Layered Interval Codes for TCAM-based Classification*

Anat Bremler-Barr † David Hay ‡ Danny Hendler §

March 29, 2012

Abstract

Ternary content-addressable memories (TCAMs) are increasingly used for high-speed packet classification. TCAMs compare packet headers against all rules in a classification database in parallel and thus provide high throughput.

TCAMs are not well-suited, however, for representing rules that contain range fields and previously published algorithms typically represent each such rule by multiple TCAM entries. The resulting *range expansion* can dramatically reduce TCAM utilization because it introduces a large number of redundant TCAM entries. This redundancy can be mitigated by making use of extra bits, available in each TCAM entry.

We present a scheme for constructing efficient representations of range rules, based on the simple observation that sets of disjoint ranges may be encoded much more efficiently than sets of overlapping ranges. Since the ranges in real-world classification databases are, in general, non-disjoint, the algorithms we present split ranges between multiple *layers*, each of which consisting of mutually disjoint ranges. Each layer is then coded and assigned its own set of extra bits.

Our layering algorithms are based on approximations for specific variants of interval-graph coloring. We evaluate these algorithms by performing extensive comparative analysis on real-life classification databases. Our analysis establishes that our algorithms reduce the number of redundant TCAM entries caused by range rules by more than 60% as compared with best range-encoding prior work.

Keywords: TCAM; Packet classification; Range rules; Routing

1 Introduction

Packet classification is an indispensable building block of numerous Internet applications in the areas of routing, monitoring, security, and multimedia. The routers use a *classification database* that consists of a set of *rule*s (a.k.a. *filter*s). Each such rule specifies a pattern, based on packet header fields, such as the source/destination addresses, source/destination port numbers and the protocol type. Each rule is associated with an action to apply to the packets that matched the pattern rule. Packet classification is often a performance bottleneck in the network infrastructure since it lies in the critical data path of routers. It is therefore important to design packet classification solutions that scale to millions of key search operations per second.

Ternary content-addressable memory (TCAM) devices are increasingly used in the industry for performing high-speed packet classification. TCAM enables parallel matching of a key against all entries and thus provides high throughput that is unparalleled by software-based solutions. A TCAM is an associative memory hardware device that can be viewed as an array of fixed-width entries. Each TCAM entry consists of ternary digits: 0, 1, or

*This work was supported by a Cisco research grant. A preliminary and partial version of this paper appeared in Proc. IEEE Infocom'09 [1].

†School of Computer Science, The Interdisciplinary Center Herzliya, Israel. Email: bremler@idc.ac.il

‡School of Engineering and Computer Science, The Hebrew University of Jerusalem Israel. Email: dhay@cs.huji.ac.il; This work was done while the 2'nd author was a postdoc fellow in the Computer Science department at Ben-Gurion University Israel

§Computer Science Dept. Ben-Gurion University Beer-Sheva, Israel. Email: hendlerd@cs.bgu.ac.il

'*' (don't-care). When a key matches multiple TCAM entries, the TCAM returns the index of the first matching entry. This index is then used to locate the information specifying which actions to apply to the packet. For classification applications, TCAMs are typically configured to be 144 bits wide. This leaves a few dozens of unused bits, called *extra bits*, per each TCAM entry.

A significant obstacle to the efficient use of TCAMs for packet classification is the fact that they are not well suited for representing rules that contain *range fields*, such as port fields. The traditional technique for range representation is the *prefix expansion* technique [2], in which a range is represented by a set of prefixes, such that each is stored in a single TCAM entry. For example, the range $[1, 6]$ can be represented by the prefix set $\{001, 01*, 10*, 110\}$. Hence, a single rule may require multiple TCAM entries, resulting in *range expansion*. Range expansion was found to cause an increase of more than 16% in TCAM space requirements for real-word databases [3].

1.1 Our Contributions

We present a scheme for constructing efficient representations of range rules, by making efficient use of TCAM extra bits. The scheme is based on the simple observation that sets of disjoint ranges may be encoded much more efficiently than sets of overlapping ranges. Since the ranges in real-world classification databases are, in general, non-disjoint, the algorithms we present split the ranges between multiple *layers*, each of which consists of mutually disjoint ranges. Each layer is then coded and assigned its own set of extra bits. We call the resulting encoding scheme a *Layered Interval Code* (LIC). Roughly speaking, our goal is to find a minimum-size LIC code, where the *size* of a LIC code is the number of extra bits it uses.

We consider two problems related to the space-efficient construction of LIC codes. An instance of the *minimum-space LIC* (MLIC) problem consists of a set $\mathcal{S}$ of ranges. The MLIC problem is to output a LIC code for the ranges of $\mathcal{S}$ that uses a minimum number of bits. We call the second problem the *budgeted minimum-space LIC* (BMLIC) problem. An instance of BMLIC consists of a set $\mathcal{S}$ of *weighted* ranges and a positive integer b, the number of available bits. The BMLIC problem is to output a LIC code for a maximum-weight subset $\mathcal{S}'$ of $\mathcal{S}$, such that the code size of $\mathcal{S}'$ is at most b.

This paper presents several novel algorithms that solve the MLIC and BMLIC problems. Our algorithms use approximations for specific variants of interval-graph coloring as layering building blocks and exhibit different tradeoffs between implementation complexity and space efficiency. We then use our approximation algorithms for the efficient encoding of classifier range rules using the available budget of extra bits that exists in each TCAM entry. We have obtained real-life experimental results, comparing our algorithms with prior work, by using a large real-life classification database consisting of more than $223K$ rules. Our empirical results show that all of our algorithms reduce the average number of redundant TCAM entries required to represent a range rule by more than 60% as compared to best range-encoding prior work. A known drawback of TCAM devices is their high power consumption and heat generation. Since the energy consumed by TCAM devices grows linearly with the number of classification rules it stores [21, 22], efficient representation of range rules results in reduced power consumption.

In practice, classification databases change over time. To maintain space efficiency, every practical range encoding scheme whose encoding is a function of the database ranges-distribution (such schemes are named *database-dependent*) must change its encoding in accordance with these changes. First and foremost, it is crucial to guarantee *hot updates*; namely, that while TCAM entries are being updated, the device can still be used to classify incoming packets. Our scheme supports a novel and efficient hot updates mechanism that can be combined with any database-dependent encoding scheme (including ours). This algorithm requires only a single extra bit per TCAM entry.

On the theoretical side, we formalize the MLIC problem and prove that it is NP-hard by a reduction from the *circular arc graph coloring* problem, whose NP-hardness was established in [4]. We also formalize the BMLIC problem and use a reduction from the MLIC problem to prove that it also is NP-hard. Since our NP-completeness results establish that finding exact solutions for the MLIC and BMLIC problems efficiently is

impossible, they motivate our use of approximation algorithms. We also prove that an algorithm presented in [5] for the closely related *chromatic sum* problem [6] is a polynomial-time 2-approximation algorithm for MLIC. We use this approximation algorithm as a building block in some of the algorithms we present.

2 Background

The issue of using TCAM devices for packet classification has received considerable attention from the research community over the past few years. A key issue dealt with by researchers in this regard is that of improving the space-efficiency of TCAM range representation. This issue was considered both from the algorithmic [7, 8, 9, 10] and the architectural [11] perspectives.

Spitznagel, Taylor, and Turner, introduced *Extended TCAM* [11], which implements range matching directly in hardware in addition to reducing power consumption by over 90% relative to standard TCAM. While this may represent a promising long-term solution, it seems that changing the ternary nature of TCAM entries while maintaining reasonable per-bit cost and addressing scalability issues will not be accomplished in the near future.

2.1 Prefix Expansion

The traditional technique for range representation, originated by Srinivasan et al. [2], is to represent a range by a set of prefixes, each of which can be stored by a single TCAM entry. The worst-case expansion ratio when using prefix expansion for ranges whose endpoints are W-bits numbers is $2W - 2$. The problematic range is $R_w = [1, 2^w - 2]$, whose smallest cover set is $\{01{\star}^{w-2}, 001{\star}^{w-3}, 0001{\star}^{w-4}, \cdots, 0^{w-1}1, 10{\star}^{w-2}, 110{\star}^{w-3}, \cdots, 1^{w-1}0\}$.

As observed by Taylor [3], a single rule that includes two 16-bit range fields could, in the worst-case, require $(2 \cdot 16 - 2)^2 = 900$ entries. Though such rules are rare in practice, analysis of real-world databases, provided by [3], revealed that prefix expansion may increase the number of required TCAM entries by a factor of more than 6.

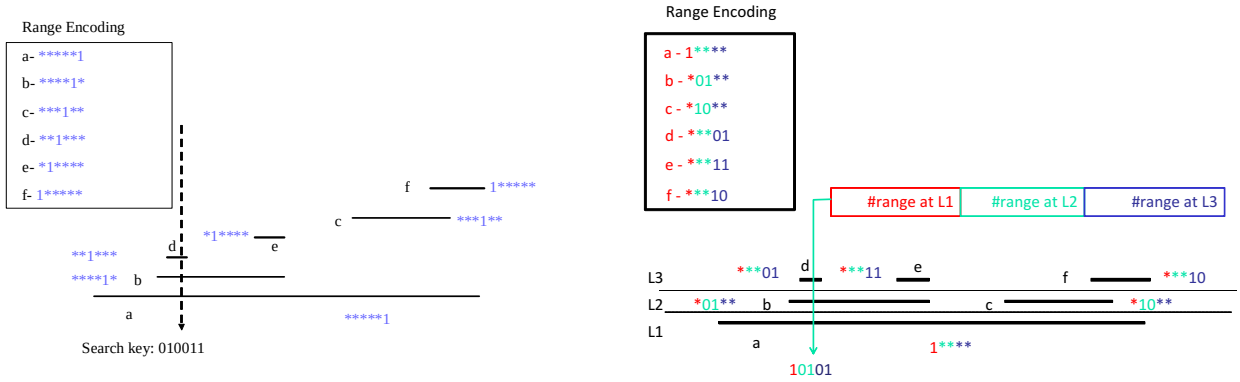
2.2 Independent Range Encoding

Prior work that deals with the range expansion problem can roughly be divided to two main categories: database-independent and database-dependent range encoding algorithms. The encoding of a range by a database-dependent scheme is a function of the distribution of ranges in the database in which it occurs. In contrast, the encoding of a specific range by a database-independent scheme does not change across different databases. Lakshminarayana et al. present a database-independent range encoding algorithm that is based on the concept of *fence encoding* [9]. Bremler-Barr and Hendler [12] present a database-independent algorithm that is based on the observation that small ranges, which occur frequently in real-world databases, are encoded more efficiently by using *Gray code* [13]. Both these works also present hybrid versions that are database-dependent.

2.3 Dependent Range Encoding

The first database-dependent prior work is due to Liu [7]. The basic idea is to use the available extra bits as a bit map: a single extra bit is assigned to each selected range r in order to avoid the need to represent r by prefix expansion. Figure 1(a) illustrates the details of this technique. If range r is assigned extra bit i, then the i'th extra bit is set in all TCAM entries that include r; all other extra bits are set to 'don't care' in these entries. In the search key, extra bit i is set to 1 if the key falls in range r or set to 0 otherwise.

