Algorithms for RNA design

Sebastian Will

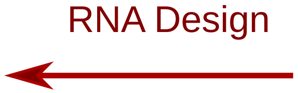
École Polytechnique - Institute Polytechnique de Paris

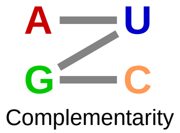
AlgoSB 2025

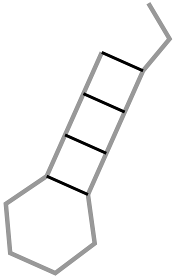
[Thanks to Hua-Ting Yao for a number of slides]

RNA Design

GAUCUCACGGUCAA



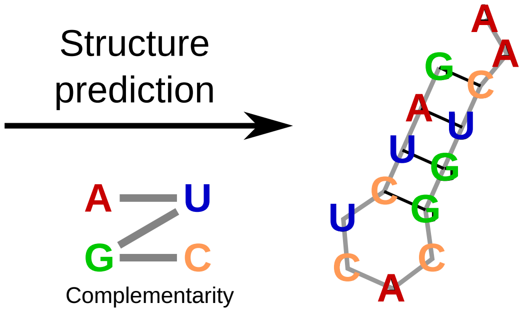




((((....))))..

RNA Design

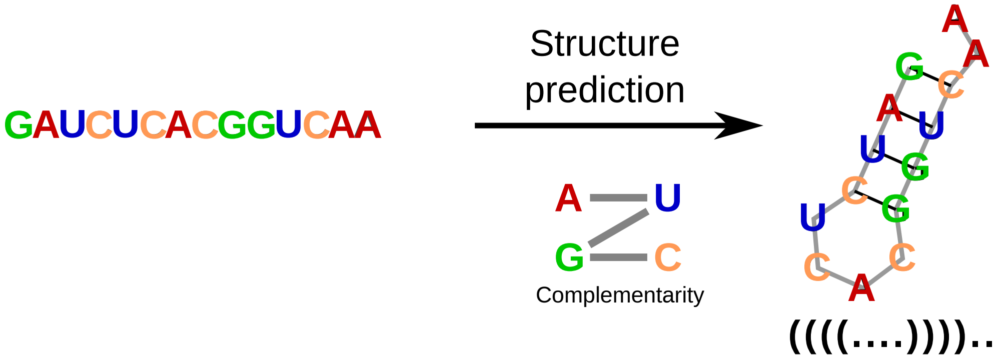
GAUCUCACGGUCAA



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

Structure design as “inverse folding”

RNA Design



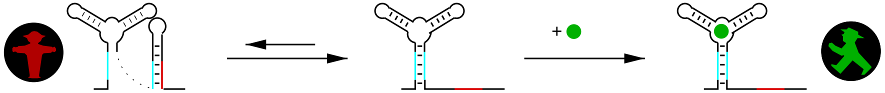
Structure design as "inverse folding"

More generally: Generating RNA sequences with desired functions

→ biotechnology, RNA-based therapies, mRNA vaccines...

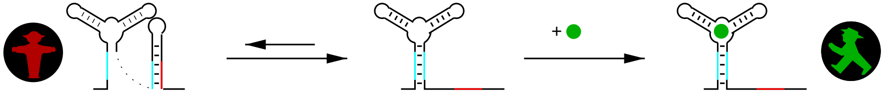
Complex RNA design

Design riboswitches for gene control

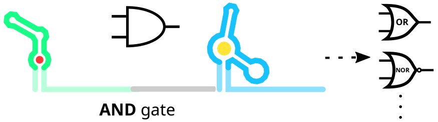


Complex RNA design

Design riboswitches for gene control

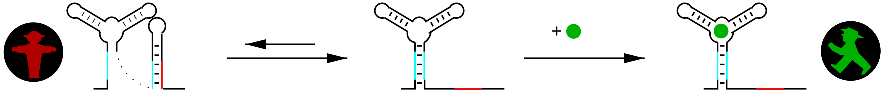


Complex constructs: **AND-Riboswitch** (with G. Domin *et al.*, 2017)

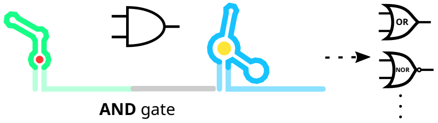


Complex RNA design

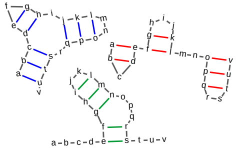
Design riboswitches for gene control



Complex constructs: **AND-Riboswitch** (with G. Domin *et al.*, 2017)

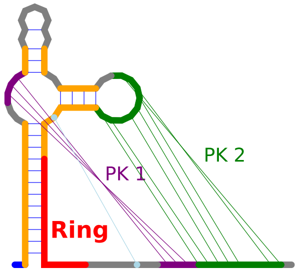


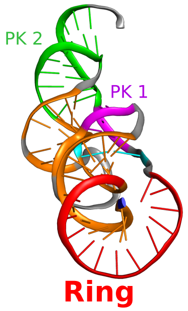
Challenge: **Design for multiple target structures**



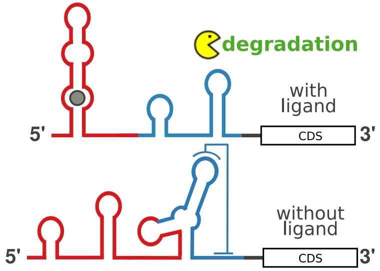
```
abcdefghijklmnopqrstuv
(((((.)).(((..))).))).
((.))((...))..(((..)))
....(((((..)))...))...
```

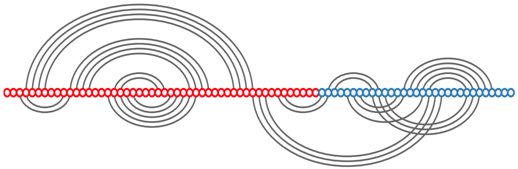
Design of xrRNA riboswitch





xrRNA

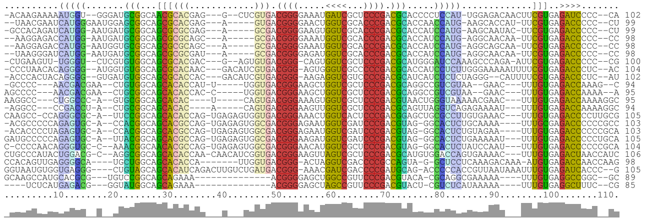




[Leonhard Sidl, MT Wolfinger, HT Yao...]

