Approximate Classifiers with Controlled Accuracy

Vitalii Demianiuk*†, Kirill Kogan*, Sergey I. Nikolenko†,‡
*IMDEA Networks Institute, Madrid
†Steklov Institute of Mathematics at St. Petersburg
‡NRU Higher School of Economics, St. Petersburg

***Abstract*—Performing exact computations can require significant resources. *Approximate computing* allows to alleviate resource constraints, sacrificing the accuracy of results. In this work, we consider a generalization of the classical *packet classification problem*. Our major contribution is to introduce various representations for *approximate packet classifiers* with controlled accuracy and optimization techniques to reduce classifier sizes exploiting this new level of flexibility. We validate our theoretical results with a comprehensive evaluation study.**

I. Introduction

Exact computations may require excessive resources. *Approximate computing* deals with potentially inaccurate computations helping alleviate resource constraints [1]. In this paper we generalize a classical *packet classification problem* (the exact case), where each rule is composed from a ternary bit string and an action; since rules can overlap they have priorities and the first *matched* rule is returned for an incoming packet. There is a long line of research exploring various optimization methods to find *semantically equivalent* packet classifiers, where each header matches the same action in an originally given and optimized classifiers [2], [3], [4]. In this work we consider the case when semantically equivalent classifiers fail to achieve desired optimization results and introduce approximate representations of packet classifiers allowing to "multiplex" multiple actions. This additional level of flexibility allows to improve resource requirements by still keeping desired level of accuracy.

Majority of proprietary heuristics optimizing packet classifiers can be reduced to well-known operators minimizing the size of Boolean expressions [5], [6]. Unfortunately, using these operators for the approximate case can lead to exponential number of the considered variants. To avoid this complexity, we generalize the basic operators (used in the exact case) to the approximate case and study the properties of different operation sequences that significantly simplify optimization process and achieve better optimization results. In the case of prefix classifiers with LPM priorities, we show how to generalize the algorithm eLP [7], [8] that constructs a classifier with minimal number of entries to the approximate case. As a byproduct, we improve its time complexity by a factor of a, where a is the number of different actions in an input classifier.

II. Motivating Examples

We begin with two motivating examples that introduce the idea of approximate representations. First, consider a common situation when the rules of a classifier map headers to a quantitative characteristic, e.g., desired latency, which is further aggregated into service classes which the packets are classified into. E.g., on Fig. 1a values 1-3 are mapped to the *Gold* class; 4-6, to *Silver*. The classifier in Fig. 1a is irreducible with respect to *semantic equivalence*: no smaller classifier maps exactly the same headers to the same classes.

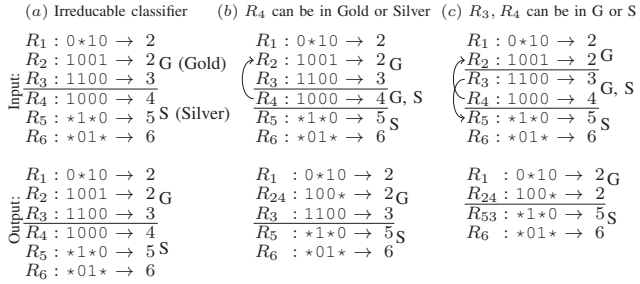


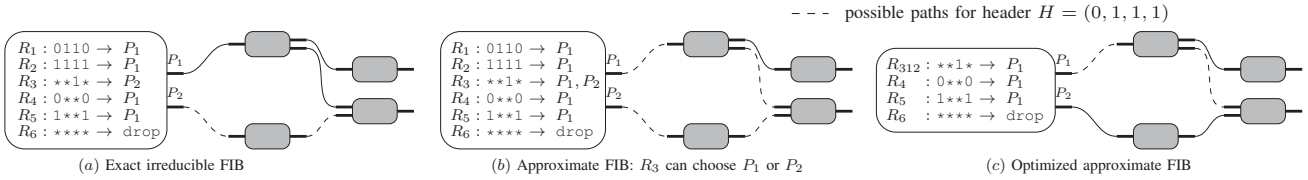
Fig. 1. Tradeoff between accuracy and required memory: (a) exact case; (b-c) approximate case.

However, let us consider "borderline" rules between service classes. Suppose that we can allow packets falling under R_4, which currently maps to *Silver* but has value 4, very close to the lower bound of the *G*old class, to be associated with either *Gold* or *Silver* classes. In this case we can perform further optimizations, associating R_4 with *Gold* or *Silver* to better reduce the classifier; Fig. 1b shows how R_4 can be merged with R_2 and replaced by R_{24} that maps to *Gold*.

We can take the approximation one step forward. Since R_3 has value 3, which is very close to the upper bound of the *Silver* class, we can allow R_3 to associate with *Silver* (Fig. 1c), and classifier optimization can now exploit this additional flexibility, merging R_3 with R_5 to get R_{53}. In general, Fig. 1 illustrates the fundamental tradeoff between accuracy and efficiency in approximate classifier representations.

Note that even in this specific example, not only borderline rules between different classes may be extended with additional actions. Although R_6 maps to 6, very far from the lower bound of the *Gold* class, it might still be allowed to extend R_6 with the *Gold* option due to some additional considerations. For instance, if we know *a priori* that required bandwidth for the traffic classified by R_6 is relatively small compared to the total volume of *Gold* traffic, we might be able to afford this approximation anyway.

The second motivating example is of a different nature, showing that approximate classifier representations are a much more general tool. The scalability problem of forwarding tables (FIBs) is a largely unsolved problem to date [9]. One can run a third-party process that estimates the "quality" of different paths for the header space covered by a given traffic matrix. It turns out that by exploiting approximate



(a) Exact irreducible FIB (b) Approximate FIB: R_3 can choose P_1 or P_2 (c) Optimized approximate FIB

Fig. 2. Approximate computing in routing.

representations and extending the actions of already computed FIBs with acceptable alternatives (based on the "quality" of estimated paths), we can often reduce the classifier size.

The exact classifier in Fig. 2a is irreducible and consists of 6 entries. For instance, the header $H = (0, 1, 1, 1)$ is classified by R_3 and follows path of length 3. If we allow R_3 to redirect traffic through the port P_1, the optimization process will be able to reduce two entries. Now headers classified by R_3 will be transmitted through the port P_1, and other headers will not be affected. In particular, the header H in Fig. 2c now follows another path with the same length.

III. Model Description

We begin with the basic notions. A packet *header* $H = (h_1, \ldots, h_w)$ is a sequence of bits $h_i \in H$, $h_i \in \{0, 1\}$, $1 \leq i \leq w$; e.g., $(1\ 0\ 1\ 0)$ is a 4-bit header. A filter $F_i = (f_1^i, \ldots, f_w^i)$ is a sequence of w values corresponding to bits in the headers, but the possible values are now 0, 1, or $*$ ("don't care"). A *rule* $R_i = (F_i, \{\mathcal{A}_i\})$ consists of a *filter* F_i and a pointer to the corresponding *action set* $\mathcal{A}_i$. A header H is *matched* by a filter F (or by a rule $R = (F, \mathcal{A})$) if for every bit of H the corresponding bit of F has either the same value or $*$.

A *classifier* $\mathcal{K} = \{R_1, \ldots, R_N\}_{\prec}$ is an ordered (by $\prec$) set of rules. If multiple rules match a header, the rule R with highest priority is preferred, and we say that H is *classified* by R (or its filter). The classifier finds the classifying action for a given header. We assume that every classifier contains a "catch-all" rule $R_\perp$ at the end, matching all headers unmatched by other rules; $R_\perp$ has a unique action D that no other rule has.

