Vol.2020-AL-176 No.3
2020/1/30

ポインタ付連分割トライに基づく決定図構築法

原田 崇司[a)]　田中 賢[b)]　三河 賢治[c)]

概要：近年，不正アクセスや DRDoS 攻撃などのサイバー脅威の増加にともない，これらの攻撃に対する防御策の根幹技術であるパケット分類が注目されている．TCAM や ASIC などの特殊なハードウェアを実装していないネットワーク機器では，パケットと分類ルールリストとを線型探索することによってパケット分類を実現する．線型探索では，パケットはほとんど全てのルールと比較されてしまうため，ルール数の増加にともなう遅延の増大が問題となる．それゆえ，探索時間がルール数に依存しない分類アルゴリズムが求められる．そのような手法として本稿では，連分割トライから構築される決定図を用いた分類法を提案する．

キーワード：パケット分類，任意のビットマスク，トライ木，決定図

Decision Diagram Construction Based on Run-Based Trie with Pointers

1. はじめに

近年，不正アクセスや DRDoS 攻撃などのサイバー脅威の増加にともない，これらの攻撃に対する防御策の根幹技術であるパケット分類が注目されている．TCAM や ASIC などの特殊なハードウェアを実装していないネットワーク機器では，パケットと分類ルールリストとを線型探索することによってパケット分類を実現する．線型探索では，パケットはほとんど全てのルールと比較されてしまうため，ルール数の増加にともなう遅延の増大が問題となる．このため，パケットとルールの比較回数を最小とするようにルールを並び替える手法 [4,12,18]，ルールリストを新たに構築する手法が提案されてきた [15]．また，ルールリストによる線型探索ではなく，ルールリストからパケット分類のためのテーブル，表，決定木などの様々なデータ構造を構築して，それらを利用したパケット分類手法も提案されてきた [1,5–7,11,14]．

ルールリストのルール数を n，ルールの条件式の長さを l とすると，線型探索によるパケット分類の領域計算量と時間計算量はともに $O(nl)$ である．ルールの並び替えや再構築により遅延を減らすことは可能だが，探索時間は必ずルール数 n に依存してしまう．パケット分類による遅延は，ルール数の増加に大きな影響を受ける．これより，時間計算量の観点からは線形探索は最善のパケット分類手法ではない．

探索時間計算量がルール数 n に依存しない手法としては，Srinivasan らによる手法がある [11]．しかし，こ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．

任意の位置にビットマスクを含むルールに対応し，探索時間がルール数 n に依存しない手法として，多値決定グラフ (multi-valued decision diagram) に基づくパケット分類アルゴリズムが存在する [5]．また，我々は連分割トライ (Run-Based Trie, RBT) に基づく決定木による探索法を提案した [8]．それぞれの手法の探索時間計算量は $O(l)$, $O(l^2)$ とルール数 n に依存しないが，それぞれ領域計算量は $O(2^l)$, $O(n^l)$ と指数オーダである．これより，これらの手法は要求メモリ量が膨大になるという欠点がある．

RBT に基づく決定木の領域計算量が $O(n^l)$ とルール長 l に対して指数オーダであり，あまつさえ底がルール数 n である．これに対して，我々はポインタ付 RBT に基づく決定木探索法を提案した [17]．この探索法の時間計算量は $O(l)$ で領域計算量は $O(2^l)$ であり，時間計算量と領域計算量をともに改善している．

本稿では，ポインタ付 RBT に基づく決定木に対して等

a) harada.takashi@kochi-tech.ac.jp
b) ktanaka@info.kanagawa-u.ac.jp
c) mikawa@cais.niigata-u.ac.jp

表 1　任意のビットマスクを含むルールのルールリスト
Table 1　Bitmask rules.

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

価な節点を共有することにより，ポインタ付 RBT に基づく決定図構築法を提案する．更に，計算機実験により，提案手法の有効性を確かめる．

2. 連分割トライ

連分割トライ (Run-Based Trie, RBT) は, 決定リスト [10] のようなルールリストによって定義される函数を計算するためのデータ構造である [8]．大半のパケット分類アルゴリズムがプレフィックスルールとレンジルールを組合せたものだけに対応するのに対して，RBT は任意の位置にビットマスクを含むルール（任意のビットマスクルール）にも対応する．

以下では，パケットを長さ l のビット列とする．パケット分類のルールは，ルール番号 $i \in [n]$，長さ l の $0, 1, *$ を要素とする文字列 $b_1 b_2 \cdots b_l$ からなる．ただし，n はルールリストのルール数，$[n]$ は集合 $\{x \in \mathbb{N} \mid 1 \leq x \leq n\}$ を表しており，‘$*$’ はワイルドカードマスクであり，$0, 1$ のどちらにも合致することを意味する．任意のビットマスクルールからなるルールリストの例を表 1 に示す．

