BIOINFORMATICS
2019

The Composition of Dense Neural Networks and Formal Grammars for Secondary Structure Analysis

Polina Lunina, **Semyon Grigorev**

JetBrains Research, Programming Languages and Tools Lab
Saint Petersburg University

Febrary 24, 2019

Long Sequences Analysis

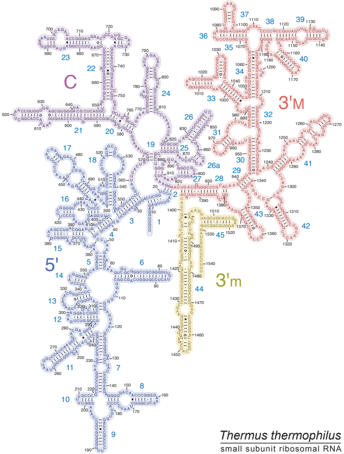
$$\underbrace{\texttt{CACATGGAGAGTTTGA...CTGGATCACCTCCTTT}}_{\sim 1500 \text{ symbols}}$$

- Classification

Long Sequences Analysis

$$\underbrace{\texttt{CACATGGAGAGTTTGA...CTGGATCACCTCCTTT}}_{\sim 1500 \text{ symbols}}$$

- Classification
  - Secondary structure handling

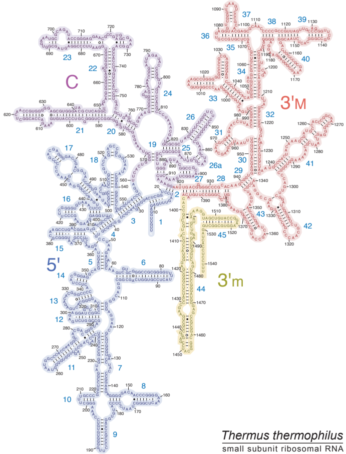


Thermus thermophilus
small subunit ribosomal RNA

Long Sequences Analysis

$$\underbrace{\texttt{CACATGGAGAGTTTGA...CTGGATCACCTCCTTT}}_{\sim 1500 \text{ symbols}}$$

- Classification
  - Secondary structure handling
- Metagenomic assembly processing
  - Filter out chimeric sequences
  - Secondary structure handling



Thermus thermophilus
small subunit ribosomal RNA

Proposed Solution: Parsing + Artificial Neural Network

- Use parsing to extract features, not to model secondary structure
  - As compared to the classical way of probabilistic CF grammars utilization

Proposed Solution: Parsing + Artificial Neural Network

- Use parsing to extract features, not to model secondary structure
  - As compared to the classical way of probabilistic CF grammars utilization
- Formal grammars as secondary structure description
- Parsing as features extraction
- Artificial neural network as probabilistic model for features processing

Solution Structure

Grammar
Fixed formal grammar (not necessarily context-free) describes features of secondary structure and can be tuned to increase the quality of result.

Sequences
Each sequence is treated as a text in $\{A, C, G, T\}$ alphabet.

Result of classification

Parser
Parser extracts features of the given sequence secondary structure. Implementation of parsing algorithm is based on matrix multiplications (Valiant, Okhotin) and utilizes GPGPU.

Matrices

$$\begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Parsing result is (0-1) matrix M which represents secondary structure features for sequence ω:
$M[i,j] = 1 \iff \texttt{s1} \xrightarrow{*} \omega[i,j]$,
and 0 otherwise.

Vectors

$$\begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\Downarrow$$

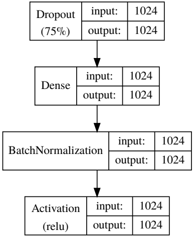
`[0,1,0,1,0,1,0,0,1,0]`

$$\Downarrow$$

`[84,128]`

Line-by-line compressed matrix representation: sequence of 8 cells (bits) is compressed into a byte. Bottom left triangle of the matrix is always empty, so can be ignored.

Neural Network
Dense neural network with more than 10 dense layers. Agressive dropout and batch normalization for learning process stabilization. Typical building block:



Solution Structure

Grammar
Fixed formal grammar (not necessarily context-free) describes features of secondary structure and can be tuned to increase the quality of result.

Sequences
Each sequence is treated as a text in $\{A, C, G, T\}$ alphabet.

Parser
Parser extracts features of the given sequence secondary structure. Implementation of parsing algorithm is based on matrix multiplications (Valiant, Okhotin) and utilizes GPGPU.

Matrices

$$\begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Parsing result is (0-1) matrix M which represents secondary structure features for sequence ω:
$M[i,j] = 1 \iff \mathtt{s1} \xrightarrow{*} \omega[i,j]$, and 0 otherwise.

