Number of Perfect Matchings in RNA Sequence

Introducing the problem, and relevance

RNA (RiboNucleic Acid) is a structure of genetic code, similar to DNA (DeoxyriboNucleic Acid). However, notable differences include the 'thymine' nucleotide in DNA being replaced by 'uracil' in RNA. RNA tends to be found as a single stranded molecule. RNA is found in prokaryotic, eukaryotic, and viral cells. A notable example is mRNA, which is a single strand of complementary bases to a region of DNA used in protein synthesis.

RNA has many functions within any organism. In humans, our basic genetic code is stored in the form of DNA; but the DNA is processed downstream in the form of RNA to be converted to a protein.

When the nucleotide sequence of two intervals within the RNA is complementary, the two regions can bind together to create hairpin loops which alter the secondary structure of the RNA. The site of the hairpin loop can also alter depending on the current point in time, which increases the variety of secondary structures. Due to the vast number of potential structures the RNA molecule can possess, it is important to filter out the secondary structures that cannot be adopted by the RNA strand. The first part of answering such a question is finding the maximum number of perfect matchings possible for an RNA sequence.

This task assumes two things. The first is that the number of complementary nucleotides (A-U, and G-C) are even. If they are uneven, a perfect matching is not possible. Secondly, it assumes that every nucleotide within the sequence contributes to bonding; there are no nucleotides which do not find a nucleotide to bond to.

Solution

A bonding graph is where the individual bases of the RNA sequence are plotted in an ordered circle, with **adjacency edges** between the bases. However, complementary bases in different positions of the single stranded RNA molecule can interact and bind together to alternate the 3D structure of the molecule. These interactions can be modelled on this bonding graph by the use of dotted **basepair edges**. There are many different permutations in which the nucleotides can bind to each other, and the maximum value is what is calculated here.

In graph theory, a potential method of solving matching problems in a bipartite graph is by forming an alternating sequence, where you alternate between two nodes which are connected, and two which are not. E.g, A-G=B-D=R, where '-' means 'not connected', and '=' means 'connected'. However, this solution poses issues for the current task. It is designed to process **maximal matching** of bipartite graphs, an emphasis on matchings where a node has limited possible connections to nodes in the adjacent set. This is not the case for us because the two sets only include complementary bases. Secondly, this algorithm is designed for when a template already exists, where two or more vertices are connected between the two sets. This is less applicable when no template is found, which is the case when a novel RNA sequence is being analysed.

The solution I have decided to follow is to convert the bonding graph into a bipartite graph. To achieve this, the pyrimidines are grouped into set-1, whereas the purines are grouped into set-2.

Now, we can calculate the number of perfect matching permutations possible. However, how can this be done?

One of the potential methods is to assign each element of each set a unique identifier, and find the number of different combinations in which it can be joined to the other set. However, programmatically, this is a tricky solution to implement.

Instead, as I played around with different sequences I noticed that we only need to play around with the order of one of the sets (set-2 for example); no need to re-order both sets as that would only cause duplicated matchings.

Now knowing that to find the number of permutations, only the order needs to be modified (no need for special identifiers), and only one set of the two needs to be re-ordered, makes a solution much easier to visualise.

So how can we find the number of different permutations in a list? The answer is the use of factorials. A factorial is denoted by the exclamation sign ( ! ), and indicates a recursive sequence of multiplication up to, and including, the integer before it. For example, 5! Is 5x4x3x2x1 (=120). Although the order of multiplication does not matter, the theory behind the descending order is important to understand. If we have 5 unique cards, and we must place them down one-by-one, the first position has 5 different cards that can be placed into that position. We now have n-1 cards available (where n is the number of cards we have in total, 5). So for position two, we have 4 different cards available. This continues until we no longer have any cards left. There are 5! number of different arrangements possible for these 5 unique cards.

Due to the unique property of RNA (and DNA) due to base pairing of complementary nucleotides means to incorporate factorials into the solution is straightforward. We know that adenine forms a two hydrogen bonds with uracil, whereas guanine forms a triple hydrogen bond with cytosine. This means we can form two separate bipartite graphs – one for adenine-uracil, and another for guanine-cytosine. The base pairs can be separated in the bipartite graph where the adenine bases are in set-1, and uracil is in set-2. The number of bases from either adenine, or uracil are calculated (since these two values must be the same for perfect matching), and the value is returned into a factorial algorithm. This process is repeated for guanine-cytosine before multiplying the two factorials together to give the total number of matching permutations possible.

A simplification of this way to solve the question ignores that individual nucleotides occupy a specific location in the RNA sequence. This application assumes that all complementary nucleotides are able to bind to each other. However, this is not the case in biology, and to enhance the precision of this application a modification is required which takes into account the position of the nucleotide and how likely it is to bind to a complementary base a certain distance away. For short RNA sequences, the current approach should be fine. An additional advantage to computing the probability of two complementary bases binding a certain distance away is it allows further quantification for how likely a secondary structure of the RNA is, to filter out improbable matchings which result in unstable secondary structures.

Implementation

First of all, the user is asked to input their RNA sequence before the number of adenine, uracil, guanine, and cytosine nucleotides are counted. The program checks whether a perfect matching is possible by comparing whether there are equal quantities of complementary nucleotides. Once that has passed, the *factorial* class is initialised:

*fact = 1*

*for i in range(bases):*

*fact \*= (i + 1)*

*return(fact)*

'fact' is a shortened form of 'factorial’, and is assigned an integer value of 1. If it was 0, then upon multiplication (as seen further) the value will remain 0. This is why the Fibonacci sequence begins with 1.

The 'for' loop will repeat for the number of times equal to the number of bases. If there are 5 uracil nucleotides, each one must be accounted for. With each loop, the value of i increases by 1, starting with a value of 0.

The 'fact' variable is multiplied by the value of 'i + 1' to form a recursion sequence. 1 is added to put the value of i in the correct 'frame'; the initial value of i is 0. 1 x 0 = 0, and would result in the wrong answer.

'fact' is returned, and stored as a variable for both adenine-uracil, and guanine-cytosine before the two variable are multiplied together.

Limitations, and Improvements

As discussed previously, a limitation of this application is that it does not account for the probability of two complementary nucleotides to bind to each other. A factor which can affect the probability is the distance between the two bases. Therefore, the purpose of this application is to find the absolute theoretical maximum number of permutations possible, but it give no insight into how probable a matching is. In more advanced applications it is important to calculate the probability of a matching to be able to filter out improbable matchings which result in unstable 3D secondary structures of the RNA strand.