



Motif Discovery

• •

Monte Anderson, Byron Bearden, Sushma Colanukudhuru, Rasmi Lamichhane, Han Ngo, Christine Samson
CSCI 4314 - Algorithms for Molecular Biology
Dr. David Knox

What problem are we tackling?

- Motif - A nucleotide/amino acid pattern that occurs in sequences
- Importance: Biological Significance
- How can we detect motifs and their locations?
- Did we find the motifs?
 - Were all expected motifs found?
 - Were any motifs missed?
- Goal: Improve upon the current motif sequence finding tools

AGCTACTCCAAC TACTGGATT TACTCAGTAACTTCCGTACTAGACACCGCTATACTGGCC

Dataset Generation

Generating a Dataset

```
GenerateData.py  Settings

1 import sys, math, operator
2 from random import randint
3 from collections import Counter
4 import re
5
6 def main(argv):
7     print("\nGenerating Random Data...")
8
9     generatedString = ""
10    motif = ["AACGCTTGCACCTTATTGCA", "ACCGCGCTGATTTGCGAA"]
11    #motif = ["---", "___"]
12
13    indexList_1 = []
14    indexList_2 = []
15    index = 0
16
17    finalString = ""
18    print("\nLooking for motifs: " + str(motif) + "\n")
19    addedLocations = []
20    currentLine = ""
21    count = 0
22
23
24    while len(generatedString) <= 1000000:
25
26        currentRandom = randint(0,12)
27        if currentRandom == 0 or currentRandom == 1 or currentRandom == 2:
28            generatedString += "A"
29            index+=1
30
31        if currentRandom == 3 or currentRandom == 4 or currentRandom == 5:
32            generatedString += "C"
33            index+=1
```

- Generate_Data.py
 - Generates random sequences
 - Expected motifs will be inserted at various positions
- Program Input
 - Random motifs desired to be found
- Program Output
 - The number of expected motifs found
 - Sequences in FASTA format
- Generated small, medium, and large files containing randomly generated sequences.

Validating Dataset Generation

- We kept track of all the motif insertion points and how many were inserted. This allows us to verify how accurate the algorithms are.
- Why random data?
 - It allows us to test for edge cases which we may not be aware of in existing DNA. This allows us to test how accurate MEME is under different circumstances, and when it begins to fail. We used a lot of cases for benchmarking MEME, to see how it performed with various parameters (like motif length, motif frequency, and overlap probability).

```
Generating Random Data...
```

```
Looking for motifs: ['AACGCTTG', 'ACCGCGCC']
```

```
We should have around 99 motifs ]
```

```
GCGAACC GCGCCAGTATATGCCGGTAAACCGCGCCTGTCCAGGTGAACGCTTGATAGCAGCATCCTTCCTGACCGCGCCCTACGTACACCGCGCCCAACTATCACGCTGATACCC
```

```
Strand 1: GCGAACC GCGCCAGTATATGCCGGTAAACCGCGCCTGTCCAGGTGAACGCTTGATAGCAGCATCCTTCCTGACCGCGCCCTACGTACACCGCGCCCAACTATCAC
```

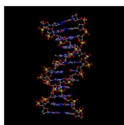
```
( 'Motif: AACGCTTG', [['47'], ['167']] )
```

```
( 'Motif: ACCGCGCC', [['5'], ['29'], ['73'], ['89'], ['127']] )
```

Benchmarking

Proposed (Initial) Benchmarking Tool

- BioProspector
 - Tool to find DNA motif sequences
 - C program



Overview

BioProspector finds enriched sequence motifs

Motif Finding

Search for interesting motifs in your sequences on our server

Input Format

How to specify the input parameters

Output Explanations

How to understand the output email we send you

Reference

Proc Pac Symp Biocomput
2001:::127-38

Contacts

People behind the project

SeqMotifs

See other motif finding algorithms we have developed.

BioProspector

Discovering Conserved DNA Motifs in Upstream Regulatory Regions of Co-Expressed Genes

[Xiaole Liu](#), [Jun S. Liu](#), [Douglas L. Brutlag](#)
[Stanford Medical Informatics](#), [Stanford University](#)

The development of high throughput genome sequencing and gene expression techniques gives rise to the demand for data-mining tools. BioProspector, a C program using a Gibbs sampling strategy, examines the upstream region of genes in the same gene expression pattern group and looks for regulatory sequence motifs. BioProspector uses Markov background to model the base dependencies of non-motif bases, which greatly improved the specificity of the reported motifs. The parameters of the Markov background model are either estimated from user-specified sequences or pre-computed from the whole genome sequences. A new motif scoring function is adopted to allow each input sequences to contain zero to multiple copies of the motif. In addition, BioProspector can model gapped motifs and motifs with palindromic patterns, which are prevalent motif patterns in prokaryotes. All these modifications greatly improve the performance of the program. Besides showing preliminary success in finding the binding motifs for *S. cerevisiae* RAP1, *B. subtilis* RNA polymerase, and *E. coli* CRP, we have used BioProspector to find s54 motif from *M. xanthus* genome, many *B. subtilis* motifs from [DBTBS](#) collection of promoters, and motifs from [yeast expression data](#).

