Markov Models and

Hidden Markov Models

(from Chapter 3 of [DEKM])

Introduction

- As opposed to pairwise alignment of sequences, now consider questions about a single sequence:
  - Does this sequence belong to a particular family?
  - What can we say about the internal structure of the sequence?
- For example, identify alpha helix or beta sheet regions in protein sequence, or CpG islands in a genomic sequence.
- Use Hidden Markov Models (HMMs) as probabilistic models for sequences.
- HMMs are traditionally used in speech recognition.

CpG Islands

- CpG: C nucleotide followed by G nucleotide ($\cdots$CG$\cdots$)
- C in CpG typically chemically modified by methylation, and methyl-C likely to mutate into T.
- Thus, CpG much rarer in human genome than expected.
- In short stretches of genome, methylation is suppressed, and CpG occurs more frequently than elsewhere.
  - E.g. promoters or 'start' regions of many genes.
- These stretches are **CpG islands**, typically a few hundred to a few thousand bases long.

Questions for CpG Islands

- Given a short stretch of DNA sequence, determine if it comes from a CpG island or not.
- Given a long piece of DNA sequence, how can we identify the CpG islands in it?

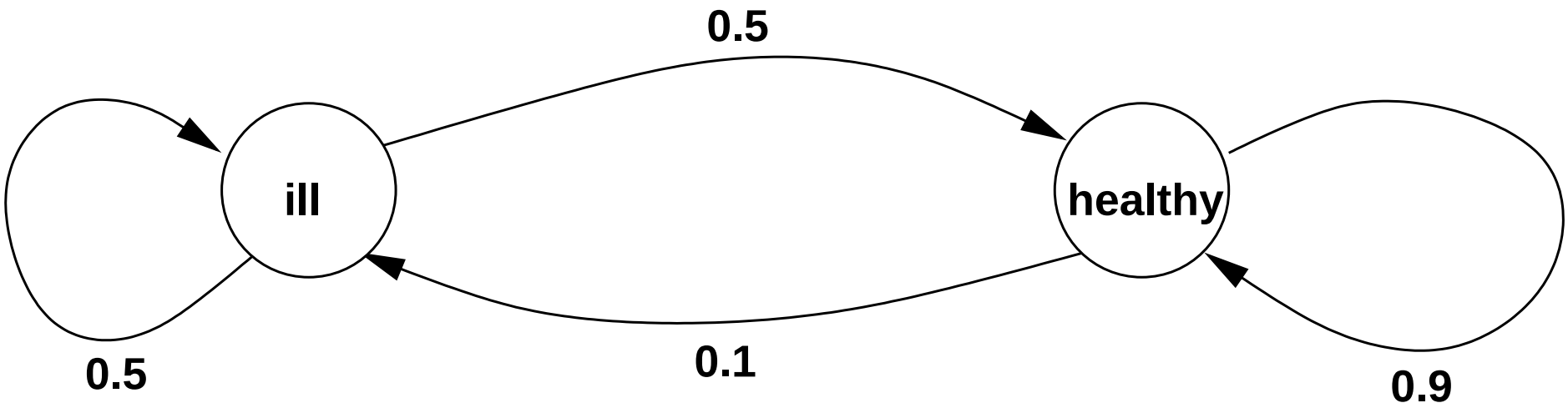
Probabilistic sequence model

- In DNA sequences, dinucleotides are important.
- We want a probabilistic model where the next nucleotide depends on the previous one.
- **Markov chain**: Probabilistic model where the next state depends (only) on the current state:
  - State at time $i = 1, 2, 3, \ldots$ is s_i.
  - Transition probability (from state u to v):

$$a_{uv} = \Pr[s_i = v \mid s_{i-1} = u]$$

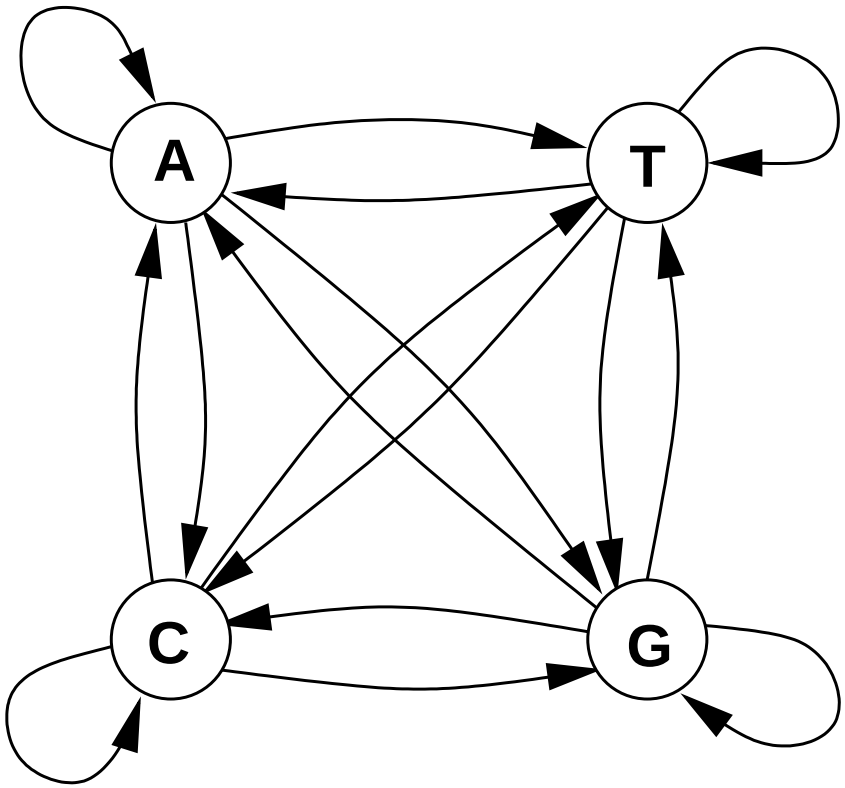
Probability to go from current state u to next state v.

Markov Chain Example



- If healthy at time i, probability is 90% to be healthy at time $i+1$, and 10% to be ill at time $i+1$.
- If ill at time i, probability is 50% to be healthy at time $i+1$, and 50% to be ill at time $i+1$.
- In general: probabilities of all transitions of the same state sum up to one.

Markov Chain for DNA



- Transition probabilities a_{AA}, a_{AT}, a_{AC}, a_{AG}, a_{CA}, etc. (not shown)

Probability of a Sequence

- What's the probability that the Markov chain generates sequence ACATGCAT?

