



# Lecture 24: Randomized Algorithms

Chapter 12

Study Session tomorrow

Virtual 4<sup>th</sup> problem set,  
Return PS#1, PS#2

# Randomized Algorithms



- Randomized algorithms incorporate random, rather than deterministic, decisions
- Commonly used in situations where no exact and/or fast algorithm is known
- Main advantage is that no input can reliably produce worst-case results because the algorithm runs differently each time.



# Select



- **Select(L, k)** finds the  $k^{\text{th}}$  smallest element in L
- **Select(L, 1)** find the smallest...
  - Well known  $O(n)$  algorithm

```
minv = HUGE
for v in L:
    if (v < minv):
        minv = v
```

- **Select(L, len(L)/2)** find the median...
  - How?
  - median = sorted(L)[len(L)/2]  $\rightarrow O(n \log n)$
- Can we find medians, or 1<sup>st</sup> quartiles in  $O(n)$ ?



# Select Recursion



- **Select(L, k)** finds the  $k^{\text{th}}$  smallest element in **L**
  - Select an element  $m$  from unsorted list **L** and partition **L** the array into two smaller lists:

$L_{lo}$  - elements smaller than  $m$

and

$L_{hi}$  - elements larger than  $m$ .

- If  $\text{len}(L_{lo}) > k$  then  
    Select( $L_{lo}$ ,  $k$ )
- else if  $k > \text{len}(L_{lo}) + 1$  then  
    Select( $L_{hi}$ ,  $k - \text{len}(L_{lo}) - 1$ )
- else  $m$  is the  $k^{\text{th}}$  smallest element



