CS6220: DATA MINING TECHNIQUES

Sequence Data

Instructor: Yizhou Sun

yzsun@ccs.neu.edu

November 16, 2014

Methods to Learn

	Matrix Data	Set Data	Sequence Data	Time Series	Graph & Network
Classification	Decision Tree; Naïve Bayes; Logistic Regression SVM; kNN		**HMM**		Label Propagation
Clustering	K-means; hierarchical clustering; DBSCAN; Mixture Models; kernel k-means				SCAN; Spectral Clustering
Frequent Pattern Mining		Apriori; FP-growth	**GSP; PrefixSpan**		
Prediction	Linear Regression			Autoregression	
Similarity Search				DTW	P-PageRank
Ranking					PageRank

Sequence Data

- What is sequence data?
- Sequential pattern mining
- Hidden Markov Model
- Summary

Sequence Database

- A sequence database consists of sequences of ordered elements or events, recorded with or without a concrete notion of time.

SID	sequence
10	<a(abc)(ac)d(cf)>
20	<(ad)c(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

Sequence Data

- What is sequence data?
- Sequential pattern mining
- Hidden Markov Model
- Summary

Sequence Databases & Sequential Patterns

- Transaction databases vs. sequence databases
- Frequent patterns vs. (frequent) sequential patterns
- Applications of sequential pattern mining
  - Customer shopping sequences:
    - First buy computer, then CD-ROM, and then digital camera, within 3 months.
  - Medical treatments, natural disasters (e.g., earthquakes), science & eng. processes, stocks and markets, etc.
  - Telephone calling patterns, Weblog click streams
  - Program execution sequence data sets
  - DNA sequences and gene structures

What Is Sequential Pattern Mining?

- Given a set of sequences, find the complete set of *frequent* subsequences

A *sequence* : < (ef) (ab) (df) c b >

An element may contain a set of items. Items within an element are unordered and we list them alphabetically.

A *sequence database*

SID	sequence
10	<a(abc)(ac)d(cf)>
20	<(ad)c(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

<a(bc)dc> is a *subsequence* of <a(abc)(ac)d(cf)>

Given *support threshold* *min_sup* =2, <(ab)c> is a *sequential pattern*

Sequence

- Event / element
  - An non-empty set of items, e.g., e=(ab)
- Sequence
  - An ordered list of events, e.g., $s = < e_1 e_2 \ldots e_l >$
- Length of a sequence
  - The number of instances of items in a sequence
  - The length of < (ef) (ab) (df) c b > is 8 (Not 5!)

Subsequence

- Subsequence
  - For two sequences $\alpha =< a_1 a_2 \dots a_n >$ and $\beta =< b_1 b_2 \dots b_m >$, α is called a subsequence of β if there exists integers $1 \leq j_1 < j_2 < \cdots < j_n \leq m$, such that $a_1 \subseteq b_{j_1}, \dots, a_n \subseteq b_{j_n}$
- Supersequence
  - If α is a subsequence of β, β is a supersequence of α

<a(bc)dc> is a *subsequence* of
<a(abc)(ac)d(cf)>

Sequential Pattern

- Support of a sequence α
  - Number of sequences in the database that are supersequence of α
  - $Support_S(\alpha)$
- α is frequent if $Support_S(\alpha) \geq \min\_support$
- A frequent sequence is called sequential pattern
  - l-pattern if the length of the sequence is l

Example

A *sequence database*

SID	sequence
10	<a(abc)(ac)d(cf)>
20	<(ad)c(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

Given *support threshold* *min_sup* =2, <(ab)c> is a *sequential pattern*

Challenges on Sequential Pattern Mining

- A huge number of possible sequential patterns are hidden in databases
- A mining algorithm should
  - find the complete set of patterns, when possible, satisfying the minimum support (frequency) threshold
  - be highly efficient, scalable, involving only a small number of database scans
  - be able to incorporate various kinds of user-specific constraints

Sequential Pattern Mining Algorithms

- Concept introduction and an initial Apriori-like algorithm
  - Agrawal & Srikant. Mining sequential patterns, ICDE'95
- Apriori-based method: GSP (Generalized Sequential Patterns: Srikant & Agrawal @ EDBT'96)
- Pattern-growth methods: FreeSpan & PrefixSpan (Han et al.@KDD'00; Pei, et al.@ICDE'01)
- Vertical format-based mining: SPADE (Zaki@Machine Leanining'00)
- Constraint-based sequential pattern mining (SPIRIT: Garofalakis, Rastogi, Shim@VLDB'99; Pei, Han, Wang @ CIKM'02)
- Mining closed sequential patterns: CloSpan (Yan, Han & Afshar @SDM'03)

The Apriori Property of Sequential Patterns

- A basic property: Apriori (Agrawal & Sirkant'94)
  - If a sequence S is not frequent
  - Then none of the super-sequences of S is frequent
  - E.g, <hb> is infrequent → so do <hab> and <(ah)b>

Seq. ID	Sequence
10	<(bd)cb(ac)>
20	<(bf)(ce)b(fg)>
30	<(ah)(bf)abf>
40	<(be)(ce)d>
50	<a(bd)bcb(ade)>

Given *support threshold* *min_sup* =2

GSP—Generalized Sequential Pattern Mining

- GSP (Generalized Sequential Pattern) mining algorithm
  - proposed by Agrawal and Srikant, EDBT'96
- Outline of the method
  - Initially, every item in DB is a candidate of length-1
  - for each level (i.e., sequences of length-k) do
    - scan database to collect support count for each candidate sequence
    - generate candidate length-(k+1) sequences from length-k frequent sequences using Apriori
  - repeat until no frequent sequence or no candidate can be found
- Major strength: Candidate pruning by Apriori

Finding Length-1 Sequential Patterns

- Examine GSP using an example
- Initial candidates: all singleton sequences
  - <a>, <b>, <c>, <d>, <e>, <f>, <g>, <h>
- Scan database once, count support for candidates

min_sup =2

Seq. ID	Sequence
10	<(bd)cb(ac)>
20	<(bf)(ce)b(fg)>
30	<(ah)(bf)abf>
40	<(be)(ce)d>
50	<a(bd)bcb(ade)>

Cand	Sup
<a>	**3**
<b>	**5**
<c>	**4**
<d>	**3**
<e>	**3**
<f>	**2**
~~<g>~~	1
~~<h>~~	1

GSP: Generating Length-2 Candidates

51 length-2 Candidates

	<a>	<b>	<c>	<d>	<e>	<f>
<a>	<aa>	<ab>	<ac>	<ad>	<ae>	<af>
<b>	<ba>	<bb>	<bc>	<bd>	<be>	<bf>
<c>	<ca>	<cb>	<cc>	<cd>	<ce>	<cf>
<d>	<da>	<db>	<dc>	<dd>	<de>	<df>
<e>	<ea>	<eb>	<ec>	<ed>	<ee>	<ef>
<f>	<fa>	<fb>	<fc>	<fd>	<fe>	<ff>

	<a>	<b>	<c>	<d>	<e>	<f>
<a>		<(ab)>	<(ac)>	<(ad)>	<(ae)>	<(af)>
<b>			<(bc)>	<(bd)>	<(be)>	<(bf)>
<c>				<(cd)>	<(ce)>	<(cf)>
<d>					<(de)>	<(df)>
<e>						<(ef)>
<f>						

Without Apriori property, 8*8+8*7/2=92 candidates

Apriori prunes 44.57% candidates

How to Generate Candidates in General?

