# RNA part

January 23, 2020

## 1. Introduction

The Nussinov algorithm is historically the first attempts at RNA secondary structure prediction. It is a dynamic programming algorithm that, given a RNA sequence, recursively finds the secondary structure that maximize the number of base pairs [1]. In the Nussinov algorithm we first initialize a scoring matrix, we decide a minimum loop size and then we start filling each cell by iterating through the diagonals of the matrix. We can give a score to a cell $i, j$ by looking at the three neighbour cells and the possible branching structures (bifurcation). If we consider $E(i, j)$ as the maximum number of base pairs for the subsuquence $x[i; j]$, than

$$E(i,j) = \max \begin{cases} E(i, j-1) \\ E(i+1, j) \\ E(i+1, j-1) + s(i,j) \\ \max_{i<k<j-1}\{E(i,k) + E(k+1, j)\} \end{cases} \tag{1}$$

[2], where $s(i, j)$ can be just 1 in case of base pair or can have different scores $(3, 2, 1)$ depending on the specific bases that form a pair in that cell. After the matrix is filled, the optimal RNA secondary structure is obtained by backtracking from the top right cell. The Nussinov algorithm is a really simplified version of RNA folding and it has several limitations. It has a computational complexity $O(N^3)$ and can not take into account pseudoknots, otherwise the computational complexity could reach $O(N^6)$. An other problem is the ambiguity because often the same structure can be procuded in several ways and the structures with the maximum number of base pairs are often not unique. The succeeding loop-based energy model uses loop decomposition for the calculation of the MFE. The free energy of the structure is obtained by summing the energy of each loop components,

$$\Delta G_{loop}(n) = \Delta G_{\text{size}}\ (n) + \Delta G_{\text{sequence}}\ + \Delta G_{\text{special}} \tag{2}$$

, in this way this method provide a much better prediction and an unambiguous solution to the folding problem using free energy minimization [3].

## 2. Materials and methods

**Nussinov implementation.** I used my implementation of the Nussinov algorithm that uses weighted scores, it is based on Giulia Corsi [4] implementation and on line guides I received in week 3 lecture exercise of Structural Bioinformatics. I start by generating a matrix (a list of list) that will be initialized with all zeros. Than I proceed by iterating through the diagonals of half of the matrix assigning a score to each cell.

[ ]:
```
# Iterate through the diagonals
for n in range(min_loop_size + 1, len(seq)):
```

```
        for j in range(n, len(seq)):
            i = j - n
            score_cell(seq, con, matrix_lst, i, j, min_loop_size)
```

The scoring of each cell is performed by considering the three neighbour cells and the branching structure with the highest score.

[ ]:
```
# The score list will store the scores of the four options for obtaining a cell
score_list = []
    bp_score = 0
    # Check if there is a base pair and attribute the relative score
    if (seq[i] == "A" and seq[j] == "U") or (seq[i] == "U" and seq[j] == "A"):
        bp_score = 2
    elif (seq[i] == "G" and seq[j] == "C") or (seq[i] == "C" and seq[j] == "G"):
        bp_score = 3
    elif (seq[i] == "G" and seq[j] == "U") or (seq[i] == "U" and seq[j] == "G"):
        bp_score = 1
    # Append the score of the diagonal cell plus the base pair score
    score_list.append(score_matrix[i+1][j-1] + bp_score)
    # Append the score of the left cell
    score_list.append(score_matrix[i][j-1])
    # Append the score of the bottom cell
    score_list.append(score_matrix[i+1][j])
    # Append the score of the branching structure with highest score
    k_scores = []
    for k in range(i,j - minimum_loop_size):
        score = score_matrix[i][k] + score_matrix[k+1][j]
        k_scores.append(score)
    score_list.append(max(k_scores))
    # Score the matrix with the highest score between the four options
    score_matrix[i][j] = max(score_list)
```

An other important part of the algorithm is the backtracking, it allows to obtain the optimal structure represented by a dot brackets notation string. I first initialize a dot bracket list filled with dots, than I used the backtracking function implemented by Giulia Corsi [4]. Giulia function use recursion and starting from the top right corner of the matrix, move to the cell from which the score at that position has been obtained. If the score derive from a bifurcation, the function call itself for obtaining the two optimal substructure. It is intersting to note that changing the order of the if statement often result in a different structure, that occurs because the Nusinov decomposition is ambiguous and there may be different structures with the same maximimum number of base pairs.

