Sequential Decoding of Binary Convolutional Codes

Yunghsiang S. Han

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

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

General Description of Sequential Decoding Algorithm

1. As a consequence of minimizing the codeword estimate error subject to an equiprobable codeword prior, the MLD rule, upon receipt of a received vector $\boldsymbol{r} = (r_0, r_1, \ldots, r_{N-1})$, outputs the codeword $\boldsymbol{c}^* = (c_0^*, c_1^*, \ldots, c_{N-1}^*)$ satisfying

$$\sum_{j=0}^{N-1} \log_2 \Pr(r_j|c_j^*) \geq \sum_{j=0}^{N-1} \log_2 \Pr(r_j|c_j) \text{ for all } \boldsymbol{c} \in \boldsymbol{C}. \qquad (1)$$

2. A natural implication of (1) is that by simply letting $\sum_{j=n(\ell-1)}^{n\ell-1} \log_2 \Pr(r_j|c_j)$ be the metric associated with a branch labeled by $(c_{n(\ell-1)}, \ldots, c_{n\ell-1})$, the MLD rule becomes a search of the code path with maximum metric, where the metric of a path is defined as the sum of the individual metrics of the branches of which the path consists.

3. For a graph over which the Algorithm A searches, a link between the origin node and any node, either directly connected or indirectly connected through some intermediate nodes, is called a *path*. Suppose that a real-valued function $f(\cdot)$, often referred to as the *evaluation function*, is defined for every path in the graph G.

Algorithm A

Step 1. *Compute the associated f-function value of the single-node path that contains only the origin node. Insert the single-node path with its associated f-function value into the stack.*

Step 2. *Generate all immediate successor paths of the top path in the stack, and compute their f-function values. Delete the top path from the stack.*

Step 3. *If the graph G is a trellis, check whether these successor paths end at a node that belongs to a path that is already in the stack. Restated, check whether these successor paths merge with a path that is already in the stack. If it does, and the f-function value of the successor path exceeds the f-function value of the sub-path that traverses the same nodes as the merged path in the stack*

but ends at the merged node, redirect the merged path by replacing its sub-path with the successor path, and update the f-function value associated with the newly redirected path.[a] *Remove those successor paths that merge with some paths in the stack.*

(Note that if the graph G is a code tree, there is a unique path connecting the origin node to each node on the graph; hence, it is unnecessary to examine the path merging.)

Step 4. Insert the remaining successor paths into the stack, and reorder the stack in descending f-function values.

Step 5. If the top path in the stack ends at a terminal node in the graph G, the algorithm stops; otherwise go to Step 2.

[a]The redirect procedure is sometimes time-consuming, especially when the f-function value of a path is computed based on the branches the path traverses.

In principle, the *evaluation function* $f(\cdot)$ that guides the search of the Algorithm A is the sum of two parts, $g(\cdot)$ and $h(\cdot)$; both range over all possible paths in the graph. The first part, $g(\cdot)$, is simply a function of all the branches traversed by the path, while the second part, $h(\cdot)$, called the *heuristic function*, help to predict a future route from the end node of the current path to a terminal node.

Fano Metric

1. Since its discovery in 1963 [3], the Fano metric has become a typical path metric in sequential decoding.
2. Originally, the Fano metric was discovered through mass simulations, and was first used by Fano in his sequential decoding algorithm on a code tree [3].
3. For any path $\boldsymbol{v}_{(\ell n-1)}$ that ends at level ℓ on a code tree, the *Fano metric* is defined as:

$$M\left(\boldsymbol{v}_{(\ell n-1)}|\boldsymbol{r}_{(\ell n-1)}\right) = \sum_{j=0}^{\ell n-1} M(v_j|r_j).$$

4. $\boldsymbol{r} = (r_0, r_1, \ldots, r_{N-1})$ is the received vector, and

$$M(v_j|r_j) = \log_2[\Pr(r_j|v_j)/\Pr(r_j)] - R$$

is the *bit metric*.

5. The calculation of $\Pr(r_j)$ follows the convention that the code bits— 0 and 1—are transmitted with equal probability, i.e.,

$$\Pr(r_j) = \sum_{v_j \in \{0,1\}} \Pr(v_j) \Pr(r_j|v_j) = \frac{1}{2} \Pr(r_j|v_j = 0) + \frac{1}{2} \Pr(r_j|v_j = 1),$$

and $R = k/n$ is the code rate.

Example of Fano Metric

1. A hard-decision decoder with
   $\Pr\{r_j = 0|v_j = 1\} = \Pr\{r_j = 1|v_j = 0\} = p$ for $0 \le j \le N-1$
   (i.e., a memoryless BSC channel with crossover probability p),
   where $0 < p < 1/2$, will interpret the Fano metric for path
   $\boldsymbol{v}_{(\ell n-1)}$ as:

$$M(\boldsymbol{v}_{(\ell n-1)}|\boldsymbol{r}_{(\ell n-1)}) = \sum_{j=0}^{\ell n-1} \log_2 \Pr(r_j|v_j) + \ell n(1-R), \qquad (2)$$

   where

$$\log_2 \Pr(r_j|v_j) = \begin{cases} \log_2(1-p), & \text{for } r_j = v_j; \\ \log_2(p), & \text{for } r_j \neq v_j. \end{cases}$$

2. In terms of the Hamming distance, (2) can be re-written as

$$M\left(\boldsymbol{v}_{(\ell n-1)}|\boldsymbol{r}_{(\ell n-1)}\right) = -\alpha \cdot d_H(\boldsymbol{r}_{(\ell n-1)}, \boldsymbol{v}_{(\ell n-1)}) + \beta \cdot \ell, \qquad (3)$$

where $\alpha = -\log_2[p/(1-p)] > 0$, and $\beta = n[1 - R + \log_2(1-p)]$.

3. An immediate observation from (3) is that a larger Hamming distance between the path labels and the respective portion of the received vector corresponds to a smaller path metric.

4. This property guarantees that if no error exists in the received vector (i.e., the bits demodulated are exactly the bits transmitted), and $\beta > 0$ (or equivalently, $R < 1 + \log_2(1-p)$), then the path metric increases along the correct code path, and the path metric along any incorrect path is smaller than that of the equally long correct path. Such a property is essential for a metric to work properly with sequential decoding.

Interpretation of Fano Metric

1. Massey [16] proved that at any decoding stage, extending the path with the largest Fano metric in the stack minimizes the probability that the extending path does not belong to the optimal code path, and the usage of the Fano metric for sequential decoding is thus analytically justified.
2. However, making such a *locally* optimal decision at every decoding stage does not always guarantee the ultimate finding of the *globally* optimal code path in the sense of the MLD rule in (1). Hence, the error performance of sequential decoding with the Fano metric is in general a little inferior to that of the MLD-rule-based decoding algorithm.
3. A striking feature of the Fano metric is its dependence on the code rate R. Introducing the code rate into the Fano metric somehow reduces the complexity of the sequential decoding

algorithm.

4. Observe from (2) that the first term, $\sum_{j=0}^{\ell n-1} \log_2 \Pr(r_j|v_j)$, is the part that reflects the maximum-likelihood decision in (1), and the second term, $\ell n(1-R)$, is introduced as a bias to favor a longer path or specifically a path with larger ℓ, since a longer path is closer to the leaves of a code tree and thus is more likely to be part of the optimal code path.

5. When the code rate increases, the number of incorrect paths for a given output length increases.[a] Hence, the confidence on the currently examined path being part of the correct code path should be weaker. Therefore, the claim that longer paths are part of the optimal code path is weaker at higher code rates.

6. The Fano metric indeed mirrors the above intuitive observation

[a] Imagine that the number of branches that leave each node is 2^k, and increasing the code rate can be conceptually interpreted as increasing k subject to a fixed n for a (n, k, m) convolutional code.

by using a linear bias with respect to the code rate.

Generalization of Fano Metric

1. Universally adding a constant to the Fano metric of all paths does not change the sorting result at each stage of the sequential decoding algorithm (cf. *Step 4* of the Algorithm A).
2. By choosing the additive constant $\sum_{j=0}^{N-1} \log_2 \Pr(r_j)$, we have

$$M\left(\boldsymbol{v}_{(\ell n-1)}|\boldsymbol{r}_{(\ell n-1)}\right) + \sum_{j=0}^{N-1} \log_2 \Pr(r_j)$$

$$= \sum_{j=0}^{\ell n-1} [\log_2 \Pr(r_j|v_j) - R] + \sum_{j=\ell n}^{N-1} \log_2 \Pr(r_j), \qquad (4)$$

   for which the two terms of (4) can be immediately re-interpreted as the g-function and the h-function from the perspective of the Algorithm A.

3. As the g-function is now defined based on the branch metric

$\sum_{j=\ell(n-1)}^{\ell n-1}[\log_2 \Pr(r_j|v_j) - R]$, the Algorithm A becomes a search to find the code path $\boldsymbol{v}^*$ that satisfies

$$\sum_{j=0}^{N-1} \left[\log_2 \Pr(r_j|v_j^*) - R\right] \geq \sum_{j=0}^{N-1} [\log_2 \Pr(r_j|v_j) - R]$$

for all code path $\boldsymbol{v}$.

4. The above criterion is equivalent to the MLD rule. Consequently, the Fano path metric indeed implicitly uses $\sum_{j=\ell n}^{N-1} \log_2 \Pr(r_j)$ as a heuristic estimate of the upcoming metric from the end node of the current path to a terminal node.

