# A Ternary Unification Framework for Optimizing TCAM-Based Packet Classification Systems

Author: Eric Norige, Alex X. Liu, and Eric Torng

Publisher: ANCS 2013

Presenter: Chang Chen

Date: 10/29/2014

# Outline

I. Introduction
II. TUF Framework
III. Prefix Minimization Using Tries
IV. Ternary Minimization Using ACLs
V. Experimental Results
VI. Conclusions

# I. Introduction

## 1.1 Background and Motivation

- TCAM-based packet classification
  - Each filter is stored in a TCAM entry in ternary format: 0, 1 and $*$ (don't care)
  - All of the filters are stored in TCAM with decreasing priority order.
  - When a packet arrives, all the entries in the TCAM are searched in parallel and the matched filter at lowest index is reported.

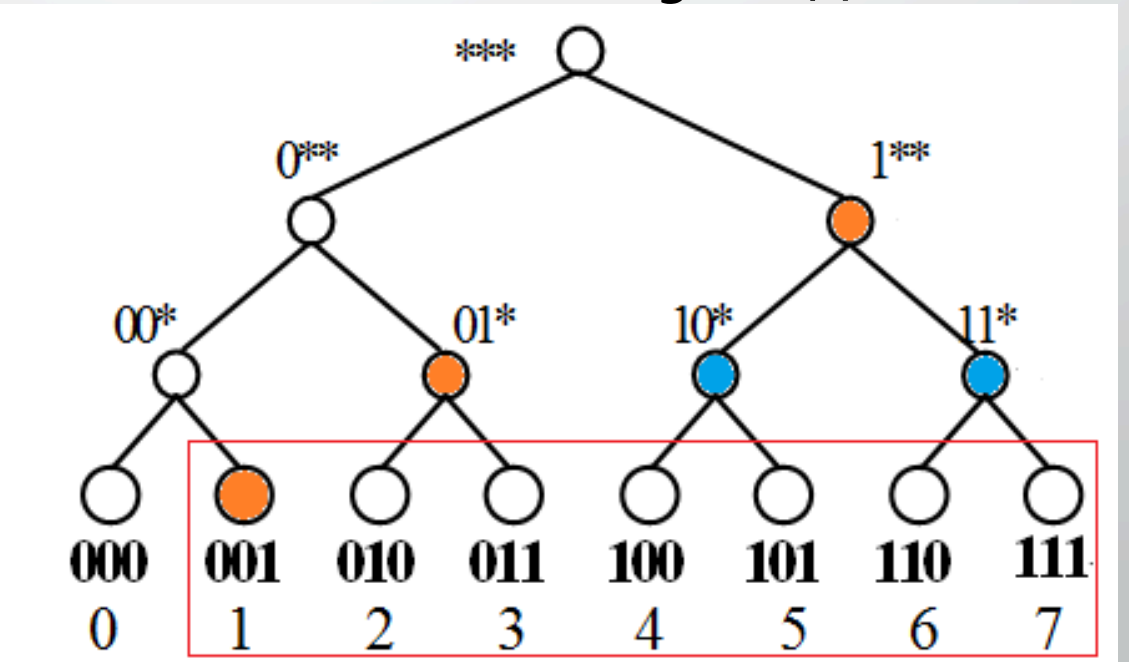
| Rule | Source IP | Dest. IP | Source Port | Dest. Port | Protocol | Action |
|---|---|---|---|---|---|---|
| $r_1$ | 1.2.0.0/16 | 192.168.0.1 | [1,65534] | [1,65534] | TCP | accept |
| $r_2$ | * | * | * | 6881 | TCP | discard |
| $r_3$ | * | * | * | * | * | accept |

Table 1: An example packet classifier

# I. Introduction (Cont.)

- The well known range expansion problem exacerbates the problem of limited capacity TCAMs (source and destination port fields)
  - Converting packet classification rules to ternary format typically results in a much larger number of TCAM entries

Convert the range [ 1 , 7 ]




