal{S}) \to \mathcal{S}$
cadd$(\mathcal{S}, \mathcal{S}) \to \mathcal{S}$
nil$(\mathcal{A}^0) \to \mathcal{S}$
drem$(\mathcal{A}^0, \mathcal{S}, \mathcal{A}^0) \to \mathcal{S}$
edl$(\mathcal{A}, \mathcal{S}, \mathcal{A}^0) \to \mathcal{S}$
edr$(\mathcal{A}^0, \mathcal{S}, \mathcal{A}) \to \mathcal{S}$
edlr$(\mathcal{A}, \mathcal{S}, \mathcal{A}) \to \mathcal{S}$
sr$(\mathcal{A}, \mathcal{S}, \mathcal{A}) \to \mathcal{S}$
hl$(\mathcal{A}, \mathcal{A}^*, \mathcal{A}) \to \mathcal{S}$
bl$(\mathcal{A}, \mathcal{A}^*, \mathcal{S}, \mathcal{A}) \to \mathcal{S}$
br$(\mathcal{A}, \mathcal{S}, \mathcal{A}^*, \mathcal{A}) \to \mathcal{S}$
pknot$(int)(\mathcal{A}, \mathcal{S}, \mathcal{A}, \mathcal{S}, \mathcal{A}, \mathcal{S}, \mathcal{A}) \to \mathcal{K}$
pkiss$(int)(\mathcal{A}, \mathcal{S}, \mathcal{A}, \mathcal{S}, \mathcal{A}, \mathcal{S}, \mathcal{A}, \mathcal{S}, \mathcal{A}, \mathcal{S}, \mathcal{A}) \to \mathcal{K}$

il$(\mathcal{A}, \mathcal{A}^*, \mathcal{S}, \mathcal{A}^*, \mathcal{A}) \to \mathcal{S}$
ml$(\mathcal{A}, \mathcal{S}, \mathcal{A}) \to \mathcal{S}$
mldl$(\mathcal{A}, \mathcal{A}, \mathcal{S}, \mathcal{A}) \to \mathcal{S}$
mldr$(\mathcal{A}, \mathcal{S}, \mathcal{A}, \mathcal{A}) \to \mathcal{S}$
mldlr$(\mathcal{A}, \mathcal{A}, \mathcal{S}, \mathcal{A}, \mathcal{A}) \to \mathcal{S}$
addss$(\mathcal{S}, \mathcal{A}^*) \to \mathcal{S}$
incl$(\mathcal{S}) \to \mathcal{S}$
pkml$(\mathcal{S}) \to \mathcal{S}$
sadd_pk$(\mathcal{A}, \mathcal{S}) \to \mathcal{S}$
midregion$(\mathcal{S}) \to \mathcal{S}$
frd$(int)(\mathcal{S}, \mathcal{A}) \to \mathcal{S}$

emptymid$(int, int)(\mathcal{A}^*) \to \mathcal{S}$
midbase$(int, int)(\mathcal{A}^*) \to \mathcal{S}$
middlro$(int, int)(\mathcal{A}^*) \to \mathcal{S}$
middl$(int)(\mathcal{A}, \mathcal{S}) \to \mathcal{S}$
middr$(int)(\mathcal{S}, \mathcal{A}) \to \mathcal{S}$
middlr$(int, int)(\mathcal{A}, \mathcal{S}, \mathcal{A}) \to \mathcal{S}$
bkd$(int)(\mathcal{A}, \mathcal{S}) \to \mathcal{S}$
pk$(\mathcal{K}) \to \mathcal{S}$
kndl$(\mathcal{S}, \mathcal{K}) \to \mathcal{S}$
kndr$(\mathcal{K}, \mathcal{S}) \to \mathcal{S}$
kndlr$(\mathcal{S}, \mathcal{K}, \mathcal{S}) \to \mathcal{S}$
$\mathbf{h}([\mathcal{S}]) \to [\mathcal{S}]$
$\mathbf{h}_{knot}([\mathcal{K}]) \to [\mathcal{K}]$

might dangle: *dangleknot* non-terminal with dangling alternatives pk, kndl, kndr and kndlr. Besides of the fact that non-terminals to create pseudoknots (prefixed with "pknot" and "pkiss") are somehow outside of the ADP concept, they are normal citizen of the grammar, i. e. they can call other non-terminals to form sub-structures for their loop regions – even recursively themselves – or be called by others, e. g. multiloops or stretches of adjacent closed sub-structures (cadd). Two of the three (in case of a csrKH) stems of a pseudoknot might stack co-axial onto each other in 3D. The stabilizing energy contribution is accounted for as a further base-pair stack in the algebra functions emptymid or midbase, should only a single base be bulged out.

According to the answer sorts ($\mathcal{S}$ or $\mathcal{K}$) of the alternative productions for the non-terminals, either objective function h (normal black arrows) or h_{knot} (red annotated arrows) must be applied. Non-terminal *knot* makes use of a special syntactic filter – similar to $\neq$ LP | LP – to choose the pseudoknot prediction strategy according to a command line parameter. The filter ignore is for filling the non-terminal DP table, but no candidates will ever be part of the search space. It seems rather useless to compute things that cannot contribute to the overall result, but we need these DP tables for efficient computation of csrKHs. Some non-terminals are annotated with gray arguments. These arguments are either used to push information into algebra functions, e. g. from $middle(\sigma, \tau)$ to midbase(σ, τ), or to control the computation of the non-terminal itself, e. g. $pknot\_free\_h(h, k_{init})$. From [79]

A non-terminal symbol can be defined with arguments (i. e. a parameter-

> ized non- terminal). The arguments, or expressions including the arguments, can be used on the right hand side as extra arguments of a function symbol, a filter function or another parametrized non-terminal call. A parametrized non-terminal cannot be tabulated, because for every combination of parameter values a separate table would be needed.

The children of the algebra function pknot and pkiss are annotated with start- and end-boundaries of the sub-string they shall consume. This is necessary, since some parts of the input string have to be consumed at the same time in different terminals / non-terminals, which is a violation of the tree structure of "normal" context free grammars. Details about the indices will be given in Section 9.2 of the appendix as source code segments of the actual Bellman's GAP code. Variables $|\alpha|$, $|\beta|$ and $|\gamma|$ hold the length of the up to three stems of pseudoknots.

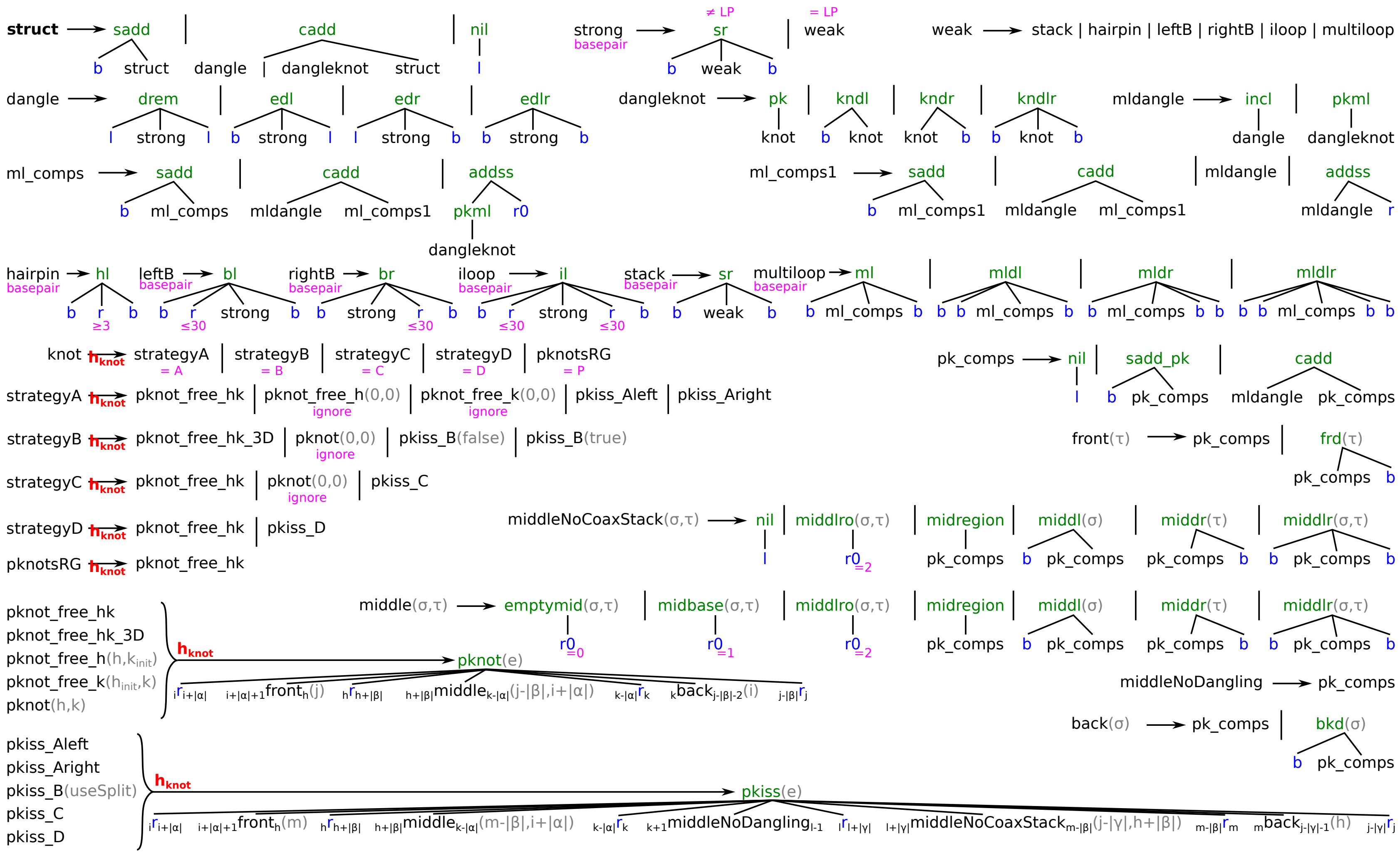

struct → sadd(b, struct) | cadd(dangle, dangleknot, struct) | nil(l)
strong basepair → sr(b, weak, b) ≠ LP | weak = LP
weak → stack | hairpin | leftB | rightB | iloop | multiloop
dangle → drem(l, strong, l) | edl(b, strong, l) | edr(l, strong, b) | edlr(b, strong, b)
dangleknot → pk(knot) | kndl(b, knot) | kndr(knot, b) | kndlr(b, knot, b)
mldangle → incl(dangle) | pkml(dangleknot)
ml_comps → sadd(b, ml_comps) | cadd(mldangle, ml_comps1) | addss(pkml(dangleknot), r0)
ml_comps1 → sadd(b, ml_comps1) | cadd(mldangle, ml_comps1) | mldangle | addss(mldangle, r)
hairpin basepair → hl(b, r ≥3, b)
leftB basepair → bl(b, r ≤30, strong, b)
rightB basepair → br(b, strong, r ≤30, b)
iloop basepair → il(b, r ≤30, strong, r ≤30, b)
stack basepair → sr(b, weak, b)
multiloop basepair → ml(b, ml_comps, b) | mldl(b, b, ml_comps, b) | mldr(b, ml_comps, b, b) | mldlr(b, b, ml_comps, b, b)
knot h_knot → strategyA = A | strategyB = B | strategyC = C | strategyD = D | pknotsRG = P
strategyA h_knot → pknot_free_hk | pknot_free_h(0,0) ignore | pknot_free_k(0,0) ignore | pkiss_Aleft | pkiss_Aright
strategyB h_knot → pknot_free_hk_3D | pknot(0,0) ignore | pkiss_B(false) | pkiss_B(true)
strategyC h_knot → pknot_free_hk | pknot(0,0) ignore | pkiss_C
strategyD h_knot → pknot_free_hk | pkiss_D
pknotsRG h_knot → pknot_free_hk
pk_comps → nil(l) | sadd_pk(b, pk_comps) | cadd(mldangle, pk_comps)
front(τ) → pk_comps | frd(τ)(pk_comps, b)
middleNoCoaxStack(σ,τ) → nil(l) | middlro(σ,τ)(r0 =2) | midregion(pk_comps) | middl(σ)(b, pk_comps) | middr(τ)(pk_comps, b) | middlr(σ,τ)(b, pk_comps, b)
middle(σ,τ) → emptymid(σ,τ)(r0 =0) | midbase(σ,τ)(r0 =1) | middlro(σ,τ)(r0 =2) | midregion(pk_comps) | middl(σ)(b, pk_comps) | middr(τ)(pk_comps, b) | middlr(σ,τ)(b, pk_comps, b)
middleNoDangling → pk_comps
back(σ) → pk_comps | bkd(σ)(b, pk_comps)
pknot_free_hk, pknot_free_hk_3D, pknot_free_h(h,k_init), pknot_free_k(h_init,k), pknot(h,k) h_knot → pknot(e)
$_i r_{i+|\alpha|}$ $_{i+|\alpha|+1}$front$_h$(j) $_h r_{h+|\beta|}$ $_{h+|\beta|}$middle$_{k-|\alpha|}$(j-|β|,i+|α|) $_{k-|\alpha|} r_k$ $_k$back$_{j-|\beta|-2}$(i) $_{j-|\beta|} r_j$
pkiss_Aleft, pkiss_Aright, pkiss_B(useSplit), pkiss_C, pkiss_D h_knot → pkiss(e)
$_i r_{i+|\alpha|}$ $_{i+|\alpha|+1}$front$_h$(m) $_h r_{h+|\beta|}$ $_{h+|\beta|}$middle$_{k-|\alpha|}$(m-|β|,i+|α|) $_{k-|\alpha|} r_k$ $_{k+1}$middleNoDangling$_{l-1}$ $_l r_{l+|\gamma|}$ $_{l+|\gamma|}$middleNoCoaxStack$_{m-|\beta|}$(j-|γ|,h+|β|) $_{m-|\beta|} r_m$ $_m$back$_{j-|\gamma|-1}$(h) $_{j-|\gamma|} r_j$

Figure 5.7: Grammar $\mathcal{G}_{\text{pKiss}}$. For special annotations consult main text. Non-terminals for computing the pseudoknots are described in Section 9.2 of the appendix.

Table 5.2: Evaluation algebra $\mathcal{I}_{db}^{knot}$ to represent a potentially pseudoknotted secondary structures with a Vienna-Dot-Bracket like string. A notation like $\texttt{[[[}_{\lvert a\rvert}$ means as many $\texttt{[}$-symbols as the length of a.

algebra function	$\mathcal{I}_{db}^{knot} \rightarrowtriangle \mathcal{I}_{db}$
pkml(x)	x
sadd_pk(a, x)	$\texttt{.}\ x$
midregion(x)	x
frd$(\tau)(x, b)$	$x\ \texttt{.}$
emptymid$(\sigma, \tau)(r)$	ε
midbase$(\sigma, \tau)(r)$	$\texttt{.}$
middlro$(\sigma, \tau)(r)$	$\texttt{..}$
middl$(\sigma)(a, x)$	$\texttt{.}\ x$
middr$(\tau)(x, b)$	$x\ \texttt{.}$
middlr$(\sigma, \tau)(a, x, b)$	$\texttt{.}\ x\ \texttt{.}$
bkd$(\sigma)(a, x)$	$\texttt{.}\ x$
pk(x)	x
kndl(a, x)	$\texttt{.}\ x$
kndr(x, b)	$x\ \texttt{.}$
kndlr(a, x, b)	$\texttt{.}\ x\ \texttt{.}$
pknot$(e)(a, u, b, v, a', r, b')$	$\texttt{[[[}_{\lvert a\rvert}\ \texttt{.}\ u\ \texttt{\{\{\{}_{\lvert b\rvert}\ v\ \texttt{]]]}_{\lvert a'\rvert}\ r\ \texttt{..}\ \texttt{\}\}\}}_{\lvert b'\rvert}$
pkiss$(e)(a, u, b, v, a', w, c, x, b', y, c')$	$\texttt{[[[}_{\lvert a\rvert}\ \texttt{.}\ u\ \texttt{\{\{\{}_{\lvert b\rvert}\ v\ \texttt{]]]}_{\lvert a'\rvert}\ \texttt{.}\ w\ \texttt{.}\ \texttt{<<<}_{\lvert c\rvert}\ x\ \texttt{\}\}\}}_{\lvert b'\rvert}\ y\ \texttt{.}\ \texttt{>>>}_{\lvert c'\rvert}$
$\mathcal{S}$	string
$\mathcal{K}$	string
$\mathbf{h}$	id
$\mathbf{h}_{knot}$	id

5.4.3 Algebras

Dot-Bracket algebra

We cherish to represent our structure predictions as Vienna-Dot-Bracket strings. In order to symbolize crossing base-pairs correctly, we need to extend the alphabet with further pairing characters, i. e. [and], { and } and finally < and >, for α, β and γ stems, respectively.

Evaluation algebra $\mathcal{I}_{db}^{knot}$ inherits all 18 algebra functions for "nested" structures from $\mathcal{I}_{db}$ (Table 3.2). Newly introduced, pseudoknot specific algebra functions are given in Table 5.2. Both sorts and both objective functions are identical.

MFE algebra

The evaluation of the free energy of a pseudoknotted structure $\mathcal{I}_{mfe}^{knot}$ (Table 5.3) basically follows the typical scheme of $\mathcal{I}_{mfe}$ (Table 3.3). Dangling adjacent unpaired bases onto stems of the pseudoknot forces us to provide their inner base-pair partners. Thus, sort $\mathcal{K}$ is a triple of the three components 1) free energy, 2) right boundary of the inner base-pair partner of left stem σ and 3) left boundary of the inner base-pair partner of right stem τ. The objective function $\mathbf{h}_{knot}$ minimizes over the first component of $\mathcal{K}$. Variable e of pknot and pkiss hold the combined free energy of all pseudoknot stems (it is the variable "stackenergies" from the source code). We see a lot of dangling contribution in $\mathcal{I}_{mfe}^{knot}$, since all the mandatory unpaired bases can dangle onto the pseudoknot stems. Consult Figure 5.6 to get an overview of those bases.

The prediction of RNA secondary structures including kissing hairpin motifs for an RNA sequence x is then the result of the Bellman's GAP instance

$$\mathcal{G}_{\text{pKiss}}(\mathcal{I}_{mfe}^{knot} * \mathcal{I}_{db}^{knot}, x).$$

5.5 Evaluation

5.5.1 A piece of anecdotal evidence

The RNA polymerase gene (gene 1) of the human coronavirus 229E is a good example for the usefulness of improved secondary structure prediction tools. Analyzing the genome of the human coronavirus, Herold and Siddell [38] guessed, that a "slippery site" together with an H-type pseudoknot acts as a frameshift inducing structure. Extensive mutational analyses showed that a kissing hairpin is required for high frequency frameshifts. Their work implied computer-assisted modeling, but prior prediction tools could not detect kissing hairpin motifs. pKiss finds the proper kissing hairpin.

5.5.2 Test set "knot"

For test set "knot" we compiled a list of 433 RNA sequences with verified pseudoknotted secondary structures. From this set, 33 sequences form kissing hairpin motifs. See

Table 5.3: Evaluation algebra $\mathcal{I}_{mfe}^{knot}$ to compute the free energy of a potentially pseudoknotted secondary structure.

algebra function	$\mathcal{I}_{mfe}^{knot} \mathrel{-\!\!\triangleright} \mathcal{I}_{mfe}$
pkml(x)	$x + \mathrm{E}_{\mathrm{init}_{pkml}}$
sadd_pk(a, x)	$x + \mathrm{E}_{\mathrm{npp}}$
midregion(x)	x
frd(τ)(x, b)	$x + \mathrm{E}_{\mathrm{dl}}({}_{i+1}b, b_{\beta^d}) + \mathrm{E}_{\mathrm{npp}}$
emptymid(σ, τ)(r)	$\mathrm{E}_{\mathrm{sr\_pk}}(({}_{i-1}r, r_{\sigma}), ({}_{i}r, r_{\tau-1}))$
midbase(σ, τ)(r)	$\mathrm{E}_{\mathrm{sr\_pk}}(({}_{i-1}r, r_{\sigma}), ({}_{i+1}r, r_{\tau-1})) + \mathrm{E}_{\mathrm{npp}}$
middlro(σ, τ)(r)	$\mathrm{E}_{\mathrm{dri}}({}_{\tau-1}r, r_{j+1}) + \mathrm{E}_{\mathrm{dli}}({}_{i-1}r, r_{\sigma+1}) + 2 \cdot \mathrm{E}_{\mathrm{npp}}$
middl(σ)(a, x)	$x + \mathrm{E}_{\mathrm{dli}}({}_{i-1}a, a_{\sigma+1}) + \mathrm{E}_{\mathrm{npp}}$
middr(τ)(x, b)	$x + \mathrm{E}_{\mathrm{dri}}({}_{\tau-1}b, b_{j+1}) + \mathrm{E}_{\mathrm{npp}}$
middlr(σ, τ)(a, x, b)	$x + \mathrm{E}_{\mathrm{dli}}({}_{i-1}a, a_{\sigma+1}) + \mathrm{E}_{\mathrm{dri}}({}_{\tau-1}b, b_{j+1}) + 2 \cdot \mathrm{E}_{\mathrm{npp}}$
bkd(σ)(a, x)	$x + \mathrm{E}_{\mathrm{dr}}({}_{\sigma}a, a_{j-1}) + \mathrm{E}_{\mathrm{npp}}$
pk((x, σ, τ))	x
kndl($a, (x, \sigma, \tau)$)	$x + \mathrm{E}_{\mathrm{dl}}({}_{i+1}a, a_{\tau}) + \mathrm{E}_{\mathrm{npp}}$
kndr($(x, \sigma, \tau), b$)	$x + \mathrm{E}_{\mathrm{dr}}({}_{\sigma}a, a_{j-1}) + \mathrm{E}_{\mathrm{npp}}$
kndlr($a, (x, \sigma, \tau), b$)	$x + \mathrm{E}_{\mathrm{dl}}({}_{i+1}a, a_{\tau}) + \mathrm{E}_{\mathrm{dr}}({}_{\sigma}b, b_{j-1}) + 2 \cdot \mathrm{E}_{\mathrm{npp}}$
pknot(e)(a, u, b, v, a', r, b')	$(e + u + v + r$ $+ \mathrm{E}_{\mathrm{init}_{pk}} + 3 \cdot \mathrm{E}_{\mathrm{npp}}$ $+ \mathrm{E}_{\mathrm{termau}}({}_{i}a, a'_{j}) + \mathrm{E}_{\mathrm{termau}}({}_{j-1}a, a'_{i+1})$ $+ \mathrm{E}_{\mathrm{termau}}({}_{i}b, b'_{j}) + \mathrm{E}_{\mathrm{termau}}({}_{j-1}b, b'_{i+1})$ $+ \mathrm{E}_{\mathrm{dli}}({}_{j-1}a, a'_{i+1})$ $+ \mathrm{E}_{\mathrm{dri}}({}_{j-1}b, b'_{i+1})$ $, {}_{i}b, a'_{j})$
pkiss(e)($a, u, b, v, a', w, c, x, b', y, c'$)	$(e + u + v + w + x + y$ $+ \mathrm{E}_{\mathrm{init}_{pkiss}} + 4 \cdot \mathrm{E}_{\mathrm{npp}}$ $+ \mathrm{E}_{\mathrm{termau}}({}_{i}a, a'_{j}) + \mathrm{E}_{\mathrm{termau}}({}_{j-1}a, a'_{i+1})$ $+ \mathrm{E}_{\mathrm{termau}}({}_{i}b, b'_{j}) + \mathrm{E}_{\mathrm{termau}}({}_{j-1}b, b'_{i+1})$ $+ \mathrm{E}_{\mathrm{termau}}({}_{i}c, c'_{j}) + \mathrm{E}_{\mathrm{termau}}({}_{j-1}c, c'_{i+1})$ $+ \mathrm{E}_{\mathrm{dli}}({}_{j-1}a, a'_{i+1})$ $+ \mathrm{E}_{\mathrm{dri}}({}_{j-1}c, c'_{i+1})$ $+ \mathrm{E}_{\mathrm{dr}}({}_{i}a, a'_{j})$ $+ \mathrm{E}_{\mathrm{dl}}({}_{i}c, c'_{j})$ $, {}_{i}c, a'_{j})$
$\mathcal{S}$	int
$\mathcal{K}$	(int,int,int)
$\mathbf{h}$	min.
$\mathbf{h}_{knot}$	min.

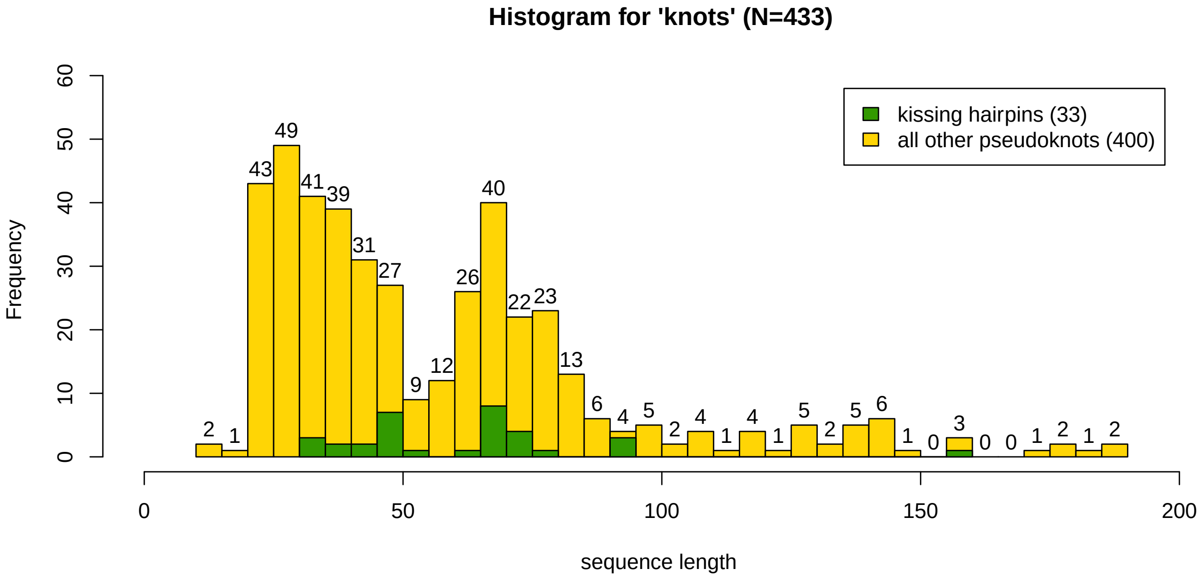


Figure 5.8: Sequence length histogram for 433 members of test set “knot”.

Figure 5.8 for a sequence length distribution. The examples stem on the one hand from PseudoBase [90] (363 sequences) and on the other hand from RNAstrand [2] (70 sequence).

Unfortunately, we had to extract the necessary information for PseudoBase from HTML pages. Sequences “PKB106”, “PKB127”, “PKB149” and “PKB217” had to be excluded, either for larger gaps or degenerated bases.

For RNAstrand, we used the elaborate search function with the following restrictions:

- Type: Any type
- Organism: Any
- Source: Any source , Source ID: Any
- Length: Less than or equal to 200
- Validated by NMR or X-Ray: Yes Number of molecules in complex: Any number
- Fragment: Any
- Duplicates: Non-redundant sequences only
- Sequence pattern: Any
- Abstract shape: Any
- allowed base letters: ACGUT, but no P!
- Number of pseudoknots per molecule: Greater than or equal to 1

5.5.3 Comparing pKiss strategies to other prediction tools

In the following, we compare run-time, memory consumption and accuracy – in different terms – of the several strategies of pKiss with pknotsRG [71], pknotsSE (version 1.05,

[76]), HotKnots (version 2.0, [5]), ProbKnot (version 5.3 of the RNAstructure package, [6]), DotKnot [83] (version 1.3.1) and as a "negative" control with nested structure prediction in terms of $\mathcal{G}_{\text{MicroState}}(\mathcal{I}_{mfe} * \mathcal{I}_{db}, x)$ results. A further strategy is added, called "pKiss A left", where we omit computation of kissing hairpin motifs, starting with an optimal right csrPK. In our terminology "pKiss with strategy P" equals pknotsRG. The HotKnots software package provides three different parameter sets (DP, CC and RE). Since we do not know a priori the best set, we test all three of them. The ProbKnot software uses an iterative approach to converge to a stable prediction. We allow for 10 such iterations at most (-i 10). We run all tests in our default environment (see Section 3.3.2) with $8GB$ memory limitation.

Run-times

Figure 5.9 summarizes the run-times for all 13 prediction tools for test set "knot" as boxplots. As expected, the extra effort for pseudoknots significantly raises run-times compared to nested folding (nested microstate). Practical run-times for strategies A to D nicely follow the theoretical considerations. Since pknotsSE covers pseudoknot classes even larger than kissing hairpin motifs, it is no surprise that its run-times exceeds even those of strategy D, while their asymptotics are within the same class of $O(n^6)$. The same argument is true for strategies P and A, where the latter covers the bigger class of pseudoknots. A bit surprising is the big impact of the chosen parameter set on run-times for HotKnots. We can only speculate that different parameter sets may lead to different foldingspace sizes. Computing only half of the kissing hairpin motif variants (strategy A left) does not yield a 2-fold speed-up.

Memory consumption

Memory consumption (Figure 5.10) follows our expectations. While asymptotic consumption lies in $O(n^2)$ for nested and knotted structures, the number of tabulated non-terminals increase memory consumption for search spaces including pseudoknots (nested microstate vs. pKiss strategies). Strategy B trades run-time for memory, which leads to a significantly larger footprint in the consumption. The larger search space of pknotsSE requires high memory consumption.

Accuracy

We ran every program on each of the test sequences of the set "knot" individually. Programs, that are written in Bellman's GAP, are able to report co-optimal results. Since HotKnots, ProbKnot, pknotsSE and DotKnot cannot, we chose to always randomly pick one co-optimal prediction as result, to compensate this disadvantage. We take the annotated structures from the databases as the "truth".

Base-pair distance Here, we use the traditional base-pair distance, which is defined as $|A \setminus B| + |B \setminus A|$, where A and B are the sets of base-pairs in structure A and structure

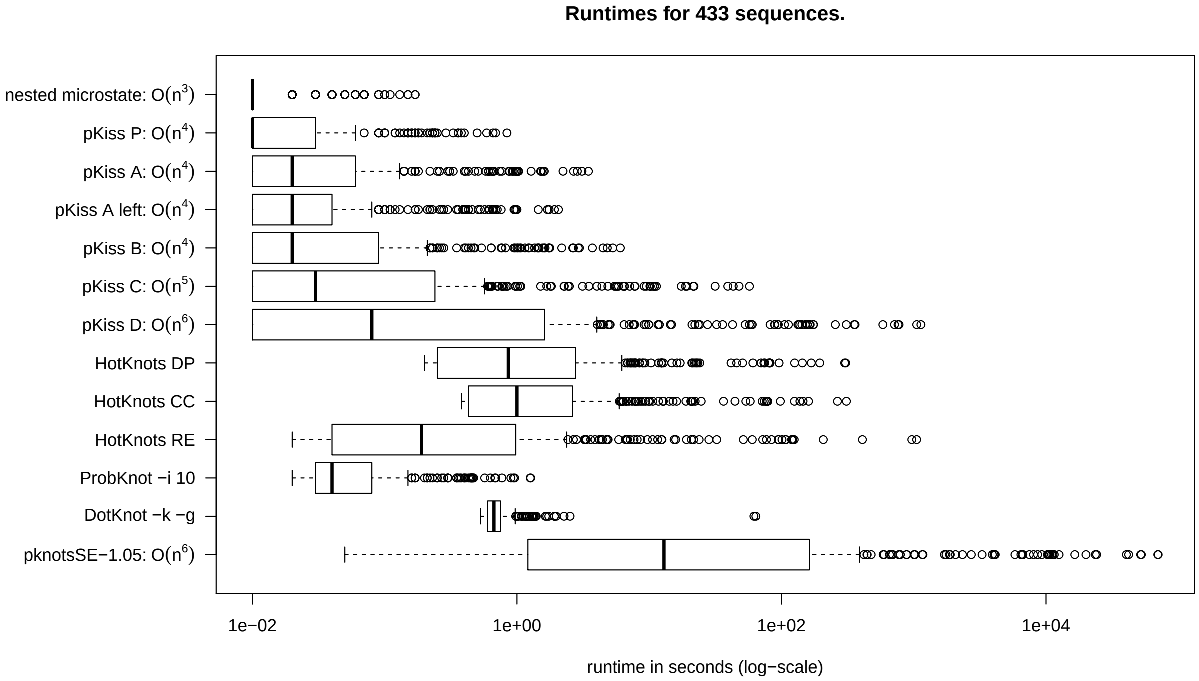


Figure 5.9: Run-times as boxplots for pseudoknot aware prediction tools for the 433 members of test set "knot". Please note the log-scaled x axis.

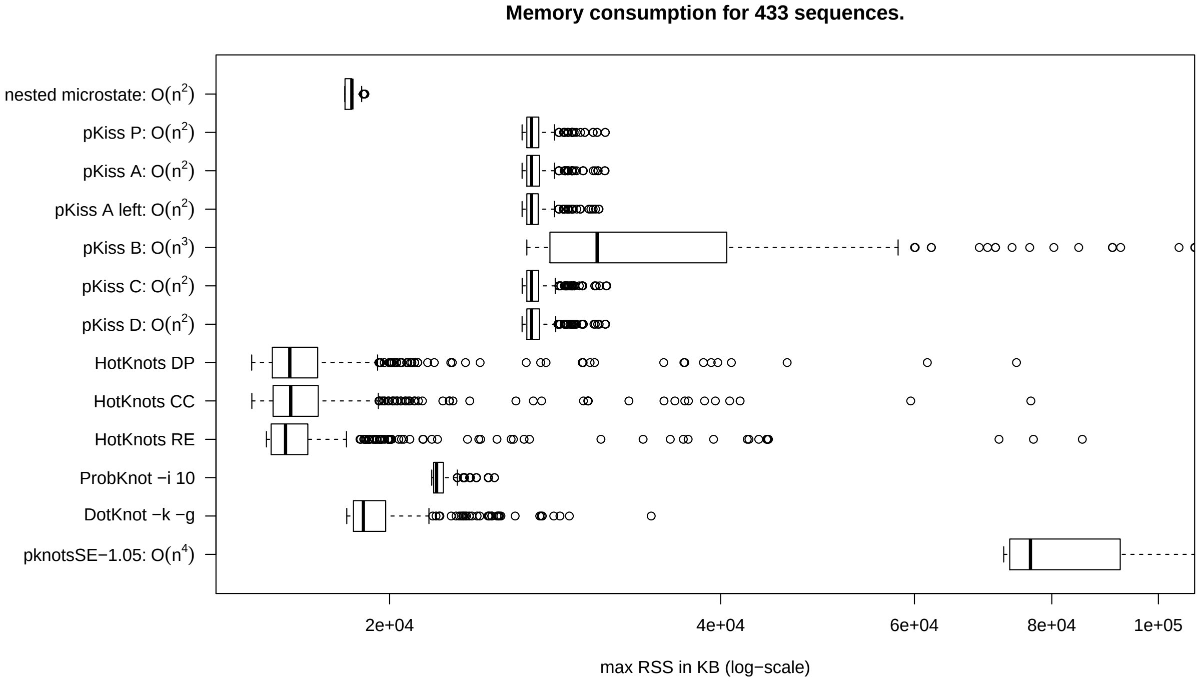


Figure 5.10: Memory consumption as boxplots for pseudoknot aware prediction tools for the 433 members of test set "knot". Please note the log-scaled x axis.

B. Figure 5.11 visualizes summed base-pair distances of all sequences in set “knot”. As expected for the negative control, “nested microstate” performs worst. Surprisingly, ProbKnot does not much better. The heuristic HotKnots heavily depends on the chosen parameter set and comes close to the best results with set “DP”. The slow and most exhaustive program pknotsSE, with the largest search space does not return best performance; all strategies of pKiss do better.

The strategies A to D perform best on the kissing hairpin sub-set (greenish), but on all sequences (yellowish) strategy P (without the ability to predict kissing hairpins) achieve smallest base-pair distance to the annotated structures. It seems to be the case that strategies A to D predict too many base-pairs. But we do not know how many of the annotated structures might turn out to be kissing hairpin motif as well as the “human coronavirus 229E” example (Section 5.5.1).

Unexpectedly, Strategy A performs best among A, B, C and D – it is faster and provides almost the same results as the most in-depth strategy D. Closer inspection showed that it is always the left pseudoknot of the overlay which was chosen optimally. One may speculate that this is because the strategy is consistent with the hierarchic folding path during transcription. Strategy “A left” seems to confirm this speculation.

Previously, we discussed about the hierarchy of search spaces of pKiss, which stem from different levels of heuristic decisions. The surprisingly small inter-strategy base-pair distances, shown in Figure 5.11, let us hope that we can tackle most input sequences with the fast strategy A, without much mispredictions.

Stem arrangement distance In Chapter 3, we used an asymmetric base-pair distance, due to the fact that all folding tools tend to predict additional base-pairs, which often are “compatible” to existing ones. With the presence of crossing base-pairs, definition of “compatible” is more complicated. Furthermore, base slippage or small stem breaks cannot be recognized as tiny changes between structures *A* and *B* within the definition of base-pair distance. Thus, we introduce here the “stem arrangement distance”.

All unpaired bases in the structures are ignored. Successive base-pairs are collapsed into one symbol for the opening part (capital letters) and another for the closing part (small letters). We than define one structure as “reference” (*R*) and the other as “prediction” (*P*). The following example illustrates the conversion from Dot-Bracket strings to stem arrangements:

	concrete secondary structure	stem arrangement
R:	`(..((..[[[[...{{..]].]].}}).))..`	`ABCbca`
P:	`.[[..[[[[.<<<...]]]].]]((.((...)))).>>.>`	`ABaCcb`

Next, we need to align both stem arrangements. Classical sequence alignment fails due to two reasons. First, symbols are not independent, they always have a partnering symbol somewhere in the string; those connections should never be violated during alignment. Second, the cost of an replacement does not depend on the symbol content, i. e. the stem representing letter or – in terms of Dot-Bracket strings – the bracket type, since both are just artifacts from the representation and do not constitute properties of the

	Truth	nested microstate	pKiss P	pKiss A	pKiss A left	pKiss B	pKiss C	pKiss D	HotKnots DP	HotKnots CC	HotKnots RE	ProbKnot -i 10	DotKnot -k -g	pknotsSE-1.05
Truth	0	6364	4563	4647	4643	4606	4641	4594	4843	5143	5459	6347	4713	4884
nested microstate	658	0	4895	5537	5473	5468	5535	5526	4509	4341	2489	2819	4749	5868
pKiss P	561	293	0	1814	1712	1691	1810	1915	3928	4302	4118	5366	3580	4369
pKiss A	483	491	300	0	184	293	44	221	4188	4578	4696	6000	4044	4987
pKiss A left	483	491	300	0	0	289	228	405	4210	4600	4632	5928	3994	4925
pKiss B	483	491	304	4	4	0	337	514	4221	4617	4627	5959	4033	5010
pKiss C	483	493	298	2	2	6	0	213	4176	4566	4684	6004	4052	4989
pKiss D	461	493	310	24	24	28	22	0	4219	4583	4715	6025	4029	4974
HotKnots DP	618	172	325	449	449	449	451	451	0	858	3328	4972	3886	4541
HotKnots CC	617	165	318	442	442	442	444	444	7	0	3430	4956	4224	4773
HotKnots RE	641	69	320	510	510	510	512	512	181	174	0	3364	3880	4725
ProbKnot -i 10	644	108	307	497	497	497	499	499	198	191	121	0	4916	5613
DotKnot -k -g	610	270	339	447	447	447	447	447	286	279	265	276	0	4339
pknotsSE-1.05	562	482	451	571	571	575	571	553	382	375	463	464	468	0

Figure 5.11: Prediction accuracy in terms of base-pair distance. Summed over all 433 sequences of “knot” in the yellowish upper right triangle. The greenish lower left triangle displays results just for the 33 sequences of “knot”, which are annotated as kissing hairpin motifs. The overall number of annotated base-pairs in the “knot” is 7,528.

objects themselves. The second point leads to the fact that any symbol pair of R might match to any symbol pair of P, which means that sequence ordering in R and P must be flexible during alignment. We are not aware of any efficient alignment algorithm fulfilling these properties. Since our input strings are short enough, we stick to the exhaustive exponential enumeration of all symbol pair combinations between both sequences. In addition to those *replacements*, we allow for *insertions* of additional stems in P relative to R and *deletions*, where a stem in R has been lost in P. Due to the tendency of predicting additional base-pairs, we chose the following scoring scheme and minimize over all candidates: *match* = 0, *insertion* = 0, *deletion* = 1.

One out of three co-optimal alignments for our example is:

```
R:  ABCb--ca
P:  -BCbDdc-
```

We renamed the stems of P (A → B, B → C, C → D) to better illustrate the matches. The distance is 1, because we match stems B and C, insert stem D and delete stem A. Only the latter implies non-zero costs.

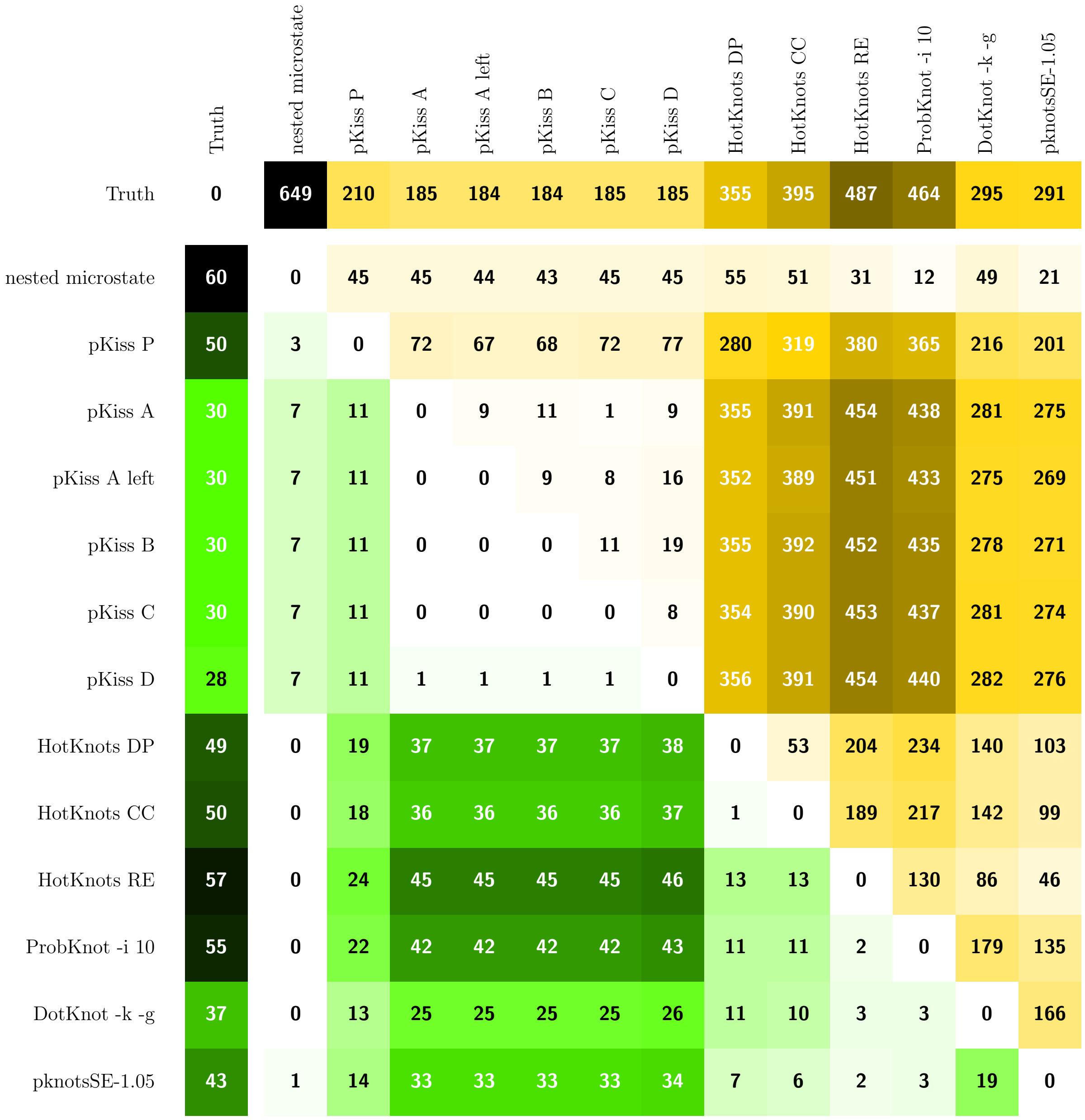
Figure 5.12 reports the summed stem arrangement distances for all tools over set "knot". Since we now deal with an asymmetric distance it is important how to read the figure. If you focus only on kissing hairpin examples (greenish), than the columns are R and the rows are P. Vice versa for all examples (yellowish).

The vast majority of "knot" is annotated as H-type pseudoknots, i. e. with basically two stems. Thus, it makes perfect sense, that the negative control "nested microstate", which can predict one of both crossing stems at most, mispredicts roughly every second stem (647 out of 1,278). Stem distance 7 between R = "nested microstate" and the pKiss strategies A to D result from multiloop enclosing stems (A in the example) in the nested structures, which are thermodynamically no longer stabilizing in the presence of pseudoknots:

```
R:  AB-bD-da
P:  -BCbDcd-
```

Except from the fact that pKiss strategies A to D now perform better than pKiss strategy P, we observe the same tendencies as for base-pair distance.

Topology distance The introduction of energetic penalties ($\mathrm{E}_{\mathrm{init}_{pk}}$, $\mathrm{E}_{\mathrm{init}_{pkiss}}$) to open pseudoknot motifs follows the idea to penalize opening a multiloop sub-structure (E_{ml}), but their actual values have never been verified in a wet lab. Thus, our programs might tend to over- or under-represent pseudoknots. In order to investigate this issue, we use an even more coarse grained distance than before, the *topology distance*. The authors of [73] propose to classify secondary structures into "γ-structures", which are sub-divided depending on their *shadow*-reducibility, see Figure 5.13. A *shadow* is the more mathematical definition of our stem arrangement. Nested structures are "0-structures", the pseudoknot class for pknotsRG is a "1-structure" of *shadow* "H", kissing hairpin motifs are "1-structures" of *shadow* "K".



