Optimal Compression for Two-Field Entries in Fixed-Width Memories

Ori Rottenstreich, *Member, IEEE,* and Yuval Cassuto, *Senior Member, IEEE*

***Abstract*—Data compression is a well-studied (and well-solved) problem in the setup of long coding blocks. But important emerging applications need to compress data to memory words of small fixed widths. This new setup is the subject of this paper. In the problem we consider we have two sources with known discrete distributions, and we wish to find codes that maximize the success probability that the two source outputs are represented in L bits or less. A good practical use for this problem is a table with two-field entries that is stored in a memory of a fixed width L. Such tables of very large sizes are common in network switches/routers and in data-intensive machine-learning applications. After defining the problem formally, we solve it optimally with an efficient code-design algorithm. We also solve the problem in the more constrained case where a single code is used in both fields (to save space for storing code dictionaries). For both code-design problems we find decompositions that yield efficient dynamic-programming algorithms. With the help of an empirical study we show the success probabilities of the optimal codes for different distributions and memory widths. In particular, the study demonstrates the superiority of the new codes over existing compression algorithms.**

***Keywords*—Data compression, fixed-width memories, table compression, Huffman coding, network switches and routers.**

I. Introduction

In the best-known data-compression problem, a discrete distribution on source elements is given, and one wishes to find a representation for the source elements with minimum expected length. This problem was solved by the well-known Huffman coding scheme [3]. The Huffman coding scheme tackles the problem through an efficient recursive algorithm that relies on a tree structure and achieves optimality. Indeed, the Huffman code has found use in numerous communication and storage applications. Minimizing the expected length of the coded sequence emitted by the source translates to optimally low transmission or storage costs in those systems. However, there is an important setup in which minimizing the expected length does *not* translate to optimal improvement in system performance. This setup is *fixed-width memory*. Information is stored in a fixed-width memory in words of L bits each. A *word* is designated to store an *entry* of d *fields*[1], such that each field is drawn independently, following a known distribution for that field source. We distinguish the word as a physical memory unit storing actual bits, from the entry and its fields that are the logical information to be stored in it. In this case, the prime objective is to maximize the probability of representing the d outputs of the field sources within the memory word boundaries of L bits, rather than minimizing the expected length for each source.

The fixed-width memory setup is extremely useful in real applications. The most immediate use of it is in networking applications, wherein switches and routers rely on fast access to entries of very large tables [4]. The fast access requires simple word addressing, and the large table size requires good compression. In addition to this principal application, data-centric computing applications can similarly benefit from efficient compression of large data sets in fixed-width memories. In this paper we consider the compression of data entries with $d = 2$ fields into memory words of L bits, where L is a parameter. This special case of two fields is motivated by switch and router tables in which two-field tables are most common. Throughout the paper we assume that the element chosen for an entry field is drawn i.i.d. from a known distribution for that field, and that elements of distinct fields are independent of each other.

Our operational model for the compression assumes that success to fit an entry to L bits translates to fast data access, while failure results in a much slower access because a secondary slower memory is accessed to fit the entry. Therefore, we want to maximize the number of entries we succeed to fit, but do allow some (small) fraction of failures. Correspondingly, the performance measure we consider throughout the paper is P_{success}: the probability that the encoding of the two-field entry fits in L bits. That is, we seek a uniquely decodable encoding function that on some entries is allowed to declare *encoding failure*, but does so with the lowest possible probability for the source distributions. We emphasize that we do *not* allow decoding errors, and all entries that succeed encoding are recovered perfectly at read time.

Maximizing P_{success} is trivial when we encode the two fields jointly by a single code: we simply take the 2^L highest probabilities in the product distribution of the two fields, and represent each of these pairs with a codeword of length L. However, this solution has prohibitive cost because it requires a dictionary with size quadratic in the support of the individual

This paper solves a problem first formulated in the paper "Compression for Fixed-Width Memories" presented at the IEEE International Symposium on Information Theory (ISIT) 2013, Istanbul, Turkey [1]. O. Rottenstreich is with the Viterbi Department of Electrical Engineering and the Department of Computer Science, Technion – Israel Institute of Technology, Haifa Israel (e-mail: ori.rot@gmail.com). Y. Cassuto is with the Viterbi Department of Electrical Engineering, Technion – Israel Institute of Technology, Haifa Israel (e-mail: ycassuto@ee.technion.ac.il). This work was supported in part by the Israel Science Foundation (ISF), by the US-Israel Binational Science Foundation (BSF), and by the Intel Center for Computing Intelligence (ICRI-CI).

[1] In a field we refer to a part of the information of an entry rather than to an algebraic field.

distributions. For example, if each field takes one of 1000 values, we will need a dictionary of order 10^6 in the trivial solution. To avoid the space blow-up, we restrict the code to use dictionaries of sizes *linear in the distribution supports*. The way to guarantee linear dictionary sizes is by composing the entry code from two codes for the individual fields. Our task is to find two codes for the two fields, such that when the two codewords are placed together in the width-L word, the entry fields can be decoded unambiguously. The design objective for the codes is to maximize P_{success}: the probability that the above encoding of the entry has length of at most L bits. We first show by example that Huffman coding is *not* optimal for this design objective. Consider a data entry with two fields, where the value of the first field is one of 5 possible source elements $\{a, b, c, d, e\}$, and where the value of the second field is one of 3 elements $\{x, y, z\}$. The distributions on these elements are given in the two left columns of Table I.(A)–(B); the values of the two fields are drawn independently. We encode the two fields using two codes σ_1, σ_2, whose encoders are specified in the right columns of Table I.(A)–(B), respectively. The codewords of σ_1, σ_2 are concatenated, and this encoding needs to be stored in the memory word. Table I.(C) enumerates the $5 \cdot 3 = 15$ possible entries (combinations of values for the two fields), and for each gives its probability, its encoding, and the encoding length. The rightmost column of Table I.(C) indicates whether the encoding succeeds to fit in $L = 4$ bits ($\surd$), or not ($-$). The success probability of the pair of codes (σ_1, σ_2) is the sum of all probabilities in rows marked with $\surd$. This amounts to

$$P_{\text{success}} = 0.20 + 0.12 + 0.08 + 0.15 + 0.09 + 0.06 + 0.08 + 0.048 + 0.032 + 0.04 + 0.03 = 0.93.$$

It can be checked that the codes (σ_1, σ_2) with their above-specified encoders give better P_{success} than the success probability of the respective Huffman codes, which equals 0.78.

In the sequel we design code pairs (σ_1, σ_2), which together with an encoder and decoder we call *entry coding schemes*. We refer to an entry coding scheme as *optimal* if its encoder maximizes P_{success} among all entry coding schemes in its class. In the paper we find optimal coding schemes for two classes with different restrictions on their implementation cost: one is given in Section III and one in Section IV.

Finding codes that give optimal P_{success} cannot be done by known compression algorithms. Also, brute-force search for an optimal code has complexity that grows exponentially with the number of source elements. Even if we consider only *monotone encoders* – those which assign codewords of lengths non-increasing with the element probabilities – we can show that the number of codeword-length combinations for a prefix code grows asymptotically as at least ρ^n, for $\rho \triangleq \sqrt[3]{2}$ and n elements. The infeasibility of code search motivates the study in this paper toward devising efficient polynomial-time algorithms for finding optimal fixed-width codes.

For the case of entries with $d = 2$ fields, this paper solves the problem of optimal fixed-width compression. We present an efficient algorithm that takes any pair of element distributions and an integer L, and outputs an entry encoder with optimal P_{success}. The encoder has the unique-decodability property, and requires a dictionary whose size equals the sum of the element counts for the two fields. We also solve the problem for the more restrictive setup in which *the same code* needs to be used for both fields. This setup is motivated by systems with a more restricted memory budget, which cannot afford implementing two different encoding/decoding maps (and dictionaries) for the two fields. For both setups finding optimal codes is formulated as efficient dynamic-programming algorithms building on decompositions of the problems to smaller problems. As a building block, our solutions make use of properly designed prefix codes (codes in which no codeword is a prefix of another codeword), as well as other (non-prefix) types of codes.

TABLE I. EXAMPLE OF ENCODING FOR TWO FIELDS. I(A) AND I(B) LIST ENTRY DISTRIBUTIONS OF THE TWO FIELDS, AND THE TWO CODES σ_1 AND σ_2 CHOSEN FOR THEM. I(C) LISTS ALL COMBINATIONS OF VALUES FROM THE TWO FIELDS, THEIR ENCODING USING THE CONCATENATION (σ_1, σ_2), AND THE ENCODING LENGTHS. SPACES ARE PRESENTED FOR CONVENIENCE, IN REAL MEMORY THE CODEWORDS ARE CONCATENATED WITHOUT SPACES AND WITH TRAILING ZEROS IF NEEDED.

(A) First field with code σ_1

Element	Prob.	codeword
a	0.4	00
b	0.3	01
c	0.16	10
d	0.08	110
e	0.06	111

(B) Second field with code σ_2

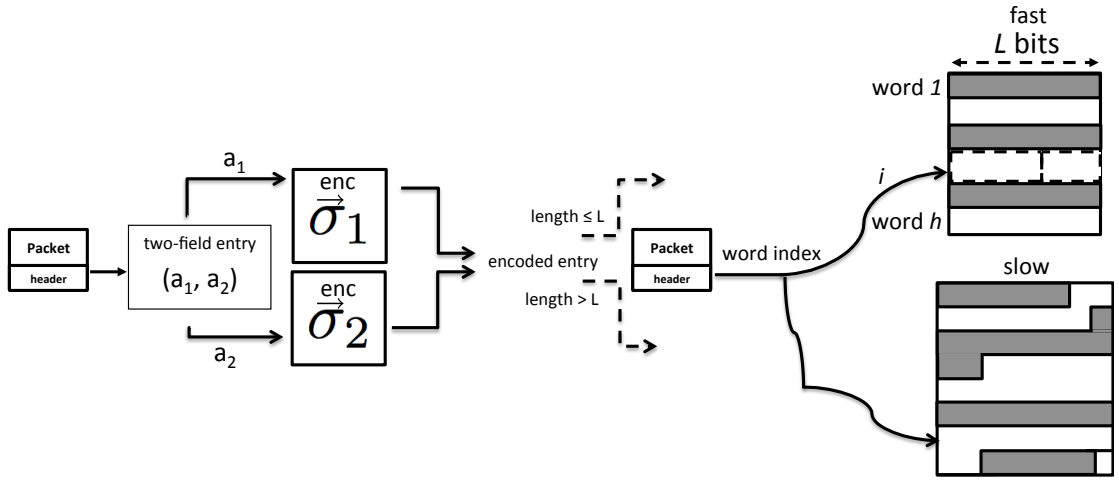
Element	Prob.	codeword
x	0.5	0
y	0.3	10
z	0.2	11

