

Lecture 9: Motif Finding (II) - October 20, 2006

Lecturer: Asst. Prof. Sung Wing Kin, Ken

Scribe: Dong Difeng & Liu Ruizhe

9.1 Introduction

Previous lecture has introduced motifs in both biological sense and computational sense. Moreover, some current motif finders such as Gibbs sampling, SP-STAR, Winnower and suffix-tree motif finding are introduced. However, those algorithms and motif finders have difficulties to deal with locating motifs. Moreover, they are unable to find motifs with gaps. In this lecture, algorithms locating motifs and finding gapped motifs are discussed. Also, we introduce 2 motif finders, LocalMotif, which is used to find and locate localized motifs, and SPACE, which is a motif finder allowing arbitrary number of gaps.

9.2 Finding Local Motifs

Current motif finding algorithms require users to specify the motif locating regions, i.e. the sequence region containing the motif. Since motif is defined as the over-represented coding region on a gene, which means it has relatively high rate of occurrences comparing with other DNA code. Hence, if a sequence gets too long, the noise, which is non-motif region, will become sensitive and confuse the algorithms. On the other hand, if a sequence picked is too short, we may miss some real motifs. For example, in the figure below (Figure 9.1), the diamond and ellipse regions are binding sites while the rectangles are noise. We can see that if the sequences get too short, within boundaries a and b , we will miss the diamond motifs. While if the sequence is too long, within boundaries a and c , the rectangles, which are noises, will be over-represented. Thus the accuracy will fall down.

In real life, some motifs are location specific. For instance, the *TTGACA* motif always occurs at region around positions -40 to -30 relative to the transcription start side (TSS). Figure 9.2 shows the concentration of *TTGACA* motif on a set of 471 *E.coli* promoters anchored by TSS. The figure shows that *TTGCCA* motif concentrates at position 40, which is 30 positions ahead of the TSS (70).

This means that motifs are not independent from positions. There are some motifs whose binding sites are localized. If we do not know the occurring region of a particular motif, we will be blind, and we are unable to accurately compute



Figure 9.1: Sequence length affects motifs

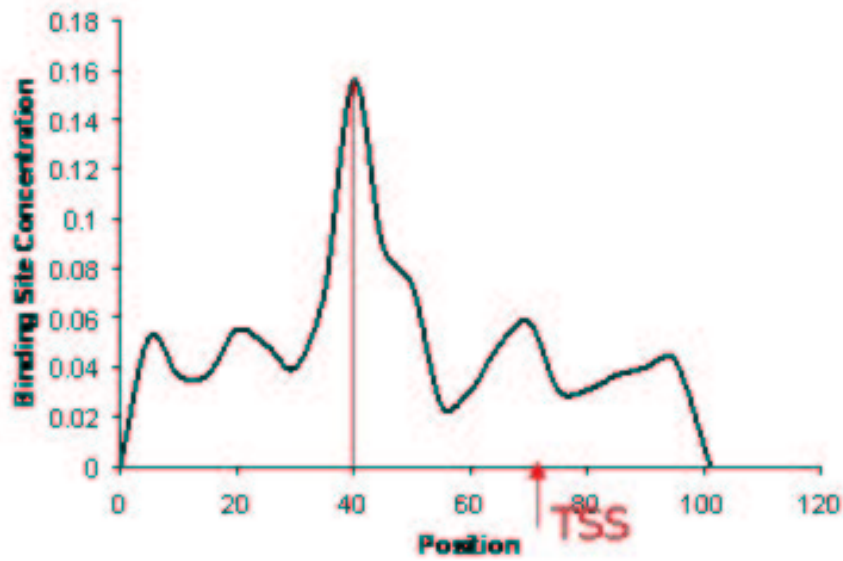


Figure 9.2: Real life experiments on E.coli promoters

the binding locations of the motifs. However, we can easily get a naive solution for finding such local motifs: For all possible sequence intervals relative to an anchor (say, TSS), Run a usual motif finding algorithm to extract motifs.

This is a definitely brute-force algorithm. It consumes too much time and computational power since we try all combinations on a set of long sequences. Another problem is that since we compute all possible intervals and get all possible motifs, we are lacking of method to pick the best motif found.

To solve these problems, we describe a solution called "LocalMotif" [Narang06]. LocalMotif takes a set of sequences with anchor as input, and aims to find a pattern M such that

- M is over-represented;
- M is significant according to a background model;
- M is localized in an interval (p_1, p_2) relative to the anchor.

The first condition is evaluated using Z-score. Recall that a word is "over-represented" if the number of occurrences is relatively larger comparing with other motifs. For example, in Figure 9.1, within the boundaries a and b, the ellipse motif is more over-represented than the rectangle one. Now we define "Z-score".

Definition 9.1 Given N sequences of length L , suppose a motif M occurs n times, based on q -order Markov model, we estimate:

- e = the expect number of occurrences of M
- σ = the standard derivation:

$$z\text{-score} = \frac{\frac{n}{NL} - e}{\sigma}$$

An entropy score is the square of z-score:

$$D\left(\frac{n}{NL} || e\right) = \frac{\left(\left(\frac{n}{NL}\right) - e\right)^2}{\sigma^2}$$

where $D(i|j)$ denotes the relative entropy of i to j .

