Gibbs Sampling and Random Projection

Gibbs Sampling

- RandomProfileMotifSearch is probably not the best way to find motifs. Depends on random guesses followed by a greedy optimization procedure.
- Major cost is Scoring, P(**a** | **P**), and updating profiles
- Gibbs Sampling estimates a distribution of each variable in turn, conditional on the current values of the other variables.
- However, we can improve the algorithm by introducing **Gibbs Sampling**, an iterative procedure that discards one *k*-mer's contribution to the profile distribution at each iteration and replaces it with a new one.
- Gibbs Sampling starts out more slowly but chooses new *k*-mers with increasing the odds that it will improve the current solution.

How Gibbs Sampling Works

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

$s_1=7$	GTAAACAATATTTATAGC
$s_2=11$	AAAATTTACCTTAGAAGG
$s_3=9$	CCGTACTGTCAAGCGTGG
$s_4=4$	TGAGTAAACGACGTCCCA
$s_5=1$	TACTTAACACCCTGTCAA

Gibbs Sampling: an Example

2) Choose one of the sequences at random:

Sequence 2: AAAATTTACCTTAGAAGG

s_1=*7*	GTAAACAATATTTATAGC
s_2=*11*	AAAATTTACCTTAGAAGG
s_3=*9*	CCGTACTGTCAAGCGTGG
s_4=*4*	TGAGTAAACGACGTCCCA
s_5=*1*	TACTTAACACCCTGTCAA

Gibbs Sampling: an Example

2) Choose one of the sequences at random:

Sequence 2: AAAATTTACCTTAGAAGG

s_1=7	GTAAACAATATTTATAGC
s_3=9	CCGTACTGTCAAGCGTGG
s_4=4	TGAGTAAACGACGTCCCA
s_5=1	TACTTAACACCCTGTCAA

Gibbs Sampling: an Example

3) Create profile ***P*** from *l*-mers in remaining 4 sequences:

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

Gibbs Sampling: an Example

4) Calculate the *prob*(***a*** | ***P***) for every possible 8-mer in the removed sequence:

Strings Highlighted in Red	*prob*(**a** | **P**)
AAAATTTACCTTAGAAGG	.000732
AAAATTTACCTTAGAAGG	.000122
AAAATTTACCTTAGAAGG	0
AAAATTTACCTTAGAAGG	0
AAAATTTACCTTAGAAGG	0
AAAATTTACCTTAGAAGG	0
AAAATTTACCTTAGAAGG	0
AAAATTTACCTTAGAAGG	.000183
AAAATTTACCTTAGAAGG	0
AAAATTTACCTTAGAAGG	0
AAAATTTACCTTAGAAGG	0

Gibbs Sampling: an Example

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

A) To create this distribution, divide each probability $prob(\boldsymbol{a} \mid \boldsymbol{P})$ by the total:

Starting Position 1: *prob(* AAAATTTA | P) = .706

Starting Position 2: *prob(* AAATTTAC | P) = .118

Starting Position 8: *prob(* ACCTTAGA | P) = .176

Gibbs Sampling: an Example

B) Select a new starting position at random according to computed distribution:

P(selecting starting position 1): .706
P(selecting starting position 2): .118
P(selecting starting position 8): .176

```
t = random.random()
if (t < .706):
    # use position 1
elif (t < (.706 + .118)):
    # use position 2
else:
    # use position 8
```

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=7	GTAAACAATATTTATAGC
s_2=1	AAAATTTACCTCGCAAGG
s_3=9	CCGTACTGTCAAGCGTGG
s_4=5	TGAGTAATCGACGTCCCA
s_5=1	TACTTCACACCCTGTCAA

Gibbs Sampling: an Example

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

Modified Profile function:

```
def Profile(seqList, k, start):
    dist = [dict([(base,0.1) for base in "acgt"]) for i in xrange(k)]
    for t in xrange(len(seqList)):
        if (start[t] < 0):
            continue
        for i, base in enumerate(seqList[t][start[t]:start[t]+k]):
            dist[i][base] += 1.0
    for i in xrange(k):
        total = sum(dist[i].values())
        for base in "acgt":
            dist[i][base] /= total
    return dist
```