This basic scheme eliminates redundancy for every range that is assigned an extra bit. However, since the number of ranges whose redundancy may be eliminated is bounded by the number of extra bits, this solution does not scale. Indeed, as observed by [9], the number of unique ranges in today's classification databases is



(a) (b)

Figure 1: Encoding Example using (a) Liu's Basic Dependent Encoding; (b) Region Partitioning encoding.

around 300 and is anticipated to continue to grow in the future. This growth is expected to accelerate, due to the use of TCAMs by Intrusion Detection Systems (IDS), which add new range fields, such as a packet-length field.

To alleviate this scalability problem, *Region Partitioning* was proposed in [7]. Region partitioning is illustrated in Figure 1(b). This partitioning may split a range into multiple sub-ranges. Each such sub-range is encoded by two numbers: the region number into which it falls, and the sub-range number within that region. Unfortunately, in practice, this algorithm often results in high expansion (see our experiment results in Section 7), since a range is divided to multiple sub-ranges and the encoding of each sub-range required a separate TCAM entry.

Che et al. present a bit-map based scheme, called DRES, that employs a *dynamic range selection* algorithm for selecting the ranges that are to be assigned extra bits [14]. DRES is a greedy algorithm that assigns extra bits to the ranges with highest prefix expansion. The key difference between DRES and our scheme is that, whereas DRES assigns a single bit per range, we use the much more compact LIC coding. This allows our scheme to give better 'bang for the (extra) bit', since we are able to eliminate more redundancy using the same number of bits as compared with DRES.

Chang and Su [15] present a range-representation scheme based on Gray code. Their algorithm aggregates the elementary intervals induced by classifier ranges by using Binary Reflected Gray Code encoding. Unlike our work, [15] focuses on minimizing the required number of extra bits without assuming any bound on the number of available extra bits per TCAM entry. Zheng et al. [16] present a distributed TCAM-based packet classification scheme. Their scheme supports the incorporation of the DRES [14] range-encoding algorithm.

Rottenstreich and Keslassy [18] present a database dependent encoding scheme that encodes any range by using at most W TCAM entries, where W is the width in bits of range fields. Their encoding algorithm relies on the fact that each action applied by a rule is associated with a "default action" that should be applied to packets that do not match the rule. A similar approach was studied also by Cohen and Raz [17].

The work most closely related to ours is that of Lunteren and Engbersen [8]. They present Parallel Packet Classification ($\mathrm{P^2C}$), for parallel field searches. The encoding employed by $\mathrm{P^2C}$ for range representation is essentially a layered interval code. However, [8] does not present explicit layering algorithms. More importantly, unlike our work, it does not address the budgeted minimum-space LIC problem, which arises in realistic settings in which the number of available per-entry extra bits is limited.

2.4 Classifier Minimization Algorithms

A more general approach is *classifier minimization*. Classifier minimization algorithms encode all classification rules and not only range rules. Key examples of such algorithms are [18, 19, 20, 21, 22, 23, 24, 25]. While classifier minimization reduces TCAM memory consumption considerably, it also has two significant drawbacks. First, the encoded classifier may be extremely different from the original classifier, as configured by the system administrator, making it very difficult for the administrator to track statistics (such as, e.g., the number of hits on each configured rule); range-expansion reduction techniques such as ours do not suffer from this problem. A second disadvantage is that classifier updates are more expensive because the update of even a single rule may significantly influence the representation's efficiency. In contrast, the efficiency of our scheme, as well as other range-expansion reduction algorithms, may decrease only when the set of *unique* ranges changes, or if there is a significant change in the number of occurrences of ranges in the database.

Pao et al. [23] present the *Prefix inclusion coding* (PIC) classifier minimization algorithm. They explain how PIC can be applied also for range encoding. However, when ranges overlap, their algorithm must perform range decomposition which may significantly reduce its efficiency.

3 The Layered Interval Encoding Scheme

The LIC encoding scheme is based on the simple observation that, while encoding n arbitrary ranges may require n bits, only $\log(n+1)$ bits are required to encode n *disjoint* ranges. Our algorithm is composed of three stages: a *layering stage*, a *bit allocation stage*, and an *encoding stage*. Note that all these stages are done in a pre-processing phase, thus lookup operations still take only a single TCAM cycle.

In the *layering stage*, we partition the intervals[1] that occur in the database into a set of interval-sets $\mathcal{C}$, such that all the intervals in any interval-set $L \in \mathcal{C}$ are mutually disjoint. We call the set $\mathcal{C}$ the *layering* of the intervals. In Section 3.2 we propose several polynomial-time layering algorithms that are based on approximation algorithms for the BMLIC and MLIC problems.

In the *bits allocation stage*, we iterate over extra bits, deciding for each to which layer it should be assigned. Roughly speaking, a bit is assigned to the layer for which the total weight of intervals that can now be coded by using this bit is maximum. This process minimizes range-expansion, while not exceeding the extra bits budget. We describe the bits allocation stage in detail in Section 3.3.

Finally, in the *encoding stage*, we construct the corresponding search keys and TCAM entries, based on the outputs of the layering and bits allocation stages. The selected ranges of each layer are encoded independently (of other layers) by using available extra-bits. Roughly speaking, a search key is constructed by encoding, for each layer, the single range of this layer to which the search key falls; if no such range exists, we encode a value representing the "area" outside all of this layer's ranges. A range-entry is constructed by encoding the range (at the single layer to which it belongs), while using 'don't care' bits for all other layers. This stage is described in more detail in Section 3.4. Figure 2 demonstrates the encoding on the same sets of ranges that were shown in Figure 1.

In the following, we describe each of the above stages in more detail. Before that, we define required terminology in Section 3.1. For presentation simplicity, the following description assumes that each rule contains at most a single range field. This assumption is later removed in Section 4.1.

3.1 Terminology

A packet header consists of fields, each of which is a bit string. A *key* is a collection of κ fields from the packet header. Keys are matched against classification *rules* stored in *entries* of a *classification database*.

[1] Throughout this section, we mostly use the term 'interval' instead or 'range', since the approximations we employ are based on algorithms for interval-graph coloring.

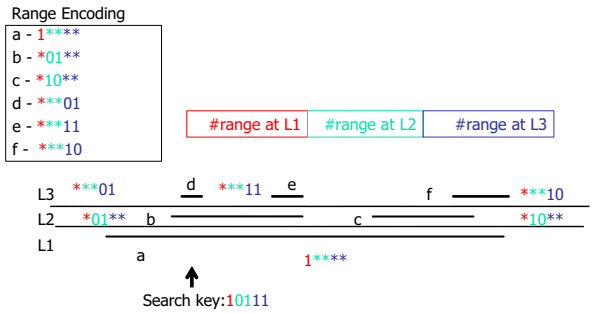


Figure 2: Example of LIC encoding.

Rules consist of κ fields matching the corresponding key fields. A packet p with header $h(p)$ matches a rule R if each of the κ key fields of $h(p)$ matches the corresponding field of R. Each rule field f can specify one of three types of *matches*: *exact match*, *prefix match*, or *range match*.

In general, rules containing a range field (henceforth called *range rules*) cannot be represented by a single TCAM entry and *range encoding schemes* are used to encode each such rule as a set of TCAM entries. An encoding scheme maps each range r to a set of TCAM entries that represent it, called the *cover set of* r.

The quality of the range encoding scheme E is measured by the number of additional entries it requires for encoding range rules. Specifically, the *expansion* of a range r is the size of its cover set under scheme E, while the *basic redundancy* of a range r is defined to be r's expansion minus 1. The *expansion of a rule* R is the product of the expansion of all its κ fields, where the expansion of non-range fields is defined as 1. The *basic redundancy* of a rule R is its expansion minus 1.

Let $\mathcal{D}$ be a classification database such that some of its rules contain range r. Under scheme E, the *total redundancy* of r in $\mathcal{D}$ is the product of r's basic redundancy and the number of rules of $\mathcal{D}$ in which r occurs. In other words, r's total redundancy is the total number of redundant entries that are required if we use encoding scheme E for r in $\mathcal{D}$. The *database expansion factor* of a database $\mathcal{D}$ using scheme E is the relative increase in the number of entries required to represent $\mathcal{D}$ in TCAM using coding scheme E.

Since this paper deals with encoding range fields, we focus our attention on *range rules*, that is, rules that contain range fields. The *range redundancy factor* of a database $\mathcal{D}$ using scheme E is the average total redundancy of the range rules in $\mathcal{D}$ using E, that is, the average number of redundant TCAM entries per range rule in $\mathcal{D}$. Thus, the range redundancy factor is a measure of an encoding scheme's efficiency, whereas the database expansion factor is also a function of the fraction of range rules within a database. Clearly, the lower the range redundancy factor of a scheme E is, the more efficient it is.

We consider two variations of the LIC problem: MLIC and BMLIC. Formally, these problems are coloring problems for interval graphs [2].

Definition 1 *Let G be an interval graph. The* minimum space layered interval-code *(MLIC) problem is to find a legal coloring $\mathcal{C}$ of G that minimizes $\sum_{i=1}^{|\mathcal{C}|} \log_2(n_i + 1)$, where n_i is the number of the nodes of G assigned color i by $\mathcal{C}$. We say that a coloring $\mathcal{C}$ of G* has LIC code-size b *if the expression above is b for that coloring. We let $L(G, \mathcal{C})$ denote the LIC code-size associated with coloring $\mathcal{C}$ of G.*

Let $\mathcal{I}$ denote the set of intervals that occur in the database. The input to the MLIC problem is the interval graph corresponding to $\mathcal{I}$ and its output is a layering of the intervals in $\mathcal{I}$ that uses a minimum number of extra bits. The MLIC problem is closely related to the *chromatic sum* (a.k.a. *sum coloring*) problem [6], in which

[2] An *interval graph* is an undirected graph $\langle V,E \rangle$. Each node $v \in V$ corresponds to an interval I_v. For every pair of distinct nodes $v, u \in V$, E contains an edge between v and u if and only if $I_v \cap I_u \neq \emptyset$. A (vertex) *coloring* of a graph G is an assignment of colors to the nodes of G such that no two adjacent vertices are assigned the same color.

vertices are colored using natural numbers and the *sum* of the colors needs to be minimized: the objective function of both these problems improves (i.e., decreases) as the distribution of intervals to colors becomes asymmetric.

Definition 2 *Let $G = \langle V, E\rangle$ be a weighted interval graph such that $w(v)$ is the weight of vertex $v \in V$, and let $b \in \mathbb{N}$ be a LIC code-size budget. The* budgeted minimum layered interval-code (BMLIC) *problem is to find a subset $V' \subseteq V$ and a coloring $\mathcal{C}$ of the subgraph of G induced by V', such that $\sum_{v \in V'} w(v)$ is maximum under the constraint that $\mathcal{C}$ has LIC code-size of at most b.*

Let $\mathcal{I}$ denote the set of *weighted intervals* that occur in the database, where the weight of each interval r in $\mathcal{I}$ is r's total redundancy in the database. Also, let b denote the number of extra bits in each TCAM entry. A solution of the BMLIC problem with inputs $\mathcal{I}$ and b returns a LIC encoding of size at most b for a subset $\mathcal{I}'$ of $\mathcal{I}$ that saves a maximum number of redundant TCAM entries.

