分类号： TN911.2　　单位代码： 10335

学　　号： 22031097

浙江大学

硕士学位论文

中文论文题目： 极化码的高效译码算法研究

英文论文题目： Research on Efficient Decoding Algorithms for Polar Codes

申请人姓名： 陆旸

指导教师： 赵民建 教授

专业名称： 信息与通信工程

研究方向： 信道编译码

所在学院： 信息与电子工程学院

论文递交日期　2023 年 1 月 13 日

极化码的高效译码算法研究

论文作者签名： 陆旸

指导教师签名： 赵民建

论文评阅人 1：盲审

评阅人 2：盲审

评阅人 3：盲审

答辩委员会主席：张宏纲 研究员 浙江大学

委员 1：史治国 教授 浙江大学

委员 2：李旻 研究员 浙江大学

委员 3：雷鸣 副研究员 浙江大学

委员 4：赵明敏 副教授 浙江大学

委员 5：

答辩日期 2023 年 3 月 6 日

浙江大学研究生学位论文独创性声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得 **浙江大学** 或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文作者签名: 陆昉　　签字日期： 2023 年 3 月 7 日

学位论文版权使用授权书

本学位论文作者完全了解 **浙江大学** 有权保留并向国家有关部门或机构送交本论文的复印件和磁盘，允许论文被查阅和借阅。本人授权 **浙江大学** 可以将学位论文的全部或部分内容编入有关数据库进行检索和传播，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后适用本授权书）

学位论文作者签名: 陆昉　　导师签名: 赵民建

签字日期： 2023 年 3 月 7 日　　签字日期： 2023 年 3 月 7 日

致谢

时光荏苒，白驹过隙，两年半的研究生生活转眼就要结束了。回顾这两年半的生活，有刚入学时的迷茫，有求知过程中的苦难，有收获后的喜悦，这一切促成了我的蜕变与成长。

首先特别致敬极化码的发明者——Erdal Arıkan 教授，感谢他在信息通信领域作出的划时代的贡献，在此基础上才有本文作出的微薄贡献，他在信息论方面十年如一日的耕耘精神这激励着我去做更加坚实的科研工作，这也成为我求学甚至人生的信条。

然后，特别感谢我的授业导师赵民建教授，赵老师为人谦和、学识渊博，对无线通信的前沿科技和发展演化具有敏锐的洞察力和独到的见解。在我攻读硕士学位期间，赵老师给了我许多关于未来规划的珍贵建议。另外还要感谢我的解惑导师赵明敏副教授，赵老师思维敏捷，指引我的研究方向，传授我许多关于论文写作的技巧，切实地培养了我严谨的写作逻辑，我的每一篇论文的修改和定稿离不开赵老师的心血。另外，非常感谢两位赵老师鼓励我积极面对难关，无论是学术能力还是为人处世方面他们都是我终生学习的榜样。

另外，我要感谢我实验室编码组的窦为龙、丘耿鑫硕士师兄，感谢他们对于我工程项目慷慨的指导，他们丰富的硬件知识与经验帮助我顺利在 FPGA 平台上部署了极化码编译器。感谢课题组同级的刘屹豪硕士、以及赵子伟、蒋宇龙硕士师弟，和他们的学术讨论开拓了我的视野，使我受益良多。感谢同实验室的陈梓健、黄哲、胡智祥博士，以及罗群平、邱伏韬、郑宇涛、杨蕊、武梦雨硕士，还有诸多实验室同窗，不一而足，感谢你们的陪伴。另外，特别感谢李旻、刘安老师，以及徐凯迪、刘彦桢、刘耀、胡棋昱、韦逸博士在我博士申请期间给予的珍贵建议和帮助。

回想三年前踏入求是园，我怀着热切追求真理的初心而来，研究带给我探索上的喜悦自不必说，而研究道路中遇到更多是挫折与失败，研究成果与他人相仿而作废，屡次论文发表不顺利，这其中的孤独、沉郁、苦闷仿佛已成为稀疏平常。如今我跨过了诸多迷茫时期，并笃定了继续深造的意愿，这段曲折的经历赋予我无与伦比勇气去面对未来科研以及人生道路上的难题。我要将最诚挚的谢意献给我的父母，感谢他们无时无刻关心我的生活起居，感谢他们在我人生每个重要节点提供的建议，感谢他们感同身受地忧心我求学道路上的挫折，在我低落时期无条件的鼓励和支持，感谢他们十多年来对我教育一如既往的重视，惭愧的是我无以为报，唯有再接再厉不负你们的期待。

未来求学之路漫漫，最后在毕业之际，我提出三点愿景：希望我仍能够怀抱初心，以我所学为科技创新作出绵薄之力；希望关爱我和我关爱的所有人身心健康、平安喜乐；希望所有学海扬帆破浪的硕博生研途顺遂，祝福我的祖国科技创新、繁荣昌盛。

陆旸

2023 年 3 月于杭州

摘要

信道编码是无线通信系统中最重要的技术之一。极化码作为唯一一种被理论证明能够达到信道容量的编码方案，已经成为目前最有前景的研究方向之一。然而，极化码的译码算法存在两个主要缺陷。首先，极化码最常用的连续消除列表（Successive Cancellation List, SCL）译码不能在实际码长和列表大小下提供很好的纠错性能。其次，基于连续消除（Successive Cancellation, SC）的译码具有逐比特译码的顺序性质，导致译码时延较高，这阻碍了极化码在低延迟通信场景的应用。因此，面向高可靠、低时延的通信场景，研究高效的极化码译码算法具有重要意义。针对上述问题，本文对极化码的高效译码算法展开了研究。

首先，本文详细阐述了极化码的基本原理，主要包括极化码编码和译码两个部分。其中极化码的编码部分介绍了信道极化原理、极化码构造算法和极化码的编码过程。对于极化码的译码，本文分别介绍了 SC 和 SCL 译码算法，并着重介绍了基于比特翻转的 SC/SCL 译码算法和快速 SC/SCL 译码算法。

其次，为了进一步提高 SCL 译码算法的性能，本文提出了适用于 SCL 译码的动态比特翻转策略。比特翻转策略是在初始译码失败后进行多次重新译码尝试，并在每次重新译码尝试过程中翻转初始译码结果，从而纠正初始译码中的错误。首先，本文基于路径度量（Path Metric, PM）提出了一种翻转度量，用以确定更容易发生 SCL 译码错误的比特位置。然后将所提出的翻转度量与动态比特翻转策略相结合，设计了一种动态 SCL 翻转（Dynamic SCL Flip, D-SCL-Flip）译码器，该译码器基于所提出的翻转度量动态地构建一系列候选的翻转比特位置，并保证每次重新译码尝试中选取的翻转位置纠正初始 SCL 译码错误的概率最大。仿真结果表明，与最先进的 SCL-Flip 译码器相比，所提出的 D-SCL-Flip 译码器能够减少 34.6% 的重新译码尝试的平均次数，并提供 0.1dB 的性能增益。此外，考虑到实际的硬件实现，本文对 D-SCL-Flip 译码器的不同方面进行了进一步优化，包括简化翻转度量的计算、利用深度学习（Deep Learning, DL）辅助译码和应用快速译码技术降低译码时延。所有这些优化方法都不会造成很大的性能退化，同时可以实现更低的计算复杂度和译码时延。

最后，针对极化码顺序性 SC 译码算法时延较高的问题，本文提出了一种快速 SC 译码算法，可以在某些具有特定信息和冻结比特模式的特殊节点处实现并行译码，从而可以简化并加速 SC 译码过程。首先本文提出了一种新的特殊节点类型，它由一串速率一

或单奇偶校验（Sequence of Rate-One or Single-Parity-Check, SR1/SPC）节点序列组成。该特殊节点常常分布于高码率的极化码中，并且能够包括许多现有的特殊节点类型。随后，本文分析了特殊节点中冻结比特引起的奇偶性约束，并基于这些约束条件为 SR1/SPC 节点设计了一种通用的快速译码算法，进一步提高了 SC 译码的并行度。仿真结果表明，所提出的 SR1/SPC 节点的译码算法可以取得接近最大似然（Maximum-Likelihood, ML）译码的性能。利用该译码算法得到的快速 SC 译码器，相比最先进的快速 SC 译码器可以节省 43.8% 的译码时延，显著提高了 SC 译码的吞吐量。

关键词：极化码，串行抵消译码，比特翻转，深度学习，特殊节点，快速译码

Abstract

Channel coding is one of the most important technologies in the wireless communication systems. Polar codes, as the only coding scheme that are therotically proven to achieve the channel capacity, have become one of the most promising research directions. However, there are two main drawbacks associated with the polar decoding algorithms. First, the popular successive cancellation list (SCL) decoding falls short in providing a reasonable error-correction performance for practical moderate code lengths and list size. Second, the sequential bit-by-bit nature of SC-based decoding leads to high decoding latency and low throughput in terms of hardware implementation, which hinders its application in low-latency communication senarios. Therefore, it is meaningful to investigate high-efficiency and low-latency decoding algorithms for polar codes under ultra-reliable and low-latency communication senarios. Aiming at tackling the aforementioned issues, this thesis develops efficient decoding algorithms for polar codes.

First, aiming at improving the decoding performance of SCL decoder, this paper focus on advancing the framework of dynamic bit-flipping strategy to make it feasible for the SCL decoder. The principle of bit-flipping is to flip the decoding output in additional re-decoding attempts in case the initial decoding fails. A flipping metric from the path metric (PM) domain is obtained to determine the bit positions that are more prone to decoding errors. The proposed flipping metric is then combined with the dynamic bit-flipping strategy to devise a new decoder, referred to as the dynamic SCL-Flip (D-SCL-Flip) decoder, which dynamically builds a set of candidate flipping bits based on the proposed flipping metric, such that the new decoding attempts are performed by adopting the flipping bits that has the highest probability of correcting the decoding errors. Simulation results show that the proposed D-SCL-Flip decoder is able to outperform the state-of-the-art, by providing better error-correcting performance (0.1 dB) yet with fewer re-decoding attempts (34.6%). Furthermore, for ease of practical implementation, several simplifications on different aspects of the D-SCL-Flip decoder are presented, including the calculation of the flipping metric, deep learning (DL) aided decoding, and fast decoding techniques. All these simplifications are shown to have minimal impact on the error-correcting performance, while much lower computational complexity and decoding latency are permitted.

Second, since the sequential nature of the SC algorithm leads to significant decoding latencies, this paper propose fast SC decoding algorithms by implementing parallel decoders at the intermediate levels of the SC decoding tree for some special nodes with specific information and frozen bit patterns, such that SC decoding process can be accelerated. First, this paper present a new class of special nodes composed of a sequence of rate one or single-parity-check (SR1/SPC) nodes, which can be easily found especially in high-rate polar code and is able to envelop a wide variety of existing special node types. Then, the parity constraints caused by the frozen bits in each descendant node are analysed. Based on these parity constraints, a generalized fast decoding algorithm is proposed to decode SR1/SPC nodes efficiently, such that the parallelism of SC decoding is further improved. Simulation results show that the proposed decoding algorithm of the SR1/SPC node can achieve near-maximum-likelihood (ML) performance, and the overall decoding latency can be significantly reduced by 43.8%, as compared to the state-of-the-art fast SC decoder.

Keywords: Polar code, successive cancellation decoding, bit-flipping, deep learning, special nodes, fast decoding

目录

图目录

缩略语表

缩略语	英文解释	中文解释
5G	The 5th Generation Mobile Networks	第五代移动通信系统
ACS	Add-Compare-Sort	加法、比较和排序
Adam	Adaptive Moment Estimation	自适应矩估计
AWGN	Additive White Gaussian Noise	加性高斯白噪声
B-DMC	Binary-Input Discrete Memoryless Channel	二进制输入离散无记忆信道
BEC	Binary Erasure Channel	二进制擦除信道
BP	Belief Propagation	置信传播
BP-Flip	BP Flip	BP 翻转
BPL	BP List	BP 列表
BPSK	Binary Phase Shift Keying	二进制相移键控
CA-SCL	CRC-Aided SCL	CRC 辅助的 SCL
CNN	Convolutional Neural Network	卷积神经网络
CRC	Cyclic Redundancy Check	循环冗余校验
DL	Deep Learning	深度学习
D-SC-Flip	Dynamic SC Flip	动态 SC 翻转
D-SCL-Flip	Dynamic SCL Flip	动态 SCL 翻转
eMBB	Enhanced Mobile Broadband	增强型移动宽带
Fast-D-SCL-Flip	Fast Dynamic SCL Flip	快速的动态 SCL 翻转
Fast-SSC	Fast Simplified SC	快速的简化 SC
Fast-SSCL	Fast Simplified SCL	快速的简化 SCL
FER	Frame Error Rate	误帧率
G-PC	Generalized PC	广义的 PC
G-REP	Generalized REP	广义的 REP

LDPC	Low-Density Parity-Check	低密度奇偶校验
LLR	Log-Likelihood Ratio	对数似然比
LP	Linear Program	线性规划
LSTM	Long Short Term-Memory	长短期记忆
ML	Maximum-Likelihood	最大似然
PG	Policy Gradient	策略梯度
PM	Path Metric	路径度量
P-PC	Parallel Parity Constraint	平行对偶约束
Rate-0	Rate-Zero	码率零
Rate-1	Rate-One	码率一
REP	Repetition	重复
SC	Successive Cancellation	串行抵消
SCAN	Soft Cancellation	软抵消
SC-Flip	SC Flip	SC 翻转
SCL	Successive Cancellation List	串行抵消列表
SCL-Flip	SCL Flip	SCL 翻转
SCL-Flip-ω	Generalized SCL Flip	广义的 SCL 翻转
SISO	Soft In Soft Out	软输入软输出
SNR	Signal-To-Noise Ratio	信噪比
SPC	Single Parity Check	单奇偶校验
S-PC	Segmental Parity Constraint	分段对偶约束
SR0/REP	Sequence Rate-0 or REP	序列 Rate-0 或 REP
SR1	Sequence Rate-1	序列 Rate-1
SSC	Simplified SC	简化 SC
SSPC	Sequence SPC	序列 SPC
URLLC	Ultra-Reliable Low-Latency Communication	超可靠低时延通信

第 1 章 绪论

1.1 研究背景及意义

极化码[1]作为目前唯一一种被理论证明可达到香农容量的信道编码方案，近年来受到了学术界和工业界的广泛关注。在最新的第五代移动通信标准（The 5th Generation Mobile Networks, 5G）[2]中，极化码被应用于增强型移动宽带（Enhanced Mobile Broadband, eMBB）场景下的控制信道。如何进一步提高现有极化码译码技术的误帧率（Frame Error Rate, FER）性能、降低译码时延是一个挑战性的难题。

在实际场景中，译码算法对极化码的纠错能力有极大影响，极化码的原生译码算法——串行抵消（Successive Cancellation, SC）译码[1]采用逐比特译码方式，有效地利用了极化效应，具有良好的纠错性能。然而，SC 译码算法有两个主要缺陷。一方面，根据极化理论，SC 译码下的极化码只有在码长趋于无穷大时才能达到信道容量，因此在实际中，SC 译码无法为有限码长的极化码提供理想的纠错性能。SC 列表（Successive Cancellation List, SCL）译码[3]通过维护一个列表保留了多个最可靠的码字序列，大大提升了 SC 译码的纠错性能。通过将极化码与循环冗余校验（Cyclic Redundancy Check, CRC）相连接，CRC 辅助的 SCL（CRC-Aided SCL, CA-SCL）译码器[4]提供的译码性能可以接近最大似然（Maximum-Likelihood, ML）译码器，这使得极化码成为低密度奇偶校验（Low-Density Parity-Check, LDPC）码[5]和 Turbo 码[6]等其他目前最先进的信道编码方案的有力竞争者[7]。然而，CA-SCL 译码器需要很大的列表数才能取得很好的纠错性能，在实际应用会导致高昂的计算成本和存储资源[8]。因此，如何在合理的硬件复杂度条件下获得更好译码性能一直是热点研究课题。另一方面，SC 译码的逐位顺序性会引起很高译码时延和低吞吐性能，阻碍了其在超可靠低时延通信（Ultra-Reliable Low-Latency Communication, URLLC）等低时延通信场景中的应用。为了实现高效译码，一类普遍的思路就是并行化译码过程，即在译码树中间节点层面而非叶节点层面译码，从而避免了对整个码树的遍历而提高了译码速度。为了保证译码性能不退化，节点层面的译码需要考虑节点中信息和冻结比特的分布特性带来的约束条件，使得译码结果有效。而高度并行化的特殊节点往往包含大量比特，其包含的各自复杂的约束条件给性能无损的译码算法的实现带来了困难。为了达到高纠错性能和低时延的目标，本文针对极化码的译码算

法展开了相关研究。

1.2 国内外研究现状

1.2.1 极化码高性能译码算法研究现状

极化码通过规则的蝶形编码结构在源信息位之间引入了相关性，这种相关性使得码字中每一位都依赖于它前面的位，而 SC 译码算法在译码过程中正是充分利用了这种相关性，即译码时按照自然顺序对发送比特进行逐级判决译码，其中先判定的比特需作为可靠信息参加后续比特的判决。然而，有限码长的极化码在串行抵消译码算法下的性能是次优的，且具有较高的译码延迟和较低的输出效率。以 SC 译码算法为基础，文献 [3] 基于路径度量（Path Metric, PM）提出了 SCL 译码算法，大大提升了 SC 的译码性能，并且在高信噪比（Signal-To-Noise Ratio, SNR）场景下取得了逼近 ML 译码的性能。文献 [4] 提出了 CA-SCL 译码算法，相比传统的 SCL 译码取得了显著性能增益。文献 [9] 提出了在对数似然比（Log-Likelihood Ratio, LLR）域的 SCL 译码算法，提高了计算稳定性并减少了存储空间。极化码的并行译码方案主要包括置信传播（Belief Propagation, BP）[10]和线性规划（Linear Program, LP）[11]译码算法，可实现并行计算并输出软信息，因此具有很低的时延。然而，极化码因子图上存在着大量的短环，导致极化码在 BP 译码下的性能距离 ML 性能有很大差距。文献 [12] 提出了 BP 列表（BP List, BPL）译码算法，该算法的原理是在多个置换因子图上实现并行的 BP 译码，提升了 BP 译码的性能。针对软输入软输出（Soft In Soft Out, SISO）系统，文献 [13] 提出了能输出软信息的软抵消（Soft Cancellation, SCAN）译码算法，具有 SC 串行调度和 BP 消息更新规则的双重特性，尽管 SCAN 译码性能优于 BP 译码，其时延却和 SC 译码相近。

随着对于极化码的译码算法研究更加的深入，一种新的基于比特翻转方案提升译码性能被提出。文献 [14] 提出了 SC 翻转译码（SC Flip, SC-Flip），这是一种低复杂度的 SC 算法变种，能够取得和列表数量较小的 SCL 译码近似的译码性能，该算法通过在额外的 SC 重复译码尝试中翻转可靠度较低的比特估计结果，在瀑布区取得了和 SC 相近的平均复杂度。文献 [15-16] 提出了能够翻转多位比特的动态 SC 翻转（Dynamic SC Flip, D-SC-Flip）译码算法，性能逼近 CA-SCL 译码算法。文献 [17] 提出了辅助 SC-Flip 译码算法的关键集，集合由 SC 译码下最容易出现错误的固定比特构成。文献 [18] 提出了 BP 翻转（BP Flip, BP-Flip）译码算法，性能超越了 BPL 译码算法。文献 [19] 提出了 SCL

翻转（SCL Flip, SCL-Flip）译码算法，取得了优于 CA-SCL 的译码性能。文献 [20] 提出了广义的 SCL-Flip（Generalized SCL Flip, SCL-Flip-ω）译码算法，能够在额外的 SCL 重复译码尝试中翻转至多 ω 个比特，使得 SCL-Flip 译码的纠错性能进一步提升。

近年来，深度学习（Deep Learning, DL）凭借其强大的学习能力、处理大数据的天然优势和对领域内专业知识的低要求，在物理层通信中得到了广泛的应用，如调制识别、信道编码和译码、信道估计和信号检测等。极化码的深度学习译码器的研究同样在推进。基于深度展开的方法，文献 [21] 向极化码展开因子图中引入了可训练的权重参数，提高了 BP 译码算法的性能。对于 SC-Flip 译码算法，文献 [22] 利用深度强化学习中的策略梯度（Policy Gradient, PG）法优化了翻转度量，进一步提高了 SC-Flip 译码性能。进一步地，为了提升 SCL-Flip 的译码性能，文献 [23] 采用长短期记忆（Long Short Term-Memory, LSTM）网络代替翻转度量预测 SCL 错误位置。类似地，文献 [24] 利用卷积神经网络（Convolutional Neural Network, CNN）从 BP 译码迭代结果中提取特征并预测翻转位置，并利用模仿学习策略优化了训练过程，提出的深度学习辅助的 BP-Flip 算法能够纠正多处译码错误，从而大大提升了译码性能。

1.2.2 极化码低复杂度译码算法研究现状

SC 算法由于串行调度和按位判决，节点更新需要之前译码判决结果，带来了高译码时延和计算复杂度。文献 [25] 借鉴 LDPC 码译码器的研究，使用半并行架构和最小和算法，降低了 SC 算法硬件复杂度和计算复杂度。在文献 [26] 中，作者发现当信息和冻结比特的排列存在某种特殊模式时，SC 译码可以被大大简化，并基于此提出了简化 SC（Simplified SC, SSC）译码算法。这些包含特定模式的极化码子码被称为特殊节点，SSC 译码考虑了码率零（Rate-Zero, Rate-0）节点和码率一（Rate-One, Rate-1）两种类型的特殊节点，对这些节点的译码在节点层面进行，从而避免了对整个码树的遍历而提高了译码速度。文献 [27] 发现重复（Repetition, REP）节点和单奇偶校验（Single Parity Check, SPC）节点，提出了快速的简化 SC（Fast Simplified SC, Fast-SSC）译码算法，进一步提高了 SSC 译码的吞吐量。文献 [28] 为五种新类型的特殊节点——Type I-V 节点提出了相应的快速译码算法，继续推动了快速 SC 译码的研究。

然而，上述关于特殊节点的工作需要为每类节点设计定制的译码器，这不可避免地增加了硬件实现的复杂度。在文献 [29] 中，作者提出了广义 REP（Generalized REP, G-REP）节点和广义奇偶校验（Generalized PC, G-PC）节点，此类节点具有通用的结构，

能够包括上述部分特殊节点类型，并具有更高的并行度以进一步减少 SC 译码的时延。文献 [30] 提出了一类序列 Rate-0 或 REP（Sequence Rate-0 or REP, SR0/REP）节点，它提供了对大多数现有特殊节点的统一描述。通过引入 SR0/REP 节点，作者提出了一种广义的快速 SC 译码算法，硬件部署证明其能够在不降低纠错性能的情况大大提高 SC 译码的吞吐量。

此外，上述特殊节点的识别和利用也扩展到 SCL、SC-Flip 和 SCAN 译码中[31-36]，其中文献 [31] 提出了简化的 SCL（Simplified SCL, SSCL）译码算法，分别为 Rate-0、Rate-1、REP 和 SPC 四类特殊节点提供了对应的 SCL 译码算法。然而对于 Rate-1 和 SPC 节点，文献 [31] 所提译码算法会有性能损失，为了解决性能退化的问题，文献 [32] 基于球状译码为 Rate-1 和 SPC 节点提供了无损的 SCL 译码方法。在文献 [33] 中，作者发现 Rate-1 和 SPC 节点具有特殊性质，使得译码时延的复杂度限制到列表长度而非原先的节点长度，提出的快速 SSCL（Fast SSCL, Fast-SSCL）译码算法进一步降低了 SSCL 的译码时延。文献 [34] 为 Type I-V 五种特殊节点提供了相应的快速 SCL 和 SC-Flip 译码算法。

1.3 论文的主要内容和结构安排

本文旨在设计高纠错性能、低复杂度的极化码译码算法，对极化码 SC/SCL 译码相关技术展开研究，主要包括

- 提出了一种动态的 SCL 翻转译码算法，并在译码性能、计算复杂度和译码时延方面对于所提的译码算法作出一系列优化；
- 提出了一类新的特殊节点，并基于此提出了快速的 SC 译码算法。

本文改进了 D-SC-Flip 译码算法，使其适用于 SCL 译码。首先基于路径度量推导出一个翻转度量，用于确定更容易发生译码错误的比特位置。然后，本文基于提出的翻转度量动态构建了一组候选翻转集合，并提出了新的基于比特翻转的 SCL 译码器，称为动态 SCL 翻转（Dynamic SCL Flip, D-SCL-Flip）译码器。仿真结果表明，所提出的 D-SCL-Flip 译码器能够提供更好的纠错性能和更少的重新译码次数。此外，为了便于硬件实现，本文对 D-SCL-Flip 译码器的不同方面进行了一些简化，包括翻转度量的计算、深度学习辅助译码和特殊节点的快速译码，所有这些简化能够大大降低译码的计算复杂度和时延。

D-SCL-Flip 译码器中移动度量涉及的指数/对数运算的数量随着信息比特数和列表

大小线性增长，导致很高的计算复杂度。本文针对简化的翻转度量进一步设计了定制的 DL 网络以提高所提出的翻转度量的性能。所提出的深度学习辅助的翻转度量不包含超越函数，因此比现有的翻转度量更利于硬件实现。仿真结果表明，与现有技术相比，所提出的 DL 辅助的 D-SCL-Flip 译码具有更好的纠错性能和更低的复杂度。

SC 译码算法的顺序特性引起了很高的译码时延。快速 SC 译码能够通过在 SC 译码树的中间层为一些具有特定信息和冻结位分布的特殊节点实施并行译码来加速 SC 译码过程。为了进一步提高 SC 译码的并行度，本文提出了一类由一个源节点和一串速率一或单偶校验子节点序列组成的新的特殊节点（Sequence Rate-1 or SPC, SR1/SPC），它包含了多种现有的特殊节点类型且更容易在高速率极化码中被发现。然后，为了保证无损的译码性能，本文分析了所提特殊节点中冻结比特引起的对偶约束条件，得出了节点译码得到的码字需要满足的两类对偶约束的具体形式。最后，本文针对 SR1/SPC 节点提出了一种广义的快速 SC 译码算法，算法保证满足了节点的两类对偶约束条件。仿真结果表明，与目前文献中最先进的快速 SC 译码算法相比，所提出的译码算法能够大大降低译码时延且不会造成性能退化。

1.4 结构安排

本文共包含五章内容，每章的具体内容如下：

第一章为绪论。该章首先阐述了信道编码在通信系统中的背景和意义。然后对信道编码的相关研究和发展历史做了详细的说明。接着介绍了极化码编译码目前已有的研究成果，下面对本文的主要研究内容和结构安排进行了阐述。

第二章首先阐述了极化码的基本原理。对于极化码，该章首先介绍了信道极化的基本原理，然后分别给出了极化码构造与编码的原理及方案，最后介绍了几种常用的极化码的译码算法过程，包括：SC 译码算法，SCL 译码算法、基于比特翻转的 SC/SCL 译码算法和快速 SC/SCL 译码算法。

第三章介绍了提出的动态 SCL 翻转译码算法。首先提出了基于 PM 的翻转度量；然后将所提翻转度量与比特翻转策略相结合，提出了动态 SCL 翻转译码算法；接下来分别介绍了对于所提译码算法的三种优化方法，包括简化翻转度量的计算、利用 DL 辅助译码提升性能、应用快速译码技术降低时延；最后通过仿真实验比较了优化的译码算法与各种基准译码算法的性能。

第四章详细阐述了基于序列节点的快速 SC 译码算法。首先提出了一种新的特殊节点类型，并分析了该节点包含的各种对偶约束条件；基于这些对偶约束，提出了高效的译码算法用以并行译码该特殊节点；最后仿真结果表明通过应用本文提出的特殊节点和译码算法可以大大降低 SC 的译码时延。

第五章是对本文研究内容进行了总结，然后对相关的研究中的潜在创新点做了简单的阐述。

第 2 章 基本原理

2.1 引言

本章主要介绍极化码的基本原理，包括极化码的编码，极化码常用的译码算法、基于比特翻转的译码算法以及快速译码算法。在极化码编码部分，本文首先介绍了信道极化的原理，随后介绍了极化码的 GA 构造算法和编码过程。在极化码译码部分，本文简要介绍了 SC 和 SCL 译码算法，在此基础上接着详细介绍了基于比特翻转的 SC/SCL 译码算法，最后介绍了针对 SC 和 SCL 的快速译码算法。本章节的内容是后续章节必要的基础知识。

