ERATO合宿@札幌 2010年12月1日

ウェーブレット木による大規模フィンガープリントデータの高速類似度検索

田部井 靖生 (ERATO研究員)
共同研究者: 津田宏治

フィンガープリント

- フィンガープリント(FP) $W=\{w \mid w \in N\}$
  自然数w(ワード)を要素とする集合

例: W_1={1,3,5,8}, W_2={2,3,4,8,10}

- FPの固定長bit列表現 $x[j]=\begin{cases}1 & if \quad j \in W\\ 0 & else\end{cases}$

例:x_1=1010100100, x_2=0111000101

- 類似度 $K_N(W_1,W_2)=\frac{|W_1 \cap W_2|}{|W_1 \| W_2|}$

例: $K_N(W_1,W_2)=\frac{|\{3,8\}|}{\sqrt{|\{1,3,5,8\}|}\sqrt{|\{2,3,4,8,10\}|}}=\frac{2}{\sqrt{4}\times\sqrt{5}}=0.45$

フィンガープリントデータの類似度検索

- 入力: FPデーターベース W={W_1,W_2,…W_N} とクエリーFP Q
- 出力: クエリーFPQに対して類似度が1-ε以上のデーターベース中のFP

$I_N=\{i \mid K_N(Q,W_i) \geq 1-\varepsilon\}$

例:1-ε=0.8

クエリー Q={2,3,8}

	データーベース W	正規化類似度
⟷	W_1={1,5,8}	$K_N(Q,W_1)$=0.3
⟷	W_2={1,3,10}	$K_N(Q,W_2)$=0.3
⟷	W_3={1,2,3,8}	$K_N(Q,W_3)$=0.9
⟷	W_4={1,3,8,9,11}	$K_N(Q,W_4)$=0.5

既存手法 (その１) 線形探索

- データーベース中のFPに対して、クエリーFPとの類似度を線形に計算

クエリー Q={2,3,8} ⟷

データーベース W	正規化類似度
W_1={1,5,8}	$K_N(Q,W_1)$=0.3
W_2={1,3,10}	$K_N(Q,W_2)$=0.3
W_3={1,2,3,8}	.
W_4={1,3,8,9,11}	.

- O(nm)の時間(n: FPの数, m:ワードの種類数)