Gibbs Sampling in Python

```
def GibbsProfileMotifSearch(seqList, k):
    start = [random.randint(0,len(seqList[t])-k+1) for t in xrange(len(seqList))]
    bestScore = 0.0
    noImprovement = 0
    while True:
        remove = random.randint(0,len(seqList)-1)
        start[remove] = -1
        distr = Profile(seqList, k, start)
        score = 0.0
        for t in xrange(len(seqList)):
            if (start[t] < 0):
                rScore = 0.0
                for i in xrange(len(seqList[remove])-k+1):
                    score = Score(seqList[remove], i, k, distr)
                    if (score > rScore):
                        rStart, rScore = i, score
                score += rScore
                start[t] = rStart
            else:
                score += Score(seqList[t], start[t], k, distr)
        if (score > bestScore):
            bestScore = score
            noImprovement = 0
        else:
            noImprovement += 1
            if (noImprovement > len(seqList)):
                break
    return score, start
```

Gibbs Sampling Performance

```
%timeit s, m = FindMotif(seqApprox, 10, 100, RandomProfileMotifSearch)
print "Random", s
for i, si in enumerate(m):
    print si, seqApprox[i][si:si+10]

print
%timeit s, m = FindMotif(seqApprox, 10, 100, GibbsProfileMotifSearch)
print "Gibbs", s
for i, si in enumerate(m):
    print si, seqApprox[i][si:si+10]
```

```
1 loops, best of 3: 473 ms per loop
Random ttgacctgat
17 tagatctgaa
47 tggatccgaa
18 tagacccgaa
33 taaatccgaa
21 taggtccaaa
0 tagattcgaa
46 cagatccgaa
70 tagatccgta
16 tagatccaaa
65 tcgatccgaa

10 loops, best of 3: 66.3 ms per loop
Gibbs ttgacctgat
17 tagatctgaa
47 tggatccgaa
18 tagacccgaa
33 taaatccgaa
21 taggtccaaa
0 tagattcgaa
46 cagatccgaa
70 tagatccgta
16 tagatccaaa
65 tcgatccgaa
```

Gibbs Sampler in Practice

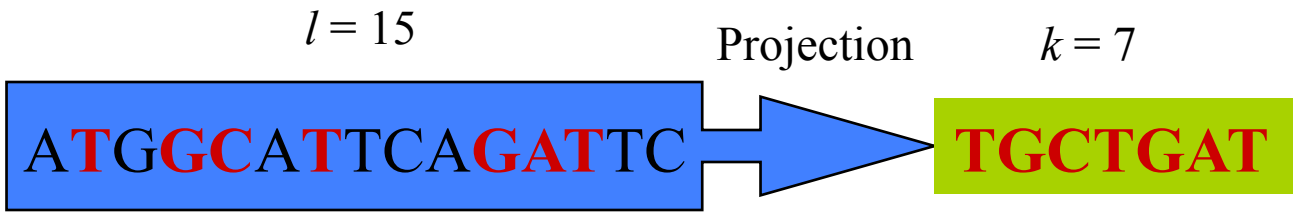
- Fewer profile searches, *O(n)*, in exchange for updating the profile, *O(kt)*, more often (tradeoff which is easier)
- Gibbs sampling can converge much faster than a fully randomized approach
- Gibbs sampling is more likely to converge to locally optimal motifs rather a fully randomized algorithm.
- Like the fully Randomized Algorithm it must be run with many randomly chosen initial seeds to achieve good results.

Another Randomized Approach

- **Random Projection Algorithm** is a different way to solve the Motif Finding Problem.
- **Guiding principle:** Instances of a good motif will likely agree at a subset of positions.
- However, it is unclear how to find these matching, "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