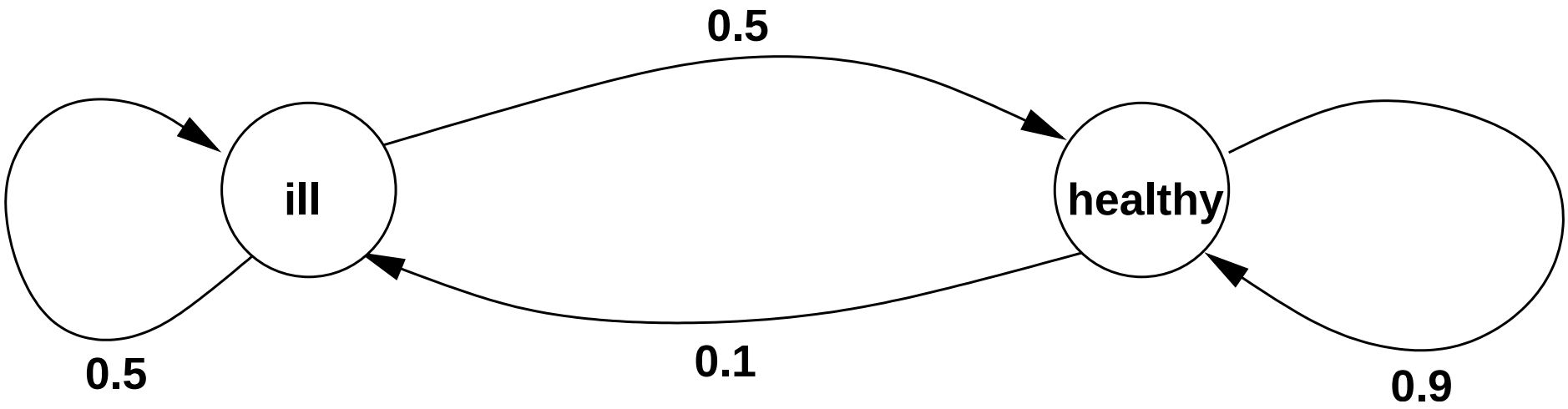
  $$\Pr[\mathrm{ACATGCAT}] = \Pr[\mathrm{A}] \cdot a_{\mathrm{AC}} \cdot a_{\mathrm{CA}} \cdot a_{\mathrm{AT}} \cdot a_{\mathrm{TG}} \cdot a_{\mathrm{GC}} \cdot a_{\mathrm{CA}} \cdot a_{\mathrm{AT}}$$

  where $\Pr[\mathrm{A}]$ is the probability for starting with A.
- In general: Probability for state sequence $x_1 x_2 x_3 \cdots x_n$:

$$\begin{aligned} \Pr[x_1 x_2 \cdots x_n] &= \Pr[x_1] \cdot \Pr[x_2 \mid x_1] \cdot \Pr[x_3 \mid x_2] \cdots \Pr[x_n \mid x_{n-1}] \\ &= \Pr[x_1] \cdot \prod_{i=2}^{n} a_{x_{i-1}, x_i} \end{aligned}$$

- Markov: $\Pr[x_i \mid x_1 x_2 \ldots x_{i-1}] = \Pr[x_i \mid x_{i-1}] = a_{x_{i-1} x_i}$

Example of Sequence Probability

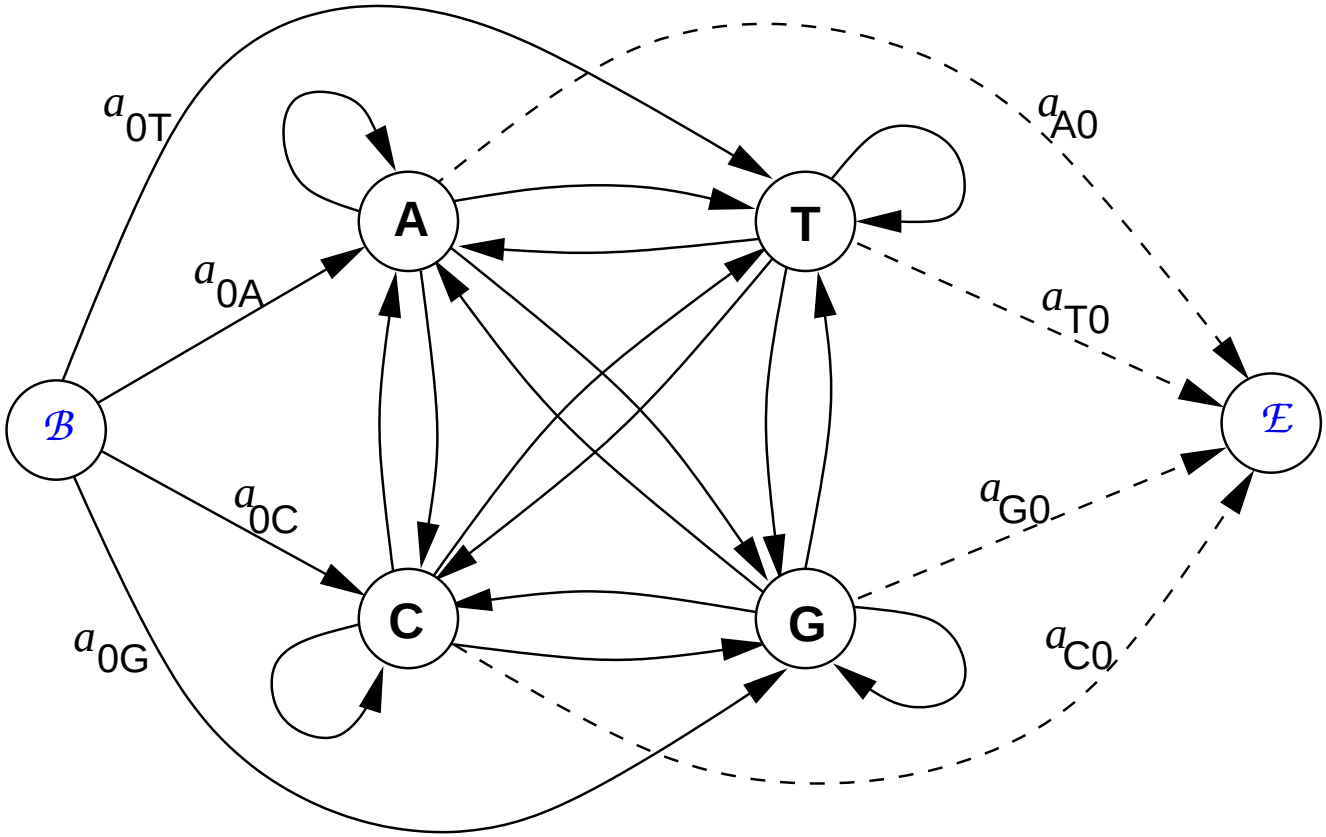


- Assume start state is 'healthy' ($\Pr[\text{healthy}] = 1$).
- Probability for sequence

  healthy, ill, ill, healthy, ill

  is $1 \cdot 0.1 \cdot 0.5 \cdot 0.5 \cdot 0.1 = 0.0025$.

Modelling Beginning and End



- To model beginning and end of sequence, can add new states $\mathcal{B}$ and $\mathcal{E}$.
- Always start in $\mathcal{B}$ ($\Pr[\mathcal{B}] = 1$) at time 0 and end in $\mathcal{E}$.

Recognising CpG Islands

- **Goal:** For a given stretch of DNA, decide whether it is from a CpG island or not.
- **Approach:** Derive two different Markov chains, one from putative CpG islands in known DNA sequences (+ Model), one from the remainder of the sequences (− Model).
- Calculate a_{uv} for each model via

$$a_{uv} = \frac{c_{uv}}{\sum_w c_{uw}}$$

where c_{xy} is the number of occurrences of xy in the respective data set.