[ 1 , 7]= 001 + 01* + 1**

# I. Introduction (Cont.)

## 1.2 Problem Statement

- A classifier $C$ is a function from binary strings of length $w$ to some decision set $D$; that is, $C : \{0, 1\}^w \rightarrow D$
- A TCAM classifier $T$ is an ordered list of rules $r1, r2, \ldots, rn$. Each rule has a ternary predicate $p_i \in \{0, 1, *\}^w$ and a decision $d_i \in D$
- The decision of a TCAM classifier $T$ for an input $p \in \{0, 1\}^w$ $T(p)$ is the decision of the first matching rule in $T$
- *TCAM Classifier Compression* problem: given a classifier $C$, construct a minimum size TCAM classifier $T$ that implements $C$

# I. Introduction (Cont.)

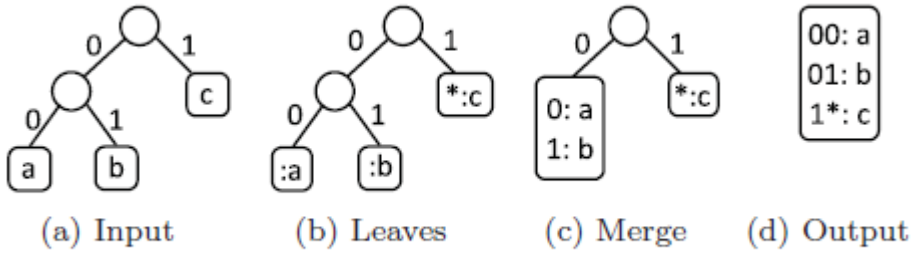
## 1.3 Limitations of Prior Art

- Prior TCAM Classifier Compression algorithms fall into two categories:
  - List based algorithms (*e.g.*, Bit Weaving [18], Redundancy Removal [13], Dong's scheme [8])
  - Tree based algorithms (*e.g.*, TCAM Razor [14] and Ternary Razor [18])
    - A key limitation of prior tree based algorithms is that at each tree level, they only try to optimize the current dimension and therefore miss some optimization opportunities.

# I. Introduction (Cont.)

## 1.4 Proposed Approach: TUF (Ternary Unification Framework)

# II. TUF Framework



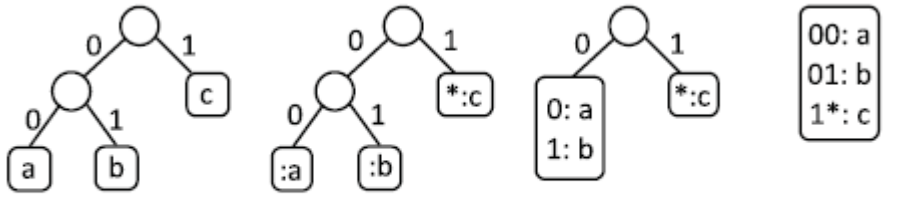

(a) Input (b) Leaves (c) Merge (d) Output

Figure 1: Structural Recursion Example

## 2.1 TUF Outline

- TUF takes a classifier as input and returns an optimized TCAM classifier a output
  - The first step of TUF is to represent the classifier as a BDD, where every node has zero or two children and the decisions are in the leaves.
  - The second step of TUF converts the leaves of the BDD into instances of a ternary data structure (tries, nested tries, and TCAM classifiers.).
  - The third step, the core of TUF, merges these ternary data structures to form ternary data structures that encode larger sections of the input space.

# II. TUF Framework




(a) Input (b) Leaves (c) Merge (d) Output

Figure 1: Structural Recursion Example

## 2.1 TUF Outline

- Two decisions that define a specific TUF algorithm:
  - (1) the ternary data structure to represent the intermediate classifiers
  - (2) the procedure to combine intermediate classifiers

# II. TUF Framework (Cont.)

- TUF requires that the ternary data structure support two operations:
  - Singleton: converts a BDD leaf to a ternary data structure
  - LRMerge: joins two ternary data structures *A* and *B* into one, *A* + *B*.

