CS 176

Exact String Matching

Def) String: ordered set of characters from alphabet Σ.

Def) Substring: P is a substring of T iff $\exists 1 \le k \le |T|$, $P[i] = T[k+i]$.

Def) Prefix: Substring that starts at the beginning of T

Def) Suffix: " ends at the end of T

Exact Matching: Given P and T, find all occurunces of P in T.

↳ Naïvely, compare every position until it fails → $O(mn)$ time
$(m := |P|, n := |T|)$

Idea: Take advantage of repeating patterns in P and T

↳ Z-Algorithm: every character is compared a constant # of times

Def] Z function: $\forall 2 \le i \le |S|$, $Z_i(S) :=$ longest substring from i matching prefix

ex) S = T T A T X T T T A T

Z(S) = / 1 0 1 0 2 4 1 0 1

↳ How can this help compute the exact string matching?

⇒ Define a string $S := P\$T$ where \$ is a character not in Σ. Compute Z(S), and return any occurences of P in T based on Z(S).

Observation) $\forall i > 1$, we know that:

- $Z_i(S) \leq m$.
- If $i \leq m+1$, then $Z_i(S) < m$.
- For $i > m+1$, $Z_i(S) = m$ iff P occurs in T at position $i-(m+1)$

Def) Z-box: Z-box $i := S[i \ldots i + Z_i(S) - 1]$ (max prefix match)

↳ The rightmost Z-box's edge in any position reveals information!

Complete Z-Algorithm: $i :=$ current index, $(l,r) :=$ indices of rightmost Z-box

Initialization:

- $(l,r) \leftarrow (0,0)$, $i \leftarrow 2$
- Compute $Z_2(S)$ directly
- If $Z_2(S) > 0$, update (l,r) accordingly

Induction: $Z_2(S) \ldots Z_{i-1}(S)$ is computed, (l,r) is based on $(i-1)$.

- Case 1) Uninformative: $r < i$, and nothing was revealed
- Case 2) Informative: $i \leq r$, then we know some characters!

↳ Be careful, however, for Z-values spanning outside the Z-box!

eg) S = B B A B B B B A B A X A

Z(S) = / 1 0 2 2 4

Concretely, define $\beta := r - i + 1$, the length of the remaing Z-box.

Define $j := i - l + 1$, the matching position in the prefix Z-box.

↳ We will use $Z_j(S)$ to compute $Z_i(S)$.

⇒ Case 1) $Z_j(S) < \beta$: $Z_i(S) \leftarrow Z_j(S)$ ($Z_i(j)$ is contained in Z-box)

Case 2) $Z_j(S) > \beta$: $Z_i(S) \leftarrow \beta$ (outside the Z-box differs)

Case 3) $Z_j(S) = \beta$: direct compare from $(r+1)$, update (l, r)

Time Complexity $\simeq$ # of character comparisons

$=$ # mismatches + # matches (every match proceeds r)

↳ $O(|S|)$ for every termination ↳ only happens after the right most Z-box → $O(|S|)$

⇒ Time(Z-Algorithm) $= O(|S|) = O(m+n)$

Space Complexity: we only need the immediate $|P|$ values of T!

Limitations: Rigid (no variations), Complexity blows up after 1-to-1

↳ Many-to-1 takes $O(kn + km)$ time for k queries of len m

Next time: How to get rid of the factor k? (use preprocessing!)

Exact String Matching II: Suffix Trees

Idea: use preprocessing to obtain a flexible data structure

↳ after preprocessing, queries will be constant time

Intuition: All substrings are a prefix of some suffix.

Naive Algo: enumerate all suffixes of T, test if P is their prefix

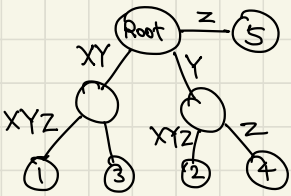
↳ Inefficient, quadratic time

Ideally, if any two suffixes share a prefix, store only once!

Def) Suffix Tree: Rooted directed tree with n leaves, each corresponding to a suffix.

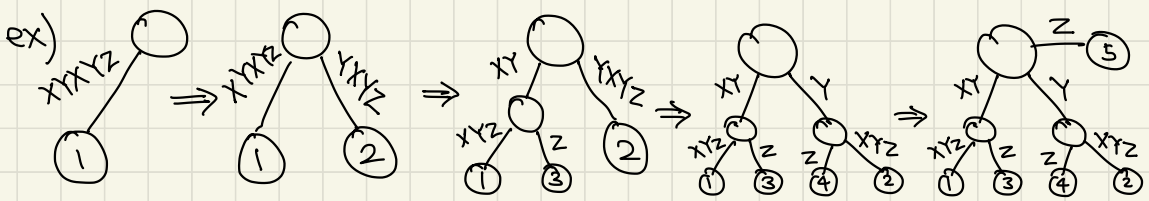
- Every edge is labeled by a nonempty substring of T
- A path to a leaf spells out a unique suffix
- Outgoing edges from the same node cannot start with the same character
- Each non-root interior vertex has at least 2 child nodes

ex) T = XYXYZ



Construction: Add suffixes from start to end, one by one.

ex)



Observation) 1) Tree → suffix is trivially easy.

2) Paths to leaves is the complete list of suffixes.

3) Paths are "compact"; if suffixes share prefixes, they share a parent.

* Not all strings have a valid suffix, specifically if ∃ indices i, j s.t. suffix i is a prefix of suffix j

Space Complexity: $O(|T|)$ ($O(\#$ of leaves in tree$)$)

Time Complexity: $O(|T|^2)$ (repeated character except last)

* There is a $O(|T|)$ construction; refer Gusfield.

⇒ The suffix tree helps solve exact matching and LCA & LCS.

Exact Matching: Traverse the tree until desired prefix is found.

All leaves of its subtree are suffixes that have a matching prefix.

Time Complexity: Time traversing P (finding P) + Time enumerating occurences (after finding P)

$\simeq \#$ char comparisons $= O(|P|)$ $\simeq \#$ of occurences in T

⇒ Overall $O(|P|+k)$ where $k := \#$ of occurences of P in T

↳ For many-to-1 searches, suffix trees runs in $O(|T| + \sum_{i=1}^{q} |P_i| + k_i)$

Longest Common Ancestor: Find the lowest common ancestor!

Runtime: Naïvely, traverse the tree for $O(|T|)$ time.

↳ There is a constant time retrival structure for LCA in trees.

⇒ Overall $O(|T| + q)$ where $q := \#$ of queries.

Longest Common Substring: Define new string $Z := S_1 \# S_2 \$$ to prevent the LCS spanning both S_1 and S_2.

↳ we must "trim down" labels that spans S_1 and S_2 and denote from which string it originated from!

⇒ find the "deepest" vertex that is a joint ancestor of both S_1 & S_2

↳ recursively check if childrens have different origins (bottom-up)

⇒ Time $O(|S_1| + |S_2|)$, can be generalized to n strings

* Complexity for suffix trees can be deceiving due to large constants

Exact Matching III: Suffix Arrays

"Do we really need to store the entire suffix tree?"

Def) Lexicographical Ordering: ordering of alphabet in Σ

↳ Assume that deliminters ($) precede all alphabet.

Intuition: Let $\Omega = \{i_1, \ldots, i_k\}$, the set of matching suffixes with P

Claim: any suffix j not in Ω must be lexicographically larger than or smaller than all suffixes in Ω.

Proof of Claim: If j is between some i_α and i_β, it must be in Ω. ∎

⇒ All possible sets of prefix matches will be clustered!

Def) Suffix Array: $A_T[i]$:= start position of lexico. smallest suffix

How to build the suffix array?

1) Merge sort: $n\log(n)$ comparisons, $O(n)$ per comp → $O(n^2 \log n)$ time

2) Use Suffix Tree: DFS the tree in lexico. order! → $O(n)$ time, space

Algorithm) Exact matching with Suffix Array:

- build the suffix array.
- binary search the array to find the L and R window.
- return all indices between L and R.

Runtime: T(binary search) + T(enumerating between L&R)

↳ $O(\log(|T|)\cdot|P|)$ ↳ $O(k)$ ⇒ total $O(\log(|T|)\cdot|P| + k)$

Speedup: If $A_T[L]$ and $A_T[R]$ share a prefix, we can ignore it.

Even better: Gusfield proposes a $O(|P|)$ comparison scheme!

↳ Practically, runtime looks like $O(\log(|T|) + |P| + k)$.

* Can we construct suffix arrays without suffix trees?

↳ Yes, KS Algorithm [2006] is linear, using mergesort & radixsort!

Algorithm) Radix Sort: set of n strings of length $\leq k \rightarrow O(nk)$ time

↳ Sort by the first character, put ties into bins, recurse on them.

⇒ Naïvely, radix sort gives $O(n^2)$ for suffix arrays ($\because k=n$).

Intuition) What if we can sorting only the first k chars of suffixes?

↳ if k is a constant, radix sort will run in $O(k(|T|) = O(|T|)$.

What if there are ties? → consult results from $(i+1)$ vs $(j+1)$ or $(i+2)$ vs $(j+2)$ or ... or $(i+k)$ vs $(j+k)$!

Naive Attempt: sort 3-suffixes via radix sort. tiebreak by looking at the lexico. order of the next 3 subsequent suffixes. In fact, replace all suffixes with lexico. order of 3 subsequent suffixes.

Runtime: $O(n^2)$ since n iteration takes $O(n)$ time.

Intuition) Divide & Conquer: create 3 copies of the string, starting from positions 1, 2, and 3. Order two out of three using recursive radix sort, then efficiently sort & merge with the remaining third.

* Details for KS is omitted in this note.

Suffix Array as a Permutation: $\exists$ an inverse Suffix Array A_S^{-1}!

↳ KS Algorithm actually wants to construct A_S^{-1} instead of A_S

Burrows-Wheeler Transform

Motivation) Efficient compression with invertability

Naive compression: count the # of repeated characters

↳ works well with low-entropy strings, not so well with higher ones.

Lexicographical sorting: efficient, but not invertible.

BWT: list of characters that precedes each row in a SA.

Intuition: Strings often have patterns, i.e. some sequences are repeated

↳ Thus, BWT is likely to contain long repeated characters!

e.g. 'h' is likely to be preceded with 't', so repeated 't's are expected

Construction of BWT: Consider the cyclic rotations of S. Sorting these yields a suffix array with nice properties containing BWT.

Def) BWT: $\pi(S)$:= rotations of S, $\pi^s(S)$:= lex. sorted list of $\pi(S)$

F, L := first & last columns of $\pi^s(S)$. ← which can be reconstructed from L as with S.

↳ F is simply the ordered list of characters, and L is the BWT. (& pointer to \$)

* Using KS algorithm, BWT can be constructed in $O(n)$.

Claim: we can unambiguously reconstruct S using only L (and F).

↳ In other words, BWT is invertible.

Key Obs: F and L are order-preserving!

Proof of Claim: all ties are broken by their next permutations, which are exactly the strings that end with that character! Then, since the SA is already sorted, appearance in L is order-preserved.

ex) $\underbrace{A\$BARBAR}_{S_1} < \underbrace{ARA\$BARB}_{S_2} \Leftrightarrow \underbrace{\$BARBARA}_{S_1} < \underbrace{RA\$BARBA}_{S_2}$

Reconstruction from L: Obviously F[1] is \$, and the last character in S is also \$. Then, we can chase down its preceding character, L[1]. Next, find the first occurence of L[1] in F, and let its index be j. Chase down the next one, L[j]. Continue finding the n-th occurence of L[i] in F, j. L[j] is the next preceding character, and update i ← j.

Efficiency: to find the m-th copy of character k, keep M, the first index for each character in F & C, the # of occurences of L[r] in string L[1:r-1]. ⇒ M[L[r]] + C[r] is the index of r in F!

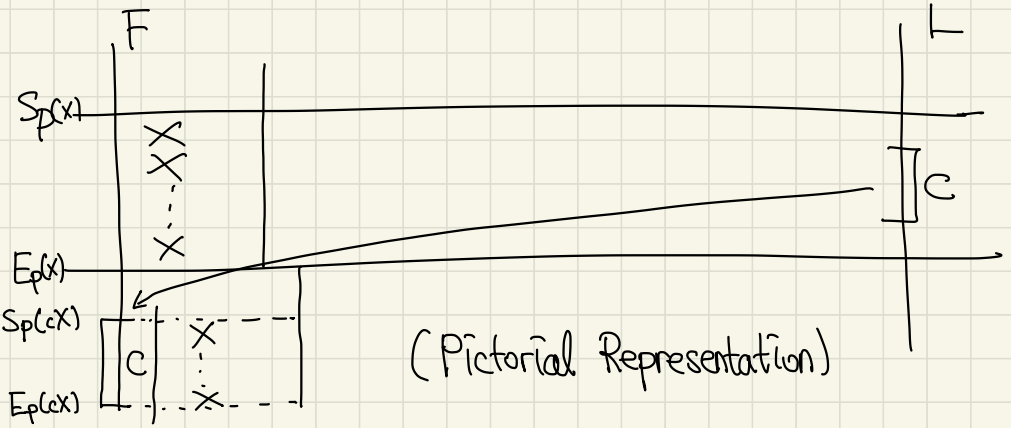
* There also exists a "forward decoding" that is equivalent.
↳ Intuitively, we chase down characters in F instead of L!

