

# Motif finding

GCB 535 / CIS 535  
M. T. Lee, 10 Oct 2004

Our goal is to identify significant patterns of letters (nucleotides, amino acids) contained within long sequences. The pattern is called a motif. We'll stick to nucleotides for the rest of this discussion.

Part I is an extensive discussion about positional weight matrices, which are the foundation for the motif finding algorithms described in Part II. Part II goes into each of three algorithms – CONSENSUS, Gibbs sampling, and EM – in a fair amount of detail. The details aren't really important as long as you understand the big picture.

## I. (More than you wanted to know about) Positional weight matrices

### A. What is a PWM?

You can think of a PWM as a way to represent the motif you are interested in. It specifies the probability that you will see a given base at each index position of the motif. For a motif of length  $m$ , your PWM will be a 4 nucleotide  $\times$   $m$  positions matrix, like the following, for a 5-nucleotide motif:

	1	2	3	4	5
A	0.25	0.30	0.90	0.10	0.35
C	0.25	0.40	0.050	0.85	0.10
G	0.25	0.10	0.025	0.025	0.35
T	0.25	0.20	0.025	0.025	0.05

Table 1: A positional weight matrix for a 5-nucleotide motif

The PWM says that in the first position all bases are equally likely, the second position is most likely to be a 'C' with probability 0.4, etc. Note that each column adds up to 1 because we are working with probabilities.

### B. Empirically creating a PWM

Given a set of aligned sequences that you think are all examples of a particular motif, you can create a PWM by counting the number of times you see each base at each position, and then dividing by the total number of sequences to create a probability. For example, lets say we have

1000 sequences of length 5 that will constitute our “training set” for what our motif should look like. If 250 of them have an ‘A’ in the first position, then we say there is a  $250/1000 = 0.25$  probability of seeing an ‘A’ in the first position, and we fill in the appropriate cell in the matrix with that value. Similarly for each of the 19 other cells.

Let’s do an easy example:

Our training set contains 10 sequences: AT, AT, TA, TC, AT, CT, CT, AT, AG, GC

Our PWM will be 4x2. For the first position, we count 5 As, 2 Cs, 1 G, and 2Ts. There are 10 total sequences, so the probability of A in the first position is  $5/10 = 0.5$ , probability of C is 0.2, of G 0.1, and of T 0.2. Similarly, in the second position we count 1 A, 2 Cs, 1 G, and 6 Ts to yield respective probabilities of 0.1, 0.2, 0.1, and 0.6. The final PWM is

	1	2
A	0.5	0.1
C	0.2	0.2
G	0.1	0.1
T	0.2	0.6

Table 2: The positional weight matrix for our 2-nucleotide motif example

### C. Using a PWM

Once we have a PWM, we can use it to score new sequences to determine how likely they are to be instances of the motif. We can calculate this as the joint probability of seeing the base we see in each position of the new sequence. An important assumption we make is that the base identity at any one position does not depend on the base identity of any other position in the sequence; therefore, we can calculate the joint probability by multiplying together all of the individual probabilities at each position.

For example, take the sequence “AATCG” and the PWM in Table 1. The probability of seeing an A in the first position according to the PWM is 0.25, 0.3 of seeing an A in second position, 0.025 for the T in the third, 0.85 for the C in the fourth, and 0.35 for the G in the fifth. The product is  $0.25*0.3*0.025*0.85*0.35 = 0.00056$ , which is the probability that our new sequence AATCG is an instance of the motif; you can think of this as the score the PWM gives your sequence. This seems very improbable, though to put this in perspective, the sequence “ACACA” yields the highest possible probability, which is only 0.027.

There are three practical considerations that affect how we use PWMs in the real world. The first is that multiplying a lot of small numbers together yields underflow, which basically means you’ll end up with 0 probabilities all of the time. So traditionally, instead of probabilities we use **log probabilities**. Then to compute the joint log probability, we sum the positional log

probabilities (remember that  $\log(A*B) = \log(A) + \log(B)$ ). The following PWM is simply the PWM from Table 2 with all probabilities logged (base e):

	1	2
A	-0.69	-2.30
C	-1.61	-1.61
G	-2.30	-2.30
T	-1.61	-0.51

Table 3: The positional weight matrix for our 2-nucleotide motif example using log probabilities.

