Motif Discovery Upper Bound

An Upper Bound on the Hardness of Exact Matrix Based Motif Discovery

Paul Horton & Wataru Fujibuchi

Computational Biology Research Center
Tokyo, Japan

Talk Outline

- Motivate Motif Discovery Application
- Brief Review of Previous Results
- Outline our Result

Experimental Approaches

It is not possible to directly observe the transcription process. But some measurement of transcription factor binding can be done:

- Footprinting
- CHip on CHip

These techniques are useful but expensive and subject to error.

Claim: Purely experimental approach not sufficient

Naïve TF Binding Site Discovery

foreach(Transcription factor t){
  foreach(short region of DNA r){
    do molecularDocking(t, r)
  }
}
print(binding pairs)

Unfortunately, molecular docking computation is very slow and somewhat unreliable. . .

Claim: Physical simulation not feasible

Sequence Comparison

- Identify or Postulate Co-expressed Genes
- Look for motifs in their Promoter Regions

Example Binding Sites

cdc9	ctttcatcaTTACCCGGATgattctctata
act1	atttctCTGTCACCCGGCCtctattttcca
fas1	ggctcctcggaggcCGGGTTAtagcagcgt
ilv1	aagaaaaagcgcagCGGGTAGCAAatttgg
rpo21	atcataCGGTAACCCGgacctCCATTACCC
swi5	tctgaatcccaaTCCGGGCAAgtagagagg
top1	aacctttttcacTCCGGGTAAtacctgctg
tpi1^r	gaaaagtcttagaaCGGGTAATCTtccacc
matches	+***

Upper case letters represent experimentally determined binding sites as annotated in SCPD.

Definitions

Input: n strings from DNA alphabet, motif width parameter w, and a background model.

Alignment: multiset of n length w substrings taken from the input sequences.

Background Model: probability distribution over DNA alphabet, e.g. length one 0° Markov Model. Represented as $B(c)$ for the probability of character c.

Motif Model: length w, 0° Markov Model (DNA alphabet). Represented as a $\sigma \times w$ matrix, $M(c, i)$ for character c at position i of the motif. *aka* Position Specific Scoring Matrix PSSM or affinity matrix.

Definitions cont.

Count Matrix: C_A is a $\sigma \times w$ matrix of the counts of each character at each position for the strings in a given alignment A

Task: Find an alignment and matrix model such that the likelihood ratio that that strings in the alignment were generated by the motif model *vs.* the background model is maximal

$$\max_{A,M} \prod_{c,i} M(c,i)^{C_A(c,i)} / B(i)^{C_A(c,i)}$$

$$\max_{A,M} \sum_{c,i} C_A(c,i)(\log(M(c,i)) - B(i))$$

Fairly high dimensional, discrete non-linear optimization

Regarding the Score Function

$$\max_{A,M} \sum_{c,i} C_A(c,i)(\log(M(c,i)) - B(i))$$

- Equivalent to entropy for uniform background model
- Column independence (somewhat) justified by properties of DNA
- Score function (somewhat) justified by models of evolution, Berg & Hippel 89?

Claim: Score Function is Reasonable

Previous Work: Complexity

- exact problem
  - NP-hard, Li *et al.*, 99
- polynomial time approximation
  - MAX SNP-hard, Akutsu *et al.*, 00
  - More work in Li *et al.*, 02

Note: All hardness results require unrealistically large motif lengths for bioinformatics applications.

Previous Work: Heuristics

- Tree Search
  - Stormo & Hartzell, 89
  - Hertz & Stormo, 99
- Expectation Maximization
  - Lawrence *et al.*, 90
  - Meme Program, Bailey & Elkan, 95
- Gibbs Sampling
  - Lawrence *et al.*, 93

Many, many works omitted. . .

Exact Algorithms: Berkeley BB

Horton, Pacific Symposium on Bioinformatics, 96.

- Branch & Bound
- Very sensitive to size of the sequence input even for small w
- No non-trival lower bound on time complexity
- Practical in some instances, often worthless

Exact Algorithms: Tsukuba BB

Horton, CPM00, Journal of Computational Biology 01.

- Branch & Bound
- Can solve large instances **if** w is very small
- No non-trival upper bound on time complexity *at the time*
- Practical in some more instances, often worthless

In practice typical motif widths are 5-20 bases long. Depending on the input size and strength of motif pattern Tsukuba BB works well for width 5-7.

Search Strategies

Task is to find the optimal pair M^*, A^*.

If A^* is known, M^* is easily obtained: $M^* = C_{A^*}/n$

Most heuristics search over alignments.

Search Strategies

If M^* is known, A^* is easily obtained:

Let U be the set of possible strings of length w.

If M^* is known, use M^* to rank the strings in U, by score. We denote this permutation of U as P^* and call it the Best Score First (**BFS**) permutation of M^*.

A^* is obtained by selecting the n highest ranking (by P^*) length w substrings in the input sequences.

Note: P^* is sufficient to obtain A^*

Example

$w = 2$, Seqs = TAG, ACT, CGA

$M =$

	0	1
A	$\frac{2}{3}$	0
C	$\frac{1}{3}$	$\frac{1}{3}$
G	0	$\frac{2}{3}$
T	0	0

$P =$ AG, AC, CG, CC, ...

Naïve Exhaustive Algorithm

Let U be the set of possible strings of length w.

Naïve Algorithm Simply search over all permutations of U.

Running time is $O(\sigma^w!)$.

Very nasty but (ignoring linear terms) *independent* of the number and length of input sequences.

Most Permutations Irrelevant

Let P^M be the permutation of U in monotonically decreasing order of the strings score by some motif model M.

