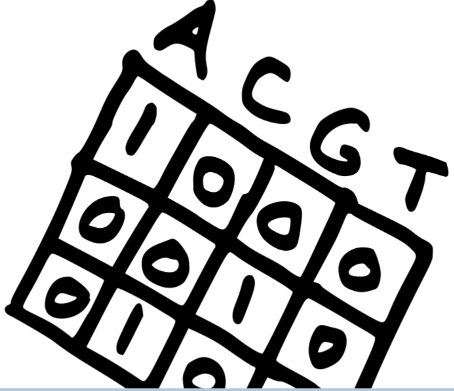
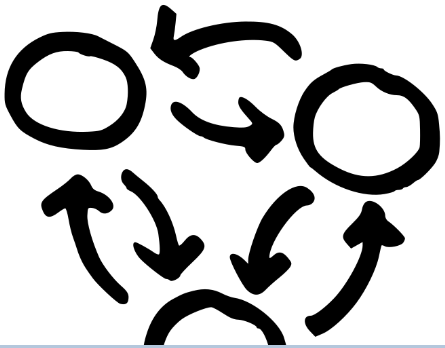
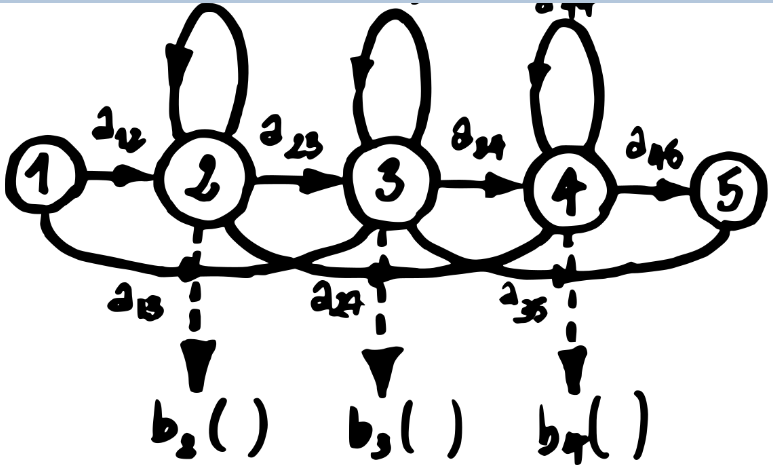
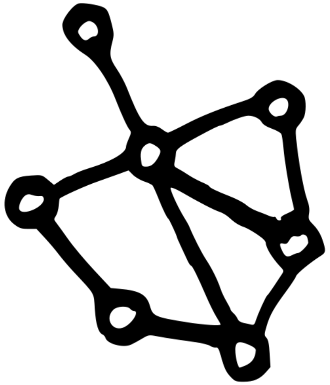




Sequence algorithms

DAG workshop, 2020
Cyril Matthey-Doret





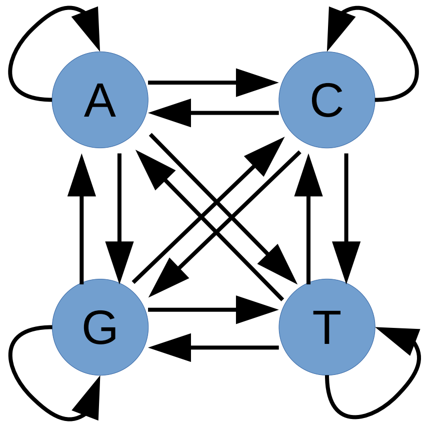
cyril.mattheydoret@gmail.com
cmdoret

Exercises from session 3

Working with sequences of states

- Different from classification: Input and output have an order
- Markov models: define states and transition probabilities
- Example: nucleotide transition matrix from DNA sequence

Input:
AACTTTGAGAC

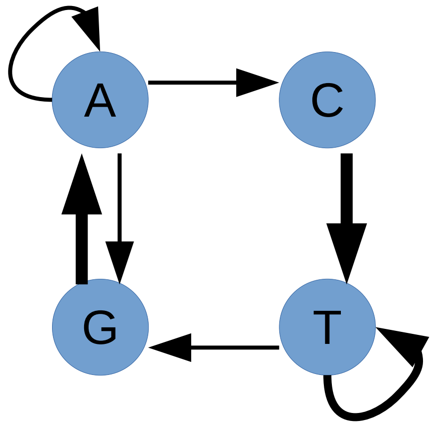


↗	A	C	G	T
A	1	1	1	0
C	0	0	0	1
G	2	0	0	0
T	0	0	1	2

Working with sequences of states

- Different from classification: Input and output have an order
- Markov models: define states and transition probabilities
- Example: nucleotide transition matrix from DNA sequence

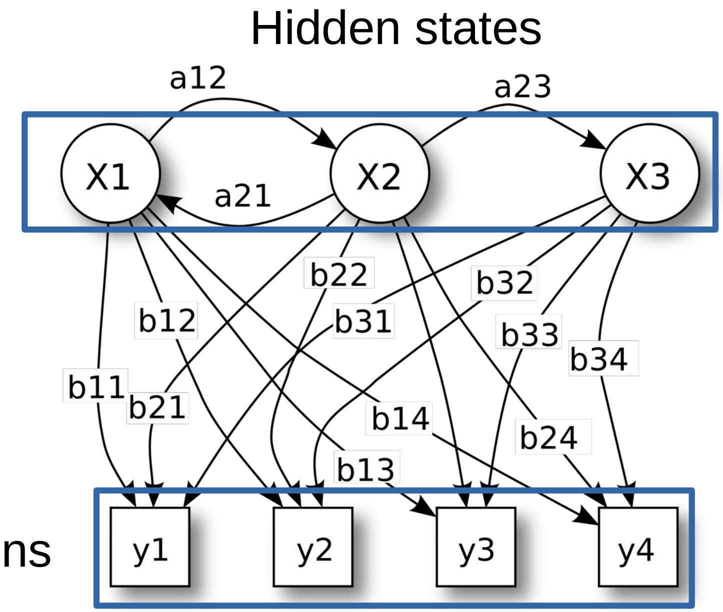
Input:
AACTTTGAGAC



↗	A	C	G	T
A	0.33	0.33	0.33	0
C	0	0	0	1
G	1	0	0	0
T	0	0	0.33	0.66

Predicting states from sequence

- Markov models by themselves have limited use in genomics
- Hidden Markov models (HMM) are more useful in genomics
    - Input: observations → DNA sequence
    - Output: states → gene / not gene



a = transition probabilities
b = emission probabilities

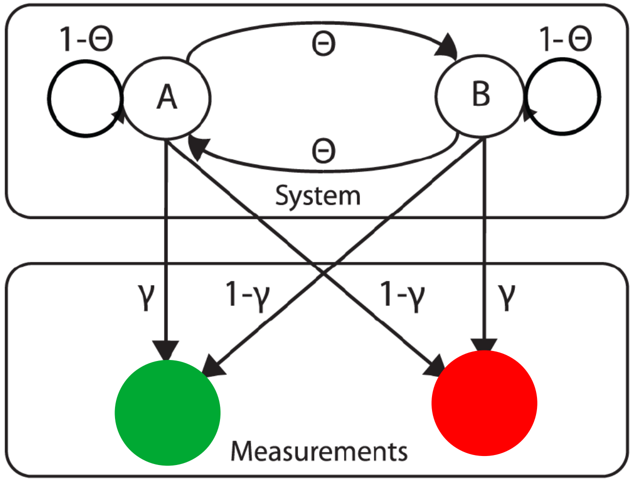
Observations

HMM: example

- A system switches between two (hidden) states A and B
- A device measures the system state, but it is error prone
- It shows 🟢 or 🔴 when it measures A or B, respectively

What we want to know →

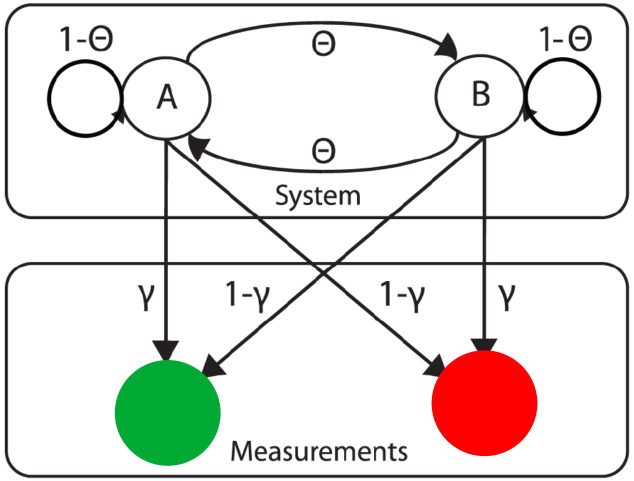
What we see →



Example from: Sunnåker et al., PloS Comp.Biol. 2013

HMM: example

- Given the measurement sequence: 🟢 🔴 🔴
- What is the probability of the observed measurement sequence ?
  - Naive: Sum probabilities of all possible state sequences



HMM: Naive sequence probability

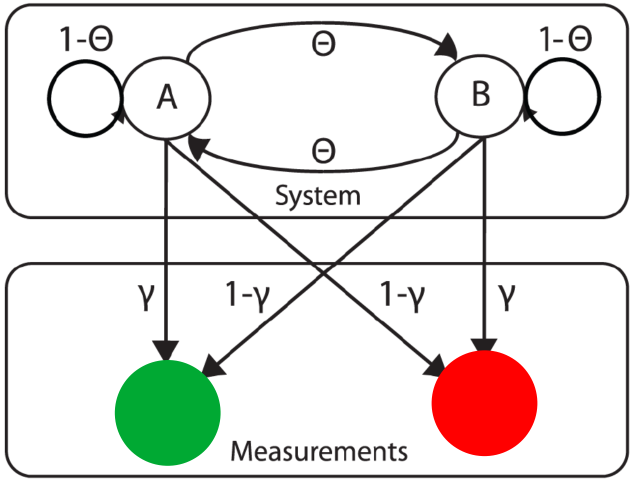
- Find the probability of measurements: 🟢 🔴 🔴
- Weighted sum probabilities of every single state sequence

With X different hidden states, the number of possible sequences **S** of length T is: **R=X^T (here: 8=2^3)**

State sequence of length T

$$p(M_T) = \sum_{r=1}^{R} p(M_T|S_T^r)p(S_T^r)$$

Measurement sequence of length T



HMM: Naive sequence probability

- Find the probability of measurements: 🟢 🔴 🔴
- Weighted sum probabilities of every single state sequence

With X different hidden states, the number of possible sequences **S** of length T is: **R=X^T (here: 8=2^3)**

State sequence of length T

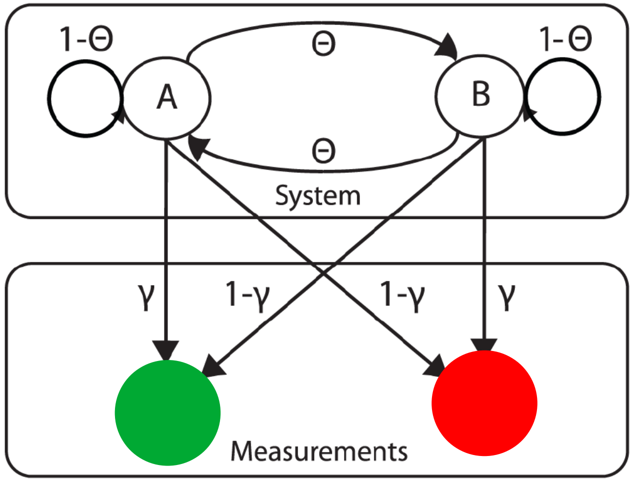
$$p(M_T) = \sum_{r=1}^{R} p(M_T|S_T^r)p(S_T^r)$$

🟢 🔴 🔴   🟢 🔴 🔴

AAA
AAB
ABB
...

AAA
AAB
ABB
...

