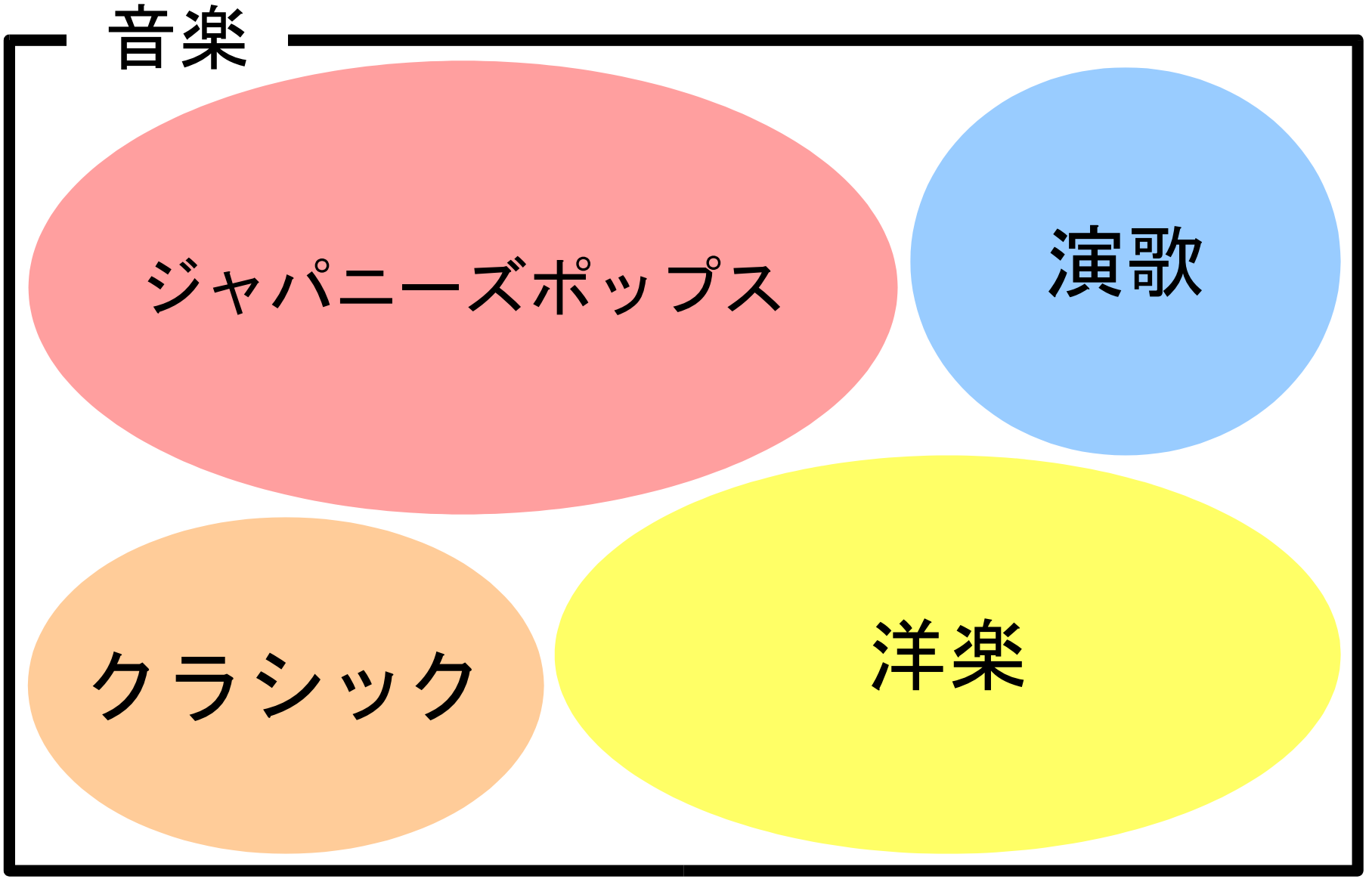
Kolmogorov Complexityを用いた民俗音楽の分類

日本大学文理学部
情報システム解析学科
谷 研究室
三井 貴子

もくじ

もくじ

はじめに



はじめに

音楽

ジャパニーズポップス　演歌

クラシック

・管弦楽曲
・独奏曲
・吹奏楽

・バッハ
・シューベルト
・ショパン

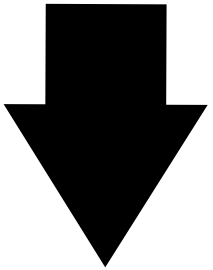
・バロック
・古典派
・ロマン派

はじめに

先行研究

Rudi Cilibrasiらが電子的に保存された音楽ファイルを自動的に分類

対象：クラシック，ジャズ，ロックなど



本研究

対象：民俗音楽(民謡)

地理的背景や歴史的背景が現れるのではないか

はじめに

similarity metric

- Ming Liらによって提案(2003)
- Kolmogorov Complexityを用いてデータ間の距離を表す

Kolmogorov Complexity

- データに対する記述量を定義
- 帰納的でないため，計算不可能

Kolmogorov Complexity

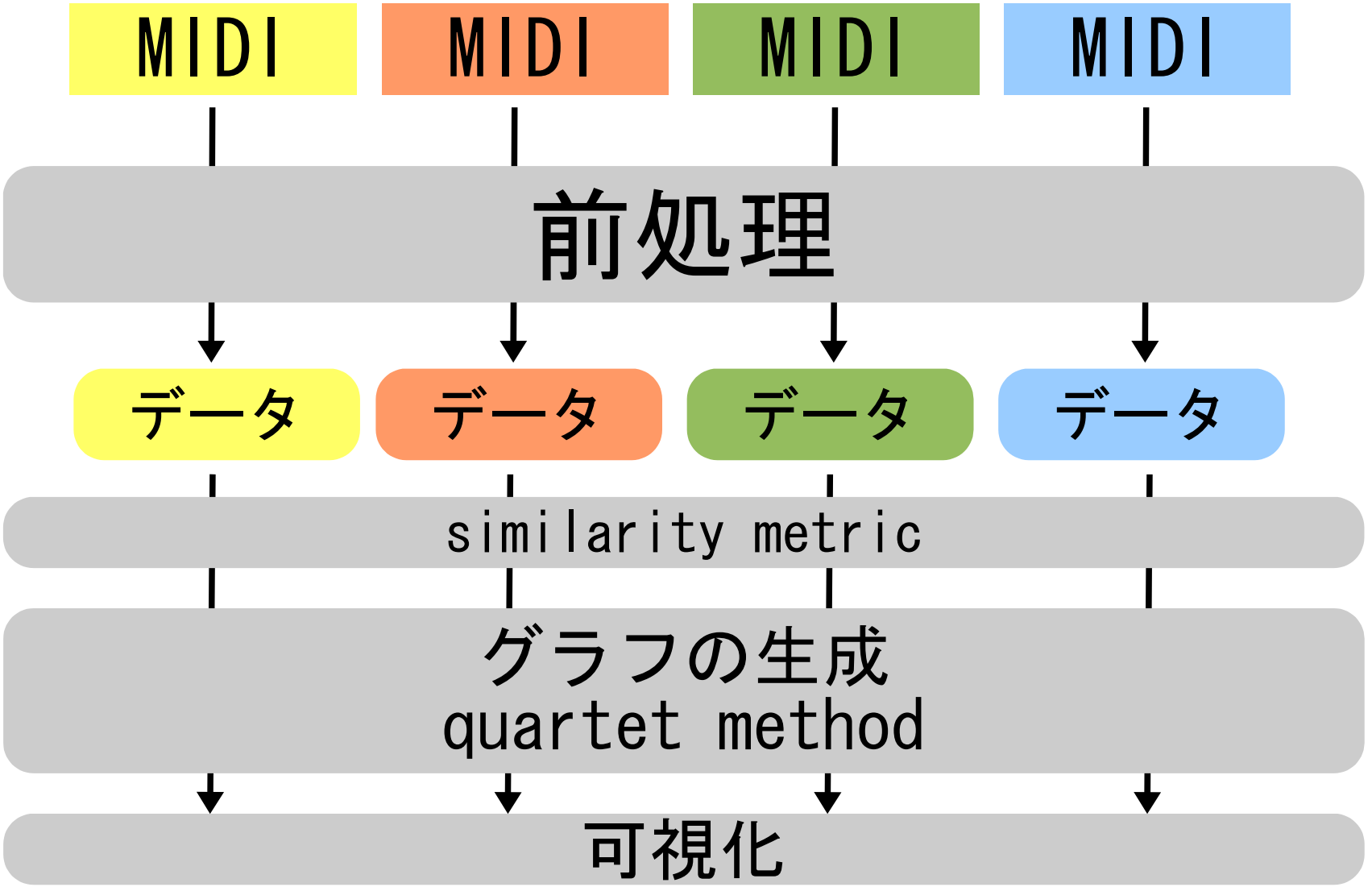
```
83 86 77 15 93 35 86 92 49 21
62 27 90 59 63 26 40 26 72 36
11 68 67 29 82 30 62 23 67 35
29  2 22 58 69 67 93 56 11 42
29 73 21 19 84 37 98 24 15 70
13 26 91 80 56 73 62 70 96 81
 5 25 84 27 36  5 46 29 13 57
24 95 82 45 14 67 34 64 43 50
87  8 76 78 88 84  3 51 54 99
32 60 76 68 39 12 26 86 94 39
```

```
#include<stdio.h>
int main(){
  int i,j;
  for(j=0;j<10;j++){
    for(i=0;i<10;i++){
    printf(" %2d ",random()%100);
    }
    printf("\n");
  }
  return(0);
}
```

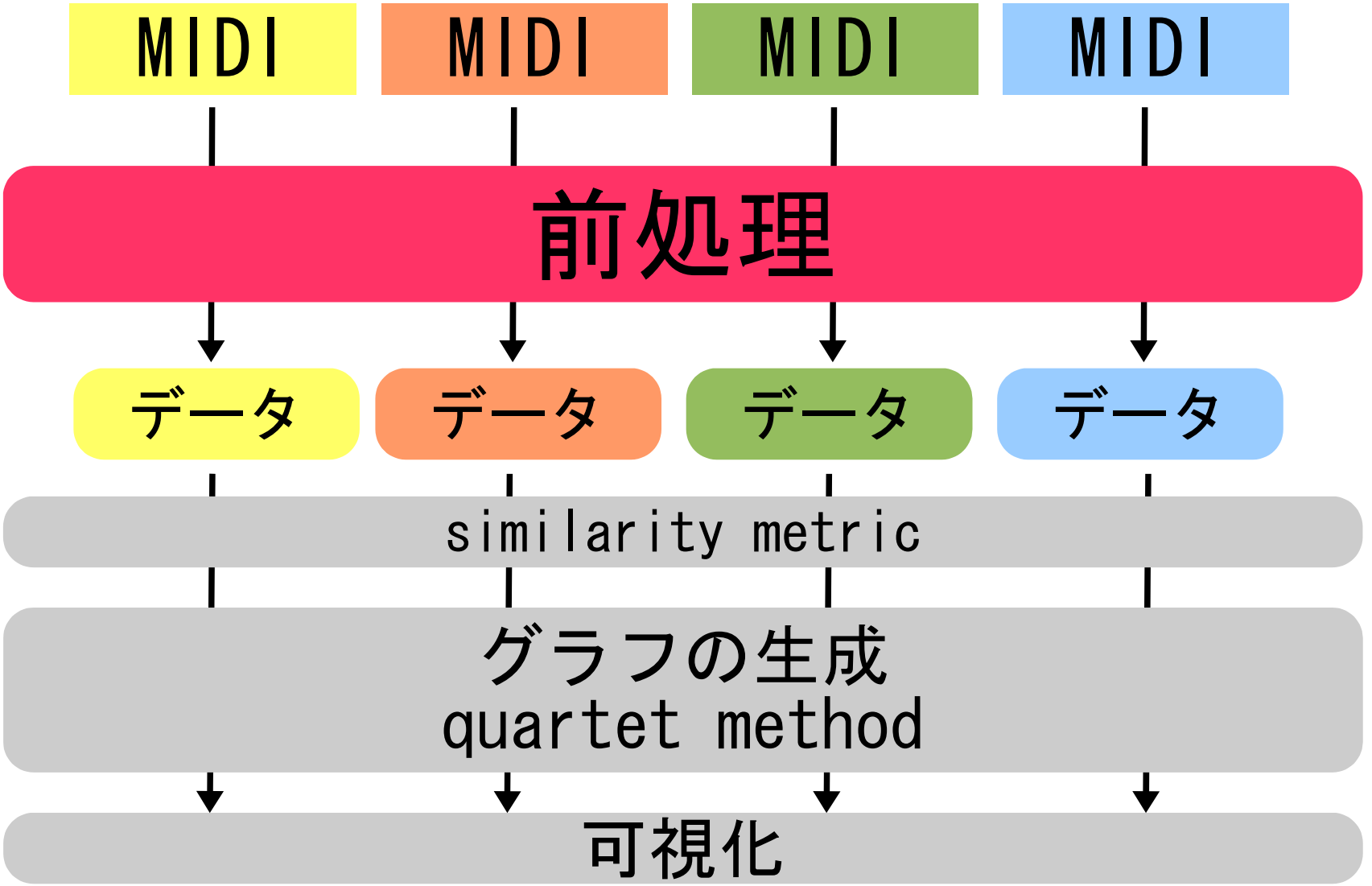
もくじ

- はじめに
- 実験方法
  - 前処理
  - similarity metric
  - quartet method
- 実験
- 今後の課題

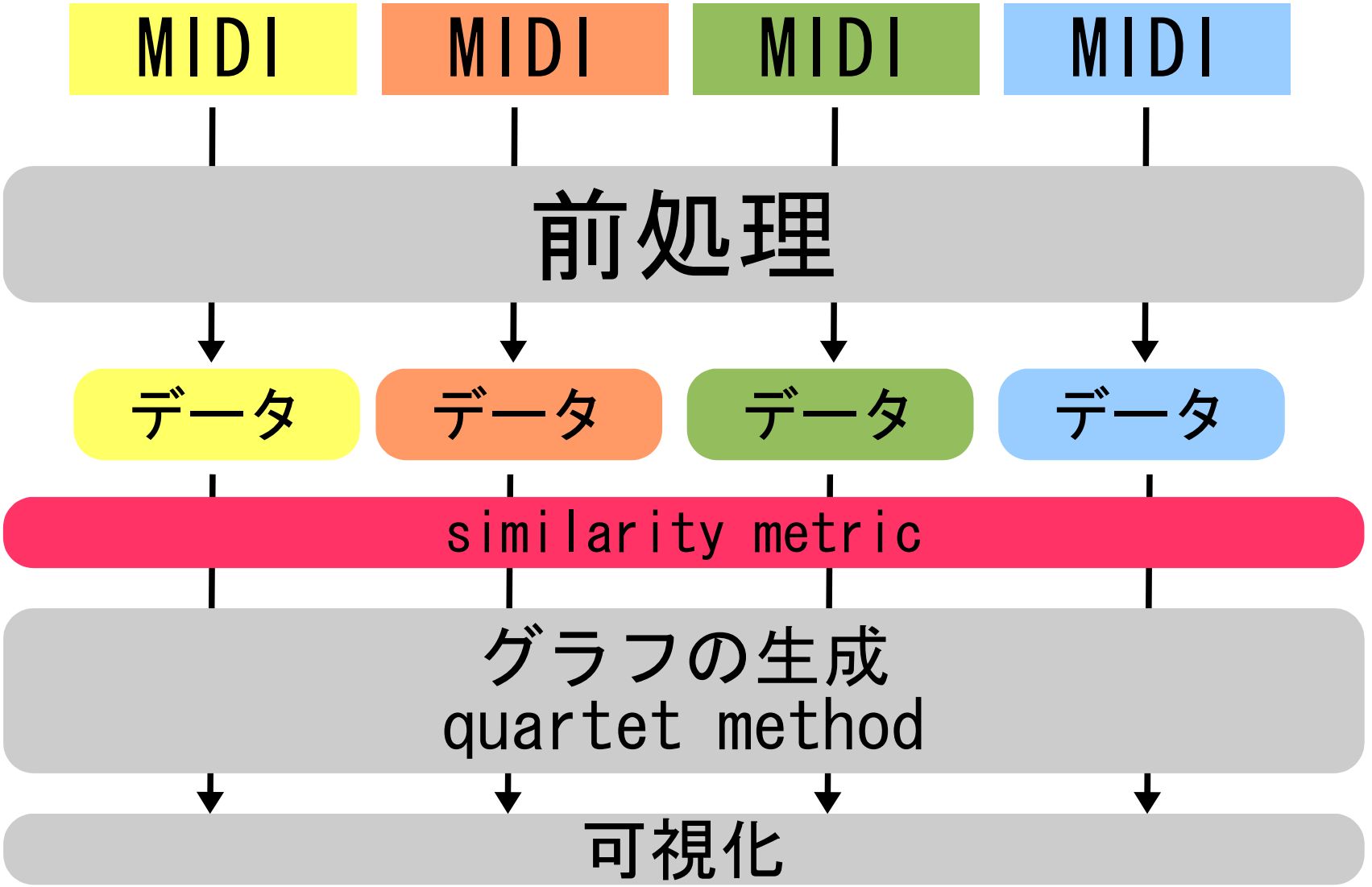
先行研究


MIDI
MIDI
MIDI
MIDI
前処理
データ
データ
データ
データ
similarity metric
グラフの生成
quartet method
可視化

