

Protein Secondary Structures Enumerations

–a summer project

Max Leung

Contents

1	Introduction	2
1.1	Proteins	2
1.2	Protein Secondary Structures	2
1.3	The Alpha(α) Helix	3
1.4	The Beta(β) Strand	3
2	Enumerations	4
2.1	What are we enumerating?	4
2.2	Alpha counting - Combinatorial Enumeration	5
2.3	Alpha counting - Dynamic Programming Enumeration	7
2.4	Beta counting - beta sheets	8
2.5	Beta counting - beta barrels	9
2.6	Combining the Alpha's and Beta's	10
2.6.1	Combining Alpha's with Beta sheets	11
2.6.2	Combining Alpha's with Beta barrels	12
2.7	Pure Alpha structures with different number of edges allowed . .	13
3	Conclusions and Improvements	14
	Appendix	16
	References	19

1 Introduction

1.1 Proteins

Proteins are polypeptide chains synthesized by the end-to-end joining of up to 20 amino acids. These amino acids form a chain which folds to give a three-dimensional structure. The folded structure can take many different shapes and each shape has a particular functional property which can vary from catalysing biochemical reactions (the enzymes) to carrying oxygen around the body (the hemoglobins).

The 20 amino acids can give rise to a vast number of possible sequences and hence possible foldings. In this project, we will mainly look at the secondary structures of proteins - structures which are formed when the amino acids join together. In particular, we will focus on the combinatorial theory involving these structures, trying to find out the number of ways to make up form a large legal secondary structure consisting of a certain number of elements.

1.2 Protein Secondary Structures

By secondary structures of proteins we are referring to structures which describe the general three-dimensional form of local segments of proteins. The most common protein secondary structures are alpha helices and beta sheets. Other helices such as the 3_{10} helix and π helix are two variations from the α helices. The former is more tightly coiled and the latter more loosely coiled than a general α helix. Both of them occur rarely as they are not energetically favorable due to their structural variations from an α helix. There are also turns and loose, flexible loops which link the more regular secondary structure elements.

In the paper by Kabsch and Sander¹, they adopted a code to describe these secondary structures. A list of the code is shown below:

- G = 3-turn helix (3_{10} helix).
- H = 4-turn helix (alpha helix).
- I = 5-turn helix (π helix).
- T = hydrogen bonded turn
- E = beta sheet with the strands arranged in parallel and/or antiparallel directions.
- ...

As we will only come across alpha and beta in this project, we choose to use the symbols α and β instead of H and E as the former are more descriptive.

¹Karsch, W. and Sander, C. Dictionary of Protein Secondary Structure: Pattern Recognition of Hydrogen-Bonded and Geometrical Features. *Biopolymers*, **22**, 2577-2637(1983)

1.3 The Alpha(α) Helix

The shape of an alpha helix is like a narrow-bore tube with its polypeptide backbone coiled up like a very tight clockwise screw thread. The amino acids in an alpha helix form hydrogen bonds which are parallel to the helical axis. The high amount of hydrogen bonds within the helix stabilises the structure so that it forms a very strong rod like structure. Alpha helix is said to have 3.6 residues since as you follow the helix around through 36 amino acid units, you will have made 10 complete 360° turns.

The pure alpha structures play a large part in the enumerations as we allow interactions between different alpha's in the structures whereas the beta's do not interact with the others. As we shall see, using combinatorial theory for enumerations is not efficient and it gets very complicated when the number of alpha's present is large. This urged us to develop a dynamic programming algorithm to count the possible structures instead.

1.4 The Beta(β) Strand

A beta strand consists of a stretch of amino acids whose peptide backbones are almost fully extended, resulting in a pleated structure which is normally about 5 to 10 residues long.