Measurement sequence of length T



HMM: Naive sequence probability

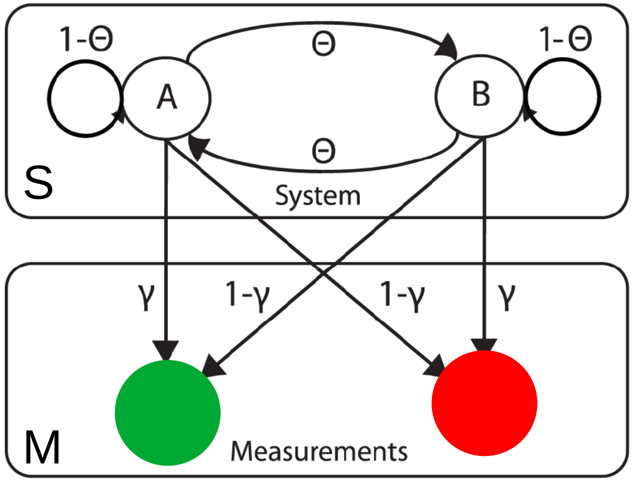
- Find the probability of measurements: 🟢 🔴 🔴
- Weighted sum probabilities of every single sequences

With X different hidden states, the number of possible sequences **S** of length T is: **R=X^T**

State sequence of length T

$$p(M_T) = \sum_{r=1}^{R} p(M_T|S_T^r)\underset{?}{}p(S_T^r)\underset{?}{}$$

Measurement sequence of length T



HMM: Naive sequence probability

- Find the probability of measurements: 🟢 🔴 🔴
- Weighted sum probabilities of every single sequences

Measurement at position t of M

$$p(M_T|S_r^T) = \prod_{t=1}^{T} p(m(t)|s(t))$$

State at position t of S

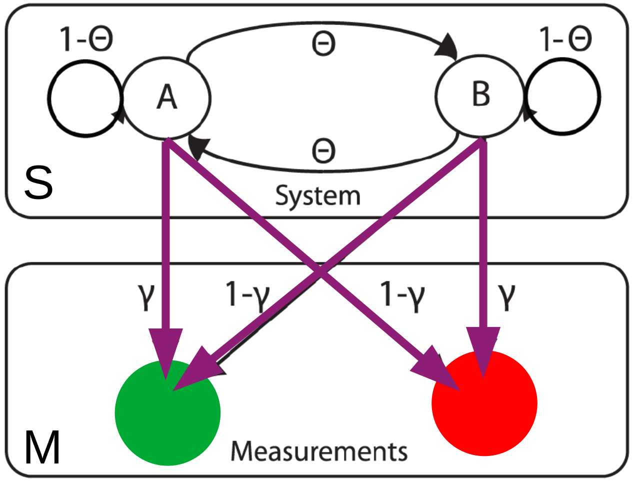
$$p(M_T) = \sum_{r=1}^{R} p(M_T|S_T^r)p(S_T^r)$$

HMM: Naive sequence probability

- Find the probability of measurements: 🟢 🔴 🔴
- Weighted sum probabilities of every single sequences

$$p(M_T|S_r^T) = \prod_{t=1}^{T} \underbrace{p(m(t)|s(t))}_{\text{Emission prob}}$$

$$p(M_T) = \sum_{r=1}^{R} p(M_T|S_T^r)p(S_T^r)$$



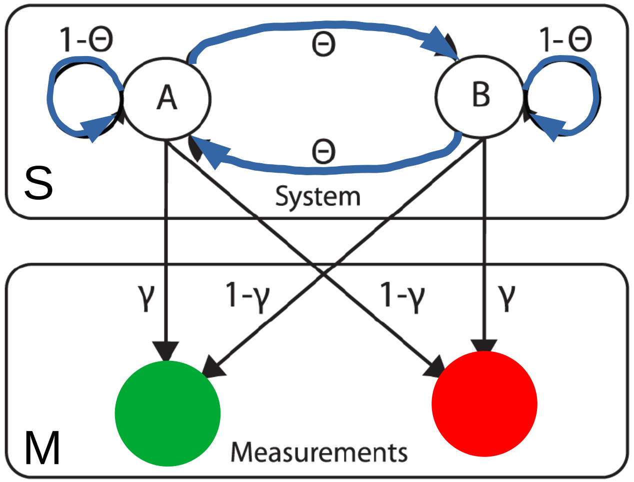
HMM: Naive sequence probability

- Find the probability of measurements: 🟢🔴🔴
- Weighted sum probabilities of every single sequences

$$p(M_T|S_r^T) = \prod_{t=1}^{T} \underbrace{p(m(t)|s(t))}_{\text{Emission prob}}$$

$$p(M_T) = \sum_{r=1}^{R} p(M_T|S_T^r)p(S_T^r)$$

$$p(S_T) = \prod_{t=1}^{T} p(s(t)|s(t-1))$$



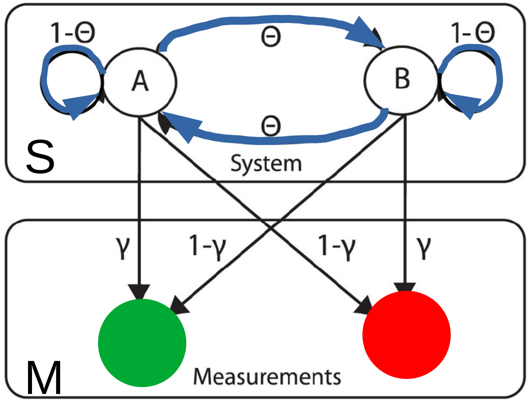
HMM: Naive sequence probability

- Find the probability of measurements: 🟢 🔴 🔴
- Weighted sum probabilities of every single sequences

$$p(M_T|S_r^T) = \prod_{t=1}^{T} \underbrace{p(m(t)|s(t))}_{\text{Emission prob}}$$

$$p(M_T) = \sum_{r=1}^{R} p(M_T|S_T^r)p(S_T^r)$$

$$p(S_T) = \prod_{t=1}^{T} \overbrace{p(s(t)|s(t-1))}^{\text{Transition prob}}$$



HMM: Naive sequence probability

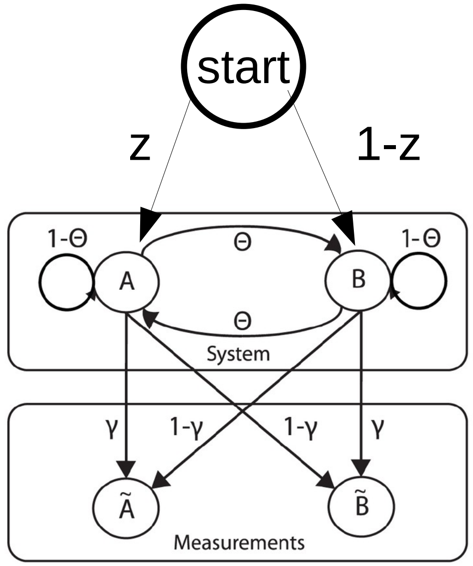
- Find the probability of measurements: 🟢 🔴 🔴
- Sum probabilities of every single sequences

$$p(M_T) = \sum_{r=1}^{R} \prod_{t=1}^{T} p(m(t)|s(t))p(s(t)|s(t-1))$$

What if t=0 and t-1 = -1 ?

HMM: Naive sequence probability

- Find the probability of measurements: 🟢 🔴 🔴
- Sum probabilities of every single sequences



$$p(M_T) = \sum_{r=1}^{R} \prod_{t=1}^{T} p(m(t)|s(t))p(s(t)|s(t-1))$$

What if t=0 and t-1 = -1 ?

HMM: Naive sequence probability

- Find the probability of measurements: 🟢 🔴 🔴
- Sum probabilities of every single sequences

Example:

P(GRR) = Σ |
p(GRR|AAA) = p(G1|A1) * z * p(R2|A2) * p(A2|A1) * p(R3|A3) * p(A3|A2)
= Ɣ * z * (1-Ɣ) * (1-Θ) * (1-Ɣ) * (1-Θ)
p(GRR|AAB) = ...
p(GRR|ABB) = …
…

$$p(M_T) = \sum_{r=1}^{R} \prod_{t=1}^{T} p(m(t)|s(t))p(s(t)|s(t-1))$$

HMM: Naive sequence probability

- Find the probability of measurements: 🟢 🔴 🔴
- Sum probabilities of every single sequences
- What is the time complexity according to T and X (number of states) ?

$$p(M_T) = \sum_{r=1}^{R} \prod_{t=1}^{T} p(m(t)|s(t))p(s(t)|s(t-1))$$

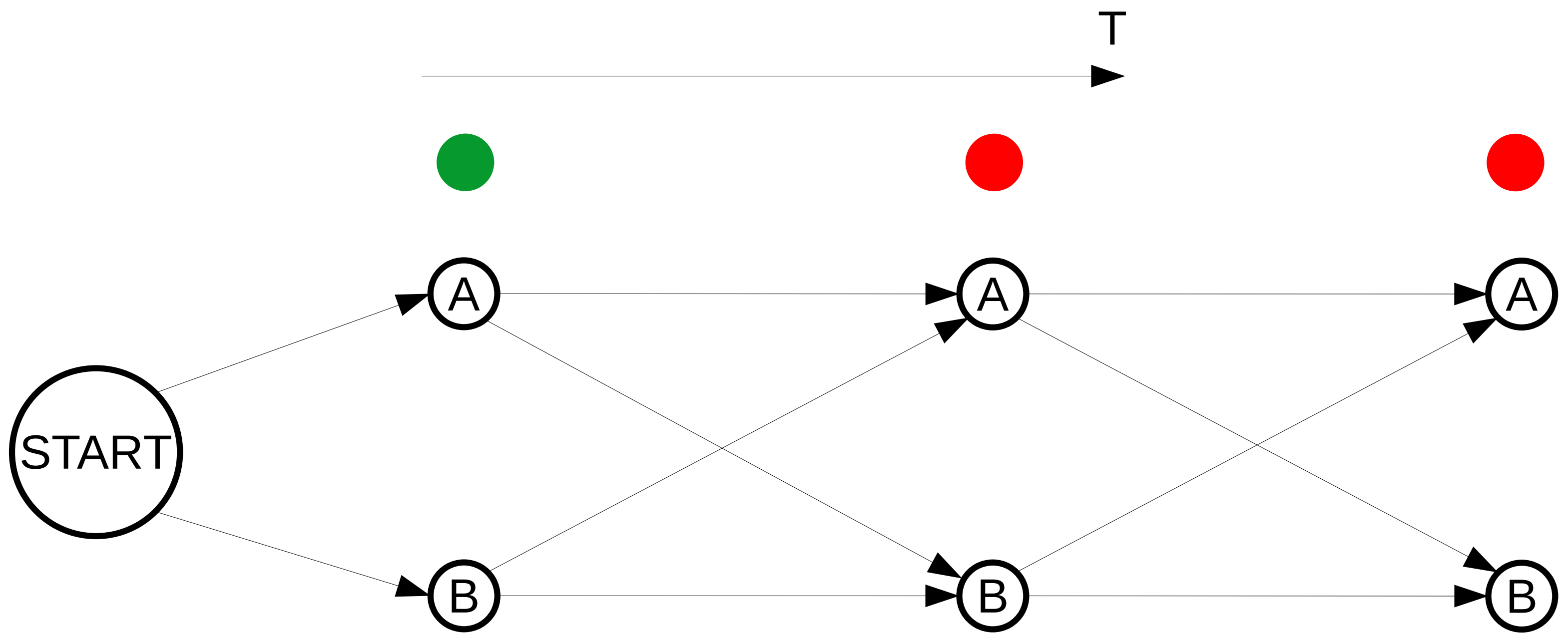
HMM: Naive sequence probability

- Find the probability of measurements: 🟢 🔴 🔴
- Sum probabilities of every single sequences
- What is the time complexity according to T and X (number of states) ? **O(X^T T)**

$$p(M_T) = \sum_{r=1}^{R} \prod_{t=1}^{T} p(m(t)|s(t))p(s(t)|s(t-1))$$

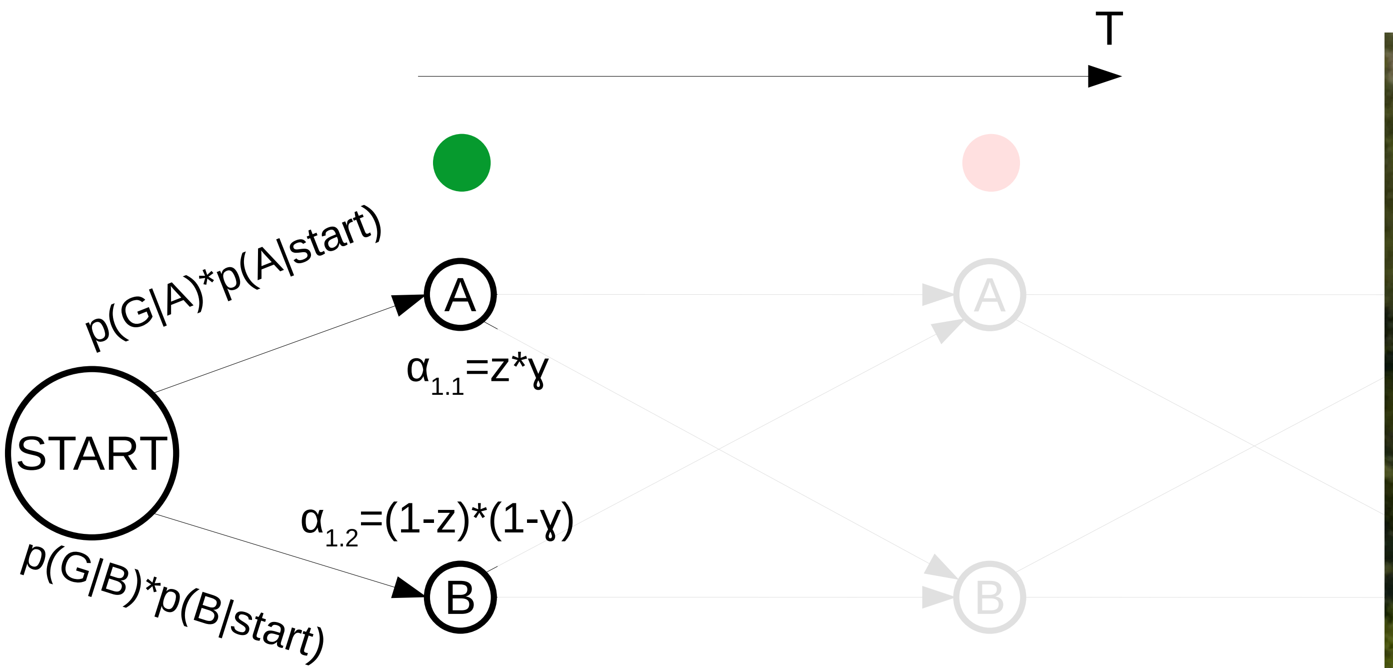
HMM: Forward algorithm

- Sequence probability can be solved using dynamic programming
- Avoid recomputing the same products many time



