Encoding Short Ranges in TCAM Without Expansion: Efficient Algorithm and Applications*

Anat Bremler-Barr
The Interdisciplinary Center
Herzliya, Israel
bremler@idc.ac.il

Yotam Harchol
The Hebrew University
Jerusalem, Israel
yotamhc@cs.huji.ac.il

David Hay
The Hebrew University
Jerusalem, Israel
dhay@cs.huji.ac.il

Yacov Hel-Or
The Interdisciplinary Center
Herzliya, Israel
toky@idc.ac.il

ABSTRACT

We present RENÉ — a novel encoding scheme for short ranges on *Ternary content addressable memory* (TCAM), which, unlike previous solutions, does not impose *row expansion*, and uses bits proportionally to the maximal range length. We provide theoretical analysis to show that our encoding is the closest to the lower bound of number of bits used. In addition, we show several applications of our technique in the field of packet classification, and also, how the same technique could be used to efficiently solve other hard problems such as the *nearest-neighbor search* problem and its variants. We show that using TCAM, one could solve such problems in much higher rates than previously suggested solutions, and outperform known lower bounds in traditional memory models. We show by experiments that the translation process of RENÉ on switch hardware induces only a negligible 2.5% latency overhead. Our nearest neighbor implementation on a TCAM device provides search rates that are up to four orders of magnitude higher than previous best prior-art solutions.

1. INTRODUCTION

Ternary content addressable memories (TCAMs) have become highly popular in networking equipment and network processing units. TCAMs are used for high-speed IP lookup and packet classification in switches and routers [24, 45]. Software defined networking (SDN) schemes such as OpenFlow [39] rely on TCAM as the main hardware for their data path. TCAM was also suggested to be used for other computationally intensive tasks such as pattern matching [9, 19], heavy-hitters detection [35], and similarity search in databases [54].

*A preliminary version of this paper was presented in DaMoN 2015 workshop.

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

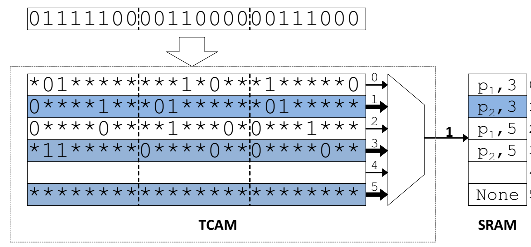

Figure 1: Diagram of the TCAM lookup process. The query is compared to all entries in parallel and the index of the first matching entry is used to find the result.

TCAM is an associative memory module. It is composed of an array of ternary words, each consisting of ternary digits, namely: 0, 1, or *. The '*' digits serve as 'wild cards' that can be matched with either '0' or '1'. Given a query word, TCAM returns the first location in the memory array that matches the query. This process is illustrated in Figure 1.

Multi-field packet classification is becoming more and more important in modern network architectures, such as SDN and *network function virtualization* (NFV) [17]. Specifically, recently suggested SDN frameworks perform more network functionalities on switches, such as load balancing [58], DDoS prevention [43], and quality of service (QoS) [53]. The initiative for NFV suggests to implement higher level tasks such as deep packet inspection and caching as virtual software services, and make traffic flow through them using smart classification rules. All such frameworks heavily rely on multi-field packet classification. Many of these fields are better expressed as ranges.

While TCAMs become more and more popular, it is still a hard problem to efficiently represent range rules on such memories. Over the last decade there has been an intense line of research on range encoding on TCAM [8, 10, 12, 13, 16, 27, 29, 31, 33, 49, 55–57]. Aside from propositions to rearchitect TCAM devices to natively support range rules [55], these solutions can roughly be classified as either *database-independent* or *database-dependent* encoding schemes. Database-independent schemes encode all possible ranges using the same technique, thus allowing fast *hot updates* [10, 27, 56]. However, these schemes use exponential TCAM row expan-

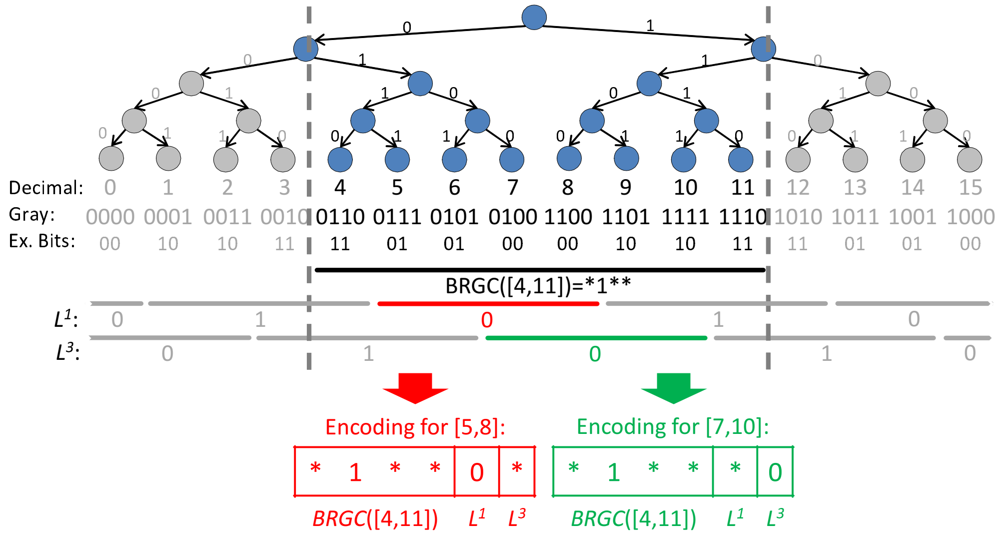

Figure 2: Toy example: A binary-reflected Gray code (BRGC) encoding tree and the encoding of ranges $[5, 8]$ and $[7, 10]$ using our scheme. In this example, maximal range length is 4. Ranges are divided into layers of non-overlapping ranges. Two layers contain only ranges that can be encoded using Gray code and therefore are not shown. Containing range $[4, 11]$ is encoded based on the BRGC values, which forms the first left four bits. Extra bits correspond to layers: Fifth bit to layer L^1 and sixth bit to layer L^3. If a range belongs to a layer, then the value of the bit corresponding to that layer is the binary value of the range for this layer. Otherwise, that bit is set to '$*$'. The total number of bits is proportional to the maximal encoded length and is independent of the number of encoded ranges.

sion, where a row is expanded into several rows, exponentially to the number of range fields in it.

Database-dependent schemes trades the hot updates flexibility for more compact codes [8, 13], but usually performs well only when the number of encoded ranges is small, as the produced code is proportional to the number of ranges in database. Database-dependent schemes do not scale for large number of ranges, as we show in Section 4.1. Therefore, this paper focuses on a database-independent approach.

In this paper we present a database-independent range encoding scheme, called RENÉ - *Range Encoding with No Expansion* - that *eliminates row expansion completely* when ranges are *short enough*. The code produced by RENÉ is *proportional to the maximal range length*, not to the number of ranges, as in database-dependent schemes. In many cases, as we show in this paper, ranges are limited in length. For example, it was shown in [10] that in real-life packet classification tables more than 60% of the TCP port ranges are short. Moreover, packet classification also uses other range fields, where all ranges are short (such as IP ToS or TTL). On some fields one may apply quantization and categorization to reduce the length of ranges without hurting classification accuracy (e.g. packet length). Nonetheless, RENÉ can be combined with other approaches to represent a wider spectrum of ranges if necessary.

In addition to packet-classification, where TCAM has already been selected as de-facto industry standard, we propose in this paper using a TCAM as a co-processor to CPU in order to solve hard problems from other domains in computer science. Specifically, we show how an encoding scheme such as RENÉ, which requires no row expansion, can be used to practically and efficiently solve the *nearest neighbor search* problem and its variants, removing the infamous curse of dimensionality from them.

To our knowledge, we are the first to present a database-independent encoding scheme for short ranges on TCAM with no row expansion. In a nutshell, RENÉ divides all ranges of some length h_{max} into h_{max} layers of disjoint ranges. Using the *binary-reflected Gray code* (BRGC) [20], which was shown to be more expressive for ranges than binary representation [10], it focuses on a specific area where the encoded range is. Using additional bits, it exactly points to the encoded range inside the area in focus, where a single additional bit represents the location of the range inside the layer it belongs to. A toy example is shown in Figure 2.