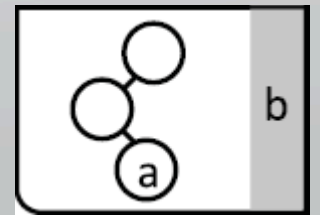
**Algorithm 1:** TUFCore(c)

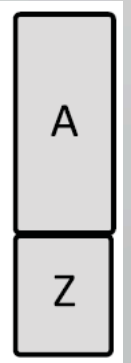
```
   Input: A 2-tree c representing a classifier
   Output: An equivalent ternary data structure
1  switch c do
2      case Leaf dec
3          return Singleton (dec);
4      case Node(left,right)
5          LeftSol := TUFCore (left);
6          RightSol := TUFCore (right);
7          return LRMerge (LeftSol,RightSol);
```

# II. TUF Framework (Cont.)

## 2.2 Efficient Solution Merging

- The minimum-size solution for a small part of the input space is often not the best representation for it in the context of the complete classifier.
- By keeping multiple representations at each step, the right representation of a subtree can be used to create the final solution.
  - This could cause an exponential amount of work to be spent creating and combining them, leading to an impractical algorithm.
- Use backgrounds as a way to limit the number of combinations that are created, allow pruning of useless solutions from the solution set and improve the speed of merging solutions.
  - A ternary classifier can be divided into two classifiers: a foreground of higher precedence and a background of lower precedence.



# II. TUF Framework (Cont.)

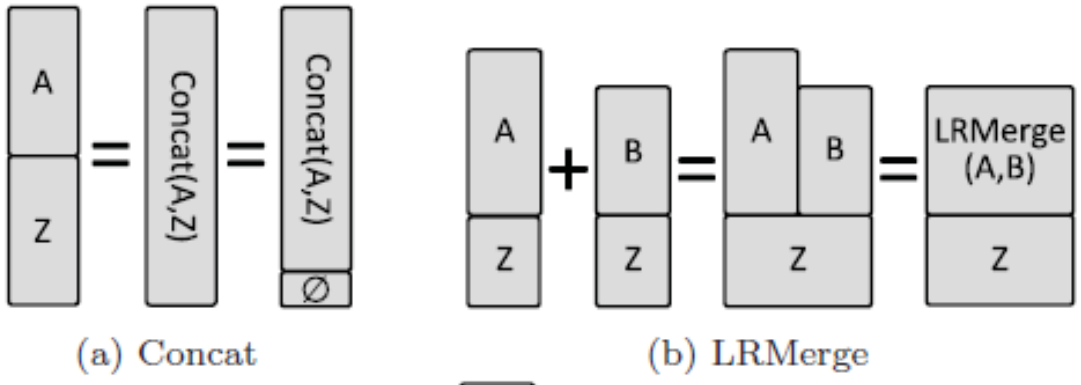
NOTE
EB: the solution as its background is empty.

- This solution set has two split classifiers, one that encodes the decision in the foreground, and one that encodes it in the background. The solution set for a BDD leaf with decision *d* is

$$\left\{ \frac{\texttt{Singleton}(d)}{\emptyset}, \frac{\emptyset}{\texttt{Singleton}(d)} \right\} \quad (1)$$

- $F/B$ to represent a classifier split into separate foreground, *F*, and background, *B*, ternary data structures.
- Concat(*A*,*Z*), denoted *A*,*Z*, joins a foreground and background ternary classifier into a single ternary classifier.

# II. TUF Framework (Cont.)




(a) Concat (b) LRMerge

(c) LRMerge with EB

Figure 2: TUF operations w/ backgrounds

$$\left\{\frac{A}{X}, \frac{B}{Y}, \frac{C}{\emptyset}\right\} + \left\{\frac{D}{X}, \frac{E}{\emptyset}\right\} = \left\{\frac{A+D}{X}, \frac{B+E}{Y}, \frac{C+E}{\emptyset}\right\}.$$

# II. TUF Framework (Cont.)

$$\left\{\frac{A}{X}, \frac{B}{Y}, \frac{C}{\emptyset}\right\} + \left\{\frac{D}{X}, \frac{E}{\emptyset}\right\} = \left\{\frac{A+D}{X}, \frac{B+E}{Y}, \frac{C+E}{\emptyset}\right\}.$$