Observation: Most permutations of U cannot be the BSF permutation for *any* M.

Detail: For a fixed matrix M, the BSF permutation P^M can be defined uniquely. (score ties can be broken by infinitesimal perturbation of M).

Example Hopeless Permutation

Consider $P =$ `AA, CC, ...`

minitheorem: P `is not the BSF permutation of any matrix.`

`Assume` P `is BSF for matrix` M`.`

$$\forall_{b \neq \mathrm{A}} M(\mathrm{A}, 1) > M(b, 1)$$
$$\forall_{b \neq \mathrm{A}} M(\mathrm{A}, 2) > M(b, 2)$$

$$\mathrm{score}(\mathrm{A}C) > \mathrm{score}(\mathrm{C}C)$$
$$\mathrm{score}(\mathrm{C}A) > \mathrm{score}(\mathrm{C}C)$$

Extending Partitions

Theorem: Given a partial permutation $P_1 \ldots P_k$, which is part of the BSF permutation for some matrix. There are at most $\sigma(w-1)$ extensions of size $k+1$ which are consistent with *any* matrix.

character debut

Definition: The first string in the BSF permutation P^M of some matrix M consists of the best scoring, *consensus* characters for each column of M.

Observation: When a character first appears in some column of P^M, it occurs as a one character substitution into the consensus string.

Example: if the consensus string is “aaa”, the first appearance of character “b” in column one must be “baa”.

character rank

Definition: We define the rank of a character b in column c as the number of unique characters which appear before it in P^M $(+1)$.

string	rank information
aa	rank('a',1) = 1, rank('a',2) = 1
ac	rank('c',2) = 2
ag	rank('g',2) = 3
ca	rank('c',1) = 2
cc	

It turns out this also corresponds exactly to the rank of character b in column c of M.

Our Permutation Generator

Algorithm to find all possible consistent extensions of a permutation $P_1 \ldots P_k$.

For each pair (column c, non-consensus character b):

Rule 1: If b is of unknown rank, add the consensus string with b substituted into column c.

Rule 2: Otherwise let a denote the character of one better rank than b, consider the strings in $P_1 \ldots P_k$ in which a occurs in column c. Substitute b for a in each one of those and pick the first one (if any) that does not appear in $P_1 \ldots P_k$.

Example

Let $P_1 \ldots P_k = \mathrm{aa}, \mathrm{ac}, \mathrm{ag}, \mathrm{ca}, \mathrm{cc}$

Candidates for P_{k+1}:

by rule 1: $\mathrm{at}, \mathrm{ga}, \mathrm{ta}$

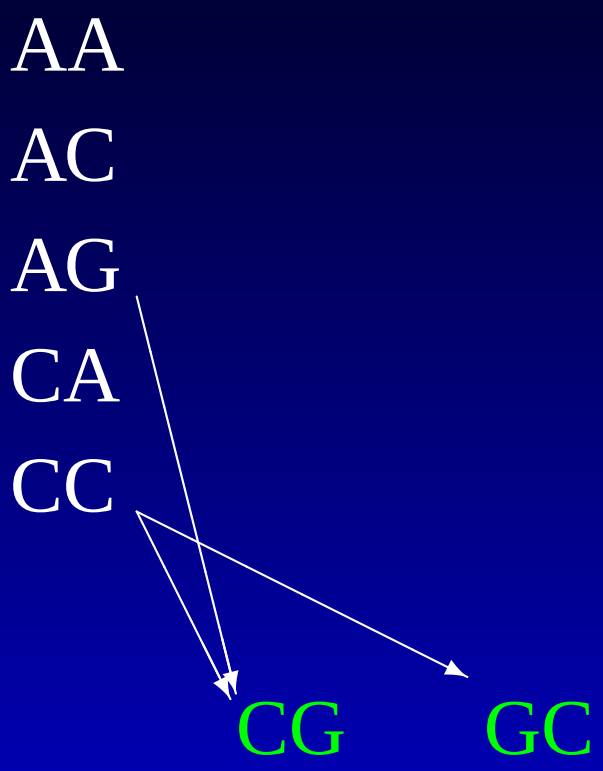
by rule 2: cg, gc

discarded: $\mathrm{gg}, \mathrm{ct}, \mathrm{gt}, \mathrm{tc}, \mathrm{tg}, \mathrm{tt}$

Rule 1: If b is of unknown rank, add the consensus string with b substituted into column c.

Rule 2: Otherwise b for a in each one of those and pick the first one (if any) that does not appear in $P_1 \ldots P_k$.

Same Example, Rule2 Shown



Candidates generated by rule 2 shown in green.

Proof of Correctness

omitted!

Key idea is the equivalence between the rank of characters in column c defined in terms of P^M and the rank of the scores of those characters in M.

Running Time

Let r be the maximum number of terms used in when applying any permutation P to the input. $r < \sigma^w$. Denote σ^w as u.

naïve: Search depth of r, branch factor is $O(\sigma^w)$. Running time: size of input plus $O(u!) \approx \frac{u}{e}^u$

our: Search depth of $\leq r$. Branch factor is $\leq w(\sigma - 1)$. Running time size of input plus $\approx O(u(\sigma \log_\sigma u)^u)$.

still a lot worse than exponential but big improvement. Remember σ and w are small in practice and this is a worst case analysis.

Conclusions

- Some real problem instances can be solved (see paper)
- Efficient algorithms are possible if w grows very slowly with size of input m,
  $\log \log m < w < \log m$.
- Open Question, can NP-hard or even MAX SNP hardness results be obtained for $w = O(\log m)$?