

Lecture 24: Randomized Algorithms

Chapter 12

11/20/2008

Comp 590/Comp 790-90

Fall 2008

1

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.



11/20/2008

Comp 590/Comp 790-90

Fall 2008

2

QuickSort



- **QuickSort** is a simple approach to sorting:
 - Select an element m from unsorted array c and divide the array into two subarrays:
 - c_{small} - elements smaller than m and
 - c_{large} - elements larger than m .
- Recursively quicksort the subarrays and combine them together into the c_{sorted}



Example of QuickSort



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

Step 1: Choose the first element as m

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



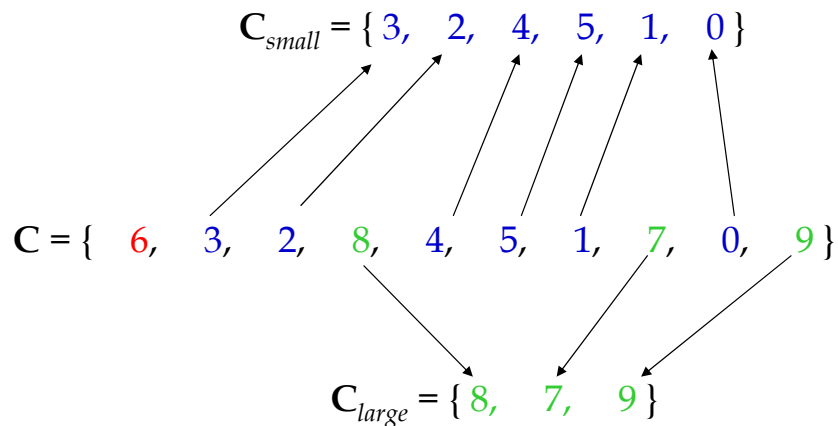
Our Selection



Example of QuickSort (cont'd)



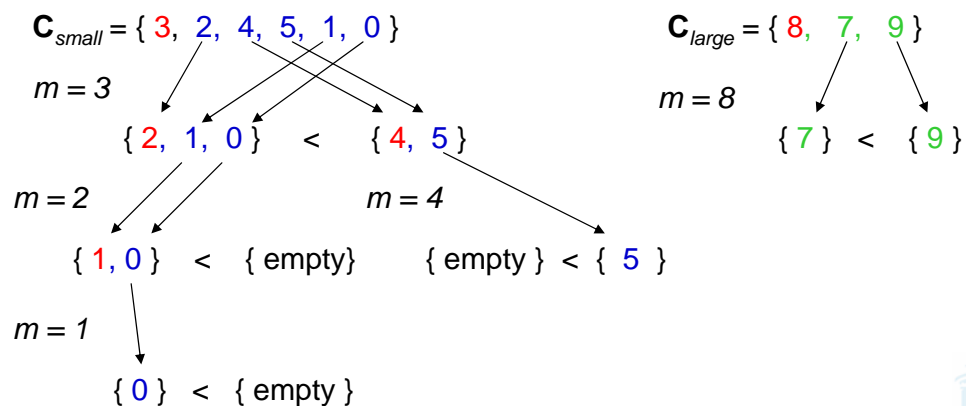
Step 2: Split the array into c_{small} and c_{large}



Example of QuickSort (cont'd)

