Fixed-parameter tractable sampling for RNA design with multiple target structures

Stefan Hammer · Yann Ponty · Wei Wang · Sebastian Will

University of Leipzig · École Polytechnique · University of Vienna

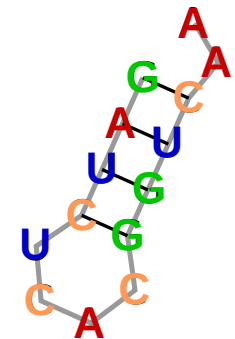
RECOMB 2018 in Paris

RNA Design

GAUCUCACGGUCAA

Structure prediction

Complementarity



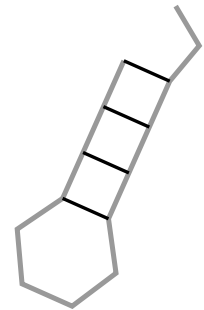
((((....))))..

RNA Design

RNA Design

GAUCUCACGGUCAA

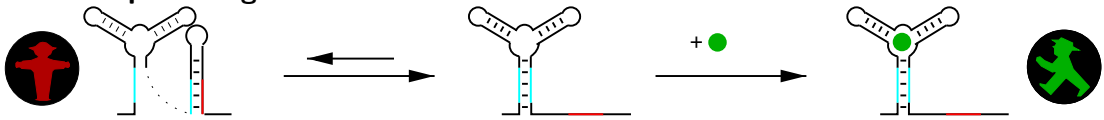
A—U
G—C
Complementarity



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

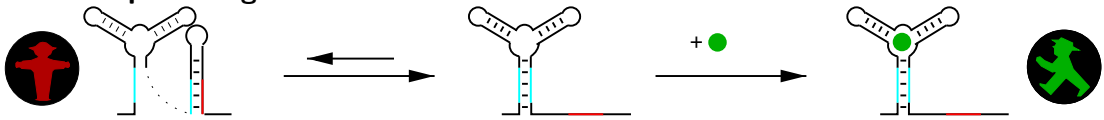
Multi-target design of RNA sequences

For example: design riboswitches for translational control

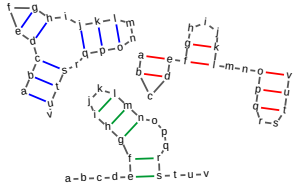


Multi-target design of RNA sequences

For example: design riboswitches for translational control



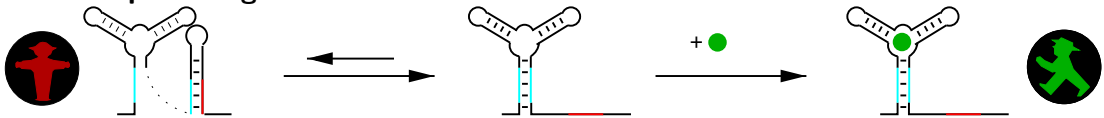
Multiple structures (=*multiple design targets*)



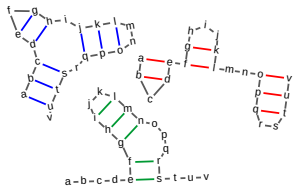

```
abcdefghijklmnopqrstuv
(((((.)).(((..))).))).
((.))((...))..(((..)))
....(((((..)))...)).
```

Multi-target design of RNA sequences

For example: design riboswitches for translational control



Multiple structures (=*multiple design targets*)



```
abcdefghijklmnopqrstuv
(((((.)).(((..))).))).
((.))((...))..(((..)))
....(((((..)))...)).
```

Task: generate seq's with *specific* properties

- low/specific energy for multiple structures
- specific GC content
- specific energy differences
- specific sequence/structure motifs

