

Investigation of the number of
possible secondary RNA structures
with reference to theoretical
expressions.

Nuffield Science Project by
Fiona Rust

Supervisor: Professor Rune Lyngsø
Oxford Centre for Gene Function
Oxford University Statistics

Abstract

The problem of counting the number of possible RNA secondary structures is closely linked to predicting RNA secondary structures. In this project the formula given in Zuker and Sankoff [1984] is investigated and compared to the actual number of structures when base complementarity is taken into account. The number of stacking structures is also investigated, and follows a similar pattern to the number of any structures. The effect of randomly shuffling sequences is also investigated. The number of structures and the number of stacking structures is found to be different for real RNA sequences and shuffled RNA sequences.

Ribonucleic acid (RNA) is a single strand molecule similar to Deoxyribonucleic acid (DNA). RNA consists of a long chain of nucleotides containing a sugar group, a phosphate group and a nucleotide base. Although RNA is normally a single strand, it can fold to form local helix structures as part of a 3D (tertiary) structure of the RNA. The tertiary structure helps determine its function, so being able to predict the tertiary structure of an RNA strand is useful. This tertiary structure is built from a secondary structure which describes the base pairings within that strand. RNA is built from nucleotide bases, Adenine (A), Cytosine (C), Guanine (G) and Uracil (U). Hydrogen bonds can form between A and U or G and C. A wobble base pair can also form between G and U.

Computer models can be used to predict the secondary structures of RNA. Starting with simple models to maximise the number of base pairs for a RNA strand, fairly inaccurate structures can be produced. The model can be improved slightly when the maximum number of stacking base pairs is used instead. A stacking base pair is one that has another consecutive base pair so it can form part of a helix. However these programs can be further adapted to count the total number of structures possible for a RNA strand. Equations that count and approximate the number of structures are shown in [Zuker and Sankoff, 1984, Eq(4) and Eq(5)] (see Appendix). These equations are investigated in this project.

The approximation (Eq.(5)) matched the counting algorithm (Eq.(4)) closely. However it was an overestimate for the number of structures if base complementarity was taken into account. The effects of base complementarity were investigated for any base pairs and for stacking base pairs structures. The values were very different from the values given using the Zuker and Sankoff formula. The effects of the probability in determining the number of possible structures was also investigated.

The difference between the number of possible structures for an actual RNA sequence and the number for a randomly shuffled RNA sequence was investigated. The prediction in Zuker and Sankoff remains the same for shuffled sequences and real sequences; however the actual number of structures is different.

Counting algorithms for stacking structures was also investigated. The number of stacking structures follow the same exponential pattern against sequence length.

These investigations could be useful in searching genetic sequences for sections that code for RNA. Knowing how random sequences deviate from real sequences and how this deviates from the original model could improve searching techniques.

Method, results and Analysis

Algorithms to find structures.

The first step in this project was to develop basic algorithms that maximise the number of base pairs possible or the maximum number of stacking base pairs, which could then be used to find a structure. This uses a technique called recursion where a function is repeatedly called to build up the final value from a set of starting cases. The recursion for the maximum number of base cases is shown below: All computer code can be found in the Appendix.

Function A

$$A(i,j) = \begin{cases} 0 & \text{if length of sequence} \leq 2 \\ \text{Max} \left\{ \begin{array}{l} A(i,j-1) \\ \text{Max} \left\{ \begin{array}{l} 1+A(i,k-1)+A(k,j-1) \\ \text{(for all } i \leq k \leq j, \text{ where } k \text{ can base pair with } j) \end{array} \right. \end{array} \right. \end{cases}$$

For any sequence tested the starting value for i is 0 and j is the length of sequence-1. The values that are passed to the function represent positions in the sequence. The letters in these positions are then used when testing for base complementarity.

In Function A there is the assumption that no adjacent bases can form a bond. Therefore if the length of the sequence is equal to or smaller than two, there are no base pairs formed. All possible positions that could form a base pair with the last position are cases and the score for each case is stored. The case where the last base is unpaired is also taken into account, and the maximum of all these cases is finally returned. The maximum number for each pair is stored as it is calculated. This increases the efficiency of the algorithm as values are not recalculated. This is called dynamic programming.

Function A was added to by using traceback to find one structure with the maximum number of base pairs. The table below shows a simple example to illustrate this principle:

	g	c	a	g	c
g	0	0	0	0	2
c		0	0	0	0
a			0	0	0
g				0	0
c					0

This table shows the maximum number of base pairs for each combination of pairs of bases. The maximum number for this sequence is 2, position 0 pairing with position 4 and position 1 pairing with position 3. This is found using the algorithm. For all values that $j-i+1 \leq 2$, the result is 0. The first base pair that can form is between position 1 and 3. One is added to this value in the cell diagonally upwards representing the base pair between the adjacent bases. This is the maximum for this sequence. The traceback follows the algorithm backwards, taking values diagonally left and downwards. The base pairs are added in reverse order to the final structure. The arrows show this.

In order to improve structure prediction algorithms, optimisation parameters must be added. One simple way to improve this basic algorithm is to only allow base pairs that form part of a stack. This is more like real RNA structures where base pairs form part of a long helix. This involves two mutually recursive functions that are called in the same way as the previous algorithm. The recursion is shown below:

Functions B and C

$$B(i,j) = \begin{cases} 0 & \text{if length of sequence} \leq 2 \\ \text{Max} \left\{ \begin{array}{l} B(i,j-1) \\ \text{Max} \left\{ \begin{array}{l} B(i,s-1)+C(s,j-1) \\ \text{(for all } i \leq k \leq j. \text{ where } k \text{ can base pair with } j) \end{array} \right. \end{array} \right. \end{cases}$$

$$C(i,j) = \begin{cases} -\infty & \text{if length of sequence} \leq 2 \text{ or no base pair can be formed} \\ \text{Max} \left\{ \begin{array}{l} B(i+1,j-1) \\ C(i+1,j-1) \end{array} \right. \end{cases}$$

The Function B tests whether the base pair could form part of a stack or could not, and takes the maximum of either of these situations by referring to Function C. Function C assumes a base pair can be formed and finds the maximum number of base pair stackings. Function C returns negative infinity if a base pair cannot form. A base pair cannot be added unless it can form part of a stacking. The above functions can also be traced back to create a structure following the same principle as before.