Note that all the intervals in $\mathcal{I} \setminus \mathcal{I}'$ must be encoded using an alternative *fall-back* scheme, since $\mathcal{I} \setminus \mathcal{I}' \neq \emptyset$ implies that all extra bits are used by LIC encoding. Database-independent encoding schemes that do not require extra bits, such as prefix expansion [2] and SRGE [12], may be used as fall-back encoding schemes.

3.2 The Layering Stage

The first stage in our encoding scheme is the *layering stage*, where we partition the intervals that occur in the database into a set of interval-sets $\mathcal{C}$, such that all the intervals in any interval-set $L \in \mathcal{C}$ are mutually disjoint. Let $G = \langle V, E\rangle$ be an interval graph. We evaluate the following four layering algorithms:

a) Maximum Size Independent Sets (MSIS): This is a greedy layering algorithm that works iteratively. Let $G_0 = G$. In iteration $i \geq 1$, the algorithm finds a *maximum size independent set*, L_i, of the interval graph $G_{i-1} = \langle V_{i-1}, E_{i-1}\rangle$ and defines $G_i = \langle V_i, E_i\rangle$ to be the subgraph of G_{i-1} induced by $V_{i-1} \setminus L_i$. The algorithm stops when V_i is empty. Let $\mathcal{C} = \{L_1, L_2, \ldots\}$. $\mathcal{C}$ is a coloring of G since if two vertices belong to the same layer L_i they are not adjacent in G. Bar-Noy et al. [26] show that MSIS is a 4-approximation to the chromatic sum problem.

b) Maximum Size Colorable Sets (MSCS): An *i-colorable* set of a graph G is a subset of G's vertices that can be colored with i colors. MSCS finds a maximum size i-colorable set of G, denoted A_i, for each $1 \leq i \leq \chi(G)$, where $\chi(G)$ denotes the chromatic number of G. The algorithm proceeds recursively on each sub-graph induced by the set of vertices $A_i \setminus A_{i-1}$ (where $A_0 = \emptyset$) and returns the union of all the layerings that result from these recursive calls (see Algorithm 1). The procedure *GRAPH* (not shown) returns the sub-graph of its first parameter, G, induced by the vertices-set passed as its second parameter. Nicolso et al. present a slightly different but equivalent version of MSCS and prove that it is a 2-approximation to the chromatic sum problem [5]. We prove in Section 8 that MSCS is also a 2-approximation for the MLIC problem.

Algorithms MSIS and MSCS do not take interval weights into consideration. If the budget of available extra bits is significantly smaller than that required for an optimal solution of MLIC, then the weight of the layering obtained may be far from optimal regardless of how bits are partitioned in the bits allocation stage (explained shortly). This motivates the following two heuristic algorithms.

c) Maximum Weight Independent Sets (MWIS): same as MSIS, except that we iteratively find a maximum *weighted* independent set.

d) Maximum Weight Colorable Sets (MWCS): same as MSCS, except that, instead of finding maximum size k-colorable sets (Line 5 in the pseudo-code of Algorithm 1), MWCS finds *maximum weight* k-colorable sets. Finding a maximum-weight k-colorable set of an interval graph can be done in polynomial time by transforming the graph G into a directed acyclic network graph G' and solving the *minimum cost flow* problem [27] on G'. The minimum cost flow problem, in turn, can be solved in polynomial time by using the widely-used paradigms of *augmentation paths* and *negative cycle cancelations* [28].

The time complexity of all the aforementioned layering algorithms is $O(n^2)$, where n is the number of input intervals [5, 26].

Algorithm 1 Maximum Size Colorable Sets (MSCS)

1: layering **procedure** MSCS(Graph $G = \langle V, E \rangle$)
2: $k := 0$, $A_0 := \emptyset$
3: **while** $|A_k| \neq |V|$ **do**
4: $k := k + 1$
5: $A_k :=$ Maximum size k-colorable set of graph G.
▷ Using ALGORITHM LB [5, PP. 114-115]
6: **end while**
7: **if** $k = 1$ **then**
8: **return** $\{A_1\}$
9: **end if**
10: $\mathcal{C}_1 :=$ MSCS(GRAPH(G, A_1))
11: **for** $i = 2$ to k **do**
12: $\mathcal{C}_i :=$ MSCS(GRAPH$(G, A_i \setminus A_{i-1})$)
13: **end for**
14: **return** $\bigcup_{i=1}^{k} \mathcal{C}_i$
15: **end procedure**

3.3 The Bits Allocation Stage

In this stage we assign extra bits to layers and determine which intervals are to be encoded. This process is done by the *Bit Auction* algorithm whose pseudo-code appears as Algorithm 2. The $assigned$ array stores the numbers of bits assigned to layers: $assigned[i]$ is the number of bits assigned to the i'th layer of C. The algorithm works in b iterations, where b is the number of available extra bits. Each iteration may be viewed as an auction, in which layers compete for the next available bit. The intervals in each layer are sorted in decreasing order of their weight (where ties are broken arbitrarily) (line 4). If a layer L_i has already been assigned $assigned[i]$ bits, then assigning it additional k bits allows us to encode L_i's next $2^{assigned[i]+k} - 2^{assigned[i]}$ intervals. Assume that all intervals are sorted in decreasing weight order within each layer and let $L[i][j]$ denote the interval of L_i with the j'th largest weight. Then the *per bit* decrease in redundancy gained by allocating the next k bits to layer L_i is: $\frac{1}{k}\sum_{j=2^{assigned[i]}}^{2^{assigned[i]+k}-1} w(L[i][j])$. We compute the above quantity for each layer L_i (line 9) and assign the next bit to a layer for which this quantity is maximum (line 11-13). Note that the first bit allocated to each layer accommodates only *a single* interval, since value 0 is interpreted as 'not in layer'.

Given a layering $\mathcal{C}$ that contains n intervals and a constant number of extra-bits b, an optimal bits allocation can be computed polynomially by an exhaustive search on all possible allocations. However, this algorithm has a prohibitive time-complexity of $O(n \cdot \log n + n \cdot |\mathcal{C}|^b)$. The Bit Auction algorithm, on the other hand, is fast (our implementation has time-complexity $O(n \cdot \log n + n \cdot b^2)$) and achieves excellent results in practice.

3.4 The Encoding Stage

The essence of the encoding stage is implemented by the *Encode* procedure that assigns codes to the intervals that were selected for encoding in the bits allocation stage. These codes are then used to determine the values of the range-field in TCAM entries and the values of the extra bits in search keys.

The code assigned to an interval is determined by the layer it belongs to (by definition, each interval belongs to a single layer). As a result of the bits allocation stage, each layer L_i is assigned a range of extra bit indices (k_{i_1}, k_{i_2}), of length $assigned[i]$, that is dedicated for encoding the intervals of that layer. The intervals of layer L_i are therefore encoded within these boundaries in a decreasing order of their weight. For example, if $assigned[i] = 3$, the interval with largest weight in layer i is assigned value 001, the intervals with the second and third largest weights in layer i are assigned values 010 and 011, respectively, and so on. All bits outside these boundaries (that is, with index at most k_{i_1} or at least k_{i_2}) are set to $\star$. The pseudo code of this stage is given by the *Encode* procedure (see Algorithm 3) that uses the following notations: bin(j) denotes the binary

Algorithm 2 Bit Auction

```
1: int[] procedure BIT-AUCTION(layering C, int b)
2:     assigned[|C|], p[|C|]: arrays of integers, initially all 0
3:     for all L_i ∈ C do
4:         ⟨L[i,1], L[i,2], ··· , L[i,|L_i|]⟩ := Sort L_i's intervals in
5:                                               decreasing weight order
6:     end for
7:     while b > 0 do
8:         for all L_i ∈ C do
9:             p[i] := max_{k∈[1,b]} (1/k) Σ_{j=2^{assigned[i]}}^{2^{assigned[i]+k}-1} w(L[i,j])
10:        end for
11:        m := arg max_i p[i]
12:        assigned[m] = assigned[m] + 1
13:        b := b − 1
14:    end while
15:    return assigned
16: end procedure
```

representation of the integer j and $\star^k$ denotes a string consisting of k 'don't-care' symbols. The concatenation of strings A and B is denoted by $A \cdot B$. The procedure returns as output the *code* array (line 13). Entry $code[i][j]$ stores the code assigned to interval j of layer i, where, as before, intervals are sorted within each layer in decreasing order of weight. The time complexity of the *Encode* procedure is $O(n)$, where n is the number of intervals, since it performs a single iteration per every input interval.

The number of extra-bits b is limited, hence not all ranges may be assigned LIC codes. Ranges that are not assigned a LIC code can be represented using any database-independent encoding scheme, such as prefix expansion or Gray code [13]. If a rule contains such a range, then the extra bits corresponding to that field are set to 'don't-care's and the range-field is encoded by the fall-back technique. If a rule's range-field *was* assigned a LIC code then the field value is set to 'don't-care's.

The second task of the encoding stage is to determine the values of search keys' extra bits: For each search key value x, the extra bits are determined by performing a bitwise OR of the vector 0^b with the codes of all the ranges that contain x. These codes are retrieved from the output of the *Encode* procedure. We assume here that an OR of a proper bit value (0 or 1) with $\star$ returns the proper bit value. Note that, since ranges of different layers use different extra bits, and since the ranges within the same layer are mutually disjoint (that is, at most one range in each layer can contain x), this bit-wise OR operation effectively concatenates the codes of all the ranges intersected by the entry; bits corresponding to layers in which no range is intersected by the entry are set to 0 by this operation. It is also important to notice that the encoding stage modifies only the extra bit values; the rest of the search-key's bits remain unchanged.

4 Implementation Considerations

In this section, we explain how the basic LIC scheme, described thus far, is modified to support multiple range fields. We also provide more details about the architecture of the LIC scheme.

4.1 Supporting Multiple Range Fields

For presentation simplicity, in the previous sections we have mostly ignored the fact that real-life classification-database ranges may occur in more than a single field. In the following we describe how our scheme supports the

Algorithm 3 Encode

```
1: int[][] procedure ENCODE(layering C, int[] assigned)
2:     for all L_i ∈ C do
3:         k_1 := Σ_{m=1}^{i-1} assigned[m]
4:         k_2 := Σ_{m=i+1}^{|C|} assigned[m]
5:         for all r_j ∈ L_i do
6:             if j ≤ 2^{assigned[i]} − 1 then
7:                 code[i][j] := *^{k_1} · bin(j) · *^{k_2}
8:             else
9:                 code[i][j] := ⊥
10:            end if
11:        end for
12:    end for
13:    return code
14: end procedure
```

two IP header range-fields: the source-port (s-port) and destination-port (d-port) fields. We note that our scheme can be easily extended to support a larger number of range-fields.

Our algorithm maintains two separate range sets: one for the ranges that occur in the s-port field and another for the ranges that occur in the d-port field. The layering stage of our algorithms is performed separately for each of these sets. Nevertheless, the encoding of these two sets is not independent, as we now explain.

