HAL
open science

Flexible RNA design under structure and sequence constraints using formal languages

Yu Zhou, Yann Ponty, Stéphane Vialette, Jérôme Waldispühl, Yi Zhang, Alain Denise

To cite this version:

Yu Zhou, Yann Ponty, Stéphane Vialette, Jérôme Waldispühl, Yi Zhang, et al.. Flexible RNA design under structure and sequence constraints using formal languages. ACM-BCB - ACM Conference on Bioinformatics, Computational Biology and Biomedical Informatics - 2013, Sep 2013, Bethesda, Washigton DC, United States. hal-00823279v2

HAL Id: hal-00823279

https://inria.hal.science/hal-00823279v2

Submitted on 1 Aug 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Flexible RNA design under structure and sequence constraints using formal languages

Yu Zhou[*] [†]
LRI, Univ. Paris-Sud
Orsay F-91405, France.

Yann Ponty[*]
LIX, Ecole Polytechnique
and AMIB, INRIA Saclay
Palaiseau F-91128,France

Stéphane Vialette
LIGM, Univ. Paris-Est
Marne-la-Vallée F-77454
France

Jérôme Waldispuhl
McGill University
Montreal, Quebec H3A 2B2
Canada

Yi Zhang
State Key Laboratory of
Virology, Wuhan University,
and ABLife Inc,
Wuhan, Hubei 430072, China

Alain Denise
LRI and IGM, Univ. Paris-Sud
Orsay F-91405
and AMIB, INRIA Saclay
France

ABSTRACT

The problem of RNA secondary structure design is the following: given a target secondary structure, one aims to create a sequence that folds into, or is compatible with, a given structure. In several practical applications in biology, additional constraints must be taken into account, such as the presence/absence of regulatory motifs, either at a specific location or anywhere in the sequence.

In this study, we investigate the design of RNA sequences from their targeted secondary structure, given these additional sequence constraints. To this purpose, we develop a general framework based on concepts of language theory, namely context-free grammars and finite state automata. We efficiently combine a comprehensive set of constraints into a unifying context-free grammar of moderate size. From there, we use generic algorithms to perform a (weighted) random generation, or an exhaustive enumeration, of candidate sequences.

The resulting method, whose complexity scales linearly with the length of the RNA, was implemented as a standalone program. The resulting software was embedded into a publicly available dedicated web server. The applicability of the method was demonstrated on a concrete case study dedicated to Exon Splicing Enhancers, in which our approach was successfully used in the design of *in vitro* experiments.

[*]Both first authors equally contributed

[†]Work was mainly performed while author was dually affiliated to: LRI, Univ. Paris-Sud, Orsay F-91405, France; and State Key Laboratory of Virology, Wuhan University, Wuhan, Hubei 430072, China. Current address: Department of Cellular and Molecular Medicine, Univ. of California, San Diego, La Jolla, CA 92093-0651, USA

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
BCB '13, September 22 - 25, 2013, Washington, DC, USA
Copyright 2013 ACM 978-1-4503-2434-2/13/09 ...$15.00.

Categories and Subject Descriptors

J.2 [**Computer Applications**]: Life and Medical Sciences—*Biology and genetics*; G.2.1 [**Combinatorics**]: Counting problems; F.4.3 [**Mathematical Logic and Formal Languages**]: Formal Languages—*Operations on languages*

Keywords

RNA design, Secondary Structure, Global Sampling, Finite-State Automata, Context-Free Grammars, Random Generation

1. INTRODUCTION

During the last years, the synthetic biology field and gene therapy techniques have considerably evolved [16, 10]. Ribonucleic acids (RNAs) emerged as versatile molecules capable to serve as logic gates [25] or to repress the replication of viruses [9]. To perform their functions, these RNAs use specific structures and nucleotide patterns enabling them to bind and interfere with other molecules. Efficient methods for designing such molecules is thus key to reach the next milestones in the emerging field of RNA nanotechnology [14].

In 1994, Hofacker and co-workers introduced the computational problem of RNA secondary structure design (a.k.a. inverse folding) [15] : Given a secondary structure, design an RNA sequence (if any), that folds into the sought structure. This definition reverses the classical RNA folding problem that consists in predicting the secondary structure from sequence data. Unfortunately, while there is a polynomial time and space solution for the folding problem without pseudoknots [28], the reciprocity remains questionable for the inverse folding problem [24]. Consequently, the Vienna group also proposed in their seminal 1994's paper a series of creative heuristics for solving the RNA inverse folding problem, which have been implemented in a program named `RNAinverse` [15].

In the late 2000s, the interest of the community for this problem grew with progresses and potential applications in nano-biotechnologies. Thereby, Andronescu *et al.* developed `RNA-SSD` [2], Busch and Backofen `INFO-RNA` [4, 5], Zadeh *et al.* `NUPACK` [27, 26], Dai and Wiese `rnaDesign` [6], Levin *et al.* `RNA-ensign` [18], and Garcia-Martin *et al.* `RNAiFold` [13].

All forth mentioned programs improved over the original `RNAinverse`.

Soon though, a need to design RNA sequences with more complex structural properties as well as a necessity to mimic properties of natural RNAs led to a second generation of software. First, the repertoire of designable shapes has been extended with `Inv` [12] and `fRNAkenstein` [20] that can design sequences with (respectively) pseudo-knots or multiple alternate secondary structures. Next, methods have been proposed to control generic sequence properties such as the genetic robustness with `RNAexinv` [3], or the `GC`-content with `IncaRNAtion` [22]

Nevertheless, as mentioned earlier, the (secondary) structure alone is not sufficient to fulfill all properties required to achieve most of RNA functions. In particular, our capacity to integrate multiple, precise and highly specific sequence constraints is key to design functional RNAs. For instance, RNAs often require specific nucleotide patterns to bind proteins (E.g. the hairpin of the TAR element). Among existing methods, `INFO-RNA` is the first to enable its users to input sequence constraints, fixing a set of positions in the sequence where only some nucleotides are allowed. `NUPACK` [27, 26] allows to prevent patterns in designed sequences, but cannot force the presence of a motif at an unspecific position.

Similar motif constraints (currently GC-content, positional constraints and a limit on runs of similar consecutive nucleotides) can also be defined with the recent `RNAiFold` [13], arguably the closest competitor to our approach. However, the constraint programming approach used by the authors is not guaranteed to run in polynomial time and, in practice, can become prohibitively slow for longer sequences, or rich sets of constraints. Furthermore, the set of constraints supported by the software is rather restrictive, and a need for more general sequence constraints often arise in real applications. This is the case in [19], where RNA structures have been designed and tested *in vitro* in order to study the effects of RNA structures on the activity of exonic splicing enhancers (ESEs). In that work the sequential constraints were given by one mandatory motif (the ESE) at a fixed position in the structure, and a (possibly large) set of forbidden motifs (other putative ESEs). Since such forbidden motifs could in principle appear anywhere in the structure, they cannot be captured by positional constraints and may be hard to avoid for large sets of forbidden sequences.

In this paper, we describe a general methodology for designing RNA secondary structures subject to multiple positive and negative sequence constraints with or without location restrictions. This methodology is based on concepts arising in formal language theory, namely context-free grammars and finite automata, and on algorithms for the exhaustive or random generation of words from a context-free language. In Section 2 we describe our approach in its full generality. Then, in Section 3, we show how the method has been applied for designing sequences that have been tested *in vitro* for a study on ESEs [19]. Section 4 describes our implementation, and a dedicated web server, offering a user-friendly environment for the generation and posterior analysis of sequences. Finally we conclude in Section 5 with some remarks and perspectives.

