Soft-Decision Decoding of Binary Linear Block Codes

Yunghsiang S. Han

Department of Electrical Engineering,
National Taiwan University of Science and Technology
Taiwan

E-mail: yshan@mail.ntust.edu.tw

List Decoding Algorithms

1. Generate a list of candidate codewords.
   (a) With the help of algebraic decoding algorithms (such as MB decoding algorithm for BCH codes).
      i. Generalized minimum-distance (GMD) decoding by Forney (1966) [11, 30, 13, 28, 16, 1].
      ii. Chase algorithms by Chase (1972) [7] and Maximum-likelihood Chase algorithm*[a] by Kaneko et al. (1994) [25, 24].
   (b) With the help of trellis or code tree of the code.
      i. Viterbi algorithm for block codes* by Wolf (1978) [32].
      ii. Generalized Dijkstra algorithm* (Algorithm A*) by Han et al. (1993) [22, 18, 10, 17, 2, 26, 23, 19, 29].
   (c) With the help of reordering the received bits according to their reliability information.
      i. Ordered statistics decoding by Fossorier and Lin (1995)

[a] * optimal (ML) decoding algorithm

[14, 15].

2. Search for closest codeword in the list.

3. May incorporate with an optimality test criterion.

Distance Metrics of MLD Rule for Word-by-Word Decoding

Let m be a function such that $m(\boldsymbol{c}) = ((-1)^{c_0}, \ldots, (-1)^{c_{n-1}})$ and let $\boldsymbol{y} = (y_0, y_1, \ldots, y_{n-1})$ be the hard decision of $\boldsymbol{\phi}$. That is

$$y_j = \begin{cases} 1 & : \quad \text{if } \phi_j < 0; \\ 0 & : \quad \text{otherwise.} \end{cases}$$

1. $d_E^2(m(\boldsymbol{c}), \boldsymbol{\phi}) = \sum_{j=0}^{n-1} (\phi_j - (-1)^{c_j})^2$

2. $\langle \boldsymbol{\phi}, m(\boldsymbol{c}) \rangle = \sum_{j=0}^{n-1} (-1)^{c_j} \phi_j$

3. $d_D(\boldsymbol{c}, \boldsymbol{\phi}) = \sum_{j=0}^{n-1} (y_j \oplus c_j) |\phi_j| = \sum_{j \in T} |\phi_j|$, where $T = \{j | y_j \neq c_j\}$.

4. Relations between the metrics:
   - $d_E^2(m(\boldsymbol{c}), \boldsymbol{\phi}) = \sum_{j=0}^{n-1} \phi_j^2 - 2\langle \boldsymbol{\phi}, m(\boldsymbol{c}) \rangle + n = \|\boldsymbol{\phi}\|^2 + n - 2\langle \boldsymbol{\phi}, m(\boldsymbol{c}) \rangle$
   -
$$\begin{aligned} \langle \boldsymbol{\phi}, m(\boldsymbol{c}) \rangle &= \sum_{j \in T^c} |\phi_j| - \sum_{j \in T} |\phi_j| \\ &= \sum_{j \in (T \cup T^c)} |\phi_j| - 2 \sum_{j \in T} |\phi_j| \\ &= \sum_{j=0}^{n-1} |\phi_j| - 2 d_D(\boldsymbol{c}, \boldsymbol{\phi}) \end{aligned}$$

     where T^c is the complement set of T.

5. When one only considers BSC, $|\phi_j| = |\ln \frac{1-p}{p}|$ will be the same

for all positions. Thus,

$$d_D(\boldsymbol{c}, \boldsymbol{\phi}) = \sum_{j=0}^{n-1} (y_j \oplus c_j)|\phi_j| = |\ln \frac{1-p}{p}| \sum_{j=0}^{n-1} (y_j \oplus c_j)$$

In this case, the distance metric will reduce to the Hamming distance between $\boldsymbol{c}$ and $\boldsymbol{y}$, i.e., $\sum_{j=0}^{n-1}(y_j \oplus c_j) = d_H(\boldsymbol{c}, \boldsymbol{y})$. Under this condition, a decoder is called a *hard-decision* decoder; otherwise, the decoder is called a *soft-decision* decoder.

GMD Decoding (1)

1. The received vector is normalized as follows:

$$\tilde{\phi}_j = \begin{cases} +1 & : \quad \phi_j > +1, \\ \phi_j & : \quad |\phi_j| \leq 1, \\ -1 & : \quad \phi_j < -1. \end{cases}$$

2. Optimality test criterion: there is at most one codeword $\boldsymbol{c}_\ell$ such that

$$\left\langle \tilde{\boldsymbol{\phi}}, m(\boldsymbol{c}_\ell) \right\rangle > n - d_{min}.$$

3. This codeword can always be found by an algebraic erasure decoder Ψ which can correct t errors and ρ erasures under the condition that $2t + \rho < d_{min}$.

4. Reordering the bits in $\boldsymbol{\phi}$ with increasing information reliability, i.e.

$$\forall (i < j) \ |\tilde{\phi}_i| \leq |\tilde{\phi}_j|.$$

GMD Decoding (2)

Initialization: Calculate $\tilde{\boldsymbol{\phi}}$ and reorder the bits in it. $\rho = 0$. Let $\tilde{\boldsymbol{y}}$ be the hard decision of $\tilde{\boldsymbol{\phi}}$ after reordering.

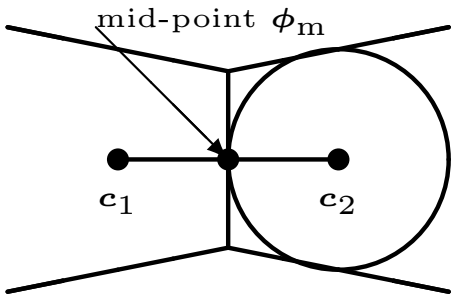
Step 1: Decode $\tilde{\boldsymbol{y}}$ with Ψ to obtain $\boldsymbol{c}$. If decoding failure, step 3.

Step 2: For $\left\langle m(\boldsymbol{c}), \tilde{\boldsymbol{\phi}} \right\rangle > n - d_{min}$, set $\boldsymbol{c}_\ell := \boldsymbol{c}$ as the closest codeword, stop.

Step 3: Set $\rho := \rho + 2$. For $\rho \geq d_{min}$, step 4. For $\rho < d_{min}$, define the first ρ positions of $\tilde{\boldsymbol{y}}$ as erasures, step 1.

Step 4: Decoding failure.

GMD Decoding (3)



1. Assume that $d_H(\boldsymbol{c}_1, \boldsymbol{c}_2) = d_{min}$.

   $$d_E^2(\boldsymbol{\phi}_{\mathrm{m}}, m(\boldsymbol{c}_2)) = \frac{1}{2} d_E^2(m(\boldsymbol{c}_1), m(\boldsymbol{c}_2)) = \frac{1}{2} \times d_{min} \times 2^2 = 2d_{min}$$

2. For any received vector $\boldsymbol{\phi}$ there is at most one codeword $\boldsymbol{c}_\ell$ such that

   $$d_E^2(\boldsymbol{\phi}, m(\boldsymbol{c}_\ell)) < 2d_{min}$$

3. If $|\tilde{\phi}_j| \leq 1$, then $\|\tilde{\boldsymbol{\phi}}\|^2 \leq n$.

4. $d_E^2(\tilde{\boldsymbol{\phi}}, m(\boldsymbol{c}_\ell)) \leq n + n - 2\left\langle \tilde{\boldsymbol{\phi}}, m(\boldsymbol{c}_\ell) \right\rangle$

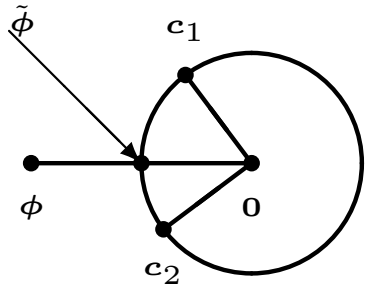
5. If $n + n - 2\left\langle \tilde{\boldsymbol{\phi}}, m(\boldsymbol{c}_\ell) \right\rangle < 2d_{min}$, i. e., $\left\langle \tilde{\boldsymbol{\phi}}, m(\boldsymbol{c}_\ell) \right\rangle > n - d_{min}$, then $d_E^2(\tilde{\boldsymbol{\phi}}, m(\boldsymbol{c}_\ell)) < 2d_{min}$.

Performance of GMD Decoding Algorithm

1. At very high SNR, GMD decoding gives effectively the same probability of error as maximum likelihood decoding.
2. By computer simulations, GMD decoding gives almost the same performance as the algebraic decoder Ψ used at low to moderate SNR.

Normalization of Received Vector

1. Forney's normalization
2. Sphere normalization
   - $\tilde{\boldsymbol{\phi}} = \dfrac{\boldsymbol{\phi}}{\frac{|\boldsymbol{\phi}|}{\sqrt{n}}}$. Thus, $\|\tilde{\boldsymbol{\phi}}\|^2 = n$.



Improvement of Optimality Test Criterion

- Taipale and Pursley's criterion (TP criterion) [30, 22]
  1. Consider all vectors $\boldsymbol{v}$ such that $d_H(\boldsymbol{c}, \boldsymbol{v}) = d_{min}$.
  2. A codeword $\boldsymbol{c}$ that satisfies
  $$d_D(\boldsymbol{c}, \boldsymbol{\phi}) \leq \min_{\boldsymbol{v}} d_D(\boldsymbol{v}, \boldsymbol{\phi})$$
  has the smallest distance to $\boldsymbol{\phi}$.
  3. The way to determine $\min_{\boldsymbol{v}} d_D(\boldsymbol{v}, \boldsymbol{\phi})$: For a given codeword $\boldsymbol{c}$, denote the set of indexes j such that $c_j \neq y_j$ by $S(\boldsymbol{c})$. Define $T(\boldsymbol{c})$ to be the set of $d_{min} - |S(\boldsymbol{c})|$ positions in $S(\boldsymbol{c})^c$ that have the smallest reliable values for ϕ_j. Then
  $$\min_{\boldsymbol{v}} d_D(\boldsymbol{v}, \boldsymbol{\phi}) = \sum_{j \in T(\boldsymbol{c})} |\phi_j|$$
  4. $\boldsymbol{\phi}$ does not need to be normalized in order to perform TP criterion.

5. From a geometrical interpretation, TP criterion is better than the one proposed by Forney. Mathematically, it can be proved as follows:
   - First we assume that $|S(\boldsymbol{c})| < d_{min}$ otherwise both criteria can not be met.
   - TP criterion can be rewritten as

$$\sum_{j \in S(\boldsymbol{c})} |\tilde{\phi}_j| < \sum_{j \in T(\boldsymbol{c})} |\tilde{\phi}_j|.$$

   - It is easy to show that $|(S(\boldsymbol{c}) \cup T(\boldsymbol{c}))^c| = n - d_{min}$.
   - The Forney's criterion can be rewritten as

$$\sum_{j=0}^{n-1} |\tilde{\phi}_j| - 2d_D(\boldsymbol{c}, \tilde{\boldsymbol{\phi}}) > n - d_{min}$$

$$\Leftrightarrow \sum_{j \in S(\boldsymbol{c}) \cup T(\boldsymbol{c}) \cup (S(\boldsymbol{c}) \cup T(\boldsymbol{c}))^c} |\tilde{\phi}_j| - 2 \sum_{j \in S(\boldsymbol{c})} |\tilde{\phi}_j| > n - d_{min}$$

$$\Leftrightarrow \sum_{j \in S(\boldsymbol{C})} |\tilde{\phi}_j| + \sum_{j \in (S(\boldsymbol{C}) \cup T(\boldsymbol{C}))^c} (1 - |\tilde{\phi}_j|) < \sum_{j \in T(\boldsymbol{C})} |\tilde{\phi}_j|$$

Apparently, the above criterion is more stringent than TP criterion since $0 \leq (1 - |\tilde{\phi}_j|) \leq 1$ for all j.

6. The codeword which satisfies TP criterion can always be found by the same decoding procedure as that of GMD if d_{min} is odd. If d_{min} is even, the erasure procedure should erase the positions up to position 1, position 3,. . ., position $d_{min} - 1$.
   - We prove the above statement by contrapositive.
   - Let d_{min} be odd and $\boldsymbol{c}_m$ be the closest codeword to the received vector.
   - Assume that the decoding procedure fail to find any codeword which satisfies TP criterion, that is, when the first j positions of the hard-decision of received vector $\boldsymbol{y}$ were erased, $j = 0, 2, 4, \ldots, d_{min} - 1$, $\boldsymbol{c}_m$ could not be

found by the algebraic decoder Ψ.