5. By studying the effect of varying weights on the g-function (i.e., the cumulative metric sum that is already known) and

h-function (i.e., the estimate) we have:

$$f_\omega\left(\boldsymbol{v}_{(\ell n-1)}\right) = \omega \sum_{j=0}^{\ell n-1} [\log_2 \Pr(r_j|v_j) - R] + (1-\omega) \sum_{j=\ell n}^{N-1} \log_2 \Pr(r_j), \tag{5}$$

where $0 \le \omega \le 1$.

6. Subtracting a universal constant $(1-\omega)\sum_{j=0}^{N-1} \log_2 \Pr(r_j)$ from (5) gives the *generalized Fano metric* for sequential decoding as:

$$M_\omega\left(\boldsymbol{v}_{(\ell n-1)}|\boldsymbol{r}_{(\ell n-1)}\right) = \sum_{j=0}^{\ell n-1} \left(\log_2 \frac{\Pr(r_j|v_j)^\omega}{\Pr(r_j)^{1-\omega}} - \omega R\right). \tag{6}$$

7. When $\omega = 1/2$, the generalized Fano metric reduces to the Fano metric with a multiplicative constant, 1/2.

8. As ω is slightly below 1/2, which can be interpreted from (5) as the sequential search is guided more by the estimate on the upcoming metrics than by the known cumulative metric sum,

the number of metric computations reduces but the decoding failure probability grows.

9. When ω is closer to one, the decoding failure probability of sequential decoding tends to be lower; however, the computational complexity increases. In the extreme case, taking $\omega = 1$ makes the generalized Fano metric completely mirror the MLD metric in (1), and the sequential decoding becomes a maximum-likelihood (hence, optimal in decoding failure probability) decoding algorithm.

10. The work in [10] thereby concluded that an (implicit) heuristic estimate can be elaborately defined to reduce fairly the complexity of sequential decoding with a slight degradation in error performance.

Stack Algorithm

1. The stack algorithm or the ZJ algorithm was discovered by Zigangirov [18] and later independently by Jelinek [12] to search a code tree for the optimal codeword.
2. It is exactly the Algorithm A with g-function equal to the Fano metric and zero h-function.
3. Because a stack is involved in searching for the optimal codeword, the algorithm is called the *stack algorithm.*

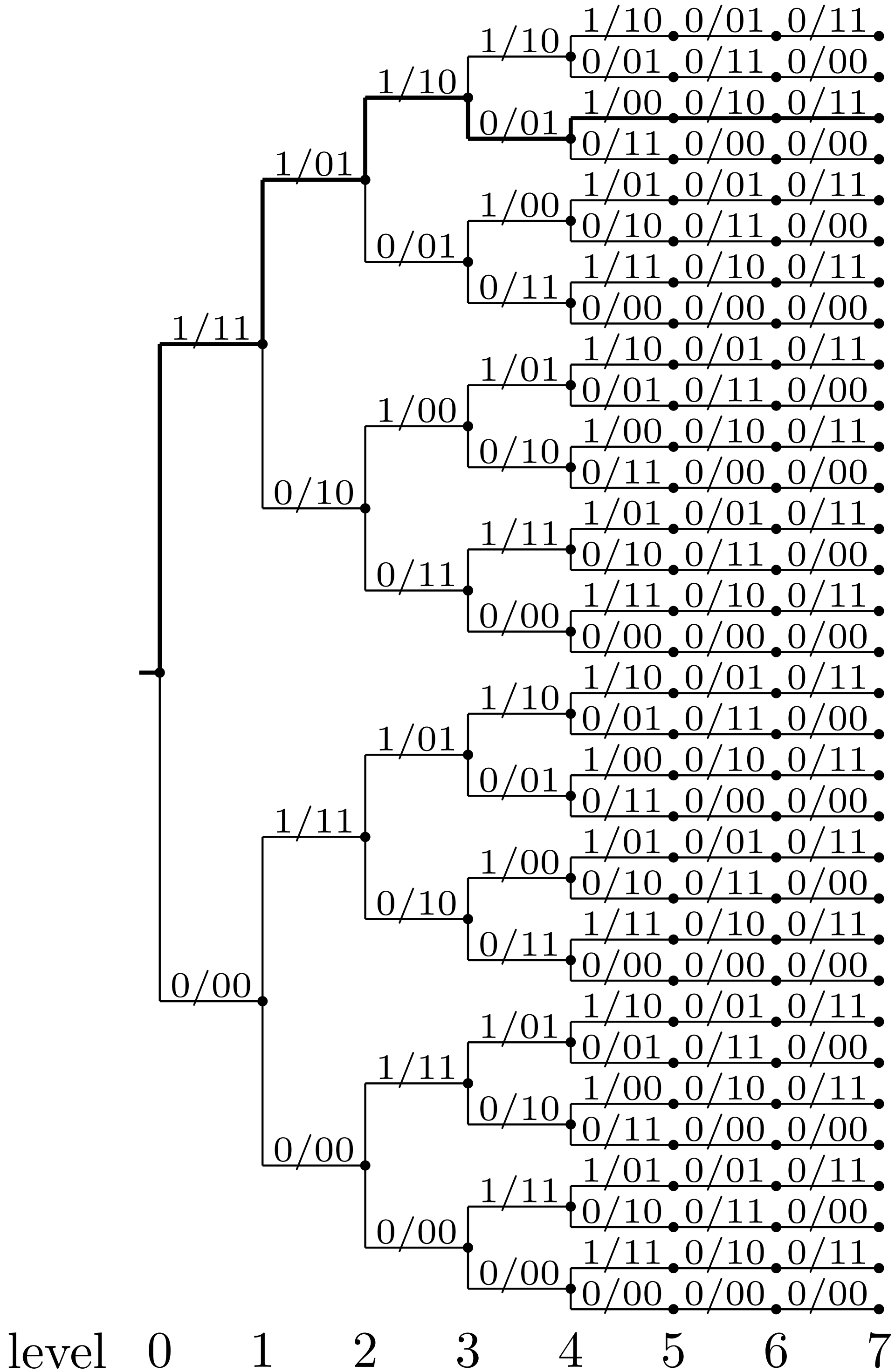
Example 1 *For a BSC with crossover probability* $p = 0.045$*, the Fano bit metric for a convolutional code with code rate* $R = 1/2$ *can be obtained from (2) as*

$$M(v_j|r_j) = \begin{cases} \log_2(1-p) + (1-R) = 0.434, & \text{for } r_j = v_j; \\ \log_2(p) + (1-R) = -3.974, & \text{for } r_j \neq v_j. \end{cases}$$

Consequently, only two Fano bit metric values are possible,

0.434 *and* -3.974. *These two Fano bit metric values can be "scaled" to equivalent "integers" to facilitate the simulation and implementation of the system. Taking the multiplicative scaling factor of* 2.30415 *yields*

$$M_{\text{scaled}}(v_j|r_j) = \begin{cases} 0.434 \times 2.30415 \approx 1, & \text{for } r_j = v_j; \\ -3.974 \times 2.30415 \approx -9, & \text{for } r_j \neq v_j. \end{cases}$$


1/10 0/01 0/11
1/10
0/01 0/11 0/00
1/10
1/00 0/10 0/11
0/01
0/11 0/00 0/00
1/01
1/01 0/01 0/11
1/00
0/10 0/11 0/00
0/01
1/11 0/10 0/11
0/11
0/00 0/00 0/00
1/11
1/10 0/01 0/11
1/01
0/01 0/11 0/00
1/00
1/00 0/10 0/11
0/10
0/11 0/00 0/00
0/10
1/01 0/01 0/11
1/11
0/10 0/11 0/00
0/11
1/11 0/10 0/11
0/00
0/00 0/00 0/00
1/10 0/01 0/11
1/10
0/01 0/11 0/00
1/01
1/00 0/10 0/11
0/01
0/11 0/00 0/00
1/11
1/01 0/01 0/11
1/00
0/10 0/11 0/00
0/10
1/11 0/10 0/11
0/11
0/00 0/00 0/00
0/00
1/10 0/01 0/11
1/01
0/01 0/11 0/00
1/11
1/00 0/10 0/11
0/10
0/11 0/00 0/00
0/00
1/01 0/01 0/11
1/11
0/10 0/11 0/00
0/00
1/11 0/10 0/11
0/00
0/00 0/00 0/00
level 0 1 2 3 4 5 6 7

loop 1	loop 2	loop 3	loop 4	loop 5
$1\,(1+1=2)$	$11\,(2+1+1=4)$	$111\,(4-9+1=-4)$	$1110\,(-4+1+1=-2)$	$110\,(-4)$
$0\,(-9-9=-18)$	$10\,(2-9-9=-16)$	$110\,(4+1-9=-4)$	$110\,(-4)$	$11100\,(-2+1-9=-1$
	$0\,(-18)$	$10\,(-16)$	$10\,(-16)$	$11101\,(-2-9+1=-1$
		$0\,(-18)$	$0\,(-18)$	$10\,(-16)$
			$1111\,(-4-9-9=-22)$	$0\,(-18)$
				$1111\,(-22)$

loop 6	loop 7	loop 8	loop 9
$11100\,(-10)$	$11101\,(-10)$	$111010\,(-10+1+1=-8)$	$1110100\,(-8+1+1=$
$11101\,(-10)$	$1100\,(-12)$	$1100\,(-12)$	$1100\,(-12)$
$1100\,(-4-9+1=-12)$	$1101\,(-12)$	$1101\,(-12)$	$1101\,(-12)$
$1101\,(-4+1-9=-12)$	$10\,(-16)$	$10\,(-16)$	$10\,(-16)$
$10\,(-16)$	$111000\,(-10-9+1=-18)$	$111000\,(-18)$	$111000\,(-18)$
$0\,(-18)$	$0\,(-18)$	$0\,(-18)$	$0\,(-18)$
$1111\,(-22)$	$1111\,(-22)$	$1111\,(-22)$	$1111\,(-22)$

Stack Implementation

1. In a straightforward implementation of the stack algorithm, the paths are stored in the stack in order of descending f-function values; hence, a sorting mechanism is required.
2. Another implementation issue of the stack algorithm is that the stack size in practice is often insufficient to accommodate the potentially large number of paths examined during the search process. The stack can therefore overflow.
3. A common way of compensating for a stack overflow is to simply discard the path with the smallest f-function value [15], since it is least likely to be the optimal code path. However, when the discarded path happens to be an early predecessor of the optimal code path, performance is degraded.
4. Jelinek proposed the so-called *stack-bucket technique* to reduce the sorting burden of the stack algorithm [12].