BioProspector requires the user to specify a motif width. Recently, JS Liu and his student have developed an algorithm [BioOptimizer](#) to automatically adjust a user-specified motif width to optimize the motif's information. The program can be downloaded from: <http://www.people.fas.harvard.edu/~junliu/BioOptimizer/>.

Obtaining a local copy of BioProspector:

BioProspector is free-of-charge to academia. Please check out:

[Brutlag Bioinformatics Group Software Download](#) and [Academic License Instructions](#) for details.

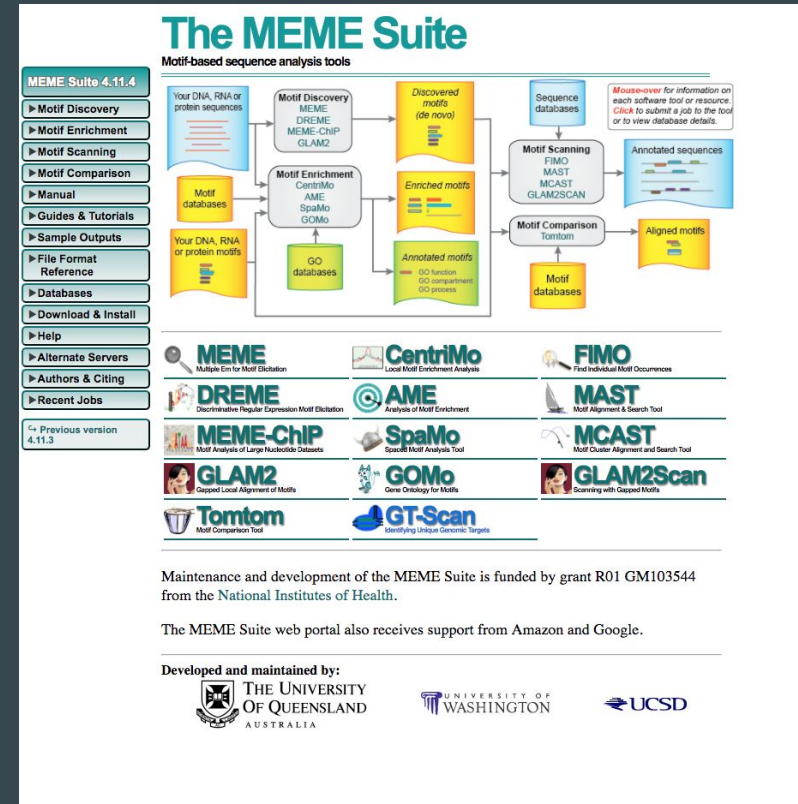
Reference:

[Liu X, Brutlag DL, Liu JS. BioProspector: discovering conserved DNA motifs in upstream regulatory regions of co-expressed genes. *Pac Symp Biocomput*, 2001:::127-38.](#)

Used Benchmarking Tool

- MEME

- Another tool for DNA motif discovery and analysis
- Can submit a **FASTA** file request into the software, and it will generate an output for you
- Uses statistical modeling techniques to choose the following for each motif:
 - Best width
 - Number of occurrences
 - Description
- Runtime:
 - Cubic under 200,000 characters
 - Quadratic over 200,000 characters



BioProspector vs. MEME

BioProspector

- Command to run:
 - `./BioProspector -i small_seq_1.fasta -W 10 -o benchmark1`
 - `-i` = flag for FASTA file
 - `-W` = flag for motif length
 - `-o` = flag for output file to write results to
- Output is harder to read and analyze than MEME
- Output doesn't detect all motifs we expect

MEME

- Software can be submitted through online
- Input is a group of sequences
- Output as many motifs as requested
- Data submission form
- Output is easier to read and analyze than BioProspector
- Output does detect all the motifs we expect