Using a general conjunction operator we present next, we are able to encode all ranges with length up to h_{max}. The total length of RENÉ's code for a w-bits field, when encoding ranges of *up to* length h_{max}, is $w - \log_2(h_{max}) + h_{max} - 1$. This means that RENÉ is feasible on contemporary TCAMs for ranges up to length of 512, depending on the available space on TCAM and number of range fields. We also present a theoretical analysis and show that at least $max(h_{max} - 1, w)$ bits are required to encode short ranges of up to length h_{max} in a w-bits field. RENÉ is closer to this lower bound than any previously-suggested technique.

We show several applications for RENÉ in the area of packet classification, along with an implementation of such an application on a powerful OpenFlow switch. We also show by experiments that the penalty in latency for translating values using RENÉ is negligible.

We evaluate and experiment our nearest-neighbor algorithms on a real TCAM device and achieve search rates up to four orders of magnitude higher than previous best prior-art solutions [3, 18, 54].

2. BACKGROUND

2.1 Ternary Content Addressable Memory (TCAM)

Contemporary TCAM devices operate at very high rates, between hundreds of millions to more than one billion queries per second [22,46]. These devices have about 20-40 megabits of memory that can be configured to accommodate entries of up to about 640-bits wide (the wider the entry, the fewer entries can be stored on the chip).

The downsides of TCAM are that it is power hungry, tends to generate high heat (thus requiring extra cooling), and relatively expensive, compared to a standard DRAM chip. A high-density TCAM consumes 12 to $15W$ per chip when the entire memory is used [45]. However, compared to compute units and coprocessors such as CPU or GPU, TCAMs' power requirement, heating and price are actually lower, and become similar only when connecting multiple TCAMs in parallel, as usually done in high end networking equipment. For example, Intel's E7-4870 CPU consumes $130W$ [15], and Nvidia's Tesla K80 GPU consumes up to $300W$ [37]. Another downside could be that currently, a TCAM cannot be easily deployed on a standard PC, as they are manufactured for networking equipment.

However, due to their impressive adoption for networking devices, TCAMs are becoming larger, faster, less power hungry and less expensive. We speculate that this trend will continue. Inspired by the adoption of Graphics Processing Units (GPUs) for general purpose parallel computing in recent years, in Section 5 we also suggest that TCAMs may be useful for other tasks outside the networking field.

2.2 Range Encoding on TCAM

The problem of range encoding on TCAM has received considerable attention in the context of packet classification.

The traditional technique for range representation [56] is prefix expansion, where a range is represented 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 a w-bit field is $2w-2$ and for an entry with d ranges it is $(2w-2)^d$. The authors of [10,27] suggest encoding schemes other than binary: In [27], the authors propose DIRPE, a hierarchical version of fence encoding. The authors of [10] propose SRGE - an encoding based on binary reflected Gray code [20]. However, these works do not reduce the range expansion to one, or, in the case of [27], it requires an infeasible exponential memory size to do that. SRGE [10] points out that more ranges can be expressed by using Gray code than when using binary representation, but it only reduces the worst case of row expansion to $2w-4$. DIRPE [27] suggests a tradeoff between row expansion and the number of bits required to code the range. For encoding without expansion, it demands the unfeasible number of $2^w - 1$ bits.

The database-dependent range encoding techniques design the encoding to efficiently encode the ranges that specifically appear in the database. These techniques [8,12,13,29, 57] use extra bits, in addition to the w bits of the range field. The basic idea [29] is to use the extra bits as a bit map: a single extra bit is assigned to each selected range in order to avoid the need to represent it by prefix expansion. Several works [8,12,13,29,57] deal with the scalability problem of this basic technique, which requires one bit per range. However, all these solutions still require either very long rows, proportionally to the number of encoded ranges, or they trade that for row expansion. Moreover, some of these solutions demand extra logic or extra memory that makes them useless in our case, where the number of ranges is high. In [47, 48] it was suggested to use negation rules on TCAM instead of row expansion, when applicable, such that rules may specify the opposite of a range and a corresponding opposite action (e.g. 'deny' instead of 'accept' in ACL). This reduces worst-case expansion factor to w but does not eliminate it, and is only applicable in certain scenarios. Other works [16,26,31–33,49] improve the overall TCAM memory requirements for classification rules, or split the rules into multiple TCAM chips [34,60]. However, these works do not focus specifically on range encoding, and can be used on top of most of the range encoding techniques including the one proposed in this paper.

Other works use TCAM for similarity search in databases. Shinde et al. [54] encode probabilistic hash functions on TCAM to implement locality-sensitive hashing [3]. Our preliminary workshop paper [7] has shown how the encoding technique presented in this paper is used to provide deterministic nearest neighbor search, as further elaborated in Section 5.

Afek et al. [1] use TCAM to implement priority queues with a constant time lookup operation and as a by-product, to provide a TCAM-based sorting algorithm with $O(n)$ time.

The limitation of all the methods has inspired a suggestion to change the TCAM hardware [55], to implement range matching directly in hardware. However this solution changes TCAM architecture dramatically, and it does not seem feasible in the near future, since TCAM is a popular memory chip that exists in tens of millions of routers and switches today. Moreover, the solution harms the flexibility of TCAM implementation, where every entry is simply encoded as a string of ternary bits, regardless of the fields type and borders.

3. ENCODING SCHEME FOR SHORT RANGES

Our goal is to encode a range up to a certain length h_{max} using a single TCAM entry of as few bits as possible. Such code will allow encoding classification rules with multiple ranges without row expansion at all.

3.1 Basic Definitions

We begin with some basic definitions that will be used throughout the rest of this section. First, we define a ternary bit-wise comparison:

DEFINITION 1. *Let $a = a_0, \ldots, a_m$ and $b = b_0, \ldots, b_m$ be two ternary words ($a_i, b_i \in \{0, 1, *\}$). a* **matches** *b, denoted $a \approx b$, if and only if for every $i \in \{0, \ldots, m\}$, either $a_i = b_i$, or a_i is $*$, or b_i is $*$.*

RENÉ encodes ranges in discrete spaces. We begin by defining an encoding function `tcode` for values and ranges. Let $\mathcal{U} = [0, 2^w) \subset \mathbb{N}_0$ be a range on the natural line. RENÉ's encoding function `tcode` encodes either a value $v \in \mathcal{U}$, or a range $R \subseteq \mathcal{U}$. It is important to note that RENÉ treats

$\mathcal{U}$ as a cyclic 'wrap-around' space and thus throughout this paper, any range $[x, y)$ refers in fact to $[x, y \bmod 2^w)$.

The result of the encoding function is either a binary word (for exact values) or a ternary word (for ranges), and we expect that a ternary match $\texttt{tcode}(v) \approx \texttt{tcode}(R)$ will imply that the value v is inside the range R. This is formally defined as follows:

DEFINITION 2. *An encoding function* `tcode` *is* **admissible** *if for every value $v \in \mathcal{U}$ and every range $R \subseteq \mathcal{U}$, $\texttt{tcode}(v) \approx \texttt{tcode}(R)$ if and only if $v \in R$. Furthermore, for any point $v \in \mathcal{U}$, $\texttt{tcode}(v)$ does not contain '*' symbols.*

3.2 Binary-Reflected Gray Code for TCAM

The *binary-reflected Gray code* (BRGC) [20] is a binary encoding of integers in a contiguous range such that the codes of any two consecutive numbers differ by a single bit. A b-bits BRGC is constructed recursively by reflecting a $(b-1)$-bits BRGC [1].

DEFINITION 3. *The BRGC encoding function $BRGC(v, 2^w)$ encodes a point p (where $0 \leqslant p < 2^w$) with w bits. It is defined recursively as follows:*

$$BRGC(0,1) = \varepsilon$$

$$BRGC(p, 2^w) = \begin{cases} 0 \cdot BRGC(p, 2^{w/2}) & \text{if } p < 2^{w/2} \\ 1 \cdot BRGC(2^w - p - 1, 2^{w/2}) & \text{otherwise} \end{cases}$$

where ε is the empty word and '$\cdot$' denotes concatenation.

For example, $BRGC(4,8) = 1 \cdot BRGC(3,4) = 11 \cdot BRGC(0,2) = 110 \cdot BRGC(0,1) = 110$. An example for values in $[0, 16)$ is shown in Figure 2.