Vectors

$$\begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\Downarrow$$

`[0,1,0,1,0,1,0,0,1,0]`

$$\Downarrow$$

`[84,128]`

Line-by-line compressed matrix representation: sequence of 8 cells (bits) is compressed into a byte. Bottom left triangle of the matrix is always empty, so can be ignored.

Neural Network
Dense neural network with more than 10 dense layers. Agressive dropout and batch normalization for learning process stabilization. Typical building block:

Dropout (75%)	input:	1024
	output:	1024

$\downarrow$

Dense	input:	1024
	output:	1024

$\downarrow$

BatchNormalization	input:	1024
	output:	1024

$\downarrow$

Activation (relu)	input:	1024
	output:	1024

Result of classification

Solution Structure

Grammar
Fixed formal grammar (not necessarily context-free) describes features of secondary structure and can be tuned to increase the quality of result.

Sequences
Each sequence is treated as a text in $\{A, C, G, T\}$ alphabet.

Parser
Parser extracts features of the given sequence secondary structure. Implementation of parsing algorithm is based on matrix multiplications (Valiant, Okhotin) and utilizes GPGPU.

Matrices

$$\begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Parsing result is (0-1) matrix M which represents secondary structure features for sequence ω: $M[i,j] = 1 \iff \mathtt{s1} \xrightarrow{*} \omega[i,j]$, and 0 otherwise.

Vectors

$$\begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\Downarrow$$

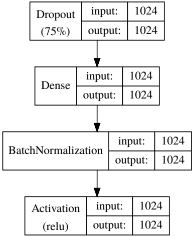
`[0,1,0,1,0,1,0,0,1,0]`

$$\Downarrow$$

`[84,128]`

Line-by-line compressed matrix representation: sequence of 8 cells (bits) is compressed into a byte. Bottom left triangle of the matrix is always empty, so can be ignored.

Neural Network
Dense neural network with more than 10 dense layers. Agressive dropout and batch normalization for learning process stabilization. Typical building block:



Result of classification

Solution Structure

Grammar
Fixed formal grammar (not necessarily context-free) describes features of secondary structure and can be tuned to increase the quality of result.

Sequences
Each sequence is treated as a text in $\{A, C, G, T\}$ alphabet.

Parser
Parser extracts features of the given sequence secondary structure. Implementation of parsing algorithm is based on matrix multiplications (Valiant, Okhotin) and utilizes GPGPU.

Matrices

$$\begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Parsing result is (0-1) matrix M which represents secondary structure features for sequence ω:
$M[i,j] = 1 \iff \mathtt{s1} \xrightarrow{*} \omega[i,j]$, and 0 otherwise.

Vectors

$$\begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$\Downarrow$

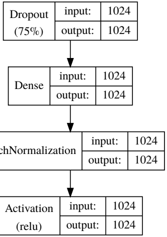
`[0,1,0,1,0,1,0,0,1,0]`

$\Downarrow$

`[84,128]`

Line-by-line compressed matrix representation: sequence of 8 cells (bits) is compressed into a byte. Bottom left triangle of the matrix is always empty, so can be ignored.

Neural Network
Dense neural network with more than 10 dense layers. Agressive dropout and batch normalization for learning process stabilization. Typical building block:



Result of classification

Solution Structure

Grammar
Fixed formal grammar (not necessarily context-free) describes features of secondary structure and can be tuned to increase the quality of result.

Sequences
Each sequence is treated as a text in $\{A, C, G, T\}$ alphabet.

Result of classification

Parser
Parser extracts features of the given sequence secondary structure. Implementation of parsing algorithm is based on matrix multiplications (Valiant, Okhotin) and utilizes GPGPU.

Matrices

$$\begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Parsing result is (0-1) matrix M which represents secondary structure features for sequence ω:
$M[i,j] = 1 \iff \mathtt{s1} \xrightarrow{*} \omega[i,j]$, and 0 otherwise.

Vectors

$$\begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\Downarrow$$

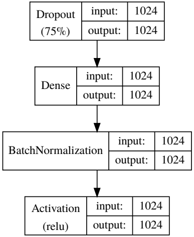
`[0,1,0,1,0,1,0,0,1,0]`

$$\Downarrow$$

`[84,128]`

Line-by-line compressed matrix representation: sequence of 8 cells (bits) is compressed into a byte. Bottom left triangle of the matrix is always empty, so can be ignored.

Neural Network
Dense neural network with more than 10 dense layers. Agressive dropout and batch normalization for learning process stabilization. Typical building block:



Solution Structure

Grammar
Fixed formal grammar (not necessarily context-free) describes features of secondary structure and can be tuned to increase the quality of result.

Sequences
Each sequence is treated as a text in $\{A, C, G, T\}$ alphabet.

