采用简化重启机制的极化码连续消除翻转译码

Ilshat Sagitov*、Charles Pillet*、Alexios Balatsoukas-Stimming†和Pascal Giard*
*电气工程系，´高等技术学院，蒙特利尔，魁北克省，加拿大。电子邮件：{ilshat.sagitov.1、charles.pillet.1}@ens.etsmtl.ca、pascal.giard@etsmtl.ca
†电气工程系，埃因霍温理工大学，荷兰埃因霍温。电子邮件：a.k.balatsoukas.stimming@tue.nl

arXiv:2303.01561v1 [cs.IT] 2 Mar 2023

摘要—极化码是一类能够被证明可达到实际信道容量的纠错码。连续消除翻转（SCF）译码器是一种低复杂度译码器，旨在提升连续消除（SC）译码器的性能，以替代复杂度较高的连续消除列表（SCL）译码器。SCF译码器提升了SC译码器的纠错性能，但其执行时间不固定且最坏情况执行时间较长，这给采用定时算法的接收机实现带来了挑战。SCF译码器的动态SCF（DSCF）变体进一步提升了纠错性能，但译码延迟问题仍然存在。本文提出一种简化重启机制（SRM），通过有条件地重启从码字后半部分开始的额外试验，减少SCF和DSCF译码器的执行时间。我们证明，该机制能够改善SCF和DSCF译码器的执行时间特性，同时保持纠错性能不变。对于一种每次译码试次最多可同时翻转3个比特的DSCF译码器，其平均执行时间、平均额外执行时间和执行时间方差分别减少约31%、37%和57%。在此设置下，该机制需要约3.9%的额外存储器。

I. 引言

极化码 [1]是一类线性纠错码，在采用低复杂度连续消除（SC）译码时，可在实际相关信道上达到信道容量。然而，在短至中等码块长度下，SC 算法的纠错性能无法满足许多实际应用的需求。为此，人们提出了连续消除列表（SCL）译码算法 [2]。该算法具有出色的纠错能力，因此极化码被选用于保护 3GPP 下一代移动通信标准（5G）中的控制信道，其中 SCL 被用作纠错性能基准 [3]。然而，SCL译码器的纠错能力是以较高的硬件实现复杂度和较低的能效为代价的 [4]。

作为 SCL 译码的一种替代方案，文献 [5] 提出了连续消除翻转（SCF）译码算法。与连续消除（SC）相比，SCF 的纠错性能有所提升，但仍逊于列表大小适中的 SCL 译码器。不过，无论在计算资源还是能耗方面，SCF译码器都比 SCL 更高效 [6]。文献 [7], 提出的动态 SCF（DSCF）译码显著提升了纠错性能。DSCF 采用更优的度量值来识别比特翻转候选项，并采用多比特翻转方法。硬件实现的初步结果表明，与列表大小适中的 SCL 译码器相比，DSCF译码器在提供相近纠错性能的同时，能效更高 [8]。

SCF 和动态连续消除翻转译码器的执行时间均不固定，且执行时间的方差可能很大。这给接收机的实现带来了挑战，因为接收机更倾向于采用定时算法。文献 [9]提出了一种针对单比特翻转DSCF译码器的早停机制，旨在改善其执行时间特性。然而，该机制会降低纠错性能。

贡献： 本文提出了一种简化重启机制（SRM），可降低极化码SCF和动态连续消除翻转（DSCF）译码器的平均执行时间、平均额外执行时间和执行时间方差。该机制的核心思想是利用初始SC遍历后保存的计算结果，有条件地从码字的后半部分重新开始额外译码试次。其纠错性能与原始非SRM译码器完全相同。对于DSCF的多比特翻转版本，平均执行时间、平均额外执行时间和执行时间方差均降低了 $31-58\%$，而额外存储开销为 $2.9-3.9\%$。

概要： 本文其余部分安排如下：第二节介绍极化码，并简要描述连续消除（SC）、SCF和DSCF译码器。第三节介绍SRM，并给出一种算法及其内存需求。第四节讨论硬件约束下译码器的执行时间特性。第五节介绍仿真方法和结果。第六节总结全文。

II. 背景

A. 极化码构造

极化码 $\mathcal{P}(N,\ k)$依赖于由 $\boldsymbol{G}^{\otimes n}$诱导的信道极化，其中 $N=2^n$ 为码长，k 为码维数；$\boldsymbol{G}^{\otimes n}$定义为二进制核 $\boldsymbol{G}=\left[\begin{smallmatrix}1&0\\1&1\end{smallmatrix}\right]$的 n^{th} 克罗内克幂。将 $(N-k)$ 个最不可靠比特称为冻结比特，并设为译码器已知的预定义值，通常全为零。 k 个信息比特置于最可靠的位置，码率为 $R=k/N$。

