Subsequence and similarity search in biological sequences

An N-Gram Graph based approach

Vasileios Charisopoulos

June 8, 2016

Outline

1 Introduction

2 Preliminaries
- N-Gram Graphs
- Uses of N-Gram Graphs
- Nuisances about biosequence indexing

3 Our contribution
- Outline
- Similar Research
- Hashed Vector Encoding
- Indexing method
- Experiments

4 Future Directions

5 References

Motivation & Challenges

- Initially, focused on a more general task: graph indexing for a special class of graphs called *N-Gram Graphs*

Motivation & Challenges

- Initially, focused on a more general task: graph indexing for a special class of graphs called *N-Gram Graphs*
- Used to obtain a graph representation of text/sequential data (more about them in a few slides)

Motivation & Challenges

- Initially, focused on a more general task: graph indexing for a special class of graphs called *N-Gram Graphs*
- Used to obtain a graph representation of text/sequential data (more about them in a few slides)
- Turns out this task is pretty hard - as we'll see later, the fact that some of these graphs may be similar does not let us make any strong assumptions about their sources

Motivation & Challenges

- Initially, focused on a more general task: graph indexing for a special class of graphs called *N-Gram Graphs*
- Used to obtain a graph representation of text/sequential data (more about them in a few slides)
- Turns out this task is pretty hard - as we'll see later, the fact that some of these graphs may be similar does not let us make any strong assumptions about their sources

Queries on nucleotide databases

Given a **database** D containing nucleotide sequences and a **query sequence** s_q, find all entries $s_i \in D$ that contain s_q (or a sequence very similar to s_q)

Table of Contents

1. Introduction
2. Preliminaries
   - N-Gram Graphs
   - Uses of N-Gram Graphs
   - Nuisances about biosequence indexing
3. Our contribution
   - Outline
   - Similar Research
   - Hashed Vector Encoding
   - Indexing method
   - Experiments
4. Future Directions
5. References

Preliminaries

What is a N-Gram Graph?

- Introduced by George Giannakopoulos

Preliminaries

What is a N-Gram Graph?

- Introduced by George Giannakopoulos
- Encodes sequential data (e.g. text) into a set of **n-grams**. Every distinct n-gram becomes a vertex.

Preliminaries

What is a N-Gram Graph?

- Introduced by George Giannakopoulos
- Encodes sequential data (e.g. text) into a set of **n-grams**. Every distinct n-gram becomes a vertex.
- N-gram pairs that exist within a certain distance (D_w) of one another are drawn as edges connecting the vertices corresponding to the co-occuring n-grams.

Preliminaries

What is a N-Gram Graph?

- Introduced by George Giannakopoulos
- Encodes sequential data (e.g. text) into a set of **n-grams**. Every distinct n-gram becomes a vertex.
- N-gram pairs that exist within a certain distance (D_w) of one another are drawn as edges connecting the vertices corresponding to the co-occuring n-grams.
- Weight of edge connecting n_1, n_2 is usually the number of times n_2 occurs before n_1 within a distance D_w in the text

Preliminaries

What is a N-Gram Graph?

- Introduced by George Giannakopoulos
- Encodes sequential data (e.g. text) into a set of **n-grams**. Every distinct n-gram becomes a vertex.
- N-gram pairs that exist within a certain distance (D_w) of one another are drawn as edges connecting the vertices corresponding to the co-occuring n-grams.
- Weight of edge connecting n_1, n_2 is usually the number of times n_2 occurs before n_1 within a distance D_w in the text
- If more than a single N are specified, create one such graph for each N value

Preliminaries

What is a N-Gram Graph?

- Introduced by George Giannakopoulos
- Encodes sequential data (e.g. text) into a set of **n-grams**. Every distinct n-gram becomes a vertex.
- N-gram pairs that exist within a certain distance (D_w) of one another are drawn as edges connecting the vertices corresponding to the co-occuring n-grams.
- Weight of edge connecting n_1, n_2 is usually the number of times n_2 occurs before n_1 within a distance D_w in the text
- If more than a single N are specified, create one such graph for each N value
- The resulting graph is called an *N-Gram Graph*.