Design of SAM-I aptamer

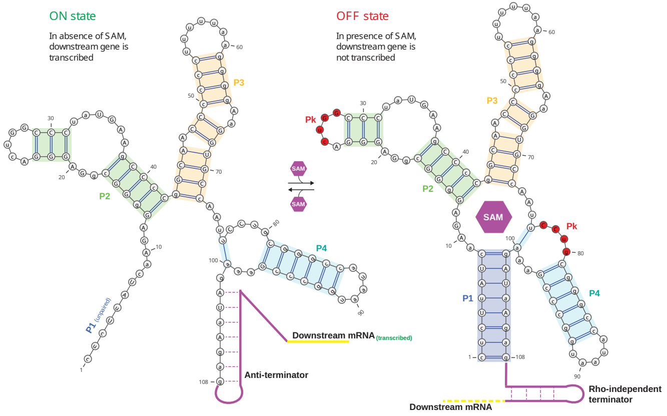
- Aim for similarity with MSA
- Learn generative model from MSA (RBM)
- Compatibility with target structure (with PKs)
- Avoid off-targets



```
..........(((((.......(((...[[[(((............))).((((.....<<<<...)))).))).....))))).............]]]..>>>>.......
-ACAAGAAAAAUGGU--GGGAUGCGGCAACGCACGAG---G--CUCGUGACGGGGAAAUGAUCGCUCCCGACGCACCCCUCCAU-UGGAGACAACUUCGUGAGAUCCCC--CA 102
--UAACGAAUCAUGGGAAUGGAGCGGCAGCGCACGAG---A-----GUGACGGGGAACUGGUCGCACCCGACGCACCAACCAUG-AAGCACCAU-UUCGUGAGACCCCC--CU 99
-GCCACAGAUCAUGG-AAUGAUGCGGCAGCGCGCGAG---A-----GCGACGGGGAAGUGGUCGCACCCGACGCACCAUCCAUG-AAGCAAUAC-UUCGUGAGACCCCC--CU 99
--AAGGAGACCAUGG-AAUGAUGCGGCAGCGCGCAGC---A-----GCGACGGGGAAAUGGUCGCACCCGACGCACCAUCCAUG-AGGCAACAA-UUCGUGAGACCCCC--CC 98
--AAGGAGACCAUGG-AAUGGUGCGGCAGCGCGCAGC---A-----GCGACGGGGAAAUGGUCGCACCCGACGCACCAUCCAUG-AGGCAGCAA-UUCGUGAGACCCCC--CC 98
--UAAGGGAUCAUGG-AAUGAUGCGGCAGCGCGCGAU---A-----GCGACGGGGAGAUGGUCGCACCCGACGCACCAUCCAUG-AGGCAACAA-UUCGUGAGACCCCC--CC 98
-CUGAAGUU-UGGGU--CUCGUGUGGCAGCGCACGAC---G--AGUGUGACGGG-CAGUGGUCGCUCCCGACGCAUGGGAUCCAAAGCCCAGA-AUUCGUGAGACCCCC--CG 100
-CCCUAACACAGGGG--AUGGUGUGGCAGCGCACAAC---GACAUCGUGACGGG-AGUGGGUCGCCCCCGACGCACCAUCCUCUUGGGAAAAAUUUUCGUGAGACCCUC--AC 104
-ACCCACUACAGGGG--GUGAUGUGGCAGCGCACCAC---GACAUCGUGACGGG-AAGAGGUCGUCCCCGACGCAUCAUCUCUCUAGGG--CAUUUUCGUGAGACCCUC--AU 102
-GCCCC---AACGACGAA--CUGUGGCAGCACACCAU-U-----UGGUGACGGGAAGCUGGUCGCUCCCGACGCAGGCCGUCGUAA--GAAC---UUUGUGAGACCAAAG--C 94
AGCCCC---AACGACGAA--CUGCGGCAGCACACCAC-C-----UGGUGACGGGAAGCUGGUCGCUCCCGACGCAGGCCGUCGUAA--GAAC---UUUGUGAGACCAAAA--A 95
AAGGCC---CUGGCCC-A--GUGCGGCAGCACAC----U-----CAGUGACGGGAAAGUGGUCGCUCCCGACGUAACUGGGUAAAAACGAAC---UUUGUGAGACCAAAAGGC 95
-AGGCC---CCGACCU-A--CUGCGGCAGCACAC----A-----CAGUGACGGGAAGUUGGUCGCUCCCGACGCAGUUAGGUCAGAGAAAAU---UUUGUGAGACCAAAAGGC 94
CAAGCC-CCAGGGCGC-A--UUCCGGCAGCACACCAG-UGAGAGUGGUGACGGGAAACUGGUCACUCCCGACGGAGCUGCGCCUUGUGAAAC---UUUGUGAGACCCCUUGCG 105
-ACGCCCCCAGAGUGC-A--CCACGGCAGCACGCCAG-UGAGAGUGGCGACGGGAGAAUGGUCGAUCCCGACGUAG-GGCACUCUGCAAAA----UUUGCGAGACCCCCCGCC 103
-ACACCCCUAGAGUGC-A--CCACGGCAGCACGCCAG-UGAGAGUGGCGACGGGAGAAUGGUCGAUCCCGACGUAG-GGCACUCUGUAGAA----UUUGUGAGACCCCCCGCA 103
GAUGCCCCCAGAGUGC-A--UUACGGCAGCACGCCAG-UGAGAGUGGCGACGGGAAGAUGGUCGAUCCCGACGUAG-GGCACUCUGAAAAAU---UUUGUGAGACCCCCUGCA 105
C-CCCCAACAGGGUGC-C--AAACGGCAACACGCCAG-UGAGAGUGGCGACGGGAACAUGGUCGCUCCCGACGUAG-GGCACUCUAUCCAAU---UUUGUGAGACCCCCCGCA 104
CUGCCCAUACUGGACG-C--AGGCGGCAGCACACCAA-CAACAUCGGUGACGGGAAGUUAGUCGCUCCCGACGCAUGUGGACCAGUGAAAAC---UUUGUGAGACUAACCAUC 106
CCACAGUUGAGGGGCA----UGCUGGCAGCACACCA-------UUGGUGACGGG-ACUAGGUCGACCCCGCCAGUA-G-GCUCCUCAAAGACAAA-AUGUGAGACCAACCAAG 98
GGUAAUGUGGUGAGGG----CUGUAGCAGCACAUCAGACUUGUCUGAUGACGGG-AAACGAUCGACCCCGAUGCAG-ACCCCCACCGUUAAUAAAUUUGUGAGAUCACCC--G 105
GCAAGCCAUGCACGCG---UGUCCGGCAGCAGAAA--------------ACGGGGAGCUGGCCGUUCCCGACGUACA-CGCAGGCGAAAAA----UUUGUGAGGCCGGC--GC 89
----UCUCAUGAGACG---GGUAUGGCAGCAGAAA--------------ACGGGGAGCUAGCCGUUCCCGACGUACU-CGUCUCAUAAAAA----UUUGUGAGGCUUUC--CG 85
.........10........20........30........40........50........60........70........80........90........100.......110.
```

[Jorge Fernandez-de-Cossio-Diaz, 2024]

Design of SAM-I riboswitch


ON state
In absence of SAM, downstream gene is transcribed
P1 (unpaired)
P2
P3
P4
Downstream mRNA (transcribed)
Anti-terminator
SAM
SAM
OFF state
In presence of SAM, downstream gene is not transcribed
Pk
P1
P2
P3
P4
Pk
SAM
Rho-independent terminator
Downstream mRNA

[Jorge Fernandez-de-Cossio-Diaz, 2024]

Generation (Sampling):

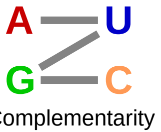
- targets: compatibility, energy
- sequence similiarity

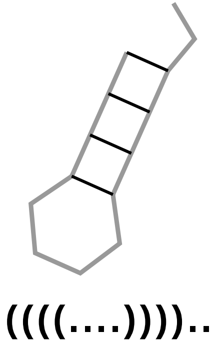
Refinement (Stochastic optimization):

- avoid off-targets
- relative stability bound/unbound

RNA Design

GAUCUCACGGUCAA





RNA structure design: positive and negative

Positive design: Target a structure

$\rightarrow$ optimize affinity to target structures t

find sequence σ
with $E(\sigma, t) = \min_{\sigma'}(\sigma', t)$

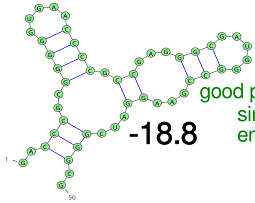
extensions: multiple targets, properties, ...

Negative design: Avoid all off-targets

$\rightarrow$ specificity for targets

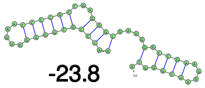
find design σ, s.t.
$E(\sigma, t) = \min_{t'} E(\sigma, t') \quad =: MFE(\sigma)$

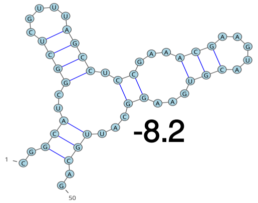
refined objectives: probability, ensemble, ...



good positive design
since all/most other sequences energetically worse

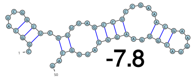
but no negative design, since

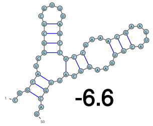




negative design
(by MFE criterion)

since all other structures energetically worse, e.g.





....

Hello world of positive design!

(single target structure, base pair energy)

IN: target structure t of length n

"base pair energy" $E(\sigma, t) = \sum_{(i,j)\in t} e_{bp}(\sigma_i, \sigma_j) + \sum_{i \text{ unpaired in } t} e_u(\sigma_i)$

Task: sample from the Boltzmann distribution of sequences; i.e. sample σ with probability

$$\Pr[\sigma] \sim \exp(-\beta E(\sigma, t)) \qquad \text{for some inverse temperature } \beta$$

Hello world of positive design!
(single target structure, base pair energy)

IN: target structure t of length n
"base pair energy" $E(\sigma, t) = \sum_{(i,j)\in t} e_{bp}(\sigma_i, \sigma_j) + \sum_{i \text{ unpaired in } t} e_u(\sigma_i)$

Task: sample from the Boltzmann distribution of sequences; i.e. sample σ with probability

$$\Pr[\sigma] \sim \exp(-\beta E(\sigma, t)) \qquad \text{for some inverse temperature } \beta$$

Observation: bases and base pairs can be generated *independently* of each other!

