entropy

MDPI

Article

Fair Numerical Algorithm of Coset Cardinality Spectrum for Distributed Arithmetic Coding

Yong Fang * and Nan Yang

School of Information Engineering, Chang'an University, Xi'an 710064, China
* Correspondence: fy@chd.edu.cn

Abstract: As a typical symbol-wise solution of asymmetric Slepian-Wolf coding problem, Distributed Arithmetic Coding (DAC) non-linearly partitions source space into disjoint cosets with unequal sizes. The distribution of DAC coset cardinalities, named the Coset Cardinality Spectrum (CCS), plays an important role in both theoretical understanding and decoder design for DAC. In general, CCS cannot be calculated directly. Instead, a numerical algorithm is usually used to obtain an approximation. This paper first finds that the contemporary numerical algorithm of CCS is theoretically imperfect and does not finally converge to the real CCS. Further, to solve this problem, we refine the original numerical algorithm based on rigorous theoretical analyses. Experimental results verify that the refined numerical algorithm amends the drawbacks of the original version.

Keywords: distributed arithmetic coding; Slepian-Wolf coding; coset cardinality spectrum; numerical algorithm

Citation: Fang, Y.; Yang, N. Fair Numerical Algorithm of Coset Cardinality Spectrum for Distributed Arithmetic Coding. *Entropy* **2023**, *25*, 437. https://doi.org/10.3390/e25030437

Academic Editors: Jun Chen and Sadaf Salehkalaibar

Received: 12 October 2022
Revised: 15 December 2022
Accepted: 16 December 2022
Published: 1 March 2023

1. Introduction

As an important branch of network information theory, *Distributed Source Coding* (DSC) can find broad potential applications in many scenarios (e.g., wireless sensor network, genome compression, etc.). Just as traditional source coding, DSC has two forms. Lossless DSC is also called *Slepian-Wolf Coding* (SWC) [1]. The general form of lossy DSC is referred to as Berger-Tung coding [2], while a special case of asymmetric lossy DSC with side information at the decoder is referred to as Wyner-Ziv Coding (WZC) [3]. Up to now, the most important works on SWC have focused on its asymmetric form. Let X and Y be two correlated discrete random variables. Given Y available only at the decoder, the achievable rates of compressing X without loss are bounded by $R \geq H(X|Y)$. Different from SWC, an important property of WZC is that it usually suffers rate loss when compared to traditional lossy coding with side information available at both the encoder and the decoder. However, if the difference between source and side information is an independent Gaussian random variable, there is no rate loss [4,5].

This paper treats only asymmetric SWC. The asymmetric SWC problem is in essence a channel coding problem [6–9]. To show this point, one can take Y as a noisy version of X corrupted by virtual channel noise and take the bitstream of X as the index of the coset containing X. If the elements in each coset are spaced as far (in Hamming distance) as possible and Y is near (in Hamming distance) enough to X, then X can be recovered from its bitstream with the help of Y. From this viewpoint, the bitstream of X is actually the syndrome of a coset code. Hence traditionally, asymmetric SWC was implemented by channel codes (e.g., Turbo codes [10], *Low-Density Parity-Check* (LDPC) codes [11], and polar codes [12], etc.).

Arithmetic coding is usually deemed as the most important method for lossless data compression [13,14]. Due to the duality between source coding and channel coding, arithmetic coding can be easily modified to achieve the purpose of error detection and correction [15–17]. In 2007, people found that after some slight modifications, arithmetic

coding can also serve as a coset code to implement asymmetric SWC. There are mainly two approaches: One is *bit puncturing* [18] and the other is *interval enlarging* [19–21]. This paper will focus only on the latter approach, which is often referred to as *Distributed Arithmetic Coding* (DAC), while ignoring the former. For *independent and identically-distributed* (i.i.d.) binary sources, DAC is in general inferior to channel codes, e.g., LDPC codes, polar codes, etc., as shown by the experimental results in [22]. However, for binary sources with memory or nonbinary sources, DAC performs significantly better than channel codes [23–26].

In nature, DAC is a many-to-one nonlinear mapping that partitions source space into disjoint cosets of unequal cardinalities. Then an important problem is how DAC coset cardinality is distributed, which can be answered by the so-called *Coset Cardinality Spectrum* (CCS). The idea of CCS budded in [27,28] and was formally defined for uniform and nonuniform binary sources in [29,30], respectively. The concept of CCS is very useful as it not only can serve as a theoretical tool to analyze the properties of DAC [31,32], but also can be used to derive correct decoding formulae [31,32] (See the discussion in the last paragraph of Section 2).

If the stream of source symbols is grouped into length-n blocks, then DAC CCS is a tuple of $n+1$ *probability density functions* (pdfs). It is impossible to deduce the exact closed form for each pdf directly. Instead, one can begin with the final pdf, which is usually simple and calculable, and then derive each pdf via a backward recursion. In [29], a numerical algorithm was proposed to implement the backward recursion for uniform binary sources, and then it was generalized to nonuniform binary sources in [30].

