Fast HMM Mapping

Fabian Kaelin
Computer Science
McGill University, Montreal
fabian.kaelin@mail.mcgill.ca

Dec 2008

1 Abstract

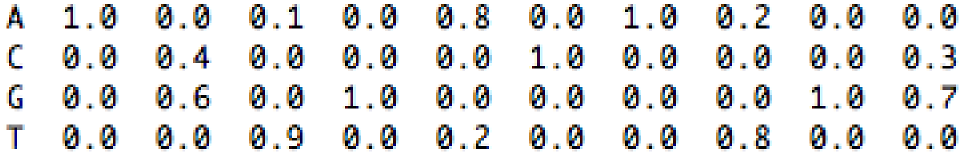
In this project I experiment with a new way to match a Hidden Markov Model to a long sequence of observations. The proposed approach is based on the same mechanism BLAST uses to align two sequences. The idea is to index the long sequence of observations and only search for good matches of the HMM in interesting regions of the observations. In this report I present the approach used and the difficulties I encountered. Further I will discuss the results of some experiments.

2 Fast HMM Mapping

2.1 Definitions

Background: The background is defined as the emission probability distribution of nucleotides in region that are not interesting at this point. As an example, we could consider a background where all the nucleotides are equiprobable. Thus, the emission probabilities in this case would be 25% each. A more realistic background would be 30%A, 20%C, 20%G, 30%T.

Position Weight Matrix PWM: The position weight matrix of a given motif contains the emission probabilities of each position of the motif. In figure 1 you can see an example of a PWM of length 10. The length of a PWM is usually between 8 and 20 nucleotides.



A	1.0	0.0	0.1	0.0	0.8	0.0	1.0	0.2	0.0	0.0
C	0.0	0.4	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.3
G	0.0	0.6	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.7
T	0.0	0.0	0.9	0.0	0.2	0.0	0.0	0.8	0.0	0.0

Figure 1: Position Weight Matrix

Database: We want to match a hidden markov model against a sequence of observations. In our case this sequence is very long. That is why we consider the observations as a database. A database consists of a long sequence of nucleotides. For this project the maximum length was one million nucleotides, but in reality this number would be much higher.

Index: To create an index of a database, we first have to choose the word length w of one index word. Then we create all possible sequences of length w. For $w = 8$ this would mean we will have $4^8 = 65536$ words. For each sequence of length w in our database (there are $len(database) - w + 1$ sequences) we assign the starting position of that sequence to its corresponding word in the index.

HMM Model: The hidden markov model used in this project is a very simple one. It is shown in figure 2. The model is very likely to stay in the background state, so p is usually a very small number (e.g. 0.001). The emission probabilities of the background state are given by the background emission probabilities defined above. It would computationally be easier to work with an equiprobable background. But in this project I assumed a more realistic distribution of 30%A, 20%C, 20%G, 30%T.

As soon as the background state is left the model iterates over the states M1 through Mn, as all the transition probabilities are equal to 1. The emission probabilities of the states M1 to Mn are defined in a position weight matrix. As soon as the model iterated over the motif states, it will jump back to the background state (or waiting state).

2.2 Approach

When searching for Transcription factor binding sites (TFBS), the motif we are looking for is usually represented by a consensus sequence or a position weight matrix. For this project we construct a simple HMM out of a position weight matrix for a given motif. The goal is to find good matches for the motif in the database.

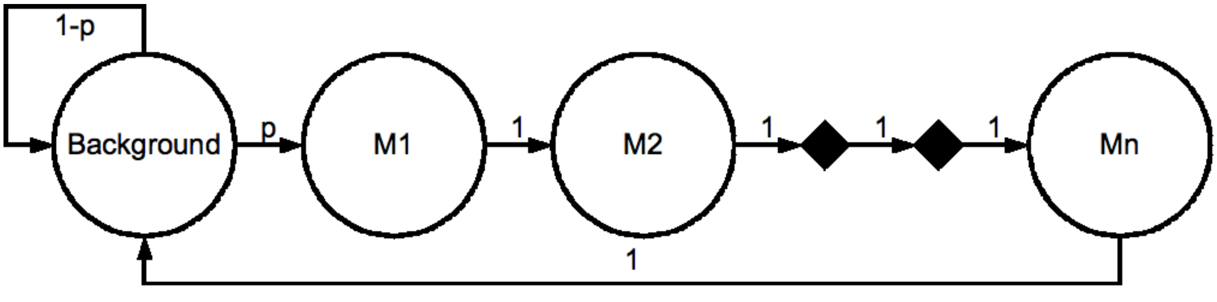


Figure 2: Simple HMM model

When considering the simple HMM proposed, finding a good match is equivalent to detecting at what points in the database the model iterated over states M1 to Mn. This could be done by implementing the Forward/Backward algorithm or the Viterbi algorithm. In this project we try to find a faster way to identify those matches based on the index of the database.

The proposed Fast HMM approach can be described by the following steps:

1. **Score index word:** Calculate a score for each index word.

2. **Discard index words:** We discard the index words that have a score that is smaller than a certain threshold.

3. **Identify starting positions:** Using the index, identify all starting positions in the database of the index words that haven't been discarded.

4. **Score database words:** Using the score calculated in the first step, calculate a new score based on the database extension of the sequence at each starting position.

5. **Report matches:** Discard all starting positions that have a score smaller than a certain threshold and report the remaining starting positions as matches of the HMM.

2.2.1 Score index word

For each word in the index of the database we calculate two probabilities.

