Lecture 5: Algorithm design and time/space complexity analysis

Torgeir R. Hvidsten

Professor
Norwegian University of Life Sciences

Guest lecturer
Umeå Plant Science Centre
Computational Life Science Cluster (CLiC)

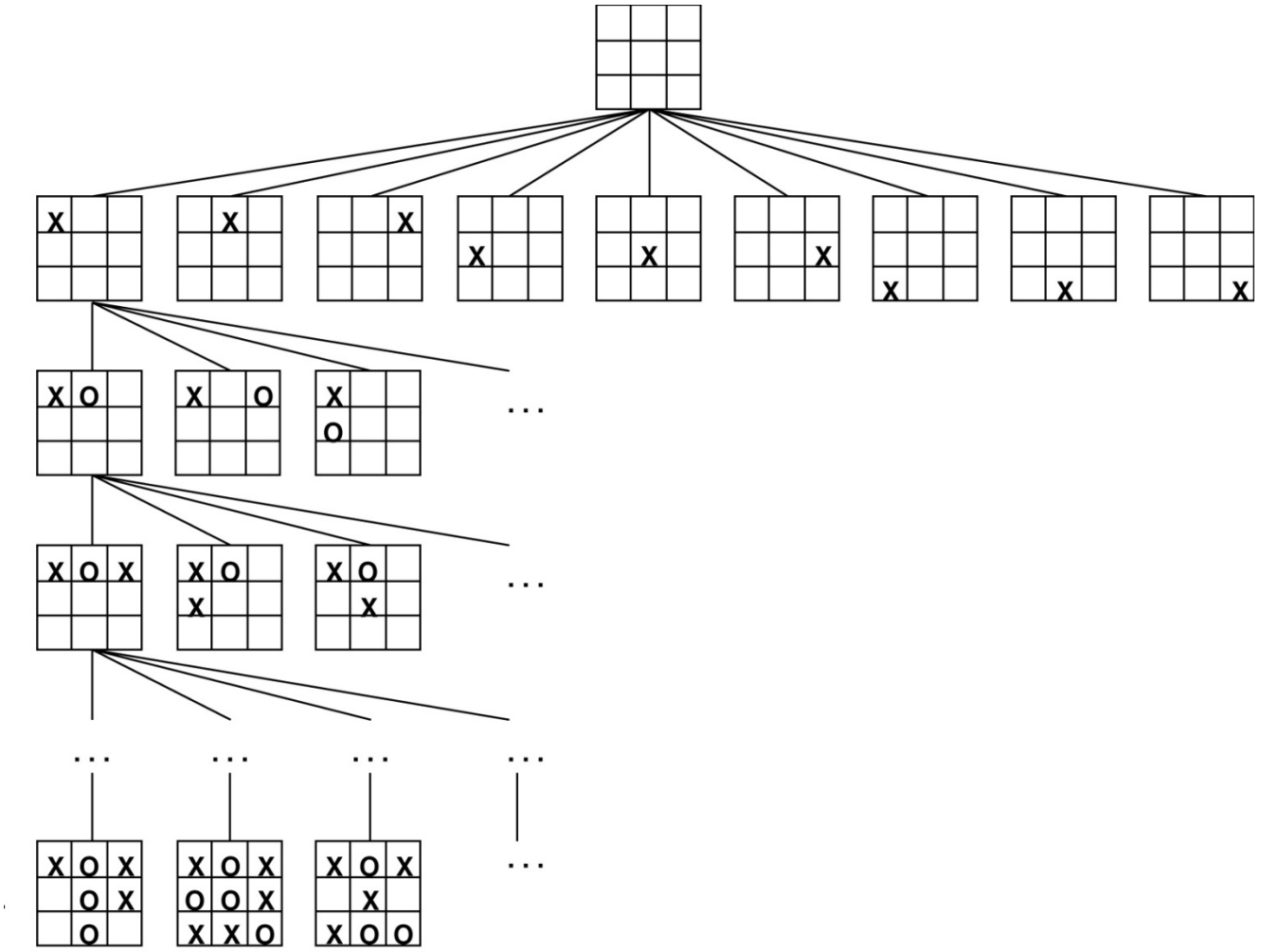
This lecture

- Basic algorithm design: exhaustive search, greedy algorithms, dynamic programming and randomized algorithms
- Correct versus incorrect algorithms
- Time/space complexity analysis
- Go through Lab 3

