Science Journal of Kanagawa University **31** ：53-60 (2020)

■原　著■　2019 年度神奈川大学総合理学研究所共同研究助成論文

連分割トライに基づく決定木の枝刈り法

原田崇司[1, 2]　田中　賢[1, 4]　三河賢治[3]

A Pruning Method for the Decision Tree Based on Run-Based Trie

Takashi Harada[1, 2], Ken Tanaka[1, 4] and Kenji Mikawa[3]

[1] Department of Information Sciences, Faculty of Science, Kanagawa University, Hiratsuka City, Kanagawa 259-1293, Japan
[2] Kochi University of Technology, 185 Miyanokuchi, Tosayamada, Kami, Kochi 782-8502, Japan
[3] Center for Academic Information Service, Niigata University, Nishi-ku, Niigata City, Niigata 950–2181, Japan
[4] To whom correspondence should be addressed. E-mail: ktanaka@info.kanagawa-u.ac.jp

Abstract: Numerous methods for packet classification have been developed. We proposed a decision tree method based on Run-Based Trie. In this paper, we propose a pruning method for the decision tree. Space complexity of the pruning method is still $O(l3^l)$ where l is the length of a rule. Applying the proposed pruning method, we can construct a decision tree where l = 64, which can not be constructed with the conventional method. This result shows that, although our decision tree method based on Run-Based Trie is not feasible in theory, it can be used in a practical sense.

Keywords: packet classification, arbitray bitmask, trie, decision tree

序論

インターネット上の DDoS 攻撃、コンピュータウィルスの蔓延、情報流出などのサイバー脅威の数は年々増加している。パケット分類はこれらの脅威を防ぐ手段の一つである。L3 スイッチなどのネットワーク機器は、到着パケットとフィルタリングルールを照らし合わせることによりパケットの通過の可否を判断する機能をもつ。これらの機器では、フィルタリングポリシをフィルタリングルールのリストで表現し, ルールリストを線型探索することによってパケットフィルタリングを実現する。しかし、この方法では全てのパケットがルールリストの大部分のルールと比較される。このことは、ルール数が多くなると通信の遅延を引き起こす。効率の良いパケットフィルタリングを行うために、従来 (1) ルールリスト最適化アルゴリズムの開発、(2) パケットに合致する一番優先度の高いルールを探索するアルゴリズムの開発という二つの研究が行われている。フィルタリングルールリスト最適化問題に対して我々は、隣接するルール間の従属関係に注目してルールリストを並び替える発見的解法[1)]と、ルールリスト中で最大の重みを持つルールとそのルールが依存するルールに注目してルールリストを並び替える発見的解法[2)]を提案した。Tapdiya らは、ルールリストをルール数分の部分リストに分解して、各部分リストの重みの平均値によってルールリストを並び替える発見的解法を提案した[3)]。

(2)の問題に対しても種々の方法が提案されてきた[4)]。Gupta らは、パケット分類問題を計算幾何学の問題として捉え、ルールの数を閾値以下になるように分類対象となる領域を超平面で再帰的に分割する HiCuts と呼ばれる手法を提案した[5)]。Singh らは, HiCuts が対象領域を超平面で分割するのに対して対象領域を超立方体で分割する HyperCuts と呼ばれる手法を提案した[6)]。Li らは、ルールリストをルールリストの特徴に従って幾つかの部分リストに分割してから、各部分リストが表す対象領域を分割する HybridCuts と呼ばれる手法を提案した[7)]。これらの三つの手法は、最終的には線型探索を行うので分類時間の計算量はルール数 n に対して O(n) である。また、これらの手法は任意の位置にビットマスクを含むルールには適用できない。

Srinivasan らは、ルールのフィールド毎にトライ

©Research Institute for Integrated Science, Kanagawa University

を作成し、それらのトライをつなげた階層型トライ用いたアルゴリズムを提案した[8]。この手法は分類時間がルール長 l に比例する手法で分類時間がルール数 n には依存しないが、やはり任意の位置にビットマスクを含むルールには適用できない。

任意のビットマスクルールに対応できる手法として、Ternary Content Addressable memories (TCAM)[9] があるが、TCAM は，あらかじめルール数が制限され、また特殊なハードと膨大な電力を必要とする。パケットフィルタリングは、携帯電話やタブレット端末などの一般的な機器でも必要とされる処理なので、特殊なハードを必要としないソフトウェアに基づくアルゴリズムの開発が必要である。Ligatti らは、ルールをいくつかのグループに分けて各グループ毎にビットマスクを 0, 1 に展開し、各グループ毎にパケットに合致するルールを調べて、その結果の bitwise-AND をとることによりパケットに合致する最優先ルールを得る、Grouper と呼ばれる手法を提案した[10]。この手法は、最終的に bitwise-AND をとるので分類時間がルール数 n に依存する。