1. The probability of observing the given word when assuming that the model stayed in the background state.

$$P(word|background) = (1-p)^{w+1} * P_A^{|A|} * P_C^{|C|} * P_G^{|G|} * P_T^{|T|} \tag{1}$$

2. The probability of observing the given index word when assuming that the model passes through the states M1 through Mn.

$$P(word|pwm) = p * \prod_{n=1}^{w} 1 * P_{n \in \{A,C,G,T\}} \tag{2}$$

The score of the index word is defined as the ratio between the two probabilities.

$$scoreWord = \frac{P(word|pwm)}{P(word|background)} \tag{3}$$

2.2.2 Discard index words

Now we have to choose which words we want to keep investigating. Usually the length of the index word w is shorter than the length of the position weight matrix. The previously calculated score serves as a heuristic to determine wether the HMM is more likely to 'choose' to stay in the background state or not. So the bigger the score, the more likely it is that the HMM would iterate over the PWM. But even if the ratio of the probabilities is bigger than 1 doesn't mean that the HMM would actually choose to traverse the motif states. This would be the case if we had a good match for the first w positions of the motif but a bad match for the extension.

2.2.3 Identify starting positions

Now that we have selected index words that are interesting for us as potential starting points for matches we can go on and look at the actual database. We compile a list of all starting positions in the database that are associated with the index words that were not discarded before.

2.2.4 Score database words

As mentioned before, the length of the index word w is usually shorter than the length of the position weight matrix. So to calculate an updated score we extend the index word we find at each starting point in our list and update the calculated probabilities [2] and [1]

$$P(dbword|BG) = P(word|BG) * (1-p)^{|pwm|-(w+1)} * P_A^{|A|} * P_C^{|C|} * P_G^{|G|} * P_T^{|T|} \quad (4)$$

$$P(dbword|pwm) = P(word|pwm) * \prod_{n=w+1}^{|pwm|} 1 * P_{n\in\{A,C,G,T\}} \quad (5)$$

$$scoreDBWord = \frac{P(dbword|pwm)}{P(dbword|background)} \quad (6)$$

2.2.5 Report matches

Again we compare the score to a threshold that can be set as parameter. In this case the final probabilities we calculated correspond to the actual probabilities calculated by the other algorithms, so a score of 1 means that the model would have actually chosen the motif in a standard algorithm like the for example the forward-backward algorithm. But this parameter gives us the flexibility to make our matches more or less specific by decreasing or increasing the threshold for the database word. Of course it is still possible that we wrongly discarded an index word in the first step in which case we will miss at least one proper match.

3 Discussion

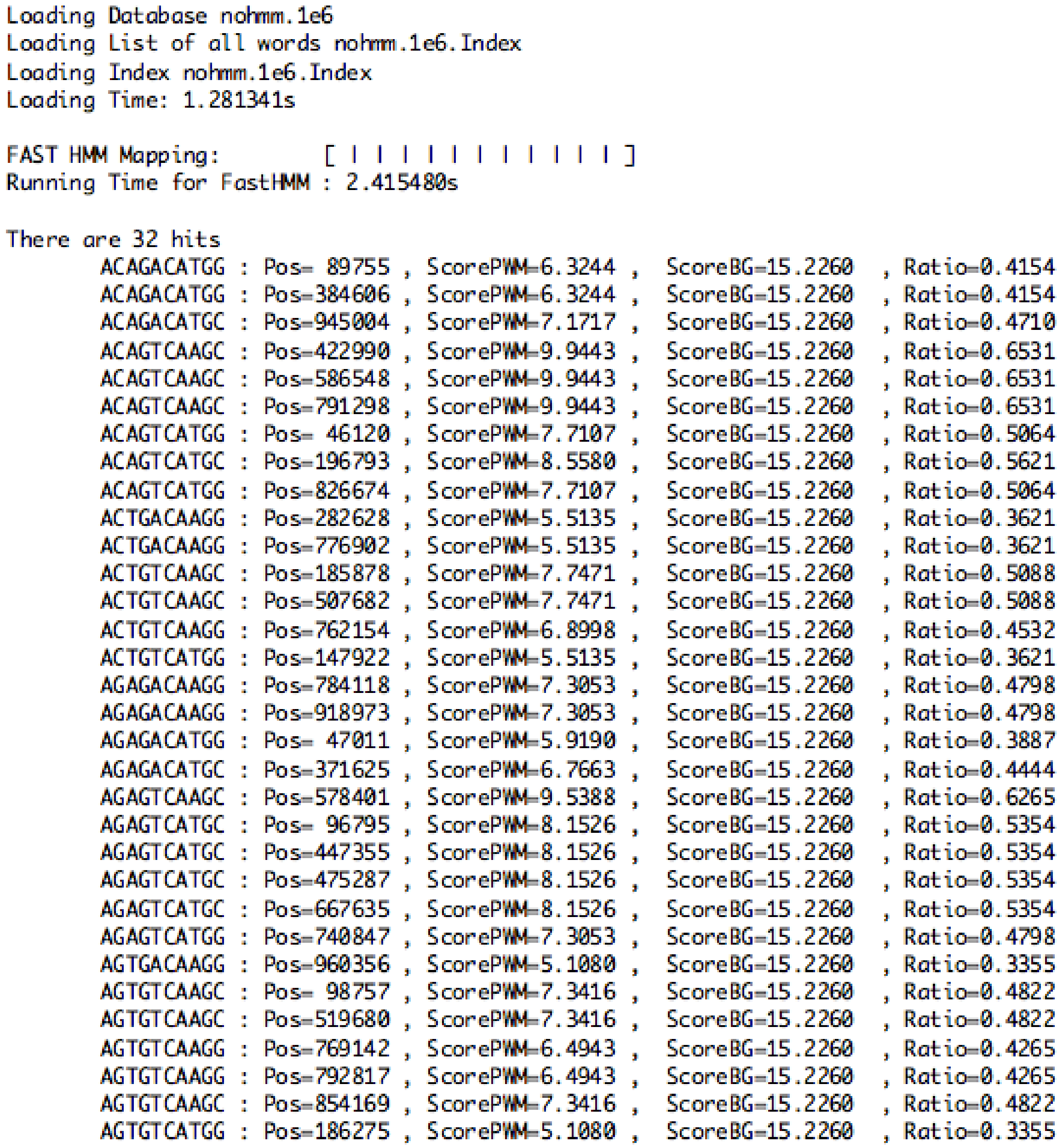
At this point I would like to present some of the preliminary results. In the subsuction 'Implementation' you can find a short introduction that shows how to setup and use the source code, in case you want to test it yourself.

