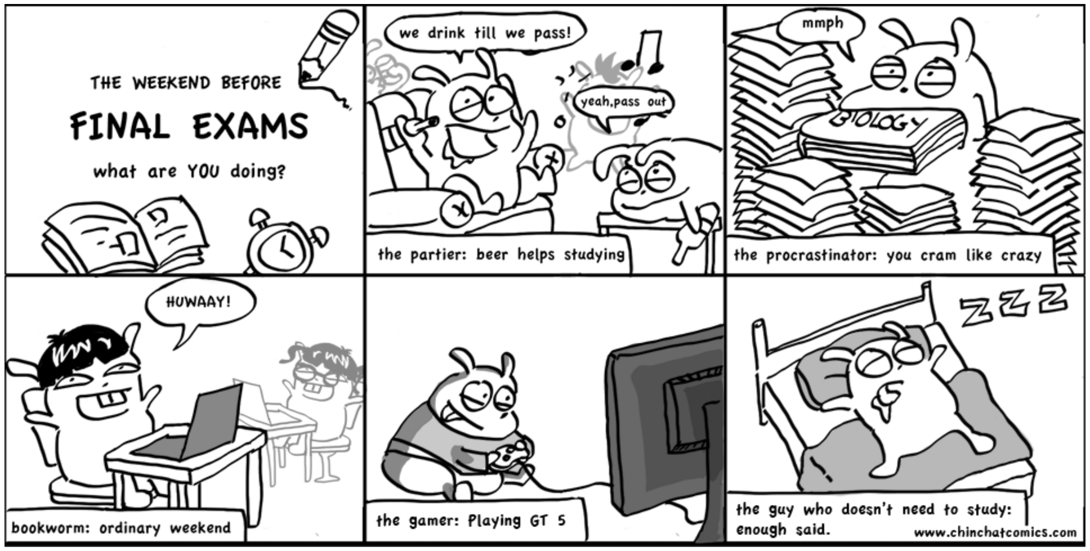
Randomized Algorithms



Motif finding using a Profile

- Profile is generated by *some* consensus
- Use to find the best match of motif in each sequence
- These matches suggest a new consensus

A	1/2	7/8	3/8	0	1/8	0
C	1/8	0	1/2	5/8	3/8	0
T	1/8	1/8	0	0	1/4	7/8
G	1/4	0	1/8	3/8	1/4	1/8

```
ctataaacgttacatc
atagcgattcgactga
cagcccagaaccctgg
cggtgaaccttacatc
tgcattcaatagctta
tgtcctgtccactcac
ctccaaatcctttaca
ggtctacctttatcct
```

1	a	a	a	c	g	t
2	a	t	a	g	c	g
3	a	a	c	c	c	t
4	g	a	a	c	c	t
5	a	t	a	g	c	t
6	g	a	c	c	t	g
7	a	t	c	c	t	t
8	t	a	c	c	t	t
A	5/8	5/8	4/8	0	0	0
C	0	0	4/8	6/8	4/8	0
T	1/8	3/8	0	0	3/8	6/8
G	2/8	0	0	2/8	1/8	2/8

Reds are probabilities that increase, and Blues decrease.

GreedyProfileMotifSearch Algorithm

```python
import random

def GreedyProfileMotifSearch(DNA, k):
    s = [-1 for i in xrange(len(DNA))]
    newS = [random.randint(0,len(DNA[i])-k) for i in xrange(len(DNA))]
    while newS != s:
        s = [i for i in newS]
        P = Profile(DNA, s, k)
        newS = Score(DNA, P)
    return newS
```

Profile Code

```
def Profile(DNA, offset, k):
    profile = []
    t = len(DNA)
    for i in xrange(k):
        counts = {base : 0.01 for base in "ACGT"}
        for j in xrange(t):
            counts[DNA[j][offset[j]+i]] += 0.96 / t
        profile.append(counts)
    return profile
```

Score Code

```python
from operator import mul

def Score(DNA, P):
    offset = []
    k = len(P)
    for j in xrange(len(DNA)):
        pBest, iBest = 0.0, -1
        for i in xrange(len(DNA[j])-k+1):
            p = reduce(mul, [P[l][DNA[j][i+l]] for l in xrange(k)], 1.0)
            if (p > pBest):
                pBest, iBest = p, i
        offset.append(iBest)
    return offset
```

Example Profile for [0,0,0,0,0,0,0,0]

