Proceedings of the Estonian Academy of Sciences,
2013, **62**, 1, 47–58
doi: 10.3176/proc.2013.1.06
Available online at www.eap.ee/proceedings

COMPUTER SCIENCE

Does the Shannon bound really apply to all data structures?

Antti Valmari

Department of Mathematics, Tampere University of Technology, PO Box 553, FI-33101 Tampere, Finland; Antti.Valmari@tut.fi

Received 18 August 2011, revised 25 November 2012, accepted 17 December 2012, available online 20 February 2013

Abstract. Shannon's information-theoretic lower bound has been developed for uniquely decodable systems of bit strings, while ordinary data structures often consist of many separate blocks of memory. One might expect that adapting the bound to data structures is trivial, but we demonstrate that this is not the case. Kraft's inequality is at the heart of information-theoretic lower bound proofs. We present a tiny distributed data structure where Kraft's inequality fails, or it is at least very difficult to give any other satisfactory explanation. Then we formalize the concept of data structure with the notion of "representation scheme" that is general enough to model the example system. We re-establish the information-theoretic lower bound by proving that Kraft's inequality applies to a subset of representation schemes that contains at least one memory-optimal scheme for each set of objects. Unlike in classical information theory, a representation scheme may be memory-optimal even if some object has more than one representation. However, this only happens if some representation has probability zero.

Key words: information theory, data structures, memory consumption.

1. INTRODUCTION

It is well known that to store one of N possibilities, $\lceil \lg N \rceil$ bits are needed, where lg is the base-2 logarithm. We call this the *information-theoretic lower bound*.

When applying similar reasoning to more complicated data objects, such as sets of numbers, some complications arise. Figure 1 shows two representations of the set $\{18, 20, 43\}$ as a chained hash table (e.g., [1, Chapter 11]). We see that the same set has many representations. A representation is not necessarily a contiguous bit string, but may consist of separate blocks of memory that could be just anywhere in the address space. The representations of different sets may be of different sizes.

In this case, it is no longer meaningful to claim that at least $\lceil \lg N \rceil$ bits are needed by necessity. In the case of sets of 32-bit integers, $N = 2^{2^{32}}$ and $\lceil \lg N \rceil = 2^{32} \approx 4 \times 10^9$. A linked list that represents the empty set consists of just the null pointer, only using 32 bits on a 32-bit computer. So individual objects can have

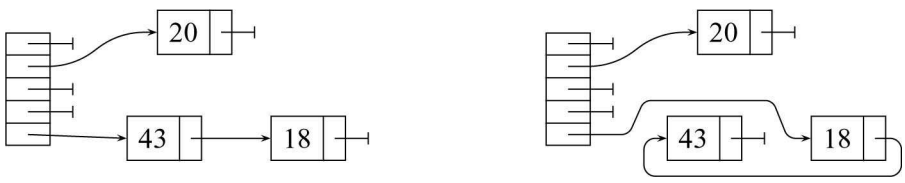

Fig. 1. Two different representations of the set $\{18, 20, 43\}$ as a hash table.

representations that are much smaller than $\lceil \lg N \rceil$. However, it is meaningful to talk of how many bits are needed on the average.

Sets of 32-bit integers are routinely represented in computers, and more than 4×10^9 bits are seldom used. So $\lceil \lg N \rceil$ is an irrelevant bound in practice. The solution to this apparent paradox is that typically in practical applications, not all sets are equally likely. Instead, small sets are much more likely than others.

According to Shannon's information theory, if each object i has probability p_i, then at least $-\sum_{i \in I} p_i \lg p_i$ bits are needed on the average, where I is the set of all distinct objects that we want to be able to store [7]. It reduces to $\lg N$ if the task is to choose one of N possibilities, each equally likely. Because $\lg 0$ is not defined, we point out that the correct result is obtained by letting $p_i \lg p_i = 0$ when $p_i = 0$.

In practice, one typically wants to be able to store any set up to some maximum size n, either because n is obtained from the intended use of the set or one wants to store as big sets as possible within the available memory. Let w be the number of bits needed to represent an individual element of the set. Then a relevant bound is obtained by letting each set with at most n elements have the same probability, while bigger sets have probability 0. Shannon's bound is then between $nw - n \lg n$ and $nw - n \lg n + n \lg e + 1$ (where $e \approx 2.71828\ldots$), depending on n and w (e.g., [8]). (The right-hand side depends on the assumption that $n \leq (2^w + 1)/3$, which clearly holds in the case of small sets.) First giving n as a 32-bit integer and then listing the elements in a sequence consumes $32 + nw$ bits. So nw seems natural to a programmer. However, the formula predicts that smaller memory consumption is possible. Indeed, [8] presents a practical example whose memory consumption is below nw (but not below $nw - n \lg n$).

In conclusion, Shannon's bound seems useful for analysing the memory consumption of data structures. However, a problem arises. The bound follows from the so-called Kraft's inequality. Kraft's inequality – and thus Shannon's bound – has been proven for self-delimiting bit strings (no codeword is a proper prefix of another) [4,7] and more generally for uniquely decodable systems of bit strings (no finite bit string can be decomposed to a sequence of codewords in more than one way) [5]. So, it automatically applies to any data structure that always uses a continuous block of memory independently of what has been stored to it. However, as was illustrated by Fig. 1, practical data structures are often not like that.

One could expect there to be a simple proof that Kraft's inequality applies to all data structures. Indeed, we will present a straightforward proof in Section 5 that covers most data structures. Unfortunately, the proof is not fully satisfactory, because it relies on the assumption that if a value or object has more than one representation, then they must be distinguishable from each other. This seems an unnecessary requirement, because they represent the same object. What is more, in Section 3 we will present and analyse a small distributed data structure that, in the opinion of the author, violates both this assumption and Kraft's inequality. During the history of this publication, alternative explanations have been suggested, but also they involve something unusual, like consider the program counter as part of the data structure or count the same bit as belonging to two different data structures simultaneously.

A subroutine that reads a data structure must never be confused about the stored value. However, this does not imply that it is never confused about what bits belong to the data structure. This is a subtle fact. Our example system is based on it.