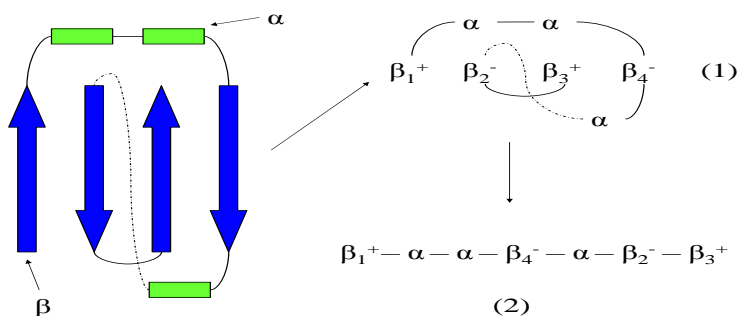
A beta strand can join other beta's to form either a pleated sheet or a barrel. In addition, there are two ways in which the beta's can interact to form a pleated sheet. The first way has the beta's all run in the same biochemical direction, from the amino terminal to the carboxy terminal, and we describe such a sheet as being parallel. Another way has the beta's running in alternating directions, from the amino terminal to the carboxy terminal in the first strand, then reverse the direction in the second strand, reverse again in the third strand, \dots , etc. They are described as anti-parallel sheets. For barrels, the beta's simply join together to form a circuit which has a three-dimensional shape of a barrel.

Our enumerations will take into account both the formation of beta sheets and beta barrels given a particular number of beta's. We will also include the effect of having different biochemical directions within a sheet and a barrel.

2 Enumerations

First of all we introduce a graphical representation for a general secondary structure involving only alpha's and beta's. Figure 1 shows an example of a secondary structure(SS) with 3 alpha's and 4 beta's.

Figure 1: Graphical representation of a SS with 3 alpha's and 4 beta's.



In the original (coloured drawing in figure 1) structure, the arrows represent the beta's and the blocks represent the alpha's. The arrow points from the amino terminal to the carboxy terminal in a beta strand. In our enumerations, we will regard all alpha's as indistinguishable units. The simplified linear representation (representation (2) in Figure 1) will be the one we will use throughout this project.

The advantage of using this representation is that the two-dimensional geometry is not lost, with the arcs joining each beta becomes a horizontal line and the connected beta's are put next to each other. We have introduced the + and - signs to specify the direction to which each beta is pointing. A beta strand pointing up is indicated by a + sign while one pointing down is indicated by a - sign. We can also tell the way in which the alpha's are connected to the beta's in the original structure from (2).

2.1 What are we enumerating?

Suppose we are given, in a secondary structure, the following:

- a - the number of α 's.
- b - the number of β 's.
- d - the number of edges each α is allowed to be connected with other α 's. (We assume that only two α 's can share an edge, but not two β 's or an α and a β .)

Our questions are: How many ways are there to arrange the alpha's and beta's? How will the number of ways change if we impose some restrictions on them? And how can we count the total number of legal structures under these restrictions?

Since we only allow interactions for alpha's, we can break down the problem into three sections: arranging the pure alpha structures, arranging the pure beta structures and inserting alpha's into pure beta structures. We will first count the pure alpha's as they turn out to be more complicated than expected due to the restriction on edges. Two approaches, one in combinatorics and the other in dynamic programming, will be used and as we shall see, using DP gives us more flexibility and greater efficiency.

2.2 Alpha counting - Combinatorial Enumeration

Given a , the number of α 's in a structure, we need to first find out the maximum number of edges allowed for the structure and we claim that $\lceil \frac{3a}{2} \rceil$ is the maximum such number, where $\lceil \cdot \rceil$ is the greatest integer function. The reason is that each alpha can have at most 3 edges incident on it and there can be as many as $\frac{3a}{2}$ edges. (We have to divide by 2 as each edge is linked to two α 's at the same time.) As the fraction may not be an integer, we take the greatest integer smaller than it. For example with 5 α 's, we can have $\lceil \frac{3 \times 5}{2} \rceil = 7$ edges.

To simplify our problem, we restrict ourselves to having one edge between two alpha's. Hence, if we let l be the number of possible edges in the whole pure alpha structure, we can imagine ourselves to be in a situation where we allocate l balls into $\sum_{i=1}^a (i-1)$ buckets and each bucket can only hold one ball at a time. We also suppose that we arrange a pure alpha structure in a line and we number them from left to right. We then name an edge between α_1 and α_2 as (1,2), α_1 and α_3 as (1,3) and so on. These () are the buckets. Now we can start our counting.