既存手法 (その２) 転置索引

- wordをkey, id listをvalueとしたハッシュ

id	word list
W_1	1,5,8
W_2	1,3
W_3	1,2,3,8
W_4	1,3,8,9

⇒

word	id list
1	W_1, W_2,W_3,W_4
2	W_3
3	W_2,W_3
8	W_3,W_4
9	W_4

- クエリーQ={2,3,8}に対する検索

word	id list
1	W_1, W_2,W_3,W_4
2	W_3
3	W_2,W_3
8	W_3,W_4
9	W_4

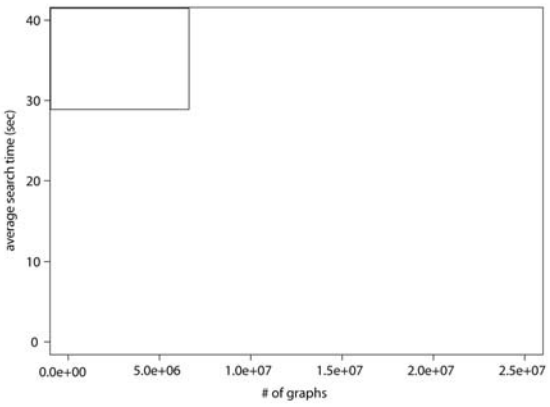
⇒ まとめる W_3,W_2,W_3,W_3,W_4

⇓ ソート

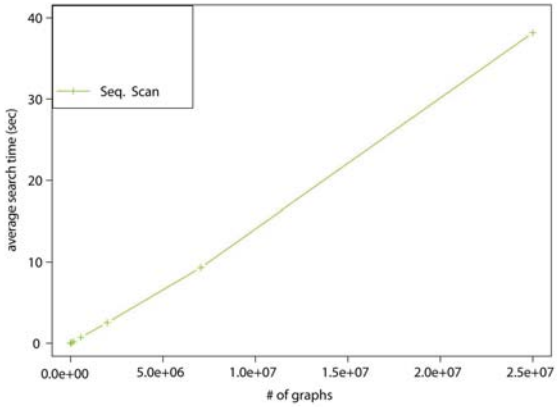
W_2,W_3,W_3,W_3,W_4

- 出力の数が多いときにソートに時間がかかる

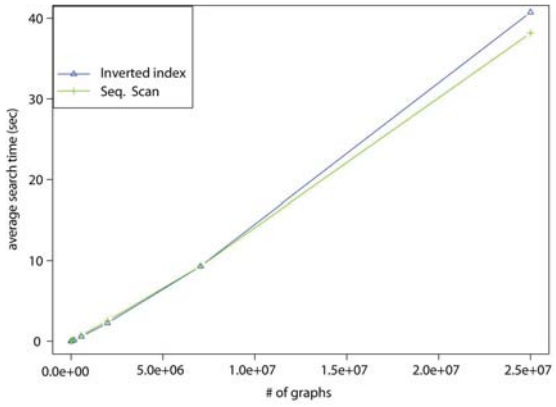
検索時間



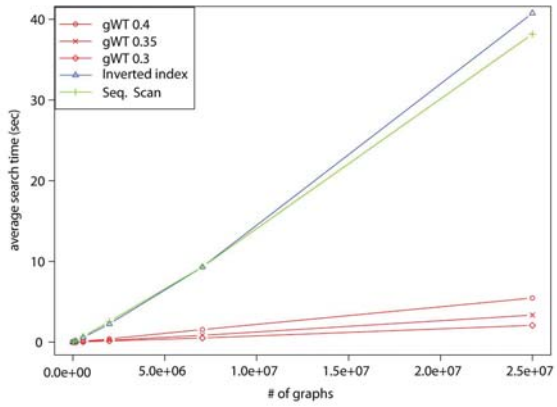
検索時間



検索時間



検索時間



概要

- ナイーブな方法(２分木索引構造)
- メモリ効率の良い索引構造
  (ウェーブレット木)

概要:2分木索引構造

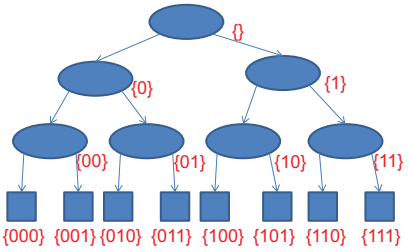
- 2分木を索引構造として利用
- 深さ優先に探索と類似度による探索空間の枝刈りを利用
- $I_N = \{i \mid K_N(Q, W_i) \geq 1-\varepsilon\}$に対する緩和解Iを検索

$I = \{i \mid \| Q \cap W_i \| \geq (1-\varepsilon)^2 \mid Q \mid\}$ $I_N \subseteq I$

-クエリQとデーターベース中のWi∈Wの共通要素数

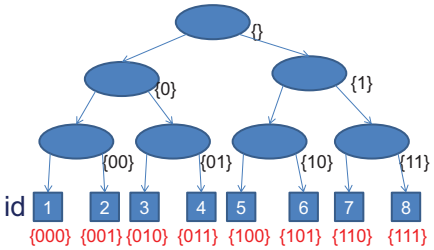
2分木索引構造

- 各ノードはビット列を表現



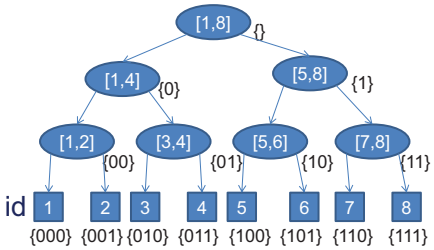
2分木索引構造

- 各ノードはビット列を表現
- 葉ノードvのビット列はidを表現 (id=int(v)+1)