一般に外部の脅威が排除されたことを確かめる方法はないので、ルールリストに一旦追加されたルールは削除されることがない。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であるので、探索時間がフィルタリングルールのルール数に依存しないフィルタリングアルゴリズムが求められる。また、今後 SDN 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ビットマスクルールに対応するアルゴリズムが求められる。

ビットマスクルールに対応してかつ分類時間がルール数に依存しないアルゴリズムが存在しない中で、我々は連分割トライ (Run-Based Trie, RBT) というデータ構造を用いたビットマスクルールに対応して分類時間がルール数に依存しないアルゴリズムを提案した[11]。しかし提案した手法は、探索に用いる決定木の領域計算量が $O(n^l)$ と膨大になるという問題がある。

本論文では、文献[11]で提案された探索法に用いられる決定木の枝刈りアルゴリズムを提案する。提案アルゴリズムよって構築される決定木の領域計算量は $O(l3^l)$ となるが、従来の方法では決定木を構築できなかったルールリストに対しても決定木を構築できるようになる。このことから提案手法の領域計算量は $O(l3^l)$ よりも低く抑えられる可能性がある。

方法

本章以降では、フィルタリングルールは任意の位置にビットマスクを含む $\{0, 1, *\}$ から構成される長さ l の文字列とする。表 1 に任意の位置にビットマスクを含むルールの例を示す。

表 1. 任意の位置にビットマスクを含むルールのリスト

Filter	F_1	Filter	F_1
r_1	$*\,0\,*\,1$	r_7	$*\,*\,1\,0$
r_2	$0\,0\,0\,0$	r_8	$0\,1\,*\,*$
r_3	$0\,*\,0\,0$	r_9	$*\,1\,1\,*$
r_4	$0\,*\,1\,*$	r_{10}	$*\,0\,0\,0$
r_5	$1\,1\,0\,0$	r_{11}	$*\,1\,*\,1$
r_6	$*\,0\,1\,*$	r_{12}	$*\,*\,*\,1$

連分割トライ (Run-Based Trie, RBT)

ビットマスクルールとパケットのビット列との比較を行う際には、ルールのビットマスク$*$部分は省略できるので、$*$以外の 0, 1 の部分が重要である。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. $r \in \{0, 1, *\}^l$ を長さ l 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 $b_i b_{i+1} \ldots b_j$ $(1 \le i \le j \le l)$ を r の連と呼ぶ。

- $b_k = 0 \vee b_k = 1$ $(i \le k \le j)$
- $(i \ge 2 \Rightarrow b_{i-1} = *) \wedge (j \le l - 1 \Rightarrow b_{j+1} = *)$

例えば、長さ 16 のビットマスクルール$*$ 10 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 $\rho_i^1, \rho_i^2, \ldots, \underline{\rho_i^k}$ $(1 \le k \le \lceil \frac{l}{2} \rceil)$ と表す。なお、ルール中の最後の連には下線を引く。

Run-Based Trie の構築

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 l 個のトライを構成する。長さ l のルールを n 個含むルールリスト

$$R = [r_1, r_2, \ldots, r_n]$$

から構成した l 個のトライを $T_1, T_2, \ldots, T_l$ 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 l 個のトライ $T_1, T_2, \ldots, T_l$ の集まりを連分割トライ (Run-BasedTrie, RBT) という。表 1 から構成した Run-Based Trie を図 1 に示す。

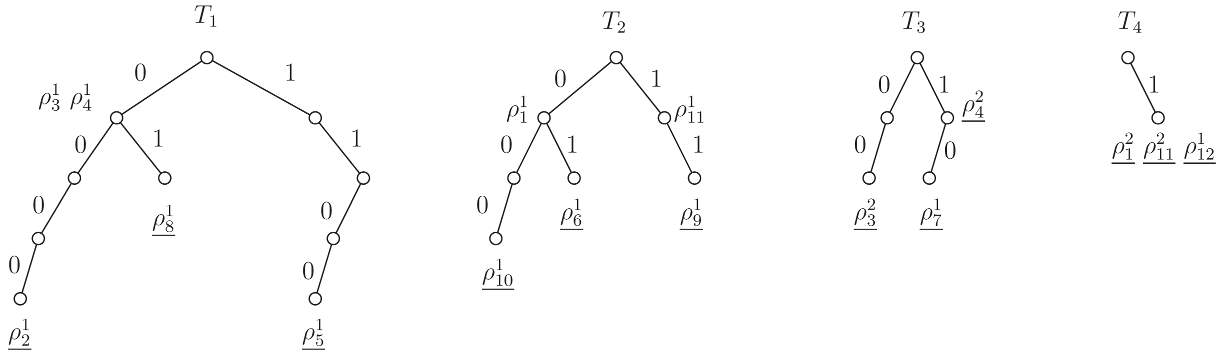

図 1.　Run-Based Trie.