编码过程表示为 $\boldsymbol{x}=\boldsymbol{u}\boldsymbol{G}^{\otimes n}$，其中 $\boldsymbol{x}$ 和 $\boldsymbol{u}$分别为码字和输入向量。输入向量 $\boldsymbol{u}$ 在预定义位置包含 k 个信息比特以及冻结比特值。我们将输入向量中的冻结比特索引集合记为 $\mathcal{A}^C$，将信息比特索引集合记为 $\mathcal{A}$。比特位置可靠性取决于信道类型和条件。本文考虑加性高斯白噪声（AWGN）信道，采用的构造方法是 Tal 和 Vardy [10]。

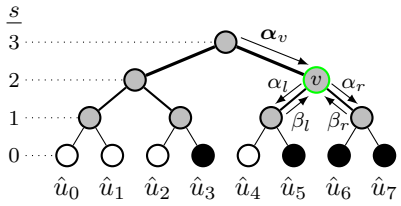

图1： $\mathcal{P}(8，4)$极化码的SC译码树。

B. 连续消除译码

SC译码调度可表示为沿层 $s\in\{0,\ldots,n\}$ 进行的二叉树遍历，从根节点（$s=n$）开始，消息先传递到译码树的左侧（LHS），然后传递到右侧（RHS）。图1展示了一个 $\mathcal{P}(8，4)$ 极化码的译码树。接收到的信道对数似然比（LLR）向量记为 $\boldsymbol{\alpha}_{\text{ch}}=[\alpha_{\text{ch}}(0).....，\ \alpha_{\text{ch}}(N-1)]$,，位于树根处。位于第 s 层、输入向量为 $\boldsymbol{\alpha}_v\in\mathbb{R}^{2^s}$ 的中间节点 v^{th} 将消息 $\boldsymbol{\alpha}_l\in\mathbb{R}^{2^{s-1}}$ 传递到其左侧 l，并将消息 $\boldsymbol{\alpha}_r\in\mathbb{R}^{2^{s-1}}$ 传递到其右侧 r，如下所示：

$$\alpha_l(j)=f\left(\alpha_v(j),\alpha_v\left(j+2^{s-1}\right)\right), \tag{1}$$

$$\alpha_r(j)=g\left(\alpha_v(j),\alpha_v\left(j+2^{s-1}\right),\beta_l(j)\right), \tag{2}$$

其中 $0\le j<2^{s-1}$，且 f： $\mathbb{R}^2\to\mathbb{R}$ 函数是 boxplus 算子，其硬件友好实现为：

$$f(a,d)=\operatorname{sign}(a)\cdot\operatorname{sign}(d)\cdot\min(|a|,|d|) \tag{3}$$

且 g： $\mathbb{R}^2\times\mathbb{F}_2\to\mathbb{R}$ 函数定义为：

$$g(a,d,b)=(1-2b)\cdot a+d. \tag{4}$$

树的叶节点处的对数似然比称为判决对数似然比，记为 $\boldsymbol{\alpha}_{\text{dec}}=[\alpha_{\text{dec}}(0)\ldots,\alpha_{\text{dec}}(N-1)]$。传输向量 $\hat{}\boldsymbol{u}=[\hat{}u_0,\ldots,\hat{}u_{N-1}]$ 中的每个信息比特，都通过对相应的判决对数似然比进行硬判决来估计。译码器已知冻结比特，因此可直接对其进行估计。在图1中，与信息比特对应的判决对数似然比节点以黑色表示，与冻结比特对应的节点以白色表示。比特估计值从树的较低层向较高层传播，并用于计算部分和。节点 v 在第 s层的部分和向量记为 $\boldsymbol{\beta}$（图1），其计算如下：

$$\beta_v(j)=\begin{cases}\beta_l(j)\oplus\beta_r(j) & \text{if}\quad j<2^{s-1},\\ \beta_r(j) & \text{otherwise},\end{cases} \tag{5}$$

其中，运算符 $\oplus$表示逐位异或运算 .

C. 连续消除翻转译码

SCF译码算法在 [5],中被提出，作者观察到，如果能在恢复SC译码之前检测并纠正第一个错误估计的比特，译码器的纠错能力将得到显著提升。为了检测码字译码失败，信息比特与 r位循环冗余校验（CRC）拼接后，通过极化编码器。CRC比特扩展了极化码的信息比特集合$\mathcal{A}$，使码率提高到 $R=(k+r)/N$。

如果在初始SC遍历结束时确认译码失败，则会构建一个比特翻转候选项列表，记为$\boldsymbol{\mathcal{B}}^{\text{flip}}$。将度量值最小的信息比特索引确定为 $\boldsymbol{\alpha}_{\text{dec}}$的绝对值，其中 $\boldsymbol{\alpha}_{\text{dec}}$为度量值。比特翻转索引按照其对应度量值的升序存储在$\boldsymbol{\mathcal{B}}^{\text{flip}}$中。

