Vol.2018-AL-167 No.3
2018/3/8

単一の連からなるRBTのリストによるパケット分類法

原田 崇司[1,a)]　石川 裕樹[1,b)]　田中 賢[1,c)]　三河 賢治[2,d)]

概要：パケット分類問題は，ネットワーク機器を通過するパケットと与えられているフィルタリングルールリストの各ルールとを照合して，パケットに合致する優先度の最も高いルールを求める問題である．与えられたルールリスト $\mathcal{R}$ が単一の連からなるルールのリストならば，$\mathcal{R}$ に対する RBT を生成し，この RBT を探索することにより探索時間がルール数に依存しないパケット分類を行える．本稿では次のパケット分類法を提案する．初めに，ルールリストを k 個の単一の連からなるルールリストへと分割する．そして，k 個の単一の連からなる RBT を含むリストを生成する．このリストの先頭の RBT から順に k 回 RBT 探索を行うことによってパケットを分類する．

A Packet Classification Method for a List of Run-Based Tries Consisting of a Single Run

Harada Takashi[1,a)]　Ishikawa Yuki[1,b)]　Tanaka Ken[1,c)]　Mikawa Kenji[2,d)]

1. はじめに

Run-Based Trie (RBT) は，アルファベット $\{0,1,*\}$ 上の長さ w の系列のリストによって定義された函数 $f:\{0,1\}^w \to \{1,2,\ldots,n+1\}$ を計算することができる，三河らが提案したデータ構造である [1].

函数 $f:\{0,1\}^w \to \{1,2,\ldots,n+1\}$ として表されるような問題にパケット分類問題がある．パケット分類問題は，ネットワーク機器を通過するパケットと与えられているフィルタリングルールリストの各ルールとを照合して，パケットに合致する優先度の最も高いルールを求める問題である．f をパケット分類のポリシーとみると w はパケット長で n はルール数である．パケット分類では，パケットヘッダの送信元 IP，宛先 IP，送信元ポート番号，宛先ポート番号，プロトコルの五つの性質に基づいてパケットを分類することが多い．今までに提案されてきたパケット分類手法 [2] が対象にしているルールの条件の多くは，前記のような幾つかのフィールドに分かれており，各フィールドは CIDR や `0-65535` のような範囲指定で表される．

文献 [2] にあるように今までに数多のパケット分類が提案されてきた．前述のような CIDR や範囲指定のルールを対象にしている手法の中で最先端の手法に HybridCuts[3] や井上らが提案した手法 [4] がある．HybridCuts は与えられたルールリストをその特徴に従って複数の部分ルールリストに分割し，それぞれの部分ルールリストに対して Cutting アルゴリズムを適用する手法である．井上らの手法は，Binary Decision Diagram (BDD) と Multi-valued Decision Diagram (MDD) を用いたパケット分類手法である．なお井上らの手法が扱う問題は，一つのスイッチによるパケット分類問題ではなく，複数のスイッチ間でパケットがどのうように転送されるかを特定する問題であり本稿で扱う問題とは異なる．

今後 Software Defined Network (SDN) 等の普及によってより複雑な条件に従ったパケット分類が必要となり，プレフィックスルールやレンジルールよりも表現力の強い任意のビットマスクルールに適したアルゴリズムが求められる．さらに，一般に管理者は外部の脅威が排除されたこ

[1] 神奈川大学大学院理学研究科情報科学領域
Kanagawa University, Hiratsuka, Kanagawa 259–1293, Japan
[2] 新潟大学学術情報基盤機構情報基盤センター
Niigata University, Niigata, Niigata 950–2181, Japan
[a)] takaharada@jindai.jp
[b)] ishikawa@jindai.jp
[c)] ktanaka@info.kanagawa-u.ac.jp
[d)] mikawa@cais.niigata-u.ac.jp

表 1 ビットマスクルール

Filter $\mathcal{R}$	
r_1	$0*1*$
r_2	0000
r_3	$*00*$
r_4	$*1*0$
r_5	$1*1*$
r_6	$**1*$

表 2 単一の連からなるルール

Filter $\mathcal{R}$	
r_1	$01**$
r_2	0000
r_3	$*00*$
r_4	$**10$
r_5	$11**$
r_6	$*1**$

とを確かめられないので，ルールリストに一旦追加されたルールは削除されない．これよりフィルタリングルールリストのルール数は増える一方なので，探索時間もそれに応じて増大していく．よって今後増大するセキュリティルールに対応するためには，分類時間がルール数に依存しないフィルタリングアルゴリズムが不可欠である．

このような状況に対して我々は，任意のビットマスクルールに適したデータ構造 RBT とこれを用いた探索法を提案し，分類時間がルール数に依存しない手法を提案した [1]．この手法 [1] に対しては，RBT から構成される決定木の領域計算量が $O(n^w)$ と膨大になるという問題点 [1] と，RBT の探索には入力系列を余分に参照する問題点があった．そこで我々は，RBT から構成される決定木の領域計算量を $O(n^w)$ から $O(3^w)$ へと減少させる決定木の枝刈り法 [5] と，ポインタと連を RBT に付与することにより RBT 探索の探索時間計算量を $O(w^2+nw)$ から $O(nw)$ へと減少させる手法を提案した [6].

