March 30, 2011, The 5th International Workshop on Data-Mining and Statistical Science@Osaka University

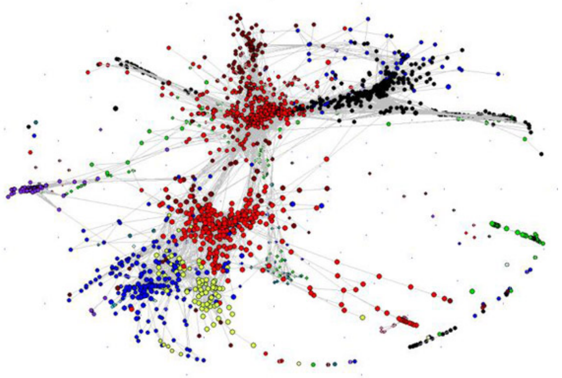
Kernel-based Similarity Search in Massive Graph Databases with Wavelet Trees

Yasuo Tabei
(JST ERATO Minato Project)
Joint work with Koji Tsuda (AIST)

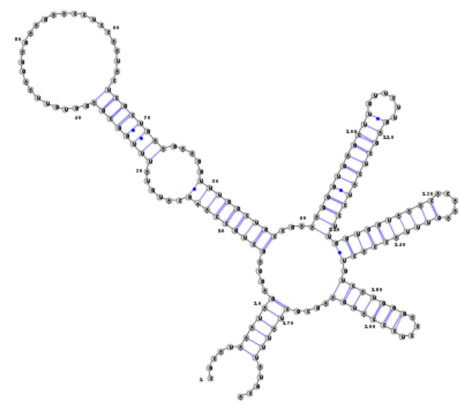
Outline

- Introduction
  - Recent development of graph databases
  - Needs for graph similarity search
  - Bag-of-words representation of a graph
  - Semi-conjunctive query
- Method
  - Scalable similarity search with wavelet trees
- Experiments
  - Use a large-scale graph dataset
  - 25 million chemical compounds

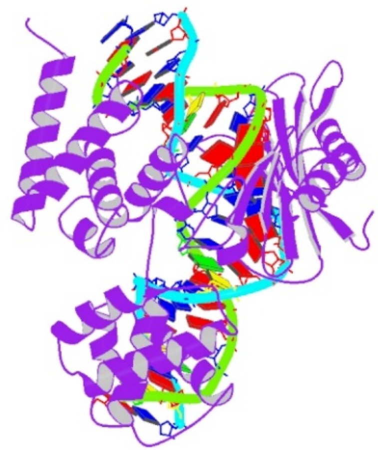
Graphs are everywhere



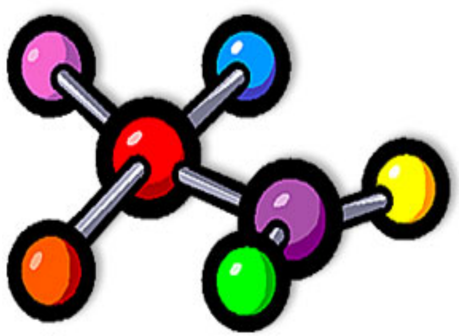
Gene co-expression network



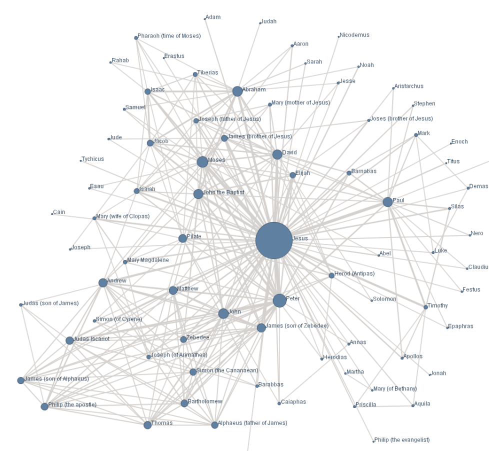
RNA 2D structure



Protein 3D structure



Chemical compound



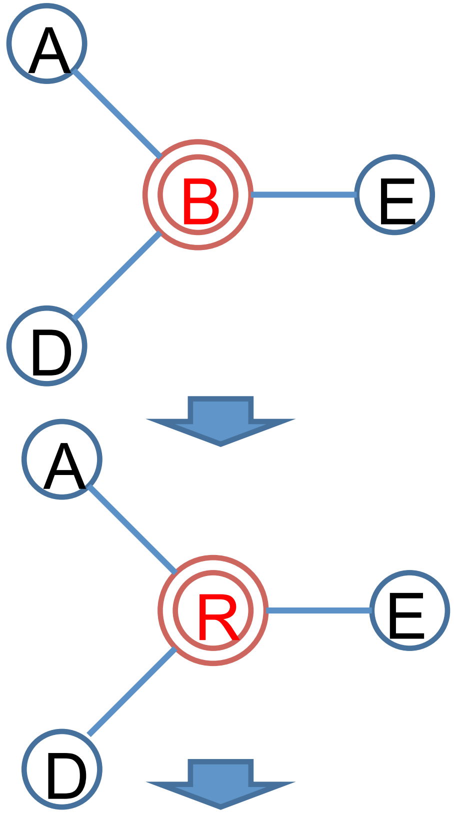
Social Network

Graph Similarity Search

- Retrieve graphs similar to the query
- Large databases
  - More than 20 million chemical compounds in PubChem database
- Bag-of-words representation of graphs
  - WL procedure (NIPS 2009)
- Why not use document retrieval methods?
  - Inverted index
  - Not that easy (explained later)

Weisfeiler-Lehman Procedure (NIPS,09)

- Convert a graph into a set of words (bag-of-words)



Bag-of-words
{A,B,D,E,…,R,…}

i) Make a label set of adjacent vertices ex) {E,A,D}

ii) Sort ex) A,D,E

iii) Add the vertex label as a prefix ex) B,A,D,E

iv) Map the label sequence to a unique value ex) B,A,D,E→R

v) Assign the value as the new vertex label

Search by cosine similarity

- Identify all graphs in the database whose cosine is larger than a threshold 1-ε

$$W_i \quad s.t \quad K_N(W_i, Q) = \frac{|W_i \cap Q|}{\sqrt{|W_i|}\sqrt{|Q|}} \geq 1-\varepsilon$$

  - W_i, Q: bag-of-words of graphs
- The above solution can be relaxed as follows,

If $K_N(Q,W) \geq 1-\varepsilon$, then

$$(1-\varepsilon)^2 |Q| \leq |W| \leq \frac{|Q|}{(1-\varepsilon)^2}$$

  - Can be used for fast search

Semi-conjunctive query

- Cosine query can be relaxed to the following form

$$W_i \quad s.t \quad |W_i \cap Q| \geq k$$

- The number of common words between two bag-of-words W_i and Q

Ex) $|W_i \cap Q| = |(A,C,E,F,H) \cap (A,E,I,J,L)|$
$= |(A,E)| = 2$

- $k=(1-\varepsilon)^2|Q|$
- No false negatives
- False positives can easily be filtered out by cosine calculations

Inverted index

- In natural language processing, inverted index has been used to solve semi-conjunctive query

Inverted Index

Word	Graph ids
A	2,8,13,15
B	4,9,13
C	8,10,16
D	1,3,11
E	4,9,13,14

- Associative map
- key ⇔word
- value⇔graph identifiers including a word

Bottom-up search

Inverted Index

Word	Graph ids
A	2,8,13,15
B	4,9,13
C	8,10,16
D	1,3,11
E	4,9,13,14

Query:(A,C,E)

Aggregation

(2,8,13,15,8,10,16,4,9,13,14)

