単一の連からなる Run-Based Trie のリストによるパケット分類法

原田 崇司[1]　石川 裕樹[1]　田中 賢[1]　三河 賢治[2]

[1] 神奈川大学大学院 理学研究科 理学専攻 情報科学領域

[2] 新潟大学学術情報機構情報基盤センター

2018 年 3 月 08 日

AL, サンポートホール高松

目次

目次

パケット分類

研究背景

Run-Based Trie

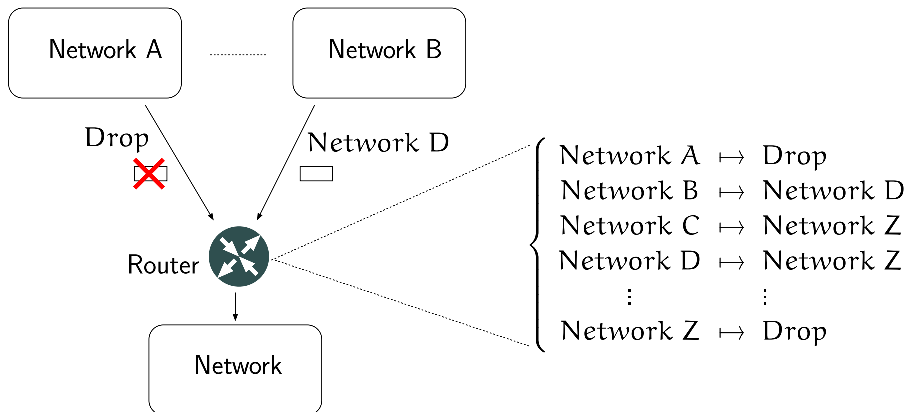
C1P

提案手法

計算機実験

まとめと課題

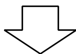
パケット分類



ポリシに従ってパケットを分類

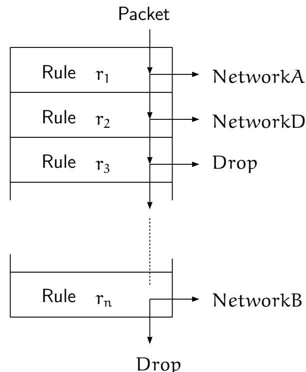
パケット分類

ポリシを満たすルールリストを作成



このルールリストによってパケットを分類

($r_1, r_2, \ldots, r_n$ の順でパケットと照合, 最初に合致したルールのアクションを適用)



$r_1 \ldots r_n$ に合致しないパケットは無条件に Deny を適用

ルールの形式

一般にパケット分類には，パケットヘッダの

送信元アドレス	(e.g. 131.10.42.40)
宛先アドレス	(e.g. 95.184.130.35)
送信元ポート番号	(e.g. 2020)
宛先ポート番号	(e.g. 22)
プロトコル	(e.g. TCP)

を使用するので，パケット分類のルールは，これら 5 **つの項目を指定**

(e.g. r_1 : 131.10.42.40/32, 95.184.130.35/32, 0 : 65535, 1724 : 1724, UDP)

抽象化してパケット（ヘッダ）を $0, 1$ **の系列，ルールを** $0, 1, *$ **の系列とみる**

ルールの形式

- ルールは,
  - ルール番号 $i \in \mathbb{N}$
  - 条件 $c \in \{0, 1, *\}^w$

  の組．ただし，'$*$' は don't care
- パケットは長さ w のビットパターン列

	Filter $\mathcal{R}$
r_1	0 * 1 *
r_2	0 0 0 0
r_3	* 0 0 *
r_4	* 1 * 0
r_5	1 * 1 *
r_6	* * 1 *

e.g. パケット 1111 は，$r_1 \ldots r_4$ に合致せず r_5 に合致

⟹ r_5 のアクションを適用（ルール番号 5 を返す）

パケット分類問題

パケット分類問題は右表のようなルールのリストで定義される関数

$$f : \{0, 1\}^w \to \{1, 2, \ldots, n + 1\}$$

を計算する問題

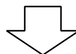
	Filter $\mathcal{R}$
r_1	$0 * 1 * \cdots 0 *$
r_2	$* 0 0 0 \cdots 1 1$
r_3	$1 * 0 1 \cdots * 1$
r_4	$1 1 * * \cdots 0 1$
$\vdots$	$\vdots$
r_{n-1}	$0 * * 1 \cdots * *$
r_n	$* 1 * * \cdots * *$

- '$*$' は don't care，即ち'0' でも'1' でも良い
- n はリスト $\mathcal{R}$ の長さ，w はルールの条件 $c \in \{0, 1, *\}^w$ の長さ
- f は長さ w の $0, 1$ の系列を取り，最初に合致するルールの番号を返す

研究背景

ルールリストのルール数 n は増える一方

線型探索によるパケット分類はルール数 n に依存するので遅い



ルール数 n に依存せずパケット分類を行いたい

既存研究

任意の位置に '∗' を含むルールに対応するアルゴリズム

1. Grouper (Ligatti et al. ' 10)
2. Run-Based Trie **による** Simple Search(**三河ら** ' 15)
3. Run-Based Trie **から構築した決定木による探索** (**三河ら** ' 15)
4. **単一の連からなる** Run-Based Trie (**原田ら** ' 17)

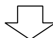
- 1., 3. **は領域計算量が指数オーダ**
- 2. **は探索時間がルール数** n **に依存**
- 4 **はルールリストが** C1P **を満たさなければ使えない**

研究目的

任意の位置に '$*$' を含むルールに対応するアルゴリズム

- "Grouper" と「RBT の決定木探索」は領域計算量が指数オーダ
- RBT 探索は探索時間がルール数 n に依存
- 単一の連からなる RBT はルールリストが C1P でないとダメ

"Grouper" と「RBT の決定木探索」の改善は困難



ルールリスト $\mathcal{R}$ を C1P を満たす k 個のルールリスト $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_k$, へ分割

Run-Based Trie (RBT)

RBT は $f : \{0,1\}^w \to \{1, 2, \ldots, n+1\}$ を計算するためのデータ構造

f の計算には '$*$' 以外の 0, 1 の部分 (赤の部分) で充分
(r_1 と合致するかに 1, 3 ビット目は無関係)

0, 1 の部分にどうやって注目するか？

0, 1 が連続している部分（連）ごとに注目

r_1	$*$ 0 $*$ 1
r_2	0 0 0 0
r_3	0 $*$ 0 0
r_4	0 $*$ 1 $*$
r_5	$*$ 1 $*$ 1
r_6	$*$ $*$ $*$ 1

0, 1 が連続している部分（連）の例

- r_2 の 1 ビット目から 4 ビット目の 0000
- r_3 の 1 ビット目からの 0 と 3 ビット目からの 00

Run-Based Trie (RBT)

RBT は $f : \{0,1\}^w \to \{1,2,\ldots,n+1\}$ を計算するためのデータ構造

f の計算には '$*$' 以外の 0, 1 の部分 (赤の部分) で充分
(r_1 と合致するかに 1, 3 ビット目は無関係)

0, 1 の部分にどうやって注目するか？

0, 1 が連続している部分（連）ごとに注目

r_1	$*$ 0 $*$ 1
r_2	0 0 0 0
r_3	0 $*$ 0 0
r_4	0 $*$ 1 $*$
r_5	$*$ 1 $*$ 1
r_6	$*$ $*$ $*$ 1

0, 1 が連続している部分（連）の例

- r_2 の 1 ビット目から 4 ビット目の 0000
- r_3 の 1 ビット目からの 0 と 3 ビット目からの 00

Run-Based Trie (RBT)

RBT は $f:\{0,1\}^w \rightarrow \{1,2,\ldots,n+1\}$ を計算するためのデータ構造

f の計算には '$*$' 以外の 0, 1 の部分 (赤の部分) で充分
(r_1 と合致するかに 1, 3 ビット目は無関係)