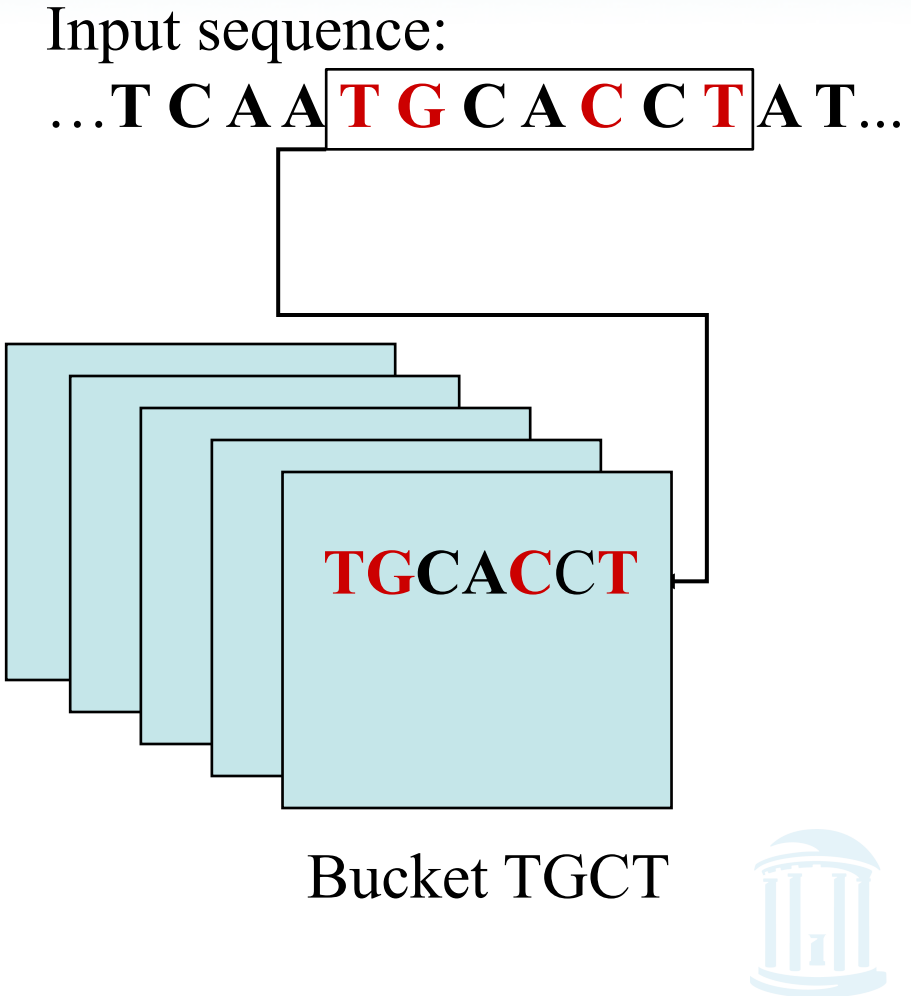
- Choose k positions in string of length l.
- Concatenate nucleotides at chosen k positions to form k-tuple.
- This can be viewed as a projection of l-dimensional space onto k-dimensional subspace.



Projection = (2, 4, 5, 7, 11, 12, 13)