	Truth	nested microstate	pKiss P	pKiss A	pKiss A left	pKiss B	pKiss C	pKiss D	HotKnots DP	HotKnots CC	HotKnots RE	ProbKnot -i 10	DotKnot -k -g	pknotsSE-1.05
Truth	0	649	210	185	184	184	185	185	355	395	487	464	295	291
nested microstate	60	0	45	45	44	43	45	45	55	51	31	12	49	21
pKiss P	50	3	0	72	67	68	72	77	280	319	380	365	216	201
pKiss A	30	7	11	0	9	11	1	9	355	391	454	438	281	275
pKiss A left	30	7	11	0	0	9	8	16	352	389	451	433	275	269
pKiss B	30	7	11	0	0	0	11	19	355	392	452	435	278	271
pKiss C	30	7	11	0	0	0	0	8	354	390	453	437	281	274
pKiss D	28	7	11	1	1	1	1	0	356	391	454	440	282	276
HotKnots DP	49	0	19	37	37	37	37	38	0	53	204	234	140	103
HotKnots CC	50	0	18	36	36	36	36	37	1	0	189	217	142	99
HotKnots RE	57	0	24	45	45	45	45	46	13	13	0	130	86	46
ProbKnot -i 10	55	0	22	42	42	42	42	43	11	11	2	0	179	135
DotKnot -k -g	37	0	13	25	25	25	25	26	11	10	3	3	0	166
pknotsSE-1.05	43	1	14	33	33	33	33	34	7	6	2	3	19	0

Figure 5.12: Prediction accuracy in terms of stem arrangement distance. Summed over all 433 sequences of “knot” in the yellowish upper right triangle. The greenish lower left triangle displays results just for the 33 sequences of “knot”, which are annotated as kissing hairpin motifs. All 7,528 annotated base-pairs in “knot” collapse into 1,278 stems.

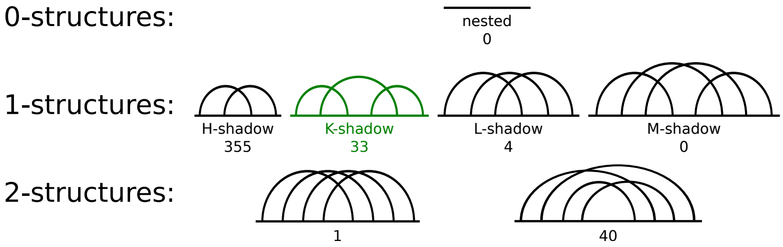


Figure 5.13: Classification of secondary structures according to [73]. Numbers show the distribution of the 433 sequences of test set "knot" into the seven classes. The green class corresponds to kissing hairpin motifs.

The topology distance between a reference structure and a prediction is 1 if their structure-shadow classes are not identical and 0 otherwise.

Our program pKiss provides the possibility to set $\mathrm{E}_{\mathrm{init}_{pk}}$ and $\mathrm{E}_{\mathrm{init}_{pkiss}}$ as command line parameters. We re-ran pKiss strategy A on "knot" with varying penalties. All combination of $\mathrm{E}_{\mathrm{init}_{pk}} \in \{0.0, 3.0, 7.0, 8.0, 9.0, 10.0, 11.0, 15.0, 18.0\}$ and $\mathrm{E}_{\mathrm{init}_{pkiss}} \in \{0.0, 5.0, 10.0, 11.0, 12.0, 13.0, 14.0, 19.0, 24.0\}$ have been tested; data not shown. Since the best combination $\mathrm{E}_{\mathrm{init}_{pk}} = 7.0$ and $\mathrm{E}_{\mathrm{init}_{pkiss}} = 10.0$ could only improve predicted topologies for 5 instances, we decided to stick to the initial parameter values. A more thorough analysis should be carried out in the future.

5.6 Conclusion

Should the observations from our evaluation on "knot" data generalize, interesting algorithmic perspectives open up. Strategy A evaluates a more complex motif than simple pseudoknots – without increasing asymptotic complexity. Unexpectedly, Strategy A performs best among A, B, C and D – it is faster and provides almost the same results as the most in-depth strategy D. Closer inspection showed that it is always the left pseudoknot of the overlay which was chosen optimally. One may speculate that this is because the strategy is consistent with the hierarchic folding path during transcription. Boldly dropping the symmetric computation starting from the right pseudoknot reduces work in the innermost loop and may provide a speed-up factor close to 2. On "knot" data, prediction results even improve, while the expected speed-up factor is just ≈ 1.2.

The more exciting perspective is the extension of the overlay idea to more complex structures. A motif of four hairpins with two kissing interactions, for example, can be overlaid as $a\_b\_a'\_c\_b'\_c'$ and $b\_c\_b'\_d\_c'\_d'$. Using ideas of Strategy A, this can, again, be achieved in $O(n^4)$ time and $O(n^2)$ space! Additionally, alternative decompositions, say $a\_b\_a'\_c\_b'\_c'$ with $c\_d\_c'\_d'$ (a kissing hairpin overlaid with a simple pseudoknot) may be investigated, without raising the asymptotics. Furthermore, two such double kissing structures can form an overlay, and so on. It appears that one can construct a variety of practically useful, albeit increasingly heuristic, programs for pseudoknotted motifs of increasingly complex topologies within $O(n^4)$ time and $O(n^2)$ space.

	Truth	nested microstate	pKiss P	pKiss A	pKiss A left	pKiss B	pKiss C	pKiss D	HotKnots DP	HotKnots CC	HotKnots RE	ProbKnot -i 10	DotKnot -k -g	pknotsSE-1.05
Truth	0	433	114	164	157	158	164	165	226	260	334	292	194	219
nested microstate	33	0	384	402	402	402	402	402	262	229	125	167	311	254
pKiss P	33	14	0	108	101	102	108	111	164	202	261	241	139	151
pKiss A	12	27	23	0	7	7	0	5	219	251	298	286	171	208
pKiss A left	12	27	23	0	0	4	7	12	212	246	297	284	166	202
pKiss B	12	27	23	0	0	0	7	12	214	248	297	285	168	203
pKiss C	12	27	23	0	0	0	0	5	219	251	298	286	171	208
pKiss D	11	27	23	1	1	1	1	0	221	252	298	286	173	209
HotKnots DP	30	7	11	24	24	24	24	24	0	48	155	214	167	130
HotKnots CC	31	7	10	23	23	23	23	23	1	0	152	216	199	151
HotKnots RE	33	1	13	27	27	27	27	27	6	6	0	175	231	165
ProbKnot -i 10	32	2	13	26	26	26	26	26	7	7	3	0	241	191
DotKnot -k -g	20	16	22	20	20	20	20	20	21	20	17	16	0	156
pknotsSE-1.05	32	4	11	24	24	24	24	24	8	7	5	6	15	0

Figure 5.14: Prediction accuracy in terms of topology distance. Summed over all 433 sequences of “knot” in the yellowish upper right triangle. The greenish lower left triangle displays results just for the 33 sequences of “knot”, which are annotated as “1-structures” of *shadow* “K”.

5.6.1 Acknowledgments

RG thanks A. Condon and H. Jabbari for discussion of the pKiss ideas in their early state.

6 Computation of McCaskill base-pair probabilities: an outside algorithm

The chapter at hand reports about a (general) scheme to render outside-in computations possible within the ADP framework, which – by design – allows only inside-out computations.

We chose the classical example of computing base-pair probabilities for RNA sequences, introduced 1990 by McCaskill [62].

Base-pair probabilities are crucial for synoptic secondary structure predictions and build the foundation of some heuristic pseudoknot aware prediction programs, e. g. DotKnot [83]. Furthermore, they allow for accessibility studies, i. e. is a sequence motif accessible for binding with other molecules? This is central for riboswitches, where accessibility, e. g. for ribosomal binding sites or Shine-Dalgarno sequences, is actively controlled by conformational changes of the secondary structure.

Many algorithmic strategies have been proposed to overcome the impediments of MFE predictions for single sequence inputs. Amongst others, like Sfold [13] or the probabilistic mode of RNAshapes [43], the maximum expected accuracy (MEA) is a synoptic approach, i. e. it justifies the decision of optimality not only on *one* candidate, but on properties of the whole foldingspace. MEA relies on base-pair probabilities. It operates in two steps. First, base-pair probabilities are computed [62], i. e. of all candidates from the foldingspace, we select those containing the connection between characters i and j. Their Boltzmann weighted summed energy contribution is related to the combined Boltzmann weight of the whole search space (the *partition function*, see Section 3.2.1), and thus gives the probability of having a connection between i and j. With little modifications to the aforementioned algorithms, this can be done in $O(n^3)$ time for all possible $i - j$ connections at once. The second phase aims to find a secondary structure composed of as many high probability connections as possible.

The above examples should have illustrated that base-pair probabilities truly are the essence of many important RNA algorithms. In the following chapter, we present all technical details which are necessary for an implementation in Bellman's GAP, especially to add the base-pair probability computation to the rapid prototyping ensemble for RNA problems, but also as a first working example of an outside-in calculation in ADP.

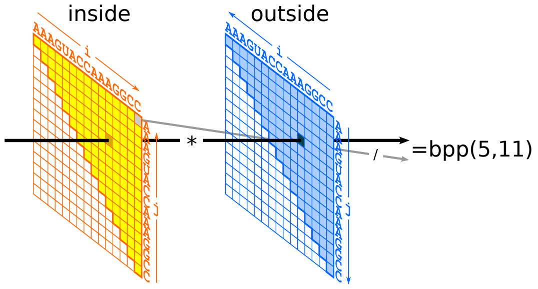


Figure 6.1: Efficient computation of base-pair probabilities (bpp): Probability of base-pair $i = 5, j = 11$ is the product of the entries from the dynamic programming matrices of inside-out and outside-in computation, hit by the black arrow, and divided by the partition function, which is given as the upper right cell of the inside-out matrix. Matrices are computed in $O(n^3)$ time, multiplication for all n^3 cells requires constant time.

6.1 Traditional algorithmic idea

The key for efficient computation of base-pair probabilities for all possible pairs (i, j) is to generate every candidate twice. Once bottom-up, a.k.a. inside-out: candidates start with an arbitrary base from the input as leaves and grow up to their roots, which consume the left- and rightmost character from the sequence. We did so in the previous chapters, e. g. with $\mathcal{G}_{\text{NoDangle}}$, Figure 3.2 on page 34. The top-down phase creates the same candidates in reverse direction, i. e. outside-in. That is not a problem for traditional recurrences, but currently impossible for ADP. Probability of the pair (i, j) is then the combination of the intermediate results of inside-out for sub-sequence i to j and outside-in for sequences 0 to i and j to n, where n is the rightmost character. Last, this value must be normalized by the partition function. See Figure 6.1. Both dynamic programming matrices for inside-out and outside-in are computed in $O(n^3)$ time and $O(n^2)$ space and provide necessary intermediate results for all possible base-pairs at once.

6.2 A general scheme for ADP

To reach the goal of computing base-pair probabilities in ADP, as sketched in Figure 6.1, we also need an outside-in construction of the search space in addition to the inside-out construction via $\mathcal{G}_{\text{NoDangle}}$.

Figure 6.1 is an oversimplification, because we know that $\mathcal{G}_{\text{NoDangle}}$ consists of more than one dynamic programming matrix. Still, it is a good sketch of the overall idea. Let us advance to a more input centric perspective. Figure 6.2 visualizes decomposition of the input sequence, ranging from 0 to n. Please note, that we use the ADP indexing style and address boundaries between characters, not characters themselves. Part a) figures the inside-out construction of candidates via $\mathcal{G}_{\text{NoDangle}}$: assuming free energy minimization,

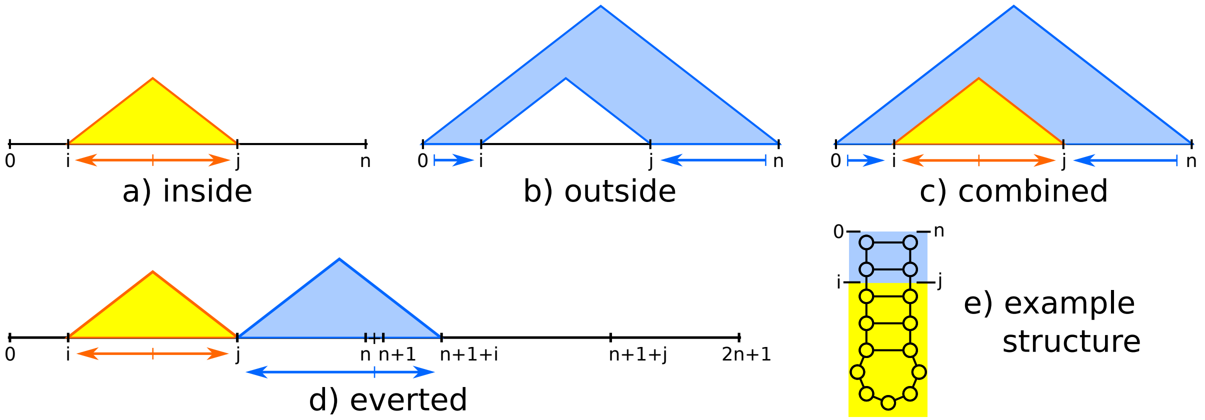


Figure 6.2: Input decomposition for a) inside and b) outside computation direction. Combining both to compute base-pair probabilities would give decomposition c). Since the information flow from inside to outside direction can never be reversed in ADP, i. e. b) becomes impossible, we use the evertion "trick" d) to provide contiguous sub-words for inside and outside halves. Figure Part e) exemplifies cutting a secondary structure into those halves.

see the result of an arbitrary sub-word (i, j), with $0 \leq i < j \leq n$ as energy value for the best contiguous sub-structure possible for this sub-word. It has grown from an empty word ε when $i = j$, positioned somewhere between i and j, heading to the outermost borders 0 and n, but a snapshot was made when it reached i and j. Its equivalent in Figure 6.1 is the yellow lower left triangle, whose summit, holding the result, would be the marked cell. Index offsets of ±1 between Figures 6.1 and 6.2 stem from different sub-word addressing methods. The empty word ε is the seed of the growing sub-word and can be positioned somewhere on the main diagonal of the yellow triangle.

For the outside-in computation of the search space, everything has to be flipped, as shown in Part b) of Figure 6.2. Even the interpretation of intermediate results. Now they are the energetically best arrangement of left and right ends of the input, where a center part is cut out. Imagine a simple hairpin loop with some stacked base-pairs on-top of it, e. g. Part e) in Figure 6.2. The choice of (i, j) cuts this structure in two halves. Let it cut somewhere within the stack. The half containing the loop is the inner part (i, j), covering the center part of the input sequence. The other half is the outer part, holding leading $(0, i)$ and trailing (j, n) ends of the input. Running indices (i, j) start as $(0, n)$ and shrink until $i = j$ for every possible position between 0 and n, i. e. the gap for the inside part has vanished in all possible ways. Translated to Figure 6.1, we would start at the upper rightmost cell and progress to the main diagonal, which then holds identical results for the complete input in every cell. To get the intermediate result for (i, j), we need to fill the rectangle spanned by (i, j) and $(0, n)$. The cell at (i, j) corresponds to the point of the blue triangle in Figure 6.2, where the tip of the white gap is located.

In theory, we can seamlessly combine inside-out and outside-in evaluation (Part c) of Figure 6.2) to compute partial results for both halves for every possible cut point (i, j). Adding up energies for both parts results in the answer for the complete input. Thus, combining intermediate results of both evaluation schemes will always provide the correct answer for the complete input sequence, as long as they are aligned, i. e. there is no gap when superimposing yellow and blue triangles in Figure 6.2. That is the same as moving the black arrow in Figure 6.1 along i or j axis, but never changing its direction.

6.2.1 Outside-in emulation

Unfortunately, reversing the information flow from inside-out to outside-in direction is impossible in ADP, i. e. candidates of the search space are always created bottom-up. Furthermore, a typical Bellman's GAP program can only handle a single *contiguous* sub-word, since intermediate results are stored in two dimensional tables. Introduction of four dimensional tables in Bellman's GAP might allow to define boundaries for the blue areas in parts b) and c) of Figure 6.2. However, this cannot reverse the information flow of ADP.

Thus, the outside-in style cannot be realized as described in Part b) of Figure 6.2, which of cause ruins the combined fashion of Part c). To make the two sub-words $(0, i)$ and (j, n) contiguous, we have to apply the following "trick"[1] : we duplicate the input sequence but separate both copies with a special character, e. g. original input `ccaaagg` becomes `ccaaagg+ccaaagg`, see Part d) of Figure 6.2. Since everything is duplicated, sub-word $(0, i)$ for the leading half is identical to $(n+1, n+1+i)$. Concatenated with the trailing half (j, n) and the separator character $(n, n+1)$, we get the contiguous sub-word (j, n)+$(n + 1, n + 1 + i)$, for which we can apply any ADP computation. Thereby we evert the outside-in direction into an inside-out computation.

By systematically mapping indices back to the original input, we can even re-use existing evaluation algebras. Only the grammar has to be adapted to operate on the everted input sequence. Since yellow and blue triangle, i. e. inside and outside results, are both calculated in inside-out style, we can then compute them in one combined run.

In the remainder of this chapter, we develop all four ADP components to predict McCaskill base-pair probabilities for the NoDangle model in Bellman's GAP, cf. Section 3.2.3. To ease understanding, we first make a detour over free energy evaluation ($\mathcal{I}_{o\_mfe}$), before we transit from MFE prediction to compute base-pair probabilities, basically by switching evaluation algebra to $\mathcal{I}_{o\_bwe}$. After the stage is set for computing base-pair probabilities, we will extend algorithms in two orthogonal directions: Section 6.2.3 covers the different dangling models OverDangle and MicroState. Our contribution to go comparative is given in Section 6.2.4, where we switch input from a single RNA sequence to an RNA alignment.

Signature and Regular Tree Grammar

In the following, we take the NoDangle ADP components of Chapter 3 as foundation and extend them to simultaneously compute inside and outside parts as in the everted

[1] first conceived by Matthias Möhl and Robert Giegerich

Table 6.1: Signature Σ_{o_RNA}, which is an extension of Σ_{RNA} by the following ten algebra functions.

makeplot$(\mathcal{A}^*)$
window$(\mathcal{A}^*, \mathcal{S}, \mathcal{A}^*)$
sep$(\mathcal{S}, \mathcal{A}, \mathcal{S})$
o_sr$(\mathcal{A}, \mathcal{S}, \mathcal{A})$
o_drem$(\mathcal{A}^0, \mathcal{S}, \mathcal{A}^0)$
o_bl$(\mathcal{A}^0, \mathcal{S}, \mathcal{A}^*)$
o_br$(\mathcal{A}^*, \mathcal{S}, \mathcal{A}^0)$
o_il$(\mathcal{A}^*, \mathcal{S}, \mathcal{A}^*)$
o_bp$(\mathcal{A}, \mathcal{S}, \mathcal{A})$
o_ml$(\mathcal{A}, \mathcal{S}, \mathcal{A})$

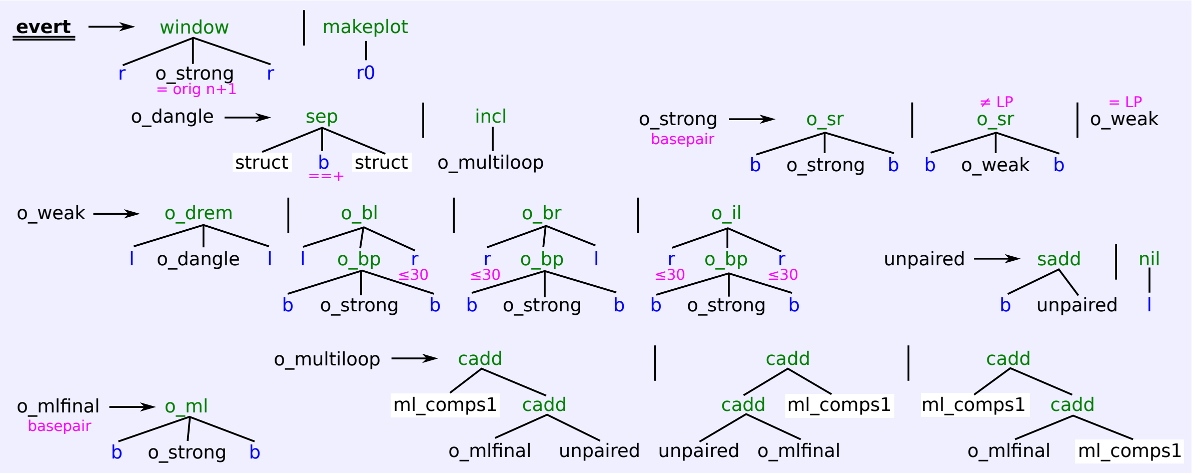


Figure 6.3: Regular tree grammar $\mathcal{G}_{\text{o\_NoDangle}}$ for outside parts of RNA secondary structures. White non-terminals are interfaces to the inside-out grammar $\mathcal{G}_{\text{NoDangle}}$. Axiom is *evert*. Input for this grammar are two copies of an RNA sequence, separated by the character `+`. Terminal parser `r0` consumes zero up to n characters.

style of Figure 6.2. Since extensions mainly cover the outside-in direction, we prefix names with the word “outside”, or “o” for short, but please remember that in fact it is about both directions.

Alphabet $\mathcal{A} = \{A, C, G, U, +\}$ had been slightly extended by the separator character `+`. Signature Σ_{o_RNA} is the extension of Σ_{RNA} (Table 3.1, page 40) and additionally holds the algebra functions given in Table 6.1.

Productions of $\mathcal{G}_{\text{o\_NoDangle}}$ are given in Figure 6.3. You can see the connection between $\mathcal{G}_{\text{o\_NoDangle}}$ and $\mathcal{G}_{\text{NoDangle}}$ by looking at algebra function sep. It uses non-terminal *struct*, which is the axiom of $\mathcal{G}_{\text{NoDangle}}$. New overall axiom is *evert*. Note that $\mathcal{G}_{\text{o\_NoDangle}}$ re-uses algebra functions cadd, incl and nil of Σ_{RNA}.

A narrative explanation of *normal* candidates begins with selection of the right sub-word. Additionally, the special candidate makeplot(`r0`) is added to the search space, that matches each and every input sequence. Terminal parser `r0` matches a sub-word of arbitrary size, it might even be empty. The additional candidate will be used to combine outside and inside information to compute base-pair probabilities. Right now, you can ignore it. Duplication of input produces contiguous sub-word (i, j) for the inside part, contiguous sub-word $(j, n + 1 + i)$ for outside part, but also unnecessary sub-words $(0, i)$ and $(n + 1 + i, 2 \cdot n + 1)$. Function window withdraws the later two from further evaluation by *o_strong*, which also ensures that (i, j) plus $(j, n + 1 + i)$ together have exactly the length of the original input (+1 for separator character) via filter = orig n+1. Depending on whether you allow lonely base-pairs or not, *o_strong* forwards to *o_weak* or initiates a stacking region, which can be extended in both cases via the first alternative. Non-terminal *o_weak* allows for an arbitrary number of nested structural motifs, like left (o_bl) or right (o_br) bulges or internal loops (o_il), but always end by using function o_drem, which refers to non-terminal *o_dangle*. Please note that motifs have been turned inside out. Loop regions are at the outsides, the “closing” base-pair (o_bp) is at the inside. First alternative sep of *o_dangle* terminates candidate construction by parsing the separator character (inner base, which is forced to be + by the filter ==+), but allowing for additional – maybe empty – sub-structures left and right of it. Each is modelled by the full $\mathcal{G}_{\text{NoDangle}}$ grammar, because of the reference to *struct*.

Do not get confused by using $\mathcal{G}_{\text{NoDangle}}$ at this point. We are still modelling the outside half. Imagine a structure of three adjacent hairpins. Further assume we want to split the structure somewhere in the stack of the central hairpin, i. e. we need the candidate where only a few inner base-pairs and the final loop of the central hairpin are missing, which constitute the inner half. Thus, left and right hairpin must inevitably be part of the outside half!

However, reference to *struct* also permits computation of *all* necessary inside halves and thus makes it possible to compute intermediate results for inside and outside within one run.

The second alternative of *o_dangle* is for cases, where the split point is located in a component of a multiloop, but not its closing stem. The distinct component $o\_mlfinal$ can be the last one (first alternative of *o_multiloop*), the first one (second alternative) or it might be surrounded by other components. We need this kind of distinction, because we have to guarantee that a multiloop has at least two enclosed components. The *unpaired* non-terminal is necessary to allow for unpaired bases adjacent to $o\_mlfinal$, i. e. located between the closing stem of the multiloop and its distinct component. Please note that we again employ non-terminals (here *ml_comps*1) from $\mathcal{G}_{\text{NoDangle}}$ for parts of the outside halves.

Evaluation Algebra and choice function

Although candidates for the everted outside-in style computation are somehow turned outside-in versions (cf. Grammar $\mathcal{G}_{\text{o\_NoDangle}}$ in Figure 6.3) of the familiar inside-out calculation, a free energy evaluation should still assign the same energies to those structural motives. Thus, we strive to use the same energy functions as in $\mathcal{I}_{mfe}$ of Table 3.3 on page 44.

As illustrated in Section 2.1.2, energy functions like $\text{E}_{\text{bl}}(i, j)$ not only need access to the bases right of i and left of j from the input, but also inspects further symbols. Due

Table 6.2: Definitions of energy $\mathcal{I}_{o\_mfe}$ and Vienna-Dot-Bracket $\mathcal{I}_{o\_db}$ evaluation algebras for outside computations. Be reminded that those algebras are extensions of their previous definitions, given in Tables 3.3 and 3.2.

algebra function	$\mathcal{I}_{o\_mfe} -\!\rhd \mathcal{I}_{mfe}$	$\mathcal{I}_{o\_db} -\!\rhd \mathcal{I}_{db}$
makeplot(r)	0	ε
window(r_2, x, r_1)	x	x
sep(y, s, x)	$y + x$	$y \oplus + \oplus x$
o_sr$(\hat{a}, x, a)$	$x + \mathrm{E}_{\mathrm{sr}}(\overleftarrow{a}, \hat{a})$	$) \oplus x \oplus ($
o_drem(l_2, x, l_1)	$x + \mathrm{E}_{\mathrm{termau}}(\overleftarrow{l_1}, l_2)$	x
o_bl(l, x, r)	$x + \mathrm{E}_{\mathrm{bl}}(\overleftarrow{r}, l)$	$x \oplus \ldots_{\lvert r \rvert}$
o_br(r, x, l)	$x + \mathrm{E}_{\mathrm{br}}(\overleftarrow{l}, r)$	$\ldots_{\lvert r \rvert} \oplus x$
o_il(r_2, x, r_1)	$x + \mathrm{E}_{\mathrm{il}}(\overleftarrow{r_1}, r_2)$	$\ldots_{\lvert r_2 \rvert} \oplus x \oplus \ldots_{\lvert r_1 \rvert}$
o_bp$(\hat{a}, x, a)$	x	$) \oplus x \oplus ($
o_ml$(\hat{a}, x, a)$	$x + \mathrm{E}_{\mathrm{ml}} + \mathrm{E}_{\mathrm{ul}} + \mathrm{E}_{\mathrm{termau}}(\overleftarrow{a}, \hat{a})$	$) \oplus x \oplus ($
$\mathcal{S}$	int	string
choice function	min.	id

to the inversed character of the grammar, application of $\mathrm{E}_{\mathrm{bl}}(i, j)$ to o_bl would run into the wrong direction and return false values or even causes segmentation faults when accessing positions that do not exist. We can prevent this defect by back transforming indices of the outside algebra functions into the save interval $(0, n)$. Thanks to the design of $\mathcal{G}_{\mathrm{o\_NoDangle}}$, we know that symbols left of + are actually in the original right sub-word (j, n) and symbols right of +, i. e. in $(n + 1, n + 1 + i)$ correspond to $(0, i)$. All we need to do to restore inside fashioned indices is to first subtract $n + 1$ from the right index and then switch left and right indices. We will denote this subtraction for sub-word a as $\overleftarrow{a}$.

Equipped with index manipulating function $\overleftarrow{a}$, we are now able to define a free energy evaluation algebra $\mathcal{I}_{o\_mfe}$ (second column in Table 6.2) and benefit from re-using already established energy look-up functions.

We could also specify an extended Vienna-Dot-Bracket algebra $\mathcal{I}_{o\_db}$ (third column of Table 6.2) to provide string representations for candidates. An analogous algebra product to RNAFOLD would be $\mathcal{I}_{o\_mfe} * \mathcal{I}_{o\_db}$. Since we do not strictly compute an outside-in operation, but an everted version, interpretation of answers is not very helpful: Applying the algebra product $\mathcal{I}_{mfe} * \mathcal{I}_{db}$ to grammar $\mathcal{G}_{\mathrm{NoDangle}}$ (as in Section 3.2.7) together with input sequence `CCCCCGGGGG`, i. e. $\mathcal{G}_{\mathrm{NoDangle}}(\mathcal{I}_{mfe} * \mathcal{I}_{db}$, `CCCCCGGGGG`), would result in `(((....)))` with -2 *kcal*/*mol* as the best candidate. However, the answer of $\mathcal{G}_{\mathrm{o\_NoDangle}}(\mathcal{I}_{o\_mfe} * \mathcal{I}_{o\_db}$, `CCCCCGGGGG+CCCCCGGGGG`) is `)))))+(((((` with -15.6 *kcal*/*mol*, which apparently is not the same thing as in the inside-out calculation. What we compute is the energetically best way to use the original input for constructing the outside half of a secondary structure, regardless of the inside half. Thus, the example sequence is completely consumed to form an ultra-stable helix. We do not have to explicitly forbid

Table 6.3: Definitions of counting evaluation algebra $\mathcal{I}_{o\_count}$ for outside computations.

algebra function	$\mathcal{I}_{o\_count}$	algebra function	$\mathcal{I}_{o\_count}$	algebra function	$\mathcal{I}_{o\_count}$
sadd(a, x)	x	mldr$(a, x, b, \hat{a})$	x	ssadd(r, x)	x
cadd(x, y)	$x * y$	mldlr$(a, b, x, c, \hat{a})$	x	trafo(x)	x
nil(l)	1	addss(x, r)	x	combine(x, y)	$x * y$
drem(l_1, x, l_2)	x	incl(x)	x	acomb(x, b, y)	$x * y$
edl(a, x, l)	x	cadd’(x, y)	$x * y$	makeplot(r)	0
edr(l, x, b)	x	cadd”(x, y)	$x * y$	window(r_2, x, r_1)	x
edlr(a, x, b)	x	cadd”’(x, y)	$x * y$	sep(y, s, x)	$y * x$
sr$(a, x, \hat{a})$	x	ambd(x, b, y)	$x * y$	o_sr$(\hat{a}, x, a)$	x
hl$(a, r, \hat{a})$	1	ambd’(x, b, y)	$x * y$	o_drem(l_2, x, l_1)	x
bl$(a, r, x, \hat{a})$	x	mladl$(a, b, x, \hat{a})$	x	o_bl(l, x, r)	x
br$(a, x, r, \hat{a})$	x	mladr$(a, x, c, \hat{a})$	x	o_br(r, x, l)	x
il$(a, r_1, x, r_2, \hat{a})$	x	mladlr$(a, b, x, c, \hat{a})$	x	o_il(r_2, x, r_1)	x
ml$(a, x, \hat{a})$	x	mldladr$(a, b, x, c, \hat{a})$	x	o_bp$(\hat{a}, x, a)$	x
mldl$(a, b, x, \hat{a})$	x	mladldr$(a, b, x, c, \hat{a})$	x	o_ml$(\hat{a}, x, a)$	x
	$\mathcal{S}$			int	
	choice function			Σ	

candidates in $\mathcal{G}_{\text{o\_NoDangle}}$ search space, which contradict typical structural asumptions like the above one, since they cannot be combined with a meaningful inside half.

Table design

Automated table design by Bellman's GAP fails, because of the non-native semantic filter = orig n+1, which cannot be correctly interpreted by the system. Algebra function window with boundaries window$({}_i\ \mathtt{r}\ {}_k\ o_strong\ {}_l\ \mathtt{r}\ {}_j)$ is falsely assumed to have the two inner moving boundaries k and l in addition to the outer boundaries i and j, resulting in a run-time of $O(n^4)$. Since o_strong must have a constant size due to the application of the filter, l is fixed once k has been chosen and thus we are down to $O(n^3)$. Nevertheless, we stick to the automatically generated suggestion to tabulate non-terminals $\{dangle$, $evert$, $iloop$, ml_comps, ml_comps1, o_dangle, $o\_mlfinal$, $o\_multiloop$, o_strong, o_weak, $strong$, $struct$, $unpaired$, $weak\}$ but not $\{stack, hairpin, leftB, rightB, multiloop\}$.

6.2.2 McCaskill base-pair probability computation

Plan 1: counting

A base-pair probability $bpp(i, j)$ can be seen, assuming all structures are equally likely, as the ratio of structures containing the specific pair (i, j) versus all structures of the search space.

For the example input ${}_0$GA${}_2$GAAAC${}_7$C${}_8$ of Figure 6.4, we know that the search space

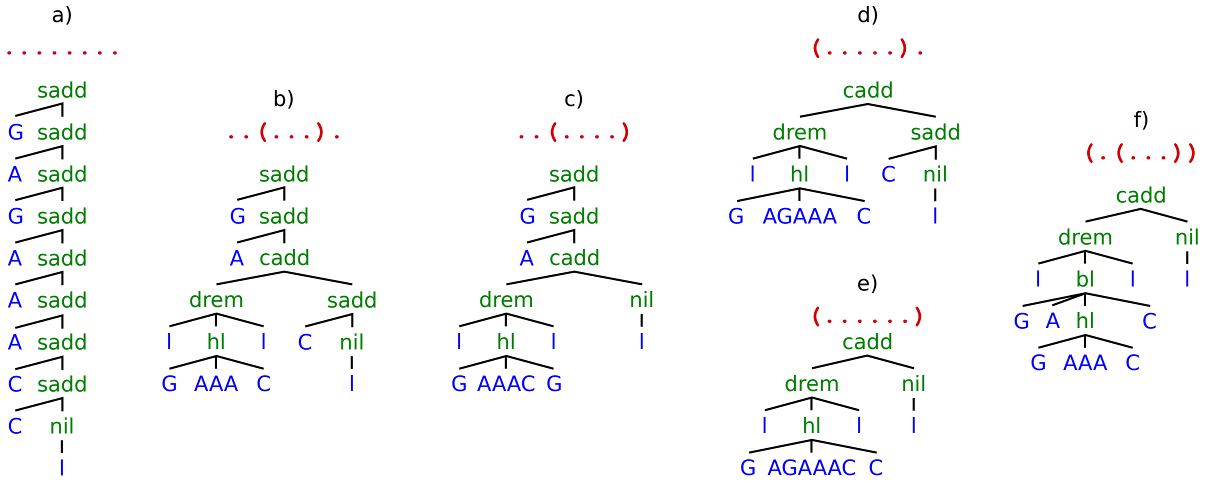


Figure 6.4: All six terms (or candidates) of $F_{\mathcal{G}_{\text{NoDangle}}}$(GAGAAACC) when allowing lonely base-pairs. Their according Vienna-Dot-Bracket strings are given in red, where dots represent unpaired bases, while a pair of parentheses indicate opening and closing position of a base-pair.

contains six candidates. The two candidates, holding the base-pair that connects the outermost bases of the sub-word $(2, 7)$, namely (G,C), are b) and f). Their Vienna-Dot-Bracket representations are ..(...). and (.(...)), respectively. Thus, $bpp(2, 7)$ is $2/6 = \frac{1}{3}$. Probabilities for the other pairs are: $bpp(2, 8) = \frac{1}{6}$, $bpp(0, 7) = \frac{1}{6}$ and $bpp(0, 8) = \frac{1}{3}$. The sum of all base-pair probabilities ($\frac{1}{3} + \frac{1}{6} + \frac{1}{6} + \frac{1}{3} = 1$) might be seen as the average number of base-pairs over all possible structures for the input sequence. In our example this number is 1, since structure a) has none but f) has two pairs. For longer sequences, this number is typically much bigger.

We use the everted style (Section 6.2) to compute all base-pair probabilities within one run of $\mathcal{G}_{\text{o\_NoDangle}}(\mathcal{I}_{o\_count},$ GAGAAACC+GAGAAACC). Denominator is always the size of the search space, which would be result of axiom *struct* of the pure inside grammar $\mathcal{G}_{\text{NoDangle}}$ applied to the original input GAGAAACC. Its correspondent in the everted computation is $struct(0, n)$. Numerator is the product of intermediate inside and outside results. To avoid double counting the base-pair of interest, we decide that it should be part of the inside half. Thus, we only consider those inside candidates that are embraced by a base-pair, which is exactly the definition of non-terminal $weak(i, j)$. Its outside counterpart is $o\_strong(j, n + 1 + i)$, a structure that finally ends with a gap, that fits a base-paired inside half.

Base-pair $(2, 7)$ provides only one situation for a base-paired inner sub-word at non-terminal $weak(2, 7)$, see Figure 6.5. The outside half offers the two candidates at $o\_strong(7, 11)$, also given in Figure 6.5. Remember that we apply counting to compute base-pair probabilities, we get $bpp(2, 7) = weak(2, 7) \cdot o\_strong(7, 11)/struct(0, 8) = 1 \cdot 2/6 = \frac{1}{3}$.

A usual Bellman's GAP program outputs the result of its axiom for the given input

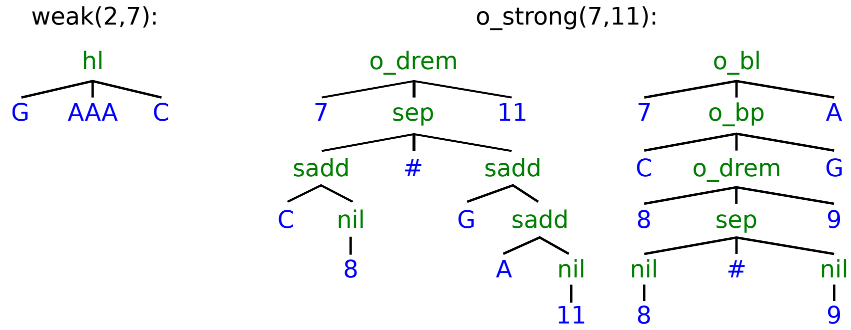


Figure 6.5: Partial candidates for inside and outside halves of the example input `GAGAAACC`. Blue numbers give positions of the `l`ocation terminal parsers.

$(0, n)$, which would be 43 for our example input. This information is quite useless for base-pair probabilities. What we really need is the result of

$$weak(i,j) \cdot o\_strong(j, i+n+1)/struct(0,n)$$

for $\forall\ 0 \leq i < j \leq n$, i. e. we have to access intermediate results of different dynamic programming matrices. That is the point where the strange extra candidate makeplot(`r0`) kicks in. Since `r0` is always satisfied, this extra candidate appears in every search space of $\mathcal{G}_{\text{o\_NoDangle}}$ exactly once. We use a `C++` macro definition to smuggle this extra functionality into the generated code for algebra function makeplot. Using a macro avoids problems with wrong scopes for data structures that are declared as private. It iterates over i and j to compute $bpp(i,j)$. Borrowing an idea from the Vienna-Package, the macro outputs its results in form of a dot-plot, which is written into a PostScript file.

Computation of $bpp(i,j)$ becomes slightly more complicated, if lonely base-pairs are not allowed. As you can see from Figure 3.2 non-terminal *strong* was basically the same as *weak* if $LP =$ **true**. The same holds true for non-terminal *o_strong* and *o_weak* in Figure 6.3. If we set $LP =$ **false**, candidates partition into now different non-terminals and we must take care to connect the right inside and outside halves when composing candidates. Thus, $bpp(i,j)$ is now computed by

$$\left(weak(i,j) \cdot o\_strong(j, i+n+1) + strong(i,j) \cdot o\_weak(j, i+n+1)\right)/struct(0,n)$$

Plan 2: Boltzmann weighted energies

Stochastics based on counting, as done in the previous Plan 1), assigns the same probability to each and every candidate from the search space. We now want to progress to a model that takes the candidates thermodynamic stability into consideration, as we already did for minimal free energy evaluation. Thus, we render the search space for input sequence x into a Boltzmann ensemble, by weighting candidates with their free energies, cf. Section 3.2.1. Base-pair probabilities $bpp(i,j)$ will still be computed as in Plan 1), but instead of the pure number of candidates, we are going to use partition

Table 6.4: Boltzmann weighted energy evaluation algebra for outside computation $\mathcal{I}_{o\_bwe}$, which inherits all algebra functions from the inside version $\mathcal{I}_{bwe}$, see Table 3.6 on page 47.

algebra function	$\mathcal{I}_{o\_bwe} \mathrel{-\!\!\triangleright} \mathcal{I}_{bwe}$		
window(r_2, x, r_1)	x		
sep(y, s, x)	$y \cdot x$		
makeplot(r)	1.0		
o_drem(l_2, x, l_1)	x	$\cdot\, pf(\mathrm{E}_{\mathrm{termau}}(\overleftarrow{l_1}, l_2))$	
o_sr$(\hat{a}, x, a)$	x	$\cdot\, pf(\mathrm{E}_{\mathrm{sr}}(\overleftarrow{a}, \hat{a}))$	$\cdot\, sc(2)$
o_bp$(\hat{a}, x, a)$	x		$\cdot\, sc(2)$
o_bl(l, x, r)	x	$\cdot\, pf(\mathrm{E}_{\mathrm{bl}}(\overleftarrow{r}, l))$	$\cdot\, sc(\lvert r\rvert)$
o_br(r, x, l)	x	$\cdot\, pf(\mathrm{E}_{\mathrm{br}}(\overleftarrow{l}, r))$	$\cdot\, sc(\lvert r\rvert)$
o_il(r_2, x, r_1)	x	$\cdot\, pf(\mathrm{E}_{\mathrm{il}}(\overleftarrow{r_1}, r_2))$	$\cdot\, sc(\lvert r_2\rvert + \lvert r_1\rvert)$
o_ml$(\hat{a}, x, a)$	x	$\cdot\, pf(\mathrm{E}_{\mathrm{ml}} + \mathrm{E}_{\mathrm{ul}} + \mathrm{E}_{\mathrm{termau}}(\overleftarrow{a}, \hat{a}))$	$\cdot\, sc(2)$
$\mathcal{S}$	float		
choice function	Σ		

function values. To do so, we need to replace $\mathcal{I}_{o\_count}$ by the new evaluation algebra $\mathcal{I}_{o\_bwe}$, given in Table 6.4, which does the Boltzmann weighting and summation.

To compute the dot-plot, we can carry over the macro mechanisms from Section 6.2.2. The only change regards the data-type $\mathcal{S}$, which is now a float instead of an int.

Plan 1.5: weight structures according to their number of base-pairs

One might also come up with a third kind of weights to compute base-pair probabilities. Plan 1.5: weighting each structure according to the number of base-pairs it contains. These scores would be more elaborate than simple counting (Plan 1), but still not as sophisticated as Boltzmann weights (Plan 2) and thus maybe a compromise between speed and accuracy. Weights of structures b) to e) of Figure 6.4 would be 1, weight for f) would be 2 and weight for a) would be 0. Applying this scheme, base-pair probabilities would be $bpp(2,7) = bpp(0,8) = \frac{3}{8}$ and $bpp(2,8) = bpp(0,7) = \frac{1}{8}$, which is a slight shift compared to Plan 1. The bad news is, that such probabilities are impossible to compute via dynamic programming! The reason is, that computation of the denominator for the probabilities, e.g. 8 for all base-pairs in $F_{\mathcal{G}_5}$ (CGUG) (see Table 2.2 on page 16), violates Bellman's Principle of Optimality (cf. Section 2.1.1).

To keep the counter example small, we use $\mathcal{G}_5$ from Figure 2.1 to model RNA structures. The algebra to compute the sum of all base-pairs in the foldingspace $\mathcal{I}_{countdb}$ is identical to $\mathcal{I}_{Nussinov}$ (Table 2.3 in on page 17), except for the choice function, which will now be summation instead of maximization.

The following condition is one of the three properties that must hold to satisfy Bellman's Principle of Optimality (Definition 7 of Section 2.1.1):

$$h([f(x_1,\dots,x_k) \mid x_1 \leftarrow Z_1,\dots,x_k \leftarrow Z_k]) = \\ h([f(x_1,\dots,x_k) \mid x_1 \leftarrow h(Z_1),\dots,x_k \leftarrow h(Z_k)]) \quad (6.1)$$

Proof. Let $f(x_1,\dots,x_k)$ be pair(a,x,b,y) with its two terminals $a = \texttt{A}$, $b = \texttt{U}$ and its two result lists for the sub-problems $x = [0,1]$, $b = [\texttt{U}]$ and $y = []$, which we assume to be correct up to this point.

We first evaluate the left hand side of Equation 6.1, by plugging in our example $h([\text{pair}(\texttt{A},x,\texttt{U},y) \mid x \leftarrow [0,1], y \leftarrow []])$, creating all the candidates $h([\text{pair}(\texttt{A},1,\texttt{U},[]),$ $\text{pair}(\texttt{A},0,\texttt{U},[])])$, evaluate each candidate with the algebra function $\text{pair}(a,x,b,y) = 1+x+y$, which gives $h([2,1])$, and finally by applying summation as the choice function, we get **3** as result. Computing the right hand side of Equation 6.1, makes us first apply the choice function to the results for the sub-problems: $h([\text{pair}(\texttt{A},x,\texttt{U},y) \mid x \leftarrow h([0,1]), y \leftarrow h([])])$, which will result in $h([\text{pair}(\texttt{A},x,\texttt{U},y) \mid x \leftarrow [1], y \leftarrow [0]])$. Again, we generate all candidates, here just one $h([\text{pair}(\texttt{A},1,\texttt{U},0)])$ and evaluate this candidate: $h([2])$. At last, the choice function has to be applied a second time, which gives **2** as the final result. But we expected to get 3!

Thus, left and right hand sides disagree; the condition does not hold and thus the problem does not satisfy Bellman's Principle. □

6.2.3 Extension 1: dangling bases

As we saw in Chapter 3, dangling bases can further stabilize RNA secondary structures, without forming actual base-pairs. Four different models have been introduced: NoDangle, OverDangle, MicroState and MacroState. During evaluation, Section 6.3, we are going to compare results of BELLMAN'S GAP outside computations against results from RNAFOLD of the Vienna-Package. Since RNAFOLD lacks the MacroState model and due to its complicated grammar design, we postpone development of an outside MacroState version to future work.

- **MicroState** (Section 3.2.5, RNAFOLD command line option `-d 1`) refines our model of secondary structures by computing with four different candidates for each helical end. The four competing cases are: an unpaired base dangles from the left onto the helix (edl, mldl, o_edl, o_mldl), or from the right (edr, mldr, o_edr, o_mldr), or from both sides while not forming a typical base-pair (edlr, mldlr, o_edlr, o_mldlr). Last case is no dangling from either side (drem, ml, o_drem, o_ml), as known from $\mathcal{G}_{\text{o\_NoDangle}}$.