Approach:
***defined* sampling**

Uniform sampling for multiple structures

	1	2	3	4	5
S1	(	.	.	)	.
S2	.	(	(	)	)
S3	(	(	.	)	)
	A	**A**	**A**	**U**	**U**

- Complementarity

Uniform sampling for multiple structures

	1	2	3	4	5
S1	(	.	.	)	.
S2	.	(	(	)	)
S3	(	(	.	)	)
	A	**A**	**A**	**U**	**U**
	A	**A**	**G**	**U**	**U**
	A	**G**	**A**	**U**	**U**
	A	**G**	**G**	**U**	**U**
	G	**A**	**A**	**U**	**C**
	G	**A**	**A**	**U**	**U**
	G	**A**	**G**	**U**	**C**
	G	**A**	**G**	**U**	**U**
	G	**G**	**A**	**U**	**C**
	G	**G**	**A**	**U**	**U**
	G	**G**	**G**	**C**	**C**
	G	**G**	**G**	**C**	**U**
	G	**G**	**G**	**U**	**C**
	G	**G**	**G**	**U**	**U**
	:				

- Complementarity 

- For uniform: choose first position
  A : **C** : **G** : **U** $= 4 : 4 : 10 : 10$
  Then, e.g. after **G**, choose second **A** : **G** $= 4 : 6, \ldots$
- $\rightarrow$ **counting**

Uniform sampling for multiple structures

	1	2	3	4	5
S1	(	.	.	)	.
S2	.	(	(	)	)
S3	(	(	.	)	)
	A	**A**	**A**	**U**	**U**
	A	**A**	**G**	**U**	**U**
	A	**G**	**A**	**U**	**U**
	A	**G**	**G**	**U**	**U**
	G	**A**	**A**	**U**	**C**
	G	**A**	**A**	**U**	**U**
	G	**A**	**G**	**U**	**C**
	G	**A**	**G**	**U**	**U**
	G	**G**	**A**	**U**	**C**
	G	**G**	**A**	**U**	**U**
	G	**G**	**G**	**C**	**C**
	G	**G**	**G**	**C**	**U**
	G	**G**	**G**	**U**	**C**
	G	**G**	**G**	**U**	**U**
	:				

- Complementarity 

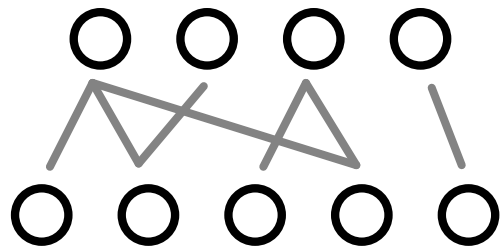
- For uniform: choose first position
  A : **C** : **G** : **U** $= 4 : 4 : 10 : 10$
  Then, e.g. after **G**, choose second **A** : **G** $= 4 : 6, \ldots$
- $\rightarrow$ **counting**

- **Theorem: Counting of sequences for multiple targets is #P-hard.**

Counting is #P-hard

Proof (sketch):

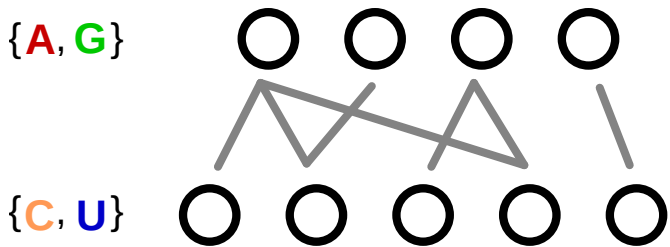
- Counting bipartite independent sets is #P-hard.
- Sequence counting is *equivalent* to counting **independent sets**.



Counting is #P-hard

Proof (sketch):

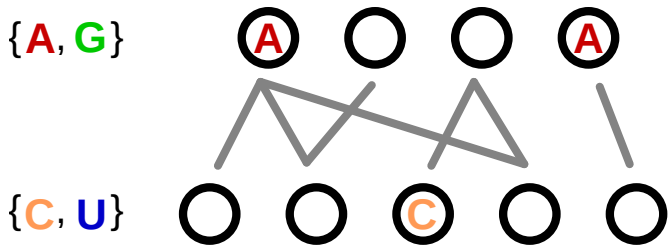
- Counting bipartite independent sets is #P-hard.
- Sequence counting is *equivalent* to counting **independent sets**.