Resulting Markov chains

+	A	C	G	T
A	0.180	0.274	0.426	0.120
C	0.171	0.368	**0.274**	0.188
G	0.161	0.339	0.375	0.125
T	0.079	0.355	0.384	0.182

−	A	C	G	T
A	0.300	0.205	0.285	0.210
C	0.322	0.298	**0.078**	0.302
G	0.248	0.246	0.298	0.208
T	0.177	0.239	0.292	0.292

Using the Markov Chains

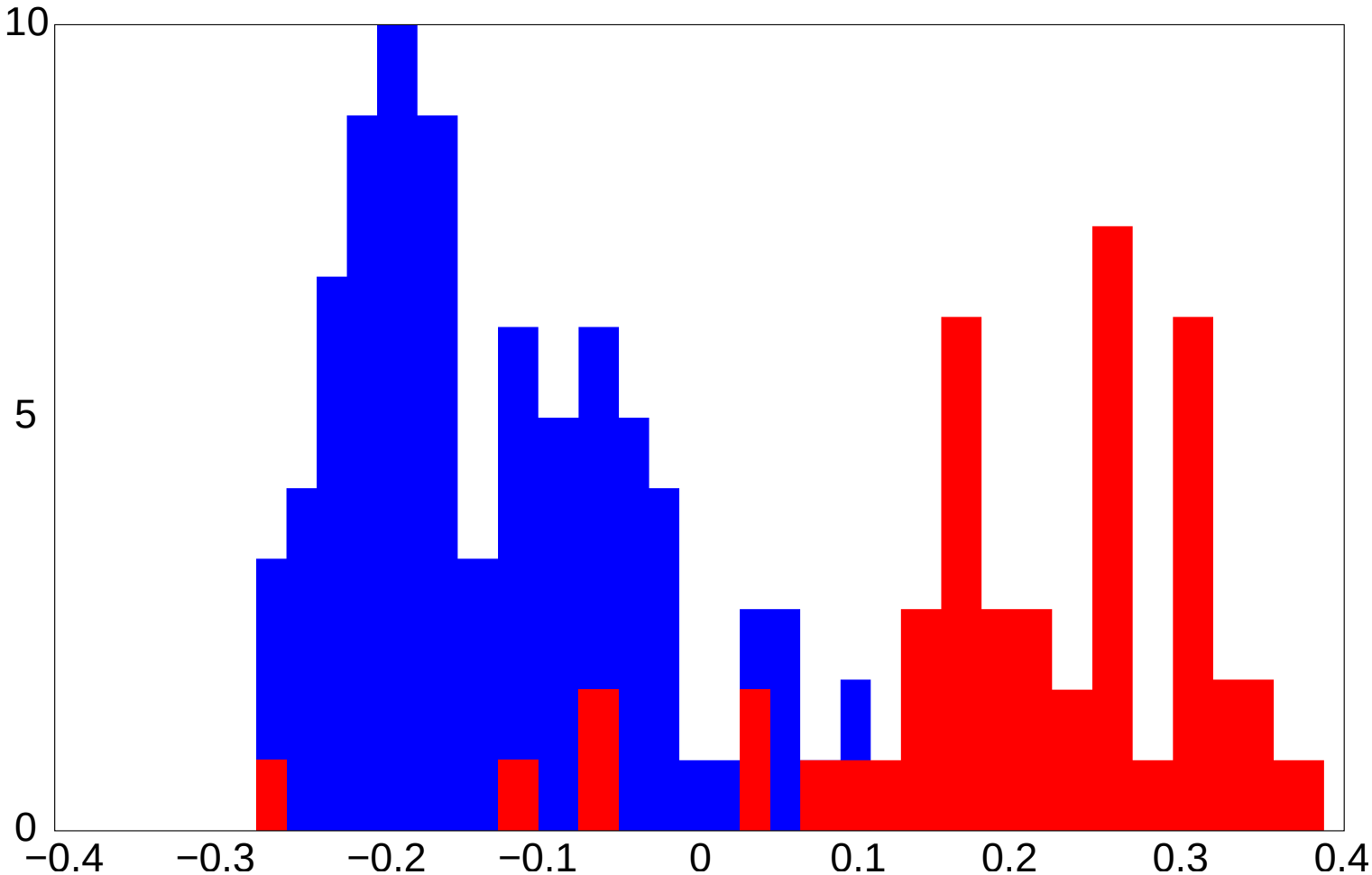
- To determine whether a sequence x of length n is from a CpG island or not, consider the log-odds ratio:

$$\begin{aligned} S(x) &= \log \frac{\Pr[x \mid \text{model } +]}{\Pr[x \mid \text{model } -]} \\ &= \log \frac{\Pr[x_1] a^+_{x_1 x_2} \cdots a^+_{x_{n-1} x_n}}{\Pr[x_1] a^-_{x_1 x_2} \cdots a^-_{x_{n-1} x_n}} \\ &= \sum_{i=2}^{n} \log \frac{a^+_{x_{i-1} x_i}}{a^-_{x_{i-1} x_i}} \end{aligned}$$

  where a^+_{uv} and a^-_{uv} are the transition probabilities in model $+$ and $-$, respectively.

- Positive $S(x)$ means: x is likely to be from a CpG island

Histogram of Scores



- Scores normalised by length
- Red: CpG islands, Blue: non-CpG islands

Identifying CpG Islands

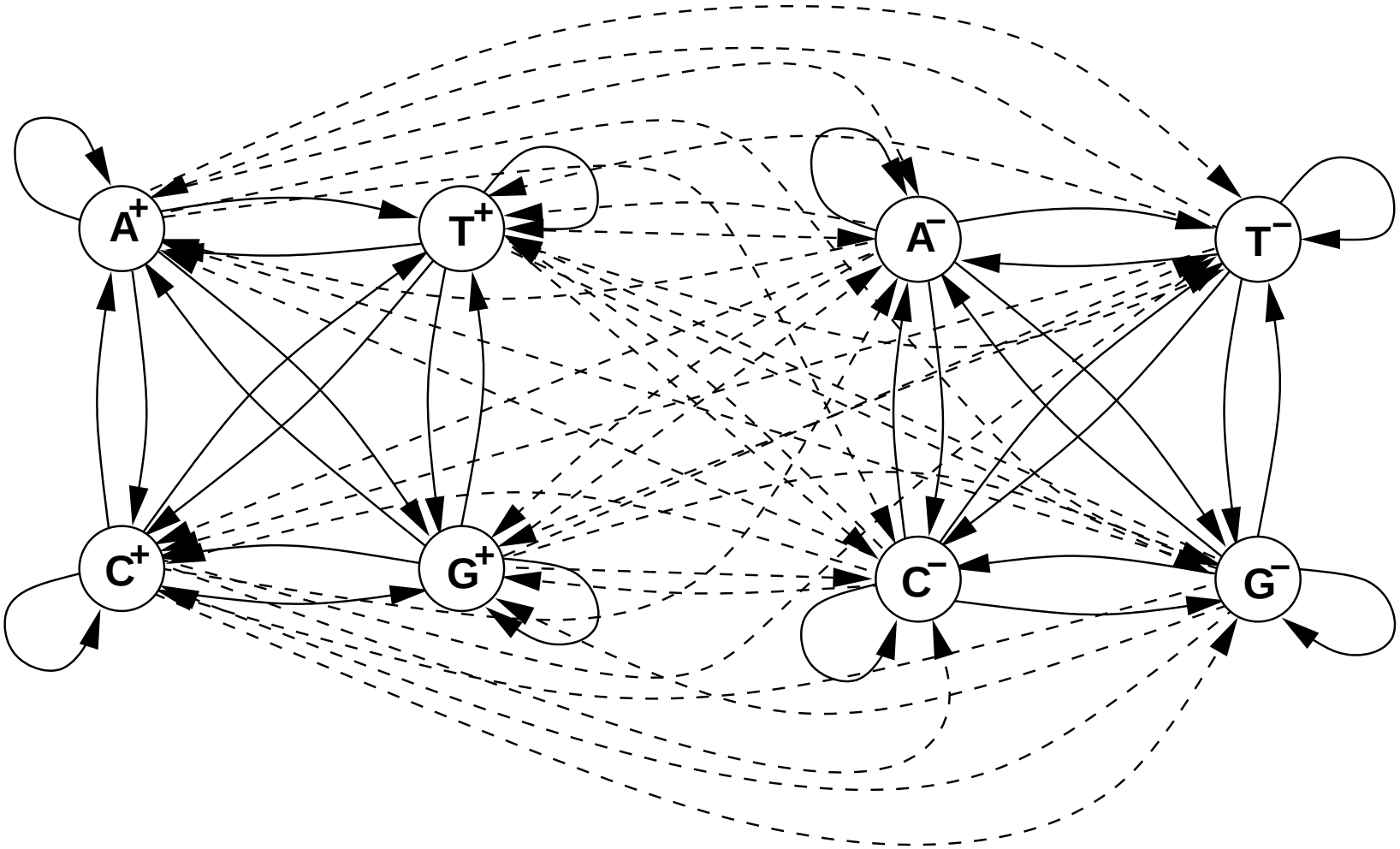
- Given a long piece of genomic sequence, how can we identify the CpG islands in it?
- First approach: Compute log-odds score for windows of length 100, and view those with positive score as CpG islands.
- Not satisfactory, as CpG islands may have different lengths, so why choose windows of length 100?
- Alternative approach: Hidden Markov Models