けれども，枝刈り法を適用した決定木の空間計算量は $O(3^w)$ と指数オーダで現実的ではなく，ポインタを付与した RBT を用いた探索法の時間計算量は $O(nw)$ とルール数 n に依存している．そこで，与えられたルールリストの各ルールが単一の連からなるルールならば，探索時間計算量が $O(w)$ とルール数 n に依存しない探索法を提案した [7]．この手法を単一の連からなる RBT 探索とよぶ．

本稿では初めに，単一の連からなる RBT の改善手法を提案する．次に，単一の連からなる RBT 探索を複数の連からなるルールリストにも適用できるように，ルールリスト $\mathcal{R}$ を k 個の単一の連からなるルールリスト $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_k$ へと分割する手法を示す．そして，$\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_k$ に対して k 個の単一の連からなる RBT を含むリストを構築し，このリストによるパケット分類法を提案する．また，計算機実験により提案手法の有効性を確かめる．

2. Run-Based Trie

任意の位置にビットマスク'$*$' を含むビットマスクルール $r \in \{0,1,*\}^w$ とパケットのビット列 $p \in \{0,1\}^w$ との照合を行う際には，ルールのビットマスク $*$ 部分は省略できるので，ビットマスクルール r の $*$ 以外の $0,1$ の部分が重要である．任意の位置にビットマスクを含むビットマスクルールで構成されたルールリストの例を表 1 に示す．我々の手法は，連とよばれるルール中で $0,1$ が連続する部分に注目する．

定義 2.1 $r \in \{0,1,*\}^w$ を長さ w のビットマスクルールとする．下記の二つの条件を満たす r の部分列 $r_i r_{i+1} \ldots r_j$ $(1 \le i \le j \le w)$ を r の連と呼ぶ．

- $r_k = 0 \vee r_k = 1 \quad (i \le k \le j)$
- $(i \ge 2 \Rightarrow r_{i-1} = *) \wedge (j \le w-1 \Rightarrow r_{j+1} = *)$

例えば，長さ 16 のビットマスクルール

$$*1\,0\,1**0\,0\,1*1**0\,1\,0$$

は，2 番目，7 番目，11 番目，14 番目から始まる四つの連 101, 001, 1, 010 を含む．ルール r_i の k 個の連を $r_i^1, r_i^2, \ldots, r_i^k$ $(1 \le k \le \lceil \frac{w}{2} \rceil)$ と表す．我々の探索法の基本的な考えは，パケットのビット列がルールリスト中のどの連と合致するかを調べ，すべての連が合致するルールを調べ，合致するルールの中で一番優先度の高いルールを返す，というものである．パケットのビット列がどの連と合致するかを調べるために，ルールリスト中の連の開始位置毎に w 個のトライを構成する．

長さ w のルールを n 個含むルールリスト $\mathcal{R} = [r_1, r_2, \ldots, r_n]$ から構成した w 個のトライを $T_1, T_2, \ldots, T_w$ とする．トライ T_k はルールリストに含まれる連の中で k ビット目から始まる連で構成される．この w 個のトライ $T_1, T_2, \ldots, T_w$ の集まりを Run-Based Trie (RBT) という．表 1 から構成した RBT を図 1 に示す．図 1 の破線は 0 枝を実線は 1 枝を表す．

RBT は長さ w，ルール数 n のルールリストに含まれる連から構成され，かつ RBT においてそれぞれの連は一カ所にのみ出現するので，RBT の領域計算量は $O(nw)$ である

2.1 Simple Search

我々が Simple Search と呼んでいる RBT の探索法は，T_1 から T_w の順にトライ T_i を入力パケットのビット列で辿り，入力パケットに合致する連を集めて合致するルールを探し，合致したルールの中で最優先のルールを返す，という探索法である．探索に先立って各ルールの連の添字を格納する配列 $A[n]$ と最優先ルールを格納する変数 B を用意し，それぞれ $A[i] := 0$ $(1 \le i \le n)$，$B = n+1$ と初期化する．そして，各トライ $T_i (1 \le i \le w)$ を入力パケット p の i 番目からのビット列 $p[i]p[i+1]\ldots p[w]$ で辿る．各トライを辿るとき，トライの節点に連の印 r_i^j が付いていたら，連の添字 j の値と配列 $A[i]$ の値とを比較して，$j = A[i-1]+1$ ならば，$A[i] := j$ とする．さらに，連の印に下線が引いてあり $i < B$ ならば，$B := i$ とする．全てのトライを辿り終えたときの B の値が，入力パケットに対する最優先のルールとなる．