  The additional candidates are realized by adding further alternatives (Figure 6.6) to the right hand sides of already existing non-terminals of $\mathcal{G}_{\text{o\_NoDangle}}$, thus specifying grammar $\mathcal{G}_{\text{o\_MicroState}}$. Newly introduced algebra functions, like edl or mldr, are defined in Table 6.5 for the use of an extended Boltzmann weighted energy algebra $\mathcal{I}^{MicroState}_{o\_bwe}$. Keeping the makeplot mechanism, a dot-plot for input sequence

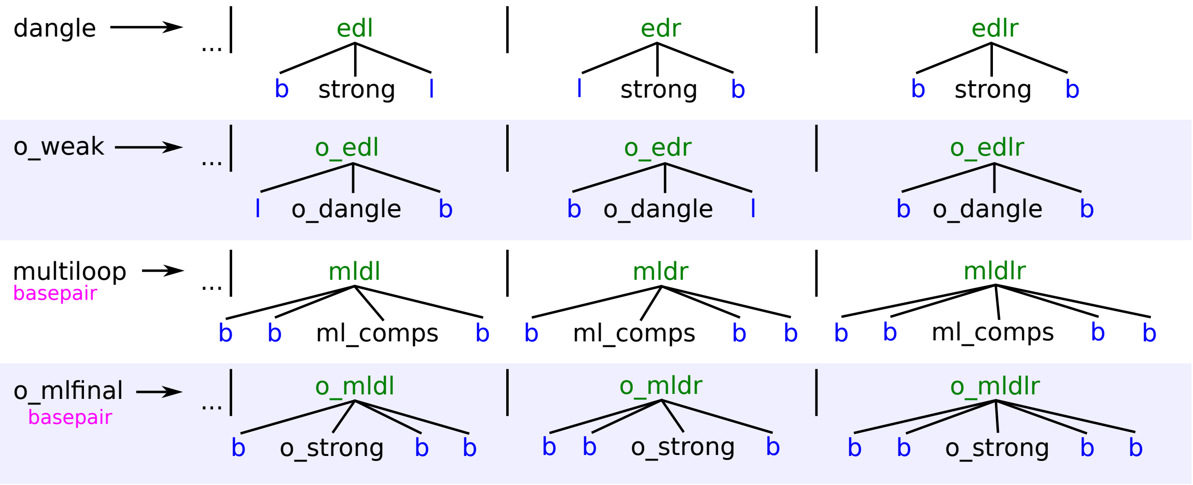


Figure 6.6: Necessary extensions of $\mathcal{G}_{\text{o\_NoDangle}}$ to specify $\mathcal{G}_{\text{o\_MicroState}}$, cf. Figures 3.2 and 6.3. Existing non-terminals, e. g. *dangle*, get additional, alternative right hand sides for dangling from left (dl), dangling from right (dr) and dangling from left and right (dlr). The case of no dangling (drem and ml) was already covered in $\mathcal{G}_{\text{o\_NoDangle}}$.

x and MicroState dangling ends is computed by:

$$\mathcal{G}_{o\_MicroState}\left(\mathcal{I}_{o\_bwe}^{MicroState}, x\right)$$

- **OverDangle** (Section 3.2.4, option `-d 2`) assigns dangling contributions to every helical end, even if neighboring bases are not available, for example because they are themselves engaged in another helix, or already dangling there. It does not increase the size of the search space, as MicroState does. Thus, we can re-use $\mathcal{G}_{\text{o\_NoDangle}}$ (Figure 6.3) and simply rename it $\mathcal{G}_{\text{o\_OverDangle}}$ for readability. Everything left to do is to re-define the four existing algebra functions drem, ml, o_drem and o_ml (see Table 6.6 and compare to Table 6.4) to compute base-pair probabilities with over dangling ends for input sequence x:

$$\mathcal{G}_{\text{o\_OverDangle}}\left(\mathcal{I}_{o\_bwe}^{OverDangle}, x\right)$$

- **NoDangle** (Section 3.2.3, option `-d 0`) simply ignores all dangling ends, which was already realized by the previously defined components:

$$\mathcal{G}_{\text{o\_NoDangle}}\left(\mathcal{I}_{o\_bwe}, x\right)$$

6.2.4 Extension 2: folding alignments

In the lucky situation of having related sequences at hand, we can make use of observing conservation or compensatory mutation throughout evolution. By increasing the information content – quite the opposite meaning of Shannon [82] – using a set of sequences

Table 6.5: Boltzmann weighted energy algebra $\mathcal{I}^{MicroState}_{o\_bwe}$ as an extension of $\mathcal{I}_{o\_bwe}$. Here, we just give definitions for the new algebra functions. Existing ones can be looked up in Table 6.4. A subscript like d_{+1} should indicate, that we address sub-word (d_{i+1}, d_{j+1}) instead of (d_i, d_j), which e. g. is necessary to direct not to the dangling base, but to the left partner of the terminal base-pair in edl.

algebra function	$\mathcal{I}^{MicroState}_{o\_bwe} \mathrel{-\!\triangleright} \mathcal{I}_{o\_bwe}$					
edl(d, x, l)	$x \cdot pf($			$\mathrm{E}_{\mathrm{termau}}(d_{+1}, l)+$	$\mathrm{E}_{\mathrm{dl}}(d_{+1}, l)$	$) \cdot sc(1)$
o_edl(l, x, d)	$x \cdot pf($			$\mathrm{E}_{\mathrm{termau}}(\overleftarrow{d_{+1}}, l)+$	$\mathrm{E}_{\mathrm{dl}}(\overleftarrow{d_{+1}}, l)$	$) \cdot sc(1)$
edr(l, x, b)	$x \cdot pf($			$\mathrm{E}_{\mathrm{termau}}(l, b_{-1})+$	$\mathrm{E}_{\mathrm{dr}}(l, b_{-1})$	$) \cdot sc(1)$
o_edr(b, x, l)	$x \cdot pf($			$\mathrm{E}_{\mathrm{termau}}(\overleftarrow{l}, b_{-1})+$	$\mathrm{E}_{\mathrm{dr}}(\overleftarrow{l}, b_{-1})$	$) \cdot sc(1)$
edlr(d, x, b)	$x \cdot pf($			$\mathrm{E}_{\mathrm{termau}}(d_{+1}, b_{-1})+$	$\mathrm{E}_{\mathrm{ext\_mm}}(d_{+1}, b_{-1})$	$) \cdot sc(2)$
o_edlr(b, x, d)	$x \cdot pf($			$\mathrm{E}_{\mathrm{termau}}(\overleftarrow{d_{+1}}, b_{-1})+$	$\mathrm{E}_{\mathrm{ext\_mm}}(\overleftarrow{d_{+1}}, b_{-1})$	$) \cdot sc(2)$
mldl$(a, d, x, \hat{a})$	$x \cdot pf($	$\mathrm{E}_{\mathrm{ml}}+$	$\mathrm{E}_{\mathrm{ul}}+$	$\mathrm{E}_{\mathrm{termau}}(a, \hat{a})+$	$\mathrm{E}_{\mathrm{dli}}(a, \hat{a})$	$) \cdot sc(3)$
o_mldl$(\hat{a}, x, d, a)$	$x \cdot pf($	$\mathrm{E}_{\mathrm{ml}}+$	$\mathrm{E}_{\mathrm{ul}}+$	$\mathrm{E}_{\mathrm{termau}}(\overleftarrow{a}, \hat{a})+$	$\mathrm{E}_{\mathrm{dli}}(\overleftarrow{a}, \hat{a})$	$) \cdot sc(3)$
mldr$(a, x, b, \hat{a})$	$x \cdot pf($	$\mathrm{E}_{\mathrm{ml}}+$	$\mathrm{E}_{\mathrm{ul}}+$	$\mathrm{E}_{\mathrm{termau}}(a, \hat{a})+$	$\mathrm{E}_{\mathrm{dri}}(a, \hat{a})$	$) \cdot sc(3)$
o_mldr$(\hat{a}, b, x, a)$	$x \cdot pf($	$\mathrm{E}_{\mathrm{ml}}+$	$\mathrm{E}_{\mathrm{ul}}+$	$\mathrm{E}_{\mathrm{termau}}(\overleftarrow{a}, \hat{a})+$	$\mathrm{E}_{\mathrm{dri}}(\overleftarrow{a}, \hat{a})$	$) \cdot sc(3)$
mldlr$(a, d, x, b, \hat{a})$	$x \cdot pf($	$\mathrm{E}_{\mathrm{ml}}+$	$\mathrm{E}_{\mathrm{ul}}+$	$\mathrm{E}_{\mathrm{termau}}(a, \hat{a})+$	$\mathrm{E}_{\mathrm{ml\_mm}}(a, \hat{a})$	$) \cdot sc(4)$
o_mldlr$(\hat{a}, b, x, d, a)$	$x \cdot pf($	$\mathrm{E}_{\mathrm{ml}}+$	$\mathrm{E}_{\mathrm{ul}}+$	$\mathrm{E}_{\mathrm{termau}}(\overleftarrow{a}, \hat{a})+$	$\mathrm{E}_{\mathrm{ml\_mm}}(\overleftarrow{a}, \hat{a})$	$) \cdot sc(4)$
$\mathcal{S}$	float					
choice function	Σ					

Table 6.6: Boltzmann weighted energy algebra $\mathcal{I}^{OverDangle}_{o\_bwe}$ as an extension of $\mathcal{I}_{o\_bwe}$. Special care must be taken to assure that $\mathrm{E}_{\mathrm{ext\_mm}}(l_1, l_2)$ does not access non-existing characters from the input, like -1 or $n+1$ if $l_1 = 0$ or $l_2 = n$, for the inside case. For outside, it gets a bit more complicated, because the separator char + must also be handled as a *virtual* border that cannot be crossed. The same holds true for E_{dl} from Table 6.5. This situation arises if we compute structures with several directly adjacent closed components, where only one of them is handled by the outside part of $\mathcal{G}_{\mathrm{o\_OverDangle}}$ and the others by the inside non-terminal *struct*.

algebra function	$\mathcal{I}^{OverDangle}_{o\_bwe} \mathrel{-\!\triangleright} \mathcal{I}_{o\_bwe}$					
drem(l_1, x, l_2)	$x \cdot pf($			$\mathrm{E}_{\mathrm{termau}}(l_1, l_2)+$	$\mathrm{E}_{\mathrm{ext\_mm}}(l_1, l_2)$	$)$
o_drem(l_2, x, l_1)	$x \cdot pf($			$\mathrm{E}_{\mathrm{termau}}(\overleftarrow{l_1}, \overleftrightarrow{l_2})+$	$\mathrm{E}_{\mathrm{ext\_mm}}(\overleftarrow{l_1}, l_2)$	$)$
ml$(a, x, \hat{a})$	$x \cdot pf($	$\mathrm{E}_{\mathrm{ml}}+$	$\mathrm{E}_{\mathrm{ul}}+$	$\mathrm{E}_{\mathrm{termau}}(a, \hat{a})+$	$\mathrm{E}_{\mathrm{ml\_mm}}(a, \hat{a})$	$) \cdot sc(2)$
o_ml$(\hat{a}, x, a)$	$x \cdot pf($	$\mathrm{E}_{\mathrm{ml}}+$	$\mathrm{E}_{\mathrm{ul}}+$	$\mathrm{E}_{\mathrm{termau}}(\overleftarrow{\hat{a}}, a)+$	$\mathrm{E}_{\mathrm{ml\_mm}}(\overleftarrow{\hat{a}}, a)$	$) \cdot sc(2)$
$\mathcal{S}$	float					
choice function	Σ					

instead of a single one, one aims to improve the structure prediction. Going comparative is a usual trick to increase reliability of predictions, but comes with computationally more expensive problems to solve. Simultaneously aligning and folding m sequences of length n requires $O(n^{3m})$ time with the Sankoff algorithm [78]. Nevertheless, sacrificing the guarantee to find the optimum and split the process into two successive tasks often yields more accurate results than single sequence predictions. Very popular is to team up CLUSTALW [88] for aligning the sequences and RNAALIFOLD [9] to fold the given alignment.

This section first reports all necessary adaptions to the previously introduced ADP components to emulate RNAALIFOLD which finds the energetically most favorable secondary structure (MFE). Second, we extend these ideas to compute base-pair probabilities for RNA alignments.

MFE structure prediction for alignments

Input alphabet The input alphabet must again be extended to cover gap characters _ and *end of row* symbols #: we get $\mathcal{A} = \{A, C, G, U, \_, \#\}$. An alignment can then be provided to BELLMAN'S GAP binaries as one lengthy string, where all alignment rows are concatenated by # characters. Internally, the alignment is stored as a matrix. Since terminal parsers address sub-words via left and right boundaries, this mechanism easily carries over from single sequences to single alignment columns (parser `b`) or from sub-strings to sub-matrices, always covering all alignment rows (parsers `r` and `r0`).

Base-pairing filter We still model secondary structures, thus signature, grammar and table design remain untouched. Only the syntactic filter basepair must be reconsidered: For the same positions i and j we might observe a variety of situations for the k rows of the alignment:

- a "normal" base-pair, e. g. (`C`, `G`)
- a further pair, but formed with different bases, e. g. (`U`, `A`), called *compensatory mutation*
- a broken pair, i. e. bases that cannot form a base-pair any longer, e. g. (`A`, `G`)
- a deleted pair, where one or both partners are lost during evolution, indicated by gap symbols, e. g. (`C`, _) or (_, _)

A measure to express the ratio of these different situations for two positions i and j of the alignment is the *covariation*: $c(i, j)$; aka *covariance*. We adopt RNAALIFOLD's definition and nomenclature of covariation [9]. That is why indices to the end of this section address characters or alignment columns, but not boundaries like in BELLMAN'S GAP. The contribution of existing base-pairs is defined as

$$\gamma'(i,j) = \frac{1}{2} \sum_{\substack{\alpha,\beta \in \mathbb{A} \\ \alpha \neq \beta}} \begin{cases} h(\alpha_i, \beta_i) + h(\alpha_j, \beta_j) & \text{if } (\alpha_i, \alpha_j) \in \mathcal{B} \\ & \wedge\ (\beta_i, \beta_j) \in \mathcal{B} \\ 0 & \text{otherwise} \end{cases}$$

where the Hamming distance is defined as

$$h(a,b) = \begin{cases} 0 & \text{if a = b} \\ 1 & \text{if } a \neq b \end{cases}$$

and $\mathcal{B} = \{(\texttt{A},\texttt{U}), (\texttt{U},\texttt{A}), (\texttt{C},\texttt{G}), (\texttt{G},\texttt{C}), (\texttt{G},\texttt{U}), (\texttt{U},\texttt{G})\}$ is the set of possible base-pairs. $\mathbb{A}$ is the alignment matrix, α and β are rows of this alignment. α_i is the i-th character of alignment row α. Thus, the sum permutes over all two rows of the alignment and accounts for all position independent pairs twice, which is the reason for the $\frac{1}{2}$ normalization. Exclusion $\alpha \neq \beta$ would save some time, but is not strictly necessary, because for identical rows Hamming distance will always be 0 and thus would not change the result. The full covariation score γ also includes penalties for sequences in which the (i,j) base-pair cannot be realized:

$$\gamma(i,j) = \gamma'(i,j) - {}^{2}\delta \sum_{\alpha \in \mathbb{A}} \begin{cases} 0 & \text{if } (\alpha_i, \alpha_j) \in \mathcal{B} \\ 0.25 & \text{if } \alpha_i \wedge \alpha_j \text{ are gaps} \\ 1 & \text{otherwise} \end{cases}$$

where δ is a weighting factor; RNAALIFOLD's default is 1.0. The actual implementation of $c(i,j)$ normalizes with the number of sequences in the alignment k, multiplies with a factor of -100 to be more comparable to free energy values and offers another factor η (again, set to 1.0 by default) to weight the whole thing:

$$c(i,j) = \eta \cdot \gamma(i,j) \cdot -100/k$$

Finally, alignment columns i and j are considered to form a base-pair iff

$$c(i,j) \leq 200/k$$

The size of the foldingspace – and thus the run-time – is mainly influenced by the number of possible base-pairs. Depending on the heterogeneity of alignment columns, this number can significantly drop if only a few columns show sufficient covariance. Thus, folding an RNA alignment might even be faster than folding a single RNA sequence of the same size, because of very different foldingspace sizes.

Evaluation algebras Since we stick to the same algebra functions in grammar and signature, we can – in principle – re-use all existing evaluation algebras. Actually, $\mathcal{I}_{db}$, $\mathcal{I}_{nussinov}$ and $\mathcal{I}_{count}$ are ready as they are. For $\mathcal{I}_{yield}$, we have to think about the representation of (successive) alignment column(s). One solution could be to report just the positions. Another way is to compute some kind of a *consensus sequence*. In the end, all kinds of ASCII representations should work.

Free energy of a structure depends on its sequence. We force all sequences, maybe interspersed with gaps, of the alignment into the same structure, although they are not

[2] There is an inconsistency between RNAALIFOLD implementation and manuscript [9], thus we replaced the plus by a minus sign here.

identical. Thus, energy contributions vary from row to row and we should use their averages for an alignment specific algebra. We will use an abbreviation to denote this averaging for an arbitrary motif x:

$$\overline{\mathrm{E}_x}(a,b) = \frac{1}{k} \cdot \sum_{\alpha \in \mathbb{A}} \mathrm{E}_x(\alpha_a, \alpha_b)$$

Remember that a and b are in fact left and right boundaries for the sub-words a and b, thus we do not access character a of row α, but sub-word $({}_ia, a_j)$ of α.

Due to the gaps, it might happen that the nature of a structural motif changes: for example, assume the structure for the alignment implies a bulge loop to the left, but in one row the bulged region is just a gap. The bulge collapses to a “normal” stacking of base-pairs. Another example: an hairpin loop with deletions in the enclosing base-pair drops back to a stretch of unpaired bases. These and all other alignment induced conversions are processed in the energy library of Bellman’s GAP. Therefore, we can simply re-use the familiar energy functions in $\mathcal{I}_{a\_mfe}$.

In addition to the free energy contribution, a structure should also be valued by the amount of its covariance. These considerations result in evaluation algebra $\mathcal{I}_{a\_mfe}$, see Table 6.7. The problem of finding a thermodynamically optimal secondary structure of a given RNA alignment $\mathbb{A}$ is then solved by $\mathcal{G}_{\mathrm{NoDangle}}(\mathcal{I}_{a\_mfe}, \mathbb{A})$.

Base-pair probabilities for alignments

Everything we are left to do to compute base-pair probabilities (*bpp*) for RNA alignments $\mathbb{A}$ is to define a Boltzmann weighted energies evaluation algebra $\mathcal{I}_{a\_o\_bwe}$ for an outside grammar like $\mathcal{G}_{\mathrm{o\_NoDangle}}$, see Table 6.8. A call of

$$\mathcal{G}_{\mathrm{o\_NoDangle}}(\mathcal{I}_{a\_o\_bwe}, \mathbb{A})$$

will then deliver *bpp* for all base-pairs via candidate makeplot.

6.3 Evaluation

Evaluation is three fold. First, we do a correctness check, i. e. ask the question “do our Bellman’s GAP implementations compute the same base-pair probabilities as our template, the Vienna-Package?” (Section 6.3.1). The second part is a short analysis about the impact of the different dangling models and lonely base-pairs (Section 6.3.2). Last (Section 6.3.3) is a practical run-time comparison.

We define the “relative error” as the distance between two base-pair probability matrices *bppm*, which contain all base-pair probabilities $bpp(i,j)\ \forall\ 0 \leq i \leq j \leq n$ for the input of length n, as

$$d(bppm_a, bppm_b) = \sum_{i<j} \begin{cases} \frac{|bpp_a(i,j) - bpp_b(i,j)|}{bpp_a(i,j)} & \text{if } bpp_a(i,j) \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

Normalization by $bpp_a(i,j)$ gives an asymmetric distance, but is necessary to compare results for inputs of different lengths. Thus, a distance might also be > 1.

Table 6.7: Free energy evaluation algebra for RNA alignments $\mathcal{I}_{a\_mfe}$. $\overline{\mathrm{E}_x}$ is the average energy contribution for all k alignment rows for motif x. As in Table 6.5, a notation like d_{+1} is short for shifting left and right boundaries of sub-word d one position to the right. $c(a, \hat{a})$ is the covariation contribution to the score as defined in the previous paragraph. It is a further abbreviation for $c({}_ia, \hat{a}_{j-1})$, which is the correct way to convert from boundary to character addressing. This table shows definitions for a *MicroState* evaluation. *NoDangle* would simply drop functions edl, edr, edlr, mldl, mldr and mldlr. Additionally using starred functions drem* and ml* instead of drem and ml results in an *OverDangle* evaluation.

algebra function	$\mathcal{I}_{a\_mfe}$		
sadd(a, x)	x	$+\overline{\mathrm{E}_{\mathrm{sbase}}}$	
cadd(x, y)	$x + y$		
nil(l)	0		
drem(l_1, x, l_2)	x	$+\overline{\mathrm{E}_{\mathrm{termau}}}(l_1, l_2)$	
edl(d, x, l)	x	$+\overline{\mathrm{E}_{\mathrm{termau}}}(d_{+1}, l) + \overline{\mathrm{E}_{\mathrm{dl}}}(d_{+1}, l)$	
edr(l, x, b)	x	$+\overline{\mathrm{E}_{\mathrm{termau}}}(l, b_{-1}) + \overline{\mathrm{E}_{\mathrm{dr}}}(l, b_{-1})$	
edlr(d, x, b)	x	$+\overline{\mathrm{E}_{\mathrm{termau}}}(d_{+1}, b_{-1}) + \overline{\mathrm{E}_{\mathrm{ext\_mm}}}(d_{+1}, b_{-1})$	
sr$(a, x, \hat{a})$	x	$+\overline{\mathrm{E}_{\mathrm{sr}}}(a, \hat{a})$	$+\ c(a, \hat{a})$
hl$(a, r, \hat{a})$		$\overline{\mathrm{E}_{\mathrm{hl}}}(r)$	$+\ c(a, \hat{a})$
bl$(a, r, x, \hat{a})$	x	$+\overline{\mathrm{E}_{\mathrm{bl}}}(r, \hat{a})$	$+\ c(a, \hat{a})$
br$(a, x, r, \hat{a})$	x	$+\overline{\mathrm{E}_{\mathrm{br}}}(a, r)$	$+\ c(a, \hat{a})$
il$(a, r_1, x, r_2, \hat{a})$	x	$+\overline{\mathrm{E}_{\mathrm{il}}}(r_1, r_2)$	$+\ c(a, \hat{a})$
ml$(a, x, \hat{a})$	x	$+\overline{\mathrm{E}_{\mathrm{ml}}} + \overline{\mathrm{E}_{\mathrm{ul}}} + \overline{\mathrm{E}_{\mathrm{termau}}}(a, \hat{a})$	$+\ c(a, \hat{a})$
mldl$(a, d, x, \hat{a})$	x	$+\overline{\mathrm{E}_{\mathrm{ml}}} + \overline{\mathrm{E}_{\mathrm{ul}}} + \overline{\mathrm{E}_{\mathrm{termau}}}(a, \hat{a}) + \overline{\mathrm{E}_{\mathrm{dli}}}(a, \hat{a})$	$+\ c(a, \hat{a})$
mldr$(a, x, b, \hat{a})$	x	$+\overline{\mathrm{E}_{\mathrm{ml}}} + \overline{\mathrm{E}_{\mathrm{ul}}} + \overline{\mathrm{E}_{\mathrm{termau}}}(a, \hat{a}) + \overline{\mathrm{E}_{\mathrm{dri}}}(a, \hat{a})$	$+\ c(a, \hat{a})$
mldlr$(a, d, x, b, \hat{a})$	x	$+\overline{\mathrm{E}_{\mathrm{ml}}} + \overline{\mathrm{E}_{\mathrm{ul}}} + \overline{\mathrm{E}_{\mathrm{termau}}}(a, \hat{a}) + \overline{\mathrm{E}_{\mathrm{ml\_mm}}}(a, \hat{a})$	$+\ c(a, \hat{a})$
incl(x)	x	$+\overline{\mathrm{E}_{\mathrm{ul}}}$	
addss(x, r)	x	$+\overline{\mathrm{E}_{\mathrm{ss}}}(r)$	
drem$^*(l_1, x, l_2)$	x	$+\overline{\mathrm{E}_{\mathrm{termau}}}(l_1, l_2) + \overline{\mathrm{E}_{\mathrm{ext\_mm}}}(l_1, l_2)$	
ml$^*(a, x, \hat{a})$	x	$+\overline{\mathrm{E}_{\mathrm{ml}}} + \overline{\mathrm{E}_{\mathrm{ul}}} + \overline{\mathrm{E}_{\mathrm{termau}}}(a, \hat{a}) + \overline{\mathrm{E}_{\mathrm{ml\_mm}}}(a, \hat{a})$	$+\ c(a, \hat{a})$
$\mathcal{S}$	int		
choice function	min.		

Table 6.8: Boltzmann weighted energy evaluation algebra for RNA alignments and outside-in computations $\mathcal{I}_{a\_o\_bwe}$. pf and sc are scaling factors (see Table 6.4), c is covariation (see Section 6.2.4) and $\overline{\mathrm{E}_x}$ is the average energy for x (see Table 6.7). Different dangling flavors (*MicroState*, *OverDangle*, *NoDangle*) can be selected as in Table 6.7.

algebra function	$\mathcal{I}_{a\_o\_bwe}$			
sadd(a, x)	$x\cdot$	$pf(\overline{\mathrm{E}_{\mathrm{sbase}}})\cdot$	$sc(1)$	
cadd(x, y)	$x \cdot y$			
nil(l)	1.0			
drem(l_1, x, l_2)	$x\cdot$	$pf(\overline{\mathrm{E}_{\mathrm{termau}}}(l_1, l_2))$		
o_drem(l_2, x, l_1)	$x\cdot$	$pf(\overline{\mathrm{E}_{\mathrm{termau}}}(\overleftarrow{l_1}, l_2))$		
edl(d, x, l)	$x\cdot$	$pf(\overline{\mathrm{E}_{\mathrm{termau}}}(d_{+1}, l) + \overline{\mathrm{E}_{\mathrm{dl}}}(d_{+1}, l))\cdot$	$sc(1)$	
o_edl(l, x, d)	$x\cdot$	$pf(\overline{\mathrm{E}_{\mathrm{termau}}}(\overleftarrow{d_{+1}}, l) + \overline{\mathrm{E}_{\mathrm{dl}}}(\overleftarrow{d_{+1}}, l))\cdot$	$sc(1)$	
edr(l, x, b)	$x\cdot$	$pf(\overline{\mathrm{E}_{\mathrm{termau}}}(l, b_{-1}) + \overline{\mathrm{E}_{\mathrm{dr}}}(l, b_{-1}))\cdot$	$sc(1)$	
o_edr(b, x, l)	$x\cdot$	$pf(\overline{\mathrm{E}_{\mathrm{termau}}}(\overleftarrow{l}, b_{-1}) + \overline{\mathrm{E}_{\mathrm{dr}}}(\overleftarrow{l}, b_{-1}))\cdot$	$sc(1)$	
edlr(d, x, b)	$x\cdot$	$pf(\overline{\mathrm{E}_{\mathrm{termau}}}(d_{+1}, b_{-1}) + \overline{\mathrm{E}_{\mathrm{ext\_mm}}}(d_{+1}, b_{-1}))\cdot$	$sc(2)$	
o_edlr(b, x, d)	$x\cdot$	$pf(\overline{\mathrm{E}_{\mathrm{termau}}}(\overleftarrow{d_{+1}}, b_{-1}) + \overline{\mathrm{E}_{\mathrm{ext\_mm}}}(\overleftarrow{d_{+1}}, b_{-1}))\cdot$	$sc(2)$	
sr($a, x, \hat{a}$)	$x\cdot$	$pf(\overline{\mathrm{E}_{\mathrm{sr}}}(a, \hat{a}))\cdot$	$sc(2)$	$\cdot pf(c(a, \hat{a}))$
o_sr($\hat{a}, x, a$)	$x\cdot$	$pf(\overline{\mathrm{E}_{\mathrm{sr}}}(\overleftarrow{a}, \hat{a}))\cdot$	$sc(2)$	$\cdot pf(c(\overleftarrow{a}, \hat{a}))$
hl($a, r, \hat{a}$)		$pf(\overline{\mathrm{E}_{\mathrm{hl}}}(r))\cdot$	$sc(2 + \lvert r\rvert)$	$\cdot pf(c(a, \hat{a}))$
o_bp($\hat{a}, x, a$)	$x\cdot$		$sc(2)$	$\cdot pf(c(\overleftarrow{a}, \hat{a}))$
bl($a, r, x, \hat{a}$)	$x\cdot$	$pf(\overline{\mathrm{E}_{\mathrm{bl}}}(r, \hat{a}))\cdot$	$sc(2 + \lvert r\rvert)$	$\cdot pf(c(a, \hat{a}))$
o_bl(l, x, r)	$x\cdot$	$pf(\overline{\mathrm{E}_{\mathrm{bl}}}(\overleftarrow{r}, l))\cdot$	$sc(\lvert r\rvert)$	
br($a, x, r, \hat{a}$)	$x\cdot$	$pf(\overline{\mathrm{E}_{\mathrm{br}}}(a, r))\cdot$	$sc(2 + \lvert r\rvert)$	$\cdot pf(c(a, \hat{a}))$
o_br(r, x, l)	$x\cdot$	$pf(\overline{\mathrm{E}_{\mathrm{br}}}(\overleftarrow{l}, r))\cdot$	$sc(\lvert r\rvert)$	
il($a, r_1, x, r_2, \hat{a}$)	$x\cdot$	$pf(\overline{\mathrm{E}_{\mathrm{il}}}(r_1, r_2))\cdot$	$sc(2 + \lvert r_1\rvert + \lvert r_2\rvert)$	$\cdot pf(c(a, \hat{a}))$
o_il(r_2, x, r_1)	$x\cdot$	$pf(\overline{\mathrm{E}_{\mathrm{il}}}(\overleftarrow{r_1}, r_2))\cdot$	$sc(\lvert r_2\rvert + \lvert r_1\rvert)$	
ml($a, x, \hat{a}$)	$x\cdot$	$pf(\overline{\mathrm{E}_{\mathrm{ml}}} + \overline{\mathrm{E}_{\mathrm{ul}}} + \overline{\mathrm{E}_{\mathrm{termau}}}(a, \hat{a}))\cdot$	$sc(2)$	$\cdot pf(c(a, \hat{a}))$
o_ml($\hat{a}, x, a$)	$x\cdot$	$pf(\overline{\mathrm{E}_{\mathrm{ml}}} + \overline{\mathrm{E}_{\mathrm{ul}}} + \overline{\mathrm{E}_{\mathrm{termau}}}(\overleftarrow{a}, \hat{a}))\cdot$	$sc(2)$	$\cdot pf(c(\overleftarrow{a}, \hat{a}))$
mldl($a, d, x, \hat{a}$)	$x\cdot$	$pf(\overline{\mathrm{E}_{\mathrm{ml}}} + \overline{\mathrm{E}_{\mathrm{ul}}} + \overline{\mathrm{E}_{\mathrm{termau}}}(a, \hat{a}) + \overline{\mathrm{E}_{\mathrm{dli}}}(a, \hat{a}))\cdot$	$sc(3)$	$\cdot pf(c(a, \hat{a}))$
o_mldl($\hat{a}, x, d, a$)	$x\cdot$	$pf(\overline{\mathrm{E}_{\mathrm{ml}}} + \overline{\mathrm{E}_{\mathrm{ul}}} + \overline{\mathrm{E}_{\mathrm{termau}}}(\overleftarrow{a}, \hat{a}) + \overline{\mathrm{E}_{\mathrm{dli}}}(\overleftarrow{a}, \hat{a}))\cdot$	$sc(3)$	$\cdot pf(c(\overleftarrow{a}, \hat{a}))$
mldr($a, x, b, \hat{a}$)	$x\cdot$	$pf(\overline{\mathrm{E}_{\mathrm{ml}}} + \overline{\mathrm{E}_{\mathrm{ul}}} + \overline{\mathrm{E}_{\mathrm{termau}}}(a, \hat{a}) + \overline{\mathrm{E}_{\mathrm{dri}}}(a, \hat{a}))\cdot$	$sc(3)$	$\cdot pf(c(a, \hat{a}))$
o_mldr($\hat{a}, b, x, a$)	$x\cdot$	$pf(\overline{\mathrm{E}_{\mathrm{ml}}} + \overline{\mathrm{E}_{\mathrm{ul}}} + \overline{\mathrm{E}_{\mathrm{termau}}}(\overleftarrow{a}, \hat{a}) + \overline{\mathrm{E}_{\mathrm{dri}}}(\overleftarrow{a}, \hat{a}))\cdot$	$sc(3)$	$\cdot pf(c(\overleftarrow{a}, \hat{a}))$
mldlr($a, d, x, b, \hat{a}$)	$x\cdot$	$pf(\overline{\mathrm{E}_{\mathrm{ml}}} + \overline{\mathrm{E}_{\mathrm{ul}}} + \overline{\mathrm{E}_{\mathrm{termau}}}(a, \hat{a}) + \overline{\mathrm{E}_{\mathrm{ml\_mm}}}(a, \hat{a}))\cdot$	$sc(4)$	$\cdot pf(c(a, \hat{a}))$
o_mldlr($\hat{a}, b, x, d, a$)	$x\cdot$	$pf(\overline{\mathrm{E}_{\mathrm{ml}}} + \overline{\mathrm{E}_{\mathrm{ul}}} + \overline{\mathrm{E}_{\mathrm{termau}}}(\overleftarrow{a}, \hat{a}) + \overline{\mathrm{E}_{\mathrm{ml\_mm}}}(\overleftarrow{a}, \hat{a}))\cdot$	$sc(4)$	$\cdot pf(c(\overleftarrow{a}, \hat{a}))$
incl(x)	$x\cdot$	$pf(\overline{\mathrm{E}_{\mathrm{ul}}})$		
addss(x, r)	$x\cdot$	$pf(\overline{\mathrm{E}_{\mathrm{ss}}}(r))\cdot$	$sc(\lvert r\rvert)$	
window(r_2, x, r_1)	x			
sep(y, s, x)	$y \cdot x$			
makeplot(r)	1.0			
drem*(l_1, x, l_2)	$x\cdot$	$pf(\overline{\mathrm{E}_{\mathrm{termau}}}(l_1, l_2) + \overline{\mathrm{E}_{\mathrm{ext\_mm}}}(l_1, l_2))$		
o_drem*(l_2, x, l_1)	$x\cdot$	$pf(\overline{\mathrm{E}_{\mathrm{termau}}}(\overleftarrow{l_1}, l_2) + \overline{\mathrm{E}_{\mathrm{ext\_mm}}}(\overleftarrow{l_2}, l_1))$		
ml*($a, x, \hat{a}$)	$x\cdot$	$pf(\overline{\mathrm{E}_{\mathrm{ml}}} + \overline{\mathrm{E}_{\mathrm{ul}}} + \overline{\mathrm{E}_{\mathrm{termau}}}(a, \hat{a}) + \overline{\mathrm{E}_{\mathrm{ml\_mm}}}(a, \hat{a}))\cdot$	$sc(2)$	$\cdot pf(c(a, \hat{a}))$
o_ml*($\hat{a}, x, a$)	$x\cdot$	$pf(\overline{\mathrm{E}_{\mathrm{ml}}} + \overline{\mathrm{E}_{\mathrm{ul}}} + \overline{\mathrm{E}_{\mathrm{termau}}}(\overleftarrow{a}, \hat{a}) + \overline{\mathrm{E}_{\mathrm{ml\_mm}}}(\overleftarrow{a}, \hat{a}))\cdot$	$sc(2)$	$\cdot pf(c(\overleftarrow{a}, \hat{a}))$
$\mathcal{S}$	float			
choice function	Σ			

6.3.1 Correctness check

For correctness, we check the following three aspects:

Q1) Vienna-Package consistency: traditional index-based dynamic programming recurrences for inside-out and outside-in matrices must seamlessly fit to compute correct base-pair probabilities, cf. Figure 6.2. Our first test is to check if this is the case for the hand crafted Vienna-Package programs. We do so just with tools provided by the Vienna-Package to reduce sources of further errors, i. e. with programs RNAFOLD, RNAALIFOLD and RNASUBOPT.

Q2) BELLMAN'S GAP consistency: Basically the same question as Q1), but now for the outside scheme for BELLMAN'S GAP. Does our everted style base-pair probability computation in $O(n^3)$ provide the same numbers as an exhaustive inside-style enumeration of the complete exponential search space?

Q3) Identity: given both software "worlds" are consistent and the fact that they use the same energy model and parameter set, we check if they really provide identical base-pair probabilities.

To answer these questions we compute base-pair probability matrices in four different ways when evaluating results for single sequence inputs, see Figure 6.7:

1. $\text{Truth}_{\text{Vienna}}$: is a call to RNASUBOPT, which exhaustively enumerates the complete folding space, i. e. each candidate is given as its dot-Bracket representation and free energy. For each candidate and every base-pair (i, j) within this candidate, a PERL script transforms the free energy into a Boltzmann weight and adds this value to a global sum (i, j). To get base-pair probabilities all position specific sums have to be divided by their overall sum – the partition function.

2. $\text{Truth}_{\text{bgap}}$: the same as $\text{Truth}_{\text{Vienna}}$, but computed via BELLMAN'S GAP instance: $\mathcal{G}_{\text{NoDangle}}(\mathcal{I}_{db} * \mathcal{I}_{mfe}, x)$ instead of RNASUBOPT.

3. RNAFOLD: a call to RNAFOLD with parameter `-p` to compute a dot-plot, which contains the base-pair probabilities.

4. outside: is the result of the BELLMAN'S GAP instance $\mathcal{G}_{\text{o\_NoDangle}}(\mathcal{I}_{o\_bwe}, x)$.

Sequences for evaluation stem from a random generator[3]. Bases are uniformly distributed. We use 46 sequences, covering all lengths from 5 to 45 nucleotides. These sequences are part of the fold-grammars repository:
`http://bibiserv.cebitec.uni-bielefeld.de/fold-grammars`.

Regarding questions Q1) to Q3) for single sequence inputs, we conclude the following:

Q1) Consistency for BELLMAN'S GAP is confirmed, since $d(\text{Truth}_{\text{bgap}}, \text{outside})$ (the green lines in Figure 6.7) is very close to zero in all six plots for every test sequence.

[3] RANDOMSEQ from Robert Homann

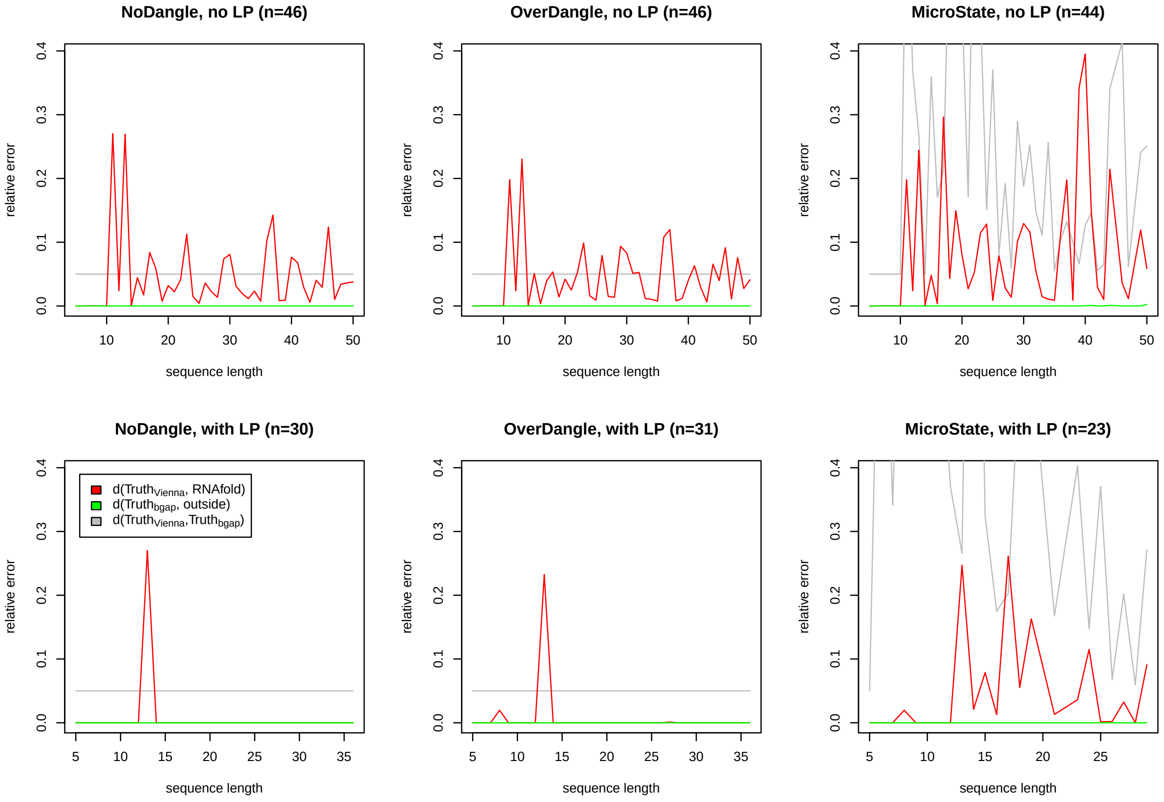


Figure 6.7: Results for correctness check for single sequence input. Plots from left to right are for the three different dangling end modes, upper row is for structures without lonely base-pairs (no LP), whereas they are allowed in the three plots of the lower row (with LP). Sequence length is on the x-axis. Relative error is on the y-axis. n is the sample size, which is not always number of test sequences, because exhaustive enumeration of the search space often produced out of memory errors. Distances $d(\text{Truth}_{\text{Vienna}}, \text{Truth}_{\text{bgap}})$ have been artificially raised by 0.05 units. Otherwise they would have been overdrawn by the green lines.

Q2) Results from the Vienna-Package[4] often do not agree (red lines in Figure 6.7)! This is surprising. The reason is that lonely base-pair exclusion and dangling ends handling are not implemented in a consistent manner throughout the package.

The entry for switching off lonely base-pairs (`--noLP`) in the manual of RNAfold state: "For partition function folding this only disallows pairs that can only occur isolated. Other pairs may still occasionally occur as helices of length 1." *partition function folding* is just another name for computing base-pair probabilities.

A similar disagreement between RNAsubopt and RNAfold regarding the *MicroState* flavor for dangling ends is explained by the manual entry of RNAfold for `-d` parameter: "With -d1 only unpaired bases can participate in at most one dangling end, this is the default for MFE folding but unsupported for the partition function folding."

Non zero red lines for all three plots with no lonely base-pairs (upper row) and both *MicroState* plots (last column) might be due to the presented implementation disagreements. The red spikes in the remaining two plots ("NoDangle, with LP" and "OverDangle, with LP") cannot be explained by these effects and seem to point to an error in the RNAfold implementation. This is supported by the fact that distances $d(\text{Truth}_{\text{bgap}}, \text{outside})$ and the distances between both truths $d(\text{Truth}_{\text{Vienna}}, \text{Truth}_{\text{bgap}})$ (gray line) are both ≈ 0.

Q3) Identity between both implementations ($d(\text{Truth}_{\text{Vienna}}, \text{Truth}_{\text{bgap}})$, gray lines, artificially raised by 0.05 units) is given, except for *MicroState*. The reason for this deviation is that RNAsubopt[5] reports only the energetically best way to dangle bases on-top of a stem. For example, sequence `GAACCGGGUCC` results in the following different search spaces:

RNAsubopt		$\mathcal{G}_{\text{MicroState}}(\mathcal{I}_{db} * \mathcal{I}_{mfe}, x)$	
`((......)).`	2.30	`((......)).`	2.30
		`((......)).`	2.70
`..((...))..`	2.70	`..((...))..`	2.70
		`..((...))..`	3.20
		`..((...))..`	3.40
		`..((...))..`	3.70
`...........`	0.00	`...........`	0.00

Thus, $\text{Truth}_{\text{Vienna}}$ always misses a significant part of the foldingspace for MicroState and truth of both software "worlds" can never provide identical results.

Situation for alignment inputs is similar, see Figure 6.8. Test alignments are randomly cut out sub-alignments of Rfam's[6] seed alignment for family `RF00040`, an "RNase E 5' UTR element". Sub-alignment start positions are randomly chosen. Sub-alignments contain always all rows from `RF00040` alignment. The 46 test alignments vary in length

[4] version 2.1.2

[5] version 2.1.2 with parameters `-d 1 --noLP -e 9999`

[6] release 11.0

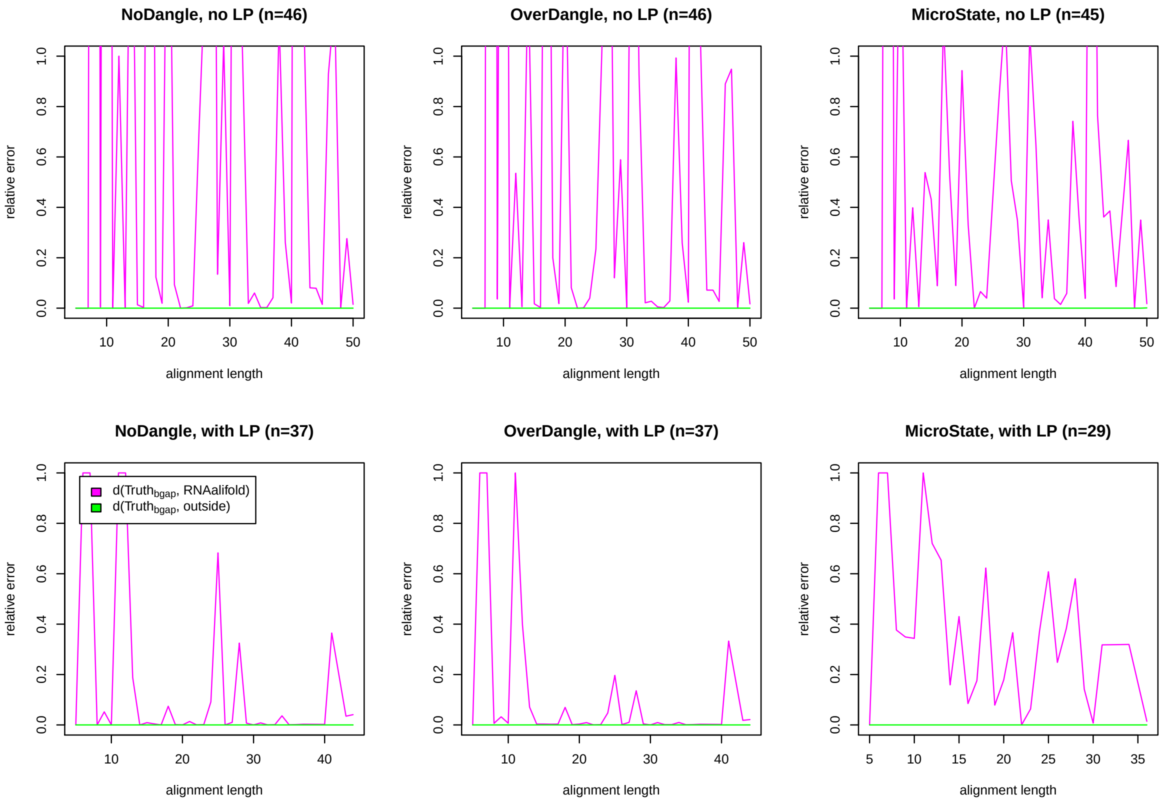


Figure 6.8: Results for correctness check for alignment input. Plots from left to right are for the three different dangling end modes, upper row is for structures without lonely base-pairs (no LP), whereas they are allowed in the three plots of the lower row (with LP). Alignment length is on the x-axis. Relative error is on the y-axis. n is the sample size, which is not always number of test alignments, because exhaustive enumeration of the search space often produced out of memory errors.

from 5 to 45 columns. Concrete alignments are part of the `http://bibiserv.cebitec.uni-bielefeld.de/fold-grammars` repository.