By wildcarding some of the bits of a BRGC codeword we can create a ternary range representation. For example, as can be seen in Figure 2, the ternary word `*1**` matches all values in the range $[4, 11]$. In fact, when looking at this tree representation of the BRGC encoding, we observe that all ranges that exactly contain a full sub-tree, or two adjacent full sub-trees, can be represented using a single *ternary* BRGC codeword (namely, a BRGC codeword where some of the 0-1 bits were replaced by '*' symbols).

Before formulating and proving this observation we define the following terms that will be used in the proof:

- *k-prefix* is a ternary word in which the k least significant bits are '*' and the rest are either 0 or 1.
- *k-semi-prefix* is a ternary word in which the k least significant bits are '*', one additional bit is also '*', and the rest are either 0 or 1.

We now formulate the following theorem. Due to space constraints, proofs for all theorems and lemmas in the paper appear in [6].

THEOREM 1. *If all values are BRGC-encoded, then a single ternary BRGC codeword suffices to admissibly encode a range $R = [x, y \bmod 2^w)$ if and only if there exist non-negative integers i, k, for which $x = i \cdot 2^k$ and $y = (i+2) \cdot 2^k$. Specifically, one of the following cases holds:*

1. *If i is even, the $(k+1)$ least significant bits of the codeword are* `*`*, and the rest are either 0 or 1. Thus, the ternary codeword is $(k+1)$-prefix.*
2. *If i is odd, the k least significant bits of the codeword are* `*`*, one additional bit is* `*`*, and the rest are either 0 or 1. Thus, the ternary codeword is k-semi-prefix.*

Theorem 1 implies that when encoding ranges of length h, the $\log_2(h) - 1$ least significant bits are always '*' thus one can save TCAM space by omitting these uninformative bits.

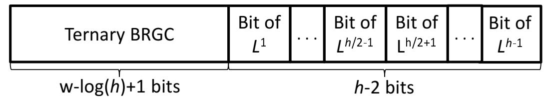

Figure 3: Encoding structure for a value or range of length h

3.3 An Encoding Function for Ranges

We call those ranges that can be encoded using a single ternary BRGC codeword *trivial ranges*, and all other ranges *nontrivial ranges*. In this section, we extend the BRGC encoding scheme so that it can encode in a single ternary word nontrivial ranges as well. We append *extra bits* to the end of BRGC codewords, as depicted in Figure 3: $w - \log_2(h) + 1$ bits are used for the binary BRGC encoding of a value $v \in \mathcal{U}$, or for the ternary BRGC encoding of some trivial range R. To encode nontrivial ranges of length $h = 2^k$ ($k \in \mathbb{N}_0$), at most $h - 2$ bits are added as extra bits.

The key idea of RENÉ is to divide all ranges of some length $h = 2^k$ ($k \in \mathbb{N}_0$) into h *layers*, such that a layer L_h^i is the set of all ranges $[x, x+h)$ for which $x \bmod h = i$. Note that two of these layers contain only trivial ranges (L_h^0 and $L_h^{2^{k-1}}$). We are left with $h-2$ layers that contain nontrivial ranges. We assign an extra bit for each layer of nontrivial ranges. The value of this bit alternates between adjacent ranges in the same layer, such that for any pair of consecutive ranges in the same layer, the value of the bit corresponding to this layer is different. Hence, for a value $v \in \mathcal{U}$, $\texttt{tcode}(v)$ is the $1 + w - log_2(h)$ most significant bits of $BRGC(v)$, concatenated with $h-2$ extra bits. The value of the i^{th} extra bit corresponds to layer L_h^i and is set to $\lfloor \frac{v-i}{h} \rfloor \bmod 2$.

For nontrivial ranges we define their *cover range* as follows:

DEFINITION 4. *For any nontrivial range of length $h = 2^k$ ($k \in \mathbb{N}_0$), $R = [x, x+h)$, let the* **cover range** *of R, denoted by $\texttt{cover}(R)$, be the range $[\lfloor x/h \rfloor \cdot h, (\lfloor x/h \rfloor + 2) \cdot h)$.*

We first notice the following property of cover ranges:

LEMMA 1. *For any range $R = [x, x+h)$ of length $h = 2^k$ ($k \in \mathbb{N}_0$), $\texttt{cover}(R)$ fully contains R.*

Note that the existence of the cover range is a unique property of the binary-reflected Gray code. The cover range $\texttt{cover}(R)$ helps us distinguish R from other ranges in the same layer. For range $R = [x, x+h) \in L_h^i$ of length $h = 2^k$ ($k \in \mathbb{N}_0$), $\texttt{tcode}(I)$ starts with the $1 + w - log_2(h)$ most

[1] The BRGC of a value x can be directly calculated using the following formula: $x \oplus (x >> 1)$, where $\oplus$ and $>>$ are the bitwise operations of XOR and Right Shift, respectively.

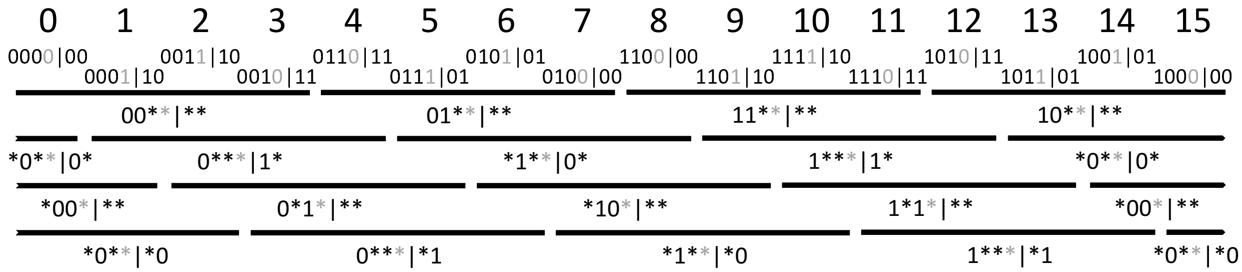

Figure 4: Encoding for all sub-ranges of length 4 and values in range $[0, 16)$. Left bits are the ternary BRGC encoding. Right bits are the extra bits for nontrivial layers. The bits in gray can be removed as explained in Section 3.2

significant bits of the ternary BRGC representation of R, if R is trivial, or of $\texttt{cover}(R)$, if R is nontrivial. Then, $h-2$ extra bits are concatenated one after the other where the i^{th} bit is either `*` if $I \notin L_h^i$ or $\lfloor \frac{x-i}{h} \rfloor \bmod 2$ otherwise (namely, all the extra bits except one are `*`).

Our main result is that RENÉ's encoding function, `tcode`, is an *admissible encoding function* for ranges of any length $h = 2^k$ $(k \in \mathbb{N}_0)$. The total length of the admissible encoding produced by `tcode` for a single value or range is hence $w - \log(h) + h - 1$.

Before proving this result (Theorem 2), we introduce the following two technical lemmas:

LEMMA 2. *If a range R is nontrivial, then no other range from the same layer is fully contained in $\texttt{cover}(R)$.*

Based on this property of cover ranges, we can completely distinguish between ranges in the same layer using the extra bits we added to the ternary BRGC encoding:

LEMMA 3. *Let $R = [x, x+h)$, where $h = 2^k$, be a range in L_h^i. For every value v in $\texttt{cover}(R)$, if $v \in R$, v has the same bit value as R, and if $v \notin R$, then v has the opposite bit value.*

We now turn to the main theorem.

THEOREM 2. *The function `tcode` is an admissible encoding function for ranges of length $h = 2^k$.*

Figure 4 shows the encoding of all sub-ranges of length 4 in range $[0, 16)$. Note that the first and third layers do not require extra bits, so these are both set to `*` in their encoding. In other layers, the corresponding extra bit alternates between ranges in the same layer. For example, the range $[1, 4]$, which cannot be encoded solely using a ternary BRGC codeword, is encoded as `0***1*`, where the fifth bit is the extra bit that corresponds to the second layer. Only points in $[1, 4]$ match this encoding.