Hidden Markov Models

Hidden Markov Models

- So far: States of Markov chain correspond directly to the symbols we observe.
- Hidden Markov model: Separation between states and observed symbols.
- Several states may 'emit' the same symbol.
- A state may 'emit' different symbols.
- If we observe a symbol, we do not necessarily know in which state the Markov chain is. (Therefore: **Hidden** Markov Model)

HMM for DNA Sequences



Explanation of HMM for DNA

- Combination of Markov chains for $+$ model and $-$ model.
- States $\{\mathrm{A}^+, \mathrm{C}^+, \mathrm{G}^+, \mathrm{T}^+\}$ correspond to CpG islands, states $\{\mathrm{A}^-, \mathrm{C}^-, \mathrm{G}^-, \mathrm{T}^-\}$ to non-CpG islands.
- Different states emit same symbol. For example, A^+ and A^- both emit A.
- Transition probabilities inside $\{\mathrm{A}^+, \mathrm{C}^+, \mathrm{G}^+, \mathrm{T}^+\}$ and inside $\{\mathrm{A}^-, \mathrm{C}^-, \mathrm{G}^-, \mathrm{T}^-\}$ are similar to $+$ and $-$ model, respectively.
- In addition, transitions between $+$ states and $-$ states are possible.

Formal Definition of HMM

- State sequence ('path'): $\pi = \pi_1 \pi_2 \cdots \pi_n$
- Emitted symbol sequence: $x = x_1 x_2 \cdots x_n$
- Transition probability from state u to v:

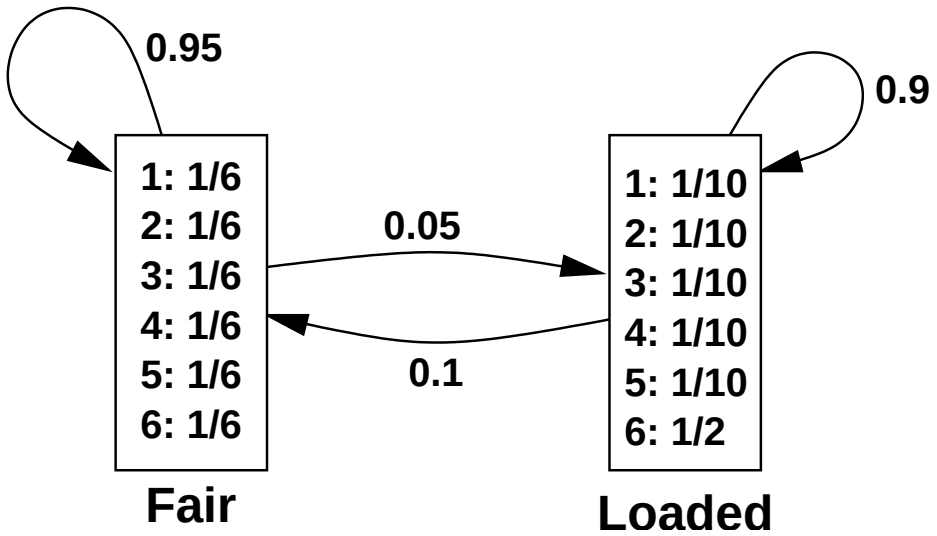
$$a_{uv} = \Pr[\pi_i = v \mid \pi_{i-1} = u]$$

- Probability a_{0u} for starting in state u
- Emission probability of symbol b in state k: $e_k(b)$

- **Remark:** In our HMM for DNA sequences, all $e_k(b)$ were 0 or 1, e.g. $e_{A^+}(A) = 1$ and $e_{A^+}(C) = 0$.

Occasionally Dishonest Casino

- **HMM example**:
  Casino that occasionally switches to a loaded die:



- After observing a sequence of rolls, how can you identify the parts of the sequence when a loaded die was used?

Probability of path and emitted sequence

- Probability of state sequence $\pi = \pi_1 \cdots \pi_n$ and observed sequence $x = x_1 \cdots x_n$ is

$$\Pr[\pi, x] = a_{0\pi_1} \cdot \prod_{i=1}^{n} e_{\pi_i}(x_i) \cdot a_{\pi_i \pi_{i+1}}$$

- **Example:** Probability of $\pi =$C$^+$, G$^-$, C$^-$, G$^+$ with sequence $x =$CGCG (in the DNA sequence HMM) is:

$$a_{0,\mathrm{C}^+} \cdot 1 \cdot a_{\mathrm{C}^+,\mathrm{G}^-} \cdot 1 \cdot a_{\mathrm{G}^-,\mathrm{C}^-} \cdot 1 \cdot a_{\mathrm{C}^-,\mathrm{G}^+} \cdot 1 \cdot a_{\mathrm{G}^+,0}$$

- **Most probable state path approach:**
  - Find path π that maximises $\Pr[\pi, x]$

Example: DNA

- The sequence x =CGCG can be produced by several state paths:
  C^+, G^-, C^-, G^+
  C^-, G^-, C^-, G^-
  C^+, G^+, C^+, G^+
  C^-, G^-, C^+, G^+
  $\vdots$
- The path π that maximises $\Pr[\pi, x]$ is:
  C^+, G^+, C^+, G^+
- This suggests that CGCG was produced by the state sequence C^+, G^+, C^+, G^+ and, hence, is part of a CpG island.

The 'Decoding' Problem

- Given: Hidden Markov Model, observed sequence x
- Goal: Find most probable state path, i.e. a path π maximising $\Pr[\pi, x]$. Call this path π^*

- Can be solved by the **Viterbi** algorithm.
- Algorithm is based on dynamic programming.

The Viterbi Algorithm

- Define $v_k(i)$ as the probability of the most probable path $\pi_1\pi_2\pi_3\cdots\pi_i$ of length i ending in state $\pi_i = k$ with observation x_i.
- Compute all values $v_k(i)$ with dynamic programming, for all states k and all times $i = 0, 1, 2, \ldots, n$.
- The probability of the most probable path generating x is $\max_k v_k(n) \cdot a_{k,0}$.
- The most probable path itself can be obtained via traceback.

