Lecture 4

Torgeir R. Hvidsten
Assistant professor in Bioinformatics
Umeå Plant Science Center (UPSC)
Computational Life Science Centre (CLiC)

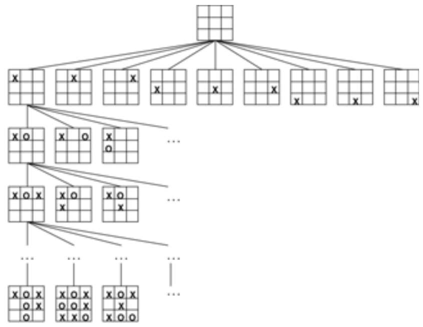
This lecture

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

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"

Search space



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)

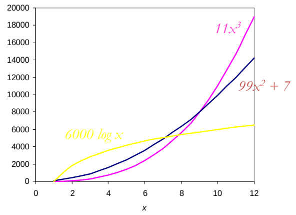
- Divide-and-conquer algorithms: splits the problem into subproblems and solve the problems independently
- Machine learning: induce models based on previously labeled observations (examples)
- Randomized algorithms: finds the solution based on randomized choices

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 $x \geq x_0$.



Sorting algorithm

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

```
SelectionSort($\boldsymbol{a}$,$n$)
    for $i \leftarrow 1$ to $n-1$
        $j \leftarrow$ Index of the smallest element
        among $a_i, a_{i+1}, \ldots, a_n$
        Swap elements $a_i$ and $a_j$
    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)

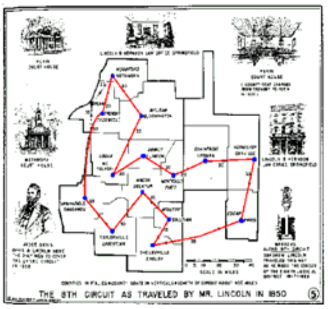
Complexity of SelectionSort

- 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:
  - $n + (n-1) + (n-2) + \ldots + 2 + 1 = n(n+1)/2$
- 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
acccctattttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaatactgggcataaggtaca
tgagtatccctgggatgacttttgggaacactatagtgctctcccgatttttgaatatgtaggatcattcgccagggtccga
gctgagaattggatgaccttgtaagtgttttccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga
tcccttttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatggcccacttagtccacttatag
gtcaatcatgttcttgtgaatggattttaactgagggcatagaccgcttggcgcacccaaattcagtgtgggcgagcgcaa
cggttttggcccttgttagaggcccccgtactgatggaaactttcaattatgagagagctaatctatcgcgtgcgtgttcat
aacttgagttggtttcgaaaatgctctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta
ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatttcaacgtatgccgaaccgaaagggaag
ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttctgggtactgatagca
```

Implanting motif **AAAAAAAGGGGGGG**

```
atgaccgggatactgatAAAAAAAAGGGGGGGggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg
acccctattttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaataAAAAAAAAGGGGGGGa
tgagtatccctgggatgacttAAAAAAAAGGGGGGGtgctctcccgatttttgaatatgtaggatcattcgccagggtccga
gctgagaattggatgAAAAAAAAGGGGGGGtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga
tcccttttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatAAAAAAAAGGGGGGGcttatag
gtcaatcatgttcttgtgaatggatttAAAAAAAAGGGGGGGgaccgcttggcgcacccaaattcagtgtgggcgagcgcaa
cggttttggcccttgttagaggcccccgtAAAAAAAAGGGGGGGcaattatgagagagctaatctatcgcgtgcgtgttcat
aacttgagttAAAAAAAAGGGGGGGctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta
ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatAAAAAAAAGGGGGGGaccgaaagggaag
ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttAAAAAAAAGGGGGGGa
```