0, 1 の部分にどうやって注目するか？

0, 1 が連続している部分（連）ごとに注目

r_1	$*\ 0\ *\ 1$
r_2	$0\ 0\ 0\ 0$
r_3	$0\ *\ 0\ 0$
r_4	$0\ *\ 1\ *$
r_5	$*\ 1\ *\ 1$
r_6	$*\ *\ *\ 1$

0, 1 が連続している部分（連）の例

- r_2 の 1 ビット目から 4 ビット目の 0000
- r_3 の 1 ビット目からの 0 と 3 ビット目からの 00

Run-Based Trie (RBT)

RBT は $f : \{0, 1\}^w \to \{1, 2, \ldots, n+1\}$ を計算するためのデータ構造

f の計算には '$*$' 以外の 0, 1 の部分 (赤の部分) で充分
(r_1 と合致するかに 1, 3 ビット目は無関係)

0, 1 の部分にどうやって注目するか？

0, 1 が連続している部分（連）ごとに注目

r_1	$*$ 0 $*$ 1
r_2	0 0 0 0
r_3	0 $*$ 0 0
r_4	0 $*$ 1 $*$
r_5	$*$ 1 $*$ 1
r_6	$*$ $*$ $*$ 1

0, 1 が連続している部分（連）の例

- r_2 の 1 ビット目から 4 ビット目の 0000
- r_3 の 1 ビット目からの 0 と 3 ビット目からの 00

連に注目して探索

RBT は $f : \{0,1\}^w \to \{1,2,\ldots,n+1\}$ を計算するためのデータ構造

入力系列 $\alpha \in \{0,1\}^w$ がどの連に合致するかを探索

α がどの連に合致するかをどうやって調べるか...

$1,2,\ldots,w$ ビット目毎に Trie 木を構築

w 本のトライを一つの BDD のようなグラフへ

構築したグラフを探索

r_1	* 0 * 1
r_2	0 0 0 0
r_3	0 * 0 0
r_4	0 * 1 *
r_5	* 1 * 1
r_6	* * * 1

連に注目して探索

RBT は $f:\{0,1\}^w \to \{1,2,\ldots,n+1\}$ を計算するためのデータ構造

入力系列 $\alpha \in \{0,1\}^w$ がどの連に合致するかを探索

α がどの連に合致するかをどうやって調べるか…

$1,2,\ldots,w$ ビット目毎に Trie 木を構築

w 本のトライを一つの BDD のようなグラフへ

構築したグラフを探索

r_1	*	0	*	1
r_2	0	0	0	0
r_3	0	*	0	0
r_4	0	*	1	*
r_5	*	1	*	1
r_6	*	*	*	1

連に注目して探索

RBT は $f : \{0, 1\}^w \to \{1, 2, \ldots, n+1\}$ を計算するためのデータ構造

入力系列 $\alpha \in \{0, 1\}^w$ がどの連に合致するかを探索

α がどの連に合致するかをどうやって調べるか…

$1, 2, \ldots, w$ ビット目毎に Trie 木を構築

w 本のトライを一つの BDD のようなグラフへ

構築したグラフを探索

r_1	*	0	*	1
r_2	0	0	0	0
r_3	0	*	0	0
r_4	0	*	1	*
r_5	*	1	*	1
r_6	*	*	*	1

連に注目して探索

RBT は $f : \{0,1\}^w \to \{1,2,\ldots,n+1\}$ を計算するためのデータ構造

入力系列 $\alpha \in \{0,1\}^w$ がどの連に合致するかを探索

α がどの連に合致するかをどうやって調べるか...

$1,2,\ldots,w$ ビット目毎に Trie 木を構築

w 本のトライを一つの BDD のようなグラフへ

構築したグラフを探索

r_1	$* \ 0 \ * \ 1$
r_2	$0 \ 0 \ 0 \ 0$
r_3	$0 \ * \ 0 \ 0$
r_4	$0 \ * \ 1 \ *$
r_5	$* \ 1 \ * \ 1$
r_6	$* \ * \ * \ 1$

連に注目して探索

RBT は $f : \{0,1\}^w \to \{1,2,\ldots,n+1\}$ を計算するためのデータ構造

入力系列 $\alpha \in \{0,1\}^w$ がどの連に合致するかを探索

α がどの連に合致するかをどうやって調べるか…

$1,2,\ldots,w$ ビット目毎に Trie 木を構築

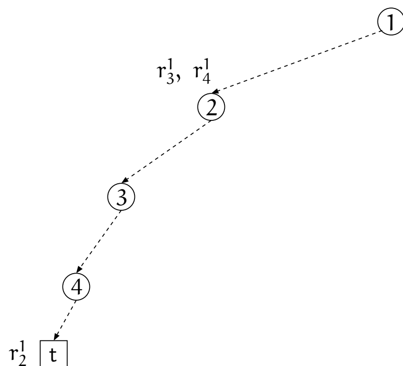
w 本のトライを一つの BDD のようなグラフへ

構築したグラフを探索

r_1	*	0	*	1
r_2	0	0	0	0
r_3	0	*	0	0
r_4	0	*	1	*
r_5	*	1	*	1
r_6	*	*	*	1

Run-Based Trie 構築

右のルールリストから Run-Based Trie を構成



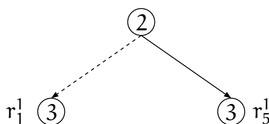
r_1	* 0 * 1
r_2	0 0 0 0
r_3	0 * 0 0
r_4	0 * 1 *
r_5	* 1 * 1
r_6	* * * 1

1 ビット目の連で T_1 を構成

r_i^j はルール r_i の j 個目の連

Run-Based Trie 構築

右のルールリストから Run-Based Trie を構成



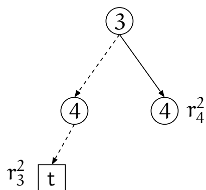
r_1	* 0 * 1
r_2	0 0 0 0
r_3	0 * 0 0
r_4	0 * 1 *
r_5	* 1 * 1
r_6	* * * 1

2 ビット目の連で T_2 を構成

r_i^j はルール r_i の j 個目の連

Run-Based Trie 構築

右のルールリストから Run-Based Trie を構成



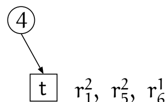
r_1	$* \ 0 \ * \ 1$
r_2	$0 \ 0 \ 0 \ 0$
r_3	$0 \ * \ 0 \ 0$
r_4	$0 \ * \ 1 \ *$
r_5	$* \ 1 \ * \ 1$
r_6	$* \ * \ * \ 1$

3 ビット目の連で T_3 を構成

r_i^j はルール r_i の j 個目の連

Run-Based Trie 構築

右のルールリストから Run-Based Trie を構成



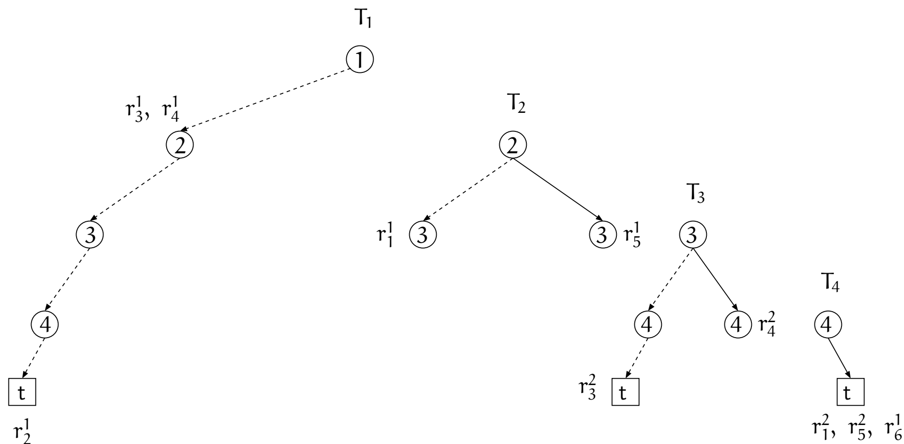
r_1	$*\ 0\ *\ 1$
r_2	$0\ 0\ 0\ 0$
r_3	$0\ *\ 0\ 0$
r_4	$0\ *\ 1\ *$
r_5	$*\ 1\ *\ 1$
r_6	$*\ *\ *\ 1$

