國 立 交 通 大 學

電信工程研究所

碩 士 論 文

結合錯誤更正與通道估計的籬笆型編碼之設計

Trellis Code Design for Combined Channel Estimation and Error Correction

研究生：張惠雅

指導教授：陳伯寧 博士

中 華 民 國 九十九 年 八 月

結合錯誤更正與通道估計的籬笆型編碼之設計

Trellis Code Design for Combined Channel Estimation and Error Correction

研究生：張惠雅　　Student : Huei-Ya Chang

指導教授：陳伯寧 博士　　Advisor : Dr. Po-Ning Chen

國 立 交 通 大 學

電信工程研究所

碩 士 論 文

A Thesis

Submitted to Institute of Communications Engineering

College of Electrical and Computer Engineering

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master

of Science

in

Communications Engineering

August 2010

Hsinchu, Taiwan, Republic of China

中 華 民 國 九 十 九 年 八 月

結合錯誤更正與通道估計的籬笆型編碼之設計

學生：張惠雅　　指導教授：陳伯寧 博士

國立交通大學

電信工程研究所碩士班

摘要

針對結合錯誤更正與通道估計的樹狀結構碼，本篇論文提供一演算法將其壓成離笆型編碼以降低解碼複雜度。此演算法中，使用者可以透過設定離笆每階層節點數上限來控制壓縮程度。模擬顯示樹狀結構碼被壓縮越多，解碼時需越少的記憶體和越低的複雜度，但效能亦會越差。而使用者可以藉由適當地設定節點數上限而從中做取捨。

Trellis Code Design for Combined Channel Estimation and Error Correction

Student : Huei-Ya Chang Advisor : Dr. Po-Ning Chen

Institute of Communications Engineering

National Chiao Tung University

Abstract

In this thesis, we attempt to decrease the decoding complexity of self-orthogonal tree-structure codes designed for combined channel estimation and error correction by fitting them into a trellis structure subject to an upper bound on the number of trellis states at each level. Specifically, we provide an algorithm to compress the tree-structure codes into a trellis structure by selectively merging tree nodes. Simulations show that the more the merged nodes, the less the memory consumption and decoding complexity, but the worse the performance. A trade-off among these factors can therefore be controlled by a proper choice of maximum number of trellis states.

Acknowledgements

I owe my gratitude to all those who have made this thesis possible, because of whom my graduate experience has been one that I will cherish forever, especially to my advisor, Po-Ning Chen, and my grandfather, Jun-Jie Yan.

Contents

Chinese Abstract **i**

Abstract **ii**

Acknowledgements **iii**

Contents **iv**

List of Figures **vi**

1 Introduction **1**

1.1 Overview 1

1.2 Acronyms and Notations 3

2 Preliminaries **4**

2.1 Joint Maximum-Likelihood Estimation of Data Sequence and Channel Effect 4

2.2 The Code Tree Structure 6

2.3 The Systematic Code Design for Combined Channel Estimation and Error Correction 8

2.4 The Priority-First Search Algorithm 10

2.5 Maximum-Likelihood Decoding Metric for Priority-First Search Algorithm . 11

3 Trellis Code Design for Block Fading Channels 14

3.1 Code Construction 14

4 Decoding Algorithm and the Simulation Results 21

4.1 Decoding Algorithm 21

4.2 Simulation Results 22

4.3 Discussions on Simulation Results 51

5 Conclusion Remarks and Future Work 53

Bibliography 55

List of Figures

2.1 Example of a code tree with b_1 fixed as -1. 6

2.2 The trellis obtained by merging equivalent sub-trees in Fig.2.1. 7

3.1 Illustration of the merging process. 18

4.1 Word error rates (WERs) for the computer-searched code found by simulated annealing in [1] (SA(ML)), the SOTS code with single code tree (SOTS(SP)) in [2] which is also the code with maximum state number 2048 (SOTS-2048(SP)), the code with maximum state number 512 (SOTS-512(SP)), and the code with maximum state number (SOTS-128(SP)). The SA(ML) code is decoded by the ML decoder, while all the other codes are decoded by the SP decoder. The code rate is 1/2 and the codeword length is $N = 22$. 24

4.2 The average numbers of path expansions per information bit for the codes examined in Figure 4.1. 25

4.3 Word error rates (WERs) for the self-orthogonal trellis-structure codes of length $N = 18$ with maximal state number 128, 256 and 512, respectively. The decoders used are the sequential decoding with heuristic prediction (SP). The code rate is 1/2. 26

4.4 The average numbers of path expansions per information bit for the codes examined in Figure 4.3. 27

4.5 The average numbers of path expansions per information bit for the codes examined in Figure 4.3 except the codes are now decoded by the sequential decoding algorithm with no heuristic prediction (SNP). 28

4.6 Word error rates (WERs) for the proposed codes respectively with $S = 128$, 256 and 512, decoded by different approaches: ML, SP and SNP. The code rate is 1/2 and the codeword length is $N = 18$. 29

4.7 The average numbers of path expansions per information bit for the codes examined in Figure 4.6. 30

4.8 Word error rates (WERs) for the self-orthogonal trellis-structure codes of length $N = 20$ with maximal state number 128, 256, 512 and 1024, respectively. The decoders used are the sequential decoding with heuristic prediction (SP). The code rate is 1/2. 31

4.9 The average numbers of path expansions per information bit for the codes examined in Figure 4.8. 32

4.10 The average numbers of path expansions per information bit for the codes examined in Figure 4.8 except the codes are now decoded by the sequential decoding algorithm with no heuristic prediction (SNP). 33

4.11 Word error rates (WERs) for the proposed codes respectively with $S = 128$, 256, 512 and 1024, decoded by different approaches: ML, SP and SNP. The code rate is 1/2 and the codeword length is $N = 20$. 34

4.12 The average numbers of path expansions per information bit for the codes examined in Figure 4.11. 35

4.13 Word error rates (WERs) for the self-orthogonal trellis-structure codes of length $N = 22$ with maximal state number 128, 256, 512, 1024 and 2048, respectively. The decoders used are the sequential decoding with heuristic prediction (SP). The code rate is 1/2. 36

4.14 The average numbers of path expansions per information bit for the codes examined in Figure 4.13. 37

4.15 The average numbers of path expansions per information bit for the codes examined in Figure 4.13 except the codes are now decoded by the sequential decoding algorithm with no heuristic prediction (SNP). 38

4.16 Word error rates (WERs) for the proposed codes respectively with $S = 128$, 256, 512, 1024 and 2048, decoded by different approaches: ML, SP and SNP. The code rate is 1/2 and the codeword length is $N = 22$. 39

4.17 The average numbers of path expansions per information bit for the codes examined in Figure 4.16. 40

4.18 Word error rates (WERs) for the self-orthogonal trellis-structure codes of length $N = 24$ with maximal state number 128, 256, 512, 1024, 2048 and 4096, respectively. The decoders used are the sequential decoding with heuristic prediction (SP). The code rate is 1/2. 41

4.19 The average numbers of path expansions per information bit for the codes examined in Figure 4.18. 42

4.20 The average numbers of path expansions per information bit for the codes examined in Figure 4.18 except the codes are now decoded by the sequential decoding algorithm with no heuristic prediction (SNP). 43

4.21 Word error rates (WERs) for the proposed codes respectively with $S = 128$, 256, 512, 1024, 2048 and 4096, decoded by different approaches: ML, SP and SNP. The code rate is 1/2 and the codeword length is $N = 24$. 44

4.22 The average numbers of path expansions per information bit for the codes examined in Figure 4.21. 45

4.23 Word error rates (WERs) for the self-orthogonal trellis-structure codes of length $N = 26$ with maximal state number 256, 1024 and 8192, respectively. The decoders used are the sequential decoding with heuristic prediction (SP). The code rate is 1/2. 46

4.24 The average numbers of path expansions per information bit for the coded examined in Figure 4.23. 47

4.25 The average numbers of path expansions per information bit for the codes examined in Figure 4.23 except the codes are now decoded by the sequential decoding algorithm with no heuristic prediction (SNP). 48

4.26 Word error rates (WERs) for the proposed codes respectively with $S = 256$, 512, 1024 and 8192, decoded by different approaches: ML, SP and SNP. The code rate is 1/2 and the codeword length is $N = 26$. 49

4.27 The average numbers of path expansions per information bit for the codes examined in Figure 4.26. 50

Chapter 1

Introduction

1.1 Overview

In a typical communication system, the receiver will perform channel estimation, channel equalization, and error correction separately in sequence. The channel estimator estimates channel parameters based on the received training sequences while the equalizer uses the estimated channel parameters to eliminate the channel effect. Error correction is subsequently performed. The good performance of such a typical system is secured based on the assumption that the estimates of channel parameters are adequately accurate. However, this assumption may be hard to achieve under severe channel fades. Some recent researches [4, 5, 6, 1] therefore proposed to jointly consider these system devices, and confirmed its potential to achieve a good performance by analysis and simulations. At this background, this thesis will focus on a trellis code design for combined channel estimation and error correction for fading channels.

In 2002, Skoglund *et al.* [1] showed that the previously mentioned joint design can perform better than the separate design under quasi-static multipath block fading. In particular, Skoglund *et al.* found the best non-linear code for this environment by computer search, and confirmed that the computer searched code could outperform the traditional separately

designed system with perfect channel estimation by 2 dB. However, only the exhaustive search can be used to decode the computer-searched code due to its unsystematic structure.

To comment on this problem, Wu *et al.* [2] proposed a systematic construction of codes for combined channel estimation and error correction. Through simulations, they showed that these codes have almost the same performance as the computer-searched codes in [1]. The main advantage of their structural codes is that the codes can be maximum-likelihoodly decodable by the priority-first search decoding algorithm; hence, the decoding complexity reduces significantly. Yet, the decoding complexity still grows exponentially with the codeword length because of the tree structure of the codes.

In this thesis, we tried to fit the code from [2] into a trellis structure in order to reduce the growing decoding complexity with respect to the codeword length. In [2], the codeword selection process is to list all the candidate sequences in their binary alphabetical orders, and pick the codewords from the ordered list in a fixed interval. We then proposed an algorithm to compress its respective code tree to fit into a trellis with the number of states at each level being smaller than a given number in a way that the intervals between chosen codewords are made as similar as possible to that between codewords in [2]. Our target is to reduce the decoding complexity with little performance degradation.

The rest of the thesis is organized as follows. Chapter 2 depicts the system model and addresses the background. Chapter 3 introduces the algorithm to generate the compressed trellis code. Chapter 4 talks about the decoder and discusses simulation results of this code. Chapter 5 concludes the thesis.

1.2 Acronyms and Notations

The acronyms and some common identifiers used in this thesis are listed as follows.

AWGN additive white Gaussian noise

BER bit error rate

ML maximum-likelihood

SNR signal-to-noise ratio

WER word error rate

N block length of a code, i.e., the number of symbols in a codeword

K number of bits in an information sequence to be encoded

P number of channel paths or taps

L $L = N + P - 1$ is the sum of a codeword length plus one minimum guard interval

The following notations are used in this thesis.

Symbol	Meaning
$\boldsymbol{v}$	a vector *(The following notations are simple representative examples. Similar notations applies to other alphabets.)*
v_k	the k-th component of a vector
$\|\boldsymbol{v}\|^2$	the norm of a vector
$\mathbb{X}$	a matrix
$x_{k,\ell}$	the element of a matrix at row k and column ℓ
$\mathbb{X}^T$	the matrix transpose operation
$\mathbb{X}^H$	the matrix Hermitian transpose operation
$\mathbb{I}_L$	the $L \times L$ identity matrix
$\|\mathcal{S}\|$	the cardinality of a set $\mathcal{S}$, i.e., the number of members in $\mathcal{S}$

Chapter 2

Preliminaries

In this chapter, some background knowledge is provided. Specifically, we introduce the joint maximum-likelihood estimation of data sequence and channel effect in Section 2.1. Section 2.2 then gives a brief introduction of the code tree for an (N, K) code, followed by Section 2.3 that addresses the systematic construction for codes in [2] . Section 2.4 talks about the maximum-likelihood priority-first search decoding algorithm, which will be used for decoding the designed code in Chapter 3. This chapter ends in Section 2.5, which presents the metric designed in [2] for use of the priority-first search decoding algorithm.

2.1 Joint Maximum-Likelihood Estimation of Data Sequence and Channel Effect

This section introduces the system model for joint maximum-likelihood estimation of data sequence and channel effect. The system model adopted in this thesis is the same as that in [2].

We transmit an length-N codeword $\boldsymbol{b} = [b_1, \ldots, b_N]^T$ of an (N, K) code $\mathcal{C}$ through the block fading channel with memory order $P - 1$. Denote by $\boldsymbol{h}$ the $P \times 1$ channel parameter vector, which is assumed to be *constant* within a coding block. Hence, the received vector

$\boldsymbol{y} = [y_1, \ldots, y_L]$ can be represented as

$$\boldsymbol{y} = \mathbb{B}\boldsymbol{h} + \boldsymbol{n}$$

where

$$\mathbb{B} \triangleq \begin{bmatrix} b_1 & 0 & \cdots & 0 \\ \vdots & b_1 & \ddots & \vdots \\ b_N & \vdots & \ddots & 0 \\ 0 & b_N & \ddots & b_1 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & b_N \end{bmatrix}_{L \times P}, \tag{2.1}$$

each $b_i \in \{-1, +1\}$, L is equal to $N + P - 1$, and $\boldsymbol{n}$ is an $L \times 1$ zero mean Gaussian noise vector with covariance matrix $\sigma^2 \mathbb{I}_L$.

The assumption in this system model is that both the transmitter and the receiver know P (or its upper bound), yet they have no idea about the channel coefficient $\boldsymbol{h}$. Because $\boldsymbol{h}$ is unknown, the joint maximum-likelihood (JML) detection becomes the best option:

$$(\hat{\boldsymbol{b}}, \hat{\boldsymbol{h}}) = \arg \max_{\boldsymbol{b}} \max_{\boldsymbol{h}} \Pr(\boldsymbol{y}|\mathbb{B}, \boldsymbol{h}).$$

For a given $\boldsymbol{b}$, the channel coefficient $\boldsymbol{h}$ that maximizes $\Pr(\boldsymbol{y}|\mathbb{B}, \boldsymbol{h})$ can be pre-determined, i.e.,

$$\hat{\boldsymbol{h}} = \arg \max_{\boldsymbol{h}} \Pr(\boldsymbol{y}|\mathbb{B}, \boldsymbol{h}).$$

Then, the JML estimate of the transmitted codeword is

$$\begin{aligned} \hat{\boldsymbol{b}} &= \arg \max_{\boldsymbol{b}} \Pr(\boldsymbol{y}|\mathbb{B}, \hat{\boldsymbol{h}}) \\ &= \arg \min_{\boldsymbol{b}} \|\boldsymbol{y} - \mathbb{B}(\mathbb{B}^H \mathbb{B})^{-1} \mathbb{B}^H \boldsymbol{y}\|^2 \\ &= \arg \min_{\boldsymbol{b}} \|\boldsymbol{y} - \mathbb{P}_B \boldsymbol{y}\|^2, \end{aligned} \tag{2.2}$$

where $\mathbb{P}_B \triangleq \mathbb{B}(\mathbb{B}^H \mathbb{B})^{-1} \mathbb{B}^H$. Noted that $\mathbb{B}$ and $\mathbb{P}_B$ are not one to one correspondence unless the first bit b_1 is fixed as, for example, -1.

In (2.2), the estimator requires only the received signal $\boldsymbol{y}$ and $\mathbb{P}_B$. To simplify the decoding process, the $\mathbb{P}_B$ table can be off-line generated.

2.2 The Code Tree Structure

An (N, K) binary code can be represented as a code tree as shown in Fig. 2.2. The code tree consists of $(N+1)$ levels. The leftmost node at level zero is usually called the *original node*, which is regarded to be the root of the code tree. There are at most two branches leaving from each node. Note that the two branches from the same node respectively correspond to code bits of different value. Hence, each branch from nodes at level i marks the $(i+1)$th code bit value in a codeword. The 2^K nodes at level N are called the *terminal nodes*.

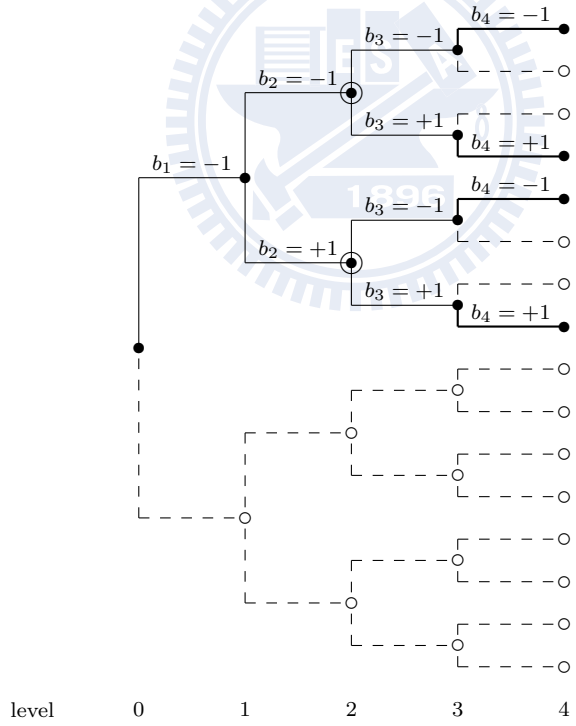

Figure 2.1: Example of a code tree with b_1 fixed as -1.

Each codeword can be fully characterized as one path from the original node to one of the terminal nodes, or equivalently as the branch labels traversed along the code path.

Furthermore, every node in the code tree can be viewed as a new original node of a smaller sub-tree. In other words, all the paths from the new original node to its inherited terminal nodes form a new sub-tree. If two different sub-trees consist of the same branch labels, we say the two sub-trees are equivalent. As an example in Fig. 2.1, the two sub-trees originated from the two circled nodes are equivalent.

In our algorithm, when two sub-trees are equivalent, we will combine or merge them. The so-called "merge" is to use one single sub-tree to substitute the original two sub-trees. Thus, when we draw a trellis, only one of the sub-trees will appear. The branches originally going through the nodes on two separate sub-trees will now go through the nodes on the merged sub-tree. Apparently, the new structure is no longer a tree. By following this process, the merged counterpart of the code tree in Fig. 2.1 is depicted in Fig. 2.2.

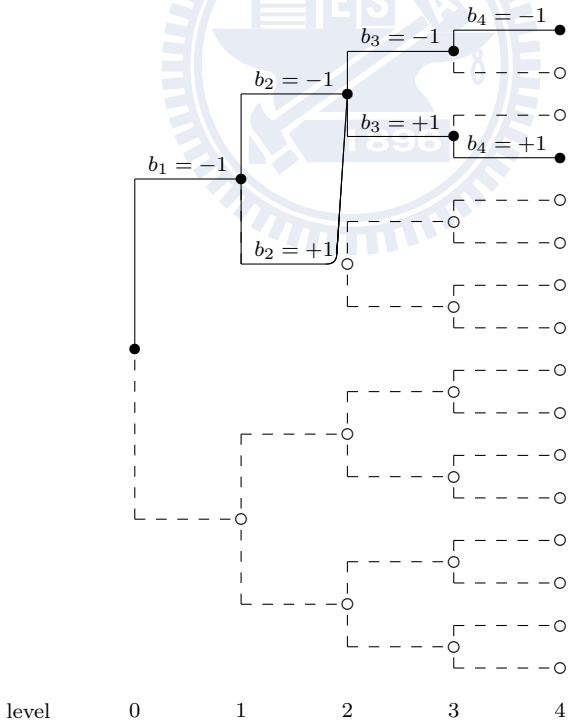

Figure 2.2: The trellis obtained by merging equivalent sub-trees in Fig.2.1.

