



Title	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
Author(s)	田部井, 靖生; 津田, 宏治
Citation	2010年度科学技術振興機構ERATO湊離散構造処理系プロジェクト講究録. p.468-475.
Issue Date	2011-06
Doc URL	<a href="http://hdl.handle.net/2115/48334">http://hdl.handle.net/2115/48334</a>
Type	conference presentation
Note	ERATO湊離散構造処理系プロジェクト: 2010年度初冬のワークショップ (ERATO合宿). 2010年11月29日 (月) ~ 12月1日 (水). 札幌北広島クラッセホテル.
File Information	23.tabei.02.pdf



[Instructions for use](#)

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)  $W = \{w \mid w \in \mathbb{N}\}$   
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  $x[j] = \begin{cases} 1 & \text{if } j \in W \\ 0 & \text{else} \end{cases}$

例:  $x_1 = 1010100100$ ,  $x_2 = 0111000101$

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

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データベース  $W = \{W_1, W_2, \dots, W_N\}$   
とクエリーFP  $Q$
- 出力: クエリーFP  $Q$  に対して類似度が  
 $1 - \epsilon$  以上のデータベース中のFP

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

例:  $1 - \epsilon = 0.8$

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

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

- $O(nm)$ 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	$W_4$

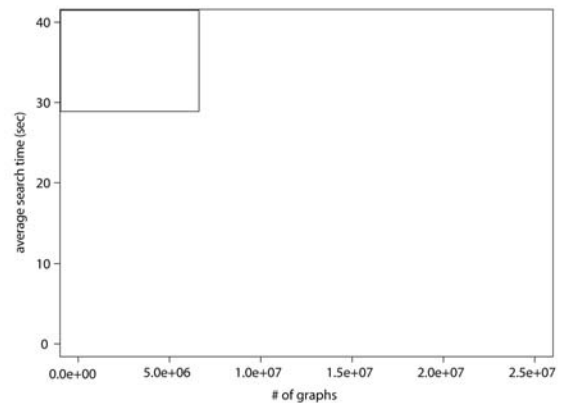
まとめる →  $W_3, W_2, W_3, W_3, W_4$

↓ ソート

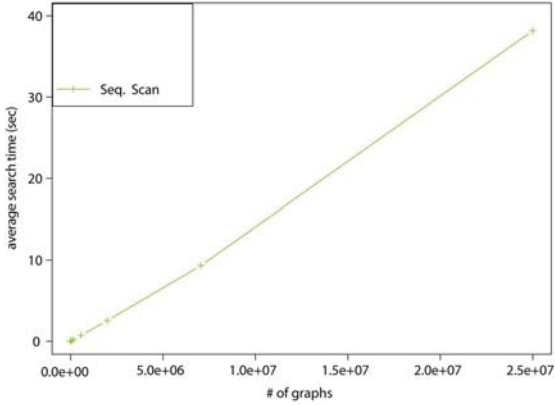