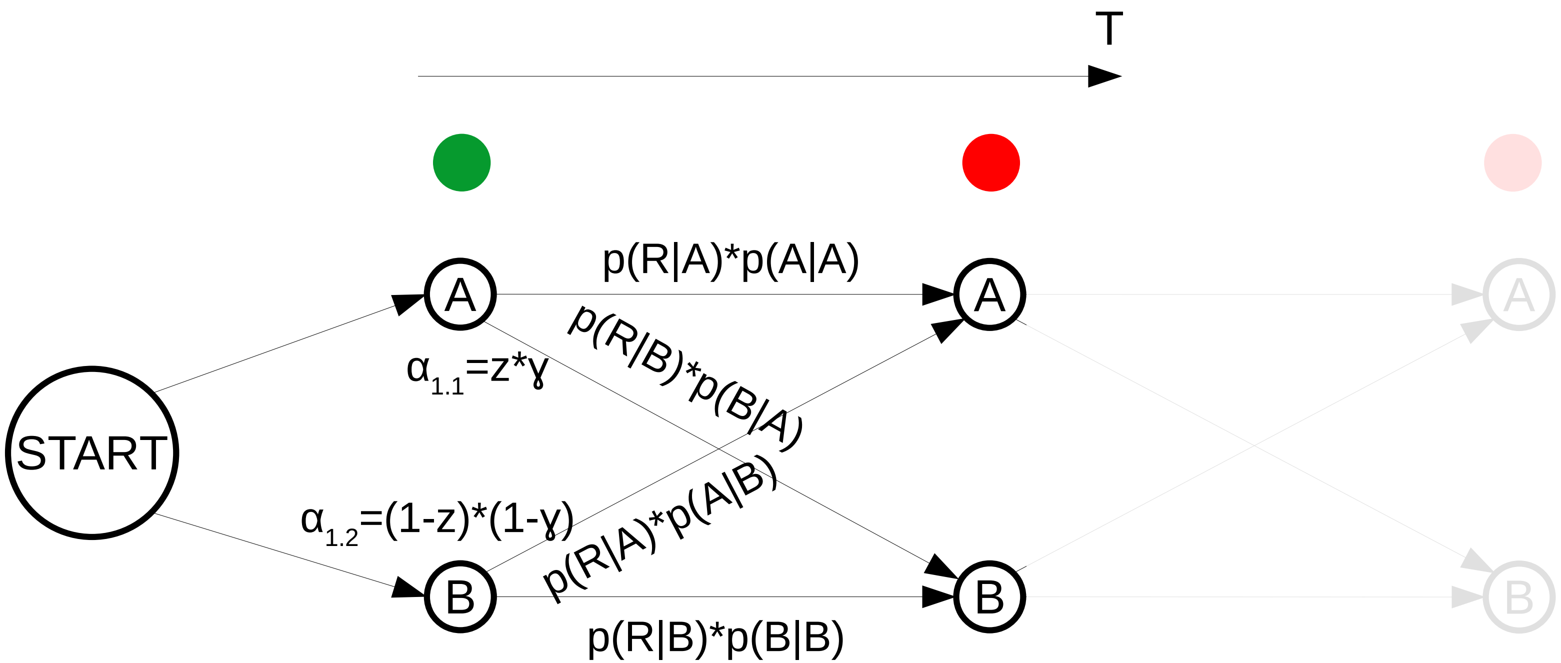
HMM: Forward algorithm

- Sequence probability can be solved using dynamic programming
- Avoid recomputing the same products many time



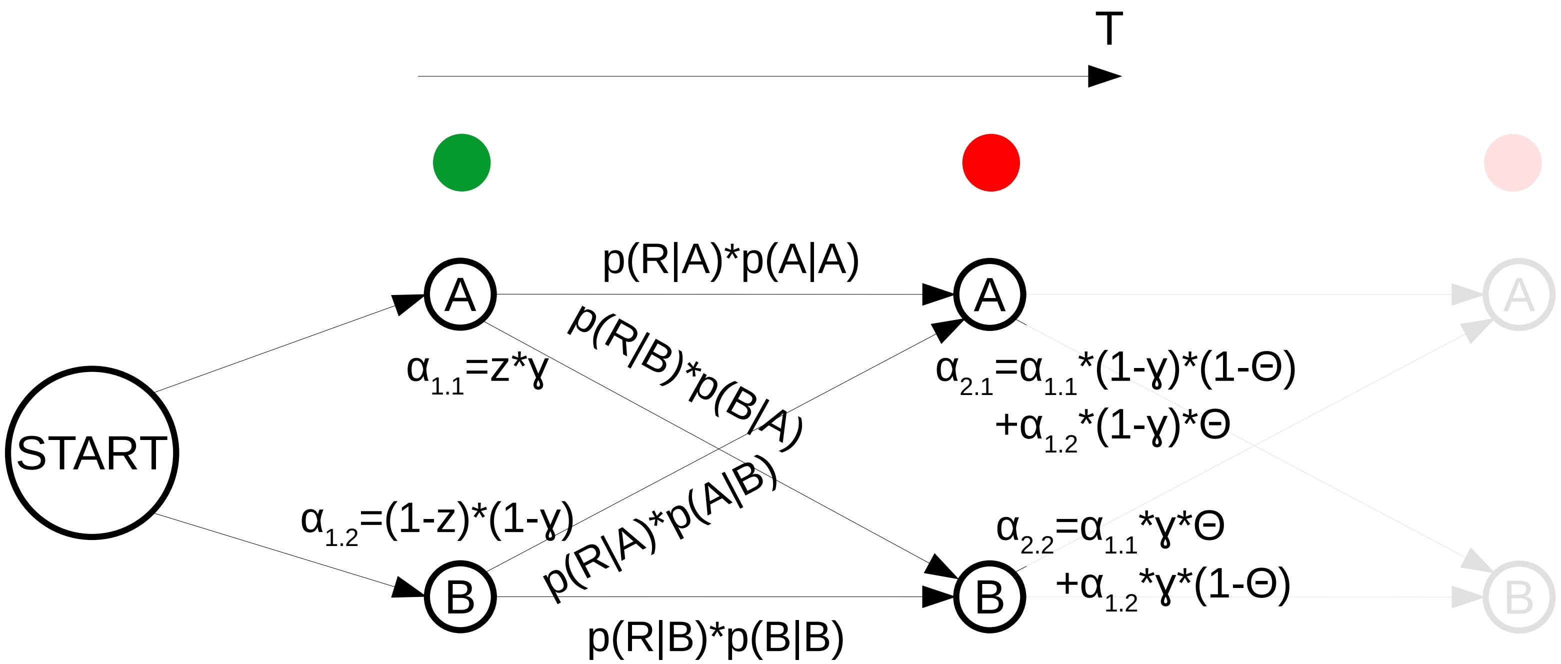
HMM: Forward algorithm

- Sequence probability can be solved using dynamic programming
- Avoid recomputing the same products many time



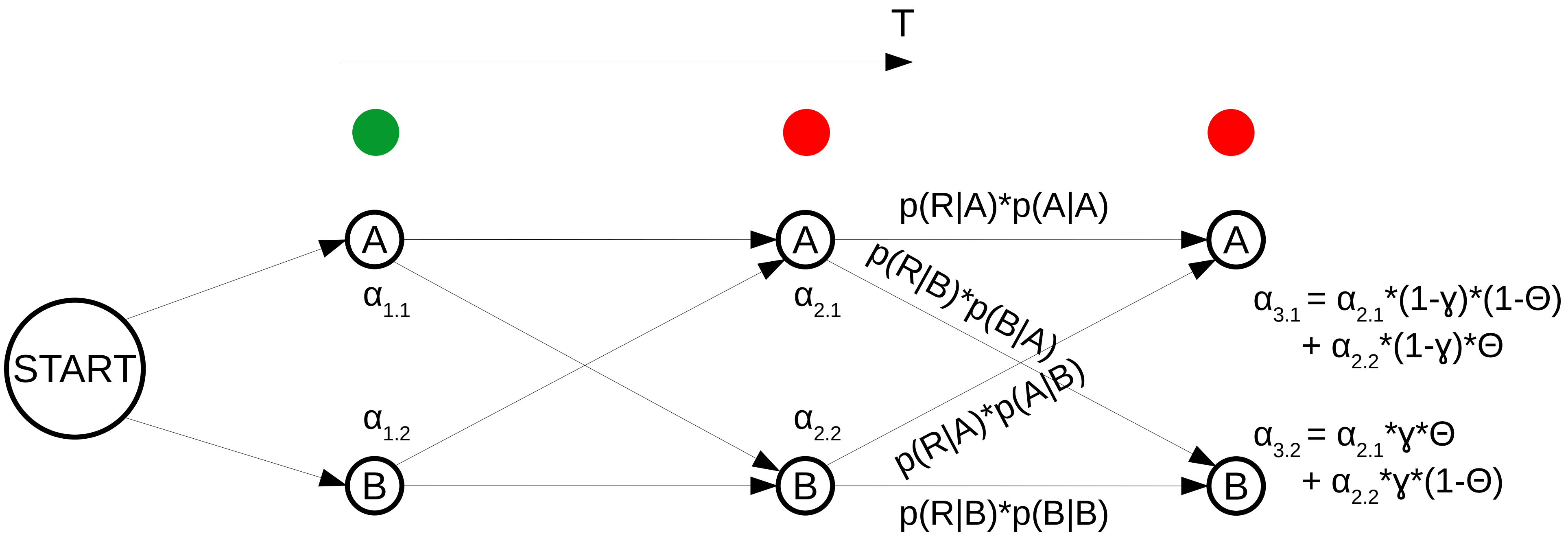
HMM: Forward algorithm

- Sequence probability can be solved using dynamic programming
- Avoid recomputing the same products many time


T
START
A
B
A
B
A
B
p(R|A)*p(A|A)
p(R|B)*p(B|A)
p(R|A)*p(A|B)
p(R|B)*p(B|B)
$\alpha_{1.1}=z*\gamma$
$\alpha_{1.2}=(1-z)*(1-\gamma)$
$\alpha_{2.1}=\alpha_{1.1}*(1-\gamma)*(1-\Theta)+\alpha_{1.2}*(1-\gamma)*\Theta$
$\alpha_{2.2}=\alpha_{1.1}*\gamma*\Theta+\alpha_{1.2}*\gamma*(1-\Theta)$

HMM: Forward algorithm

- Sequence probability can be solved using dynamic programming
- Avoid recomputing the same products many time