为了限制SCF解码的译码延迟，定义最大试验次数 $T_{\max}$，其中 $T_{\max}\in\mathbb{N}^+$ 和 $1\le T_{\max}\le(k+r+1)$，包括初始SC遍历。当执行额外试验时，从$\boldsymbol{\mathcal{B}}^{\text{flip}}$ 中选择一个比特进行翻转。令 $T_{\max}=1$ 时，SCF等价于SC译码。如果在 $T_{\max}$ 次试验后循环冗余校验失败，则停止译码并声明失败。需要强调的是， $T_{\max}-1$是比特翻转候选项的总数。

*D.*动态*SC*翻转译码

文献 [7]提出了 DSCF译码，并对原始 SCF 做出了两项重要改进。首先，推导出一种更准确的度量值，用于构建 $\boldsymbol{\mathcal{B}}^{\text{flip}}$ 。其次，提出了一种翻转多个比特的方法，即译码器在每个译码试次中能够翻转多个比特。

通过逐步更新比特翻转候选项集合 $\boldsymbol{\varepsilon}_t=\{i_\lambda\}$，即可实现多个比特的翻转，其中 t 是额外试验的索引。当前集合大小记为 λ、 $1\le\lambda\le\omega$ 和 $i_1\le i_\lambda\le i_\omega$。最大集合大小（即译码阶数） [7], 记为 ω，表示每个试次中比特翻转的最大数量。

对于每个比特翻转集合 $\boldsymbol{\varepsilon}_t$，度量计算和更新均按照以下方式进行：

$$\mathcal{M}_{\text{flip}}(\boldsymbol{\varepsilon}_t)=\sum_{j\in\boldsymbol{\varepsilon}_t}|\alpha_{\text{dec}}(j)|+\Pi(\boldsymbol{\varepsilon}_t), \tag{6}$$

其中， $\Pi(\boldsymbol{\varepsilon}_t)$ 为：

$$\Pi(\boldsymbol{\varepsilon}_t)=\frac{1}{c}\sum_{\substack{j\le i_\lambda\\ j\in\boldsymbol{\mathcal{A}}}}\ln\left(1+e^{-c\cdot|\alpha_{\text{dec}}(j)|}\right), \tag{7}$$

其中 $0<c\le1$。针对不同的码块长度、码率和信道条件，对 c 的取值进行优化。与 SCF译码类似，每个集合 $\boldsymbol{\varepsilon}_t$ 均按度量值 $\mathcal{M}_{\text{flip}}(\boldsymbol{\varepsilon}_t)$ 的升序存储在列表 $\boldsymbol{\mathcal{B}}^{\text{flip}}$ 中。

初始 SC 遍历译码失败后，比特翻转候选项的构建方式与 SCF 类似，但使用度量值 (7)，其中 $\lambda=1$。如果最大集合大小为$\omega=1$，则不再执行额外的度量值更新：比特翻转

候选项按度量值升序排列，并据此选择每次比特翻转，以进行额外的译码试次。如果$\omega > 1$，则应用多次比特翻转，并在每次译码尝试失败后更新译码集合。每次尝试时，都会逐步将新的信息索引插入临时构建的集合，并对该集合执行度量值更新。如果临时集合的度量值超过列表中的最大度量值，则将其丢弃；否则，将其加入列表，同时保持度量值列表有序。集合达到最大大小 ω后，不再继续扩展。

该译码器称为 DSCF-ω，以强调其对参数 ω 的依赖。总共进行了 $T_{\max}$ 次尝试，包含 $T_{\max}-1$ 个比特翻转集合。因此，集合的索引范围为 $1 \le t \le T_{\max}-1$。我们强调，当 $\omega > 1$ 时，$T_{\max} \ge (k+r+1)$ 适用，因为可以构造多个由单个或多个比特翻转集合产生的集合，并将其用作比特翻转候选项。

度量近似：度量值 II （7）包含对数和指数运算。为了使度量值更新更具硬件友好性，文献 [11] 提出了一种近似方法。我们将其记为 II′，定义如下：

$$\Pi'(\varepsilon_t) = \begin{cases} 1.5\,, & \text{if } |\alpha_{\text{dec}}(j)| \le 5.0\,, \\ 0\,, & \text{otherwise.} \end{cases} \tag{8}$$

研究表明，该近似方法造成的编码损失可以忽略不计 [8], [11]。在下文中，度量计算以及 DSCF-ω 译码的度量值更新均采用近似方法 II′。

III. 简化重启机制

本节介绍我们针对 SCF 和 DSCF-ω 译码提出的简化重启机制（SRM）。SRM 通过将第一次 SC 遍历期间获得的必要比特存入额外存储器，有条件地避免冗余计算。本节还将对存储器进行分析。

A. SRM 的描述

SCF 译码（及其变体）中的每次额外试验都从重新执行 SC 计算开始，以估计第一个信息比特，然后一直进行到需要翻转的信息比特所在的位置。然而，我们观察到，比特 $\hat{u}_0$和 $\hat{u}_{N/2}$ 的译码都从译码树的根层 $s = n$ 开始，在此处使用信道对数似然比。后者在当前码字的 SCF 译码过程中保持不变。因此，如果需要翻转的信息比特位于右侧树中，则可以完全避免左侧树的中间对数似然比计算。这些观察结果与信息比特和冻结比特的具体模式无关。右侧树的 SC 计算仍然需要左侧树的估计比特和部分和结果。