Using the non-log probability PWM, 'AT' and 'CG' would have gotten scores of 0.3 and 0.02 respectively; using log probabilities, we get  $-0.69 + -0.51 = -1.2$  and  $-1.61 + -2.30 = -3.91$ . 'AT's score is less negative than 'CG's, so it is more likely to be an instance of the motif.

The second wrinkle concerns zero-probabilities. Let's say based on your training set you end up with a PWM like the following:

	1	2	3	4	5
A	0.33	0.30	0.90	0.10	0.35
C	0.34	0.40	0.050	0.85	0.10
G	0.33	0.10	0.025	0.025	0.35
T	0	0.20	0.025	0.025	0.05

Table 4: A positional weight matrix for a 5-nucleotide motif

Notice that in position 1, the probability of seeing a T is 0. This is because none of the sequences in the training set had a T in the first position. This could be an important component of the motif, or it could be that our training set wasn't big enough and this happened by chance. Regardless, it's not usually a good idea to leave 0 probabilities around because no matter how well your sequence scores for all the other positions, one zero multiplied to your product yields a zero probability overall. Similarly, log of 0 is negative infinity, so your overall log probability would also be negative infinity.

To prevent this from happening, PWMs can be **smoothed** – that is, when calculating frequencies of bases at each position, we add on a small smoothing factor that prevents any of the counts from being zero. This factor can be the same for each base, say 0.0001, or it can depend on the background probability of that base (see below). When calculating probabilities, you must take this smoothing factor into account, otherwise your columns will not sum to 1. For example, let's say that over 100 training sequences, we see an A in the first position 33 times, a C 34 times, a G 33 times, and a T 0 times. We'll augment each of these counts by a smoothing factor of 0.1 to yield counts of 33.1 for A, 34.1 for C, 33.1 for G, and 0.1 for T. Then the probability of A will be  $33.1 / (33.1 + 34.1 + 33.1 + 0.1) = 0.32968$ ; the probability of T will be  $0.1 / 100.4 = 0.000996$ , which is really small, but not 0.

The final bit of real world practicality comes from the fact that thus far we've been assuming that each of the four nucleotides occurs with equal probability – the strength of any claim that it is significant to see a particular nucleotide at some position depends on how often you would expect to see that nucleotide by chance. The PWM in Table 2 assigns a probability of 0.5 for seeing an A in the first position. But what if 40% of the genome's bases were As? Then you shouldn't be surprised to see a greater proportion of As anywhere in your sequence, so maybe the 0.5 doesn't mean much. Conversely, if As were extremely rare in your genome, seeing a position that is A half the time would be a very significant observation.

So typically, we divide all of our probabilities by the **background probability** of seeing each nucleotide. If all nucleotides are equally likely, the scaling doesn't do anything, but if the proportions are uneven, we should see a dampening of the significance of seeing a common nucleotide at any position and an increase in the significance of seeing a rarer nucleotide.

For example, let's take two sets of background probabilities and see the effect on the PWM from Table 2.

	1	2
A	1.25	0.25
C	2	2
G	1	1
T	0.5	1.2

Table 5: The positional weight matrix using background probabilities of A=0.4, C=0.1, G=0.1, T=0.4

	1	2
A	5	1
C	0.5	0.5
G	0.25	0.25
T	2	6

Table 6: The positional weight matrix using background probabilities of A=0.1, C=0.4, G=0.4, T=0.1

Note that these are not probabilities anymore, so just think of them as scores. Again, we usually log these and sum the scores for each position. Compare the strings 'AT' and 'CG' again with the two different background probabilities. Using Table 5, 'AT' yields  $\log(1.25) + \log(1.2) = 0.41$  while 'CG' yields 1.39. Using Table 6, 'AT' yields 1.61 while 'CG' yields -1.39. This is an extreme case, but it's a good illustration of the effect that background probabilities can have.

Incidentally, what constitutes "background"? That's a difficult question to answer, but in practice, when you have a large number of aligned sequences and are looking for a short motif

contained within those sequences, your background is everything that does not fall within your motif window.