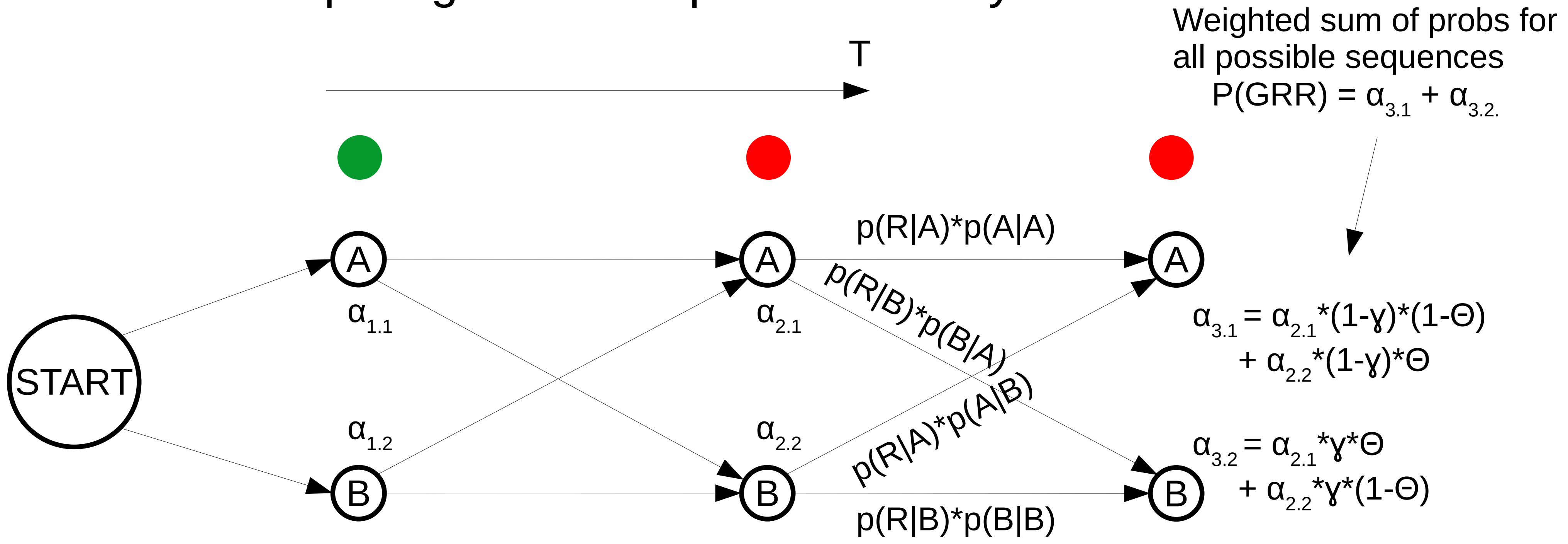


$\alpha_{3.1} = \alpha_{2.1} * (1-\gamma) * (1-\Theta)$
$+ \alpha_{2.2} * (1-\gamma) * \Theta$

$\alpha_{3.2} = \alpha_{2.1} * \gamma * \Theta$
$+ \alpha_{2.2} * \gamma * (1-\Theta)$

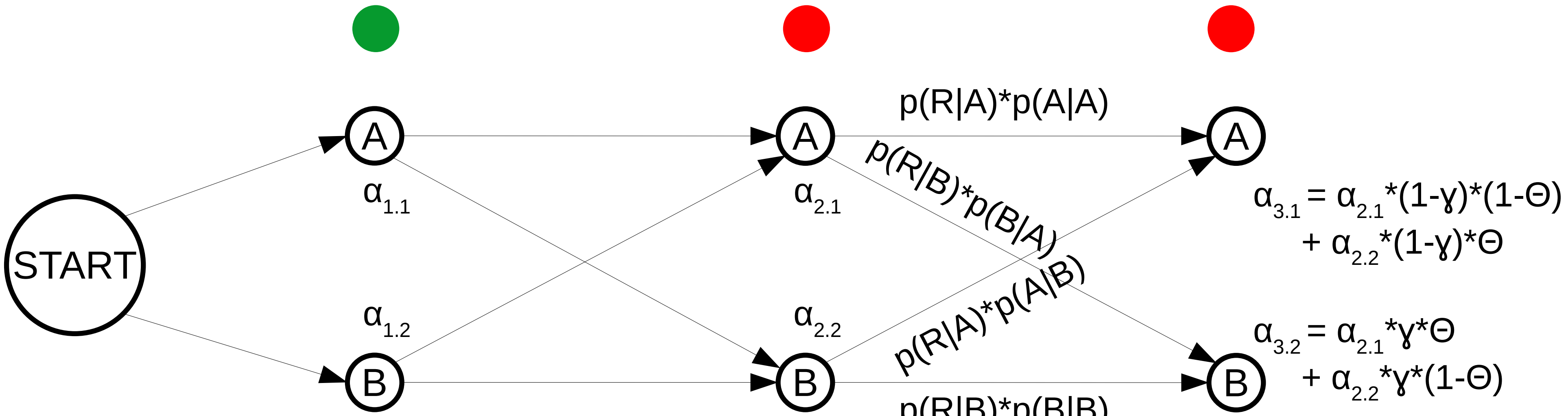
HMM: Forward algorithm

- Sequence probability can be solved using dynamic programming
- Avoid recomputing the same products many time


Weighted sum of probs for all possible sequences
$P(GRR) = \alpha_{3.1} + \alpha_{3.2.}$
T
START
A
B
$\alpha_{1.1}$
$\alpha_{1.2}$
$\alpha_{2.1}$
$\alpha_{2.2}$
p(R|A)*p(A|A)
p(R|B)*p(B|A)
p(R|A)*p(A|B)
p(R|B)*p(B|B)
$\alpha_{3.1} = \alpha_{2.1}*(1-\gamma)*(1-\Theta) + \alpha_{2.2}*(1-\gamma)*\Theta$
$\alpha_{3.2} = \alpha_{2.1}*\gamma*\Theta + \alpha_{2.2}*\gamma*(1-\Theta)$

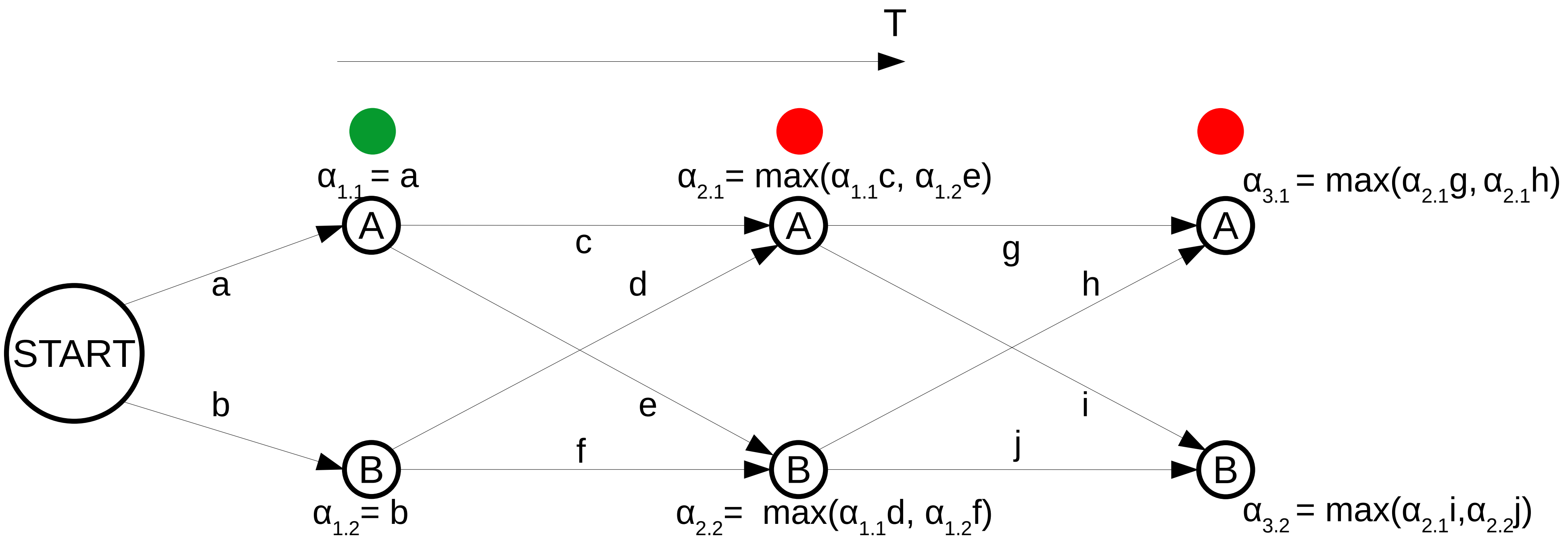
HMM: Forward algorithm

- Complexity of the forward algorithm is $O(TX^2)$ which is $<< O(TX^T)$
- Example use: Distinguishing between two models
  - e.g. Is my sequence more probable with a strand-aware model ?


START
A
B
$\alpha_{1.1}$
$\alpha_{1.2}$
$\alpha_{2.1}$
$\alpha_{2.2}$
p(R|A)*p(A|A)
p(R|B)*p(B|A)
p(R|A)*p(A|B)
p(R|B)*p(B|B)
$\alpha_{3.1} = \alpha_{2.1}*(1-\gamma)*(1-\Theta) + \alpha_{2.2}*(1-\gamma)*\Theta$
$\alpha_{3.2} = \alpha_{2.1}*\gamma*\Theta + \alpha_{2.2}*\gamma*(1-\Theta)$

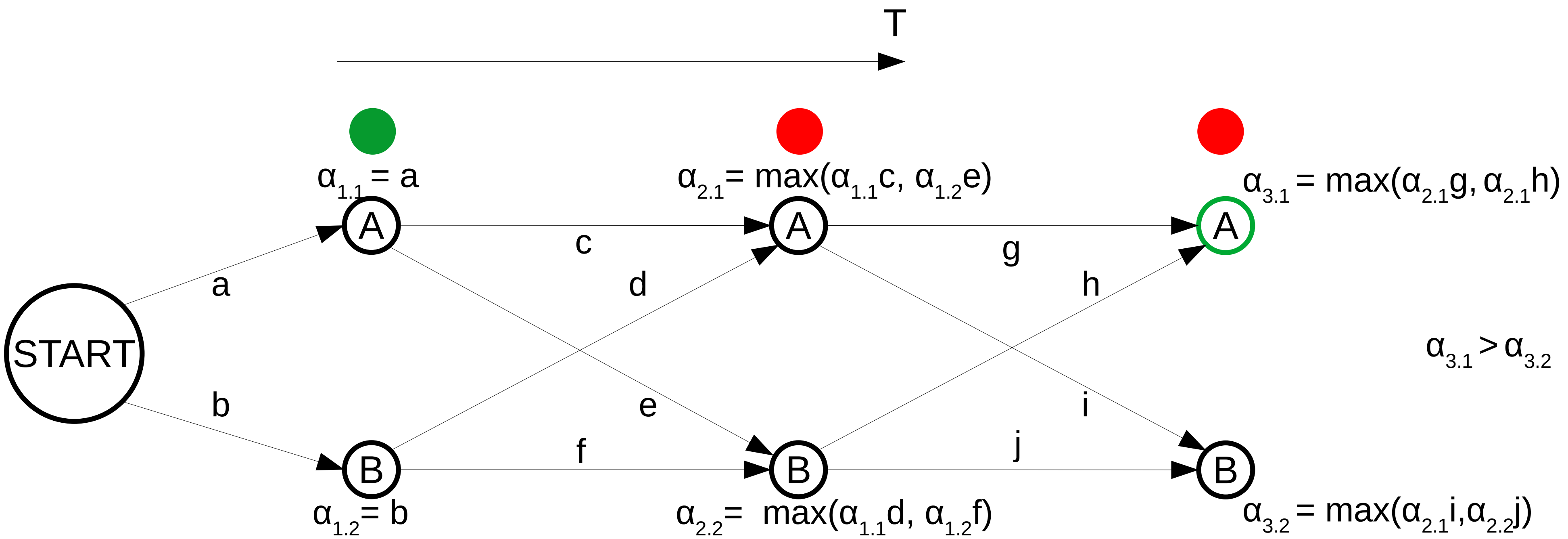
HMM: Viterbi algorithm

- What is the most likely sequence of states, given the observations ?
- Just like the forward algorithm, but using max() instead of sum()


T
$\alpha_{1.1} = a$
$\alpha_{2.1} = \max(\alpha_{1.1}c, \alpha_{1.2}e)$
$\alpha_{3.1} = \max(\alpha_{2.1}g, \alpha_{2.1}h)$
START
A
A
A
B
B
B
a
b
c
d
e
f
g
h
i
j
$\alpha_{1.2} = b$
$\alpha_{2.2} = \max(\alpha_{1.1}d, \alpha_{1.2}f)$
$\alpha_{3.2} = \max(\alpha_{2.1}i, \alpha_{2.2}j)$