4 ビット目の連で T_4 を構成

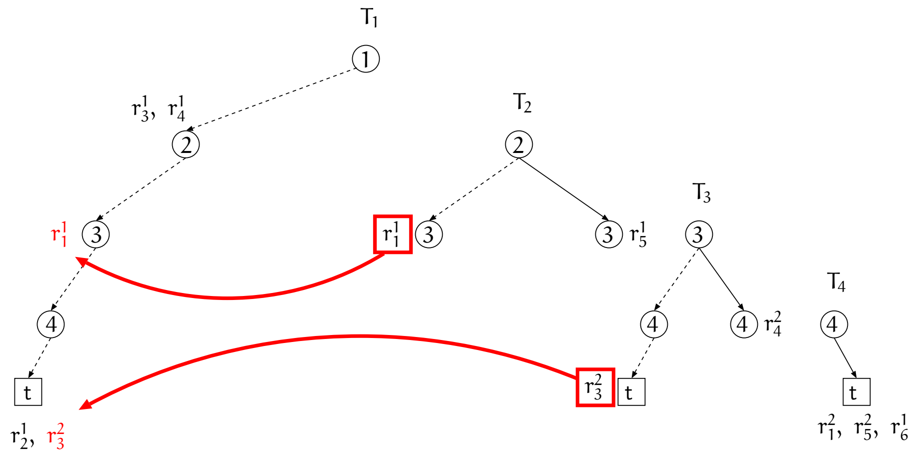
r_i^j はルール r_i の j 個目の連

Run-Based Trie 構築

前スライドのルールリストから構成される T_1, T_2, T_3, T_4

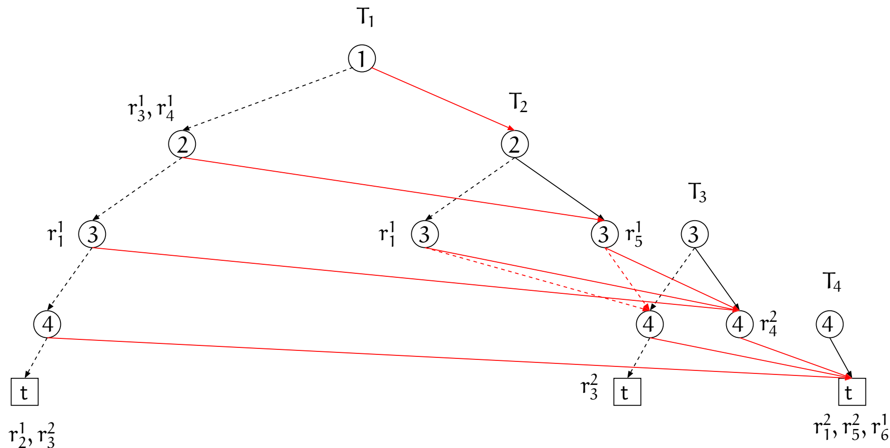


下位トライの連を上位トライへ付与



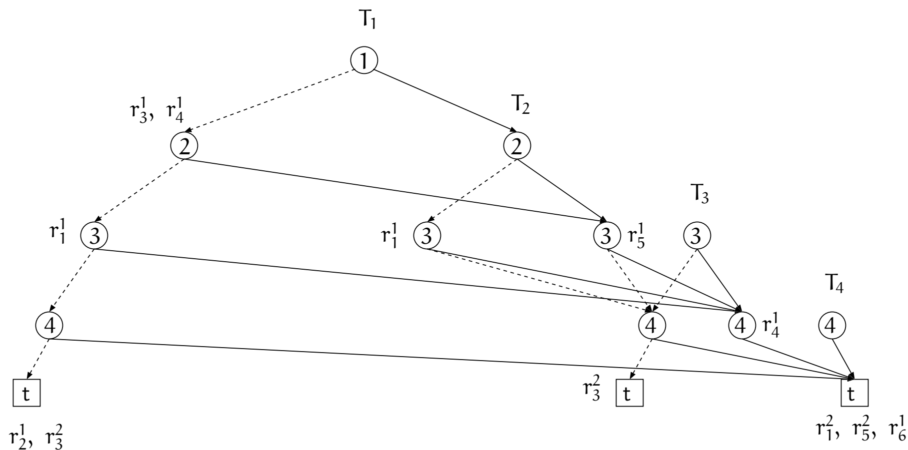
T_1 の 0 節点と T_2 の根，T_1 の 00 節点と T_3 の根を重ねる