The second condition is evaluated using relative entropy. Firstly, a word M is said to be "significant" if M looks different to the patterns in the background model. For example, consider two motifs $AATAA$ and $CCCCC$ and a background sequences (Figure 9.3). Motif $AATAA$ has 6 occurrences in the background while motif $CCCCC$ has 0 occurrences. In this case, we say $CCCCC$ is more significant than $AATAA$ with respect to the background. Such significance can be evaluated by "KL-distance".

acacgaataacgaataaggca
 ttaataatacatttgga
 agtacataataacgaataacg
 gggataccaataacatagacc

Figure 9.3: A background

Definition 9.2 Given a positional weight matrix PWM of a motif, denote $f_{b,i}$ to be the probability that the i th position of M is nucleotide b . The KL-distance is computed by:

$$KL\text{-distance} = \sum_{i=1}^l \sum_b f_{b,i} \ln\left(\frac{f_{b,i}}{p_b}\right)$$

where p_b is the probability of the occurrence of the nucleotide b in the background sequences.

If the motif M looks similar to the background, the KL-distance is smaller.

The third condition is called the localization score. Basically, a motif which is more concentrated in a short interval will be assigned a higher localization score than a motif which is less concentrated (Figure 9.4). The localization score is computed as follows:

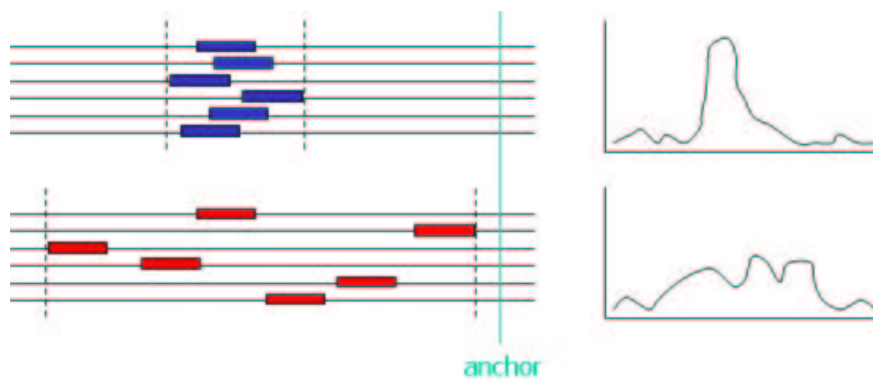


Figure 9.4: Localization: Blue (upper) is more localized in an interval

Definition 9.3 Given a histogram of the binding site density (Figure 9.5), the localization score of interval (p_1, p_2) is defined as the area difference between the area under the density curve and the area under the uniform distribution line. Given the density line f and the uniform distribution e , the scoring function is defined as:

$$\text{Score} = \int_{p_1}^{p_2} f_M(p) - e(p) dp$$

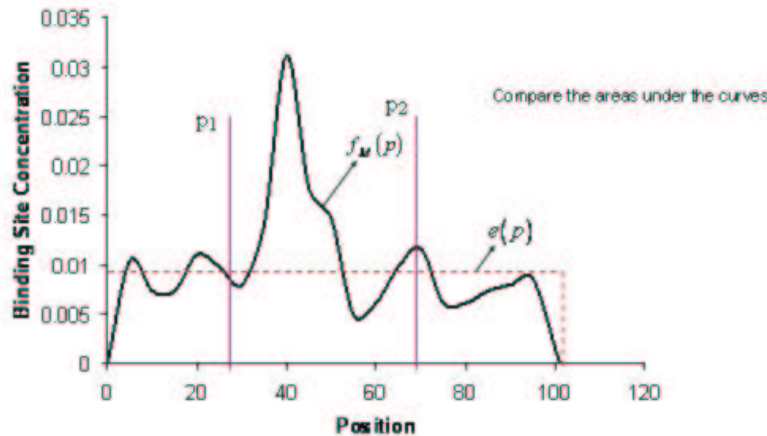


Figure 9.5: Binding site density and scoring function

However, in practice the distribution of binding sites is not accurately known due to the insufficient available sampling sequences. This means we do not know the PWM and the density curve. Instead of computing the relative entropy between the PWM and the background, we compute the relative entropy between the observed occurrences of M and the expected occurrences of M . Consider a motif M occurring in (p_1, p_2) . By random, we expect the proportion of occurrences of M in (p_1, p_2) is $c_0 = \frac{|p_1 - p_2|}{L}$. Denote the actual occurrences of M in (p_1, p_2) to be n_1 , and the actual occurrences of M in the whole region to be n_2 . Then the actual proportion of occurrences of M in (p_1, p_2) is $c = \frac{n_1}{n_2}$. (Figure 9.6)

Hence the localization score now is defined to be:

$$D(\hat{c}||c_0) = \hat{c} \ln\left(\frac{\hat{c}}{c_0}\right) + (1 - \hat{c}) \ln\left(\frac{1 - \hat{c}}{1 - c_0}\right)$$

Finally, the total score is just the sum over the localization score, the relative entropy and the z-score.

Total-score = Localization-score + Relative-entropy + Z-score

$$D(\hat{c}||c_0) + KL\text{-distance} + D\left(\frac{n}{NL}||e\right)$$

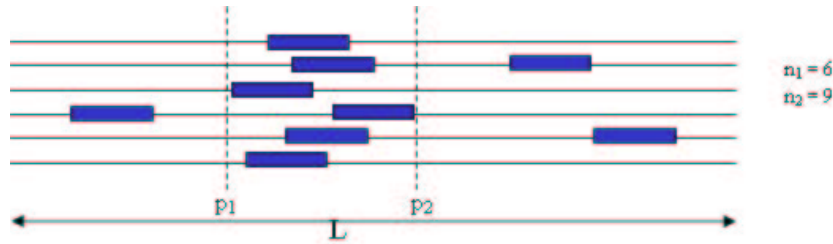


Figure 9.6: Statistical measure for localization score

$$= \hat{c} \ln\left(\frac{\hat{c}}{c_0}\right) + (1 - \hat{c}) \ln\left(\frac{1 - \hat{c}}{1 - c_0}\right) + \sum_{i=1}^l \sum_b f_{b,i} \ln\left(\frac{f_{b,i}}{p_b}\right) + \frac{\left(\left(\frac{n}{NL}\right) - e\right)^2}{\sigma^2}$$

The algorithm is illustrated as following:

Input: A set of N DNA sequences of length L each, aligned relative to an anchor point.