Where is the implanted motif?

```
atgaccgggatactgataaaaaaaagggggggggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg
acccctattttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaataaaaaaaaaggggggga
tgagtatccctgggatgacttaaaaaaaagggggggtgctctcccgatttttgaatatgtaggatcattcgccagggtccga
gctgagaattggatgaaaaaaaagggggggtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga
tcccttttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaataaaaaaaagggggggcttatag
gtcaatcatgttcttgtgaatggatttaaaaaaaaggggggggaccgcttggcgcacccaaattcagtgtgggcgagcgcaa
cggttttggcccttgttagaggcccccgtaaaaaaaagggggggcaattatgagagagctaatctatcgcgtgcgtgttcat
aacttgagttaaaaaaaagggggggctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta
ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcataaaaaaaaggggggggaccgaaagggaag
ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttaaaaaaaaggggggga
```

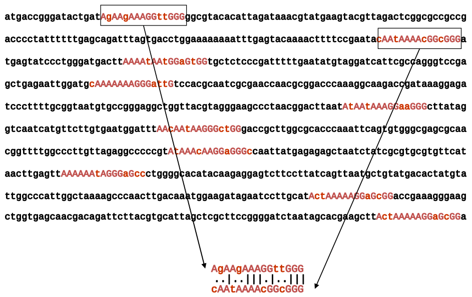
Implanting motif **AAAAAAGGGGGGG**
with four random mutations

```
atgaccgggatactgatAgAAgAAAGGttGGGggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg
acccctattttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaatacAAtAAAAcGGcGGGa
tgagtatccctgggatgacttAAAAtAAtGGaGtGGtgctctcccgatttttgaatatgtaggatcattcgccagggtccga
gctgagaattggatgcAAAAAAAGGGattGtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga
tcccttttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatAtAAtAAAGGaaGGGcttatag
gtcaatcatgttcttgtgaatggatttAAcAAtAAGGGctGGgaccgcttggcgcacccaaattcagtgtgggcgagcgcaa
cggttttggcccttgttagaggcccccgtAtAAAcAAGGaGGGccaattatgagagagctaatctatcgcgtgcgtgttcat
aacttgagttAAAAAAtAGGGaGccctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta
ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatActAAAAAGGaGcGGaccgaaagggaag
ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttActAAAAAGGaGcGGa
```

Where is the motif?

```
atgaccgggatactgatagaagaaaggttgggggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg
acccctattttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaatacaataaaacggcggga
tgagtatccctgggatgacttaaaataatggagtggtgctctcccgatttttgaatatgtaggatcattcgccagggtccga
gctgagaattggatgcaaaaaaagggattgtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga
tcccttttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatataataaaggaagggcttatag
gtcaatcatgttcttgtgaatggatttaacaataagggctgggaccgcttggcgcacccaaattcagtgtgggcgagcgcaa
cggttttggcccttgttagaggcccccgtataaacaaggagggccaattatgagagagctaatctatcgcgtgcgtgttcat
aacttgagttaaaaaatagggagccctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta
ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatactaaaaaggagcggaccgaaagggaag
ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttactaaaaaggagcgga
```

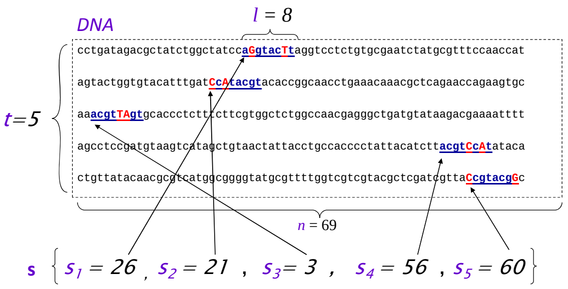
Why finding motif is difficult



```
atgaccgggatactgatAgAAgAAAGGttGGGggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg
acccctattttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaatacAAtAAAAcGGcGGGa
tgagtatccctgggatgacttAAAAtAAtGGaGtGGtgctctcccgatttttgaatatgtaggatcattcgccagggtccga
gctgagaattggatgcAAAAAAAGGGattGtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga
tcccttttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatAtAAtAAAGGaaGGGcttatag
gtcaatcatgttcttgtgaatggatttAAcAAtAAGGGctGGgaccgcttggcgcacccaaattcagtgtgggcgagcgcaa
cggttttggcccttgttagaggcccccgtAtAAAcAAGGaGGGccaattatgagagagctaatctatcgcgtgcgtgttcat
aacttgagttAAAAAAtAGGGaGccctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta
ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatActAAAAAGGaGcGGaccgaaagggaag
ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttActAAAAAGGaGcGGa
```

```
AgAAgAAAGGttGGG
 ..|..|||.|..|||
cAAtAAAAcGGcGGG
```