Sort

(2,4,8,8,9,10,13,13,14,15,16)

k=2

i) Look the index up with query bag-of-words

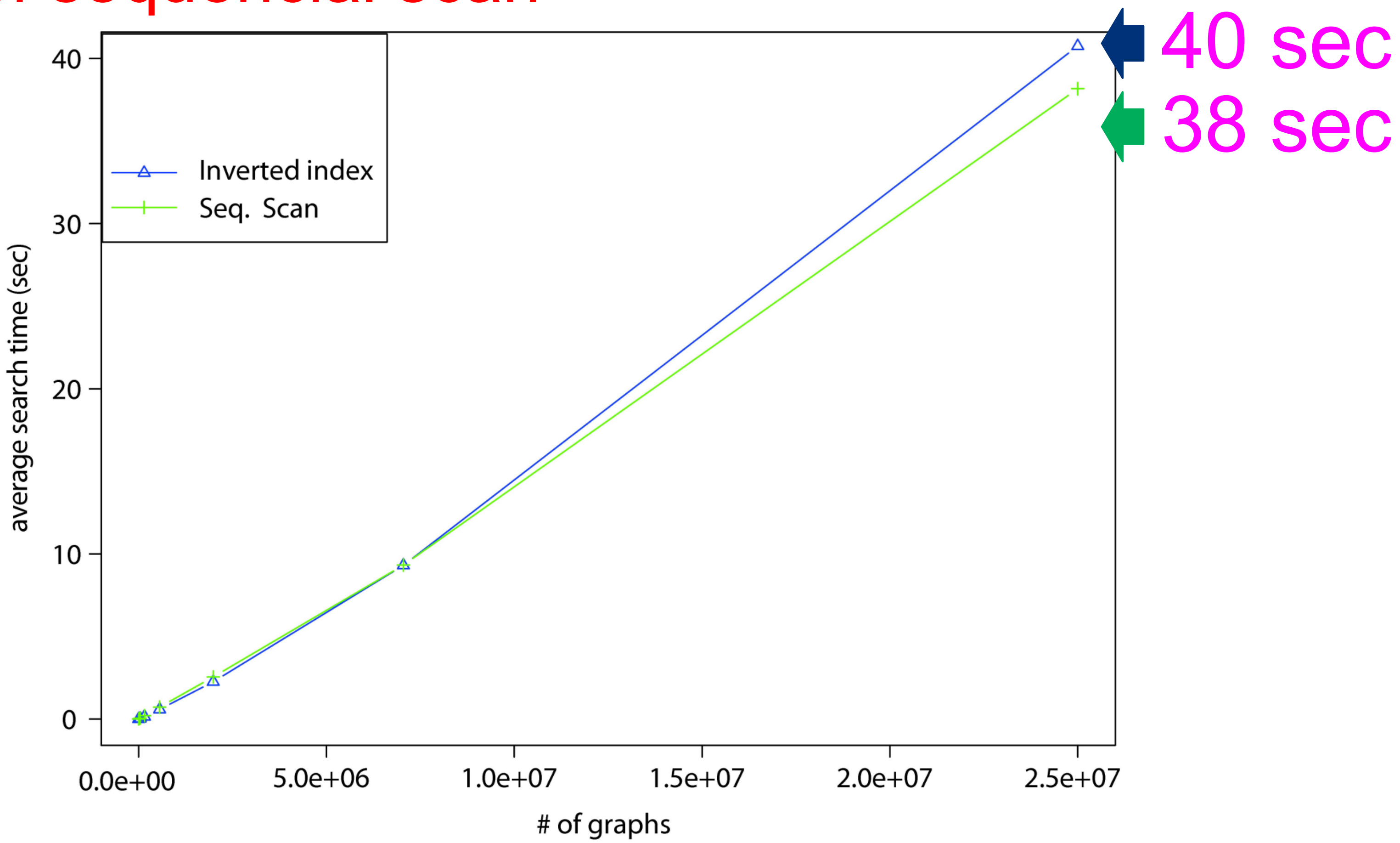
ii) Aggregate all the lists of graph indices

iii) Sort

iv) Scan

Search time of inverted index on 25 million graphs

- Search time of inverted index is not so different from that of sequencial scan



Why?

Query:(A,C,E)

⇩ Aggregation

(2,8,13,15,8,10,16,4,9,13,14)

⇩ Sort

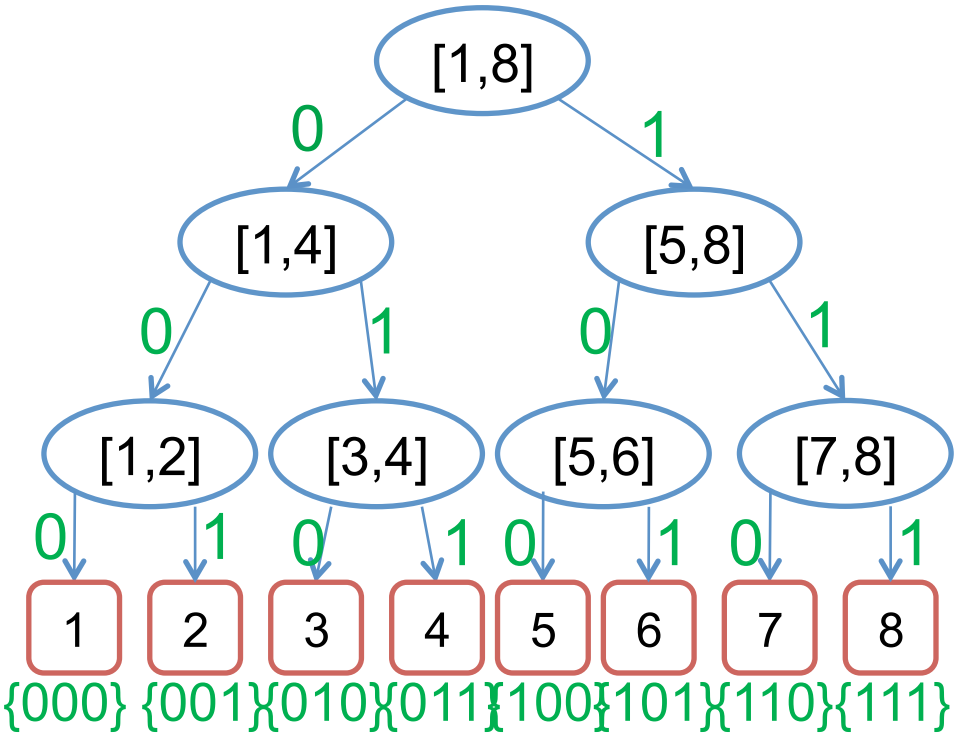
$\underbrace{(2,4,\underline{8,8},9,10,\underline{13,13},14,15,16)}_{C}$

- Each word is not specific enough
- Query contains 1000s of words
- Aggregated array is VERY long
- Sorting takes O(ClogC) in time

Overview of our method

- Top-down search in a tree over the series of graphs
- Huge memory, if tree is implemented with pointers
- Wavelet Tree: Succinct data structure
- The smaller the similarity threshold is, the quicker the algorithm finishes
    - Not the case in inverted index

Binary tree over graphs



- leaf ⇔ graph
- node ⇔ interval
- Each node is identified by a bit string (v={01})
- At the leaves, the graph indices correspond to int(v)+1

(e.g., int(010)+1=2+1=3)

Summarization of bag-of-words

- Represent bag-of-words as a bit array

1 2 3 4 5 6 7 8

Ex) $W_i=(1,3,4,7,8)$ ➡ $x_i=(1,0,1,1,0,0,1,1)$

- Take disjunction $\vee$ of all bit arrays in the interval of a node v

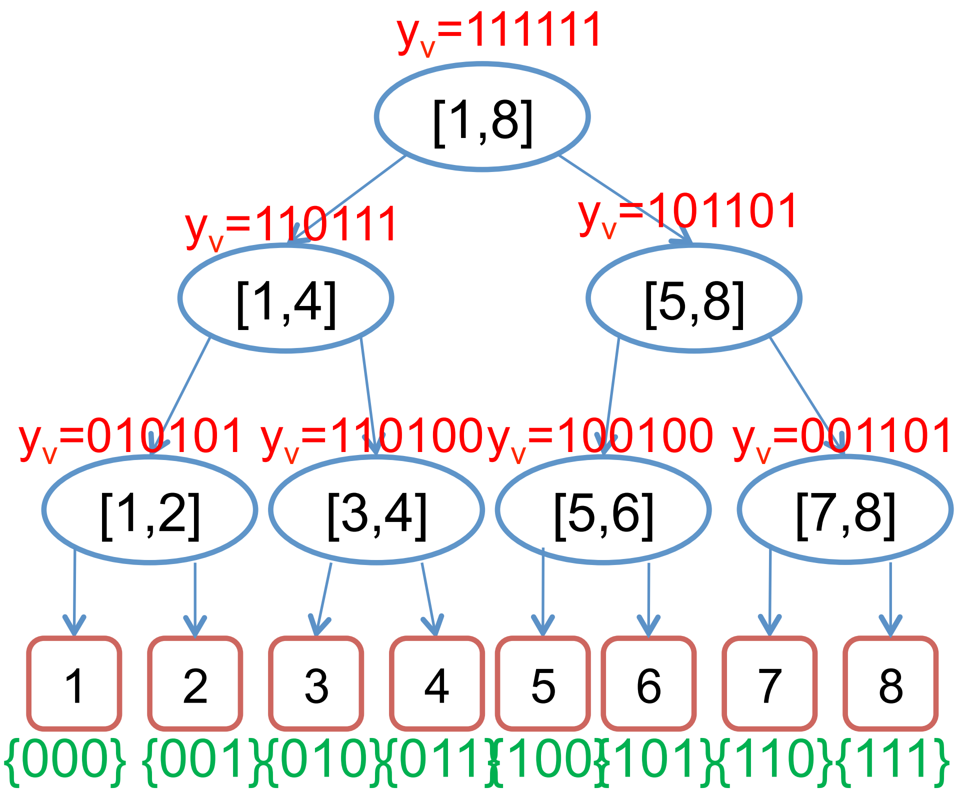
Ex) For an interval [1,4]