3.4 Encoding Multiple Range Lengths

Given RENÉ's encoding function for ranges of some maximal length h_{max} we can encode, without using more bits, *all ranges whose lengths are smaller than h_{max}* as well. We define a logical conjunction operation, denoted by $\sqcap$, to encode the intersection of two ranges. The truth table of such a conjunction is given in Table 1. $\perp$ means an undefined output,

a_i	b_i	$a_i \sqcap b_i$
*	*	*
0	*	0
*	0	0
0	0	0
1	*	1

a_i	b_i	$a_i \sqcap b_i$
*	1	1
1	1	1
0	1	$\perp$
1	0	$\perp$

Table 1: The truth table of a ternary logical conjunction, denoted by the $\sqcap$ operator.

and we later make sure to never get such an output when using this operation. For two ternary words, $a = a_0, \ldots, a_m$ and $b = b_0, \ldots, b_m$, the conjunction $c = a \sqcap b$ is the ternary word where $c_i = a_i \sqcap b_i$. If at least one of the symbols c_i is $\perp$, then c is also marked as $\perp$ and is not defined. Note that the conjunction operation is independent of the specific encoding function.

The essence of the conjunction operation is captured in the following two lemmas:

LEMMA 4. *For any value $v \in \mathcal{U}$ and any two ranges $R_1, R_2 \subseteq \mathcal{U}$, if `tcode` is an admissible encoding function for R_1 and R_2, then $\texttt{tcode}(v) \approx \texttt{tcode}(R_1) \sqcap \texttt{tcode}(R_2)$ if and only if $v \in R_1 \cap R_2$.*

LEMMA 5. *If `tcode` is an admissible encoding function for R_1 and R_2, and the result of $\texttt{tcode}(R_1) \sqcap \texttt{tcode}(R_2)$ is $\perp$ then $R_1 \cap R_2 = \emptyset$.*

Note that the other direction of Lemma 5 is not necessarily true: the conjunction of codes of two disjoint ranges may not be $\perp$.

We assume that there is a value $h_{max} = 2^{k_{max}}$, which is the maximum length we should consider. Note that any range $[x, x+h)$ of length $h < h_{max}$ (h is not necessarily a power of 2 anymore) can be written as the intersection of two ranges of length h_{max} as follows:

$$[x, x+h) = [x+h-h_{max}, x+h) \cap [x, x+h_{max}).$$

Using the conjunction operator and Lemma 4 we can construct the code for ranges of any length $h \leqslant h_{max}$, with $h_{max} - 2$ extra-bits:

$$\texttt{tcode}([x, x+h)) =$$

$$\texttt{tcode}([x+h-h_{max}, x+h)) \sqcap \texttt{tcode}([x, x+h_{max})).$$

Algorithm 1 Encoding Function for a Value v

```
function tcode (v, h_max)
                                  ▷ v - value, h_max - maximal range length
    word ← BRGC(p) >> (log_2(h_max) − 1)    ▷ Bitwise shift
    for i ← 0 to (h_max − 1) do
        if layer is skipped then
            continue                  ▷ Optional - encode less layers
        end if
        if layer is nontrivial (i ≠ 0 and i ≠ h_max/2) then
            b ← ⌊(v−i)/h_max⌋ mod 2
            word ← (word << 1)|b                ▷ Bitwise OR
        end if
    end for
    return word
end function
```

Algorithm 2 Encoding Function for a Range $[s, t]$

```
function tcode ([s, t], h_max)
                         ▷ [s, t] - range, h_max - maximal range length
    if t − s + 1 ≠ h_max then
                                     ▷ Encode range as an intersection
        R_1 = [s, s + h_max − 1]
        R_2 = [t − h_max + 1, t]
        Γ ← {R_1, R_2}
    else                                      ▷ Encode range directly
        Γ ← {[s, t]}
    end if
    result ← 0
    count ← 0
    for [x, y] ∈ Γ do
        mask ← 0
        for i ← x + 1 to y do
            mask ← mask|(BRGC(i − 1) ⊕ BRGC(i))
                                              ▷ bitwise OR and XOR
        end for
        word ← BRGC(x) >> (log_2(h_max) − 1)
        mask ← mask >> (log_2(h_max) − 1)
        for i ← 0 to (h_max − 1) do
            if layer is skipped then
                continue              ▷ Optional - encode less layers
            end if
            if layer is nontrivial (i ≠ 0 and i ≠ h_max/2) then
                if x mod h_max ≠ i then         ▷ Irrelevant layer
                    mask ← (mask << 1)|1              ▷ Put a '*'
                    word ← word << 1
                else                          ▷ [x, y] is in this layer
                    mask ← mask << 1
                    b ← ⌊(x−i)/h_max⌋ mod 2
                    word ← (word << 1)|b
                end if
            end if
        end for
        if count > 0 then
            result ← result ⊓ (word, mask)
        else
            result ← (word, mask)
        end if
        count ← count + 1
    end for
    return result
end function
```

We also know from Lemma 5 that $\texttt{tcode}([x, x + h))$ is not $\perp$ as the intersection is never empty.

The encoding function $\texttt{tcode}(v)$ for some value v when using any range lengths up to h_{max} is shown in Algorithm 1. When encoding a range R of length $h \leqslant h_{max}$ that is centered at some point v, Algorithm 2 is used to obtain $\texttt{tcode}(R)$.

The length of the resulting encoding of a value $v \in \mathcal{U}$ or a range $R \subseteq \mathcal{U}$ is therefore $w - \log_2(h_{max}) + h_{max} - 1$.

3.5 Running Time Analysis

Computing `tcode` for either a value or a range is simple: results only depend on the value or range themselves, and the maximal range length h_{max}. The running time of both functions, for a value and a range, is linear with h_{max}, and does not depend in the number of encoded ranges: $O(h_{max})$ when encoding a value and $O(h_{max} + h)$ when encoding a range of length $h \leqslant h_{max}$.

3.6 Lower Bound on the Number of Bits per Range

As previously recalled, Lakshminarayanan et al. [27] introduced the worst-case necessary condition of $2^w - 1$ bits to encode a w-bit range. We use this observation to introduce a lower bound for the number of bits required to encode ranges, when their lengths are limited by some upper bound h_{max}:

THEOREM 3. *In order to represent any ranges up to length h_{max} without row expansion, in a field of width w bits, at least $max(h_{max} - 1, w)$ bits are necessary.*

4. RENÉ FOR PACKET CLASSIFICATION

Range encoding on TCAM has been used for packet classification for long time. Row expansion significantly limited its usage when multiple header fields are ranges, leading vendors and administrators to avoid such situations as much as possible [41]. However, next generation SDN applications, such as load balancers, security tools, and quality of service, rely on sophisticated packet classification that is performed on the datapath itself (i.e. the switch) [5, 43, 53, 58]. Most of these solutions require range based matching on multiple header fields. We summarize several examples for such fields and metadata information that can benefit when using RENÉ:

- **TCP/UDP Port Fields:** In real-life datasets, short ranges (up to length 64) sometimes consist more than 60% of the unique ranges [10]. Thus, if a network administrator uses mainly short ranges for TCP/UDP port fields, or even for only one of these fields, RENÉ may suit their needs.

- **Network ToS (or DSCP):** In both the deprecated ToS field and the new DSCP field the precedence is set using an increasing value, and to specify one or more precedence classes, either an exact value or a short range should be used.

- **Packet Size:** Packet size (e.g. IP total length field) can be a useful piece of information for packet classification. When classifying according to this property, a categorization can be done in order to reduce range lengths. As usually one does not classify packets according to a specific length, but rather according to categories (small, medium, large, etc.), short ranges can be used to represent multiple categories. For example, a recent attack named *Tsunami SYN Flood Attack* can be identified based on the size of packets (about 1000 bytes or more) [44].

- **Timestamp and Counters:** Recent works suggest adding packet's metadata such as hit counters and timestamps (or time deltas) to classification data path, for example in OpenFlow switches [5]. It is likely that classification on such fields would be based on ranges and not on exact values, and thus RENÉ may be used.

- **IP Spoofing Detection:** In order to protect against IP spoofing and attacks that use this technique (e.g., DDoS), it was suggested to inspect the IP TTL value and conclude about possible spoofed packets [25, 40]. The detection is based on the fact that the TTL value does not change dramatically over short time for the same host or subnet, and these values can be found using `ping` and other tools. Thus, if a packet with IP from a known subnet comes with a TTL value that is too far from the expected value, it is classified as spoofed and dropped.

  Since the TTL value is not compared to an exact value, but rather to a short range, RENÉ can be used in order to implement IP spoofing detection on classification hardware with TCAM.