- From L_{k-1} to C_k
- Step 1: join
  - s_1 *and* s_2 can join, if dropping first item in s_1 is the same as dropping the last item in s_2
  - Examples:
    - <(12)3> join <(2)34> = <(12)34>
    - <(12)3> join <(2)(34)> = <(12)(34)>
- Step 2: pruning
  - Check whether all length k-1 subsequences of a candidate is contained in L_{k-1}

The GSP Mining Process

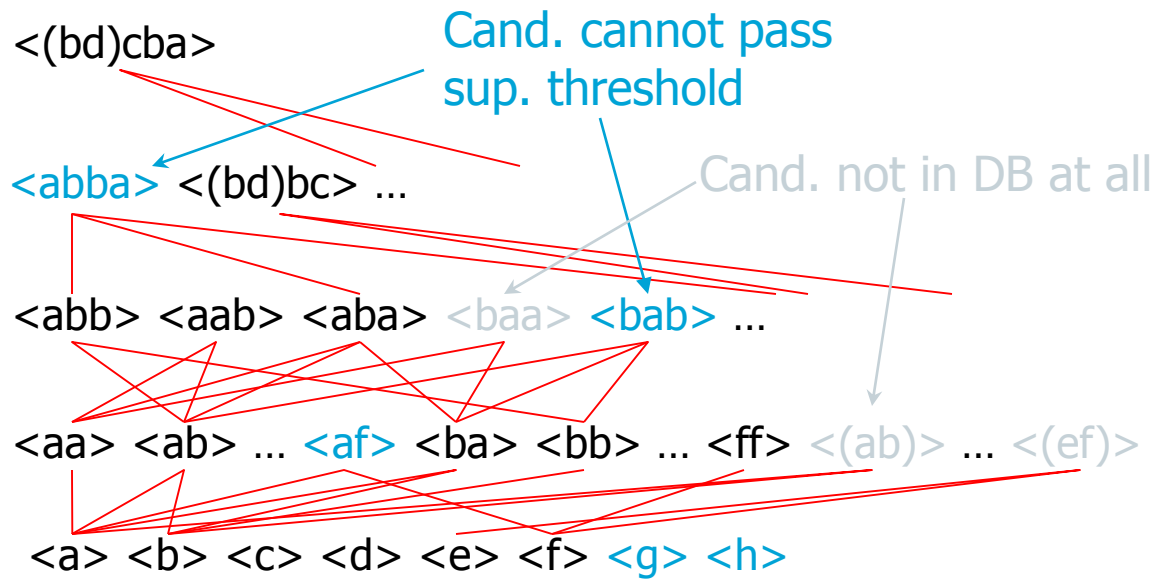
5th scan: 1 cand. 1 length-5 seq. pat.

4th scan: 8 cand. 7 length-4 seq. pat.

3rd scan: 46 cand. 20 length-3 seq. pat. 20 cand. not in DB at all

2nd scan: 51 cand. 19 length-2 seq. pat. 10 cand. not in DB at all

1st scan: 8 cand. 6 length-1 seq. pat.



min_sup =2

Seq. ID	Sequence
10	<(bd)cb(ac)>
20	<(bf)(ce)b(fg)>
30	<(ah)(bf)abf>
40	<(be)(ce)d>
50	<a(bd)bcb(ade)>

Candidate Generate-and-test: Drawbacks

- A huge set of candidate sequences generated.
  - Especially 2-item candidate sequence.
- Multiple Scans of database needed.
  - The length of each candidate grows by one at each database scan.
- Inefficient for mining long sequential patterns.
  - A long pattern grow up from short patterns
  - The number of short patterns is exponential to the length of mined patterns.

*The SPADE Algorithm

- SPADE (Sequential PAttern Discovery using Equivalent Class) developed by Zaki 2001
- A vertical format sequential pattern mining method
- A sequence database is mapped to a large set of
  - Item: <SID, EID>
- Sequential pattern mining is performed by
  - growing the subsequences (patterns) one item at a time by Apriori candidate generation

The SPADE Algorithm

SID	EID	Items
1	1	a
1	2	abc
1	3	ac
1	4	d
1	5	cf
2	1	ad
2	2	c
2	3	bc
2	4	ae
3	1	ef
3	2	ab
3	3	df
3	4	c
3	5	b
4	1	e
4	2	g
4	3	af
4	4	c
4	5	b
4	6	c

a		b		...
SID	EID	SID	EID	...
1	1	1	2	
1	2	2	3	
1	3	3	2	
2	1	3	5	
2	4	4	5	
3	2			
4	3			

Join two tables

ab			ba			...
SID	EID (a)	EID(b)	SID	EID (b)	EID(a)	...
1	1	2	1	2	3	
2	1	3	2	3	4	
3	2	5				
4	3	5				

aba			...
SID	EID (a)	EID(b)	EID(a)	...
1	1	2	3	
2	1	3	4	

Bottlenecks of GSP and SPADE

- A huge set of candidates could be generated
  - 1,000 frequent length-1 sequences generate s huge number of length-2 candidates!

$$1000 \times 1000 + \frac{1000 \times 999}{2} = 1{,}499{,}500$$

- Multiple scans of database in mining
- Breadth-first search
- Mining long sequential patterns
  - Needs an exponential number of short candidates
  - A length-100 sequential pattern needs 10^{30} candidate sequences!

$$\sum_{i=1}^{100} \binom{100}{i} = 2^{100} - 1 \approx 10^{30}$$

Prefix and Suffix (Projection)

Assume a pre-specified order on items, e.g., alphabetical order

- <a>, <aa>, <a(ab)> and <a(abc)> are *prefixes* of sequence <a(abc)(ac)d(cf)>
  - Note <a(ac)> is not a prefix of <a(abc)(ac)d(cf)>
- Given sequence <a(abc)(ac)d(cf)>

Prefix	*Suffix* (Prefix-Based *Projection)*
<a>	<(abc)(ac)d(cf)>
<aa>	<(_bc)(ac)d(cf)>
<a(ab)>	<(_c)(ac)d(cf)>

Mining Sequential Patterns by Prefix Projections

- Step 1: find length-1 sequential patterns
  - <a>, <b>, <c>, <d>, <e>, <f>
- Step 2: divide search space. The complete set of seq. pat. can be partitioned into 6 subsets:
  - The ones having prefix <a>;
  - The ones having prefix <b>;
  - …
  - The ones having prefix <f>

SID	sequence
10	<a(abc)(ac)d(cf)>
20	<(ad)c(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

Finding Seq. Patterns with Prefix <a>

- Only need to consider projections w.r.t. <a>
  - <a>-projected database:
    - <(abc)(ac)d(cf)>
    - <(_d)c(bc)(ae)>
    - <(_b)(df)cb>
    - <(_f)cbc>

SID	sequence
10	<a(abc)(ac)d(cf)>
20	<(ad)c(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

- Find all the length-2 seq. pat. Having prefix <a>: <aa>, <ab>, <(ab)>, <ac>, <ad>, <af>
  - Further partition into 6 subsets
    - Having prefix <aa>;
    - …
    - Having prefix <af>

Why are those 6 subsets?