**Algorithm 2:** SetMerge(l,r)

**Input:** solution sets $l$ and $r$
**Output:** merged solution set

```
1 Out = empty solution set;
2 NullLeft = foreground of ∅ in l;
3 NullRight = foreground of ∅ in r;
4 foreach bg in l.backgrounds ∪ r.backgrounds do
5 |   ForeLeft = foreground of bg in l or NullLeft;
6 |   ForeRight = foreground of bg in r or NullRight;
7 |   Out.add(LRMerge (ForeLeft,ForeRight),bg);
8 return Out
```

# II. TUF Framework (Cont.)

$$\left\{\frac{A}{X}, \frac{B}{Y}, \frac{C}{\emptyset}\right\} + \left\{\frac{D}{X}, \frac{E}{\emptyset}\right\} = \left\{\frac{A+D}{X}, \frac{B+E}{Y}, \frac{C+E}{\emptyset}\right\}.$$

- Backgrounds simplify LRMerge's search for commonalities by allowing LRMerge to focus on merging sibling ternary data structures that have the same background, instead of trying to merge all pairs of solutions.
- To simplify a set of solutions, TUF incorporates a cost function Cost(*C*) which returns the cost of any ternary classifier. Let *A* be the foreground of the EB in a solution set. For any solution $X/Y$ , if Cost(*A*) ≤ Cost(*X*), then TUF removes $X/Y$ from the set.
- TUF can also replace the EB by $\langle X,Y\rangle/\emptyset$ if there is a solution $X/Y$ for which Cost(*A*) > Cost( 〈 *X*, *Y* 〉 ).

# II. TUF Framework (Cont.)

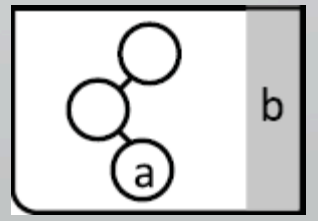
**Algorithm 3:** TUFCore(b) with backgrounds

```
   Input: A BDD b
   Output: A solution set of (fg,bg) pairs
 1 switch b do
 2     case Leaf d          /* BDD Leaf w/ decision d */
 3         return {(Singleton (d),∅),
 4                 (∅,Singleton (d))};
 5     case Node(left,right)          /* Internal Node */
 6         LeftPairs := TUFCore (left);
 7         RightPairs := TUFCore (right);
 8          //next line uses LRMerge;
 9         MergedPairs := SetMerge (LeftPairs,
           RightPairs);
10         Solutions := Concat to all of MergedPairs;
11         BestSol := lowest Cost solution in Solutions;
12         MergedPairs.removeIf(Cost (x) ≥ Cost
           (BestSol));
13         MergedPairs.add(BestSol, ∅);
14         if at field boundary then
15             Encap (MergedPairs);
16             MergedPairs.add(∅, Encap (BestSol));
17         return MergedPairs;
```

# III. Prefix Minimization Using Tries

## 3.1 1-Dimensional Prefix Minimization

- Singleton(*d*): produce a classifier where all inputs have decision *d*, simply create a trie node with decision *d*.
- The Cost(t) of a trie *t:* the number of nodes with a decision, which corresponds to the number of TCAM rules needed for that trie.
- LRMerge(l,r) two tries: we create a new trie node with no decision and assign the left child as *l* and the right child as *r*.
- Concat(f, b): only has to handle the case where the foreground has a root node without decision and the background is a singleton trie node with a decision.



