奇偶校验极化码的快速串行抵消列表译码算法

李俊毅,邢莉娟*,李 卓
(西安电子科技大学通信工程学院,陕西西安 710071)

摘 要: 为了缩短奇偶校验极化码串行抵消列表(Parity-Check Successive Cancellation List,PC-SCL)译码算法的时延,本文提出了快速奇偶校验串行抵消列表(Fast Parity-Check Successive Cancellation List,Fast-PC-SCL)译码算法.该算法首先分析并研究了奇偶校验极化码(Parity-Check Polar,PC-Polar)中存在的2类特殊节点——奇偶校验重复(PC-REPetition,PC-REP)类节点和单奇偶校验(PC-Single-Parity-Check,PC-SPC)类节点,并通过理论证明了PC-REP类节点具有码字序列周期性重复、PC-SPC类节点具有码字和为特定值的性质.其次根据上述性质,给出了这2类节点的码字列表估计方法,使得包含这2类节点的极化码在译码时可以通过并行执行来大幅度缩短译码的时间延迟.最后结合这2类节点的码字列表估计方法,提出了Fast-PC-SCL译码算法.该算法可以在不完全遍历串行抵消(Successive Cancellation,SC)译码树的情况下进行译码,同时充分保留PC比特校验的效果.与PC-SCL译码算法相比,在不损失性能的前提下,该算法显著缩短了译码时延.实验数据表明,最多可缩短55.13%的时间延迟.

关键词: 极化码;串行抵消列表;奇偶校验;译码时延;特殊节点
基金项目: 国家自然科学基金(No.61372072);111工程基金(No.B08038)
中图分类号: TN911.22 **文献标识码:** A **文章编号:** 0372-2112(2025)07-2210-12
电子学报URL:http://www.ejournal.org.cn **DOI:**10.12263/DZXB.20240496

The Fast Successive Cancellation List Decoding Algorithm for Parity-Check Polar Codes

LI Jun-yi, XING Li-juan*, LI Zhuo
(*School of Telecommunications Engineering*, *Xidian University*, *Xi'an*, *Shaanxi* 710071, *China*)

Abstract: To reduce the latency of the parity-check successive cancellation list (PC-SCL) decoding algorithm for parity-check polar codes, a fast parity-check successive cancellation list (Fast-PC-SCL) decoding algorithm is proposed. Firstly, the algorithm analyzes and studies two types of special nodes in parity-check polar (PC-Polar) codes: PC-Repetition (PC-REP) nodes and PC-single parity-check (PC-SPC) nodes, and theoretically proves that PC-REP nodes exhibit cyclic repetition of codeword sequences, while PC-SPC nodes have the property of the sum of codewords equals a specific value. Secondly, based on these properties, codeword list estimation methods are given for these two types of nodes. This enables the decoding of these nodes to be executed in parallel, significantly reducing decoding latency. Finally, by combining the codeword list estimation methods for these two types of nodes, the Fast-PC-SCL decoding algorithm is presented. This algorithm can decode without completely traversing the successive cancellation (SC) decoding tree, while fully retaining the effect of PC bit checks. Compared to the PC-SCL algorithm, it significantly reduces decoding latency without sacrificing performance. Experimental data shows that it can reduce latency by up to 55.13%.

Key words: polar codes; successive cancellation list; parity-check; decoding latency; special nodes
Foundation Item(s): National Natural Science Foundation of China (No.61372072); Project 111 Fund (No.B08038)

1 引言

极化码[1]于2009年由土耳其的Arikan提出,它是一种基于信道极化理论的信道编码方法,也是唯一一种可以通过数学证明达到信道容量的编码方法.极化码由于其优异的性能已经成功入选5G增强移动宽带(enhanced Mobile BroadBand,eMBB)场景标准[2].Arikan在最初提出

收稿日期:2024-05-29;录用日期:2024-11-14;责任编辑:孙瑶
*通讯作者:邢莉娟

极化码这一概念时就已提出串行抵消(Successive Cancellation,SC)译码算法[1],该算法虽然复杂度较低,但是其译码性能较差.2011年,Tal等人[3]提出了串行抵消列表(Successive Cancellation List,SCL)译码算法.该算法在译码过程中考虑对多条路径进行保留,其性能逼近最大似然译码算法,得到了较为广泛的应用.无论是SC译码算法还是SCL译码算法,由于它们串行译码的天然特性,都存在较长的译码时延.因此,如何缩短极化码的译码时延成为极化码能否广泛应用的关键.

2011年,针对如何缩短极化码译码算法时延的问题,Alamdar-Yazdi等人[4]从极化码本身特性出发,提出了一种简化的SC(Simplified Successive Cancellation,SSC)译码算法.该算法讨论了Rate-0和Rate-1这2种特殊节点(所谓节点,即SC译码树中的中间节点),以及在译码时无须完全遍历码树,而是选择在这2种特殊节点上终止译码并直接进行多比特判决.这一发现为广大研究者提供了思路,各种关于极化码的特殊节点研究也逐渐展开.2013年,Sarkis等人[5]定义了最大似然(Maximum-Likelihood,ML)节点,提出采用穷举的办法对最大似然节点进行译码以避免串行译码.2014年,又提出了单奇偶校验(Single-Parity-Check,SPC)节点和重复(REPetition,REP)节点[6],大幅度缩短了译码的时延.但上述研究主要针对SC译码算法而非SCL译码算法进行简化,其主要原因是SCL译码算法中节点的度量值难以计算.直到2016年,Hashemi等人[7]总结了SCL译码算法中各种节点下路径度量值的计算方法;2017年,给出了Rate-1节点的快速SCL译码算法[8],同年,给出了Rate-0、Rate-1、SPC、REP节点的综合快速SCL译码算法[9].该算法在能达到同等SCL译码算法性能的同时,缩短了译码的时延.2017年,Hanif等人[10]又从现有的节点出发,提出了5种普适性更强的节点.这些节点分别具有固定的冻结比特和信息比特分布,使用者利用分布规律,可以极大程度提高译码的并行性.2019年,Ardakani等人[11]针对文献[10]中提出的5种节点,提出了对应的快速SCL译码算法.2021年,Condo等人[12]定义了一种全新的序列重复(Sequence Repetition,SR)节点.该节点通常位于极化码码树的高层并包含现有的大多数节点,形式上更为简洁.2022年,Zheng等人[13]描述了SR节点的快速译码算法.该算法对缩短极化码的译码时延具有明显作用.在众多研究者的努力下,对于极化码的快速译码算法的研究目前已经取得了显著的成果.

极化码的码字结构中只包含信息比特和冻结比特,这使译码端缺乏路径校验信息,从而对极化码的性能产生一定影响.针对该问题,我们可以通过在码字中添加辅助码字如循环冗余校验(Cyclic Redundancy Check,CRC)码字、奇偶校验(Parity Check,PC)码字等加以改善.其中,PC码字可以通过影响极化码的权重谱来改善其性能,目前在5G标准场景下已有广泛应用[14,15].虽然目前对于极化码的快速译码算法已有广泛研究,但是由于奇偶校验极化码(Parity-Check Polar,PC-Polar)中存在码字制约关系,极化码的快速译码算法对于PC-Polar码不甚适用.因此对于PC-Polar码的快速译码算法的研究进展缓慢,目前只有Zhou等人[16]给出了PC-Polar码的一些节点的快速SC译码算法.本文研究了PC-Polar码中存在的2类特殊节点,并提出了快速奇偶校验串行抵消列表(Fast Parity-Check Successive Cancellation List,Fast-PC-SCL)译码算法.该算法在译码进行到特殊节点时,采用并行的策略给出最可能路径的列表,同时充分保留了PC校验比特的效果.仿真实验证明,在不影响译码性能的前提下,该算法较大程度地缩短了译码的时延.

2　极化码的编译码原理

2.1　编码原理

2.1.1　极化码的编码

极化码是一种基于信道极化理论的线性分组码,通过信道极化可以选出信息位和冻结位.信息位的索引使用A来表示,冻结位的索引使用A^C来表示.在信息位集合放置信息比特,在冻结位统一传输0,从而构成信息序列$u_0^{N-1}=(u_0,u_1,u_2,\cdots,u_{N-1})$.对于码长为$N=2^n$、信息位长度为$K$的极化码,其编码过程如下:

$$x_0^{N-1}=u_0^{N-1}\boldsymbol{G}_N \tag{1}$$

其中,x_0^{N-1}表示生成的码字,$\boldsymbol{G}_N$表示生成矩阵.

$$\boldsymbol{G}_N=\boldsymbol{B}_N\boldsymbol{F}^{\otimes n} \tag{2}$$

其中,$\boldsymbol{B}_N$为比特置换矩阵,$\boldsymbol{F}^{\otimes n}$表示$\boldsymbol{F}$的$n$次克罗内克幂,$\boldsymbol{F}=\begin{pmatrix}1 & 0\\ 1 & 1\end{pmatrix}$称为极化核.

2.1.2　PC-Polar码的编码

PC-Polar码的编码和极化码的编码类似,区别在于增加了PC位置的选择和PC比特的生成.PC位置的选择和PC比特的生成存在多种方法,下面给出应用较广的算法[17].本文后续研究中的PC-Polar码的编码也是基于此算法开展.

步骤1:预先计算PC位置的数量F_p.

$$F_p=\log_2 N\times\left(\alpha-\left|\alpha\times(K/N)-1/2\right|^2\right) \tag{3}$$

其中,α的值可选,本文及下述仿真中的α值取1.

步骤2:从N个比特信道内选取$K+F_p$个可靠信道,计算这些信道对应的行重(即生成矩阵中对应行1的个数),并求得最小行重$w_{\min}$和具有最小行重的子信道个数n_1.

