Flexible RNA design under structure and sequence constraints using a language theoretic approach

Yu Zhou[♣,♢,⊙] Yann Ponty[•] Stéphane Vialette[†]
Jérôme Waldispühl[‡] Yi Zhang[♢] Alain Denise[♣]

[♣]LRI, Univ. Paris-Sud, Orsay F-91405, France

[♢]State Key Laboratory of Virology, Wuhan University, China

[⊙] Cellular and Molecular Medicine, Univ. of California San Diego, USA

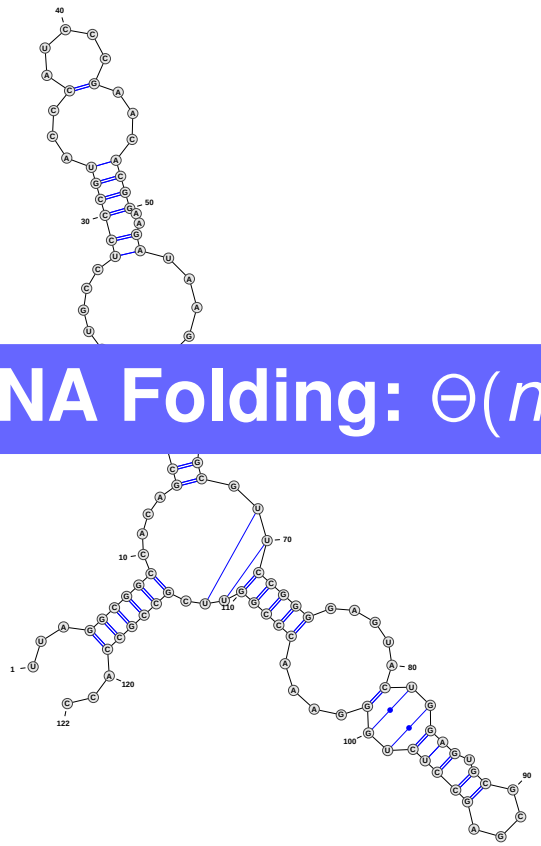
[•] LIX, CNRS/Ecole Polytechnique, France + PIMS, Vancouver, Canada

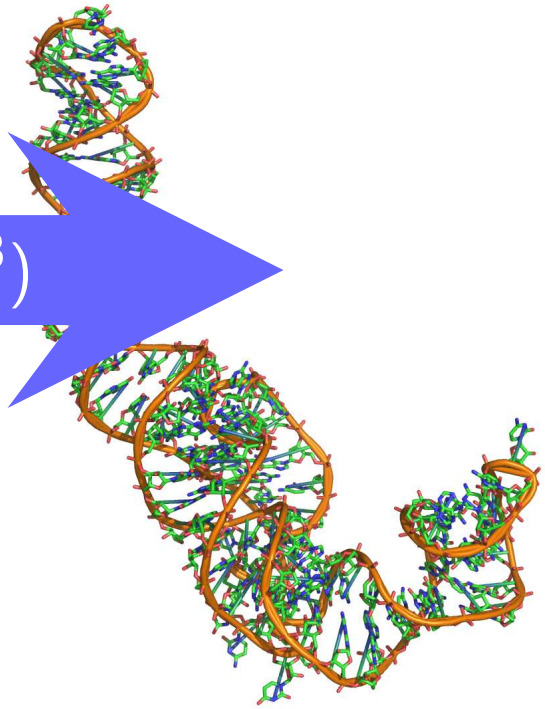
[†] LIGM, Université Paris-Est, France

[‡] Computer Science, Mc Gill University, Canada

RNA structure

UUAGGCGGCCACAGC
GGUGGGGU
CGUACCCA
CACGGAAGAUAAGCC
CACCAGCGUUCCGGG
GAGUACUGGAGUGCG
CGAGCCUCUGGGAAA
CCCGGUUCGCCGCCA
CC





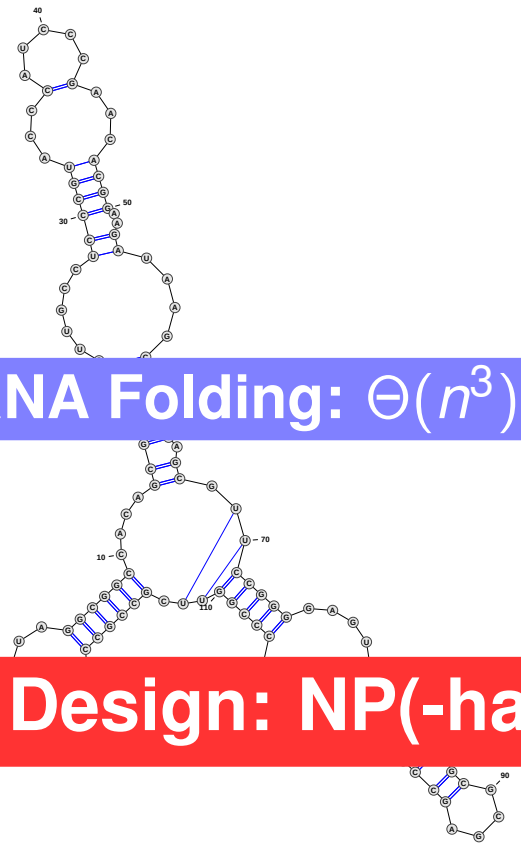
RNA Folding: $\Theta(n^3)$

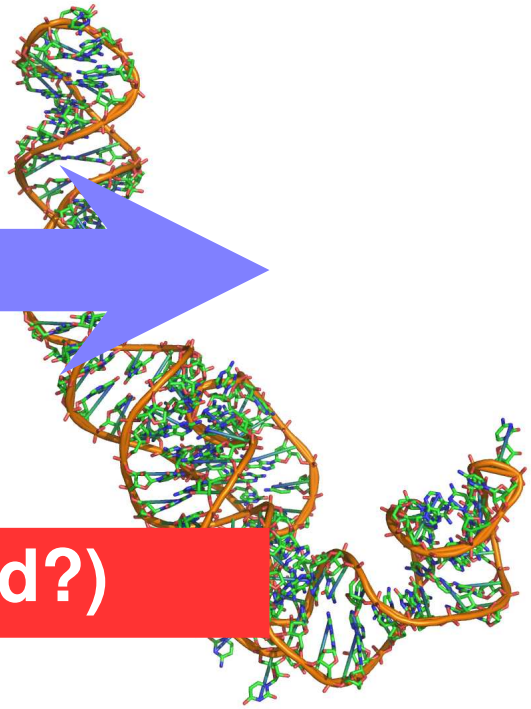
Primary structure | Secondary structure | Tertiary structure

5s rRNA (PDBID: 1K73:B)

RNA structure

UUAGGCGGCCACAGC
GGUGGGGU
CGUACCCAUCCCGAA
CACGGAAGAUAAGCC
CACCAGCGUUCCGGG
GAGUACUGGAGUGCC
CGAGCCUCUC
CCCGGUUCGCCG
CC





RNA Folding: $\Theta(n^3)$

RNA Design: NP(-hard?)

Primary structure | Secondary structure | Tertiary structure

5s rRNA (PDBID: 1K73:B)

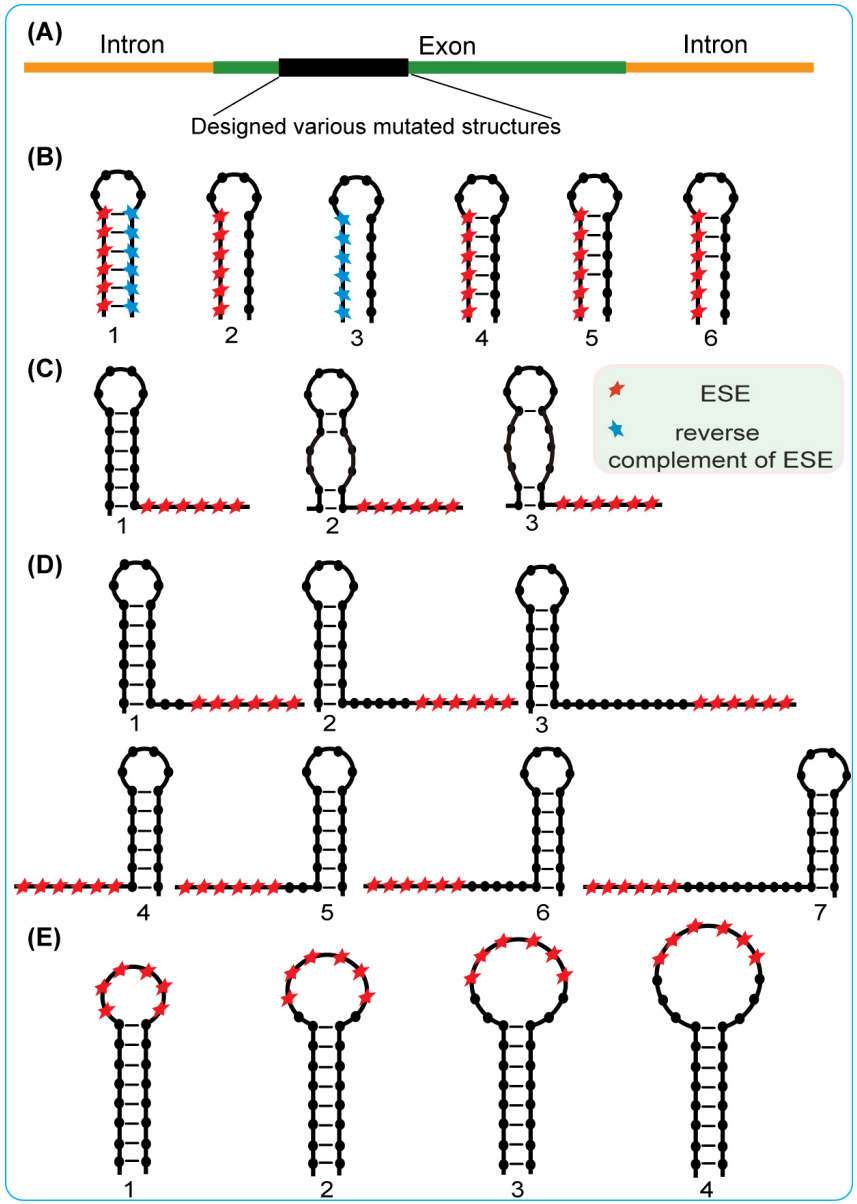
Motivation for constrained design

Motivation: Impact of structure on function of Exon Splicing Enhancers (ESE).

- Different structures
- Presence of ESE motif
- Avoid other ESEs (∼1500!)

Structural context of ESE motif in transcript was shown to affects its functionality.

[Liu *et al*, FEBS Lett. 2010]



Design goals

Positive design

Optimize **affinity** of designed sequences towards target structure

Negative design

Limit affinity of designs towards **alternative structures**

Additional constraints:

- **Forbid** motif list to appear **anywhere** in design
- **Force** motif list to appear **each at least once**
- **Limit** available alternatives for certain positions

Existing approaches for negative design

Based on local search...

- RNAInverse
- Info-RNA
- RNA-SSD
- NUPack

...genetic algorithms...

- RNAFBinv
- FRNAKenstein

...or exact approaches

- RNAIFold
- CO4

Few algorithms support avoided/mandatory motifs...

...none guarantees *reasonable* runtimes for any RNA/motifs set.

Typical reasons:

- Mandatory motifs ⇒ Late deadends (Branch and Bound)
- Forbidden motifs ⇒ Search space disconnection (Local Search)

Existing approaches for negative design

Based on local search...

- RNAInverse
- Info-RNA
- RNA-SSD
- NUPack

...genetic algorithms...

- RNAFBinv
- FRNAKenstein

...or exact approaches

- RNAIFold
- CO4

Few algorithms support avoided/mandatory motifs...