Run-Based Trie は長さ l, 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 Run-Based Trie においてそれぞれの連は一カ所にのみ出現するので、Run-Based Trie の領域計算量は $O(nl)$ である。

Simple Search

我々が Simple Search と呼んでいる Run-Based Trie の直感的な探索法を示す。Simple Search は、トライを T_1 から T_l まで順に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。Simple Search では、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 $A[i] \coloneqq 0\ (1 \leq i \leq n)$、$B = n + 1$ と初期化する。そして、各トライ $T_i (1 \leq i \leq l)$ を入力パケット p の i 番目からのビット列 $p[i]p[i+1]\ldots p[l]$ で辿る。各トライを辿るとき、トライのノードに連の印 ρ_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、$A[i] \coloneqq j$ とする。さらに、連の印に下線が引いてあり $i < B$ ならば、$B \coloneqq i$ 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示す。各トライ $T_1, T_2, \ldots, T_l$ を探索するのに $l+(l-1)+\cdots+1$ ステップ必要なので、各トライ T_i の探索時間は $O(l_2)$ である。そして、ビット長 l のルールを n 個含むルールリストの連の総数は高々 nl 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 $O(nl)$ 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 $O(l^2 + nl)$ となる。

決定木

本章では Run-Based Trie 用いた決定木による探索法を示す。この探索法は Simple Search に基づいているが、Simple Search の探索時間がルール数 n に依存してしまう原因を取り除くことを主眼として提案された探索法である。

Simple Search の探索時間がルール数 n に依存してしまうのは、各トライ T_i を辿る中で、連の照合を順に行っているからである。一方、Simple Search におけるパケットの $T_1, T_2, \ldots, T_l$ の探索パタンは有限である。よって、パケットの探索パタンを列挙して、それぞれのパタンにおける最優先ルールを前もって求めることができる。この点に注目して、パケットで Run-Based Trie を辿りパケットがどのパタンに対応するかを調べるだけでパケットに合致する最優先ルールを返す決定木を構築できる。

合致連集合

前記の考えに基づいて Run-Based Trie を探索した際に合致する連の組み合わせの集合 S_i を各トライ毎に前もって作る。そして、各トライ T_i での S_i の直積 $S_1 \times S_2 \times \cdots \times S_l$ をとり、すべての l 項組に対してそれぞれの l 項組が保持する連を照合して対応するルール番号をつけ加える。つまり、$|S_1| \times |S_2| \times \cdots \times |S_l|$ 個の $l+1$ 項組を構成する。

この合致連集合をもとに決定木を次のように構築する。はじめに、Simple Search での各トライ T_i を探索した際のパケットと連との合致の組み合わせ $S_1, S_2, \ldots, S_l$ を構成する。

$$
\begin{aligned}
S_1 &= \{\ S_1^1,\ S_1^2, \ldots, S_1^{m_1},\ \phi\ \} \\
S_2 &= \{\ S_2^1,\ S_2^2, \ldots, S_2^{m_2},\ \phi\ \} \\
&\ \vdots \\
S_l &= \{\ S_l^1,\ S_l^2, \ldots, S_l^{m_l},\ \phi\ \}
\end{aligned}
$$

但し、各 S_i において ϕ は入力パケットが T_i のトライでいずれの連とも合致しないパタンを表す。この集合のことを合致連集合と呼ぶ。例えば，図 1 の Run-Based Trie から構成される合致連集合は(1)のようになる。