Example 1. In the following classifier with $w = 4$, $H = (0\ 1\ 1\ 0)$ matches R_1, R_4, $R_\perp$, so H is classified by R_1.

$\mathcal{K}$	#1	#2	#3	#4	Exact	Approx
R_1	0	1	1	0	A_1	A_1, A_4
R_2	1	$*$	0	$*$	A_2	A_2
R_3	1	0	1	$*$	A_3	A_3
R_4	0	$*$	1	$*$	A_4	A_4
R_5	$*$	0	0	$*$	A_5	A_5
$R_\perp$	$*$	$*$	$*$	$*$	D	D

We say that a rule R_i *covers* a rule R_j if all headers that match R_j also match R_i. $R_\perp$ always covers all other rules. Two rules R_i, R_j *intersect* ($R_i \cap R_j \neq \emptyset$) if there exists a header matched by both rules. In Example 1, R_4 covers R_1, while R_2 and R_5 intersect.

The notion of *exact* and *approximate* classifiers is central for this work. We say that a classifier $\mathcal{K}$ is *exact* if for each rule $R_i \in \mathcal{K}$ $\mathcal{A}_i$ consists of a single element; otherwise, $\mathcal{K}$ is *approximate*. Exact classifiers are a traditional way to represent packet classifiers. Approximate classifiers are a generalization where $\mathcal{A}_i$ represents multiple options that can be taken for the matched rule. In Example 1, the filters of the classifier combined with the action sets in Exact and Approx columns form the exact $\mathcal{K}_e$ and approximate $\mathcal{K}_a$ classifiers, respectively. E.g., if classifiers represent forwarding tables, action set $\mathcal{A}_1 = \{A_1, A_4\}$ means that a packet can be transmitted through either A_1 or A_4. Note that in this case $\mathcal{A}_1 = \{A_1, A_4\}$ does not represent possibilities for load balancing but adds options for an *optimization process* whose output is always an exact classifier. We say that a classifier $\mathcal{K}'$ is *approximately equivalent* to a classifier $\mathcal{K}$ if for every header H, the classifying action set $\mathcal{A}' \in \mathcal{K}'$ is a subset of the classifying action set $\mathcal{A} \in \mathcal{K}$. In Example 1, the classifier $\mathcal{K}_e$ is approximately equivalent to $\mathcal{K}_a$.

IV. Basic Operators

In this section, we show basic operators reducing the number of entries in exact and approximate classifiers.

A. Classifiers as Boolean Expressions

Optimization methods that minimize the required space for classifiers [2], [3], [4] usually can be reduced to methods minimizing size of Boolean expressions. There are two main cases here. If no pair of rules intersect then the classifier represents a Boolean formula in DNF, with every filter as a clause. The *MinDNF* problem (minimizing DNF size) has been extensively studied in complexity theory. In theory, it is $\mathrm{NP}^{\mathrm{NP}}$-complete ($\Sigma_2^P$-complete), with some inapproximability results as well [5], [6]. In practice, classical heuristics for *MinDNF* are based on Karnaugh maps.

In the order-dependent case, rule priorities can be encoded with circuits of depth 3 with OR–AND–OR alternation, i.e., AC^0 circuits of depth 3. However, even in the order-independent case there is no tractable algorithm minimizing Boolean expressions, so we cannot hope for polynomial optimal algorithms minimizing size of classifiers. Existing heuristics include a set of basic operators applied to a classifier while possible. They consist of two major blocks: *basic operators* and an *optimization process* that constructs a sequence of basic operations to reduce the classifier.

B. Generalized basic operators

Usually, three basic operators are considered in the scope of packet classifiers [4] since they have the right balance between operational complexity and applicability. These basic operators decide when a rule can be removed or replaced by another rule. Here, we further generalize them for the approximate case.

If action sets consist of a single action, they can be directly applied to exact classifiers.

Example 2. Through this section we will use the following classifier as a running example. Actions in the Exact column correspond to the exact classifier $\mathcal{K}_e$; the Approx column corresponds to the approximate classifier $\mathcal{K}_a$:

$\mathcal{K}$	#1	#2	#3	#4	#5	Exact	Approx
R_1	0	1	1	$*$	0	A_1	A_1
R_2	$*$	1	$*$	0	$*$	A_1	A_1, A_2
R_3	1	1	$*$	1	1	A_2	A_1, A_2
R_4	$*$	1	1	$*$	$*$	A_1	A_1, A_2
R_5	0	1	$*$	1	1	A_2	A_1, A_2
R_6	$*$	0	$*$	1	1	A_2	A_2
R_7	0	0	0	1	1	A_3	A_3
$R_\perp$	$*$	$*$	$*$	$*$	$*$	D	$\{D\}$

Forward Subsumption $\mathcal{F}(R_i)$. This operator removes all unreachable rules and does not depend on the structure of action sets. Formally, a rule $R_i \in \mathcal{K}$ can be removed if there exists a rule $R_j \prec R_i$ that covers R_i. Since forward subsumptions do not depend on actions, the resulting classifier is approximately equivalent to the original. In Example 2, R_7 can be removed since R_6 covers it.

Backward Subsumption $\mathcal{F}(R_i)$. Sometimes a rule can still be removed if it is covered by a rule with lower priority. Formally, $R_i \in \mathcal{K}$ can be removed if the following conditions hold:

1) there is a rule $R_j \succ R_i$ that covers R_i such that their action sets $\mathcal{A}_i$ and $\mathcal{A}_j$ *intersect*;
2) for every rule $R_t \in \mathcal{K}$ such that $R_i \prec R_t \prec R_j$ and $R_t \cap R_i \neq \emptyset$, the *intersection* of $\mathcal{A}_t$ and $\mathcal{A}_i$ is also nonempty; after applying backward subsumption, we set $\mathcal{A}_t := \mathcal{A}_t \cap \mathcal{A}_i$ for each such rule.

If several R_j are possible, we choose R_j with the largest priority.

In Example 2 (approximate case), R_1 can be removed by backward subsumption since it is covered by R_4, and even though R_1 intersects with R_2 they have intersecting action sets. Note that the header (0 1 1 0 0) that was previously classified by R_1 will now be classified by R_2, so we replace $\mathcal{A}_2, \mathcal{A}_4$ by the sets $\mathcal{A}_2 \cap \mathcal{A}_1 = \{A_1\}$ and $\mathcal{A}_4 \cap \mathcal{A}_1 = \{A_1\}$. The exact case is very similar but now action sets of rules R_1, R_2, R_4 are equal, therefore, this operation is allowed.

Resolution $R'_i = \mathcal{R}(R_i, R_j)$. This operator comes from propositional proof theory: in propositional logic, expressions $(x \wedge C) \vee (\bar{x} \wedge C)$ and $(x \vee C) \wedge (\bar{x} \vee C)$ are both obviously equivalent to C and can be simplified to C.

Two rules $R_i \prec R_j$ can be combined and replaced by a new rule R'_i (in place of R_i in the list) if:

1) filters F_i and F_j coincide in all bit indices except k, and the k-th bit is not $*$ in F_i or F_j (when it is, subsumption applies);
2) the set $\mathcal{A}'_i = \mathcal{A}_i \cap \mathcal{A}_j \cap \bigcap_{t: R_i \prec R_t \prec R_j \text{ and } R_t \cap R_j \neq \emptyset} \mathcal{A}_t$ (intersection of action sets over all intersecting rules) is nonempty; after applying resolution we set $\mathcal{A}_i := \mathcal{A}'_i$.