步骤3：若 $F_p \leqslant n_1$，则在 $K+F_p$ 集合内选取行重为 $w_{\min}$ 的最可靠的 F_p 个信道作为PC位置；反之则在 $K+F_p$ 集合中选取 $(F_p-n_1)\times 3/4$ 个重量为 $2w_{\min}$ 的最可靠信道作为PC位置，从 $K+F_p$ 集合中选取 K 个信息位，之后将其他所有位置作为冻结位.

步骤4：PC比特通过PC校验方程进行计算，对于放置在 i 位置的PC比特，其具体值由 i 之前与 i 相距为 d 的整数倍的信息比特取模2得到，本文仿真中采用的 $d=5$.

冻结比特和信息比特放置规则与极化码相同，之后按照同样的步骤进行编码.

2.2 译码原理

2.2.1 SC译码算法

Arikan在文献[1]中给出了极化码的SC译码算法，该算法的核心是在接收到 y_0^{N-1} 之后得到 u_0^{N-1} 的估计值 $\hat{u}_0^{N-1}$. 首先将 x_0^{N-1} 经过信道传输得到 y_0^{N-1}，然后根据公式进行递归计算，可以获得完整的似然比序列 LR_N，最后根据 LR_N 进行判决从而得到译码结果. 在判决过程中，若 $i\in A^C$，则 $u_i=0$；若 $i\in A$，则进行如下判决：

$$\hat{u}_i=\begin{cases}0, & \mathrm{LR}_N^{(i)}\left(y_1^N,\hat{u}_1^{i-1}\right)\geqslant 1\\ 1, & \text{otherwise}\end{cases} \tag{4}$$

而在实际应用中，通常使用对数似然比(Log-Likelihood Ratio，LLR)计算，LLR=ln(LR)，一般记作 α，对应 α 的判决公式如下：

$$h(\alpha)=\begin{cases}0, & \alpha\geqslant 0\\ 1, & \text{otherwise}\end{cases} \tag{5}$$

2.2.2 SCL译码算法

SC译码算法作为极化码的一种传统译码方法，其复杂度较低，但是由上文可以看出，由于SC译码算法是串行译码，因此如果前面比特判决错误，将会对后续的码字产生影响，即会出现错误传播现象. 为了避免这种情况的发生，Tal等人[3]提出了SCL译码算法. 该算法在SC译码算法的基础上进行改进，同时保留至多 L 条路径. 当第 i 个位置为冻结位的时候，各个路径和SC译码算法相同；反之，当其为信息位的时候，所有存活路径会同时保留 $u_i=0$ 和 $u_i=1$ 这2条路径，同时进行度量值的更新：

$$\mathrm{PM}_i^l=\sum_{k=1}^{i}\ln\left(1+\mathrm{e}^{-(1-2u_k^l)\alpha_k^l}\right) \tag{6}$$

其中，u_k^l 为第 l 条路径的第 k 个信息比特，α_k^l 为第 l 条路径的第 k 个比特对应的似然比. 文献[7]证明了使用 u_k^l 和 x_k^l 计算PM值可以取得相同的效果. x_k^l 为第 l 条路径的第 k 个码字比特，而采用 x_k^l 计算则为快速译码算法提供了便利，即

$$\mathrm{PM}_i^l=\sum_{k=1}^{i}\ln\left(1+\mathrm{e}^{-(1-2x_k^l)\alpha_k^l}\right) \tag{7}$$

在实际运算中可以采用 $\ln(1+\mathrm{e}^{\alpha})-\ln(1+\mathrm{e}^{-\alpha})=\alpha$ 进行简化，即

$$\begin{aligned}\ln(1+\mathrm{e}^{-(1-2(x\oplus 1))\alpha})&=\ln\left(1+\mathrm{e}^{(1-2x)\alpha}\right)\\&=\ln\left(1+\mathrm{e}^{-(1-2x)\alpha}\right)+(1-2x)\alpha\\&=\ln\left(1+\mathrm{e}^{-(1-2x)\alpha}\right)+|\alpha|\end{aligned} \tag{8}$$

由于 $x=h(\alpha)$，$x\oplus 1$ 表示对判决结果取反，式(8)表示若符合硬判，则度量值为 $\ln(1+\mathrm{e}^{-(1-2x)\alpha})$，否则会加上 $|\alpha|$ 的惩罚值. 本文后续对于度量值计算方法的研究也基于文献[7]开展.

当保存的路径数量大于 L 条时，译码器根据度量值的排序进行剪枝，即删除掉度量值较大的路径，只保留 L 条路径，重复此过程直到译出所有的码字.

2.2.3 PC-SCL译码算法

PC-SCL译码算法是基于SCL译码算法产生的，其路径分裂方式和度量值计算方式均与SCL算法相同. 文献[14]给出了PC-SCL译码算法，以下为具体描述：由于PC-Polar码的编码方式较为特殊，校验比特 p_j 对应的校验方程中的信息比特均在 p_j 之前，因此在PC-SCL译码算法中将PC位置当作动态冻结位来处理，其值由本身的校验方程所决定. 为了更方便地描述PC-SCL算法的译码过程，通常使用码树来表示整个过程，其译码二叉树图如图1所示：第0层的 α_0^{N-1} 是从信道中接收到的原始的LLR序列，其LLR序列优先向左节点传播，经过 f 运算得到 $(\alpha_0^l,\alpha_1^l,\cdots,\alpha_{N/2-1}^l)$，如式(9)所示：

$$\begin{aligned}\alpha_i^l&=f(\alpha_i^{l-1},\alpha_{i+N/2}^{l-1})\\&=\mathrm{sign}(\alpha_i^{l-1})\mathrm{sign}(\alpha_{i+N/2}^{l-1})\min\left(|\alpha_i^{l-1}|,|\alpha_{i+N/2}^{l-1}|\right)\end{aligned} \tag{9}$$

此时的左节点可以看作一个子极化码，仍然执行 f 运算的操作，重复这个过程，直到第 n 层. 若第 n 层的对应位置是冻结位，则判决为0；若为信息位，则同时保留 $u_i=0$ 和 $u_i=1$ 这2条路径，并计算度量值；若为PC位置，则按照校验方程进行PC比特的计算，将该位置直接判为计算出的PC值. 然后对每一条路径的右节点再进行 g 运算，并进行 h 判决.

$$\alpha_i^l=g(\alpha_i^{l-1},\alpha_{i+N/2}^{l-1},x_i^l)=(1-2x_i^l)\alpha_i^{l-1}+\alpha_{i+N/2}^{l-1} \tag{10}$$

第 $n-1$ 层对应第1个节点 $[x_0,x_1]=[u_0,u_1]F$，其中 u_0,u_1 为第 n 层计算出的比特，将第1层获得的码字比特 x_0,x_1 带入 g 运算，得到第 $n-1$ 层第2个节点对应的LLR值，重复上述过程直到码字的末尾. 最终从码字列表中选择度量值最小的一个，u_0^{N-1} 即为最终的译码结果.

本文提出的Fast-PC-SCL译码算法核心思想就是在第 j 层节点时，若该节点属于本文中提出的2类特殊节点，则不进行递归运算，直接给出该节点的估计码字

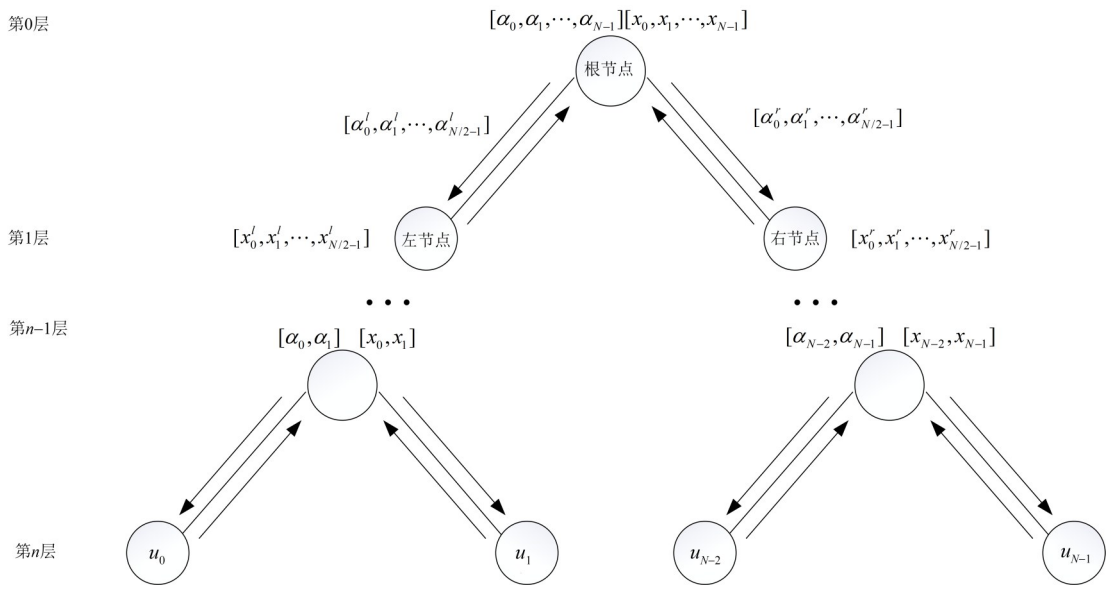


图1　PC-SCL算法的译码二叉树图

列表,并给出每一条路径的度量值;若不属于特殊节点,则按照正常的PC-SCL译码算法进行,最终输出度量值最小的码字序列作为译码结果. 值得注意的是,当使用PC-SCL译码算法时,算法最终给出的结果是信息估值序列 $\hat{u}_0^{N-1}$,即未进行极化码编码的序列. 在接收端,只需要按照极化码构造时的信息位和冻结位分布进行挑选即可恢复原始的信息序列 u_0^{K-1}. 但是若在快速译码中给出的是码字估值序列 $\hat{x}_0^{N-1}$,则需要进行额外的运算来得到最终的信息估值序列.