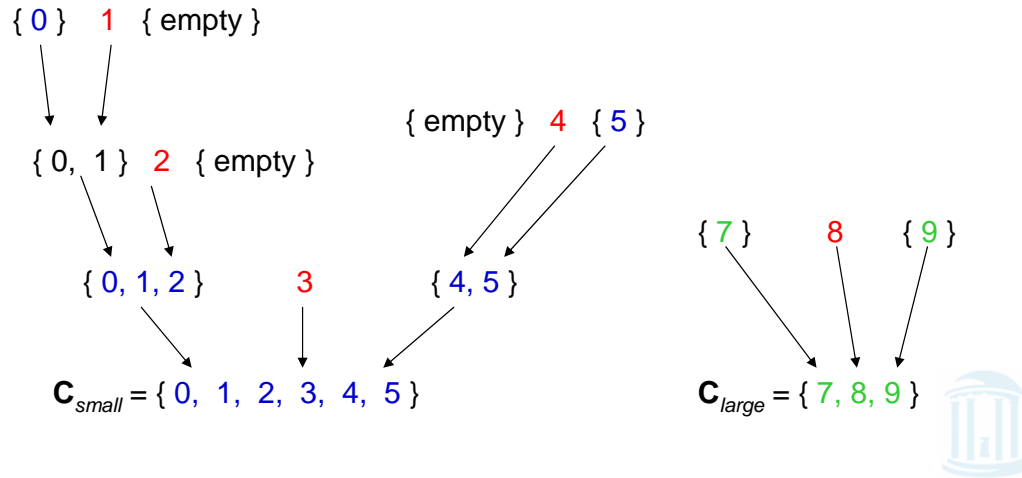


Step 3: Recursively do the same thing to c_{small} and c_{large} until each subarray has only one element or is empty.



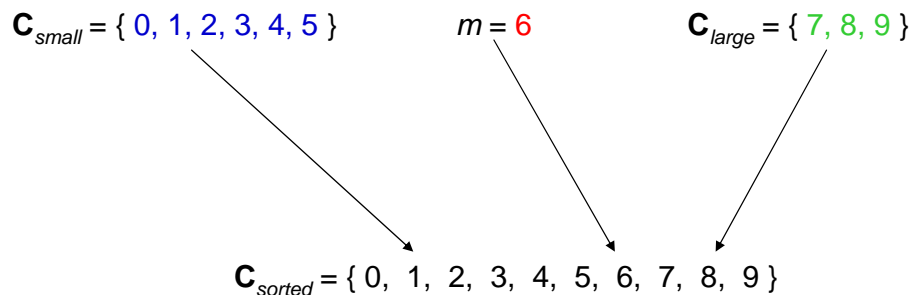
Example of QuickSort (cont'd)

Step 4: Combine the two arrays with m working back out of the recursion and as we build together the sorted array.



Example of QuickSort (cont'd)

Finally we can assemble c_{small} and c_{large} with our original choice of m , creating the sorted array.



The QuickSort Algorithm

1. **QuickSort(c)**
2. if **c** consists of a single element
3. return **c**
4. $m \leftarrow c_1$
5. Determine the set of elements c_{small} smaller than m
6. Determine the set of elements c_{large} larger than m
7. **QuickSort**(c_{small})
8. **QuickSort**(c_{large})
9. Combine c_{small} , m , and c_{large} into a single array, c_{sorted}
10. return c_{sorted}



QuickSort Analysis: Optimistic Outlook

- Runtime depends on our selection of m :

-A good selection will split c evenly such that

$|c_{small}| = |c_{large}|$, then the runtime is $O(n \log n)$.

-For a good selection, the recurrence relation is:

$$T(n) = 2T(n/2) + \alpha n$$

The time it takes to sort two smaller arrays of size $n/2$



Same recurrence as Merge Sort

Time it takes to split the array into 2 parts where $const$ is a positive constant



QuickSort Analysis: Pessimistic Outlook

However, a poor selection will split c unevenly and in the worst case, all elements will be greater or less than m so that one subarray is full and the other is empty. In this case, the runtime is $O(n^2)$.

For a poor selection, the recurrence relation is:

$$T(n) = T(n-1) + \alpha \cdot n$$

The time it takes to sort one array containing $n-1$ elements



Same recurrence as Selection Sort

Time it takes to split the array into 2 parts where $const$ is a positive constant



QuickSort Analysis (cont'd)

- QuickSort seems like an inefficient MergeSort
- To improve QuickSort, we need to choose m to be a good 'splitter.'
- It can be proven that, to achieve $O(n \log n)$ running time, we don't need a perfect split, just reasonably good one. In fact, if both subarrays are at least of size $n/4$, then running time will be $O(n \log n)$.
- This implies that half of the choices of m make good splitters.



A Randomized Approach



- To improve QuickSort, *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.



The RandomizedQuickSort Algorithm



1. RandomizedQuickSort(c)
2. if c consists of a single element
3. return c
4. Choose element m uniformly at random from c
5. Determine the set of elements c_{small} smaller than m
6. Determine the set of elements c_{large} larger than m
7. **RandomizedQuickSort(c_{small})**
8. **RandomizedQuickSort(c_{large})**
9. Combine c_{small} , m , and c_{large} into a single array, c_{sorted}
10. return c_{sorted}

**Lines Highlighted in red indicate the differences between QuickSort and RandomizedQuickSort*



RandomizedQuickSort Analysis



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



Two Types of Randomized Algorithms



- **Las Vegas Algorithms** – always produce the correct solution (ie. RandomizedQuickSort)
- **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 have converged to the hidden motif.



