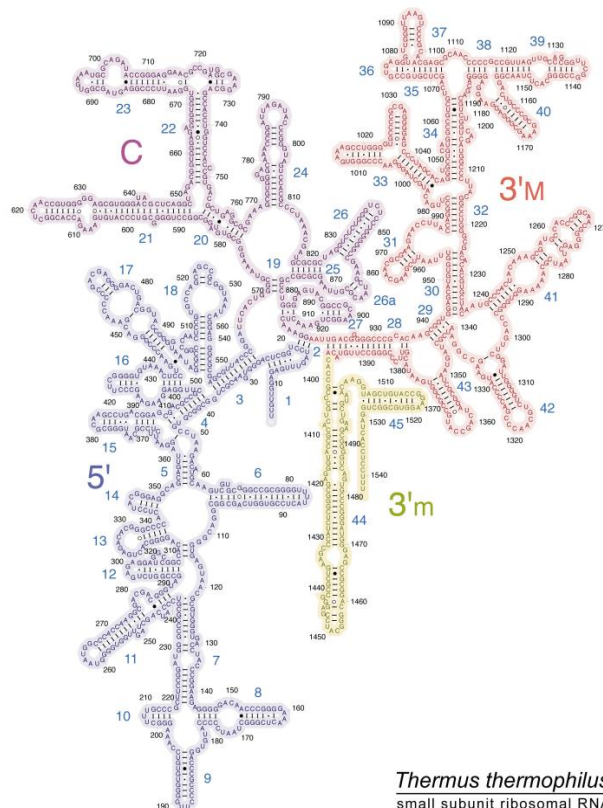
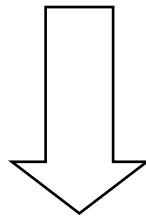


RNA Secondary Structure Prediction: The Co-transcriptional effect on RNA folding

agucacaaacgcu
agugcuaguuuu
uaugcagucuuu



Abstract

RNA secondary structure prediction is an area of bioinformatics that is the subject of significant amounts of research; it is the process of turning a sequence of bases on an RNA strand, for example, AUGUAGCAUC into a map of the interactions between the base pairs. This report looks at different methods of approaching this problem using dynamic programming algorithms. Nussinov's algorithm was used as a starting point to give a rough idea of what we were aiming for in terms of accuracy of any future prediction. This algorithm would also provide a good starting point for beginner computer scientists as it is a fairly simple program that can then be built upon. Next, an enhanced version of the Nussinov which looked at base pair stackings was used. For a final improvement, the effects of "co-transcriptional folding" were incorporated into the algorithm. When comparing the predicted structures of some RNA strings from a dataset to their actual structures, it was found that while the improved Nussinov correctly predicted 29.4% of base pairs, the co-transcriptional algorithm correctly predicted 30.2% of base pairs, an absolute increase of 0.8%, or a relative increase of about 2.7%, a result that fitted with what was expected.

Introduction

What exactly is RNA? When DNA is mentioned, people get an image of two intertwining strands forming a double helix shape, but a similar mental image doesn't really exist for RNA, despite their similarities. The main difference is that whereas DNA is formed of two strands, RNA is a copy of one of the strands in the DNA, the "coding strand". Among other functions this allows it to transfer genetic information between DNA and proteins. When the RNA has copied the coding strand of the DNA, the new strand folds into a 3 dimensional structure. The secondary structure of the RNA describes the base pair interactions within this structure, and it is with the prediction of this structure that the focus of my project lies on. The other main differences between the molecules are:

- Where DNA stands for deoxyribonucleic acid, RNA stands for ribonucleic acid
- RNA has Uracil as a nucleotide base instead of Thymine

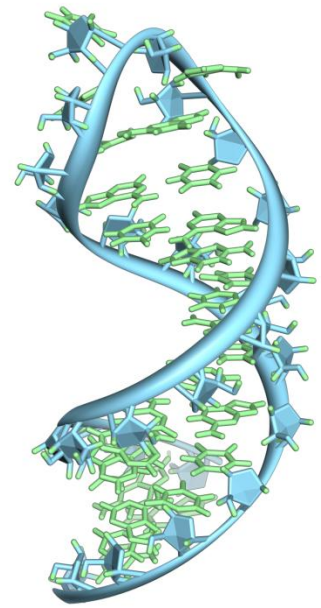


Diagram 1

An RNA molecule exhibiting a "hairpin loop"

The prediction of the secondary structures of RNA molecules is of great interest to bioinformaticists and this report will be looking at different methods of approaching this problem using dynamic programming algorithms; Dynamic programming is the use of programming to break a large problem into smaller subproblems. Another idea that was used was recursive programming, for an in depth definition, see the appendix. The programming language Python was chosen due to its simplicity, which allows beginners to pick it up easily and not get weighed down by confusing syntax.

