Finding Regulatory Motifs in DNA Sequences

Micro-array experiments indicate that sets of genes are regulated by common "transcription factors (TFs)". These attach to the DNA upstream of the coding sequence, at certain binding sites. Such a site displays a short motif of DNA that is specific to a given type of TF.

Combinatorial Gene Regulation

- A microarray experiment showed that when gene X is knocked out, 20 other genes are not expressed
  - **How can one gene have such drastic effects?**

Regulatory Proteins

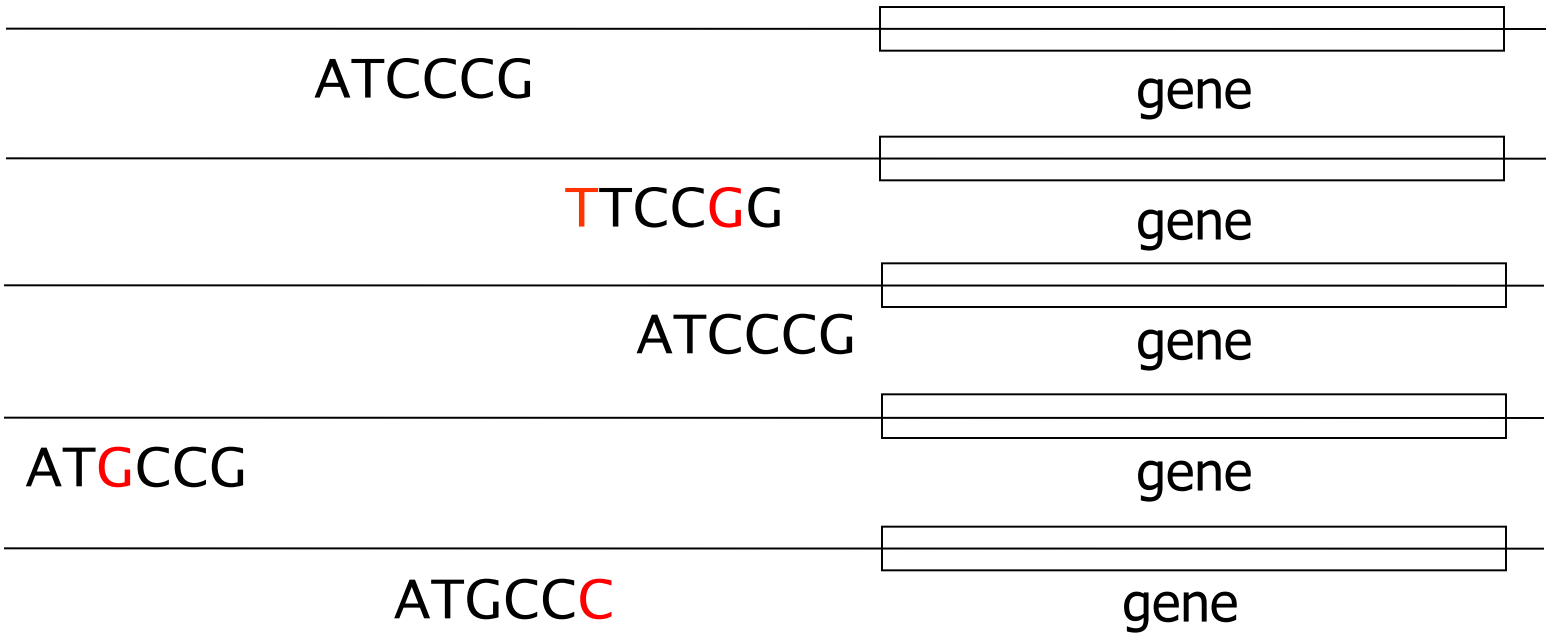
- Gene X encodes regulatory protein, a.k.a. a ***transcription factor*** (TF)
- The 20 unexpressed genes rely on gene X's TF to induce transcription
- A single TF may regulate multiple genes

Regulatory Regions

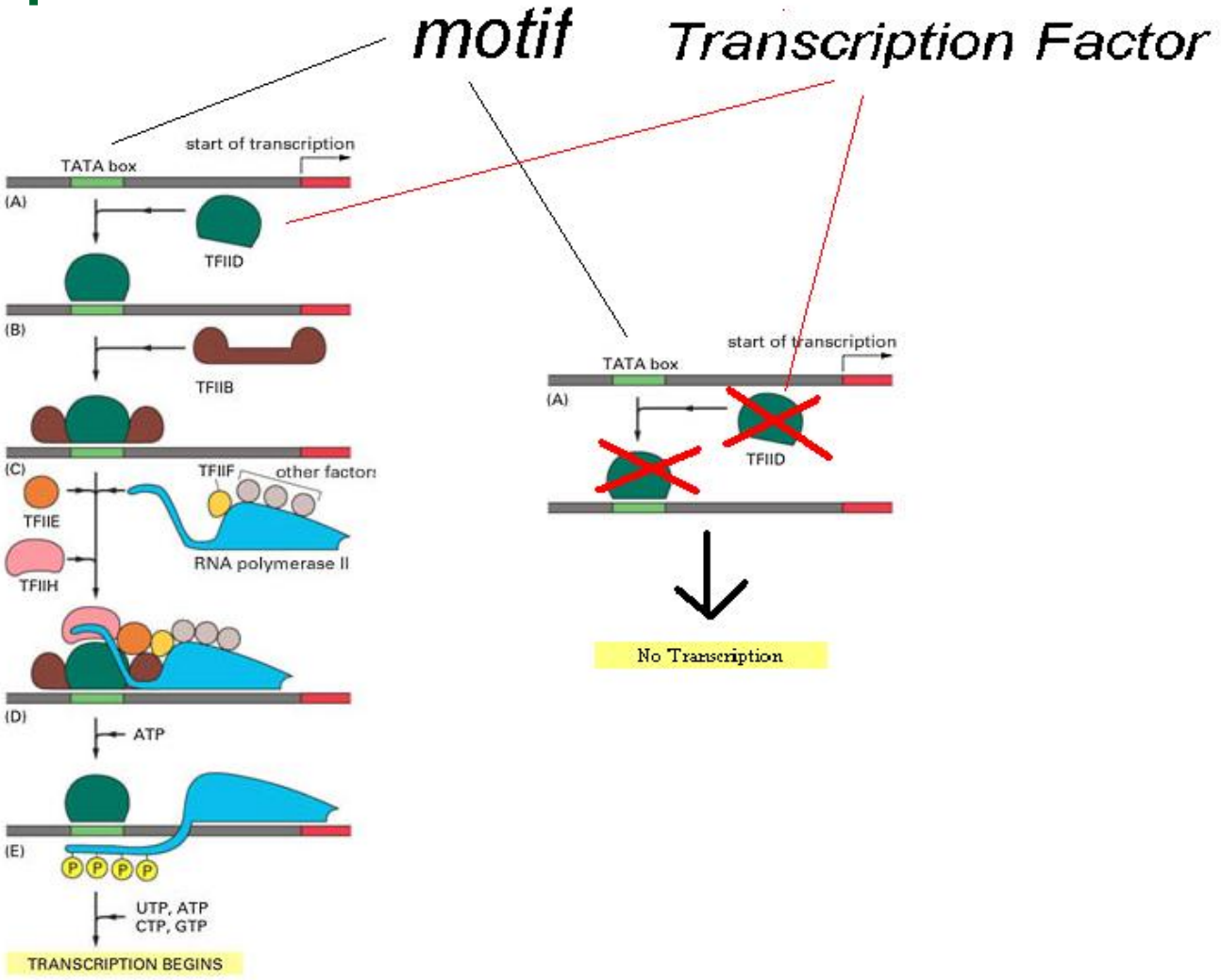
- Every gene contains a regulatory region (RR) typically stretching 100-1000 bp upstream of the transcriptional start site
- Located within the RR are the ***Transcription Factor Binding Sites*** (TFBS), also known as ***motifs***, specific for a given transcription factor
- TFs influence gene expression by binding to a specific location in the respective gene's regulatory region – TFBS
- So finding the same motif in multiple genes' regulatory regions suggests a regulatory relationship amongst those genes

Motifs and Transcriptional Start Sites

- A TFBS can be located anywhere within the Regulatory Region.
- TFBS may vary slightly across different regulatory regions since non-essential bases could mutate



Transcription Factors and Motifs



Challenge Problem
(Pevzner and Sze)

- Find a motif in a sample of
  - 20 "random" sequences (e.g. 600 nt long)
  - each sequence containing an implanted pattern of length 15,
  - each pattern appearing with 4 mismatches as (15,4)-motif.

Random Sample

10 random sequences

atgaccgggatactgataccgtatttggcctaggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg

acccctatttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaatactgggcataaggtaca

tgagtatccctgggatgacttttgggaacactatagtgctctcccgatttttgaatatgtaggatcattcgccagggtccga

gctgagaattggatgaccttgtaagtgttttccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga

tccctttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatggcccacttagtccacttatag

gtcaatcatgttcttgtgaatggatttttaactgagggcatagaccgcttggcgcacccaaattcagtgtgggcgagcgcaa

cggttttggcccttgttagaggcccccgtactgatggaaactttcaattatgagagagctaatctatcgcgtgcgtgttcat

aacttgagttggtttcgaaaatgctctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta

ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatttcaacgtatgccgaaccgaaagggaag

ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttctgggtactgatagca

Implanting Motif AAAAAAAGGGGGGG

Implanting Motif AAAAAAAGGGGGGG

Where is the Implanted Motif?

atgaccgggatactgataaaaaaaaggggggggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg

acccctatttttgagcagatttagtgacctggaaaaaaatttgagtacaaaacttttccgaataaaaaaaaggggggga

tgagtatccctgggatgacttaaaaaaaaggggggtgctctcccgattttgaatatgtaggatcattcgccagggtccga

gctgagaattggatgaaaaaaaaggggggtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga

tccctttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaataaaaaaaaggggggcttatag

gtcaatcatgttcttgtgaatggatttaaaaaaaaggggggggaccgcttggcgcacccaaattcagtgtgggcgagcgcaa

cggttttggcccttgttagaggcccccgtaaaaaaaaggggggcaattatgagagagctaatctatcgcgtgcgtgttcat

aacttgagttaaaaaaaaggggggctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta

ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcataaaaaaaaggggggggaccgaaagggaag

ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttaaaaaaaaggggggga

Implanting Motif AAAAAAGGGGGGG with Four Mutations

atgaccgggatactgatAgAAgAAAGGttGGGggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg

acccctatttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaatacAAtAAAAcGGcGGGa

tgagtatccctgggatgacttAAAAtAAtGGaGtGGtgctctcccgattttgaatatgtaggatcattcgccagggtccga

gctgagaattggatgcAAAAAAAGGGattGtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga

tccctttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatAtAAtAAAGGaaGGGcttatag

gtcaatcatgttcttgtgaatggatttAAcAAtAAGGGctGGgaccgcttggcgcacccaaattcagtgtgggcgagcgcaa

cggttttggcccttgttagaggcccccgtAtAAAcAAGGaGGGccaattatgagagagctaatctatcgcgtgcgtgttcat

aacttgagttAAAAAAtAGGGaGccctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta

ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatActAAAAAGGaGcGGaccgaaagggaag

ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttActAAAAAGGaGcGGa

Where is the Motif???

atgaccgggatactgatagaagaaaggttgggggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg

acccctatttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaatacaataaaacggcggga

tgagtatccctgggatgacttaaaataatggagtggtgctctcccgattttgaatatgtaggatcattcgccagggtccga

gctgagaattggatgcaaaaaaagggattgtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga

tccctttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatataataaaggaagggcttatag

gtcaatcatgttcttgtgaatggatttaacaataagggctgggaccgcttggcgcacccaaattcagtgtgggcgagcgcaa

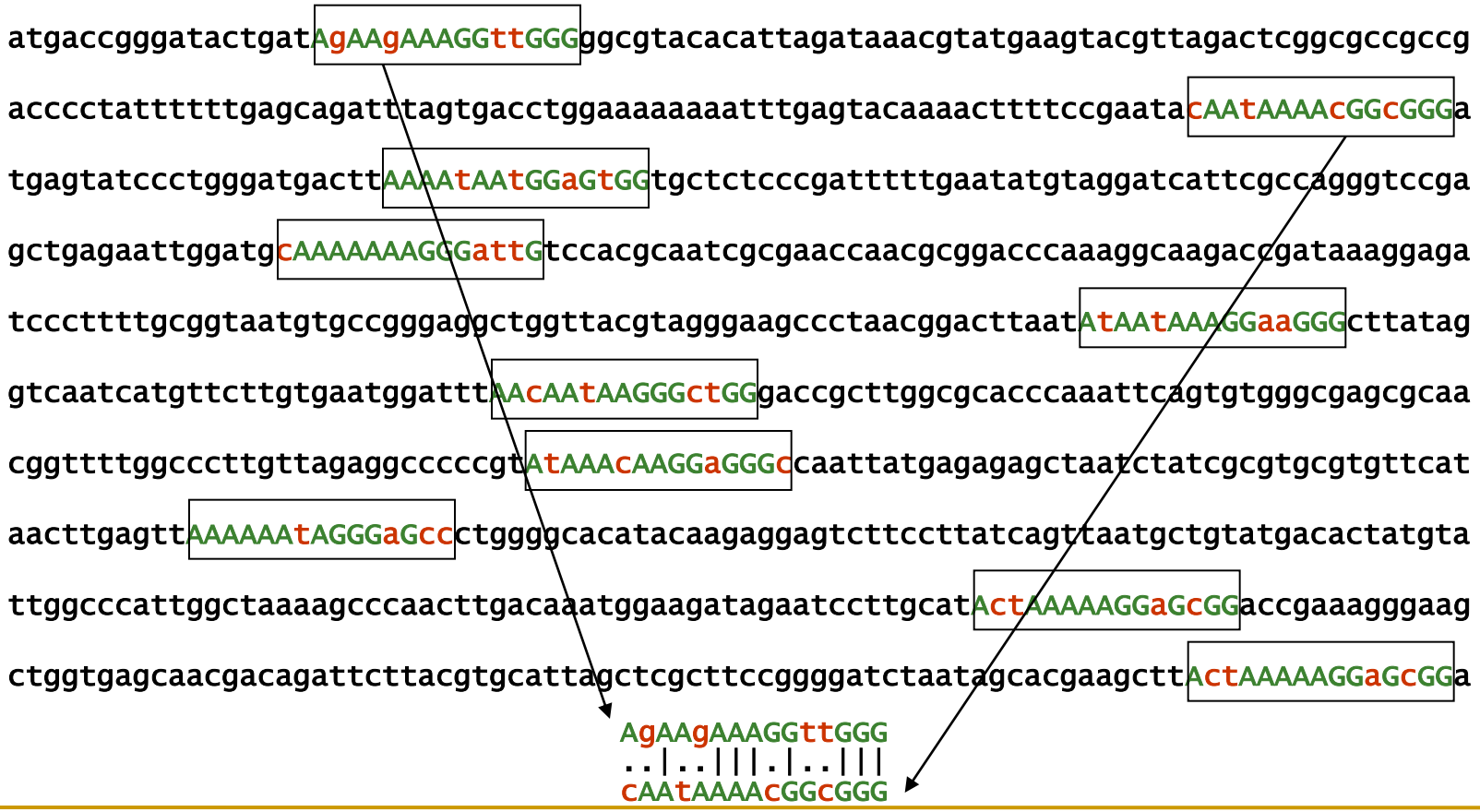
cggttttggcccttgttagaggcccccgtataaacaaggagggccaattatgagagagctaatctatcgcgtgcgtgttcat

aacttgagttaaaaaatagggagccctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta

ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatactaaaaaggagcggaccgaaagggaag

ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttactaaaaaggagcgga

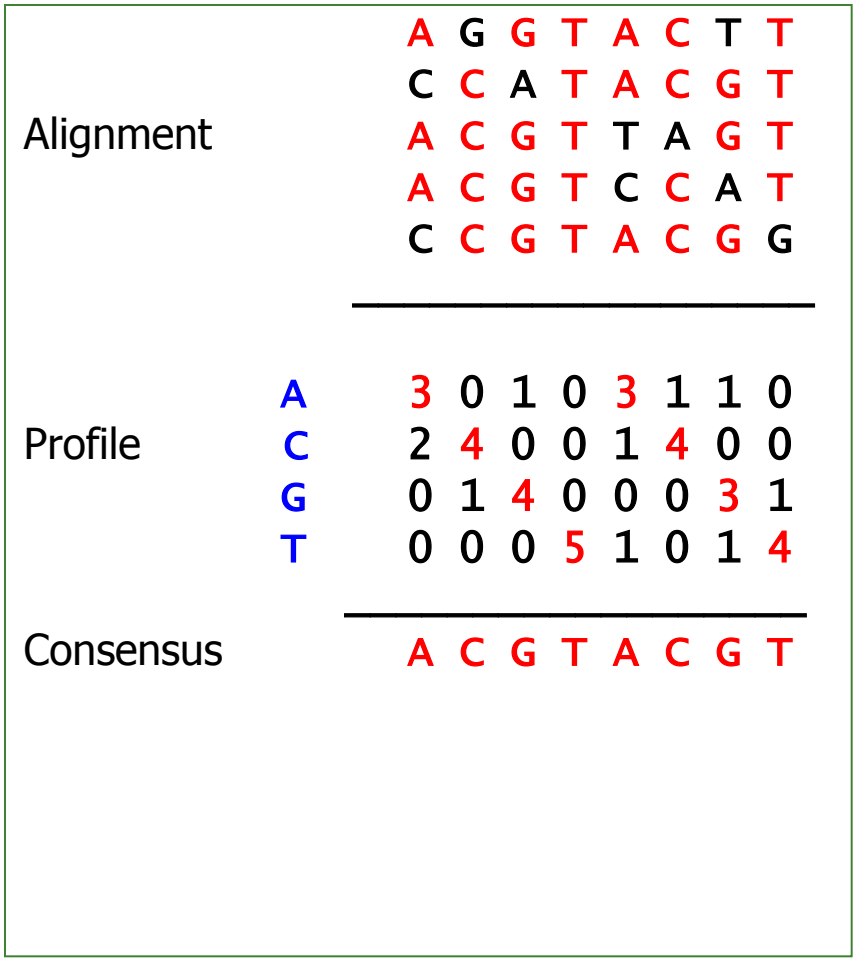
Why Finding (15,4) Motif is Difficult?



Defining Motifs

- To define a motif, let us say we know where the motif starts in the sequence
- The motif start positions in their sequences can be represented as $s = (s_1, s_2, s_3, \ldots, s_t)$

Motifs: Profiles and Consensus



- Line up the patterns by their start indexes

  $s = (s_1, s_2, s_3, \ldots, s_t)$

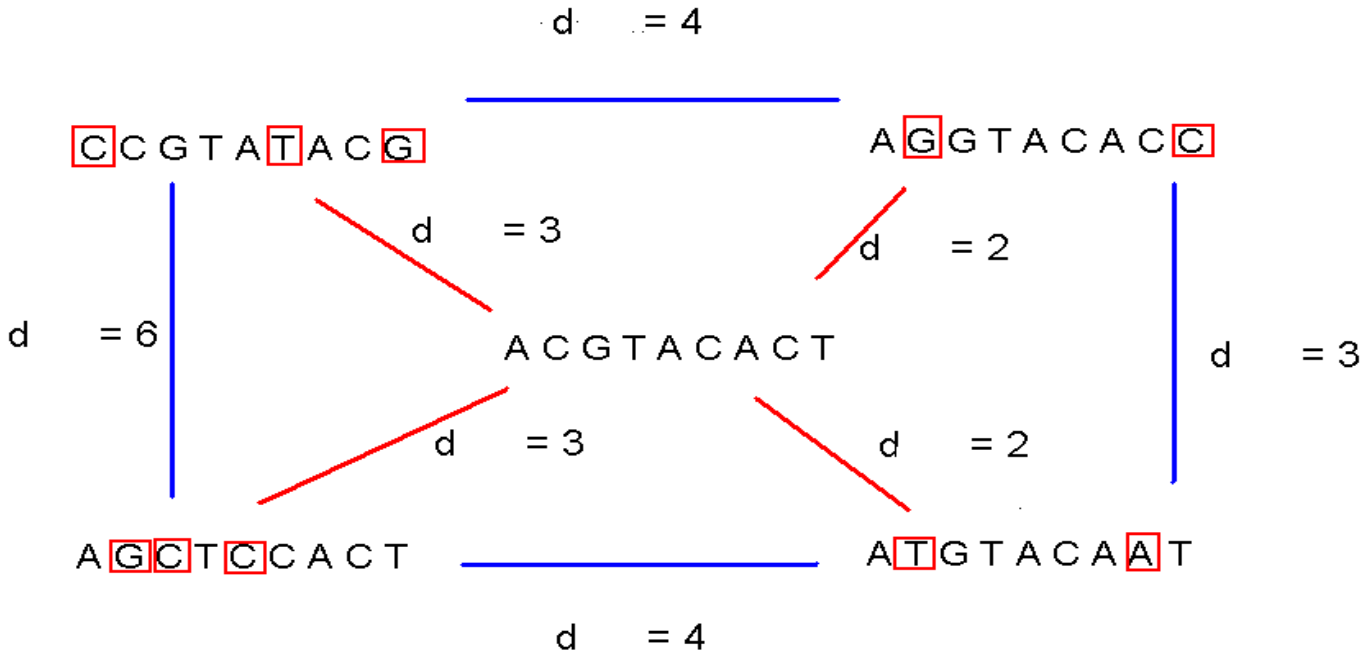
- Construct matrix profile with frequencies of each nucleotide in columns

- Consensus nucleotide in each position has the highest score in the column

Consensus

- Think of consensus as an "ancestor" motif, from which mutated motifs emerged

- The *distance* between a real motif and the consensus sequence is generally less than that for two real motifs

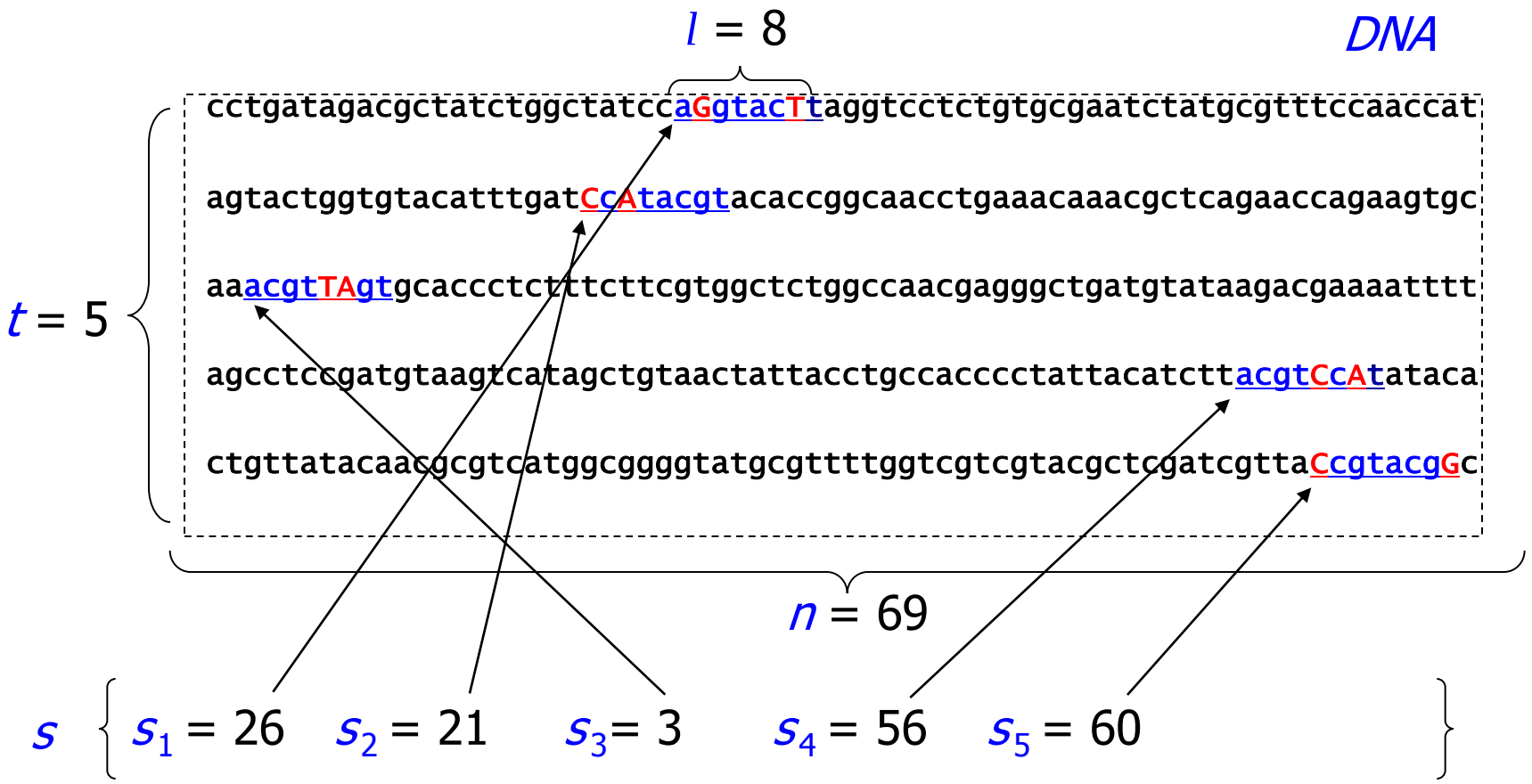


Evaluating Motifs

- We have a guess about the consensus sequence, but how "good" is this consensus?
- Need to introduce a scoring function to compare different guesses and choose the "best" one.

- t – number of sample DNA sequences
- n – length of each DNA sequence
- DNA – sample of DNA sequences (t x n array)
- l – length of the motif (l -mer)
- s_i – starting position of an l -mer in sequence i
- $s = (s_1, s_2, \ldots, s_t)$ – array of motif's starting positions

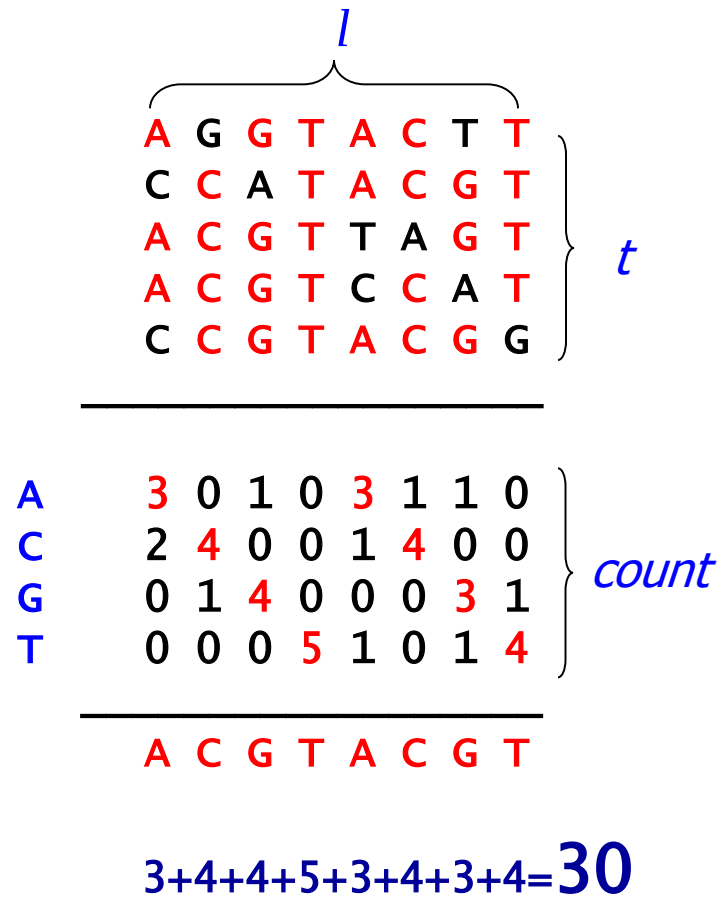
Parameters


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

- Given $s = (s_1, \ldots s_t)$ and *DNA*:

$$Score(s, DNA) = \sum_{i=1}^{l} \max_{k \in \{A,T,C,G\}} count(k,i)$$



The Motif Finding Problem

- If starting positions $s = (s_1, \ldots s_t)$ are given, finding consensus is easy even with mutations in the sequences because we can simply construct the profile to find the motif (consensus)
- But… the starting positions s are usually not given. How can we find the "best" profile matrix?