2. MODELING CONSTRAINTS USING LANGUAGE THEORETICAL CONSTRUCTS

It has been known for decades that concepts of language theory, namely regular expressions and finite state automata (FSA) on one hand, and context-free languages and grammars on the other hand, are useful to the fields of genomics and bioinformatics. Regular expressions and automata, and their probabilistic variants, as for example Markov models for sequences, have been widely used for describing motifs in genomic sequences. Furthermore context-free grammars, again with probabilistic variants like stochastic context-free grammars or covariance models, have been used for structural bioinformatics, as a basis for numerous works on prediction, comparison, and detection of RNA structures.

Here we combine these two formalisms (closely related within the Chomsky/Schutzenberger hierarchy of formal languages) to address the problem of designing RNA sequences that can fold to a given structure, while respecting a set of constraints given by mandatory motifs (having fixed or variable position in the sequence) and forbidden motifs.

2.1 Method overview

As stated above, our experimental applications require a rich combination of constraints. Let $\mathcal{B} := \{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{U}\}$ be the set of nucleotides, i.e. the vocabulary for our language constructs, these constraints can be broken down into the following four categories:

- **Secondary structure constraint (CT):** Designed sequences are expected to be compatible with a target secondary structure S, i.e. base-pairing positions in the structure must correspond to canonical base-pairs in the sequence (Watson-Crick G–C, A–U, or Wobble G–U).
- **Base positional constraint (CB):** Some positions within designed sequences must be chosen within position-specific restricted lists of bases. In other words, each position i is associated with a subset $\mathcal{C}_i \subset \mathcal{B}$. Note that such constraints can be adequately summarized by a sequence of IUPAC symbols, including the non-specific $\mathsf{N} \equiv \mathcal{B}$.
- **Mandatory motifs constraint (CM):** A predefined set $\mathcal{M}$ of sequences must be found **at least once**, at an unspecified position, in any generated sequence.
- **Forbidden motifs constraint (CF):** Any designed sequence must avoid a predefined set $\mathcal{F}$ of forbidden sequences.

Notice that the length $n := |S|$ of the designed sequence is implicitly set by the CB constraint. Now, our problem can be formally stated as:

> *Generate a set of k sequences of length n, which are compatible with the target S and $\mathcal{C}$, feature occurrences of each of the motifs in $\mathcal{M}$, and avoid the forbidden motifs in $\mathcal{F}$.*

Our method can be divided into four main steps:

1. Build a context-free grammar $\mathcal{G}$ that captures both the structure (CT) and base positional (CB) constraints, generating the language $\mathcal{L}_\mathcal{G}$ of all sequences that are compatible with the base-pairing constraints induced by S, and the IUPAC sequence $\mathcal{C}$.

2. Build a deterministic finite-state automaton (DFA for short) $\mathcal{A}$ which recognizes the language $\mathcal{L}_{\mathcal{A}}$ of all sequences that feature a set $\mathcal{M}$ of motifs (CM), while avoiding a set $\mathcal{F}$ of motifs (CF).

3. Construct the intersection context-free grammar $\mathcal{G}_{\cap}$ that generates the language $\mathcal{L}_{\mathcal{G}_{\cap}} := \mathcal{L}_{\mathcal{A}} \cap \mathcal{L}_{\mathcal{G}}$.

4. Use a (weighted) random generation algorithm to generate k sequences from $\mathcal{G}_{\cap}$. Optionally, filter sequences satisfying additional properties, such as their affinity towards the target structure, and other criteria, like the probability for certain regions to remain unpaired in the Boltzmann equilibrium.

Note that $\mathcal{L}_{\mathcal{G}_{\cap}}$ may be empty, as may result from a delicate interplay between the automaton and the grammar, if the constraints are too stringent. This property, which may not be obvious from the grammar, can nevertheless be efficiently tested.

2.2 Structural constraints and CFGs

This first step consists in building a formal grammar, whose language is exactly the set of sequences that are compatible with the input structure and the set of positional constraints (CB). Here, we assume that the secondary structure is free of pseudoknots, given as a dot-parenthesis notation and that corresponding parentheses are at least separated by one (unpaired) base ($\theta = 1$, using standard nomenclature).

One builds a context-free grammar $\mathcal{G} = (\mathcal{B}, \mathcal{V}, \mathcal{R}, S)$, such that:

- $S := V_1$ is the axiom.

- $\mathcal{V} = \{V_i\}_{i \in \mathcal{X}}$, where $\mathcal{X}$ is the set of positions corresponding to an unpaired base, or to the 5' end of some base pair (denoted by an opening parenthesis in the dot-bracket notation).

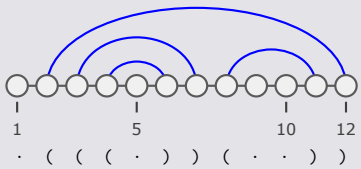
- The set of production rules is $\mathcal{R} = \bigcup_{i \in \mathcal{X}} \mathcal{R}_i$, where $\mathcal{R}_i$ is a set of productions associated with each non-terminal V_i, and depends on the status of the position i in the target structure S.

  For **unpaired positions** ($S_i =$ "."), any nucleotide may be generated, provided that the positional constraint is satisfied. If there is no follow-up sequence at this level of nesting, then the generation stops here. Otherwise, the sequence must be extended, a task which is delegated to another non-terminal V_{i+1}.

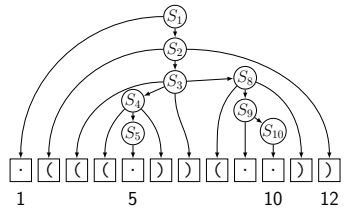
  $$\mathcal{R}_i := \begin{cases} \{V_i \to b\}_{b \in \mathcal{C}_i} & \text{[If } S_{i+1} = \text{"\)" or } i+1 > n,] \\ \{V_i \to b\, V_{i+1}\}_{b \in \mathcal{C}_i} & \text{[Otherwise.]} \end{cases}$$

  For **paired positions** ($S_i =$ "(", paired with j), the nucleotides chosen for the paired positions (i, j) must belong to a list $\mathcal{P}$ of valid base-pairs. Again, there may or may not be a follow up sequence at the same level of nesting, leading to the following cases:

  $$\mathcal{R}_i := \begin{cases} \{V_i \to b\, V_{i+1}\, b' \mid (b, b') \in (\mathcal{C}_i \times \mathcal{C}_j) \cap \mathcal{P}\} \\ \text{[If } S_{j+1} = \text{"\)" or } j+1 > n,] \\ \{V_i \to b\, V_{i+1}\, b'\, V_{j+1} \mid (b, b') \in (\mathcal{C}_i \times \mathcal{C}_j) \cap \mathcal{P}\} \\ \text{[Otherwise.]} \end{cases}$$



A – Target Secondary Structure



B – Generating tree

$S_1 \to .\, S_2 \quad S_2 \to (\, S_3\,) \quad S_3 \to (\, S_4\,)\, S_8 \quad S_4 \to (\, S_5\,)$
$S_5 \to . \quad S_8 \to (\, S_9\,) \quad S_9 \to .\, S_{10} \quad S_{10} \to .$