Algorithm

1) for each unpaired i: choose σ_i with $Pr[\sigma_i] \sim exp(-\beta e_u(\sigma_i))$
2) for each pair $(i,j) \in t$: choose σ_i and σ_j with $Pr[\sigma_i, \sigma_j] \sim exp(-\beta e_{bp}(\sigma_i, \sigma_j))$

Hello world of positive design!
(single target structure, base pair energy)

IN: target structure t of length n

"base pair energy" $E(\sigma, t) = \sum_{(i,j)\in t} e_{bp}(\sigma_i, \sigma_j) + \sum_{i \text{ unpaired in } t} e_u(\sigma_i)$

Task: sample from the Boltzmann distribution of sequences; i.e. sample σ with probability

$$\Pr[\sigma] \sim \exp(-\beta E(\sigma, t)) \qquad \text{for some inverse temperature } \beta$$

Observation: bases and base pairs can be generated *independently* of each other!

Algorithm

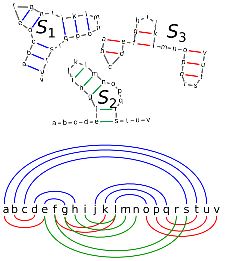
1) for each unpaired i: choose σ_i with $Pr[\sigma_i] \sim exp(-\beta e_u(\sigma_i))$
2) for each pair $(i,j) \in t$: choose σ_i and σ_j with $Pr[\sigma_i, \sigma_j] \sim exp(-\beta e_{bp}(\sigma_i, \sigma_j))$

🤔 What happens for complex energy models or multiple targets? (*dependencies!*)

Infrared

General efficient framework for weighted constraint solving.
$\Rightarrow$ Rapid development of bioinformatics tools (including design)

Input



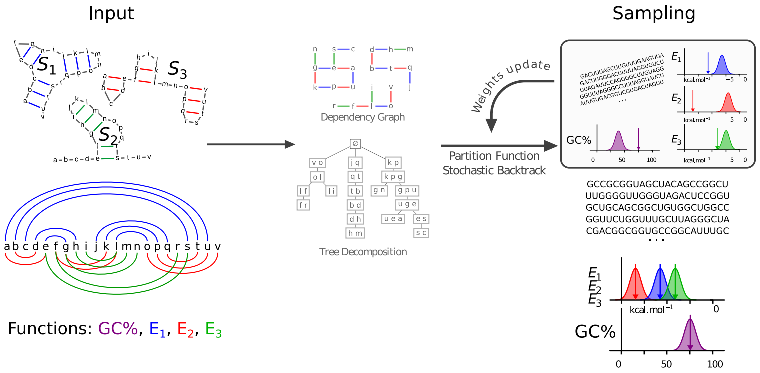
Functions: GC%, E_1, E_2, E_3

Infrared solves a weighted form of constraint problems (CSP $\rightarrow$ Feature Networks)
It allows us to describe ("model") problems; then solves them automatically.

[Hua-Ting Yao et al., 2024]

Infrared

General efficient framework for weighted constraint solving.
$\Rightarrow$ Rapid development of bioinformatics tools (including design)

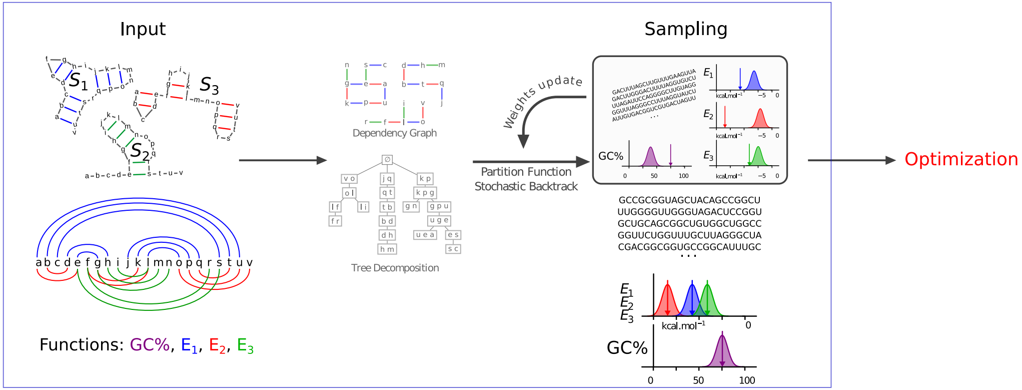


Infrared solves a weighted form of constraint problems (CSP $\rightarrow$ Feature Networks)
It allows us to describe ("model") problems; then solves them automatically.

[Hua-Ting Yao et al., 2024]

Infrared

General efficient framework for weighted constraint solving.
$\Rightarrow$ Rapid development of bioinformatics tools (including design)



Positive Design

Infrared solves a weighted form of constraint problems (CSP $\rightarrow$ Feature Networks)
It allows us to describe ("model") problems; then solves them automatically.

[Hua-Ting Yao et al., 2024]

Constraint Satisfaction Problems (CSPs)

Definition: A *CSP* is a tuple $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, where

- $\mathcal{X} = \{X_1, \dots, X_n\}$ is a set of *variables*
- $\mathcal{D} = \{D_1, \dots, D_n\}$ is a set of corresponding finite *domains*
- $\mathcal{C}$ is a finite set of *constraints*

Each *constraint* C is associated with k variables.

Solutions of a CSP are **assignments** of domain values to the variables that satisfy all constraints (**valid assignments**).

Constraint Satisfaction Problems (CSPs)

Definition: A *CSP* is a tuple $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, where

- $\mathcal{X} = \{X_1, \dots, X_n\}$ is a set of *variables*
- $\mathcal{D} = \{D_1, \dots, D_n\}$ is a set of corresponding finite *domains*
- $\mathcal{C}$ is a finite set of *constraints*

Each *constraint* C is associated with k variables.

Solutions of a CSP are **assignments** of domain values to the variables that satisfy all constraints (**valid assignments**).

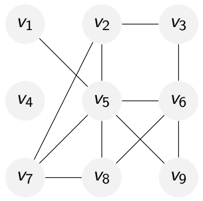
🧐 General solving is NP-hard! Solving strategies?

- Generic, heuristic solving strategies: backtracking search + constraint propagation
- ...Infrared is specialized to problems with nearly tree-like dependencies

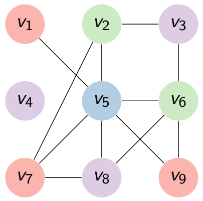
we gain: efficient (fpt) exact optimization + controlled sampling!

CSP Examples: Graph Coloring and N-Queens

A

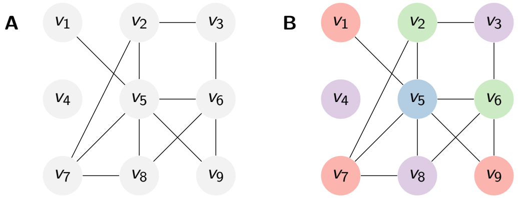


B



Constraints: Adjacent nodes differ in color!

CSP Examples: Graph Coloring and N-Queens



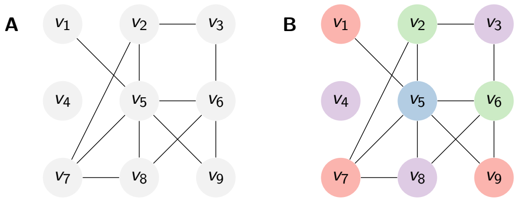
Constraints: Adjacent nodes differ in color!

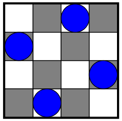
$CSP = (\mathcal{X}, \mathcal{D}, \mathcal{C})$

- $\mathcal{X} = \{X_1, \dots, X_9\}$
- $\mathcal{D} = \{X_1 \mapsto [1..4], \dots, X_4 \mapsto [1..4]\}$
- $\mathcal{C} = \{X_i \neq X_j \mid i, j \text{adjacent}\}$

```
model = Model(9,(1,4))
model.add_constraints(
    NotEquals(i,j) for i,j in edges)
```

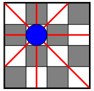
CSP Examples: Graph Coloring and N-Queens





$X_1 = 3$
$X_2 = 1$
$X_3 = 4$
$X_4 = 2$

Constraints: no attacks!