- **Goal:** Given a set of DNA sequences, find a set of l-mers, one from each sequence, that maximizes the consensus score

- **Input:** A t x n matrix of DNA, and l, the length of the pattern to find

- **Output:** An array of t starting positions $s = (s_1, \ldots s_t)$ maximizing $Score(s, DNA)$

The Motif Finding Problem: Brute Force Solution

- Compute the scores for each possible combination of starting positions s
- The best score will determine the best profile and the consensus pattern in DNA
- The goal is to maximize $Score(s, DNA)$ by varying the starting positions s_i, where:

$$s_i = [1, \ldots, n - l + 1]$$
$$i = [1, \ldots, t]$$

BruteForceMotifSearch

BruteForceMotifSearch(*DNA*, *t*, *n*, *l*)
bestScore ← 0
for each $s = (s_1, \ldots s_t)$ from (1,1 . . . 1) to (n- l +1, . . ., n- l +1)
    if (*Score* (*s*,*DNA*) > *bestScore*)
        bestScore ← *Score* (*s*, *DNA*)
        bestMotif ← $(s_1, \ldots s_t)$
return *bestMotif*

Running Time of BruteForceMotifSearch

- Varying $(n - l + 1)$ positions in each of t sequences, we are looking at $(n - l + 1)^t$ sets of starting positions
- For each set of starting positions, the scoring function makes lt operations, so complexity is
  $lt(n - l + 1)^t = O(lt\, n^t)$
- That means that for $t = 8$, $n = 1000$, $l = 10$ we must perform approximately 10^{25} computations – it will take thousands of years
- **!!! NOT USABLE !!!**

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

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

$$d_H(\text{AAAAAA}, \text{ACAAAC}) = 2$$

Total Distance: An Example

- Given v = "acgtacgt" and $s = (s_1, \ldots s_t)$

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

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

Total Distance: Definition

- For each DNA sequence DNA_i, compute all $d_H(v, x)$, where x is an l -mer with starting position s_i ($1 \leq s_i \leq n - l + 1$)
- Find minimum of $d_H(v, x)$ among all l -mers in sequence DNA_i
  $TotalDistance(v, DNA)$ is the sum of the minimum Hamming distances for each DNA sequence DNA_i
- $TotalDistance(v, DNA) = min_s\ d_H(v, s)$, where s is the set of starting positions $s_1, s_2, \ldots s_t$

The Median String Problem:
Formulation

- **Goal:** Given a set of DNA sequences, find a median string
- **Input:** A t x n matrix DNA, and l, the length of the pattern to find
- **Output:** A string v of l nucleotides that minimizes $TotalDistance(v, DNA)$ over all strings of that length

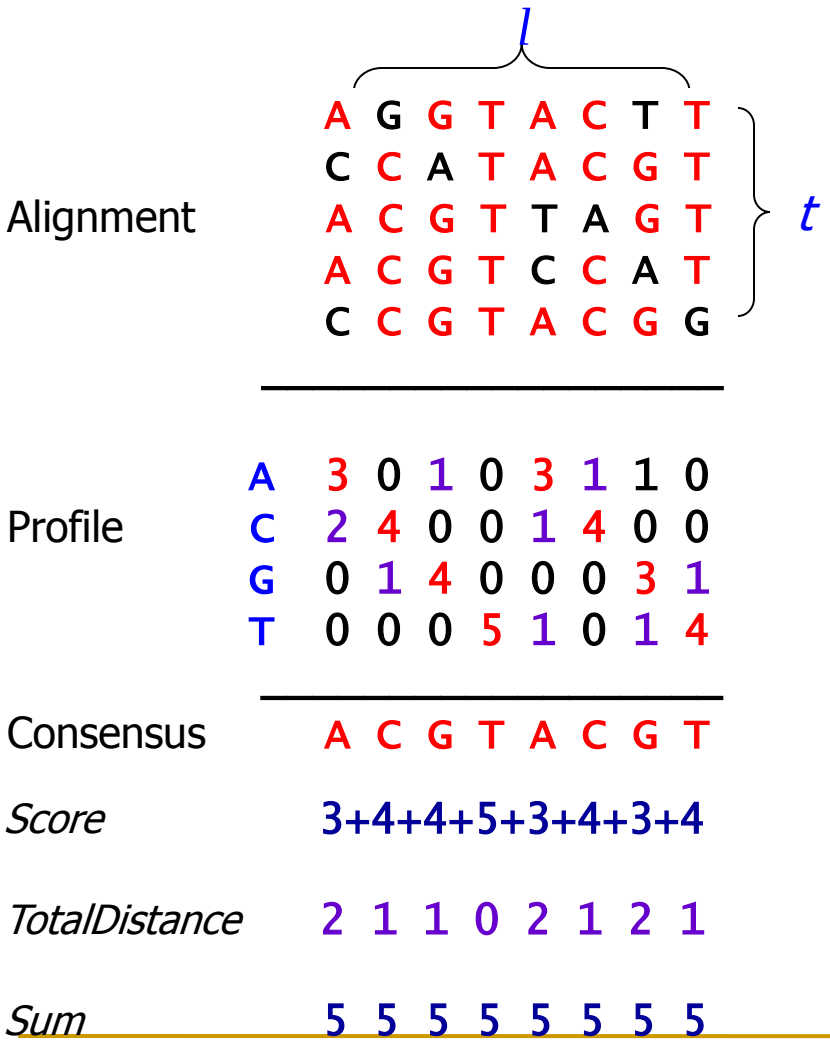
Median String Search Algorithm

```
MedianStringSearch(DNA, t, n, l )
bestWord ← AAA…A
bestDistance ← ∞
    for each l -mer s from AAA…A to TTT…T
      if TotalDistance (s,DNA ) < bestDistance
          bestDistance ← TotalDistance (s,DNA )
          bestWord ← s
  return bestWord
```

Motif Finding Problem = Median String Problem

- The *Motif Finding* is a maximization problem while *Median String* is a minimization problem
- However, the *Motif Finding* problem and *Median String* problem are computationally equivalent
- We will show that minimizing *TotalDistance* is equivalent to maximizing *Score*

We are looking for the same thing



- 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 the minimization of the right side is equivalent to the maximization of the left side

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 to examine all 4^l combinations for v. This number is relatively smaller

Motif Finding: Improving the Running Time

Recall the BruteForceMotifSearch:

```
1.  BruteForceMotifSearch(DNA, t, n, l )
2.   bestScore ← 0
3.  for each s=(s1,s2 , . . ., s t ) from (1,1 . . . 1) to (n- l +1, . . ., n- l+1)
4.        if (Score(s,DNA) > bestScore )
5.              bestScore ← Score(s, DNA )
6.              bestMotif ← (s 1,s2 , . . . , s t)
7.  return bestMotif
```

Branch-bound searching

Structuring the Search

- How can we perform the line

for each $s=(s_1, s_2, \ldots, s_t)$ from $(1,1 \ldots 1)$ to $(n-l+1, \ldots, n-l+1)$?

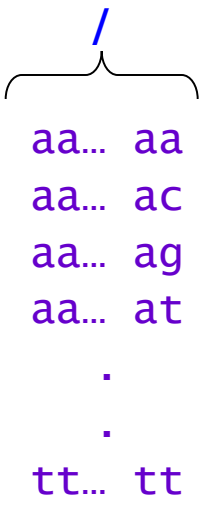
- We need a method for efficiently structuring and navigating the many possible motifs
- This is not very different than exploring all t-digit numbers

Median String: Improving the Running Time

```
1.  MedianStringSearch (DNA, t, n, l )
2.  bestWord ← AAA…A
3.  bestDistance ← ∞
4.      for each l-mer s from AAA…A to TTT…T
5.          if TotalDistance (s,DNA ) < bestDistance
6.              bestDistance ← TotalDistance(s,DNA )
7.              bestWord ← s
8.      return bestWord
```

Structuring the Search

- For the Median String Problem we need to consider all 4^l possible l -mers:

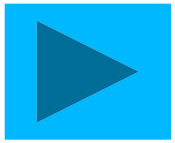


How to organize this search?

Alternative Representation of the Search Space

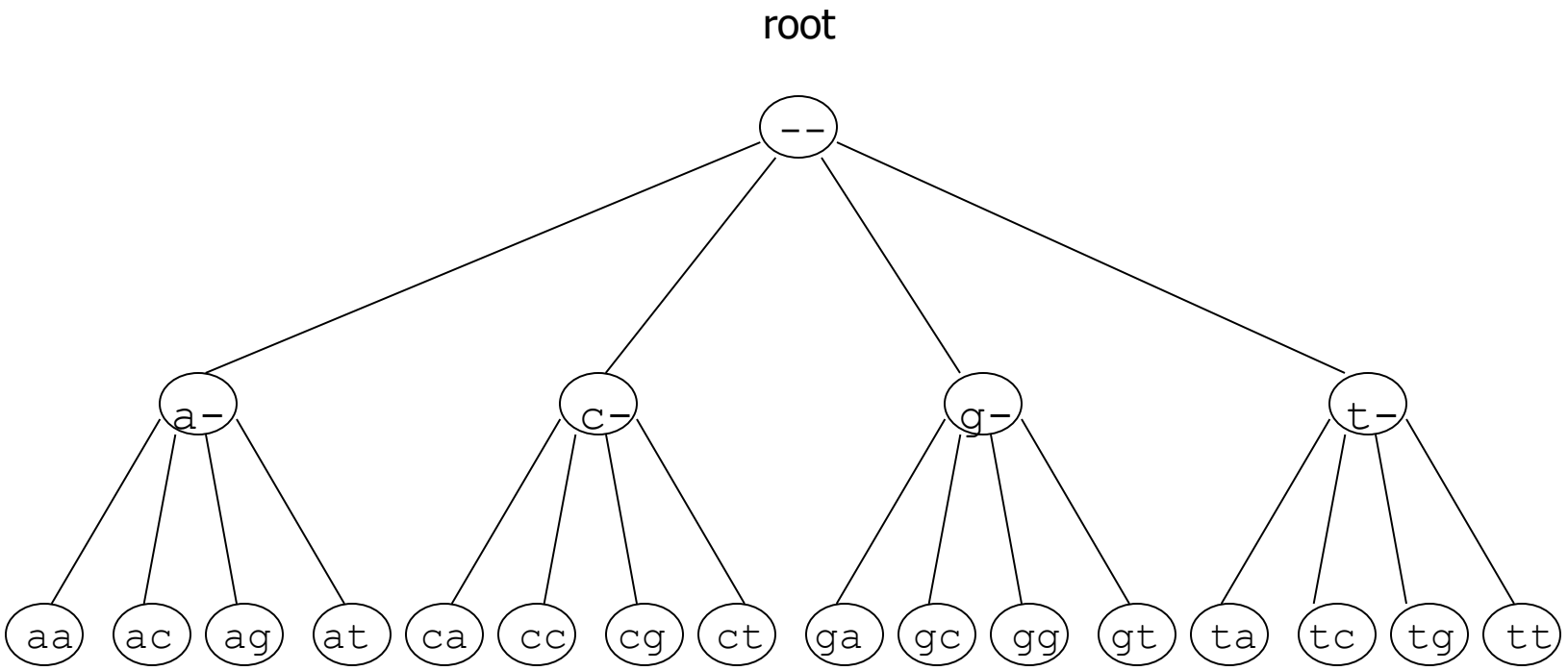
- Let A = 1, C = 2, G = 3, T = 4
- Then the sequences from AA…A to TT…T become:

$\overbrace{11\ldots11}^{l}$
11…12
11…13
11…14
.
.
44…44



- Notice that the sequences above simply list all numbers as if we were counting on base 4 without using 0 as a digit

Search Tree



Analyzing Search Trees

- Characteristics of the search trees:
  - The sequences are contained in its leaves
  - The parent of a node is the prefix of its children
- How can we move through the tree?

Moving through the Search Trees

- Four common moves in a search tree that we are about to explore:
  - Move to the next leaf
  - Visit all the leaves
  - Visit the next node
  - Bypass the children of a node

Visit the Next Leaf

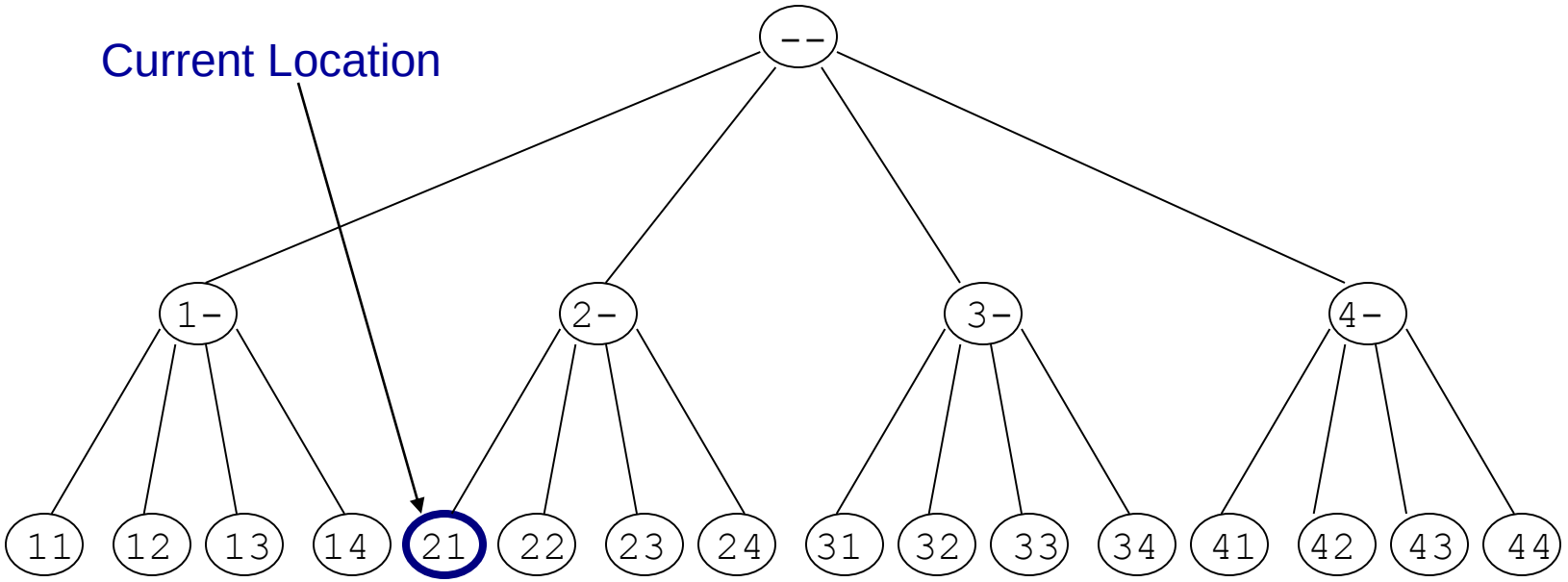
Given a current leaf *a*, we need to compute the "next" leaf:

```
1.  NextLeaf( a,L,k )          // a : the array of digits
2.  for i ← L downto 1         // L: length of the array
3.      if a_i < k             // k : max digit value
4.          a_i ← a_i + 1
5.              return a
6.          a_i ← 1
7.  return a
```

- The algorithm is common addition in radix *k:*
- Increment the least significant digit
- "Carry the one" to the next digit position when the digit is at maximal value