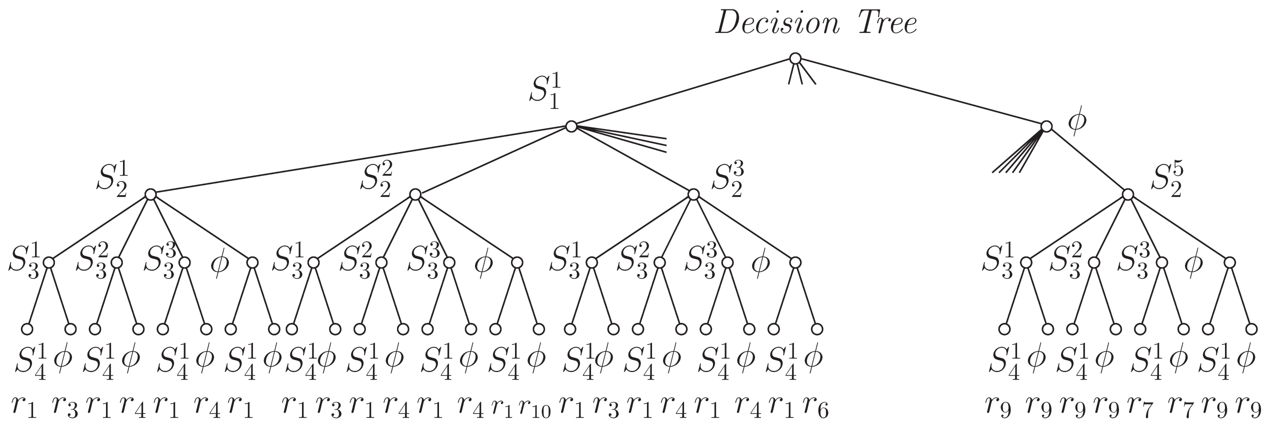

図 2. 決定木 .

$$
\begin{aligned}
S_1 &= \{\ \{\rho_3^1, \rho_4^1\},\ \{\underline{\rho_2^1}, \rho_3^1, \rho_4^1\},\ \{\rho_3^1, \rho_4^1, \underline{\rho_8^1}\},\ \{\underline{\rho_5^1}\},\ \phi\ \}\\
S_2 &= \{\ \{\rho_1^1\},\ \{\rho_1^1, \underline{\rho_{10}^1}\},\ \{\rho_1^1, \underline{\rho_6^1}\},\ \{\rho_{11}^1\},\ \{\underline{\rho_9^1}, \rho_{11}^1\}\ \}\\
S_3 &= \{\ \{\underline{\rho_3^2}\},\ \{\underline{\rho_4^2}\},\ \{\underline{\rho_4^2}, \underline{\rho_7^1}\},\ \phi\ \}\\
S_4 &= \{\ \{\underline{\rho_1^2}, \underline{\rho_{11}^2}, \underline{\rho_{12}^1}\},\ \phi\ \}
\end{aligned} \tag{1}
$$

ここで、図 1 の例では S_2 の元として ϕ が存在しないことを注意しておく。図 1 の T_2 を見ると、高さ 1 のラベルが 0 のノード 0 とラベルが 1 のノード 1 に ρ_1^1 と ρ_{11}^1 という連が存在する。これより、すべてのパケットは T_2 において ρ_1^1 か ρ_{11}^2 のいずれか一方だけに必ず合致する。よって、(1)の S_2 には ϕ を含めない。

決定木は、合致連集合 $S_1, S_2, \ldots, S_l$ の直積 $S_1 \times S_2 \times \cdots \times S_l$ の各要素に最優先ルールのルール番号を加えたものとする。直積の各要素に対応する最優先ルールは、Simple Search を用いて計算する。図 1 から構成した決定木を図 2 に示す。ノード数が $\sum_{i=1}^{l}\prod_{j=1}^{i}|S_i| = 5+25+100+200 = 30$ と大きくなるので、一部を省略している。各パスに対応する最優先ルールを葉の下に記している。このようにして構成された決定木の領域計算量は $O(n^l)$ となる [11]。

決定木探索

前節のようにして構築した決定木を用いた探索法を下記に示す。Run-Based Trie 上のトライ T_i を入力パケットの i 番目からのビット列で辿り、パケットと連との合致のパタンを調べる。そして、合致のパタンと決定木の i 番目の高さのノードを比較して一致している決定木のノードを辿る。これを、決定木の根から葉に至るまで繰り返す。葉に至ったとき、葉に与えられているルールが、入力パケットに対する最優先ルールとなる。

例として、図 1 の Run-Based Trie と図 2 の決定木を用いて、入力パケット 0010 の最優先ルールを調べる。はじめに、T_1 を 1 ビット目からの 0010 で辿ると、連との合致のパタンは、$\{\rho_3^1, \rho_4^1\} = S_1^1$ となるので、決定木を根から S_1^1 のノードへと辿る。次に、T_2 を 2 ビット目からの 010 で辿ると、連との合致のパタンは、$\{\rho_1^1, \underline{\rho_6^1}\} = S_2^3$ となるので、決定木を S_1^1 のノードから S_2^3 のノードへと辿る。次に、T_3 のノードを 3 ビット目の 10 で辿ると、連との合致のパタンは、$\{\underline{\rho_4^2}, \underline{\rho_7^1}\} = S_3^3$ となるので、決定木を S_2^3 のノードから S_3^3 のノードへと辿る。最後に、T_4 のノードを 3 ビット目の 0 で辿ると、連との合致のパタンは ϕ となるので、決定木のノードを S_3^3 から ϕ へと辿る。ここで、$S_1^1 - S_2^3 - S_3^3 - \phi$ のパスの葉には、r_4 が与えられているので、入力パケット 0010 に対応する最優先ルールは r_4 となる。