上位トライから下位トライへの枝を追加

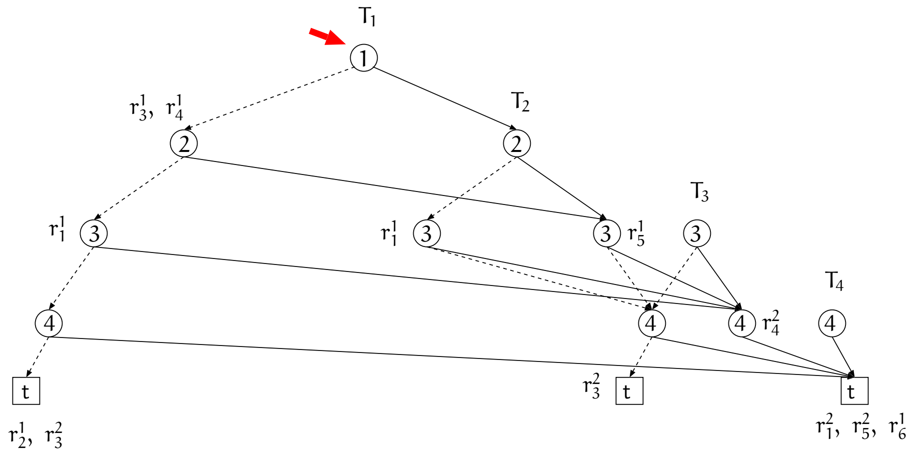


例：T_1 の 00，T_2 の 0, 1 節点から T_3 の 1 節点への枝

連と枝を追加した Run-Based Trie

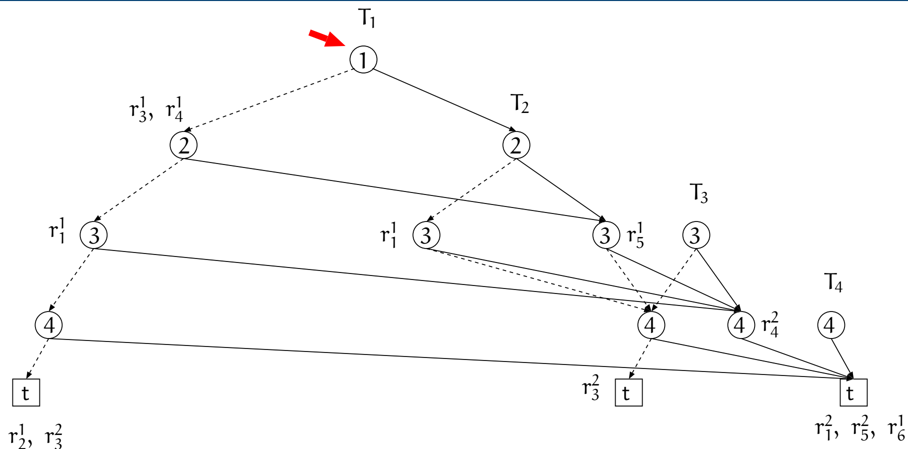


Run-Based Trie 探索



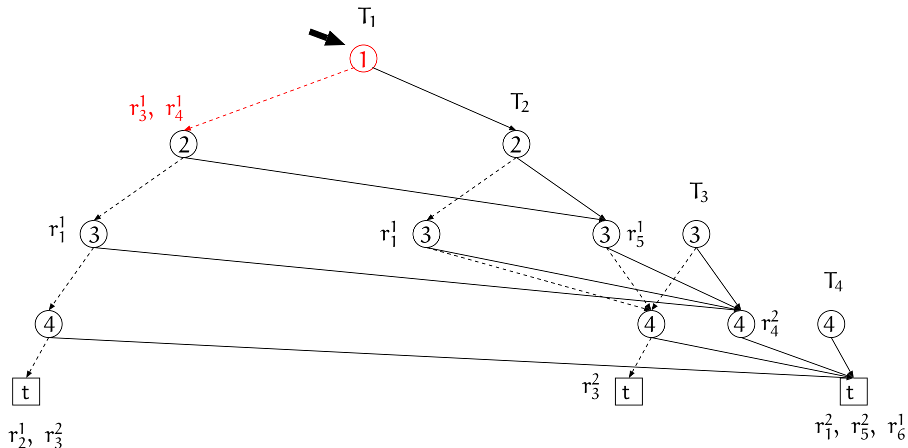
長さ n の配列 A と変数 B を用意し，T_1 の根から探索

Run-Based Trie 探索の例：0011 を探索

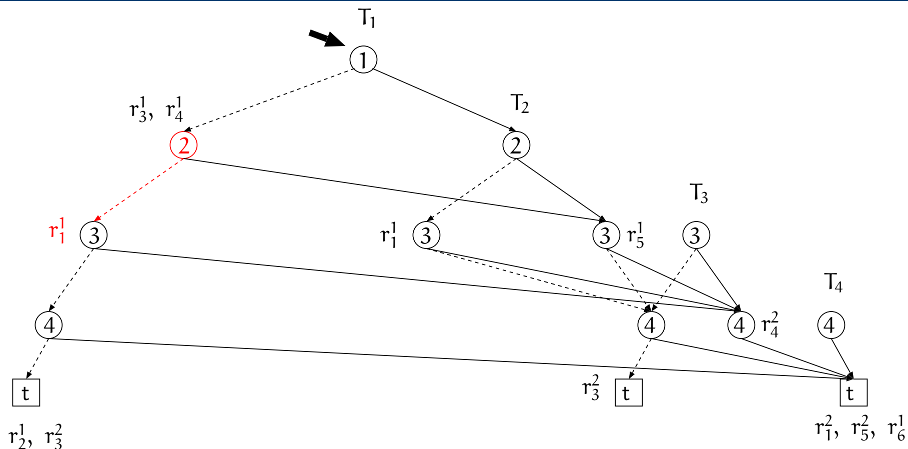


$B = 7$ A | 0 | 0 | 0 | 0 | 0 | 0 | $\alpha = 0011$

Run-Based Trie 探索の例：0011 を探索

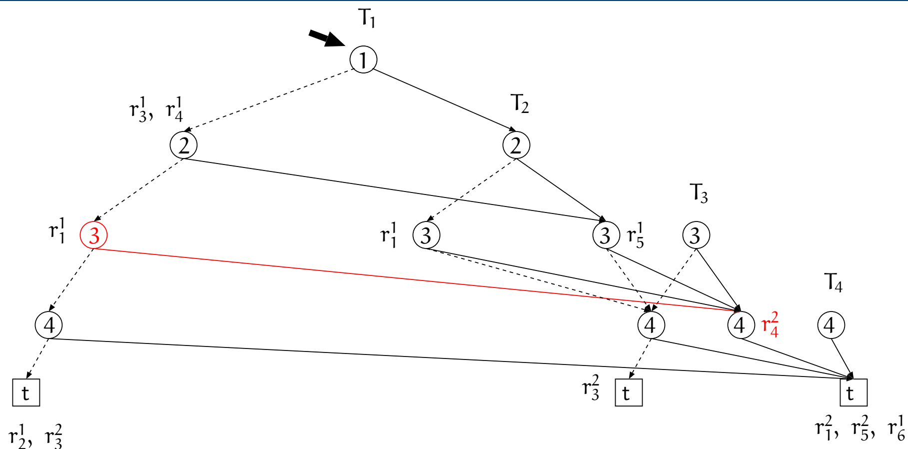


Run-Based Trie 探索の例：0011 を探索



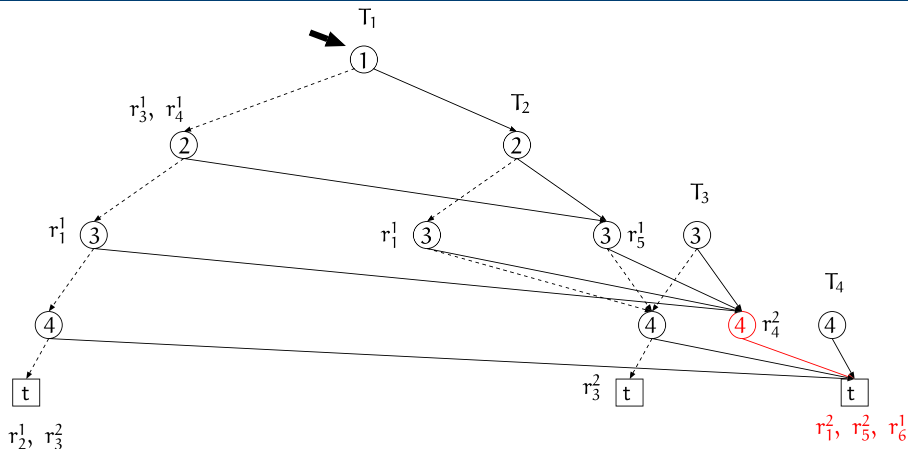
$B = 7$ A | 1 | 0 | 1 | 1 | 0 | 0 | $\alpha = 0011$

Run-Based Trie 探索の例：0011 を探索



$B = 4$ A | 1 | 0 | 1 | 2 | 0 | 0 | $\alpha = 0011$

Run-Based Trie 探索の例：0011 を探索



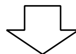
$\underline{B = 1}$ A

2	0	1	2	1	1

$\alpha = 0011$

RBT 探索の時間計算量

- **入力系列 α で RBT を辿る計算量 $O(w)$**
- **α に合致した連の照合，最優先ルールとの比較の計算量 $O(nw)$ (連の数は高々$nw/2$)**

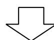


RBT **探索の時間計算量は** $O(nw)$

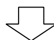
探索時間が n に依存しないアルゴリズムが欲しい

ルール数 n に依存しない探索法

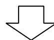
一つのルール r_i が複数の連 $r_i^1, r_i^2, \ldots, r_i^k$ を持つ可能性がある



連に合致する度に照合を行わなければならないので探索時間が n に依存



各ルールの連が一つだけならば，連に合致する度の照合は不要



RBT 探索の時間計算量が n に依存せず $O(w)$

単一の連からなる Run-Based Trie

r_1	$* \; 0 \; * \; 1$
r_2	$0 \; 0 \; 0 \; 0$
r_3	$0 \; * \; 0 \; 0$
r_4	$0 \; * \; 1 \; *$
r_5	$* \; 1 \; * \; 1$
r_6	$* \; * \; * \; 1$