2.3 The Systematic Code Design for Combined Channel Estimation and Error Correction

This section is to give a brief introduction of the code proposed in [2] as well as the basic idea behind our trellis construction. For convenience, we will call the code in [2] *the self-orthogonal tree-structure* (SOTS) code in the following. In this paper, authors found that the average SNR can be optimized if the codeword is *self-orthogonal* to its shift counterpart. In other words, the code is designed to satisfy

$$\frac{1}{N}\mathbb{B}^T\mathbb{B} = \mathbb{I}_P. \tag{2.3}$$

Unfortunately, codewords that satisfy this formula exist only when the codeword length N is odd. When N is even, some off-diagonal elements in $\mathbb{B}^T\mathbb{B}$ must be relaxed to either -1 or 1. The case $P = 2$ is specifically considered in [2], where

$$\mathbb{B}^T\mathbb{B} = \begin{bmatrix} N & \pm 1 \\ \pm 1 & N \end{bmatrix} \text{ for } N \text{ is even}, \tag{2.4}$$

and

$$\mathbb{B}^T\mathbb{B} = \begin{bmatrix} N & 0 \\ 0 & N \end{bmatrix} \text{ for } N \text{ is odd}. \tag{2.5}$$

The best computer-searched code with $N = 22$ in [1] actually all satisfies (2.4) even though it is kind of structureless. In [2], the codeword selection process that intends to generate a code with a structure of use to the decoder is to list all the codewords satisfying (2.4) or (2.5) in a binary-alphabetical order, and then uniformly pick the codewords from this list in every Δ_θ interval with

$$\Delta_\theta = \frac{|\mathcal{A}(b_1 = -1|\mathbb{G}_\theta)| - 1}{2^K/|\Theta| - 1} \quad \text{for } \theta \in \Theta, \tag{2.6}$$

where

$$\mathbb{G}_\theta \triangleq \begin{bmatrix} N & \theta \\ \theta & N \end{bmatrix} \text{ for } \theta \in \{-1, 0, +1\},$$

and $\mathcal{A}(b_1 = -1|\mathbb{G}_\theta)$ is the set of all length-N binary sequences whose first bit is fixed as -1, and for which $\mathbb{B}^T\mathbb{B} = \mathbb{G}_\theta$ is satisfied. Besides, Θ is $\{0\}$ for N odd, and Θ is $\{-1, 1\}$ for N even. In this thesis, we only consider the *single tree* case in [2] because to combine two trees respectively from $\theta = -1$ and $\theta = +1$ would require additional consideration, and hence we defer it as a future work. As a result, Θ is $\{0\}$ for N odd, and Θ is $\{1\}$ for N even in this thesis.

For completeness, the codeword selection algorithm of the SOTS code built over a single tree is reproduced in the following:

Step 1. Input the index i of the requested codeword in the (N, K) block code, where $0 \le i \le 2^K - 1$.

Step 2. Set $\Theta = \{0\}$ for N odd, and $\Theta = \{1\}$ for N even. Also, set

$$\theta = ((N+1) \bmod 2) \cdot (-1)^{\lceil (i+1)/(2^K/|\Theta|) \rceil}.$$

Compute Δ_θ according to (2.6). Initialize $b_1 = -1$, $\ell = 1$ and

$$\rho = \lfloor (i \bmod (2^K/|\Theta|)) \cdot \Delta_\theta \rfloor.$$

Let the minimum sequence index $\rho_{\min} = 0$.

Step 3. Execute $\ell = \ell + 1$, and compute $\gamma_\ell = |\mathcal{A}(\boldsymbol{b}_{(\ell-1)}, b_\ell = -1|\mathbb{G}_\theta)|$.

If $\rho < \rho_{\min} + \gamma_\ell$, then choose the next code bit to be $b_\ell = -1$;

otherwise choose the next code bit to be $b_\ell = 1$, and readjust $\rho_{\min} = \rho_{\min} + \gamma_\ell$.

Step 4. If $\ell = N$, output the corresponding codeword $\boldsymbol{b}$, and the algorithm stops; otherwise, go to Step 3.

In this algorithm, the codewords are built over a single tree. In the following sections, we will attempt to combine these codewords so as to fit them into a trellis with bounded number of nodes at each level.

2.4 The Priority-First Search Algorithm

The *priority-first search algorithm* is a common graph search algorithm that traverses a graph by expanding the most possible path. Considering the code tree introduced in Figure 2.2, we can use the priority-first search algorithm to decode the respective code over it. A typical priority-first search algorithm can be described as follows:

Step 1. Load the stack with the path that ends at the original node.

Step 2. Evaluate the metric values of the successor paths of the current top path in the stack. Then delete this top path from the stack.

Step 3. Insert the successor paths obtained in Step 2 into the stack such that the paths in the stack are ordered according to their ascending metric values.

Step 4. If the top path in the stack ends at a terminal node in the code tree, output the labels corresponding to the top path, and the algorithm stops; otherwise, go to Step 2.

One main difference among various priority-first search algorithms is the *metric*. Some *metric* designs guarantee the optimal search results, while some others cannot. For the SOTS code, the metric designed in [2] can achieve the same performance as an ML decoder. We will introduce this optimal decoding metric in the next section.

2.5 Maximum-Likelihood Decoding Metric for Priority-First Search Algorithm

Since we will use the metric designed in [2] to do the decoding, the introduction of the idea behind its design becomes necessary. For simplicity, we denote the path ending at a node at level ℓ by the sequence of branch labels $\boldsymbol{b}_{(\ell)} = [b_1, b_2, \ldots, b_\ell]$ it traverses. The subscript in $\boldsymbol{b}_{(\ell)}$ will be dropped when $\ell = N$. The metric can be represented as the sum of two components:

$$f(\boldsymbol{b}_{(\ell)}) \triangleq g(\boldsymbol{b}_{(\ell)}) + \varphi(\boldsymbol{b}_{(\ell)}).$$

The former component is directly designed based on the maximum-likelihood metric such that

$$\arg\min_{\boldsymbol{b}\in\mathcal{C}} g(\boldsymbol{b}) = \arg\min_{\boldsymbol{b}\in\mathcal{C}} \|\boldsymbol{y} - \mathbb{P}_B \boldsymbol{y}\|^2.$$

The second component φ is the prediction part, which is usually called heuristic function. The metric function values for all codeword portions traversed thus far are all compared and sorted in a stack. The path chosen to be extended is the one whose metric value is the smallest in the stack. Let the (i,j)th entry of matrix $(\mathbb{G}_\theta)^{-1}$ be denoted by $\delta_{i,j}^{(\theta)}$. Then

$$g(\boldsymbol{b}_{(\ell)}) \triangleq \sum_{m=1}^{\ell} \max_{\eta\in\Theta} \left(\sum_{n=1}^{m-1} |w_{m,n}^{(\eta)}| + \frac{1}{2}|w_{m,m}^{(\eta)}| \right) - \frac{1}{2}\sum_{m=1}^{\ell}\sum_{n=1}^{\ell} w_{m,n}^{(\theta)} b_m b_n,$$

where

$$w_{m,n}^{(\theta)} = \sum_{i=0}^{P-1}\sum_{j=0}^{P-1} \delta_{i,j}^{(\theta)} \mathrm{Re}\{y_{m+i} y_{n+j}^*\}.$$

Accordingly,

$$g(\boldsymbol{b}_{(\ell+1)}) \;=\; g(\boldsymbol{b}_{(\ell)}) + \max_{\eta\in\Theta} \alpha_{\ell+1}^{(\eta)} - b_{\ell+1} \sum_{i=0}^{P-1}\sum_{j=0}^{P-1} \delta_{i,j}^{(\theta)} \mathrm{Re}\left\{ y_{\ell+i+1} \cdot u_j(\boldsymbol{b}_{(\ell+1)}) \right\}, \qquad (2.7)$$

where

$$\alpha_{\ell+1}^{(\eta)} \triangleq \sum_{n=1}^{\ell} |w_{\ell+1,n}^{(\eta)}| + \frac{1}{2}|w_{\ell+1,\ell+1}^{(\eta)}|,$$

and for $0 \le j \le P-1$,

$$u_j(\boldsymbol{b}_{(\ell+1)}) \triangleq \sum_{n=1}^{\ell} b_n y^*_{n+j} + \frac{1}{2} b_{\ell+1} y^*_{\ell+j+1} = u_j(\boldsymbol{b}_{(\ell)}) + \frac{1}{2}\left(b_\ell y^*_{\ell+j} + b_{\ell+1} y^*_{\ell+1+j}\right).$$

By recursion, we can obtain $g(\boldsymbol{b}_{(\ell+1)})$ and $\{u_j(\boldsymbol{b}_{(\ell+1)})\}_{0\le j\le P-1}$ from $g(\boldsymbol{b}_{(\ell)})$ and $\{u_j(\boldsymbol{b}_{(\ell)})\}_{j=0}^{P-1}$ using $y_{\ell+1}, y_{\ell+2}, \ldots, y_{\ell+P}$ and $b_{\ell+1}$ with $g(\boldsymbol{b}_{(0)}) = u_j(\boldsymbol{b}_{(0)}) = b_0 = 0$ for $0 \le j \le P-1$ as the initial condition.

Next, we consider the second prediction part. It can be viewed as the prediction to help speeding up the decoding. If no prediction is made, $\varphi(\boldsymbol{b}_{(\ell)})$ is set to zero, which is termed *zero heuristic function* in [2]. In order to remain optimality, the heuristic function with prediction should satisfy

$$\begin{aligned}\varphi(\boldsymbol{b}_{(\ell)}) \;\le\; & \sum_{m=\ell+1}^{N} \max_{\eta\in\Theta} \alpha_m^{(\eta)} \\ & - \max_{\{\tilde{\boldsymbol{b}}\in\mathcal{C}:\, \tilde{\boldsymbol{b}}_{(\ell)}=\boldsymbol{b}_{(\ell)}\}} \left(\sum_{m=\ell+1}^{N} \tilde{b}_m \sum_{n=1}^{\ell} w_{m,n}^{(\theta)} b_n + \frac{1}{2} \sum_{m=\ell+1}^{N} \sum_{n=\ell+1}^{N} w_{m,n}^{(\theta)} \tilde{b}_m \tilde{b}_n \right), \quad (2.8)\end{aligned}$$

which the zero heuristic fuction apparently upholds.

Upon reception of all $y_1, \ldots, y_L$, the heuristic function that satisfies (2.8) regardless of $\tilde{b}_{\ell+1}, \ldots, \tilde{b}_N$ can be increased up to

$$\varphi(\boldsymbol{b}_{(\ell)}) \;\triangleq\; \sum_{m=\ell+1}^{N} \max_{\eta\in\Theta} \alpha_m^{(\eta)} - \sum_{m=\ell+1}^{N} \left| v_m^{(\theta)}(\boldsymbol{b}_{(\ell)}) \right| - \beta_\ell^{(\theta)}, \quad (2.9)$$

where for $1 \le \ell, m \le N$ and $\theta \in \Theta$,

$$v_m^{(\theta)}(\boldsymbol{b}_{(\ell)}) \triangleq \sum_{n=1}^{\ell} w_{m,n}^{(\theta)} b_n = v_m^{(\theta)}(\boldsymbol{b}_{(\ell-1)}) + b_\ell w_{\ell,m}^{(\theta)}$$

and

$$\beta_\ell^{(\theta)} \triangleq \sum_{m=\ell+1}^{N} \left(\sum_{n=\ell+1}^{m-1} |w_{m,n}^{(\theta)}| + \frac{1}{2}|w_{m,m}^{(\theta)}| \right) = \beta_{\ell-1}^{(\theta)} - \sum_{n=\ell+1}^{N} |w_{\ell,n}^{(\theta)}| - \frac{1}{2}|w_{\ell,\ell}^{(\theta)}|$$

with initial conditions $v_m^{(\theta)}(\boldsymbol{b}_{(0)}) = b_0 = 0$, and $\beta_0^{(\theta)} = \sum_{m=1}^{N} \alpha_m^{(\theta)}$.

The above metric design guarantees maximum likelihood performance for the code design in [2] when it is decoded by the priority-first search decoding. Note that after merging the code tree into a trellis, the same metric no longer guarantees optimal performance because the branch metric to be added in the recursive computation in (2.7) depends on all the branches it has traversed. The contribution of this thesis is therefore the provision of a merging strategy such that the resultant suboptimal performance is still close to the optimal one.

Chapter 3

Trellis Code Design for Block Fading Channels

The tree structure makes the decoding complexity of the SOTS code grow exponentially with the codeword length. The goal of this thesis is then to reduce the decoding complexity by bringing in the trellis structure. In order to simplify the design, we consider only single-tree codes in [2] with codeword length N being even in this thesis.

In the sequel, we will fit the tree-based SOTS code into a trellis structure by selectively merging tree nodes through an algorithmic procedure. This will lead to an apparent trade-off between memory consumption, decoding complexity and performance. Generally speaking, the more the merged nodes, the less the memory consumption and decoding complexity, but the worse the performance. The contribution of this thesis is that we can systematically control such a trade-off through a proper choice of number of trellis state.

3.1 Code Construction

As shown in Figure 2.1, an SOTS code tree is composed of multiple levels with exponentially increasing number of nodes. We wish to merge these nodes so that the number of remaining nodes at each level is bounded above by a chosen constant S.

The node-merging algorithm can be roughly described as follows. We first examine the number of nodes at each level, starting at level 0. If the number of nodes at the current level is larger than the upper limit S, the nodes that root identical sub-codetrees will be located and merged into one node until the number of remaining nodes at the current level is no larger than S. In case there exist no nodes that root identical sub-codetrees, we will find and adjust similar sub-codetrees, and then merge their root nodes. Note that after adjustment, the codewords will be different from those in the original SOTS code; hence, the performance may degrade. The adjustment should therefore be designed such that the performance degradation can be made as small as possible. We will carry out the merging process for all levels. For ease of referring them, the self-orthogonal trellis-structure codes with number of states no larger than S will be abbreviated as SOTS-S.

Next, we introduce some notations used in the code construction algorithm. $\mathsf{D}(\cdot)$ is a mapping function that simply returns the decimal value of a binary sequence $\boldsymbol{a} = [a_1, a_2, \ldots, a_\ell]$, where each $a_j \in \{0, 1\}$. Specifically,

$$\mathsf{D}([a_1, a_2, \ldots, a_\ell]) = a_1 \cdot 2^{\ell-1} + a_2 \cdot 2^{\ell-2} + \cdots + a_\ell.$$

Note that the antipodal binary sequence $\boldsymbol{b}$ introduced in Chapter 2 can be obtained from $\boldsymbol{a}$ through $b_j = (-1)^{a_j}$.

In a code tree as well as a sub-codetree, there is only one code path entering each node; hence, we can refer to this node at level ℓ by the decimal value V of $\boldsymbol{a}_{(\ell)}$ labelling the entering path, i.e., $V = \mathsf{D}(\boldsymbol{a}_{(\ell)})$. The same function will also be used to generate the decimal value corresponding to a node in the code trellis. Apparently, a trellis node may have multiple entering paths; hence in a code trellis, the smallest decimal value corresponding to these entering paths will be chosen as its representative identification decimal value.

In a code tree, all the paths originating from a node at level ℓ to all terminal nodes form a sub-codetree. The sub-codetree with root node V is denoted by T_V. In order to

facilitating the identification of a path in a sub-codetree, we further denote by $\tilde{\boldsymbol{a}}_{(\ell+1)} = [a_{\ell+1}, a_{\ell+2}, \ldots, a_N]$ the path from a node at level ℓ to a terminal node.

Before adjusting two similar sub-codetrees prior to their merging, we will index their code paths in binary alphabetical order. In notations, $I_{V,\tilde{\boldsymbol{a}}_{(\ell+1)}}$ denotes the ordered index of path $\tilde{\boldsymbol{a}}_{(\ell+1)}$ in sub-codetree T_V, and $N_{V,\tilde{\boldsymbol{a}}_{(\ell+1)}}$ is the number of code paths $\boldsymbol{a} = [a_1, a_2, \ldots, a_N]$ whose last $(N-\ell)$ bits are equal to $\tilde{\boldsymbol{a}}_{(\ell+1)}$ in sub-codetree T_V. Denote by $D_{V,I}$ and $\tilde{\boldsymbol{a}}(V, I)$ the decimal value and code labels of the I-th path in sub-codetree T_V, respectively. In other words, $D_{V,I} = \mathsf{D}(\tilde{\boldsymbol{a}}(V, I))$. We also let $\mathcal{E}_V$ and $\tilde{\mathcal{E}}_V$ respectively be the sets containing all code path portions ending at and originating from node V. In addition, $\mathcal{T}(\ell)$ is the set that consists of all sub-codetrees with root nodes at level ℓ.

All the above notations regard the code paths on an SOTS code tree. Next we introduce the notations concerning those of general paths not necessarily on the original SOTS code tree. A set $\tilde{\mathcal{O}}_V$ is one containing all binary sequence $\tilde{\boldsymbol{a}}_{(\ell+1)}$ satisfying $\mathbb{B}^T\mathbb{B} = \mathbb{G}_\theta$ for every $\boldsymbol{a}_{(\ell)}$ in $\mathcal{E}_V$, where $\mathbb{B}$ is defined in (2.1) with $\boldsymbol{b} = [\boldsymbol{b}_{(\ell)} \;\; \tilde{\boldsymbol{b}}_{(\ell+1)}]^T$, and each $b_j = (-1)^{a_j}$ and $\tilde{b}_j = (-1)^{\tilde{a}_j}$. The purpose of defining the set $\tilde{\mathcal{O}}_V$ is to guarantee that the adjustment of sub-codetrees is always conducted over self-orthogonal binary sequences. Similarly, we let $\overline{\mathcal{O}}_V$ be the set that contains all binary sequence $\tilde{\boldsymbol{a}}_{(\ell+1)}$ satisfying $\mathbb{B}^T\mathbb{B} = \mathbb{G}_\theta$ for some $\boldsymbol{a}_{(\ell)}$ in $\mathcal{E}_V$. We emphasize that $\tilde{\boldsymbol{a}}_{(\ell+1)} \in \overline{\mathcal{O}}_V$ as long as there exists *one* $\boldsymbol{a}_{(\ell)} \in \mathcal{E}_V$ such that $\mathbb{B}^T\mathbb{B} = \mathbb{G}_\theta$, while $\tilde{\boldsymbol{a}}_{(\ell+1)} \in \tilde{\mathcal{O}}_V$ only when $\mathbb{B}^T\mathbb{B} = \mathbb{G}_\theta$ for *every* $\boldsymbol{a}_{(\ell)} \in \mathcal{E}_V$; hence, by their definitions, we have $\tilde{\mathcal{O}}_V \subset \overline{\mathcal{O}}_V$.