Unfortunately, there is no equivalent to RNAsubopt or RNAeval in the Vienna-Package for alignments. Thus, questions Q2) and Q3) cannot be answered. We can compare the template program RNAalifold only to the truth which is computed by our Bellman's GAP program, resulting in distances $d(\text{Truth}_{\text{bgap}}, \text{RNAalifold})$; magenta lines in Figure 6.8.

The flat green lines for all six plots indicate that we accomplish consistency for Bellman's GAP programs, which is the answer to question Q1).

Figure 6.8 (magenta lines) shows significantly more deviations than Figure 6.7 (red lines), even for the "unproblematic" situations of *NoDangle* or *OverDangle* and lonely base-pairs. We suspect further implementation problems in Vienna's software. Providing an alignment, consisting only of the single sequence AUGGCG, `RNAalifold -p -d0 --bppmThreshold 0` finds not a single base-pair. There is no covariation for just one sequence. Thus, we expected to recover the same results with RNAfold, but that is not the case. The sequence AUGGCG with `RNAfold -p -d0 --bppmThreshold 0` returns a base-pair with significant probability, namely for bases $(2, 6) = (\mathtt{U}, \mathtt{G})$.

In conclusion we can say that our Bellman's GAP implementations pass the correctness tests, even better than the template software from the Vienna-Package.

6.3.2 Comparing model variants

In the course of this study, we presented not only one model for RNA secondary structures, but all-together six variants. Models vary by allowing or forbidding the presents of lonely base-pairs and by three ways of dealing with dangling bases. This brief evaluation shall give a broad overview about the impact of the six variants on base-pair probabilities. Since we do not have – and presumably will never have – confirmed biological data at our disposal, we cannot tell which variant performs best.

Distance between two model variants is the mean distance, averaged over all 46 test inputs. Variance between different inputs is quite high (not shown). Figure 6.9 shows the results for single sequence inputs on the left and alignment inputs on the right heatmap. Asymmetry of our definition of distance becomes very obvious here. Results are not crystal clear but two trends are vaguely perceptible:

1. Neglecting additional dangling energies (*NoDangle*) cause more disparity, compared to the other two variants, than between *OverDangle* and *MicroState*. That might be surprising, since the search space of *MicroState* is significantly larger than of the other two.

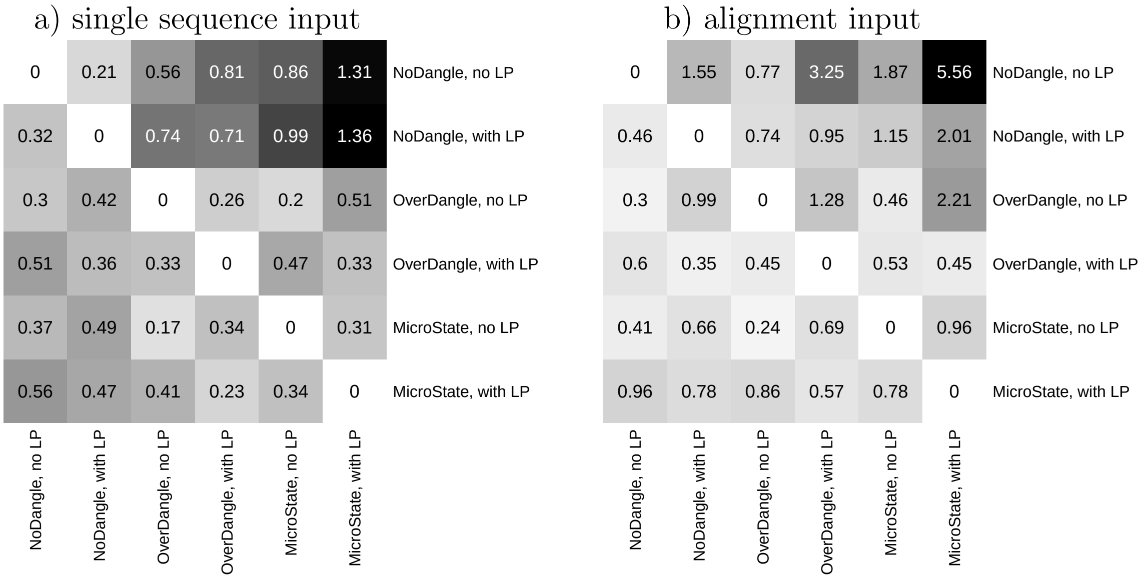
2. Presents or absence of lonely base-pairs seem to have less impact than the dangling variants.

6.3.3 Run-time analysis

Last evaluation is about practical run-times for the template program from the Vienna-Package and our Bellman's GAP compiled binaries.

Single sequence inputs have again been generated randomly. Sequence length grows with a step-size of 5 nucleotides. Test alignments are randomly cut out sub-alignments of Rfam's[7] seed alignment for family `RF01960`, an “Eukaryotic small subunit ribosomal RNA”, with 84 members and 2,438 columns. Concrete sequences and sub-alignments are included in the supplementary file `testInputs.tgz`. Run-times are measured on the usual evaluation environment, see Section 3.3.2.

[7] release 11.0



a) single sequence input

	NoDangle, no LP	NoDangle, with LP	OverDangle, no LP	OverDangle, with LP	MicroState, no LP	MicroState, with LP
NoDangle, no LP	0	0.21	0.56	0.81	0.86	1.31
NoDangle, with LP	0.32	0	0.74	0.71	0.99	1.36
OverDangle, no LP	0.3	0.42	0	0.26	0.2	0.51
OverDangle, with LP	0.51	0.36	0.33	0	0.47	0.33
MicroState, no LP	0.37	0.49	0.17	0.34	0	0.31
MicroState, with LP	0.56	0.47	0.41	0.23	0.34	0

b) alignment input

	NoDangle, no LP	NoDangle, with LP	OverDangle, no LP	OverDangle, with LP	MicroState, no LP	MicroState, with LP
NoDangle, no LP	0	1.55	0.77	3.25	1.87	5.56
NoDangle, with LP	0.46	0	0.74	0.95	1.15	2.01
OverDangle, no LP	0.3	0.99	0	1.28	0.46	2.21
OverDangle, with LP	0.6	0.35	0.45	0	0.53	0.45
MicroState, no LP	0.41	0.66	0.24	0.69	0	0.96
MicroState, with LP	0.96	0.78	0.86	0.57	0.78	0

Figure 6.9: Comparison of all six presented variants of the model for RNA secondary structures.

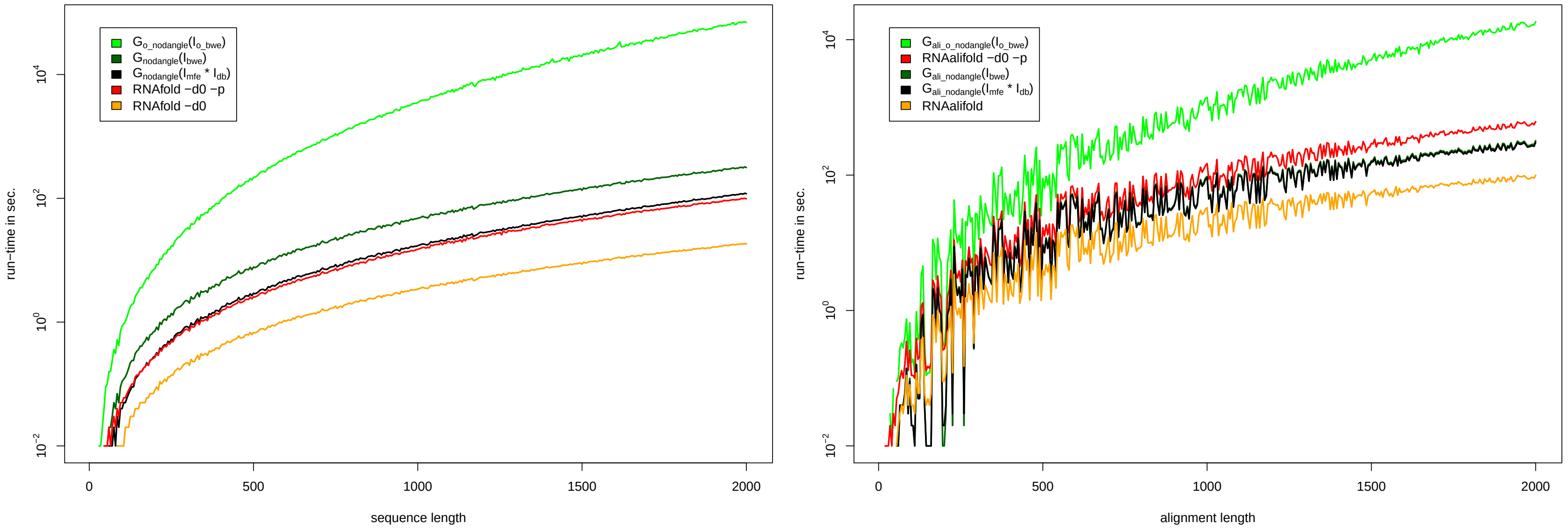


Figure 6.10: Comparison of practical run-times. X-axis is sequence or alignment length, logarithmic scaled y-axis is run-time in seconds. Chosen model variant is *NoDangle* and presents of lonely base-pairs, because this variant causes fewest differences between both software worlds. Left graph shows run-times for single sequence input, right graph for alignment input.

Figure 6.10 confirms several assumptions:

1. Handcrafted Vienna software is significantly faster than our generated binaries. Compare orange (`RNAFOLD-d0`) with black ($\mathcal{G}_{\text{NoDangle}}(\mathcal{I}_{mfe} * \mathcal{I}_{db})$) lines for prediction of "the" single best secondary structure and red (`RNAFOLD-d0 -p`) with light green ($\mathcal{G}_{\text{o\_NoDangle}}(\mathcal{I}_{o\_bwe})$) lines for computation of base-pair probabilities.

2. Calculating base-pair probabilities in the everted style leads to an increased search space, thus a higher run-time: light green ($\mathcal{G}_{\text{o\_NoDangle}}(\mathcal{I}_{o\_bwe})$) line.

3. Computing base-pair probabilities is more costly than MFE prediction, due to exponential scalings. This surplus is a constant factor for the Vienna-Package. For Bellman's GAP, it is only constant if we look at the same search space; compare black ($\mathcal{G}_{\text{NoDangle}}(\mathcal{I}_{mfe} * \mathcal{I}_{db})$) and dark green ($\mathcal{G}_{\text{NoDangle}}(\mathcal{I}_{bwe})$) lines. However, due to the everted style, the search space of $\mathcal{G}_{\text{o\_NoDangle}}(\mathcal{I}_{o\_bwe})$ (light green line) grows faster with the input size than inside-out computations.

4. Asymptotic run-time of RNAalifold is $O(n^2 + n^3)$, compared to RNAfold with $O(n^3)$. The extra n^2 time is for pre-computing covariation for all i, j columns. This cost is higher than the constant factor of exponential scaling for Boltzmann weighted energies. Thus, the aforementioned difference collapses for alignment inputs: dark green and black lines are almost on top of each other.

A bit surprising are the facts that:

1. run-times for alignments are that shaky. Most probable reason is the high variance of the search space size, due to the definition of valid base-pairing positions, see Section 6.2.4. There are huge highly gaped parts in the `RF01960` alignment. A sub-alignment composed of these region will hardly form any base-pair at all, while other sub-alignments show "normal" exponential growth with input length.

2. Vienna software for single sequence input outperformed all Bellman's GAP implementations. That does not hold for alignment inputs. While being significantly faster than its Bellman's GAP counter part RNAalifold, when computing base-pair probabilities, is slower than Bellman's GAP inside-out calculations. That might point to some potential for improving RNAalifold's speed.

6.4 Conclusion

For the concrete example of base-pair probabilities, we could show that it is possible to compute outside information with Bellman's GAP. Naturally, this appears impossible with ADP, but due to the everted style grammar, input duplication and index transformation we could achieve this goal. Thanks to the high abstraction level, we could easily implement variants that have not been introduced to the template Vienna-Package, like a *MicroState* model or consistent absence of lonely base-pairs. Albeit carefully optimized software can significantly outperform Bellman's GAP programs in terms of run-time, later provide sufficient speed for typical problem sizes and tremendously ease the process of rapid prototyping.

Generalization of outside concepts for other ADP problems should be within reach, since they seem to be very mechanistic transformations. Future work may integrate them directly into Bellman's GAP's compiler.

Experimental SHAPE [94] data might be used to measure the agreement of base-pair predictions and biological truth.

7 Covariance Models

Introduction A *covariance model* (CM) [23] is a stochastical tool to quantify homology of an RNA sequence to a *family* of sequences. The family consists of an aligned set of RNA sequences (MSA), which are believed to share the same functionality, shape or other grouping properties, together with one pseudoknot free consensus secondary structure (SS_{cons}). The machinery follows the Bayesian interpretation of probabilities, by updating family independent expert knowledge (priors) with those frequencies observed in the given family (posteriors). Amalgamation of priors and posteriors is called "training" and has to be done just once. Result of the training is a *stochastical context free grammar* (SCFG) [29] – an automaton with a set of states, state-to-state-transition- and state-emission-probabilities. It follows the principles of *Hidden Markov Models* [21], but is more powerful to account for distant but coupled positions, which represent the base-pairs of SS_{cons}.

If we apply the CM to an RNA sequence, we get the probability of the most likely state-path through the automaton, which exactly emits the nucleotides of this RNA. Since this probability tends to be very small, it is related – in terms of a log odds ratio, which provides a bit-score – to some kind of a background model. Finally, the bit-score expresses homology between sequence and family. It is up to the user, to decide if the bit-score suffices to see the input sequence as a new family member.

Reference implementation The INFERNAL software package [65], a product of 20 years of careful software engineering, is *the* reference implementation of CMs. It does not only provide programs for the above described training (CMBUILD) and searching (CMSEARCH) tasks, but e. g. also tools to "calibrate" the CM – basically to provide e- and p- values for the bit-scores – or to align the new member to the existing family. Scoring a sequences s of length n to a family model of length m takes $O(n * m^3)$ time and $O(n * m^2)$ space. Thus, INFERNAL provides a whole set of pre-filtering strategies to mask large regions of s. Remaining sub-sequences of s are scored with a heuristics, called "query dependent banding" [64], which heavily restricts the search space while hopefully retaining the most likely candidate. Furthermore, statistics are enhanced for training by sequence weighting and expectation maximization. Sequence weighting is to adjust for potential sub-grouping of the family members. Expectation maximization should compensate for over-fitting, by finding a suitable trade-off between priors and posteriors. CMSEARCH can be run in glocal (aka "small in large" or "free shift") or local mode. The reported bit-score might be either the mentioned most likely state-path probability (Cocke-Younger-Kasami = CYK algorithm, which is the analogue to the "Viterbi" algorithm in HMMs), or probability sum of all possible state paths ("inside" algorithm; HMM analogue is called "forward" or "backward").

7.1 Three contributions to Covariance Models

Our contribution on CMs is three fold:

1. Section 7.3: The CYK algorithm, which provides the probability of the *single* best state path, is fast to compute, but seems to lack accuracy. Predictions, based on a single (co-)optimal solution, are often of minor quality, compared to holistic considerations. We already saw this effect in Chapter 3, where we compared MFE predictions with synoptic approaches, e. g. CentroidFold. Therefore, Infernal switched from CYK to inside computation, some years ago.

   The inside algorithm sums up probabilities for all state paths. The difference between both algorithms is just the choice function, which is maximization for CYK and summation for inside. Both variants are calculated in log-space to keep numerical stability. While this is no problem for maximization, it requires a back- and forth transformation to correctly sum up two probabilities in log-space. Thus, inside scores are computationally significantly more costly, although they share the same asymptotic class.

   Apart from the two extrema – considering all or just one candidate – we are going to explore two middle ways to compute bit-scores. The idea is inspired by the RNA shape class concept 3.2.7, where a number of (sub-optimal-)solutions are clustered and evaluated as a group.

2. Section 7.4: The second contribution is of technical nature. Counting observed frequencies from the family alignment for posteriors usually requires a huge case distinction, which somehow incorporates the state-model architecture. Special care must be taken to keep the two implementations for counting and scoring of the same architecture in sync.

   We will propose a general training approach for SCFGs, where the case distinction comes for free by using basically the same automaton for training and scoring. In the training phase, two inputs are used for each member of the family: the alignment row and the consensus structure.

3. Section 7.5: Covariance models, as defined by Sean Eddy, do not reward insertions of valid base-pairs or complete sub-structures relative to SS_{cons}. Nevertheless, some families, e. g. tRNA, contain minorities with additional informative substructures, which are crashed by the majority during formation of one consensus secondary structure.

   We introduce a method to construct CMs with ambivalent but compatible structural alternatives and show how they can improved accuracy.

Section 7.2: Integrating our approaches into the highly optimized Infernal software seemed to be impossible in reasonable time. Thus, we decide to re-implement the core functions in Bellman's GAP and use this re-implementation as starting point for our proposed extensions. For this reason, we first of all have to show that we actually replicate the same results as Infernal.

7.2 Faithful CM re-implementation in Bellman's GAP

We will split up the re-implementation of CMs into two steps. First, we build upon the results of the training phase from cmbuild, namely the produced CM files. Thus, divergences from choosing priors, counting posteriors, sequence weighting, expectation maximization and calibration cannot have any effect, since we use the very same log-odds values. Giegerich et al. did the same for Haskell ADP in [30] and termed it "upward compilation of Infernal generated covariance models".

In a second step, we start from scratch, i. e. from the MSA and SS_{cons} and focus on recovering Infernal's guide tree.

7.2.1 Upward compilation in Bellman's GAP

The state architecture of a CM automaton is determined by SS_{cons} and has a very repetitive structure. Each symbol, or pair of symbols for base-pairs, of SS_{cons} yields a *node* for the CM. Since there arise different situations how to align a base from the input sequence to the model, each node consists of several *states*, e. g. a `MatL` node for an unpaired base in the model has states to match the base (`ML`), delete the base from the input (`D`) or to insert an additional base relative to the model (`IL`). Each state has a very specific set of outgoing edges to transit to the next state. A full example for consensus structure `<*<>>` is given in Figure 7.5 – ignore the colors for now.

The following node types are used in CMs: `Root` for the start, `MatP` for a base-pair, `MatL` for an unpaired base left of some other sub-structure, `MatR` the symmetric case for a right unpaired base, `Bif` for structural bifurcations with left `BegL` and right `BegR` parts and finally a set of `End` nodes. Nodes always have the following states and the associated meaning:

`Root`: `S`: jump to the next node.
`IL`: insert a base left of the sub-structure.
`IR`: insert a base right of the sub-structure.

`MatP`: `MP`: match both partners of the base-pair.
`ML`: match only the left position of the base-pair, the right position is an insertion.
`MR`: match only the right position of the base-pair, the left position is an insertion.
`D`: both positions of the base-pair are insertions.
`IL`: insert a base left of the sub-structure.
`IR`: insert a base right of the sub-structure. This state is only available if the node type of the child is not `End`.

`MatL`: `ML`: match the unpaired position, which is left of some sub-structure.
`D`: the unpaired position is an insertion. This state is only available if the node type of the child is not `End`.
`IL`: insert a base left of the sub-structure.

`MatR`: `MR`: match the unpaired position, which is right of some sub-structure.
`D`: the unpaired position is an insertion.

		target node							
		Root	MatP	MatL	MatR	BegL	BegR	Bif	End
source node	Root	∅	{IL,IR,MP,ML,MR,D}	{IL,IR,ML,D}	{IL,IR,MR,D}	∅	∅	{IL,IR,B}	∅
	MatP	∅	{IL,IR,MP,ML,MR,D}	{IL,IR,ML,D}	{IL,IR,MR,D}	∅	∅	{IL,IR,B}	{IL,IR,E}
	MatL	∅	{IL, MP,ML,MR,D}	{IL, ML,D}	{IL, MR,D}	∅	∅	{IL, B}	{IL, E}
	MatR	∅	{ IR,MP,ML,MR,D}	∅	{ IR,MR,D}	∅	∅	{ IR,B}	∅
	BegL	∅	{MP, ML,MR,D}	∅	∅	∅	∅	{ B}	∅
	BegR	∅	{IL, MP,ML,MR,D}	{IL, ML,D}	∅	∅	∅	{IL, B}	∅
	Bif	Bif always transitions to both of its children BegL and BegR.							
	End	∅	∅	∅	∅	∅	∅	∅	∅

Table 7.1: Alternative transitions for a state x to a following state y, if x has node type "source node" and y will have node type "target node". To avoid ambiguous paths, the IL alternative is revoked if x itself is of type IR.

IR: insert a base right to the sub-structure.

Bif: B: jump to children.

BegL: S: jump to the next node.

BegR: S: jump to the next node.
IL: insert a base left to the sub-structure.

End: E: end the path through the CM.

The scheme of state-to-state transitions within the CM automaton is reported in Table 7.1. The presence of a transition depends on the current source node (y-axis), its target node (x-axis) and the current state for a special case: To avoid ambiguous paths through the model, IR to IL transitions are generally forbidden.

The family specific architecture is encoded in the result file of the CMBUILD process. All states of the automaton are serially numbered in the result file. By parsing these files, we upward compile an INFERNAL-like BELLMAN'S GAP grammar, where each state becomes a non-terminal. Outgoing transitions are alternative productions for the same left hand side. From here on, generated – or better: family f specific – ADP components, like grammar or algebras, will be marked by a hat. We obtain a SCFG for family f by applying the generated grammar $\widehat{\mathcal{G}_{infernal}}$ to an generated evaluation algebra $\widehat{\mathcal{I}^{infernal}_{cyk}}$, which multiplies transition- and emission-probabilities gained from the CMBUILD process. Maximizing over all candidates of the search space for input sequence x is the CYK algorithm:

$$\widehat{\mathcal{G}_{infernal}}(\widehat{\mathcal{I}^{infernal}_{cyk}}, x).$$

To check if the BELLMAN'S GAP version is really capable of reproducing CMSEARCH, we performed the following test. For all 2,208 seed CMs in the RFAM 11.0 release, we run CMSEARCH[1], version 1.0.2, with one randomly picked "original" gap-free sequence

[1] used command line call is `cmsearch -g --toponly --cyk --no-qdb --fil-no-hmm --fil-no-qdb --noalign --rna -T -10000`

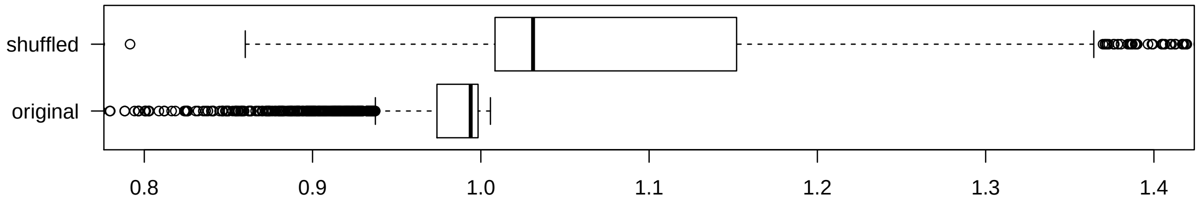


Figure 7.1: Ratio of CYK bit-scores between CMSEARCH and its BELLMAN'S GAP analogue $\widehat{\mathcal{G}_{infernal}(\mathcal{I}_{cyk}^{infernal}}, x)$ for one randomly picked gap-free "original" sequence and its di-nucleotide "shuffled" counterpart for each of the 2,208 families of RFAM 11.0, except for RF00177, RF01800, RF01825, RF01959, RF01960, RF02088, where memory has been exceeded.

from MSA and its di-nucleotide "shuffled"[2] version. Both sequences have also been scored with $\widehat{\mathcal{G}_{infernal}(\mathcal{I}_{cyk}^{infernal}}, x)$. The ratio between INFERNAL and BELLMAN'S GAP CYK scores for all 2,208 sequences are depicted in Figure 7.1.

7.2.2 Approving faithful re-implementation

Results for the "original" inputs for CMSEARCH and BELLMAN'S GAP are nearly identical, with a median ratio of 0.994. The remaining small differences stem from different rounding methods between the `C` and the PERL/`C++` languages. The "shuffled" sequences performed just a bit worse. Their median ratio is 1.031. Aside from these numbers, reported alignments are identical (data not shown). Thus, we conclude that we faithfully re-implemented CMSEARCH in BELLMAN'S GAP.

7.2.3 Determine Infernal's guide-tree to construct identical state architecture.

As stated earlier, the architecture of a CM is primarily shaped by SS_{cons}, namely the series of nodes. These nodes build the so called *guide-tree*. Molding the state architecture for a given guide-tree follows more or less simple rules. Assume some grammar would parse SS_{cons} and produce parse trees / candidates composed only of the algebra functions defined in Signature $\Sigma_{infernal}$, see Table 7.2. We can evaluate every candidate, which complies with $\Sigma_{infernal}$, with $\mathcal{I}_{\mathcal{G}_{infernal}}$ (see Table 7.3) and thus generate a BELLMAN'S GAP version for the state automaton, where `start` will be the axiom.

[2] shuffled by USHUFFLE [46]

Table 7.2: Signature $\Sigma_{infernal}$ for INFERNAL's guide-trees.

$\mathrm{MatP}(\mathcal{A}, \mathcal{S}, \mathcal{A})$	$\mathrm{Bif}(\mathcal{S}, \mathcal{S})$	$\mathrm{root}(\mathcal{S})$
$\mathrm{MatL}(\mathcal{A}, \mathcal{S})$	$\mathrm{BegL}(\mathcal{S})$	$\mathrm{End}()$
$\mathrm{MatR}(\mathcal{S}, \mathcal{A})$	$\mathrm{BegR}(\mathcal{S})$	

Table 7.3: Evaluation algebra $\mathcal{I}_{\mathcal{G}_{infernal}}$ to generate a BELLMAN'S GAP grammar for given SS_{cons}. Function *getStates* executes a look-up of the outgoing states in Table 7.1. Position specific index j is basically the position of the read symbol from SS_{cons}, but for non-consuming functions Bif, BegL and BegR we need to extend the index to start and stop positions of the containing sub-word. Otherwise, uniqueness for indices cannot be guaranteed.

algebra function	$\mathcal{I}_{\mathcal{G}_{infernal}}$
$\mathrm{root}((t, i, p))$	$(\mathrm{root}, j, p$ $\cup\ \texttt{start} \rightarrow \mathrm{begin}(\texttt{r0}, \texttt{S}^j, \texttt{r0})$ $\cup\ \texttt{S}^j \rightarrow \texttt{S}^j\_X^i(X^i)$ $\cup\ \texttt{IL}^j \rightarrow \texttt{IL}^j\_X^i(\texttt{b}, X^i)$ $\cup\ \texttt{IR}^j \rightarrow \texttt{IR}^j\_X^i(X^i, \texttt{b})$ if $t \neq \mathrm{End}$ $\setminus\ \texttt{IR}^j \rightarrow \texttt{IR}^j\_\texttt{IL}^i(\texttt{IL}^i, \texttt{b})$ $\forall\ X \in getStates(\mathrm{root}, t))$
$\mathrm{MatP}(a, (t, i, p), b)$	$(\mathrm{MatP}, j, p$ $\cup\ \texttt{MP}^j \rightarrow \texttt{MP}^j\_X^i(\texttt{b}, X^i, \texttt{b})$ $\cup\ \texttt{ML}^j \rightarrow \texttt{ML}^j\_X^i(\texttt{b}, X^i)$ $\cup\ \texttt{MR}^j \rightarrow \texttt{MR}^j\_X^i(X^i, \texttt{b})$ $\cup\ \texttt{D}^j \rightarrow \texttt{D}^j\_X^i(X^i)$ $\cup\ \texttt{IL}^j \rightarrow \texttt{IL}^j\_X^i(\texttt{b}, X^i)$ $\cup\ \texttt{IR}^j \rightarrow \texttt{IR}^j\_X^i(X^i, \texttt{b})$ if $t \neq \mathrm{End}$ $\setminus\ \texttt{IR}^j \rightarrow \texttt{IR}^j\_\texttt{IL}^i(\texttt{IL}^i, \texttt{b})$ $\forall\ X \in getStates(\mathrm{MatP}, t))$
$\mathrm{MatL}(a, (t, i, p))$	$(\mathrm{MatL}, j, p$ $\cup\ \texttt{ML}^j \rightarrow \texttt{ML}^j\_X^i(\texttt{b}, X^i)$ $\cup\ \texttt{D}^j \rightarrow \texttt{D}^j\_X^i(X^i)$ if $t \neq \mathrm{End}$ $\cup\ \texttt{IL}^j \rightarrow \texttt{IL}^j\_X^i(\texttt{b}, X^i)$ $\forall\ X \in getStates(\mathrm{MatL}, t))$
$\mathrm{MatR}(a, (t, i, p), b)$	$(\mathrm{MatR}, j, p$ $\cup\ \texttt{MR}^j \rightarrow \texttt{MR}^j\_X^i(X^i, \texttt{b})$ $\cup\ \texttt{D}^j \rightarrow \texttt{D}^j\_X^i(X^i)$ $\cup\ \texttt{IR}^j \rightarrow \texttt{IR}^j\_X^i(X^i, \texttt{b})$ if $t \neq \mathrm{End}$

Continued on next page

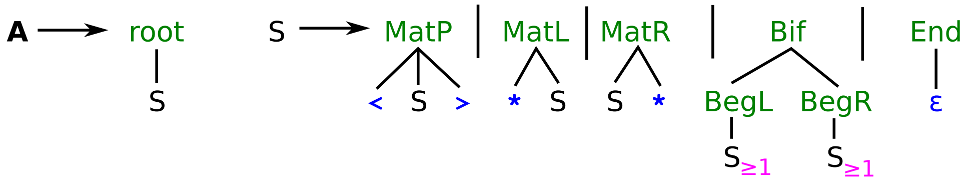


Figure 7.2: INFERNAL's syntactically ambiguous prototype grammar $\mathcal{G}_{infernal}$ to parse SS_{match} and construct a *guide-tree*. Axiom is A.

Continued from previous page

algebra function	$\mathcal{I}_{\mathcal{G}_{infernal}}$
	$\forall\ X \in getStates(\text{MatR}, t))$
$\text{Bif}((t, i, p), (t', i', p'))$	$(\text{Bif}, j, p \cup p'$ $\cup\ \mathtt{B}^j \rightarrow \mathtt{B}^j\_\mathtt{S}^i\_\mathtt{S}^{i'}(\mathtt{S}^i, \mathtt{S}^{i'})$
$\text{BegL}((t, i, p))$	$(\text{BegL}, j, p$ $\cup\ \mathtt{S}^j \rightarrow \mathtt{S}^j\_X^i(X^i)$ $\forall\ X \in getStates(\text{BegL}, t))$
$\text{BegR}((t, i, p))$	$(\text{BegR}, j, p$ $\cup\ \mathtt{S}^j \rightarrow \mathtt{S}^j\_X^i(X^i)$ $\cup\ \mathtt{IL}^j \rightarrow \mathtt{IL}^j\_X^i(\mathtt{b}, X^i)$ $\forall\ X \in getStates(\text{BegR}, t))$
$\text{End}(l)$	$(\text{End}, {}_il, \mathtt{E}^{il} \rightarrow \text{Nil}())$
$\mathcal{S}$	$(string, string, string)$
choice function	id

Unfortunately, the grammar $\mathcal{G}_{infernal}$ (Figure 7.2) used by INFERNAL to parse SS_{cons} is syntactically ambiguous! A given consensus structure will not yield one guide-tree, but several alternatives. The short example structure <*<>> of Figure 7.5, can already be parsed with $\mathcal{G}_{infernal}$ in three different ways. For families of typical length, the number of guide-trees easily reaches the millions. Since INFERNAL is a deterministic program, it must apply some criteria to pick one guide-tree out of the forest. These criteria are formulated in [22]:

> "In general there will be more than one possible guide tree for any given consensus structure. Almost all of this ambiguity is eliminated by three conventions: (1) MATL nodes are always used instead of MATR nodes where possible, for instance in hairpin loops; (2) in describing interior loops, MATL nodes are used before MATR nodes; and (3) BIF nodes are only invoked

> where necessary to explain branching secondary structure stems (as opposed to unnecessarily bifurcating in single stranded sequence). One source of ambiguity remains. In invoking a bifurcation to explain alignment columns $i..j$ by two sub-structures on columns $i..k$ and $k+1..j$, there will be more than one possible choice of k if $i..j$ is a multifurcation loop containing three or more stems. The choice of k impacts the performance of the divide and conquer algorithm; for optimal time performance, we will want bifurcations to split into roughly equal sized alignment problems, so I choose the k that makes $i..k$ and $k+1..j$ as close to the same length as possible."

We can precisely reformulate these objectives by a set of evaluation algebras (Table 7.4). A formal proof of the un-ambiguity is pending, but for each of the 2,208 Rfam families just one guide-tree survived the algebra product:

$$\mathcal{P}_{select} = ((((\mathcal{I}_{noSSbif} * \mathcal{I}_{minRight}) * \mathcal{I}_{lowerBif}) * \mathcal{I}_{pushRight}) * \mathcal{I}_{balance}) * \mathcal{I}_{lighterLeft}.$$

Ordering of algebras is important. Their intentions are explained below:

- $\mathcal{I}_{noSSbif}$
  Bifurcations are only allowed if both children contain at least one base-pair. Thus, there cannot be single stranded regions, produced by bifurcations.

- $\mathcal{I}_{minRight}$
  Whenever an unpaired base can be modelled via a MatL or MatR, MatL is preferred. For example `***` will result in three successive MatLs, not e.g. in two MatRs and one MatL. Thus, this criterion minimizes the use of right bases (MatR).

- $\mathcal{I}_{lowerBif}$
  A Bif is applied as late as possible, thus its parent is as long as possible. For example, `**<><>` will be MatL(*, MatL(*, Bif(BegL(MatP(<, End(), >)), BegR(MatP(<, End(), >))))) but not Bif(BegL(MatL(*, MatL(*, MatP(<, End(), >)))), BegR(MatP(<, End(), >))). Seen as a tree, INFERNAL lowers `Bif` as much as possible.

- $\mathcal{I}_{pushRight}$
  The use of MatR is applied as late as possible. For example, `**()*` will be MatL(*, MatL(*, Bif(BegL(MatP(<, End(), >)), BegR(MatR(END(), *))))) but neither MatL(*, MatR(MatL(*, MatP(<, End(), >)), *)) nor MatR(MatL(*, MatL(*, MatP(<, End(), >))), *). Seen as a tree, MatRs are pushed to the right side of the tree.

- $\mathcal{I}_{balance}$
  Whenever three or more stems have to be connected to one structure there are several ways to do so via Bif. INFERNAL prefers the arrangement that results in a most balanced guide-tree. The *weight* of a stem, or better a sub-tree, is the size of its yield of SS_{cons}, i.e. the number of characters consumed. MatL and MatR each consume one character, while MatP consumes two; thus we can simply count those node-types in the sub-trees.

- $\mathcal{I}_{lighterLeft}$
  Should there be two balanced guide-tree candidates, the one with <u>lighter</u> (or equal weight) <u>left</u> child will be the winner.

Just one step is missing to replicate INFERNAL's state architecture for a given SS_{cons}, namely removing positions of SS_{cons} where the according MSA columns contain more gaps than defined by a threshold. Simply removing these columns might brake base-pairs. Thus, care must be taken to convert the former pairing-partner into an unpaired base, or remove it as well, if its gap contents is too high. Hence, the given SS_{cons} is transformed to SS_{match}, containing only those positions with sufficient sequence content.

7.2.4 Conclusion

The final BELLMAN'S GAP grammar $\widehat{\mathcal{G}_{infernal}}$ for family f with reduced consensus structure SS_{match} can now be generated by a call to the instance $\mathcal{G}_{infernal}(\mathcal{P}_{select} * \mathcal{I}_{\mathcal{G}_{infernal}}, SS_{match})$. To compute homology between a sequence and a family, we can either resort to the log-odds ratios from the CM file as before, or use our own training mechanism, see Section 7.4. For all families of RFAM 11.0, upward compiled grammar – see previous section – and the result of $\mathcal{G}_{infernal}(\mathcal{P}_{select} * \mathcal{I}_{\mathcal{G}_{infernal}}, SS_{match})$ are identical.

Table 7.4: Set of evaluation algebras to pick the same guide-tree as INFERNAL out of the ambiguous search space, spanned by $\mathcal{G}_{infernal}$ for SS_{cons}. $\min_1$. means minimizing over the first element of a pair.

algebra function	$\mathcal{I}_{noSSbif}$	$\mathcal{I}_{minRight}$	$\mathcal{I}_{lowerBif}$	$\mathcal{I}_{pushRight}$	$\mathcal{I}_{balance}$	$\mathcal{I}_{lighterLeft}$
root(x)	x	x	$x+1$	0	x	x
MatP(a,x,b)	1	x	$x+1$	0	(x_1, x_2+2)	(x_1, x_2+2)
MatL(a,x)	x	x	$x+1$	0	(x_1, x_2+1)	(x_1, x_2+1)
MatR(a,x,b)	x	$x+1$	$x+1$	1	(x_1, x_2+1)	(x_1, x_2+1)
Bif(x,y)	$\begin{cases} -1 & \text{if } x<1 \mid\mid y<1 \\ 1 & \text{otherwise} \end{cases}$	$x+y$	0	0	$(\lvert x_2-y_2\rvert, x_2+y_2)$	$\begin{cases} (1, x_2+y_2) & \text{if } x_2>y_2 \\ (0, x_2+y_2) & \text{otherwise} \end{cases}$
BegL(x)	x	x	x	x	x	x
BegR(x)	x	x	x	x	x	x
End(l)	0	0	1	0	$(0,0)$	$(0,0)$
$\mathcal{S}$	int	int	int	int	(int,int)	(int,int)
choice function	min.	min.	max.	min.	$\min_1$.	$\min_1$.

7.3 Alternative semantics

In [30], Giegerich et al. introduced two new interpretations how to understand matching an RNA sequence to an RNA family, namely the *trace semantics* and the *structure semantics*. Together with INFERNAL's *alignment semantics* and the total search space which might be seen as an *inside semantics*, the four semantics form a proper hierarchy: A candidate from the search space is unambiguously described as an alignment. Several alignments have identical trace representations and thus are grouped into a trace-class. Even more alignments, maybe from different trace-classes, follow the same structure and thus form a structure-class. All alignments scattered into trace-classes which seamlessly cluster into structure-classes add up to the single inside-class, which is the complete search space. Growing circles of Figure 7.3 shall visualize the increasing abstraction level. In terms of number of classes we have:

$$\text{alignments} \geq \text{traces} \geq \text{structures} \geq \text{inside} = 1.$$

We will first informally introduce alignment and trace semantics (Sections 7.3.1 and 7.3.2), see why we should switch from $\mathcal{G}_{infernal}$ to a different CM generating grammar (Section 7.3.3), where we also catch up on formal definitions of the semantics and finally show some evaluation results (Section 7.3.4).

7.3.1 Trace semantics

Let us step back to pairwise sequence alignment to introduce the *trace semantics*. The evolutionary perspective on sequence alignments is that both sequences stem from a common ancestor. Matches are conserved bases. Mismatches should be interpreted as point mutations, while gaps can be explained as insertions or deletions of sub-sequences. Thus, alignments should give us rise to the evolutionary history, where both sequences have developed from a common ancestor.

Here are two alignments for sequences $x =$ `ACAGGGGCAC` and $y =$ `ACATTTCAC`:

```
(1)  x:   ACAGGGG---CAC        (2)  x:   ACA---GGGGCAC
     y:   ACA----TTTCAC             y:   ACATTT----CAC
```

Both alignments state that the first three and last three positions are conserved. The only difference is, that alignment (1) first inserts four `G`s into x and than three `T`s into y, while alignment (2) does the same in reverse order. Since we cannot observe in time which event really happened first, we should not favor one alignment over the other. Therefore, both alignments have the same meaning and we should find a common representation as a *trace*, e. g.:

```
x:   ACA[GGGG]CAC
y:   ACA[TTT] CAC
```

The search space of all alignments can be partitioned into trace-classes, similar to shape-classes, which partition the foldingspace of all possible secondary structures. A trace-class might be represented with the above notation, or by one of its alignments. The

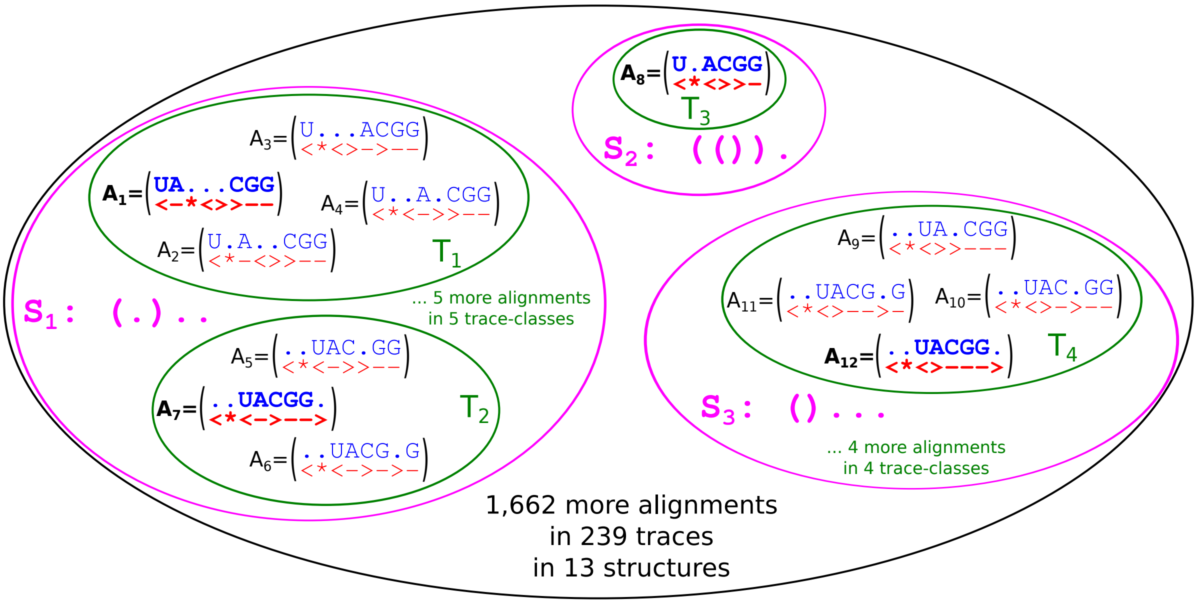


Figure 7.3: The four semantics of CMs: The search space of matching the sequence UACGG to the model <*<>> is populated with 1,683 candidates. Due to the non-ambiguity of CMs regarding *alignments* each candidate is represented by exactly one alignment on the lowest abstraction level (A_1 to A_{1683}, where the last 1,671 are not explicitly shown). Alignments telling the same evolutionary history are grouped on a second level into 252 *trace*-classes (T_1 to T_{252}, only the first four are shown). The alignment which represents its trace-class is given in bold font. On a third level of abstraction, the 252 trace-classes are grouped into 16 *structure*-classes (S_1 to S_{16}). Maximal abstraction is reached by the *inside* semantics, which contains the complete search space.

latter is frequently used, e. g. by choosing the one alignment where insertions always precedes deletions.

For CMs, an RNA sequence is “aligned” to a family model. An example is given in Figure 7.3. Candidates A_1, A_2, A_3 and A_4 are four different options to align sequence UACGG to the model <*<>>. Seen as traces, they all tell the same evolutionary history and thus are grouped into trace-class T_1. Alignment A_1 shall represent trace-class T_1, since it is the only alignment of that group which satisfies the “Insert-before-Deletion” convention. The concept of traces for alignments is carried over to CMs, such that one (e. g. T_3) or several alignments constitutes the same trace. Or in other words: CMs, as defined in INFERNAL, are semantically ambiguous regarding the trace semantics, cf. Section 2.1.3.

7.3.2 Structure semantics

Matching the query RNA sequence to a family model always implies a secondary structure. This structure is not freely folded regarding some general grammar, e. g. $\mathcal{G}_{NoDangle}$, but restricted to the encoded consensus structure of the model SS_{match}. However, due to deleting base-pairs or unpaired bases or inserting additional unpaired bases[3], the structure can be different from SS_{match}. Neglecting base-pairing rules, the sequence UACGG would have 21 secondary structures for a $\mathcal{G}_5$-like grammar. Indeed, the consensus structure <*<>> of the example in Figure 7.3 prohibits formation of the secondary structures .()(), ().(), ()()., ()(.) and (.)(). Thus, only 16 instead of 21 secondary structure can be observed when matching sequence UACGG to the model <*<>>.

Since CMs deal with *structured* RNA sequences, it would be natural to ask for the most probable structure. Again, the structure semantics partitions the search space of the alignments between an RNA sequence and a family model. Furthermore, alignment semantics, trace semantics, structure semantics and ultimately the inside idea form a strict semantic hierarchy of growing abstraction; this is comparable to shape classes of different levels.

7.3.3 Ambiguity compensation

Ambiguity compensation by classification With CMs, we are in a probabilistic setup, where semantic ambiguity corrupts results. The CYK algorithm – $\widehat{\mathcal{G}_{infernal}}(\widehat{\mathcal{I}_{CYK}^{infernal}}, x)$ – aims to find the most likely alignment between a sequence x and the model f. Correct results, i. e. the most likely candidate from the search space equals the most likely alignment, are only guaranteed, iff the search space is semantically non-ambiguous. That this is the case, has been shown by [30] - not in general, but for all Rfam families of that time.

What automatically follows is that the same search space generation, must inevitably be semantically ambiguous regarding trace- and structure- semantics, due to their hierarchy. Thus, CYK cannot provide most likely traces or structures. By replacing the maximization objective function of $\widehat{\mathcal{I}_{CYK}^{infernal}}$ with summation, we gain $\widehat{\mathcal{I}_{inside}^{infernal}}$. Thus, at least inside scores (the most abstract semantics), can correctly be computed by the instance $\widehat{\mathcal{G}_{infernal}}(\widehat{\mathcal{I}_{inside}^{infernal}}, x)$.

Assume for a while, that we are aware of some evaluation algebras $\widehat{\mathcal{I}_{trace}^{infernal}}$ and $\widehat{\mathcal{I}_{structure}^{infernal}}$, which can evaluate any candidate from the search space into its trace- or structure- representation, respectively. By using these algebras for classification, i. e. $\widehat{\mathcal{G}_{infernal}}(\widehat{\mathcal{I}_{trace}^{infernal}} * \widehat{\mathcal{I}_{inside}^{infernal}}, x)$ and $\widehat{\mathcal{G}_{infernal}}(\widehat{\mathcal{I}_{structure}^{infernal}} * \widehat{\mathcal{I}_{inside}^{infernal}}, x)$, summing up probabilities class-wise via *inside* and determining the class with maximum sum, we can compute the desired results – at least in principle. Additional effort for classification depends on the number of classes. Since this number grows exponentially with sequence-

[3] Interestingly, Infernal does not allow insertion of additional base-pairs.

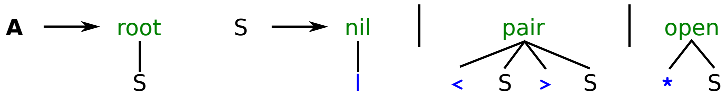


Figure 7.4: Prototype grammar $\mathcal{G}_5$ to parse SS_{match} to generate CMs. Axiom is **A**.

and model- length (some numbers are given in [30]), we created algorithms with exponential run-time. Execution becomes infeasible, even for very small examples. Thus, theoretically elegant classification is of no practical use, here.