Querying BWT

The Perfect Match Problem: Is BWT any useful for this?
↳ yes, BWT compactly stores the information of the suffix array!

Idea: scan the query string backwards and update pointers Sp & Ep that points to the first & last matching rotations in $\pi^s(T)$.

Update Rules: For a query of length 1, consult M
↳ $Sp(c) \leftarrow M[c]$, $Ep(c) \leftarrow M[next(c)]-1$.
Then, look at $L[Sp]$ and $L[Ep]$ for their matching occurrence in F again using M and OCC, where $OCC[c,i]$:= # of occurrences of character c in the first i positions in L.
↳ $Sp(cX) \leftarrow M[c] + OCC[c, Sp(X)-1]$
$Ep(cX) \leftarrow M[c] + OCC[c, Ep(X)] - 1$
Why does this work? OCC increases on the interval $[Sp(X), Ep(X)]$ as long as $L[i]=c$, the next character we are chasing!



(Pictorial Representation)

⇒ This solves half of the problem: the # of occurrences of P in T.

↳ If P is exhausted, output$(E_P - S_P + 1)$, if $E_P < S_P$ output NULL.

Claim: If X occurs in T, then cX occurs in T if $S_P(cX) \leq E_P(cX)$.

Proof: $OCC[c, E_P(X)] = OCC[c, S_P(X)-1]$ (c never occurs btwn $[S_P, E_P]$)

$\Rightarrow M[c] + OCC[c, E_P(X)] - 1 \leq M[c] + OCC[c, S_P(X)-1]$

$\Rightarrow E_P(cX) < S_P(cX)$. //

Runtime: $O(|P|)$, but still need to find locations of matching

(also OCC is very demanding in space)

Locating Matches: Simply return $\{A_T[S_P], \ldots, A_T[E_P]\}$

... but this defeats the purpose of not storing A_T!

Idea: Only store some values of A_T, then recover as necessary.

Sparse Storage of A_T: evenly store a small fraction (~2%) of A_T.

↳ If $A_T[i]$ is not found, consult $j = M[C] + OCC[C, i-1]$, which is the index of cX if i corresponds to Xc. Then, if $A_T[j]$ is found, return $A_T[j]+1$ for $A_T[i]$. Else, recurse on $i \leftarrow j$.

⇒ This introduces a time vs space tradeoff on how much to store!

Next Idea: Approximately matching strings!

Inexact String Matching

Why? Sequencing Errors & Genetic Variation

Given two strings: overall level of similarity, which parts are similar?

Pairwise Global Alignment ⇒ Inserting gaps to enforce matching

Events: Substitution $\binom{A}{C}$, Insertion $\binom{C}{-}$, Deletion $\binom{-}{C}$ ... Descendant ... Ancestor

↳ Between siblings, insertion/deletion is called indels

Scoring Alignment: Total Score = $\sum$(score of aligned pair) + $\sum$(score of indel)

↳ amino acids $C_1 \leftrightarrow C_2$ will have a higher substitution score if that substitution happened frequently in evolution (assumption)

Score := $\log\left(\frac{Pr[C_1 \to C_2 | R]}{Pr[C_1 \to C_2 | U]}\right)$ where R := related, U := unrelated

$= \log\left(\frac{p_{C_1,C_2}}{q_{C_1} \cdot q_{C_2}}\right)$ where p_{C_1,C_2} := freguencies of C_1 & C_2 both in a same index,

q_{C_1}, q_{C_2} := marginal probabilities of C_1 and C_2

$= \sigma(C_1, C_2) \propto$ amount of expected $C_1 \to C_2$ mutations

$\Rightarrow$ Score $= \sum_{\text{aligned}(C_1,C_2)} \sigma(C_1, C_2) + \sum(\text{indel score})$ (<0)

Indel scoring: Linear → treat every indel independently: ($W_s \times$ (# of indels))

Affine → treat consecutive indels as a single event:

$\left(\sum_{\text{gaps}} (W_g + W_s \cdot (\#\text{ of indels in the gap}))\right)$ where $|W_g|$(gap penalty) $\gg |W_s|$.

$\Rightarrow$ Score $= \sum_{\text{aligned}(C_1,C_2)} \sigma(C_1, C_2) + (\#\text{gaps}) \times W_s = \sum_{(C_1,C_2)} \sigma(C_1, C_2)$ where $\sigma(C, -) = W_s$.

Global alignment: Given strings x, y and σ, W_s, find maximal alignment.

↳ Use DP through defining appropriate subproblems!

Claim: If V_{ij} is the value of optimal alignment for $X[1:i], y[1:j]$, then V_{ij} can be derived trivially from $\{V_{i-1,j}, V_{i,j-1}, V_{i-1,j-1}\}$. → find V_{nm}!

Algorithm) Needleman-Wunsch: $X, y, \sigma, W_s \to V(n,m)$ $(|x|=n, |y|=m)$

Base case: $V(0,0) = 0$, for $0 < i \le n$: $V(i,0) = \sum_{k \in [i]} \sigma(X[k], -) = i \cdot W_s$

$V(0,j) = \sum_{k \in [j]} \sigma(-, y[k]) = j \cdot W_s$.

Recursion: $V(i,j)$ can end with 1) $(X[i], y[j])$ 2) $(X[i], -)$ 3) $(-, y[j])$

Then, we need information from 1) $V(i-1, j-1)$ 2) $V(i-1, j)$ 3) $V(i, j-1)$ to calculate each possible V_{ij} score.

Take the max of: $\{V(i-1,j-1) + \sigma(x_i, y_j), V(i-1,j) + \sigma(x_i, -), V(i,j-1) + \sigma(-, y_j)\}$ ($\sigma(x_i,-) = W_s$, $\sigma(-,y_j) = W_s$)

→ Fill the $n \times m$ matrix, then backtrack pointers (path not unique!)

Runtime: if $n = O(m)$, total $O(n^2)$ time.

Space: if $n = O(m)$, naively $O(n^2)$, can reuse one row for $O(n)$ space.

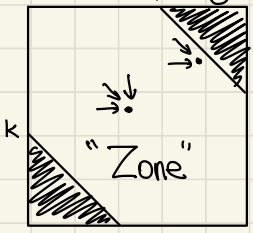
↳ can also do a diagonal zig-zag fill out, which is parallelizable!

Variations on String Matching I

1) K-bounded Alignment: allow at most k mismatches (sub & indels)

Assume edit distance $\sigma(n,m) = \begin{cases} 0 & \text{if } n = m \\ 1 & \text{if } n \neq m \text{ OR Gap.} \end{cases}$ Obviously, $|n-m| \leq k$.

Idea: only fill in the relevant portions of the matrix V.



Border cells ignore cells outside the zone.

However, a path in the zone can still have more than k alignment. ⇒ If $V(n,m) \leq k$, trace back.

Otherwise, there is no k-bounded alignment (iff relation!)

$\text{Area}(\text{Zone}) \simeq nm - 2 \cdot \frac{(n-k)(m-k)}{2} = k(m+n) - k^2 \rightarrow O(m+n) = O(n)$ time!

2) Affine Gap Penalty: treat consecutive indels as single events

→ Score $= \sum_{(c_1,c_2)} \sigma(c_1,c_2) + \sum_{x:\text{gap}} W_{gap} + |X| \cdot W_s$.

↳ NW algorithm no longer applies since we introduce dependencies!

(We might want to go back and fix previous indices now)

Naive Algorithm: Consider all possible consecutive gaps → $O(n^3)$

Idea: keep track of three possibilities (→ matrices) for preceding solutions

$M(i,j)$:= best alignment of $x[1:i], y[1:j]$ ending with x_i & y_j match

$I_x(i,j)$:= // ending with x_i & gap

$I_y(i,j)$:= // ending with gap & y_j

$V(i,j) \leftarrow \max\{M(i,j), I_x(i,j), I_y(i,j)\}$.

$$\Rightarrow I_x(i,j) = \max \begin{cases} M(i-1,j) + W_g + W_s \\ I_x(i-1,j) + W_s \\ I_y(i-1,j) + W_g + W_s \end{cases}$$

$$\Rightarrow I_y(i,j) = \max \begin{cases} M(i,j-1) + W_g + W_s \\ I_x(i,j-1) + W_g + W_s \\ I_y(i,j-1) + W_s \end{cases}$$

$$\Rightarrow M(i,j) = \max \begin{cases} M(i-1,j-1) + \sigma(x_i, y_j) \end{cases}$$

$$\begin{cases} I_x(i-1,j-1) + \delta(x_i, y_j) \Leftrightarrow V(i-1,j-1) + \delta(x_i, y_i) \\ I_y(i-1,j-1) + \delta(x_i, y_j) \end{cases}$$

Base Cases: $M(0,0) = I_x(0,0) = I_y(0,0) = 0$.

$I_x(i,0) = W_g + i \cdot W_s$, $I_y(i,0)$ & $M(i,0)$ are not defined. $(0 < i \leq n)$

$I_y(0,j) = W_g + j \cdot W_s$, $I_x(0,j)$ & $M(0,j)$ are not defined. $(0 < i \leq m)$

Variations on String Matching II

1) Local Alignment: alignment of substrings of X and Y

↳ find the highest scoring alignment, considering all substrings of X&Y

* makes sense when X and Y are very distant or have very different length

Pairwise Local Alignment: outputs substrings α and β, naïve solution is to enumerate all $O(n^2m^2)$ pairs of substrings → $O(n^3m^3)$ runtime

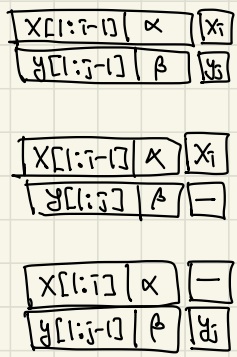
Intuition: {all suffixes of all prefixes} = {all substrings}

↳ V_{ij} := best alignment score between a suffix of $X[1:i]$ and $Y[1:j]$.

⇒ best local alignment is $\max_{i,j} \{V_{ij}\}$. [Smith-Waterman]

Recursion:

$$V(i,j) := \max \begin{cases} V(i-1,j-1) + \sigma(x_i, y_j) \\ V(i-1,j) + W_s \quad (\sigma(x_i, -)) \\ V(i,j-1) + W_s \quad (\sigma(-, y_j)) \\ 0 \text{ (is selected when all above are negative)} \end{cases}$$



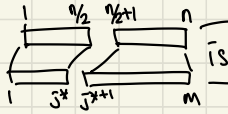
Base Case: $V(0,0) = V(i,0) = V(0,j) = 0$.

* After taking $\max_{i,j}\{V_{ij}\}$, backtrack until we hit a 0 (restart point).

2) Linear Space Global Alignment: How to recover with only the score?

Idea: Record "midway" points, then recursively continue. [Hirschberg]

↳ How to find where it passes at $i = n/2$?

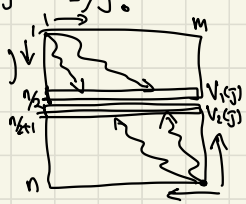


Intuition: $\exists j^*,\ V(X[1:n], Y[1:m]) = V(X[1:n/2], Y[1:j^*]) + V(X[n/2+1:n], Y[j^*+1:m])$

↳ in other words, $j^* = \arg\max_j \{ \underbrace{V(X[1:n/2], Y[1:j^*])}_{(V_1(j))} + \underbrace{V(X[n/2+1:n], Y[j^*+1:m])}_{(V_2(j))} \}$.

→ Compute NW with $X[1:n/2]$ and Y → last row gives $V_1(j)$

For the bottom half, find NW of $X[n:n/2+1]$ (↳ Reversed!) and Y^R → $V_2(m-j)$



⇒ $j^* = \arg\max_j \{V_1(j) + V_2(j)\}$, path passes through point $(n/2, j^*)$!

↳ For the rest of the path, recursively find smaller midpoints until M is small.