# Example of Select(L, 5)



Given an array:  $L = \{ 6, 3, 2, 8, 4, 5, 1, 7, 0, 9 \}$

**Step 1**: Choose the first element as  $m$

$L = \{ 6, 3, 2, 8, 4, 5, 1, 7, 0, 9 \}$



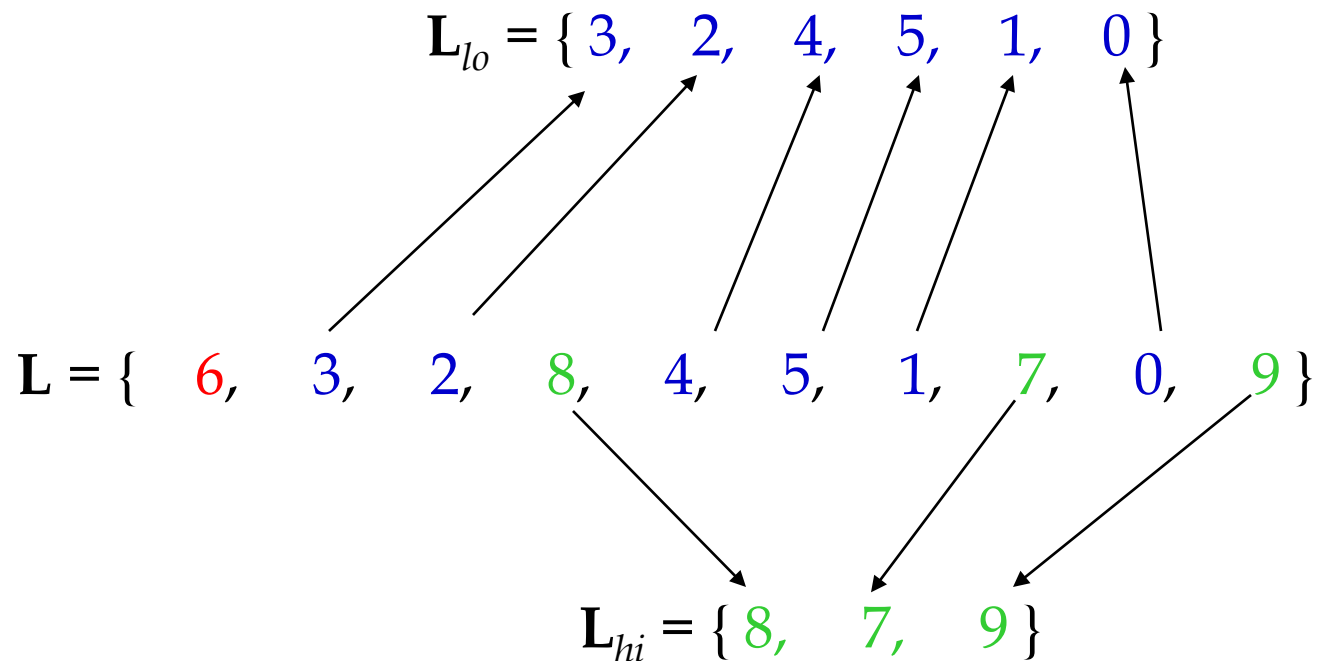
Our Selection



# Example of Select(cont'd)



**Step 2:** Split the array into  $L_{lo}$  and  $L_{hi}$

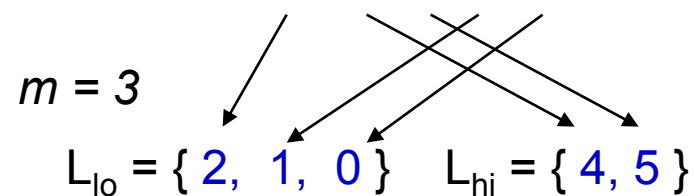


# Example of Select(cont'd)

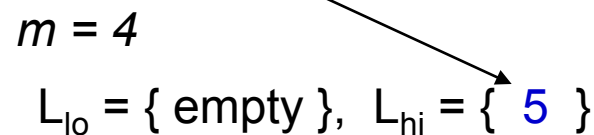


**Step 3:** Recursively call Select on either  $L_{lo}$  or  $L_{hi}$  until  $\text{len}(L_{lo}) = k$ , then return  $m$ .

$\text{len}(L_{lo}) > k = 5 \rightarrow \text{Select}(\{ \textcolor{red}{3}, \textcolor{blue}{2}, \textcolor{blue}{4}, \textcolor{blue}{5}, \textcolor{blue}{1}, \textcolor{blue}{0} \}, 5)$



$k = 5 > \text{len}(L_{lo}) + 1 \rightarrow \text{Select}(\{ \textcolor{red}{4}, \textcolor{blue}{5} \}, 5 - 3 - 1)$



$k = 1 == \text{len}(L_{lo}) + 1 \rightarrow \text{return } 4$



# Select Code



```
def select(L, k):
    value = L[0]
    Llo = [t for t in data if t < value]
    Lhi = [t for t in data if t > value]
    below = len(Llo) + 1
    if (k < len(Llo)):
        return select(Llo, k)
    elif (k > below):
        return select(Lhi, k - below)
    else:
        return value
```





# Select Analysis with Good Splits



- Runtime depends on our selection of  $m$ :
  - A good selection will split  $L$  evenly such that

$$|L_{lo}| = |L_{hi}| = |L|/2$$

- The recurrence relation is:
$$T(n) = T(n/2)$$

$$- n + n/2 + n/4 + n/8 + n/16 + \dots = 2n \rightarrow O(n)$$



# Select Analysis with Bad Splits



However, a poor selection will split  $L$  unevenly and in the worst case, all elements will be greater or less than  $m$  so that one Sublist is full and the other is empty.

For a poor selection, the recurrence relation is

$$T(n) = T(n-1)$$

In this case, the runtime is  $O(n^2)$ .



Our dilemma:

$O(n)$  or  $O(n^2)$ ,

depending on the list... or  $O(n \log n)$  independent of it



# Select Analysis (cont'd)



- Select seems risky compared to sort
- To improve Select, we need to choose  $m$  to give good 'splits'
- It can be proven that to achieve  $O(n)$  running time, we don't need a perfect splits, just reasonably good ones.
- In fact, if both subarrays are at least of size  $n/4$ , then running time will be  $O(n)$ .
- This implies that half of the choices of  $m$  make good splitters.



# A Randomized Approach



- To improve Select, *randomly* select  $m$ .
- Since half of the elements will be good splitters, if we choose  $m$  at random we will get a 50% chance that  $m$  will be a good choice.
- This approach will make sure that no matter what input is received, the expected running time is small.



# Randomized Select



```
def randomizedSelect(L, k):  
    value = random.choice(L)  
    Llo = [t for t in data if t < value]  
    Lhi = [t for t in data if t > value]  
    below = len(Llo) + 1  
    if (k < len(Llo)):  
        return randomizedSelect(Llo, k)  
    elif (k > below):  
        return randomizedSelect(Lhi, k-below)  
    else:  
        return value
```



# RandomizedSelect Analysis



- Worst case runtime:  $O(n^2)$
- *Expected runtime*:  $O(n)$ .
- Expected runtime is a good measure of the performance of randomized algorithms, often more informative than worst case runtimes.
- Worst case runtimes are rarely repeated
- RandomizedSelect always returns the correct answer, which offers a way to classify Randomized Algorithms.