(C) Possible entries encoded by the encoding function $\Sigma = (\sigma_1, \sigma_2)$

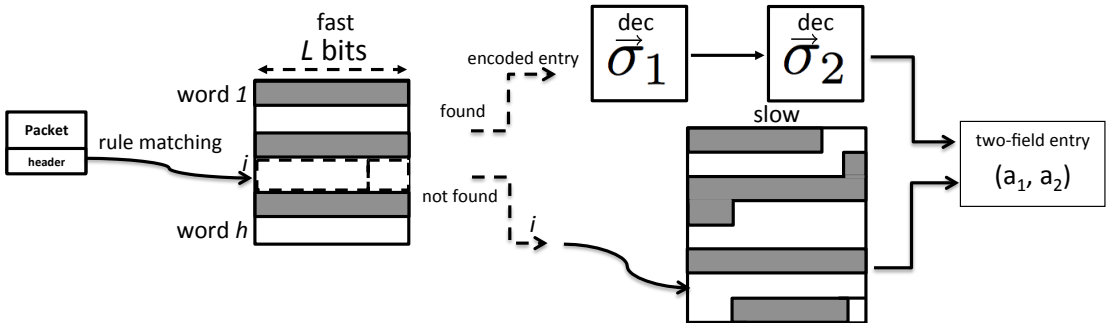
Entry	Prob.	Encoding	Width	Width $\leq (L = 4)$
(a, x)	0.20	00 0	3	$\surd$
(a, y)	0.12	00 10	4	$\surd$
(a, z)	0.08	00 11	4	$\surd$
(b, x)	0.15	01 0	3	$\surd$
(b, y)	0.09	01 10	4	$\surd$
(b, z)	0.06	01 11	4	$\surd$
(c, x)	0.08	10 0	3	$\surd$
(c, y)	0.048	10 10	4	$\surd$
(c, z)	0.032	10 11	4	$\surd$
(d, x)	0.04	110 0	4	$\surd$
(d, y)	0.024	110 10	5	-
(d, z)	0.016	110 11	5	-
(e, x)	0.03	111 0	4	$\surd$
(e, y)	0.018	111 10	5	-
(e, z)	0.012	111 11	5	-

Fig. I shows an illustration of a plausible realization of fixed-width compression in a networking application. First, Fig. I(a) illustrates the write operation where the entry is encoded and is written to a word in the fast SRAM memory if the encoded entry has a length of at most L bits. Otherwise, it is stored in the slow DRAM memory. Fig. I(b) shows the read operation where the encoded data is first searched in the fast SRAM memory. If it is not found, the slow DRAM memory is also accessed after the word index is translated.

Paper organization: In Section II, we present the formal model for the problem of compression for fixed-width memories. Setting up the problem requires some definitions extending classical single-source compression to 2 sources modeling the $d = 2$ fields in the entry. Another important feature in Section II is the notion of an encoder that is



(a) Write



(b) Read

Fig. 1. A plausible realization of a memory system with fixed-width compression. It includes a fast fixed-width (e.g., SRAM) memory with words of L bits, and an additional slower (e.g. DRAM) memory. Entries that can be encoded within L bits are stored in the fast memory while other entries are stored in the slow memory without the width limit. The encoding and decoding are performed based on the designed code pair (σ_1, σ_2).

allowed to fail on some of the input pairs. In Section III we derive an algorithm for finding an optimal coding scheme for fixed-width compression with $d = 2$ fields. The algorithm is composed of two main pieces: 1) an algorithm finding an optimal prefix code for the first field given the code for the second field, and 2) a code for the second field that is optimal for any code used in the first field. For the second field, it turns out that the code needs not be a prefix code, but it does need to satisfy a property which we call *padding invariance*. Next, in Section IV we study the special case where the two fields of an entry need to use the same code. The restriction to use one code renders the algorithm of Section III useless for this case. Hence we develop a completely different decomposition of the problem that can efficiently optimize a single code jointly for both fields. In Section V we present empirical results showing the success probabilities of the optimal codes for different values of the memory width L. The results show some significant advantage of the new codes over Huffman coding. In addition, they compare the performance of the schemes using two codes (Section III) to those that use a single code (Section IV). Finally, Section VI summarizes our results and discusses directions for future work.

Relevant prior work: There have been several extensions of the problem solved by Huffman coding that received attention in the literature [5]–[10]. These are not directly related to fixed-width memories, but they use clever algorithmic techniques that may be found useful for this problem too. The notion of fixed-width representations is captured by the work of Tunstall on variable-to-fixed coding [11], but without the feature of multi-field entries. The problem of fixed-width coding is also related to the problem of compression with low probability of buffer overflow [12], [13]. However, to the best of our knowledge the prior work only covers a single distribution and an asymptotic number of encoding instances (in our problem the number of encoding instances is a fixed d). The most directly related to the results of this paper is our prior work [14] considering a combinatorial version of this problem, where instead of source distributions we are given an instance with element pairs. This combinatorial version is less realistic for applications, because a new code needs to be designed for any particular instance of element pairs stored in the memory. More broadly, there is a great recent interest in size reduction of tables in Software-Defined Networking (SDN) applications. For example, memory-efficient representations of

routing tables is a topic of particular interest [17]–[20].

II. Terminology

In this section, we present the model definitions upon which we cast the formal problem formulation. Throughout the paper we assume that data placed in the memory come from known distributions, as defined in the following. Together with the allowed memory width, these distributions serve as the input to the code design. We also define the entry encoding and decoding functions as the components of two-field entry coding schemes. Lastly, we define the encoding success probability which is the design objective we consider along the paper, under various requirements on the encoding function.

Definition 1 (Element Distribution)**.** *An* element distribution $(S,P)\triangleq((s_1,\ldots,s_n),(p_1,\ldots,p_n))$ *is characterized by an (ordered) set of elements with their corresponding positive appearance probabilities. An element of S is drawn randomly (and independently between drawings) according to the distribution P, i.e.,* $\Pr(a=s_i)=p_i$*, with $p_i>0$ and $\Sigma_{i=1}^{n}p_i=1$.*

Throughout the paper we assume that the elements s_i are ordered in *non-increasing* order of their corresponding probabilities p_i. Given an element distribution, a *variable-length code* σ, which we call *code* for short, is a set of binary codewords of different lengths in general. The number of codewords in σ is denoted $|\sigma|$, and for $b\in\sigma$ the codeword length as a binary string is denoted $\ell(b)$. For a code σ we define the *monotone encoder* function $\vec{\sigma}(\cdot)$ mapping s_i to the i-th shortest codeword in σ, where ties between codewords are broken lexicographically. The *monotone decoder* function $\vec{\sigma}^{-1}(\cdot)$ is simply the inverse mapping of $\vec{\sigma}(\cdot)$ from codewords of σ to the s_i elements.

In this paper we are interested in coding data entries composed of *element pairs*, so we extend Definition 1 to distributions of two-element data entries. For short, we refer to a data entry simply as an *entry*.

Definition 2 (Entry Distribution)**.** *An* entry distribution $D\triangleq[(S_1,P_1),(S_2,P_2)]\triangleq\Big[((s_{1,1},\ldots,s_{1,n_1}),(p_{1,1},\ldots,p_{1,n_1})),\ ((s_{2,1},\ldots,s_{2,n_2}),\ (p_{2,1},\ldots,p_{2,n_2}))\Big]$ *is a juxtaposition of two element distributions. Each field in an entry is drawn randomly and independently according to its corresponding distribution in D, i.e., $P(a_1=s_{1,i},a_2=s_{2,i})=P(a_1=s_{1,i})\cdot P(a_2=s_{2,i})=p_{1,i}\cdot p_{2,i}$, with $p_{1,i},p_{2,i}>0$. The numbers of possible elements in the first and second field of an entry are $n_1\triangleq|S_1|$ and $n_2\triangleq|S_2|$, respectively.*

Example 1. *The entry distribution of the two fields illustrated in Table I can be specified as* $D=[(S_1,P_1),(S_2,P_2)]=\Big[((a,b,c,d,e),(0.4,0.3,0.16,0.08,0.06)),((x,y,z),(0.5,0.3,0.2))\Big]$.

Once we have defined the distributions governing the creation of entries, we can proceed to definitions addressing the encoding of such entries. In all our coding schemes, entries are encoded to bit strings of fixed length L (possibly with some padding), where L is a parameter equal to the width of the memory in use. An entry coding scheme is defined through its *encoding function*.

Definition 3 (Entry Encoding Function)**.** *An* entry encoding function Σ *is a mapping $\Sigma:(S_1,S_2)\to\{0,1\}^L\cup\{\perp\}$, where $\{0,1\}^L$ is the set of binary vectors of length L, and $\perp$ is a special symbol denoting encoding failure.*

The input to the encoding function are two elements, one for field 1 taken from S_1, and one for field 2 taken from S_2. In successful encoding, the encoder outputs a binary vector of length L to be stored in memory, and this vector is obtained uniquely for this input entry. The encoding function is allowed to fail, in which case the output is the special symbol $\perp$. When the encoding function fails, we assume that an alternative representation is found for the entry, and stored in a different memory not bound to the width-L constraint. This alternative representation is outside the scope of this work, but because we know it results in higher cost (in space and/or in time), our objective here is to minimize the failure probability of the encoding function. Before we formally define the failure probability, we give a definition of the decoding function matching the encoding function of Definition 3.

Definition 4 (Entry Decoding Function)**.** *An* entry decoding function Π *is a mapping $\Pi:\{0,1\}^L\to(S_1,S_2)$, such that if $\Sigma(a_1,a_2)=b\neq\perp$, then $\Pi(b)=(a_1,a_2)$.*

The definition of the decoding function is straightforward: it is the inverse mapping of the encoding function when encoding succeeds. With encoding and decoding in place, we turn to define the important measure of encoding success probability.

Definition 5 (Encoding Success Probability)**.** *Given an entry distribution $D=[(S_1,P_1),(S_2,P_2)]$ and an entry encoding function Σ, define the* encoding success probability *as*

$$P_{\text{success}}(D,\Sigma)\triangleq\Pr(\Sigma(a_1,a_2)\neq\perp),\tag{1}$$

where the probability is calculated over all pairs of (a_1,a_2) from (S_1,S_2) according to their corresponding probabilities (P_1,P_2).

Concluding this section, we define an *entry coding scheme* as a pair of an entry encoding function Σ and a matching entry decoding function Π, meeting the respective definitions above. Our principal objective in this paper is to find entry coding schemes that maximize the encoding success probability given the entry distribution and the output length L.