In Example 2, F_3 and F_5 differ only in the first bit. In the approximate classifier, we can apply $R'_3 := \mathcal{R}(R_3, R_5)$ since action sets of R_3, R_4, R_5 are all equal to $\{A_1, A_2\}$; in the exact case, however, we would not be able to apply $\mathcal{R}(R_3, R_5)$ since actions for R_4, R_5 differ.

Both subsumption operators are unary operators (removing one rule), and resolution is a binary operator (replacing two rules with one). Applying the introduced operators to exact and approximate classifiers in Example 2 leads to classifiers with six and four rules, respectively.

C. Why to generalize

In general, we can choose a single action for every rule in an approximate classifier, and each such combination is an exact classifier. In this way we could exploit already existing heuristics applicable to previously studied basic operators in the exact case. The problem is not only the exponential number of different exact classifiers but also that generalized versions of the basic operators extend their applicability.

Example 3. We show two approximate classifiers, where generalized operators can achieve better results than even brute-force search over all exact classifiers.

$\mathcal{K}$	#1	#2	#3	#4	$\mathcal{A}$
R_1	0	1	0	$*$	A_1, A_2
R_2	0	$*$	$*$	0	A_1
R_3	$*$	$*$	0	$*$	A_2
R_4	0	1	$*$	$*$	A_1, A_2
$R_\perp$	$*$	$*$	$*$	$*$	D

Applying approximate operators in this classifier, R_1 can be removed by backward subsumption: R_4 covers R_1, and all necessary conditions hold. On the other hand, in neither of the four possible exact specializations of $\mathcal{K}$ we can reduce R_1 or any other rule.

Similarly, for the following classifier two resolution operations can be applied, $\mathcal{R}(R_1, R_5)$ and $\mathcal{R}(R_2, R_4)$, but in any exact case only one of them remains:

$\mathcal{K}$	#1	#2	#3	#4	$\mathcal{A}$
R_1	0	0	$*$	1	A_1
R_2	1	1	$*$	0	A_2
R_3	$*$	$*$	0	$*$	A_1, A_2
R_4	1	0	$*$	0	A_2
R_5	0	1	$*$	1	A_1
$R_\perp$	$*$	$*$	$*$	$*$	D

V. Properties of Heuristics

An *optimization process* consists of a set of *basic operators* (that we have introduced above) and a *heuristic* that chooses a sequence of applied operations (*operation sequence*). In this section we study properties of operation sequences to improve optimization results and reduce the time complexity of considered heuristics.

By definition, each applied operation removes exactly one rule, so after an operation sequence $\mathcal{S}$ on a classifier $\mathcal{K}$ the optimized classifier has $|\mathcal{K}| - |\mathcal{S}|$ rules. Therefore, considered heuristics can shift from minimizing number of rules to maximizing length of a constructed operation sequence. A sequence of applied operations is *optimal* if its length is maximal among all other sequences. Similarly, an operation sequence $\mathcal{S}$ is *extensible*, if $\mathcal{S}$ can be appended with additional

operations; otherwise, $\mathcal{S}$ is *inextensible*. An optimal sequence is always inextensible, but there are suboptimal inextensible.

Example 4. The following example shows that different operation sequences even in the exact case can lead to different results.

P	#1	#2	#3	#4	#5	*Action*
R_1	1	1	1	$*$	$*$	A_1
R_2	1	0	$*$	$*$	0	A_1
R_3	1	1	0	$*$	0	A_1
R_4	1	1	1	$*$	0	A_1
$R_\perp$	$*$	$*$	$*$	$*$	$*$	D

Here, we can apply either forward subsumption $\mathcal{F}(R_4)$ or resolution $R = \mathcal{R}(R_3, R_4)$ followed by another resolution $\mathcal{R}(R_2, R)$. Both operation sequences are inextensible, but only one of them is optimal, and forward subsumption cannot lead to new operations.

Example 4 shows that a combinatorial search for optimal operation sequences is a hard problem even in the exact case. Therefore, in what follows we will study a specific fixed set of operations applicable to the original classifier. If we can find the longest operation sequence among a reasonably wide class of such sets, it will provide us with a lower bound on the length of the optimal operation sequence.

We denote by $\mathbf{O}$ a set of operations applicable to the initial classifier that does not contain *conflicting* operations using the same rule; that is, if there are multiple operations applicable to the same rule we choose one of them. Note that there can be many different $\mathbf{O}$ for the same classifier $\mathcal{K}$; at this point we fix $\mathcal{K}$ and one specific $\mathbf{O}$. Operations in $\mathbf{O}$ implicitly depend on each other; for example, a resolution operation can create a new possible rule that will intersect with another rule in backward subsumption. We make one additional assumption: if for some rule R_i both $\mathcal{F}(R_i)$ and $\mathcal{B}(R_i)$ are applicable in the original classifier, we assume that $\mathbf{O}$ contains $\mathcal{F}(R_i)$ since it makes no sense to apply backward instead of forward subsumption.

A. Operation sequences in the exact case

The following theorem shows that in the exact case, we can apply all operations in $\mathbf{O}$ regardless of these implicit dependencies. Later we will see that in the approximate case maximal sequences on $\mathbf{O}$ have a more complex structure. Despite the popularity of Boolean minimization in the exact case, to the best of our knowledge we know of no prior work where this result was proven.

Theorem 1. *If all action sets in $\mathcal{K}$ contain exactly 1 action (the exact case), there is an applicable sequence consisting of all operations from $\mathbf{O}$.*

Proof. First we apply all forward subsumptions, then all backward subsumptions. We can apply all forward subsumptions because for any applicable $\mathcal{F}(R_i)$ classifier will contain a rule $R_j \prec R_i$ that covers R_i regardless of already applied forward subsumptions. Also, forward subsumptions cannot change the covering rule R_j for a backward subsumption $\mathcal{B}(R_i)$ since otherwise either R_i were subject to forward subsumption in the original classifier or the rule that covers R_j is covering in $\mathcal{B}(R_i)$. Therefore, after forward subsumptions we can apply $\mathcal{B}(R_i)$ in the order of priority of R_i since applied $\mathcal{B}(R_i)$ does not affect backward subsumptions further down the list. Subsumptions only remove rules that are not used by resolutions in $\mathbf{O}$; therefore, we can apply all resolutions after all subsumptions. We apply $\mathcal{R}(R_i, R_j)$ in the order of priority on R_i. Again, no conflicts arise in this order since $\mathcal{R}(R_i, R_j)$ can only remove a rule between R'_i and R'_j for some further applicable resolution $\mathcal{R}(R'_i, R'_j)$. □

The optimal sequence of operations does not always apply all operations from $\mathbf{O}$: some rules can find better use in new operations that were not possible in $\mathbf{O}$. But there is a lower bound on the length of the optimal sequence.