– Since $\boldsymbol{c}_m$ could not be found when no positions was erased, there were at least $t+1$ errors exists on $\boldsymbol{y}$. Consequently, $d_D(\boldsymbol{c}_m, \boldsymbol{\phi}) \geq \sum_{j \in S'} |\phi_j|$ where $S' = \{1, 2, \ldots, t+1\}$. Furthermore, Since $\boldsymbol{c}_m$ could not be found when the first two positions were erased, there were at least t errors exists at positions j, $j = 3, 4, \ldots, n$, of $\boldsymbol{y}$. Consequently, $d_D(\boldsymbol{c}_m, \boldsymbol{\phi}) \geq \sum_{j \in S'} |\phi_j|$ where $S' = \{1, 3, 4, \ldots, t+2\}$.

– If we continue in this fashion, it is easy to show that $d_D(\boldsymbol{c}_m, \boldsymbol{\phi}) \geq \sum_{j \in S'} |\phi_j|$ where $S' = \{1, 3, 5, \ldots, d_{min}\}$. Furthermore,

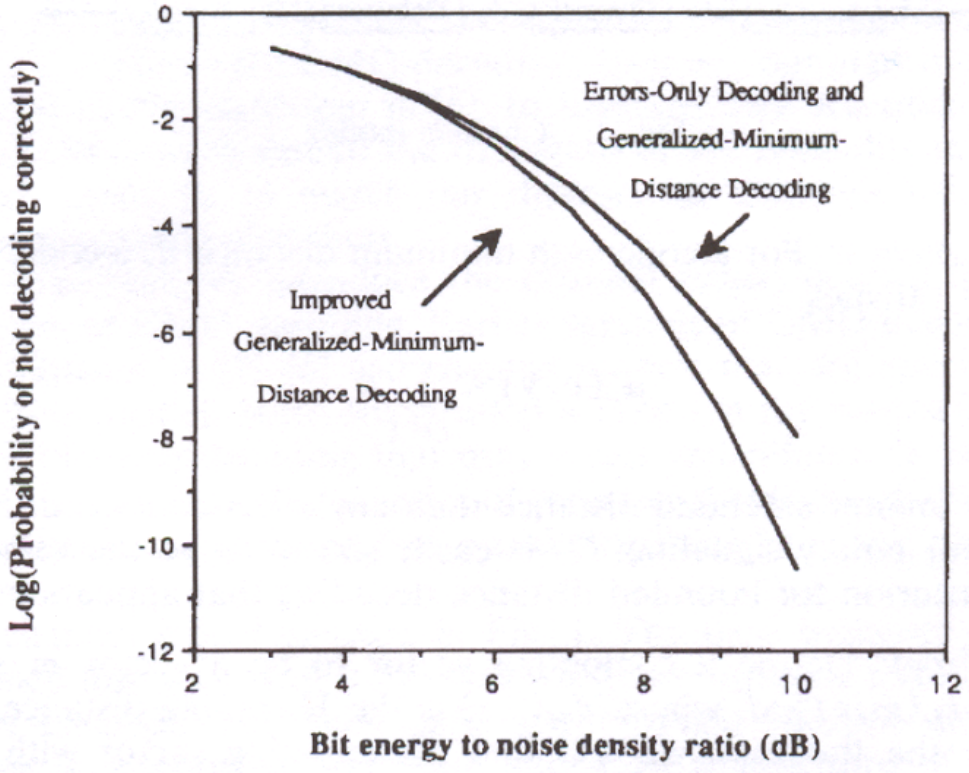
$$\min_{\boldsymbol{v}} d_D(\boldsymbol{v}, \boldsymbol{\phi}) = \sum_{j \in T(\boldsymbol{c}_m)} |\phi_j| \leq \sum_{j \in T'} |\phi_j|$$

where $T' = \{2, 4, 6, \ldots, d_{min} - 1\}$. Clearly,

$$\min_{\boldsymbol{v}} d_D(\boldsymbol{v}, \boldsymbol{\phi}) < d_D(\boldsymbol{c}_m, \boldsymbol{\phi})$$

The Performance of Improved GMD Decoding Algorithm

Comparison of improved GMD decoding with GMD decoding for the $(31, 15, 7)$ BCH code (on word error)



Bounded-Distance Decoding Algorithm [28]

1. A bounded-distance decoding algorithm is one that decodes correctly whenever the received vector is within Euclidean distance ρ of a codeword, where ρ is the guaranteed error-correction radius of the code.

2. Define the decision regions $D_A(\boldsymbol{c})$ of an decoding algorithm as the set of received vectors $\boldsymbol{\phi}$ that are decoded to $\boldsymbol{c}$.

3. Let $O(\boldsymbol{c})$ be the open ball consisting of all vectors within distance $\rho = \sqrt{d_{min}}$ of $m(\boldsymbol{c})$. That is,

$$O(\boldsymbol{c}) = \{\boldsymbol{\phi} | d_E^2(m(\boldsymbol{c}), \boldsymbol{\phi}) < \rho^2 = d_{min}\}$$

4. A decoding algorithm A is a bounded-distance decoding algorithm if $O(\boldsymbol{c}) \subset D_A(\boldsymbol{c})$ for all $\boldsymbol{c} \in \boldsymbol{C}$.

5. ML decoding is a bounded-distance decoding algorithm.

6. GMD decoding with sphere normalization and improved GMD decoding are bounded-distance decoding algorithms.
7. The *effective error coefficient* is the coefficient of the first (minimum-distance) term of the union bound on error probability for a bounded-distance decoding algorithm.
8. The effective error coefficient of ML decoding is $A_{d_{min}}$, the number of codewords with minimum Hamming weight.
9. The effective error coefficient of GMD decoding with TP criterion can be shown to be $\binom{n}{d_{min}}$ [28].
10. Since $\binom{n}{d_{min}} \gg A_{d_{min}}$ one may expect that the bit error probability of GMD decoding is much larger than that of ML decoding.

Full GMD and Modified GMD Decoding

1. If one calculates the distances between the received vector and all codewords obtained by the erasure procedure, and output the closest codeword among these codewords, then the GMD decoding becomes a full GMD decoding. Clearly, the performance of full GMD is better than those of the GMD decoding and GMD with TP criterion.
2. The modified GMD is obtained by adding a d_{min}-erasure-correction step to the full GMD decoding [28, 13].
3. The step is implemented as follows:
   - Erase the d_{min} least reliable bits;
   - Make the hard-decision on the remaining $n - d_{min}$ bits;
   - Use these hard-decision bits to solve for d_{min} erased (unknown) bits by the parity-check equations which define $\boldsymbol{C}$;

- Compare distances to the received vector of the codeword reported by the full GMD decoding and codewords obtained above if they exits; Output the one with smaller distance.

4. The time complexity of the above step is of the order $O(d_{min}^3)$.

5. It can be proved that the effective error coefficient for modified GMD decoding is $A_{d_{min}}$.

6. Even through the modified GMD decoding has the same effective error coefficient as an ML decoder, they have different performance on bit error probability. The reasons for this unexpected result are (1) the coefficient of the term involving $d_{min} + 1$ of union bound for modified GMD decoding is still much larger than that for ML decoding (2) there might be points on the boundary of the modified GMD decoding region whose distance from $m(\mathbf{0})$ is between d_{min} and $d_{min} + 1$ [16, 1].

Chase Algorithms (1)

1. Candidate codeword is generated by an algebraic decoding algorithm Ψ which can correct up to $t = \lfloor \frac{d_{min}-1}{2} \rfloor$ errors.
2. Reordering the bits in $\boldsymbol{\phi}$ with increasing information reliability.
3. A set $\mathcal{T}$ of test solutions, which is based on a binary test vector $\boldsymbol{v} \in GF(2)^n$, is generated.
4. Every test vector $\boldsymbol{v}$ is added to the hard-decision received vector $\boldsymbol{y}$ and the modified received vector $\boldsymbol{y} + \boldsymbol{v}$ is decoded using algorithm Ψ to produce candidate codeword.

Chase Algorithms (2)

There are three methods to generate $\mathcal{T}$:

Variant 1 The set $\mathcal{T}$ is given through all binary vectors of weight less than or equal to $\lfloor \frac{d_{min}}{2} \rfloor$ e,g, $|\mathcal{T}| = \sum_{i=0}^{\lfloor \frac{d_{min}}{2} \rfloor} \binom{n}{i}$.

Variant 2 The test solution set $\mathcal{T}$ is calculated using all binary vector combinations corresponding to the $t = \lfloor \frac{d_{min}}{2} \rfloor$ most unreliable positions (least reliable positions). That is, $|\mathcal{T}| = 2^{\lfloor d_{min}/2 \rfloor}$.

Variant 3 Determine the $d_{min} - 1$ most unreliable positions. The vectors in $\mathcal{T}$ have ones in the i most unreliable positions and zeros elsewhere ($i = 0, 2, \ldots, d_{min} - 1$ and $i = 0, 1, 3, \ldots, d_{min} - 1$ for odd and even d_{min} respectively). The cardinality of $\mathcal{T}$ is $|\mathcal{T}| = \lfloor \frac{d_{min}}{2} \rfloor + 1$.

Chase Algorithms (3)

Initialization: Reordering the bits in $\boldsymbol{\phi}$ with increasing information reliability. Calculate the set of test vectors $\mathcal{T}$ according to variant 1,2 or 3. Set $i = 0$ and $\mathcal{L} = \emptyset$ and max $= 0$.

Step 1: Decode $\hat{\boldsymbol{y}} = \boldsymbol{y} \oplus \boldsymbol{v}_i, \boldsymbol{v}_i \in \mathcal{T}$, with Ψ to give $\hat{\boldsymbol{c}}$. If a decoding failure is received then step 3.

Step 2: If $\langle m(\hat{\boldsymbol{c}}), \boldsymbol{\phi} \rangle >$ max, then max $= \langle m(\hat{\boldsymbol{c}}), \boldsymbol{\phi} \rangle$ and $\mathcal{L} = \{\hat{\boldsymbol{c}}\}$.

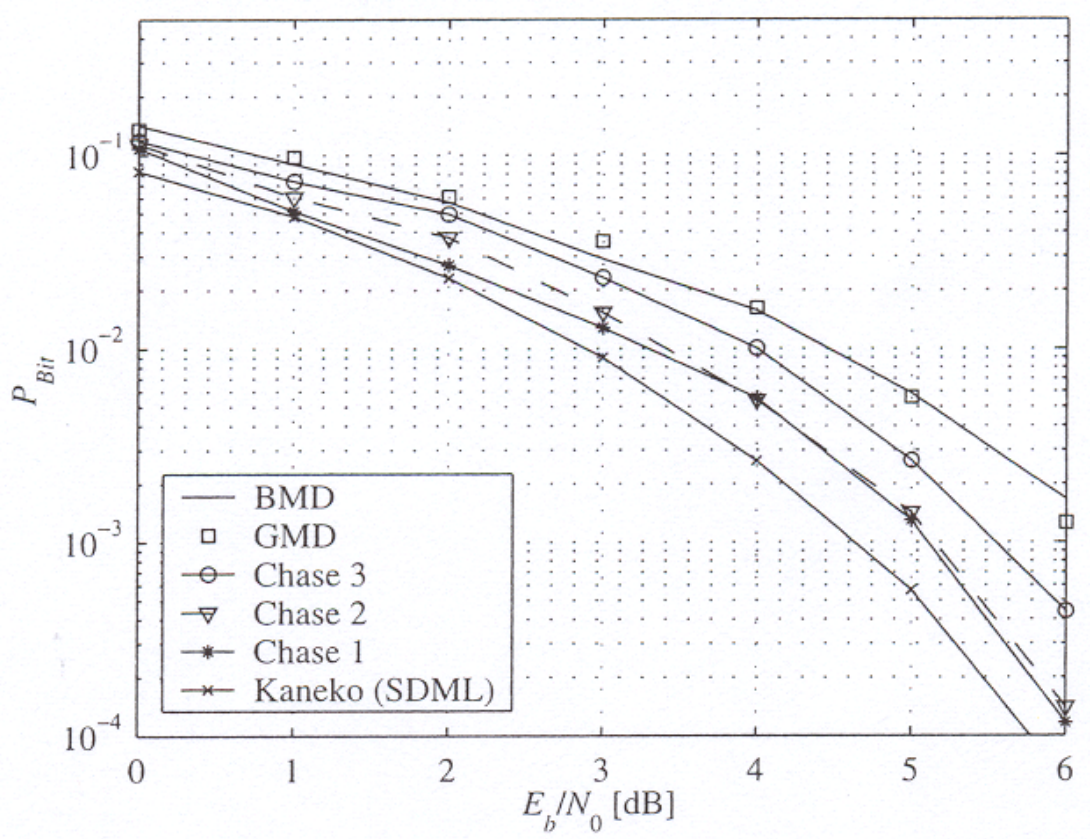
Step 3: For $i < |\mathcal{T}|$, set $i = i + 1$, step 1; otherwise step 4.