Algorithm 1: Construct the Decision Tree

input : Match Runs Tries $\bar{S}_1, \bar{S}_2, \ldots, \bar{S}_l$
output: Decision Tree D

1 make the root node of Decision Tree D
2 Algorithm2(1, &$\bar{S}_1$, D)
3 return D

決定木探索の時間計算量のオーダは、Run-Based Trie を入力パケットで探索するのにかかる時間のオーダとなるので、$O(l^2)$ となり n に依存しない。

決定木の枝刈り法

Run-Based Trie から素朴に決定木を構成すると、決定木の領域計算量は $O(n^l)$ となってしまう [11]。本章では、素朴に構成した場合に不要となる辺を含まないような決定木を構成する方法を示す。決定木を枝刈りする方法は大きく分けて三つある。そのうち二つは、決定木の辺を構成する際に，その辺を構成しようとするまでに既に構成した決定木を参照することにより、合致連のパタンとしてあり得ない辺を排

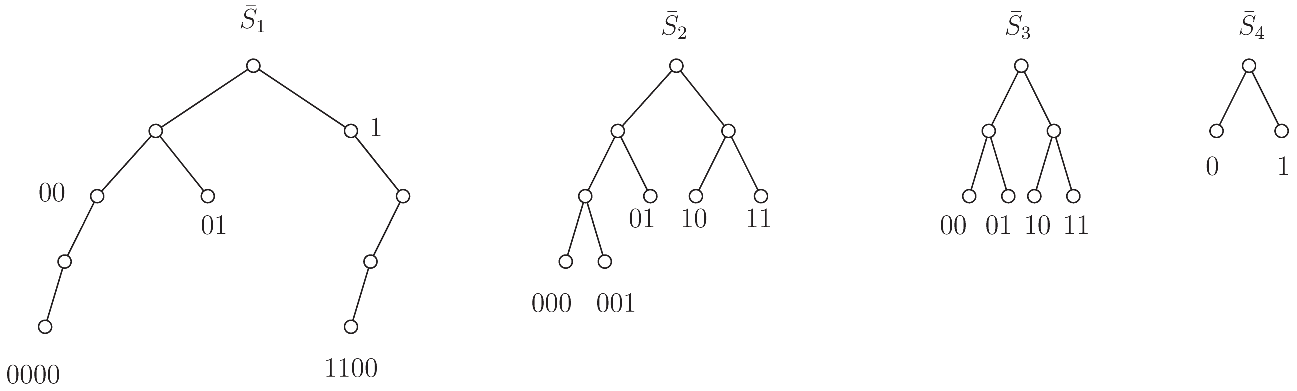

図 3.　合致連集合(3)に対応する二分木 .

Algorithm 2: Traverse $\bar{S}_h$ and make a Node

input : height h
input : A pointer v to a node on $\bar{S}_h$
input : A node of Dtree p

1 **if** $v = NULL$ **then**
　return;
end
2 Algorithm2($h+1$, v.left, p)
3 Algorithm2($h+1$, v.right, p)
4 **if** $*v$ *has a element of* $\bar{S}_h$ **then**
5 　make a node u as the child of v
6 　**if** $h < l$ **then**
7 　　A pointer q points to the root of $\bar{S}_{h+1}$
8 　　modifyTrieS($\bar{S}_{h+1}$, u)
9 　　modifyPointer($\bar{S}_{h+1}$, q, u)
10 　　Algorithm2($h+1$, q, u)
11 　　revert($\bar{S}_{h+1}$)
　end
end

除するというものである。もうひとつは，前記の方法で決定木を構成した後に、決定木の葉のルールを確認して不要な部分が存在したら枝刈りするというものである。

これ以降の枝刈りアルゴリズムの説明のために、SimpleSearch における連とパケットとの合致パタン、合致連集合S_i^jの元を、その元を得るパケットのビットバタンに置き換える。ただし、ϕ は置き換えない。例えば、図 1 から構成される合致連集合(1)を次の(2)ように表す。

$$
\begin{aligned}
S_1' &= \{\ 0,\ 0000,\ 01,\ 1100,\ \phi\ \} \\
S_2' &= \{\ 0,\ 000,\ 01,\ 1,\ 11\ \} \\
S_3' &= \{\ 00,\ 1,\ 10,\ \phi\ \} \\
S_4' &= \{\ 1,\ \phi\ \}
\end{aligned} \tag{2}
$$