由于信道极化，信息比特主要位于译码树的右侧。在图1中，3/4 个信息比特位于右侧。自然，在SCF中，比特翻转通常会发生在右侧。对于每次翻转索引位于右侧的额外试验，我们提出跳过译码树中未改变的左侧，即保留初始 $[\hat{u}_0, \ldots, \hat{u}_{N/2-1}]$，并译码 $[\hat{u}_{N/2}, \ldots, \hat{u}_{N-1}]$。为此，必须能够获取初始SC遍历中的左侧计算结果，即部分和 $\boldsymbol{\beta}_{\text{rest}}$ 和估计比特$\hat{\boldsymbol{u}}_{\text{rest}}$。

算法1 嵌入 SRM 的 SCF解码。

过程 采用 SRM 的 SCF($\boldsymbol{\alpha}_{\text{ch}}$, $\mathcal{A}$, $T_{\max}$)
  $\boldsymbol{\beta}_{\text{rest}} \leftarrow [0, 0, \ldots, 0]$
    $\hat{\boldsymbol{u}}_{\text{rest}} \leftarrow [0, 0, \ldots, 0]$
  $\psi_{\text{rest}} \leftarrow N/2 - 1$
  对于 $t = 1,\ t \le T_{\max},\ t = t+1$，**执行**
    若 $t > 1$，**则** ▷仅在额外试验时
      若 $\mathcal{B}^{\text{flip}}(t) > \psi_{\text{rest}}$ **则** ▷按需启用 SRM
        $\boldsymbol{\beta}(0, \ldots, \psi_{\text{rest}}) \leftarrow \boldsymbol{\beta}_{\text{rest}}$ ▷加载部分和
        $\hat{\boldsymbol{u}}(0, \ldots, \psi_{\text{rest}}) \leftarrow \hat{\boldsymbol{u}}_{\text{rest}}$ ▷加载比特估计
        $srm_act \leftarrow True$ ▷ 设置SRM标志
      else
        $srm_act \leftarrow False$ ▷重置SRM标志 g
      结束 if
    else
      $srm_act \leftarrow False$
    结束 if
    $(\hat{\boldsymbol{u}}, \boldsymbol{\alpha}_{\text{dec}}) \leftarrow$ 连续消除 $(\boldsymbol{\alpha}_{\text{ch}}, \mathcal{A}\ \mathcal{B}_{\text{flip}}(t)\ \boldsymbol{\beta}, srm_act)$
    如果 $CRC(\hat{\boldsymbol{u}}) = failure$ **则**
      如果 $t = 1$ **则** ▷仅在初始试验时
        $\mathcal{B}_{\text{flip}} \leftarrow$ 初始化翻转集合 $(\boldsymbol{\alpha}_{\text{dec}},\ T_{\max} - 1)$
        $\boldsymbol{\beta}_{\text{rest}} \leftarrow \boldsymbol{\beta}(0,\ \ldots\ldots,\ \psi_{\text{rest}})$ ▷存储部分和
        $\hat{\boldsymbol{u}}_{\text{rest}} \leftarrow \hat{\boldsymbol{u}}(0, \ldots, \psi_{\text{rest}})$ ▷存储比特估计
      else
        继续
      结束 if
    else
      跳出
    结束 if
  结束 for
  返回 $\hat{\boldsymbol{u}}$
结束过程

算法1总结了嵌入本文所提出SRM的SCF版本的工作方式。该算法遵循第II-C节所述的原始译码流程。初始SC遍历后，如果循环冗余校验失败，则将比特估计和部分和存入重启列表。随后初始化比特翻转候选项。如果在额外试验期间，识别出位于树的右侧的比特翻转索引，则置位SRM标志，并将存储的列表复制到 $\boldsymbol{\beta}$ 和$\hat{\boldsymbol{u}}$的活动列表中。然后从比特$\hat{u}_{N/2}$处恢复SC译码。

采用 $\omega = 1$的DSCF-ω 译码所提出的SRM遵循与算法1相同的译码调度，但使用了会产生不同比特翻转列表的度量函数。对于每个比特翻转集合 $\varepsilon_t = \{i_\lambda\}$，只有第一个比特 i_1 的位置决定SRM的激活条件。请注意，比特翻转索引会逐步添加到任意集合 ε_t 中，使得 $i_1 < \ldots < i_\lambda$。因此，如果 i_1位于码字的后半部分，则该集合中的其余比特也位于该处。

SRM 可以集成到其他类型的基于SCF的译码器中，例如那些采用 [12], [13]的译码器。将该机制适配到这些译码器只需很少的工作，而且只能改善其性能。本文重点研究 SCF 和 DSCF-ω 译码器，以展示该机制的功能。