Step 4: Determine the decoding result according to

$$\boldsymbol{c}_f = \begin{cases} \hat{\boldsymbol{c}} \in \mathcal{L} & : \quad \text{if } \mathcal{L} \neq \emptyset, \\ \boldsymbol{y} & : \quad \text{otherwise.} \end{cases}$$

Simulation Results

Comparison of GMD, Chase with Maximum-likelihood Chase algorithms for the $(15, 7, 5)$ BCH code



Maximum-Likelihood Chase Algorithm (1) [5]

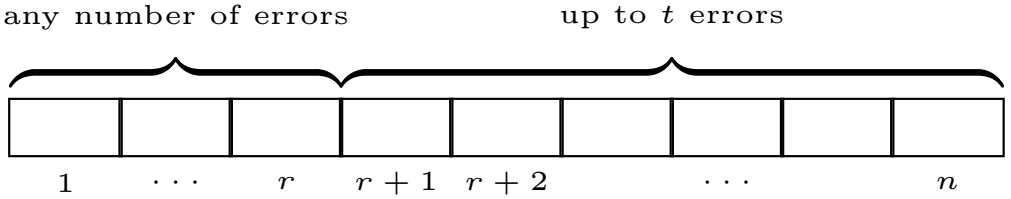
1. Maximum-likelihood Chase algorithm is a maximum-likelihood version of Chase algorithm.
2. As the Chase algorithm, the test solution set are subdivided based on the Hamming weight.
3. All bit solutions are considered that are made up of any combination of ones and zeros occupying the r most unreliable positions (similar to variant 2 of Chase algorithm). That is

$$\begin{aligned} \mathcal{T}_r &= \{\boldsymbol{v} = (v_1, v_2, \ldots, v_n) | (v_1, v_2, \ldots, v_r, 0, \ldots, 0), \\ &\qquad v_i \in \{0, 1\} \text{ for all } i = 1, 2, \ldots, r\} \end{aligned}$$

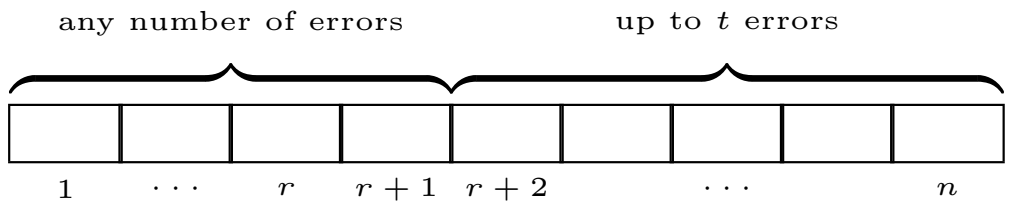
4. Let $\mathcal{L}_r$ be the set of all codewords that are contained in the test solution set $\mathcal{T}_r$.
5. $\mathcal{T}_r \subseteq \mathcal{T}_{r+1}$ and $\mathcal{L}_r \subseteq \mathcal{L}_{r+1}$.

Maximum-Likelihood Chase Algorithm (2)

Codeword structure in $\mathcal{L}_r$

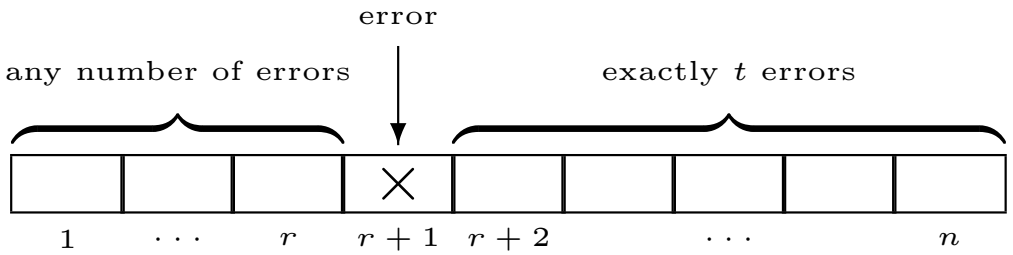


Codeword structure in $\mathcal{L}_{r+1}$



Maximum-Likelihood Chase Algorithm (3)

Codeword structure in $\mathcal{L}_{r+1} \setminus \mathcal{L}_r$



1. $\mathcal{L}_{r+1} \setminus \mathcal{L}_r$ represents the codewords that are in $\mathcal{L}_{r+1}$ but not in $\mathcal{L}_r$.
2. Recall that the bits in $\boldsymbol{\phi}$ are reordered with increasing information reliability, i.e.,

$$\forall (i < j) |\phi_i| \leq |\phi_j|.$$

3. For all $\boldsymbol{c} \in \mathcal{L}_{r+1} \setminus \mathcal{L}_r$, we have

$$d_D(\boldsymbol{c}, \boldsymbol{\phi}) \geq \sum_{j=r+1}^{r+1+t} |\phi_j|.$$

4. If the codeword $\widehat{\boldsymbol{c}}$ that has the smallest distance belonging to the test solution set $\mathcal{T}_r$ and $d_D(\widehat{\boldsymbol{c}}, \boldsymbol{\phi}) \leq \sum_{j=r+1}^{r+1+t} |\phi_j|$, then $\widehat{\boldsymbol{c}}$ is the maximum-likelihood codeword.

5. Optimality test criterion :

$$d_D(\widehat{\boldsymbol{c}}, \boldsymbol{\phi}) \leq \sum_{j=r+1}^{r+1+t} |\phi_j| \text{ and } \widehat{\boldsymbol{c}} = \min_{\boldsymbol{c}} \{d_D(\boldsymbol{c}, \boldsymbol{\phi}) | \boldsymbol{c} \in \mathcal{L}_r\}$$

Maximum-Likelihood Chase Algorithm (4)

Initialization: Reordering the bits in $\boldsymbol{\phi}$ with increasing information reliability. Set $r = n, j = 0, d_D(\widehat{\boldsymbol{c}}, \boldsymbol{\phi}) = \infty$.

Step 1: If $(j \leq 2^r - 1)$ then

(a) Decode $\boldsymbol{y} \oplus \boldsymbol{v}_j$, $v_j \in \mathcal{T}_r$, with Ψ to give $\boldsymbol{c}$. For a decoding failure, step (c).

(b) If $d_D(\boldsymbol{c}, \boldsymbol{\phi}) < d_D(\widehat{\boldsymbol{c}}, \boldsymbol{\phi})$, then

- If $\boldsymbol{c}$ satisfies the optimality test criterion, $\boldsymbol{c}_\ell = \boldsymbol{c}$, stop.
- Set $\widehat{\boldsymbol{c}} = \boldsymbol{c}$.
- Update r according to the optimality test criterion.

(c) j=j+1, step 1.

Step 2: $\boldsymbol{c}_\ell = \widehat{\boldsymbol{c}}$.

The Time Complexity of Maximum-Likelihood Chase Algorithm

Comparison of maximum-likelihood Chase algorithm with A* decoding algorithm for the $(128, 64, 22)$ BCH code

SNR	5 dB		6 dB		7 dB	
	max	ave	max	ave	max	ave
A* algorithm*	216052	42	13603	2	1143	1
ML Chase algorithm**	2097152	283	1024	1	1	1

* the number of nodes visited during decoding of $\boldsymbol{\phi}$;

** the number of algebraic decoder called.

Equivalent Code of the Transmitted Code (1) [22, 18]

1. Let $\boldsymbol{C}$ be the transmitted code. An equivalent code $\boldsymbol{C}^*$ of $\boldsymbol{C}$ can be obtained by permuting the positions of codewords of $\boldsymbol{C}$ in such a way that the first k positions of codewords in $\boldsymbol{C}^*$ correspond to the "most reliable linearly independent" positions in the received vector $\boldsymbol{\phi}$.
2. The algorithm given later shows how to transform the generator matrix of $\boldsymbol{C}$ to that of $\boldsymbol{C}^*$ whose first k columns form the $k \times k$ identity matrix.
3. The time complexity of this algorithm is $O(k^2 \times n)$.
4. The vector $\boldsymbol{\phi}^* = (\phi_0^*, \phi_1^*, \ldots, \phi_{n-1}^*)$ can be used as the "received vector." It is obtained by permuting the positions of $\boldsymbol{\phi}$ in the same manner in which the columns of $\boldsymbol{G}$ can be permuted to obtain $\boldsymbol{G}^*$.

Equivalent Code of the Transmitted Code (2)[4, 18]

1. Let $\boldsymbol{\phi}' = (\phi'_0, \phi'_1, \ldots, \phi'_{n-1})$ be a vector obtained by permuting the positions of $\boldsymbol{\phi}$ such that $|\phi'_i| \geq |\phi'_{i+1}|$ for $0 \leq i < n-1$.
2. The $k \times n$ matrix $\boldsymbol{G}'$ is obtained from $\boldsymbol{G}$ by applying this same permutation to the columns of $\boldsymbol{G}$.
3. Let $\boldsymbol{A}$ be an $r \times m$ matrix. Given a set $S = \{i_1, i_2, \ldots, i_s\} \subset \{0, 1, 2, \ldots, m-1\}$ we say that S is a sub-information set of $\boldsymbol{A}$ iff the columns of $\boldsymbol{A}$ indexed by $i_1, i_2, \ldots, i_s$ are linearly independent.
4. For $0 \leq i, j < m$, $SW(\boldsymbol{A}, i, j)$ is the $r \times m$ matrix obtained from $\boldsymbol{A}$ by swapping columns i and j of $\boldsymbol{A}$.

Equivalent Code of the Transmitted Code (3) [18]

The following is an algorithm to obtain $\boldsymbol{G}^*$ from $\boldsymbol{G}'$ for $2 \leq k < n$.

Initialization: $i \leftarrow 1; j \leftarrow 1; S = \{0\}$; $\boldsymbol{G}_1^* \leftarrow \boldsymbol{G}'$.

Step 1: If $S \cup \{j\}$ is a sub-information set of $\boldsymbol{G}_1^*$, then

$\boldsymbol{G}_1^* \leftarrow SW(\boldsymbol{G}_1^*, i, j)$;

else

$j \leftarrow j + 1$;

step 1.

Step 2: $S \leftarrow S \cup \{i\}$.

Step 3: If $|S| = k$, then stop;

else

$i \leftarrow i + 1$;

$j \leftarrow j + 1$;

step 1.

Step 4: Transform $\boldsymbol{G}_1^*$ into $\boldsymbol{G}^*$ by row operation such that the first k columns of $\boldsymbol{G}^*$ form a $k \times k$ identity matrix.

Equivalent Code of the Transmitted Code (4)

It can be shown that the above algorithm always finds "the most reliable linearly independent" k positions in $\boldsymbol{\phi}'$ and thus in $\boldsymbol{\phi}$.

Proof: Assume that $S = \{s_1, s_2, \ldots, s_k\}$ is the positions found by the previous algorithm and $T = \{t_1, t_2, \ldots, t_k\}$ is another set of "linearly independent" k positions in $\boldsymbol{\phi}'$ such that

$$\sum_{j \in T} |\phi'_j| > \sum_{j \in S} |\phi'_j|.$$

Without loss of generality, we may assume that $|\phi'_{s_i}| \geq |\phi'_{s_j}|$ and $|\phi'_{t_i}| \geq |\phi'_{t_j}|$ for $i < j$. It is easy to see that there must exist j such that $|\phi'_{t_j}| > |\phi'_{s_j}|$ otherwise $\sum_{j \in T} |\phi'_j| \leq \sum_{j \in S} |\phi'_j|$. Let j be the smallest number with the above property. Since $t_i < s_j$ for all $i \leq j$, t_i either has been selected in S before s_j by the algorithm or can not be selected due to linearity constraint. Then t_i for $i \leq j$ are in the subspace spanned by $\{s_1, s_2, \ldots, s_{j-1}\}$. However,

$\{t_1, t_2, \ldots, t_j\}$ are linearly independent which can not all from a subspace with dimension less than j. Contradiction.