Section 4 presents a new abstract simple definition of data structures that is general enough to model both familiar data structures and our weird example system. In this new theory Kraft's inequality does not always hold, but we will show in Section 6 that such data structures cannot consume less memory than the best of those for which Kraft's inequality does hold. Therefore, Shannon's bound is re-established. This proof of Shannon's bound applies as long as the user's idea of data structures satisfies the assumptions made in the new theory. These assumptions are very weak. Therefore, the author believes that all reasonable notions of data structure satisfy them. Also the example system in Section 3 satisfies them.

The existence of multiple representations for the same object rules out memory-optimality in classical information theory, but not in the new theory. Fortunately, to re-establish the result it suffices to add the assumption that no representation has probability zero. This difference and the violation of Kraft's inequality do not change the practical consequences of the results of information theory. Furthermore, the cases where Kraft's inequality fails are certainly rare in practice. So, for practical purposes, this publication would be

largely unnecessary. The significance of this publication is in presenting an example that challenges the standard proof, and giving an alternative proof.

In Section 2, we recall the results from classical information theory that are needed by the rest of this publication. The tiny weird example data structure is presented and discussed in Section 3. Section 4 motivates and presents our new formalization of data structures that is general enough to cover practical complicated data structures and the example in Section 3. We will call the formalization "representation schemes". The main goal of Section 5 is to prove that Kraft's inequality does apply to a significant special case of representation schemes. It is shown in Section 6 that representation schemes that are outside this special case cannot be better than the best of those inside. Assuming that no representation has zero probability of being used, they cannot even be as good. Section 7 concludes the publication.

2. RECALLING CLASSICAL INFORMATION THEORY

In this section, we recall some basic results from information theory in a form that is suitable for the rest of this publication. We present them with proofs, because the proofs are short, help reading the novel results in subsequent sections, and, unlike classical information theory, we also consider multiple representations of the same object.

To start with, there is some finite set I which we will call the set of *objects*, and we want to be able to represent any element of I in computer memory. A simple example of such a set is the integers in the range -2^{31} to $2^{31}-1$. In this example, an object is an integer value in the given range. A more interesting example is the set of all possible phone books up to some maximum size, where a phone book is a set of pairs consisting of a person's name and phone number. Please notice that it is the contents of the phone book as a whole, not an individual name–number pair, that is an object in the present sense of the word.

In classical information theory, objects are represented by bit strings that are self-delimiting or, more generally, whose system is uniquely decodable. A bit string that represents an object is a *codeword* of that object. Each object has at least one codeword, and no codeword represents more than one object. In this section, we denote the set of codewords by R or something related, like R_0. In subsequent sections, R may also consist of other kinds of representations. We will use r and s to denote the elements of R, and $|r|$ is the number of bits used by r. When r is a bit string, then $|r|$ is the length of r.

If each representation r has probability p_r of occurring, then the average memory consumption of the representation system is $\sum_{r\in R} p_r|r|$. We will soon see that in classical information theory, if an object has more than one representation, then the representation system is not memory-optimal. This holds even if the redundant representation has probability zero. In this publication, the issue is less trivial: we will see that a representation system with multiple representations can be memory-optimal. It only occurs if the redundant representation has probability zero, so the result has no practical significance. However, it forces us to develop some parts of the theory anew.

Fortunately, it is obvious that if one representation of minimal size is chosen for each object and its other representations are thrown away, then memory consumption does not increase. Therefore, when studying lower bounds for memory consumption, the possibility of multiple representations for the same object may be ignored. By doing so we perhaps lose some, but not all, memory-optimal representation systems for the set of objects in question.

When no object has more than one representation, the average memory consumption of the representation system is $\sum_{i\in I} p_i|r_i| = \sum_{r\in R} p_r|r|$, where p_i is the probability that the object is i and r_i is the unique representative of i. So we may concentrate on $\sum_{r\in R} p_r|r|$ from now on.

A simple induction argument yields the following result, known as *Kraft's inequality* (e.g., [6, Chapter 1]).

Lemma 1. *If R is a finite set of finite bit strings such that none of its elements is a proper prefix of another one, then $\sum_{r\in R} 2^{-|r|} \le 1$.*

Proof. We use induction on an upper bound b to $|r|$. When $b = 0$, then $R = \emptyset$ or $R = \{\varepsilon\}$ (where ε is the empty bit string), so the claim holds. Consider an R such that $|r| \leq b+1$ for every $r \in R$. If $\varepsilon \in R$, then $R = \{\varepsilon\}$, because no r is a proper prefix of another. Otherwise the elements of R can be partitioned to two sets R_0 and R_1, those elements that start with 0 and those elements that start with 1. Removing the first bit from each element of R_0 yields a set S_0 of bit strings such that none of them is a proper prefix of another. By the induction assumption, $\sum_{r\in S_0} 2^{-|r|} \leq 1$. This implies that $\sum_{r\in R_0} 2^{-|r|} = \sum_{r\in S_0} 2^{-(|r|+1)} \leq \frac{1}{2}$. The set R_1 is handled similarly. Altogether $\sum_{r\in R} 2^{-|r|} = \sum_{r\in R_0} 2^{-|r|} + \sum_{r\in R_1} 2^{-|r|} \leq \frac{1}{2} + \frac{1}{2} = 1$, completing the proof. □

Kraft's inequality is valid also in the more general framework of uniquely decodable systems of bit strings, but the proof is more difficult (although still short) [2,3,5].

A classical result of great interest is that, no matter what the representation system is, if Kraft's inequality holds, then the average memory consumption is at least $-\sum_{i\in I} p_i \lg p_i$. The following lemma formulates this result. In it, I is the set of objects, R is the set of all possible representations (now they need not be bit strings and not necessarily all of them are in use), and $R(i)$ is the set of the representations of object i.