Simple Search の探索時間計算量を考える．各トライ

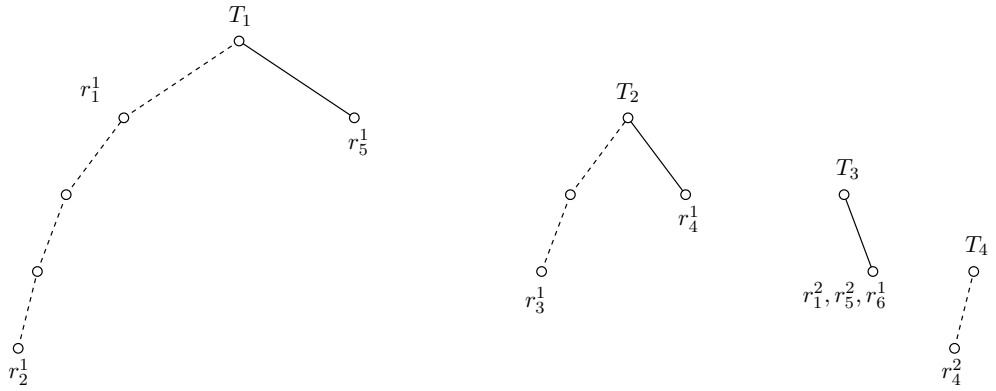

図 1 表 1 のルールリストから構成される Run-Based Trie

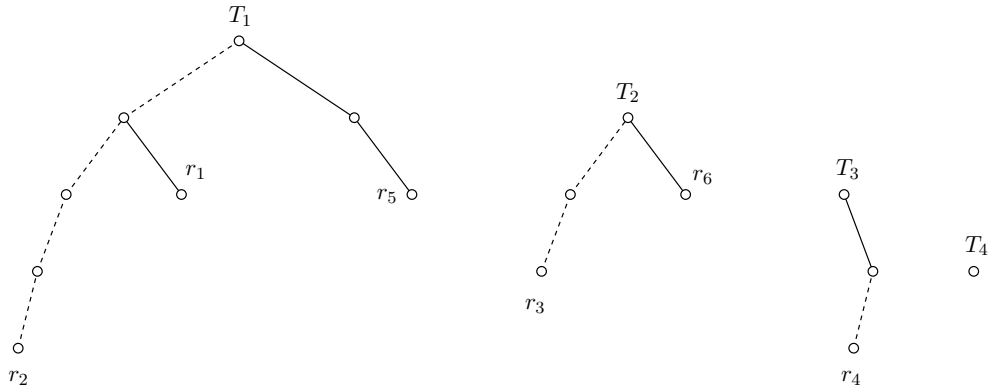

図 2 表 2 のルールリストから構成される RBT

$T_1, T_2, \ldots, T_w$ を探索するのに $w+(w-1)+\cdots+1$ ステップ必要なので，各トライ T_i の探索時間は $O(w^2)$ である．そして，ビット長 w のルールを n 個含むルールリストの連の総数は高々 nw なので，合致した連と配列との比較に必要な時間計算量は $O(nw)$ である．また，合致したルールと最優先ルールとの比較回数は高々ルール数分の n 回なので，最優先ルールを見つけるのに必要な時間計算量は $O(n)$ である．よって，Simple Search の時間計算量は $O(w^2+nw)$ となる．

3. 単一の連からなる RBT

Simple Search の時間計算量が $O(nw+w^2)$ とルール数 n に依存してしまうのは，RBT の節点に連が高々 $n\lceil\frac{w}{2}\rceil$ 個，散らばって存在し，節点に連が存在する度に照合を行わなければならないからである．

本章では，ルールリスト中の各ルールが表 2 のように一つの連だけからなる場合を考える．これより，Simple Search でトライを探索していて連 r_i^j に合致したときに，ルール r_i に合致しているかを判定する必要がない．各ルールの連の数が一つだと連 r_i^j の j は 1 となり冗長なので，以降はルール r_i の連から j を削除して r_i と表す．また，各ルールの連の数が一つだけであると，RBT の各節点は高々一つだけの連を持つ．二つの連 r_i, r_j $(i<j)$ が同一の節点にある場合は，連 r_j が冗長なので削除できる．表 2 のルールリストから構成される RBT を図 2 に示す．

Simple Search には入力パケット p の系列を何度も参照するという問題点がある．Simple Search は，トライ T_k を深さ d の節点まで探索しても，T_k を探索した後はトライ T_{k+1} の根から探索を行う．けれども，トライ T_k を深さ d まで探索したということは，$p[1]$ から $p[k+d-1]$ まで参照していることを意味するので，トライ T_{k+1} での深さ $d-1$ までの探索は冗長である．この点に注目して，入力パケット p の系列を一度だけ参照する探索法を提案した [6]．図 2 の RBT へ 0,1 枝を新たに追加し，T_2 の r_3 を T_1 へ加えた RBT を図 3 に示す．この RBT の探索法は，Simple Search と同様である．Simple Search が T_k の探索を終えたあとに T_{k+1} の根から探索を行うのに対して，0,1 枝を追加した RBT の探索法は，T_1 の根節点から 0,1 枝を辿れなくなるまで高々長さ w だけ探索する．