Decoding Based on Ordered Statistics

1. The decoding based on ordered statistics use $\boldsymbol{\phi}^*$ and $\boldsymbol{G}^*$ to generate a test solution set based on the most reliable information positions k, and the $n-k$ remaining positions are recalculated.

2. A test solution set of codewords is created where the values in successive positions of the first k information positions are inverted, and the corresponding codewords are constructed by $\boldsymbol{G}^*$.

Decoding of Order l (Order-l Reprocessing)

Initialization: Calculate $\boldsymbol{G}^*$, $\boldsymbol{\phi}^*$ and the hard-decision $\boldsymbol{y}^*$ of $\boldsymbol{\phi}^*$. Let $\boldsymbol{y}^* = (y_0^*, y_1^*, \ldots, y_{k-1}^*, y_k^*, y_{k+1}^*, \ldots, y_{n-1}^*) = (\boldsymbol{y}_k^* | \boldsymbol{y}_{n-k}^*)$. Set $\boldsymbol{c}_f = \boldsymbol{y}_k^* \boldsymbol{G}^*$. $j = 1$

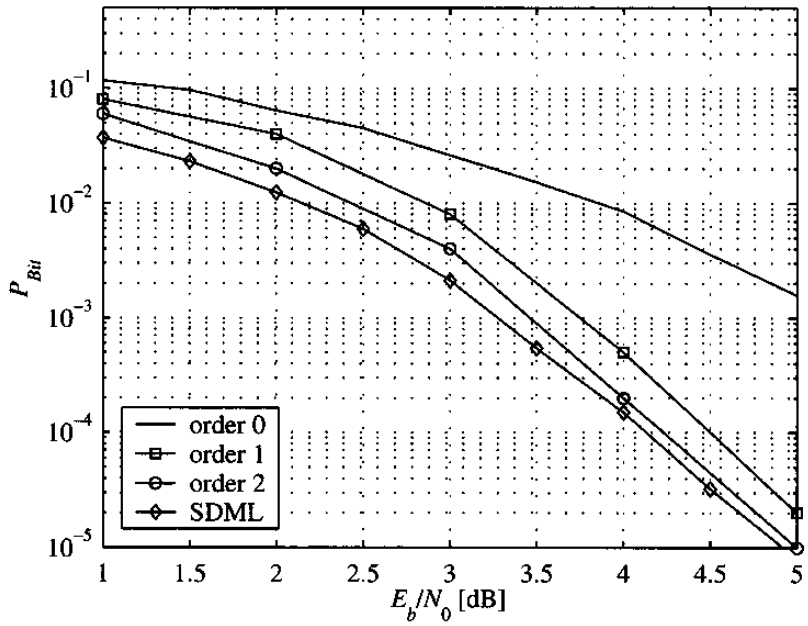
Step 1: If $j \leq l$,

- Calculate the test solution set $\mathcal{T}_j = \{\boldsymbol{v} \in GF(2)^k | W_H(\boldsymbol{v}) = j\}$.
- Calculate the test codeword set $\mathcal{L}_j = \{(\boldsymbol{y}_k^* \oplus \boldsymbol{v}) \cdot \boldsymbol{G}^* | \boldsymbol{v} \in \mathcal{T}_j\}$.
- Find the codeword $\widehat{\boldsymbol{c}}$ in $\mathcal{L}_j$ that is closest to the received vector $\boldsymbol{\phi}^*$.
- If $d_D(\widehat{\boldsymbol{c}}, \boldsymbol{\phi}^*) < d_D(\boldsymbol{c}_f, \boldsymbol{\phi}^*)$, then $\boldsymbol{c}_f = \widehat{\boldsymbol{c}}$.
- j=j+1; step 1.

Step 2: Convert $\boldsymbol{c}_f$ to the corresponding codeword of transmitted code.

Performance of Decoding Based on Ordered Statistics

Comparison of decoding based on order statistics of orders 0, 1, and 2 with maximum likelihood decoding for the $(64, 42, 8)$ RM code



Trellis of Linear Block Codes [3]

1. Let $\boldsymbol{H}$ be a parity-check matrix of $\boldsymbol{C}$, and let $\boldsymbol{h_j}$, $0 \le j \le n-1$ be the column vectors of $\boldsymbol{H}$.

2. Let $\boldsymbol{c} = (c_0, c_1, \ldots, c_{n-1})$ be a codeword of $\boldsymbol{C}$. With respect to this codeword, we recursively define the states $\boldsymbol{s_t}$, $-1 \le t \le n-1$, as

$$\boldsymbol{s}_{-1} = \boldsymbol{0}$$

and

$$\boldsymbol{s}_t = \boldsymbol{s}_{t-1} + c_t \boldsymbol{h}_t = \sum_{j=0}^{t} c_j \boldsymbol{h}_j, \quad 0 \le t \le n-1.$$

3. $\boldsymbol{s}_{n-1} = \boldsymbol{0}$ for all codewords of $\boldsymbol{C}$.

4. The above recursive equation can be used to draw a trellis diagram.

5. In this trellis, $\boldsymbol{s}_{-1} = \boldsymbol{0}$ identifies the start node at level -1;

$\boldsymbol{s}_{n-1} = \mathbf{0}$ identifies the goal node at level $n-1$; and each state $\boldsymbol{s}_t, 0 \leq t \leq n-2$ identifies a node at level t.

6. Each transition (branch) is labelled with the appropriate codeword bit c_t.

7. There is a one-to-one correspondence between the codewords of $\boldsymbol{C}$ and the sequences of labels encountered when traversing a path in the trellis from the start node to the goal node.

Example of Trellis

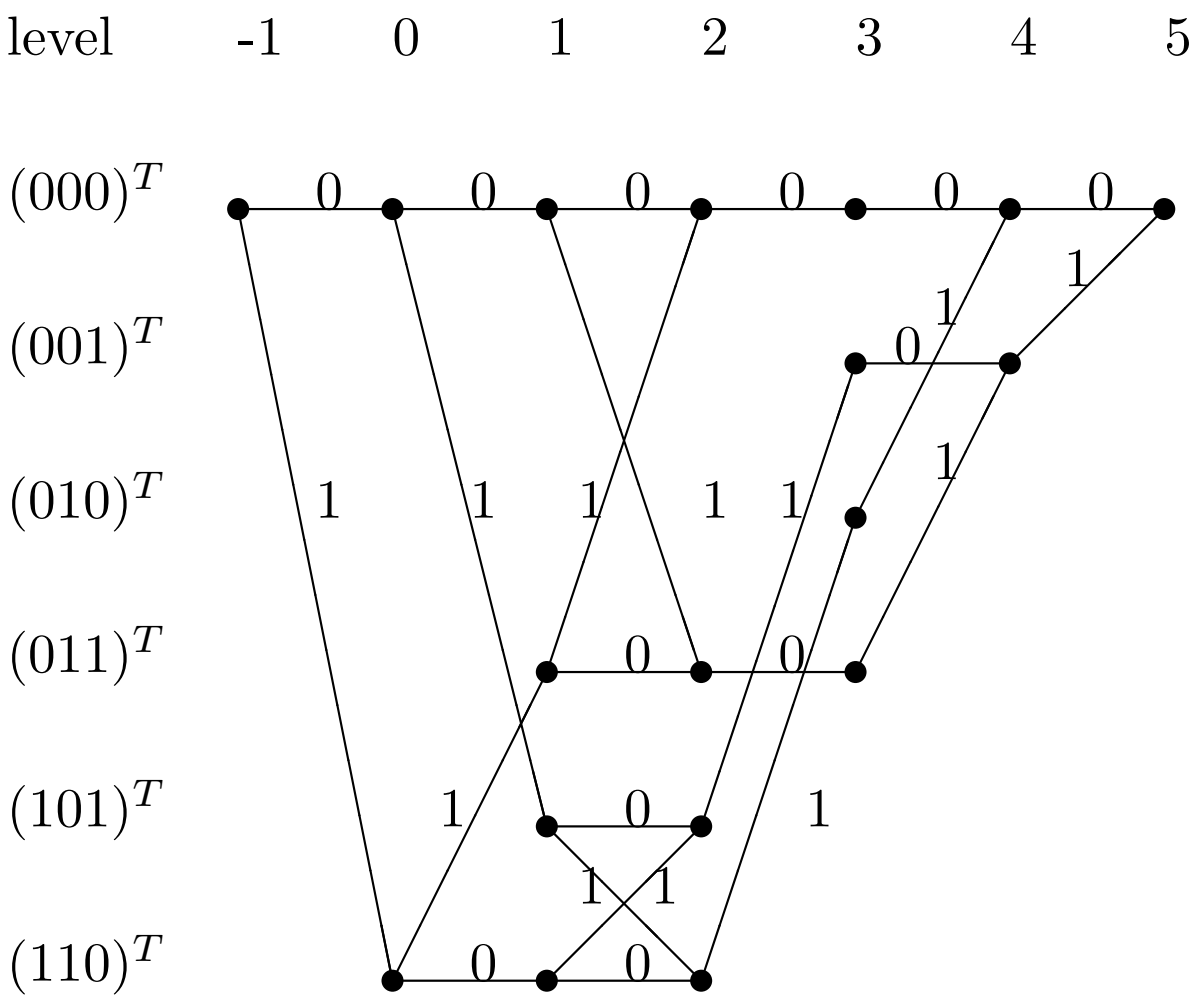
Let

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

be a generator matrix of $\boldsymbol{C}$ and let

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

be a parity-check matrix of $\boldsymbol{C}$.


level -1 0 1 2 3 4 5
$(000)^T$
$(001)^T$
$(010)^T$
$(011)^T$
$(101)^T$
$(110)^T$

Code Tree of Linear Block Code

1. A code tree is another way to represent every codeword of an (n, k) code $\boldsymbol{C}$ as a path through a tree containing $n + 1$ levels.
2. The code tree can be treated as an expanded version of the trellis, where every path is totally distinct from every other path.
3. The leftmost node is called the *start node*.
4. There are two branches, labelled by 0 and 1, respectively, that leave each node at the first k levels. After the k levels, there is only one branch leaving each node.
5. The 2^k rightmost nodes are called *goal nodes*.
6. Next, we describe how to determine the sequence of labels encountered when traversing a path from a node at level k to a goal node:

- Let $\boldsymbol{G}$ be a generating matrix of $\boldsymbol{C}$ whose first k columns form the $k \times k$ identity matrix.
- Let $c_0, c_1, \ldots, c_{k-1}$ be the sequence of labels encountered when traversing a path from the start node to a node m at level $k-1$.
- $c_k, c_{k+1}, \ldots, c_{n-1}$, the sequence of labels encountered when traversing a path from node m to a goal node, can be obtained as follows:

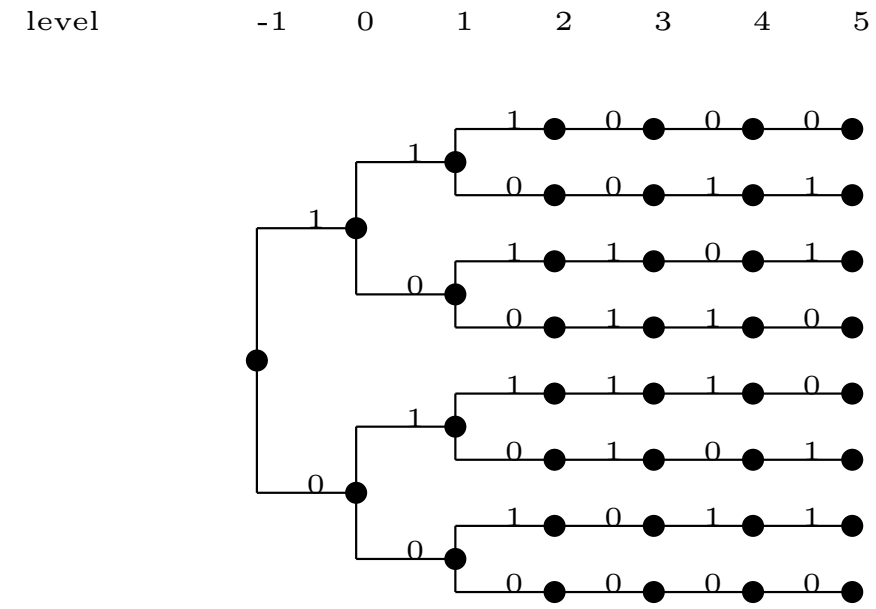
$$(c_0, c_1, \ldots, c_k, c_{k+1}, \ldots, c_{n-1}) = (c_0, c_1, \ldots, c_{k-1})\boldsymbol{G}.$$

Example of Code Tree

Let

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

be a generator matrix of $\boldsymbol{C}$.