ルール r_i の条件式に対して，ワイルドカードマスク ‘$*$’ を含まない極大なビット列のことを r_i の連 (run) と呼ぶ．r_i が k 個の連を含むとき，各連の開始位置の添字が小さい順に $\rho_i^1, \rho_i^2, \ldots, \rho_i^k$ と表す．例えば，表 1 の r_1 は，3 つの連 0, 1, 00 を含み．各連の開始位置の添字は 1, 3, 5 であり，$\rho_1^1 = 0$, $\rho_1^2 = 1$, $\rho_1^3 = 00$ となる．

2.1 RBT の構築

RBT は，ルール長 l 個のトライ T_i $(1 \leq i \leq l)$ からなる森である．n 個のルールからなるルールリスト $\mathcal{R} = \langle r_1, r_2, \ldots, r_n \rangle$ に対して，トライ T_i をルールリスト中の i ビット目から始まる連によって構成する．表 1 のルールリストに対応する RBT を図 1 に示す．トライの点線と実線は 0 枝と 1 枝とを表す．更に，連 ρ_j^k がトライ T 上の経路 $p_1 p_2 \cdots p_n$ に対応するとき，T の節点 p_n に ρ_j^k という印を付ける．

ルールリスト $\mathcal{R}$ に含まれる全てのルール r_i に対して上記の方法を適用して，トライ $T_1, T_2, \ldots, T_l$ を構成する．トライ T_i の高さは高々，$l - i + 1$ となる．

2.2 RBT による素朴な探索法

本稿においてパケット分類とは，入力パケットに合致するルールの中で優先度が最も高いルールのルール番号を見つけることをいう．ただし，最も優先度が高いルールとはルール番号が一番小さいルールのことである．

例えば，パケット $p = 001011$ は，表 1 のルールリストの r_3 と r_5 とに合致する．そして，$3 < 5$ なので，表 1 のリストによる 011011 に対するパケット分類の結果は 3 となる．

本節では，我々が文献 [8] で提案した RBT による素朴なパケット分類法を概説する．この探索アルゴリズムは，全てのトライ $T_i (1 \leq i \leq l)$ をパケットの部分ビット列 $p_i p_{i+1} \cdots p_l$ で探索して，パケットに合致する連を集める．そして集めた連から合致するルールを計算して，合致するルールの中から最優先のルールを計算する．

例えば，パケット 001011 は，図 1 の RBT を太線のように探索し，連 $\rho_1^1, \rho_2^1, \rho_3^1, \rho_1^2, \rho_2^2, \rho_3^2, \rho_5^2, \rho_3^3, \rho_6^2$ に合致する．これらの連に合致するパケットは，ルール r_3, r_5 に合致するので，優先度の最も高いルール r_3 のルール番号 3 がパケット分類の結果として返される．

この探索法は，l 個のトライ全てを根から探索し，合致した連の照合と最優先ルールの計算を行う．トライ上の連の数は高々 $nl/2$ なので，探索時間計算量は $O(l^2 + nl)$ となる [8]．

3. ポインタ付連分割トライ

RBT の素朴な探索法は，トライ T_i を探索する際に $T_1, T_2, \ldots, T_{i-1}$ をどのように探索したかを考慮しないので，パケットの部分ビット列を何度も参照するという問題点がある．本章では，この冗長さを省いた，ポインタ付連分割トライ (ポインタ付 RBT) について説明する [16]．

RBT の素朴な探索法の問題点を，例を用いて説明する．図 1 の RBT をパケット 001011 で探索すると，$T_1, T_2, T_3, T_4, T_5, T_6$ のそれぞれにて，パケットの $p[1]p[2]p[3] (= 001)$, $p[2] (= 0)$, $p[3]p[4] (= 10)$, $p[4]p[5]p[6] (= 011)$, $p[5] (= 11)$, $p[6] (= 1)$ にアクセスする．この例では，パケットの $p[2]$, $p[3]$, $p[4]$, $p[5]$, $p[6]$ は複数回アクセスされており無駄である．

トライ T_i を高さ $k (k \leq w - i + 1)$ まで探索できても，トライ T_{i+1} を根から探索しているので，このような冗長さが生じている．しかし，トライ T_{i+1} の根から高さ $k - 1$ の節点までの経路に連が存在する可能性があるので，このままでは高さ $k + 1$ から探索することはできない．