$X_1=(0,1,0,0,0,0,1,0)$
$X_2=(1,0,1,1,0,0,0,0)$
$X_3=(1,0,0,0,0,0,1,1)$
$X_4=(1,0,0,0,0,0,0,1)$

➡

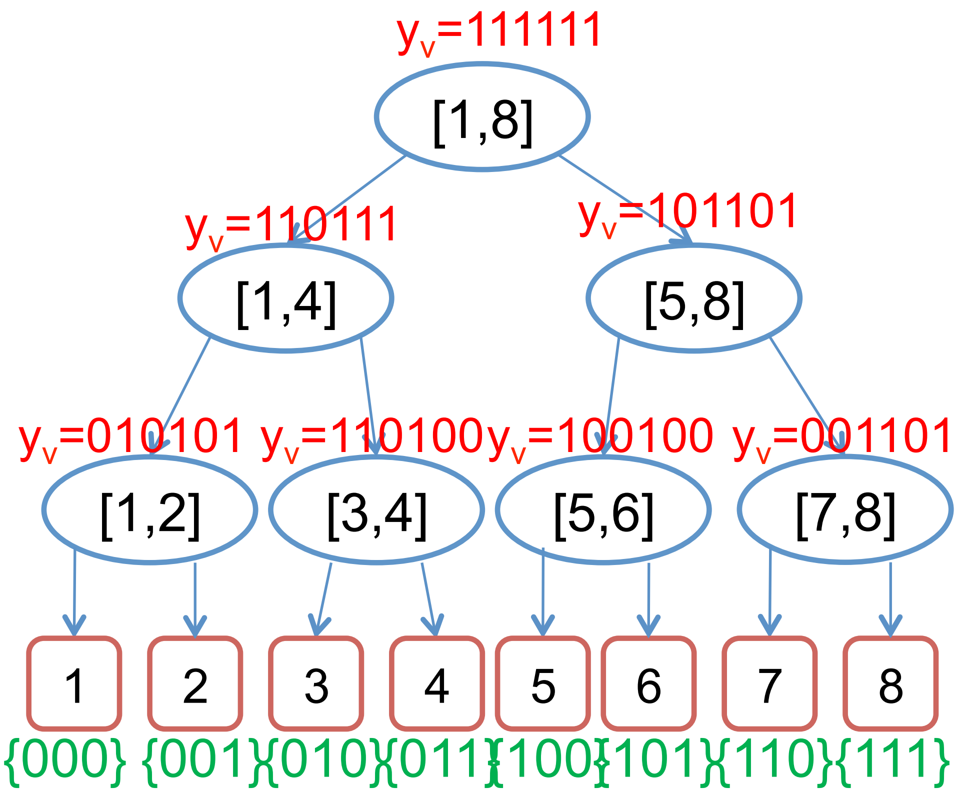
$$y_v=x_1 \vee x_2 \vee x_3 \vee x_4$$
$$=(1,1,1,1,0,0,1,1)$$

Binary tree over graphs



- Assign to each node v a bit arrays y_v
- y_v : bit array
  - i-th bit is 1 if graphs in an interval have the corresponding word.

Top-down traversal



- Q: bag-of-words of a query
- Perform top-down traversal
- Prune the search space if $\sum_{j \in Q} y_v[j] \le k$
- The larger k is, the smaller the search space is

Huge Memory

- Time is $O(\tau m)$: Very fast
  - τ: the number of traversed node
  - m: the number of bag-of-words in a query
- Space is $O(Mn\log n)$ bit
  - M: the number of unique words
  - n: the number of graphs

Wavelet Tree! (SODA,03)

- Replace y_v in each node v by a rank dictionary
  - explained in next slides
- Implement a tree without using pointers
- Only 60% memory overhead compared to the inverted index (Vigna,08)
- Access to the summary information in any internal node

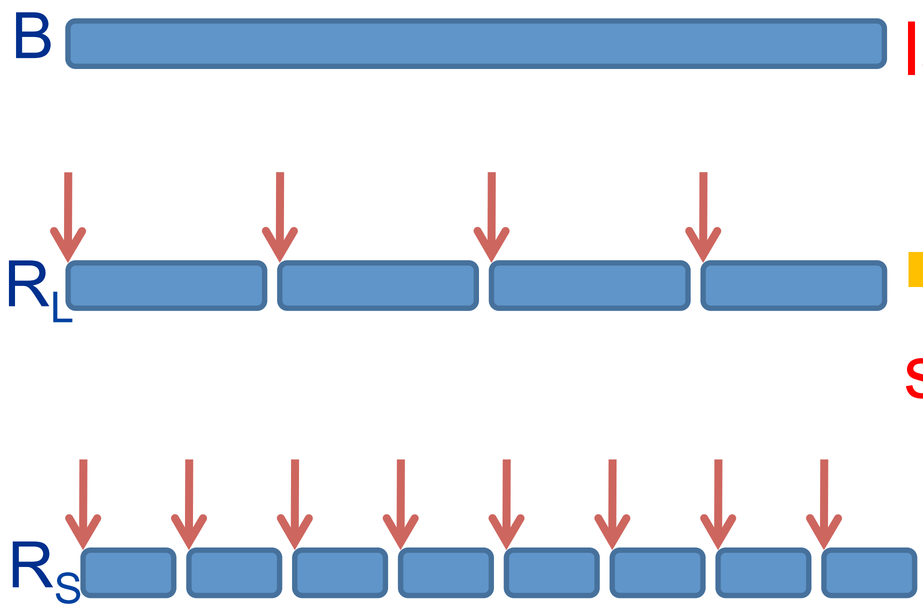
Rank dictionary (Raman,02)

- Give bit array B[1,n] the following operation:
  - $rank_c(B,i)$: return the number of $c \in \{0,1\}$ in B[1…i]

Ex) B=0110011100

i	1	2	3	4	5	6	7	8	9	10
$rank_1(B,8)=5$	0	1	1	0	0	1	1	1	0	0
$rank_0(B,5)=3$	0	1	1	0	0	1	1	1	0	0

Implementation of rank dictionary



- Divide the bit array B into large blocks of length $\ell=\log^2 n$

  R_L=Ranks of large blocks

- Divide each large block to small blocks of length $s=\log n/2$

  R_s=Ranks of small blocks relative to the large block

$rank_1(B,i)=R_L[i/\ell]+R_s[i/s]+(\text{remaining rank})$

Time:O(1)