3.1 Sample Output

To show what the inputs and outputs of the program are, consider the following example. The settings used for the simulation are:

- A database containing one million nucleotides that has been generated randomly with the probabilities 30%A, 20%C, 20%G, 30%T.
- The index word length is eight.
- The motif HMM is described by the position weight matrix in figure 1 and by the background used to generate the database : 30%A, 20%C, 20%G, 30%T.
- The transition probability to enter the position weight matrix is 0.001. Therefore the probability to stay in the background state is 0.999
- The threshold for the index word [3] is set to 1.
- The threshold for the database word extension [6] is set to 1.

In figure 3 you can see the output of that experiment. The first time a new database name is specified, the database and the index are created and saved to the disk. So if you want to perform another experiment using the same database, it will be loaded automatically. Of course you can also use a different index word length with the same database. In our example we can see that the database and the index have previously been created, and the loading time for both was 1.28 seconds. The next step is the Fast HMM algorithm which took 2.41 seconds. Below we can see the proposed matches with their positions and scores. The algorithm lists 32 matches.



```
Loading Database nohmm.1e6
Loading List of all words nohmm.1e6.Index
Loading Index nohmm.1e6.Index
Loading Time: 1.281341s

FAST HMM Mapping:        [ | | | | | | | | | | | | ]
Running Time for FastHMM : 2.415480s

There are 32 hits
        ACAGACATGG : Pos= 89755 , ScorePWM=6.3244 ,  ScoreBG=15.2260  , Ratio=0.4154
        ACAGACATGG : Pos=384606 , ScorePWM=6.3244 ,  ScoreBG=15.2260  , Ratio=0.4154
        ACAGACATGC : Pos=945004 , ScorePWM=7.1717 ,  ScoreBG=15.2260  , Ratio=0.4710
        ACAGTCAAGC : Pos=422990 , ScorePWM=9.9443 ,  ScoreBG=15.2260  , Ratio=0.6531
        ACAGTCAAGC : Pos=586548 , ScorePWM=9.9443 ,  ScoreBG=15.2260  , Ratio=0.6531
        ACAGTCAAGC : Pos=791298 , ScorePWM=9.9443 ,  ScoreBG=15.2260  , Ratio=0.6531
        ACAGTCATGG : Pos= 46120 , ScorePWM=7.7107 ,  ScoreBG=15.2260  , Ratio=0.5064
        ACAGTCATGC : Pos=196793 , ScorePWM=8.5580 ,  ScoreBG=15.2260  , Ratio=0.5621
        ACAGTCATGG : Pos=826674 , ScorePWM=7.7107 ,  ScoreBG=15.2260  , Ratio=0.5064
        ACTGACAAGG : Pos=282628 , ScorePWM=5.5135 ,  ScoreBG=15.2260  , Ratio=0.3621
        ACTGACAAGG : Pos=776902 , ScorePWM=5.5135 ,  ScoreBG=15.2260  , Ratio=0.3621
        ACTGTCAAGC : Pos=185878 , ScorePWM=7.7471 ,  ScoreBG=15.2260  , Ratio=0.5088
        ACTGTCAAGC : Pos=507682 , ScorePWM=7.7471 ,  ScoreBG=15.2260  , Ratio=0.5088
        ACTGTCAAGG : Pos=762154 , ScorePWM=6.8998 ,  ScoreBG=15.2260  , Ratio=0.4532
        ACTGTCATGG : Pos=147922 , ScorePWM=5.5135 ,  ScoreBG=15.2260  , Ratio=0.3621
        AGAGACAAGG : Pos=784118 , ScorePWM=7.3053 ,  ScoreBG=15.2260  , Ratio=0.4798
        AGAGACAAGG : Pos=918973 , ScorePWM=7.3053 ,  ScoreBG=15.2260  , Ratio=0.4798
        AGAGACATGG : Pos= 47011 , ScorePWM=5.9190 ,  ScoreBG=15.2260  , Ratio=0.3887
        AGAGACATGC : Pos=371625 , ScorePWM=6.7663 ,  ScoreBG=15.2260  , Ratio=0.4444
        AGAGTCAAGC : Pos=578401 , ScorePWM=9.5388 ,  ScoreBG=15.2260  , Ratio=0.6265
        AGAGTCATGC : Pos= 96795 , ScorePWM=8.1526 ,  ScoreBG=15.2260  , Ratio=0.5354
        AGAGTCATGC : Pos=447355 , ScorePWM=8.1526 ,  ScoreBG=15.2260  , Ratio=0.5354
        AGAGTCATGC : Pos=475287 , ScorePWM=8.1526 ,  ScoreBG=15.2260  , Ratio=0.5354
        AGAGTCATGC : Pos=667635 , ScorePWM=8.1526 ,  ScoreBG=15.2260  , Ratio=0.5354
        AGAGTCATGG : Pos=740847 , ScorePWM=7.3053 ,  ScoreBG=15.2260  , Ratio=0.4798
        AGTGACAAGG : Pos=960356 , ScorePWM=5.1080 ,  ScoreBG=15.2260  , Ratio=0.3355
        AGTGTCAAGC : Pos= 98757 , ScorePWM=7.3416 ,  ScoreBG=15.2260  , Ratio=0.4822
        AGTGTCAAGC : Pos=519680 , ScorePWM=7.3416 ,  ScoreBG=15.2260  , Ratio=0.4822
        AGTGTCAAGG : Pos=769142 , ScorePWM=6.4943 ,  ScoreBG=15.2260  , Ratio=0.4265
        AGTGTCAAGG : Pos=792817 , ScorePWM=6.4943 ,  ScoreBG=15.2260  , Ratio=0.4265
        AGTGTCAAGC : Pos=854169 , ScorePWM=7.3416 ,  ScoreBG=15.2260  , Ratio=0.4822
        AGTGTCATGG : Pos=186275 , ScorePWM=5.1080 ,  ScoreBG=15.2260  , Ratio=0.3355
```