Goal: Find an unknown pattern M of length l with at most d mutations within an unknown sequence interval (p_1, p_2) .

Steps: For all 4^l patterns in all possible intervals:

1. Compute the total score of the pattern.
2. find best score and report the motif.

The algorithm is a brute force approach and hence is the most sensitive algorithm. We can speed up it by some heuristic observations.

Firstly, scores for longer intervals can be instantly computed from those of shorter constituent intervals. Hence we can weed out similar consensus and overlapping intervals. For example, consider two constituents: *AAACATGG*[100, 200] with score x and *AAGCATGG*[50, 70] with score y . If $x > y$, then the former one is remained and the latter one is weeded out as the former is longer. Needleman Wunch algorithm is employed to test similarity of consensus.

Secondly, only l -mers found within the sequence set are scored first. Here some heuristic method is employed to extend the score to other patterns:

Step 1: Compute average score of the n top-scoring motifs.

Step 2: Cluster the n top-scoring l -mers as per goodness of their alignment.

Step 3: For each cluster:

1. Compute the majority pattern

2. Score the majority pattern in all intervals with fast method
3. Insert the majority pattern records in the motifs list

Step 4: Repeat until the average score stops increasing.

A simulation experiment is done to test the algorithm. In the experiment, each dataset consists of N nucleotide sequences of length L , each of which is generated from a background Markov model of order q . $k\%$ of those N sequences are implanted with instances of (l, d) -motif M within a local position interval (p_1, p_2) . Then each interval is of length $|p_1 - p_2| = \bar{p}L$. Each data point in the simulation is based on experiment on 100 simulated dataset. The parameters are illustrated in the following figure. (Figure 9.7)

Parameter	Range
N	50-100
L	200-1000
q	0-2
k	20-100
(l, d)	(6, 1)-(10, 3)
\bar{p}	10-100

Figure 9.7: Parameters of simulation experiment

The result is shown in the following figures (Figures 9.8,9.9,9.10). Here three well-known algorithms are compared to run the simulation. The rectangle data points represent the results of MEME ; the triangle data points are the results of Weeder; and the diamond data points are from our algorithms. Figure 9.8 shows the correctness of motif detections related to the sequence length. We see that overall the accuracy falls down as the sequence length grows. Here, LocalMotif gives the highest accuracy over all three finders. Moreover, MEME and Weeder drops more quickly than the LocalMotif. Figure 9.9 shows the correctness of motif detections depends on the percentage of sequences containing motifs. Overall, the higher percentage of sequences containing motifs, the higher accuracy it gives. LocalMotif still perform best here. It beats other two motif finders on accuracy by at least 10 percents. Figure 9.10 shows the percentage of predictions falling in the overlap range.