[ ]:
```
# Add stop criteria
while row < col + min_loop_size and score_matrix[row][col]!=0:
    # Check if the score of the cell is obtained from the diagonal (match)
```

```
    if (score_matrix[row][col] == score_matrix[row+1][col-1] + 2 and ((seq[row]␣
↪== "A" and seq[col] == "U") or (seq[row] == "U" and seq[col] == "A"))) or␣
↪(score_matrix[row][col] == score_matrix[row+1][col-1] + 3 and ((seq[row] ==␣
↪"G" and seq[col] == "C") or (seq[row] == "C" and seq[col] == "G"))) or␣
↪(score_matrix[row][col] == score_matrix[row+1][col-1] + 1 and ((seq[row] ==␣
↪"G" and seq[col] == "U") or (seq[row] == "U" and seq[col] == "G"))):
        dot_brackets_lst[col] = ")"
        dot_brackets_lst[row] = "("
        col -= 1
        row += 1
    # Check if the score of the cell is obtained from the bottom
    elif score_matrix[row][col] == score_matrix[row+1][col]:
        row += 1
    # Check if the score of the cell is obtained from the left
    elif score_matrix[row][col] == score_matrix[row][col-1]:
        col -= 1
    else:
        # Check for possible bifurcations
        for k in range(row, col - min_loop_size):
            if score_matrix[row][col] == score_matrix[row][k] +␣
↪score_matrix[k+1][col]:
                db_build(row, k, dot_brackets_lst, min_loop_size, seq,␣
↪score_matrix)
                db_build(k+1, col, dot_brackets_lst, min_loop_size, seq,␣
↪score_matrix)
                break
        else:
            raise RuntimeError("\n ERRORE not match with anything in Cell: (" +␣
↪str(row) + ", " + str(col) + ")")
        break
```

**Constraint implementation.** In order to obtain the constraint requested in the exercise I simply added three conditions in the scoring function, and I added the option *constraint* as an argument of the function. If the constraint is set to true and a constraint sequence is added, the function will look at each position of the constraint sequence. If it finds "(" and ")" at position $i, j$ it will only allows base pairs, if it finds an "x" at position $i$ it will not allows any base pairs, and finally if it finds a "." it will not give any restriction. Than the bifurcation score is added as usual.

[ ]:
```
# Check if there is a base pair: if yes force to have base pair
if (constraint[i] == "(" and constraint[j] == ")") or (constraint[j] == "(" and␣
↪constraint[i] == ")"):
    if (seq[i] == "A" and seq[j] == "U") or (seq[i] == "U" and seq[j] == "A"):
        bp_score = 2
    elif (seq[i] == "G" and seq[j] == "C") or (seq[i] == "C" and seq[j] == "G"):
        bp_score = 3
    elif (seq[i] == "G" and seq[j] == "U") or (seq[i] == "U" and seq[j] == "G"):
```

```
        bp_score = 1
    score_list.append(score_matrix[i+1][j-1] + bp_score)
# Check if the base pair is forbidden: if yes force to avoid base pair
elif (constraint[i] == "x" or constraint[j] == "x"):
    score_list.append(score_matrix[i][j-1])
    score_list.append(score_matrix[i+1][j])
# Check if there are not constraints: if yes don't use restrictions
elif (constraint[i] == "." and constraint[j] == "."):
    ...
```

If the constraint is set to False, a dot brackets sequence containing all dots will be generated and will be used as constraint sequence, therefore allowing all possible structures.