Space: $O(m+n) = O(n)$ (no saving entire matrix)

Time: First iteration $\rightarrow O(nm)$

Second iteration $\rightarrow O(\frac{nm}{2})$

Third iteration $\rightarrow O(\frac{nm}{4})$

... i-th iteration $\rightarrow O(\frac{nm}{2^{i-1}}) \Rightarrow$ Overall $O(nm \sum_{i=0}^{\log(n)} \frac{1}{2^{i-1}}) = \underline{O(nm)}$. //

$\Rightarrow$ We trade some constant multiplicative factor in time to reduce space requirement from quadratic to linear!

Phylogeny

Def) Phylogeny: a weighted tree capturing ancestral relationships

↳ leaves represent extant species, branching is a species divergence

↳ a rooted tree will encode a "flow of time" for mutations

Why Trees? → we only consider "splitting" and not "joining" (not always...)

Applications: 1) Taxonomy 2) Ancestral Relations 3) Human Evolution

4) Evolution within an Individual (cancer mutations)

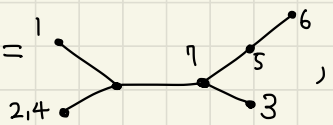
Two Algorithms: 1) Distance-Based 2) Character-Based

↳ max likelihood, max parsimony, ...

Distance-Based: Given a pairwise dissimilarity table, label a tree s.t. path lengths correspond to the dissimilarity

Parsimony-Based: Find a tree that require the minimum # of changes

Def) X-Tree: $T := (T, \varphi)$ where (T: tree, φ: label map) $\varphi(X)$ contains all vertices with degree ≤ 2, and the label set X (φ is not one-to-one nor onto)

ex) $T(T,\varphi) =$ 

, $X = [7]$ * degree 2 vertex only is introduced to put a label there!

Def) Phylogenetic Tree: X-Tree where φ is a bijection of $X \leftrightarrow T$.

Theorem) [Schröder] $B(n) := \{$all inequivalent unrooted binary phylogenetic trees with n leaves$\}$, then $|B(n)|$ is super exponential.

Def) Dissimilarity Map: function $\delta: X \times X \to \mathbb{R}$ s.t. $\delta(x,x)=0, \delta(x,y)=\delta(x,y)$.

* DM is pseudometric if it satisfies the triangle inequality, and it is metric if $\delta(x,y)=0 \Leftrightarrow x=y$.

↳ How can δ be "represented" by a tree, and how do we find it?

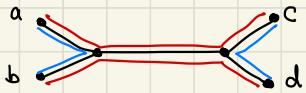
⇒ Make T with edge weight $w(e)$ s.t. $d_{(T,w)}(u,v) := \sum_{e \in path(u,v)} w(e)$.

Def) Tree Metric: δ on X s.t. $\exists T, w: E \to \mathbb{R}_{>0}$ s.t. $\delta(x,y) = d_{(T,w)}(\varphi(x), \varphi(y))$

↳ What is the necessary & sufficient condition?

Def) 4 Point Condition: $\forall a,b,c,d \in X$, $\delta(a,b)+\delta(c,d) \leq$

$\max\{\delta(a,d)+\delta(b,c), \delta(a,c)+\delta(b,d)\}$



* 4PC implies TI, δ's nonnegativity, and δ's pseudometricness.

Theorem) (Tree-Metric) For a $\delta \geq 0$ on X, 4PC $\iff$ Tree-Metric

Def) Ultrametric: $\forall a,b,c \in X$, $\delta(a,b) \leq \max\{\delta(a,c), \delta(b,c)\}$.
(↗ two are equal and ≥ than the third)

* Ultrametric $\Rightarrow$ Tree-Metric, and it arises naturally in phylogenetic X-trees.

Def) Equidistant Representation: A phylogenetic X-tree $T(T,\varphi)$ rooted at ρ and an edge weight $w: E(T) \to \mathbb{R}$ s.t.:

1) $d_{(T,w)}(\rho, \varphi(x)) = d_{(T,w)}(\rho, \varphi(y))$ $\forall x,y \in X$,

2) $w(e) \geq 0$ $\forall e \in E^{\circ}(T)$ where $E^{\circ}(T) :=$ set of interior edges of T.

↳ useful to think that current species have roughly the same # of mutations (distance) when compared with a common ancestor (root node).

Theorem) If δ is ultrametric on X, then $\exists$ an equidistant representation (T,w) such that $\delta = d_{(T,w)}$.

Def) Gromov Product: For a δ on X, fix an element $r \in X$. The Gromov product of x and y in $X \setminus \{r\}$ is:

$$\delta_r(x,y) = \frac{1}{2}[\delta(x,y) - \delta(x,r) - \delta(r,y)] \text{ if } x \neq y, 0 \text{ otherwise.}$$

Theorem) A non-negative δ on X satisfies 4PC $\Leftrightarrow$ δ_r is an ultrametric on $X \setminus \{r\}$ $\forall r \in X$. (Proof in HW, key to proving Tree Metric Theorem!)

Phylogeny Reconstruction Problem: Given δ on X, find a phylogenetic X-tree and weight function $w: E \to \mathbb{R}_{>0}$ that minimizes:

$$L(d_{(T,w)}, \delta) := \sum_{a,b \subset X} g_{ab}[\delta(a,b) - d_{(T,w)}(a,b)]^2. \quad (g_{ab} \text{ are scaling factors})$$

↳ If $T(T, \varphi)$ is given, optimizing w is in P. However, if T is unknown, this problem is NP-Complete.

Distance-Based Phylogeny

"Given a dissimilarity matrix, output a tree that is as consistent as possible."

Two greedy algorithms: 1) UPGMA (ultrametric) 2) Neighbor Joining (tree metric)

Vertex Representation of Ultrametric: Let $h: V(T) \to \mathbb{R}$ be a height function s.t. $h(u) \geq h(v)$ if u is an ancestor of v, and $d_{(T,h)} := \begin{cases} 2h(LCA_T(\varphi(x), \varphi(y))) \\ 0 \text{ if } x = y. \end{cases}$

Algo) UPGMA: Unweighted Pair Group Method with Arithmetic Mean

↳ Input: δ on X, Output: Phylogenetic X-tree T & height function h.

* Always returns an equidistant (ultrametric) tree.

Intuition: Find the "closest" pair and merge them, recursively. Then, we need to update δ based on the new supervertex (collapsing).

Algorithm: $\forall i \in X$, assign cluster $C_i := \{i\}$. Initialize $\Delta(C_i, C_j) := \delta(i, j)$. Define a leaf node $\forall i$ and assign a height of 0. Iterate:

Find $\min\{\Delta(C_i, C_j)\}$. Merge them into $C_m := C_i \cup C_j$. Assign a height of $\Delta(C_i, C_j)/2$ for a new node C_m. Update dissimilarity by

$$\Delta(C_i \cup C_j, C_k) := \frac{|C_i|}{|C_i|+|C_j|}\Delta(C_i, C_k) + \frac{|C_j|}{|C_i|+|C_j|}\Delta(C_j, C_k).$$

$$(* \Delta(C_i, C_j) := \frac{1}{|C_i| \times |C_j|}\sum_{x \in C_i, y \in C_j} \delta(x, y) \text{ originally})$$

Runtime: $O(n^3)$ (n iterations, n^2 to find the min-pair at each step)

↳ Can reduce down to $O(n^2)$ with optimization.

But what if δ is not ultrametric, i.e. one edge of a sibling is longer?

↳ UPGMA might merge the wrong nodes!

Algo) Neighbor Joining: Return the correct tree if δ is tree metric.

Intuition: Find clusters to merge like UPGMA, but the update rule is different!

Pair Selection: $\min_{(i,j)} \{Q(i,j) := \delta(i,j) - (r_i + r_j)\}$ where $r_i := \frac{1}{|L|-2} \sum_{k \in L} \delta(i,k)$. (????)

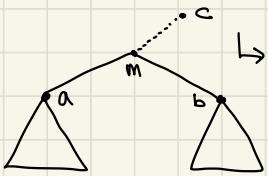
↳ $Q(i,j) \simeq$ distance btwn (i,j) minus mean distance to all other active leaves.

Theorem) If δ is tree metric, then $\arg\min_{(i,j)} \{Q(i,j)\}$ is guaranteed to be neighbors.

↳ Justifies Neighbor Joining, proof in HW. (intuitively, confidence boosting)

Update: Leverage the tree metric property to derive the rule:

$\delta(c,m) = (\delta(c,b) + \delta(c,a) - \delta(a,b))/2$ when m is a merge of a and b.



↳ $d(a,m) = \frac{1}{2}[d(a,b) + r_a - r_b]$, $d(b,m) = \frac{1}{2}[d(a,b) + r_b - r_a]$

$\forall$ active vertices c, $d(m,c) = \frac{1}{2}[d(a,c) - d(a,m)] + \frac{1}{2}[d(b,c) - d(b,m)]$

↳ intuitively, take the average of the two estimators

Runtime: $O(n^3)$.

Character-Based Phylogeny

A character on X captures attributes of the sample. ex) DNA seq. alignment

Def) Character: function $\chi: X' \to S$ where $X' \subseteq X$ and S is a set of states.

↳ X' is defined in this way to represent missing data.

	χ_1	χ_2	...
1	A	A	
2	T	C	
3	*	C	

A character χ is said to be:

1) Full if $X' = X$. 2) k-state if $|\chi(X')| = k$ (# of state outcomes)

3) Trivial if there exists at most one state with multiplicity ≥ 2.

* Nontrivial characters imply information about tree topology!

Def) Extension: for $T(T, \omega)$, an extension $\bar{\chi}: V \to S$ of χ on T has $\bar{\chi} \cdot \varphi(a) = \chi(a)$ $\forall a \in X$. (i.e. interior vertices also gets a state $\in S$)

Parsimony: Given a collection C of characters on X, provide a tree that gives the simplest explanation, i.e. minimizes the number of mutations.

Def) Changing Number: given T & $\bar{\chi}$, $ch(\bar{\chi}, T) := |\{(u,v) \in E(T) \mid \bar{\chi}(u) \neq \bar{\chi}(v)\}|$.

Def) Parsimony Score: $\pi(\chi, T) = \min_{\bar{\chi} \in \{ext(T, \chi)\}} \{ch(\bar{\chi}, T)\}$.

Small Parsimony Problem: given a rooted Phy. X-tree $T(T, \varphi)$ & a full character χ on T, find an extension $\bar{\chi}$ of χ minimizing $ch(\bar{\chi}, T)$.

Algorithm) Fitch's Algorithm: $\forall$ leaf v, $\varphi(v) := \{\chi(\varphi^{-1}(v))\}$. $\forall$ interior vertex v with children u, w, let $\varphi(v) := \begin{cases} \varphi(u) \cap \varphi(w) & \text{if } \varphi(u) \cap \varphi(w) \neq \emptyset \\ \varphi(u) \cup \varphi(w) & \text{if } \varphi(u) \cap \varphi(w) = \emptyset \end{cases}$. Then, $\pi(\chi, T)$ is the number of union operations called. To obtain an

optimal extension, pick an arbitrary state in $\psi(root)$, then for a parent p with state s_p, the child c's state will be $s_c \leftarrow \begin{cases} s_p & \text{if } s_p \in \psi(c) \\ s \in \psi(c) & \text{otherwise.} \end{cases}$

Runtime: $O(kn)$ where $k=|S|$, $n=|X|$ ($\forall v \in V$, takes $O(k)$ time to find $\psi(v)$).

Weighted Small Parsimony: $\forall i,j \in S$, a mutation $i \to j$ costs $m(i,j)$.

Algorithm) Sankoff's Algorithm: $\forall v \in V$, $A_v :=$ an array indexed by S such that $A_v[s] :=$ OPT cost for the subtree rooted at v with state $s \in S$.

Base Case: $\forall$ leaf l, $A_v[s] = \begin{cases} 0 & \text{if } s = state(l) \\ \infty & \text{otherwise} \end{cases}$ ← penalize as much as possible.

Recursion: $\forall v$ with childs u,w, $A_v[i] := \min_{j \in S}[A_u[j] + m(i,j)] + \min_{k \in S}[A_w[k] + m(i,k)]$

↳ keep a pointer for $\underset{j \in S}{\text{argmin}}$ and $\underset{k \in S}{\text{argmin}}$. Then, $\Pi(X,T) = \min_{s \in S}\{A_{root}[s]\}$.

To obtain the optimal extension, find $s_c = \underset{j \in S}{\text{argmin}}\{A_c[j] + m(s_p, j)\}$.

Runtime: $O(k^2 n)$.

Key Idea: There is a fundamental equivalence between X-trees and a certain type of collection of binary characters on X.