在进行极化码相关理论介绍之前，本文首先对下文使用的符号做如下定义：标量、向量和矩阵分别用小写字母、粗体小写字母和粗体大写字母表示。对于任意向量 $\boldsymbol{a}$，$a[i:k:j]$ 表示 $\boldsymbol{a}$ 的子向量，该子向量由 $(a[i], a[i+k], \ldots, a[i+mk])$ 构成，其中 k 是步长，$m = \lfloor (j-i)/k \rfloor$。如果 $k = 1$，$a[i:k:j]$ 简单地写为 $a[i:j]$。$\mathrm{sgn}(a)$ 表示标量 a 的符号，$\min(\boldsymbol{a})$ 返回向量 $\boldsymbol{a}$ 中的最小元素。此外，$\boldsymbol{I}_N$ 和 $\boldsymbol{0}_N$ 分别表示 $N \times N$ 的单位矩阵和 $N \times N$ 的全零矩阵。对于矩阵 $\boldsymbol{A}$，$(\boldsymbol{A})_i$ 表示其第 i 列的向量。此外，$\{\cdot\}$ 表示一个集合，$\lceil \cdot \rceil$ 和 $\lfloor \cdot \rfloor$ 分别表示向上舍入和向下舍入，操作 $\oplus$ 表示按位 XOR 运算，$\otimes$ 表示克罗内克积，$\boldsymbol{A}^{\otimes n}$ 表示 $\boldsymbol{A}$ 的第 n 次克罗内克幂。最后，$\mathcal{P}(N, K)$ 表示一个码长为 $N = 2^n$、信息比特数量为 K 的极化码，其中码率定义为 $R = K/N$。

2.2 极化码编码

2.2.1 信道极化基本原理

信道极化包括信道合并和信道分裂两个部分，经过信道极化后，N 组独立的二进制离散无记忆信道 W 可以转化为一组分裂子信道。当 N 无限大时，分裂子信道的信道容量要么趋近于 0，要么趋近于 1。信道容量趋向 1 的好的子信道用于传输信息比特，而信道容量趋向 0 的坏的子信道用于传输固定值的冻结比特，这样可以实现整体信道容量的逼近。

2.2.1.1 信道合并

信道合并是信道极化现象的第一个阶段，指的是 N 个独立的二进制输入离散无记忆信道（Binary-Input Discrete Memoryless Channel, B-DMC）信道 W 通过递归合成的新的信道 $W_N : \mathcal{X}^N \to \mathcal{Y}^N$，其中 $\mathcal{X}^N$ 为输入字符集，$\mathcal{X}^N$ 为输出字符集。信道合并的递归流程如下：在第 1 层，如图2.1所示，信道 W 的输入和信道 W_2 的输入之间的关系为

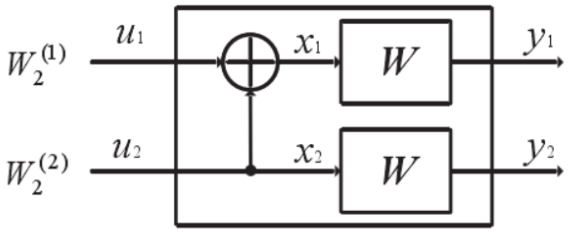


图 2.1 第 1 层的信道合并

$x_1 = u_1 \oplus u_2, x_2 = u_2$，其矩阵形式可以表示为：

$$x_1^2 = u_1^2 \mathbf{G}_2, \tag{2-1}$$

其中 $\mathbf{G}_2 = \left[\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right]$，合成信道 W_2 可以通过如下公式得到：

$$W_2(y_1, y_2|u_1, u_2) = W(y_1|u_1 \oplus u_2)W(y_2|u_2). \tag{2-2}$$

在第 2 层，信道合并方式如图2.2所示，x_1^4 和 u_1^4 间的关系为：

$$x_1^4 = u_1^4 \mathbf{G}_4 = u_1^4 \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}, \tag{2-3}$$

两个独立的 W_2 通过如下公式合成 W_4:

$$W_4(y_1^4|u_1^4) = W(y_1^2|u_1 \oplus u_2, u_3 \oplus u_4)W(y_3^4|u_2, u_4). \tag{2-4}$$

以此类推至 N 个信道的合成，即 W_N 由两个独立的 $W_{N/2}$ 合成，一般情况下 W_N 的递归公式可以表示为：

$$W_N(y_1^N|u_1^N) = W(y_1^{N/2}|u_{1,o}^N \oplus u_{1,e}^N)W(y_{N/2+1}^N|u_{1,e}^N) = W^N(y_1^N|u_1^N \mathbf{G}_N), \tag{2-5}$$

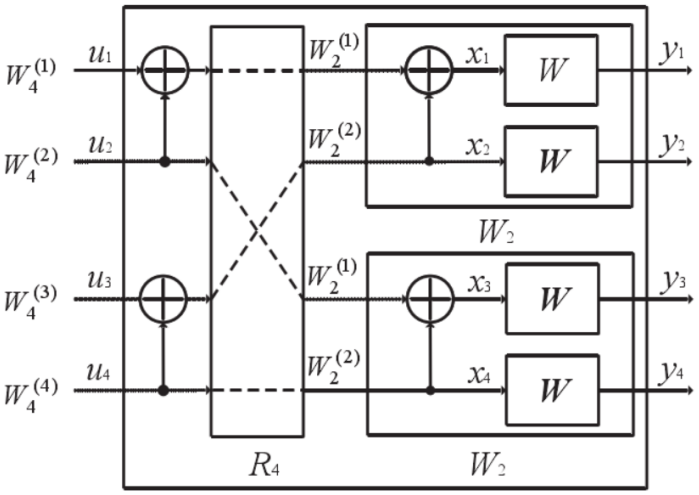


图 2.2 第 2 层的信道合并

其中，$\mathbf{G}_N$ 为生成矩阵，可以通过 $\mathbf{G}_2$ 的克罗内克积运算构造：

$$\mathbf{G}_N = \mathbf{B}_N \mathbf{G}_2^{\otimes n}, \tag{2-6}$$

其中 $\mathbf{B}_N$ 为比特置换矩阵。

2.2.1.2 信道分裂

信道分裂使 W_N 分裂为 N 个极化信道 $W_N^{(i)}$，信道 $W_N^{(i)}\left(\mathbf{y}_1^N, \mathbf{u}_1^{i-1} \mid u_i\right)$ 的输入是 u_i，输出是 y_1^N 和 u_1^{i-1}，其转移概率定义如下：

$$\begin{aligned} W_N^{(i)}\left(\mathbf{y}_1^N, \mathbf{u}_1^{i-1} \mid u_i\right) &= \Pr\left(\mathbf{y}_1^N, \mathbf{u}_1^i\right) / \Pr\left(u_i\right) \\ &= 2 \sum_{u_{i+1}^N} \Pr\left(\mathbf{y}_1^N, \mathbf{u}_1^N\right) \\ &= \frac{2}{\Pr\left(\mathbf{u}_1^N\right)} \sum_{u_{i+1}^N} \Pr\left(\mathbf{y}_1^N \mid \mathbf{u}_1^N\right) \\ &= \frac{1}{2^{N-1}} \sum_{u_{i+1}^N} \Pr\left(\mathbf{y}_1^N \mid \mathbf{x}_1^N\right) \\ &= \frac{1}{2^{N-1}} \sum_{u_{i+1}^N} \prod_{i=1}^{N} W\left(y_i \mid x_i\right), \end{aligned} \tag{2-7}$$

该式的递归表达式如下所示：

$$
\begin{aligned}
&W_N^{(2i-1)}\left(y_1^N, u_1^{2i-2} \mid u_{2i-1}\right) \\
&=\frac{1}{2}\sum_{u_{2i}} W_{N/2}^{(i)}\left(y_1^{N/2}, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2} \mid u_{2i-1} \oplus u_{2i}\right) W_{N/2}^{(i)}\left(y_{N/2+1}^N, u_{1,e}^{2i-2} \mid u_{2i}\right), \\
&W_N^{(2i)}\left(y_1^N, u_1^{2i-2} \mid u_{2i}\right) \\
&=\frac{1}{2} W_{N/2}^{(i)}\left(y_1^{N/2}, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2} \mid u_{2i-1} \oplus u_{2i}\right) W_{N/2}^{(i)}\left(y_{N/2+1}^N, u_{1,e}^{2i-2} \mid u_{2i}\right),
\end{aligned}
\tag{2-8}
$$

其中 $u_{1,o}^{2i-2}$ 表示 u_1^{2i-2} 中索引为奇数的元素，$u_{1,e}^{2i-2}$ 表示 u_1^{2i-2} 中索引为偶数的元素。

对于 $\mathcal{P}(1024, 512)$ 的极化码，采用巴氏参数构造法[1]计算擦除概率为 0.5 的二进制擦除信道（Binary Erasure Channel, BEC）极化后子信道的信道容量，仿真结果如图2.3所示。其中红色和蓝色圆点分别代表信息比特和冻结比特。从图中可以看出信道极化效应：经过极化后的子信道容量趋近于 0 或趋近于 1。

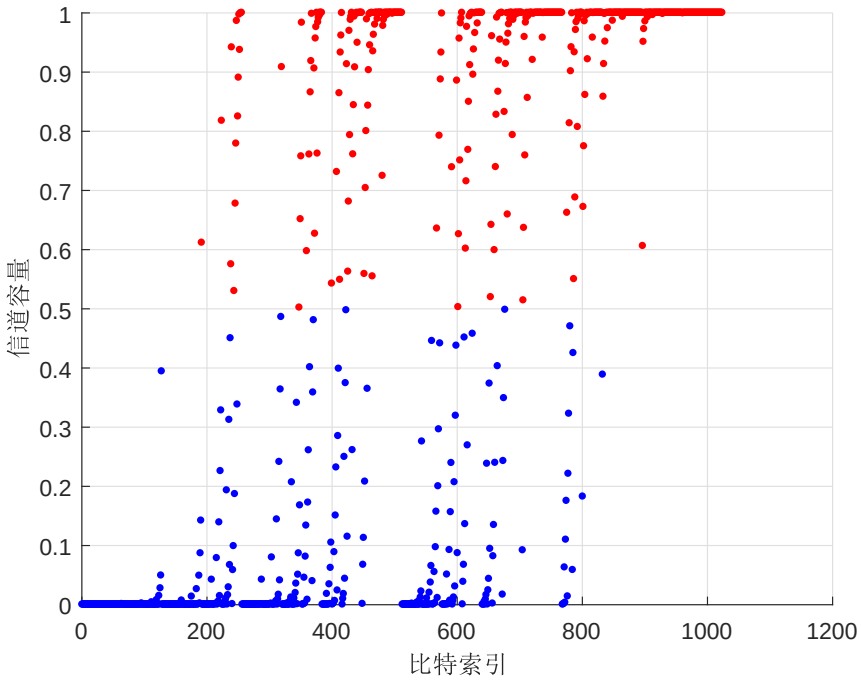


图 2.3 信道极化后子信道的对称容量

2.2.2 极化码的构造

在实际的系统应用中，只能选取有限码长的极化码，而在有限码长条件下，信道极化得不够充分，不仅存在容量趋于 1 或者 0 的信道，还有一些分裂信道的信道容量处于中间值，如何用具体数值去刻画信道容量是极化码的构造问题，即通过极化码的构造描

述出分裂信道的可靠性程度，从而找到信道可靠性高的分裂信道进行信息的传输。目前较主流的方法包括巴氏参数构造法[1]、信道退化和提升构造法[37]、高斯近似（Gaussian Approximation, GA）构造法[38]和极化重量法[39]等。下面，本文介绍了最常见的 GA 构造算法。

假设信道 $W(y|x)$ 是加性高斯白噪声（Additive White Gaussian Noise, AWGN）信道，信息比特 u_1^N 的值都是零，相应地码字 x_1^N 自然也是全零码字。经过二进制相移键控（Binary Phase Shift Keying, BPSK）调制后，即 $s_i = 1 - 2x_i, 1 \le i \le N$，其中 s_i 是第 i 个 BPSK 发送符号，x_i 是第 i 个码字比特，那么 BPSK 发送符号就是全 1 序列。每个 BPSK 接收符号是 $y_i = 1 + n_i, 1 \le i \le N$, 其中 n_i 是独立同分布（Independent Identical Distribution, i.i.d.）、均值为 0、方差为 σ^2 的加性高斯白噪声。因此 y_i 服从 i.i.d. 的高斯分布 $\mathcal{N}(1, \sigma^2)$，那么其 LLR 可以根据下式计算：

$$L_i = \ln \frac{\frac{1}{\sqrt{2\pi}} e^{-(y_i-1)^2/\left(2\sigma^2\right)}}{\frac{1}{\sqrt{2\pi}} e^{-(y_i+1)^2/(2\sigma^2)}} = \frac{2}{\sigma^2} y_i, \tag{2-9}$$

注意到 L_i 服从 $\mathcal{N}\left(\frac{2}{\sigma^2}, \frac{4}{\sigma^2}\right)$，其方差是均值的两倍。GA 的思想是：假设在极化过程 $\left(W_N^{(i)}, W_N^{(i)}\right) \to \left(W_{2N}^{(2i-1)}, W_{2N}^{(2i)}\right)$ 中，三个极化信道 $W_N^{(i)}$、$W_{2N}^{(2i-1)}$ 以及 $W_{2N}^{(2i)}$ 的 LLR 都是方差为均值两倍的高斯随机变量。在这种假设下，极化信道 $W_{2N}^{(2i-1)}$ 和 $W_{2N}^{(2i)}$ 的 LLR 均值可以分别由下式计算得到：

$$\begin{aligned}
L_{2N}^{(2i-1)}\left(y_1^{2N}, u_1^{2i-2}\right) &= \ln \frac{W_{2N}^{(2i-1)}\left(y_1^{2N}, u_1^{2i-2} \mid u_{2i-1} = 0\right)}{W_{2N}^{(2i-1)}\left(y_1^{2N}, u_1^{2i-2} \mid u_{2i-1} = 1\right)} \\
&= 2\tanh^{-1}\left(\tanh \frac{L_N^{(i)}\left(y_1^N, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2}\right)}{2} \tanh \frac{L_N^{(i)}\left(y_{N+1}^{2N}, u_{1,e}^{2i-2}\right)}{2}\right), \\
L_{2N}^{(2i)}\left(y_1^{2N}, u_1^{2i-1}\right) &= \ln \frac{W_{2N}^{(2i)}\left(y_1^{2N}, u_1^{2i-1} \mid u_{2i} = 0\right)}{W_{2N}^{(2i)}\left(y_1^{2N}, u_1^{2i-1} \mid u_{2i} = 1\right)} \\
&= \left(1 - 2u_{2i-1}\right) L_N^{(i)}\left(y_1^N, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2}\right) + L_N^{(i)}\left(y_{N+1}^{2N}, u_{1,e}^{2i-2}\right),
\end{aligned} \tag{2-10}$$

其中 $L_N^{(i)}\left(y_1^N, u_1^{i-1}\right)$ 表示第 i 个极化信道的 LLR。基于高斯近似的假设，即所有 LLR 都是方差为均值两倍的高斯随机变量，公式(2-10)取均值后化简为:

$$\begin{aligned}
m_{2N}^{(2i)} &= 2m_N^{(i)}, \\
m_{2N}^{(2i-1)} &= \phi^{-1}\left(1 - \left(1 - \phi\left(m_N^{(i)}\right)\right)^2\right),
\end{aligned} \tag{2-11}$$

其中 $m_N^{(i)} = E\{L_N^{(i)}\}$ 表示极化信道 LLR 的均值，$\phi(t)$ 的一种简化形式如下所示：

$$\phi(t) = \begin{cases} \mathrm{e}^{-0.4527t^{0.86}+0.0218}, & 0 < t < 10 \\ \sqrt{\frac{\pi}{t}}\mathrm{e}^{-\frac{t}{4}}\left(1-\frac{10}{7t}\right), & t \geq 10 \end{cases}, \tag{2-12}$$

然后根据极化信道的均值可以计算每个子信道的错误概率:

$$P_c(i) = Q\left(\sqrt{m_N^{(i)}/2}\right), \tag{2-13}$$

其中:

$$\begin{aligned} Q(t) &= \frac{1}{2}\operatorname{erfc}\left(\frac{t}{\sqrt{2}}\right) \\ \operatorname{erfc}(t) &= \frac{2}{\sqrt{\pi}}\int_t^{\infty} e^{-t^2}dt. \end{aligned} \tag{2-14}$$

最后选取错误概率最小的 K 个子信道传输信息比特，而剩余的 $N-K$ 个子信道则传输固定值的冻结比特。信息比特和冻结比特的位置记录在集合 $\mathcal{A}$ 和 $\mathcal{A}^c$ 中，通常情况下这些集合可以根据构造算法离线确定，且对于编码端和译码端都是已知的。5G 标准提供了嵌套的极化序列，可以很方便地构造码长不大于 1024 的极化码。本文选择 5G 极化序列和 GA 法作为极化码的构造方法。

2.2.3 极化码的编码

作为一类线性分组码，极化码的编码过程可以完全由生成矩阵 $\mathbf{G}_N$ 所表征，即其编码后的码字可以描述为消息向量 u_1^N 与生成矩阵 $\mathbf{G}_N$ 的乘积：

$$x_1^N = u_1^N \mathbf{G}_N, \tag{2-15}$$

其中 u_1^N 由 K 个信息比特和 $N-K$ 个冻结比特构成，$\mathbf{G}_N$ 的一种置换形式如公式(2-6)所示。$\mathbf{G}_N$ 的另一种非置换形式如图2.4所示，该编码方式没有考虑比特置换矩阵 $\mathbf{B}_N$，实际上是否采用置换矩阵不会对极化码性能造成影响。基于复杂度方面的考虑，5G 标准已经放弃采用置换形式，本文同样默认使用的是非置换图编码方式。

2.3 极化码译码

2.3.1 SC 译码算法

SC 译码算法源于文献 [1]，这是一种译码树的贪心搜索算法，在译码的每一步里只选择最好的结果保留。SC 译码可以认为是译码树的节点间一系列软硬信息的传递过程，

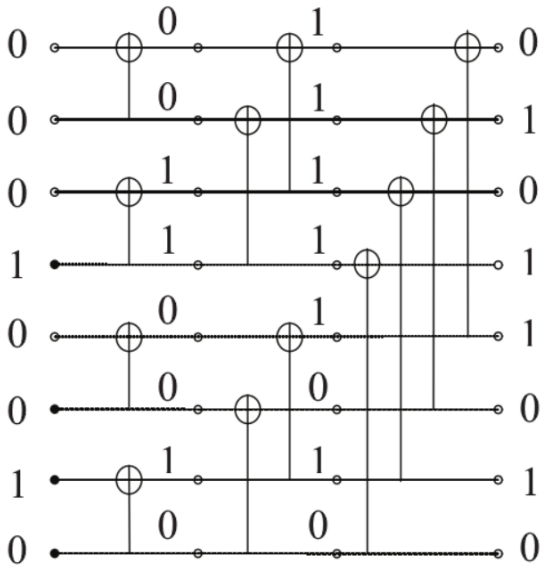


图 2.4 $N = 8$ 极化码的非置换形式编码图

其中软信息指的是节点的 LLR 数据，而硬信息指的是节点的估计码字。特别地，根节点处的软信息使用信道 LLR，定义为 $L_i^n \triangleq \frac{\Pr(y_i|x_i=0)}{\Pr(y_i|x_i=1)}$。通常，在译码树的第 s 层，LLR 向量 $\boldsymbol{L}^s = (L_1^s, L_2^s, \ldots, L_{N_s}^s)$ 由父节点传给其子节点，而码字向量 $\boldsymbol{\beta}^s = (\beta_1^s, \beta_2^s, \ldots, \beta_{N_s}^s)$ 由子节点传给父节点，其中 $0 \le s \le n$。左右子节点的 LLR 向量 $\boldsymbol{L}^l = (L_1^l, L_2^l, \ldots, L_{\frac{N_s}{2}}^l)$ 和 $\boldsymbol{L}^r = (L_1^r, L_2^r, \ldots, L_{\frac{N_s}{2}}^r)$ 可以由下式计算得到[25]：

$$L_i^l = \operatorname{sgn}(L_i^s)\operatorname{sgn}(L_{i+\frac{N_s}{2}}^s)\min(|L_i^s|, L_{i+\frac{N_s}{2}}^s), \quad \forall 1 \le i \le N, \tag{2-16}$$

$$L_i^r = L_{i+\frac{N_s}{2}}^s + (1-2\beta_i^l)L_i^s, \quad \forall 1 \le i \le N. \tag{2-17}$$

类似地，父节点的码字向量 $\boldsymbol{\beta}^s$ 也基于左右子节点的码字向量 $\boldsymbol{\beta}^l = (\beta_1^l, \beta_2^l, \ldots, \beta_{\frac{N_s}{2}}^l)$ 和 $\boldsymbol{\beta}^r = (\beta_1^r, \beta_2^r, \ldots, \beta_{\frac{N_s}{2}}^r)$ 计算得到：

$$\beta_i^s = \begin{cases} \beta_i^l \oplus \beta_i^r, & \text{if } i \le \frac{N_s}{2} \\ \beta_{i-\frac{N_s}{2}}^r, & \text{otherwise} \end{cases}. \tag{2-18}$$

在译码树最底层 $s = 0$，即叶节点处，通过硬判决 L_i^0 估计比特 u_i，如下式表示：

$$\hat{u}_i = \begin{cases} \mathrm{HD}(L_i^0), & \text{if } i \in \mathcal{A} \\ 0, & \text{if } i \in \mathcal{A}^c \end{cases}, \tag{2-19}$$

其中 $\hat{u}_i$ 是 u_i 的估计，$\mathrm{HD}(L_i^0) = \frac{1-\operatorname{sgn}(L_i^0)}{2}$。

2.3.2 SCL 译码算法

为了提升 SC 译码算法的纠错性能，SCL 译码算法不再仅保留一个译码结果，通过估计每个信息比特是 0 或者 1，SCL 译码算法能够实时选择译码成功率最高的 L 条路径保留下来。定义 $\hat{u}_{1,l}^{i}=(\hat{u}_{1,l},\hat{u}_{2,l},\ldots,\hat{u}_{i,l})$ 为译码比特 u_i 时第 l 条路径，其中 $1\leq l\leq L$。为了评估每条路径的可靠度，定义第 l 条路径 $\hat{u}_{1,l}^{i}$ 的后验概率为路径度量 $PM_{i,l}$[9]，即：

$$PM_{i,l}\triangleq-\ln\Pr(\hat{u}_{1,l}^{i}=u_1^i|y_1^N). \tag{2-20}$$

PM 按照下式更新：

$$\begin{aligned}PM_{i,l}=&PM_{i-1,l}+\ln(1+e^{-(1-2\hat{u}_{i,l})L_{i,l}})\\ \overset{(a)}{\approx}&\begin{cases}PM_{i-1,l}, & \text{if } 1-2\hat{u}_{i,l}=\operatorname{sgn}(L_{i,l})\\ PM_{i-1,l}+|L_{i,l}|, & \text{otherwise}\end{cases},\end{aligned} \tag{2-21}$$

其中 $PM_{0,l}=0$，$L_{i,l}\triangleq\log\frac{\Pr(u_i=0|y_1^N,\hat{u}_{1,l}^{i-1})}{\Pr(u_i=1|y_1^N,\hat{u}_{1,l}^{i-1})}$，(a) 表示 PM 的硬件友好型近似[9].

在不失一般性的情况下，假设在译码的每一步产生的 $2L$ 条临时路径按非降序排序，即 $PM_{i,1}\leq PM_{i,2}\leq\cdots\leq PM_{i,2L}$。令 $PM_i\triangleq\{PM_{i,1},PM_{i,2},\ldots,PM_{i,2L}\}$ 和 $\mathcal{L}_i\triangleq\{\hat{u}_{1,1}^{i},\hat{u}_{1,2}^{i},\ldots,\hat{u}_{1,2L}^{i}\}$ 分别表示 PM 列表和所有路径的列表。如公式(2-20)所示，PM 值较小的路径译码正确概率更高。因此，SCL 译码器通过使用以下修剪方案 $h_0(\cdot)$ 从 $2L$ 条临时路径中选择具有最小 PM 值的 L 条路径保留：

$$\mathcal{R}_i=h_0(\mathcal{L}_i)\triangleq\{\hat{u}_{1,1}^{i},\hat{u}_{1,2}^{i},\ldots,\hat{u}_{1,L}^{i}\}, \tag{2-22}$$

其中 $\mathcal{R}_i$ 是保留路径的列表，保留的路径被用于后续译码。相应地，淘汰路径列表定义为 $\mathcal{D}_i=\mathcal{L}_i\setminus\mathcal{R}_i=\{\hat{u}_{1,L+1}^{i},\hat{u}_{1,L+2}^{i},\ldots,\hat{u}_{1,2L}^{i}\}$。当最后一位比特译码完成后，具有最小 PM 的路径被选为译码结果作为最终输出。

CA-SCL 译码器是一种利用 CRC 辅助的改进的 SCL 译码器。具体地，在编码器处，信息比特首先通过 CRC 编码器，将得到的 r 位 CRC 校验比特级联到原信息比特之后，构造 CRC 级联的极化码，这样的极化码由 $\mathcal{P}(N,K+r)$ 表示。在译码器处，当译码完最后一个比特后，SCL 译码器器的 L 条路径被输出到 CRC 检测器，能通过 CRC 校验的路径被认为是正确的译码结果，因此能够通过 CRC 校验并且具有最小 PM 的路径即是 CA-SCL 译码器的最终输出。

2.3.3 基于比特翻转的 SC/SCL 译码算法

基于比特翻转的 SC/SCL 译码器的主要思想是在普通 SC/SCL 译码失败后使用一系列额外的译码尝试来纠正由信道噪声引起的译码错误。在 SC 译码中，有两种类型的译码错误，即信道错误和传播错误[15]。前者仅由信道噪声引起，并且彼此独立，而后者是由顺序 SC 译码期间先前的比特错误而引起。如果纠正了所有信道噪声引起的错误，则产生的传播错误将消失。因此，现有的基于比特翻转的 SC 译码器通过在额外译码尝试中翻转比特估计值试图纠正信道噪声引起的错误。然而，在 SCL 译码算法中，信道噪声对译码过程的影响是不同的。由于 SCL 译码器维护了一个包含许多译码路径的列表，所以可以不论错误类型（信道错误或者传播错误）容忍多个译码错误，而代价是惩罚正确路径的可靠度，即正确路径的 PM 值会增大。如果正确的路径被多次惩罚导致可靠度过低，则它可能在路径剪枝的过程中被消除，从而导致译码结果不正确。

另一方面，现有的基于比特翻转的 SCL 译码器旨在使用不同的比特翻转策略避免正确路径被消除。通常，适用于 SCL 译码的比特翻转策略包括路径翻转策略[40]、移位剪枝策略[41]和反向路径选择策略[19]。将 $\mathcal{E}$ 定义为一个包含 ω 个需要翻转的比特位置的翻转元，即 $\mathcal{E}=\{i_1,\dots i_\omega\}$，其中 $i_1<\cdots<i_\omega$ 且 ω 定义为该翻转元的阶数。若当前位是冻结比特或者 $\mathcal{L}_i$ 的大小不大于 L 时，不需要比特翻转操作，因此翻转元应满足 $\mathcal{E}\subseteq\mathcal{A}^*$，其中 $\mathcal{A}^*=\mathcal{A}\setminus\mathcal{A}_0$ 是可行的翻转比特位置的集合，而 $\mathcal{A}_0$ 表示由 $\mathcal{A}$ 中前 $\log_2 L$ 个比特位置构成的集合。通用比特翻转策略 $H_{\mathcal{E}}(\mathcal{L}_i)$ 的定义如为下式所示：

$$\mathcal{R}_i=H_{\mathcal{E}}(\mathcal{L}_i)=\begin{cases}h_0(\mathcal{L}_i), & \text{if } i\notin\mathcal{E}\\ h(\mathcal{L}_i), & \text{if } i\in\mathcal{E}\end{cases}, \tag{2-23}$$

其中 $h(\cdot)$ 是用于选择 L 条保留路径的广义路径剪枝函数，其具体形式取决于所选择的比特翻转策略。例如，路径翻转策略[40]翻转每个路径中当前比特的估计值；移位剪枝策略[41]通过移动剪枝窗，从而选择具有移位偏移为 k 的 L 条路径：

$$\begin{aligned}H_{\mathcal{E}}(\mathcal{L}_i)&=\begin{cases}h_0(\mathcal{L}_i), & \text{if } i\notin\mathcal{E}\\ h_k(\mathcal{L}_i), & \text{if } i\in\mathcal{E}\end{cases}\\ &=\begin{cases}\{\hat{u}^i_{1,1},\hat{u}^i_{1,2},\dots,\hat{u}^i_{1,L}\}, & \text{if } i\notin\mathcal{E}\\ \{\hat{u}^i_{1,k+1},\hat{u}^i_{1,k+2},\dots,\hat{u}^i_{1,k+L}\}, & \text{if } i\in\mathcal{E}\end{cases},\end{aligned} \tag{2-24}$$