由极化码编码可知 $x_0^{N-1}=u_0^{N-1}\boldsymbol{G}_N=u_0^{N-1}\boldsymbol{B}_N\boldsymbol{F}^{\otimes n}$, $n=\log_2 N$, $\boldsymbol{G}_N$ 满足:

$$\boldsymbol{G}_N=\boldsymbol{B}_N\boldsymbol{F}^{\otimes n}=\boldsymbol{F}^{\otimes n}\boldsymbol{B}_N \tag{11}$$

式(11)中,$\boldsymbol{B}_N$ 和 $\boldsymbol{F}^{\otimes n}$ 的相对顺序不会对编码结果产生影响,$\boldsymbol{B}_N$ 在其中只起到顺序置换的作用,因此在研究极化码的快速译码算法时可以不考虑 $\boldsymbol{B}_N$ 的作用,只考虑 $\boldsymbol{F}^{\otimes n}$ 的作用. 同时由于 $\boldsymbol{F}^{\otimes n}$ 为正交对称矩阵,在得到码字估值序列后,可以通过式(12)得到原始的信息序列:

$$u_0^{N-1}=x_0^{N-1}\boldsymbol{G}_N=x_0^{N-1}\boldsymbol{F}^{\otimes n} \tag{12}$$

下面分2种情况进行讨论. 若极化码进行系统码编码,则其原始的信息序列 u_0^{K-1} 按照信息位集合的顺序分布在码字 x_0^{N-1} 之中,因此在快速译码得到码字估值序列 $\hat{x}_0^{N-1}$ 之后,只需要按照信息位集合的顺序取出 $\hat{u}_0^{K-1}$ 即可. 若极化码进行非系统码编码,在得到码字估值序列 $\hat{x}_0^{N-1}$ 之后,需要进行 $\hat{u}_0^{N-1}=\hat{x}_0^{N-1}\boldsymbol{F}^{\otimes n}$ 的运算,之后再按照信息位集合的顺序从 $\hat{u}_0^{N-1}$ 中取出 $\hat{u}_0^{K-1}$. 本文后续研究基于非系统编码开展.

3　Fast-PC-SCL译码算法

为了更好地描述节点的特征,通常使用 A 表示信息位的索引集合,A^C 表示冻结位的索引集合,$P=\{p_0,p_1,\cdots,p_N\}$ 表示PC校验位置的索引集合. 使用该方法可以快速地表示节点的特征. 例如,Rate-0节点是只包含冻结比特的节点,可表示为 $A=\varnothing$, $A^C=\{0,1,\cdots,N-1\}$;Rate-1节点是只包含信息比特的节点,可表示为 $A=\{0,1,\cdots,N-1\}$, $A^C=\varnothing$;SPC节点是只有首位为冻结比特的节点,可表示为 $A=\{1,2,\cdots,N-1\}$, $A^C=\{0\}$;REP节点是只有末位为信息比特的节点,可表示为 $A=\{N-1\}$, $A^C=\{0,1,\cdots,N-2\}$. 而SR节点是一类包含众多特殊节点的高层节点,一般表示为SR(v, SNT, r),其中 v 是与左子节点相关的参数,SNT表示源节点的类型,r 表示该节点在译码树中的层级.

本节分别介绍PC-REP类节点和PC-SPC类节点,这2类节点分别具有码字重复和码字和为定值的特点,下面进行具体描述.

3.1　PC-REP类节点

PC-REP类节点具有以下典型特征,即该码字中前 $N-4$ 位为冻结位,后面4位是信息位、冻结位或PC位,且后面4位中存在至少1位PC位.

$$p=[\underbrace{0,0,\cdots,0}_{N},v_0,v_1,v_2,v_3] \tag{13}$$

定理1　PC-REP类节点存在以下特性:经过编码的码字 x_0^{N-1} 是以4为周期的重复序列,即

$$\begin{aligned}x_0^{N-1}=(&x_{N-4},x_{N-3},x_{N-2},x_{N-1},\cdots,\\&x_{N-4},x_{N-3},x_{N-2},x_{N-1})\end{aligned} \tag{14}$$

且

$$x_{N-4}^{N-1}=\left[v_0\oplus v_1\oplus v_2\oplus v_3,v_1\oplus v_3,v_2\oplus v_3,v_3\right] \tag{15}$$

下面采用数学归纳法对定理1进行证明.

证明:　由极化码编码特性可知,$x_0^{N-1}=u_0^{N-1}\boldsymbol{F}^{\otimes n}$,

其中$\boldsymbol{F}^{\otimes n}$是Arikan核的$n$次克劳内克幂，定义如下：

$$\boldsymbol{F}^{\otimes n}=\begin{bmatrix}\boldsymbol{F}^{\otimes(n-1)} & \boldsymbol{0}\\ \boldsymbol{F}^{\otimes(n-1)} & \boldsymbol{F}^{\otimes(n-1)}\end{bmatrix} \tag{16}$$

令$u_0^{N-1}=[\boldsymbol{u}_0,\boldsymbol{u}_1]$，$x_0^{N-1}=[\boldsymbol{x}_0,\boldsymbol{x}_1]$，带入式(16)可得$\boldsymbol{x}_0=\boldsymbol{u}_0\boldsymbol{F}^{\otimes n-1}+\boldsymbol{u}_1\boldsymbol{F}^{\otimes n-1}$和$\boldsymbol{x}_1=\boldsymbol{u}_1\boldsymbol{F}^{\otimes n-1}$，即

$$\boldsymbol{u}_0\boldsymbol{F}^{\otimes n-1}=\boldsymbol{x}_0+\boldsymbol{x}_1 \tag{17}$$

对于PC-REP类节点而言，当$N=4$时，$u_0^3=[v_0,v_1,v_2,v_3]$，有

$$x_0^3=u_0^3\boldsymbol{F}^{\otimes 2}=[v_0\oplus v_1\oplus v_2\oplus v_3,v_1\oplus v_3,v_2\oplus v_3,v_3] \tag{18}$$

假设当$N=N_1$，$u_0^{N_1-1}=[0,0,0,\cdots,0,v_0,v_1,v_2,v_3]$时该结论成立，即

$$\begin{aligned}x_0^{N_1-1}&=u_0^{N_1-1}\boldsymbol{F}^{\otimes\log_2 N_1}\\&=[v_0\oplus v_1\oplus v_2\oplus v_3,v_1\oplus v_3,v_2\oplus v_3,v_3,\cdots,\\&\quad v_0\oplus v_1\oplus v_2\oplus v_3,v_1\oplus v_3,v_2\oplus v_3,v_3]\end{aligned} \tag{19}$$

当$N=2N_1$时，有

$$\begin{aligned}x_0^{2N_1-1}&=u_0^{2N_1-1}\boldsymbol{F}^{\otimes\log_2 2N_1}\\&=[\boldsymbol{u_0},\boldsymbol{u_1}]\boldsymbol{F}^{\otimes\log_2 2N_1}\\&=[\boldsymbol{0},u_0^{N_1-1}]\begin{bmatrix}\boldsymbol{F}^{\otimes\log_2 N_1} & \boldsymbol{0}\\ \boldsymbol{F}^{\otimes\log_2 N_1} & \boldsymbol{F}^{\otimes\log_2 N_1}\end{bmatrix}\\&=[x_0^{N_1-1},x_0^{N_1-1}]\end{aligned} \tag{20}$$

即

$$\begin{aligned}x_0^{2N_1-1}=[&v_0\oplus v_1\oplus v_2\oplus v_3,v_1\oplus v_3,v_2\oplus v_3,v_3,\cdots,\\&v_0\oplus v_1\oplus v_2\oplus v_3,v_1\oplus v_3,v_2\oplus v_3,v_3]\end{aligned} \tag{21}$$

证毕.

针对PC-REP类节点，定理1给出了其典型特征，具体的码字还有更简洁的性质，这些性质可以根据上述结论很快地推导出来. 表1给出了Type-I~Type-VI节点的具体定义和性质，注意PC-REP类节点中的部分节点与文献[16]中提出的部分节点类似.

结合表1中各节点的性质可知，各个节点中PC比特的值可以由前面的信息序列计算得到，即在处理PC-REP类节点时，PC比特的值已知. 因此，在译码时可以按照遍历的方法给出候选的码字列表. 下面给出各个节点的快速SCL译码算法.

3.1.1 Type-I节点

Type-I节点的性质为$x_0=x_1=x_2=\cdots=x_{N-1}=p_0$，根据PC-Polar码的编码特点，$p_0$一定在该节点之前可以通过前面的信息比特计算出来. 因此在SCL译码过程中，若长度为N_v的节点是Type-I节点，则对该节点的码字比特估计为$(p_0)_0^{N_v-1}$. 对于每一条译码路径l，只需在原译码序列基础上再延续保存$u_0^{N_v-1}=(p_0)_0^{N_v-1}\boldsymbol{F}^{\otimes\log_2 N_v}$这样一条路径即可，同时更新度量值$\mathrm{PM}_{\mathrm{ML}}^{l}=\mathrm{PM}^{l}+\sum_{i=0}^{N_v-1}\ln(1+\mathrm{e}^{-(1-2p_0)\alpha_i})$. 其中，$\mathrm{PM}^{l}$表示第$l$个译码器之前累计的度量值，$\alpha_0^{N_v-1}$表示该子码节点接收到的初始LLR序列.