Ambiguity compensation by search space modification Besides classification, another path to avoid ambiguity is to directly modify search space generation, i. e. use a different grammar, in such a way that only the class representatives are enumerated. Of course, priors and posteriors must be modified in a way that the probabilistic weight of the former class members now coalesce into the one representative. Unfortunately, we are not aware of such a grammar modification for the structure semantics. And there might not be any efficient method, since ambiguity compensation is NP hard in general [11].

Modifying Infernal's grammar Furthermore, modification of $\widehat{\mathcal{G}_{infernal}}$ for the trace semantics seems to be impossible, too. To provide a semantically non-ambiguous grammar regarding alignments, some of the state transitions are forbidden, e. g. if there are base insertions on both sides of a base-pair, they cannot be inserted in arbitrary order. All left bases must already be inserted, before the first right insertion is allowed. This is realized by having `IL` to `IR` transitions, but no `IR` to `IL` transitions. One might think, that $\widehat{\mathcal{G}_{infernal}}$ can be modified in the same way, i. e. deleting a special set of state transitions, to prune the alignment search space and end up with a trace search space. We will demonstrate with a counter example that this is not true. Assume a model where *SS_match* is `<*<>>` and query sequence is `ACGU`. Here are three out of 681 possible alignments for model and sequence:

```
...ACGU    ..ACG.U    ..AC.GU
<*<>-->    <*<-->>    <*<->->
1234567    1234567    1234567
```

The corresponding paths for the above three alignments are highlighted by color in the complete state graph for this model (see Figure 7.5). While the alignments blue and green are also valid traces, the red one is not. Deletion of the right partner (`ML 13`, alignment position 5) of the inner base-pair `MatP 3` before insertion of a single base (`IR 8`, alignment position 6) on the right of the outer base-pair `MatP 1` violates the “Insert-before-Deletion” convention for traces. For a trace search space the red path must somehow be prohibited. But as you can see, there is no transition uniquely used by the red alignment. The removal of any red transition would also render blue or green

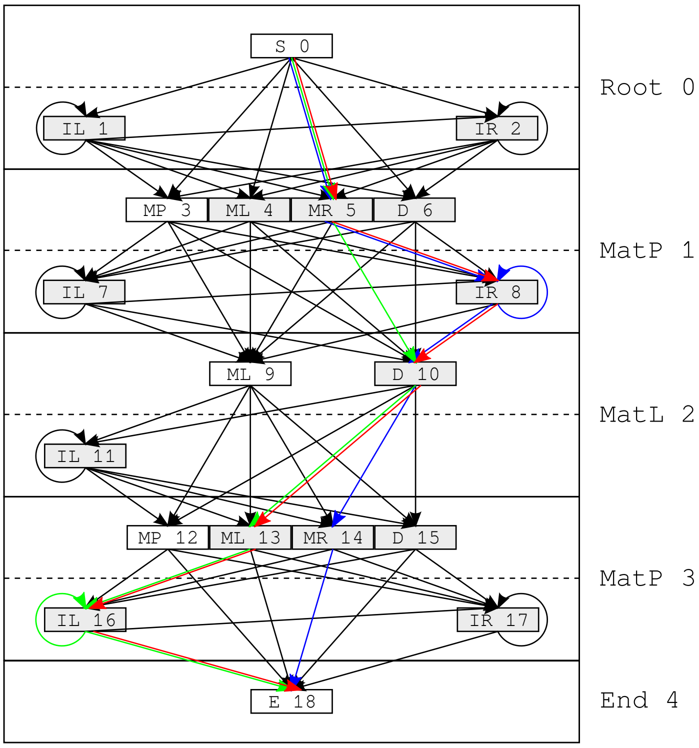


Figure 7.5: Grammar $\widehat{\mathcal{G}_{infernal}}$ for SS_{match} = <*<>>, given in INFERNAL's notation of a state automaton. The three colored paths correspond to three out of 681 possible alignments with sequence ACGU. While blue and green alignments are also valid traces, the red alignment is not, because deletion of the right partner of the inner base-pair (ML 13) happens before insertion of a single base (IR 8).

alignments impossible, which are valid traces and thus must be enumerated by a trace grammar.

The general problem is, that states of a later node must be aware of decisions made in earlier nodes. In the example, node 3 must know about node 1. But within the INFERNAL architecture a node just knows about its direct predecessor node.

Thus, we conclude that search space modification for $\widehat{\mathcal{G}_{infernal}}$ is impossible to get a trace search space.

Modifying $\widehat{\mathcal{G}_5}$ In [30], it was proposed to use the syntactically unambiguous grammar $\mathcal{G}_5$ (see Figure 7.4) to gain the guide-tree. Number of different state types have also been thinned out. The authors furthermore showed that both kinds of CM generation provide identical search spaces. A CM for an RNA family f with consensus structure SS_{match} can now be either constructed by the instance $\mathcal{G}_5(\mathcal{I}_{\mathcal{G}_5}, SS_{match})$ or by $\mathcal{G}_{infernal}(\mathcal{P}_{select} * \mathcal{I}_{\mathcal{G}_{infernal}}, SS_{match})$. While candidates of both search spaces will look different, they represent the same objects and sizes of the search spaces are identical.

Table 7.5: Evaluation algebra $\mathcal{I}_{\mathcal{G}_5}$ to generate a BELLMAN'S GAP grammar for given SS_{match} and, $\mathcal{I}_{\mathcal{G}_{5\_trace}}$ to generate grammars that only enumerate trace representatives. Algebra $\mathcal{I}_{\mathcal{I}_{5\_trace}}$ is a classification algebra to partition the space of alignments into trace classes. The sort of the generated evaluation algebra $\widehat{\mathcal{I}_{5\_trace}}$ will be a two component column vector of strings. The upper string is to indicate the RNA sequence, the lower string reflects the RNA structure. The choice function for $\widehat{\mathcal{I}_{5\_trace}}$ will be "unique". Special operator $\blacktriangleright$ is explained in the main text. Algebra $\mathcal{I}_{\mathcal{I}_{5\_structure}}$ is the formal definition for a classifying structure semantics.

algebra function	$\mathcal{I}_{\mathcal{G}_5}$	$\mathcal{I}_{\mathcal{G}_{5\_trace}}$	$\mathcal{I}_{\mathcal{I}_{5\_trace}}$	$\mathcal{I}_{\mathcal{I}_{5\_structure}}$
$\mathrm{root}((i,p))$	$(i,p$ $\cup\ \mathtt{start} \rightarrow \mathrm{begin}(\mathtt{r0}, \mathtt{S}^i, \mathtt{r0})$ $)$	$(i,p$ $\cup\ \mathtt{start} \rightarrow \mathrm{begin}(\mathtt{r0}, \mathtt{S}^i, \mathtt{r0})$ $)$	$(i,p$ $\cup\ \mathrm{begin}(l,x,r) = x$ $)$	$(i,p$ $\cup\ \mathrm{begin}(l,x,r) = x$ $)$
$\mathrm{nil}(l)$	$(l_j,$ $S^{l_j} \rightarrow \mathtt{INS}^{l_j}(\mathtt{b}, S^{l_j})$ $\cup\ S^{l_j} \rightarrow \mathtt{NIL}^{l_j}(\mathtt{l})$ $)$	$(l_j,$ $S^{l_j} \rightarrow \mathtt{INS}^{l_j}(\mathtt{b}, S^{l_j})$ $\cup\ S^{l_j} \rightarrow T^{l_j}$ $\cup\ T^{l_j} \rightarrow \mathtt{NIL}^{l_j}(\mathtt{l})$ $)$	$(l_j,$ $\mathtt{INS}^{l_j}(a,x) = \langle {a \atop -} \rangle + x$ $\cup\ \mathtt{NIL}^{l_j}(l) = \varepsilon$ $)$	$(l_j,$ $\mathtt{INS}^{l_j}(a,x) = .x$ $\cup\ \mathtt{NIL}^{l_j}(l) = \varepsilon$ $)$
$\mathrm{open}(b,(i,p))$	$(i-1,p$ $\cup\ S^{i-1} \rightarrow \mathtt{INS}^{i-1}(\mathtt{b}, S^{i-1})$ $\cup\ S^{i-1} \rightarrow \mathtt{MAT}^{i-1}(\mathtt{b}, S^{i})$ $\cup\ S^{i-1} \rightarrow \mathtt{DEL}^{i-1}(S^{i})$ $)$	$(i-1,p$ $\cup\ S^{i-1} \rightarrow \mathtt{INS}^{i-1}(\mathtt{b}, S^{i-1})$ $\cup\ S^{i-1} \rightarrow T^{i-1}$ $\cup\ T^{i-1} \rightarrow \mathtt{MAT}^{i-1}(\mathtt{b}, S^{i})$ $\cup\ T^{i-1} \rightarrow \mathtt{DEL}^{i-1}(T^{i})$ $)$	$(i-1,p$ $\mathtt{INS}^{i-1}(a,x) = \langle {a \atop -} \rangle + x$ $\cup\ \mathtt{MAT}^{i-1}(a,x) = \langle {a \atop *} \rangle + x$ $\cup\ \mathtt{DEL}^{i-1}(x) = \langle {\cdot \atop *} \rangle \blacktriangleright x$ $)$	$(i-1,p$ $\mathtt{INS}^{i-1}(a,x) = .x$ $\cup\ \mathtt{MAT}^{i-1}(a,x) = .x$ $\cup\ \mathtt{DEL}^{i-1}(x) = x$ $)$
$\mathrm{pair}(a,(i,p),b,(i',p'))$	$(i-1,p \cup p'$ $\cup\ S^{i-1} \rightarrow \mathtt{INS}^{i-1}(\mathtt{b}, S^{i-1})$ $\cup\ S^{i-1} \rightarrow \mathtt{PK}^{i-1}(\mathtt{b}, S^{i}, \mathtt{b}, S^{i'})$ $\cup\ S^{i-1} \rightarrow \mathtt{Lr}^{i-1}(\mathtt{b}, S^{i}, S^{i'})$ $\cup\ S^{i-1} \rightarrow \mathtt{lR}^{i-1}(S^{i}, \mathtt{b}, S^{i'})$ $\cup\ S^{i-1} \rightarrow \mathtt{bg}^{i-1}(S^{i}, S^{i'})$ $)$	$(i-1,p \cup p'$ $\cup\ S^{i-1} \rightarrow \mathtt{INS}^{i-1}(\mathtt{b}, S^{i-1})$ $\cup\ S^{i-1} \rightarrow T^{i-1}$ $\cup\ T^{i-1} \rightarrow \mathtt{PK}^{i-1}(\mathtt{b}, S^{i}, \mathtt{b}, S^{i'})$ $\cup\ T^{i-1} \rightarrow \mathtt{Lr}^{i-1}(\mathtt{b}, S^{i}, T^{i'})$ $\cup\ T^{i-1} \rightarrow \mathtt{lR}^{i-1}(T^{i}, \mathtt{b}, S^{i'})$ $\cup\ T^{i-1} \rightarrow \mathtt{bg}^{i-1}(T^{i}, T^{i'})$ $)$	$(i-1,p \cup p'$ $\mathtt{INS}^{i-1}(a,x) = \langle {a \atop -} \rangle + x$ $\cup\ \mathtt{PK}^{i-1}(a,x,b,y) = \langle {a \atop <} \rangle + x + \langle {b \atop >} \rangle + y$ $\cup\ \mathtt{Lr}^{i-1}(a,x,y) = \langle {a \atop <} \rangle + x \blacktriangleright \langle {\cdot \atop >} \rangle \blacktriangleright y$ $\cup\ \mathtt{lR}^{i-1}(x,b,y) = \langle {\cdot \atop <} \rangle \blacktriangleright x + \langle {b \atop >} \rangle + y$ $\cup\ \mathtt{bg}^{i-1}(x,y) = \langle {\cdot \atop <} \rangle \blacktriangleright x \blacktriangleright \langle {\cdot \atop >} \rangle \blacktriangleright y$ $)$	$(i-1,p \cup p'$ $\mathtt{INS}^{i-1}(a,x) = .x$ $\cup\ \mathtt{PK}^{i-1}(a,x,b,y) = (x)y$ $\cup\ \mathtt{Lr}^{i-1}(a,x,y) = .xy$ $\cup\ \mathtt{lR}^{i-1}(x,b,y) = x.y$ $\cup\ \mathtt{bg}^{i-1}(x,y) = xy$ $)$
$\mathcal{S}$	$(int, string)$	$(int, string)$	$(int, string)$	$(int, string)$
choice function	id	id	id	id

The big advantage of $\widehat{\mathcal{G}_5}$ over $\widehat{\mathcal{G}_{infernal}}$ is, that we know how to modify it to generate search spaces $\widehat{\mathcal{G}_{5\_trace}}$, which only contains trace representatives. Only $\mathcal{I}_{\mathcal{G}_5}$ has to be replaced by $\mathcal{I}_{\mathcal{G}_{5\_trace}}$ (see Table 7.5):

$$\mathcal{G}_5(\mathcal{I}_{\mathcal{G}_{5\_trace}}, SS_{match}).$$

Now, the S non-terminals are just for incorporating insertions. All other situations are handled by the new T non-terminals. It is no longer possible to re-enter a S non-terminal once after a deletion or match has been issued via a T non-terminal. In other words, if we want to use an insertion, we can only do so prior to any match or deletion.

To validate that $\widehat{\mathcal{G}_{5\_trace}}$ correctly creates trace search spaces, we can compare its size with the number of trace-classes of $\widehat{\mathcal{G}_5}(\widehat{\mathcal{I}_{5\_trace}}, x)$ – at least for small x. The evaluation algebra $\mathcal{I}_{\mathcal{I}_{5\_trace}}$, which generates family specific trace classification algebras is given as the second last column in Table 7.5. Besides using a string tuple as sort, where $+$ should denote string concatenation in each component individually, i. e. $\langle \genfrac{}{}{0pt}{}{a}{b} \rangle + \langle \genfrac{}{}{0pt}{}{c}{d} \rangle = \langle \genfrac{}{}{0pt}{}{a+c}{b+d} \rangle$, the special operator $\blacktriangleright$ is employed. This operator shifts all trailing deletions of the left part behind all leading insertions of the right part. If there are no trailing deletions or no leading insertions $\blacktriangleright$ works as simple concatenation $+$. Replacing every $+$ with $\blacktriangleright$ would yield the same results, but is computationally more expensive. Here is a HASKELL like definition of $\blacktriangleright$:

$$\begin{array}{llll}
(x+d) & \blacktriangleright & \varepsilon & = (x+d) \\
\varepsilon & \blacktriangleright & (i+y) & = (i+y) \\
(x+d) & \blacktriangleright & (i+y) & = \begin{cases} (x \blacktriangleright i) \blacktriangleright (d \blacktriangleright y) & d == \langle \genfrac{}{}{0pt}{}{\beta}{\texttt{-}} \rangle \wedge i == \langle \genfrac{}{}{0pt}{}{\dot{}}{\alpha} \rangle \\ x+d+i+y & \text{otherwise} \end{cases}
\end{array}$$

where $\alpha \in \{\texttt{<}, \texttt{>}, \texttt{*}\}$, i.e. i is an insertion of a model position and $\beta \in \{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{U}\}$, i.e. d is a deletion of a base relative to the model.

Notes on the original definition of $\mathcal{I}_{\mathcal{I}_{5\_trace}}$ in [30] Algebra $\mathcal{I}_{\mathcal{I}_{5\_trace}}$ arose from a similar version in [30]. But we think, that this version suffers from some problems. The authors use a similar, but not identical shift operator $\triangleright$. For example, take a look at their algebra function $\texttt{bg}^{i-1}(x, y) = (((\langle \genfrac{}{}{0pt}{}{\texttt{.}}{\texttt{<}} \rangle \triangleright x) + \langle \genfrac{}{}{0pt}{}{\texttt{.}}{\texttt{>}} \rangle) \triangleright y)$, where we wrote implicit brackets explicitly. The function $\texttt{bg}^{i-1}$ is asymmetric. Leading insertions of x will be moved out of the deleted base-pair, but leading insertions of y cannot enter the deleted base-pair, because of the definition of $\triangleright$, where the left part d is never split. Should y contain leading insertions, they would be moved left over the complete base-pair. Consequently, this asymmetry is also present for $\texttt{Lr}^{i-1}$ and $\texttt{lR}^{i-1}$.

Besides asymmetry, moving an insertion over a base-pair might be also unwanted, because it changes the order of the input RNA sequence. Assume input sequence is AC and SS_{match} would be <**>. One of the candidates of the alignment search space is

$$\texttt{bg}^0\left(\texttt{MAT}^1\left(\texttt{A}, \texttt{DEL}^2\left(\texttt{NIL}^3(\texttt{1})\right)\right), \texttt{INS}^4\left(\texttt{C}, \texttt{NIL}^4(\texttt{1})\right)\right).$$

If we evaluate this candidate with the generated trace algebra, we will face the situation $\langle \genfrac{}{}{0pt}{}{\texttt{.A..}}{\texttt{<**>}} \rangle \triangleright \langle \genfrac{}{}{0pt}{}{\texttt{C}}{\texttt{-}} \rangle$ when applying algebra function $\texttt{bg}^0$. For simplicity, we have already fully evaluated the left side, since only the right $\triangleright$ operation of $\mathtt{bg}^0$ is of interest for this example. The result is $\left\langle \begin{smallmatrix}\mathtt{C.A..}\\ \mathtt{-<**>}\end{smallmatrix} \right\rangle$. As you can see, `A` and `C` are transposed!

Excursion on the inferior run-time of $\widehat{\mathcal{G}_5}$ The authors of [30] have shown that CMs following $\mathcal{G}_5$ are significantly smaller in terms of non-terminals and productions than $\mathcal{G}_{infernal}$. They argued that the decreased size should speed-up run-time substantially. Quite the opposite is true! The tool `multi_rt_all`, shipped with the Bellman's GAP compiler, analyzes the grammar – assuming all non-terminals will be tabulated – and derives asymptotic run-times including constant factors. Table 7.6 exemplifies these findings with the two Rfam families covered in [30] plus – as an extreme case – family RF01960, which has the MSA with most columns in the Rfam 11.0 release. Empirical measurements (rows “real-rt”) with *Random set 1* sequences confirm the theoretical considerations of `multi_rt_all`. Note that even the asymptotic class can be reduced from $O(n^3)$ to $O(n^2)$ for $\widehat{\mathcal{G}^{RF01380}_{infernal}}$, if SS_{match} does not contain bifurcations. The expensive difference between both prototype grammars is that in $\widehat{\mathcal{G}_5}$ stretches of base-pairs, which are essentially for stable RNA structures, always bifurcate – often with empty right parts, except the potential to insert additional bases. Because of these insertions, the yield-size analysis of the Bellman's GAP compiler cannot automatically correct for the expensive sub-word splits. The open question is, can we refine $\mathcal{G}_5$ to avoid unnecessary splits, correctly position all insertions, use less production rules than $\mathcal{G}_{infernal}$ and remain syntactic non-ambiguity?

The tendency to gain speed by introducing more non-terminals can be also observed between $\widehat{\mathcal{G}_5}$ and $\widehat{\mathcal{G}_{5\_trace}}$. But please be reminded, that the search space of $\widehat{\mathcal{G}_{5\_trace}}$ is significantly smaller.

Table 7.6: Effects of different grammar designs for identical search spaces, regarding CMs. *rules* is the number of algebra functions; *NTs* is number of non-terminals; asymptotic runtime has been determined by the tool `multi_rt_all`, shipped with the Bellman's GAP compiler. *real-rt* are real runtime measurements for *Random set 1* sequences, fitted with gnuplot via initial asymp-rt values. RF00163 and RF01380 are amongst the smallest Rfam families with and without structural bifurcations, respectively. RF01960 holds the MSA with most columns in Rfam 11.0. Last row is for the smaller search spaces of all trace representatives.

	family	RF00163	RF01380	RF01960
	$\lvert SS_{match} \rvert$	45	19	1,806
$\widehat{\mathcal{G}_{infernal}}$	rules	622	286	24,482
	NTs	143	61	5,543
	asymp-rt	$2n^3 + 1754n^2 + 18n + 19$	$790n^2 + 18n + 19$	$60n^3 + 66060n^2 + 95$
	real-rt	$-4.5^{-e8}n^3 + 2.7^{-e4}n^2 - 4.1^{-e2}n + 2.3$	$3.8^{-e5}n^2 + 9.4^{-e3}n - 1.2$	–
$\widehat{\mathcal{G}_5}$	rules	152	67	5,870
	NTs	47	21	1,808
	asymp-rt	$168n^3 + 315n^2 + 2$	$84n^3 + 135n^2 + 2$	$5376n^3 + 12667n^2 + 2$
	real-rt	$250^{-e8}n^3 + 7.9^{-e4}n^2 - 30.5$	$6.3^{-e7}n^3 + 76^{-e5}n^2 - 30.8$	–
$\widehat{\mathcal{G}_{5\_trace}}$	rules	198	87	7,676
	NTs	93	41	3,615
	asymp-rt	$105n^3 + 394n^2 + 2$	$54n^3 + 169n^2 + 2$	$3566n^3 + 15386n^2 + 2$
	real-rt	$140^{-e8}n^3 + 5.6^{-e4}n^2 - 20.6$	$3.9^{-e7}n^3 + 51^{-e5}n^2 - 21.1$	–

Table 7.7: Bellman's GAP instances to evaluate performance of the four different semantics of CMs.

Semantics	Bellman's GAP instance
"inside"	$\widehat{\mathcal{G}^f_{5\_trace}}(\widehat{\mathcal{I}_{inside}}, x)$
"structure"	$\widehat{\mathcal{G}^f_{5\_trace}}(\widehat{\mathcal{I}_{structure}} * \widehat{\mathcal{I}_{inside}}, x)$
"trace"	$\widehat{\mathcal{G}^f_{5\_trace}}(\widehat{\mathcal{I}_{inside}}, x)$
"alignment"	$\widehat{\mathcal{G}^f_5}(\widehat{\mathcal{I}_{CYK}}, x)$

7.3.4 Evaluation

To assess performance of the four different semantics, we compiled a set of qualified families, using the same criteria as Nawrocki et al. did for their benchmark Rmark [64]. From Rfam release 10.1 and 23 families of high quality, provided by Manja Marz (private communication and [56, 57, 58, 59]), we selected 148 families with sufficient size, to be split into training- and test- sequences and proper sequence identity, or better – sequence variation. For each family f, grammars $\widehat{\mathcal{G}^f_5}$ and $\widehat{\mathcal{G}^f_{5\_trace}}$, together with according algebras $\widehat{\mathcal{I}^f_{CYK}}$, $\widehat{\mathcal{I}^f_{trace}}$, $\widehat{\mathcal{I}^f_{structure}}$ and $\widehat{\mathcal{I}^f_{inside}}$, where generated and trained with SS^f_{match} and the aligned training sequences.

To gain test sequences with increasing dissimilarity to the training families, we composed the classes, listed in Table 7.8. Altogether, we got 1, 121 sequences.

The Bellman's GAP instances of Table 7.7 have been compiled and run with every test sequence x to compute the required bit-scores. For some semantics, e. g. "inside", we could have used $\widehat{\mathcal{G}^f_5}$ instead of $\widehat{\mathcal{G}^f_{5\_trace}}$ to obtain identical results. We prefer $\widehat{\mathcal{G}^f_{5\_trace}}$, because it requires less run-time, cf. Table 7.6.

Figure 7.6 summarizes the bit-score distribution over the sequence classes and semantics. As expected, the scores drop with growing dissimilarity to the family. Unfortunately, just very few sequences could be analyzed regarding their most probable structure; but this was expected, too. Surprising is the fact that scores from "inside", "trace" and "alignment" semantics seem to agree in most cases. We anticipated that the semantic hierarchy is well reflected in significantly different bit-scores. But in positive cases (left four sequence classes), the single optimal alignment already holds the majority of total available probability mass, i. e. ratio of "alignment" vs. "inside" (see Figure 7.7), and the optimal trace-class most often consists of just this optimal alignment. Even if the optimal alignment leaves some probabilistic space (negative cases), the optimal trace cannot accumulate much more probability mass, due to extremely few class members. Structure classes seem to be larger ("orignalTestseq") and can successfully accumulate sufficient probability mass to indicate homology, where "trace" and "alignment" fail – at least for the very few computable examples.

Table 7.8: Test classes to evaluate performance of the four different semantics of CMs. Sequence s is randomly selected from the set of training sequences.

Class name	Concept
unchanged	Remain s unchanged.
pointmutation	Point mutate 10% of the bases of s.
deletedHairpin	Delete those bases of s which belong to one arbitrary substructure of SS^f_{match} ending in a hairpin. (55 families had to be omitted, since they contain only a single hairpin.)
originalTestseq	Randomly choose one test sequence.
dinuc-shuffled	Shuffle bases of s di-nucleotide wise.
rand-hmm-0.2divlen	Emitting a random sequence with Nawrocki's 15 state HMM [64], with a lengths between 50% and 200% of MSA length.
rand-uniform-0.2divlen	Generating a uniformly distributed sequence of ± 20% of MSA length.
partialSeed1	Randomly select a sub-sequence of s of at least 10% length.

7.3.5 Conclusion on the new semantics

In conclusion, the “trace semantics” does not abstract sufficiently from the “alignment semantics”, while the more promising “structure semantics” is infeasible to compute. Thus, we suggest to keep using the “inside semantics” for now, but want to encourage future work on the “structure semantics”, which seems to have more distinctive power.

7.4 Two Track Counting

For CMs, determining posterior frequencies is just counting how often which nucleotide is in the training sequences in which situation relative to the consensus structure. Aligning a query sequence to a model becomes an optimization problem over many different possibilities, because we do not know the best situation relative to the consensus structure for each nucleotide in advance (called *state path*). But we do know the state path for training. And thus, we have only one candidate in the search space, if we force a (training) sequence into a concrete state path. We can do so by providing the state path as a second input, called *track*, besides the nucleotide sequence itself. The state path SS_{train} is SS_{cons}, where unpaired positions become gaps, and partnering positions become unpaired bases, if these positions are marked as insert-columns, due to too high gap frequencies in MSA.

Mapping a single row of MSA to SS_{train} might produce columns consisting of a gap in sequence *and* model: $\langle \overset{\cdot}{\text{-}} \rangle$. These columns must be removed from both input tracks.

An enumeration algebra $\mathcal{I}_{enum}$ is a generic representation for candidates of the search space, which can automatically be produced by the Bellman's GAP compiler. It

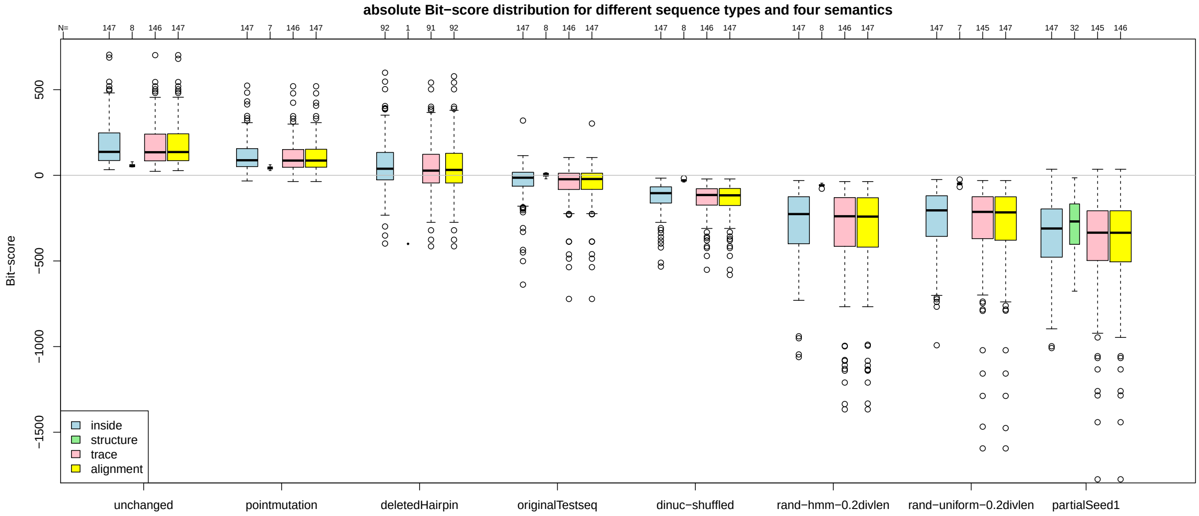


Figure 7.6: Box plots for all bit-scores, obtained by running the 1, 121 sequences with for all four semantics, grouped by sequence classes (see Table 7.8) and semantics (see Table 7.7). Sample size, particularly for the "structure" semantics, sometimes is smaller than the number of available sequences, because execution exceeded available memory.

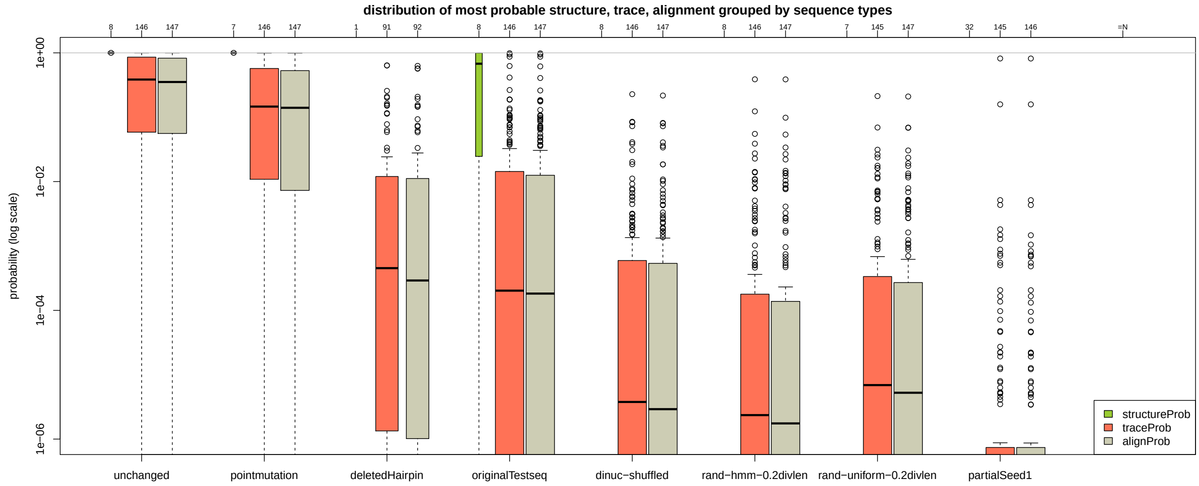


Figure 7.7: Basically the same bit-score distributions as in Figure 7.6, but displayed relative to total probability mass, i. e. "inside" bit-score.

Table 7.9: Evaluation algebra $\mathcal{I}_{\mathcal{G}_{5\_train}}$ to generate a family specific grammar, capable of parsing an RNA sequence and a secondary structure as two tracks, written as 2-component column vectors.

algebra function	$\mathcal{I}_{\mathcal{G}_{5\_train}}$
$\mathrm{root}((i,p))$	$(i, p$ $\cup\ \mathtt{start} \rightarrow \mathtt{S}^{i}$ $)$
$\mathrm{nil}(l)$	$(l_j,$ $S^{l_j} \rightarrow \mathtt{INS}^{l_j}(\langle \substack{\mathtt{b}\\ \mathtt{-}} \rangle, S^{l_j})$ $\cup\ S^{l_j} \rightarrow \mathtt{NIL}^{l_j}(\varepsilon)$ $)$
$\mathrm{open}(b,(i,p))$	$(i-1, p$ $\cup\ S^{i-1} \rightarrow \mathtt{INS}^{i-1}(\langle \substack{\mathtt{b}\\ \mathtt{-}} \rangle, S^{i-1})$ $\cup\ S^{i-1} \rightarrow \mathtt{MAT}^{i-1}(\langle \substack{\mathtt{b}\\ \mathtt{*}} \rangle, S^{i})$ $\cup\ S^{i-1} \rightarrow \mathtt{DEL}^{i-1}(\langle \substack{\mathtt{.}\\ \mathtt{*}} \rangle, S^{i})$ $)$
$\mathrm{pair}(a,(i,p),b,(i',p'))$	$(i-1, p \cup p'$ $\cup\ S^{i-1} \rightarrow \mathtt{INS}^{i-1}(\langle \substack{\mathtt{b}\\ \mathtt{-}} \rangle, S^{i-1})$ $\cup\ S^{i-1} \rightarrow \mathtt{PK}^{i-1}(\langle \substack{\mathtt{b}\\ \mathtt{<}} \rangle S^{i}, \langle \substack{\mathtt{b}\\ \mathtt{>}} \rangle, S^{i'})$ $\cup\ S^{i-1} \rightarrow \mathtt{Lr}^{i-1}(\langle \substack{\mathtt{b}\\ \mathtt{<}} \rangle S^{i}, \langle \substack{\mathtt{.}\\ \mathtt{>}} \rangle, S^{i'})$ $\cup\ S^{i-1} \rightarrow \mathtt{lR}^{i-1}(\langle \substack{\mathtt{.}\\ \mathtt{<}} \rangle S^{i}, \langle \substack{\mathtt{b}\\ \mathtt{>}} \rangle, S^{i'})$ $\cup\ S^{i-1} \rightarrow \mathtt{bg}^{i-1}(\langle \substack{\mathtt{.}\\ \mathtt{<}} \rangle S^{i}, \langle \substack{\mathtt{.}\\ \mathtt{>}} \rangle, S^{i'})$ $)$
$\mathcal{S}$	$(int, string)$
choice function	id

records which algebra functions were called with which parts of the inputs, but gives no hint about the used non-terminals, i. e. the grammar production rules. The trace of the algebra functions is exactly what we need, because they give rise to the path through the CM states and the sub-words of the nucleotide input, i. e. single bases, informs us about the concrete emissions. The sequence of non-terminals would correspond to the node types of the guide-tree, which is not of interest here.

The family specific training grammar $\widehat{\mathcal{G}_{5\_train}}$, which later parses the two tracks of the training sequence and SS_{train}, is generated by the Bellman's GAP instance (see Table 7.9):

$$\widehat{\mathcal{G}_{5\_train}} = \mathcal{G}_5(\mathcal{I}_{\mathcal{G}_{5\_train}}, SS_{match}).$$

By creating enum-representations for all training pairs, consisting of the MSA rows and SS_{train}, and accumulating the numbers of seen algebra functions, we simply get the desired counts. Converting these counts into frequencies is achieved by normalizing the counts by the sum of counts for all alternatives of a non-terminal. In the general case, we after all have to be informed about the non-terminals of the training grammar – in contradiction of what we stated earlier. The special case of $\widehat{\mathcal{G}_{5\_train}}$ can cope without this additional information, due to unambiguous combinations of algebra function names and indices.

Training different semantics Two track counting also supports training for different semantics. If we want to deduce transition and emission counts for e. g. trace semantics, we are faced with the situation that the training data, in form of MSA and SS_{cons}, follows the alignment semantics, i. e. some alignments $A = \langle \, {}^{x}_{SS_{train}} \rangle$, where x is a row of MSA, might not be valid traces, since they violate the "insert-before-delete" convention. These situations can easily resolved by the semantics classification algebra, e. g. $\widehat{\mathcal{I}_{5\_trace}}$. If a run of $T = \widehat{\mathcal{G}_{5\_train}}(\widehat{\mathcal{I}_{5\_trace}}, A)$ reveals that $T \neq A$, we know that A falls into the trace class, which is represented by T. Therefore, we simply re-run $\widehat{\mathcal{G}_{5\_train}}(T)$.

7.5 Ambivalent Covariance Models

The dominating source for RNA family models, and thus the most important use-case for CMs, is Rfam. And in fact, curators of Rfam and developers of Infernal do closely cooperate. Infernal's CMs require all family member sequences to fold into *one* shared structure with only small deviations to gain good bit-scores. However, the grouping criteria of Rfam is different[4]:

> The ideal basis for a new family is an RNA element that has some known functional classification, is evolutionarily conserved, and has evidence for a secondary structure.

Rfam does not strictly insists on a single common structure. For example, take the tRNA family (RF00005, Rfam release 10.1). It is known that a minority of the tRNA structures form a stabilizing "variable stem-loop" in addition to the classical clover leaf structure. The WikipediA article on tRNA, which Rfam uses to explain the family, does not fail to point to this fact. By a simple sequence length comparison, we found that 147 out of the 967 seed members constitute this minority. We also constructed a plausible consensus structure for this minority by aligning (RNAforester [39]) individual structure predictions from tRNAscan-SE [52].

Replacing original SS_{cons} with our variable stem-loop enriched version cannot yield a different CM, because the majority of 820 members have gaps at the variable stem-loop positions and therefore model them as insertions, i. e. the introduced variable stem-loop sub-structure is taken out off SS_{match}. Informative covariance from base-pairs of this stem cannot be captured. On the other hand, enforcing $SS_{cons} = SS_{match}$ by increasing allowed gap ratio, would impose large deletion costs on the 820 members, when aligning to the new model.

With the introduction of Ambivalent Covariance Models (aCMs), we hope to open up a route to get over these shortcomings. Basically, an aCM is a CM constructed

[4] from `http://rfam.sanger.ac.uk/help`, seen on March 7th, 2014

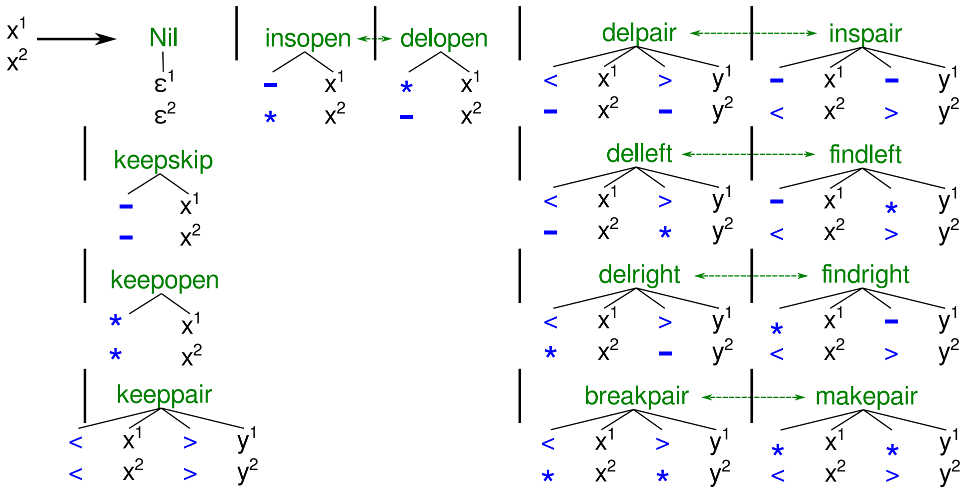


Figure 7.8: Two-track ADP grammar $\mathcal{G}_{ali\_SS}$ to align two consensus structures. Please note, that non-terminal x is composed of two tracks, x^1 and x^2. Green dashed arrows are just to indicate symmetric operations.

by multiple guide-trees, i. e. containing multiple consensus sequences SS^i_{cons}. These consensus structures should not be arbitrary structures, but be somehow compatible, i. e. can form one multiple alignment. This requires that all consensus structures have the same length, which can simply be achieved by introducing gaps. More thoughts have to be spend on defining what kinds of alignment columns should be allowed. For example, we think that the two consensus structures $\left\langle \begin{smallmatrix}\texttt{<<-*>**>}\\ \texttt{<<-*>-->}\end{smallmatrix} \right\rangle$ can explain a common evolution up to the point where two additional unpaired bases are inserted into the upper consensus. The double gaped column $\left\langle \begin{smallmatrix}\texttt{-}\\ \texttt{-}\end{smallmatrix} \right\rangle$ should be allowed to enable inclusion of rare sequences into MSA, which hold insertions relative to both consensus structures.

But we disallow alternative base-pair partners, like $\left\langle \begin{smallmatrix}\texttt{<<**>>}\\ \texttt{<*<*>>}\end{smallmatrix} \right\rangle$, since we cannot see a convincing evolutionary explanation. For example, base-pair slippage should better be modeled via insertions. A formal definition of valid pairwise consensus structure alignments is given in the form of the ADP grammar $\mathcal{G}_{ali\_SS}$ in Figure 7.8. Multiple consensus structures $SS^1_{cons}, \dots, SS^n_{cons}$ can be checked for compatibility by successively aligning the pairs $(SS^1_{cons}, SS^i_{cons})$ $\forall i > 1$.

If all consensus structures are compatible, we need to compute their individual guide-trees and fuse them into one multiple guide-tree. This multiple guide-tree should be the smallest common tree, to keep the aCM– and thus run-time of the aCM– as small as possible. During fusion, we also have to make sure that no unseen combinations become possible, e. g. $\left\langle \begin{smallmatrix}\texttt{<<*>*-->}\\ \texttt{<---*<>>}\end{smallmatrix} \right\rangle$ shall not impose model structure `<---*-->`. Furthermore, for training the aCM we have to reconsider sequence weighting, transition probabilities for entering alternative paths and priors. Counting can be accomplished without much changes via the “Two Track Counting” of Section 7.4.

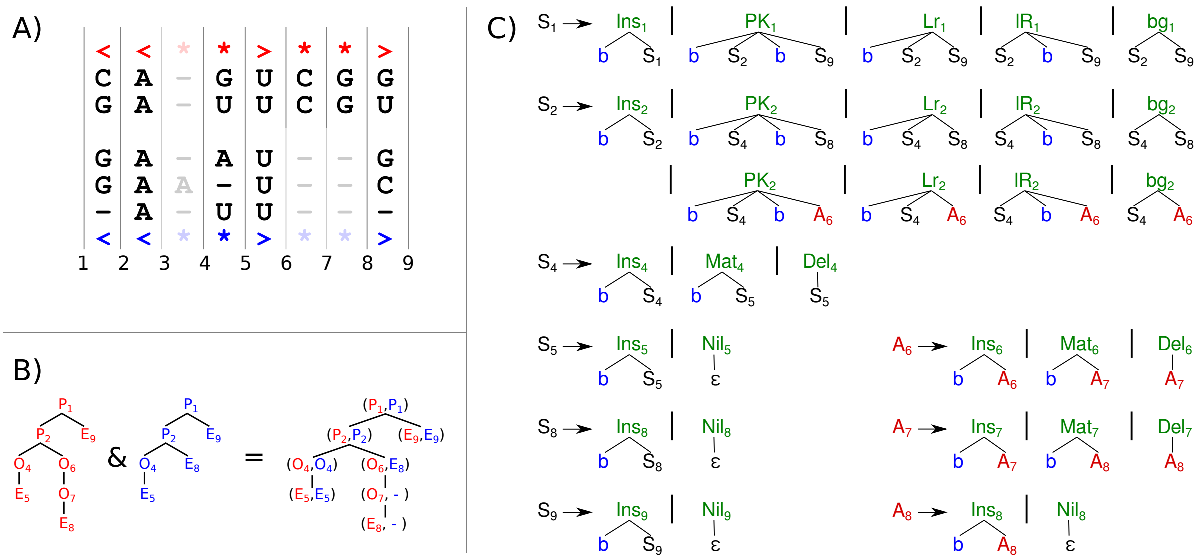

A)
< < * * > * * >
C A – G U C G G
G A – U U C G U
G A – A U – – G
G A A – U – – C
– A – U U – – –
< < * * > * * >
1 2 3 4 5 6 7 8 9
B)
P1 P2 E9 O4 O6 E5 O7 E8
&
P1 P2 E9 O4 E8 E5
=
(P1,P1) (P2,P2) (E9,E9) (O4,O4) (O6,E8) (E5,E5) (O7, -) (E8, -)
C)
S1 → Ins1 (b S1) | PK1 (b S2 b S9) | Lr1 (b S2 S9) | IR1 (S2 b S9) | bg1 (S2 S9)
S2 → Ins2 (b S2) | PK2 (b S4 b S8) | Lr2 (b S4 S8) | IR2 (S4 b S8) | bg2 (S4 S8)
| PK2 (b S4 b A6) | Lr2 (b S4 A6) | IR2 (S4 b A6) | bg2 (S4 A6)
S4 → Ins4 (b S4) | Mat4 (b S5) | Del4 (S5)
S5 → Ins5 (b S5) | Nil5 (ε)
S8 → Ins8 (b S8) | Nil8 (ε)
S9 → Ins9 (b S9) | Nil9 (ε)
A6 → Ins6 (b A6) | Mat6 (b A7) | Del6 (A7)
A7 → Ins7 (b A7) | Mat7 (b A8) | Del7 (A8)
A8 → Ins8 (b A8) | Nil8 (ε)

Figure 7.9: Toy example for an ACM with two consensus structures. **Part A)** *MSA* with the two consensus structures `<<-*>**>` and `<<-*>-->`. *MSA* columns with borders (3,4) in both and (6,8) for the blue consensus structure are modelled as insertions. **Part B)** Individual- and smallest common guide-tree for the two consensus structures. **Part C)** Generated ACM grammar for an ambivalent model.

Generation of a model specific BELLMAN'S GAP grammar with n consensus structures $\widehat{\mathcal{G}_5^{(1,\dots,n)}}$ is similar to normal generation for CMs, except that we operate on one multiple guide-tree. Some effort has to be spend on redirecting to the correct position-specialized non-terminals. We point the interested reader to the HASKELL source code in the fold-grammars repository, for reasons of space: `http://bibiserv.cebitec.uni-bielefeld.de/fold-grammars`. Instead of a verbose algorithm description, we show the exemplary grammar for the consensus structures $\left\langle \begin{smallmatrix}\texttt{<<-*>**>}\\ \texttt{<<-*>-->}\end{smallmatrix} \right\rangle$ in Figure 7.9. It basically follows the $\mathcal{G}_5$ prototype grammar. For nodes with different sub-trees for the consensus structures in the multiple guide-tree, additional production rules are added to the according non-terminal. (Second row for S_2 in the example.) This might also imply generation of further non-terminals (A_i in the example).

7.5.1 Evaluation

Are ACMs really necessary to find homologs to RNA families with more than one consensus structure? What if we rigorously split this family into one sub-family for each consensus structure and use all of these models to score potential candidates? We can detect homology if just one of these models gains high bit-scores. This is true!

However, since ACMs contain more information, i. e. common sub-structures stronger deviate from the background model and predetermined sub-structures do not get penalized, discriminatory power can be increased. Thus, even those true candidates can get a positive score for aCMs, which falls below the threshold of the individual CMs.

We show this effect for two examples: tRNA in Figure 7.10 and the spliceosomal U5 RNA in Figure 7.11.

tRNA The first column ("model: all default") of Figure 7.10 contains CYK bit-scores for all seed sequences, matched against the original CM, i. e. using the complete MSA and the default (green) SS_cons. The blue box-plot is for all 967 scores, the green one contains only scores for those sequences following the "default" SS_{cons}, the orange one for sequences including the variable stem-loop ("varloop"). As expected, "varloop" sequences get penalized, reducing also the median for all sequences. Median score for the original CM is ≈ 52.04 bits (brownish horizontal line).

For the second column ("model: all varloop"), we constructed a CM for the complete MSA, but this time with a SS_{cons} including the variable stem-loop. Since SS_{cons} is transformed into the same SS_{match} as before, producing an identical CM, results do not change.