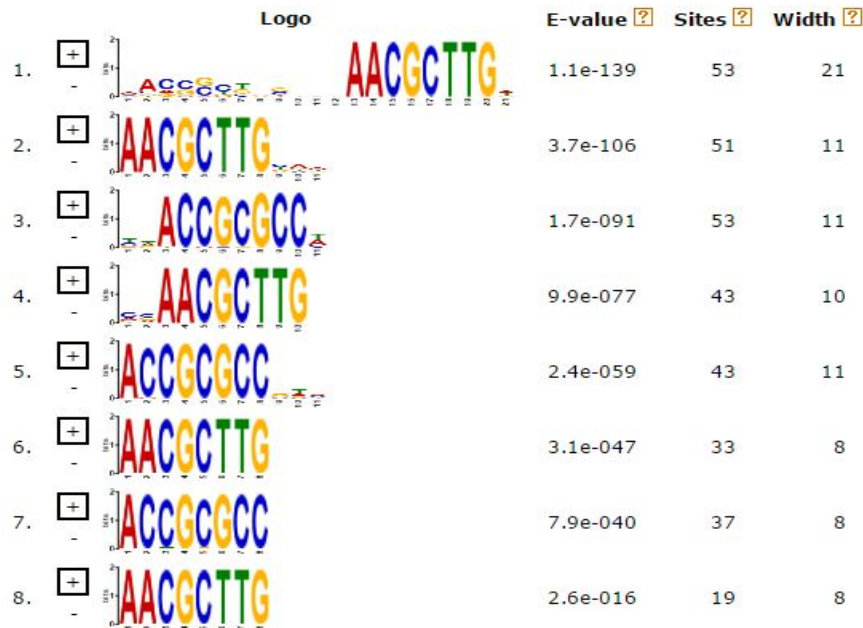
BioProspector vs. MEME

BioProspector

MEME

```
247 Motif #2: (TCGCAATCA/TGATTGCGA)
248 *****
249 Width (10, 0); Gap [0, 0]; MotifScore 5.079; Sites 49
250
251 Blk1 A C G T Con rCon Deg rDeg
252 1 0.03 0.04 0.04 99.89 T A T A
253 2 0.03 99.90 0.04 0.03 C G C G
254 3 0.03 0.04 99.90 0.03 G C G C
255 4 0.03 99.90 0.04 0.03 C G C G
256 5 99.89 0.04 0.04 0.03 A T A T
257 6 99.89 0.04 0.04 0.03 A T A T
258 7 99.89 0.04 0.04 0.03 A T A T
259 8 0.03 0.04 0.04 99.89 T A T A
260 9 0.03 99.90 0.04 0.03 C G C G
261 10 99.89 0.04 0.04 0.03 A T A T
262
263 >seq_1 len 160 site #1 r 24
264 TCGCAATCA
265 >seq_2 len 160 site #1 r 130
266 TCGCAATCA
267 >seq_5 len 160 site #1 r 89
268 TCGCAATCA
269 >seq_7 len 160 site #1 r 38
270 TCGCAATCA
271 >seq_8 len 160 site #1 r 17
272 TCGCAATCA
273 >seq_10 len 160 site #1 r 83
274 TCGCAATCA
275 >seq_11 len 160 site #1 r 93
276 TCGCAATCA
277 >seq_13 len 160 site #1 r 144
278 TCGCAATCA
279 >seq_14 len 160 site #1 r 34
280 TCGCAATCA
281 >seq_14 len 160 site #2 r 14
282 TCGCAATCA
283 >seq_17 len 160 site #1 r 81
284 TCGCAATCA
```

DISCOVERED MOTIFS



Parameters

In our code:

- Strand Length - Small (2,000), Medium (3,500), Large (5,000), Huge (10,000)
- Frequency - Sparse($\sim 0.1\%$), Semi-Saturated(1-3%), Saturated (7-10%)
- Motif Length - 8mer, 14mer, 20mer

In MEME:

- Number of motifs MEME is looking for.
- Motif minimum / max length - How long the motifs must be to be considered.

Benchmarking Results

Time to Run: 4.2 seconds

Small | 8mer | Saturated | 180 Sequence Length | 2000 Character Length




Type of Result:	Count:
True Positive	Found all 93 of the motifs we inserted.
True Negative	0
False Positive	Discovered an additional motif at two sites.
False Negative	Zero, all motifs found

Time to Run: 595.72 seconds









Large | 8mer | Saturated | 180 Sequence Length | 5000 Character Length

Type of Result:	Count:
True Positive	366/460
True Negative	0
False Positive	540 sites for 90 motifs were found that we did not place
False Negative	94 sites for our two motifs were not found

DISCOVERED MOTIFS

	Logo	E-value ?	Sites ?	Width ?
1.		5.5e-094	43	9
2.		5.7e-090	50	8
3.		1.9e+004	2	7

DISCOVERED MOTIFS

	Logo	E-value ?	Sites ?	Width ?
1.		1.1e-139	53	21
2.		3.7e-106	51	11
3.		1.7e-091	53	11
4.		9.9e-077	43	10
5.		2.4e-059	43	11
6.		3.1e-047	33	8
7.		7.9e-040	37	8
8.		2.6e-016	19	8

Benchmarking Results

Time to Run: 293.97 seconds

Huge | 20mer | Sparse | 180 Sequence Length | 10000 Character Length

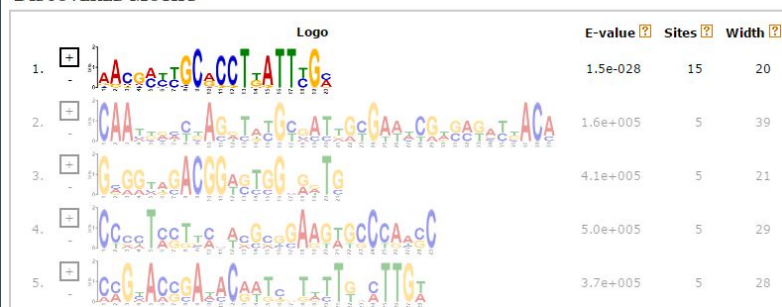
Type of Result:	Count:
True Positive	7/14 motifs were found
True Negative	0
False Positive	13 motifs at 57 sites were found that were not placed by our code
False Negative	Zero, all motifs found