Figure 3: Sample Output

To see wether this is a reasonable number we can calculate an approximate number of matches would we would expect: $E[number of matches] \doteqdot (1e6-10+1)*32*0.25^{10} =$ 30.5 matches. Taking into account that this is only one single experiment, the number 32 seems to be in an acceptable range.

3.2 Time Measurements

It would be interesting to compare the running time of the Fast HMM to the running time of for example the Forward/Backward algorithm. But the implementation turned out be more complicated than anticipated and it wasn't in the scope of this project neither. The Forward/Backward algorithm has been implemented though, but it is not reasonable to compare their performance just yet. In the subsection 'Proof of Concept' you can find a comparison of the matches that were found. The time measurements in this section are based on the Fast HMM algorithm only.

3.2.1 Time per Database

In figure 4 we can see that the longer the database, the longer the running time. The graph seems to have an exponential form which suggests that the running time increases exponentially. This is not very surprising but bad news nevertheless.

3.2.2 Time per Index word length w

In figure 4 we can see a comparison of the running time for different index word lengths. The database used consists of 1e6 nucleotides and the length of the position weight matrix is 20. The graph was somewhat counterintuitive at first, because I expected the running time to decrease with the length of an index word. To understand why the opposite is true, there are two interesting things to observe. In left part of the graph, for index word lengths smaller than seven, the running time is close to zero. Why this is the case is explained in the section 'Matches per Index word length' (figure 6). For index word lengths greater than seven, we can