Convert a Decoding Problem to a Graph-Search Problem

1. According to the MLD rule presented, we want to find a codeword $\boldsymbol{c_\ell}$ such that $d_E^2(m(\boldsymbol{c}_\ell), \boldsymbol{\phi}) = \sum_{j=0}^{n-1} (\phi_j - (-1)^{c_{\ell j}})^2$ is the minimum among all the codewords.
2. If we properly specify the branch costs (branch metrics) on a trellis or a code tree, the **MLD** rule can be written as follows:

   Find a path from the start node to a goal node such that the cost of the path is minimum among all the paths from the start node to a goal node, where a cost of a path is the summation of the cost of branches in the path. Such a path is called an *optimal path*.
3. In the trellis of $\boldsymbol{C}$ the cost of the branch from $\boldsymbol{s}_{t-1}$ to $\boldsymbol{s}_t = \boldsymbol{s}_{t-1} + c_t \boldsymbol{h}_t$ is assigned the value $(\phi_t - (-1)^{c_t})^2$.

4. In the code tree of $\boldsymbol{C}$, the cost of the branch from a node at level $t-1$ to a node at level t is assigned the value $(\phi_t - (-1)^{c_t})^2$, where c_t is the label of the branch.

5. The solution of the decoding problem is converted into finding a path from the start node to a goal node in the trellis (code tree), that is, a codeword $\boldsymbol{c}= (c_0, c_1, \ldots, c_{n-1})$ such that $d_E^2(m(\boldsymbol{c}), \boldsymbol{\phi})$ is minimum among all paths from the start node to a goal node.

6. The *path metric* for a codeword $\boldsymbol{c}= (c_0, c_1, \ldots, c_{n-1})$ is defined as

$$\sum_{j=0}^{n-1} (\phi_j - (-1)^{c_j})^2$$

7. The *ℓth partial path cost (metric)* $M^{\ell}(\boldsymbol{P}_m)$ for a path $\boldsymbol{P}_m$ is obtained by summing the branch metrics for the first ℓ branches of the path, where node m is the ending node of $\boldsymbol{P}_m$.

8. If $\boldsymbol{P}_m$ has labels $\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell$, then

$$M^\ell(\boldsymbol{P}_m) = \sum_{i=0}^{\ell} \left(\phi_i - (-1)^{\overline{v}_i}\right)^2$$

Viterbi Decoding Algorithm (Wolf Algorithm) (1)

1. Wolf algorithm [32] finds an optimal path by applying the Viterbi algorithm [31] to search through the trellis for $\boldsymbol{C}$.
2. In Wolf algorithm, each node in the trellis is assigned a cost. This cost is the partial path cost of the path starting at start node and ending at that node.
3. When more than one path enters a node at level ℓ in the trellis, one chooses the path which has the best (minimum) ℓth partial path cost as the cost of the node. The path with the best cost is the *survivor*.
4. The algorithm terminates when all nodes in the trellis have been labelled and their entering survivors determined.
5. Viterbi algorithm and Wolf algorithm are special types of

dynamic programming, consequently, they are ML decoding algorithms.

6. This decoding algorithm uses a breadth-first search strategy to accomplish this search. That is, the nodes will be labelled level by level until the last level containing the goal node in the trellis.

7. The time and space complexities of Wolf algorithm are of $O(n \times \min(2^k, 2^{n-k}))$ [8], since it traverses the entire trellis.

8. Forney has given a procedure to reduce the number of states in the trellis for $\boldsymbol{C}$ [12] and now it becomes an active research area.

Viterbi Decoding Algorithm (Wolf Algorithm)(2)

Initialization: $\ell = -1$. Let $g_\ell(m)$ be the partial path cost assigned to node m at level ℓ. $g_\ell(m) = 0$.

Step 1: Compute the partial metrics for all paths entering each node at level ℓ.

Step 2: Set $g_\ell(m)$ equal to the best partial path cost entering the node m. Delete all non survivors from the trellis.

Step 3: If $\ell = n - 1$, then stop and output the only survivor ending at the goal node; otherwise $\ell = \ell + 1$, step 1.

Viterbi Decoding Algorithm (Wolf Algorithm) (3)

Let

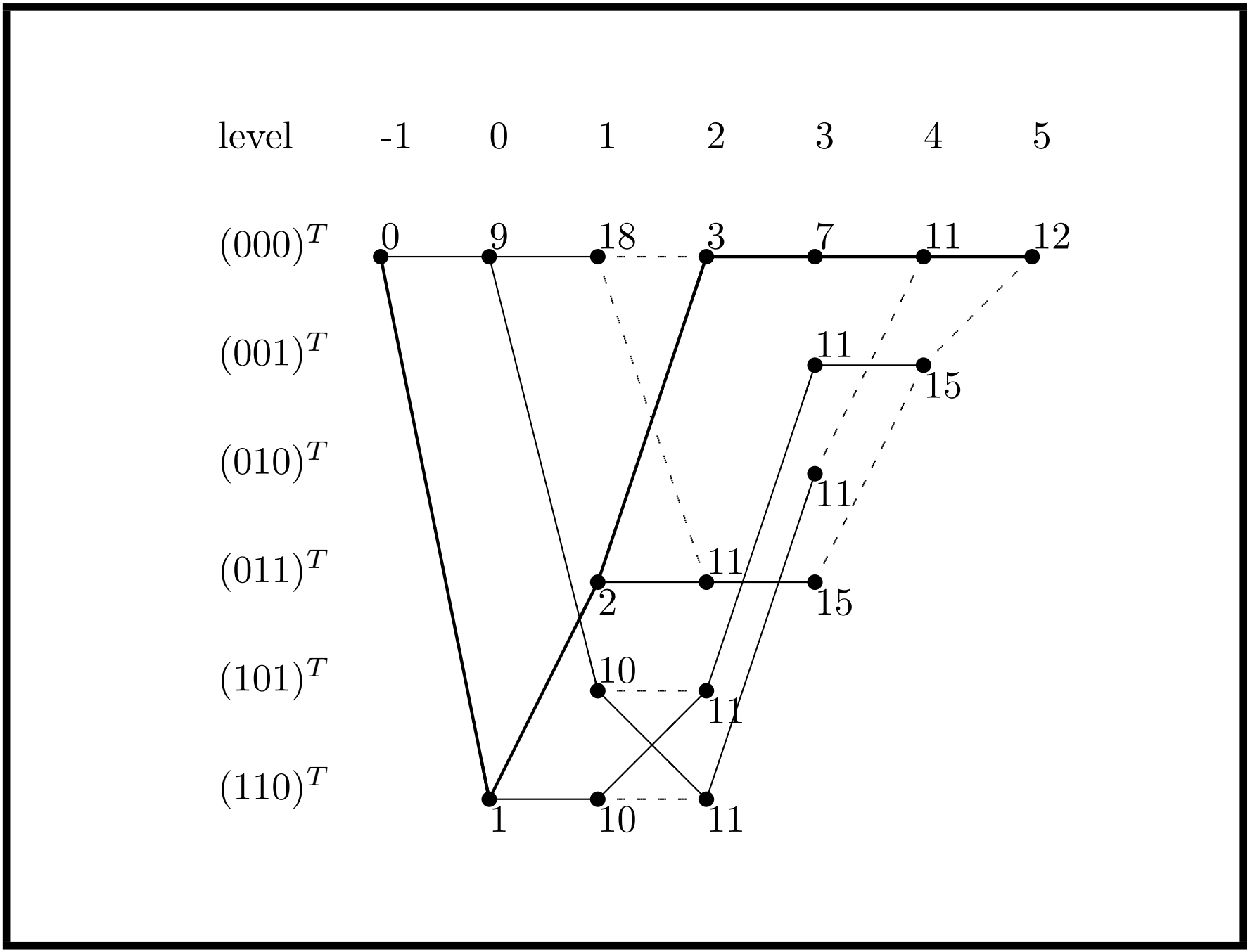
$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

be a generator matrix of $\boldsymbol{C}$ and let

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

be a parity-check matrix of $\boldsymbol{C}$. Assume the received vector is

$$\boldsymbol{\phi} = (-2, -2, -2, -1, -1, 0)$$


level
-1
0
1
2
3
4
5
$(000)^T$
$(001)^T$
$(010)^T$
$(011)^T$
$(101)^T$
$(110)^T$
0
9
18
3
7
11
12
11
15
11
11
15
2
10
11
1
10
11

Priority-First Search Algorithm (PFSA) for Code Tree (1)

1. In order to avoid traversing the entire trellis, a different search strategy—such as the priority-first search—must be employed.
2. This modified algorithm may not visit all the nodes in the graph.
3. A *successor operator*, when applied to a node m, (1) gives all the immediate successors of node m; (2) for every immediate successor of node m, stores the partial path cost of the path ending at it. We assign the cost found in (2) to the corresponding successor m_j and call it the cost (metric) of m_j, $g(m_j)$.
4. We call this process of applying the successor operator to a node *expanding* the node.

5. In PFSA, the next node to be expanded is one with the smallest cost on the list of all leaf nodes (stack OPEN) of the subtree constructed so far by the algorithm. Thus, stack OPEN must be kept ordered according to the costs of its nodes.
6. When the algorithm chooses to expand a goal node, it output the path associated with the cost of the goal node.
7. PFSA requires that for all nodes m_i and m_j such that node m_j is an immediate successor of node m_i,

$$g(m_i) \leq g(m_j) \tag{1}$$

   This requirement guarantees that PFSA will always find an optimal path.
8. Owing to above requirement , the cost of any node is non-decreasing along any path in a code tree.
9. When a goal node is chosen for expansion, all costs of nodes on

stack OPEN are greater than or equal to the cost of the goal node. Since all successors of any node on stack OPEN will have costs no less than that of the node, one will not find any path with smaller cost than that of the goal node.

Priority-First Search Algorithm (PFSA) for Code Tree (2)

Step 1: Load the Open Stack with the start node whose cost is assigned to be zero.

Step 2: Expanding the top node in the Open Stack. Delete the top node from the Open Stack.

Step 3: Insert the successors into the Open Stack, and reorder the Open Stack according to ascending costs.

Step 4: If the top node in the Open Stack is a goal node in the code tree, then the algorithm stops; otherwise go to Step 2.