Assume that range r occurs in a certain rule $\mathcal{R}$ as the s-port field and that another range s occurs in the d-port field of rule $\mathcal{R}$. In this case, if neither r nor s are assigned LIC codes, then the number of TCAM entries required to represent $\mathcal{R}$ is the product of the expansion of both ranges. More generally, the contribution of r to the total database redundancy depends on the basic redundancy (see Section 3.1) of s and whether or not s is assigned a LIC code.

We deal with this issue as follows. The weight attributed to r on account of a rule where it co-occurs with s is computed as $b(r) \cdot b(s) - b(s)$, where $b(x)$ denotes the basic redundancy of x. This is exactly the number of redundant entries that can be saved in the representation of the rule if r is assigned a LIC code assuming s is not assigned a LIC code.

The bits allocation stage is done on the union of layers that were obtained for both range sets, with the following modification applied to the algorithm described in Section 3.3: At the end of each algorithm iteration, the weight of a range r that co-occurs with a range s is decreased if s was assigned a LIC code in the course of the iteration. Finally, the encoding stage is performed separately for each of the range sets.

4.2 LIC Scheme Architecture

In all database-dependent range-encoding schemes, incoming headers have to be preprocessed to create a search key. For the LIC scheme, the values of all fields (i.e. the source/destination IP, source/destination ports and protocol fields) are simply copied from the header to the corresponding fields of the search key. Preprocessing is required only for determining the values of the extra bits. Our scheme employs pre-computed tables that map the ranges to the extra bits for fast preprocessing. We now describe the preprocessing process in more detail.

Two direct-access lookup tables are used, called SOURCE-KEYS and DEST-KEYS, for storing the extra bits of the search key corresponding to the specific value of s-port and d-port fields, respectively (see Section 3.4 for details about the encoding procedure of a specific search key). The SOURCE-KEYS and DEST-KEYS tables are accessed by using the s-port and d-port field values, respectively, as indices. The extracted values are then concatenated to construct the extra-bits part of the search key.

These two tables are small and can therefore be stored in fast memory. Each of the port fields is 16-bit wide, and hence the two tables have a total of $128K$ entries. Assuming a typical budget of 36 extra bits per TCAM

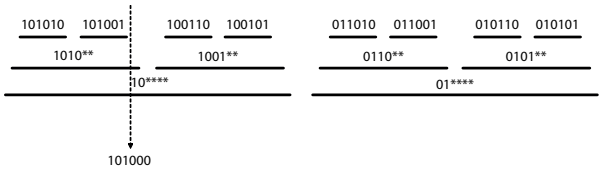


Figure 3: A problem instance demonstrating that the optimal LIC code may be far from the optimal encoding by a factor of $\Theta(\log n)$. The arrow points to the encoding of a specific search-key point that falls within the leftmost intervals of the first two layers but falls outside the intervals of layer 3. It is therefore encoded by 101000.

entry and an SRAM word-size of 32 bits, the two tables occupy at most $384K$ SRAM words altogether.

5 Hot Updates Support

In general, the rules of a classification database change over time. To maintain space efficiency, every practical database-dependent range-encoding scheme must eventually change its encoding in accordance with these changes. First and foremost, it is crucial to guarantee that, while TCAM entries are being updated, one can still use the device to classify incoming packets. If this requirement is met, we say that the scheme supports *hot updates*.

We now describe a simple general scheme for hot updates that can be combined with any database-dependent range-encoding scheme. Our approach uses a single extra bit for every rule, called the *phase bit*.

We assume the existence of two procedures, called ADD_ENTRY and DELETE_ENTRY, that respectively add and delete a single TCAM entry. Efficient implementations of such procedures were presented in prior art (see, e.g., [29]). Our algorithm alternates between 0-phases and 1-phases, depending on the value of a global *phase* variable.

Whenever enough updates have occurred to trigger an encoding re-computation, the following two steps are performed:

1. We employ the ADD_ENTRY procedure for adding all the rules that we want to encode using the new encoding. The phase bits of these rules are set to the value of the new phase, that is, they are set to 1 if the current phase is 0 or set to 1 otherwise. Note that, right after adding these entries, each range rule might be represented *twice* in the database, since the range may be encoded differently in the current and new phases. Non-range rules, however, are never duplicated.

2. After performing step 1, newly-encoded rules are still ineffective, since the phase-bit's value in all search keys is still set to the value of the current phase. The algorithm therefore proceeds to update the search keys. For example, assume that the current phase is a 0-phase (hence the new phase is a 1-phase) and consider a search key of some port-range x which equals y in the current phase and is changed to y' in the new phase. Then, the value of the SRAM array entry corresponding to x is modified from $0y$ to $1y'$. Since all relevant range rules are currently duplicated in the TCAM device, search-key updates may proceed lazily, and there is no need to duplicate SRAM array entries.

   After the update process of the search keys is completed, the value of the $phase$ variable is toggled (in our example it is changed from 0 to 1). Finally, all the obsolete rules corresponding to the old phase can be deleted from the TCAM by invoking the DELETE_ENTRY procedure. Since these rules are never matched in the course of the new phase, deletions may proceed lazily as well.

6 Cross-Layered Encoding (CLE)

The LIC algorithm achieves efficient encoding by exploiting the fact that the ranges within each layer are disjoint. In this section, we show that whenever a range in some layer is fully contained within a range in another layer, encoding efficiency may be improved even further. We present an enhancement of LIC, which we call *Cross-Layer Encoding* (CLE), that exploits containment relations between ranges in different layers. To motivate CLE, we now describe a family of problem instances on which it outperforms LIC asymptotically by a factor of $\log n$.

Consider a problem instance that consists of $\log n$ layers, where, for $i = 1, 2, \cdots, \log n$, layer L_i contains 2^i ranges. Thus, there are $\Theta(n)$ ranges overall. Let L_i^j denote the j'th range in layer L_i, then, for $i = 1, 2, ..., \log n - 1$ and $1 \leq j \leq |L_i|$, ranges L_{i+1}^{2j-1} and L_{i+1}^{2j} are contained in range L_i^j. Figure 3 presents such a problem instance with 3 layers.[3] It is easily seen that the optimal LIC encodes each of the $\log n$ layers independently. This is also the LIC code that is produced by the MSIS and MSCS layering algorithms. The size of the resulting LIC is $\sum_{i=1}^{\log n} \log(2^i + 1) \approx \frac{\log n \cdot (\log n + 1)}{2} = \Theta(\log^2 n)$.

Figure 3 also shows a CLE encoding that exploits the special structure of these problem instances. It encodes the ranges in a problem instance comprising $\log n$ layers by using only $2 \log n$ bits as follows. The $\log n - 2$ rightmost bits of the ranges in layer 1 are set to don't-cares. As for the 2 leftmost bits - L_1^1 is encoded by 10, L_1^2 by 01, and the area not in either of these two ranges is encoded by by 00. For each level $i > 1$, encoding proceeds as follows. The leftmost $2(i-1)$ bits of L_i^j are the concatenation of the bits of the $i - 1$ ranges that contain it in the lower layers. Bits $2i - 1$ and $2i$ of L_i^j are 10 if j is odd or 10 otherwise. Bits $2i - 1$ and $2i$ of the layer's area that is outside the intervals are set to 00. All the bits to the right of bit $2i$ (if any) are set to don't-cares. It is easily seen that this is a valid encoding that uses only $2 \cdot \log n$ bits to encode the problem instance with $\log n$ layers, as compared with the $\Theta(\log^2 n)$ bits required by LIC.

In the rest of this section we describe the CLE algorithm in more detail. In a sense, CLE creates *sub-layers* within each LIC layer. For example, the third layer of Figure 3 consists of four sub-layers, each of which consists of two ranges. The codes "01" and "10" are used in *each* of these sub-layers. This is in contrast with the LIC encoding which assigns *distinct* codes to all the ranges within the same layer. As we soon describe, CLE guarantees that ranges that belong to different sub-layers of the same layer can be distinguished based on codes that they "inherit" from the ranges that contain them in other layers. On the other hand, ranges that belong to the same sub-layer are assigned distinct sub-layer codes and thus can also be distinguished from one another.

6.1 The CLE Sub-Layering Stage

Consider a layering $\mathcal{C}$ (produced, e.g., by one of the layering algorithms presented in Section 3.2) and assume an assignment of extra bits to these layers, computed by the *Bit Auction* algorithm and represented by the $assigned$ array (see Section 3.3). In the following description, we only consider the ranges that are encoded according to the bit assignment represented by the $assigned$ array, that is, for each layer i we only consider the $2^{assigned[i]} - 1$ most heavy-weight ranges of layer i.

The sub-layering stage proceeds in two phases. In the first phase, we associate with each range r in layer L_i a vector v_r of length $\mathcal{C}$, whose j-th element $v_r[j]$ may assume the following values: 1) $\perp$ if $i = j$, 2) the identifier of a range in layer L_j that contains r, if such a range exists, 3) 0 if there is no intersection between r and any of the ranges in layer L_j, or 4) '$\star$' otherwise. We note that vector v_r is well-defined, since the ranges within any single layer are disjoint.

In the second phase, we define, for each layer i, an undirected graph G_{L_i} that represents the encoding-dependencies between each pair of ranges within layer i. The vertices of G_{L_i} are L_i's ranges; there is an edge between two ranges r and r' in G_{L_i} if and only if, for every $j \in \{1, \ldots |\mathcal{C}|\}$, either $v_r[j] = v_{r'}[j]$, or $v_r[j] = \star$ or $v_{r'}[j] = \star$. Such an edge indicates that r and r' must receive distinct layer i codes, in other words, they must

[3] In Figure 3, j increases from left to right.

be in the same sub-layer of layer i. Let k denote the number of connected components in G_{L_i}. We denote the sub-layers of L_i by $L_i^1, \ldots, L_i^k$.

To illustrate the sub-layering process, consider Layer 3 in Figure 4(a). The following vectors are associated with ranges d-g:

$$\begin{array}{lcl} v_d & = & [a, \quad b, \quad \perp, \quad 0] \\ v_e & = & [a, \quad b, \quad \perp, \quad h] \\ v_f & = & [a, \quad c, \quad \perp, \quad h] \\ v_g & = & [\star, \quad 0, \quad \perp, \quad 0] \end{array}$$

In this case, the graph G_{L_3} contains no edges and therefore the resulting sub-layering is composed of four singletons, one per each of the ranges d-g. Thus, each of these ranges receives the same layer-3 code - '1'. Now, consider Figure 4(b), in which the vector associated with range g becomes $v_g = [\star, \star, \perp, 0]$, implying the existence of an edge between ranges g and d. Thus, in this example, the sub-layering of L_3 is $\{\{d, g\}, \{e\}, \{f\}\}$.

The number of bits required for the CLE encoding of layer L_i is $allocated[i] = \left\lceil \log_2 \max\{|L_i^1| + 1, ..., |L_i^k| + 1\} \right\rceil$, where k is the number of layer L_i's sub-layers. Thus, $\sum_{i=1}^{\mathcal{C}} assigned[i] - allocated[i]$ bits are "saved" by the CLE encoding.