Profiles Revisited



- Let $\mathbf{s}=(s_1, \dots, s_t)$ be the set of starting positions for l -mers in our t sequences.
- The substrings corresponding to these starting positions will form:
 - $t \times l$ *alignment matrix* and
 - $4 \times l$ *profile matrix** \mathbf{P} .

*We make a special note that the profile matrix will be defined in terms of the frequency of letters, and not as the count of letters.



Scoring Strings with a Profile



- $Prob(\mathbf{a} | \mathbf{P})$ 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 $Prob(\mathbf{a} | \mathbf{P})$ will be high
- If \mathbf{a} is very different, then $Prob(\mathbf{a} | \mathbf{P})$ will be low.

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



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:

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



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:

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



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:

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

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 \mathbf{P} .

$\mathbf{P} =$

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

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 $Prob(a|P)$ of every possible 6-mer:

First try: **ctataa**accttacatc

Second try: **ctataaa**ccttacatc

Third try: **ctataaac**ccttacatc

-Continue this process to evaluate every possible 6-mer

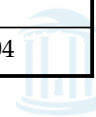


P-Most Probable l -mer (cont'd)



Compute $prob(a|P)$ for every possible 6-mer:

String, Highlighted in Red	Calculations	$prob(a P)$
ctataa ac cttacat	$1/8 \times 1/8 \times 3/8 \times 0 \times 1/8 \times 0$	0
ctataaa ac cttacat	$1/2 \times 7/8 \times 0 \times 0 \times 1/8 \times 0$	0
ctataaac ac cttacat	$1/2 \times 1/8 \times 3/8 \times 0 \times 1/8 \times 0$	0
ctataaac ct cttacat	$1/8 \times 7/8 \times 3/8 \times 0 \times 3/8 \times 0$	0
ctataaac ctt cttacat	$1/2 \times 7/8 \times 3/8 \times 5/8 \times 3/8 \times 7/8$.0336
ctataaac cttt cttacat	$1/2 \times 7/8 \times 1/2 \times 5/8 \times 1/4 \times 7/8$.0299
ctataaac ctttac cttacat	$1/2 \times 0 \times 1/2 \times 0 \times 1/4 \times 0$	0
ctataaac ctttacat cttacat	$1/8 \times 0 \times 0 \times 0 \times 0 \times 1/8 \times 0$	0
ctataaac ctttacatc cttacat	$1/8 \times 1/8 \times 0 \times 0 \times 3/8 \times 0$	0
ctataaac ctttacatc 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	$Prob(a P)$
ctata aa ccttacat	$1/8 \times 1/8 \times 3/8 \times 0 \times 1/8 \times 0$	0
ctata aa ccttacat	$1/2 \times 7/8 \times 0 \times 0 \times 1/8 \times 0$	0
ctata aa ccttacat	$1/2 \times 1/8 \times 3/8 \times 0 \times 1/8 \times 0$	0
ctata aa ccttacat	$1/8 \times 7/8 \times 3/8 \times 0 \times 3/8 \times 0$	0
ctataaaccttacat	$1/2 \times 7/8 \times 3/8 \times 5/8 \times 3/8 \times 7/8$.0336
ctata aa ccttacat	$1/2 \times 7/8 \times 1/2 \times 5/8 \times 1/4 \times 7/8$.0299
ctata aa ccttacat	$1/2 \times 0 \times 1/2 \times 0 \times 1/4 \times 0$	0
ctata aa ccttacat	$1/8 \times 0 \times 0 \times 0 \times 0 \times 1/8 \times 0$	0
ctata aa ccttacat	$1/8 \times 1/8 \times 0 \times 0 \times 3/8 \times 0$	0
ctata aa ccttacat	$1/8 \times 1/8 \times 3/8 \times 5/8 \times 1/8 \times 7/8$.0004

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



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

ctataaaccttacatc

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

Dealing with Zeroes



- In our toy example $prob(\mathbf{a} | \mathbf{P})=0$ 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 $prob(\mathbf{a} | \mathbf{P})=0$, 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 (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 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
 tgcattcaatagctta
 tatcctttccactcac
 ctccaaatcctttaca
 ggtcatcctttatcct



