Entropy and Information Content of Peptide Libraries

Aviv Keshet
(Dated: May 17, 2007)

PROJECT DESCRIPTION

This project attempts to analyze the entropy and information content of peptide libraries, and in particular to study the dependence of the information content of a subsequence on the subsequence length. The approach is outlined below. Further details on the code are included in the final section.

1. The peptide library is sliced, generating a database of all subsequences of length n. As there are 20 amino acids, the number of possible subsequences is 20^n.

2. The number of repetitions N_α of each subsequence α is determined. For each subsequence, the probability of drawing that subsequence from the slice database is estimated as $p_\alpha = N_\alpha/N$ where $N = \sum_\alpha N_\alpha$.

3. The entropy of the subsequence distribution is estimated as $S = -\sum_\alpha p_\alpha \ln(p_\alpha)$.

If the peptide library were randomly generated and infinitely long, with each amino acid having an equal probability, then the entropy of any subsequence distribution would be $-\ln(1/20) = 2.995$ time the subsequence length.

There are some pitfalls in attempting to estimate entropy this way, particularly for large subsequence lengths. Analysis has been attempted for subsequences up to length 20. A peptide library length is on the order of 10^6, whereas there are $20^{20} \approx 10^{26}$ amino acid sequences of that length. Thus, the number of drawn samples will be astronomically smaller than the number of possible sequences. If the peptide library were random, this method of computing entropy would provide a vast underestimate for long subsequence lengths. These issues will be addressed in below.

RESULTS: (20).E-COLI.SAKAI

The steps discussed above were applied to the (20).E-coli.Sakai peptide library, obtained from `ftp://ftp.ebi.ac.uk/pub/databases/integr8/fasta/proteomes/`. The length of this library is 1.6 million amino acids. The entropy of probability distributions of subsequences of each length (calculated by the prescription above) are depicted in Fig 1.

For short subsequences, the entropy is nearly constant and equal to 2.885 per length, somewhat less than that of a random database (2.995 per length). As is evident from the behavior of this estimate at long sequence lengths, which appears to "saturate" at length 6, this estimate suffers from the finite database size. For subsequences longer than 6, the vast majority of subsequences appear only once in the slice database (Fig 3). Nevertheless, there is indication that even for longer sequences, the peptide library is non-random. The fraction of singlets appears to saturate rather than approach 1 (as would be expected from a random database), and likewise a histogram of subsequence frequencies appears to approach a distribution which indicates enhanced frequency of certain subsequences (Fig 2).

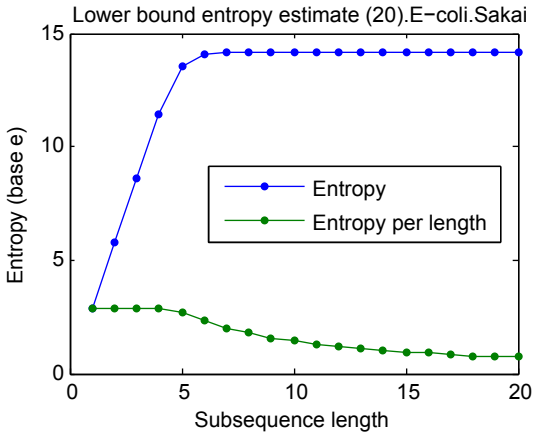

FIG. 1: Lower bound estimate of entropy.

To assess the reliability of our estimate of entropy for long sequences, it is instructive to try to estimate an upper bound for the entropy. For long sequence lengths, the majority of sequences in the database are singlets, but we can over-estimate the entropy by assuming that these singlets have the same "probability" as the huge set of possible subsequences that were not drawn. If f_1 is the number of singlet subsequences, then an upper bound estimate of entropy is provided by

$$S^{\text{upper}} = \frac{nf_1}{N} \times S_1 - \sum_{\alpha, N_\alpha \neq 1} p_\alpha \ln(p_\alpha) \qquad (1)$$

where S_1 is the entropy of 1-length subsequences. This upper bound effectively comes from estimating that all

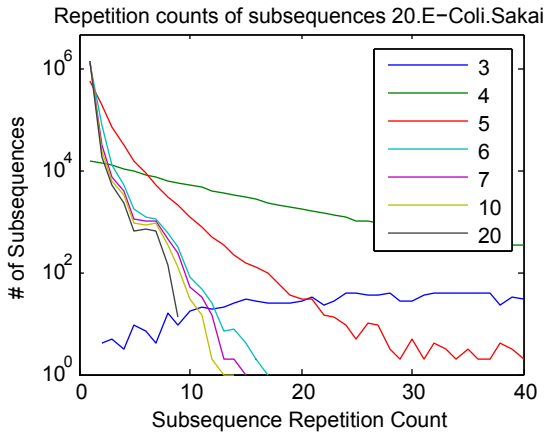

FIG. 2: Histograms of subsequence frequency (semilog plot). Legend indicates subsequence length.