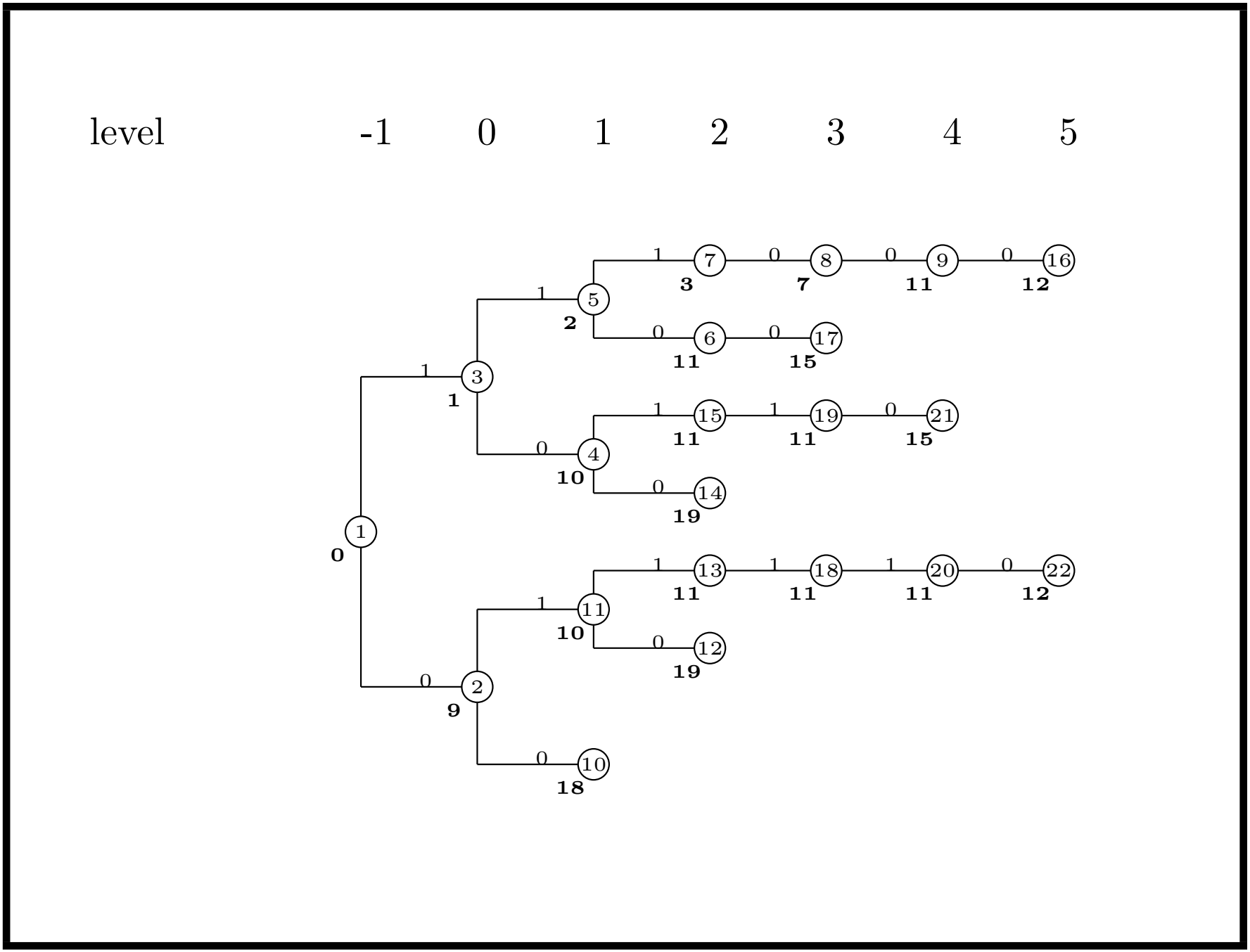
Priority-First Search Algorithm (PFSA) for Code Tree (3)

Let

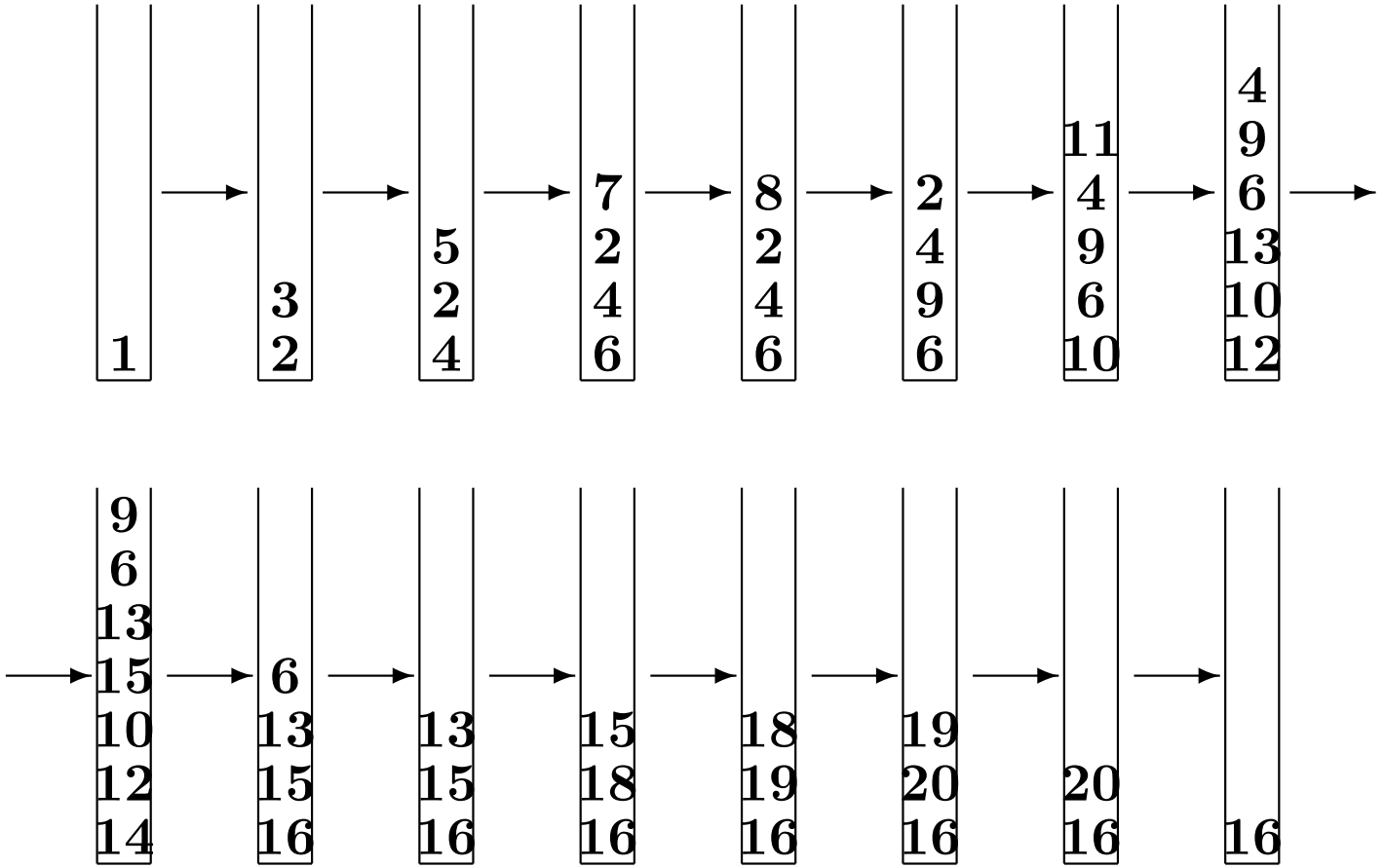
$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

be a generator matrix of $\boldsymbol{C}$. Assume the received vector is

$$\boldsymbol{\phi} = (-2, -2, -2, -1, -1, 0)$$


level -1 0 1 2 3 4 5
1
0
2
9
3
1
4
10
5
2
6
11
7
3
8
7
9
11
10
18
11
10
12
19
13
11
14
19
15
11
16
12
17
15
18
11
19
11
20
11
21
15
22
12

Priority-First Search Algorithm (PFSA) for Code Tree (4) – Stack Management



Generalized Dijkstra Algorithm (Algorithm A*) for Code Tree (1)

1. As pointed out before, PFSA associates to every node m visited by the value $g(m)$. This value can be treated as an estimate of the cost of the minimum cost path from the start node to node m.
2. In many graph search problems we may be able to obtain an estimate, $h(m)$, of the cost of the minimum cost path from node m to a goal node.
3. Thus $f(m) = g(m) + h(m)$ can be treated as an estimate of the minimum cost path from the start node to the goal node that goes through node m. We will assign $f(m)$ to be the cost of node m.
4. We will impose $h(\text{goal node}) = 0$ on the heuristic function since no estimation is necessary for a goal node.

5. The Generalized Dijkstra's Algorithm (GDA) uses $f(m)$ instead of $g(m)$ to guide the search through the code tree.
6. Function h is known as a heuristic function, and function f as an evaluation function [27].

Generalized Dijkstra Algorithm (Algorithm A*) for Code Tree (2)

1. In order to guarantee that GDA finds an optimal path, we impose the following condition on the heuristic function h.

   Condition. For all nodes m_i and m_j such that node m_j is an immediate successor of node m_i,

   $$h(m_i) \le h(m_j) + c(m_i, m_j), \tag{2}$$

   where $c(m_i, m_j)$ is the branch cost between node m_i and node m_j.

2.

$$\begin{aligned} & h(m_i) \le h(m_j) + c(m_i, m_j) \\ \Leftrightarrow \quad & g(m_i) + h(m_i) \le g(m_i) + h(m_j) + c(m_i, m_j) \\ \Leftrightarrow \quad & g(m_i) + h(m_i) \le g(m_j) + h(m_j) \end{aligned}$$

$$\Leftrightarrow \quad f(m_i) \leq f(m_j)$$

The condition guarantees the non-decreasing property on the path in the code tree. Thus when GDA selects to expand the goal node, it has already found an optimal path.

3. PFSA is the special case of GDA when the heuristic function is zero.

4. GDA is a particular case of Algorithm A^* that is widely used in Artificial Intelligence search problems [27]. Algorithm A^* finds an optimal path, but it imposes less restriction on the heuristic function h than we have imposed.

5. Algorithm A^* can be considered as a branch-and-bound type algorithm. In general, it is difficult to give any idea of how well a branch-and-bound algorithm will perform on a given problem. Nevertheless, the technique is sufficiently powerful that it is often used in practical applications [6].

Generalized Dijkstra Algorithm (Algorithm A*) for Code Tree (3)

The following results of GDA will be used in the design of the decoding algorithm.

Result 1: For every node m,

$$h(m) \leq h^*(m),$$

where $h^*(m)$ is the actual cost of a minimum cost path from node m to a goal node.

Result 2: If node m_i is selected for expansion, then $f(m_i) \leq f(m_j)$, where m_j is an immediate successor of node m_i.

Result 3: Let $\boldsymbol{P}$ be a path found by GDA from the start node to the goal node with cost UB. GDA still finds an optimal path if it removes from stack OPEN any node m for which

$f(m) \geq UB$.

Result 4: Let two functions

$$f_1(m) = g(m) + h_1(m)$$

and

$$f_2(m) = g(m) + h_2(m)$$

satisfy

$$h_1(m) \leq h_2(m)$$

for every non-goal node m. Furthermore, there exists a unique optimal path. Then the GDA, using evaluation function f_2, will never expend more nodes than the algorithm using evaluation function f_1.

The condition in Result 1 can be used as the condition guaranteeing the optimality of GDA.

Generalized Dijkstra Algorithm (Algorithm A*) for Code Tree (4)

1. From the description of GDA it is clear that the most important factor in the efficiency of GDA is the selection of the heuristic function h and, consequently, the evaluation function f.

2. Let m be a node at level ℓ. Define h_s as

$$h_s(m) = \sum_{j=\ell+1}^{n-1} \left(|\phi_j| - 1\right)^2 .$$

3. Since $\left(|\phi_i| - 1\right)^2 = \min\left\{(\phi_j - 1)^2, (\phi_j + 1)^2\right\}$, it is easy to show that h_s satisfies optimality condition of GDA.