0,1 枝を追加した図 3 の RBT において，T_1 の r_1 に合致した場合は，以降の探索において r_1 よりも優先度の高い連に合致する可能性がない．よって，r_1 がある節点に対する 0,1 枝を削除して，この節点を終端節点とする．さらに，T_4 に r_{n+1} の連を持つ終端節点を追加して，任意の非終端

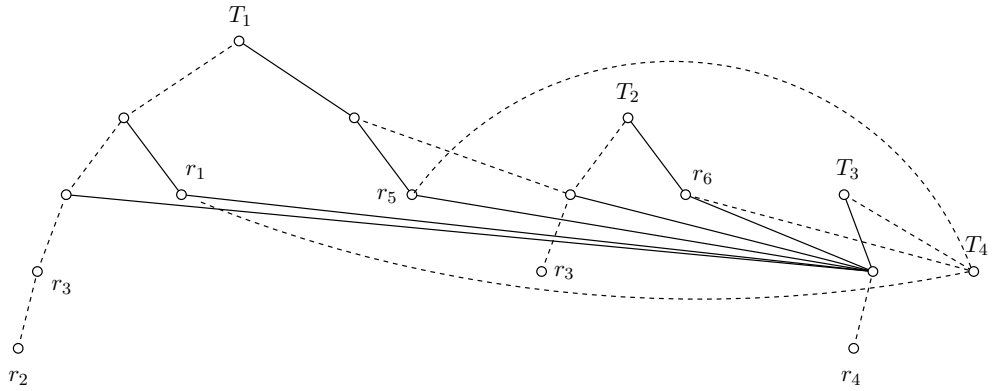

図 3 図 2 の RBT 全体に枝と T_1 に r_3 を追加

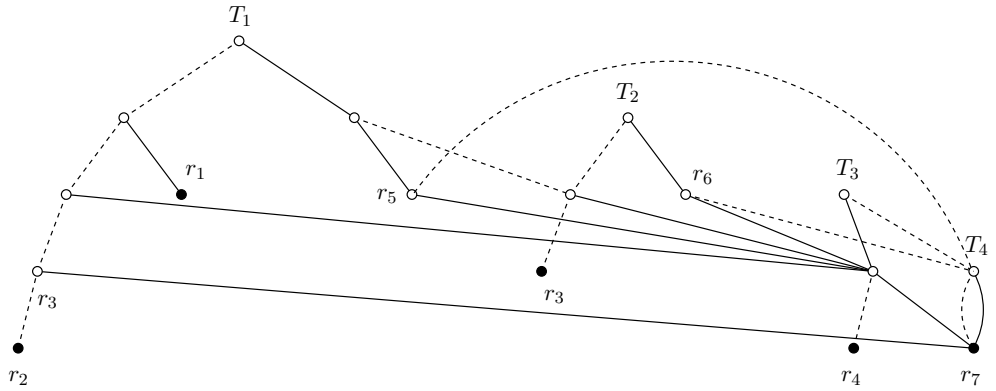

図 4 図 3 の T_4 に r_7 と終端節点を追加，r_1 を持つ節点から 0,1 枝を削除

節点が 0,1 枝を持つようにした単一の連からなる RBT を提案した [7]．図 3 から不要な枝を削除し，T_4 に r_7 を追加した RBT を図 4 に示す．ただし，黒丸は終端節点を表す．

単一の連からなる RBT によるパケット分類は，入力パケットの系列に従って T_1 の根から終端節点まで高々長さ w だけ探索を行う．単一の連からなる RBT の各節点は高々一つだけ連を持つので，単一の連からなる RBT の探索時間計算量は $O(w)$ となり，ルール数 n に依存しない．また，構成される節点の数は高々 nw で，連の数も高々 n なので，単一の連からなる RBT の領域計算量は $O(nw)$ となる．

3.1 単一の連からなる RBT の不要な節点と枝の削除

単一の連からなる RBT では，節点 v が連 r_i を持ち，v_i の 0,1 枝の両方から到達可能な節点に r_i よりも優先度の高いルールが存在しないときに v を非終端節点とする [7]．けれども，図 4 の T_1 の r_5 を持つ節点 v_5 の 0 枝から到達可能な節点に r_5 よりも優先度の高いルールは存在しないので，v_5 の 0 枝は不要である．これより，節点 v が連 r_i を持ち，v の b 枝から到達可能な節点に r_i よりも優先度の高いルールが存在しない場合は，b 枝を削除するように枝の削除規則を変更する．これに伴い，T_w に r_{n+1} を追加することを止め，非終端節点と終端節点の区別も止める．そして，単一の連からなる RBT の探索法を，終端節点まで探索するのではなく，T_1 の根から 0,1 枝を辿れなくなるまで探索するように変更する．図 4 の単一の連からなる RBT から不要な節点と枝を削除した単一の連からなる RBT を図 5 に示す．