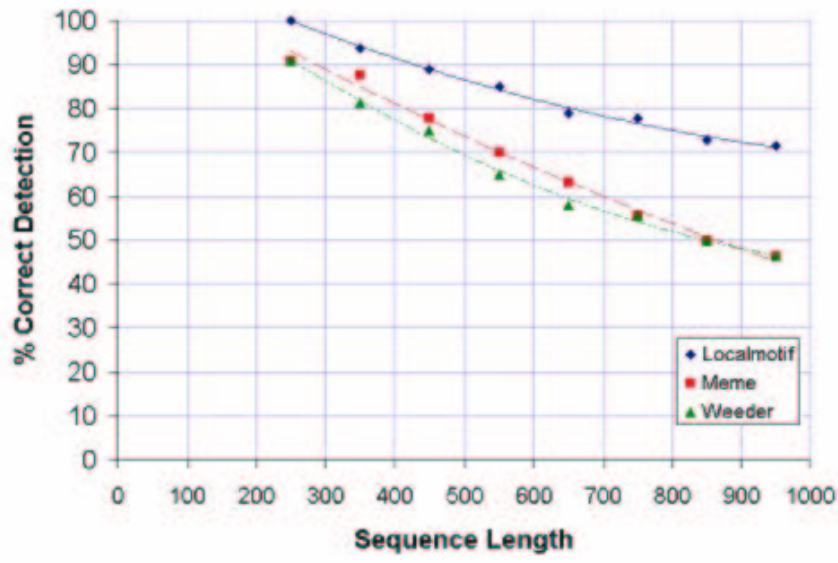


Figure 9.8: Correct detections to the sequence length

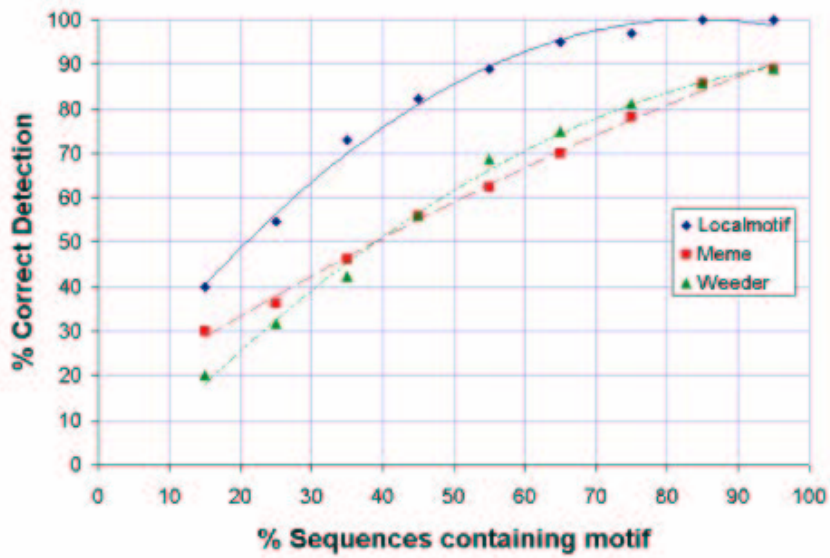


Figure 9.9: Correct detections to the percentage of sequences containing motifs

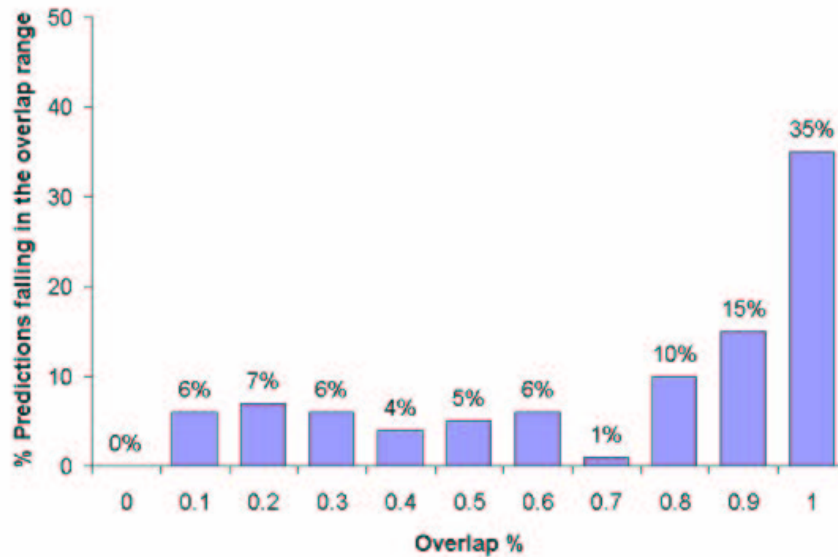


Figure 9.10: Percentage of predictions falling in the overlap range

Experiments on real life datasets also have been done. One experiment is done on *E.coli*. In the literature, *E.coli* has two motifs in their promoter regions relative to TSS. One is *TATAAT* at position -10, which is the TATA box; the other is *TTGACA* at position -35. The experiment is performed on 471 *E.coli* promoters of length 101, with TSS starting at position 70. The background is set to be order 2 Markov model trained using *E.coli* intergenetic sequences. The result shows the top 3 (6, 1)-motifs reported by LocalMotif:

- *TATACT* at [50, 70] (*score* = 165.623232) ← *TATAAT*
- *TACAAT* at [50, 70] (*score* = 94.976349) ← *TATAAT*
- *TTGACA* at [30, 50] (*score* = 92.0044189) ← *TTGACA*

Another experiment is done on *Drosophila* data to detect TFBS in *Drosophila* proximal promoter sequences. The dataset contained 1941 promoter sequences of length 300 bp each with TSS at position 250. LocalMotif sets its background to be order 2 Markov model trained using *Drosophila* intron sequences. Comparing with MEME, the following 2 tables (Figures 9.11, 9.12) shows the result. LocalMotif is able to find 4 motifs and 1 new motif. While MEME only find DPE and also the same new motif. The second table restricts the search for MEME to the core promoter sequence region (-60 to 40 respect to TSS). The table shows that if we restrict MEME to search a small region, From Figure 9.11, the result is better. This experiment demonstrate that MEME becomes inefficient if the sequences are long.

LocalMotif Results (-250 to +50)				MEME Results (-250 to +50)		
Rank	Motif	Score	Position	Rank	Motif	Score
1	TTCAGTTC	581.53	[245,255] → Initiator	1	GCTCACACT	5.0e-369 → new motif
2	CTATAAAA	322.39	[215,225] → TATA box	2	CTCTCTC	1.7e-203
3	GGACGTGT	216.07	[275,285] → DPE	3	CGCCGCC	1.1e-151
4	GCTCACAC	203.52	[180,275] → new motif	4	TTTTTTT	1.5e-155
5	TATCGATA	124.87	[140,215] → DRE	5	TATCGATA	4.4e-78 → DRE
6	ATATATAT	108.63	[0,190]	6	CAGCCTG	1.5e-80
7	CTCGAGGT	79.67	[235,245]	7	GGCAACGC	1.4e-55
8	CGACGGGT	77.00	[265,275]	8	GTGTGTGT	6.4e-96
9	AGAGAGCG	70.86	[50,195]	9	TGCTTTTG	1.2e-39
10	CACACACA	66.38	[95,190]	10	GCGCTTAC	9.5e-24

Figure 9.11: Results on Drosophila data

MEME Results (-250 to +50)			MEME Results (-60 to +40)		
Rank	Motif	Score	Rank	Motif	Score
1	GCTCACACT	5.0e-369 → new motif	1	GCTCACACT	5.1e-415 → new motif
2	CTCTCTC	1.7e-203	2	TATCGATA	1.7e-183 → DRE
3	CGCCGCC	1.1e-151	3	TATAAA	2.1e-138 → TATA box
4	TTTTTTT	1.5e-155	4	TCAGTT	3.4e-117 → Initiator
5	TATCGATA	4.4e-78 → DRE	5	CAGCCTG	2.9e-93
6	CAGCCTG	1.5e-80	6	GTATTTT	1.9e-62
7	GGCAACGC	1.4e-55	7	CACTCT	1.9e-63
8	GTGTGTGT	6.4e-96	8	GGCAACGC	5.1e-29
9	TGCTTTTG	1.2e-39	9	GCGTGGG	1.9e-12 → DPE
10	GCGCTTAC	9.5e-24	10	CGAACGGAACG	8.3e-9