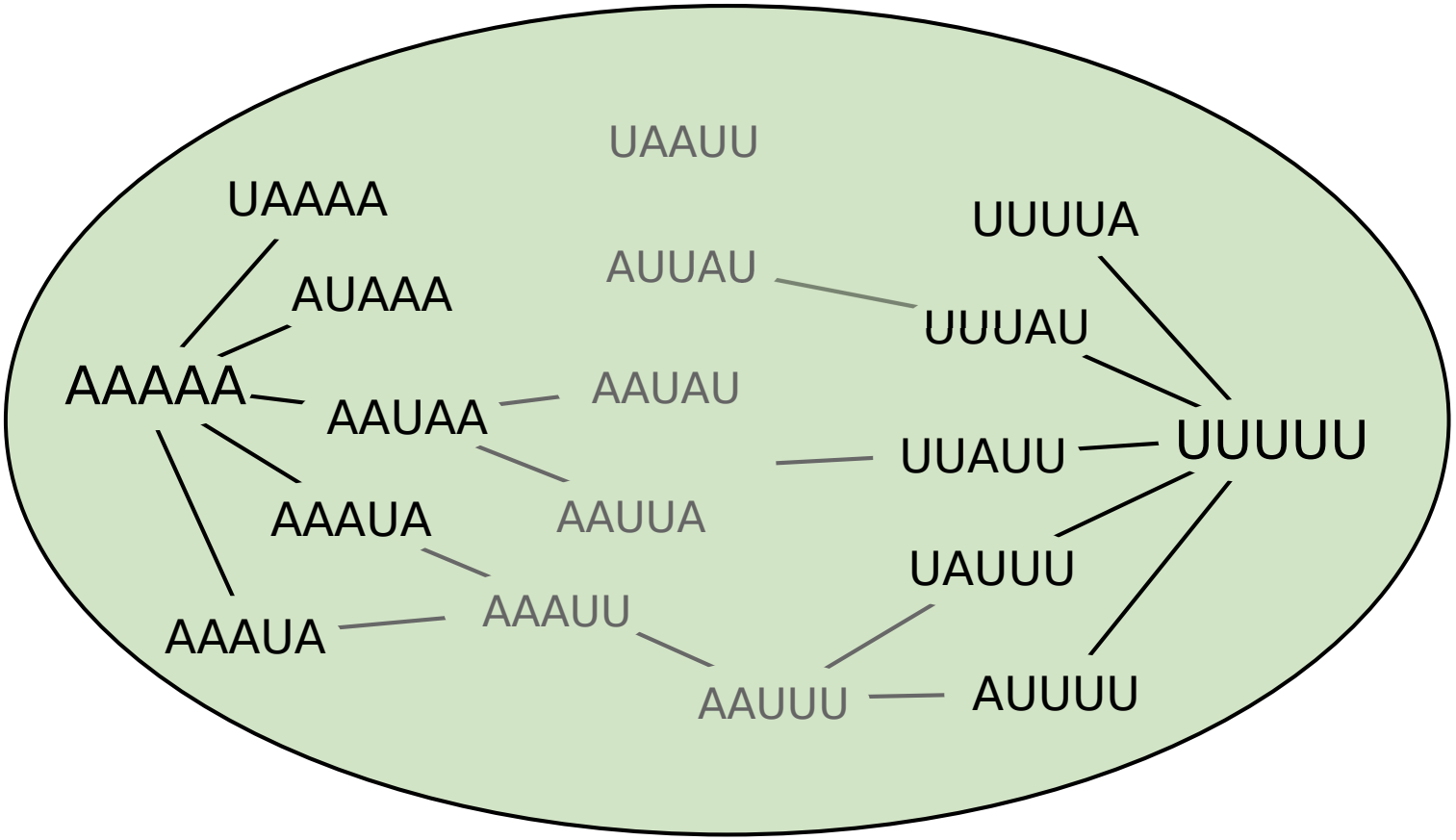
...none guarantees *reasonable* runtimes for any RNA/motifs set.

Typical reasons:

- Mandatory motifs $\Rightarrow$ Late deadends (Branch and Bound)
- Forbidden motifs $\Rightarrow$ Search space disconnection (Local Search)

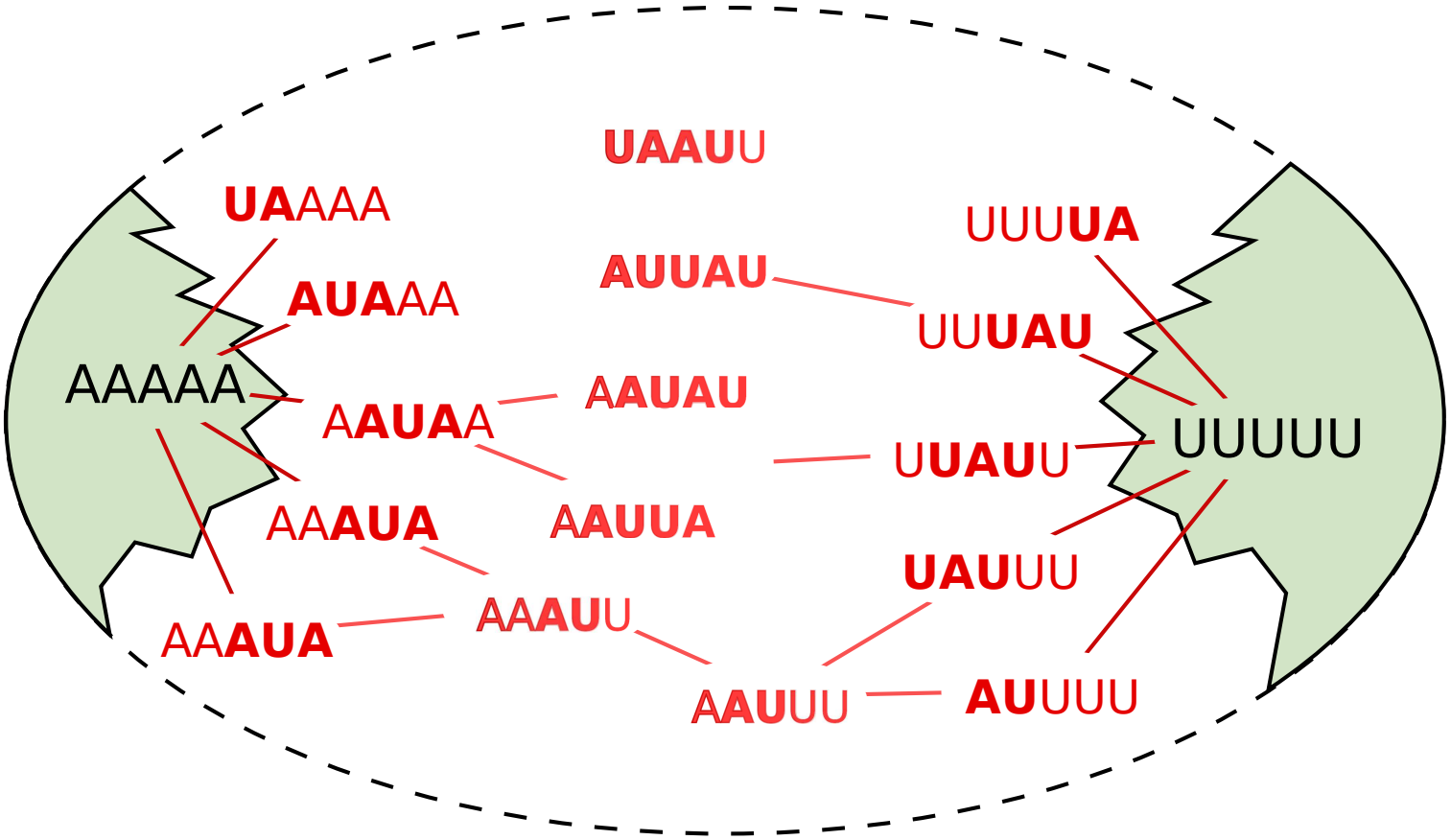
Problem with local approaches: An example

Simplified vocabulary $\{A, U\}$



Problem with local approaches: An example

Simplified vocabulary $\{A, U\}$ **+** **Forbidden motifs** $\mathcal{F} = \{AU, UA\}$



$\Rightarrow \mathcal{F}$ may **disconnect** search space (holds for **any** move set!)

Global Sampling [Levin *et al*, NAR 12]

- **Boltzmann Distribution (BD)** based on **affinity** towards target S
- **Generate at random** candidate sequences w.r.t. **BD**
- **Refold** generated sequences and **compare** to target

Boltzmann factor:

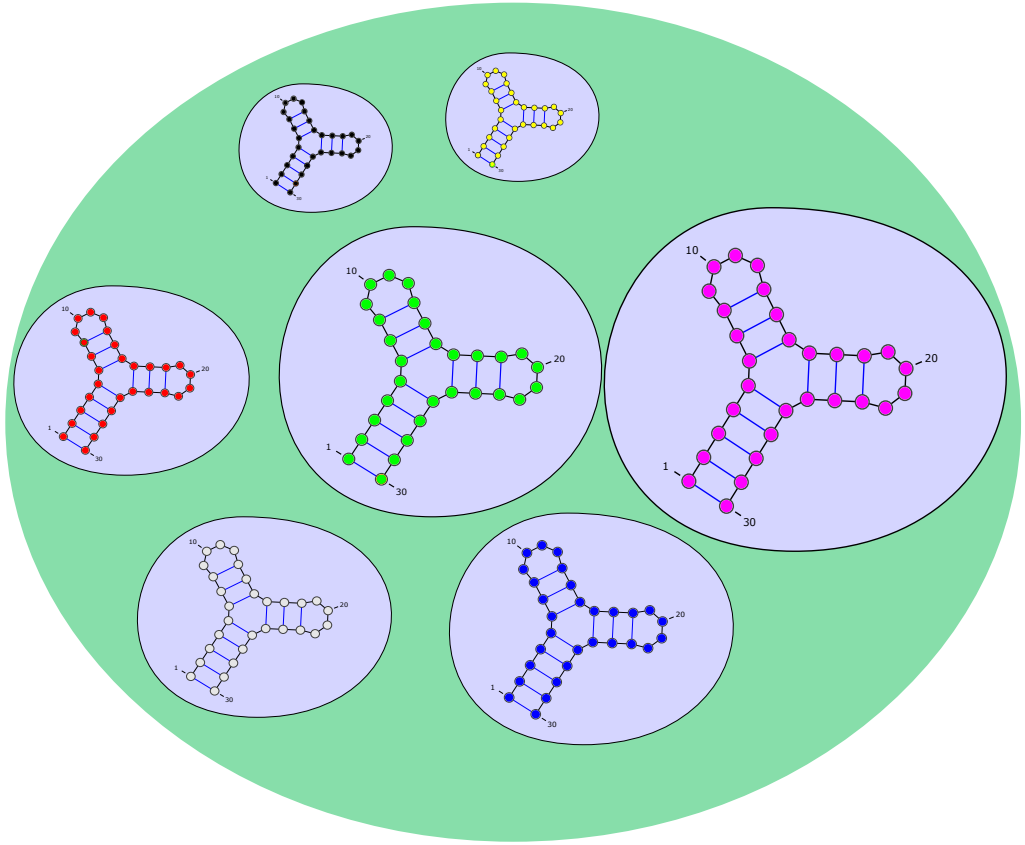
$$\mathcal{B}(s) := e^{\frac{-E(s)}{RT}}$$

Partition function:

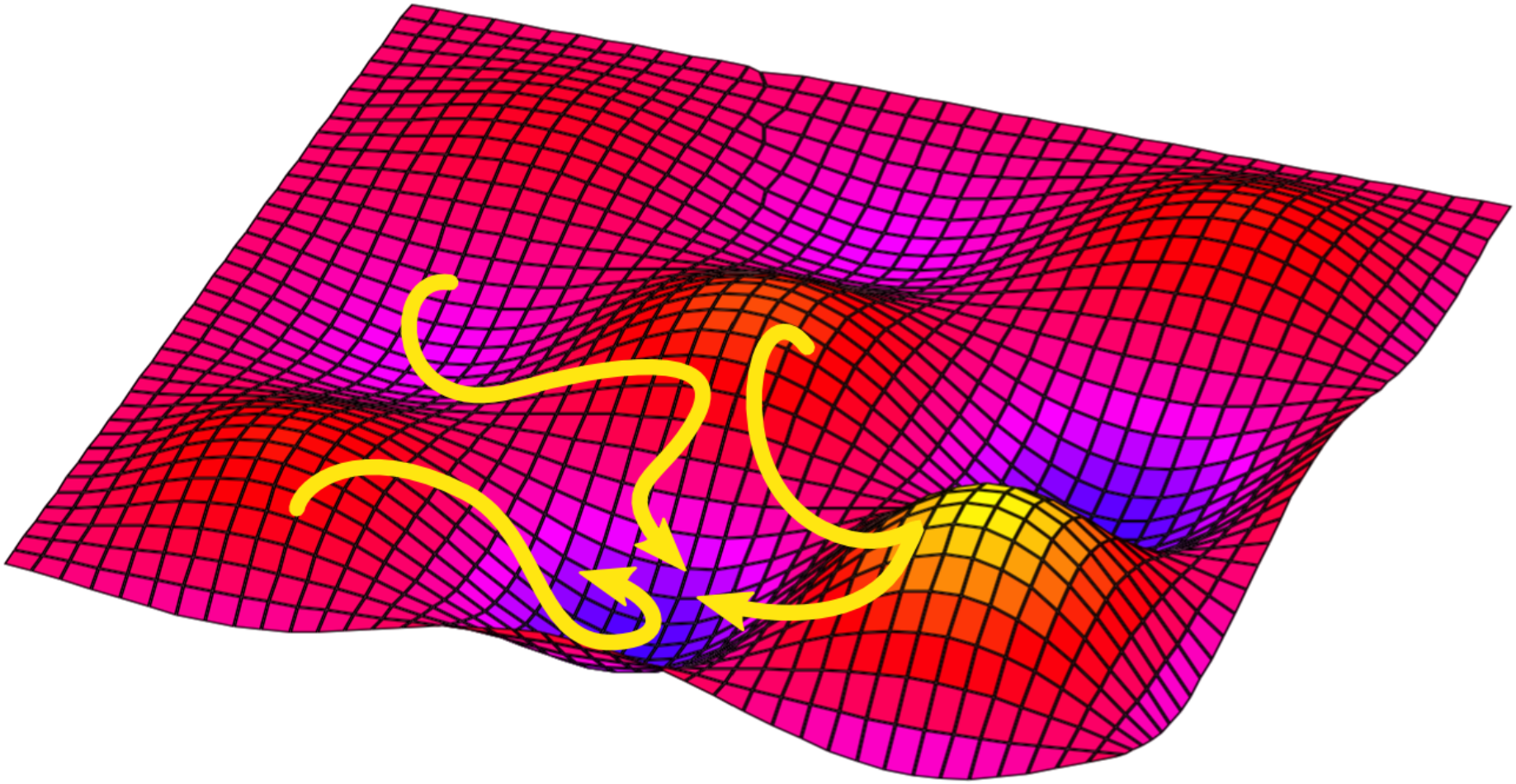
$$\mathcal{Z} = \sum_{s} \mathcal{B}(s)$$

Boltzmann probability:

$$p(s) := \frac{\mathcal{B}(s)}{\mathcal{Z}}$$

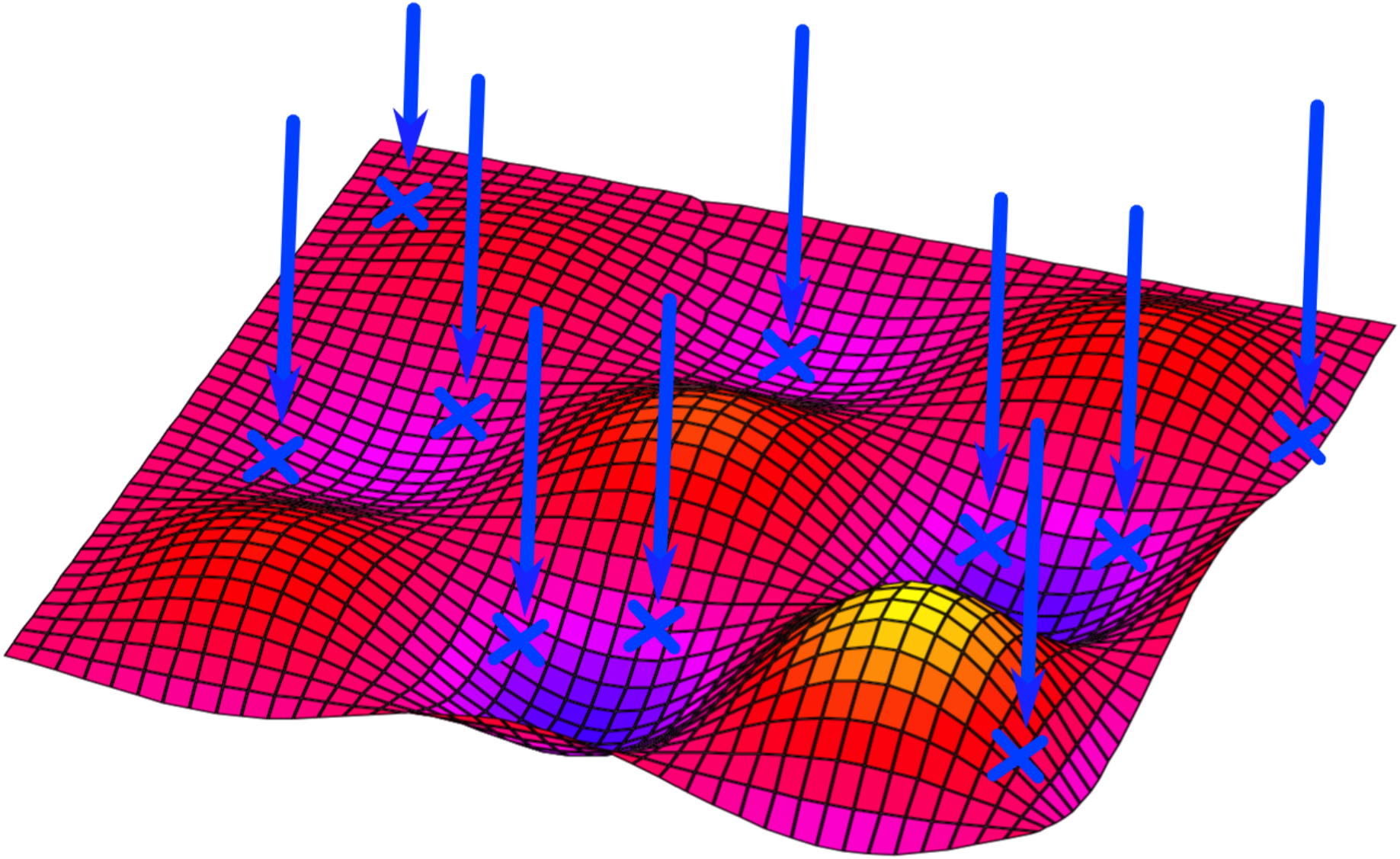


Global Sampling [Levin *et al*, NAR 12; Reinharz *et al*, ISMB 13]

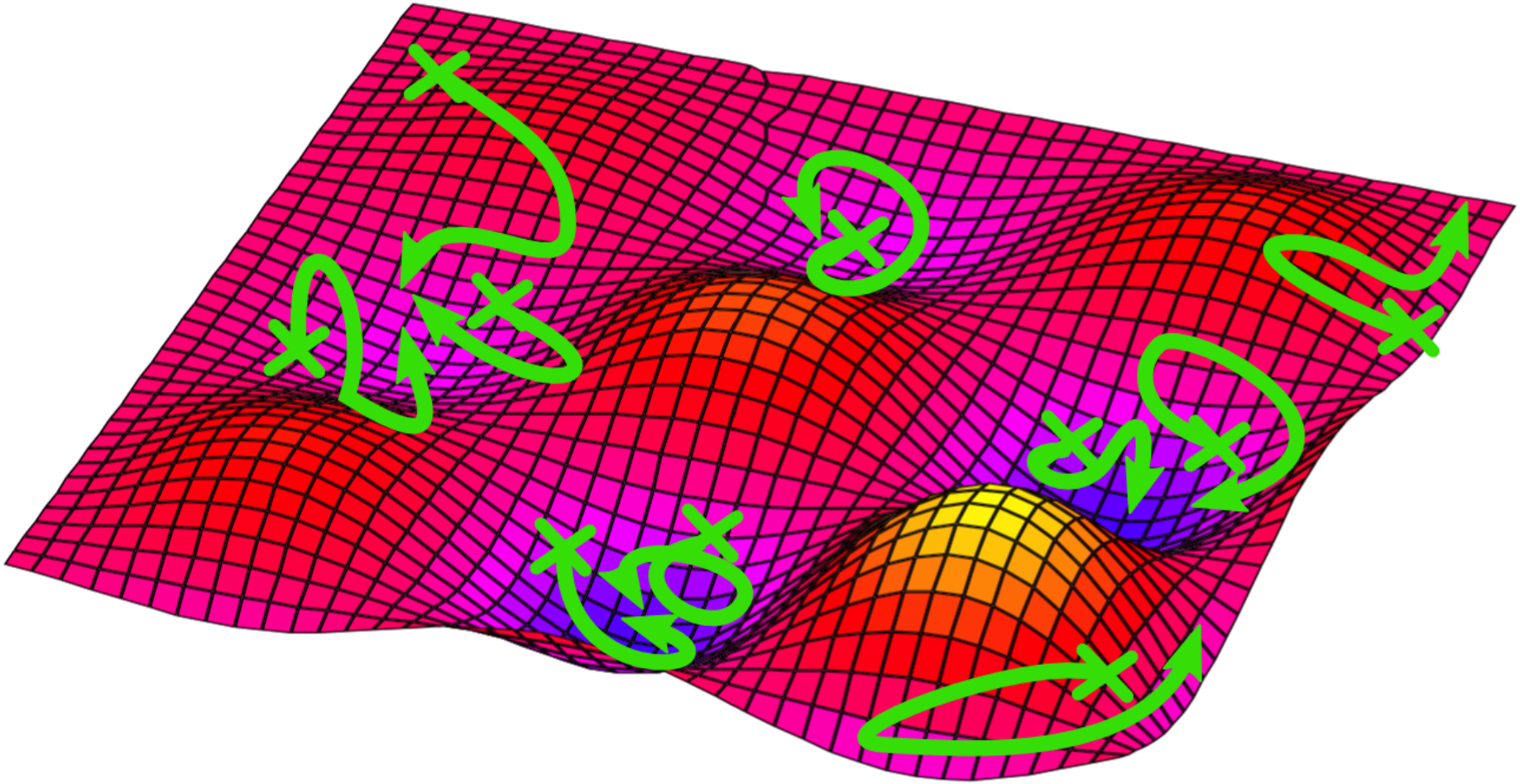


Global Sampling [Levin *et al*, NAR 12; Reinharz *et al*, ISMB 13]

Complementarity: + Diversity + Flexible composition - Accuracy



Glocal approach [Reinharz *et al*, ISMB 13]



Outline

Use formal language constructs to constrain global sampling

Forced motifs
Avoided motifs

$\rightarrow$ Regular language $\mathcal{L}_C \in$ Reg

Structure compatibility
+ Positional constraints
+ Energy Model

$\rightarrow$ **Weighted** Context-Free Lang $\mathcal{L}_S \in$ CFL

Folklore theorem (constructive): Reg $\cap$ (**W**)CFL $\subseteq$ (**W**)CFL

Build weighted context-free grammar $\mathcal{G}$ for $\mathcal{L}_C \cap \mathcal{L}_S$
+ Random generation
$\Rightarrow$ Global sampling under constraints

Outline

Use formal language constructs to constrain global sampling

Forced motifs
Avoided motifs $\rightarrow$ Regular language $\mathcal{L}_C \in$ Reg

Structure compatibility
+ Positional constraints $\rightarrow$ **Weighted** Context-Free Lang $\mathcal{L}_S \in$ CFL
+ Energy Model