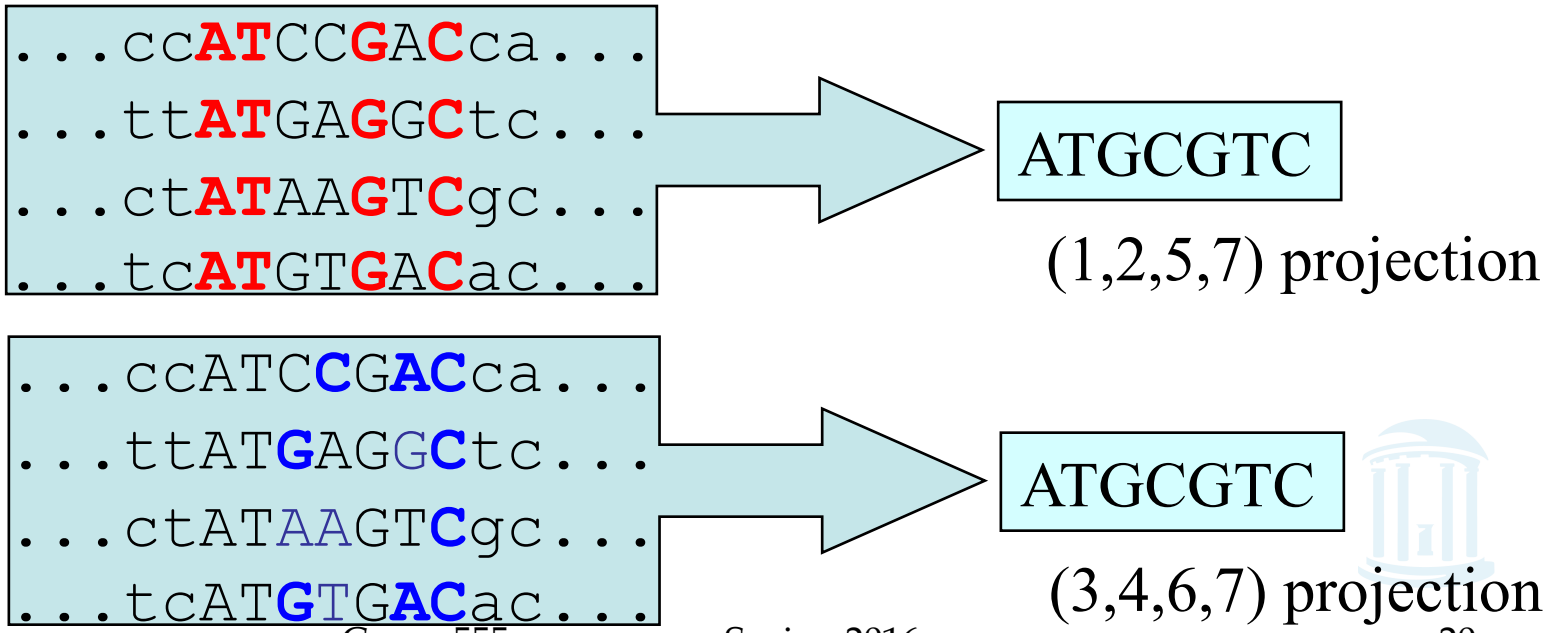
Random Projections Algorithm

- 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.



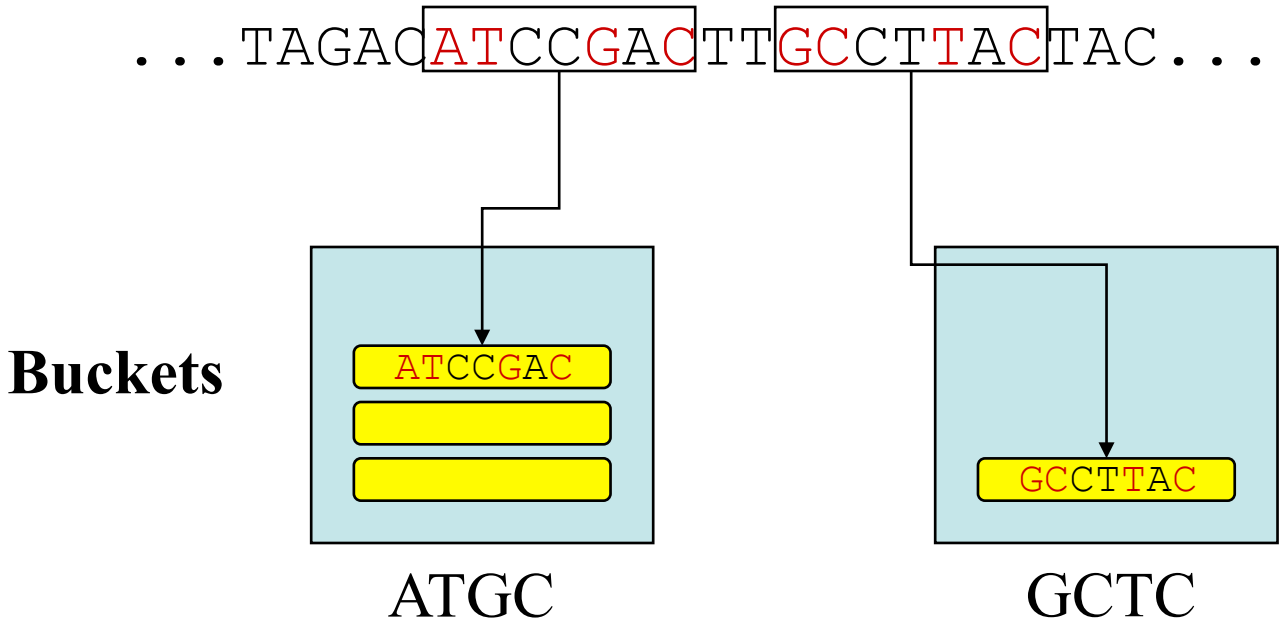
Random Projections Algorithm (cont'd)

- 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"