Different SS_{match} can only be achieved by also using different parts of the MSA. Column "model: single default" visualizes scores for a CM, which is constructed by the default SS_{cons} and only by those rows of the MSA that belong to it. Again, not much changes, besides the fact that "varloop" sequences perform even worse.

Good scores for "varloop" sequences can be obtained, if we use a CM constructed only by the "varloop" consensus and MSA rows (column "model: single varloop"). Unfortunately, performance for the majority of the sequences nosedive.

Should we be able to a priori use the correct sub-model for each sequence, we would gain the theoretical mean score of ≈ 54.95 bits (magenta horizontal line). Splitting the family into two separate sub-families seems to be worth it.

Our proposed aCMs can even do better. As you can see from the last column ("model: aCM"), not only the scores for the "varloop" sequences are lifted to a reasonable amount, also the "default" sequences benefit from this information enrichment. aCMs mean score is ≈ 55.88 bits (purple horizontal line) and thus ≈ 7% better than the median from the original CM. Bit scores of negative sequences, do not change significantly (data not shown).

U5 spliceosomal RNA The U5 spliceosomal RNA is a widespread family. Its concrete secondary structure changes over the different taxons. We used the original alignment from [57][5], identified six taxons, grouped the sequences and defined according consensus structures. Figure 7.11 is the result of the same kind of analysis as in Figure 7.10; negative, i. e. di-nucleotide shuffled sequences, are included here. In conclusion, the mean bit-score could be increased by 27%, compared to the original CM, while bit-scores for negative sequences remain low, thus improving discriminatory power.

[5] www.bioinf.uni-leipzig.de/Publications/SUPPLEMENTS/08-001/ALIGNMENTS/ALL.U5.stk

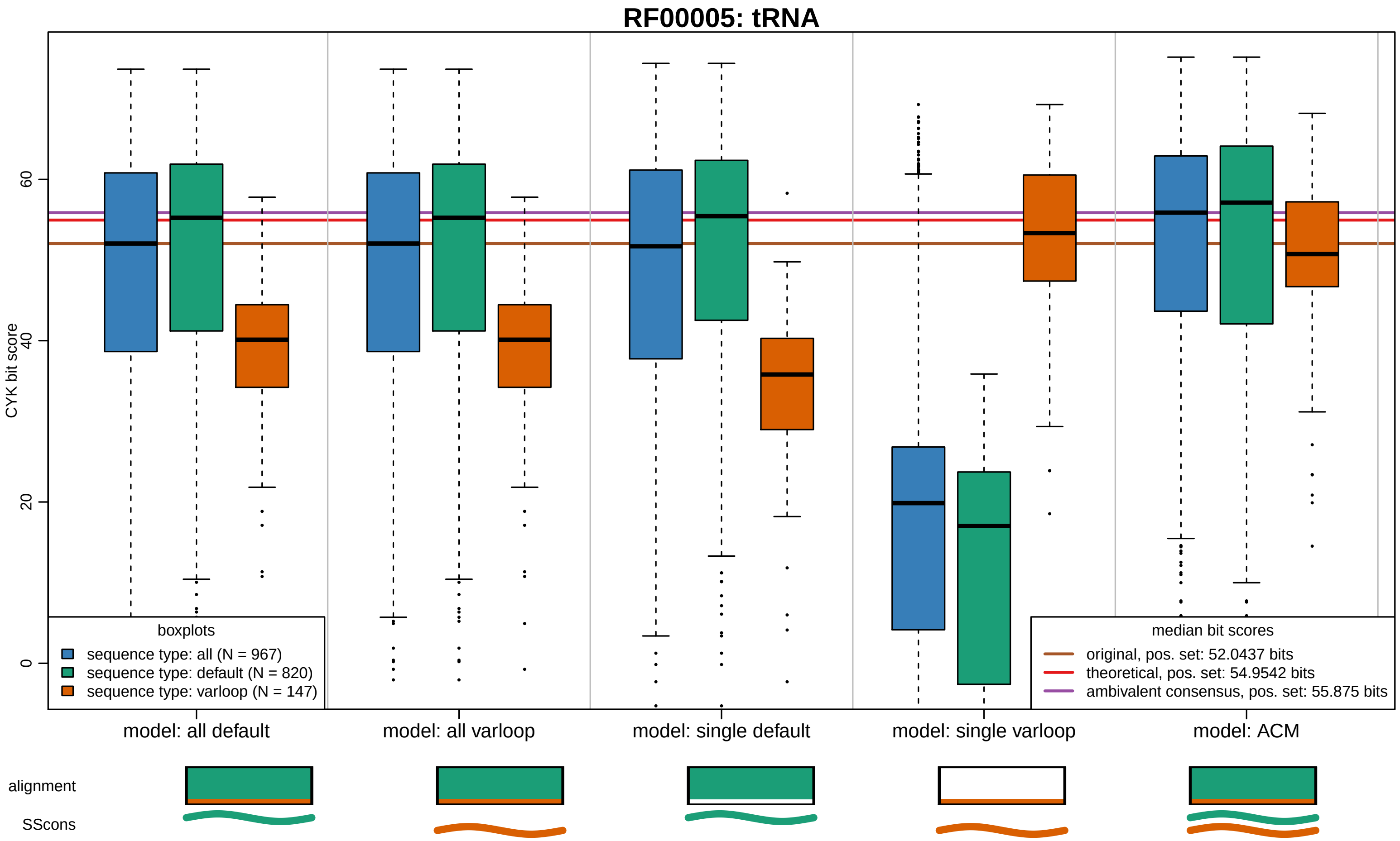


Figure 7.10: Evaluation of different ways to construct individual CMs for sub-groups for the tRNA family, compared with one ACM for both sub-groups. Detailed explanations are in the main text.

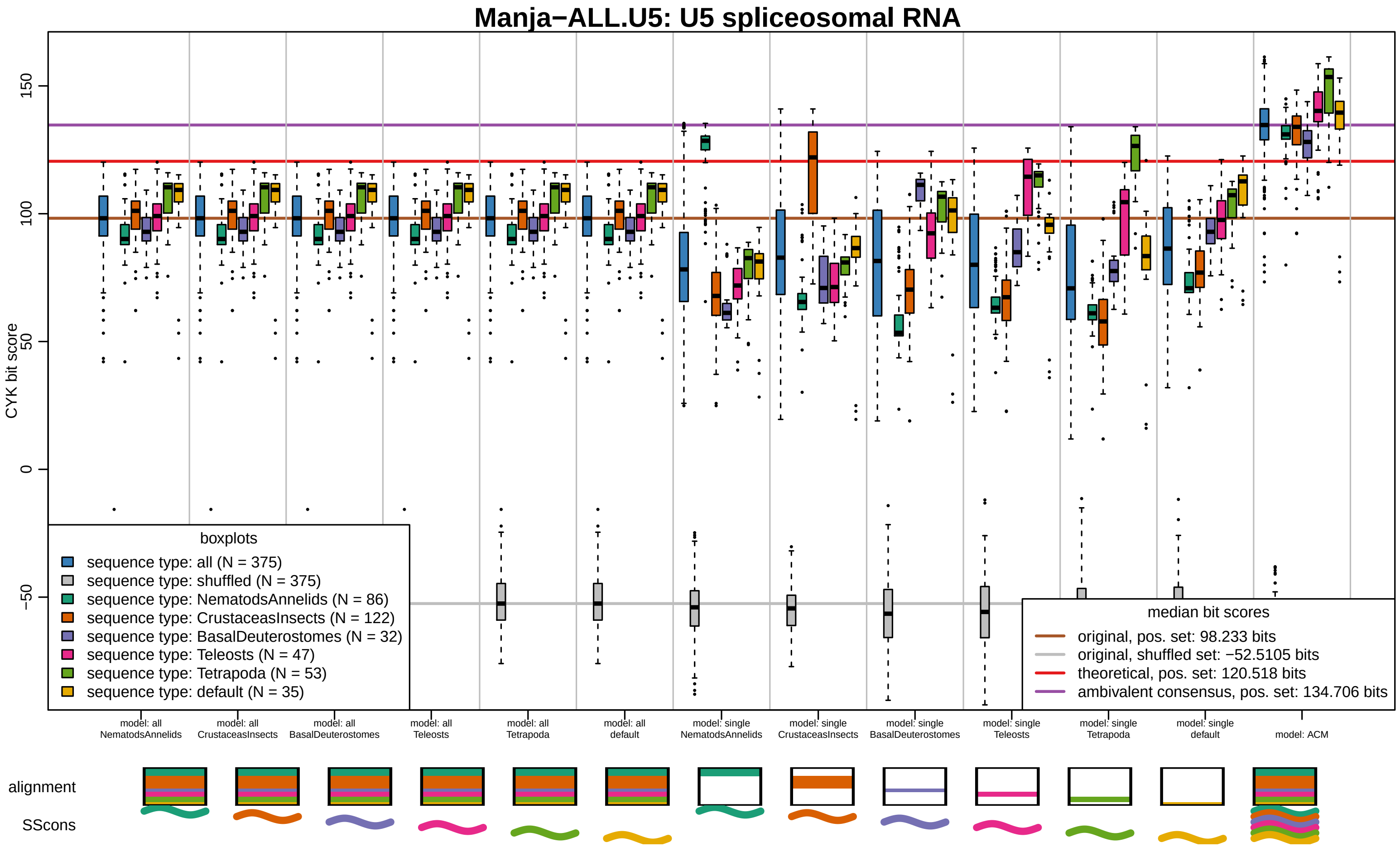


Figure 7.11: Evaluation of different ways to construct individual CMs for sub-groups for the U5 family, compared with one aCM for all six sub-groups.

7.5.2 Conclusion

The Rfam database contains several families that cross the limitations of a single consensus structure. The new concept of an Rfam "clan" is a set of closely related families. It seems to be a workaround to cope with several consensus structures within one former family, which is now torn apart. Our evaluation shows that a segmentation of a family corrupts homology analysis. We suggest extending covariance models to hold multiple consensus structures to keep up with biological needs. Adapting Infernal's innermost mechanisms to ambivalent structures seems to be worth the effort.

8 Bibliography

[1] Tatsuya Akutsu. Dynamic programming algorithms for RNA secondary structure prediction with pseudoknots. *Discrete Appl. Math.*, 104(1–3):45–62, 2000.

[2] Mirela S. Andronescu, Vera Bereg, Holger H. Hoos, and Anne E. Condon. RNA STRAND: The RNA Secondary Structure and Statistical Analysis Database. *BMC Bioinformatics*, 9(1):340, 2008.

[3] Mirela S. Andronescu, Anne E. Condon, Holger H. Hoos, David H. Mathews, and Kevin P. Murphy. Efficient parameter estimation for RNA secondary structure prediction. *Bioinformatics*, 23(13):i19–28, 2007.

[4] Mirela S. Andronescu, Anne E. Condon, Holger H. Hoos, David H. Mathews, and Kevin P. Murphy. Computational approaches for RNA energy parameter estimation. *RNA*, 16(12):2304–2318, 2010.

[5] Mirela S. Andronescu, Cristina Pop, and Anne E. Condon. Improved free energy parameters for RNA pseudoknotted secondary structure prediction. *RNA*, 16(1):26–42, 2010.

[6] Stanislav Bellaousov and David H. Mathews. ProbKnot: Fast prediction of RNA secondary structure including pseudoknots. *RNA*, 16(10):1870–1880, 2010.

[7] Richard Bellmann. Dynamic Programming. *Princeton, NJ: Princeton University Press*, 1957.

[8] Eugene Berezikov, Geert van Tetering, Mark Verheul, Jose van de Belt, Linda van Laake, Joost Vos, Robert Verloop, Marc van de Wetering, Victor Guryev, Shuji Takada, Anton Jan van Zonneveld, Hiroyuki Mano, Ronald Plasterk, and Edwin Cuppen. Many novel mammalian microRNA candidates identified by extensive cloning and RAKE analysis. *Genome Res*, 16(10):1289–1298, 2006.

[9] Stephan Bernhart, Ivo L. Hofacker, Sebastian Will, Andreas Gruber, and Peter F. Stadler. RNAalifold: improved consensus structure prediction for RNA alignments. *BMC Bioinformatics*, 9(1):474, 2008.

[10] Philip N. Borer, Barbara Dengler, Jr. Ignacio Tinoco, and Olke C. Uhlenbeck. Stability of ribonucleic acid double-stranded helices. *Journal of Molecular Biology*, 86:843–853, 1974.

[11] Broňa Brejová, Daniel G. Brown, and Tomáš Vinař. The most probable annotation problem in HMMs and its application to bioinformatics. *Journal of Computer and System Sciences*, 73(7):1060–1077, 2007.

[12] Mark E. Burkard, Ryszard Kierzek, and Douglas H. Turner. Thermodynamics of unpaired terminal nucleotides on short RNA helixes correlates with stacking at helix termini in larger RNAs. *Journal of Molecular Biology*, 290(5):967–982, 1999.

[13] Chi Yu Chan, Charles E. Lawrence, and Ye Ding. Structure clustering features on the Sfold Web server. *Bioinformatics*, 21(20):3926–3928, 2005.

[14] KY Chang and Jr. Ignacio Tinoco. Characterization of a "kissing" hairpin complex derived from the human immunodeficiency virus genome. *Proceedings of the National Academy of Sciences of the United States of America*, 91(18):8705–8709, 1994.

[15] Ho-Lin Chen, Anne E. Condon, and Hosna Jabbari. An $O(n^5)$ Algorithm for MFE Prediction of Kissing Hairpins and 4-Chains in Nucleic Acids. *Journal of Computational Biology*, 16(6):803–815, 2009.

[16] Peter Clote, Fabrizio Ferre, Evangelos Kranakis, and Danny Krizanc. Structural RNA has lower folding energy than random RNA of the same dinucleotide frequency. *RNA*, 11(5):578–591, 2005.

[17] Anne E. Condon and Hosna Jabbari. Computational prediction of nucleic acid secondary structure: Methods, applications, and challenges. *Theoretical Computer Science*, 410(4–5):294–301, 2009.

[18] Ye Ding and Charles E. Lawrence. A statistical sampling algorithm for RNA secondary structure prediction. *Nucleic Acids Research*, 31(24):7280–7301, 2003.

[19] Chuong B. Do, Daniel A. Woods, and Serafim Batzoglou. CONTRAfold: RNA secondary structure prediction without physics-based models. *Bioinformatics*, 22(14):e90–98, 2006.

[20] Robin Dowell and Sean R. Eddy. Evaluation of several lightweight stochastic context-free grammars for RNA secondary structure prediction. *BMC Bioinformatics*, 5(1):71, 2004.

[21] Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids.* Cambridge University Press, July 2002.

[22] Sean R. Eddy. A memory-efficient dynamic programming algorithm for optimal alignment of a sequence to an RNA secondary structure. *BMC Bioinformatics*, 3(1):18, 2002.

[23] Sean R. Eddy and Richard Durbin. RNA sequence analysis using covariance models. *Nucleic Acids Research*, 22(11):2079–2088, 1994.

[24] Andrew Fire, SiQun Xu, Mary K. Montgomery, Steven A. Kostas, Samuel E. Driver, and Craig C. Mello. Potent and specific genetic interference by double-stranded RNA in Caenorhabditis elegans. *Nature*, 391(6669):806–811, Feb 1998.

[25] Paul P. Gardner and Robert Giegerich. A comprehensive comparison of comparative RNA structure prediction approaches. *BMC Bioinformatics*, 5(1):140, 2004.

[26] Costin M. Gherghe, Zahra Shajani, Kevin A. Wilkinson, Gabriele Varani, and Kevin M. Weeks. Strong correlation between SHAPE chemistry and the generalized NMR order parameter (S2) in RNA. *J Am Chem Soc.*, 130(37):12244–5, 2008.

[27] Robert Giegerich. A systematic approach to dynamic programming in bioinformatics. *Bioinformatics*, 16(8):665–677, 2000.

[28] Robert Giegerich. Explaining and Controlling Ambiguity in Dynamic Programming. In *Proc. Combinatorial Pattern Matching*, volume 1848 of *Springer Lecture Notes in Computer Science*, pages 46–59. Springer, 2000.

[29] Robert Giegerich. Introduction to Stochastic Context Free Grammars. In Jan Gorodkin and Walter L Ruzzo, editors, *RNA Sequence, Structure and Function : Computational and Bioinformatic Methods*. Springer, 2014.

[30] Robert Giegerich and Christian Höner zu Siederdissen. Semantics and Ambiguity of Stochastic RNA Family Models. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 8:499–516, 2011.

[31] Robert Giegerich, Carsten Meyer, and Peter Steffen. A discipline of dynamic programming over sequence data. *Science of Computer Programming*, 51(3):215–263, June 2004.

[32] Robert Giegerich and Georg Sauthoff. Yield grammar analysis in the Bellman's GAP compiler. In *Proceedings of the Eleventh Workshop on Language Descriptions, Tools and Applications*, LDTA '11. ACM, 2011.

[33] Robert Giegerich and Peter Steffen. Challenges in the compilation of a domain specific language for dynamic programming. In *Proceedings of the 2006 ACM symposium on Applied computing*, SAC '06, pages 1603–1609, New York, NY, USA, 2006. ACM.

[34] Robert Giegerich, Björn Voß, and Marc Rehmsmeier. Abstract shapes of RNA. *Nucleic Acids Research*, 32(16):4843–4851, 2004.

[35] Walter Gilbert. Origin of life: The RNA world. *Nature*, 319(6055):618–618, Feb 1986.

[36] Alan Gillett, Petra Bergman, Roham Parsa, Andreas Bremges, Robert Giegerich, and Maja Jagodic. A Silent Exonic SNP in *Kdm3a* Affects Nucleic Acids Structure but Does Not Regulate Experimental Autoimmune Encephalomyelitis. *PLoS ONE*, 8(12):e81912, 12 2013.

[37] Michiaki Hamada, Hisanori Kiryu, Kengo Sato, Toutai Mituyama, and Kiyoshi Asai. Prediction of RNA secondary structure using generalized centroid estimators. *Bioinformatics*, 25(4):465–473, February 2009.

[38] J. Herold and S.G. Siddell. An 'elaborated' pseudoknot is required for high frequency frameshifting during translation of HCV 229E polymerase mRNA. *Nucleic Acids Research*, 21(25):5838–5842, 1993.

[39] Matthias Höchsmann, Björn Voß, and Robert Giegerich. Pure Multiple RNA Secondary Structure Alignments: A Progressive Profile Approach. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1(1):53–62, 2004.

[40] Ivo L. Hofacker, Walter Fontana, Peter F. Stadler, Sebastian L. Bonhoeffer, Manfred Tacker, and Peter Schuster. Fast Folding and Comparison of RNA Secondary Structures. *Monatsh. Chem.*, 125:167–188, 1994.

[41] Christian Höner zu Siederdissen. Sneaking around concatMap: efficient combinators for dynamic programming. *SIGPLAN Not.*, 47(9):215–226, September 2012.

[42] Philippe Horvath and Rodolphe Barrangou. CRISPR/Cas, the immune system of bacteria and archaea. *Science*, 327(5962):167–170, 2010.

[43] Stefan Janssen and Robert Giegerich. Faster computation of exact RNA shape probabilities. *Bioinformatics*, 26(5):632–639, 2010.

[44] Stefan Janssen, Jens Reeder, and Robert Giegerich. Shape based indexing for faster search of RNA family databases. *BMC Bioinformatics*, 9:131+, February 2008.

[45] Stefan Janssen, Christian Schudoma, Gerhard Steger, and Robert Giegerich. Lost in folding space? Comparing four variants of the thermodynamic model for RNA secondary structure prediction. *BMC Bioinformatics*, 12(1):429, 2011.

[46] Minghui Jiang, James Anderson, Joel Gillespie, and Martin Mayne. uShuffle: A useful tool for shuffling biological sequences while preserving the k-let counts. *BMC Bioinformatics*, 9(1):192, 2008.

[47] Karim Lari and Steve J. Young. The estimation of stochastic context-free grammars using the Inside-Outside algorithm. *Computer Speech & Language*, 4(1):35–56, 1990.

[48] Pan T. X. Li, Carlos Bustamante, and Jr. Ignacio Tinoco. Unusual mechanical stability of a minimal RNA kissing complex. *Proceedings of the National Academy of Sciences*, 103(43):15847–15852, 2006.

[49] John D. Liu, Liang Zhao, and Tianbing Xia. The Dynamic Structural Basis of Differential Enhancement of Conformational Stability by 5- and 3-Dangling Ends in RNA. *Biochemistry*, 47(22):5962–5975, 2008. PMID: 18457418.

[50] Ronny Lorenz, Stephan Bernhart, Christian Höner zu Siederdissen, Hakim Tafer, Christoph Flamm, Peter F. Stadler, and Ivo L. Hofacker. ViennaRNA Package 2.0. *Algorithms for Molecular Biology*, 6(1):26, 2011.

[51] W. Andy Lorenz, Yann Ponty, and Peter Clote. Asymptotics of RNA Shapes. *J Comput Biol*, 15(1):31–63, 2008.

[52] Todd M. Lowe and Sean R. Eddy. tRNAscan-SE: A Program for Improved Detection of Transfer RNA Genes in Genomic Sequence. *Nucleic Acids Research*, 25(5):0955–964, 1997.

[53] Jian Lu, Yang Shen, Qingfa Wu, Supriya Kumar, Bin He, Suhua Shi, Richard W. Carthew, San Ming Wang, and Chung-I Wu. The birth and death of microRNA genes in Drosophila. *Nat Genet*, 40(3):351–355, 2008.

[54] Rune B. Lyngsø and Christian N. S. Pedersen. RNA Pseudoknot Prediction in Energy-Based Models. *Journal of Computational Biology*, 7(3–4):409–427, 2000.

[55] Nicholas R. Markham and Michael Zuker. UNAFold: software for nucleic acid folding and hybridization. *Methods in molecular biology (Clifton, N.J.)*, 453:3–31, 2008.

[56] Manja Marz, Alexander Donath, Nina Verstraete, Van Trung Nguyen, Peter F. Stadler, and Olivier Bensaude. Evolution of 7SK RNA and Its Protein Partners in Metazoa. *Molecular Biology and Evolution*, 26(12):2821–2830, 2009.

[57] Manja Marz, Toralf Kirsten, and Peter F. Stadler. Evolution of Spliceosomal snRNA Genes in Metazoan Animals. *Journal of Molecular Evolution*, 67(6):594–607, December 2008.

[58] Manja Marz, Axel Mosig, Bärbel M.R. Stadler, and Peter F. Stadler. U7 snRNAs: A computational survey. *Genomics, Proteomics & Bioinformatics*, 5(3):187–195, 2007.

[59] Manja Marz and Peter F. Stadler. Comparative analysis of eukaryotic U3 snoRNA. *RNA Biol*, 6(5):503–507, 2009.

[60] David H. Mathews, Matthew D. Disney, Jessica L. Childs, Susan J. Schroeder, Michael Zuker, and Douglas H. Turner. Incorporating chemical modification constraints into a dynamic programming algorithm for prediction of RNA secondary structure. *Proceedings of the National Academy of Sciences of the United States of America*, 101(19):7287–7292, 2004.

[61] David H. Mathews, Jeffrey Sabina, Michael Zuker, and Douglas H. Turner. Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure. *Journal of Molecular Biology*, 288:911–940, 1999.

[62] John S. McCaskill. The Equilibrium Partition Function and Base Pair Binding Probabilities for RNA Secondary Structure. *Biopolymers*, 1990.

[63] Willem J. G. Melchers, Joost G. J. Hoenderop, Hilbert J. Bruins Slot, Cornelis W. A. Pleij, Evgeny V. Pilipenko, Vadim I. Agol, and Joep M. D. Galama. Kissing of the two predominant hairpin loops in the coxsackie B virus 3' untranslated region is the essential structural feature of the origin of replication required for negative-strand RNA synthesis. *J. Virol.*, 71(1):686–696, 1997.

[64] Eric P. Nawrocki and Sean R. Eddy. Query-Dependent Banding (QDB) for Faster RNA Similarity Searches. *PLoS Comput Biol*, 3(3):e56, 03 2007.

[65] Eric P. Nawrocki and Sean R. Eddy. Infernal 1.1: 100-fold faster RNA homology searches. *Bioinformatics*, 29(22):2933–2935, 2013.

[66] Markus E. Nebel and Anika Scheid. On quantitative effects of RNA shape abstraction. *Theory in Biosciences*, 128:211–225, 2009. 10.1007/s12064-009-0074-z.

[67] Ruth Nussinov, George Pieczenik, Jerrold R. Griggs, and Daniel J. Kleitman. Algorithms for Loop Matchings. *SIAM Journal on Applied Mathematics*, 35(1):68–82, 1978.

[68] Tatsuo Ohmichi, Shu-ichi Nakano, Daisuke Miyoshi, and Naoki Sugimoto. Long RNA dangling end has large energetic contribution to duplex stability. *J. Am. Chem. Soc.*, 124:10367–10372, 2002.

[69] Janina Reeder, Jens Reeder, and Robert Giegerich. Locomotif: from graphical motif description to RNA motif search. *Bioinformatics*, 23(13):i392, 2007.

[70] Janina Reeder, Peter F. Steffen, and Robert Giegerich. Effective ambiguity checking in biosequence analysis. *BMC Bioinformatics*, 6(1):153, 2005.

[71] Jens Reeder and Robert Giegerich. Design, implementation and evaluation of a practical pseudoknot folding algorithm based on thermodynamics. *BMC Bioinformatics*, 5(1):104, 2004.

[72] Jens Reeder and Robert Giegerich. Consensus shapes: an alternative to the Sankoff algorithm for RNA consensus structure prediction. *Bioinformatics*, 21(17):3516–3523, 2005.

[73] Christian M. Reidys, Fenix W. D. Huang, Jørgen E. Andersen, Robert C. Penner, Peter F. Stadler, and Markus E. Nebel. Topology and prediction of RNA pseudoknots. *Bioinformatics*, 27(8):1076–1085, 2011.

[74] Jihong Ren, Baharak Rastegari, Anne E. Condon, and Holger H. Hoos. HotKnots: Heuristic prediction of RNA secondary structures including pseudoknots. *RNA*, 11(10):1494–1504, 2005.

[75] Jessica S. Reuter and David H. Mathews. RNAstructure: software for RNA secondary structure prediction and analysis. *BMC Bioinformatics*, 11:129, 2010.

[76] Elena Rivas and Sean R. Eddy. A dynamic programming algorithm for RNA structure prediction including pseudoknots. *Journal of Molecular Biology*, 285(5):2053–2068, 1999.

[77] Einar Andreas Rødland. Pseudoknots in RNA Secondary Structures: Representation, Enumeration, and Prevalence. *Journal of Computational Biology*, 13(6):1197–1213, 2006.

[78] David Sankoff. Simultaneous Solution of the RNA Folding, Alignment and Protosequence Problems. *SIAM Journal on Applied Mathematics*, 45(5):810–825, 1985.

[79] Georg Sauthoff. *Bellman's GAP: A 2nd Generation Language and System for Algebraic Dynamic Programming.* PhD thesis, Bielefeld University, 2011.

[80] Georg Sauthoff, Stefan Janssen, and Robert Giegerich. Bellman's GAP - A Declarative Language for Dynamic Programming. In *Proceedings of 13th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming*, PPDP '11. ACM, 2011.

[81] Georg Sauthoff, Mathias Möhl, Stefan Janssen, and Robert Giegerich. Bellman's GAP–a language and compiler for dynamic programming in sequence analysis. *Bioinformatics*, 29(5):551–560, 2013.

[82] Claude E. Shannon and Warren Weaver. *A mathematical theory of communication.* American Telephone and Telegraph Company, 1948.

[83] Jana Sperschneider, Amitava Datta, and Michael J. Wise. Heuristic RNA pseudoknot prediction including intramolecular kissing hairpins. *RNA*, 17(1):27–38, 2011.

[84] Peter Steffen. *Compiling a Domain Specific Language for Dynamic Programming.* PhD thesis, University of Bielefeld, Germany, 2006.

[85] Peter Steffen and Robert Giegerich. Versatile and declarative dynamic programming using pair algebras. *BMC Bioinformatics*, 6(1):224, 2005.

[86] Peter Steffen and Robert Giegerich. Table Design in Dynamic Programming. *Information and Computation*, 204(9):1325–1345, 2006.

[87] Corinna Theis, Stefan Janssen, and Robert Giegerich. Prediction of RNA Secondary Structure Including Kissing Hairpin Motifs. In Vincent Moulton and Mona Singh, editors, *Algorithms in Bioinformatics*, volume 6293 of *Lecture Notes in Computer Science*, pages 52–64. Springer Berlin Heidelberg, 2010.

[88] Julie D. Thompson, Desmond G. Higgins, and Toby J. Gibson. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22(22):4673–4680, 1994.

[89] Jr. Ignacio Tinoco and Carlos Bustamante. How RNA folds. *Journal of Molecular Biology*, 293(2):271–281, 1999.

[90] F. H. D. van Batenburg, A. P. Gultyaev, and Cornelis W. A. Pleij. PseudoBase: structural information on RNA pseudoknots. *Nucl. Acids Res.*, 29(1):194–195, 2001.

[91] Björn Voß, Robert Giegerich, and Marc Rehmsmeier. Complete probabilistic analysis of RNA shapes. *BMC Biology*, 4(1):5, 2006.

[92] Amy E. Walter, Douglas H. Turner, James Kim, Matthew H. Lyttle, Peter Müller, David H. Mathews, and Michael Zuker. Coaxial stacking of helixes enhances binding of oligoribonucleotides and improves predictions of RNA folding. *Proc. Nat. Acad. Sci. U.S.A.*, 91:9218–9222, 1994.

[93] Michael S. Waterman. *Introduction to computational biology. Maps, sequences and genomes.* Chapman & Hall, London, 1995.

[94] Kevin A. Wilkinson, Edward J. Merino, and Kevin M. Weeks. Selective 2-hydroxyl acylation analyzed by primer extension (SHAPE): quantitative RNA structure analysis at single nucleotide resolution. *Nature protocols*, 1(3):1610–1616, 2006.

[95] Stefan Wuchty, Walter Fontana, Ivo L. Hofacker, and Peter Schuster. Complete suboptimal folding of RNA and the stability of secondary structures. *Biopolymers*, 49(2):145–165, February 1999.

[96] Tianbing Xia, John Jr. SantaLucia, Mark E. Burkard, Ryszard Kierzek, Susan J. Schroeder, Xiaoqi Jiao, Christopher Cox, and Douglas H. Turner. Thermodynamic parameters for an expanded nearest-neighbor model for formation of RNA duplexes with Watson-Crick base pairs. *Biochemistry*, 37:14719–14735, 1998.

[97] Michael Zuker. Mfold web server for nucleic acid folding and hybridization prediction. *Nucleic Acids Research*, 31:3406–3415, 2003.

Contributions

My scientific contributions comprises the following list of peer-reviewed articles, a textbook chapter and conference talks or posters:

Peer-reviewed articles

- Parts of Chapter 3 have been published as:
  Stefan Janssen, Christian Schudoma, Gerhard Steger, and Robert Giegerich. Lost in folding space? Comparing four variants of the thermodynamic model for RNA secondary structure prediction. *BMC Bioinformatics*, 12(1):429, 2011.

- Parts of Chapter 4 have been published as:
  Stefan Janssen and Robert Giegerich. Faster computation of exact RNA shape probabilities. *Bioinformatics*, 26(5):632–639, 2010.

- Parts of Chapter 5 have been published as:
  Corinna Theis, **Stefan Janssen**, and Robert Giegerich. Prediction of RNA Secondary Structure Including Kissing Hairpin Motifs. In Vincent Moulton and Mona Singh, editors, *Algorithms in Bioinformatics*, volume 6293 of *Lecture Notes in Computer Science*, pages 52–64. Springer Berlin Heidelberg, 2010.

- **Stefan Janssen**, Jens Reeder, and Robert Giegerich. Shape based indexing for faster search of RNA family databases. *BMC Bioinformatics*, 9:131+, February 2008.

- Jan-Philip Schlüter, Jan Reinkensmeier, Svenja Daschkey, Elena Evguenieva-Hackenberg, **Stefan Janssen**, Sebastian Jänicke, Jörg Becker, Robert Giegerich, and Anke Becker. A genome-wide survey of sRNAs in the symbiotic nitrogen-fixing alpha-proteobacterium Sinorhizobium meliloti. *BMC Genomics*, 11(1):245, 2010.

- Florian Grond, **Stefan Janssen**, Stefanie Schirmer, and Thomas Hermann.
  Browsing RNA structures by interactive sonification. *Proceedings of the 3rd Interactive Sonification Workshop*, (Human Interaction with Auditory Displays), 2010.

- Georg Sauthoff, **Stefan Janssen**, and Robert Giegerich. Bellman's GAP - A Declarative Language for Dynamic Programming. In *Proceedings of 13th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming*, PPDP '11. ACM, 2011.

- Georg Sauthoff, Mathias Möhl, **Stefan Janssen**, and Robert Giegerich. Bellman's GAP–a language and compiler for dynamic programming in sequence analysis. *Bioinformatics*, 29(5):551–560, 2013.

Textbook Chapter

- **Stefan Janssen** and Robert Giegerich. Abstract Shape Analysis of RNA. *RNA Bioinformatics tba*, 2014.

Conferences

Talk *Ambivalent Covariance Models.*, “10. Herbstseminar der Bioinformatik”, Doubice, Czech Republic, 2 – 7 Oct 2012.

Talk *Multiple Consensus Structures for Infernal Style Covariance Models.*, “Computational methods for RNA analysis.”, Benasque, Spain, 22 Jul – 3 Aug 2012.

Talk *Longer, deeper and affordable kisses*, “LIX Bioinformatics Colloquium”, Paris, France, 8 – 10 Nov 2010.

Poster *Longer, deeper and affordable kisses*, “The Non-Coding Genome”, Heidelberg, Germany, 13 – 16 Oct 2010.

Talk *Prediction of RNA secondary structure including kissing hairpin motifs*, “10th Workshop on Algorithms in Bioinformatics”, Liverpool, UK, 6 – 10 Sep 2010.

Lecture *RNAshapes*, “Theory and Practice of Computational RNA Biology – An EMBO Practical Course”, Cargèse, France, 26 Apr – 1 May 2010

Talk *Role of structure on Rfam.*, “Computational methods for RNA analysis.”, Benasque, Spain, 26 Jul – 8 Aug 2009.

Talk *A runtime heuristic for faster probabilistic analysis of RNA abstract shapes.*, “Bioinformatics Research and Education Workshop (BREW).”, Helsinki, Finland, 26 – 29 April 2009.

Talk *Shape based indexing for faster search of RNA family databases.*, “Non-Coding RNAs: Computational Challenges and Applications”, Antalya, Turkey, 28 – 29 April 2008.

Software

In the course of this work, I developed and maintain the following software packages. They can be downloaded as source code, installed as Debian packages or used as webservices or submission pages via our webserver BiBiServ. All algorithms are coded in Bellman’s GAP. User interaction is realized via Perl wrappers.

- RNAshapes: RNA secondary structure prediction and shape abstraction for RNA sequences.
  `http://bibiserv.cebitec.uni-bielefeld.de/rnashapes`

- RNAalishapes: RNA secondary structure prediction and shape abstraction for sequence alignments.
  `http://bibiserv.cebitec.uni-bielefeld.de/rnaalishapes`
- pKiss: A tool for secondary structure prediction including kissing hairpin motifs.
  `http://bibiserv.cebitec.uni-bielefeld.de/pkiss`
- RapidShapes: A heuristics which computes the shapes above a specified probability threshold T by generating a list of promising shapes and constructing specialized folding programs for each shape to compute its probability.
  `http://bibiserv.cebitec.uni-bielefeld.de/rapidshapes`
- KnotInFrame: A pipeline to predict ribosomal -1 frameshift sites with a simple pseudoknot as secondary structure in DNA and RNA sequences.
  `http://bibiserv.cebitec.uni-bielefeld.de/knotinframe`
- aCMs: Ambivalent Covariance Models are stochastic homology search tools that support more than one consensus secondary RNA structures.
  `http://bibiserv.cebitec.uni-bielefeld.de/acms`

The site `http://bibiserv.cebitec.uni-bielefeld.de/fold-grammars` shall serve as a lean documentation of the different code components.

9 Appendices

9.1 Appendix A: Complete set of all 20 TDM generators.

The basic concept of a TDM generator for a given shape string has been explained in Section 4.2.2, i. e. we parse the shape string with a TDM grammar (e. g. $\mathcal{G}_{TDM_5}$) and evaluate the single candidate with a TDM algebra (e. g. $\mathcal{I}_{tdm\_OverDangle\_5}$) into Bellman's GAP code for a specialized grammar. The previous text was about generating a TDM grammar, following the OverDangle prototype, for shape strings of level 5. Here, we give details about generators for the missing 4 shape levels, first for OverDangle prototypes, later for the three remaining prototype grammars NoDangle, MicroState and MacroState. We start by completing the signature Σ_{TDM} in Table 9.1, where all algebra functions are defined. Then, the parsing grammars are presented and finally we give all detailes about the evaluation algebras.

9.1.1 Common Signature

Table 4.2 in Section 4.2.2 of the main text contains only a small part of the complete signature Σ_{TDM}, namely those algebra functions used by grammar $\mathcal{G}_{\mathrm{TDM}_5}$. The advantage of using just one signature for different grammars is the re-usability of large code blocks due to inheritance. On the downside, we have to define and implement all functions, even if they are not used by a specific grammar (here $\mathcal{G}_{\mathrm{TDM}_5}$, which only uses 11 out of 83 functions). Thus, we decided to show only this small part of Σ_{TDM} in the main text and postpone full definition to Table 9.1.

9.1.2 TDM grammars

Shape level 5

See Section 4.2.2, Figure 4.2.2.

Shape level 4

Compared to level 5 shapes, level 4 additionally recognizes helix interruptions if they stem from internal loops, cf. algebras $\mathcal{I}_{\pi_5}$ and $\mathcal{I}_{\pi_4}$ in Table 3.2. To respond to this interruption, $\mathcal{G}_{\mathrm{TDM}_4}$ – see Figure 9.1 – has the extra production:

$$comp \rightarrow \mathrm{helixinterrupt}(\texttt{[}, \mathrm{strong}(comp), \texttt{]}).$$

Table 9.1: Common Signature Σ_{TDM} for $\mathcal{G}_{\mathrm{TDM}_5}$ and the other 5 TDM grammars, see Section 9.1

root$(\mathcal{S})$
dangle$(\mathcal{S})$
next_hlmode$(\mathcal{S}, \mathcal{S})$
last_hlmode$(\mathcal{S})$
unpaired$(\mathcal{A})$
strong$(\mathcal{S})$
hairpin$(\mathcal{A}, \mathcal{A})$
multiloop$(\mathcal{A}, \mathcal{S}, \mathcal{A})$
next_mlmode$(\mathcal{S}, \mathcal{S})$
last_mlmode$(\mathcal{S}, \mathcal{S})$
internalloop$(\mathcal{A}, \mathcal{A}, \mathcal{S}, \mathcal{A}, \mathcal{A})$
leftbulge$(\mathcal{A}, \mathcal{A}, \mathcal{S}, \mathcal{A})$
rightbulge$(\mathcal{A}, \mathcal{S}, \mathcal{A}, \mathcal{A})$
helixinterrupt$(\mathcal{A}, \mathcal{S}, \mathcal{A})$
mladdss$(\mathcal{A})$
mlend$()$
sadd$(\mathcal{A}, \mathcal{S})$
saddml$(\mathcal{A}, \mathcal{S})$
drem$(\mathcal{S})$
edl$(\mathcal{S})$
edr$(\mathcal{S})$
edlr$(\mathcal{S})$
mldl$(\mathcal{S})$
mladl$(\mathcal{A}, \mathcal{S})$
mldr$(\mathcal{S})$
mladr$(\mathcal{S}, \mathcal{A})$
mldlr$(\mathcal{S})$
mladlr$(\mathcal{A}, \mathcal{S}, \mathcal{A})$

mldladr$(\mathcal{S}, \mathcal{A})$
mladldr$(\mathcal{A}, \mathcal{S})$
ml$(\mathcal{S})$
next_hlmode_r$(\mathcal{S}, \mathcal{A}, \mathcal{S})$
last_r$(\mathcal{S}, \mathcal{A}, \mathcal{S})$
last_$(\mathcal{S}, \mathcal{S})$
last_ml_$(\mathcal{S})$
last_ml_r$(\mathcal{S}, \mathcal{S})$
next_ml_r$(\mathcal{S}, \mathcal{A}, \mathcal{S})$
mlend_$(\mathcal{A})$
mlnil$(\mathcal{A})$
p$(\mathcal{A}, \mathcal{S})$
nil$()$
trafo$(\mathcal{S})$
nextambd$(\mathcal{A}, \mathcal{S}, \mathcal{A}, \mathcal{S})$
nextcadda$(\mathcal{A}, \mathcal{S}, \mathcal{S})$
nextcadd$(\mathcal{A}, \mathcal{S}, \mathcal{S})$
lastcadda$(\mathcal{A}, \mathcal{S})$
lastcadd$(\mathcal{A}, \mathcal{S}, \mathcal{S})$
nextcaddc$(\mathcal{S}, \mathcal{S})$
nextambda$(\mathcal{S}, \mathcal{A}, \mathcal{S})$
lastcaddb$(\mathcal{S}, \mathcal{S})$
comb1_a$(\mathcal{A}, \mathcal{S}, \mathcal{A}, \mathcal{S})$
combine1_a$(\mathcal{A}, \mathcal{S}, \mathcal{S})$
nextcombine1_b$(\mathcal{A}, \mathcal{S}, \mathcal{S})$
lastcombine1_b$(\mathcal{A}, \mathcal{S}, \mathcal{A}, \mathcal{S})$
nextcombine2_b$(\mathcal{S}, \mathcal{S})$
acomb2_a$(\mathcal{S}, \mathcal{A}, \mathcal{S})$

lastcombine2_b$(\mathcal{S}, \mathcal{A}, \mathcal{S})$
nextcombine3_a$(\mathcal{S}, \mathcal{S})$
nextcombine3_b$(\mathcal{S}, \mathcal{S})$
nextacomb3_a$(\mathcal{S}, \mathcal{A}, \mathcal{S})$
lastcombine3_a$(\mathcal{S}, \mathcal{A}, \mathcal{S}, \mathcal{A})$
lastcombine3_b$(\mathcal{S}, \mathcal{S}, \mathcal{A})$
lastacomb3_a$(\mathcal{S}, \mathcal{A}, \mathcal{S}, \mathcal{A})$
nextcombine4_a$(\mathcal{A}, \mathcal{S}, \mathcal{S})$
nextcombine4_b$(\mathcal{A}, \mathcal{S}, \mathcal{S})$
nextacomb4_a$(\mathcal{A}, \mathcal{S}, \mathcal{A}, \mathcal{S})$
lastcombine4_a$(\mathcal{A}, \mathcal{S}, \mathcal{S}, \mathcal{A})$
lastcombine4_b$(\mathcal{A}, \mathcal{S}, \mathcal{A}, \mathcal{S}, \mathcal{A})$
lastacomb4_a$(\mathcal{A}, \mathcal{S}, \mathcal{A}, \mathcal{S}, \mathcal{A})$
block_dl$(\mathcal{S})$
block_dlr$(\mathcal{S})$
no_dl_ss_end__next$(\mathcal{S})$
no_dl_ss_end__last$(\mathcal{S})$
no_dl_no_ss_end__next$(\mathcal{S})$
no_dl_no_ss_end__last$(\mathcal{S})$
dl_or_ss_left_no_ss_end__next$(\mathcal{S})$
dl_or_ss_left_no_ss_end__last$(\mathcal{S})$
dl_or_ss_left_ss_end__next$(\mathcal{S})$
dl_or_ss_left_ss_end__last$(\mathcal{S})$
leftunpairedend$(\mathcal{A})$
leftunpaired$(\mathcal{S})$
unpaired_macrostate$(\mathcal{S})$

$code = \text{convert}(\mathcal{S})$

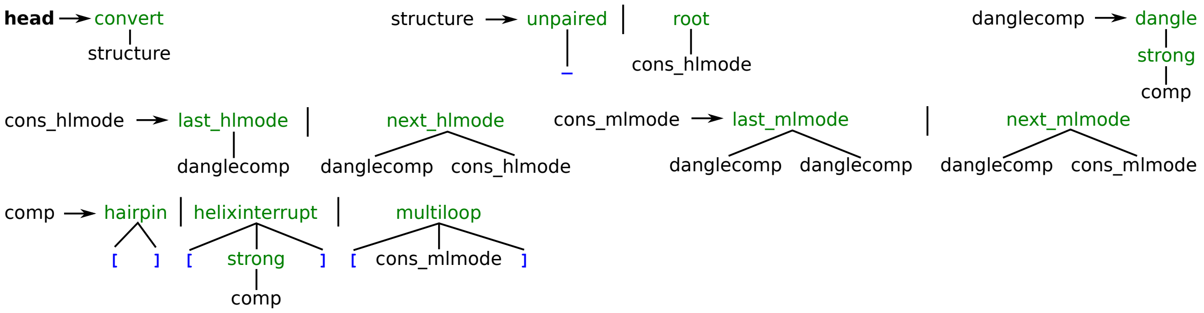


Figure 9.1: Grammar $\mathcal{G}_{\mathrm{TDM}_4}$ to generate specialized tree grammars for shape levels 4, which is identical to grammar $\mathcal{G}_{\mathrm{TDM}_3}$ for shape level 3.

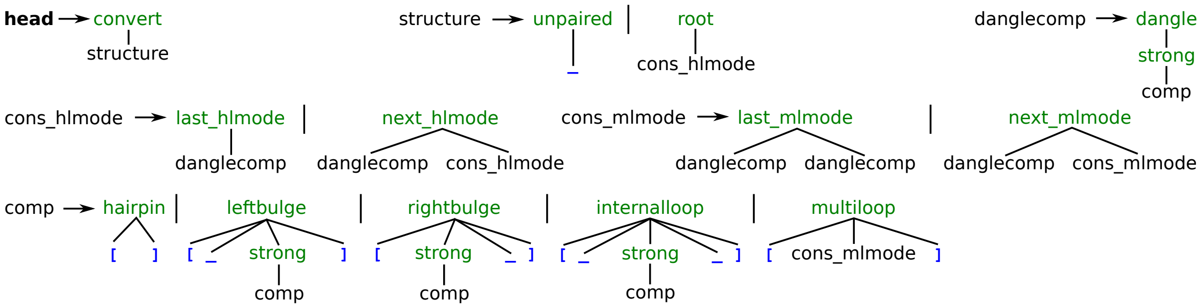


Figure 9.2: Grammar $\mathcal{G}_{\mathrm{TDM}_2}$ to generate specialized tree grammars for shape levels 2.

Everything else is identical to $\mathcal{G}_{\mathrm{TDM}_5}$ (Figure 4.1).

Shape level 3

The difference between shape levels 4 and 3 is that helix interruptions by left or right bulges are recognized in level 3, but not in level 4. The shape string remains of the same kind, just . . . [[. . .]] . . . situations appear more often due to more recorded stem interruptions. Level 3 reports more helix interruptions, but from the shape string itself we cannot conclude the type of the interruption, i. e. it is treated identically. Thus, we can reuse the same grammar of Figure 9.1 for $\mathcal{G}_{\mathrm{TDM}_3}$.