III. Optimal Efficient Entry Coding

In this section we develop a general entry coding scheme that yields optimal encoding success probability among a class of entry coding schemes with efficient encoding and decoding. The efficiency of the coding schemes we seek in the paper is defined next.

Definition 6 (Efficient Entry Coding Scheme)**.** *An entry coding scheme is called* efficient *if its encoding function Σ*

and decoding function Π *are limited to use* $O(n_1+n_2)$ *space to store the mappings.*

Limiting the encoding and decoding function to use $O(n_1+n_2)$ space mappings avoids "blowing up" the space requirement as a result of joining two fields in an entry. In particular, it excludes a practically infeasible solution of joining the two fields into one product code with an $O(n_1 \cdot n_2)$-space dictionary (when n_1, n_2 are ~ 100 each, $n_1 \cdot n_2$ becomes a significant amount of costly fast memory to deploy). To guarantee $O(n_1+n_2)$ space, we define an entry coding scheme in the sequel using a pair of codes: σ_1 for S_1, σ_2 for S_2, each using its own $O(n_1)$, $O(n_2)$ dictionary. Our objective is that σ_1 and σ_2 are chosen such that jointly they maximize the encoding success probability of the entry coding scheme. We further restrict ourselves to entry coding schemes we call *monotone*, as defined next. We subsequently show that this restriction is without loss of optimality.

Definition 7 (Monotone Entry Coding Scheme)**.** *An entry coding scheme with* $|S_1| = n_1$, $|S_2| = n_2$ *employing a pair of codes* (σ_1, σ_2) *with sizes* $n_1' \leq n_1$, $n_2' \leq n_2$ *is called* monotone *if its entry encoding function is*

$$\Sigma(a_1,a_2) \quad \begin{cases} \succeq (\vec{\sigma}_1(a_1), \vec{\sigma}_2(a_2)), & \text{if } a_1 \in \{s_{1,1},\ldots,s_{1,n_1'}\} \\ & \text{and } a_2 \in \{s_{2,1},\ldots,s_{2,n_2'}\} \\ =\perp, & \text{otherwise} \end{cases} \tag{2}$$

where $x \succeq (y_1, y_2)$ *means that each of* y_1, y_2 *appears in* x *as a sub-string. Recall that* $\vec{\sigma}_1(a_1), \vec{\sigma}_2(a_2)$ *are the monotone encoders defined in Section II and* $\perp$ *is a special symbol denoting encoding failure.*

In words, a monotone entry coding scheme employing the codes (σ_1, σ_2) uses their monotone encoders $(\vec{\sigma}_1, \vec{\sigma}_2)$, and their codewords are mapped to the first n_1', n_2' elements in S_1, S_2, respectively. We next prove the intuitive fact that using monotone encoders $(\vec{\sigma}_1, \vec{\sigma}_2)$ in the entry encoding function is without loss of optimality.

Proposition 1. *For any entry encoding function* Σ *employing a pair of codes* (σ_1, σ_2) *with their encoders* $\tilde{\sigma}_1(\cdot)$, $\tilde{\sigma}_2(\cdot)$, P_{success} *is maximized when* (i) *for* $t \in \{1,2\}$*, the codewords of* σ_t *are assigned to the first* n_t' *elements of* S_t*.* (ii) *for* $t \in \{1,2\}$*, the inequality* $i < i'$ *implies that* $\ell(\tilde{\sigma}_t(s_{t,i})) \leq \ell(\tilde{\sigma}_t(s_{t,i'}))$ *for two elements* $s_{t,i}, s_{t,i'} \in S_t$*.*

Proof: We show how to convert any Σ employing (σ_1, σ_2) to one using monotone encoders on the first elements of S_1, S_2, without reducing P_{success}. For $t \in \{1,2\}$ consider two arbitrary indices i, i' that satisfy $i < i'$; then necessarily $p_{t,i} \geq p_{t,i'}$. If codewords are assigned to the two elements and $\ell(\tilde{\sigma}_t(s_{t,i})) > \ell(\tilde{\sigma}_t(s_{t,i'}))$, we can exchange the codewords of $s_{t,i}, s_{t,i'}$ in the mapping. With this change, an entry (a_1, a_2) with $a_t = s_{t,i}$ is encoded successfully after the change if the corresponding entry with $a_t = s_{t,i'}$ was encoded successfully before the change. Then, we deduce that such a change cannot decrease P_{success}, showing that monotone encoders maximize P_{success}. In addition, if the domain of $\tilde{\sigma}_t(\cdot)$ is not taken as the first n_t' elements of S_t, then we can replace $s_{t,i'}$ by $s_{t,i}$, $i < i'$ without reducing P_{success}. ∎

Not emphasized in Definition 7, but crucial for the optimal entry coding schemes coming next, is the use of codes σ_1, σ_2 with potentially fewer codewords than the respective element-set sizes $|S_1|$, $|S_2|$. This is necessary for optimality, as not assigning codewords to the lowest-probability elements may be beneficial to P_{success} by allowing higher-probability elements to be assigned shorter codewords.

A. Uniquely decodable entry encoding

The requirement from the decoding function to invert the encoding function for all successful encodings implies that the encoding function must be *uniquely decodable*, that is, that any encoder-output bit-vector b represents a unique entry (a_1, a_2). In efficient entry coding schemes our way to achieve unique decodability is by requiring the code σ_1 to be a *prefix code*, thereby allowing the entry decoding function to "parse" the vector b to its two parts. We next recall the definition of prefix codes.

Definition 8 (Prefix Code)**.** *A code* σ *is called a* prefix code *if in its codeword set no codeword is a prefix (start) of any other codeword.*

Thanks to using a prefix code for S_1, the decoder can read the length-L vector b from left to right and unequivocally find the last bit representing a_1, and in turn the remaining bits representing a_2 (plus some potential padding). Hence in the remainder of the section we restrict the entry coding schemes to have a constituent prefix code for their first field. This restriction of the encoder to be prefix is reminiscent of the classical result in information theory by Kraft-McMillan [15] that prefix codes are sufficient for optimal fixed-to-variable uniquely decodable compression. In our case we cannot prove that entry coding schemes with prefix σ_1 are always sufficient for optimality, but we also could not find better coding schemes without prefix σ_1 that still attain the $O(n_1+n_2)$-space efficiency. To be formally correct, we call an entry coding scheme optimal in this section if it has the maximum success probability among coding schemes whose first code is prefix.

Unique decodability requires the second-field code σ_2 to have a weaker property than prefix, which we call *padding invariance*.

Definition 9 (Padding-Invariant Code)**.** *A code* σ *is called a padding-invariant code if after eliminating the trailing (last) zero bits from its codewords (possibly including the empty codeword* $\emptyset$ *of length 0),* $|\sigma|$ *distinct binary vectors are obtained.*

Example 2. *Consider the following codes:* $\sigma = \{0, 11, 100, 101\}$*,* $\sigma' = \{\emptyset, 1, 110\}$ *and* $\sigma'' = \{1, 10, 00\}$*. The code* σ *is a prefix code since none of its codewords is a prefix of another codeword. It is also a padding-invariant code since after eliminating the trailing zeros, the four distinct codewords* $\emptyset$ *(the empty codeword),* $11, 1$ *and* 101 *are obtained. While* σ' *is also a padding-invariant code (the codewords* $\emptyset, 1, 11$ *are distinct), it is not a prefix code since*

1 is a prefix of 110 and $\emptyset$ is a prefix of 1 and 110. The code σ'' is neither a prefix code (1 is a prefix of 10) nor a padding-invariant code (the codeword 1 is obtained after eliminating the trailing zero bits of 1 and 10).

It is easy to see that in any padding-invariant code we can remove the trailing zeros from all codewords and remain with a padding-invariant code of the same size (and codewords not longer than in the original code). Hence without loss of generality we can assume that there are no trailing zeros in a padding-invariant code. We now show how a prefix code σ_1 and a padding-invariant code σ_2 are combined to obtain a uniquely-decodable entry encoding function. The following is not yet an explicit construction, but rather a general construction method we later specialize to obtain an optimal construction.

Construction 0. *Let σ_1 be a prefix code and σ_2 be a padding-invariant code. The vector $b \triangleq \Sigma(a_1, a_2)$ output by the entry encoding function $\Sigma = (\sigma_1, \sigma_2)$ is defined as follows. The vector b is formed by concatenating $\vec{\sigma}_2(a_2)$ to the right of $\vec{\sigma}_1(a_1)$, and if needed, padding the resulting binary vector with zeros on the right to get L bits total. If the concatenated two codewords exceed L bits before padding, the entry encoding function returns $\perp$.*

We show that because σ_1 is a prefix code while σ_2 is a padding-invariant code (not necessarily prefix), the entry encoding function of Construction 0 is uniquely decodable. It is easy to see also that padding invariance of the second code is necessary to guarantee that. We use the symbol of $\cdot$ to represent the concatenation operator of two strings.

Property 1. *Let $\Sigma = (\sigma_1, \sigma_2)$ be an entry encoding function specified by Construction 0. Let $x \triangleq (x_1, x_2)$, $y \triangleq (y_1, y_2)$ be two entries composed of elements $x_1, y_1 \in S_1$, $x_2, y_2 \in S_2$. Let $\Sigma(x) \triangleq \vec{\sigma}_1(x_1) \cdot \vec{\sigma}_2(x_2) \cdot 0 \cdot \ldots \cdot 0$ and $\Sigma(y) \triangleq \vec{\sigma}_1(y_1) \cdot \vec{\sigma}_2(y_2) \cdot 0 \cdot \ldots \cdot 0$ be the two length-L vectors output by the entry encoding function Σ. If $\Sigma(x) = \Sigma(y)$, then necessarily $x = y$, i.e., $x_1 = y_1, x_2 = y_2$.*

To show the correctness of this property, we prove that the decoding of an encoded entry is unique.