C – Intermediate Grammar

$V_1 \to \mathsf{A}\, V_2 \mid \mathsf{C}\, V_2 \mid \mathsf{G}\, V_2 \mid \mathsf{U}\, V_2$
$V_2 \to \mathsf{A}\, V_3\, \mathsf{U} \mid \mathsf{C}\, V_3\, \mathsf{G} \mid \mathsf{G}\, V_3\, \mathsf{C} \mid \mathsf{G}\, V_3\, \mathsf{U} \mid \mathsf{U}\, V_3\, \mathsf{A} \mid \mathsf{U}\, V_3\, \mathsf{G}$
$V_3 \to \mathsf{A}\, V_4\, \mathsf{U}\, V_8 \mid \mathsf{C}\, V_4\, \mathsf{G}\, V_8 \mid \mathsf{G}\, V_4\, \mathsf{C}\, V_8 \mid \mathsf{G}\, V_4\, \mathsf{U}\, V_8 \mid \mathsf{U}\, V_4\, \mathsf{A}\, V_8 \mid \mathsf{U}\, V_4\, \mathsf{G}\, V_8$
$V_4 \to \mathsf{A}\, V_5\, \mathsf{U} \mid \mathsf{C}\, V_5\, \mathsf{G} \mid \mathsf{G}\, V_5\, \mathsf{C} \mid \mathsf{G}\, V_5\, \mathsf{U} \mid \mathsf{U}\, V_5\, \mathsf{A} \mid \mathsf{U}\, V_5\, \mathsf{G}$
$V_5 \to \mathsf{A} \mid \mathsf{C} \mid \mathsf{G} \mid \mathsf{U}$
$V_8 \to \mathsf{A}\, V_9\, \mathsf{U} \mid \mathsf{C}\, V_9\, \mathsf{G} \mid \mathsf{G}\, V_9\, \mathsf{C} \mid \mathsf{G}\, V_9\, \mathsf{U} \mid \mathsf{U}\, V_9\, \mathsf{A} \mid \mathsf{U}\, V_9\, \mathsf{G}$
$V_9 \to \mathsf{A}\, V_{10} \mid \mathsf{C}\, V_{10} \mid \mathsf{G}\, V_{10} \mid \mathsf{U}\, V_{10}$
$V_{10} \to \mathsf{A} \mid \mathsf{C} \mid \mathsf{G} \mid \mathsf{U}$

D – Final Grammar

Figure 1: Illustration of the grammar construct in the absence of positional constraints: Any target secondary structure S (A) can be abstracted as a tree (B) through a linear-time parsing, from which one derives an intermediate grammar (C) whose sole production is S. The rules of this intermediate grammar can then be duplicated to allow for alternative nucleotides, resulting into a final grammar $\mathcal{G}$ (D), which generates every RNA sequence compatible with S. Additionally, the rules of $\mathcal{G}$ can be restricted to account for further positional constraints.

In the above productions, we may typically allow only canonical base-pairs, thus:

$$\mathcal{P} := \{(\mathsf{A}, \mathsf{U}), (\mathsf{C}, \mathsf{G}), (\mathsf{G}, \mathsf{C}), (\mathsf{G}, \mathsf{U}), (\mathsf{U}, \mathsf{A}), (\mathsf{U}, \mathsf{G})\}.$$

Note that these productions can be adapted in a straightforward manner from those of a specialized grammar that only generates the targeted secondary structure. The whole process of going from the secondary structure to the corresponding grammar is illustrated by Figure 1. The resulting grammar has $\Theta(n)$ rules, and can be generated in $\Theta(n)$ time and space from the target secondary structure, given in dot-parenthesis notation, and a sequence of positional constraints, given as a set of IUPAC symbols.

2.3 Mandatory/forbidden motifs and automata

We consider the language $\mathcal{L}_{\mathcal{A}}$ of words that obey both the CM and CF constraints at the primary sequence level,

and describe how to construct a deterministic finite-state automaton (DFA for short) $\mathcal{A}$ that recognizes $L_{\mathcal{A}}$. We assume some level of familiarity with the basic notions and properties of finite-state automata, and direct the interested reader to Salomaa's classic reference [23].

The existence of $\mathcal{L}_{\mathcal{A}}$ follows from closure properties of regular languages through classical operations on sets (union, intersection, complement), and by concatenation. This means that the automaton that satisfies our various constraints can be incrementally built from automata associated with each of the individual constraints. However, the resulting automaton could become too large thereby negatively impacting on the complexity of the whole method. Therefore, we propose ad-hoc constructs based on the Aho-Corasick automaton, leading to much smaller automata.

Formal language constructs. Let $\mathcal{L}_{CM}$ and $\mathcal{L}_{CF}$ be the languages of words that respect the constraints CM and CF, respectively. Cleary, one has $\mathcal{L}_{\mathcal{A}} = \mathcal{L}_{CM} \cap \mathcal{L}_{CF}$, so it suffices to derive automata for $\mathcal{L}_{CM}$ and $\mathcal{L}_{CF}$.

Let us first consider the language $\mathcal{L}_{CF}$ that contains all the words of N^* which avoid all motifs in $\mathcal{F} = \{f_1, f_2, \ldots, f_{k'}\}$. This language can be described as the complement in N^* of $\mathcal{L}_{\overline{CF}}$ the language that asks for at least one occurrence of a motif in $\mathcal{F}$. The complement language can be further described as the union $\mathcal{L}_{\overline{CF}} = \bigcup_{i=1}^{k'} \mathcal{L}_{f_i}$, where $\mathcal{L}_{f_i}$ is the set of words having at least one occurrence of $f_i \in \mathcal{F}$. Clearly, $\mathcal{L}_{f_i}$ is generated by the regular expression $N^*.f_i.N^*$, and hence can be recognized by an automaton (having $|f_i| + 1$ states). Furthermore, regular languages are stable by union, so there exists an DFA that recognizes $\mathcal{L}_{\overline{CF}}$ and, taking the complement, $\mathcal{L}_{CF}$ is also recognizable by an automaton.

The general scheme for constructing a DFA that recognizes $\mathcal{L}_{CM}$ is similar. Let $\mathcal{M} = \{m_1, m_2, \ldots, m_{k''}\}$ be the set of mandatory motifs. Clearly, one has $\mathcal{L}_{CM} = \bigcap_{i=1}^{k''} \mathcal{L}_{m_i}$, where $\mathcal{L}_{m_i}$ is the language of all words containing at least one occurrence of m_i, and can be constructed as described above. Since regular languages are closed by intersection, then $\mathcal{L}_{CM}$ is also regular and can be recognized by a DFA.

Aho-Corasick inspired automata. The closure properties used by the above constructions can be proven constructively, thus they can be used to build $\mathcal{A}$. However, a naive implementation of these operations would yield a number of states in $\Theta((\sum_i |f_i|) \times \prod_j (1 + |m_j|))$. Such a potentially huge state space would lead to time-consuming computations while occupying a large amount of memory. However, it can be shown that a much smaller automaton exists for the same language.

Indeed, let us remind that automata for the language of words ending with some fixed motif can be adapted from the Aho-Corasick (AC) automaton [1]. As illustrated by Figure 2B and 2E, an AC automaton is a simple prefix tree, complemented with failure transitions (dashed edges), which allows for an efficient, single-pass, string matching for a complete set of motifs $\mathcal{W}$. Namely, the AC automaton uses its states to represent the maximal prefix of a word in $\mathcal{W}$ which has been read/generated at any given moment. Reading a new symbol either leads to a extended prefix, meaning a progression within the prefix tree, or a failure, resulting in a transition to a state associated with another smaller (possibly empty) prefix. Note that, despite its classic representation as a non-deterministic automaton, the AC automaton can easily be made deterministic in linear time, as