Corollary 1. *The length of an optimal operation sequence in the exact case is at least the maximal length of all possible $\mathbf{O}$; moreover, this lower bound can be computed in time $O(N^2 \cdot w + C_{\mathcal{R}}^{1.5})$, where $C_{\mathcal{R}}$ is the number of resolutions applicable in the original classifier.*

Proof. To construct a maximal $\mathbf{O}$, first we take all subsumptions on different rules and then choose a maximal number of non-conflicting resolutions. This is an optimal algorithm since we look only on operations applicable to the original classifier, and it makes to sense to apply resolution instead of subsumption if both apply. We ignore all resolutions that conflict with a chosen subsumption rule. On the other applicable resolutions, we construct a bipartite graph $G = (V_0, V_1, E)$: the set of vertices V_0 contains rules whose filters have even number of bits with value 1, V_1 consists of rules with odd number of such bits, and edges consist of possible resolutions, i.e., each resolution connects a vertex from V_0 with a vertex from V_1 in G. Edges in a maximal matching in G form a maximal set of non-conflicting resolutions that can be added to $\mathbf{O}$. A maximal matching can be found in time $O(N + C_{\mathcal{R}} \cdot min(N, C_{\mathcal{R}})^{0.5})$. □

So far, we have seen that while finding the best possible operation sequence is always hard, in the exact case it is relatively easy to construct a maximal set of non-conflicting operations $\mathbf{O}$ that can be applied to the original classifier. Next we turn to the approximate case.

B. Operation sequences in the approximate case

In the approximate case, $\mathbf{O}$ can increase compared to the exact case since generalized operations are applicable more often (recall Example 3). But unlike the exact case, Theorem 1 does not hold for the approximate case, and an operation sequence of length $|\mathbf{O}|$ is not guaranteed to exist.

Example 5. Operations from $\mathbf{O}$ are not always applicable together.

$\mathcal{K}$	#1	#2	#3	#4	$\mathcal{A}$
R_1	0	1	1	$*$	A_1
R_2	1	1	$*$	1	A_1, A_2
R_3	$*$	1	1	$*$	A_1, A_2
R_4	0	1	$*$	1	A_2
$R_\perp$	$*$	$*$	$*$	$*$	D

For instance here, we can apply resolution $\mathcal{R}(R_2, R_4)$ or backward subsumption $\mathcal{B}(R_1)$, but we cannot apply both since otherwise $\mathcal{A}_2$ would become empty.

The complexity of constructing the longest operation sequence on $\mathbf{O}$ arises from backward subsumptions.

Theorem 2. *A maximal sequence of applicable operations from* $\mathbf{O}$ *can be found in polynomial time if* $\mathbf{O}$ *consists of forward subsumptions and resolutions.*

Proof. Apply all operations from $\mathbf{O}$ as follows: first all forward subsumptions, then all resolutions applied in the order of decreasing priority of the top rule. □

But if $\mathbf{O}$ has backward subsumptions then finding the longest sequence turns into an intractable problem.

Theorem 3. *Finding a maximal sequence of applicable backward subsumptions from* $\mathbf{O}$ *is an NP-complete problem; moreover, it is not approximable up to a constant factor in polynomial time.*

Proof. We reduce the Maximal Independent Set problem: by a graph $G = (V, E)$ we construct a classifier where every possible sequence of applied backward subsumptions will correspond to an independent set of G. We denote $n = |V|$ and $m = |E|$. The classifier will consist of $2n + m$ rules. The first n rules $R_1, \ldots, R_n$ and last n rules $R_{n+m+1}, \ldots, R_{2n+m}$ correspond to vertices, and the m rules in the middle correspond to edges. A rule R_i for $1 \leq i \leq n$ is covered by a rule R_{n+m+i} and additionally it intersects exactly those rules that correspond to the edges adjacent to v. For all rules R_i that correspond to vertices their $\mathcal{A}_i$ will contain a special action A_0. For each edge $e = (u, v)$, we create two actions $A_{e,u}$ and $A_{e,v}$: action sets for rules corresponding to u contain $A_{e,u}$; to v, $A_{e,v}$; to e, both. In this construction, for any edge $e = (u, v)$ backward subsumptions on the rules corresponding to u and v are allowed separately but not together, and this implies that the set of applied backward subsumptions corresponds to an independent set in G, and finding the largest such set is equivalent to finding the largest independent set. □

C. How to extend operation sequences

So far we have studied properties of operation sequences constructed on a set $\mathbf{O}$ of initially non-conflicting operations for both exact and approximate cases. In this part, we study the properties of optimal operation sequences in the approximate case. The following structural property of operation sequences allows us to reduce the space of operation sequences considered to find the longest (optimal) one.

Lemma 1. *For an operation sequence* $\mathcal{S}$*, there exists another operation sequence* $\mathcal{S}'$ *that satisfies the following:*

1) $\mathcal{S}'$ *consist of the same operations, but a backward subsumption* $\mathcal{B}(R)$ *can be changed to a forward subsumption* $\mathcal{F}(R)$*;*
2) *resolutions and forward subsumptions are applied before backward subsumptions;*
3) *backward subsumptions* $\mathcal{B}(R)$ *are applied in the reverse order of the priorities of* R*;*
4) *for each rule* R_i*, its final action set* $\mathcal{A}_i$ *obtained after* S *is applied is a subset of the action set* $\mathcal{A}'_i$ *after* S' *is applied.*

Due to the space limits, the proof of Lemma 1 is omitted. Lemma 1 immediately implies the following corollary.

Corollary 2. *There exists an optimal operation sequence where all resolutions and forward subsumptions are applied before backward subsumptions, and all backward subsumptions are applied in the increasing order of priorities of their rules.*

The impact of this transformation is that after changing an operation sequence $\mathcal{S}$ to such an $\mathcal{S}'$, inextensible sequences may become extensible, as the following example shows.

Example 6. Suppose that in Example 2, we apply $\mathcal{F}(R_7)$ followed by $\mathcal{B}(R_1)$ and then $R'_3 = \mathcal{R}(R_3, R_5)$, getting the following classifier:

$\mathcal{K}$	#1	#2	#3	#4	#5	$\mathcal{A}$
R_2	$*$	1	$*$	0	$*$	$\{A_1\}$
R'_3	$*$	1	$*$	1	1	$\{A_1\}$
R_4	$*$	1	1	$*$	$*$	$\{A_1\}$
R_6	$*$	0	$*$	1	1	$\{A_2\}$
$R_\perp$	$*$	$*$	$*$	$*$	$*$	$\{D\}$

But if we applied $R'_3 = \mathcal{R}(R_3, R_5)$ before $\mathcal{B}(R_1)$, we would have $\mathcal{A}'_3 = \{A_1, A_2\}$. Therefore, we can apply $\mathcal{R}(R'_3, R_6)$ and get the following classifier:

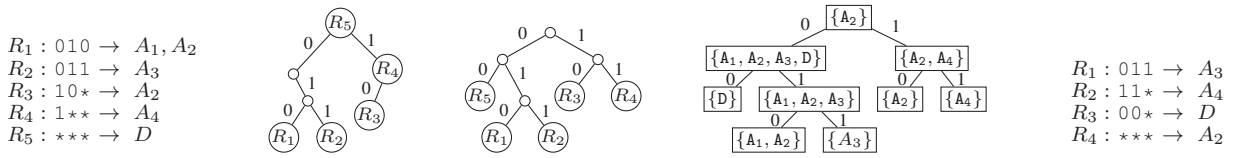
$\mathcal{K}$	#1	#2	#3	#4	#5	$\mathcal{A}$
R_2	$*$	1	$*$	0	$*$	$\{A_1\}$
R''_3	$*$	$*$	$*$	1	1	$\{A_2\}$
R_4	$*$	1	1	$*$	$*$	$\{A_1\}$
$R_\perp$	$*$	$*$	$*$	$*$	$*$	$\{D\}$

This observation allows us to improve optimization results: if we have obtained an inextensible operation sequence $\mathcal{S}$ that does not satisfy the conditions of Lemma 1, we can transform it into a new sequence $\mathcal{S}'$, as explained in the constructive proof of Lemma 1, and $\mathcal{S}'$ may become extensible.