CLE encoding and bit assignment are combined as follows. For each number of bits b starting from the number of available extra bits e and up to the number of bits required to encode *all* ranges a, the bit-auction algorithm is performed assuming b extra bits are available and then CLE encoding is applied based on the corresponding bit assignment. We call the resulting encoding *CLE encoding #b*. Observe that CLE encoding #b may require less than b bits. If CLE encoding #b requires at most e bits, we say that it is *feasible*. We use CLE encoding #b', where $b' \in \{e, \ldots, a\}$ is the maximum such that CLE encoding b' is feasible. We note that other heuristics for combining CLE encoding and bit assignment are possible. However, based on our evaluation, the aforementioned combination strikes a good balance between time-complexity and code efficiency.

6.2 The CLE Encoding Stage

CLE encodes each sub-layer separately. When i bits are allocated to a layer L, then $2^i - 1$ ranges *in each of L's sub-layers* can be encoded, whereas, with LIC, only $2^i - 1$ ranges are encoded *in the entire layer*. For instance, a single bit suffices for a CLE encoding of all the ranges of Layer 3 in the example of Figure 4(a), and two bits are required in the example in Figure 4(b), whereas LIC requires three bits to encode Layer 3 in both cases.

The encoding is done as follows. Let $L_i^j = \{r_1, r_2, \ldots, r_{|L_i^j|}\}$ be the j'th sub-layer of layer i. The *local code* of range $r_m \in L_i^j$ is simply bin(m) The code of range r is constructed by concatenating the local codes of all the elements of v_r. More specifically, if $v_r[j]$ is a range identifier, we concatenate the local code of that range; if $v_r[j] = 0$ or $v_r[j] = \star$, we concatenate $0^{allocated[j]}$ or $\star^{allocated[j]}$, respectively; if $v_r[j] = \perp$ (that is, $i = j$), we concatenate r's local code.

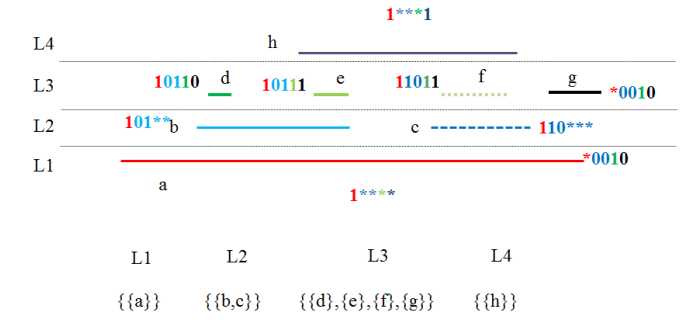
For example, in Figure 4(a), the local codes of ranges a, d, e, f, g, and h are all 1, while the local codes of b and c are 01 and 10, respectively. Therefore, g's encoding is $\star 0010$, while the encoding of d is 10110. As another example, in Figure 4(b), the local codes of d, e, and f are all 01 and the local code of g is 10 (all other codes do not change), hence g's code is $\star\star\star 100$ and d's code is 101010.

Finally, as done in LIC's encoding scheme, the search keys' extra bits are constructed by performing a bitwise OR of the vector 0^b with the codes of all the ranges that contain the search key.

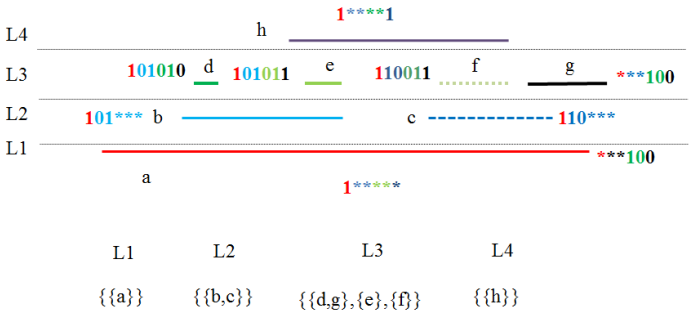
The correctness of the CLE encoding scheme is established by the following theorem:

Theorem 1 *The extra bits of a search key p match the code assigned to range r if and only if $p \in r$.*

Proof: The " if" direction follows immediately from the construction of p's extra bits as a bitwise OR of the vector 0^b with the codes of all the ranges that contain p.



(a) CLE encodes Layer 3 with 1 bit.



(b) CLE encodes Layer 3 with 2 bits.

Figure 4: Two examples of the CLE encoding. In both examples, LIC requires 3 bits for Layer 3.

As for the "only if" direction, suppose by way of contradiction that p's extra bits match the code of some range r such that $p \notin r$, and let L_i be r's layer. If there is no range $r' \in L_i$ such that $p \in r'$, then p's extra bits corresponding to layer L_i are set by the CLE encoding to 0, while r's local code is at least 1. This is a contradiction.

Therefore, there must be a range $r' \in L_i$ such that $p \in r'$. It follows that p's extra bits match the code of r'. This implies, in turn, that r and r' have the same local codes in L_i and they are therefore in different sub-layers of L_i. But since p's extra bits match the full codes of both r and r', the local codes of r and r' match in *every* layer . By the construction of G_{L_i}, this implies that there is an edge between r and r' in G_{L_i} and therefore these ranges are in the same sub-layer of L_i, which is a contradiction.

7 Experimental Results

We evaluate the performance of the LIC scheme and its CLE variant on a real-life database, which is a collection of 120 separate rule files originating from various applications (such as firewalls, acl-routers and intrusion prevention systems) collected over the year 2004. This is the same database that was used by [12, 9, 3].

Our database consists of a total of approximately $223,000$ rules that contain 280 unique ranges. Approximately 28% of the rules contain range fields and about half of these include the range $[1024, 2^{16} - 1]$. When applied to this database, prefix expansion results in an expansion factor of 2.68 and redundancy factor of 6.53 (see Table 1).

Table 1 shows the expansion and redundancy factors obtained by prior art and by our algorithms when 36 extra bits are available, which is the common case in contemporary IPv4 TCAM classifiers. When all extra bits are exhausted, our LIC scheme can use any independent encoding algorithm as a fall-back scheme. We evaluated

Algorithm	Expansion	Redundancy
Prefix Expansion [2]	2.68	6.53
Region Partitioning [7]	1.64	2.51
hybrid DIRPE [9]	1.2	-
hybrid SRGE [12]	1.03	1.2
DRES [14]	1.025	0.09
LIC/MSIS, Prefix Expansion	1.0076	0.029
LIC/MSIS, SRGE	1.0061	0.024
LIC/MSCS, Prefix Expansion	1.0088	0.034
LIC/MSCS, SRGE	1.0075	0.029
LIC/MWIS ,Prefix Expansion	1.0089	0.034
LIC/MWIS, SRGE	1.0074	0.029
LIC/MWCS, Prefix Expansion	1.0088	0.034
LIC/MWCS, SRGE	1.0074	0.029
CLE/MSIS, Prefix expansion	1.0000744	0.0002888

Table 1: Expansion and redundancy factors, using 36 extra bits, for different range encoding algorithms.

the LIC scheme using two fall-back schemes: (binary) prefix expansion and SRGE [12]. The SRGE algorithm is a database-independent encoding scheme that encodes ranges using binary-reflected Gray code. We observe that the expansion factor is always smaller than the redundancy factor, since the latter quantifies the average redundancy per range rules, whereas the former quantifies the relative increase in the *total* number of TCAM entries required for representing the database.

It is clear from the data of Table 1 that both the LIC and the CLE algorithms, regardless of the fall-back scheme they use, obtain smaller expansion and redundancy factors as compared to prior art algorithms. Specifically, the redundancy factor obtained by all LIC algorithms is smaller by more than 62% as compared to DRES. It can also be seen that using SRGE as a fall-back scheme for LIC reduces redundancy as compared to using (binary) prefix expansion. SRGE saves 17.2% in redundancy when used in conjunction with LIC/MSIS and 14.7% when used in conjunction with LIC/MSCS, LIC/MWIS or LIC/MWCS as compared to a binary prefix expansion fall-back scheme. Finally, when 36 bits are available, CLE used in conjunction with MSIS remarkably reduces redundancy by at least two orders of magnitude as compared with all other algorithms.

In absolute numbers, our algorithms reduce database expansion by approximately $2\% - 2.5\%$ as compared with best prior art [14]. As the number of rules in packet classifiers is constantly and rapidly increasing [21], and as the real-life database we have access to is from 2004, we estimate that this figure is significantly higher for contemporary classification databases.

Table 2 shows the number of bits required to encode all the ranges that occur in our database when using different encoding schemes. Liu's algorithm [7] requires 235 bits, since a single bit must be used for every range with expansion larger than 1. All variants of the LIC scheme achieve a very significant decrease in the number of required extra bits as compared to the Liu algorithm. Surprisingly, LIC with the simple MSIS layering algorithm achieves the best result - only 85 bits. This is, in fact, the optimal solution of the MLIC problem on our database. Note that the DRES algorithm is not shown in Table 2 since, when the number of extra bits is not restricted, it degenerates to the Liu algorithm. Finally, the CLE variant of LIC reduces the number of bits by approximately 20% as compared with LIC, under all layering algorithms. CLE used in conjunction with MSCS achieves the best result and requires only 67 extra bits to encode all database ranges.

It can be observed from the data of Table 2 that when the number of extra bits is not restricted (i.e. when we solve the MLIC problem rather than the BMLIC problem), the un-weighted layering algorithms (that is, MSIS and MSCS) are superior to the weighted layering algorithms (MWIS and MWCS). The reason for this is the following: if all ranges can be encoded, then there is no use in giving higher-weight ranges precedence when constructing layers. When solving MLIC, the layering algorithm should *ignore* weights; taking weights into consideration will, in general, increase the number of bits eventually used.

Figure 5 compares the redundancy factor of different range encoding algorithms as a function of the number

Layering Algorithm	Number of Required Bits Without CLE	Number of Required Bits With CLE
LIU	235	N/A
LIC/MWCS	96	76
LIC/MWIS	94	72
LIC/MSCS	86	67
LIC/MSIS	85	68
Optimal MLIC Solution	85	N/A

Table 2: The number of bits required to encode all the ranges that occur in our database as a function of the encoding scheme employed.