さらに、S′₁ には 0 があるので S′₁ の ϕ のパタンに合致するパケットの一ビット目は 1 のはずである。よって ϕ を 1 と表す。同様に S′₁ には 01 があるので、S′₁ の 0 は 00 に対応する。このように、冗長なパタンをあらかじめ排除し、合致するパタンの文字列を更新して(2)から(3)を構成する。(3)を二分木で表現したのが図 3 である。

$$
\begin{aligned}
\bar{S}_1 &= \{\ 00,\ 0000,\ 01,\ 1100,\ 1\} \\
\bar{S}_2 &= \{\ 001,\ 000,\ 01,\ 10,\ 11\ \} \\
\bar{S}_3 &= \{\ 00,\ 11,\ 10,\ 01\ \} \\
\bar{S}_4 &= \{\ 1,\ 0\ \}
\end{aligned} \tag{3}
$$

このようにすることで、決定木の各ノードはラベルとして長さ l の系列 $a_1a_2\cdots a_l$ を持ち、子孫のラベルとしてあり得ないラベルの集合として k 個の系列 $b_{k1}b_{k2}\cdots b_{kl}$, $b_{ki}\in\{0, 1, \epsilon\}$ を持つ. 但し、a_i, $b_{ki}\in\{0, 1, \epsilon\}$ $0\le i\le 2^{l-1}-2$ である。また、系列内で一番右に表れる 0 または 1 までの長さをその系列の長さとする。

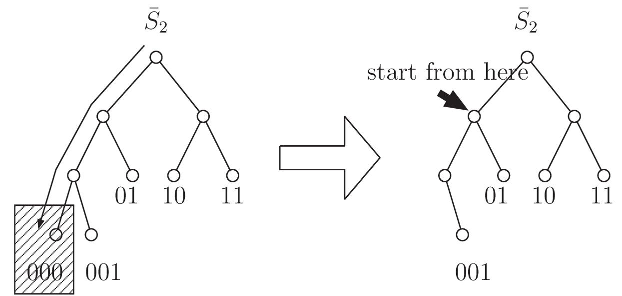

図 4.　modifyTrieS, modifyPointer によってあり得ない候補を削除 .

枝刈りを伴う決定木構築アルゴリズムを Algorithm 1, 2 に示す。アルゴリズムの本体は Algorithm 2 である。合致連集合の要素をノードに持つ各$\bar{S}_h$をそれぞれ後行順で辿り、合致連集合の要素があったら、その要素に対応する決定木のノードを作成し、次に$\bar{S}_{h+1}$の二分木を辿るという再帰的な方法で決定木を構築する。

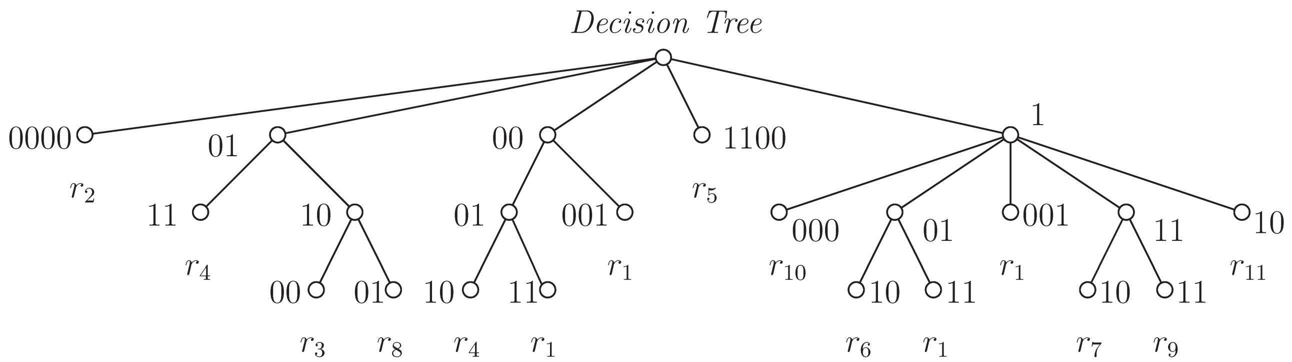

図 5. 枝刈り法を適用して構成した決定木.