- **AS Numbering:** BGP routers and SDX [21] sometimes make classification decisions based on autonomous systems (ASes) numbers. Large ISPs and content providers usually hold multiple, consecutive AS numbers [14], which form one or more short ranges. For example, Comcast has multiple such short ranges 7015-7016, 33489-33491, 33650-33668 (in addition to five more non-consecutive AS numbers). Grouping AS numbers to ranges can reduce the total number of classification rules, as long as no row expansion is induced. RENÉ fits this goal as the ranges are short and it induces no row expansion.

4.1 Evaluation and Experiments

4.1.1 Experiment on an OpenFlow Switch

We implemented RENÉ and a sample SDN application that uses it for packet classification over the Ryu SDN controller [51]. We use a NoviFlow NoviKit 250 hardware switch that supports OpenFlow 1.3 [39] and has an internal TCAM. Our code is available at `https://github.com/yotamhc/rene`.

Classification uses OpenFlow table pipeline in the following way: First table, given a destination TCP port for a packet, writes its translation into RENÉ's encoding to the metadata field (using the OpenFlow's Write-Metadata instruction). This table is precomputed on the controller and contains up to 64K entries - an entry for each port number. Then, second table matches the packet according to the metadata only (original port information is not necessary at this stage), and forwards it accordingly.

On the same switch, when a packet is classified based on only its TCP port, without table pipeline, the total round-trip time to and from the switch, using a 1Gbps copper link, is 157μs. Using our table pipeline, such round-trip takes 161μs. Thus, latency increases by only 2.5%, which is a negligible factor.

4.1.2 Quantitative Evaluation

To evaluate the quality of RENÉ's encoding function `tcode` we compare it with best prior-art encoding techniques that

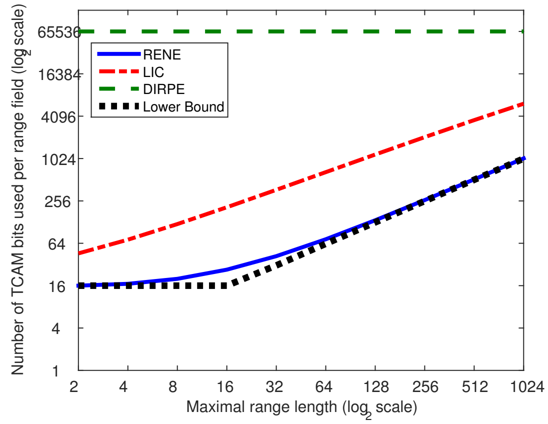

Figure 5: Analysis of the number of TCAM bits required for a 16-bits range field when representing all ranges of up to a given length.

can provide no row expansion: DIRPE [27], a database-independent encoding scheme and LIC [8], a database-dependent encoding scheme. We do not compare RENÉ to SRGE [10], for example, as it requires row expansion. We evaluate the database dependent scheme LIC both in its worst case, where all ranges are to be represented, and using a commercial classification dataset with 257 range rules. Since our goal is no expansion of TCAM entries, we compare the amount of TCAM bits required for a single range field, such that no expansion is induced whatsoever. Using the classification database, LIC performs worse than RENÉ on ranges up to length 32. When the dataset contains much higher number of ranges, LIC always performs worse than RENÉ.

Figure 5 shows the bit requirement of each encoding technique, given the maximal length of encoded ranges, assuming a 16-bits range field. In addition, it shows the lower bound of $max(h_{max} - 1, w)$ bits (see Theorem 3), as a black, dotted line. Evidently, RENÉ (blue, solid line) is much closer to the lower bound than all other techniques. Moreover, the bit requirements for database-dependent techniques such as LIC [8] are higher by an order of magnitude, when *all* ranges up to a certain length should be encoded. The database-independent technique DIRPE [27] always requires $2^w - 1$ bits as it does not use the additional information on the maximal range length.

5. RENÉ FOR NEAREST NEIGHBOR SEARCH

TCAM is a powerful device with high parallelism that can also be used for tasks outside of the networking fields. Just as TCAM has broken the performance limits of packet classification and IP lookup in networking, it can also be used to break such computational limits in problems from other fields, serving as a coprocessor for the CPU, similarly to a GPU or FPGA. RENÉ can be used to implement on such TCAM applications that use the nearest-neighbor problem or its variants. We show several such variants in this section, and by experiments and simulations we show that RENÉ can improve their performance by orders of magnitudes.

Multidimensional nearest neighbor search (NN) lies at the core of many computer science applications. In the space of integers, the problem is formally defined as follows:

Definition 5. *Given a set of data points* $S = \{p_i\}_{i=1}^{n}$, $p_i \in \mathbb{Z}^d$ *and a query point* $q \in \mathbb{Z}^d$, The Nearest-Neighbor Problem *is to find the point* $s^* = \arg\min_{s \in S} D(s, q)$, *where* $D(s, q)$ *is a distance between* s *and* q.

The NN problem and its variants are utilized in a wide range of applications, such as spatial search, object recognition, image matching, image segmentation, classification and detection, to name a few [11, 28, 42].

When the dimensionality of the points is small, many solutions were proven to be very effective. These include mainly space partitioning techniques [52]. However, when the number of the data points is large (in the order of tens of thousands or higher) and the dimensionality is high (in the order of tens or hundreds), the exact solutions break down and produce exponential time complexity[2] [3, 59]. This problem is widely known as the *curse of dimensionality.*

To overcome the curse of dimensionality, *approximated nearest neighbor* (ANN) solutions are commonly used. In particular, a c-ANN is a solution where the distance of the retrieved point from q is at most c times the true distance from the nearest point. For the ANN problem, probabilistic dimensionality reduction such as *locality sensitive hashing* (LSH) [3] was proven to be useful, with query time sublinear in n but linear in d. For very-high dimensional space this may still pose a problem [36]. Note also that the solution provided by LSH is correct only with high probability.

In this section we present super high-speed algorithms for the NN problem using TCAM as a coprocessor, and our encoding scheme RENÉ. The proposed algorithms solve the ANN problem with ℓ_∞-normed distance using *a single TCAM lookup* and linear space.

To the best of our knowledge, using TCAM for nearest neighbor search has been considered only once, by Shinde et al. [54] who proposed the TLSH scheme, where a TCAM device is utilized to implement LSH with a series of TCAM lookup cycles. Our algorithms, however, are deterministic and they provide higher throughput with less memory required on TCAM.

A simpler problem that we will use as a building block in our algorithms only searches for a neighbor close enough to the query point, or discovers that there is no such neighbor at all:

Definition 6. *Given a set of points* $S = \{p_i\}_{i=1}^{n}$, $p_i \in \mathbb{Z}^d$, *and a query point* $q \in \mathbb{Z}^d$, *let* $d^* = \min_{s \in S} D(s, q)$. *The* r–Near-Neighbor Report Problem *is to find the point* $s' \in S$ *such that* $D(s', q) \le r$ *if* $d^* \le r$, *and to return* false *if* $d^* > r$.

Note that under ℓ_∞, a solution for the r–Near-Neighbor Report Problem is a data point within the d-dimensional cube of edge length $2r$ that is centered in the query point. Thus, our framework for solving c-ANN can be viewed as solving (either in parallel or sequentially) a series of r–Near-Neighbor Report Problem instances for increasing values of r. As pointed out in [23], this solves the c-Approximate Nearest Neighbor Problem, where the approximation

[2]Exact brute-force search works in time that is linear to n and d, but is very slow for high n and d. Space partitioning techniques are exponential in d.

Algorithm 3 Encoding Function for a d-Dimensional Point

```
function TCODE(p[], d, h_max)
    word ← ε
    for i ← 1 to d do
        word ← word + tcode(p[i], h_max)
    end for
    return word
end function
```

Algorithm 4 Encoding Function for a d-Dimensional Cube

```
function TCODE(p[], d, h, h_max)
    word ← ε
    for i ← 1 to d do
        word ← word + tcode([p[i] − ⌊h/2⌋ , p[i] + ⌊h/2⌋], h_max)
    end for
    return word
end function
```

ratio is determined by the maximum ratio between consecutive values of r.

Using RENÉ's encoding function `tcode` , a single ternary match can determine whether a given d-dimensional point is inside a given d-dimensional cube: To encode a d-dimensional point, or a d-dimensional cube, each coordinate is encoded using the `tcode` function, and the codewords of all d coordinates are concatenated into a single ternary word. The encoding functions for a d-dimensional point and for a d-dimensional cube are shown in Algorithm 3 and in Algorithm 4, respectively.