Time to Run: 141.45 seconds

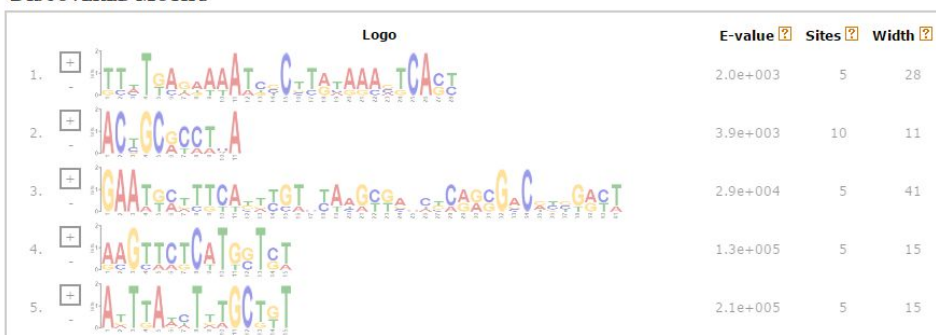
Large | 20mer | Sparse | 180 Sequence Length | 5000 Character Length

Type of Result:	Count:
True Positive	This trial showed 0/4 matches
True Negative	0
False Positive	25 additional motifs were found at 76 separate sites.
False Negative	4 of the sites from the two motifs we placed were not found



DISCOVERED MOTIFS



DISCOVERED MOTIFS

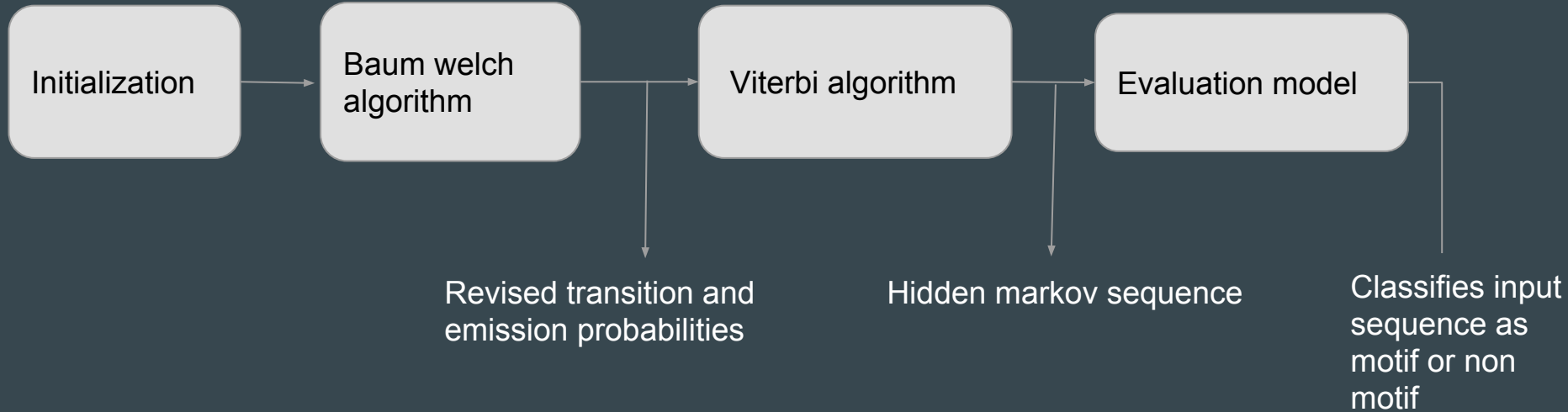


Benchmarking Analysis

- A direct relationship between the size of the dataset file and time MEME took to discover motifs
- Motif frequency (saturated)  , accuracy 
 - MEME works best with:
 - Smaller, saturated files
 - Small motif lengths
 - Small number of total motifs to find
 - MEME doesn't work well with:
 - Larger, sparsely populated files
 - Large motif lengths
 - Large number of total motifs to find
- Additional character(s) to the beginning or end of the motif
- Different motifs for a single long sequence

Our Implementation

Hidden Markov Model (HMM)



Snippets of code:

Forward and Backward Algorithm

```
void forward(){
    int s,s0,i;
    for(i=0; i<N-1; i++){
        int v = V[i+1];
        double C = log(0.0);
        for(s=0; s<STATE_SIZE; s++){
            alpha[i][s] = log(0.0);
            for(s0=0; s0<STATE_SIZE; s0++){
                double alpha0 = i==0 ? log(1.0) : alpha[i-1][s0];
                double delta = alpha0 + p[s][s0] + q[v][s0];
                alpha[i][s] = logAdd(alpha[i][s], delta);
            }
            C = logAdd(C, alpha[i][s]);
        }
        /* Normalize beta */
        for(s=0; s<STATE_SIZE; s++)
            alpha[i][s] -= C;
    }
    return;
}

/* Compute beta */
void backward(){
    int s,s1,i;
    for(i=N-1; i>=0; i--){
        int v = V[i+1];
        double C = log(0.0);
        for(s=0; s<STATE_SIZE; s++){
            beta[i][s] = log(0.0);
            for(s1=0; s1<STATE_SIZE; s1++){
                double beta1 = i==N-1 ? log(1.0) : beta[i+1][s1];
                double delta = alpha[i+1][s1] + p[s1][s] + q[v][s1];
                beta[i][s] = logAdd(beta[i][s], delta);
            }
            C = logAdd(C, beta[i][s]);
        }
        /* Normalize gamma */
        for(s=0; s<STATE_SIZE; s++)
            beta[i][s] -= C;
    }
    return;
}
```