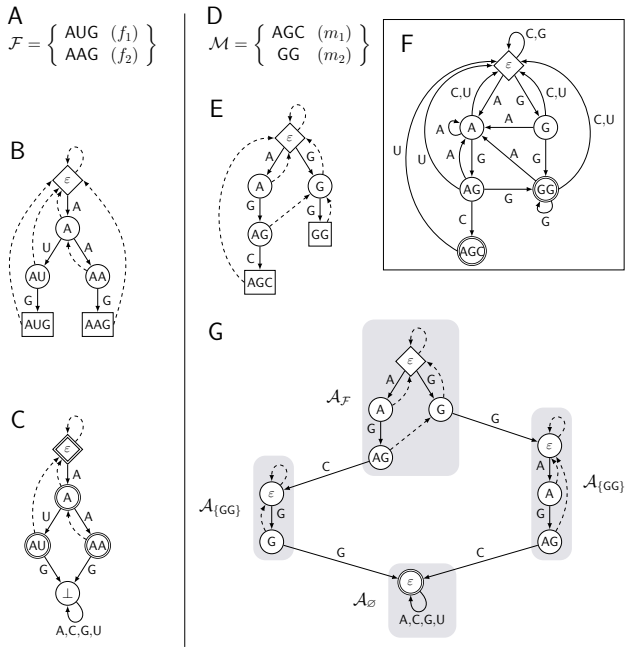


Figure 2: Using the Aho-Corasick automaton to build compact automata for $\mathcal{L}_{CF}$ and $\mathcal{L}_{CM}$. From a set $\mathcal{F}$ (A), an AC automaton can be built (B) and transformed into an automaton for the language that excludes $\mathcal{F}$ (C). For a set of mandatory motifs (D), a similar initial construct can be used (E), later to be duplicated to retain a memory of the motifs that are still to be generated (G). The resulting automata can easily be made deterministic and completed (F).

illustrated by Figure 2E and 2F, increasing the number of edges (in $\Theta(\sum_i |w_i|)$) at most by a constant factor $|\mathcal{B}|$.

From such a construction, applied to the set $\mathcal{F}$ of forbidden motifs (cf Figure 2B), an automaton for $\mathcal{L}_{CF}$ can be obtained by simply disallowing (rerouting to a non-accepting ground state $\perp$) each transition leading to a complete motif in $\mathcal{F}$. Any other state in the AC automaton is set as final/accepting, and the initial state is the one associated with the suffix ε, as illustrated by Figure 2C.

A similar approach can be used for $\mathcal{L}_{CM}$. This time, one may simply create $2^{|\mathcal{M}|}$ versions of the AC automaton, each built for a subset of motifs $\mathcal{M}' \subseteq \mathcal{M}$ that remains to be generated. Reflecting these semantics, any transition leading to a complete motif m_i in an automaton $\mathcal{M}'$ with be rerouted to a state, in the automaton $\mathcal{M}'' := \mathcal{M}'/\{m_i\}$, which corresponds to the longest suffix of m_i which is a prefix of a word in $\mathcal{M}''$. The initial state will be the one associated with the prefix ε in the automaton $\mathcal{M}$, and any state in the automaton $\varnothing$ is accepting/final, as illustrated by Figure 2G.

These two constructs may even be merged into a single one (omitted from the example), that directly builds an automaton for $\mathcal{L}_{CM} \cap \mathcal{L}_{CF}$. One simply builds AC automata for each set of words $\mathcal{F} \cup \mathcal{M}'$, $\forall \mathcal{M}' \subseteq \mathcal{M}$. Any transition towards a state associated with a forbidden word $f \in \mathcal{F}$ is rerouted to the non-accepting ground state $\perp$ and any transition leading to a state $m \in \mathcal{M}$ in the automaton $\mathcal{M}'$ is rerouted to the automaton $\mathcal{M}'/\{m\}$ as described above. The initial state is then $(\varepsilon, \mathcal{M})$, and any state $(-, \varnothing)$ is fi-

nal. As summarized in Table 1, the resulting automaton has $\Theta(2^{|\mathcal{M}|} \cdot (\sum_i |f_i| + \sum_j |m_j|))$ states, and can be built in linear time. Note that the final automaton is not necessarily minimal, and one may obtain an even more compact automaton using any minimization algorithm.

2.4 Combining structural and sequential constraints

It is well-known that context-free languages are not closed under intersection. However, the intersection of a context-free language with a regular language is always context-free. The standard proof of the latter statement involves running a deterministic finite state automaton in parallel with a pushdown automaton, and noting that the whole process can itself be simulated by a pushdown automaton that accepts by final states. Of particular interest in our context, there exists an efficient algorithm that computes the context-free grammar of the intersection of a context-free language with a regular language, as soon as the former is given as a context-free grammar, and the latter is given as a deterministic finite state automaton [23] (the algorithm does not need to construct nor explicitly simulate a pushdown automaton). In its original form, the algorithm needs the input grammar to be transformed into an equivalent grammar in Chomsky Normal Form. However, since our grammars have peculiar kinds of rules, we can design a variant of the original algorithm that avoids this step. Furthermore, this algorithm possibly generates useless rules and symbols, which do not generate any word due to conflicting constraints in both languages. A *cleaning* step allows to remove these rules and symbols from the grammar.

The general principles of the algorithm are the following. We start from a grammar $G =< N, V, P, S >$, where $N = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{U}\}$ is the set of terminal symbols, V is the set of non-terminal symbols, P is the set of rules, and $S \in V$ is the axiom; and from a DFA $\mathcal{A} =< N, Q, q_0, F, \delta >$, where Q is the set of states, $q_0 \in Q$ is the initial state, F is the set of final states, and δ is the transition function. The output grammar G' has its set of non-terminal symbols $V' \subset Q \times V \times Q \cup S'$, where S' is the axiom of G'. Any symbol $(q, T, q') \in V'$ generates the words which are generated by T in the grammar G, and which correspond to a path from q to q' in the automaton $\mathcal{A}$. Below are examples of this construction:

$$
\begin{array}{lcl}
V_i \to b & \Rightarrow & \{V_i^{q\to r} \to b \mid \delta(q,b) = r\} \\
V_i \to b\, V_{i+1} & \Rightarrow & \{V_i^{q\to r} \to b\, V_{i+1}^{s\to r} \mid \delta(q,b) = s\} \\
V_i \to b\, V_{i+1}\, b' & \Rightarrow & \{V_i^{q\to r} \to b\, V_{i+1}^{s\to t}\, b' \\
 & & \qquad \mid \delta(q,b) = s \text{ and } \delta(t,b') = r\} \\
V_i \to b\, V_{i+1}\, b'\, V_{j+1} & \Rightarrow & \{V_i^{q\to r} \to b\, V_{i+1}^{s\to t}\, b'\, V_{j+1}^{u\to r} \\
 & & \qquad \mid \delta(q,b) = s \text{ and } \delta(t,b') = u\}
\end{array}
$$

The algorithm begins by creating the axiom S' and one rule $S' \to [q_0, S, q]$ for every $q \in F$. A stack allows to store every newly created non-terminal symbol. When a symbol is popped out, all the rules starting with it are generated and then pushed up.

Algorithm 1 presents a basic implementation of these principles (the easy cleaning step is omitted for the sake of simplification). Additional technical improvements may be added (and are implemented in our webserver) for the algorithm to become more space and time efficient, but they are beyond the scope of this paper.

The algorithm produces a grammar having a set of rules in $\Theta(|\mathcal{R}| \cdot |Q|^3)$, each produced in constant average time. In the specific context of RNA rational design, let us remind that $|\mathcal{R}| \in \Theta(n)$ and $|Q| \in \mathcal{O}(2^{|\mathcal{M}|} \cdot (f + m))$ after minimization, where $f := \sum_i |f_i|$ and $m := \sum_j |m_j|$. It follows that the overall time and space complexities of the complete algorithm grow like $\mathcal{O}(n \cdot 8^{|\mathcal{M}|} \cdot (f + m)^3)$, i.e. linearly on the sequence length.

2.5 Sequence generation

From the final context-free grammar, a random generation of sequences can be done using `GenRGenS` [21] or `GrgFreqs` (a C++ re-implementation of `GenRGenS`). Both implementations feature procedures for counting the total number of words compatible with the constraints, their uniform random generation and their exhaustive enumerations. These procedures rely on a recursive precomputation analogous to a dynamic-programming scheme [8] for counting the number of words generated from each non-terminal. From a weighted count, one can easily compute probabilities associated with the choice of a given production rule among the ones accessible from a given non-terminal, in such a way that applying the process recursively will result in a uniform random generation of admissible sequences. A naive implementation of these procedures requires $\Theta(n^2.|\mathcal{R}|)$ arithmetic operations for general grammars.