Def) X-Splits: a partition of X into two nonempty subsets $(A|B)$, that is, 1) $A \cup B = X$, 2) $A \cap B = \emptyset$, 3) $A \neq \emptyset, B \neq \emptyset$.

↳ Removing an edge e introduces an X-split (a cut).

Claim: For a given X-tree, no two different edges induce the same X-split.

Proof: If so, we will have an unlabeled degree 2 vertex, which is invalid. //

Def) Compatibility: two X-splits $A_1|B_1$ and $A_2|B_2$ if at least one of $A_1 \cap A_2, A_1 \cap B_2, B_1 \cap A_2, B_1 \cap B_2$ are empty. Otherwise, they are incompatible. → all four "quadrants" are nonempty.

	A_1	B_1
A_2	$A_1 \cap A_2$	$B_1 \cap A_2$
B_2	$A_1 \cap B_2$	$B_1 \cap B_2$

Def) 4-Gamete Test: If all four combinations {00, 01, 10, 11} appear in the two columns corresponding to a given pair of binary characters, they are incompatible.

Theorem) Splits-Equivalents [Buneman, 1971]: Let C be a collection of X-splits. Then, there exists an X-tree $T(T, \varphi)$ s.t. $\Sigma(T) = C$ iff
↳ = full binary characters
the splits in C are pairwise compatible. Furthermore, the X-tree is unique upto isomorphism. ($\Sigma(T) = \{X \text{ split}(e) \mid e \in E(T)\}$)

Def) Convexity: a character $\chi: X' \to S$ on $T(T, \varphi)$ s.t. ∃ an extension $\overline{\chi}: V \to S$ of χ s.t. $\forall s \in S$, the subgraph $T := \{v \in V \mid \overline{\chi}(v) = s\}$ is connected, i.e. we can label inner vertices such that each subgraph consisted of the same state is connected.

Remark: a binary character χ on T if $\exists$ a bipartition of X induced by χ is an X-split induced by an edge of T. (and that edge is unique)

→ Biologically, homoplasy-free (no recurrent or back mutations)

Def) Compatible: a collection C of characters on X s.t. $\exists$ an X-tree on which all of the characters in C are convex. (generalized to k-states)

Def) Perfect Phylogeny Problem: Determine whether C is compatible, and if so, find the X-tree on which they are convex.

↳ For the binary case, C is compatible iff pairwise compatible!

Naive Algorithm: If $|X|=n$, $|C|=m$, $O(nm^2)$ for checking all pairs.

Gusfield's $O(nm)$ Algorithm: Assume that root = all-zero string.

1) View each column as an n-bit binary number, radix sort in decreasing order.
2) $\forall$ row r, let $k_r :=$ string of labels of columns with 1s in row r + \$ at the end.
3) Build a "keyword tree" for strings $k_1, k_2, \ldots, k_n$, akin to suffix trees.
4) Check whether each column label appears exactly once. If so, the tree is correct upto permutation of labels on the same edge.

Why does this work? → For each character χ_i, let $O_i := \{x \in X \mid \chi_i(x) = 1\}$.

If $\exists$ a perfect phylogeny solution, then after radix sorting, if character χ_i appears to the left of χ_j, then either $O_j \subseteq O_i$ or $O_j \cap O_i = \emptyset$.

For the unknown root case, take majority string as the root, and swap $1 \to 0$.

↳ If data is consistent, the majority string will appear somewhere in the tree!

For $k \geq 3$, we check for compatibility via convexity (pairwise not sufficient)

↳ Necessary & Sufficient condition is a chordal completion of $\text{Int}(C)$, the partition intersection graph $\left((\chi_a, B_{ai}), (\chi_b, B_{bj})\right) \in E(\text{Int}(C))$ if $B_{ai} \cap B_{bj} \neq \emptyset$.

$\Rightarrow$ full characters run in $O(2^{2k} n m^2)$ time, any partial is NP-C.

Probability

Probability Space: $(\Omega, \mathbb{F})$, sample space (set) and σ-field (certain subsets of Ω) (or σ-algebra).

Probability Measure: $\mathbb{P}$, a function $\mathbb{F} \to [0,1]$ with such conditions:

1) $\mathbb{P}[\emptyset] = 0$. 2) $\mathbb{P}[\Omega] = 1$. (normalization)

3) $A_1, A_2 \ldots \in \mathbb{F}$. $A_i \cap A_j = \emptyset \ \forall i \neq j$. Then, $\mathbb{P}[\bigcup_{i=1}^{\infty} A_i] = \sum_{i=1}^{\infty} \mathbb{P}[A_i]$ (additive)

Random Variable: X, a function $\Omega \to \mathbb{R}$ s.t. "$X \leq x$" $:= \{\omega \in \Omega \mid X(\omega) \leq x\} \in \mathbb{F}$ $\forall x$.

Cumulative Distribution Function: $F_X(x) := \mathbb{P}[X \leq x]$.

Convergence in Distribution: seq. $X_1, X_2, \ldots$ and X. $X_n \xrightarrow{d} X$ as $n \to \infty$ if

$\lim_{n\to\infty} F_{X_n}(x) = F_X(x),\ \forall x \in C(F_X)$ i.e. points in $\mathbb{R}$ where F_X is continuous.

Central Limit Theorem: $X_1, X_2, \dots$ i.i.d with finite mean μ, finite variance σ^2.

Let $S_n := X_1 + \cdots + X_n$. $\frac{\sqrt{n}}{\sigma}\left(\frac{S_n}{n} - \mu\right) \xrightarrow{d} N(0,1)$ as $n \to \infty$.

Conditional Probability: Let B be some event s.t. $\mathbb{P}[B] > 0$. E_1, E_2 are events.

We want to define a new prob. space $(B, \mathbb{F}_B, \mathbb{P}_B)$ s.t. $\mathbb{P}_B$ is consistent.

$\mathbb{P}[B \cap E_1] / \mathbb{P}[B \cap E_2] = \mathbb{P}_B[B \cap E_1] / \mathbb{P}_B[B \cap E_2]$ (relative size)

$\Rightarrow \mathbb{P}_B = c\mathbb{P}$ where c is a constant $\frac{1}{\mathbb{P}[B]}$. $\Rightarrow \mathbb{P}[A|B] := \mathbb{P}_B[B \cap A] = \frac{\mathbb{P}[B \cap A]}{\mathbb{P}[B]}$.

Independence: $A \perp\!\!\!\perp B \iff \mathbb{P}[A|B] = \mathbb{P}[A], \mathbb{P}[B|A] = \mathbb{P}[B]$.

$\Rightarrow$ If $A \perp\!\!\!\perp B$, $\mathbb{P}[A \cap B] = \mathbb{P}[A]\mathbb{P}[B]$.

Theorem) Law of Total Probability: Let $B_1, B_2, \dots \in \mathbb{F}$ be a partition of Ω and $A \in \mathbb{F}$. Then, $\sum_{i=1}^{\infty} \mathbb{P}[A|B_i]\mathbb{P}[B_i] = \sum_{i=1}^{\infty} \mathbb{P}[A \cap B_i] = \mathbb{P}[\bigcup_{i=1}^{\infty}(A \cap B_i)]$

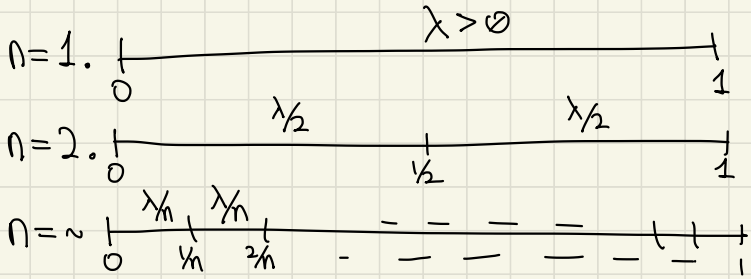
$= \mathbb{P}[A]$ by additivity of $\mathbb{P}$. //

Stochasic Process: $I \sim Ber(p)$ where $\mathbb{P}[I=0] = p$, $\mathbb{P}[I=1] = (1-p)$.

$I_1, I_2, \dots$ i.i.d. $Ber(p)$. $S_n :=$ # of 1s upto n trials $= \sum_{i=1}^{n} I_i \sim Bi(n, p)$.

$W_i :=$ waiting time of next 1 $\sim Geo(p)$. $\mathbb{P}[W_i = k] = (1-p)^{k-1}p$, $k \in \mathbb{N}^+$.

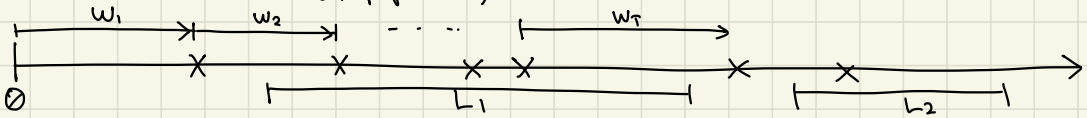
↳ How do we think about continuous processes?



$Y_n :=$ # of successes $\sim Bi(n, \lambda/n)$. $Y_1, Y_2, \dots Y_n \xrightarrow{d} Po(\lambda)$ as $n \to \infty$.

Similarly, $G_n \sim Geo(\frac{\lambda}{n})$, then $X_n := \frac{G_n}{n} \xrightarrow{d} Exp(\lambda)$ as $n \to \infty$.

Poisson Point Process: $PPP(\lambda)$, $\lambda > 0$.



Each $W_i \sim Exp(\lambda)$. $N_i :=$ # of hits in interval $L_i \sim Po(\lambda \cdot |L_i|)$.

↳ This will be useful in describing mutation events!

Markov Chains

Def) Discrete Time Markov Chain: $\{X_n \mid n = 0, 1, \dots\}$, $X_n : \Omega \to S$. (finite) (state space)

$$\mathbb{P}[X_n = j \mid X_0 = i_0, \dots, X_{n-1} = i_{n-1}] = \mathbb{P}[X_n = j \mid X_{n-1} = i_{n-1}] \quad \forall n \in \mathbb{N}, \forall i_0, \dots, i_{n-1}, j \in S.$$

The one-step $\mathbb{P}[X_{n+1} = j \mid X_n = i] = p_{ij}$ $\forall n \in \mathbb{N}$ is homogeneous, and the elements can be represented as a matrix $P[i,j] = p_{ij}$.

Some Notations: $\vec{u}_n = (\mathbb{P}(X_n = i))_{i \in S}$. How to relate this to $\vec{u}_{n-1}$?

$\mathbb{P}(X_n=j) = \sum_{i\in S} \mathbb{P}(X_n=j \mid X_{n-1}=i) \cdot \mathbb{P}(X_{n-1}=i) = \sum_{i\in S} p_{ij} \cdot \mathbb{P}(X_{n-1}=i)$

↳ $\vec{u}_n = \vec{u}_{n-1} P \to \vec{u}_n = \vec{u}_{n-2} P^2$, and so on... $\to \vec{u}_n = \vec{u}_0 P^n$.

Denote $P^n[i,j] = p_{ij}^{(n)}$.

Chapman-Kolmogorov: $p_{ij}^{(n+m)} = \sum_{k\in S} p_{ik}^{(n)} \cdot p_{kj}^{(m)}$ (total probability again)

Def) Accessible: j is accessible from i, $(i \to j)$, if $\exists n \in \mathbb{N}$ s.t. $p_{ij}^{(n)} > 0$.

Def) Communicate: $(i \leftrightarrow j)$ iff $(i \to j) \wedge (j \to i)$.

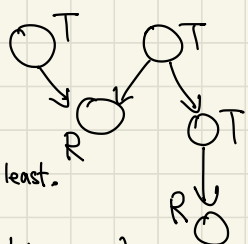
Def) Strongly Connected Component: Set of states s.t. $\forall i,j \in S$, $(i \leftrightarrow j)$.

↳ We can decompose states into SCCs & topologically sort them into DAGs.

Def) Transient Component: All states not in a sink SCC.

Def) Reccurent Component: All states in a sink SCC.

↳ for finite S, at least.



Def) First Passage Time: $f_{ij}^{(n)} := \mathbb{P}(X_1 \neq j, \dots, X_{n-1} \neq j, X_n = j \mid X_0 = i)$.

Def) Return Probability: $f_{ii} := \sum_{n=1}^{\infty} f_{ii}^{(n)}$ (every $f_{ii}^{(n)}$ are disjoint!)

Obs) Transient $\Rightarrow f_{ii} < 1$. Reccurent $\Rightarrow f_{ii} = 1$. (actual definitions of these)

Def) Irreducibility: MC s.t. $\forall i,j \in S$, $(i \leftrightarrow j)$ (one SCC for entire graph!)