3.1.2 Type-II~Type-III节点

Type-II节点的性质为$x_0=x_2=\cdots=x_{N-2}=p_0\oplus x_{N-1}$，$x_1=x_3=\cdots=x_{N-1}$，同样，$p_0$可通过前面的信息比特计算出来. 由于$x_{N-1}\in\{0,1\}$，若长度为$N_v$的节点是Type-II节点，则对于该节点的2个码字比特估计分别为$x(0)_0^{N_v-1}=\{p_0,0,\cdots,p_0,0\}$和$x(1)_0^{N_v-1}=\{p_0,1,\cdots,p_0,1\}$，

表1 Type-I~Type-VI节点定义及性质

名称	定义	性质
Type-I	$A=\varnothing,A^C=\{0,1,\cdots,N-2\}$, $P=\{N-1\},N\geqslant 4$	$x_0=x_1=\cdots=x_{N-1}=p_0$
Type-II	$A=\{N-1\},A^C=\{0,1,\cdots,N-3\}$, $P=\{N-2\},N\geqslant 4$	$x_0=x_2=\cdots=x_{N-2}=p_0\oplus x_{N-1}$ $x_1=x_3=\cdots=x_{N-1}$
Type-III	$A=\{N-1\},A^C=\{0,1,\cdots,N-4\}$, $P=\{N-3,N-2\},N\geqslant 4$	$x_0=x_4=\cdots=x_{N-4}=p_0\oplus p_1\oplus x_{N-1}$ $x_1=x_5=\cdots=x_{N-3}=p_0\oplus x_{N-1}$ $x_2=x_6=\cdots=x_{N-2}=p_1\oplus x_{N-1}$ $x_3=x_7=\cdots=x_{N-1}=x_{N-1}$
Type-IV	$A=\{N-2,N-1\},A^C=\{0,1,\cdots,N-4\}$, $P=\{N-3\},N\geqslant 4$	$x_0=x_4=\cdots=x_{N-4}=p_0\oplus x_{N-2}$ $x_1=x_5=\cdots=x_{N-3}=p_0\oplus x_{N-1}$ $x_2=x_6=\cdots=x_{N-2}$ $x_3=x_7=\cdots=x_{N-1}$
Type-V	$A=\{N-2,N-1\},A=\{0,1,\cdots,N-5\}$, $P=\{N-4,N-3\},N\geqslant 4$	$x_0=x_4=\cdots x_{N-4}=p_0\oplus p_1\oplus x_{N-2}$ $x_1=x_5=\cdots=x_{N-3}=p_1\oplus x_{N-1}$ $x_2=x_6=\cdots=x_{N-2}$ $x_3=x_7=\cdots=x_{N-1}$
Type-VI	$A=\{N-3,N-2,N-1\},A^C=\{0,1,\cdots,N-5\}$, $P=\{N-4\},N\geqslant 4$	$x_0=x_4=\cdots=x_{N-4}=p_0\oplus x_{N-3}\oplus x_{N-2}\oplus x_{N-1}$ $x_1=x_5=\cdots=x_{N-3}$ $x_2=x_6=\cdots=x_{N-2}$ $x_3=x_7=\cdots=x_{N-1}$

每一条译码路径 l 在此时扩展为 2 条,分别延续保存 $u(0)_0^{N_v-1}=x(0)_0^{N_v-1}\boldsymbol{F}^{\otimes\log_2 N_v}$ 和 $u(1)_0^{N_v-1}=x(1)_0^{N_v-1}\boldsymbol{F}^{\otimes\log_2 N_v}$,并分别更新度量值 $\mathrm{PM}_{\mathrm{ML}}^{l}=\mathrm{PM}_l+\sum_{i=0}^{N_v-1}\ln\left(1+\mathrm{e}^{-(1-2x(0)_i)\alpha_i}\right)$ 和 $\mathrm{PM}_{\mathrm{ML}}^{l}=\mathrm{PM}_l+\sum_{i=0}^{N_v-1}\ln\left(1+\mathrm{e}^{-(1-2x(1)_i)\alpha_i}\right)$. 此时的路径数量扩展为原来的 2 倍,译码器根据度量值排序并进行路径删除,以维持大小为 L 的列表宽度.

针对 Type-III 节点,由于其码字序列是以 2 为周期的重复序列,则该类节点的码字估计列表表示为 $x(0)_0^{N_v-1}=\{p_0\oplus p_1,p_0,p_1,0,\cdots,p_0\oplus p_1,p_0,p_1,0\}$, $x(1)_0^{N_v-1}=\{p_0\oplus p_1\oplus 1,p_0\oplus 1,p_1\oplus 1,1,\cdots,p_0\oplus p_1\oplus 1,\ p_0\oplus 1,\ p_1\oplus 1,1\}$,之后按照与 Type-II 节点类似的方案进行.

3.1.3　Type-IV~TypeV 节点

Type-IV 节点的性质为 $x_0=x_4=\cdots=x_{N-4}=p_0\oplus x_{N-2}$, $x_1=x_5=\cdots=x_{N-3}=p_0\oplus x_{N-1}$, $x_2=x_6=\cdots=x_{N-2}$, $x_3=x_7=\cdots=x_{N-1}$,同样 p_0 可通过前面的信息比特计算出来. 若有 $x_{N-2}\in\{0,1\}$ 和 $x_{N-1}\in\{0,1\}$,则 Type-IV 节点的码字估计列表表示为

$$\begin{aligned}
&x(0)_0^{N_v-1}=\{p_0,p_0,0,0,\cdots,p_0,p_0,0,0\}\\
&x(1)_0^{N_v-1}=\{p_0,p_0\oplus 1,0,1,\cdots,p_0,p_0\oplus 1,0,1\}\\
&x(2)_0^{N_v-1}=\{p_0\oplus 1,p_0,1,0,\cdots,p_0\oplus 1,p_0,1,0\}\\
&x(3)_0^{N_v-1}=\{p_0\oplus 1,p_0\oplus 1,1,1,\cdots,p_0\oplus 1,p_0\oplus 1,1,1\}
\end{aligned}\tag{22}$$

每一条译码路径 l 在此时扩展为 4 条,分别延续保存 $u(0)_0^{N_v-1}=x(0)_0^{N_v-1}\boldsymbol{F}^{\otimes\log_2 N_v}$、$u(1)_0^{N_v-1}=x(1)_0^{N_v-1}\boldsymbol{F}^{\otimes\log_2 N_v}$、$u(2)_0^{N_v-1}=x(2)_0^{N_v-1}\boldsymbol{F}^{\otimes\log_2 N_v}$ 和 $u(3)_0^{N_v-1}=x(3)_0^{N_v-1}\boldsymbol{F}^{\otimes\log_2 N_v}$,此时的路径数量扩展为原来的 4 倍,译码器进行同样的度量值排序和路径删除.

Type-V 节点与 Type-IV 节点的处理方法类似,若长度为 N_v 的节点是 Type-VI 节点,则该类节点的码字估计列表为

$$\begin{aligned}
x(0)_0^{N_v-1}=&\{p_0\oplus p_1,p_1,0,0,\cdots,\\
&p_0\oplus p_1,p_1,0,0\}\\
x(1)_0^{N_v-1}=&\{p_0\oplus p_1,p_1\oplus 1,0,1,\cdots,\\
&p_0\oplus p_1,p_1\oplus 1,0,1\}\\
x(2)_0^{N_v-1}=&\{p_0\oplus p_1\oplus 1,p_1,1,0,\cdots,\\
&p_0\oplus p_1\oplus 1,p_1,1,0\}\\
x(3)_0^{N_v-1}=&\{p_0\oplus p_1\oplus 1,p_1\oplus 1,1,1,\cdots,\\
&p_0\oplus p_1\oplus 1,p_1\oplus 1,1,1\}
\end{aligned}\tag{23}$$

其余的处理与 Type-IV 节点相同.

3.1.4　Type-VI 节点

对于码字长度为 N_v 的 Type-VI 节点而言,由于 $x_{N-3}\in\{0,1\}$、$x_{N-2}\in\{0,1\}$ 和 $x_{N-1}\in\{0,1\}$,p_0 的值在之前也已经计算出来,则该类节点的码字估计列表为

$$\begin{aligned}
&x(0)_0^{N_v-1}=\{p_0,0,0,0,\cdots,p_0,0,0,0\}\\
&x(1)_0^{N_v-1}=\{p_0\oplus 1,0,0,1,\cdots,p_0\oplus 1,0,0,1\}\\
&x(2)_0^{N_v-1}=\{p_0\oplus 1,0,1,0,\cdots,p_0\oplus 1,0,1,0\}\\
&x(3)_0^{N_v-1}=\{p_0,0,1,1,\cdots,p_0,0,1,1\}\\
&x(4)_0^{N_v-1}=\{p_0\oplus 1,1,0,0,\cdots,p_0\oplus 1,1,0,0\}\\
&x(5)_0^{N_v-1}=\{p_0,1,0,1,\cdots,p_0,1,0,1\}\\
&x(6)_0^{N_v-1}=\{p_0,1,1,0,\cdots,p_0,1,1,0\}\\
&x(7)_0^{N_v-1}=\{p_0\oplus 1,1,1,1,\cdots,p_0\oplus 1,1,1,1\}
\end{aligned}\tag{24}$$

每一条路径 l 扩展为 8 条,分别延续保存 $u(j)_0^{N_v-1}=x(j)_0^{N_v-1}\boldsymbol{F}^{\otimes\log_2 N_v}$, $0\leqslant j\leqslant 7$,然后根据式(7)进行 PM 值的更新,并维持大小为 L 的列表.