Counting is #P-hard

Proof (sketch):

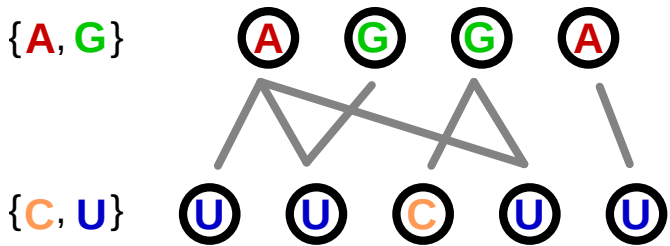
- Counting bipartite independent sets is #P-hard.
- Sequence counting is *equivalent* to counting **independent sets**.



Counting is #P-hard

Proof (sketch):

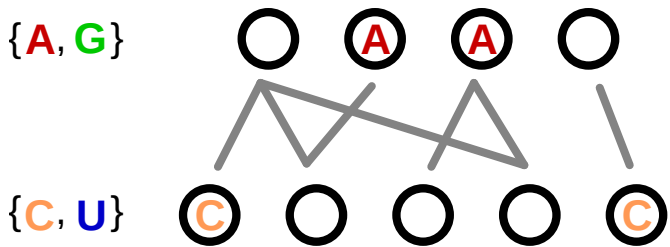
- Counting bipartite independent sets is #P-hard.
- Sequence counting is *equivalent* to counting **independent sets**.



Counting is #P-hard

Proof (sketch):

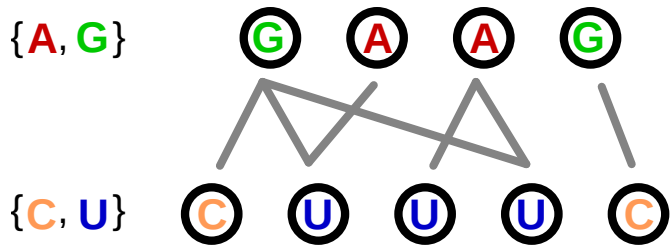
- Counting bipartite independent sets is #P-hard.
- Sequence counting is *equivalent* to counting **independent sets**.



Counting is #P-hard

Proof (sketch):

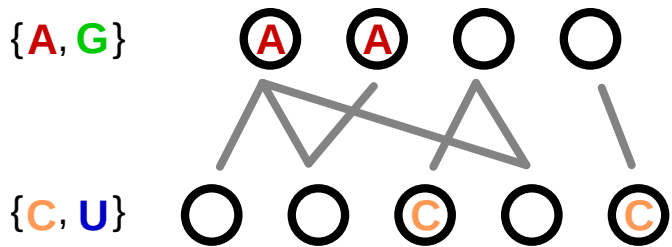
- Counting bipartite independent sets is #P-hard.
- Sequence counting is *equivalent* to counting **independent sets**.



Counting is #P-hard

Proof (sketch):

- Counting bipartite independent sets is #P-hard.
- Sequence counting is *equivalent* to counting **independent sets**.



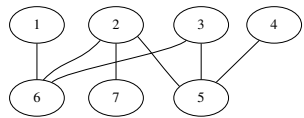
Systematic counting and sampling

Recipe:

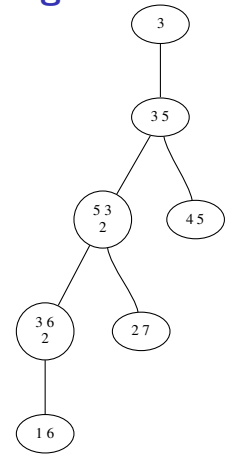
1. Decompose dependency graph
2. Apply **dynamic programming** ↑
3. **Sample** ↓

```
1 2 3 4 5 6 7
( ( . . ) ) .
. ( ( ( ) ) )
. ( ( . ) ) .
```

target structures



dependency graph



tree decomposition

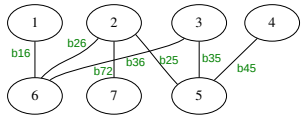
Systematic counting and sampling

Recipe:

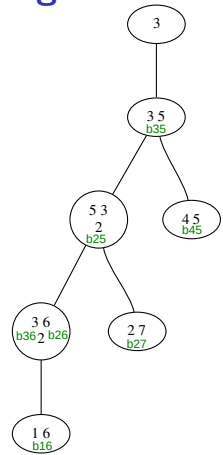
1. Decompose dependency graph
2. Apply **dynamic programming** ↑
3. **Sample** ↓

```
1 2 3 4 5 6 7
( ( . . ) ) .
. ( ( ( ) ) )
. ( ( . ) ) .
```

target structures



dependency graph



tree decomposition

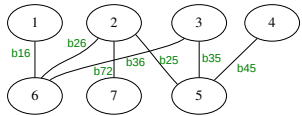
Systematic counting and sampling

Recipe:

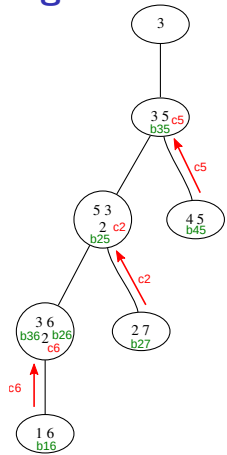
1. Decompose dependency graph
2. Apply **dynamic programming** ↑
3. **Sample** ↓

```
1 2 3 4 5 6 7
( ( . . ) ) .
. ( ( ( ) ) )
. ( ( . ) ) .
```

target structures



dependency graph



tree decomposition

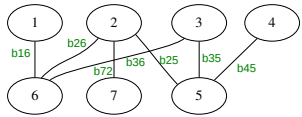
Systematic counting and sampling

Recipe:

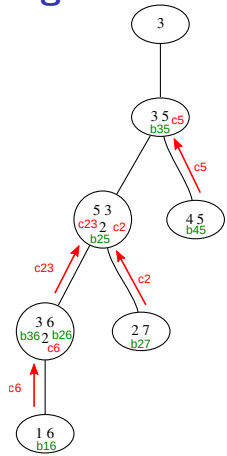
1. Decompose dependency graph
2. Apply **dynamic programming** ↑
3. **Sample** ↓

```
1 2 3 4 5 6 7
( ( . . ) ) .
. ( ( ( ) ) )
. ( ( . ) ) .
```

target structures



dependency graph



tree decomposition

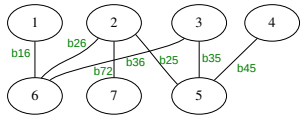
Systematic counting and sampling

Recipe:

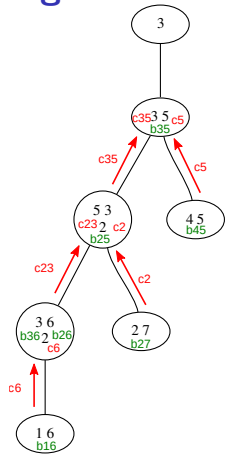
1. Decompose dependency graph
2. Apply **dynamic programming** ↑
3. **Sample** ↓

```
1 2 3 4 5 6 7
( ( . . ) ) .
. ( ( ( ) ) )
. ( ( . ) ) .
```

target structures



dependency graph



tree decomposition

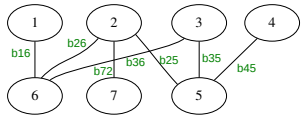
Systematic counting and sampling

Recipe:

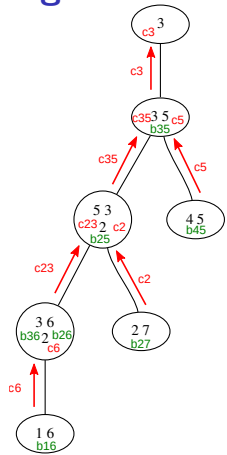
1. Decompose dependency graph
2. Apply **dynamic programming** ↑
3. **Sample** ↓

```
1 2 3 4 5 6 7
( ( . . ) ) .
. ( ( ( ) ) )
. ( ( . ) ) .
```

target structures



dependency graph



tree decomposition

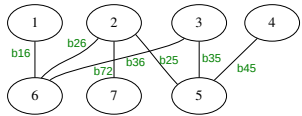
Systematic counting and sampling

Recipe:

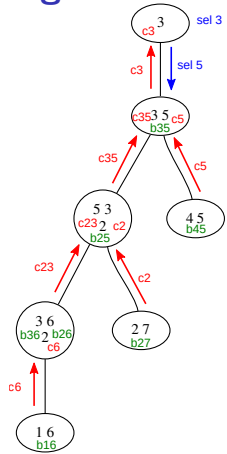
1. Decompose dependency graph
2. Apply **dynamic programming** ↑
3. **Sample** ↓

```
1 2 3 4 5 6 7
( ( . . ) ) .
. ( ( ( ) ) )
. ( ( . ) ) .
```

target structures



dependency graph



tree decomposition

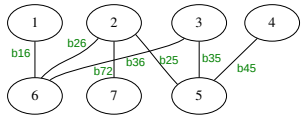
Systematic counting and sampling

Recipe:

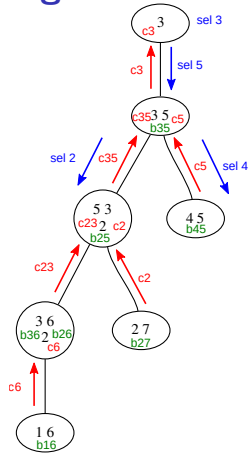
1. Decompose dependency graph
2. Apply **dynamic programming** ↑
3. **Sample** ↓

```
1 2 3 4 5 6 7
( ( . . ) ) .
. ( ( ( ) ) )
. ( ( . ) ) .
```

target structures



dependency graph



tree decomposition

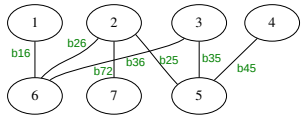
Systematic counting and sampling

Recipe:

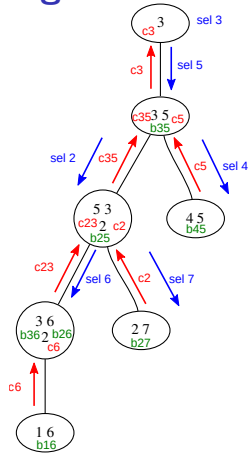
1. Decompose dependency graph
2. Apply **dynamic programming** ↑
3. **Sample** ↓

```
1 2 3 4 5 6 7
( ( . . ) ) .
. ( ( ( ) ) )
. ( ( . ) ) .
```

target structures



dependency graph



tree decomposition

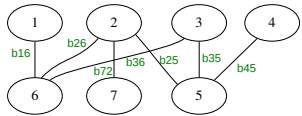
Systematic counting and sampling

Recipe:

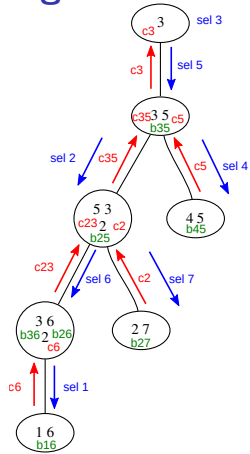
1. Decompose dependency graph
2. Apply **dynamic programming** ↑
3. **Sample** ↓

```
1 2 3 4 5 6 7
( ( . . ) ) .
. ( ( ( ) ) )
. ( ( . ) ) .
```

target structures



dependency graph



tree decomposition

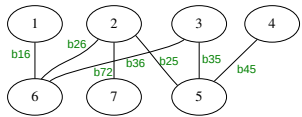
Systematic counting and sampling

Recipe:

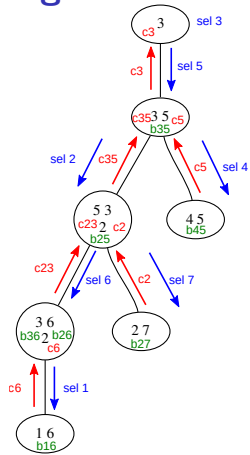
1. Decompose dependency graph
2. Apply **dynamic programming** ↑
3. **Sample** ↓

```
1 2 3 4 5 6 7
( ( . . ) ) .
. ( ( ( ) ) )
. ( ( . ) ) .
```

target structures



dependency graph



tree decomposition

Theorem: Counting and sampling is efficient for fixed tree width

$$\mathcal{O}(n\,k\,\mathbf{4^w} + t\,n\,k)$$

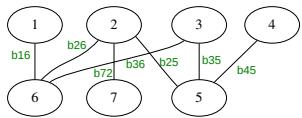
Systematic counting and sampling

Recipe:

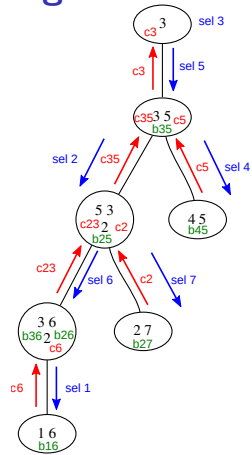
1. Decompose dependency graph
2. Apply **dynamic programming** ↑
3. **Sample** ↓

```
1 2 3 4 5 6 7
( ( . . ) ) .
. ( ( ( ) ) )
. ( ( . ) ) .
```

target structures



dependency graph



tree decomposition

Theorem: Counting and sampling is efficient for fixed tree width

$$\mathcal{O}(n\,k\,\mathbf{4^w} + t\,n\,k) \longrightarrow \mathcal{O}(n\,k\,\mathbf{2^{w+c}} + t\,n\,k)$$

From uniform to Boltzmann sampling

uniform sampling $\longleftarrow$ **counts**
Boltzmann sampling $\longleftarrow$ **partition functions**

Boltzmann sampling: $\mathrm{P}(S) \propto \exp(-\beta E(S))$.

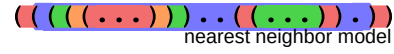
From uniform to Boltzmann sampling

uniform sampling $\longleftarrow$ **counts**
Boltzmann sampling $\longleftarrow$ **partition functions**

$$\textit{Boltzmann sampling}: \quad \mathrm{P}(S) \;\propto\; \exp(-\beta E(S)).$$

Energy $E(S) := \sum$ weighted energies of single structures

- **energy models**
  - **Base pair model**
    "like counting"
  - **Nearest neighbor model (Turner)**
    requires multi-ary dependencies: **constraint framework***
  - **Stacking model**
    "in-between", scores stacks

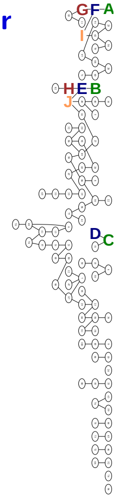


*Constraint networks / cluster tree elimination [Rina Dechter]

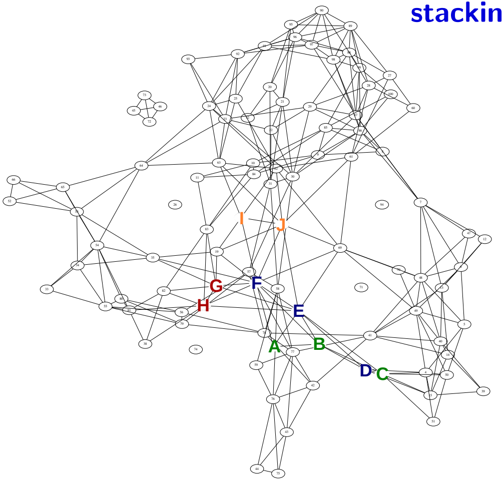
Dependency graphs

```
((((.((....)).)))).((.(((.((((.....(((..((((((.((..(((.(.....).)))..)).)).)))).)))..)))).))).))....
..(((((.....(((.(((((((.....))))..))).))).....)))))..((((((((((...))).)....))))))...((((((....))))))
......(((((.....(((...(((.((.((.(((....((......))...))).)).)))))..)))).............))))).((((...)))).
ABC                         DEF                        GH                               IJ
```