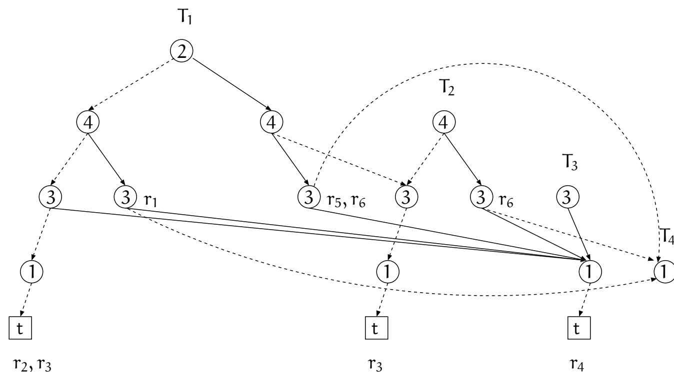
$\downarrow \quad (4\ 1\ 3\ 2)$

r_1	$0 \; 1 \; * \; *$
r_2	$0 \; 0 \; 0 \; 0$
r_3	$* \; 0 \; 0 \; 0$
r_4	$* \; * \; 1 \; 0$
r_5	$1 \; 1 \; * \; *$
r_6	$* \; 1 \; * \; *$

各ルール r_i の連の数が一つだけだと

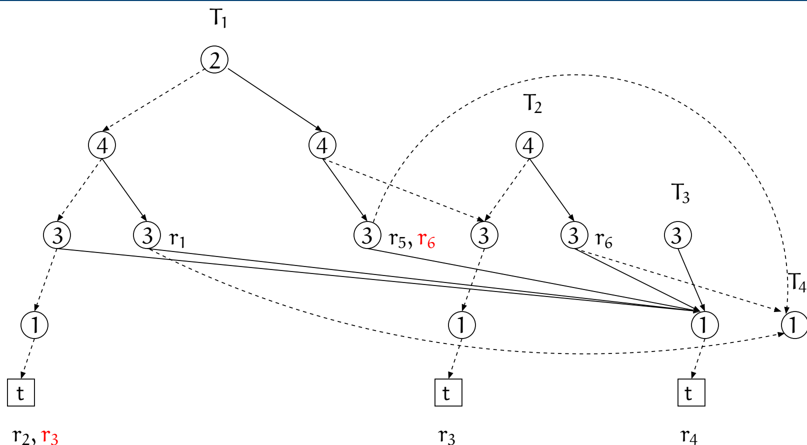
1. **節点 v に r_i, r_j $(i < j)$ がある場合，r_j は不要**
2. **連 r_j をもつ節点 v_j の b 枝から，連 r_i $(i < j)$ をもつ節点に到達不可能な場合，b 枝は不要**

単一の連からなる Run-Based Trie



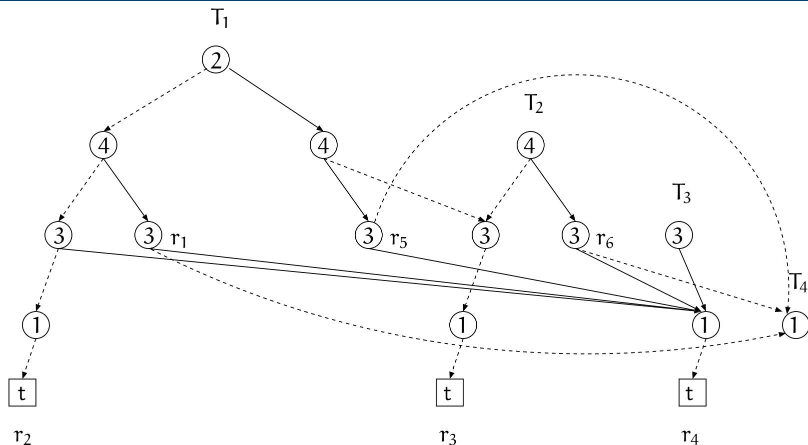
	2	4	3	1
r_1	0	1	*	*
r_2	0	0	0	0
r_3	*	0	0	0
r_4	*	*	1	0
r_5	1	1	*	*
r_6	*	1	*	*

単一の連からなる Run-Based Trie



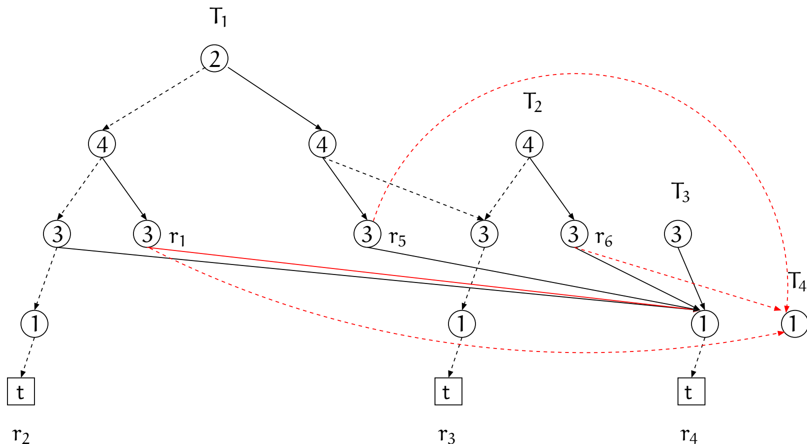
節点 v に r_i, r_j $(i < j)$ がある場合，r_j は不要

単一の連からなる Run-Based Trie



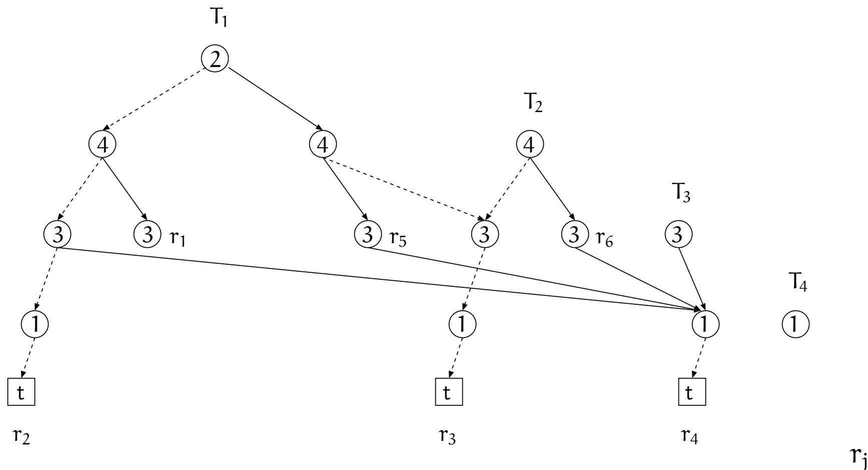
節点 v **に** r_i, r_j $(i < j)$ **がある場合，** r_j **は不要**

単一の連からなる Run-Based Trie



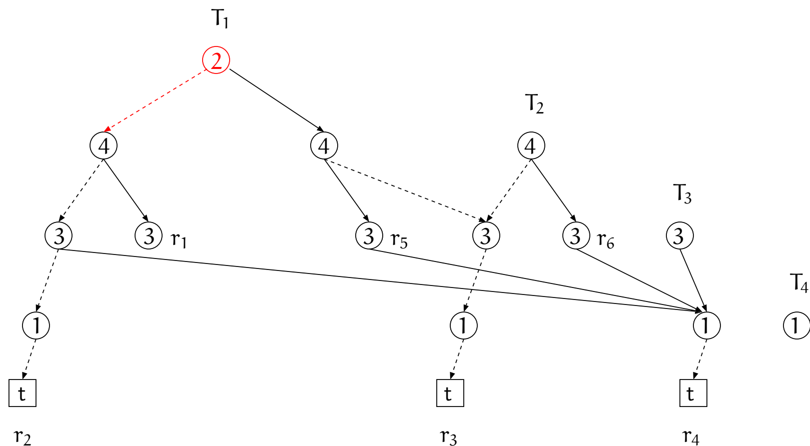
r_j の節点 v_j の b 枝から r_i $(i < j)$ の節点に到達不能なら b 枝は不要