5. In his proposal, the range of possible path metric values is divided into a number of intervals with pre-specified, fixed spacing. For each interval, a separate storage space, a *bucket*, is allocated. The buckets are then placed in order of descending interval endpoint values. During decoding, the next path to be extended is always the top path of the first non-empty bucket, and every newly generated path is directly placed on top of the bucket in which interval the respective path metric lies.
6. The sorting burden is therefore removed by introducing the stack-buckets. The time taken to locate the next path no longer depends on the size of the stack, rather on the number of buckets, considerably reducing the time required by decoding.
7. Consequently, the stack-bucket technique was used extensively in the software implementation of the stack algorithm for applications in which the decoding time is precisely restricted [7, 9, 15]

8. The drawback of the stack-bucket technique is that the path with the best path metric may not be selected, resulting in degradation in performance.

9. A so-called *metric-first stack-bucket* implementation overcomes the drawback by sorting the top bucket when it is being accessed. However, Anderson and Mohan [1] indicated that the access time of the metric-first stack-buckets will increase at least to the order of $S^{1/3}$, where S is the total number of the paths ever generated.

10. Mohan and Anderson [17] suggested the adoption of a *balanced binary tree* data structure, such as an AVL tree [13], to implement the stack, offering the benefit that the access time of the stack becomes of order $\log_2(S)$, where S represents the momentary stack size.

11. Briefly, a balanced binary tree is a sorted structure with node

insertion and deletion schemes such that its depth is maintained equal to the logarithm of the total number of nodes in the tree, whenever possible.

12. As a result, inserting or deleting a path (which is now a node in the data structure of a balanced binary tree) in a stack of size S requires at most $\log_2(S)$ comparisons (that is, the number of times the memory is accessed). The balanced binary tree technique is indeed superior to the metric-first stack-bucket implementation, when the stack grows beyond certain size.

13. In 1994, a novel systolic priority queue, called the *Parallel Entry Systolic Priority Queue* (PESPQ), was proposed to replace the stack-buckets [14]. Although it does not arrange the paths in the queue in strict order, the systolic priority queue technique can identify the path with the largest path metric within a constant time.

14. This constant time was shown to be comparable to the time required to compute the metric of a new path. Experiments revealed that the PESPQ stack algorithm is several times faster than its stack-bucket counterpart.

15. Most importantly, the invention of the PESPQ technique has given a seemingly promising future to hardware implementation of the stack algorithm.

Fano Algorithm

1. The Fano algorithm is a sequential decoding algorithm that does not require a stack [3].
2. The Fano algorithm can only operate over a code tree because it cannot examine path merging.
3. At each decoding stage, the Fano algorithm retains the information regarding three paths: the current path, its immediate predecessor path, and one of its successor paths.
4. Based on this information, the Fano algorithm can move from the current path to either its immediate predecessor path or the selected successor path; hence, no stack is required for queuing all examined paths.
5. The movement of the Fano algorithm is guided by a dynamic threshold T that is an integer multiple of a fixed step size Δ.

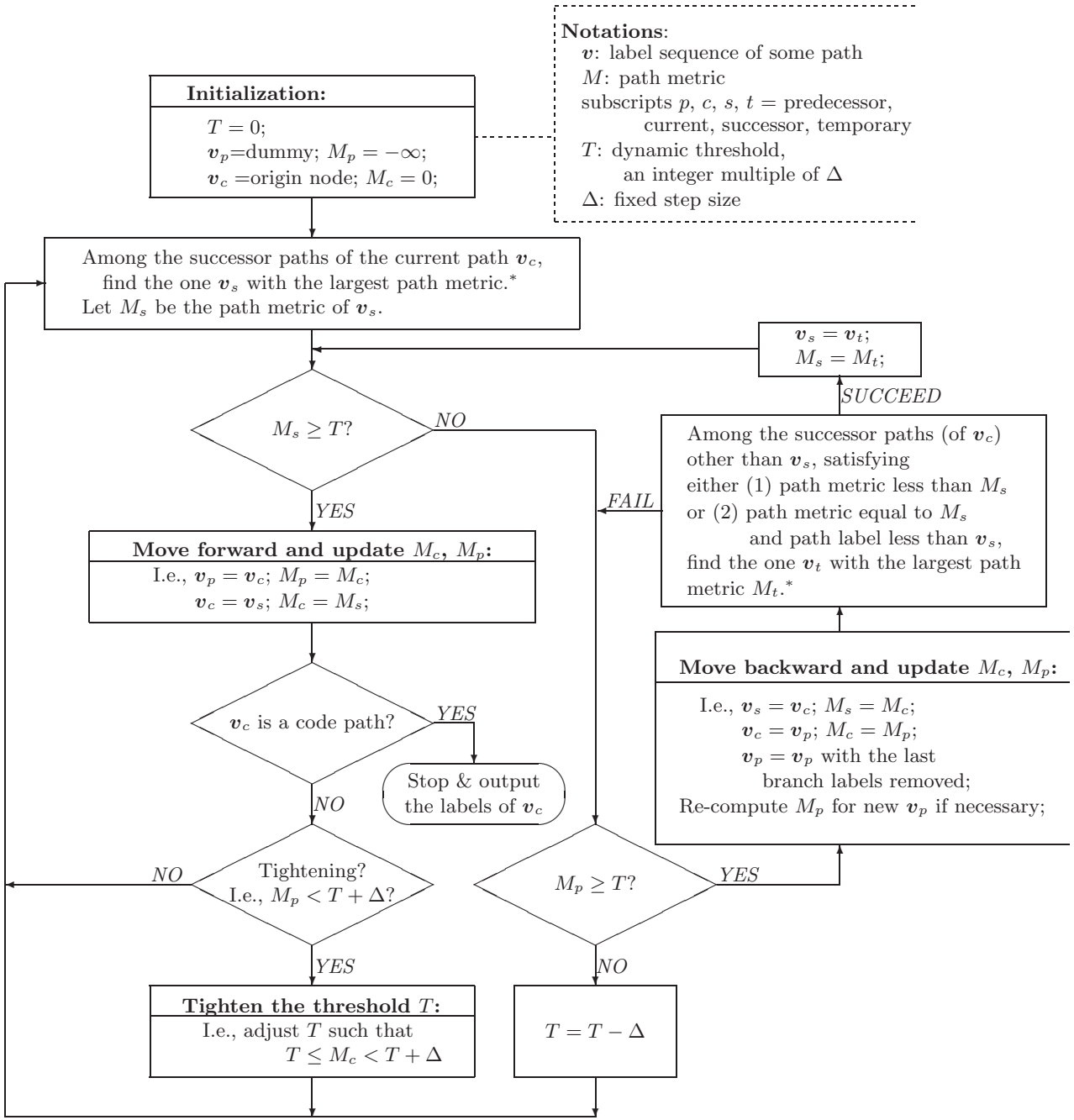
6. Only the path whose path metric is no less than T can be next visited. According to the algorithm, the process of codeword search continues to move forward along a code path, as long as the Fano metric along the code path remains non-decreasing.

7. Once all the successor path metrics are smaller than T, the algorithm moves backward to the predecessor path if the predecessor path metric beats T; thereafter, threshold examination will be subsequently performed on another successor path of this revisited predecessor.

8. In case the predecessor path metric is also less than T, the threshold T is one-step lowered so that the algorithm is not trapped on the current path.

9. For the Fano algorithm, if a path is revisited, the presently examined dynamic threshold is always lower than the momentary dynamic threshold at the previous visit,

guaranteeing that looping in the algorithm does not occur, and that the algorithm can ultimately reach a terminal node of the code tree, and stop.