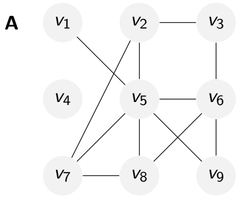
Constraints: Adjacent nodes differ in color!

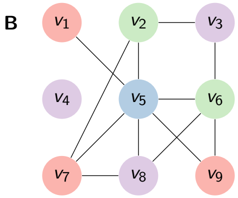
$CSP = (\mathcal{X}, \mathcal{D}, \mathcal{C})$

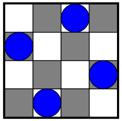
- $\mathcal{X} = \{X_1, \dots, X_9\}$
- $\mathcal{D} = \{X_1 \mapsto [1..4], \dots, X_4 \mapsto [1..4]\}$
- $\mathcal{C} = \{X_i \neq X_j \mid i, j \text{adjacent}\}$

```
model = Model(9,(1,4))
model.add_constraints(
    NotEquals(i,j) for i,j in edges)
```

CSP Examples: Graph Coloring and N-Queens

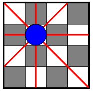






$X_1 = 3$
$X_2 = 1$
$X_3 = 4$
$X_4 = 2$

Constraints: no attacks!



Constraints: Adjacent nodes differ in color!

$CSP = (\mathcal{X}, \mathcal{D}, \mathcal{C})$

- $\mathcal{X} = \{X_1, \dots, X_9\}$
- $\mathcal{D} = \{X_1 \mapsto [1..4], \dots, X_4 \mapsto [1..4]\}$
- $\mathcal{C} = \{X_i \neq X_j \mid i, j \text{adjacent}\}$

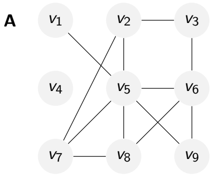
```
model = Model(9,(1,4))
model.add_constraints(
    NotEquals(i,j) for i,j in edges)
```

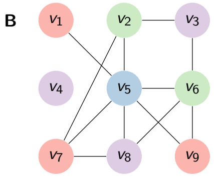
$CSP = (\mathcal{X}, \mathcal{D}, \mathcal{C})$

- $\mathcal{X} = \{X_1, \dots, X_4\}$
- $\mathcal{D} = \{X_1 \mapsto [1..4], \dots, X_4 \mapsto [1..4]\}$
- $\mathcal{C} = \{X_i \neq X_j \mid 1 \leq i < j \leq 4\}$
  $\cup \{X_i + i \neq X_j + j \mid 1 \leq i < j \leq 4\}$
  $\cup \{X_i - i \neq X_j - j \mid 1 \leq i < j \leq 4\}$

```
model = Model(4,(1,4))
model.add_constraints(NotEquals(i,j)
    for i in range(4) for j in range(i+1,4))
model.add_constraint( ...
```

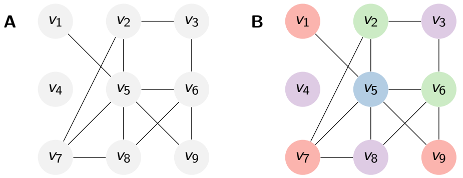
Dependency graph and tree decomposition



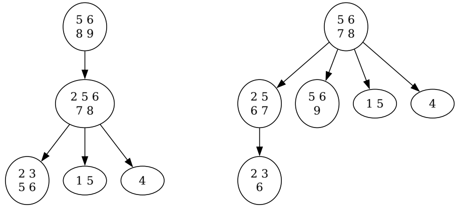


```
model = Model(9,(1,4))
edges = [(1,5), (2,3), (2,5), ... ]
model.add_constraints(NotEquals(i,j)
    for i,j in edges)
```

Dependency graph and tree decomposition



```
model = Model(9,(1,4))
edges = [(1,5), (2,3), (2,5), ... ]
model.add_constraints(NotEquals(i,j)
    for i,j in edges)
```



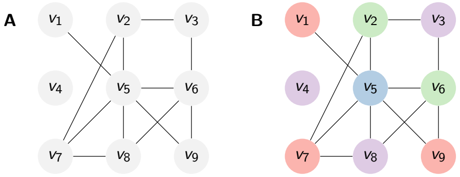
Tree decomposition (T,χ); $T=(V,E)$:

1. every variable occurs in one bag $v \in V$
2. for every constraint and function: there is one bag that contains its variables
3. for each variable: the bags containing it are connected

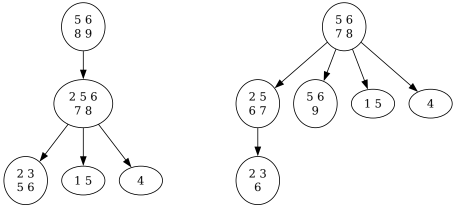
Tree width $=$ size of largest bag - 1

Remarks:

Dependency graph and tree decomposition



```
model = Model(9,(1,4))
edges = [(1,5), (2,3), (2,5), ... ]
model.add_constraints(NotEquals(i,j)
    for i,j in edges)
```



Tree decomposition (T, χ); $T = (V, E)$:

1. every variable occurs in one bag $v \in V$
2. for every constraint and function: there is one bag that contains its variables
3. for each variable: the bags containing it are connected

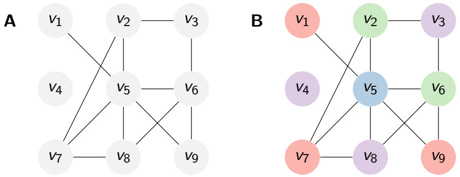
Tree width = size of largest bag - 1

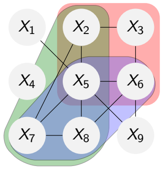
Remarks:

- Conditions 1–3 allow solving by dynamic programming (solve from smaller to larger subtrees)
- Condition 2 $\rightarrow$ every constraint and function can be processed in some bag

Adding functions $\rightarrow$ Objective function

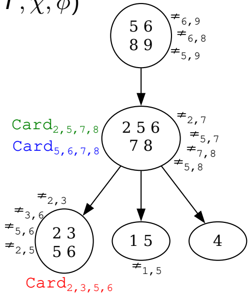
Make it more interesting by adding some functions





```
model = Model(9,(1,4))
model.add_constraints(NotEquals(i,j)
    for i,j in edges)

# extend by card feature
model.add_functions([Card(i,j,k,l)
    for i,j,k,l in fourcycles],'card')
```

Cluster tree (T, χ, ϕ)



Feature Networks

Feature networks add quality of solutions $\rightarrow$ **features**.

Definition

A *Feature Network* is a tuple $\mathcal{N} = (\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{F})$, where

- $\mathcal{X} = \{X_1, \dots, X_n\}$ is a set of *variables*
- $\mathcal{D} = \{D_1, \dots, D_n\}$ is a set of corresponding finite *domains*
- $\mathcal{C}$ is a finite set of *constraints*
- $\mathcal{F}$ is a finite set of *features*, which consist of *feature functions*

Features ...

- evaluate assignments as $F(x) = \sum_{f \in F} f(x)$
- define the *evaluation function* $E_{\mathcal{N}}(x, \alpha) = \sum_{F \in \mathcal{F}} \alpha_F F(x)$ for weights α_F

Infrared solves the sampling problem

Problem (Assignment sampling)

INPUT: *Feature Network $\mathcal{N}$, feature weights α*

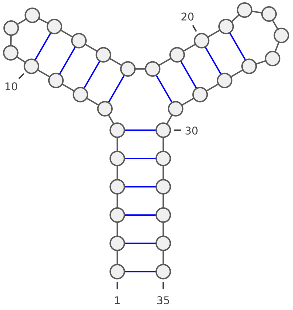
OUTPUT: *Valid assignment $x \in \mathcal{A}_{\mathcal{X}}$ generated with a probability that is proportional to its Boltzmann weight*

$$\mathbb{P}(x) \propto \exp\left(E_{\mathcal{N}}(x, \alpha)\right).$$

$\alpha = (\alpha_F)_{F \in \mathcal{F}}$ vector of weights

Evaluation function: $E_{\mathcal{N}}(x, \alpha) = \sum_{F \in \mathcal{F}} \alpha_F F(x)$.