#### **D. Information content of a PWM**

An effective PWM should allow us to distinguish a motif from a random sequence. Intuitively, this means that at various positions in the PWM, there should be a clear preference for some nucleotides over others. Take the PWM in Table 1. In the first position, all the nucleotides have equal probability, so essentially we get no help choosing among A, C, G, or T – i.e., there is no information contained in the first position. Conversely, the third position indicates a clear preference for A, so we say there is a lot of information contained at that position.

A measure called the **information content**, which is based on Shannon's entropy, allows us to quantify how informative a position in a PWM is. For nucleotides, the formula for information content at a position  $i$  is

$$\begin{aligned} & 2 + \text{Probability}(\text{position } i \text{ is A}) * \log_2(\text{Prob}(\text{position } i \text{ is A})) \\ & \quad + \text{Prob}(\text{position } i \text{ is C}) * \log_2(\text{Prob}(\text{position } i \text{ is C})) \\ & \quad + \text{Prob}(\text{position } i \text{ is G}) * \log_2(\text{Prob}(\text{position } i \text{ is G})) \\ & \quad + \text{Prob}(\text{position } i \text{ is T}) * \log_2(\text{Prob}(\text{position } i \text{ is T})) \end{aligned}$$

There is some amount of theory behind why this works, but you can get an intuition for what it measures by taking a couple of extreme cases. For position 1 in Table 2, the information content is  $2 + (.25)\log_2(.25) + (.25)\log_2(.25) + (.25)\log_2(.25) + (.25)\log_2(.25) = 0$ , or no information. Conversely, if we instead had  $A = 0.9999$ ,  $C = 0.00003$ ,  $G = 0.00003$ ,  $T = 0.00004$  at the first position, the information content would be 1.9987, or very high information – the theoretical maximum is 2, which is unattainable. Thus, the more skewed the probabilities, the more information contained at that position.

(A handy identity to know when attempting to do these calculations yourself is that  $\log_2(x) = \log_{10}(x) / \log_{10}(2)$  )

Again, we are making an assumption about what the background nucleotide frequencies are when we use information content as a measure – namely, that all nucleotides are equally likely. In order to take different background probabilities into account, we need to use a slightly different measure, called **relative entropy**.

Let  $P_A(i)$  = the probability that position  $i$  is an A,  
Let  $P_C(i)$  = the probability that position  $i$  is a C,  
etc.

Let  $Q_A$  = the background probability of an A – that is, given a random nucleotide not in the motif, how likely is it to be an A.  
Similarly for  $Q_C$ ,  $Q_G$ , and  $Q_T$ .

Then relative entropy at a position  $i$  is defined as

$$\begin{aligned} &P_A(i) * \log_2(P_A(i) / Q_A) \\ &+ P_C(i) * \log_2(P_C(i) / Q_C) \\ &+ P_G(i) * \log_2(P_G(i) / Q_G) \\ &+ P_T(i) * \log_2(P_T(i) / Q_T) \end{aligned}$$

Let's compare a couple of different background probabilities. For equal background probabilities, position 1 in Table 2 would yield a relative entropy of  $(.25)\log_2(.25/.25) + (.25)\log_2(.25/.25) + (.25)\log_2(.25/.25) + (.25)\log_2(.25/.25) = 0$

For background probabilities  $A = T = 0.1$ ,  $C = G = 0.4$ , position 1 would have a relative entropy of  $(.25)\log_2(.25/.1) + (.25)\log_2(.25/.4) + (.25)\log_2(.25/.1) + (.25)\log_2(.25/.4) = 0.32$

## II. Motif finding algorithms

The discussion about training PWMs thus far has assumed that you know where your motif boundaries are in each of your input sequences. This is not always the case, so the following algorithms are designed to find these boundaries, with the help of various amounts of prior information, and then create PWMs based on these predictions. Note also that each of these algorithms in their basic forms assume that there is only one motif to be found in each input sequence.