However, it can be remarked that the grammar $\mathcal{G}_\cap$ is rather peculiar, as each of its rules only produces words of a single length. Consequently, the recursions used to compute the number of generated sequences greatly simplify (as the convolution products responsible for the superlinear behavior are no longer justified). They can consequently be computed in $\Theta(|\mathcal{R}|)$ time and $\Theta(|V|)$ space, each generation requires $\Theta(|Q| \cdot n)$ arithmetic operations. `GrgFreqs` has been extended to detect automatically such grammars, and adapt its algorithm accordingly.

2.6 Filtering generated sequences

Sequences generated from the grammar are compatible with the target structure, and also satisfy all the constraints on primary sequence. However, they may not necessarily fold into the MFE structure, i.e. admit the target secondary structure as their MFE. To work around such a potential issue, we use the following approach:

Lang.	#States	Remark
$\mathcal{L}_{CF}$	$\leq 2 + \sum_{i=1}^{\lvert\mathcal{F}\rvert} \lvert f_i \rvert$	Aho-Corasick
$\mathcal{L}_{CM}$	$\leq 2^{\lvert\mathcal{M}\rvert} \sum_{m_i \in \mathcal{M}} \lvert m_i \rvert$	Aho-Corasick
$\mathcal{L}_{\mathcal{A}}$	$\mathcal{O}\left(2^{\lvert\mathcal{M}\rvert} \sum_{i=1}^{\lvert\mathcal{M}\rvert} \lvert m_i \rvert \sum_{i=1}^{\lvert\mathcal{F}\rvert} \lvert f_i \rvert\right)$	Intersection
$\mathcal{L}_{\mathcal{A}}$	$\Theta\left(2^{\lvert\mathcal{M}\rvert} \cdot (\sum_i \lvert f_i \rvert + \sum_j \lvert m_j \rvert)\right)$	Aho-Corasick

Table 1: Number of states for the sequence constraint automaton.

Algorithm 1: Intersection of CFG with DFA

Input : CFG $G =< N, V, P, S >$ and DFA $\mathcal{A} =< N, Q, q_0, F, \delta >$
Output: $G' =< N, V', P', S' >$

```
begin
    Create new axiom S'
    P' ← ∅
    NTqueue ← ∅
    foreach q in F do
        Add S^{q0→q} to NTQueue
        Add (S' → S^{q0→q}) to P'
    while NTqueue ≠ ∅ do
        V^{q→q'} ← pop NTqueue
        for p ∈ P where p.leftHandSide = V do
            switch p do
                case V → u0        // Terminal (a|u|c|g)
                    q' ← δ{q, u0}
                    Add (V^{q→q'} → u0) to P'
                case V → u0V1          // (aV|uV|..)
                    q1 ← δ{q, u0}
                    Add (V^{q→q'} → u0 V1^{q1→q'}) to P'
                case V → u0V1u1        // (aVu|cVg|..)
                    q1 ← δ{q, u0}
                    for q'1 ∈ Q where δ{q'1, u1} = q' do
                        Add (V^{q→q'} → u0 V1^{q1→q'1} u1) to
                        P'
                case V → u0V1u1V2  // (aVuW|cVgW|..)

                    q1 ← δ{q, u0}
                    for q'1 ∈ Q do
                        q2 ← δ{q'1, u1}
                        Add
                        (V^{q→q'} → u0 V1^{q1→q'1} u1 V2^{q2→q'}) to
                        P'
                case V → V1        // Only when V=S in
                current grammar
                    if q = q0 and q' ∈ F then
                        Add (V^{q→q'} → V1^{q→q'}) to P'
                otherwise        // No other cases in
                current grammar
                    Exception
            if NonTerminal in above productions is new
            then  Add it to NTqueue
            ;
```

- First, we use a weighting scheme to direct the random generation towards sequences of high affinity. Indeed, it was shown by some of the authors that using a weighted sampling strategy based on the free-energy greatly increases the probability of randomly generated sequences to fold into the target structure [18, 22]. Since `GrgFreqs/GenRGenS` support such a non-uniform model [21, 8], we included in our software an option to draw each sequence with respect to a pseudo Boltzmann-distribution. In this distribution, any admissible sequence is generated with probability proportional to $e^{-E/RT}$, where R is the universal gas constant, T is the temperature in Kelvin and E is the free-energy of the target secondary-structure S. For the latter, we used a simplified version of the Turner 2004 energy model restricted to stacking base-pairs, requiring minor, yet technical, modifications of the grammar $\mathcal{G}_\cap$ (omitted in this presentation).
- Then, a naive *filtering* step selects sequences which fold to the given structure and have high self-containment index [17], probability to maintain the structure under its sequence context. Other filters, like unpaired probability value for a specific motif, can be applied to find better sequences according to the requirement.

For longer sequences, this strategy may still reveal insufficient to produce sequences that fold exactly into the target structure. In such situations, generated sequences may still be used as initial sequences (aka seeds) for traditional local search methods. Indeed, the combination of random generation (global sampling) and local search into a *glocal* hybrid strategy was shown to outperform individual approaches, both in term of accuracy and diversity of the produced sequences [22].

3. EXPERIMENTS

We report here a case study that was done by some of the authors, using an early version of our software. This experimental study aimed to characterize how the structural context of an exon slicing enhancer (ESE) in a transcript affects its functionality [19]. It is known that strong RNA secondary structures decrease the accessibility of the embedded exonic splicing enhancer (ESE) and thus hinder its recognition by single-stranded RNA binding proteins. This can result in enhanced or repressed exon splicing and different mRNA isoforms, which may have great impact on the functionalities of the proteins. In order to further characterize the effect of different RNA secondary structures on ESE functions, we needed to place the ESE motif in different structural context to test its functions experimentally.

An *in vitro* splicing system was established by cloning the SMN1 splicing cassette into the GFP gene. The cassette sequence contained the SMN1 alternative exon 7 and its partial flanking introns. Various constructs (one ESE in different context) could be inserted into the exon (see Figure 3A), and RT-PCRs were used to quantify the splicing outcomes reflecting the contextual effect on the ESE.

Among the many known variants of the ESE motif [11], representative ESEs such as Tra2β- and ASF-ESE) with high enhancer activities were selected to be tested. These variants were used as mandatory motif (**CM**) in the constructs. We chose to consider a simple structure, a stem-loop, and to modify the structural context by putting it at different positions relatively to the ESE. More precisely, we designed four groups of constructs, including: 1) ESE in stem regions with varying stabilities by using different number of base-pairs (Figure 3B); 2) ESE flanking by a varying stability stem (Figure 3C); 3) ESE upstream or downstream a strong stem-loop with different lengths of spacers (Figure 3D); 4) ESE in the varying-size loop of a strong stem-loop (Figure 3E). The reverse complementary sequence of the ESE in Figure 3B was used as an negative control ESE, leading to an experiment using another mandatory motif. Meanwhile, to avoid the confounding effects by other ESE motifs in the sequence, the constructs had to not contain