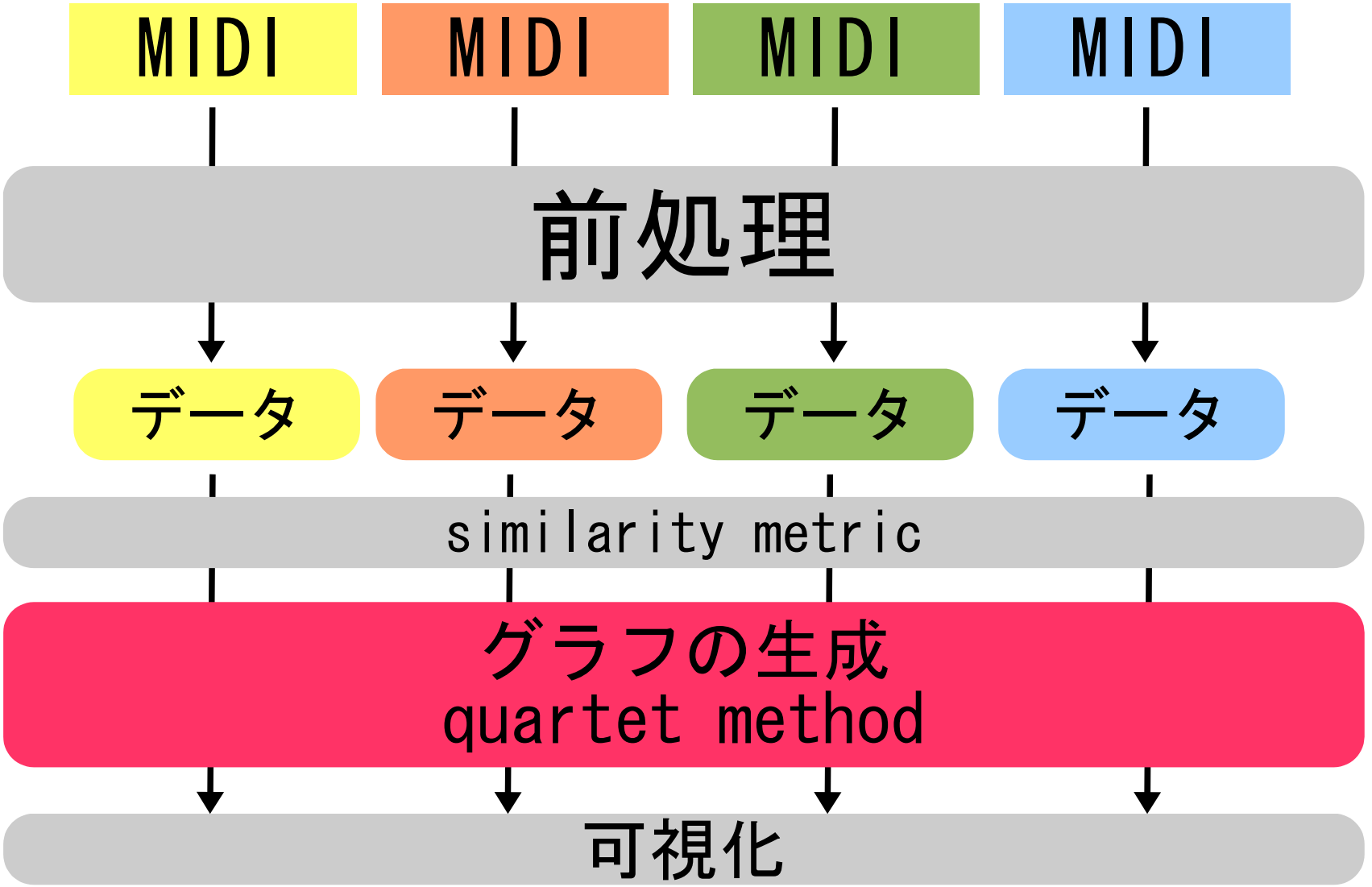
先行研究


MIDI
MIDI
MIDI
MIDI
前処理
データ
データ
データ
データ
similarity metric
グラフの生成
quartet method
可視化

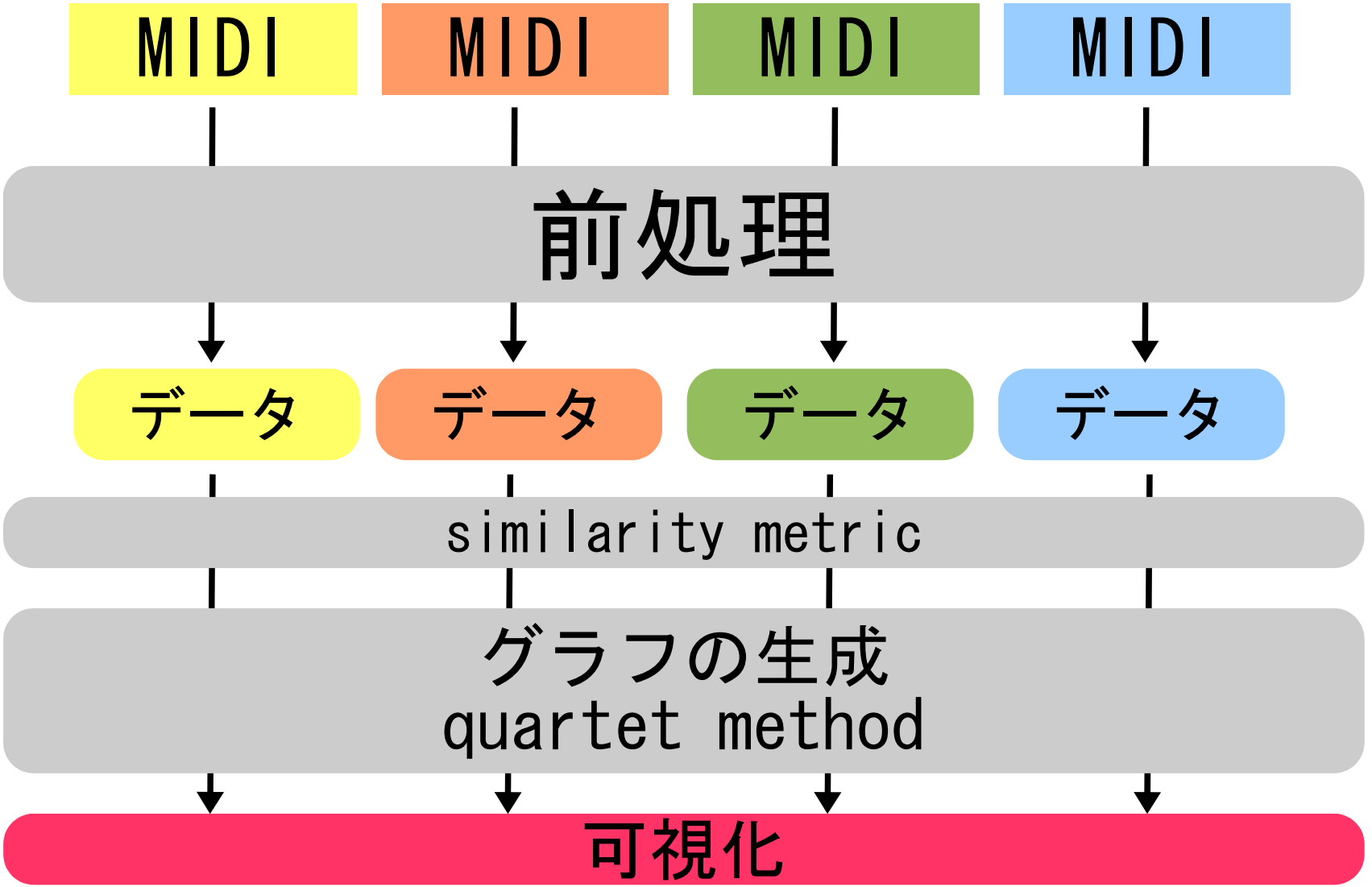
先行研究


MIDI
MIDI
MIDI
MIDI
前処理
データ
データ
データ
データ
similarity metric
グラフの生成
quartet method
可視化

先行研究


MIDI
MIDI
MIDI
MIDI
前処理
データ
データ
データ
データ
similarity metric
グラフの生成
quartet method
可視化

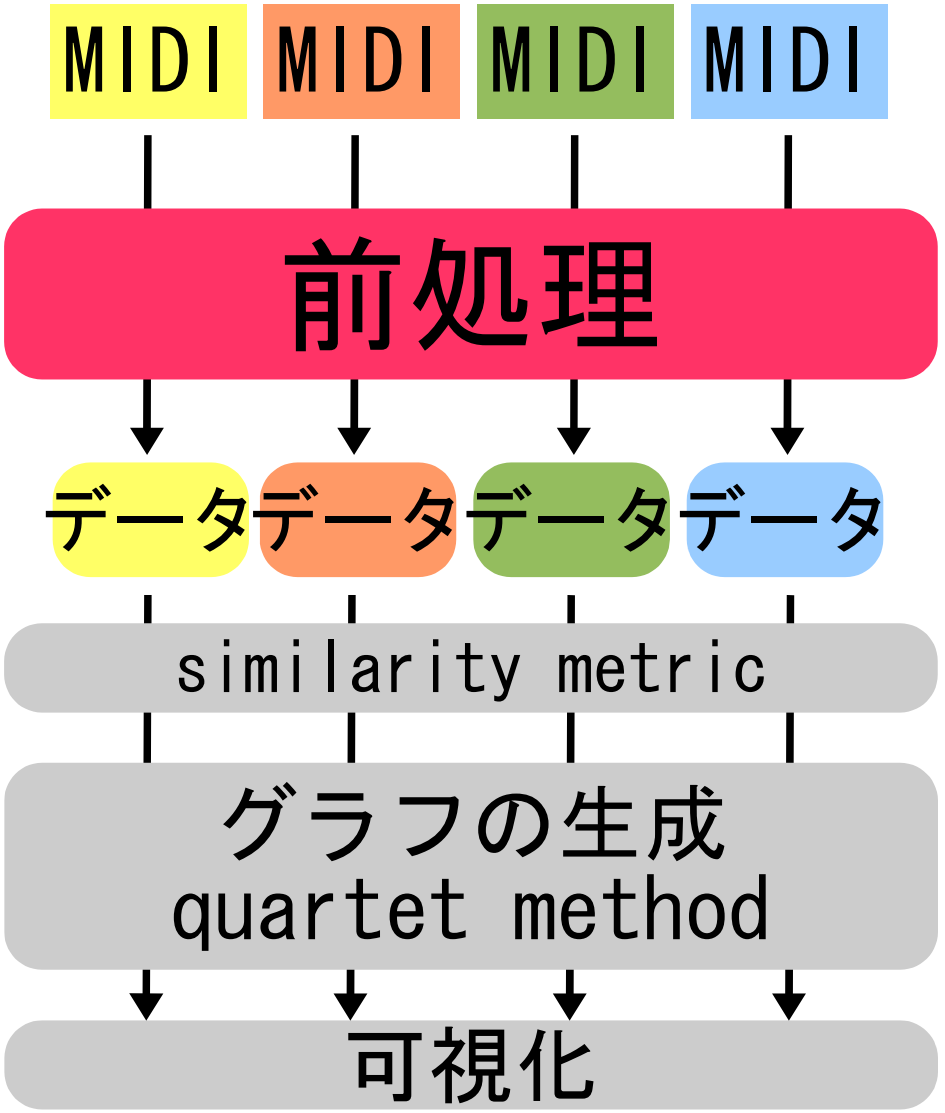
先行研究

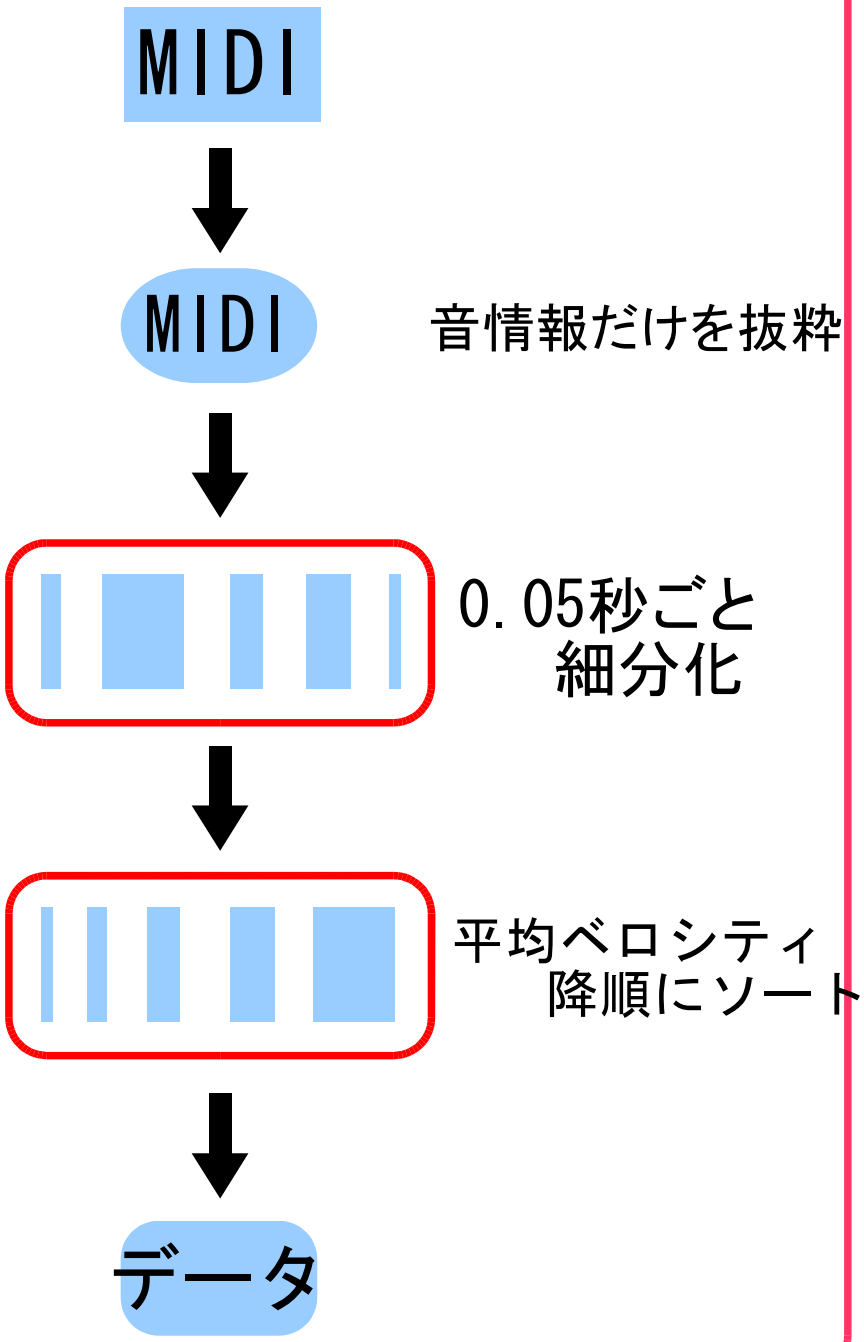

MIDI
MIDI
MIDI
MIDI
前処理
データ
データ
データ
データ
similarity metric
グラフの生成
quartet method
可視化