そこで，トライ番号の小さい上位のトライから番号の大きい下位のトライへポインタを張り，下位のトライの連を

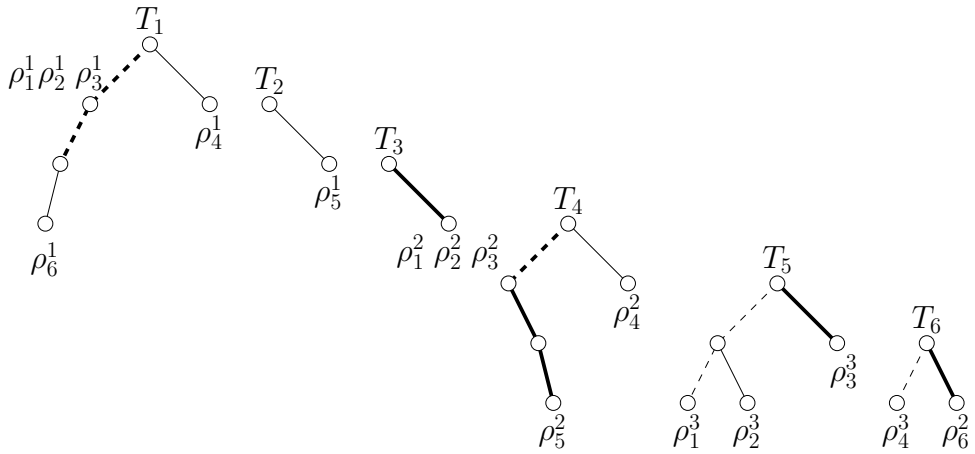

図 1 表 1 から構成される RBT
Fig. 1 Run-based trie for rule list in Table 1.

上位のトライへ複製する．こうすると，パケット $p[1], p[2], \ldots, p[l]$ の各要素に一回アクセスするだけで探索できる．図 1 の RBT に連とポインタを追加した，ポインタ付 RBT を図 2 に示す．

ポインタ付 RBT の探索法は RBT の素朴な探索法と同様である．ただし，RBT の探索と異なり高々 l だけの節点を辿るので，その時間計算量は $O(l + nl) = O(nl)$ となる．図 2 の太線はパケット 001011 によるポインタ付 RBT の探索を表している．

4. ポインタ付 RBT に基づく決定図

オリジナルの RBT [8] に基づく決定木探索法の探索時間計算量と領域計算量が $O(l^2)$ と $O(n^l)$ である．これに対して，我々は探索時間計算量と領域計算量が $O(l)$ と $O(2^l)$ となる，ポインタ付 RBT に基づく決定木探索法を提案した [17]．この決定木は同じ部分木を含むので，多値決定図のように等価節点の共有と冗長節点削除を行うことにより要求メモリ量を減らすことができる．よって，本章ではポインタ付 RBT に基づく決定図構築法とその探索法とを示す．

4.1 節点番号と到達可能節点集合

ポインタ付 RBT に基づく決定図の構築法は，文献 [17] で提案したポインタ付 RBT から構築される決定木の構築法と途中まで全く一緒である．

ポインタ付 RBT に基づく決定図を構築するために，ポインタ付 RBT に二つの操作を行う．初めに，連が付与されている節点に対して識別番号を割当てる．次に，各節点に対して，その節点から識別番号を持つ節点の中で到達可能な節点を探索し，到達可能な節点の識別番号の集合を与える．ただし，節点 u から識別番号を持つ節点 v へ到達可能であるとは，u から v への経路が存在して且つ u から v への全ての経路において，識別番号を持つ節点が存在しないことである．図 2 のポインタ付 RBT に対して，識別番号と到達可能節点集合を追加したポインタ付 RBT を図 3 に示す．図 3 の節点について，{} で囲まれた要素が，その節点から到達可能な節点を表している．また，{} の左にある数字は，その節点の識別番号を表す．

4.2 決定木構築

ポインタ付 RBT の探索によって合致する連の組合せは有限通りである．この組合せを全て列挙して，各組合せにおける最優先ルールを与えたものがポインタ付 RBT に基づく決定木である [17]．

ポインタ付 RBT の探索における連の合致パタンは，到達可能節点集合を用いることにより，実際にポインタ付 RBT を探索することなく列挙できる．図 3 のポインタ付 RBT に基づいて構築される決定木の一部を図 4 に示す．決定木の丸で囲まれた節点は内部節点を表し，四角で囲まれた節点は終端節点（葉）を表す．節点の数字は，ポインタ付 RBT を探索することにより，その節点に到達した時点での最優先ルールを表している．枝の横にある数字は，ポインタ付 RBT を探索して到達した節点の識別番号に対応する．

ポインタ付 RBT をパケットのビット列で探索して，識別番号を持つ節点に到達したら，その識別番号で決定木を探索する．このように決定木を辿り，到達した葉に付与されている値が，パケットに合致する最優先ルールとなる．図 4 の太線は，パケット 001011 で決定木を探索した際の経路を表す．ポインタ付 RBT を高々 l 回辿ることによって決定木を探索できるので，決定木探索の時間計算量は $O(l)$ となる．