$W_2, W_3, W_3, W_3, W_4$

- 出力の数が多いときにソートに時間がかかる

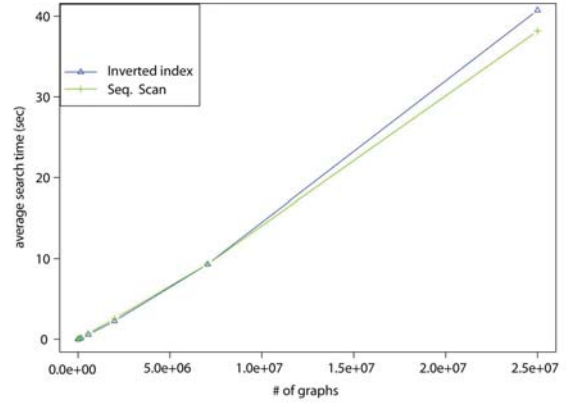
## 検索時間



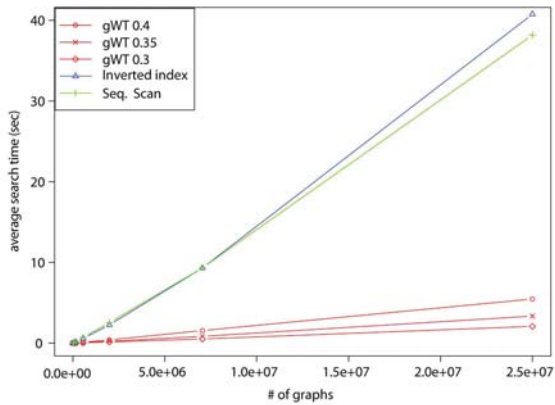
## 検索時間



## 検索時間



## 検索時間



## 概要

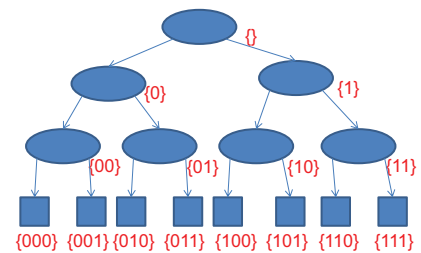
- ナイーブな方法(2分木索引構造)
- メモリ効率の良い索引構造 (ウェーブレット木)

## 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
  - $I_N = \{i \mid K_N(Q, W_i) \geq 1 - \epsilon\}$  に対する緩和解  $I$  を検索
  - $I = \{i \mid |Q \cap W_i| \geq (1 - \epsilon)^2 |Q|\}$   $I_N \subseteq I$
- クエリ  $Q$  とデータベース中の  $W_i \in W$  の共通要素数

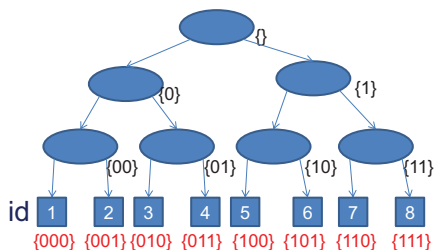
## 2分木索引構造

- 各ノードはビット列を表現



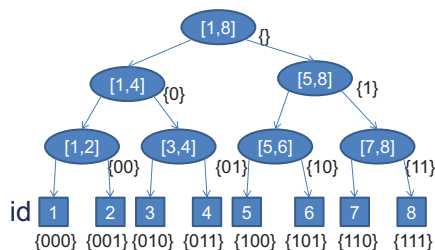
## 2分木索引構造

- 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

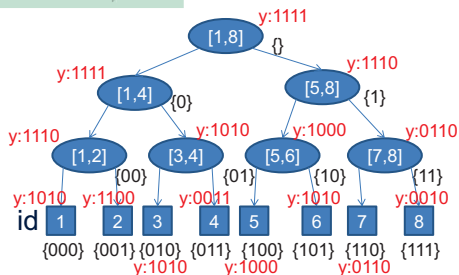


## 2分木索引構造

- 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 $I_v$ のすべてのビット列の論理和を格納

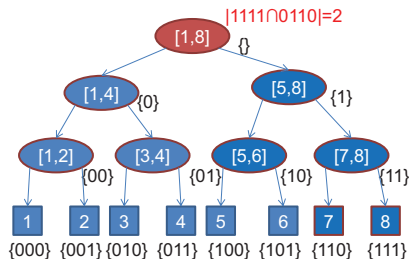
$$y_v[j] = \bigvee_{i \in I_v} x_i[j]$$

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



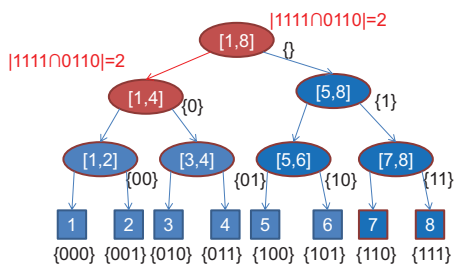
## 2分木索引構造

- クエリーFP $q$ 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 $k=(1-\epsilon)^2|Q|=2$ で検索するとき
- ✕ : 枝刈り



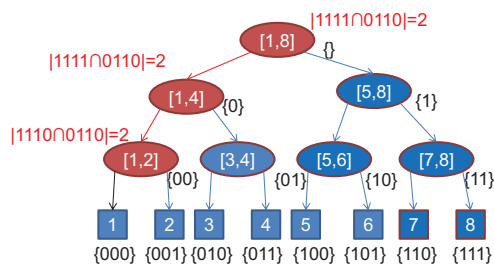
## 2分木索引構造

- クエリーFP $q$ に対する検索は木の深さ優先
- 例)  $q:0110$ を閾値 $k=(1-\epsilon)^2|Q|=2$ で検索するとき
- ✕ : 枝刈り