Parameters



```
cctgatagacgctatctggctatccaGgtacTtaggtcctctgtgcgaatctatgcgtttccaaccat
agtactggtgtacatttgatCcAtacgtacaccggcaacctgaaacaaacgctcagaaccagaagtgc
aaacgtTAgtgcaccctctttcttcgtggctctggccaacgagggctgatgtataagacgaaaatttt
agcctccgatgtaagtcatagctgtaactattacctgccacccctattacatcttacgtCcAtataca
ctgttatacaacgcgtcatggcggggtatgcgttttggtcgtcgtacgctcgatcgttaCcgtacgGc
```

s $\{ s_1 = 26\ ,\ s_2 = 21\ ,\ s_3 = 3\ ,\ s_4 = 56\ ,\ s_5 = 60 \}$

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

Score  3+4+4+5+3+4+3+4=30
```

- Line up the patterns by their start indexes

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

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

BruteForceMotifSearch

```
BruteForceMotifSearch(DNA, t, n, l)
  bestScore ← 0
  for each s=(s_1,s_2 , . . ., s_t) from (1,1 . . .,1) to (n-l+1, . . ., n-l+1)
    if (Score(s,DNA) > bestScore)
      bestScore ← Score(s, DNA)
      bestMotif ← (s_1,s_2 , . . . , s_t)
  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!

The median string problem

- Given a set of *t* DNA sequences, find a pattern that appears in all *t* sequences with the minimum number of mutations
- This pattern will be the motif

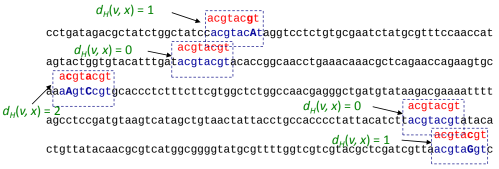
Hamming Distance

- Hamming distance:
  - $d_H(v,w)$ is the number of nucleotide pairs that do not match when *v* and *w* are aligned. For example:

d_H(AAAAAA,ACAAAC) = 2

Total Distance: Example

- Given *v* = "acgtacgt"



$d_H(v, x) = 1$
acgtacgt
cctgatagacgctatctggctatccacgtacAtaggtcctctgtgcgaatctatgcgtttccaaccat
$d_H(v, x) = 0$
acgtacgt
agtactggtgtacatttgatacgtacgtacaccggcaacctgaaacaaacgctcagaaccagaagtgc
acgtacgt
aaaAgtCcgtgcaccctctttcttcgtggctctggccaacgagggctgatgtataagacgaaaatttt
$d_H(v, x) = 2$
$d_H(v, x) = 0$
acgtacgt
agcctccgatgtaagtcatagctgtaactattacctgccacccctattacatcttacgtacgtataca
$d_H(v, x) = 1$
acgtacgt
ctgttatacaacgcgtcatggcggggtatgcgttttggtcgtcgtacgctcgatcgttaacgtaGgtc

v is the sequence in red, *x* is the sequence in blue

- TotalDistance(*v*,***DNA***) = *1+0+2+0+1* = *4*

Median string search algorithm

```
BruteForceMedianStringSearch ($\boldsymbol{DNA}$, $t$, $n$, $l$)
    $bestWord \leftarrow$ AAA…A
    $bestDistance \leftarrow \infty$
    for each l-mer $v$ from AAA…A to TTT…T
        if $TotalDistance(v,\boldsymbol{DNA}) < bestDistance$
            $bestDistance \leftarrow TotalDistance(v,\boldsymbol{DNA})$
            $bestWord \leftarrow v$
    return $bestWord$
```