ポインタ付 RBT のトライ $T_1, T_2, \ldots, T_l$ の全てが，高さ 1，要素数 3 の二分木であるとき，このポインタ付 RBT に基づく決定木は高さ l の完全二分木になる．よって，ポインタ付 RBT に基づく決定木の領域計算量は $O(2^l)$ であ

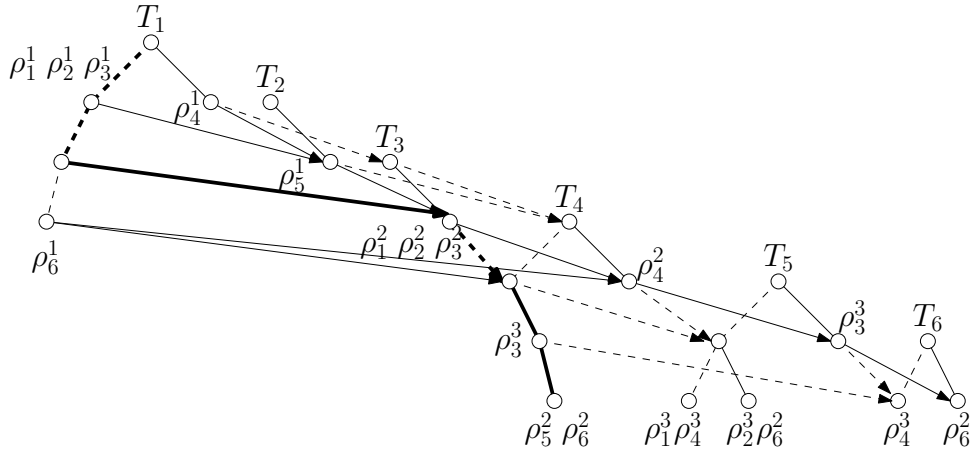

図 2 ポインタ付 RBT
Fig. 2 Run-based trie with pointers.

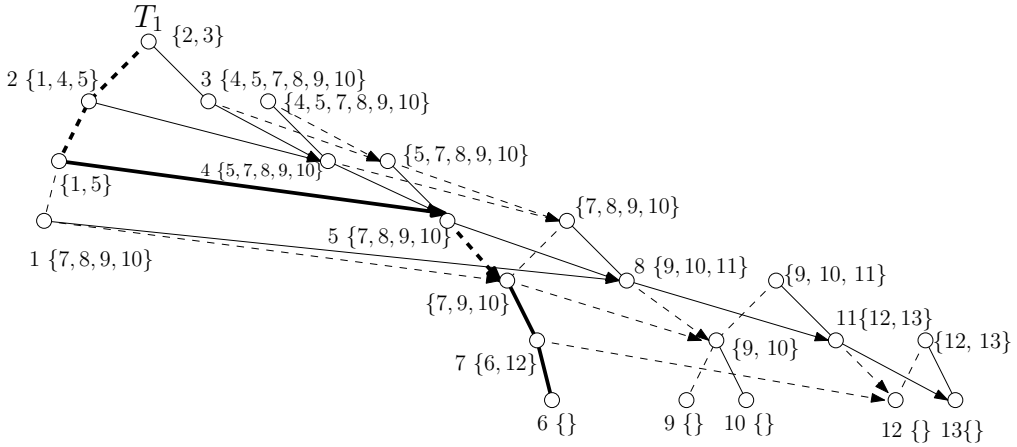

図 3 節点番号，到達可能節点集合の追加
Fig. 3 Adding node numbers and reachable sets.

る [17].

4.3 決定図構築

ポインタ付 RBT に基づく決定木に対して，Binary Decision Diagram (BDD) [3] や Zero-suppressed Binary Decision Diagram (ZDD) [9] と同じように，等価節点の共有と冗長節点の削除を行って，ポインタ付 RBT に基づく決定図を構築する．RBT に基づく決定図において，節点 v と u が等価であるとは，v と u の最優先ルールが等しく，v の子供と u の子供が全て等しいことをいう．また，節点 v が冗長であるとは，節点 v から張られる全ての枝が同じ節点を指すことをいう．RBT に基づく決定図における等価節点の共有と冗長節点の削除を図 5，6 に示す．

更に，ポインタ付 RBT に基づく決定図を構築するにあたって，BDD や ZDD と同じようにユニークテーブルを用いて節点を管理する [2,9].