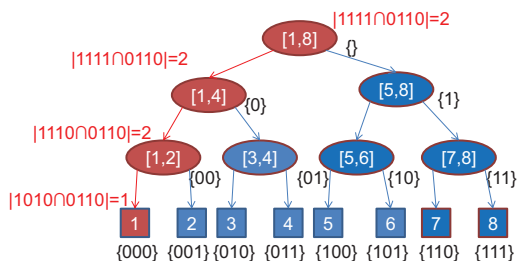
## 2分木索引構造

- クエリーFP $q$ に対する検索は木の深さ優先
- 例)  $q:0110$ を閾値 $k=(1-\epsilon)^2|Q|=2$ で検索するとき
- ✕ : 枝刈り



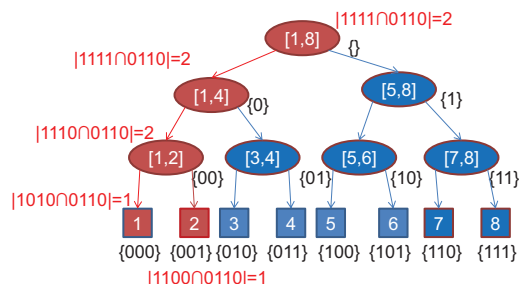
## 2分木索引構造

- クエリーFP $q$ に対する検索は木の深さ優先例)q:0110を閾値 $k=(1-\epsilon)^2|Q|=2$ で検索するとき
- ✕ : 枝刈り



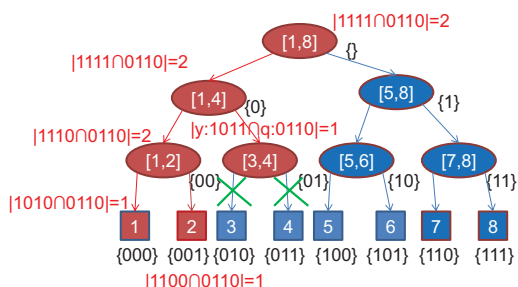
## 2分木索引構造

- クエリーFP $q$ に対する検索は木の深さ優先例)q:0110を閾値 $k=(1-\epsilon)^2|Q|=2$ で検索するとき
- ✕ : 枝刈り



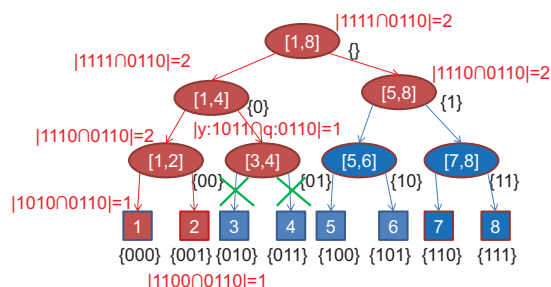
## 2分木索引構造

- クエリーFP $q$ に対する検索は木の深さ優先例)q:0110を閾値 $k=(1-\epsilon)^2|Q|=2$ で検索するとき
- ✕ : 枝刈り



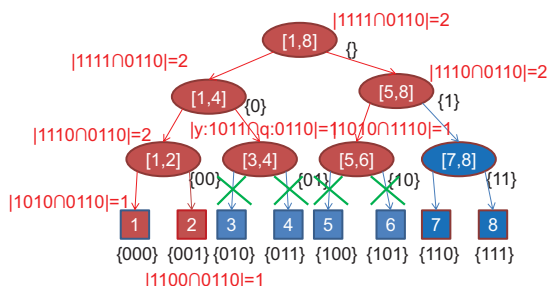
## 2分木索引構造

- クエリーFP $q$ に対する検索は木の深さ優先例)q:0110を閾値 $k=(1-\epsilon)^2|Q|=2$ で検索するとき
- ✕ : 枝刈り



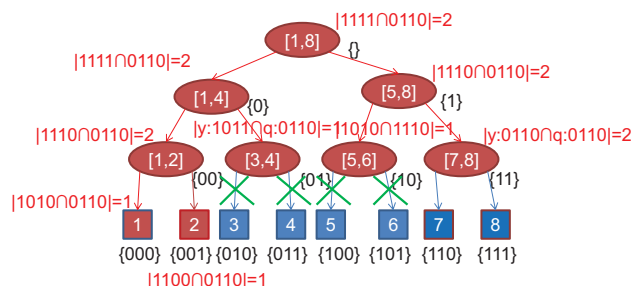
## 2分木索引構造

- クエリーFP $q$ に対する検索は木の深さ優先例)q:0110を閾値 $k=(1-\epsilon)^2|Q|=2$ で検索するとき
- ✕ : 枝刈り



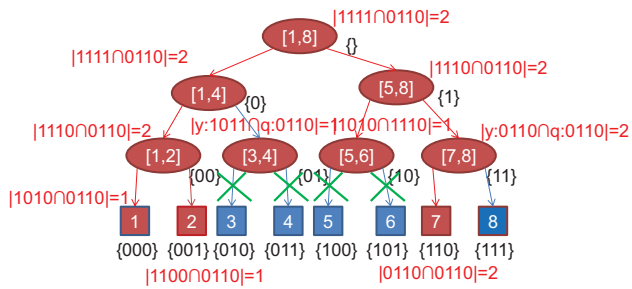
## 2分木索引構造

- クエリーFP $q$ に対する検索は木の深さ優先例)q:0110を閾値 $k=(1-\epsilon)^2|Q|=2$ で検索するとき
- ✕ : 枝刈り



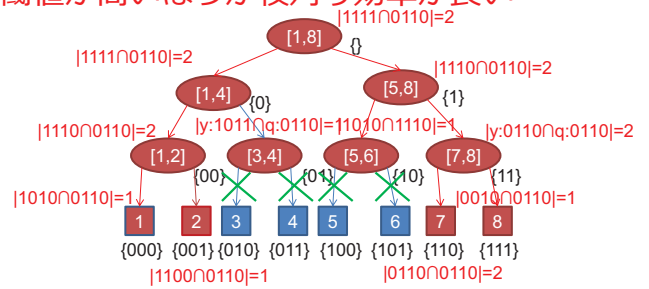