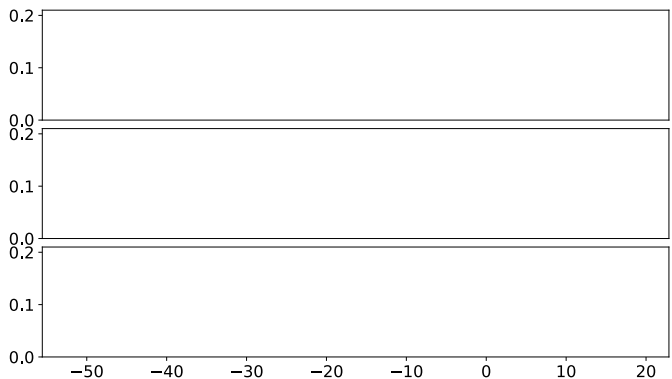
base pair



stacking



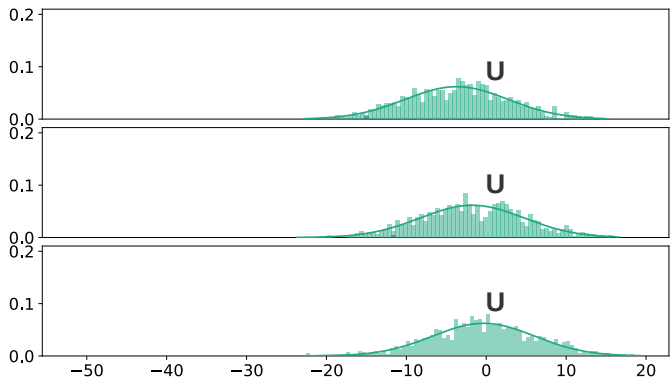
Targeting specific properties: multi-dimensional Boltzmann sampling



Weight and combine single structure energies and features

Learn weights (adaptively) → target specific energies and GC content

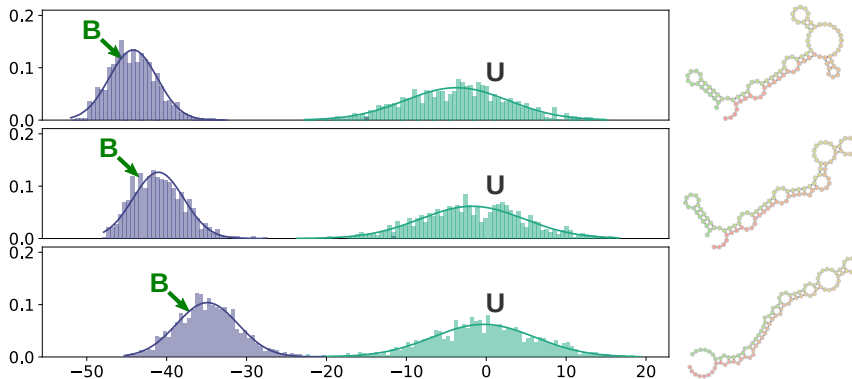
Targeting specific properties: multi-dimensional Boltzmann sampling



Weight and combine single structure energies and features

Learn weights (adaptively) → target specific energies and GC content

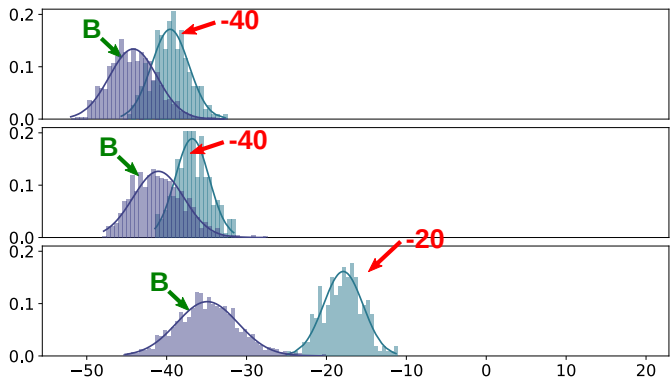
Targeting specific properties: multi-dimensional Boltzmann sampling



