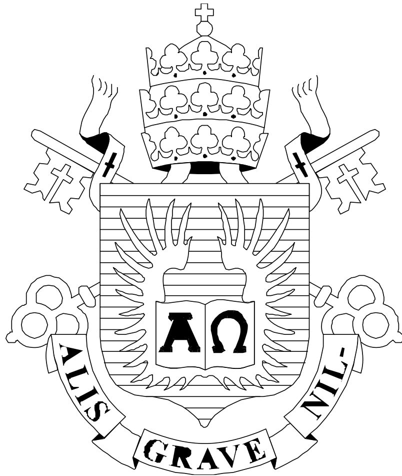

PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 05/09

Merge Source Coding Revisited

Bruno Tenório Ávila
Eduardo Sany Laber

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22451-900

RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 05/09 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena February, 2009

Merge Source Coding Revisited

Bruno Tenório Ávila and Eduardo Sany Laber

bavila@inf.puc-rio.br, laber@inf.puc-rio.br

Abstract. We show that any comparison-based merging algorithm can be naturally mapped into a source coder via a conversion function introduced here. By applying this function over some well known merging algorithms, namely Binary Merging and Recursive Merging, we realize that they are closely related to a runlength-based coder with Rice coding and to the Binary Interpolative Coder, respectively. Furthermore, by applying the conversion function over the Probabilistic Merging algorithm we obtain a new runlength-based coder that uses a variant of the Rice code, namely *Randomized Rice Code*. This new code uses a random source of bits with the aim of reducing its average redundancy with high probability.

Keywords: Source Coding, Merging Algorithms

Resumo. Nós mostramos que qualquer algoritmo de intercalação baseado em comparações pode ser naturalmente mapeado em um codificador de fonte por meio de uma função de conversão introduzida aqui. Aplicando esta função em alguns algoritmos de intercalação conhecidos, especialmente Intercalação Binária e Intercalação Recursiva, nós percebemos que eles são relacionados à um codificador baseado em comprimento de carreiras com codificação Rice e ao Codificador de Interpolação Binária, respectivamente. Além disso, aplicando a função de conversão no Algoritmo de Interpolação Probabilístico nós obtemos um novo codificador baseado em comprimento de carreiras que usa um variante da codificação Rice, chamado *Codificação Rice Aleatorizada*. Esse novo código usa uma fonte aleatória de bits com o objetivo de reduzir a redundância média com alta probabilidade.

Palavras-chave: Codificação de Fonte, Algoritmos de Intercalação

In charge of publications:
Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22451-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3527-1516 Fax: +55 21 3527-1530
E-mail: bib-di@inf.puc-rio.br
Web site: http://bib-di.inf.puc-rio.br/techreports/

1 Introduction

One of the fundamental goals of data compression is to efficiently represent data into bits by removing its redundancy. For a discrete memoryless source that outputs symbols from an alphabet with d symbols according to a probability distribution $X = (x_1, \ldots, x_d)$, the Shannon's source coding theorem [18] shows that the average number of bits per symbol required to represent the data generated by the source is lower bounded by its entropy, given by:

$$H(X) = -\Sigma_{i=1}^{d} x_i \log_2 x_i.$$

A typical data compression algorithm can be split into two phases [17]: the modelling phase and the coding phase. The former is in charge of generating a sequence of probabilities while the latter, implemented by a *coder*, is responsible to represent these probabilities into bits. A coder designed for maximizing compression rate by reaching entropy lower bound is classified as *entropy coder*. The most popular entropy coders are Huffman [10] and Arithmetic [17] coders.

In modern applications, however, reaching the entropy limit is not necessarily the main goal. Some factors, such as, coding speed, hardware constraints, memory limitation and cache optimization may be more important. Coders that prioritise some of these factors rather than achieving the entropy lower bound are classified as *non-entropy coders* [14]. In general they use the same representation regardless of the probability distribution. Examples of non-entropy coders are Elias gamma coder [8], Golomb [9] and Rice coders [16].