其中 $h_k(\mathcal{L}_i) \triangleq \{\hat{u}^i_{1,k+1}, \hat{u}^i_{1,k+2}, \ldots, \hat{u}^i_{1,k+L}\}$ 表示移位剪枝函数，其能够选择 $k+1$ 到 $k+L$ 这些路径保留；反向路径选择策略可被视为具有移位偏移 kL 的移位剪枝策略的特殊情况，其实际上选择 L 个原本淘汰的路径作为新的保留路径，即选择具有最大 PM 的 L 路径保留，其路径剪枝函数如下式所示：

$$\begin{aligned} H_{\mathcal{E}}(\mathcal{L}_i) &= \begin{cases} h_0(\mathcal{L}_i), & \text{if } i \notin \mathcal{E} \\ h_L(\mathcal{L}_i), & \text{if } i \in \mathcal{E} \end{cases} \\ &= \begin{cases} \{\hat{u}^i_{1,1}, \hat{u}^i_{1,2}, \ldots, \hat{u}^i_{1,L}\}, & \text{if } i \notin \mathcal{E} \\ \{\hat{u}^i_{1,L+1}, \hat{u}^i_{1,L+2}, \ldots, \hat{u}^i_{1,2L}\}, & \text{if } i \in \mathcal{E} \end{cases}. \end{aligned} \tag{2-25}$$

与路径翻转策略相比，反向路径选择策略显示出更好的纠错性能和更低的译码复杂度，因此它被目前最先进的 SCL-Flip 和 SCL-Flip-ω 译码器采用[19-20]。

译码算法方面，基于比特翻转的 SCL 译码器将进行最多 $T+1$ 的译码尝试，包括一次初始 SCL 译码尝试和最多 T 次的重新译码尝试。每次译码中，译码器根据翻转集 $\mathcal{S} = \{\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_T\}$ 中选取的一个翻转元 $\mathcal{E}_t$ 执行比特翻转策略。具体地，SCL-Flip 译码器仅在一个比特位置上进行比特翻转，即仅考虑 1 阶的翻转元，并且根据正确路径消除的概率从 LLR 域导出了如下所示的翻转度量：

$$M_{\text{std}}(i) = \sum_{l=L+1}^{2L} \frac{\prod\limits_{j \in \bar{\mathcal{A}}_l^{(i)}} e^{-|L_{j,l}|}}{\prod\limits_{j \le i, j \in \mathcal{A}} (1 + e^{-|L_{j,l}|})}, \tag{2-26}$$

其中 $\bar{\mathcal{A}}_l^{(i)}$ 代表一个位置集合，它记录了路径 $\hat{u}^i_{1,l}$ 中比特估计值与硬判决结果不同的比特位置。为了降低计算复杂度，文献 [20] 提出了另一个简化的翻转度量：

$$M_{\text{max-log}}(i) = -\sum_{j \in \bar{\mathcal{A}}_{l^*}^{(i)}} |L_{j,l^*}| - \sum_{j \le i, j \in \mathcal{A}} \ln(1 + e^{-|L_{j,l^*}|}), \tag{2-27}$$

其中 $l^* = \arg\max_{l \in \{L+1, L+2, \ldots, 2L\}} \frac{\prod\limits_{j \in \bar{\mathcal{A}}_l^{(i)}} e^{-|L_{j,l}|}}{\prod\limits_{j \le i, j \in \mathcal{A}} (1+e^{-|L_{j,l}|})}$。

在 SCL-Flip-ω 译码器中，每次译码尝试可以翻转最多 ω 位，整个译码过程包括初始的 SCL 译码尝试，然后是分为 ω 个阶段的译码尝试，不同阶段中翻转的位数随着阶数而逐次增加。图2.5提供了 SCL-Flip-ω 译码算法的流程示意图。如图所示，整个译码过程基于一个树结构，树中的每个节点表示一次译码尝试，树由以下两种类型的参数表

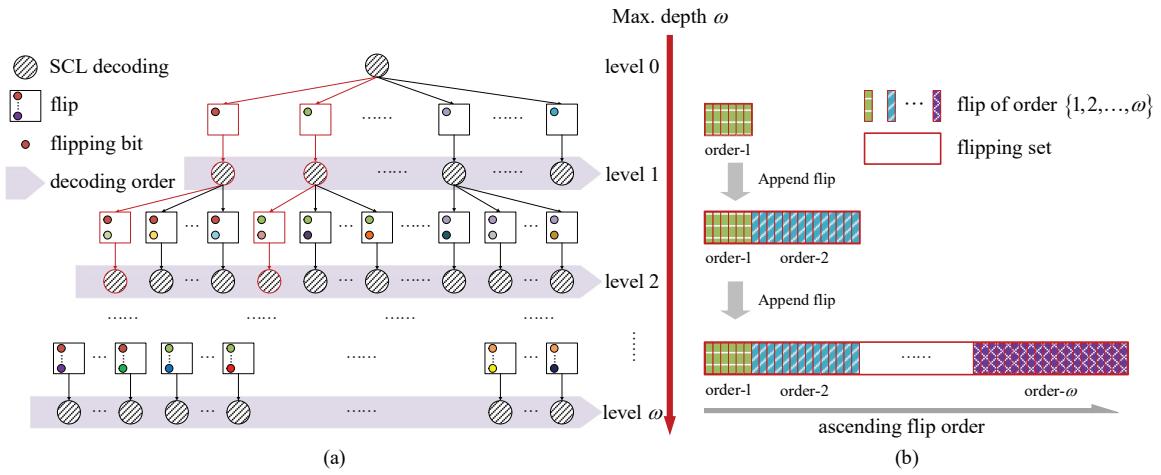


图 2.5 SCL-Flip-ω 译码流程示意图：(a) 算法流程，(b) 基于树状图构造翻转集合

征：(1) 最大深度 ω，它表示每次译码的最大阶段数，也即最大翻转次数；(2) 一系列参数 $\{T_{\omega'}^1,\ldots,T_{\omega'}^{\omega'}\}$，其中 $1\le\omega'\le\omega$，$T_{\omega'}^{\Omega}$ 代表节点的子节点数量，其中 $1\le\Omega\le\omega'$。例如，通过设置 $\{T_2^1=2,T_2^2=1\}$ 可以选中图2.5 (a) 中标红的节点。给定以上参数，阶段 ω' 中最大译码尝试的数量为 $T_{\omega'}=\prod_{\Omega=1}^{\omega'}T_{\omega'}^{\Omega}$。为了纠正译码错误，SCL-Flip-ω 译码器根据广度优先的变量确定嵌套的翻转元。例如，在第 ω' 阶段中，每次译码尝试中共有 ω' 个比特被翻转，其中前 $\omega'-1$ 个翻转比特来自于第 $\omega'-1$ 阶段中的译码尝试，最后一个翻转比特根据如下的基于 PM 的翻转度量确定：

$$M_{\mathrm{PM}}(i)=1.2\ln\left(\sum_{l=L+1}^{2L}e^{-PM_{i,l}}\right)-\ln\left(\sum_{l=1}^{L}e^{-PM_{i,l}}\right). \tag{2-28}$$

相应地，一旦当前译码尝试失败了，算法将生产 $T_{\omega'}^{\omega'}$ 个 ω' 阶的嵌套翻转元，并把它们添加进翻转集中，因此翻转中翻转元是按阶数累加的顺序排列的，如图2.5 (b) 所示。若第 ω' 阶段中所有的译码尝试都失败了，算法会执行第 $\omega'+1$ 阶段继续尝试纠正译码错误，在下个阶段中会有 $\omega'+1$ 个比特被翻转。整个算法遍历完树总计需要最多 $T=\sum_{\omega'=1}^{\omega}T_{\omega'}$ 次译码尝试，若在遍历过程中存在译码尝试的结果能够通过 CRC 校验，则输出这个结果并且算法终止。

2.3.4 快速 SC/SCL 译码算法

在 SC 和 SCL 译码中，每个比特估计取决于所有先前的比特估计，这样的顺序性质会导致很高的译码时延。SC/SCL 的译码速度可以通过部署并行译码器得到提升，这些并行译码器不需要顺序遍历整个译码树，它们可以在译码树的中间层节点处直接同时并行输出包含多个比特的估计码字。

基于此思想，文献 [42] 为 SC 译码提供了一种高度并行的 ML 译码器，该译码器可以并行求解一个通用节点的 ML 码字，然而特别是当节点长度较长的情况下该 ML 译码器的计算复杂度非常高。文献 [27] 发现对于某些具有特殊的信息和冻结比特分布模式的节点，SC 译码可以得到显著的简化。考虑一个长度为 M 的节点，其根节点处的 LLR 为 L_1^M，对应码字 β_1^M，该节点的在第 0 层的叶节点类型由向量 $\boldsymbol{c}$ 指示：若第 i 位是信息比特，$c_i = 1$，否则 $c_i = 0$。最典型的有 Rate-0、Rate-1、REP 和 SPC 这四种类型的特殊节点，它们的结构描述如下：

- Rate-0：所有的比特都是冻结比特，$\boldsymbol{c} = (0, 0, \ldots, 0)$；
- Rate-1：所有的比特都是信息比特，$\boldsymbol{c} = (1, 1, \ldots, 1)$；
- REP：最后一位是信息比特，其余都是冻结比特，$\boldsymbol{c} = (0, \ldots, 0, 1)$；
- SPC：第一位是冻结比特，其余都是信息比特，$\boldsymbol{c} = (0, 1, \ldots, 1)$。

文献 [27] 给出了 Rate-0 和 Rate-1 节点的快速 SC 译码方法，其中，Rate-0 节点码字是全零码字：

$$\beta_1^M = \mathbf{0}, \tag{2-29}$$

而 Rate-1 节点码字可以直接通过对根节点处的 LLR 作硬判决得到：

$$\beta_i = \frac{1 - \operatorname{sgn}(L_i)}{2}, \tag{2-30}$$

文献 [27] 拓展了 REP 和 SPC 节点的快速 SC 译码算法，REP 节点的码字要么是全零要么是全一，其 SC 译码结果如下：

$$\beta_1^M = \begin{cases} \mathbf{0}, & \sum_{i=1}^{M} L_i > 0 \\ \mathbf{0}, & \text{否则} \end{cases}, \tag{2-31}$$

SPC 节点的 SC 译码结果是通过翻转硬判决中最不可靠的比特实现的：

$$\begin{aligned} \hat{\beta}_i &= \mathrm{HD}(L_i), \\ \beta_i &= \begin{cases} \hat{\beta}_i \oplus \text{parity}, & i = k \\ \hat{\beta}_i, & \text{否则} \end{cases}, \end{aligned} \tag{2-32}$$

其中 $k = \arg\min_i |L_i|$ 是最不可靠的比特位置，校验值通过下式确定：

$$\text{parity} = \bigoplus_{i=1}^{M} \hat{\beta}_i \tag{2-33}$$

此外，文献 [28] 通过研究五个额外的特殊节点及其快速 SC 译码算法，进一步提高了 SC 的译码速度。这五个特殊节点的结构总结如下：

- Type I：最后两位是信息比特，其余都是冻结比特，$\boldsymbol{c} = (0, \ldots, 0, 1, 1)$.
- Type II：最后三位是信息比特，其余都是冻结比特，$\boldsymbol{c} = (0, \ldots, 0, 1, 1, 1)$.
- Type III：前两位是冻结比特，其余都是信息比特，$\boldsymbol{c} = (0, 0, 1, \ldots, 1)$.
- Type IV：前三位是冻结比特，其余都是信息比特，$\boldsymbol{c} = (0, 0, 0, 1, \ldots, 1)$.
- Type V：最后三位和倒数第五位是信息比特，其余都是冻结比特，$\boldsymbol{c} = (0, \ldots, 0, 1, 0, 1, 1, 1)$.

在文献 [29] 中，作者介绍了两种广义的 G-REP 和 G-PC 节点。G-REP 是一种位于译码树 p 层的节点，其每一层的左子节点均为 Rate-0 节点，而在 $q < p$ 层的右子节点是一个码率为 C 的通用节点。类似地，G-PC 是一种位于译码树 p 层的节点，其所有右子节点都是 Rate-1 节点，而在 $q < p$ 层的左子节点是 Rate-0 节点。

2.4 本章小结

本章介绍了极化码的基本原理和常见的译码算法，为本文后续的阐述奠定了基础。本章首先介绍了信道极化现象的原理，接下来介绍了极化码常用的 GA 构造算法，以及极化码具体的编码方式。随后，本章详细介绍了极化码的 SC、SCL、基于比特翻转的 SC/SCL 以及快速 SC/SCL 等译码算法的原理，后文针对极化码译码算法的进一步研究离不开本章的阐述内容。

第 3 章 动态 SCL 翻转译码算法

3.1 引言

在 SCL-Flip 和 SCL-Flip-ω 译码算法中，翻转度量仅能估计第一个错误比特位置，即仅考虑了一阶翻转元，而忽略了在顺序 SCL 译码过程中可能发生的更高阶的译码错误。此外，SCL-Flip-ω 译码算法通过经验观察到高阶的译码错误更为罕见，从而基于树状图的广度优先遍历构造了翻转集合，其中高阶翻转元的优先级更低。然而当初始 SCL 译码发生高阶错误时，该算法可能导致大量不必要的译码尝试，从而导致最差情况下译码复杂度非常高。事实上，好的翻转度量应该能够评估不同阶数的翻转元成功纠正译码错误的概率，通过评估所有翻转元的成功概率，译码尝试可以优先选择成功概率最高的一系列候选翻转元。本章基于这样的思想提出新的 SCL 翻转译码算法，并在该算法基础上对其计算复杂度、纠错性能以及译码时延做进一步的优化。

本章用到的符号定义如下：定义 ω 阶的翻转元为 $\mathcal{E}_\omega = \{i_1, i_2, \ldots, i_\omega\}$，其中 $\mathcal{E}_\omega \subset \mathcal{A}^*$ 且比特索引 $i_1, i_2, \ldots, i_\omega$ 按升序排列。SCL($\mathcal{E}_\omega$) 表示使用翻转元 $\mathcal{E}_\omega$ 的 SCL 译码尝试，下文中使用符号 $[\mathcal{E}_\omega]$ 用以说明对应的变量来自译码尝试 SCL($\mathcal{E}_\omega$)，例如，$PM[\mathcal{E}_\omega]_i$、$\mathcal{L}[\mathcal{E}_\omega]_i$、$\mathcal{R}[\mathcal{E}_\omega]_i$ 和 $\mathcal{D}[\mathcal{E}_\omega]_i$ 分别表示 SCL($\mathcal{E}_\omega$) 中的 PM 列表、路径列表、保留路径列表和淘汰路径列表。

3.2 动态翻转度量

在介绍 SCL 翻转译码算法之前，本节首先介绍算法中用到的翻转度量。如本章第 1 节所述，本文提出的翻转度量目标是评估任意翻转元成功纠正初始 SCL 译码错误的概率。

首先，对于一个 ω 阶的翻转元 $\mathcal{E}_\omega$，取出其前 ω' 个（$\omega' < \omega$）比特组成一个 ω' 阶的翻转元 $\mathcal{E}_{\omega'} = \{i_1, i_2, \ldots, i_{\omega'}\}$，那么译码尝试 SCL($\mathcal{E}_\omega$) 和 SCL($\mathcal{E}_{\omega'}$) 中各个变量的关系表示如下：

$$\begin{aligned} PM[\mathcal{E}_\omega]_i &= PM[\mathcal{E}_{\omega'}]_i, \\ \mathcal{L}[\mathcal{E}_\omega]_i &= \mathcal{L}[\mathcal{E}_{\omega'}]_i, \\ \mathcal{R}[\mathcal{E}_\omega]_{i_{\omega'}} &= \mathcal{D}[\mathcal{E}_{\omega'}]_{i_{\omega'}}, \end{aligned} \tag{3-1}$$

其中 $i \le i_{\omega'}$。接下来，定义 $P(\mathcal{E}_\omega)$ 为 $\mathcal{E}_\omega$ 纠正 SCL 译码错误的概率，那么根据公式(3-1)和条件概率链式法则，$P(\mathcal{E}_\omega)$ 可以由 $P(\mathcal{E}_{\omega-1})$ 得到，即：

$$\begin{aligned}
P(\mathcal{E}_\omega) =& \Pr(u_1^{i_\omega} \subset \mathcal{R}[\mathcal{E}_\omega]_{i_\omega} \mid y_1^N) \\
=& \Pr(u_1^{i_{\omega-1}} \subset \mathcal{R}[\mathcal{E}_{\omega-1}]_{i_{\omega-1}}, u_1^{i_\omega} \subset \mathcal{D}[\mathcal{E}_{\omega-1}]_{i_\omega} \mid y_1^N) \\
=& \Pr(u_1^{i_\omega} \subset \mathcal{D}[\mathcal{E}_{\omega-1}]_{i_\omega} \mid y_1^N, u_1^{i_\omega-1} \subset \mathcal{R}[\mathcal{E}_{\omega-1}]_{i_\omega-1}) \\
& \prod_{j=i_{\omega-1}+1}^{i_\omega-1} \Pr(u_1^j \subset \mathcal{R}[\mathcal{E}_{\omega-1}]_j \mid y_1^N, u_1^{j-1} \subset \mathcal{R}[\mathcal{E}_{\omega-1}]_{j-1}) \Pr(u_1^{i_{\omega-1}} \subset \mathcal{R}[\mathcal{E}_{\omega-1}]_{i_{\omega-1}} \mid y_1^N) \\
=& p(\mathcal{E}_{\omega-1})_{i_\omega} \prod_{j=i_{\omega-1}+1}^{i_\omega-1} (1 - p(\mathcal{E}_{\omega-1})_j) P(\mathcal{E}_{\omega-1}),
\end{aligned} \tag{3-2}$$

其中 $p(\mathcal{E}_{\omega-1})_j \triangleq \Pr(u_1^j \subset \mathcal{D}[\mathcal{E}_{\omega-1}]_j \mid y_1^N, u_1^{j-1} \subset \mathcal{R}[\mathcal{E}_{\omega-1}]_{j-1})$。从公式(3-2)可以看出 $P(\mathcal{E}_\omega)$ 取决于 $P(\mathcal{E}_{\omega-1})$，而 $P(\mathcal{E}_{\omega-1})$ 又取决于 $P(\mathcal{E}_{\omega-2})$，以此类推。通过展开这样的递归关系，可以得到：

$$\begin{aligned}
P(\mathcal{E}_\omega) &= \prod_{j \in \mathcal{E}_\omega} p(\mathcal{E}_{\omega-1})_j \prod_{\substack{j<i_\omega \\ j \notin \mathcal{E}_\omega}} (1 - p(\mathcal{E}_{\omega-1})_j) \\
&\overset{(a)}{=} \prod_{j \in \mathcal{E}_\omega} p(\mathcal{E}_{\omega-1})_j \prod_{\substack{j<i_\omega \\ j \in \mathcal{A}^* \setminus \mathcal{E}_\omega}} (1 - p(\mathcal{E}_{\omega-1})_j),
\end{aligned} \tag{3-3}$$

其中 (a) 是因为 $p(\mathcal{E}_{\omega-1})_j = 0$ 对于任意 $j \notin \mathcal{A}^*$ 成立。

假设当译码进行到比特 u_{j-1} 前译码没有发生错误，即正确路径被保留在已有列表中（$u_1^{j-1} \subset \mathcal{R}[\mathcal{E}_{\omega-1}]_{j-1}$），那么无论接下来的 u_{j-1} 是信息比特还是冻结比特，正确路径都一定被 $\mathcal{L}[\mathcal{E}_{\omega-1}]_j$ 所包含。因此，可以得到 $\Pr(u_1^j \subset \mathcal{D}[\mathcal{E}_{\omega-1}]_j \mid y_1^N, u_1^{j-1} \subset \mathcal{R}[\mathcal{E}_{\omega-1}]_{j-1}) = \Pr(u_1^j \subset \mathcal{D}[\mathcal{E}_{\omega-1}]_j \mid y_1^N, u_1^j \subset \mathcal{L}[\mathcal{E}_{\omega-1}]_j)$，相应地 $p(\mathcal{E}_{\omega-1})_j$ 可以重写为下式：

$$\begin{aligned}
p(\mathcal{E}_{\omega-1})_j &= \Pr(u_1^j \subset \mathcal{D}[\mathcal{E}_{\omega-1}]_j \mid y_1^N, u_1^j \subset \mathcal{L}[\mathcal{E}_{\omega-1}]_j) \\
&= \frac{\Pr(u_1^j \subset \mathcal{D}[\mathcal{E}_{\omega-1}]_j \mid y_1^N)}{\Pr(u_1^j \subset \mathcal{L}[\mathcal{E}_{\omega-1}]_j \mid y_1^N)} \\
&= \frac{\sum_{l=L+1}^{2L} \Pr(u_1^j = \hat{u}[\mathcal{E}_{\omega-1}]_{0,l}^j \mid y_1^N)}{\sum_{l=1}^{2L} \Pr(u_1^j = \hat{u}[\mathcal{E}_{\omega-1}]_{0,l}^j \mid y_1^N)} \\
&\overset{(a)}{=} \frac{\sum_{l=L+1}^{2L} e^{-\alpha PM[\mathcal{E}_{\omega-1}]_{j,l}}}{\sum_{l=1}^{2L} e^{-\alpha PM[\mathcal{E}_{\omega-1}]_{j,l}}}, \quad \forall j \in \mathcal{A}^*,
\end{aligned} \tag{3-4}$$

其中 (a) 是由公式(2-20)中 PM 的定义得到的，α 是一个用于近似 $\Pr(u_1^j = \hat{u}[\mathcal{E}_{\omega-1}]_{0,l}^j \mid y_1^N)$ 的扰动参数。值得说明的是，扰动参数被广泛应用于提高以软信息为表征的概率的准

确度，这样能够进一步提升译码的性能[16,19-20]，通常这些参数可藉由蒙特卡洛仿真于译码开始前离线确定。将公式(3-4)中的 $p(\mathcal{E}_{\omega-1})_j$ 代入公式(3-3)，可以得到如下的翻转度量 $M(\mathcal{E}_\omega)$：

$$M(\mathcal{E}_\omega)=\prod_{j\in\mathcal{E}_\omega}\frac{\sum_{l=L+1}^{2L}e^{-\alpha PM[\mathcal{E}_{\omega-1}]_{j,l}}}{\sum_{l=1}^{2L}e^{-\alpha PM[\mathcal{E}_{\omega-1}]_{j,l}}}\prod_{\substack{j<i_\omega\\ j\in\mathcal{A}^*\setminus\mathcal{E}_\omega}}\frac{\sum_{l=1}^{L}e^{-\alpha PM[\mathcal{E}_{\omega-1}]_{j,l}}}{\sum_{l=1}^{2L}e^{-\alpha PM[\mathcal{E}_{\omega-1}]_{j,l}}}. \tag{3-5}$$

公式(3-5)中的翻转度量评估了翻转元 $\mathcal{E}_\omega$ 能够纠正 SCL 译码错误的概率。此外，将(3-4)代入(3-2)，可以得到递归形式的翻转度量：

$$M(\mathcal{E}_\omega)=M(\mathcal{E}_{\omega-1})\frac{\sum_{l=L+1}^{2L}e^{-\alpha PM[\mathcal{E}_{\omega-1}]_{i_\omega,l}}}{\sum_{l=1}^{2L}e^{-\alpha PM[\mathcal{E}_{\omega-1}]_{i_\omega,l}}}\prod_{\substack{j=i_{\omega-1}+1\\ j\in\mathcal{A}^*}}^{i_\omega-1}\frac{\sum_{l=1}^{L}e^{-\alpha PM[\mathcal{E}_{\omega-1}]_{j,l}}}{\sum_{l=1}^{2L}e^{-\alpha PM[\mathcal{E}_{\omega-1}]_{j,l}}}. \tag{3-6}$$

公式(3-6)表明 ω 阶翻转元 $\mathcal{E}_\omega$ 的翻转度量 $M(\mathcal{E}_\omega)$ 可以根据 $\omega-1$ 阶翻转元 $\mathcal{E}_{\omega-1}$ 的翻转度量 $M(\mathcal{E}_{\omega-1})$ 计算，相比公式(3-6)，利用公式(3-6)计算高阶翻转元的翻转度量可以降低计算复杂度。特别地，对于 1 阶的翻转元 $\mathcal{E}_1=\{i_1\}$，$M(\mathcal{E}_1)$ 的计算如下所示：

$$M(\mathcal{E}_1)=\frac{\sum_{l=L+1}^{2L}e^{-\alpha PM[\emptyset]_{i_1,l}}}{\sum_{l=1}^{2L}e^{-\alpha PM[\emptyset]_{i_1,l}}}\prod_{\substack{j<i_1\\ j\in\mathcal{A}^*}}\frac{\sum_{l=1}^{L}e^{-\alpha PM[\emptyset]_{j,l}}}{\sum_{l=1}^{2L}e^{-\alpha PM[\emptyset]_{j,l}}}, \tag{3-7}$$

其中 $PM[\emptyset]_{j,l}$ 来自初始的 SCL 译码尝试 SCL($\emptyset$)。

注 1. 注意公式(3-2)-公式(3-7)是基于反向路径选择策略推导出的。通过遵循类似的推导，也可以获得其他比特翻转策略的翻转度量。首先，定义 $P_H(\mathcal{E}_\omega)$ 为使用广义比特翻转策略 $H(\cdot)$ 成功纠正 *SCL* 译码错误的概率。类似于公式(3-2)中的推导，$P_H(\mathcal{E}_\omega)$ 可以由下式得到：

$$\begin{aligned}
P_H(\mathcal{E}_\omega)=&\Pr(u_1^{i_\omega}\subset\mathcal{R}[\mathcal{E}_\omega]_{i_\omega}\mid y_1^N)\\
=&\Pr(u_1^{i_{\omega-1}}\subset h(\mathcal{L}[\mathcal{E}_{\omega-1}]_{i_{\omega-1}}),u_1^{i_\omega}\subset\mathcal{R}[\mathcal{E}_{\omega-1}]_{i_\omega}\mid y_1^N)\\
=&\Pr(u_1^{i_\omega}\subset h(\mathcal{L}[\mathcal{E}_{\omega-1}]_{i_\omega})\mid y_1^N,u_1^{i_\omega-1}\subset\mathcal{R}[\mathcal{E}_{\omega-1}]_{i_\omega-1})\\
&\prod_{j=i_{\omega-1}+1}^{i_\omega-1}\Pr(u_1^j\subset\mathcal{R}[\mathcal{E}_{\omega-1}]_j\mid y_1^N,u_1^{j-1}\subset\mathcal{R}[\mathcal{E}_{\omega-1}]_{j-1})\Pr(u_1^{i_{\omega-1}}\subset\mathcal{R}[\mathcal{E}_{\omega-1}]_{i_{\omega-1}}\mid y_1^N)\\
=&p_h(\mathcal{E}_{\omega-1})_{i_\omega}\prod_{j=i_{\omega-1}+1}^{i_\omega-1}(1-p(\mathcal{E}_{\omega-1})_j)P_H(\mathcal{E}_{\omega-1})\\
=&\prod_{j\in\mathcal{E}_\omega}p_h(\mathcal{E}_{\omega-1})_j\prod_{\substack{j<i_\omega\\ j\in\mathcal{A}^*\setminus\mathcal{E}_\omega}}(1-p(\mathcal{E}_{\omega-1})_j),
\end{aligned} \tag{3-8}$$

其中 $p_h(\mathcal{E}_{\omega-1})_j \triangleq \Pr(u_1^j \subset h(\mathcal{L}[\mathcal{E}_{\omega-1}]_j) \mid y_1^N, u_1^{j-1} \subset \mathcal{R}[\mathcal{E}_{\omega-1}]_j)$，其计算公式如下：

$$p_h(\mathcal{E}_{\omega-1})_j = \frac{\sum_{l\in\mathbb{L}} e^{-\alpha PM[\mathcal{E}_{\omega-1}]_{j,l}}}{\sum_{l=1}^{2L} e^{-\alpha PM[\mathcal{E}_{\omega-1}]_{j,l}}}, \quad \forall j \in \mathcal{A}^*, \tag{3-9}$$

其中 $\mathbb{L}$ 是由剪枝函数 $h(\mathcal{L}_i)$ 决定的保留路径的索引集合。最后，将(3-9)代入(3-8)，可以得到与 $H(\cdot)$ 相关的翻转度量：

$$M(\mathcal{E}_\omega) = \prod_{j\in\mathcal{E}_\omega} \frac{\sum_{l\in\mathbb{L}} e^{-\alpha PM[\mathcal{E}_{\omega-1}]_{j,l}}}{\sum_{l=1}^{2L} e^{-\alpha PM[\mathcal{E}_{\omega-1}]_{j,l}}} \prod_{\substack{j<i_\omega \\ j\in\mathcal{A}^*\setminus\mathcal{E}_\omega}} \frac{\sum_{l=1}^{L} e^{-\alpha PM[\mathcal{E}_{\omega-1}]_{j,l}}}{\sum_{l=1}^{2L} e^{-\alpha PM[\mathcal{E}_{\omega-1}]_{j,l}}}. \tag{3-10}$$