ポインタ付 RBT に基づく決定図を構築するにあたって，Algorithm 1 に示すサブルーチン getnode を定義する．getnode は，節点 v と v の子 $(c_1, c_2, \ldots, c_m)$ を入力として受け取り，v と $(c_1, c_2, \ldots, c_m)$ の組がユニークテーブルに含まれているかを判定し，含まれていたらそのアドレスを返す．ユニークテーブルに含まれない場合は，その組に対応するエントリを生成してユニークテーブルに追加して，そのアドレスを返す．

Algorithm 1 getnode

Input: 決定図の節点 v と v の子 $(c_1, c_2, \ldots, c_m)$
1: **if** $c_1 = c_2 = \ldots = c_m$ **then**
2: **return** c_1
3: **end if**
4: $V \leftarrow$ ユニークテーブルに $(v, (c_1, c_2, \ldots, c_m))$ が存在するか
5: **if** V が存在 **then**
6: **return** V
7: **end if**
8: $V \leftarrow (v, (c_1, c_2, \ldots, c_m))$ に対応する節点を生成
9: V をユニークテーブルに追加
10: **return** V

ポインタ付 RBT に基づく決定図構築アルゴリズムを Algorithm 2 に示す．Algorithm 2 は，再帰的なアルゴリズムである．このアルゴリズムは，ポインタ付 RBT の識別番号の列 $(i_0, i_1, \ldots, i_k)$ を入力として受け取り，i_k に対

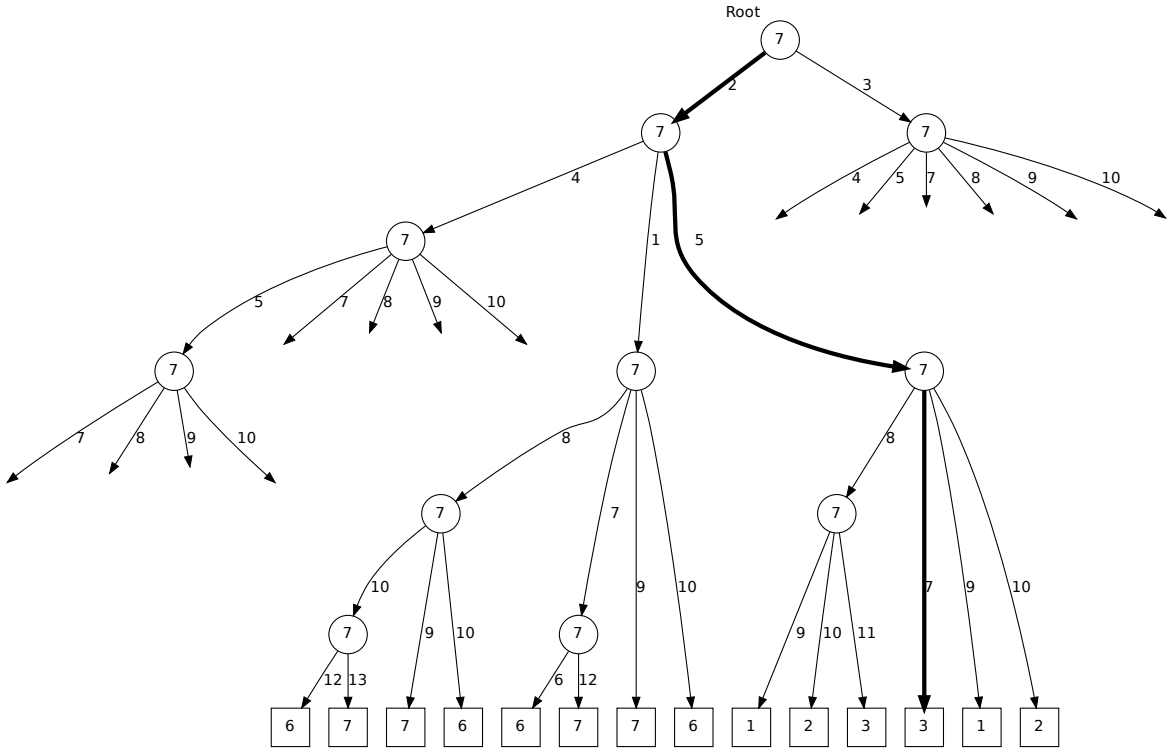

図 4　決定木
Fig. 4　Decision Tree.

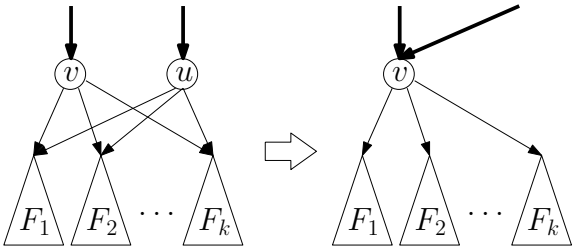

図 5　等価節点の共有
Fig. 5　Sharing.