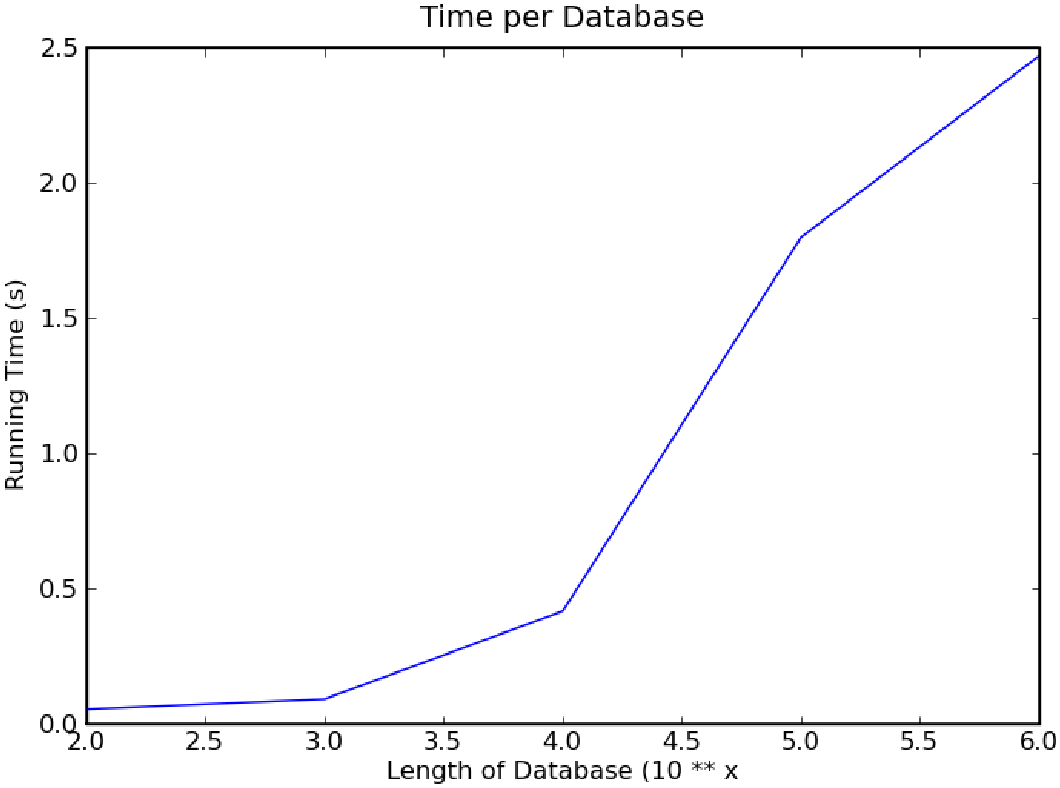


Figure 4: Time per Database

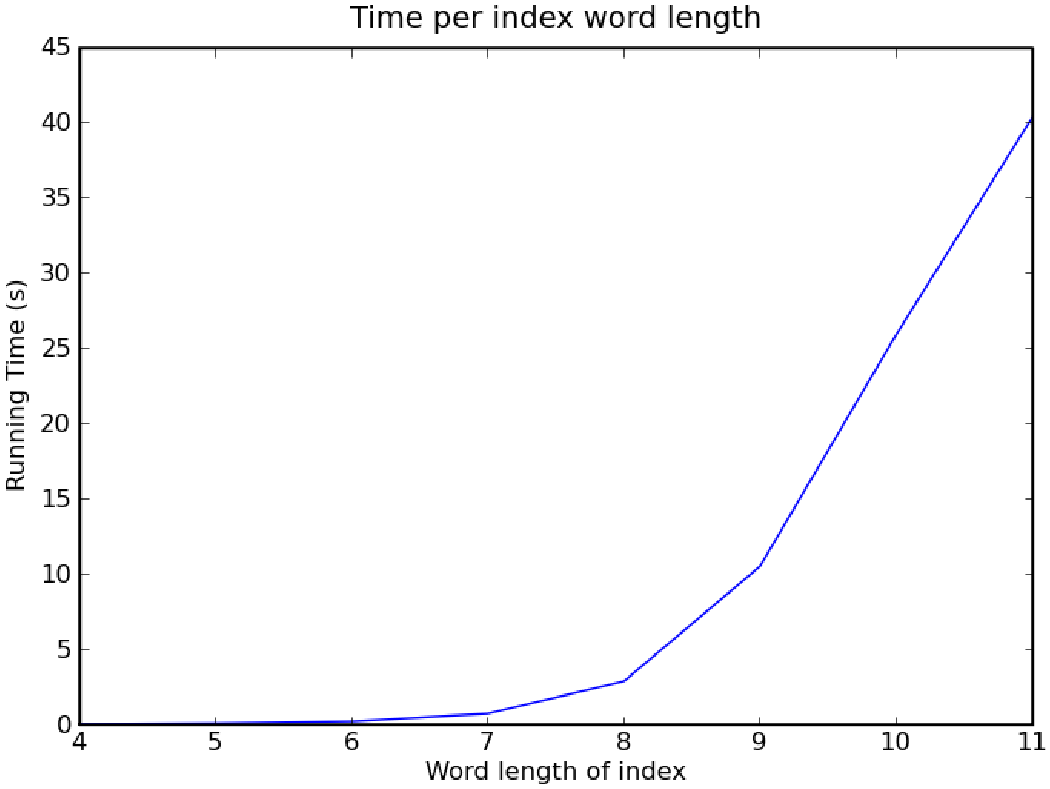


Figure 5: Time per Index word length

see that the running time increases exponentially. This makes sense if you consider that the number of index words increases exponentially as well, it is 4^w, and that the algorithm first calculates a score for each index word. The effect of having to calculate fewer extensions because there are fewer starting positions per index is far outweighed by the increased number of index words that the algorithm has to score.

3.3 Number of Matches

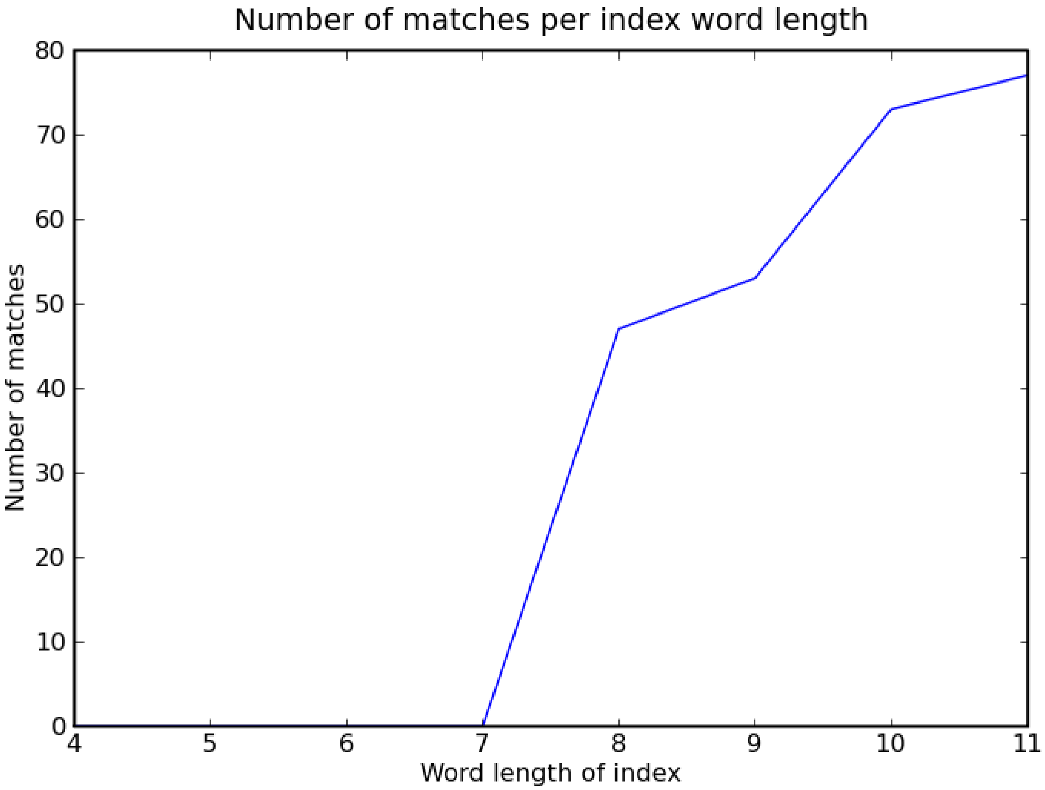


Figure 6: Matches per Index word length

3.3.1 Number of Matches per Index word length w

Now let us look at how the number of matches depends on the parameters. The database used consists of 1e6 nucleotides and the length of the position weight matrix is 20. As you can see in figure 6, there are no matches for index word lengths shorter than seven. This can be explained by the fact that the probability to enter the motif is fairly small (e.g. 0.001), so for the HMM to leave the background state the probability of the observations must be much bigger for the motif states than for the background state. If the index word length for the first score [3] is very short, it becomes very hard for the motif to 'convince' the HMM to leave the background state.

In the same graph we can also see that the number of matches always increases with the index word length. The same logic as above can be applied, but another effect comes into play. The compromise we accept is that we take the risk of discarding positive matches. This happens if we have a somewhat low probability for the first w positions of the database word, but for the positions $[w, (w + |pwm|)]$ we have a very good match. In that case it is likely that the index word is discarded but the database word would have been reported. Logically the probability of this happening decreases when we choose a longer index word length w, and as consequence the number of matches increases with w.

