Seamless Data-Rate Change Using Punctured Convolutional Codes for Time-Varying Signal-to-Noise Ratios

Y. Feria and K.-M. Cheung
Communications Systems Research Section

In a time-varying signal-to-noise ratio (SNR) environment, symbol rate is often changed to maximize data return. However, the symbol-rate change has some undesirable effects, such as changing the transmission bandwidth and perhaps causing the receiver symbol loop to lose lock temporarily, thus losing some data. In this article, we are proposing an alternate way of varying the data rate without changing the symbol rate and, therefore, the transmission bandwidth. The data rate change is achieved in a seamless fashion by puncturing the convolutionally encoded symbol stream to adapt to the changing SNR environment. We have also derived an exact expression to enumerate the number of distinct puncturing patterns. To demonstrate this seamless rate-change capability, we searched for good puncturing patterns for the Galileo (14,1/4) convolutional code and changed the data rates by using the punctured codes to match the Galileo SNR profile of November 9, 1997. We show that this scheme reduces the symbol-rate changes from nine to two and provides a comparable data return in a day and a higher symbol SNR during most of the day.

I. Introduction

In deep-space communications and other space communications, the signal-to-noise ratio (SNR) varies during a day. The degree of variation is determined by weather conditions, antenna elevation angle, antenna-pointing accuracy (both the transmitter and receiver antennas), changes in satellite latitude, and many other factors. For example, the total signal power-to–noise density ratio, P_t/N_o, during a typical 24-hour pass for the Galileo Mission can fluctuate in a range between 16 and 22 dB-Hz. In order to maximize the data return in this time-varying SNR environment, the transmitted symbol rate is varied as a function of the estimated P_t/N_o at the antenna. The symbol rate is set as high as possible under the constraint that the symbol SNR is high enough for the tracking loops to remain in lock and that the bit-error-rate (BER) requirement is met. In the Galileo Mission, there are six different symbol rates, and there can be as many as eight symbol-rate changes (from 10 to 640 symbols/s) during a day. One problem associated with the symbol-rate change at a low operating symbol SNR is that the symbol synchronization loop may have to reacquire the symbol phase, which may cause real-time data loss. A technique that involves opening the symbol loop at the moment of the symbol-rate change has been proposed,[1] but this

[1] J. Berner, "GLL Data Rate Changes," Project Notes (internal document), Jet Propulsion Laboratory, Pasadena, California, June 11, 1993.

technique requires very accurate time predicts on the moment of change. It is not clear if the predict information can be obtained within the required accuracy.

In this article, we are proposing a simple and low-cost alternative solution to the data rate–change problem by changing the data rate at the error-correcting coding stage rather than at the transmission stage. The data rate is changed by puncturing the low-rate convolutional code while the symbol rate is kept constant. In this way, the basic structures of the encoder and decoder remain unchanged, making the scheme simple and less costly. The idea is to minimize the number of symbol-rate changes and still maintain a high enough symbol SNR for the loops to remain in lock and the BER to stay low. Symbol rate is changed only if the symbol SNR goes too high (wasting bit SNR) or too low (making the receiver unable to track the symbols).

By allowing the code-rate change, we are essentially adding a degree of freedom in the data return–maximization problem. The code rate will share a part of the necessary data-rate changes with the symbol rate, therefore reducing the number of symbol-rate changes. This feature becomes even more important when the available bandwidth is fixed.

In Section II, we will present the definition and an overview of puncturing patterns. In Section III, we will discuss our procedure for selecting good puncturing patterns, and Section IV will provide an example of using the punctured convolutional code for the SNR profile of the Galileo Mission on November 9, 1997, which is an arbitrarily chosen day. In Section V, we will give some concluding remarks.