- By scanning the <*a*>-projected database once, its locally frequent items are identified as
  - $a : 2,\ b : 4,\ \_b : 2,\ c : 4,\ d : 2,$ and $f : 2.$
- Thus all the length-2 sequential patterns prefixed with <*a*> are found, and they are:
  - $\langle aa \rangle : 2,\ \langle ab \rangle : 4,\ \langle (ab) \rangle : 2,\ \langle ac \rangle : 4,\ \langle ad \rangle : 2,$ and $\langle a\ f \rangle : 2.$

Completeness of PrefixSpan

SDB

SID	sequence
10	<a(abc)(ac)d(cf)>
20	<(ad)c(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

Length-1 sequential patterns
<a>, <b>, <c>, <d>, <e>, <f>

Having prefix <a>

Having prefix <b>

Having prefix <c>, …, <f>

<a>-projected database
<(abc)(ac)d(cf)>
<(_d)c(bc)(ae)>
<(_b)(df)cb>
<(_f)cbc>

Length-2 sequential patterns
<aa>, <ab>, <(ab)>, <ac>, <ad>, <af>

<b>-projected database

…

… …

Having prefix <aa>

Having prefix <af>

<aa>-proj. db

…

<af>-proj. db

Efficiency of PrefixSpan

- No candidate sequence needs to be generated
- Projected databases keep shrinking
- Major cost of PrefixSpan: Constructing projected databases
  - Can be improved by pseudo-projections

Speed-up by Pseudo-projection

- Major cost of PrefixSpan: projection
  - Postfixes of sequences often appear repeatedly in recursive projected databases
- When (projected) database can be held in main memory, use pointers to form projections
  - Pointer to the sequence
  - Offset of the postfix

s=<a(abc)(ac)d(cf)>

↓ <a>

s|<a>: (, 2) <(abc)(ac)d(cf)>

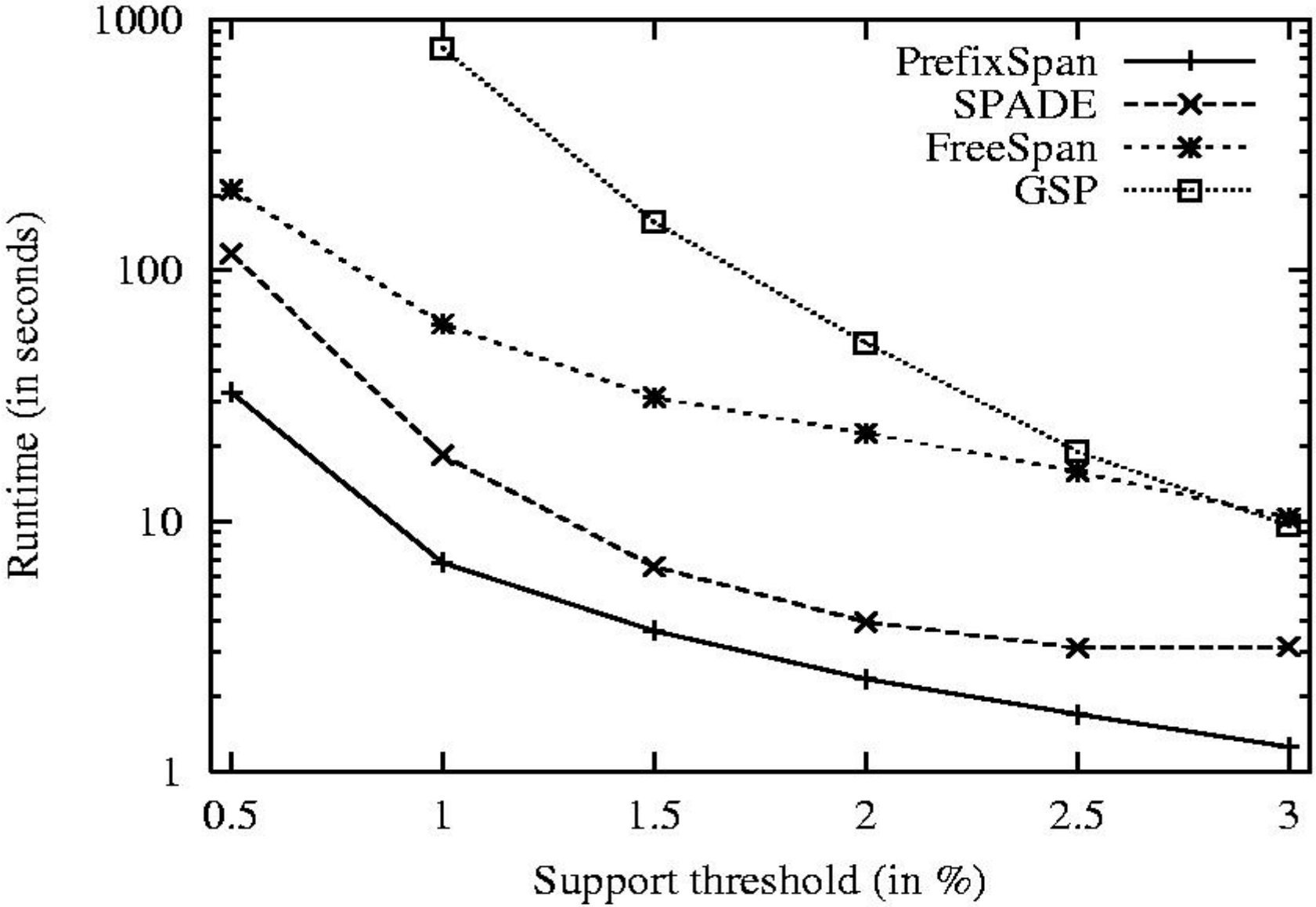
↓ <ab>

s|<ab>: (, 4) <(_c)(ac)d(cf)>

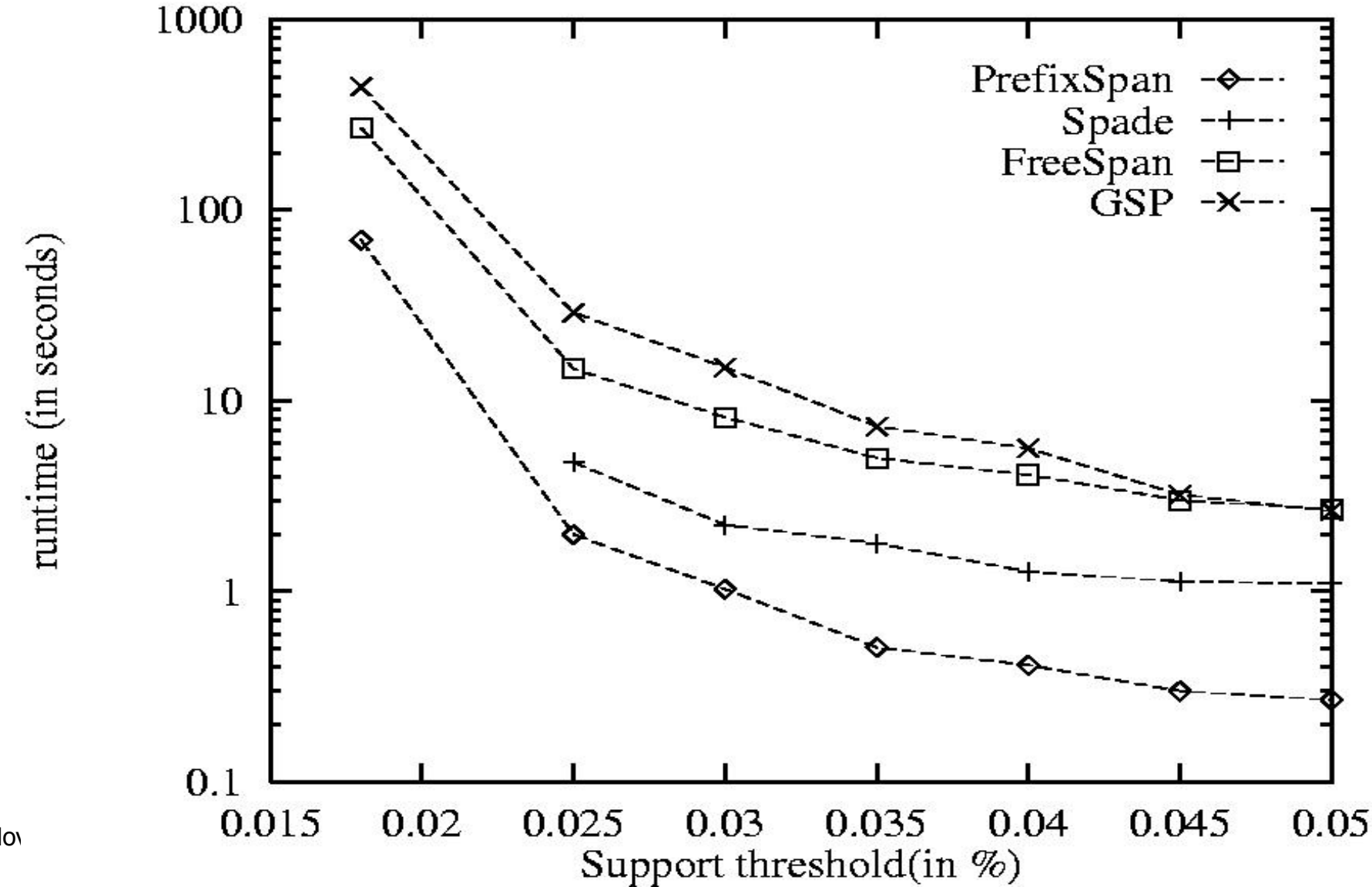
Pseudo-Projection vs. Physical Projection