単一の連からなる Run-Based Trie



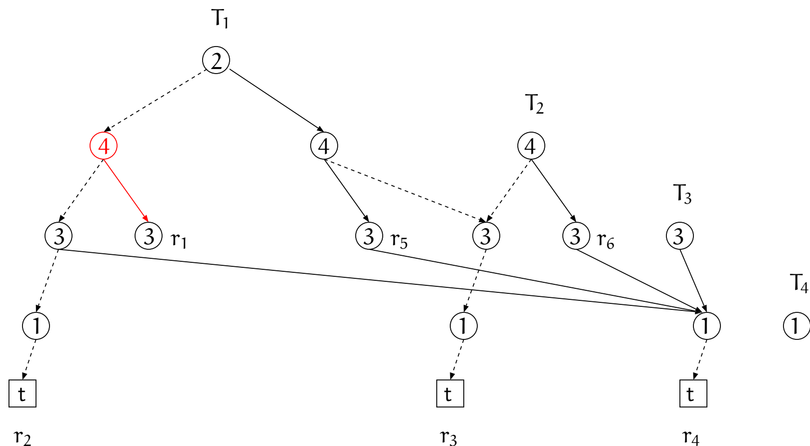
の節点 v_j の b 枝から r_i $(i < j)$ の節点に到達不能なら b 枝は不要

単一の連からなる RBT の探索の例：0011 を探索



$B = 7 \qquad \alpha = 0011$

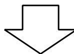
単一の連からなる RBT の探索の例：0011 を探索



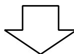
$B = 1 \qquad \alpha = 0011$

単一の連からなる RBT 探索の時間計算量

- T_1 の根からの探索経路の長さは $O(w)$
- 各節点にルールは高々一つ



照合回数は $O(w)$ で，RBT を辿るのも $O(w)$



単一の連からなる RBT 探索の時間計算量は n に依存せず $O(w)$

Consecutive Ones Property (C1P)

Consecutive Ones Property (C1P)

ブール行列 M が C1P を満たす
$\iff$ M の各行の 1 が連続するように M の列を並び替えられる

Table: C1P を満たす

c_1	c_2	c_3	c_4
1	0	1	0
1	1	1	1
0	1	1	0
0	1	0	1
1	0	1	0
0	0	1	0

Table: C1P を満たさない

c_1	c_2	c_3	c_4
1	1	0	0
1	1	1	1
0	1	1	1
0	0	1	1
0	1	1	0
0	1	0	1

Consecutive Ones Property (C1P)

Consecutive Ones Property (C1P)

ブール行列 M が C1P を満たす
$\iff$ M の各行の 1 が連続するように M の列を並び替えられる

Table: C1P を満たす

c_1	c_3	c_2	c_4
1	1	0	0
1	1	1	1
0	1	1	0
0	0	1	1
1	1	0	0
0	1	0	0

Table: C1P を満たさない

c_1	c_2	c_3	c_4
1	1	0	0
1	1	1	1
0	1	1	1
0	0	1	1
0	1	1	0
0	1	0	1

C1P 判定

文献 (Habib '00) **の定理** 2 **より，**

$G(\tilde{M})$ **が区間グラフ** $\Rightarrow$ $m \times n$ **のブール行列** M **が** C1P **を満たす**

ただし， $\tilde{M} = \begin{pmatrix} M \\ I \end{pmatrix}$,

$V(G(\tilde{M})) = \{r_1, r_2, \ldots, r_m\}$, $E(G(\tilde{M})) = \{r_i r_j \mid \exists k \ r_{ik} = r_{jk} = \text{'1'}\}$

文献 (Li '14) **より，区間グラフの判定は線型オーダで可能**
(G **が区間グラフならば，** G **の** I-ordering **を返す**)

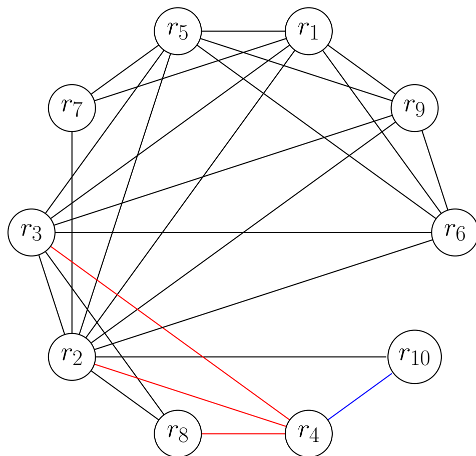
⇨ '0','1' **を** '1'**，** '$*$' **を** '0' **と見做し，ルールリスト** $\mathcal{R}$ **から行列** $M_{\mathcal{R}}$ **を生成**

⇨ **行列** $M_{\mathcal{R}}$ **から** $G(\tilde{M}_{\mathcal{R}})$ **を生成，** $G(\tilde{M}_{\mathcal{R}})$ **が区間グラフか判定**

行列 M に対するグラフ $G(\tilde{M})$

$$V(G(\tilde{M})) = \{r_1, r_2, \ldots, r_m\},\ E(G(\tilde{M})) = \{r_i r_j \mid \exists k\ r_{ik} = r_{jk} = \text{'1'}\}$$

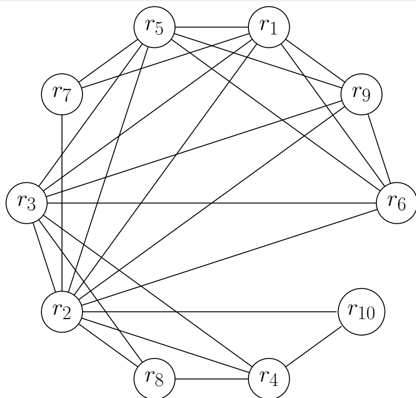
	c_1	c_2	c_3	c_4
r_1	1	0	1	0
r_2	1	1	1	1
r_3	0	1	1	0
r_4	0	1	0	1
r_5	1	0	1	0
r_6	0	0	1	0
r_7	1	0	0	0
r_8	0	1	0	0
r_9	0	0	1	0
r_{10}	0	0	0	1

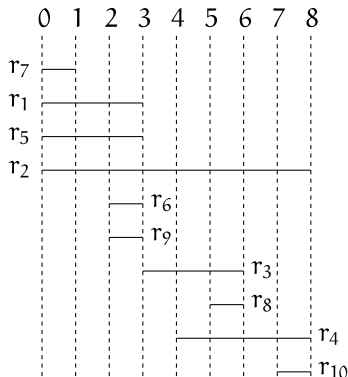


区間グラフ

定義

$G = (V, E)$ が区間グラフ $\iff \exists I. \forall x, y.\ I(x) \cap I(y) \iff xy \in E$
ただし，I は節点に対する区間の割当



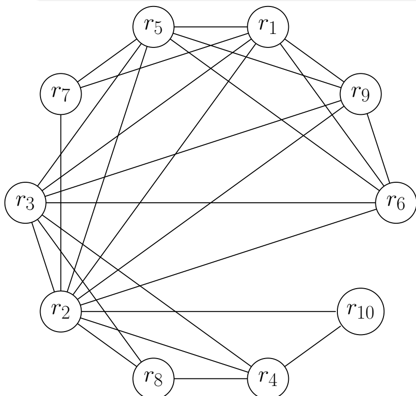


I-ordering

定義

順序 σ は $G = (V, E)$ の I-ordering

$\iff \forall i, k \in [|V|], \sigma(i)\sigma(k) \in E \Rightarrow \forall j \in [i+1, k-1] \Rightarrow \sigma(i)\sigma(j) \in E$



$$[i, j] = \{\ k \mid i \leq k \leq j,\ k \in Z\ \}$$

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 7 & 1 & 5 & 2 & 6 & 9 & 3 & 8 & 4 & 10 \end{pmatrix}$$