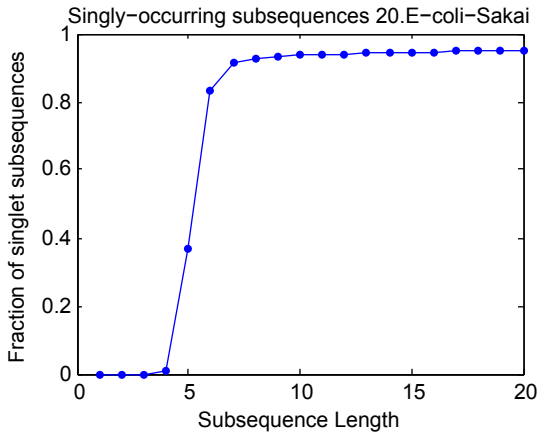

FIG. 3: Fraction of subsequences which appear only once in the slice database ("singlets").

singlet subsequences belong to a background distribution with the same entropy per length as 1-length subsequences. A comparison between the upper and lower bound estimates is shown in Fig 4. The upper bound begins to appreciably diverge from the lower bound for subsequences around length 6.

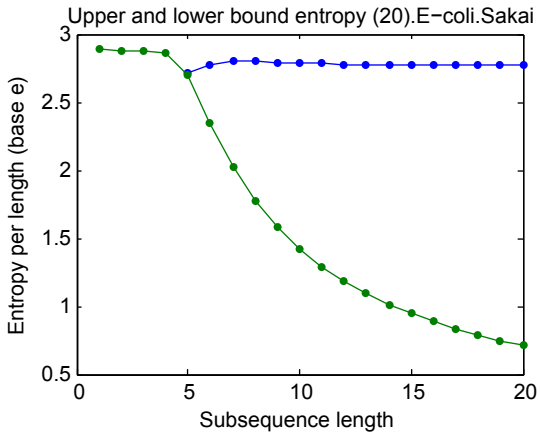

FIG. 4: Upper and lower bound entropy estimates.

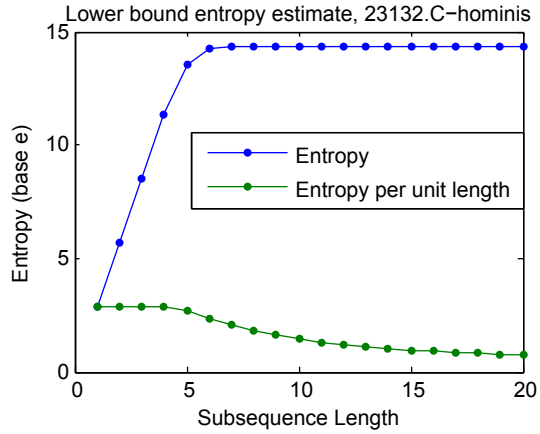

FIG. 5: Lower bound estimate of entropy.

RESULTS: (23132).C-HOMINIS

As a consistency check, we apply the above analysis to a second peptide library, 23132.C-hominis. The data is plotted in Figs 5, 6, 7, 8. The entropy of 1-length subsequences is slightly higher than in the E. Coli library, 2.859. Results seem qualitatively similar to the E. Coli library, aside from the fact that for long sequences, this library seems to approach all singlets (Fig 7) unlike in E. Coli.

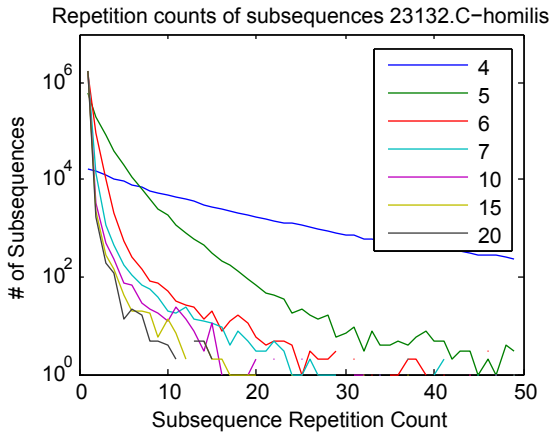

FIG. 6: Histograms of subsequence frequency (semilog plot). Legend indicates subsequence length.

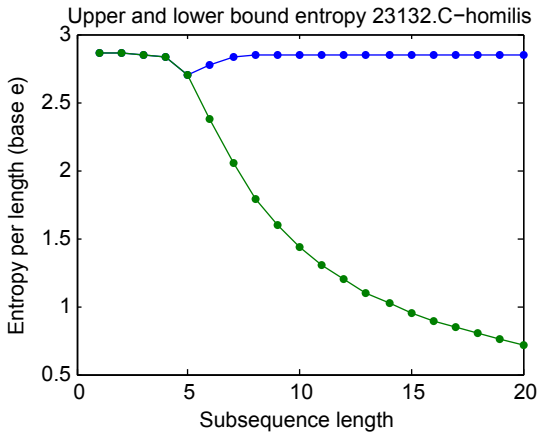

FIG. 8: Upper and lower bound entropy estimates.