3.2　PC-SPC 类节点

这类节点有如下典型特征,即该码字中前 3 位为冻结位、信息位或者 PC 位置,其余位置为信息位.

$$\boldsymbol{p}=[\underbrace{v_0,v_1,v_2,I_0,\cdots,I_{N-3},I_{N-4}}_{N}]\tag{25}$$

定理 2　PC-SPC 类节点存在以下特性:经过编码的码字 x_0^{N-1} 满足:

$$\begin{aligned}
&\oplus_{i=0}^{\frac{N}{4}-1}x_{4i}=v_0\oplus v_1\oplus v_2\oplus z\\
&\oplus_{i=0}^{\frac{N}{4}-1}x_{4i+1}=v_1\oplus z\\
&\oplus_{i=0}^{\frac{N}{4}-1}x_{4i+2}=v_2\oplus z\\
&z=\oplus_{i=0}^{\frac{N}{4}-1}x_{4i+3}
\end{aligned}\tag{26}$$

下面采用数学归纳法对定理 2 进行证明.

证明:　对于 PC-SPC 类节点而言,当 $N=4$ 时,$u_0^3=[v_0,v_1,v_2,I_0]$,有

$$\begin{aligned}
x_0^3&=u_0^3\boldsymbol{F}^{\otimes 2}\\
&=[v_0\oplus v_1\oplus v_2\oplus I_0,v_1\oplus I_0,v_2\oplus I_0,I_0]
\end{aligned}\tag{27}$$

假设当 $N=N_1$, $u_0^{N_1-1}=\boldsymbol{u}_0=[v_0,v_1,v_2,I_0,\cdots,I_{N_1-4}]$ 时该结论成立,即

$$\begin{aligned}
&\oplus_{i=0}^{\frac{N_1}{4}-1}x_{4i}=v_0\oplus v_1\oplus v_2\oplus z\\
&\oplus_{i=0}^{\frac{N_1}{4}-1}x_{4i+1}=v_1\oplus z\\
&\oplus_{i=0}^{\frac{N_1}{4}-1}x_{4i+2}=v_2\oplus z\\
&z=\oplus_{i=0}^{\frac{N_1}{4}-1}x_{4i+3}
\end{aligned}\tag{28}$$

当 $N=2N_1$ 时,$u_0^{2N_1-1}=[\boldsymbol{u}_0,\boldsymbol{u}_1]=[v_0,v_1,v_2,I_0,\cdots,I_{2N_1-1}]$,有

$$\begin{aligned}
x_0^{2N_1-1}&=u_0^{2N_1-1}\boldsymbol{F}^{\otimes\log_2 2N_1}\\
&=[\boldsymbol{u}_0,\boldsymbol{u}_1]\boldsymbol{F}^{\otimes\log_2 2N_1}\\
&=[\boldsymbol{u}_0,\boldsymbol{u}_1]\begin{bmatrix}\boldsymbol{F}^{\otimes\log_2 N_1} & \boldsymbol{0}\\ \boldsymbol{F}^{\otimes\log_2 N_1} & \boldsymbol{F}^{\otimes\log_2 N_1}\end{bmatrix}\\
&=[\boldsymbol{x}_0,\boldsymbol{x}_1]
\end{aligned}\tag{29}$$

由式(17)可知 $\boldsymbol{u}_0\boldsymbol{F}^{\otimes n-1}=\boldsymbol{x}_0+\boldsymbol{x}_1$，即 $x_i=x_i\oplus x_{i+N_1}$，$0\leqslant i\leqslant N_1-1$，即

$$\begin{aligned}&\oplus_{i=0}^{\frac{N_1}{4}-1}x_{4i}+x_{4i+N_1}=v_0\oplus v_1\oplus v_2\oplus z\\&\oplus_{i=0}^{\frac{N_1}{4}-1}x_{4i+1}+x_{4i+N_1+1}=v_1\oplus z\\&\oplus_{i=0}^{\frac{N_1}{4}-1}x_{4i+2}+x_{4i+N_1+2}=v_2\oplus z\\&z=\oplus_{i=0}^{\frac{N_1}{4}-1}x_{4i+3}+x_{4i+N_1+3}\end{aligned}\tag{30}$$

对式(30)进行简化，可得：

$$\begin{aligned}&\oplus_{i=0}^{\frac{2N_1}{4}-1}x_{4i}=v_0\oplus v_1\oplus v_2\oplus z\\&\oplus_{i=0}^{\frac{2N_1}{4}-1}x_{4i+1}=v_1\oplus z\\&\oplus_{i=0}^{\frac{2N_1}{4}-1}x_{4i+2}=v_2\oplus z\\&z=\oplus_{i=0}^{\frac{2N_1}{4}-1}x_{4i+3}\end{aligned}\tag{31}$$

证毕.

定理2给出了PC-SPC类节点的典型性质，具体的码字存在更简洁的性质，这些性质可以根据上述结论快速地推导出来，表2给出了Type-VII~Type-XI节点的具体定义和性质，注意PC-SPC类节点中的部分节点与文献[16]中提出的部分节点类似.

由表2可知，PC-SPC类节点的特点是某些固定位置的码字满足和为固定值，这些固定位置的码字称为SPC码字. 为了简化后续算法的描述过程，本文按照文献[18]中给出的SPC码字的解码方法给出下面的函数定义：

$$(q,\epsilon,i)=\text{SPCDecoder}(\alpha_0^{N-1})\tag{32}$$

其中，$q=\oplus_{i=0}^{N-1}h(\alpha_i)$，$i=\text{argmin}_{0\leqslant i\leqslant N-1}\{|\alpha_i|\}$，$\epsilon=\min|\alpha_i|$.

下面具体描述各个节点的快速SCL译码算法.

3.2.1 Type-VII ~ Type-IX 节点

Type-VII类节点的性质为 $\oplus_{i=0}^{\frac{N}{4}-1}x_{4i}=\oplus_{i=0}^{\frac{N}{4}-1}x_{4i+2}=p_0\oplus z$，$\oplus_{i=0}^{\frac{N}{4}-1}x_{4i+1}=\oplus_{i=0}^{\frac{N}{4}-1}x_{4i+3}=z$，即Type-VII构成4个SPC码字，其中 $(x_0,x_4,\cdots,x_{N-4})$ 和 $(x_2,x_6,\cdots,x_{N-2})$ 是满足码字和为 $p_0\oplus z$ 校验的SPC码字，$(x_1,x_5,\cdots,x_{N-3})$ 和 $(x_3,x_7,\cdots,x_{N-1})$ 是满足码字和为 z 校验的SPC码字. 因此对于长度为 N_v 的Type-VII节点，其码字列表的估计方法如下.

步骤1：对上述4个SPC码字进行计算，

$$(q_j,\epsilon_j,i_j)=\text{SPCDecoder}(\alpha_{4k+j})\tag{33}$$

其中，$0\leqslant k\leqslant N_v/4-1$，$0\leqslant j\leqslant 3$.

计算 t 值并同时保留 $z=0$ 和 $z=1$ 对应的码字，即

$$\begin{cases}t_0=q_0\oplus p_0\oplus z\\t_1=q_1\oplus z\\t_2=q_2\oplus p_0\oplus z\\t_3=q_3\oplus z\\x_i=h(\alpha_i), & 0\leqslant i\leqslant N_v-1,i\neq 4i_j+j\\x_{4i_j+j}=h(\alpha_{4i_j+j})\oplus t_j, & 0\leqslant j\leqslant 3\end{cases}\tag{34}$$

得到该节点的2个码字估计分别为 $x(0)_0^{N_v-1}$ 和 $x(1)_0^{N_v-1}$（这2个码字的差异主要在于 z 值的选取）.

表2 Type-VII~Type-XI节点定义及性质

名称	定义	性质
Type-VII	$A=\{3,4,\cdots,N-1\},A^C=\{0,1\},$ $P=\{2\},N\geqslant 4$	$\oplus_{i=0}^{\frac{N}{4}-1}x_{4i}=\oplus_{i=0}^{\frac{N}{4}-1}x_{4i+2}=p_0\oplus z$ $\oplus_{i=0}^{\frac{N}{4}-1}x_{4i+1}=\oplus_{i=0}^{\frac{N}{4}-1}x_{4i+3}=z$
Type-VIII	$A=\{3,4,\cdots,N-1\},A^C=\{0\},$ $P=\{1,2\},N\geqslant 4$	$\oplus_{i=0}^{\frac{N}{4}-1}x_{4i}=p_0\oplus p_1\oplus z$, $\oplus_{i=0}^{\frac{N}{4}-1}x_{4i+1}=p_0\oplus z$ $\oplus_{i=0}^{\frac{N}{4}-1}x_{4i+2}=p_1\oplus z$, $\oplus_{i=0}^{\frac{N}{4}-1}x_{4i+3}=z$
Type-IX	$A=\{3,4,\cdots,N-1\},A^C=\varnothing,$ $P=\{0,1,2\},N\geqslant 4$	$\oplus_{i=0}^{\frac{N}{4}-1}x_{4i}=p_0\oplus p_1\oplus p_2\oplus z$, $\oplus_{i=0}^{\frac{N}{4}-1}x_{4i+1}=p_1\oplus z$ $\oplus_{i=0}^{\frac{N}{4}-1}x_{4i+2}=p_2\oplus z$, $\oplus_{i=0}^{\frac{N}{4}-1}x_{4i+3}=z$
Type-X	$A=\{2,3,\cdots,N-1\},A^C=\{0\},$ $P=\{1\},N\geqslant 4$	$\oplus_{i=0}^{\frac{N}{2}-1}x_{2i}=\oplus_{i=0}^{\frac{N}{2}-1}x_{2i+1}=p_0$
Type-XI	$A=\{1,2,\cdots,N-1\},A^C=\varnothing,$ $P=\{0\},N\geqslant 4$	$\oplus_{i=0}^{N-1}x_i=p_0$