Claim) Suppose $j \in S$ is transient. Then, $\lim_{n\to\infty} p_{ij}^{(n)} = 0 \ \forall i \in S$. (without proof here)

Decompose $S = A \sqcup B$ where A is absorbing ($\in$ reccurent SCC), B is transient.

For $i \in A, j \in B$, what is $h_{ij} = \mathbb{P}(\underbrace{\text{Enter } A \text{ through } j}_{\mathcal{E}_j} \mid X_0 = i)$.

First Step Analysis: $h_{ij} = \sum_{k \in S} \mathbb{P}(\mathcal{E}_j \mid X_1 = k, \cancel{X_0 = i}) \cdot \overbrace{\mathbb{P}(X_1 = k \mid X_0 = i)}^{p_{ik}}$

$= \sum_{k \in S} \mathbb{P}(\mathcal{E}_j \mid X_1 = k) \cdot p_{ik}$. $\mathbb{P}(\mathcal{E}_j \mid X_1 = k) = \begin{cases} \delta_{jk}, & k \in A \\ h_{kj}, & k \in B \end{cases} \Rightarrow \underline{h_{ij} = p_{ij} + \sum_{k \in B} h_{kj} p_{ik}}$.

We can decompose P into $\begin{matrix} \\ A \\ B \end{matrix}\begin{bmatrix} \overset{B}{Q} & \overset{A}{R} \\ 0 & S \end{bmatrix}$. Then, $\underline{H = R + QH} \rightarrow \underline{H = (I-Q)^{-1}R}$.

→ Of course, $(I-Q)$ needs to be invertible, which it always is!

Lemma) Suppose a square matrix M satisfies $\lim_{n \to \infty} M^n = 0$. Then, $(I-M)^{-1}$ exists, and it is given by $\sum_{n=0}^{\infty} M^n$. (proof omitted)

↳ Then, because $\lim_{n \to \infty} Q^n = 0$, $\underline{(I-Q)^{-1}}$ exists. //

→ Fundamental Matrix

Hitting Time: $T_A = |\{n \in \mathbb{N}_0 \mid X_n \in A\}| = \sum_{n=0}^{\infty} \sum_{j \in B} 1\{X_n = j\}$. → $E[T_A \mid X_0 = i]$

$= \sum_{j \in B} E[\overbrace{\sum_{n=0}^{\infty} 1\{X_n = j\}}^{t_{ij}} \mid X_0 = i]$. (exact derivation in notes)

Recall that $\vec{u}_n = \vec{u}_{n-1} P$. If $\pi = \pi P$, π is a <u>stationary distribution</u>.

That is, π is a left eigenvector with $\lambda = 1$ and nonnegative elements.

If MC is finite, $\exists$ a SD. <u>If MC is irreducible, π is unique</u>.

Def) Period of State: For $i \in S$, $d(i) = \gcd(\{n \in \mathbb{N} \mid p_{ii}^{(n)} > 0\})$.

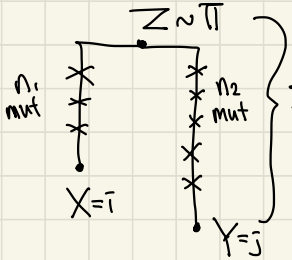
↳ MC is aperiodic if $d(i) = 1$.

Consider a MC that is finite, irreducible, & aperiodic. $\lim_{n\to\infty}[P^n]_{ij} = \pi_j$.

Def) Reversible MC: $\pi_i P_{ij} = \pi_j P_{ji}$ $\forall i,j \in S$. (aka detailed balance)

↳ In fact, $\pi_i[P^n]_{ij} = \pi_j[P^n]_{ji}$ holds by induction.

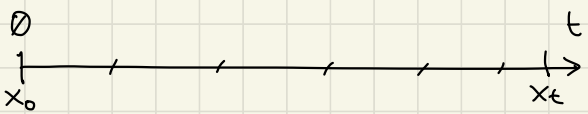
A simple model: Latent root $Z \sim \pi$, n_1 & n_2 mutations for X&Y branches.



$$\mathbb{P}(X=i \wedge Y=j \mid E) = \sum_{k\in S} \mathbb{P}(X=i \wedge Y=j \mid E, Z=k)\,\mathbb{P}(Z=k \mid E)$$
$$= \sum_{k\in S} \mathbb{P}(X=i \mid E, Z=k)\,\mathbb{P}(Y=j \mid E, Z=k)\,\pi_k.$$
$$= \sum_{k\in S} [P^{n_1}]_{ki}[P^{n_2}]_{kj}\pi_k. \quad \text{Assuming reversibility,}$$
$$\to \sum_{k\in S} [P^{n_1}]_{ki}\pi_j[P^{n_2}]_{jk} = \pi_j[P^{n_1+n_2}]_{ji} = \pi_i[P^{n_1+n_2}]_{ij}.$$

Continuous Time Markov Chains

PPP(μ), $\mu > 0$.



$\{X_t \mid t \in \mathbb{R}_{\geq 0}\}$.

Let $N(t) := \#$ of mutations on the edge $\sim Po(\mu t)$.

↳ $\mathbb{P}(N(t)=k) = e^{-\mu t}\cdot\frac{(\mu t)^k}{k!}$ $\forall k = 0,1,2,\cdots$.

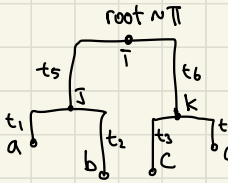
$$\mathbb{P}(X_t=j \mid X_0=i) = \sum_{k=0}^{\infty} \mathbb{P}(X_t=j \mid X_0=i, N(t)=k)\cdot\mathbb{P}(N(t)=k \mid X_0=i)$$
(indep.)
$$= \sum_{k=0}^{\infty}[P^k]_{ij}\cdot e^{-\mu t}\frac{(\mu t)^k}{k!} = e^{-\mu t}\sum_{k=0}^{\infty}\frac{[\mu t P]_{ij}^k}{k!} = e^{-\mu t}\cdot[e^{\mu t P}]_{ij}.$$

Consider $e^{-\mu t I} = \begin{cases} e^{-\mu t}, & i=j \\ 0, & i\neq j \end{cases}$. $\to e^{-\mu t}[e^{\mu t P}]_{ij} = [e^{-\mu t I}e^{\mu t P}]_{ij}$.

generator / rate matrix

* If $[A,B]=0$ ($AB-BA=0$) (commutes), then $e^A e^B = e^{A+B}$. $[e^{\mu t(P-I)}]_{ij}$. //

A Tree Example: full character X, edge weighted phylo. X-tree T, P, μ.



↳ Compute $\mathbb{P}(X|T,P,\mu)$. Define $M(t) = e^{\mu t(P-I)}$.

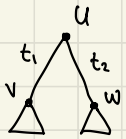
$$\sum_i \sum_j \sum_k \pi_i [M(t_5)]_{ij} [M(t_6)]_{ik} [M(t_1)]_{ja} [M(t_2)]_{jb} [M(t_3)]_{kc} [M(t_4)]_{kd}$$

→ This gives $O(|S|^3)$, but with varying $|X|=n$, $O(|S|^{n-1})$ → intractable!

⇒ With DP, we can achieve $O(n|S|^2)$. Similar to Sankoff's Algorithm!

Consider a subtree rooted at v with children restricted to X.

→ $f_v(i)$ = Prob. of observing children with X given v is state i.



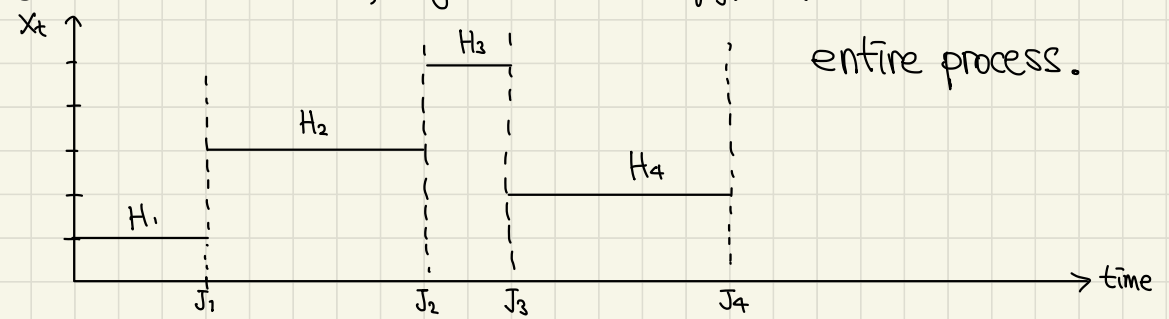
Initialization: For leaf nodes l, $f_l[i] = \mathbb{1}\{i = X_l\}$.

Iteration: $f_u[i] = \left[\sum_{j \in S} [M(t_1)]_{ij} f_v[j]\right] \times \left[\sum_{k \in S} [M(t_2)]_{ik} f_w[k]\right]$ (cond. prob & LTP)

↳ Runtime: $O(|S|^2)$ for every node, $O(n)$ nodes → $O(n|S|^2)$!

Termination: $\mathbb{P}(X|T,P,\mu) = \sum_{k \in S} \pi_k f_{root}[k]$. // (Felsenstein's Algorithm)

General CTMC: Q, a generator. $Q = (q_{ij})\ \forall i,j \in S$ describes the entire process.



$\mathbb{P}(X_{u+t} = j \mid X_u = i) = P_{ij}(t)$. → $P(t) = (P_{ij}(t))$ satisfies $\frac{dP(t)}{dt} = QP(t) = P(t)Q$.

→ $P(t) = e^{tQ}$ is the general ODE solution. $= e^{\mu t(P-I)}$

Properties of Q: $\sum_{j\in S} q_{ij} = 0$. $q_{i} = \sum_{j\neq i} q_{ij}$, $q_{ij} \mid i\neq j > 0$, $q_{ii} = -q_i$.

$H_i := \inf\{u \mid X(t+u)\neq i\} \sim \mathrm{Exp}(q_i)$.

Embedded Jump Chain: $\{Y_n = X_{J_n} \mid n\in\mathbb{N}_0\}$. $\mathbb{P}(Y_{n+1}=j \mid Y_n = i) = \frac{q_{ij}}{q_i}$ $(i\neq j)$ (normalized)

$\mathbb{P}(Y_{n+1}=i \mid Y_n=i) = \begin{cases} 1 & \text{if } q_{ii}=0. \to \text{degenerate case} \\ 0 & \text{if } q_{ii}\neq 0. \end{cases}$ Finally, $H_n \perp\!\!\!\perp Y_n$.

For $0<h\ll 1$, we assume that $p_{ij}(h) = q_{ij}\cdot h + o(h)$, $p_{ii}(h) = 1 - q_i h + o(h)$.

↳ This yields $\frac{dP}{dt} = PQ = QP$ (KFE, KBE).

ex) PPP counter: states are enumerated, incremented at every arrival.

Q looks like $\begin{bmatrix} -\lambda & \lambda & & \\ & -\lambda & \lambda & \\ & & -\lambda & \ddots \\ 0 & & & \ddots \end{bmatrix}$ where diagonals are $-\lambda$ and directly upper entries are λ. $\to P_{00}'(t) = [PQ]_{00} = \sum_{k\in S} P_{0k}Q_{k0} = P_{00}(t)\cdot(-\lambda)$.

Assume initial condition is $P_{00}(0)=1$. $\to P_{00}(t) = e^{-\lambda t}$.

$P_{0j}'(t) = [PQ]_{0j} = \sum_{k\in S} P_{0k}Q_{kj} = \left(P_{0(j-1)}(t) - P_{0j}(t)\right)\lambda \to P_{0j}(t) = \frac{(\lambda t)^j}{j!}e^{-\lambda t}$. $\sim Po(\lambda t)$!

Hidden Markov Models

Motivating Example: Occasionally Dishonest Casino that has fair/loaded dice.

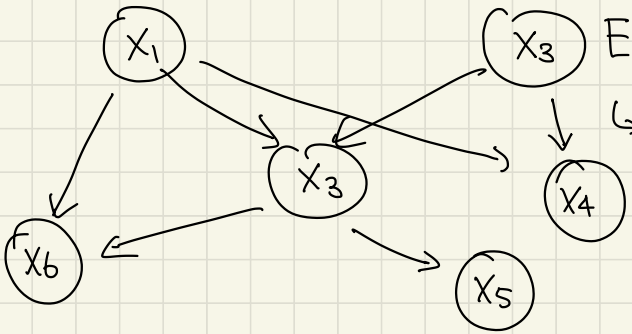
loaded probability α, initial fair dice Prob π, change prob a_{FL}, a_{LF}.

↳ After observing a sequence of rolls, what can we say about parameters?