- Pseudo-projection avoids physically copying postfixes
  - Efficient in running time and space when database can be held in main memory
- However, it is not efficient when database cannot fit in main memory
  - Disk-based random accessing is very costly
- Suggested Approach:
  - Integration of physical and pseudo-projection
  - Swapping to pseudo-projection when the data set fits in memory

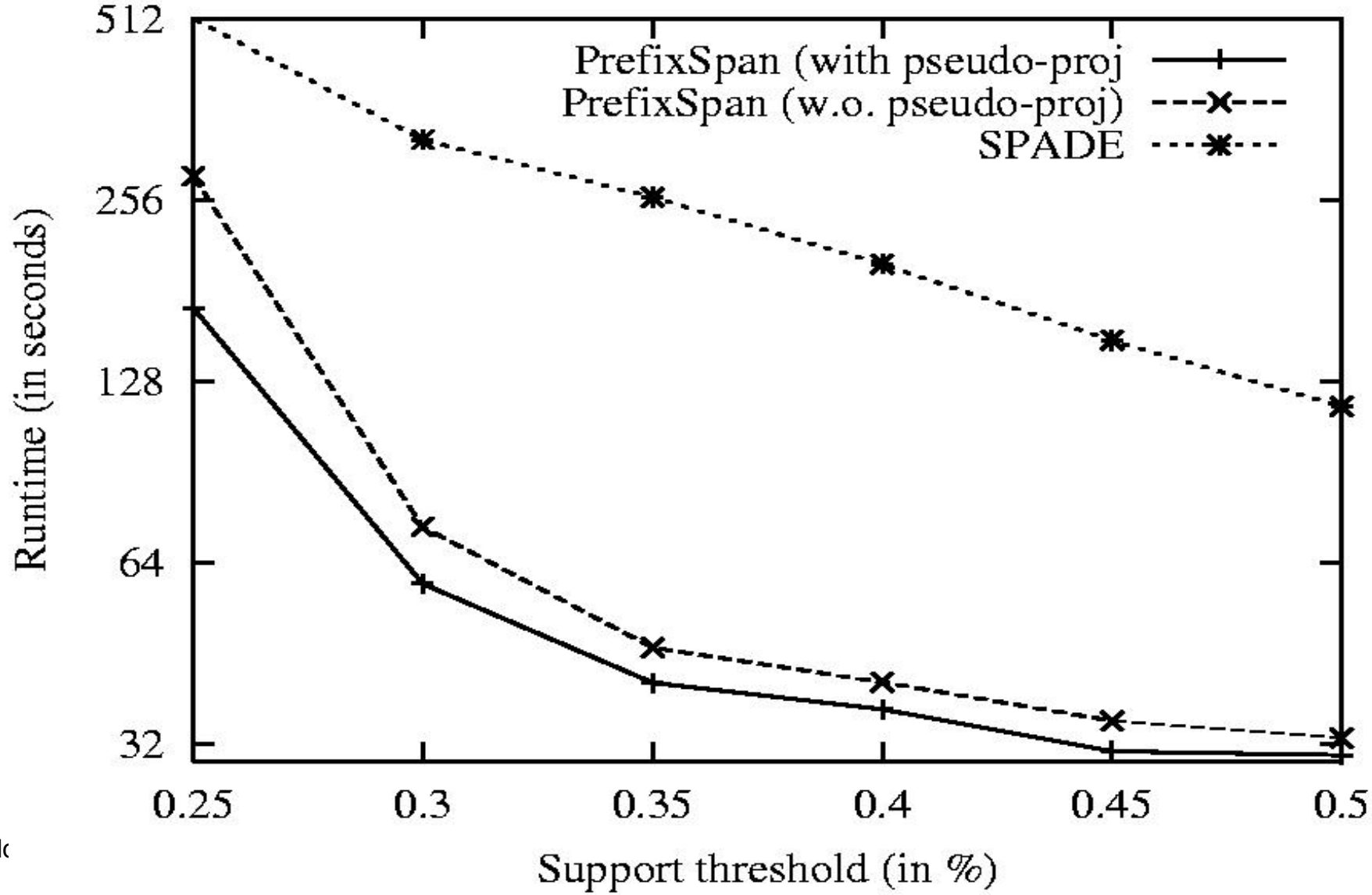
Performance on Data Set C10T8S8I8



Performance on Data Set Gazelle



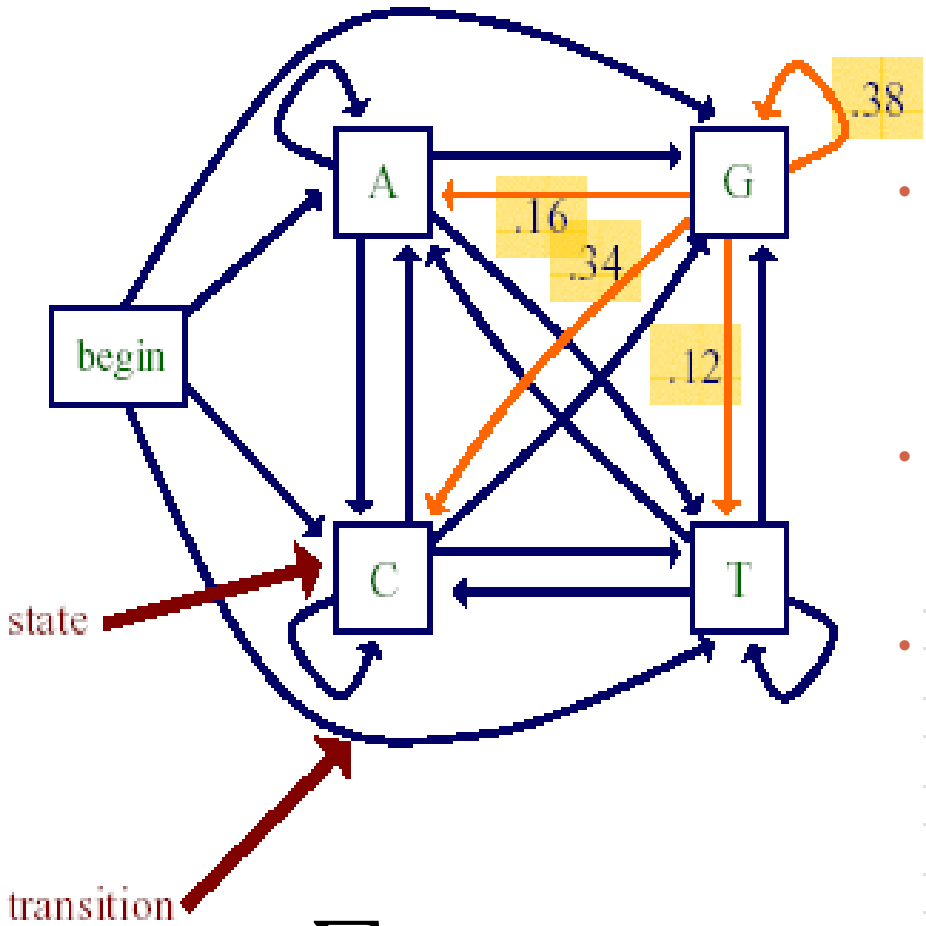
Effect of Pseudo-Projection



Sequence Data

- What is sequence data?
- Sequential pattern mining
- Hidden Markov Model
- Summary