Weight and combine single structure energies and features

Learn weights (adaptively) → target specific energies and GC content

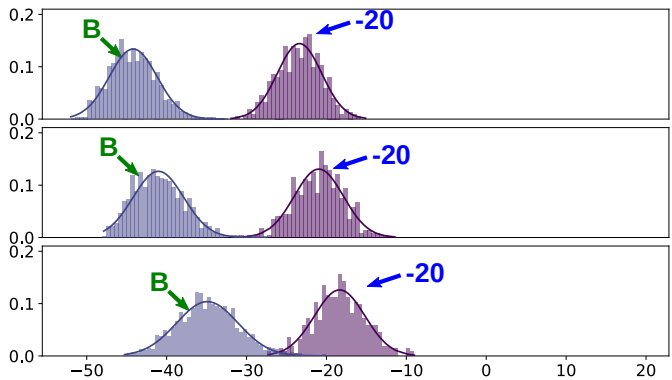
Targeting specific properties: multi-dimensional Boltzmann sampling



Weight and combine single structure energies and features

Learn weights (adaptively) → target specific energies and GC content

Targeting specific properties: multi-dimensional Boltzmann sampling



Weight and combine single structure energies and features

Learn weights (adaptively) → target specific energies and GC content

Boltzmann vs. uniform sampling for multi-target RNA design

	Dataset	RedPrint	Uniform	Improvement
Seeds	2str	21.67 ($\pm$4.38)	37.74 ($\pm$6.45)	73%
	3str	18.09 ($\pm$3.98)	30.49 ($\pm$5.41)	71%
	4str	19.94 ($\pm$3.84)	32.29 ($\pm$5.24)	63%
Optimized	2str	5.84 ($\pm$1.31)	7.95 ($\pm$1.76)	28%
	3str	5.08 ($\pm$1.10)	7.04 ($\pm$1.52)	31%
	4str	8.77($\pm$1.48)	13.13 ($\pm$2.13)	37%

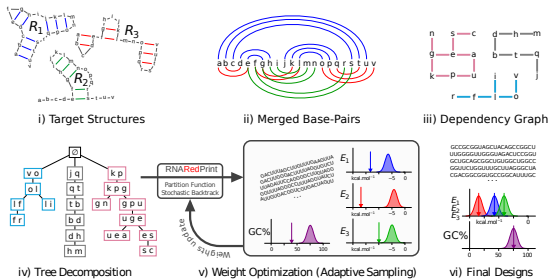
Multi-target design objective[*Blueprint*] on the **Modena benchmark**

`https://github.com/yannponty/RNARedPrint`

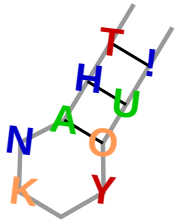
[Modena] Taneda. *BMC Bioinformatics*, 2015.
[Blueprint] Hammer et al. *Bioinformatics*, 2017.

Summary

- FPT Boltzmann sampling for multi-target RNA design (counting is #P-hard)
- Targets specific properties
- Versatile framework w/ multi-ary constraints
- Supports complex RNA design scenarios and various RNA energy models (NN, PKs)
- Perspectives: towards FPT negative design; apply to Riboswitch design



(workflow for the base pair energy model; our approach supports complex models and scenarios by n-ary constraints)



Co-authors

Leipzig:  Stefan Hammer

Paris:  Yann Ponty  Wei Wang

Team TBI at universität wien

Funding

ANR Federal Ministry of Education and Research FWF SEVENTH FRAMEWORK PROGRAMME