根据公式(3-10)，可以得到适用于移位剪枝策略（公式(2-24)）的翻转度量：

$$M(\mathcal{E}_\omega) = \prod_{j\in\mathcal{E}_\omega} \frac{\sum_{l=k+1}^{k+L} e^{-\alpha PM[\mathcal{E}_{\omega-1}]_{j,l}}}{\sum_{l=1}^{2L} e^{-\alpha PM[\mathcal{E}_{\omega-1}]_{j,l}}} \prod_{\substack{j<i_\omega \\ j\in\mathcal{A}^*\setminus\mathcal{E}_\omega}} \frac{\sum_{l=1}^{L} e^{-\alpha PM[\mathcal{E}_{\omega-1}]_{j,l}}}{\sum_{l=1}^{2L} e^{-\alpha PM[\mathcal{E}_{\omega-1}]_{j,l}}}, \tag{3-11}$$

适用于其他比特翻转策略的翻转度量也可以用类似方法得到。

3.3 动态 SCL 翻转译码算法

本节根据 D-SC-Flip 译码算法[16]的思想构造翻转集合 $\mathcal{S}$，即根据初始 SCL 译码中的信息初始化 $\mathcal{S}$ 并在每次译码尝试后动态更新 $\mathcal{S}$。结合公式(3-6)中的翻转度量和动态构建翻转集合的方法，本文提出了一种新的动态 SCL 翻转译码算法，即 D-SCL-Flip，并将其总结在算法1中。

在所提出的 D-SCL-Flip 译码算法中，如果初始的 SCL 译码尝试结果未能通过 CRC 校验，则考虑所有可能的 1 阶翻转元以初始化翻转集合 $\mathcal{S}$ 和度量集 $\mathcal{M}$，如算法1中第 7-10 行所示。对于每一个 1 阶翻转元 $\mathcal{E} = \{i\}$，首先使用公式(3-7)计算翻转度量 $M(\mathcal{E})$。然后，分别向 $\mathcal{S}$ 和 $\mathcal{M}$ 中添加 $\mathcal{E}$ 和 $M(\mathcal{E})$。初始化过程中使用从初始 SCL 译码 SCL($\emptyset$) 得到的 PM 值 $\{PM[\emptyset]_i\}_{i\in\mathcal{A}^*}$ 计算翻转度量。根据具有更大度量值的翻转元拥有更高的优先级这一规则，翻转集合 $\mathcal{S}$ 根据索引向量 I 重新排列，使得 $\mathcal{M}(I)$ 按降序排序。最后，前 T 个 1 阶翻转元被保留在 $\mathcal{S}$ 中，其对应的度量值存储在 $\mathcal{M}$ 中（第 11 行）。

然后在每次译码尝试中，从 $\mathcal{S}$ 中选择一个翻转元（第 13-18 行），第 t 次译码尝试的翻转元由 $\mathcal{E}_t = \{i_1, \ldots, i_{\omega_t}\}$ 表示，其中 ω_t 表示 $\mathcal{E}_t$ 的阶数。重复该过程，直到一个译码尝试的结果能够通过 CRC 校验或译码尝试次数超过 T。对于每次失败的译码尝试 SCL($\mathcal{E}_t$)，利用 SCL($\mathcal{E}_t$) 中的 PM 更新 $\mathcal{S}$ 和 $\mathcal{M}$，如第 19-24 行所述。在第 18 行中，将当前翻转元 $\mathcal{E}_t$ 与索引 i 相连接可以构造高阶翻转元，即阶数为 $(\omega_t + 1)$ 的新翻转元

算法 1 D-SCL-Flip 译码算法

输入: y_1^N，L 和 T

输出: $\hat{u}_1^N$

$(\mathcal{R}[\emptyset]_N, \{PM[\emptyset]_i\}_{i\in\mathcal{A}^*}) \leftarrow \mathrm{SCL}(\emptyset)$
for $l = 1, 2, \ldots, L$ **do**
  if $\mathrm{CRC}(\mathcal{R}[\emptyset]_{N,l}) = \mathrm{success}$ **then**
    return $\hat{u}_1^N = \mathcal{R}[\emptyset]_{N,l}$
  end if
end for
$\mathcal{S} = \emptyset, \mathcal{M} = \emptyset$
for $\forall i \in \mathcal{A}^*$ **do**
  $\mathcal{E} = \{i\}$，根据公式(3-7)计算 $M(\mathcal{E})$，$\mathcal{S} = \{\mathcal{S}, \mathcal{E}\}, \mathcal{M} = \{\mathcal{M}, M(\mathcal{E})\}$
end for
$I \leftarrow \mathrm{sort\_index}(\mathcal{M})$，$\mathcal{S} = \mathcal{S}(I(1:T)), \mathcal{M} = \mathcal{M}(I(1:T))$
for $t = 1, 2, \ldots, T$ **do**
  $(\mathcal{R}[\mathcal{E}_t]_N, \{PM[\mathcal{E}_t]_i\}_{i\in\mathcal{A}^*}) \leftarrow \mathrm{SCL}(\mathcal{E}_t)$
  for $l = 1, 2, \ldots, L$ **do**
    if $\mathrm{CRC}(\mathcal{R}[\mathcal{E}_t]_{N,l}) = \mathrm{success}$ **then**
      return $\hat{u}_1^N = \mathcal{R}[\mathcal{E}_t]_{N,l}$
    end if
  end for
  for $\forall i \in \mathcal{A}^*$ and $i > i_{\omega_t}$ **do**
    $\mathcal{E} = \mathcal{E}_t \cup \{i\}$，根据公式(3-6)计算 $M(\mathcal{E})$
    if $M(\mathcal{E}) > \mathrm{last}(\mathcal{M})$ or $\mathrm{size}(\mathcal{S}) < T$ **then**
      $(\mathcal{S}, \mathcal{M}) \leftarrow \mathrm{Insert}(\mathcal{S}, \mathcal{M}, \mathcal{E}, M(\mathcal{E}))$
    end if
  end for
end for
return $\hat{u}_1^N = \mathcal{R}[\mathcal{E}_t]_{N,1}$

$\mathcal{E} = \mathcal{E}_t \cup \{i\}$，其中 $i \in \mathcal{A}^*$ 且 $i > i_{\omega_t}$。接下来，算法根据公式(3-6)比较了新的翻转元与其他现有翻转元的翻转度量值，新的翻转元基于 $M(\mathcal{E}_t)$ 和 $\{PM[\mathcal{E}_t]_i\}_{i\in\mathcal{A}^*}$ 计算翻转度量值，其中 $\{PM[\mathcal{E}_t]_i\}_{i\in\mathcal{A}^*}$ 来自于当前译码尝试 SCL($\mathcal{E}_t$)。最后，根据度量值 $M(\mathcal{E})$ 的大小将 $\mathcal{E}$ 和 $M(\mathcal{E})$ 插入到合适的位置（这个步骤可以用 22 行中的函数 Insert 表示），完成 $\mathcal{S}$ 和 $\mathcal{M}$ 的更新过程。给定最大尝试次数 T，新的翻转元仅在如下两种情况下才会被插入到 $\mathcal{S}$ 中：（1）$\mathcal{S}$ 的大小不足 T；（2）新翻转元的度量值 $M(\mathcal{E})$ 比 $\mathcal{S}$ 中最后一个翻转元的度量值大，即 $M(\mathcal{E}) > \text{last}(\mathcal{M})$，这意味着与最后一个翻转元相比新翻转元成功纠正 SCL 译码错误的概率更大。

D-SCL-Flip 译码算法中的翻转集合 $\mathcal{S}$ 能够被动态更新，保证了每次使用的翻转元能够纠正 SCL 译码错误的概率最高。图3.1描述了所提出的 D-SCL-Flip 译码器中翻转集合的结构示意图。可以观察到翻转集合中翻转元是根据翻转度量值排列的，使得具有较高纠正错误概率的翻转元在 $\mathcal{S}$ 中具有更高的优先级。另一方面，如图2.5所示，SCL-Flip-ω 译码算法中低阶翻转元具有更高的优先级，因此在发生高阶译码错误时算法需要进行大量冗余的译码尝试。相比之下本章所提的 D-SCL-Flip 译码算法将不同阶数的翻转元按照其纠正错误概率的大小排列，避免了冗余的译码尝试，从而取得更低的译码延迟和计算复杂度。

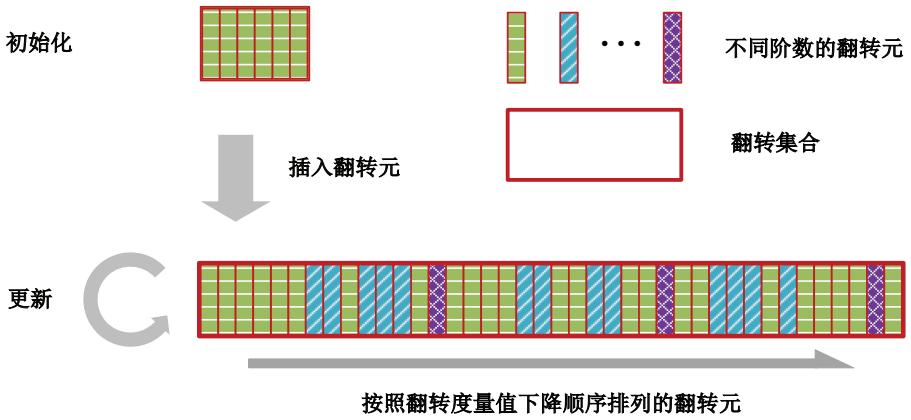


图 3.1 D-SCL-Flip 译码算法中翻转集合的结构示意图

3.4 简化的翻转度量

从公式(3-6)中可以看出，D-SCL-Flip 译码算法中的翻转度量涉及大量对数和指数运算，这导致了较高的计算复杂度。因此，可以仅考虑 1 和 $L+1$ 的两条路径来近似

$p_i(\mathcal{E}_{\omega-1})$：

$$\begin{aligned} q_i(\mathcal{E}_{\omega-1}) &= \frac{1}{1+e^{\alpha(PM[\mathcal{E}_{\omega-1}]_{j,L+1}-PM[\mathcal{E}_{\omega-1}]_{j,1})}} \\ &= \frac{e^{\alpha\Delta_{PM}[\mathcal{E}_{\omega-1}]_j}}{1+e^{\alpha\Delta_{PM}[\mathcal{E}_{\omega-1}]_j}}, \quad \forall j \in \mathcal{A}^*, \end{aligned} \tag{3-12}$$

其中 $q_i(\mathcal{E}_{\omega-1})$ 是 $p_i(\mathcal{E}_{\omega-1})$ 的近似，$\Delta_{PM}[\mathcal{E}_{\omega-1}]_j = PM[\mathcal{E}_{\omega-1}]_{j,1} - PM[\mathcal{E}_{\omega-1}]_{j,L+1}$。此外，为计算数值的稳定性，将 $p_i(\mathcal{E}_{\omega-1}) \approx q_i(\mathcal{E}_{\omega-1})$ 代入公式(3-3)中并将 $M(\mathcal{E}_\omega)$ 化简成如下的对数形式:

$$\begin{aligned} M(\mathcal{E}_\omega) =& \sum_{j\in\mathcal{E}_\omega} \Delta_{PM}[\mathcal{E}_{\omega-1}]_j - \frac{1}{\alpha}\sum_{\substack{j\le i_\omega \\ j\in\mathcal{A}^*}} \log\left(1+e^{\alpha\Delta_{PM}[\mathcal{E}_{\omega-1}]_j}\right) \\ =& M(\mathcal{E}_{\omega-1}) + \Delta_{PM}[\mathcal{E}_{\omega-1}]_{i_\omega} - \frac{1}{\alpha}\sum_{\substack{j=i_{\omega-1}+1 \\ j\in\mathcal{A}^*}}^{i_\omega} \log\left(1+e^{\alpha\Delta_{PM}[\mathcal{E}_{\omega-1}]_j}\right). \end{aligned} \tag{3-13}$$

基于假设 $\Delta_{PM}[\mathcal{E}_{\omega-1}]_j \ll 0$，可以忽略公式(3-13)中的指数项，从而得到进一步化简的翻转度量 M_{sim}：

$$M_{\text{sim}}(\mathcal{E}_\omega) = \sum_{j\in\mathcal{E}_\omega} \Delta_{PM}[\mathcal{E}_{\omega-1}]_j = M(\mathcal{E}_{\omega-1}) + \Delta_{PM}[\mathcal{E}_{\omega-1}]_{i_\omega}. \tag{3-14}$$

与公式(3-6)中的形式相比，公式(3-14)中的翻转度量不包含超越函数因而更利于硬件实现。然而，原先公式(3-6)中的翻转度量源于成功纠正错误的概率，公式(3-14)采取的近似和化简操作可能会导致 D-SCL-Flip 译码性能上的损失。

3.5 深度学习辅助的翻转度量

由于使用的近似和化简技术可能会影响的译码错误位置的估计准确度，本节向简化的翻转度量 M_{sim} 中引入了扰动参数向量 $\boldsymbol{W} = (w_1, w_2, \ldots, w_{2L})$，用以提高 D-SCL-Flip 的译码性能。加入了扰动参数的新翻转度量如下式所示：

$$M_{\text{DL}}(\mathcal{E}_\omega) = \sum_{j\in\mathcal{E}_\omega}\sum_{l=1}^{2L} w_l PM[\mathcal{E}_{\omega-1}]_{j,1}. \tag{3-15}$$

在公式(3-15)中，$\boldsymbol{W}$ 最优值的确定可以被视为一项学习任务，本文采用 DL 技术来解决这个问题。为了降低学习难度，可以对参数 $\boldsymbol{W}$ 的优化问题作出以下两点的简化：（1）由于在 D-SCL-Flip 译码算法中翻转元的阶数不定，$\boldsymbol{W}$ 最优值会随阶数变化，本文仅考虑 1 阶情况下 $\boldsymbol{W}$ 的最优值，对于 1 阶的翻转元公式(3-15)可以化简为如下形式：

$$M_{\text{DL}}(i) = \sum_{l=1}^{2L} w_l PM_{i,l}. \tag{3-16}$$

（2）在 SCL 译码过程中，比特翻转策略仅可适用于 $\mathcal{A}^*$ 包含的比特位置上，因此忽略其他冗余的比特可以降低数据维度，从而降低学习的难度与复杂度，相应的 M_{DL} 可以用以下矩阵形式表示：

$$\tilde{M}_{\text{DL}} = \boldsymbol{P}\boldsymbol{W}, \tag{3-17}$$

其中 $\boldsymbol{P}$ 是一个 $(K + r - \log_2 L) \times 2L$ 的矩阵，其元素由 $P(i,l) = PM_{\mathcal{A}^*(i),l}$ 确定。

然而，公式(3-17)中翻转度量并没有可供于训练的标签值，因此本文将重点放在正确路径被淘汰的比特位置（即需要翻转的比特位置）i_E 上，其中 $i_E \in \mathcal{A}^*$。利用长度为 $K + r - \log_2 L$ 的一位有效编码向量 $\boldsymbol{O}$ 表示需要进行翻转的比特位置，其中 $\boldsymbol{O}$ 的具体形式可以表示如下：

$$O_i = \begin{cases} 1, & \text{若 } \mathcal{A}^*(i) = i_E \\ 0, & \text{否则} \end{cases}, \tag{3-18}$$

$O_i = 1$ 和 $O_i = 0$ 分别表示在位置 $\mathcal{A}^*(i)$ 处进行或不进行翻转。此外，令 $\hat{\boldsymbol{O}}$ 表示 $\boldsymbol{O}$ 的估计值，即：

$$\hat{O}_i = \begin{cases} 1, & \text{若 } i = \arg\max_{1 \le i \le K+r-\log_2 L} \tilde{M}_{\text{DL}}(i) \\ 0, & \text{否则} \end{cases}. \tag{3-19}$$

学习任务的目标是通过反向传播算法[43]最小化估计值 $\hat{\boldsymbol{O}}$ 和标签值 $\boldsymbol{O}$ 之间的误差，从而优化参数 $\boldsymbol{W}$。然而，公式(3-19)中 $\arg\max$ 函数是一个不可微的离散函数，它阻碍了梯度在网络间的反向传播。注意到 $\arg\max$ 函数可以通过连续可导的 softmax 函数来逼近，那么 $\hat{\boldsymbol{O}}$ 可以近似为 $\tilde{\boldsymbol{O}}$：

$$\tilde{O}_i = \text{softmax}(\tilde{M}_{\text{DL}})_i = \frac{e^{\tilde{M}_{\text{DL}}(i)}}{\sum_{j=1}^{K+r-\log_2 L} e^{\tilde{M}_{\text{DL}}(j)}}. \tag{3-20}$$

对于采用 $L = 8$ 的极化码 $\mathcal{P}(16, 9)$，本节所提的 DL 方法总结在如图3.2所示的 DNN 模型中。可以观察到，该 DNN 的输入是 PM 矩阵 $\boldsymbol{P}$，输入层包含 $(K + r - \log_2 L) \times 2L$ 个神经元，且 $\boldsymbol{P}$ 中每一行的神经元共享相同的权重 $\boldsymbol{W}$。全连接层中的神经元数量为 $K + r - \log_2 L$，且均不使用激活函数。输出层具有 $K + r - \log_2 L$ 个神经元，每个神经元代表 $\mathcal{A}^*$ 中的一个信息比特，其中用绿色标记的神经元代表估计的翻转位置，对应于译码树中的比特 u_{12}。

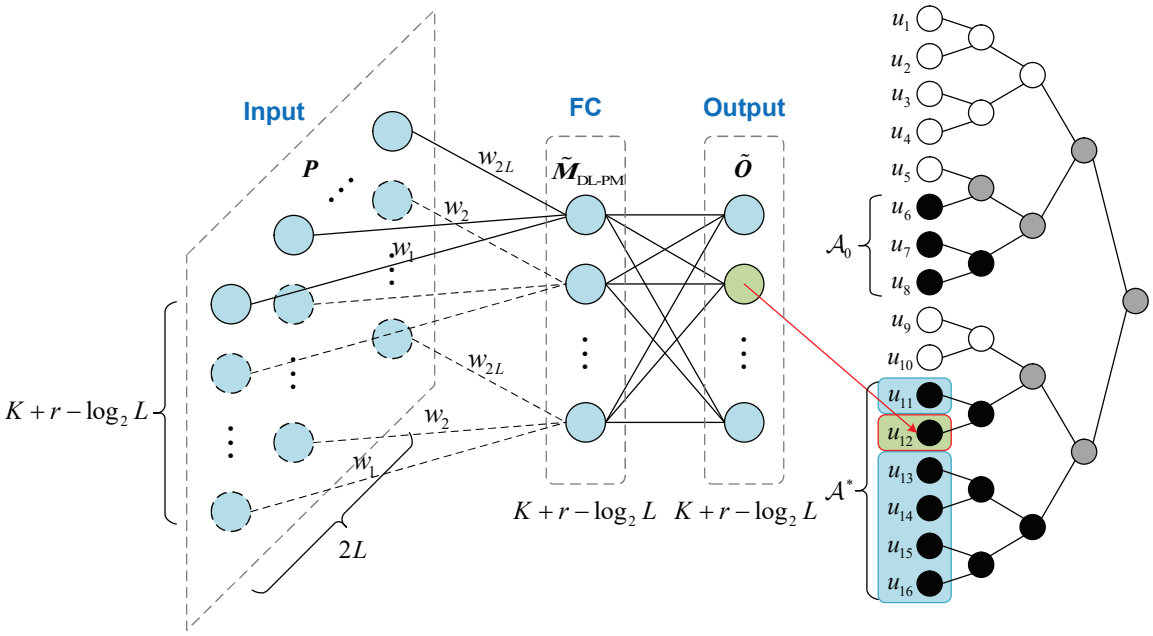


图 3.2 DNN 结构示意图

在训练阶段，DNN 的损失函数采用 $\boldsymbol{O}$ 和 $\tilde{\boldsymbol{O}}$ 间的交叉熵：

$$Loss = \frac{1}{K+r-\log_2 L} \sum_{i=1}^{K+r-\log_2 L} O_i \log \tilde{O}_i + (1-O_i)\log(1-\tilde{O}_i), \tag{3-21}$$

通过自适应矩估计（Adaptive Moment Estimation, Adam）优化器最小化该损失函数训练整个 DNN。值得说明的是，在训练开始时采用初始化 $\boldsymbol{W} = (1,0,\ldots,0,-1,0,\ldots,0)^T$ 可以加速训练过程。离线训练得到的 $\boldsymbol{W}$ 可以用于公式(3-15)进行翻转度量的线上计算，而文献 [23] 则是将整个 DL 网络模型应用于线上计算中，即每次估计翻转位置都需要网络模型的一次正向传播计算，这造成了额外的存储和计算开销，相比之下本节所提的 DNN 模型并未实际参与线上计算，从而避免了高昂的存储和计算成本。

3.6 快速译码

SC 和 SCL 译码从上到下和从左到右遍历译码树的过程会导致很高译码时延。为了解决这一问题，现有文献已经为特殊节点设计了快速 SC 和 SCL 译码技术[31-34]。这些译码器能够并行地输出多个比特，从而避免了译码树的完全遍历并显著地加速译码过程。由于 D-SCL-Flip 译码器在每次重新译码尝试中使用 SCL 译码，因此实现快速 SCL 译码器也将提高 D-SCL-Fip 译码器的吞吐量。然而，在 D-SCL-Flip 译码器中，比特翻转操作需要在叶子节点处执行，这意味着现有的快速译码技术不能直接用于加速 D-SCL-Fip 译码器。因此，本节针对现有的特殊节点提出了一种新的比特翻转策略，该策略能够翻

转根节点处的估计码字。

此外，现有的翻转度量的计算依赖于比特级的 PM，而快速 SCL 译码[33-34]得到的是节点级的 PM，因此利用节点级的 PM 计算比特级的 PM 将会引入额外的时延。为了实现快速译码，D-SCL-Flip 译码算法中的翻转度量需要调整以使其能够基于根节点处 PM 直接计算。值得说明的是，根节点处 PM 的计算过程（称为估计步骤）与快速 SCL 译码中的过程相同，本文的目标是提出适用于特殊节点的比特翻转策略和翻转度量。将 j 定义为特殊节点索引（包含的第一个比特的位置索引），特殊节点可以由坐标 $\{j,i\}$ 表示，其中 $1 \leq i \leq E$，E 表示利用快速 SCL 译码技术所需的总时间步长。在下文中，原始比特索引 i 将更改为 $\{j,i\}$，以指示译码以节点的方式进行。此外，$j_1 \leq \cdots \leq j_\omega$，$1 \leq k_1, \ldots, k_\omega \leq D-L$ 表示节点级的翻转元，其中 $j_1 \leq \cdots \leq j_\omega, 1 \leq k_1, \ldots, k_\omega \leq D-L$ 是移位偏移量，且 D 取决于特殊节点类型。接下来，本文将详细阐述在 D-SCL-Flip 译码算法中 Rate-0、REP、Rate-1、SPC 和 Type I-V 节点处的快速译码方法。

3.6.1 Rate-0、REP、Type I、Type II 和 Type V 节点

根据定义，Rate-0、REP、Type I、Type II 和 Type V 节点中分别包含 $d=\{0,1,2,3,4\}$ 个信息比特。假设在译码当前节点之前存在 L 路径，快速 SCL 译码通过估计每个信息比特为 0 或者 1，可以得到总计 DL 条路径，其中 $D=2^d$。由于 Rate-0 节点不包含任何信息比特，因此在 Rate-0 节点处不会应用比特翻转策略。对于 REP、Type I、Type II 和 Type V 这些节点，如果在重新译码尝试期间发生翻转事件，则根据移位偏移量 k 从 $L+1$ 到 DL 的所有路径中选择 L 条路径，相应的路径剪枝函数具体如下所示：

$$H_{\mathcal{E}}(\mathcal{L}_{\{j,1\}}) = \begin{cases} h_0(\mathcal{L}_{\{j,1\}}), & \text{if } \{j,1,k\} \notin \mathcal{E}_\omega \\ h_k(\mathcal{L}_{\{j,1\}}), & \text{if } \{j,1,k\} \in \mathcal{E}_\omega \end{cases}, \tag{3-22}$$

另外，节点处的翻转度量根据下式进行更新：

$$M(\mathcal{E}_\omega) = \begin{cases} M(\mathcal{E}_{\omega-1}) \dfrac{\sum_{l=1}^{L} e^{-\alpha PM[\mathcal{E}_{\omega-1}]_{\{j_\omega,1\},l}}}{\sum_{l=1}^{EL} e^{-\alpha PM[\mathcal{E}_{\omega-1}]_{\{j_\omega,1\},l}}}, & \text{if } \{j,1,k\} \notin \mathcal{E}_\omega \\ M(\mathcal{E}_{\omega-1}) \dfrac{\sum_{l=k+1}^{k+L} e^{-\alpha PM[\mathcal{E}_{\omega-1}]_{\{j_\omega,1\},l}}}{\sum_{l=1}^{EL} e^{-\alpha PM[\mathcal{E}_{\omega-1}]_{\{j_\omega,1\},l}}}, & \text{if } \{j,1,k\} \in \mathcal{E}_\omega \end{cases}. \tag{3-23}$$

3.6.2 Rate-1、SPC、Type III 和 Type IV 节点

Rate-1、SPC、Type III 和 Type IV 节点的快速 SCL 译码算法基于球形译码算法[33-34]。在算法的每个估计步骤 $1 \leq i \leq E$ 中，每条路径分裂两条新路径，同时更新每条路径的 PM，然后算法根据 PM 值将 $2L$ 条候选路径进行排序，从而保留 PM 最小的 L 条路径。如果翻转事件 $\{j,i,k\}$ 发生在 Rate-1、SPC、Type III 或 Type IV 节点的译码过程中，则算法将在第 i 个估计步骤中选择 PM 最大的 L 条路径保留，相应的路径剪枝函数如下所示：

$$H_{\mathcal{E}}(\mathcal{L}_{\{j,i\}}) = \begin{cases} h_0(\mathcal{L}_{\{j,i\}}), & \text{if } \{j,i,L\} \notin \mathcal{E}_\omega \\ h_k(\mathcal{L}_{\{j,i\}}), & \text{if } \{j,i,L\} \in \mathcal{E}_\omega \end{cases}. \tag{3-24}$$

同时，节点处的翻转度量按下式更新：

$$M(\mathcal{E}_\omega) = M(\mathcal{E}_{\omega-1}) \prod_{\substack{i=1 \\ \{j,i,L\} \notin \mathcal{E}_\omega}}^{E} \frac{\sum_{l=1}^{L} e^{-\alpha PM[\mathcal{E}_{\omega-1}]_{\{j,i\},l}}}{\sum_{l=1}^{2L} e^{-\alpha PM[\mathcal{E}_{\omega-1}]_{\{j,i\},l}}} \prod_{\substack{i=1 \\ \{j,i,L\} \in \mathcal{E}_\omega}}^{E} \frac{\sum_{l=L+1}^{2L} e^{-\alpha PM[\mathcal{E}_{\omega-1}]_{\{j,i\},l}}}{\sum_{l=1}^{EL} e^{-\alpha PM[\mathcal{E}_{\omega-1}]_{\{j,i\},l}}}. \tag{3-25}$$

基于上述比特翻转策略和翻转度量，算法2提出了 Fast-D-SCL-Flip 译码算法，其中 Fast-SCL(·) 表示考虑 Rate-0、Rate-1、REP、SPC 和 Type I-V 节点的快速 SCL 译码尝试。值得说明的是，算法2与算法1相似，不同之处在于算法1逐位计算翻转度量，而算法2中以逐节点方式执行这个过程。

算法 2 Fast-D-SCL-Flip 译码算法

输入： y_1^N，L 和 T

输出： $\hat{u}_1^N$

1: 执行算法 1中的 1-6 行的步骤，使用 Fast-SCL(∅) 替换 SCL(∅)
2: **for** 特殊节点 j **do**
3: **if** 当前特殊节点是 REP、Type I、Type II 或者 Type V 节点 **then**
4: **for** $k = L, 2L, \ldots, D-L$ **do**
5: $\mathcal{E} = \{j, 1, k\}$，根据公式(3-23)计算 $M(\mathcal{E})$
6: $\mathcal{S} = \{\mathcal{S}, \mathcal{E}\}, \mathcal{M} = \{\mathcal{M}, M(\mathcal{E})\}$
7: **end for**
8: **else if** 当前特殊节点是 Rate-1、SPC、Type III 或者 Type IV 节点 **then**
9: **for** $i = 1, 2, \ldots, E$ **do**

10: $\mathcal{E} = \{j, i, L\}$，根据公式(3-25)计算 $M(\mathcal{E})$
11: $\mathcal{S} = \{\mathcal{S}, \mathcal{E}\}, \mathcal{M} = \{\mathcal{M}, M(\mathcal{E})\}$
12: **end for**
13: **end if**
14: **end for**
15: 执行算法 1中的 7-11 行的步骤对 $\mathcal{S}$ 和 $\mathcal{M}$ 进行初始化
16: **for** $t = 1, 2, \ldots, T$ **do**
17: 执行算法 1中的 13-18 行的步骤，使用 Fast-SCL($\mathcal{E}_t$) 替换 SCL($\mathcal{E}_t$)
18: **for** 特殊节点 j 且 $j \geq j_{\omega_t}$ **do**
19: **if** 当前特殊节点是 Rate-0、REP、Type I、Type II 或者 Type V 节点 **then**
20: **for** $k = L, 2L, \ldots, D - L$ **do**
21: $\mathcal{E} = \mathcal{E}_t \cup \{j, 1, k\}$，根据公式(3-23)计算 $M(\mathcal{E})$
22: 执行算法 1中的 21-23 行的步骤对 $\mathcal{S}$ 和 $\mathcal{M}$ 进行更新
23: **end for**
24: **else if** 当前特殊节点是 Rate-1、SPC、Type III 或者 Type IV 节点 **then**
25: **for** $i = 1, 2, \ldots, E$ **do**
26: $\mathcal{E} = \mathcal{E}_t \cup \{j, i, L\}$，根据公式(3-25)计算 $M(\mathcal{E})$
27: 执行算法 1中的 21-23 行的步骤对 $\mathcal{S}$ 和 $\mathcal{M}$ 进行更新
28: **end for**
29: **end if**
30: **end for**
31: **end for**
32: **return** $\hat{u}_1^N = \mathcal{R}[\mathcal{E}_t]_{N,1}$