For $a = 2$, the alpha's are either connected or they are not, therefore we have 2 possible structures.

For $a = 3$, we can have a maximum of 3 edges within the structure. (Not 4 since having 4 edges will mean two alpha's will be connected by two edges and such a structure is not allowed.)

- With 0 edge, we have $\binom{3}{0}$ legal structure.
- With 1 edge, we have $\binom{3}{1}$ legal structures.
- With 2 edges, we have $\binom{3}{2}$ legal structures.
- With 3 edges, we have $\binom{3}{3}$ legal structure.

Hence in total we have $1 + 3 + 3 + 1 = 8$ possible structures.

For $a = 4$, we can have a maximum of 6 edges. We argue with similar reasoning as before that we have $\binom{6}{0} + \binom{6}{1} + \dots + \binom{6}{6} = 64$ possible structures.

This approach encounters the biggest problem when we have 5 or more alpha's. The reason is that we have at least 4 possible edges coming out of an alpha and we will have to take away the illegal structures from the total possible ones.

For $a = 5$, we can have at most 7 edges. With some structures have to be ruled out, our aim is to restrict ourselves to picking at most 3 buckets out of $(X, S), (X, T), (X, Y), (X, Z)$, where $X, Y, Z, S, T \in \{1, 2, 3, 4, 5\}$ when we have more than 3 edges.

- With l edges where $l = 0, 1, 2, 3$, we have $\binom{10}{l}$ possible structures. (In total there are $\binom{10}{0} + \binom{10}{1} + \binom{10}{2} + \binom{10}{3} = 176$ such structures.)
- With 4 edges, we have $\binom{10}{4} = 210$ possible structures but not all of them are legal. The illegal one are listed below:
 1. (1,2) (1,3) (1,4) (1,5)
 2. (1,2) (2,3) (2,4) (2,5)
 3. (1,3) (2,3) (3,4) (3,5)
 4. (1,4) (2,4) (3,4) (4,5)
 5. (1,5) (2,5) (3,5) (4,5)

Therefore we have $210 - 5 = 205$ legal structures with 4 edges.

- With 5 edges, we have to eliminate structures which contain substructures listed in the previous case. Each such structure gives rise to 6 illegal structures. Therefore we have $\binom{10}{5} - 5 \times 6 = 222$ legal structures.
- Similarly, for 6 and 7 edges, we found the number of legal structures to be 135 and 30 respectively.

Hence in total we have $176 + 205 + 222 + 135 + 30 = 768$ legal structures with 5 alpha's.

It appears that combinatorial counting is easy and manageable up to 5 alpha's. However, as we proceed to more alpha's in the pure structure, counting becomes very difficult as it is very difficult to establish a systematic way to identify all the illegal structures and thus eliminate them from our enumerations. Hence, with the help from Dr. Lygnsø², we developed a simple counting tool using dynamic programming.

²Dr. Rune Lygnsø, Department of Statistics, University of Oxford

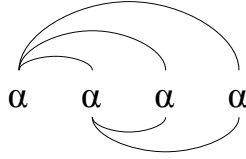
2.3 Alpha counting - Dynamic Programming Enumeration

The following definitions help us define the tools necessary for the counting:

Definition 2.1 Define (e_1, e_2, \dots, e_a) to be the vector of degrees of α 's in a pure alpha secondary structure, where e_i are the number of edges incident on α_i for $i = 1, 2, \dots, a$. We call this the **degree vector**.

Definition 2.2 Define $[e_1, e_2, \dots, e_a]$ to be the number of pure alpha structures which have degree vector (e_1, e_2, \dots, e_a) .

Figure 2: Pure alpha structure with degree vector $(3, 3, 1, 1)$.



For example, the degree vector of the above structure is $(3, 3, 1, 1)$. In fact, the degree vector tells us how many edges are connected to a particular alpha in the structure.