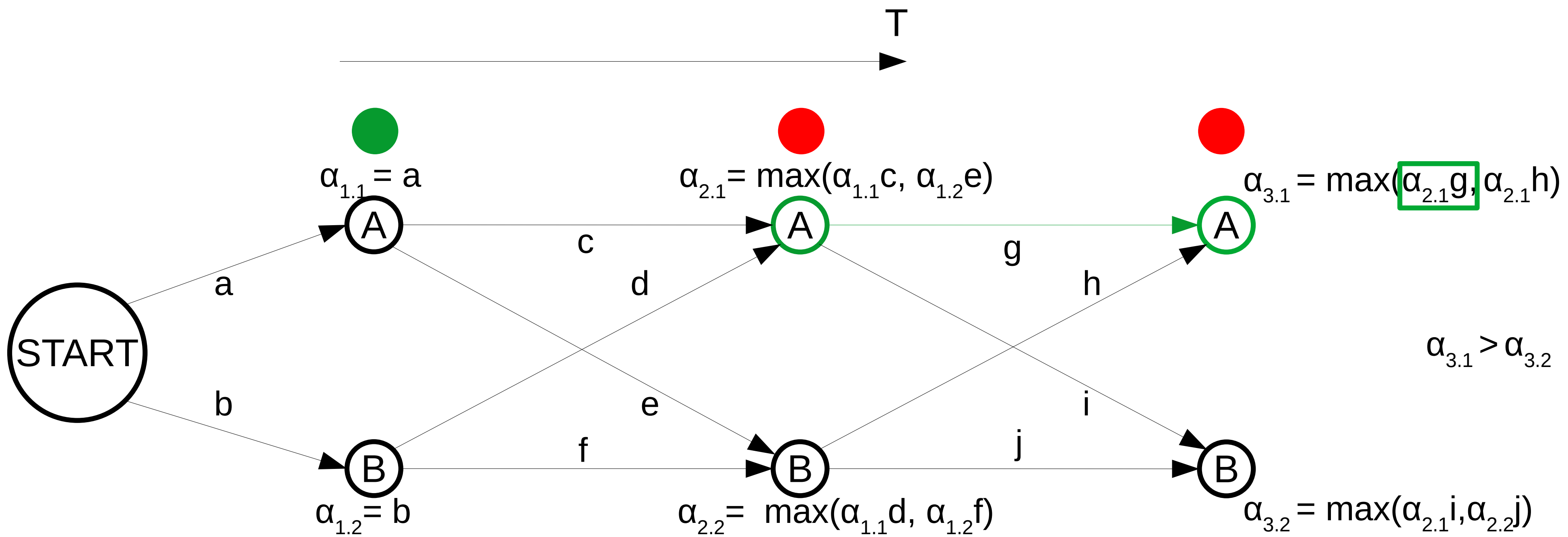
HMM: Viterbi algorithm

- Backtrack from T to 0 (similar to Needleman-Wunsch)


T
$\alpha_{1.1} = a$
$\alpha_{2.1} = \max(\alpha_{1.1}c, \alpha_{1.2}e)$
$\alpha_{3.1} = \max(\alpha_{2.1}g, \alpha_{2.1}h)$
START
A
B
a
b
c
d
e
f
g
h
i
j
$\alpha_{3.1} > \alpha_{3.2}$
$\alpha_{1.2} = b$
$\alpha_{2.2} = \max(\alpha_{1.1}d, \alpha_{1.2}f)$
$\alpha_{3.2} = \max(\alpha_{2.1}i, \alpha_{2.2}j)$

HMM: Viterbi algorithm

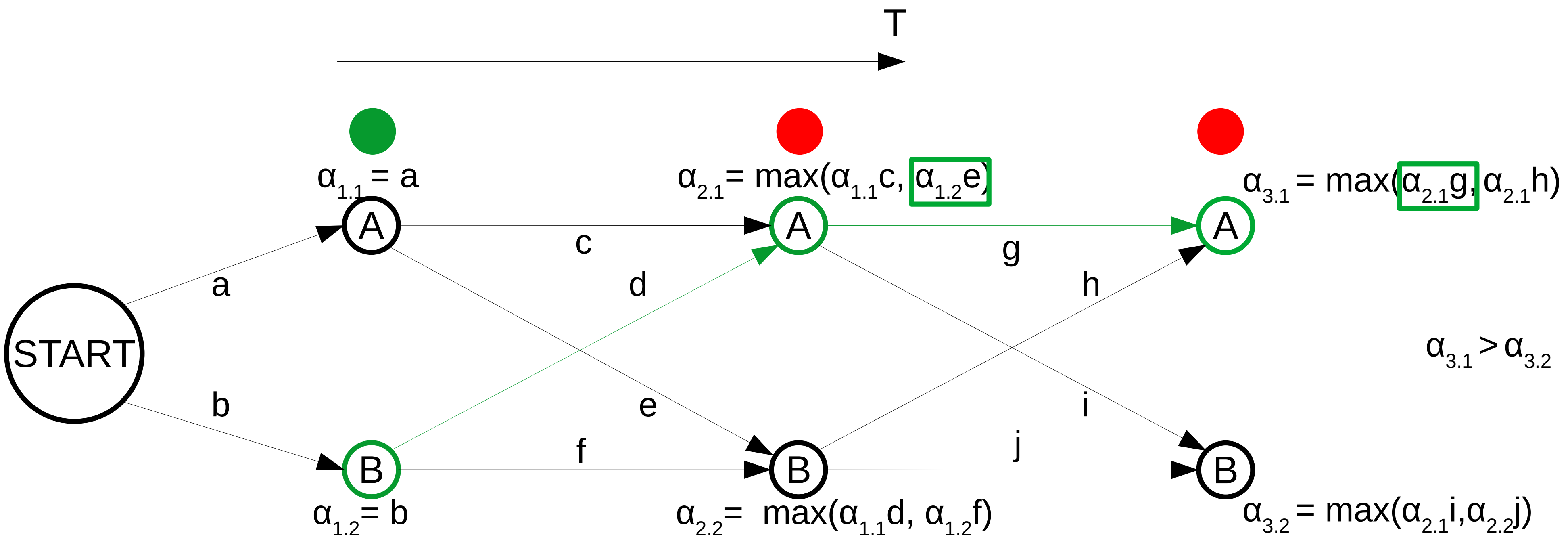
- Backtrack from T to 0 (similar to Needleman-Wunsch)



T
$\alpha_{1.1} = a$
$\alpha_{2.1} = \max(\alpha_{1.1}c, \alpha_{1.2}e)$
$\alpha_{3.1} = \max(\boxed{\alpha_{2.1}g,} \alpha_{2.1}h)$
START
A
A
A
B
B
B
a
b
c
d
e
f
g
h
i
j
$\alpha_{3.1} > \alpha_{3.2}$
$\alpha_{1.2} = b$
$\alpha_{2.2} = \max(\alpha_{1.1}d, \alpha_{1.2}f)$
$\alpha_{3.2} = \max(\alpha_{2.1}i, \alpha_{2.2}j)$

HMM: Viterbi algorithm

- Backtrack from T to 0 (similar to Needleman-Wunsch)



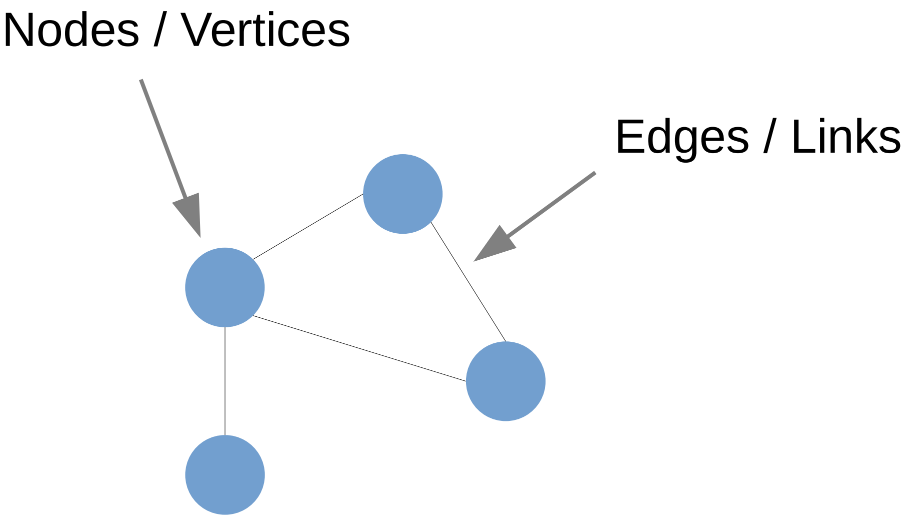
T
$\alpha_{1.1} = a$
$\alpha_{2.1} = \max(\alpha_{1.1}c, \boxed{\alpha_{1.2}e})$
$\alpha_{3.1} = \max(\boxed{\alpha_{2.1}g,} \alpha_{2.1}h)$
START
A
A
A
B
B
B
a
b
c
d
e
f
g
h
i
j
$\alpha_{3.1} > \alpha_{3.2}$
$\alpha_{1.2} = b$
$\alpha_{2.2} = \max(\alpha_{1.1}d, \alpha_{1.2}f)$
$\alpha_{3.2} = \max(\alpha_{2.1}i, \alpha_{2.2}j)$

HMM: Applications

- Gene annotation: Genome sequence → Intron/exon/splice site
  - AUGUSTUS
- Long read correction: Noisy long reads → most probable sequence
  - HERCULES
- Sequence alignment: Sequences → Insertion/Deletion/(Mis)match
  - HMMER
- Chromatin states: Histone marks → regulatory state
  - ChromHMM

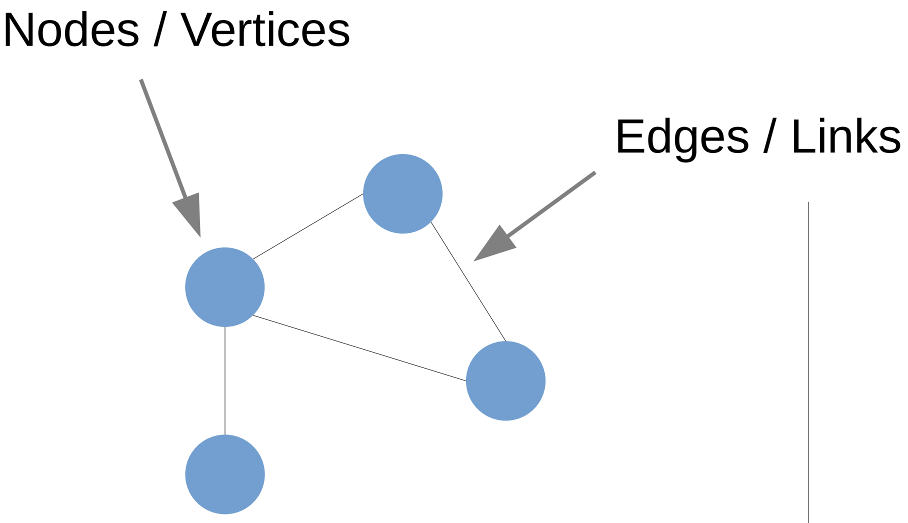
Graph theory

- Many biological systems can be represented with graphs
  - Protein interactions, gene regulation, residues in proteins
- Nodes and edges (Markov models are also graphs)



Graph theory

- Many biological systems can be represented with graphs
  - Protein interactions, gene regulation, residues in proteins
- Nodes and edges (Markov models are also graphs)

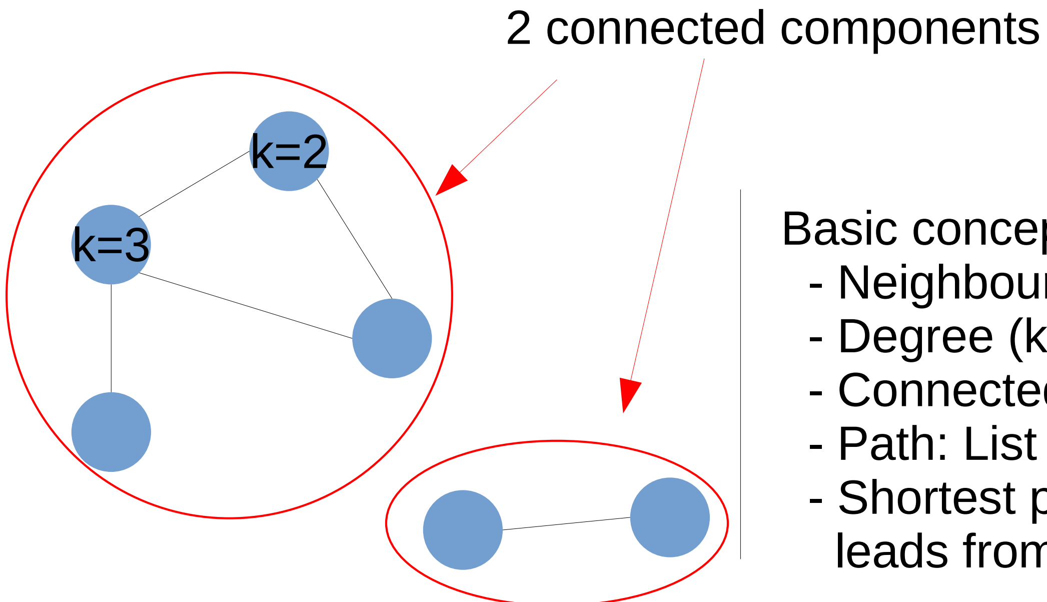


A graph is a set of nodes V and edges E: $G = (V, E)$

Number of nodes: $|V| = N$
Number of edges: $|E|$

Graph theory

- Various metrics can give information about a graph or its nodes

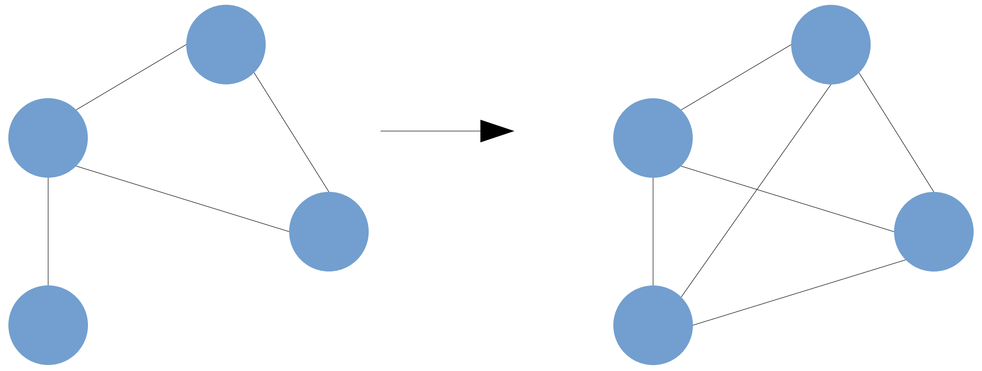


Basic concepts:
- Neighbours: adjacent nodes
- Degree (k): Number of neighbours
- Connected component: Group of connected nodes
- Path: List of adjacent nodes
- Shortest path: Shortest list of adjacent nodes which leads from one node to another

Graph theory

- Maximum number of edges in a network of N nodes ?
  - Hint: How many max edges / node ?