Finally, we provide two functions $h(\cdot,\cdot)$ and $\hat{h}(\cdot,\cdot)$ for the selection of sub-codetrees to be merged. Function $h(\cdot,\cdot)$ is used to select two among identical sub-codetrees to be merged, while $\hat{h}(\cdot,\cdot)$ is for the situation when there are no identical sub-codetrees at the same level. Different design of $h(\cdot,\cdot)$ and $\hat{h}(\cdot,\cdot)$ may yield different code construction. The ones we used

are given by

$$h(V_1, V_2) \triangleq \frac{|\overline{\mathcal{O}}_{V_1}| + |\overline{\mathcal{O}}_{V_2}|}{\max\limits_{\tilde{\boldsymbol{a}}_{(\ell+1)} \in \tilde{\mathcal{E}}_{V_1}} N_{V_1, \tilde{\boldsymbol{a}}_{(\ell+1)}} + \max\limits_{\tilde{\boldsymbol{a}}_{(\ell+1)} \in \tilde{\mathcal{E}}_{V_2}} N_{V_2, \tilde{\boldsymbol{a}}_{(\ell+1)}}},$$

and

$$\hat{h}(V_1, V_2) \triangleq \frac{|\tilde{\mathcal{O}}_{V_1} \cap \tilde{\mathcal{O}}_{V_2}|}{1 + |D_{V_1,1} - D_{V_2,1}|},$$

where we abuse the notation to use $|\cdot|$ to denote both the number of elements in a set and the absolute value (of the difference of two decimal values). The objective of our design criterion is to have self-orthogonal codewords as many as possible after the two selected sub-codetrees are merged; so we put $|\overline{\mathcal{O}}_{V_1}| + |\overline{\mathcal{O}}_{V_2}|$ and $|\tilde{\mathcal{O}}_{V_1} \cap \tilde{\mathcal{O}}_{V_2}|$ in the nominators of $h(\cdot, \cdot)$ and $\hat{h}(\cdot, \cdot)$, respectively.

What follows is the algorithm to construct the trellis code based on an SOTS code:

Step 1. Initialize level $i = 0$. Set the maximal state number S.

Step 2. If $|\mathcal{T}(i)| \leq S$, go to Step 7.

Step 3. If there exist more than two identical sub-codetrees in $\mathcal{T}(i)$, merge the two with decimal values V_1 and V_2 that maximize h-function value among all identical sub-codetree pair in $\mathcal{T}(i)$ and go to Step 2.

Step 4. If $|\mathcal{T}(i)| \leq S$, go to Step 7.

Step 5. If there exist sub-codetree pairs in $\mathcal{T}(i)$ satisfying $|\tilde{\mathcal{E}}_{V_1}| = |\tilde{\mathcal{E}}_{V_2}|$, merge the two that maximize $\hat{h}$-function value among all equal-$|\tilde{\mathcal{E}}_V|$-value sub-codetree pair in $\mathcal{T}(i)$; else, go to Step 7.

Step 6. For the two merged sub-codetrees T_{V_1} and T_{V_2} in Step 5, calculate

$$W_s = \frac{N_{V_1,s} \cdot D_{V_1,s} + N_{V_2,s} \cdot D_{V_2,s}}{N_{V_1,s} + N_{V_2,s}}$$

for $1 \leq s \leq |\tilde{\mathcal{E}}_{V_1}|$. *Then, adjust*

$$\tilde{\boldsymbol{a}}(V_1, s) = \tilde{\boldsymbol{a}}(V_2, s) = \arg \min_{\tilde{\boldsymbol{a}}_{(i+1)} \in \tilde{\mathcal{O}}_{V_1} \cap \tilde{\mathcal{O}}_{V_2}} \left| \mathsf{D}\left(\tilde{\boldsymbol{a}}_{(i+1)}\right) - W_s \right|$$

for $1 \leq s \leq |\tilde{\mathcal{E}}_{V_1}|$. *Go to Step 4.*

Step 7. if $i < N$, $i = i + 1$ *and go to Step 2; else, stop the algorithm.*

As shown in Step 6, when merging two non-identical sub-codetrees, we will calculate the *weighted average* W_s for the respective codepaths in the two sub-codetrees. We then choose the one with decimal value nearest to W_s among all self-orthogonal candidates in $\tilde{\mathcal{O}}_{V_1} \cap \tilde{\mathcal{O}}_{V_2}$. The intuition behind this design is in a sense to "minimize" the adjustment so that the "decimal value interval" among codewords is kept as similar to that of the original codewords as possible.

In order to facilitate the understanding of the above algorithm, an example is provided below.

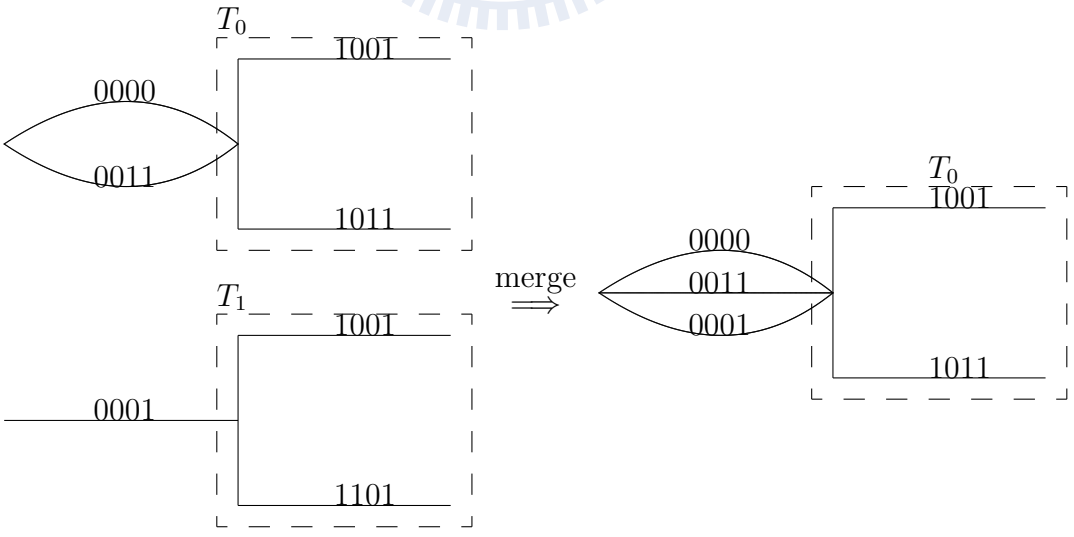

Figure 3.1: Illustration of the merging process.

In Fig. 3.1, we have two sub-codetrees indicated by two dashed boxes respectively with root nodes

$$V_1 = \min\{\mathsf{D}([0000]), \mathsf{D}([0011])\} = \min\{0, 3\} = 0 \quad \text{and} \quad V_2 = \mathsf{D}([0001]) = 1$$

at level $i = 3$. Hence, the two sub-codetrees are referred to as T_0 and T_1, respectively. Suppose the algorithm selects to merge these two sub-codetrees (with $|\tilde{\mathcal{E}}_1| = |\tilde{\mathcal{E}}_2| = 2$). We then have that $I_{0,[1001]} = 1$, $I_{0,[1011]} = 2$, $I_{1,[1001]} = 1$ and $I_{1,[1101]} = 2$. In addition, there are two codewords with suffix [1001] passing through V_1, i.e., [00001001] and [00111001]; hence, $N_{0,[1001]} = 2$. Similarly, $N_{0,[1011]} = 2$. By the same rule, we can easily obtain that $N_{1,[1001]} = N_{1,[1101]} = 1$.

We then base on $\mathcal{E}_0 = \{[0000], [0011]\}$ and $\mathcal{E}_1 = \{[0001]\}$ to get that

$$\tilde{\mathcal{O}}_0 \cap \tilde{\mathcal{O}}_1 = \{[1001], [1011], [1101]\}.$$

Note that every concatenation of sequence $\boldsymbol{a}_{(4)} \in \mathcal{E}_0 \cup \mathcal{E}_1$ and sequence $\tilde{\boldsymbol{a}}_{(5)} \in \tilde{\mathcal{O}}_0 \cap \tilde{\mathcal{O}}_1$ should be a self-orthogonal sequence (of length N). Next, we merge the two sequences with the same index $s = 1, 2$. The first sequence (i.e., $s = 1$) in T_0 is the same as the first sequence in T_1; therefore, $\tilde{\boldsymbol{a}}(0, 1) = \tilde{\boldsymbol{a}}(1, 1)$ remains unchanged by following Step 6 of the algorithm. When $s = 2$, the two sequences respectively in T_0 and T_1 are different. Following Step 6, we compute

$$W_2 = \frac{N_{0,[1011]} \cdot \mathsf{D}([1011]) + N_{1,[1101]} \cdot \mathsf{D}([1101])}{N_{0,[1011]} + N_{1,[1101]}} = \frac{2 \cdot 11 + 1 \cdot 13}{2 + 1} = 11.7.$$

We then determine the sequence in $\tilde{\mathcal{O}}_0 \cap \tilde{\mathcal{O}}_1$ whose decimal value is closest to W_2, which is [101] in this example. So, both $\tilde{\boldsymbol{a}}(0, 2)$ and $\tilde{\boldsymbol{a}}(1, 2)$ are replaced by [1011]. This results in a merged sub-codetree shown on the right of Fig. 3.1.

After merging, the new sub-codetree will be referred to as T_0 since

$$\min\{\mathsf{D}([0000]), \mathsf{D}([0011]), \mathsf{D}([0001])\} = 0.$$

$N_{0,\tilde{\boldsymbol{a}}_{(5)}}$ becomes 3 for all $\tilde{\boldsymbol{a}}_{(5)}$ in the new sub-codetree. $\mathcal{T}(4)$ has one less element; hence, $|\mathcal{T}(4)|$ is reduced by one. Other variables are changed accordingly. The algorithm will proceed until it reaches the final level of the code tree.

Chapter 4

Decoding Algorithm and the Simulation Results

This chapter we will show the decoding algorithm and the simulation results. The decoding algorithm is listed in section 4.1, simulation results are listed in section 4.2, and discussions are given at section 4.3.

4.1 Decoding Algorithm

Our decoding algorithm is slightly different from the one introduced in Section 2.4. The major change is due to that we need to consider that two paths may merge at a node in the trellis structure. In such case, only the most promising path remains in the stack. For completeness, the refined algorithm is given below.

Step 1. Load the stack with the path that ends at the original node.

Step 2. Evaluate the metric values of the successor paths of the current top path in the stack. Then delete this top path from the stack.

Step 3. If the successor paths end at the same node as some paths already in the stack as well as the other successor paths, delete all of them except the one with least

metric value.

Step 4. If there are undeleted successor paths after Step 3, insert these undeleted successor paths into the stack such that the paths in the stack are ordered according to their ascending metric values.

Step 5. If the top path in the stack ends at a terminal node in the code trellis, output the labels corresponding to the top path, and stop the algorithm; otherwise, go to Step 2.

The metric we use is the one defined in Section 2.5. However, it shall be pointed out that the metric no longer guarantees finding the ML code path in a code trellis because Step 3 may remove the ML code path when merging of two or more paths occur.

4.2 Simulation Results

In this section, we examine the codes proposed in Chapter 3 by simulations. In order to facilitate the comparison of our codes with the ones proposed in [1] and [2], we adopt the same system parameters therein. Specifically, channel coefficient $\boldsymbol{h}$ obeys zero-mean complex-Gaussian law with $E[\boldsymbol{h}\boldsymbol{h}^H] = (1/P)\mathbb{I}_P$ and $P = 2$, where $\mathbb{I}_P$ denotes the P-by-P identity matrix. The average system SNR is given by

$$\frac{N}{L\sigma_n^2}\mathrm{tr}\left(E[\boldsymbol{h}\boldsymbol{h}^H]\frac{1}{N}\mathbb{B}^T\mathbb{B}\right) = \frac{N}{L\sigma_n^2}\mathrm{tr}\left(\frac{1}{NP}\mathbb{B}^T\mathbb{B}\right) = \frac{N}{L\sigma_n^2} \tag{4.1}$$

because $\mathrm{tr}\left(\mathbb{B}^T\mathbb{B}\right) = NP$.

There will be three kinds of decoders used in our simulations. For simplify, we will adopt the following abbreviations, 'ML', 'SP' and 'SNP,' where they respectively stand for the maximum-likelihood (ML) decoder, the sequential (S) decoder introduced in Section

4.1 with heuristic prediction (P) function defined in (2.9), and the sequential (S) decoder introduced in Section 4.1 but with no heuristic prediction (NP) function.

As often done in literature [2], the decoding complexity is measured by the average path expansions per information bit. This complexity measure is conventionally considered to be proportional to the average number of metric calculations per information bit. In our case, however, such a convention is not exact because some expansions may require only one metric calculation while others need two metric calculations. Since the memory access such as insertion and deletion of path elements is in a sense a more significant key factor for decoding efficiency in nowadays computer technology, the adopted complexity measure is justified from the stack maintenance aspect.

Figures 4.1 and 4.2 compare our codes with the codes in [1] and [2]. The performance and complexity are both provided.

Then we examine the proposed codes of different codeword lengths. Specifically, Figures 4.3-4.7 are for codes of codeword length 18. Figures 4.8-4.12 are for codes of codeword length 20. Figures 4.13-4.17 are for codes of codeword length 22. Figures 4.18-4.22 are for codes of codeword length 24. And Figures 4.23-4.27 are for codes of codeword length 26.

For each length, we examine the proposed codes for different maximum state numbers and using different decoders. This results in five plots for each codeword length. The first plot shows the difference of word error rates (WER) among codes with different maximum state numbers. The second plot shows the complexity of the SP decoder for the codes examined in the first plot. The third plot turns to the complexity of the SNP decoder. And the fourth and fifth plots respectively compare the performance and complexity among different decoders.

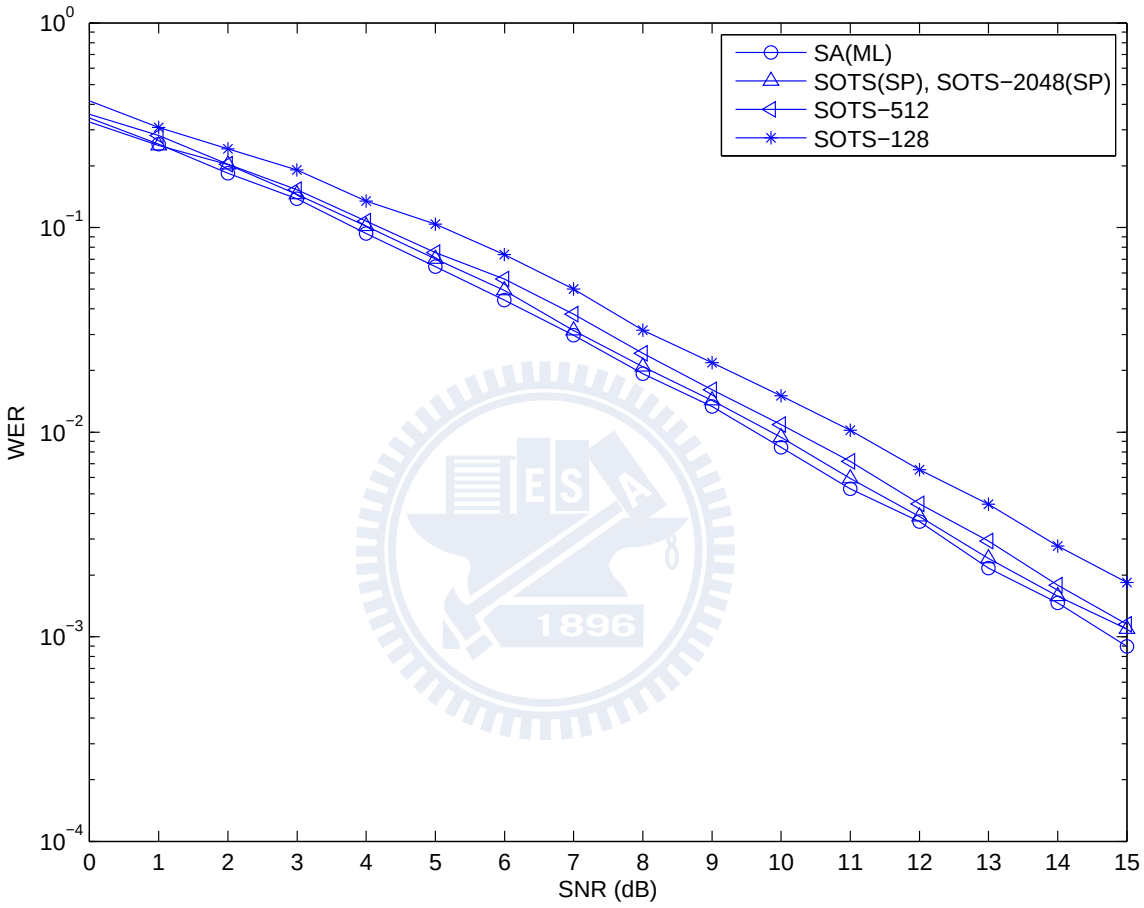

Figure 4.1: Word error rates (WERs) for the computer-searched code found by simulated annealing in [1] (SA(ML)), the SOTS code with single code tree (SOTS(SP)) in [2] which is also the code with maximum state number 2048 (SOTS-2048(SP)), the code with maximum state number 512 (SOTS-512(SP)), and the code with maximum state number (SOTS-128(SP)). The SA(ML) code is decoded by the ML decoder, while all the other codes are decoded by the SP decoder. The code rate is 1/2 and the codeword length is $N = 22$.

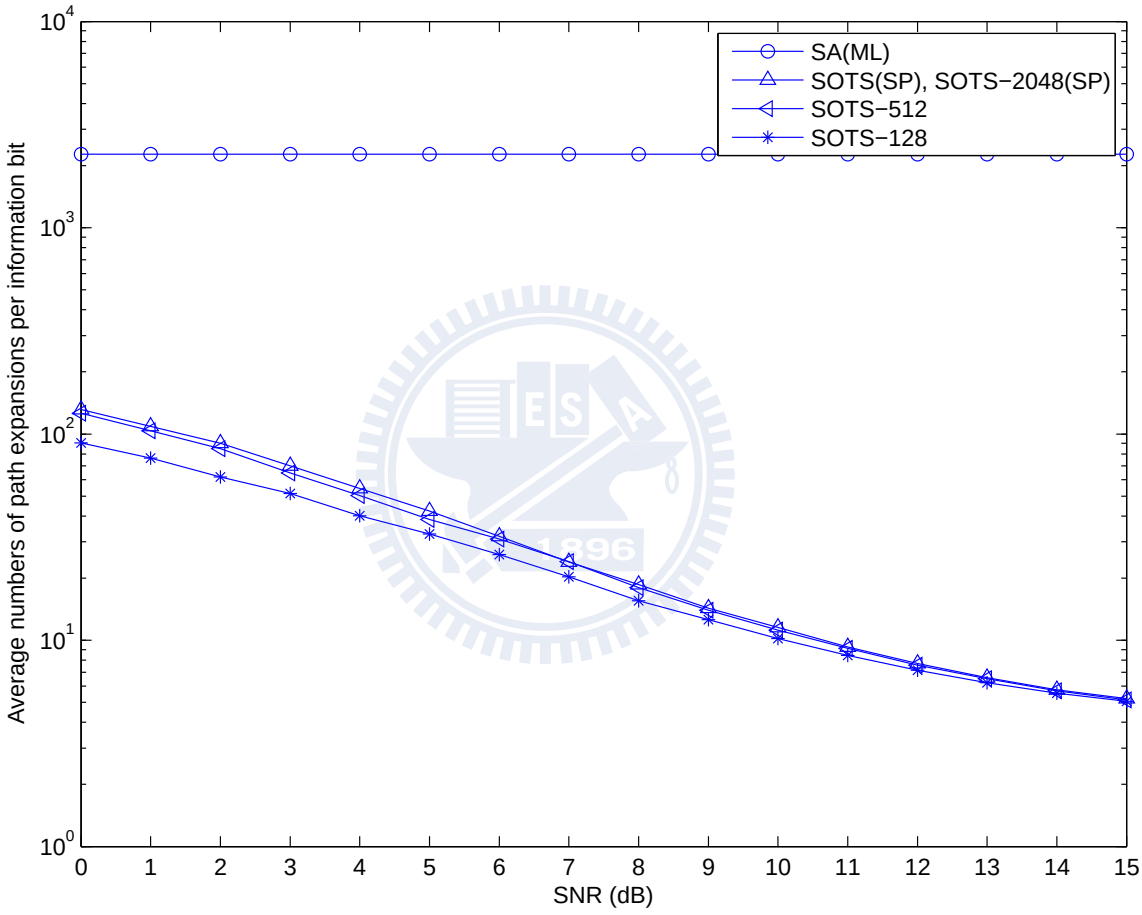

Figure 4.2: The average numbers of path expansions per information bit for the codes examined in Figure 4.1.