A Markov Chain Model



- **Markov property: *Given the present state*, future states are independent of the past states**
- At each step the system may change its state from the current state to another state, or remain in the same state, according to a certain probability distribution
- The changes of state are called ***transitions***, and the probabilities associated with various state-changes are called ***transition probabilities***
- Transition probabilities
  - $Pr(x_i=a|x_{i-1}=g)=0.16$
  - $Pr(x_i=c|x_{i-1}=g)=0.34$
  - $Pr(x_i=g|x_{i-1}=g)=0.38$
  - $Pr(x_i=t|x_{i-1}=g)=0.12$

$$\sum \Pr(x_i \mid x_{i-1} = g) = 1$$

Definition of Markov Chain Model

- A Markov chain model is defined by

Each event of a sequence here is Considered containing only one state

  - A set of states
    - Some states emit symbols
    - Other states (e.g., the begin state) are silent
  - A set of transitions with associated probabilities
    - The transitions emanating from a given state define a distribution over the possible next states

Markov Chain Models: Properties

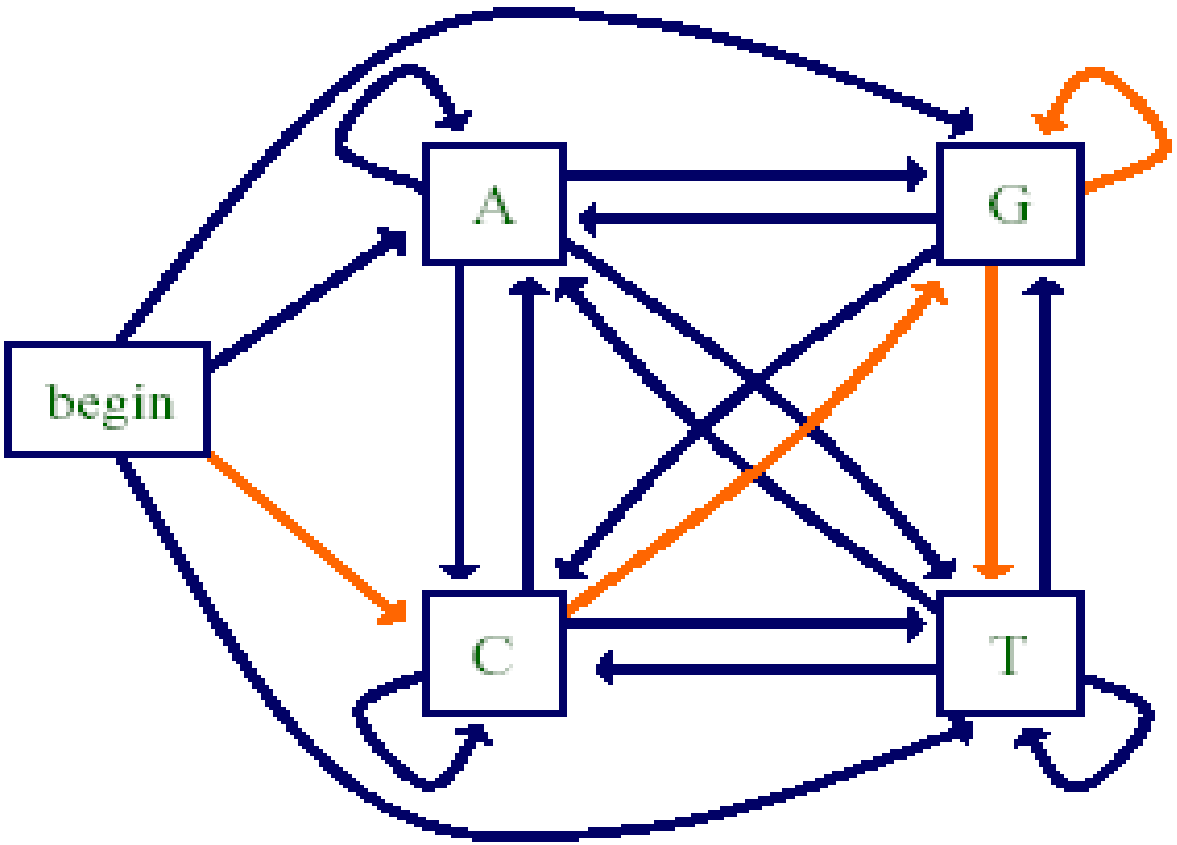
- Given some sequence x of length L, we can ask how probable the sequence is given our model
- For any probabilistic model of sequences, we can write this probability as

$$\Pr(x) = \Pr(x_L, x_{L-1}, \ldots, x_1)$$
$$= \Pr(x_L \mid x_{L-1}, \ldots, x_1) \Pr(x_{L-1} \mid x_{L-2}, \ldots, x_1) \ldots \Pr(x_1)$$

- key property of a (1st order) Markov chain: the probability of each x_i depends only on the value of x_{i-1}

$$\Pr(x) = \Pr(x_L \mid x_{L-1}) \Pr(x_{L-1} \mid x_{L-2}) \ldots \Pr(x_2 \mid x_1) \Pr(x_1)$$
$$= \Pr(x_1) \prod_{i=2}^{L} \Pr(x_i \mid x_{i-1})$$

The Probability of a Sequence for a Markov Chain Model



$$Pr(cggt)=Pr(c)Pr(g|c)Pr(g|g)Pr(t|g)$$

*Example Application

- CpG islands
  - CG dinucleotides are rarer in eukaryotic genomes than expected given the marginal probabilities of C and G
  - but the regions upstream of genes are richer in CG dinucleotides than elsewhere – CpG islands
  - useful evidence for finding genes
- Application: Predict CpG islands with Markov chains
  - one to represent CpG islands
  - one to represent the rest of the genome

*Markov Chains for Discrimination

- Suppose we want to distinguish CpG islands from other sequence regions
- Given sequences from CpG islands, and sequences from other regions, we can construct
  - a model to represent CpG islands
  - a null model to represent the other regions
- can then score a test sequence by:

$$score(x) = \log \frac{\Pr(x \mid CpGModel)}{\Pr(x \mid nullModel)}$$

*Markov Chains for Discrimination

- Why use

$$score(x) = \log \frac{\Pr(x \mid CpGModel)}{\Pr(x \mid nullModel)}$$

- According to Bayes' rule

$$\Pr(CpG \mid x) = \frac{\Pr(x \mid CpG)\Pr(CpG)}{\Pr(x)}$$

$$\Pr(null \mid x) = \frac{\Pr(x \mid null)\Pr(null)}{\Pr(x)}$$

- If we are not taking into account of prior probabilities of two classes, we just need to compare Pr(x|CpG) and Pr(x|null)

*Higher Order Markov Chains

- The Markov property specifies that the probability of a state depends only on the probability of the previous state
- But we can build more "memory" into our states by using a higher order Markov model
- In an n-th order Markov model

$$\Pr(x_i \mid x_{i-1}, x_{i-2}, \ldots, x_1) = \Pr(x_i \mid x_{i-1}, \ldots, x_{i-n})$$

*Higher Order Markov Chains

- An n-th order Markov chain over some alphabet A is equivalent to a first order Markov chain over the alphabet of n-tuples: A^n
- Example: A 2nd order Markov model for DNA can be treated as a 1st order Markov model over alphabet