Figure 9.12: MEME results on Drosophila data

Experiments also were done on human estrogen receptor (ER) element data. Up to now, 57 ER target sequences from human chr.21 and 22 are discovered by ChIP analysis. Almost all sequences lie distal from the TSS beyond the promoter region. A motif “Forkhead” was discovered around the ER sites. 34 sequences had a full ER site. The dataset used in the experiments are these 34 sequences with full ER along with 300 bp surrounding region. The following figure (Figure 9.13) shows the data, where the diamond data points indicate the binding sites of the “Forkhead” motif, and the rectangle data points represent ER sites, which is the central vertical line. We can easily see that most “Forkhead” motifs appear around the ER site.

Three motif finders are compared in the experiments: MEME, Weeder and LocalMotif. The following figure (9.14) shows the motifs predicted. The ER are found by all of the three motif finders. MEME and LocalMotif also detect a new motif *CTTCCCTC*. The motif FH, can be only detected by LocalMotif. Moreover, LocalMotif not only gives the motifs but also gives the positions of the motifs predicted.

Figure 9.15 shows the time complexity. We can see as the sequence length grows, the MEME becomes much more expensive. Weeder is the fastest one. LocalMotif are between them. In total, LocalMotif can handle long sequences. It can analyze sequences of any length without weakening the motif. Moreover, LocalMotif not only identify the motif, it also identifies the location of the motifs.

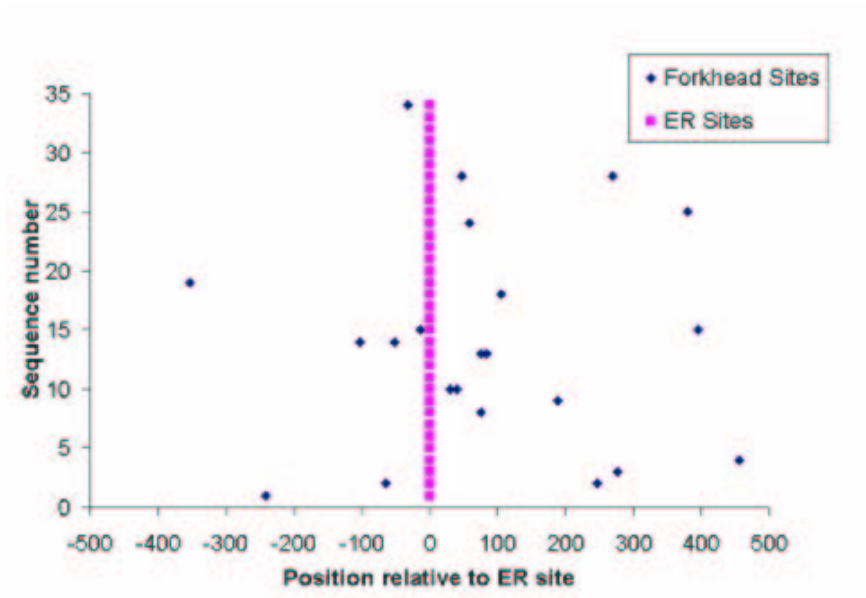


Figure 9.13: “Forkhead” positions related to ER site

<i>Software</i>	<i>Motifs Predicted</i>
MEME	TCAAGGTCAG/CTG GACCT TGA →ER AGAGGGAAGA/TCT CCCTCT →new
Weeder	GTT GACTTTG /CAAAGTCAAC →ER
LocalMotif	GGTCACCC TG /CAGGGT GACC [-20,+30] →ER AAGAAAAAAA/ TTTTTTCTT [-100,+300] →FH GGGAGGGAAG/ CTCCTCCC [-190,+190] →new

Figure 9.14: Motifs predicted in human ER data

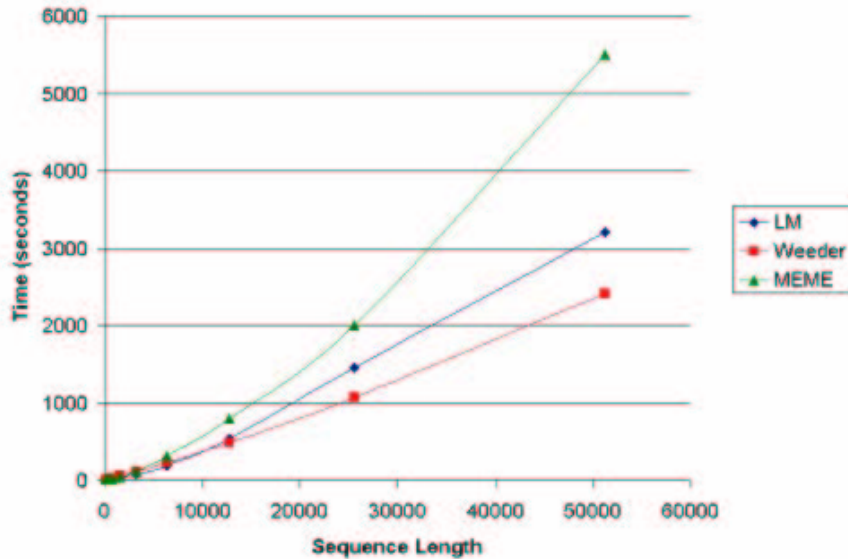


Figure 9.15: Motifs predicted in human ER data

It is also able to removes the redundant motifs.