We can now build up a recursion relation, taking into account the degree 3 restriction in the formula. Any structure which is illegal (e.g. with some illegal degree vector (e_1, e_2, \dots, e_k)) will have $[e_1, e_2, \dots, e_k] = 0$. So when we add another alpha to such a structure, the new structure will still be disallowed and not counted.

For $a = 1$ (one-alpha structure), we have $[0] = 1, [1] = [2] = [3] = 0$.

For $a = 2$, we add another α to the existing legal structures. The recursion formula from $a = 2$ is as follows:

- $[e_1, e_2, \dots, e_{r-1}, 0] = [e_1, e_2, \dots, e_{r-1}]$
- $[e_1, e_2, \dots, e_{r-1}, 1] = \sum_{i=1}^{r-1} [e_1, \dots, e_{i-1}, e_i - 1, e_{i+1}, \dots, e_{r-1}]$
- $[e_1, e_2, \dots, e_{r-1}, 2] = \sum_{\substack{i,j=1 \\ i < j}}^{r-1} [e_1, \dots, e_i - 1, \dots, e_j - 1, \dots, e_{r-1}]$

- $[e_1, e_2, \dots, e_{r-1}, 3] = \sum_{\substack{i,j,k=1 \\ i < j < k}}^{r-1} [e_1, \dots, e_i - 1, \dots, e_j - 1, \dots, e_k - 1, \dots, e_{r-1}]$
- $[e_1, e_2, \dots, e_{r-1}, l] = 0$ for all $l > 3$ or $l < 0$

For example, we have $[1, 1] = [0, 0] = [0] = 1$ and $[1, 1, 2] = [0, 0] = 1$.

Remarks:

1. The sum of all terms in the a degree vector must be even, otherwise the number of possible structures with such a configuration is zero.
2. The problem of double looping the same two alpha's have been taken into account because we disallowed the configurations $[2, 2]$ and $[3, 3]$. This means that double (or even triple) looping is not allowed in later stages.
3. The information about how the α 's are joined is not revealed in the degree vector. e.g. Given $[e_1, e_2, \dots] = [3, 2, \dots]$, we may not be able to work out how the α 's are joined unless we have very few α 's. However, as far as only enumerations are concerned, such a loss in information will not bring many problems to us.

Dr. Lyngsø has written a python program called **count_graphs.py N** (the program code is available in the appendix) where N is the number of alpha's in a pure structure. It gives the number of legal structures from 2 to N when run. The following table displays the results when we run N=10: (for N=1, the number is obvious and we do not need the program to enumerate)

N	1	2	3	4	5
No. of structures	1	2	8	64	768

N	6	7	8	9	10
No. of structures	12068	2.37×10^5	5.65×10^6	1.60×10^8	5.28×10^9

Table 1: Number of distinct legal structures with different number of α 's

As we can see from the table, the number of legal structures can reach as many as 2.37×10^5 with 7 alpha's. In fact, it takes the computer a long time (nearly 2 minutes) to calculate the number for 10 alpha's. The number of structures enumerated for 5 alpha's, 768, is the same as what we obtained from our combinatorial approach. Yet it is much quicker to enumerate using dynamic programming.

2.4 Beta counting - beta sheets

We now start counting our pure beta structures in which the beta strands join together to form sheets. Let b be the number of beta's in a pure beta sheet and

each beta can take one of two possible biochemical directions (either pointing up or pointing down). Arranging the beta's in a sheet is equivalent to constructing a list of length b where the elements of the list are the b beta's. Hence there are $b!$ ways to arrange these beta's.

In addition, as each beta can take two different directions, we need to multiply $b!$ by 2^b . The total number of possible sheets with b beta's is thus $2^b(b!)$. Table 2 summarises the number of possible pure beta sheets from $b=1$ to 10.