Counting Structures

The functions described above can be adapted to count the number of possible structures for a sequence. However the first thing to test is how accurate the approximation given in Eq. (5) is. In order to do this Eq. (4) was used to count the number of possible structures for a set of real RNA sequences from the data set (summary of data set can be found in the Appendix). The probability (p) was generated by taking the total number of each base pair in a sequence and using this formula:

$$p = (2au + 2cg + 2gu) / n^2$$

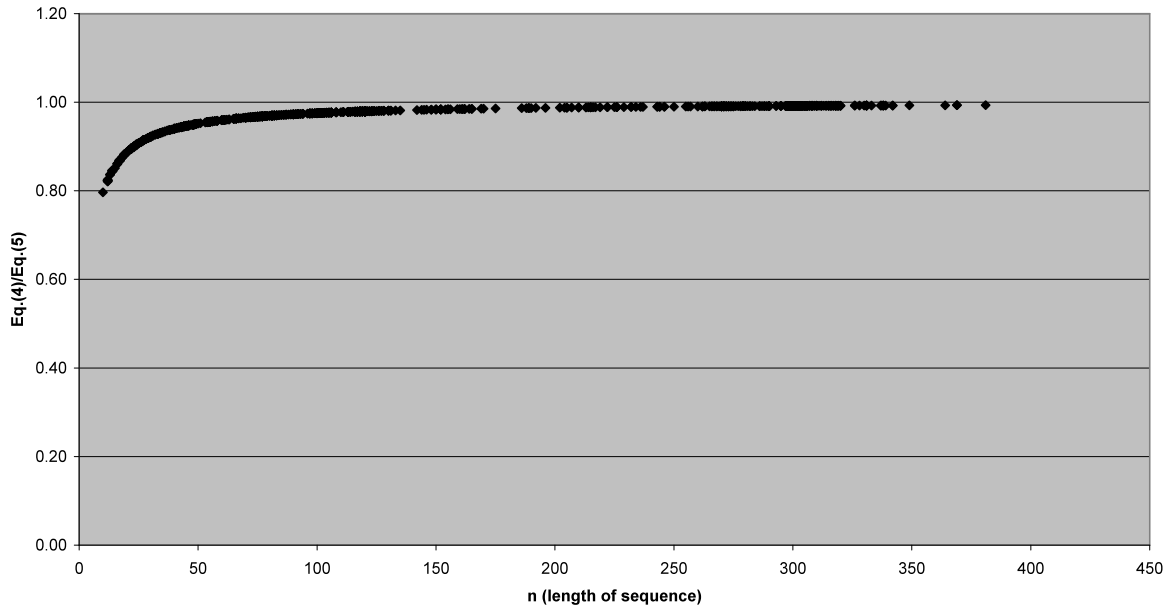
(1)

(a=number of As, c=number of Cs, g=number of Gs, u=number of Us, n=length of sequence)

This formula takes into account the possibility of G pairing with U. Below is a graph to show the results of this experiment.

Graph 1

Graph to show the accuracy of the approximation given in [Zuker and Sankoff, 1984, Eq.(5)] over a range of sequence lengths



Graph 1 shows that the accuracy increases with increasing sequence length. However the way of counting the number of structures does not take into account the base pair complementarity. This could be taken into account by modifying Function A which as shown below.

Function D

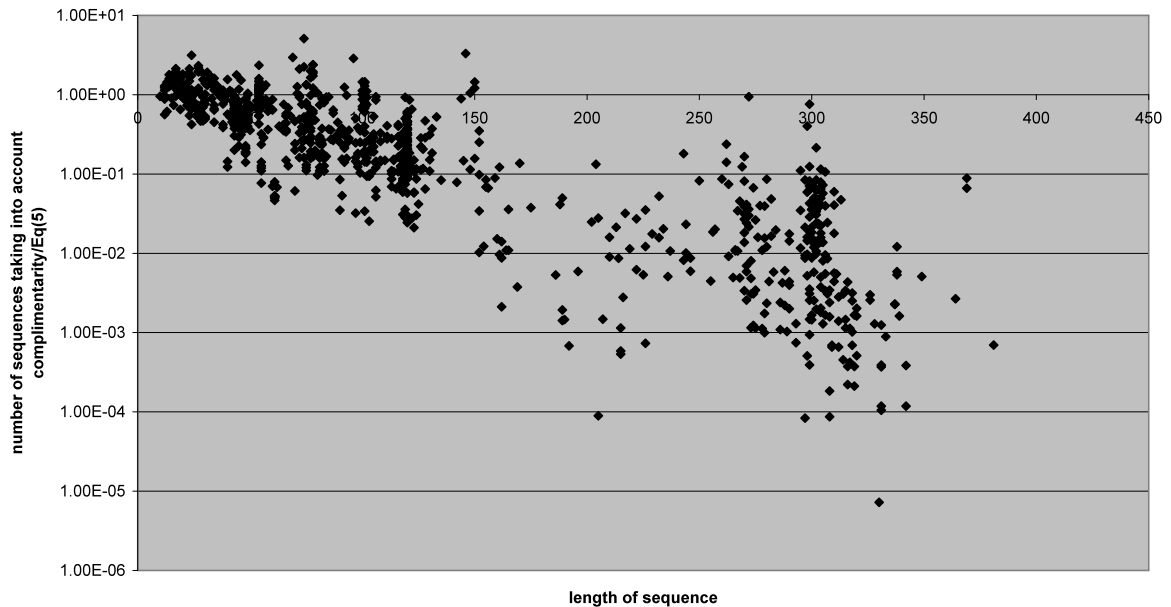
$$D(i,j) = \begin{cases} 1 & \text{if length of sequence} \leq 2 \\ \sum \left\{ \begin{array}{l} \sum_{k=i}^{j-1} \{ D(i,k-1) \times D(k+1,j) \} \\ D(i,j-1) \end{array} \right. & \text{(for all k that base pairs with j)} \end{cases}$$

This is very similar to Function A. There is only one possible structure if the length of the sequence is less than 3. Otherwise for each position that can base pair with the last position, the number of structures between the beginning of the sequence and the base pair is multiplied by the number of structures between the base pairs. The other situation is that the last position is unpaired and the number of structures for this situation is calculated. The sum of all the results is returned.