9.3 Finding Motifs with Gaps

Most previous work usually does not allow gap in a motif. Some algorithms such as MITRA, allow only one gap. In real life, motif contains gaps. For example, ECB motif is $TTACCnAATnGGTAA$, where n is a wild card, i.e. can be any one of $ACGT$. Gal4p motif is $CGGn^{11}CCG$, where between the third character G and the last third character C there is a gap of length 11. In this section, we would like to study the possibility of improving motif finding by allowing arbitrary number of gaps. Hence, we redefine the word “motif”:

Definition 9.4 Define a motif M as a length- l string using characters $\{A, C, G, T, *\}$ with at least c non- N characters which are not wild cards. $*$ is a wild card, can be any one of $ACGT$.

For example, when $l = 20$ and $c = \frac{1}{2}$, $M = ACGTC**GACGT*CAGTTCA$ is a valid motif.

Definition 9.5 Define an instance of a motif M to be a length- l DNA sequence I , if every segment share (ls, d) -substrings.

For example, assume $(ls, d) = (5, 1)$. The intuitive meaning is that we allow at most 1 mismatch for all substrings of length 5. For example consider the

following two instances of I and I' of M . I' is not an instance of M since the segment $CACAT$ have 2 mismatches comparing with M . On the other hand, I is an instance of M since any length-5 segments of I have and only have 1 mismatch with M . Note that $*$ can match any single character.

M	=	A	C	G	T	C	*	*	G	A	C	G	T	*	C	A	G	T	T	C	A
I	=	A	T	G	T	C	c	G	G	A	C	A	T	t	T	A	G	T	T	T	A
I'	=	A	T	G	T	C	c	G	C	A	C	A	T	t	C	A	G	T	T	T	A

The motif finding problem is defined as follows:

Definition 9.6 Define motif finding problem to be: Given a set S of t sequences, find all motif M such that M has at least q instances.

For example, for the following 4 sequences, assume $l = 20, c = 20, (ls, d) = (5, 1)$, and $q = 3, M = ACGTC**GACGT* CAGTTCA$ is a valid motif.

```

TTCAACGTCACGACGTTCAGTTCAGCTCAGCTGAACGTCGGTCA
CGACATTCCACGGCTATGTCCGGACATACAGTGCACCGTACGCT
CGGCAATTACTCGTCAGTTCACGTCGCGACGACCAGTCCATCC
ACTGCGCATGAACACTTCCTGAAGTGAAGTTCACTAAAGTTCGG

```

SPACE

We propose to find gapped motif using the SPACE algorithm, which consists of 4 steps:

Step I: Input sequences S:

```

TTGATACCGAAGATACCGATTAGAAATCACTCA
ACTACAGAAAAGCAGTAGTAAACTGTACAGTC
GAAGACCGTCATGAGAAATCGCATAACACGAGCA
TTCACCCGATAAAAATAAGGCTGTCTGGACTAA
TCGGAACAATTACGAAGAAAAGCAGTAGAAAA

```

Step II: Find motif instance: Consider any length l substring in S , says, GAA-GATACCGATTAGAAATC in S_1 , position 9

G	A	A	G	A	T	A	C	C	G	A	T	T	A	G	A	A	A	T	C
G	A	A	A	A								T	A	A	A	A			
G	A	A	A	A	G	C	A	G	T	A	G	T	A	A	A	A	C	T	G

TTGATACCGAAGATACCGATTAGAAATCACTCA
ACTACAGAAAAGCAGTAGTAAACTGTACAGTC
GAAGACCGTCATGAGAAATCGCATAACACGAGCA
TTCACCCGATAAAAAATAAGGCTGTCTGGACTAA
TCGGAACAATTACGAAGAAAAGCAGTAGAAAA

- Look for some length- l substring in S such that the sharing (ls, d) -substrings cover at least c bases. For example, if $l = 20$, $ls = 5$, $d = 1$, and $c = 10$, then the following instances are found:

Go back to S , we mark the instances by underlines:

Step III: Extracting sharing substrings Compare each instance with M , record the starting position of the matching segments and we get a set of lists, whose elements are the starting positions of (ls, d) substrings.

For the above example, we have: where position 1 is the starting position

G	A	A	A	A								T	A	A	A	A			
G	A	A	A	A	G	C	A	G	T	A	G	T	A	A	A	A	C	T	G

of the first (ls, d) segment $GAAAA$, and position 13 is the second (ls, d) segment $TAAAA$.

where position 1 is the starting position of the first (ls, d) segment $GAAGA$, position 11 is the second (ls, d) segment $ATGAG$, position 12 is the third (ls, d) segment $TGAGA$ and so on. Note that segments can be overlapped.

where position 3 is the starting position of the first (ls, d) segment $CGATA$, position 4 is the second (ls, d) segment $GATAA$ which overlaps with the first segment, and position 12 is the second (ls, d) segment $ATAAG$.

where position 1 is the starting position of the first (ls, d) segment $GAAGA$, position 2 is the second (ls, d) segment $AAGAC$ and so on.