3.3.2 Number of Matches and Thresholds

As mentioned above, the approach of the Fast HMM algorithm is a compromise between speed and accuracy. Now we will have a look at how the two thresholds defined in the 'Approach' section affect the matches reported by the Fast HMM. First the algorithm calculates the score of the index word [3], and if that score is above a certain threshold it extends the score for all the starting positions associated with the index word [6]. The value of the two scores are the ratio between the probability of the observations given the motif divided by the probability of the observations given the background. A natural choice for this threshold would be 1. A value greater than 1 indicates that the likelihood of the motif is bigger than the likelihood of the background. Both thresholds, but mainly the threshold for the index word score [3], allow us to 'play' with this compromise by soften or harden the constraints on the likelihood of the motif when we decide wether to further investigate a certain index word. By choosing a softer threshold we can also lower the sensitivity of our motif. The results presented in all of the graphs below are based on an experiment run with a database of length 1e6 that was generated with the HMM that we are looking for. The transition probability to enter the motif is 0.001. So we expect around $1e6 * 0.001 = 1000$ matches.

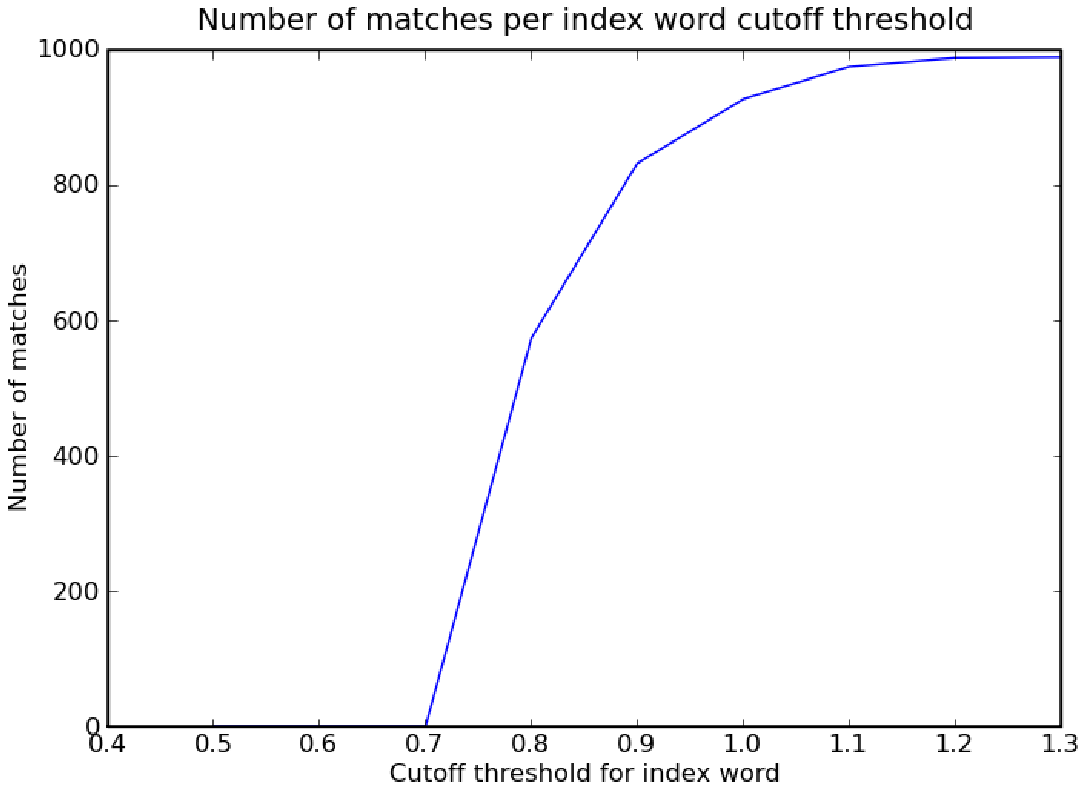


Figure 7: Vary index word threshold

In figure 7 we can see the effect of varying the threshold for the index word score [3] on the number of matches reported. Keep in mind that the threshold for the extension is fixed at 1. In this plot we can clearly see what has been discussed before. The number of matches keeps increasing even when we choose a threshold greater than 1. We approach the expected number of a 1000 matches at a threshold of 1.2. It is worth pointing out that the line is leveling out partly because we have zero probabilities in our position weight matrix and if we added pseudo counts to avoid zero probabilities all the index words would have a finite score.