Coders can also be classified as *static* or *adaptive*. Another way to classify coders is according to the kind of source they can model: binary or multialphabet; memoryless or Markov; stationary or nonstationary. In summary, there are several types of coders with different characteristics and the best of them depends on the requirements of the target application. In fact, as pointed out by Moffat *et al.* [14], *there is no single "best" coding method. There are, however, a relatively small number of competing alternatives.* Therefore, there is a strong motivation for the development of new kinds of coders for specific applications that fit better than the available ones.

Let us turn our attention, for a while, to the problem of merging two sorted lists, a basic and well studied problem in theoretical computer science [12]. Given two sorted lists A and B, the problem consists of generating a sorted list C that contains the elements in $A \cup B$. This problem arises naturally in numerous application domains, such as database design and management, information retrieval, among others.

Merging k sorted sequences with different sizes is a more general problem that has also received a lot of attention. It can be either solved by a k-way merging algorithm, which simultaneously merges k lists, or by a sequence of $k-1$ executions of a 2-way merge algorithm. In the latter, two of the k lists are merged via a 2-way merging algorithm and then the $k-1$ remaining lists are recursively merged. An interesting observation concerning this strategy appears in [2]: if the 2-way merge is the tape merge algorithm [19] then the best strategy to select the two lists to be merged at each step is obtained by following the execution of Huffman's coding algorithm [10] over a set of symbols $\{l_1, \ldots, l_k\}$, with frequencies $\{f_1, \ldots, f_k\}$, where l_i corresponds to the i-th list and f_i corresponds to its size.

Despite this already known relation between coding and merging, we are not aware of any result indicating how merge algorithms can be useful to design source coders.

This paper shows that any comparison-based merging algorithm can be naturally mapped into a source coder via a conversion function that we introduce here. This result is interesting because it makes possible the interpretation of the vast literature on merging algorithms (e.g. online and offline algorithms, distributed and parallel strategies, hardware implementations, algorithm analysis, etc.) in the context of coding methods.

In fact, by investigating this conversion process we realize that some of the most popular merging algorithms are closely related to some well known coders. As an example, the Binary Merging algorithm proposed in [11] corresponds to a runlength-based encoder that uses Rice coding [16] to represent repetition of bits. Another example is the Recursive Merging algorithm proposed in [7]. It is closely related to the Binary Interpolative Coder proposed by Moffat et. al. [15] for information retrieval applications.

In addition, we propose a new coder that is obtained by applying our conversion function over the Probabilistic Merging algorithm presented in [5]. We find out that it was possible to extract, from the converted coder, a variant of the Rice code namely *Randomized Rice Code*. This new code uses a random source with the aim to reduce its average redundancy with high probability.

We believe that further investigation of the connection established here may lead to new useful source coders.

The rest of the paper is organized as follows. In Section 2, we explain how to map merging algorithms into binary coders. In Section 3, we discuss the connection of two well known merging algorithms with two well known coding methods. Furthermore, we introduce a new coder that is obtained by applying our conversion function over a probabilistic merging algorithm proposed in [5].

2 Merge-based Source Coders

2.1 Comparison-based Merge Algorithms

Given two disjoint linearly ordered subsets $A = \{a_1 < ... < a_m\}$ and $B = \{b_1 < ... < b_n\}$, with $m \leq n$, of a linearly ordered set C, the merging problem consists of determining the linear ordering of their union (i.e. to merge A and B). Let $compare(u, v)$ be an operation that takes as input two elements u and v from a linearly ordered set and outputs either '$<$' if $u < v$ or '$>$' if $u > v$. A comparison-based merge algorithm merges A and B by performing a sequence of *compare* operations, where one of the arguments of *compare* belongs to A and the other one belongs to B.

For a comparison-based merge algorithm r, define $M^r(m, n)$ as the number of comparisons performed by r to merge two ordered lists of sizes m and n in the worst case. In addition, define $M(m, n) = min_{r \in \mathcal{R}} M^r(m, n)$, where r is optimized over the class $\mathcal{R}$ of all possible comparison-based merge algorithms. For all $m, n \geq 1$, the following inequalities hold:

$$I(m, n) \leq M(m, n) \leq m + n - 1,$$

where $I(m, n) = \left\lceil log_2 \binom{m+n}{m} \right\rceil$ is the information theoretical lower bound.