## Two Types of Randomized Algorithms



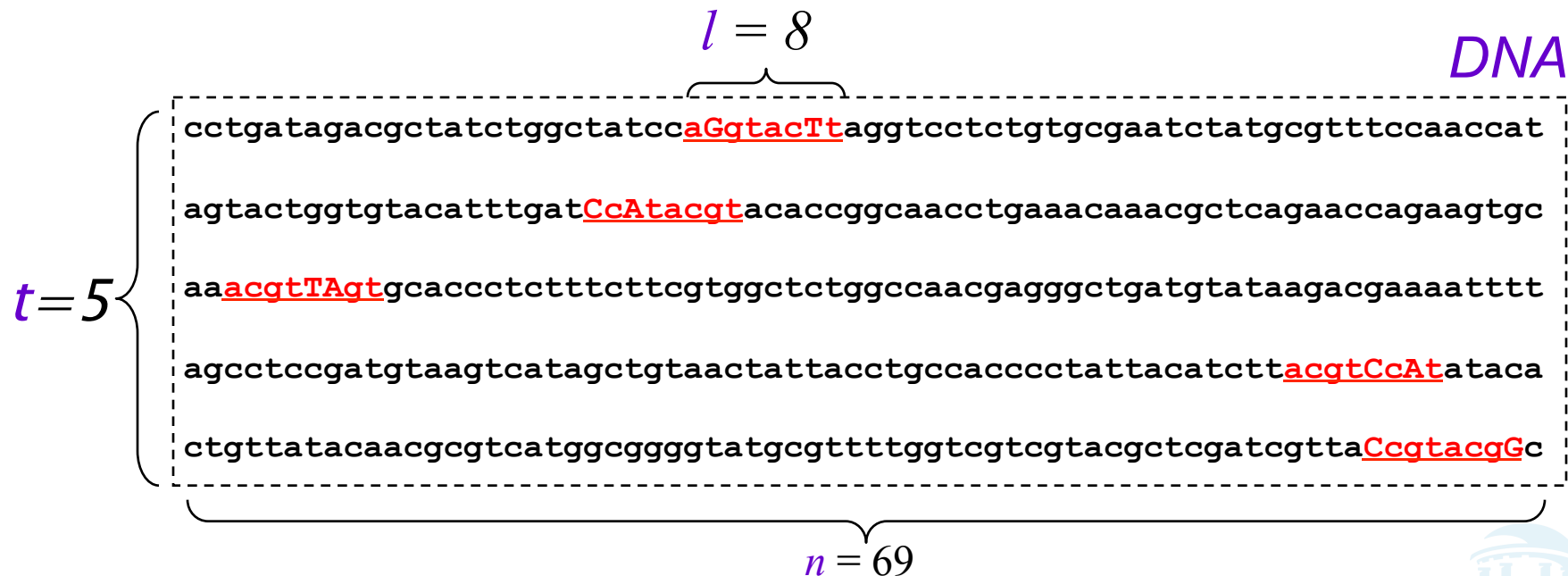
- **Las Vegas Algorithms** – always produce the correct solution (i.e. randomizedSelect)
- **Monte Carlo Algorithms** – do not always return the correct solution.
- Las Vegas Algorithms are always preferred, but they are often hard to come by.



# The Motif Finding Problem



**Motif Finding Problem:** Given a list of  $t$  sequences each of length  $n$ , find the “best” pattern of length  $l$  that appears in each of the  $t$  sequences.





# A New Motif Finding Approach



- **Motif Finding Problem:** Given a list of  $t$  sequences each of length  $n$ , find the “best” pattern of length  $l$  that appears in each of the  $t$  sequences.
- **Previously:** we solved the Motif Finding Problem using a Branch and Bound or a Greedy technique.
- **Now:** **randomly** select possible locations and find a way to greedily change those locations until we converge to the hidden motif.



# Profiles Revisited



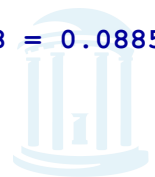
- Let  $s = (s_1, \dots, s_t)$  be the starting positions for  $l$ -mers in our  $t$  sequences.
- The substrings corresponding to these starting positions will form:

- $t \times l$  alignment matrix
- $4 \times l$  profile matrix\*

$l$										
a	G	g	t	a	c	T	t	}	$t$	
C	c	A	t	a	c	g	t			
a	c	g	t	T	A	g	t			
a	c	g	t	C	c	A	t			
C	c	g	t	a	c	g	G			
<hr/>										
A	0.6	0.0	0.2	0.0	0.6	0.2	0.2	0.0	}	4
C	0.4	0.8	0.0	0.0	0.2	0.8	0.0	0.0		
G	0.0	0.2	0.8	0.0	0.0	0.0	0.6	0.2		
T	0.0	0.0	0.0	1.0	0.2	0.0	0.2	0.8		
<hr/>										
x	a	c	g	t	a	c	g	t		

$$P(X|\text{profile}) = 0.6 * 0.8 * 0.8 * 1.0 * 0.6 * 0.8 * 0.6 * 0.8 = 0.0885$$

\* Note that we now define the profile matrix in terms of frequency, not counts as in Lecture 5.