Result of classification

Parser
Parser extracts features of the given sequence secondary structure. Implementation of parsing algorithm is based on matrix multiplications (Valiant, Okhotin) and utilizes GPGPU.

Matrices

$$\begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Parsing result is (0-1) matrix M which represents secondary structure features for sequence ω:
$M[i,j] = 1 \iff \mathtt{s1} \xrightarrow{*} \omega[i,j]$, and 0 otherwise.

Vectors

$$\begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\Downarrow$$

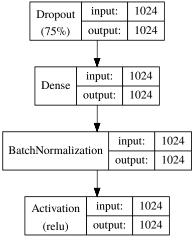
`[0,1,0,1,0,1,0,0,1,0]`

$$\Downarrow$$

`[84,128]`

Line-by-line compressed matrix representation: sequence of 8 cells (bits) is compressed into a byte. Bottom left triangle of the matrix is always empty, so can be ignored.

Neural Network
Dense neural network with more than 10 dense layers. Agressive dropout and batch normalization for learning process stabilization. Typical building block:



Solution Structure

Grammar
Fixed formal grammar (not necessarily context-free) describes features of secondary structure and can be tuned to increase the quality of result.

Sequences
Each sequence is treated as a text in $\{A, C, G, T\}$ alphabet.

Result of classification

Parser
Parser extracts features of the given sequence secondary structure. Implementation of parsing algorithm is based on matrix multiplications (Valiant, Okhotin) and utilizes GPGPU.

Matrices

$$\begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Parsing result is (0-1) matrix M which represents secondary structure features for sequence ω:
$M[i,j] = 1 \iff \mathtt{s1} \xrightarrow{*} \omega[i,j]$, and 0 otherwise.

Vectors

$$\begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$\Downarrow$

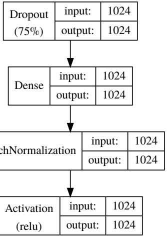
[0,1,0,1,0,1,0,0,1,0]

$\Downarrow$

[84,128]

Line-by-line compressed matrix representation: sequence of 8 cells (bits) is compressed into a byte. Bottom left triangle of the matrix is always empty, so can be ignored.

Neural Network
Dense neural network with more than 10 dense layers. Agressive dropout and batch normalization for learning process stabilization. Typical building block:



Grammar

```
s1: stem<s0>
any_str: any_smb*[2..10]
any_smb: A | T | C | G
stem1<s>:                  \\ stem of height exactly 1
      A s T | T s A | C s G | G s C
stem3<s>:                  \\ stem of height exactly 3
      stem1< stem1< stem1<s> > >
stem<s>:                   \\ stem of height 3 or more
      A stem<s> T
    | T stem<s> A
    | C stem<s> G
    | G stem<s> C
    | stem3<s>
s0: any_str | any_str stem<s0> s0
```

Grammar

```
s1: stem<s0>
any_str: any_smb*[2..10]
any_smb: A | T | C | G
stem1<s>:                  \\ stem of height exactly 1
      A s T | T s A | C s G | G s C
stem3<s>:                  \\ stem of height exactly 3
      stem1< stem1< stem1<s> > >
stem<s>:                   \\ stem of height 3 or more
      A stem<s> T
    | T stem<s> A
    | C stem<s> G
    | G stem<s> C
    | stem3<s>
s0: any_str | any_str stem<s0> s0
```

Grammar

```
s1: stem<s0>
any_str: any_smb*[2..10]
any_smb: A | T | C | G
stem1<s>:                   \\ stem of height exactly 1
      A s T | T s A | C s G | G s C
stem3<s>:                   \\ stem of height exactly 3
      stem1< stem1< stem1<s> > >
stem<s>:                    \\ stem of height 3 or more
      A stem<s> T
    | T stem<s> A
    | C stem<s> G
    | G stem<s> C
    | stem3<s>
s0: any_str | any_str stem<s0> s0
```

Grammar

```
s1: stem<s0>
any_str: any_smb*[2..10]
any_smb: A | T | C | G
stem1<s>:                 \\ stem of height exactly 1
      A s T | T s A | C s G | G s C
stem3<s>:                 \\ stem of height exactly 3
      stem1< stem1< stem1<s> > >
stem<s>:                  \\ stem of height 3 or more
      A stem<s> T
    | T stem<s> A
    | C stem<s> G
    | G stem<s> C
    | stem3<s>
s0: any_str | any_str stem<s0> s0
```

Grammar

```
s1: stem<s0>
any_str: any_smb*[2..10]
any_smb: A | T | C | G
stem1<s>:                  \\ stem of height exactly 1
      A s T | T s A | C s G | G s C
stem3<s>:                  \\ stem of height exactly 3
      stem1< stem1< stem1<s> > >
stem<s>:                   \\ stem of height 3 or more
      A stem<s> T
    | T stem<s> A
    | C stem<s> G
    | G stem<s> C
    | stem3<s>
s0: any_str | any_str stem<s0> s0
```