However, the numerical algorithm proposed in [29,30] is very primitive and lacks theoretical justification. This paper will make an in-depth analysis of the numerical algorithm proposed in [29,30] and observe the results carefully. It will be found that the numerical algorithm proposed in [29,30] does not exactly converge to the real CCS. After a strict theoretical analysis, this paper will propose a novel numerical algorithm, which perfectly overcomes the drawbacks of the original numerical algorithm.

The rest of this paper is arranged as below. Section 2 briefly reviews the background knowledge of DAC and CCS. Section 3 presents the original numerical algorithm and its trivial upgrade version. Section 4 proposes the novel numerical algorithm for DAC CCS based on solid theoretical analyses. Section 5 reports some experimental results to compare three numerical algorithms. Finally, Section 6 concludes this paper.

2. Review on DAC and CCS

Let $X^n \triangleq (X_1, \ldots, X_n)$ be a length-n binary source block with bias probability $\Pr(X_i = 1) = p$. The entropy of X is $H(X) = -p\log_2 p - (1-p)\log_2(1-p)$. The DAC codec recursively maps every source symbol onto an interval in $[0,1)$ according to the following rule

$$x \in \mathbb{B} \triangleq \{0,1\} \to [x(1-p^r), (1-x)(1-p)^r + x) \subset [0,1), \tag{1}$$

where $r \in [0,1]$ is called the normalized rate or overlapping factor. Let $[l(X^n), h(X^n)) \subset [0,1)$ denote the mapping interval of X^n. In theory, $[l(X^n), h(X^n))$ can be represented by $-\log_2(h(X^n) - l(X^n)) \in \mathbb{R}$ bits. However, due to the indivisibility of a bit, the bitstream of X^n actually includes $-\lfloor \log_2(h(X^n) - l(X^n)) \rfloor \in \mathbb{Z}$ bits, which can be explained as a real number U_0 in $[l(X^n), h(X^n))$. Hence, there is a rate loss of δ bits, where

$$\delta \triangleq \log_2(h(X^n) - l(X^n)) - \lfloor \log_2(h(X^n) - l(X^n)) \rfloor \in [0,1). \tag{2}$$

If we decode the bitstream U_0 along the path X^n, then we will obtain a tuple of $n+1$ real numbers $U_0^n \triangleq (U_0, U_1, \ldots, U_n)$ which can be deduced recursively. The forward recursion of U_0^n can be easily transformed into an equivalent backward recursion as shown below [30].

$$\begin{cases} U_{i-1} = (1-p)^r U_i, & X_i = 0 \\ U_{i-1} = p^r U_i + (1-p^r), & X_i = 1 \end{cases}. \tag{3}$$

The pdf of U_i is called the i-th CCS and denoted by $f_{n,i}(u)$, $0 \le u < 1$, which is usually simplified as $f_i(u)$. Especially, $f_0(u)$ is called the initial CCS and $f_n(u)$ is called the final CCS.

At this point, we have two choices: One is beginning from the initial CCS $f_0(u)$ and then deducing $f_i(u)$ for $0 < i \le n$ via a forward recursion; the other is beginning from the final CCS $f_n(u)$ and then deducing $f_i(u)$ for $0 \le i < n$ via a backward recursion. Unfortunately, it is impossible to deduce the initial CCS $f_0(u)$ directly, so forward recursion is infeasible. Instead, it is proved in [30] that depending on the parameters p and r, $f_n(u)$ will tend to be a piecewise uniform function or

$$\lim_{n\to\infty} f_n(u) = \frac{\log_2 e}{1+|1-2u|}. \tag{4}$$

Once $f_n(u)$ is known, $f_i(u)$ for $0 \le i < n$ can be deduced via a backward recursion [30].

$$f_{i-1}(u) = (1-p)^{1-r} f_i(u(1-p)^{-r}) + p^{1-r} f_i((u-(1-p^r))p^{-r}). \tag{5}$$

In general, it is impossible to deduce $f_{i-1}(u)$ from $f_i(u)$ analytically, even though the closed form of $f_i(u)$ is known. Hence, we have to resort to a numerical algorithm, which mimics (5) in a discrete way.

Regarding the physical meaning of CCS, there is a detailed explanation in [28]. Let us take uniform binary sources as an example. For any real number $u \in [0,1)$, if u is fed into a DAC decoder with overlapping factor r, then we will get an incomplete binary tree. In this tree, the number of level-i nodes is roughly $(\lim_{n\to\infty} f_{n,n-i}) \cdot 2^{i(1-r)}$. If we implement the decoder in a breadth-first way, i.e., with the M-algorithm, for every level-i node, the sub-tree grown from this level-i node will have about $(\lim_{n\to\infty} f_{n,i}(u)) \cdot 2^{(n-i)(1-r)}$ leaf nodes, where u is the real number at the level-i node. Obviously, those level-i nodes with larger $(\lim_{n\to\infty} f_{n,i}(u))$ should be more likely, and vice versa. That is why DAC CCS can be used to derive correct decoding formulae.