II. Definition and Enumeration of Puncturing Patterns

A. Definition of Puncturing Patterns

A regular-rate $1/N$ convolutional code generates N code symbols per bit. By periodically and systematically refraining from transmission of some of the code symbols, a higher rate code can be constructed from an original lower rate $1/N$ code. Let the period be L bits or NL code symbols. We define a puncturing pattern P of period NL symbols to be an NL binary-tuple, where a 1 denotes that the symbol in the corresponding location is to be sent and a 0 denotes that the symbol is to be deleted. If there are m zeros in P, the resulting punctured code is a higher rate $L/(NL-m)$ code, where $0 \leq m < (N-1)L$. For example, let $N = 4$, $L = 4$, and one puncturing pattern be $P = \{0111\ 1110\ 1011\ 1101\}$. We define the rightmost digit to correspond to the first symbol and the rightmost group of four digits to correspond to the four symbols of the first bit. The puncturing pattern, P, indicates that the second symbol in the first bit, the third symbol in the second bit, the first symbol in the third bit, and the fourth symbol in the fourth bit in a period are not transmitted. The resulting punctured code is a rate $4/(4\times4-4) = 1/3$ code. With the leftmost digit being the most significant bit and the rightmost digit being the least significant bit, the puncturing pattern, P, can be represented as $7ebd$ in hexadecimal form.

B. Enumeration of Puncturing Patterns

In this section, we develop an exact expression to enumerate the number of unique puncturing patterns.

Clearly, there are $\binom{NL}{m}$ different possible patterns for P. Since P is repeated every L bits or NL symbols, any cyclic shift of N symbols in P gives the same code performance as P. However, this does not reduce the number of patterns that give a distinct code performance by a factor of N, as some of the $\binom{NL}{m}$ patterns may have a smaller period L_i. That is, L_i divides L, which is denoted by $L_i \mid L$. Let $f(L_i)$ denote the number of puncturing patterns with period L_i exactly (including 1). Notice that $f(L_i) = 0$ if the m zeros cannot be evenly divided among L/L_i partitions (i.e., $(L/L_i) \nmid m$). Also, among the $\binom{NL_i}{mL_i/L}$ patterns with period L_i, some may have smaller periods. Let p be a prime that divides L_i. If $p|(mL_i/L)$,

then there are $\binom{NL_i/p}{mL_i/(Lp)}$ patterns of P with period L_i/p. The total number of distinct puncturing patterns is, therefore,

$$\sum_{L_i|L} \frac{1}{L_i} f(L_i)$$

where $f(L_i)$ can be enumerated as follows:

$$f(L_i) = \binom{NL_i}{\frac{mL_i}{L}} - \sum_{p|L_i} \binom{\frac{NL_i}{p}}{\frac{mL_i}{Lp}}$$

Notice that we define the combinatoric function $\binom{m}{n} = 0$ if either m or n is not an integer. In the above example with $N = 4$, $L = 4$, and $m = 4$, an exhaustive search requires checking $\binom{16}{4} = 1820$ puncturing patterns. By taking into account the cyclic property of the puncturing patterns, the number of distinct puncturing patterns is now reduced to

$$\frac{1}{4}\left[\binom{16}{4} - \binom{8}{2}\right] + \frac{1}{2}\left[\binom{8}{2} - \binom{4}{1}\right] + \binom{4}{1} = 464$$

which is a reduction by almost a factor of 4.

III. Puncturing Pattern Search Procedure

In this section, we describe the search procedure that we used to find good puncturing patterns for a rate-$1/N$ convolutional code. Using this procedure, we searched for punctured patterns for the (14,1/4) convolutional code used for Galileo. We punctured it from rate 1/4 to rate 1/3, then to rate 1/2. A rate compatibility [1] restriction is added in the puncturing-pattern search. That is, a code symbol used in the high-rate code is also used in the low-rate code. For example, to search for a rate-1/2 punctured code, we puncture the rate-1/3 code found a step before, not the rate-1/4 code. This was necessary mainly because of limited computing resources.