Proof: We first observe that there is a single element $u_1 \in S_1$ whose encoding $\vec{\sigma}_1(u_1)$ is a prefix of $\Sigma(x) = \Sigma(y)$. Assume the contrary, and let $u, u' \in S_1$ be two (distinct) elements such that $\vec{\sigma}_1(u)$ and $\vec{\sigma}_1(u')$ are both prefixes of $\Sigma(x) = \Sigma(y)$. It then follows that either $\vec{\sigma}_1(u)$ is a prefix of $\vec{\sigma}_1(u')$ or $\vec{\sigma}_1(u')$ is a prefix of $\vec{\sigma}_1(u)$. Since $u \neq u'$ and σ_1 is a prefix code, both options are impossible, and we must have that there is indeed a single element $u_1 \in S_1$ whose encoding $\vec{\sigma}_1(u_1)$ is a prefix of $\Sigma(x) = \Sigma(y)$. Thus necessarily $\vec{\sigma}_1(x_1) = \vec{\sigma}_1(y_1)$ and $x_1 = y_1$. By eliminating the first identical bits that represent $\vec{\sigma}_1(x_1) = \vec{\sigma}_1(y_1)$ from $\Sigma(x) = \Sigma(y)$, we are left with the encoding of the second field, possibly including trailing zero bits. By the properties of the padding-invariant code σ_2, it has a single element that maps to the bits remaining after eliminating some trailing zero bits. We identify this codeword as $\vec{\sigma}_2(x_2) = \vec{\sigma}_2(y_2)$, and deduce the only possible element $x_2 = y_2$ of this field that has this codeword. ■

To decode an encoded entry composed of the encodings $\vec{\sigma}_1(x_1)$, $\vec{\sigma}_2(x_2)$, possibly with some padded zero bits, a decoder can simply identify the single element $(u_1 = x_1) \in S_1$ whose encoding $\vec{\sigma}_1(u_1 = x_1)$ is a prefix of the encoded entry. As σ_1 is a prefix code, there cannot be more than one such element. The decoder can then identify the beginning of the second part of the encoded entry, remove all trailing zeros, and get x_2 by invoking $\vec{\sigma}_2^{-1}(\cdot)$.

Toward finding an entry coding scheme with maximal encoding-success probability, we would now like to find codes σ_1 and σ_2 for Construction 0 that are optimal given an entry distribution D and the output length L. Since the direct joint computation of optimal σ_1 and σ_2 codes is difficult (and requires exponential complexity for brute-force search), we take a more indirect approach to achieve optimality. 1) we first derive an efficient algorithm to find an optimal prefix code for S_1 given a code for S_2, and then 2) we find a (padding-invariant) code for S_2 that is universally optimal for any code used for S_1. This way we reduce the joint optimization of σ_1, σ_2 (hard) to a conditional optimization of σ_1 given σ_2 (easier). We prove that this conditionally optimal σ_1 is also unconditionally optimal. To do so, we show that the code σ_2 assumed in the conditioning is optimal for any choice of σ_1. These two steps are detailed in the next two sub-sections, and later used together to establish an optimal coding scheme.

B. An optimal prefix code for S_1 conditioned on a code for S_2

Our objective in this sub-section is to find a prefix code σ_1 that maximizes the encoding success probability given a code σ_2 for the second field. We show constructively that this task can be achieved with an efficient algorithm.

We denote $W_t \triangleq \lceil \log_2(n_t) \rceil$ for $t \in \{1, 2\}$ and $W \triangleq W_1 + W_2$. Finding an optimal coding scheme when $L \geq W$ is easy. In that case we could allocate W_1 bits to σ_1 and W_2 bits to σ_2 and apply two independent fixed-length codes with a total of W bits and obtain a success probability 1. Thus we focus on the interesting case where $L \leq W - 1$.

Given a code $\sigma_2 = \sigma$ for S_2, we show a polynomial-time algorithm[2] that finds an optimal conditional prefix code σ_1 for S_1. This code σ_1 will give an entry encoding function maximizing the probability $P_{\text{success}}(D, \Sigma = (\sigma_1, \sigma_2 = \sigma))$ given $\sigma_2 = \sigma$, when σ_1 is restricted to be prefix. Then, we also say that $\Sigma = (\sigma_1, \sigma_2)$ with its corresponding decoding function Π is an optimal conditional coding scheme.

To build the code σ_1 we assign codewords to n_1' elements of S_1, where $n_1' \leq n_1$. Clearly, all such codewords have lengths of at most L bits. Recall that the length of a binary string x is denoted $\ell(x)$. Since for every element $a \in S_1$ which is assigned a codeword, the code σ_1 satisfies $\ell(\vec{\sigma}_1(a)) \leq L$, it holds that $2^{-\ell(\vec{\sigma}_1(a))}$ must be an integer multiple of 2^{-L}. We define the *weight* of a codeword of length ℓ_0 as the number of units of 2^{-L} in $2^{-\ell_0}$, denoted by $N_{\ell_0} \triangleq 2^{-\ell_0}/2^{-L} = 2^{L-\ell_0}$. The $n_1 - n_1'$ elements of S_1 not represented by σ_1 are said

[2] An analysis of the algorithm complexity is provided later.

to have length $\ell_0 \triangleq L+1$. This symbolic value means that entries with such elements cannot be encoded successfully, and accordingly they have a codeword weight of zero. A prefix code exists with prescribed codeword lengths if they satisfy Kraft's inequality [16]. In our terminology, this means that the sum of weights of the codewords of σ_1 need to be at most 2^L.

Definition 10. *For $k \in [0, n_1]$, consider entries composed of an element from the first k elements of S_1 (i.e., highest probability elements of S_1), and an arbitrary element of S_2. For $N \in [0, 2^L]$ and $k \in [0, n_1]$, we denote by $F(k, N)$ the maximal sum of probabilities of such entries that can be encoded successfully by a prefix code σ_1 whose sum of weights for the first k codewords is at most N. Formally,*

$$F(k,N) \triangleq \max_{\sigma_1 : \left(\sum_{j=1}^{k} N_{\ell(\vec{\sigma}_1(s_{1,j}))} \le N\right)} \Bigg(\sum_{i=1}^{k} \sum_{j=1}^{n_2} p_{1,i} \cdot p_{2,j} \cdot I\Big[\ell(\vec{\sigma}_1(s_{1,i})) + \ell(\vec{\sigma}_2(s_{2,j})) \le L\Big]\Bigg), \quad (3)$$

where $I[\cdot]$ is the indicator function. Note that $F(k, N)$ depends on the conditioned σ_2, but we keep this dependence implicit to simplify notation.

The following theorem relates the maximal success probability of a conditional coding scheme and the function $F(k, N)$.

Theorem 2. *The maximal encoding success probability of a conditional entry coding scheme is given by*

$$\max_{\sigma_1} P_{\text{success}}(D, \Sigma = (\sigma_1, \sigma_2 = \sigma_2)) = F(k = n_1, N = 2^L).$$

Proof: To satisfy Kraft's inequality we should limit the sum of weights N to 2^L. In addition, the success probability of the coding scheme with an encoding function Σ is calculated based on entries with any of the n_1 elements of S_1. ■

We next show how to compute $F(k, N)$ efficiently for all k, N, in particular for $k = n_1$, $N = 2^L$ that yield the optimal conditional σ_1. To do that, we use the following recursive formula for $F(k, N)$. First note the boundary cases $F(k = 0, N) = 0$ for $N \ge 0$ and $F(k, N) = -1$ for $N < 0$ (this means an invalid code). We can now present the formula of $F(k, N)$ that calculates its values for k based on the values of the function for $k-1$ and $N' \le N$.

Lemma 3. *The function $F(k, N)$ satisfies for $N \ge 0, k \ge 1$*

$$F(k,N) = \max\Bigg(\max_{\ell_0 \in [1,L]} \Bigg(F(k-1, N - N_{\ell_0}) + p_{1,k} \cdot \sum_{i=1}^{n_2} p_{2,i} \cdot I\Big[\ell_0 + \ell(\vec{\sigma}_2(s_{2,i})) \le L\Big]\Bigg), F(k-1, N)\Bigg). \quad (4)$$

Proof: The optimal code that attains $F(k, N)$ either assigns a codeword to $s_{1,k}$ or does not. The two arguments of the outer max function in (4) are the respective success probabilities for these two choices. In the former case we consider all possible lengths of the codeword of $s_{1,k}$. A codeword length of ℓ_0 reduces the available sum of weights for the first $k-1$ elements by $N_{\ell_0} = 2^{L-\ell_0}$. In addition, an entry $(s_{1,k}, s_{2,i})$ contributes to the success probability the value $p_{1,k} \cdot p_{2,i}$ if its encoding width (given ℓ_0) is at most L. In the latter case the k^{th} element does not contribute to the success probability and has no weight, hence $F(k, N) = F(k-1, N)$ in this case. ■

Finally, the pseudocode of the dynamic-programming algorithm that finds the optimal conditional code based on the above recursive formula is given in Algorithm 1. It iteratively calculates the values of $F(k, N)$. It also uses a vector $Q(k, N)$ to represent the codeword lengths for the first at most k elements of S_1 in a solution achieving $F(k, N)$.

Time Complexity: By the above description, there are n_1 iterations and in every iteration $O(2^L)$ values are calculated, each by considering $O(L)$ sums of n_2 elements. It follows that the time complexity of the algorithm is $O(n_1 \cdot 2^L \cdot L \cdot n_2) = O(n_1^2 \cdot n_2^2 \cdot L)$, which is polynomial in the size of the input. The last equality follows from the fact that $L < \lceil \log_2(n_1) \rceil + \lceil \log_2(n_2) \rceil$ in a non-trivial instance where some entries fail encoding.

Algorithm 1: Optimal Conditional Prefix Code

input : Entry distribution D, memory width L, code σ_2
output: Conditionally optimal prefix code σ_1
initialization:
foreach $N \in [0, 2^L]$ **do**
$F(k=0, N) := 0, Q(k=0, N) := ()$;
foreach $N \in [-2^L, -1]$, $k \in [0, n_1]$ **do** $F(k, N) := -1$;
intermediate solutions:
for $k = 1 : n_1$ **do**
 for $N = 0 : 2^L$ **do**
 $F(k,N) := \max\Big(\max_{\ell_0 \in [1,L]} \Big(F(k-1, N - N_{\ell_0}) + p_{1,k} \cdot \sum_{i=1}^{n_2} p_{2,i} \cdot I\Big[\ell_0 + \ell(\vec{\sigma}_2(s_{2,i})) \le L\Big]\Big), F(k-1, N)\Big)$
 if outer max attained by first argument **then**
 $Q(k,N) := (Q(k-1, N - N_{\ell_0^*}), \ell_0^*)$, where ℓ_0^* is the ℓ_0 that attains the inner max
 else
 $Q(k,N) := (Q(k-1, N), L+1)$
 end
 end
end
Calculate prefix code σ_1 based on codeword lengths given by $Q(k = n_1, N = 2^L)$
output results:
Encoding success probability
$P_{\text{success}} := F(k = n_1, N = 2^L)$
Optimal conditional prefix code σ_1

C. A universally optimal code for S_2

We now develop the second component required to complete an unconditionally optimal entry coding scheme: a code

for the S_2 elements that is optimal for any code σ_1 used for the S_1 elements.

The optimality of the code to be specified for S_2 will be established by showing that it attains the following upper bound on the success probability.