步骤2：每一条译码路径 l 扩展为2条，分别暂时保存 $x(0)_0^{N_v-1}$ 和 $x(1)_0^{N_v-1}$ 这2个码字结果，并计算每条路径的度量值 $\text{PM}_{\text{ML}}^l=\text{PM}^l+\sum_{i=0}^{N-1}\ln\left(1+\text{e}^{-|\alpha_i|}\right)+\sum_{j=0}^{3}t_j|\alpha_{4i_j+j}|$. 其中，$\text{PM}^l$ 表示第 l 个译码器之前累计的度量值. 此时整个译码器中的路径需要按照度量值进行剪枝，维持 L 的列表大小. 而 $x(0)_0^{N_v-1}$ 和 $x(1)_0^{N_v-1}$ 未必是正确的码字估计，因此需要通过翻转 $x(0)_0^{N_v-1}$ 和 $x(1)_0^{N_v-1}$ 中的码字比特来获得其他候选码字. 由于Type-VII节点中包含的信息位可能较大，如果仍然采用PC-REP类节点中遍历的方法则复杂度太高. 可以参考文献[11]中的方法来生成其他码字序列，即从可靠度最低码字比特开始，同时

保留其判决值为0和1的码字估计，并按照度量值大小进行剪枝.

步骤3：重新定义一组加权似然比值用于计算.

$$\begin{aligned} &y_{4k+j}=|\alpha_{4k+j}|+(1-2t_j)|\epsilon_j| \\ &0\leqslant k\leqslant N_v/4-1, 0\leqslant j\leqslant 3 \end{aligned} \tag{35}$$

使用二维数组 $\mathrm{arr}[i][y_i], 0\leqslant i\leqslant N_v-1$ 对加权似然比值进行储存，数组第0行为 y_i 对应索引，数组第1行为 y_i. 按照 y_i 的大小对数组进行升序排序（不包含 $4i_j+j$ 等4个位置），即 $y_k\leqslant y_{k+1}$.

步骤4：从 $k=0$ 开始，每条路径 l 复制为2条路径，分别将 $x_0^{N_v-1}$（即当前路径 l 对应的Type-VI节点的码字估计）中的 $x_{\mathrm{arr}[0][k]}$ 设置为0和1. 同时为了满足Type-VII性质中的校验特性，需要按照校验规则选择翻转 $4i_j+j$ 位置的比特，使其满足原始的校验和为固定值的规则. 同时更新度量值PM，再更新 t 值，PM和 t 的计算规则如下：

$$\mathrm{PM}_{(k)}^{l}=\begin{cases} \mathrm{PM}_{(k-1)}^{l}, & x_k=h(\alpha_{\mathrm{arr}[0][k]}) \\ \mathrm{PM}_{(k-1)}^{l}+\left|\alpha_{\mathrm{arr}[0][k]}\right| & \\ \quad +(1-2t_j)|\alpha_{4i_j+j}|, & \text{otherwise} \end{cases} \tag{36}$$

$$t_j=\begin{cases} t_j, & x_k=h(\alpha_{\mathrm{arr}[0][k]}) \\ 1-t_j, & \text{otherwise} \end{cases} \tag{37}$$

其中，$\mathrm{PM}_{(-1)}^{l}=\mathrm{PM}_{\mathrm{ML}}^{l}$，为步骤2计算所得的路径度量值. 由此每条路径扩展为2条，译码器在 $2L$ 条路径中按照度量值大小进行剪枝，保留 L 条路径，之后重复步骤4中的翻转过程，直到 $k=\min(L-2, N-5)$.

此时对于每条路径 l 都得到了码字比特的最终估计，在路径中延续保存 $u_0^{N_v-1}=x_0^{N_v-1}\boldsymbol{F}^{\otimes\log_2 N_v}$，Type-VII节点的快速译码结束.

Type-VIII节点和Type-VII节点类似，其特点是 $(x_0,x_4,\cdots,x_{N-4})$，$(x_1,x_5,\cdots,x_{N-3})$，$(x_2,x_6,\cdots,x_{N-2})$，$(x_3,x_7,\cdots,x_{N-1})$ 分别构成4个满足 $(p_0\oplus p_1\oplus z, p_0\oplus z, p_1\oplus z, z)$ 校验的SPC码字. 因此可以采用与上述Type-VII节点类似的方法，仅 t 值的定义更改如下：

$$\begin{cases} t_0=q_0\oplus p_0\oplus p_1\oplus z \\ t_1=q_1\oplus p_0\oplus z \\ t_2=q_2\oplus p_1\oplus z \\ t_3=q_3\oplus z \end{cases} \tag{38}$$

同样，Type-IX节点也具有和Type-VII相似的性质：$(x_0,x_4,\cdots,x_{N-4})$，$(x_1,x_5,\cdots,x_{N-3})$，$(x_2,x_6,\cdots,x_{N-2})$，$(x_3,x_7,\cdots,x_{N-1})$ 分别构成4个满足 $(p_0\oplus p_1\oplus p_2\oplus z, p_1\oplus z, p_2\oplus z, z)$ 校验的SPC码字，同样可以采用类似Type-VII节点的方法进行译码，t 值更新如下：

$$\begin{cases} t_0=q_0\oplus p_0\oplus p_1\oplus p_2\oplus z \\ t_1=q_1\oplus p_1\oplus z \\ t_2=q_2\oplus p_2\oplus z \\ t_3=q_3\oplus z \end{cases} \tag{39}$$

3.2.2　Type-X节点

Type-X节点具有偶数位置和奇数位置码字分别构成2个满足 p_0 校验的SPC码字的性质，故长度为 N_v 的Type-X节点码字列表的估计方法如下.

步骤1：针对上述2个SPC码字进行计算，

$$(q_j,\epsilon_j,i_j)=\mathrm{SPCDecoder}(\alpha_{2k+j}) \tag{40}$$

其中，$0\leqslant k\leqslant N_v/2-1, 0\leqslant j\leqslant 1$.

然后计算 t 值，并根据式(34)进行码字计算，得到码字估计 $x_0^{N_v-1}$.

$$\begin{cases} t_0=q_0\oplus p_0 \\ t_1=q_1\oplus p_0 \\ x_i=h(\alpha_i), & 0\leqslant i\leqslant N_v-1, i\neq 2i_j+j \\ x_{2i_j+j}=h(\alpha_{2i_j+j})\oplus t_j, & 0\leqslant j\leqslant 1 \end{cases} \tag{41}$$

步骤2：译码路径 l 暂时保存 $x_0^{N_v-1}$ 码字结果，并计算度量值 $\mathrm{PM}^{l}=\mathrm{PM}_{\mathrm{ML}}^{l}+\sum_{i=0}^{N-1}\ln(1+\mathrm{e}^{-|\alpha_i|})+t_0|\alpha_{2i_0}|+t_1|\alpha_{2i_1+1}|$，其中 $\mathrm{PM}_{\mathrm{ML}}^{l}$ 表示第 l 个译码器之前累计的度量值. 同样，$x_0^{N_v-1}$ 未必是最佳的码字估计，因此需要生成其他的码字估计.

步骤3：重新定义一组加权似然比值用于计算.

$$\begin{cases} y_{2k}=|\alpha_{2k}|+(1-2t_0)|\epsilon_0|, \\ y_{2k+1}=|\alpha_{2k+1}|+(1-2t_1)|\epsilon_1|, \end{cases} \quad 0\leqslant k\leqslant N_v/2-1 \tag{42}$$

使用二维数组 $\mathrm{arr}[i][y_i], 0\leqslant i\leqslant N_v-1$ 对加权似然比值进行储存，数组第0行为 y_i 对应索引，数组第1行为 y_i. 按照 y_i 的大小对数组进行升序排序（不包含 $2i_0, 2i_1+1$ 这2个位置），即 $y_k\leqslant y_{k+1}$.

步骤4：从 $k=0$ 开始，每条路径 l 复制为2条路径，分别将 $x_0^{N_v-1}$（即当前路径 l 对应的Type-X节点的码字估计）中的 $x_{\mathrm{arr}[0][k]}$ 设置为0和1. 同时为了满足Type-X性质中的校验特性，需要按照校验规则来选择翻转 $2i_0$ 和 $2i_1+1$ 位置的比特，使其满足原始的校验和为固定值的规则. 同时更新度量值PM，再更新 t 值，PM和 t 计算规则如下：

$$\mathrm{PM}_{(k)}^{l}=\begin{cases} \mathrm{PM}_{(k-1)}^{l}, & x_k=h(\alpha_{\mathrm{arr}[0][k]}) \\ \mathrm{PM}_{(k-1)}^{l}+\left|\alpha_{\mathrm{arr}[0][k]}\right| & \\ \quad +(1-2t_j)|\alpha_{2i_j+j}|, & \text{otherwise} \end{cases} \tag{43}$$

$$t_j=\begin{cases} t_j, & x_k=h(\alpha_{\mathrm{arr}[0][k]}) \\ 1-t_j, & \text{otherwise} \end{cases} \tag{44}$$

其中，$\mathrm{PM}_{(-1)}^{l}=\mathrm{PM}_{\mathrm{ML}}^{l}$，而 $\mathrm{PM}_{\mathrm{ML}}^{l}$ 为步骤2计算所得的路径

度量值. 由此每条路径扩展为2条,译码器在2L条路径中按照度量值大小进行剪枝,保留L条路径,重复步骤4中的翻转过程,直到$k=\min(L-2,N-3)$.