Preliminaries
Some examples

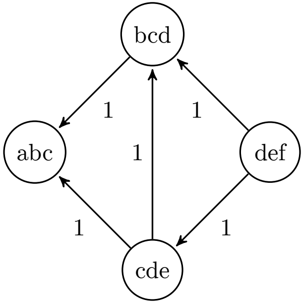


Figure: N-Gram Graph for “abcdef”, $N = 3, D_w = 2$

Preliminaries
Some examples

What about multiple occurences of n-grams?

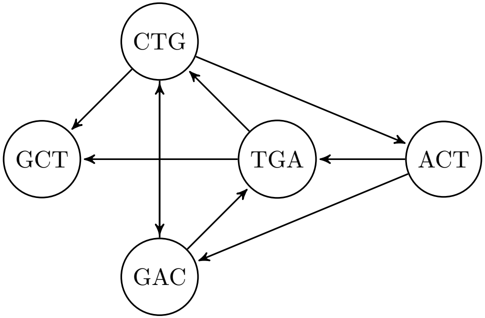


Figure: N-Gram Graph for “GCTGACTG”, $N = 3, D_w = 2$

N-Gram Graph Similarity

In [Giannakopoulos,], a standard way to measure similarity is defined:

Value Similarity VS(G_i, G_j)

Equal to the sum of the value ratios, $\frac{\min(w_e^i, w_e^j)}{\max(w_e^i, w_e^j)}$ for all edges that exist in both graphs, over the size of the largest of the two graphs ($\rightarrow$ measure based on edge weight distribution).

Summarization systems with higher VS scores performed better than other existing systems (Giannakopoulos et. al).

Table of Contents

1 Introduction

2 Preliminaries
- N-Gram Graphs
- Uses of N-Gram Graphs
- Nuisances about biosequence indexing

3 Our contribution
- Outline
- Similar Research
- Hashed Vector Encoding
- Indexing method
- Experiments

4 Future Directions

5 References

Preliminaries

Uses of N-Gram Graphs

1. Powerful tool for document summarization

Preliminaries

Uses of N-Gram Graphs

1. Powerful tool for document summarization
2. Have also been applied for classification of biological data [Polychronopoulos et al., 2014]

Preliminaries

Uses of N-Gram Graphs

1. Powerful tool for document summarization
2. Have also been applied for classification of biological data [Polychronopoulos et al., 2014]
3. Spam filters :)

Preliminaries

Uses of N-Gram Graphs

1. Powerful tool for document summarization
2. Have also been applied for classification of biological data [Polychronopoulos et al., 2014]
3. Spam filters :)
4. Potentially useful for many machine learning tasks involving sequential or serializable data

Preliminaries

Uses of N-Gram Graphs

1. Powerful tool for document summarization
2. Have also been applied for classification of biological data [Polychronopoulos et al., 2014]
3. Spam filters :)
4. Potentially useful for many machine learning tasks involving sequential or serializable data

Yet another task

Time-efficient indexing of biological sequences

Table of Contents

1. Introduction
2. Preliminaries
    - N-Gram Graphs
    - Uses of N-Gram Graphs
    - Nuisances about biosequence indexing
3. Our contribution
    - Outline
    - Similar Research
    - Hashed Vector Encoding
    - Indexing method
    - Experiments
4. Future Directions
5. References

Nuisances about biosequence indexing

Traditional methods

1. Unlikely to avoid exhaustive searches (e.g. BLAST performs a linear scan)
2. Might end up operating on whole sequences (oftentimes several kbp long)
3. Lots of optimization involved even for marginally better performance

Nuisances about biosequence indexing

Traditional methods

1. Unlikely to avoid exhaustive searches (e.g. BLAST performs a linear scan)
2. Might end up operating on whole sequences (oftentimes several kbp long)
3. Lots of optimization involved even for marginally better performance

Our contribution

We attack the above issues, especially (1) and (2), by encoding sequences using N-Gram Graphs and using PATRICIA tries to find the most “similar” representations in computational time that is only slightly affected by database size.