Notations:
$\boldsymbol{v}$: label sequence of some path
M: path metric
subscripts p, c, s, t = predecessor, current, successor, temporary
T: dynamic threshold, an integer multiple of Δ
Δ: fixed step size
Initialization:
$T = 0$;
$\boldsymbol{v}_p$=dummy; $M_p = -\infty$;
$\boldsymbol{v}_c$ =origin node; $M_c = 0$;
Among the successor paths of the current path $\boldsymbol{v}_c$, find the one $\boldsymbol{v}_s$ with the largest path metric.*
Let M_s be the path metric of $\boldsymbol{v}_s$.
$\boldsymbol{v}_s = \boldsymbol{v}_t$;
$M_s = M_t$;
SUCCEED
$M_s \geq T$?
NO
YES
FAIL
Among the successor paths (of $\boldsymbol{v}_c$) other than $\boldsymbol{v}_s$, satisfying either (1) path metric less than M_s or (2) path metric equal to M_s and path label less than $\boldsymbol{v}_s$, find the one $\boldsymbol{v}_t$ with the largest path metric M_t.*
Move forward and update M_c, M_p:
I.e., $\boldsymbol{v}_p = \boldsymbol{v}_c$; $M_p = M_c$;
$\boldsymbol{v}_c = \boldsymbol{v}_s$; $M_c = M_s$;
Move backward and update M_c, M_p:
I.e., $\boldsymbol{v}_s = \boldsymbol{v}_c$; $M_s = M_c$;
$\boldsymbol{v}_c = \boldsymbol{v}_p$; $M_c = M_p$;
$\boldsymbol{v}_p = \boldsymbol{v}_p$ with the last branch labels removed;
Re-compute M_p for new $\boldsymbol{v}_p$ if necessary;
$\boldsymbol{v}_c$ is a code path?
YES
Stop & output the labels of $\boldsymbol{v}_c$
NO
NO
Tightening?
I.e., $M_p < T + \Delta$?
$M_p \geq T$?
YES
YES
NO
Tighten the threshold T:
I.e., adjust T such that $T \leq M_c < T + \Delta$
$T = T - \Delta$

Iteration	$\boldsymbol{v}_p$	$\boldsymbol{v}_c$	$\boldsymbol{v}_s$	M_p	M_c	M_s	T	Action
0	D	S	1	$-\infty$	0	2	0	MFTT
1	S	1	11	0	2	4	0	MFTT
2	1	11	111	2	4	-4	4	LT
3	1	11	111	2	4	-4	0	MBS
4	S	1	10	0	2	-16	0	MBS
5	D	S	0	$-\infty$	0	-18	0	LT
6	D	S	1	$-\infty$	0	2	-4	MF
7	S	1	11	0	2	4	-4	MF
8	1	11	111	2	4	-4	-4	MF
9	11	111	1110	4	-4	-2	-4	MFTT
10	111	1110	11100	-4	-2	-10	-4	MBS
11	11	111	1111	4	-4	-22	-4	MBS
12	1	11	110	2	4	-4	-4	MF
13	11	110	1100	4	-4	-12	-4	MBF
14	1	11	110	2	4	-4	-4	MBS
15	S	1	10	0	2	-16	-4	MBS
16	D	S	0	$-\infty$	0	-18	-4	LT
17	D	S	1	$-\infty$	0	2	-8	MF
18	S	1	11	0	2	4	-8	MF
19	1	11	111	2	4	-4	-8	MF
20	11	111	1110	4	-4	-2	-8	MF
21	111	1110	11100	-4	-2	-10	-8	MBS
22	11	111	1111	4	-4	-22	-8	MBS
23	1	11	110	2	4	-4	-8	MF
24	11	110	1100	4	-4	-12	-8	MBF
25	1	11	110	2	4	-4	-8	MBS
26	S	1	10	0	2	-16	-8	MBS
27	D	S	0	$-\infty$	0	-18	-8	LT
28	D	S	1	$-\infty$	0	2	-12	MF
29	S	1	11	0	2	4	-12	MF
30	1	11	111	2	4	-4	-12	MF
31	11	111	1110	4	-4	-2	-12	MF
32	111	1110	11100	-4	-2	-10	-12	MF
33	1110	11100	111000	-2	-10	-18	-12	MBS
34	111	1110	11101	-4	-2	-10	-12	MF
35	1110	11101	111010	-2	-10	-8	-12	MFTT
36	11101	111010	1110100	-10	-8	-6	-8	Stop

Step Size of Fano algorithm

1. As stated in [15], the load of branch metric computations executed during the finding of a qualified successor path becomes heavy when Δ is too small.
2. When Δ is too large, the dynamic threshold T might be lowered too much in a single adjustment, forcing an increase in both the decoding error and the computational effort.
3. Experiments suggest [15] that a better performance can be obtained by taking Δ within 2 and 8 for the unscaled Fano metric (Δ should be analogously scaled if a scaled Fano metric is used).