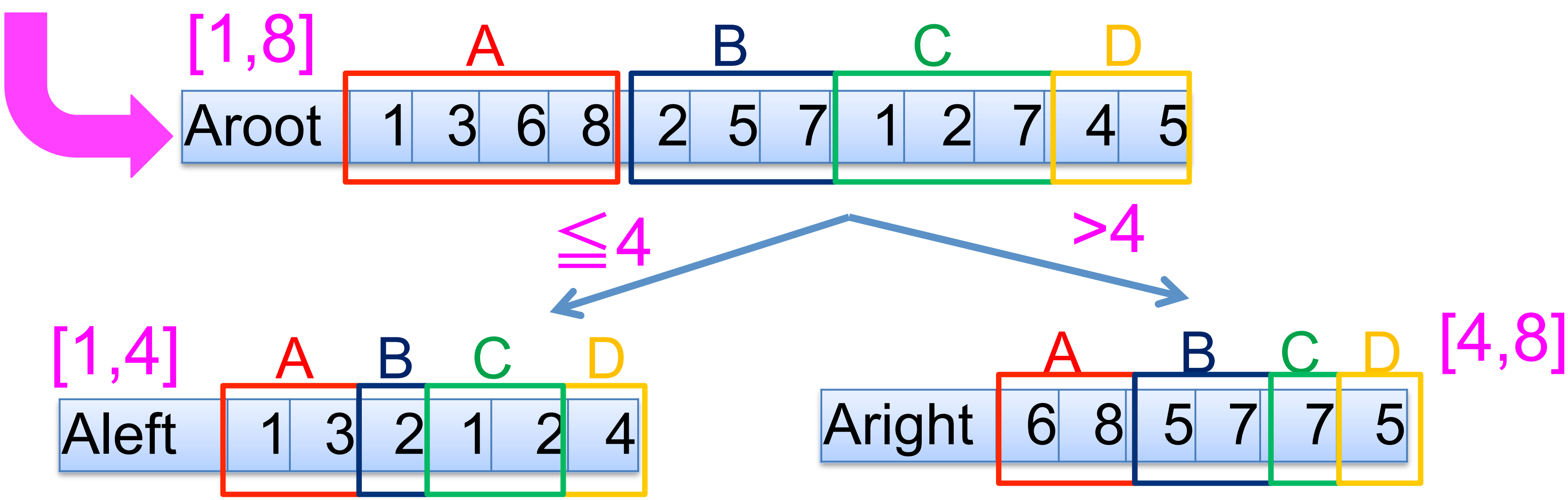
Memory: n +o(n) bits

Restricted inverted index

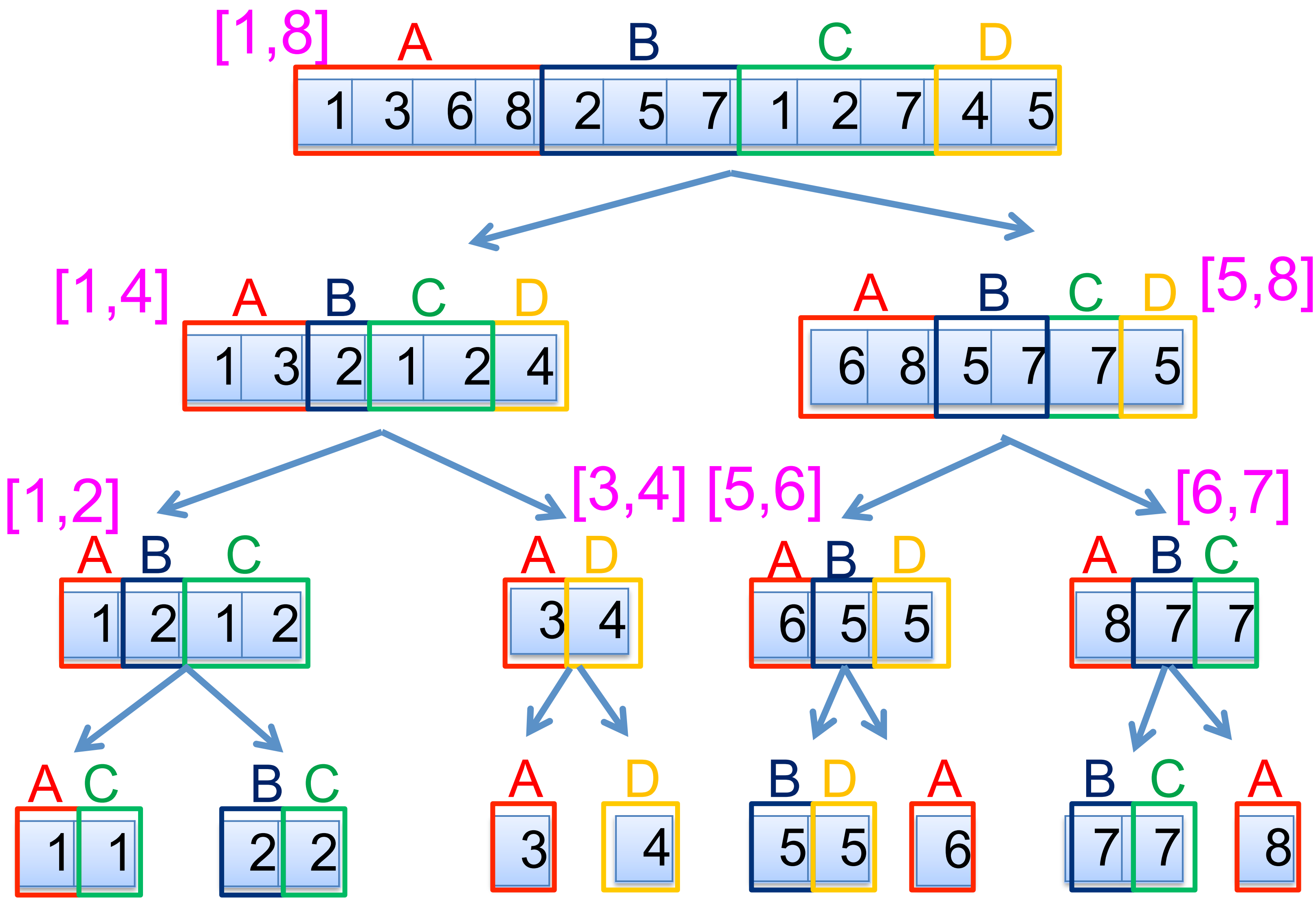
Inverted Index

Word	Graph ids
A	1,3,6,8
B	2,5,7
C	1,2,7
D	4,5

- Concatenate graph ids for words in the root
- Restrict the inverted index for the interval $[s_v,t_v]$ of a node v