# III. Prefix Minimization Using Tries (Cont.)

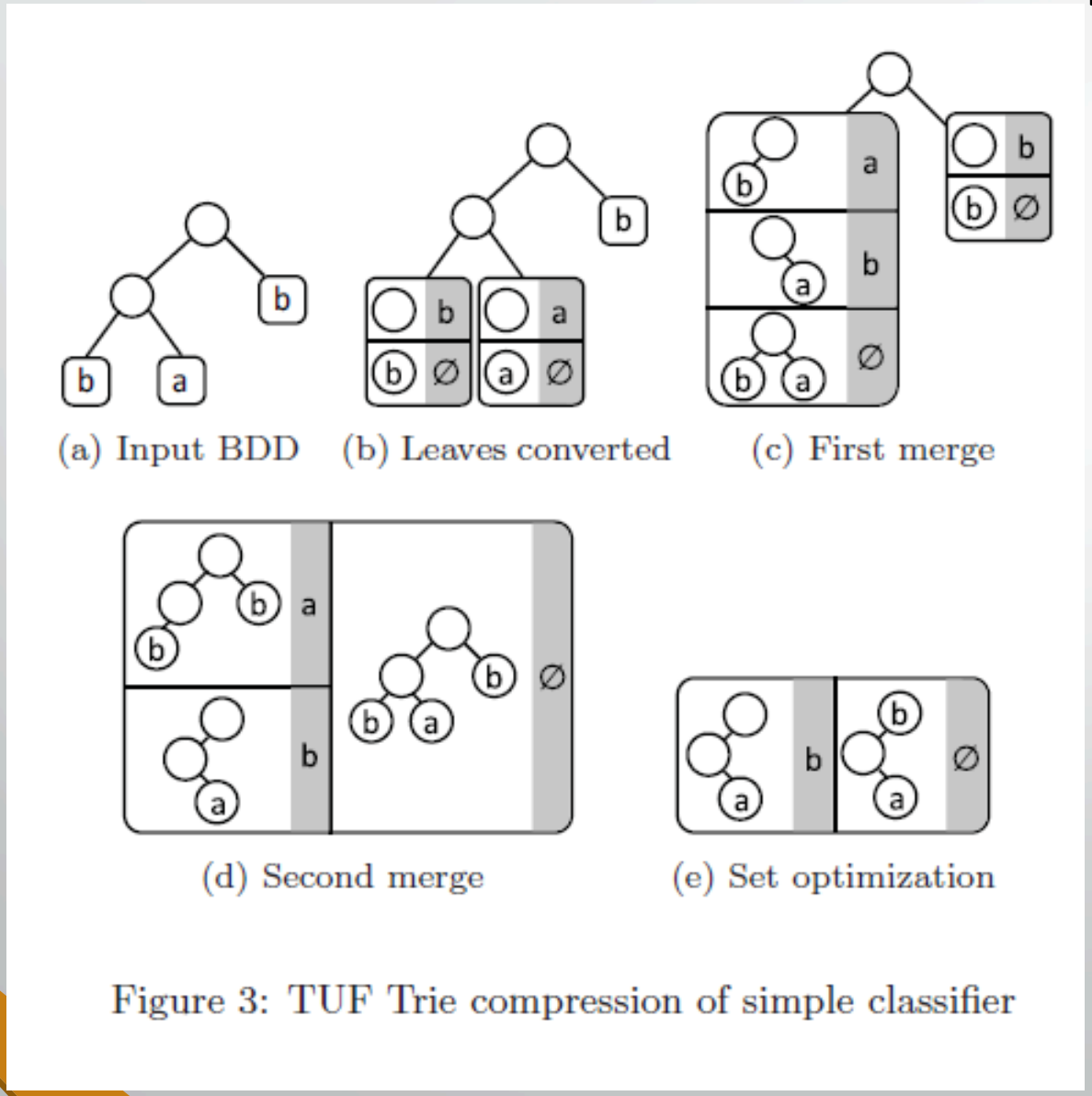



Figure 3: TUF Trie compression of simple classifier

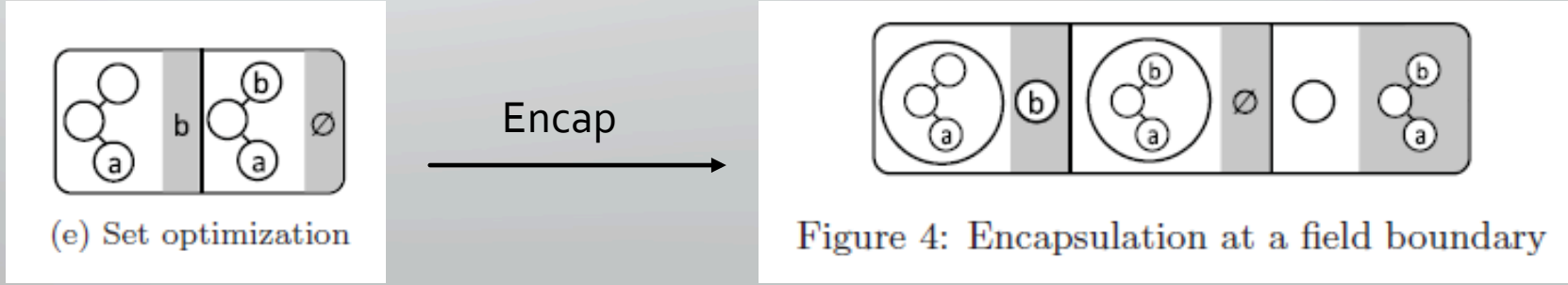
$$\left\{\frac{\texttt{Singleton}(d)}{\emptyset}, \frac{\emptyset}{\texttt{Singleton}(d)}\right\} \quad (1)$$

$$\left\{\frac{A}{X}, \frac{B}{Y}, \frac{C}{\emptyset}\right\} + \left\{\frac{D}{X}, \frac{E}{\emptyset}\right\} = \left\{\frac{A+D}{X}, \frac{B+E}{Y}, \frac{C+E}{\emptyset}\right\}.$$

# III. Prefix Minimization Using Tries (Cont.)

## 3.2 Multi-Dimensional Prefix Minimization

- To represent a multi-dimensional prefix classifier, tries of tries are the natural solution.
- In a trie of tries, each decision of an *n*-dimensional trie is an (*n* – 1)-dimensional trie.
- Tries of tries are turned into decisions for the next level trie at field boundaries using an **Encap** function.