Modeling: Single structure design

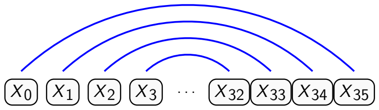


```
import infrared as ir
import infrared.rna as rna

model = ir.Model(35, 4)
```

0 : A 1 : C 2 : G 3 : U

```
target = "((((((((((...))))((((....))))))))))"
model.add_constraints(rna.BPComp(i, j)
    for (i, j) in rna.parse(target))
```

AU, CG, · · ·

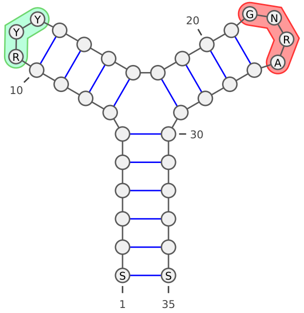


X_0 X_1 X_2 X_3 · · · X_{32} X_{33} X_{34} X_{35}

0:**A** 1:**C** 2:**G** 3:**U**

```
sampler = ir.Sampler(model)
samples = [sampler.sample() for _ in range(1000)]
```

Modeling: Single structure design



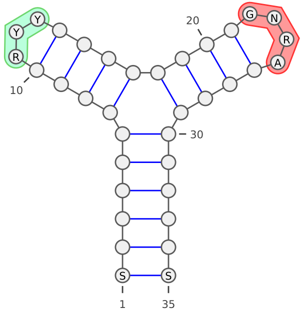
```
import infrared as ir
import infrared.rna as rna

model = ir.Model(35, 4)
```
0 : A 1 : C 2 : G 3 : U

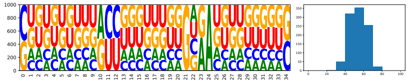
```
target = "((((((((((...)))))((((....)))))))))))"
model.add_constraints(rna.BPComp(i, j)
    for (i, j) in rna.parse(target))
```
AU, CG, · · ·

N : ACGU S : CG R : AG Y : CU

```
iupac_seq = "SNNNNNNNNNRYYNNNNNNNNGNRANNNNNNNNNS"
for i, x in enumerate(iupac_seq):
    model.add_constraints(
        ir.ValueIn(i, rna.iupacvalues(x)))

sampler = ir.Sampler(model)
samples = [sampler.sample() for _ in range(1000)]
```

Modeling: Single structure design





```
import infrared as ir
import infrared.rna as rna

model = ir.Model(35, 4)
```

0 : A   1 : C   2 : G   3 : U

```
target = "((((((((((...))))((((....)))))))))))"
model.add_constraints(rna.BPComp(i, j)
    for (i, j) in rna.parse(target))
```

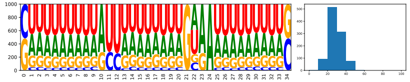
AU, CG, · · ·

N : ACGU   S : CG   R : AG   Y : CU

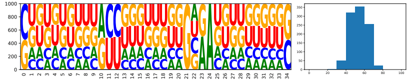
```
iupac_seq = "SNNNNNNNNNRYYNNNNNNNNGNRANNNNNNNNNS"
for i, x in enumerate(iupac_seq):
    model.add_constraints(
        ir.ValueIn(i, rna.iupacvalues(x)))


sampler = ir.Sampler(model)
samples = [sampler.sample() for _ in range(1000)]
```

Control GC-content

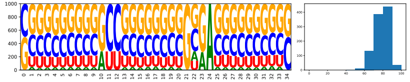
$\alpha = -1$



$\alpha = 0$



$\alpha = +1$



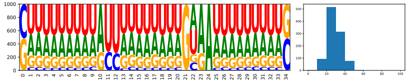
Method 1:

```
model.add_functions([rna.GCCont(i))
    for i in range(n)], 'gc')
model.set_feature_weight(α, 'gc')

sampler = ir.Sampler(model)
samples = [sampler.sample() for _ in range(1000)]
```

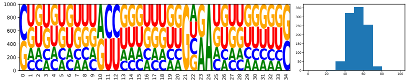
CG : 1   AU : 0

Control GC-content

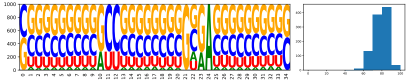
$\alpha = -1$



$\alpha = 0$



$\alpha = +1$



Method 1:

```
model.add_functions([rna.GCCont(i))
    for i in range(n)], 'gc')
model.set_feature_weight(α, 'gc')
```

CG : 1 AU : 0

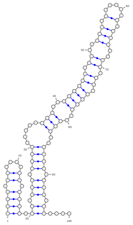
```
sampler = ir.Sampler(model)
samples = [sampler.sample() for _ in range(1000)]
```

Method 2 (Targeted sampling):

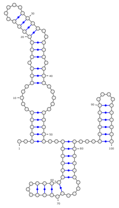
```
sampler = ir.Sampler(model)
sampler.set_target( 0.75 * n, 0.01 * n, 'gc' )
samples = [sampler.targeted_sample()
    for _ in range(1000)]
```

Automatically learn α

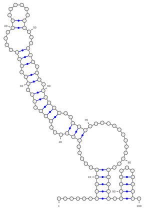
Multitarget design



t_1



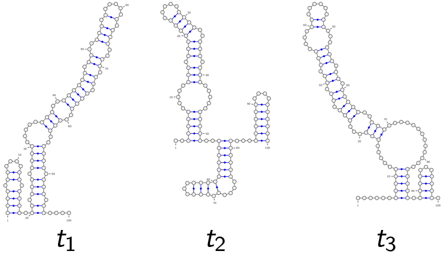
t_2

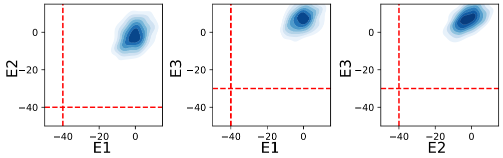


t_3

```
model = ir.Model(n,4)
for k, target in enumerate(targets):
    bps = rna.parse(target)
    model.add_constraints(rna.BPComp(i, j)
        for (i, j) in bps)
```

Multidimensional Boltzmann sampling





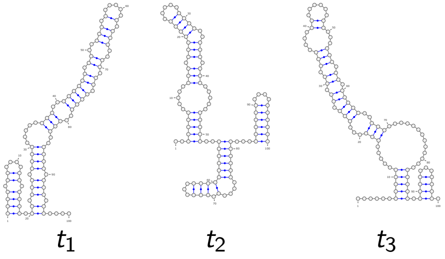
uniform

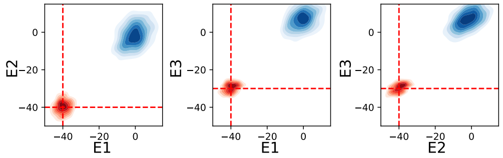
```
model = ir.Model(n,4)
for k, target in enumerate(targets):
    bps = rna.parse(target)
    model.add_constraints(rna.BPComp(i, j)
        for (i, j) in bps)

    model.add_functions([rna.BPEnergy(i, j)
        for (i, j) in bps], f'energy{k}')

for k, target in enumerate(targets):
    model.add_feature(f'E{k+1}', f'energy{k}',
        lambda sample, target=target:
            energy_of_struct(sample, target))

sampler = ir.Sampler(model)
```

Simplified energy model

ViennaRNA energy model

Multidimensional Boltzmann sampling





uniform targeted

```
model = ir.Model(n,4)

for k, target in enumerate(targets):
    bps = rna.parse(target)
    model.add_constraints(rna.BPComp(i, j)
        for (i, j) in bps)

    model.add_functions([rna.BPEnergy(i, j)
        for (i, j) in bps], f'energy{k}')
```

Simplified energy model

```
for k, target in enumerate(targets):
    model.add_feature(f'E{k+1}', f'energy{k}',
        lambda sample, target=target:
            energy_of_struct(sample, target))
```

ViennaRNA energy model

```
sampler = ir.Sampler(model)
sampler.set_target( -40, 0.5, 'E1')
sampler.set_target( -40, 0.5, 'E2')
sampler.set_target( -30, 0.5, 'E3')
```

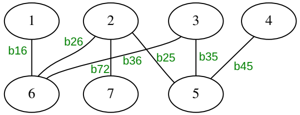
Fixed-parameter tractable sampling in Infrared

Recipe:

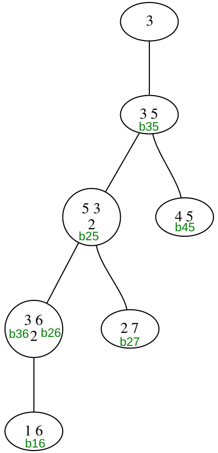
1. **Tree-Decompose** dependency graph
2. Apply **dynamic programming** ↑ (partition functions)
3. **Sample** ↓ (stochastic traceback)

```
1  2  3  4  5  6  7

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

target structures



dependency graph



tree decomposition

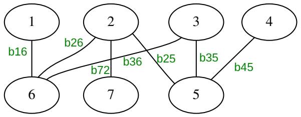
Fixed-parameter tractable sampling in Infrared

Recipe:

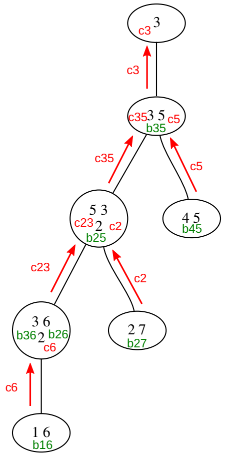
1. **Tree-Decompose** dependency graph
2. Apply **dynamic programming** ↑ (partition functions)
3. **Sample** ↓ (stochastic traceback)

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

target structures



dependency graph



tree decomposition

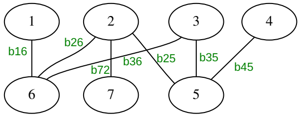
Fixed-parameter tractable sampling in Infrared

Recipe:

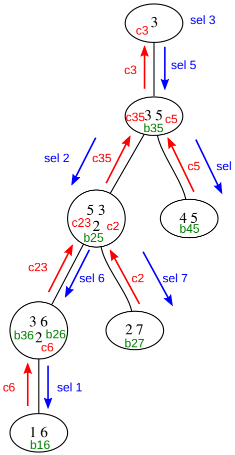
1. **Tree-Decompose** dependency graph
2. Apply **dynamic programming** $\uparrow$ (partition functions)
3. **Sample** $\downarrow$ (stochastic traceback)

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

target structures



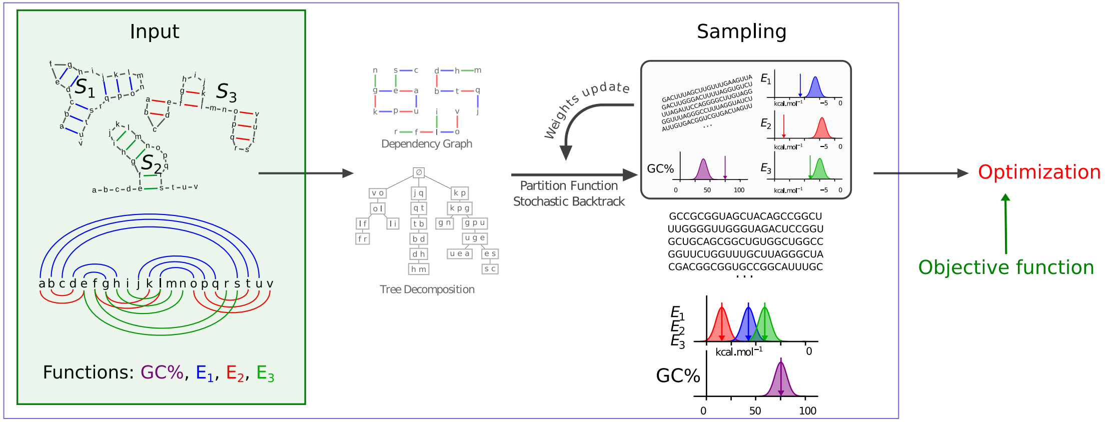
dependency graph



tree decomposition

Theorem: Design sampling is efficient for fixed tree width w: $\mathcal{O}(n\,k\,\mathbf{4}^{\mathbf{w+1}} + t\,n\,k)$

Look back at positive design by Infrared



Positive Design

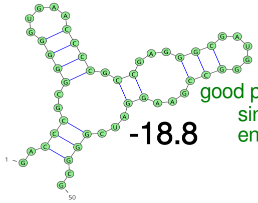
RNA structure design: positive and negative

Positive design: Target a structure

$\rightarrow$ optimize affinity to target structures t

find sequence σ
with $E(\sigma, t) = \min_{\sigma'}(\sigma', t)$

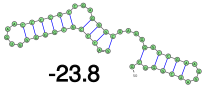
extensions: multiple targets, properties, ...



-18.8

good positive design
since all/most other sequences
energetically worse

but no negative design, since



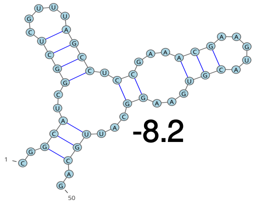
-23.8

Negative design: Avoid all off-targets

$\rightarrow$ specificity for targets

find design σ, s.t.
$E(\sigma, t) = \min_{t'} E(\sigma, t') \quad =: MFE(\sigma)$

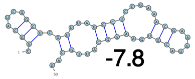
refined objectives: probability, ensemble, ...



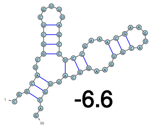
-8.2

negative design
(by MFE criterion)

since all other structures
energetically worse, e.g.



-7.8



-6.6

....

Avoiding off-targets: negative design as optimization

minimize objective function over all sequences σ w.r.t. a target structure t

- *MFE defect:* base pair distance of MFE structure of σ and t,

$$D_{MFE}(\sigma) = d(MFE(\sigma), t)$$

where *base pair distance* $d(s,t) := \sum_{(i,j)\notin s,(i,j)\in t} 1 + \sum_{(i,j)\in s,(i,j)\notin t} 1$

Ex.: $d_{bp}\left(\begin{array}{c} (((\ldots).).), \\ ((.(\ldots)).) \end{array} \right) = 2$

Avoiding off-targets: negative design as optimization

minimize objective function over all sequences σ w.r.t. a target structure t

- *MFE defect:* base pair distance of MFE structure of σ and t,

$$D_{MFE}(\sigma) = d(MFE(\sigma), t)$$

where *base pair distance* $d(s,t) := \sum_{(i,j)\notin s,(i,j)\in t} 1 + \sum_{(i,j)\in s,(i,j)\notin t} 1$

Ex.: $d_{bp}\left(\begin{array}{c} (((...).).), \\ ((.(...)).) \end{array}\right) = 2$

- *probability defect*: $D_{Pr}(\sigma) = 1 - \Pr[t \mid \sigma]$
  maximize probability of the target t in the ensemble of σ

Avoiding off-targets: negative design as optimization

minimize objective function over all sequences σ w.r.t. a target structure t

- *MFE defect:* base pair distance of MFE structure of σ and t,

$$D_{MFE}(\sigma) = d(MFE(\sigma), t)$$

where *base pair distance* $d(s,t) := \sum_{(i,j)\notin s,(i,j)\in t} 1 + \sum_{(i,j)\in s,(i,j)\notin t} 1$

Ex.: $d_{bp}\begin{pmatrix} (((\ldots).).), \\ ((.(\ldots)).) \end{pmatrix} = 2$

- *probability defect*: $D_{Pr}(\sigma) = 1 - \Pr[t \mid \sigma]$
  maximize probability of the target t in the ensemble of σ

🤔 this does not consider whether the probable structures are close to target

Avoiding off-targets: negative design as optimization

minimize objective function over all sequences σ w.r.t. a target structure t

- *MFE defect:* base pair distance of MFE structure of σ and t,

$$D_{MFE}(\sigma) = d(MFE(\sigma), t)$$

where *base pair distance* $d(s,t) := \sum_{(i,j)\notin s,(i,j)\in t} 1 + \sum_{(i,j)\in s,(i,j)\notin t} 1$

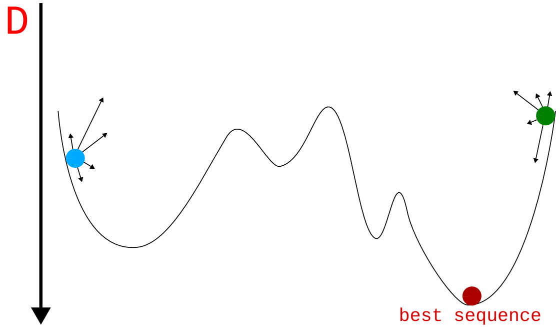
$$\text{Ex.: } d_{bp}\left(\begin{array}{c} (((\ldots).).), \\ ((.(\ldots)).) \end{array}\right) = 2$$

- *probability defect*: $D_{Pr}(\sigma) = 1 - \Pr[t \mid \sigma]$
  maximize probability of the target t in the ensemble of σ

- *ensemble defect*: expected distance of ensemble structures s of σ to the target t

$$D_{ens}(\sigma) = \sum \Pr[s \mid \sigma] d(s,t) = \sum_{1\leq i<j\leq n,(i,j)\in t} 1 - p_{ij} + \sum_{1\leq i<j\leq n,(i,j)\notin t} p_{ij}$$

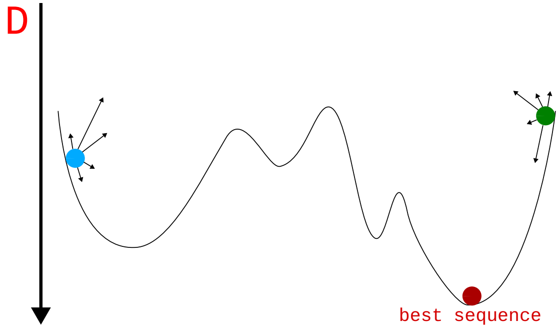
Algorithms for negative design: Stochastic Optimization



🤔 rugged landscape, local optima

🤔 random starts; neighbors, mutations

Algorithms for negative design: Stochastic Optimization



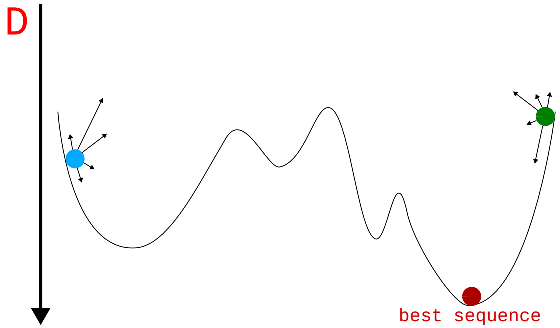
Hill Climbing

