# N-Gram Graph Indexing for Queries on Nucleotide Databases

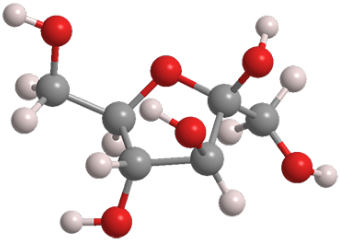
Original Work by *Vasileios Charisopoulos, George Giannakopoulos*
Presenter: Nikos Gialitsis

National and Kapodistrian University of Athens
National Center of Scientific Research Demokritos

*nikolasyal@di.uoa.gr*

February 16, 2021

# Graph Ecosystems



- NoSQL/Graph databases
- Semantic Web
- Gene/Potein/Molecule interactions
- Citation Networks
- Social Networks

DEMOKRITOS

# Graph Query Problem

## Typical Graph Query

Given a graph database $D = \{g_1, g_2, \ldots, g_n\}$ a query graph $q$, a graph query algorithm returns the answer set $D_q = \{g_i | q \subseteq g_i ; g_i \in D\}$

## Graph Similarity Query

Given a graph database $D = \{g_1, g_2, \ldots, g_n\}$ a query graph $q$, a graph query algorithm returns the answer set $D_q = \{g_i | q \approxeq g_i\}$

DEMOKRITOS

# Pipeline for Similarity Queries

1. Index Construction
2. Query Processing
   1. Searching : compute the candidate query answer set
   2. Verification : check if query graph is subgraph isomorphic to candidate graphs

- Query Complexity: $O(C_f + |C_q| \times C_v)$
  - $C_f$: time to compute candidate set
  - $|C_q|$ : size of candidate set
  - $C_v$ : cost of verification

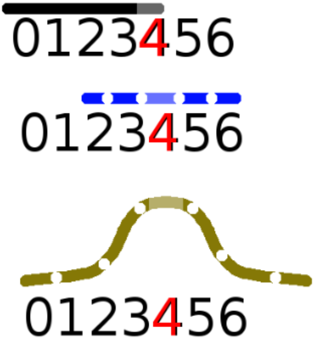
# N-Gram Graphs

1. extract n-grams from sequence
   - node = n-gram
   - edge = neighboring n-grams
2. if two n-grams co-occur within a window of size $D_{win}$ connect them with an edge
3. edges weighted by number of co-occurrences

DEMOKRITOS

# On neighborhood windows

Figure 3.1: Different types of n-gram windows (top to bottom): non-symmetric, symmetric and Gauss-normalized symmetric. N-gram *4* is the n-gram of interest.



# N-Gram Graph Properties

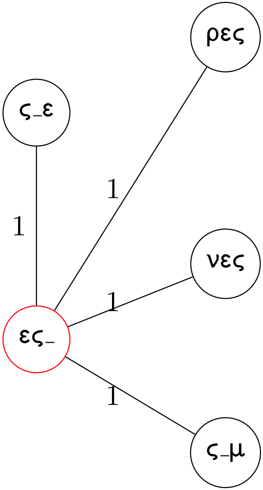
N-Gram Graphs

1. encode local-global information of sequences
2. are agnostic to the position of a co-occurrence
3. assign unique labels to nodes
   - key to efficient indexing!
4. applied to diverse domains
   - text summarization (initially)
   - genomics
   - nanotechnology
   - customer satisfaction

# Constructing N-Gram Graphs

"οι επόμενες μέρες είναι κρίσιμες" n=3 , Dwin = 1
{"οι_", "ι_ε", "_επ", "επο", "πομ", "ομε", "μεν", "ενε", "νες",
"**ες_**", "ς_μ", "_με", "μερ", "ερε", "ρες", "**ες_**", "ς_ε", "_ει",
"ειν", "ινα", "ναι", "αι_", "ι_κ", "_κρ", "κρι", "ρισ", "ισι", "σιμ",
"ιμε", "μες"}