Motif finding problem = median string problem

```
                        l
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   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

Score           3+4+4+5+3+4+3+4

TotalDistance   2+1+1+0+2+1+2+1

Sum             5 5 5 5 5 5 5 5
```

- At any column *i*
  $Score_i$ + $TotalDistance_i$ = *t*
- Because there are *l* columns
  Score + TotalDistance = *l* × *t*
- Rearranging:
  Score = *l* × *t* - TotalDistance
- *l* × *t* is constant, thus the minimization of TotalDistance is equivalent to the maximization of Score

Motif finding problem vs. median string problem

Why bother reformulating the *motif finding* problem into the *median string* problem?

- The motif finding problem needs to examine all the combinations for ***s***. That is $(n - l + 1)^t$ combinations
- The median string problem needs only to examine all 4^l combinations for *v*.

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 $\boldsymbol{\pi}$ is:

$$A(\boldsymbol{\pi}) \;/\; OPT(\boldsymbol{\pi})$$

where

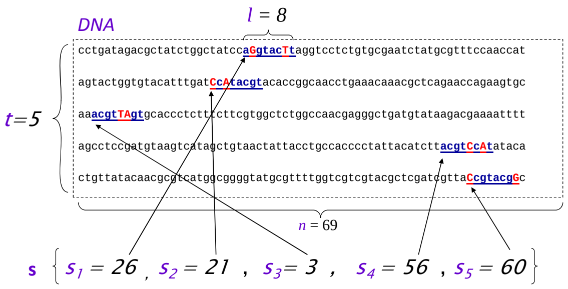
$A(\boldsymbol{\pi})$ - solution produced by algorithm A

$OPT(\boldsymbol{\pi})$ - 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):
  - $\max_{|\boldsymbol{\pi}| = n} A(\boldsymbol{\pi}) \;/\; OPT(\boldsymbol{\pi})$
- For maximization algorithms
  - $\min_{|\boldsymbol{\pi}| = n} A(\boldsymbol{\pi}) \;/\; OPT(\boldsymbol{\pi})$

Parameters



DNA

$l = 8$

$t = 5$

```
cctgatagacgctatctggctatccaGgtacTtaggtcctctgtgcgaatctatgcgtttccaaccat
agtactggtgtacatttgatCcAtacgtacaccggcaacctgaaacaaacgctcagaaccagaagtgc
aaacgtTAgtgcaccctctttcttcgtggctctggccaacgagggctgatgtataagacgaaaatttt
agcctccgatgtaagtcatagctgtaactattacctgccacccctattacatcttacgtCcAtataca
ctgttatacaacgcgtcatggcggggtatgcgttttggtcgtcgtacgctcgatcgttaCcgtacgGc
```

$n = 69$

$\mathbf{s} \; \{ s_1 = 26 \,, s_2 = 21 \;,\; s_3 = 3 \;,\; s_4 = 56 \;,\; s_5 = 60 \}$