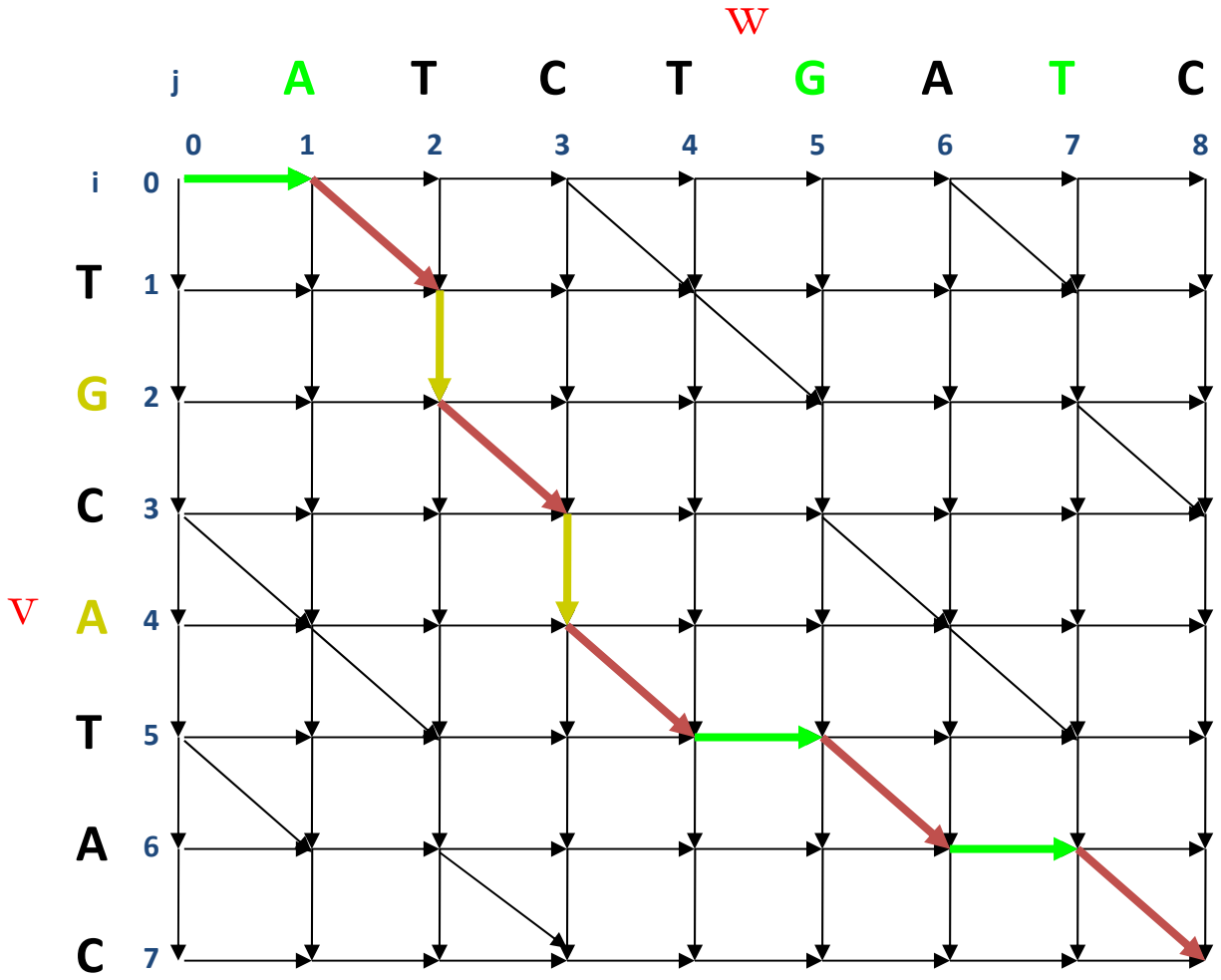
Algorithm

- Algorithm: a sequence of instructions that one must perform in order to solve a well-formulated problem
- Correct algorithm: translate every input instance into the correct output
- Incorrect algorithm: there is at least one input instance for which the algorithm does not produce the correct output
- Many successful algorithms in bioinformatics are not "correct" (optimal)

Search space



Sequence alignment as a search problem



Deletion

Matches

Insertion

-TGCAT-A-C
AT-C-TGATC

Algorithm design (I)

- Exhaustive algorithms (brute force): examine every possible alterative to find the solution
- Branch-and-bound algorithms: omit searching through a large number of alternatives by branch-and-bound or pruning
- Greedy algorithms: find the solution by always choosing the currently ”best” alternative
- Dynamic programming: use the solution of the subproblems of the original problem to construct the solution

Algorithm design (II)

- Divide-and-conquer algorithms: splits the problem into subproblems and solve the problems independently
- Randomized algorithms: finds the solution based on randomized choices

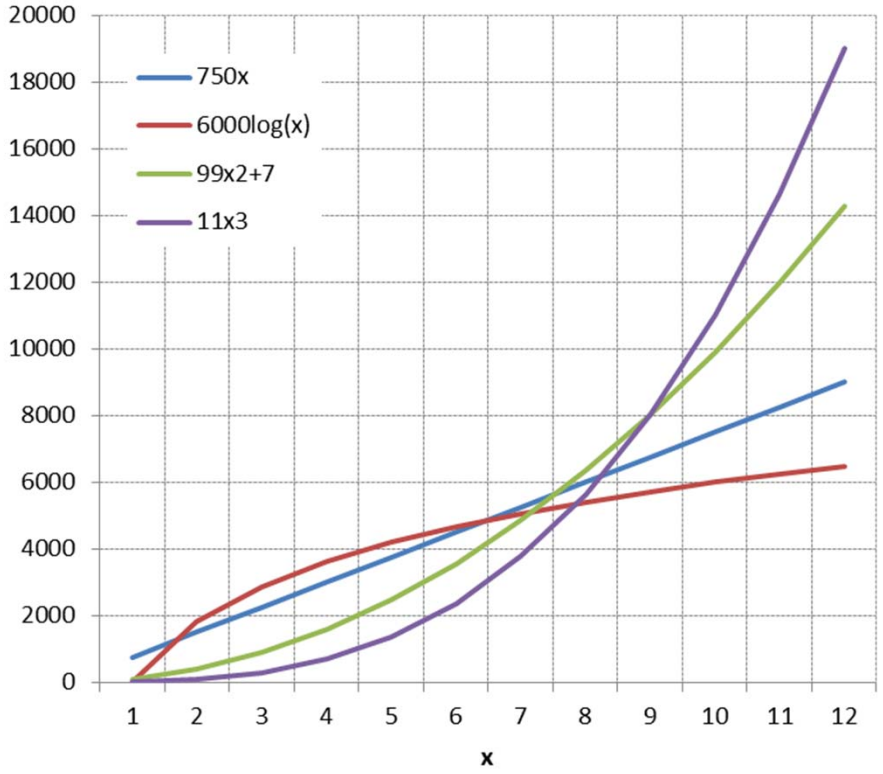
- Machine learning: induce models based on previously labeled observations (examples)