もくじ

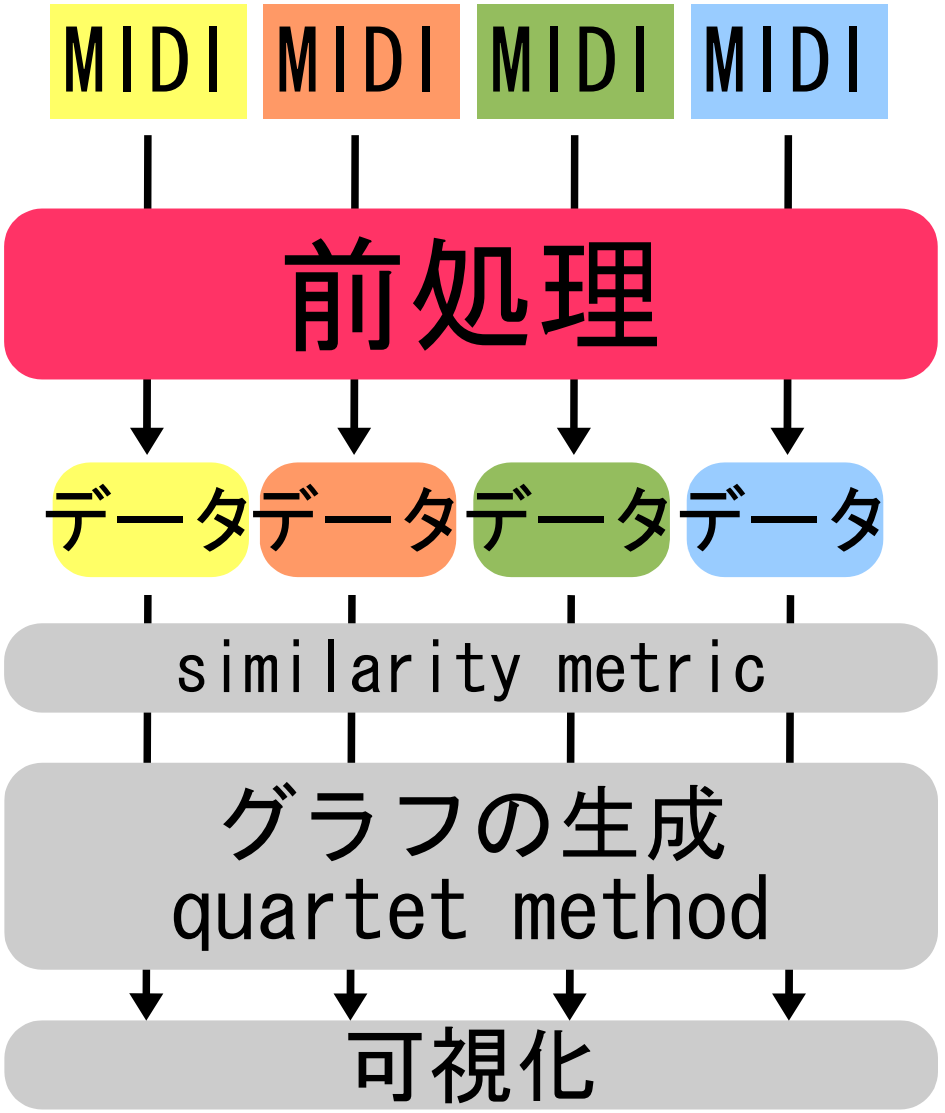
- はじめに
- 実験方法
  - 前処理
  - similarity metric
  - quartet method
- 実験
- 今後の課題

先行研究


MIDI
MIDI
MIDI
MIDI
前処理
データ
データ
データ
データ
similarity metric
グラフの生成
quartet method
可視化

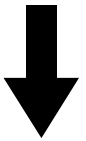

MIDI
MIDI
音情報だけを抜粋
0.05秒ごと
細分化
平均ベロシティ
降順にソート
データ

本研究



MIDI

```
$ ./a.exe ../MIDI/original/canada.mid
4D 54 68 64 00 00 00 06 00 00 00 01 01 E0
4D 54 72 6B 00 00 15 A6 00 FF 03 40 82 B1
82 CC 8E F0 82 C8 82 C2 82 A9 82 B5 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 00 FF 51 03 08 7A 23 00
FF 59 02 01 00 00 FF 58 04 02 02 18 08 00
FF 21 01 00 00 B4 00 00 00 B4 20 00 00 C4
40 00 B4 01 14 00 B4 07 6E 00 B4 0A 2D 00
```

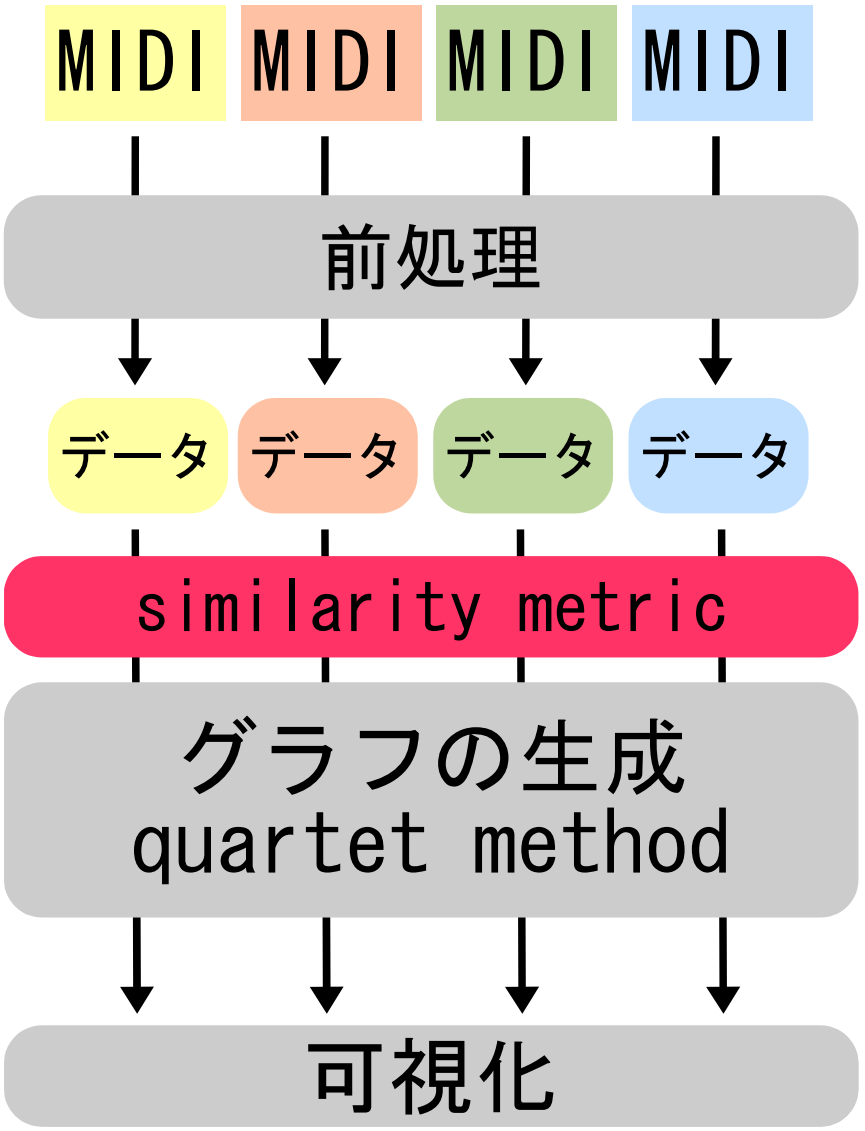


```
$ ./a.exe ../MIDI/preprocess/canada.mid
4D 54 68 64 00 00 00 06 00 00 00 01 01 E0
4D 54 72 6B 00 00 15 A6 00 90 43 55 00 90
2B 46 14 90 47 55 81 34 90 43 00 14 90 47
00 14 90 43 55 14 90 47 55 81 34 90 43 00
14 90 47 00 14 90 43 55 14 90 47 55 81 34
90 43 00 14 90 47 00 14 90 3E 55 14 90 45
55 81 34 90 3E 00 00 90 2B 00 14 90 45 00
14 90 37 4B 00 90 3B 4B 00 90 43 55 00 90
2B 46 14 90 47 55 81 34 90 37 00 00 90 3B
00 28 90 37 4B 00 90 3B 4B 81 48 90 37 00
```

もくじ

- はじめに
- 実験方法
  - 前処理
  - similarity metric
  - quartet method
- 実験
- 今後の課題

先行研究



similarity metric

2003年
Ming Li, Xi Chenらによってに提案

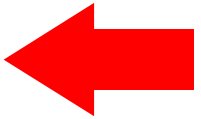
$$d(x,y) = \frac{max\{K(x|y), K(y|x)\}}{max\{K(x), K(y)\}}$$

K：Kolmogorov Complexity

Kolmogorov Complexity

x:1111100000

$K(x)$



```
#include<stdio.h>
int main(){
  int i;
   for(i=0;i<5;i++){
    printf("1");
   }
   for(i=0;i<5;i++){
    printf("0");
   }
  return(0);
}
```

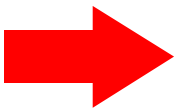
どのようなプログラム言語を選んでも
最小のプログラムサイズは定数しか違わない

Kolmogorov Complexity

y:11111000001111100000111110000011111000001111100000

```
#include<stdio.h>
int main(){
  int i,x=1111100000;
   for(i=0;i<5;i++){
    printf("%d",x);
   }
  return(0);
}
```

補助情報



$K(y|x)$

Kolmogorov Complexity

$K(x)$: x の記述量

$K(y|x)$: x を用いた y の記述量

$K(xy)$: x と y の記述量

y に含まれる x の情報量 → $I(x:y)$

$$I(x:y) = K(y) - K(y|x)$$

$O(\log K(xy))$ の誤差範囲

$$K(xy) = K(x) + K(y|x)$$

→ $$I(x:y) = I(y:x)$$

Kolmogorov Complexity

$$I(x:y) = K(y) - K(y|x)$$

$O(\log K(xy))$の範囲では,

$$K(xy) = K(x) + K(y|x)$$

$$d(x,y) = \frac{max\{K(x|y), K(y|x)\}}{max\{K(x), K(y)\}}$$

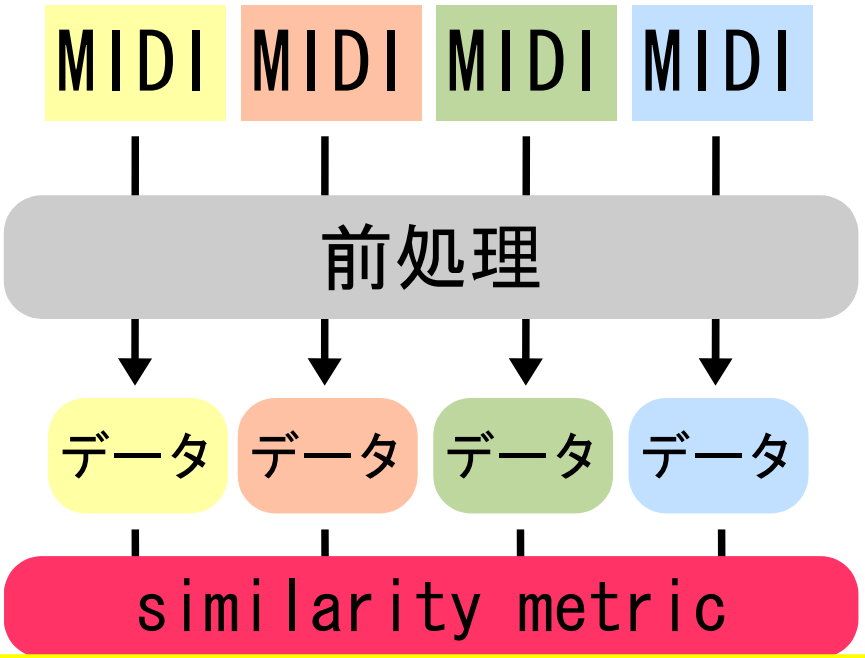
$K(y) \geq K(x)$ としたとき,

$$d(x,y) = \frac{K(y|x)}{K(y)}$$

$$= \frac{K(y) - I(x:y)}{K(y)}$$

$$= \frac{K(xy) - K(x)}{K(y)}$$

先行研究



similarity metric

2003年
Ming Li, Xi Chenに提案

$$d(x,y) = \frac{K(xy) - K(x)}{K(y)}$$

(1) 要求された情報距離 $d(x,y)$ は漠然と長い文字列 x,y によって得られる.

(2) Kolmogorov Complexity は帰納的でないため，計算不可能である.

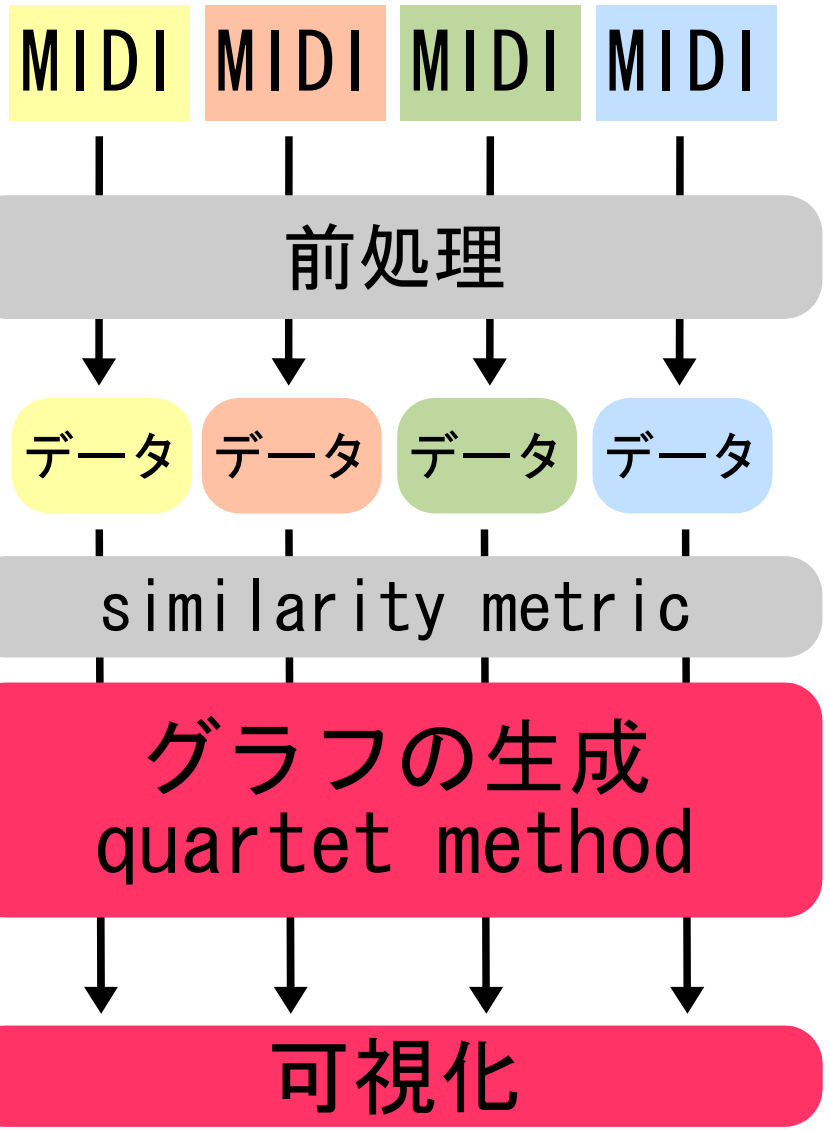
(3) 実用的な方法で情報距離を近似する際，圧縮方の一つである“bzip2”を用いる.