3. Original Numerical Algorithms

3.1. Rounding Numerical Algorithm

The first version of the numerical algorithm was proposed in [28,30] for uniform and nonuniform binary sources, respectively. This algorithm divides the interval $[0,1)$ into N segments and then uses a finite number of $f_i(j/N)$'s, where $j \in [0:N) \triangleq \{0, \ldots, N-1\}$, to approximate $f_i(u)$. For simplicity, we use $\hat{f}_i(j)$ to denote the approximation of $f_i(j/N)$. Then (5) can be discretized as

$$\hat{f}_{i-1}(j) = (1-p)^{1-r} \hat{f}_i(j') + p^{1-r} \hat{f}_i(j''). \tag{6}$$

Now the key is to find a good mapping from j to j' and j''. Let $\lfloor \cdot \rceil$ denote the rounding operation. This problem was solved by a brute-force method in [28,30] as follows:

$$\begin{cases} j' = \lfloor j(1-p)^{-r} \rceil \\ j'' = \lfloor (j - N(1-p^r))p^{-r} \rceil \end{cases}. \tag{7}$$

For some $j \in [0:N)$, we will have $j' \notin [0:N)$ or $j'' \notin [0:N)$. It does not matter because we have $\hat{f}(j) \equiv 0$ for any $j \notin [0:N)$ according to the definition of CCS. Since the core of (7) is the rounding operation, this method will be formally referred to as *rounding numerical algorithm* below.

Though the rounding numerical algorithm works well at first glance [28,30], its rationality is indeed flawed. On one hand, since $(1-p)^{-r} > 1$, we have $j(1-p)^{-r} \geq j$ and thus among N points of $\hat{f}_i(j)$'s, only about $N(1-p)^r < N$ points, i.e., $0 \leq j < j_{\max} \approx N(1-p)^r$, are used for the first line of (7). On the other hand,

$$\begin{aligned}(j - N(1-p^r))p^{-r} &= N - (N-j)p^{-r} \\ &= (N-j) - (N-j)p^{-r} + j \\ &= (N-j)(1-p^{-r}) + j < j,\end{aligned} \tag{8}$$

where the last inequality is due to $N > j$ and $p^{-r} > 1$. Thus among N points of $\hat{f}_i(j)$'s, only about $Np^r < N$ points, i.e., $N(1-p^r) \approx j_{\min} \leq j < N$, are used for the second line of (7). In summary, not all N points of $\hat{f}_i(j)$'s will be used to generate $\hat{f}_{i-1}(j)$'s. In other words, some points of $\hat{f}_i(j)$'s are discarded without being used when we try to deduce $\hat{f}_{i-1}(j)$ according to (6), which is equivalent to the case that partial information of $f_i(u)$ is lost when we try to deduce $f_{i-1}(u)$ according to (5). This is imperfect in theory and may bring two negative effects in practice:

- The rounding numerical algorithm cannot generate an accurate approximation of CCS. This phenomenon will be observed in the experimental results of Section 5.
- Even if $\sum_{j=0}^{N-1} \hat{f}_i(j) = N$, (6) does not strictly satisfy the normalization condition $\sum_{j=0}^{N-1} \hat{f}_{i-1}(j) = N$. Thus an extra re-normalization step is needed after (6).

3.2. Linear Numerical Algorithm

To improve the performance of the rounding numerical algorithm, more points of $\hat{f}_i(j)$'s should be involved in generating $\hat{f}_{i-1}(j)$'s. This problem can be solved by linear interpolation. Below we will propose a trivial upgrade of the rounding numerical algorithm, which we will refer to as *linear numerical algorithm*. Let $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ denote the flooring and ceiling operations, respectively. Let us define $\alpha \triangleq j(1-p)^{-r} \geq j$ and $\beta \triangleq (j - N(1-p^r))p^{-r} < j$. In general, (6) can be refined as

$$\hat{f}_{i-1}(j) = (1-p)^{1-r} g_0(\alpha) + p^{1-r} g_1(\beta), \tag{9}$$

where

$$\begin{cases} g_0(\alpha) = (\lceil \alpha \rceil - \alpha)\hat{f}_i(\lfloor \alpha \rfloor) + (\alpha - \lfloor \alpha \rfloor)\hat{f}_i(\lceil \alpha \rceil) \\ g_1(\beta) = (\lceil \beta \rceil - \beta)\hat{f}_i(\lfloor \beta \rfloor) + (\beta - \lfloor \beta \rfloor)\hat{f}_i(\lceil \beta \rceil) \end{cases}. \tag{10}$$

Then we discuss some special cases:

- If $\alpha \in \mathbb{Z}$, then $g_0(\alpha) = \hat{f}_i(\alpha)$. If $\beta \in \mathbb{Z}$, then $g_1(\beta) = \hat{f}_i(\beta)$.
- If $\lfloor \alpha \rfloor \geq N$, then $g_0(\alpha) = 0$. If $\lceil \beta \rceil < 0$, then $g_1(\beta) = 0$.
- If $\lfloor \alpha \rfloor < N$ and $\lceil \alpha \rceil \geq N$, then $g_0(\alpha) = \hat{f}_i(\lfloor \alpha \rfloor)$. If $\lfloor \beta \rfloor < 0$ and $\lceil \beta \rceil \geq 0$, then $g_1(\beta) = \hat{f}_i(\lceil \beta \rceil)$.

However, it must be pointed out that the linear numerical algorithm still does not guarantee that all N points of $\hat{f}_i(j)$'s are used to generate $\hat{f}_{i-1}(j)$'s, and a renormalization step is still needed after (9).