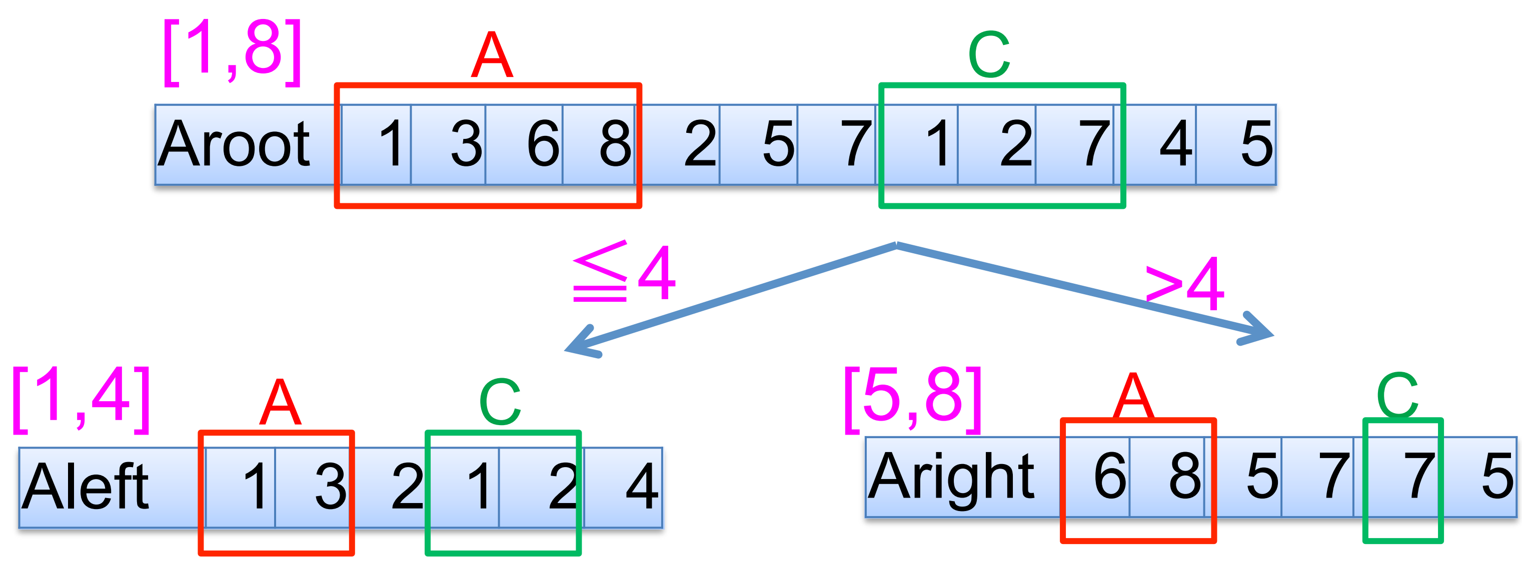


Whole structure of restricted inverted index



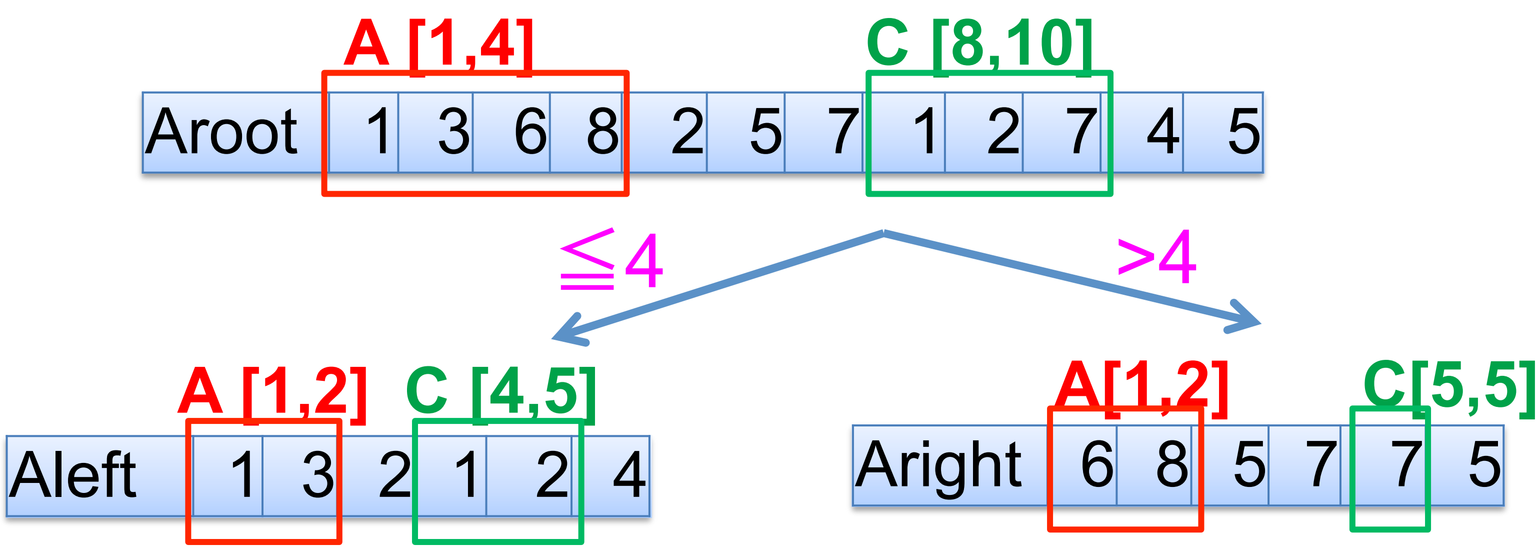
Similarity search

- To retrieve graphs similar to a query Q=(A,C), the tree is traversed in the top-down manner.



Similarity search

- To retrieve graphs similar to a query Q=(A,C), the tree is traversed in a top-down manner.
- Observation
  - To perform top-down traversal, only intervals of words in each node are necessary



Similarity search

- Replace restricted inverted index Av in each node v with a bit array bv.
  - bv[i]=1 if Av[i] goes to the right child

broot	0	0	1	1	0	1	1	0	0	1	0	1
Aroot	1	3	6	8	2	5	7	1	2	7	4	5

0

1

bleft	0	1	0	0	0	1
Aleft	1	3	2	1	2	4

bright	0	1	0	1	1	0
Aright	6	8	5	7	7	5

Similarity search

- Intervals of child nodes can be computed by rank operations

- $s_{left(v),j}=rank_0(bv,s_{vj}-1)+1, t_{left(v),j}=rank_0(bv,t_{vj})$
- $s_{right(v),j}=rank_1(bv,s_{vj}-1)+1, t_{right(v),j}=rank_1(bv,t_{vj})$

Ex)

C [8,10]

broot	0	0	1	1	0	1	1	0	0	1	0	1
Aroot	1	3	6	8	2	5	7	1	2	7	4	5

$rank_0(broot,8-1)+1=4$,
$rank_0(broot,10)=5$

$rank_1(broot,8-1)+1=5$,
$rank_1(broot,10)=5$

C [4,5]

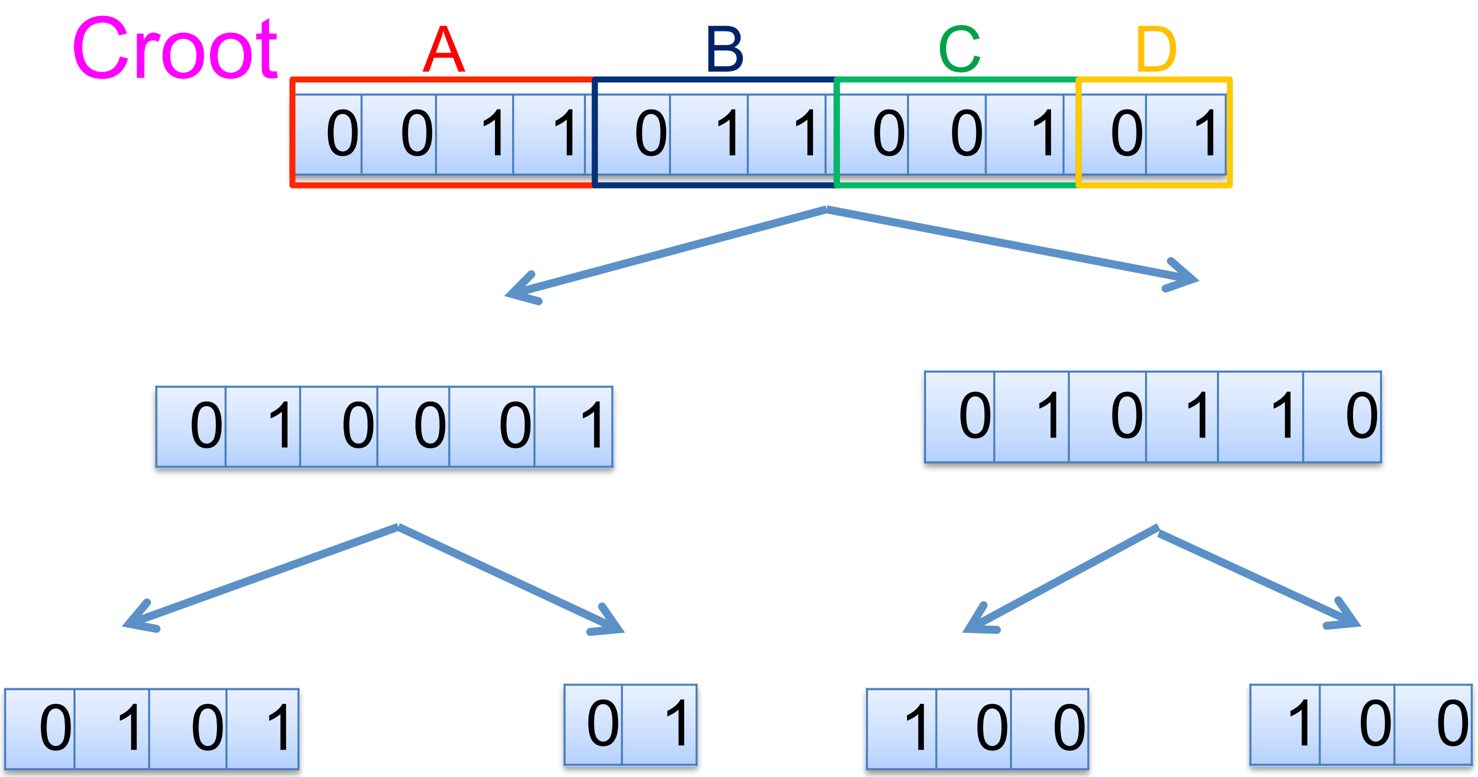
bleft	0	1	0	0	0	1
Aleft	1	3	2	1	2	4

C [5,5]