3.7 数值分析与性能仿真

在本节中，本文通过仿真实验评估了本章所提的翻转度量、以及 D-SCL-Flip 和 Fast-D-SCL-Flip 译码算法的性能和复杂度。本章的仿真设置如表3.1所示。除非另有说明，对于所有考虑的译码算法，默认列表大小 L 设置为 4，且 D-SCL-Flip 译码算法使用公式(3-6)中的翻转度量。

表 3.1 仿真设置

仿真环境	MATLAB R2018b、Python 3.6、TensorFlow 1.13.1，Intel i5-9400 CPU@2.9 GH 16G RAM、NVIDIA GeForce GTX 2080 Ti GPU
极化码参数	$\mathcal{P}(256, 128+8)$，$L \in \{4, 8, 16\}$，GA 构造法
CRC 生成多项式	$x^8+x^7+x^6+x^5+x^4+x^3+1$
训练集样本信道环境	$E_b/N_0 = 2$ dB
训练集样本数	10^4
训练轮次数	50
批量大小	200
学习率	10^{-3}
优化器	Adam

3.7.1 译码性能和复杂度对比

首先，为了探究本章第 2 节中的扰动参数 α 对译码性能的影响，图3.3绘制了 D-SCL-Flip 译码算法使用不同的 α 值与使用 $\alpha = 1$ 时的 FER 比值，其中 $T = 10$。当 $\alpha = 0.5$ 时，D-SCL-Flip 译码算法取得了最佳的译码性能。文献 [19] 指出，在选择相同 α 的情况下，改变极化码参数（如 L、N 和 R）不会影响 SCL-Flip 译码算法的性能。考虑到 SCL-Flip 和 D-SCL-Flip 两者的相似性，本文假设 D-SCL-Flip 译码算法中最佳的 α 同样与 L、N 和 R 无关。因此，本章后续所有仿真统一设置 $\alpha = 0.5$。

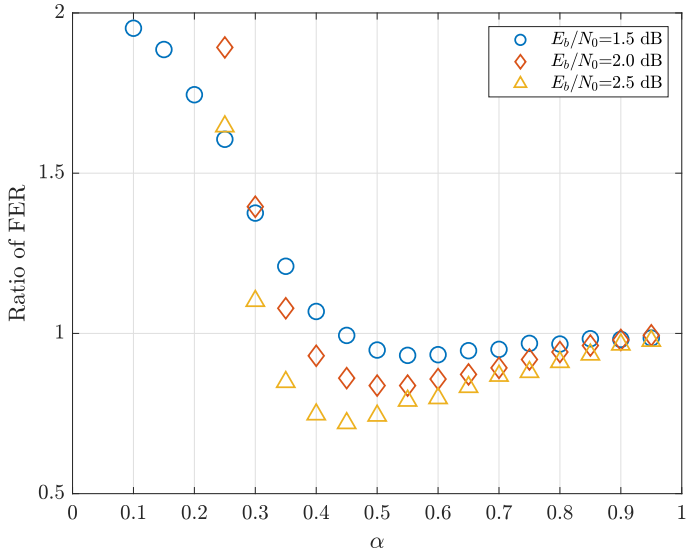


图 3.3 扰动参数对译码性能的影响

图3.4对比了 D-SCL-Flip、SCL-Flip 和 SCL-Flip-ω（$\omega = 2$）译码算法的 FER 性能。对于所有考虑的译码算法，最大译码尝试次数设置为 $T = 50$ 和 $T = 200$，其中在 SCL-

Flip-ω 译码算法中，$T = 50$ 的参数设置为 $T_1 = 25, T_2^1 = T_2^2 = 5$，$T = 200$ 的参数设置为 $T_1 = 100, T_2^1 = T_2^2 = 10$。此外，本节引入完美的 SCL-Flip-$\omega$ 译码算法，即图中标注的 OA-SCL-Flip-ω，该算法始终能够成功纠正至多 ω 次译码错误。如图3.4所示，D-SCL-Flip 译码算法的性能超越了 OA-SCL-Flip-1 译码性能，例如在 FER $= 10^{-3}$ 处，D-SCL-Flip 译码算法可以取得高达 0.3dB 的性能增益。这证明所提的 D-SCL-Flip 译码算法可以纠正高阶的 SCL 译码错误。另外，可以观察到 D-SCL-Flip 的译码性能同样优于 SCL-Flip-ω 的译码性能，同时 SCL-Flip-ω 译码算法的性能严重依赖于参数设置，而所提出的 D-SCL-Flip 不需要精心设置各种复杂的参数。即便使用最佳参数设置，SCL-Flip-ω 译码算法的性能也略差于 D-SCL-Flip 译码算法的性能，这意味着本章提出的动态翻转度量能够更准确地纠正高阶译码错误。

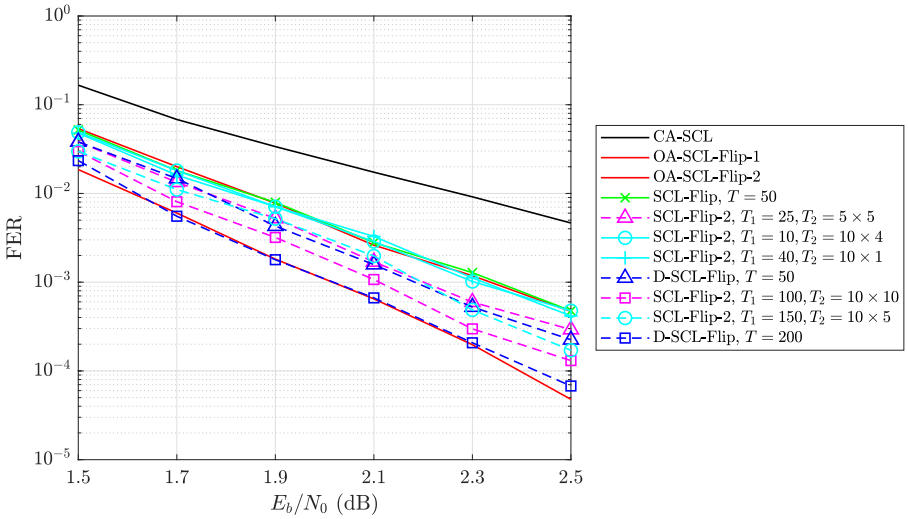


图 3.4 不同译码算法的 FER 性能对比图

然后，图3.5对比了各种译码算法需要进行译码尝试的平均次数，其中译码算法的参数与图3.4中相同。在最大译码尝试次数 T 相同的情况下，与 SCL-Flip 和 SCL-Flip-ω 译码算法相比，D-SCL-Flip 译码算法不仅取得了更好的 FER 性能，而且需要更少的译码尝试次数，这意味着该算法的译码复杂度和时延更低。特别是，在 $T = 200$ 且 $E_b/N_0 = 2.5$ dB 时，所提出的 D-SCL-Flip 译码算法所需译码尝试的平均次数比 SCL-Flip-ω 译码算法少 34.6%，这说明了本章利用翻转度量动态构建翻转集合方法在复杂度方面同样具有优越性。

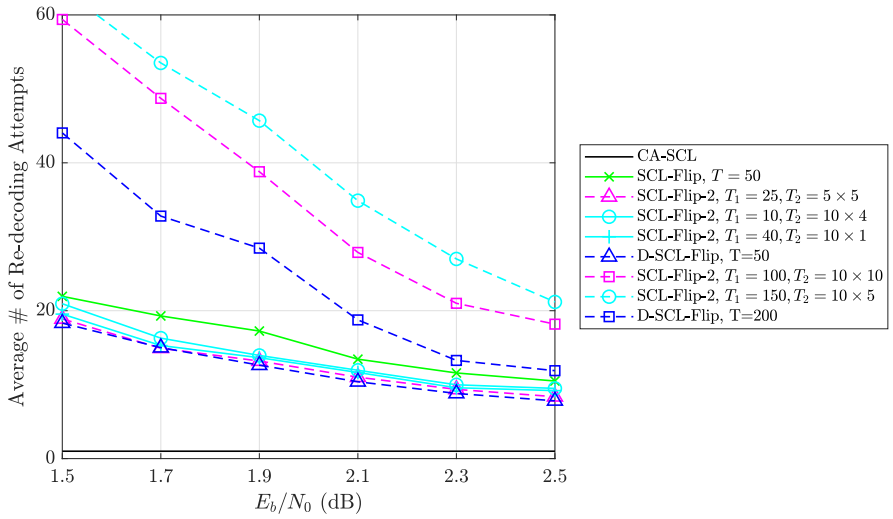


图 3.5 不同译码算法的复杂度对比图

3.7.2 比特翻转策略对比

如本章第 2 节中的注 1 所述，通过将公式(2-24)中的移位剪枝策略和公式(3-11)中翻转度量应用于算法1，可以获得动态 SCL 移位剪枝译码算法（D-SCL-SP）。图3.6比较了所提出的 D-SCL-SP 和 D-SCL-Flip 译码算法的性能，其中 D-SCL-SP 译码算法中的移位偏移量设置为 $k = L/2$。从该图中可以观察到，对于所有考虑的 T，D-SCL-Flip 的译码性能都优于 D-SCL-SP 的译码性能，这表明与移位剪枝策略不如反向路径选择策略有效。

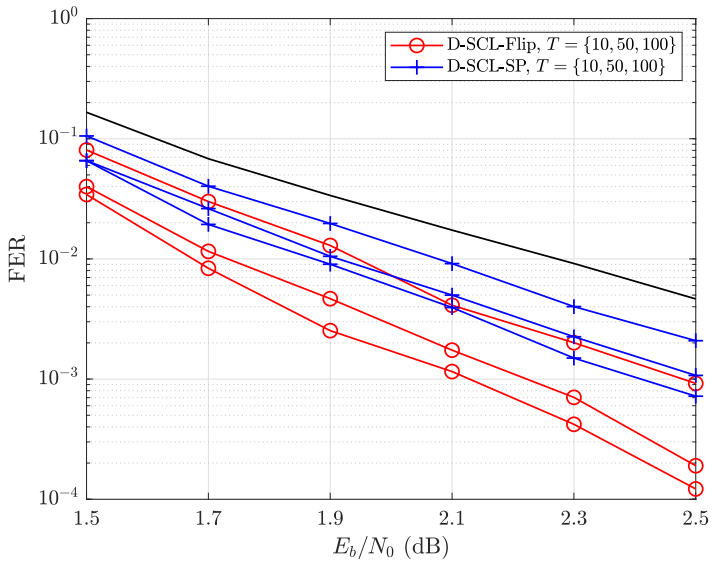


图 3.6 采用不同比特翻转策略的译码性能对比图

3.7.3 翻转度量对比

为了验证各种翻转度量的有效性，本节重点关注在 T 次重新译码尝试内 D-SCL-Flip 译码器纠正 SCL 译码错误的概率，该概率可以由 i_E 出现在翻转集合中的频率统计。在 $E_b/N_0 = 2.0$ dB 时，图3.7比较了各种翻转度量的预测精度，其中 $T \in \{3, 10, 20, 30\}$。可以观察到，文献 [17] 提出的名为关键集合的预测精度比其他翻转度量的预测精度更低，这是因为静态翻转集合没有充分利用 SCL 译码中的软信息。此外，文献中 M_{std}（见公式(2-26)）和 $M_{\text{max-log}}$（见公式(2-27)）具有相似的预测精度，并且性能都不如本文提出的各个翻转度量。此外，还值得一提的是，$T = 10$ 时的 M_{DL} 所实现的预测精度为 79.78%，与 $T = 30$ 时静态关键集合的预测精度相当，这表明通过采用高效的翻转度量，可以显著减少重新译码尝试的次数。

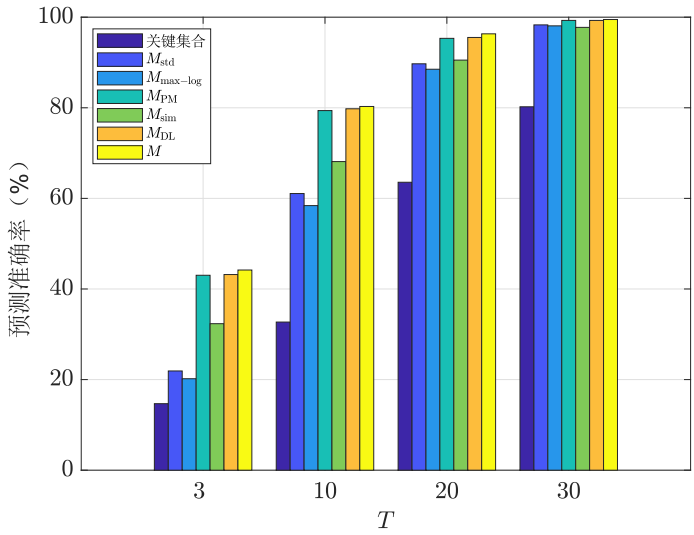


图 3.7 不同翻转度量的预测准确度对比图

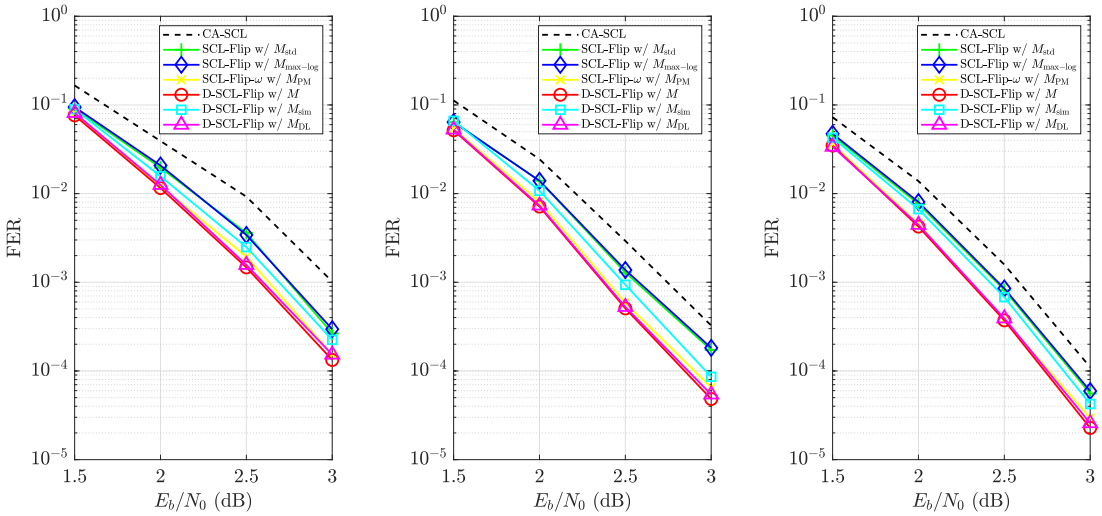
表3.2统计了不同翻转度量的复杂度，包括翻转度量在一次译码尝试期间所需的指数/对数计算次数以及内存空间。如表3.2所示，M_{std} 和 $M_{\text{max-log}}$ 的计算复杂度随 L 和 K 增加。此外，这两个翻转度量需要额外的存储器空间来存储信息比特的 LLR。另一方面，M_{PM} 和本章所提的各个翻转度量利用了实时的 PM 信息，因此它们不需要额外的内存成本。然而，M_{PM} 需要在每个信息比特处计算超越函数而仍然具有较高的计算复杂度，相比之下本文提出的 M_{sim} 和 M_{DL} 具有更低的计算复杂度而更利于硬件实现。

图3.8比较了 $L \in \{4, 8, 16\}$ 时 CA-SCL、D-SCL-Flip、SCL-Flip 和 SCL-Flip-ω 译码算法的 FER 性能，其中最大重新译码尝试次数 T 均设置为 10。如图3.8所示，使用预测精

表 3.2　不同翻转度量的复杂度统计

	指数/对数操作	内存
M_{std}	$> L(K+r-\log_2 L)(K+r+1+\log_2 L)$	$L(K+r-\log_2 L)$
$M_{\text{max-log}}$	$(K+r-\log_2 L)(K+r+1+\log_2 L)$	$L(K+r-\log_2 L)$
M_{PM}	$(2L+2)(K+r-\log_2 L)$	0
M	$L(K+r-\log_2 L)(K+r+1+\log_2 L)$	0
M_{sim}	0	0
M_{DL}	0	0

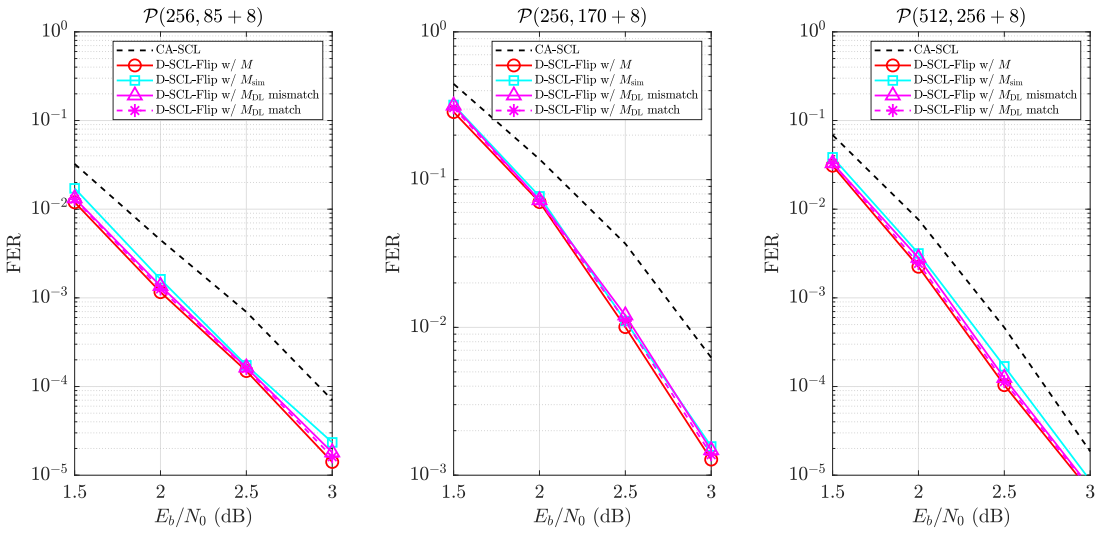
度更高的翻转度量可以获得更好的译码性能，其中本文提出的 M_{sim} 性能不如 M_{PM}，但它仍然优于两个基于 LLR 的翻转度量 M_{std} 和 $M_{\text{max-log}}$。与其他翻转度量相比，M_{DL} 取得了最佳性能，且具有最低的计算复杂度，这表明本章提出的 DL 方法是有效的。



(a) $L=4$　　(b) $L=8$　　(c) $L=16$

图 3.8　采用不同翻转度量的译码性能对比图

当极化码使用不同参数时，图3.9比较了 D-SCL-Flip 译码算法的 FER 性能，其中图例“匹配”表示训练集和性能测试时使用的极化码参数相同，而“不匹配”则表示两者参数不同。如图3.9所示，当码率 R 发生变化时，采用 M_{DL} 和 M_{PM} 能够取得相近的译码性能。从图3.9 (c) 中可以观察到，尽管训练集数据对应极化码 $\mathcal{P}(256, 128+8)$，但这种情况下训练得到的 M_{DL} 应用于极化码 $\mathcal{P}(512, 256+8)$ 并不会造成性能退化。因此，本章提出的 DL 方法对极化码码长和码率的变化具有鲁棒性，即当极化码参数发生变化时不需要重新训练 M_{DL} 中的参数 $\boldsymbol{W}$。



(a) $R = 1/3$ (b) $R = 2/3$ (c) $N = 512$

图 3.9 采用不同极化码参数的译码性能对比图

3.7.4 快速译码增益

最后，图3.10比较了 CA-Fast-SCL、D-SCL-Flip 和 Fast-D-SCL-Flip 译码算法的 FER 性能，其中 $T \in \{10, 50, 200\}$。从该图中可以观察到，与 D-SCL-Flip 译码器相比，采用快速译码技术的 Fast-D-SCL-Flip 译码器会导致小于 0.2 dB 的性能退化，这主要是因为节点级译码和比特级译码是不等效的。

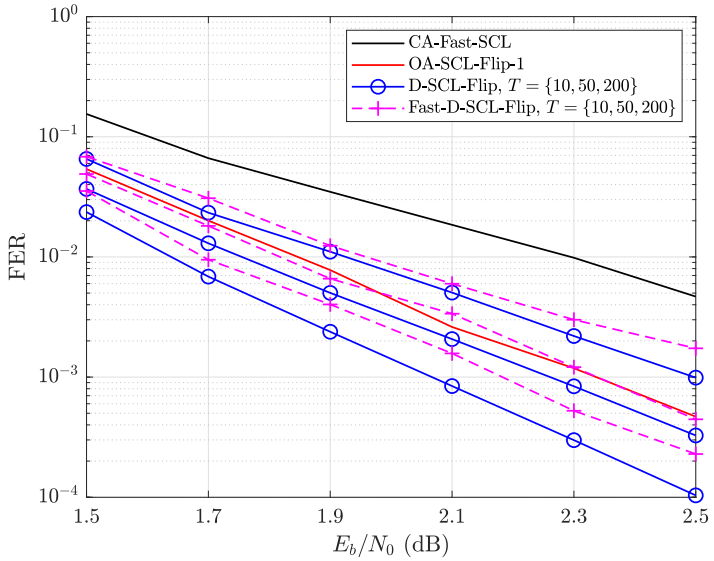


图 3.10 快速译码算法的译码性能对比图

图3.11进一步比较了各个译码算法所需的平均译码时间步长，其中每次译码尝试中

各个特殊节点译码所需的时间步长数是根据文献[34]中的表 I 计算得到的，而整体的译码时间步长等于译码尝试次数与单次译码尝试所需的时间步长的乘积。可以看出，通过在特殊节点上应用快速译码技术，可以节省相当多的译码时延。当 $T = 10$ 时，相比 D-SCL-Flip 译码算法，Fast-D-SCL-Flip 译码算法可以节省 89.9% 的平均译码时延。此外，在 E_b/N_0 较高的情况下，Fast-D-SCL-Flip 的平均译码时延接近 CA-Fast-SCL 的译码时延，这表明 Fast-D-SCL-Flip 译码算法可以提供显著的性能增益，同时在 E_b/N_0 较高时其平均译码时延很低。

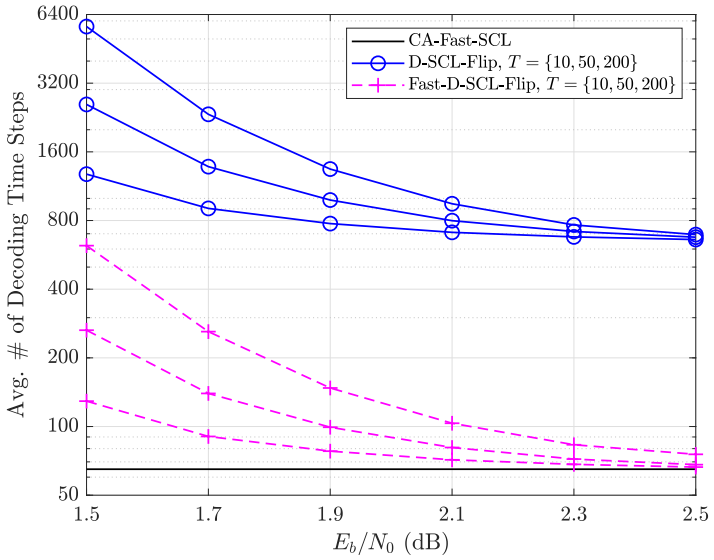


图 3.11 快速译码算法的平均译码时延对比图

3.8 本章小结

本章面向高可靠通信传输场景，提出了极化码高性能的动态 SCL 翻转译码算法。首先本章提出了一种翻转度量用以评估翻转比特能够成功纠正 SCL 译码错误的概率。接着基于该翻转度量进一步提出了动态的 SCL 翻转译码算法，该算法能够纠正多个 SCL 译码错误从而提升了 CA-SCL 的译码性能。考虑到实际的计算复杂度，本章进一步对所提出的翻转度量作了化简，并利用 DL 技术弥补化简带来的译码性能损失。最后，为了将快速译码技术应用于所提的译码算法中，本章提出了节点级的比特翻转策略和翻转度量。仿真结果表明相比现有的基于比特翻转的 SCL 译码算法，本章提出的各个译码算法在性能、复杂度和时延方面均具有较明显的提升。

第 4 章 基于序列节点的快速 SC 译码算法

4.1 引言

目前，基于 SC 的译码算法是极化码最主流的译码方式，然而其串行译码的特性引起了较大的时延，导致极化码在低时延场景中的应用受限。因此，对于基于 SC 的译码算法，加速译码过程并降低时延是一个重要的课题。本章提出了一种新的快速 SC 译码算法，以进一步降低极化码的译码时延。首先，本章介绍了一类新的特殊节点——SR1/SPC 节点，它由源节点和一系列 Rate-1 或 SPC 节点组成，从而提供了对多种现有特殊节点的统一描述。然后，本章分析了由所提 SR1/SPC 节点中冻结比特引入的奇偶约束。最后，基于该约束条件，本章提出了针对 SR1/SPC 节点的快速译码算法，仿真结果表明利用该算法译码 SR1/SPC 节点可以取得近似 ML 的纠错性能，同时能够大大降低 SC 译码的整体时延。

4.2 快速 SC 译码

在 SC 译码中，每个比特估计取决于所有先前的比特估计，这样的顺序性质会导致很高的译码时延。SC 的译码速度可以通过部署并行译码器得到提升，这些并行译码器不需要顺序遍历整个译码树，它们可以在译码树的中间层节点处直接输出多个比特的估计码字。

最近，文献 [30] 提出了一类新的特殊节点——SR0/REP 节点，它能够包括大多数现有的低码率特殊节点。对于位于译码树 p 层的 SR0/REP 节点，其所有左子节点都是 Rate-0 或者 REP 节点，而在 q 层的右子节点是一个通用的源节点。图4.1展示了 SR0/REP 节点的一般结构，其中 $\mathcal{N}_q^{R_q}$ 代表位于 q 层的源节点，$R_q = 2^{p-q}i$。值得注意的是，若 SR0/REP 节点中除源节点之外所有的子节点都是 Rate-0 节点，则 SR0/REP 节点将退化为 G-REP 节点[29]。

SR0/REP 节点可以基于其源节点的码字来译码。$\boldsymbol{s}_l$ 表示下式确定的重复序列：

$$\boldsymbol{s}_l = (\eta_q, 1) \otimes (\eta_{q+1}, 1) \otimes \cdots \otimes (\eta_{p-1}, 1), \qquad l \in \{1, 2, \ldots, |\mathbb{S}|\}, \tag{4-1}$$

其中 $\mathbb{S}$ 表示包含所有可能的 $\boldsymbol{s}_l$ 的集合，该集合可以在译码之前基于 SR0/REP 节点结构

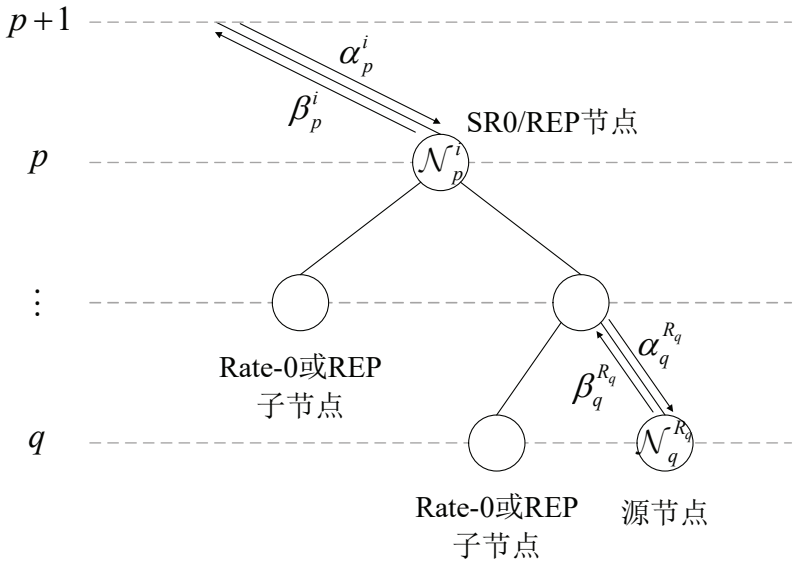


图 4.1 SR0/REP 节点的一般结构

预先确定，η_r 代表 Rate-0/REP 子节点 $\mathcal{N}_r^{R_r-1}$ 最后一位的比特估计值，即：

$$\eta_k = \begin{cases} 1, & \text{若 } \mathcal{N}_r^{R_r-1} \text{ 是 Rate-0 节点} \\ 1-2\beta_r^{R_r-1}[2^r], & \text{若 } \mathcal{N}_r^{R_r-1} \text{ 是 REP 节点} \end{cases}, \tag{4-2}$$