Grammar

```
s1: stem<s0>
any_str: any_smb*[2..10]
any_smb: A | T | C | G
stem1<s>:                  \\ stem of height exactly 1
      A s T | T s A | C s G | G s C
stem3<s>:                  \\ stem of height exactly 3
      stem1< stem1< stem1<s> > >
stem<s>:                   \\ stem of height 3 or more
      A stem<s> T
    | T stem<s> A
    | C stem<s> G
    | G stem<s> C
    | stem3<s>
s0: any_str | any_str stem<s0> s0
```

Grammar

```
s1: stem<s0>
any_str: any_smb*[2..10]
any_smb: A | T | C | G
stem1<s>:                   \\ stem of height exactly 1
      A s T | T s A | C s G | G s C
stem3<s>:                   \\ stem of height exactly 3
      stem1< stem1< stem1<s> > >
stem<s>:                    \\ stem of height 3 or more
      A stem<s> T
    | T stem<s> A
    | C stem<s> G
    | G stem<s> C
    | stem3<s>
s0: any_str | any_str stem<s0> s0
```

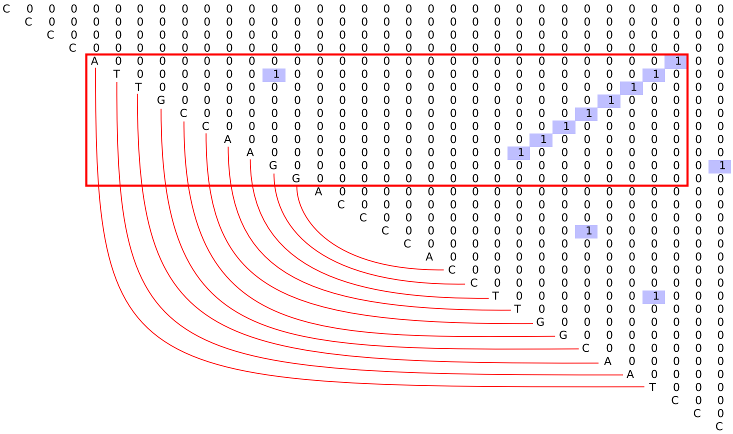
Example 1: Stem

CCCCATTGCCAAGGACCCCACCTTGGCAATCCC

C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
			C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
				A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
					T	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
						T	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
							G	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
								C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
									C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
										A	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
											A	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
												G	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
													G	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
														A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
															C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
																C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
																	C	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
																		C	0	0	0	0	0	0	0	0	0	0	0	0	0	0
																			A	0	0	0	0	0	0	0	0	0	0	0	0	0
																				C	0	0	0	0	0	0	0	0	0	0	0	0
																					C	0	0	0	0	0	0	0	0	0	0	0
																						T	0	0	0	0	0	0	1	0	0	0
																							T	0	0	0	0	0	0	0	0	0
																								G	0	0	0	0	0	0	0	0
																									G	0	0	0	0	0	0	0
																										C	0	0	0	0	0	0
																											A	0	0	0	0	0
																												A	0	0	0	0
																													T	0	0	0
																														C	0	0
																															C	0
																																C