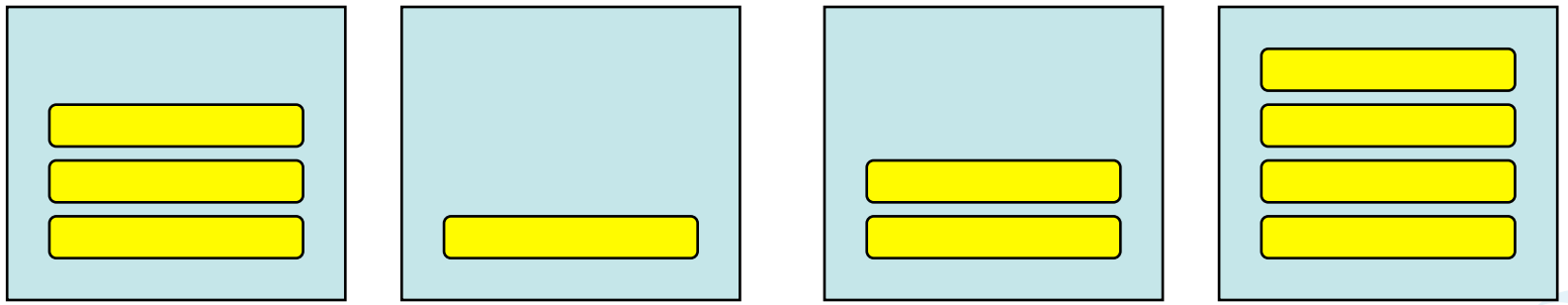
Example

- l = 7 (motif size) , k = 4 (projection size)
- Choose projection (1,2,5,7)



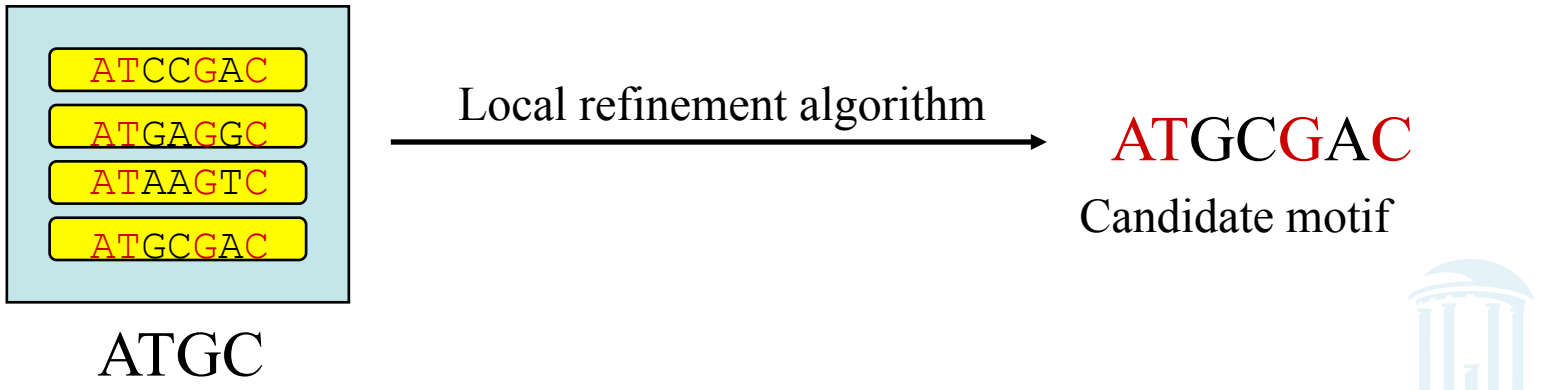
Hashing and Buckets

- Hash function *h(x)* obtained from *k* positions of projection.
- Buckets are labeled by values of *h(x)*.
- *Enriched buckets*: contain more than *s* *l*-tuples, for some parameter *s* with representatives from all sequences



Motif Refinement

- How do we recover the motif from the sequences in enriched buckets?
- *k* nucleotides are exact matches, (hash key of bucket).
- Use information in other *l*-*k* positions as starting point for local refinement scheme, e.g. Gibbs sampler.



Synergy of Random Projection

- 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, or fully Randomized Search) provides good local search in vicinity of every starting point.
- These algorithms work particularly well for "good" starting points.