2分木索引構造

- 各ノードはビット列を表現
- 葉ノードvのビット列はidを表現 (id=int(v)+1)
- 各ノードvは下位の葉のidに対する区間I_vを表現

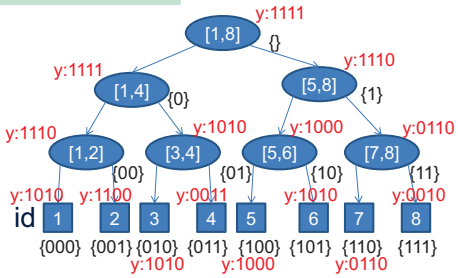


2分木索引構造

- 各ノードはビット列を表現
- 葉ノードvのビット列はidを表現 (id=int(v)+1)
- 各ノードvは下位の葉のidに対する区間I_vを表現
- 各ノードvは区間Ivのすべてのビット列の論理和を格納

$$y_v[j] = \bigvee_{i \in I_v} x_i[j]$$

i:x_i
1:1010
2:1100
3:1010
4:0011
5:1000
6:1010
7:0110
8:0010

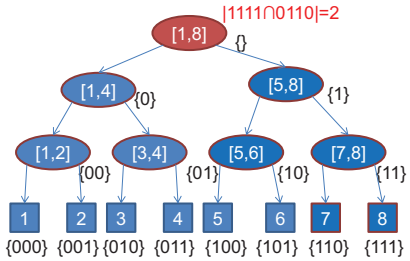


2分木索引構造

- クエリーFPQに対する検索は木の深さ優先で各ノードvでのy_v類似度を計算する

例)Q:0110を閾値$k=(1-\varepsilon)^2|Q|=2$で検索するとき

- ×：枝刈り

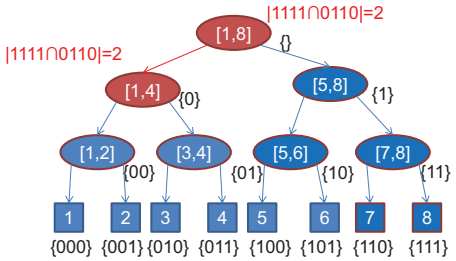


2分木索引構造

- クエリーFPqに対する検索は木の深さ優先

例)q:0110を閾値$k=(1-\varepsilon)^2|Q|=2$で検索するとき

- ×：枝刈り

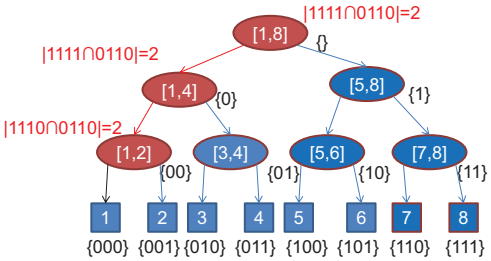


2分木索引構造

- クエリーFPqに対する検索は木の深さ優先

例)q:0110を閾値$k=(1-\varepsilon)^2|Q|=2$で検索するとき

- ×：枝刈り

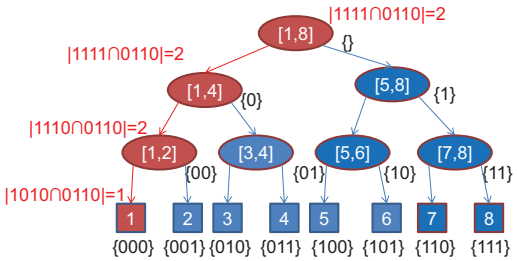


2分木索引構造

- クエリーFPqに対する検索は木の深さ優先

例)q:0110を閾値k=(1-ε)²|Q|=2で検索するとき

- ×：枝刈り

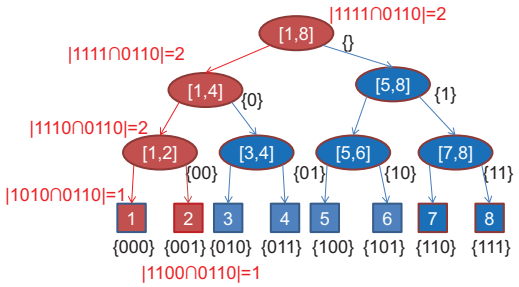


2分木索引構造

- クエリーFPqに対する検索は木の深さ優先

例)q:0110を閾値k=(1-ε)²|Q|=2で検索するとき

- ×：枝刈り