RNA secondary structure prediction was pioneered by Ruth Nussinov in the late 70s and early 80s, her original algorithm looked at maximising the number of base pairs in the final structure, although this had many problems, the main one being that a single sequence can lead to 2 different structures with the same number of base pairs with no way of deciding between them.

RNA secondary structure prediction is an important area of study as the prediction of RNA molecule structures with 100% accuracy would lead to a greater understanding of the building blocks of life.

Base pairings

An RNA strand is a long strand of nucleotide bases. Each nucleotide base can be one of four types Adenine, Guanine, Cytosine or Uracil, and these are denoted by the letters A, G C and U. Hydrogen bonds can form between A and U or G and C, and these pairings are generally what is being referred to when a "base pair" is mentioned. In

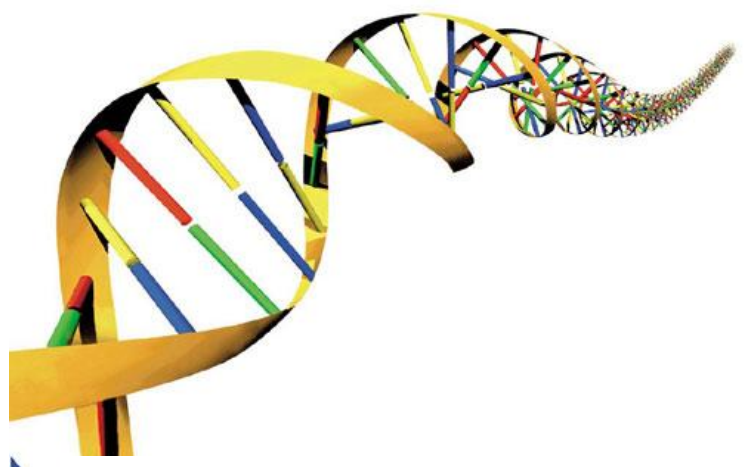


Diagram 2

A DNA molecule showing the Double helix shape, although this shape is also exhibited by RNA

this report however, the “wobble” base pair is being considered as well; a hydrogen bond between U and G. This base pairing is sometimes ignored in the prediction of secondary structures as it is held together by a weaker bond, and therefore less likely to occur.