Concretely, what is $\mathbb{P}(Z_i = \text{fair} \mid \text{Data})$? (If we don't have the model, we can estimate it first)

Applications in Comp Bio: CpG finding, Coalescent HMM → finding time from most recent common ancestor

Graphical Model: A DAG of R.V.s (X_i).

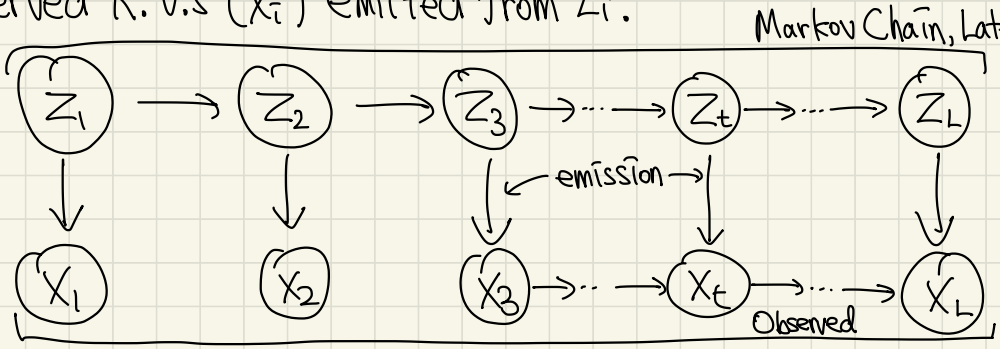


Edges represent conditional dependence.

↳ $\mathbb{P}(X_1 = x_1, \dots, X_n = x_n)$

$= \prod_{i=1}^{n} \mathbb{P}(X_i = x_i \mid \text{Parents of } X_i)$

HMMs are special graphical models governed by a MC (Z_i) (latent) and observed R.V.s (X_i) emitted from Z_i.



↳ This way, X_i will be conditionally independent of all other X_i given Z_i!

Denote the state space of $Z_i \in S$, $X_i \in A$ (some alphabet).

The HMM is specified by tuple (A, S, θ) where $\theta :=$ some parameters.

We don't know ground truth of θ, so we estimate $\theta^* := \underset{\theta}{\operatorname{argmax}}\, \mathbb{P}_\theta(X_{1:L} = x_{1:L})$.

↳ How to calculate $\mathbb{P}_\theta(x_{1:L})$? → enumerating manually, $\sum_{z_1 \dots z_L} \mathbb{P}_\theta(x_{1:L} \mid z_{1:L})\, \mathbb{P}(z_{1:L})$.

↳ $= \prod_{t=1}^{L} \mathbb{P}_\theta(x_t \mid z_t)$ (emission)

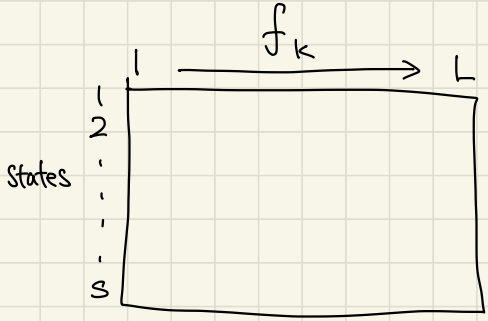
⇒ This sum involves $O(|S|^L)$ terms, intractable!

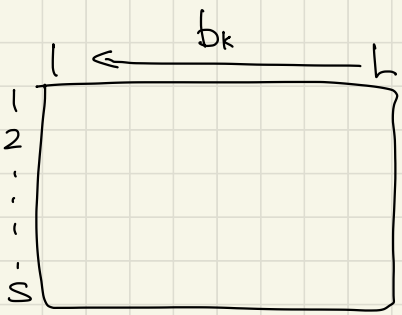
Inference on Z_t: $\mathbb{P}(Z_t = k \mid x_{1:L}) = \mathbb{P}(x_{1:L}, Z_t = k) / \mathbb{P}(x_{1:L})$.

$\mathbb{P}(x_{1:t}, Z_t = k)\,\mathbb{P}(x_{t+1:L} \mid x_{1:t}, Z_t = k) \to \mathbb{P}(x_{t+1:L} \mid Z_t = k)$ by cond. indep.

$=: f_k(t)$, forward probability; $=: b_k(t)$, backward probability

↳ given $f_k(t)$ & $b_k(t)$, $\mathbb{P}(Z_t = k \mid x_{1:L}) = \dfrac{f_k(t) \cdot b_k(t)}{\mathbb{P}(x_{1:L})}$.





Observe that $\mathbb{P}(x_{1:L}) = \sum_k \mathbb{P}(x_{1:L}, Z_t = k)$ by marginalization.

Then, these values are exactly $\sum_k f_k(L)$! Using backward probs,

$\mathbb{P}(x_{1:L}) = \sum_k \mathbb{P}(x_{1:L} \mid Z_1 = k)\,\mathbb{P}(Z_1 = k)$ (π_k) $\mathbb{P}(x_{2:L} \mid Z_1 = k)\,\mathbb{P}(x_1 \mid Z_1 = k)$

$= \sum_k b_k(1) \cdot e_k(x_1) \cdot \pi_k$. ($b_k(1)$; $e_k(x_1)$; $(x_{2:L} \perp\!\!\!\perp x_1 \mid Z_1)$)

How to fill in $f_k(t)$ & $b_k(t)$? → Use DP! For the forward table:

Base Case: $f_k(1) = \mathbb{P}(x_1, Z_1 = k) = \mathbb{P}(x_1 \mid Z_1 = k)\,\mathbb{P}(Z_1 = k) = e_k(x_1) \cdot \pi_k$.

General Case: $f_\ell(t) = \mathbb{P}(x_{1:t}, Z_t = \ell) = \sum_k \mathbb{P}(x_{1:t}, Z_{t-1} = k, Z_t = \ell)$ (cond. indep.)

$\mathbb{P}(x_{1:t-1}, Z_{t-1} = k)\,\mathbb{P}(x_t, Z_t = \ell \mid x_{1:t-1}, Z_{t-1} = k) \to \mathbb{P}(x_t, Z_t = \ell \mid Z_{t-1} = k)$

$f_k(t-1)$

$\mathbb{P}(x_t \mid Z_{t-1} = k, Z_t = \ell)\,\mathbb{P}(Z_t = \ell \mid Z_{t-1} = k)$

$e_\ell(x_t)$; $a_{k\ell}$ (given by the MC)

$\Rightarrow f_\ell(t) = e_\ell(x_t) \sum_{k \in S} f_k(t-1)\, a_{k\ell}$. //

Upshot: in the parameter θ, we need $\{a_{kl} \mid k,l \in S\}$, $\{e_k(\sigma) \mid k \in S, \sigma \in A\}$, and $\{\pi_k \mid k \in S\}$ to specify the entire table of $f_k(t)$.

Backward Table: $b_k(t) = \mathbb{P}(x_{t+1:L} \mid Z_t = k)$.

Base Case: $b_k(L) = 1 \ \forall k \in S$.

General Case: $b_k(t) = \mathbb{P}(x_{t+1:L} \mid Z_t = k) = \sum_{l \in S} \mathbb{P}(x_{t+1:L} \mid Z_t = k, Z_{t+1} = l) \cdot \underbrace{\mathbb{P}(Z_{t+1} = l \mid Z_t = k)}_{a_{kl}}$

Observe that $(x_{t+1:L} \perp\!\!\!\perp Z_t \mid Z_{t+1}) \rightarrow \mathbb{P}(x_{t+1:L} \mid Z_{t+1} = l)$. This is then

$\mathbb{P}(x_{t+1} \mid Z_{t+1} = l)\mathbb{P}(x_{t+2:L} \mid Z_{t+1} = l) = e_l(x_{t+1}) \cdot b_l(t+1)$.

$\Rightarrow \underline{b_k(t) = \sum_{l \in S} e_l(x_{t+1})\, b_l(t+1)\, a_{kl}}$. //

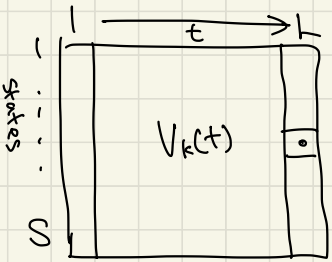
Posterior Decoding: for $t = 1 \dots L$, let $Z_t^* := \operatorname{argmax}_{k \in S} \overbrace{\mathbb{P}(Z_t = k \mid x_{1:L})}^{\frac{f_k(t) b_k(t)}{\mathbb{P}(x_{1:L})}}$ //

↳ This has a pathological issue that $Z_1^* \dots Z_L^*$ may be an invalid sequence!

Viterbi Decoding: goal is to find $Z_{1:L}^{**} = \operatorname{argmax}_{z_1 \dots z_L} \mathbb{P}(z_{1:L} \mid x_{1:L})$.

Observe that $\mathbb{P}(z_{1:L} \mid x_{1:L}) = \mathbb{P}(z_{1:L}, x_{1:L}) / \mathbb{P}(x_{1:L})$, but the denom. never changes! $\rightarrow Z_{1:L}^{**} = \operatorname{argmax}_{z_1 \dots z_L} \mathbb{P}(z_{1:L}, x_{1:L})$ is equivalent.

$\rightarrow$ Let $V_k(t) := \max_{z_1 \dots z_{t-1}} \mathbb{P}(z_{1:t-1}, Z_t = k, x_{1:t})$. We fill in the DP table:



Base Case: $V_k(1) = \mathbb{P}(Z_1 = k, X_1) = \mathbb{P}(X_1 \mid Z_1 = k)\mathbb{P}(Z_1 = k)$

$= \underline{e_k(x_1) \cdot \pi_k}$.

General Case: $V_l(t) = \max_{z_{1:t-2}} \max_{k \in S} \underline{\mathbb{P}(z_{1:t-2}, Z_{t-1} = k, Z_t = l, x_{1:t})}$

We can rewrite the inner prob. as: $\mathbb{P}(z_{1:t-2}, z_{t-1}=k, x_{1:t-1})\mathbb{P}(z_t=l, x_t \mid z_{1:t-2}, z_{t-1}=k, x_{1:t-1})$. By cond. ind., this is $\underbrace{\mathbb{P}(z_{1:t-2}, z_{t-1}=k, x_{1:t-1})}_{\text{"almost" } V_k(t-1)}\mathbb{P}(z_t=l, x_t \mid z_{t-1}=k)$.

$\mathbb{P}(x_t \mid z_t=l, z_{t-1}=k)\mathbb{P}(z_t=l \mid z_{t-1}=k) = \underbrace{e_l(x_t)\cdot a_{kl}}$.

$\rightarrow \underbrace{V_l(t) = e_l(x_t)\max_{k\in S} a_{kl}\cdot V_k(t-1)}$. Keep pointers of argmax to back trace!

At column L, $z_L^{**} = \arg\max_{k\in S} V_k(L)$, then back trace to find optimal path.

Parameter Estimation

Want to find $\theta^* = \arg\max_\theta \mathbb{P}_\theta(x_{1:L})$, but this is not necessarily convex!

Case 1: Latent Variables are known, $(z_1, \dots, z_L)\in S^L$.

Let A_{kl} := # of transitions from state k to l in the sequence $z_{1:L}$.

Let $E_k(\sigma)$:= # of emissions of $\sigma\in \underbrace{A}_{\text{alphabet}}$ from state $k\in S$.

$\mathbb{P}_\theta(x_{1:L}, z_{1:L}) = \underbrace{\mathbb{P}_\theta(x_{1:L}\mid z_{1:L})}\underbrace{\mathbb{P}_\theta(z_{1:L})} = \underbrace{\left[\prod_{k\in S}\prod_{\sigma\in A}[e_k(\sigma)]^{E_k(\sigma)}\right]}\cdot\underbrace{\left[\prod_{k\in S}[\pi_k]^{\mathbb{1}[z_1=k]}\right]}\cdot \underbrace{\left[\prod_{i,j\in S}(a_{ij})^{A_{ij}}\right]}$. Since this is monotone, we can take a log in the argmax.

(In HW) MLE given $z_{1:L}$ are: $\hat{a}_{ij} = \frac{A_{ij}}{\sum_k A_{ik}}$, $\hat{e}_k(\sigma) = \frac{E_k(\sigma)}{\sum_{\sigma'} E_k(\sigma')}$.

Case 2: Expectation Maximization. Observed RV X, Latent RV Z.

$\log p(x) = \text{ELBO} + KL(\underbrace{q(z\mid x)}_{\text{arbitrary}} \,\|\, p(z\mid x))$. $\leftarrow$ not symmetric on q & p!!

For discrete RV X, μ, ν := prob. measures on the same prob. space.