bright	0	1	0	1	1	0
Aright	6	8	5	7	7	5

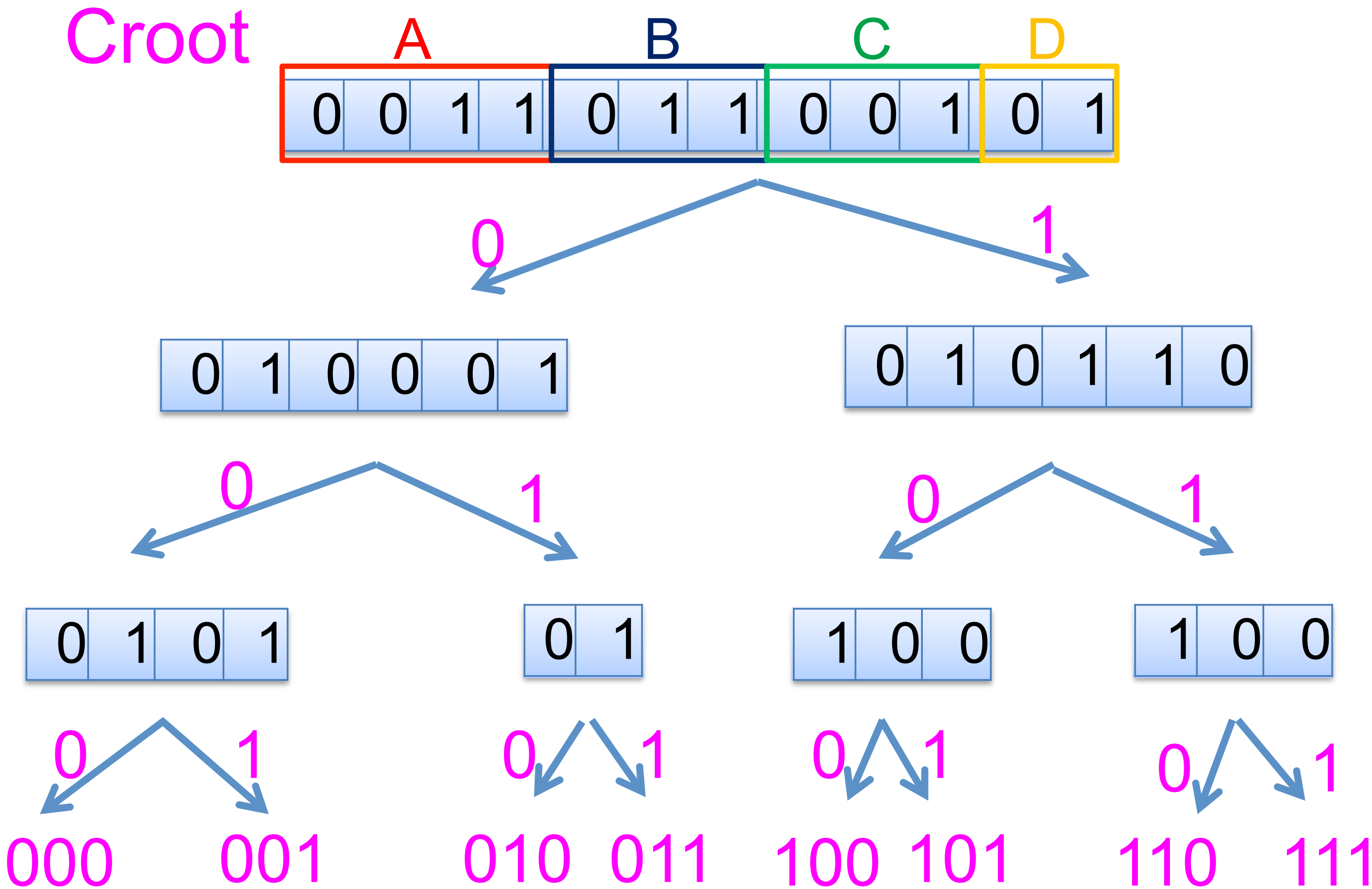
Wavelet Tree

- Wavelet tree can be obtained to replace the restricted Inverted indices with bit arrays
- Wavelet tree consists of bit arrays bv and initial intervals Croot.



Wavelet Tree

- Graph ids can be recovered from bit strings on the path from the root to leaves



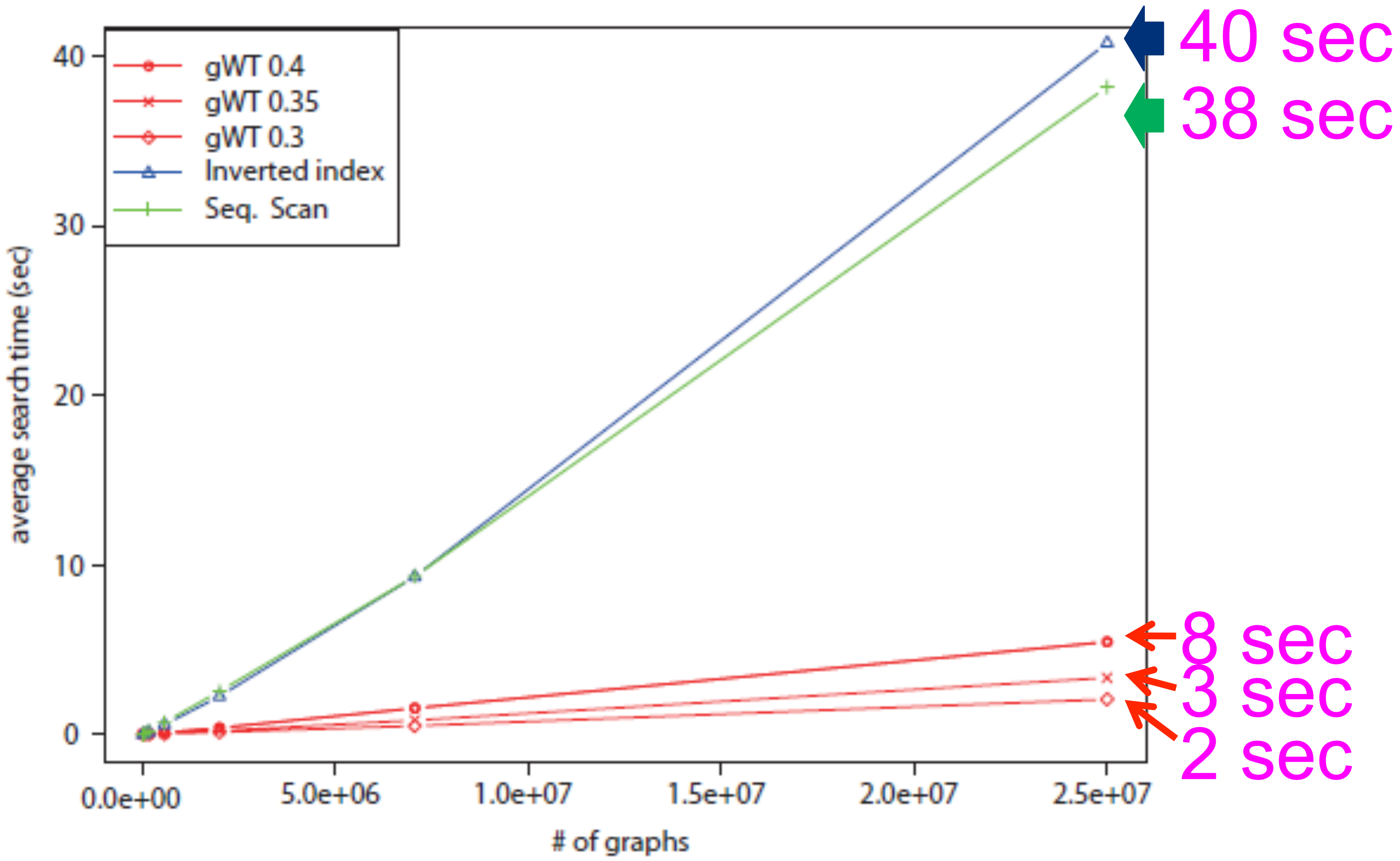
Memory

- $\underbrace{(1+\alpha)N\log n}_{\text{Bit arrays bv}} + \underbrace{M\log N}_{\text{Initial intervals Croot}}$ bits
  - N: the number of all words in the database
  - n: the number of graphs
  - α: overhead for rank dictionary (α=0.6)
- For inverted index, Nlogn bits
- About 60% overhead to inverted index!!