Lemma 2. *Let I and R be finite sets. For each $i \in I$, let $R(i)$ be a set such that $R(i) \subseteq R$ and if $i \neq j$, then $R(i) \cap R(j) = \emptyset$. For each $r \in R$, let $|r|$ be a natural number and p_r be the probability of r. For each $i \in I$, let $p_i = \sum_{r\in R(i)} p_r$. If $\sum_{r\in R} 2^{-|r|} \leq 1$, then $\sum_{r\in R} p_r|r| \geq -\sum_{i\in I} p_i \lg p_i$.*

Proof. Consider any non-negative number C. Also consider any non-negative numbers $q_1, \ldots, q_n$ such that $\sum_{h=1}^{n} q_h = 1$. We first find the minimum of $-\sum_{h=1}^{n} q_h \lg x_h$ under the condition $\sum_{h=1}^{n} x_h = C$ and $x_h \geq 0$ for $1 \leq h \leq n$. If $q_h \neq 0 \neq q_k$, let $x = x_h$ and $c = x_h + x_k$. The minimum of $f(x) = -q_h \lg x - q_k \lg(c-x)$ in the range $0 \leq x \leq c$ is obtained when $q_h/x_h = q_k/x_k$, because (only) then $\frac{d}{dx} f(x) = 0$. If $q_h = 0 \neq q_k$, then the minimum is when $x_h = 0$. These imply that there is a constant C' such that the original minimum is obtained when $x_h = C' q_h$ for $1 \leq h \leq n$. Because $\sum_{h=1}^{n} q_h = 1$ and $\sum_{h=1}^{n} x_h = C$, we get $C' = C$. The value of the minimum is thus $-\sum_{h=1}^{n} q_h \lg C q_h = -\lg C - \sum_{h=1}^{n} q_h \lg q_h$.

Letting $n = |R|$, $q_h = p_r$, and $x_h = 2^{-|r|}$, we get $C = \sum_{r\in R} 2^{-|r|}$ and $\sum_{r\in R} p_r|r| \geq -\lg C - \sum_{r\in R} p_r \lg p_r$. Kraft's inequality says that $C \leq 1$, implying that $\sum_{r\in R} p_r|r| \geq -\sum_{r\in R} p_r \lg p_r$. Then $p_i = \sum_{r\in R(i)} p_r$ and $\sum_{r\in R(i)} p_r \lg p_r \leq \sum_{r\in R(i)} p_r \lg p_i = p_i \lg p_i$. Thus $-\sum_{r\in R} p_r \lg p_r \geq -\sum_{i\in I} \sum_{r\in R(i)} p_r \lg p_r \geq -\sum_{i\in I} p_i \lg p_i$, giving the claim. □

The next result adds to the importance of Kraft's inequality. It says that in classical information theory, and more generally in any theory that allows the use of self-delimiting bit strings as representations, Kraft's inequality is as tight as it could be.

Lemma 3. *If $c_1, c_2, \ldots, c_n$ are natural numbers such that $\sum_{i=1}^{n} 2^{-c_i} \leq 1$, then there are bit strings $r_1, r_2, \ldots, rh_n$ such that none is a prefix of another, and $|r_i| = c_i$ for $1 \leq i \leq n$.*

Proof. We may assume without loss of generality that $c_1, c_2, \ldots, c_n$ are given in non-decreasing order. Let $1 \leq i \leq n$. Then $C_i = \sum_{j=1}^{i-1} 2^{c_i - c_j}$ is a natural number, because $j < i$ implies $c_j \leq c_i$. It is less than 2^{c_i}, because $\sum_{j=1}^{i-1} 2^{-c_j} < \sum_{j=1}^{n} 2^{-c_j} \leq 1$. So c_i bits suffice to represent C_i as a binary number. Let r_i be the representation of C_i as a binary number with c_i bits, starting with the most significant bit. If $i < k \leq n$, then $C_k = C_i 2^{c_k - c_i} + \sum_{j=i}^{k-1} 2^{c_k - c_j} \geq C_i 2^{c_k - c_i} + 2^{c_k - c_i}$. So C_k and $C_i 2^{c_k - c_i}$ disagree on the value of at least one of the first c_i bits, implying that neither of r_k and r_i is a prefix of the other. □

If all probabilities are non-zero, and if the representations can be replaced such that one object gets a shorter representation and no object gets a longer representation than it had, then memory consumption is reduced. The next lemma together with Lemma 3 says that if the upper bound in Kraft's inequality is not reached, then such a change is possible.

Lemma 4. *If $c_1, c_2, \ldots, c_n$ are natural numbers such that $n \geq 1$ and $\sum_{i=1}^{n} 2^{-c_i} < 1$, then there are natural numbers $d_1, d_2, \ldots, d_n$ such that $\sum_{i=1}^{n} 2^{-d_i} \leq 1$, $d_i = c_i - 1$ for one i, and $d_i = c_i$ for the other values of i.*

Proof. At least one of the c_i is at least as big as the others. Let that one be c_j. We have $c_j > 0$, because otherwise $\sum_{i=1}^{n} 2^{-c_i} < 1$ would not hold. Let $d_j = c_j - 1$, and $d_i = c_i$ for the other values of i. Because $2^{c_j} \sum_{i=1}^{n} 2^{-c_i}$ is an integer and less than 2^{c_j}, we have $2^{c_j} \sum_{i=1}^{n} 2^{-c_i} + 1 \leq 2^{c_j}$ and $\sum_{i=1}^{n} 2^{-c_i} + 2^{-c_j} \leq 1$. Thus $\sum_{i=1}^{n} 2^{-d_i} = \sum_{i=1}^{n} 2^{-c_i} - 2^{-c_j} + 2^{-(c_j-1)} = \sum_{i=1}^{n} 2^{-c_i} + 2^{-c_j} \leq 1$. □

We may now conclude that in any theory where Kraft's inequality holds and self-delimiting bit strings can be used as representations, also zero probability implies non-optimality. Namely, the removal of the codewords that have zero probability does not change average memory consumption. However, it strictly reduces $\sum_{r \in R} 2^{-|r|}$, so Lemmas 3 and 4 guarantee the existence of a better representation system. Also the existence of many representations for the same object implies non-optimality. Their joint probability can be concentrated on the shortest representation, after which the other representations have zero probability, and the previous reasoning applies.

Corollary 5. *Assume that Kraft's inequality applies and self-delimiting bit strings can be used as representations. If some object has more than one representation or if some object has probability zero, then the representation system is not memory-optimal.*

3. AN EXAMPLE OF A WEIRD TINY DATA STRUCTURE

In this section, we discuss a situation that is intuitively a representation for one bit of information, but where Kraft's inequality does not apply, or it is at least difficult to give any other satisfactory explanation.