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

Score   3+4+4+5+3+4+3+4=30
```

- Line up the patterns by their start indexes

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

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

Greedy motif finding

- Partial score: Score($\boldsymbol{s}$, i, $\boldsymbol{DNA}$)
  - 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)
    $s \leftarrow (1,1, \ldots, 1)$
    bestMotif ← s
    for $s_1 \leftarrow 1$ to $n - l + 1$
        for $s_2 \leftarrow 1$ to $n - l + 1$
            if Score(s, 2, DNA) > Score(bestMotif, 2, DNA)
                $bestMotif_1 \leftarrow s_1$
                $bestMotif_2 \leftarrow s_2$
    $s_1 \leftarrow bestMotif_1$
    $s_2 \leftarrow bestMotif_2$
    for $i \leftarrow 3$ to $t$
        for $s_i \leftarrow 1$ to $n - l + 1$
            if Score(s, i, DNA) > Score(bestMotif, i, DNA)
                $bestMotif_i \leftarrow s_i$
        $s_i \leftarrow bestMotif_i$
    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: $(n - l + 1)^t$
  - BruteForceMedianStringSearch: 4^l

Dynamic programming: Sequence alignment

DNA sequence comparison: First success story

- In 1984 Russell Doolittle and colleagues found similarities between a cancer-causing gene and a normal growth factor (PDGF) gene using a database search
- Finding sequence similarities with genes of known function is a common approach to infer the function of a newly sequenced gene

Longest common subsequence (LCS) – alignment without mismatches

i coords:	0	0	1	2	3	4	5	5	6	6	7
elements of ***v***		–	T	G	C	A	T	–	A	–	C
elements of ***w***		A	T	–	C	–	T	G	A	T	C
j coords:	0	1	2	2	3	3	4	5	6	7	8

Matches shown in red

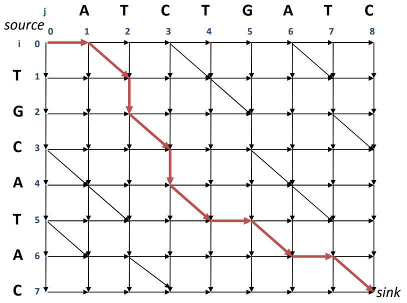
positions in ***v*** : 1 < 3 < 5 < 6 < 7

positions in ***w*** : 2 < 3 < 4 < 6 < 8

TCTAC is a common subsequence of ***v*** and ***w***

Every common subsequence is a path in a 2-D grid

Edit graph for the longest common substring (LCS) problem

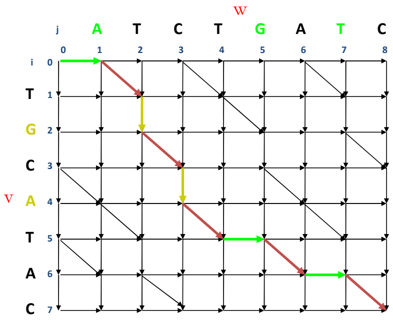


Every path from source to sink is a common subsequence (CS)

Every diagonal edge adds an extra element to the CS

LCS Problem: Find the path with the maximum number of diagonal edges

Edit graph for the LCS problem



Deletion

Matches

Insertion

```
-TGCAT-A-C
AT-C-TGATC
```

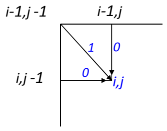
Computing LCS (I)

Let $\mathbf{v}_i = v_1 \ldots v_i$ (prefix of $\mathbf{v}$ of length i)

and $\mathbf{w}_j = w_1 \ldots w_j$ (prefix of $\mathbf{w}$ of length j)

The length of LCS($\mathbf{v}_i$,$\mathbf{w}_j$) is equal to:

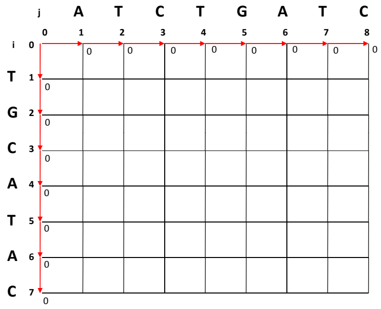
$$s_{i,j} = \max \begin{cases} s_{i-1,j} & \text{Insertion} \\ s_{i,j-1} & \text{Deletion} \\ s_{i-1,j-1} + 1 \text{ if } v_i = w_j & \text{Match} \end{cases}$$



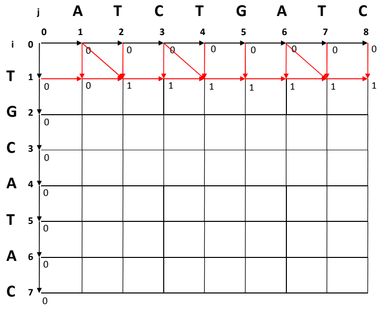
LCS algorithm