- One new solution is added at this step (the rightmost solution in Figure 4) where the EB is set as a background decision to an empty foreground.




(e) Set optimization

Figure 4: Encapsulation at a field boundary

# IV. Ternary Minimization Using ACLs

- Merging TCAM classifiers without factoring out any commonalities can be done by prefixing all rules in the left input by 0 and those from the right input by 1 and concatenating them: $A + B = \langle 0A, 1B \rangle$ . As there is no overlap between the two groups of rules, The order of concatenation doesn't matter.
- Factoring out commonalities: $\langle T1, x, B1 \rangle + \langle T2, x, B2 \rangle = \langle (T1 + T2), *x, (B1 + B2) \rangle$


Figure 5: Recursive merging left and right ACLs

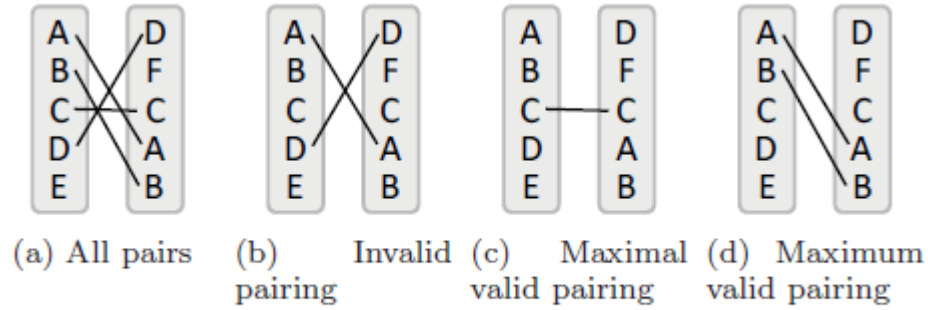



Figure 6: ACL pairing example

The problem of finding a maximum pairing can be reduced to the maximum common subsequence problem.