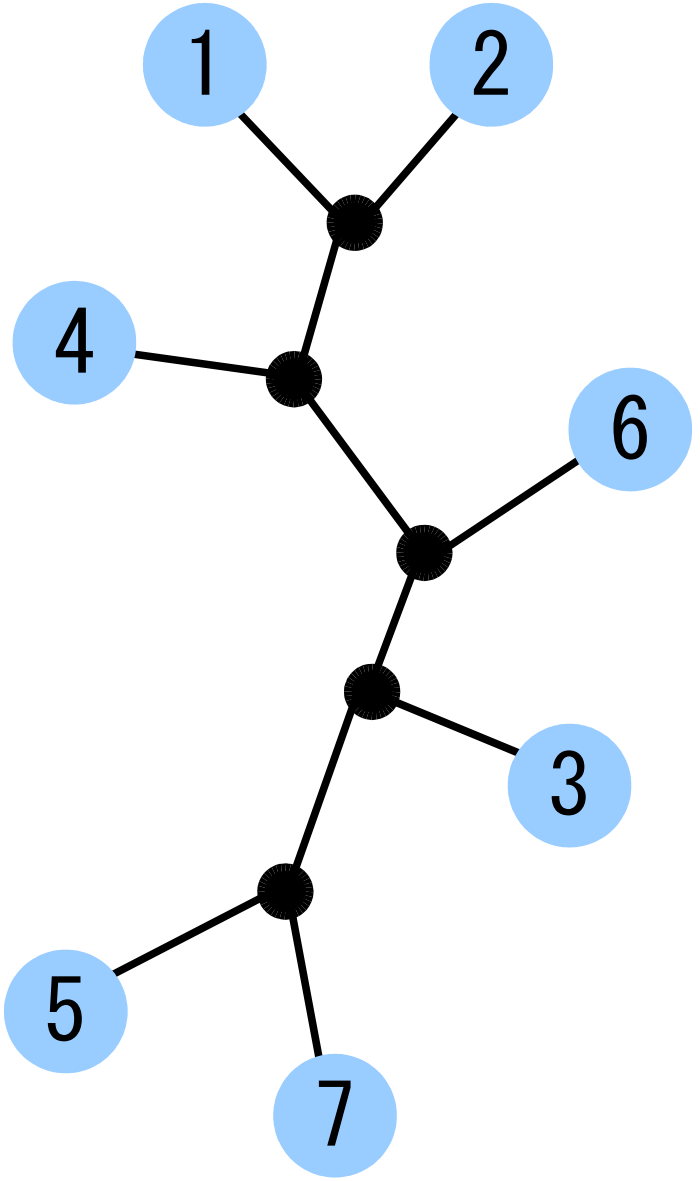
$$d(x,y) \fallingdotseq \frac{bzip(xy) - bzip(x)}{bzip(y)}$$

もくじ

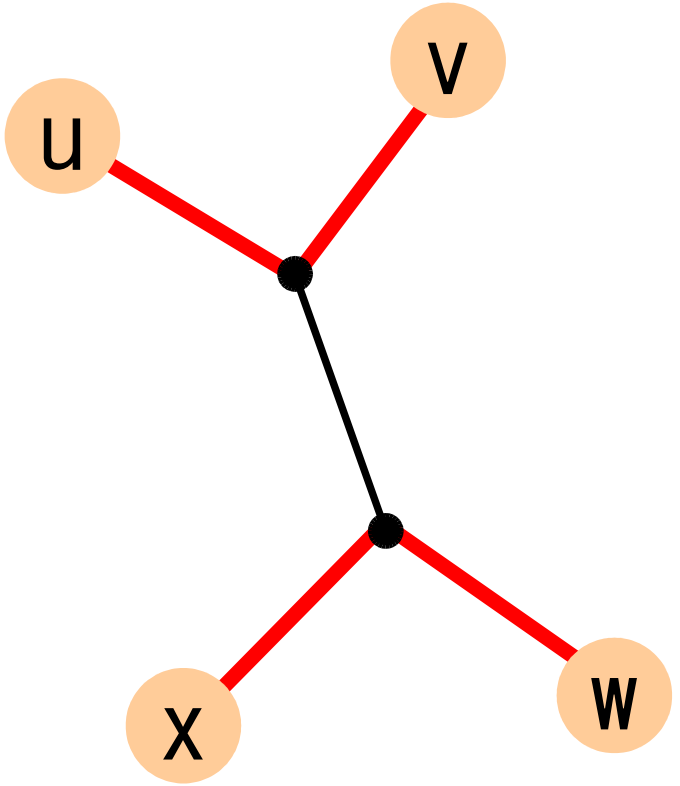
- はじめに
- 実験方法
  - 前処理
  - similarity metric
  - quartet method
- 実験
- 今後の課題

quartet method





quartet method



quartet

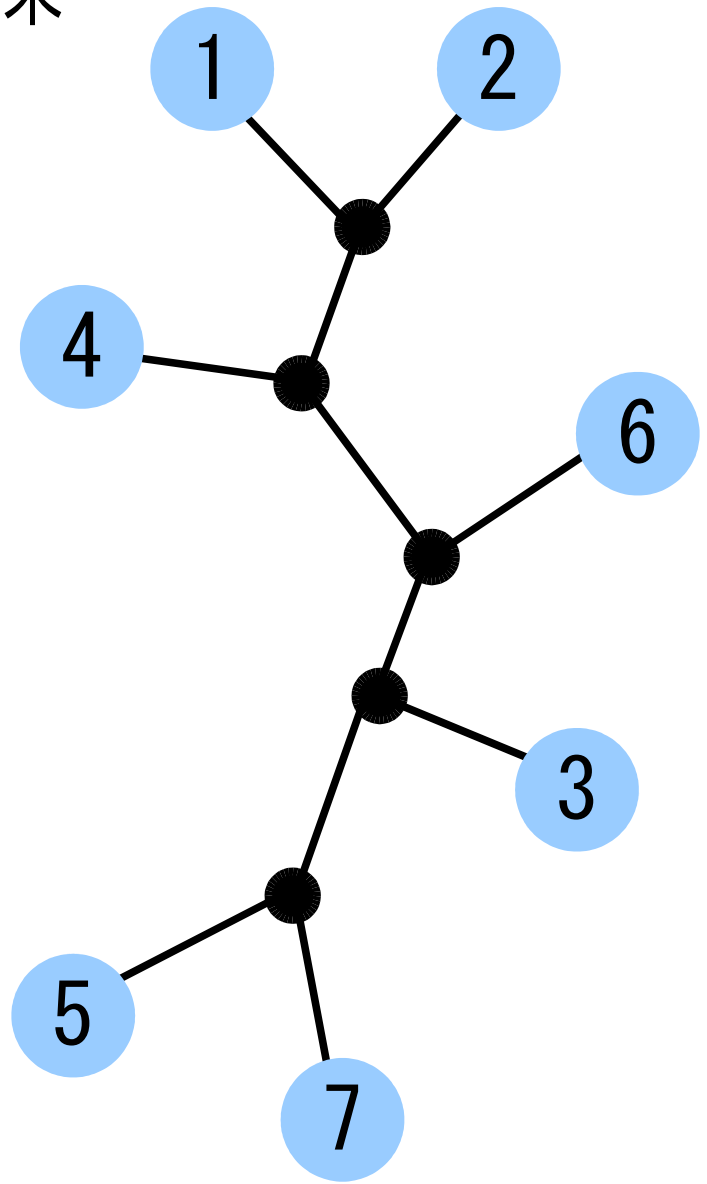
$$uv|wx$$

$$C_{uv|wx} = d(u,v) + d(w,x)$$

$$d(x,y) \fallingdotseq \frac{bzip(xy) - bzip(x)}{bzip(y)}$$

quartet method

T:木

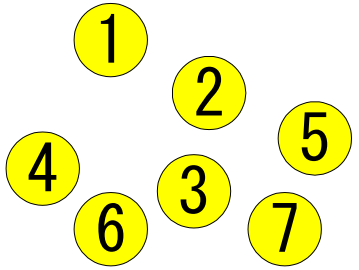


$$C_{uv|wx} = d(u,v) + d(w,x)$$

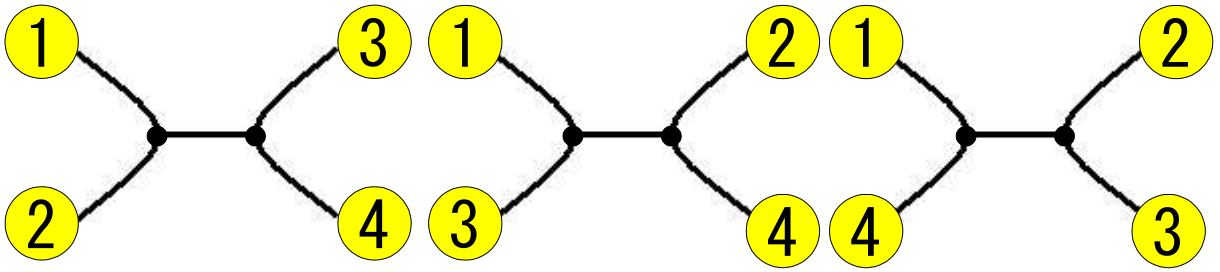
$$d(x,y) \doteqdot \frac{bzip(xy) - bzip(x)}{bzip(y)}$$