## 2分木索引構造

- クエリーFP $q$ に対する検索は木の深さ優先例) $q:0110$ を閾値 $k=(1-\epsilon)^2|Q|=2$ で検索するとき
- ×: 枝刈り



## 2分木索引構造

- クエリーFP $q$ に対する検索は木の深さ優先例) $q:0110$ を閾値 $k=(1-\epsilon)^2|Q|=2$ で検索するとき
- ×: 枝刈り
- $l=\{7\}$
- 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 $I_v$ 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:  $O(Mn \log n)$ 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 $y_v$ メモリー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  $O(Mn \log n)$ メモリーから  $O(N \log n)$ メモリーに削減可能
- $n$ : データベース中のFPの数
- $N$ : データベース中の全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
- $\text{Rank}_c(B,i)$ :  $B[1..i]$ 中の $c \in \{0,1\}$ の数を返す

例:  $B=011001110$

$\text{Rank}_1(B,7)=4$       $\begin{array}{c|cccccccc} i & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array}$   
 $\text{Rank}_0(B,5)=3$       $\begin{array}{c|cccccccc} & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \end{array}$

- $O(1)$ 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 $I_v$ のidから転置索引 $Z_{v,j}$ を作る  $Z_{v,j} = \{i \mid x_i[j]=1, i \in I_v\}$