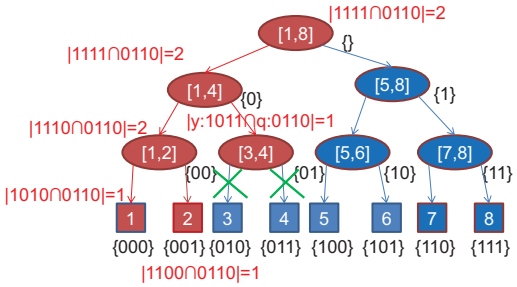


2分木索引構造

- クエリーFPqに対する検索は木の深さ優先

例)q:0110を閾値k=(1-ε)²|Q|=2で検索するとき

- ×：枝刈り

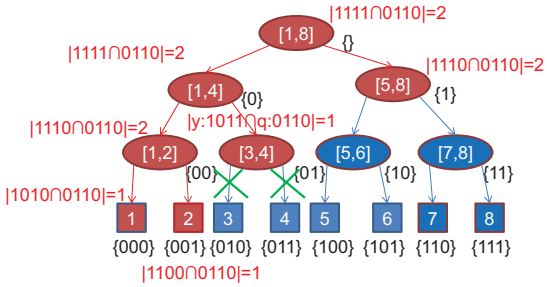


2分木索引構造

- クエリーFPqに対する検索は木の深さ優先

例)q:0110を閾値k=(1-ε)²|Q|=2で検索するとき

- ×：枝刈り

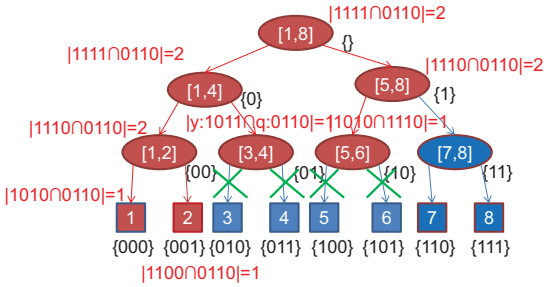


2分木索引構造

- クエリーFPqに対する検索は木の深さ優先

例)q:0110を閾値k=(1-ε)²|Q|=2で検索するとき

- ×：枝刈り

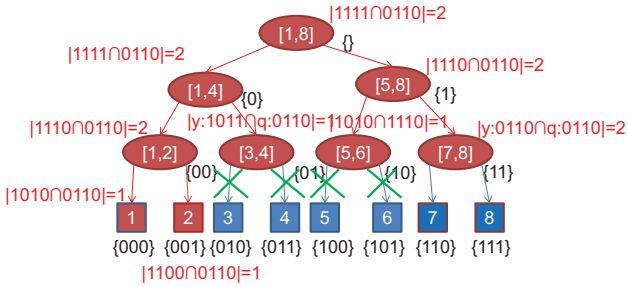


2分木索引構造

- クエリーFPqに対する検索は木の深さ優先

例)q:0110を閾値k=(1-ε)²|Q|=2で検索するとき

- ×：枝刈り

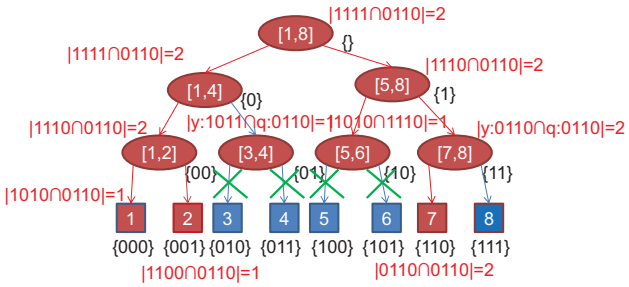


2分木索引構造

- クエリーFPqに対する検索は木の深さ優先

例)q:0110を閾値k=(1-ε)²|Q|=2で検索するとき

- ×：枝刈り

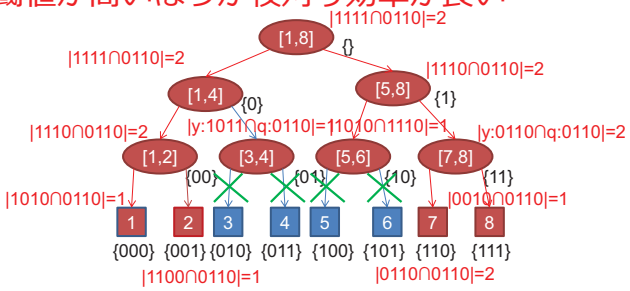


2分木索引構造

- クエリーFPqに対する検索は木の深さ優先

例)q:0110を閾値k=(1-ε)²|Q|=2で検索するとき

- ×：枝刈り
- I={7}
- 閾値が高いほうが枝刈り効率が良い



利点と欠点

- 利点:各ノードvの区間Ivの FPに対応する論理和$y_v[j]$にアクセス可能

→効率的な検索可能

- 欠点: O(Mnlogn)のメモリー

M:ユニークなワード数

n:FPの数

概要:ウェーブレット木

- ノード v での区間I_vでのidでのFPのy_vメモリを削減するためのビット列上のRank辞書を使う

- 効率性を保ちながらO(Mnlogn)メモリからO(Nlogn)メモリに削減可能

n: データーベース中のFPの数

N:データーベー中の全FPの数 $N = \sum_{i=1}^{n} |W_i|$

Rank辞書