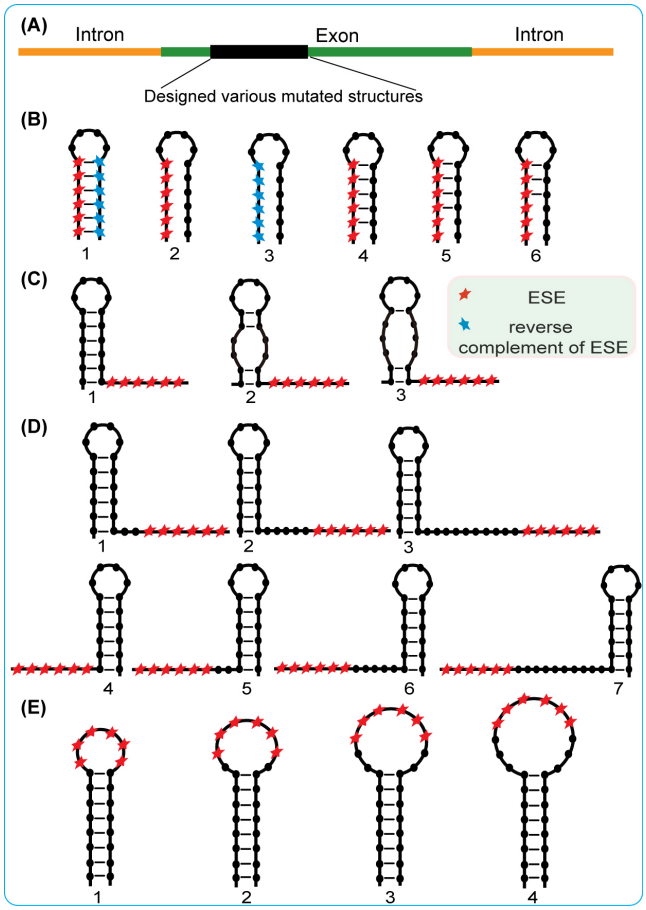


Figure 3: Illustration of the experimental design (A) and various constructs containing the ESE motif in different structural context (B-D). ESE location are marked by the red star symbols.

any other potential ESE sequence (**CF**), nor even any ESS sequence (Exonic Splicing Silencer). This set of forbidden motifs could be up to 1475 out of 4096 hexamers, if all the computationally identified motifs were used [11].

Starting from these constraints and using our methodology, we generated a set of candidates, further restricted by our experimental collaborators, later to be constructed and experimentally tested. The results further supported that stem structure could block embedded ESE function, and showed that ESEs following a stable hairpin structure could be highly active. It was also found that different ESEs in the loop may function differently because of potential non-canonical base-pairings. This suggested that the modulation on ESE functions by RNA structure could be prevalent and in different modes of actions. More details regarding the experimental aspects of the study can be found in Li *et al* [19].

4. SOFTWARE AND WEB SERVER

The package named CFGRNAD, was mainly written in Python, and integrates the `OCaML FSA` program (`http://www.linguistics.ucla.edu/people/grads/jheinz/software/`), and `GrqFreqs` [21]. The package including the programs can be downloaded from `https://code.google.com/p/cfgrnad/`.

4.1 Web server

`CFGRNAD` was embedded in a web server available at: `http://www.lix.polytechnique.fr/RNADesignStudio/` The server enables a user to submit a design task, specifying various constraints and options, such as:

- Targeted secondary structure in dot-bracket notation;
- Position-specific constraints denoted by a sequence of IUPAC codes;
- Lists of forbidden and mandatory motifs, either manually specified or uploaded as a file;
- Number of generated sequences;
- Random generation model, chosen between a uniform model and a Boltzmann/weighted model. The latter weighs each eligible sequence with a probability $p \propto e^{-E/RT}$, where E is the sequence free-energy assuming the targeted structure, R is the universal gas constant and T is the temperature in Kelvin. A simplified version of the 2004 Turner model for the free-energy is used, assigning individual contributions to each stacking pairs and disregarding other terms.

After computation, the resulting set of sequences is displayed and can be further analyzed within a dedicated web page, accessible at a later time using a unique accession id. In this web space, the structure/sequence can be visualized using the VARNA software [7] and each candidate sequence can be evaluated using various statistics, including:

- G+C-content;
- Free-energy of sequence upon forming the targeted structure, evaluated within the Turner model;
- Boltzmann probability of the targeted structure;
- Difference between the minimal free-energy structure and its second best suboptimal;
- Whether or not the targeted structure is the MFE for the candidate sequence.

These statistics are computed on-demand from the result page using software from the Vienna RNA software package [15] (with the exception of the straightforward GC-content).

4.2 Runtime and robustness testing

To confirm its linear complexity, we measured the runtime of our random generation software. To that purpose, we used a realistic random model used by Levin *et al* [18], which performs a weighted random generation to draw structures which, on the average, have similar structural properties (#base-pairs/length, #helices/length...) as experimentally-determined ribosomal RNAs. For length varying from 50 to 300nts by steps of 25nts, we generated 20 random secondary structures. For each of the generated structure, we designed 100 candidates, using a single mandatory motif $\mathcal{M} = \{\mathsf{CGU}\}$ and three arbitrary forbidden motifs $\mathcal{F} = \{\mathsf{AAU}, \mathsf{CGC}, \mathsf{UGC}\}$. The results, summarized in Figure 4 by averaging the runtime of sequences of equal length, confirm the linear complexity of the software, while exhibiting a fairly low variance.

We also performed more involved tests of our implementation on experimentally-determined structures, leading to

Test	Sequence/Structure	M	#F	#state	#trans	GI	GA	GB	#seqAG
1	CUCGAACGCAANNNNNNNNNAAUUC .....((((((....))))))....	ACGCAA	1	902	3062	14142,17522	2010,6405	630,803	346
2	CUCGAACGCAANNNNNNNNNAAUUC	ACGCAA	1	902	3062	906, 3064	903, 3064	903, 3064	457647
3	CUCGAACGCAANNNNNNNNNAAUUC .....(((((....))))).....	ACGCAA	1	902	3062	24220, 29483	3123, 10438	636, 820	1038
4	CUCGAACGCAANNNNNNNNNAAUUC .....((((....))))....	ACGCAA	1	902	3062	38507, 48783	5434, 18910	645, 856	4844
5	CUCGAACGCAANNNNNNNNNAAUUC .....(((....))).....	ACGCAA	1	902	3062	48087, 56849	5826, 20167	663, 914	7958
6	CUCGANNNNNNUACAGANNNNNNAAUUC .....((((((.....)))))).....	UACAGA	1	222	712	16643, 47520	14946,45655	1998, 5005	13948
7	CUCGANNNNNNNNNNNNNNNNUACAGAAAUUC .....((((((....)))))).....	UACAGA	0	0	0	-	-	-	0
8	NNNNNNNNNNUCGUCG (((....))).....	UCGUCG	1	1117	3889	716553, 1710596	219601, 885384	18223,34324	42234
9	UCGUCGNNNNNNNNNN .....(((....)))	UCGUCG	1	939	3209	115152, 252205	32620, 122590	11469,20532	35209

Table 2: Statistics of the RNA design on test data. All the 9 tests use the same set of forbidden motifs which contains 238 putative hexamer ESEs. Sequence: expected sequence, N means [AUCG]; Structure: bracket notation, '()' means pairing; M: Mandatory motif, present only once. Forbidden: forbidden motif set, in which the motifs cannot be present; #F, #state and #trans means the number of final states, total states, and transitions, independently; GI, GA, GB represent the (numbers of non-terminal states, number of productions) pair statistics for the grammar after Initial intersection, after removing productions containing non-accessible non-terminals, and after clean the productions, independently; #seqAG: number of sequences under both constraints of motifs and structure.

