

Techniques for Using Wild-Card Characters for Automated Peptide Extraction in Phylogeny

Joshua Lothian <lothian@cs.utk.edu>

Table of Contents

Introduction	1
Considerations	2
Initial approach	2
Final Serial Implementation	3
Speed	3
Parallel Implementation	3
Other information	4

Introduction

aacode3 is a program that converts a text file in FASTA format to matrices containing word count frequencies that can be utilised by other applications. The original version of this program was able to do this only for exact matches of words. This project extended the original idea to include "wildcards" in the words. For example, if the protein sequence being examined is ABCDEFGH with a wordsize of 4, the following words would be generated:

```
ABCD
BCDE
DEFG
EFGH
```

The modified version of **aacode3** can generate wildcards within these sequences. Matching on these can be thought of as accounting for random mutations in these proteins. Perhaps one out of the sequence was changed. Throughout the rest of this paper a X/Y gram will refer to words generated with a window size of Y with X characters that are not wildcards. Taking the previous sequence, we generate a 4/7 gram:

```
xxxDEFG
xxCxEFG
xxCDxFG
xxCDExG
xxCDEFx
xAxDEFx
xABxEFx
xABCxFx
xABCDxx
```

AxxxEFG
 AxxDxFG
 AxxDExG
 AxxDEFx
 AxCxEFx
 AxCDxFx
 AxCDExx
 etc...

So with a 4/7 gram, we have the same effective word size, however there are (6 choose 3) times as many more possibilities. The goal is to have these extra possibilities provide a kind of fuzzy-matching based on the wildcards we encode.

Considerations

It can also be noted that some sequences should not be generated. For instance, consider the sequence ABCD generated as a 2/3 gram.

xBC *
 AxB
 ABx
 xCD
 BxD
 BCx *

Notice the two starred entries are referring to the same sequence of known amino acids. For this reason, the decision was made that there will not be a wildcard generated as the first character of a word. If there are multiple consecutive ones, they will be moved to the end of the sequence for counting purposes. For example, xxDEF becomes DEFxx.

Initial approach

The algorithm used by the original **aacode3** worked like this:

```

for every protein
  for every word in current protein
    hash the value of the word add it to the global weight

for every protein
  for every word in current protein
    hash the value of the word
    compute the local weight
    write out value to file
  
```

The initial treatment of wildcards was just to extend the alphabet by one more character and using the original hasing algorithm. All possible combinations of wildcards within a given window were generated and hashed. However, this was far too memory intensive to be useful. A 4/7 gram required $21^7 * 8 \text{bytes} \approx 13.74 \text{ gigabytes}$ of memory to store the matrix. Obviously this was not practical.

The next approach changed the actual implementation of the hashing algorithm, illustrated by the following pseudocode.

```

for every protein
  for every word in current protein
    for every possible combination of wildcards
      hash the value of the word
      add it to the global weight

for every protein
  for every word in current protein
    for every possible combination of wildcards
      hash the value of the word
      compute the local weight
  write out value to file

```

This was implemented using the binary values of the wildcards positions as column indices for the array. For example, if there are wildcards in the 3, 5 and 6 positions and the other positions hashing to 43, the array index becomes $43 \cdot (2^3 + 2^5 + 2^6) = 43 \cdot (8 + 32 + 64) = 4472$. In this way it was ensured that no collisions would occur when computing the hash with the wildcards.

This method, however, still required large amounts of memory. Assuming a window size $w=7$, wildcards $d=3$, and letters $L=20$ in our alphabet the resulting in-memory matrices were $L^4 \cdot 2^d \cdot 8\text{bytes} = 20^4 \cdot 2^7 = 164\text{megabytes}$.

Final Serial Implementation

It must be noted that while the previous approach uses 2^d columns for storing the matrix, there are in fact only $(w \text{ choose } d)$ possible values that could be reached. By creating a mapping between the actual values taken on and the possible values, it was possible to reduce the storage required by the matrix down to for a 4/7 gram $20^4 \cdot (6 \text{ choose } 3) \cdot 8\text{bytes} \approx 25\text{megabytes}$. This map is precomputed before any other processing.

Speed

Once storage concerns were addressed, the issue of speed came up. To process a 5000 protein data set took around 18 hours to process as a 4/7 gram with the modified serial code. In the original **aacode3**, this data set took around 5 minutes to process. This slowdown can be attributed to two factors: the added work done to process the wildcards, and the reduction of large areas of repeated code to functions. Since these pieces of code ran on every word, the function calls added a non-trivial amount of computational overhead. However, the code is much more maintainable in this state.

Parallel Implementation

Since this problem consists of many distinct parts that are not interrelated, using a technology such as MPI was an obvious solution for speeding up processing. The conversion to MPI was fairly straightforward. During the first processing pass that calculates global weights, each processor is given a range of the proteins to be processed. Once all processors are done with their global weights, `MPI_Reduce()` is called to gather all of the data on to the master host.

- -4pc - -4pc

For the second pass, each processor works on one protein at a time and then transmits the results to the master. This must be done in order, because the order the proteins are listed in the `.pinfo` file must be preserved when writing the matrix. This does introduce some unnecessary idleness on the machine, however there were issues when the processors were allowed to send as much data as they could.

Other information

- [OpenOffice Presentation \[presentation.sxi\]](#)
- [aacode3 MAN page \[aacode3.1.html\]](#)
- [References \[references.html\]](#)
- [Postscript version of this document \[aacode.ps\]](#)
- [PDF version of this document \[index.pdf\]](#)
- [PDF version of the manpage \[aacode3.1.pdf\]](#)