Shape level 2

Shape level 2 is also about helix interruptions. Due to the newly introduced underscore character _, we can distinguish between internal loops ([_. . . _]) and left ([_. . .]) or right bulges ([. . . _]). This is reflected by the three separate productions internalloop, leftbulge and rightbulge for the non-terminal *comp* of $\mathcal{G}_{\mathrm{TDM}_2}$, see Figure 9.2.

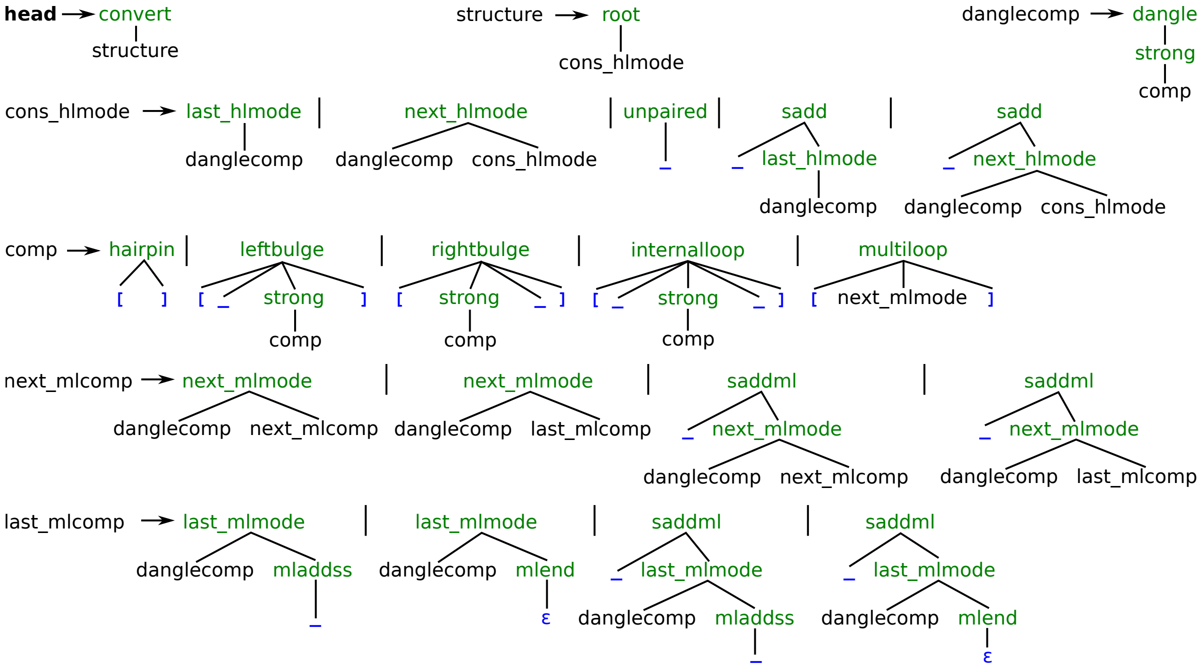


Figure 9.3: Grammar $\mathcal{G}_{\mathrm{TDM}_1}$ to generate specialized tree grammars for shape levels 1.

Shape level 1

Shape level 1 also accounts for all stretches of unpaired bases, which might precede (sadd, saddml) or follow (mladdss) any closed component. Grammar $\mathcal{G}_{\mathrm{TDM}_1}$ must reflect all these situations individually, thus bloating the set of productions rules, see Figure 9.3.

Shape level 1 for MacroState The prototype grammar $\mathcal{G}_{\mathrm{MacroState}}$ has been developed with the objective to be semantically unambiguous regarding the Vienna-Dot-Bracket semantics. A level 1 shape string `[]_[]` might produce structures like `((...)).((...))` or `((...))..((...))`. It is fundamental for being unambiguous that the central unpaired base of the first structure is parsed by the $\mathcal{G}_{\mathrm{MacroState}}$ production $noleft\_dangle \rightarrow$ ambd'($nodangle$, `b`, $noleft\_dangle$) and no other. For the second structure, the middle two unpaired bases should only be parsable by the production $noleft\_dangle \rightarrow$ cadd"($edangler, \{left\_dangle|left\_unpaired\}$). Compare with Figure 3.2.6. The last alternative cadd"' of non-terminal $noleft\_dangle$ from $\mathcal{G}_{\mathrm{MacroState}}$ can never be used for the complete shape `[]_[]`. Thus, a generator grammar $\mathcal{G}_{\mathrm{TDM}_1^{\mathrm{MacroState}}}$ – see Figures 9.4 and 9.5 – has to track the different ways to parse the input shape string, which directly comes with more than one parse. Now, the generator grammar is not syntactically unambiguous any more.

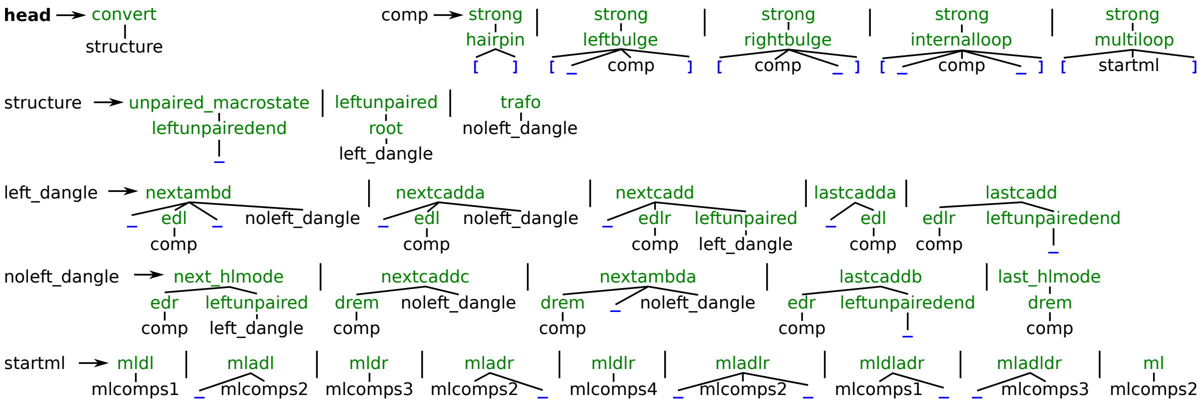


Figure 9.4: Part 1 of 2: Grammar $\mathcal{G}_{\mathrm{TDM}_1^{\mathrm{MacroState}}}$ to generate specialized tree grammars for shape levels 1 for the MacroState prototype.

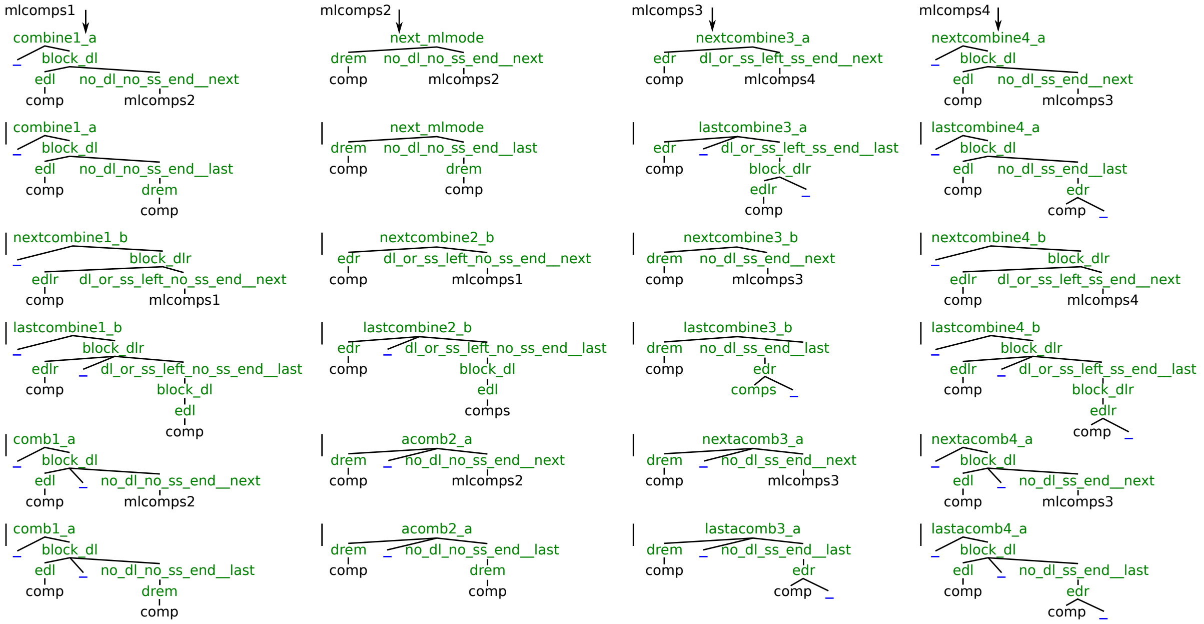


Figure 9.5: Part 2 of 2: Grammar $\mathcal{G}_{\mathrm{TDM}_1^{\mathrm{MacroState}}}$ to generate specialized tree grammars for shape levels 1 for the MacroState prototype.

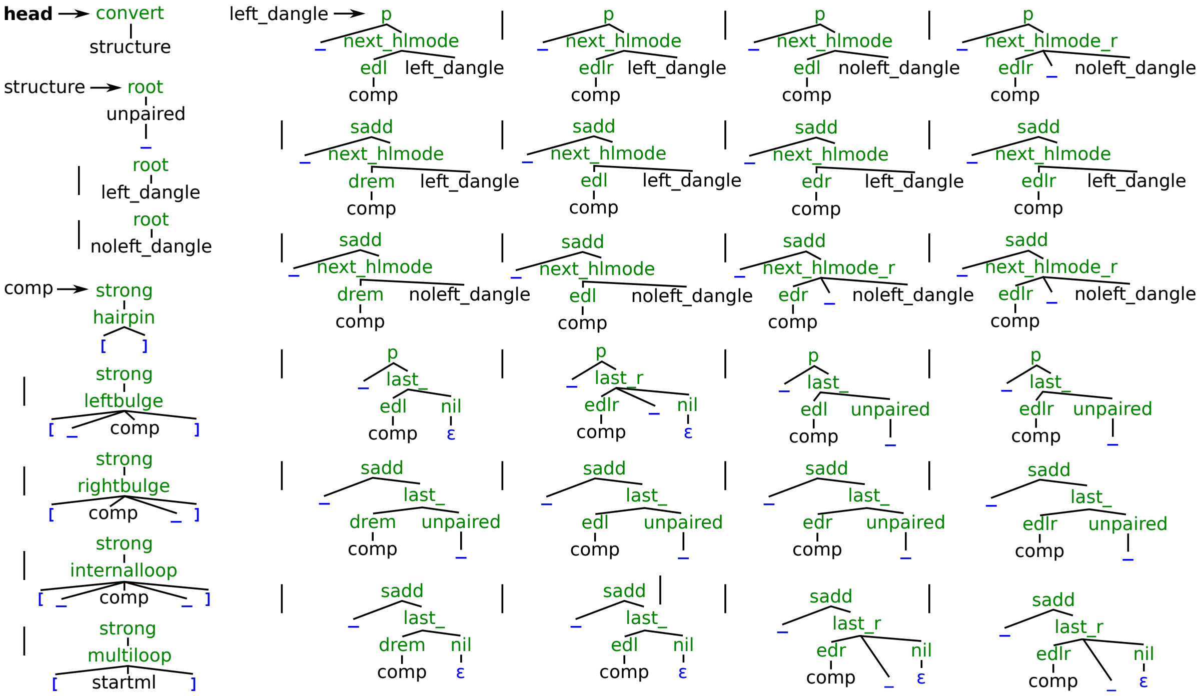


Figure 9.6: Part 1 of 2: Grammar $\mathcal{G}_{\text{TDM}_1^{\text{MicroState}}}$ to generate specialized tree grammars for shape levels 1 for the MicroState prototype.

Shape level 1 for MicroState Similar to MacroState, we need a set of parses for the input shape string when generating grammars for MicroState. Each stem of a MicroState structure is the source for maximal four possible ways how to dangle neighboring unpaired bases; but the same base cannot be dangled to two different stems. Thus, shape string `[]_[]` must facilitate three different dangling situations for a structure like `((...)).((...))`: 1) central unpaired base dangles to the left component, 2) to the right component or 3) to none of them. A situation where the central unpaired base dangles to both components must be forbidden, which means that $\mathcal{G}_{\text{TDM}_1^{\text{MicroState}}}$ cannot treat the components separately. All the positions where one or more unpaired bases can be part of a secondary structure forces $\mathcal{G}_{\text{TDM}_1^{\text{MicroState}}}$ to have a tremendous amount of production rules, see Figures 9.6 and 9.7.

9.1.3 TDM algebras

Shape level 5, OverDangle prototype

See Section 4.2.2, Table 4.1.

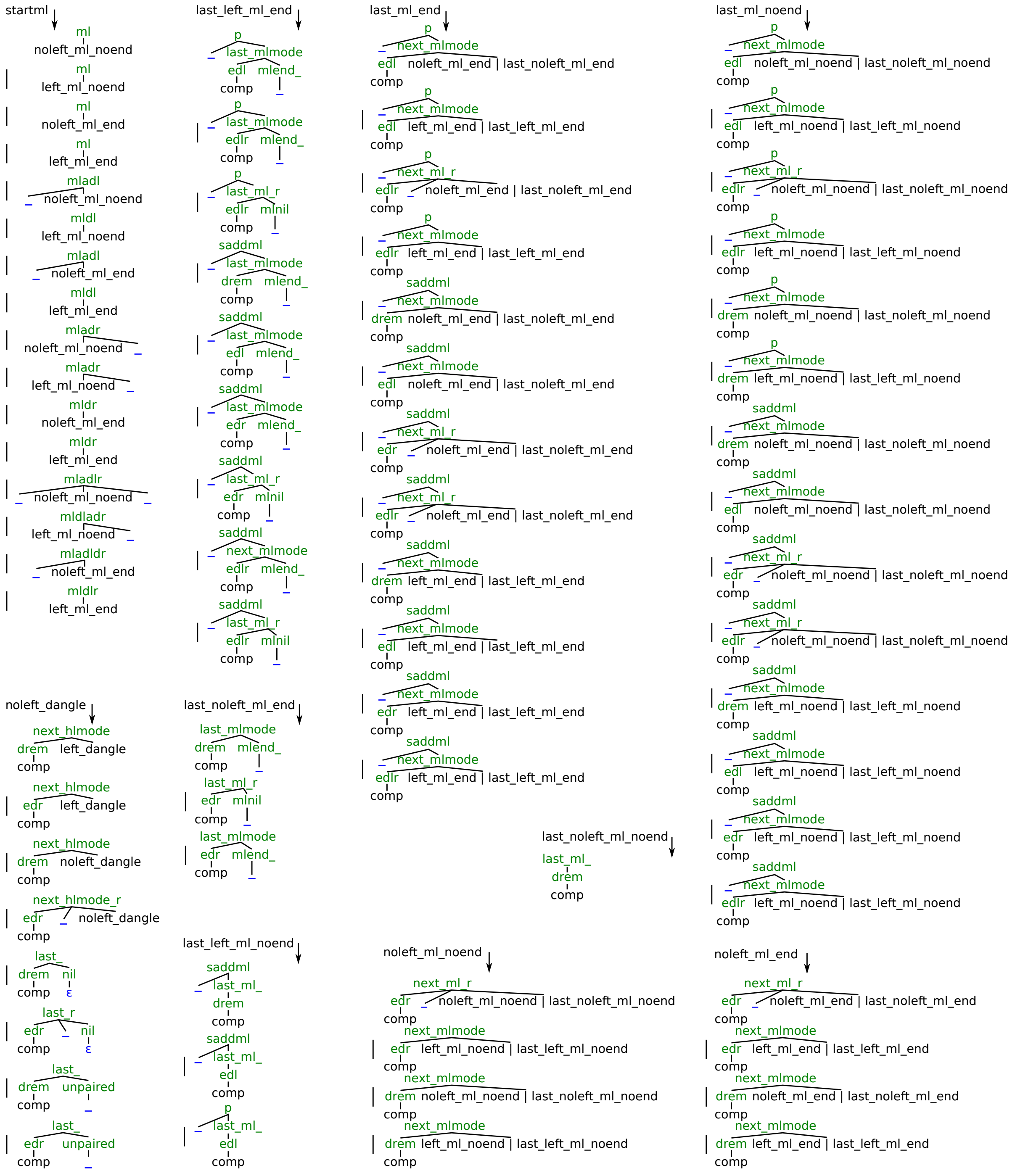


Figure 9.7: Part 2 of 2: Grammar $\mathcal{G}_{\mathrm{TDM}_1^{\mathrm{MicroState}}}$ to generate specialized tree grammars for shape levels 1 for the MicroState prototype.

Table 9.2: TDM algebra for shape level 4, following the OverDangle prototype grammar.

algebra function	$\mathcal{I}_{tdm\_OverDangle\_4} \mathrel{-\!\!\rhd} \mathcal{I}_{tdm\_OverDangle\_5}$
$\text{strong}(x)$	$\text{strong}^{\mathtt{x}} \to \text{sr}(\mathtt{b}, \text{weak}^{\mathtt{x}}, \mathtt{b})_{bp, \neq LP}$
	$\text{strong}^{\mathtt{x}} \to \text{weak}^{\mathtt{x}}_{= LP}$
	$\text{weak}^{\mathtt{x}} \to \text{stack}^{\mathtt{x}}$
	$\text{weak}^{\mathtt{x}} \to \text{leftB}^{\mathtt{x}}$
	$\text{weak}^{\mathtt{x}} \to \text{rightB}^{\mathtt{x}}$
	$\text{stack}^{\mathtt{x}} \to \text{sr}(\mathtt{b}, \text{weak}^{\mathtt{x}}, \mathtt{b})_{bp}$
	$\text{leftB}^{\mathtt{x}} \to \text{bl}(\mathtt{b}, \mathtt{r}_{\leq 30}, \text{strong}^{\mathtt{x}}, \mathtt{b})_{bp}$
	$\text{rightB}^{\mathtt{x}} \to \text{br}(\mathtt{b}, \text{strong}^{\mathtt{x}}, \mathtt{r}_{\leq 30}, \mathtt{b})_{bp}$
$\text{helixinterrupt}(a, x, b)$	$\text{weak}^{\mathtt{[x]}} \to \text{iloop}^{\mathtt{[x]}}$
	$\text{iloop}^{\mathtt{[x]}} \to \text{il}(\mathtt{b}, \mathtt{r}_{\leq 30}, \text{strong}^{\mathtt{x}}, \mathtt{r}_{\leq 30}, \mathtt{b})_{bp}$

Table 9.3: TDM algebra for shape level 3, following the OverDangle prototype grammar.

algebra function	$\mathcal{I}_{tdm\_OverDangle\_3} \mathrel{-\!\!\rhd} \mathcal{I}_{tdm\_OverDangle\_5}$
$\text{strong}(x)$	$\text{strong}^{\mathtt{x}} \to \text{sr}(\mathtt{b}, \text{weak}^{\mathtt{x}}, \mathtt{b})_{bp, \neq LP}$
	$\text{strong}^{\mathtt{x}} \to \text{weak}^{\mathtt{x}}_{= LP}$
	$\text{weak}^{\mathtt{x}} \to \text{stack}^{\mathtt{x}}$
	$\text{stack}^{\mathtt{x}} \to \text{sr}(\mathtt{b}, \text{weak}^{\mathtt{x}}, \mathtt{b})_{bp}$
$\text{helixinterrupt}(a, x, b)$	$\text{weak}^{\mathtt{[x]}} \to \text{iloop}^{\mathtt{[x]}}$
	$\text{weak}^{\mathtt{[x]}} \to \text{leftB}^{\mathtt{[x]}}$
	$\text{weak}^{\mathtt{[x]}} \to \text{rightB}^{\mathtt{[x]}}$
	$\text{leftB}^{\mathtt{[x]}} \to \text{bl}(\mathtt{b}, \mathtt{r}_{\leq 30}, \text{strong}^{\mathtt{x}}, \mathtt{b})_{bp}$
	$\text{rightB}^{\mathtt{[x]}} \to \text{br}(\mathtt{b}, \text{strong}^{\mathtt{x}}, \mathtt{r}_{\leq 30}, \mathtt{b})_{bp}$
	$\text{iloop}^{\mathtt{[x]}} \to \text{il}(\mathtt{b}, \mathtt{r}_{\leq 30}, \text{strong}^{\mathtt{x}}, \mathtt{r}_{\leq 30}, \mathtt{b})_{bp}$

Shape level 4, OverDangle prototype

In shape level 4 a stem and a stem interruption by an internal loop has to be distinguished to control the presents of such an interruption. Thus, $\mathcal{I}_{tdm\_OverDangle\_4}$ – see Table 9.2 – basically inherits all algebra functions from $\mathcal{I}_{tdm\_OverDangle\_5}$, but the productions for an internal loop (iloop and weak) are moved from strong to helixinterrupt.

Shape level 3, OverDangle prototype

The same argument as in level 4 applies to shape level 3. Stem interruptions are recognized still for internal loops but now also for left and right bulges. Thus, in $\mathcal{I}_{tdm\_OverDangle\_3}$ (Table 9.3) productions for these three structural motifs move from strong to helixinterrupt.

Table 9.4: TDM algebra for shape level 2, following the OverDangle prototype grammar.

algebra function	$\mathcal{I}_{tdm\_OverDangle\_2} \mathrel{-\!\!\triangleright} \mathcal{I}_{tdm\_OverDangle\_5}$
$\text{strong}(x)$	$\text{strong}^{\texttt{x}} \rightarrow \text{sr}(\texttt{b}, \text{weak}^{\texttt{x}}, \texttt{b})_{bp, \neq LP}$
	$\text{strong}^{\texttt{x}} \rightarrow \text{weak}^{\texttt{x}}_{= LP}$
	$\text{weak}^{\texttt{x}} \rightarrow \text{stack}^{\texttt{x}}$
	$\text{stack}^{\texttt{x}} \rightarrow \text{sr}(\texttt{b}, \text{weak}^{\texttt{x}}, \texttt{b})_{bp}$
$\text{leftbulge}(a, b, x, d)$	$\text{weak}^{\texttt{[\_x]}} \rightarrow \text{leftB}^{\texttt{[\_x]}}$
	$\text{leftB}^{\texttt{[\_x]}} \rightarrow \text{bl}(\texttt{b}, \texttt{r}_{\leq 30}, \text{strong}^{\texttt{x}}, \texttt{b})_{bp}$
$\text{rightbulge}(a, x, c, d)$	$\text{weak}^{\texttt{[x\_]}} \rightarrow \text{rightB}^{\texttt{[x\_]}}$
	$\text{rightB}^{\texttt{[x\_]}} \rightarrow \text{br}(\texttt{b}, \text{strong}^{\texttt{x}}, \texttt{r}_{\leq 30}, \texttt{b})_{bp}$
$\text{internalloop}(a, b, x, c, d)$	$\text{weak}^{\texttt{[\_x\_]}} \rightarrow \text{iloop}^{\texttt{[\_x\_]}}$
	$\text{iloop}^{\texttt{[\_x\_]}} \rightarrow \text{il}(\texttt{b}, \texttt{r}_{\leq 30}, \text{strong}^{\texttt{x}}, \texttt{r}_{\leq 30}, \texttt{b})_{bp}$

Shape level 2, OverDangle prototype

Level 2 is still about helix interruptions, but their types are now treated differently. Thus, $\mathcal{I}_{tdm\_OverDangle\_2}$ (Table 9.4) contains not just one general algebra function helixinterrupt, but three interruption specific ones leftbulge, rightbulge and internalloop.

Shape level 1, OverDangle prototype

Shape level 1 represents every unpaired base in the shape string with *one* underscore (_). Situations where two stretches of unpaired bases (both represented with an _) for two sub-motifs are combined, have to be handled with care, such that the concatenation cannot have two adjacent underscores. For functions last_hlmode and last_mlmode, the algebra function of $\mathcal{I}_{tdm\_OverDangle\_1}$ (Table 9.5) itself rather than the grammar design must cope with this concatenation problem. Read an unsatisfied if-clause as redirecting the so far composed set of grammar rules to the next algebra function without adding any further rules.

Shape levels 5 to 1, NoDangle prototype

Since the different scoring evaluation between OverDangle and NoDangle stem from using different algebras – namely $\mathcal{I}_{bwe}^{OverDangle}$ and $\mathcal{I}_{bwe}$ – and not from different grammars, the TDM grammar generator for NoDangle is identical to OverDangle. Assume we generate $\mathcal{G}_p$, the result of folding sequence s will follow the OverDangle model when executing the Bellman's GAP instance $\mathcal{G}_p(\mathcal{I}_{bwe}^{OverDangle}, s)$ and it would follow the NoDangle model with instance $\mathcal{G}_p(\mathcal{I}_{bwe}, s)$ – using the very same Grammar $\mathcal{G}_p$.

Shape levels 5 to 2, MicroState prototype

Difference between MicroState and OverDangle is the way of dangling bases directly adjacent to stems. While OverDangle always dangles everything next to a stem (drem or

Table 9.5: TDM algebra for shape level 1, following the OverDangle prototype grammar.

algebra function	$\mathcal{I}_{tdm\_OverDangle\_1} \rightarrowtriangle \mathcal{I}_{tdm\_OverDangle\_5}$
$\text{root}(x)$	$\text{struct} \to \text{struct}^{\mathtt{x}}$
$\text{next\_hlmode}(x,y)$	$\text{struct}^{\mathtt{xy}} \to \text{cadd}(\text{dangle}^{\mathtt{x}}, \text{struct}^{\mathtt{y}})$
$\text{last\_hlmode}(x)$	$\{\ \text{struct}^{\mathtt{x}} \to \text{dangle}^{\mathtt{x}} \quad \text{if } x \neq \_$
$\text{unpaired}(a)$	$\text{struct}^{\_} \to \text{sadd}(\mathtt{b}, \text{struct}^{\_})$
	$\text{struct}^{\_} \to \text{sadd}(\mathtt{b}, \text{nil}(\mathtt{l}))$
$\text{strong}(x)$	$\text{strong}^{\mathtt{x}} \to \text{sr}(\mathtt{b}, \text{weak}^{\mathtt{x}}, \mathtt{b})_{bp, \neq\ LP}$
	$\text{strong}^{\mathtt{x}} \to \text{weak}^{\mathtt{x}}_{=\ LP}$
	$\text{weak}^{\mathtt{x}} \to \text{stack}^{\mathtt{x}}$
	$\text{stack}^{\mathtt{x}} \to \text{sr}(\mathtt{b}, \text{weak}^{\mathtt{x}}, \mathtt{b})_{bp}$
$\text{hairpin}(a,b)$	$\text{weak}^{\mathtt{[]}} \to \text{hairpin}^{\mathtt{[]}}$
	$\text{hairpin}^{\mathtt{[]}} \to \text{hl}(\mathtt{b}, \mathtt{r}_{\geq\ 3}, \mathtt{b})_{bp}$
$\text{multiloop}(a,x,b)$	$\text{weak}^{\mathtt{[x]}} \to \text{multiloop}^{\mathtt{[x]}}$
	$\text{multiloop}^{\mathtt{[x]}} \to \text{ml}(\mathtt{b}, \text{ml\_comps}^{\mathtt{x}}, \mathtt{b})_{bp}$
$\text{next\_mlmode}(x,y)$	$\text{ml\_comps}^{\mathtt{xy}} \to \text{cadd}(\text{incl}(\text{dangle}^{\mathtt{x}}), \text{ml\_comps}^{\mathtt{y}})$
$\text{last\_mlmode}(x,y)$	$\begin{cases} \text{ml\_comps}^{\mathtt{xy}} \to \text{addss}(\text{incl}(\text{dangle}^{\mathtt{x}}), \mathtt{r}) & \text{if } y = \_ \\ \text{ml\_comps}^{\mathtt{xy}} \to \text{incl}(\text{dangle}^{\mathtt{x}}) & \text{otherwise} \end{cases}$
$\text{leftbulge}(a,b,x,d)$	$\text{weak}^{\mathtt{[\_x]}} \to \text{leftB}^{\mathtt{[\_x]}}$
	$\text{leftB}^{\mathtt{[\_x]}} \to \text{bl}(\mathtt{b}, \mathtt{r}_{\leq\ 30}, \text{strong}^{\mathtt{x}}, \mathtt{b})_{bp}$
$\text{rightbulge}(a,x,c,d)$	$\text{weak}^{\mathtt{[x\_]}} \to \text{rightB}^{\mathtt{[x\_]}}$
	$\text{rightB}^{\mathtt{[x\_]}} \to \text{br}(\mathtt{b}, \text{strong}^{\mathtt{x}}, \mathtt{r}_{\leq\ 30}, \mathtt{b})_{bp}$
$\text{internalloop}(a,b,x,c,d)$	$\text{weak}^{\mathtt{[\_x\_]}} \to \text{iloop}^{\mathtt{[\_x\_]}}$
	$\text{iloop}^{\mathtt{[\_x\_]}} \to \text{il}(\mathtt{b}, \mathtt{r}_{\leq\ 30}, \text{strong}^{\mathtt{x}}, \mathtt{r}_{\leq\ 30}, \mathtt{b})_{bp}$
$\text{mladdss}(a)$	$\_$
$\text{mlend}()$	ε
$\text{sadd}(a,x)$	$\text{struct}^{\_\mathtt{x}} \to \text{sadd}(\mathtt{b}, \text{struct}^{\_\mathtt{x}})$
	$\text{struct}^{\_\mathtt{x}} \to \text{sadd}(\mathtt{b}, \text{struct}^{\mathtt{x}})$
$\text{saddml}(a,x)$	$\text{ml\_comps}^{\_\mathtt{x}} \to \text{sadd}(\mathtt{b}, \text{ml\_comps}^{\_\mathtt{x}})$
	$\text{ml\_comps}^{\_\mathtt{x}} \to \text{sadd}(\mathtt{b}, \text{ml\_comps}^{\mathtt{x}})$

Table 9.6: TDM algebra for shape level 5, following the MicroState prototype grammar.

algebra function	$\mathcal{I}_{tdm\_MicroState\_5} \rightarrowtriangle \mathcal{I}_{tdm\_OverDangle\_5}$ $\mathcal{I}_{tdm\_MicroState\_4} \rightarrowtriangle \mathcal{I}_{tdm\_OverDangle\_4}$ $\mathcal{I}_{tdm\_MicroState\_3} \rightarrowtriangle \mathcal{I}_{tdm\_OverDangle\_3}$ $\mathcal{I}_{tdm\_MicroState\_2} \rightarrowtriangle \mathcal{I}_{tdm\_OverDangle\_2}$
$\text{dangle}(x)$	$\text{dangle}^{\mathtt{x}} \rightarrow \text{drem}(\mathtt{l}, \text{strong}^{\mathtt{x}}, \mathtt{l})$
	$\text{dangle}^{\mathtt{x}} \rightarrow \text{edl}(\mathtt{b}, \text{strong}^{\mathtt{x}}, \mathtt{l})$
	$\text{dangle}^{\mathtt{x}} \rightarrow \text{edr}(\mathtt{l}, \text{strong}^{\mathtt{x}}, \mathtt{b})$
	$\text{dangle}^{\mathtt{x}} \rightarrow \text{edlr}(\mathtt{b}, \text{strong}^{\mathtt{x}}, \mathtt{b})$
$\text{multiloop}(a, x, b)$	$\text{weak}^{[\mathtt{x}]} \rightarrow \text{multiloop}^{[\mathtt{x}]}$
	$\text{multiloop}^{[\mathtt{x}]} \rightarrow \text{ml}(\mathtt{b}, \text{ml\_comps}^{\mathtt{x}}, \mathtt{b})_{bp}$
	$\text{multiloop}^{[\mathtt{x}]} \rightarrow \text{mldl}(\mathtt{b}, \mathtt{b}, \text{ml\_comps}^{\mathtt{x}}, \mathtt{b})_{bp}$
	$\text{multiloop}^{[\mathtt{x}]} \rightarrow \text{mldr}(\mathtt{b}, \text{ml\_comps}^{\mathtt{x}}, \mathtt{b}, \mathtt{b})_{bp}$
	$\text{multiloop}^{[\mathtt{x}]} \rightarrow \text{mldlr}(\mathtt{b}, \mathtt{b}, \text{ml\_comps}^{\mathtt{x}}, \mathtt{b}, \mathtt{b})_{bp}$

ml) – be it available or not – MicroState discriminates four different possibilities: 1) drem and ml, 2) edl and mldl, 3) edr and mldr and 4) edlr and mldlr. Thus, TDM algebras $\mathcal{I}_{tdm\_MicroState\_i}$ for $5 \leq i \leq 2$ (see Table 9.6) inherit everything from $\mathcal{I}_{tdm\_OverDangle\_i}$, but extend the dangling possibilities in algebra functions dangle and multiloop.

Shape level 1, MicroState prototype

Due to the elaborate design of Grammar $\mathcal{G}_{\text{TDM}_1^{\text{MicroState}}}$ (Figures 9.6 and 9.7), which uses a lot of additional algebra functions, TDM algebra $\mathcal{I}_{tdm\_MicroState\_1}$ (Table 9.7) can inherit the core from $\mathcal{I}_{tdm\_OverDangle\_5}$ but needs to define all these new algebra functions. Since $\mathcal{G}_{\text{TDM}_1^{\text{MicroState}}}$ is syntactically ambiguous, the choice function here must merge ($\cup_{\text{rules}}$) the set of rules from several different parses. It is implemented as an external `C++` function `merge`, which basically uniquely joins the keys of the two level hashes for the rules.

Table 9.7: TDM algebra for shape level 1, following the MicroState prototype grammar.

algebra function	$\mathcal{I}_{tdm\_MicroState\_1} \rightarrowtriangle \mathcal{I}_{tdm\_OverDangle\_5}$
$\text{root}(x)$	$\text{struct} \rightarrow \text{struct}^{\mathtt{x}}$
$\text{unpaired}(a)$	$\text{struct}^{\mathtt{-}} \rightarrow \text{sadd}(\mathtt{b}, \text{struct}^{\mathtt{-}})$
	$\text{struct}^{\mathtt{-}} \rightarrow \text{sadd}(\mathtt{b}, \text{nil}(\mathtt{l}))$
$\text{nil}()$	$\text{struct} \rightarrow \text{nil}(\mathtt{l})$
$\text{next\_hlmode\_r}(x, a, y)$	$\text{struct}^{\mathtt{x\_y}} \rightarrow \text{cadd}(\text{dangle}^{\mathtt{x}}, \text{struct}^{\mathtt{y}})$
$\text{next\_hlmode}(x, y)$	$\text{struct}^{\mathtt{xy}} \rightarrow \text{cadd}(\text{dangle}^{\mathtt{x}}, \text{struct}^{\mathtt{y}})$
$\text{last\_r}(x, a, y)$	$\text{struct}^{\mathtt{x\_y}} \rightarrow \text{cadd}(\text{dangle}^{\mathtt{x}}, \text{struct}^{\mathtt{y}})$
$\text{last}(x, y)$	$\text{struct}^{\mathtt{xy}} \rightarrow \text{cadd}(\text{dangle}^{\mathtt{x}}, \text{struct}^{\mathtt{y}})$
$\text{drem}(x)$	$\text{dangle}^{\mathtt{x}} \rightarrow \text{drem}(\mathtt{l}, \text{strong}^{\mathtt{x}}, \mathtt{l})$

Continued on next page

Continued from previous page

algebra function	$\mathcal{I}_{tdm\_MicroState\_1} \mathrel{-\!\!\rhd} \mathcal{I}_{tdm\_OverDangle\_5}$
edl(x)	$\text{dangle}^{\_\mathtt{x}} \rightarrow \text{edl}(\mathtt{b}, \text{strong}^{\mathtt{x}}, \mathtt{l})$
edr(x)	$\text{dangle}^{\mathtt{x}\_} \rightarrow \text{edr}(\mathtt{l}, \text{strong}^{\mathtt{x}}, \mathtt{b})$
edlr(x)	$\text{dangle}^{\_\mathtt{x}\_} \rightarrow \text{edlr}(\mathtt{b}, \text{strong}^{\mathtt{x}}, \mathtt{b})$
strong(x)	$\text{strong}^{\mathtt{x}} \rightarrow \text{sr}(\mathtt{b}, \text{weak}^{\mathtt{x}}, \mathtt{b})_{bp, \neq\ LP}$
	$\text{strong}^{\mathtt{x}} \rightarrow \text{weak}^{\mathtt{x}}_{=\ LP}$
	$\text{weak}^{\mathtt{x}} \rightarrow \text{stack}^{\mathtt{x}}$
	$\text{stack}^{\mathtt{x}} \rightarrow \text{sr}(\mathtt{b}, \text{weak}^{\mathtt{x}}, \mathtt{b})_{bp}$
hairpin(a, b)	$\text{weak}^{\mathtt{[]}} \rightarrow \text{hairpin}^{\mathtt{[]}}$
	$\text{hairpin}^{\mathtt{[]}} \rightarrow \text{hl}(\mathtt{b}, \mathtt{r}_{\geq\ 3}, \mathtt{b})_{bp}$
leftbulge(a, b, x, d)	$\text{weak}^{\mathtt{[\_x]}} \rightarrow \text{leftB}^{\mathtt{[\_x]}}$
	$\text{leftB}^{\mathtt{[\_x]}} \rightarrow \text{bl}(\mathtt{b}, \mathtt{r}_{\leq\ 30}, \text{strong}^{\mathtt{x}}, \mathtt{b})_{bp}$
rightbulge(a, x, c, d)	$\text{weak}^{\mathtt{[x\_]}} \rightarrow \text{rightB}^{\mathtt{[x\_]}}$
	$\text{rightB}^{\mathtt{[x\_]}} \rightarrow \text{br}(\mathtt{b}, \text{strong}^{\mathtt{x}}, \mathtt{r}_{\leq\ 30}, \mathtt{b})_{bp}$
internalloop(a, b, x, c, d)	$\text{weak}^{\mathtt{[\_x\_]}} \rightarrow \text{iloop}^{\mathtt{[\_x\_]}}$
	$\text{iloop}^{\mathtt{[\_x\_]}} \rightarrow \text{il}(\mathtt{b}, \mathtt{r}_{\leq\ 30}, \text{strong}^{\mathtt{x}}, \mathtt{r}_{\leq\ 30}, \mathtt{b})_{bp}$
multiloop(a, x, b)	$\text{weak}^{\mathtt{x}} \rightarrow \text{multiloop}^{\mathtt{x}}$
ml(x)	$\text{multiloop}^{\mathtt{[x]}} \rightarrow \text{ml}(\mathtt{b}, \text{ml\_comps}^{\mathtt{x}}, \mathtt{b})_{bp}$
mldl(x)	$\text{multiloop}^{\mathtt{[\_x]}} \rightarrow \text{mldl}(\mathtt{b}, \mathtt{b}, \text{ml\_comps}^{\mathtt{x}}, \mathtt{b})_{bp}$
mladl(a, x)	$\text{multiloop}^{\mathtt{[\_x]}} \rightarrow \text{mldl}(\mathtt{b}, \mathtt{b}, \text{ml\_comps}^{\mathtt{x}}, \mathtt{b})_{bp}$
mldr(x)	$\text{multiloop}^{\mathtt{[x\_]}} \rightarrow \text{mldr}(\mathtt{b}, \text{ml\_comps}^{\mathtt{x}}, \mathtt{b}, \mathtt{b})_{bp}$
mladr(x, b)	$\text{multiloop}^{\mathtt{[x\_]}} \rightarrow \text{mldr}(\mathtt{b}, \text{ml\_comps}^{\mathtt{x}}, \mathtt{b}, \mathtt{b})_{bp}$
mldlr(x)	$\text{multiloop}^{\mathtt{[\_x\_]}} \rightarrow \text{mldlr}(\mathtt{b}, \mathtt{b}, \text{ml\_comps}^{\mathtt{x}}, \mathtt{b}, \mathtt{b})_{bp}$
mladldr(a, x)	$\text{multiloop}^{\mathtt{[\_x\_]}} \rightarrow \text{mldlr}(\mathtt{b}, \mathtt{b}, \text{ml\_comps}^{\mathtt{x}}, \mathtt{b}, \mathtt{b})_{bp}$
mldladr(x, b)	$\text{multiloop}^{\mathtt{[\_x\_]}} \rightarrow \text{mldlr}(\mathtt{b}, \mathtt{b}, \text{ml\_comps}^{\mathtt{x}}, \mathtt{b}, \mathtt{b})_{bp}$
mladlr(a, x, b)	$\text{multiloop}^{\mathtt{[\_x\_]}} \rightarrow \text{mldlr}(\mathtt{b}, \mathtt{b}, \text{ml\_comps}^{\mathtt{x}}, \mathtt{b}, \mathtt{b})_{bp}$
next_mlmode(x, y)	$\text{ml\_comps}^{\mathtt{xy}} \rightarrow \text{cadd}(\text{incl}(\text{dangle}^{\mathtt{x}}), \text{ml\_comps}^{\mathtt{y}})$
next_ml_r(x, a, y)	$\text{ml\_comps}^{\mathtt{x\_y}} \rightarrow \text{cadd}(\text{incl}(\text{dangle}^{\mathtt{x}}), \text{ml\_comps}^{\mathtt{y}})$
last_ml_(x)	$\text{ml\_comps}^{\mathtt{x}} \rightarrow \text{incl}(\text{dangle}^{\mathtt{x}})$
last_ml_r(x, y)	$\text{ml\_comps}^{\mathtt{xy}} \rightarrow \text{incl}(\text{dangle}^{\mathtt{x}})$
last_mlmode(x, y)	$\text{ml\_comps}^{\mathtt{xy}} \rightarrow \text{addss}(\text{incl}(\text{dangle}^{\mathtt{x}}), \mathtt{r})$
mlend_(a)	$\_$
mlnil(a)	$\_$
sadd(a, x)	$\text{struct}^{\_\mathtt{x}} \rightarrow \text{sadd}(\mathtt{b}, \text{struct}^{\_\mathtt{x}})$
	$\text{struct}^{\_\mathtt{x}} \rightarrow \text{sadd}(\mathtt{b}, \text{struct}^{\mathtt{x}})$
saddml(a, x)	$\text{ml\_comps}^{\_\mathtt{x}} \rightarrow \text{sadd}(\mathtt{b}, \text{ml\_comps}^{\_\mathtt{x}})$
	$\text{ml\_comps}^{\_\mathtt{x}} \rightarrow \text{sadd}(\mathtt{b}, \text{ml\_comps}^{\mathtt{x}})$
p(a, x)	x
objective function	$\cup_{\text{rules}}$

Shape level 5, MacroState prototype

Prototype MacroState is more complex to guarantee the semantically unambiguity regarding Vienna-Dot-Bracket strings. Therefore, TDM algebra $\mathcal{I}_{tdm\_MacroState\_5}$ (Table 9.8) uses the same algebra functions as $\mathcal{I}_{tdm\_OverDangle\_5}$, but produces much more production rules for the specialized grammar.