# Scoring Strings with a Profile



- Let  $l$ -mer  $\mathbf{a} = a_1, a_2, a_3, \dots, a_l$
- $P(\mathbf{a} | \mathbf{P})$  is defined as the probability that an  $l$ -mer  $\mathbf{a}$  was created by the Profile  $\mathbf{P}$ .
- If  $\mathbf{a}$  is very similar to the consensus string of  $\mathbf{P}$  then  $P(\mathbf{a} | \mathbf{P})$  will be high
- If  $\mathbf{a}$  is very different, then  $P(\mathbf{a} | \mathbf{P})$  will be low.

$$Prob(\mathbf{a} | \mathbf{P}) = \prod_{i=1}^l p(a_i, i)$$



# Scoring Strings with a Profile (cont'd)



Given a profile:  $\mathbf{P} =$

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

The probability of the consensus string:

$$Prob(\mathbf{aaacct}|\mathbf{P}) = ???$$



# Scoring Strings with a Profile (cont'd)



Given a profile:  $\mathbf{P} =$

A	$\frac{1}{2}$	$\frac{7}{8}$	$\frac{3}{8}$	0	$\frac{1}{8}$	0
C	$\frac{1}{8}$	0	$\frac{1}{2}$	$\frac{5}{8}$	$\frac{3}{8}$	0
T	$\frac{1}{8}$	$\frac{1}{8}$	0	0	$\frac{1}{4}$	$\frac{7}{8}$
G	$\frac{1}{4}$	0	$\frac{1}{8}$	$\frac{3}{8}$	$\frac{1}{4}$	$\frac{1}{8}$

The probability of the consensus string:

$$Prob(\mathbf{aaacct}|\mathbf{P}) = \frac{1}{2} \times \frac{7}{8} \times \frac{3}{8} \times \frac{5}{8} \times \frac{3}{8} \times \frac{7}{8} = .033646$$



# Scoring Strings with a Profile (cont'd)



Given a profile:  $\mathbf{P} =$

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

The probability of the consensus string:

$$Prob(\mathbf{aaacct}|\mathbf{P}) = 1/2 \times 7/8 \times 3/8 \times 5/8 \times 3/8 \times 7/8 = .033646$$

Probability of a different string:

$$Prob(\mathbf{atacag}|\mathbf{P}) = 1/2 \times 1/8 \times 3/8 \times 5/8 \times 1/8 \times 1/8 = .001602$$



# P-Most Probable $l$ -mer



- Define the **P**-most probable  $l$ -mer from a sequence as an  $l$ -mer in that sequence which has the highest probability of being created from the profile **P**.

**P** =

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

Given a sequence = ctataaaccttacatc, find the P-most probable  $l$ -mer



# P-Most Probable $l$ -mer (cont'd)



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

Find the  $Prob(\mathbf{a}|\mathbf{P})$  of every possible 6-mer:

First try: **c t a t a a a c c t t a c a t c**

Second try: **c t a t a a a c c t t a c a t c**

Third try: **c t a t a a a c c t t a c a t c**

-Continue this process to evaluate every possible 6-mer





# P-Most Probable $l$ -mer (cont'd)



Compute  $prob(\mathbf{a}|\mathbf{P})$  for every possible 6-mer:

String, Highlighted in Red	Calculations	$prob(\mathbf{a}   \mathbf{P})$
ctataa <b>ac</b> cttacat	$1/8 \times 1/8 \times 3/8 \times 0 \times 1/8 \times 0$	0
ctataa <b>aa</b> cttacat	$1/2 \times 7/8 \times 0 \times 0 \times 1/8 \times 0$	0
ctataa <b>ac</b> cttacat	$1/2 \times 1/8 \times 3/8 \times 0 \times 1/8 \times 0$	0
ctata <b>aa</b> ccttacat	$1/8 \times 7/8 \times 3/8 \times 0 \times 3/8 \times 0$	0
ctata <b>aa</b> ccttacat	$1/2 \times 7/8 \times 3/8 \times 5/8 \times 3/8 \times 7/8$	.0336
ctataa <b>ac</b> cttacat	$1/2 \times 7/8 \times 1/2 \times 5/8 \times 1/4 \times 7/8$	.0299
ctataa <b>ac</b> cttacat	$1/2 \times 0 \times 1/2 \times 0 \times 1/4 \times 0$	0
ctataaa <b>c</b> cttacat	$1/8 \times 0 \times 0 \times 0 \times 0 \times 1/8 \times 0$	0
ctataaa <b>c</b> cttacat	$1/8 \times 1/8 \times 0 \times 0 \times 3/8 \times 0$	0
ctataaa <b>c</b> cttacat	$1/8 \times 1/8 \times 3/8 \times 5/8 \times 1/8 \times 7/8$	.0004