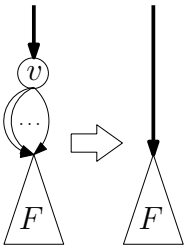

図 6　冗長節点の削除
Fig. 6　Deletion.

応する決定図の節点 d_{i_k} を返す．初めに，識別番号の列とそれらに対応する連の集合から，この時点での最優先ルール *candidate* を計算する．次に，識別番号 i_k の到達可能集合 S_{i_k} が空集合ならば，*candidate* に対応するアドレス（決定図の節点）を返す．空集合でないならば，$(i_0, i_1, \ldots, i_k)$

Algorithm 2 ConstructDD

Input: ポインタ付 RBT の識別番号の列 $(i_0, i_1, \ldots, i_k)$
1: $(i_0, i_2, \ldots, i_k)$ から最優先ルールを計算
2: *candidate* を i_k における最優先ルールの番号とする
3: i_k に対応するポインタ付 RBT の到達可能節点集合を S_{i_k} とする
4: **if** $S_{i_k} = \emptyset$ **then**
5:　　**return** *candidate* に対応する終端節点のアドレス
6: **end if**
7: $j \leftarrow 1$
8: **while** $j \leq |S_{i_k}|$ **do**
9:　　w を S_{i_k} の j 番目の要素とする
10:　　$c_j \leftarrow$ constructDD$((i_0, i_1, \ldots, i_k, w))$
11: **end while**
12: **return** getnode$(d_{i_k}, c_1, c_2, \ldots, c_{|S_v|})$

に対して S_{i_k} のそれぞれの要素を加えた列を constructDD に与え，d_{i_k} の子を生成する．決定図の節点は，Algorith 1 で等価な節点がないか確認されてから生成されている．

図 3 のポインタ付 RBT から構成される決定図を図 7 に示す．決定木の節点数が 70 となるのに対して，決定図の節点数は 23 となり，表 1 のルールリストでは 60%以上の節点を削減している．

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，メモリ 4GB，CPU

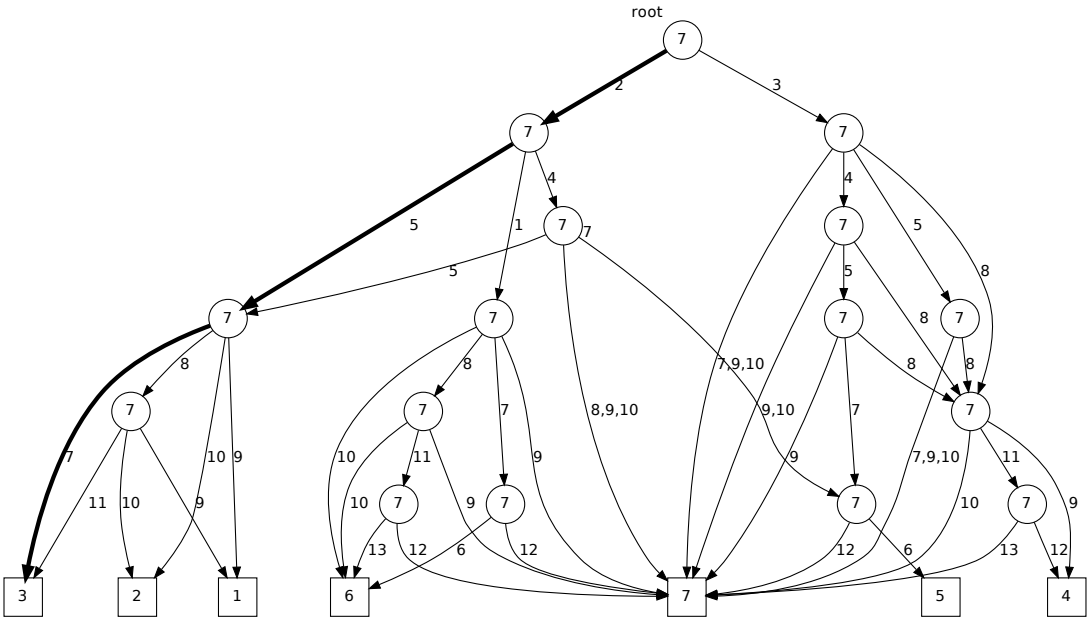

図 7　決定図
Fig. 7　Decision Diagram.