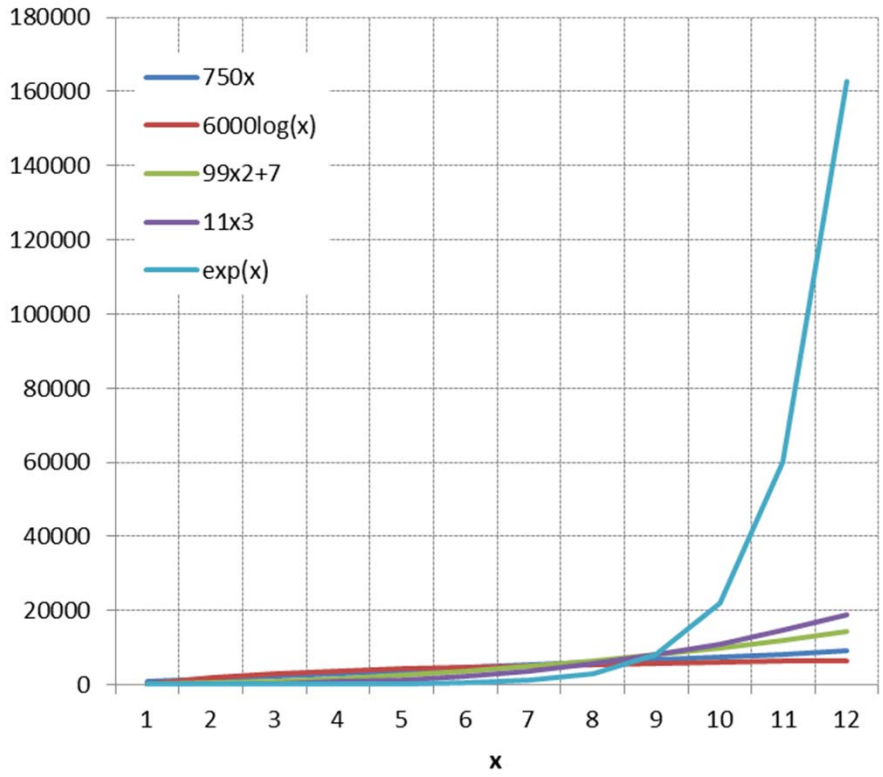
Algorithm complexity

- The Big-O notation:
  - the running time of an algorithm as a function of the size of its input
  - worst case estimate
  - asymptotic behavior
- $O(n^2)$ means that the running time of the algorithm on an input of size n is limited by the quadratic function of n

Big-O Notation

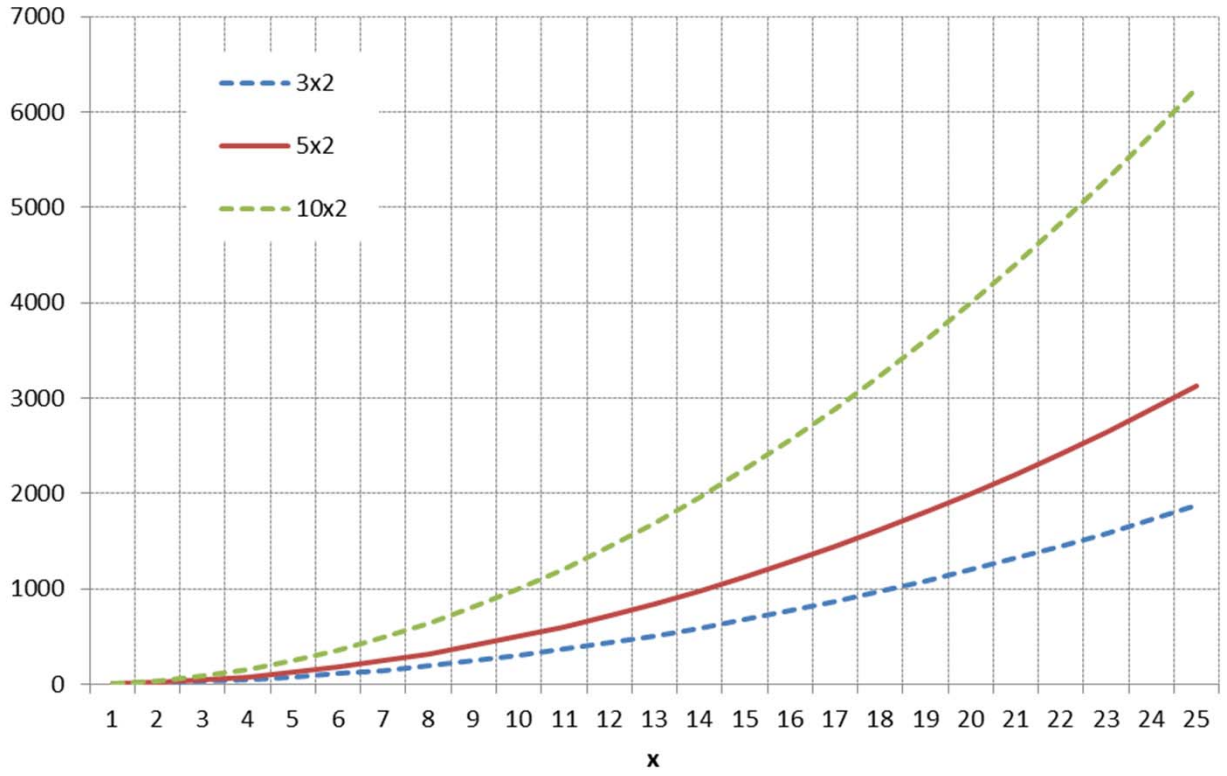
- A function $f(x)$ is $O(g(x))$ if there are positive real constants c and x_0 such that $f(x) \leq cg(x)$ for all values of $x \geq x_0$.





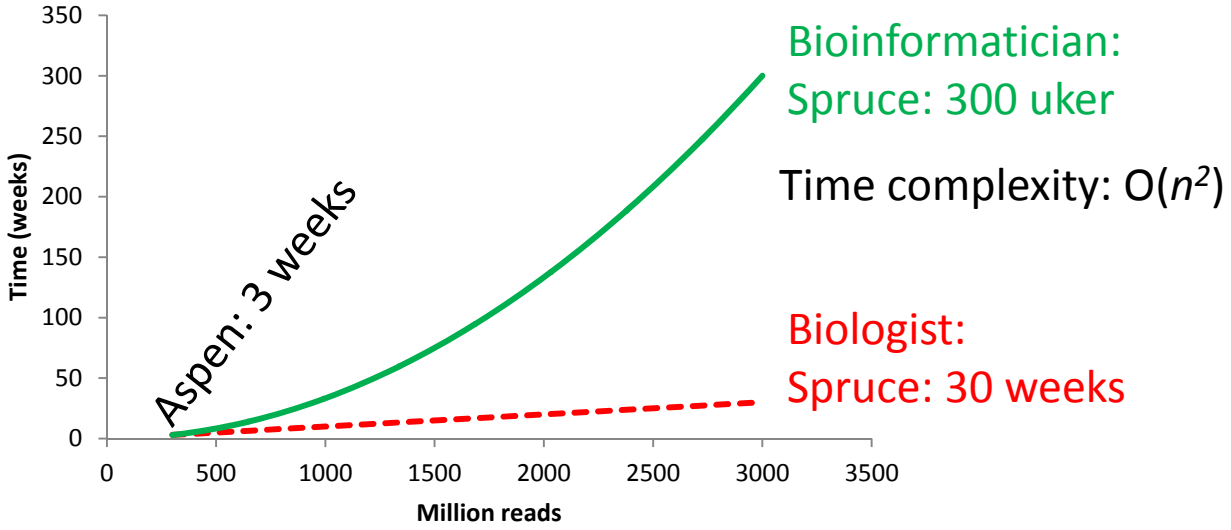
Big-O Notation

- A function $f(x)$ is $O(g(x))$ if there are positive real constants c and x_0 such that $f(x) \leq cg(x)$ for all values of $x \geq x_0$.