Folklore theorem (constructive): Reg $\cap$ (**W**)CFL $\subseteq$ (**W**)CFL

Build weighted context-free grammar $\mathcal{G}$ for $\mathcal{L}_C \cap \mathcal{L}_S$
+ Random generation
$\Rightarrow$ Global sampling under constraints

Outline

Use formal language constructs to constrain global sampling

Forced motifs
Avoided motifs $\rightarrow$ Regular language $\mathcal{L}_C \in$ Reg

Structure compatibility
+ Positional constraints $\rightarrow$ **Weighted** Context-Free Lang $\mathcal{L}_S \in$ CFL
+ Energy Model

Folklore theorem (constructive): Reg $\cap$ (**W**)CFL $\subseteq$ (**W**)CFL

Build weighted context-free grammar $\mathcal{G}$ for $\mathcal{L}_C \cap \mathcal{L}_S$
+ Random generation
$\Rightarrow$ Global sampling under constraints

Outline

Use formal language constructs to constrain global sampling

Forced motifs
Avoided motifs $\rightarrow$ Regular language $\mathcal{L}_C \in$ Reg

Structure compatibility
+ Positional constraints $\rightarrow$ **Weighted** Context-Free Lang $\mathcal{L}_S \in$ CFL
+ Energy Model

Folklore theorem (constructive): Reg $\cap$ (**W**)CFL $\subseteq$ (**W**)CFL

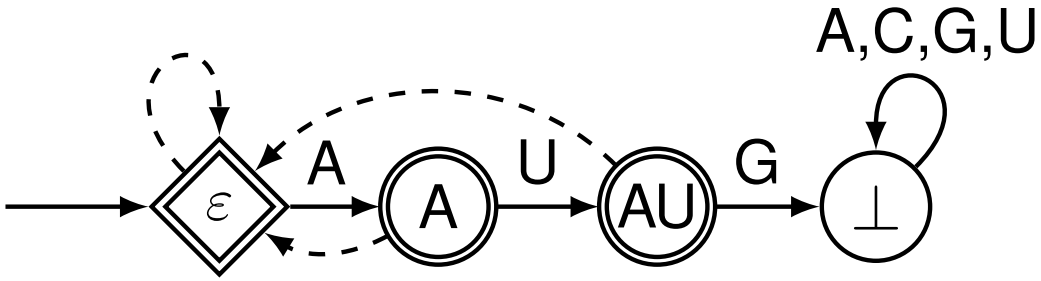
Build weighted context-free grammar $\mathcal{G}$ for $\mathcal{L}_C \cap \mathcal{L}_S$
+ Random generation
$\Rightarrow$ Global sampling under constraints

Building the Finite State Automaton

Forbidden motifs $\mathcal{F} = \{f_1, f_2 \ldots\}$ Mandatory motifs $\mathcal{M} = \{m_1, m_2 \ldots\}$

- Avoiding a **single word** $\Rightarrow$ Linear automaton
- Avoiding **multiple words** $\Rightarrow$ **Efficient Aho-Corasick** construct
- Forcing a **single word** $\Rightarrow$ Linear automaton (Complemented)
- Forcing **several words disjunctively** $\Rightarrow$ Aho-Corasick

Example: Avoid AUG ($\mathcal{L}_{\overline{\text{AUG}}}$)

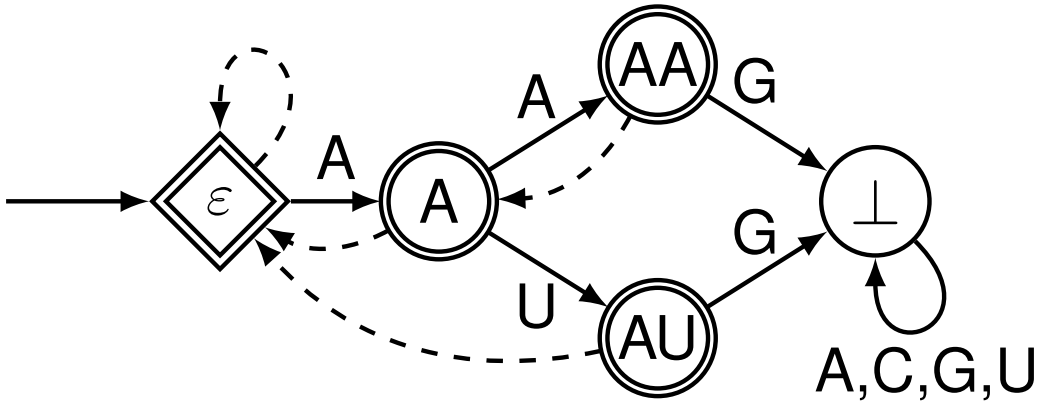


Building the Finite State Automaton

Forbidden motifs $\mathcal{F} = \{f_1, f_2 \ldots\}$ Mandatory motifs $\mathcal{M} = \{m_1, m_2 \ldots\}$

- Avoiding a **single word** $\Rightarrow$ Linear automaton
- Avoiding **multiple words** $\Rightarrow$ **Efficient Aho-Corasick** construct
- Forcing a **single word** $\Rightarrow$ Linear automaton (Complemented)
- Forcing **several words disjunctively** $\Rightarrow$ Aho-Corasick

Example: Avoid AUG **+** AAG ($\mathcal{L}_{\overline{\text{AUG}}} \cap \mathcal{L}_{\overline{\text{AAG}}}$)

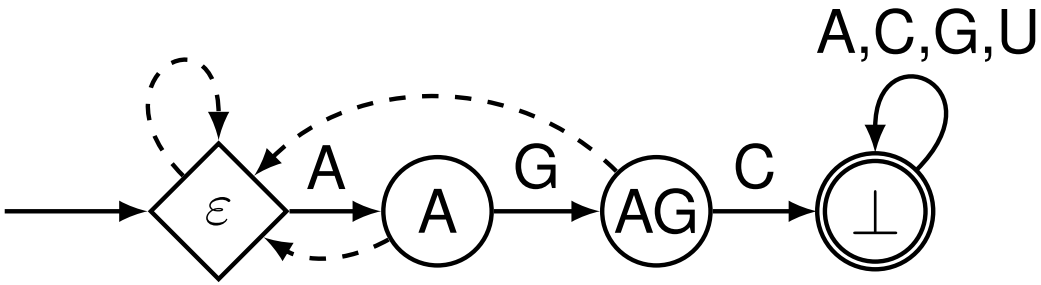


Building the Finite State Automaton

Forbidden motifs $\mathcal{F} = \{f_1, f_2 \ldots\}$ Mandatory motifs $\mathcal{M} = \{m_1, m_2 \ldots\}$

- Avoiding a **single word** $\Rightarrow$ Linear automaton
- Avoiding **multiple words** $\Rightarrow$ **Efficient Aho-Corasick** construct
- Forcing a **single word** $\Rightarrow$ Linear automaton (Complemented)
- Forcing **several words disjunctively** $\Rightarrow$ Aho-Corasick

Example: Forcing AGC ($\mathcal{L}_{\text{AGC}}$)

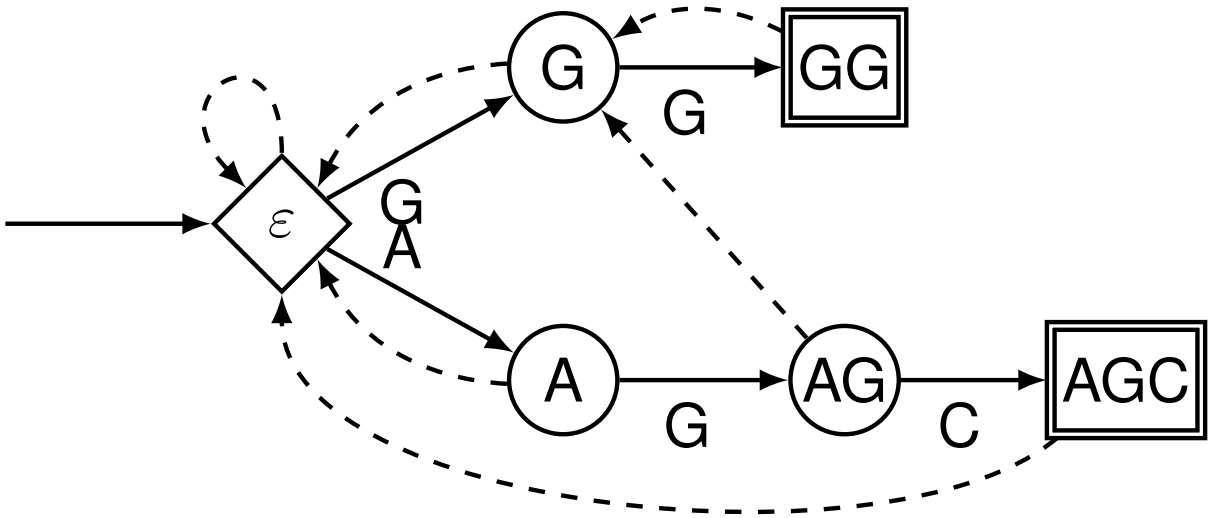


Building the Finite State Automaton

Forbidden motifs $\mathcal{F} = \{f_1, f_2 \ldots\}$ Mandatory motifs $\mathcal{M} = \{m_1, m_2 \ldots\}$

- Avoiding a **single word** $\Rightarrow$ Linear automaton
- Avoiding **multiple words** $\Rightarrow$ **Efficient Aho-Corasick** construct
- Forcing a **single word** $\Rightarrow$ Linear automaton (Complemented)
- Forcing **several words disjunctively** $\Rightarrow$ Aho-Corasick

Example: Forcing AGC **or** GG ($\mathcal{L}_{\text{AGC}} \bigcup \mathcal{L}_{\text{GG}}$)



Building the Finite State Automaton

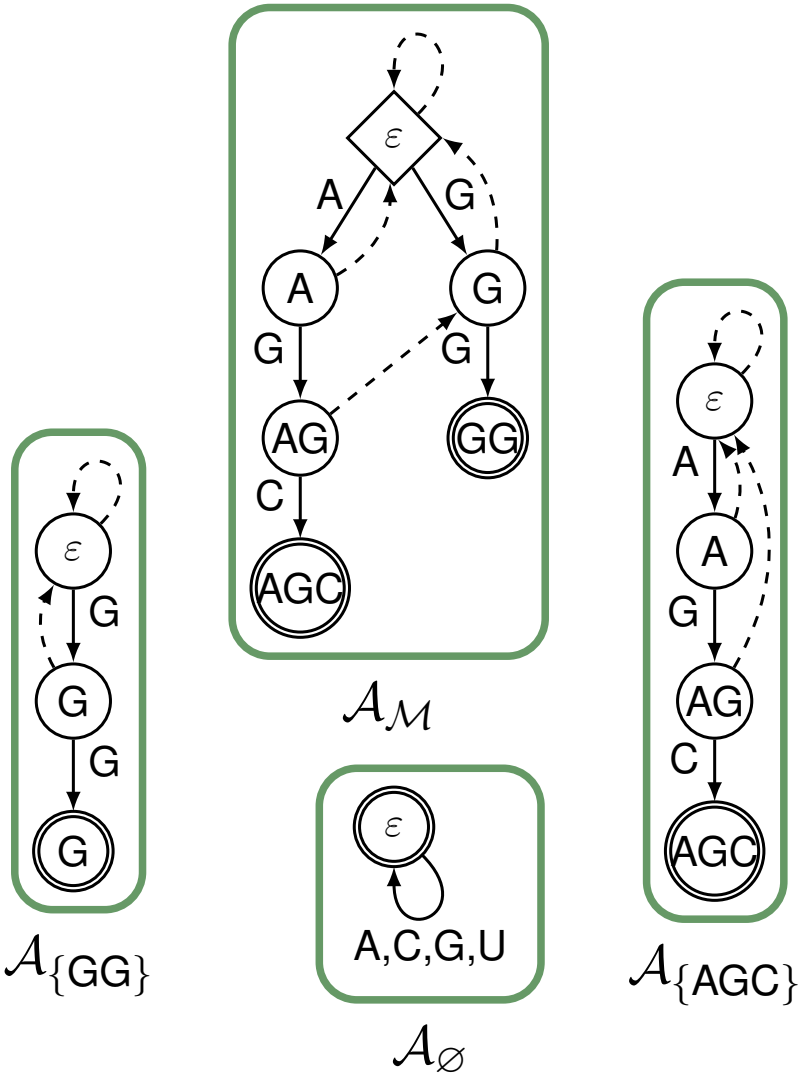
To force multiple words, **keep track** of generated words:

- Create disjunctive automaton for each $\mathcal{M}' \subseteq \mathcal{M}$
- **Reroute** accepting states
- Accepting state = no **forced word** remaining (ε in $\mathcal{A}_\varnothing$)
- Forbidden words can be added to sub-automata

#States:

$$O\left(2^{|\mathcal{M}|} \cdot \left(\sum_i |f_i| + \sum_j |m_j|\right)\right)$$

Example: $\mathcal{M} = \{\mathsf{AGC}, \mathsf{GG}\}$

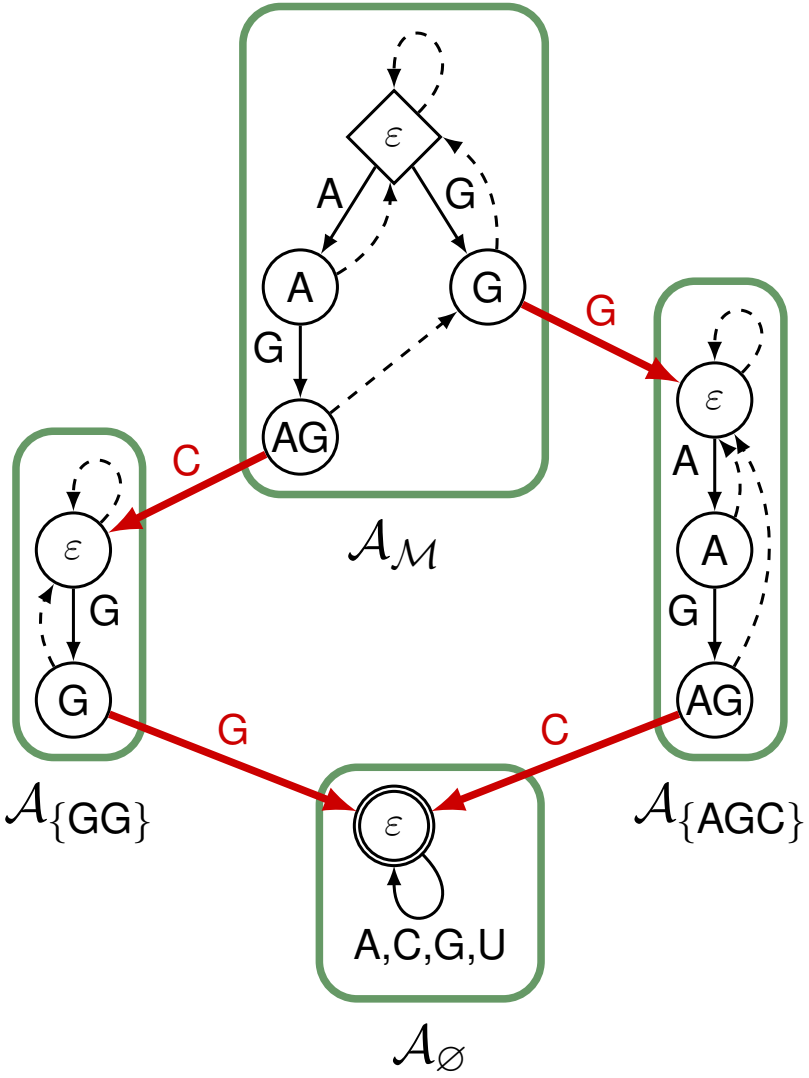


Building the Finite State Automaton

To force multiple words, **keep track** of generated words:

- Create disjunctive automaton for each $\mathcal{M}' \subseteq \mathcal{M}$
- **Reroute** accepting states
- Accepting state = no **forced word** remaining (ε in $\mathcal{A}_\varnothing$)
- Forbidden words can be added to sub-automata

Example: $\mathcal{M} = \{\mathsf{AGC}, \mathsf{GG}\}$



#States:

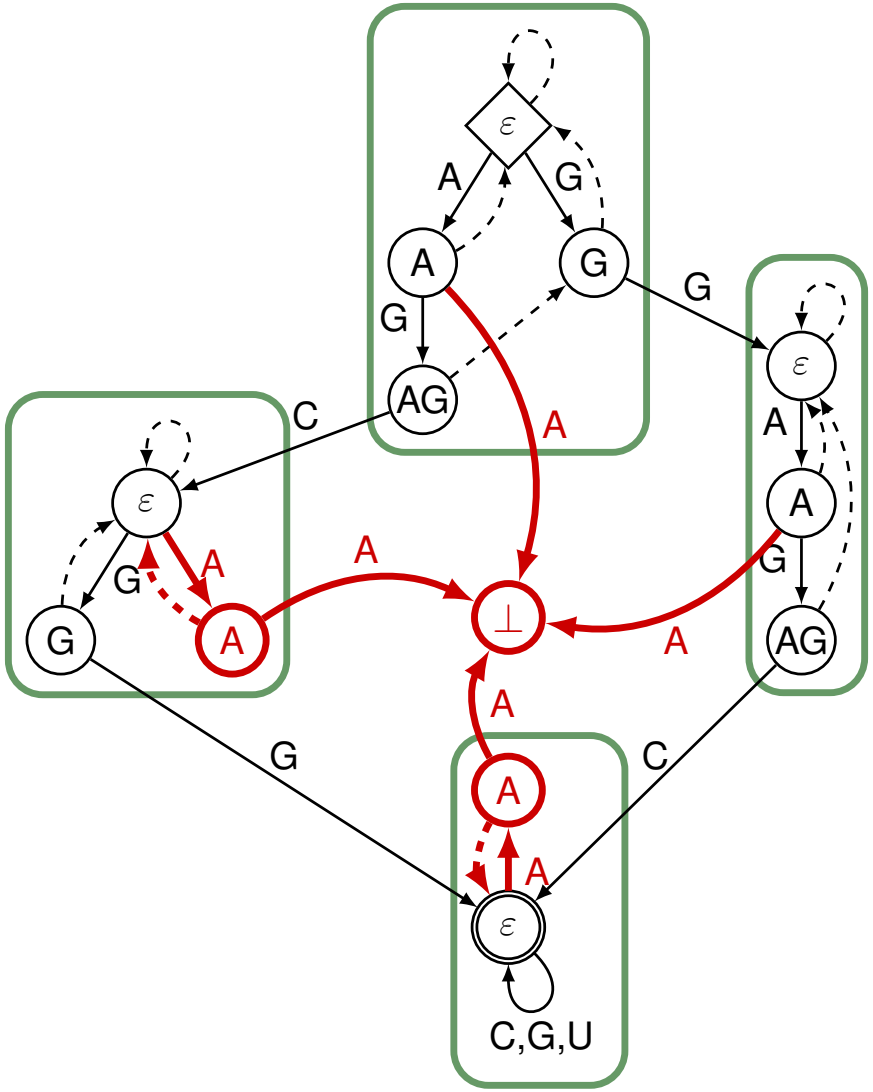
$$O\left(2^{|\mathcal{M}|} \cdot \left(\sum_i |f_i| + \sum_j |m_j|\right)\right)$$

Building the Finite State Automaton

To force multiple words, **keep track** of generated words:

- Create disjunctive automaton for each $\mathcal{M}' \subseteq \mathcal{M}$
- **Reroute** accepting states
- Accepting state = no **forced word** remaining (ε in $\mathcal{A}_\varnothing$)
- Forbidden words can be added to sub-automata

Example: $\mathcal{M} = \{\mathsf{AGC}, \mathsf{GG}\}; \mathcal{F} = \{\mathsf{AA}\}$



#States:

$$O\left(2^{|\mathcal{M}|} \cdot \left(\sum_i |f_i| + \sum_j |m_j|\right)\right)$$

Building the grammar

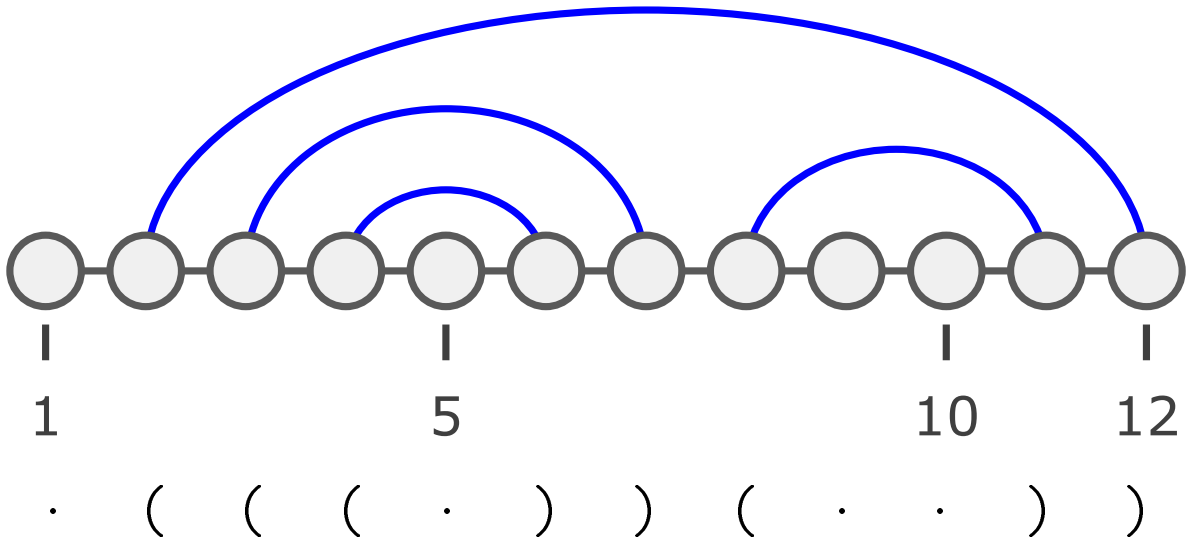
Input: Secondary Structure S + Positional constraints

A **Create Parse Tree** for secondary structure

B **Translate Parse Tree** into single-word grammar

C **Expand** grammar to instantiate compatible base/base-pairs

D **Restrict** to bases/base-pairs allowed at each position



Building the grammar

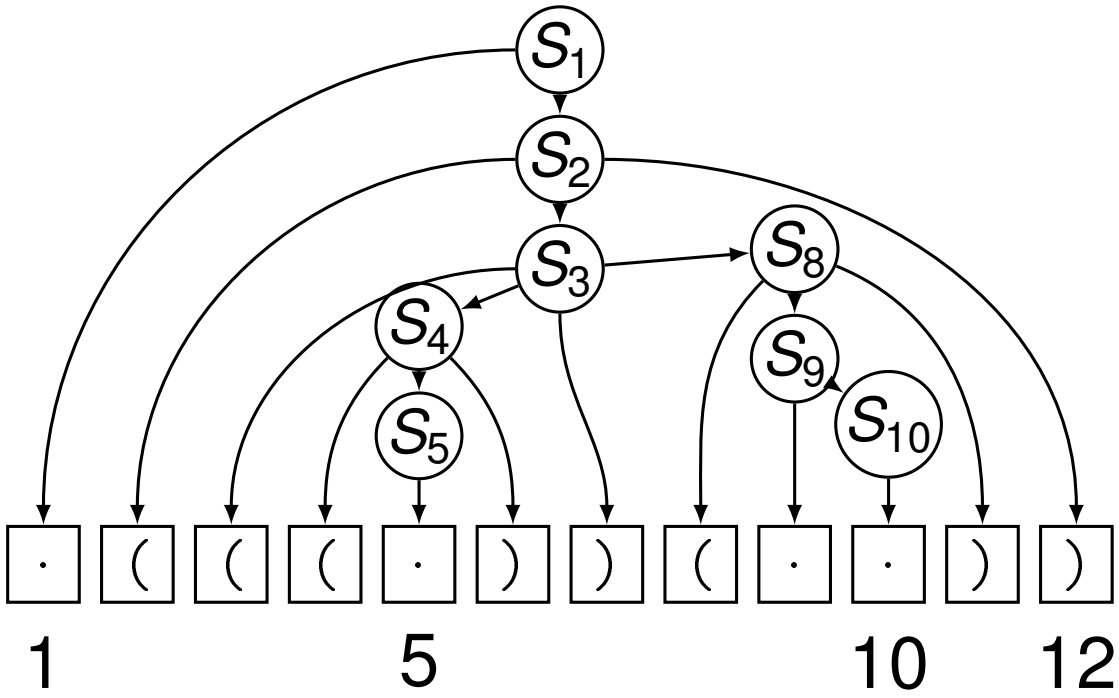
Input: Secondary Structure S + Positional constraints