# P-Most Probable $l$ -mer (cont'd)



P-Most Probable 6-mer in the sequence is aaacct:

String, Highlighted in Red	Calculations	$Prob(a   P)$
ctataa <b>ac</b> cttacat	$1/8 \times 1/8 \times 3/8 \times 0 \times 1/8 \times 0$	0
ctataa <b>aa</b> cttacat	$1/2 \times 7/8 \times 0 \times 0 \times 1/8 \times 0$	0
ctataa <b>ac</b> cttacat	$1/2 \times 1/8 \times 3/8 \times 0 \times 1/8 \times 0$	0
ctata <b>aa</b> cccttacat	$1/8 \times 7/8 \times 3/8 \times 0 \times 3/8 \times 0$	0
ctata <b>aa</b> cc <b>tt</b> acat	$1/2 \times 7/8 \times 3/8 \times 5/8 \times 3/8 \times 7/8$	.0336
ctataa <b>ac</b> cttacat	$1/2 \times 7/8 \times 1/2 \times 5/8 \times 1/4 \times 7/8$	.0299
ctataa <b>ac</b> cttacat	$1/2 \times 0 \times 1/2 \times 0 \times 1/4 \times 0$	0
ctataaa <b>cc</b> ttacat	$1/8 \times 0 \times 0 \times 0 \times 0 \times 1/8 \times 0$	0
ctataaa <b>c</b> cttacat	$1/8 \times 1/8 \times 0 \times 0 \times 3/8 \times 0$	0
ctataaa <b>ac</b> cttacat	$1/8 \times 1/8 \times 3/8 \times 5/8 \times 1/8 \times 7/8$	.0004

## P-Most Probable $l$ -mer (cont'd)



**aaacct** is the **P**-most probable 6-mer in:

ctat**aaacct**tacatc

because  $Prob(\mathbf{aaacct}|\mathbf{P}) = .0336$  is greater than the  $Prob(\mathbf{a}|\mathbf{P})$  of any other 6-mer in the sequence.



# Dealing with Zeroes



- In our toy example  $\text{prob}(\mathbf{a} \mid \mathbf{P})=0$  in many cases. In practice, there will be enough sequences so that the number of elements in the profile with a frequency of zero is small.
- To avoid many entries with  $\text{prob}(\mathbf{a} \mid \mathbf{P})=0$ , there exist techniques to equate zero to a very small number so that one zero does not make the entire probability of a string zero (assigning a *prior* probability, we will not address these techniques here).



# P-Most Probable $l$ -mers in Many Sequences



- Find the **P**-most probable  $l$ -mer in each of the “t” sequences.

**P**=

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

atagcgattcgactg

cagcccagaaccct

cggtataccttacatc

tgcatccaatagctta

tatcctttccactcac

ctccaaatcctttaca

ggtcatcctttatcct



# P-Most Probable $l$ -mers in Many Sequences (cont'd)



ctataaacgttacatc

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

atagcgattcgactg

cagcccagaaccct

cggtgaaccttacatc

tgcaattcaatagctta

tgtcctgtccactcac

ctccaaatcctttaca

ggctacctttatcct

P-Most Probable  $l$ -mers form a new profile



# Comparing New and Old Profiles



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

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

**Red** – frequency increased, **Blue** – frequency decreased



# Greedy Profile Motif Search



Use  $P$ -Most probable  $l$ -mers to adjust start positions until we reach a “best” profile; this is the motif.

- 1) Select random starting positions.
- 3) Create a profile  $\mathbf{P}$  from the substrings at these starting positions.
- 4) Find the  $\mathbf{P}$ -most probable  $l$ -mer  $\mathbf{a}$  in each sequence and change the starting position to the starting position of  $\mathbf{a}$ .
- 5) Compute a new profile based on the new starting positions after each iteration and proceed until we cannot increase the score anymore.





# GreedyProfileMotifSearch Algorithm



1. GreedyProfileMotifSearch( $DNA, t, n, l$ )
2. Randomly select starting positions  $s=(s_1, \dots, s_t)$  from  $DNA$
3.  $bestScore \leftarrow 0$
4. while  $Score(s, DNA) > bestScore$
5.     form profile  $P$  from  $s$
6.      $bestScore \leftarrow Score(s, DNA)$
7.     for  $i \leftarrow 1$  to  $t$
8.         Find a  $P$ -most probable  $l$ -mer  $a$  from the  $i^{th}$  sequence
9.          $s_i \leftarrow$  starting position of  $a$
10. return  $bestScore$



# 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, this algorithm is run many times with the hope that 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  $l$ -mer after each iteration and replaces it with a new one.
- Gibbs Sampling proceeds more slowly and chooses new  $l$ -mers at random increasing the odds that it will converge to the correct solution.