B. 存储结构

图2展示了一种内存架构，其设计灵感来自 [14] 和[15]，适用于集成了所提出的SRM的SCF译码器。

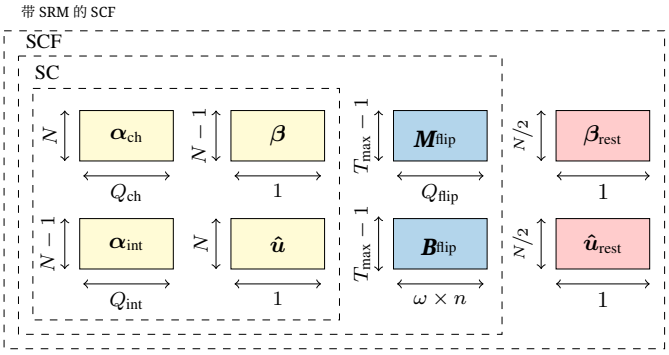

图2：嵌入 SRM 的 SCF 的内存架构。

标签表示存储器的内容。深度表示数据向量的长度，而宽度表示每个条目的比特数。如图2所示，LLR值和比特翻转度量采用不同的量化方式。信道对数似然比使用 Q_{ch} 比特，中间对数似然比使用 Q_{int}比特，比特翻转度量使用 Q_{flip} 比特。比特翻转候选列表需要$(T_{\max}-1)\times\omega\times n$ 比特，其中 n为比特翻转索引的二进制表示长度。其余存储块均为宽度为1比特的二进制向量。总计，$\boldsymbol{\alpha}_{\text{ch}}$ 需要 $N\times Q_{\text{ch}}$ 比特，$\boldsymbol{\alpha}_{\text{int}}$需要$(N-1)\times Q_{\text{int}}$，而$\boldsymbol{\mathcal{M}}^{\text{flip}}$需要$(T_{\max}-1)\times Q_{\text{flip}}$比特。请注意，SRM仅需要在初始SC试次中从LHS树计算得到的部分和与比特估计。因此，SRM的存储开销为 N 比特。

IV. 硬件约束下的执行时间模型

为了估算能够反映架构设计的时延，我们实现了一个处理单元数量有限的模型，记为 P。该方法基于半并行SC译码器的架构 [14]。

SC译码树的节点（图1）使用数量有限的处理单元并行计算函数 (1)–(2)。单个LLR计算的时延视为一个时钟周期（CC）。对于每个节点，部分和向量通过函数 (5) 在一个CC内计算。考虑到逐位异或运算的简便性，这种部分和处理方法是可行的。

单次 SC遍历的时延（以 CC 计）记为 $\mathcal{L}_{\text{SC}}$，其由以下公式给出 [14]:

$$\begin{aligned}\mathcal{L}_{\text{SC}} &= \mathcal{L}_{\alpha}+\mathcal{L}_{\beta}\,,\\ &= \left(2N+\frac{N}{P}\cdot\log_2\left(\frac{N}{4P}\right)\right)+(N-n-1)\,,\end{aligned}\tag{9}$$

其中，$n=\log_2(N)$，$\mathcal{L}_{\alpha}$ 是对数似然比计算的时延，$\mathcal{L}_{\beta}$ 是部分和计算的时延。

SCF译码器译出一个码字的执行时间等于SCpass时延与所需译码试次的乘积。所需的尝试次数记为 t_{req}，总执行时间计算如下:

$$l_{\text{SCF}} = t_{\text{req}}\cdot\mathcal{L}_{\text{SC}}\,,\tag{10}$$

其中 $1\le t_{\text{req}}\le T_{\max}$。如果 $t_{\text{req}}=T_{\max}$，l_{SCF} 表示最坏情况下的执行时间，因此它就是译码时延。

由于执行时间具有可变性，以下特性值得关注：平均执行时间、平均额外执行时间和执行时间方差。这些度量值通过仿真以实验方式获得。平均执行时间 $\mathcal{L}_{\text{SCF}}$ 估计如下:

$$\mathcal{L}_{\text{SCF}} = \frac{1}{S}\sum_{s=1}^{S} l_{\text{SCF}}\,,\tag{11}$$

其中，S 是仿真的码字总数。平均额外执行时间 $\mathcal{L}'_{\text{SCF}}$ 估计如下:

$$\mathcal{L}'_{\text{SCF}} = \frac{1}{S'}\sum_{s=1}^{S'}\left(l_{\text{SCF}}-\mathcal{L}_{\text{SC}}\right)\,,\tag{12}$$

其中，$S'\le S$ 表示需要多于一次SC遍历才能通过SCF解码完成译码的码字数量。执行时间方差 $\mathcal{V}_l$ 估计如下:

$$\mathcal{V}_l = \frac{1}{S-1}\sum_{s=1}^{S}\left(l_{\text{SCF}}-\mathcal{L}_{\text{SCF}}\right)^2\,.\tag{13}$$

对于DSCF-ω 译码器，由于与SC译码相比，这些操作对执行时间的影响可以忽略不计，因此忽略与比特翻转集合更新相关的时延。因此，式（11）–（13）用于表征SCF和DSCF-ω 译码器的执行时间特性。