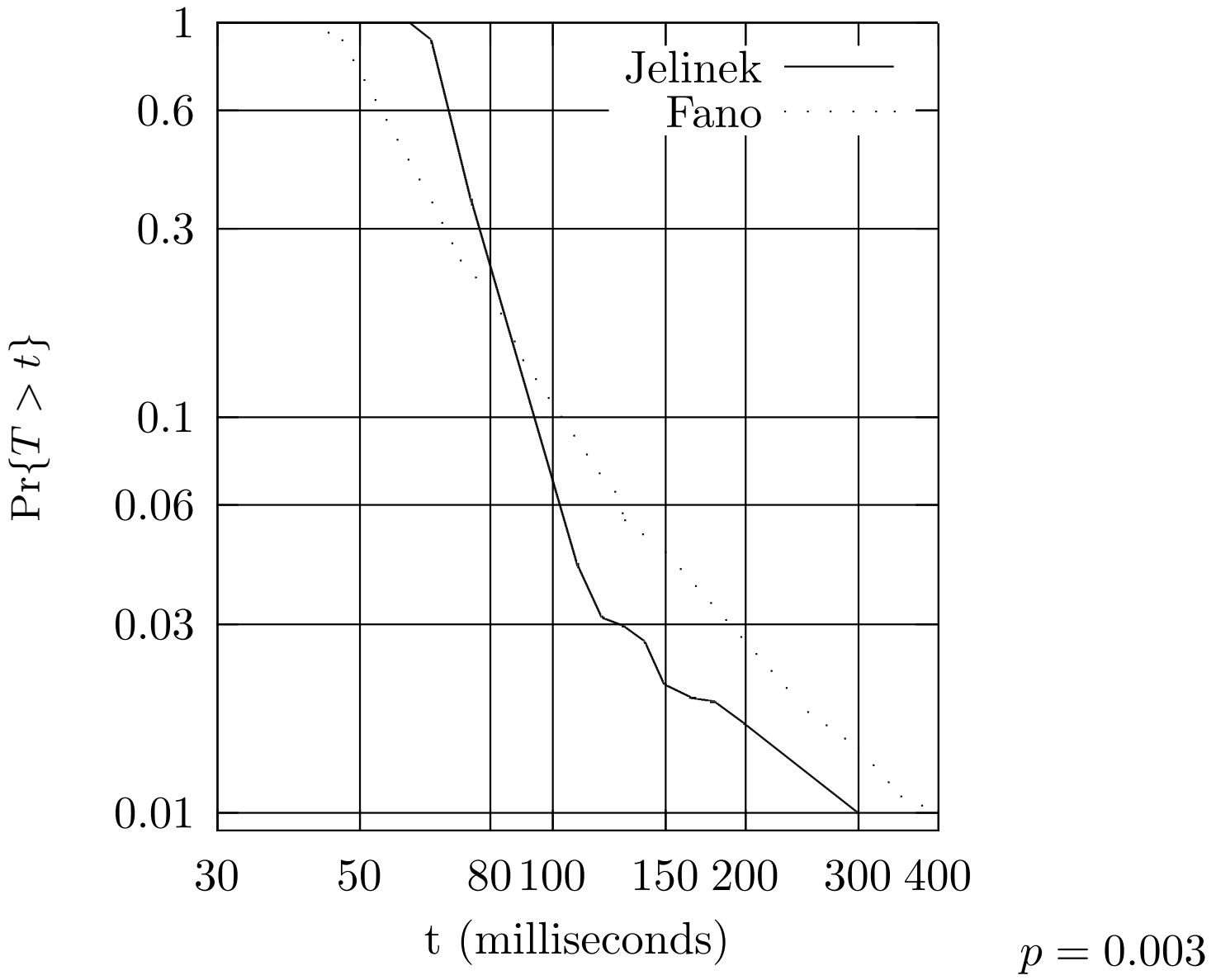
Comparison of Stack and Fano algorithms

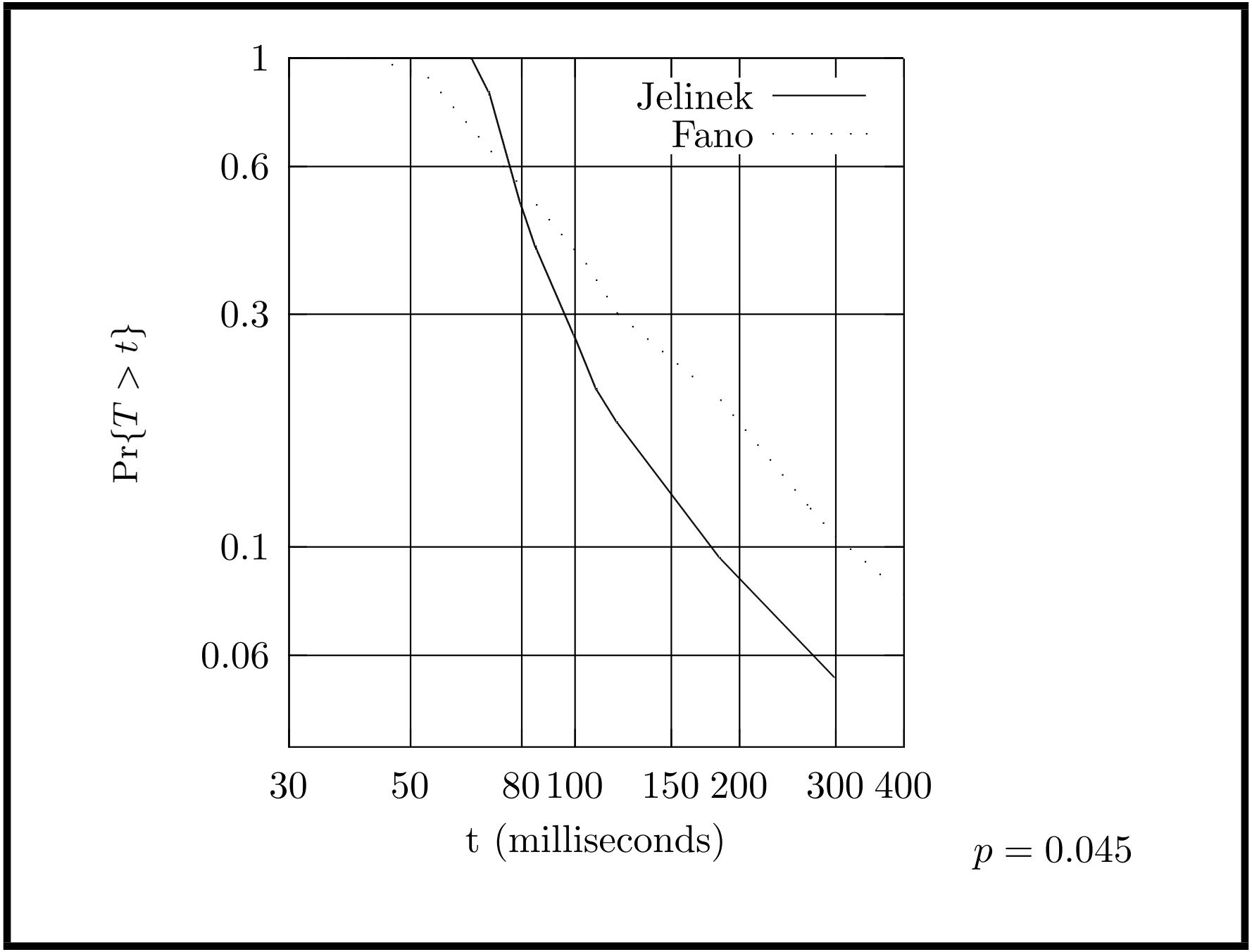
1. The Fano algorithm, perhaps surprisingly, while quite different in its design from the stack algorithm, exhibits broadly the same searching behavior as the stack algorithm.
2. In fact, with a slight modification to the update procedure of the dynamic threshold, both algorithms have been proven to visit almost the same set of paths during the decoding process [8].
3. Their only dissimilarity is that unlike the stack algorithm which visits each path only once, the Fano algorithm may revisit a path several times, and thus has a higher computational complexity.
4. The next three figures show comparisons of computational complexities of the Fano algorithm and the stack algorithm (with stack-bucket modification) based on the $(2, 1, 35)$

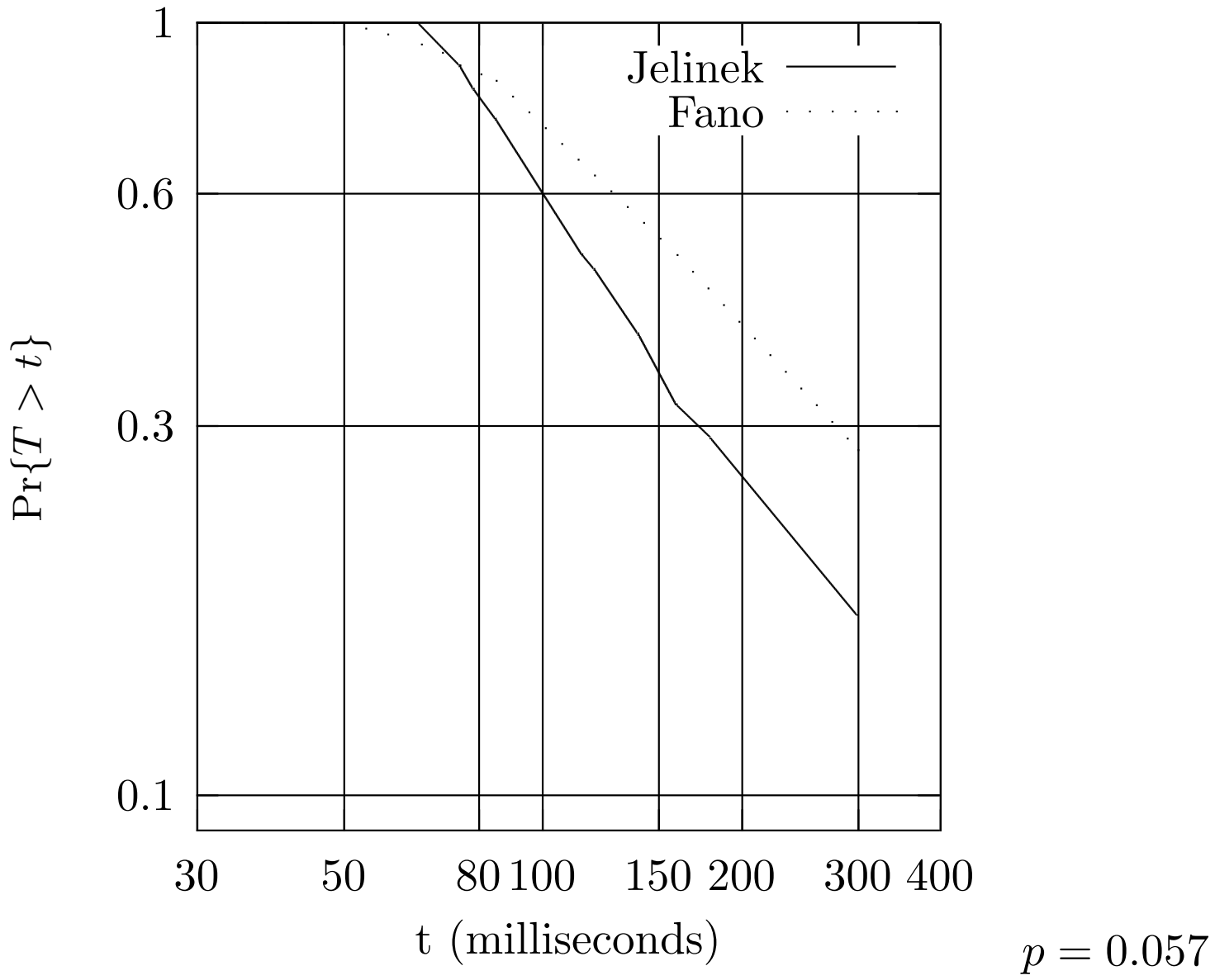
convolutional code and a single input sequence of length 256.

5. Simulations are performed over the binary symmetric channel with crossover probability p. $\Pr\{T \geq t\}$ is the empirical complement cumulative distribution function for the software computation time T.

6. In simulations, $(\log_2[2(1-p)] - 1/2, \log_2(2p) - 1/2)$, which is derived from the Fano metric formula, is scaled to $(2, -18)$, $(2, -16)$ and $(4, -35)$ for $p = 0.033$, $p = 0.045$ and $p = 0.057$, respectively.

