# RNA Secondary Structure Prediction

Fernando Javier López Cerezo

November 18, 2021

## 1 Objectives

The objective of this project is to estimate the complexity of an algorithm which given a RNA molecule represented as a string where each character represents a nucleotide (A, C, G, U) finds a set of compatible base pairs of maximum cardinality. This set should also respect any structural contraint that may exist. In particular we want to find the average complexity of an algorithm implementing a dynamic programming approach.

## 2 Experimental Setup

For this project we used:

- MacBook Air 13 (1,8 GHz Dual-Core Intel Core i5 CPU, 8 GB 1600 MHz DDR3 RAM and macOS Catalina 10.15.5 OS)
- Java 15.0.2 and R 4.1.1

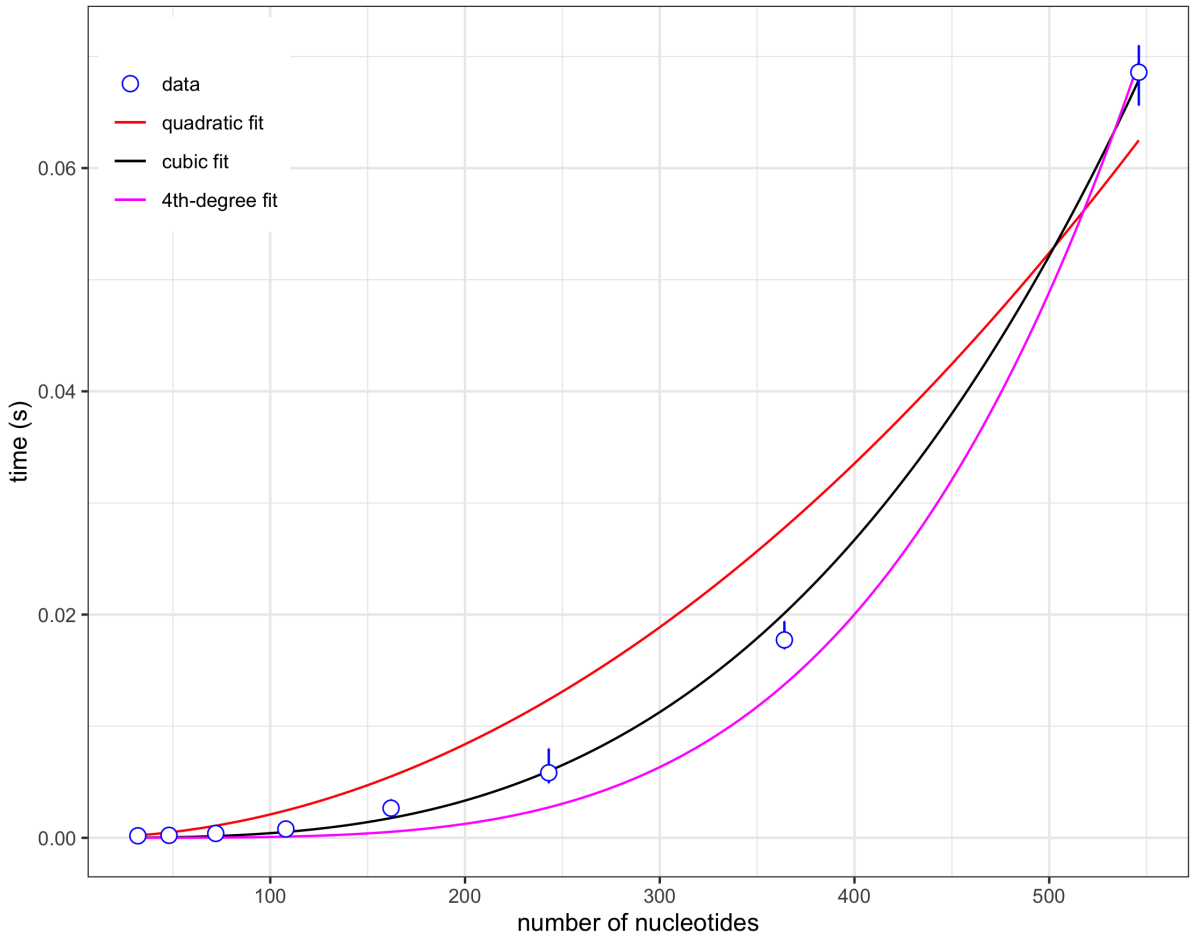
The method we used to obtain our data was the following:
We implemented an algorithm using a dynamic programming approach using Java. This algorithm took as input parameters the RNA sequence and the minimal stem-loop (L). It then created 3 matrices, a matrix M used to store costs (where M[i][j] is the maximum number of base pairs in nucleotide i to nucleotide j), a matrix D to store decisions and a matrix B to store which base pairs match. B was precomputed at the beginning of the algorithm. Matrix M was then filled using the following Bellman equation:

$$\mathrm{M}_{i,j} = \begin{cases} 0 & if \quad j-i \leq L \\ \max(M_{i,j-1}, \max_{\substack{pair(s_k,s_j) \\ i \leq k < j-L}}(M_{i,k-1} + M_{k+1,j-1} + 1)) & if \quad j-i > L \end{cases}$$

We then implemented a recursive algorithm which was used to reconstruct an optimal solution and return it in dot-bracket notation. Finally we used an algorithm to determine the time required to find the optimal folding. We used RNA sequences with an initial length of 32 and a size increment of 8, each size was tested with 100 RNA sequences and a minimum loop size of 5. At last all of this data was anaylzed using R and studied.

# 3 Empirical Results

Using R we plotted the following graph showing how the time the algorithm takes to be executed varies with the number of nucleotides in the RNA sequence.




R approximated our data to a quadratic fit, a cubic fit and a 4th-degree fit. Simply put, if we call our independent variable (number of nucleotides) $n$, it plotted the functions of the type $an^2$ (quadratic), $an^3$(cubic) and $an^4$ (4th-degree) that best fitted our data. R also gave us estimated values for this parameters. If our data was to follow a quadratic pattern of growth the closest polynomial to it would be $2.096^{-7}n^2$. If, on the other hand, our algorithm followed a cubic pattern of growth that estimated cubic funcion would be $-2.385^{-3}n^3$. Lastly if our data was to follow a 4th-degree pattern of growth the closest polynomial to it would be $7.822^{-13}n^4$. In order to estimate the order of growth of our algorithm we can actually discard the constants ($a$) as it won't affect its order of growth (which is what we are looking for).

# 4 Discussion

We are interested in estimating the order of growth of the computational cost of our algorithm when the number of nucleotides in the RNA sequence becomes arbitararily large. Lets call this order of growth $t(n)$ where $n$ is the number of nucleotides.

In the above graph we see the pattern that an algorithm with a quadratic order of growth would follow (red line). This algorithm would have an order $O(n^2)$. Taking a look at the pattern that our algorithm follows we can clearly see that the time taken for our algorithm to be executed is smaller. In other words, $O(n^2)$ bounds $t(n)$ from above for arbitrarily large $n$. Therefore $t(n) \in O(n^2)$.

If we now look at how an algorithm with a 4th-degree order of growth ($O(n^4)$) would behave and compare it to our data we can see that our algorithm is bounded by $O(n^4)$ from below for arbitrarily large n. Therefore $t(n) \in \Omega(n^2)$. Finally If we now look at how an algorithm with a cubic order of growth ($O(n^3)$) would behave and compare it to our data we can see that our algorithm behaves in a very similar way for all values of $n$. We can therefore conclude that the order of growth of our algorithm is cubic, $t(n) \in \Theta(n^3)$.

It should be taken into account that this is the complexity of our algorithm in the average case. Normally we would be interested in finding the complexity in the worst case which in this case would also be $O(n^3)$. In this case we are also exclusively studying the time complexity but the space complexity should also be taken into account. For this algorithm the space complexity would be $\Theta(n^2)$

In conclusion the average time complexity of RNA secondary structure prediction problem using dynamic programming is $O(n^3)$.