- ビット列B[1,n]に対し次の操作を備える

- $Rank_c(B,i)$:B[1…i]中のc∈{0,1}の数を返す

例: B=011001110

$Rank_1(B,7)=4$

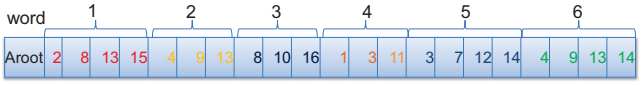
$Rank_0(B,5)=3$

i	123456789
	011001110

- O(1)時間、0.63のメモリーオーバーヘッドで実装可能 (Vigna, 08)

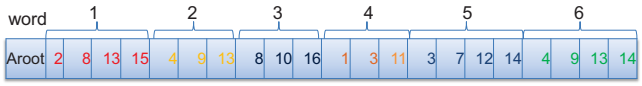
木の各ノードに制限をかけた転置索引

- 木の各ノードvにおいて,区間Ivのidから転置索引Z_{vj}を作る $Z_{vj} = \{i \mid x_i[j]=1, i \in I_v\}$



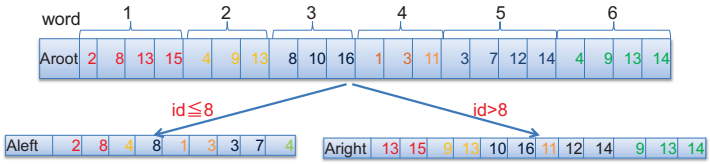
木の各ノードに制限をかけた転置索引

- 木の各ノードvにおいて,区間Ivのidから転置索引Z_{vj}を作る $Z_{vj}=\{i \mid x_i[j]=1, i \in I_v\}$
- 各ノードvはZ_{vj}の連結Avを保持



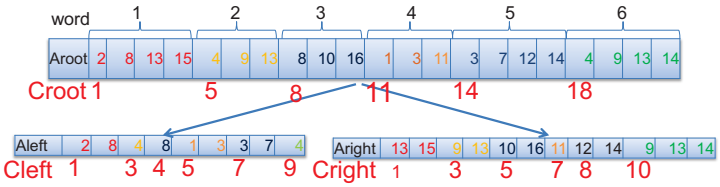
木の各ノードに制限をかけた転置索引

- 木の各ノードvにおいて,区間Ivのidから転置索引Z_{vj}を作る $Z_{vj}=\{i \mid x_i[j]=1, i \in I_v\}$
- 各ノードvはZ_{vj}の連結Avを保持
- 各ノードで中央値でidを分割して、子に対する転置索引を作成



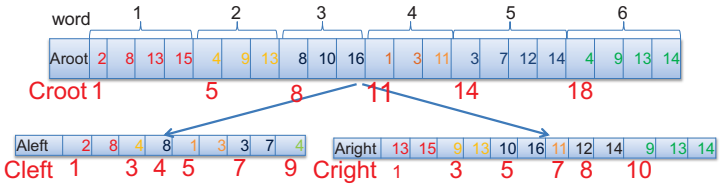
木の各ノードに制限をかけた転置索引

- 各ノードはAvにおけるZ_{vj}の開始点$C_v[j]$を保持



木の各ノードに制限をかけた転置索引

- 各ノードはAvにおけるZ_{vj}の開始点$C_v[j]$を保持
- クエリー$Q=\{q_1,q_2,\ldots,q_m\}$の各要素q_jに対し区間$I_v=[s_{vj},t_{vj}]=[C_v[j], C_v[j+1]-1]$ $(j \in Q)$で計算
- $s_{vj}>t_{vj}$または葉に到達したら終了

