Final Project: Nussinov Implementation

Alice Huang

Introduction

RNA is primarily responsible for the creation of proteins via translation [1]. Because of this function, analyzing the functionality/structure of RNA is a very important and interesting problem in bioinformatics. RNA structure is comprised of the following: primary, secondary, and tertiary [2]. In this report, we will be focusing on the secondary structure of RNA (the 2D analysis of bonds between different parts of RNA).

Secondary RNA structures “impart catalytic, ligand binding, and scaffolding functions to a wide array of RNAs, forming a critical node of biological regulation,”, thus “the study of RNA secondary structure is critical to understanding the function and regulation of RNA transcripts” [3].

Now that we have established the importance of understanding secondary RNA structure, we will be discussing the core of this project: Nussinov’s Algorithm. Nussinov is a dynamic programming algorithm that predicts the secondary folding of an RNA structure given its primary sequence. Below is the algorithm that we went over in class that we hope to implement in this project [4]:

$$s[i,j] = \max \begin{cases} 0, & \text{if } i \geq j, \\ s[i+1,j-1]+1, & \text{if } i < j \text{ and } (v_i, v_j) \in \Gamma, \quad \textbf{(1)} \\ s[i+1,j-1], & \text{if } i < j \text{ and } (v_i, v_j) \notin \Gamma, \quad \textbf{(1*)} \\ s[i+1,j], & \text{if } i < j, \quad \textbf{(2)} \\ s[i,j-1], & \text{if } i < j, \quad \textbf{(3)} \\ \max_{i<k<j}\{s[i,k]+s[k+1,j]\}, & \text{if } i < j, \quad \textbf{(4)} \end{cases}$$

Figure 1

s[i,j] is the maximum number of base pairs in a sequence from index i to j. The final answer to this algorithm that we are interested in (maximum number of base pairs in the whole sequence) will be s[0, n] where n is the length of the sequence. Note that Nussinov ignores pseudoknots, so we will only be analyzing sequences that do not contain them. The algorithm's recursive cases correspond to the following [4]:

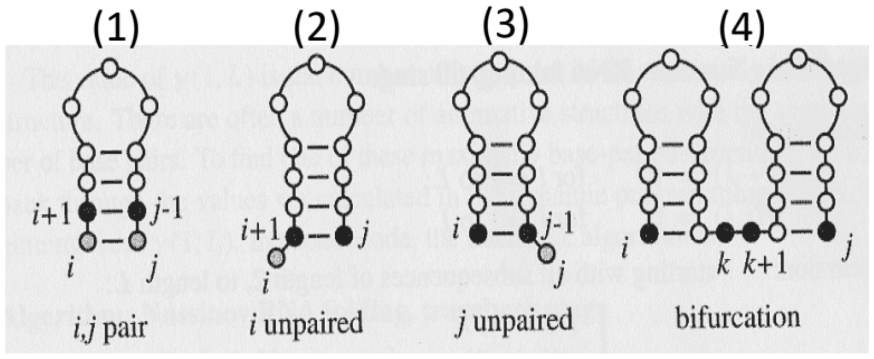


Figure 2

The most interesting part of the recursive definition is the fact that it accounts for bifurcations (case 4), which adds an extra dimension to make the runtime of the algorithm O(n^3). The bifurcation step splits the sequence[i, j] into two parts and then performs the algorithm again on those two parts, checking for cases that would not be accounted for if the bifurcation step did not exist.

In this project, we aim to implement Nussinov's Algorithm in code, something we never did in class. The goal is to fill out the dynamic programming table accordingly and then perform a traceback to find the secondary structure of the sequence. Note that a traceback can yield multiple results, so the order in which we check the recursive cases for the backtrace matters since each order can lead to a different backtrace. The following image depicts the multiple possible secondary structures we can predict while performing a backtrace for a sequence's filled Nussinov table [4]:

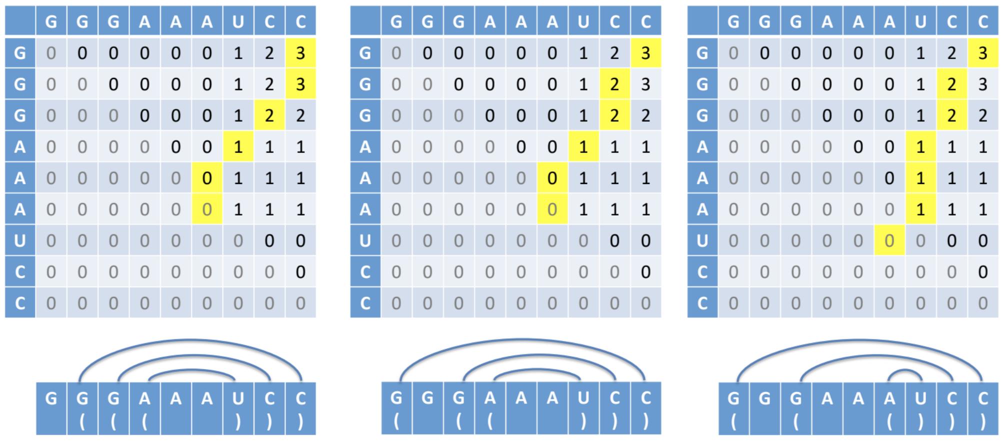


Figure 3

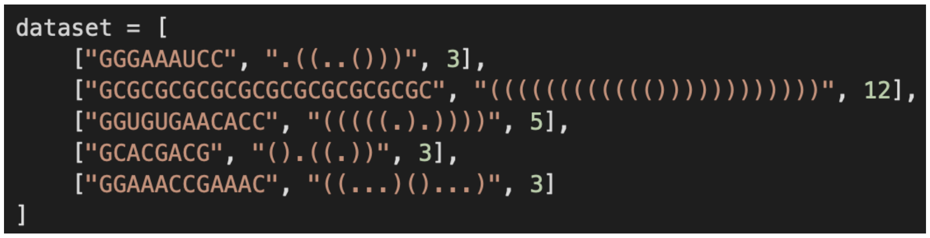
Methods

Google Colab

We put our python implementation in a Google Colab notebook. We chose to do this since previous programming assignments have been completed using Google Colab, so we could use those assignments as inspiration for how we want to format our code and testing. For example, we also use nose.tools to test our implementation.

Dataset

The dataset we use to test our implementation was obtained from RCSB Protein Data Bank [5]. The dataset is essentially a list, where each entry of the list contains three strings. The first string is the RNA sequence, the second string is the dot-parentheses structure that corresponds to its secondary structure, and the last string is the number of base pairs in the secondary structure (equal to the number of pairs of parentheses). Below is what the dataset looks like in our Colab notebook:



```
dataset = [
    ["GGGAAAUCC", ".((..()))", 3],
    ["GCGCGCGCGCGCGCGCGCGCGCGC", "(((((((((((())))))))))))", 12],
    ["GGUGUGAACACC", "(((((.).))))", 5],
    ["GCACGACG", "().((.))", 3],
    ["GGAAACCGAAAC", "((...)()...)", 3]
]
```

Figure 4

Note that this dataset is not large, but we are confident in its ability to evaluate our implementation's correctness. The dataset is able to evaluate the cases that are handled by Nussinov, such as bifurcation for the fourth and fifth entries. It also does not contain sequences that have pseudoknots.

As we mentioned in the introduction section, the Nussinov backtrace can result in multiple solutions, so the order in which we check the different recursive cases matters. We take this into account when finding the right dot-parentheses notation to use.

For certain sequences in this dataset, we also check the complete dynamic programming table returned from Nussinov and compare it to the known tables to make sure our algorithm is filling out the tables correctly. We do this for the first and fourth sequences since we went over both of those sequences in class and looked at their complete tables during Lecture 10 [4].

Helper Functions

Before we discuss the main Nussinov algorithm functions we implemented, we will go over some of the helper functions we created. The first is called is_base_pair, which takes in a tuple of two values and returns True if the two values form a base pair, and False otherwise. We call this helper function in our fill_nussinov function while checking for case 1 (see Figures 1 and 2).

The second function we have is convert_pairs, which takes in a sequence and an array of tuples as parameters. convert_pairs takes these inputs and outputs the dot-parentheses notation of the sequence. Note that the array input contains tuples that are comprised of two indices in the sequence that indicate those two indices contain parentheses. For example, if our sequence was "AGGU" and the array was [(0,3)], the output would be "(..)".