Algorithm 2 の詳細について説明する。5 行目で v の子ノード u を作成する時、v のラベルが u のラベルより 2 以上長い場合は、作成される u のノードのラベルを v のラベルで上書きする。8 行目の modifyTrieS は、親ノードが持つ、子ノードとしてあり得ないラベルで下位の$\bar{S}_{h+1}$を辿りあり得ないラベルの子を作るのを防ぐ。図 4 は、決定木の高さ 1 の親ノード 00 から子ノードを作成する状況を示している。図 4 の例では、$\bar{S}_1$に 0000 があるので、決定木の高さ 1 の 00 の子ノードを$\bar{S}_2$から作るときは、先に$\bar{S}_2$を 000 で辿った先の網掛け部分の木を削除する。9 行目の modifyPointer では、u の子ノード作成のために$\bar{S}_{h+1}$を u のラベルで先に$\bar{S}_{h+1}$を辿る。これより、u のラベルと一致しない子ノードが決定木の高さ h+1 に作成されない. 図 4 の例では､00 の 2 ビット目の 0 で$\bar{S}_2$を辿ってから$\bar{S}_2$を探索している。11 行目の revert はトライを編集する前の状態に戻す関数である。Algorithm 1, 2 で構成した決定木の中で、一本道になっているパスや葉が同じ部分木を縮約すると図 5 の決定木が得られる。表 1 の例では決定木のノード数が 330 から 23 となり、約 93% のノードを削減できた。

決定木の領域計算量

本章では、提案手法によって構成される決定木の領域計算量がルール長 l に関して $O(l3^l)$ となることを示す。そのために、まず二つの補題を示す。

補題 2.1. 決定木の各高さ h において､それぞれのノードのラベルは互いに異なる。

Proof. 決定木の高さ h に関する数学的帰納法で示す。

・$h = 1$ で成り立つことを示す。

高さ 0 の根ノードは$\bar{S}$を変更するための要素を持たないので、決定木の高さ 1 の各ノードが持つラベルは$\bar{S}$の要素のラベルのいずれかであり、重複はない。よって、高さ 1 のノードのラベルは互いに異なる。

・$h = k$ で成り立つと仮定して、$h = k + 1$ で成り立つことを示す。

枝刈りアルゴリズムにより、決定木の高さ k + 1 のノードのラベルは高さ k にある親ノードのラベルの長さ分は親ノードのラベルと必ず一致する。

帰納法の仮定より、決定木の高さ k のノードのラベルは全て異なるので、親ノードのラベルの長さ分で見れば，決定木の高さ k + 1 でのラベルは全て異なる。

さらに枝刈りアルゴリズムにより、決定木の高さ k のノードから高さ $k + 1$ へ子ノードを作る際には子孫ノードのラベルとしてあり得ない系列を参照するので、高さ $k + 1$ の各ノードのラベルは親ノードのラベルの長さ分に子ノードのラベルを足した分もそれぞれ異なる。よって､$h = k + 1$ でも各ノードのラベルは異なる。以上より、補題 2.1 は成り立つ。

定理 2.1. 決定木の領域計算量は $O(l3^l)$ である。

Proof. 補題 2.1 より、高さ h にあるノードが持つラベルはそれぞれ異なるので高さ k に現れる系列の数は高々 3^l である。決定木全体のノード数は木の高さを掛けた高々 $l3^l$ である。

決定木の領域計算量は構築されるノード数により支配されるので、決定木の領域計算量はルールリストのルール長 l に関して $O(l3^l)$ である。

結果

提案した枝刈り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 Corei7-980X 3.33GHz, OS は CentOS Release6.2 である。

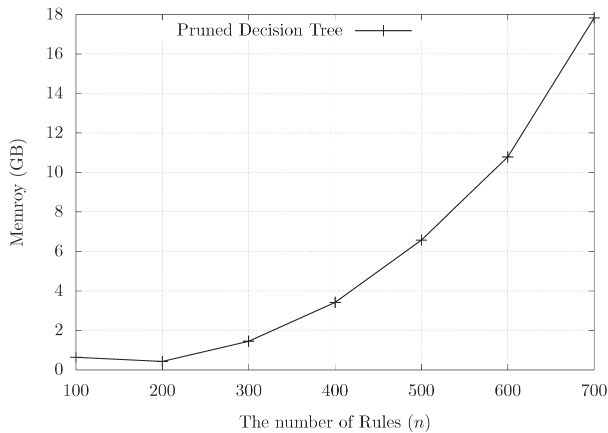

図 6. 決定木構築に必要なメモリ量 .

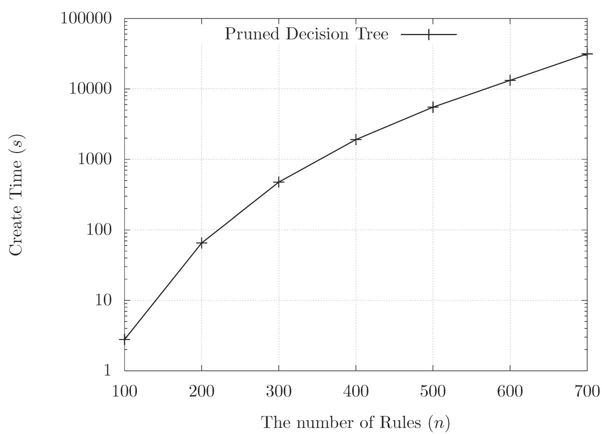

図 7. 決定木の構築時間 .