令 $\alpha_{q_l}^{R_q}[1:2^p]$ 表示当使用重复序列 $\boldsymbol{s}_l$ 时源节点的 LLR 向量，其计算公式如下所示：

$$\alpha_{q_l}^{R_q}[k] = \sum_{m=1}^{2^{p-q}} \alpha_p^i[(m-1)2^q + k]s_l[m], \qquad k \in \{1, 2, \ldots, 2^q\}. \tag{4-3}$$

源节点最佳的 LLR 向量的索引 $\hat{l}$ 可以由下式确定：

$$\hat{l} = \underset{l\in\{1,\ldots,|\mathbb{S}|\}}{\arg\max} \sum_{k=1}^{2^q} |\alpha_{q_l}^{R_q}[k]|, \tag{4-4}$$

随后，利用 $\alpha_{q_l}^{R_q}[1:2^p]$ 对源节点进行译码。具体地，如果源节点具有特殊结构，则可以使用第 2 章第 3 节中现有的快速译码技术，否则使用传统 SC 译码。最后，SR0/REP 节点的译码结果通过以下方式获得：

$$\beta_p^i[k:2^q:2^p] = \beta_q^{R_q}[k] \oplus \boldsymbol{s}_{\hat{l}}, \qquad k \in \{1, 2, \ldots, 2^q\}. \tag{4-5}$$

上述 SR0/REP 节点统一了大多数低码率的特殊节点，其快速译码算法的思想源于 ML 译码，即以穷举的方法估计每一个 REP 子节点中的信息比特。然而目前高码率特殊节点还没有通用的描述，另外由于这些特殊节点包含更多的信息比特，信息与冻结比特

分布模式更加复杂，导致高度并行的基于 ML 译码的方法计算复杂度太高。因此，如何统一描述高码率的特殊节点，并为统一的节点设计通用高效的译码算法，是目前快速译码的一个挑战性的难题。

4.3 SR1/SPC 节点

假设 SR1/SPC 节点位于译码树的第 p 层，那么其位于 r 层（$q \leq r < p$）上的右子节点都是 Rate-1 或 SPC 节点，在 q 层的左子节点被称为源节点（码率为 C 的通用节点）。为了便于说明，SR1/SPC 节点的结构如图 4.2所示。

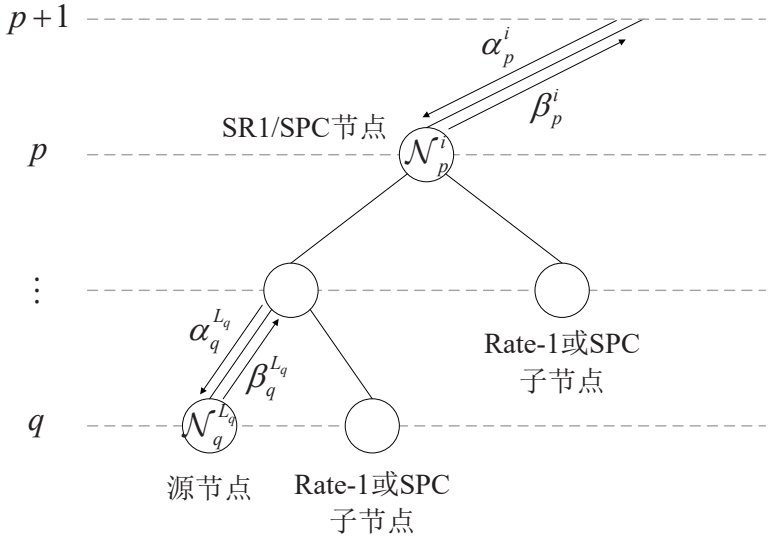


图 4.2 SR1/SPC 节点的一般结构

SR1/SPC 节点的结构可以由三个关键参数 p、q 和 $\mathbb{R}$ 表示，其中 p 是根节点层数，q 是源节点层数，而 $\mathbb{R}$ 表示由每个 SPC 子节点的层数索引组成的集合，即

$$\mathbb{R} = \{r | \mathcal{N}_r^{L_r+1} \text{ 是 SPC 节点，其中 } q \leq r < p\}.$$

现有快速 SC 译码算法的时延随着 $p-q$ 的增加而线性增加，因此本文将 $d = p - q$ 定义为 SR1/SPC 节点的深度，以反映该节点的并行度。此外，在不失一般性的情况下，本文假设集合 $\mathbb{R}$ 中层数索引按升序排序。例如，$\mathcal{P}(32, 27)$ 极化码可以表示为 $p = 5$、$q = 2$、$\mathbb{R} = \{2, 3\}$ 的 SR1/SPC 节点

$$\boldsymbol{c} = \{\overbrace{0,0,0,1}^{\text{REP}}, \overbrace{0,1,1,1}^{\text{SPC}}, \overbrace{0,1,\ldots,1}^{\text{SPC}}, \overbrace{1,\ldots,1}^{\text{Rate-1}}\}.$$

由于源节点是具有任意形式的通用节点，因此所提出的 SR1/SPC 节点可以包括多种特殊节类型点。如表4.1所示，大多数现有的高码率特殊节点都可以视为 SR1/SPC 节

点的特殊形式。然而，现有的特殊节点要么并行度较低，要么出现的频率（极化码中的分布频率）较低，这两者都会导致应用这些节点能够提升的译码吞吐量有限。值得注意的是，在 $\mathbb{R}=\{q,q+1,\ldots,p-1\}$ 或 $\mathbb{R}=\emptyset$ 的情况下，SR1/SPC 节点将分别退化为序列 Rate-1（Sequence Rate-1, SR1）或序列 SPC（Sequence SPC, SSPC）节点。

表 4.1 SR1/SPC 节点的特殊形式

节点类型	节点结构	
	节点参数	源节点
P-01[44]	$q=p-1$、$\mathbb{R}=\emptyset$	Rate-0
P-0SPC[44]	$q=p-1$、$\mathbb{R}=\{p-1\}$	Rate-0
P-R1[44]	$q=p-1$、$\mathbb{R}=\emptyset$	Rate-C
P-RSPC[44]	$q=p-1$、$\mathbb{R}=\{p-1\}$	Rate-C
Type III[28]	$q=1$、$\mathbb{R}=\emptyset$	Rate-0
Type IV[28]	$q=2$、$\mathbb{R}=\emptyset$	REP
G-PC[29]	$\mathbb{R}=\emptyset$	Rate-0
RG-PC[29]	$\mathbb{R}\neq\emptyset$	Rate-0
EG-PC[30]	$\mathbb{R}=\emptyset$	Rate-0/REP
SR1	$\mathbb{R}=\emptyset$	Rate-C
SSPC	$\mathbb{R}=\{q,q+1,\ldots,p-1\}$	Rate-C

4.4 SR1/SPC 节点的对偶约束

文献 [29] 指出，译码树叶子结点的每个冻结比特将对中间层节点的码字施加奇偶约束。这意味着特殊节点根节点处的码字需要满足奇偶约束以确保其合法有效，节点码字不满足奇偶约束会导致译码性能退化。例如，RG-PC 的节点译码忽略了子节点中冻结比特带来的奇偶约束，导致整体的 SC 译码在高 SNR 场景下纠错性能显著退化[29]。本节详细分析了施加在 SR1/SPC 根节点处的奇偶约束，合法的译码码字应该满足这些奇偶变量，从而保证译码性能不会出现严重退化。

第一种对偶约束由源节点引起，其具体形式由如下定理所示：

定理 4.1. 对于 *SR1/SPC* 节点 $\mathcal{N}_p^i$，位于译码树 q 层的源节点将对 p 层的根节点施加以下 2^q 个平行对偶约束（*Parallel Parity Constraint, P-PC*）：

$$\bigoplus_{j=1}^{2^d}\beta_p^i[(j-1)2^q+k]=\beta_q^{L_q}[k],\qquad k\in\{1,2,\ldots,2^q\}. \tag{4-6}$$

证明. 详见附录A。 □

特别地，对于 SR1 节点，冻结比特仅分布于源节点中，因此 SR1 节点只包含 P-PC

这一种类型的对偶约束。对于一般情况下的 SR1/SPC 节点，还存在由 SPC 子节点引起的第二种对偶约束，该约束可以由如下定理表示：

定理 4.2. 对于 *SR1/SPC* 节点 $\mathcal{N}_p^i$，位于译码树 r 层的 *SPC* 子节点 $N_r^{L_r+1}$ 将对 p 层的根节点施加以下的分段对偶约束（*Segmental Parity Constraint, S-PC*）：

$$\bigoplus_{j=1}^{2^{p-r-1}} \bigoplus_{k=1}^{2^r} \beta_p^i[(2j-1)2^r+k] = 0. \tag{4-7}$$

证明. 详见附录B。 □

为了便于说明，图4.3描述了以 $p=5$、$q=2$、$\mathbb{R}=\{2,3\}$ 为例的 SR1/SPC 节点包含的所有 P-PC 和 S-PC。可以观察到，长度为 4 的源节点在根节点上施加了 4 个 P-PC，而位于 2、3 层的 SPC 子节点分别引入了两个独立的 S-PC。

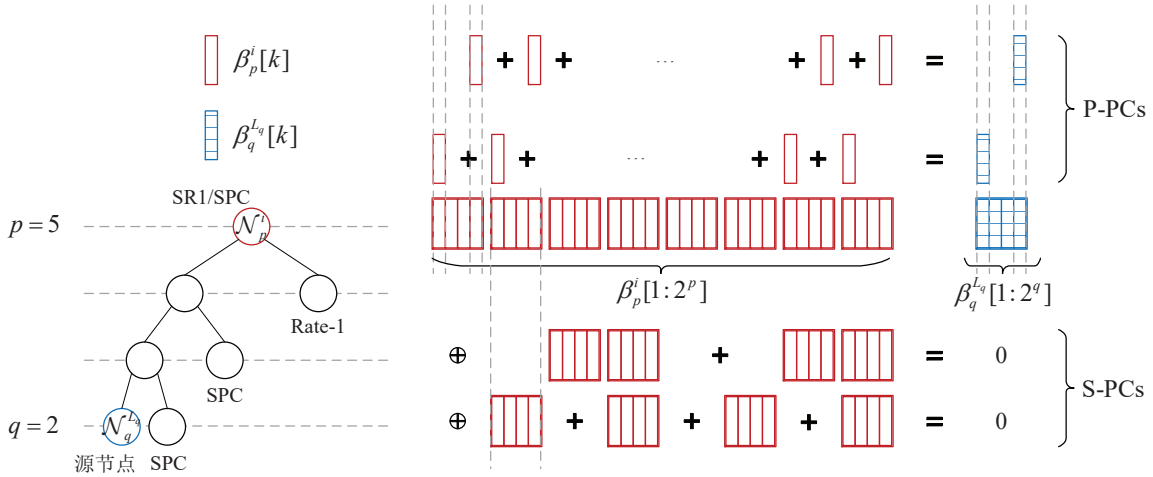


图 4.3　SR1/SPC 节点 $\mathrm{NS}(5,2,\{2,3\})$ 中包含的对偶约束

4.5　基于 SR1/SPC 节点的 SC 快速译码算法

4.5.1　整体方案

定理4.1和4.2中的结果表明，SR1/SPC 节点译码输出是码字应同时满足 P-PC 和 S-PC，一种简单的译码方法思想是通过翻转硬判决码字中的比特使得奇偶约束满足。然而，根据公式(4-6)和公式(4-7)，每个 S-PC 依赖于 P-PC 和其他 S-PC，导致码字中每一个比特可能会同时与 P-PC 和 S-PC 相关，从而为高效译码的实现带来了困难。

为了解决这个问题，本节提出了一种高度并行的 SR1/SPC 节点译码算法，该算法通过两个阶段的译码过程以逐步满足 P-PC 和 S-PC，每个阶段采用比特翻转的方式对根

节点 LLR 硬判决码字进行调整以获得候选的估计码字。算法的第一阶段首先忽略 S-PC，根据 P-PC 的校验值翻转码字中最不可靠的比特，从而顺利校正所有 P-PC。在第二阶段中，算法在不违反 P-PC 的情况下校正 S-PC，产生了一组同时满足所有的奇偶约束的候选估计码字，接着本文提出了一种惩罚度量衡量每个候选估计码字的可靠性，最终算法选择惩罚度量最小的候选码字作为译码输出，取得了接近 ML 的译码性能。值得说明的是，现有的高度并行的 ML 译码变体，例如猜测随机加性噪声译码算法[45]和综合症检验译码算法[46]也可以取得接近 ML 译码的性能，但通常这些算法的计算复杂度非常高。

4.5.2 第一阶段：纠正 P-PC

在算法的第一阶段中，首先将 SR1/SPC 节点的 LLR 和码字分成如下 2^q 部分：

$$\begin{aligned} \alpha_k^{\text{P-PC}}[1:2^d] &= \alpha_p^i[k:2^q:2^p], \qquad k \in \{1,2,\ldots,2^q\}, \\ \beta_k^{\text{P-PC}}[1:2^d] &= \beta_p^i[k:2^q:2^p], \qquad k \in \{1,2,\ldots,2^q\}. \end{aligned} \tag{4-8}$$

其中 $\alpha_k^{\text{P-PC}}[1:2^d]$ 表示与第 k 个 P-PC 相关联的 LLR 子向量，$\beta_k^{\text{P-PC}}[1:2^d]$ 表示 $\alpha_k^{\text{P-PC}}[1:2^d]$ 的对应码字。然后，源节点的 LLR 向量 $\alpha_q^{L_q}[1:2^q]$ 可以通过以下式获得：

$$\alpha_q^{L_q}[k] = \prod_{j=1}^{2^d} \operatorname{sgn}(\alpha_k^{\text{P-PC}}[j]) \min(|\alpha_k^{\text{P-PC}}[1:2^d]|), \qquad k \in \{1,2,\ldots,2^q\}, \tag{4-9}$$

随后，通过传统的 SC 译码或快速 SC 译码算法对源节点进行译码，以获得码字 $\beta_q^{L_q}[1:2^q]$。具体地，如果源节点具有特殊结构，则利用该节点对应的快速 SC 译码算法得到码字；如果源节点仅具有通用的结构，则利用传统的 SC 译码算法得到码字。根据定理4.1，根节点中每个 LLR 子向量 $\alpha_k^{\text{P-PC}}[1:2^d]$ 对应的比特组成了一个 SPC 子码[1]，并且相应码字所有比特的异或和应该满足奇偶校验值 $\beta_q^{L_q}[k]$。利用 Wagner 译码器可以对这些 SPC 子码进行译码以获得根节点码字，具体过程如下。对于第 k 个 SPC 子码，用 $\gamma_{\text{P-PC}}[k]$ 和 $\rho[k]$ 分别表示第 k 个 P-PC 的奇偶校验值和最不可靠的比特位置，它们可以分别由下式确定：

$$\gamma_{\text{P-PC}}[k] = \bigoplus_{j=1}^{2^d} \text{HD}(\alpha_k^{\text{P-PC}}[j]) \oplus \beta_q^{L_q}[k], \tag{4-10}$$

$$\rho[k] = \underset{j \in \{1,\ldots,2^d\}}{\arg\min} |\alpha_k^{\text{P-PC}}[j]|, \tag{4-11}$$

[1]与传统 SPC 极化码子码的定义略有不同，这里的“SPC”实际上指的是广义 SPC 子码。传统的 SPC 子码需要满足偶数校验，而广义 SPC 子码的校验值可以是偶数或奇数，这取决于 $\beta_q^{L_q}[k]$ 的值。

最后，利用 2^q 个 Wagner 译码器对硬判决码字应用比特翻转操作，获得满足所有 P-PC 的初步估计码字，每个 Wagner 译码器得到的码字如下所示：

$$\beta_k^{\text{P-PC}}[j] = \begin{cases} \text{HD}(\alpha_k^{\text{P-PC}}[j]) \oplus \gamma_{\text{P-PC}}[k], & \text{若 } j = \rho[k] \\ \text{HD}(\alpha_k^{\text{P-PC}}[j]), & \text{否则,} \end{cases} \tag{4-12}$$

算法3总结了校正 P-PC 的步骤。在算法3中，首先根据源节点类型（第 1-6 行）对其进行译码。如果源节点是 Rate-0 节点，那么可以直接获得它的码字（第 2 行）；否则，执行 LLR 计算的步骤（第 4 行），然后执行特定译码过程以获得源节点的码字（第 5 行）。由于源节点可能由一个或多个特殊节点组成，因此可以应用文献 [26-30,42] 中的快速 SC 译码技术。最后，在第 7 行中，根据公式(4-12)并行执行 2^q 个 Wagner 译码器。值得注意的是，SR1 节点由于不包含 S-PC 可以直接通过算法3得到最终的译码结果。

算法 3 校正 P-PC

输入: $\alpha_p^i[1:2^p]$

输出: $\beta_p^i[1:2^p]$

1: **if** $\mathcal{N}_q^{L_q}$ 是 Rate-0 节点 **then**

2: $\beta_q^{L_q}[1:2^q] = \mathbf{0}$

3: **else**

4: 根据公式(4-9)计算 $\alpha_q^{L_q}[1:2^q]$

5: 对节点 $\mathcal{N}_q^{L_q}$ 进行译码获取 $\beta_q^{L_q}[1:2^q]$

6: **end if**

7: 根据公式(4-12)获取 $\beta_p^i[1:2^p]$

8: **return** $\beta_p^i[1:2^p]$

4.5.3 第二阶段：纠正 S-PC

算法的第二阶段进行 S-PC 的纠正。与公式(4-8)类似，SR1/SPC 节点的 LLR 和码字被划分为如下 2^d 个部分：

$$\begin{aligned} \alpha_k^{\text{S-PC}}[1:2^q] &= \alpha_p^i[(k-1)2^q+1:k2^q], \qquad k \in \{1,2,\ldots,2^d\}, \\ \beta_k^{\text{S-PC}}[1:2^q] &= \beta_p^i[(k-1)2^q+1:k2^q], \qquad k \in \{1,2,\ldots,2^d\}, \end{aligned} \tag{4-13}$$

其中，k 表示段索引，$\alpha_k^{\text{S-PC}}[1:2^q]$ 表示从 $(k-1)2^q+1$ 到 $k2^q$ 这些比特位置的 LLR 子向量，$\beta_k^{\text{S-PC}}[1:2^q]$ 表示对应 $\alpha_k^{\text{S-PC}}[1:2^q]$ 的码字。对于深度为 d 的 SR1/SPC 节点，令 $\gamma_{\text{S-PC}}[1:d]$ 表示 S-PC 的奇偶校验向量，根据定理4.2，该向量的计算公式如下：

$$\gamma_{\text{S-PC}}[t]=\begin{cases}\bigoplus_{j=1}^{2^{d-t}}\bigoplus_{k=1}^{2^{q+t-1}}\beta_p^i[(2j-1)2^{q+t-1}+k]=\bigoplus_{j=1}^{2^{d-t}}\bigoplus_{k=(2j-1)2^{t-1}+1}^{j2^t}\bigoplus_{m=1}^{2^q}\beta_k^{\text{S-PC}}[m] & \\ & \text{若 } \mathcal{N}_r^{L_r+1} \text{ 是 SPC 节点} \\ -1, & \text{若 } \mathcal{N}_r^{L_r+1} \text{ 是 Rate-1 节点}\end{cases}, \tag{4-14}$$

其中 $t=r-q+1\in\{1,2,\ldots,d\}$。$\gamma_{\text{S-PC}}[t]$ 能够指示码字是否满足第 t 个 S-PC，满足的 S-PC 导致偶校验，而不满足的则导致奇校验。特别地，当 SR1/SPC 节点中的 SPC 子节点 $\mathcal{N}_r^{L_r+1}$ 是 Rate-1 节点时，用 -1 表示此子节点不会引入 S-PC。

为了保证第二阶段的码字依旧满足第 m 个 P-PC，需要同时翻转码字中任意偶数个段中的第 m 位比特，其中 $m\in\{1,2\ldots2^q\}$。本文使用 $\mathcal{E}=\{k_1,k_2,m\}$ 表示翻转坐标，用以指示码字中需要翻转的一对比特位置，即 $\beta_{k_1}^{\text{S-PC}}[m]$ 和 $\beta_{k_2}^{\text{S-PC}}[m]$，其中 $k_1,k_2\in\{1,2,\ldots,2^d\}$ 是段索引且 $k_1<k_2$。为了简单起见，本节暂时省去了比特索引 m，后文中 $\mathcal{E}=\{k_1,k_2,m\}$ 将简单表示为 $\mathcal{E}=\{k_1,k_2\}$。通过翻转第一阶段估计码字中翻转坐标指示的比特位置，能够保持 P-PC 不被违反。接下来，本节将进一步考虑利用翻转坐标纠正所有 S-PC，且将能够纠正所有 S-PC 的翻转坐标称作可行翻转坐标。

如上所述，可行的翻转坐标应当纠正具有奇校验的 S-PC，同时保持具有偶校验的 S-PC。给定一个可行的翻转坐标 $\mathcal{E}=\{k_1,k_2\}$，对于奇校验的 S-PC，k_1 和 k_2 两个段索引中只有一个与该 S-PC 相关，这样翻转码字中 $\mathcal{E}$ 指示的比特可以将该 S-PC 从奇校验纠正为偶校验；而对于偶校验的 S-PC，k_1 和 k_2 应该同时与该 S-PC 相关或者不相关，这样翻转码字中 $\mathcal{E}$ 指示的比特能够保持该 S-PC 的偶校验值不变。为了简单起见，本文定义若 k_1 和 k_2 两段与第 t 个 S-PC 同时相关或不相关，则这两段是等效的；相反地，若第 t 个 S-PC 中只与 k_1 和 k_2 其中一段相关，则这两段不等效。根据公式(4-14)，可以很容易地判断一个段是否与第 t 个 S-PC 相关。具体地，拥有如下索引的段与第 t 个 S-PC 相关：

$$\begin{aligned}\mathcal{I}_t=\{&2^{t-1}+1,2^{t-1}+2,\ldots,2^t,\ldots,(2j-1)2^{t-1}+1,(2j-1)2^{t-1}+2,\ldots,j2^t,\ldots,\\&2^d-2^{t-1}+1,2^d-2^{t-1}+2,\ldots,2^d\},\end{aligned} \tag{4-15}$$

而拥有如下索引的段与第 t 个 S-PC 不相关：

$$\begin{aligned}\mathcal{I}_t^c = \{&1,2,\ldots,2^{t-1},\ldots,(j-1)2^{t-1}+1,(j-1)2^{t-1}+2,\ldots,(2j-1)2^{t-1},\ldots,\\&2^d-2^t+1,2^d-2^t+2,\ldots,2^d-2^{t-1}\},\end{aligned} \tag{4-16}$$

其中 $j \in \{1,2,\ldots,2^{d-t}\}$。利用公式(4-15)和公式(4-16)可以确定影响 S-PC 的所有段，而如下定理可以进一步判断任意两段是否等效：

定理 4.3. 对于第 t 个 *S-PC*，$k+\mu 2^t$ 和 k 两段等效，其中 $\mu \in \{\lceil -k/2^t \rceil,\ldots,\lfloor (2^d-k)/2^t \rfloor\} \setminus \{0\}$。此外，$k+(2\nu-1)2^{t-1}$ 和 k 两段不等效，其中 $\nu \in \{\lceil -k/2^t+1/2 \rceil,\ldots,\lfloor (2^d-k)/2^t+1/2 \rfloor\}$.

证明. 详见附录C。 □

定理4.3意味着翻转坐标 $\mathcal{E}=\{k,k+\mu 2^t\}$ 指示的比特位置能够保持第 t 个 S-PC 的校验值，而翻转坐标 $\mathcal{E}=\{k,k+(2\nu-1)2^{t-1}\}$ 指示的比特位置会改变第 t 个 S-PC 的校验值。对于 $\gamma_{\text{S-PC}}[1:d]=(0,\ldots,0,1)$ 这样的特殊情况，可以通过以下引理来确定可行的翻转坐标：

引理 4.1. 对于 *SR1/SPC* 节点，$\gamma_{\textit{S-PC}}[1:d]=(0,\ldots,0,1)$ 情况下所有的可行翻转坐标为：

$$\mathcal{E}=\{k,k+2^{d-1}\}, \qquad k \in \{1,2,\ldots,2^{d-1}\}. \tag{4-17}$$

证明. 详见附录D。 □

引理4.1仅为 $\gamma_{\text{S-PC}}[1:d]=(0,\ldots,0,1)$ 的特殊情况提供了可行的翻转坐标。接下来，本文将通过归纳构造更一般情况下的可行翻转坐标。给定深度为 $(d-1)$ 的 SR1/SPC 节点，考虑深度为 d 的扩展 SR1/SPC 节点，其左子节点是深度 $(d-1)$ 的原始 SR1/SPC 节点，右子节点是 Rate-1 或 SPC 节点，该右子节点引入了一个新的 S-PC$\gamma_{\text{S-PC}}[d]$。基于深度为 $(d-1)$ 的原始翻转坐标，引理4.2构造了扩展 SR1/SPC 节点的可行翻转坐标：

引理 4.2. 对于深度为 $(d-1)$ 的 *SR1/SPC* 节点，给定 $\gamma_{\textit{S-PC}}[1:d-1]$ 情况下的可行翻转坐标 $\mathcal{E}=\{k_1,k_2\}$，并定义四个新翻转坐标 $\mathcal{E}_1=\{k_1,k_2\},\mathcal{E}_2=\{k_1+2^{d-1},k_2+2^{d-1}\},\mathcal{E}_3=\{k_1+2^{d-1},k_2\},\mathcal{E}_4=\{k_1,k_2+2^{d-1}\}$，根据 $\gamma_{\textit{S-PC}}[d]$ 的不同情况，扩展 *SR1/SPC* 节点中如下的翻转坐标是可行的：

$$\mathcal{E}_1,\mathcal{E}_2, \quad \text{若 } \gamma_{\textit{S-PC}}[d]=0 \tag{4-18}$$

$$\mathcal{E}_3, \mathcal{E}_4, \quad \text{若 } \gamma_{S\text{-}PC}[d] = 1 \tag{4-19}$$

$$\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3, \mathcal{E}_4, \quad \text{若 } \gamma_{S\text{-}PC}[d] = -1 \tag{4-20}$$

证明. 详见附录E。 □

引理4.2中的结果揭示了可行翻转坐标的构造过程，图4.4提供了 $d = 3$ 且 $\gamma_{\text{S-PC}}[1:d-1] = (0,1)$ 情况下 SR1/SPC 节点可行翻转坐标的构造过程。在图4.4中，每个方框表示一个段，带虚线的方框表示与 S-PC 相关的段。此外，用颜色标记的方框表示需要翻转的段，相连接的两个不同颜色的方框表示一个翻转坐标。对于深度为 $d-1$ 的 SR1/SPC 节点，且 $\gamma_{\text{S-PC}}[1:d-1] = (0,1)$，可行的翻转坐标可以根据引理4.1确定为 $\mathcal{E} = \{1,3\}$，此时所有 S-PC 都被校正了。对于深度为 d 的扩展 SR1/SPC 节点，原先的翻转坐标 $\mathcal{E}$ 通过分裂过程被构造为新的可行翻转坐标，这些新坐标是基于 $\gamma_{\text{S-PC}}[d]$ 的值确定的，分裂过程分别如图4.4 (a)、(b) 和 (c) 所示。若 $\gamma_{\text{S-PC}}[d] = 0$，则分裂过程遵循引理4.2中的公式(4-18)，生成新的可行坐标 $\mathcal{E}_1 = \{1,3\}$ 和 $\mathcal{E}_2 = \{5,7\}$；若 $\gamma_{\text{S-PC}}[d] = 1$，则 $\mathcal{E}_1 = \{1,3\}$ 根据公式(4-19)分裂为 $\mathcal{E}_3 = \{3,5\}$ 和 $\mathcal{E}_4 = \{1,7\}$；对于 $\gamma_{\text{S-PC}}[d] = -1$ 的情况，根据公式(4-20)，四个新的可行翻转坐标分别为 $\mathcal{E}_1 = \{1,3\}$、$\mathcal{E}_2 = \{5,7\}$、$\mathcal{E}_3 = \{3,5\}$ 和 $\mathcal{E}_4 = \{1,7\}$。