Figure 2 shows a distributed system consisting of a master process and two server processes. The service requests to the system by the outside world arrive via the master, but the servers reply directly to the original requesters. Each server can also ask the other server to perform a sub-task. In this case, the former server sends a message directly to the latter server, which sends the result directly back to the former server. There can be only one level of sub-tasking; that is, a server performing a sub-task cannot ask for help from the other server. If a server is computing an original task while the other server sends it a sub-task, it puts the original task aside until it has completed the sub-task. We assume that message passing is synchronous, like in the Ada and Occam programming languages. That is, messages arrive immediately to their destinations instead of waiting in queues or channels. As a consequence, communication links do not have any memory.

For the purpose of this discussion, each server can be in one of five different states: idle, serving an original task (it has not asked for help or has already got a reply from the other server), waiting for a reply from the other server, and serving a sub-task for the other server without or with an original task put aside. These states are not necessarily represented explicitly via specific bits. Instead, they may be represented implicitly by a reply address (should the server send its reply to the other server or the outside world), the value of the program counter (is the execution at the statement that inputs the reply from the other server), and so on.

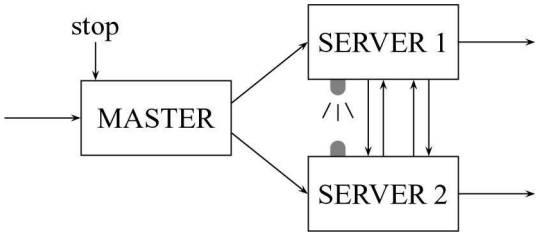

Fig. 2. A distributed server system.

The master may be requested to run the system down in a controlled way. Upon such a request, the master stops passing new service requests to the servers. Then, before finally switching off the system, it must wait until the processing of earlier service requests is completed. For that purpose, each server has a bit, called the *activity bit*, that the master sees and whose value denotes the status of the server, $0 =$ "inactive" and $1 =$ "active". The activity bits are shown as lamps in Fig. 2, with Server 1 active and Server 2 inactive. Each server switches its activity bit to 1 when it inputs a request and to 0 when sending the reply. So the bit is 0 precisely when the server is idle. From the point of view of the master, $(0,0)$ means "can switch off" and $(0,1)$, $(1,0)$, and $(1,1)$ mean "must still wait". The master thus gets one bit of information from these two bits. We call this bit *wait info*. Its value is either "can switch off" or "must still wait".

In principle, the activity bit would not need memory of its own, because its value is determined by the value of the program counter, etc., of the server. However, it is much easier to use an explicit bit of memory than to implement a memoryless connection from the program counter of the server to the master.

To improve the use of memory, we now make a modification to the system. From now on, each server uses its activity bit to denote activity only when it is idle, serving an original task, or waiting for a reply. If it is serving a request from the other server, it uses its activity bit as working storage. That is, while a server is processing a sub-task, its activity bit is not used for denoting activity, but for some other purpose. This other purpose sovereignly determines the value of the activity bit. It may thus be 0 or 1, and it may change during the processing of the sub-task. Even so, the master gets the correct "must still wait" information, because the other server is waiting for a reply. Thus its activity bit is 1, and the disjunction of the activity bits is 1 independently of the value of the other activity bit.

At the level of the server, nothing odd has happened. The server gets its own status from the program counter, etc. The activity bit is an extra bit of memory. When the server is idle, serving an original task, or waiting for a reply from the other server, the value of its activity bit reflects this fact; and when the server is serving a sub-task, its activity bit is used as working storage.

Things start getting odd when we try to determine how many bits are used for representing the wait info. While an activity bit is used as working storage, the master may read it, getting 0 or 1. However, independently of whether it is 0 or 1, the master gets the value "must still wait". If we take the fact that the master may read the bit as sufficient indication of the bit belonging to the wait info, then we must conclude that the bit is simultaneously used both as working storage and by the wait info. Counting the same bit as being simultaneously used for two different purposes seems dubious. Furthermore, there is no pressing need to do so, because the value of the bit does not affect the value of the wait info. So it seems best to conclude that when an activity bit is being used as working storage, it is not being used for representing the wait info. Then we denote the value of the bit by "$-$" when discussing the wait info, although even then the master may read it, getting either 0 or 1.

We see that *it is not strictly necessary that a process that reads a data structure "knows" precisely which bits belong to it. Reading of extraneous bits that are used for other purposes is acceptable, if it does not affect the value that is obtained.*

A server may be processing a sub-task only when the other server is waiting for a reply. So the possible combinations are $(0,0)$, $(0,1)$, $(1,0)$, $(1,1)$, $(1,-)$, and $(-,1)$. The value "can switch off" has one representation. Its size is two. The value "must still wait" has five representations, two of which are of size one and the remaining are of size two. Therefore, $\sum_{r\in R} 2^{-|r|} = 4\times 2^{-2} + 2\times 2^{-1} = 2$, violating Kraft's inequality. One may argue that the representations $(0,1)$, $(1,0)$, and $(1,1)$ must be counted out, because the same bit combinations are covered also by $(1,-)$ and $(-,1)$. However, then $\sum_{r\in R} 2^{-|r|} = 2^{-2} + 2\times 2^{-1} = 1\frac{1}{4}$, still violating Kraft's inequality. It is reasonable to claim that $(1,-)$ covers $(1,0)$ but unreasonable to claim that $(-,1)$ covers $(1,0)$, so $(1,-)$ and $(-,1)$ are different from each other in at least some sense. If we admit that the wait info uses only one bit when the opposite activity bit is used as working storage, and if we admit that $(1,-)$ and $(-,1)$ are different representations, then the sum is by necessity greater than 1. The essential difference of this system to usual situations where Kraft's inequality holds is that the reader does not "know" precisely which bits belong to the data structure.

That a particular bit sometimes belongs to one data structure and at another time belongs to another data structure is very common in practice. This happens, for instance, when subroutine P has a local data structure D_P and subroutine Q has a different local data structure D_Q, and the main program calls first P and then Q. When P is started, memory for D_P is reserved from the run-time stack. When P terminates, D_P is popped from the run-time stack. When Q is started, the same memory is occupied by D_Q. In this respect, the only unusual feature of our example is that the master does not "know" that the activity bit denoted by "$-$" is not used by the wait info, and may thus read it. However, as we have already pointed out, this does not confuse the value that the master gets.