quartet method

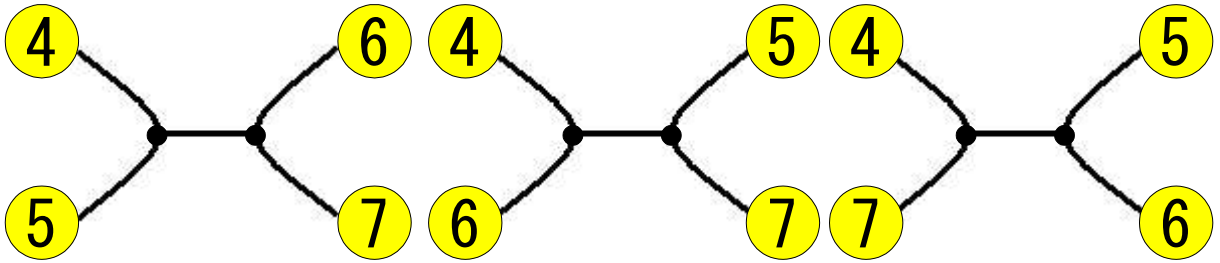
データ：7



{1, 2, 3, 4}

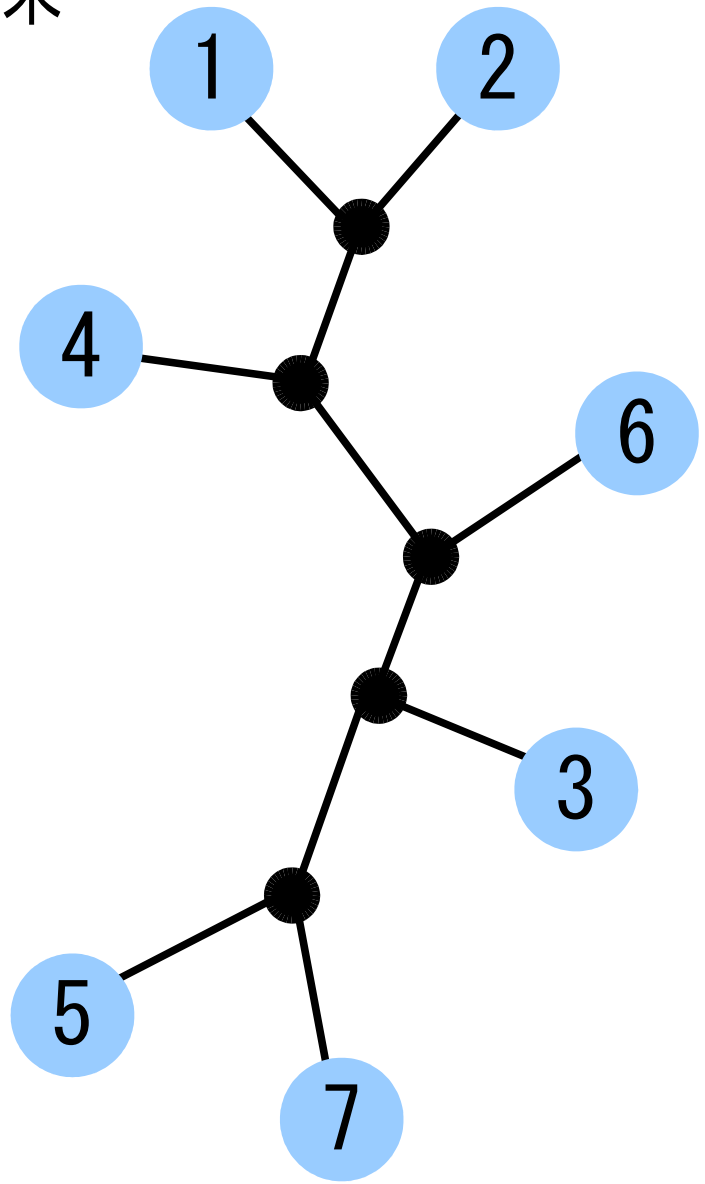


{4, 5, 6, 7}

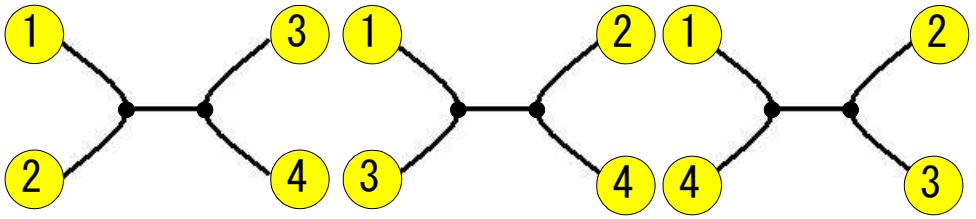


quartet method

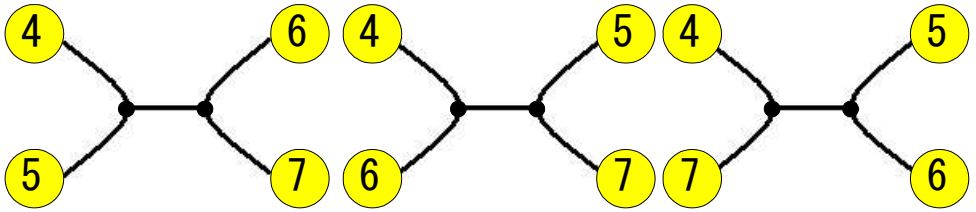
T:木



{1, 2, 3, 4}

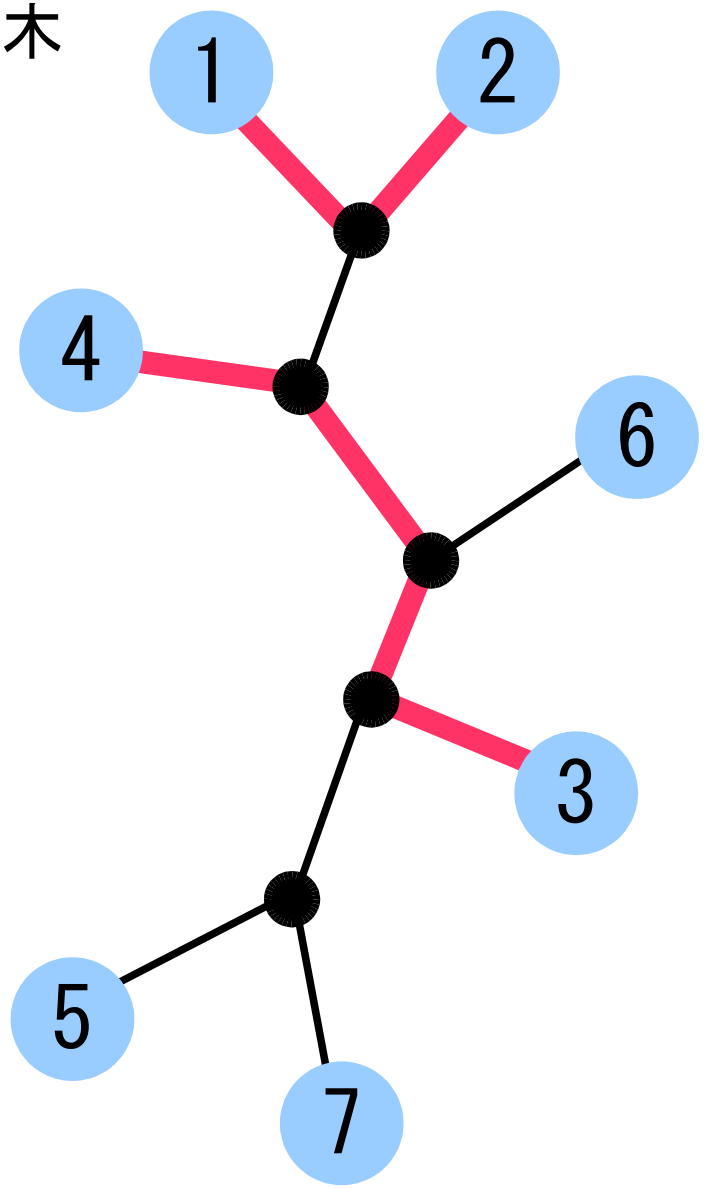


{4, 5, 6, 7}

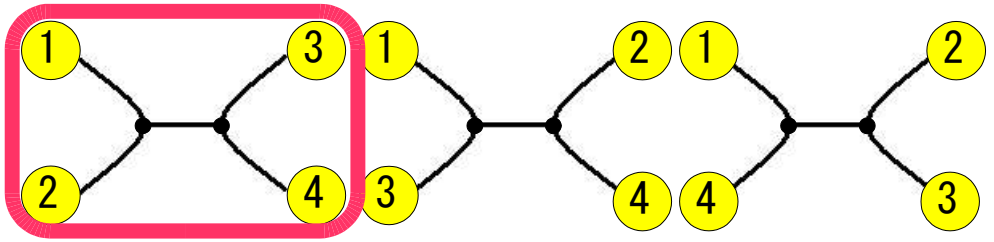


quartet method

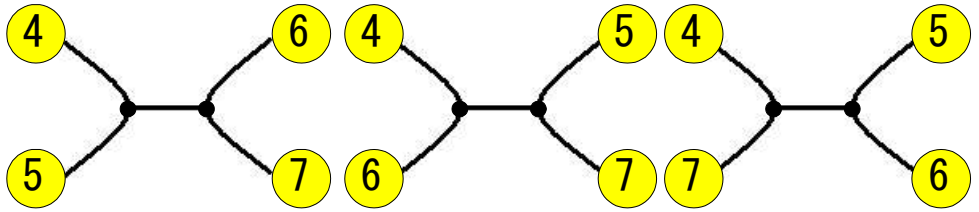
T:木



{1, 2, 3, 4}



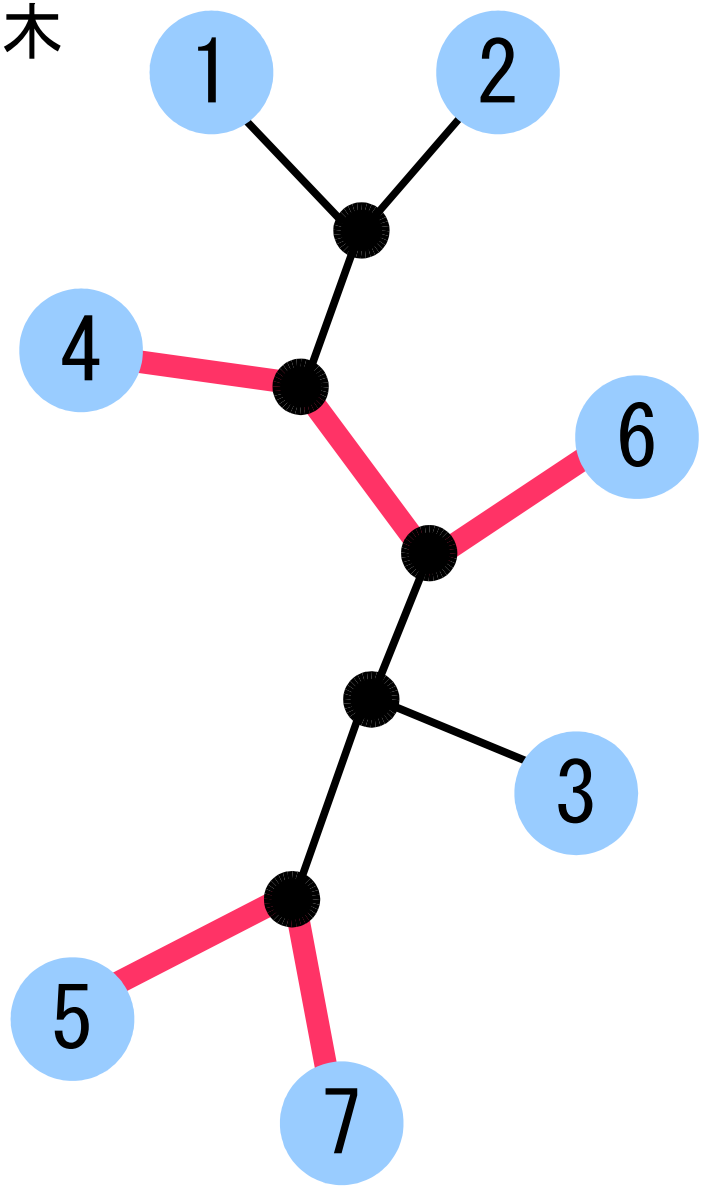
{4, 5, 6, 7}



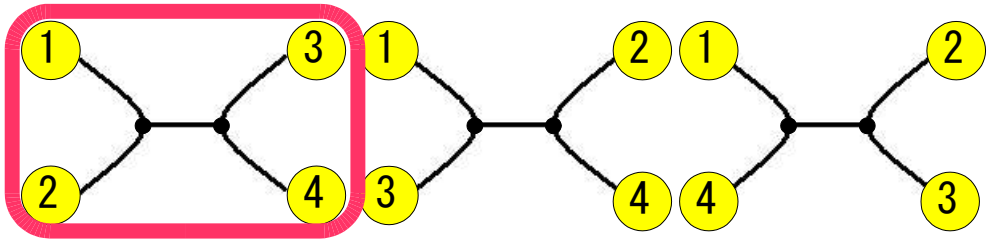
consistent

quartet method

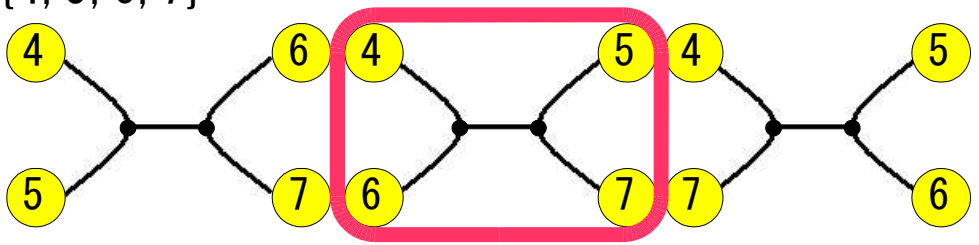
T:木



{1, 2, 3, 4}



{4, 5, 6, 7}



consistent

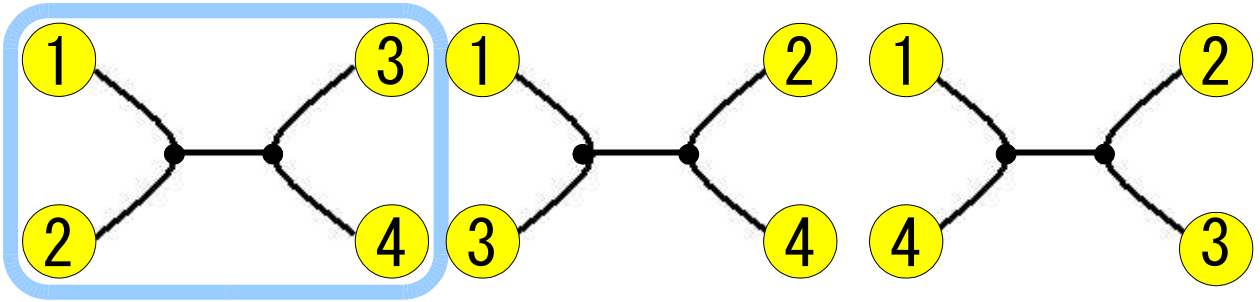
5. quartet method

$$C_{uv|wx} = d(u,v) + d(w,x)$$

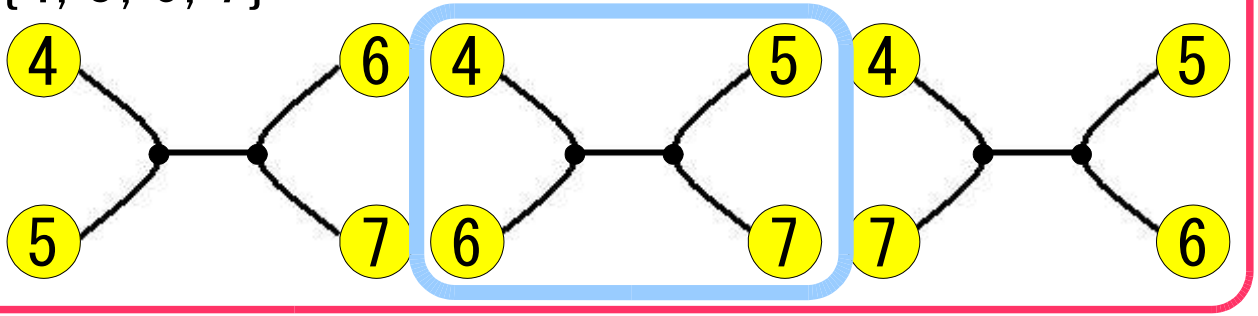
$$d(x,y) \fallingdotseq \frac{bzip(xy) - bzip(x)}{bzip(y)}$$

best cost

{1, 2, 3, 4}



{4, 5, 6, 7}



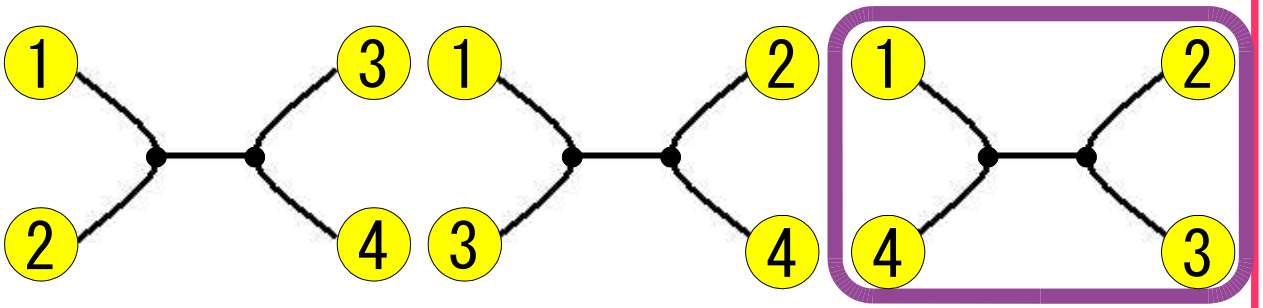
5. quartet method

$$C_{uv|wx} = d(u,v) + d(w,x)$$

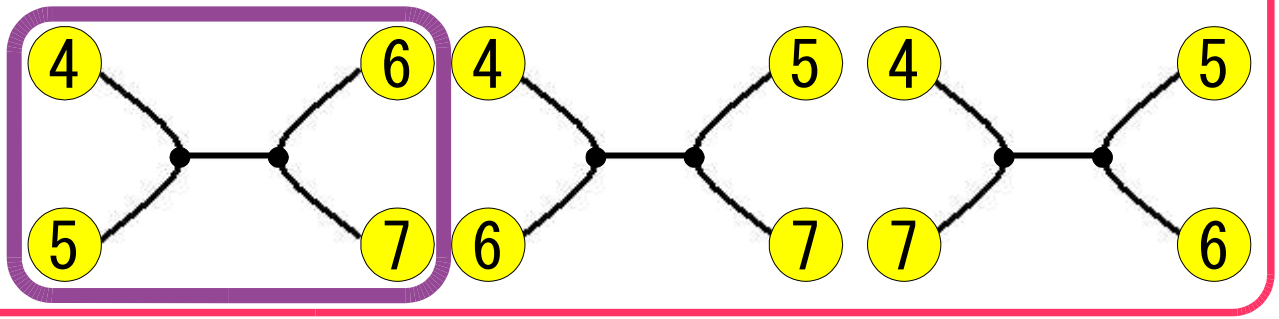
$$d(x,y) \fallingdotseq \frac{bzip(xy) - bzip(x)}{bzip(y)}$$