Time complexity

- Genome assembly: pice together a genome from short reads (~200bp)
  - Aspen: 300M reads
  - Spruce: 3000M reads
- Pair-wise all-against all alignment for Aspen takes 3 weeks on 16 porcessors
- What about spruce?



Sorting algorithm

Sorting problem: Sort a list of n integers $\boldsymbol{a} = (a_1, a_2, \ldots, a_n)$

SelectionSort($\boldsymbol{a}$,n)

1 **for** $i \leftarrow 1$ **to** $n-1$

2 $j \leftarrow$ Index of the smallest element among $a_i, a_{i+1}, \ldots, a_n$

3 Swap elements a_i and a_j

4 **return** $\boldsymbol{a}$

Example run

$i = 1$: (**7**,92,87,1,4,3,2,6)

$i = 2$: (1,**92**,87,7,4,3,2,6)

$i = 3$: (1,2,**87**,7,4,3,92,6)

$i = 4$: (1,2,3,**7**,4,87,92,6)

$i = 5$: (1,2,3,4,**7**,87,92,6)

$i = 6$: (1,2,3,4,6,**87**,92,7)

$i = 7$: (1,2,3,4,6,7,**92**,87)

(1,2,3,4,6,7,87,92)

Complexity of SelectionSort

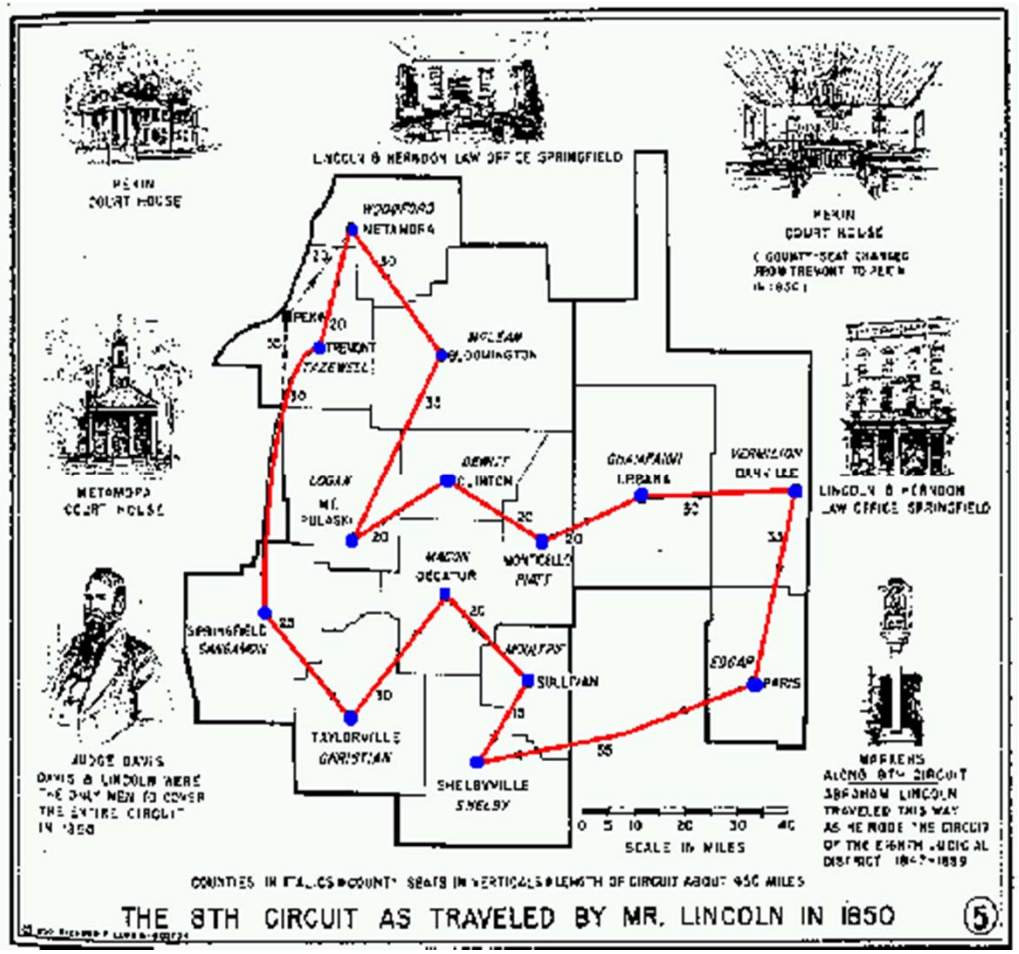
- Makes $n - 1$ iterations in the for loop
- Analyzes $n - i + 1$ elements $a_i, a_{i+1}, \ldots, a_n$ in iteration i
- Approximate number of operations:
  - $n + (n-1) + (n-2) + \ldots + 2 + 1 = n(n+1)/2$
  - plus the swapping: $n(n+1)/2 + 3n = 1/2\ n^2 + 3n + 1/2$

- Thus the algorithm is $O(n^2)$

Tractable versus intractable problems

- Some problems requires polynomial time
  - e.g. sorting a list of integers
  - called tractable problems
- Some problems require exponential time
  - e.g. listing every subset in a list
  - called intractable problems
- Some problems lie in between
  - e.g. the traveling salesman problem
  - called NP-complete problems
  - nobody have proved whether a polynomial time algorithm exists for these problems

Traveling salesman problem



Exhaustive search: Finding regulatory motifs in DNA sequences

Random sample

atgaccgggatactgataccgtatttggcctaggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg

acccctatttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaatactgggcataaggtaca

tgagtatccctgggatgacttttgggaacactatagtgctctcccgatttttgaatatgtaggatcattcgccagggtccga

gctgagaattggatgaccttgtaagtgttttccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga

tccctttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatggcccacttagtccacttatag

gtcaatcatgttcttgtgaatggatttttaactgagggcatagaccgcttggcgcacccaaattcagtgtgggcgagcgcaa

cggttttggcccttgttagaggcccccgtactgatggaaactttcaattatgagagagctaatctatcgcgtgcgtgttcat

aacttgagttggtttcgaaaatgctctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta

ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatttcaacgtatgccgaaccgaaagggaag

ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttctgggtactgatagca

Implanting motif **AAAAAAAGGGGGGG**

atgaccgggatactgatAAAAAAAAGGGGGGGggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg

acccctatttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaataAAAAAAAAGGGGGGGa

tgagtatccctgggatgacttAAAAAAAAGGGGGGGtgctctcccgatttttgaatatgtaggatcattcgccagggtccga

gctgagaattggatgAAAAAAAAGGGGGGGtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga

tccctttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatAAAAAAAAGGGGGGGcttatag

gtcaatcatgttcttgtgaatggatttAAAAAAAAGGGGGGGgaccgcttggcgcacccaaattcagtgtgggcgagcgcaa

cggttttggcccttgttagaggcccccgtAAAAAAAAGGGGGGGcaattatgagagagctaatctatcgcgtgcgtgttcat

aacttgagttAAAAAAAAGGGGGGGctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta

ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatAAAAAAAAGGGGGGGaccgaaagggaag

ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttAAAAAAAAGGGGGGGa

Where is the implanted motif?

atgaccgggatactgataaaaaaaaggggggggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg

acccctatttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaactttccgaataaaaaaaaggggggga

tgagtatccctgggatgacttaaaaaaaaggggggtgctctcccgatttttgaatatgtaggatcattcgccagggtccga

gctgagaattggatgaaaaaaaaggggggtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga

tccctttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaataaaaaaaaggggggcttatag

gtcaatcatgttcttgtgaatggatttaaaaaaaaggggggggaccgcttggcgcacccaaattcagtgtgggcgagcgcaa

cggttttggcccttgttagaggcccccgtaaaaaaaaggggggcaattatgagagagctaatctatcgcgtgcgtgttcat

aacttgagttaaaaaaaaggggggctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta

ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcataaaaaaaaggggggggaccgaaagggaag

ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttaaaaaaaaggggggga

Implanting motif **AAAAAAGGGGGGG** with four random mutations

atgaccgggatactgatAgAAgAAAGGttGGGggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg

acccctatttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaatacAAtAAAAcGGcGGGa

tgagtatccctgggatgacttAAAAtAAtGGaGtGGtgctctcccgatttttgaatatgtaggatcattcgccagggtccga

gctgagaattggatgcAAAAAAAGGGattGtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga

tccctttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatAtAAtAAAGGaaGGGcttatag

gtcaatcatgttcttgtgaatggatttAAcAAtAAGGGctGGgaccgcttggcgcacccaaattcagtgtgggcgagcgcaa

cggttttggcccttgttagaggcccccgtAtAAAcAAGGaGGGccaattatgagagagctaatctatcgcgtgcgtgttcat

aacttgagttAAAAAAtAGGGaGccctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta

ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatActAAAAAGGaGcGGaccgaaagggaag

ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttActAAAAAGGaGcGGa

Where is the motif?

atgaccgggatactgatagaagaaaggttgggggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg

acccctatttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaatacaataaaacggcggga

tgagtatccctgggatgacttaaaataatggagtggtgctctcccgatttttgaatatgtaggatcattcgccagggtccga

gctgagaattggatgcaaaaaaagggattgtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga

tccctttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatataataaaggaagggcttatag

gtcaatcatgttcttgtgaatggatttaacaataagggctgggaccgcttggcgcacccaaattcagtgtgggcgagcgcaa

cggttttggcccttgttagaggcccccgtataaacaaggagggccaattatgagagagctaatctatcgcgtgcgtgttcat

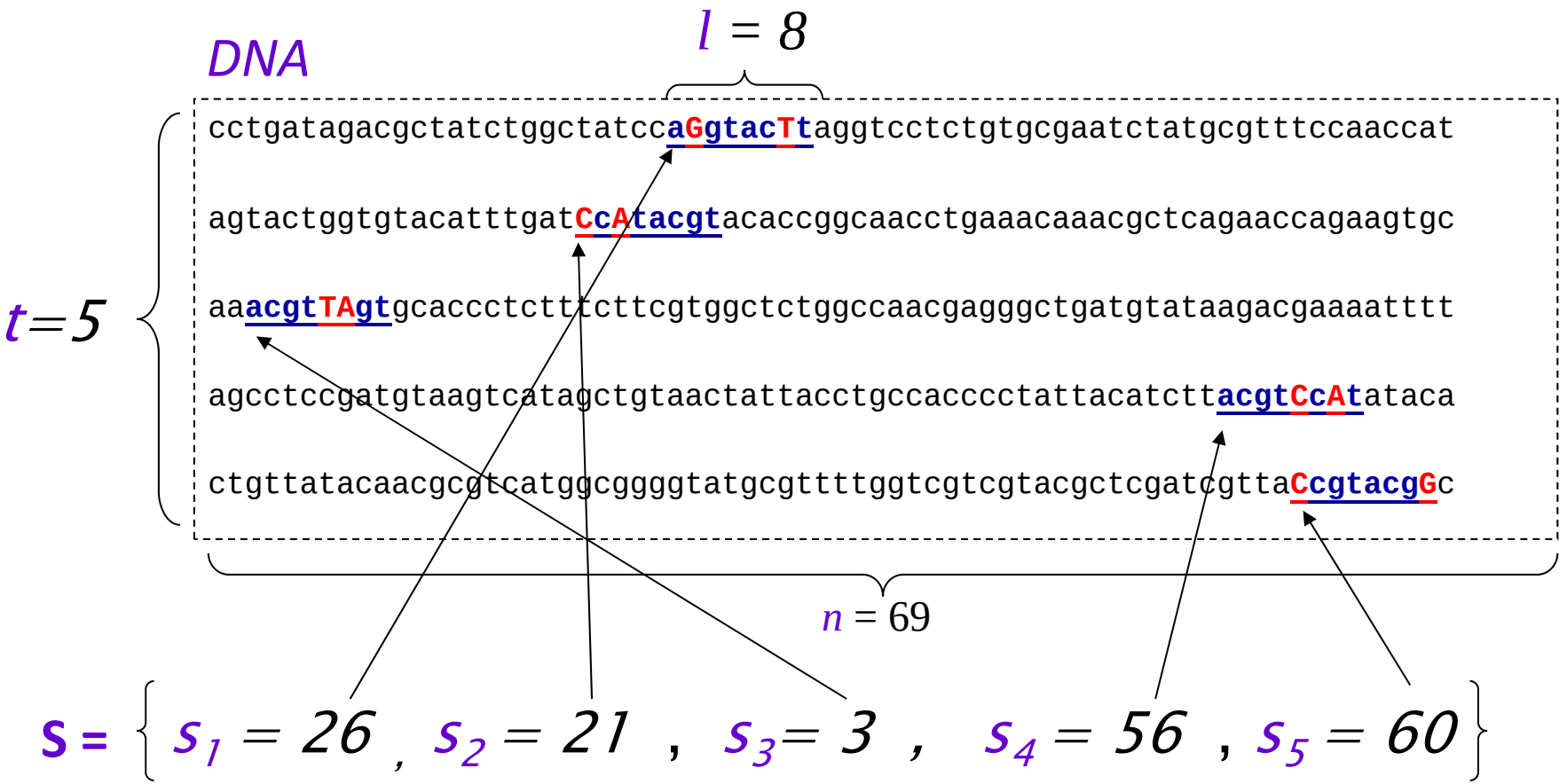
aacttgagttaaaaaatagggagccctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta

ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatactaaaaaggagcggaccgaaagggaag

ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttactaaaaaggagcgga

Why finding motif is difficult

Parameters


l = 8
DNA
cctgatagacgctatctggctatccaGgtacTtaggtcctctgtgcgaatctatgcgtttccaaccat
agtactggtgtacatttgatCcAtacgtacaccggcaacctgaaacaaacgctcagaaccagaagtgc
aaacgtTAgtgcaccctctttcttcgtggctctggccaacgagggctgatgtataagacgaaaatttt
agcctccgatgtaagtcatagctgtaactattacctgccacccctattacatcttacgtCcAtataca
ctgttatacaacgcgtcatggcggggtatgcgttttggtcgtcgtacgctcgatcgttaCcgtacgGc
t=5
n = 69
S = { s1 = 26 , s2 = 21 , s3 = 3 , s4 = 56 , s5 = 60 }

Motifs: Profiles and consensus

Alignment

```
a G g t a c 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
```

Profile

```
A  3 0 1 0 3 1 1 0
C  2 4 0 0 1 4 0 0
G  0 1 4 0 0 0 3 1
T  0 0 0 5 1 0 1 4
```

Consensus

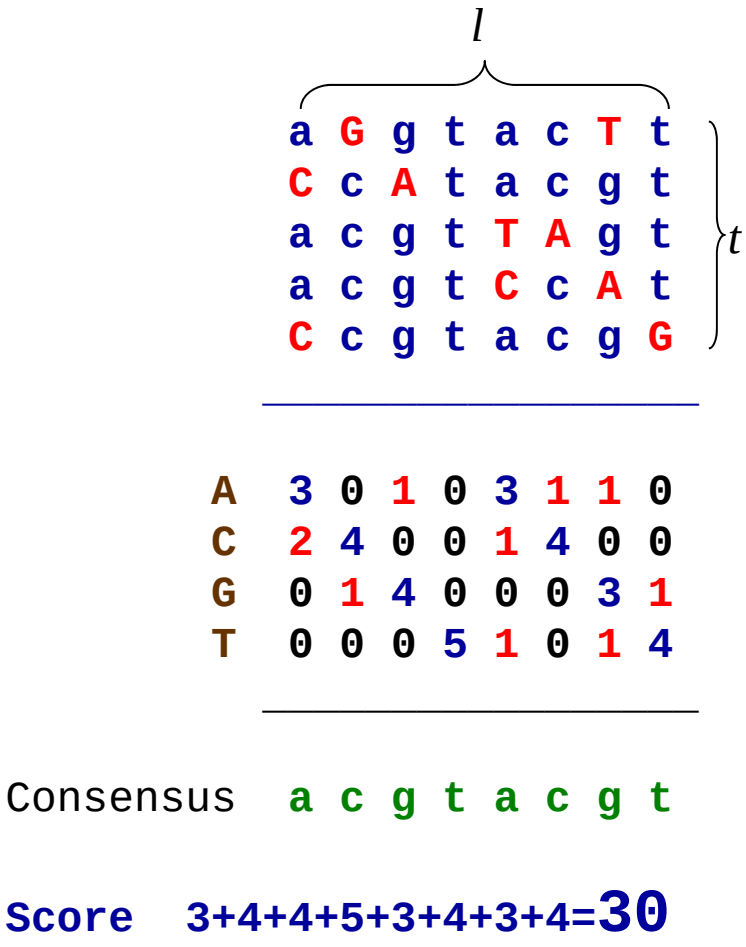
```
A C G T A C G T
```

- Line up the patterns by their start indexes

$$\mathbf{s} = (s_1, s_2, \ldots, s_t)$$

- Construct matrix profile with frequencies of each nucleotide in columns
- Consensus nucleotide in each position has the highest score in column

Scoring motifs: consensus score



BruteForceMotifSearch

BruteForceMotifSearch(***DNA***, *t*, *n*, *l*)

1 *bestScore* ← *0*

2 **for** each **s=**(s_1,s_2 , . . ., s_t) from (*1*,*1* . . .,*1*) to (*n*-*l*+*1*, . . ., *n*-*l*+*1*)

3 **if** (Score(***s***,***DNA***) > *bestScore*)

4 *bestScore* ← Score(**s, *DNA***)

5 *bestMotif* ← (s_1,s_2 , . . . , s_t)

6 **return** *bestMotif*

Running Time of BruteForceMotifSearch

- Varying $(n - l + 1)$ positions in each of t sequences, we're looking at $(n - l + 1)^t$ sets of starting positions

- For each set of starting positions, the scoring function makes l operations, so complexity is
  $l(n - l + 1)^t = O(ln^t)$

- That means that for $t = 8$, $n = 1000$, and $l = 10$ we must perform approximately 10^{20} computations – it will take billions of years!