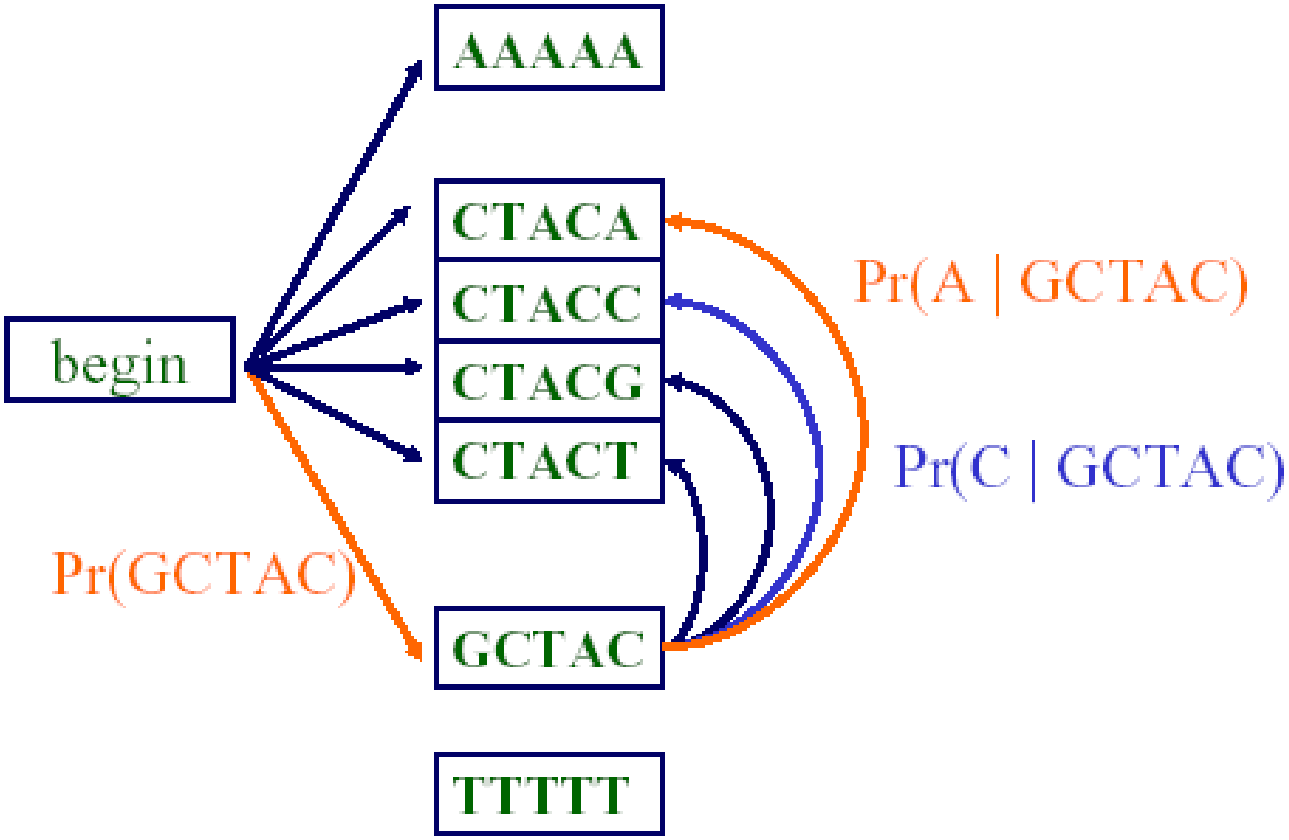
  AA, AC, AG, AT

  CA, CC, CG, CT

  GA, GC, GG, GT

  TA, TC, TG, TT

*A Fifth Order Markov Chain



Pr(gctaca)=Pr(gctac)Pr(a|gctac)

Hidden Markov Model

- A **hidden Markov model** (**HMM**): A statistical model in which the system being modeled is assumed to be a Markov process with *unknown parameters*
  - The state is not directly visible, but variables influenced by the state are visible
  - Each state has a probability distribution over the possible output tokens. Therefore the sequence of tokens generated by an HMM gives some information about the sequence of states.
- The challenge is to determine the hidden parameters from the observable data. The extracted model parameters can then be used to perform further analysis
- An HMM can be considered as the simplest dynamic Bayesian network

Learning and Prediction Tasks

- Learning
  - Given a model, a set of training sequences
  - Find model parameters that explain the training sequences with relatively high probability (goal is to find a model that *generalizes* well to sequences we haven't seen before)
- Classification
  - Given a set of models representing different sequence classes, a test sequence
  - Determine which model/class best explains the sequence
- Segmentation
  - Given a model representing different sequence classes, a test sequence
  - Segment the sequence into subsequences, predicting the class of each subsequence

The Parameters of an HMM

- Transition Probabilities

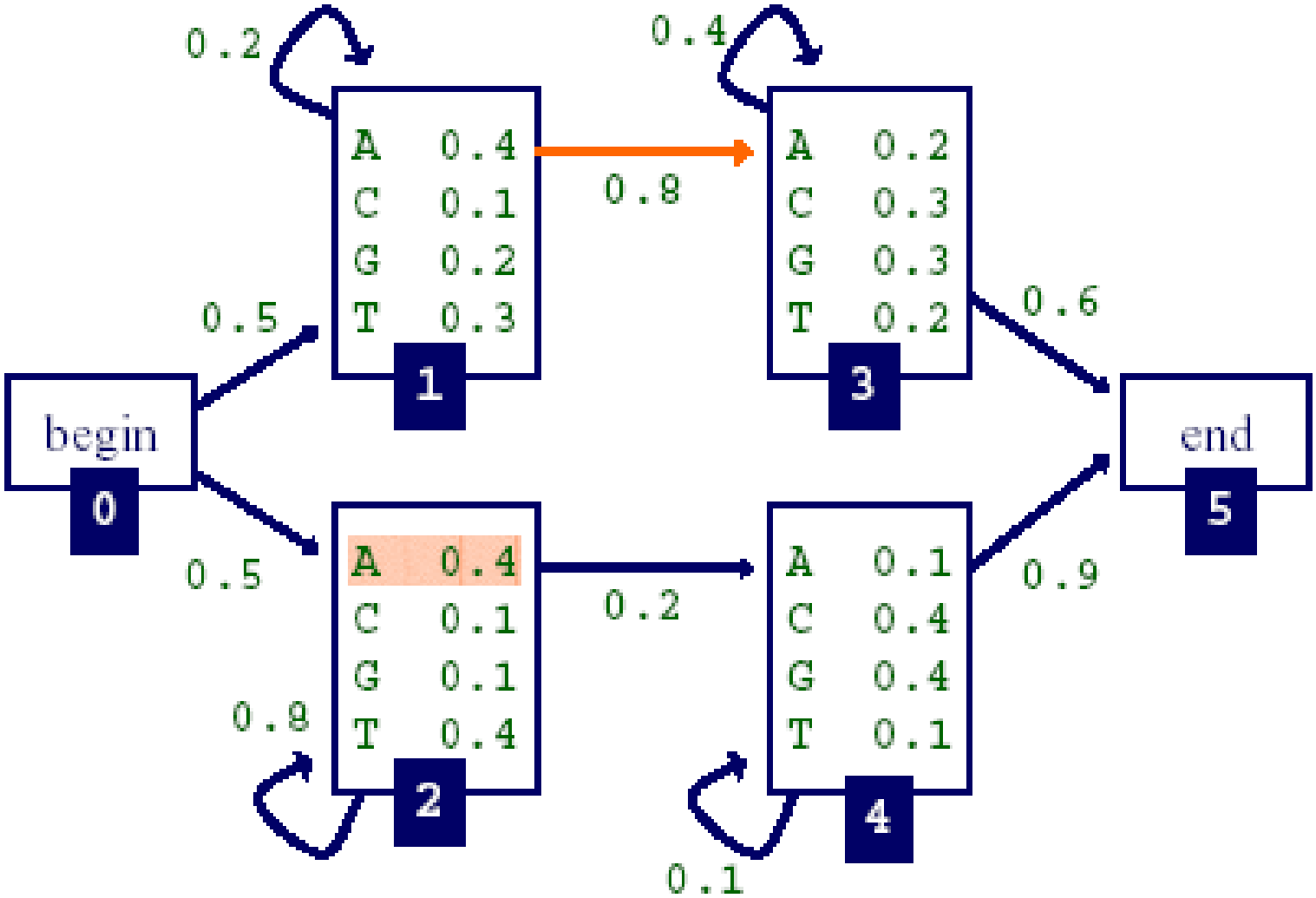
$$a_{kl} = \Pr(\pi_i = l \mid \pi_{i-1} = k)$$