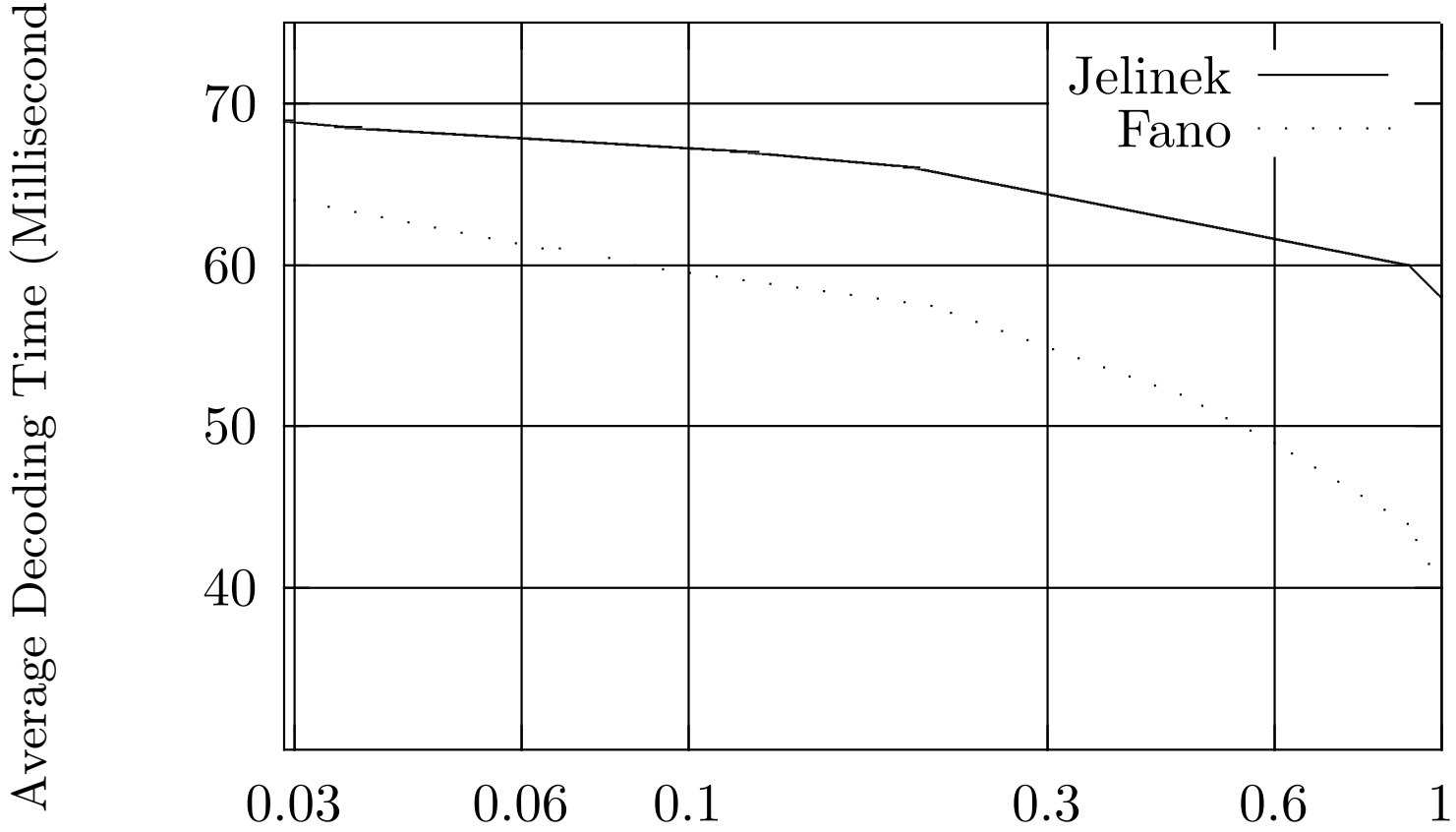

1
0.6
0.3
0.1
0.06
0.03
0.01
$\Pr\{T > t\}$
Jelinek
Fano
30
50
80
100
150
200
300
400
t (milliseconds)
$p = 0.003$


1
0.6
0.3
0.1
0.06
Pr{T > t}
Jelinek
Fano
30
50
80
100
150
200
300
400
t (milliseconds)
p = 0.045

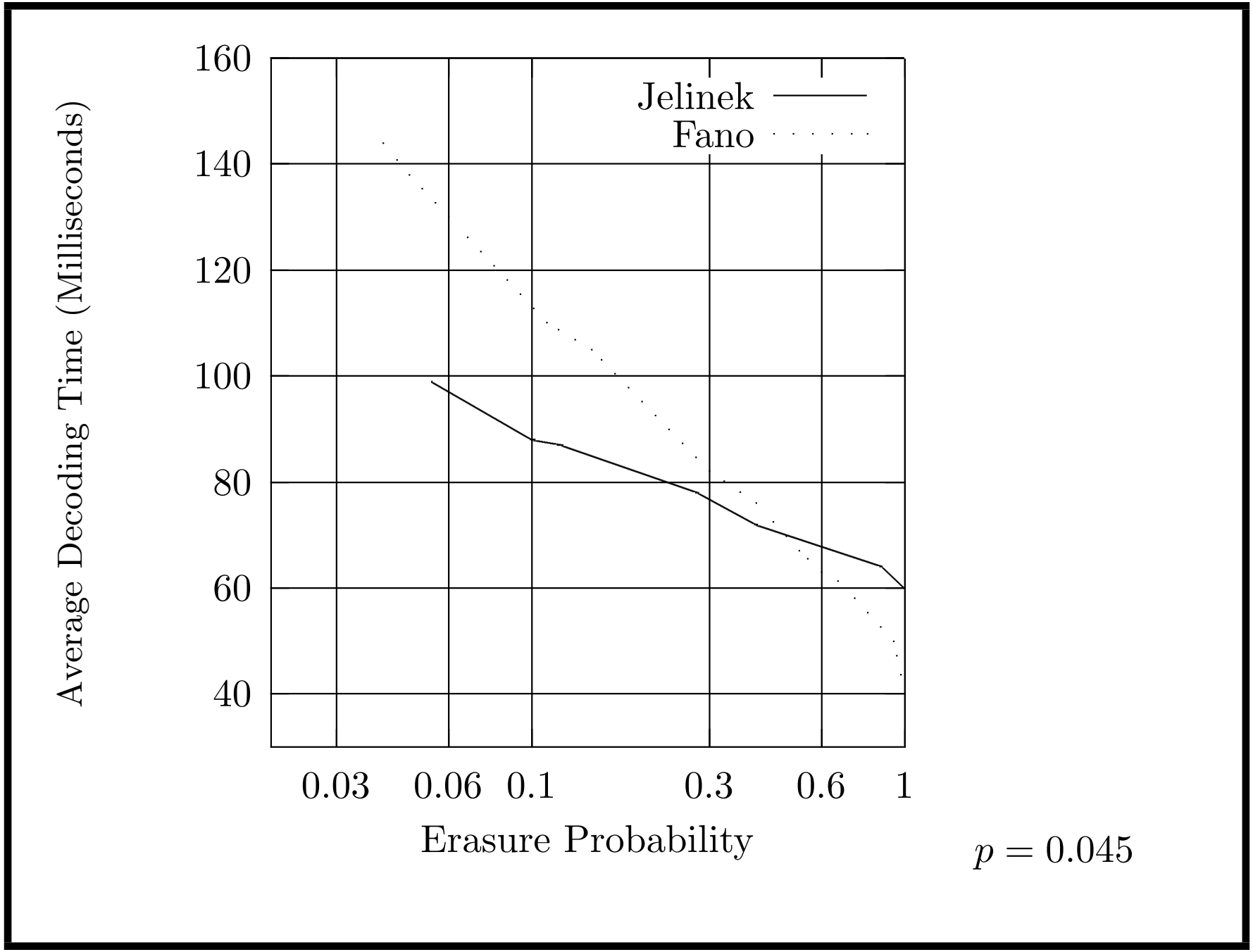

1
0.6
0.3
0.1
$\Pr\{T > t\}$
Jelinek
Fano
30
50
80
100
150
200
300
400
t (milliseconds)
$p = 0.057$