worst cost

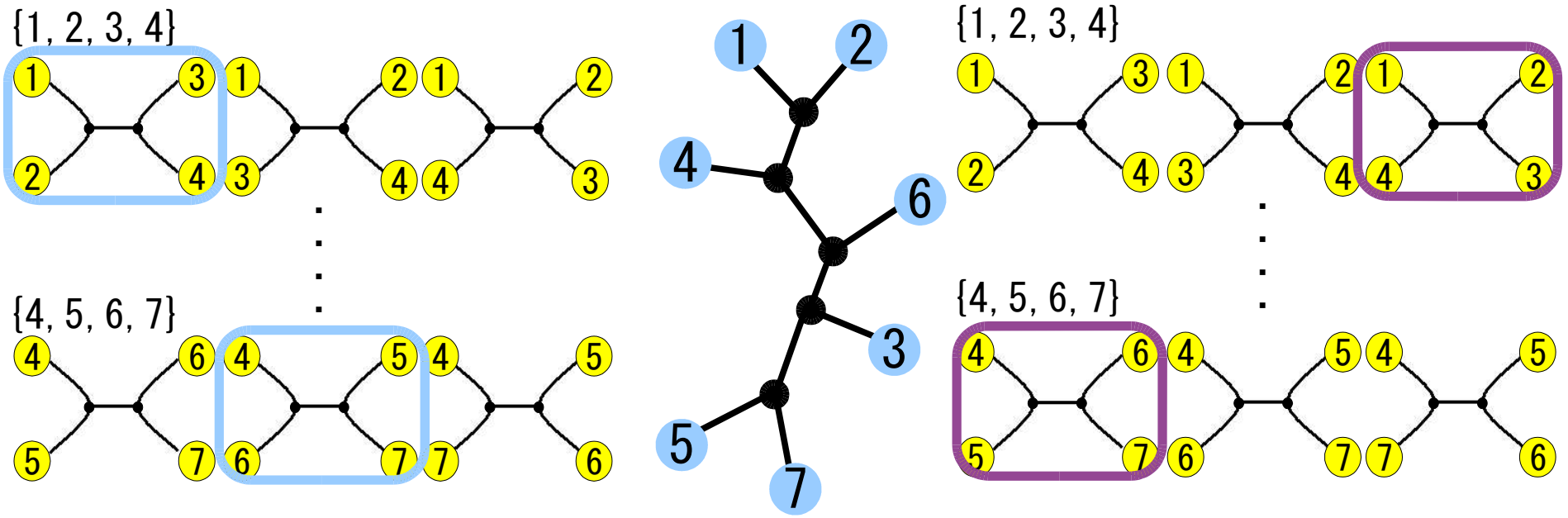
{1, 2, 3, 4}



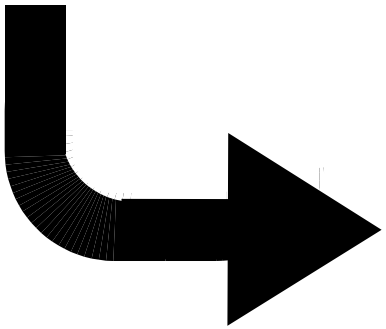
{4, 5, 6, 7}



木の評価方法



best cost ≦≦ 木のcost ≦≦ worst cost



S(T)=1 → NP−困難

quartet method

ランダムに木を生成

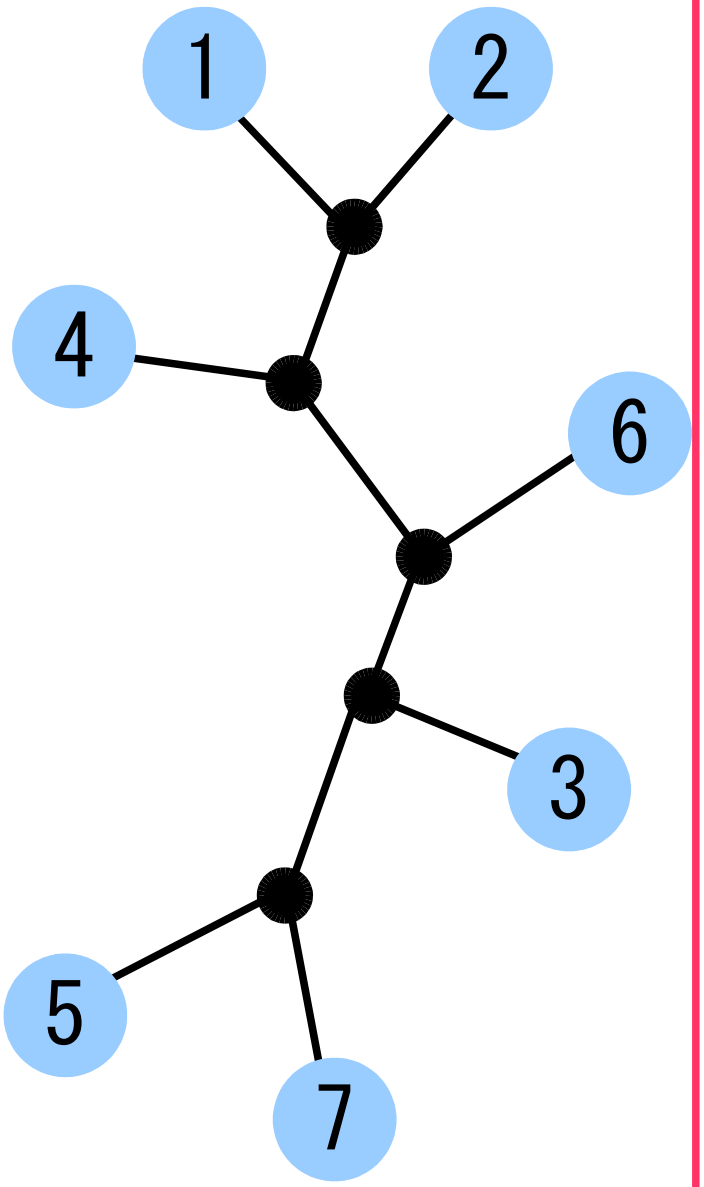
S(T)の値を計算

以下の操作からランダムに選ぶ

- leaf swap
- subtree swap
- subtree trancefer

k回繰り返す

S(T)の値を計算



quartet method

ランダムに木を生成

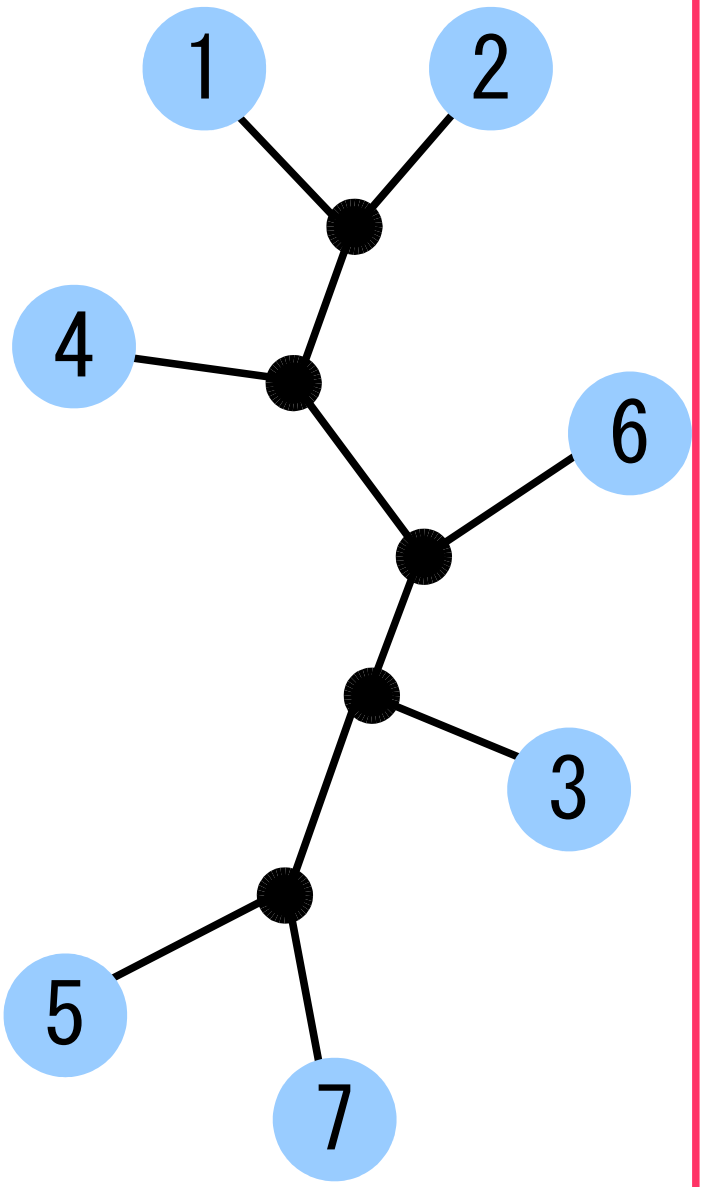
S(T)の値を計算

以下の操作からランダムに選ぶ

- leaf swap
- subtree swap
- subtree trancefer

k回繰り返す

S(T)の値を計算



quartet method

ランダムに木を生成

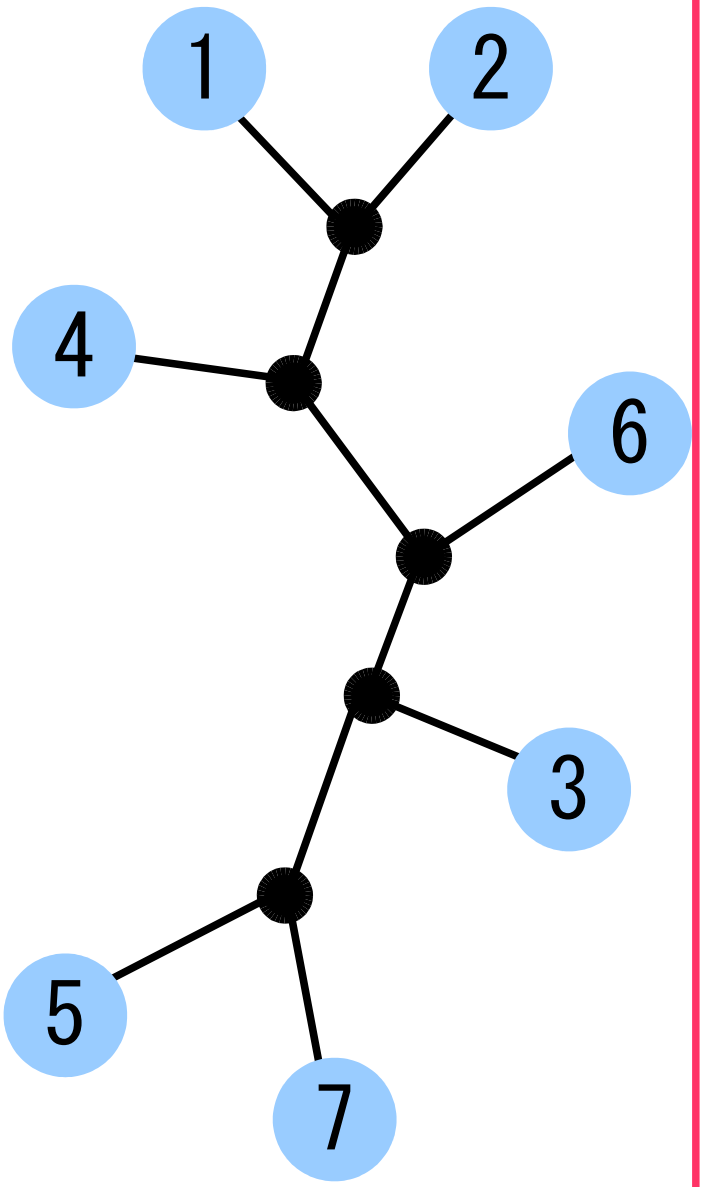
S(T)の値を計算

以下の操作からランダムに選ぶ

- leaf swap
- subtree swap
- subtree trancefer

k回繰り返す

S(T)の値を計算



quartet method

ランダムに木を生成

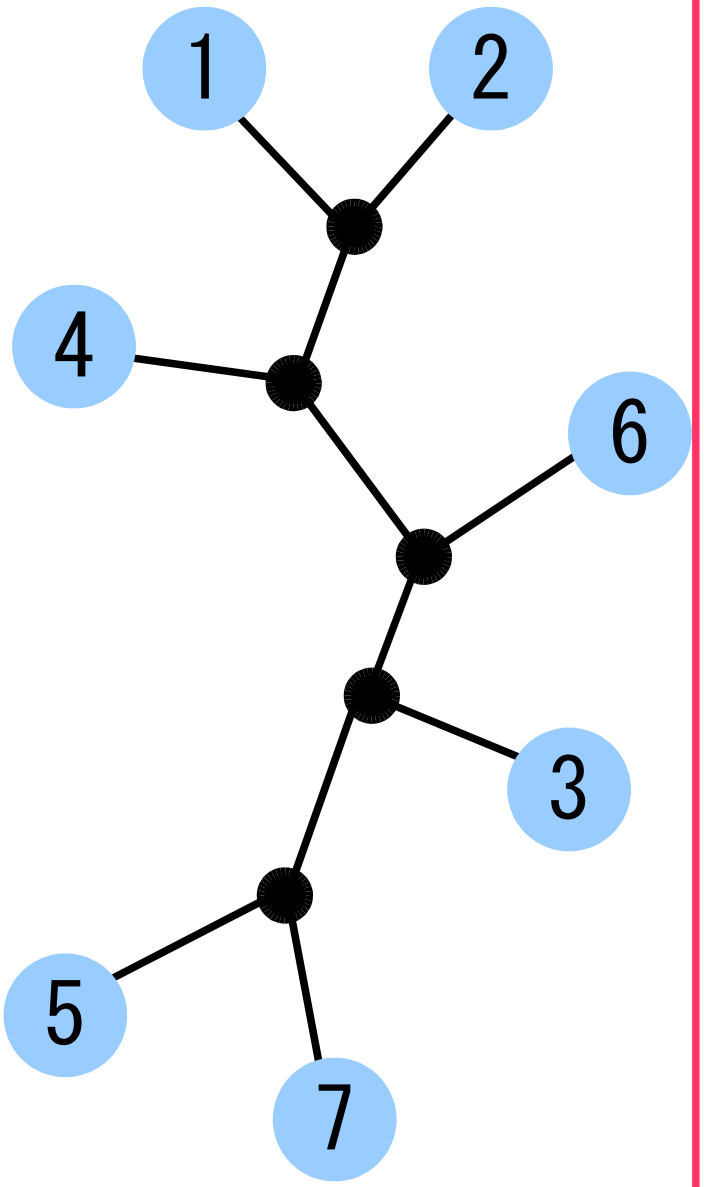
S(T)の値を計算

以下の操作からランダムに選ぶ

- leaf swap
- subtree swap
- subtree trancefer

k回繰り返す

S(T)の値を計算



quartet method

ランダムに木を生成

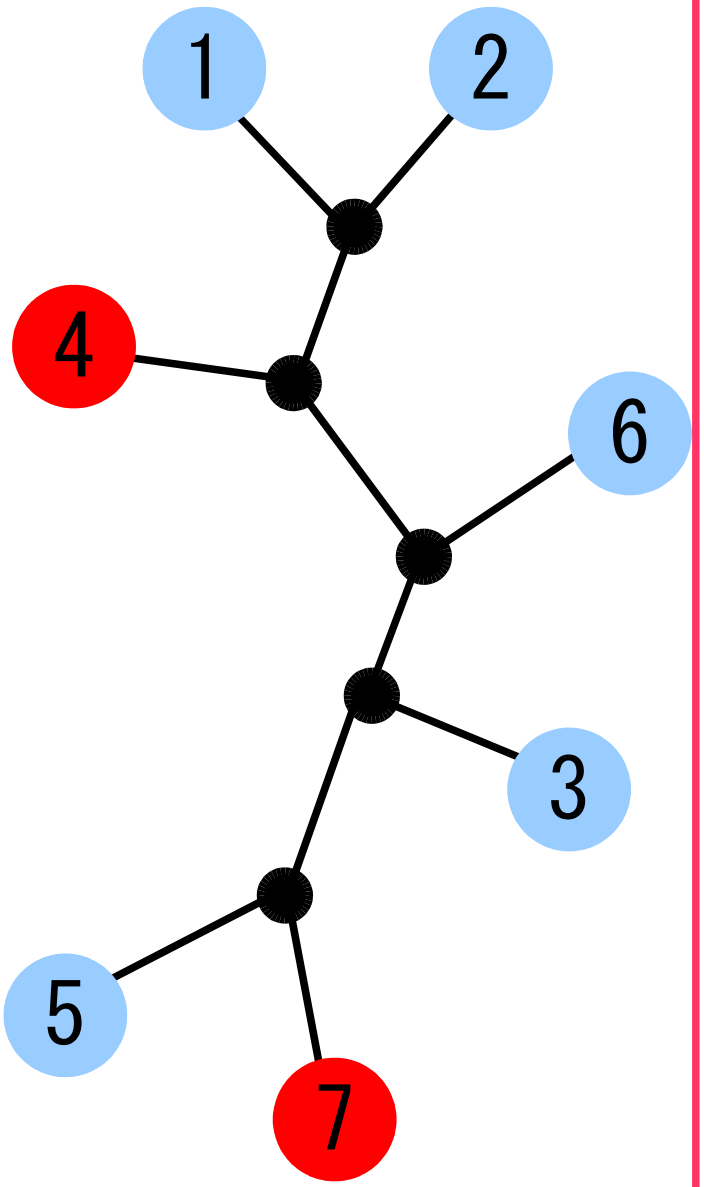
S(T)の値を計算

以下の操作からランダムに選ぶ

- leaf swap
- subtree swap
- subtree trancefer

k回繰り返す

S(T)の値を計算



quartet method

ランダムに木を生成

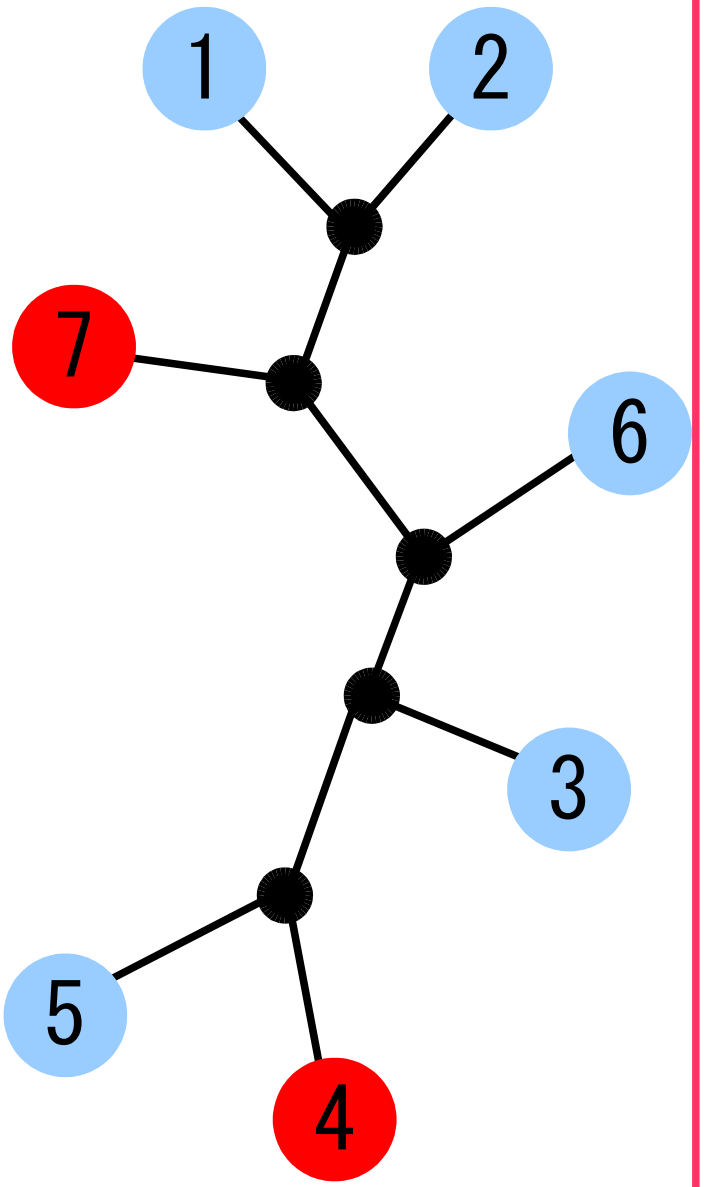
S(T)の値を計算

以下の操作からランダムに選ぶ

- leaf swap
- subtree swap
- subtree trancefer

k回繰り返す

S(T)の値を計算



quartet method

ランダムに木を生成