```
LCS(v, n, w, m)
for i ← 1 to n
    s_{i,0} ← 0
for j ← 1 to m
    s_{0,j} ← 0
for i ← 1 to n
    for j ← 1 to m
        s_{i,j} ← max { s_{i-1,j} ; s_{i,j-1} ; s_{i-1,j-1} + 1, if v_i = w_j }
return s_{n,m}
```

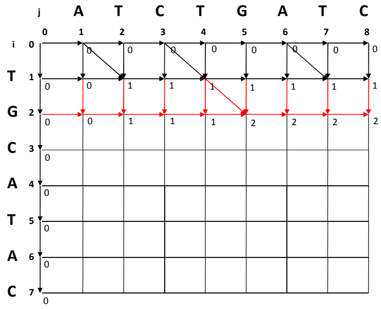
Example: initiation



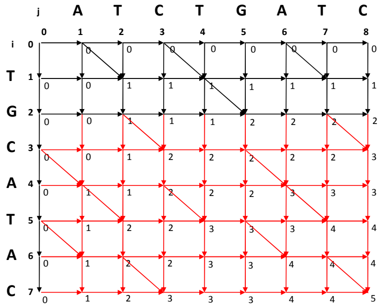
Example: For *i = 1, j = 1… m*



Example: For *i = 2, j = 1… m*



Example: For *i = 3 … n, j = 1… m*



		j	A	T	C	T	G	A	T	C
		0	1	2	3	4	5	6	7	8
i	0		0	0	0	0	0	0	0	0
T	1	0	0	1	1	1	1	1	1	1
G	2	0	0	1	1	1	2	2	2	2
C	3	0	0	1	2	2	2	2	2	3
A	4	0	1	1	2	2	2	3	3	3
T	5	0	1	2	2	3	3	3	4	4
A	6	0	1	2	2	3	3	4	4	4
C	7	0	1	2	3	3	3	4	4	5

LCS Runtime

- It takes O(nm) time to fill in the $n \times m$ dynamic programming matrix
- The pseudocode consists of a nested "**for**" loop inside of another "**for**" loop to set up a $n \times m$ matrix

What's so great about dynamic programming?

- A naive exhaustive search would have the running time O($3^{f(n,m)}$)
- An exhaustive search would recompute the same subpaths several times
- Dynamic programming takes advantage of the rich computational structure in the search space, and reuse already computed subpaths

Scoring matrix: Example

	A	R	N	K
A	5	-2	-1	-1
R	-	7	-1	3
N	-	-	7	0
K	-	-	-	6

- Notice that although **R** and **K** are different amino acids, they have a positive score
- Why? They are both positively charged amino acids and will not greatly change the function of protein

Scoring matrices and the global alignment problem

- To generalize scoring, consider a *(4+1)* × *(4+1)* scoring matrix δ
- In the case of an amino acid sequence alignment, the scoring matrix would be *(20+1)* × *(20+1)*
- The addition of *1* is to include the score for comparison of a gap character "-" (indels)

$$s_{i,j} = max \begin{cases} s_{i-1,\,j} + \delta\,(v_i,\,-) \\ s_{i,\,j-1} + \delta\,(-,\,w_j) \\ s_{i-1,\,j-1} + \delta\,(v_i,\,w_j) \end{cases}$$

Local vs. global alignment (I)

- The Global alignment problem : find the longest path between vertices (0,0) and (n,m) in the edit graph
- The Local alignment problem tries to find the longest path between arbitrary vertices (i, j) and (i', j') in the edit graph

Local vs. global alignment (II)

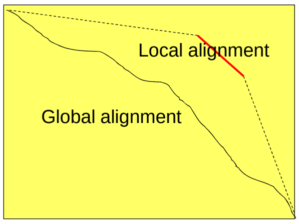
- Global Alignment