Greedy search: Finding regulatory motifs in DNA sequences

Approximation algorithms

- These algorithms find approximate solutions rather than optimal solutions
- The approximation ratio of an algorithm A on input π is:

$$A(\pi) / OPT(\pi)$$

where

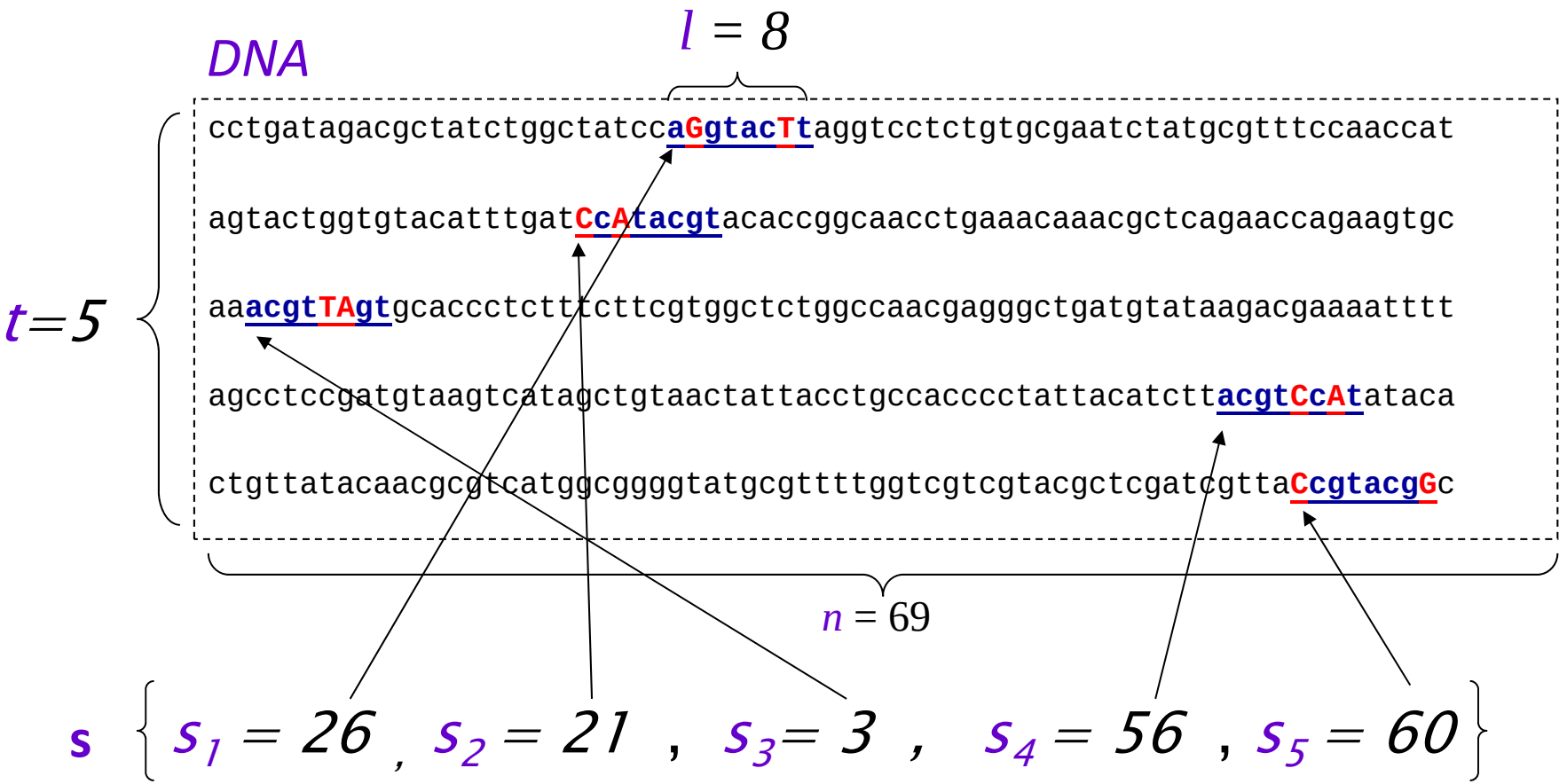
$A(\pi)$ - solution produced by algorithm A