Another line of thought that deserves discussion is clumsy to illustrate unless the number of states of each server is a power of two. Therefore, we temporarily drop the state where the server is serving a sub-task with an original task put aside. Then the state of each server may be represented by two bits. That makes four bits, yielding six when also the activity bits are taken into account. These six bits represent altogether eight different value combinations (four where the masters are idle or serving an original task, and the activity bits faithfully represent this; and four where one master is waiting for a reply, its activity bit is 1, the other master is serving a sub-task, and its activity bit is used as working storage and is 0 or 1). The corresponding sum is $8 \times 2^{-6} = \frac{1}{8} \leq 1$. Indeed, as was pointed out earlier, originally the activity bits would not have needed memory of their own. From this point of view it is not surprising that each activity bit can every now and then be used as working storage.

However, a fully satisfactory theory of data structures should make it possible to analyse each data structure in isolation from other data structures and from the program counter and related run-time information. Often the latter determine the lifetime of a data structure. For instance, in the above example, D_P exists while the program counter points to a location inside P or the run-time stack contains a return address leading to inside P. Analogously, the local state of the server determines when the activity bit is being used as working storage. On the other hand, during the lifetime of a data structure, the program counter is usually not considered. The lifetime of the wait info is the full time the system is running. So it should be possible to analyse the wait info without appealing to the local state of the server.

One may wonder what is the role of the information content of the program code that reads the data structure. Usually there may be arbitrarily many instances of a data structure, each with its own contents, but they all are read by the same code (or identical copies of some code). Therefore, if there are n copies of the data structure, an instance of the data structure contains d bits of information, and the reading code contains c bits of information, then there are altogether $nd + c$ bits. This makes $d + c/n$ bits per instance, and approaches d as n grows. So the information content of the program code does not play any role. We are not discussing Kolmogorov complexity.

4. REPRESENTATION SCHEMES

In Section 2, objects were represented by bit strings such that none of them is a proper prefix of another. To meet the needs of complicated data structures, we now develop a more general theory of representations of objects. We try to make as few and as weak assumptions as possible, to obtain as widely applicable a theory as possible. This will make our definition rather abstract. Our theorems apply to all notions of data structures that satisfy the assumptions made below.

We have used integers in the range from -2^{31} to $2^{31} - 1$, sets of 32-bit integers, and phone books as examples. Such integers are almost always represented by 32 bits that are next to each other in memory. The representations of phone books are far more varied. For instance, to facilitate efficient searching, addition, and removal of name–number pairs, a chained hash table may be used. It consists of an array of pointers. Each pointer starts a linked list, whose each record contains a name, a number, and a pointer to the next record. The records are allocated from dynamic memory and may reside in almost any memory address. If the hash table is implemented in C in the usual way, then the entry for the name in the record consists of a pointer to somewhere else in the memory, where the name is stored as a 0-terminated byte sequence of unspecified length. Figure 1 shows a simpler example of hash tables.

A *representation scheme* is a system according to which objects are represented in memory, possibly in a scattered way. It is our formalization of the notion of "data structure", and its formal definition will be given below. Let us first discuss its intuition.

A representation of an object occupies some bits in memory and gives values 0 or 1 to them. The occupied bits are not necessarily next to each other in memory, and what bits are occupied may depend on the contents of other bits – that is what pointers are for. Therefore, we will define the representation via two sets B_0 and B_1. They are understood as the locations of bits in memory that are set to 0 and those that are set to 1, specified relative to the starting point (that is, the address) of the representation in memory. The elements of $B_0 \cup B_1$ are thus relative memory addresses. In practice, they are usually integers. However, their precise nature is irrelevant for our theory, so we do not make any assumption about it. Because no bit can be simultaneously 0 and 1, we require that $B_0 \cap B_1 = \emptyset$.

It is, of course, required that each object has at least one representation. More than one is allowed, because, as already mentioned, many common data structures associate many representations to the same object. For instance, swapping the ordering of records in a linked list like in Fig. 1 does not change the set that is stored (the object), although it changes the bit pattern in memory (the representation).

Another requirement is that there must never be confusion of which object is represented. To ensure that, it is not necessary to insist that any two representations can be distinguished from each other. It suffices to require that if they represent different objects, then it must be possible to distinguish between them. Indeed, in the example in Section 3, both $(1,1)$, $(1,-)$, and $(-,1)$ represent "must still wait", and they cannot be distinguished from each other when the unused bits are 1, but they all can be distinguished from $(0,0)$ that represents "can switch off". Distinguishability of different objects is guaranteed by requiring the existence of a bit, called "conflicting bit", such that one of the representations assigns 0 and the other assigns 1 to it.

We are ready to present the formal definition of representation schemes and some related concepts.

Definition 6. *A* representation *of an object is a pair (B_0,B_1) of finite sets such that $B_0 \cap B_1 = \emptyset$. The* size *of the representation is $|(B_0,B_1)| = |B_0| + |B_1|$.*

A representation scheme *R for a set of objects is a set of representations such that each object has at least one representation, and if (B_0,B_1) and (B'_0,B'_1) are representations of two different objects, then $B_0 \cap B'_1 \neq \emptyset$ or $B_1 \cap B'_0 \neq \emptyset$. Elements of these two intersections are called* conflicting bits.

A representation scheme is unambiguous, *if and only if no object has more than one representation. Otherwise it is* ambiguous.

A representation scheme is fully conflicting, *if and only if each pair of distinct representations has a conflicting bit.*

An unambiguous representation scheme is obviously fully conflicting.

Representations that consist of bit strings such that none is a proper prefix of another are a special case of fully conflicting representation schemes. There B_0 and B_1 consist of non-negative integers such that $B_0 \cup B_1 = \{0,1,\ldots,b-1\}$, where $b = |B_0 \cup B_1|$. Practical data structures are often ambiguous. For instance, all data structures that rely on links or pointers are ambiguous, because the whole idea of the pointer is that the data may reside in several different memory locations and the pointer tells where it is. On the other hand, data structures are usually fully conflicting. There is, however, no fundamental reason *forcing* them to be fully conflicting. The "wait info" data structure in Section 3 is neither fully conflicting nor unambiguous.