N	1	2	3	4	5
No. of sheets	2	8	48	384	3840

N	6	7	8	9	10
No. of sheets	46080	6.45×10^5	1.03×10^7	1.86×10^8	3.72×10^9

Table 2: Number of distinct beta sheets formed with different number of β 's

By these enumerations we have ruled out the possibilities for the beta's to form barrels since there is an implicit assumption in our enumerations which is the linear representation of the beta's cannot form a closed circuit, i.e. the numbers above are the number of possible structures which form beta sheets.

2.5 Beta counting - beta barrels

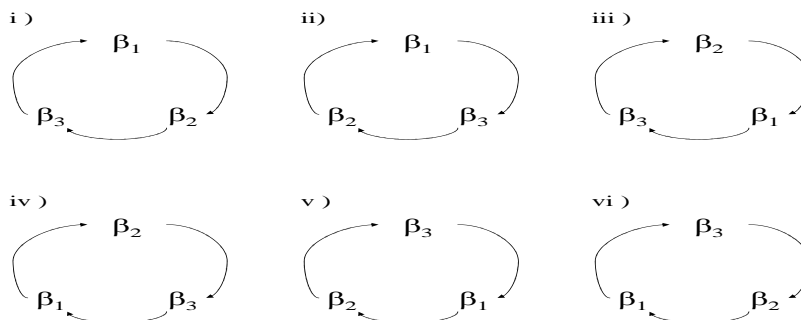
Forming a barrel simply means we join the leftmost and the rightmost beta to form a closed circuit. The +/- signs of the beta's do not affect the formation of the circuits because barrels with all beta strands in the same biochemical directions are allowed³, provided they are properly folded.

Suppose we have b beta's in a pure beta structure and each can take on of the two possible biochemical directions as before. We first ignore the +/- signs for the moment. Joining beta's up to form a barrel is equivalent to arranging them in b positions in a circle, e.g. with 3 beta's, there are 6 ways to allocate them in 3 positions. (See Figure 3.) Note that we used arrows instead of lines to join the beta's since two barrels are identical if they represent the same permutation. For example i), iv) and v) in Figure 3 all represent the permutation (1,2,3) whereas ii), iii) and vi) represent (1,3,2). Hence we only have 2 distinct barrels with 3 beta's. If we take into considerations the biochemical directions of the beta's, we have $2 \times 2 = 4$ distinct barrels.

In general, with b beta's, we can have $b!$ permutations but only $\frac{b!}{b} = (b-1)!$ of them are distinct (we have to divide by b since for each of the permutations, there are b ways to start). Hence, given b beta's in a pure beta structure, there are $2^b(b-1)!$ distinct barrels which can be formed. Table 4 tells us how many ways there are to form barrels given b beta's, with b running from 1 to 10.

³See page 48, Fig. 4.1(a), Introduction to Protein Structures by Branden and Tooze

Figure 3: Allocating 3 beta's to three positions in a circle.



N	1	2	3	4	5
No. of barrels	2	4	16	96	768

N	6	7	8	9	10
No. of barrels	7680	92160	1.29×10^6	2.06×10^7	3.72×10^8

Table 3: Number of distinct beta barrels formed with different number of β 's.

Hence, the number of barrels formed with b beta's is always b times less than the number of sheets that can be formed. We are now equipped with enough tools to combine the alpha's and beta's to give mixed alpha-beta structures.

Note: The combinatorial approaches for counting both beta sheets and beta barrels are simple enough that we do not have to construct a dynamic programming algorithm for counting.

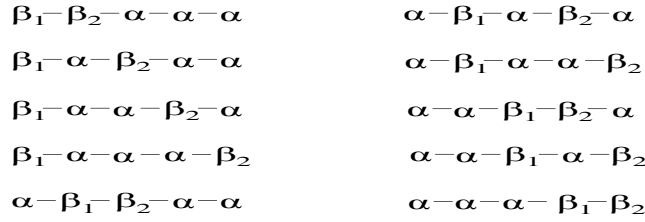
2.6 Combining the Alpha's and Beta's

We have already known how many pure alpha and pure beta structures there are, given the number of the corresponding elements. We now put the two together, with the total number of alpha's and beta's known in a mixed alpha-beta structure. Our approach to counting will be to fix the number of beta's while we add more and more alpha's to the structure. As before, we will regard different beta's at different positions in a chain (linear arrangement) as different and they are thus numbered, whereas the alpha's will remain to indistinguishable from each other. (Note that in reality, the alpha's are very likely to be non-identical.) We will also separate the combination into two cases - one for the beta sheets and one for the beta barrels.