Step IV: Mining the frequent pattern We construct a frequent tree to do frequent close pattern mining. The following figure (Figure 9.16) shows the mining on the above lists. The tree records frequencies for all elements

G	A	A	G	A						A	T	G	A	G	A	A	A	T	C
G	A	A	G	A	C	C	G	T	C	A	T	G	A	G	A	A	A	T	C

		C	G	A	T	A	A			A	T	A	A	G					
C	C	C	G	A	T	A	A	A	A	A	T	A	A	G	G	C	T	G	T

appearing in all lists. Starting from the root, the first level contains all elements since each element appears at least once. The second level elements are expanded from the first level from the elements whose frequencies are larger than 1. Similarly, the third level is expanded from the elements in second level whose frequencies are larger than 2, and so on. The elements in the lower level are elements appear in the same lists as their parent. For the above example, the tree is as Figure 9.16 below. Starting from the root, the first level contains all elements in all lists. Element 1 and 13 appear more than one list; hence we expand the second level from them. In the second level, the element 13 appearing more than twice in all lists; hence we expand the tree from element 13 and get the third level. In the third level, there are no elements appearing more than three times. Hence the expanding stops and the tree construction completes.

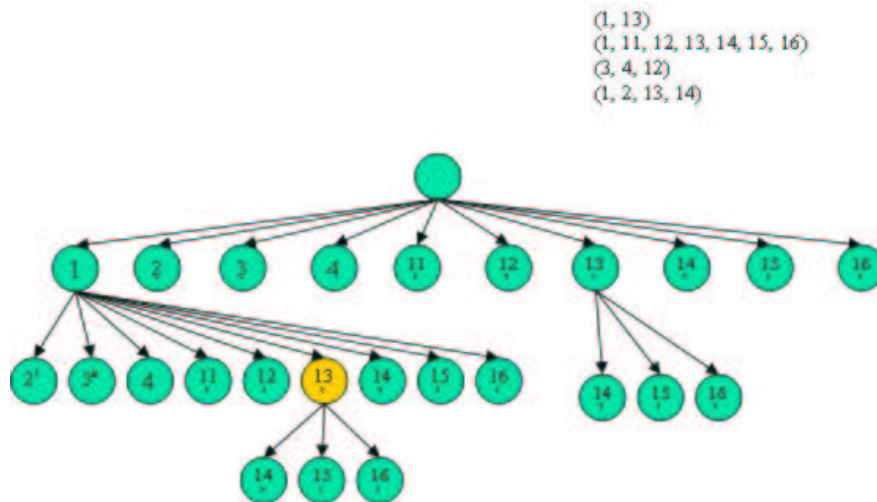


Figure 9.16: Frequent close pattern mining

Given q in is 3, which means we want the element with frequency at least 3. We find which node still has children in third level of the tree, and get 13. Then we track back and obtain the result (1, 13). Hence, we get the motif with gaps $GAAGAnnnnnnTAGAA$ since $ls = 5$, which means the

G	A	A	G	A	C							T	A	G	A	A	A		
G	A	A	G	A	C	C	A	G	C	A	G	T	A	G	A	A	A	A	A

5 characters start from position 1 and 13 from M are remained, and others are wild cards. then $M = GAAGAnnnnnnnTAGAA$, is scored using a weeder-like score.

SPEED UP

The algorithm stated above is very slow because finding motif instances requires finding all possible substrings of S and computing the mismatches. Hence, it needs to be improved.

If we shift the window which contains current substring right one position (Figure 9.17), only the first ls -mer and the last ls -mer are affected. Hence we do not need to consider all substrings in the middle part. This accelerates the algorithm. For example, suppose we know the coverage between P and I :

P	=	G	A	A	G	A	T	A	C	C	G	A	T	T	A	G	A	A	A	T	C
		G	A	A	G	A						A	T	G	A	G	A	A	A	T	C
I	=	G	A	A	G	A	C	C	G	T	C	A	T	G	A	G	A	A	A	T	C

The P and I have coverage 15. They have (ls, d) list $(1, 11, 12, 13, 14, 15, 16)$. We can find coverage between P' and I' where P' is one symbol shifting right from P and I' is one symbol shifting right from I :

P'	=	A	A	G	A	T	A	C	C	G	A	T	T	A	G	A	A	A	T	C	G
											A	T	G	A	G	A	A	A	T	C	T
I'	=	A	A	G	A	C	C	G	T	C	A	T	G	A	G	A	A	A	T	C	T

Note that the middle part is not affected. The new stuff only comes from both ends.

Now P' and I' have coverage 11. Their (ls, d) list is $(10, 11, 12, 13, 14, 15, 16)$.

Thus, by shifting one symbol by one symbol, we can find all coverage of the segments on the whole sequence. (Figure 9.17)

EXPERIMENTAL RESULT

The experiment is run on Motif Assessment Benchmark Datasets [Tompa05]. The dataset consists of 56 datasets from 4 different species, namely, fly, human, mouse and yeast. Each dataset contains 1 to 35 sequences. Each sequence length is up to 3K bp.

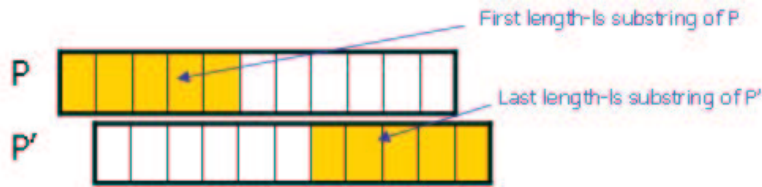


Figure 9.17: Window shifting: Only the filling areas are affected.

The algorithm is performed with 12 runs of all combinations of these parameters:

- $l = 8, 15, 20$ (max motif length)
- $c = 0.5l, 0.8l$ (coverage)
- $q = t, 0.5t$ (min support)
- $ls = 5$ (substring length)
- $d = 1$ (mismatches)

The evaluations of the results follow the following evaluation measures:

- $S_n(\text{Recall}) = \frac{TP}{TP+FN}$ (Sensitivity)
- $PPV(\text{Precision}) = \frac{TP}{TP+FP}$ (Correctness)
- $PC(\text{PerformanceCoefficient}) = \frac{TP}{TP+FN+FP}$
- $CC(\text{CorrelativeCoefficient}) = \frac{TP \times TN - FN \times FP}{\sqrt{(TP+FN)(TN+FP)(TP+FP)(TN+FN)}}$

where T means true, F means false, P means positive and N means negative.