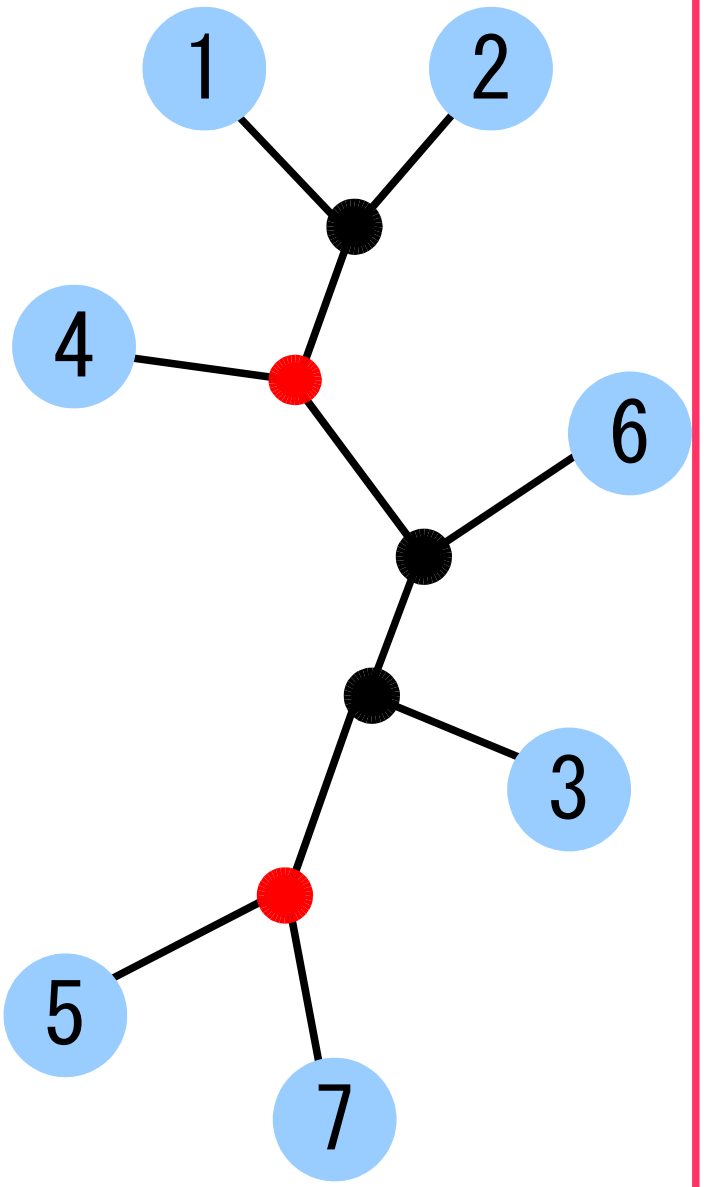
S(T)の値を計算

以下の操作からランダムに選ぶ

- leaf swap
- subtree swap
- subtree trancefer

k回繰り返す

S(T)の値を計算



quartet method

ランダムに木を生成

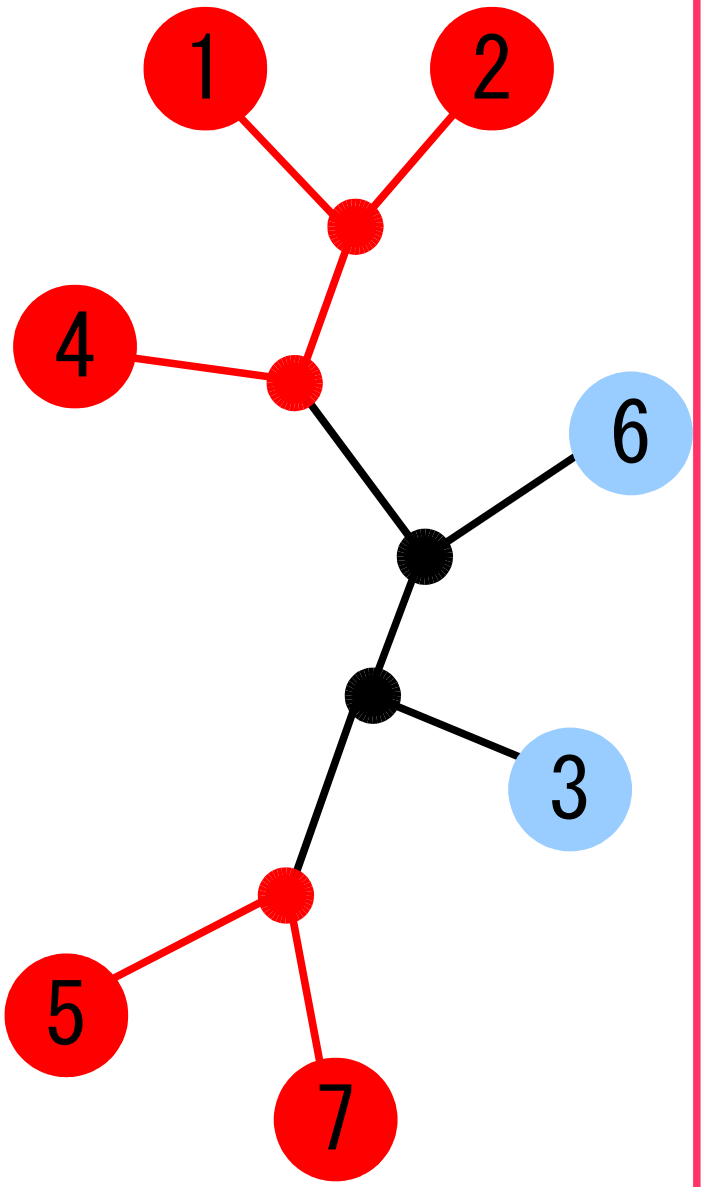
S(T)の値を計算

以下の操作からランダムに選ぶ

- leaf swap
- subtree swap
- subtree trancefer

k回繰り返す

S(T)の値を計算



quartet method

ランダムに木を生成

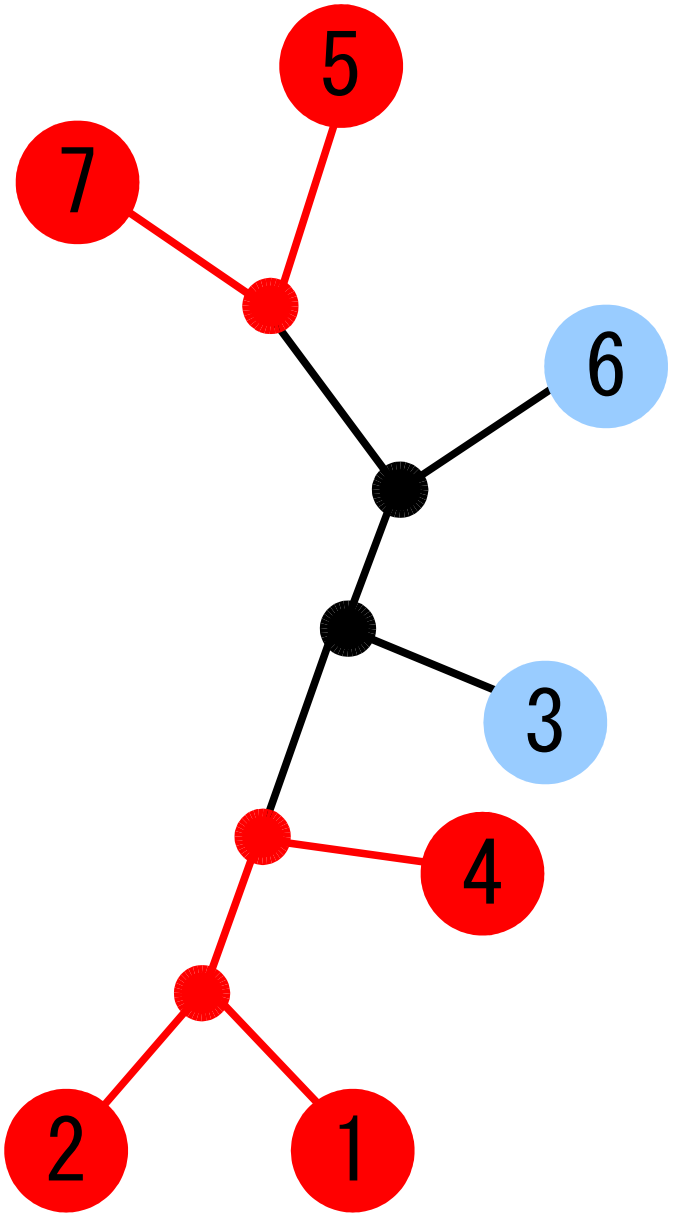
S(T)の値を計算

以下の操作からランダムに選ぶ

- leaf swap
- subtree swap
- subtree trancefer

k回繰り返す

S(T)の値を計算



quartet method

ランダムに木を生成

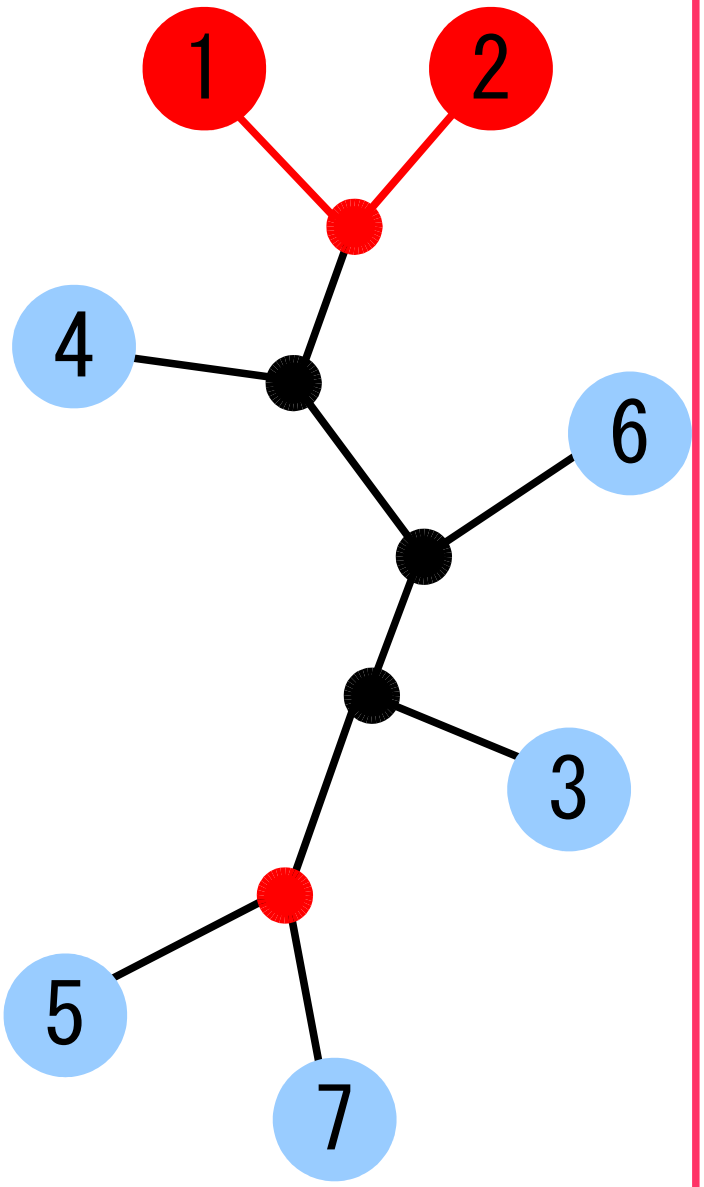
S(T)の値を計算

以下の操作からランダムに選ぶ

- leaf swap
- subtree swap
- subtree trancefer

k回繰り返す

S(T)の値を計算



quartet method

ランダムに木を生成

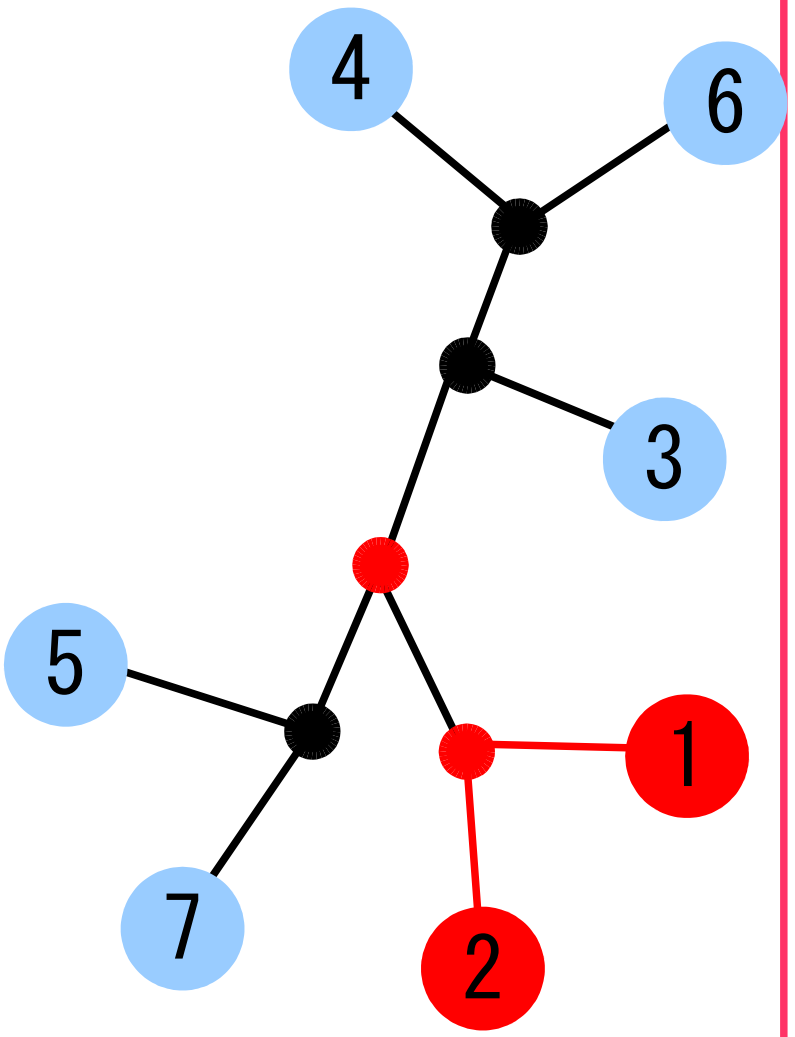
S(T)の値を計算

以下の操作からランダムに選ぶ

- leaf swap
- subtree swap
- subtree trancefer

k回繰り返す

S(T)の値を計算



quartet method

ランダムに木を生成

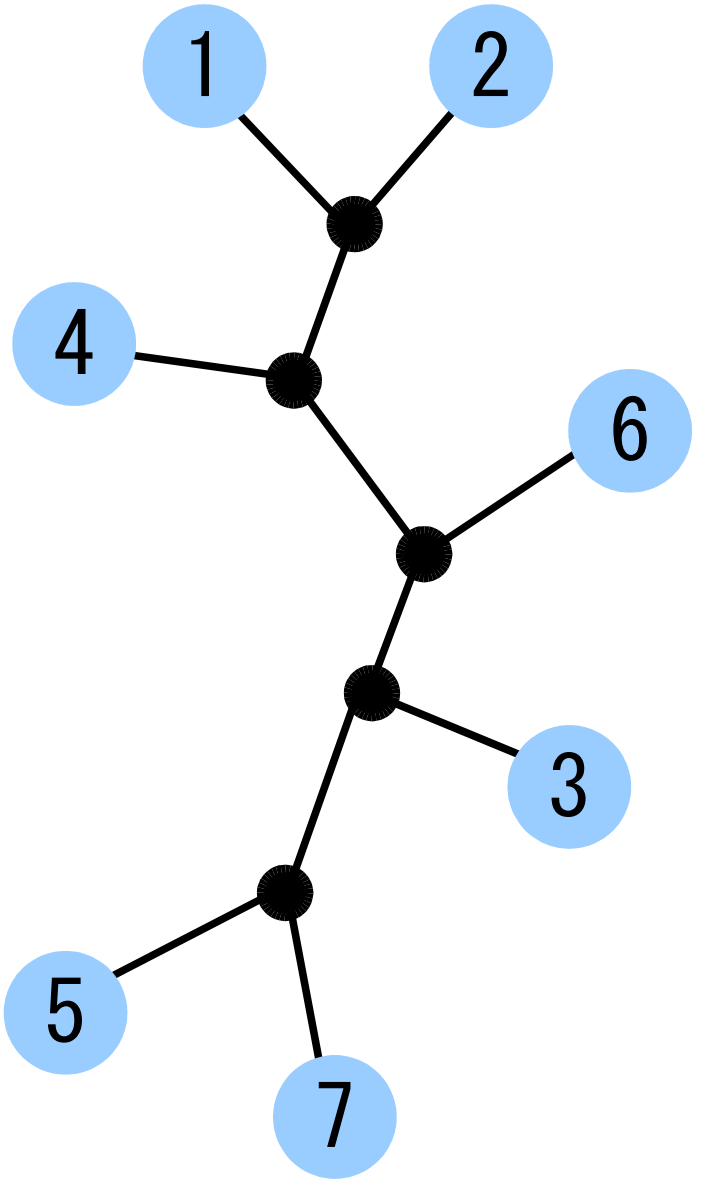
S(T)の値を計算

以下の操作からランダムに選ぶ

- leaf swap
- subtree swap
- subtree trancefer

k回繰り返す

S(T)の値を計算



もくじ

- はじめに
- 実験方法
  - 前処理
  - similarity metric
  - quartet method
- 実験
- 今後の課題

実験方法

～実験方法～

実験Ⅰ：先行研究と同じデータ
使用データ(Bach,Debussy,Chopinよりそれぞれ4曲)

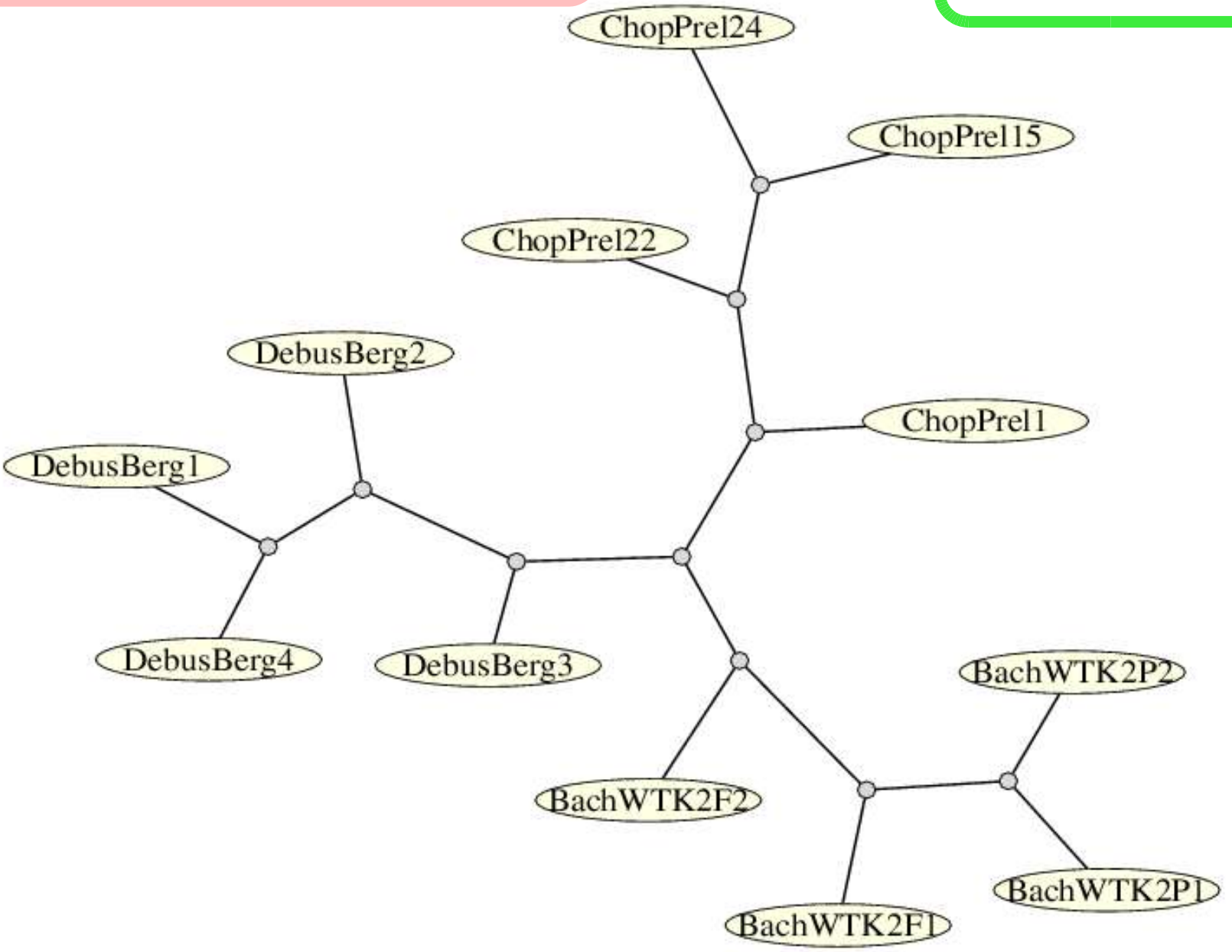
実験Ⅱ：世界各国の民謡
使用データ(同じ国より3,4曲ずつ,
世界27カ国から1曲ずつ)