The determination of the exact value of $M(m, n)$ has been studied in several papers [12][19][4][13]. Moreover, there are several algorithms [11][7][5][4][13] that perform $O(I(m, n)) = O(M(m, n)) = O\left(m \log_2 \left(\frac{n}{m} + 1\right)\right)$ comparisons in the worst case. All of them are asymptotically optimal in the comparison-based model.

2.2 Mapping Merging Algorithms into Binary Source Coders

Given a binary string x, let $x(i)$ be the i-th symbol of x. In addition, let $A^x = \{i|x(i) = 0\}$ and $B^x = \{i|x(i) = 1\}$, where a_i^x and b_i^x are the i-th element of A^x and B^x, respectively. As an example, if $x = (11011110010001110111)$, then $A^x = \{3, 8, 9, 11, 12, 13, 17\}$ and $B^x = \{1, 2, 4, 5, 6, 7, 10, 14, 15, 16, 18, 19, 20\}$. This example will be further used in section 3.

Let $\mathcal{R}$ be the class of all comparison-based merging algorithms and let $\mathcal{S}$ be the class of all binary coders. We introduce a function $\varphi : \mathcal{R} \mapsto \mathcal{S}$ that converts an arbitrary merging algorithm $r \in \mathcal{R}$ into a binary coder $s \in \mathcal{S}$. Fix a merging algorithm $r \in \mathcal{R}$. The binary coder $\varphi(r)$ maps a binary string x into a binary string y, where $y(i) = 1$ (resp. $y(i) = 0$) if and only if the i-th operation *compare*, performed by r to merge A^x and B^x, outputs '$>$' (resp '$<$').

Figure 1 illustrates the merging tree corresponding to the tape merging algorithm [19] to merge the sets $A = \{a_1 < a_2\}$ and $B = \{b_1 < b_2\}$. Ellipses are used to represent comparisons and rectangles to represent the final sorted set. In order to encode the string $x = (1001)$, as an example, we first construct the sets $A^x = \{2, 3\}$ and $B^x = \{1, 4\}$. Then, we apply the merge procedure over A^x and B^x. The sequence $>, <, <$ is output by the *compare* operations so that the code $y = (100)$ is generated.

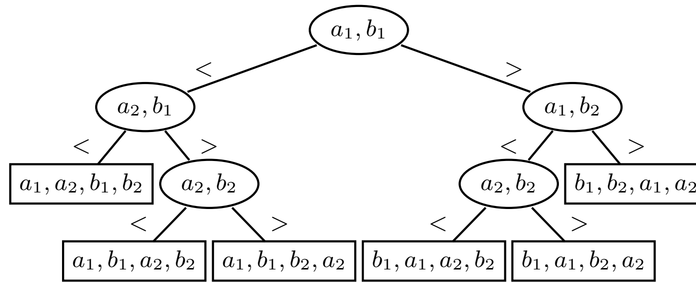

Figure 1: Example of merging tree.

We should note that for a merging algorithm both *compare*(u, v) and *compare*(v, u) have the same effect. However, for encoding purposes we should fix from which list comes the left and the right argument of *compare*. As an example, if we replace *compare*(a_1, b_2) by *compare*(b_2, a_1) in Figure 1 then the encoding of $x = (1001)$ becomes (110) instead of (100).

Given the encoded string y and the merge algorithm r we can recover the original string x. For that, let us assume that the numbers m and n of bits with values 0 and 1 in x are known. We construct two linearly ordered sets $A = \{a_1 < \ldots < a_m\}$ and $B = \{b_1 < \ldots < b_n\}$. Then, we apply the algorithm r over A and B. Although we do not know the values of the elements in A and in B, we can merge these sets because the result of the i-th comparison performed by r is stored in $y(i)$. The output of the merging algorithm r is a linearly ordered set $C = \{c_1 < \ldots < c_{m+n}\}$ containing the elements of $A \cup B$. We reconstruct x by setting $x(i) = 0$ if $c_i \in A$ and $x(i) = 1$, otherwise.