Table 9.8: TDM algebra for shape level 5, following the MacroState prototype grammar.

algebra function	$\mathcal{I}_{tdm\_MacroState\_5} \mathrel{-\!\rhd} \mathcal{I}_{tdm\_OverDangle\_5}$
root(x)	struct → left_dangle^{x}
	struct → trafo(noleft_dangle^{x})
	struct → left_unpaired^{x}
dangle(x)	nodangle^{x} → drem(l, strong^{x}, l)
	edanglel^{x} → edl(b, strong^{x}, l)
	edangler^{x} → edr(l, strong^{x}, b)
	edanglelr^{x} → edlr(b, strong^{x}, b)
next_hlmode(x, y)	left_unpaired^{xy} → sadd(b, left_unpaired^{xy})
	left_unpaired^{xy} → sadd(b, left_dangle^{xy})
	left_dangle^{xy} → ambd(edanglel^{x}, b, noleft_dangle^{y})
	left_dangle^{xy} → cadd’(edanglel^{x}, noleft_dangle^{y})
	left_dangle^{xy} → cadd(edanglelr^{x}, {left_dangle^{y} $\mid$ left_unpaired^{y}})
	noleft_dangle^{xy} → cadd”(edangler^{x}, {left_dangle^{y} $\mid$ left_unpaired^{y}})
	noleft_dangle^{xy} → cadd”’(nodangle^{x}, noleft_dangle^{y})
	noleft_dangle^{xy} → ambd’(nodangle^{x}, b, noleft_dangle^{y})
last_hlmode(x)	left_unpaired^{x} → sadd(b, left_unpaired^{x})
	left_unpaired^{x} → sadd(b, left_dangle^{x})
	left_dangle^{x} → cadd’(edanglel^{x}, nil(l))
	left_dangle^{x} → cadd(edanglelr^{x}, {nil(l) $\mid$ left_unpairedEnd})
	noleft_dangle^{x} → cadd”(edangler^{x}, {nil(l) $\mid$ left_unpairedEnd})
	noleft_dangle^{x} → cadd”’(nodangle^{x}, nil(l))
unpaired(a)	struct → nil(l)
	struct → left_unpairedEnd
	left_unpairedEnd → sadd(b, left_unpairedEnd)
	left_unpairedEnd → sadd(b, nil(l))
strong(x)	strong^{x} → sr(b, weak^{x}, b)$_{bp, \neq LP}$
	strong^{x} → weak$^{x}_{= LP}$
	weak^{x} → stack^{x}
	weak^{x} → leftB^{x}
	weak^{x} → rightB^{x}
	weak^{x} → iloop^{x}
	stack^{x} → sr(b, weak^{x}, b)$_{bp}$
	leftB^{x} → bl(b, r$_{\leq 30}$, strong^{x}, b)$_{bp}$
	rightB^{x} → br(b, strong^{x}, r$_{\leq 30}$, b)$_{bp}$
	iloop^{x} → il(b, r$_{\leq 30}$, strong^{x}, r$_{\leq 30}$, b)$_{bp}$
	left_unpairedEnd → sadd(b, left_unpairedEnd)

Continued on next page

Continued from previous page

algebra function	$\mathcal{I}_{tdm\_MacroState\_5} \mathrel{-\!\!\rhd} \mathcal{I}_{tdm\_OverDangle\_5}$
	left_unpairedEnd → sadd(b, nil(l))
hairpin(a, b)	weak$^{[]}$ → hairpin$^{[]}$
	hairpin$^{[]}$ → hl(b, r$_{\geq 3}$, b)$_{bp}$
multiloop(a, x, b)	weak$^{[x]}$ → multiloop$^{[x]}$
	multiloop$^{[x]}$ → ml(b, ml_comps2^{x}, b)$_{bp}$
	multiloop$^{[x]}$ → mldl(b, b, ml_comps1^{x}, b)$_{bp}$
	multiloop$^{[x]}$ → mladl(b, b, ml_comps2^{x}, b)$_{bp}$
	multiloop$^{[x]}$ → mldr(b, ml_comps3^{x}, b, b)$_{bp}$
	multiloop$^{[x]}$ → mladr(b, ml_comps2^{x}, b, b)$_{bp}$
	multiloop$^{[x]}$ → mldlr(b, b, ml_comps4^{x}, b, b)$_{bp}$
	multiloop$^{[x]}$ → mladlr(b, b, ml_comps2^{x}, b, b)$_{bp}$
	multiloop$^{[x]}$ → mldladr(b, b, ml_comps1^{x}, b, b)$_{bp}$
	multiloop$^{[x]}$ → mladldr(b, b, ml_comps3^{x}, b, b)$_{bp}$
next_mlmode(x, y)	ml_comps1^{xy} → combine(block_dl^{x}, no_dl_no_ss_end^{y})
	ml_comps1^{xy} → combine(block_dlr^{x}, dl_or_ss_left_no_ss_end^{y})
	ml_comps1^{xy} → acomb(block_dl^{x}, b, no_dl_no_ss_end^{y})
	ml_comps2^{xy} → combine(incl(nodangle^{x}), no_dl_no_ss_end^{y})
	ml_comps2^{xy} → combine(incl(edangler^{x}), dl_or_ss_left_no_ss_end^{y})
	ml_comps2^{xy} → acomb(incl(nodangle^{x}), b, no_dl_no_ss_end^{y})
	ml_comps3^{xy} → combine(incl(edangler^{x}), dl_or_ss_left_ss_end^{y})
	ml_comps3^{xy} → combine(incl(nodangle^{x}), no_dl_ss_end^{y})
	ml_comps3^{xy} → acomb(incl(nodangle^{x}), b, no_dl_ss_end^{y})
	ml_comps4^{xy} → combine(block_dl^{x}, no_dl_ss_end^{y})
	ml_comps4^{xy} → combine(block_dlr^{x}, dl_or_ss_left_ss_end^{y})
	ml_comps4^{xy} → acomb(block_dl^{x}, b, no_dl_ss_end^{y})
	no_dl_no_ss_end^{y} → ml_comps2^{y}
	dl_or_ss_left_no_ss_end^{y} → mlcomps_1^{y}
	no_dl_ss_end^{y} → ml_comps3^{y}
	dl_or_ss_left_ss_end^{y} → ml_comps4^{y}
	block_dl^{x} → ssadd(r, edanglel^{x})
	block_dl^{x} → incl(edanglel^{x})
	block_dlr^{x} → ssadd(r, edanglelr^{x})
	block_dlr^{x} → incl(edanglelr^{x})
last_mlmode(x, y)	ml_comps1^{xy} → combine(block_dl^{x}, no_dl_no_ss_end^{y})
	ml_comps1^{xy} → combine(block_dlr^{x}, dl_or_ss_left_no_ss_end^{y})
	ml_comps1^{xy} → acomb(block_dl^{x}, b, no_dl_no_ss_end^{y})
	ml_comps2^{xy} → combine(incl(nodangle^{x}), no_dl_no_ss_end^{y})
	ml_comps2^{xy} → combine(incl(edangler^{x}), dl_or_ss_left_no_ss_end^{y})
	ml_comps2^{xy} → acomb(incl(nodangle^{x}), b, no_dl_no_ss_end^{y})
	ml_comps3^{xy} → combine(incl(edangler^{x}), dl_or_ss_left_ss_end^{y})
	ml_comps3^{xy} → combine(incl(nodangle^{x}), no_dl_ss_end^{y})
	ml_comps3^{xy} → acomb(incl(nodangle^{x}), b, no_dl_ss_end^{y})
	ml_comps4^{xy} → combine(block_dl^{x}, no_dl_ss_end^{y})

Continued on next page

Table 9.9: TDM algebra for shape level 4, following the MacroState prototype grammar.

algebra function	$\mathcal{I}_{tdm\_MacroState\_4} \;-\!\!\rhd\; \mathcal{I}_{tdm\_MacroState\_5}$
$\text{strong}(x)$	$\text{strong}^{\mathtt{x}} \rightarrow \text{sr}(\mathtt{b}, \text{weak}^{\mathtt{x}}, \mathtt{b})_{bp, \neq LP}$
	$\text{strong}^{\mathtt{x}} \rightarrow \text{weak}^{\mathtt{x}}_{= LP}$
	$\text{weak}^{\mathtt{x}} \rightarrow \text{stack}^{\mathtt{x}}$
	$\text{weak}^{\mathtt{x}} \rightarrow \text{leftB}^{\mathtt{x}}$
	$\text{weak}^{\mathtt{x}} \rightarrow \text{rightB}^{\mathtt{x}}$
	$\text{stack}^{\mathtt{x}} \rightarrow \text{sr}(\mathtt{b}, \text{weak}^{\mathtt{x}}, \mathtt{b})_{bp}$
	$\text{leftB}^{\mathtt{x}} \rightarrow \text{bl}(\mathtt{b}, \mathtt{r}_{\leq 30}, \text{strong}^{\mathtt{x}}, \mathtt{b})_{bp}$
	$\text{rightB}^{\mathtt{x}} \rightarrow \text{br}(\mathtt{b}, \text{strong}^{\mathtt{x}}, \mathtt{r}_{\leq 30}, \mathtt{b})_{bp}$
	$\text{left\_unpairedEnd} \rightarrow \text{sadd}(\mathtt{b}, \text{left\_unpairedEnd})$
	$\text{left\_unpairedEnd} \rightarrow \text{sadd}(\mathtt{b}, \text{nil}(\mathtt{l}))$
$\text{helixinterrupt}(a, x, b)$	$\text{weak}^{[\mathtt{x}]} \rightarrow \text{iloop}^{[\mathtt{x}]}$
	$\text{iloop}^{[\mathtt{x}]} \rightarrow \text{il}(\mathtt{b}, \mathtt{r}_{\leq 30}, \text{strong}^{\mathtt{x}}, \mathtt{r}_{\leq 30}, \mathtt{b})_{bp}$

Continued from previous page

algebra function	$\mathcal{I}_{tdm\_MacroState\_5} \;-\!\!\rhd\; \mathcal{I}_{tdm\_OverDangle\_5}$
	$\text{ml\_comps4}^{\mathtt{xy}} \rightarrow \text{combine}(\text{block\_dlr}^{\mathtt{x}}, \text{dl\_or\_ss\_left\_ss\_end}^{\mathtt{y}})$
	$\text{ml\_comps4}^{\mathtt{xy}} \rightarrow \text{acomb}(\text{block\_dl}^{\mathtt{x}}, \mathtt{b}, \text{no\_dl\_ss\_end}^{\mathtt{y}})$
	$\text{no\_dl\_no\_ss\_end}^{\mathtt{y}} \rightarrow \text{incl}(\text{nodangle}^{\mathtt{y}})$
	$\text{dl\_or\_ss\_left\_no\_ss\_end}^{\mathtt{y}} \rightarrow \text{block\_dl}^{\mathtt{y}}$
	$\text{no\_dl\_ss\_end}^{\mathtt{y}} \rightarrow \text{incl}(\text{edangler}^{\mathtt{y}})$
	$\text{no\_dl\_ss\_end}^{\mathtt{y}} \rightarrow \text{addss}(\text{incl}(\text{edangler}^{\mathtt{y}}), \mathtt{r})$
	$\text{dl\_or\_ss\_left\_ss\_end}^{\mathtt{y}} \rightarrow \text{block\_dlr}^{\mathtt{y}}$
	$\text{dl\_or\_ss\_left\_ss\_end}^{\mathtt{y}} \rightarrow \text{addss}(\text{block\_dlr}^{\mathtt{y}}, \mathtt{r})$
	$\text{block\_dl}^{\mathtt{x}} \rightarrow \text{ssadd}(\mathtt{r}, \text{edanglel}^{\mathtt{x}})$
	$\text{block\_dl}^{\mathtt{x}} \rightarrow \text{incl}(\text{edanglel}^{\mathtt{x}})$
	$\text{block\_dlr}^{\mathtt{x}} \rightarrow \text{ssadd}(\mathtt{r}, \text{edanglelr}^{\mathtt{x}})$
	$\text{block\_dlr}^{\mathtt{x}} \rightarrow \text{incl}(\text{edanglelr}^{\mathtt{x}})$
	$\text{block\_dl}^{\mathtt{y}} \rightarrow \text{ssadd}(\mathtt{r}, \text{edanglel}^{\mathtt{y}})$
	$\text{block\_dl}^{\mathtt{y}} \rightarrow \text{incl}(\text{edanglel}^{\mathtt{y}})$
	$\text{block\_dlr}^{\mathtt{y}} \rightarrow \text{ssadd}(\mathtt{r}, \text{edanglelr}^{\mathtt{y}})$
	$\text{block\_dlr}^{\mathtt{y}} \rightarrow \text{incl}(\text{edanglelr}^{\mathtt{y}})$

Shape levels 4 to 2, MacroState prototype

The three TDM algebras $\mathcal{I}_{tdm\_MacroState\_4}$ (Table 9.9), $\mathcal{I}_{tdm\_MacroState\_3}$ (Table 9.10) and $\mathcal{I}_{tdm\_MacroState\_2}$ (Table 9.11) follow the same helix interruption arguments for distributing the same rules to different algebra functions as their OverDangle counterparts.

Table 9.10: TDM algebra for shape level 3, following the MacroState prototype grammar.

algebra function	$\mathcal{I}_{tdm\_MacroState\_3} -\!\!\rhd \mathcal{I}_{tdm\_MacroState\_5}$
$\mathrm{strong}(x)$	$\mathrm{strong}^{\texttt{x}} \to \mathrm{sr}(\texttt{b}, \mathrm{weak}^{\texttt{x}}, \texttt{b})_{bp, \neq LP}$
	$\mathrm{strong}^{\texttt{x}} \to \mathrm{weak}^{\texttt{x}}_{= LP}$
	$\mathrm{weak}^{\texttt{x}} \to \mathrm{stack}^{\texttt{x}}$
	$\mathrm{stack}^{\texttt{x}} \to \mathrm{sr}(\texttt{b}, \mathrm{weak}^{\texttt{x}}, \texttt{b})_{bp}$
	$\mathrm{left\_unpairedEnd} \to \mathrm{sadd}(\texttt{b}, \mathrm{left\_unpairedEnd})$
	$\mathrm{left\_unpairedEnd} \to \mathrm{sadd}(\texttt{b}, \mathrm{nil}(\texttt{l}))$
$\mathrm{helixinterrupt}(a, x, b)$	$\mathrm{weak}^{\texttt{[x]}} \to \mathrm{leftB}^{\texttt{[x]}}$
	$\mathrm{weak}^{\texttt{[x]}} \to \mathrm{rightB}^{\texttt{[x]}}$
	$\mathrm{weak}^{\texttt{[x]}} \to \mathrm{iloop}^{\texttt{[x]}}$
	$\mathrm{leftB}^{\texttt{[x]}} \to \mathrm{bl}(\texttt{b}, \texttt{r}_{\leq 30}, \mathrm{strong}^{\texttt{x}}, \texttt{b})_{bp}$
	$\mathrm{rightB}^{\texttt{[x]}} \to \mathrm{br}(\texttt{b}, \mathrm{strong}^{\texttt{x}}, \texttt{r}_{\leq 30}, \texttt{b})_{bp}$
	$\mathrm{iloop}^{\texttt{[x]}} \to \mathrm{il}(\texttt{b}, \texttt{r}_{\leq 30}, \mathrm{strong}^{\texttt{x}}, \texttt{r}_{\leq 30}, \texttt{b})_{bp}$

Table 9.11: TDM algebra for shape level 2, following the MacroState prototype grammar.

algebra function	$\mathcal{I}_{tdm\_MacroState\_2} -\!\!\rhd \mathcal{I}_{tdm\_MacroState\_5}$
$\mathrm{strong}(x)$	$\mathrm{strong}^{\texttt{x}} \to \mathrm{sr}(\texttt{b}, \mathrm{weak}^{\texttt{x}}, \texttt{b})_{bp, \neq LP}$
	$\mathrm{strong}^{\texttt{x}} \to \mathrm{weak}^{\texttt{x}}_{= LP}$
	$\mathrm{weak}^{\texttt{x}} \to \mathrm{stack}^{\texttt{x}}$
	$\mathrm{stack}^{\texttt{x}} \to \mathrm{sr}(\texttt{b}, \mathrm{weak}^{\texttt{x}}, \texttt{b})_{bp}$
	$\mathrm{left\_unpairedEnd} \to \mathrm{sadd}(\texttt{b}, \mathrm{left\_unpairedEnd})$
	$\mathrm{left\_unpairedEnd} \to \mathrm{sadd}(\texttt{b}, \mathrm{nil}(\texttt{l}))$
$\mathrm{leftbulge}(a, b, x, d)$	$\mathrm{weak}^{\texttt{[\_x]}} \to \mathrm{leftB}^{\texttt{[\_x]}}$
	$\mathrm{leftB}^{\texttt{[\_x]}} \to \mathrm{bl}(\texttt{b}, \texttt{r}_{\leq 30}, \mathrm{strong}^{\texttt{x}}, \texttt{b})_{bp}$
$\mathrm{rightbulge}(a, x, c, d)$	$\mathrm{weak}^{\texttt{[x\_]}} \to \mathrm{rightB}^{\texttt{[x\_]}}$
	$\mathrm{rightB}^{\texttt{[x\_]}} \to \mathrm{br}(\texttt{b}, \mathrm{strong}^{\texttt{x}}, \texttt{r}_{\leq 30}, \texttt{b})_{bp}$
$\mathrm{internalloop}(a, b, x, c, d)$	$\mathrm{weak}^{\texttt{[\_x\_]}} \to \mathrm{iloop}^{\texttt{[\_x\_]}}$
	$\mathrm{iloop}^{\texttt{[\_x\_]}} \to \mathrm{il}(\texttt{b}, \texttt{r}_{\leq 30}, \mathrm{strong}^{\texttt{x}}, \texttt{r}_{\leq 30}, \texttt{b})_{bp}$

Shape level 1, MacroState prototype

TDM algebra $\mathcal{I}_{tdm\_MacroState\_1}$ (Table 9.12) has to deal with the syntactic ambiguity of the according TDM grammar $\mathcal{G}_{\mathrm{TDM}_1^{\mathrm{MacroState}}}$, which is the reason for the merging choice function $\cup_{\mathrm{rules}}$.

Table 9.12: TDM algebra for shape level 1, following the MacroState prototype grammar.

algebra function	$\mathcal{I}_{tdm\_MacroState\_1} -\!\!\triangleright \mathcal{I}_{tdm\_MacroState\_5}$
unpaired_macrostate(x)	struct → nil($\texttt{l}$)
	struct → left_unpairedEnd
root(x)	struct → left_dangle$^{\texttt{x}}$
	struct → left_unpaired$^{\texttt{x}}$
trafo(x)	struct → trafo(noleft_dangle$^{\texttt{x}}$)
nextambd(a, x, b, y)	left_dangle$^{\texttt{\_x\_y}}$ → ambd(edanglel$^{\texttt{x}}$, $\texttt{b}$, noleft_dangle$^{\texttt{y}}$)
nextcadda(a, x, y)	left_dangle$^{\texttt{\_xy}}$ → cadd'(edanglel$^{\texttt{x}}$, noleft_dangle$^{\texttt{y}}$)
nextcadd(a, x, y)	left_dangle$^{\texttt{\_xy}}$ → cadd(edanglelr$^{\texttt{x}}$, {left_dangle$^{\texttt{y}}$ $\mid$ left_unpaired$^{\texttt{y}}$})
lastcadda(a, x)	left_dangle$^{\texttt{\_x}}$ → cadd'(edanglel$^{\texttt{x}}$, nil($\texttt{l}$))
lastcadd(a, x, y)	left_dangle$^{\texttt{\_xy}}$ → cadd(edanglelr$^{\texttt{x}}$, {nil($\texttt{l}$) $\mid$ left_unpairedEnd})
next_hlmode(x, y)	noleft_dangle$^{\texttt{xy}}$ → cadd''(edangler$^{\texttt{x}}$, {left_dangle$^{\texttt{y}}$ $\mid$ left_unpaired$^{\texttt{y}}$})
nextcaddc(x, y)	noleft_dangle$^{\texttt{xy}}$ → cadd'''(nodangle$^{\texttt{x}}$, noleft_dangle$^{\texttt{y}}$)
nextambda(x, a, y)	noleft_dangle$^{\texttt{x\_y}}$ → ambd'(nodangle$^{\texttt{x}}$, $\texttt{b}$, noleft_dangle$^{\texttt{y}}$)
lastcaddb(x, y)	noleft_dangle$^{\texttt{xy}}$ → cadd''(edangler$^{\texttt{x}}$, {nil($\texttt{l}$) $\mid$ left_unpairedEnd})
last_hlmode(x)	noleft_dangle$^{\texttt{x}}$ → cadd'''(nodangle$^{\texttt{x}}$, nil($\texttt{l}$))
hairpin(a, b)	weak$^{\texttt{[]}}$ → hairpin$^{\texttt{[]}}$
	hairpin$^{\texttt{[]}}$ → hl($\texttt{b}$, $\texttt{r}_{\geq 3}$, $\texttt{b}$)$_{bp}$
leftbulge(a, b, x, d)	weak$^{\texttt{[\_x]}}$ → leftB$^{\texttt{[\_x]}}$
	leftB$^{\texttt{[\_x]}}$ → bl($\texttt{b}$, $\texttt{r}_{\leq 30}$, strong$^{\texttt{x}}$, $\texttt{b}$)$_{bp}$
rightbulge(a, x, c, d)	weak$^{\texttt{[x\_]}}$ → rightB$^{\texttt{[x\_]}}$
	rightB$^{\texttt{[x\_]}}$ → br($\texttt{b}$, strong$^{\texttt{x}}$, $\texttt{r}_{\leq 30}$, $\texttt{b}$)$_{bp}$
internalloop(a, b, x, c, d)	weak$^{\texttt{[\_x\_]}}$ → iloop$^{\texttt{[\_x\_]}}$
	iloop$^{\texttt{[\_x\_]}}$ → il($\texttt{b}$, $\texttt{r}_{\leq 30}$, strong$^{\texttt{x}}$, $\texttt{r}_{\leq 30}$, $\texttt{b}$)$_{bp}$
multiloop(a, x, b)	weak$^{\texttt{x}}$ → multiloop$^{\texttt{x}}$
ml(x)	multiloop$^{\texttt{[x]}}$ → ml($\texttt{b}$, ml_comps2$^{\texttt{x}}$, $\texttt{b}$)$_{bp}$
mldl(x)	multiloop$^{\texttt{[x]}}$ → mldl($\texttt{b}$, $\texttt{b}$, ml_comps1$^{\texttt{x}}$, $\texttt{b}$)$_{bp}$
mladl(a, x)	multiloop$^{\texttt{[\_x]}}$ → mladl($\texttt{b}$, $\texttt{b}$, ml_comps2$^{\texttt{x}}$, $\texttt{b}$)$_{bp}$

Continued on next page

Continued from previous page

algebra function	$\mathcal{I}_{tdm\_MacroState\_1} \mathrel{-\!\!\rhd} \mathcal{I}_{tdm\_MacroState\_5}$
$\text{mldr}(x)$	$\text{multiloop}^{\texttt{[x]}} \rightarrow \text{mldr}(\texttt{b}, \text{ml\_comps3}^{\texttt{x}}, \texttt{b}, \texttt{b})_{bp}$
$\text{mladr}(x, a)$	$\text{multiloop}^{\texttt{[x\_]}} \rightarrow \text{mladr}(\texttt{b}, \text{ml\_comps2}^{\texttt{x}}, \texttt{b}, \texttt{b})_{bp}$
$\text{mldlr}(x)$	$\text{multiloop}^{\texttt{[x]}} \rightarrow \text{mldlr}(\texttt{b}, \texttt{b}, \text{ml\_comps4}^{\texttt{x}}, \texttt{b}, \texttt{b})_{bp}$
$\text{mladldr}(a, x)$	$\text{multiloop}^{\texttt{[\_x]}} \rightarrow \text{mladldr}(\texttt{b}, \texttt{b}, \text{ml\_comps3}^{\texttt{x}}, \texttt{b}, \texttt{b})_{bp}$
$\text{mldladr}(x, a)$	$\text{multiloop}^{\texttt{[x\_]}} \rightarrow \text{mldladr}(\texttt{b}, \texttt{b}, \text{ml\_comps1}^{\texttt{x}}, \texttt{b}, \texttt{b})_{bp}$
$\text{mladlr}(a, x, b)$	$\text{multiloop}^{\texttt{[\_x\_]}} \rightarrow \text{mladlr}(\texttt{b}, \texttt{b}, \text{ml\_comps2}^{\texttt{x}}, \texttt{b}, \texttt{b})_{bp}$
$\text{combine1\_a}(a, x, y)$	$\text{ml\_comps1}^{\texttt{\_xy}} \rightarrow \text{combine}(\text{block\_dl}^{\texttt{x}}, \text{no\_dl\_no\_ss\_end}^{\texttt{y}})$
$\text{nextcombine1\_b}(a, x, y)$	$\text{ml\_comps1}^{\texttt{\_xy}} \rightarrow \text{combine}(\text{block\_dlr}^{\texttt{x}}, \text{dl\_or\_ss\_left\_no\_ss\_end}^{\texttt{y}})$
$\text{comb1\_a}(a, x, b, y)$	$\text{ml\_comps1}^{\texttt{\_x\_y}} \rightarrow \text{acomb}(\text{block\_dl}^{\texttt{x}}, \texttt{b}, \text{no\_dl\_no\_ss\_end}^{\texttt{y}})$
$\text{lastcombine1\_b}(a, x, b, y)$	$\text{ml\_comps1}^{\texttt{\_x\_y}} \rightarrow \text{combine}(\text{block\_dlr}^{\texttt{x}}, \text{dl\_or\_ss\_left\_no\_ss\_end}^{\texttt{y}})$
$\text{next\_mlmode}(x, y)$	$\text{ml\_comps2}^{\texttt{xy}} \rightarrow \text{combine}(\text{incl}(\text{nodangle}^{\texttt{x}}), \text{no\_dl\_no\_ss\_end}^{\texttt{y}})$
$\text{nextcombine2\_b}(x, y)$	$\text{ml\_comps2}^{\texttt{xy}} \rightarrow \text{combine}(\text{incl}(\text{edangler}^{\texttt{x}}), \text{dl\_or\_ss\_left\_no\_ss\_end}^{\texttt{y}})$
$\text{acomb2\_a}(x, a, y)$	$\text{ml\_comps2}^{\texttt{x\_y}} \rightarrow \text{acomb}(\text{incl}(\text{nodangle}^{\texttt{x}}), \texttt{b}, \text{no\_dl\_no\_ss\_end}^{\texttt{y}})$
$\text{lastcombine2\_b}(x, a, y)$	$\text{ml\_comps2}^{\texttt{x\_y}} \rightarrow \text{combine}(\text{incl}(\text{edangler}^{\texttt{x}}), \text{dl\_or\_ss\_left\_no\_ss\_end}^{\texttt{y}})$
$\text{nextcombine3\_a}(x, y)$	$\text{ml\_comps3}^{\texttt{xy}} \rightarrow \text{combine}(\text{incl}(\text{edangler}^{\texttt{x}}), \text{dl\_or\_ss\_left\_ss\_end}^{\texttt{y}})$
$\text{nextcombine3\_b}(x, y)$	$\text{ml\_comps3}^{\texttt{xy}} \rightarrow \text{combine}(\text{incl}(\text{nodangle}^{\texttt{x}}), \text{no\_dl\_ss\_end}^{\texttt{y}})$
$\text{nextacomb3\_a}(x, a, y)$	$\text{ml\_comps3}^{\texttt{x\_y}} \rightarrow \text{acomb}(\text{incl}(\text{nodangle}^{\texttt{x}}), \texttt{b}, \text{no\_dl\_ss\_end}^{\texttt{y}})$
$\text{lastcombine3\_a}(x, a, y, b)$	$\text{ml\_comps3}^{\texttt{x\_y\_}} \rightarrow \text{combine}(\text{incl}(\text{edangler}^{\texttt{x}}), \text{dl\_or\_ss\_left\_ss\_end}^{\texttt{y}})$
$\text{lastcombine3\_b}(x, y, b)$	$\text{ml\_comps3}^{\texttt{xy\_}} \rightarrow \text{combine}(\text{incl}(\text{nodangle}^{\texttt{x}}), \text{no\_dl\_ss\_end}^{\texttt{y}})$
$\text{lastacomb3\_a}(x, a, y, b)$	$\text{ml\_comps3}^{\texttt{x\_y\_}} \rightarrow \text{acomb}(\text{incl}(\text{nodangle}^{\texttt{x}}), \texttt{b}, \text{no\_dl\_ss\_end}^{\texttt{y}})$
$\text{nextcombine4\_a}(a, x, y)$	$\text{ml\_comps4}^{\texttt{\_xy}} \rightarrow \text{combine}(\text{block\_dl}^{\texttt{x}}, \text{no\_dl\_ss\_end}^{\texttt{y}})$
$\text{nextcombine4\_b}(a, x, y)$	$\text{ml\_comps4}^{\texttt{\_xy}} \rightarrow \text{combine}(\text{block\_dlr}^{\texttt{x}}, \text{dl\_or\_ss\_left\_ss\_end}^{\texttt{y}})$
$\text{nextacomb4\_a}(a, x, b, y)$	$\text{ml\_comps4}^{\texttt{\_x\_y}} \rightarrow \text{acomb}(\text{block\_dl}^{\texttt{x}}, \texttt{b}, \text{no\_dl\_ss\_end}^{\texttt{y}})$
$\text{lastcombine4\_a}(a, x, y, b)$	$\text{ml\_comps4}^{\texttt{\_xy\_}} \rightarrow \text{combine}(\text{block\_dl}^{\texttt{x}}, \text{no\_dl\_ss\_end}^{\texttt{y}})$
$\text{lastcombine4\_b}(a, x, b, y, c)$	$\text{ml\_comps4}^{\texttt{\_x\_y\_}} \rightarrow \text{combine}(\text{block\_dlr}^{\texttt{x}}, \text{dl\_or\_ss\_left\_ss\_end}^{\texttt{y}})$
$\text{lastacomb4\_a}(x, a, y, b, c)$	$\text{ml\_comps4}^{\texttt{\_x\_y\_}} \rightarrow \text{acomb}(\text{block\_dl}^{\texttt{x}}, \texttt{b}, \text{no\_dl\_ss\_end}^{\texttt{y}})$
$\text{strong}(x)$	$\text{strong}^{\texttt{x}} \rightarrow \text{sr}(\texttt{b}, \text{weak}^{\texttt{x}}, \texttt{b})_{bp, \neq LP}$
	$\text{strong}^{\texttt{x}} \rightarrow \text{weak}^{\texttt{x}}_{= LP}$

Continued on next page

Continued from previous page

algebra function	$\mathcal{I}_{tdm\_MacroState\_1} \mathrel{-\!\!\rhd} \mathcal{I}_{tdm\_MacroState\_5}$
	weak^{x} → stack^{x}
	stack^{x} → sr(b, weak^{x}, b)$_{bp}$
drem(x)	nodangle^{x} → drem(l, strong^{x}, l)
edl(x)	edanglel^{x} → edl(b, strong^{x}, l)
edr(x)	edangler^{x} → edr(l, strong^{x}, b)
edlr(x)	edanglelr^{x} → edlr(b, strong^{x}, b)
block_dl(x)	block_dl^{x} → ssadd(r, edanglel^{x})
	block_dl^{x} → incl(edanglel^{x})
block_dlr(x)	block_dlr^{x} → ssadd(r, edanglelr^{x})
	block_dlr^{x} → incl(edanglelr^{x})
no_dl_ss_end__next(x)	no_dl_ss_end^{x} → ml_comps3^{x}
no_dl_ss_end__last(x)	no_dl_ss_end^{x} → addss(incl(edangler^{x}), r)
	no_dl_ss_end^{x} → incl(edangler^{x})
no_dl_no_ss_end__next(x)	no_dl_no_ss_end^{x} → ml_comps2^{x}
no_dl_no_ss_end__last(x)	no_dl_no_ss_end^{x} → incl(nodangle^{x})
dl_or_ss_left_no_ss_end__next(x)	dl_or_ss_left_no_ss_end^{x} → ml_comps1^{x}
dl_or_ss_left_no_ss_end__last(x)	dl_or_ss_left_no_ss_end^{x} → block_dl^{x}
dl_or_ss_left_ss_end__next(x)	dl_or_ss_left_ss_end^{x} → ml_comps4^{x}
dl_or_ss_left_ss_end__last(x)	dl_or_ss_left_ss_end^{x} → block_dlr^{x}
	dl_or_ss_left_ss_end^{x} → addss(block_dlr^{x}, r)
leftunpairedend(a)	left_unpairedEnd → sadd(b, left_unpairedEnd)
	left_unpairedEnd → sadd(b, nil(l))
leftunpaired(a)	left_unpaired^{x} → sadd(b, left_unpaired^{x})
	left_unpaired^{x} → sadd(b, left_dangle^{x})
objective function	$\cup_{\text{rules}}$

9.2 Appendix B: Source code for pKiss pseudoknot non-terminals.

Following is the Bellman's GAP source code for the pseudoknot non-terminals. Compared to Section 5.3, index ranges and index ordering might look different. Here, we optimize for run-time efficiency, in Section 5.3 we tried to ease understanding. At the end, both versions describe exactly the same iteration about those indices.

We apply one further hack to same run-time. In principle, an ADP grammar should never be aware of any scoring scheme, because the scoring is a matter of the algebra, which we might replace by other algebras. However, the stabilizing energy for the pseudoknot stems is pre-computed in the $O(n^2)$ DP table "stacklen". We transfer this result from the grammar, i. e. the non-terminal construction, to the algebra functions pknot and pkiss as the additional parameter "stackenergies". Furthermore, it relieves us of the burden to push the four to six inner stem boundaries into the algebras.

The lines INNER(CODE) are the boundary specific calls to the children terminals / non-terminals, as described in Figure 5.7.

```
pknot_free_hk =
.[
 int i = t_0_i;
 int j = t_0_j;
 if ((i+11 <= j) && (j-i <= maxPseudoknotSize())) {
  for (int k = i+7; k <= j-4; k=k+1) {
   int alphamaxlen = length(stacklen(t_0_seq, i, k));
   if (alphamaxlen < 2) continue;
   for (int h = i+3; h <= k-4; h=h+1) {
    int alphareallen = min(k-h-2, min(alphamaxlen, h-i-1));
    if (alphareallen < 2) continue;
    int betamaxlen = length(stacklen(t_0_seq, h, j));
    if (betamaxlen < 2) continue;
    int betareallen = min(min(betamaxlen, j-k-2), k-h-alphareallen);
    if (betareallen < 2) continue;
    int stackenergies =
          energy(stacklen(t_0_seq, i,                 k                ))
        + energy(stacklen(t_0_seq, h,                 j                ))
        - energy(stacklen(t_0_seq, i+alphareallen-1, k-alphareallen+1))
        - energy(stacklen(t_0_seq, h+betareallen -1, j-betareallen +1));
    INNER(CODE);
   }
  }
 }
].


pknot_free_hk_3D =
.[
 int i = t_0_i;
 int j = t_0_j;
 if ((i+4*2+3 <= j)  && (j-i <= maxPseudoknotSize())) {
  for (int k = i+3*2+1; k <= j-2*2; k=k+1) {
   int alphamaxlen = length(stacklen(t_0_seq, i, k));
   if (alphamaxlen < 2) continue;
   for (int h = i+2+1; h <= k-2*2; h=h+1) {
    int alphareallen = min(k-h-2, min(alphamaxlen, h-i-1));
    if (alphareallen < 2) continue;
    int betamaxlen = length(stacklen(t_0_seq, h, j));
    if (betamaxlen < 2) continue;
    int betareallen = min(min(betamaxlen, j-k-2), k-h-alphareallen);
    if (betareallen < 2) continue;
    int stackenergies =
          energy(stacklen(t_0_seq, i,               k               ))
        + energy(stacklen(t_0_seq, h,               j               ))
        - energy(stacklen(t_0_seq, i+alphareallen-1, k-alphareallen+1))
        - energy(stacklen(t_0_seq, h+betareallen -1, j-betareallen +1));
    INNER(CODE);
    int n = size(t_0_seq);
    if (!isEmpty(answers)) {
      answer_pknot_mfe mfe = get_pk(i,j,h,k);
      if (mfe.energy < get_energy(subopt_left, i, j, h, n))
        set(subopt_left, i, j, h, k, mfe.energy, n);
      int splitPositionLeft = h+(j-h)/2;
      if ((h <= splitPositionLeft) &&
          (mfe.energy < get_energy(subopt_left_heuristic, i, j, h, n)))
        set(subopt_left_heuristic, i, j, h, k, mfe.energy, n);
      if (mfe.energy < get_energy(subopt_right, i, j, k, n))
        set(subopt_right, i, j, k, h, mfe.energy, n);
      int splitPositionRight = i+(k-i)/2;
      if ((k >= splitPositionRight) &&
          (mfe.energy < get_energy(subopt_right_heuristic, i, j, k, n)))
        set(subopt_right_heuristic, i, j, k, h, mfe.energy, n);
    }
   }
  }
 }
].

pknot_free_h(int kk, int startH) =
.[
 int i = t_0_i;
 int j = t_0_j;
 int k = kk;
 if ((i+11 <= j) && (j-i <= maxPseudoknotSize())) {
  int alphamaxlen = length(stacklen(t_0_seq, i, k));
  if (alphamaxlen >= 2) {
   for (int h = startH; h <= k-4; h=h+1) {
    int alphareallen = min(k-h-2, min(alphamaxlen, h-i-1));
    if (alphareallen < 2) continue;
    int betamaxlen = length(stacklen(t_0_seq, h, j));
    if (betamaxlen < 2) continue;
    int betareallen = min(min(betamaxlen, j-k-2), k-h-alphareallen);
    if (betareallen < 2) continue;
    int stackenergies =
          energy(stacklen(t_0_seq,i,               k               ))
        + energy(stacklen(t_0_seq,h,               j               ))
        - energy(stacklen(t_0_seq,i+alphareallen-1,k-alphareallen+1))
        - energy(stacklen(t_0_seq,h+betareallen -1,j-betareallen +1));
    INNER(CODE);
   }
  }
 }
].

pknot_free_k(int h, int endK) =
.[
 int i = t_0_i;
 int j = t_0_j;
 if ((i+11 <= j) && (j-i <= maxPseudoknotSize())) {
  int betamaxlen = length(stacklen(t_0_seq, h, j));
  if (betamaxlen >= 2) {
   for (int k = h+4; k <= endK; k=k+1) {
    int alphamaxlen = length(stacklen(t_0_seq, i, k));
    if (alphamaxlen < 2) continue;
    int alphareallen = min(k-h-2, min(alphamaxlen, h-i-1));
    if (alphareallen < 2) continue;
    int betareallen = min(min(betamaxlen, j-k-2), k-h-alphareallen);
    if (betareallen < 2) continue;
    int stackenergies =
          energy(stacklen(t_0_seq,i,               k               ))
        + energy(stacklen(t_0_seq,h,               j               ))
        - energy(stacklen(t_0_seq,i+alphareallen-1,k-alphareallen+1))
        - energy(stacklen(t_0_seq,h+betareallen -1,j-betareallen +1));
    INNER(CODE);
   }
  }
 }
].

pknot(int h, int kindex) =
.[
 int i = t_0_i;
 int j = t_0_j;
 if ((i+2+1<=h && h+2*2<=kindex && kindex+2+2<=j) && (j-i <= maxPseudoknotSize())) {
  int betamaxlen = length(stacklen(t_0_seq, h, j));
  if (betamaxlen >= 2) {
   int alphamaxlen = length(stacklen(t_0_seq, i, kindex));
   if (alphamaxlen >= 2) {
    int alphareallen = min(kindex-h-2, min(alphamaxlen, h-i-1));
    if (alphareallen >= 2) {
     int betareallen = min(min(betamaxlen, j-kindex-2), kindex-h-alphareallen);
     if (betareallen >= 2) {
         int stackenergies =
               energy(stacklen(t_0_seq,i,               kindex               ))
             + energy(stacklen(t_0_seq,h,               j                    ))
             - energy(stacklen(t_0_seq,i+alphareallen-1,kindex-alphareallen+1))
             - energy(stacklen(t_0_seq,h+betareallen -1,j-betareallen +1     ));
         INNER(CODE);
     }
    }
   }
  }
 }
].

pkiss_Aleft  =
.[
 int i = t_0_i;
 int j = t_0_j;
 if (j-i <= maxPseudoknotSize()) {
  for (int m = i+3*minLengthKissingHairpinStems()+7;
       m<=j-minLengthKissingHairpinStems()-1;
       m=m+1) {
   answer_pknot_mfe leftPK = get_pk_free_hk(i, m);
   if (isEmpty(leftPK)) continue;
   int h = leftPK.betaLeftOuter;
   int k = leftPK.alphaRightOuter;
   int alphamaxlen = length(stacklen(t_0_seq, i, k));
   if (alphamaxlen < minLengthKissingHairpinStems()) continue;
   int alphareallen = min(alphamaxlen, h-i-1);
   if (alphareallen < minLengthKissingHairpinStems()) continue;
   int betamaxlen = length(stacklen(t_0_seq, h, m));
   if (betamaxlen < 2) continue;
   answer_pknot_mfe rightPK = get_pk_free_h(h, j, m, k+2);
   if (isEmpty(rightPK)) continue;
   int l = rightPK.betaLeftOuter;
   int gammamaxlen = length(stacklen(t_0_seq, l, j));
   if (gammamaxlen < minLengthKissingHairpinStems()) continue;
   int gammareallen = min(gammamaxlen, j-m-1);
   if (gammareallen < minLengthKissingHairpinStems()) continue;
   int betareallen = min(min(betamaxlen, k-h-alphareallen),
                         min(betamaxlen, m-l-gammareallen));
   if (betareallen < 2) continue;
   int stackenergies = energy(stacklen(t_0_seq,i,               k               ))
```

```
                     + energy(stacklen(t_0_seq,h,                 m                ))
                     + energy(stacklen(t_0_seq,l,                 j                ))
                     - energy(stacklen(t_0_seq,i+alphareallen-1,k-alphareallen+1))
                     - energy(stacklen(t_0_seq,h+betareallen  -1,m-betareallen  +1))
                     - energy(stacklen(t_0_seq,l+gammareallen-1,j-gammareallen+1));
   INNER(CODE);
  }
 }
].

pkiss_Aright   =
.[
 int i = t_0_i;
 int j = t_0_j;
 if (j-i <= maxPseudoknotSize()) {
  for (int h = i+minLengthKissingHairpinStems()+1;
       h<=j-3*minLengthKissingHairpinStems()-7;
       h=h+1) {
   answer_pknot_mfe rightPK = get_pk_free_hk(h, j);
   if (isEmpty(rightPK)) continue;
   int l = rightPK.betaLeftOuter;
   int m = rightPK.alphaRightOuter;
   int gammamaxlen = length(stacklen(t_0_seq, l, j));
   if (gammamaxlen < minLengthKissingHairpinStems()) continue;
   int gammareallen = min(gammamaxlen, j-m-1);
   if (gammareallen < minLengthKissingHairpinStems()) continue;
   int betamaxlen = length(stacklen(t_0_seq, h, m));
   if (betamaxlen < 2) continue;
   answer_pknot_mfe leftPK = get_pk_free_k(i, m, h, l-2);
   if (isEmpty(leftPK)) continue;
   int k = leftPK.alphaRightOuter;
   int alphamaxlen = length(stacklen(t_0_seq, i, k));
   if (alphamaxlen < minLengthKissingHairpinStems()) continue;
   int alphareallen = min(alphamaxlen, h-i-1);
   if (alphareallen < minLengthKissingHairpinStems()) continue;
   int betareallen = min(min(betamaxlen, k-h-alphareallen),
                         min(betamaxlen, m-l-gammareallen));
   if (betareallen < 2) continue;
   int stackenergies = energy(stacklen(t_0_seq,i,                 k                ))
                     + energy(stacklen(t_0_seq,h,                 m                ))
                     + energy(stacklen(t_0_seq,l,                 j                ))
                     - energy(stacklen(t_0_seq,i+alphareallen-1,k-alphareallen+1))
                     - energy(stacklen(t_0_seq,h+betareallen  -1,m-betareallen  +1))
                     - energy(stacklen(t_0_seq,l+gammareallen-1,j-gammareallen+1));
   INNER(CODE);
  }
 }
].

pkiss_B(bool useSplitpoint)   =
.[
 int i = t_0_i;
 int j = t_0_j;
 if (j-i <= maxPseudoknotSize()) {
  for (int m = i+3+3*minLengthKissingHairpinStems()+2*2;
       m<=j-minLengthKissingHairpinStems()-1;
       m=m+1) {
   for (int h = i+minLengthKissingHairpinStems()+1;
        h<=m-2*minLengthKissingHairpinStems()-2*2-2;
        h=h+1) {
    int betamaxlen = length(stacklen(t_0_seq, h, m));
    if (betamaxlen < 2) continue;
    int n = size(t_0_seq);
    int k = get_index(subopt_left, i, m, h, n);
    int l = get_index(subopt_right, h, j, m, n);
    if (useSplitpoint) {
      k = get_index(subopt_left_heuristic, i, m, h, n);
      l = get_index(subopt_right_heuristic, h, j, m, n);
    }
    if (k > j || l > j) continue;
    if (l < k+2) continue;
    int alphamaxlen = length(stacklen(t_0_seq, i, k));
    if (alphamaxlen < minLengthKissingHairpinStems()) continue;
    int alphareallen = min(alphamaxlen, h-i-1);
    if (alphareallen < minLengthKissingHairpinStems()) continue;
    int gammamaxlen = length(stacklen(t_0_seq, l, j));
    if (gammamaxlen < minLengthKissingHairpinStems()) continue;
    int gammareallen = min(gammamaxlen, j-m-1);
    if (gammareallen < minLengthKissingHairpinStems()) continue;
    int betareallen = min(min(betamaxlen, k-h-alphareallen),
                          min(betamaxlen, m-l-gammareallen));
    if (betareallen < 2) continue;
    int stackenergies = energy(stacklen(t_0_seq,i,               k               ))
                      + energy(stacklen(t_0_seq,h,               m               ))
                      + energy(stacklen(t_0_seq,l,               j               ))
                      - energy(stacklen(t_0_seq,i+alphareallen-1,k-alphareallen+1))
                      - energy(stacklen(t_0_seq,h+betareallen -1,m-betareallen +1))
                      - energy(stacklen(t_0_seq,l+gammareallen-1,j-gammareallen+1));
    INNER(CODE);
   }
  }
 }
].


pkiss_C   =
.[
 int i = t_0_i;
 int j = t_0_j;
 if (j-i <= maxPseudoknotSize()) {
  for (int h = i+minLengthKissingHairpinStems()+1;
       h<=j-3*minLengthKissingHairpinStems()-2*2-3;
       h=h+1) {
   for (int m = h+2*minLengthKissingHairpinStems()+2*2+2;
        m<=j-minLengthKissingHairpinStems()-1;
        m=m+1) {
    rpk_setup(m-2-minLengthKissingHairpinStems());
    for (int k = m-minLengthKissingHairpinStems()-2-2;
         k>=h+minLengthKissingHairpinStems()+2;
         k=k-1) {
     int betamaxlen = length(stacklen(t_0_seq, h, m));
     if (betamaxlen < 2) continue;
     if (i+minLengthKissingHairpinStems()+1>h ||
         h+minLengthKissingHairpinStems()+2>k ||
         m+minLengthKissingHairpinStems()+1>j) continue;
     answer_pknot_mfe optPK = get_pk(h,j,k+2,m);
     if ((!isEmpty(optPK)) && (optPK.energy < rpk_energy(k+1))) {
       rpk_set(k, optPK.energy, optPK.betaLeftOuter);
     } else {
       rpk_set(k);
     }
     int l = rpk_index(k);
     int alphamaxlen = length(stacklen(t_0_seq, i, k));
     if (alphamaxlen < minLengthKissingHairpinStems()) continue;
     int alphareallen = min(alphamaxlen, h-i-1);
     if (alphareallen < minLengthKissingHairpinStems()) continue;
     if (k+2>l || l+2+minLengthKissingHairpinStems()>m) continue;
     int gammamaxlen = length(stacklen(t_0_seq, l, j));
     if (gammamaxlen < minLengthKissingHairpinStems()) continue;
     int gammareallen = min(gammamaxlen, j-m-1);
     if (gammareallen < minLengthKissingHairpinStems()) continue;
     int betareallen = min(
                         min(betamaxlen, k-h-alphareallen),
                         min(betamaxlen, m-l-gammareallen)
                       );
     if (betareallen < 2) continue;
     int stackenergies = energy(stacklen(t_0_seq,i,                k                ))
                       + energy(stacklen(t_0_seq,h,                m                ))
                       + energy(stacklen(t_0_seq,l,                j                ))
                       - energy(stacklen(t_0_seq,i+alphareallen -1,k-alphareallen+1))
                       - energy(stacklen(t_0_seq,h+betareallen  -1,m-betareallen  +1))
                       - energy(stacklen(t_0_seq,l+gammareallen-1,j-gammareallen+1));
     INNER(CODE);
    }
   }
  }
 }
].

pkiss_D =
.[
 int i = t_0_i;
 int j = t_0_j;
 if (j-i <= maxPseudoknotSize()) {
  for (int h = i+minLengthKissingHairpinStems()+1;
       h<=j-3*minLengthKissingHairpinStems()-2*2-3;
       h=h+1) {
   for (int k = h+2+minLengthKissingHairpinStems();
        k<=j-2*minLengthKissingHairpinStems()-2-3;
        k=k+1) {
    for (int l = k+2;
         l<=j-2*minLengthKissingHairpinStems()-2-1;
         l=l+1) {
     for (int m = l+2+minLengthKissingHairpinStems();
          m<=j-minLengthKissingHairpinStems()-1;
          m=m+1) {
      if (i+minLengthKissingHairpinStems()+1>h ||
          h+2+minLengthKissingHairpinStems()>k ||
          k+2>l ||
          l+2+minLengthKissingHairpinStems()>m ||
          m+minLengthKissingHairpinStems()+1>j) continue;
      int alphamaxlen = length(stacklen(t_0_seq, i, k));
      if (alphamaxlen < minLengthKissingHairpinStems()) continue;
      int alphareallen = min(alphamaxlen, h-i-1);
      if (alphareallen < minLengthKissingHairpinStems()) continue;
      int gammamaxlen = length(stacklen(t_0_seq, l, j));
      if (gammamaxlen < minLengthKissingHairpinStems()) continue;
      int gammareallen = min(gammamaxlen, j-m-1);
      if (gammareallen < minLengthKissingHairpinStems()) continue;
      int betamaxlen = length(stacklen(t_0_seq, h, m));
      int betareallen = min(
                          min(betamaxlen, k-h-alphareallen),
                          min(betamaxlen, m-l-gammareallen)
                        );
      if (betareallen < 2) continue;
      int stackenergies = energy(stacklen(t_0_seq,i,                k                ))
                        + energy(stacklen(t_0_seq,h,                m                ))
                        + energy(stacklen(t_0_seq,l,                j                ))
                        - energy(stacklen(t_0_seq,i+alphareallen -1,k-alphareallen+1))
                        - energy(stacklen(t_0_seq,h+betareallen  -1,m-betareallen  +1))
                        - energy(stacklen(t_0_seq,l+gammareallen-1,j-gammareallen+1));
      INNER(CODE);
     }
    }
   }
  }
 }
].
```