が Intel(R) Core(TM) i5-8400 CPU @ 2.80GHz，OS は FreeBSD 12.0-RELEASE である．

実験に使用したルールリストはパケット分類アルゴリズムのベンチマークツールである ClassBench [13] によって生成した．ルールは送信元アドレス，送信先アドレス，送信元ポート，送信先ポート，プロトコルの 5 つのフィールドに分かれている．それぞれのフィールドの長さは 32,32,16,16,8 ビットなので，ルール長 l は 104 となる．ルール生成の際には，Access Control List (acl)，Firewall (fw)，IP Chain (ipc) を模した 3 つのシードファイルを用いた．

ルール数 100-500 のルールリストを各々 5 つずつ生成して，ポインタ付 RBT に基づく決定木と決定図について構築時間と節点数とを計測した．

ポインタ付 RBT に基づく決定木と提案手法の構築時間と節点数とを図 8, 9, 10, 11 に示す．提案手法の構築時間を表す図 9 は，y 軸が対数目盛であることに注意されたい．

図 8, 9 より，ACL は構築時間が FW と IPC に比べて長い．これは，ACL が他の二つに比べて連の種類が多く，決定木・決定図を構築する際の基本的な組合せの数が多いからであると考える．

図 8, 9 に示すように，ルール数 500 において提案手法は決定木よりも 100 倍の構築時間を必要としてしまっている．けれども，図 10, 11 に示すように，ルール数 500 で提案手法の節点数は決定木の節点数の 1/100 である．

これより，決定木と同程度の速度で決定図を構築することができれば，提案手法は有効であると考える．

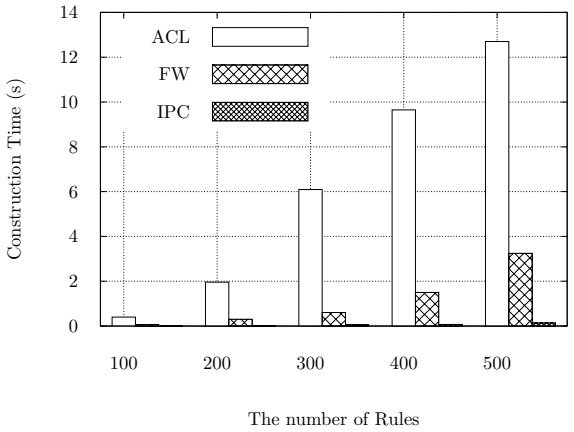

図 8　決定木 [17] の構築時間
Fig. 8　Construction time for the decision tree [17].

6. おわりに

本稿では，ポインタ付 RBT に基づく決定図の構築法とその探索法とを提案した．決定図は，決定木を構築できないサイズのルールリストに対しても適用できる可能性がある．

今後の課題は主に四つある．一つ目は，提案手法のデータ構造を高速に構築できるように改良することである．ポインタ付 RBT から構築される決定木 [17] と同様に，提案手法には適用しやすいルールリストとそうでないルールリストが存在する．これより，提案手法がどのようなルール

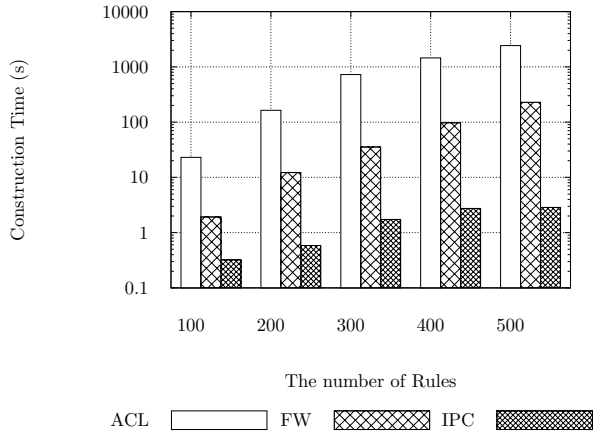

図 9　提案手法の構築時間

Fig. 9　Construction time for the proposed algorithm.

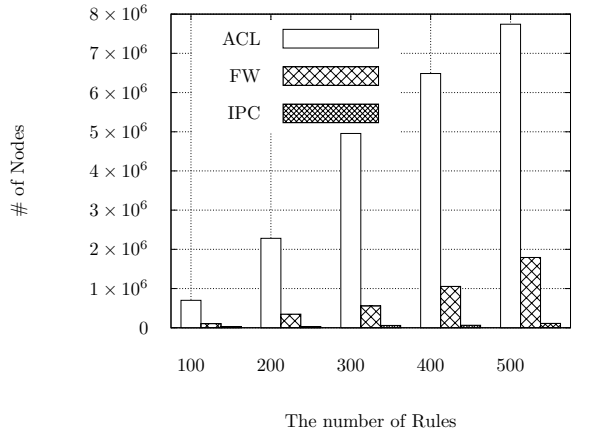

図 10　決定木 [17] の節点数

Fig. 10　# of nodes for the decision tree [17].