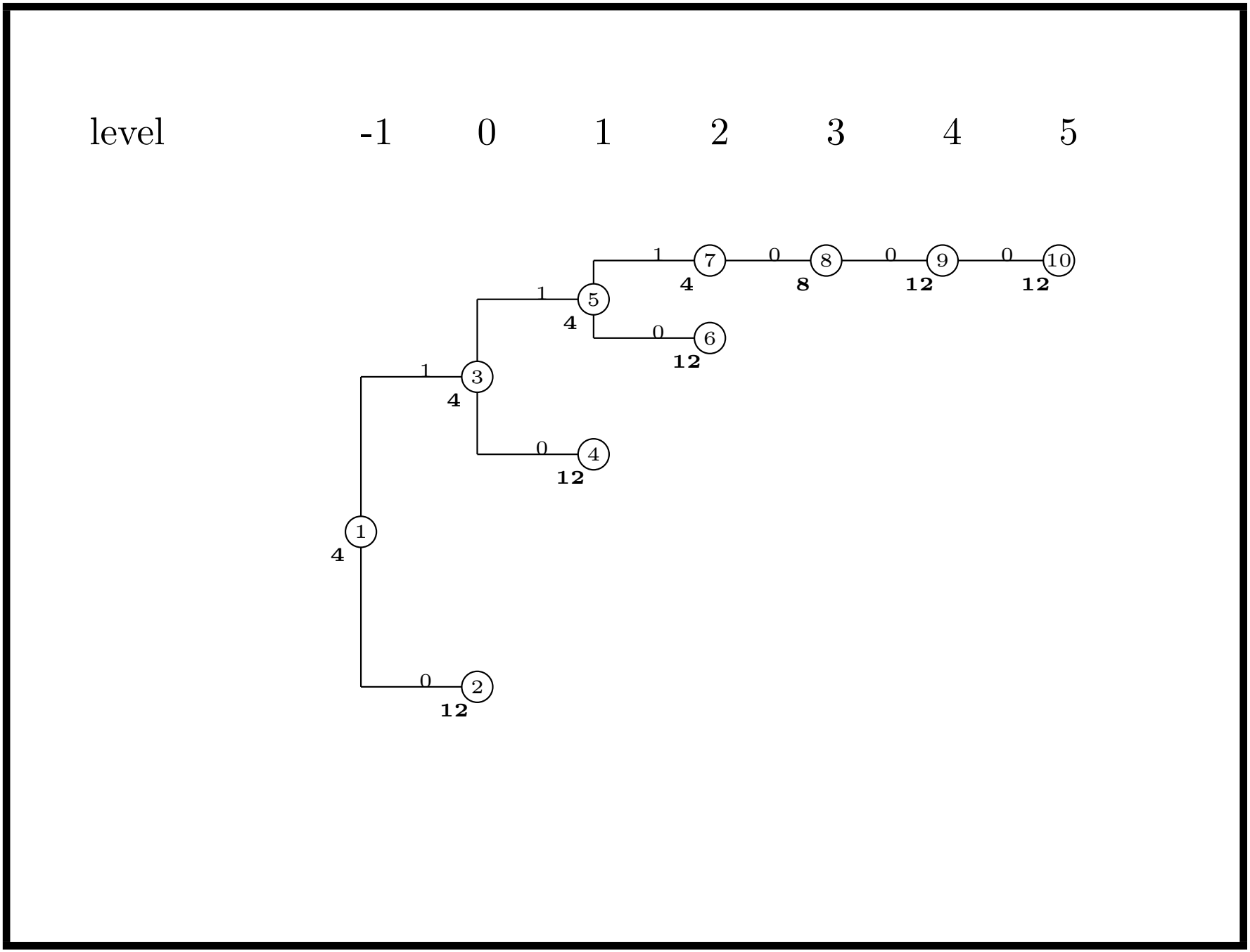
Generalized Dijkstra Algorithm (Algorithm A*) for Code Tree (5)

Let

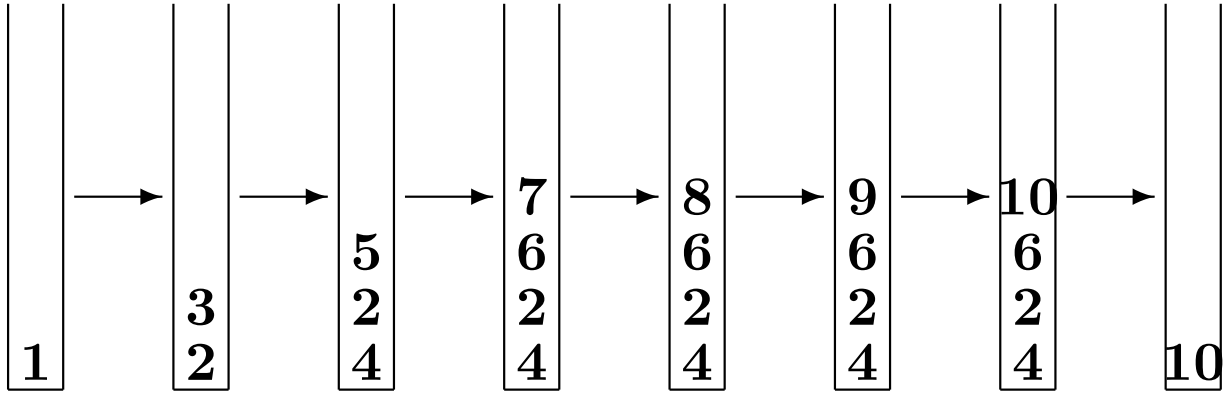
$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

be a generator matrix of $\boldsymbol{C}$. Assume the received vector is

$$\boldsymbol{\phi} = (-2, -2, -2, -1, -1, 0)$$


level -1 0 1 2 3 4 5
1
7
0
8
0
9
0
10
4
8
12
12
1
5
4
0
6
12
1
3
4
0
4
12
1
4
0
2
12

Generalized Dijkstra Algorithm (Algorithm A*) for Code Tree (6)– Stack Management



$$h_s(1) = 1 + 1 + 1 + 1 = 4, h_s(2) = h_s(3) = 3, h_s(4) = h_s(5) = 2$$

$$h_s(6) = h_s(7) = h_s(8) = h_s(9) = 1$$

Generalized Dijkstra Algorithm (Algorithm A*) for Code Tree (7)

1. Since d_D is also a distance metric for MLD rule, one may replace the branch metric $(\phi_t - (-1)^{c_t})^2$ with $(y_t \oplus c_t)|\phi_t|$ in the trellis or the code tree.
2. It has been show in [19] that GDA using the h_s defined before expands the same set of nodes as that using the new metric assignment and $h_{new}(m) = 0$ for any node m.
3. More complicate but more efficient heuristic functions were defined in [21, 20, 22].

Generalized Dijkstra Algorithm (Algorithm A*) for Code Tree (8)

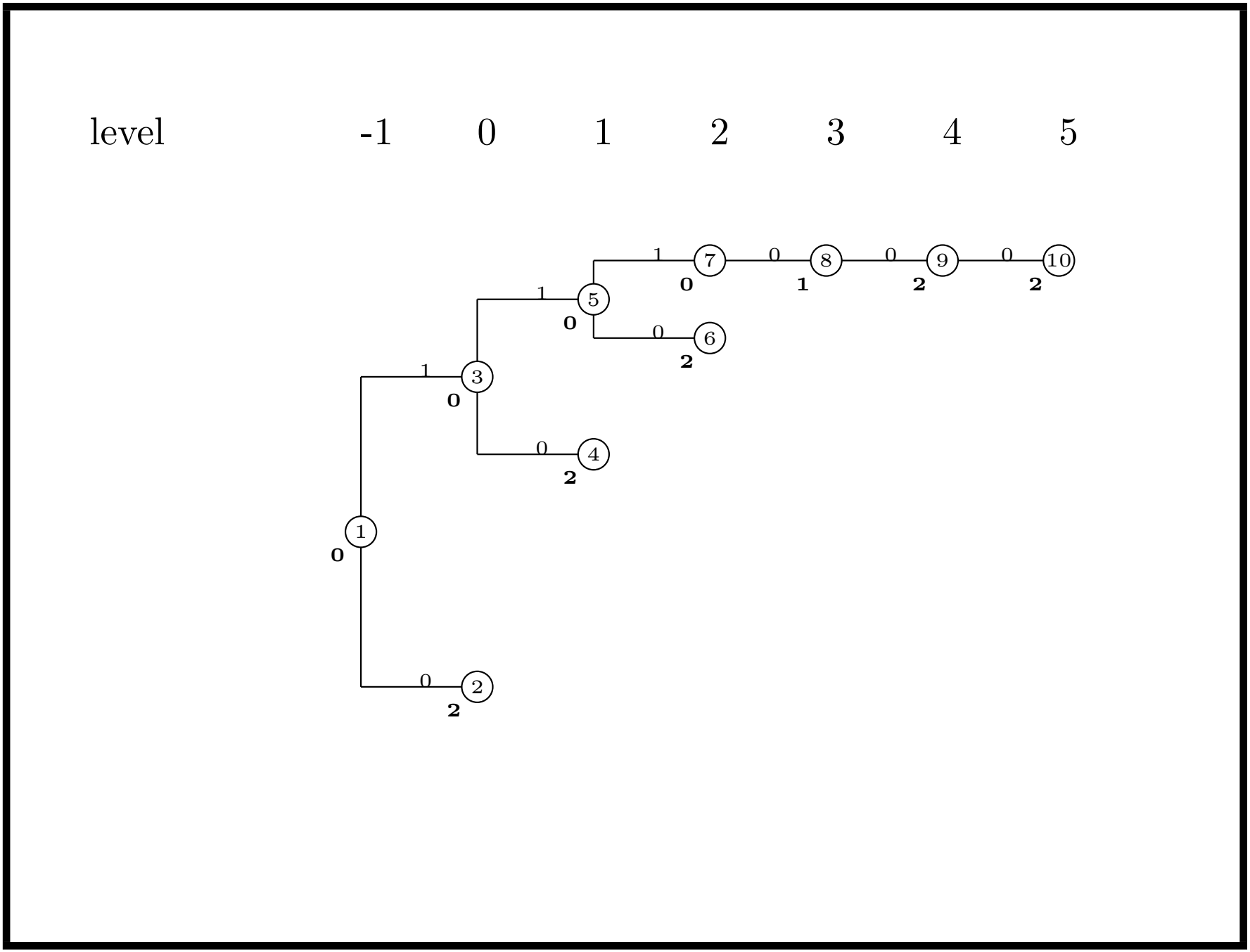
Let

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

be a generator matrix of $\boldsymbol{C}$. Assume the received vector is

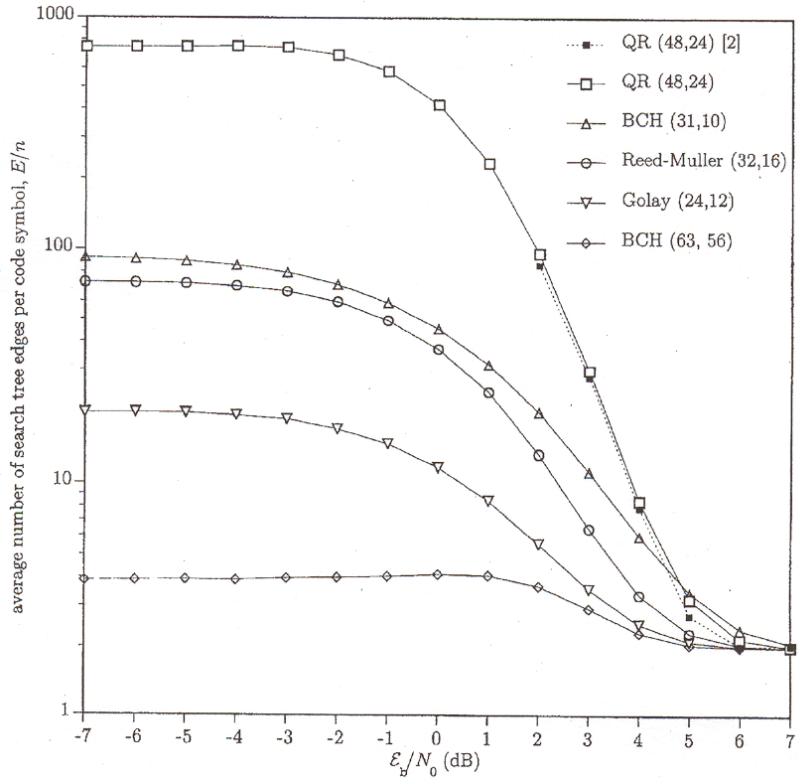
$$\boldsymbol{\phi} = (-2, -2, -2, -1, -1, 0)$$

The branch metric $(y_t \oplus c_t)|\phi_t|$ is assigned to the branch from a node at level $t-1$ to a node at level t, where c_t is the label of the branch. The heuristic function is 0.


level -1 0 1 2 3 4 5
1
2
3
4
5
6
7
8
9
10
0
1
2
0
1
2
2
0
2
0
2
2

Performance of Generalized Dijkstra Algorithm (Algorithm A*) for Code Tree (1)

The average number of search tree edges per code symbol [10]