```
--T—-CC-C-AGT—-TATGT-CAGGGGACACG—A-GCATGCAGA-GAC
  |  || |  ||  | | | |||    || | | |  | ||||   |
AATTGCCGCC-GTCGT-T-TTCAG----CA-GTTATG—T-CAGAT--C
```

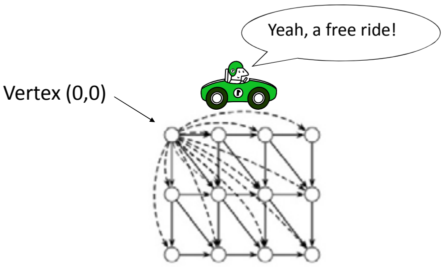
- Local Alignment—better alignment to find conserved segment

```
                  tccCAGTTATGTCAGgggacacgagcatgcagagac
                     ||||||||||||
aattgccgccgtcgttttcagCAGTTATGTCAGatc
```

Local vs. global alignment (III)



Free rides



The dashed edges represent the free rides from (0,0) to every other node.

The local alignment recurrence

- ➢ The largest value of $s_{i,j}$ over the whole edit graph is the score of the best local alignment

$$s_{i,j} = \max \begin{cases} 0 \\ s_{i-1,j} + \delta(v_i, -) \\ s_{i,j-1} + \delta(-, w_j) \\ s_{i-1,j-1} + \delta(v_i, w_j) \end{cases}$$

- ➢ The 0 is the only difference from the recurrence of the global alignment problem

Gap penalties

In nature, a series of *k* indels often come as a single event rather than a series of *k* single nucleotide events:

```
ATA– –GC     ATAG– GC
ATATTGC      AT– GTGC
```

This is more likely

Normal scoring would give the same score for both alignments

This is less likely

BLAST (I)

- Basic Local Alignment Search Tool (BLAST) finds regions of local similarity between sequences
- The program compares nucleotide or protein sequences to sequence databases and calculates the statistical significance of matches

BLAST (II)

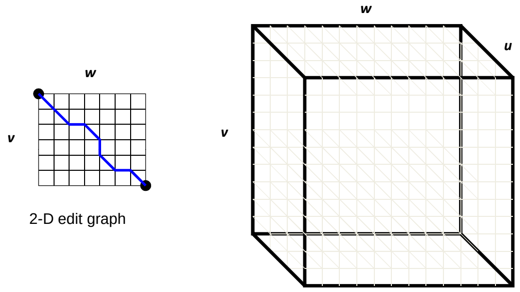
- First stage: Identify exact matches of length W (default W=3) between the query and the sequences in the database
- Second stage: Extend the match in both directions in an attempt to boost the alignment score (insertions and deletions are not considered)
- Third stage: If a high-scoring ungapped alignment is found: Perform a gapped local alignment using dynamic programming

Multiple alignment

- A faint similarity between two sequences becomes significant if present in many
- Multiple alignments can reveal subtle similarities that pairwise alignments do not reveal

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

2D vs 3D edit graph



2-D edit graph

3-D edit graph

Multiple alignment: Running time

- For two sequences of length n, the run time is $O(n^2)$
- For three sequences of length n, the run time is $O(n^3)$
- …
- For k sequences, build a k-dimensional edit graph, with run time $O(n^k)$
- Conclusion: dynamic programming approach for alignment between two sequences is easily extended to k sequences, but it is impractical due to exponential running time

Multiple alignment induces pairwise alignments

Every multiple alignment:

```
x: AC-GCGG-C
y: AC-GC-GAG
z: GCCGC-GAG
```

induces pairwise alignment:

```
x: ACGCGG-C   x: AC-GCGG-C   y: AC-GCGAG
y: ACGC-GAC   z: GCCGC-GAG   z: GCCGCGAG
```

Reverse problem: Constructing multiple alignment from pairwise alignments

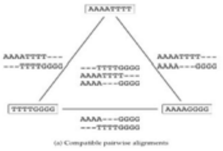
Given three pairwise alignments:

```
x: ACGCTGG-C   x: AC-GCTGG-C   y: AC-GC-GAG
y: ACGC--GAC   z: GCCGCA-GAG   z: GCCGCAGAG
```

can we construct the multiple alignment that induces them?

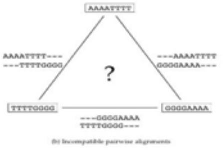
Combining optimal pairwise alignments into multiple alignment

Can combine pairwise alignments into multiple alignment



(a) Compatible pairwise alignments

Can not combine pairwise alignments into multiple alignment



(b) Incompatible pairwise alignments

Profile representation of multiple alignment

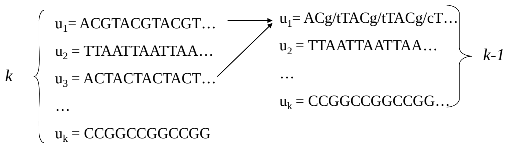
```
        -  A  G  G  C  T  A  T  C  A  C  C  T  G
        T  A  G  –  C  T  A  C  C  A  -  -  -  G
        C  A  G  –  C  T  A  C  C  A  -  -  -  G
        C  A  G  –  C  T  A  T  C  A  C  –  G  G
        C  A  G  –  C  T  A  T  C  G  C  –  G  G

A          1              1        .8
C      .6           1        .4 1     .6 .2
G            1 .2                  .2       .4  1
T      .2              1     .6             .2
-      .2       .8                    .4 .8 .4
```

PSSM: Position Specific Scoring Matrix

- In the past we were aligning a sequence against a sequence
- With profiles we can align a sequence against a profile and even a profile against a profile

Multiple alignment: Greedy approach

- Choose most similar pair of strings and combine into a profile, thereby reducing the alignment of *k* sequences to an alignment of *k-1* sequences/profiles. Repeat!
- This is a heuristic greedy method



k

u_1= ACGTACGTACGT…

u_2 = TTAATTAATTAA…

u_3 = ACTACTACTACT…

…

u_k = CCGGCCGGCCGG

u_1= ACg/tTACg/tTACg/cT…

u_2 = TTAATTAATTAA…

…

u_k = CCGGCCGGCCGG…

k-1

CLUSTALW

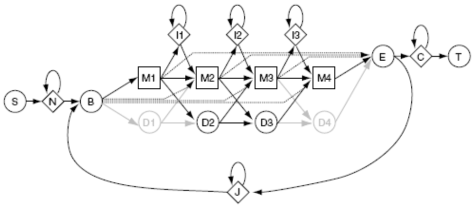
1. Determine all pairwise alignments between sequences and the degree of similarity between them.
2. Construct a similarity tree.
3. Combine the alignments from 1 in the order specified in 2 using the rule "once a gap always a gap“.

PSI-BLAST

- Position-Specific Iterative (PSI) BLAST detect weak relationships between the query and sequences in the database (higher sensitivity than BLAST)
- PSI-BLAST first constructs a multiple alignment from the highest scoring hits in a initial BLAST search and generate a profile from this alignment i.e. PSSM
- The profile is used to iteratively perform additional BLAST searches (called iterations) and the results of each iteration is used to refine the profile
- The iteration stops when no new matches with a satisfactory score are obtained

Pfam

Pfam is a set of protein families (multiple alignments) represented by Hidden Markov Models (HMMs)

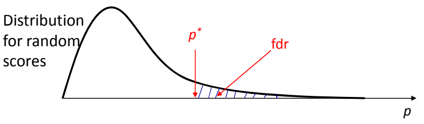


Scoring matches

Given a protein sequence ***x*** and an BLAST/PSI-BLAST/HMM, what is a significant score?

- The score for the sequence ***x***: p^*
- Generate 1000 random sequences and score them:

  $p_{rand\ 1}, p_{rand\ 2}, \dots, p_{rand\ 1000}$
- Fit a distribution to the random scores and calculate the false discover rate (fdr)
- *E-score* = *fdr* · *Size of query database* (the expected number of false positive hits)



Randomized algorithms

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

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

How Gibbs sampling works

1) Randomly choose starting positions **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, ***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:

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

T.R. Hvidsten: 1MB304: Discrete structures for bioinformatics II

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

To create a proper distribution, divide each probability prob(***a*** | ***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
$s_2=1$	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