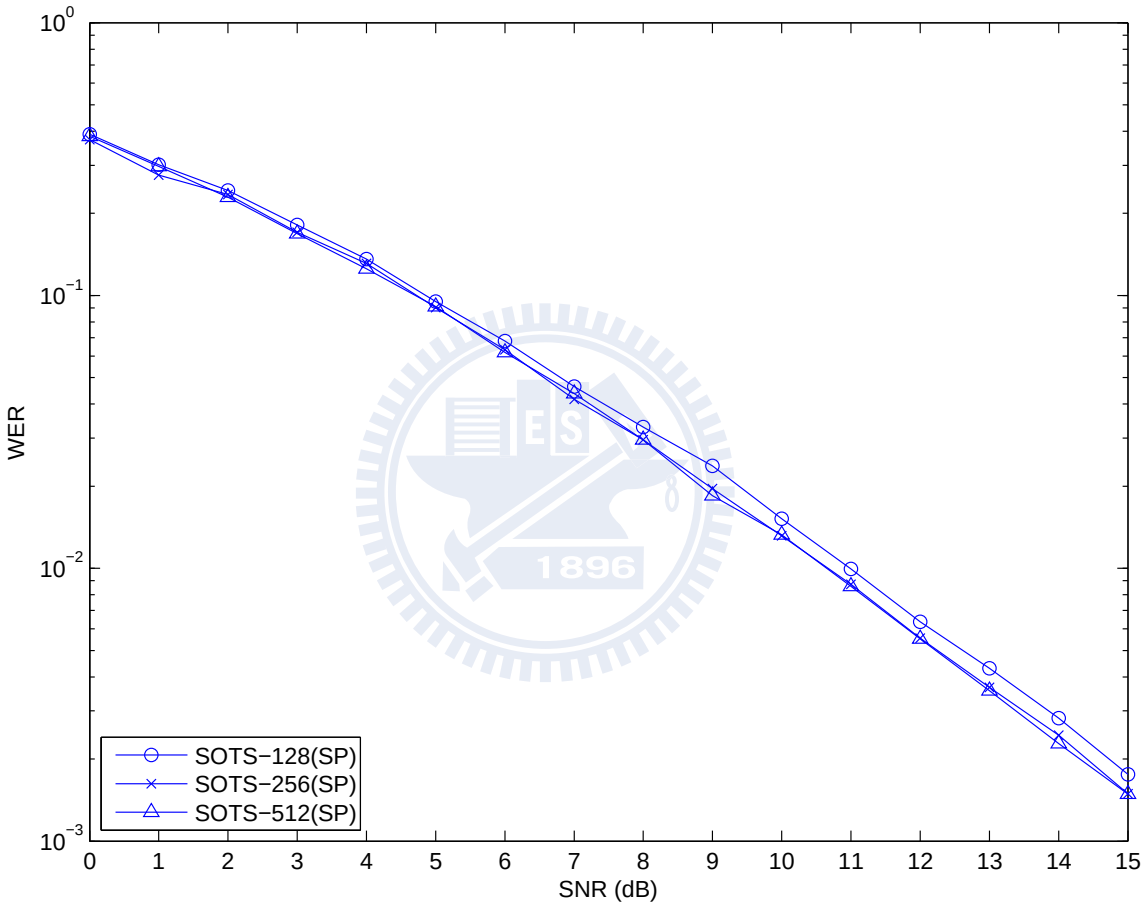

Figure 4.3: Word error rates (WERs) for the self-orthogonal trellis-structure codes of length $N = 18$ with maximal state number 128, 256 and 512, respectively. The decoders used are the sequential decoding with heuristic prediction (SP). The code rate is 1/2.

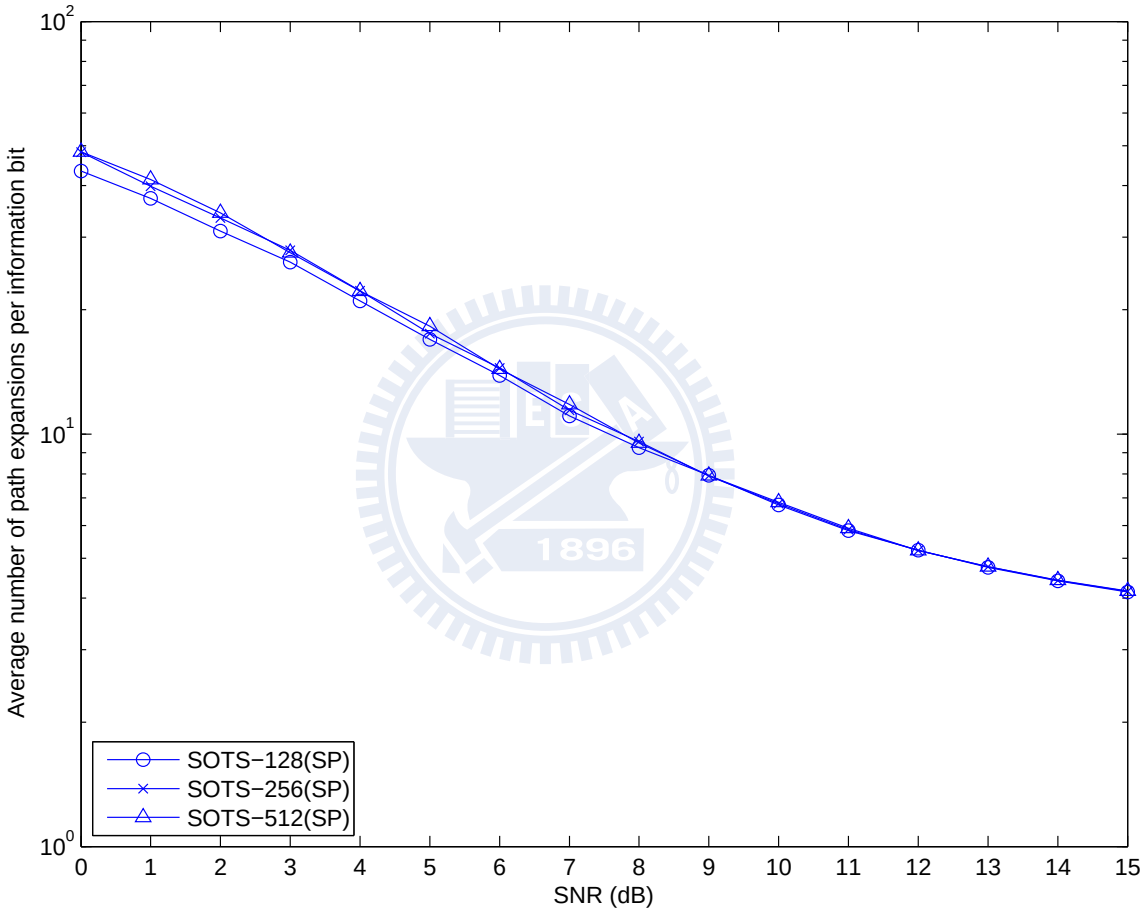

Figure 4.4: The average numbers of path expansions per information bit for the codes examined in Figure 4.3.

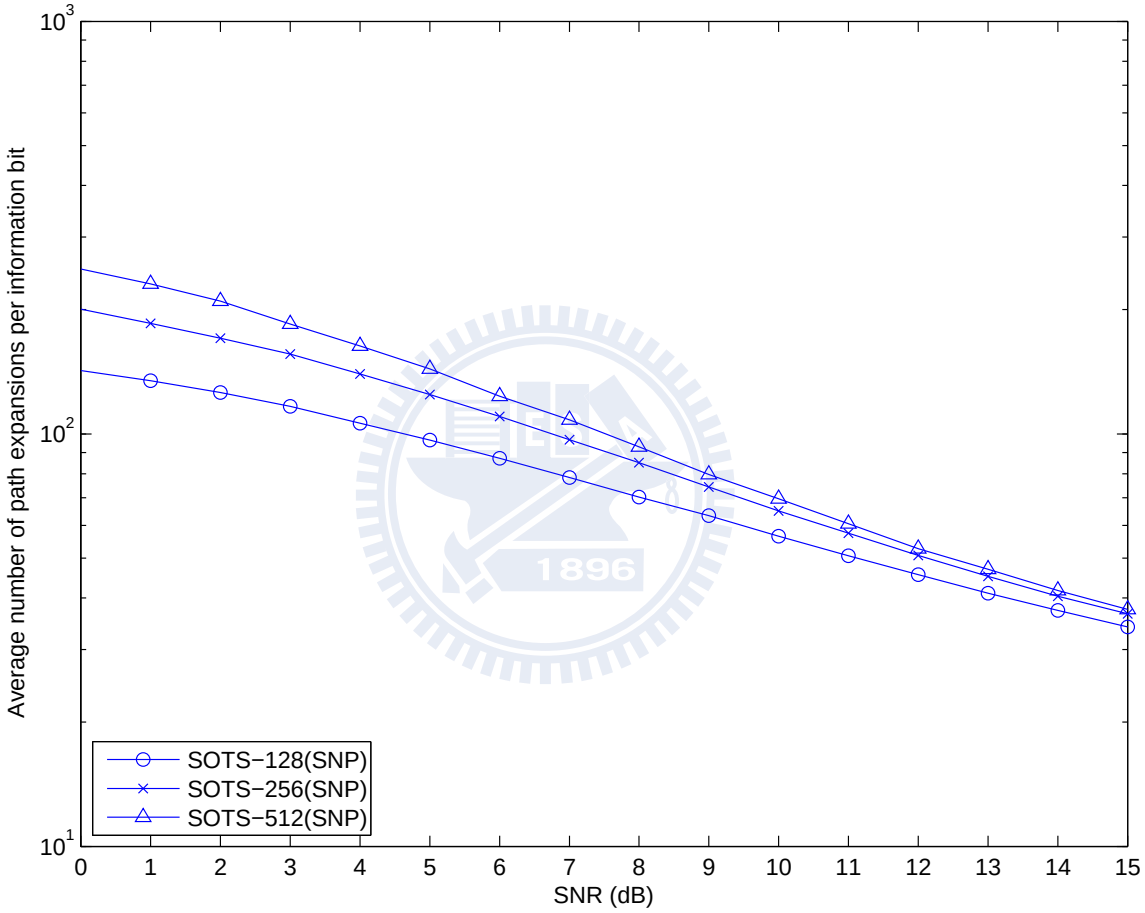

Figure 4.5: The average numbers of path expansions per information bit for the codes examined in Figure 4.3 except the codes are now decoded by the sequential decoding algorithm with no heuristic prediction (SNP).

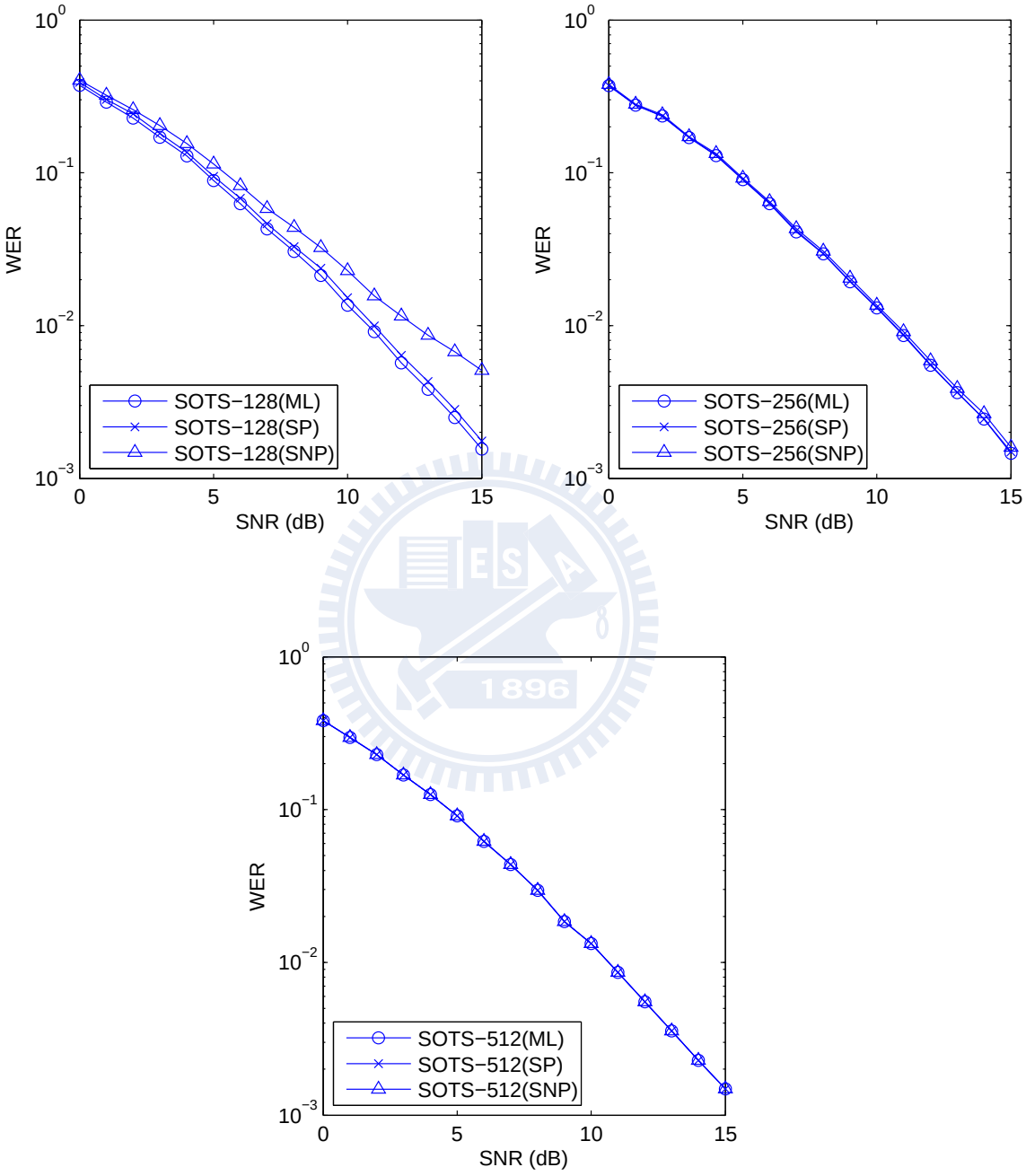

Figure 4.6: Word error rates (WERs) for the proposed codes respectively with $S = 128$, 256 and 512, decoded by different approaches: ML, SP and SNP. The code rate is 1/2 and the codeword length is $N = 18$.

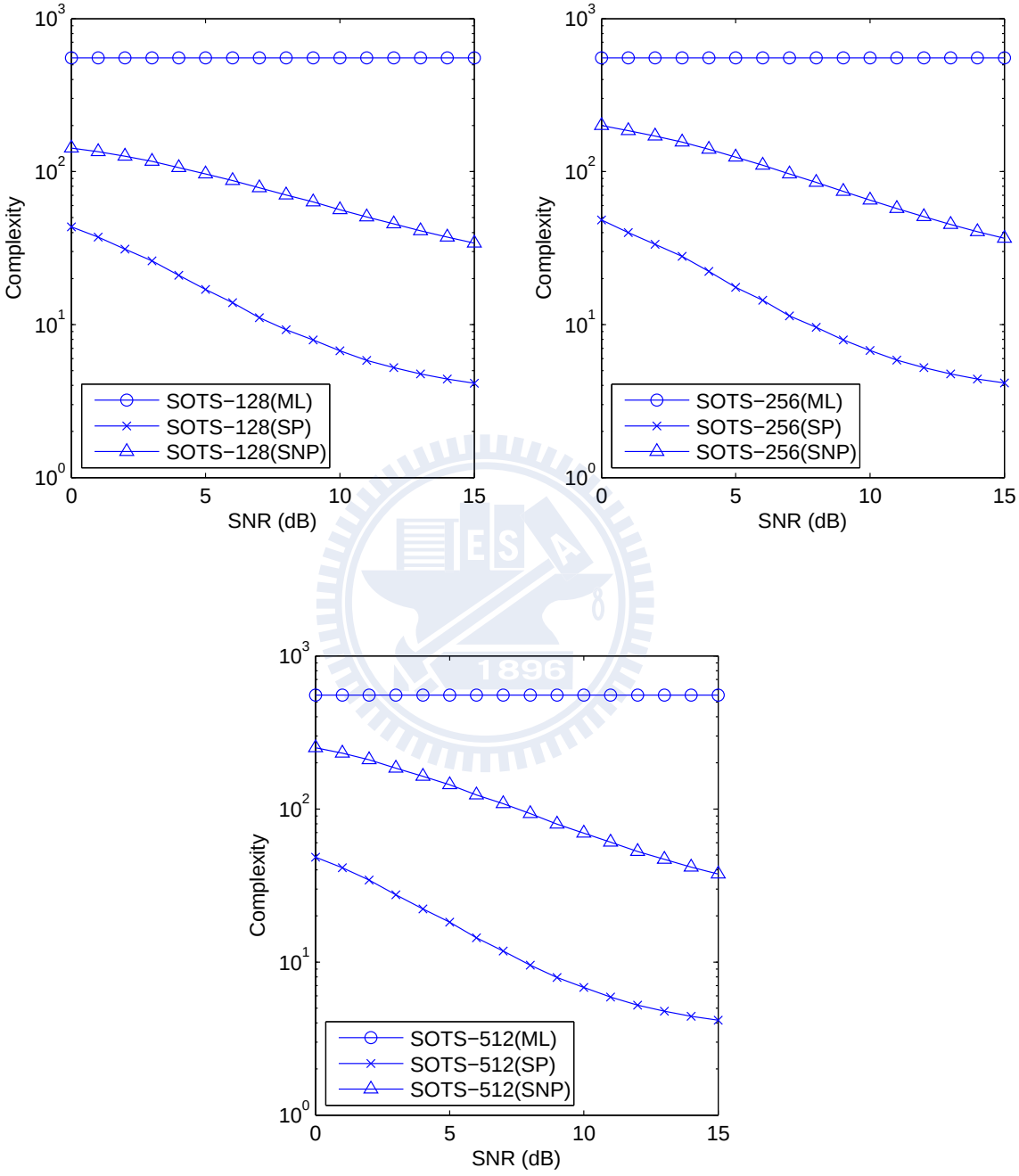

Figure 4.7: The average numbers of path expansions per information bit for the codes examined in Figure 4.6.