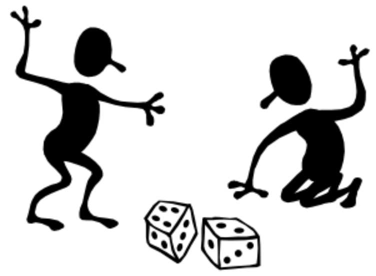
```
DNA = ["CTATAAACGTTACATC",
       "ATAGCGATTCGACTGA",
       "CAGCCCAGAACCCTGG",
       "CGGTGAACCTTACATC",
       "TGCATTCAATAGCTTA",
       "TGTCCTGTCCACTCAC",
       "CTCCAAATCCTTTACA",
       "GGTCTACCTTTATCCT"]

{'A': 0.13, 'C': 0.49, 'T': 0.25, 'G': 0.13},
{'A': 0.13, 'C': 0.01, 'T': 0.37, 'G': 0.49},
{'A': 0.25, 'C': 0.25, 'T': 0.25, 'G': 0.25},
{'A': 0.13, 'C': 0.49, 'T': 0.25, 'G': 0.13},
{'A': 0.25, 'C': 0.37, 'T': 0.25, 'G': 0.13},
{'A': 0.49, 'C': 0.13, 'T': 0.25, 'G': 0.13}
```

Testing GreedyProfileMotifSearch

```
# Try running it a few times

DNA = ["CTATAAACGTTACATC",
       "ATAGCGATTCGACTGA",
       "CAGCCCAGAACCCTGG",
       "CGGTGAACCTTACATC",
       "TGCATTCAATAGCTTA",
       "TGTCCTGTCCACTCAC",
       "CTCCAAATCCTTTACA",
       "GGTCTACCTTTATCCT"]

k = 6
offsets = GreedyProfileMotifSearch(DNA, k)

P = Profile(DNA, offsets, k)
print offsets
print ''.join([b for p, b in [max([(v, b) for b, v in row.iteritems()]) for row in P]]),
print reduce(mul, [p for p, b in [max([(v, b) for b, v in row.iteritems()]) for row in P]], 1.0)
for i, j in enumerate(offsets):
    print DNA[i][:j].lower()+DNA[i][j:j+k]+DNA[i][j+k:].lower()
```

```
[1, 6, 3, 10, 4, 7, 1, 8]
TTCAAA 0.025390493905
cTATAAAcgttacatc
atagcgATTCGActga
cagCCCAGAaccctgg
cggtgaacctTACATC
tgcaTTCAATagctta
tgtcctgTCCACTcac
cTCCAAAtcctttaca
ggtctaccTTTATCct
```

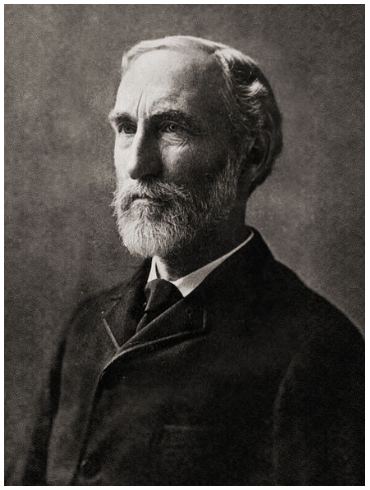
GreedyProfileMotifSearch( ) Analysis



- Since we choose starting positions randomly, there is little chance that our guess will be close to an optimal motif, meaning it will take a very long time to find the optimal motif.
- It is unlikely that the random starting positions will lead us to the correct solution at all.
- In practice, such an algorithm would be run many times with the hope that *some* random starting positions will be close to the optimum solution simply by chance.

Gibbs Sampling

- GreedyProfileMotifSearch is probably not the best way to find motifs.
- However, we can improve the algorithm by introducing Gibbs Sampling, an iterative procedure that discards one k-mer after each iteration and replaces it with a totally new one.
- Gibbs Sampling proceeds more slowly and chooses new k-mers at random increasing the odds that it will converge to the correct solution.