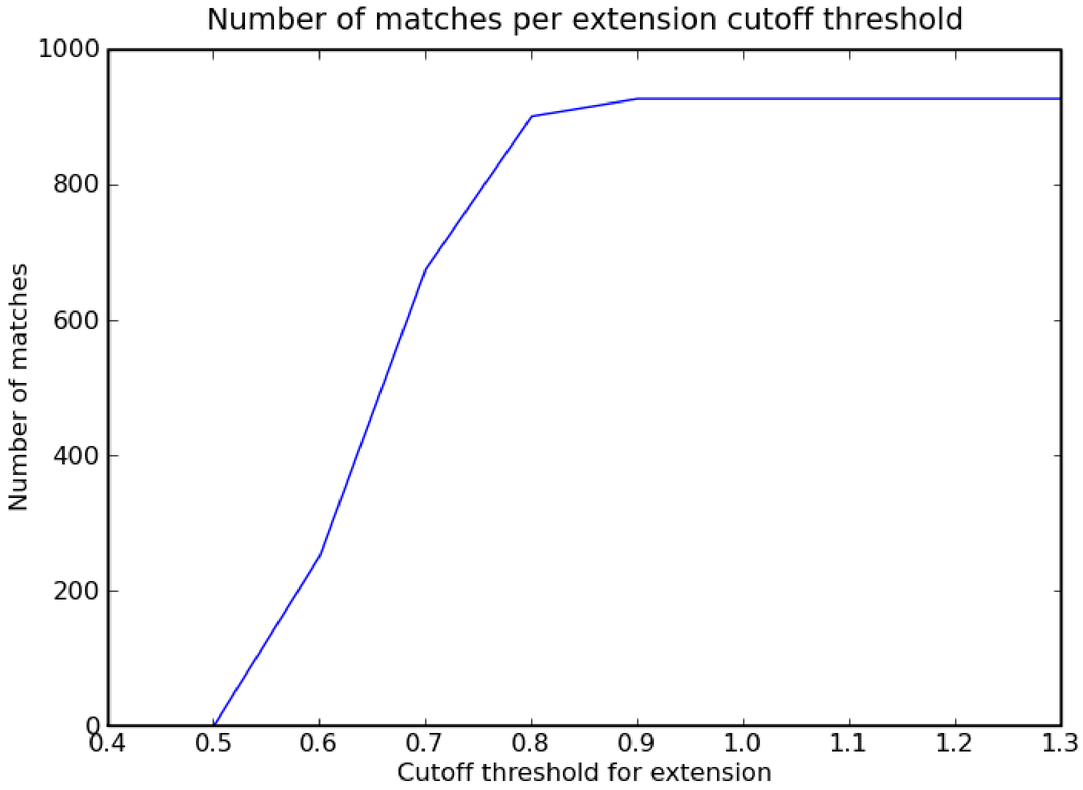


Figure 8: Vary the database extension threshold

Figure 8 is the same plot but instead of varying the index word threshold we change the extension threshold [6]. The graph reconfirms what we found before.

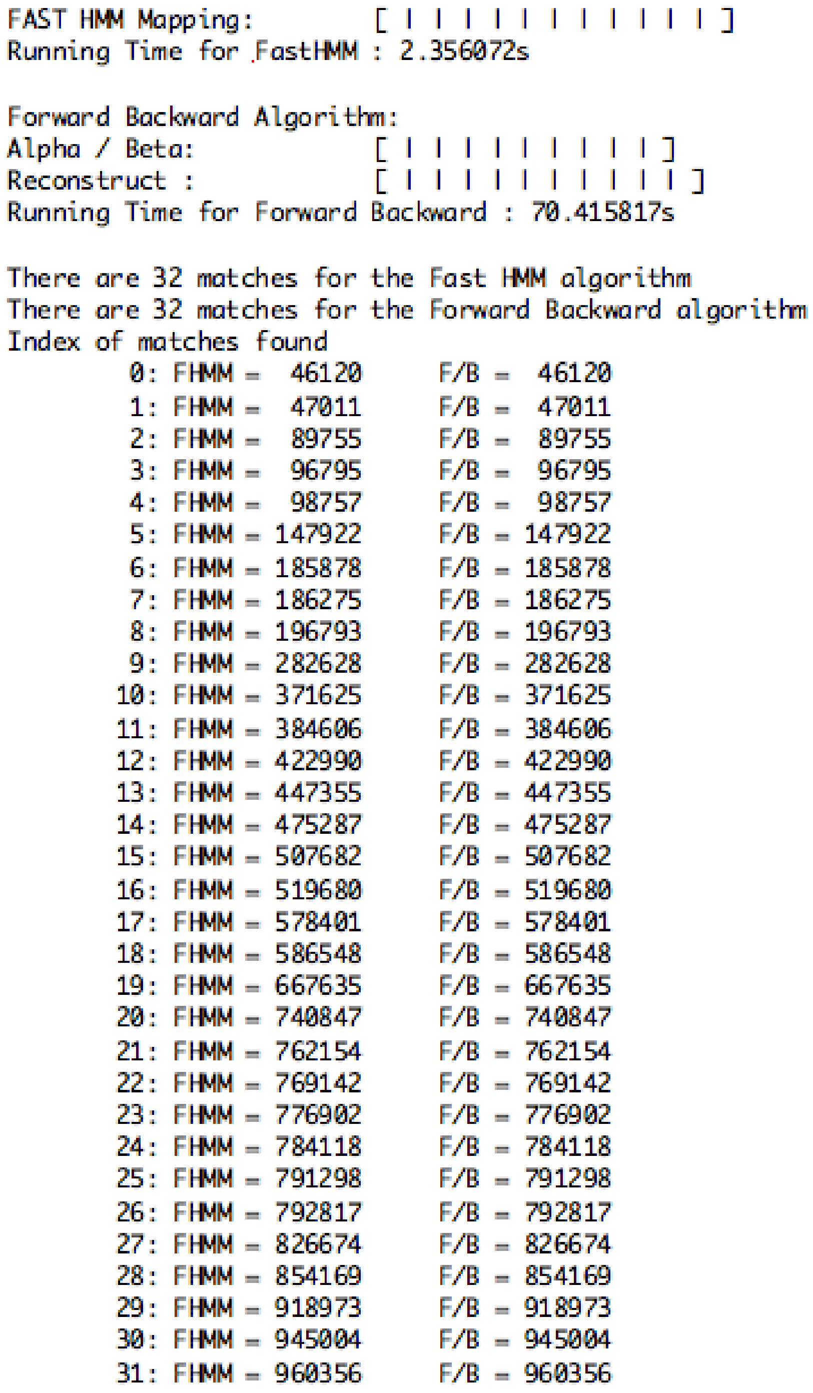
Remember that the index word threshold is fixed at 1. So all the positive matches that the Fast HMM algorithm is wrongly discarding after the first step can't be reported whatever the threshold for the database extension. As a consequence the line levels out at the same number of matches (ca. 920) that we found at an index word threshold of 1 in figure 7.