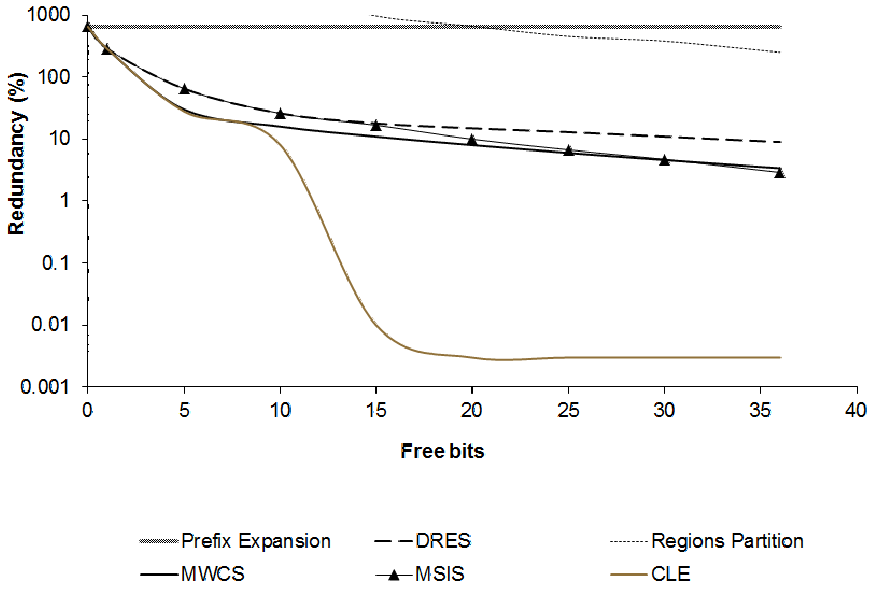


Figure 5: Redundancy factor as a function of the number of extra bits for different encoding schemes.

of available extra bits. Since the curves of LIC/MSIS and LIC/MWCS are almost identical to those of LIC/MSCS and LIC/MWIS, we only show the former curves in Figure 5. Regardless of the number of extra-bits available, our scheme outperforms all prior-art dependent range encoding algorithms. LIC, using either one of the four layering algorithms we analyze, reduces the redundancy factor by 50%-70% (depending on the number of available extra bits) as compared with DRES [14], which is the best prior art range encoding algorithm. Specifically, when 36 extra bits are available (as is typical for IPv4 TCAM-based classification databases), our scheme (using any of the 4 layering algorithms we described) reduces the redundancy factor as compared to DRES by either 62.2% or 67.8%, depending on whether the fall-back scheme used is prefix expansion or Gray coding, respectively. Note that the region partitioning algorithm [7] improves over prefix expansion only when the number of extra bits approaches 20 and even then achieves only a minor improvement of the redundancy factor. The CLE variant of LIC yields a remarkable further reduction of the redundancy factor by two order of magnitude when the number of extra bits exceeds 15.

Figure 6 shows the redundancy factor obtained by LIC when used with the different layering algorithms we analyze. It can be seen that the weighted layering algorithms (MWIS and MWCS) are superior when the number of extra bits is 34 or less but the un-weighted algorithms fare better when the number of extra bits exceeds 34. As mentioned before, the benefit gained from giving precedence to high-weight ranges in the layering process decreases when the number of extra bits increases.

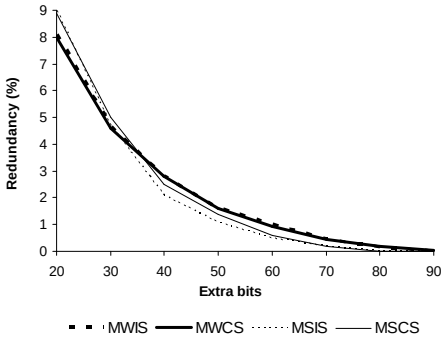


Figure 6: Redundancy factor as a function of the number of extra bits for different layering algorithms.

8 Theoretical Results

We start by proving that the MLIC problem, formally defined in Section 3.2, is NP-hard. MLIC is similar to the chromatic sum problem [6, 30]. Marx [31] provided a short and elegant proof that chromatic sum is NP-hard on interval graphs. We employ similar ideas to prove the NP-hardness of MLIC.

8.1 Hardness Results

A graph G is a *circular arc graph* if it is the intersection graph of arcs on a circle. In other words, the vertices of G have a one-to-one correspondence with a set of arcs on a circle so that two vertices in G are adjacent in G if and only if their corresponding arcs intersect each other.

Given a circular graph G and an integer k, the *circular graph coloring problem* is to decide whether G can be colored by at most k colors. We prove the NP-hardness of MLIC by a reduction from this problem, whose NP-hardness was established in [4].

Theorem 2 *The Minimum space Layered Interval-Code (MLIC) problem is NP-hard.*

Proof: Let G be a circular arc graph with n nodes (each representing an arc) and let k be an integer. The circular graph coloring problem is to decide whether G can be colored using k colors. It can be assumed that the arcs represented by G are open: two arcs that share only an endpoint are assumed not to intersect each other. Let x be an arbitrary point on the circle that is not the endpoint of any of the arcs. If x is contained in more than k arcs, then a polynomial-time test can determine that G cannot be colored with k colors. Hence, it can be assumed that x is contained in exactly k arcs, since if x is contained only in the arcs $a_1, \cdots, a_{k'}$, for $k' < k$, then we can add $k - k'$ sufficiently small arcs that intersect only $a_1, \cdots, a_{k'}$. Clearly, this cannot increase the chromatic number of G beyond k.

Let $a_1, \cdots, a_k$ be the arcs that contain x (see Figure 7). Split each arc a_i into two parts l_i and r_i at point x. Let x be the clockwise (respectively counter-clockwise) endpoint of l_i (respectively r_i) and let G' denote the resulting graph. G' is an interval graph, since x is not contained in any interval of G'. It follows that G' has an interval representation where the left endpoint of each interval l_i is 0, the right endpoint of each r_i is some positive M, and no interval extends to the left of 0 or to the right of M. We can modify the left endpoint of l_i to $-k + i$ and the right endpoint of r_i to $M + k - i$ without changing G'. See Figure 8.

Let $R = k^7(n+k+1)$ and for every $i \in [k]$ define $R_i = (k-i+1)R$. Let S_x be a set of $R/2$ equal-sized and contiguously-extending intervals, each of length $2/R$, such that $\inf\{I \in S_x\} = x$ and $\sup\{I \in S_x\} = x + 1$; that is, S_x consists of $R/2$ disjoint intervals that cover $(x, x + 1)$.

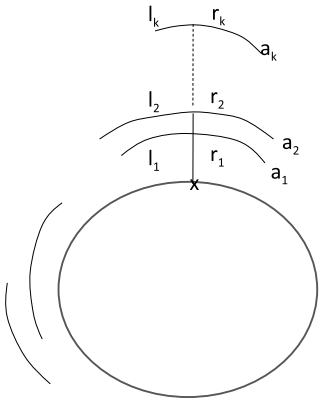


Figure 7: The circular arc underlying graph G.

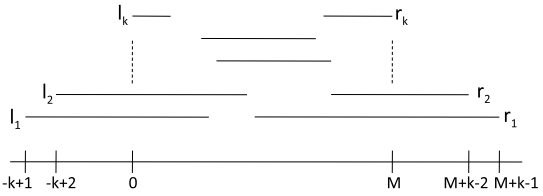


Figure 8: The intervals underlying graph G'.

We proceed with constructing another interval graph denoted G''. G'' is constructed by adding the following sets of intervals in addition to those represented by graph G': for each $i \in \{-k, \ldots, -1\}$ we add $-i$ copies of the set S_i. Similarly, for each $j \in \{M, \ldots, M+k-1\}$, we add $j - M + 1$ copies of the set S_j. We denote these sets of intervals by SL and SR, respectively. See Figure 9.

Lemma 3 *A circular arc graph G can be colored with k colors if the corresponding graph G'' has a coloring with LIC-code size less than $B = 1/k^5 + \sum_{i=1}^{k} \log_2 R_i$.*

Proof:

The proof will be derived from the following two claims.

Claim 4 *If a coloring C'' of graph G'' has LIC-code size less than B and uses k colors, then C'' assigns the same color to intervals l_i and r_i, for every $1 \leq i \leq k-1$.*

Proof: We focus on the intervals of SL. Without loss of generality, assume that l_i is colored by color i under C'', for all $i \in \{1, \ldots, k\}$. Let $\overline{V}$ be a vector of length k such that V_i is the number of intervals in SL that were assigned the color i. We next show that $\overline{V} = \frac{R}{2} \cdot < 1, 2, \ldots, k >$.

Consider interval $(-1, 0)$. This interval intersects all intervals $l_1, \ldots, l_{k-1}$, implying that the entire single copy of interval set S_{-1} (which consists of $R/2$ intervals) must be colored by color k. Similarly, interval $(-2, -1)$ intersects all intervals $l_1, \ldots, l_{k-2}$, implying that all the intervals of the two copies of S_{-2} must be colored by either $k-1$ or k. Since each interval in one copy of S_{-2} intersects exactly one interval in the second copy, this implies that $R/2$ intervals must be colored by $k1$ and $R/2$ intervals must be colored by k. Continuing in this manner, it follows by induction that, for every $i \in [k]$, a total of $i \cdot R/2$ intervals will be colored by color i, implying that $\overline{V} = \frac{R}{2} \cdot < 1, 2, \ldots, k >$ holds.

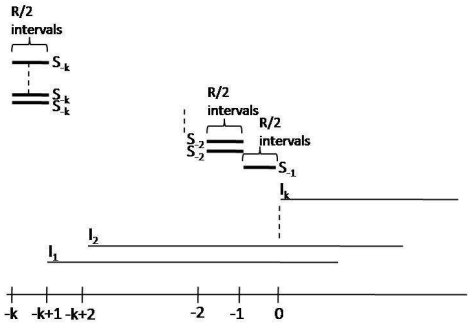


Figure 9: The intervals underlying graph G''. (Only intervals l_i and the intervals of SL are shown.)

Analogously, we let $\overline{V'}$ denote the vector of length k such that V'_i is the number of intervals in SR that were assigned the color i. Observe that, while we have chosen color name i for each interval l_i, we cannot do the same for intervals r_i. However, it follows by reasoning similar to that applied for SL that $\frac{2}{R}\overline{V'}$ is a permutation over $\{1,\ldots,k\}$. We proceed to show that $\overline{V'} = \overline{V}$. We use the following fact, whose proof appears in the appendix:

Fact 1 *For any constant $c \geq 1$, the permutation π over $\{1,\ldots,k\}$ which minimizes the function $f(\pi) = \sum_{i=1}^{k} \log(c(\pi_i + i))$ is the identity permutation. The second smallest value of is obtained by f on the permutation $\langle 1,2,\ldots,k-2,k,k-1\rangle$.*

For a coloring C of G'', we let $L_S(C)$ denote the LIC-code size induced by C computed only over the nodes of G'' that represent intervals of S_L or S_R. By Fact 1, L_S is minimized when $\overline{V'} = \overline{V}$. Furthermore, the second smallest value of L_S is obtained when $\overline{V'} = \frac{R}{2}\langle 1,2,\ldots,k-2,k,k-1\rangle$. We next show that any coloring induced by this vector, denoted $C_{\langle 1,2,\ldots,k-2,k,k-1\rangle}$, results in LIC-code size greater than B:

$$\begin{aligned}
L(G'', C_{\langle 1,2,\ldots,k-2,k,k-1\rangle}) &> L_S(C_{\langle 1,2,\ldots,k-2,k,k-1\rangle}) \\
&\geq \sum_{i=3}^{k} \log R_i + 2\log\left(\frac{R_1+R_2}{2}\right) \\
&= \sum_{i=1}^{k} \log R_i + 2\log\left(\frac{R_1+R_2}{2}\right) - (\log R_1 + \log R_2) \\
&= \sum_{i=1}^{k} \log R_i + \log \frac{\left(\frac{R_1+R_2}{2}\right)^2}{R_1 \cdot R_2}
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{i=1}^{k} \log R_i + \log \frac{\left(\frac{kR+(k-1)R}{2}\right)^2}{kR \cdot (k-1)R} \\
&= \sum_{i=1}^{k} \log R_i + \log \frac{\left(k-\frac{1}{2}\right)^2}{k(k-1)} \\
&= \sum_{i=1}^{k} \log R_i + \log \left(1 + \frac{1}{4(k^2-k)}\right) \\
&> \sum_{i=1}^{k} \log R_i + \frac{\log e}{\left(4(k^2-k)\right)^2} \\
&\quad \text{Since } \ln(1+x) > x - \tfrac{x^2}{2} > x^2 \text{ for every } x \in (0, \tfrac{2}{3}). \\
&> \sum_{i=1}^{k} \log R_i + \frac{1}{k^5} \quad = B.
\end{aligned}
$$

This implies that C'' colors SL and SR according to the same vector. Recall that the i-th element in $\overline{V}$ corresponds to l_i's color, while the i-th element in $\overline{V'}$ corresponds to r_i's color. Since $\overline{V} = \overline{V'}$, it follows that for every i, C'' colors l_i and r_i with the same color.