# V. Experimental Results

| Cat. | Count | Avg # Non-Red. | Avg. # Prefix Exp. |
|---|---|---|---|
| Small | 13 | 9 | 1578 |
| Large | 8 | 3221 | 7525 |
| Med. | 17 | 209 | 641 |

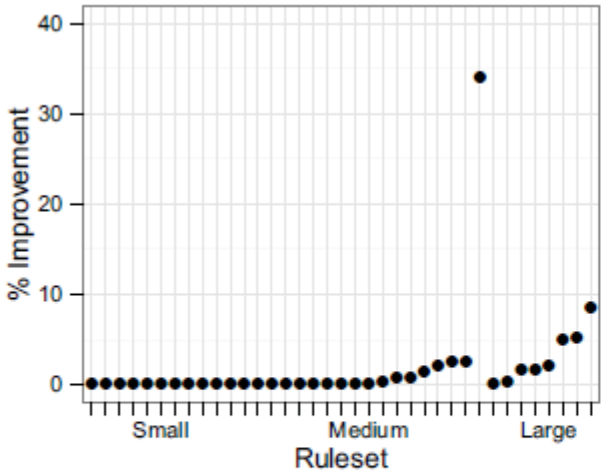
Table 2: Classifier categories

ACR: Average Compression Ratio

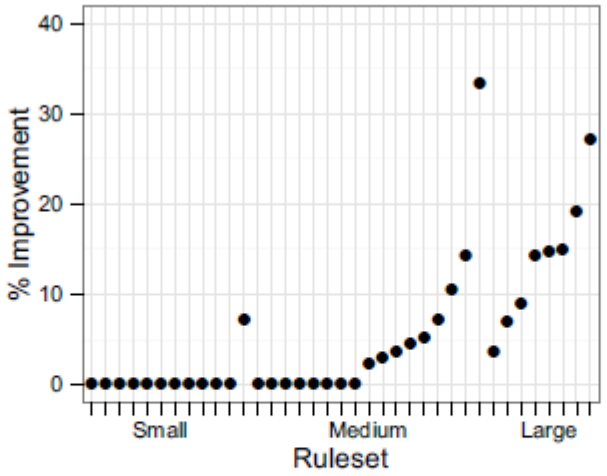
| | Algorithm | All | Large | Med. | Small |
|---|---|---|---|---|---|
| Orig. | TUF Trie | 26.2% | 30.8 | 41.8 | 0.8 |
| | TUF ACL | 22.8 | 22.8 | 38.4 | 0.7 |
| Uniq. | TUF Trie | 45.3 | 56.9 | 70.2 | 2.4 |
| | TUF ACL | 43.3 | 50.9 | 68.8 | 2.1 |

Table 3: ACR on real world classifiers

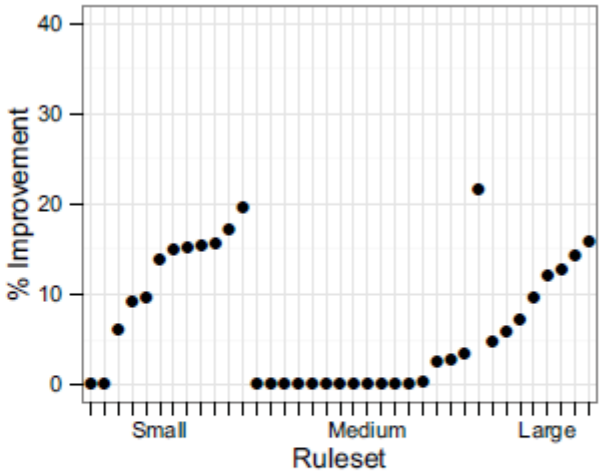
# V. Experimental Results (Cont.)




(a) Prefix Compression




(b) Ternary Compression




(c) Ternary Compression, Unique decisions

Figure 10: Improvement of TUF over the state of the art for real life classifiers

(a) TCAM Razor v.s. TCAM Razor with TUF Trie
(b) Ternary Razor and BitWeaving v.s. Ternary Razor and BitWeaving with TUF ACL

For small classifiers, TUF-based Razor can be more than 1000 times faster than the original TCAM razor.

For larger classifiers, the speed difference is an average of twenty times faster, achieving the exact same level of compression in much less time.