In this section, we have seen algorithms that try to construct an optimal operation sequence from a set of applicable operations $\mathbf{O}$. They are guaranteed to find a maximal sequence only in special cases, and it is a hard computational problem in general. Still, in many practical situations algorithms presented here work well, and we have also introduced a heuristic for further optimization that reorders operations.

VI. Approximate LPM Classifiers

So far we considered a general case where each filter is a ternary bit string. Since minimizing a given classifier is an intractable problem, we defined our optimization process as constructing a maximal sequence over a predefined set of operators. There is, however, an important special case when all rules are prefixes ordered with longest-prefix-match (LPM) priorities; now the alphabet is binary, and the first $*$ is just a delimiter that defines prefix length.



(a) Original classifier *(b)* T_o trie *(c)* T_e *(d)* T_e with calculated $L(x)$ *(e)* Optimized classifier

Fig. 3. Example of running the aLP algorithm.

In the exact case, the works [7], [8] introduce the algorithm eLP that computes a minimal LPM classifier in time $O(N \cdot w \cdot a)$, where N is the number of rules in the input, w is the classification width (in bits), and a is the number of different actions. In this section, we show how to generalize eLP to the approximate case while preserving optimality; we call the generalized version aLP. As a side effect, we improve the time complexity of eLP to $O(N \cdot w)$.

A. The eLP Algorithm

We begin by briefly explaining the basic intuition behind eLP [7], [8]. First, the filters of all rules from the input classifier $\mathcal{K}$ are organized in a binary prefix trie T_o (Fig 3a-b). Note that prefixes of rules in $\mathcal{K}$ do not necessarily end in leaves, and not every internal node in T_o has two sons. To avoid prefixes in internal nodes of T_o, eLP pushes them to leaves, extending the trie to T_e as follows: each node with a single child is complemented with another one; for each added leaf x its corresponding rule inherits the action from x's closest ancestor with rule (see Fig 3c). As a result, in T_e only leaves correspond to rules. We denote by $T_o(x)$ a subtree of T_o rooted in x; by $T_e(x)$, the corresponding subtree of the extended trie. Both $T_o(x)$ and $T_e(x)$ represent equivalent classifiers. $T_e(x)$ can have more corresponding rules than $T_o(x)$ but one can show that optimal classifiers for $T_e(x)$ and $T_o(x)$ are the same.

We denote by R_x a rule that will be created for a node $x \in T_e$ with an action A_x. The algorithm eLP is based on the following process. Let l and r be the left and right children of a node $x \in T_e$. Consider already computed optimal classifiers $\mathcal{K}_l$ and $\mathcal{K}_r$ corresponding to $T_e(l), T_e(r)$ respectively. Now:

- $\mathcal{K}_x = \mathcal{K}_l \cup \mathcal{K}_r \cup \{R_x\}$;
- if $A_x = A_l$, remove R_l from $\mathcal{K}_x$;
- if $A_x = A_r$, remove R_r from $\mathcal{K}_x$.

It can be shown that the resulting classifier $\mathcal{K}_x$ is optimal.

Denoting by $f(x, A_x)$ the size of the optimal classifier $\mathcal{K}_x$, we get the following recurrent expression:

$$f(x, A_x) = \min_{A_l, A_r} f(l, A_l) + f(r, A_r) + 1 - [A_l = A_x] - [A_r = A_x]$$

To implement this process, eLP maintains in each node $x \in T_e$ an explicitly computed list $L(x)$ of actions A that minimizes $f(x, A)$. The list $L(x)$ can be calculated from $L(x_l)$ and $L(x_r)$: if $L(x_l) \cap L(x_r) \neq \varnothing$ then $L(x) = L(x_l) \cap L(x_r)$ (in this case, the corresponding rules in classifiers for x_l and x_r have the same action, and this action will be chosen for the last rule of a classifier at node x), otherwise $L(x) = L(x_l) \cup L(x_r)$ (in this case the rules for x_l, x_r have different actions, and one of them will be chosen for x).

Note that originally, the work [7] only defined the eLP algorithm; subsequent complexity analysis in [8] showed that it runs in time $O(N \cdot w \cdot a)$. In the following, we will show how to generalize eLP to the approximate case and remove the a factor from the complexity bounds for the exact case.

B. Optimality in the approximate case

Let us generalize eLP to the approximate case; we call the new algorithm aLP. Suppose that all $\mathcal{A}_i$ are given as lists of actions, and actions appear in these lists in the same order. The above process for deciding which rules to add or remove remains unchanged. In the approximate case, the recurrence relation for $f(x, A)$ still equals the size of the optimal classifier $\mathcal{K}_x$. Now actions can be recovered up to sets $\mathcal{A}_i$s. This modification does not change $f(x, A)$ and does not affect reconstruction rules for $L(x)$ in internal nodes of T_e. Only the definition of $L(x)$ for leaf nodes changes: $L(x) = \mathcal{A}_{i_x}$, where R_{i_x} is the rule corresponding to a leaf x in T_e. Figure 3 illustrates the aLP algorithm.

Theorem 4. *For the approximate case, time complexity of aLP is $O(N \cdot w \cdot l_{\max})$, where $l_{\max} = \max_i |\mathcal{A}_i|$.*

Proof. We denote by $V(T)$ the set of nodes in a trie (or subtree) $T \subseteq T_e$. The total time complexity of the proposed method is $O(\sum_{x \in V(T_e)} |L(x)|)$ since union and intersection of sets can be done in linear time. Summing $|L(x)|$ over all leaves, we get $O(N \cdot w \cdot l_{\max})$ since T_e has $O(N \cdot w)$ leaves.

Each internal node x of T_e also occurs in T_o. For rules corresponding to a node in $T_o(x)$, all action sets $\mathcal{A}_i$ can participate in the construction of $L(x)$; among other rules, only one rule can participate in $L(x)$ – the one that ends in the nearest ancestor of x. Therefore, $|L(x)| \leq l_{\max} \cdot (s(x)+1)$, where $s(x)$ is number of nodes in $T_o(x)$ where some rules end.

Each node in T_o corresponding to a rule increments $s(x)$ for at most w internal nodes, namely its ancestors. Therefore, $\sum_{x \in T_o} s(x) \leq N \cdot w$. This immediately implies that the sum of $|L(x)|$ over all internal nodes is $O(N \cdot w \cdot l_{\max})$. □

Since approximate case degrades to the exact case when all $|\mathcal{A}_i| = 1$, by Theorem 4 we have improved the time complexity of eLP compared to [8, p. 294] by a factor of a, the number of different actions in the original classifier.

VII. Evaluation

In this section we evaluate the efficiency of proposed approaches on LPM and general approximate classifiers. Optimization results depend not only on the quality of optimization

methods but also on classifier structure; we validate the fundamental tradeoff between memory requirements and desired level of accuracy.

A. LPM classifiers

Heuristics. We compare two algorithms for minimizing approximate LPM classifiers: aLP from Section VI and aBM for the general case (ternary bit strings with general priorities) which applies the basic operators we introduced. In the general case, by Lemma 1 the heuristic applies forward subsumptions and resolutions while possible, followed by backward subsumptions in reverse priority order.