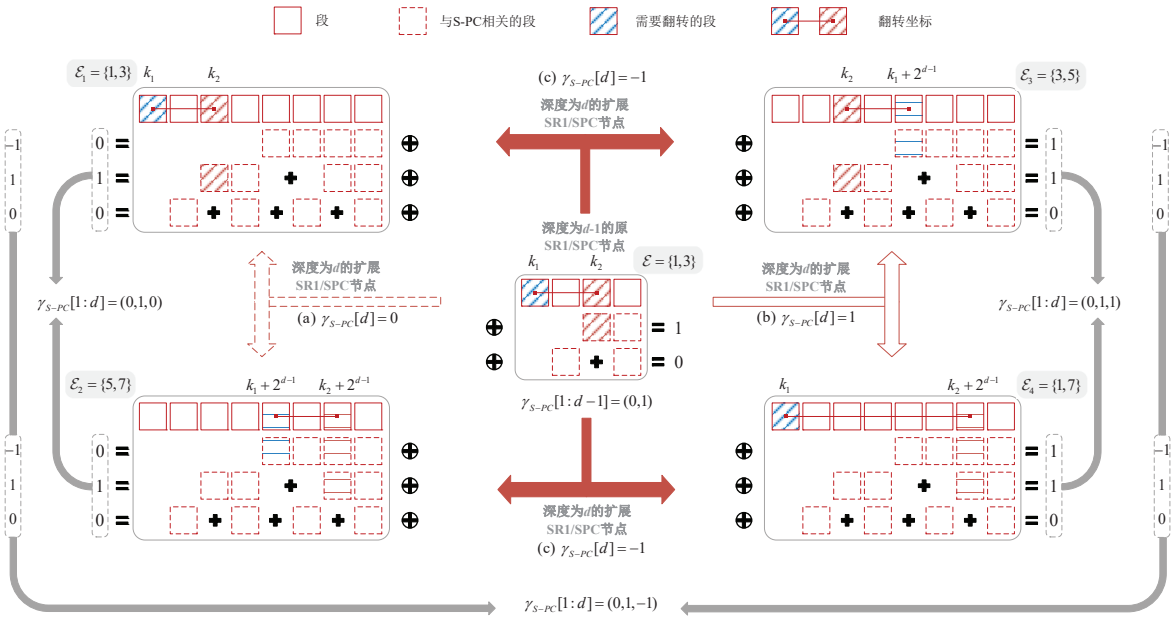


图 4.4　基于分裂过程构造可行的翻转坐标，当 (a) $\gamma_{\text{S-PC}}[d] = 0$，(b) $\gamma_{\text{S-PC}}[d] = 1$ 和 (c) $\gamma_{\text{S-PC}}[d] = -1$

基于引理4.1和4.2，算法4总结了构造可行翻转集合 $\mathcal{F}_{\gamma_{\text{S-PC}}}$ 的递归过程，其中 $\mathcal{F}_{\gamma_{\text{S-PC}}}$ 包含了 SR1/SPC 节点所有的可行翻转坐标。如算法4第 1 行所示，在 $\gamma_{\text{S-PC}}[1:d] =$

$(0,\ldots,0,1)$ 的边界条件下可以直接获得 $\mathcal{F}_{\gamma_{\text{S-PC}}}$。算法第 7-24 行根据引理4.2执行分裂过程，其中 $\mathcal{F}_{\gamma_{\text{S-PC}}}$ 是基于 $\gamma_{\text{S-PC}}[1:d-1]$ 和 $\mathcal{F}$ 构造的。

算法4提供了许多可行的翻转坐标用以校正 S-PC，并且在这些坐标指示的位置上执行比特翻转不会违反 P-PC。然而，应用这些可行的翻转坐标会获得许多不同的候选码字，译码算法需要选择一个最优的码字作为最终的输出结果。考虑到文献 [42] 提出的 ML 译码：

$$\beta_p^i[1:2^p] = \underset{\beta_p^i[1:2^p]\in\mathbb{B}_p^i}{\arg\max} \sum_{k=1}^{2^p} (-1)^{\beta_p^i[k]}\alpha_p^i[k], \tag{4-21}$$

其中 $\mathbb{B}_p^i$ 是节点 $\mathcal{N}_p^i$ 所有可能码字的集合，ML 译码的准则实际上是选取与硬判决结果欧式距离最短的码字作为译码结果。根据 ML 译码准则，本文引入了与翻转坐标 $\mathcal{E}=\{k_1,k_2,m\}$ 相关联的惩罚度量 $\lambda_{\mathcal{E}}$，该度量可以由下式计算：

$$\lambda_{\mathcal{E}} = (1-2\beta_{k_1}^{\text{S-PC}}[m])\alpha_{k_1}^{\text{S-PC}}[m] + (1-2\beta_{k_2}^{\text{S-PC}}[m])\alpha_{k_2}^{\text{S-PC}}[m]. \tag{4-22}$$

那么最佳翻转坐标可以由具有最小惩罚度量的翻转坐标确定，即

$$\mathcal{E}_{\text{opt}} = \underset{\mathcal{E}\in\mathcal{F}_{\gamma_{\text{S-PC}}}}{\arg\min}\, \lambda_{\mathcal{E}}. \tag{4-23}$$

最后，根据 $\mathcal{E}_{\text{opt}}$ 对从第一阶段（算法3）中获得的初步估计码字执行比特翻转操作，具体如下所示：

$$\beta_{k_1}^{\text{S-PC}}[m] = \beta_{k_1}^{\text{S-PC}}[m]\oplus 1, \qquad \beta_{k_2}^{\text{S-PC}}[m] = \beta_{k_2}^{\text{S-PC}}[m]\oplus 1. \tag{4-24}$$

4.5.4 整体的译码算法

基于算法3和算法4，算法5总结了 SR1/SPC 节点的整体译码过程。在算法5中，首先使用算法3（第 1 行）对 SR1/SPC 节点进行第一阶段的译码，以获得初步的估计码字 $\beta_p^i[1:2^p]$，此时 SR1/SPC 节点的所有 P-PC 均得到校正。然后，计算所有 S-PC 的奇偶校验值 $\gamma_{\text{S-PC}}[1:d]$（第 2 行）。若所有 S-PC 均被满足，这表明当前码字 $\beta_p^i[1:2^p]$ 是最优的，则终止整个算法。否则，需要对 $\beta_p^i[1:2^p]$ 进行比特翻转操作，以纠正奇校验的 S-PC（第 3-10 行）。所有的可行翻转坐标都包含在可行翻转集合中，可行翻转集合由算法4（第 4 行）确定。值得说明的是，通过考虑 $\gamma_{\text{S-PC}}[1:d]$ 的所有可能值，可以在译码之前离线生成并存储所有的可行翻转集合，对于特定的 S-PC 奇偶校验值，相应的可行翻转集合可以在译码过程中直接读取。接下来，根据公式(4-22)计算可行翻转集合中每个

算法 4 构造可行翻转集合

输入: $\gamma_{\text{S-PC}}[1:p-q]$

输出: $\mathcal{F}_{\gamma_{\text{S-PC}}}$

if $\gamma_{\text{S-PC}}[1:d]=(0,\ldots,0,1)$ **then**
  for $k=\{1,2,\ldots,2^{d-1}\}$ **do**
    $\mathcal{E}=\{k,k+2^{d-1}\}$，$\mathcal{F}_{\gamma_{\text{S-PC}}}=\{\mathcal{F}_{\gamma_{\text{S-PC}}},\mathcal{E}\}$
  end for
  return $\mathcal{F}_{\gamma_{\text{S-PC}}}$
else
  执行算法 4，输入 $\gamma_{\text{S-PC}}[1:d-1]$，得到 $\mathcal{F}$
  if $\gamma_{\text{S-PC}}[d]=0$ **then**
    for $\mathcal{F}$ 中每个坐标 $\mathcal{E}$ **do**
      $\mathcal{E}_1=\{k_1,k_2\}$，$\mathcal{E}_2=\{k_1+2^{d-1},k_2+2^{d-1}\}$，$\mathcal{F}_{\gamma_{\text{S-PC}}}=\{\mathcal{F}_{\gamma_{\text{S-PC}}},\mathcal{E}_1,\mathcal{E}_2\}$
    end for
    return $\mathcal{F}_{\gamma_{\text{S-PC}}}$
  else if $\gamma_{\text{S-PC}}[d]=1$ **then**
    for $\mathcal{F}$ 中每个坐标 $\mathcal{E}$ **do**
      $\mathcal{E}_3=\{k_1+2^{d-1},k_2\}$，$\mathcal{E}_4=\{k_1,k_2+2^{d-1}\}$，$\mathcal{F}_{\gamma_{\text{S-PC}}}=\{\mathcal{F}_{\gamma_{\text{S-PC}}},\mathcal{E}_3,\mathcal{E}_4\}$
    end for
    return $\mathcal{F}_{\gamma_{\text{S-PC}}}$
  else
    for $\mathcal{F}$ 中每个坐标 $\mathcal{E}$ **do**
      $\mathcal{E}_1=\{k_1,k_2\}$，$\mathcal{E}_2=\{k_1+2^{d-1},k_2+2^{d-1}\}$，
      $\mathcal{E}_3=\{k_1+2^{d-1},k_2\}$，$\mathcal{E}_4=\{k_1,k_2+2^{d-1}\}$，$\mathcal{F}_{\gamma_{\text{S-PC}}}=\{\mathcal{F}_{\gamma_{\text{S-PC}}},\mathcal{E}_1,\mathcal{E}_2,\mathcal{E}_3,\mathcal{E}_4\}$
    end for
    return $\mathcal{F}_{\gamma_{\text{S-PC}}}$
  end if
end if

可行翻转坐标的惩罚度量（第 5-7 行），然后比较这些惩罚度量来选择最佳的翻转坐标（第 8 行）。最后，翻转初步估计码字中最佳翻转坐标指示的比特位置，获得的码字即为最后的译码结果（第 9 行）。综上所述，针对 SR1/SPC 节点的译码算法5可以应用于 SC 译码中，并与其他现有的快速 SC 译码算法兼容，从而进一步降低 SC 译码时延。

算法 5 SR1/SPC 节点译码算法

输入: $\alpha_p^i[1:2^p]$

输出: $\beta_p^i[1:2^p]$

1: 利用算法3得到 $\beta_p^i[1:2^p]$
2: 根据公式(4-14)计算 $\gamma_{\text{S-PC}}[1:d]$
3: **if** $\exists t \in \{1,2,\ldots,d\}, \gamma_{\text{S-PC}}[t]=1$ **then**
4: 利用算法4生成 $\mathcal{F}_{\gamma_{\text{S-PC}}}$
5: **for** $\mathcal{F}_{\gamma_{\text{S-PC}}}$ 中每一个坐标 $\mathcal{E}$ **do**
6: 根据公式(4-22)计算 $\lambda_{\mathcal{E}}$
7: **end for**
8: 根据公式(4-23)选取 $\mathcal{E}_{\text{opt}}$
9: 根据公式(4-24)对 $\beta_p^i[1:2^p]$ 进行比特翻转操作
10: **end if**
11: **return** $\beta_p^i[1:2^p]$

4.5.5 简化的译码算法

为了得到最佳的码字，算法5需要计算和比较可行翻转集合中每个翻转坐标的惩罚度量，这导致了较高的计算复杂度。对于翻转坐标 $\mathcal{E}=\{k_1,k_2,m\}$，如果 $\beta_{k_1}^{\text{S-PC}}[m] \neq \text{HD}(\alpha_{k_1}^{\text{S-PC}}[m])$ 或 $\beta_{k_2}^{\text{S-PC}}[m] \neq \text{HD}(\alpha_{k_2}^{\text{S-PC}}[m])$，则 $\mathcal{E}$ 被称为污染的翻转坐标，这表明在第一阶段中该坐标指示的一个比特位置已经被翻转过。否则，如果 $\beta_{k_1}^{\text{S-PC}}[m]=\text{HD}(\alpha_{k_1}^{\text{S-PC}}[m])$ 并且 $\beta_{k_2}^{\text{S-PC}}[m]=\text{HD}(\alpha_{k_2}^{\text{S-PC}}[m])$，$\mathcal{E}$ 被称为无辜的翻转坐标，这表明在第一阶段中该坐标指示的两个比特位置均未被翻转过。给定一个污染的翻转坐标 $\mathcal{E}_1$ 和一个无辜的翻转坐标 $\mathcal{E}_2$，它们的惩罚度量可以分别表示为 $\lambda_{\mathcal{E}_1}=|\alpha_1|-|\alpha_2|$ 和 $\lambda_{\mathcal{E}_2}=|\alpha_3|+|\alpha_4|$，其中 α_i 表示服从高斯分布的根节点 LLR。直观来说，不等关系 $\lambda_{\mathcal{E}_1}<\lambda_{\mathcal{E}_2}$ 很大概率是成立的，这表明与污染的翻转坐标相比无辜的翻转坐标具有更大的惩罚度量，因此污染的翻转坐标

更有可能成为最佳翻转坐标。

基于这一观察，本节提出了一种简化的译码算法以避免在算法5第 5 行步骤中对整个可行翻转集合的穷举搜索。首先，在译码第一阶段中，最多有 2^q 个比特被翻转，这些翻转比特的位置由公式(4-11)确定。然后，对于 $m = \{1, 2, \ldots, 2^q\}$，可以将与 $\rho[m]$ 具有相同的段索引的可行翻转坐标标记为可能的污染翻转坐标。那么在译码的第二阶段中，简化的译码算法只需要计算和比较所有可能的污染翻转坐标的惩罚度量。与算法5中的普通译码相比，这种简化的译码算法能够减少一些冗余计算，因此更利于硬件实现。两种算法的计算复杂度参见本章第 6 节中的表4.4，另外值得注意的是，这种简化技术不会降低译码性能，具体参见本章第 6 节中的图4.6。

4.6 数值分析与性能仿真

本节通过 MATLAB 仿真比较了不同特殊节点的分布特性，以及不同 SC 译码算法的平均时延、计算复杂度和纠错性能。本节考虑码长为 $N \in \{512, 1024, 4096\}$ 的极化码。当 $N \leq 1024$ 时，采用 5G 标准中的极化序列[2]构造极化码，而对于 $N = 4096$，则采用 GA 法[38]进行极化码构造。

4.6.1 节点分布

表4.2对比了具有不同码长 N 和码率 R 的特定极化码中包含的 EG-PC 和 SR1/SPC 节点的数量。可以看出，通常 SR1/SPC 节点的数量随着码长的增加而增加，这表明对于较长的极化码译码时延节省的空间越大。当 $R > 1/2$ 时，可以观察到 SR1/SPC 节点的总数较少，但节点深度随着码长的增加而增大。与 EG-PC 节点相比，SR1/SPC 节点显然在极化码中分布地更加频繁，这表明利用 SR1/SPC 节点可以进一步降低 SC 译码时延。值得说明的是，本文所提的 SR1/SPC 是一种包含目前许多特殊节点类型的通用节点，因此在实际中不需要为表4.1中所示的每个特殊节点类型实现专用的译码器，这将会大大降低快速 SC 译码器实际硬件部署的复杂度。例如，码长为 $N = 128$ 的 5G 极化码可以仅由 SR0/REP 和 SR1/SPC 两类节点组合表示。

4.6.2 译码时延和复杂度分析

本节评估了译码算法的理论时延，该时延通过统计时间步长获得，统计过程中采用了与文献 [26,28,30] 中相同的全并行假设条件：

- 执行实数的加减法、比较、排序操作均会花费一个时间步；

表 4.2　EG-PC 和 SR1/SPC 的节点数量

N	R	SR1 节点深度 d						SSPC 节点深度 d						SR1/SPC（其他情况）节点深度 d						Total	EG-PC 节点深度 d						总数
		1	2	3	4	5	≥ 6	1	2	3	4	5	≥ 6	1	2	3	4	5	≥ 6		1	2	3	4	5	≥ 6	
512[1]	1/6	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	3	0	1	0	1	0	0	2
	1/3	0	1	1	0	0	0	0	1	1	0	0	0	0	0	1	1	0	0	6	0	1	1	0	0	0	2
	1/2	0	1	1	0	0	0	1	1	0	0	0	0	0	1	1	1	0	0	7	0	1	1	0	0	0	2
	2/3	0	0	1	0	1	0	0	2	1	0	0	0	0	0	0	2	0	0	7	0	0	1	0	1	0	2
	5/6	0	0	0	1	0	0	0	1	1	0	0	1	0	1	0	0	0	0	5	0	0	0	1	0	0	1
4096[2]	1/6	0	2	1	0	0	1	1	3	4	2	0	0	0	0	0	1	0	0	15	0	2	1	0	0	1	4
	1/3	0	2	2	1	0	0	1	8	6	3	1	0	0	0	2	0	1	1	28	0	2	2	1	0	0	5
	1/2	0	1	3	1	1	0	0	7	8	2	0	2	0	1	2	1	1	0	30	0	1	3	1	1	0	5
	2/3	0	5	4	0	0	1	4	7	4	4	0	0	0	0	1	1	2	1	34	0	5	4	0	0	1	9
	5/6	0	4	2	2	0	1	1	6	1	0	0	0	0	1	1	1	2	2	24	0	4	2	2	0	1	8

[1] 根据 5G 标准[2]构造的极化码
[2] 根据 GA 法[38]构造的极化码

- 比特操作（组合逻辑操作）不会造成时延；
- Wagner 译码会花费一个时间步。

算法5中第一和第二阶段所需的时间步长分别用 T_1 和 T_2 表示。首先，根据算法3，第一阶段的译码时延 T_1 取决于 SR1/SPC 的源节点结构。当源节点为 Rate-0 节点时，其码字可以直接获得；否则，计算源节点 LLR 和源节点译码所需的时间步长分别为 1 和 T_{SN}。此外，Wagner 译码需要额外的一个时间步。综上所述，T_1 可以表示为：

$$T_1 = \begin{cases} 1, & \text{若源节点是 Rate-0 节点} \\ T_{\text{SN}} + 2, & \text{否则} \end{cases}. \tag{4-25}$$

对于 T_2，在算法5的第 2 行中，计算 S-PC 的奇偶校验值只涉及异或的比特操作，因此该步骤不会引入时延。由于可行翻转集合可以离线生成，因此第 4 行中的步骤同样不会产生时延。第 5-7 行中的惩罚度量的计算仅涉及加法操作，并且不同翻转坐标的惩罚度量可以并行计算，因此该步骤仅需一个时间步。第 8 行中确定最佳翻转坐标基于比较操作，需要花费一个时间步。根据公式(4-24)，第 9 行中执行比特翻转操作只涉及 XOR 位操作而不需要时间步。综上所述，T_2 可以表示为：

$$T_2 = \begin{cases} 2, & \text{若 } \exists t \in \{1, 2, \ldots, d\}, \gamma_{\text{S-PC}}[t] = 1 \\ 0, & \text{否则} \end{cases}. \tag{4-26}$$

可以看出，T_2 是一个取决于 $\gamma_{\text{S-PC}}[1:d]$ 的变量，这说明 SR1/SPC 节点的译码时延实际取决于信道条件。最后，译码 SR1、SSPC 和 SR1/SPC 节点所需的总时间步长分别

由下式给出：

$$T_{\text{SR1}} = T_1, \qquad T_{\text{SSPC}} = T_{\text{SR1/SPC}} = T_1 + T_2. \tag{4-27}$$

此外，$T_{\text{SR1/SPC}}$ 的下限 T_{LB} 和上限 T_{UB} 分别由下式确定：

$$T_{\text{LB}} = T_1 + \min(T_2) = T_1, \qquad T_{\text{UB}} = T_1 + \max(T_2) = T_1 + 2. \tag{4-28}$$

对于给定的极化码并应用本章提出的SR1/SPC节点的快速译码算法,可以基于公式(4-27)和公式(4-28)计算整个译码过程需要的总时延、最低和最高时延。

对于两种码长 $N \in \{512, 4096\}$ 和五种码率 $R \in \{1/6, 1/3, 1/2, 2/3, 5/6\}$ 的不同极化码，表4.3比较了文献 [27-30] 中各个最先进的快速 SC 译码算法以及本章所提的快速 SC 译码算法所需的时间步长。本节选择了文献 [27] 中提出的 Fast-SSC（简称为 FSSC）译码算法作为基准，该算法考虑了 Rate-0、Rate-1、REP 和 SPC 节点。进一步地，将 Type I-V[28]、G-REP 和 G-PC[29]以及 SR0/REP[30]的节点译码逐个应用于 FSSC 中，可以得到三种混合的快速 SC 译码器，即 HFSC1、HFSC2 和 HFSC3。除此之外，将 Rate-0、Rate-1、REP、SPC 和 SR0/REP 节点的译码与算法5中所提出的 SR1/SPC 节点的译码相结合，得到新的基于序列节点的快速 SC 译码器（简称为 SN-FSC），其所需的最低和最高时间步长分别用“LB”和“UB”表示。为了取得更低的译码时延，本文根据文献 [29] 中的方法忽略了 SR1/SPC 节点中的 S-PC，使得算法5跳过第二阶段，使用这样的方法可以得到一种松弛的 SN-FSC 译码器，即 SN-RFSC。如表4.3所示，对于所有考虑的 N 和 R，所提出的 SN-FSC 译码器相对于其他译码器具有更低的译码时延，且译码 SR1/SPC 节点所带来的加速增益随着 R 的提高而增加。特别地，当 $N = 512, R = 5/6$ 且 $E_b/N_0 = 4.0$ dB 时，SN-FSC 译码器相比 FSSC 和 HFSC2 译码器可以分别节省 62.9% 和 43.8% 的译码时延。然而，当码长较短且码率较低时，SR1/SPC 节点的数量较少而使得加速增益有限。此外，当 E_b/N_0 更高时，SR1/SPC 节点的译码具有更低的时延，而当 $E_b/N_0 = 4.0$ dB 时，译码时延能够逼近下限，这意味着 SR1/SPC 节点的 S-PC 在良好的信道条件下会自动满足而不需要校正。另外，SN-RFSC 译码器可以在任意信道条件下达到 SN-FSC 译码器的译码时延下限。然而，从图4.5和图4.6可以看出，使用这种松弛的译码器也会导致严重的性能退化，因此该译码器并不适合在实际中应用。

上述针对算法5的时延分析基于资源不限的全并行假设条件，但在另一种极端情况下，即在各种操作完全串行执行的情况下，译码算法的复杂度可以由算术操作的数量统

表 4.3　不同 SC 译码算法的时延比较

N	R	FSSC[27]	HFSC1[1]	HFSC2[2]	HFSC3[3]	SN-FSC[4] E_b/N_0 (dB) 0.0	1.0	2.0	3.0	4.0	LB	UB	SN-RFSC[5]
512	1/6	89	63	55	36	34.86	34.29	34.06	34.01	34.00	34	36	34
	1/3	128	85	86	63	47.94	44.87	43.39	43.05	43.00	43	51	43
	1/2	126	89	101	70	59.27	57.32	54.63	54.09	54.01	54	64	54
	2/3	129	87	103	74	58.12	56.87	53.54	50.50	50.03	50	60	50
	5/6	88	64	72	58	38.40	37.88	37.84	35.14	32.61	32	40	32
4096	1/6	446	347	339	234	197.75	185.37	184.03	184.00	184.00	184	206	184
	1/3	639	485	525	368	289.28	262.85	255.07	255.00	255.00	255	301	255
	1/2	674	508	547	402	309.26	301.29	271.53	271.01	308.00	271	319	271
	2/3	590	434	474	354	273.04	273.24	248.48	237.15	237.01	237	285	237
	5/6	409	282	321	244	185.28	184.96	184.68	173.33	163.23	163	193	163

[1] FSSC + Type I-V[28]
[2] FSSC + G-REP[29] + G-PC[29]
[3] FSSC + EG-PC[30] + SR0/REP[30]
[4] FSSC + SR0/REP[30] + SR1/SPC.
[5] FSSC + SR0/REP[30] + RG-PC[29]

计，其中算术操作包括加法、比较和排序（Add-Compare-Sort, ACS）操作。表4.4比较了极化码 $\mathcal{P}(1024, 512)$ 译码所需的 ACS 操作数。对于本文所提的 SN-FSC 译码器，表4.4中上下两行的数字分别表示普通和简化译码器所需的 ACS 操作数。由表可知，HFSC3 相比其他基准算法需要的 ACS 操作数更多，这意味着 SR0/REP 节点的译码主要增加了加法运算，这是因为译码需要针对不同重复序列分别计算源节点的 LLR（公式(4-3)），文献[47] 中实际硬件部署结果显示采用 SR0/REP 节点确实会增加计算资源开销。与 SR0/REP 节点类似，SR1/SPC 节点的译码引入了更多排序操作，导致了总 ACS 操作数的增加。然而，与其他快速 SC 译码器相比，所提出的普通 SN-FSC 译码器所需的 ACS 操作数仅略微增加，并且仍然显著低于原始 SC 译码器。此外，利用所提出的简化技术，SN-FSC 译码器可以获得比 HFSC3 译码器更低的计算复杂度。综上所述，SR0/REP 和 SR1/SPC 这两种序列节点的快速 SC 译码实际是以略微增加的硬件资源消耗为代价大大降低了译码时延。

表 4.4　不同 SC 译码算法的计算复杂度比较

ACS 操作	SC	FSSC	HFSC1	HFSC2	HFSC3	SN-FSC E_b/N_0 (dB) 0.0	1.0	2.0	3.0	4.0	LB	UB
加法	5120	3110	3056	3110	5379	5244.36 4999.12	5144.23 4976.62	4958.23 4936.26	4930.72 4928.95	4928.19 4928.09	4928 4928	5440 5056
比较	5120	2742	2612	2604	2220	2216 2216	2216 2216	2216 2216	2216 2216	2216 2216	2216 2216	2216 2216
排序[1]	0	364	436	384	480	970.37 725.12	870.23 702.62	684.23 662.26	656.72 654.95	654.19 654.09	654 654	1166 782
总数	10240	6216	6104	6098	8079	8430.71 7940.24	8230.45 7895.24	7858.46 7814.52	7803.44 7799.91	7798.37 7798.19	7798 7798	8822 8054

[1] 此度量是统计参与排序的数据数量的总数得到。

4.6.3 译码性能比较

为了避免高昂的计算成本，本节采用列表大小为 32 的 SCL 译码器近似 ML 译码性能。图4.5比较了不同译码算法在译码一个极化码子码时的 FER 性能，其中该极化子码源自 GA 法构造的码长为 $N = 4096$ 极化码，且可以表示为一个 $p = 6$、$q = 2$ 和 $\mathbb{R} = \{2, 3, 4, 5\}$ 的 SSPC 节点。如图4.5所示，虽然所提出的译码算法的性能不如列表大小为 32 的 SCL 译码性能，但仍然优于传统的 SC 译码，这意味着 SR1/SPC 节点的译码算法可以取得接近 ML 的性能。然而，采用文献 [29] 中忽略 S-PC 的方法会导致译码性能出现大于 1 dB 的严重退化，这表明特殊节点包含的所有奇偶约束必须被满足才能保证译码性能。

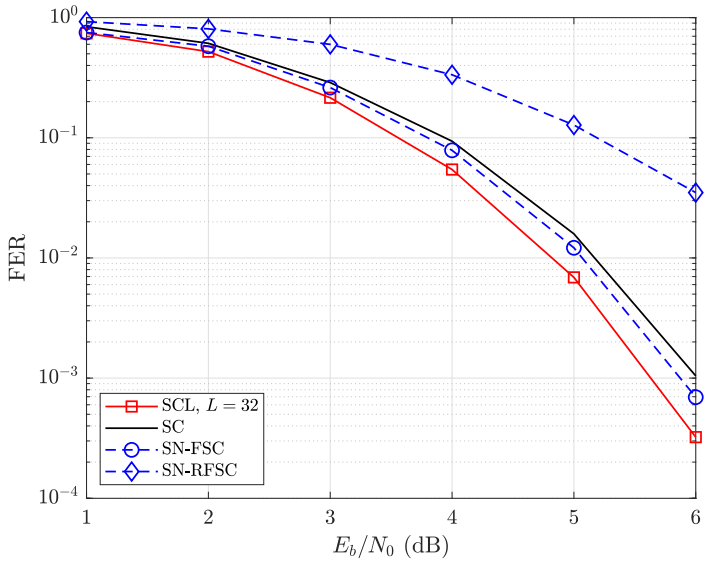


图 4.5 对于 SSPC 极化码子码，不同快速 SC 译码算法的译码性能比较

对于 $N = 1024$ 和 $R \in \{1/3, 1/2, 2/3\}$ 的 5G 极化码，图4.6比较了不同译码算法的 FER 性能。与图4.5中的现象类似，从图4.6可以观察到 SN-FSC 译码器性能超越了传统 SC 译码器。值得注意的是，这是所提译码算法的一个重要优点，因为先前文献中的快速 SC 译码器的性能均不能超越传统 SC 译码器。此外，图4.6还表明采用本章第 5 节提出的简化技术并不会导致性能退化。