For each punctured code rate, the goal is to find the puncturing pattern, P, that gives the lowest BER at that rate for a range of SNR values. Direct simulation of the punctured convolutional code is not viable since there are so many different puncturing patterns. As a first step in selecting the puncturing patterns, we computed the weight profile of each punctured code that includes the free distance, d_{free}, the number of paths of weight d, a_d, and the information bit error weight, c_d. To further simplify our search, we only searched for paths of weight d such that $d_{free} \leq d \leq d_{cut}$, where d_{cut} is some predetermined value that is large enough to infer the code's BER performance and small enough to complete the search in a reasonable time. Note that there are L different starting points for diverging paths, where L is the period of the puncturing patterns. The worst case is considered in comparing the puncturing patterns.

A systematic search is carried out to find the patterns with the maximum free distance and the minimum number of paths of weight d for Viterbi decoding. Three patterns with the largest free distance and smallest number of paths of weight d are then simulated to obtain three BER curves. The lowest BER curve is selected and used further to compute the BERs for the concatenated codes of convolutional code as the inner code and the Reed–Solomon (RS) (255,223) code as the outer code, assuming infinite interleaving.

Once we have the points of E_b/N_o versus BER of the concatenated code, we fit a curve through these points. These curves are used to determine the BER for a given SNR profile of the Galileo Mission. When the BER is less than 10^{-7}, we determine that the code rate can be used in that time period.

Note that the (14,1/4) Galileo code is used here only to demonstrate the alternative possibility of using punctured codes. In fact, the (14,1/4) code is composed of a (11,1/2) convolutional code and the NASA standard (7,1/2) code. The NASA standard (7,1/2) code was necessary because the hardware encoder on the spacecraft cannot be altered or bypassed.

A. Upper Bound on Free Distance

Before searching for the maximum free distances, we compute the upper bounds of the free distances to see the effect of the puncturing period on the free-distance bound of the punctured codes. The upper bounds on the free distances for convolutional codes can be computed using expressions given in [2]. Figure 1 shows some of the bounds on the free distances for codes punctured from code (14,1/4), with the minimum period from 1 to 4. By minimum period we mean that the period 4 does not include period 1 or 2. The results show that a shorter puncturing period gives a higher upper bound on the free distance, but the shorter puncturing period provides a smaller set of possible code rates.