5.1 Approximate Nearest-Neighbor Search

Our Approximate Nearest-Neighbor Search algorithms solve in fact multiple instances of the r-Near Neighbor Report Problem for increasing values of r. In ℓ_∞, the value of r defines a cube around each data point p such that for all query points q inside that cube, p is a valid solution of the r-Near Neighbor Report Problem with q, and for all query points outside that cube p is not a valid solution.

Our *time-efficient method* solves the Approximate Nearest-Neighbor Search in ***a single TCAM lookup***. Given a set $\mathcal{H}$ of edge lengths, let $h_{max} = \max_h \mathcal{H}$. For each point $p \in S$ and $h \in \mathcal{H}$ we store a TCAM entry representing a d-dimensional cube centered at p, with edge length h. Entries are sorted by the value of h: the smaller h is, the higher the priority of the entry.

Given a query point $q \in [0, w)^d$, we use `tcode` to build a d-dimensional point representation for maximal edge length of h_{max}, and use a *single TCAM lookup* to find the smallest cube that contains the point q. The TCAM returns the highest priority entry that matches, which is the entry of the cube that is centered at some point p, has the shortest edge length, and contains q. An example is shown in Figure 6 (left). Note that in general, p is not necessarily the exact nearest neighbor of q (as there may be more than one such cube with the same edge length h). However, the distance (under ℓ_∞) of q from its exact nearest neighbor is strictly more than $\left\lfloor \frac{1}{2} \max_{h' \in \mathcal{H}} \{h' < h\} \right\rfloor$. As we will show later, by carefully choosing the edge length set, we can obtain a $c = 1 + \varepsilon$ approximation factor, where the size of $\mathcal{H}$ is inversely proportional to ε.

In our *memory-efficient method*, the data points and query points switch roles: we store in the TCAM a single entry for each data point. The order of the entries does not matter.

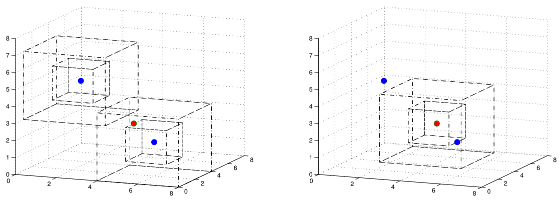

Figure 6: Illustration of the two alternative algorithms for Nearest Neighbor Search using TCAM. Left: Encoding nested cubes around each point in the database. A query is a point in this encoding, and the result is the smallest cube encompassing the query point. Right: Encoding cubes around the query points and the data is encoded as points. A query is a sequence of nested cubes in increasing edge length. The result is the first data point that matches a cube.

Upon a query q, we construct a sequence of $|\mathcal{H}|$ cubes centered in p with edge lengths in $\mathcal{H}$. Then, we perform TCAM lookups with cubes of increasing edge length values until a match is found. As in the previous method, if a point was matched with a query of edge length h, then it is a solution of the $h/2$-NEAR NEIGHBOR REPORT PROBLEM.

5.1.1 Analysis of Approximation

Let $\mathcal{H} = \{h_1, h_2, \ldots, h_{max}\}$ such that $h_i < h_j$ for each $i < j$. Matching a data point p corresponding to a cube with edge length h_i implies that $D(p, q) \leq \left\lfloor \frac{h_i}{2} \right\rfloor$ (where D is defined under ℓ_∞). Since h_i is the first edge length to be matched, $D(s^*, q) \geq \left\lfloor \frac{h_{i-1}}{2} \right\rfloor + 1$ (s^* is the exact nearest neighbor of q). This implies that under ℓ_∞, both methods solve the c-APPROXIMATE-NEAREST-NEIGHBOR PROBLEM for $c = \max_{h_i \in \mathcal{H}} \frac{\lfloor h_i/2 \rfloor}{\lfloor h_{i-1}/2 \rfloor + 1}$, where h_i is the i^{th} smallest element in $\mathcal{H}$ and $h_1 = 1 \in \mathcal{H}$. [3]

In order to get the exact nearest neighbor in ℓ_∞, one can choose $\mathcal{H}$ to be the set of odd numbers. Reducing the size of $\mathcal{H}$ reduces the number of required entries, but decreases the quality of the results. For example, to get a c-approximate solution, $\mathcal{H}$ can consist of all even values up to $2/(c-1)$, along with the values of a geometric series starting at $2/(c-1)$, with the parameter c.

When distances are defined under ℓ_p norms, for finite values of $p \geq 1$, the approximation ratio is at most $c \cdot \sqrt[p]{d}$ in ℓ_p, where c is the approximation ratio in ℓ_∞.

5.1.2 Database Update

Our algorithms allow efficient hot updates in the lookup database (the set S). Deletion of data points is trivial (simply delete all corresponding entries from the TCAM). When using the *time-efficient method*, efficient addition of new points is possible by keeping some empty TCAM entries between entries of different edge length, by adding entries for the corresponding cubes in these empty slots. Also, one can track deletion for more empty slots. Nevertheless, this further increases space requirement.

[3] To get a bounded approximation ratio, 1 must be added to $\mathcal{H}$.

When using the *memory-efficient method*, the situation is simpler: since the order of entries is not important, point addition or deletion requires a single TCAM entry update.

5.2 Exact Nearest-Neighbor Search in ℓ_p

Our algorithms achieve $\sqrt[p]{d}$ approximate solution under ℓ_p norm. We suggest the following extension to find (exactly) the nearest neighbor under ℓ_p: For each data point $s \in S$ and for each edge length $h \in \mathcal{H}$, we *precompute* the *neighborhood set* $\mathcal{N}(s, h) = \{s' \in S \mid D_p(s, s') \leq h\sqrt[p]{d}\}$, where D_p is the distance between the two points under the p-norm. The neighborhood sets are stored in memory. Precomputing these sets is possible since datasets are relatively static and the neighborhood sets do not depend on the query points.

Since for every two points, the distance in ℓ_p is at most $\sqrt[p]{d}$ the distance in ℓ_∞, we immediately conclude that if the algorithms described in Section 5.1 return a data point s for query point q with some distance $h \in \mathcal{H}$, then the exact nearest neighbor in ℓ_p of q is in $\mathcal{N}(s, h)$.

While this method requires additional computations following the TCAM lookup, in most datasets the number of points in $\mathcal{N}(s, h)$ would be very small. In our experiments (see Section 5.5) $\mathcal{N}(s, h)$ contained only s itself for lower values of h in most cases and was small even for higher values of h. Thus, the time required to find the exact nearest neighbor is still much shorter than that required for brute-force over all points in the database.

The precomputed neighborhood sets can also be used to find k-nearest neighbors instead of only one. However, the number of neighbors in these sets might be smaller than k, so one TCAM lookup might not suffice. To find the set of k exact nearest neighbors, the lookup process should continue until k or more neighbors are found, and also until no more neighbors are found in cubes whose edge length is equal to that of previous neighbors. This process is formally described in [6, Algorithm 5], assuming a multi-match technique such as the one presented in [27] is used.

5.3 Algorithms for the Partial Match Problem

The PARTIAL MATCH PROBLEM is defined as follows:

DEFINITION 7. *Given a set of data points $S = \{p_i\}_{i=1}^n$, $p_i \in \mathbb{Z}^d$, a query point $q \in \mathbb{Z}^d$, and a subset of the dimensions $D_q \subseteq \{1, \ldots, d\}$ of size $d_q < d$,* THE PARTIAL MATCH PROBLEM *is to find the point $s^* = \arg\min_{s' \in S} D(s', q)|_{D_q}$, where $D(a, b)|_{D_q}$ is the distance between a and b under some norm in the d_q-dimensional space. Namely, for a p-norm,*

$$D(a, b)|_{D_q} = \left(\sum_{i \in D_q} |a_i - b_i|^p \right)^{1/p}.$$

This problem is useful when some features in the vector are not important for a specific query or user, and in traditional computing models it is known to be *more difficult* [4] than the nearest neighbor problem, where all relevant dimensions are given a-priori. For example, LSH (and its extension to TCAMs, TLSH [54]) cannot be used to solve this problem. However, our solution for the NN problem can be used instantly to solve the partial match problem.