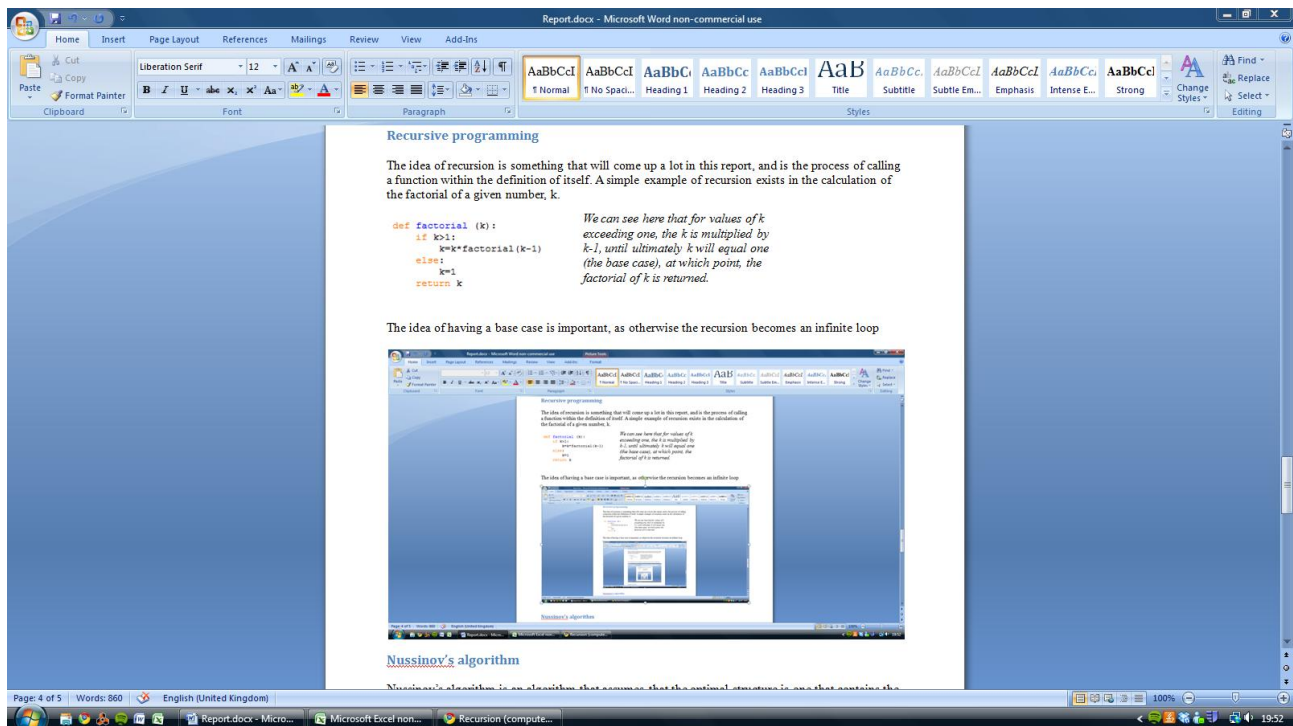
Recursive programming

Recursion is the process of calling a function within the definition of itself. A simple example of recursion exists in the calculation of the factorial of a given number, k .

```
def factorial (k):  
    if k>1:  
        k=k*factorial(k-1)  
    else:  
        k=1  
    return k
```

We can see here that for values of k exceeding one, the k is multiplied by $k-1$, until ultimately k will equal one (the base case), at which point, the factorial of k is returned.

The idea of having a base case that the k reduces to is important, as otherwise the recursion becomes an infinite loop



Method

Nussinov's algorithm

```
def match (a):
    def matchcheck(i,j):
        if (a[i]=="a" and a[j]=="u") or (a[i]=="g" and a[j]=="u") or (a[i]=="c" and a[j]=="g"...
            m=True
        else:
            m=False
        return m
    result=[]
    s=0
    if len (a)<2:
        t=0
        result.append(t)
    elif len (a)==2:
        if matchcheck(0,1)==True:
            t=1
            result.append(t)
        else:
            t=0
            result.append(t)
    else:
        while s<len(a):
            t=0
            j=len(a)-1
            if matchcheck(s,j)==True:
                t=1+(match(a[0:s])+match(a[s+1:j]))
                result.append(t)
            s=s+1
        t=match(a[0:len(a)-1])
        result.append(t)
    return max(result)
print match(raw_input("Sequence?"))
```

Nussinov's algorithm is an algorithm that assumes that the optimal structure is one that contains the maximum possible number of base pairings. In reality, there are many different factors to consider, but it is an appropriate place to start.

This algorithm actually returns the number of base pairs in the structure that can contain the maximum number of base pairs, and not an actual structure itself. There is a version of this algorithm with a trace back function in the appendix, but for ease of understanding, I have included the basic algorithm in the main body of the text.

The algorithm essentially works by recursion (see appendix) where the first and last positions in the



sequence are labelled i and j respectively. The sequence is then broken down into smaller and smaller subsequences which become so small that they become one of

the base cases, that is, where the recursion stops and scores are outputted. In this case, the base cases are:

- i becomes greater than j , in which case 0 is returned as there can be no possible match
- i and j are next to each other in the sequence
 - and can form a base pair, in which case a score of 1 is returned
 - and can't form a base pair, in which case a score of 0 is returned