An interesting subset of $\mathcal{R}$ is the subset $\mathcal{R}'$ of asymptotically optimal merging algorithms. The main characteristic of $\mathcal{R}'$ is that the number of comparisons performed by

any merging algorithm $r \in \mathcal{R}'$ is $O(I(m,n))$ in the worst case. Since

$$I(m,n) \in O\left((m+n)H\left(\frac{m}{m+n}, \frac{n}{m+n}\right)\right),$$

it follows that φ converts any merging algorithm $r \in \mathcal{R}'$ into a binary entropy coder.

3 Applications

In this section, we show how some of the most popular merging algorithms relate with some well known entropy coders. In the description of the merging algorithms we omit the assignment steps and some implementation details, because only the comparisons steps are important for the conversion process.

3.1 Binary Merging Algorithm

The Binary Merging algorithm (BM) [11] is a well known algorithm that was originally designed for merging two files stored in tapes. At each step, it verifies whether the first element of the smallest list is larger than the 2^t-th element of the largest list, where t is an integer that depends on the size of both lists. In the positive case, the first 2^t elements of the largest precede the first element of the smallest list. In the negative case, it determines which elements of the largest list are smaller than the first element of the smallest list. Its pseudocode is presented below:

$BM(A, B)$

1. $U \leftarrow$ smallest set between A and B;
2. $V \leftarrow$ largest set between A and B;
3. If both $U \neq \emptyset$ and $V \neq \emptyset$ then
   (a) $t = \lfloor \log_2 \frac{|V|}{|U|} \rfloor$;
   (b) *compare*(u_1, v_{2^t}):
      i. If $u_1 < v_{2^t}$ then
         A. do a binary search to find the integer q for which $v_q < u_1 < v_{q+1}$;
         B. $BM(U[2, \ldots, |U|], V[q+1, \ldots, |V|])$;
      ii. If $u_1 > v_{2^t}$ then
         A. $BM(U[1, \ldots, |U|], V[2^t + 1, \ldots, |V|])$;

Here, we convert a slight variation of the above algorithm into a binary coder. In this variation, at each step, an hybrid search (a sequential search combined with a binary search) is performed to find the number of elements in the largest list that are smaller than the first element of the smallest list or, in other words, the position of u_1 in V. This new algorithm is obtained from BM by replacing line b) with:

b')

1. Do a sequential search to find the smallest integer i in the set $\{1, 2, 3, \ldots\}$ for which $u_1 < v_{i2^t}$;
2. Do a binary search in the interval $[i2^t - 2^t, i2^t - 1]$ to find an integer q for which $v_q < u_1 < v_{q+1}$;
3. $BM(U[2, \ldots, |U|], V[q + 1, \ldots, |V|])$;

Conversion. By converting this modified algorithm we obtain a runlength-based coder that uses Rice coding to represent repetitions of the same bit. The pseudocode for encoding a binary string x into a binary string y is given below:

Encode(x)

1. $m \leftarrow$ number of bits 0 in x;
2. $n \leftarrow$ number of bits 1 in x;
3. If both $m, n > 0$ then:
   (a) $t = \left\lfloor \log_2 \frac{\max\{m,n\}}{\min\{m,n\}} \right\rfloor$;
   (b) if $m > n$ then $j^* \leftarrow \min\{j | x(j) = 1\}$
   else $j^* \leftarrow \min\{j | x(j) = 0\}$;
   (c) append the Rice code for $j^* - 1$, with parameter t, to the output y;
   (d) *Encode*$(x[j^* + 1, \ldots, |x|])$;

In order to illustrate the encoding scheme, we use the example of section 2.2. The encoding method calculates, at each recursion, a value for j^* which generates the sequence of values $(3, 5, 1, 2, 1, 1, 4)$. It encodes these values using Rice code with the respective parameter $(0, 0, 0, 0, 1, 1, 2)$ producing the sequence of Rice codes $(001, 00001, 1, 01, 10, 10, 111)$.