### A. CONSENSUS (*Hertz and Stormo 1999*)

CONSENSUS requires no additional prior information other than the size of the desired motif. Generally, it works by extracting all possible subsequences of the correct length that are found in the sequences. Then it iteratively combines these subsequences together and calculates the PWM for each set, keeping the best ones at each step. Thus it is a greedy algorithm. In detail:

1. Start with  $k$  sequences (**input sequences**) with the goal of finding motifs of length  $L$ . Using a sliding window of length  $L$ , extract all possible **subsequences** from every input sequence. For example, an input sequence of 'AATCGG' will yield 4 subsequences of length 3: AAT, ATC, TCG, and CGG.
2. For each subsequence extracted from step 1, create a new set containing only that subsequence. Note that these sets need not contain unique subsequences, since it is possible and indeed likely that different input sequences will contain identical subsequences. If each of the  $k$  subsequences are 6 nucleotides long, then for  $L=3$  there will be  $k*4$  different sets.

Definition: we say that an input sequence is **represented** in a set if that set contains a subsequence that was created from that input sequence.

Iterate k-1 times:

3. Choose a set, call it A. We can now create several new sets consisting of the contents of A and one more subsequence, and remove the original A from our pool of sets. Create one such set for each possible subsequence derived from an input sequence that was not represented in A. Do this for every original set.

4. For each new set, calculate the PWM and the relative entropy according to pre-defined background probabilities (this can just be the overall base probabilities in all the input sequences).

5. Rank the sets according to relative entropy, and keep only the top d sets.

At the end of the algorithm, each set should contain k subsequences (one from each input sequence); each set represents a potential motif.

A short trivial example:

<u>input sequences</u>	<u>possible subsequences of length 2</u>
1. ATA	AT, TA
2. CGA	CG, GA
3. CTA	CT, TA

0<sup>th</sup> iteration:

6 starting sets, with represented input sequences noted as subscripts:  
{AT<sub>1</sub>}, {TA<sub>1</sub>}, {CG<sub>2</sub>}, {GA<sub>2</sub>}, {CT<sub>3</sub>}, {TA<sub>3</sub>}

(Although {TA<sub>1</sub>} and {TA<sub>3</sub>} both contain the same subsequence, they are distinct because they derived from different input sequences.)

1<sup>st</sup> iteration: 24 sets

{AT <sub>1</sub> } + CG <sub>2</sub> → { AT <sub>1</sub> , CG <sub>2</sub> }	{ TA <sub>1</sub> } + CG <sub>2</sub> → { TA <sub>1</sub> , CG <sub>2</sub> }
{AT <sub>1</sub> } + GA <sub>2</sub> → { AT <sub>1</sub> , GA <sub>2</sub> }	{ TA <sub>1</sub> } + GA <sub>2</sub> → { TA <sub>1</sub> , GA <sub>2</sub> }
{AT <sub>1</sub> } + CT <sub>3</sub> → { AT <sub>1</sub> , CT <sub>3</sub> }	{ TA <sub>1</sub> } + CT <sub>3</sub> → { TA <sub>1</sub> , CT <sub>3</sub> }
{AT <sub>1</sub> } + TA <sub>3</sub> → { AT <sub>1</sub> , TA <sub>3</sub> }	{ TA <sub>1</sub> } + TA <sub>3</sub> → { TA <sub>1</sub> , TA <sub>3</sub> }
{ CG <sub>2</sub> } + AT <sub>1</sub> → { CG <sub>2</sub> , AT <sub>1</sub> }	{ GA <sub>2</sub> } + AT <sub>1</sub> → { GA <sub>2</sub> , AT <sub>1</sub> }
{ CG <sub>2</sub> } + TA <sub>1</sub> → { CG <sub>2</sub> , TA <sub>1</sub> }	{ GA <sub>2</sub> } + TA <sub>1</sub> → { GA <sub>2</sub> , TA <sub>1</sub> }
{ CG <sub>2</sub> } + CT <sub>3</sub> → { CG <sub>2</sub> , CT <sub>3</sub> }	{ GA <sub>2</sub> } + CT <sub>3</sub> → { GA <sub>2</sub> , CT <sub>3</sub> }
{ CG <sub>2</sub> } + TA <sub>3</sub> → { CG <sub>2</sub> , TA <sub>3</sub> }	{ GA <sub>2</sub> } + TA <sub>3</sub> → { GA <sub>2</sub> , TA <sub>3</sub> }
{ CT <sub>3</sub> } + AT <sub>1</sub> → { CT <sub>3</sub> , AT <sub>1</sub> }	{ TA <sub>3</sub> } + AT <sub>1</sub> → { TA <sub>3</sub> , AT <sub>1</sub> }
{ CT <sub>3</sub> } + TA <sub>1</sub> → { CT <sub>3</sub> , TA <sub>1</sub> }	{ TA <sub>3</sub> } + TA <sub>1</sub> → { TA <sub>3</sub> , TA <sub>1</sub> }
{ CT <sub>3</sub> } + CG <sub>2</sub> → { CT <sub>3</sub> , CG <sub>2</sub> }	{ TA <sub>3</sub> } + CG <sub>2</sub> → { TA <sub>3</sub> , CG <sub>2</sub> }
{ CT <sub>3</sub> } + GA <sub>2</sub> → { CT <sub>3</sub> , GA <sub>2</sub> }	{ TA <sub>3</sub> } + GA <sub>2</sub> → { TA <sub>3</sub> , GA <sub>2</sub> }

