Recitation 5 Notes: Motif Discovery
10/14/2016

Agenda
1. Review of Lecture/Motif Discovery
2. Suffix Trees and WEEDER (*Updated to reflect recitation question on mismatches)
3. Gapped Motif Discovery
4. In-Vitro Motif Discovery: PBM, Selex

Overview of Motif Discovery
In lecture, we saw that there were various ways to identify regulatory “motifs” (Slide 11 in Lecture 10 Notes):
1. Maintaining and iterating to find a PWM (covered in lecture): “guess” the representation for the motif and check on the sequence that we have, continue until convergence
2. Enrichment: Motifs are likely to be found than the background expectation for the kmer.
3. In Vitro Analysis: Can we chemically identify where in the genome various TFs and DNA binding proteins (the functional definition of motif)
4. Conservation analysis: common patterns that occur in multiple genomes in similar locations are likely motifs. We can discover by looking at conservation level (covered in lecture)

And that there were some special cases:
1. How do we handle gaps/variable length motifs?

Today in recitation, we’ll:
1. Briefly review PWMs
2. Cover an example of motif discovery by “frequency” (WEEDER)
3. Cover two in-vitro methods for motif discovery (SELEX and PBMs)
4. Discuss how to identify variable-length/gapped motifs

Review of PWM
When using the PWM and an EM/Gibbs Sampling approach as described in lecture, we framed the problem as follows:
1. B: the background model for nucleotides in the genome
2. M: a position weight matrix that represents the probability of a nucleotide at each point in the motif
3. X: a sequence of nucleotides

Example
PWM:
Interpreting a Motif as a probability distribution allows us to:

1. Be more flexible defining a motif (allow for variability in nucleotides)
2. Gives intuition for which nucleotides in our guessed motif are most important or if our guessed length is too long (then the remaining nucleotides will not be important)

To discover motifs using this representation, we need to find the probability weights for the PWM via expectation maximization:

1. Estimate the starting positions of the motif based on the existing PWM
2. Use those starting positions to update the probabilities within the PWM

Our goal is to converge to a motif’s true representation and the starting locations of that motif within the genome (that is the maximum/optimal step in the EM and we can’t do “better”)

Estimate the starting positions:
For a position i from 1...L, the probability that the motif starts at k, is $P(B_{1...k-1})P(M_{k...k+w})P(B_{k+w+1...L})$ where $w$ is the length of the PWM and the motif we’re expecting to find.

$P(B_{1...k-1})P(M_{k...k+w})P(B_{k+w+1...L}) =$
P(B from 1...L)*P(M from k... k + w)/P(B from k...k+w). The P(B from 1...L) is the same for each, so in essence, we’re just comparing the ratio of the probability between the motif and background model for each index in the sequence.

From the starting position, obtaining the maximum likelihood motif: you look at the starting positions and compute the frequency counts of the nucleotides based on that starting position, and then use that to populate the matrix (see lecture slide 31)

1. EM: average across all starting positions weighting by the probability that the motif starts at that point
2. Gibbs: sample one starting point based on the probability distribution of starting positions and use that to populate the PWM
3. Greedy: select the maximum probability

Why gibbs in the pset:
- Much faster (sample one)
- More versatile
- Less dependent on the initial parameters (can change the PWM more)

Why gibbs might not be the best in all cases:
- More dependent on signal: you need to have the motif in all sequences

**Frequencies: WEEDER**

WEEDER Algorithm (Pavesi, et al - Bioinformatics 2001)

How can we keep track of common k-mers and their close variants? One reason this is complicated is that we might want to consider very similar sequences that deviate by ~1 character (AGCTC, AGCTG) as part of the same motif, but hashing might place them at farther distances

Framing the problem: we want to find all patterns of length M that occur with at most e mutations in at least q sequences.

In other words, we want to discover motifs that occur in a lot of sequences but with some tolerance for error (the same type of tolerance we get with the PWM approach)

We use a **generalized suffix tree** (relevant for problem set 4)

**Suffix Trees**

Core principle: the string obtained by concatenating all the string-labels found on the path from the root to leaf i spells out S[i..n]

Example: Banana. Add a special character ($) to indicate termination.