Table of Contents

1 Introduction

2 Preliminaries
- N-Gram Graphs
- Uses of N-Gram Graphs
- Nuisances about biosequence indexing

3 Our contribution
- Outline
- Similar Research
- Hashed Vector Encoding
- Indexing method
- Experiments

4 Future Directions

5 References

Outline of our method

1. Devise an encoding method for N-Gram Graphs so that similarity between graphs can be computed in constant time t_c. (*Hashed vector encoding*, more in a few slides later)

Outline of our method

1. Devise an encoding method for N-Gram Graphs so that similarity between graphs can be computed in constant time t_c. (*Hashed vector encoding*, more in a few slides later)
2. Serialize this encoding to utilize the performance of PATRICIA tries

PATRICIA trie

Also known as radix tree - able to perform closest key queries in $\mathcal{O}(a(K))$ expected time, where $a(K)$ is the average number of bits of all items in the trie. [Morrison, 1968]

Outline of our method

1. Devise an encoding method for N-Gram Graphs so that similarity between graphs can be computed in constant time t_c. (*Hashed vector encoding*, more in a few slides later)
2. Serialize this encoding to utilize the performance of PATRICIA tries
3. Split all database sequences in small, non-overlapping chunks (typically not longer than 100-200 bp).

PATRICIA trie

Also known as radix tree - able to perform closest key queries in $\mathcal{O}(a(K))$ expected time, where $a(K)$ is the average number of bits of all items in the trie. [Morrison, 1968]

Outline of our method

1. Devise an encoding method for N-Gram Graphs so that similarity between graphs can be computed in constant time t_c. (*Hashed vector encoding*, more in a few slides later)
2. Serialize this encoding to utilize the performance of PATRICIA tries
3. Split all database sequences in small, non-overlapping chunks (typically not longer than 100-200 bp).
4. Index all chunks by encoding each chunk with its hash vector representation and inserting the serialized version of this into a PATRICIA trie.

PATRICIA trie

Also known as radix tree - able to perform closest key queries in $\mathcal{O}(a(K))$ expected time, where $a(K)$ is the average number of bits of all items in the trie. [Morrison, 1968]

Table of Contents

1 Introduction

2 Preliminaries
- N-Gram Graphs
- Uses of N-Gram Graphs
- Nuisances about biosequence indexing

3 Our contribution
- Outline
- Similar Research
- Hashed Vector Encoding
- Indexing method
- Experiments

4 Future Directions

5 References

Similar research:

(i) *Xia Cao, Shuai Cheng Li, and Anthony K. H. Tung - Indexing DNA sequences using q-grams (2005)* [Cao et al., 2005] - Use the presence or absence of q-grams to construct a binary vector which is stored in a tree index $\rightarrow$ does not use any information about neighborhood structure

(ii) *Rakthanmanon et. al. - Searching and Mining Trillions of Time Series Subsequences under Dynamic Time Warping* [Rakthanmanon et al., 2012] (2012 SIGKDD best paper award) - General purpose mining in very large scale $\rightarrow$ lots of pruning, but doesn't avoid exhaustive scans

Table of Contents

1 Introduction

2 Preliminaries
- N-Gram Graphs
- Uses of N-Gram Graphs
- Nuisances about biosequence indexing

3 Our contribution
- Outline
- Similar Research
- Hashed Vector Encoding
- Indexing method
- Experiments

4 Future Directions

5 References

Let's vectorize our graphs

Outline of the idea: use edge connections to include some info about the neighborhood structure into a vector representation.

(i) "Hash" vertices by the prefix of their labels, so that vertices with the same prefix hash to the same values (e.g. " AC T" - " AC G" hash to the same value) $\rightarrow$ 16 distinct hash values.

(ii) Assign an encoding value to each "bucket" - the number of distinct vertices that are connected to any one of the vertices in that bucket $\rightarrow$ value between $[0, 4^N - 1]$ for n-gram size N.

For large enough N (practically, $N \geq 3$), we expect that different DNA sequences of equal lengths will generally not produce the same encoding vector.