# How Gibbs Sampling Works



- 1) Randomly choose starting positions  $\mathbf{s} = (s_1, \dots, s_t)$  and form the set of  $l$ -mers associated with these starting positions.
- 2) Randomly choose one of the  $t$  sequences.
- 3) Create a profile  $\mathbf{P}$  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  $\mathbf{P}$ .
- 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,  
 $s=(s_1, s_2, s_3, s_4, s_5)$  in the 5 sequences:

$$s_1=7$$

GTAAAC**AATATTT**ATAGC

$$s_2=11$$

AAAATTTACCT**TTAGAAGG**

$$s_3=9$$

CCGTACTGT**CAAGCGT**GG

$$s_4=4$$

TGAG**GTAAACG**ACGTCCCA

$$s_5=1$$

**TACTTAAC**ACCCTGTCAA



# 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: AAAATTACCTTAGAAGG**

$s_1=7$

GTAAAC**AATATTTA**TAGC

$s_3=9$

CCGTACTGT**CAAGCGT**GG

$s_4=4$

TGAG**TAAACGA**CGTCCCA

$s_5=1$

**TACTTAAC**ACCCTGTCAA





# 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

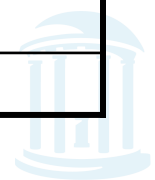


4) Calculate the  $prob(a | P)$  for every possible 8-mer in the removed sequence:

Strings Highlighted in Red

$prob(a | P)$

AAAATTACCTTAGAAGG	.000732
AAAATTACCTTAGAAGG	.000122
AAATTACCTTAGAAGG	0
AAAATTACCTTAGAAGG	0
AAAATTACCTTAGAAGG	0
AAAATTACCTTAGAAGG	0
AAAATTACCTTAGAAGG	0
AAAATTACCTTAGAAGG	0
AAAATTACCTTAGAAGG	.000183
AAAATTACCTTAGAAGG	0
AAAATTACCTTAGAAGG	0
AAAATTACCTTAGAAGG	0



# Gibbs Sampling: an Example



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

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

Starting Position 1:  $prob(\text{AAAATTTA} | P) = .706$

Starting Position 2:  $prob(\text{AAATTTAC} | P) = .118$

Starting Position 8:  $prob(\text{ACCTTAGA} | P) = .176$



# 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.



# 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 locally optimal motifs rather than globally optimal motifs.
- Must be run with many randomly chosen 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 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.

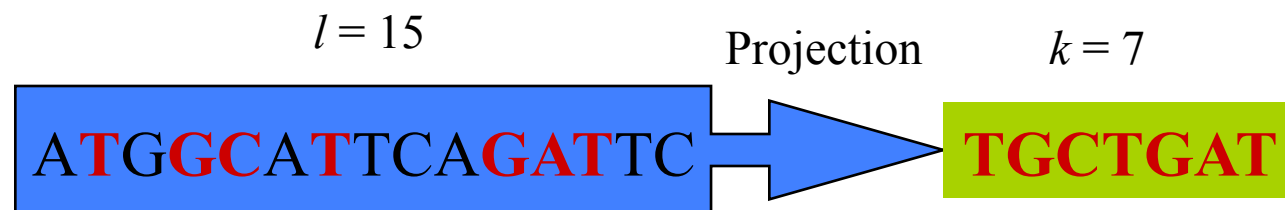




# Projections



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



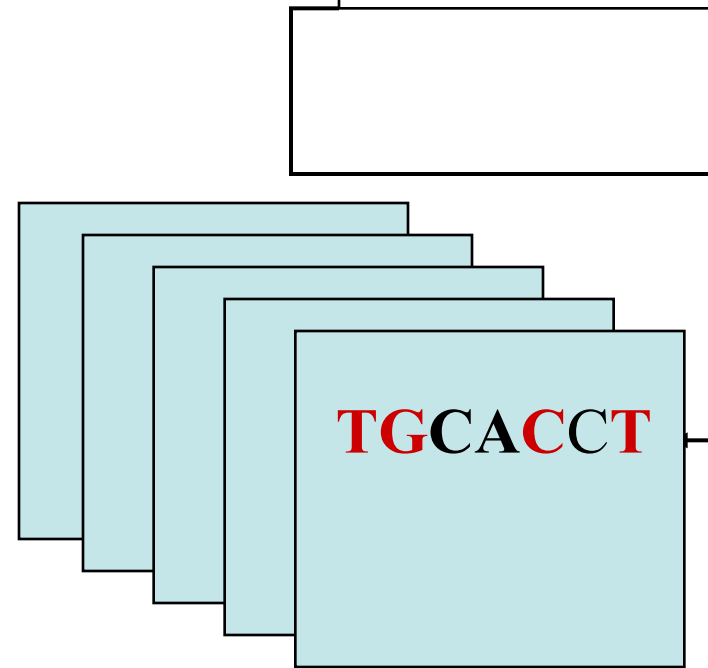
# Random Projections Algorithm



- 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.