Example with 4 nodes:

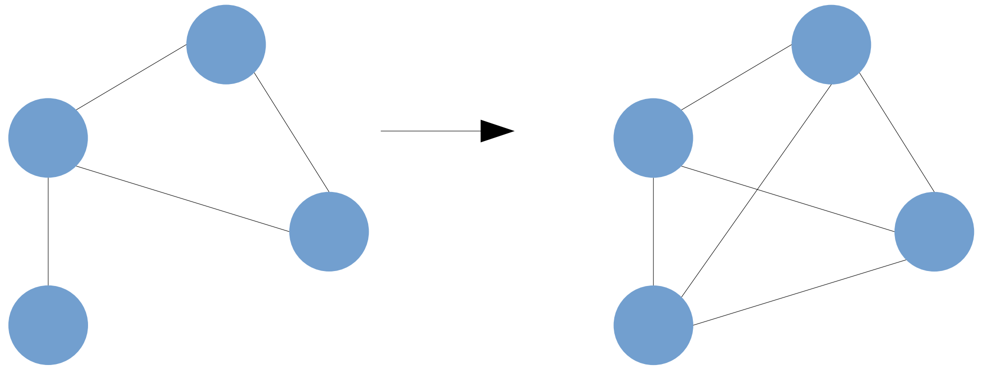


Fully connected:
4 Nodes, 6 edges

Graph theory

- Maximum number of edges in a network of N nodes ? **(N-1)!**
  - Hint: How many max edges / node ?
- In practice, edges scale in O(N) for most networks (“sparse”)

Example with 4 nodes:

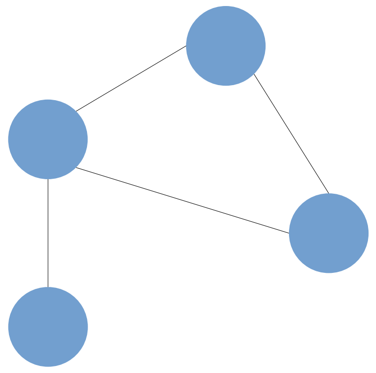


Fully connected:
4 Nodes, 6 edges

Graph theory

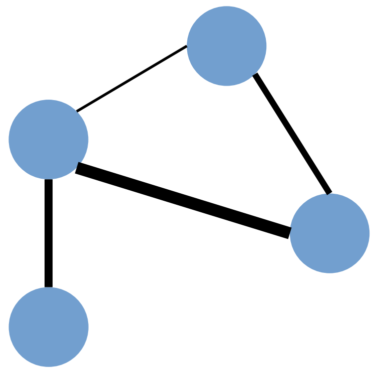
- Edges can be 0/1 or have weights or even directions
- This allows to model different types of processes

Undirected



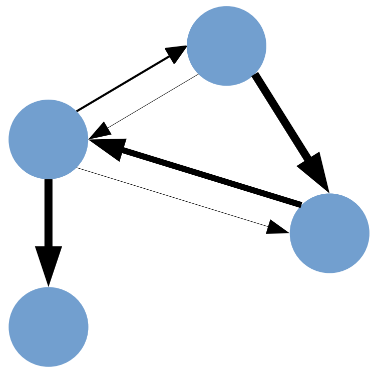
Ex: interacts

Weighted



Ex: binding affinity

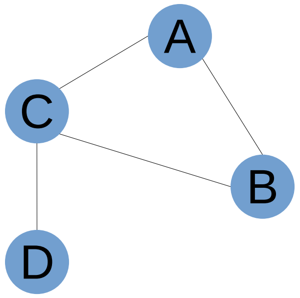
Directed



Ex: promotes/inhibits

Mathematical representation of graphs

- To work with graph, we need to use a formal representation
- There are 3 common representations:



Adjacency list

A – B
A – C
B – C
C – D

Adjacency matrix

$$\begin{array}{ccccc} & A & B & C & D \\ A & 0 & 1 & 1 & 0 \\ B & 1 & 0 & 1 & 0 \\ C & 1 & 1 & 0 & 1 \\ D & 0 & 0 & 1 & 0 \end{array}$$

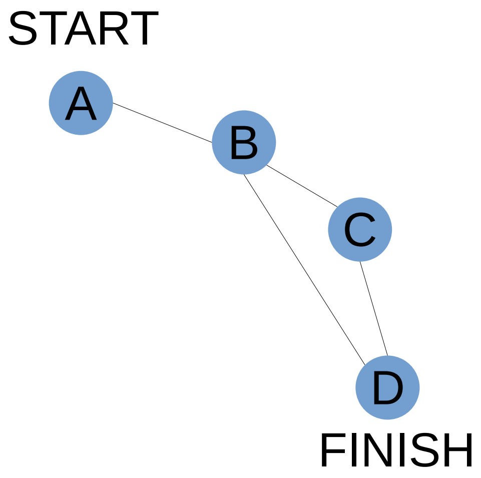
Note: for undirected graphs: $A[i, j] = A[j,i]$

Neighbour list

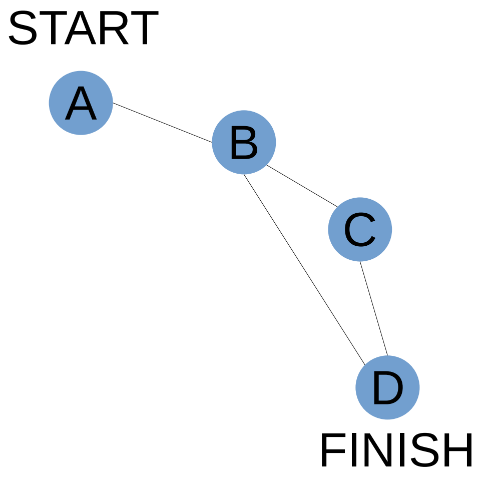
A – B,C
B – A,B
C – A,B,D
D – C

Random walks on a graph

- Imagine you move randomly on the graph
- Probability to go from node A to D in 2 steps ?



Random walks on a graph

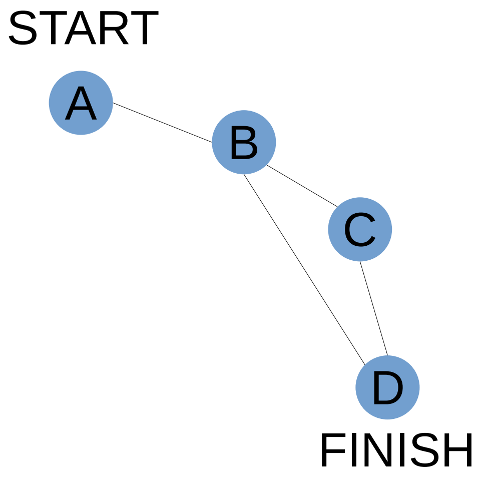


- Imagine you move randomly on the graph
- Probability to go from node A to D in 2 steps ?

All possible paths:

ABA
ABC
ABD 1 success / 3 paths = 0.33

Random walks on a graph

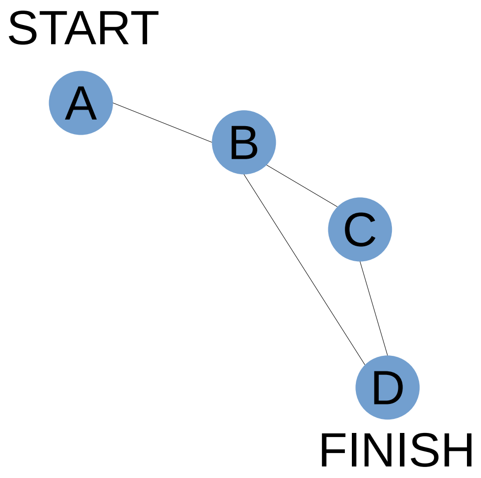


- As the graph grows larger, it becomes infeasible to list each path
- There are $O(K^T)$ paths of length T where K is the average degree

All possible paths:

ABA
ABC
ABD 1 success / 3 paths = 0.33

Random walks, but faster



- All neighbours of A $\rightarrow$ X^1
- All the neighbours of X^1 $\rightarrow$ X^2

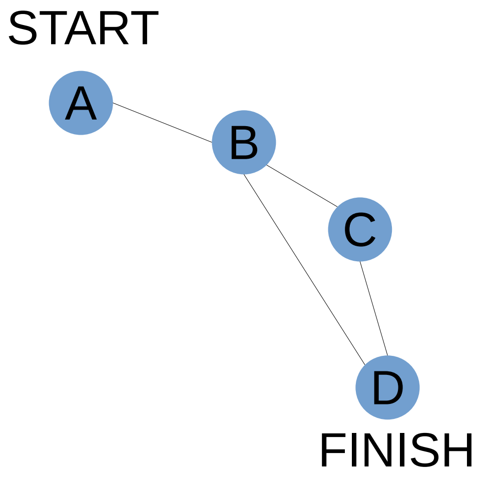
1) Probability of A $\rightarrow$ X^1: $\boxed{P(A, X^1)}$
2) Probability of X^1 $\rightarrow$ X^2: $P(X^1, X^2)$
3) Sum $P(A, X^1)$ x $\boxed{P(X^1, D)}$, all cases reaching D

4) Generalization to N steps:

$$\sum P(A, X^1)P(X^1, X^2)...P(X^{N-1}, D)$$

Random walks, but faster

- All neighbours of A $\rightarrow$ X^1
- All the neighbours of X^1 $\rightarrow$ X^2



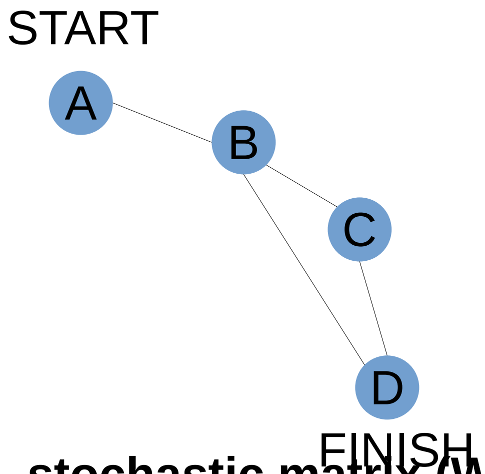
1) Probability of A $\rightarrow$ X^1: $\boxed{P(A, X^1)}$
2) Probability of X^1 $\rightarrow$ X^2: $P(X^1, X^2)$
3) Sum $P(A, X^1)$ x $\boxed{P(X^1, D)}$, all cases reaching D

4) Generalization to N steps:

$$\sum P(A, X^1)P(X^1, X^2)...P(X^{N-1}, D)$$

	A	B	C	D
A	0	1	0	0
B	1	0	1	1
C	0	1	0	1
D	0	1	1	0

Random walks, but faster



- All neighbours of A → X^1
- All the neighbours of X^1 → X^2

1) Probability of A → X^1: $\boxed{P(A, X^1)}$
2) Probability of X^1 → X^2: $P(X^1, X^2)$
3) Sum $P(A, X^1)$ x $\boxed{P(X^1, D)}$, all cases reaching D

4) Generalization to N steps:

$$\sum P(A, X^1)P(X^1, X^2)...P(X^{N-1}, D)$$

Adjacency → stochastic matrix (W)

	A	B	C	D
A	0	1	0	0
B	0.33	0	0.33	0.33
C	0	0.5	0	0.5
D	0	0.5	0.5	0

Random walks, but faster

- The T'th power of the stochastic matrix gives the probability of going from node i to j in T steps
- Scales with O(T) instead of O(K^T)

P(A, D) in 1 step: W[A, D]
P(A, D) in 2 steps: W^2[A,D]
P(A, D) in T steps: W^T[A, D]