If we want / need less bits per vector, we can quantize our values (resolution $\downarrow$, efficiency $\uparrow$).

Example

Let's revisit a previous graph:

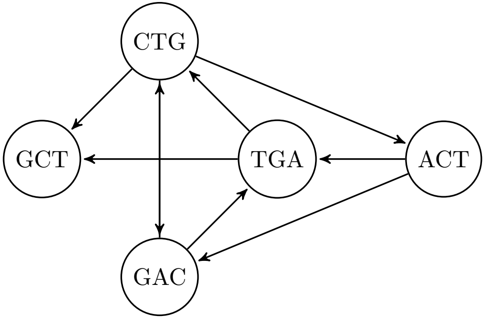


Figure: N-Gram Graph for “GCTGACTG”, $N = 3, D_w = 2$

Example

- 5 distinct prefixes: {GC, CT, TG, GA, AC}
- Hash function h (essentially lexicographic ordering on dinucleotides):

$$h([c_1 c_2 c_3]) = \begin{cases} 0 & [c_1 c_2] = \text{AA} \\ 1 & [c_1 c_2] = \text{AC} \\ & \dots \\ 15 & [c_1 c_2] = \text{TT} \end{cases}$$

Example

- 5 distinct prefixes: {GC, CT, TG, GA, AC}
- Hash function h (essentially lexicographic ordering on dinucleotides):

$$h([c_1c_2c_3]) = \begin{cases} 0 & [c_1c_2] = \text{AA} \\ 1 & [c_1c_2] = \text{AC} \\ & \dots \\ 15 & [c_1c_2] = \text{TT} \end{cases}$$

- Every “bucket” (set of vertices that hash to the same value) has 2 incoming connections from distinct vertices, except for “ACT”.
- Encoding value:

$$v(G_1) = \{0, 1, 0, 0, 0, 0, 0, 2, 2, 2, 0, 0, 0, 0, 2, 0\}$$

Example

Now, change the graph slightly:

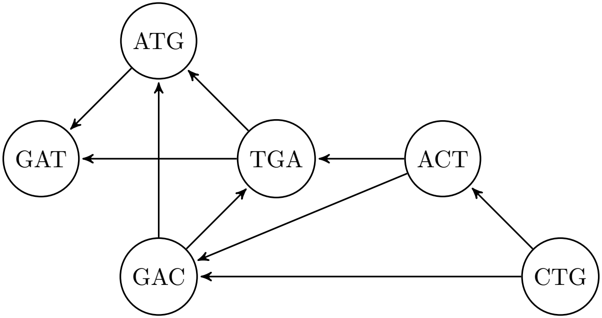


Figure: N-Gram Graph for “GATGACTG”, $N = 3, D_w = 2$

Example

- 5 distinct prefixes: {GA, AT, TG, AC, CT}
- Hash function h same as before

Example

- 5 distinct prefixes: {GA, AT, TG, AC, CT}
- Hash function h same as before
- The GA "bucket" has incoming connections from:

$$\{\text{ATG, TGA, ACT, CTG}\}$$

- Encoding value:

$$v(G_2) = \{0, 1, 0, 2, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 2, 0\}$$

Example

- 5 distinct prefixes: {GA, AT, TG, AC, CT}
- Hash function h same as before
- The GA "bucket" has incoming connections from:

$$\{\text{ATG, TGA, ACT, CTG}\}$$

- Encoding value:

$$v(G_2) = \{0, 1, 0, 2, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 2, 0\}$$

Example

- 5 distinct prefixes: {GA, AT, TG, AC, CT}
- Hash function h same as before
- The GA "bucket" has incoming connections from:

$$\{\text{ATG, TGA, ACT, CTG}\}$$

- Encoding value:

$$v(G_2) = \{0, 1, 0, 2, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 2, 0\}$$