4 Proof of Concept

To determine how accurate the matches of the proposed Fast HMM algorithm are, I implemented the standard Forward/Backward algorithm. In figure 9 you can see the output of an experiment comparing the matches of the algorithms. The settings for the experiments are identical to the one for the 'Sample Output':

- A database containing one million nucleotides that has been generated randomly with the probabilities 30%A, 20%C, 20%G, 30%T.
- The index word length is eight.
- The motif HMM is described by the position weight matrix in figure 1 and by the background used to generate the database : 30%A, 20%C, 20%G, 30%T.
- The transition probability to enter the position weight matrix is 0.001. Therefore the probability to stay in the background state is 0.999
- The threshold for the index word [3] is set to 1.
- The threshold for the database word extension [6] is set to 1.

As you can see, the matches are completely identical. This is very good, but at the same time there is not much room for error because the index word length of 8 is only a little shorter than the position weight matrix of length 10. But this result is satisfying nevertheless because it confirms that the Fast HMM algorithm seems to be a valid approach.



```
FAST HMM Mapping:         [ | | | | | | | | | | | ]
Running Time for FastHMM : 2.356072s

Forward Backward Algorithm:
Alpha / Beta:             [ | | | | | | | | | ]
Reconstruct :             [ | | | | | | | | | | ]
Running Time for Forward Backward : 70.415817s

There are 32 matches for the Fast HMM algorithm
There are 32 matches for the Forward Backward algorithm
Index of matches found
         0: FHMM =  46120      F/B =  46120
         1: FHMM =  47011      F/B =  47011
         2: FHMM =  89755      F/B =  89755
         3: FHMM =  96795      F/B =  96795
         4: FHMM =  98757      F/B =  98757
         5: FHMM = 147922      F/B = 147922
         6: FHMM = 185878      F/B = 185878
         7: FHMM = 186275      F/B = 186275
         8: FHMM = 196793      F/B = 196793
         9: FHMM = 282628      F/B = 282628
        10: FHMM = 371625      F/B = 371625
        11: FHMM = 384606      F/B = 384606
        12: FHMM = 422990      F/B = 422990
        13: FHMM = 447355      F/B = 447355
        14: FHMM = 475287      F/B = 475287
        15: FHMM = 507682      F/B = 507682
        16: FHMM = 519680      F/B = 519680
        17: FHMM = 578401      F/B = 578401
        18: FHMM = 586548      F/B = 586548
        19: FHMM = 667635      F/B = 667635
        20: FHMM = 740847      F/B = 740847
        21: FHMM = 762154      F/B = 762154
        22: FHMM = 769142      F/B = 769142
        23: FHMM = 776902      F/B = 776902
        24: FHMM = 784118      F/B = 784118
        25: FHMM = 791298      F/B = 791298
        26: FHMM = 792817      F/B = 792817
        27: FHMM = 826674      F/B = 826674
        28: FHMM = 854169      F/B = 854169
        29: FHMM = 918973      F/B = 918973
        30: FHMM = 945004      F/B = 945004
        31: FHMM = 960356      F/B = 960356
```

Figure 9: Output of comparison between Fast HMM and Forward/Backward

Originally my plan was to compare the performances. As you can see in figure 9, the Fast HMM took only 2.35 seconds compared to the 70 seconds for the Forward/Backward. But I have the feeling that my implementation of the Forward/Backward algorithm is not the fastest and so I content myself with the fact that results seem to be similar.

5 Next Steps

The proposed Fast HMM algorithm seems to be a promising approach and it would be interesting to investigate further. Here are some ideas for what could be done next:

- Implement an efficient Forward/Backward algorithm and compare the performance between the Fast HMM algorithm and the standard Forward/Backward algorithm.
- Do some data processing on the starting positions of the index words that have not been discarded after the first step [3]. For example we could cluster them into groups to identify regions of interest in the database (e.g. promoter regions).
- Refine to proposed simple HMM. For example one could conceive an HMM that also detects promotor regions and is therefore able to 'point' to interesting regions of the database.
- Test the Fast HMM algorithm with longer databases. The maximum of one million nucleotides for this project is not very realistic.
- It is interesting to note that it is possible that Fast HMM algorithm reports matches that overlap each other, whereas for the Forward/Backward algorithm this is impossible. What are the consequences of this fact?

- Another way to minimize the number of wrongly discarded matches after the first step [3] would be to decrease the transition probability from background to motif. The idea would be to adapt this probability to the length of index word.

6 Implementation

All the code is written in python. You will need the pylab, numpy, scipy and the matplotlib packages. The folder structure is :

db contains the databases that are created. The filename corresponds to the database name, so if there is a file that corresponds with the database name at runtime, the file will be loaded as database

graphs contains all the plots

hmm contains files that describe the motif and files that define a background

index contains the index files that are created. The filename corresponds to the index name, so if there is a file that corresponds with the index name at runtime, the file will be loaded as index of the database

scripts contains a number of configuration files for different experiments. Most of them create some sort of plot from aggregated data

The files in the root folder:

db.py describes a database and the index

fhmmm.py Fast HMM Mapping: Main run file used to setup an experiment

hmm.py contains the different algorithms : Fast HMM and Forward/Backward

utils.py as the name says

The output works best in the eclipse environment, so I would suggest to install pydef for eclipse. To run an experiment yourself, you should edit the file 'fhmmm.py' to configure the parameters. Next just execute 'python fhmmm.py' to start the experiment.

7 References

[**1**] A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, Lawrence R. Rabiner, 1989

[**2**] Hidden Markov models: inference and parameter fitting, Ted Perkins, 2008

[**3**] Forward-backward algorithm, http://en.wikipedia.org/wiki/Forward-backward_algorithm

[**4**] Profile HMMs, Class notes