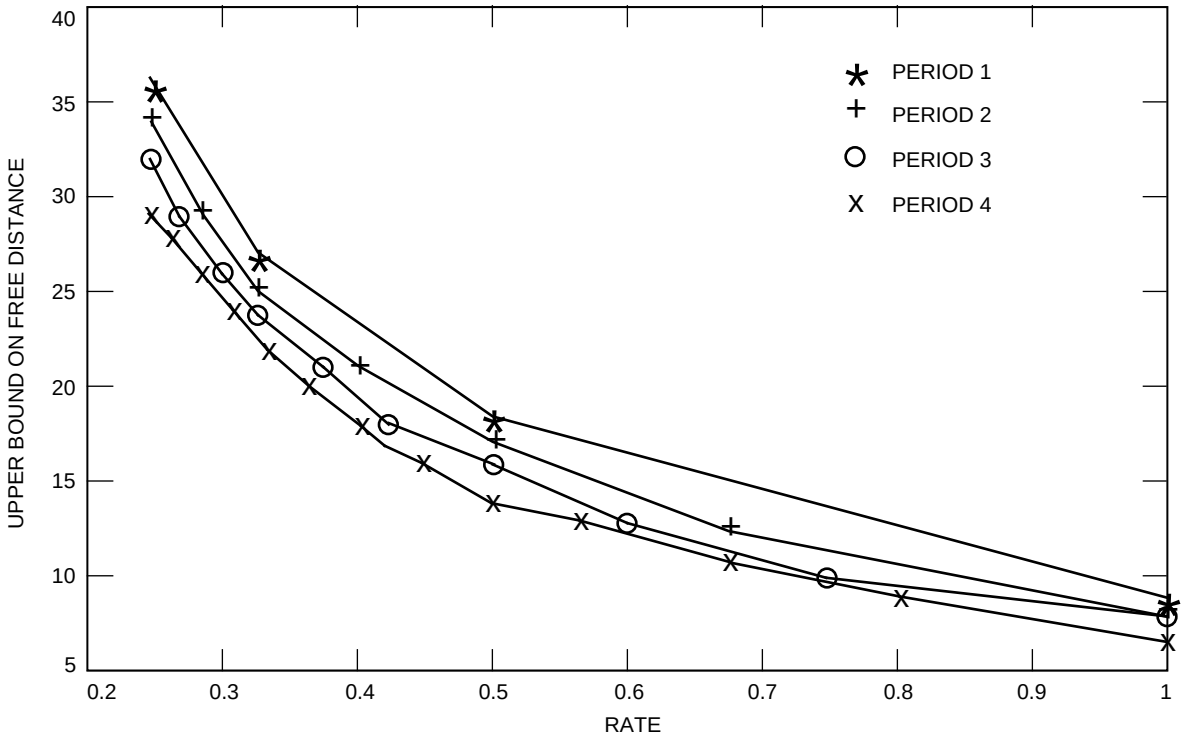

Fig. 1. Upper bounds on free distance for punctured codes from (14,1/4) code.

B. Weight Spectra of Punctured Codes

The parent code in this case has the following polynomials: $2c22$, $3d7d$, $2bcd$, and $1dd3$. First, we search for punctured codes from (14,1/4) to (14,1/3) and find the weight spectra corresponding to all different punctured patterns. The period in this case is 4, which corresponds to 464 different puncturing patterns. We then sort the weight spectra in ascending order according to the number of paths of weight d, a_d. Finally, we pick the best three patterns, and their weight spectra are shown in Table 1. According to the weight spectra, the best pattern is *bbbb*. This implies that the puncturing pattern has period 1, and the third symbol is punctured out every time. This corresponds to the (14,1/3) code with polynomials $2c22$, $3d7d$, and $1dd3$.

To further puncture the code to rate 1/2, we use the best 1/3 code found earlier as the parent code. The following patterns are found to be the best: 3636, 3535, and 3333 in octal numbers. The weight spectra of the three best puncturing patterns are shown in Table 2. Note that when searching for puncturing patterns with period 4, those patterns with period 1 and 2 are included.

Table 1. Weight spectra of punctured codes (14,1/3) from (14,1/4) parent code.

Pattern	d	23	24	25	26	27	28	29	30	31	32	33	34	35
bbbb	a_d	0	4	6	4	9	14	22	48	93	130	237	389	638
bdbd	a_d	1	1	3	8	13	21	27	54	68	137	225	400	652
bbbd	a_d	1	5	5	5	13	16	38	54	101	146	288	481	800
bbbb	c_d	0	14	18	18	55	72	122	322	641	920	1853	3134	5530
bdbd	c_d	1	3	9	35	60	121	139	320	486	938	1699	3150	5368
bbbd	c_d	5	14	19	24	77	91	240	347	724	1080	2313	4067	7068

Table 2. Weight spectra of punctured codes (14,1/2) from (14,1/3) parent code.

Pattern	d	13	14	15	16	17	18	19	20	21	22	23	24
3636	a_d	0	2	6	10	24	51	142	344	824	1956	4726	11363
3535	a_d	0	2	8	9	35	70	154	371	931	2286	5464	13234
3333	a_d	0	3	0	14	0	73	0	545	0	2884	0	16679
3636	c_d	0	5	20	70	146	354	1144	2914	7780	20229	52967	5525
3535	c_d	0	9	37	53	251	550	1298	3370	9353	25245	64261	35749
3333	c_d	0	9	0	71	0	520	0	4686	0	29943	0	4011

C. BER of Punctured Convolutional Code From Simulation

The weight-spectra search is only the first step in the code puncturing pattern search. To further compare their performance, the punctured codes are simulated with an encoder and the Viterbi decoder for several bit-SNR values. The traceback length used in the Viterbi decoder in this case is at least 160, and the input soft symbols are quantized with 8 bits. The simulated results are shown in Tables 3 and 4. Generally, the three puncturing patterns give similar BERs.