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



ctat**aaacg**tacatc

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

cagcccaga**aaccct**

cggt**gaacct**tacatc

tgcat**tcaatagct**ta

tgtcctgtccactcac

ctccaa**atcctt**taca

gg**tacctt**tatcct

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.
- 2) Create a profile \mathbf{P} from the substrings at these starting positions.
- 3) 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} .
- 4) 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 $\mathbf{s}=(s_1, \dots, s_t)$ from DNA
3. $bestScore \leftarrow 0$
4. **while** $Score(\mathbf{s}, DNA) > bestScore$
5. Form profile \mathbf{P} from \mathbf{s}
6. $bestScore \leftarrow Score(\mathbf{s}, DNA)$
7. **for** $i \leftarrow 1$ to t
8. Find a \mathbf{P} -most probable l -mer \mathbf{a} from the i^{th} sequence
9. $s_i \leftarrow$ starting position of \mathbf{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 $s = (s_1, \dots, s_t)$ 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:

$s_1=7$	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: AAAATTACCTTAGAAGG

$s_1=7$ GTAAACA**AATATTTA**TAGC

$s_3=9$ CCGTACTGT**CAAGCGT**GG

$s_4=4$ TGAG**TAAACGAC**GTCCCA

$s_5=1$ **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	$prob(a P)$
AAAATTTACCTTAGAAGG	.000732
AAAATTTACCTTAGAAGG	.000122
AAATTTACCTTAGAAGG	0
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 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 lowest probability:

$$\text{Starting Position 1: } prob(\text{AAAATTTA} | P) = .000732 / .000122 = 6$$

$$\text{Starting Position 2: } prob(\text{AAATTTAC} | P) = .000122 / .000122 = 1$$

$$\text{Starting Position 8: } prob(\text{ACCTTAGA} | P) = .000183 / .000122 = 1.5$$

$$\text{Ratio} = 6 : 1 : 1.5$$

Turning Ratios into Probabilities



b) Define probabilities of starting positions according to computed ratios

Probability (Selecting Starting Position 1): $6/(6+1+1.5)= 0.706$

Probability (Selecting Starting Position 2): $1/(6+1+1.5)= 0.118$

Probability (Selecting Starting Position 8): $1.5/(6+1+1.5)=0.176$



Gibbs Sampling: an Example



c) Select the start position according to computed ratios:

P(selecting starting position 1): .706

P(selecting starting position 2): .118

P(selecting starting position 8): .176



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 unequal distributions of nucleotides (*relative entropy* approach).
- Gibbs sampling often converges to locally optimal motifs rather than globally optimal motifs.
- Needs to be run with many randomly chosen seeds to achieve good results.



Another Randomized Approach

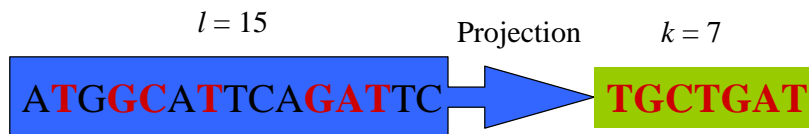


- **Random Projection Algorithm** is a different way to solve the Motif Finding Problem.
- **Guiding principle:** Some instances of a motif agree on a subset of positions.
- However, it is unclear how to find these “non-mutated” positions.
- To bypass the effect of mutations within a motif, we randomly 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.



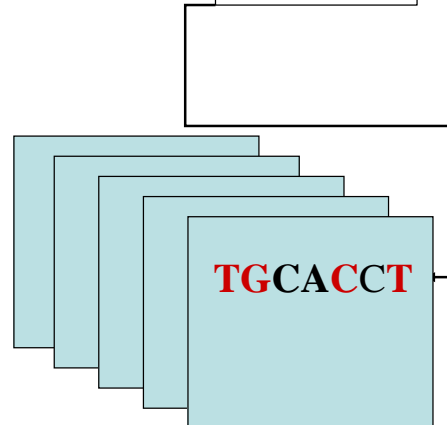
Projection = (2, 4, 5, 7, 11, 12, 13)



Random Projections Algorithm

- Select k out of l positions uniformly at random.
- For each l -tuple in input sequences, hash into bucket based on letters at k selected positions.
- Recover motif from *enriched* bucket that contain many l -tuples.

Input sequence:
...TCAATGCACCTAT...