If the approximation given in Eq. (5) was correct then the number of structures from Function D divided by the number from Eq. (5) should be 1. Graph 2 below shows this ratio against sequence length.

Graph 2

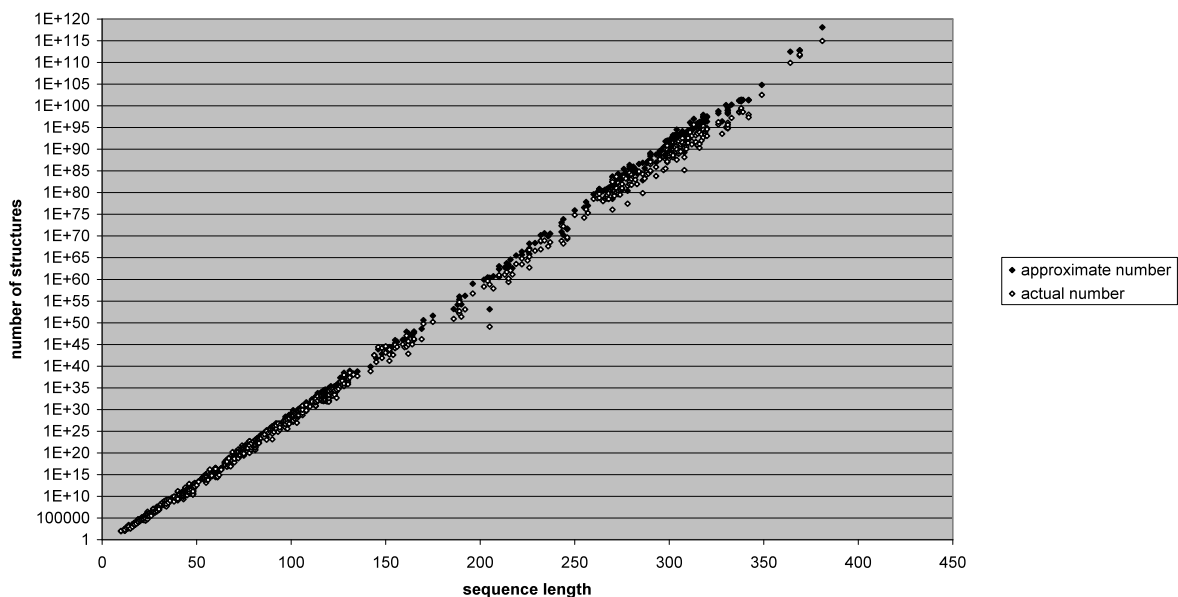
Graph to show how the accuracy of Eq(5) against number os sequences taking into account complimentarity



Graph 2 shows that there is an overestimate for the number of structures which tends to be a bigger overestimate for longer sequences. The equation for the expected number of structures is exponential against sequence length. The actual number of structures also follows an exponential pattern (shown in Graph 3 in appendix). The difference between the approximation and the actual number of structures is shown below.

Graph 4

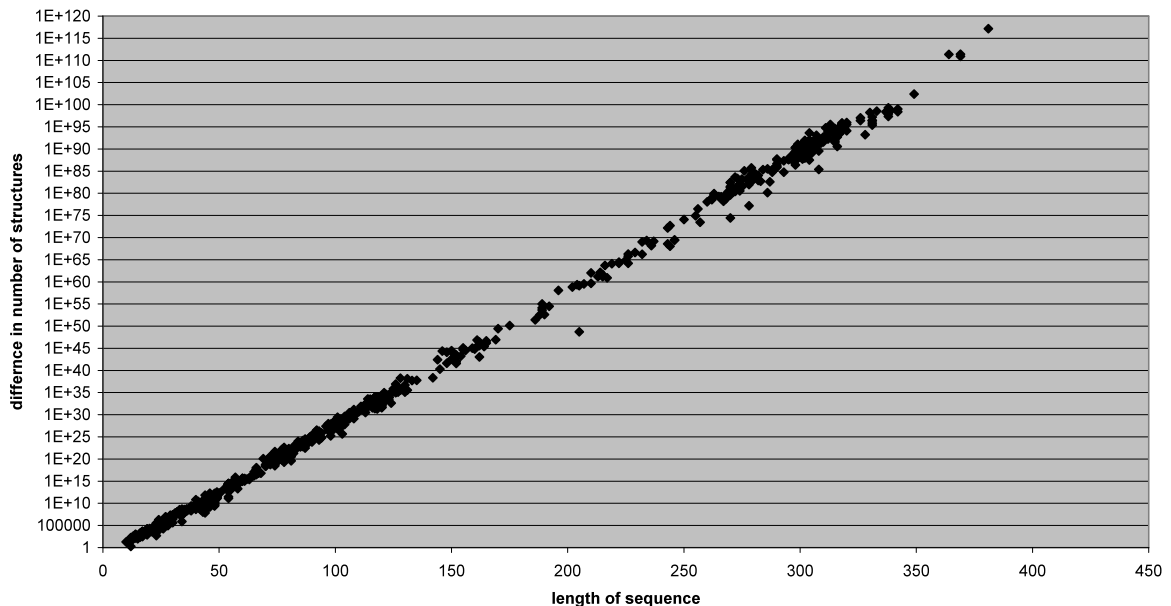
Graph to show the difference between the actual number of sequences and the approximation for real RNA sequences



Graph 4 shows that the difference between the two values increases as the length of the sequence increases. The next thing to investigate is whether there is a difference in the number of structures for real sequences and for randomly shuffled sequences. In these shuffled sequences the order of the base pairs changes while the number of each base remains the same. The shuffled sequences are generated from the real sequences in my data set. The probability also remains the same for shuffled and real sequences. Below is a graph showing the absolute values of the number of structures for real sequences minus the number of structures for random sequences. Real sequences tend to have less possible structures than random sequences.