Input sequence:

...T C A A **T G C A C C T** A T...



Bucket TGCT

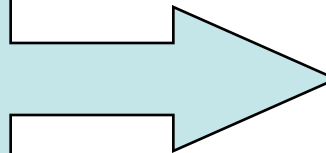


# Random Projections Algorithm (cont'd)



- 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”

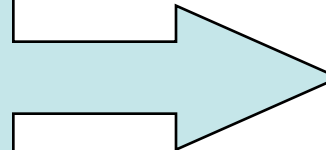
```
...ccATCCGACca...  
...ttATGAGGCtc...  
...ctATAAGTCgc...  
...tcATGTGACac...
```



ATGCGTC

(1,2,5,7) projection

```
...ccATCCGACca...  
...ttATGAGCtc...  
...ctATAAGTgc...  
...tcATGTGACac...
```



ATGCGTC

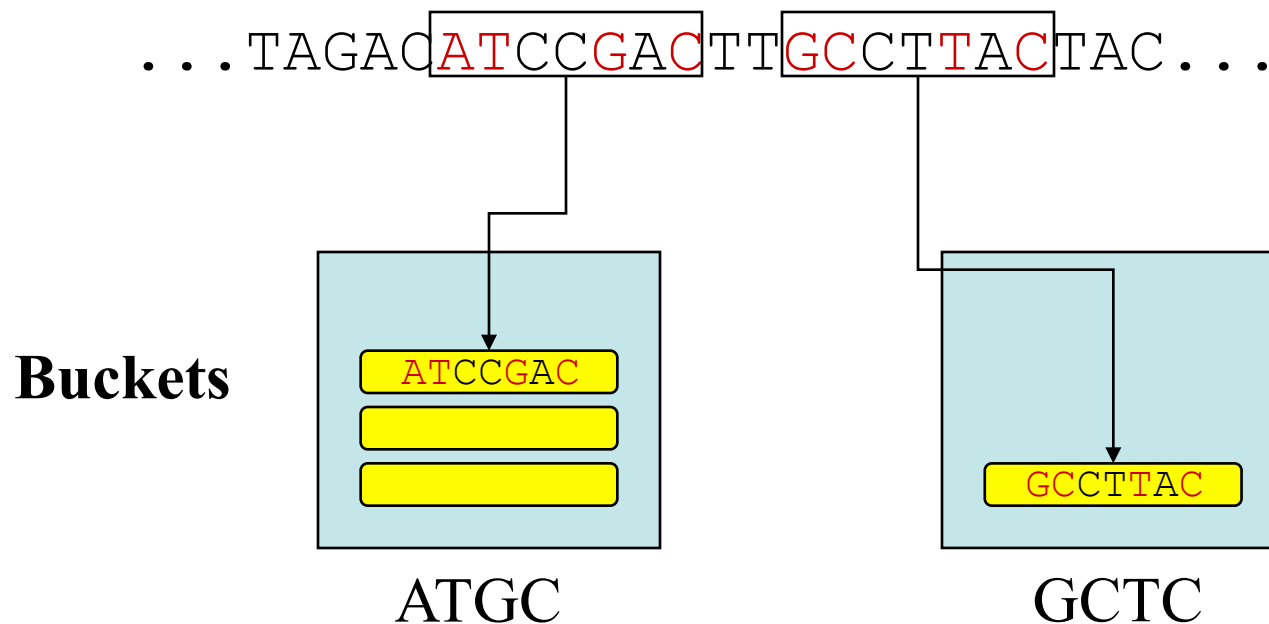
(3,4,6,7) projection



# Example



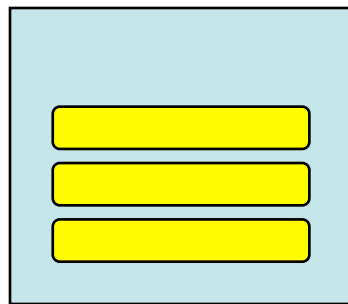
- $l = 7$  (motif size) ,  $k = 4$  (projection size)
- Choose projection (1,2,5,7)