Like in Section 2 the average memory consumption of a representation scheme R is $\sum_{r\in R} p_r|r|$, where p_r is the probability of r occurring. If $R(i)$ denotes the set of representations of object i, then the probability of i is $p_i = \sum_{r\in R(i)} p_r$. When we compare the performance of two representation schemes with probabilities, we assume that they represent the same objects and yield the same probability for each object. We say that a representation scheme with probabilities is *memory-optimal*, if and only if there is no representation scheme with probabilities for the same set of objects and the same probability for each object, with smaller average memory consumption. Because repeatedly saying "with probabilities" is clumsy, we usually skip it from here on.

We will discuss fully conflicting representation schemes in Section 5 and the remaining representation schemes in Section 6.

5. FULLY CONFLICTING REPRESENTATION SCHEMES

We now prove that Kraft's inequality applies to fully conflicting representation schemes.

Theorem 7. *If R is a fully conflicting finite representation scheme, then $\sum_{r\in R} 2^{-|r|} \leq 1$.*

Proof. We use induction on $|R|$. The claim clearly holds when $|R| \leq 1$. In the opposite case, there are at least two different representations r_1 and r_2, and they have at least one conflicting bit b. The set R can be partitioned into three subsets R_0, R_1, and $R_\perp$, consisting of those representations that set b to 0, set b to 1, or do not use b. That is, $R_0 = \{(B_0,B_1) \in R \mid b \in B_0\}$, $R_1 = \{(B_0,B_1) \in R \mid b \in B_1\}$, and $R_\perp = \{(B_0,B_1) \in R \mid b \notin B_0 \cup B_1\}$. Thanks to r_1 and r_2, we have $R_0 \neq \emptyset$ and $R_1 \neq \emptyset$.

Let R'_0 be the set of representations that do not use b but are otherwise the same as the elements of R_0. That is, $R'_0 = \{(B_0 \setminus \{b\}, B_1) \mid (B_0,B_1) \in R_0\}$. Thus $\sum_{r\in R_0} 2^{-|r|} = \sum_{r\in R'_0} 2^{-(|r|+1)} = \frac{1}{2}\sum_{r\in R'_0} 2^{-|r|}$. Any two distinct elements of $R_0 \cup R_\perp$ have a conflicting bit, because R is fully conflicting. The bit b cannot be a conflicting bit for them, because elements of $R_\perp$ do not use it and elements of R_0 agree that its value is 0. As a consequence, $R'_0 \cap R_\perp = \emptyset$ and any two distinct elements of $R'_0 \cup R_\perp$ have a conflicting bit. Thus $R'_0 \cup R_\perp$ is a fully conflicting representation scheme. We have $|R'_0 \cup R_\perp| = |R_0 \cup R_\perp| = |R \setminus R_1| < |R|$. By the induction assumption, $\sum_{r\in R'_0\cup R_\perp} 2^{-|r|} \leq 1$. The reasoning can be repeated with 0 and 1 swapped, yielding $\sum_{r\in R_1} 2^{-|r|} = \frac{1}{2}\sum_{r\in R'_1} 2^{-|r|}$ and $\sum_{r\in R'_1\cup R_\perp} 2^{-|r|} \leq 1$. Therefore,

$$\begin{aligned}
\sum_{r\in R} 2^{-|r|} &= \sum_{r\in R_0} 2^{-|r|} + \sum_{r\in R_1} 2^{-|r|} + 2\times\frac{1}{2}\sum_{r\in R_\perp} 2^{-|r|} \\
&= \frac{1}{2}\sum_{r\in R'_0} 2^{-|r|} + \frac{1}{2}\sum_{r\in R_\perp} 2^{-|r|} + \frac{1}{2}\sum_{r\in R'_1} 2^{-|r|} + \frac{1}{2}\sum_{r\in R_\perp} 2^{-|r|} \\
&= \frac{1}{2}\sum_{r\in R'_0\cup R_\perp} 2^{-|r|} + \frac{1}{2}\sum_{r\in R'_1\cup R_\perp} 2^{-|r|} \leq \frac{1}{2}+\frac{1}{2} = 1,
\end{aligned}$$

concluding the proof. □

Now that Kraft's inequality is available, and because we pointed out that self-delimiting bit strings are a special case of fully conflicting representation schemes, Lemma 2 and Corollary 5 yield the following result.

Theorem 8. *Let I be a finite set whose elements are called objects, and let p_i be the probabilities of the objects. The average memory consumption of any fully conflicting finite representation scheme that represents the objects in I is at least $-\sum_{i\in I} p_i \lg p_i$. If some object has more than one representation or if some object has probability zero, then the representation scheme is not memory-optimal.*

A fully conflicting representation scheme may be memory-optimal even if no single representation uses all the bits that the representations use altogether. A simple example uses three bits and gives them values as follows: $(0,0,-)$, $(0,1,-)$, $(1,-,0)$, and $(1,-,1)$. Like in Section 3, also here "$-$" denotes that the bit in the corresponding position is not used by the representation.

6. NON-FULLY CONFLICTING REPRESENTATION SCHEMES

We would like to extend the results in Theorem 8 to representation schemes that are not necessarily fully conflicting. The lower bound result generalizes easily. To prove it we define the following.

Definition 9. *Let R be a representation scheme. An* unambiguity *of R is obtained by removing all but one representations of each object, and letting the probability of the remaining representation be the sum of the probabilities of the original representations of the same object. An unambiguity is* minimal, *if and only if, for each object, none of its original representations is smaller than the representation that is kept.*

Each unambiguity of R is trivially unambiguous. An ambiguous representation scheme obviously can never use less memory than its minimal unambiguity. So, for every representation scheme, there is an unambiguous representation scheme for the same objects that uses at most the same memory on the average. As a consequence, the $-\sum_{i\in I} p_i \lg p_i$ bound of Theorem 8 applies to all representation schemes, although the rest of the theorem does not necessarily apply.