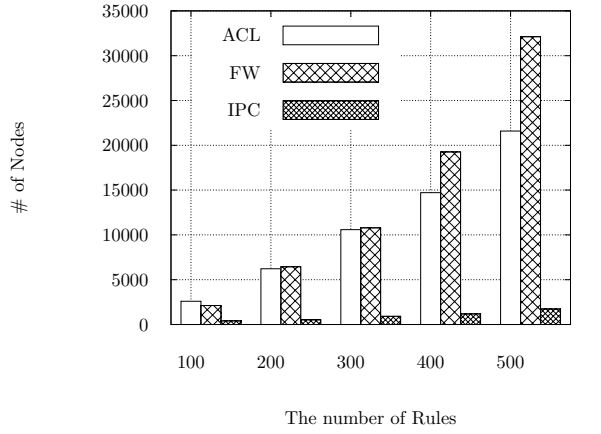

図 11　提案手法の節点数

Fig. 11　# of nodes for the proposed algorithm.

リストに対して有効であるかを調べることが二つ目の課題である．三つ目の課題は，他のパケット分類アルゴリズムとの比較である．最後の課題は，探索を行う際に，ポインタ付 RBT を使用せずに決定木のみで探索できるようにすることである．

謝辞　提案手法の実装に関して助言を下さった，高知工科大学情報学群の竹内聖悟講師に感謝する．また，本研究の一部は，JSPS 科研費 19K11953, 19K11959 の助成を受けたものである．

参考文献

[1] Baboescu, F. and Varghese, G.: Scalable Packet Classification, *SIGCOMM Comput. Commun. Rev.*, Vol. 31, No. 4, pp. 199–210 (online), DOI: 10.1145/964723.383075 (2001).

[2] Brace, K. S., Rudell, R. L. and Bryant, R. E.: Efficient implementation of a BDD package, *27th ACM/IEEE Design Automation Conference*, pp. 40–45 (online), DOI: 10.1109/DAC.1990.114826 (1990).

[3] Bryant: Graph-Based Algorithms for Boolean Function Manipulation, *IEEE Transactions on Computers*, Vol. C-35, No. 8, pp. 677–691 (online), DOI: 10.1109/TC.1986.1676819 (1986).

[4] Fuchino, T., Harada, T., Tanaka, K. and Mikawa, K.: Acceleration of Packet Classification Using Adjacency List of Rules, *2019 28th International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–9 (online), DOI: 10.1109/ICCCN.2019.8846923 (2019).

[5] 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).

[6] 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).

[7] Ligatti, J., Kuhn, J. and Gage, C.: A Packet-classification Algorithm for Arbitrary Bitmask Rules, with Automatic Time-space Tradeoffs, *Proceedings of the International Conference on Computer Communication Networks (ICCCN)*, pp. 145–150 (2010).

[8] 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).

[9] Minato, S.: Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems, *30th ACM/IEEE Design Automation Conference*, pp. 272–277 (online), DOI: 10.1145/157485.164890 (1993).

[10] Rivest, R. L.: Learning Decision Lists, *Mach. Learn.*, Vol. 2, No. 3, pp. 229–246 (online), DOI: 10.1023/A:1022607331053 (1987).

[11] Srinivasan, V., Varghese, G., Suri, S. and Waldvogel, M.: Fast and Scalable Layer Four Switching, *SIGCOMM Comput. Commun. Rev.*, Vol. 28, No. 4, pp. 191–202 (1998).

[12] T. Harada, K. Tanaka, K. M.: Simulated Annealing Method for Relaxed Optimal Rule Ordering (2020 (accepted)).

[13] Taylor, D. E. and Turner, J. S.: ClassBench: A Packet Classification Benchmark, *IEEE/ACM Trans.*

Netw., Vol. 15, No. 3, pp. 499–511 (online), DOI: 10.1109/TNET.2007.893156 (2007).

[14] Yingchareonthawornchai, S., Daly, J., Liu, A. X. and Torng, E.: A sorted partitioning approach to high-speed and fast-update OpenFlow classification, *2016 IEEE 24th International Conference on Network Protocols (ICNP)*, pp. 1–10 (online), DOI: 10.1109/ICNP.2016.7784429 (2016).

[15] 田中　賢，伊藤　聖：ネットワーク機器の負荷を軽減するフィルタリングルール再構成法，電子情報通信学会論文誌 B 通信， Vol. 88, No. 5, pp. 905–912（オンライン），入手先 ⟨https://ci.nii.ac.jp/naid/110003203101/⟩ (2005).

[16] 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，信学技報， Vol. 116, No. 315, pp. 13–18 (2016).

[17] 石川裕樹，原田崇司， 田中賢，三河賢治：ポインタ付 RBT に基づく決定木構築法 (2020 掲載予定).

[18] 文岩涼祐，山田敏規：ファイアウォールルール整列問題に対する厳密アルゴリズム，情報処理学会研究報告アルゴリズム (AL)，Vol. 2019-AL-175, No. 15, pp. 1–6 (2019).