Under the maximum norm ℓ_∞, a PARTIAL MATCH solution is to replace, *in the queries*, all the bits corresponding to coordinates in irrelevant dimensions with * bits. We replace coordinates in queries and not for data point, as the relevant

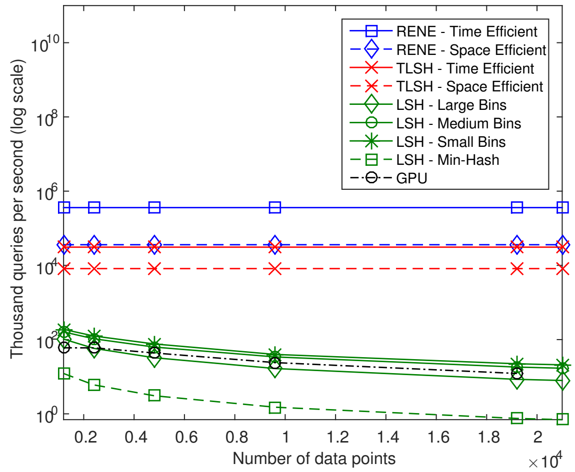

Figure 7: Throughput comparison of the various algorithms for solving the nearest neighbor problem, as a function of the size of the search database. Throughput of TCAM-based algorithms is simulated based on $360MHz$ TCAM throughput. We denote TLSH with one TCAM lookup per r-Near Neighbor Report Problem instance as *time-efficient*, and TLSH with $\log(1/\varepsilon)$ TCAM lookups per instance as *space-efficient*.

dimensions are selected per query. This technique works both for our *time-efficient* and *memory-efficient methods*.

For ℓ_p, our solution results in $\sqrt[p]{d_q}$ approximation, where d_q is the dimension of the specific query. The extensions to EXACT NEAREST NEIGHBOR SEARCH and k–NEAREST NEIGHBORS SEARCH, as described in Section 5.2, work also for this problem. The neighborhood sets are precomputed on the d-dimensional space, but queries and distance computations after queries are done on the specific d_q dimensional space. The results are still correct as distances in the d_q-dimensional space are bounded by distances in the d-dimensional space.

5.4 Geometric Clustering on TCAM

Another closely related problem that could benefit from using TCAM with RENÉ is high-dimensional geometric clustering. The k-MEANS CLUSTERING problem, for example, is usually solved as a sequence of nearest-neighbor search problems, each of these consists of a database with k d-dimensional points [30].

Algorithm [6, Algorithm 6] shows how the traditional k-MEANS CLUSTERING algorithm can be implemented on TCAM using our encoding function `tcode`, under ℓ_∞ norm. As k is usually much smaller than the number of points, this solution requires relatively low TCAM space. Still, a standard TCAM can support up to thousands of clusters using this algorithm.

5.5 Evaluation and Experimental Results

We evaluate our nearest-neighbor algorithms using an image similarity search application (using GIST [38] descriptors), on a real-life image dataset [50]. We then compare the results and performance with prior-art solutions. Our evaluation is based both on experiments with real-life TCAM devices and simulations. Each image in the dataset was encoded as a GIST vector in $\mathbb{R}^{512}$, downsampled to $\mathbb{R}^{40}$ and quantified to $\{0,\ldots,255\}^{40}$ before performing search. Images were randomly partitioned to a dataset of $21,019$ images a query set of $1,000$ images.

5.5.1 *Experiment with a TCAM Device*

Since there is no evaluation board for such devices, we used a commodity network switch (Quanta T1048-LB9) that contains a TCAM for our experiment (similarly to [54]). This switch has 48 1 Gbps ports, each handling at most $1.5M$ packets per second. TCAM is used for packet classification for OpenFlow 1.3. Using the OpenFlow interface to the switch, we mapped each entry produced by our algorithms to a set of header fields. A commercial traffic generator injected manually crafted packets that contain the queries in their headers, where each query is broken into header fields in the same way TCAM entries are stored.

We verified correctness by counting the number of matches of each TCAM entry. Using one ingress port the switch easily achieved a throughput of $1.5M$ queries per second, which is the upper bound of the link between the traffic generator and the switch (but not of the TCAM). Using 24 ingress ports we achieved throughput of $35.69M$ queries per second (almost $1.5M \times 24$). Hence, the bottleneck was not in the TCAM: If we had more ports we could have reached much higher throughput.

5.5.2 *Simulation Results*

We compared our results to the results of brute-force exact nearest neighbor (using MATLAB or on GPU [18]), locality sensitive hashing [3,23] (using implementation from [2]), and TLSH [54]. LSH approximated results in ℓ_2 were comparable to our approximated results in ℓ_2 only when LSH used the most complex hash functions, or when used very large bins. Both options mean longer computation time due to either more complex hash computation or much more distance computations.

Figure 7 presents a comparison of the throughput (queries per second) of CPU implementations of LSH [2], GPU implementation [18], TLSH [54] simulation, and RENÉ simulation. Each line in the figure presents the throughput of a single algorithm/implementation, as a function of the number of data points in the dataset. [4]

The required TCAM space for our *space efficient* method is $8M$ bits and for our *time efficient* method with 10 different cube sizes is $80M$ bits, with 440 bits wide entries. These requirements are available in most modern TCAM devices. TLSH requires much higher TCAM capacity and much wider TCAM entries.

6. CONCLUSIONS

While the problem of range encoding on TCAM has been deeply investigated over recent years, the proven theoretical limits on the number of bits one must use have diverted researchers to use row expansion. However, row expansion

[4]LSH implementations were ran on an Intel Core i7 2600 3.4GHz CPU. We used the same dataset and queries for our algorithms, LSH and TLSH. TCAM algorithms used 10 different cube sizes. GPU throughput is as reported in [18] for the closest lower values of n and d.

causes exponential increment in the number of TCAM entries. New applications such as SDN implementations for load balancing, security tools, and NFV frameworks use more than a few range fields. Thus, row expansion makes solutions that use it impractical.

In this paper we introduce the sub-problem of *short range encoding*, and we show that the theoretical limits on the number of required bits can be lowered in this situation. We present RENÉ: An encoding scheme for short ranges, and show that it is closer to the lower bound than any other technique. We then present multiple applications that may benefit from such short range encoding, in the area of packet classification. Furthermore, we propose to use TCAM as a co-processor for solving problems outside of the networking field, such as the nearest neighbor problem and its variants, which so far has been known to take very long time to compute. We show that using TCAM, one could solve such problems in much higher rates than previously suggested solutions, and outperform known lower bounds in traditional memory models.

Acknowledgements

This research was supported by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013)/ERC Grant agreement n^{o} 259085 and the Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11).

7. REFERENCES

[1] Yehuda Afek, Anat Bremler-Barr, and Liron Schiff. Recursive design of hardware priority queues. In *SPAA*, pages 23–32, 2013.

[2] M. Aly, M. Munich, and P. Perona. Indexing in large scale image collections: Scaling properties and benchmark. In *WACV*, pages 418–425, 2011.

[3] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Comm. of the ACM*, 51(1):117–122, 2008.

[4] Allan Borodin, Rafail Ostrovsky, and Yuval Rabani. Lower bounds for high dimensional nearest neighbor search and related problems. In *STOC*, pages 312–321, 1999.

[5] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: programming protocol-independent packet processors. *CCR*, 44(3):87–95, 2014.

[6] Anat Bremler-Barr, Yotam Harchol, David Hay, and Yacov Hel-Or. Encoding short ranges in tcam without expansion: Efficient algorithm and applications. Technical report, IDC Herzliya / Hebrew University of Jerusalem, 2015. http://www.deepness-lab.org/pubs/spaa2016_tcam_techreport.pdf.

[7] Anat Bremler-Barr, Yotam Harchol, David Hay, and Yacov Hel-Or. Ultra-fast similarity search using ternary content addressable memory. In *DaMoN*, 2015.

[8] Anat Bremler-Barr, David Hay, and Danny Hendler. Layered interval codes for TCAM-based classification. *Comput. Netw.*, 56(13):3023–3039, Sep 2012.

[9] Anat Bremler-Barr, David Hay, and Yaron Koral. CompactDFA: Scalable pattern matching using longest prefix match solutions. *IEEE/ACM Trans. Netw.*, 22(2):415–428, 2014.