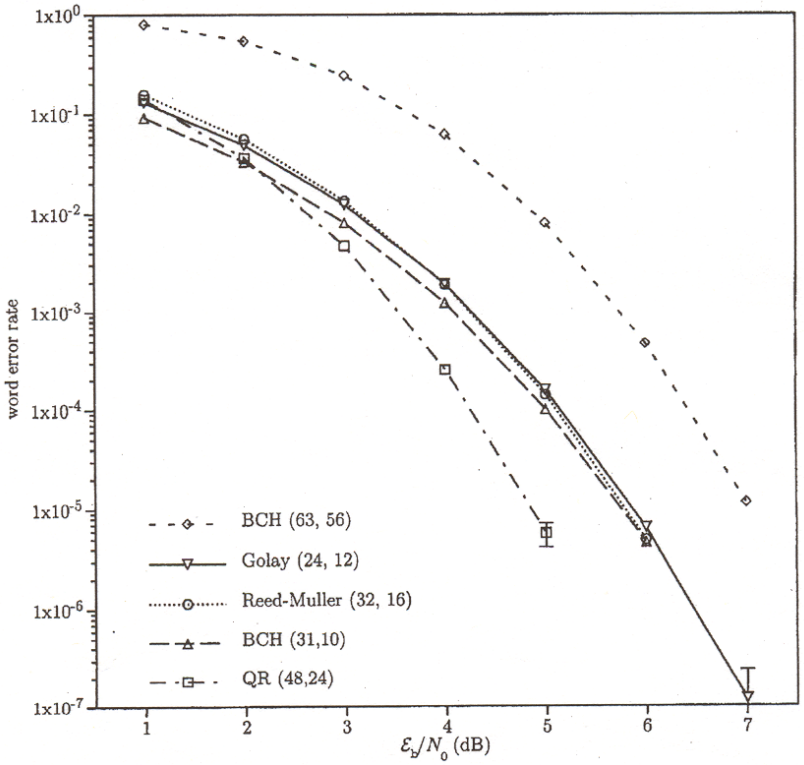
Performance of Generalized Dijkstra Algorithm (Algorithm A*) for Code Tree (2)

Comparison for decoding complexity for some maximum-likelihood decoding technique [10]

Number of edges for various maximum likelihood decoders	BCH (31, 10)	Golay (24, 12)	Reed-Muller (32, 16)	Quadratic residue (48, 24)	BCH (63, 56)
A* algorithm search tree at $+\infty$ dB (average $\pm$ standard dev.)	62 ± 0	48 ± 0	64 ± 0	96 ± 0	126 ± 0
A* algorithm search tree at $-\infty$ dB (average $\pm$ standard dev.)	2903 $\pm$1660	469 $\pm$327	2303 $\pm$1912	34,429 $\pm$31,940	245 $\pm$151
Minimal trellis for the best code permutation (fixed)	≤ 7068 ≥ 3484	3580	6396	860,156	≤ 5068 ≥ 4892
Minimal trellis for the worst code permutation (fixed)	15,356	16,380	262,140	67,108,860	13,052
Full code tree (fixed)	23,550	57,342	1,179,646	436,207,614	6.49×10^{17}
Exhaustive search (fixed)	31,744	98,304	2,097,152	805,306,368	4.54×10^{18}

Performance of Generalized Dijkstra Algorithm (Algorithm A*) for Code Tree (3)

Simulated word error rate [10]



Performance of Generalized Dijkstra Algorithm (Algorithm A*) for Code Tree (4)

The average and maximum number of nodes opened during the decoding of $(104, 52)$ code

γ_b	5 dB		6 dB		7 dB		8 dB		9 dB	
	max	ave	max	ave	max	ave	max	ave	max	ave
$N(\boldsymbol{\phi})$	$218,226$	915	$11,477$	194	1,376	115	95	62	57	54

$N(\boldsymbol{\phi})$= the number of nodes opened during the decoding of $\boldsymbol{\phi}$

The average and maximum number of nodes opened for Wolf algorithm is $2^{52} \approx 10^{15}$

Generalized Dijkstra Algorithm (Algorithm A*) for Trellis (1)

1. As in Wolf algorithm, a node in the trellis may be visited twice. Thus, for each visited node m, we store the path $\boldsymbol{P_m}$ from the start node to node m with minimum cost $g_\ell(m)$ found so far by the algorithm.
2. The value of $g_\ell(m)$ may be updated, since the minimum cost path from start node to node m may change as the search progresses.
3. A *successor operator* when applied to a node m, (1) gives all the immediate successors of node m; (2) for every immediate successor of node m, checks if such a node was visited before; (3) for every immediate successor of node m, stores the minimum cost path from the start node to this node and the cost, of this path found so far by the algorithm.

4. We call this process of applying the successor operator to a node *expanding* the node.
5. PFSA maintains two stacks of nodes of the given trellis, namely, stack CLOSED and stack OPEN.
6. Stack CLOSED contains the set of nodes that were expanded.
7. Stack OPEN contains the set of nodes that were visited, but not expanded.
8. The algorithm selects node m on stack OPEN with minimum $g_\ell(m)$. It expands this node and inserts it into stack CLOSED. It inserts into stack OPEN only the immediate successors of node m that have not been expanded before.
9. When the algorithm selects to expand the goal node it has found an optimal path.
10. When a node is selected for expansion, the algorithm has

already found a minimum cost path from the start node to this node [9] and, consequently, we do not need to update the lowest cost path from the start node to any node that is already on stack CLOSED.

11. when expanding node m, we do not need to update the lowest cost path from the start node to any descendant of an immediate successor of node m that is already on stack CLOSED.

12. The problem of determining whether a newly visited node is on stack OPEN or stack CLOSED can be computationally expensive, and we may therefore decide to avoid making this check, in which case the search tree may contain several repeated nodes, and stack CLOSED does not need to be maintained.

Generalized Dijkstra Algorithm (Algorithm A*) for Trellis (2)

Step 1: Load the Open Stack with the origin node whose metric $g_{-1}(m) = 0$.

Step 2: Expanding the top node in the Open Stack, and put the ending state and the ending level of this top node into the Closed Stack. Delete the top node from the Open Stack.

Step 3: Whenever any of the new nodes (i.e., the successors of the top node in the Open Stack in Step 2) merges with a node already in the Open Stack, eliminate the one with higher metric value. If any of the new nodes merges with a node already in the Closed Stack, discard the new node.

Step 4: Insert the remaining new nodes into the Open Stack, and reorder the Open Stack according to ascending metric values.

Step 5: If the top node in the Open Stack is the goal node in the trellis, the algorithm stops; otherwise go to Step 2.

References

[1] D. Agrawal and A. Vardy, Generalized Minimum Distance Decoding in Euclidean Space: Performance Analysis *IEEE Trans. Inform. Theory*, pp. 60–83, January 2000.

[2] L. E. Aguado and P. G. Farrell, On Hybrid Stack Decoding Algorithms for Block Codes *IEEE Trans. Inform. Theory*, pp. 398–409, January 1998.

[3] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate *IEEE Trans. Inform. Theory*, pp. 284–287, March 1974.

[4] G. Battail, Simplified Optimal Soft Decoding of Linear Block Codes *IEEE Int. Symp. on Information Theory*, St Jovite, Québac, Canada, 1983.

[5] M. Bossert, *Channel Coding for Telecommunications*. New York, NY: John Wiley and Sons, 1999.

[6] G. Brassard and P. Bratley, *Algorithmics Theory and Practice*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1988.

[7] D. Chase, A Class of Algorithms for Decoding Block Codes with Channel Measurement Information *IEEE Trans. Inform. Theory*, pp. 170–181, January 1972.

[8] J. H. Conway and N. J. A. Sloane, Soft Decoding Techniques for Codes and Lattices, Including the Golay Code and the Leech Lattice *IEEE Trans. Inform. Theory*, pp. 41–50, January 1986.

[9] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1991.

[10] L. Ekroot and S. Dolinar, A* Decoding of Block Codes *IEEE Trans. Commun.*, vol. 44, no. 9, pp. 1052–1056, September 1996.

[11] G. D. Forney, Jr., Generalized Minimum Distance Decoding *IEEE Trans. Inform. Theory*, pp. 125–131, April 1966.

[12] G. D. Forney, Jr., Coset Codes–Part II: Binary Lattices and Related Codes *IEEE Trans. Inform. Theory*, pp. 1152–1187, September 1988.

[13] G. D. Forney, Jr. and A. Vardy, Generalized Minimum-Distance Decoding of Euclidean-Space Codes and Lattices *IEEE Trans. Inform. Theory*, pp. 1992–2026, November 1996.

[14] M. P. C. Fossorier and S. Lin, Soft-Decision Decoding of

Linear Block Codes Based on Ordered Statistics *IEEE Trans. Inform. Theory*, vol. 41, no. 5, pp. 1379–1396, September 1995.

[15] M. P. C. Fossorier and S. Lin, Computationally Efficient Soft-Decision Decoding of Linear Block Codes Based on Ordered Statistics *IEEE Trans. Inform. Theory*, pp. 738–751, May 1996.

[16] M. P. C. Fossorier and S. Lin, A Unified Method for Evaluating the Error-Correction Radius of Reliability-Based Soft-Decision Algorithms for Linear Block Codes *IEEE Trans. Inform. Theory*, vol. 44, no. 2, pp. 691–700, March 1998.

[17] D. Gazelle and J. Snyders, Reliability-Based Code-Search Algorithm for Maximum-Likelihood Decoding of Block Codes *IEEE Trans. Inform. Theory*, pp. 239–249, January 1997.

[18] Y. S. Han, *Efficient Soft-Decision Decoding Algorithms for Linear Block Codes Using Algorithm A**, PhD thesis, School of Computer and Information Science, Syracuse University, Syracuse, NY 13244, 1993.

[19] Y. S. Han, A New Treatment of Priority-First Search Maximum-Likelihood Soft-Decision Decoding of Linear Block Codes *IEEE Trans. Inform. Theory*, vol. 44, no. 7, pp. 3091–3096, November 1998.

[20] Y. S. Han and C. R. P. Hartmann, Designing efficient maximum-likelihood soft-decision decoding algorithms for linear block codes using algorithm A^*, Technical Report SU-CIS-92-10, School of Computer and Information Science, Syracuse University, Syracuse, NY 13244, June 1992.

[21] Y. S. Han, C. R. P. Hartmann, and C.-C. Chen, Efficient

maximum-likelihood soft-decision decoding of linear block codes using algorithm A^*, Technical Report SU-CIS-91-42, School of Computer and Information Science, Syracuse University, Syracuse, NY 13244, December 1991.

[22] Y. S. Han, C. R. P. Hartmann, and C.-C. Chen, Efficient Priority-First Search Maximum-Likelihood Soft-Decision Decoding of Linear Block Codes *IEEE Trans. Inform. Theory*, vol. 39, no. 5, pp. 1514–1523, September 1993.

[23] Y. S. Han, C. R. P. Hartmann, and K. G. Mehrotra, Decoding Linear Block Codes Using a Priority-First Search: Performance Analysis and Suboptimal Version *IEEE Trans. Inform. Theory*, pp. 1233–1246, May 1998.

[24] T. Kaneko, T. Nishijima, and S. Hirasawa, An Improvement of Soft-Decision Maximum-Likelihood Decoding Algorithm Using

Hard-Decision Bounded-Distance Decoding *IEEE Trans. Inform. Theory*, pp. 1314–1319, July 1997.

[25] T. Kaneko, T. Nishijima, H. Inazumi, and S. Hirasawa, An Efficient Maximum-Likelihood Decoding Algorithm for Linear Block Codes with Algebraic Decoder *IEEE Trans. Inform. Theory*, pp. 320–327, March 1994.

[26] F. Kschischang and V. Sorokine, A Sequential Decoder for Linear Block Codes with a Variable Bias Term Metric *IEEE Trans. Inform. Theory*, pp. 410–411, January 1998.

[27] N. J. Nilsson, *Principle of Artificial Intelligence*. Palo Alto, CA: Tioga Publishing Co., 1980.

[28] B.-Z. Shen, K. K. Tzeng, and C. Wang, A Bounded-Distance Decoding Algorithm for Binary Linear Block Codes Achieving

the Minimum Effective Error Coefficient *IEEE Trans. Inform. Theory*, pp. 1987–1991, November 1996.

[29] C.-C. Shih, C. R. Wulff, C. R. P. Hartmann, and C. K. Mohan, Efficient Heuristic Search Algorithms for Soft-Decision Decoding of Linear Block Codes *IEEE Trans. Inform. Theory*, pp. 3023–3038, November 1998.

[30] D. J. Taipale and M. B. Pursley, An Improvement to Generalized-Minimum-Distance Decoding *IEEE Trans. Inform. Theory*, pp. 167–172, January 1991.

[31] A. J. Viterbi, Error bound for convolutional codes and an asymptotically optimum decoding algorithm *IEEE Trans. Inform. Theory*, vol. IT-13, no. 2, pp. 260–269, April 1967.

[32] J. K. Wolf, Efficient Maximum Likelihood Decoding of Linear

Block Codes Using a Trellis *IEEE Trans. Inform. Theory*, pp. 76–80, January 1978.