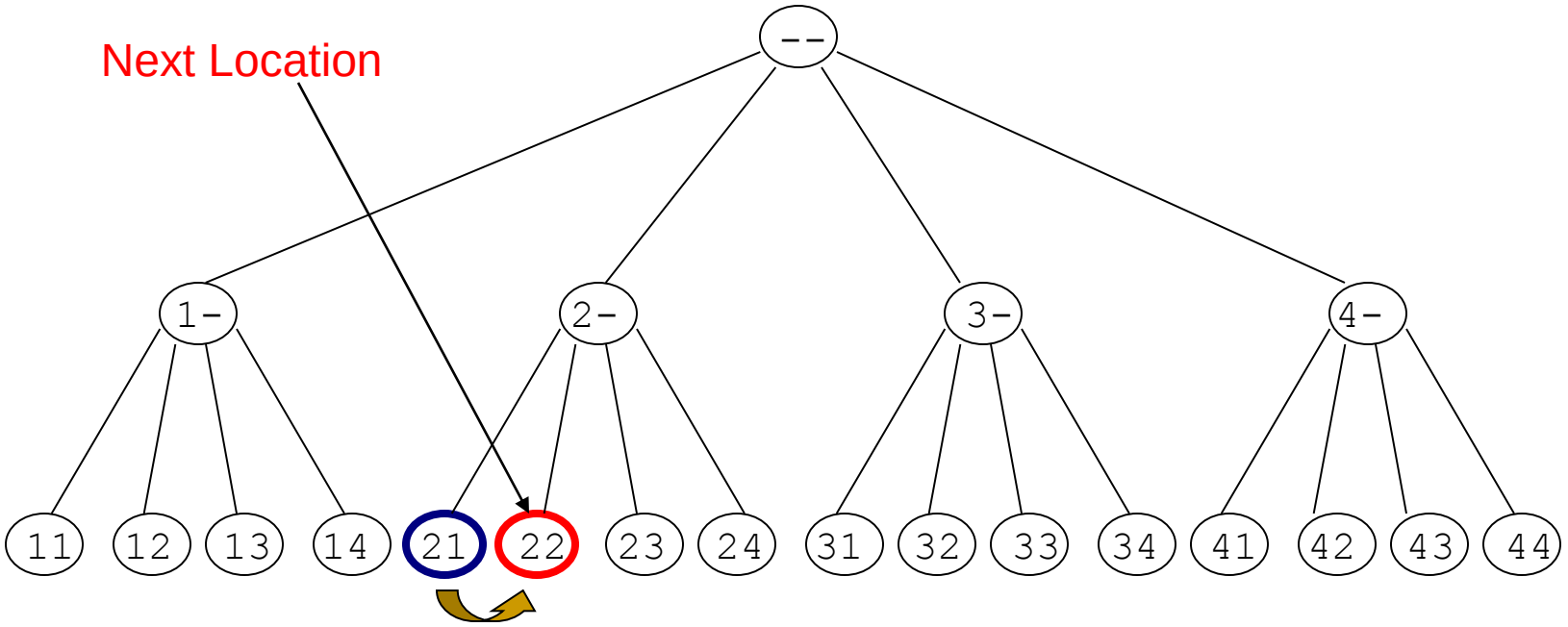
NextLeaf: Example

- Moving to the next leaf:



NextLeaf: Example (cont'd)

- Moving to the next leaf:



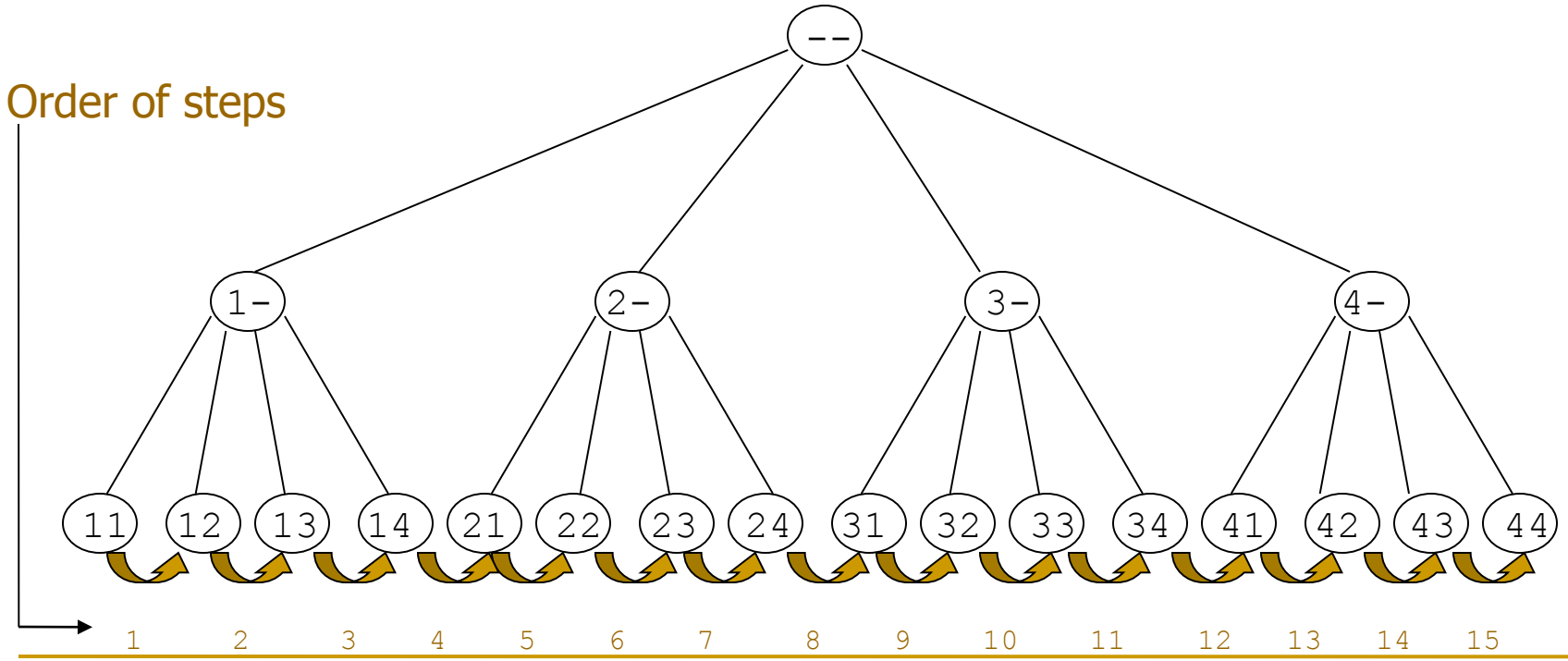
Visit All Leaves

- Printing all permutations in ascending order:

```
1.  AllLeaves(L,k)                  // L: length of the sequence
2.  a ← (1,...,1)                   // k : max digit value
3.  while forever                   // a : array of digits
4.  output a
5.  a ← NextLeaf(a,L,k)
6.  if a = (1,...,1)
7.      return
```

Visit All Leaves: Example

- Moving through all the leaves in order:



Depth First Search

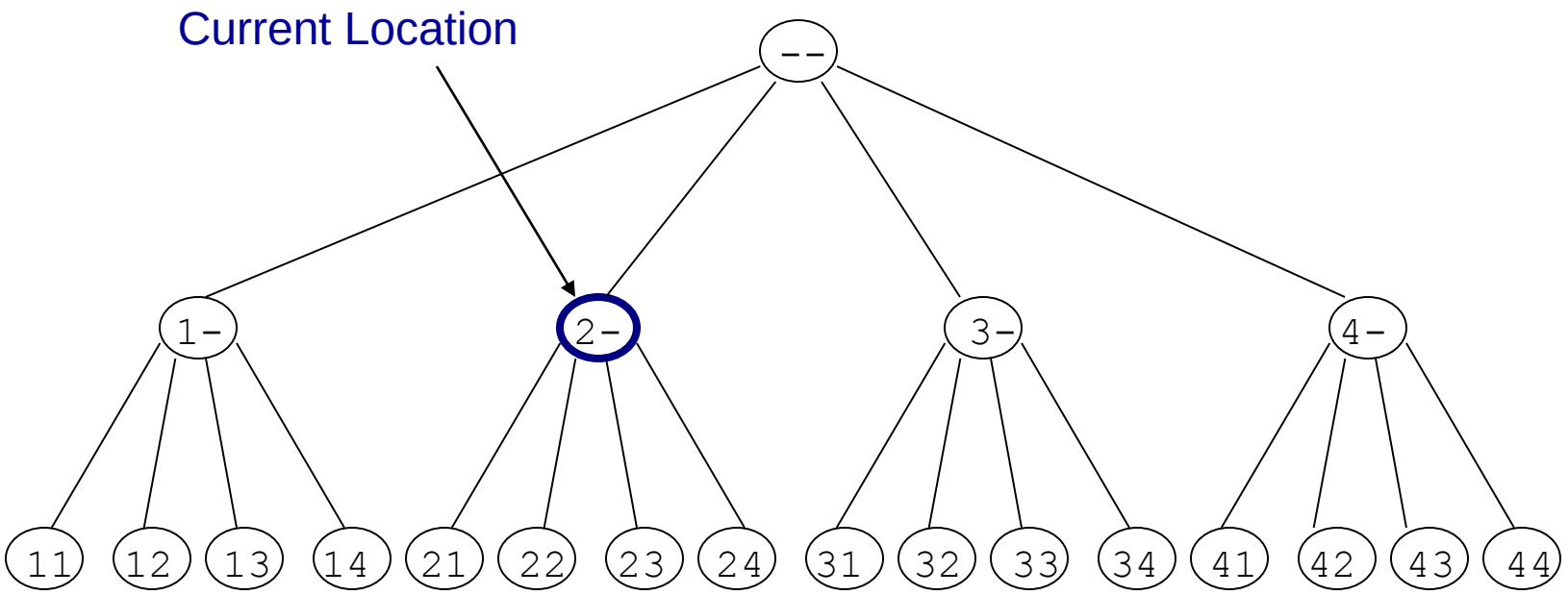
- So we can search leaves
- How about searching all vertices of the tree?
- We can do this with a *depth first* search

Visit the Next Vertex

```
1.  NextVertex(a,i,L,k)          // a : the array of digits
2.    if i < L                   // i  : prefix length
3.      a i+1 ← 1                // L: max length
4.      return ( a, i +1)        // k : max digit value
5.    else
6.      for j ← l  to 1
7.        if aj < k
8.          aj ← aj +1
9.          return( a , j )
10.   return(a, 0)
```

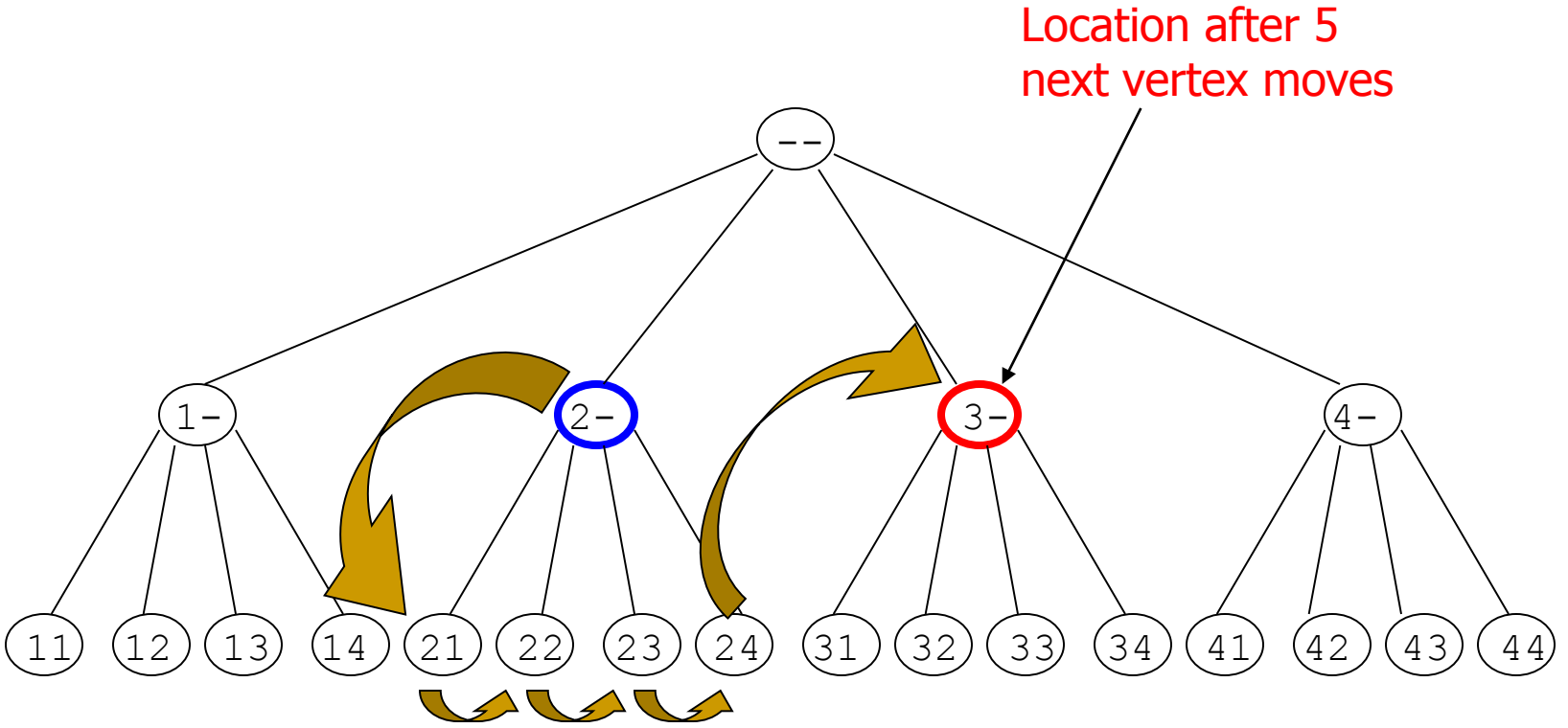
Example

- Moving to the next vertex:



Example

- Moving to the next vertices:



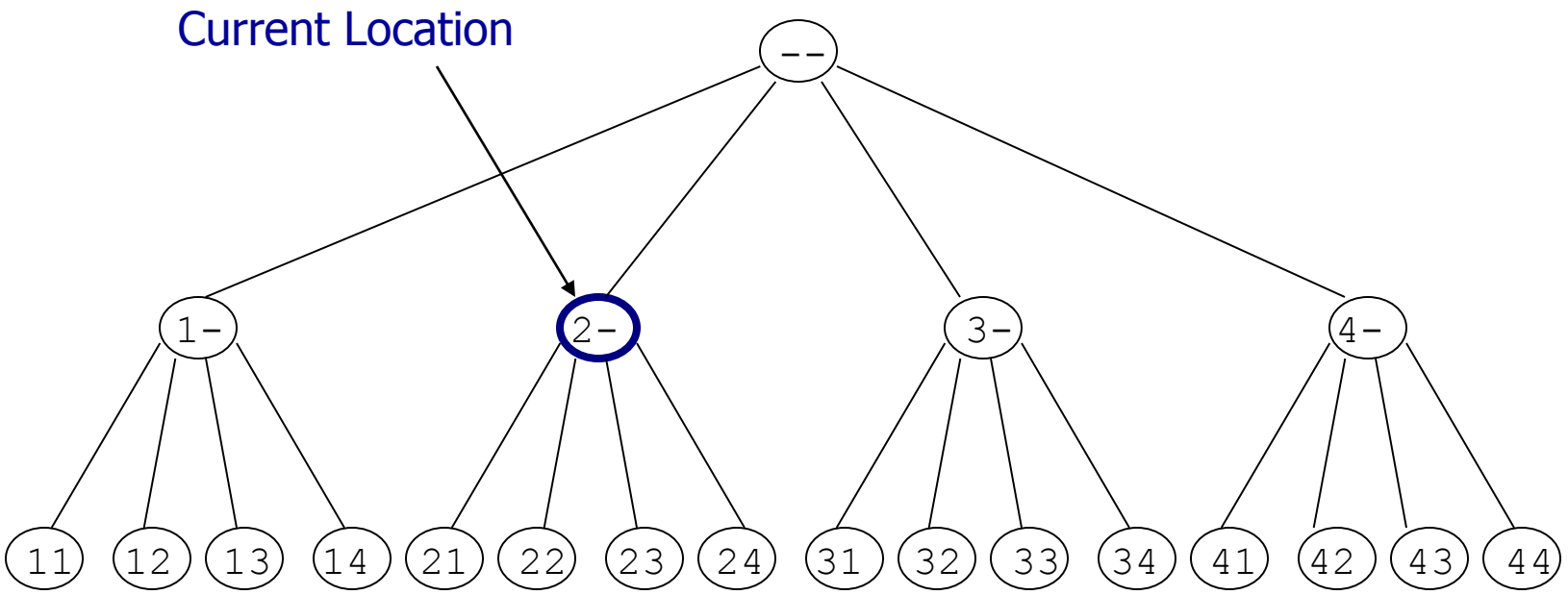
Bypass Move

- Given a prefix (internal vertex), find next vertex after skipping all its children

```
1.  Bypass (a,i,L,k)                    // a : array of digits
2.        for j ← i to 1                          // i : prefix length
3.         if a_j < to k                 // L : maximum length
4.            a_j ← a_j +1      // k : max digit value
5.            return(a,j)
6.      return(a,0)
```