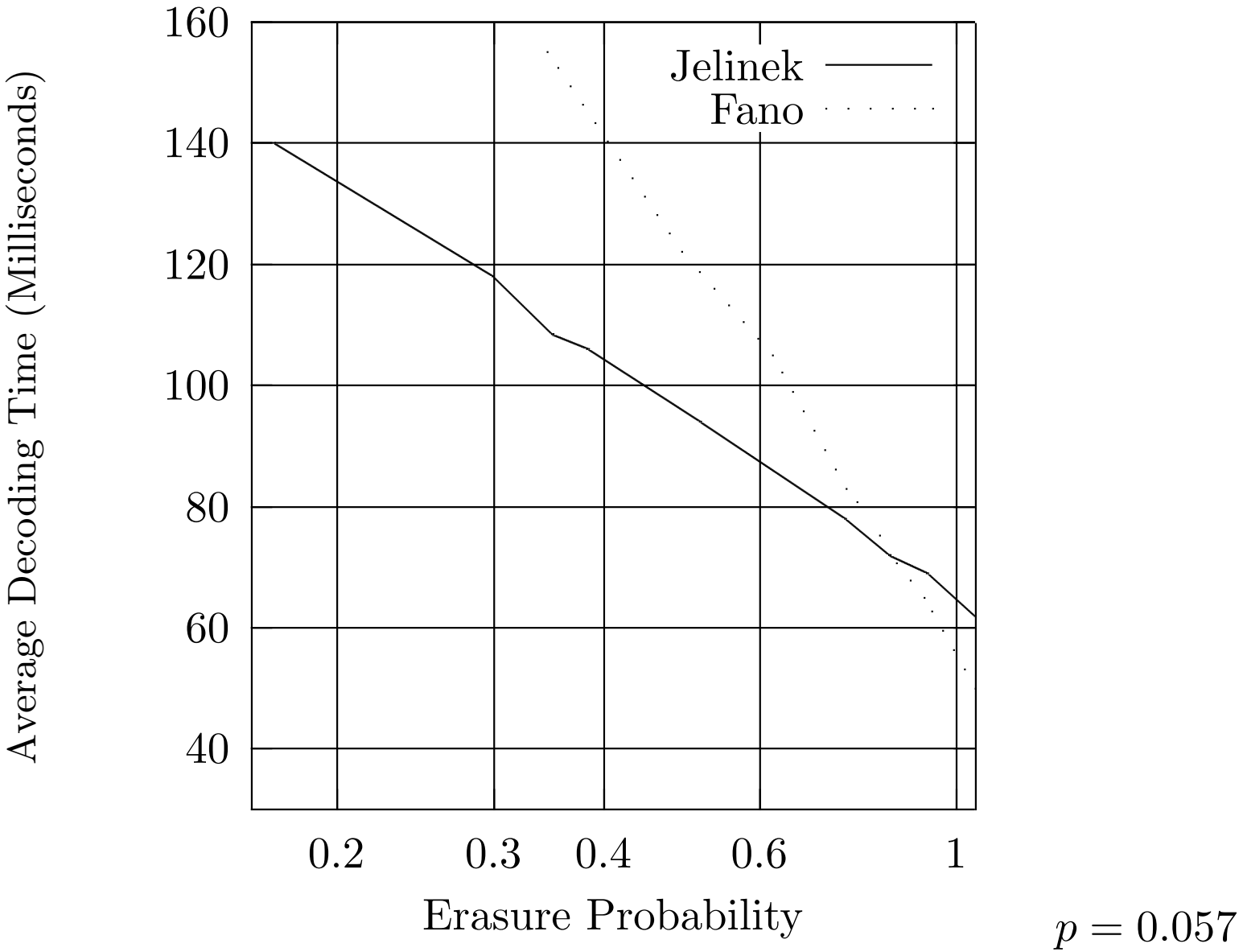
Erasure Probability

1. In practice, the time taken to decode an input sequence often has an upper limit. If the decoding process is not completed before the time limit, the undecoded part of the input sequence must be aborted or erased.
2. Hence, the probability of input erasure is another system requirement for sequential decoding.
3. The next three figures confirm that the stack algorithm with stack-bucket modification remains faster than the Fano algorithm except when either admitting a high erasure probability (for example, erasure probability > 0.5 for $p = 0.045$, and erasure probability > 0.9 for $p = 0.057$) or experiencing a less noisier channel (for example, $p = 0.033$) [7].


Average Decoding Time (Milliseconds)
70
60
50
40
Jelinek
Fano
0.03
0.06
0.1
0.3
0.6
1
Erasure Probability

$p = 0.033$


Average Decoding Time (Milliseconds)
160
140
120
100
80
60
40
Jelinek
Fano
0.03
0.06
0.1
0.3
0.6
1
Erasure Probability
$p = 0.045$


Average Decoding Time (Milliseconds)
160
140
120
100
80
60
40
Jelinek
Fano
0.2
0.3
0.4
0.6
1
Erasure Probability
$p = 0.057$

Hardware Implementation

1. An evident drawback of the stack algorithm in comparison with the Fano algorithm is its demand for an extra stack space. However, with recent advances in computer technology, a large memory requirement is no longer a restriction for software implementation.
2. In hardware implementation, stack maintenance normally requires accessing external memory a certain number of times, which usually bottlenecks the system performance.
3. Furthermore, the hardware is renowned for its efficient adaptation to a big number of computations. These hardware implementation features apparently favor the no-stack Fano algorithm, even when the number of its computations required is larger than the stack algorithm.
4. In fact, a hard-decision version of the Fano algorithm has been

hardware-implemented, and can operate at a data rate of 5 Mbits/second [5]. The prototype employs a systematic convolutional code to compensate for the input erasures so that whenever the pre-specified decoding time expires, the remaining undecoded binary demodulator outputs that directly correspond to the input sequences are immediately outputted.

Generalized Stack Algorithm

1. Sequential decoding was mostly operated over a code tree in early publications, although some early published work already hinted at the possibility of conducting sequential decoding over a trellis [6, 4].
2. The first feasible algorithm that sequentially searches a trellis for the optimal codeword is the *generalized stack algorithm* [9].
3. The generalized stack algorithm simultaneously extends the top M paths in the stack. It then examines, according to the trellis structure, whether any of the extended paths merge with a path that is already in the stack. If so, the algorithm deletes the newly generated path after ensuring that its path metric is smaller than the cumulative path metric of the merged path up to the merged node.
4. No redirection on the merged path is performed, even if the

path metric of the newly generated path exceeds the path metric of the sub-path that traverses along the merged path, and ends at the merged node. The newly generated path and the merged path may coexist in the stack.

5. The generalized stack algorithm, although it generally yields a larger average computational complexity than the stack algorithm, has lower variability in computational complexity and a smaller probability of decoding error [9].

6. The main obstacle in implementing the generalized stack algorithm by hardware is the maintenance of the stack for the simultaneously extended M paths.

7. One feasible solution is to employ M independent stacks, each of which is separately maintained by a processor [2].

8. In such a multiprocessor architecture, only one path extraction and two path insertions are sufficient for each stack in a

decoding cycle of a $(2, 1, m)$ convolutional code [2].

9. Simulations have shown that this multiprocessor counterpart not only retained the low variability in computational complexity as the original generalized stack algorithm, but also had a smaller average decoding time.

Maximum-Likelihood Trellis-Based Sequential Decoding

1. When the trellis-based generalized stack algorithm simultaneously extends 2^K most likely paths in the stack (that is, $M = 2^K$), where $K = \sum_{j=1}^{k} K_j$ and K_j is the length of the jth shift register in the convolutional code encoder, the algorithm becomes the maximum-likelihood Viterbi decoding algorithm.
2. The optimal codeword is thereby sought by exhausting all possibilities, and no computational complexity gain can be obtained at a lower noise level.
3. Han, et al. [11] recently proposed a true noise-level-adaptable trellis-based maximum-likelihood sequential decoder, called *maximum-likelihood soft-decision decoding algorithm* (MLSDA).
4. The MLSDA adopts a new metric, other than the Fano metric, to guide its sequential search over a trellis for the optimal code

path, which is now the code path with the minimum path metric.

5. The new path metric associated with a path $\boldsymbol{v}_{(\ell n-1)}$ is given by

$$M_{\mathrm{ML}}\left(\boldsymbol{v}_{(\ell n-1)}|\boldsymbol{r}_{(\ell n-1)}\right) = \sum_{j=0}^{\ell n-1} M_{\mathrm{ML}}(v_j|r_j), \tag{7}$$

where $M_{\mathrm{ML}}(v_j|r_j) = (y_j \oplus v_j) \times |\phi_j|$ is the jth bit metric, $\boldsymbol{r}$ is the received vector, $\phi_j = \ln[\Pr(r_j|0)/\Pr(r_j|1)]$ is the jth log-likelihood ratio, and

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

is the hard-decision output due to ϕ_j.

6. For AWGN channels, the ML bit metric can be simplified to

$M_{\mathrm{ML}}(v_j|r_j) = (y_j \oplus v_j) \times |r_j|$, where

$$y_j = \begin{cases} 1, & \text{if } r_j < 0; \\ 0, & \text{otherwise.} \end{cases}$$

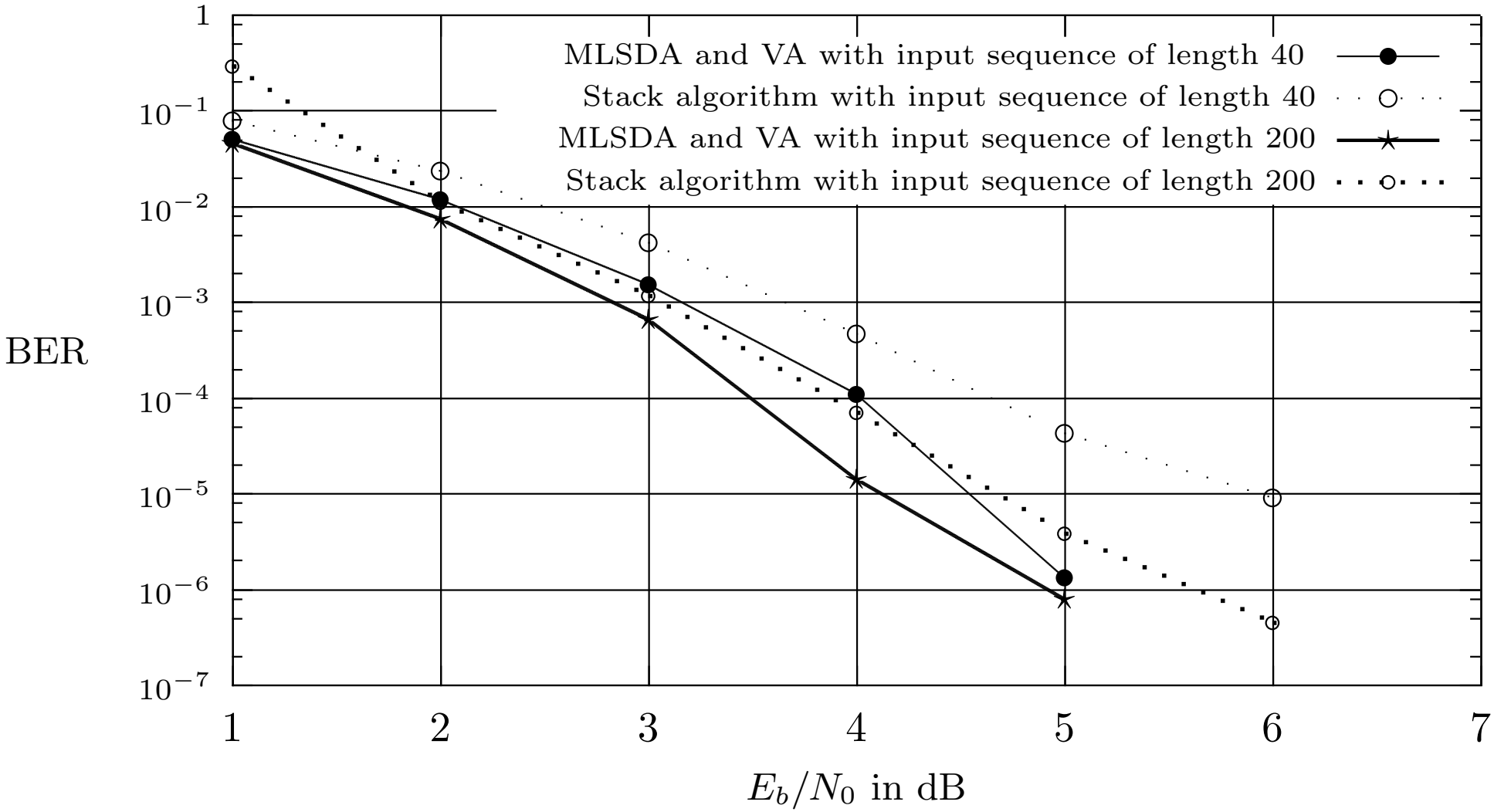
7. The MLSDA genuinely redirects and merges any two paths that share a common node, resulting in a stack without coexistence of crossed paths.

