Honors Project

Introduction

Genome projects are producing almost an infinite amount of molecular data. These genomic sequences present molecular information constructing current organisms. They are also the results of long molecular evolution since the origin of life. The genomic sequences are, therefore, filled with evolutionary "footprints." Interestingly, recent research has indicated that the amino acid composition of the protein sequences of these organisms is a good method for being able to discriminate between proteins that have different function.

The goal of this project is to create a program that will calculate the frequency of appearance of individual building blocks of protein sequences, amino acids. Beginning with a set of "positive" data, which consists protein sequences known to be from a particular protein family, the G protein-coupled receptors (GPCRs), the program will calculate the average frequencies of the amino acids. This phase is *training* the program to recognize the amino acid frequencies of the target protein family.

The next phase of the program is to take another set of protein sequences, and attempt to find sequence that are similar to those upon which it trained, based on the amino acid frequencies. The algorithm can then predict which sequences it believes to be of the same type.

This project will be incremental, and each section will build on top of the previous work. When this project is completed, the goal is to have a program that can predict with good accuracy whether an unknown protein sequence is of the type the algorithm trained on. If you are unclear on any phase of this project, DO NOT BE AFRAID TO ASK FOR CLARIFICATION. I want you to succeed in this project.

Basics

In order to accomplish this task, some basics of biological sequences must first be covered.

G protein-coupled receptors (GPCRs)

G-protein coupled receptors (GPCRs) form a large super-family of proteins composed of three major classes and more than 30 subfamilies. They are integral membrane proteins characterized by seven membrane- spanning (transmembrane; TM) regions. They are involved with signal transduction across cell membranes. Many medically and pharmacologically important proteins are included in this super-family.

Amino Acids

In order to calculate the amino acid frequency, you need to be familiar with the amino acids themselves. Table **??** is a listing of the amino acid name, the three letter abbreviation, and the one letter abbreviation of the set of 20 amino acids that make up the protein alphabet.

Table 1: The 20 amino acids that are the building blocks of protein sequences, along with the abbreviations commonly used to specify them.

Alanine	ala	A
Arginine	arg	R
Asparagine	asn	N
Aspartic Acid	asp	D
Cysteine	cys	C
Glutamine	gln	Q
Glutamic Acid	glu	E
Glycine	gly	G
Histidine	his	H
Isoleucine	ile	I
Leucine	leu	L
Lysine	lys	K
Methionine	met	M
Phenylalanine	phe	F
Proline	pro	P
Serine	ser	S
Threonine	thr	T
Tryptophan	trp	W
Tyrosine	tyr	Y
Valine	val	V
Other	oth	

Project

Part I

Due: *Monday, November 14^{th}, 2005*

The first task in the creation of the data structures necessary for the algorithm, as well as the initial reading in of the data sets. The tasks necessary for this section are:

- Data structures: The data structures that will be necessary for the program will include
  1. An array named `train_aa_freq[]` of type `double` that will be used to calculate the amino acid frequencies of the training data set.
  2. Two arrays for the test data set, `char test_id[][]` and `double test_aa_freq[][]`. These will be parallel arrays that will keep track of the protein sequence id and the amino acid frequencies of the sequence. For example, `test_id[1]` will contain the full name of a sequence, call it `gpcr-1`, where `test_id[1][0]` will be the character `g`. The parallel array `test_aa_freq[1][0]` through `test_aa_freq[1][19]` will contain the frequencies of the amino acids in `gpcr-1`.
  3. An array named `double sum_of_squares[]` that is also parallel to the previous arrays. This array holds the similarity value that we are testing.
- For this first part, the following items should be accomplished:
  1. Enumerating the different types of amino acids. Enumerate these types using the three letter code, using the order given in Table **??**.
  2. Declaring the previous arrays as global arrays (declared outside of the `int main` function, before the function prototypes. Define global variables `NUM_TRAIN` and `NUM_TEST` with the values 100 and 10, respectively. Define one more global variable `MAX_SEQ_SIZE` that holds the longest sequence in either the training or test set, set it to 500.
  3. Create a function `void Initialize_Arrays()` that initializes the arrays above to initial values. The character array values should be initialized to the null character, '`\0`'. The other arrays should be initialized to zero.
  4. Make a function `int AA(char aa)` that takes the input of a single character and returns the integer value that is associated with it. For example, on input 'A', the function should recognize the letter 'A', and return the enumerated type value `ala`.
  5. Read in the training data set, `train.dat`, and calculate the amino acid frequencies of these sequences. Do this in a function named `void Read_Sequences(double train_aa_freq[])`. In order to accomplish this, you must:
     - Open the file, using a file pointer (`FILE *in`).
     - The file format of `train.dat` is as follows:
       * A line beginning with the character '>' is a sequence id line. It signifies that all characters beginning with the character following the '>' to the endline ('`\n`') are the name of a sequence.