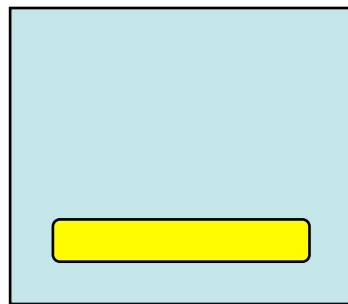
# Hashing and Buckets



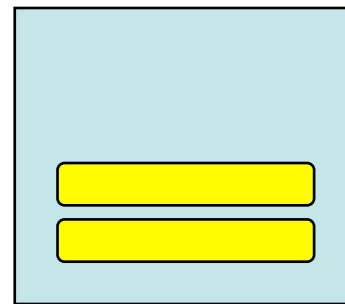
- Hash function  $h(x)$  obtained from  $k$  positions of projection.
- Buckets are labeled by values of  $h(x)$ .
- *Enriched buckets*: contain more than  $s$   $l$ -tuples, for some parameter  $s$  with representatives from all sequences



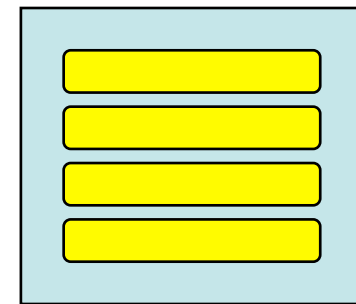
ATGC



GCTC



CATC



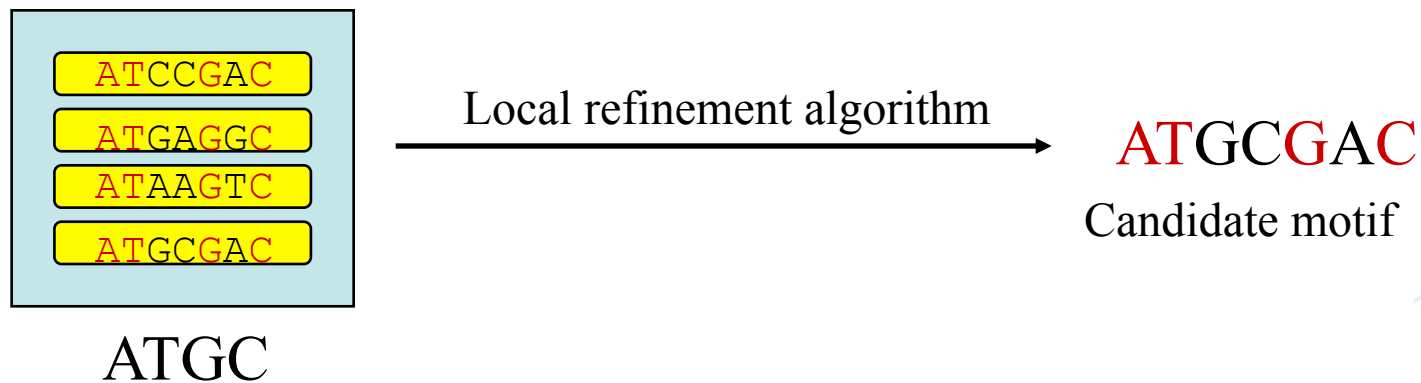
ATTC



# 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 between Random Projection and Gibbs Sampler



- 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.



# Building Profiles from Buckets



ATGC

A	1	0	.25	.50	0	.50	0
C	0	0	.25	.25	0	0	1
G	0	0	.50	0	1	.25	0
T	0	1	0	.25	0	.25	0

**Profile P**

**Gibbs sampler**

**Refined profile P\***





# Motif Refinement



- For each bucket  $h$  containing more than  $s$  sequences, form profile  $\mathbf{P}(h)$
- Use Gibbs sampler algorithm with starting point  $\mathbf{P}(h)$  to obtain refined profile  $\mathbf{P}^*$



# Random Projection Algorithm



## A Single Iteration:

- Choose a random  $k$ -projection.
- Hash each  $l$ -mer  $x$  in input sequence into bucket labeled by  $h(x)$
- From each enriched bucket (e.g., a bucket with more than  $s$  sequences), form profile  $\mathbf{P}$  and perform Gibbs sampler motif refinement
- Candidate motif is best found by selecting the best motif among refinements of all enriched buckets.



# Choosing Projection Size



- Projection size  $k$ 
  - choose  $k$  small enough so that several motif instances hash to the same bucket.
$$k \ll l, \quad l/2 < k < l - \text{const}$$
  - choose  $k$  large enough to avoid contamination by spurious  $l$ -mers:

$$4^k \gg t(n - l + 1)$$



# It's Over



- Final Thursday, 4/30
  - 12:00-3:00PM
  - This room: SN011
  - Open book, open notes, no computer, tablet, eReader, or smartphone
  - Will covers material since midterm
  - Study session?  
Sunday or Monday?