4. Fair Numerical Algorithm

In the rounding/linear numerical algorithms, we take $\hat{f}_i(j)$ as an approximation of $f_i(j/N)$, i.e., $\hat{f}_i(j) \approx f_i(j/N)$ for any $j \in [0:N)$, and for each $\hat{f}_{i-1}(j)$, we try to find the corresponding $\hat{f}_i(j')$. This understanding is however incorrect. The correct physical

meaning of $\hat{f}_i(j)$ should be an approximation of the scaled-up probability of U_i falling into the interval $[j/N, (j+1)/N))$, i.e.,

$$\hat{f}_i(j) \approx N \int_{j/N}^{(j+1)/N} f_i(u)\, du. \tag{11}$$

According to (5), we have

$$\int_{j/N}^{(j+1)/N} f_{i-1}(u)\, du = \int_{j/N}^{(j+1)/N} \Big((1-p)^{1-r} f_i(u(1-p)^{-r}) + p^{1-r} f_i((u-(1-p^r))p^{-r}) \Big)\, du. \tag{12}$$

Let $u' \triangleq u(1-p)^{-r}$ and $u'' \triangleq (u-(1-p^r))p^{-r}$. Let

$$\begin{cases} \mathcal{I}_0 \triangleq [j(1-p)^{-r}/N, (j+1)(1-p)^{-r}/N) \\ \mathcal{I}_1 \triangleq [(j-N(1-p^r))p^{-r}/N, (j+1-N(1-p^r))p^{-r}/N) \end{cases}. \tag{13}$$

Then we can obtain

$$\begin{aligned} \int_{j/N}^{(j+1)/N} f_{i-1}(u)\, du &= (1-p) \int_{u' \in \mathcal{I}_0} f_i(u')\, du' + p \int_{u'' \in \mathcal{I}_1} f_i(u'')\, du'' \\ &= (1-p) \int_{u \in \mathcal{I}_0} f_i(u)\, du + p \int_{u \in \mathcal{I}_1} f_i(u)\, du. \end{aligned} \tag{14}$$

Naturally, we have

$$\hat{f}_{i-1}(j) = (1-p) \cdot g_0(j) + p \cdot g_1(j), \tag{15}$$

where

$$\begin{cases} g_0(j) \approx N \int_{u \in \mathcal{I}_0} f_i(u)\, du \\ g_1(j) \approx N \int_{u \in \mathcal{I}_1} f_i(u)\, du \end{cases}. \tag{16}$$

In plain words, $g_0(j)$ is an approximation of the scaled-up probability of U_i falling into the interval $\mathcal{I}_0$, and $g_1(j)$ is an approximation of the scaled-up probability of U_i falling into the interval $\mathcal{I}_1$.

4.1. Calculation of $g_0(j)$

Let $\lambda_0 \triangleq j(1-p)^{-r} \geq 0$ and $\eta_0 \triangleq (j+1)(1-p)^{-r} > \lambda_0 \geq 0$. Then we can obtain

$$\begin{aligned} N \int_{u \in \mathcal{I}_0} f_i(u)\, du = N \int_{\lambda_0/N}^{(\lfloor \lambda_0 \rfloor + 1)/N} f_i(u)\, du + N \int_{(\lfloor \lambda_0 \rfloor + 1)/N}^{(\lfloor \lambda_0 \rfloor + 2)/N} f_i(u)\, du + \dots \\ + N \int_{(\lfloor \eta_0 \rfloor - 1)/N}^{\lfloor \eta_0 \rfloor / N} f_i(u)\, du + N \int_{\lfloor \eta_0 \rfloor / N}^{\eta_0/N} f_i(u)\, du. \end{aligned} \tag{17}$$

For N sufficiently large, $f_i(u)$ can be taken as uniform over $[j/N, (j+1)/N)$. Hence we have

$$\begin{aligned} N \int_{\lambda_0/N}^{(\lfloor \lambda_0 \rfloor + 1)/N} f_i(u)\, du &\approx (\lfloor \lambda_0 \rfloor + 1 - \lambda_0) \cdot N \int_{\lfloor \lambda_0 \rfloor / N}^{(\lfloor \lambda_0 \rfloor + 1)/N} f_i(u)\, du \\ &\approx (\lfloor \lambda_0 \rfloor + 1 - \lambda_0) \cdot \hat{f}_i(\lfloor \lambda_0 \rfloor) \end{aligned} \tag{18}$$

and

$$N\int_{\lfloor\eta_0\rfloor/N}^{\eta_0/N} f_i(u)\,du \approx (\eta_0-\lfloor\eta_0\rfloor)\cdot N\int_{\lfloor\eta_0\rfloor/N}^{(\lfloor\eta_0\rfloor+1)/N} f_i(u)\,du \approx (\eta_0-\lfloor\eta_0\rfloor)\cdot \hat{f}_i(\lfloor\eta_0\rfloor). \tag{19}$$

According to the above analysis, we can obtain the following results.