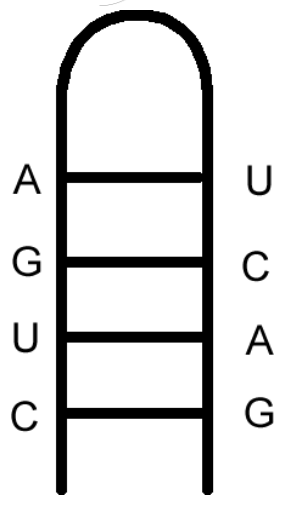
This is expressed in the diagram below

$N(i,j)$
=

- 0 if $i \geq j$
- 1 if $j-i=1$ and i and j can form a base pair
- 0 if $j-i=1$ and i and j can't form a base pair
- $\max \left\{ \begin{array}{l} N(i,j-1) \\ \max_{i \leq s < j} \left\{ \begin{array}{l} 1 + N(i,s-1) \\ 1 + N(s,j-1) \end{array} \right. \end{array} \right.$

Improved Nussinov's algorithm (incorporating base stacking)

A base pair stacking is defined as more than one nucleotide base pair in a row, and a Helix with n consecutive base pairs is given a stack value of $n-1$. Maximising the stack value as opposed to the number of base pairs is considered to be a more appropriate method of secondary structure prediction as it can be observed that RNA molecules tend to prioritise stackings over pairings in the folding process, and maximising the stack value in the algorithm leads to much greater accuracy. Adding this modification to the original Nussinov's algorithm is the next step in improving secondary structure prediction accuracy



This section of RNA would take a stack value of 3

```
class matchdict:
    def __init__(self, s):
        self.dict={}
        self.a = s
        self.sdict={}
        self.hdict={}
    def matchcheck(self,i,j):
        if (self.a[i]=="a" and self.a[j]=="u") or (se
            m=True
        else:
            m=False
        return m
    def match (self,i,j):
        result=[]
        if (i,j) in self.dict:
            t=self.dict[i,j]
            result.append(t)
        else:
            if j-i+1<=2:
                t=0
                result.append(t)
            else:
                s=i
                while s<j:
                    if self.matchcheck (s,j)==True:
                        t=self.match(i,s-1)+self.stac
                        result.append(t)
                        s=s+1
                        t=self.match(i,j-1)
                        result.append(t)
                    self.dict[i,j]=max(result)
                return max(result)
    def trace(self,i,j):
        structure=[]
        if j-i+1<=2:
            pass
        else:
            s=1
            while s<j:
                if self.matchcheck(s,j)==True:
                    if self.match(i,s-1)+self.stack(s,
                        structure=self.trace(i,s-1)+self.tracestack(s,
                            s=s+1
                    if len(structure)==0:
                        structure=self.trace(i,j-1)
                    return structure
                def tracestack(self,i,j):
                    structure=[]
                    if j-i+1==2:
                        if self.matchcheck(i,j)==True:
                            structure.append((i,j))
                    elif j-i+1<=2:
                        pass
                    else:
                        if self.matchcheck(i,j)==True:
                            if i+1<j-1 and self.matchcheck(i+1,j-1)==True and self.stack(i,j)==self.stack(i+1,j-1)+1:
                                structure=[(i,j)]+self.tracestack(i+1,j-1)
                            else:
                                structure=self.trace(i+1,j-1)+[(i,j)]
                        return structure
                def stack(self,i,j):
                    result=[]
                    if (i,j) in self.sdict:
                        t=self.sdict[i,j]
                        result.append(t)
                    else:
                        if i>=j:
                            t=0
                            print "u"
                            result.append(t)
                        elif self.matchcheck(i,j)==False:
                            t=0
                            print "u"
                            result.append(t)
                        elif self.matchcheck (i,j)==True:
                            if i+1<j-1 and self.matchcheck(i+1,j-1)==True:
                                t=self.stack(i+1,j-1)+1
                                result.append(t)
                            t=self.match(i+1,j-1)
                            result.append(t)
                    self.sdict[i,j]=max(result)
                return max(result)
```

This algorithm works on similar principles to the original Nussinov algorithm, and like the Nussinov, can be better understood when represented in the form of the diagram below.