```
for i in range(steps):
    x = random_mutate(seq)
    if D(x) < D(seq):
        seq = x
return seq
```

🤔 rugged landscape, local optima

🤔 random starts; neighbors, mutations

Algorithms for negative design: Stochastic Optimization



🤔 rugged landscape, local optima

🤔 random starts; neighbors, mutations

Hill Climbing

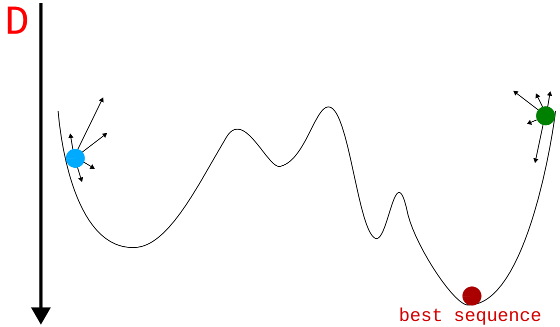
```
for i in range(steps):
    x = random_mutate(seq)
    if D(x) < D(seq):
        seq = x
return seq
```

Metropolis-Hastings MC Algorithm

```
best = seq
for i in range(steps):
    x = random_mutate(seq)
    if D(x) < D(seq) or
      random()<=exp((D(x)-D(seq))/T)
        seq = x
        if D(seq)<D(best): best=seq
return best
```

Control acceptance of worse neighbors by T
(MCMC, Boltzmann distribution)

Algorithms for negative design: Stochastic Optimization



🤔 rugged landscape, local optima

🤔 random starts; neighbors, mutations

Hill Climbing

```
for i in range(steps):
    x = random_mutate(seq)
    if D(x) < D(seq):
        seq = x
return seq
```

Metropolis-Hastings MC Algorithm

```
best = seq
for i in range(steps):
    x = random_mutate(seq)
    if D(x) < D(seq) or
      random()<=exp((D(x)-D(seq))/T)
        seq = x
        if D(seq)<D(best): best=seq
return best
```

SA, Replica exchange, Genetic algos, ...

RNAinverse - Classical RNA design

The *single sequence design* tool of the Vienna RNA package

- Optimizes MFE defect or probability defect

[Ivo Hofacker et al., 1994]

RNAinverse - Classical RNA design

The *single sequence design* tool of the Vienna RNA package

- Optimizes MFE defect or probability defect
- Try **random start sequences** and optimize by **Hill climbing**

needs many evaluations of objective 🧐 expensive?

[Ivo Hofacker et al., 1994]

RNAinverse - Classical RNA design

The *single sequence design* tool of the Vienna RNA package

- Optimizes MFE defect or probability defect
- Try **random start sequences** and optimize by **Hill climbing**

  needs many evaluations of objective 🧐 expensive?

- for MFE design, RNA-tailored strategy:
  start at small substrutures, proceed to larger ones
  avoid getting stuck; reduce folding of long sequences

[Ivo Hofacker et al., 1994]

RNAinverse - Classical RNA design

The *single sequence design* tool of the Vienna RNA package

- Optimizes MFE defect or probability defect
- Try **random start sequences** and optimize by **Hill climbing**

  needs many evaluations of objective 🧐 expensive?

- for MFE design, RNA-tailored strategy:
  start at small substrutures, proceed to larger ones
  avoid getting stuck; reduce folding of long sequences

```
((((((((((...))))((((....))))))))))
???????????????????????????????????
```

[Ivo Hofacker et al., 1994]

RNAinverse - Classical RNA design

The *single sequence design* tool of the Vienna RNA package

- Optimizes MFE defect or probability defect
- Try **random start sequences** and optimize by **Hill climbing**

  needs many evaluations of objective 🧐 expensive?

- for MFE design, RNA-tailored strategy:
  start at small substrutures, proceed to larger ones
  avoid getting stuck; reduce folding of long sequences

```
((((((((((...))))((((....))))))))))
??????CGGCAAAGCCG??????????????????
```

[Ivo Hofacker et al., 1994]

RNAinverse - Classical RNA design

The *single sequence design* tool of the Vienna RNA package

- Optimizes MFE defect or probability defect
- Try **random start sequences** and optimize by **Hill climbing**

  needs many evaluations of objective expensive?

- for MFE design, RNA-tailored strategy:
  start at small substrutures, proceed to larger ones
  avoid getting stuck; reduce folding of long sequences

```
((((((((((...))))((((....)))))))))))
??????CGGCAAAGCCGGGCCAAUUGGCC??????
```

[Ivo Hofacker et al., 1994]

RNAinverse - Classical RNA design

The *single sequence design* tool of the Vienna RNA package

- Optimizes MFE defect or probability defect
- Try **random start sequences** and optimize by **Hill climbing**

  needs many evaluations of objective 🤔 expensive?

- for MFE design, RNA-tailored strategy:
  start at small substrutures, proceed to larger ones
  avoid getting stuck; reduce folding of long sequences

```
((((((((((...))))((((....)))))))))))
??????CGGCAAAGCCGGGCCUUUUGGCC??????
```

[Ivo Hofacker et al., 1994]

RNAinverse - Classical RNA design

The *single sequence design* tool of the Vienna RNA package

- Optimizes MFE defect or probability defect
- Try **random start sequences** and optimize by **Hill climbing**

  needs many evaluations of objective 🧐 expensive?

- for MFE design, RNA-tailored strategy:
  start at small substrutures, proceed to larger ones
  avoid getting stuck; reduce folding of long sequences

```
((((((((((...))))((((....))))))))))
??????CGGCAAAGCCGGGCCAAAUGGCC??????
```

[Ivo Hofacker et al., 1994]

RNAinverse - Classical RNA design

The *single sequence design* tool of the Vienna RNA package

- Optimizes MFE defect or probability defect
- Try **random start sequences** and optimize by **Hill climbing**

  needs many evaluations of objective 🧐 expensive?

- for MFE design, RNA-tailored strategy:
  start at small substrutures, proceed to larger ones
  avoid getting stuck; reduce folding of long sequences

For global optima, subsequence designs must be optimal!

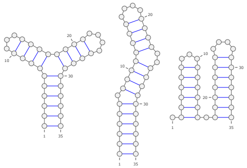
🧐 converse?? 🧐 why does this still work well?

RNAinverse still remarkably competitive (using good starting sequences) [Boury et al., 2024]

[Ivo Hofacker et al., 1994]

Beyond single targets: objectives for multi-target design