Reestimation of parameters

```
void compute_Gamma_Sigma(){
    /* Compute gamma */
    int i,v,s1,s;
    for(i=0; i<N; i++){
        double C = log(0.0);
        for(s=0; s<STATE_SIZE; s++){
            gammas[i][s] = alpha[i][s] + beta[i][s];
            C = logAdd(C, gammas[i][s]);
        }
        /* Normalize gamma */
        for(s=0; s<STATE_SIZE; s++)
            gammas[i][s] -= C;
    }

    if(i == N-1)
        return;

    /* Compute sigma */
    int v = V[i+1];
    double C = log(0.0);
    for(s=0; s<STATE_SIZE; s++){
        sigma[i][v] = log(0.0);
        for(s1=0; s1<STATE_SIZE; s1++){
            double sigma1 = i==N-1 ? log(1.0) : sigma[i+1][v];
            double delta = gammas[i+1][s1] + p[s1][s] + q[v][s1];
            sigma[i][v] = logAdd(sigma[i][v], delta);
        }
        C = logAdd(C, sigma[i][v]);
    }
    /* Normalize sigma */
    for(s=0; s<STATE_SIZE; s++)
        sigma[i][v] -= C;
}
```

Outputs:
Transition probability matrix

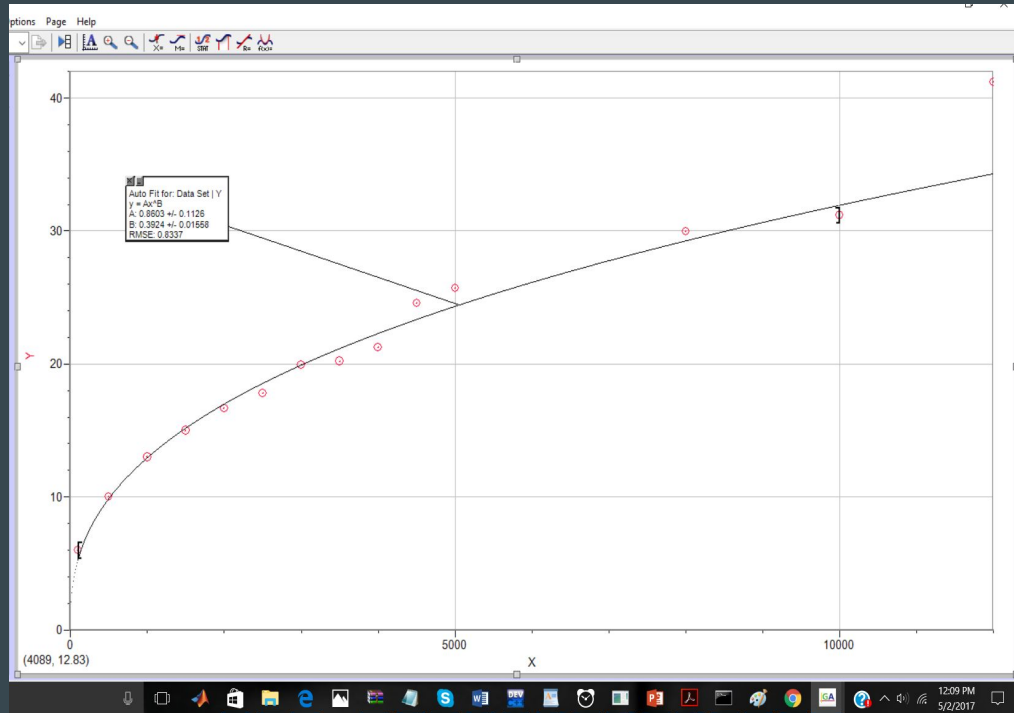
```
Select C:\phmm\bw.exe
success openingTransition probabil
state 1 -> state 1 0.646677
state 1 -> state 2 1.317000
state 2 -> state 1 1.472868
state 2 -> state 2 0.492372
```

Emission Probability Matrix

Emission probabilities:

```
state 1 : a 0.872098
state 2 : a 0.883400
state 1 : b 0.002271
state 2 : b 0.000000
state 1 : c 0.006820
state 2 : c 0.000000
state 1 : d 0.010949
state 2 : d 0.000000
state 1 : e 0.000000
state 2 : e 0.027574
state 1 : f 0.006299
state 2 : f 0.000000
state 1 : g 0.003884
state 2 : g 0.000000
state 1 : h 0.011120
state 2 : h 0.000000
state 1 : i 0.000000
state 2 : i 0.018035
```

Analysis of learning algorithm:



- Power function
- Average runtime for learning was impressive for training dataset sizes ranging between 35-40 KB containing about 10000 characters