2.6.1 Combining Alpha's with Beta sheets

First let us take a look at how many possible structures (where the beta's form sheets) there are with 3 alpha's and 2 beta's. As we regarded $\beta_1 - \beta_2$ and $\beta_2 - \beta_1$ as different structures, figure 4 shows structures which are in the form $\beta_1 - \beta_2$ combined with the alpha's.

Figure 4: Combining 3 alpha's and 2 beta's.



There are 10 such structures and as there are $2!$ ways to rearrange the beta's (as we have two beta's, β_1 and β_2), giving us in total $2! \times 10 = 20$ ways to construct an alpha-beta structure (in sheet) with 3 alpha's and 2 beta's.

In general, with a α 's, b β 's, ignoring the α -interactions and β -directions, we have $\binom{a+b}{b} \times b! = \frac{(a+b)!}{a!}$ ways of combining them.

Adding the alpha interactions will mean that we have to multiply $\frac{(a+b)!}{a!}$ by the number of possible structures with a alpha's as enumerated in Table 1. On top of it, adding the beta directions will mean that we have to multiply the total by a further 2^b .

Table 4 gives us the numbers of possible structures (with beta's forming sheets) for $a, b = 1, \dots, 5$.

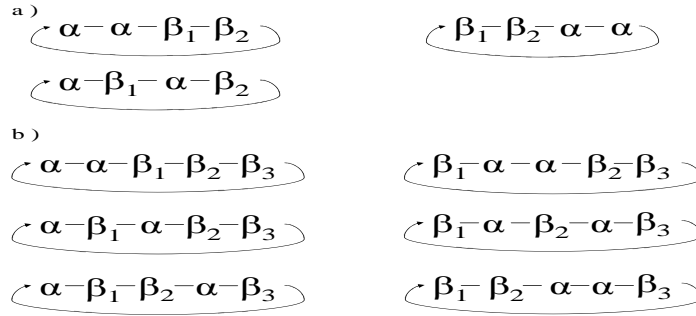
a\b	1	2	3	4	5
1	4	48	1152	46080	2.76×10^6
2	12	192	5760	2.76×10^5	1.94×10^7
3	64	1280	46080	2.58×10^6	2.06×10^8
4	640	15360	6.45×10^5	4.13×10^7	3.72×10^9
5	9216	2.58×10^5	1.24×10^7	8.92×10^8	8.92×10^{10}

Table 4: Number of mixed alpha-beta structures with different number of α 's and β 's (with the beta's forming sheets.)

2.6.2 Combining Alpha's with Beta barrels

The combination of alpha's and beta barrels is similar to that of alpha's and beta sheets. We have to consider how many distinct circular structures there are given a alpha's and b beta's. Figure 5 shows us there all the ways of joining (a) 2 alpha's and 2 beta's and (b) 3 alpha's and 2 beta's to give barrels.

Figure 5: Combining (a) 2 alpha's and 2 beta's and (b) 3 alpha's and 2 beta's.



There are 3 ways for (a) and 6 ways for (b). In general, given a and b , we have $\binom{a+b-1}{a}$ distinct ways to form barrels. As before, we have to multiply $\binom{a+b-1}{a}$ by the number of possible structures a alpha's from Table 1 and a further 2^b (to consider the +/- signs of the beta's.)

Table 5 gives us the number of structures (with beta's forming barrels) for $a, b = 1, \dots, 5$.

a\b	1	2	3	4	5
1	2	8	48	384	3840
2	4	24	192	1920	23040
3	16	128	1280	15360	2.15×10^6
4	128	1280	15360	2.15×10^6	3.44×10^7
5	1536	18432	2.58×10^6	4.12×10^7	7.43×10^8

Table 5: Number of mixed alpha-beta structures with different number of α 's and β 's (with the beta's forming barrels.)