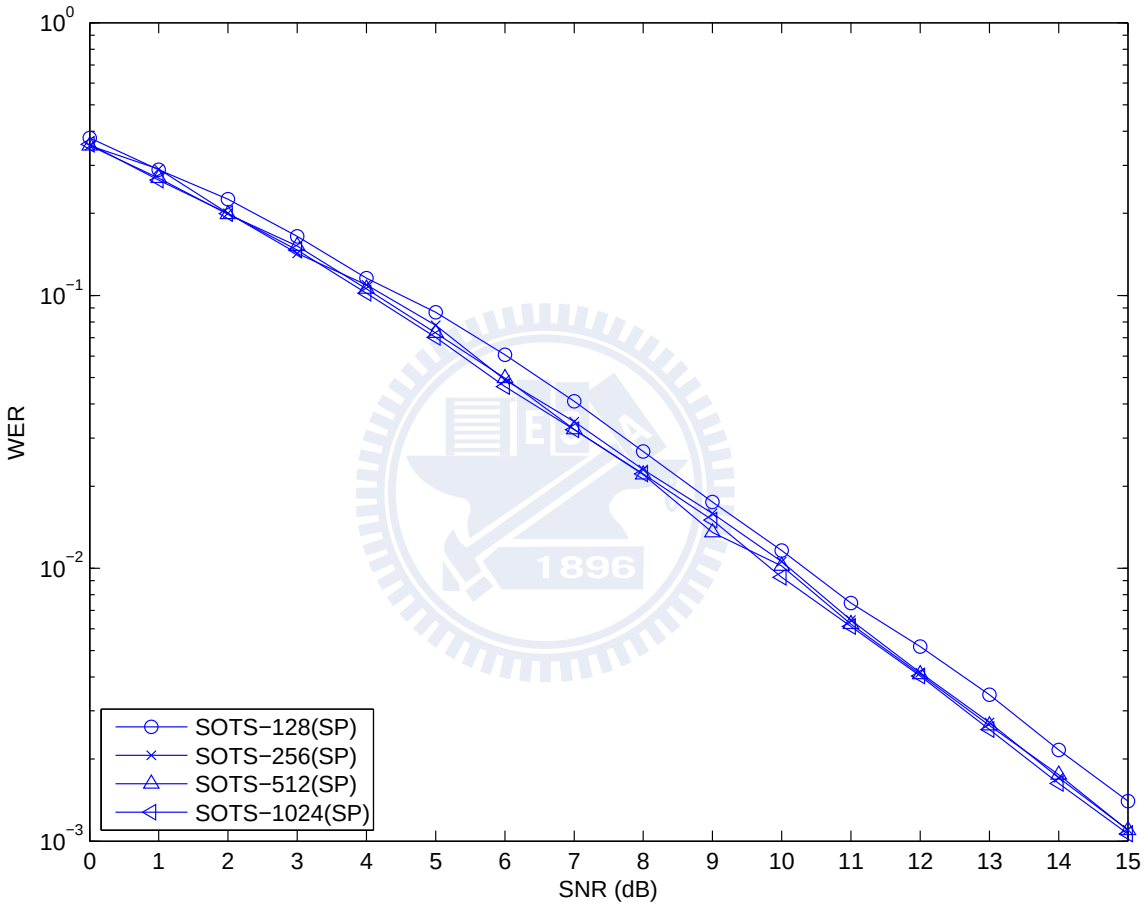

Figure 4.8: Word error rates (WERs) for the self-orthogonal trellis-structure codes of length $N = 20$ with maximal state number 128, 256, 512 and 1024, respectively. The decoders used are the sequential decoding with heuristic prediction (SP). The code rate is 1/2.

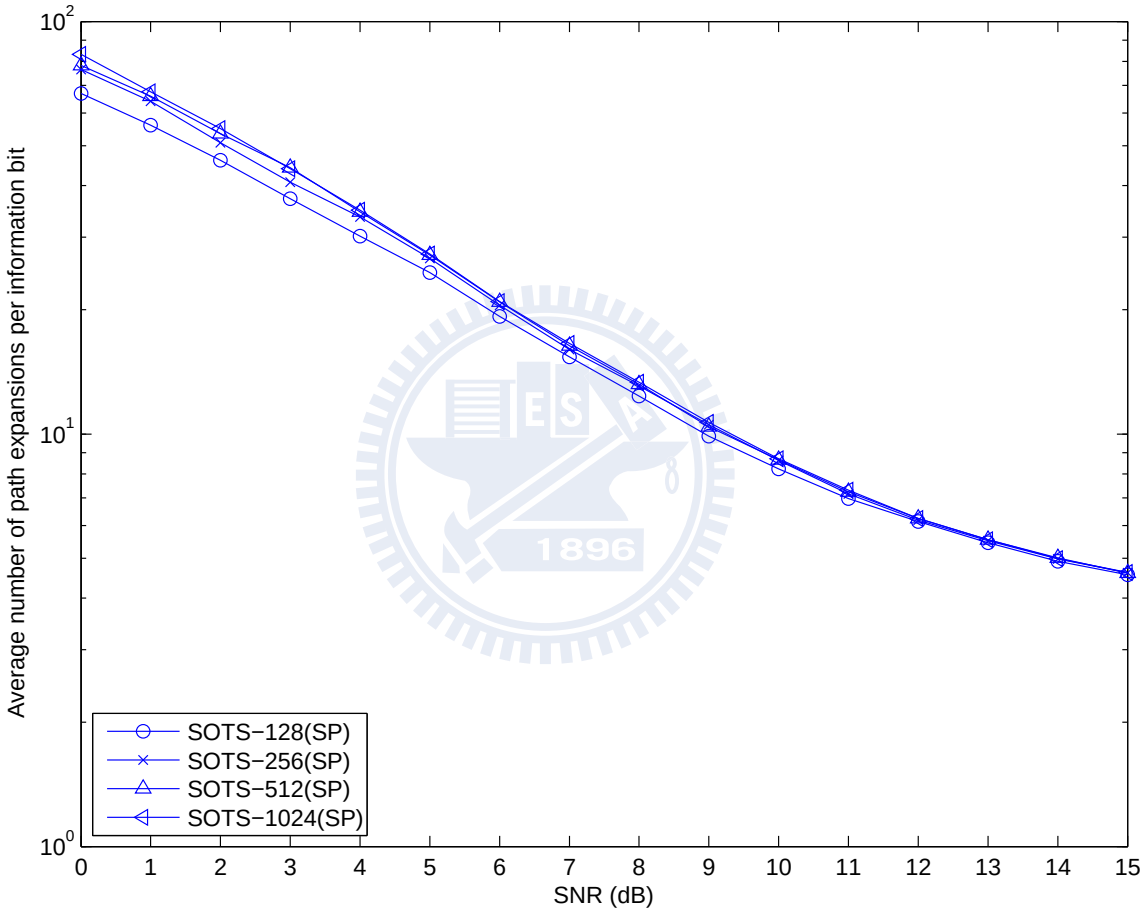

Figure 4.9: The average numbers of path expansions per information bit for the codes examined in Figure 4.8.

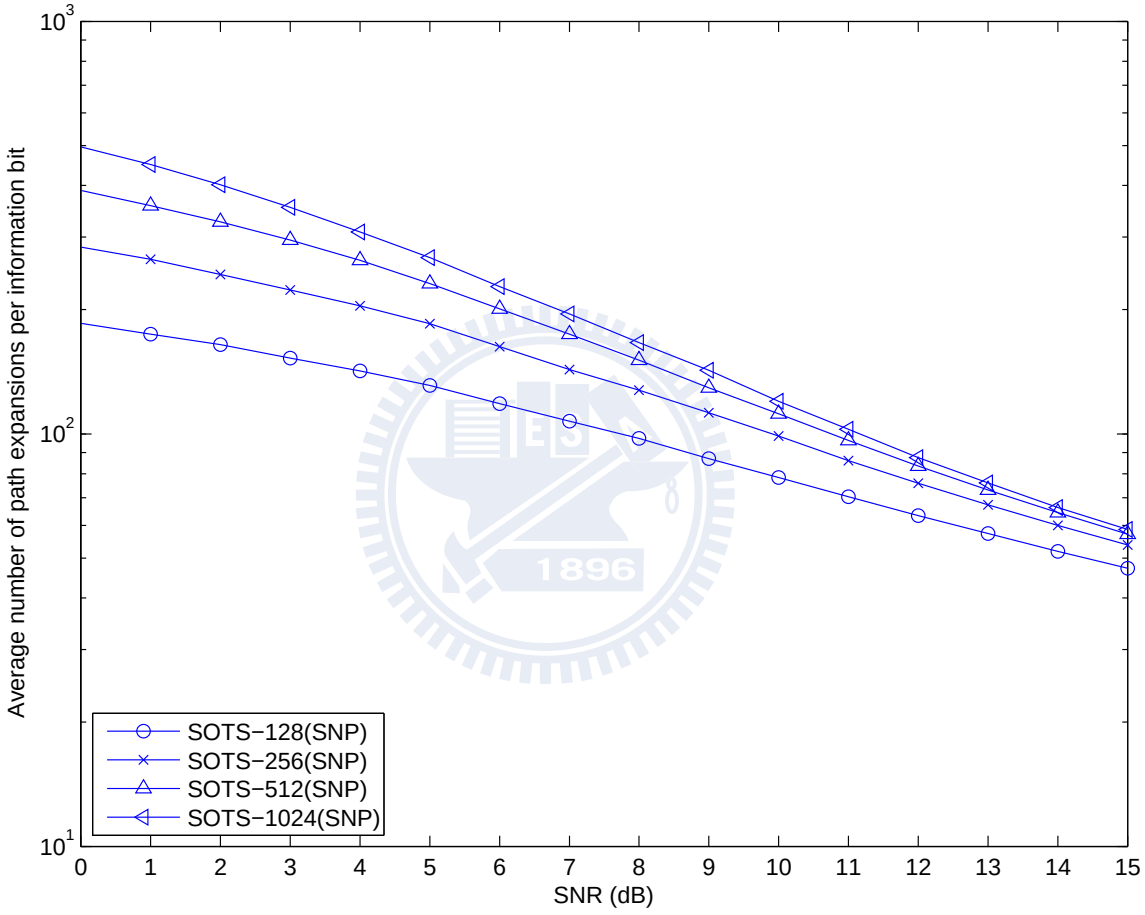

Figure 4.10: The average numbers of path expansions per information bit for the codes examined in Figure 4.8 except the codes are now decoded by the sequential decoding algorithm with no heuristic prediction (SNP).

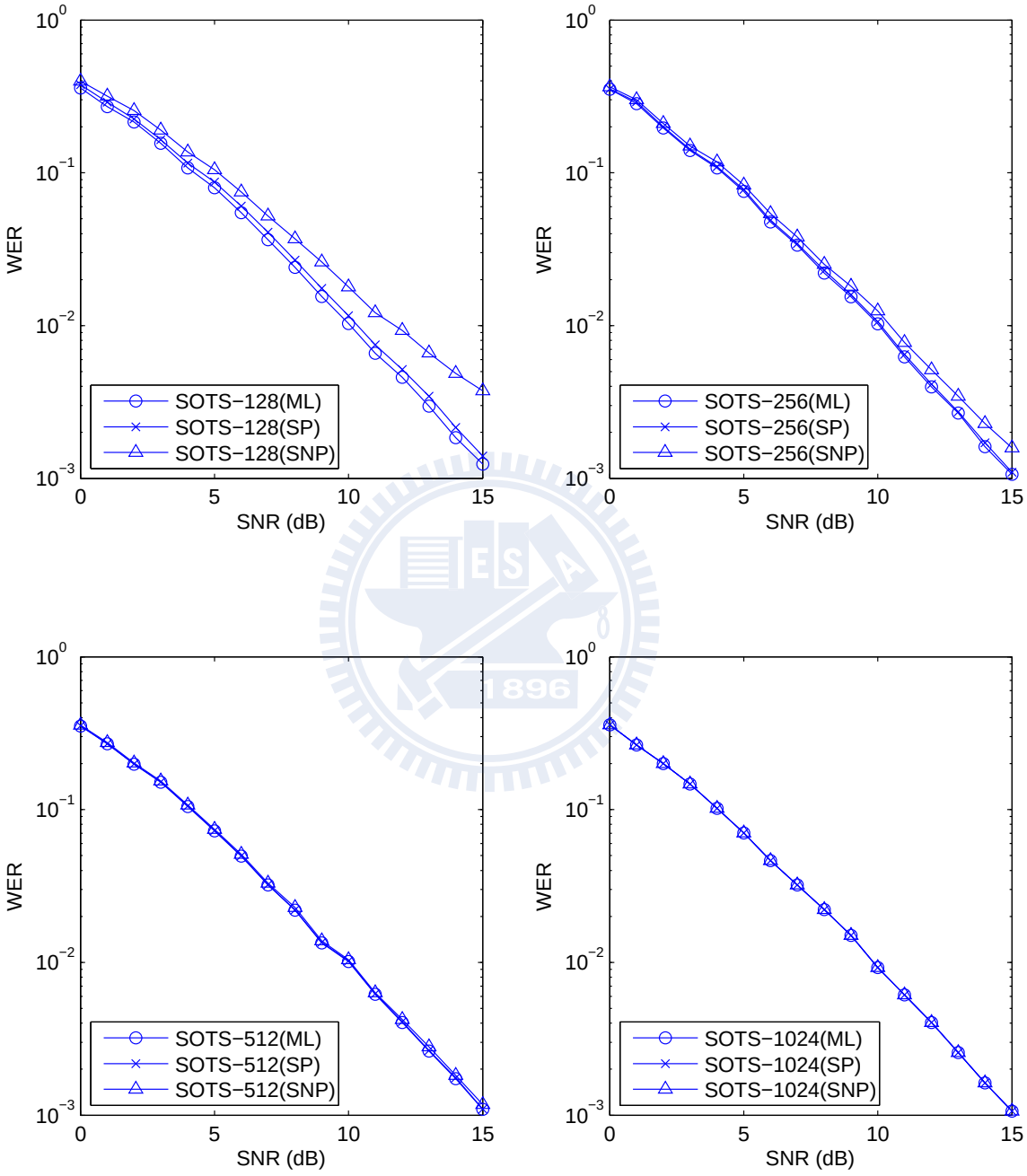

Figure 4.11: Word error rates (WERs) for the proposed codes respectively with $S = 128$, 256, 512 and 1024, decoded by different approaches: ML, SP and SNP. The code rate is 1/2 and the codeword length is $N = 20$.

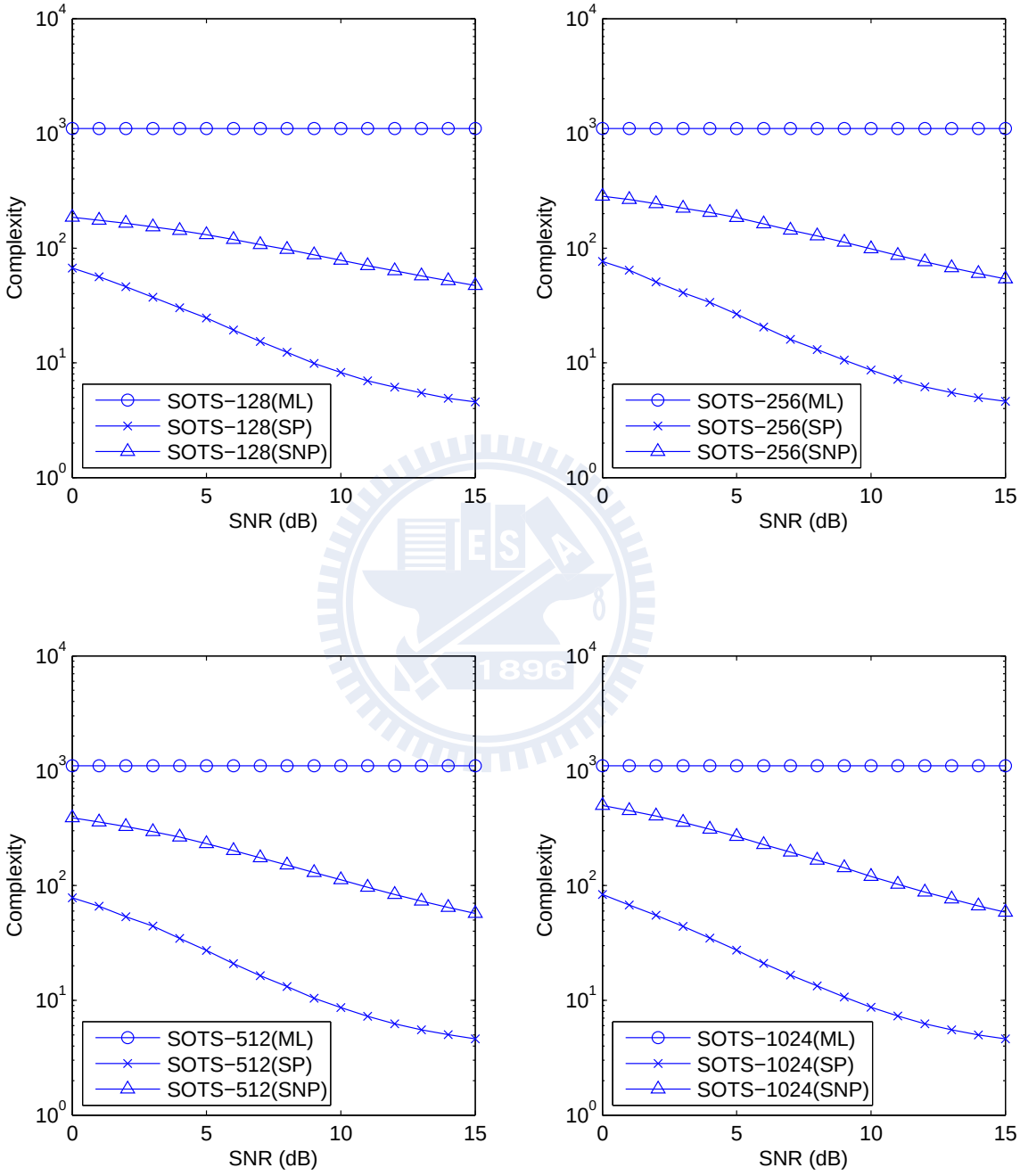

Figure 4.12: The average numbers of path expansions per information bit for the codes examined in Figure 4.11.