Calculating the $v_k(i)$

- $v_k(i)$ is the probability of the most probable path π ending in state $\pi_i = k$ with observation x_i.
- Dynamic programming equations:
  - At time 0 the HMM is always in state 0 (the begin state), hence we can calculate $v_k(0)$ as follows:

$$\begin{aligned} v_0(0) &= 1 \\ v_k(0) &= 0 \quad \text{for } k \neq 0 \end{aligned}$$

  - To compute $v_k(i)$, $1 \leq i \leq n$, consider all best paths of length $i-1$ for extension to a path of length i:

$$v_k(i) = \max_{\ell} \left(v_\ell(i-1) \cdot a_{\ell,k} \cdot e_k(x_i) \right)$$

Traceback

- The probability of the path π maximising $\Pr[\pi, x]$ is $\max_k v_k(n) \cdot a_{k,0}$. How do we get the actual path π?
- Storing traceback information: When computing

$$v_k(i) = \max_\ell \left(v_\ell(i-1) \cdot a_{\ell,k} \cdot e_k(x_i) \right)$$

  we store the value of ℓ that gives the maximum in $p_k(i)$.
- Traceback:
  - Let j be such that $v_j(n) \cdot a_{j,0} = \max_k v_k(n) \cdot a_{k,0}$. Set $\pi^*_n = j$.
  - For $i = n, n-1, n-2, \ldots, 2$, set $\pi^*_{i-1} = p_{\pi^*_i}(i)$.
  - Path $\pi^*_1 \pi^*_2 \cdots \pi^*_n$ is the most probable path.

Example

Dynamic programming table $v_k(i)$ for the DNA sequence HMM and observed sequence x =CGCG:

v		C	G	C	G
	0	1	2	3	4
0	1	0	0	0	0
A^+	0	0	0	0	0
C^+	0	**0.13**	0	**0.012**	0
G^+	0	0	**0.034**	0	**0.0032**
T^+	0	0	0	0	0
A^-	0	0	0	0	0
C^-	0	0.13	0	0.0026	0
G^-	0	0	0.010	0	0.00021
T^-	0	0	0	0	0

Best path π^*:
C^+,G^+,C^+,G^+

Example: Casino

```
Rolls   5335166214165254121223125222631435325614141455414524563411 45
Die     LLLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFLLFFFFFFFFFFFFFFFFFFF
Viterbi FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

Rolls   4124662645566634454325215125333453161265243131316445463416 53
Die     FFFFFFFFFLLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
Viterbi FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

Rolls   2151166234261666563566226165125562242455314532415636321532 56
Die     FFFLLLLLLLLLLLLLLLLLLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
Viterbi FFFFFFFFFFLLLLLLLLLLLLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

Rolls   1156463366666466666541236123666621145621432554113356452511 66
Die     FFLLLLLLLLLLLLLLLFFFFFFFFFFLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFL
Viterbi FFFLLLLLLLLLLLLLLLLLLLLLLLLLLLLFFFFFFFFFFFFFFFFFFFFFFFFF
```

Running the Viterbi Algorithm

- Running-time of Viterbi algorithm: $O(nm^2)$, where n is sequence length and m is number of states.
- Implementation problem: Calculated probabilities $v_k(i)$ become very small.
- This leads to rounding errors (underflow) and thus numerical instabilities.
- Therefore, it is advisable to calculate the logarithms of the values instead:

$$\log v_k(i) = \max_\ell \left(\log v_\ell(i-1) + \log a_{\ell,k} + \log e_k(x_i) \right)$$

- With this modification, the numbers stay reasonable.

Results of Viterbi Algorithm

- In [DEKM], Viterbi algorithm was used to identify CpG islands in 41 sequences, each containing one putative CpG island.
- Results of computation:
  - All CpG islands found, except 2 (false negatives).
  - 121 new CpG islands found (false positives).
  - Real CpG islands are long (1,000 bases), algorithm tends to find shorter ones (e.g. yielding several shorter CpG islands for one real CpG island).
  - Postprocessing (merging short CpG islands that are close to each other, discarding very short CpG islands) reduces number of false positives to 67.

Further remarks about HMMs

Probability of Emitted Sequence

- For a given HMM, we might be interested in the probability that it generates an observed sequence x: $\Pr[x]$
- In principle, this can be calculated as follows:

$$\Pr[x] = \sum_{\pi} \Pr[\pi, x]$$

  However, this is a sum over all possible paths π of length n, and there can be exponentially many such paths.
- Fortunately, $\Pr[x]$ can be calculated efficiently using dynamic programming, similar to the Viterbi algorithm (called the **forward algorithm**). See [DEKM].

Posterior State Probabilities

- The posterior probability that the HMM was in state k in step i is:

$$\Pr[\pi_i = k \mid x]$$

- This can also be calculated efficiently using dynamic programming (using forward and backward algorithm). See [DEKM].
- Alternative approach to decoding problem (instead of using the path π that maximises $\Pr[\pi, x]$):
  - Define path $\hat{\pi}$ by selecting $\hat{\pi}_i$ as the state k with largest posterior probability $\Pr[\pi_i = k \mid x]$.

Parameter and Model Estimation

- We have assumed that we know the structure and parameters of the HMM: states, transitions, transition probabilities a_{uv}, emission probabilities $e_k(b)$
- If parameters a_{uv} and $e_k(b)$ are unknown, estimate them from training data:
  - Baum-Welch training
  - Viterbi training
- Model structure is often constructed manually by carefully deciding states and allowed transitions based on knowledge about the problem under consideration. (Automated methods exist as well.)

- **Remark:** HMMs are also used for pairwise alignment.

Restriction Mapping

and Exhaustive Search

(from Chapter 4 of [JP])

Restriction Enzymes

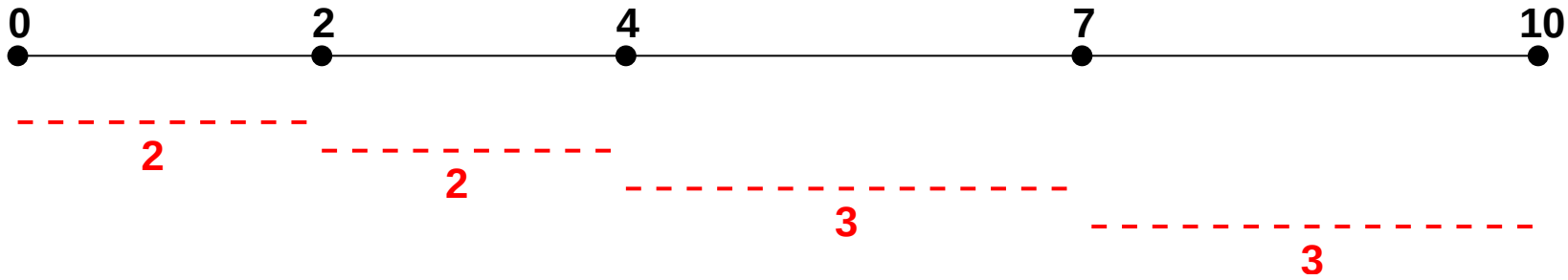
- Hamilton Smith discovered in 1970 that the **restriction enzyme** HindII cleaves DNA molecules at every occurrence (site) of GTGCAC or GTTAAC.
- This breaks a long DNA molecule into **restriction fragments**.
- Can use experimental data to create **restriction maps**, showing the location of restriction sites on DNA sequences.
- Restriction maps were a very popular research tool in the late 1980s, but are less important today because of availability of efficient DNA sequencing technologies.
- We discuss them to illustrate the algorithmic design principle of exhaustive search.