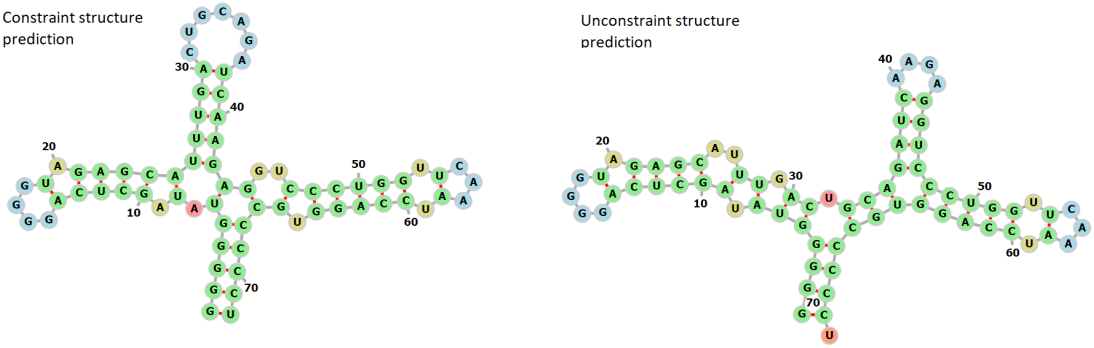
```
[ ]: def dinamic_programming_folding(seq, con, matrix_lst, min_loop_size, constrain␣
     ↪= False):
        if constrain == False:
            con = create_lst_db(seq)
```

## 3. Results

**Results of applying my implementation.** The base pair distance between the two dot-bracket strings I obtained is 29.

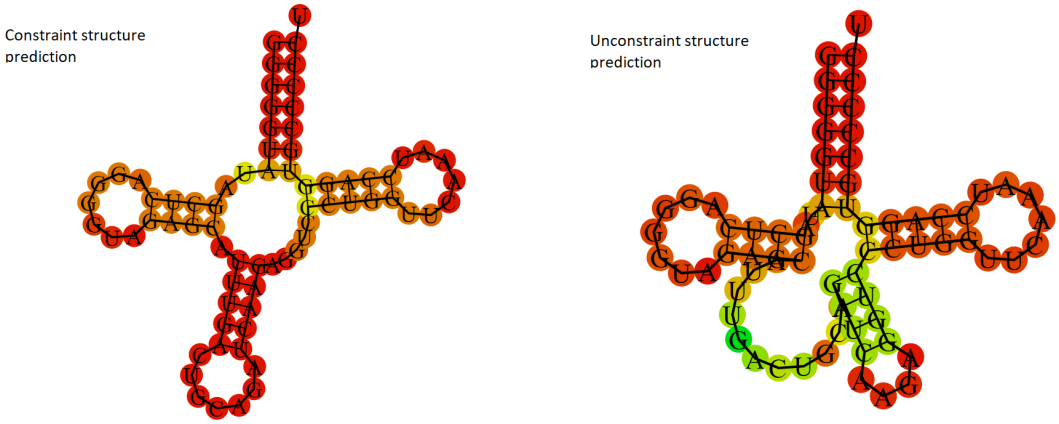
```
Min loop size: 3
Seq:            GGGGGUAUAGCUCAGGGGUAGAGCAUUUGACUGCAGAUCAAGAGGUCCCUGGUUCAAAUCCAGGUGCCCCCU
Sdb normal:     (((((((.((((((....).)))))..)).)).(((((((....)))))))(((.(...).))))))))).
Sdb constraint: ((((((.(.((((((....).))))))(((((.......))))))(..((((((.(...).)))))).)))))))
BP distance:  29
```

**Sketch of the two structures.** The structure obtained by the constraint structure prediction (on the left) resemble a tRNA structure but, especially in the area close to the central inner loop, it has an excessive amount of base pairs. The structure predicted with unconstraint prediction (on the right) doesn't resemble any known structure. The sketch of the structures are obtained with forna, it is a RNA secondary structure visualization tool provided by the University of Vienna [8].