Josiah W Gibbs

How Gibbs Sampling Works

1. Randomly choose starting positions $\bar{s} = (s_1, \ldots, s_t)$ and form the set of k-mers associated with these starting positions.
2. Randomly choose one of the t sequences.
3. Create a profile P from the other t -1 sequences.
4. For each position in the removed sequence, calculate the probability that the k-mer starting at that position was generated by P.
5. Choose a new starting position for the removed sequence at random based on the probabilities calculated in step 4.
6. Repeat steps 2-5 until there is no improvement

Gibbs Sampling: an Example

Input: $t = 5$ sequences, motif length, $l = 8$

```
1.  GTAAACAATATTTATAGC
2.  AAAATTTACCTCGCAAGG
3.  CCGTACTGTCAAGCGTGG
4.  TGAGTAAACGACGTCCCA
5.  TACTTAACACCCTGTCAA
```

Gibbs Sampling: an Example

1) Randomly choose starting positions, $\overline{s} = (s_1, s_2, s_3, s_4, s_5)$ in the 5 sequences:

$s_1=6$	GTAAACAATATTTATAGC
$s_2=10$	AAAATTTACCTTAGAAGG
$s_3=8$	CCGTACTGTCAAGCGTGG
$s_4=3$	TGAGTAAACGACGTCCCA
$s_5=0$	TACTTAACACCCTGTCAA

Gibbs Sampling: an Example

2) Choose one of the sequences at random: ex. Sequence 2

```
s1=6     GTAAACAATATTTATAGC
s2=10    AAAATTTACCTTAGAAGG
s3=8     CCGTACTGTCAAGCGTGG
s4=3     TGAGTAAACGACGTCCCA
s5=0     TACTTAACACCCTGTCAA
```

Gibbs Sampling: an Example

3) Remove it and create a profile from the remaining sequences

```
s1=6    GTAAACAATATTTATAGC

s3=8    CCGTACTGTCAAGCGTGG
s4=3    TGAGTAAACGACGTCCCA
s5=0    TACTTAACACCCTGTCAA
```

1	A	A	T	A	T	T	T	A
3	T	C	A	A	G	C	G	T
4	G	T	A	A	A	C	G	A
5	T	A	C	T	T	A	A	C
A	1/4	2/4	2/4	3/4	1/4	1/4	1/4	2/4
C	0	1/4	1/4	0	0	2/4	0	1/4
T	2/4	1/4	1/4	1/4	2/4	1/4	1/4	1/4
G	1/4	0	0	0	1/4	0	3/4	0
Consensus String	T	A	A	A	T	C	G	A

Profile Matrix for sequences 1,3,4, and 5

Gibbs Sampling: an Example

4) Calculate the $prob(a|P)$ for every possible k-mer in the removed sequence:

k-mer highligted in red	p
AAAATTTACCTTAGAAGG	.000732
AAAATTTACCTTAGAAGG	.000122
AAAATTTACCTTAGAAGG	0
AAAATTTACCTTAGAAGG	0
AAAATTTACCTTAGAAGG	0
AAAATTTACCTTAGAAGG	0
AAAATTTACCTTAGAAGG	0
AAAATTTACCTTAGAAGG	.000183
AAAATTTACCTTAGAAGG	0
AAAATTTACCTTAGAAGG	0
AAAATTTACCTTAGAAGG	0

1	A	A	T	A	T	T	T	A
3	T	C	A	A	G	C	G	T
4	G	T	A	A	A	C	G	A
5	T	A	C	T	T	A	A	C
A	1/4	2/4	2/4	3/4	1/4	1/4	1/4	2/4
C	0	1/4	1/4	0	0	2/4	0	1/4
T	2/4	1/4	1/4	1/4	2/4	1/4	1/4	1/4
G	1/4	0	0	0	1/4	0	3/4	0
Consensus String	T	A	A	A	T	C	G	A

Profile Matrix for sequences 1,3,4, and 5

Gibbs Sampling: an Example

5) Create a distribution of probabilities of k-mers $prob(a|P)$, and randomly select a new starting position based on this distribution.

To create this distribution, divide each probability $prob(a|P)$ by the total of all probabilities:

- Starting Position 1:
  $prob(AAAATTTA|P) = .000732/(.000732 + .000122 + .000183) = .706$
- Starting Position 2:
  $prob(AAATTTAC|P) = .000122/(.000732 + .000122 + .000183) = .118$
- Starting Position 8:
  $prob(ACCTTAGA|P) = .000183/(.000732 + .000122 + .000183) = .176$

Gibbs Sampling: an Example

```python
import random

def sample(cdf):
    t = random.random()
    for i in xrange(len(cdf)):
        if (t < cdf[i]):
            break
    return i

p = [0.000732, 0.000122, 0.0, 0.0, 0.0, 0.0, 0.0, 0.000183, 0.0, 0.0, 0.0]
pdf = [v/sum(p) for v in p]
cdf = [sum(pdf[:i]) for i in xrange(1,len(pdf))]
```

Gibbs Sampling: an Example

Assume we select the substring with the highest probability – then we are left with the following new substrings and starting positions.