Methodology. We have evaluated our approach on five IPv4 FIB classifiers; Table I shows their most important characteristics. We have made the source code of our simulations available at [10]; unfortunately, due to NDA restrictions we cannot make the classifiers available online. To make an approximate classifier, we extend actions of an exact instance with alternative options (paths), with actions more common in the exact version appearing more frequently in the approximate instance as well.

Each approximate instance is characterized by two parameters: p_{A}, the share of rules with alternative paths, and l_A, the number of alternative actions for each rule. For each rule in the selected share p_{A}, the action set $\mathcal{A}_i$ is extended with randomly selected alternatives until the size of $\mathcal{A}_i$ reaches $l_A + 1$. We choose each next action for $\mathcal{A}_i$ with a probability proportional to its number of occurrences in the exact classifier (except for actions already in $\mathcal{A}_i$).

We denote by $\mathcal{K}'_{\mathrm{aLP}}$ and $\mathcal{K}'_{\mathrm{aBM}}$ the optimized classifiers obtained by running aLP and aBM, respectively, on the original classifier $\mathcal{K}$. The third and fourth columns in Table I show the number of rules in $\mathcal{K}'_{\mathrm{aLP}}$ and $\mathcal{K}'_{\mathrm{aBM}}$ for all five FIBs. For each FIB, we have also constructed a classifier where all possible actions except the default can be chosen as alternative. This is the best case for optimization for a given set of filters; we denote by $\mathcal{K}''_{\mathrm{aLP}}$ and $\mathcal{K}''_{\mathrm{aBM}}$ classifiers optimized with the corresponding algorithms. For all five FIBs, the sizes of $\mathcal{K}''_{\mathrm{aLP}}$ and $\mathcal{K}''_{\mathrm{aBM}}$ are given in the fifth and sixth columns of Table I. During the generation of approximate classifiers, we varied the parameter p_A from 0 to 1 and l_A from 1 to 3, running aLPM and aBM on each generated classifier.

Size of optimized classifiers. Figs. 4(a,d,g) show the number of rules in optimized classifier $\mathcal{K}_{\mathrm{A}}$ as a function of p_{A} (plots with $l_A > 1$ do not show fib1 curves because the fib1 classifier has only two actions). We see that classifiers optimized by aLP are 10-20% smaller than the same classifiers optimized by aBM. The aLP algorithm wins because it is guaranteed to construct the optimal solution in the class of LPM classifiers. The main advantage of Boolean minimization techniques is their flexibility: they can be used for more general classifiers.

Savings in comparison to exact optimization. Plots on Figs. 4(b,e,h) show additional savings $W(\mathcal{K}_{\mathrm{A}})$ for the optimized approximate classifier compared to the corresponding optimized exact version, i.e., $W(\mathcal{K}_{\mathrm{A}}) = \frac{|\mathcal{K}'_{\mathrm{aLP}}|-|\mathcal{K}_{\mathrm{A}}|}{|\mathcal{K}_{\mathrm{aLP}'}|}$ if $\mathcal{K}_{\mathrm{A}}$ was optimized by the aLP algorithm and $W(\mathcal{K}_{\mathrm{A}}) = \frac{|\mathcal{K}'_{\mathrm{aBM}}|-|\mathcal{K}_{\mathrm{A}}|}{|\mathcal{K}'_{\mathrm{aBM}}|}$ in case of aBM. The plots support our theoretical results: the number of rules in optimized approximate classifiers may be significantly smaller, up to 20-50%, than in the optimized exact classifiers. Note that the slope of the curves on the plots for W is different for different datasets, which is a feature of our procedure for generating alternative actions. Classifiers where a small subset of actions covers a large share of the rules are easier to optimize because alternative actions in $\mathcal{A}$ will be likely chosen from this small subset. The largest slope is for fib1 because it has only two actions, and all rules with alternative actions will have the same pair of actions. We have chosen this generation procedure specifically to show off the effect of different action coverage.

TABLE I
FIB DATASETS

Name	$\lvert P\rvert$	$\lvert\mathcal{K}'_{\mathrm{aLP}}\rvert$	$\lvert\mathcal{K}'_{\mathrm{aBM}}\rvert$	$\lvert\mathcal{K}''_{\mathrm{aLP}}\rvert$	$\lvert\mathcal{K}''_{\mathrm{aBM}}\rvert$
fib1	137150	94232	109594	50411	70394
fib2	202671	160836	176446	49838	76718
fib3	180655	140482	154049	50384	75497
fib4	149432	107740	121974	49624	70707
fib5	180429	134580	149974	55512	81918

Comparison to the hypothetical perfect reduction. Figs. 4(c,f,i) show the savings in the optimized classifier compared to the hypothetical reduction in case when every action was allowed for every rule; in this hypothetical case, the classifier simply serves as a "match/no match" binary filter, distinguishing rules that match at least one rule and rules that are matched by $R_\perp$. Obviously, this is the case when the largest reductions are possible; we denote this fraction by $W^+(\mathcal{K}_{\mathrm{A}})$, defined as $W^+(\mathcal{K}_{\mathrm{A}}) = \frac{|\mathcal{K}'_{\mathrm{aLP}}|-|\mathcal{K}_{\mathrm{A}}|}{|\mathcal{K}'_{\mathrm{aLP}}|-|\mathcal{K}''_{\mathrm{aLP}}|}$ for the aLPM algorithm and $W^+(\mathcal{K}_{\mathrm{A}}) = \frac{|\mathcal{K}'_{\mathrm{aBM}}|-|\mathcal{K}_{\mathrm{A}}|}{|\mathcal{K}'_{\mathrm{aBM}}|-|\mathcal{K}''_{\mathrm{aBM}}|}$ for aBM algorithm. The plots show how W^+ grows with $\mathcal{K}_A$. For fib1, this curve goes from $(0, 0)$ to $(1, 1)$ since if all rules in fib1 have an alternative action then $\mathcal{A}$ for all rules will have the same pair of actions, and the optimized classifier will be the same as $\mathcal{K}''_{\mathrm{aLP}}$ or $\mathcal{K}''_{\mathrm{aBM}}$, i.e., $W^+ = 1$. A similar situation occurs with fib4 for $l_A = 2$ since 88% of its rules are covered by only two actions, so almost all rules contain these two most frequent actions. Values of W and W^+ are much smaller for fib2 since the distribution of actions in fib2 is more uniform. Note how W and W^+ grow in fib2 as l_A increases; e.g., for $p_A = 0.5$ W is 2.5 times higher for $l_A = 3$ than for $l_A = 1$.

B. General classifier experiments

Heuristics. The same aBM heuristic as in Section VII-A is used to reduce size of general ternary approximate classifiers.