4. 単一の連からなる RBT のリスト

単一の連からなる RBT は，与えられたルールリストの各ルールが一つの連だけからなる場合にのみ適用できる．けれども，与えられるルールリストは一般に単一の連からなるルールリストではない．本章では，単一の連からなるルールリストではないルールリスト $\mathcal{R}$ を単一の連からなるルールリスト $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_k$ へと分割し，各ルールリスト $\mathcal{R}_i$ に対する単一の連からなる RBT $\mathcal{F}_i$ を含むリスト $\mathcal{L} = [\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_k]$ を生成し，リスト $\mathcal{L}$ を探索してパケット分類を行う手法を提案する．

4.1 Consecutive Ones Property

ルールリスト $\mathcal{R}$ の列を並べ替えることにより，単一の連からなるルールリスト $\mathcal{R}'$ へ変換できるかという問題を考える．この問題は，ルールの 0,1 を 1，$*$ を 0 へと変換することにより，ブール行列が Consecutive Ones Property

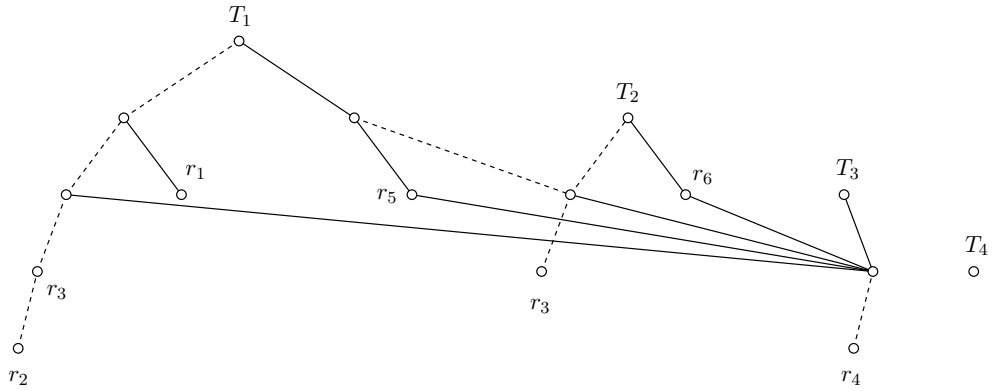

図 5 図 4 の単一の連からなる RBT の不要な節点と枝を削除

(C1P)[8], [9] を満たすかという問題へと帰着できる．

ブール行列 M が C1P を満たすとは，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ことをいう．表 3 の行列は，表 4 に示すように c_1, c_3, c_2, c_4 の順に列を並べれば，各行の 1 が連続するので，C1P を満たす．これに対して，表 5 の行列はどのように列を並べても，各行の 1 を連続させられないので C1P を満たさない．

表 1 のルールリストに対して，0,1 を 1 に，$*$ を 0 に変換して生成されるブール行列は表 3 のブール行列となる．

4.2 区間グラフ

文献 [9] の定理 2 より，ブール行列 M が C1P を満たすことと $G(\tilde{M})$ が区間グラフであることが同値である．ただし，$\tilde{M} = \begin{pmatrix} M \\ I \end{pmatrix}$ であり，$G(M)$ は，M の各行 r_i を節点とし，M_{ik} と M_{jk} が共に 1 となるような k が存在するとき，r_i と r_j が隣接するようなグラフである．表 6 の行列 M に対するグラフ $G(M)$ を図 6 に示す．表 6 の行列 M において，r_1 と r_8 は 1 列目が共に 1 なので，グラフ $G(M)$ で r_1 と r_8 は隣接する．

定理 4.1 ([9], Theorem 2) 任意のブール行列 M に対して以下は同値である：

(1) $G(M)$ は区間グラフであり，M は $G(M)$ の極大クリーク行列である．

(2) M の列は全て極大であり，M は *C1P* を満たす．

グラフ G の極大クリーク行列 M とは，行列 M の行 r_i が G の節点であり，列 c_j において 1 となる行の集合が G の極大クリークとなり，$M_{ij} = 1$ ならば r_i が極大クリーク c_j に含まれる行列のことである．行列 M の列 c_i が極大であるとは，$c_i \subset c_j$ となるような列 c_j が存在しない列のことをいう．ただし，列 $c_j = \{i \mid M_{ij} = 1\}$ とする．例えば，表 3 の行列において，列 c_1, c_4 は極大ではなく，c_2, c_3 は極大である．

定義 4.1 グラフ $G = (V, E)$ に対して，任意の節点 $x, y \in V$ について，$I(x) \cap I(y) \iff xy \in E$ となるような区間の割当 I が存在するとき，G は区間グラフであるという．

図 6 のグラフは，図 7 のような節点に対する区間の割当が存在するので区間グラフである．図 7 は，r_1 に区間 $[0, 3]$ を，r_3 に区間 $[3, 6]$ を割り当てることを意味する．