Proposition 4. *Given any code σ_1, the encoding success probability of any entry encoding function is bounded from above as follows.*

$$P_{\text{success}} \leq \sum_{i=1}^{n_1'} p_{1,i} \sum_{j=1}^{2^{L-\ell(\vec{\sigma}_1(s_{1,i}))}} p_{2,j}, \tag{5}$$

where n_1' is the highest index of an element $s_{1,i}$ that is assigned a codeword by σ_1.

Proof: First by the monotonicity property proved in Proposition 1, the codewords of σ_1 are assigned to elements $s_{1,i}$ with indices $i \in \{1, \ldots, n_1'\}$, for some $n_1' \leq n_1$. Hence for indices greater than n_1' the success probability is identically zero. Given an element $s_{1,i}$ with a σ_1 codeword of length ℓ_i, at most $2^{L-\ell_i}$ elements of S_2 can be successfully encoded with it in an entry. So the inner sum in (5) is the maximal success probability given $s_{1,i}$. Summing over all i and multiplying by $p_{1,i}$ gives the upper bound. ∎

It turns out that there exists a padding-invariant code $\hat{\sigma}_2$ that attains the upper bound of Proposition 4 with equality for any code σ_1. Let $\hat{\sigma}_2$ be the code with n_2 codewords where the j-th codeword is the shortest binary representation of $j-1$ for $j \geq 2$, and $\emptyset$ for $j = 1$. The binary representation is put in the codeword from left to right, least-significant bit (LSB) first. Then we have the following.

Proposition 5. *Given any code σ_1, the encoding success probability of $\hat{\sigma}_2$ is*

$$P_{\text{success}} = \sum_{i=1}^{n_1'} p_{1,i} \sum_{j=1}^{2^{L-\ell(\vec{\sigma}_1(s_{1,i}))}} p_{2,j}, \tag{6}$$

where n_1' is the highest index of an element $s_{1,i}$ that is assigned a codeword by σ_1.

Proof: If the codeword of σ_1 uses ℓ bits, there are $L-\ell$ bits left vacant for σ_2. The mapping specified for $\hat{\sigma}_2$ allows encoding successfully the first $2^{L-\ell}$ elements of S_2, which gives the stated success probability. ∎

In particular, when the encoding of σ_1 has length L, the single element $s_{2,1}$ is encoded successfully by the empty codeword $\emptyset$. Other examples are the two codewords ($\emptyset$, 1) when $\ell = L-1$, and the four codewords ($\emptyset$, 1, 01, 11) when $\ell = L-2$. It is clear that the code $\hat{\sigma}_2$ is padding invariant, because its codewords are minimal-length binary representations of integers. Now we are ready to specify the optimal entry coding scheme in the following.

Construction 1. *Given L and D, let $\hat{\sigma}_1$ be the prefix code obtained by applying Algorithm 1 on the code $\hat{\sigma}_2$. Then the entry encoding function $\hat{\Sigma} = (\hat{\sigma}_1, \hat{\sigma}_2)$ is defined by applying Construction 0 on $\hat{\sigma}_1$, $\hat{\sigma}_2$.*

Algorithm 2:

input : Entry distribution D, memory width L
output: Optimal entry coding scheme
calculation:
Calculate padding invariant code $\sigma_2{:=}\hat{\sigma}_2$ by assigning the n_2 elements the codewords that correspond to $[0, n_2-1]$
Calculate prefix code σ_1 by Algorithm 1 given the code σ_2
output results:
An optimal entry coding scheme with an encoding function $\Sigma = (\sigma_1, \sigma_2)$

Theorem 6. *For any entry distribution D and a memory width L, Construction 1 gives an optimal entry coding scheme, that is, a coding scheme that maximizes the success probability among all uniquely-decodable coding schemes with a prefix code in the first field.*

From Theorem 6 we can readily obtain an efficient algorithm finding an optimal two-field entry encoding, which is given in Algorithm 2.

For the special case when σ_2 is $\hat{\sigma}_2$, the recursive formula for the calculation of the function $F(k, N)$ can be simplified as follows.

Lemma 7. *When σ_2 is $\hat{\sigma}_2$, the function $F(k, N)$ satisfies for $N \geq 0, k \geq 1$*

$$F(k,N) = \max\left(\max_{\ell_0 \in [1,L]} \left(F(k-1, N-N_{\ell_0}) + p_{1,k} \cdot \sum_{i=1}^{\min(n_2, 2^{L-\ell_0})} p_{2,i} \right), F(k-1,N) \right). \tag{7}$$

It is easily seen that (7) is obtained from (4) by replacing the indicator function with the partial sum that accommodates all the S_2 elements that have a short enough representation to fit alongside the S_1 element.

The following example illustrates Construction 1 on the entry distribution from the Introduction.

Example 3. *Consider the entry distribution $D \triangleq \Big[((a,b,c,d,e), (0.4, 0.3, 0.16, 0.08, 0.06)), ((x,y,z), (0.5, 0.3, 0.2))\Big]$ from Table I with $n_1 = 5, n_2 = 3$. The width parameter is $L = 4$. For the ordered set $S_2 = (x,y,z)$, we select the code $\hat{\sigma}_2$ by mapping $s_{2,1}$ to $\emptyset$ and $s_{2,j}$ (for $j = 2, 3$) to the shortest binary representation of $j-1$. Then, $\vec{\hat{\sigma}}_2(x) = \emptyset, \vec{\hat{\sigma}}_2(y) = 1, \vec{\hat{\sigma}}_2(z) = 01$ and $\ell(\vec{\hat{\sigma}}_2(x)) = 0, \ell(\vec{\hat{\sigma}}_2(y)) = 1, \ell(\vec{\hat{\sigma}}_2(z)) = 2$. The code $\hat{\sigma}_2$ is a padding-invariant code. To get the prefix code $\hat{\sigma}_1$, we apply Algorithm 1 on the code $\hat{\sigma}_2$. We recursively calculate the values of $F(k,N)$ and $Q(k,N)$ for $k \in [1, n_1 = 5]$, $N \in [0, 2^L = 16]$. In particular, for each value of k the values are calculated based on the previous value of k. The values are listed in Table II. Each column describes a different value of k. (Whenever values of F and Q are not shown, a specific*

value of N does not improve the probability achieved for a smaller value of N in the same column.) The value of N implies a restriction on the values of the codeword lengths. If the lengths of the codewords are described by a set Q, they must satisfy $\Sigma_{\ell \in Q} 2^{-\ell}/2^{-L} \leq N$, i.e., $\Sigma_{\ell \in Q} 2^{-\ell} \leq N/16$.

We first explain the values for $k = 1$, considering the contribution to the success probability of data entries with $a \in S_1$ as the first element. This happens w.p. $p_{1,1} = 0.4$. For $N = 1$ we must have $Q(1,1) = (4)$, i.e., the element a is assigned a codeword of length 4. Then, there is a single pair (a, x) that can be encoded successfully and P_{success} given by $F(1,1)$ is $0.4 \cdot 0.5 = 0.2$. Likewise, for $N = 2$, we can have a codeword of length 3 for a and the two pairs $(a, x), (a, y)$ can be encoded within $L = 4$ bits, such that $F(1,2) = 0.4 \cdot (0.5 + 0.3) = 0.32$. If $N = 3$ we cannot further decrease the codeword length and improve the success probability. For $N = 4$ we can have a codeword length of 2 bits, as described by $Q(1,4) = (2)$. This enables encoding successfully the three pairs $(a, x), (a, y), (a, z)$ with a success probability of 0.4 as given by $F(1,4)$. The values for larger values of k are calculated in a similar manner based on the recursive formulas. The optimal codeword lengths for $\hat{\sigma}_1$ are given by $Q(k = n_1 = 5, N = 2^L = 16) = (2, 2, 2, 3, 3)$. This enables to encode successfully all pairs except $(d, z), (e, z)$, achieving $P_{\text{success}} = 0.972$ as given by $F(k = 5, N = 16)$.

Finally, by applying Construction 0 on $\hat{\sigma}_1$, $\hat{\sigma}_2$ we obtain the entry encoding function $\hat{\Sigma} = (\hat{\sigma}_1, \hat{\sigma}_2)$.

Non-binary alphabet: While the scheme was presented for the binary case, it can be easily generalized to *any (finite) α-ary alphabet A*. Clearly the assumption on the monotonicity of the codes applies also here. The construction of the scheme is similar. Consider for instance an alphabet with the symbols $0, 1, 2$. First we can design a padding-invariant code for the second field. The code has the codewords $\emptyset$, 1, 2, 01, 11, 21, 02, 12, 22, 001, etc, which are simply given by the representation with the alphabet of the integer numbers as for the binary case. Then, we can derive a conditionally optimal code for the first field. This can simply rely on a generalization of the Kraft's inequality for a non-binary alphabet, saying that there exists a set of prefix-free codewords of lengths $\ell_1, \ell_2, \ldots, \ell_n$ in the alphabet A if the inequality $\sum_{i=1}^{n} \alpha^{-\ell_i} \leq 1$ holds. By defining the weight of a codeword of length ℓ_0 as $N_{\ell_0}^A \triangleq \alpha^{L-\ell_0}$, we can apply the dynamic-programming algorithm in Algorithm 1 for finding the optimal conditional code for the first field, allowing a maximal total weight of $N = \alpha^L$.

IV. Optimal Entry Coding with the Same Code for Both Fields

In this section we move on to study the problem of entry coding schemes for the special case where we require that both fields use the same code. The coding scheme from Section III used two codes, one for each of the fields. Moving to use the same code for both fields has several motivations. First, it reduces the implementation complexity by approximately halving the space required to store the encoding and decoding maps (dictionaries). For example, in the optimal coding scheme from Section III the encoder needs two different mappings from source elements to codewords, and the decoder needs two different mappings (and efficient lookups) to convert an encoded entry to a pair of elements. Second, using the same code allows moving and copying coded elements between memory words and fields without re-coding. And third, working with one (prefix) code is natural toward extending the scheme to more than two fields in future work, because then the prefix property is needed in all fields except the last one.

TABLE II. The values of $F(k, N)$ (top of each pair in the table) and $Q(k, N)$ (bottom) for $k \in [1, n_1 = 5]$, $N \in [0, 2^L = 16]$ for Example 3 with $L = 4, n_1 = 5, n_2 = 3$. The optimal value of P_{success} is given by $F(5, 16)$ and the codeword lengths for σ_1 by $Q(5, 16)$.