Challenges encountered :

Numbers too small
to manipulate

Iterative

Deciding on
appropriate training
methods

Incorrect starting positions of
motifs are returned

How it was tackled:

- Used log adds and subs instead of multiplications and divisions
- An average of 600 iterations

Disadvantages of HMM

Deterministic vs. Probabilistic approach:

- Guarantees more accuracy
- Less time consuming

Disadvantages of HMM why it won't work for motif discovery

- Viterbi Space complexity: $O(s^*e)$; Time complexity: $O(n^*e)$
- HMMs needs more sequences to train on than simple MMs
- There can be multiple HMMs for a given training set

HMM → Suffix Tree

MOTIF FINDING

```
graph TD; A[MOTIF FINDING] --> B[Known/generated datasets]; B --> C[Probabilistic Methods  
HIDDEN MARKOV]; B --> D[Deterministic Methods  
SUFFIX TRIE]; C --> E[Optimistic about  
Local Alignment]; E --> F[Involved chances  
of uncertainty]; D --> G[Data Objectivity]; G --> H[Precisely  
determines outcomes];
```

Known/generated datasets

Probabilistic Methods
HIDDEN MARKOV

Deterministic Methods
SUFFIX TRIE

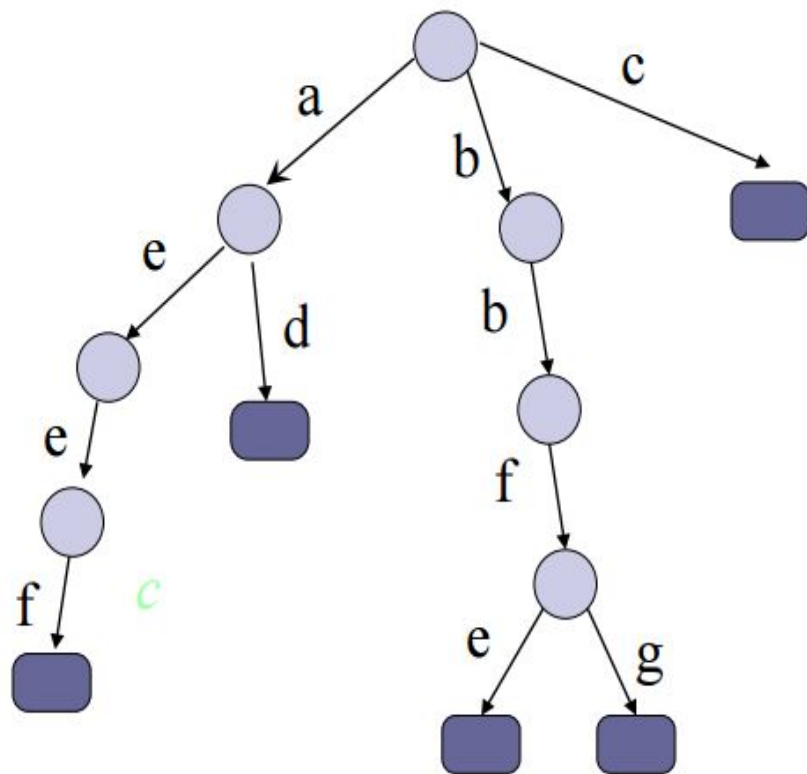
Optimistic about
Local Alignment

Data **Objectivity**

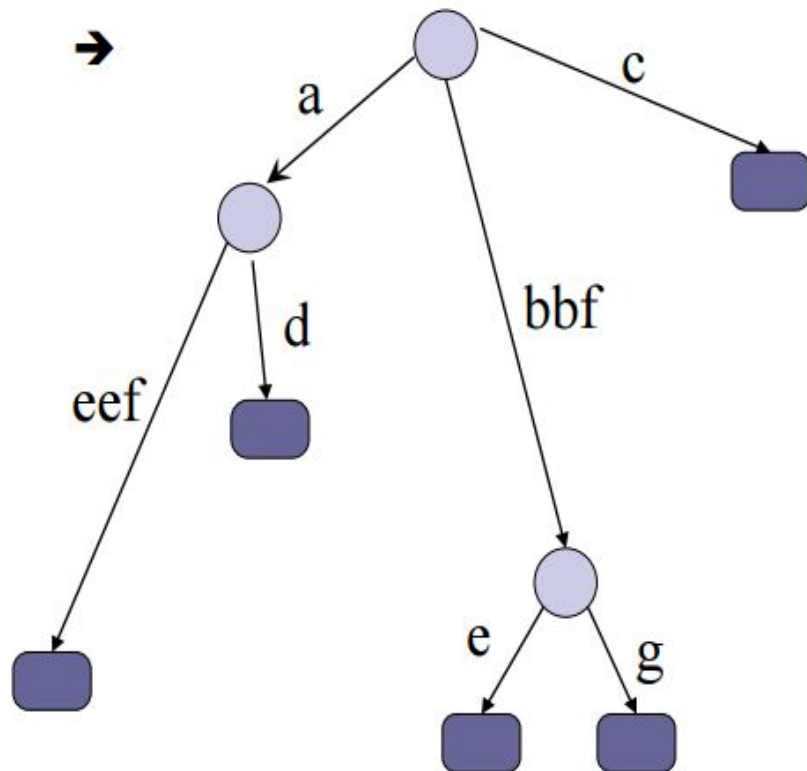
Involved chances
of **uncertainty**

Precisely
determines outcomes

GENERAL TRIE



COMPRESSED TRIE

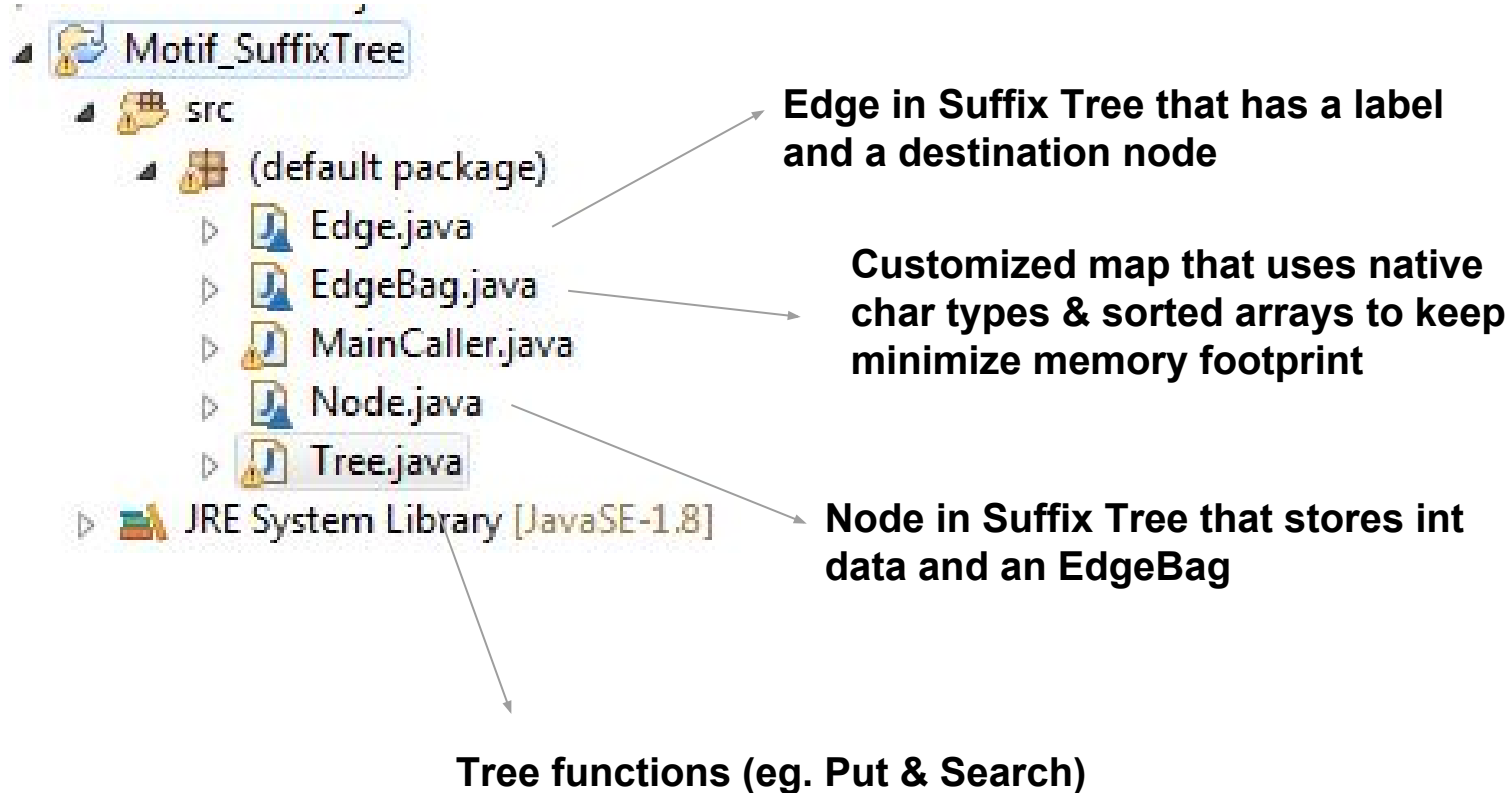


Building STs in linear time: Ukkonen's algorithm

- Ukkonen's algorithm gets subsequences and constructs them into a Suffix Tree
- Compared with older approaches, Ukkonen's particular techniques:
 - removing \$ from all edge labels
 - removing any edge that now has no label
 - removing any node with only one child
- Ukkonen reduces to $O(n)$ (linear) time, for constant-size alphabets, and $O(n \log n)$ in general.
- Suffix tree $\rightarrow O(n^2)/O(n^3)$