V. 仿真结果

A. 方法

建立了一套仿真设置，以分析我们所提出机制对 SCF 和 DSCF-ω 译码的影响。对随机数据块采用长度为 $N=1024$、三种不同码率 $R\in\{1/8,1/4,1/2\}$的极化码，以及长度为 $N=512$、码率为 $R=1/8$的极化码进行编码。采用具有多项式$z^{16}+z^{15}+z^2+1$的 $r=16$ 比特循环冗余校验。对于长度$N=1024$ 、码率分别为 $1/8$、$1/4$和 $1/2$的极化码，其设计 E_b/N_0 分别为 1.25 分贝、1.25 分贝和 2.5 分贝。对于长度为 $N=512$ 、码率为 $1/8$ 的极化码，其设计 E_b/N_0 为 1.25 分贝。在加性高斯白噪声信道上采用二进制相移键控调制。仿真至少运行了 $S=10^5$ 个随机码字，直到发现 10^3 个错误帧。考察了具有 $\omega\in\{1,2,3\}$ 的 DSCF-ω 译码器。

所有仿真中的处理单元数量均限制为 $P=64$。DSCF-ω 译码器使用便于硬件实现的 (8) 函数进行度量计算。SCF 的最大试验次数 $T_{\max}$ 设为 13，而 DSCF-ω的最大试验次数设为 $T_{\max}\in\{8,51,301\}$（针对 $\omega\in\{1,2,3\}$）。选择 $T_{\max}$ 的取值，是为了在目标帧错误率（FER）下实现接近理想辅助解码器 [7] 的纠错性能。

我们从纠错性能、执行时间特性和存储器需求方面，对采用和未采用 SRM 的 $\mathcal{P}$(1024，128) 极化码译码器进行比较。我们重点介绍执行时间

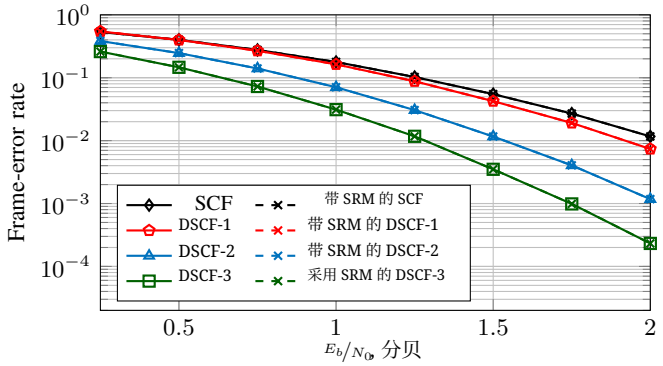

图 3：对于 $\mathcal{P}(1024, 128)$ 极化码，带有 $\omega \in \{1, 2, 3\}$ 和不带 SRM 的 SCF 和 DSCF-ω译码器的纠错性能，其中 $r = 16$。

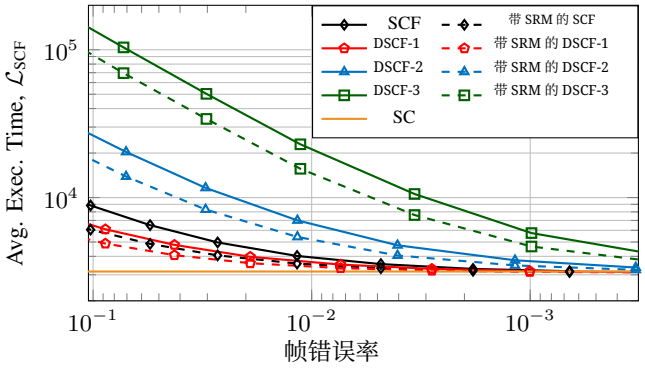

图 4：对于 $\mathcal{P}(1024, 128)$ 极化码，带有和不带 SRM 的 SCF 和 DSCF-ω 译码器的平均执行时间为 $\omega \in \{1, 2, 3\}$ $r = 16$。普通 SC译码器用作参考。

特性，并展示在目标帧错误率为 10^{-2} 时，针对不同码长和码率的极化码采用 SRM 的情况。

B. 纠错性能

图 3 展示了 $\mathcal{P}(1024, 128)$ 极化码下 SCF 和 DSCF-ω 译码器以帧错误率 FER 衡量的纠错性能。带有和不带 SRM 的译码器分别以虚线和实线表示。SCF译码器以黑色表示，DSCF-2 以浅蓝色表示，DSCF-3 以浅绿色表示，并使用不同的标记。

从图 3 可以看出，SRM 不会改变纠错性能。这是意料之中的，也符合第三节-A节中所述该机制的定义。结果还与 [7], 一致，即表明 DSCF-ω 优于标准 SCF。DSCF-3译码器提供了最佳性能，因此有必要降低该算法的执行时间特性。