Observation

2 very similar sequences result in a different graph
Manhattan distance:

$$||\mathbf{d}||_1 = |v(G_1) - v(G_2)| = 8$$

Table of Contents

1 Introduction

2 Preliminaries
- N-Gram Graphs
- Uses of N-Gram Graphs
- Nuisances about biosequence indexing

3 Our contribution
- Outline
- Similar Research
- Hashed Vector Encoding
- Indexing method
- Experiments

4 Future Directions

5 References

The Big Picture
Index construction

In a nutshell

1. Pick parameters:
   - Hash function h and n-gram size N
   - Length of indexed subsequences L_S
2. For all sequences s in database D:
   - Split s into non-overlapping subsequences of length L_S
   - Create an index entry containing the hashed vector encoding of each subsequence, obtain its string representation, and store it in a PATRICIA Trie.

Assuming a fixed-size string representation of length K in bits and average database sequence length M, the above incurs a computational cost of

$$\mathcal{O}\left(\frac{|D|M}{L_S} \cdot K\right)$$

The Big Picture
Query processing

Now, assume a query sequence s_q is checked against the database to find matching entries:

Query Processing

1. Split s_q in $|s_q| - L_S - 1$ subsequences of length L_S with $L_S - 1$ overlap.

The Big Picture
Query processing

Now, assume a query sequence s_q is checked against the database to find matching entries:

Query Processing

1. Split s_q in $|s_q| - L_S - 1$ subsequences of length L_S with $L_S - 1$ overlap.
2. For every subsequence, select the "nearest" entry (or entries) in the Trie. Retain only those results whose manhattan distance from the subsequence falls below a threshold T.

The Big Picture
Query processing

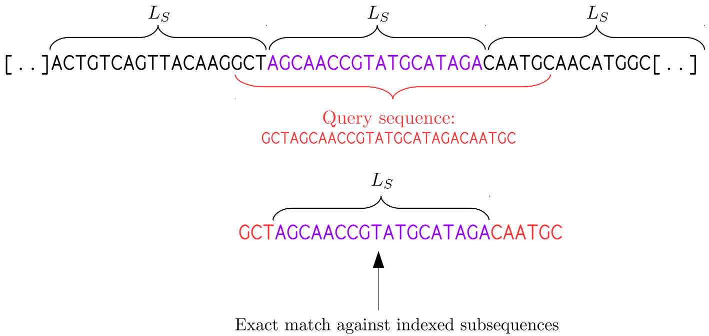
Now, assume a query sequence s_q is checked against the database to find matching entries:

Query Processing

1. Split s_q in $|s_q| - L_S - 1$ subsequences of length L_S with $L_S - 1$ overlap.
2. For every subsequence, select the "nearest" entry (or entries) in the Trie. Retain only those results whose manhattan distance from the subsequence falls below a threshold T.
3. Stop searching after L_S steps if a matching subsequence with $||\mathbf{d}||_1 = 0$ was found.

Claim: If the original sequence is part of the database (i.e. contains no mutations), a matching subsequence with $||\mathbf{d}|| = 0$ will be found after $\mathcal{O}(L_S)$ steps.

The Big Picture


L_S
L_S
L_S
[..]ACTGTCAGTTACAAGGCTAGCAACCGTATGCATAGACAATGCAACATGGC[..]
Query sequence:
GCTAGCAACCGTATGCATAGACAATGC
L_S
GCTAGCAACCGTATGCATAGACAATGC
Exact match against indexed subsequences

Remarks

Time complexity of queries

1. If a query sequence is part of some database sequence, the expected time complexity of a query is

$$\mathcal{O}(L_S \cdot K \cdot c(D))$$

2. If a query sequence s_q does not appear in any of the database sequences, the expected time of a query is

$$t_q \sim (|s_q| - L_S - 1) \cdot K \cdot c(D)$$

L_S: size of indexed subsequences
K: the length (in bits) of the encoding vectors' string representation
$c(D)$: collision factor $\rightarrow$ the average number of entries in the constructed index that have the same encoding vector

Remarks

Advantages & Limitations

- Query sequences s_q must have length $|s_q| \geq 2 \cdot L_S$. If $|s_q| << L_S$, no proper alignment is possible.