定義 4.2 S をサイズ n の集合とし，S の順序を $[n]$ から S への全単射とする．ただし，$[n] = \{1, 2, ..., n\}$ とする．$G = (V, E)$ を n 個の節点からなるグラフとし，σ を V の順序とする．G の順序 σ について，任意の $i, k \in [n]$ に対して，$\sigma(i)\sigma(k) \in E$ ならば任意の $j \in [i+1, k-1]$ に対して $\sigma(i)\sigma(j) \in E$ が成り立つとき，σ は *I-ordering* であるという．ただし，整数 i, j に対して $[i, j] = \{k \mid i \leq k \leq j, k \in \mathbb{Z}\}$ とする．

定理 4.2 ([10]) グラフ G が区間グラフであるとき，またそのときに限り G は *I-ordering* を持つ．

図 6 のグラフは区間グラフであり，

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

はこのグラフの I-ordering である．

定義 4.3 グラフ $G = (V, E)$ の極大クリーク $C_1, C_2, \ldots, C_k$ の列 S について，任意の $v \in V$ に対して v を含むクリークが S において連続しているとき，S をクリークチェインという．

図 6 のグラフにおいて，極大クリークは

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

の四つであり，C_1, C_3, C_2, C_4 の列がクリークチェインとなる．

以上より，与えられたルールリスト $\mathcal{R}$ を単一の連からなるルールリスト $\mathcal{R}'$ へと変換できるかは，$G(M_{\mathcal{R}})$ が I-ordering を持つかを判定すれば良い．ただし，$M_{\mathcal{R}}$ は，$\mathcal{R}$

表 3 C1P を満たす行列

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

表 4 表 3 の列を並び替えた行列

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

表 5 C1P を満たさない行列

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

表 6 表 3 の行列に単位行列 I を追加

	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

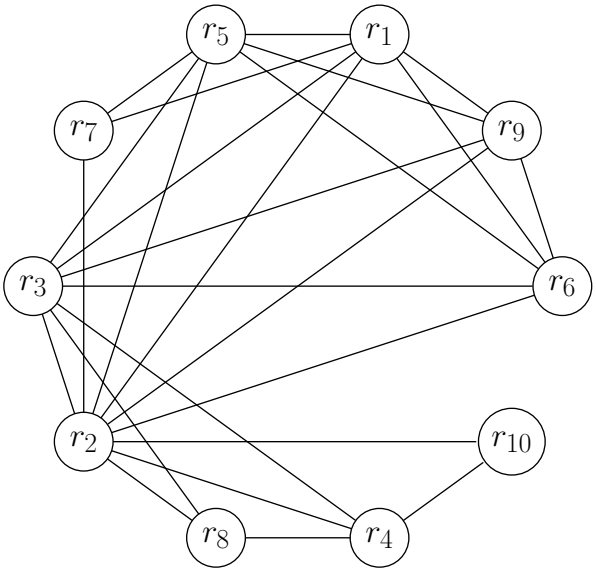

図 6 表 6 の行列 M に対するグラフ $G(M)$

の 0,1 を 1，$*$ を 0 へと変換することにより生成した行列に，単位行列 I_w を追加した行列である．提案手法では，グラフ G が I-ordering を持つかの判定に，文献 [11] のアルゴリズムを用いた．文献 [11] のアルゴリズムは，与えられたグラフ G が区間グラフであれば，G の I-ordering を返す．

定理 4.1 より，行列 M の列 c_i と $G(M)$ の極大クリークは C_i が対応しており，$G(M)$ のクリークチェインの順に M の列を並べれば，M の各行の 1 が連続する．これより，M の各行の 1 が連続する列の順序を求めるためには，グラフの I-ordering σ からクリークチェインを求めればよい．I-ordering σ における極大クリークの順序関係を

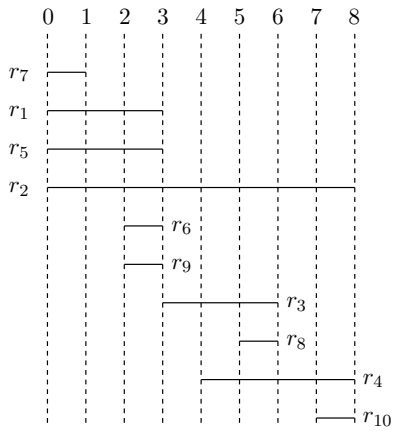

図 7 図 6 の節点に対する割当

$$C_i < C_j \equiv C_i = \emptyset \vee \\ C_i \subset C_j \vee \\ \sigma\big(\min\{\sigma^{-1}(x) \mid x \in C_i \triangle C_j\}\big) \in C_i \quad (1)$$

のように定める．ただし，$X \triangle Y = (X \setminus Y) \cup (Y \setminus X)$ である．たとえば，$C_1 = \{1,2,5,7\}, C_3 = \{1,2,3,5,6,9\}$, $\sigma^{-1} = (2\ 4\ 7\ 9\ 3\ 5\ 1\ 8\ 6\ 10)$ に対して

$$\begin{aligned}\min\{\sigma^{-1}(x) \mid x \in C_1 \triangle C_3\} &= \min\{\sigma^{-1}(x) \mid x \in \{3,6,7,9\}\} \\ &= \min\{7,5,1,6\} \\ &= 1\end{aligned}$$