# Constructing N-Gram Graphs



# Compacting Arbitrary Graphs to N-Gram Graphs

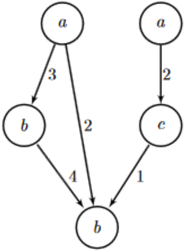



Fig. 1: An arbitrary graph $G$

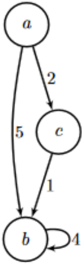



Fig. 2: Compact graph $G_{\text{ngg}}$ of Fig 1

- $D_{win} = 2$: $a \to b \to c$ contributes to edge $(a, c)$
- weight aggregation function: case-dependent
  - min/max of weights: distances e.g road networks
  - suum of weights : flow networks/traffic

# Value Similarity (VS)


$$\text{VS} = \frac{\text{1 common edge} \times \frac{\text{min weight}}{\text{max weight}}}{\text{2 maximum edges}} = \frac{1 \times \frac{1}{2}}{2} = \frac{1}{4}$$

# Vectoring the N-Gram Graph

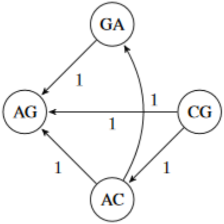



Fig. 3: N-Gram Graph $G_q$ for data string "AGACG"

$$h(i) = \begin{cases} 0 & i = A \\ 1 & i = C \\ 2 & i = G \\ 3 & i = T \end{cases}$$

$$\mathbf{u} = \textsc{Encoding}(G_q) = [4, 0, 2, 0]$$

$$key_{G_q} = \textsc{Serialize}(\mathbf{u}) = \underbrace{0\ldots0}_{12}1111\underbrace{0\ldots0}_{30}11\underbrace{0\ldots0}_{16}$$

# Key Proximity

- $XOR(u_1, u_2) = d_{manhattan}(u_1, u_2)$
- XOR metric between two encodings corresponds to vector distance
- Patricia Tries support efficient search operations based on the XOR metric
  - The Patricia Trie functions as an inverted index mapping encodings to graphs
  - Not "1-1" correspondence between encodings and graphs

# Indexing Algorithm

**Algorithm 3** Index Construction

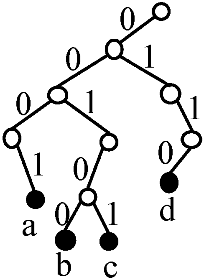
**function** BUILDINDEX($S = \{G_1, G_2, \ldots G_n\}, h$)
▷ where $h$ is the chosen hash function
  Initialize trie $T$
  **for** every graph $G_i \in S$ **do**
    $\mathbf{u} \leftarrow$ ENCODING($h, G_i$)
    *key* $\leftarrow$ SERIALIZE($\mathbf{u}$)
    INSERT($T$, *key*)
  **end for**
  **return** $T$
**end function**

# Approximate Search

**Algorithm 4** Approximate NN query

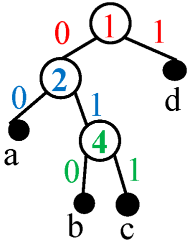
**function** APPROXIMATENN($T, q$)
    $\mathbf{u} \leftarrow$ ENCODING($q$)
    $key \leftarrow$ SERIALIZE($\mathbf{u}$)
    $G \leftarrow$ SELECT($T, key$)
**return** $G$
**end function**

# Trie vs Patricia Trie




prefixes

| | |
|---|---|
| a | 0 0 0 1 |
| b | 0 0 1 0 0 |
| c | 0 0 1 0 1 |
| d | 0 1 1 0 |




| *bit position* | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| a | 0 | 0 | 0 | 1 | |
| b | 0 | 0 | 1 | 0 | 0 |
| c | 0 | 0 | 1 | 0 | 1 |
| d | 0 | 1 | 1 | 0 | |

# Complexity Analysis

- entries returned will be approximate nearest neighbors
- if query exists in DB, perfect recall
- expected running time not directly dependent on the number of entries in database $D$
  - dependent on the uniformity of data distributed in the encoding vector subspace
  - scalable with big data

# Searching for nucleotide sequences

## Nucleotide Graph Query

given a set $S = \{s_1, s_2, \ldots, s_n\}$ of labelled DNA sequences and an unlabelled query sequence $s_q$, the set

$$N_s = argmin_{s \in S} \quad d(s, s_q) \tag{1}$$

is requested where $d(\cdot, \cdot)$ is an arbitrary distance measure between two sequences.