	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
$N=0$	0	0	0	0	0
	(-)	(-,-)	(-,-,-)	(-,-,-,-)	(-,-,-,-,-)
$N=1$	0.2	0.2	0.2	0.2	0.2
	(4)	(4,-)	(4,-,-)	(4,-,-,-)	(4,-,-,-,-)
$N=2$	0.32	0.35	0.35	0.35	0.35
	(3)	(4,4)	(4,4,-)	(4,4,-,-)	(4,4,-,-,-)
$N=3$		0.47	0.47	0.47	0.47
		(3,4)	(3,4,-)	(3,4,-,-)	(3,4,-,-,-)
$N=4$	0.4	0.56	0.56	0.56	0.56
	(2)	(3,3)	(3,3,-)	(3,3,-,-)	(3,3,-,-,-)
$N=5$			0.64	0.64	0.64
			(3,3,4)	(3,3,4,-)	(3,3,4,-,-)
$N=6$		0.64	0.688	0.688	0.688
		(2,3)	(3,3,3)	(3,3,3,-)	(3,3,3,-,-)
$N=7$			0.72	0.728	0.728
			(2,3,4)	(3,3,3,4)	(3,3,3,4,-)
$N=8$		0.7	0.768	0.768	0.768
		(2,2)	(2,3,3)	(2,3,3,-)	(2,3,3,-,-)
$N=9$			0.78	0.808	0.808
			(2,2,4)	(2,3,3,4)	(2,3,3,4,-)
$N=10$			0.828	0.832	0.838
			(2,2,3)	(2,3,3,3)	(2,3,3,4,4)
$N=11$				0.868	0.868
				(2,2,3,4)	(2,2,3,4,-)
$N=12$			0.86	0.892	0.898
			(2,2,2)	(2,2,3,3)	(2,2,3,4,4)
$N=13$				0.9	0.922
				(2,2,2,4)	(2,2,3,3,4)
$N=14$				0.924	0.94
				(2,2,2,3)	(2,2,3,3,3)
$N=15$					0.954
					(2,2,2,3,4)
$N=16$				0.94	0.972
				(2,2,2,2)	(2,2,2,3,3)

Throughout this section we assume that both fields have the same distribution, but the results can be extended to the case where a single code is sought for distinct distributions (we comment about this extension at the closing of the section). Formally, in this section our problem is to efficiently design a single code σ that offers optimal encoding success probability in a width-L memory. In the special case of a single distribution we have an element distribution $(S, P) = ((s_1, \ldots, s_n), (p_1, \ldots, p_n))$, and the entry distribution is $D = [(S, P), (S, P)]$. Given (S, P), an optimal code σ will be found for use in both fields, and σ will be a prefix code so we can parse the two fields of the entry.

A. Observations on the problem

Before moving on to solve the problem, it will be instructive to first understand the root difficulty in restricting both fields

to use the same code. If we try to extend the dynamic-programming solution of Section III to the single-distribution, single-code case, we get the following maximization problem

$$F(k,N) = \max_{\sigma:\left(\sum_{j=1}^{k} N_{\ell(\vec{\sigma}(s_j))} \le N\right)} \Bigg(\sum_{i=1}^{k}\sum_{j=1}^{n} p_i \cdot p_j \cdot I\Big[\ell(\vec{\sigma}(s_i)) + \ell(\vec{\sigma}(s_j)) \le L\Big]\Bigg), \quad (8)$$

where we adapted the expression from (3) to the case of a single distribution and a single code. But now trying to extend the recursive expression for $F(k,N)$ in (4) gives

$$F(k,N) = \max\Bigg(\max_{\ell_0 \in [1,L]} \Big(F(k-1, N-N_{\ell_0}) + p_k \cdot \sum_{i=1}^{n} p_i \cdot I\Big[\ell_0 + \ell(\vec{\sigma}(s_i)) \le L\Big]\Big), F(k-1,N)\Bigg), \quad (9)$$

which cannot be used by the algorithm because the indicator function now depends on lengths of codewords that were not assigned yet, namely for the elements $i \in [k+1, n]$. So even though we now only have a single code to design, this task is considerably more challenging than the conditional optimization of Section III-B. At this point the only apparent route to solve (8) is by trying exhaustively all length assignments to S satisfying Kraft's inequality, and enumerating the arguments of the max function in (8) directly. But this would be intractable.

B. Efficient algorithm for optimal entry encoding

In the remainder of the section we show an algorithm that offers an efficient way around the above-mentioned difficulty to assign codeword lengths to elements. We present this efficient algorithm formally, but first we describe its main idea. **The main idea**: we showed in Section IV-A that it is *not* possible to maximize the single-code success probability for k elements given the optimal codeword lengths for $k-1$ elements. So it does not work to successively add elements to the solution while maintaining optimality as an invariant. But fortunately, it turns out that it does work to successively add *codeword lengths* to the solution while maintaining optimality as an invariant. The subtle part is that the lengths need to be added in a carefully thought-of sequence, which in particular, is *not* the linear sequence $(1, 2, \ldots, L-1)$ or its reverse-ordered counterpart. We show that if the codeword lengths are added in the order of the sequence

$$(L/2, L/2+1, L/2-1, L/2+2, L/2-2, \ldots, L-1, 1) \quad (10)$$

(for even[3] L), then for any sub-sequence we can maximize the success probability given the optimal codeword lengths taken from the sub-sequence that is one shorter. For example, when $L = 8$ our algorithm will first find an optimal code only using codeword length $L/2 = 4$; based on this optimum it will find an optimal code with lengths 4 and 5, and then continue to add the codeword lengths $3, 6, 2, 7, 1$ in that order.

We now turn to a more formal treatment of the algorithm. We first define the function holding the optimal success probabilities for sub-problems of the problem instance. The following Definition 11 is the adaptation of Definition 10 to the sequence of codeword lengths applicable in the single-code case.

Definition 11. *Consider assignments of finite codeword lengths to the consecutive elements $\{s_{k_1}, \ldots, s_{k_2}\}$ from S, where the lengths are assigned from the values $\{L/2, L/2+1, L/2-1, \ldots, l\}$ taken from the sub-sequence of* (10) *that ends with l. For $N \in [0, 2^L]$ we denote by $G(l, [k_1, k_2], N)$ the maximal success probability for such an assignment whose sum of weights for these $k_2 - k_1 + 1$ codewords is at most N. Formally,*

$$G(l,[k_1,k_2],N) \triangleq \max_{\sigma:\left(\forall i \in [k_1,k_2]: \ell(\vec{\sigma}(s_i)) \in \{L/2,\ldots,l\}\ ,\ \sum_{i=k_1}^{k_2} N_{\ell(\vec{\sigma}(s_i))} \le N\right)} \Bigg(\sum_{i=k_1}^{k_2}\sum_{i'=k_1}^{k_2} p_i \cdot p_{i'} \cdot I\Big[\ell(\vec{\sigma}(s_i)) + \ell(\vec{\sigma}(s_{i'})) \le L\Big]\Bigg). \quad (11)$$

The following two theorems are the key drivers of the efficient dynamic-programming algorithm finding the optimal code. We first explain the intuition behind them. For the above sequence from (10), the theorems describe a step where an additional length among the first lengths in the sequence is allowed. Theorem 8 describes a step allowing using some even number of the first lengths in the sequence, where in Theorem 9 following the step an odd number of lengths can be used. Based on the sequence definition, there is an inherent difference between the two cases. In the first case, the additional allowed length is larger than all previously allowed lengths, and the sum of this length and each of the previous lengths in the sequence is above L. On the other hand, in the second case, the additional allowed length is smaller than all previously allowed lengths and the sum of this length and each of the previous lengths in the sequence is not larger than L. In both cases these properties mean that the success of encoding an element with the new length does not depend on the exact lengths assigned to the other elements among the previous lengths in the sequence. This independence allows the development of the algorithm below. We start with the first theorem.

Theorem 8. *Let $l = L/2 - r + 1$ for some integer $1 \le r \le L/2 - 1$. We denote by l' the length that follows l in the sequence from* (10)*. For this length, satisfying $l' = L/2 + r$, we have*

$$G(l',[k_1,k_2],N) = \max_{j \in [0, k_2-k_1+1]} G(l,[k_1,k_2-j], N - j \cdot N_{l'}), \quad (12)$$

where we define

$$G(l,[k_1,k_1-1],N) \triangleq 0, \text{ for } N \ge 0. \quad (13)$$

[3]For convenience we assume that L is even, but all the results extend to odd L.

Proof: We calculate the values of the function G with parameter l' based on the values of G with parameter l. Since l' follows l in the sequence, the function G with the parameter l' covers solutions potentially using the codeword length l' in addition to the lengths up to l. For a range of elements $[k_1, k_2]$, this additional allowed length l' can be assigned to between 0 and $k_2 - k_1 + 1$ elements. By the monotonicity of p_i, l' which is *larger* than all previous lengths in the sequence, must be assigned to the highest j indices in the range $[k_1, k_2]$. Thus for each j the success probability is the value of G for the corresponding range of elements $[k_1, k_2 - j]$ with the residual weight $N - j \cdot N_{l'}$. In particular, the elements assigned length l' do not add to the success probability, because l' plus any length in the sub-sequence up to l exceeds L. In the extreme case when $j = k_2 - k_1 + 1$ (all elements assigned length l'), the definition (13) when appearing in the right-hand side of (12) gives a valid assignment with success probability 0 if N in the left-hand side is sufficiently large. ∎

As mentioned, Theorem 8 relies on the fact that the calculated success probability is not influenced by the exact assignment of the previously allowed lengths from the sequence in (10) to the other elements. With this property Theorem 8 allows to efficiently extend the optimality from l of type $l = L/2 - r + 1$ (with $r \geq 1$) to the next length in the sequence. To complete what is required for an efficient algorithm, we need the same extension of optimality from l of type $l = L/2 + r$ to the next length in the sequence. We do this in the next theorem.