Calculate PWMs and relative entropies for each of these, retain the best ones according to some criterion. (Note that there are only 15 unique sets...you can remove duplicates before proceeding.)

Repeat for one more iteration ( $k=3$ ), such that each set has three members. Keep the sets with the best relative entropies – these are the putative motifs.

## ***B. Gibbs sampling***

The Gibbs sampling approach starts with a guess for where a motif is located in each input sequence, then uses those guesses to make more informed guesses. It chooses motif locations in a semi-random fashion, so it is not a greedy algorithm, but it is affected by where the initial guesses are located.

1. Start with  $N$  sequences and one initial guess for motif position for each sequence.

Iterate:

2. Pick one sequence at random. Call it  $S$ .

3. Compute a PWM using the motif locations for the remaining  $N-1$  sequences.

4. Use the PWM to assign a score to each possible motif location in  $S$ . Specifically, use the PWM score divided by the score the potential motif would get using the background nucleotide probabilities rather than the PWM.

5. Rather than just choose the highest scoring motif location in  $S$ , we are going to randomly choose a motif location. However, this is not like flipping a coin or rolling a fair die, otherwise there would be no point in using the PWM to assign scores. Rather, think of rolling a weighted die, where each motif location is weighted according to its PWM score. The highest scoring motif location will have the highest probability of being chosen, but it is also possible that some other lower-scoring location will be chosen. This degree of uncertainty is a hallmark of the Gibbs sampling strategy. The new motif location replaces the original motif location on  $S$ .

The algorithm stops after a predefined number of iterations, or until the motif locations don't change (much) anymore.

An example of one iteration of Gibbs:

5 sequences, initial guesses bolded

1. TCG**TAT**CAGCT
2. TCGAT**TTA**ACGT
3. GAT**TAG**GCAT
4. TAAGCT**CC**GAT
5. GCAT**TCAG**CTGCT

Estimate background probability by using the nucleotides in the non-motif locations: 10 As, 10 Cs, 12 Gs, 8 Ts; A = .25, C = .25, G = .3, T = .2

Leave out 4th sequence. Compute a PWM for TAT, TTA, TTA, TCA, using 0.01 to smooth all counts:

	1	2	3
A	0.002	0.250	0.745
C	0.002	0.250	0.002
G	0.002	0.002	0.002
T	0.993	0.498	0.250

Sequence 4 has 9 possible motif locations to try. (Natural log of the) ratio of probability under PWM to probability under background model

- Index 1 (TAA):  $\log( (0.993 * 0.25 * 0.745) / (0.2 * 0.25 * 0.25) ) = 2.7$
- Index 2 (AAG):  $\log( (0.002 * 0.25 * 0.002) / (0.25 * 0.25 * 0.3) ) = -9.8$
- Index 3 (AGC): -14.7
- Index 4 (GCT): -4.8
- Index 5 (CTC): -8.7
- Index 6 (TCC): -3.2
- Index 7 (CCG): -9.8
- Index 8 (CGA): -8.7
- Index 9 (GAT): -4.8

It turns out that if we randomly choose an index according to these weights, more than 99% of the time we will choose index 1, so let's say that the algorithm chooses index 1. Thus the new set of motifs is:

1. TCG**TAT**CAGCT
2. TCGAT**TTA**ACGT
3. GAT**TAG**GCAT
4. **TAA**GCTCCGAT
5. GCAT**TCAG**CTGCT

### C. Expectation Maximization

EM is a term for a class of algorithms that estimates the values of some set of unknowns based on a set of parameters (the so-called “**Expectation step**”), then uses those estimated values to refine the parameters (the “**Maximization step**”), over several iterations. In the case of motif detection, our parameters are the entries in the PWM and the background nucleotide probabilities, and our unknowns are the scores for each possible motif position in all of the sequences.