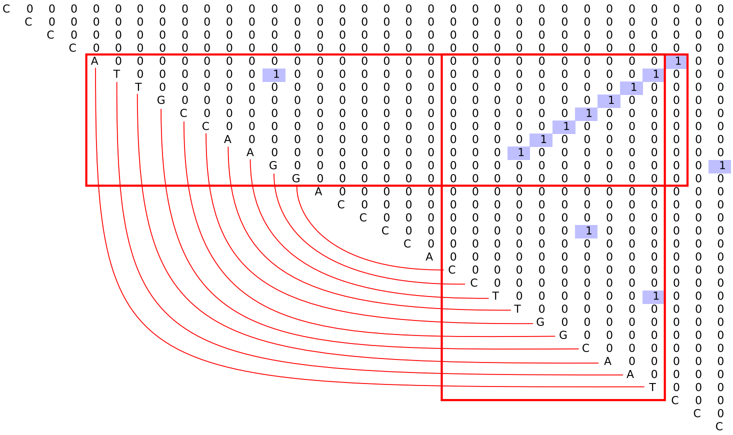
Example 1: Stem

CCCCATTGCCAAGGACCCCACCTTGGCAATCCC



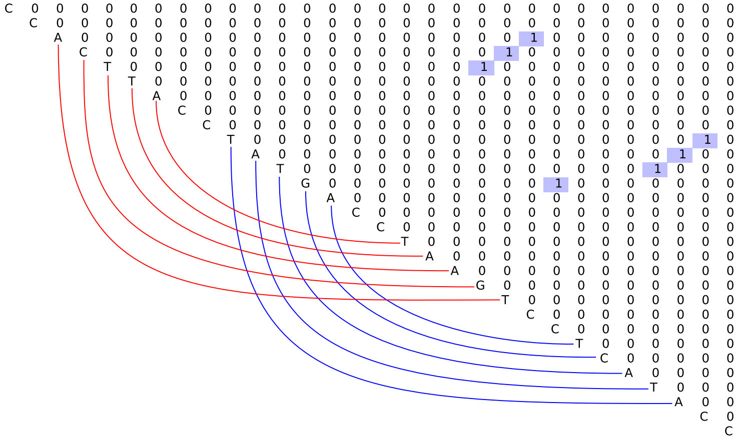
Example 1: Stem

CCCCATTGCCAAGGACCCCACCTTGGCAATCCC



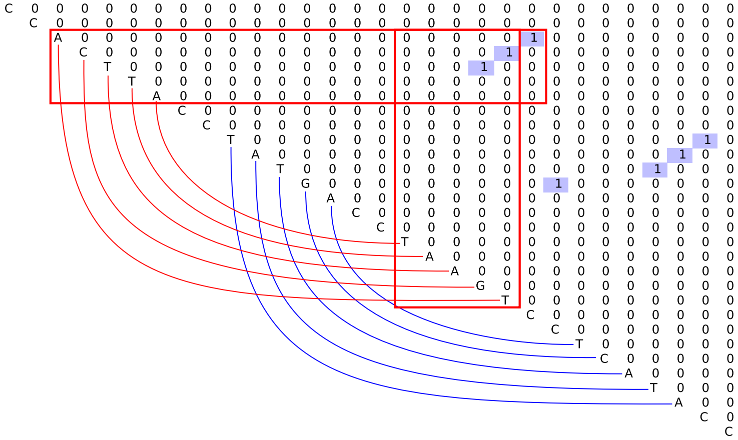
Example 2: Pseudoknot

CCACTTACCTATGACCTAAGTCCTCATACC



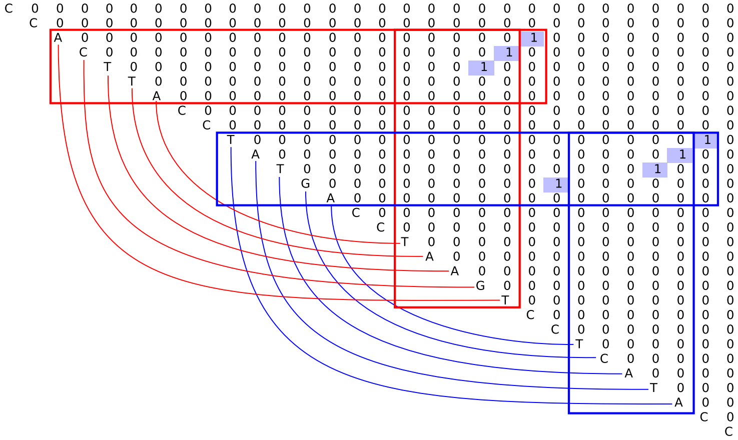
Example 2: Pseudoknot

CCACTTACCTATGACCTAAGTCCTCATACC



Example 2: Pseudoknot