$$D(i,j) = \begin{cases} 0 & \text{if } i \geq j \\ 0 & \text{if } i = j - 1 \\ \max \begin{cases} D(i,j-1) \\ \max_{s} \{ D(i,s-1) + E(s,j-1) \} \end{cases} \end{cases}$$

$$E(i,j) = \begin{cases} -\infty & -\text{len}(a) & \text{if } i \geq j \\ -\infty & -\text{len}(a) & \text{if } i \text{ and } j \text{ can not form a base pair} \\ \max \begin{cases} E(i+1,j-1) + 1 \\ D(i+1,j-1) \end{cases} \end{cases}$$

Co-transcriptional folding effect

The idea of co-transcriptional folding is that when an RNA strand folds back on itself, it isn't done in a single movement, but rather, it folds its way along the strand from the 5' end to the 3' end, or vice versa. One of the implications of this is that it is possible for another base, c , near a helix to "compete" with a pair ij in the helix, provided that ci could form a helix with length min_{stem} or more. In our experiment, we have taken the value of min_{stem} as being 9. This number was chosen as it was also used as the min_{stem} value for another piece of research into the co-transcriptional folding effect by Meyer and Miklós². This effect was applied to the algorithm as a weighting, specifically:

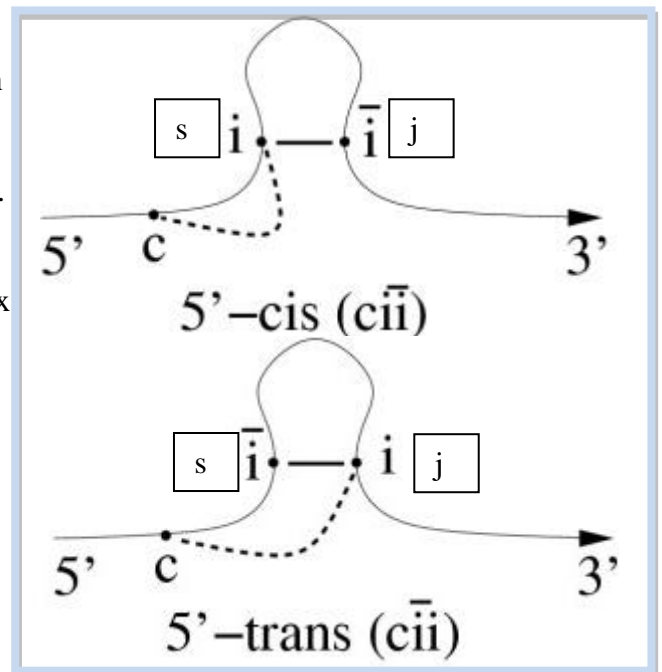
This means that when s and j match, it runs through

```
def cotran(self,s,j):
    if self.matchcheck (s,j)==True:
        t=1
        for c in range (0,s-1):
            if self.helist[c][s]>=9 and self.matchcheck(c,s)==True:
                t=t+(self.b/((s-c)*math.log10(s)))
            elif self.helist[c][j]>=9 and self.matchcheck(c,j)==True:
                t=t-(self.d/((s-c)*math.log10(s)))
        self.clist[s][j]=t
```

and s can pair.

And subtracts a score of $\frac{b}{(s-c) \log(s)}$ for every case where c and j can pair

Used under creative commons attributive license attributed to reference 2



every c value from 0 (the 5' end) to $s-1$, and:

adds a score of: $\frac{a}{(s-c) \log(s)}$ If c

These additions and subtractions are with respect to the final score for base pair sj (b and d were used instead of a and b in the actual program due to a shortage of variable names)

In this situation, a and b are two unknown parameters. The best 10 parameters that were found can be seen in the results table

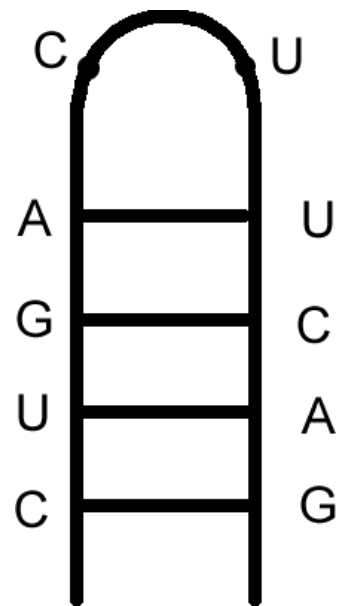
Results

The results of this report were generated using a dataset of known RNA molecules and their structures. For each RNA molecule, the sequence is given, followed by the structure, expressed in bracket notation. Bracket notation is a commonly used method of simply representing the structure of an RNA molecules through open and close brackets, as well as full stops. An example of the database entry for the adjacent RNA strand would read as follows (assuming the two ends are where the molecule stops)

Structure id: 348

CUGACUUCAG

(((((. .))))))