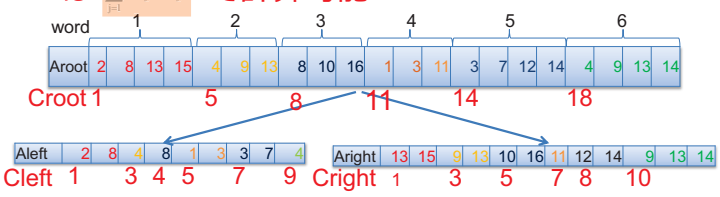


木の各ノードに制限をかけた転置索引

- 各ノードはAvにおけるZ_{vj}の開始点$C_v[j]$を保持
- クエリー$Q=\{q_1,q_2,\ldots,q_m\}$の各要素q_jに対し区間$I_v=[s_{vj},t_{vj}]=[C_v[j], C_v[j+1]-1]$で計算
- $s_{vj}>t_{vj}$または葉に到達したら終了

例: $[s_{root,1},t_{root,1}]=[C_{root}[1],C_{root}[2]-1]=[1,5-1]=[1,4]$

- 各ノードvでの区間IvとクエリーQとの類似度は $\sum_{j=1}^{m} I[t_j > s_j]$ で計算可能



ウェーブレット木

(Grossi et al., SODA'03)

- 木の各ノードvで区間$C_v[j]=[s_{vj},t_{vj}]$さえ、計算できれば明示的にAvをもたなくても検索できる

→ Rank辞書で計算

ウェーブレット木
(Grossi et al., SODA'03)

- 木の各ノードvで区間$C_v[j]=[s_{vj},t_{vj}]$さえ、計算できれば明示的にAvをもたなくても検索できる

→ Rank辞書で計算

- bv[k]: Avのk番目のエントリーが左または右の子へ行くかを指し示すビット列　0⇒左, 1⇒右

$$b_v[k]=\begin{cases}1 & A_v[k]>\lfloor (i+i')/2 \rfloor \\ 0 & A_v[k]\le \lfloor (i+i')/2 \rfloor\end{cases}$$

Aroot	2	8	13	15	4	9	13	8	10	16	1	3	11	3	7	12	14	4	9	13	14
broot	0	0	1	1	0	1	1	0	1	1	0	0	1	0	0	1	1	0	1	1	1

0　　1

Aleft	2	8	4	8	1	3	3	7	4
bleft	0	1	0	1	0	0	0	1	0

Aright	13	15	9	13	10	16	11	12	14	9	13	14
bright	1	1	0	1	0	1	0	0	0	0	1	1

ウェーブレット木
(Grossi et al., SODA'03)

- ノードvの左右の子left(v), right(v)に対する区間$[s_{left(v),j},t_{left(v),j}],[s_{right(v),j},t_{right(v),j}]$は$b_v$はrank操作で求めることができる。

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

例) I_{root}=[1,4]の左/右の子に対する区間I_{left},I_{child}は

I_{left}=[$rank_0$(bv,0),$rank_1$(bv,1)]=[1,2]

I_{right}=[$rank_1$(bv,1),$rank_1$(bv,1)]=[1,2]

Aroot	2	8	13	15	4	9	13	8	10	16	1	3	11	3	7	12	14	4	9	13	14
broot	0	0	1	1	0	1	1	0	1	1	0	0	1	0	0	1	1	0	1	1	1

Aleft	2	8	4	8	1	3	3	7	4
bleft	0	1	0	1	0	0	0	1	0

Aright	13	15	9	13	10	16	11	12	14	9	13	14
bright	1	1	0	1	0	1	0	0	0	0	1	1

探索空間の枝刈り

Lemma

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

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

- 探索空間の枝刈り可能

検索アルゴリズム

入力:クエリーFPQ={$W_1,W_2,\ldots,W_m$}

i)v=rootのとき

$[s_{rootj},t_{rootj}]=[C_{root}[j], C_{root}[j+1]-1](j\in Q)$

ii)v≠rootのとき

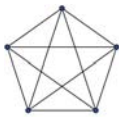
$$s_{left(v),j}=rank_0(b_v,s_{vj}-1)+1$$
$$t_{left(v),j}=rank_0(b_v,t_{vj})$$
$$s_{right(v),j}=rank_1(b_v,s_{vj}-1)+1$$
$$t_{right(v),j}=rank_1(b_v,t_{vj})$$