We argue that the above encoder is obtained by applying our conversion function over the modified BM. Let us assume that u_1 is always the left argument of the operation *compare* at step (b'). The key observation is that the sequence of comparisons performed by the merge algorithm to find the integer q in the line (b') corresponds, in the sense of our conversion function, to the Rice code of q, with parameter t. In fact, the unary code corresponds to the sequential search while the binary code corresponds to the binary search.

Thus, it remains to show that the integer q is exactly $j^* - 1$. First note that $m = |A^x|$, $n = |B^x|$ and

$$t = \left\lfloor \log_2 \frac{\max\{m, n\}}{\min\{m, n\}} \right\rfloor = \left\lfloor \log_2 \frac{\max\{|A^x|, |B^x|\}}{\min\{|A^x|, |B^x|\}} \right\rfloor.$$

If $|A^x| > |B^x|$, then q is such that $a^x_q < b^x_1 < a^x_{q+1}$. It follows from the definition of A^x and B^x that $j^* = b^x_1 = q + 1$, and as a consequence, $q = j^* - 1$. A similar argument holds when $|A^x| < |B^x|$.

Discussion. Note that when $m = n$, the Binary Merging algorithm reduces to the tape merging and, when $m = 1$, it reduces to a decentralised binary search. It presents a good balance for the ranges of m and n. The authors proved that the number of comparisons

M^{BM} performed by Binary Merging is upper bounded by $\lceil log_2\binom{m+n}{m}\rceil + m$. We observe that $M^{BM}(m,n) < \lceil log_2\binom{m+n}{m}\rceil + m < (m+n)H(m/(m+n), n/(m+n)) + m$. In addition, since $m \leq n$, it follows that $M^{BM}(m,n) < (m+n)\left[H(m/(m+n), n/(m+n)) + 0.5\right]$.

The proposed modification to the original binary merge algorithm resulted in a variant of the runlength coder that uses Rice's scheme. We shall mention that this modification does not change the asymptotic analysis; therefore, the resulting coder is still an entropy one.

3.2 Recursive Merging Algorithm

The Recursive Merging algorithm (RM) [7] is also a simple and well known algorithm that merges two sets A and B of sizes m and n, respectively, as follows: first it finds the location of a_i, where $i = \lfloor m/2 \rfloor$, in the list B, that is, the integer q for which $b_q < a_i < b_{q+1}$. Then, it recursively merges the lists $A[1,\ldots,i-1]$ and $B[1,\ldots,q]$ and then the lists $A[i+1,\ldots,m]$ and $B[q+1,\ldots,n]$. Its pseudocode is presented below:

$RM(A,B)$

1. Let $m = |A|$, $n = |B|$ and $i = \lfloor m/2 \rfloor$;
2. If $n = 0$ or $m = 0$ then exit;
3. If $n < m$ then swap m with n and A with B;
4. Do a binary search to find the location of a_i in B, that is, the integer q for which $b_q < a_i < b_{q+1}$;
5. RM($A[1,\ldots,i-1], B[1,\ldots,q]$);
6. RM($A[i+1,\ldots,m], B[q+1,\ldots,n]$).

By applying the conversion function over this merging algorithm we obtain a binary entropy coder that is closely related to the Binary Interpolative Coder (BIC) proposed by Moffat et. al. [15] for compressing inverted indexes. First, we describe BIC and then we explain its connection with RM.