此时对于每条路径l都得到了码字比特的最终估计,在路径中延续保存$u_0^{N_v-1}=x_0^{N_v-1}\boldsymbol{F}^{\otimes\log_2 N_v}$,Type-VII节点的快速译码结束.

3.2.3 Type-XI节点

由于Type-XI节点具有$\oplus_{i=0}^{N-1}x_i=p_0$的性质,因此该节点就是一个特殊的SPC码字. 故长度为N_v的Type-XI节点码字列表的估计方法如下.

步骤1:针对该SPC码字进行计算,

$$\begin{cases}(q,\epsilon,j)=\text{SPCDecoder}(\alpha_i), & 0\leqslant i\leqslant N-1\\ x_i=h(\alpha_i)\oplus q\oplus p_0 & \\ x_j=h(\alpha_j), & 0\leqslant i\leqslant N-1,i\neq j\end{cases}\tag{45}$$

步骤2:译码路径l暂时保存$x_0^{N_v-1}$码字结果,并计算度量值$\text{PM}_{\text{ML}}^{l}=\text{PM}^{l}+\sum_{i=0}^{N-1}\ln(1+\text{e}^{-|\alpha_i|})+(q\oplus p_0)|\alpha_j|$. 同样,$\text{PM}^{l}$为当前路径之前累计的度量值.

步骤3:使用二维数组arr[i][α_i],$0\leqslant i\leqslant N_v-1$对似然比值进行储存,数组定义与前面相同. 按照$y_i$的大小对数组进行升序排序(不包含$\alpha_j$),即$y_k\leqslant y_{k+1}$.

步骤4:从$k=0$开始,每条路径l复制为2条路径,分别将$x_0^{N_v-1}$(即当前路径l对应的Type-XI节点的码字估计)中的$x_{\text{arr}[0][k]}$设置为0和1,同时选择翻转j位置对应的比特并更新度量值PM. 由此每条路径扩展为2条,译码器在2L条路径中按照度量值大小进行剪枝,保留L条路径,重复步骤4中的翻转过程,直到$k=\min(L-2,N-2)$.

此时对于每条路径l都得到了码字比特的最终估计,在路径中延续保存$u_0^{N_v-1}=x_0^{N_v-1}\boldsymbol{F}^{\otimes\log_2 N_v}$,Type-XI节点的快速译码结束.

算法1和算法2分别给出了Fast-PC-SCL中的节点识别算法和Fast-PC-SCL译码算法.

算法1 节点识别算法

输入: 信息位索引集合A,PC位置索引集合A_{PC},码长N

输出: 节点统计列表$\boldsymbol{T}$,列表$\boldsymbol{T}$第一项为节点基本类型,第二项为节点具体类型,第三项为节点起始位置,第四项为节点长度

1. 生成初始长度为N的全1的标记序列$\boldsymbol{f}$,$\boldsymbol{f}(A)=0$,$\boldsymbol{f}(A_{\text{PC}})=2$;生成0~$N-1$的自然数序列$\boldsymbol{z}$,设置计数器cnt=0
2. FUNCTION identifier($\boldsymbol{f}$,$\boldsymbol{z}$)
3. IF length($\boldsymbol{z}$)≥4
4. IF all($\boldsymbol{z}$(4:end)==1)
5. 判断为PC-REP类节点,设置$\boldsymbol{T}$(cnt,0)=1
6. 详细判断具体类型为Type-k型节点,设置$\boldsymbol{T}$(cnt,1)=k;$\boldsymbol{T}$(cnt,2)=$\boldsymbol{z}$(0);$\boldsymbol{T}$(cnt,3)=N;cnt++;
7. ELSEIF all($\boldsymbol{z}$(3:end)==0)
8. 判断为PC-SPC类节点,设置$\boldsymbol{T}$(cnt,0)=2
9. 详细判断具体节点为Type-k型节点,设置$\boldsymbol{T}$(cnt,1)=k;$\boldsymbol{T}$(cnt,2)=$\boldsymbol{z}$(0);$\boldsymbol{T}$(cnt,3)=N;cnt++
10. ELSE
11. identifier($\boldsymbol{f}$(0:N/2−1),$\boldsymbol{z}$(0:N/2−1))
12. identifier($\boldsymbol{f}$(N/2:end),$\boldsymbol{z}$(N/2:end))
13. END
14. END
15. 将除去PC-REP和PC-SPC类子码节点之间的位置分割成为最接近2的幂次方长度的节点,并按照列表要求添加入$\boldsymbol{T}$内
16. 对$\boldsymbol{T}$按照第三项起始位置进行整体排序

4 Fast-PC-SCL译码算法的性能和时延分析

4.1 性能分析

本仿真实验采用文献[2]中论及的方法进行极化码构造,使用BPSK调制方式并在AWGN信道条件下进行传输. Fast-PC-SCL中的PC-REP类节点和PC-SPC类节点通过遍历和翻转的方式,可以确保具有最小度量值的码字保存在码字列表中,因此可以获得与PC-SCL算法相同的性能. 下面的仿真结果也证明了这一结论. 图2和图3展示了PC-SCL算法和Fast-PC-SCL算法在上述条件下的性能对比. 观察图2与图3可知,在不同码长和不同列表宽度下,Fast-PC-SCL算法能达到和PC-SCL算法相同的性能.

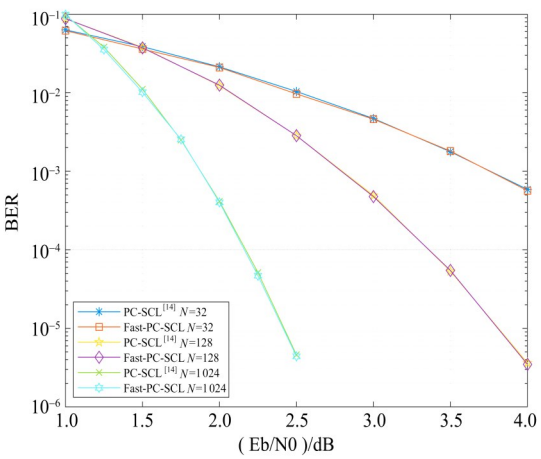


图2 不同码长下2种算法的性能对比($L=8,R=0.5$)

4.2 时延分析

关于时延分析,参考文献[4,6,10]给出如下假设:

硬件资源充足,可以并行化的指令将在1个时钟周期内执行.

h运算,即LLR值的硬判决可以立即执行,不消耗时钟周期,且比特运算的硬判决也是立即执行的.

f运算、g运算以及SPCDecoder、PM的计算都会消耗1个时钟周期.

算法 2　Fast-PC-SCL 译码算法

输入: 信息位索引集合 A,PC 位置索引集合 A_{PC},码长 N, SCL 译码列表宽度 L,信道接收的似然比序列 $\mathrm{LLR}(y_j),0\leqslant j\leqslant N$

输出: 译码估计序列 $\hat{u}_0^{K-1}$

1. 按照节点识别算法识别节点,获得节点统计列表 $\boldsymbol{T}$
2. 根据 $\mathrm{LLR}(y_j),0\leqslant j\leqslant N$ 获得第一个节点的似然比序列 $\alpha(l)_0^{T(0,3)-1}$
3 FOR $i=0$ to $M-1$ do
4. SWITCH $\boldsymbol{T}(i,0)$ do
5. CASE 0:
6. 即为 PC-PER 类子码节点,执行 Type-($\boldsymbol{T}(i,1)$)的译码列表估计方法,给出译码列表估计;$\forall l\in L$,给出下一个节点的似然比序列 $\alpha(l)_{\boldsymbol{T}(i,2)+\boldsymbol{T}(i,3)}^{\boldsymbol{T}(i+1,3)}$(若 $i=M-1$ 则不执行此操作)
7. BREAK;
8. CASE 1:
9. 即为 PC-SPC 类子码节点,执行 Type-($\boldsymbol{T}(i,1)$)的译码列表估计方法,给出译码列表估计;$\forall l\in L$,给出下一个节点的似然比序列 $\alpha(l)_{\boldsymbol{T}(i,2)+\boldsymbol{T}(i,3)}^{\boldsymbol{T}(i+1,3)}$(若 $i=M-1$ 则不执行此操作)
10. BREAK
11. CASE 2:
12. 为普通节点,执行 PC-SCL 译码算法;$\forall l\in L$,给出下一个节点的似然比序列 $\alpha(l)_{\boldsymbol{T}(i,2)+\boldsymbol{T}(i,3)}^{\boldsymbol{T}(i+1,3)}$(若 $i=M-1$ 则不执行此操作)
13. BREAK
14. END
15. END
16. 对 L 个译码列表估计按照度量值 PM 进行升序排序
17. $\hat{u}_0^{K-1}=\hat{u}_0^{N-1}(0)(A)$;

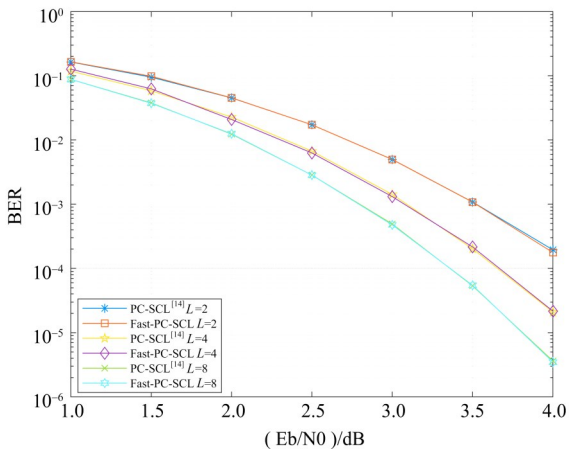


图 3　不同列表宽度下 2 种算法的性能对比($N=128, R=0.5$)

译码器可以在 1 个时钟周期内完成复制和排序的工作.