The 3 algorithms were adapted so that they could run through this list and predict the base pairs, then compare them to the actual structure. It was also possible to specify certain parameters such as how many structures it should predict before stopping, or the range of the lengths of structures to be considered. This second parameter is important as the algorithms used had order of n^3 time complexity. This means that when the length of a string that was being predicted doubled in length, it took 8 times (2^3) as long to predict (or thereabouts). While this isn't as dramatic as exponential time complexity, it is certainly a large enough increase in time that it was useful to ignore sequences of over 250 length in a lot of the cases.

When the algorithm was run on 20 different cases of between length 100 (this minimum length was chosen as the algorithm is disproportionately better at predicting smaller molecules) and 250, the results were as follows:

Table showing Algorithm type against prediction accuracy

Nussinov's algorithm	Improved Nussinov (with base stacking)
10.77%	36.33%

The algorithm incorporating the co-transcriptional effect was then run for a number of different a and b parameters, and this table highlights the best 10 of 157 parameters used.

Accuracy	a	b
39.16%	5	3
39.16%	5	4
39.16%	5	5
38.98%	5	7
38.90%	5	6
38.76%	5	1
38.76%	5	2
38.50%	1	3
38.50%	1	4
38.48%	5	0

It is plainly visible that 5 as an a value leads to more accurate prediction levels, yet such an observation can't be made with regards to the b values, although they have an average of 3.5

These results were only run on 20 different cases each to get a rough idea of which parameters would work best and give the best representation of the co-transcriptional effect, so after running the algorithm again but for up to 500 RNA sequences per parameter set, with a length range of between 60 and 350 bases long, for the top 5 *b* parameters the new results were as follows

Accuracy	a	b
30.21%	5	7
30.13%	5	6
30.08%	5	4
30.04%	5	5
29.97%	5	3

The improved Nussinov run with these criteria gave 29.4% accuracy

Conclusion

This report looked at different methods of approaching the problem of predicting RNA secondary structures using dynamic programming algorithms. An enhanced version of Nussinov's algorithm was built upon by incorporating the effects of "co-transcriptional folding" into the algorithm. When comparing the predicted structures of some unfolded RNA strings from a dataset to their actual structures, it was found that while the improved Nussinov correctly predicted 29.4% of base pairs, the co-transcriptional algorithm correctly predicted 30.2% of base pairs, an absolute increase of 0.8%, or a relative increase of about 2.7%, a result that fitted with what was expected.

Evaluation

Due to the environment in which this project was undertaken, and the timescales allowed, there is scope for improvement in terms of the amount of data obtained. For further research into this subject, it is recommended that this problem is approached by either:

- Using a compiled language to implement the algorithms such as C or Java, as their execution times would be radically faster than Python's, due to its interpreted nature.
- Use of cluster computing, or faster computers generally would speed up the execution time

Or a combination of these two. This speeding up of the execution time would potentially lead to more available data, and thus make the data more reliable.

Whilst there are many factors that could be changed in order to improve the algorithm, one stands out as being quite a simple modification that could possibly postulate base pairs with greater accuracy, which is assigning the different base pairings a weighted score, that is taking into account the effect of the different strengths of the hydrogen bonds. To be more specific: as was mentioned earlier in the report, the hydrogen bond that holds the wobble base pair (U to G) together is substantially weaker than the conventional GC or AU bonds, and assigning a base pair stacking involving this base pair a weighted score that reflects this weak hydrogen bond could be another potential improvement to the algorithm.

It is recommended that more research is done into the specific parameters of the co-transcriptional effect, as due to certain limitations, there were only a small number of different samples tried

References

1. <http://www.pnas.org/content/101/19/7287.full>
2. <http://www.ncbi.nlm.nih.gov/pmc/articles/PMC514895/?tool=pubmed> Co-transcriptional folding is encoded within RNA genes Irmtraud M Meyer and István Miklós