で，$\sigma(1) = 7 \in C_1$ より，$C_1 < C_3$ となる．順序 1 に従って，極大クリーク $C_1, C_2, \ldots, C_k$ をソートするとクリークチェインが得られる．

4.3 ルールリスト分割

ルールリスト $\mathcal{R}$ を単一の連からなるルールリストのリスト $L_{\mathcal{R}} = [\mathcal{R}_1, \mathcal{R}_2, \ldots \mathcal{R}_k]$ へと分割するアルゴリズムを Algorithm1 に示す．Algorithm1 において，$\mathcal{R}$ が C1P を満たすとは，ルールリスト $\mathcal{R}$ から生成したブール行列 $M_{\mathcal{R}}$ が C1P を満たすことをいう．ルール数が 3 未満のルールリストは必ず C1P を満たすので，ルールリストのサイズ $|\mathcal{R}|$ が 3 未満ならば，$\mathcal{R}$ のみを含むルールリストのリスト $L_{\mathcal{R}}$ を返す．ルールリスト $\mathcal{R}$ のルール数が 3 以上の場合は，5 から 8 行目で $\mathcal{R}$ の先頭と先頭の次のルールから成るルールリスト S を $L_{\mathcal{R}}$ に挿入する．11 から 17 行目は，C1P を

Algorithm 1: Rulelist Partiion

```
   input  : Rulelist R
   output: List of Rulelist consisting of Single Run L_R
 1 make an empty list of rulelist L_R ;
 2 if |R| < 3 then
 3 |   add R to L_R ;
 4 |   return L_R ;
   end
 5 make a new rulelist S;
 6 add head(R) to S and remove head(R) ;
 7 add head(R) to S and remove head(R) ;
 8 add S to L_R ;
 9 while R ≠ ∅ do
   |   r ← head(R) ;
   |   it ← L_R.begin ;
10 |   flag = false ;
11 |   while it ≠ L_R.end do
12 |   |   if *it ∪ {r} holds C1P then
13 |   |   |   add r to *it ;
14 |   |   |   flag = true ;
15 |   |   |   break ;
   |   |   end
16 |   |   it ← it.next ;
   |   end
17 |   if flag ≠ true then
18 |   |   make a new rulelist S;
19 |   |   add r to S ;
20 |   |   add S to L_R ;
   |   end
21 |   remove r from R ;
   end
22 return L_R ;
```

満たすルールリスト $\mathcal{R}_1, \mathcal{R}_2, \ldots \mathcal{R}_k$ の先頭から，ルール r を加えても C1P を満たすルールリスト $\mathcal{R}_i$ を探し，そのようなルールリストがあれば，そのルールリストに r を加える．そのようなルールリストがなければ，r のみから成るルールリスト S を新たに生成し，S を $L_{\mathcal{R}}$ に加える．

4.4 単一の連からなる RBT のリストによる探索

前節の Algorithm1 で生成されるルールリストのリスト $L_{\mathcal{R}} = [\mathcal{R}_1, \mathcal{R}_2, \ldots \mathcal{R}_k]$ の各 $\mathcal{R}_i$ に対して，単一の連からなる RBT $\mathcal{F}_i$ を構築する．この単一の連からなる RBT のリスト $L_{\mathcal{F}} = [\mathcal{F}_1, \mathcal{F}_2, \ldots \mathcal{F}_k]$ を用いたパケット分類アルゴリズムを Algorithm2 に示す．4 行目の $\mathcal{F}_i(p)$ は，パケット p で単一の連からなる RBT $\mathcal{F}_i$ を探索して得られるルール番号を返す．

5. 計算機実験

提案手法の有効性を確かめるために C 言語を用いて計算機実験を行った．実験環境は，主記憶容量が 24GB，CPU が Intel Core i7-980X 3.33 GHz の上 CentOS Release 6.2 である．ルール数が $100, 200, 300, 400, 500$ のルールリストとヘッダ数が十万のヘッダリストを Class Bench[12] を用いて生成した．

線型探索 (Linear Search) と提案手法による分類時間を図 8 に示す．ただし，単位は秒である．なお，提案手法のルールリスト分割によって生成されたルールリストの数は，ルール数 $100, 200, 300, 400, 500$ に対してそれぞれ $5, 5, 6, 6, 6$ である．

図 8 とルールリスト分割により生成されたルールリストの数より，少ない数のルールリストへとルールリストを分割できれば提案手法は有効である．

Algorithm 2: List of Single-RBT Search

```
   input  : List of RBT consisting of Single Run L_F,
            packet p
   output: highest priority rule number
 1 cand ← n + 1   // n is the number of rules ;
 2 i ← 0 ;
 3 while i < |L_F| do
 4 |   c ← F_i(p) ;
 5 |   if c < cand then cand ← c;
   end
 6 return cand ;
```