Corollary 10. *Let I be a finite set whose elements are called objects, and let p_i be the probabilities of the objects. The average memory consumption of any finite representation scheme that represents the objects in I is at least $-\sum_{i\in I} p_i \lg p_i$.*

Unfortunately, the non-optimality result stated in Corollary 5 or the latter part of Theorem 8 does not generalize in full. Consider the representation scheme that uses altogether two bits, and assigns the representations $(0,-)$ and $(0,0)$ to the first object, and $(1,-)$ to the second object. If the probability of $(0,0)$ is 0, then this scheme is memory-optimal although it is ambiguous.

This counterexample has a representation whose probability is zero. Such a representation is not relevant in practice, because it will never be used. We could as well remove it from the scheme. Therefore, a weaker form of the non-optimality result that assumes the absence of zero probabilities would be interesting. The rest of this section is devoted to developing one.

In the absence of zero probabilities, it is clear that if some object has two representations of different sizes, then the scheme is not memory-optimal. The case remains where an object may have many representations, but they are all of the same size. Unfortunately, Kraft's inequality does not necessarily apply even in this case. An example is the scheme in Section 3 with $(0,1)$, $(1,0)$, and $(1,1)$ left out. There one object is represented by $(1,-)$ or $(-,1)$ and another object is represented by $(0,0)$. In this case, $\sum_{r\in R} 2^{-|r|} = 2^{-1} + 2^{-1} + 2^{-2} > 1$.

So we cannot directly appeal to Kraft's inequality. Therefore, we introduce a transformation that will let us use Kraft's inequality indirectly. Let (D_0, D_1) and (C_0, C_1) be representations (not necessarily from the same scheme). We say that (D_0, D_1) *contains* (C_0, C_1) if and only if $C_0 \subseteq D_0$ and $C_1 \subseteq D_1$.

Let R be an ambiguous representation scheme. The *expansion of an object* consists of replacing its representations in a certain way. Each of the new representations occupies every bit that any of the original representations occupies. The expansion consists of all assignments of values to these bits that contain at least one original representation. For instance, if the original representations of an object are $(1,-,-)$, $(0,0,-)$, and $(-,0,1)$, then the expansion of the object is $\{(0,0,0), (0,0,1), (1,0,0), (1,0,1), (1,1,0), (1,1,1)\}$. The triple $(0,0,1)$ is in the expansion, because it contains $(0,0,-)$ (and also $(-,0,1)$). The triple $(0,1,0)$ is not in, because its first bit conflicts with $(1,-,-)$ and its second bit with $(0,0,-)$, and $(-,0,1)$.

The expansion of an object is denoted with $\mathscr{E}_R(r)$, where r is any original representation of the object. It depends on R, but does not depend on which representation r of the object in question is chosen. The *expansion of R* is denoted with $\mathscr{E}(R)$ and consists of the expansions of the objects of R. That is, $\mathscr{E}(R)$ is the union of $\mathscr{E}_R(r)$ for $r \in R$. We now present its definition formally.

Definition 11. *Let R be a representation scheme and $r \in R$. The set of all elements of R that represent the same object as r is denoted with $[\![r]\!]$. The* expansion *of R is $\mathscr{E}(R) = \bigcup_{r\in R} \mathscr{E}_R(r)$, where*

$$\begin{aligned}\mathscr{E}_R((B_0,B_1)) \;=\; \{\, (D_0,D_1) \mid\;& D_0 \cup D_1 = \textstyle\bigcup_{(C_0,C_1)\in[\![(B_0,B_1)]\!]} (C_0 \cup C_1)\\ \wedge\;& D_0 \cap D_1 = \emptyset\\ \wedge\;& \exists (C_0,C_1) \in [\![(B_0,B_1)]\!] : C_0 \subseteq D_0 \wedge C_1 \subseteq D_1 \,\}\,.\end{aligned}$$

Given any new representations of two different objects, they inherit a conflicting bit from the original representations that they contain. Any two different new representations of the same object use precisely the same bits and thus have a conflicting bit. So $\mathscr{E}(R)$ is a fully conflicting representation scheme, and satisfies Kraft's inequality by Theorem 7.

We are now ready to complete the proof that ambiguous representation schemes that lack zero probabilities are not memory-optimal.

Theorem 12. *If R is an ambiguous finite representation scheme such that $p_r > 0$ for each $r \in R$, then R is not memory-optimal.*

Proof. The average memory consumption is $\sum_{r\in R} p_r|r|$. If some object has two representations r_1 and r_2 such that $|r_2| > |r_1|$, then the removal of r_2 from R in favour of r_1 modifies the average memory consumption by $p_{r_2}(|r_1| - |r_2|) < 0$, so R is not memory-optimal. The rest of the proof discusses the case where R does not have representations of different size for the same object.

Let $\mathscr{U}(R)$ be an unambiguity of R. The common size of all representations of an object in R is also the size of its representation in $\mathscr{U}(R)$. Therefore, $\mathscr{U}(R)$ has the same average memory consumption as R. If r_1 and r_2 represent the same object, then $\mathscr{E}_R(r_1) = \mathscr{E}_R(r_2)$. Therefore, $\mathscr{E}(R) = \bigcup_{r\in\mathscr{U}(R)} \mathscr{E}_R(r)$.

Let $r \in R$, and let b be the (common) size of the elements of $\mathscr{E}_R(r)$. Then $\mathscr{E}_R(r)$ has at least the $2^{b-|r|}$ representations that contain r, yielding $\sum_{s\in\mathscr{E}_R(r)} 2^{-|s|} = |\mathscr{E}_R(r)|2^{-b} \geq 2^{b-|r|}2^{-b} = 2^{-|r|}$. Because R is ambiguous, there is at least one object that has two different representations r and r'. Since r' is of the same size as r but not the same, it assigns a different value to some bit from what r assigns or fails to occupy a bit that r occupies. By letting that bit have the opposite value from what r assigns to it, and by letting the bits occupied by r' have the values given by r', a representation in $\mathscr{E}_R(r)$ is found that contains r' but not r. For the object in question, $|\mathscr{E}_R(r)| \geq 2^{b-|r|} + 1$ and $\sum_{s\in\mathscr{E}_R(r)} 2^{-|s|} \geq (2^{b-|r|} + 1)2^{-b} > 2^{-|r|}$.