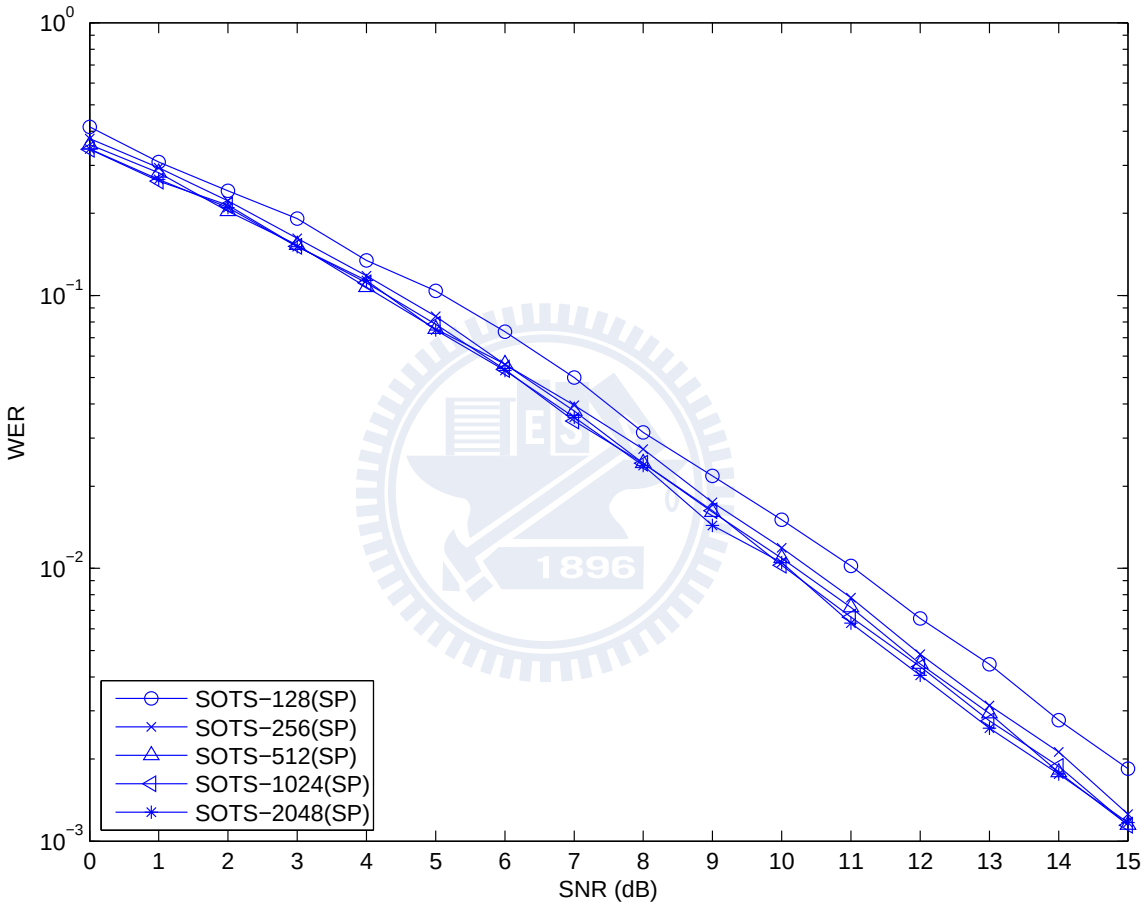

Figure 4.13: Word error rates (WERs) for the self-orthogonal trellis-structure codes of length $N = 22$ with maximal state number 128, 256, 512, 1024 and 2048, respectively. The decoders used are the sequential decoding with heuristic prediction (SP). The code rate is 1/2.

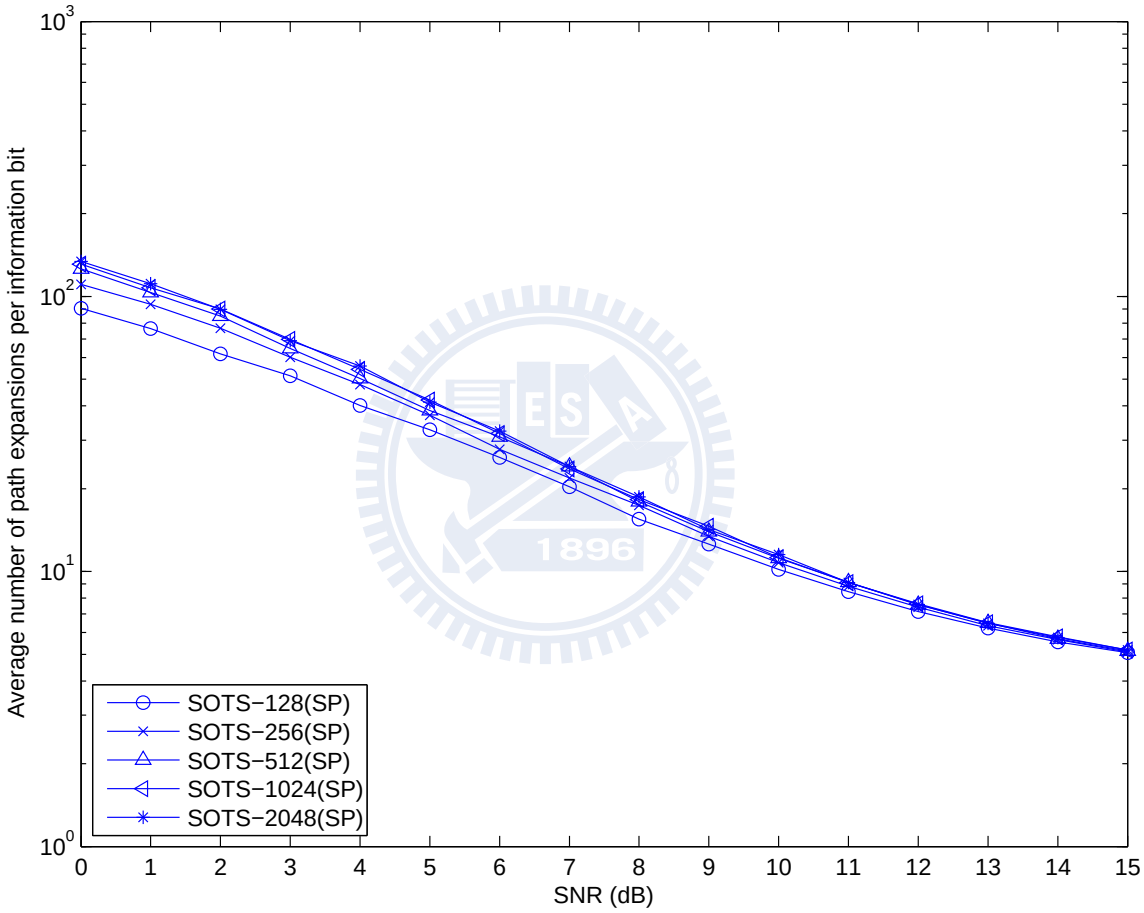

Figure 4.14: The average numbers of path expansions per information bit for the codes examined in Figure 4.13.

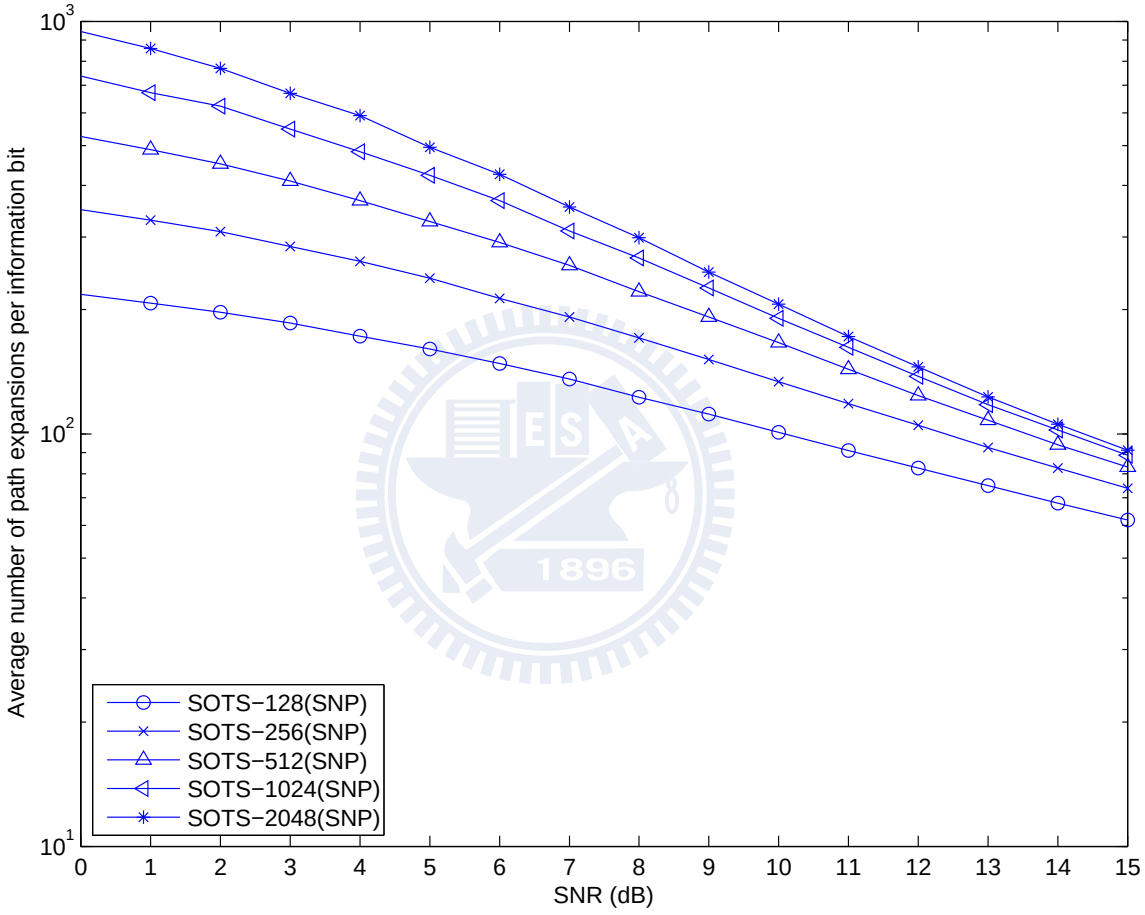

Figure 4.15: The average numbers of path expansions per information bit for the codes examined in Figure 4.13 except the codes are now decoded by the sequential decoding algorithm with no heuristic prediction (SNP).

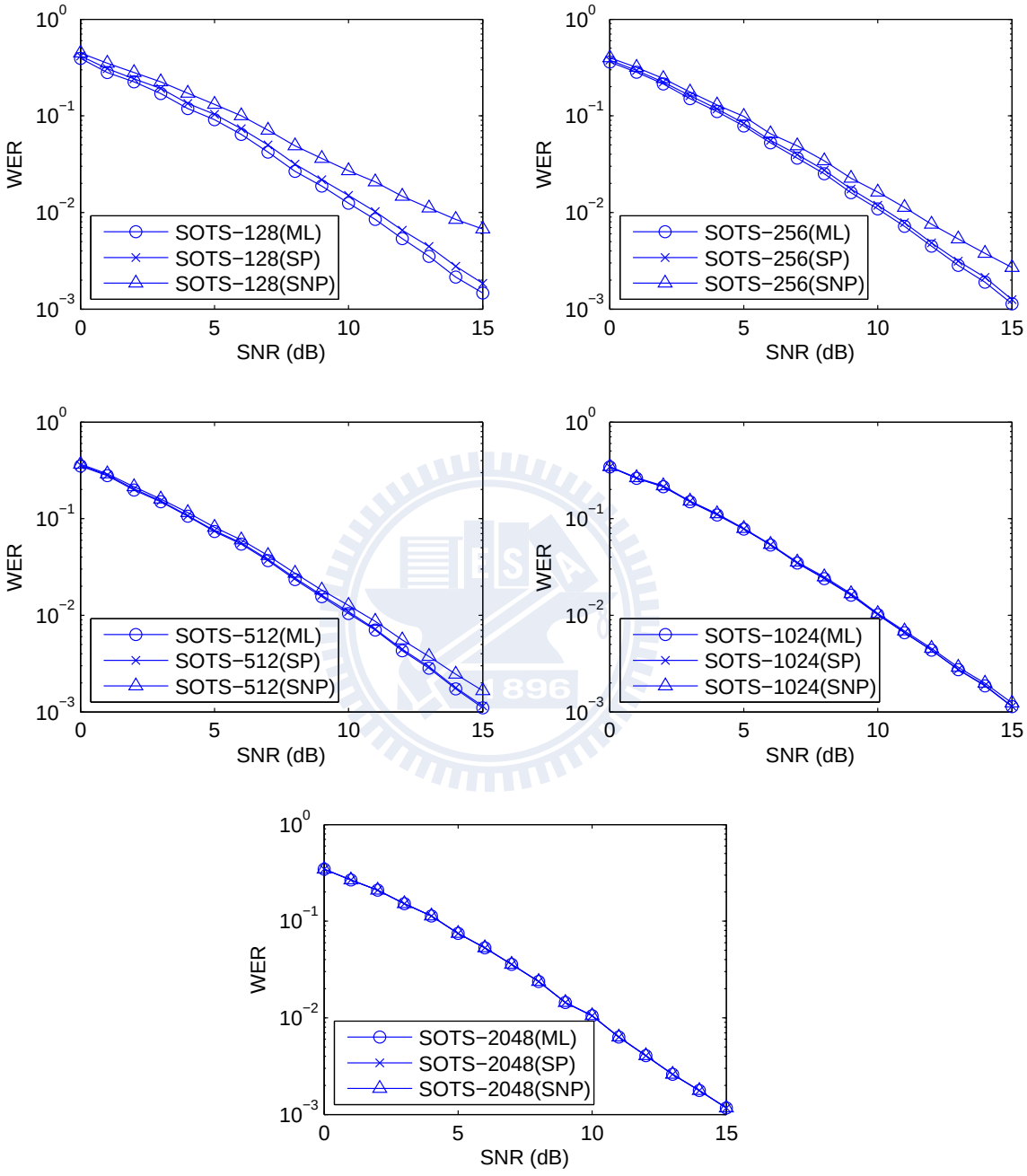

Figure 4.16: Word error rates (WERs) for the proposed codes respectively with $S = 128$, 256, 512, 1024 and 2048, decoded by different approaches: ML, SP and SNP. The code rate is $1/2$ and the codeword length is $N = 22$.

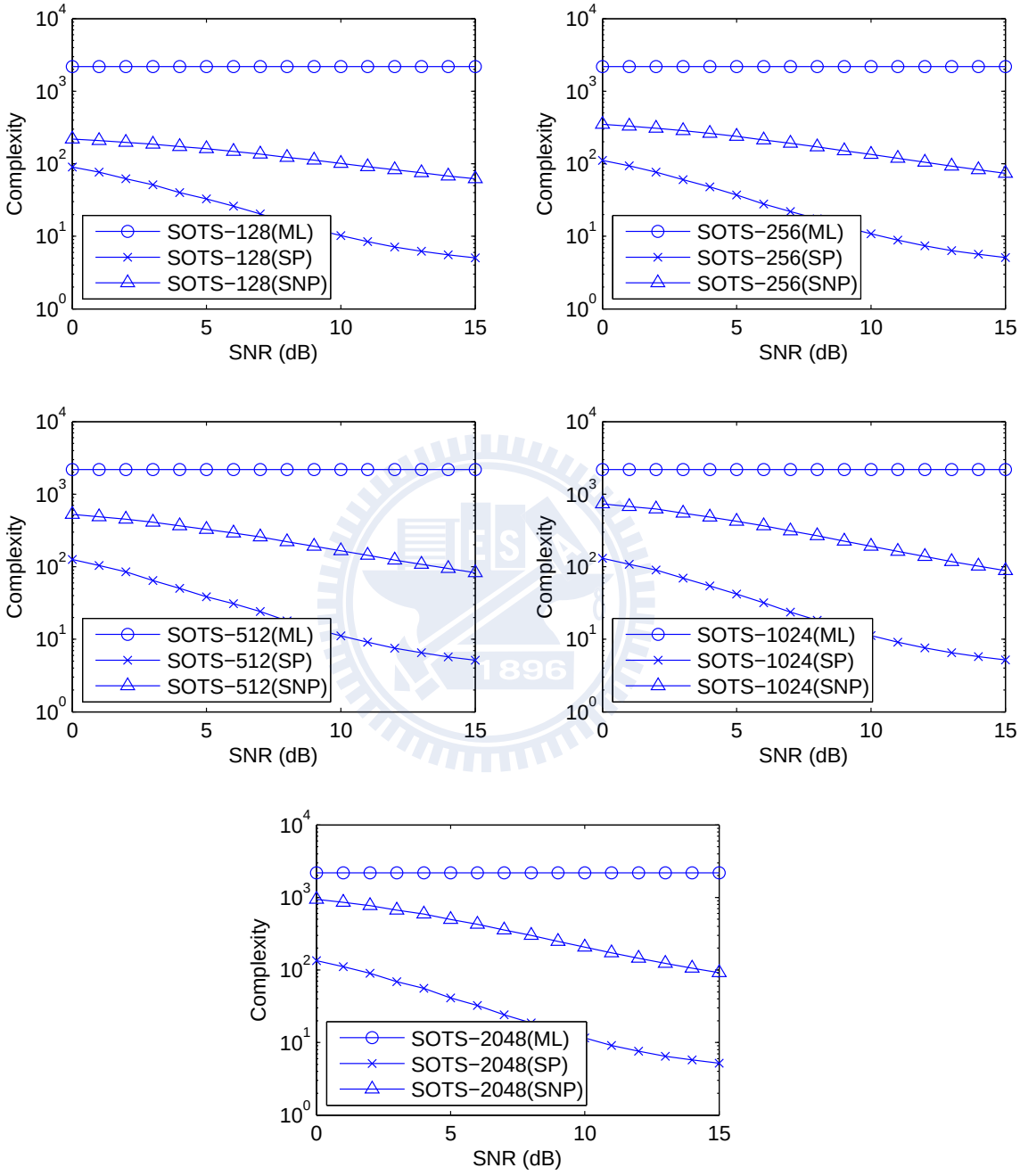

Figure 4.17: The average numbers of path expansions per information bit for the codes examined in Figure 4.16.

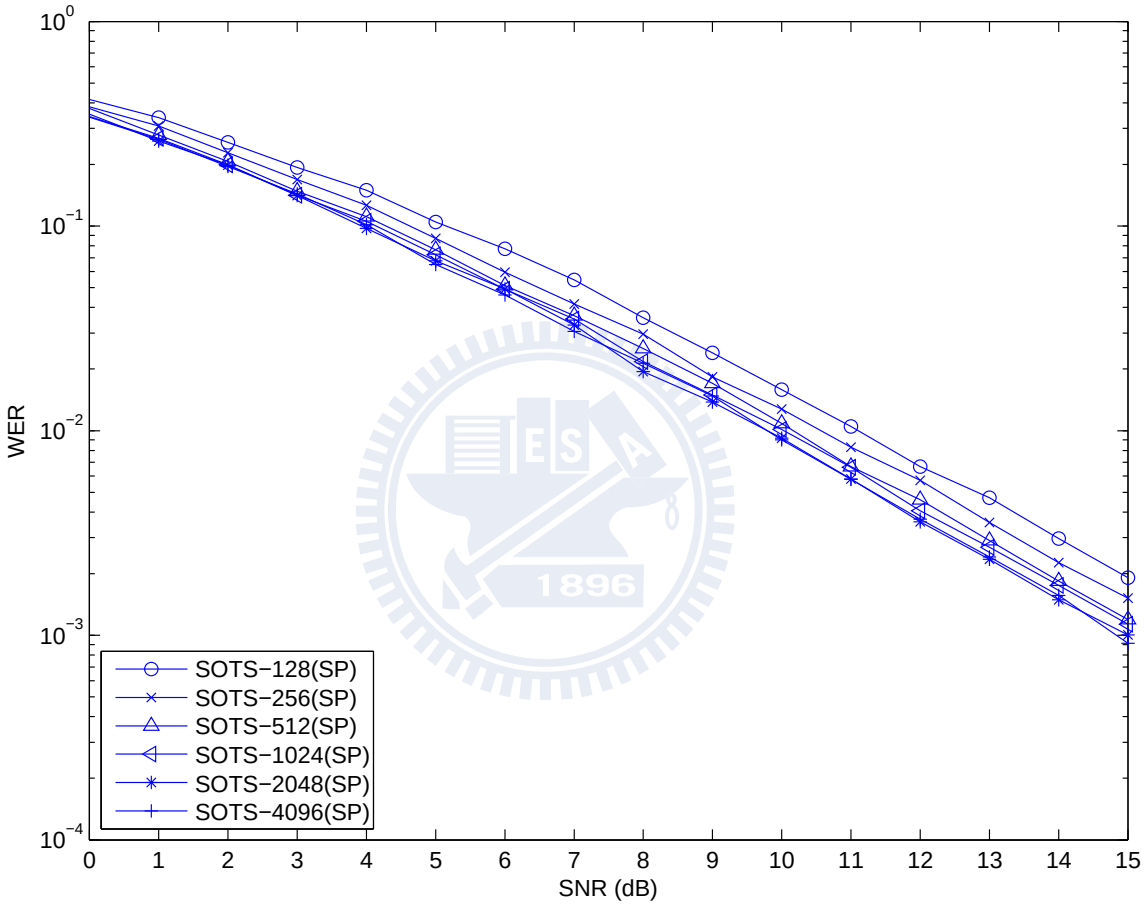

Figure 4.18: Word error rates (WERs) for the self-orthogonal trellis-structure codes of length $N = 24$ with maximal state number 128, 256, 512, 1024, 2048 and 4096, respectively. The decoders used are the sequential decoding with heuristic prediction (SP). The code rate is 1/2.