$OPT(\pi)$ - optimal solution of the problem

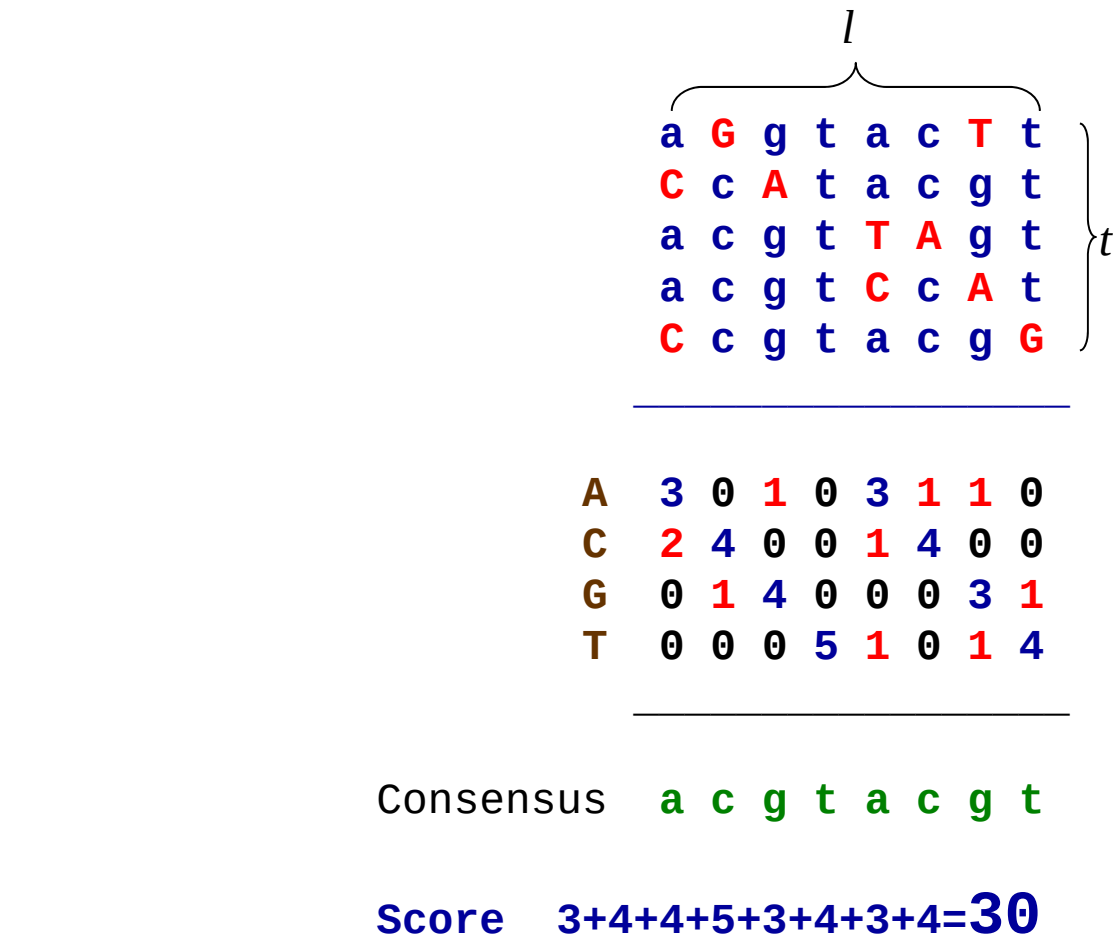
Performance guarantee

- Performance guarantee of algorithm A is the maximal approximation ratio of all inputs of size n
- For algorithm A that minimizes the objective function (minimization algorithm):
  - $\max_{|\pi|=n} A(\pi) / OPT(\pi)$
- For maximization algorithms
  - $\min_{|\pi|=n} A(\pi) / OPT(\pi)$

Parameters


l = 8
DNA
cctgatagacgctatctggctatccaGgtacTtaggtcctctgtgcgaatctatgcgtttccaaccat
agtactggtgtacatttgatCcAtacgtacaccggcaacctgaaacaaacgctcagaaccagaagtgc
aaacgtTAgtgcaccctctttcttcgtggctctggccaacgagggctgatgtataagacgaaaatttt
agcctccgatgtaagtcatagctgtaactattacctgccacccctattacatcttacgtCcAtataca
ctgttatacaacgcgtcatggcggggtatgcgttttggtcgtcgtacgctcgatcgttaCcgtacgGc
t=5
n = 69
s
s1 = 26 , s2 = 21 , s3= 3 , s4 = 56 , s5 = 60

Scoring motifs: consensus score



Greedy motif finding

- Partial score: Score(***s***, *i*, ***DNA***)
  - The consensus score for the first *i* sequences
- Algorithm:
  - Find the optimal motif for the two first sequences
  - Scan the remaining sequences only once, and choose the motif with the best contribution to the partial score

Greedy motif finding

```
GreedyMotifSearch(DNA, t, n, l)
1       s ← (1,1, …, 1)
2       bestMotif ← s
3       for s_1 ← 1 to n – l + 1
4               for s_2 ← 1 to n – l + 1
5                       if Score(s, 2, DNA) > Score(bestMotif, 2, DNA)
6                               bestMotif_1 ← s_1
7                               bestMotif_2 ← s_2
8       s_1 ← bestMotif_1
9       s_2 ← bestMotif_2
10      for i ← 3 to t
11              for s_i ← 1 to n – l + 1
12                      if Score(s, i, DNA) > Score(bestMotif, i, DNA)
13                              bestMotif_i ← s_i
14               s_i ← bestMotif_i
15      return bestMotif
```

Running time

- Optimal motif for the two first sequences
  - $l(n - l + 1)^2$ operations
- The remaining *t-2* sequence
  - $(t - 2)l(n - l + 1)$ operations
- Running time
  - $O(ln^2 + tln)$ or $O(ln^2)$ if $n >> t$
- Vastly better than
  - BruteForceMotifSearch: $O(ln^t)$

Dynamic programming: Sequence alignment

Lecture 6

Randomized algorithms: Finding regulatory motifs in DNA sequences

Randomized algorithms

- Randomized algorithms make random rather than deterministic decisions
- The main advantage is that no input can reliably produce worst-case results because the algorithm runs differently each time
- These algorithms are commonly used in situations where no correct polynomial algorithm is known

Two types of randomized algorithms

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

Profiles

- Let **s**=(s_1,...,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* x *l* **alignment** and

  - 4 x *l* **profile** **P**

Scoring strings with a profile

Given a profile: **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(**aaacct**|**P**) = 1/2 x 7/8 x 3/8 x 5/8 x 3/8 x 7/8 = .033646

Probability of a different string:

Prob(**atacag**|**P**) = 1/2 x 1/8 x 3/8 x 5/8 x 1/8 x 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

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

String, Highlighted in Red	Calculations	*Prob*(**a** | **P**)
ctataaaccttacat	1/8 x 1/8 x 3/8 x 0 x 1/8 x 0	0
ctataaaccttacat	1/2 x 7/8 x 0 x 0 x 1/8 x 0	0
ctataaaccttacat	1/2 x 1/8 x 3/8 x 0 x 1/8 x 0	0
ctataaaccttacat	1/8 x 7/8 x 3/8 x 0 x 3/8 x 0	0
ctataaaccttacat	**1/2 x 7/8 x 3/8 x 5/8 x 3/8 x 7/8**	**.0336**
ctataaaccttacat	1/2 x 7/8 x 1/2 x 5/8 x 1/4 x 7/8	.0299
ctataaaccttacat	1/2 x 0 x 1/2 x 0 1/4 x 0	0
ctataaaccttacat	1/8 x 0 x 0 x 0 x 0 x 1/8 x 0	0
ctataaaccttacat	1/8 x 1/8 x 0 x 0 x 3/8 x 0	0
ctataaaccttacat	1/8 x 1/8 x 3/8 x 5/8 x 1/8 x 7/8	.0004

Gibbs sampling

1) Randomly choose starting positions $\mathbf{s} = (s_1,...,s_t)$ and form the set of l-mers associated with these starting positions
2) Randomly choose one of the t sequences
3) Create a profile **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

3) Create profile ***P*** from *l*-mers in the 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 2:

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 *l*-mers $prob(\boldsymbol{a}|\boldsymbol{P})$, and randomly select a new starting position based on this distribution

   To create a proper distribution, divide each probability $prob(\boldsymbol{a}|\boldsymbol{P})$ by the sum of probabilities over all position:

   Probability (Selecting Starting Position 1) = 0.706

   Probability (Selecting Starting Position 2) = 0.118

   …

   Probability (Selecting Starting Position 8) = 0.176

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
$s_2=1$	AAAATTTACCTTAGAAGG
$s_3=9$	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