Random Projection

```
def RandomProjectionMotifStart(seqList, k):
    N = len(seqList)
    matches = random.randint(k/3,(3*k)/4)
    positions = sorted(random.sample(range(k), matches))
    hash = {}
    for t in xrange(N):
        for i in xrange(len(seqList[t])-k+1):
            pattern = ''.join([seqList[t][i+j] for j in positions])
            plist = hash.get(pattern,[[] for j in xrange(N)])
            plist[t] += [i]
            hash[pattern] = plist
    result = []
    for key in hash.iterkeys():
        start = []
        skipped = 0
        for posList in hash[key]:
            if (len(posList) == 0):
                skipped += 1
                if (skipped > N/3):
                    break
                start.append(-1)
            else:
                start.append(posList[0])
        else:
            result.append(start)
    return result
```

Good Projections

```
seqApprox = [
    'tagtggtcttttgagtgtagatctgaagggaaagtatttccaccagttcggggtcacccagcagggcagggtgacttaat',
    'cgcgactcggcgctcacagttatcgcacgtttagaccaaaacggagttggatccgaaactggagtttaatcggagtcctt',
    'gttacttgtgagcctggttagacccgaaatataattgttggctgcatagcggagctgacatacgagtaggggaaatgcgt',
    'aacatcaggctttgattaaacaatttaagcacgtaaatccgaattgacctgatgacaatacggaacatgccggctccggg',
    'accaccggataggctgcttattaggtccaaaaggtagtatcgtaataatggctcagccatgtcaatgtgcggcattccac',
    'tagattcgaatcgatcgtgtttctccctctgtgggttaacgaggggtccgaccttgctcgcatgtgccgaacttgtaccc',
    'gaaatggttcggtgcgatatcaggccgttctcttaacttggcggtgcagatccgaacgtctctggaggggtcgtgcgcta',
    'atgtatactagacattctaacgctcgcttattggcggagaccatttgctccactacaagaggctactgtgtagatccgta',
    'ttcttacacccttctttagatccaaacctgttggcgccatcttcttttcgagtccttgtacctccatttgctctgatgac',
    'ctacctatgtaaaacaacatctactaacgtagtccggtctttcctgatctgccctaacctacaggtcgatccgaaattcg']
```

```
?at?c?aa??  [-1, 49, -1, 35, -1, 2, 48, 12, 18, 67]
at?c?aa???  [-1, 50, -1, 36, -1, 3, 49, 13, 19, 68]
gat???aa??  [19, 49, -1, 50, -1, 2, 48, -1, 18, 67]
?g??cc?aa?  [-1, 48, 19, -1, 22, 62, 47, -1, 17, 66]
??gat???aa  [17, 47, -1, 48, -1, 0, 46, -1, 16, 65]
t?g?tc?g??  [17, 47, -1, -1, 48, -1, 4, 70, 67, 65]
?t?g?tc?g?  [16, 46, -1, -1, 47, -1, 3, 69, 66, 64]
?g??cc?aa?  [-1, 48, 19, -1, 22, 62, 47, -1, 17, 66]
g??cc?aa??  [-1, 49, 20, -1, 23, 63, 48, -1, 18, 67]
atc??aa???  [20, 50, -1, 36, 38, -1, 49, -1, 19, 68]
```

Random Projection Performance

```
def RandomProjectionMotif(seqList,k):
    startList = []
    while len(startList) < 10:
        startList += RandomProjectionMotifStart(seqList, k)
    highScore = 0.0
    for start in startList:
        score, start = ProfileMotifSearch(seqList, k, start)
        if score > highScore:
            motif = [s for s in start]
            highScore = score
    return highScore, motif

%timeit s,m = RandomProjectionMotif(seqApprox,10)
print "RandomProjection", s
for i, si in enumerate(m):
    print si, seqApprox[i][si:si+10]
```

```
10 loops, best of 3: 65.1 ms per loop
RandomProjection ttgacctgat
17 tagatctgaa
47 tggatccgaa
18 tagacccgaa
33 taaatccgaa
21 taggtccaaa
0 tagattcgaa
46 cagatccgaa
70 tagatccgta
16 tagatccaaa
65 tcgatccgaa
```

Random Projection Synopsis

- Fewer starts than other randomized algorithms (10x fewer)
- Each projection is *O(tn)*, which is the same as the full scan of in the fully Randomized Profile scan
- Generates good starts but requires either Gibbs sampling or Randomized search to refine the final solution

It's Over

- Final Monday, 5/2
  - 12:00-3:00PM
  - This room: FB007
  - Open book, open notes, open internet, online
  - Will cover material since midterm
  - Final Study session:
    - Friday 4/29, FB007 5pm-7pm