A **Create Parse Tree** for secondary structure

B **Translate Parse Tree** into single-word grammar

C **Expand** grammar to instantiate compatible base/base-pairs

D **Restrict** to bases/base-pairs allowed at each position



Building the grammar

Input: Secondary Structure S + Positional constraints

A **Create Parse Tree** for secondary structure

B **Translate Parse Tree** into single-word grammar

C **Expand** grammar to instantiate compatible base/base-pairs

D **Restrict** to bases/base-pairs allowed at each position

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

Building the grammar

Input: Secondary Structure S + Positional constraints

A **Create Parse Tree** for secondary structure

B **Translate Parse Tree** into single-word grammar

C **Expand** grammar to instantiate compatible base/base-pairs

D **Restrict** to bases/base-pairs allowed at each position

$$
\begin{aligned}
V_1 &\to \mathrm{A}\ V_2 \mid \mathrm{C}\ V_2 \mid \mathrm{G}\ V_2 \mid \mathrm{U}\ V_2 \\
V_2 &\to \mathrm{A}\ V_3\ \mathrm{U} \mid \mathrm{C}\ V_3\ \mathrm{G} \mid \mathrm{G}\ V_3\ \mathrm{C} \mid \mathrm{G}\ V_3\ \mathrm{U} \mid \mathrm{U}\ V_3\ \mathrm{A} \mid \mathrm{U}\ V_3\ \mathrm{G} \\
V_3 &\to \mathrm{A}\ V_4\ \mathrm{U}\ V_8 \mid \mathrm{C}\ V_4\ \mathrm{G}\ V_8 \mid \mathrm{G}\ V_4\ \mathrm{C}\ V_8 \mid \mathrm{G}\ V_4\ \mathrm{U}\ V_8 \mid \mathrm{U}\ V_4\ \mathrm{A}\ V_8 \mid \mathrm{U}\ V_4\ \mathrm{G}\ V_8 \\
V_4 &\to \mathrm{A}\ V_5\ \mathrm{U} \mid \mathrm{C}\ V_5\ \mathrm{G} \mid \mathrm{G}\ V_5\ \mathrm{C} \mid \mathrm{G}\ V_5\ \mathrm{U} \mid \mathrm{U}\ V_5\ \mathrm{A} \mid \mathrm{U}\ V_5\ \mathrm{G} \\
V_5 &\to \mathrm{A} \mid \mathrm{C} \mid \mathrm{G} \mid \mathrm{U} \\
V_8 &\to \mathrm{A}\ V_9\ \mathrm{U} \mid \mathrm{C}\ V_9\ \mathrm{G} \mid \mathrm{G}\ V_9\ \mathrm{C} \mid \mathrm{G}\ V_9\ \mathrm{U} \mid \mathrm{U}\ V_9\ \mathrm{A} \mid \mathrm{U}\ V_9\ \mathrm{G} \\
V_9 &\to \mathrm{A}\ V_{10} \mid \mathrm{C}\ V_{10} \mid \mathrm{G}\ V_{10} \mid \mathrm{U}\ V_{10} \\
V_{10} &\to \mathrm{A} \mid \mathrm{C} \mid \mathrm{G} \mid \mathrm{U}
\end{aligned}
$$

Building the grammar

Input: Secondary Structure S **+ Positional constraints**

- **A** **Create Parse Tree** for secondary structure
- **B** **Translate Parse Tree** into single-word grammar
- **C** **Expand** grammar to instantiate compatible base/base-pairs
- **D** **Restrict** to bases/base-pairs allowed at each position

$$
\begin{aligned}
V_1 &\to \mathrm{A}\ V_2 \mid \mathrm{C}\ V_2 \mid \mathrm{G}\ V_2 \mid \mathrm{U}\ V_2 \\
V_2 &\to \mathrm{A}\ V_3\ \mathrm{U} \mid \cancel{\mathrm{C}\ V_3\ \mathrm{G}} \mid \cancel{\mathrm{G}\ V_3\ \mathrm{C}} \mid \cancel{\mathrm{G}\ V_3\ \mathrm{U}} \mid \mathrm{U}\ V_3\ \mathrm{A} \mid \cancel{\mathrm{U}\ V_3\ \mathrm{G}} \\
V_3 &\to \mathrm{A}\ V_4\ \mathrm{U}\ V_8 \mid \cancel{\mathrm{C}\ V_4\ \mathrm{G}\ V_8} \mid \cancel{\mathrm{G}\ V_4\ \mathrm{C}\ V_8} \mid \cancel{\mathrm{G}\ V_4\ \mathrm{U}\ V_8} \mid \mathrm{U}\ V_4\ \mathrm{A}\ V_8 \mid \cancel{\mathrm{U}\ V_4\ \mathrm{G}\ V_8} \\
V_4 &\to \mathrm{A}\ V_5\ \mathrm{U} \mid \cancel{\mathrm{C}\ V_5\ \mathrm{G}} \mid \cancel{\mathrm{G}\ V_5\ \mathrm{C}} \mid \cancel{\mathrm{G}\ V_5\ \mathrm{U}} \mid \mathrm{U}\ V_5\ \mathrm{A} \mid \cancel{\mathrm{U}\ V_5\ \mathrm{G}} \\
V_5 &\to \mathrm{A} \mid \mathrm{C} \mid \mathrm{G} \mid \mathrm{U} \\
V_8 &\to \mathrm{A}\ V_9\ \mathrm{U} \mid \cancel{\mathrm{C}\ V_9\ \mathrm{G}} \mid \cancel{\mathrm{G}\ V_9\ \mathrm{C}} \mid \cancel{\mathrm{G}\ V_9\ \mathrm{U}} \mid \mathrm{U}\ V_9\ \mathrm{A} \mid \cancel{\mathrm{U}\ V_9\ \mathrm{G}} \\
V_9 &\to \mathrm{A}\ V_{10} \mid \mathrm{C}\ V_{10} \mid \mathrm{G}\ V_{10} \mid \mathrm{U}\ V_{10} \\
V_{10} &\to \mathrm{A} \mid \mathrm{C} \mid \mathrm{G} \mid \mathrm{U}
\end{aligned}
$$

Random generation

Combine CFG and automaton $\rightarrow$ CFG (Multiplying #Rules by $|Q|^3$)
(W)CFG $\Rightarrow$ Recurrence on #words (resp. weight) of each NT

GenRGenS [Ponty *et al*, Bioinformatics 06]:

- Precomputes #words for each non-terminal
- Random Generation w.r.t. **weighted distribution**

Uniform distribution: $\mathrm{A}, \mathrm{C}, \mathrm{G}$ or U anywhere $\rightarrow$ 1

Nussinov energy model:
$$\begin{cases} \mathrm{G}\cdot\mathrm{U} \text{ or } \mathrm{U}\cdot\mathrm{G} & \rightarrow & e^{1/RT} \\ \mathrm{A}\cdot\mathrm{U} \text{ or } \mathrm{U}\cdot\mathrm{A} & \rightarrow & e^{2/RT} \\ \mathrm{G}\cdot\mathrm{C} \text{ or } \mathrm{C}\cdot\mathrm{G} & \rightarrow & e^{3/RT} \end{cases}$$

Stacking-pairs model (Turner 2004) $\mathrm{WX}\cdot\mathrm{YZ} \rightarrow e^{-E^{\mathrm{XY}}_{\mathrm{WZ}}/RT}$
(Based on refined, yet similar, grammar)

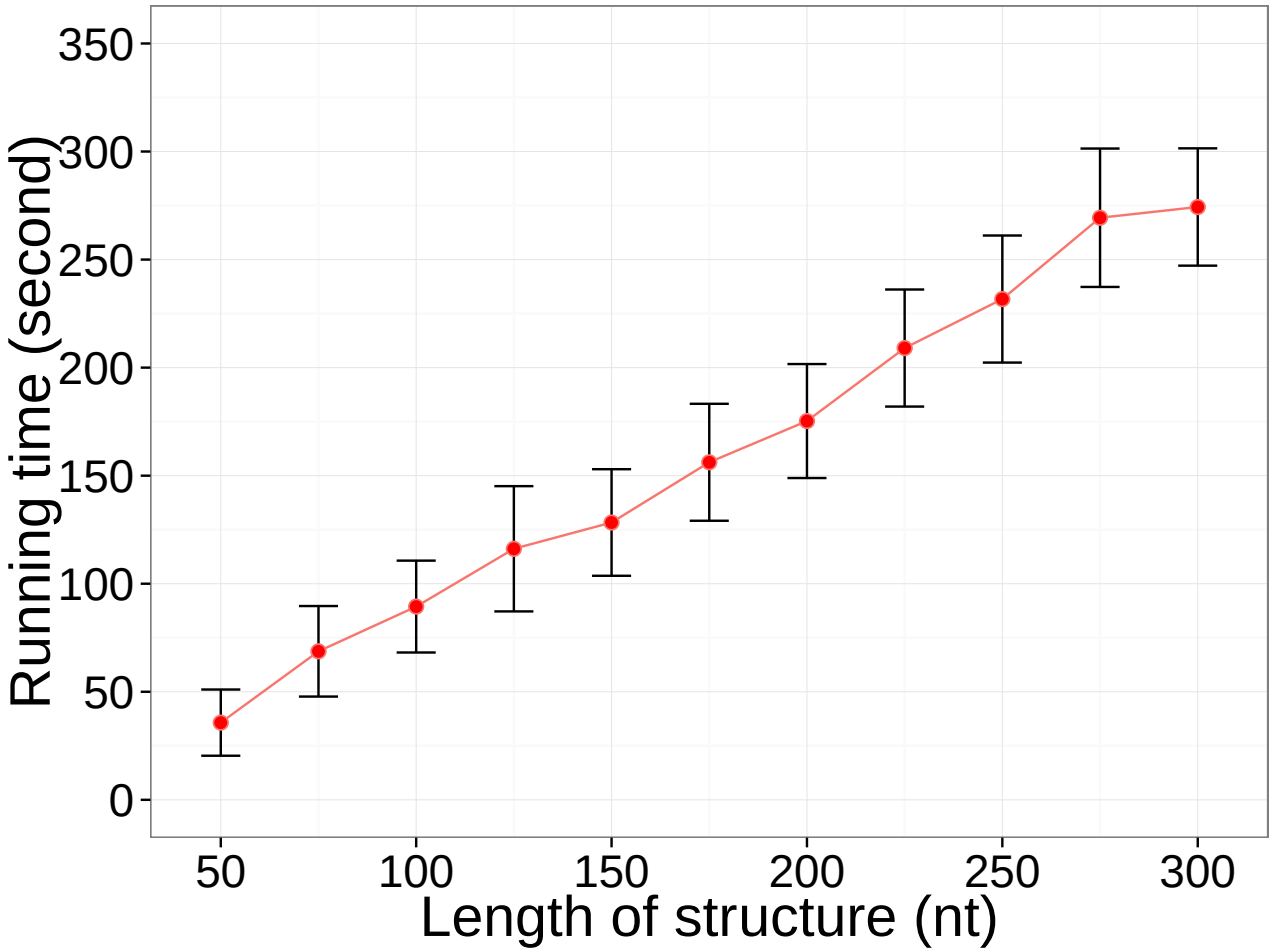
Complexity

- **Automaton:** Time $\approx$ #States $|Q| \in \mathcal{O}\left(2^{|\mathcal{M}|} \cdot \left(\sum_i |f_i| + \sum_j |m_j|\right)\right)$
- **Initial CFG:** #Rules $\approx$ #Non Term. $\in \Theta(n)$
- **Final CFG:** #Non Term. $|N| \in \mathcal{O}(n \cdot |Q|^2)$, #Rules $|R| \in \mathcal{O}(n \cdot |Q|^3)$
- **Random generation:**
  Any NT fully determines the length of its produced words.