Methodology. To evaluate our methods on general classifiers, we used 4 classifiers generated by ClassBench [11]. Filters of generated rules have width 112 bits. For every classifier, we have removed all rules with at least 69 bits with value $*$ since such short filters cover many more rules and if we leave them in, the resulting size will mostly depend on what common actions these general rules have with more specific rules. For each of the 4 classifiers, we have considered three

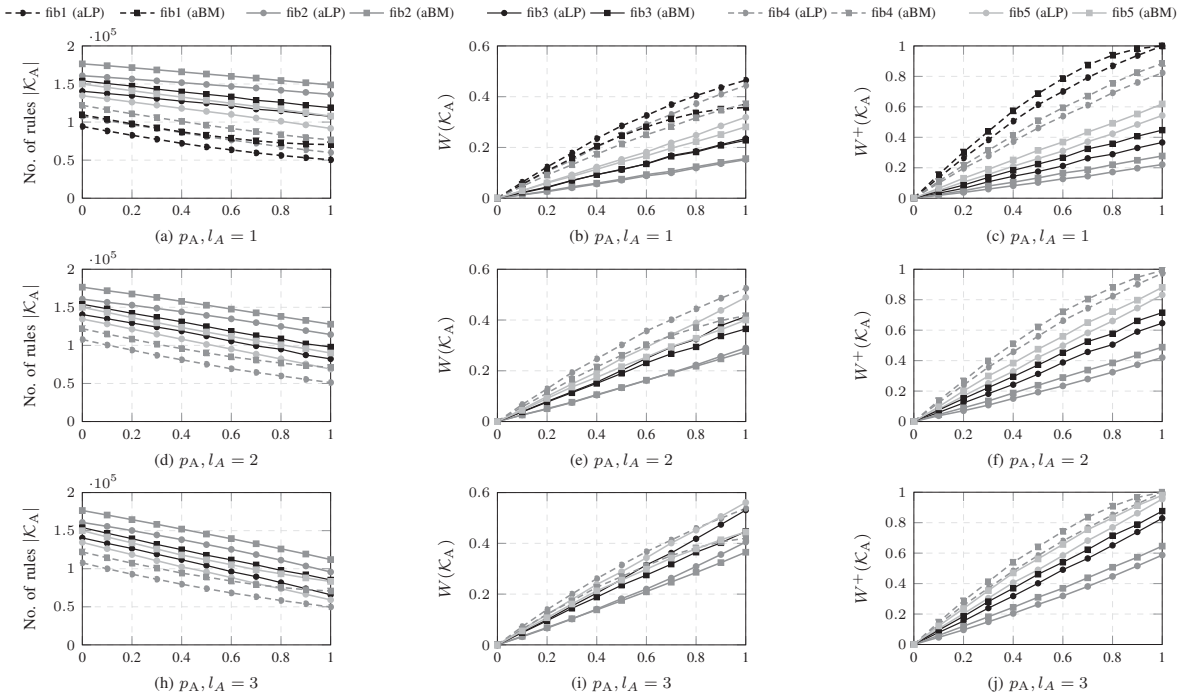


Fig. 4. FIB classifier optimization; columns show the number of rules, $W(\mathcal{K}_A)$, and $W^+(\mathcal{K}_A)$ metrics as functions of the p_A fraction of rules with: (a-c) one additional alternative action, $l_A = 1$; (d-f) $l_A = 2$; (h-j) $l_A = 3$.

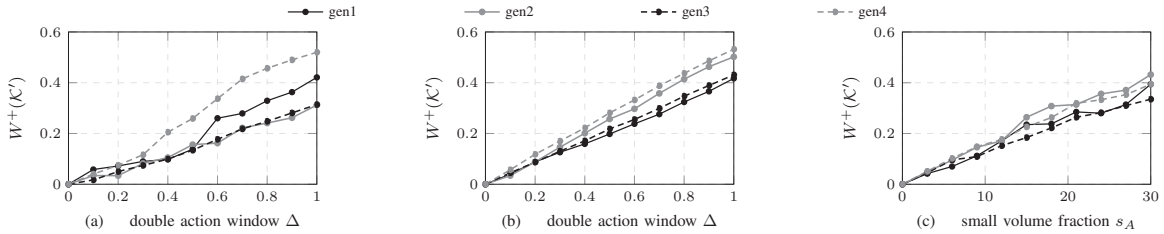


Fig. 5. General classifier optimization: $W^+(\mathcal{K}')$ as a function of (a) window size Δ for the same rule priority and action ordering; (b) window size Δ for different rule priority and action orderings; (c) fraction of "small volume" rules s_A.

kinds of instances of approximate classifiers that differ in the rules for constructing the $\mathcal{A}$ sets.

Action sets from monotone action values. The first type of approximate classifiers is constructed similarly to the motivational example from Fig. 1. We assume that the classifier defines a value of latency that decreases with the rule's priority. The first third of the rules corresponds to the Gold service class; the second third, Silver; the last third, Bronze. Action set $\mathcal{A}$ of each rule contains the corresponding action. Parameter Δ reflects the size of the window where rules have two possible actions: for a rule $R_i \in \mathcal{K}$ with $(\frac{1}{3} - \frac{\Delta}{6}) \cdot |\mathcal{K}| \leq i \leq (\frac{1}{3} - \frac{\Delta}{6}) \cdot |\mathcal{K}|$, we set $\mathcal{A}_i = \{\text{Gold}, \text{Silver}\}$; for $(\frac{2}{3} - \frac{\Delta}{6}) \cdot |\mathcal{K}| \leq i \leq (\frac{2}{3} - \frac{\Delta}{6}) \cdot |\mathcal{K}|$, $\mathcal{A}_i = \{\text{Silver}, \text{Bronze}\}$. Figure 5(a) shows the values of W^+ for the optimized classifiers with Δ varying from 0 to 1. Note how, as Δ increases, the value of W^+ increases as well and may reach 0.5 (for gen4), i.e., the number of rules deleted by optimization in the approximate case is comparable with the number of rules that could be deleted if all $\mathcal{A}$ contained all three actions.

Action sets from reordered monotone action values. The second type is the same as the first except that the order of decreasing latency now differs from the order of decreasing rule priority. We randomly shuffle the rules and then generate $\mathcal{A}$ sets in the same way as in the first type. Then we optimize the classifier with $\mathcal{A}$ labels generated by the new order and rule priorities in the original order. In this case, W^+ for reduced classifiers is shown on Fig. 5(b); similar to the previous case, it can also reach 0.5 (for gen4 and gen2).

Full action sets for small traffic volumes. In the third type of classifiers, we still assume that the first third of the rules corresponds to the Gold service class, second to Silver, and last to Bronze. But now we assume that fraction s_A of the rules have a very small volume, so the quality of service can be arbitrary and their $\mathcal{A}_i$ contain all three actions. For different s_A from 0 to 0.3 the values of W^+ for reduced classifiers of the third type are shown on Fig. 5(c); note that it reaches 0.4 for $s_A = 0.3$.

General classifiers in this experiment have 50-100 thousand rules, and Boolean minimization techiniques cannot remove more than 6000 rules even when all actions are the same. This means that ClassBench has generated classifiers close to irreducible; that is why we only show the W^+ metric on

Fig. 5. Notice, however, that we are not increasing the error for a large number of rules to reduce the classifier only a little: for rules left untouched by our heuristics (in this case, an overwhelming majority of them) one can easily cut $\mathcal{A}$ back to the original action.

Generally, practical evaluations fully support our basic idea and theoretical results: allowing for a small error in the actions of a classifier can lead to very significant space reductions, often comparable to the largest theoretical reduction in the trivial case when all rules have the same action.