Unsurprisingly, the number of distinct barrels that can be formed with the same number of alpha's and beta's is less than the number of distinct sheets, since being a circular allows less distinct structures to be formed as there are many repeated structures due to symmetry.

2.7 Pure Alpha structures with different number of edges allowed

A slight change to the `count_graphs.py` `N` program will allow us to enumerate the total number of pure alpha structures formed with different number of edges allowed to incident on a single alpha. For example, we change the condition to allowing 2 edges only will give us the following results:

N	1	2	3	4	5
No. of structures	1	2	8	41	253

N	6	7	8	9	10
No. of structures	1858	15796	1.52×10^6	1.64×10^7	1.95×10^8

Table 6: Number of distinct legal structures with different number of α 's, allowing for 2 edges only.

It is also possible to increase the number of edges allowed by modifying the program. However, the complexity arises when we are to write a general program $N(a, d)$ which gives us the total number of legal pure alpha structures given a , the number of alpha's and d , the number of edges allowed, as the algorithms used in the original program will need to be converted into a recursion instead and it takes more than the time we have to complete this project, although it is anticipated that the results to be increasing with d .

3 Conclusions and Improvements

Although we have been trying to be as realistic as possible in this project given the amount of time available, there are many areas which we can improve on and further work could be done.

Some shortcomings found in our enumerations include:

1. the loss of information using the dynamic program in pure alpha structures enumerations. If we are to construct any evolution model based on the enumerations, the exact alpha-alpha interactions must be known in the secondary structures.
2. the inclusion of many structures in our enumerations which may not exist. When we enumerated the pure alpha structures, the only constraint we had on our structures was that each alpha can only interact with a certain number of other alpha's. This is not sufficient to identify the structures which may not exist in nature. For example, a structure with 10 alpha's and 60 edges amongst them can be too complicated that it may be absent from nature.

Further restrictions such as how far apart two alpha's can be joined, which of the alpha's in a structure may/may not be joined, etc. will help us narrow the scope of the problem that we are looking at.

3. the assumption that all alpha's are identical is not entirely realistic. In reality, all the alpha's should be considered distinct. The combinatorial approach that we used could be extended to distinguishable alpha's provided the a more general formula can be found. The dynamic programming does in theory take into account the non-identical property of the alpha's.

Another problem arising from this arises when we carry out the combinations. We assumed that the alpha's are identical so we were combining the beta barrels with the alphas, symmetry eliminates many repeated figures. If the alpha's were considered distinct, the combination process would have to be amended accordingly.

The three types of structures that we enumerated - pure alpha, pure beta and alpha-beta mixed - are usually important base units of building large proteins. They are then combined (and folded if we are to consider the differences between different proteins built from the same units). Examples of such basic units include

- for pure beta structures: the Greek key motifs, the hairpin motif and the jelly roll motif.
- for pure alpha structures: the four-helix bundle which is a common domain structure in α proteins.

- for alpha-beta structures: Alpha/Beta barrels which occur in many different enzymes.

If we are to further our studies in protein secondary structures, a possible direction will be on how to predict a tertiary structure given a secondary structure, which is predicted from the primary structure (the amino acids sequence of the protein concerned).

In this project, we tried to solve a graph counting problem with both combinatorics and dynamic programming. The enumerations raised many questions which have not been addressed in combinatorics before. Counting with such a number of restrictions has made pure combinatorial theory insufficient and inefficient in solving the problems. Yet we could see the trade-off between information and efficiency when we used different approaches - the loss of information is compensated by the greater efficiency in counting and vice versa.