Main Functions

Now we will go over the two main functions that perform Nussinov's algorithm. The first function is called fill_nussinov which just takes a sequence as input. It will then fill a n x n table (where n = length of the sequence) using Nussinov's algorithm in $O(n^3)$ time and $O(n^2)$ space.

This algorithm follows the recursive definition of Figure 1. fill_nussinov returns the filled dynamic programming table and the final answer to Nussinov, s[0, n].

The next function is traceback, which takes in the filled-out table, the sequence, an empty array, and i and j indices as inputs. This function is recursive and performs a backtrace in the filled-out table, keeping track of where parentheses should go in the dot-parentheses notation by appending (i,j) to the empty array. The function then returns this array, which will be the array we pass to convert_pairs later to see the final dot-parentheses notation of the sequence.

Testing and Results

We test our python implementation using the dataset we mentioned earlier. For the sequences “GGGAAAUCC” and “GCACGACG”, we perform more extensive tests. We check if the final answer is correct, if the final dot-parentheses notation is correct, and if the final table is equal to the final table we know to be true from our lecture slides [4]. This is because we want to make sure our fill_nussinov function is filling out the dynamic programming table correctly. We only check the tables for these two sequences because we know what the final table should look like for those two.

After checking those specific sequences, we loop through the rest of the dataset and check if the final answer is correct and if the dot-parentheses sequence is correct. We considered our implementation as “finished” only when everything mentioned so far in this section was passing and did not throw errors.

Originally, we had trouble finding the correct dot-parentheses notation for some of our sequences. For example, “GGUGUGAACACC” can have both “(((((.).))))” and “(((((..)))))” as its final dot-parenthese notation. We specifically chose to put “(((((.).))))” as the final answer in our dataset to check for while testing because of the order we decided to check for the optimal backtrace.

Future Work

Like we just mentioned in the previous section, we had trouble deciding which dot-parentheses notation to check for while testing, since many sequences can have multiple optimal backtraces. In the future, we would want to choose a dataset that contains all possible dot-parentheses notations for each sequence, so during testing we could just check if one of those notations was returned from our implementation.

Note that our dataset also only has five sequences. We believe that this is enough because of the extensive testing done on two sequences and also because those five sequences capture all the cases of Nussinov's. In the future, we would want to get more sequences to test to make sure that our implementation is working as expected.

In the future, we could also have more visualization functionality, where we can display the final table and corresponding backtrace in a more visually-appealing manner, similar to what Figure 3.

Conclusion

Nussinov's algorithm is an important algorithm that assists in predicting RNA secondary structure. While working on our own python implementation, we discovered that Nussinov might lead to some trouble when it comes to having one deterministic answer for the final structure. The answer for the final structure depends on the order in which we check the different recursive cases in our backtrace, which is something that we did not account for while writing our proposal.

However, we learned a lot while completing this project and hope to add to it with the items we brought up in the "Future Work" section. Like we mentioned in our introduction, the prediction of secondary RNA structure is an important and interesting problem in bioinformatics, and through this project we learned how to actually implement an algorithm that completes that task through programming.

Citations

1. “Biochemistry, RNA Structure - StatPearls - NCBI Bookshelf.” 2022. NCBI. https://www.ncbi.nlm.nih.gov/books/NBK558999/.

2. Mittal, Aditya. 2021. “Optimal Secondary RNA Alignment and Nussinov's Algorithm | by Aditya Mittal | Medium.” Aditya Mittal. https://adityamittal307.medium.com/optimal-secondary-rna-alignment-and-nussinovs-algorithm-c352b3445adf.

3. Vandivier LE, Anderson SJ, Foley SW, Gregory BD. The Conservation and Function of RNA Secondary Structure in Plants. Annu Rev Plant Biol. 2016 Apr 29;67:463-88. doi: 10.1146/annurev-arplant-043015-111754. Epub 2016 Feb 8. PMID: 26865341; PMCID: PMC5125251.

4. “CS 466: Introduction to Bioinformatics · Mohammed El-Kebir.” n.d. Mohammed El-Kebir. Accessed December 13, 2022. https://www.el-kebir.net/teaching/CS466/Fall_2022/CS466.html.

5. RCSB PDB: Homepage. Accessed December 13, 2022. https://www.rcsb.org.