  - Probability of transition from state k to state l
- Emission Probabilities

$$e_k(b) = \Pr(x_i = b \mid \pi_i = k)$$

  - Probability of emitting character b in state k

An HMM Example



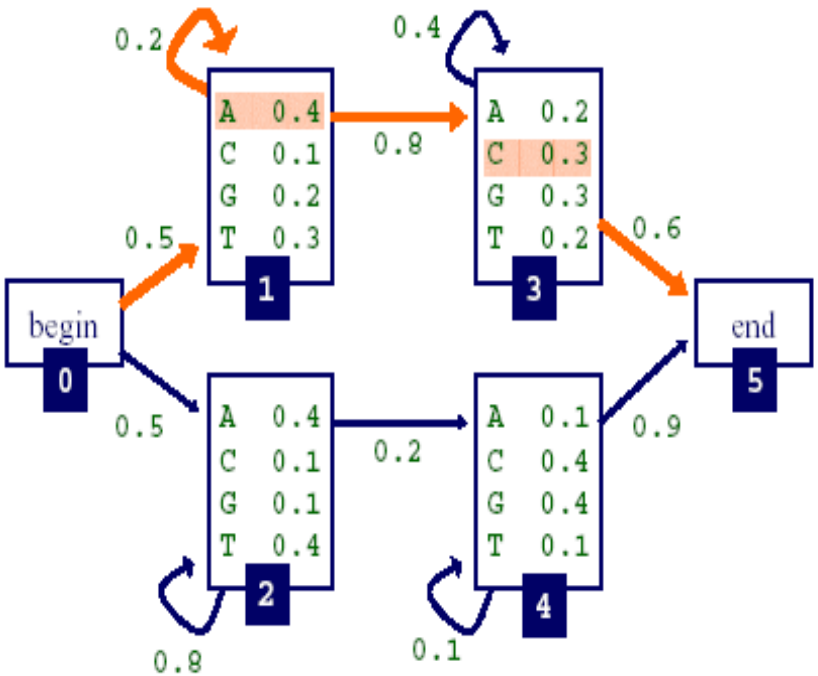
Three Important Questions

- How likely is a given sequence?
  - The Forward algorithm
- What is the most probable "path" for generating a given sequence?
  - The Viterbi algorithm
- How can we learn the HMM parameters given a set of sequences?
  - The Forward-Backward (Baum-Welch) algorithm

How Likely is a Given Sequence?

- The probability that the path is taken and the sequence is generated:

$$\Pr(x_1 \ldots x_L, \pi_0 \ldots \pi_N) = a_{0\pi_1} \prod_{i=1}^{L} e_{\pi_i}(x_i) a_{\pi_i \pi_{i+1}}$$



$$\Pr(AAC, \pi)$$
$$= a_{01} \times e_1(A) \times a_{11} \times e_1(A)$$
$$\times a_{13} \times e_3(C) \times a_{35}$$
$$= .5 \times .4 \times .2 \times .4 \times .8 \times .3 \times .6$$

How Likely is a Given Sequence?

- The probability over all paths is

$$\Pr(x_1 \ldots x_L) = \sum_{\pi} \Pr(x_1 \ldots x_L, \underbrace{\pi_0 \ldots \pi_N}_{\pi})$$

- But the number of paths can be exponential in the length of the sequence...
- The Forward algorithm enables us to compute this efficiently
  - Define $f_k(i)$ to be the probability of being in state *k* having observed the first *i* characters of sequence x
  - To compute *Pr(x)*, the probability of being in the end state having observed all of sequence x
  - Can define this recursively
  - use dynamic programming

The Forward Algorithm

- Initialization
  - $f_0(0) = 1$ for start state; $f_i(0) = 0$ for other state
- Recursion
  - For emitting state (i = 1, … L)

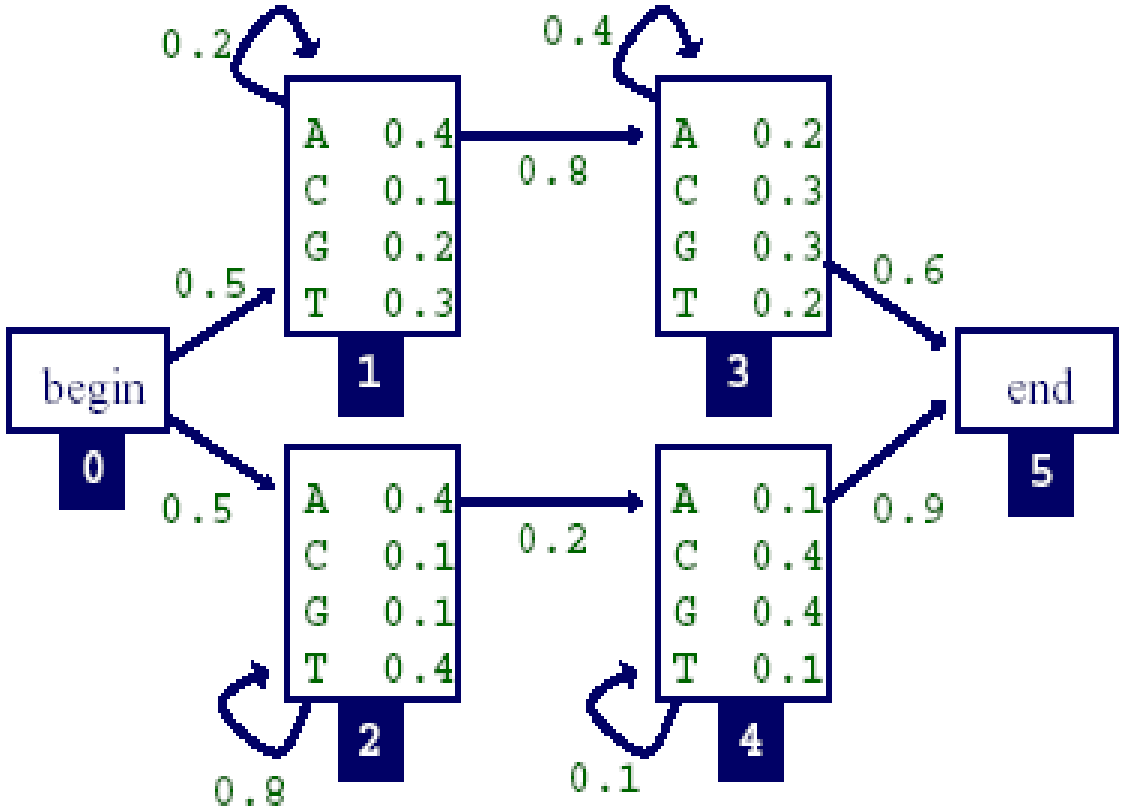
$$f_l(i) = e_l(x_i)\sum_k f_k(i-1)a_{kl}$$