4.2.1　PC-REP 类节点

PC-REP 类节点只存在 PM 的计算,考虑到在译码过程中最多一次性计算 $2L$ 个 PM 值,因此可以得知:Type-I 节点需要消耗 1 个时钟周期;Type-II~Type-III 节点需要计算最多 $2L$ 个 PM 值,需要消耗 1 个时钟周期;Type-IV~Type-V 节点需要计算最多 $4L$ 个 PM 值,需要消耗 2 个时钟周期;而 Type-VI 节点需要计算最多 $8L$ 个 PM 值,需要消耗 3 个时钟周期.

4.2.2　PC-SPC 类节点

在 Type-VII~Type-IX 节点的快速译码中,首先需要 1 个时钟周期计算最可能的 2 个候选码字,之后需要进行 $\min(L-2,N-5)+1$ 次的翻转计算,需要消耗 $\min(L-2,N-5)+1$ 个时钟周期,因此时间延迟是 $\min(L,N-3)$. 同理,Type-X 节点首先需要 1 个时钟周期计算可能的候选码字,之后进行 $\min(L-2,N-3)+1$ 个时钟周期的翻转计算,总时延是 $\min(L,N-1)$. Type-XI 节点的译码时延为 $\min(L,N)$. 表 3 给出了 PC-SCL 算法和 Fast-PC-SCL 算法所有类型节点的译码时延分析,可以看出,本文提出的 Fast-PC-SCL 算法在所有节点中的译码时延均显著缩短.

表 3　Type-I~Type-XI 节点时延分析表

译码算法 节点类型	PC-SCL[14]	Fast-PC-SCL
Type-I	$2N-2$	1
Type-II	$2N-1$	1
Type-III	$2N-1$	1
Type-IV	$2N$	2
Type-V	$2N$	2
Type-VI	$2N+1$	3
Type-VII	$3N-5$	$\min(L,N-3)$
Type-VIII	$3N-5$	$\min(L,N-3)$
Type-XI	$3N-5$	$\min(L,N-3)$
Type-X	$3N-4$	$\min(L,N-1)$
Type-XI	$3N-3$	$\min(L,N)$

在参数为(32,16)并包含 5 位 PC 校验比特的 PC-Polar 码中,存在 5 个特殊节点,包含 2 个 Type-II 节点、1 个 Type-VI 节点和 1 个 Type-XI 节点. 表 4 给出了分别采用 PC-SCL 算法和本文提出的 Fast-PC-SCL 算法所需的时间步长以及时延缩短比例.

表 4　参数为(32,16)的 PC-Polar 码在 2 种算法下的译码时延

列表宽度	PC-SCL[14]	Fast-PC-SCL	缩短比例/%
L=2	78	35	55.13
L=4	78	37	52.56
L=8	78	41	47.44

在参数为(128,64)并包含 7 位 PC 校验比特的 PC-Polar 码中,存在 6 个特殊节点,包含 1 个 Type-III 节点、2 个 Type-VI 节点和 3 个 Type-XI 节点. 具体的译码时间步长以及时延缩短比例如表 5 所示.

在参数为(1 024,512)并包含 10 位 PC 校验比特的 PC-Polar 中,存在 9 个特殊节点,包含 1 个 Type-VI 节点

表5 参数为(128,64)的PC-Polar码在2种算法下的译码时延

列表宽度	PC-SCL[14]	Fast-PC-SCL	缩短比例/%
L=2	318	219	31.13
L=4	318	225	29.25
L=8	318	237	25.47

和8个Type-XI节点. 具体的译码时间步长以及时延缩短比例如表6所示.

表6 参数为(1 024,512)的PC-Polar码在2种算法下的译码

列表宽度	PC-SCL[14]	Fast-PC-SCL	缩短比例/%
L=2	2 558	1 968	23.06
L=4	2 558	1 984	22.44
L=8	2 558	2 016	21.19

由上述数据可以得出,本文提出的Fast-PC-SCL算法与PC-SCL算法相比,在未降低译码性能的前提下能缩短21.19%~55.13%的时间延迟.

Fast-PC-SCL算法在短码方面表现优异,这是因为在短码中,PC-REP类节点和PC-SPC类节点通常处于译码树的高层,从而能缩短更高比例的时延.

5 结论

针对5G中的PC-Polar码译码算法时延较长的问题,本文提出了PC-Polar码的快速译码算法——Fast-PC-SCL译码算法. 该算法从码字结构出发,找到了PC-Polar码中的PC-REP和PC-SPC 2类特殊节点,并给出了这2类节点的快速SCL译码算法. 与PC-SCL译码算法相比,该算法能在不遍历码树的情况下,给出节点的候选码字列表,提高了算法的并行度. 经过实验仿真验证,在达到与PC-SCL译码算法相同性能的前提下,本文提出的Fast-PC-SCL译码算法有效地缩短了译码时间延迟. 同时,Fast-PC-SCL译码算法适用于所有包含PC码字的极化码,如CRC辅助的PC-Polar码等,因此具有广泛的应用前景.

参考文献

[1] ARIKAN E. Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels[J]. IEEE Transactions on Information Theory, 2009, 55(7): 3051-3073.

[2] 3GPP. NR; Multiplexing and channel coding: 3GPP TS 38.212 [S/OL]. (2022-09-21)[2024-05-29]. https://itecspec.com/archive/3gpp-specification-ts-38-212/.

[3] TAL I, VARDY A. List decoding of polar codes[J]. IEEE Transactions on Information Theory, 2015, 61(5): 2213-2226.

[4] ALAMDAR-YAZDI A, KSCHISCHANG F R. A simplified successive-cancellation decoder for polar codes[J]. IEEE Communications Letters, 2011, 15(12): 1378-1380.

[5] SARKIS G, GROSS W J. Increasing the throughput of polar decoders[J]. IEEE Communications Letters, 2013, 17(4): 725-728.

[6] SARKIS G, GIARD P, VARDY A, et al. Fast polar decoders: Algorithm and implementation[J]. IEEE Journal on Selected Areas in Communications, 2014, 32(5): 946-957.

[7] HASHEMI S A, CONDO C, GROSS W J. Simplified successive-cancellation list decoding of polar codes[C]//2016 IEEE International Symposium on Information Theory (ISIT). Piscataway: IEEE, 2016: 815-819.

[8] HASHEMI S A, CONDO C, GROSS W J. Fast simplified successive-cancellation list decoding of polar codes[C]//2017 IEEE Wireless Communications and Networking Conference Workshops (WCNCW). Piscataway: IEEE, 2017: 1-6.

[9] HASHEMI S A, CONDO C, GROSS W J. Fast and flexible successive-cancellation list decoders for polar codes[J]. IEEE Transactions on Signal Processing, 2017, 65(21): 5756-5769.

[10] HANIF M, ARDAKANI M. Fast successive-cancellation decoding of polar codes: Identification and decoding of new nodes[J]. IEEE Communications Letters, 2017, 21(11): 2360-2363.

[11] ARDAKANI M H, HANIF M, ARDAKANI M, et al. Fast successive-cancellation-based decoders of polar codes[J]. IEEE Transactions on Communications, 2019, 67(7): 4562-4574.

[12] CONDO C, BIOGLIO V, LAND I. Generalized fast decoding of polar codes[C]//2018 IEEE Global Communications Conference (GLOBECOM). Piscataway: IEEE, 2018: 1-6.

[13] ZHENG H T, HASHEMI S A, BALATSOUKAS-STIMMING A, et al. Threshold-based fast successive-cancellation decoding of polar codes[J]. IEEE Transactions on Communications, 2021, 69(6): 3541-3555.

[14] WANG T, QU D M, JIANG T. Parity-check-concatenated polar codes[J]. IEEE Communications Letters, 2016, 20(12): 2342-2345.

[15] PARK J, KIM I, SONG H Y. Construction of parity-check-concatenated polar codes based on minimum Hamming weight codewords[J]. Electronics Letters, 2017, 53(14): 924-926.

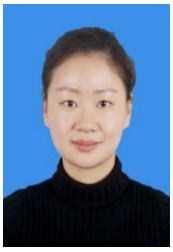
[16] ZHOU Y C, ZHENG Y J, WANG Z F. Fast successive-cancellation decoding of 5G parity-check polar codes[J]. IEEE Communications Letters, 2021, 27(1): 37-40.

[17] HiSilicon. 3GPP TSG RAN WG1 meeting AH NR R1-1700088: Summary of Polar Code Design for Control Channels[R]. Spokane: HiSilicon, 2017.

[18] BALSER M, SILVERMAN R A. Coding for constant-data-rate systems-part II multiple-error-correcting codes[J]. Proceedings of the IRE, 2007, 43(6): 728-733.

作者简介

李俊毅　男,1999年4月出生于陕西省西安市. 现为西安电子科技大学硕士研究生. 主要研究方向为5G中的信道编码与调制技术.
E-mail: 2197438914@qq.com



邢莉娟　女,1982年9月出生于陕西省宝鸡市. 现为西安电子科技大学副教授. 主要研究方向为信道编码与调制技术、量子信息论.
E-mail: ljxing@mail.xidian.edu.cn

李　卓　男,1980年4月出生于陕西省西安市. 现为西安电子科技大学教授. 主要研究方向为量子计算、量子信息论、5G中的编码调制技术.
E-mail: lizhuo@xidian.edu.cn