Suffix Tree Implementation



Results of Our Implementation

Suffix Tree Results

```
// Adds the specified index to the GSI under the given key.
// Entries must be inserted so that their indexes are in non-decreasing order,
// otherwise an IllegalStateException will be raised.

public void put(String key, int index) throws IllegalStateException {
    if (index < last) {
        throw new IllegalStateException("Waaittt index must not be less than any of the");
    } else {
        last = index;
    }

    // reset activeLeaf
    activeLeaf = root;

    String remainder = key;
    Node s = root;

    String text = "";
    // iterate over the string, one char at a time
    for (int i = 0; i < remainder.length(); i++) {
        // line 6
        text += remainder.charAt(i);
        // use intern to make sure the resulting string is in the pool.
        text = text.intern();

        // line 7: update the tree with the new transitions due to this new char
        Pair<Node, String> active = update(s, text, remainder.substring(i), index);
        // line 8: make sure the active pair is canonical
        // ...
    }
}
```

Put method of inserting/updating nodes, which helps keep the sequence indexes

```
File: small_sparse_1.txt
k-mer Length: 20
Motifs found: 11
AACGCTTGACCTTATTCA
1 4 6 9 22 27 37 44 45 48 49 51 52 57
ACCGCGCTGATTGCGAA
2 3 6 7 14 20 28 37 40 41 51 56 66
TTCTGGAGATAAGATCCGT
20 21 24 37 44 46 48 51 52 63 71
AATATGTATTAGAAACGCT
1 8 9 12 27 33 50 54 70 71
CGAAACTTCTGGAGATGT
2 3 11 12 25 40 45 51 53 66
AGCGCGCTGATTGCGAA
4 6 21 24 26 27 33 35 42 60
GCCGCTTATTATTGCGAA
2 3 14 21 29 37 43 55 59
GATCCGTCGAAAACCGCG
32 38 39 42 45 46 53 57 64
CTGATTTGCGAACTGATT
4 7 11 45 50 51 64 67
GCGAAGCGAACGCTGATT
1 7 9 17 19 21 31
TGCGAAATGCCCTCCATG
6 28 31 36 46 50 54
```

```
File: small_semi_saturated_2.txt
k-mer Length: 20
Motifs found: 85
TTCTGGAGATAAGATCCGT
11 25 33 46 50 20 52 63 71
AATATGTATTAGAAACGCT
```

Sequence
numbers
specified

Analysis of our Implementation

- All of our implementation are strongly object oriented

Suffix Tree → **EdgeBag** → **Edge** → **Node**

- For EdgeBag implementation, we **customized the Map** which stores the **native character data type** and **sorted array** of edges

Help minimize the memory taken

- Unlike generalize common suffix trees, which are generally used to build an index out of one very long string, this Suffix Trie can be used to build indexes over many strings.

Algorithm Shortcomings and Unsolved Challenges

- When the size of datasets grows
 - Memory space expands, causing the program to easily crash
- Implementation issue:
 - Our choice to represent tree branches: **HASHING**
 - Running time is optimized - takes $O(1)$ for each retrieval
 - Should find a way to balance between restrictions of space and speed

Alternative may be Balanced Tree and Suffix Links

MEME vs. Our Suffix Tree Implementation

Testing Files	Our Outputs	Designated Results
small_sparse_1.txt	11 motifs found	8+ motifs expected
small_sparse_2.txt	15 motifs found	13+ motifs expected
small_semi_saturated_2.txt	85 motif found	92+ motifs expected
medium_sparse_2.txt	132 motif found	130+ motifs expected
medium_semi_saturated_1.txt	998 motif found	1005+ motifs expected
medium_semi_saturated_2.txt	1029 motifs found	1032+ motifs expected

	MEME SUITE	SUFFIX TRIE
True Positive	No motifs found	~98% motifs found
True Negative	N/A	N/A
False Positive	~.5-.8% out of possible matches	0
False Negative	minimal	minimal

Extreme cases of sparse motif datasets

Milestones Hit

- Project Proposal ✓
- Dataset Generation ✓
- Learned a ton about BioProspector ✓
- Benchmarking on MEME ✓
- Our Implementation of the Algorithm ✓
- Visualization
- Presentation ✓ (Right now!)
- Final Project Report (TO DO)



Future Work

Potential Future Work

- Visualization
 - 3D representation of motif detection
 - Virtual Reality - motif detection in the human body
- Increase speed and accuracy of the suffix tree implementation
- Use an algorithm that combines deterministic and probabilistic methods
- More MEME analyzation and could test out the other tools in the MEME suite:
 - MEME
 - DREME
 - MEME-ChIP
 - GLAM2

References

- Timothy L. Bailey, Mikael Bodén, Fabian A. Buske, Martin Frith, Charles E. Grant, Luca Clementi, Jingyuan Ren, Wilfred W. Li, William S. Noble, "MEME SUITE: tools for motif discovery and searching", Nucleic Acids Research, 37:W202-W208, 2009.
- Github- <https://github.com/rala8730/python-motif-finding>
- Esko Ukkonen, "Finding approximate patterns in strings" ,Journal of Algorithms, Volume 6, Issue 1, March 1985
- E Ukkonen , "Algorithms for approximate string matching", Information and Control , 1985
- HMMF: An Hidden Markov Model Based Approach for Motif Finding - IEEE Xplore Document <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5162905>

Thank you for listening! Questions?