1. Start with N sequences and one initial guess for motif position for each sequence.
2. Compute a PWM using the motif locations for each of the N sequences, and the background probabilities for each base using the non-motif locations.

Iterate until the values in the PWM **converge** (i.e., the values don’t change between iterations):

3. (Expectation) Use the PWM and background probabilities to calculate the probability of each possible motif location in each sequence: for some motif location in a sequence, use the PWM to calculate the probability of that motif; then multiply that by the probability that the remaining non-motif nucleotides in the sequence are background nucleotides. You will then need to normalize all the probabilities so that they sum to 1 over each sequence. Thus, for each sequence, each index (minus a few at the end depending on the length of your motif) will have associated with it a probability of being an instance of the motif.

4. (Maximization) Now use the probabilities of motif locations to recalculate the PWM and background probabilities – instead of using raw base counts for the number of times you observe each base in the position, use weighted counts based on the probabilities. This is easiest to see in an example:

2 sequences, initial guesses bolded

1. **CT****AT**G
2. G**AT****T**A

Estimate background probability by using the nucleotides in the non-motif locations: 1 A, 1 C, 2 Gs, 1 T; A = .2, C = .2, G = .4, T = .2

Compute a PWM for TAT and TTA, using 0.01 to smooth all counts:

	1	2	3
A	0.005	0.495	0.495
C	0.005	0.005	0.005
G	0.005	0.005	0.005
T	0.985	0.495	0.495

### Expectation

There are 3 possible motif locations in sequence 1:

$$\text{Index 1(CTA): } \log(\text{probability (CTA is motif)} * \text{probability (TG is background)}) \\ = \log((0.005 * 0.985 * 0.005) * (.2 * .4)) = -13.8$$

$$\text{Index 2(TAT): } \log(\text{prob (C is bkgrd)} * \text{prob (TAT is motif)} * \text{prob (G is bkgrd)}) \\ = \log(.2 * (0.985 * 0.495 * 0.495) * .4) = -3.94$$

$$\text{Index 3(ATG): } -14.5$$

There are 4 possible motif locations in sequence 2:

$$\text{Index 1(GAT): } -11.5$$

$$\text{Index 2(ATT): } -10.8$$

$$\text{Index 3(TTA): } -5.6$$

$$\text{Index 4(TAT): } -5.6$$

These are normalized so that (non log) probabilities sum to zero in each sequence. The resulting log probabilities are as follows:

	1	2	3	4
seq 1	-9.86	0.01	-10.56	
seq 2	-6.60	-5.90	-0.70	-0.70

### Maximization

There are 7 potential motif locations with log probabilities as listed in the above table. To fill the PWM, we need to calculate the probabilities of seeing each base at each position.

For position 1, A occurs as the first base in the motif starting at index 3 in sequence 1, and as the first base in the motif starting at index 2 in sequence 2. That means that the probability of seeing an A in the first position of the motif is equal to the probability that motif starts at either of these two locations, which is equal to the probability of the motif starting at index 3 in sequence 1 **plus** the probability of the motif starting at index 2 in sequence 2, divided by the sum of the probabilities of starting in all 7 locations (which is 2 because all the probabilities for each sequence sum to 1, and we have 2 sequences).  $(e^{-10.56} + e^{-5.90}) / 2 = 0.0014$

(e raised to the power of the log probability gives you the un-logged probability)

C only occurs at index 1 in sequence 1, so the probability that a C is in the first motif position is  $e^{-9.86} / 2 = 0.000026$

Similarly, the probability of G and T can be calculated; they are 0.00068 and ~1 respectively.

For position 2, A appears only twice in legal positions – as part of the motif that starts at index 2 in sequence 1, and as part of the motif that starts at index 1 in sequence 2. This leads to a probability of  $(e^{0.01} + e^{-6.60}) / 2 = 0.51$

And so on.