Remarks

Advantages & Limitations

- Query sequences s_q must have length $|s_q| \geq 2 \cdot L_S$. If $|s_q| << L_S$, no proper alignment is possible.
- Distance used is not edit distance - however, we know that two identical sequences produce exactly the same encoding vector.

Remarks

Advantages & Limitations

- Query sequences s_q must have length $|s_q| \geq 2 \cdot L_S$. If $|s_q| << L_S$, no proper alignment is possible.
- Distance used is not edit distance - however, we know that two identical sequences produce exactly the same encoding vector.
- Computing that distance can be done in $\mathcal{O}(4^L)$, where $L < N$. Value Similarity required $\mathcal{O}(E) = \mathcal{O}(4^{2N})$.

Remarks

Advantages & Limitations

- Query sequences s_q must have length $|s_q| \geq 2 \cdot L_S$. If $|s_q| << L_S$, no proper alignment is possible.
- Distance used is not edit distance - however, we know that two identical sequences produce exactly the same encoding vector.
- Computing that distance can be done in $\mathcal{O}(4^L)$, where $L < N$. Value Similarity required $\mathcal{O}(E) = \mathcal{O}(4^{2N})$.
- If we can avoid large collision coefficients, query time is not affected by database size!

Parameter tuning

Everything is a tradeoff

- We should avoid small prefix lengths L (typically $L > 2$) in order to avoid colliding entries. Larger L gives us more resolution but also increases key size K.

Parameter tuning

Everything is a tradeoff

- We should avoid small prefix lengths L (typically $L > 2$) in order to avoid colliding entries. Larger L gives us more resolution but also increases key size K.
- Using vertex degree as the encoding value, we can guarantee that it will be in $[0, 4^N - 1]$ for every index $\rightarrow$ fixed range for serialization.

Parameter tuning

Everything is a tradeoff

- We should avoid small prefix lengths L (typically $L > 2$) in order to avoid colliding entries. Larger L gives us more resolution but also increases key size K.
- Using vertex degree as the encoding value, we can guarantee that it will be in $[0, 4^N - 1]$ for every index $\rightarrow$ fixed range for serialization.
- A large L_S typically increases the chances that two subsequences will produce the same encoding vector, as well as the time complexity for query sequences that are not part of D. On the other hand, small L_S add more entries to the database.

Other concerns

Memory Footprint

In Theory: K bits per encoding vector, subsequence length L_S. Denoting the average length (in bp) by $\overline{L}$, we need roughly

$$n_{\text{entries}} = \Theta(|D| \cdot \frac{\overline{L}}{L_S})$$

entries in the constructed trie for a total (in bits) of

$$K_{\text{total}} = \Theta(|D| \cdot K\frac{\overline{L}}{L_S})$$

Other concerns

Memory Footprint

In Theory: K bits per encoding vector, subsequence length L_S. Denoting the average length (in bp) by $\overline{L}$, we need roughly

$$n_{\text{entries}} = \Theta(|D| \cdot \frac{\overline{L}}{L_S})$$

entries in the constructed trie for a total (in bits) of

$$K_{\text{total}} = \Theta(|D| \cdot K \frac{\overline{L}}{L_S})$$

The truth is always a little worse

In Java, a large memory overhead is incurred by the lack of ease in bit manipulation and the Patricia trie implementation.

Table of Contents

1 Introduction

2 Preliminaries
- N-Gram Graphs
- Uses of N-Gram Graphs
- Nuisances about biosequence indexing

3 Our contribution
- Outline
- Similar Research
- Hashed Vector Encoding
- Indexing method
- Experiments

4 Future Directions

5 References

Experiments

In the following experiments, we set $L_S = 150, N = 3, L = 2$ (key size $K = 1024$ bits).

E.Coli experiment

- Database size: $\simeq 5MB$, contains 400 entries in FASTA format – Number of indexed fragments: $30,875$
- 200 randomly selected query sequences of length $L_q = 300$.

Experiments

In the following experiments, we set $L_S = 150, N = 3, L = 2$ (key size $K = 1024$ bits).

