# RNA structure prediction using Nussinov's algorithm with Four Russians speedup

- **Group Members:** Chan Tong Kou, Yves Chan, Andrew Taeyeon Kim, Pranav Garg
- **Public to Classmates:** yes
- **Project Type:** Take a paper, implement it and run it on some interesting data

RNA structure prediction is a problem in computational biology where given a sequence of nucleotides, we aim to predict the configuration that maximises the number of favourable base pairings. In 1978, Nussinov *et al.* [1] published a dynamic programming algorithm that solves this problem in $O(n^3)$ time. Subsequently, in 2014, Venkatachalam *et al.* [2] published a variant that uses the Four Russians technique to speed up this calculation to $O(n^3/log(n))$. They also published parallel versions of these algorithms.

In this project, we shall implement some of these algorithms. We will use our programs to predict structures of well known RNA molecules like ribosomal RNA and tRNAs and compare the obtained structures to their known structures. This will also be used to compare run-times of the two algorithms.

In addition, we would like to investigate whether the scaffold sequence of the sgRNA's used in Crispr-Cas9 experiments is compatible with most target sequences. This would also give us a tool to check whether a target sequence is "good" depending on how much it interrupts the native structure of the scaffold sequence.

## 1 Nussinov's algorithm

This is a dynamic programming algorithm that looks for the best folded configuration of an RNA sequence using the structure of its subsequences as subproblems. Since base pairings are (almost) always nested, an extension of one base on either side involves either preserving the original structure and adding the new bases on either side (which may or may not be complementary) or splitting the existing structure into two (smaller) new parts, each containing one of the newly added bases. Thus, by solving the structure of small sequences, we can build up the full optimal structure. More formally, let $D(i, j)$ be the maximum number of pairings in the subsequence starting at $i$ and ending at $j$. Let b be a boolean function that returns 1 if it's input nucleotides are matched and 0 otherwise.

- Problem: Maximize the number of non-crossing complementary base pair matchings in an RNA sequence.

[1] Nussinov, Ruth, George Pieczenik, Jerrold R. Griggs, and Daniel J. Kleitman. "Algorithms for loop matchings." SIAM Journal on Applied mathematics 35, no. 1 (1978): 68-82.

[2] Venkatachalam, Balaji, Dan Gusfield, and Yelena Frid. "Faster algorithms for RNA-folding using the Four-Russians method." Algorithms for Molecular Biology 9, no. 1 (2014): 5.

- Initialization: $D(i,j) = 0$, for $|i-j| < 4$
- Recurrence:

$$D(i,j) = \max\left[\{b(S(i),S(j)) + D(i+1,j-1)\}, \max_{i+1 \leq k \leq j}\{D(i+1,k-1), D(k,j)\}\right] \quad (1)$$

- Termination: Maximum number of pairings in the full sequence is $D(1,N)$.

## 2 The Two Vector Method

Dynamic programming algorithms typically proceed by filling out the answers to problems of increasing size in a table. The Four Russians Method is a tactic used to speed up calculations involved in filling up the table by pre-computing and storing answers to all possible pieces of the table. If we imagine the table as being made up of blocks of some size, each block is filled up using values stored in a boundary layer of cells, say top row and left column. If we can pre-compute what block values correspond to each set of boundary values we are able to speed up the table-filling process (provided we choose the block size appropriately).

This idea is ported over to our problem by speeding up the calculation of the second term in the recurrence (1) by breaking the $i^{\text{th}}$ row and $j^{\text{th}}$ column into vectors. If we pre-computed the answer for each possible row vector with each possible column vector, we could work vector by vector instead of cell by cell. This longer stride length gives a speed boost. To reduce the size of the lookup table though, we work with difference vectors which store the difference between consecutive cells. This condenses many possible vectors into one entry in the table. Thus, we write the values of cells in a row vector of length $q$ starting at $(i,l)$ as

$$D(i,l) + V_p,\ p \in \{0,...,q-1\} \text{ where } V_p = \sum_{i=0}^{p} v_i$$

and $v$ is the difference vector in our lookup table. It is binary because entries in a row can only increase by either 1 or 0. The same works for column difference vectors except that they contain either 0 or -1.

## 3 Parallelization

Nussinov's algorithm calculates the 2D table one cell at a time, looking up $\Theta(n)$ cells to fill each of the $\Theta(n^2)$ cells which makes the time complexity $\Theta(n^3)$. But not every subsequent calculation depends on the ones before it. In fact, the entries on a particular diagonal can be filled independently.

With a concurrent-read concurrent-write parallel random access memory machine, the table can be accessed at the same time by multiple processes. With parallelization, each diagonal line can be filled up in one time step with one process allocated to each column. These $n$ processes fill the bottom most cell of their column at the same time. Since there are n diagonal lines, and each iteration requires n evaluations, time complexity is reduced to $O(n^2)$.

The same scheme works for the Two Vector method.

# 4 Goals:

## 4.1 Programming

The first step is to write programs to implement Nussinov's algorithm. This will be done by Yves Chan and Andrew Taeyeon Kim. We will also implement the improved version of this using the Four Russians method. This will be done by Chan Tong Kou and Pranav Garg. We plan to divide the work by individually writing self-contained functions that can be called from the main routine.

We will use a standard output format such as BPSEQ so that we can use existing tools like RNA viewer to visualise the output.

## 4.2 Testing

Once we have working programs, we will test them on a few sequences that have known structure: specifically, about ten well known transfer RNA sequences from E. coli and Homo sapiens. Then we will test it on about four sequences of rRNAs present in the small and large subunits of ribosomes in E. coli and Homo sapiens. Known structure data will be obtained from RNA STRAND.

During this basic testing, we will examine whether the run time of these two programs is significantly different, and how that changes between the small tRNA sequences compared to larger rRNA sequences.

## 4.3 Predicting Poor Crispr Targets

Crispr-Cas9 is a gene editing technique where an endonuclease Cas9 makes reliable cuts at a predefined location in a genome using a target sequence as reference. An RNA molecule is responsible for bringing the Cas9 molecule to the correct location. This sgRNA consists of a 20 nucleotide target sequence (ending in GG) which recognises the target via base pairing and a scaffold sequence of about 82 nucleotides which the Cas9 binds to.

For a well functioning sgRNA sequence, the target sequence should not disrupt the structure of the scaffold sequence because doing so would hinder its binding to the Cas9 protein. Using the standard scaffold sequence [3] we will search for target sequences that do disrupt its structure, and the common traits of such sequences. This can also be used to check whether a given target sequence is "good".

## 4.4 Extended Goal

If time permits, we would like to implement parallelized versions of these two algorithms and compare the runtimes.

[3] `GTTTTAGAGCTAGAAATAGCAAGTTAAAATAAGGCTAGTCCGTTATCAACTTGAAAAAGTGGCACCGAGTCGGTGCTTTTTTT`, from Mali, Prashant, Luhan Yang, Kevin M. Esvelt, John Aach, Marc Guell, James E. DiCarlo, Julie E. Norville, and George M. Church. "RNA-guided human genome engineering via Cas9." Science 339, no. 6121 (2013): 823-826.