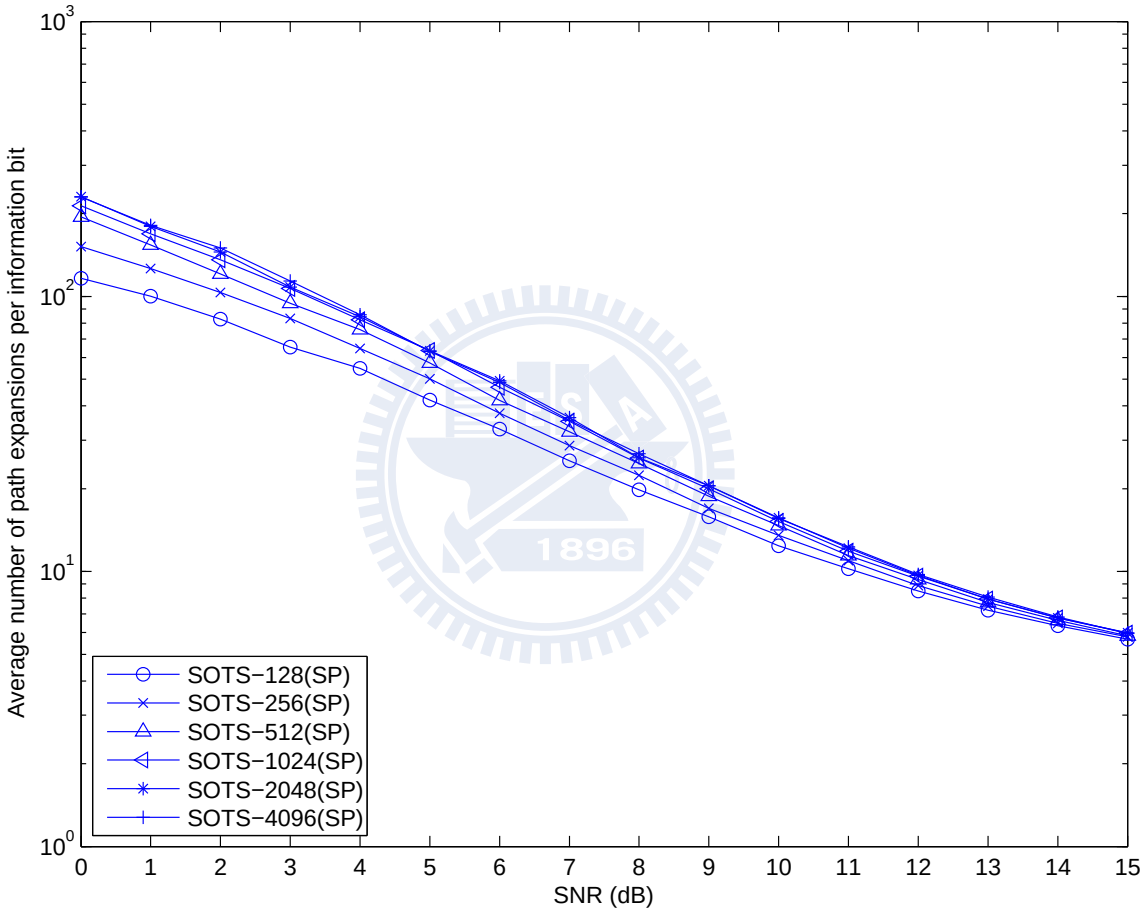

Figure 4.19: The average numbers of path expansions per information bit for the codes examined in Figure 4.18.

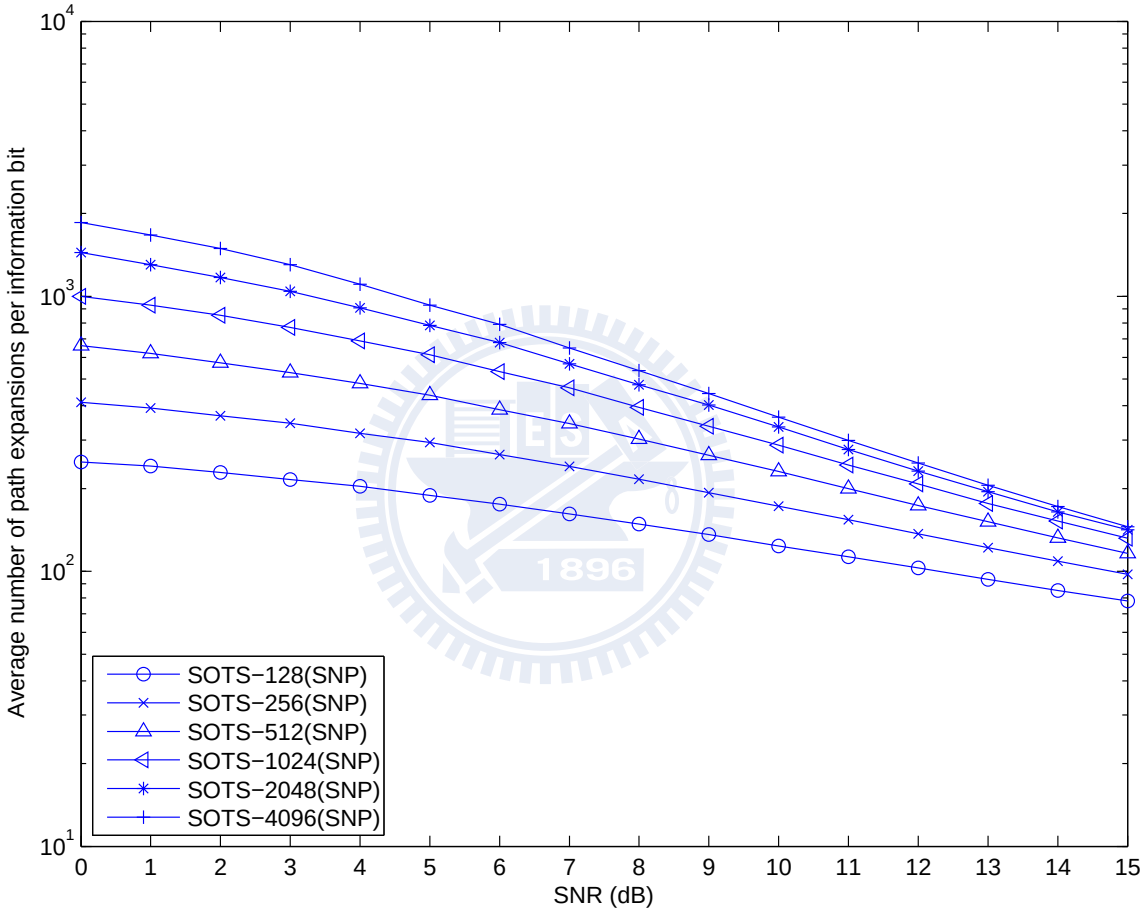

Figure 4.20: The average numbers of path expansions per information bit for the codes examined in Figure 4.18 except the codes are now decoded by the sequential decoding algorithm with no heuristic prediction (SNP).

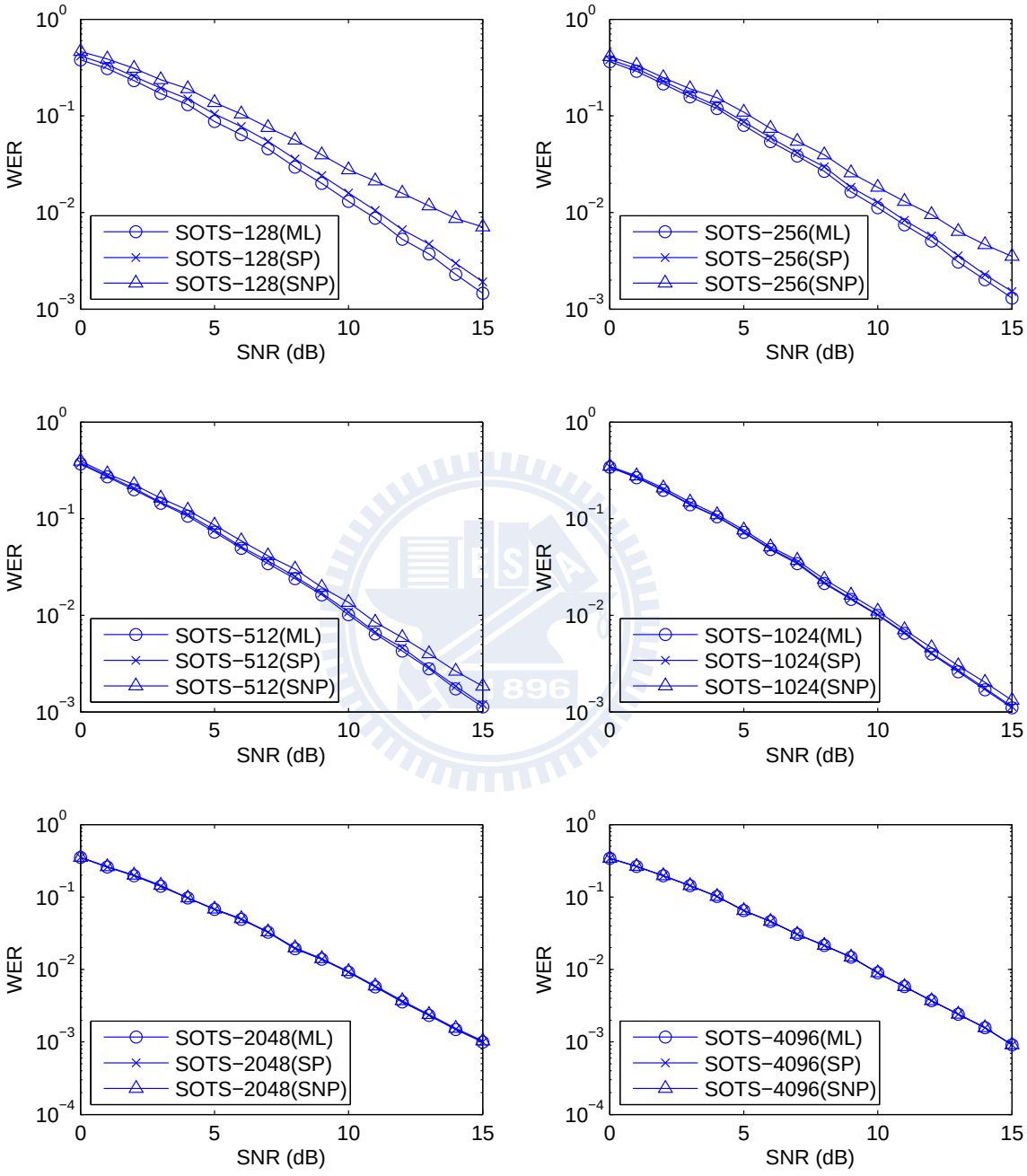

Figure 4.21: Word error rates (WERs) for the proposed codes respectively with $S = 128$, 256, 512, 1024, 2048 and 4096, decoded by different approaches: ML, SP and SNP. The code rate is $1/2$ and the codeword length is $N = 24$.

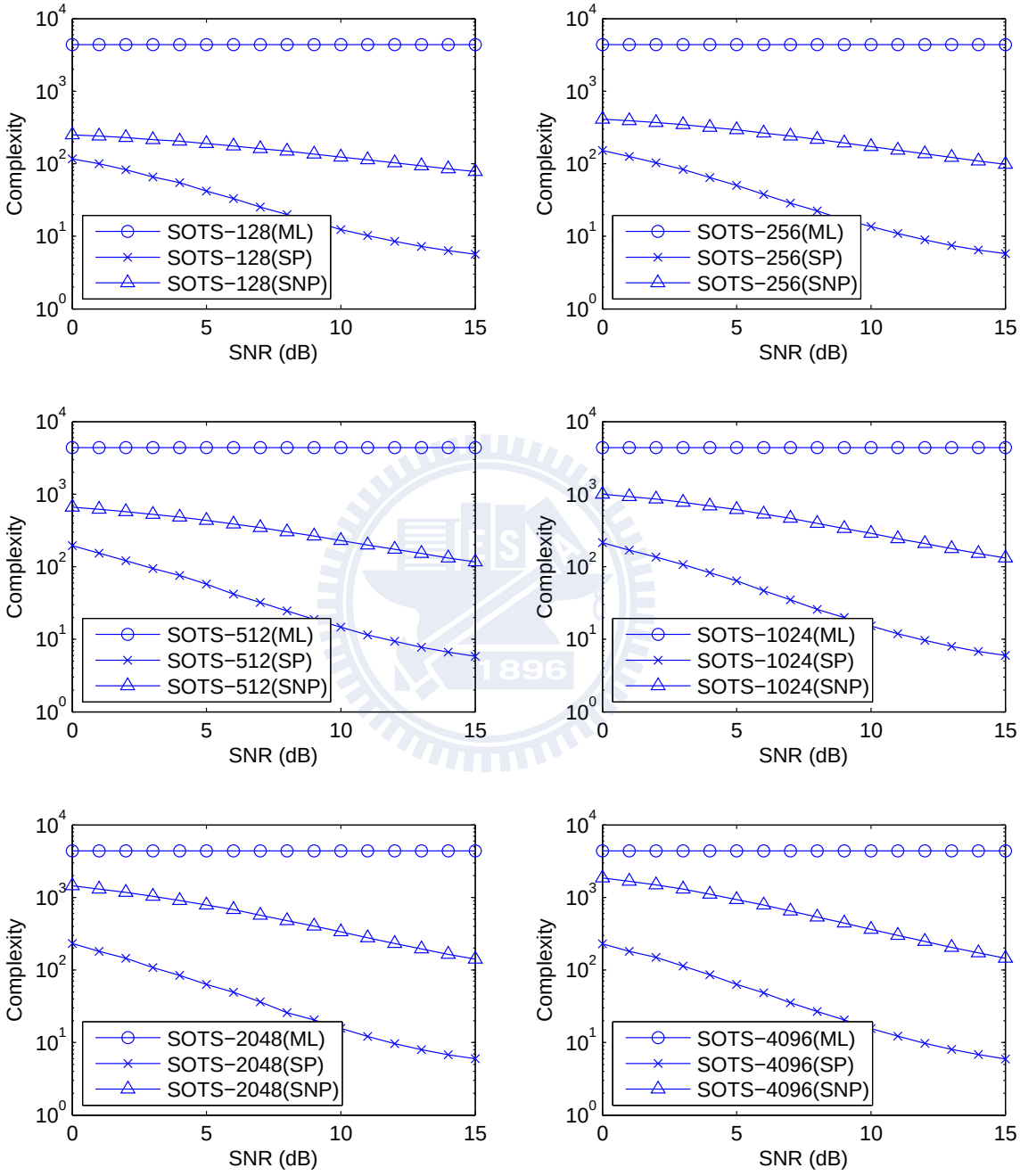

Figure 4.22: The average numbers of path expansions per information bit for the codes examined in Figure 4.21.

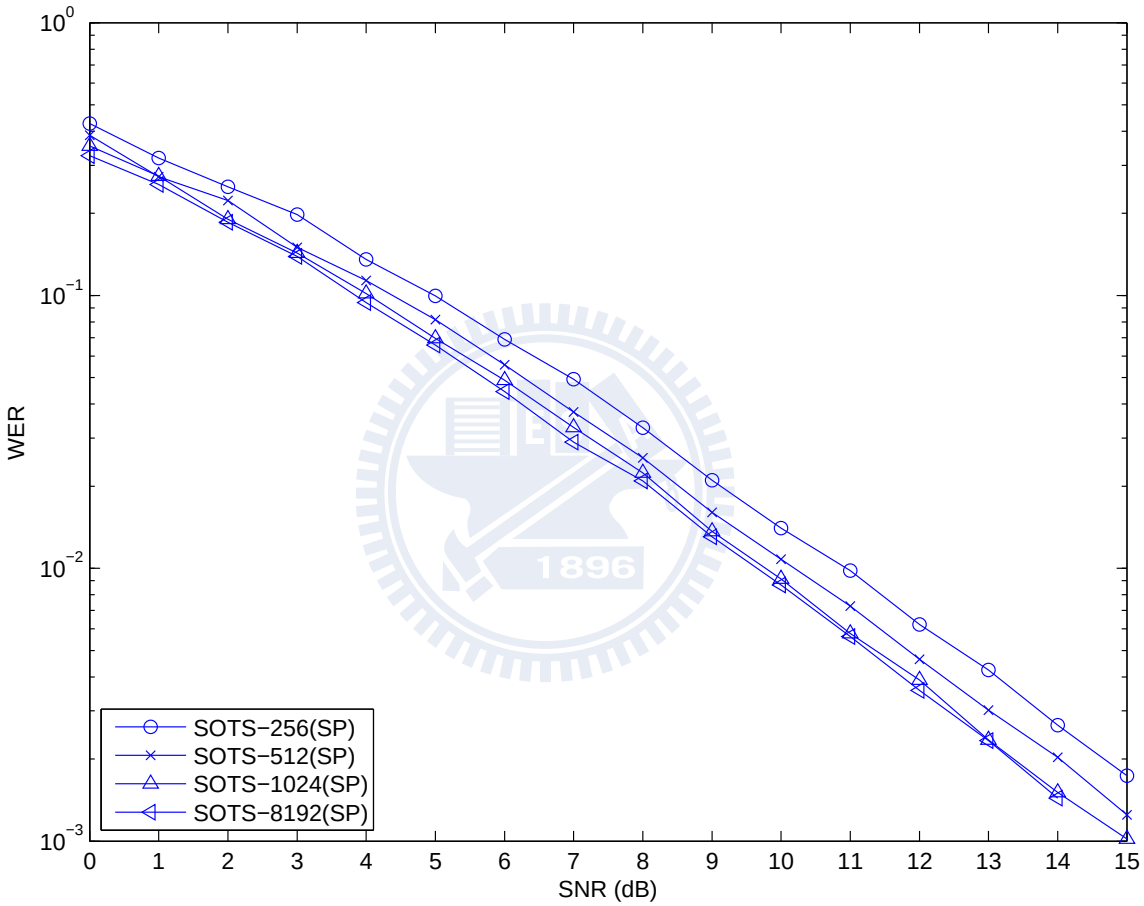

Figure 4.23: Word error rates (WERs) for the self-orthogonal trellis-structure codes of length $N = 26$ with maximal state number 256, 1024 and 8192, respectively. The decoders used are the sequential decoding with heuristic prediction (SP). The code rate is 1/2.