    - Following each sequence id line is the amino acid sequence, where there are 60 amino acids per line. For example, an amino acid sequence that is 200 amino acids long will span 4 lines in the file.
  - Read in the sequences. This can be accomplished using a while loop, as follows in pseudocode:

```
while((fscanf(in,"%c",&aa) != EOF) {
  1. If line starts with '>', read until next '\n'.
  2. else increment train_aa_freq with the value of aa.
     Increment the amino acid counter.
}
```

6. In the end, divide all of the amino acids counted by the amino acid counter. This will give the amino acid frequencies in percentage. Mathematically,

$$\sum_{i=0}^{19} \texttt{train\_aa\_freq[i]} = 1.$$

- Print the amino acid frequencies to the screen.

This is the end of the first part!

Part II

Due: *Monday, November 28th, 2005*

You now have the data from the "training" set of amino acids. The training data is a set of sequences that we are interested in, the GPCRs. We know that every sequence in the training data set is a GPCR because they have been experimentally validated as GPCRs. Thus, this training data set should give us a good approximation for the underlying amino acid frequencies for other GPCR proteins.

Since we have an idea of the approximate amino acid frequencies of GPCRs, we now would like to use this information to try and predict GPCR sequences from a set of unknown protein sequences. This set of unknown protein sequences is referred to as the "test set". There are 10 protein sequences in the test set, "`test.dat`".

Part II of this project involves the reading in and handling of the test set. This new data set consists of 10 sequences, of which we know there is at least one GPCR sequence. The first order of business with this data set is to read in the data. To do this:

- The handling of the protein sequence names for the test data set is different than the procedure for the training data set. This is because we will need to refer back to these names when we predict whether or not the sequence is a GPCR. Therefore, we must read in the names of the sequences, into the array `char test_id[][]`. This is a two-dimensional array because there are 10 sequences, and each sequence has a name. So for example, if we have two sequences `test_1` and `test_2`, the array `test_id` array will look like:

t	e	s	t	_	1
t	e	s	t	_	2

where `test_id`[0][3] `= 't'`, and `test_id`[1][5] `= '2'`.

- After reading in the sequence id, we now need to read in the sequence, and find the amino acid composition of the sequence. Notice that we are no longer pooling the amino acid frequencies. Each test sequence *must* have it's own amino acid frequency, otherwise we cannot make a prediction whether this sequence is a GPCR. Because of this, we must also have a two-dimensional array to keep track of the amino acid frequencies of each of the test sequences. This array, `test_aa_freq[][]` will be a parallel array with the array `test\_id`. The first subscript for this array will denote the test sequence (1...10) that the amino acid frequencies in the second subscript apply to. For example, the reference `test_aa_freq[6][10]` references the frequency of the amino acid serine:

0	1	2	3	4	5	6	7	8	9	10	...
A	R	N	D	C	Q	E	G	F	P	S	...

  for the seventh sequence in the test data set. seventh sequence in the test set.

After you are done reading in all of the sequences, you will print out the 10 sequence id's, along with the corresponding amino acid frequencies for these sequences. This is the end of Part II!

Hint: When printing the sequence id, it is easier to print a string, rather than character to character. This can be accomplished for the `test_id[][]` *array by using the command* `printf("%s",test_id[1]);` *; in this statement, you are printing the sequence id of the second sequence in the test set.*