[10] Anat Bremler-Barr and Danny Hendler. Space-efficient TCAM-based classification using Gray coding. In *INFOCOM*, pages 1388–1396, 2007.

[11] Matthew Brown and David G Lowe. Recognising panoramas. In *ICCV*, volume 3, page 1218, 2003.

[12] Yeim-Kuan Chang and Cheng-Chien Su. Efficient TCAM encoding schemes for packet classification using gray code. In *GLOBECOM*, 2007.

[13] Hao Che, Z. Wang, Kai Zheng, and Bin Liu. DRES: Dynamic range encoding scheme for TCAM coprocessors. *Transactions on Computers*, 57(7):902–915, 2008.

[14] AS names - CIDR report. http://www.cidr-report.org/as2.0/autnums.html.

[15] Intel Corp. Intel Xeon processor E7-4870, 2011. http://ark.intel.com/products/53579/.

[16] Qunfeng Dong, Suman Banerjee, Jia Wang, Dheeraj Agrawal, and Ashutosh Shukla. Packet classifiers in ternary CAMs can be smaller. In *SIGMETRICS*, pages 311–322, 2006.

[17] ETSI. Network function virtualization, Oct 2012. http://portal.etsi.org/NFV/NFV_White_Paper.pdf.

[18] Vincent Garcia, Eric Debreuve, Frank Nielsen, and Michel Barlaud. K-nearest neighbor search: Fast GPU-based implementations and application to high-dimensional feature matching. In *ICIP*, pages 3757–3760, 2010.

[19] Ashish Goel and Pankaj Gupta. Small subset queries and bloom filters using ternary associative memories, with applications. In *SIGMETRICS*, pages 143–154, 2010.

[20] F. Gray. Pulse code communication. US Patent 2,632,058, March 17 1953 (filed November 13 1947).

[21] Arpit Gupta, Laurent Vanbever, Muhammad Shahbaz, Sean P. Donovan, Brandon Schlinker, Nick Feamster, Jennifer Rexford, Scott Shenker, Russell J. Clark, and Ethan Katz-Bassett. SDX: a software defined internet exchange. In *SIGCOMM*, pages 551–562, 2014.

[22] Cavium Inc. NEURON search processors, 2014. http://www.cavium.com/processor_NEURON-Search.html.

[23] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, pages 604–613, 1998.

[24] Weirong Jiang, Qingbo Wang, and Viktor K Prasanna. Beyond TCAMs: An SRAM-based parallel multi-pipeline architecture for terabit IP lookup. In *INFOCOM*, pages 1786–1794, 2008.

[25] Cheng Jin, Haining Wang, and Kang G. Shin. Hop-count filtering: An effective defense against spoofed DDoS traffic. In *CCS*, pages 30–41, 2003.

[26] Kirill Kogan, Sergey I. Nikolenko, Ori Rottenstreich, William Culhane, and Patrick Eugster. SAX-PAC (scalable and expressive packet classification). In *SIGCOMM*, pages 15–26, 2014.

[27] Karthik Lakshminarayanan, Anand Rangarajan, and Srinivasan Venkatachary. Algorithms for advanced packet classification with ternary CAMs. In *SIGCOMM*, pages 193–204, 2005.

[28] Lin Liang, Ce Liu, Ying-Qing Xu, Baining Guo, and

Heung-Yeung Shum. Real-time texture synthesis by patch-based sampling. *ACM ToG*, 20(3):127–150, 2001.

[29] Huan Liu. Efficient mapping of range classifier into ternary-CAM. In *10th Symposium on High Performance Interconnects*, pages 95–100, 2002.

[30] Stuart P. Lloyd. Least squares quantization in PCM. pages 129–136, 1982.

[31] Chad R. Meiners, Alex X. Liu, and Eric Torng. TCAM Razor: A systematic approach towards minimizing packet classifiers in TCAMs. In *ICNP*, pages 266–275, 2007.

[32] Chad R. Meiners, Alex X. Liu, and Eric Torng. Bit Weaving: A non-prefix approach to compressing packet classifiers in TCAMs. In *ICNP*, pages 93–102, 2009.

[33] Chad R. Meiners, Alex X. Liu, and Eric Torng. Topological transformation approaches to optimizing TCAM-based packet classification systems. In *SIGMETRICS*, pages 73–84, 2009.

[34] Chad R. Meiners, Alex X. Liu, Eric Torng, and Jignesh Patel. Split: Optimizing space, power, and throughput for TCAM-based classification. In *ANCS*, pages 200–210, 2011.

[35] Masoud Moshref, Minlan Yu, Ramesh Govindan, and Amin Vahdat. DREAM: dynamic resource allocation for software-defined measurement. In *SIGCOMM*, pages 419–430, 2014.

[36] Marius Muja and David G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISAPP*, pages 331–340, 2009.

[37] Nvidia. Tesla K80 GPU accelerator, Nov. 2014. http://international.download.nvidia.com/pdf/kepler/BD-07317-001_v04.pdf.

[38] Aude Oliva and Antonio Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *IJCV*, 42:145–175, 2001.

[39] Open Networking Foundation. *OpenFlow Switch Specification Version 1.3.2*, April 2013.

[40] Guy Pazi, Anat Bremler-Barr, Rami Rivlin, and Dan Touitou. Protecting against distributed denial of service attacks, 2003. US Patent App. 10/232,993.

[41] Personal communication with engineers in networking companies, 2015.

[42] James Philbin, Ondrej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *CVPR*, pages 1–8, 2007.

[43] Radware. DefenseFlow - SDN applications and DDoS attack defense, 2014. http://www.radware.com/Products/DefenseFlow/.

[44] Radware. Tsunami SYN flood attack - a new trend in DDoS attacks?, October 2014. http://blog.radware.com/security/2014/10/tsunami-syn-flood-attack/.

[45] VC Ravikumar and Rabi N Mahapatra. TCAM architecture for IP lookup using prefix properties. *Micro, IEEE*, 24(2):60–69, 2004.

[46] Renesas Electronics America Inc. 20Mbit QUAD-search content addressable memory. http://www.renesas.com/products/memory/TCAM/index.jsp.

[47] Ori Rottenstreich and Isaac Keslassy. On the code length of TCAM coding schemes. In *ISIT*, pages 1908–1912, 2010.

[48] Ori Rottenstreich and Isaac Keslassy. Worst-case TCAM rule expansion. In *INFOCOM*, 2010.

[49] Ori Rottenstreich, Issac Keslassy, Avinatan Hassidim, Haim Kaplan, and Ely Porat. On finding an optimal TCAM encoding scheme for packet classification. In *INFOCOM*, 2013.

[50] Bryan C. Russell, Antonio Torralba, Kevin P. Murphy, and William T. Freeman. Labelme: A database and web-based tool for image annotation. *Intl. Journal of Computer Vision*, 77(1-3):157–173, 2008.

[51] Ryu SDN controller, 2014. http://osrg.github.io/ryu/.

[52] Hanan Samet. *Foundations of multidimensional and metric data structures*. Morgan Kaufmann, 2006.

[53] M. Said Seddiki, Muhammad Shahbaz, Sean Donovan, Sarthak Grover, Miseon Park, Nick Feamster, and Ye-Qiong Song. FlowQoS: QoS for the rest of us. In *HotSDN*, pages 207–208, 2014.

[54] Rajendra Shinde, Ashish Goel, Pankaj Gupta, and Debojyoti Dutta. Similarity search and locality sensitive hashing using ternary content addressable memories. In *SIGMOD*, pages 375–386, 2010.

[55] E. Spitznagel, D. Taylor, and J. Turner. Packet classification using extended TCAMs. In *ICNP*, 2003.

[56] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel. Fast and scalable layer four switching. In *SIGCOMM*, pages 191–202, 1998.

[57] J. Van Lunteren and T. Engbersen. Fast and scalable packet classification. *IEEE JSAC*, 21(4):560–571, 2003.

[58] Richard Wang, Dana Butnariu, and Jennifer Rexford. Openflow-based server load balancing gone wild. In *Hot-ICE*, 2011.

[59] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, pages 194–205, 1998.

[60] Kai Zheng, Hao Che, Zhijun Wang, Bin Liu, and Xin Zhang. DPPC-RE: TCAM-based distributed parallel packet classification with range encoding. *IEEE Trans. Computers*, 55(8):947–961, 2006.