Def) KL Divergence: $KL(\mu|\nu) := \sum_x \mu(x) \log\left(\frac{\mu(x)}{\nu(x)}\right)$. (assume $\nu(x)=0 \Rightarrow \mu(x)=0$)

↳ $KL(\mu|\nu) \geq 0$, and is $=0$ iff $\mu(x)=\nu(x)\ \forall x$ (same dist.)

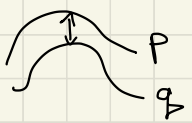
→ Since $KL \geq 0$, $\log p(x) \geq ELBO$, and we can optimize this value.

Def) Evidence Lower Bound: $ELBO = \underset{z\sim q(\cdot|x)}{E}[\log(p(x|z))] - KL(q(z|x) \| p(z))$.

$= \underset{z\sim q(\cdot|x)}{E}[\log(p(x,z)) - \log(q(x,z))] + \log(q(x))$.

$\Rightarrow \log p(x) \geq \underset{z\sim q(\cdot|x)}{E}[\log p(x,z) - \log(q(x,z))] + \log q(x)$

$\rightarrow \log p(x) - \log q(x) \geq \underset{z\sim q(\cdot|x)}{E}[\log p(x,z) - \log(q(x,z))]$.



Back to HMM, defined as: $Q(\theta|\theta_m) - Q(\theta_m|\theta_m) \geq 0$. We maximize θ', given by $\theta' = \underset{\theta}{\text{argmax}}\, Q(\theta|\theta_m)$.

Idea: Devise an iterative algorithm that finds $\theta_0, \theta_1, \ldots, \theta_m$ where $P_{\theta_i}(X_{1:L}) \leq P_{\theta_{i+1}}(X_{1:L})\ \forall i$, i.e. monotonicity on likelihood. → EM!

Analysis of ELBO bound: Let $q := P_{\theta_m}$, the current estimate, and $p := P_\theta$, some arbitrary measure. The equation becomes: ($x := (x_1, \ldots, x_L)$)

$\log P_\theta(x) - \log P_{\theta_m}(x) \geq Q(\theta|\theta_m) - Q(\theta|\theta_m)$ where

(constant!) (constant!)

$\underset{z\sim P_\theta(\cdot|x)}{E}[\log(P_\theta(x,z))] = E[\log(P_\theta(x,z))|x,\theta_m] =: Q(\theta|\theta_m)$.

Now, if the RHS ≥ 0, then $\log P_\theta(x) \geq \log P_{\theta_m}(x)$, so $P_\theta(x) \geq P_{\theta_m}(x)$!

Then, our goal is to find $\theta_{m+1} = \arg\max_\theta Q(\theta|\theta_m)$, which satisfies the condition $P_{\theta_{m+1}}(x) \geq P_{\theta_m}(x)$.

$\theta_{m+1} = \arg\max_\theta E[\log P_\theta(x,z)|x,\theta_m]$. (Baum-Welch)

↳ Upshot: In HMM, this has a closed form, so we can do argmax analytically!

Recall that when $Z(:=(z_1,\dots,z_L))$ is known, $\hat{a}_{ij} = \frac{A_{ij}}{\sum_k A_{ik}}$, $\hat{e}_k(\sigma) = \frac{E_k(\sigma)}{\sum_{\sigma'} E_k(\sigma')}$

from maximizing $\log(P_\theta(x,z)) = \prod_l [\pi_l]^{1\{z_1=l\}} \cdot \prod_{i,j} (a_{ij})^{A_{ij}} \cdot \prod_{\sigma,k} [e_k(\sigma)]^{E_k(\sigma)}$.

If Z is unknown, the change is that now $z_1, A_{ij}, E_k(\sigma)$ depends on Z!!

↳ Taking expectation of log, we have a nice disjoint expression by variables:

$$E\Big[\sum_l 1\{z_1=l\}\log(\pi_l) + \sum_{i,j} A_{ij}(z)\log(a_{ij}) + \sum_\sigma \sum_k E_k(\sigma,z)\log(e_k(\sigma))\Big].$$

$E[\sum_{i,j} A_{ij}(z)\log(a_{ij})] = \sum_{i,j}\log(a_{ij})E[A_{ij}(z)|x,\theta_m]$. Solving this:

$\hat{a}_{ij}^{(m+1)} = \frac{E[A_{ij}(z)|x,\theta_m]}{\sum_k E[A_{ik}(z)|x,\theta_m]}$. $A_{ij}(z)$ is represented as $\sum_{t=1}^{L-1} 1\{z_t=i \wedge z_{t+1}=j\}$.

Then, $E[A_{ij}(z)|x,\theta_m] = \sum_{t=1}^{L-1} E[1\{z_t=i\wedge z_{t+1}=j\}|x,\theta_m] = \sum_{t=1}^{L-1} P(z_t=i, z_{t+1}=j|x,\theta_m)$

$= \sum_{t=1}^{L-1} f_i^{(m)}(t)\cdot a_{ij}^{(m)}\cdot e_j^{(m)}(x_{t+1})\cdot b_j^{(m)}(t+1) \Big/ P_{\theta_m}(x)$. (derivation in discussion)

Pseudoalignment [Guest Lecture by Sina Booeshaghi]

De Bruijn Graphs: Construction & Properties

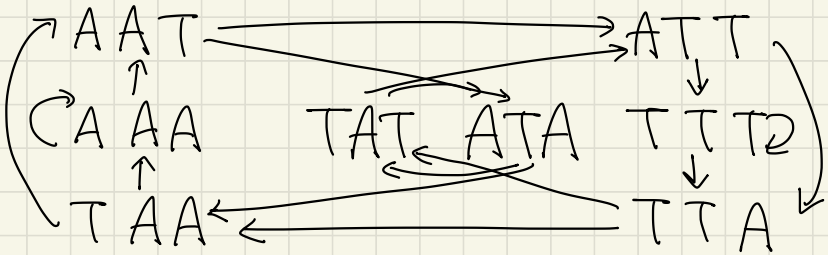
↳ Directed Graph, Alphabet $\Sigma = \{A, C, G, T\}$, integer $K > 0$.

Each node represents a char sequence of length K.

An edge is drawn between nodes sharing (K-1)-length overlapping.

i.e, $a \to b$ implies $a = X\underline{\ S\ }$ & $b = \underline{\ S\ }Y$

ex) $\Sigma = \{A, T\}$, $K=2$, then $\underline{AA}A \to A\underline{AT}$. full graph:

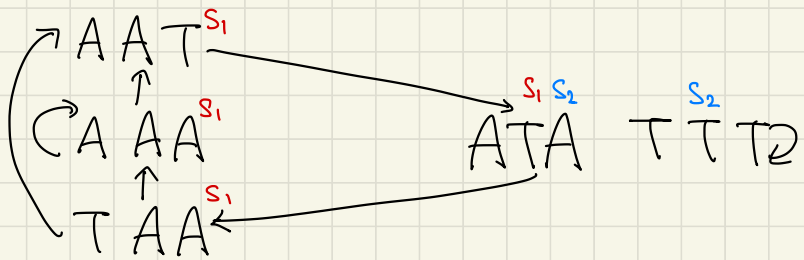


Some Properties of $U = DBG(\Sigma, k)$: ($|\Sigma| \cdot n$)

1) # of nodes $= |\Sigma|^K$. 2) Indegree = Outdegree $= |\Sigma|$, # of edges $= |\Sigma|^{k+1}$.

Coloring DBG: Let $S_1 := \{AAT, AAA, TAA, ATA\}$, $S_2 := \{TTT, ATA\}$.

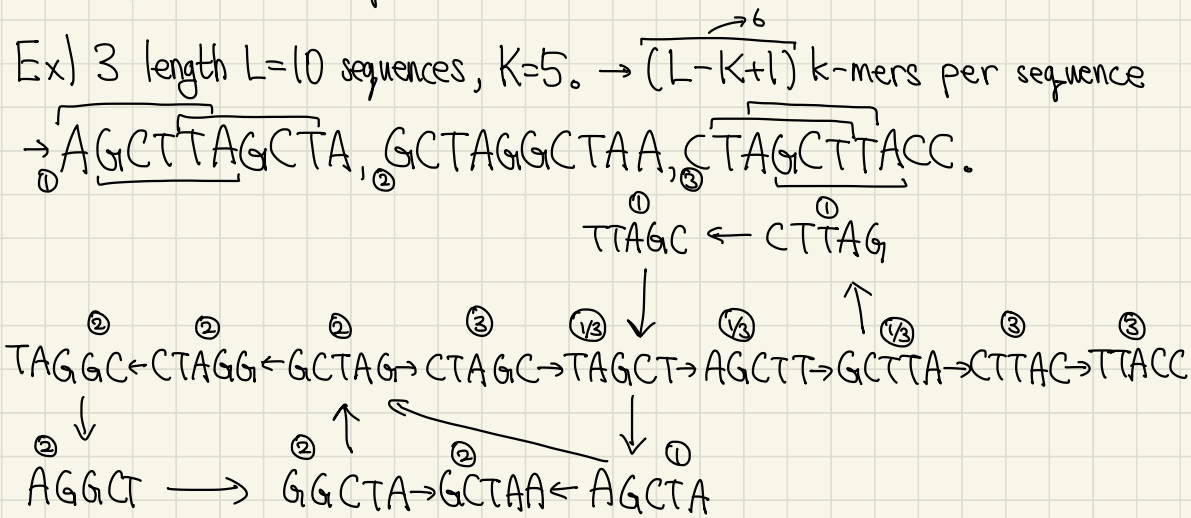
$S_1, S_2 \subset U$, $S_1 \cap S_2 = \{ATA\}$. Let $W := S_1 \cup S_2$. Coloring is membership:



Construction of DBG: Given a set of sequences, extract the overlapping k-mers. e.g. AAGAT, K=4 → {AAGA, AGAT}. Then, identify unique

k-mers & annotate them with its sequence origin. Create directed edges that link (K-1) overlaps.

Ex) 3 length L=10 sequences, K=5. → (L-K+1) k-mers per sequence



New sequences: CTAGGCT. Could this have originated from one of ①,②, or ③?

↳ Break it up into k-mers: CTAGG, TAGGC, AGGCT. → ②! unique mapping

AGCTTAG. This does not have a match → unmapped

TAGCTTA. ① & ③ are both feasible matches → ambiguous mapping

AGCTTAC. ⅓, ⅓, ①, intersection is ① → resolved to unique mapping

Memory: $O(|\Sigma|^K)$, Time: $O(L)$ construction, $O(m)$ query

RNA-Sequencing (Pseudo) Alignment: Experiment quantifying RNAs in cell pop. We want genes that the RNAs transcribed from. From references of the list of gene sequences, we can construct a colored DBG. (Kallisto!)

↳ By aligning with DBG, we get a vector of counts of each gene.

To resolve ambiguities, use EM to estimate α_T, abundance of gene T.

Define a binary matrix $(y)_{r,t} := \mathbb{1}\{\text{read } r \text{ could be gene } t\}$.

RNA Seq

Simplified Model: Let $T := \{t_1, \ldots, t_k\}$ be the set of transcripts.

Let $R := \{r_1, \ldots, r_N\}$ be the set of reads from a given sample/cell.

Assume: single-ended reads, error free reads, all transcripts are known, and reads originated from the reference transcriptome.

Goal: Find the transcript from which read originated.

↳ not always unambiguously determined!

Rephrased Goal: Find the feasible set of transcripts

Solution 1: align reads against transcriptomes using algorithms & heuristics.

↳ ex) STAR alignment using suffix trees & arrays

Solution 2: Pseudoalignment; only care about whether it aligns, not where it aligns. → Compatibility Matrix $Y = (y_{i,k})$ with DBG!

↳ one issue: some ambiguous sequences are actually not ambiguous...

DBG does not require consecutive k-mers to appear consecutive in order!

→ however, for sufficiently large k, this is unlikely.

For matrix Y, define an equivalence class for each $e_i = \{t \in T \mid y_{f,t} = 1\}$; basically, if the rows look the same, they are functionally the same.

Assuming that a read is generated from a transcript $t \in T$ w.p $\propto \frac{\text{abundance of } t}{\text{length of } t}$ (α_t), and is a small uniformly sampled segment from t.

Unambiguous Model α_t: $\hat{\alpha}_t = \frac{X_t}{N}$ where $X_t :=$ # of times t is observed.

↳ When reads are ambiguous, split it into each X_t relevant. But this is not optimal. → Use EM! Observed variable is matrix Y that does not commit to a single t, and latent is matrix Z that has a "committed mapping" s.t. each row has only one possible t.

↳ Intuitively, if we have a bunch of one t in multiple reads, we should weight it more when splitting into X_m.