- Termination

$$\Pr(x) = \Pr(x_1 \ldots x_L) = \sum_k f_k(L)a_{kN}$$

N: ending state;
denoted as 0 in textbook

Forward Algorithm Example



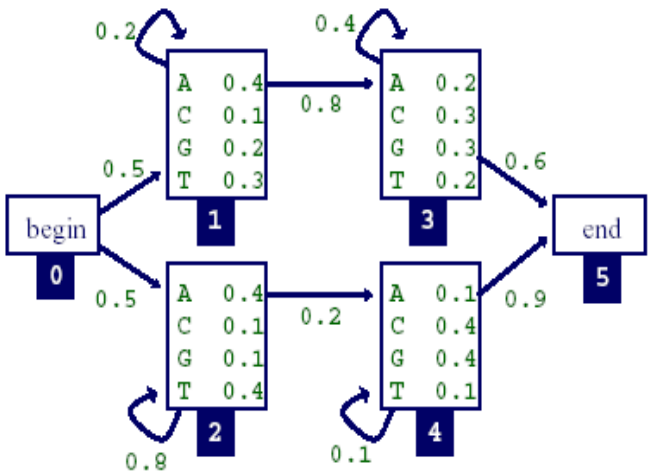
Given the sequence x=TAGA

Forward Algorithm Example

- Initialization
  - $f_0(0)=1,\ f_1(0)=0 \ldots f_5(0)=0$
- Computing other values
  - $f_1(1)=e_1(T)*(f_0(0)a_{01}+f_1(0)a_{11})$
    $=0.3*(1*0.5+0*0.2)=0.15$
  - $f_2(1)=0.4*(1*0.5+0*0.8)$
  - $f_1(2)=e_1(A)*(f_0(1)a_{01}+f_1(1)a_{11})$
    $=0.4*(0*0.5+0.15*0.2)$

…

  - $Pr(TAGA)= f_5(4)=f_3(4)a_{35}+f_4(4)a_{45}$



Three Important Questions

- How likely is a given sequence?
- What is the most probable "path" for generating a given sequence?
- How can we learn the HMM parameters given a set of sequences?

Find the most probable "path" for generating a given sequence

- Decoding
  - $\pi^* = \mathrm{argmax}_\pi \mathrm{P}(\pi|x)$
- Given a length L sequence, how many possible underlying paths?
  - $|\mathrm{Q}|$^L
  - Where $|\mathrm{Q}|$ is the number of possible state

Finding the Most Probable Path: The Viterbi Algorithm

- Define $v_k(i)$ to be the probability of the most probable path accounting for the first i characters of x and ending in state k
- We want to compute $v_N(L)$, the probability of the most probable path accounting for all of the sequence and ending in the end state
- Can define recursively
- Can use DP to find $v_N(L)$ efficiently

Algorithm

Method:

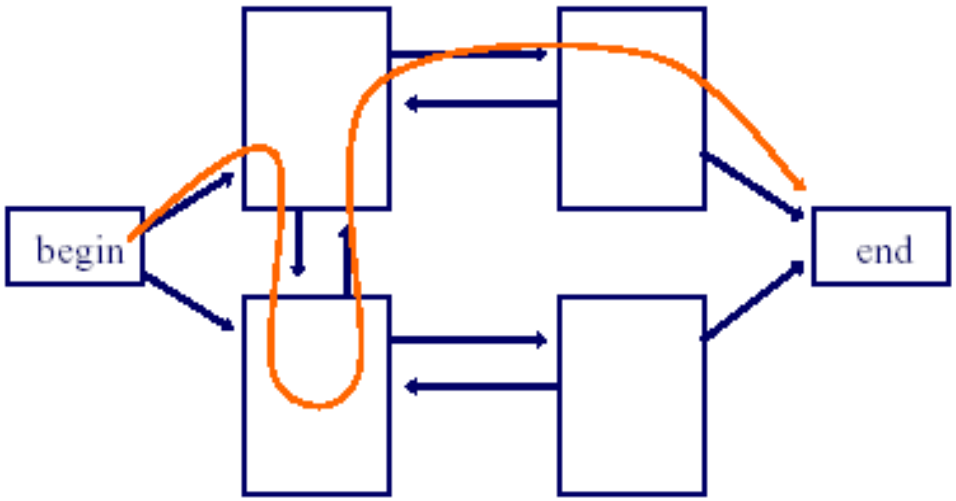
(1)	Initialization ($i = 0$):	$v_0(0) = 1, v_k(0) = 0$ for $k > 0$
(2)	Recursion ($i = 1 \ldots L$):	$v_l(i) = e_l(x_i) max_k(v_k(i-1)a_{kl})$
		$ptr_i(l) = argmax_k(v_k(i-1)a_{kl})$
(3)	Termination:	$P(x, \pi^*) = max_k(v_k(L)a_{k0})$
		$\pi^*_L = argmax_k(v_k(L)a_{k0})$

Three Important Questions

- How likely is a given sequence?
- What is the most probable "path" for generating a given sequence?
- How can we learn the HMM parameters given a set of sequences?

Learning Without Hidden State

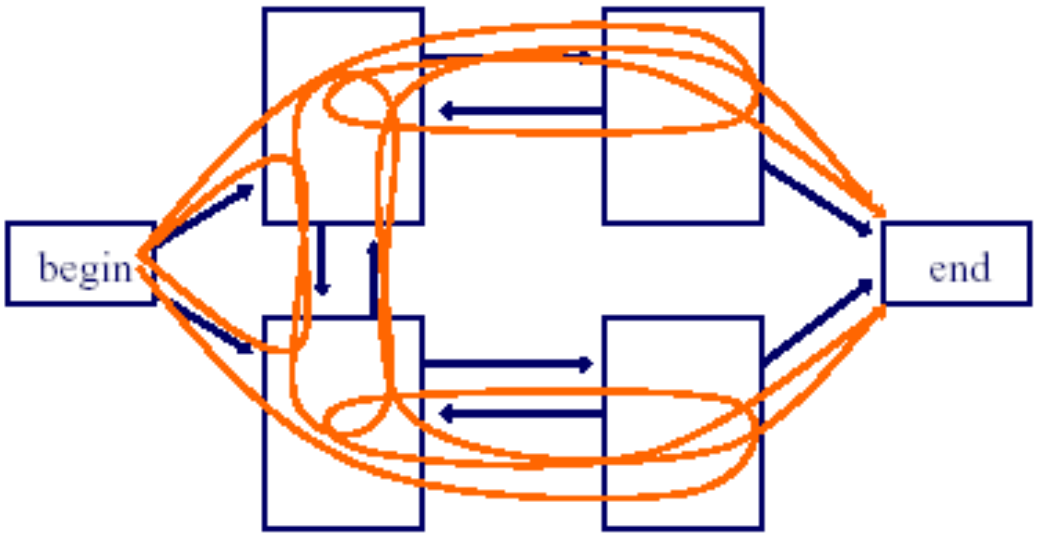
- Learning is simple if we know the correct path for each sequence in our training set



- estimate parameters by counting the number of times each parameter is used across the training set

Learning With Hidden State

- If we don't know the correct path for each sequence in our training set, consider all possible paths for the sequence



- Estimate parameters through a procedure that counts the expected number of times each parameter is used across the training set

*Learning Parameters: The Baum-Welch Algorithm

- Also known as the Forward-Backward algorithm
- An Expectation Maximization (EM) algorithm
  - EM is a family of algorithms for learning probabilistic models in problems that involve hidden state
- In this context, the hidden state is the path that best explains each training sequence

*Learning Parameters: The Baum-Welch Algorithm

- Algorithm sketch:
  - initialize parameters of model
  - iterate until convergence
    - calculate the *expected* number of times each transition or emission is used
    - adjust the parameters to *maximize* the likelihood of these expected values

Computational Complexity of HMM Algorithms

- Given an HMM with S states and a sequence of length L, the complexity of the Forward, Backward and Viterbi algorithms is $O(S^2 L)$
  - This assumes that the states are densely interconnected
- Given M sequences of length L, the complexity of Baum Welch on each iteration is $O(MS^2 L)$

Sequence Data

- What is sequence data?
- Sequential pattern mining
- Hidden Markov Model
- Summary

Summary

- Sequential Pattern Mining
  - GSP, SPADE, PrefixSpan
- Hidden Markov Model
  - Markov chain, HMM