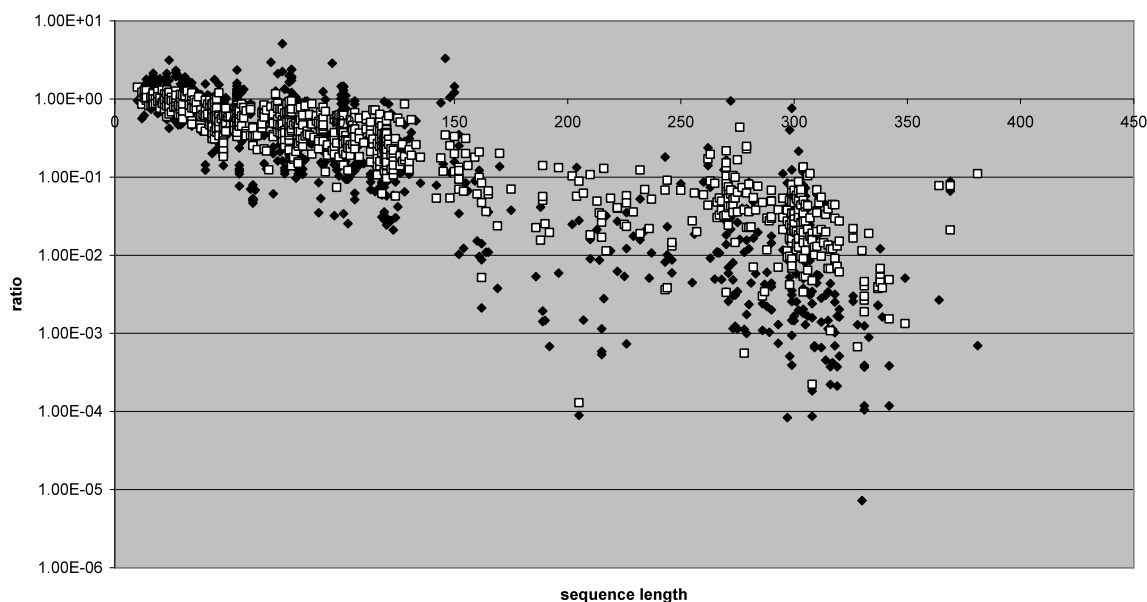
Graph 5

Graph to show the absolute difference between the number of structures for real sequences and shuffled sequences



Graph 6

Graph to show the difference in the ratio of the number of structure compared to the predicted number between real and shuffled sequences



Graph 6 shows the ratio of structures for real sequences and random sequences. The ratio tends to be closer to one for shuffled sequences. This suggests the prediction is more accurate for shuffled sequences.

Investigating effects of counting stacking structures

A new function was needed to investigate the number of possible structures consisting of stacks of base pairs. The Functions B and C that maximised the number of stacking base pairs were unsuitable as there was an overlap between the cases. In order to count the number of structures there needs to be disjoint cases. Below are the functions (E, F and G) for the maximising base pairs followed by the functions (H, I and J) for counting this type of structure.

Functions E, F and G

$$\left\{ \begin{array}{l}
 E(i,j) = \begin{cases} -\infty \text{ if } i \text{ and } j \text{ cannot form a base pair} \\ F(i+1,j-1) \end{cases} \\
 F(i,j) = \begin{cases} -\infty \text{ if } i \text{ and } j \text{ cannot form a base pair} \\ F(i+1,j-1)+G(i+1,j-1) \end{cases} \\
 G(i,j) = \begin{cases} 0 \text{ if length of sequence } < 4 \\ \max \begin{cases} G(i,j-1) \\ \text{Max } \{G(i,k-1)+ E(i,k-1) + E(k,j)\} \end{cases} \end{cases}
 \end{array} \right.$$