定理

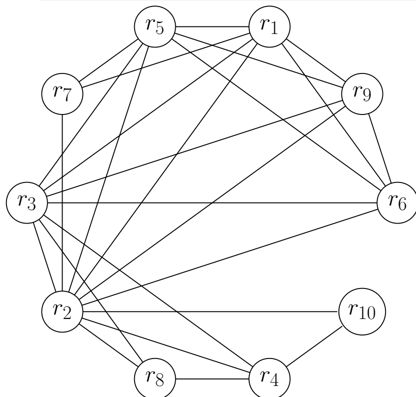
G が区間グラフ

$\iff$ G の I-ordering が存在

クリークチェイン (Clique Chain)

定義

G の極大クリークの列 $S = C_1, C_2, \ldots, C_k$ がクリークチェイン
$\iff$ 任意の節点 v に対して，v を含むクリークが S で連続



$$
\begin{aligned}
C_1 &= \{r_1, r_2, r_5, r_7\}, \\
C_2 &= \{r_2, r_3, r_4, r_8\}, \\
C_3 &= \{r_1, r_2, r_3, r_5, r_6, r_9\}, \\
C_4 &= \{r_2, r_4, r_{10}\}
\end{aligned}
$$

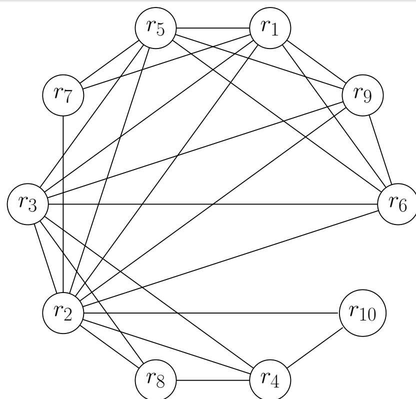
$$S = [C_1, C_3, C_2, C_4]$$

クリークチェイン (Clique Chain)

定理

$\tilde{M}$ の各列 c_j に対して，$\{ r_i \mid c_{ij} = 1 \}$ は $G(\tilde{M})$ の極大クリーク

	c_1	c_2	c_3	c_4
r_1	1	0	1	0
r_2	1	1	1	1
r_3	0	1	1	0
r_4	0	1	0	1
r_5	1	0	1	0
r_6	0	0	1	0
r_7	1	0	0	0
r_8	0	1	0	0
r_9	0	0	1	0
r_{10}	0	0	0	1



クリークチェイン (Clique Chain) と C1P

定理

$\tilde{M}$ の列に対するクリークチェインの順序は，1 を連続させる

	c_1	c_2	c_3	c_4
r_1	1	0	1	0
r_2	1	1	1	1
r_3	0	1	1	0
r_4	0	1	0	1
r_5	1	0	1	0
r_6	0	0	1	0
r_7	1	0	0	0
r_8	0	1	0	0
r_9	0	0	1	0
r_{10}	0	0	0	1

$$
\begin{aligned}
C_1 &= \{r_1, r_2, r_5, r_7\},\\
C_2 &= \{r_2, r_3, r_4, r_8\},\\
C_3 &= \{r_1, r_2, r_3, r_5, r_6, r_9\},\\
C_4 &= \{r_2, r_4, r_{10}\}
\end{aligned}
$$

$$S = [C_1, C_3, C_2, C_4] \ ([c_1, c_3, c_2, c_4])$$

クリークチェイン (Clique Chain) と C1P

定理

$\tilde{M}$ の列に対するクリークチェインの順序は，1 を連続させる

	c_1	c_3	c_2	c_4
r_1	1	1	0	0
r_2	1	1	1	1
r_3	0	1	1	0
r_4	0	0	1	1
r_5	1	1	0	0
r_6	0	1	0	0
r_7	1	0	0	0
r_8	0	0	1	0
r_9	0	1	0	0
r_{10}	0	0	0	1

$$\begin{aligned} C_1 &= \{r_1, r_2, r_5, r_7\}, \\ C_2 &= \{r_2, r_3, r_4, r_8\}, \\ C_3 &= \{r_1, r_2, r_3, r_5, r_6, r_9\}, \\ C_4 &= \{r_2, r_4, r_{10}\} \end{aligned}$$

$$S = [C_1, C_3, C_2, C_4] \ ([c_1, c_3, c_2, c_4])$$

I-ordering $\Rightarrow$ Clique Chain

$\tilde{M}$ に対する I-ordering σ により，$G(\tilde{M})$ の極大クリーク上の順序を定義

$$X <_\sigma Y \equiv \sigma\left(\min\{\sigma^{-1}(x) \mid x \in X \triangle Y\}\right) \in X$$

ただし，$X \triangle Y = (X \setminus Y) \cup (Y \setminus X)$

$\tilde{M}$ の極大クリーク (列) を $<_\sigma$ で整列したものはクリークチェイン

e.g. $X = \{1, 2, 5, 7\}, Y = \{1, 2, 3, 5, 6, 9\},$
$\sigma = (7\ 1\ 5\ 2\ 6\ 9\ 3\ 8\ 4\ 10),\ \sigma^{-1} = (2\ 4\ 7\ 9\ 3\ 5\ 1\ 8\ 6\ 10)$

$$\min\{\sigma^{-1}(x) \mid x \in X \triangle Y\} = \min\{\sigma^{-1}(x) \mid x \in \{3, 6, 7, 9\}\} = \min\{7, 5, 1, 6\}$$

かつ，$sigma(1) = 7 \in X$ より，$X <_\sigma Y$

ルールリスト $\mathcal{R} \Rightarrow$ 単一の連からなるルールリスト $\mathcal{R}'$

ルールリスト $\mathcal{R}$ **が** C1P **を満たす（各行の** 1 **を連続させる列の順序あり）**

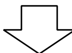
$\equiv$ $\mathcal{R}$ **の** '0', '1' **を** '1' **に** '$*$' **を** '0' **としたブール行列** M **が** C1P **を満たす**

1. **ルールリスト** $\mathcal{R}$ **からブール行列** $M_{\mathcal{R}}$ **を生成**
2. $M_{\mathcal{R}}$ **から** $\tilde{M}_{\mathcal{R}}$ **を生成**
3. $\tilde{M}_{\mathcal{R}}$ **から** $G(\tilde{M}_{\mathcal{R}})$ **を生成**
4. **文献** (Li '14) **のアルゴリズムより，** $G(\tilde{M}_{\mathcal{R}})$ **に対する順序** σ **を計算**
5. σ **が** I-ordering **ならば** $\mathcal{R}$ **は** C1P **を満たす**
6. **順序** $<_{\sigma}$ **により，** $G(\tilde{M}_{\mathcal{R}})$ **のクリークチェイン** S **を計算**
7. S **に従って** $\mathcal{R}$ **の列を並び替え**

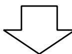
提案手法概略

ルールリスト $\mathcal{R}$ が C1P を満たせば，ルール数 n に依存しない探索が可能

入力ルールリストは一般に C1P を満たさない



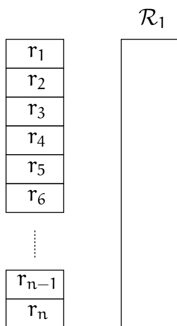
$\mathcal{R}$ を C1P を満たす部分ルールリスト $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_k$ へ分割



$\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_k$ を用いてパケットを分類

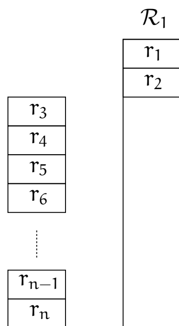
(分割数 k が n に依存しなければ，ルール数に依存しない探索手法だが..)