  Preprocessing: $\Theta(|R|)$ time/$\Theta(|N|)$ space
  Generation: $\Theta(n \cdot |Q|)$ time for each design

Complete algorithm has **linear overall complexity**
but **total length of motifs** impacts preprocessing stage.

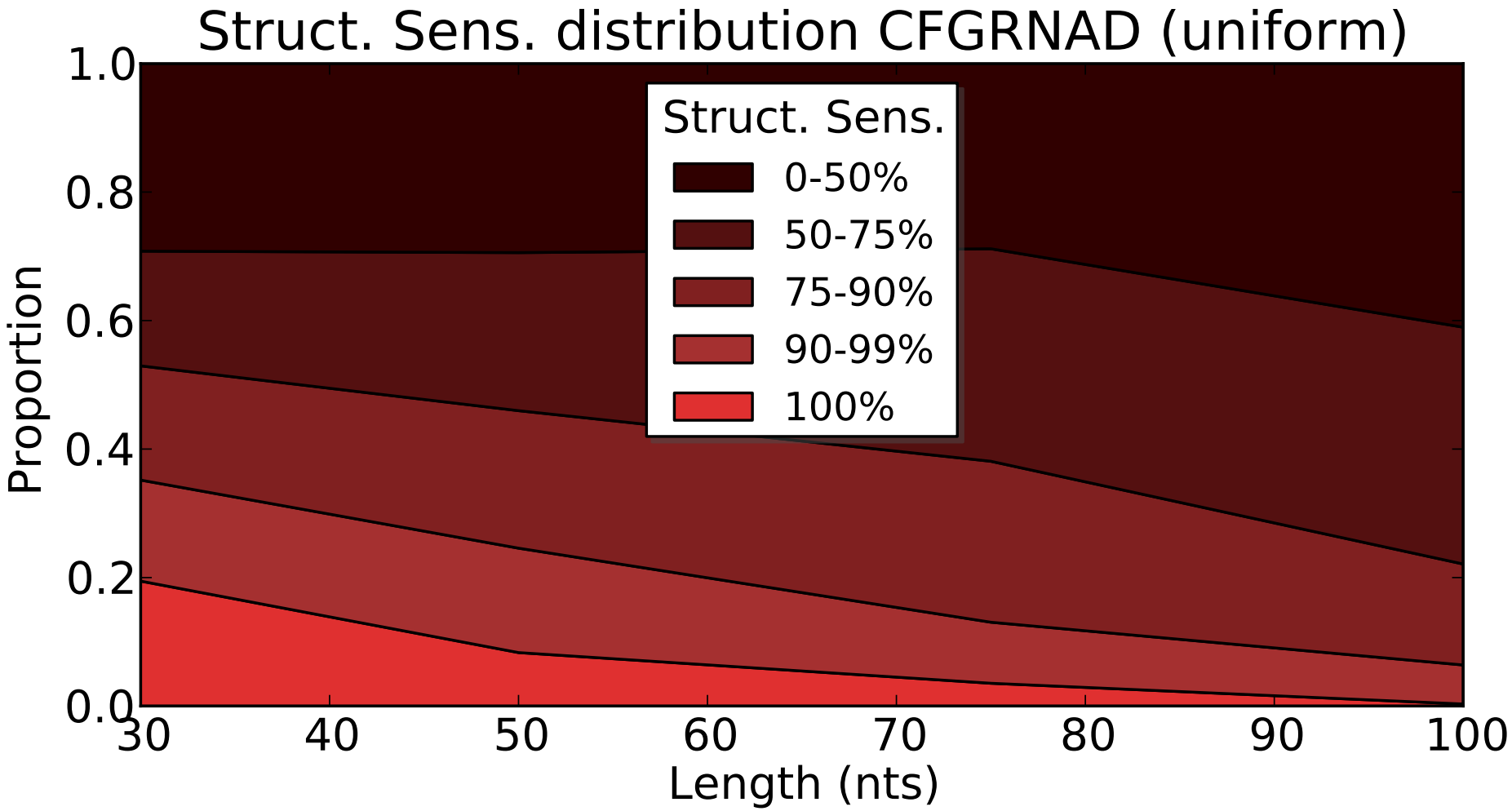
Time benchmark



- Random **realistic** secondary structures **[Denise *et al*, TCS 10]**
- Mandatory motifs $\mathcal{M} = \{\text{CGU}\}$
- Forbidden motifs $\mathcal{F} = \{\text{AAU}, \text{CGC}, \text{UGC}\}$

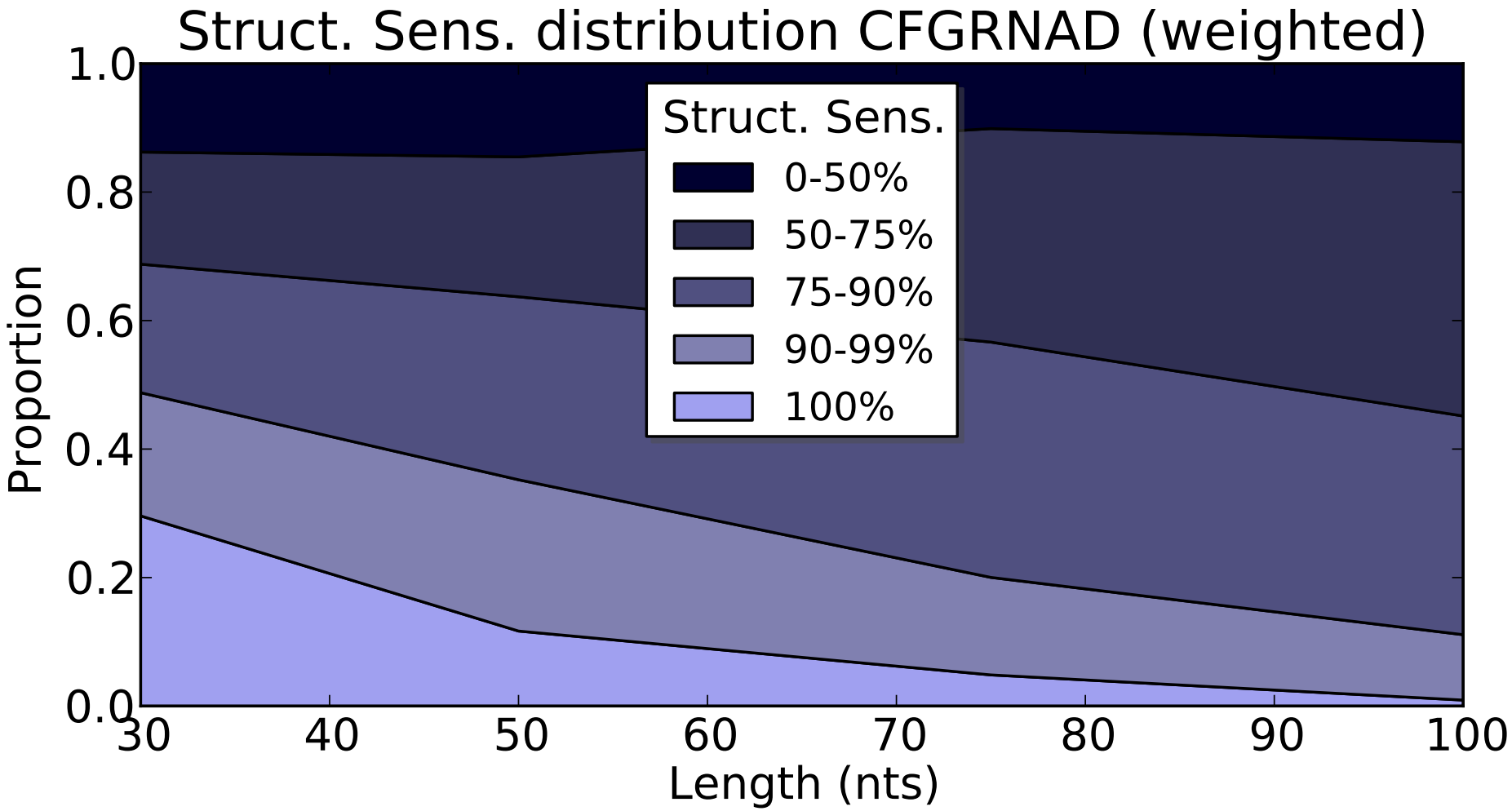
CFGRNAD vs NUPACK: *k* design

Benchmark: Random target structure + Design *k* sequences
Refold design + Structural Sensitivity using RNAFold (Turner 2001)



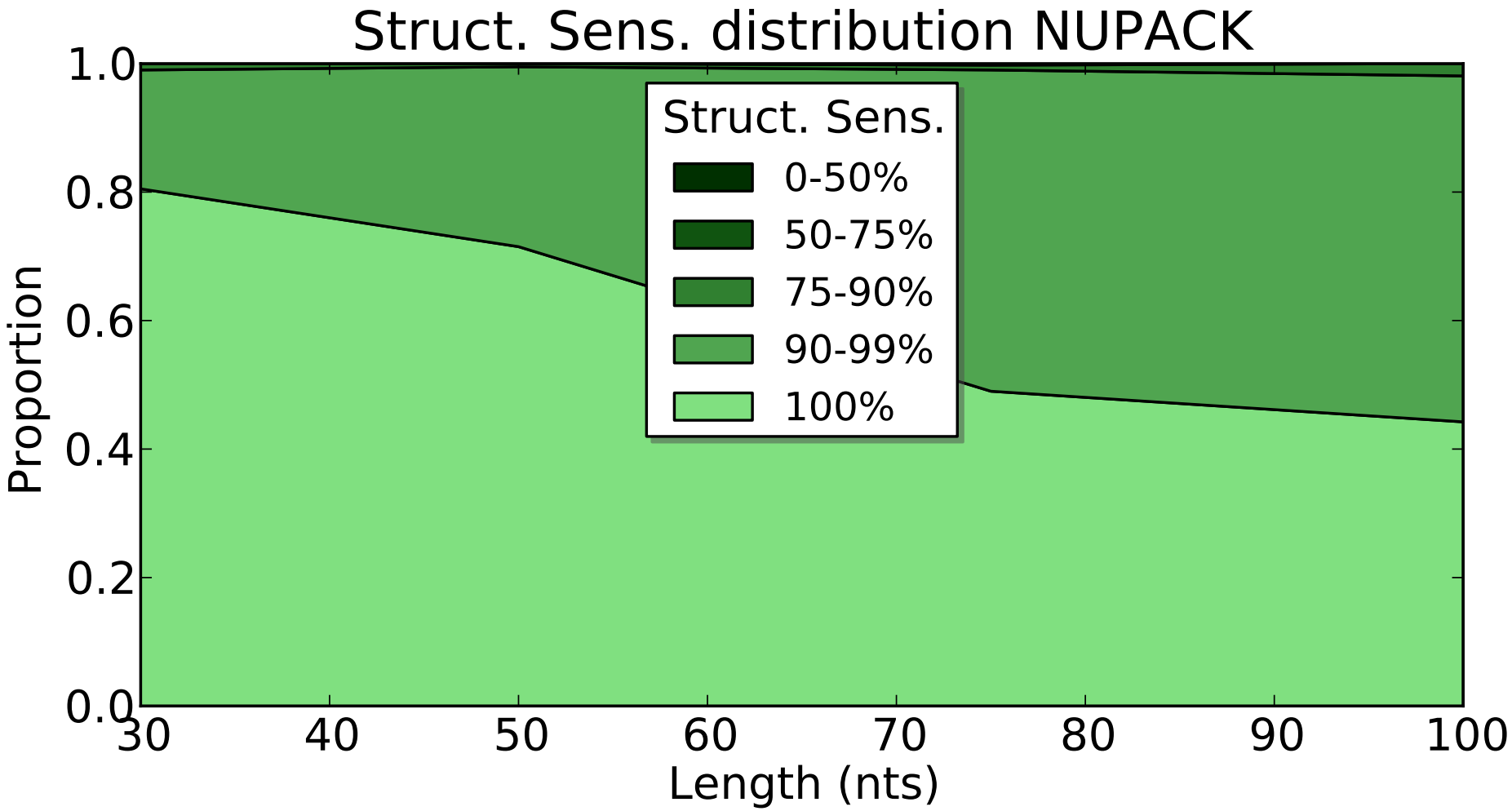
CFGRNAD vs NUPACK: *k* design

Benchmark: Random target structure + Design *k* sequences
Refold design + Structural Sensitivity using RNAFold (Turner 2001)