Functions H, I and J

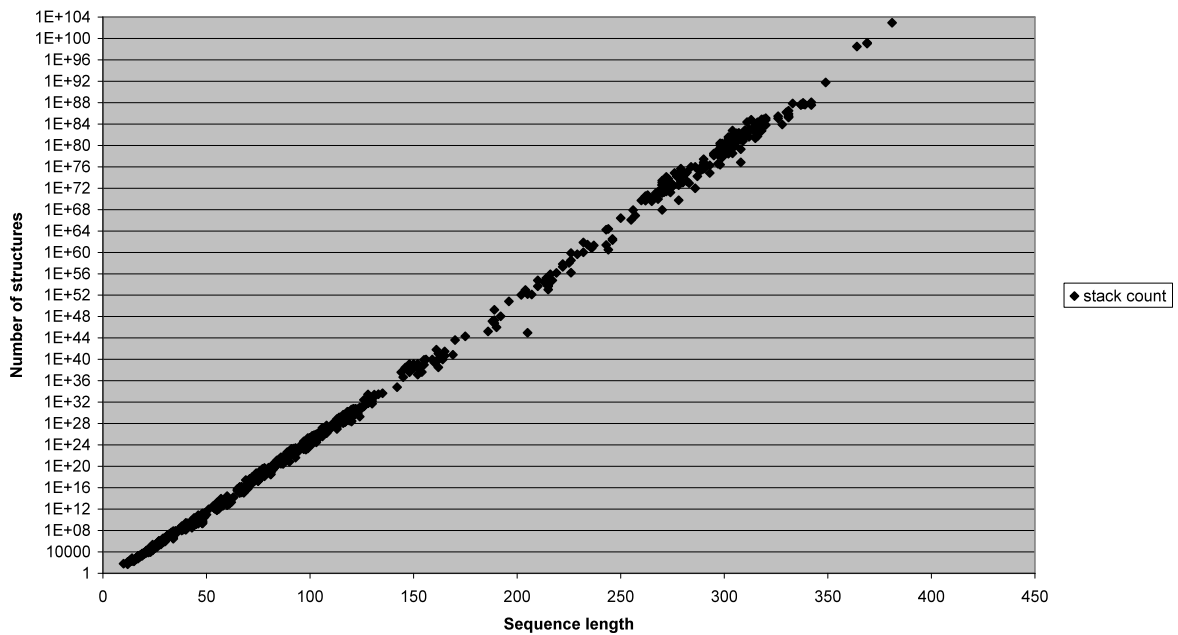
$$\left. \begin{array}{l}
 H(i,j) = \begin{cases} 0 \text{ if length of sequence} < 4 \\ 0 \text{ if } i \text{ and } j \text{ cannot form a base pair} \\ I(i+1,j-1) \end{cases} \\
 I(i,j) = \begin{cases} 0 \text{ if length of sequence} < 2 \\ 0 \text{ if } i \text{ and } j \text{ cannot form a base pair} \\ I(i+1,j-1)+J(i+1,j-1) \end{cases} \\
 J(i,j) = \begin{cases} 1 \text{ if length of sequence} < 5 \\ \sum \left\{ \begin{array}{l} J(i,j-1) \\ \sum_{k=i+1}^{j-1} \{ J(i,j-1) \times H(k,j) + H(i,k-1) \times H(k,j) \} \end{array} \right. \end{cases}
 \end{array} \right\}$$

In order to create stacking structures or count possible stacking structures 3 different functions are needed. Functions E and H presume there is already enough base pairings found to form a stack (i.e. two consecutive base pairing). Functions F and I presume one base pair has been found and either a stack could or could not be formed. Functions G and J presume nothing. Both Function H and J are called and added together to find the total number of possible structures.

The number of this type of structure was a lot less than the previous case. However Graph 7 below shows the number of stacking structures still follows an exponential pattern based on sequence length. This means a similar approximation should be possible for this type of structure. Stacking structures are more realistic than base pair maximising structures so this could be explored further.

Graph 7

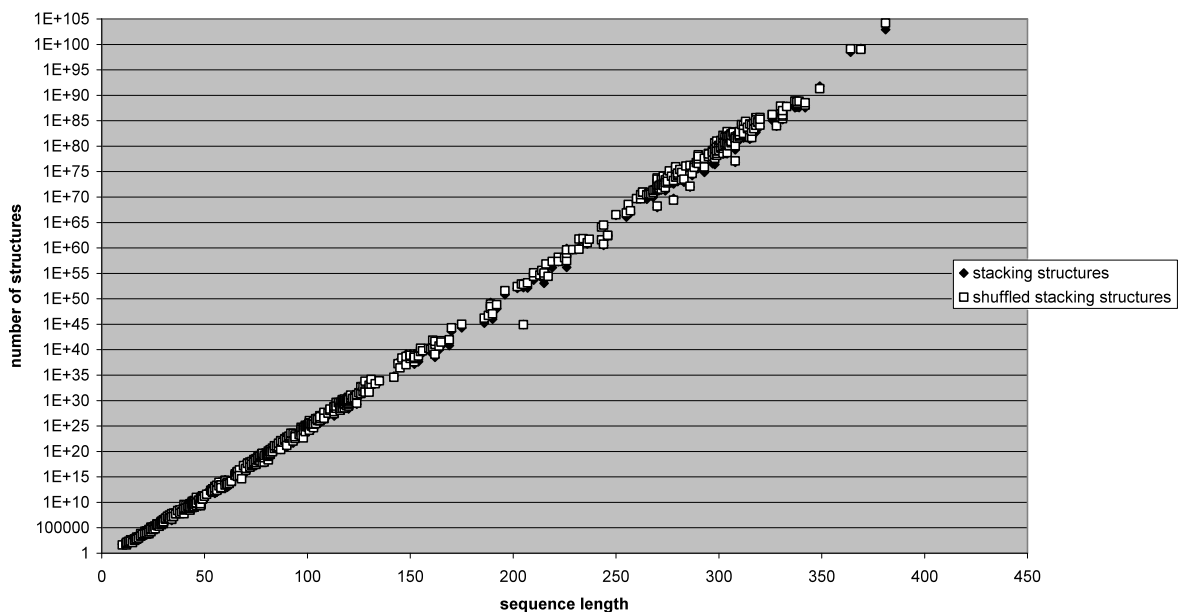
Graph to show the number of stacking structures against sequence length



The number of stacking structures for shuffled sequences was also investigated. Graph 8 below shows that the difference between the number of structures for real and stacking structures is a lot less for stacking structures. This may also mean more accurate predictions could be made for stacking structures.