実験Ⅲ：日本の民謡

それぞれtranceformやS(T)の値を固定

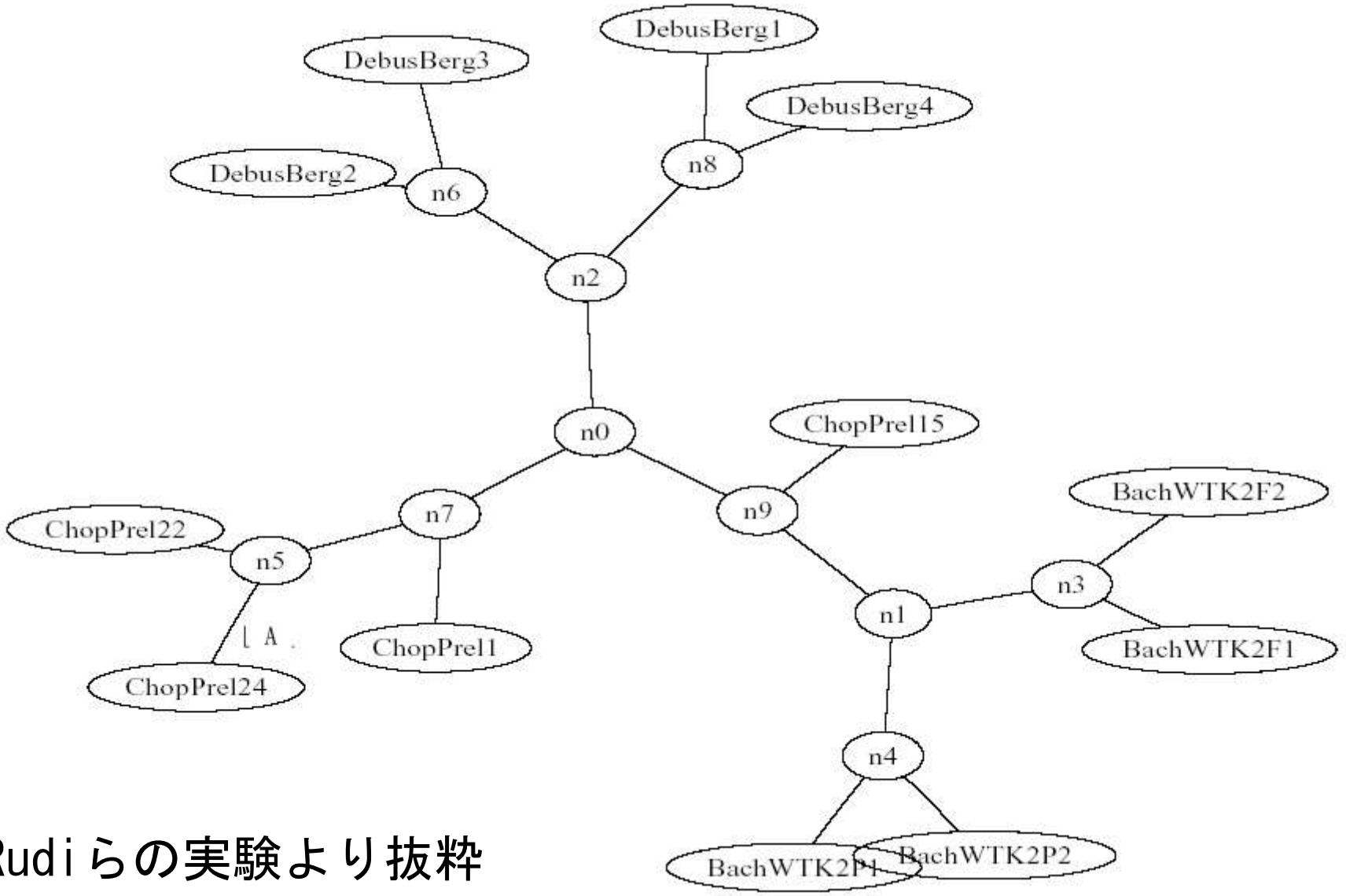
実験　～実験Ⅰ～

S(T)=0.9488



Rudiらの結果

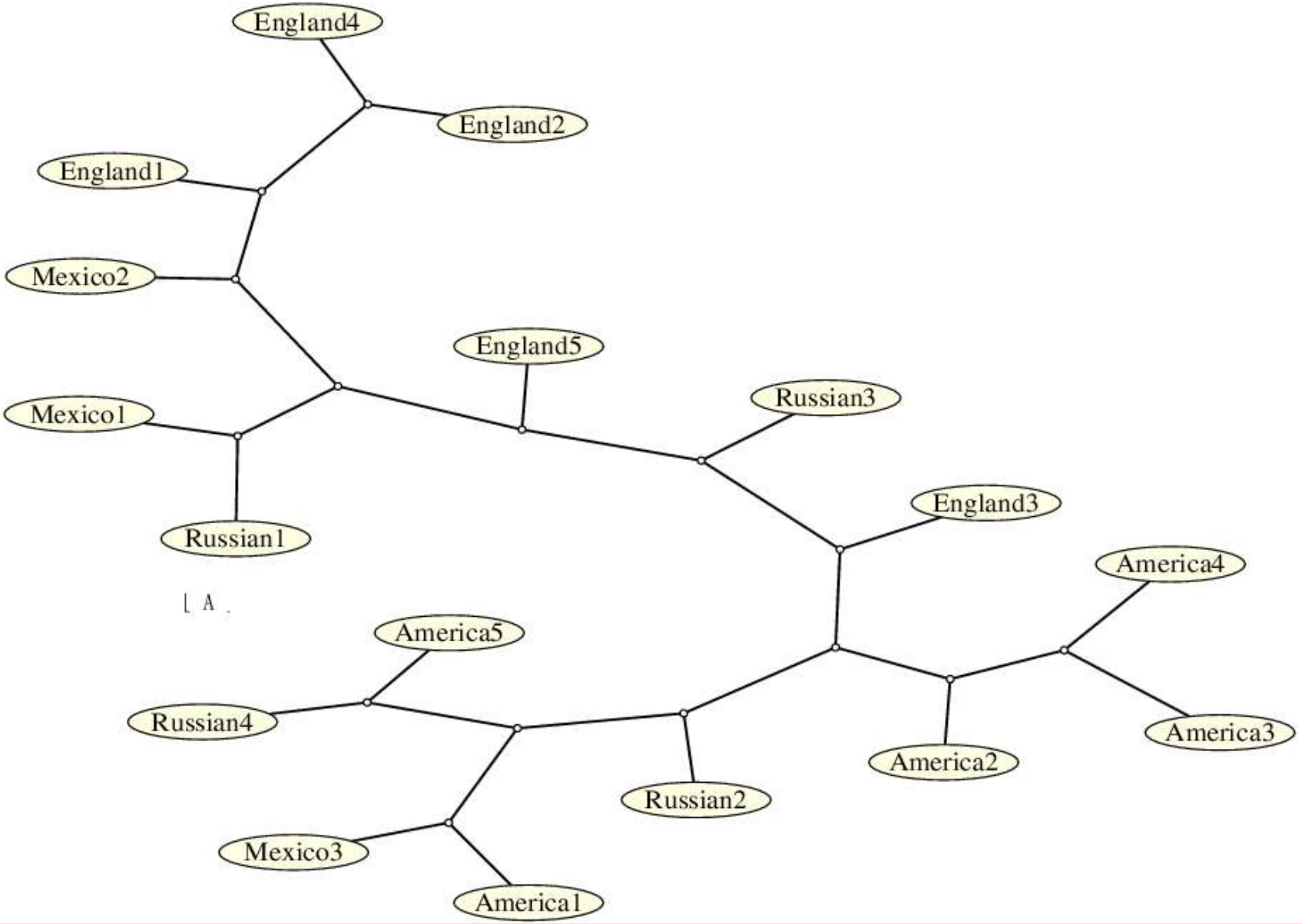
S(T)=0.958



Rudiらの実験より抜粋

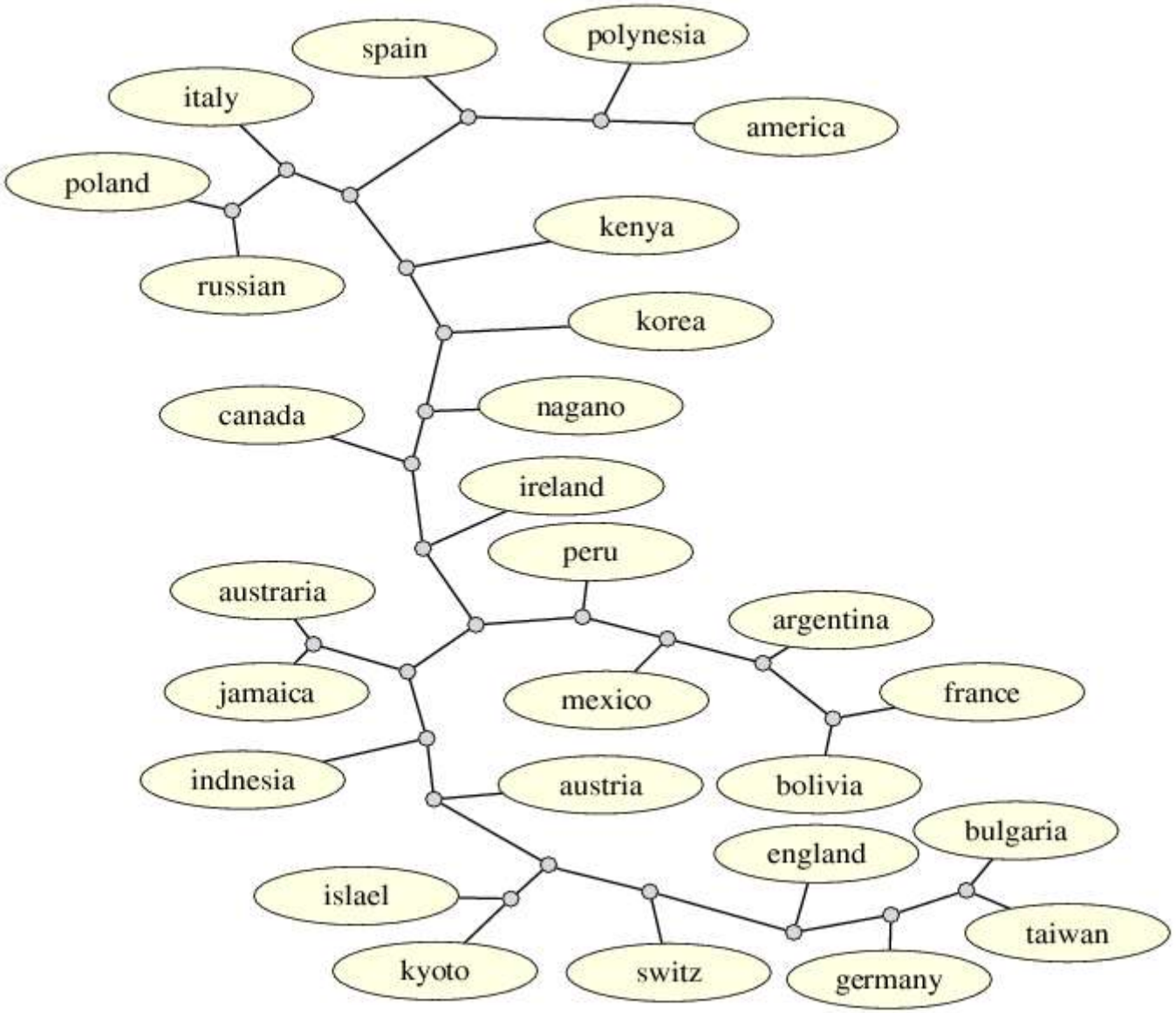
実験　～実験Ⅱ～

S(T)=0.76698



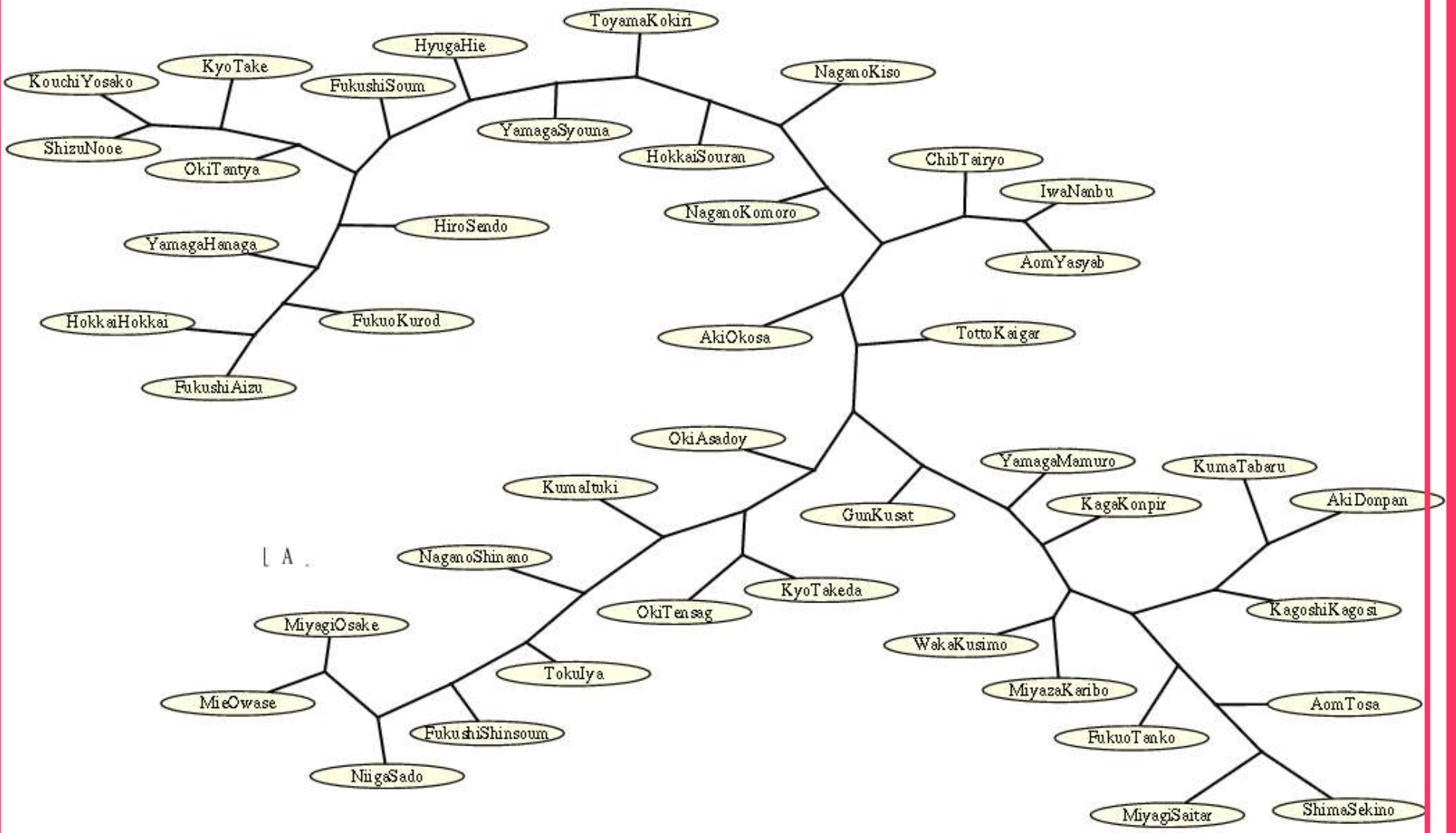
実験　～実験Ⅱ～

S(T)=0.64777



実験　～実験Ⅲ～

S(T)=0.4612



もくじ

- はじめに
- 実験方法
  - 前処理
  - similarity metric
  - quartet method
- 実験
- 今後の課題

今後の課題

- 今後の課題
  - tranceformを行う際に条件を付ける
  - 圧縮方法を変える
  - 前処理方法
  - 新たなターゲット(民謡の歌詞・文章)

- 活用例
  - データ探索への活用(Webなど)
  - 複製したデータの発見

おしまい。