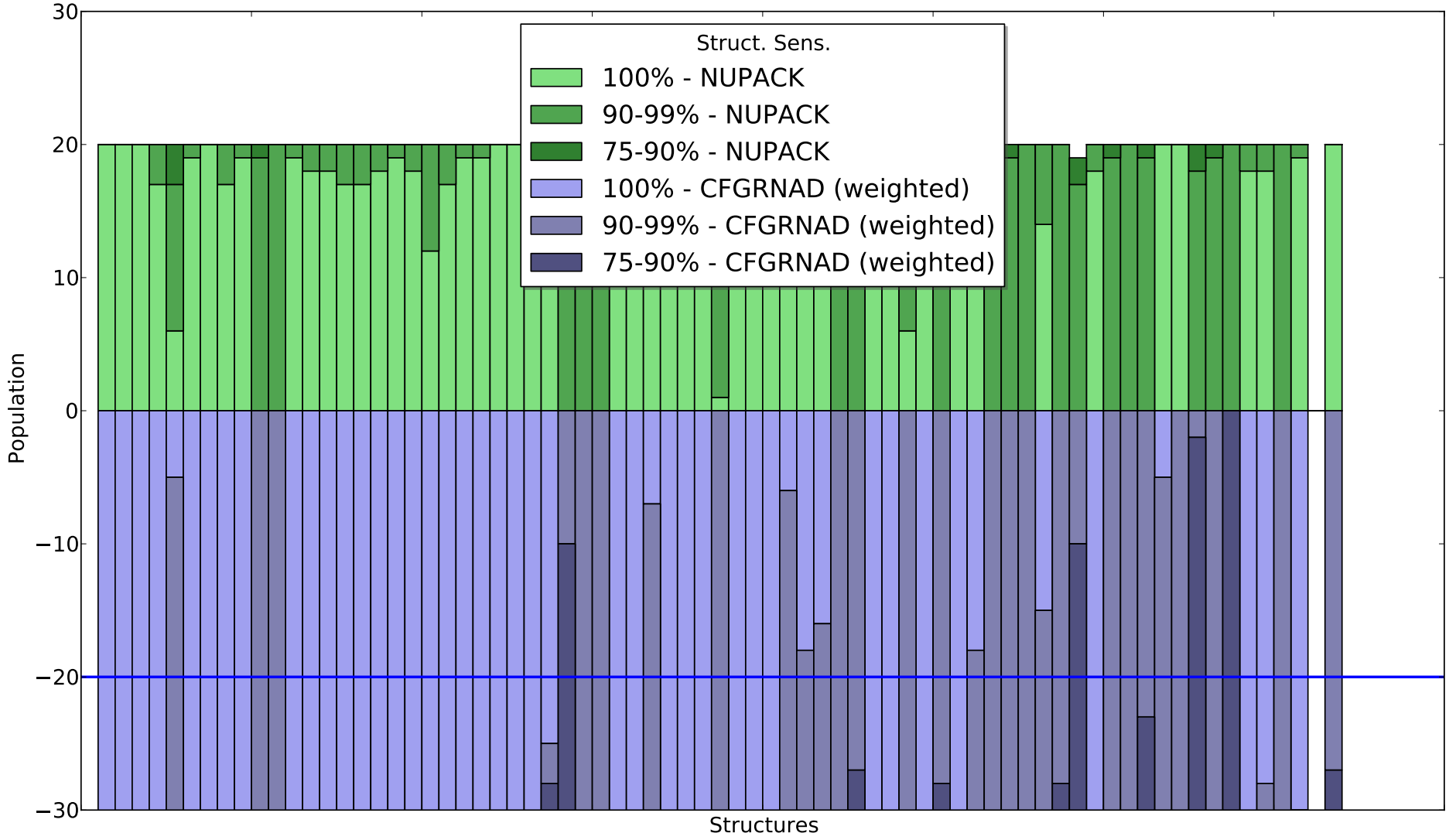
CFGRNAD vs NUPACK: k design

Benchmark: Random target structure + Design k sequences
Refold design + Structural Sensitivity using RNAFold (Turner 2001)



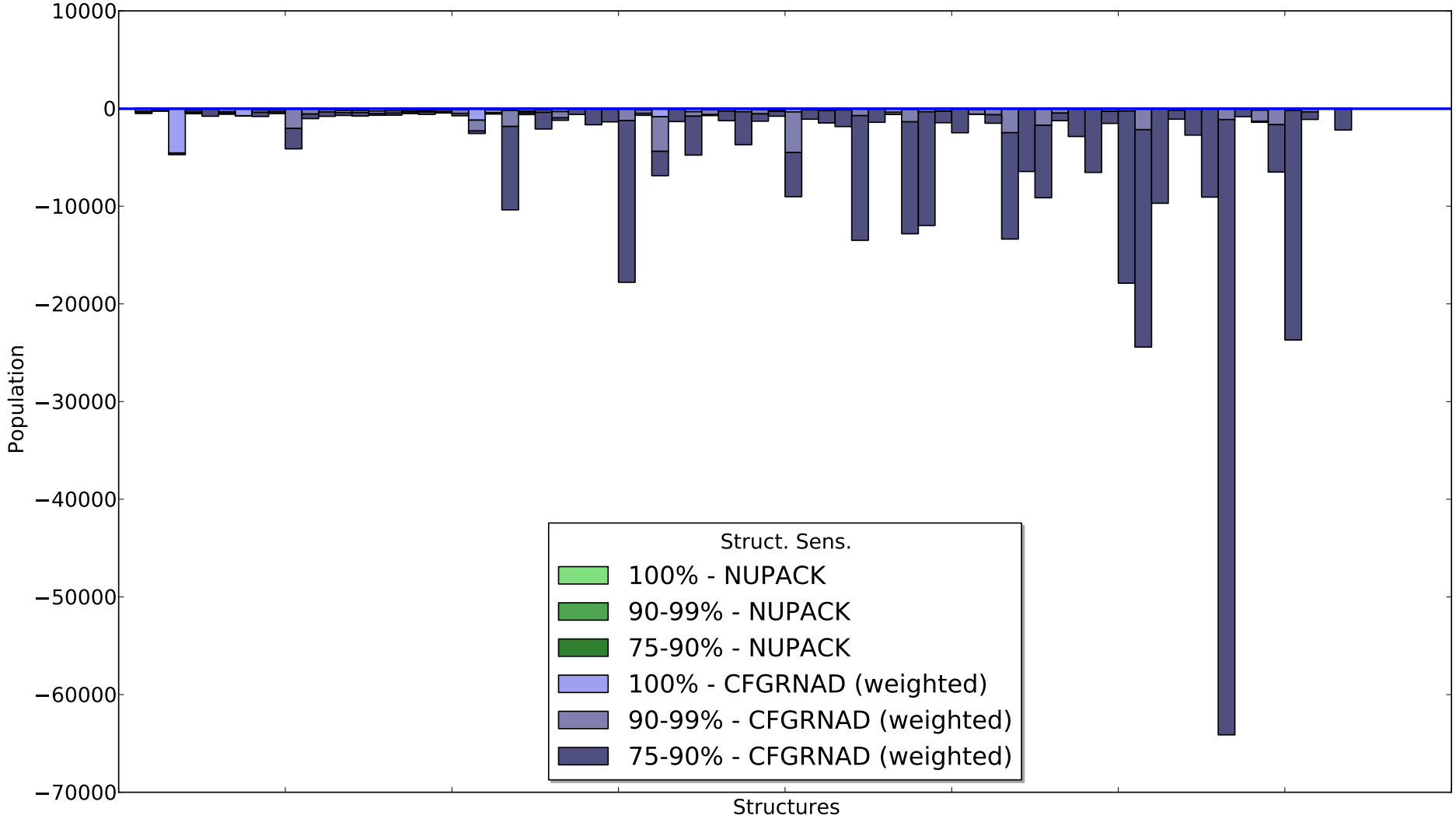
CFGRNAD vs NUPACK: Same time consumption

For RNAs up to 100 nts, #perfect/correct/decent designs are **favorable to CFGRNAD** if given **same amount of time** as NUPACK (inc. refolding)



CFGRNAD vs NUPACK: Same time consumption

For RNAs up to 100 nts, #perfect/correct/decent designs are **favorable to CFGRNAD** if given **same amount of time** as NUPACK (inc. refolding)



Conclusion/Perspectives

Done:

- Linear time algorithm for positive design under constraints
- Practical runtime complexity
- Standalone Python implementation + Webserver

Conclusion/Perspectives

Done:

- Linear time algorithm for positive design under constraints
- Practical runtime complexity
- Standalone Python implementation + Webserver

To Do*s*:

- Combined with *soft constraints* (Boltzmann sampling) for compositional control
- Full Turner energy model
- Investigation of constrained *glocal* approach
- Better products for grammars/automata

Acknowledgments

- **Coauthors**
  - Yu Zhou (UCSD)
  - Stéphane Vialette (LIGM)
  - Jérôme Waldispühl (McGill)
  - Yi Zhang (Wuhan, China)
  - Alain Denise (LRI/IGM)
- James Regis (Polytechnique) for technical help
- Vlad Reinharz (McGill)

You for your attention!

Acknowledgments

- **Coauthors**
  - Yu Zhou (UCSD)
  - Stéphane Vialette (LIGM)
  - Jérôme Waldispühl (McGill)
  - Yi Zhang (Wuhan, China)
  - Alain Denise (LRI/IGM)
- James Regis (Polytechnique) for technical help
- Vlad Reinharz (McGill)

You for your attention!

Combining grammar and automaton

Ideas:

- **Simulate run** in automaton while generating word in grammar
- Each new non-terminal ***commits*** to ***connecting*** two states
- Terminal rules **check** if commitment is fulfilled
- Some non-terminals may become **unproductive** $\rightarrow$ clean-up

$$
\begin{array}{lcl}
V \rightarrow t\, V' & \Longrightarrow & V_{q \rightarrow q'} \rightarrow t\, V'_{\delta(q,t) \rightarrow q'} \\
V \rightarrow t\, V'\, t' & \Longrightarrow & V_{q \rightarrow q'} \rightarrow t\, V'_{\delta(q,t) \rightarrow q''}\, t' \text{ s.t. } \delta(q'', t') = q' \\
V \rightarrow t\, V'\, t'\, V'' & \Longrightarrow & V_{q \rightarrow q'} \rightarrow t\, V'_{\delta(q,t) \rightarrow q''}\, t'\, V''_{\delta(q'',t') \rightarrow q'} \\
V \rightarrow t & \Longrightarrow & \left\{ \begin{array}{ll} V_{q \rightarrow q'} \rightarrow t & \text{if } \delta(q,t) = q' \\ \varnothing & \text{otherwise} \end{array} \right.
\end{array}
$$