VIII. Related Work

Majority of efficient implementations of packet classifiers in the exact case can be classified into two major categories: *software-based* and TCAM-based; The comprehensive surveys can be found in [12], [13]. Softwared-based solutions mainly rely on decision trees, hashing, or coding-based compression. The works [14], [15] suggest how to partition the multi-dimensional rule space, finding possible matching rules by tracing a path in a decision tree. Techniques to balance the partition in each node exist, but rule replication often is unavoided [16]. There is a fundamental tradeoff between space and time complexities in these approaches. The ABC algorithm for filter distribution offers higher throughput with lower memory overhead [17]. The works [18], [19] discuss hash-based solutions to match a packet to its possible matching rules. Efficient coding-based representations are shown in [20], [21]. Different approaches have been described to reduce number of entries: [22], [23], [24]. They include removing redundancies [2], [3], [4], applying block permutations in the header space [25], transformations [22], [23], [26], [27], [28], [29]. In particular [30], [31], [32] considered representations based on rule disjointness composed with prefix-reorderability to cut down classification width.

IX. Conclusion

In this work, we have generalized the classical *packet classification problem* (exact case), introducing a new abstraction of approximate classifiers, where we control the accuracy by labeling in advance which filters can be assigned to which actions. We have designed optimization methods exploiting this additional flexibility in actions to optimize the classifier size. We believe that approximate classifiers exploit an interesting and unexplored tradeoff between required resources and accuracy of results.

Acknowledgments The work of Vitalii Demianiuk and Kirill Kogan was partially supported by a grant from the Cisco University Research Program Fund, an advised fund of Silicon Valley Community Foundation. The work of Sergey Nikolenko shown in Section VI has been supported by the Russian Science Foundation grant no. 17-11-01276.

References

[1] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 62:1–62:33, 2016.

[2] A. Liu, C. Meiners, and E. Torng, "TCAM razor: a systematic approach towards minimizing packet classifiers in TCAMs," *Trans. Networking*, vol. 18, no. 2, pp. 490–500, 2010.

[3] D. A. Applegate, G. Calinescu, D. S. Johnson, H. Karloff, K. Ligett, and J. Wang, "Compressing rectilinear pictures and minimizing access control lists," in *SODA*, 2007, pp. 1066–1075.

[4] Q. Dong, S. Banerjee, J. Wang, D. Agrawal, and A. Shukla, "Packet classifiers in ternary CAMs can be smaller," in *SIGMETRICS*, 2006, pp. 311–322.

[5] C. Umans, "The minimum equivalent DNF problem and shortest implicants," *J. Comput. Syst. Sci.*, vol. 63, no. 4, pp. 597–611, 2001.

[6] S. Khot and R. Saket, "Hardness of minimizing and learning DNF expressions," in *FOCS*, 2008, pp. 231–240.

[7] R. P. Draves, C. King, S. Venkatachary, and B. D. Zill, "Constructing optimal IP routing tables," in *INFOCOM*, 1999, pp. 88–97.

[8] S. Suri, T. Sandholm, and P. Warkhede, "Compressing two-dimensional routing tables," *Algorithmica*, vol. 35, no. 4, pp. 287–300, 2003.

[9] A. Elmokashfi and A. Dhamdhere, "Revisiting BGP churn growth," *CCR*, vol. 44, no. 1, pp. 5–12, 2014.

[10] Anonymous, "Code for simulations," 2018. [Online]. Available: https://github.com/ACCAAPPROX/Acca

[11] D. Taylor and J. Turner, "Classbench: A packet classification benchmark," *Trans. Networking*, vol. 15, no. 3, pp. 499–511, June 2007.

[12] D. Taylor, "Survey and taxonomy of packet classification techniques," *Comput. Surv.*, vol. 37, no. 3, pp. 238–275, 2005.

[13] G. Varghese, *Network Algorithmics: An Interdisciplinary Approach to Designing Fast Networked Devices*. Morgan Kaufmann, 2005.

[14] P. Gupta and N. McKeown, "Classifying packets with hierarchical intelligent cuttings," *Micro*, vol. 20, no. 1, pp. 34–41, 2000.

[15] B. Vamanan, G. Voskuilen, and T. N. Vijaykumar, "EffiCuts: optimizing packet classification for memory and throughput," in *SIGCOMM*, 2010, pp. 207–218.

[16] X. Zhao, Y. Liu, L. Wang, and B. Zhang, "On the aggregatability of router forwarding tables," in *Infocom*, 2010, pp. 848–856.

[17] H. Song and J. Turner, "ABC: Adaptive binary cuttings for multidimensional packet classification," *Trans. Networking*, vol. 21, no. 1, pp. 98–109, 2013.

[18] S. Dharmapurikar, H. Song, J. Turner, and J. Lockwood, "Fast packet classification using Bloom filters," in *ANCS*, 2006, pp. 61–70.

[19] Y. Kanizo, D. Hay, and I. Keslassy, "Optimal fast hashing," in *Infocom*, 2009, pp. 2500–2508.

[20] A. Korsi, J. Tapolcai, B. Mihlka, G. Mszros, and G. Rtvri, "Compressing IP forwarding tables: Realizing information-theoretical space bounds and fast lookups simultaneously," in *ICNP*, 2014, pp. 332–343.

[21] O. Rottenstreich, M. Radan, Y. Cassuto, I. Keslassy, C. Arad, T. Mizrahi, Y. Revah, and A. Hassidim, "Compressing forwarding tables for datacenter scalability," *JSAC*, vol. 32, no. 1, pp. 138 – 151, 2014.

[22] A. Kesselman, K. Kogan, S. Nemzer, and M. Segal, "Space and speed tradeoffs in TCAM hierarchical packet classification," *J. Comput. Syst. Sci.*, vol. 79, no. 1, pp. 111–121, 2013.

[23] K. Kogan, S. Nikolenko, P. Eugster, and E. Ruan, "Strategies for mitigating TCAM space bottlenecks," in *HOTI*, 2014, pp. 111–121.

[24] O. Rottenstreich, I. Keslassy, A. Hassidim, H. Kaplan, and E. Porat, "Optimal in/out TCAM encodings of ranges," *Trans. Networking*, vol. 24, no. 1, pp. 555–568, 2016.

[25] R. Wei, Y. Xu, and H. J. Chao, "Block permutations in Boolean space to minimize TCAM for packet classification," in *INFOCOM*, 2012, pp. 2561–2565.

[26] C. Meiners, A. Liu, and E. Torng, "Topological transformation approaches to TCAM-based packet classification," *Trans. Networking*, vol. 19, no. 1, pp. 237–250, 2011.

[27] P. Chuprikov, K. Kogan, and S. Nikolenko, "General ternary bit strings on commodity longest-prefix-match infrastructures," in *ICNP*, 2017, pp. 1–10.

[28] ——, "How to implement complex policies on existing network infrastructure," in *SOSR*, 2018, pp. 9:1–9:7.

[29] V. Demianiuk, S. I. Nikolenko, P. Chuprikov, and K. Kogan, "New alternatives to optimize policy classifiers," in *ICNP*, 2018, pp. 121–131.

[30] K. Kogan, S. I. Nikolenko, O. Rottenstreich, W. Culhane, and P. Eugster, "Exploiting order independence for scalable and expressive packet classification," *Trans. Networking*, vol. 24, no. 2, pp. 1251–1264, 2016.

[31] K. Kogan, S. I. Nikolenko, P. Eugster, A. Shalimov, and O. Rottenstreich, "FIB efficiency in distributed platforms," in *ICNP*, 2016, pp. 1–10.

[32] S. I. Nikolenko, K. Kogan, G. Rétvári, E. R. Kovács, and A. Shalimov, "How to represent IPv6 forwarding tables on IPv4 or MPLS dataplanes," in *Infocom Workshops*, 2016.