Conversion. BIC receives three input parameters (A, lb, ub), where $A = \{a_1 < \ldots < a_m\}$ is an ordered set of integers and lb and ub are integers that satisfy $lb \leq a_1$ and $a_m \leq ub$, respectively. Let $i = \lfloor m/2 \rfloor$. Since all the integers of A are distincts it follows that a_i belongs to the interval $[lb+i, ub-i]$ of size $w = ub - lb - 2i + 1$. The first step of BIC consists of encoding a_i. For that it uses the minimal centered binary code [15] which generates $2^{\lceil \log_2 w \rceil} - w$ codewords of size $\lfloor \log_2 w \rfloor$ and $2w - 2^{\lceil \log_2 w \rceil}$ codewords of size $\lceil \log_2 w \rceil$. Next, BIC is recursively called for the input $(A[1,...,i-1], lb, a_i - 1)$ and then for the input $(A[i+1,...,m], a_i + 1, ub)$.

The connection between RM and BIC relies on the fact that the encoding produced by RM for a binary string x is similar to that produced by BIC for input $(A^x, 1, |x|)$. In fact, both encodings are exactly the same if the Step (3) is removed from RM and the binary search performed by RM at Step (4) is consistent with the encoding of a_i at the first step of BIC.

We illustrate this connection through an example. We use again the binary string x of the example of section 2.2. The first element to be merged is $A^x[4] = 11$. By applying the binary search induced by the tree of Figure 2, the following operations are executed $compare(11, 10)$, $compare(11, 16)$ and $compare(11, 14)$ which generate code 100. Then, RM is recursively called with input $(A^x[1, \ldots, 3], B^x[1, \ldots, 7])$ and then with input $(A^x[5, \ldots, 7], B^x[8, \ldots, 13])$.

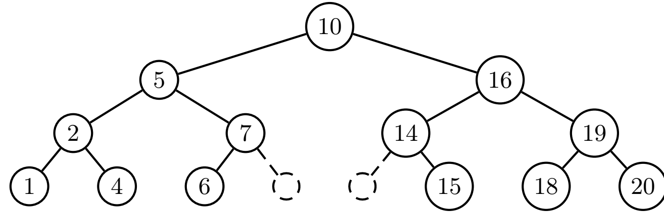

Figure 2: Minimal centered binary tree.

On the other hand, BIC receives $(A^x, 1, 20)$ as input parameters. It first encodes the number 11 using the interval $[1 + 3, 20 - 3] = [4, 17]$. Since there are $17 - 4 + 1 = 14$ possible numbers, then it requires 4 bits to encode any number in $[4..9, 12..17]$ and 3 bits to encode any number in $[10..11]$. Therefore, the number 11 is encoded as 100. Then, BIC is recursively called with input $(A^x[1, \ldots, 3], 1, 10)$ and then with input $(A^x[4, \ldots, 7], 12, 20)$.

Discussion. The Recursive Merging algorithm is asymptotically optimal and the authors did a simple analysis to show that is runs in $M^{RM}(m, n) = O\left(m \log_2\left(\frac{n}{m} + 1\right)\right)$. Nevertheless, we can use the more precise analysis of the BIC to show that it runs in $M^{RM}(m, n) < m\left(2.5783 + \log_2\left(\frac{n}{m} + 1\right)\right)$.

As we have already mentioned, the authors of BIC developed this coder for the problem of compressing inverted indexes. According to their research, this coder is suitable for encoding data that presents nonuniform (clustered) distribution. It is interesting to note that several merging algorithms [6][1][3] have been studied for this kind of data and, in the light of our result, they may be converted into suitable encoders for non-stationary sources, such as, inverted indexes and bilevel document images.

3.3 Probabilistic Merging Algorithm

The Probabilistic Merging (PM) algorithm [5] uses randomization techniques for reducing the average number of comparisons. Let $s = (\sqrt{5}-1)/2 \approx 0.618$ and $r = (\sqrt{2}-1+\sqrt{2}s)^2 \approx 1.659$. La Vega *et. al.* shows that PM performs better than the Binary Merging, presented in Section 3.1, when $|B|/|A| > 1 + s$, assuming that A and B are the linearly ordered subsets to be merged and that $|B| \geq |A|$. They further recommend the usage of tape merging [19] when $|B|/|A| \leq 1 + s$.