How to create: List out all the suffixes of the string
BANANA$
ANANA$
NANA$
ANA$
NA$
A$

Guiding properties:
- A String of length n has n leaves, each of which represents an index in the string itself. For example, in the example above
  - Every path from root to leaf i gives the suffix S[i...n]

Create the root from the shortest suffix

R → A → $
NA$ starts differently, so create another edge from root.
The ANA$ suffix also starts with A, so reuse the A from the smaller suffix
NANA$ suffix starts with NA, so reuse from NA$
ANANA$ suffix starts with ANA, so reuse from ANA$
BANANA$ suffix starts with new character, so create as its own


Computational properties:
1. Gives $O(|\text{query}|)$ lookup times for substring matches (find the prefix of a suffix)
2. $O(n^2)$ letters, but optimization gives you $O(n)$ space if you store each string as a position and length of substring of S
**Generalizing Suffix Trees:**

Question: What if we wanted to encode the fact that we had two separate strings BANANA? If we retain the same structure, then we would overwrite the same positions in the suffix tree.

Answer: use a different escape character.

Specifically, we:
1. Add a marker symbol for end of each sequence that is unique
2. Say we have k sequence: add on each node a string of k bits where the ith bit is 1 if sequence 1 has a suffix ending at a node

Now, if we want to find a pattern p in our k sequences, we can start at the root until we have matched all the characters, and then we can find which sequences we matched with.

Handling errors: we can even accommodate mismatches (flexible matching of character k in a sequence of length L) by traversing all paths at a node and keeping track of the number of mismatches. Very useful for motif discovery

**Weeder Algorithm:**

*Remember the goal:* we want to find all patterns of length M that occur with at most e mutations in at least q sequences.

**Step 1:** Preprocessing. Given your k sequences, form the generalized suffix tree.
**Step 2:** Search for patterns in the tree that meet the q,e requirements

Start with empty pattern p = "" at root
While paths still exist:
   - Match the first symbol on each edge leaving node
   - If at least q sequences are at the end of this edge, continue
     (adding 1 to our mismatch count if necessary)
   - If threshold (see below) not satisfied: terminate path
   - If path length == M: return path
   - get_new_paths(node) (see below -- adds paths to check)

What we get: a list of substrings (paths from root to another node) that are at least of length M with at most e mismatches (meaning we sum both edges of the tree) where there are at least q sequences of the k that have this substring

**Handling Mismatches**

WEEDER is “almost exact”: as a heuristic, WEEDER only considers paths with “valid” prefixes as “valid” themselves. A “valid” sequence is where the ratio between the number of mismatches and the length of the pattern is below some threshold E. For example, if E is 0.2, then, we can tolerate 0 mismatches for sequences of 4, 1 for sequences of length 5, etc…. However, the
prefix constraint also prevents us from having those mismatches occurring in the first 4 nucleotides (otherwise the prefix would not be valid).

At each node in the suffix tree in the pseudocode above:

```python
get_new_paths(node):
    # gets the rest of the paths to check at those node
    # in search, think about this method as getting the new search paths to try and adding to the agenda
    For each letter in {A, G, T, C}:
        For each edge from node:
            Next_letter = node.edge.value
            If next_letter != letter:
                # mismatch is present
                Mismatch += 1
            If num_sequences(node.edge) > q and mismatch < E*path.length:
                Add_path()
                Pattern += letter
```

To summarize:
At each node, given the previous pattern p1….pn, we’re considering all the possible sequences that have that pattern and an extra nucleotide at the end (p1….pnb for b in [a, g, t, c]) and then looking at that pattern for each of the additional letters that are contained within the suffix tree (some of these involve a mismatch).

For example:
If our current sequence is AGCTGC (with 1 mismatch and E = 0.2) and the node in the suffix tree for C has edges to T, C, G, then:
1. we will consider 4 new patterns (AGCTGCA, AGCTGCC, AGCTGCT, AGCTGCG)
2. Those patterns will have corresponding mismatches increments (+1, 0, 0, 0)
3. For each of the edges T, C, G
   a. The first pattern with invalid because it has too many mismatches 2/7 > 0.2, so we don’t expand that one anymore
   b. The patterns where the additional letter does not match the edge letter are discarded because they are no longer valid
   c. For each of the ones that remain, we check whether there are q sequences left that have this pattern in the tree. If not, discard them also.
Variable Length Motifs/Gaps: GLAM

Problems with the PWM representation
- requires us to know the length of the motif to find it
- sometimes, positions in a motif are not independent

Don’t Know the Length:
1. You can try many different lengths and pick the “best one”. Which length is best?
   a. Assume that there is a small number of mismatches and choose the best motif based on that (WEEDER)
   b. Try a fixed length, and then use EM to extend the sequence to see if the motif model is better. M: add one more column. E: recompute the PWM + parameters.