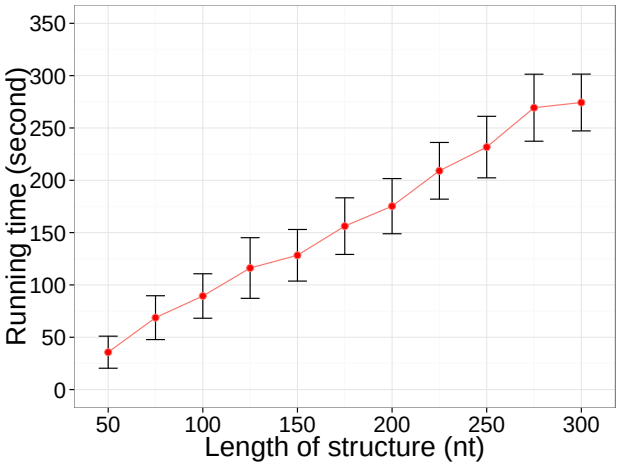


Figure 4: Typical runtime on random realistic secondary structures and sets of motif $\mathcal{M} = \{\mathsf{CGU}\}$ and forbidden motifs $\mathcal{F} = \{\mathsf{AAU}, \mathsf{CGC}, \mathsf{UGC}\}$.

statistics are summarized in Table 2. For all these tests, we used a much larger set of forbidden motifs proposed by Fairbrother *et al* [11], consisting of 238 putative ESEs. We noticed that the size of FSA became much larger than that on a small dataset, which contained only 4 or 5 2-mers, and the time required to build the FSA also increased. The cost of the intersection between the grammar and the FSA increases with the size of the automaton. Test 8 has the highest run time for intersection, the reason for which is probably that both side of the stem do not have motif base constraints. It is interesting that the polarity of the structure influences the time of intersection, as can be noticed by comparing test 8 to test 9. One of the reasons may be that the stem on the 5' end in test 8 generates too much unused non-terminals, or productions, than the stem at the 3' end in test 9. Finally, test 7 reveals that the list of putative ESEs conflicts with the mandatory constraints, revealing an unfeasible design objective. Being able to identify such situation is one of the strength of our method, especially in comparison with local search alternatives.

4.3 Potential for inverse folding

As emphasized in our introduction, our approach focuses on positive design principles (target + constraints compatibility), but does not explicitly capture negative design goals (specificity, avoidance of more favorable folds). However, its low complexity allows for a generation of a large number of independent candidates, which can be refolded and tested. A first benchmark was proposed in Reinharz *et al* [22], with a strong emphasis on the `RNASSD` software [2]. Here, we supplement this study by comparing the global sampling approach implemented by `CFGRNAD` with `NUPACK` [26].

Our benchmark consists in sets of 20 random target secondary structures for each length in $[30, 50, 75, 100]$, generated as described in Reinharz *et al* [22]. For each target structure, we designed a set of 20 design candidates using `NUPACK` (random initial seed + Turner 1999 energy model). For difficult designs, `NUPACK` may produce sequences whose MFE-predicted structure does not exactly match the target structure, yet share many base-pairs with it. In order to assess the quality of the designed sequence, we define the structural sensitivity of a sequence w with respect to a target S as:

$$\mathsf{Sen}(w, S) = \frac{|\{(i,j) \mid (i,j) \in S \text{ and } (i,j) \in S_w\}|}{|w|} \qquad (1)$$

where S_w is the MFE structure for w, predicted using `RNAfold` (Turner 1999 energy model). Then we ran our software `CFGRNAD` for a total runtime which is equal to that of `NUPACK`. We include in both measured runtimes the computation of structural sensitivities. This property seems essential for a fair comparison, as `CFGRNAD` produces much more candidate sequences than its competitor. Furthermore, we emphasize

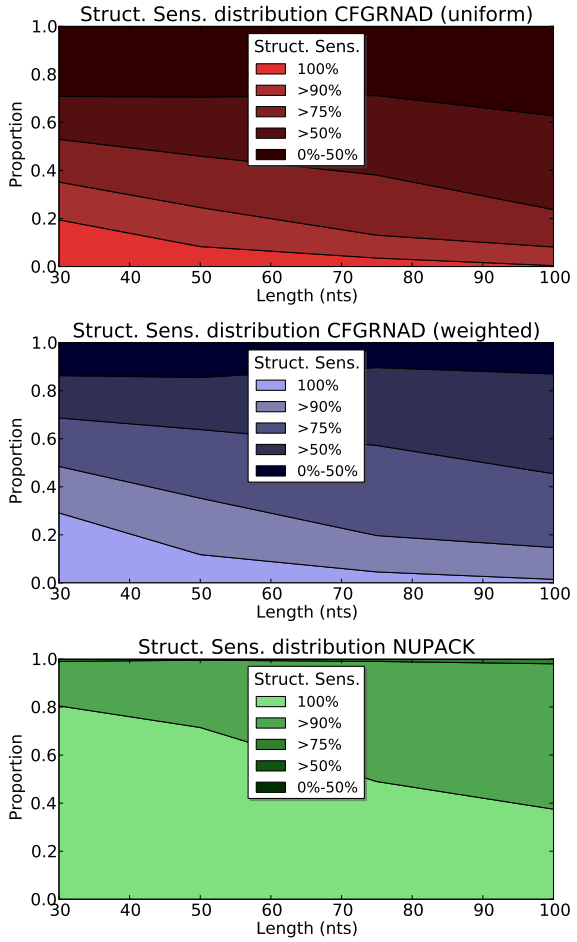


Figure 5: Structural sensitivity distribution for sequences designed using CFGRNAD (uniform distribution, top), CFGRNAD (weighted distribution, middle) and NUPACK (bottom).

that the energy model (Turner 1999) used to score a design is the same as the one used by NUPACK in its optimization, but differs from that of CFGRNAD (modified Turner 2004).

An analysis of the distribution of structural sensitivities, shown in Figure 5, confirms the positive impact of weights on the quality of CFGRNAD designs. The relative proportions of successful designs remains highly favorable to NUPACK. For the latter, the average probabilities of complete success ranged from 80% down to 40% for sequence lengths between 30 and 100 nts, and 97% of refolded designs shared at least 90% of base-pairs with their target. By contrast, CFGRNAD only achieved 30%-2% for complete success and 50-18% for >90% sensitivity.

However, the linear time complexity of CFGRNAD allows for the generation of larger sets of candidates, even when the cubic-time RNAfold is included in the postprocessing. For instance, using NUPACK to generate 20 candidates for a structure of 100 nts may require as much as 6 800 seconds on a standard laptop, allowing for the generation and evaluation of 197 000 CFGRNAD candidates. As shown in Figure 6, this larger number of candidates produced by CFGRNAD in a given time apparently compensates its lower probability

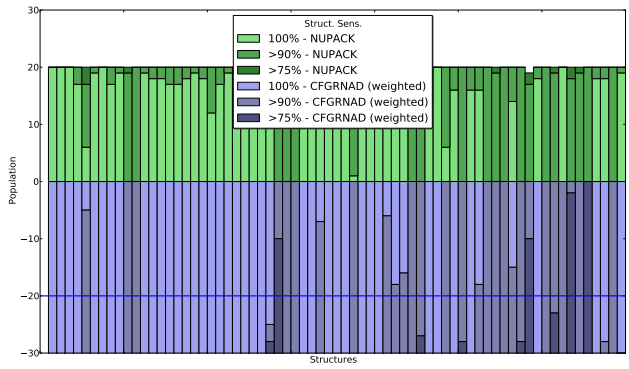


Figure 6: Raw population and structural sensitivity of candidate sequences generated by NUPACK (Green, upward) and CFGRNAD (Blue, downward) for target structures of increasing lengths, clustered according to their structural sensitivity.

of success and, with very few notable exceptions, the raw numbers of perfect, good and acceptable designs produced by CFGRNAD seems to exceed that of NUPACK. One should however remain cautious before extrapolating this claim for larger RNAs, since the probability of producing perfect designs through random generation is expected to decrease exponentially with the length.