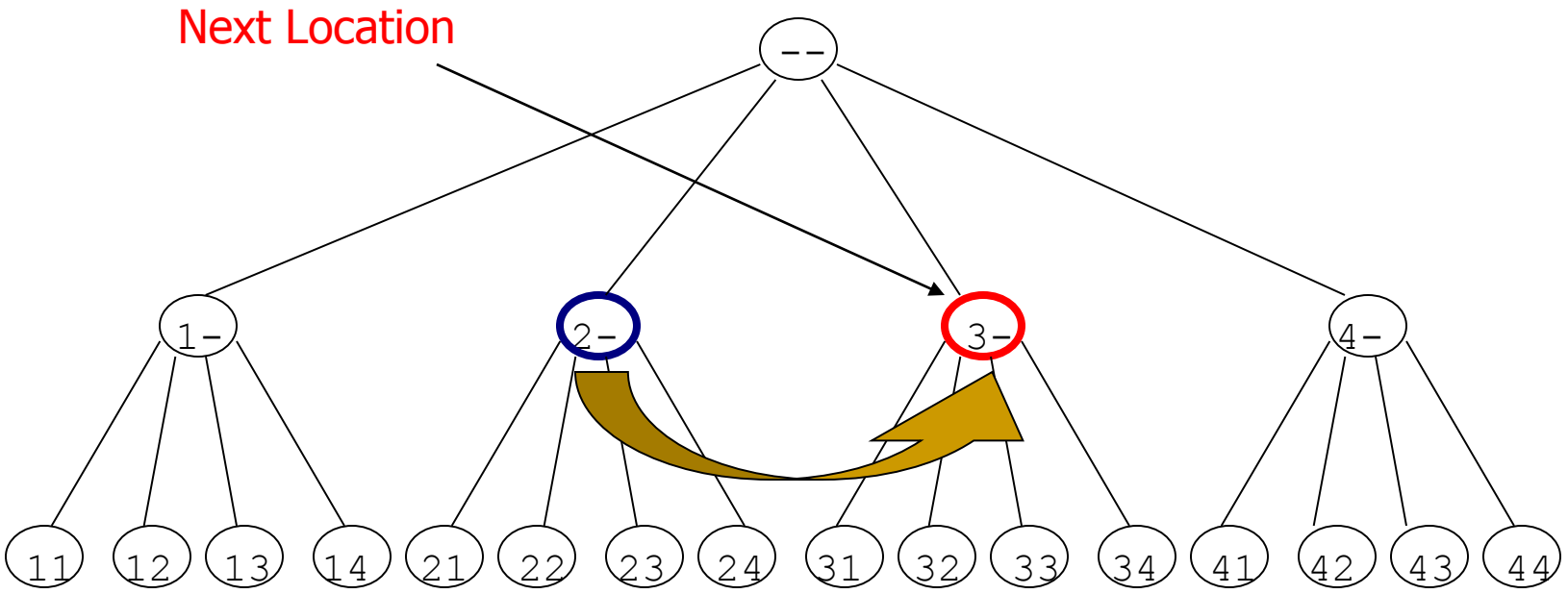
Bypass Move: Example

- Bypassing the descendants of "2-":



Example

- Bypassing the descendants of "2-":



Brute Force Search Again

```
1.  BruteForceMotifSearchAgain(DNA, t, n, l )
2.  s ← (1,1,…, 1)
3.  bestScore ← Score(s,DNA)
4.  while forever
5.          s ←  NextLeaf (s, t, n - l +1)
6.          if (Score (s,DNA ) > bestScore)
7.                  bestScore ← Score (s, DNA )
8.                  bestMotif ← (s1,s2, . . . , st)
9.  return bestMotif
```

Can We Do Better?

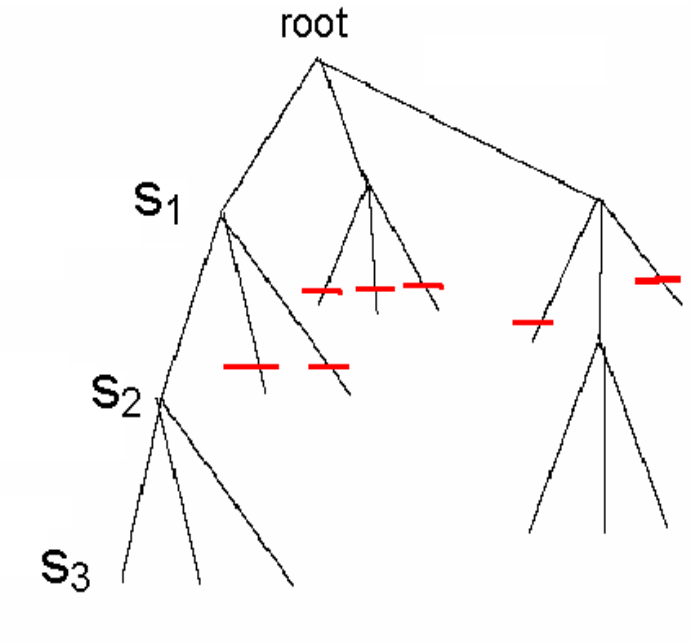
- Sets of $s = (s_1, s_2, \ldots, s_t)$ may have a weak profile for the first i positions $(s_1, s_2, \ldots, s_i)$
- Every row of alignment may add at most l to *Score*
- New notation: *Score* (s,i,DNA) denotes a partial consensus score of the $i \times l$ alignment matrix involving only the first i rows of DNA with starting positions $(s_1, s_2, \ldots, s_i)$
- Optimism: if all subsequent $(t-i)$ positions $(s_{i+1}, \ldots s_t)$ add

  $(t-i) * l$ to *Score* (s,i,DNA)
- If *Score* $(s,i,DNA) + (t-i) * l < BestScore$, it makes no sense to search in vertices of the current subtree
  - Use *ByPass* ()

Branch and Bound Algorithm for Motif Search

- Since each level of the tree goes deeper into search, discarding a prefix discards all following branches

- This saves us from looking at $(n - l + 1)^{t-i}$ leaves
  - Use **NextVertex()** and **ByPass()** to navigate the tree



Branch and Bound Motif Search

```
1.  BranchAndBoundMotifSearch(DNA,t,n,l )
2.  s ← (1,…,1)
3.  bestScore ← 0
4.  i ← 1
5.  while i > 0
6.      if i < t
7.          optimisticScore ← Score ( s, i, DNA ) +(t – i ) * l
8.          if optimisticScore < bestScore
9.            (s, i ) ← Bypass(s,i, n- l +1)          //examine next prefix
10.                                                   //of length i if possible
11.         else
12.           (s, i ) ← NextVertex(s, i, n- l+1)      //enlarge the prefix if
13.                                                   //possible
14.     else
15.         if Score(s,DNA ) > bestScore
16.            bestScore ← Score(s)
17.            bestMotif ← (s1 , s2 , s3 , …, st)
18.        (s,i ) ← NextVertex(s,i,t, n - l+ 1)
19. return bestMotif
```

Median String Search Improvements

- Recall the computational differences between motif search and median string search
  - The Motif Finding Problem needs to examine all $(n - l + 1)^t$ combinations for s.
  - The Median String Problem needs to examine 4^l combinations of v. This number is relatively small
- We want to use median string algorithm with the Branch and Bound trick!

Branch and Bound Applied to Median String Search

- Note that if the total distance for a prefix is greater than that for the best word so far:

  $TotalDistance(prefix, DNA) > BestDistance$

  there is no use exploring the remaining part of the word

- We can eliminate that branch and BYPASS exploring that branch further

Bounded Median String Search

```
1.  BranchAndBoundMedianStringSearch(DNA,t,n,l )
2.  s ← (1,…,1)
3.  bestDistance ← ∞
4.   i ← 1
5.  while i > 0
6.    if i < l
7.       prefix ←  string corresponding to the first i nucleotides of s
8.       optimisticDistance ← TotalDistance(prefix,DNA)
9.          if optimisticDistance > bestDistance
                    (s, i ) ← Bypass(s,i, l, 4 )
            //examine next prefix
                                                          //of length i  if
    possible
            else
1.                  (s, i ) ← NextVertex(s, i, l, 4)      //enlarge the
    prefix if possible
1.    else
2.          word ← nucleotide string corresponding to s
3.          if  TotalDistance(s,DNA) < bestDistance
4.                  bestDistance ← TotalDistance(word, DNA)
5.                  bestWord ← word
6.          (s,i ) ← NextVertex(s,i, l, 4)
7.  return bestWord
```

Improving the Bounds

- Given an l -mer w, divided into two parts at point i
  - u : prefix $w_1, \ldots, w_i$,
  - v : suffix $w_{i+1}, \ldots, w_l$
- Find minimum distance for u in a sequence
- No instances of u in the sequence have distance less than the minimum distance
- Note this doesn't tell us anything about whether u is part of any motif. We only get a minimum distance for prefix u

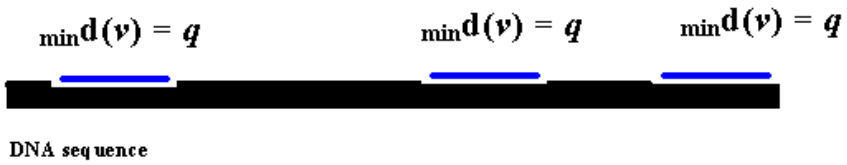
Improving the Bounds (cont'd)

- Repeating the process for the suffix v gives us a minimum distance for v

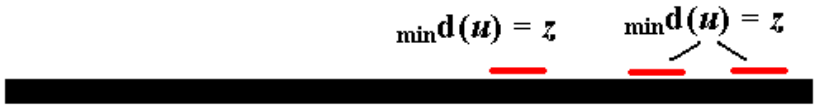
- Since u and v are two substrings of w, and included in motif w, we can assume that the minimum distance of u plus minimum distance of v can only be less than the minimum distance for w

Better Bounds

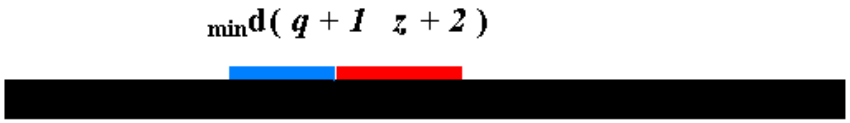
Searching for prefix *V*
We may find many instances of prefix *V* with a minimum distance *q*



Likewise for *U*



But for U and V combined, U is not at its minimum distance location, neither is V



But at least we know w (prefix *u* suffix *v*) cannot have distance *less* than $_{min}d(v) + {}_{min}d(u)$

Better Bounds (cont'd)

- If d(*prefix*) + d(*suffix*) $\geq$ *bestDistance*:
  - Motif *w* (*prefix* · *suffix*) cannot give a better (lower) score than d(*prefix*) + d(*suffix*)
  - In this case, we can ByPass()

Better Bounded Median String Search

```
1.  ImprovedBranchAndBoundMedianString(DNA,t,n,l )
2.    s = (1, 1, …, 1)
3.    bestdistance  = ∞
4.    i = 1
5.    while i > 0
6.          if i < l
7.            prefix = nucleotide string corresponding to (s1, s2, s3, …, si )
8.            optimisticPrefixDistance = TotalDistance (prefix, DNA)
9.               if (optimisticPrefixDistance < bestsubstring [ i ])
10.                 bestsubstring [ i ] = optimisticPrefixDistance
11.                 if (l - i < i )
12.                  optimisticSufxDistance = bestsubstring [l -i ]
13.                 else
14.                    optimisticSufxDistance = 0;
15.                 if optimisticPrefixDistance + optimisticSufxDistance ≥ bestDistance
16.                    (s, i ) = Bypass(s, i, l, 4)
17.                 else
18.                    (s, i ) = NextVertex(s, i, l, 4)
19.         else
20.            word = nucleotide string corresponding to (s1,s2, s3, …, st)
21.            if TotalDistance( word, DNA) < bestDistance
22.              bestDistance = TotalDistance(word, DNA)
23.              bestWord = word
24.              (s,i ) = NextVertex(s, i, l, 4)
25.   return bestWord
```

More on the Motif Problem

- Exhaustive Search and Median String are both exact algorithms
- They always find the optimal solution, though they may be too slow to perform practical tasks
- Many algorithms sacrifice optimal solution for speed

CONSENSUS: Greedy Motif Search

- Find two closest l -mers in sequences 1 and 2 and forms $2 \times l$ *alignment matrix* with *Score* (s,*2*,*DNA*)
- At each of the following $t-2$ iterations CONSENSUS finds a "best" l-mer in sequence i from the perspective of the already constructed $(i-1) \times l$ alignment matrix for the first $(i-1)$ sequences
- In other words, it finds an l-mer in sequence i maximizing

  Score (s,*i*,*DNA*)

  under the assumption that the first $(i-1)$ l -mers have been already chosen
- CONSENSUS sacrifices optimal solution for speed: in fact the bulk of the time is actually spent locating the first two l -mers

Some Motif Finding Programs

- **CONSENSUS**
  Hertz, Stromo (1989)
- **GibbsDNA**
  Lawrence et al (1993)
- **MEME**
  *Bailey, Elkan (*1995)
- **RandomProjections**
  Buhler, Tompa (2002)
- **MULTIPROFILER** *Keich, Pevzner (2002)*
- **MITRA**
  Eskin, Pevzner (2002)
- **Pattern Branching**
  Price, Pevzner (2003)

Planted Motif Challenge

- Input:
  - n sequences of length m each
  - l,d
- Output:
  - Motif M, of length l
  - Variants of interest have a hamming distance of d from M

When is the Problem Solvable?

- Assume that the background sequences are **independent and identically-distributed** (**i.i.d.**)
- the probability that a given l -mer C occurs with up to d substitutions at a given position of a random sequence is:

$$p_{(l,d)} = \sum_{i=0}^{d} \binom{l}{i} \left(\frac{3}{4}\right)^{i} \left(\frac{1}{4}\right)^{l-i}$$

- the expected number of length l motifs that occur with up to d substitutions at least once in each of the t random length n sequences is:

$$E(l,d,t,n) = 4^{l}(1-(1-p_{(l,d)})^{n-l+1})^{t}$$

- Very rough estimate – overlapping motifs not modelled, and the assumption of i.i.d. background distribution is usually incorrect.

When is the Problem Solvable?