- $N \le \lambda_0 < \eta_0$: It is easy to know $g_0(j) \equiv 0$.
- $\lambda_0 < N \le \eta_0$: In general, we have

$$g_0(j) = (1-(\lambda_0-\lfloor\lambda_0\rfloor))\cdot \hat{f}_i(\lfloor\lambda_0\rfloor) + \sum_{j'=\lfloor\lambda_0\rfloor+1}^{N-1} \hat{f}_i(j'). \tag{20}$$

  Especially, if $\lambda_0 \in \mathbb{Z}$, then

$$g_0(j) = \sum_{j'=\lambda_0}^{N-1} \hat{f}_i(j'). \tag{21}$$

- $\lambda_0 < \eta_0 < N$: In general, we have

$$g_0(j) = (1-(\lambda_0-\lfloor\lambda_0\rfloor))\cdot \hat{f}_i(\lfloor\lambda_0\rfloor) + (\eta_0-\lfloor\eta_0\rfloor)\cdot \hat{f}_i(\lfloor\eta_0\rfloor) + \sum_{j'=\lfloor\lambda_0\rfloor+1}^{\lfloor\eta_0\rfloor-1} \hat{f}_i(j'). \tag{22}$$

  Let us consider three special cases:

  – If $\lambda_0 \in \mathbb{Z}$ and $\eta_0 \notin \mathbb{Z}$, then

$$g_0(j) = (\eta_0-\lfloor\eta_0\rfloor)\cdot \hat{f}_i(\lfloor\eta_0\rfloor) + \sum_{j'=\lambda_0}^{\lfloor\eta_0\rfloor-1} \hat{f}_i(j'). \tag{23}$$

  – If $\lambda_0 \notin \mathbb{Z}$ and $\eta_0 \in \mathbb{Z}$, then

$$g_0(j) = (\lceil\lambda_0\rceil-\lambda_0)\cdot \hat{f}_i(\lfloor\lambda_0\rfloor) + \sum_{j'=\lceil\lambda_0\rceil}^{\eta_0-1} \hat{f}_i(j'). \tag{24}$$

  – If $\lambda_0 \in \mathbb{Z}$ and $\eta_0 \in \mathbb{Z}$, then

$$g_0(j) = \sum_{j'=\lambda_0}^{\eta_0-1} \hat{f}_i(j'). \tag{25}$$

4.2. Calculation of $g_1(j)$

Let $\eta_1 \triangleq (j+1-N(1-p^r))p^{-r} \le N$ and $\lambda_1 \triangleq (j-N(1-p^r))p^{-r} < \eta_1 \le N$. Similarly, we can get the following results.

- $\lambda_1 < \eta_1 < 0$: It is easy to know $g_1(j) \equiv 0$.
- $\lambda_1 < 0 \le \eta_1$: In general, we have

$$g_1(j) = (\eta_1-\lfloor\eta_1\rfloor)\cdot \hat{f}_i(\lfloor\eta_1\rfloor) + \sum_{j'=0}^{\lfloor\eta_1\rfloor-1} \hat{f}_i(j'). \tag{26}$$

Especially, if $\eta_1 \in \mathbb{Z}$, then

$$g_1(j) = \sum_{j'=0}^{\eta_1-1} \hat{f}_i(j'). \tag{27}$$

- $0 \le \lambda_1 < \eta_1$: In general, we have

$$g_1(j) = (1-(\lambda_1 - \lfloor \lambda_1 \rfloor)) \cdot \hat{f}_i(\lfloor \lambda_1 \rfloor) + (\eta_1 - \lfloor \eta_1 \rfloor) \cdot \hat{f}_i(\lfloor \eta_1 \rfloor) + \sum_{j'=\lfloor \lambda_1 \rfloor+1}^{\lfloor \eta_1 \rfloor-1} \hat{f}_i(j'). \tag{28}$$

Let us consider three special cases:

– If $\lambda_1 \in \mathbb{Z}$ and $\eta_1 \notin \mathbb{Z}$, then

$$g_1(j) = (\eta_1 - \lfloor \eta_1 \rfloor) \cdot \hat{f}_i(\lfloor \eta_1 \rfloor) + \sum_{j'=\lambda_1}^{\lfloor \eta_1 \rfloor-1} \hat{f}_i(j'). \tag{29}$$

– If $\lambda_1 \notin \mathbb{Z}$ and $\eta_1 \in \mathbb{Z}$, then

$$g_1(j) = (\lceil \lambda_1 \rceil - \lambda_1) \cdot \hat{f}_i(\lfloor \lambda_1 \rfloor) + \sum_{j'=\lceil \lambda_1 \rceil}^{\eta_1-1} \hat{f}_i(j'). \tag{30}$$

– If $\lambda_1 \in \mathbb{Z}$ and $\eta_1 \in \mathbb{Z}$, then

$$g_1(j) = \sum_{j'=\lambda_1}^{\eta_1-1} \hat{f}_i(j'). \tag{31}$$

4.3. Discussion

From the above analysis, it can be found that all N points of $\hat{f}_i(j)$'s are made use of to generate $\hat{f}_{i-1}(j)$'s, and such treatment is fair for every $\hat{f}_i(j)$. For this reason, we will formally refer to this method as *fair numerical algorithm*. Given $\sum_{j=0}^{N-1} \hat{f}_i(j) = N$, it is easy to know $\sum_{j=0}^{N-1} g_0(j) = \sum_{j=0}^{N-1} g_1(j) = N$. Hence, (15) satisfies the normalization property by itself and no renormalization step is needed after (15).