Theorem 9. *Let $l = L/2 + r$ for some integer $1 \leq r \leq L/2 - 1$. We denote by l' the length that follows l in the sequence from* (10)*. For this length, satisfying $l' = L/2 - r$, we have*

$$G(l', [k_1, k_2], N) = \tag{14}$$
$$\max_{j \in [0, k_2 - k_1 + 1]} \Bigg(G(l, [k_1 + j, k_2], N - j \cdot N_{l'}) + \left(\sum_{i=k_1}^{k_1+j-1} p_i \right) \left(\sum_{i=k_1}^{k_2} p_i \right) + \left(\sum_{i=k_1+j}^{k_2} p_i \right) \left(\sum_{i=k_1}^{k_1+j-1} p_i \right) \Bigg).$$

Proof: Given maximal values G for all values of N and with lengths up to l in the sequence, the maximal value G when l' is also allowed is obtained by assigning length l' to between 0 and $k_2 - k_1 + 1$ elements in the range $[k_1, k_2]$. By the monotonicity of p_i, l' which is *smaller* than all previous lengths must be assigned to the lowest j indices in the range $[k_1, k_2]$. Now the success probability has two components: first is the success between pairs of elements assigned lengths up to l in the sequence, and second is the success between element pairs that involve the new length l' (recall that in Theorem 8 the second component did not exist, but here it does because l' is lower than all lengths up to l.) The first of the three terms in the summation of (14) gives the first component, and the latter two terms give the second component. Maximization is done as before by considering all possible j with the residual weight $N - j \cdot N_{l'}$. ∎

As in Theorem 8, the length sequence provides the property that maximization in Theorem 9 only depends on the success probabilities of the previous length sub-sequence, without care to the assignment of lengths among the lengths in that sub-sequence. In Algorithm 3 we formally present the algorithm for finding an optimal single prefix code, building on Theorems 8 and 9. For terseness we only track the optimal success probabilities G, omitting the more technical task of tracking the optimal assigned lengths, which is required to find the optimal code in a real implementation. The noteworthy parts

Algorithm 3: Optimal Single Prefix Code

input : Element distribution (S, P), memory width L
output: Prefix code σ with optimal entry-encoding success probability

initialization:
foreach $N \in [-n \cdot 2^L, -1]$ **do** $G(l, [k_1, k_2], N) := -1$ for all indices $l \in [1, L-1]$, $k_1 \leq k_2$ (satisfying $k_1, k_2 \in [1, n]$) ;
foreach $N \in [0, 2^L]$ **do** $G(l, [k, k-1], N) := 0$ for all indices $l \in [1, L-1]$ and $k \in [1, n]$;
codewords of length $L/2$:
foreach $[k_1, k_2] \subseteq [1, n]$, $k_1 \leq k_2$ **do**
 for $N = (k_2 - k_1 + 1) \cdot N_{L/2} : 2^L$ **do**
 $G(L/2, [k_1, k_2], N) := \left(\sum_{i=k_1}^{k_2} p_i \right)^2$
 end
end
main iteration:
for $r = 1 : L/2 - 1$ **do**
 go right to length $L/2 + r$:
 for $N = 0 : 2^L$ **do**
 foreach $[k_1, k_2] \subseteq [1, n]$, $k_1 \leq k_2$ **do**
 $G(L/2 + r, [k_1, k_2], N) := \max_{j \in [0, k_2 - k_1 + 1]} G(L/2 - r + 1, [k_1, k_2 - j], N - j \cdot N_{L/2+r})$
 end
 end
 go left to length $L/2 - r$:
 for $N = 0 : 2^L$ **do**
 foreach $[k_1, k_2] \subseteq [1, n]$, $k_1 \leq k_2$ **do**
 $G(L/2 - r, [k_1, k_2], N) := \max_{j \in [0, k_2 - k_1 + 1]} \Bigg(G(L/2 + r, [k_1 + j, k_2], N - j \cdot N_{L/2-r}) + \left(\sum_{i=k_1}^{k_1+j-1} p_i \right) \left(\sum_{i=k_1}^{k_2} p_i \right) + \left(\sum_{i=k_1+j}^{k_2} p_i \right) \left(\sum_{i=k_1}^{k_1+j-1} p_i \right) \Bigg)$
 end
 end
end
output results:
Encoding success probability
$P_{\text{success}} := \max_{k \in [1,n]} G(1, [1, k], 2^L)$

of Algorithm 3 are:

- The initialization of G to -1 for negative N, and to 0 for non-negative N and empty ranges of elements (according to (13)).
- Starting the length sequence at $l = L/2$ and calculating the success probability when all elements in the range are assigned that length.
- The main iteration following the progressions in the length sequence using Theorems 8 and 9.
- Outputing the optimal success probability for the code parameters as a maximization over all n possible numbers of elements assigned codewords, starting from the first element.

Time Complexity: There are $L-1$ iterations and in every iteration $O(2^L)$ values are calculated for each of $O(n^2)$ ranges of elements, each by considering $O(n)$ possibilities for the number of elements with the new codeword length. It follows that the time complexity of the algorithm is $O(n^3 \cdot 2^L \cdot L)$, which is polynomial in the size of the input, or $O(n^5 \cdot L)$ (recall that 2^L is at most quadratic in n). It is interesting to compare this complexity as one order of n higher than that of Algorithm 2 ($O(n^4 \cdot L)$) for the two-code case. We note that the n^5 complexity term is a loose bound, because many of the counted iterations are not exercised in a given run.

Example 4. *Consider the entry distribution* $D = [(S,P),(S,P)]$ *with* $(S,P) = ((s_1,\ldots,s_n),(p_1,\ldots,p_n))$ *satisfying* $(p_1,\ldots,p_n) = (0.4, 0.4, 0.08, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01)$ *with* $n = 15$. *The width parameter is* $L = 6$ *and assume that a single code is allowed. A simple possible coding scheme can assign codewords of length* $L/2 = 3$ *to the* $2^3 = 8$ *most probable elements* $s_1,\ldots,s_8$, *and codewords are not assigned for the remaining elements* $s_9,\ldots,s_{15}$. *This scheme successfully encodes all pairs of elements composed of two elements from* $s_1,\ldots,s_8$, *achieving success probability of* $P'_{\text{success}} = \left(\sum_{i=1}^{8} p_i\right)^2 = 0.93^2 = 0.8649$. *Alternatively, the above optimal algorithm finds a code that achieves a higher success probability. In this code elements* s_1, s_2 *are assigned codewords of length 2 while elements* $s_3,\ldots,s_{10}$ *are assigned codewords of length 4. Later elements are not assigned codewords. This optimal scheme achieves an improved success probability of* $P_{\text{success}} = \left(\sum_{i=1}^{2} p_i\right)^2 + \sum_{i=1}^{2} p_i \cdot \sum_{i=3}^{10} p_i + \sum_{i=3}^{10} p_i \cdot \sum_{i=1}^{2} p_i = 0.8^2 + 0.8 \cdot 0.15 + 0.15 \cdot 0.8 = 0.88$. *We also compare the above results to the performance of the optimal encoding scheme from Section III that does not restrict the two fields to share a code. For this entry distribution* $D = [(S,P),(S,P)]$, *this scheme achieves a success probability of 0.9704. We can observe that the restriction of a shared code can sometimes have a large negative impact on the obtained performance.*

As in the previous section, the coding scheme of this section can be generalized to a non-binary alphabet using the same sequence of codeword lengths. With a codeword weight of $N^A_{\ell_0} \triangleq \alpha^{L-\ell_0}$ and an allowed total weight of $N = \alpha^L$, Algorithm 3 solves this case too. As mentioned at the beginning of the section, Algorithm 3 can be extended to the case where the two fields have *different distributions* with the same n, and a single code with optimal success probability is found. We do not explore this generalization in the paper, but it is as simple as replacing some instances of p_i in the algorithm by q_i, corresponding to the distribution of the second field. If we want to use the same encoder/decoder in both fields (in addition to the same code), then we need the property that both fields have the same elements $(s_1,\ldots,s_n)$ with the same order of their probabilities.

V. Empirical Results

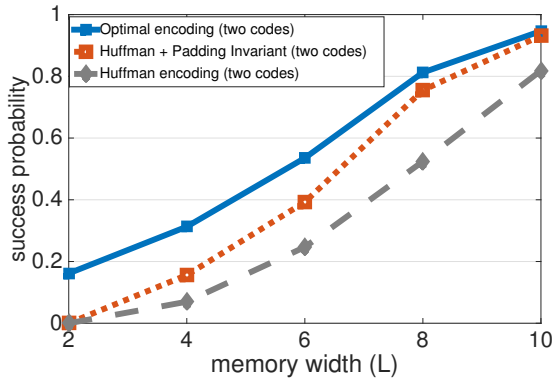
We examine the performance of the suggested coding schemes. In the experiments, the probabilities in the element distributions follow the Zipf distribution with different parameters. A low positive Zipf parameter μ results in a distribution that is close to the uniform distribution, while for a larger parameter the distribution is more biased. An element distribution $(S,P) = ((s_1,\ldots,s_n),(p_1,\ldots,p_n))$ with a Zipf distribution of parameter μ satisfies $p_i = \frac{1}{\eta} \cdot i^{-\mu}$, where $\eta = \sum_{j=1}^{n} j^{-\mu}$.

As a first step, we compare the suggested optimal coding schemes to schemes using the Huffman code [3]. The results are shown in Fig. 2. First, in Fig. 2(a), we pick two different distributions on 128 elements for the two fields, and set the assumption that the codes of the two fields may be different. The distributions of the first and second field are Zipf with parameters $\mu_1 = 0.8$ and $\mu_2 = 2$, respectively. We compute the optimal codes σ_1 and σ_2 using Algorithm 2 from Section III, and plot their success probability. In comparison, we also plot the success probability of a Huffman-based coding scheme composed of a Huffman code for each field according to its distribution. It can be seen that while a Huffman coding scheme minimizes the expected codeword length, in a fixed-width memory the encoding success probability of the Huffman-based coding scheme is inferior to the optimal scheme for all values of L. For $L = 2$ for instance, the Huffman-based scheme does not encode successfully any pair of elements, while the optimal coding scheme achieves a probability of 0.162. A maximal difference in the success probability of 0.289 is obtained for $L = 6$ (0.5354 vs. 0.2468). In addition, we also plot the performance of a combined scheme of the two above, where σ_1 is the Huffman code and σ_2 is the padding invariant code $\hat{\sigma}_2$ (as the code for this field in the coding scheme from Section III). We can see that the optimal scheme is superior to both Huffman-based alternatives for every L.

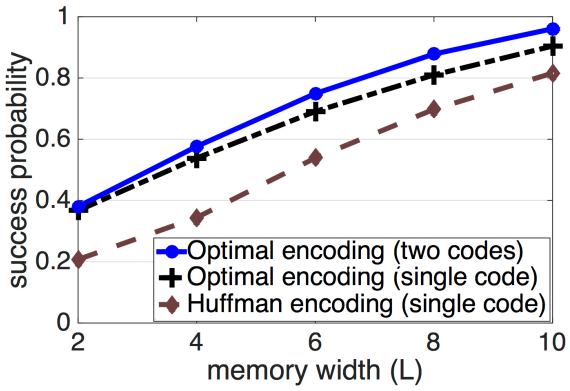
In Fig. 2(b) we examine the scenario of the two fields having the same element distribution, with the requirement that a single code is shared by the two fields. An optimal coding scheme for this case was given in Section IV. The two fields have the same 128 elements with a Zipf $\mu = 1.6$ distribution. We compute the optimal code σ using Algorithm 3, and plot its success probability. In comparison, we also plot the success probability of the Huffman-based coding scheme with two identical Huffman codes computed from the

common element distribution. Here too the Huffman-based coding scheme gives inferior success to the optimal code, with a maximal probability difference of 0.194 at $L = 4$. From a practical perspective the most significant advantage occurs in $L = 10$, where the complement $(1 - P_{\text{success}})$ of single-code optimal encoding is factor 1.92 lower than single-code Huffman encoding, implying factor 1.92 fewer accesses to slower memory due to encoding failures, and average access perfromance roughly twice better. We also plot for comparison the success probability of the optimal coding scheme when allowing two different codes for the two fields. The results underscore the fact that allowing two different codes can improve success probability even when the two fields follow the same distribution.