E.Coli experiment

- Database size: $\simeq 5MB$, contains 400 entries in FASTA format – Number of indexed fragments: $30,875$
- 200 randomly selected query sequences of length $L_q = 300$.
- **Results**:
  - Recall: 200/200
  - Avg. Query Time $\overline{t_q}$: 213.05 ms
  - Avg. Answer Set Size: 4.665

Experiments

In the following experiments, we set $L_S = 150, N = 3, L = 2$ (key size $K = 1024$ bits).

E.Coli experiment

- Database size: $\simeq 5MB$, contains 400 entries in FASTA format – Number of indexed fragments: $30,875$
- 200 randomly selected query sequences of length $L_q = 300$.
- **Results**:
  - Recall: 200/200
  - Avg. Query Time $\overline{t_q}$: 213.05 ms
  - Avg. Answer Set Size: 4.665

D. Melanogaster Experiment

- Database size: $\simeq 119MB$, contains 1170 entries in FASTA format – Number of indexed fragments: $817,091$
- 100 randomly selected query sequences of length $L_q = 1500$

Experiments

In the following experiments, we set $L_S = 150, N = 3, L = 2$ (key size $K = 1024$ bits).

E.Coli experiment

- Database size: $\simeq 5MB$, contains 400 entries in FASTA format – Number of indexed fragments: $30,875$
- 200 randomly selected query sequences of length $L_q = 300$.
- **Results**:
  - Recall: 200/200
  - Avg. Query Time $\overline{t_q}$: 213.05 ms
  - Avg. Answer Set Size: 4.665

D. Melanogaster Experiment

- Database size: $\simeq 119MB$, contains 1170 entries in FASTA format – Number of indexed fragments: $817,091$
- 100 randomly selected query sequences of length $L_q = 1500$
- **Results**:
  - Recall: 100/100
  - Avg. Query Time $\overline{t_q}$: 224.91ms
  - Avg. Answer Set Size: 1.05

Effect of L_S on $\bar{t}_q$

Reminder

L_S: length of indexed subsequences

Expectation: if $L_S \uparrow$, $t_q \uparrow$ because of the larger window that must be checked exhaustively. We repeat the D. Melanogaster experiments with $L_S = \{100, 300\}$:

Effect of L_S on $\bar{t}_q$

Reminder

L_S: length of indexed subsequences

Expectation: if $L_S \uparrow$, $t_q \uparrow$ because of the larger window that must be checked exhaustively. We repeat the D. Melanogaster experiments with $L_S = \{100, 300\}$:

$L_S = 100$

- Number of indexed fragments: $1,255,968$
- **Results:**
  - Recall: $100/100$
  - Avg. Query Time $\bar{t}_q$: 122.87ms
  - Avg. Answer Set Size: 1.03

Effect of L_S on $\bar{t}_q$

Reminder

L_S: length of indexed subsequences

Expectation: if $L_S \uparrow$, $t_q \uparrow$ because of the larger window that must be checked exhaustively. We repeat the D. Melanogaster experiments with $L_S = \{100, 300\}$:

$L_S = 100$

- Number of indexed fragments: $1,255,968$
- **Results:**
  - Recall: 100/100
  - Avg. Query Time $\bar{t}_q$: 122.87ms
  - Avg. Answer Set Size: 1.03

$L_S = 300$

- Number of indexed fragments: $408,249$
- **Results:**
  - Recall: 100/100
  - Avg. Query Time $\bar{t}_q$: 649.97ms
  - Avg. Answer Set Size: 1.03

Experimental Validation

- Despite the fact that $|D|_{\text{melanogaster}} \geq 20 \cdot |D|_{\text{ecoli}}$, the average query times for the same L_S only differ by roughly 5% $\rightarrow$ query time only slightly affected by $|D|$!
- On the other hand, increasing L_S had a clear effect on execution time. In the case where $L_S = 300$, the query time is almost 3 times as large as when $L_S = 150$ $\rightarrow$ suggests using a small L_S.