Graph 8

Graph to show the difference between the number of stacking structures for real and shuffled sequences



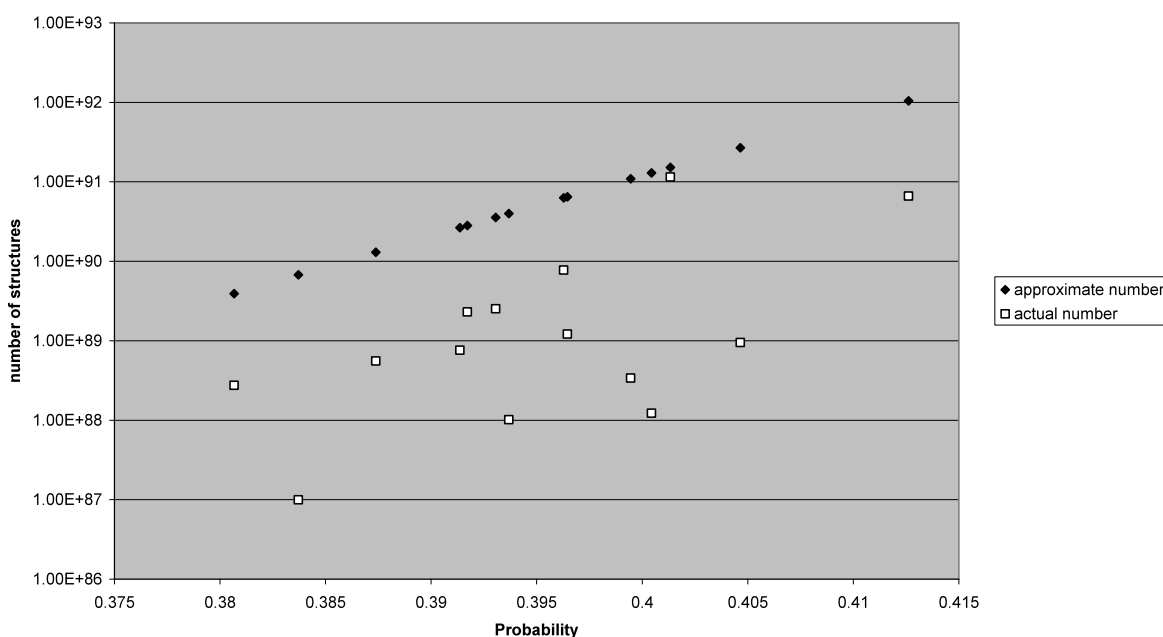
The effects of probability (p)

As the number of structures is definitely based on an exponential of the sequence length, it is of interest to assess how the value for p affects the prediction. This is even more important as the probability is the same for a real sequence and a corresponding shuffled sequence although the number of structures is different.

In order to test how relevant the probability is for both random and real sequences, a sample of sequences with the same length but different p values. The sequence length chosen was 299, as this length was the most common length over a length of 180. A length of 180 is about the length where the approximation became most accurate. Below are two graphs comparing the approximation with the actual number of sequences for both real and shuffled sequences.

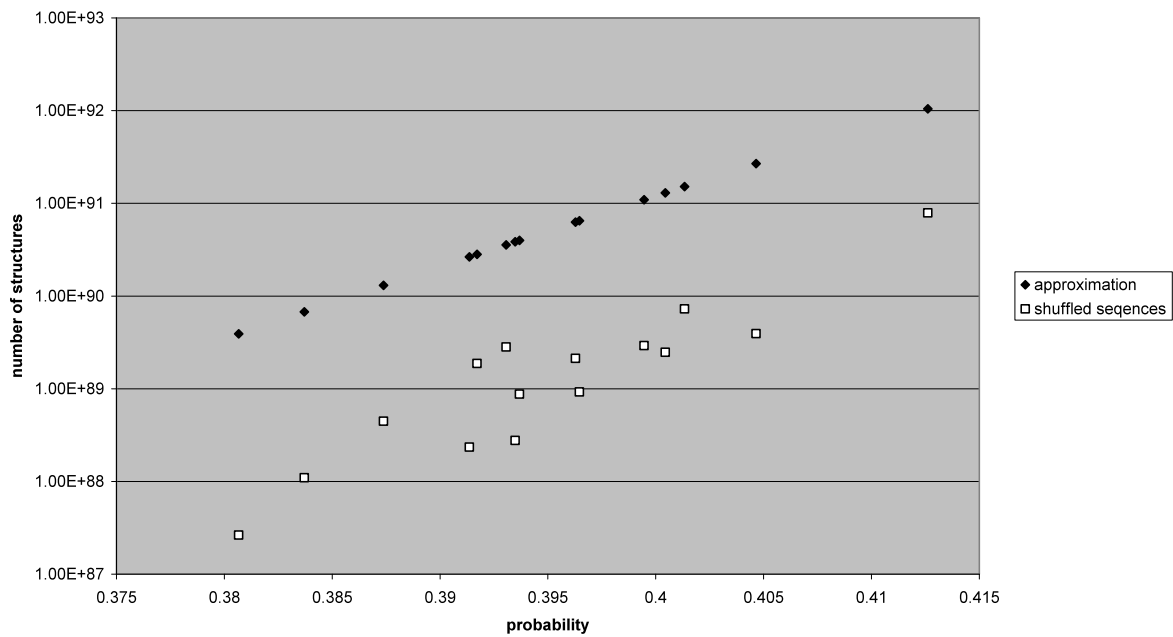
Graph 9

Graph to show the difference between the approximation and real sequences for a length of 299



Graph 10

Graph to show the difference between the approximation and shuffled sequences for length of 299