D. BER of Concatenated Code

Once we obtain the BER from the Viterbi decoder, we can compute the bit-error rate at the output of the RS decoder, assuming infinite interleaving using the expression given in [3, p. 256]. In the case of the Galileo Mission, there are 8 bits in a codeword, 255 codewords in a frame, and the number of correctable errors is 16. The computed BERs at the output of the RS decoder are shown in Tables 5 through 7.

IV. Example Using the Galileo Profile

We use the predicted SNR profile of the Galileo Mission on November 9, 1997, as an example to explain how the number of symbol-rate changes can be reduced with code-rate changes. For a given SNR

Table 3. BER of punctured convolutional codes (14,1/3).

E_b/N_o	Puncturing patterns		
	bbbb	*bdbd*	*bbbd*
−1.2494	0.3528	0.3532	0.3533
−0.7494	0.2320	0.2336	0.2330
−0.2494	0.1083	0.1100	0.1090
0.2506	0.0342	0.0345	0.0347
0.7506	0.0076	0.0077	0.0076
1.2506	0.0012	0.0012	0.0012

Table 4. BER of punctured convolutional codes (14,1/2).

E_b/N_o	Puncturing patterns		
	3636	3535	3333
−1.0103	0.4389	0.4339	0.4445
−0.5103	0.3573	0.3519	0.3625
−0.0103	0.2230	0.2206	0.2279
0.4897	0.0924	0.0934	0.0924
0.9897	0.0230	0.0243	0.0226
1.4897	0.0040	0.0043	0.0035
1.9897	0.0005	0.0005	0.0004
2.4897	4.7×10^{-5}	5.4×10^{-5}	3.5×10^{-5}

Table 5. BER output of RS decoder using punctured code (14,1/4).

E_b/N_o	BER input to RS decoder	BER output of RS decoder
−2.0	0.4218	0.4218
−1.5	0.3408	0.3408
−1.0	0.2139	0.2139
−0.5	0.1023	0.1023
0.0	0.0326	0.0194
0.5	0.0070	5.4×10^{-9}
1.0	0.0013	2.3×10^{-20}
1.5	0.0002	1.1×10^{-49}

Table 6. BER output of RS decoder using punctured codes (14,1/3).

E_b/N_o	BER input to RS decoder	BER output of RS decoder
−1.2494	0.3528	0.3528
−0.7494	0.2320	0.2320
−0.2494	0.1083	0.1083
0.2506	0.0342	0.0230
0.7506	0.0076	1.8×10^{-8}
1.2506	0.0012	1.0×10^{-20}

Table 7. BER output of RS decoder using punctured codes (14,1/2).

E_b/N_o	BER input to RS decoder	BER output of RS decoder
−1.0103	0.4389	0.4389
−0.5103	0.3573	0.3573
−0.0103	0.2230	0.2230
0.4897	0.0924	0.0924
0.9897	0.0230	0.0029
1.4897	0.0040	1.6×10^{-12}
1.9897	0.0005	3.7×10^{-27}
2.4897	4.7×10^{-5}	1.6×10^{-44}

profile—for example, the one shown in Fig. 2—the objective is to get the maximum data return under the conditions that the bit-error rate is below 10^{-7} and the symbol SNR is maintained above −6 dB for the carrier, subcarrier, and symbol loops to track. To achieve this goal, the current plan is to change the symbol rate using a fixed code rate, 1/4, and an alternate way is to allow the code rate to change as well, thus reducing the number of symbol-rate changes.