CCACTTACCTATGACCTAAGTCCTCATACC

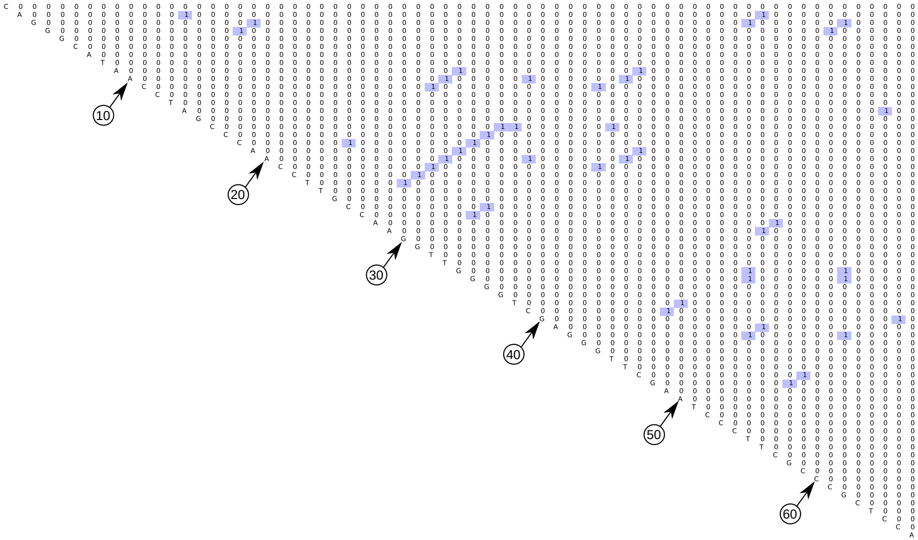


Example 3: real tRNA

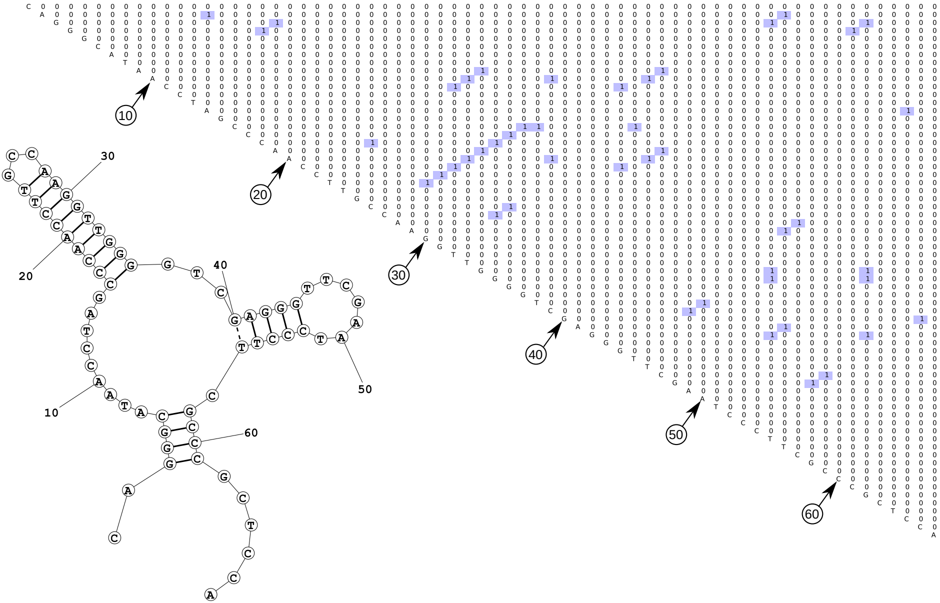
```
CAGGGCATAACCTAGCCCAACCTTGCCAAGG
TTGGGGTCGAGGGTTCGAATCCCTTCGCCCGCTCCA
```

- Novosphingobium aromaticivorans DSM 12444 chr.trna57-GlyGCC(268150-268084) Gly (GCC) 67 bp Sc: 22.9, from GtRNAdb
- Predicted secondary structures are given by using the Fold Web Server with default settings