The algorithm receives two other parameters t and p, besides the two linearly ordered subsets A and B. The parameter t is related to the variable t of BM, but it remains constant during the merging process. The parameter p is used to define the probability of performing certain comparisons during the algorithm's execution. A discussion of how to set the values of p and t in order to minimize the average number of comparisons employed by PM is presented in [5].

PM can be seen as a variation of the BM, where a biased coin is flipped, at each step,

to determine whether b_{2^t} or $b_{2^{t+1}}$ should be compared with the first element of list A. Our presentation of PM is a bit different from that given in [5]:

$PM(A, B, p, t)$

1. If $A = \emptyset$ or $B = \emptyset$ then exit;

2. Initialize $i = 0$ and $j = 0$;

3. Flip a biased coin that outputs $'H'$ with probability p and $'T'$ with probability $1-p$:

   (a) If $'T'$ then $compare(a_1, b_{2^t})$:

      i. if $a_1 < b_{2^t}$ then binary search a_1 into the list $\{b_1 < \ldots < b_{2^t-1}\}$ and set $i = 1$;

      ii. if $a_1 > b_{2^t}$ then set $j = 2^t$;

   (b) if $'H'$ then $compare(a_1, b_{2^{t+1}})$:

      i. if $a_1 < b_{2^{t+1}}$ then binary search a_1 into the list $\{b_1 < \ldots < b_{2^{t+1}-1}\}$, set $j = 2^t$ if $a_1 > b_{2^t}$ and, finally, set $i = 1$;

      ii. if $a_1 > b_{2^{t+1}}$ then set $j = 2^{t+1}$;

4. $PM(A[1+i, \ldots, |A|], B[1+j, \ldots, |B|], p, t)$;

In order to maintain compatibility with the original algorithm of [5], we have to insert the list $\{-\infty < \ldots < -\infty\}$ of size $2^t - 1$ at the beginning of the subset B. Also, we assume that $b_k = \infty$ for $k > |B|$.

Conversion. By converting PM, we obtain a runlength-based coder that uses a variant of Rice code to represent the relative position of the bits zero. The pseudocode for encoding a binary string x into a binary string y is given below. The values of t and p are calculated the same way as the merging algorithm.

$ProbEncode(x, t, p)$

1. $j^* \leftarrow \min\{j | x(j) = 0\}$;

2. Append the output of $RandomizedRiceEncode(j^* - 1, t, p)$ to the output y;

3. Let x' be the binary string $(1, \ldots, 1)$ of size $j^* - 1 \bmod 2^t$;

4. $ProbEncode(x' \circ x[j^* + 1, \ldots, |x|], t, p)$;

The operation $u \bmod v$ calculates the rest of the integer division of u by v. The operation $u \circ v$ is the concatenation of the strings u and v. Again, for maintain compatibility with the merging algorithm, we insert at the beginning of x the binary string $(1, \ldots, 1)$ of size $2^t - 1$.

$RandomizedRiceEncode(j, t, p)$

1. Initialize the code s as empty;

2. $\{j \geq 2^{t+1}\}$ While $j \geq 2^{t+1}$ do:

(a) append one bit 0 to s;

(b) set $z = 0$ with probability $1 - p$ and $z = 1$ with probability p; decrement j by 2^{t+z};

3. $\{j < 2^{t+1}\}$ Set $z = 0$ with probability $1 - p$ and $z = 1$ with probability p;

   (a) if $z = 0$ then compare j with 2^t; if $j \geq 2^t$ then append one bit 0 to s, decrement j by 2^t and go to step 3; otherwise, if $j < 2^t$ then append one bit 1 to s;

   (b) if $z = 1$ then append one bit 1 to s and compare j with 2^t; if $j \geq 2^t$ then append one bit 0 to s and decrement j by 2^t; otherwise, if $j < 2^t$ then append one bit 1 to s;

4. $\{j < 2^t\}$ Append the binary code of j using t bits to s;

5. Return the code s.

The procedure above generates a variant of the Rice code, namely *Randomized Rice Code*. It is formed by two parts as in the original Rice code [16]: the unary and the binary part. Except that, in this case, the unary part is also parameterised by a probability p, besides the usual parameter t. This parameter is used to flip a biased coin for each unary bit and then, decide, by how much, decrement the value j. Therefore, for the same parameter values, it might be generated a different unary code.