Let us briefly discuss the convergence of numerical algorithms. Let $f(u)$ denote the asymptotic form of $f_{n,0}(u)$ as $n \to \infty$. According to (5), it is obvious that

$$f(u) = (1-p)^{1-r} f(u(1-p)^{-r}) + p^{1-r} f((u-(1-p^r))p^{-r}). \tag{32}$$

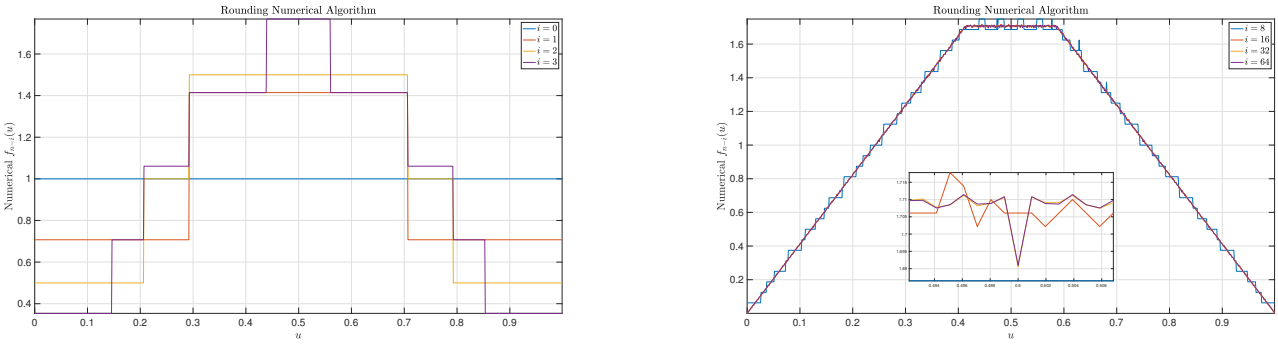
For the rounding/linear numerical algorithms, as analyzed above, partial parts of $f(u)$ have been lost, so the discrete version of (32) will not exactly hold, while for the fair numerical algorithm, since all information of $f(u)$ is reserved, the discrete version of (32) will exactly hold as the number of segments N goes to infinity.

5. Experimental Results

We use a classical CCS to compare three numerical algorithms. Given $p = r = 0.5$, for n sufficiently large, as i increases, f_{n-i} will tend to be ladder-shaped and we have [27,28].

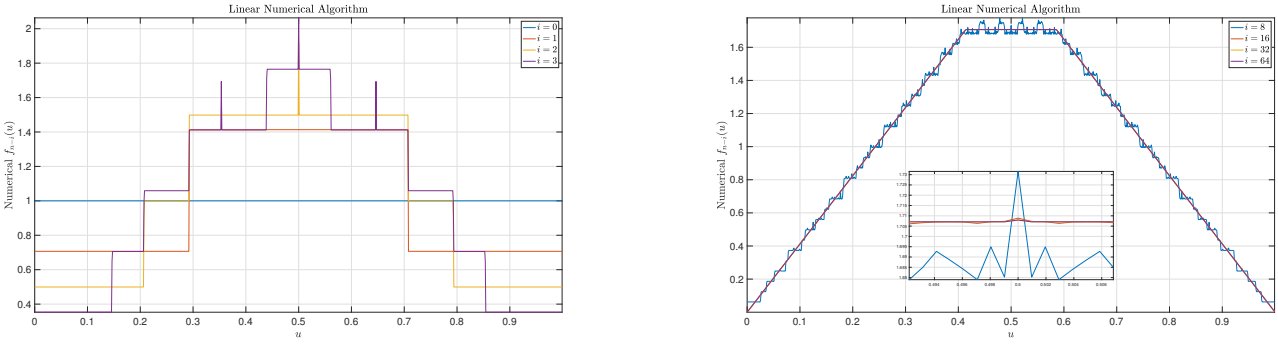
$$\lim_{i\to\infty} f_{\infty-i}(u) = \begin{cases} \frac{u}{3\sqrt{2}-4}, & 0 \le u < \sqrt{2}-1 \\ \frac{1}{2-\sqrt{2}}, & \sqrt{2}-1 \le u < 2-\sqrt{2} \\ \frac{1-u}{3\sqrt{2}-4}, & 2-\sqrt{2} \le u < 1 \end{cases}. \tag{33}$$

For simplicity, we set the final CCS $f_n(u) = \Pi(u)$, where $\Pi(u)$ is a uniform function over $[0, 1)$. Some results are included in Figures 1–3. In these figures, we set $n = 128$ and $N = 1024$. It can be observed that for small i (e.g., 1, 2, and 3), the rounding numerical algorithm performs almost as well as the fair numerical algorithm, while the linear numerical algorithm performs very poorly as there are some big spikes. When $i = 8$, small spikes are also observed for the rounding numerical algorithm, but no spike is observed for the fair numerical algorithm. Finally, for large i (e.g., 16, 32, and 64), the resulting curves of the rounding numerical algorithm always fluctuate slightly along the real CCS given by (33), and more experiments show that there is no trend that these curves will finally converge to the real CCS as i increases. On the contrary, for both the linear numerical algorithm and the fair numerical algorithm, their curves coincide with the real CCS very well for large i, and there is a trend that these curves will finally converge to the real CCS as i increases.