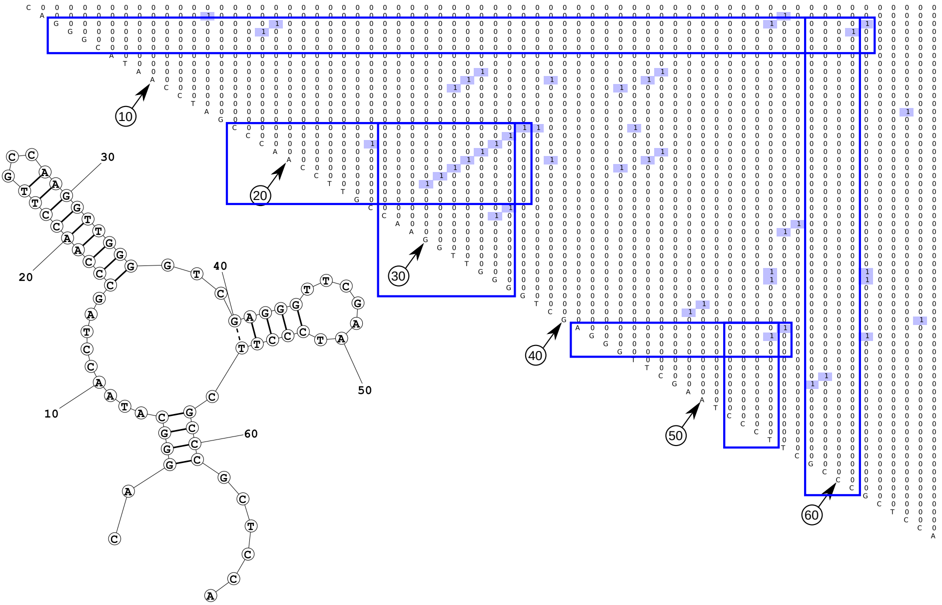
Example 3: real tRNA



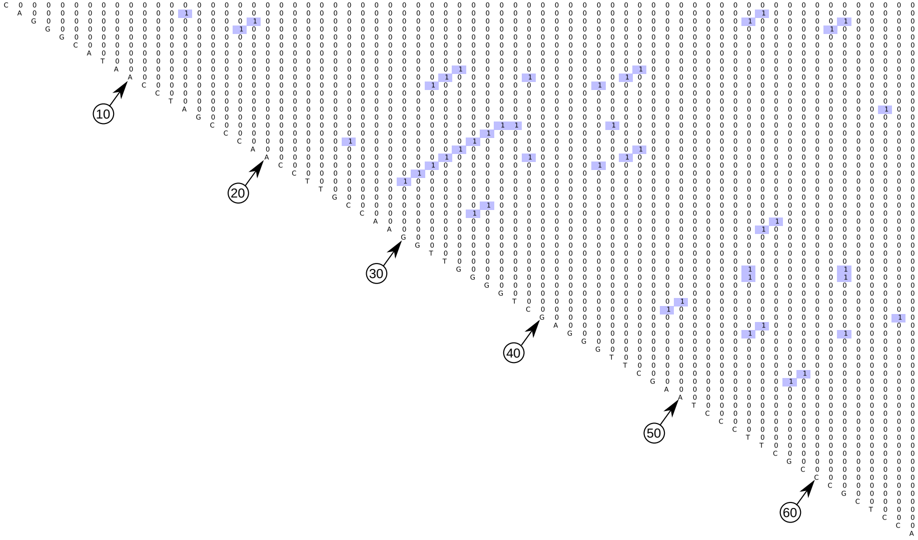
Example 3: real tRNA



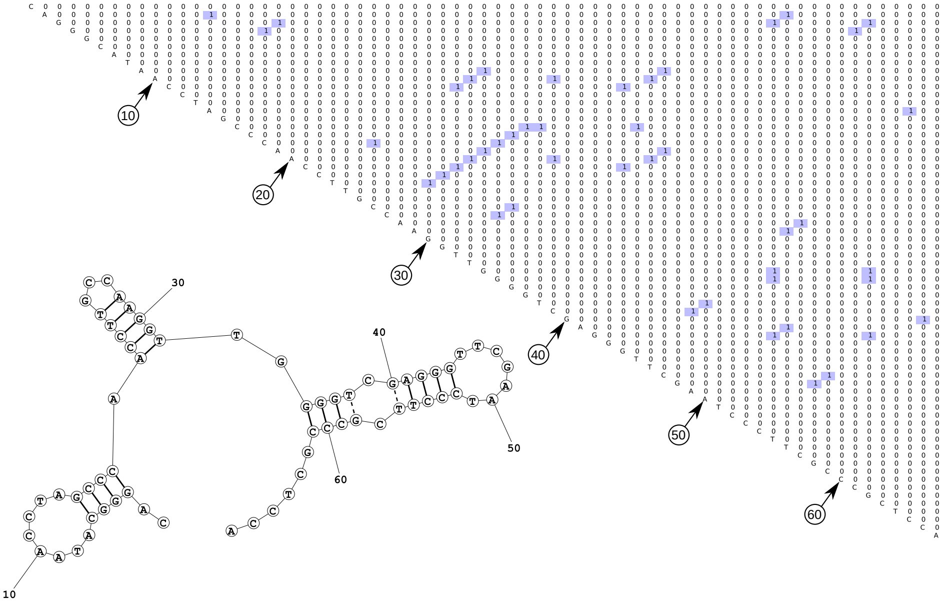
Example 3: real tRNA



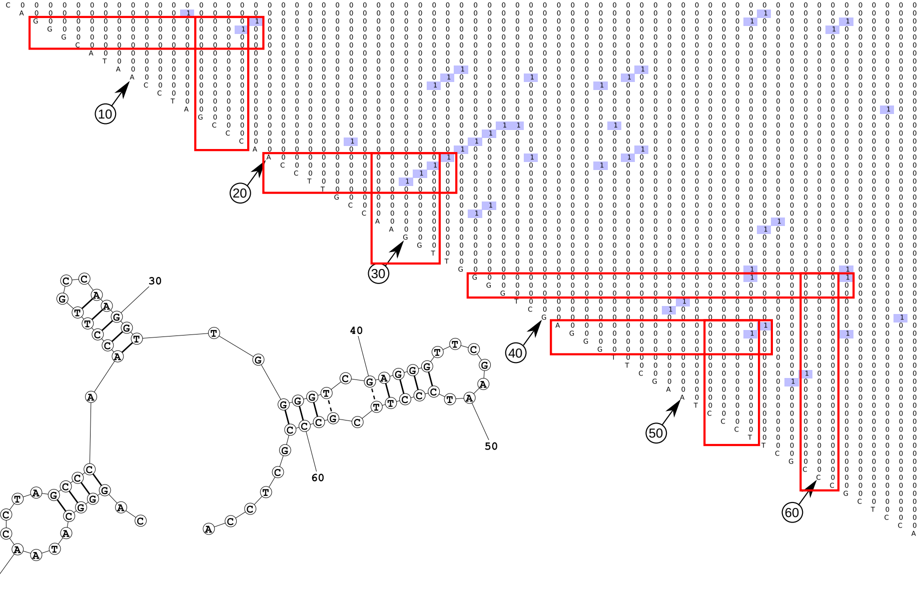
Example 3: real tRNA



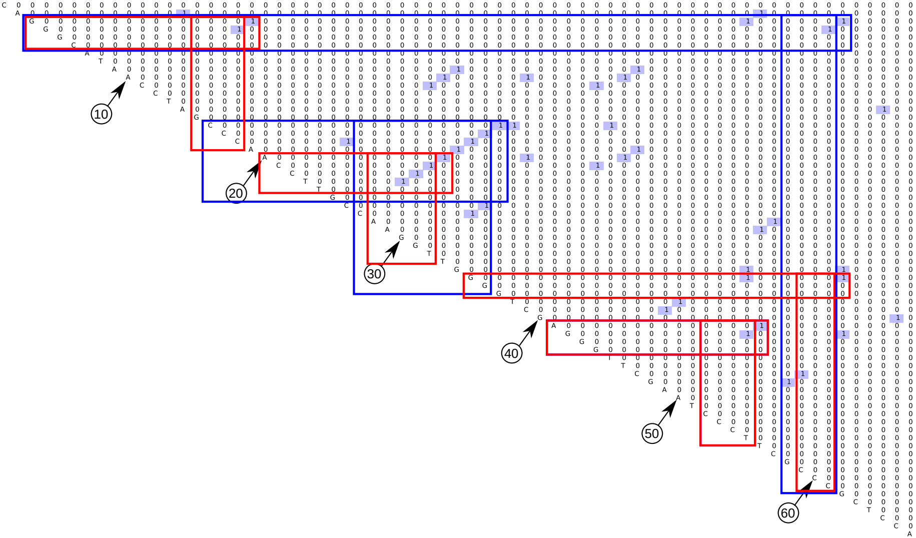
Example 3: real tRNA



Example 3: real tRNA



Example 3: real tRNA



Artificial Neural Networks

- Dense neural network
  - About 10 dense layers
  - `Relu` activation function
- Agressive dropout (up to 90% after each layer) and batch normalization (after each layer) for learning stabilization

Evaluation: 16s rRNA detection

- Training data
  - All sequences are 512 symbols in length
  - Totally up to 310000 sequences
  - Positive: random subsequences of 16s rRNA sequences from the Green Genes database
  - Negative: random subsequences of full genes from the NCBI database
- Validation set: up to 81000 sequences
- Accuracy is 90% after training

Evaluation: tRNA classification

- Training data: 50000 sequences from GtRNADB
- Input data normalization
  - Set the upper bound of sequence length to 220
  - First k symbols of the input are tRNA and the rest $220 - k$ symbols are filled by the special symbol
- Validation set: 217984 sequences for prokaryotes and 62656 sequences for eukaryotes from tRNADB-CE 3
- Accuracy is 97% after training
  - 3276 of eukaryotes (5.23% of all eukaryotes in the validation set) are classified as prokaryotes