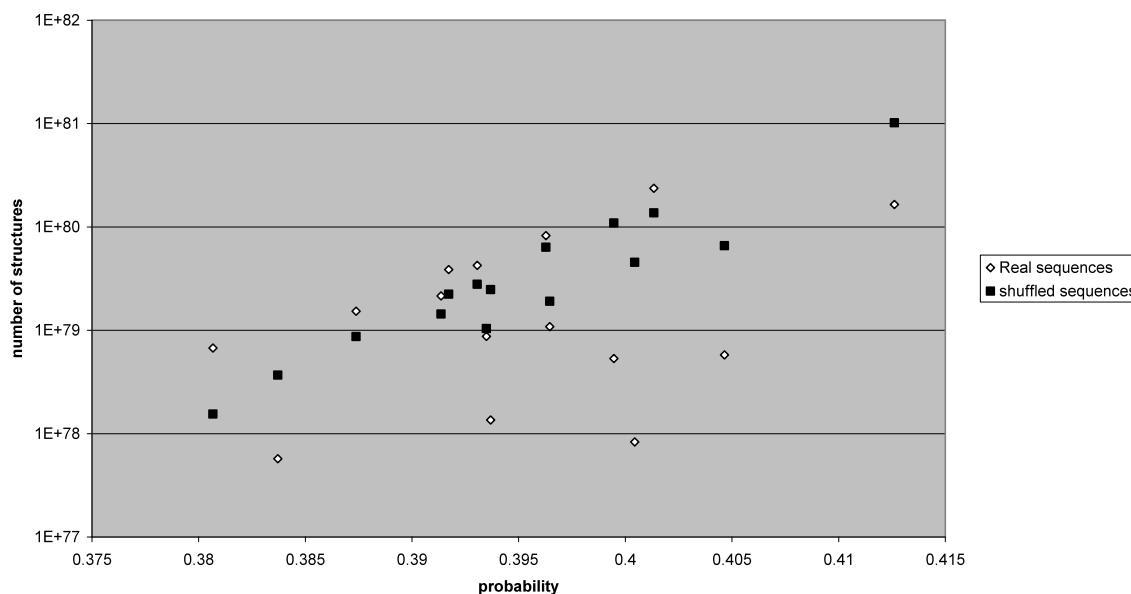


Graph 9 shows that for real sequences the number of structures are more scattered than for the shuffled sequences on Graph 10. For both types of sequences the number of structures is lower than predicted by the equation. The number of sequences grows exponentially with probability so it is reasonable to use probability as a way of predicting the number of structures for sequences with the same length.

The effects of probability on stacking structures was also investigated. The same sequences were used as before with a length of 299. Graph 11 below shows the number of structures for both shuffled and real sequences.

Graph 11

Graph to show the difference in the effect of probability for stacking structures for random and real sequences.



The number of stacking structures also grows exponentially with probability. However the real structures are more scattered. A prediction similar Eq. (5) for any structures, should be possible for stacking structures.

Conclusion

From the data collected, it is shown the approximation tested is a slight overestimate of the actual number of sequences. As the number of structures grow exponentially with sequence length, the differences are greater for longer sequences. For the same sequence length, the number of structures does depend on the probability.

It was also found that there is a difference between the number of structures for real sequences and randomly shuffled sequences. This affects the accuracy of prediction. However the shuffling carried out in this investigation only kept the number of each base the same. A better way of shuffling sequences is to keep the number of dinucleotides the same for shuffled sequences as in the real sequence. This may affect the results as Workman and Krogh [1990] show that there is no difference in free energies when dinucleotide pairs are kept.

A different type of structure could be counted in a similar way as they follow a similar pattern. The number of stacking structures grows exponentially with sequence length. Also for the same sequence length the number of structures grows exponentially with probability. Using sequence length and probability as variables an estimating equation should be possible.

The shuffling method was the same when counting any structure and counting stacking structures. Therefore the difference between shuffled sequences and real sequences for stacking structures may change if dinucleotide pairs were kept.

Appendix

Data Set

The RNA sequences came from a data set. The length of these sequences ranged from 10 to 381. The results of all experiments can be found at end of appendix.

Equations from Suker and Sankoff:

Zuker and Sankoff Eq(4) to estimate the number of possible structures for sequence length N. This assumes that two adjacent pairs cannot form a base pair.

$$E(N + 1) = E(N) + \sum_{k=0}^{N-2} pE(k)E(N - k - 1)$$

Zuker and Sankoff Eq(5) to approximate Eq(4)

$$E(N) \approx HN^{-3/2} \alpha^N$$

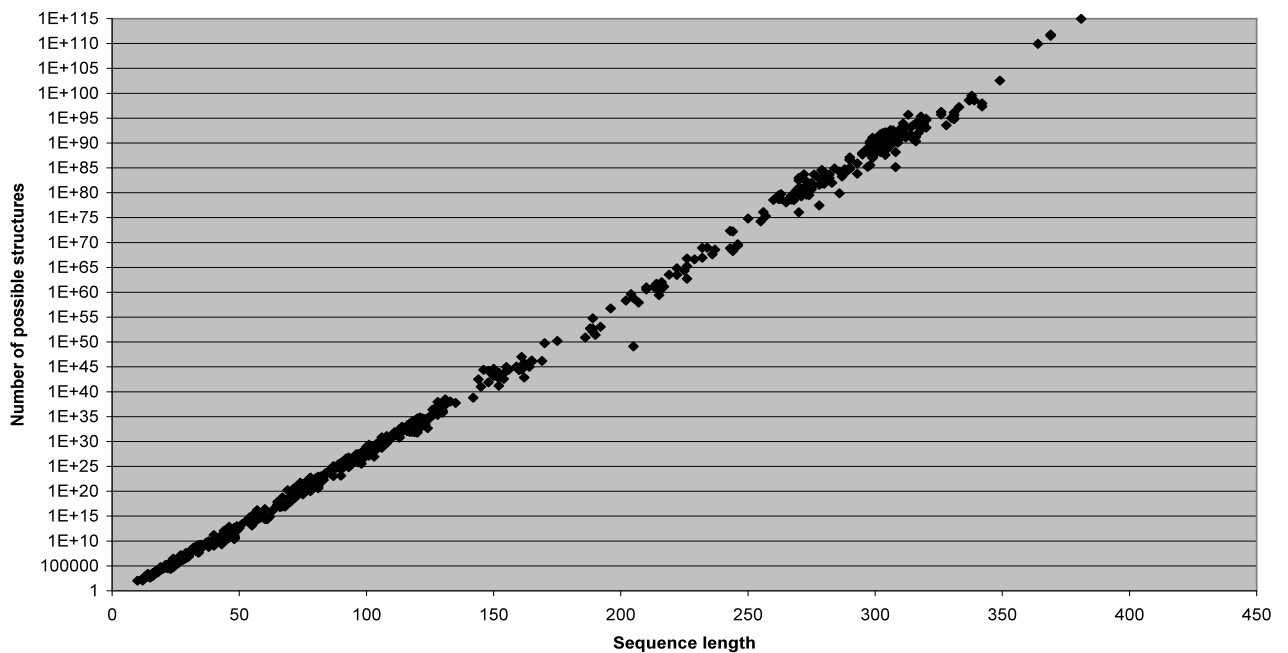
$$\alpha = \left(\frac{1 + \sqrt{1 + 4\sqrt{p}}}{2} \right)^2$$

$$H = \frac{\alpha(1 + 4\sqrt{p})^{1/4}}{2\sqrt{\pi} p^{3/4}}$$

Graphs

Graph 3

Graph to show the number of structures of real RNA sequences against length of sequence