Need for a more sophisticated model: what if the motifs themselves could have variable length gaps?
1. HMM (as discussed in lecture): an HMM for the motif where emissions are the nucleotides that comprise the motif, but a “gap” state is included that can self-transition
2. GLAM

GLAM: Gapped Local Alignment of Motifs (paper link)
Key idea: key positions of a motif must match in alignment, but with insertions/deletions in between that are not penalized much
New representation: a motif is a set of important residues that are interspersed between positions where the nucleotide doesn’t matter

GLAM2 creates a scoring scheme for alignments. With this scheme, it finds the motif alignment amongst k sequences with maximum score. It samples this score and iteratively tries to improve it (simulated annealing) until a maximum is reached.

Score each motif’s alignment via:

\[
\log \left( \prod_{k=1}^{W} \frac{p(c_k)}{\prod_{k=1}^{W} p(c_k)} \right) \\
W = \text{width of the alignment} \\
A = \text{number of possible boxes} \\
p(c_k) = \text{how often the } k^{th} \text{ box occurs} \\
c_{k} = \text{how often the } k^{th} \text{ box occurs in the } k^{th} \text{ position of the alignment} \\
P(c_{k}) = \text{how likely are we to see the counts of boxes at the } k^{th} \text{ position of the alignment if we do in fact have an alignment?}
\]

In other words: score each alignment by comparing:
- Numerator: multiply for each base pair in the alignment, the probability of seeing the alignment we have at that base pair
- Denominator: multiply for each base pair position: for every possible base we could have, the probability that the ith base occurs cki times in the alignment

Example: scoring the first column of this alignment

Alignment
AGTCAG
GCATAC
GCATAC


How do we calculate P([A, G, G]) in a way that can incorporate information about gaps/insertions/deletions?

GLAM2, in addition, allows deletions and insertions in the alignment. The numerator in the log likelihood ratio formula now becomes:

\[
\prod_{k=1}^{W} P(\bar{c}_k) \prod_{k=1}^{W} P(d_k) \prod_{k=1}^{W-1} P(r_k)
\]

In other words, in the numerator, we also add in probabilities for deletions at index k (dk) and insertions at index k (rk)

In GLAM, what do alignments look like?
*: key position (don't need to have consensus in all sequences)
**   *****   **

ATTCAGAGGC...AT
A.TG.GA.TA...AT
ATGC.GA.GA...AT
T....GAGTC...AT

**In-vitro Motif Discovery: Selex, PBM**

**PBM**s (Protein Binding Microarrays) ([paper](#))
Goal: Identify the specificity of a transcription factor binding to all possible DNA binding sites, which thus lets us find motifs. Based on PBM technology ([paper](#))
The PBM provides reads for TF interaction, but a key issue is that there is too many different k-mers to test for (4^k of them) when testing all possible combinations of a motif of size k. Instead, we need to strategically test for smaller k-mers and aggregate the results together to find TF-binding motifs.

Strategy:
- Design an array with probes of size b > k that together contain all possible k-mer
- Find the shortest string that contains all the kmers as use that as a probe

Imagine directed graph, each node is a possible kmer (total 4^k nodes) and edges exist between k-mers that overlap in k-1 base pairs. We can choose a value b by “joining” some of these k-mers together (specifically b -( k -1)) of them, and therefore, with probes of size b, we only need *(this is a high level estimation - for more detailed discussion of algorithm, please see paper)

N = 4^k/ (b - k + 1) probes. For k-mers of length 10, this saves a factor of about 20 in terms of number of probems required

Paper experimentally verifies that this strategy works (number of binding sites found are same)

**Selex (Systematic Evolution of Ligands by Exponential Enrichment)**
A chemical way of finding DNA oligonucleotides that bind to a specific ligand

Process:
1. Generate a large oligonucleotide library by transcription of DNA
2. Expose this large library to the target ligand
3. After giving time for binding, remove/elute the DNA via affinity chromatography/some other method
4. Elute the bound sequences, and amplify via PCR (Reverse Transcriptase-PCR)
5. Recurse to step 1, but now make your binding conditions even harsher
Eventually, through many cycles, you’ll get the DNA sequences that bind most tightly to a ligand of choice