Next, we examine the impact of the number of possible elements in the fields for different Zipf distributions. We assume a single shared code and that the two fields follow the same distribution. In Fig. 3, we consider two such distributions and present the optimal success probability. In Fig. 3(a), the distribution has a parameter $\mu = 0.5$. We assume that there are n elements in each field. The minimal width required to obtain a success probability of 1 for n elements is given by $L = 2 \cdot \log_2 n$, i.e., $L \geq 6$ for $n = 8$ and $L \geq 8$ for $n = 16$. For a given L, the optimal success probability decreases when the number of elements in each field increases. For $L = 8$ it equals 0.449, 0.208 and 0.099 for $n = 32, 64$ and 128, respectively. Likewise, in Fig. 3(b), the distribution has a parameter $\mu = 2$, which results in a more biased element distribution. While again for $n = 8$, a success probability of 1 is observed only for $L \geq 6 = 2 \cdot \log_2 n$, the more biased distribution allows achieving high success probabilities even for larger values of n. For instance, while for $n = 128$ a memory width of $L = 2 \cdot \log_2 n = 14$ is required to guarantee a success probability of 1, already for $L = 8$ we obtain probability of 0.939 when $\mu = 2$.

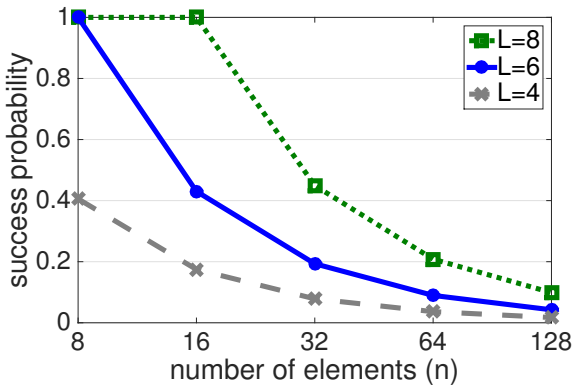


(a) two distributions, $\mu_1 = 0.8, \mu_2 = 2$

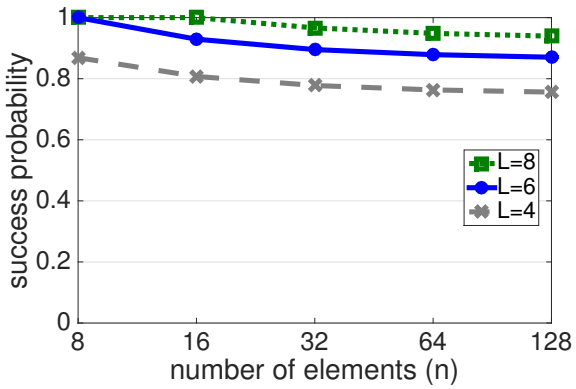


(b) single distribution, $\mu = 1.6$

Fig. 2. Comparison of the optimal coding schemes for two codes and a single shared code vs. Huffman-based schemes. In (a), with two codes for fields with two distributions. The optimal coding scheme (from Section III) is compared with a scheme composed of the two Huffman codes for the two fields, and a third scheme with a Huffman code for the first field and a padding-invariant code for the second field. In (b), with a single shared code for field with the same distribution. The optimal coding scheme (from Section IV) is compared with a scheme composed of the same Huffman code for the two fields. Probabilities follow Zipf distribution with parameters $\mu_1 = 0.8, \mu_2 = 2$ (in (a)) and $\mu = 1.6$ (in (b)).



(a) single code, $\mu = 0.5$



(b) single code, $\mu = 2$

Fig. 3. The success probability of the optimal coding scheme for a shared single code vs. the number of elements in the distribution for different values of the memory width L. Probabilities follow the Zipf distribution with parameters $\mu = 0.5$ (in (a)) and $\mu = 2$ (in (b)).

VI. Conclusion

In this paper we developed compression algorithms for fixed-width memories. We presented a new optimization problem of maximizing the probability to encode entries within the memory width, and described constraints on the codes for the different input fields that guarantee uniquely-decodable data entries. We derived algorithms that find an optimal coding scheme with maximum success probability for the scenario of two codes for two fields, as well as for a single shared code. We also presented experimental results to evaluate the success-probability performance. The central open problem left by this paper is extending optimal coding to $d > 2$ fields.

VII. Acknowledgment

We would like to thank Neri Merhav for fruitful discussions and pointers.

References

[1] O. Rottenstreich, A. Berman, Y. Cassuto, and I. Keslassy, "Compression for fixed-width memories," in *IEEE International Symposium on Information Theory (ISIT)*, Istanbul, Turkey, July 7–12 ,2013.

[2] A. Hari, U. Niesen and G. T. Wilfong, "On the problem of optimal path encoding for Software-Defined networks," *IEEE/ACM Trans. on Networking*, vol. 25, no. 1, pp. 189–198, 2017.

[3] D. Huffman, "A method for the construction of minimum redundancy codes," *Proc. IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[4] A. Kirsch and M. Mitzenmacher, "The power of one move: Hashing schemes for hardware," *IEEE/ACM Trans. on Networking*, vol. 18, no. 6, pp. 1752–1765, 2010.

[5] T. Hu and K. Tan, "Path length of binary search trees," *SIAM Journal on Applied Mathematics*, vol. 22, no. 2, pp. 225–234, 1972.

[6] M. Garey, "Optimal binary search trees with restricted maximal depth," *SIAM Journal on Computing*, vol. 3, no. 2, pp. 101–110, 1974.

[7] R. G. Gallager, "Variations on a theme by Huffman," *IEEE Trans. on Information Theory,* vol. 24, no. 6, pp. 668–674, 1978.

[8] L. L. Larmore and D. S. Hirschberg, "A fast algorithm for optimal length-limited Huffman codes," *Journal of the ACM (JACM)*, vol. 37, no. 3, pp. 464–473, 1990.

[9] J. Abrahams, "Code and parse trees for lossless source encoding," in *IEEE Compression and Complexity of Sequences*, Positano, Amalfitan Coast, Salerno, Italy, June 11–13, 1997.

[10] M. B. Baer, "Source coding for quasiarithmetic penalties," *IEEE Trans. on Information Theory,* vol. 52, no. 10, pp. 4380–4393, 2006.

[11] B. P. Tunstall, *Synthesis of Noiseless Compression Codes*. Ph.D. dissertation, Georgia Inst. Tech., Atlanta, GA, 1967.

[12] F. Jelinek, "Buffer overflow in variable length coding of fixed rate sources," *IEEE Trans. on Information Theory*, vol. 14, no. 3, pp. 490–501, 1968.

[13] P. A. Humblet, "Generalization of Huffman coding to minimize the probability of buffer overflow," *IEEE Trans. on Information Theory*, vol. 27, no. 2, pp. 230–232, 1981.

[14] O. Rottenstreich, M. Radan, Y. Cassuto, I. Keslassy, C. Arad, T. Mizrahi, Y. Revah, and A. Hassidim, "Compressing forwarding tables for datacenter scalability," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 1, 2014.

[15] N. Abramson, *Information Theory and Coding.* McGraw-Hill, 1963.

[16] T. M. Cover and J. A. Thomas, *Elements of Information Theory* (2. ed.). Wiley, 2006.

[17] R. Draves, C. King, V. Srinivasan, and B. Zill, "Constructing optimal IP routing tables," in *IEEE Conference on Computer Communications (INFOCOM)*, New York, NY, USA, March 21–25, 1999.

[18] O. Rottenstreich, I. Keslassy, A. Hassidim, H. Kaplan and E. Porat, "Optimal In/Out TCAM Encodings of Ranges," *IEEE/ACM Trans. on Networking*, vol. 24, no. 1, pp. 555–568, 2016.

[19] G. Rétvári, J. Tapolcai, A. Korösi, A. Majdán and Z. Heszberger, "Compressing IP forwarding tables: Towards entropy bounds and beyond," *IEEE/ACM Trans. on Networking*, vol. 24, no. 1, pp. 149–162, 2016.

[20] O. Rottenstreich and J. Tapolcai, "Optimal Rule Caching and Lossy Compression for Longest Prefix Matching," *IEEE/ACM Trans. on Networking*, vol. 25, no. 2, pp. 864–878, 2017.

Ori Rottenstreich Ori Rottenstreich (S'10-M'15) is a researcher in the field of computer networks. In 2015-2017 he was a Postdoctoral Research Fellow at the Department of Computer Science, Princeton university. Earlier, he received the B.Sc in Computer Engineering (summa cum laude), and Ph.D degree from the Electrical Engineering Department of the Technion, Haifa, Israel in 2008 and 2014, respectively. He was a recipient of the Rothschild Yad-Hanadiv postdoctoral fellowship and the Google Europe PhD Fellowship in Computer Networking. He also received the Best Paper Runner Up Award at the 2013 IEEE Infocom conference as well as the Best Paper Award at the 2017 ACM Symposium on SDN Research (SOSR).

Yuval Cassuto Yuval Cassuto (S'02-M'08-SM'14) is a faculty member at the Viterbi Department of Electrical Engineering, Technion - Israel Institute of Technology. His research interests lie at the intersection of the theoretical information sciences and the engineering of practical computing and storage systems. During 2010-2011 he has been a Scientist at EPFL, the Swiss Federal Institute of Technology in Lausanne. From 2008 to 2010 he was a Research Staff Member at Hitachi Global Storage Technologies, San Jose Research Center. He received the B.Sc degree in Electrical Engineering, summa cum laude, from the Technion in 2001, and the M.S. and Ph.D. degrees in Electrical Engineering from the California Institute of Technology, in 2004 and 2008, respectively. From 2000 to 2002, he was with Qualcomm, Israel R&D Center, where he worked on modeling, design and analysis in wireless communications. Dr. Cassuto has won the 2010 Best Student Journal Paper Award in data storage from the IEEE Communications Society, as well as the 2001 Texas Instruments DSP and Analog Challenge $100,000 prize.