Ambiguous Model α_t: Let $z_{ik}^{(t+1)} := E[Z_{ik}^{(t)} \mid Y_i, \theta^{(t)}] = Pr[Z_{ik} = 1 \mid Y_i, \theta^{(t)}] = \frac{\alpha_k^{(t)} y_{ik}}{\sum_k \alpha_k^{(t)} y_{ik}}$ ($\theta^{(t)} = \{\alpha_k^{(t)}\}_{k=1}^K$). E step (revising rows)

Then let $n_k^{(t+1)} := \sum_{i=1}^{N} z_{ik}^{(t+1)}$. $\alpha_k^{(t+1)} := \frac{n_k^{(t+1)}}{N}$. ← M step (aggregating columns)

Hypothesis Testing

Paremeterized HT: data $X_1, \dots, X_n \overset{iid}{\sim} f_\theta, \theta \in \Theta$. Two candidates Θ_0, Θ_1. H_0 is when $\theta \in \Theta_0$ (null), H_1 is when $\theta \in \Theta_1$ (alternative).

ex) coin flips $X_1, \dots, X_n \sim Ber(p)$. $H_0: p = 0.5$. $H_1: p > 0.5$. (head prob.)

a natural statistic is $T(X) = \sum_{i=1}^{n} X_i$. → how do we interpret this value?

Def) Power Function: $\pi(\theta|\delta) := Pr[T(X) \in R | \delta]$ (test, rejection region)

Neyman-Pearson Framework: $(\pi(\delta) = 1 - \beta(\delta))$

reality / decision	H_0 is true	H_1 is true
decide H_0	good	Type II Error (β)
decide H_1	Type I Error (α)	good

We will consider H_0 as status quo and only reject when the evidence is strong enough to change my world view!

Define $\alpha = Pr[T(X) \in R | \theta \sim \Theta_0]$, $\beta = Pr[T(X) \notin R | \theta \sim \Theta_1]$.

We want to keep $\alpha < c$, say $c = 0.05$, while minimizing β.

Lemma) Let δ^* be a test that H_0 is not rejected if: $af_0(X) > bf_1(X)$ & rejects if $<$. Then for every other test δ, $a\alpha(\delta^*) + b\beta(\delta^*) \leq a\alpha(\delta) + b\beta(\delta)$. (for simple tests)

Proof: Let S_1 be the rejection region of some δ. $S_0 := \overline{S_1}$.

$a\alpha(\delta) + b\beta(\delta) = a\sum_{x\in S_1} f_0(x) + b\sum_{x\in S_0} f_1(x) = a\sum_{x\in S_1} f_0(x) + b\left[1 - \sum_{x\in S_1} f_1(x)\right]$

$= b + \sum_{x\in S_1}\left[a f_0(x) - b f_1(x)\right]$. Then δ^* minimizes such value. //

→ the ratio $\frac{f_1(x)}{f_0(x)} > C$ determines rejection (likelihood HT).

ex) $X_1, \dots, X_n \overset{iid}{\sim} N(\mu, \sigma^2)$. $H_0: \mu = \mu_0$, $H_1: \mu = \mu_1 > \mu_0$.

$\frac{f_1(X)}{f_0(X)} = \exp\left(\overline{\sigma^2}(\mu_0 - \mu_1)\sum\left[X_i - \frac{n}{2\sigma^2}(\mu_1^2 - \mu_0^2)\right]\right)$. We want

$P_{H_0}(T(X) > C) = \alpha \rightarrow P_{H_0}(\overline{X_n} > C) = \alpha$. $\rightarrow$ $C = \mu_0 + Z_{1-\alpha}\frac{\sigma}{\sqrt{n}}$.

↳ But what if we want to test $H_0: \mu < \mu_0$, $H_1: \mu \geq \mu_0$? (composite)

Def) Uniformly Most Powerful (UMP) Test: δ^* at level α_0 where $\forall \delta$ s.t. $\alpha(\delta) \leq \alpha_0$, $\forall \theta \in \Theta_1$, $\Pi(\theta|\delta) \leq \Pi(\theta|\delta^*)$.

ex) $X_1, \dots, X_n \overset{iid}{\sim} Ber(p)$. Let $Y = \sum_{i=1}^n X_i$. $f_p(x) = p^Y(1-p)^{n-Y}$. $H_0: p = p_0$, $H_1: p = p_1$. $\frac{f_1(X)}{f_0(X)} = \left[\frac{p_1(1-p_0)}{p_0(1-p_1)}\right]^Y\left[\frac{(1-p_1)}{(1-p_0)}\right]^n$. Observe that $\frac{f_1(x)}{f_0(x)}$ 1) only depends on X through $Y = T(X)$, 2) is monotone on Y → These conditions ensure an UMP for $\begin{cases} H_0: p \leq p_0 \\ H_1: p > p_0 \end{cases}$ and $\begin{cases} H_0: p \geq p_0 \\ H_1: p < p_0 \end{cases}$. For example, if $p_0 = 0.5$, $n = 20$, we can set $Y \geq 18$ for rejection to get some α_0.

However, we do not have UMP for $\begin{cases} H_0: p = p_0 \\ H_1: p \neq p_0 \end{cases}$.

⇒ How do we explicitly calculate the α_0 for each Y?

ex) $X_1,\dots,X_n \overset{iid}{\sim} N(\mu,\sigma^2)$ where σ^2 is known, μ is unknown. Recall that $f(x)=\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \to f_p(\vec{x})=\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2\right)$

$\to \frac{f_1(x)}{f_0(x)}=\exp\left(\frac{n(\mu_1-\mu_0)}{\sigma^2}\left(\bar{X}_n-\frac{1}{2}(\mu_1+\mu_0)\right)\right)$. reject when $\bar{X}_n > C$ is a UMP for $H_0:\mu\le\mu_0$, $H_1:\mu>\mu_0$. But what is C? It depends on α_0.

$\Rightarrow$ Fix α_0. We want $Pr[\bar{X}_n > C \mid H_0]=\alpha_0$, and equivalently, $Pr[\bar{X}_n \le C \mid H_0]=1-\alpha_0 \to Pr\left[\frac{\bar{X}_n-\mu_0}{\sigma/\sqrt{n}} \le \frac{C-\mu_0}{\sigma/\sqrt{n}}\right]=1-\alpha_0$. Then, $\frac{\bar{X}_n-\mu_0}{\sigma/\sqrt{n}}$ is a RV $\sim N(0,1)$. Thus $\frac{C-\mu_0}{\sigma/\sqrt{n}}=\Phi^{-1}(1-\alpha_0) \Rightarrow \underline{C=\mu_0+\frac{\sigma}{\sqrt{n}}\Phi^{-1}(1-\alpha_0)}$.

(* $\Phi^{-1}(k)$ is the inverse Gaussian cdf.)

Def) P-Value: For a sample X, the lowest α for which we will reject the null for test $\delta(\alpha)$. Equivalently, it is also the probability of receiving a statistic "more extreme" than $T(X)$, $Pr[T \ge T(X) \mid H_0]$.

ex) For coin flipping, say $H_0: p=1/2$, $H_1: p>1/2$. We get 18/20 heads. Then p-value is $Pr[T=18]+Pr[T=19]+Pr[T=20]$ for H_0.

ex) For mean-finding, if $\bar{X}_n = m$, p-value $= Pr[\bar{X}_n \ge m \mid \mu=\mu_0]$

$= 1-Pr\left[\frac{\bar{X}_n-\mu_0}{\sigma/\sqrt{n}} \le \frac{m-\mu_0}{\sigma/\sqrt{n}}\right]=1-\Phi^{-1}\left(\frac{m-\mu_0}{\sigma/\sqrt{n}}\right)$.

(* In real experiments, we only report p-values instead of αs.)

$\Rightarrow$ Can we find a way to test $H_0:\mu=\mu_0$, $H_1:\mu\neq\mu_0$? (two-sided)

We want to find <u>two</u> thresholds c_1, c_2 s.t. $\Pr[\bar{X}_n \le c_1 \mid H_0] + \Pr[\bar{X}_n \ge c_2 \mid H_0] = \alpha$. We <u>could</u> vary c_1, c_2, but an intuitive choice would be to set each of the probabilities to $\frac{1}{2}\alpha$. Also, <u>σ is unknown</u> now.

$\Rightarrow$ We can estimate $S \simeq \sigma$ via $S^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X}_n)^2$. Then, we have $T = \frac{\bar{X}_n - \mu_0}{S/\sqrt{n}} \overset{H_0}{\sim} t_{n-1}$, the "Student t" distribution.

(* S^2 is defined with $\frac{1}{n-1}$ for unbiased estimating!)

Inferences on samples: rows are genes, columns are samples. Samples are split into control/treatment groups. Consider gene 1. Let $X_1, \dots, X_n \overset{iid}{\sim} N(\mu_1, \sigma^2)$ be cells before treatment, and $Y_1, \dots Y_n \overset{iid}{\sim} N(\mu_2, \sigma^2)$ be cells after treatment. This is a <u>paired sample.</u> Then, we can take $D_i := X_i - Y_i$, and under $H_0: \mu_1 = \mu_2$, $D_1, \dots, D_n \overset{iid}{\sim} N(0, \sigma_D^2)$. $\rightarrow \bar{D}_n, S_D^2$ gives $T = \frac{\bar{D}_n}{S_D/\sqrt{n}} \overset{H_0}{\sim} t_{n-1}$.

2 sample case: In this case, <u>no samples are paired.</u> $X_1, \dots, X_n \overset{iid}{\sim} N(\mu_1, \sigma_1^2)$ and $Y_1, \dots, Y_m \overset{iid}{\sim} N(\mu_2, \sigma_2^2)$. We define a new statistic S_p, the pooled variance $S_p^2 := \frac{(n-1)S_x^2 + (m-1)S_y^2}{n+m-2}$. Then, $T = \frac{\bar{X}_n - \bar{Y}_m}{S_p\sqrt{1/n + 1/m}} \overset{H_0}{\sim} t_{n+m-2}$.

$\rightarrow$ Can we test multiple rows (genes)? There is a glaring problem.

Type 1 error is $\Pr[\text{reject } H_0 \mid H_0 \text{ is true}] \leq \alpha$, say 5%. If we have $n=10$, $\Pr[\text{never makes T1 error}] \leq 1-(1-0.05)^{10}$, which is bad.

Def) Family-Wide Error Rate (FWER): $\Pr[\text{at least one T1 error}]$.

→ Given multiple null hypotheses $H_1,\dots,H_m$, we compute p-values $p_1,\dots,p_m$. We want the FWER over $p_1,\dots,p_m \leq \alpha$.

Bonferroni Method: test $p_i \leq \frac{1}{m}\alpha$.

Proof: $\text{FWER} = \Pr[\bigcup_{i\in I_0} p_i \leq \frac{\alpha}{m}]$ where $I_0 :=$ set of true nulls. By union bound, $\leq \sum_{i\in I_0} \underbrace{\Pr[p_i \leq \frac{\alpha}{m}]}_{\frac{\alpha}{m}} = m_0 \cdot \frac{\alpha}{m}$ where $m_0 :=$ # of true nulls, $|I_0|$. $m_0 \leq m$, so $\text{FWER} \leq \alpha$. // ($p_i \overset{H_i}{\sim} \text{Uni}(0,1)$)

↳ This is nice, but we lose in statistical power (T2 error blows up).

Bonferroni-Holm Method: $H_1,\dots,H_m \to p_1,\dots,p_m$. Sort in ascending p-values, $H_{(1)},\dots,H_{(m)} \to p_{(1)} \leq \cdots \leq p_{(m)}$. If $p_{(j)} \leq \frac{\alpha}{m-j+1}$ for $j = 1,\dots,i$, reject $p_{(i)}$. Then, $\text{FWER} \leq \alpha$ while having more statistical power!

↳ However, even this does not give enough power.

Def) False Discovery Rate:

reality \ decision	accept H_0	reject H_0	total
H_0 true	U	V	m_0
H_0 false	T	S	$m-m_0$
total	$m-R$	R	m

Observe that

FWER = $Pr[V \geq 1]$. Let the False Discovery Proportion FDP := $\frac{V}{\max(R,1)}$.

FDR := $E[FDP]$. We want to control this value.

Benjamini-Hochberg Procedure: Sort $p_{(1)} \leq \dots \leq p_{(m)}$. Let i_0 be the largest i s.t. $p_{(i)} \leq \frac{i}{m}\alpha$. Reject $H_{(1)}, \dots, H_{(i_0)}$.

Theorem) If $p_1, \dots, p_m$ are independent (can be relaxed a bit), then the B-H procedure guarantees FDR = $\frac{m_0}{m}\alpha \leq \alpha$.