5. CONCLUSION

In this work, we have proposed a novel approach based on language-theoretical tools to perform a rational design of RNA under a biologically-motivated set of constraints. Namely, we showed that large set of diverse constraints could be addressed within a unifying framework, leading to an algorithm whose time and space requirements scale linearly with the length of designed secondary structures. Furthermore, such a framework paves the road for novel extensions, and our program is currently the only available tool to perform RNA design with forced/forbidden motifs. Such a feature has proven its utility in the design of sequences that have been tested *in vitro*.

A first extension of this work may assess the impact of, possibly stringent, constraints on the probability of actually folding into the MFE. In the absence of such constraints, it was shown that a substantial proportion of sequences designed using a weighted random generation scheme (also denoted as *global sampling strategy*) actually folds back *in silico* into the target structure [18, 22]. The success rate is sufficiently high to make the approach competitive against the alternatives based on various heuristics or exponential-time exact resolution, especially when additional design goals are captured [22]. However, certain constraints may possibly have a drastic, non-trivial, effect on the success rate, for instance by preventing motifs favored by stable folds from forming (e.g. forbidding GC, CG, CC motifs would forbid any occurrence of a $\frac{\mathsf{GC}}{\mathsf{GC}}$ stacking pairs). Such an effect might also be interpreted in the light of evolution, as the low designability (i.e. evolutionary accessibility) of experimental secondary structures may indicate a strong selective pressure, and suggest an essential position in the RNA regulatory network.

Another illustration of the flexibility of our approach is

the inclusion of structural (possibly non-canonical) motifs, which could be expressed as multi-strand (or gapped) motifs, contextualized by the target secondary structure. These may be prescribed either at a specific location in the sequence, or anywhere, depending on the intended application. The design of sequences of variable lengths may also constitute a natural extension, at the cost of more demanding algorithms. Future developments will aim at capturing more sophisticated free-energy models, such as the full Turner model, or crossing interactions (pseudoknots), relying on other language-theoretical formalisms such as multiple grammars.

Acknowledgment

The authors would like to thank James Regis (LIX, Polytechnique) for technical help with the webserver infrastructure.

6. REFERENCES

[1] Aho, A.V., Corasick, M.J.: Efficient string matching: an aid to bibliographic search. Commun. ACM 18(6), 333–340 (Jun 1975), http://doi.acm.org/10.1145/360825.360855

[2] Andronescu, M., Fejes, A.P., Hutter, F., Hoos, H.H., Condon, A.: A new algorithm for RNA secondary structure design. J Mol Biol. 336(3), 607–624 (2004)

[3] Avihoo, A., Churkin, A., Barash, D.: RNAexinv: An extended inverse RNA folding from shape and physical attributes to sequences. BMC Bioinformatics 12, 319 (2011)

[4] Busch, A., Backofen, R.: INFO-RNA–a fast approach to inverse RNA folding. Bioinformatics 22(15), 1823–18310 (2006)

[5] Busch, A., Backofen, R.: INFO-RNA–a server for fast inverse RNA folding satisfying sequence constraints. Nucleic Acids Res. 35, W310–313 (2007), web Server issue

[6] Dai, D.C., Tsang, H.H., Wiese, K.C.: rnaDesign: Local search for RNA secondary structure design. In: CIBCB. pp. 1–7 (2009)

[7] Darty, K., Denise, A., Ponty, Y.: VARNA: Interactive drawing and editing of the RNA secondary structure. Bioinformatics 25(15), 1974–1975 (2009)

[8] Denise, A., Ponty, Y., Termier, M.: Controlled non uniform random generation of decomposable structures. Theoretical Computer Science 411(40-42), 3527–3552 (2010)

[9] Ehsani, A., Saetrom, P., Zhang, J., Alluin, J., Li, H., Snøve, Jr, O., Aagaard, L., Rossi, J.J.: Rational design of micro-RNA-like bifunctional siRNAs targeting HIV and the HIV coreceptor CCR5. Mol Ther 18(4), 796–802 (Apr 2010)

[10] Elowitz, M., Lim, W.A.: Build life to understand it. Nature 468(7326), 889–90 (Dec 2010)

[11] Fairbrother, W.G., Yeo, G.W., Yeh, R., Goldstein, P., Mawson, M., Sharp, P.A., Burge, C.B.: RESCUE-ESE identifies candidate exonic splicing enhancers in vertebrate exons. Nucleic Acids Res. 32, W187–W190 (2004)

[12] Gao, J., Li, L., , Reidys, C.: Inverse folding of RNA pseudoknot structures. Algorithms for Molecular Biology 5(27) (2010)

[13] Garcia-Martin, J.A., Clote, P., Dotu, I.: RnaIFold: A constraint programming algorithm for rna inverse folding and molecular design. J Bioinform Comput Biol 11(2), 1350001 (Apr 2013)

[14] Guo, P.: The emerging field of RNA nanotechnology. Nat Nanotechnol 5(12), 833–42 (Dec 2010)

[15] Hofacker, I.L., Fontana, W., Stadler, P.F., Bonhoeffer, L.S., Tacker, M., Schuster, P.: Fast folding and comparison of RNA secondary structures. Monatsh. Chem. 125, 167–188 (1994)

[16] Khalil, A.S., Collins, J.J.: Synthetic biology: applications come of age. Nat Rev Genet 11(5), 367–79 (May 2010)

[17] Lee, M.T., Kim, J.: Self containment, a property of modular RNA structures, distinguishes microRNAs. PLoS Comput Biol 4(8), e1000150 (2008), http://dx.doi.org/10.1371/journal.pcbi.1000150

[18] Levin, A., Lis, M., Ponty, Y., O'Donnell, C.W., Devadas, S., Berger, B., Waldispühl, J.: A global sampling approach to designing and reengineering RNA secondary structures. Nucleic Acids Res 40(20), 10041–52 (Nov 2012)

[19] Liu, W., Zhou, Y., Hu, Z., Sun, T., Denise, A., Fu, X.D., Zhang, Y.: Regulation of splicing enhancer activities by RNA secondary structures. FEBS Letters 584(21), 4401–4407 (2010)

[20] Lyngsø, R.B., Anderson, J.W.J., Sizikova, E., Badugu, A., Hyland, T., Hein, J.: Frnakenstein: multiple target inverse RNA folding. BMC Bioinformatics 13, 260 (2012)

[21] Ponty, Y., Termier, M., Denise, A.: GenRGenS: Software for generating random genomic sequences and structures. Bioinformatics 22, 1534–1535 (2006)

[22] Reinharz, V., Ponty, Y., Waldispühl, J.: A weighted sampling algorithm for the design of rna sequences with targeted secondary structure and nucleotide distribution. Bioinformatics 29(13), i308–i315 (Jul 2013), http://dx.doi.org/10.1093/bioinformatics/btt217

[23] Salomaa, A.: Formal languages. Academic Press, New York (1973), 335 pp

[24] Schnall-Levin, M., Chindelevitch, L., Berger, B.: Inverting the Viterbi algorithm: an abstract framework for structure design. In: ICML. pp. 904–911 (2008)

[25] Win, M.N., Smolke, C.D.: Higher-order cellular information processing with synthetic RNA devices. Science 322(5900), 456–60 (Oct 2008)

[26] Zadeh, J.N., Steenberg, C.D., Bois, J.S., Wolfe, B.R., Pierce, M.B., Khan, A.R., Dirks, R.M., Pierce, N.A.: NUPACK: Analysis and design of nucleic acid systems. J Comput Chem 32(1), 170–3 (Jan 2011)

[27] Zadeh, J.N., Wolfe, B.R., Pierce, N.A.: Nucleic acid sequence design via efficient ensemble defect optimization. J Comput Chem 32(3), 439–52 (Feb 2011)

[28] Zuker, M., Stiegler, P.: Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. Nucleic Acids Res 9(1), 133–48 (Jan 1981)