**Annotation.** In order to annotate the sequence I performed a search in the Rfam database, which is a collection of RNA families, each represented by multiple sequence alignments and consensus secondary structures [5]. As expected the result of my search is that the sequence is a tRNA. The structure prediction compatible with the annotation is the constraint one.

**RNAfold webserver.** Running the RNAfold webserver [9] with both constraint and uncontraint structure prediction, it is possible to observe that the structure obtained by uncontraint prediction has a minimum free energy of -26 kcal/mol but it has an additional loop which is not found in tRNA structure. The base pair probabilities in the additional loop and in an adjacent loop are lower than the base pair probabilities found in the structure predicted with constraint, which has a minimum free energy of -24.5 kcal/mol.




**Energy folding model.** The MFE folding predict the secondary structure of an RNA sequence by minimizing its free energy, the algorithms based on this method attempt to find the structure of minimal free energy among all possible structures. As mentioned in the introduction, while the Nussinov algorithm attempt to obtain optimal structure by simply maximizing the number of base pairs, the succeeding loop-based energy model finds the minimum free energy optimal structure by distinguish which type of loop is closed by each pair [3]. In this more complex model, the secondary structure can be uniquely decomposed into loops (loop decomposition) and the total free energy of a structure become the sum over the energy of its constituent loops [1].

## 4. Conclusions

The loop-based enegy model provide a much better prediction than the Nussinov algorithm. In fact, as we observed in the results, often the base pairs maximization doesn't reflect biological structures. We also observed that the accuracy of the prediction can be increased forcing the algorithm to produce a known substructure in a specific location. One useful method for analyzing the possible alternative structures of a given sequence is the partition function calculation for RNA secondary structure. Basically the partition function prediction will not consider only the MFE structure but it will provide the base pair probabilities considering all possible structures. Other advanced methods, like the energy-based RNA consensus secondary structure prediction (e.g. RNAalifold program), extend the RNA structure prediction algorithm based on the loop-based model. These methods take into account the energy contributions of all sequences in the alignments and add phylogenetic information into the energy model [6]. Finally there are also methods the simultaneously align and predict the secondary structure of multiple RNA sequences

by free energy minimization. These methods, based on the Sankof algorithm, have the advantage to increase the accuracy of the prediction but the disadvantage is that they require an increased computational memory. Therefore, in order to decrease the computational complexity, all the implementations that use these methods are heuristic solutions [7].

## References


[1] Hofacker, I. L., Stadler, P. F., & Stadler, P. F. (2006). RNA Secondary Structures. Encyclopedia of Molecular Cell Biology and Molecular Medicine.

[2] Gorodkin Jan; Hofacker, Ivo L.; Ruzzo, Walter L. (2014) Concepts and introduction to RNA bioinformatics. Methods in molecular biology, 1097:1-31.

[3] Hofacker IL. (2014) Energy-directed RNA structure prediction. Methods Mol Biol. 1097:71-84.

[4] Giulia Corsi. PhD fellow Animal Genetics, Bioinformatics and Breeding. University of Copenhagen.

[5] Sam Griffiths-Jones, Alex Bateman, Mhairi Marshall, Ajay Khanna and Sean R. Eddy. (2019) Rfam: an RNA family database.

[6] Washietl S1, Bernhart SH, Kellis M. (2014) Energy-based RNA consensus secondary structure prediction in multiple sequence alignments. Methods Mol Biol. 1097:125-41.

[7] Havgaard JH1, Gorodkin J. (2014) RNA structural alignments, part I: Sankoff-based approaches for structural alignments. Methods Mol Biol. 1097:275-90.

[8] Kerpedjiev P, Hammer S, Hofacker IL. (2015) Forna (force-directed RNA): Simple and effective online RNA secondary structure diagrams. Bioinformatics 31(20):3377-9.

[9] Ivo L. Hofacker. (2003) Vienna RNA secondary structure server. Nucleic Acids Res. 31(13): 3429–3431.