Measuring DNA Length

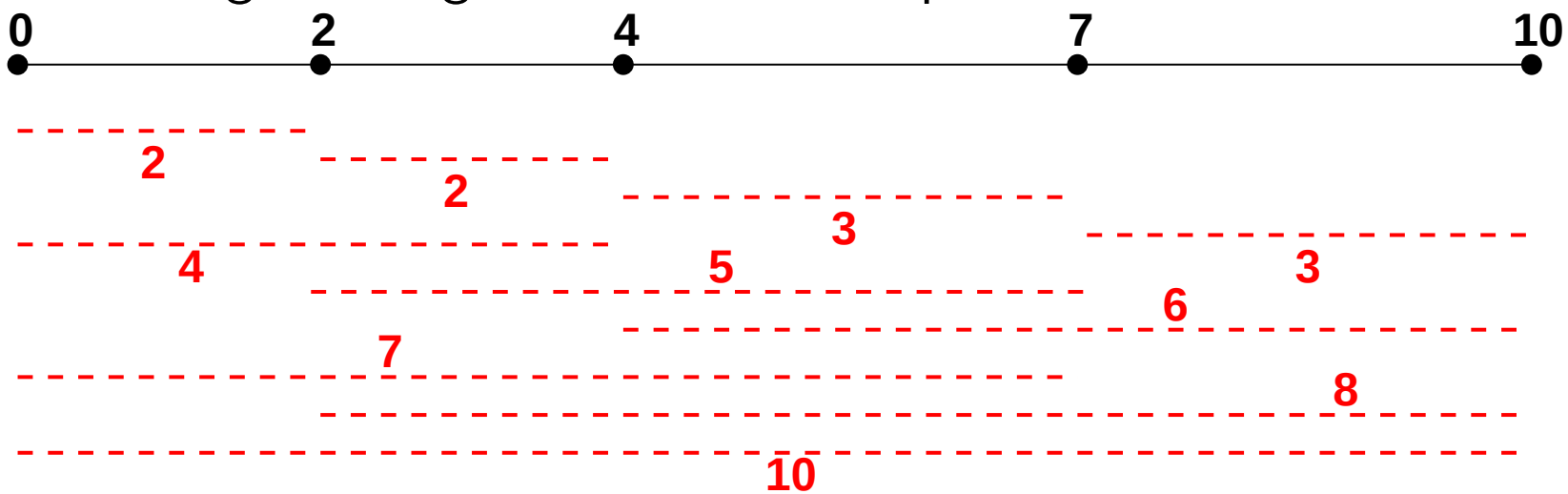
- If a DNA molecule is cut into restriction fragments, we want to measure the size of the fragments.
- DNA is a negatively charged molecule and migrates towards the positive pole of an electric field.
- If the DNA molecules are put in a gel, longer molecules move slower than faster ones.
- We can estimate the size of the fragments by observing their migration speed (after attaching 'molecular light bulbs', i.e. fluorescent compounds).
- This technique of measuring DNA length is called **gel electrophoresis**.

Complete versus Partial Digest

- Complete digest: Fragments between consecutive sites



- Partial digest: Fragments between all pairs of sites



Some Notation

- A **multiset** is a set allowing duplicate elements, e.g. $\{2, 2, 2, 3, 3, 4, 5\}$.
- If $X = \{x_1, \ldots, x_n\}$ is a set of points on a line in increasing order, then ΔX denotes the multiset of pairwise distances between them:

$$\Delta X = \{x_j - x_i \mid 1 \leq i < j \leq n\}$$

- **Example:**
  $X = \{0, 2, 4, 7, 10\}$, $\Delta(X) = \{2, 2, 3, 3, 4, 5, 6, 7, 8, 10\}$
- Note: If X has n points, $\Delta(X)$ contains $\binom{n}{2} = \frac{n(n-1)}{2}$ distances.

Partial Digest Problem

- Restriction mapping can be formalised as the problem of reconstructing the positions of points on a line, given the pairwise distances of the points.
- **Partial Digest Problem (PDP):**
  - **Input:** A multiset of pairwise distances L, containing $\binom{n}{2}$ positive integers.
  - **Solution:** Set X of n integers with $\Delta X = L$ (if it exists).
- If X is a solution, then the set Y obtained by adding the same number to all points in X is also a solution, and $Z = \{-x \mid x \in X\}$ is also a solution.
  - For $L = \{1, 2, 3\}$, sets $X = \{0, 1, 3\}$ and $Y = \{5, 6, 8\}$ and $Z = \{0, -1, -3\}$ are possible solutions.

➠ Consider solutions that include 0 and are non-negative.

Homometric Sets

- Two sets A and B are called **homometric** if $\Delta(A) = \Delta(B)$.
- One more example:

  $$A = \{0, 1, 3, 8, 9, 11, 12, 13, 15\}$$

  and

  $$B = \{0, 1, 3, 4, 5, 7, 12, 13, 15\}$$

  are homometric. We have:

  $$\Delta(A) = \Delta(B) = \{1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 5, 5,$$
  $$6, 6, 7, 7, 8, 8, 8, 9, 9, 10, 10, 11, 11, 12, 12, 12, 13, 14, 15\}$$

- Variation of PDP: Instead of finding just one solution X, find **all** homometric sets (that contain 0 and are non-negative).

First Attempt: Brute-Force

Algorithm BruteForcePDP

Input: distance set L with $\binom{n}{2}$ numbers
Output: set X with $\Delta X = L$

$M \leftarrow$ maximum element in L
for every set of $n-2$ integers $0 < x_2 < \cdots < x_{n-1} < M$ **do**
    $X \leftarrow \{0, x_2, x_3, \ldots, x_{n-1}, M\}$
    compute ΔX from X
    if $\Delta X = L$ **then**
        return X
return "No Solution"

Analysis of BruteForcePDP

- Correctness of algorithm is easy to see.
- Algorithm tries all subsets of size $n-2$ of numbers between 0 and M.
- There are $\binom{M-1}{n-2}$ such subsets, and dealing with each of them requires $O(n^2)$ time for computing ΔX.
- $\binom{M-1}{n-2}$ is not much better than $O(M^{n-2})$.
- This is a **very slow** algorithm.

- **Idea for improvement:** Instead of trying all possible integers between 0 and M for positions of points, try only those that are contained in L.

Second Brute-Force Approach

Algorithm AnotherBruteForcePDP

Input: distance set L with $\binom{n}{2}$ numbers
Output: set X with $\Delta X = L$

$M \leftarrow$ maximum element in L
for every set of $n-2$ integers $0 < x_2 < \cdots < x_{n-1} < M$ from L **do**
    $X \leftarrow \{0, x_2, x_3, \ldots, x_{n-1}, M\}$
    compute ΔX from X
    if $\Delta X = L$ **then**
        return X
return "No Solution"

Analysis of Second Brute-Force Algo

- Correctness is clear, as every positive position x in X must occur as an element of L (as distance between 0 and x).
- Algorithm tries all subsets of size $n-2$ of L.
- There are $\binom{|L|}{n-2}$ such subsets, where $|L| = \binom{n}{2} = \frac{n(n-1)}{2}$.
- Number of subsets is bounded roughly by $O(n^{2n-4})$, which is better than $O(M^{n-2})$ for large values of M.
- Nevertheless, this is also a **very slow** algorithm.

A Practical Algorithm

- Described by **Steven Skiena** in 1990.
- Find the largest distance M in L. The first and last point must be 0 and M. Put these into X and remove M from L.
- At any step, consider the largest distance δ among those remaining in L.
- We must have a point at δ or at $M - \delta$.
- For each of these two candidate points, check whether its distances to all points already added to X are still in L. If so, remove those distances from L, add the point to X, and repeat (recursively).
- If L becomes empty, we have found a solution.
- If none of the two candidate points works, backtrack.