The following figures (Figure 9.18,9.19,9.20,9.21) show the result comparing with various motif finders. (Our motif finder is called SPACE) In Figure 9.18, we see that SPACE has highest accuracy by giving the highest PPV. Its PPV is much higher than other motif finders. Moreover, its sensitivity is the best as it has the highest S_n value. The improvement shows that SPACE improves both the S_n and PPV by 36 percents and 11 percents respectively.

Figure 9.19 shows the results on 4 different species, namely, fly, human, mouse and yeast. We illustrate the result by a group of 4 bars for each species, which are the highest S_n value over all motif finders, the S_n value of SPACE, the highest PPV value over all motif finders, and the PPV value of SPACE, from left to right. We can see that over all 4 species, SPACE beats the best motif finders by giving both higher S_n value and PPV value. For human, it improves most, since human genome is very complex and contains many gaps.

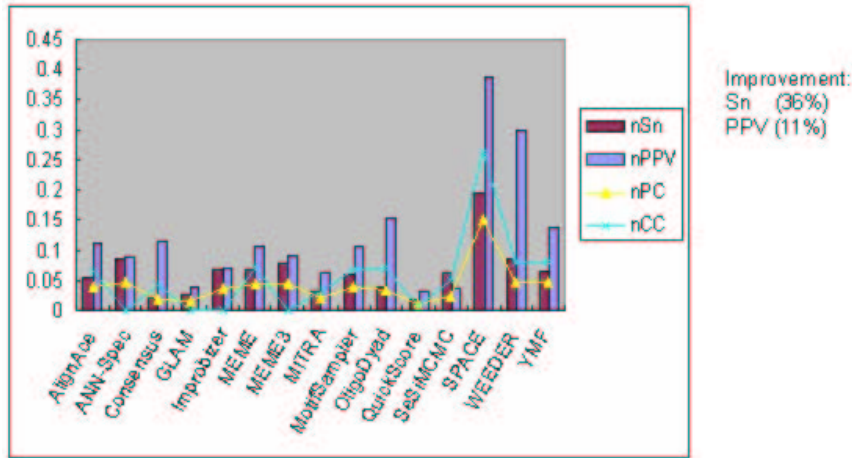


Figure 9.18: Experimental results - Benchmark Dataset

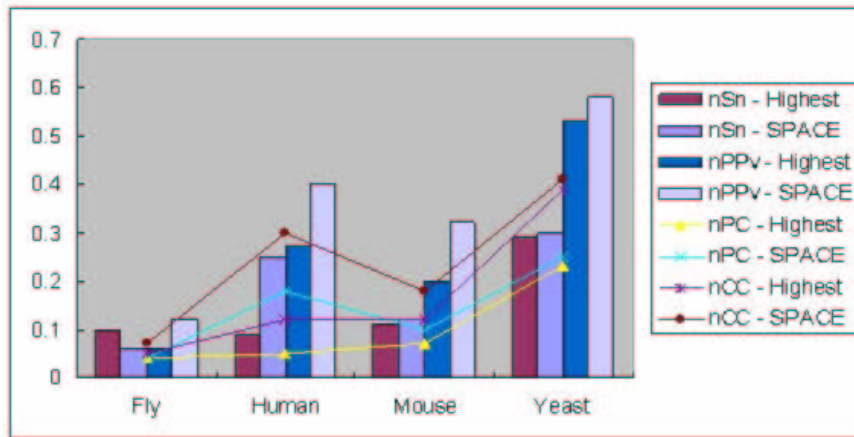


Figure 9.19: Comparison on four organisms



Figure 9.20: Example on hm22m

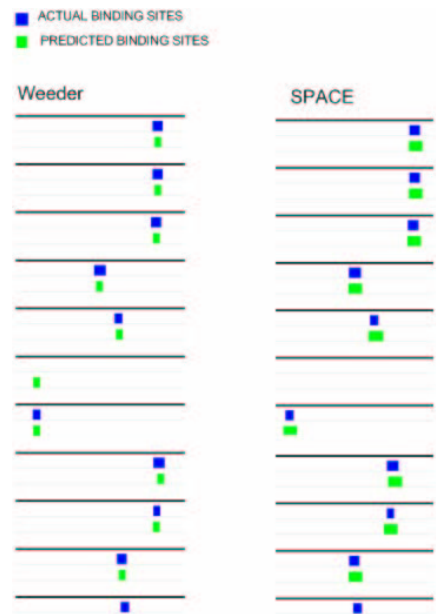


Figure 9.21: Example on hm17g

References

- [1] V. Narang, W. Sung and A. Mittal *LocalMotif - An In-Silico Approach To Detecting Localized Motifs in Regulatory Sequences*. ICTAI, 2006.
- [2] M. Tompa, N. Li, T. Bailey, G. Church, B. Moor, E. Eskin, A. Favorov, M. Frith, Y. Fu, W. Kent, V. Makeev, A. Mironov, W. Noble, G Pavesi, G. Pesole, M. Regnier, N. Simonis, S. Sinha, G. Thijs, J. Helden, M. Vandenbergert, Z. Weng, C. Workman, C. Ye and Z. Zhu *Assesing computational tools for the discovery of transcription factor binding sites*. Nature Biotechnology, 23:137-144, 2005.