$s_1=6$	GTAAACAATATTTATAGC
$s_2=0$	AAAATTTACCTTAGAAGG
$s_3=8$	CCGTACTGTCAAGCGTGG
$s_4=3$	TGAGTAAACGACGTCCCA
$s_5=0$	TACTTAACACCCTGTCAA

6) We then repeat the procedure with the above starting positions until we cannot improve the score any more.

Gibbs Sampler in Practice

- Gibbs sampling needs to be modified when applied to samples with biased distributions of nucleotides (relative entropy approach).
- Gibbs sampling often converges to a locally optimal motif rather than to the globally optimal motif.
- Should be run with many randomly chosen seeds to achieve good results.

Another Randomized Approach

- A *Random Projection Algorithm* is a different way to solve the Motif Finding Problem.
- Guiding principle: Instances of a motif agree at a subset of positions.
- However, it is unclear how to find these “non-mutated” positions.
- To bypass the effect of mutations within a motif, we randomly select a subset of positions in the pattern creating a projection of the pattern.
- Search for that projection in a hope that the selected positions are not affected by mutations in most instances of the motif.

Projections

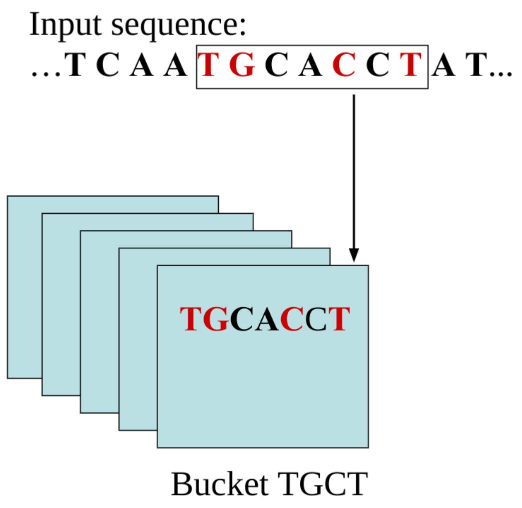
- Choose k positions in string of length l.
- Concatenate nucleotides at chosen k positions to form k-tuple.
- This can be viewed as a projection of l-dimensional space onto k-dimensional subspace.

Projection = (1, 3, 4, 6, 10, 11, 12)

aTgGCaTtcaGATtc → TGCTGAT

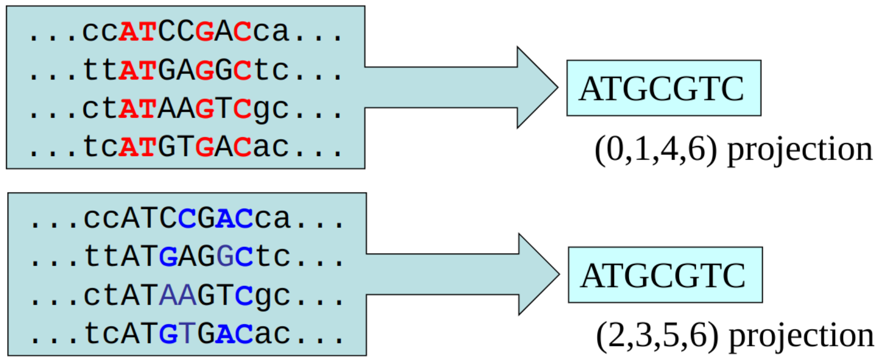
Random Projections Algorithm

- Select k out of l positions uniformly at random.
- For each l-tuple in input sequences, hash into buckets based on the k selected positions.
- Recover motif from enriched buckets that contain many l-tuples with at least one from each sequence.



Random Projections Algorithm finer points

- Some projections will fail to detect motifs but if we try many of them the probability that one of the buckets fills increases.
- In the example below, the bucket --GC-AC is "bad" while the bucket AT--G-C is "good"



Combining Random Projection and Gibbs Sampling

- Random Projection is a procedure for finding good starting points: every enriched bucket is a potential starting point.
- Feeding these starting points into existing algorithms (like Gibbs sampler) provides good local search in vicinity of every starting point.
- These algorithms work particularly well for “good” starting points.

It's over

- Final Next Friday, 5/4
- 8:00am - 11:00am
- This room: SN011

```
Open book, open notes,
Will covers material since midterm
```

Study session? Monday Night?