```
                 1         2         3
       01234567890123456789012345678901234
t1  =  ((((((((((...)))))((((....)))))))))
t2  =  ((((((.((((((((....))))..))))))))))
t3  =  .((((((...)))))).(((((((....)))))))
```



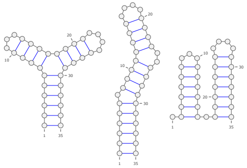
- "Multi-defect" for targets $t_1, \dots t_m$ [Hammer et al., "RNA Blueprint" 2017]

$$D_{multi}(\sigma) = \begin{cases} \frac{1}{m} \sum_{1 \le \ell \le m} E(\sigma, t_\ell) - G(\sigma) & \text{(dominate ensemble)} \\ + \quad \xi \frac{1}{2\binom{m}{2}} \sum_{1 \le k < \ell \le m} |E(\sigma, t_k) - E(\sigma, t_\ell)| & \text{(similar energies)} \end{cases}$$

$G(\sigma) = -RT \ln Z(\sigma)$ "ensemble energy"

Beyond single targets: objectives for multi-target design

```
                     1         2         3
           01234567890123456789012345678901234
t1  =  ((((((((((...)))))((((....)))))))))))
t2  =  ((((((.((((((((....))))..))))))))))
t3  =  .(((((((...))))))).((((((((....))))))))
```



- "Multi-defect" for targets $t_1, \dots t_m$ [Hammer et al., "RNA Blueprint" 2017]

$$D_{multi}(\sigma) = \begin{cases} \frac{1}{m} \sum_{1 \le \ell \le m} E(\sigma, t_\ell) - G(\sigma) & \text{(dominate ensemble)} \\ + \quad \xi \frac{1}{2\binom{m}{2}} \sum_{1 \le k < \ell \le m} |E(\sigma, t_k) - E(\sigma, t_\ell)| & \text{(similar energies)} \end{cases}$$

- Aim for ensemble dominance with certain energy differences of targets..., e.g.

$$D_{ex}(\sigma) = |E(\sigma, t_1) - G(\sigma)| + |E(\sigma, t_2) - E(\sigma, t_1) - 3| + |E(\sigma, t_3) - E(\sigma, t_1) - 4|$$

$G(\sigma) = -RT \ln Z(\sigma)$ "ensemble energy"

Stochastic optimization in Infrared

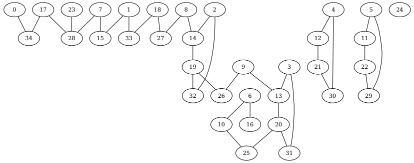
How to find valid neighbors in complex design problems?

```
          1         2         3
01234567890123456789012345678901234
((((((((((...))))((((....))))))))))
((((((.((((((((....))))..))))))))))
.((((((...)))))).(((((((....)))))))
GCGUGCGGGGGAGUCUCUCCGUCAAUGGGGCACGC
```

Stochastic optimization in Infrared

How to find valid neighbors in complex design problems?

```
          1         2         3
01234567890123456789012345678901234
((((((((((...))))((((....))))))))))
((((((.((((((((....))))..))))))))))
.((((((...)))))).(((((((....)))))))
GCGUGCGGGGGAGUCUCUCCGUCAAUGGGGCACGC
```



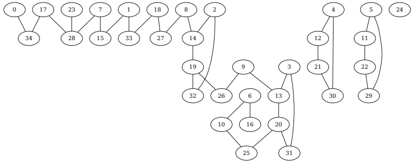
- resample connected components (of dependency graph)

  Idea: independence of cc preserves all other constraints

Stochastic optimization in Infrared

How to find valid neighbors in complex design problems?

```
          1         2         3
01234567890123456789012345678901234
((((((((((...)))))((((....)))))))))
((((((.((((((((....))))..))))))))))
.((((((...)))))).(((((((....)))))))
GCGUGCGGGGAGUCUCUCCGUCAAUGGGGCACGC
```



- resample connected components (of dependency graph)

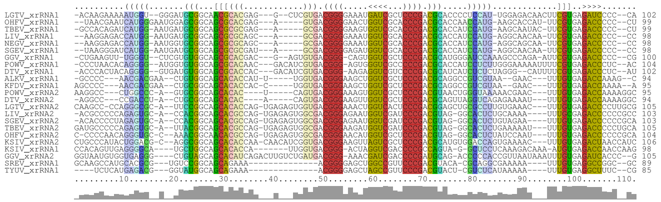
  Idea: independence of cc preserves all other constraints

- sample neighbors in a targeted distance

  Idea: • extend model by distance function
  • sample neighbors, controlled by distance to current sequence

Learning design from evolution (Generative Models)

General idea: learn from homology information / MSAs

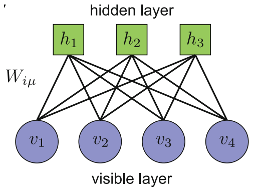


There is information in MSA beyond position-wise frequencies! ⇒ covariation...

⇒ Restricted Boltzmann Machines (RBM)

- (bipartite) two layer neural networks
- can be trained to evaluate sequences
- shown to design SAM aptamer

[FdCD et al., 2023]

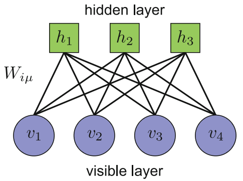


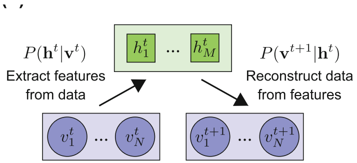
$E(\mathbf{v}, \mathbf{h})$

[Jorge Fernandez-de-Cossio-Diaz, 2024]

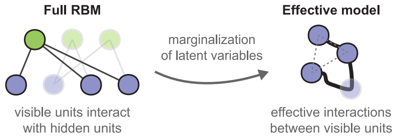
Restricted Boltzmann Machines (RBMs)

v_μ: `A`, `C`, `G`, `U`, `-`
h_i: dependencies




$P(\mathbf{h}^t|\mathbf{v}^t)$
Extract features from data
h_1^t ... h_M^t
$P(\mathbf{v}^{t+1}|\mathbf{h}^t)$
Reconstruct data from features
v_1^t ... v_N^t
v_1^{t+1} ... v_N^{t+1}

PCD, Gibbs sampler


Full RBM
visible units interact with hidden units
marginalization of latent variables
Effective model
effective interactions between visible units

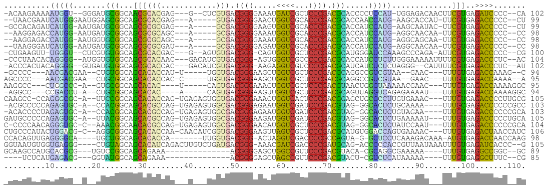
$E_{eff}(\mathbf{v}) \leftarrow$ marginalization

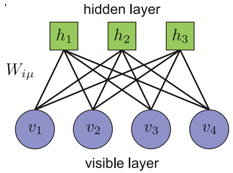
$$E(\mathbf{v},\mathbf{h}) = \sum_{i=1}^{N} \mathcal{V}_i(v_i) + \sum_{\mu=1}^{M} \mathcal{U}_\mu(h_\mu) - \sum_{i=1}^{N}\sum_{\mu=1}^{M} w_{i\mu}(v_i)h_\mu$$

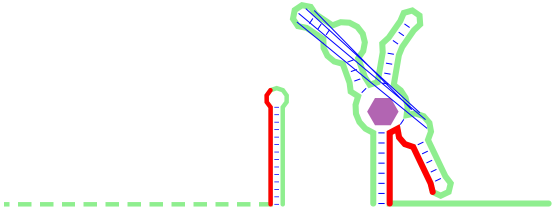
- Effective training: maximize log likelihood of training data by *persistent contrastive divergence (PDC)* [Hinton, 2012]
- Evaluation of designs
- Positive design / sampling

[Jorge Fernandez-de-Cossio-Diaz, 2024]

Hands on session







- Use of Infrared
- Design of SAM-I aptamer
- Design of SAM-I on-switch (simplified)

- Sample and optimize
- Integrate homology and thermodynamic info
- Integrate evaluation by RBM