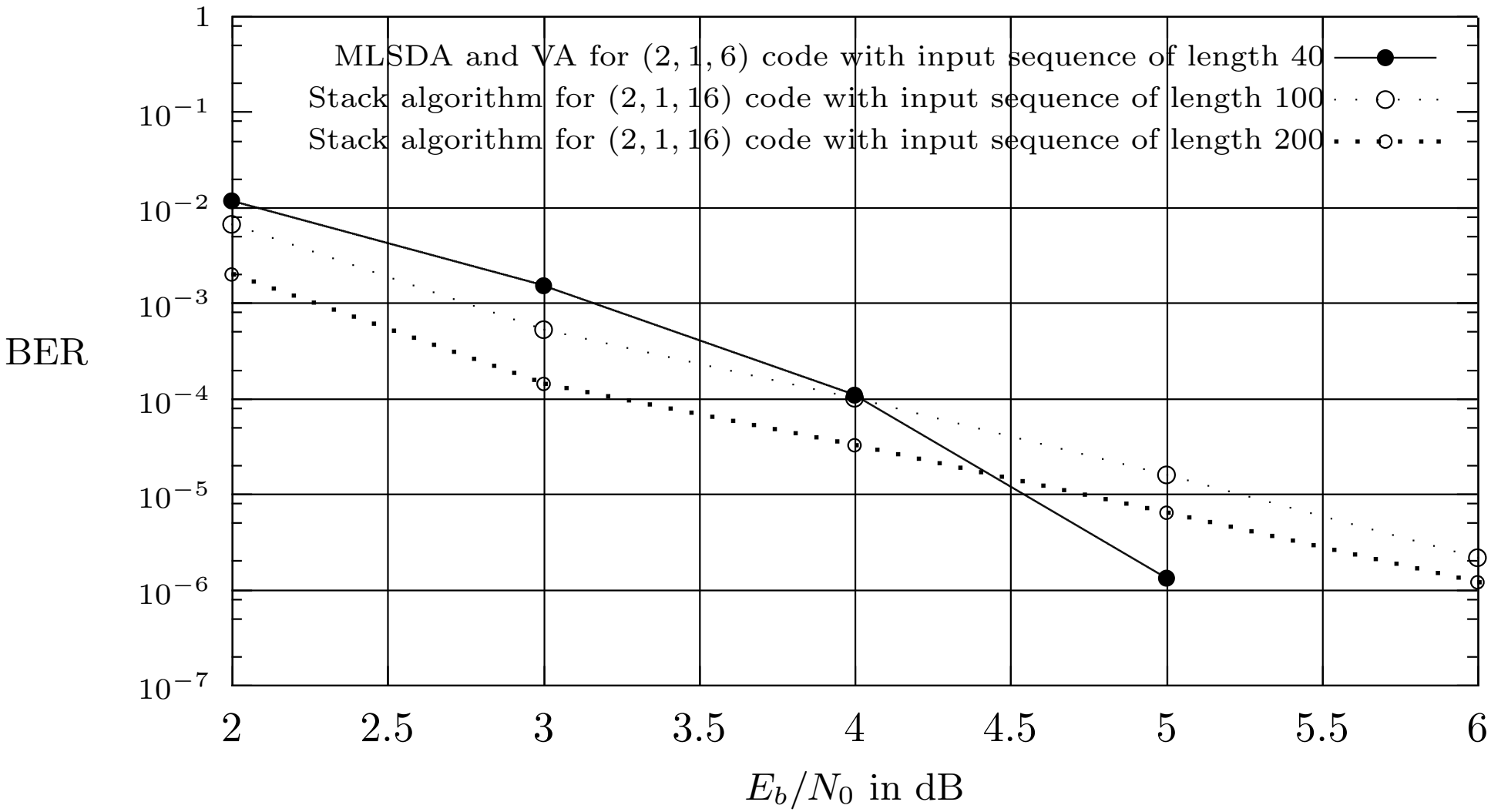
8. A remarkable feature of the new ML path metric is that when a newly extended path merges with an existing path of longer length, the ML path metric of the newly extended path is always greater than or equal to the cumulative ML metric of the existing path up to the merged node.

9. A newly generated path that is shorter than its merged path can be immediately deleted, reducing the redirection overhead of the MLSDA only to the case in which the newly generated

path and the merged existing path are equally long.

$(2, 1, 6)$ code


MLSDA and VA with input sequence of length 40
Stack algorithm with input sequence of length 40
MLSDA and VA with input sequence of length 200
Stack algorithm with input sequence of length 200
BER
1
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
1
2
3
4
5
6
7
E_b/N_0 in dB


MLSDA and VA for (2, 1, 6) code with input sequence of length 40
Stack algorithm for (2, 1, 16) code with input sequence of length 100
Stack algorithm for (2, 1, 16) code with input sequence of length 200
BER
1
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
2
2.5
3
3.5
4
4.5
5
5.5
6
E_b/N_0 in dB

References

[1] J. B. Anderson and S. Mohan, Sequential coding algorithms: a survey and cost analysis *IEEE Trans. Commun.*, vol. COM-32, no. 2, pp. 169–176, February 1984.

[2] N. Bélanger, D. Haccoun, and Y. Savaria, A multiprocessor architecture for multiple path stack sequential decoders *IEEE Trans. Commun.*, vol. 42, no. 2/3/4, pp. 951–957, February/March/April 1994.

[3] R. M. Fano, A heuristic discussion of probabilistic decoding *IEEE Trans. Inform. Theory*, vol. IT-9, no. 2, pp. 64–73, April 1963.

[4] G. D. Forney, Jr., Convolutional codes III: Sequential Decoding *Inf. Control, 25*, pp. 267–269, July 1974.

[5] Jr. G. D. Forney and E. K. Bower, A high-speed sequential decoder: prototype design and test *IEEE Trans. Commun. Technol.*, vol. COM-19, no. 5, pp. 821–835, October 1971.

[6] J. Geist, *Algorithmic aspects of sequential decoding*, PhD thesis, Dep. Elec. Eng., Univ. Notre Dame, Notre Dame, Ind., 1970.

[7] J. M. Geist, Am empirical comparison of two sequential decoding algorithms *IEEE Trans. Commun. Technol.*, vol. COM-19, no. 4, pp. 415–419, August 1971.

[8] J. M. Geist, Search properties of some sequential decoding algorithms *IEEE Trans. Inform. Theory*, vol. IT-19, no. 4, pp. 519–526, July 1973.

[9] D. Haccoun and M. J. Ferguson, Generalized stack algorithms

for decoding convolutional codes *IEEE Trans. Inform. Theory*, vol. IT-21, no. 6, pp. 638–651, November 1975.

[10] Y. S. Han, P.-N. Chen, and M. P. C. Fossorier, A Generalization of the Fano Metric and Its Effect on Sequential Decoding Using a Stack *IEEE Int. Symp. on Information Theory*, Lausanne, Switzerland, 2002.

[11] Y. S. Han, P.-N. Chen, and H.-B. Wu, A Maximum-Likelihood Soft-Decision Sequential Decoding Algorithm for Binary Convolutional Codes *IEEE Trans. Commun.*, vol. 50, no. 2, pp. 173–178, February 2002.

[12] F. Jelinek, A fast sequential decoding algorithm using a stack *IBM J. Res. and Dev., 13*, pp. 675–685, November 1969.

[13] D. E. Knuth, *The Art of Computer Programming. Volume III:*

Sorting and Searching. Reading, MA: Addison-Wesley, 1973.

[14] P. Lavoie, D. Haccoun, and Y. Savaria, A Systolic Architecture for Fast Stack Sequential Decoders *IEEE Trans. Commun.*, vol. 42, no. 5, pp. 324–335, May 1994.

[15] S. Lin and D. J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1983.

[16] J. L. Massey, Variable-Length Codes and the Fano Metric *IEEE Trans. Inform. Theory*, vol. IT-18, no. 1, pp. 196–198, January 1972.

[17] S. Mohan and J. B. Anderson, Computationally optimal metric-first code tree search algorithms *IEEE Trans. Commun.*, vol. COM-32, no. 6, pp. 710–717, June 1984.

[18] K. Sh. Zigangirov, Some sequential decoding procedures *Probl. Peredachi Inf., 2*, pp. 13–25, 1966.