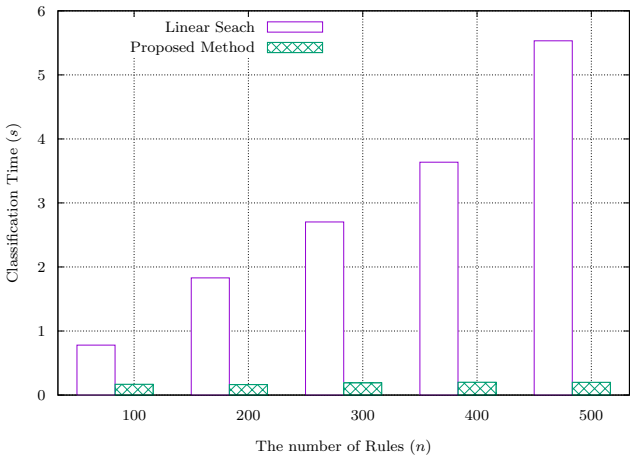

図 8 分類時間（秒）

6. まとめ

本稿では，単一の連からなる RBT を改善し，単一の連からなる RBT のリストによるパケット分類法を提案した．提案手法は，ルールリストを単一の連からなるルールリストのリスト $L_{\mathcal{R}} = [\mathcal{R}_1, \mathcal{R}_2, \ldots \mathcal{R}_k]$ へと分割し，各 $\mathcal{R}_i$ に対して，単一の連からなる RBT $\mathcal{F}_i$ を構築する．分割数 k が少ないときに提案手法の探索時間計算量がルール数 n に依存しないことを計算機実験で確かめた．

今後の主な課題は，提案手法を線型探索だけでなく他のパケット分類手法と比較すること，Algorithm1 のような逐次探索よりも分割数を小さくするルールリスト分割アルゴリズムを考案すること，単一の連からなる RBT のリストを Algorithm2 のように素朴に探索するよりも効率よく探索する手法を考案すること，の三つである．

参考文献

[1] MIKAWA, K. and TANAKA, K.: Run-Based Trie Involving the Structure of Arbitrary Bitmask Rules, *IEICE Transactions on Information and Systems*, Vol. E98.D, No. 6, pp. 1206–1212 (online), DOI: 10.1587/transinf.2013EDP7087 (2015).

[2] Taylor, D. E.: Survey and Taxonomy of Packet Classification Techniques, *ACM Comput. Surv.*, Vol. 37, No. 3, pp. 238–275 (2005).

[3] Li, W. and Li, X.: HybridCuts: A Scheme Combining Decomposition and Cutting for Packet Classification, *2013 IEEE 21st Annual Symposium on High-Performance Interconnects*, pp. 41–48 (online), DOI: 10.1109/HOTI.2013.12 (2013).

[4] Inoue, T., Mano, T., Mizutani, K., Minato, S. and Akashi, O.: Rethinking Packet Classification for Global Network View of Software-Defined Networking, *2014 IEEE 22nd International Conference on Network Protocols*, pp. 296–307 (online), DOI: 10.1109/ICNP.2014.52 (2014).

[5] 原田崇司，　田中賢，三河賢治：Run-Based Trie から構成される決定木の枝刈り法 (技術と社会・倫理)，電子情報通信学会技術研究報告 = IEICE technical report : 信学技報，Vol. 115, No. 294, pp. 11–17 (2015).

[6] 原田崇司，　田中賢，三河賢治：ポインタ付与による Run-Based Trie 探索の高速化 (回路とシステム)，電子情報通信学会技術研究報告 = IEICE technical report : 信学技報，Vol. 116, No. 315, pp. 13–18 (2016).

[7] 原田崇司，　田中賢，三河賢治：単一の連からなる Run-Based Trie によるルール探索の高速化，研究報告アルゴリズム (AL)，Vol. 2017, No. 2, pp. 1–7 (2017).

[8] Booth, K. S. and Lueker, G. S.: Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms, *Journal of Computer and System Sciences*, Vol. 13, No. 3, pp. 335 – 379 (online), DOI: https://doi.org/10.1016/S0022-0000(76)80045-1 (1976).

[9] Habib, M., McConnell, R., Paul, C. and Viennot, L.: Lex-BFS and Partition Refinement, with Applications to Transitive Orientation, Interval Graph Recognition and Consecutive Ones Testing, *Theor. Comput. Sci.*, Vol. 234, No. 1-2, pp. 59–84 (online), DOI: 10.1016/S0304-3975(97)00241-7 (2000).

[10] Ramalingam, G. and Rangan, C.: A unified approach to domination problems on interval graphs, *Information Processing Letters*, Vol. 27, No. 5, pp. 271 – 274 (online), DOI: https://doi.org/10.1016/0020-0190(88)90091-9 (1988).

[11] Li, P. and Wu, Y.: A four-sweep LBFS recognition algorithm for interval graphs, *Discrete Mathematics & Theoretical Computer Science*, Vol. Vol. 16 no. 3 (online), available from ⟨http://dmtcs.episciences.org/2094⟩ (2014).

[12] Taylor, D. E. and Turner, J. S.: ClassBench: A Packet Classification Benchmark, *IEEE/ACM Trans. Netw.*, Vol. 15, No. 3, pp. 499–511 (2007).