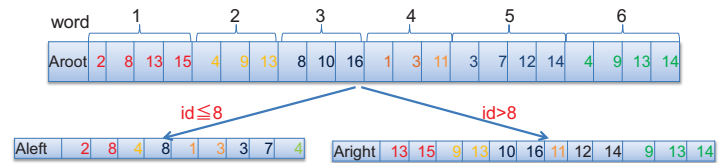
### 木の各ノードに制限をかけた転置索引

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



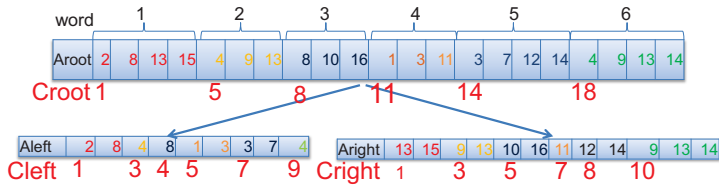
### 木の各ノードに制限をかけた転置索引

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



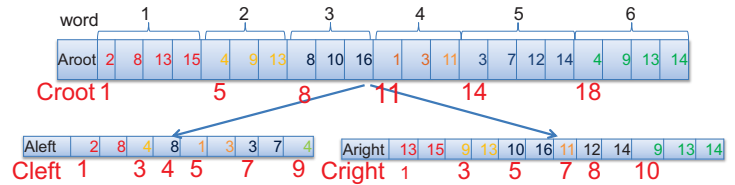
### 木の各ノードに制限をかけた転置索引

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

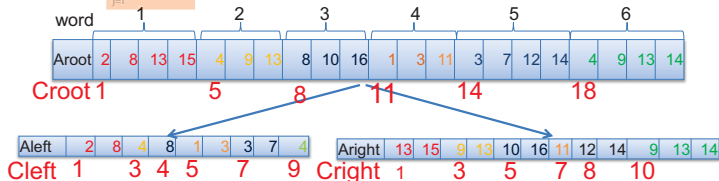


### 木の各ノードに制限をかけた転置索引

- 各ノードは $Av$ における $Z_{v,j}$ の開始点 $C_{v,j}$ を保持
- クエリー $Q = \{q_1, q_2, \dots, q_m\}$ の各要素 $q_j$ に対し区間 $I_v = [s_{v,j}, t_{v,j}] = [C_{v,j}, C_{v,j+1} - 1]$ で計算
- $s_{v,j} > t_{v,j}$ 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 $I_v$ とクエリー $Q$ との類似度は  $\sum_{j=1}^m \mathbb{1}[t_j > s_j]$  で計算可能



### ウェーブレット木

(Grossi et al., SODA'03)

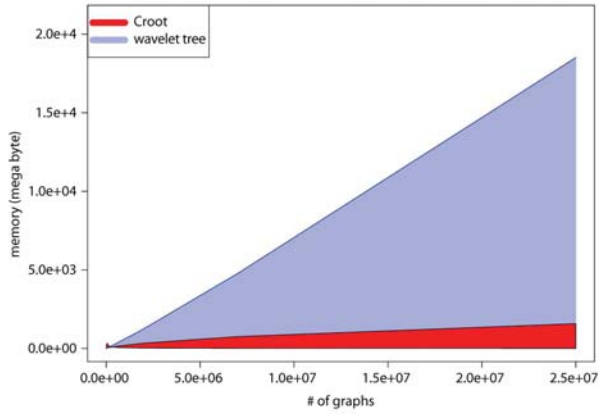
- 木の各ノード $v$ で区間 $C_{v,j} = [s_{v,j}, t_{v,j}]$ さえ、計算できれば明示的に $Av$ をもたなくても検索できる

→ Rank辞書で計算

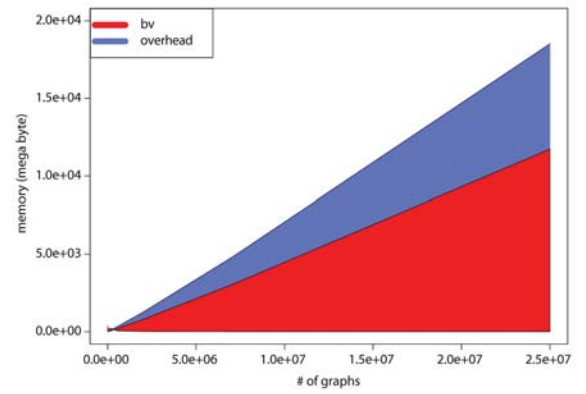




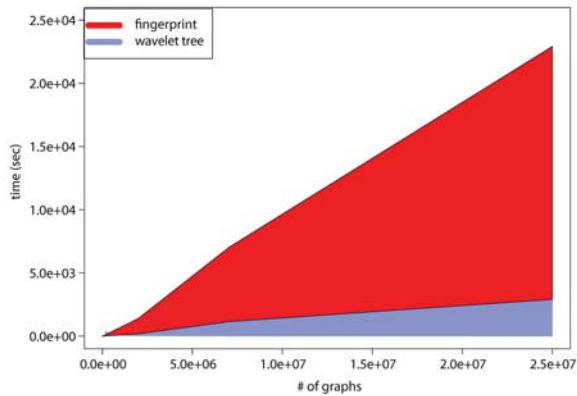
## メモリ



## 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なら何でも応用可能