Appendix

Code for count_graphs.py N by Dr. Rune Lyngsø

```
#!/usr/bin/env python
from sys import argv, exit

if len(argv) != 2:
    print "Usage: %s N\n where N is number of nodes" % argv[0]
    exit(1)
n = int(argv[1])

def index(entry):
    return reduce(lambda x, y: 4 * x + y, entry, 0) / 2

def entry(index):
    def foo(i):
        if i == 0:
            return []
        else:
            return foo(i / 4) + [i % 4]
    e = foo(2 * index)
    if sum(e) % 2 != 0:
        e = e[:-1] + [e[-1] + 1]
    return e

def size(n):
    return n * (n - 1) / 2

def fourpower(n):
    if n == 0:
        return 1
    else:
        return 4 * fourpower(n - 1)

c = [1]
for i in range(1, n):
    d = (fourpower(i + 1) / 2) * [0]
    for e in range(len(c)):
        if c[e] == 0:
            continue
        # Add no edges to new node
        d[e] = c[e]
        count = c[e]
        # Add one edge to new node
        e = entry(e)
```

```

if len(e) > i + 1 or index([0] + e) != index(e) or entry(index(e)) != e:
    print "Foo"
    exit(1)
e = (i - len(e)) * [0] + e
for j in range(i):
    if e[j] < 3:
        d[index([1] + e[:j] + [e[j] + 1] + e[j + 1:])]
        = d[index([1] + e[:j] + [e[j] + 1] + e[j + 1:])] + count
# Add two edges to new node
for j in range(i):
    if e[j] < 3:
        for k in range(j):
            if e[k] < 3:
                d[index([2] + e[:k] + [e[k] + 1] + e[k + 1:j] + [e[j] + 1]
                + e[j + 1:])]
                = d[index([2] + e[:k] + [e[k] + 1] + e[k + 1:j] + [e[j] + 1]
                + e[j + 1:])] + count
# Add three edges to new node
for j in range(i):
    if e[j] < 3:
        for k in range(j):
            if e[k] < 3:
                for l in range(k):
                    if e[l] < 3:
                        d[index([3] + e[:l] + [e[l] + 1] + e[l + 1:k] + [e[k] + 1]
                        + e[k + 1:j] + [e[j] + 1] + e[j + 1:])]
                        = d[index([3] + e[:l] + [e[l] + 1] + e[l + 1:k] + [e[k] + 1]
                        + e[k + 1:j] + [e[j] + 1] + e[j + 1:])] + count

c = d
print "Number of distinct graphs with", i + 1,
"nodes and degree bounded by 3 is", sum(c)
exit(0)

# Alternative implementation running through all graphs - slow
def increment(v):
    try:
        i = 0
        while v[i][1] == 1:
            v[i][1] = 0
            i += 1
        v[i][1] = 1
    except IndexError:
        return None
    return v

def check(v):

```

```
c = n * [0]
for i in v:
    if i[1] == 1:
        c[i[0][0]] += 1
        if c[i[0][0]] > 3:
            return False
        c[i[0][1]] += 1
        if c[i[0][1]] > 3:
            return False
    return True

v = []
for i in range(n):
    for j in range(i):
        v.append([(i, j), 0])
c = 0
while v != None:
    if check(v):
        c += 1
    v = increment(v)
print c
```

References

- [1] Branden, C. and Tooze, J. *Introduction to Protein Structure*, Garland, 1991
- [2] Karsch, W. and Sander, C. Dictionary of Protein Secondary Structure: Pattern Recognition of Hydrogen-Bonded and Geometrical Features, *Biopolymers*, **52**, 2577-2637
- [3] Patthy, L. *Protein Evolution*, Blackwell Science, 1999
- [4] Taylor, W. R. and Aszódi, A. *Protein Geometry, Classification, Topology and Symmetry*, Institute of Physics, 2005
- [5] Anderson, I. *A First Course in Combinatorial Mathematics*, OUP, 1989
- [6] Aigner, M. and Ziegler, G. M. *Proofs from THE BOOK*, Springer, 1999
- [7] Marshall Hall, Jr. *Combinatorial Theory*, Wiley-Interscience, 1986
- [8] Lint, J. H. and Wilson, R. M. *A Course in Combinatorics*, Cambridge, 1998
- [9] Tomescu, I. and Melter, R. A. *Problems in Combinatorics and Graph Theory*, Wiley-Interscience, 1985