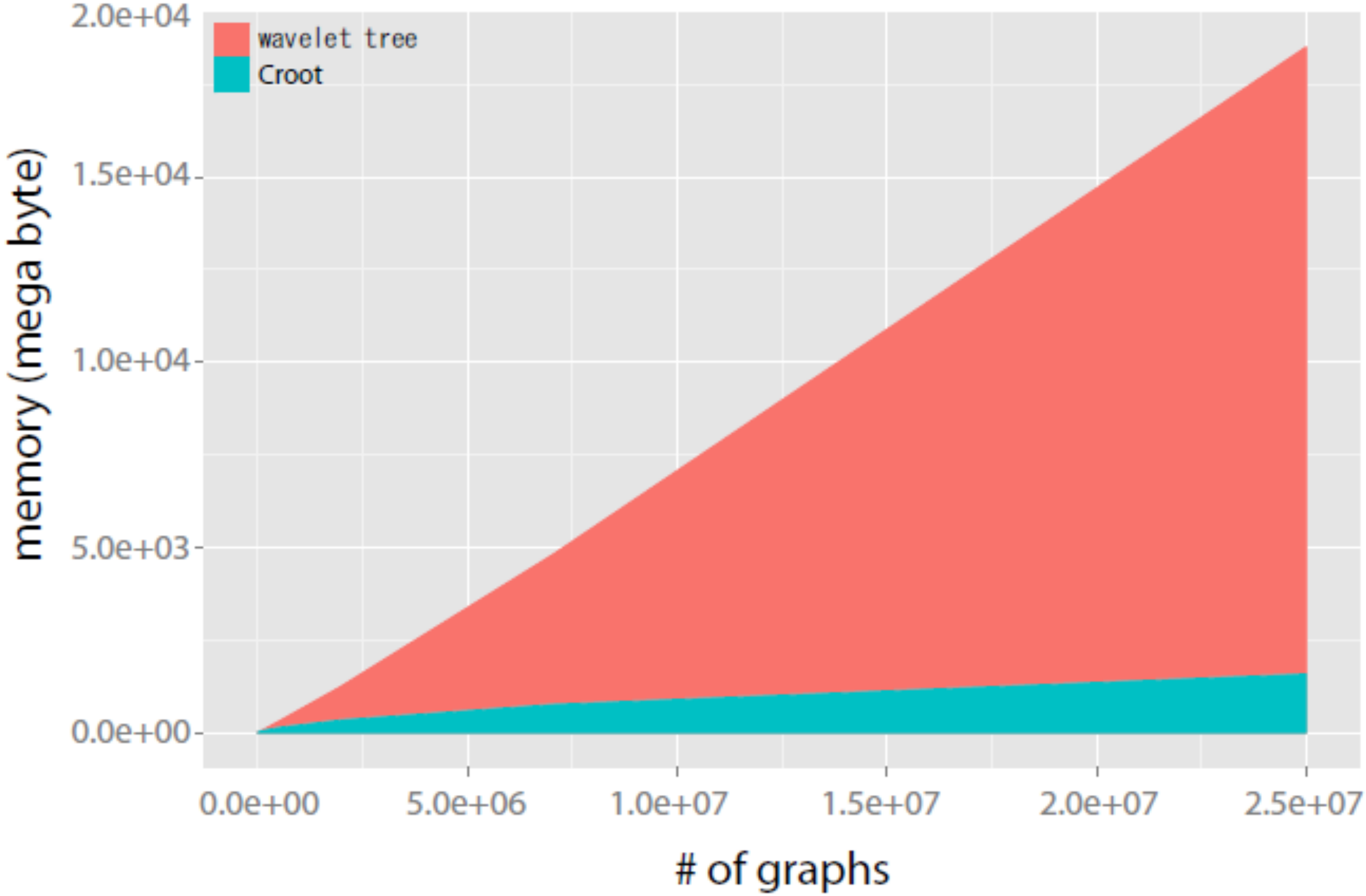
Experiments

- 25 million chemical compounds from PubChem database
- Use search time and memory as evaluation measures
- Compare our method gWT to
  - inverted index
  - sequential scan implemented in G-Hash (Wang et al, 2009)