- the expected number of length l motifs that occur with up to d substitutions at least once in each of the t random length n sequences is:

Probability that a sequence of length n contains an l -mer with Hamming distance at most d from a given l -mer

$$E(l,d,t,n) = 4^l(1-(1-p_{(l,d)})^{n-l+1})^t$$

Probability that a l -mer at a given position has Hamming distance > d from a given l -mer

When is the Problem Solvable?

- 20 random sequences of length 600 are expected to contain more than one (9, 2)-motif by chance, whereas the chances of finding a random (10, 2)-motif are less than 10^{-7}.
- So, the (9, 2) problem is impossible to solve, because "random motifs" are as likely as the planted motif. However, for the (10, 2) the probability of a random motif occurring is very small.

How to proceed?

- Exhaustive search?

- Run time is high

Heuristic search

- Searching the space of starting positions
  - Gibbs sampling
  - The Projection Algorithm
- Searching the space of motifs
  - Pattern Branching
  - Profile Branching

Notation

- t sequences $DNA_1,\ldots,DNA_t$, each of length n
- $l > 0$ integer; the goal is to find an l -mer in each of the sequences such that the „similarity" between these l -mers is maximized
- Let $(a_1, \ldots, a_t)$ be a list of l -mers contained in $DNA_1, \ldots, DNA_t$. These form a $t \times l$ alignment matrix.
- Let $X(a) = (x_{ij})$ denote the corresponding $4 \times l$ profile, where x_{ij} denotes the frequency with which we observe nucleotide i at position j. Usually, we add pseudo counts to ensure that X does not contain any zeros (Laplace correction)
  - Let $n_{a_i j}$ denote the number of occurrences of nucleotide a_i at position j, p_{a_i} denote the probability of the occurrence of nucleotide a in all $DNA_1,\ldots,DNA_t$
  - β is a weight of the correction

$$x_{a_i j} = \frac{n_{a_i j} + \beta p_{a_i}}{t + \beta}$$

Greedy profile search

- the probability that a given l -mer $z=z_1 \ldots z_l$ was generated by a given profile X

$$P(z, X) = \prod_{i=1}^{l} z_{w_i\, i}$$

- Any l -mer that is similar to the consensus string of X will have a "high" probability, while dissimilar ones will have "low" probabilities.

Greedy profile search

$$P(z, X) = \prod_{i=1}^{l} z_{w_i\, i}$$

	1	2	3	4	5	6	7	8	9
A	.33	.60	.08	0	0	.49	.71	.06	.15
C	.37	.13	.04	0	0	.03	.07	.05	.19
G	.18	.14	.81	1	0	.45	.12	.84	.20
T	.12	.13	.07	0	1	.03	.09	.05	.46

- P(CAGGTAAGT | X) = 0.02417294365920 and
- P(TCCGTCCCA | X) = 0.00000000982800

Greedy profile search

- For a given profile X and a sequence s we can find the X-most probable l -mer in s

$$z = \arg\max P(z \mid X)$$

- **Algorithm:** *"start with a random **seed** profile and then attempt to improve on it using a greedy strategy"*
  - Given sequences $DNA_1,\ldots,DNA_t$ of length n, randomly select one l -mer a_i from each sequence DNA_i and construct an initial profile X. For each sequence DNA_i, determine the X-most probable l -mer a'_i . Set X equal to the profile obtained from $a'_1,\ldots, a'_t$ and repeat.
- Does not work well
  - the number of possible seeds is huge
  - In each iteration, the greedy profile search method can change any or all t of the profile l -mers and thus will jump around in the search space.

Gibbs Sampling

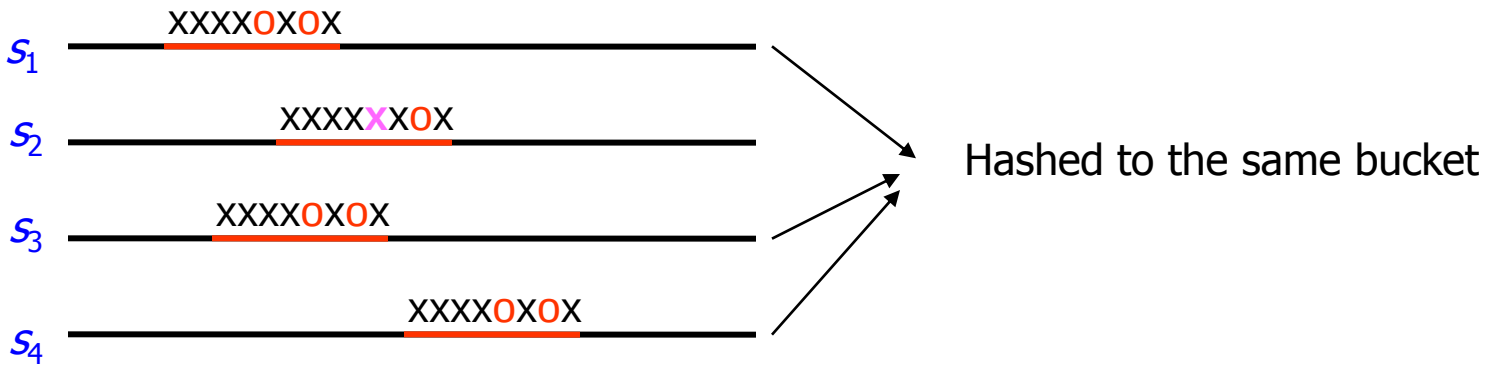
- *"start with a random seed profile, then change one l-mer per iteration."*
- Randomly select an l -mer a_i in each input sequence DNA_i.
- Randomly select one input sequence DNA_h.
- Build a $4 \times l$ profile X from $a_1,\ldots, a_{h-1}, a_{h+1},\ldots, a_t$.
- Compute background frequencies Q from input sequences $DNA_1,\ldots, DNA_{h-1}, DNA_{h+1},\ldots, DNA_t$.
- For each l -mer $a \in DNA_h$, compute $w(a) = \frac{P(a \mid X)}{P(a \mid Q)}$
- Set $a_h = a$, for some $a \in DNA_h$ chosen randomly with probability $\frac{w(a)}{\sum_{a' \in DNA_h} w(a')}$
- Repeat until "converged"

Gibbs Sampling

- often works well in practice
- difficulties:
  - finding subtle motifs
  - its performance degrades if the input sequences are skewed, that is, if some nucleotides occur much more often than others. The algorithm may be attracted to low complexity regions like AAAAAA....
- modifications:
  - use "relative entropies" rather than frequencies
  - Another modification is the use of "phase shifts": The algorithm can get trapped in local minima that are shifted up or down a few positions from the strongest pattern. To address this, in every M th iteration the algorithm tries shifting some a_i up or down a few positions

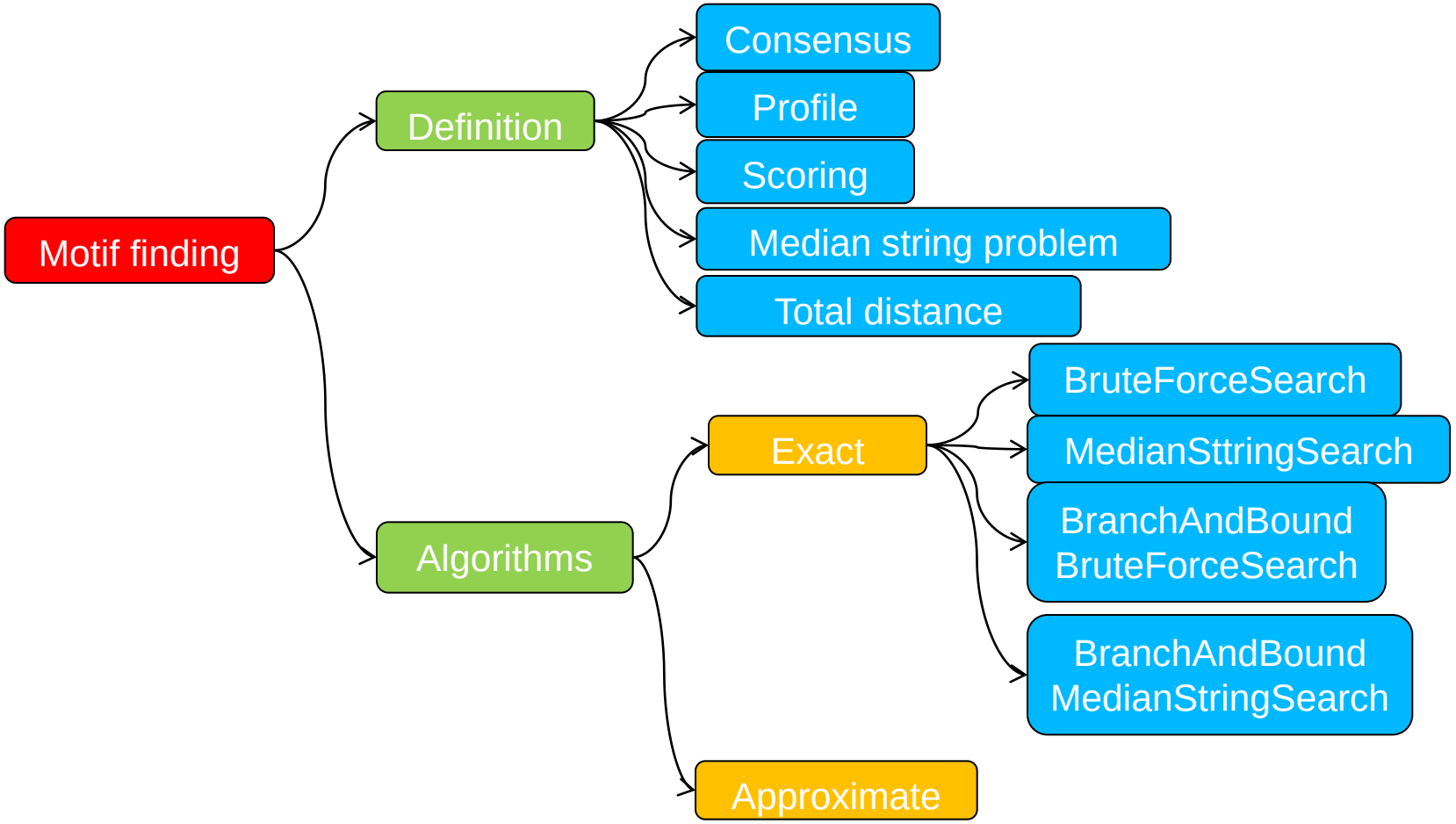
The Projection Algorithm

- *"choose k of l positions at random, then use the k selected positions of each l-mer x as a hash function $h(x)$. When a sufficient number of l-mers hash to the same bucket, it is likely to be enriched for the planted motif"*

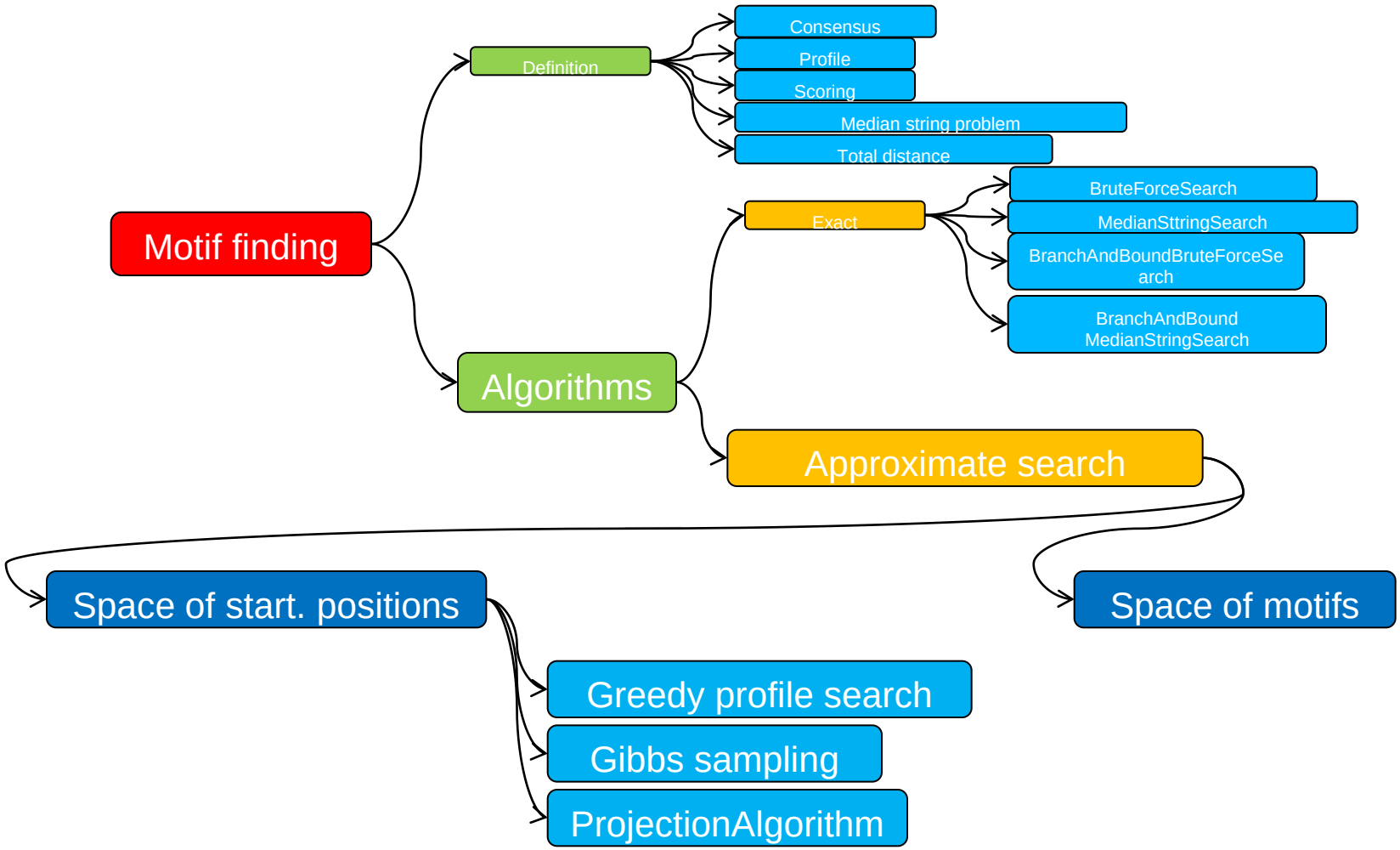


- Viewing x as a point in l -dimensional Hamming space, $h(x)$ is the projection of x onto a k-dimensional subspace.

Motif finding



Motif finding



The Projection Algorithm

- Choose distinct k of the l positions at random. For an l-mer x, the hash function $h(x)$ is obtained by concatenating the selected k residues of x.
- If M is the (unknown) motif, then we call the bucket with hash value $h(M)$ the planted bucket.
- The key idea is that, if $k < l - d$, then there is a good chance that some of the t planted instances of M will be hashed to the planted bucket, namely all planted instances for which the k hash positions and d substituted positions are disjoint.
- So, there is a good chance that the planted bucket will be enriched for the planted motif, and will contain more entries than an average bucket.