	A	B	C	D
A	0	1	0	0
B	0.33	0	0.33	0.33
C	0.	0.5	0	0.5
D	0	0.5	0.5	0

X

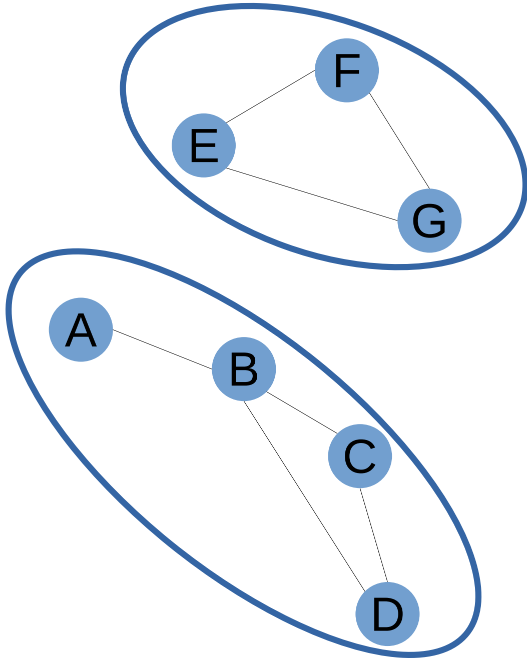
	A	B	C	D
A	0	1	0	0
B	0.33	0	0.33	0.33
C	0.	0.5	0	0.5
D	0	0.5	0.5	0

=

	A	B	C	D
A	0.33	0	0.33	0.33
B	0	0.66	0.17	0.17
C	0.17	0.5	0.42	0.17
D	0.17	0.5	0.17	0.42

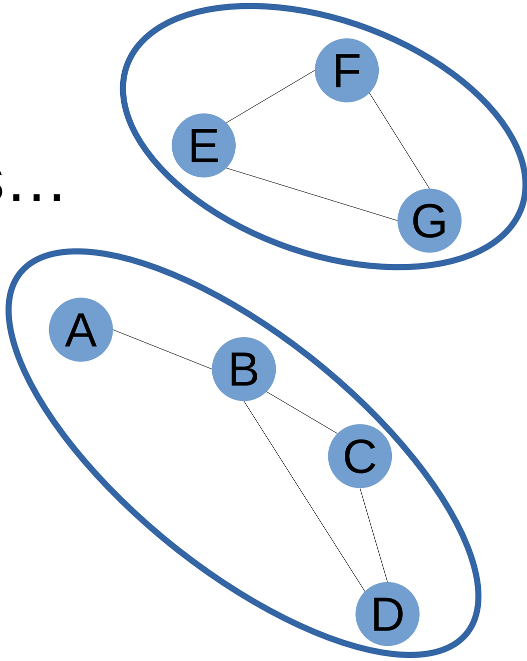
Network clustering

- Easiest scenario: Identify all disconnected components
- How would you do it ?



Identification of disconnected components

- Easiest scenario: Identify all disconnected components
- How would you do it ?
  - Start from any node
  - Explore neighbours, neighbours of neighbours…
  - Skip nodes already visited



Identification of disconnected components

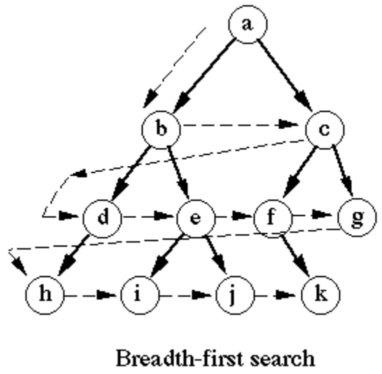
- Breadth first search

```
visited = []
queue = [initial]

while queue:
    node = queue.pop(0)

    if node not in visited:
        visited.append(node)
        neighbours = graph[node]

            for neighbour in neighbours:
                queue.append(neighbour)
```



Repeat with the remaining nodes, if any

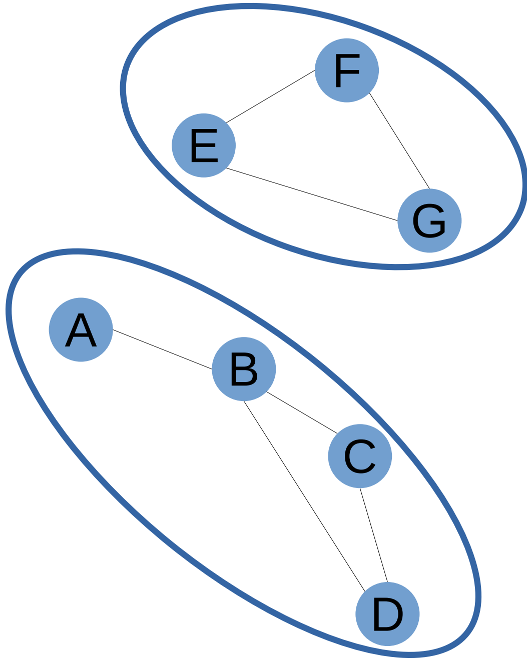


Image: https://www.cse.unsw.edu.au/~billw/Justsearch.html

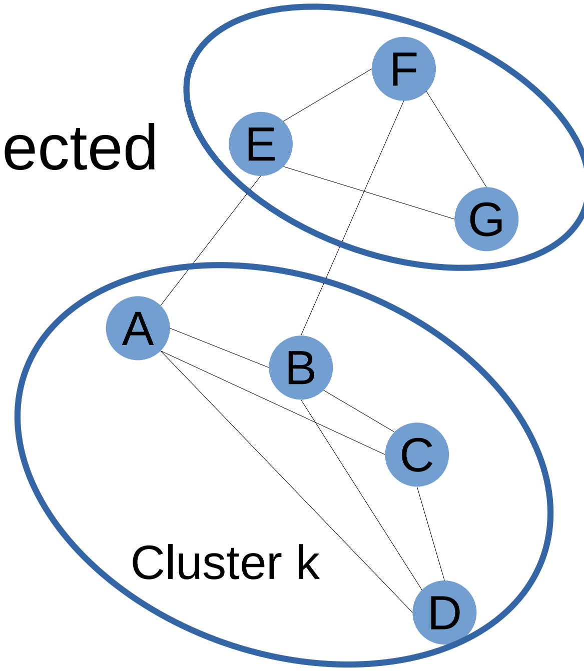
How to define clusters within a network ?

- We need a metric to quantify the accuracy of clusters
- We use modularity
  - Nodes in a cluster should have more edges among them than with other nodes than expected by chance

Number of edges among nodes in k: $l_k = 6$

Number of edges between k and outside: $o_k = 2$

Sum of nodes degrees in k: $d_k = 2l_k + o_k = 14$



How to define clusters within a network ?

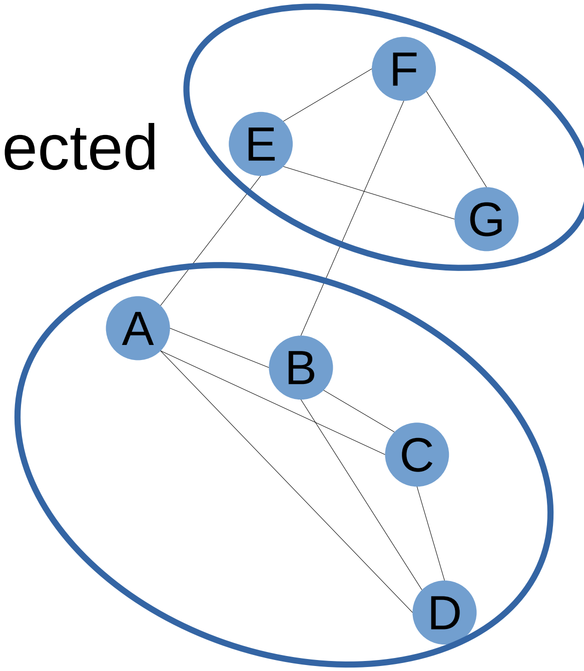
- We need a metric to quantify the accuracy of clusters
- We use modularity
  - Nodes in a cluster should have more edges among them than with other nodes than expected by chance

Probability that one edge is in cluster:

$$\frac{d}{2|E|}$$

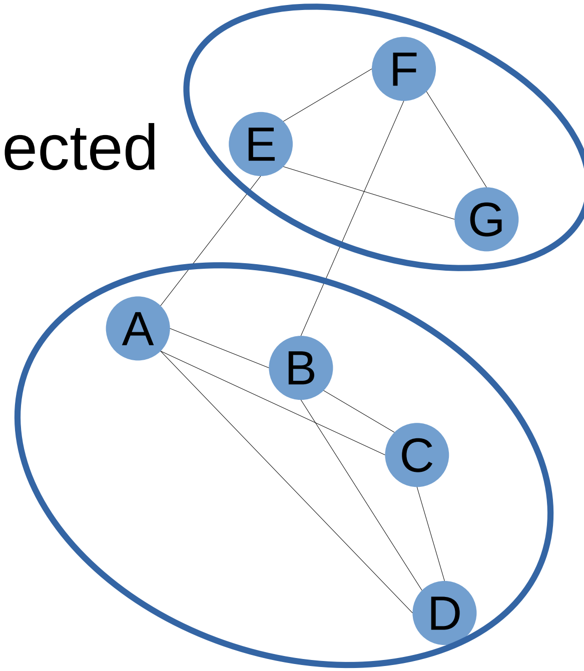
Number of possible edges in k

Number of possible edges in the whole network



How to define clusters within a network ?

- We need a metric to quantify the accuracy of clusters
- We use modularity
  - Nodes in a cluster should have more edges among them than with other nodes than expected by chance



Expected number of internal nodes in a cluster:

$$\frac{d}{2|E|} \cdot d \cdot \frac{1}{2} = \frac{d^2}{4|E|}$$

How to define clusters within a network ?

- We need a metric to quantify the accuracy of clusters
- We use modularity
  - Nodes in a cluster should have more edges among them than with other nodes than expected by chance

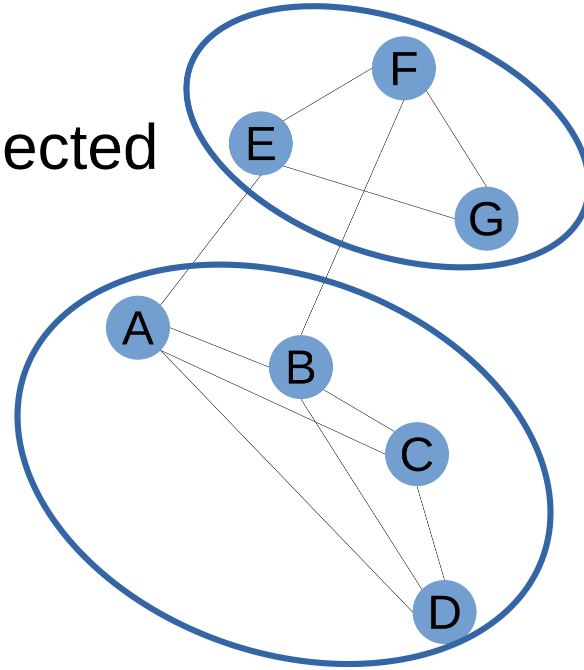
Normalizing factor

# expected internal edges

Network modularity:

$$Q = \frac{1}{|E|} \sum_{k=1}^{K} \left(l_k - \frac{d_k^2}{4|E|} \right)$$

# observed internal edges

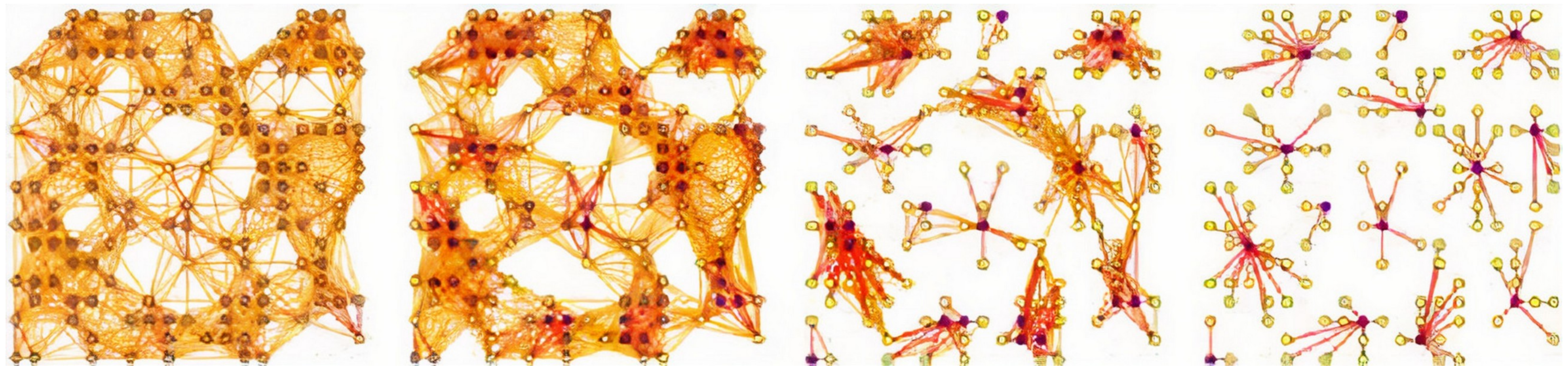


Network clustering

- There are several algorithms to find clusters in a network
- Modularity can be used to measure the validity of the clusters
- Hierarchical clustering: Merge nodes until modularity stops increasing.
- Markov clustering: Number of clusters automatically detected