Altogether $\sum_{r\in\mathscr{E}(R)} 2^{-|r|} = \sum_{r\in\mathscr{U}(R)} \sum_{s\in\mathscr{E}_R(r)} 2^{-|s|} > \sum_{r\in\mathscr{U}(R)} 2^{-|r|}$, because $r \in \mathscr{U}(R)$ scans the expansion of each object precisely once. By Theorem 7, $1 \geq \sum_{r\in\mathscr{E}(R)} 2^{-|r|}$. So $\sum_{r\in\mathscr{U}(R)} 2^{-|r|} < 1$. Thanks to the object mentioned above, $\mathscr{U}(R) \neq \emptyset$. By Lemmas 3 and 4, another representation scheme exists which gives each object a representation of at most the same size as $\mathscr{U}(R)$ gives, and at least one object a smaller representation than $\mathscr{U}(R)$ gives. Because all probabilities are non-zero, this scheme consumes less memory than $\mathscr{U}(R)$. Thus $\mathscr{U}(R)$, and consequently R, is not memory-optimal. □

7. DISCUSSION

Classical information theory assumes that objects are represented as bit strings, but this does not usually hold for typical data structures. Therefore, we developed a more general notion of representation schemes, and analysed information-theoretic lower bounds using it.

We proved that Shannon's famous lower bound applies to all representation schemes. That is, the average memory consumption of any data structure is at least $-\sum_{i\in I} p_i \lg p_i$ bits, where p_i is the probability that the stored object is i. In classical information theory, the bound follows from Kraft's inequality. However, there are representation schemes to which Kraft's inequality does not apply. We proved the result by proving Kraft's inequality for a class of representation schemes, and observing that representation schemes that do not belong to this class cannot be better.

In classical information theory, if any object has more than one representation or if any object has zero probability, the representation system is not memory-optimal. This result also follows from Kraft's inequality, and thus holds for the class of representation schemes mentioned above. Perhaps surprisingly, outside that class there are representation schemes for which the result does not hold. However, this only happens if the probability of some representation is zero. Such situations do not have practical significance. Even so, they act as a warning that one should not take it for granted that classical information-theoretic results apply to data structures. Instead, proofs are needed.

The class of representation schemes to which Kraft's inequality applies covers all usual data structures. On the other hand, we presented an unusual but realistic distributed system example that is outside that class. Therefore, it was necessary to extend our theory from that class to all representation schemes. For the lower bound it was trivial, but it was not trivial for the result that multiple representations of non-zero probability imply non-optimality.

In conclusion, the classical lower bound and non-optimality results (the latter with the extra assumption of non-zero probabilities) also apply to data structures, even exotic ones. However, they had to be proven anew. The exotic feature in our counter-example is that, although the process that reads the current value of a data structure always gets the intended value, it does not necessarily always "know" what bits are used for representing the value. Therefore, it may accidentally read some bits that are used for some other purpose – but only in such a way that it is not fooled to get a wrong value.

We did not discuss representation systems for countably infinite sets of objects. Such systems exist. For instance, any n-bit natural number can be represented as a bit string of length $2n+1$ where 10 denotes bit 0, 11 denotes bit 1, and 0 indicates the end of the string. If the probabilities of long representations are small enough, then the average memory consumption may be finite. Therefore, extending the theory to countably infinite sets of objects could be interesting.

ACKNOWLEDGEMENTS

There was an exceptionally long discussion between the anonymous reviewers and myself, mediated by the editor. The publication has definitely benefited a lot from the discussion and the amount of time and energy that the reviewers and the editor spent.

REFERENCES

1. Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to Algorithms, 2nd edition*. The MIT Press, 2001.
2. Foldes, S. On McMillan's theorem about uniquely decipherable codes. arXiv:0806.3277v2, 2008.
3. Karush, J. A simple proof of an inequality of McMillan. *IRE Trans. Inform. Theory*, 1961, **7**(2), 118.
4. Kraft, L. G. *A Device for Quantizing, Grouping, and Coding Amplitude-Modulated Pulses*. Master's thesis, Massachusetts Institute of Technology, Cambridge, USA, 1949.
5. McMillan, B. Two inequalities implied by unique decipherability. *IRE Trans. Inform. Theory*, 1956, **2**(4), 115–116.
6. Reingold, E. M. Algorithm design and analysis techniques. In *Algorithms and Theory of Computation Handbook* (Atallah, M. J., ed.). CRC Press, 1999.
7. Shannon, C. E. A mathematical theory of communication. *Bell System Techn. J.*, 1948, **27**(3), 379–423 and **27**(4), 623–656.
8. Valmari, A. What the small Rubik's cube taught me about data structures, information theory and randomisation. *STTT*, 2006, **8**(3), 180–194.

Kas Shannoni alamtõke on ikka kõigile andmestruktuuridele kohaldatav?

Antti Valmari

Shannoni infoteoreetiline alamtõke kehtib üheselt dekodeeritavate bitistringide süsteemide kohta, samal ajal kui reaalsed andmestruktuurid koosnevad paljudest eraldi mäluplokkidest. Võiks arvata, et alamtõkke rakendamine andmestruktuuridele on triviaalne, kuid artiklis on näidatud, et see pole nii. Krafti võrratus on infoteoreetilise alampiiri tõestuse keskseks komponendiks. On tutvustatud väikest hajusat andmestruktuuri, mille korral Krafti võrratus ei kehti või siis on vähemalt väga raske sellele rahuldavat selgitust anda. Seejärel on formuleeritud andmestruktuuri mõiste "esitusskeemi" kaudu, mis on näitesüsteemi modelleerimiseks piisavalt üldine. On taasesitatud infoteoreetiline alampiir, tõestades, et Krafti võrratus kehtib esitusskeemi korral, mis sisaldab vähemalt üht mäluoptimaalset skeemi iga objektide hulga jaoks. Sellest erinevalt võib klassikalises infoteoorias olla esitusskeem mäluoptimaalne isegi juhul, kui mõnel objektil on rohkem kui üks esitus. Kuid see juhtub ainult siis, kui mingi esituse esinemise tõenäosus on null.