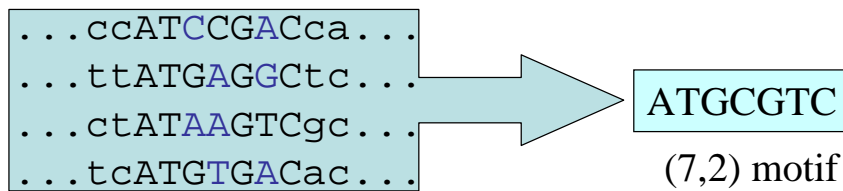
Bucket TGCT



Random Projections Algorithm (cont'd)



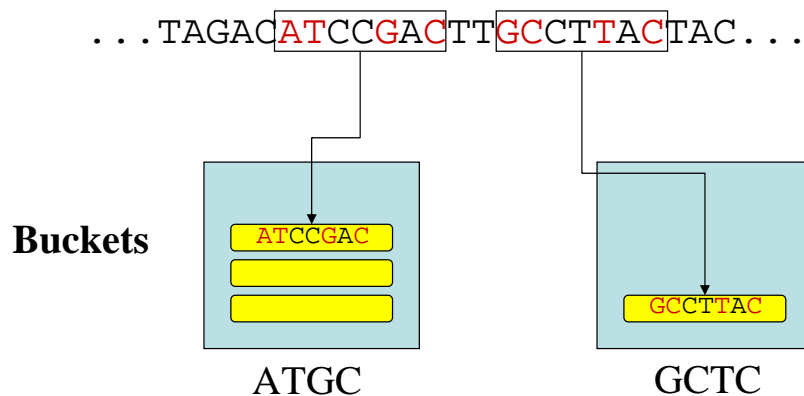
- Some projections will fail to detect motifs but if we try many of them the probability that one of the buckets fills in is increasing.
- In the example below, the bucket ****GC*AC** is “bad” while the bucket **AT**G*C** is “good”



Example



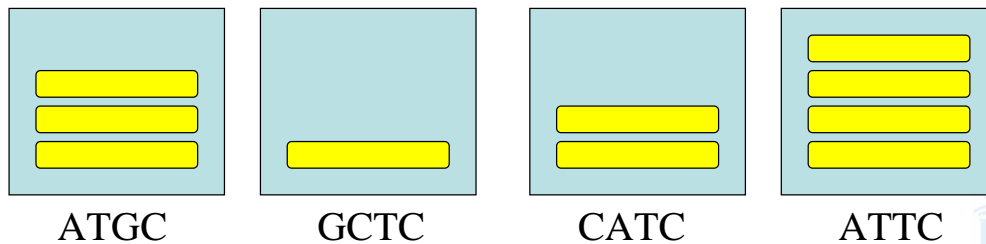
- $l = 7$ (motif size) , $k = 4$ (projection size)
- Choose projection (1,2,5,7)



Hashing and Buckets



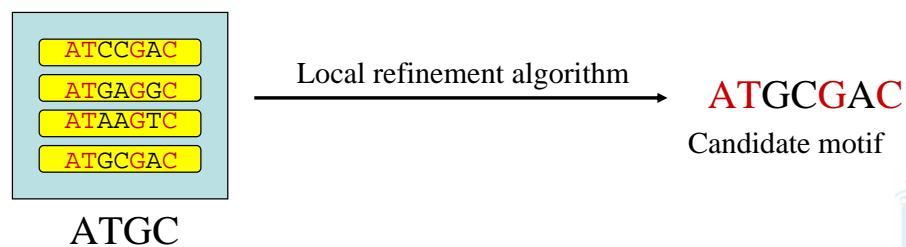
- Hash function $h(x)$ obtained from k positions of projection.
- Buckets are labeled by values of $h(x)$.
- *Enriched buckets*: contain more than s l -tuples, for some parameter s .



Motif Refinement



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



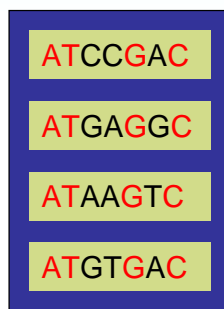
Synergy between Random Projection and Gibbs Sampler



- 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



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.
 - choose k large enough to avoid contamination by spurious l -mers:

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



How Many Iterations?



- *Planted bucket* : bucket with hash value $h(M)$, where M is the motif.
- Choose m = number of iterations, such that

Pr(planted bucket contains at least s sequences
in at least one of m iterations) = 0.95

- Probability is readily computable since iterations form a sequence of independent Bernoulli trials