The Projection Algorithm – an example

- s_1 cagtaat
- s_2 ggaactt
- s_3 aagcaca

- and the (unknown) (3, 1)-motif M = aaa, hashing with $k = 2$ using the first 2 of $l = 3$ positions produces the following hash table:

$h(x)$	positions	$h(x)$	positions	$h(x)$	positions
aa	(1,5), (2,3), (3,1)	cg	-	ta	(1,4)
ac	(2,4), (3,5)	ct	(2,5)	tc	-
ag	(1,2), (3,2)	ga	(2,2)	tg	-
at	(1,6)	gc	(3,3)	tt	(2,6)
ca	(1,1), (3,4), (3,6)	gg	(2.1)		
cc	-	gt	(1.2)		

- The motif M is planted at positions (1, 5), (2, 3) and (3, 1) and in this example, all three instances hash to the planted bucket $h(M)$ =aa.

The Projection Algorithm – finding the planted bucket

- the algorithm does not know which bucket is the planted bucket.
- it attempts to recover the motif from every bucket that contains at least s elements, where s is a threshold that is set so as to identify buckets that look as if they may be the planted bucket.
- In other words, the first part of the Projection algorithm is a heuristic for finding promising sets of l-mers in the sequence. It must be followed by a refinement step that attempts to generate a motif from each such set.
- The algorithm has *three main parameters*:
  - the projection size k,
  - the bucket (inspection) threshold s, and
  - and the number of independent trails m.

The Projection Algorithm – projection size

- the algorithm should hash a significant number of instances of the motif into the planted bucket, while avoiding contamination of the planted bucket by random background l-mers.
- What k to choose so that the average bucket will contain less than 1 random l -mer?
- Since we are hashing $t(n - l + 1)$ l-mers into 4^k buckets, if we choose k such that $4^k > t(n - l + 1)$, then the average bucket will contain less than one random l-mer.
- For example, in the Challenge (15, 4)-problem, with $t = 20$ and $n = 600$, we must choose k to satisfy $k < l - d = 15 - 4 = 11$ and

$$k > \frac{\log(t(n-l+1))}{\log(4)} = \frac{\log(20(600-15+1))}{\log(4)} \approx 6{,}76$$

The Projection Algorithm – bucket threshold

- In the Challenge Problem, a bucket size of $s = 3$ or 4 is practical, as we should not expect too many instances to hash to the same bucket in a reasonable number of trials.
- If the total amount of sequence is very large, then it may be that one cannot choose k to satisfy both

  $k < l - d$ and $4^k > t(n - l + 1)$.

  In this case, set $k = l - d - 1$, as large as possible, and set the bucket threshold s to twice the average bucket size

  $$t(n - l + 1)/4^k.$$

The Projection Algorithm

– Number of independent trials

- Our goal: to choose m so that the probability is at least $q = 0.95$ that the planted bucket contains s or more planted motif instances in at least one of the m trials.
- let $p'(l, d, k)$ be the probability that a given planted motif instance hashes to the planted bucket, that is:

$$p'(l,d,k) = \frac{\binom{l-d}{k}}{\binom{l}{k}}$$

- Then the probability that fewer than s planted instances hash to the planted bucket in a given trial is $B_{t,p'(l,d,k)}(s)$. Here, $B_{t,p'(s)}$ is the probability that there are fewer than s successes in t independent Bernoulli trials, each trial having probability p of success (binomial probability distribution function).

The Projection Algorithm

– Number of independent trails

- Binomial distribution

$$B_{t,p}(s) = \sum_{i=0}^{s} \binom{t}{i} p^i (1-p)^{t-i}$$

- If the algorithm is run for m trails, the probability that s or more planted instances hash to the planted bucket in at least one trial is:

$$1 - (B_{t,p'(l,d,k)}(s))^m \geq q.$$

- To satisfy this equation, choose:

$$m = \left\lceil \frac{\log(1-q)}{\log\left(B_{t,p'(l,d,k)}(s)\right)} \right\rceil$$

- Using this criterion for m, the choices for k and s above require at most thousands of trails, and usually many fewer, to produce a bucket containing sufficiently many instances of the planted motif.

Projection Algorithm

1. Choose k of the l positions at random
2. Hash all l -mers of the given sequences into buckets
3. Inspect all buckets with more than s positions and refine the found motifs
4. Repeat m times, return the motif with the best score

Motif refinement

- we have already found k of the planted motif residues. These, together with the remaining $l - k$ residues, should provide a strong signal that makes it easy to obtain the motif in only a few iterations of refinement.
- We will process each bucket of size s to obtain a candidate motif. Each of these candidates will be "refined" and the best refinement will be returned as the final solution.
- Candidate motifs are refined using the expectation maximization (EM) algorithm. This is based on the following probabilistic model:
  - An instance of some length-l motif occurs exactly once per input sequence.
  - Instances are generated from a $4 \times l$ weight matrix model W, whose (i, j)-th entry gives the probability that base i occurs in position j of an instance, independent of its other positions.
  - The remaining $n-l$ residues in each sequence are chosen randomly and independently according to some background distribution.

Motif refinement
- expectation maximization

- Based on the following probabilistic model:
  - An instance of some length-l motif occurs exactly once per input sequence.
  - Instances are generated from a $4 \times l$ weight matrix model W, whose (i, j)-th entry gives the probability that base i occurs in position j of an instance, independent of its other positions.
  - The remaining $n - l$ residues in each sequence are chosen randomly and independently according to some background distribution.
- Let S be a set of t input sequences, and let P be the background distribution. EM-based refinement seeks a weight matrix model W that maximizes the likelihood ratio

$$\frac{\Pr(S \mid W^*, P)}{\Pr(S \mid P)}$$

that is, a motif model that explains the input sequences much better than P alone.

Motif refinement

- The position at which the motif occurs in each sequence is not fixed a priori, making the computation of W^* difficult, because $Pr(S \mid W, P)$ must be summed over all possible locations of the instances.
- To address this, the EM algorithm uses an iterative calculation that, given an initial guess W_0 at the motif model, converges linearly to a locally maximum-likelihood model in the neighbourhood of W_0.
- An initial guess W_h for a bucket h is formed as follows: set $W_h(i, j)$ to the frequency of base i among the j-th positions of all l-mers in h.
- This guess forms a centroid for h, in the sense that positions that are well conserved in h are strongly biased in W_h, while poorly conserved positions are less biased. To avoid zero entries in W_h, add a Laplace correction of b_i, to $W_h(i, j)$, where b_i is the background probability of residue i in the input.

Motif refinement

- Once we have used the EM algorithm to obtain a refinement W_h^* of W_h, the final step is to identify the planted motif from W_h^*. *(Details of EM skipped.)*
- To do so, we select from each input sequence the l-mer x with the largest likelihood ratio:

$$\frac{\Pr(x \mid W_h^*)}{\Pr(x \mid P)}$$

- The resulting multiset T of l-mers represents the motif in the input that is most consistent with W_h^*.
- Let C_T be the consensus of T, and let $s(T)$ be the number of elements of T whose Hamming distance to C_T is $\leq d$. The algorithm returns the sequence C_T^* that minimizes $s(T)$, over all considered buckets h and over all trials.

Summary of Projection Algorithm

Input: sequences $s_1, \ldots, s_t$, parameters k, s and m

Output: best guess motif

for $i = 1$ to m do

choose k different positions $I_k \subset \{1, 2, \ldots, l\}$

for each l-mer $x \in s_1, \ldots, s_t$ do

compute hash value $h_{Ik}(x)$

Store x in hash bucket

for each bucket with $\geq s$ elements do

refine bucket using EM algorithm

return consensus pattern of the best refined bucket

Performance of Projection Algorithm

- The performance of PROJECTION compared to other motif finders on the (l, d)-motif problem. The measure is the average performance defined as $|K \cap P| / |K \cup P|$ where K is the set of the lt residue positions of the planted motif instances, and P is the corresponding set of positions predicted by the algorithm.

l	d	Gibbs	WINNOWER	SP-STAR	PROJECTION
10	2	0.20	0.78	0.56	0.82
11	2	0.68	0.90	0.94	0.91
12	3	0.03	0.75	0.33	0.81
13	3	0.60	0.92	0.92	0.92
14	4	0.02	0.02	0.20	0.77
15	4	0.19	0.92	0.73	0.93
16	5	0.02	0.03	0.04	0.70
17	5	0.28	0.03	0.69	0.93
18	6	0.03	0.03	0.03	0.74
19	6	0.05	0.03	0.40	0.96

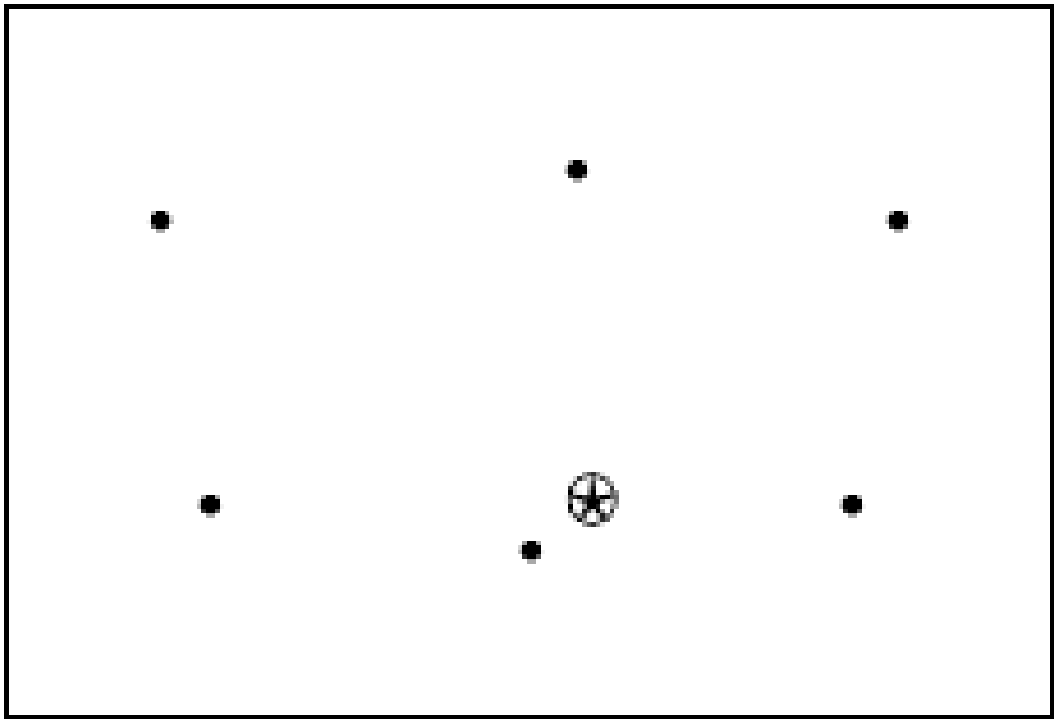
Pattern Branching Algorithm

- let M be an unknown motif of length l, and let A_0 be an occurrence of M in the sample with exactly k substitutions.
- Given A_0, how do we determine M? Since the Hamming distance $d(M,A_0) = k$, we have $M \in D_{=k}(A_0)$, defined as the set of patterns of distance exactly k from A_0.
- We could look at all

$$\binom{l}{k} 3^k$$

elements of $D_{=k}(A_0)$ and score each pattern as a guess of M. However, as this must be applied to all sample strings A_0 of length l, it would be too slow.

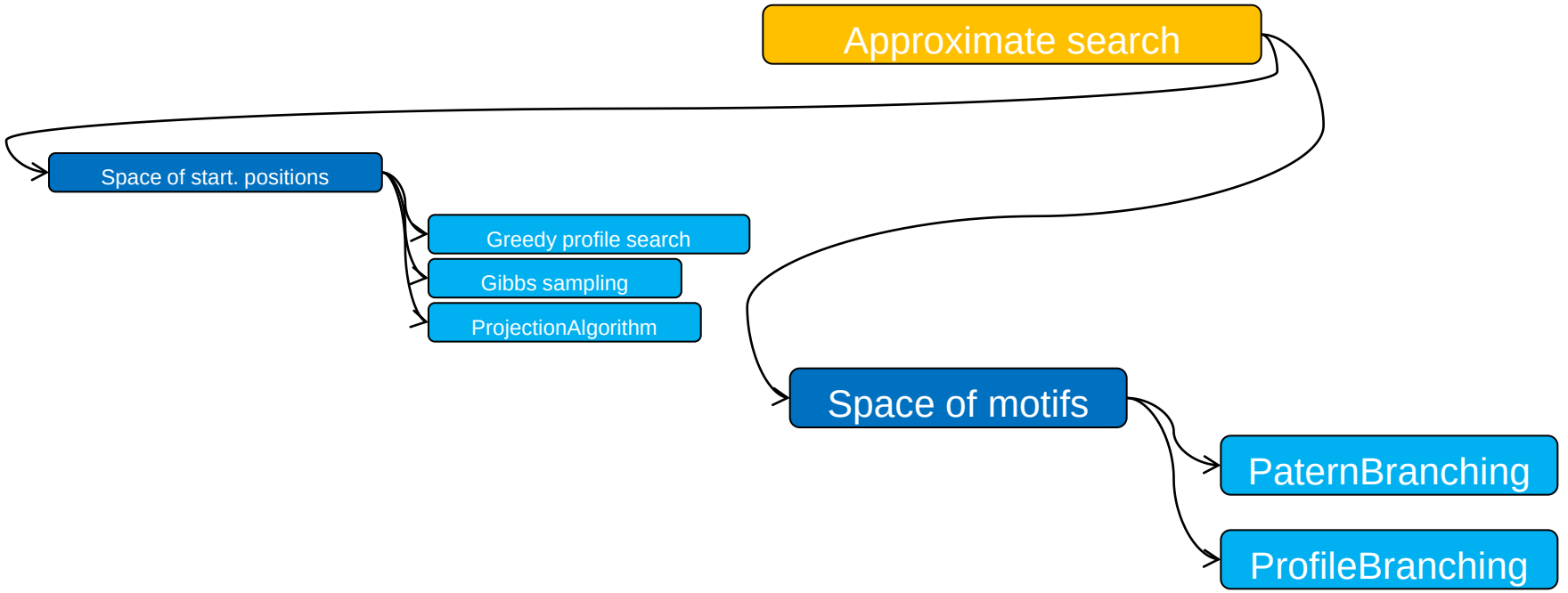
How to search motif space?