ルールリスト分割



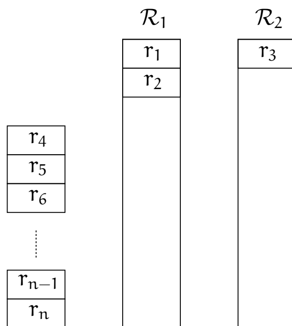
1. r_1, r_2 を $\mathcal{R}_1$ へ追加； $i \leftarrow 3, j \leftarrow 1$， $i = n$ まで下記の操作を繰り返す
2. $\{r_i\} \cup \mathcal{R}_j$ が C1P となる j を探し，$\{r_i\} \cup \mathcal{R}_j$; $i \leftarrow i + 1$.
   そのような j がなければ，$\mathcal{R}_{j+1} (= \{r_i\})$ を作成； $i \leftarrow i + 1$.

ルールリスト分割



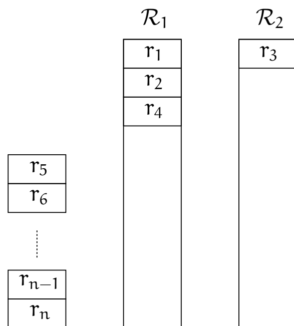
1. r_1, r_2 を $\mathcal{R}_1$ へ追加； $i \leftarrow 3, j \leftarrow 1$, $i = n$ まで下記の操作を繰り返す
2. $\{r_i\} \cup \mathcal{R}_j$ が C1P となる j を探し，$\{r_i\} \cup \mathcal{R}_j$; $i \leftarrow i + 1$.
   そのような j がなければ，$\mathcal{R}_{j+1} (= \{r_i\})$ を作成； $i \leftarrow i + 1$.

ルールリスト分割



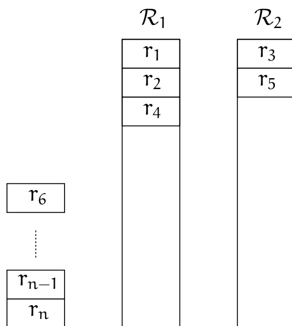
1. r_1, r_2 を $\mathcal{R}_1$ へ追加； $i \leftarrow 3, j \leftarrow 1$, $i = n$ まで下記の操作を繰り返す
2. $\{r_i\} \cup \mathcal{R}_j$ が C1P となる j を探し，$\{r_i\} \cup \mathcal{R}_j$; $i \leftarrow i + 1$.
   そのような j がなければ，$\mathcal{R}_{j+1} (= \{r_i\})$ を作成； $i \leftarrow i + 1$.

ルールリスト分割



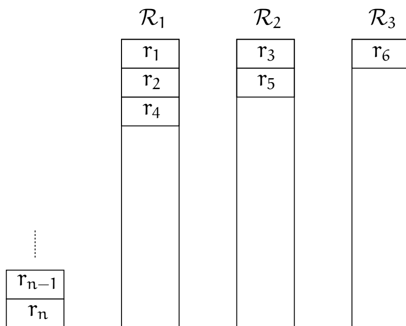
1. r_1, r_2 を $\mathcal{R}_1$ へ追加； $i \leftarrow 3, j \leftarrow 1$， $i = n$ まで下記の操作を繰り返す
2. $\{r_i\} \cup \mathcal{R}_j$ が C1P となる j を探し，$\{r_i\} \cup \mathcal{R}_j$; $i \leftarrow i + 1$.
   そのような j がなければ，$\mathcal{R}_{j+1} (= \{r_i\})$ を作成； $i \leftarrow i + 1$.

ルールリスト分割



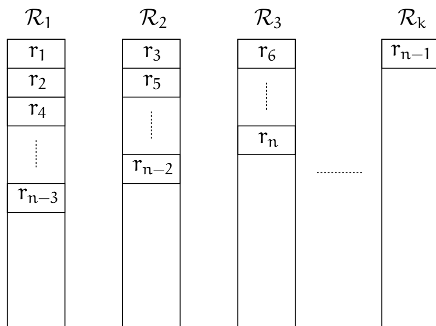
1. r_1, r_2 を $\mathcal{R}_1$ へ追加； $i \leftarrow 3, j \leftarrow 1$, $i = n$ まで下記の操作を繰り返す
2. $\{r_i\} \cup \mathcal{R}_j$ が C1P となる j を探し，$\{r_i\} \cup \mathcal{R}_j$; $i \leftarrow i + 1$.
   そのような j がなければ，$\mathcal{R}_{j+1} (= \{r_i\})$ を作成； $i \leftarrow i + 1$.

ルールリスト分割



1. r_1, r_2 を $\mathcal{R}_1$ へ追加； $i \leftarrow 3, j \leftarrow 1$, $i = n$ まで下記の操作を繰り返す
2. $\{r_i\} \cup \mathcal{R}_j$ が C1P となる j を探し，$\{r_i\} \cup \mathcal{R}_j$; $i \leftarrow i + 1$.
   そのような j がなければ，$\mathcal{R}_{j+1} (= \{r_i\})$ を作成； $i \leftarrow i + 1$.

ルールリスト分割

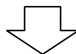


1. r_1, r_2 を $\mathcal{R}_1$ へ追加； $i \leftarrow 3, j \leftarrow 1$, $i = n$ まで下記の操作を繰り返す
2. $\{r_i\} \cup \mathcal{R}_j$ が C1P となる j を探し，$\{r_i\} \cup \mathcal{R}_j$; $i \leftarrow i + 1$.
そのような j がなければ，$\mathcal{R}_{j+1} (= \{r_i\})$ を作成； $i \leftarrow i + 1$.

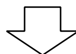
単一の連からなるRBTのリストによる探索

C1P を満たすルールリストのリスト $\mathcal{L}_{\mathcal{R}} = [\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_k]$ から

単一の連からなる RBT のリスト $\mathcal{L}_{\mathcal{F}} = [\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_k]$ を生成



入力系列で全ての単一の連からなる RBT $\mathcal{F}_i$ を探索
それぞれの $\mathcal{F}_i$ での最優先ルール c_i を取得



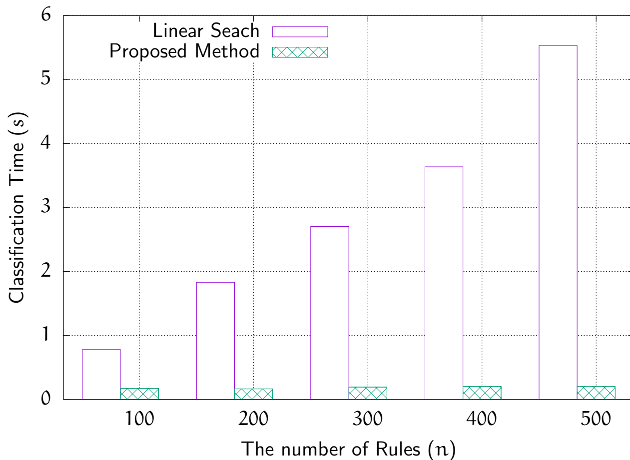
$\min\{c_1, c_2, \ldots, c_k\}$ が最優先ルールの番号，時間計算量は $O(kw)$

実験環境

OS	:	CentOS Release 6.2
CPU	:	Intel Core i7-980X 3.33 GHz
主記憶容量	:	24GB
実装言語	:	C
コンパイラ	:	gcc version 4.8.2

- **ルール長** $w = 104$, **ルール数** $n = 100, 200, \ldots, 500$ **のルールリストを** ClassBench **を用いて生成**
- **ヘッダ数が約** 10 **万のヘッダリストを各ルールリストに対して作成**
- **探索時間** (s) **を計測**

実験結果： 探索時間（秒）



探索時間がルール数に依存していない

まとめと今後の課題

まとめ

- ルールリストを C1P を満たすルールリストへ分割する手法を提案
- 提案法は ClassBench のルールリストでは探索時間が n に依存せず

今後の課題

- 線型探索以外の手法との比較
- ClassBench 以外のルールリストでの実験
- 逐次探索よりも分割数を小さくする分割法の考案
- 単一の連からなる RBT のリストの効率的な探索法の考案