iii)v=葉または$\sum_{j=1}^{m} I[t_j>s_j]<k$ またはlemma不成立で探索終了

そうでなければ(ii)へ

実験

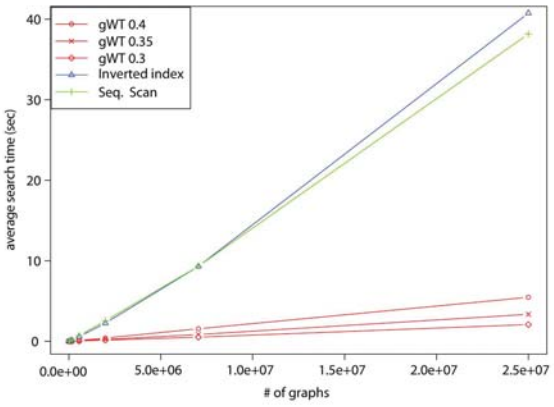
- PubChemデーターベースから25,000,000化合物
- Weisfeiler and Lehman Kernel (NIPS,2009)によりフィンガープリントを作成



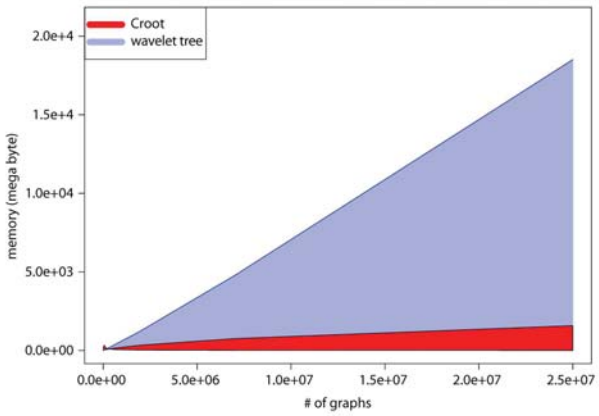
(4,8,12,16,20,…)

- 線形探索、転置索引と比較

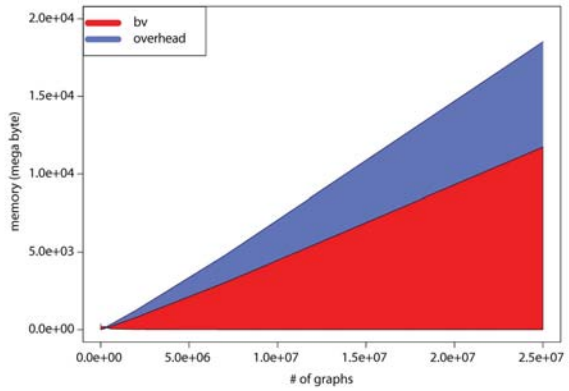
検索速度



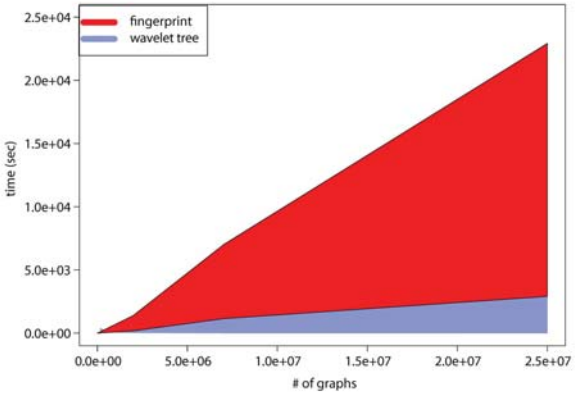
メモリ



Rank辞書のメモリ



構築時間



まとめ

- 大規模フィンガープリントデータに対する高速類似度検索手法
- 線形探索、転置索引よりも高速
- 25,000,000フィンガープリントに対しても適応可能
- Top-kを求めることも可能
- いろいろな応用
  - フィンガープリントデーターなら何でも応用可能