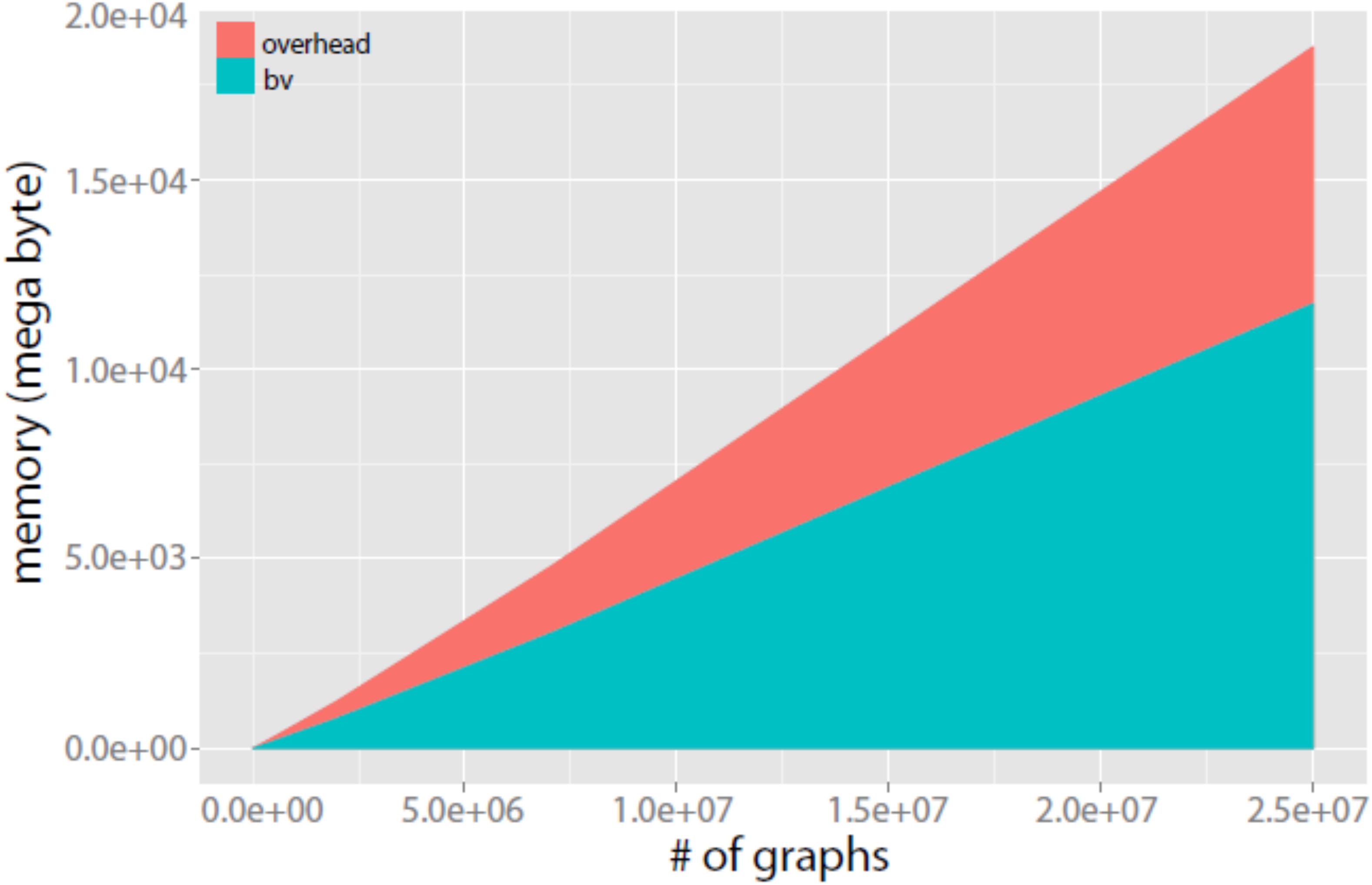
Search time on 25 million graphs



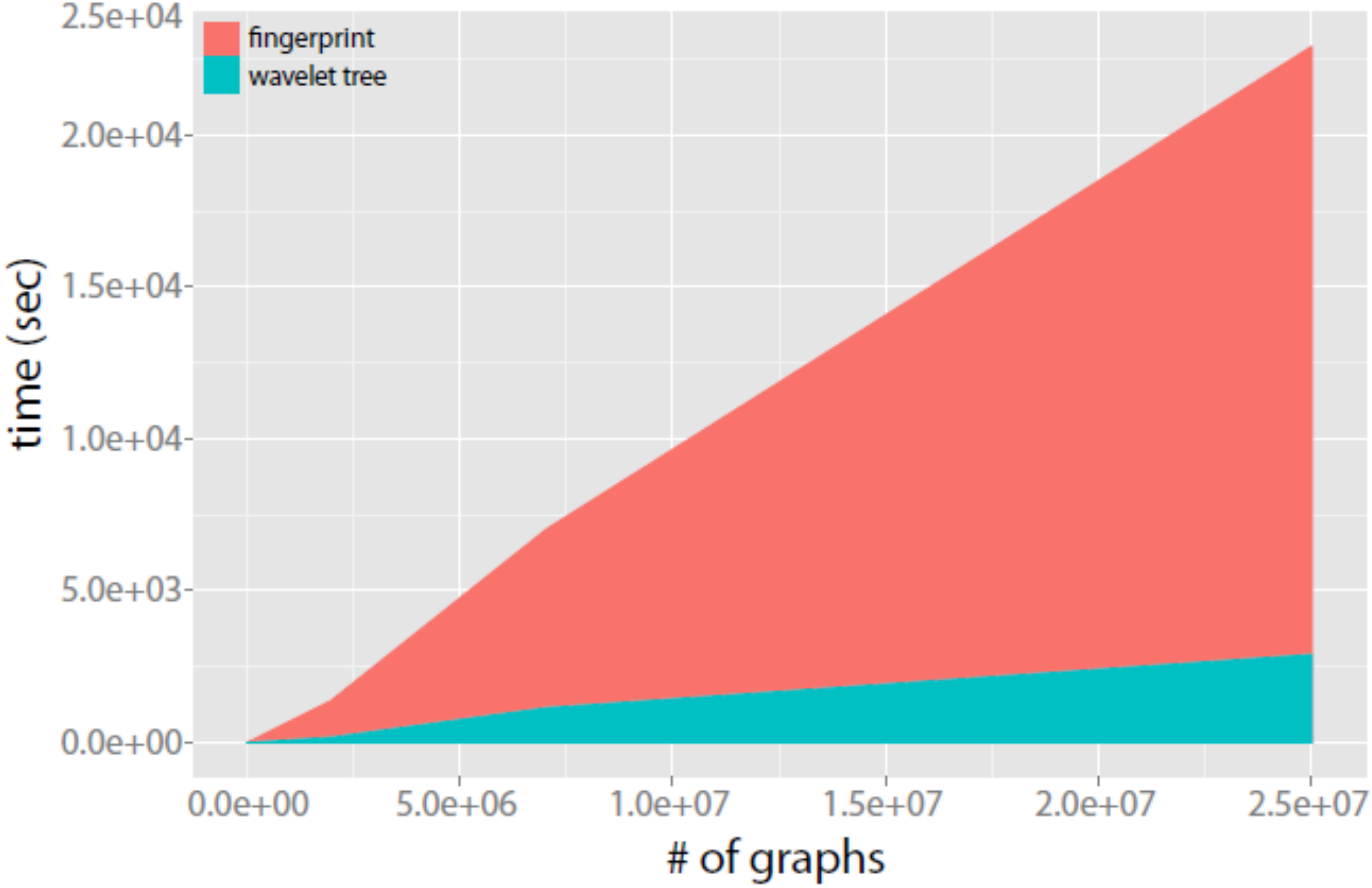
Memory usage



Overhead of rank dictionary



Construction time



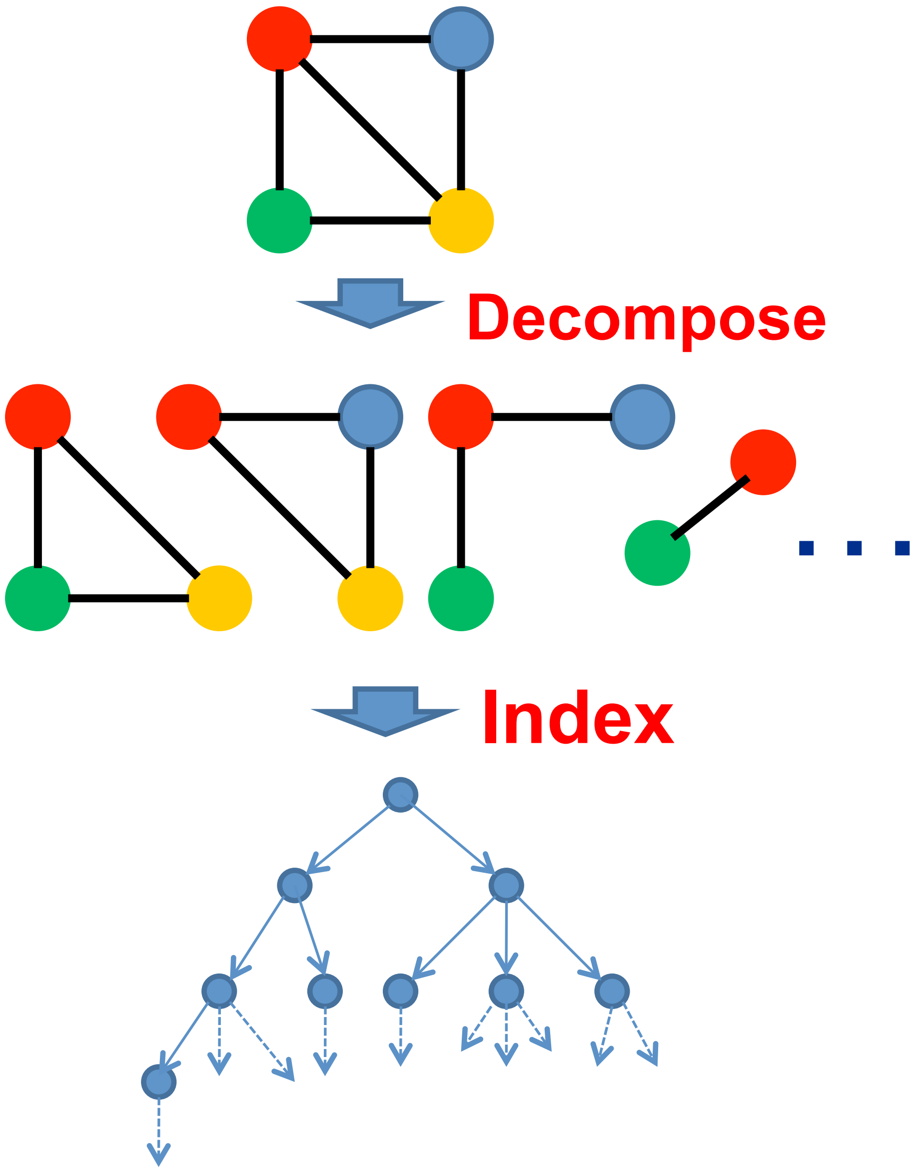
Related work

- A lot of methods have been proposed so far.

1. gIndex [Yan et al., 04]
2. Graph Grep [Shasha et al., 07]
3. Tree+Delta [Zhao et al., 07]
4. TreePi [Zhang et al., 07]
5. Gstring [Jiang et al., 07]
6. FG-Index [Cheng et al., 07]
7. GDIndex [Williams et al., 07]

etc

Related work



- Decompose graphs into a set of substructures
  - subgraphs, trees, paths etc
- Build a substructure-based index

Drawbacks

- Require frequent subgraph mining
- Do not scale to millions of graphs

Summary

- Efficient similarity search method for massive graph databases
- Solve semi-conjunctive query efficiently
- Built on wavelet trees
- Use Weisfeiler-Lehman procedure to convert graphs into bag-of-words
- Applicable to more than 20 million graphs
- Software

  http://code.google.com/p/gwt/

Acknowledgements

- Prof. Shin-ichi Minato (Hokkaido Univ.)
- Dr. Daisuke Okanohara (PFI)
- Members in ERATO Minato Project