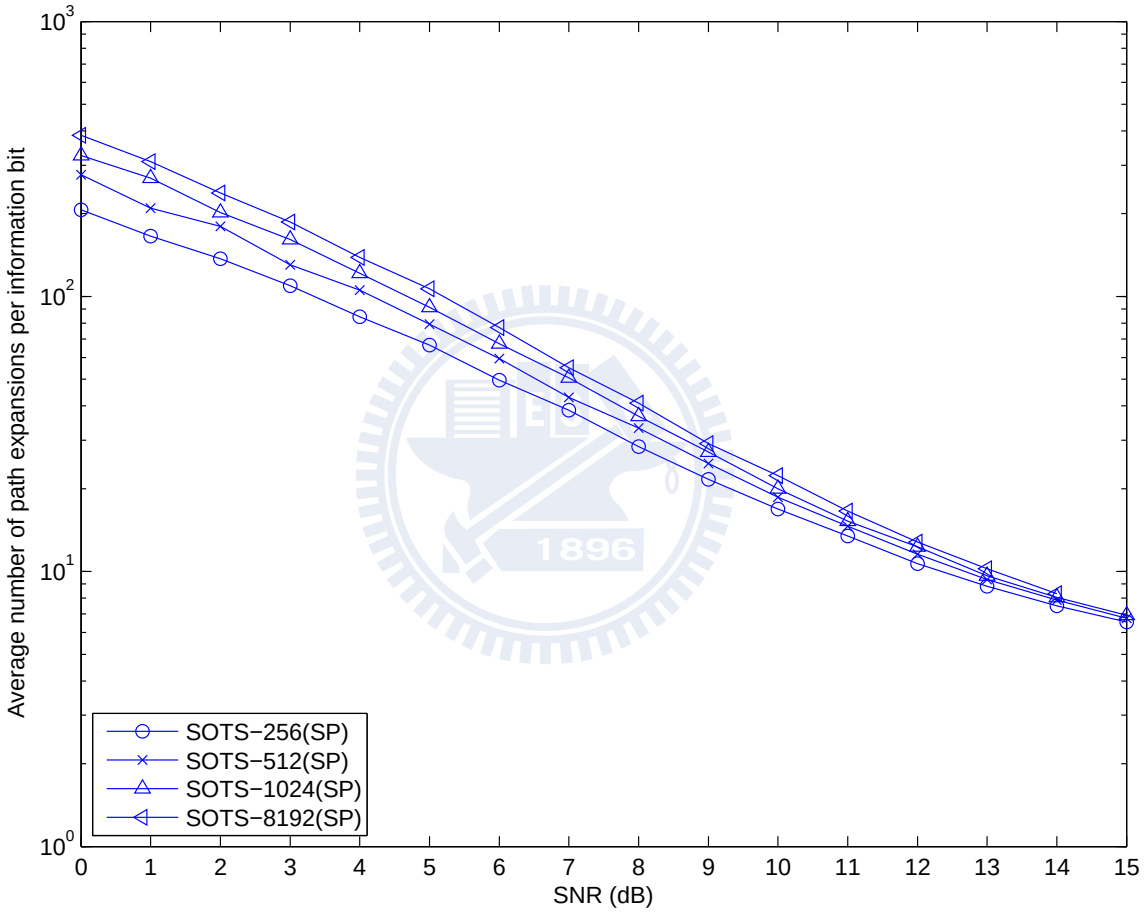

Figure 4.24: The average numbers of path expansions per information bit for the coded examined in Figure 4.23.

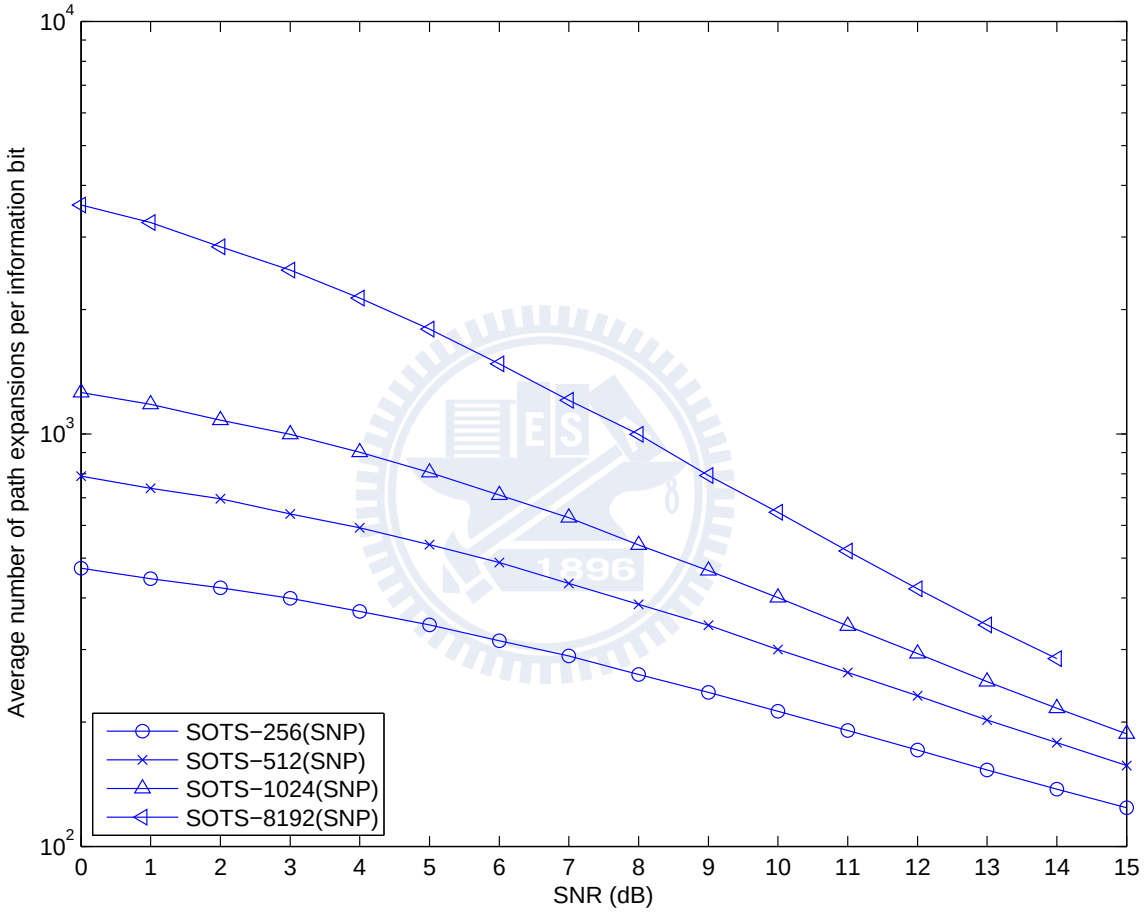

Figure 4.25: The average numbers of path expansions per information bit for the codes examined in Figure 4.23 except the codes are now decoded by the sequential decoding algorithm with no heuristic prediction (SNP).

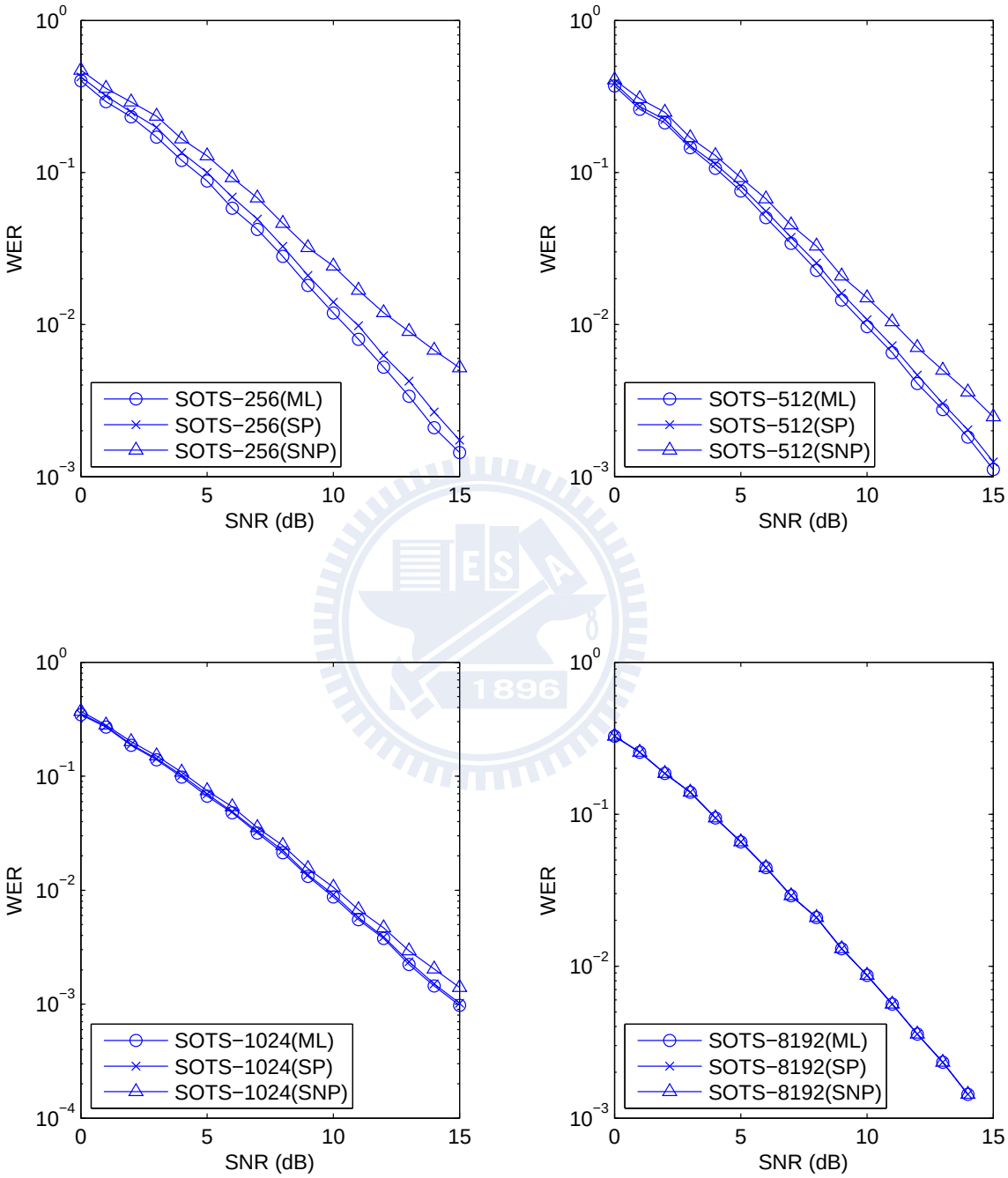

Figure 4.26: Word error rates (WERs) for the proposed codes respectively with $S = 256$, 512, 1024 and 8192, decoded by different approaches: ML, SP and SNP. The code rate is $1/2$ and the codeword length is $N = 26$.

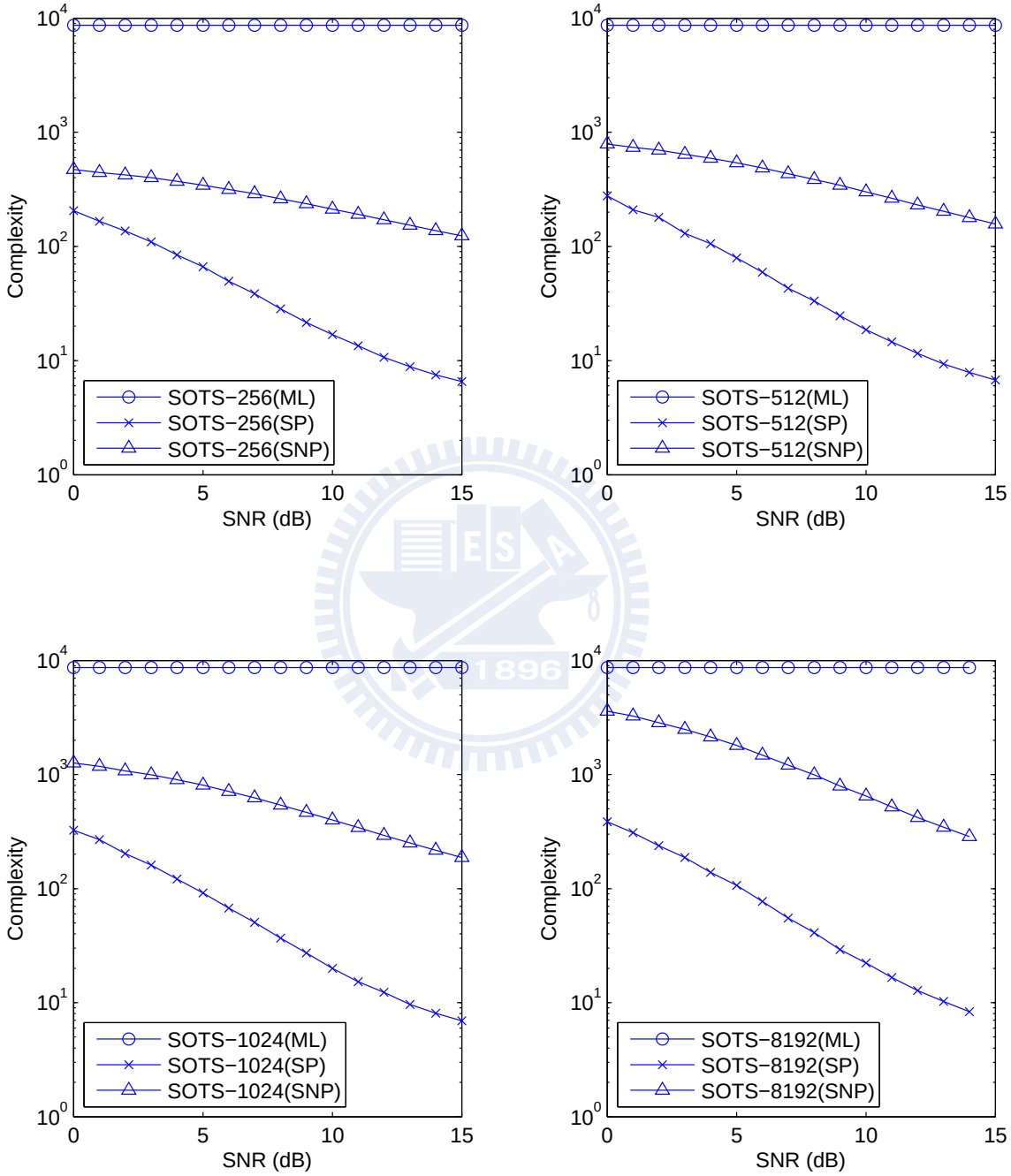

Figure 4.27: The average numbers of path expansions per information bit for the codes examined in Figure 4.26.

4.3 Discussions on Simulation Results

It can be observed from Figure 4.1 that the trellis-structure codes (SOTS-512) with maximum state number 512 has 0.3–0.4 dB performance degradation when being compared with the tree-structure codes (equivalently, the trellis-structure codes with maximum state 2048). The performance degradation will be increased up to 0.5 dB if the reference becomes the best computer-searched codes (SA) by simulated annealing in [1]. As anticipated, the code with maximum state number 128 has the worst performance, but it requires the least complexity.

Most of the experimental results behave similarly even if we further increase the codeword length. As an example, Figure 4.18 shows that increasing the codeword length from 22 to 24, the SOTS-128(SP) remains 1 dB away from the original tree-structure code (SOTS-4096(SP)). Yet, the performance degradations of all the proposed codes with maximum state number more than 512 are still satisfactorily maintained within 0.5 dB. It is worth noting that the performance degradation of the trellis-structure codes tends to be slightly larger at lower SNR. When the SNP decoder is used instead, a larger performance degradation will be resulted.

Regarding the decoding complexity, we can observe from Figures 4.7, 4.12, 4.17, 4.22 and 4.27 that the ML (exhaustive) decoder has much higher decoding complexity than both the SP and SNP decoders. Comparison between the SP and SNP decoders give that the former can yield around ten-fold saving in decoding complexity from the latter. As far as their decoding word error rates are concerned, Figures 4.6, 4.11, 4.16, 4.21 and 4.26 show that the ML decoder is better than the SP decoder, while the SP decoder is better than the SNP decoder. Besides, the SP decoder can achieve the ML performance when the maximum state number is not much below $2^{N/2}$. Based on these simulations, we would suggest to use the SP decoder rather than the SNP decoder if the decoding process is allowed to begin after

the reception of the entire received vector.

In closing, we conclude that the proposed codes with smaller maximum state number has less decoding complexity but worse performance. However, the performance degradation can still be controlled within 0.5–1 dB. Thus, it is still advantageous to adopt the trellis-structure codes in certain applications where the reduction of decoding complexity is more critical in system design than the performance.

Chapter 5

Conclusion Remarks and Future Work

In this thesis, we proposed an algorithmic procedure to fit the self-orthogonal tree-structure (SOTS) codes in [2] into a trellis by selectively merging tree nodes. Through the algorithmic node merging procedure, we can control the trade-off between memory consumption, decoding complexity and performance. Simulations showed that the performance degradation can be controlled within 0.5–1 dB even for a smaller maximum state number such as 128. It is then concluded that the trellis-structure codes can moderately reduce the decoding complexity without sacrificing much in performance.

This thesis mainly put efforts in the encoder design. From the system aspect, a corresponding decoder should also be an essential part if our codes are considered to be put into use. The SP and SNP decoders are straightforward extended and adopted from [2] and are originally designed for the tree-structure codes. An interesting future work would be to develop a proper decoder that is specifically suitable for the algorithmically constructed trellis codes.

Our original goal in this research is to design a self-orthogonal trellis-structure code so that the large decoding complexity of the self-orthogonal tree-structure codes can be reduced. With a trellis structure, such a design becomes much more complicated and therefore we

turned to the provision of an algorithm to "compress" an available tree-structure code into a corresponding trellis-structure one. The complexity of such a compression, as anticipated, grows exponentially when the codeword length increases. How to find a good theory for self-orthogonal trellis-structure codes so that they can be constructed directly (without using a tree-structure code) is another subject for future research.

Bibliography

[1] M. Skoglund, J. Giese and S. Parkvall, "Code design for combined channel estimation and error protection," *IEEE Trans. Inform. Theory*, vol. 48, no. 5, pp. 1162-1171, May 2002.

[2] C.-L. Wu, P.-N. Chen, Yunghsiang S. Han and M.-Hsin Kuo, "Maximum-likelihood priority-first search decodable codes for combined channel estimation and error protection," *IEEE Trans. Inform. Theory*, vol. 55, no. 9, pp. 4191-4203, Sept. 2009.

[3] J. Proakis, *Digital Communications*, McGraw-Hill, fifth edition, 2007.

[4] O. Coskun and K. M. Chugg, "Combined coding and training for unknown ISI channels," *IEEE Trans. Commun.*, vol. 53, no. 8, pp. 1310-1322, Aug. 2005.

[5] J. Giese and M. Skoglund, "Space-time code design for unknown frequency-selective channels," *Proc. IEEE. Int. Conf. Acoust., Speech, Signal Process.*, Orlando, FL, USA, May 2002.

[6] N. Seshadri, "Joint data and channel estimation using blind trellis search techniques," *IEEE Trans. Commun.* , vol. 42, no. 2/3/4, pp. 1000-1011, February/March/April 1994.