Example of Skiena's Algorithm

- Input: $L = \{2, 2, 3, 3, 4, 5, 6, 7, 8, 10\}$
- $|L| = 10 = \frac{n(n-1)}{2}$, so $n = 5$.
- $M = 10$, so the algorithm starts with:

$$X = \{0, 10\} \qquad L = \{2, 2, 3, 3, 4, 5, 6, 7, 8\}$$

- Largest remaining distance is $\delta = 8$.
  - ➠ Candidate points 8 and $10 - 8 = 2$.
- Consider candidate 2. Distances to X are $\{2, 8\}$.
- These are in L, so we can continue with:

$$X = \{0, 2, 10\} \qquad L = \{2, 3, 3, 4, 5, 6, 7\}$$

Example of Skiena's Algorithm

- Input: $L = \{2, 2, 3, 3, 4, 5, 6, 7, 8, 10\}$
- $|L| = 10 = \frac{n(n-1)}{2}$, so $n = 5$.
- $M = 10$, so the algorithm starts with:

$$X = \{0, 10\} \qquad L = \{2, 2, 3, 3, 4, 5, 6, 7, 8\}$$

- Largest remaining distance is $\delta = 8$.
  - ➠ Candidate points 8 and $10 - 8 = 2$.
- Consider candidate 2. Distances to X are $\{2, 8\}$.
- These are in L, so we can continue with:

$$X = \{0, 2, 10\} \qquad L = \{2, 3, 3, 4, 5, 6, 7\}$$

Example of Skiena's Algorithm

- Input: $L = \{2, 2, 3, 3, 4, 5, 6, 7, 8, 10\}$
- $|L| = 10 = \frac{n(n-1)}{2}$, so $n = 5$.
- $M = 10$, so the algorithm starts with:

$$X = \{0, 10\} \qquad L = \{2, 2, 3, 3, 4, 5, 6, 7, 8\}$$

- Largest remaining distance is $\delta = 8$.
  - ➠ Candidate points 8 and $10 - 8 = 2$.
- Consider candidate 2. Distances to X are $\{2, 8\}$.
- These are in L, so we can continue with:

$$X = \{0, 2, 10\} \qquad L = \{2, 3, 3, 4, 5, 6, 7\}$$

Example of Skiena's Algorithm

- Input: $L = \{2, 2, 3, 3, 4, 5, 6, 7, 8, 10\}$
- $|L| = 10 = \frac{n(n-1)}{2}$, so $n = 5$.
- $M = 10$, so the algorithm starts with:

$$X = \{0, 10\} \qquad L = \{2, 2, 3, 3, 4, 5, 6, 7, 8\}$$

- Largest remaining distance is $\delta = 8$.

  ➠ Candidate points 8 and $10 - 8 = 2$.
- Consider candidate 2. Distances to X are $\{2, 8\}$.
- These are in L, so we can continue with:

$$X = \{0, 2, 10\} \qquad L = \{2, 3, 3, 4, 5, 6, 7\}$$

Example of Skiena's Algorithm

- Input: $L = \{2, 2, 3, 3, 4, 5, 6, 7, 8, 10\}$
- $|L| = 10 = \frac{n(n-1)}{2}$, so $n = 5$.
- $M = 10$, so the algorithm starts with:

$$X = \{0, 10\} \qquad L = \{2, 2, 3, 3, 4, 5, 6, 7, 8\}$$

- Largest remaining distance is $\delta = 8$.
  - ➠ Candidate points 8 and $10 - 8 = 2$.
- Consider candidate 2. Distances to X are $\{2, 8\}$.
- These are in L, so we can continue with:

$$X = \{0, 2, 10\} \qquad L = \{2, 3, 3, 4, 5, 6, 7\}$$

Example of Skiena's Algorithm (2)

$$X = \{0, 2, 10\} \qquad L = \{2, 3, 3, 4, 5, 6, 7\}$$

- Largest distance 7. Candidates: 7 and $10 - 7 = 3$.
- Try candidate 3. Distances to X are $\{3, 1, 7\}$.
- But 1 is not in L, hence candidate 3 is not possible.
- Try candidate 7. Distances to X are $\{7, 5, 3\}$.
- These are in L, so we can continue with:

$$X = \{0, 2, 7, 10\} \qquad L = \{2, 3, 4, 6\}$$

Example of Skiena's Algorithm (2)

$$X = \{0, 2, 10\} \qquad L = \{2, 3, 3, 4, 5, 6, 7\}$$

- Largest distance 7. Candidates: 7 and $10 - 7 = 3$.
- Try candidate 3. Distances to X are $\{3, 1, 7\}$.
- But 1 is not in L, hence candidate 3 is not possible.
- Try candidate 7. Distances to X are $\{7, 5, 3\}$.
- These are in L, so we can continue with:

$$X = \{0, 2, 7, 10\} \qquad L = \{2, 3, 4, 6\}$$

Example of Skiena's Algorithm (2)

$$X = \{0, 2, 10\} \qquad L = \{2, 3, 3, 4, 5, 6, 7\}$$

- Largest distance 7. Candidates: 7 and $10 - 7 = 3$.
- Try candidate 3. Distances to X are $\{3, 1, 7\}$.
- But 1 is not in L, hence candidate 3 is not possible.
- Try candidate 7. Distances to X are $\{7, 5, 3\}$.
- These are in L, so we can continue with:

$$X = \{0, 2, 7, 10\} \qquad L = \{2, 3, 4, 6\}$$

Example of Skiena's Algorithm (2)

$$X = \{0, 2, 10\} \qquad L = \{2, 3, 3, 4, 5, 6, 7\}$$

- Largest distance 7. Candidates: 7 and $10 - 7 = 3$.
- Try candidate 3. Distances to X are $\{3, 1, 7\}$.
- But 1 is not in L, hence candidate 3 is not possible.
- Try candidate 7. Distances to X are $\{7, 5, 3\}$.
- These are in L, so we can continue with:

$$X = \{0, 2, 7, 10\} \qquad L = \{2, 3, 4, 6\}$$

Example of Skiena's Algorithm (2)

$$X = \{0, 2, 10\} \qquad L = \{2, 3, 3, 4, 5, 6, 7\}$$

- Largest distance 7. Candidates: 7 and $10 - 7 = 3$.
- Try candidate 3. Distances to X are $\{3, 1, 7\}$.
- But 1 is not in L, hence candidate 3 is not possible.
- Try candidate 7. Distances to X are $\{7, 5, 3\}$.
- These are in L, so we can continue with:

$$X = \{0, 2, 7, 10\} \qquad L = \{2, 3, 4, 6\}$$

Example of Skiena's Algorithm (2)

$$X = \{0, 2, 10\} \qquad L = \{2, 3, 3, 4, 5, 6, 7\}$$

- Largest distance 7. Candidates: 7 and $10 - 7 = 3$.
- Try candidate 3. Distances to X are $\{3, 1, 7\}$.
- But 1 is not in L, hence candidate 3 is not possible.
- Try candidate 7. Distances to X are $\{7, 5, 3\}$.
- These are in L, so we can continue with:

$$X = \{0, 2, 7, 10\} \qquad L = \{2, 3, 4, 6\}$$

Example of Skiena's Algorithm (3)

$$X = \{0, 2, 7, 10\} \qquad L = \{2, 3, 4, 6\}$$

- Largest distance 6. Candidates: 6 and $10 - 6 = 4$.
- Try candidate 4. Distances to X are $\{4, 2, 3, 6\}$.
- These are in L, so we can continue with:

$$X = \{0, 2, 4, 7, 10\} \qquad L = \emptyset$$

- Now L is empty, hence we have found a solution:

$$X = \{0, 2, 4, 7, 10\}, \quad \Delta X = \{2, 2, 3, 3, 4, 5, 6, 7, 8, 10\}$$

Example of Skiena's Algorithm (3)

$$X = \{0, 2, 7, 10\} \qquad L = \{2, 3, 4, 6\}$$

- Largest distance 6. Candidates: 6 and $10 - 6 = 4$.
- Try candidate 4. Distances to X are $\{4, 2, 3, 6\}$.
- These are in L, so we can continue with:

$$X = \{0, 2, 4, 7, 10\} \qquad L = \emptyset$$

- Now L is empty, hence we have found a solution:

$$X = \{0, 2, 4, 7, 10\}, \quad \Delta X = \{2, 2, 3, 3, 4, 5, 6, 7, 8, 10\}$$

Example of Skiena's Algorithm (3)

$$X = \{0, 2, 7, 10\} \qquad L = \{2, 3, 4, 6\}$$

- Largest distance 6. Candidates: 6 and $10 - 6 = 4$.
- Try candidate 4. Distances to X are $\{4, 2, 3, 6\}$.
- These are in L, so we can continue with:

$$X = \{0, 2, 4, 7, 10\} \qquad L = \emptyset$$

- Now L is empty, hence we have found a solution:

$$X = \{0, 2, 4, 7, 10\}, \quad \Delta X = \{2, 2, 3, 3, 4, 5, 6, 7, 8, 10\}$$

Example of Skiena's Algorithm (3)

$$X = \{0, 2, 7, 10\} \qquad L = \{2, 3, 4, 6\}$$

- Largest distance 6. Candidates: 6 and $10 - 6 = 4$.
- Try candidate 4. Distances to X are $\{4, 2, 3, 6\}$.
- These are in L, so we can continue with:

$$X = \{0, 2, 4, 7, 10\} \qquad L = \emptyset$$

- Now L is empty, hence we have found a solution:

$$X = \{0, 2, 4, 7, 10\}, \quad \Delta X = \{2, 2, 3, 3, 4, 5, 6, 7, 8, 10\}$$

Example of Skiena's Algorithm (3)

$$X = \{0, 2, 7, 10\} \qquad L = \{2, 3, 4, 6\}$$

- Largest distance 6. Candidates: 6 and $10 - 6 = 4$.
- Try candidate 4. Distances to X are $\{4, 2, 3, 6\}$.
- These are in L, so we can continue with:

$$X = \{0, 2, 4, 7, 10\} \qquad L = \emptyset$$

- Now L is empty, hence we have found a solution:

$$X = \{0, 2, 4, 7, 10\}, \quad \Delta X = \{2, 2, 3, 3, 4, 5, 6, 7, 8, 10\}$$

Skiena's Algorithm in Pseudo-Code

Algorithm PartialDigest

Input: distance set L with $\binom{n}{2}$ numbers
Output: all sets X with $\Delta X = L$

$M \leftarrow$ maximum element in L
delete M from L
$X \leftarrow \{0, M\}$
`Place`(L, X, M)

Here, `Place` is a recursive method that tries to extend the partial solution consisting of point set X (with smallest point 0 and largest point M) and unused distances L into a complete solution.

The Recursive Method Place

Algorithm Place(L, X, M)

if L is empty **then**
    output X
    return

$y \leftarrow$ maximum element in L
if $\Delta(y, X) \subseteq L$ **then**
    Add y to X and remove $\Delta(y, X)$ from L
    `Place`(L, X, M)
    Remove y from X and insert $\Delta(y, X)$ into L
if $\Delta(M - y, X) \subseteq L$ **then**
    Add $M - y$ to X and remove $\Delta(M - y, X)$ from L
    `Place`(L, X, M)
    Remove $M - y$ from X and insert $\Delta(M - y, X)$ into L
return

Explanation of Notation

- $\Delta(y, X)$ is the multiset of distances from y to all points in X, for example:

$$\Delta(2, \{1, 3, 4, 5\}) = \{1, 1, 2, 3\}$$

- $\Delta(y, X) \subseteq L$ means that all elements of $\Delta(y, X)$ occur in L with at least the same multiplicity. For example,

$$\{1, 1, 2, 3\} \subseteq \{1, 1, 1, 2, 2, 3, 4, 5\} \text{ is true}$$

$$\{1, 1, 2, 3\} \subseteq \{1, 2, 3, 4\} \text{ is false.}$$

- Removing $\{1, 1, 2, 3\}$ from $\{1, 1, 1, 2, 2, 3, 4, 5\}$ yields $\{1, 2, 4, 5\}$.
- Inserting $\{1, 1, 2\}$ into $\{1, 2, 3\}$ yields $\{1, 1, 1, 2, 2, 3\}$.

Remarks

- The method `Place` changes L and X before it makes a recursive call.
- It undoes the changes when the recursive call returns.
- Therefore, L and X can be passed by reference, i.e., there is no need to copy the contents of L and X so that the called method can work on its own copy. (This is default behaviour for Java objects.)
- L and X have the same contents just before a recursive call to `Place` and immediately after the call returns.
- For storing X and L, balanced search trees are a suitable data structure. For easier (but less efficient) implementation, linked lists could be used.

Analysis of Skiena's Algorithm

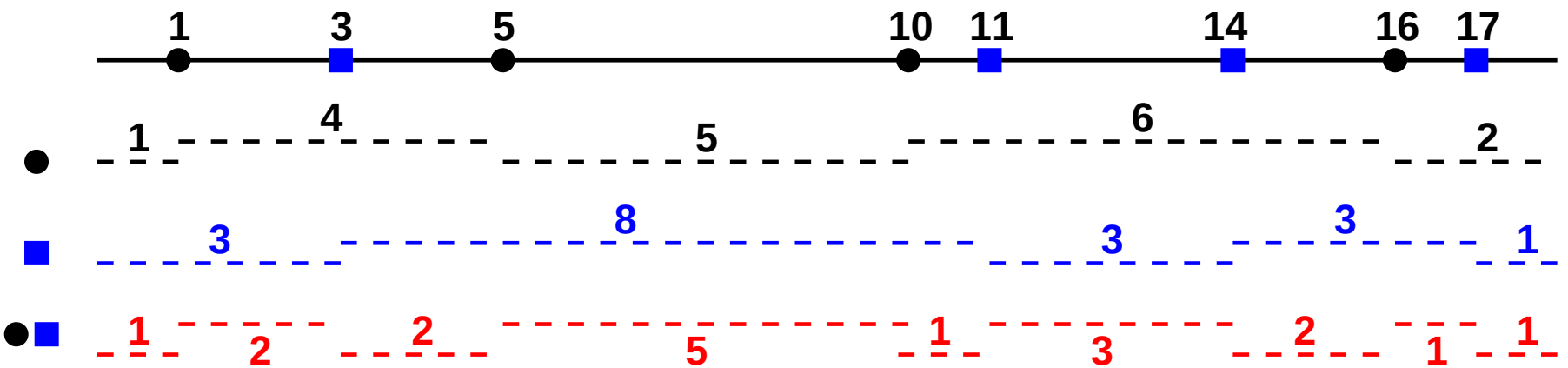
- In the worst case, the algorithm tries two possible candidate positions for every point, leading to a factor of 2^n in the running-time.
- In the best case, always only one of the two possible positions y and $M - y$ satisfies $\Delta(., X) \subseteq L$. Then the running-time is roughly quadratic.
- In practice, the algorithm performs very well and produces solutions very quickly.
- However, one can construct worst-case instances where the running-time of the algorithm is indeed exponential.
- **Open research problem:** Find an algorithm for PDP with running-time polynomial in n.

Double Digest Mapping

- Another approach (experimentally simpler) to restriction mapping is double digest mapping.
- Uses two different restriction enzymes.
- Perform three complete digestions: one with the first restriction enzyme, one with the second restriction enzyme, and one with both restriction enzymes.
- In each of the three experiments, measure length of resulting DNA fragments using gel electrophoresis.

Double Digest Problem

- Example of double digest experiment:



- **Double Digest Problem:** Given the three sets of distances (from three digestion experiments), reconstruct the points.
- Problem is known to be *NP*-hard.
- Exhaustive search algorithms and heuristics.