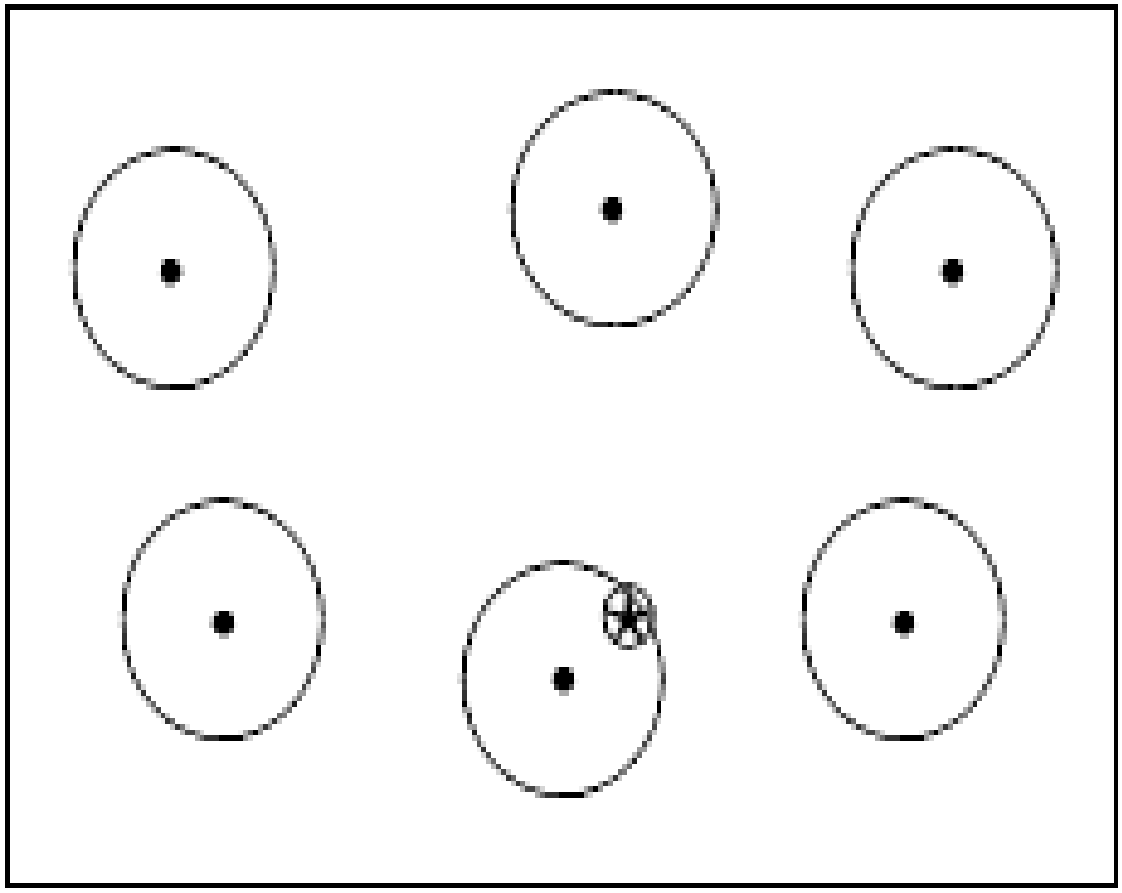
Start from random sample strings (A_0)

Search motif space for the star

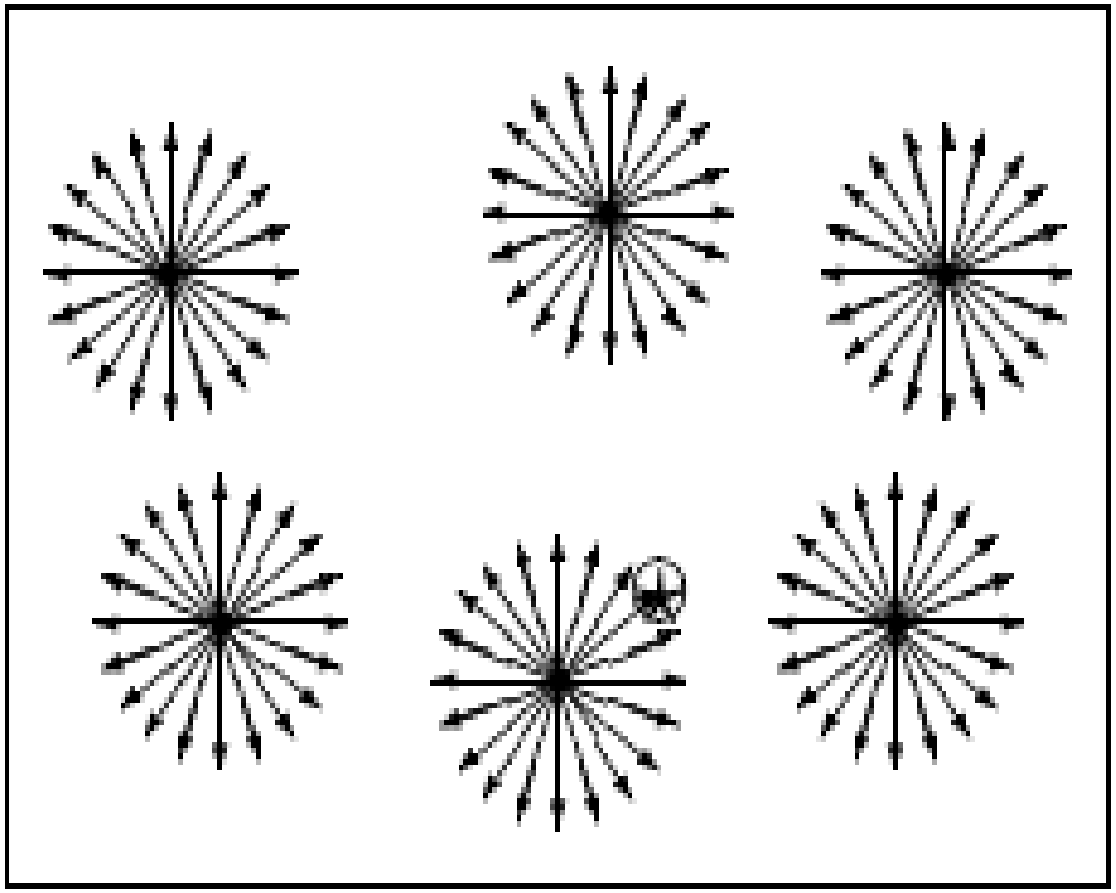
Motif finding



Search small neighborhoods

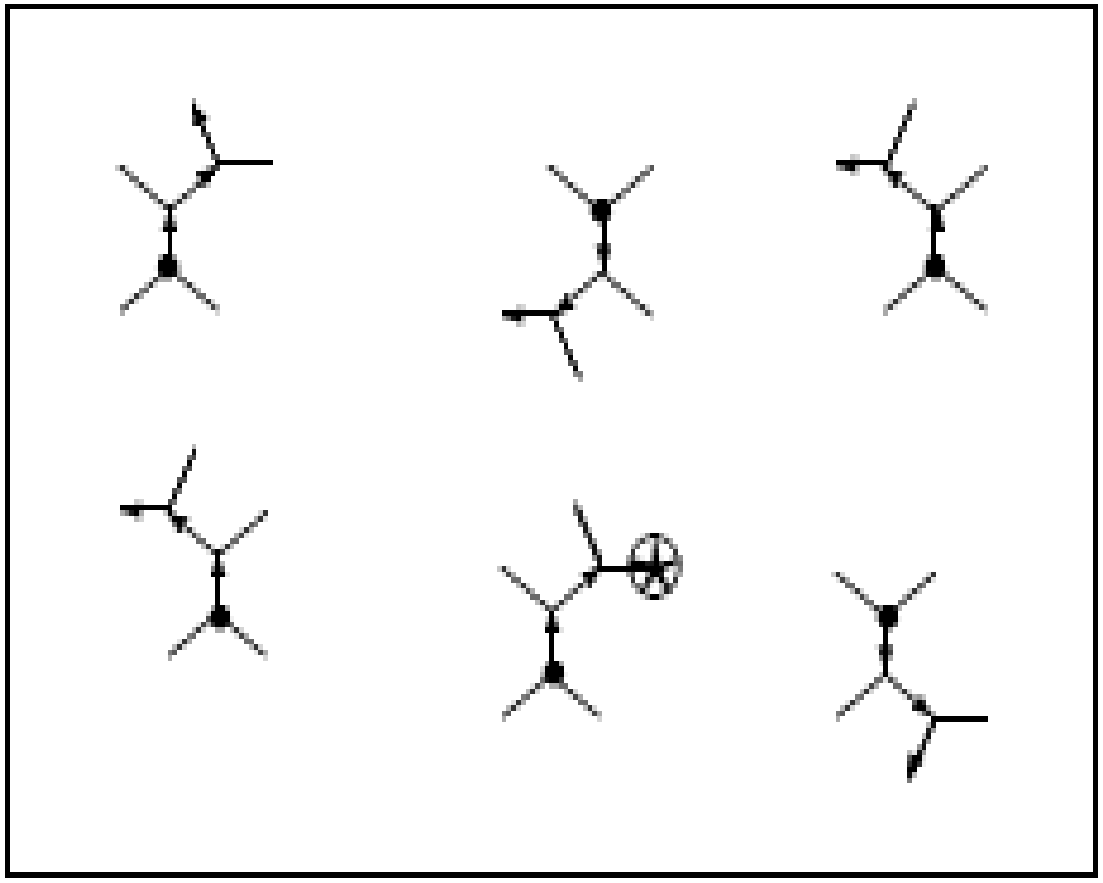


Exhaustive local search



A lot of work, most of it unnecessary

Best Neighbor



Branch from the seed strings

Find best neighbor – highest score

Don't consider branches where the upper bound is not as good as best score so far – in each step move to the "best neighbor" in $D_{=1}(A_i)$

Scoring

- **PatternBranching** uses total distance score:
- For each sequence S_i in the sample $DNA = \{DNA_1, \ldots, DNA_n\}$, let

$$d(A, DNA_i) = min\{d(A, P) \mid P \in DNA_i\}.$$

- Then the total distance of pattern A from the sample is

$$d(A, S) = \sum_{DNA_i \in DNA} d(A, DNA_i).$$

- For a pattern A, let $D=Neighbor(A)$ be the set of patterns which differ from A in exactly 1 position.
- We define $BestNeighbor(A)$ as the pattern $B \in D=Neighbor(A)$ with lowest total distance $d(B, DNA)$.

PatternBranching Algorithm

- **PatternBranching** (*DNA*,*l*,*k*):

  Motif = arbitrary motif pattern

  For each *l* -mer A_0 **in** *DNA*

  For $j = 0$ **to** k

  If $d(A_j, DNA) < d(Motif, DNA)$

  $Motif = A_j$

  $A_{j+1} = BestNeighbor(A_j)$

  Output *Motif*

- *More thorough search*: instead of single pattern we can keep r patterns $B \in D_{=1}(A_j)$ with the lowest total distance $d(B, DNA)$.

PatternBranching Performance

- PatternBranching is faster than other pattern-based algorithms
- Motif Challenge Problem:
  - sample of n = 20 sequences
  - N = 600 nucleotides long
  - implanted pattern of length l = 15
  - k = 4 mutations

Algorithm	Success Rate	Running Time
PROJECTION	about 100%	2 minutes
MITRA	100%	5 minutes
MULTIPROFILER	99.7%	1 minute
PatternBranching	99.7%	3 seconds

Profile Branching

- Profile Branching algorithm is similar to the Pattern Branching Algorithm. However, the search is in the space of motif profiles, instead of motif patterns. The algorithm is obtained from the Pattern Branching Algorithm by making the following changes:
  1. convert each sample string A_0 to a profile $X(A_0)$,
  2. generalize the scoring method to score profiles,
  3. modify the branching method to apply to profiles, and
  4. use the top-scoring profile found as a seed for the EM algorithm.
- Details omitted

Profile Branching

- Profile Branching is about 5 times slower than the Pattern Branching algorithm
- The Pattern Branching Algorithm clearly outperforms the Profile Branching Algorithm on Challenge-like problems. However, pattern-based algorithms have difficulty finding motifs with many degenerate positions.

PMS (Planted Motif Search)

- Generate all possible l-mers out of the input sequence DNA_i. Let C_i be the collection of these l-mers.
- Example:

  AAGTCAGGAGT

  C_i = 3-mers:

  AAG AGT GTC TCA CAG AGG GGA GAG AGT

All patterns at Hamming distance $d = 1$

AAGTCAGGAGT

AAG	**AGT**	**GTC**	**TCA**	**CAG**	**AGG**	**GGA**	**GAG**	**AGT**
CAG	CGT	ATC	ACA	AAG	CGG	AGA	AAG	CGT
GAG	GGT	CTC	CCA	GAG	TGG	CGA	CAG	GGT
TAG	TGT	TTC	GCA	TAG	GGG	TGA	TAG	TGT
ACG	ACT	GAC	TAA	CCG	ACG	GAA	GCG	ACT
AGG	ATT	GCC	TGA	CGG	ATG	GCA	GGG	ATT
ATG	AAT	GGC	TTA	CTG	AAG	GTA	GTG	AAT
AAC	AGA	GTA	TCC	CAA	AGA	GGC	GAA	AGA
AAA	AGC	GTG	TCG	CAC	AGT	GGG	GAC	AGC
AAT	AGG	GTT	TCT	CAT	AGC	GGT	GAT	AGG

Sort the lists

AAG	AGT	GTC	TCA	CAG	AGG	GGA	GAG	AGT
AAA	AAT	ATC	ACA	AAG	AAG	AGA	AAG	AAT
AAC	ACT	CTC	CCA	CAA	ACG	CGA	CAG	ACT
AAT	AGA	GAC	GCA	CAC	AGA	GAA	GAA	AGA
ACG	AGC	GCC	TAA	CAT	AGC	GCA	GAC	AGC
AGG	AGG	GGC	TCC	CCG	AGT	GGC	GAT	AGG
ATG	ATT	GTA	TCG	CGG	ATG	GGG	GCG	ATT
CAG	CGT	GTG	TCT	CTG	CGG	GGT	GGG	CGT
GAG	GGT	GTT	TGA	GAG	GGG	GTA	GTG	GGT
TAG	TGT	TTC	TTA	TAG	TGG	TGA	TAG	TGT

Eliminate duplicates

AAG	AGT	GTC	TCA	CAG	AGG	GGA	GAG	AGT
AAA	~~AAT~~	ATC	ACA	AAG	~~AAG~~	~~AGA~~	~~AAG~~	~~AAT~~
AAC	ACT	CTC	CCA	CAA	~~ACG~~	CGA	~~CAG~~	~~ACT~~
AAT	AGA	GAC	GCA	CAC	~~AGA~~	GAA	~~GAA~~	~~AGA~~
ACG	AGC	GCC	TAA	CAT	AGC	~~GCA~~	~~GAC~~	~~AGC~~
AGG	~~ACG~~	GGC	TCC	CCG	AGT	~~GGC~~	GAT	~~AGG~~
ATG	ATT	GTA	TCG	CGG	~~ATG~~	~~GGG~~	GCG	~~ATT~~
CAG	CGT	GTG	TCT	CTG	~~CGG~~	~~GGT~~	~~GGG~~	~~CGT~~
GAG	GGT	GTT	TGA	~~GAG~~	GGG	~~GTA~~	~~GTG~~	~~GGT~~
TAG	TGT	TTC	TTA	~~TAG~~	TGG	~~TGA~~	~~TAG~~	~~TGT~~

Let L denote the obtained list of l -mers

Find motif common to all lists

- Follow this procedure for all sequences
- Find the motif common to all L_i (once duplicates have been eliminated)
- This is the planted motif

PMS Running Time

- It takes time to
  - Generate variants
  - Sort lists
  - Find and eliminate duplicates

$$O\left(m\binom{l}{d}3^{d} \right)$$

(m denotes the number of different l-mers which are in the first DNA sequence)

- Running time of this algorithm:

$$O\left(t\, m\binom{l}{d}3^{d}\, \frac{l}{w} \right)$$

w is the word length of the computer