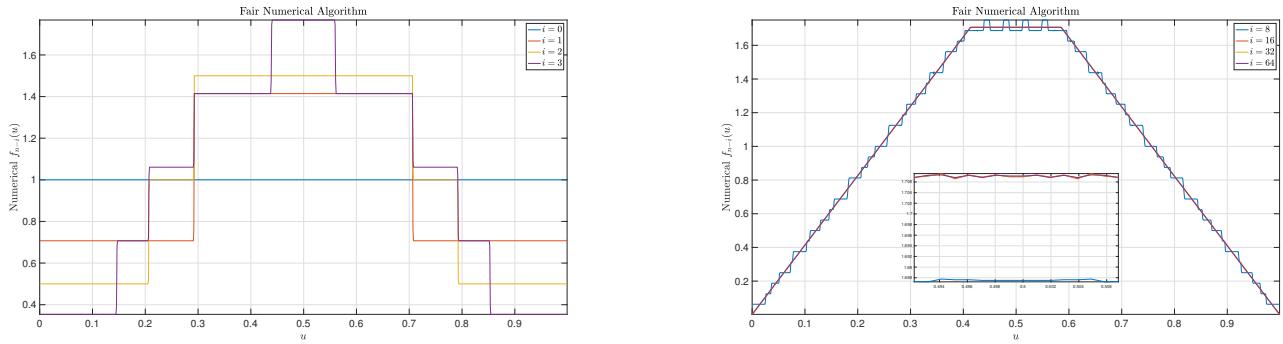
(**a**) (**b**)

Figure 1. Some examples of the rounding numerical algorithm. Though the rounding numerical algorithm performs well for small i, it always fluctuates slightly along the real CCS for large i and will not finally converge to the real CCS as i increases. (**a**): Rounding numerical algorithm for ending symbols. (**b**): Rounding numerical algorithm for starting symbols.



(**a**) (**b**)

Figure 2. Some examples of the linear numerical algorithm. Though the linear numerical algorithm performs well for large i, i.e., the curves coincide with the real CCS well, it will cause big spikes for small i. (**a**): Linear numerical algorithm for ending symbols. (**b**): Linear numerical algorithm for starting symbols.



(**a**) (**b**)

Figure 3. Some examples of the fair numerical algorithm. It overcomes all weaknesses of the rounding/linear numerical algorithms. (**a**): Fair numerical algorithm for ending symbols. (**b**): Fair numerical algorithm for starting symbols.

6. Conclusions

As an important property of DAC, CCS finds its broad applications in many scenarios. However, CCS is usually incalculable so we have to resort to numerical algorithms. This paper finds that the original numerical algorithm proposed in our previous papers is not sound in theory and does not work well in practice (fails to generate an accurate approximation of CCS). Based on a strict theoretical analysis, this paper proposes a novel numerical algorithm that overcomes the weaknesses of the original numerical algorithm. The superiority of the newly-proposed numerical algorithm is well validated by experimental results.

A software package of source codes to reproduce the experimental results in this paper has been released in [33].

Author Contributions: Conceptualization, Y.F.; Methodology, Y.F.; Software, Y.F. and N.Y.; Writing—original draft, Y.F.; Writing—review & editing, N.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Science Foundation of China under Grant 62141101.

Institutional Review Board Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Slepian, D.; Wolf, J.K., Noiseless coding of correlated information sources. *IEEE Trans. Inf. Theory* **1973**, *19*, 471–480. [CrossRef]
2. Berger, T. Multiterminal source coding. In *The Information Theory Approach to Communications*; Longo, G., Ed.; Springer: New York, NY, USA, 1977.
3. Wyner, A.; Ziv, J. The rate-distortion function for source coding with side information at the decoder. *IEEE Trans. Inf. Theory* **1976**, *22*, 1–10. [CrossRef]
4. Wyner, A. The rate-distortion function for source coding with side information at the decoder-II: General sources. *Inf. Control* **1978**, *38*, 60–80. [CrossRef]
5. Pradhan, S.; Chou, J.; Ramchandran, K. Duality between source coding and channel coding and its extension to the side information case. *IEEE Trans. Inf. Theory* **2003**, *49*, 1181–1203. [CrossRef]
6. Chen, J.; He, D.-K.; Jagmohan, A. On the duality between Slepian-Wolf coding and channel coding under mismatched decoding. *IEEE Trans. Inf. Theory* **2009**, *55*, 4006–4018. [CrossRef]
7. Chen, J.; He, D.-K.; Jagmohan, A.; Lastras-Montano, L.A.; Yang, E.-H. On the linear codebook-level duality between Slepian-Wolf coding and channel coding. *IEEE Trans. Inf. Theory* **2009**, *55*, 5575–5590. [CrossRef]
8. Chen, J.; He, D.-K.; Jagmohan, A. The equivalence between Slepian-Wolf coding and channel coding under density evolution. *IEEE Trans. Commun.* **2009**, *57*, 2534–2540. [CrossRef]