# Case Study: Indexing Genomes

Two genomes selected:

1. E.coli (bacterium) - 4.7MB
2. D. melanogaster (common fruitfly) - 120MB

Pipeline:

1. index and query each genome independently
2. verify that dataset size does not influence execution time
3. compare with BLAST (sota)

# Index Construction

1. Split all sequences into non-overlapping segments of length $L_S \in [20, 100]$ bps
2. construct a N-Gram Graph for each sequence to construct a graph database $D = \{G_1, G_2, \ldots, G_M\}$
3. For every $G_i$:
    1. Encode
    2. Serialize
    3. Insert to Patricia Trie

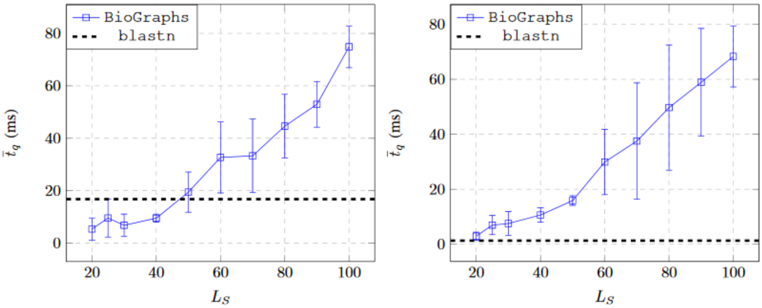
## Query Processing

Let query of length $|q|$:

1. Split into $|q| - L_S - 1$ overlapping segments (sliding window)
2. For each segment:
   1. get obtain candidate nearest neighbors
   2. filter out those whose Manhattan distance $< T_S$ threshold

# Execution Time vs Fragment Size

7

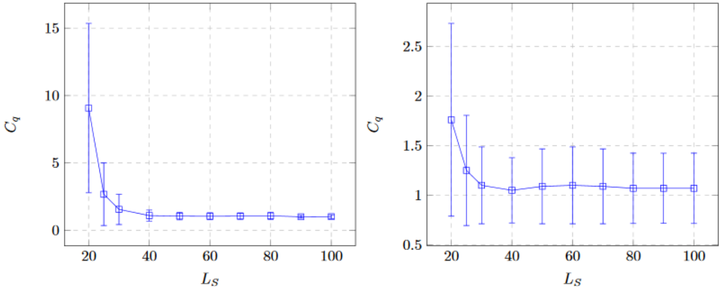



(a) $t_q$ on `drosoph.nt` (b) $t_q$ on `ecoli.nt`

Fig. 4: Query execution time over fragment size $L_S$

DEMOKRITOS

# Candidate Size vs Fragment Size

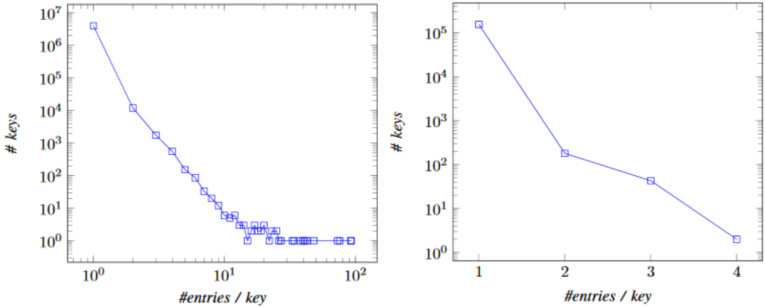



(a) $|C_q|$ on `drosoph.nt`

(b) $|C_q|$ on `ecoli.nt`

Fig. 5: Answer set size over fragment size $L_S$

# Distribution of Entries/Key




(a) drosoph.nt

(b) ecoli.nt

Fig. 6: Distribution of #entries / key, $L_S = 30$

# Recap

What was discussed:

1. How to perform approximate nearest neighbor search in NGGs
2. How to convert arbitrary graphs to NGGs
3. How to obtain vector representations from NGGs
   - hash-vector encoding
4. How to construct an inverted index on a Patricia Trie
5. How to apply above to nucleotide data

# Future Directions

- Experiment on larger and more diverse datasets
- Examine special cases of graphs for which we can take advantage of their properties for indexing
- Examine the task of checking for graph-isomorphism in NGGs in polynomial time (normally NP)
- Examine Graph Kernel Methods for NGGs

DEMOKRITOS

# Thank you! Questions?