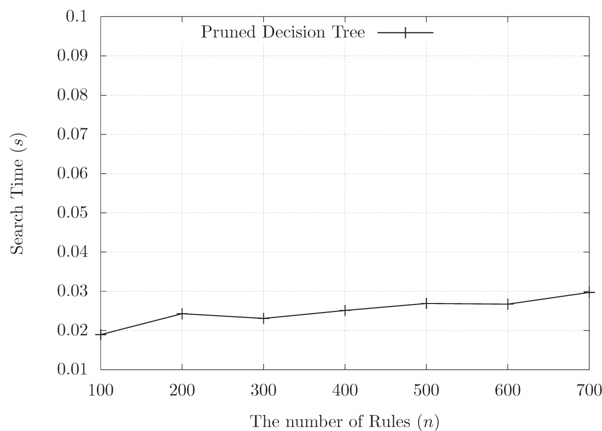

図 8. 決定木探索の探索時間 .

ルール数が 100 から 700 までのルールリストをランダムに作成して実験を行った。ルール長は 64 で、長さが 16 以上の連を丁度二つ持つよう設定した。パケットもランダムに作成した。

枝刈りを行わずに合致連集合の直積をそのまま取って構成する方法では、ルール数が 100 のルールリストでメインメモリ 24GB を使い切ってしまい決定木を構成できなかった。一方、提案枝刈り手法は決定木を構築することできた。結果を図 6 と図 7 に示す。図 8 に、決定木を用いた探索時間を示す。これより枝刈り法を適用して構成した決定木が、最も重要な性質である探索時間がルール数に依存しないという性質を保つことがわかる。

討論

本稿では Run-Based Trie に基づいて構成された決定木の枝刈り法を提案した。決定木の領域計算量は $O(l3^l)$ となるものの、提案した枝刈り法は長さ 16 以上の連を丁度二つ含むルールのルールリストではルール数 700 まで有効であることを計算機実験で確認した。本手法により素朴な構成法では構成できなかったルールリストに対しても決定木が構成きる。

本稿で提案した枝刈り法は合致連集合のラベルのみを基に枝刈りを行っており、どの連に合致しているかという情報は葉ノードまで活用されない。今後の課題は、このような情報にも基づいた決定木の枝刈り法を考案し、より多くのルール数に対して決定木を構成可能とすることである。

謝辞

本研究の一部は、神奈川大学総合理学研究所共同研究助成金（RIIS201903）の援助を受けた。

文献

1) Tanaka K, Mikawa K and Hikin M (2013) A heuristic algorithm for reconstructing a packet filter with dependent rules. *IEICE Trans. Commun.* **96**: 155-162.
2) Tanaka K, Mikawa K and Takeyama K (2013) Optimization of packet filter with maintenance of rule dependencies. *IEICE Communications Express* **2**: 80-85.
3) Tapdiya A and Fulp EW (2009) Towards optimal firewall rule ordering utilizing directed acyclical graphs. In:*Computer Communications and Networks, 2009. ICCCN 2009. Proceedings of 18th Internatonal Conference*. pp.1-6.
4) Taylor DE (2005) Survey and taxonomy of packet classification techniques. *ACM Comput. Surv.* **37**: 238-275.
5) Gupta P and McKeown N (2000) Classifying packets with hierarchical intelligent cuttings. *Micro. IEEE* **20**: 34-41.
6) Singh S, Baboescu F, Varghese G and Wang J (2003) Packet classification using multidimensional cutting. In: *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM* '03. pp. 213-224.
7) Li W and Li X (2013) Hybridcuts: A scheme combining decomposition and cutting for packet classification. In: *2013 IEEE 21st Annual Symposium on High-Performance Interconnects*. pp. 41-48.

8) Srinivasan V, Varghese G, Suri S and Waldvogel M (1998) Fast and scalable layer four switching. *SIGCOMM Comput. Commun. Rev.* **28**: 191-202.
9) Pagiamtzis K and Sheikholeslami A (2006) Contentaddressable memory (CAM) circuits and architectures: tutorial and survey. *IEEE. Solid-state Circuits* **41**: 712-727.
10) Ligatti J, Kuhn J and Gage C (2010) A packetclassification algorithm for arbitrary bitmask rules, with automatic time-space tradeoffs. In: *Proceedings of the International Conference on Computer Communication Networks (ICCCN)*. pp. 145-150.
11) Mikawa K and Tanaka K (2015) Run-based trie involving the structure of arbitrary bitmask rules. *IEICE Transactions on Information and Systems* E98.D(6): 1206-1212.