Claim 5 *If a coloring $\mathtt{C}''$ of graph G'' has LIC-code size less than B then it uses exactly k colors.*

Proof: First note that the k copies of the set S_{-k} intersect at $-k+1/2$, implying that $\mathcal{C}''$ uses at least k colors. We now focus on $G'' \setminus G = SL \cup SR$ and show that $\mathtt{C}''$ uses at most k colors for these intervals.

Assume towards a contradiction that $\mathtt{C}''$ uses $\ell > k$ colors for $G'' \setminus G$. We first show that the LIC-code size can be decreased by altering $\mathtt{C}''$ to use $\ell - 1$ colors. Denote the *least-used* color by x and let I be an arbitrary interval with color x. Since I intersects at most $k-1$ other intervals, it can be legally colored in another color y such that $n_y > n_x$. Since the logarithm function is concave, this strictly reduces the LIC-code size. Repeatedly applying this procedure to all the intervals of N_x results in a coloring with $\ell - 1$ colors and strictly reduces the LIC-code size.

Consider all colorings with $k+1$ colors and, without loss of generality, denote the least-used color by $k+1$. It follows that a coloring $\mathtt{C}$ which minimizes the LIC-code size among all these colorings must assign the color $k+1$ to a *single* interval I. Observe that re-coloring interval I to a color in $\{1, \ldots, k\}$ reduces the LIC-code size by at least $1 - \log\left(\frac{3}{2}\right) = 0.4150$.

From Claim 4, $SL \cup SR$ contribute at least $\sum_{i=1}^{k} \log R_i$ to the LIC-code size even under the minimal coloring with k colors. It follows that $SL \cup SR$ contributes at least $0.4150 + \sum_{i=1}^{k} \log R_i > 1/k^5 + \sum_{i=1}^{k} \log R_i = B$ to the LIC-code size in every coloring with more than k colors, . This implies in turn that $\mathtt{C}''$ uses at most k colors for $G'' \setminus G = SL \cup SR$.

We conclude the proof by showing that it is not possible that $\mathtt{C}''$ uses more than k colors for graph $G'' = (SL \cup SR) \cup G$. Since $SL \cup SR$ is colored by k colors, it contributes at least $\sum_{i=1}^{k} \log R_i$ to the LIC-code size. Any additional color will add at least 1 to the code size and it will surpass B. Hence, $\mathtt{C}''$ uses exactly k colors.

The proof of Lemma 3 can now be derived as follows: from Claim 5, a coloring of G'' with LIC-code size less than B uses exactly k colors and, from Claim 4, it assigns the same color to intervals l_i and r_i, for every $i \in [k]$. Thus, it induces a legal coloring of G that uses k colors. The 'if' direction follows.

We now prove that the inverse of Lemma 3 is also true.

Lemma 6 *If a circular arc graph G can be colored with k colors, then the corresponding graph G'' has a coloring with LIC-code size less than B.*

Proof:

The proof relies on the following fact, whose proof is brought in the appendix.

Fact 2 *If $R = k^7(n+k+1)$ and, for every $i \in [k]$, $R_i = (k-i+1)R$, then, for every $k \geq 3$ and $i \in [k]$, $\log(R_i + (n+k+1)) - \log(R_i) < \frac{1}{k^6}$.*

Assume that G can be colored with k colors. Let $\mathcal{C}$ be that coloring and let i denote the color of a_i under $\mathcal{C}$. $\mathcal{C}$ can be extended to a coloring $\mathcal{C}''$ of G'' as follows: For every $1 \leq i \leq k$, assign color i to the intervals l_i and r_i, to the $i \cdot R/2$ intervals in $G'' \setminus G'$ that contiguously cover $(-k, -k+i)$, and to the $i \cdot R/2$ intervals in $G'' \setminus G'$ that contiguously cover $(M+k-i, M+k)$.

From our selection of R_k, and since $R_i > R_k$ for every $1 \leq i \leq k-1$, the following holds regardless of how the intervals of G are partitioned between the k colors.

$$L(\mathbb{G}'', \mathbb{C}'') \;<\; \sum_{i=1}^{k} \log(R_i + n + k + 1) \;<\; \sum_{i=1}^{k} \left(\log(R_i) + \frac{1}{k^6} \right) \;=\; \frac{1}{k^5} + \sum_{i=1}^{k} \log(R_i) = B,$$

where the second inequality follows from Fact 2.

The proof of Theorem 2 now follows from Lemmas 3, 6 and from the NP-hardness of the circular graph coloring problem [4].

The hardness of the BMLIC problem follows by reduction from the MLIC problem:

Theorem 7 *The Budgeted Minimum Layered Interval-Code problem is NP-hard.*

Proof: Let $G = \langle V, E \rangle$ be an instance of the MLIC problem. We show that the minimum LIC-code of G can be found in polynomial time by an oracle machine with an oracle for the BMLIC problem.

This is done by iteratively running the BMLIC oracle with input G and increasing b by 1 in every iteration (starting with $b = 1$). The algorithm stops when the oracle first returns G (that is, the graph G can be colored completely with an interval code size exactly b). The solution to the MLIC problem is therefore the coloring $\mathcal{C}$ returned by the oracle (and the optimal MLIC value is b).

Since a coloring $\mathcal{C}'$ of G that colors each vertex $v \in V$ with a unique color is always feasible and satisfies $L(G, \mathcal{C}') = |V|$, the optimal MLIC value is always bounded by $|V|$. Therefore, the algorithm described above runs in polynomial time and the claim follows.

8.2 A 2-approximation Algorithm for the MLIC Problem

We next show that the MSCS algorithm presented in Section 3.2 (see Algorithm 1 for the pseudo-code) is a polynomial 2-approximation algorithm for the MLIC problem. The algorithm and the proof follow closely the 2-approximation algorithm for *sum coloring on interval graph* (and its proof) presented in [5].

Theorem 8 *Algorithm MSCS is a polynomial 2-approximation algorithm for the MLIC problem.*

Proof: Let G be an instance of the MLIC problem, and let $\mathcal{C}^*$ be the optimal solution for G (i.e., $L(G, \mathcal{C}^*)$ is minimal among all legal colorings of G). We denote by $\chi(G)$ the chromatic number of G. Note that $|\mathcal{C}^*| \geq \chi(G)$.

We first present an *illegal* coloring $\mathcal{C}_1$, such that $L(G,\mathcal{C}_1) \leq L(G,\mathcal{C}^*)$. Then, we show how to correct the coloring C_1 to a legal coloring $\mathcal{C}_2$ for which $L(G,\mathcal{C}_2) < 2 \cdot L(G,\mathcal{C}_1)$. This implies that $L(G,\mathcal{C}_2) < 2 \cdot L(G,\mathcal{C}^*)$ and $\mathcal{C}_2$ is a 2-approximation for the problem.

Coloring $\mathcal{C}_1$ is obtained by running Algorithm 1 without the recursive calls (that is, Lines 1-9). $\mathcal{C}_1$ assigns color k to $A_k \setminus A_{k-1}$ for $k = 1, \ldots, \chi(G)$ (for convention, $A_0 = \emptyset$). Let n_i be the number of vertices assigned color i by $\mathcal{C}_1$.

We assume ALGORITHM LB [5, pp. 114-115] is used to compute the k-colorable sets in Line 5. This implies that the sets A_i have the following properties that were proved in [5]:

Property 1 *For every $i \in \{1, \ldots, \chi(G)-1\}$, $n_i \geq n_{i+1}$.*

Property 2 *For every $k \in \{2, \ldots, \chi(G)\}$, the graph induced by $A_k \setminus A_{k-1}$ is a 2-colorable interval graph.*

We next show that $L(G,\mathcal{C}_1) \leq L(G,\mathcal{C}^*)$.

Denote by n_i^* the number of vertices assigned color i by $\mathcal{C}^*$. Without loss of generality we assume that $n_i^* \geq n_{i+1}^*$ for every $i \in \{1, \ldots, |\mathcal{C}^*|\}$. Since $\mathcal{C}^*$ is a legal coloring, the number of vertices colored with k colors cannot exceed the size of largest k-colorable subgraph of G. This implies that $\mathcal{C}^*$ must satisfy the following constraints:

$$\sum_{i=1}^{k} n_i^* \leq |A_k|, \qquad \text{for every } k \in \{1, \ldots, \chi(G)\}.$$

It is important to notice that among all (not necessarily legal) colorings satisfying these constraints, $\mathcal{C}_1$ has minimum LIC-code size: Since the logarithm function is concave, re-coloring a vertex from color i to color j always reduces the LIC-code size if $m_i \leq m_j$. This implies that the minimal code size is achieved when $\sum_{i=1}^{k} m_i = |A_k|$ for every $k \in \{1, \ldots, \chi(G)\}$ (that is, $n_k = |A_k| - |A_{k-1}|$). Since $\mathcal{C}_1$ maintains this property, it implies that $L(G,\mathcal{C}_1) \leq L(G,\mathcal{C}^*)$.

Coloring $\mathcal{C}_2$ is the legal coloring obtained by Algorithm MSCS (Lines 1-15). Let $A_1, \ldots, A_k$ be the sets obtained in Line 5 of the top recursion level. By Property 2, each subgraph GRAPH$(G, A_k \setminus A_{k-1})$ is 2-colorable. This implies that the recursion depth is at most 2 and that each set A_k can be partitioned into two independent sets, denoted odd_k and even_k (for $k = 1$, $\chi(G'_1) = 1$ implying that $\text{odd}_1 = \emptyset$). Note that $\{\text{odd}_k, \text{even}_k\}$ is the result of applying MSCS on GRAPH$(G, A_k \setminus A_{k-1})$.

The legal coloring $\mathcal{C}_2$ is defined as follows:

$$\mathcal{C}_2(v) = \begin{cases} 2 \cdot \mathcal{C}_1(v) - 1 & \text{if } v \in \text{odd}_{\mathcal{C}_1(v)} \\ 2 \cdot \mathcal{C}_1(v) & \text{if } v \in \text{even}_{\mathcal{C}_1(v)} \end{cases}$$

Intuitively, the conflicts of $\mathcal{C}_1$ are resolved independently for each color; it suffices to add an extra color (for each color) in order to resolve all conflicts. It follows that $\mathcal{C}_2$ uses at most $2\chi(G) - 1$ colors (since odd_1 is always empty). Let n_i'' denote the number of vertices assigned with color i under $\mathcal{C}_2$.