C. 执行时间特性

图 4 展示了 $\mathcal{P}(1024, 128)$ 极化码下 SCF 和 DSCF-ω 译码器的平均执行时间。图中分别用虚线和实线表示采用和未采用 SRM 的译码器，并提供 SC 译码的时延作为参考。从图中可以看出，在整个帧错误率范围内，采用 SRM 都能为 DSCF-2 和 DSCF-3 译码器带来更大的收益。我们认为这是因为多比特翻转译码器平均执行的额外试验次数较多。我们还观察到，与其他译码器相比，采用 SRM 的 DSCF-1 译码器所实现的缩减幅度最小。这可以用其额外译码尝试次数较少、译码时延较低来解释（在其他译码器中最小的 $T_{\max}$ ）。图 4 还显示，在较低帧错误率下，采用和未采用 SRM 的所有译码器的平均执行时间都趋近于 SC 译码器的时延。

各译码器在不同码率下对$N = 1024$ 极化码的执行时间特性改善情况汇总于表I。表II展示了 $R = 1/8$ 中 $N = 512$的结果，这些结果体现了 SRM 带来的较大增益。特性指标的符号说明见第IV节。差值以 Δ 表示，并以百分比给出。每个译码器的 E_b/N_0 点均标出了对应目标帧错误率为 10^{-2}的点。表中的结果表明，对于 $R = 1/8$下的 $N = 1024$极化码，应用 SRM 带来的增益最大。与原始 SCF译码器相比，嵌入 SRM 的 SCF 的平均执行时间、平均额外执行时间和执行时间方差分别降低了 11.17%、48.30% 和 73.50%。嵌入 SRM 的 DSCF-1译码器分别降低了 7.57%、43.57% 和 67.28%。值得注意的是，DSCF-3 译码器的纠错性能最强；应用 SRM 后，其执行时间特性分别降低了 31.70%、37.08% 和 57.28%。观察 $N = 1024$下较高码率的极化码结果可见，尽管执行时间特性的降幅较小，但总体趋势保持不变。比较 $R = 1/8$下 $N = 1024$和 $N = 512$ 极化码的结果可见，各项特性的降幅几乎相同。对 $R = 1/2$ 下的 $N = 1024$ 极化码所用的 DSCF-3译码器应用 SRM 后，执行时间特性分别降低了 7.33%、9.03% 和 12.20%。

D. 内存估算

存储量按照第III-B节所述方法计算，其中使用了与 [8] 相同的量化方案。因此，信道对数似然比、中间对数似然比和比特翻转度量分别量化为 $Q_{\text{ch}} = 6$、 $Q_{\text{int}} = 7$ 和 $Q_{\text{flip}} = 7$比特。在这些比特中， Q_{ch} 和 Q_{int}各使用2比特表示小数部分，而 Q_{flip}使用3比特。表III列出了所有所考虑译码器的内存估算及其存储开销百分比。结果针对不同码长的极化码给出，而码率不会影响存储量。从表III可以看出，所提出的SRM带来的存储开销为2.86%至6.62%。与其他译码器相比，将SRM嵌入DSCF-3译码器所产生的存储开销最小，因为DSCF-3需要更多存储空间来保存比特翻转候选项列表$\mathcal{B}^{\text{flip}}$ 及相应的比特翻转度量列表$\mathcal{M}^{\text{flip}}$。

表I：对于目标帧错误率 10^{-2}，在各种码率下，使用 SRM 对码长为$N = 1024$ 的极化码的 SCF 和 DSCF-ω 译码器的执行时间特性进行缩减。

		$\mathcal{P}(1024,\ 128),\ r = 16$				$\mathcal{P}(1024,256),\ r = 16$				$\mathcal{P}(1024,\ 512),\ r = 16$			
	$T_{\max}$	E_b/N_0	$\Delta\mathcal{L}_{\text{SCF}}$	$\Delta\mathcal{L}'_{\text{SCF}}$	$\Delta\mathcal{V}_l$	E_b/N_0	$\Delta\mathcal{L}_{\text{SCF}}$	$\Delta\mathcal{L}'_{\text{SCF}}$	$\Delta\mathcal{V}_l$	E_b/N_0	$\Delta\mathcal{L}_{\text{SCF}}$	$\Delta\mathcal{L}'_{\text{SCF}}$	$\Delta\mathcal{V}_l$
SCF	13	2.0	11.17	48.30	73.50	1.875	9.00	44.68	69.15	2.375	6.54	37.44	59.71
DSCF-1	8	1.875	7.57	43.57	67.28	1.75	5.01	33.33	50.87	2.25	2.95	22.04	32.35
DSCF-2	51	1.5	22.73	40.71	63.26	1.44	14.74	27.60	41.74	2.00	6.96	13.83	19.00
DSCF-3	301	1.25	31.70	37.08	57.28	1.25	19.09	22.57	34.78	1.875	7.33	9.03	12.20
		以分贝为单位	Δ以百分比表示			以分贝为单位	Δ以百分比表示			以分贝为单位	Δ占 %		

表I：对于 $R = 1/8$ 、目标帧错误率为 10^{-2}的$N = 512$ 极化码，使用 SRM 后 SCF 和 DSCF-ω 译码器的执行时间特性降低情况。