9. Chen, J.; He, D.-K.; Jagmohan, A.; Lastras-Montano, L.A. On the reliability function of variable-rate Slepian-Wolf coding. *Entropy* **2017**, *19*, 389. [CrossRef]
10. Garcia-Frias, J.; Zhao, Y. Compression of correlated binary sources using turbo codes. *IEEE Commun. Lett.* **2001**, *5*, 417–419. [CrossRef]
11. Liveris, A.; Xiong, Z.; Georghiades, C. Compression of binary sources with side information at the decoder using LDPC codes. *IEEE Commun. Lett.* **2002**, *6*, 440–442. [CrossRef]
12. Bilkent, E. Polar coding for the Slepian-Wolf problem based on monotone chain rules. In Proceedings of the IEEE International Symposium on Information Theory and Its Applications (ISITA2012), Honolulu, HI, USA, 28–31 October 2012; pp. 566–570.
13. Rissanen, J. Generalized Kraft inequality and arithmetic coding. *IBM J. Res. Dev.* **1976**, *20*, 198–203. [CrossRef]
14. Witten, I.; Neal, R.; Cleary, J. Arithmetic coding for data compression. *Commun. ACM* **1987**, *30*, 520–540. [CrossRef]
15. Boyd, C.; Cleary, J.; Irvine, S.; Rinsma-Melchert, I.; Witten, I. Integrating error detection into arithmetic coding. *IEEE Trans. Commun.* **1997**, *45*, 1–3. [CrossRef]
16. Anand, R.; Ramchandran, K.; Kozintsev, I.V. Continuous error detection (CED) for reliable communication. *IEEE Trans. Commun.* **2001**, *49*, 1540–1549. [CrossRef]
17. Grangetto, M.; Cosman, P.; Olmo, G. Joint source/channel coding and MAP decoding of arithmetic codes. *IEEE Trans. Commun.* **2005**, *53*, 1007–1016. [CrossRef]
18. Malinowski, S.; Artigas, X.; Guillemot, C.; Torres, L. Distributed coding using punctured quasi-arithmetic codes for memory and memoryless sources. *IEEE Trans. Signal Process.* **2009**, *57*, 4154–4158. [CrossRef]
19. Grangetto, M.; Magli, E.; Olmo, G. Distributed arithmetic coding. *IEEE Commun. Lett.* **2007**, *11*, 883–885. [CrossRef]
20. Grangetto, M.; Magli, E.; Olmo, G. Distributed arithmetic coding for the Slepian-Wolf problem. *IEEE Trans. Signal Process.* **2009**, *57*, 2245–2257. [CrossRef]
21. Artigas, X.; Malinowski, S.; Guillemot, C.; Torres, L. Overlapped quasi-arithmetic codes for distributed video coding. *Proc. IEEE ICIP* **2007**, *II*, 9–12.
22. Yang, N.; Fang, Y.; Wang, L.; Wang, Z.; Jiang, F. Approximation of initial coset cardinality spectrum of distributed arithmetic coding for uniform binary sources. *IEEE Commun. Lett.* 2022, *in progress to appear*. [CrossRef]
23. Fang, Y.; Jeong, J. Distributed arithmetic coding for sources with hidden Markov correlation. *arXiv* **2008**, arXiv:2101.02336.
24. Fang, Y. *Q*-ary distributed arithmetic coding for uniform *Q*-ary sources. *IEEE Trans. Inf. Theory*, *in progress to appear*. [CrossRef]
25. Zhou, J.; Wong, K.; Chen, J. Distributed block arithmetic coding for equiprobable sources. *IEEE Sens. J.* **2013**, *13*, 2750–2756. [CrossRef]
26. Wang, Z.; Mao, Y.; Kiringa, I. Non-binary distributed arithmetic coding. In Proceedings of the IEEE 14th Canadian Workshop Information Theory (CWIT), St. John's, NL, Canada, 6–9 July 2015; pp. 5–8.
27. Fang, Y. Distribution of distributed arithmetic codewords for equiprobable binary sources. *IEEE Signal Process. Lett.* **2009**, *16*, 1079–1082. [CrossRef]
28. Fang, Y. DAC spectrum of binary sources with equally-likely symbols. *IEEE Trans. Commun.* **2013**, *61*, 1584–1594. [CrossRef]
29. Fang, Y.; Stankovic, V.; Cheng, S.; Yang, E.-H. Analysis on tailed distributed arithmetic codes for uniform binary sources. *IEEE Trans. Commun.* **2016**, *64*, 4305–4319. [CrossRef]
30. Fang, Y.; Stankovic, V. Codebook cardinality spectrum of distributed arithmetic coding for independent and identically-distributed binary sources. *IEEE Trans. Inf. Theory* **2020**, *66*, 6580–6596. [CrossRef]
31. Fang, Y.; Chen, L. Improved binary DAC codec with spectrum for equiprobable sources. *IEEE Trans. Commun.* **2014**, *62*, 256–268. [CrossRef]
32. Fang, Y. Two applications of coset cardinality spectrum of distributed arithmetic coding. *IEEE Trans. Inf. Theory* **2021**, *67*, 8335–8350. [CrossRef]
33. GitHub. Available online: https://github.com/fy79/dac_ccs_num (accessed on 15 December 2022).