4.7 本章小结

本章面向低时延通信传输场景，提出了一类新的特殊节点类型以及其快速 SC 译码算法。新提出的节点具有很高的并行度，且常常出现在高码率的极化码中。为了保证译

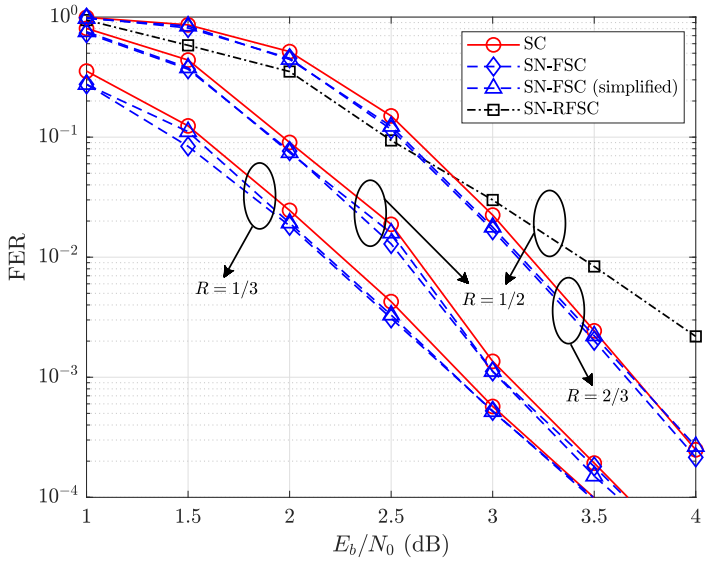


图 4.6　对于 5G 极化码，不同快速 SC 译码算法的译码性能比较

码码字合法，本章接下来分析了特殊节点中冻结比特在根节点处引起的对偶约束，为提出性能无损的译码算法奠定了基础。最后本章为所提的特殊节点设计了高效的译码算法，使得所提特殊节点能够获得接近 ML 的译码性能，同时译码所需要时延很低。仿真结果表明应用本章提出的特殊节点译码可以大大提升 SC 的译码速度。

第 5 章 总结与展望

本文以面向高性能低时延的极化码译码技术为主要研究对象，主要研究了 SCL 翻转译码算法和快速 SC 译码算法，主要成果如下：

（1）针对高性能译码，提出了动态 SCL 翻转译码算法，能够纠正 SCL 译码过程中的多个错误。首先，本文基于 SCL 译码错误发生的概率提出了一种评估 SCL 译码错误概率的翻转度量。随后，本文提出了一种新的 D-SCL-Flip 译码器，该译码器能够根据翻转度量选取具有最高概率纠正译码错误的翻转元进行重新译码尝试。仿真结果表明，与现有技术相比，所提出的译码器可以在平均重新解码尝试次数更低的情况下取得更好的 FER 性能。

（2）基于实际应用的考虑，本文对所提的 D-SCL-Flip 译码算法作出进一步优化。首先，现有的翻转度量均包含大量的超越函数的计算，不利于硬件实现，因此本文对原先的翻转度量作了计算方面的简化，简化后的翻转度量不包括复杂的指数或对数计算，大大降低了计算复杂度。然而，翻转度量的化简会导致一定的译码性能退化，为了解决这一问题，本文向简化的翻转度量中引入扰动参数，并采用 DL 技术优化这些参数，从而弥补了译码性能损失，同时 DL 辅助的翻转度量同样具有很低的计算复杂度。最后，本文改进了原先逐比特的译码方式，提出了节点级别的比特翻转策略和翻转度量，使得译码过程按照节点方式进行，大大降低了整个译码算法的时延。

（3）针对低时延译码，本文引入了一种新的 SR1/SPC 特殊节点类型，并提出了相应的快速 SC 译码算法。所提出的 SR1/SPC 节点可以提供对各种现有特殊节点类型的统一描述。本文推导了节点译码所需要满足的对偶约束，并进一步提出了高效的译码算法。仿真结果表明，利用 SR1/SPC 节点可以实现更低的 SC 译码时延，同时译码性能略有提升。

在本文基础上，可以扩展的研究内容有：

（1）本文研究的主要对象是 SC 和 SCL 译码，然而这些译码算法不能够输出软信息，因此通信接收机的其他模块无法充分利用译码过程中的信息。可以考虑研究其他 SISO 的译码方法，也可以考虑其他模块与译码器的联合优化问题。

（2）本文主要从译码角度提高极化码的纠错性能，未来的研究工作可以基于编码方

向。现有的编码方案大多是以优化 ML 性能为目标，而实际使用的译码器很难达到性能下界，可以考虑利用智能化算法研究适用于实际译码器的极化码构造方案，使得极化码在有限码长和特定译码方案下的纠错性能进一步提升。

（3）在接下来的研究中，可以对本文提出的快速 SC 译码算法做出硬件实现，以准确评估译码吞吐量方面的增益。

（4）所提出了 SR1/SPC 节点可以拓展到其他基于 SC 的译码算法中，如 SC-Flip、SCAN 和 SCL 译码算法。本文推导的对偶约束仅取决于 SR1/SPC 节点结构，而与所采用的译码算法无关。因此，当 SR1/SPC 节点用于其他基于 SC 的译码算法中时，节点的估计码字也应当满足 P-PC 和 S-PC 这两类对偶约束，这样能够确保码字合法从而使得译码性能不会严重退化。如何利用 SR1/SPC 节点提高其他译码算法的并行度，是一个可待研究的方向。

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 (Release15)[Z]. Tech. Rep. TS 38.212V15.2.0 (2018-06), Jan. 2018. [Online]. Available: http://www.3gpp.org/ftp//Specs/archive/38_series/38.212/38212-f20.zip.

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

[4] NIU K, CHEN K. CRC-Aided Decoding of Polar Codes[J]. IEEE Communications Letters, 2012, 16(10): 1668-1671.

[5] GALLAGER R. Low-density parity-check codes[J]. IRE Transactions on Information Theory, 1962, 8(1): 21-28.

[6] BERROU C, GLAVIEUX A, THITIMAJSHIMA P. Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1[C]//In Proc. of IEEE International Conference on Communications (ICC): vol. 2. 1993: 1064-1070.

[7] BALATSOUKAS-STIMMING A, GIARD P, BURG A. Comparison of Polar Decoders with Existing Low-Density Parity-Check and Turbo Decoders[C]//In Proc. of IEEE Wireless Communication Network Conference Workshops (WCNCW). 2017: 1-6.

[8] ERCAN F, CONDO C, HASHEMI S A, et al. On error-correction performance and implementation of polar code list decoders for 5G[C]//In Proc. of IEEE Annual Allerton Conference on Communication, Control, and Computing (Allerton). 2017: 443-449.

[9] BALATSOUKAS-STIMMING A, PARIZI M B, BURG A. LLR-Based Successive Cancellation List Decoding of Polar Codes[J]. IEEE Transactions on Signal Processing, 2015, 63(19): 5165-5179.

[10] YUAN B, PARHI K K. Early Stopping Criteria for Energy-Efficient Low-Latency Belief-Propagation Polar Code Decoders[J]. IEEE Transactions on Signal Processing, 2014, 62(24): 6496-6506.

[11] GOELA N, KORADA S B, GASTPAR M. On LP decoding of polar codes[C]//In Proc. of IEEE Information Theory Workshop (ITW). 2010: 1-5.

[12] ELKELESH A, EBADA M, CAMMERER S, et al. Belief Propagation List Decoding of Polar Codes[J]. IEEE Communications Letters, 2018, 22(8): 1536-1539.

[13] FAYYAZ U U, BARRY J R. Low-Complexity Soft-Output Decoding of Polar Codes[J]. IEEE Journal on Selected Areas in Communications, 2014, 32(5): 958-966.

[14] AFISIADIS O, BALATSOUKAS-STIMMING A, BURG A. A low-complexity improved successive cancellation decoder for polar codes[C]//In Proc. of IEEE Asilomar Conference on Signals, Systems and Computers (Asilomar). 2014: 2116-2120.

[15] CHANDESRIS L, SAVIN V, DECLERCQ D. An Improved SCFlip Decoder for Polar Codes[C]//In Proc. of IEEE Global Communications Conference (GLOBECOM). 2016: 1-6.

[16] CHANDESRIS L, SAVIN V, DECLERCQ D. Dynamic-SCFlip Decoding of Polar Codes[J]. IEEE Transactions on Communications, 2018, 66(6): 2333-2345.

[17] ZHANG Z, QIN K, ZHANG L, et al. Progressive Bit-Flipping Decoding of Polar Codes: A Critical-Set Based Tree Search Approach[J]. IEEE Access, 2018, 6: 57738-57750.

[18] YU Y, PAN Z, LIU N, et al. Belief Propagation Bit-Flip Decoder for Polar Codes[J]. IEEE Access, 2019, 7: 10937-10946.

[19] CHENG F, LIU A, ZHANG Y, et al. Bit-Flip Algorithm for Successive Cancellation List Decoder of Polar Codes[J]. IEEE Access, 2019, 7: 58346-58352.

[20] PAN Y H, WANG C H, UENG Y L. Generalized SCL-Flip Decoding of Polar Codes[C]//In Proc. of IEEE Global Communications Conference (GLOBECOM). 2020: 1-6.

[21] XU W, WU Z, UENG Y L, et al. Improved polar decoder based on deep learning[C]//In Proc. of IEEE International Workshop on Signal Processing Systems (SiPS). 2017: 1-6.

[22] DOAN N, HASHEMI S A, ERCAN F, et al. Fast SC-Flip Decoding of Polar Codes with Reinforcement Learning[C]//In Proc. of IEEE International Conference on Communications (ICC). 2021: 1-6.

[23] CHEN C H, TENG C F, WU A Y. Low-complexity LSTM-assisted bit-flipping algorithm for successive cancellation list polar decoder[C]//In Proc. of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). 2020: 1708-1712.

[24] TENG C F, WU A Y A. Convolutional Neural Network-Aided Tree-Based Bit-Flipping Framework for Polar Decoder Using Imitation Learning[J]. IEEE Transactions on Signal Processing, 2021, 69: 300-313.

[25] LEROUX C, RAYMOND A J, SARKIS G, et al. A Semi-Parallel Successive-Cancellation Decoder for Polar Codes[J]. IEEE Transactions on Signal Processing, 2013, 61(2): 289-299.

[26] ALAMDAR-YAZDI A, KSCHISCHANG F R. A Simplified Successive-Cancellation Decoder for Polar Codes[J]. IEEE Communications Letters, 2011, 15(12): 1378-1380.

[27] 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.

[28] 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.

[29] CONDO C, BIOGLIO V, LAND I. Generalized Fast Decoding of Polar Codes[C]//In Proc. of IEEE Global Communications Conference (GLOBECOM). 2018: 1-6.

[30] ZHENG H, 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.

[31] SARKIS G, GIARD P, VARDY A, et al. Fast List Decoders for Polar Codes[J]. IEEE Journal on Selected Areas in Communications, 2016, 34(2): 318-328.

[32] HASHEMI S A, CONDO C, GROSS W J. A fast polar code list decoder architecture based on sphere decoding[J]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2016, 63(12): 2368-2380.

[33] 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.

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

[35] PILLET C, CONDO C, BIOGLIO V. Fast-SCAN decoding of Polar Codes[C]//In Proc. of IEEE International Symposium on Topics in Coding (ISTC). 2021: 1-5.

[36] SHEN Y, ZHOU W, HUANG Y, et al. Fast Iterative Soft-Output List Decoding of Polar Codes[J]. IEEE Transactions on Signal Processing, 2022, 70: 1361-1376.

[37] TAL I, VARDY A. How to Construct Polar Codes[J]. IEEE Transactions on Information Theory, 2013, 59(10): 6562-6582.

[38] TRIFONOV P. Efficient design and decoding of polar codes[J]. IEEE Transactions on Communications, 2012, 60(11): 3221-3227.

[39] HE G, BELFIORE J C, LAND I, et al. Beta-Expansion: A Theoretical Framework for Fast and Recursive Construction of Polar Codes[C]//In Proc. of IEEE Global Communications Conference (GLOBECOM). 2017: 1-6.

[40] YONGRUN Y, ZHIWEN P, NAN L, et al. Successive Cancellation List Bit-flip Decoder for Polar Codes[C]//In Proc. of IEEE International Wireless Communications and Signal Processing (WCSP). 2018: 1-6.

[41] ROWSHAN M, VITERBO E. Improved List Decoding of Polar Codes by Shifted-pruning[C]//In Proc. of IEEE Information Theory Workshop (ITW). 2019: 1-5.

[42] SARKIS G, GROSS W J. Increasing the Throughput of Polar Decoders[J]. IEEE Communications Letters, 2013, 17(4): 725-728.

[43] HAYKIN S. Neural networks and learning machines, 3/E[M]. Pearson Education India, 2010.

[44] ERCAN F, TONNELLIER T, CONDO C, et al. Operation Merging for Hardware Implementations of Fast Polar Decoders[J]. Journal of Signal Processing Systems for Signal, Image, and Video Technology, 2019, 91(9): 995-1007.

[45] DUFFY K R, LI J, MÉDARD M. Capacity-Achieving Guessing Random Additive Noise Decoding[J]. IEEE Transactions on Information Theory, 2019, 65(7): 4023-4040.

[46] HASHEMI S A, MONDELLI M, CIOFFI J, et al. Successive Syndrome-Check Decoding of Polar Codes[C]//In Proc. of IEEE Asilomar Conference on Signals, Systems and Computers (Asilomar). 2021: 943-947.

[47] ZHENG H, BALATSOUKAS-STIMMING A, CAO Z, et al. Implementation of a High-Throughput Fast-SSC Polar Decoder with Sequence Repetition Node[C]//In Proc. of IEEE Workshop on Signal Processing Systems (SiPS). 2020: 1-6.

附录

A 定理4.1的证明

给定在译码树 q 层中各个节点的码字，可以基于比特递归公式(2-18)推导出 p 层根节点的码字：

$$\beta_p^i[1:2^p] = \boldsymbol{\beta}_q(\boldsymbol{F}^{\otimes d} \otimes \boldsymbol{I}_{2^q}), \tag{A-1}$$

其中 $\boldsymbol{\beta}_q = (\beta_q^{L_q}[1:2^q], \beta_q^{L_q+1}[1:2^q], \ldots, \beta_q^{R_q}[1:2^q])$。接着，利用等式 $(\boldsymbol{A} \otimes \boldsymbol{B}) \otimes \boldsymbol{C} = \boldsymbol{A} \otimes (\boldsymbol{B} \otimes \boldsymbol{C})$，并使 $\boldsymbol{A} = \boldsymbol{B} = \boldsymbol{F}$ 和 $\boldsymbol{C} = \boldsymbol{I}_{2^q}$，公式(A-1)可以进一步写为：

$$\begin{aligned} \beta_p^i[1:2^p] &= \boldsymbol{\beta}_q(\boldsymbol{F}^{\otimes d-1} \otimes (\boldsymbol{F} \otimes \boldsymbol{I}_{2^q})) \\ &= \boldsymbol{\beta}_q\left(\boldsymbol{F}^{\otimes d-1} \otimes \begin{pmatrix} \boldsymbol{I}_{2^q} & \boldsymbol{0}_{2^q} \\ \boldsymbol{I}_{2^q} & \boldsymbol{I}_{2^q} \end{pmatrix}\right). \end{aligned} \tag{A-2}$$

重复 $d-1$ 次上述过程可以得到：

$$\beta_p^i[1:2^p] = \boldsymbol{\beta}_q \boldsymbol{G}'_{2^d}, \tag{A-3}$$

其中 $\boldsymbol{G}'_{2^d}$ 是通过分别用 $\boldsymbol{0}_{2^q}$ 和 $\boldsymbol{I}_{2^q}$ 替换 $\boldsymbol{G}_{2^d}$ 中的元素 0 和 1 得到的。然后对于任意 $k \in \{1, 2, \ldots, 2^q\}$，可以得到：

$$\begin{aligned} \bigoplus_{j=1}^{2^d} \beta_p^i[(j-1)2^q + k] &= \bigoplus_{j=1}^{2^d} (\boldsymbol{\beta}_q \boldsymbol{G}'_{2^d})_{(j-1)2^q+k} \\ &= \boldsymbol{\beta}_q \bigoplus_{j=1}^{2^d} (\boldsymbol{G}'_{2^d})_{(j-1)2^q+k} \\ &\overset{(a)}{=} \boldsymbol{\beta}_q((\boldsymbol{I}_{2^q}, \overbrace{\boldsymbol{0}_{2^q}, \ldots, \boldsymbol{0}_{2^q}}^{2^d-1})^T)_k \\ &= \boldsymbol{\beta}_q(\overbrace{0, \ldots, 0}^{k-1}, 1, \overbrace{0, \ldots, 0}^{2^p-k})^T \\ &= \beta_q[k] \overset{(b)}{=} \beta_q^{L_q}[k], \end{aligned} \tag{A-4}$$

其中 (a) 是由于生成矩阵的性质，即 $\bigoplus_{j=1}^{2^d} (\boldsymbol{G}_{2^d})_j = (1, 0, \ldots, 0)^T$，而 (b) 是根据 $\boldsymbol{\beta}_q$ 的定义得到的。定理4.1证毕。

B 定理4.2的证明

类似于定理4.1的证明，可以得到下面的等式：

$$\begin{aligned}\bigoplus_{j=1}^{2^{p-r-1}} \beta_p^i[(2j-1)2^r+k] &\overset{(a)}{=} \boldsymbol{\beta}_r \bigoplus_{j=1}^{2^{p-r-1}} (\boldsymbol{G}')_{(2j-1)2^r+k} \\ &\overset{(b)}{=} \boldsymbol{\beta}_r((\mathbf{0}_{2^r}, \boldsymbol{I}_{2^r}, \overbrace{\mathbf{0}_{2^r}, \ldots, \mathbf{0}_{2^r}}^{2^{p-r}-2})^T)_k \\ &= \boldsymbol{\beta}_r(\overbrace{0, \ldots, 0}^{2^r+k-1}, 1, \overbrace{0, \ldots, 0}^{2^p-2^r-k})^T \\ &= \beta_q[2^r+k] \\ &= \beta_r^{L_r+1}[k],\end{aligned} \tag{B-1}$$

其中 (a) 基于公式(A-3)，(b) 是由于 $\bigoplus_{j=1}^{2^{p-r-1}}(\boldsymbol{G}_{2^{p-r}})_{2j} = (0, 1, \ldots, 0)^T$。因为 $\mathcal{N}_r^{L_r+1}$ 是一个 SPC 节点，其码字应该满足[27]：

$$\bigoplus_{k=1}^{2^r} \beta_r^{L_r+1}[k] = 0. \tag{B-2}$$

使用公式(B-1)替换公式(B-2)中等式左边的 $\beta_r^{L_r+1}[k]$，定理4.2证毕。

C 定理4.3的证明

首先证明段索引 k 和 $k+\mu 2^t$ 的等效性，其中 $\mu = \{\lceil -k/2^t \rceil, \ldots, \lfloor (2^d-k)/2^t \rfloor\} \setminus \{0\}$。从 $\mu = 1$ 开始，若段 k 与第 t 个 S-PC 相关，即 $k \in \mathcal{I}_t$，$k+2^t$ 可以确定为：

$$\begin{aligned}k \in \mathcal{I}_t &\Rightarrow (2j-1)2^{t-1} + 1 + 2^t \le k + 2^t \le j2^t + 2^t \\ &\Rightarrow (2(j+1)-1)2^{t-1} + 1 \le k + 2^t \le (j+1)2^t \\ &\Rightarrow k + 2^t \in \mathcal{I}_t.\end{aligned}$$

将 $j+1$ 视为新的 j，从公式(4-15)中可以观察到段 $k+2^t$ 同样与第 t 个 S-PC 相关。类似地，若段 k 与第 t 个 S-PC 不相关，即 $k \in \mathcal{I}_t^c$，可以根据公式(4-16)确定 $k+2^t$ 的范围：

$$\begin{aligned}k \in \mathcal{I}_t^c &\Rightarrow (j-1)2^t + 1 + 2^t \le k + 2^t \le (2j-1)2^{t-1} + 2^t \\ &\Rightarrow j2^t + 1 \le k + 2^t \le (2(j+1)-1)2^{t-1} \\ &\Rightarrow k + 2^t \in \mathcal{I}_t^c,\end{aligned}$$

这说明段 $k+2^t$ 与第 t 个 S-PC 无关。因此对于 $j=1$，k 和 $k+2^t$ 这两段是等效的。以此类推，可以证明 $k+2^t$ 和 $k+2\times 2^t$ 两段等效，而 $k+2\times 2^t$ 又与 $k+3\times 2^t$ 等效。最终

可以证明 k 与 $k+\lceil -k/2^t \rceil 2^t, \ldots, k-2^t, k+2^t, \ldots, k+\lfloor (2^d-k)/2^t \rfloor 2^t$ 都是等效的，从而证明了 $k+\mu 2^t, \mu = \{\lceil -k/2^t \rceil, \ldots, \lfloor (2^d-k)/2^t \rfloor\}$ 相互等效。

接下来证明 k 和 $k+(2\nu-1)2^{t-1}$ 是不等效的，其中 $\nu \in \{\lceil -k/2^t+1/2 \rceil, \ldots, \lfloor (2^d-k)/2^t+1/2 \rfloor\}$。令 $\nu=1$，根据段 k 是否与第 t 个 S-PC 相关，可以利用公式(4-15)和公式(4-16)确定 $k+2^{t-1}$ 的范围如下：

$$\begin{aligned} k \in \mathcal{I}_t &\Rightarrow (2j-1)2^{t-1}+1+2^{t-1} \le k+2^{t-1} \le j2^t+2^{t-1} \\ &\Rightarrow j2^t+1 \le k+2^{t-1} \le (2j+1)2^{t-1} \\ &\Rightarrow k+2^t \in \mathcal{I}_t^c, \end{aligned}$$

$$\begin{aligned} k \in \mathcal{I}_t^c &\Rightarrow (j-1)2^t+1+2^{t-1} \le k+2^{t-1} \le (2j-1)2^{t-1}+2^{t-1} \\ &\Rightarrow (2j-1)2^{t-1}+1 \le k+2^{t-1} \le 2j2^{t-1} \\ &\Rightarrow k+2^t \in \mathcal{I}_t. \end{aligned}$$

可以看出 k 和 $k+2^{t-1}$ 两段中只有一个和第 t 个 S-PC 相关。因此对于第 t 个 S-PC，k 和 $k+2^{t-1}$ 这两段是不等效的，从而 $\nu=1$ 的情况证明完毕。接着，根据上述证明可以得到 $k+2^{t-1}$ 和 $k+2^{t-1}+\mu 2^t$ 是等效的，那么可以推出 k 和 $k+(2\nu-1)2^{t-1}$ 也是不等效的，其中 $\nu \in \{\lceil -k/2^t+1/2 \rceil, \ldots, \lfloor (2^d-k)/2^t+1/2 \rfloor\}$。定理4.3证毕。

D 引理4.1的证明

为了保持从 $t=1$ 到 $t=d-1$ 中偶数校验的 S-PC，一个可行的翻转坐标必须包含两个等效段的索引。此外，如果第 d 个 S-PC 的奇偶校验是奇数，则必须通过翻转两个不等效的段来纠正。根据公式(4-15)和公式(4-15)，与 $t=1$ 个 SPC 相关和不相关的段索引可以分别确定为 $\mathcal{I}_1 = \{2,4,6,\ldots,2^d\}$ 和 $\mathcal{I}_1^c = \{1,3,5,\ldots,2^d-1\}$。可以推断对于任何可行的翻转坐标，两个段索引之间的间隔必须是偶数，因此可行的间隔可以由集合 $\{2,4,6,\ldots,2^d\}$ 确定。然后，对于 $t=2$ 的情况，基于定理4.3，所有值为 $(2\nu-1)2^{t-1}$ 的间隔都是不可行的，这将使得可行的间隔减少为 $\{4,8,12,\ldots,2^d\}$。同时，$\{4,8,12,\ldots,2^d\}$ 中所有间隔是有效的，因为根据定理4.3，间隔为 $\mu 2^t$ 的两个段是等效的。接着，对于 $t=3$ 的情况，通过再次引用定理4.3，可行间隔的集合变为 $\{8,16,24,\ldots,2^d\}$。重复上述过程多次，直到 $t=d-1$，可以得到唯一可行的间隔是 2^{d-1}。最后，对于 $t=d$ 的情况，根据定理4.3，可以容易地证明间隔为 2^{d-1} 的两段是不等效的。因此，对于 $t=1$ 到

$t=d-1$，间隔为 2^{d-1} 的两段是等效的，而对于 $t=d$，间隔为 2^{d-1} 的两段是不等效的，这样就完成了引理4.1的证明。

E 引理4.2的证明

首先，对于 $\gamma_{\text{S-PC}}[d]=0$ 的情况，可行翻转坐标中的两个段应该等效从而保持第 d 个 S-PC 为偶校验，同时保持 $t=1$ 到 $t=d-1$ 这些 S-PC 的奇偶校验值不变。从公式(4-15)和公式(4-15)中可以观察到，第 d 个 S-PC 与 k_1 和 k_2 两段不相关，而与 k_1+2^{d-1} 和 k_2+2^{d-1} 两段都相关，因此 $\mathcal{E}_1=\{k_1,k_2\}$ 和 $\mathcal{E}_2=\{k_1+2^{d-1},k_2+2^{d-1}\}$ 可以保持第 d 个 S-PC 的奇偶校验。此外，根据定理4.3，k 和 $k+2^{d-1}$ 这两段对于 $t=1$ 到 $t=d-1$ 这些 S-PC 是等效的，这意味着 $\mathcal{E}_2=\{k_1+2^{d-1},k_2+2^{d-1}\}$ 同样不会改变这些 S-PC 的奇偶校验值。因此当 $\gamma_{\text{S-PC}}[d]=0$ 时，$\mathcal{E}_1$ 和 $\mathcal{E}_2$ 是可行的翻转坐标。

接下来，如果 $\gamma_{\text{S-PC}}[d]=1$，可行翻转坐标中的两个段应该不等效从而纠正第 d 个 S-PC 为偶校验，同时保持 $t=1$ 到 $t=d-1$ 这些 S-PC 的奇偶校验值不变。从公式(4-15)和公式(4-15)中可以观察到，第 d 个 S-PC 与 k_1 和 k_2 两段不相关，而与 k_1+2^{d-1} 和 k_2+2^{d-1} 两段都相关，因此 $\mathcal{E}_3=\{k_1+2^{d-1},k_2\}$ 和 $\mathcal{E}_4=\{k_1,k_2+2^{d-1}\}$ 可以纠正第 d 个 S-PC 的奇偶校验。此外，根据定理4.3，k 和 $k+2^{d-1}$ 这两段对于 $t=1$ 到 $t=d-1$ 这些 S-PC 是等效的，这意味着 $\mathcal{E}_3=\{k_1+2^{d-1},k_2\}$ 和 $\mathcal{E}_4=\{k_1,k_2+2^{d-1}\}$ 不会改变这些 S-PC 的奇偶校验值。因此当 $\gamma_{\text{S-PC}}[d]=0$ 时，$\mathcal{E}_3$ 和 $\mathcal{E}_4$ 是可行的翻转坐标。

最后，由于在 $\gamma_{\text{S-PC}}[d]=-1$ 的情况下，扩展的 Rate-1 子节点没有引入 S-PC，因此对应 $\gamma_{\text{S-PC}}[1:d]$ 的可行翻转坐标只需要保持 $t=1$ 到 $t=d-1$ 这些 S-PC 奇偶校验不变。如上所述，k 和 $k+2^{d-1}$ 这两段对于 $t=1$ 到 $t=d-1$ 这些 S-PC 是等价的，因此将此等效关系应用于 $\mathcal{E}=\{k_1,k_2\}$ 中的一个或两个元素会产生四个新的可行翻转坐标 $\mathcal{E}_1$、$\mathcal{E}_/2$、$\mathcal{E}_3$ 和 $\mathcal{E}_4$，因此引理4.2证毕。

攻读硕士学位期间主要的研究成果

参研项目

1. 《xxx 网络系统设计项目》：极化码编译码算法研究，以及基于 FPGA 平台的硬件实现。

发表及再审的学术论文

A. 期刊论文

1. Y. Lu, M. M. Zhao, M. Lei, et al, Deep Learning Aided SCL Decoding of Polar Codes with Shifted-Pruning[J], China Communications, 2023, 20(1): 153-170.（SCI 检索，第一作者）
2. Y. Lu, M. M. Zhao, M. Lei, et al, Fast Successive-Cancellation Decoding of Polar Codes with Sequence Nodes[J], IEEE Transactions on Communications (TCOM), 2022: under review. （SCI 检索，第一作者，在审）

B. 会议论文

1. Y. Lu, M. M. Zhao, M. Lei, et al, Fast Decoding of Sequence Rate-1 or SPC Nodes for Polar Codes[C], in Proc. of IEEE International Conference on Communications (ICC), 2023: accepted. （EI 检索，第一作者）

受理的国家发明专利

1. 陆旸，赵明敏，雷鸣，赵民建，极化码特殊节点的快速连续抵消译码方法及装置，专利申请号：202211399373.1。（第一发明人，已受理）

在校获得荣誉

1. 2021-2022 年度，获得浙江大学优秀研究生、三好研究生荣誉称号