The language used was Python. Below is the code for all the different algorithms described in the main report:

Function A:

```
class counter:
    def __init__(self,s):
        self.a=s
        self.mdic={ }
    def matchcheck(self,i,j):
        if (self.a[i]=="a" and self.a[j]=="u") or (self.a[i]=="c" and
self.a[j]=="g") or (self.a[i]=="u" and self.a[j]=="a") or (self.a[i]=="g"
and self.a[j]=="c") or (self.a[i]=="g" and self.a[j]=="u") or (self.a[i]=="u"
and self.a[j]=="g"):
            m=True
        else:
            m=False
        return m
    def match (self,i,j):
        result=[]
        if (i,j) in self.mdic:
            t=self.mdic[i,j]
            result.append(t)
        else:
            if j-i+1<=2:
                t=0
                result.append(t)
            else:
                s=i
                while s<j+1:
                    t=0
                    if self.matchcheck (s,j)==True:
                        t=1+self.match(i,s-1)+self.match(s+1,j-1)
                        result.append(t)
                    s=s+1
                t=self.match(i,j-1)
                result.append(t)
            self.mdic[i,j]=max(result)
        return max(result)
```

Function B and C:

```

class matstack:
    def __init__(self,s):
        self.a=s
        self.dict={}
        self.stackdict={}
    def matchcheck(self,i,j):
        if (self.a[i]=="a" and self.a[j]=="u") or (self.a[i]=="g" and
self.a[j]=="u") or (self.a[i]=="c" and self.a[j]=="g") or (self.a[i]=="u"
and self.a[j]=="a") or (self.a[i]=="u" and self.a[j]=="g") or
(self.a[i]=="g" and self.a[j]=="c"):
            m=True
        else:
            m=False
        return m
    def stack(self,i,j):
        result=[]
        if (i,j) in self.stackdict:
            t=self.stackdict[i,j]
            result.append(t)
        else:
            if j-i+1<=2:
                t=-len(self.a)
                result.append(t)
            elif self.matchcheck(i,j)==False:
                t=-len(self.a)
                result.append (t)
            else:
                # self.matchcheck (i,j)==True:
                t=self.stack(i+1,j-1)+1
                result.append(t)
                t=self.match(i+1,j-1)
                result.append (t)
            self.stackdict[i,j]=max(result)
        return max(result)
    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+1:
                    t=0
                    if self.matchcheck (s,j)==True:
                        t=self.match(i,s-1)+self.stack(s,j)
                        result.append(t)
                    s=s+1
                t=self.match(i,j-1)
                result.append(t)
            self.dict[i,j]=max(result)
        return max(result)

```

Function D, E and F:

```

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") or (a[i]=="u" and a[j]=="a") or (a[i]=="u" and a[j]=="g") or
(a[i]=="g" and a[j]=="c"):
        m=True
    else:
        m=False
    return m
def twos (i,j):
    result=[]
    if j-i+1<2:
        t=-len(a)
        result.append(t)
    elif matchcheck(i,j)==False:
        t=-len(a)
        result.append(t)
    else:
        t=stack(i+1,j-1)+1
        result.append(t)
    return max(result)
def stack(i,j):
    result=[]
    if j-i+1<2:
        t=-len(a)
    if matchcheck(i,j)==False:
        t=-len(a)
        result.append(t)
    else:
        t=stack(i+1,j-1)
        result.append(t)
        t=match(i+1,j-1)
        result.append(t)
    return max(result)
def match(i,j):
    result=[]
    if j-i+1<4:
        t=0
        result.append(t)
    else:
        s=i+1
        while s<j:
            if matchcheck(i,j)==True:
                t=match(i,s-1)+twos(i,s-1)+twos(s,j)
                result.append(t)
            s=s+1
        t=match(i,j-1)
        result.append(t)
    return max(result)

```

```
class countstack:
    def __init__(self,s):
        self.cd=dict()
        self.td=dict()
        self.a=s
    def matchcheck(self,i,j):
        if (self.a[i]=="a" and self.a[j]=="u") or (self.a[i]=="c" and
self.a[j]=="g") or (self.a[i]=="u" and self.a[j]=="a") or (self.a[i]=="g"
and self.a[j]=="c"):
            m=True
        else:
            m=False
        return m
    def twos (self,i,j):
        if (i,j) in self.td:
            c=self.td[(i,j)]
        else:
            if j-i+1<4:
                c=0
            elif self.matchcheck (i,j)==False:
                c=0
            else:
                c=self.stack(i+1,j-1)
                self.td[(i,j)]=c
        return c
    def stack (self,i,j):
        if self.matchcheck(i,j)==False:
            c=0
        elif j-i+1<2:
            c=0
        else:
            c=self.count(i+1,j-1)+self.stack(i+1,j-1)
        return c
    def count(self,i,j):
        if (i,j) in self.cd:
            c=self.cd[(i,j)]
        else:
            c=0
            if j-i+1<5:
                c=1
            else:
                s=i+1
                while s<j:
                    c=c+self.count(i,s-1)*self.twos(s,j)
                    +self.twos(i,s-1)*self.twos(s,j)
                    s=s+1
                c=c+self.count(i,j-1)+self.twos(i,j-1)
            self.cd[(i,j)]=c
        return c
```