Experimental Validation

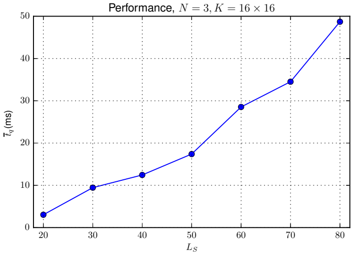
- Despite the fact that $|D|_{\text{melanogaster}} \geq 20 \cdot |D|_{\text{ecoli}}$, the average query times for the same L_S only differ by roughly 5% $\rightarrow$ query time only slightly affected by $|D|$!
- On the other hand, increasing L_S had a clear effect on execution time. In the case where $L_S = 300$, the query time is almost 3 times as large as when $L_S = 150 \rightarrow$ suggests using a small L_S.

Pending improvement

Solving the memory overhead problem is our greatest concern.

More fine-tuning

Noticing that the average answer set size is considerably small for both cases, we gave a shot at fine tuning by further reducing L_S and K:



L_S	Recall	$\bar{t}_q$(ms)	#Matches
20	100%	3.04	9.07
30	100%	9.45	1.53
40	100%	12.45	1.09
50	100%	17.40	1.05
60	100%	28.54	1.04
70	100%	34.52	1.06
80	100%	48.71	1.06

Running the BLAST algorithm (`ncbi-blast+` package in Debian) for the same query gave a performance of $\bar{t}_q \simeq 18$(ms).

Pending improvements

Implementation

(i) The current `PatriciaTrie` implementation incurs a heavy overhead on memory because it only allows `String` keys $\rightarrow$ 2 bytes per digit as per the Java Standard.

(ii) Our implementation only works with databases that are fully loaded in RAM so far.

(iii) Potential for parallelization; could solve the above issue in a distributed computing model.

Pending improvements

Implementation

(i) The current `PatriciaTrie` implementation incurs a heavy overhead on memory because it only allows `String` keys $\rightarrow$ 2 bytes per digit as per the Java Standard.

(ii) Our implementation only works with databases that are fully loaded in RAM so far.

(iii) Potential for parallelization; could solve the above issue in a distributed computing model.

Research

(i) More work (i.e. experiments & verification) needed on how to choose N, h, L_S etc.

(ii) Need to test on larger datasets ($\geq 1GB$) to measure scaling potential after solving the memory overhead issue.

References I

Cao, X., Li, S. C., and Tung, A. K. (2005).
Indexing dna sequences using q-grams.
In *Database Systems for Advanced Applications*, pages 4–16. Springer.

Giannakopoulos, G.
Automatic summarization from multiple documents.
PhD thesis.

Morrison, D. R. (1968).
Patricia—practical algorithm to retrieve information coded in alphanumeric.
Journal of the ACM (JACM), 15(4):514–534.

Papadakis, G., Giannakopoulos, G., and Paliouras, G. (2015).
Graph vs. bag representation models for the topic classification of web documents.
World Wide Web, pages 1–34.

References II

Polychronopoulos, D., Krithara, A., Nikolaou, C., Paliouras, G., Almirantis, Y., and Giannakopoulos, G. (2014).
Analysis and classification of constrained dna elements with n-gram graphs and genomic signatures.
In *Algorithms for Computational Biology*, pages 220–234. Springer International Publishing.

Rajaraman, A., Ullman, J. D., Ullman, J. D., and Ullman, J. D. (2012).
Mining of massive datasets, volume 1.
Cambridge University Press Cambridge.

Rakthanmanon, T., Campana, B., Mueen, A., Batista, G., Westover, B., Zhu, Q., Zakaria, J., and Keogh, E. (2012).
Searching and mining trillions of time series subsequences under dynamic time warping.
In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 262–270. ACM.

Reproducible research

- Source code is available in `Github` under the `GPLv3` license: https://github.com/VHarisop/BioGraphs (`git clone` us!)
- All of the datasets are available from the NCBI FTP Server (`ecoli.nt`, `drosoph.nt`).
- The query sequences were generated using `query_generator.py` from the `scripts/tools/` directory of `BioGraphs`.

Thanks to: George Giannakopoulos for supervising this project :)

Thank you!
Questions?