		$\mathcal{P}(512,\ 64),\ r = 16$			
	$T_{\max}$	E_b/N_0	$\Delta\mathcal{L}_{\text{SCF}}$	$\Delta\mathcal{L}'_{\text{SCF}}$	$\Delta\mathcal{V}_l$
SCF	13	2.625	10.36	47.49	72.27
DSCF-1	8	2.625	6.03	43.93	67.94
DSCF-2	51	2.0	24.01	40.52	63.08
DSCF-3	301	1.75	32.33	37.59	58.67
		以分贝为单位	Δ以 % 为单位		

表III：SRM 对 $N \in \{1024, 512\}$ 的极化码译码器造成的内存估算和存储开销。

极化码（ $N = 1024$）

	$T_{\max}$	不使用 SRM，比特	使用 SRM，比特	内存增量，%
SCF	13	15556	16580	6.58
DSCF-1	8	15471	16495	6.62
DSCF-2	51	16702	17726	6.13
DSCF-3	301	26452	27476	3.87

带有 $N = 512$ 的极化码

	$T_{\max}$	无 SRM ，bits	含 SRM ，bits	内存增量，%
SCF	13	7864	8376	6.51
DSCF-1	8	7784	8296	6.58
DSCF-2	51	8922	9434	5.74
DSCF-3	301	17872	18384	2.86

VI. 结论

本文提出了简化重启机制（SRM）。该机制通过在翻转索引落入译码树右侧时，使除初始尝试外的其他尝试从译码过程的中间阶段开始，从而降低SCF和DSCF-ω 译码器的执行时间特性。在访问完译码树左侧后，该机制需要存储初始SC遍历中的少量结果。我们展示了在DSCF-ω 译码器中使用该机制所需进行的少量修改。所提出的机制可以集成到其他基于SCF的译码算法中，不会影响纠错性能，并且适用于任意码长和码率。对于码长为1024比特的极化码DSCF-3译码器，结果显示，平均执行时间、平均额外执行时间和执行时间方差分别减少了31%、37%和57%，代价是增加3.9%的存储开销。

致谢

作者感谢 Tannaz Kalatian 对该主题的初步研究。本研究得到 NSERC探索资助（编号 #651824）的支持。

参考文献

[1] E. Arıkan，“信道极化：一种为对称二进制输入无记忆信道构造容量逼近码的方法”，*IEEE*信息论汇刊，第7期，2009年7月。[2] I. Tal 和 A. Vardy，“ 极化码的列表译码 ”，*IEEE* 信息论汇刊，2015 年 3 月。[3] 3GPP，“NR；复用与信道编码”，技术报告 TS 38.212，2018年1月，Release 16.5。 [在线]。可访问：http://www.3gpp.org/DynaReport/38-series.htm [4] F. Ercan、C. Condo 等，“关于 5G 极化码列表译码器的纠错性能与实现”，载于*Allerton*通信、控制与计算会议（*Allerton*）论文集，2017年10月。

[5] O. Afisiadis、A. Balatsoukas-Stimming 和 A. Burg，“一种低复杂度改进型极化码连续消除译码器”，载于阿西洛马会议：信号、系统与计算（*ACSSC*），2014年11月。[6] P. Giard、A. Balatsoukas-Stimming 等，“POLARBEAR：用于极化码译码的28纳米FD-SOI ASIC”，*IEEE*新兴电路与系统专题期刊，第7卷，第4期，2017年12月。[7] L. Chandesris、V. Savin 和 D. Declercq，“极化码的动态SCFlip译码”，*IEEE*通信汇刊，第6期，2018年6月。[8] F. Ercan、T. Tonnellier 等，“实用动态SC-Flip极化码译码器：算法与实现”，*IEEE*信号处理汇刊，2020年9月。[9] I. Sagitov 和 P. Giard，“用于极化码DSCF译码的提前停止机制”，载于*IEEE*信号处理系统国际研讨会（*SiPS*），2020年9月。[10] I. Tal 和 A. Vardy，“如何构造极化码”，*IEEE*信息论汇刊，第10期，2013年10月。[11] F. Ercan、T. Tonnellier 等，“简化的动态SC-flip极化码译码”，载于*IEEE*声学、语音与信号处理国际会议（*ICASSP*），2020年5月。[12] P. Giard 和 A. Burg，“极化码的快速SSC-Flip译码”，载于*IEEE*无线通信与网络会议（*WCNC*），2018年4月。[13] F. Ercan、C. Condo 和 S. Hashemi，“极化码的分区连续消除翻转译码”，载于*IEEE*国际通信会议（*ICC*），2018年5月。[14] C. Leroux、A. Raymond 等，“一种用于极化码的半并行连续消除译码器”，*IEEE*信号处理汇刊，2012年10月。[15] S. Hashemi、C. Condo 和 W. Gross，“用于极化码的快速灵活连续消除列表译码器”，*IEEE*信号处理汇刊，2017年11月。