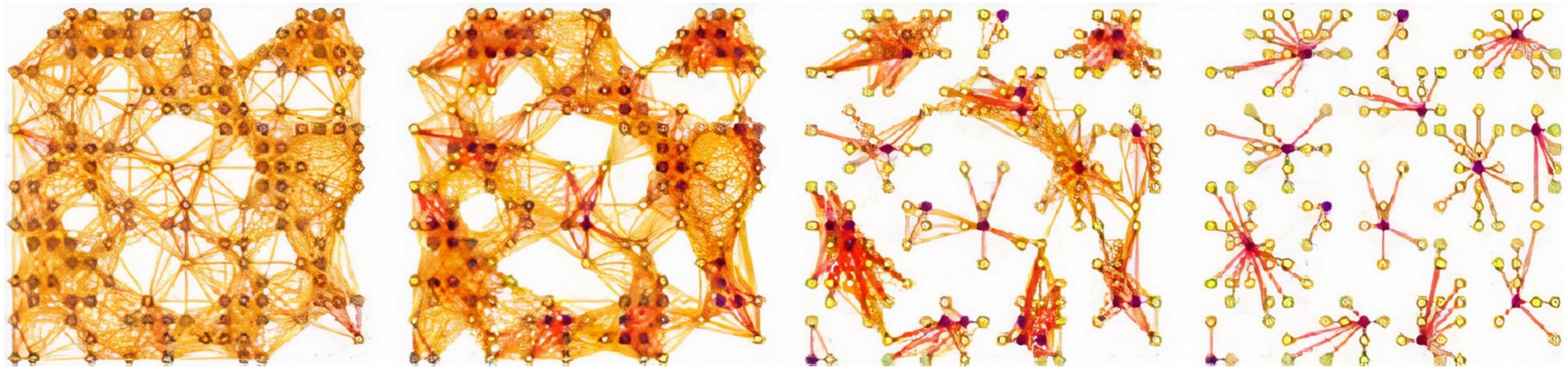
Markov clustering

- Consider the network as a markov process
- Nodes as states, edges as transition probabilities
- Random walks on the graph will infrequently go from one natural cluster to another



Markov clustering

- 3 operations repeated until convergence:
  - Expansion: Simulate random walk on the graph
  - Pruning: Remove the least frequented (low probability) paths
  - Inflation: Increase contrast between probabilities



Markov clustering

Convert to transition probabilities

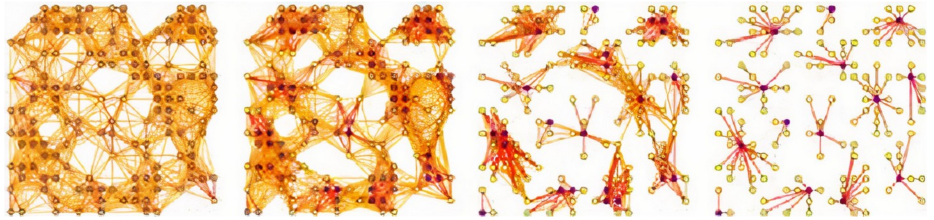
$$\begin{tikzcd}
A \arrow[r] & \boxed{A \text{ -> } W} \arrow[d] \\
& \boxed{W \text{ -> } W^2} \arrow[d] \\
& \boxed{(W^2)_{ij} \text{-> } [(W^2)_{ij}]^r} \arrow[d] \\
& \boxed{M[M<p] = 0} \arrow[uuu, bend right=60]
\end{tikzcd}$$

Expansion: Enhance flow to well connected nodes

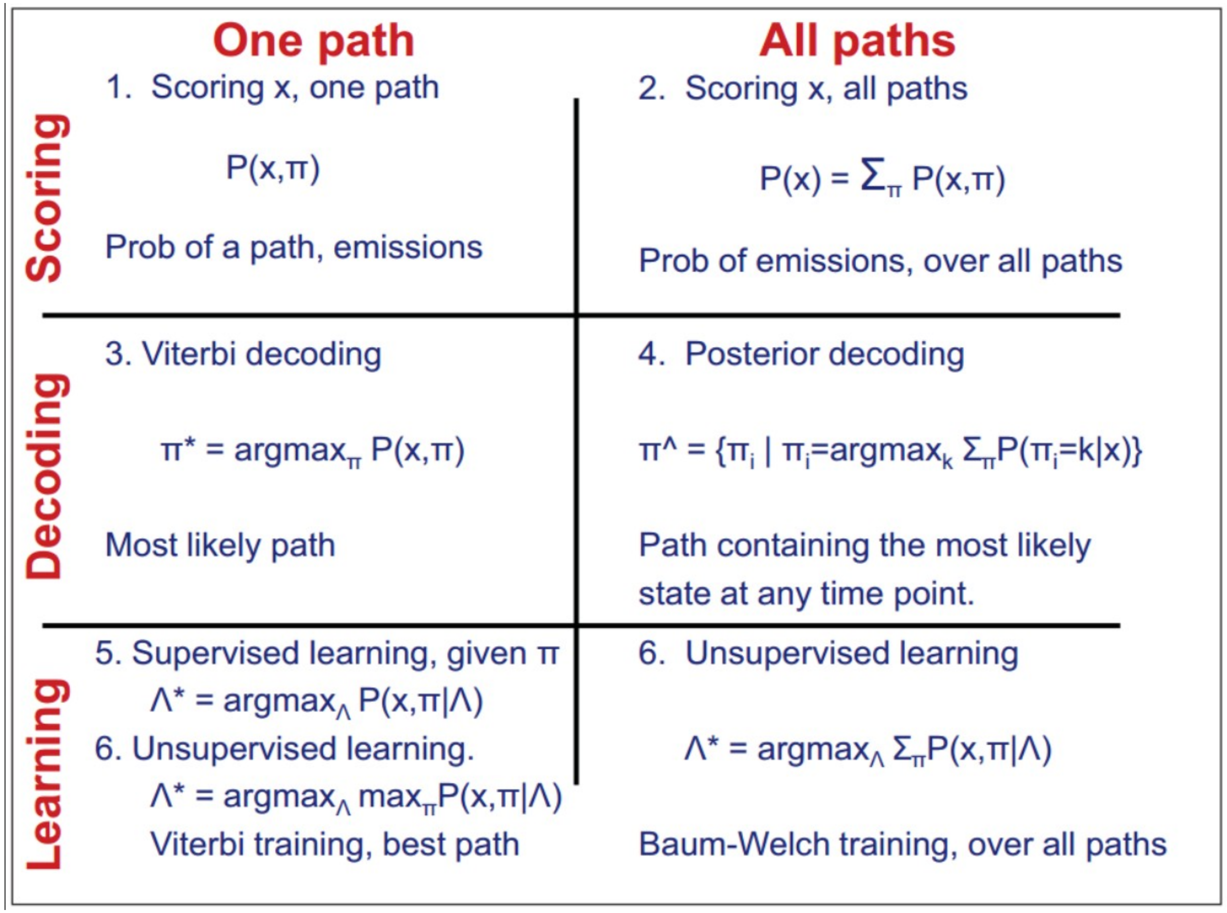
Inflation: Increase inequalities among nodes

Pruning: Remove near-0 values to accelerate convergence

Repeat until convergence: matrix values change very little



If you want to learn more about HMM



	One path	All paths
Scoring	1. Scoring x, one path $P(x,\pi)$ Prob of a path, emissions	2. Scoring x, all paths $P(x) = \Sigma_\pi P(x,\pi)$ Prob of emissions, over all paths
Decoding	3. Viterbi decoding $\pi^* = \text{argmax}_\pi P(x,\pi)$ Most likely path	4. Posterior decoding $\pi\hat{} = \{\pi_i \mid \pi_i=\text{argmax}_k \Sigma_\pi P(\pi_i=k \mid x)\}$ Path containing the most likely state at any time point.
Learning	5. Supervised learning, given π $\Lambda^* = \text{argmax}_\Lambda P(x,\pi \mid \Lambda)$ 6. Unsupervised learning. $\Lambda^* = \text{argmax}_\Lambda \max_\pi P(x,\pi \mid \Lambda)$ Viterbi training, best path	6. Unsupervised learning $\Lambda^* = \text{argmax}_\Lambda \Sigma_\pi P(x,\pi \mid \Lambda)$ Baum-Welch training, over all paths

6.047/6.878 Computational Biology: Genomes, Networks, Evolution (Fall 2012) Lecture 07 - HMMs II (2012-09-29): `link

The main questions on HMMs

1. Scoring x, one path = Joint probability of a sequence and a path, given the model

- GIVEN a HMM M, a path π, and a sequence x,
- FIND Prob[x, π | M]

➔ "Running the model", simply multiply emission and transition probabilities

➔ Application: "all promoter" vs. "all backgorund" comparisons

2. Scoring x, all paths = total probability of a sequence, summed across all paths

- GIVEN a HMM M, a sequence x
- FIND the total probability P[x | M] summed across all paths

➔ Forward algorithm, sum score over all paths (same result as backward)

SCORING

3. Viterbi decoding = parsing a sequence into the optimal series of hidden states

- GIVEN a HMM M, and a sequence x,
- FIND the sequence π^* of states that maximizes P[x, π | M]

➔ Viterbi algorithm, dynamic programming, max score over all paths, trace pointers find path

4. Posterior decoding = total prob that emission x_i came from state k, across all paths

- GIVEN a HMM M, a sequence x
- FIND the total probability P[π_i = k | x, M)

➔ Posterior decoding: run forward & backward algorithms to & from state π_I =k

PARSING

5. Supervised learning = optimize parameters of a model given training data

- GIVEN a HMM M, with unspecified transition/emission probs., labeled sequence x,
- FIND parameters $\theta = (e_i, a_{ij})$ that maximize P[x | θ]

➔ Simply count frequency of each emission and transition observed in the training data

6. Unsupervised learning = optimize parameters of a model given training data

- GIVEN a HMM M, with unspecified transition/emission probs., unlabeled sequence x,
- FIND parameters $\theta = (e_i, a_{ij})$ that maximize P[x | θ]

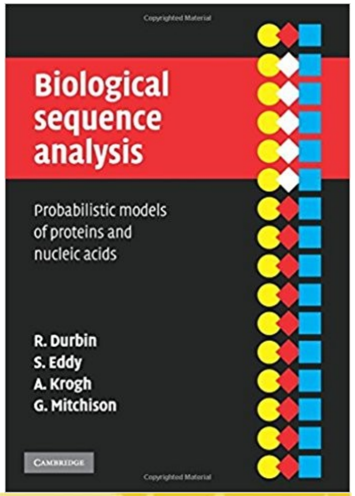
➔ Viterbi training: guess parameters, find optimal Viterbi path (#2), update parameters (#5), iterate

➔ Baum-Welch training: guess, sum over all emissions/transitions (#4), update (#5), iterate

LEARNING

Additional Resources

- Yoon, 2009, Curr. Genomics: Review on the applications of HMM in genomics: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2766791/
- Blog post on Markov clustering: https://medium.com/analytics-vidhya/demystifying-markov-clustering-aeb6cdabbfc7
- Book on graph theory: Graphs, Networks and Algorithms, Dieter Jungnickel, 2013
- Book on probabilitic models in genomics: Biological Sequence Analysis Probabilistic Models of Proteins and Nucleic Acids by Richard Durbin, Sean R. Eddy, Anders Krogh, Graeme Mitchison


Copyrighted Material

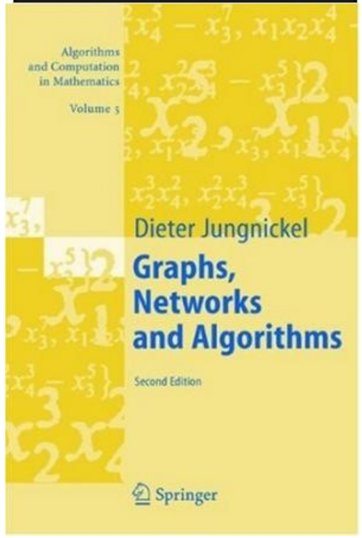
Biological sequence analysis

Probabilistic models of proteins and nucleic acids

R. Durbin
S. Eddy
A. Krogh
G. Mitchison

CAMBRIDGE

Copyrighted Material


Algorithms and Computation in Mathematics
Volume 5

Dieter Jungnickel

Graphs, Networks and Algorithms

Second Edition

Springer