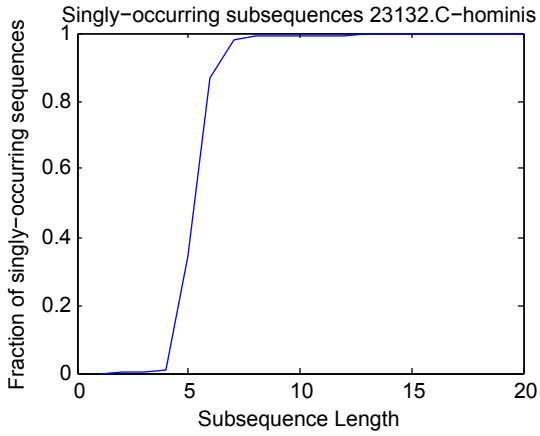

FIG. 7: Fraction of subsequences which appear only once in the slice database ("singlets").

CONCLUSIONS

The method used here to determine entropy/information content of peptide subsequences has yielded sensible results for short sequences (shorter than length 6 or so). Beyond this length, the length of the libraries is too short for them to provide a good sampling of the "subsequence space". Indeed, it is not even clear that the concept of subsequence entropy and subsequence space is well defined in this regime. For the length range of 1-4 amino acids, subsequence entropy per length appears to be independent of length.

Estimates of the upper and lower bound of entropy begin to diverge appreciably above subsequence length 5, indicating that the two entropy estimates used in this project are not very reliable above this threshold. Nevertheless, in both libraries studied, there appeared to be a significant drop in entropy per length for subsequences of length 5. This may be an indication that there are certain favored 5-length peptide subsequence "modules" that are used throughout the proteome. This apparent drop occurs at the threshold length beyond which the entropy estimate becomes unreliable. This could be interpreted as an indication that the proteome evolved from uniformly containing all subsequences of that length to one which makes repeated uses of the subsequence modules found to be useful.

Finally, we can compare the results of the two peptide libraries studied. In the E-coli library, there seemed to be a saturation in the fraction of singlet subsequences (see Fig 3), while this did not occur in the C-hominis peptide library (Fig 7). This may be an indication that certain peptides have repeated occurrences in the E-coli library (either identically or with high similarity), while this does not appear to be the case for the C-hominis library.

ALGORITHM TECHNICALITIES

This section outlines the steps undertaken to convert a .fasta peptide library into a subsequence frequency count. The frequency count data is then analyzed using Matlab.

First, a nawk script is used to remove the headers (which separate peptides in the library and provide de-

scriptions) and replace them with a * character:

```
{if ($1 ~ />/) print "*";
 else print $0}
```

A nawk script is then used to remove all newline characters from the file.

```
:a;N;$!ba;s/\n//g
```

A sed script is used to replace all * characters with newlines.

```
s/*/\
/g
```

A nawk script is used to chop all the subsequences of a given length.

```
{for (i=1;i<=length($0)-(n-1);i+=1)
print(substr($0, i, n))}
```

```
#include <iostream>

using namespace std;



class Record {
public:
  Record * children[26];
  long count;

  Record();

  Record * child(int i);

  ~Record();

  void output_counts(void);

};

Record::Record() {
  for (int i=0; i<26; i++)
    children[i] = (Record *) 0;

  count=0;
}

Record::~Record() {
  for (int i=0; i<26; i++) {
    if (children[i]!=(Record *) 0) delete children[i];
  }
}
```

Finally, a custom-written C++ program is used to count the frequency of each subsequence. The way this counting is done deserves some explanation. Because there are 10^{26} possible 20-length subsequences, it is clearly impossible to have a computer program which would count subsequence occurrences using a large array of integers corresponding to every possible subsequence, as this would take far more memory than all computers ever built. Instead, this program uses a hierarchical storage system to only store and count the subsequences which actually occur.

The subsequence counts are stored in a tree of Record classes. The first "letter" of a sequence is used to chose which branch of the Record tree to start with, and each subsequent letter determines the branch in the next layer of the tree. New branches are created dynamically only when needed. Thus it is possible to count subsequences that occur without using memory to count the astronomical number of subsequences that do not occur.

```
Record * Record::child(int i) {
  if (children[i]==(Record *) 0) children[i] = new Record;
  return children[i];
}

void Record::output_counts(void) {
  if (count!=0)
    cout << count << '\n';
  else {
    for (int i=0; i<26; i++) {
      if (children[i]!=(Record *) 0) children[i]->output_counts();
    }
  }
}


int main(void) {

  Record baserecord;
  char storagestring[100];
  int j;
  Record * currentrecord;

  while ( (cin >> storagestring) ) {
    j=0;
    currentrecord = &baserecord;
    while (storagestring[j]!='\0') {
      currentrecord = currentrecord->child( storagestring[j]-'A' );
      j++;
    }

    currentrecord->count++;

  }


  baserecord.output_counts();

}
```