  - 4373 of prokaryotes (2.01% of all prokaryotes in the validation set) are classified as eukaryotes

Future work

- Create DNN which does not requre input parsing
  - Create a training set of matrices using parsing
  - Train the network NN_1 which can handle vectorized matrices
  - Create network NN_2 by extending NN_1 with a set of layers which convert the sequence to input for NN_1
  - Train NN_2, weights of layers from NN_1 are fixed
- Try to use other types of neural networks: bitwise networks, convolutional networks
- Do more evaluation
- Perform comparison with other tools

Conclusion

- We propose the approach to handle secondary structure of sequences
  - Parser is only a features extractor
  - Parsing result contains all possible foldings (w.r.t. grammar) including impossible in practice
  - Grammar is a parameter: one can add a `G-T` pair, change minimal height of the stem, etc
  - It is possible to detect features which are not expressible in the language class in use
  - It is possible to use more expressive classes of formal languages
    - Conjunctive and boolean grammars
    - Multiple context-free grammars
- This approach can be applied for real data processing

Contact Information

- Semyon Grigorev:
  - s.v.grigoriev@spbu.ru
  - Semen.Grigorev@jetbrains.com
- Polina Lunina:
  - lunina_polina@mail.ru
- Trained models: https://github.com/YaccConstructor/YC.Bio

Thanks!