We next bound the LIC-code size of C_2:

$$\begin{aligned} L(G,\mathcal{C}_2) &= \sum_{j=1}^{2\chi(G)} \log(n_j'' + 1) = \sum_{i=1}^{\chi(G)} \left(\log(n_{2i-1}'' + 1) + \log(n_{2i}'' + 1)\right) \\ &= \sum_{i=1}^{\chi(G)} \left(\log(|\text{odd}_i| + 1) + \log(|\text{even}_i| + 1)\right) < 2 \cdot \sum_{i=1}^{\chi(G)} \log(|A_i| + 1) \\ &= 2 \cdot L(G,\mathcal{C}_1) \leq 2 \cdot L(G,\mathcal{C}^*) \end{aligned}$$

Hence, this perturbation of $\mathcal{C}_1$ at most doubles the LIC-code size and the claim follows.

9 Conclusions

This paper presents the *Layered Interval Code* (LIC) range-encoding scheme for constructing space-efficient TCAM representations of range rules. The gist of our scheme is the use of graph-coloring algorithms to partition the ranges between multiple *layers*. Each of these layers consists of mutually disjoint ranges that are encoded by using the extra-bits that are typically available in TCAM-based classifier entries. We present several LIC construction algorithms that are based on approximation algorithms for specific variants of interval-graph coloring. We also evaluate two encoding schemes, one which encodes each layer independently and another - cross-layer encoding - which exploits containment relationships between intervals in different layers. We suggest a novel hot updates algorithm (used by our LIC scheme) that can be used with any database-dependent encoding scheme. We evaluate these algorithms by performing extensive comparative analysis on real-life classification databases. Our analysis establishes that all our algorithms reduce the number of redundant TCAM entries caused by range rules by more than 60% as compared with best range-encoding prior art.

Acknowledgments The authors are deeply indebted to Cisco Systems, Will Eatherton, and David Taylor for kindly providing us the classification database we used for this work. We would also like to thank Yehuda Afek, Karthik Lakshminarayanan, Anand Rangarajan , and Srinivasan Venkatachary for assisting us in the process of obtaining access to the database. Finally, we thank the anonymous reviewers for their many helpful comments.

References

[1] *INFOCOM 2009. 28th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 19-25 April 2009, Rio de Janeiro, Brazil*. IEEE, 2009.

[2] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast and scalable layer four switching," in *ACM SIGCOMM*, Sep. 1998, pp. 191–202.

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

[4] M. R. Garey, D. S. Johnson, G. L. Miller, and C. H. Papadimitriou, "The complexity of coloring circular arcs and chords," *SIAM Journal on Algebraic and Discrete Methods*, vol. 2, no. 1, pp. 216–227, 1980.

[5] S. Nicoloso, M. Sarrafzadeh, and X. Song, "On the sum coloring problem on interval graph," *Algorithmica*, vol. 23, pp. 109–126, 1999.

[6] E. Kubicka, "The chromatic sum of a graph," Ph.D. dissertation, Western Michigan University, 1989.

[7] H. Liu, "Efficient mapping of range classifier into ternary-cam," in *Hot Interconnects*, 2002.

[8] J. van Lunteren and T. Engbersen, "Fast and scalable packet classification," *JSAC*, 2003.

[9] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary, "Algorithms for advanced packet classification with ternary CAMs," in *ACM SIGCOMM*, 2005.

[10] F. Yu and R. H. Katz, "Efficient multi-match packet classification with tcam," in *HOTI*, 2004.

[11] E. Spitznagel, D. Taylor, and J. Turner, "Packet classification using extended TCAMs," in *ICNP*, 2003.

[12] A. Bremler-Barr and D. Hendler, "Space-efficient tcam-based classification using gray coding," in *IEEE INFOCOM*, 2007, pp. 1388–1396.

[13] F. Gray, "Pulse code communication," 1953, united States Patent Number 2632058.

[14] H. Che, Z. Wang, K. Zheng, and B. Liu, "DRES: Dynamic range encoding scheme for tcam coprocessors," *IEEE Transaction on Computers*, vol. 57, no. 6, 2008.

[15] Y.-K. Chang and C.-C. Su, "Efficient tcam encoding schemes for packet classification using gray code," in *Global Telecommunications Conference, GLOBECOM*, 2007.

[16] K. Zheng, H. Che, Z. Wang, B. Liu, and X. Zhang, "DPPC-RE: TCAM-based distributed parallel packet classification with range encoding," *IEEE Trans. Computers*, vol. 55, no. 8, pp. 947–961, 2006.

[17] R. Cohen and D. Raz, "Simple efficient tcam based range classification." in *INFOCOM*. IEEE, 2010, pp. 461–465. [Online]. Available: http://dblp.uni-trier.de/db/conf/infocom/infocom2010.html#CohenR10

[18] O. Rottenstreich and I. Keslassy, "Worst-case tcam rule expansion," in *Proceedings of the 29th conference on Information communications*, ser. INFOCOM'10. Piscataway, NJ, USA: IEEE Press, 2010, pp. 456–460. [Online]. Available: http://dl.acm.org/citation.cfm?id=1833515.1833607

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

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

[21] C. R. Meiners, A. X. Liu, and E. Torng, "TCAM razor: A systematic approach towards minimizing packet classifiers in TCAMs," in *ICNP*, 2007.

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

[23] D. Pao, Y. K. Li, and P. Zhou, "Efficient packet classification using TCAMs," *Comput. Netw.*, vol. 50, pp. 3523–3535, December 2006. [Online]. Available: http://dl.acm.org/citation.cfm?id=1228646.1228647

[24] D. C.-W. Pao, P. Zhou, B. L. 0001, and X. Zhang, "Enhanced prefix inclusion coding filter-encoding algorithm for packet classification with ternary content addressable memory," *IET Computers & Digital Techniques*, vol. 1, no. 5, pp. 572–580, 2007.

[25] C. R. Meiners, A. X. Liu, and E. Torng, "Topological transformation approaches to optimizing tcam-based packet classification systems," in *Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*, ser. SIGMETRICS '09. New York, NY, USA: ACM, 2009, pp. 73–84. [Online]. Available: http://doi.acm.org/10.1145/1555349.1555359

[26] A. Bar-Noy, M. Bellare, M. M. Halldorsson, H. Shachnai, and T. Tamir, "On chromatic sums and distributed resource allocation," *Information and Computation*, vol. 140, no. 2, pp. 183–202, 1998. [Online]. Available: citeseer.ist.psu.edu/bar-noy98chromatic.html

[27] M. C. Carlisle and E. L. Lloyd, "On the k-coloring of intervals," *Discrete Applied Mathematics*, vol. 59, no. 3, pp. 225–235, 1995.

[28] R. E. Tarjan, *Data structures and network algorithms*. Society for Industrial and Applied Mathematics, 1983.

[29] D. Shah and P. Gupta, "Fast updating algorithms for tcams," *IEEE Micro*, vol. 21, no. 1, pp. 36–47, 2001.

[30] K. J. Supowit, "Finding a maximum planar subset of nets in a channel," *IEEE Transactions on Computer-Aided Design*, vol. 6, no. 1, pp. 93–94, 1987.

[31] D. Marx, "A short proof of the NP-completeness of minimum sum interval coloring," *Operations Research Letters*, vol. 33, no. 4, pp. 382–384, 2005.

A Proofs omitted from Section 8

In this section we complete the details of the proof of Theorem 2 by providing the proofs of the following two arithmetic facts.

Fact 1: For any constant $c \geq 1$, the permutation π over $\{1, \ldots, k\}$ which minimizes the function $f(\pi) = \sum_{i=1}^{k} \log\left(c\left(\pi_i + i\right)\right)$ is the identity permutation. The second smallest value is obtained by the permutation $\langle 1, 2, \ldots, k-2, k, k-1\rangle$.

Proof: Assume π is not the identity permutation, and denote by x the largest element in π which does not equal π_x (that is $x = \max_i\{\pi_i \neq i\}$. Let y be the place of x in π (that is, $\pi_y = x$), and let $\pi_x = z$. By the choice of x, it follows that $x > y$ and $x > z$; let $y = x - c_1$, $z = x - c_2$, for some integers $c_1, c_2 > 0$

Consider the following permutation π', which is identical to permutation π, with the following perturbation: $\pi'_x = x$ and $\pi'_y = z$ (that is, in π', x was swapped with the element $\pi_x = z$ and therefore was put in its natural place). We next show that $f(\pi') < f(\pi)$:

$$\begin{aligned}
f(\pi')-f(\pi) &= \sum_{i=1}^{k} \log\left(c\left(\pi'_i + i\right)\right) - \sum_{i=1}^{k} \log\left(c\left(\pi'_i + i\right)\right) \\
&= \log\left(c\left(\pi'_x + x\right)\right) + \log\left(c\left(\pi'_y + y\right)\right) - \log\left(c\left(\pi_x + x\right)\right) - \log\left(c\left(\pi_y + y\right)\right) \\
&= \log\left(\frac{c^2(\pi'_x + x)(\pi'_y + y)}{c^2(\pi_x + x)(\pi_y + y)}\right) = \log\frac{2x(y+z)}{(x+z)(x+y)} \\
&= \log\frac{2x(2x - c_1 - c_2)}{(2x - c_1)(2x - c_2)} = \log\frac{4x^2 - 2xc_1 - 2xc_2}{4x^2 - 2xc_1 - 2xc_2 + c_1c_2} < 0
\end{aligned}$$

where the last inequality follows since $c_1 \cdot c_2 > 0$, and therefore $4x^2 - 2xc_1 - 2xc_2 < 4x^2 - 2xc_1 - 2xc_2 + c_1c_2$.

The above claim yields the following observation: For any permutation π, a series of swaps in which the highest elements is put into its natural place, yields a *strict decrease* in the function f. This immediately implies that the identity permutation minimizes the function f. Furthermore, it follows that in the second-smallest permutation only two elements are not in their natural place (that is, a single swap will turn it to the identity permutation). By similar calculations, it follows that among these permutations, the permutation which minimizes f (and hence the second-smallest permutation) is $\langle 1, \ldots, k-2, k, k-1\rangle$.

Fact 2: If $R = k^7(n+k+1)$ and, for every $i \in [k]$, $R_i = (k-i+1)R$ then for every $k \geq 3$ and $i \in [k]$, $\log(R_i + (n+k+1)) - \log(R_i) < \frac{1}{k^6}$.

Proof: The proof follows by the next sequence of inequalities:

$$\begin{aligned}
\log(R_i + (n+k+1)) - \log(R_i) &= \log\left(\frac{R_i + (n+k+1)}{R_i}\right) \\
&= \log\left(1 + \frac{n+k+1}{(k-i+1)k^7(n+k+1)}\right) \\
&= \log\left(1 + \frac{1}{(k-i+1)k^7}\right) \\
&\leq \log\left(1 + \frac{1}{k^7}\right) \\
&< \frac{\log e}{k^7} < \frac{1}{k^6},
\end{aligned}$$

where the last two inequalities are since for any $x \in (-1, 1]$, $\ln(1+x) < x$, and because $k \geq 3 > \log e$, respectively.