We use again the example of section 2.2 to illustrate the encoding method. The value of the parameters t and p is 0 and 0.618, respectively, since $|B^x|/|A^x| = 1.857$. At each recursion, the encoding method calculates a value for j^* which generates the sequence of values $(3, 5, 1, 2, 1, 1, 4)$. It encodes these values using randomized Rice code with the respective random binary string (0011110111101) producing the sequence of codes $(0011, 0011, 1, 10, 11, 11, 0011)$. If we associate a different random binary string (10111010111) then it produces the sequence of codes $(01, 0011, 1, 10, 1, 11, 010)$.

Discussion. Let $E^{PM}(m, n)$ be the expected number of comparisons made by PM. The authors showed that $E^{PM}(m, n) < I(m, n) + 0.471m$, for values of m and n sufficiently large. We further observed that $E^{PM}(m, n) < (m+n)[H(m/(m+n), n/(m+n)) + 0.2355]$.

It is interesting to note that the new coder uses a random binary source to reduce its average redundancy. Therefore, the decoder has to receive a seed, which can be an integer, to generate the same random binary string employed by the encoder. Besides that, the decoder has also to receive the values m and n, if it implements a static model.

Acknowledgments

The authors are supported by CNPq.

References

[1] J. Barbay and C. Kenyon. Alternation and redundancy analysis of the intersection problem. *ACM Transactions on Algorithms*, 4(1), 2008.

[2] W. H. Burge. Sorting, trees, and measures of order. *Information and Control*, 1(3):181–197, 1958.

[3] S. Carlsson, C. Levcopoulos, and O. Petersson. Sublinear merging and natural mergesort. In *Proceedings of the Intl. Symposium on Algorithms*, pages 251–260, 1990.

[4] C. Christen. Improving the bound on optimal merging. In *Proceedings of the 19th IEEE Symposium on Foundation of Computer Science)*, pages 259–266, 1978.

[5] W. F. de La Vega, S. Kannan, and M. Santha. Two probabilistic results on merging. *SIAM Journal of Computing*, 22(2):261–271, April 1993.

[6] E. D. Demaine, A. López-Ortiz, and J. I. Munro. Adaptive set intersections, unions and differences. In *Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 743–752, 2000.

[7] K. Dudzinski and A. Dydek. On a stable storage merging algorithm. *Information Processing Letters*, 12:5–8, 1981.

[8] P. Elias. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory*, IT-21(2):194–203, January 1975.

[9] S. W. Golomb. Run-length codings. *IEEE Transactions on Information Theory*, 12(7):399–401, May 1966.

[10] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, pages 1098–1101, September 1952.

[11] F. K. Hwang and S. Lin. A simple algorithm for merging two disjoint linearly ordered lists. *SIAM Journal of Computing*, 1:31–39, 1972.

[12] D. E. Knuth. *The Art of Computer Programming: Sorting and Searching*, volume 3. Addison-Wesley, 1973.

[13] G. K. Manacher. Significant improvements to the Hwang-Lin merging algorithm. *Journal of ACM*, 26:434–440, 1979.

[14] A. Moffat, T. C. Bell, and I. H. Witten. Lossless compression for text and images. *International Journal of High Speed Electronics and Systems*, 8(1):179–231, October 1997.

[15] A. Moffat and L. Stuiver. Binary interpolative coding for effective index compression. *Information Retrieval*, 3(1):25–47, 2000.

[16] R. F. Rice. Some practical universal noiseless coding techniques. Technical Report 79-22, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, 1979.

[17] J. J. Rissanen and G. G. Langdon Jr. Arithmetic coding. *IBM Journal of Research and Development*, 23(2):146–162, March 1979.

[18] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 623–656, July 1948.

[19] P. K. Stockmeyer and F. F. Yao. On the optimality of linear merge. *SIAM Journal of Computing*, 9:85–90, 1980.