We arbitrarily select a set of three code rates, namely, 1/4, 1/3, and 1/2. The variable code rate can only take values from this set. Figure 3 shows the symbol rates using fixed and variable code rates. In the fixed–code rate case, there are nine symbol-rate changes, compared to two symbol-rate changes in the variable–code rate case. With these symbol rates, each of the two systems will have a symbol SNR above −6 dB, as required, where the variable–code rate case has a slightly higher symbol SNR for most of the day, as shown in Fig. 4. The code-rate changes are shown in Fig. 5.

Multiplying the code rates by the symbol rates, we obtain the bit rates as shown in Fig. 6 for the fixed– and variable–code rate cases. The areas under the two curves in Fig. 6 are the total data returns for the day. The data return using the variable code rate is found to be comparable with that using the fixed code rate.

V. Conclusions

In this article, we have described a simple and low-cost method to change the data rate to match the time-varying P_t/N_0 environment. This is done by puncturing the convolutional code at the error-correction coding stage rather than by changing the symbol rate at the transmission stage. The main advantages of this method are that it allows seamless transition from one data rate to another and that,

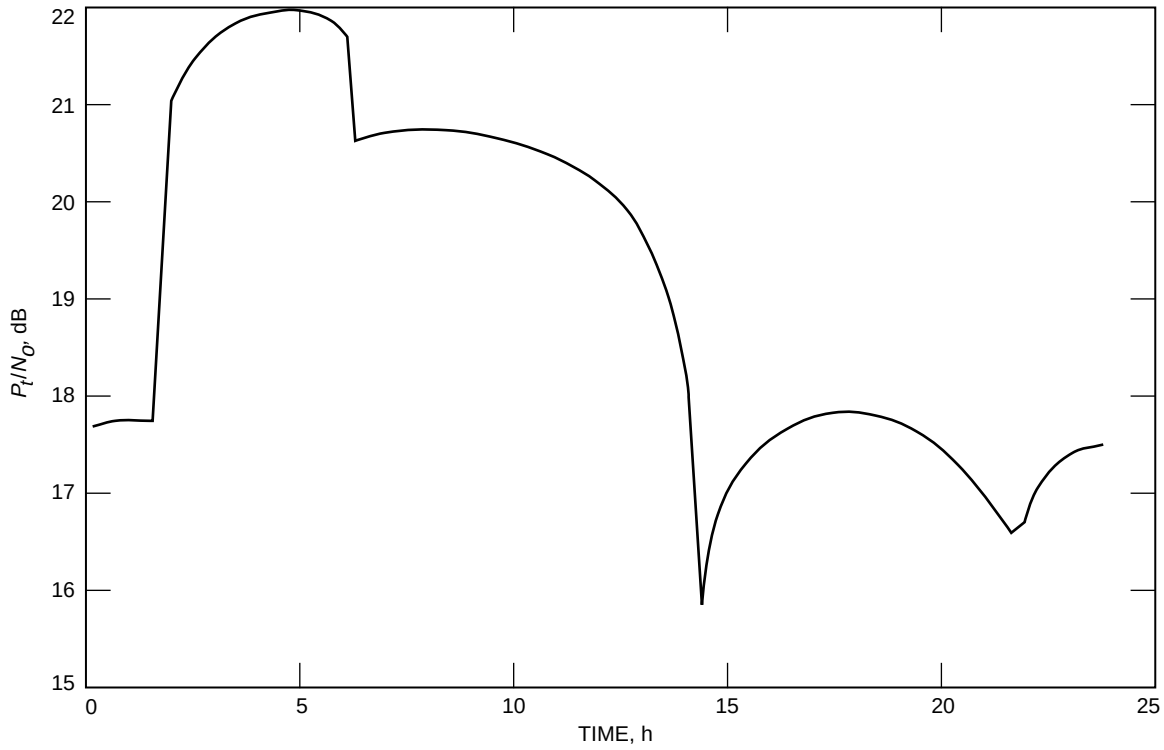

Fig. 2. Arrayed P_t/N_O on November 9, 1997, from Galileo.

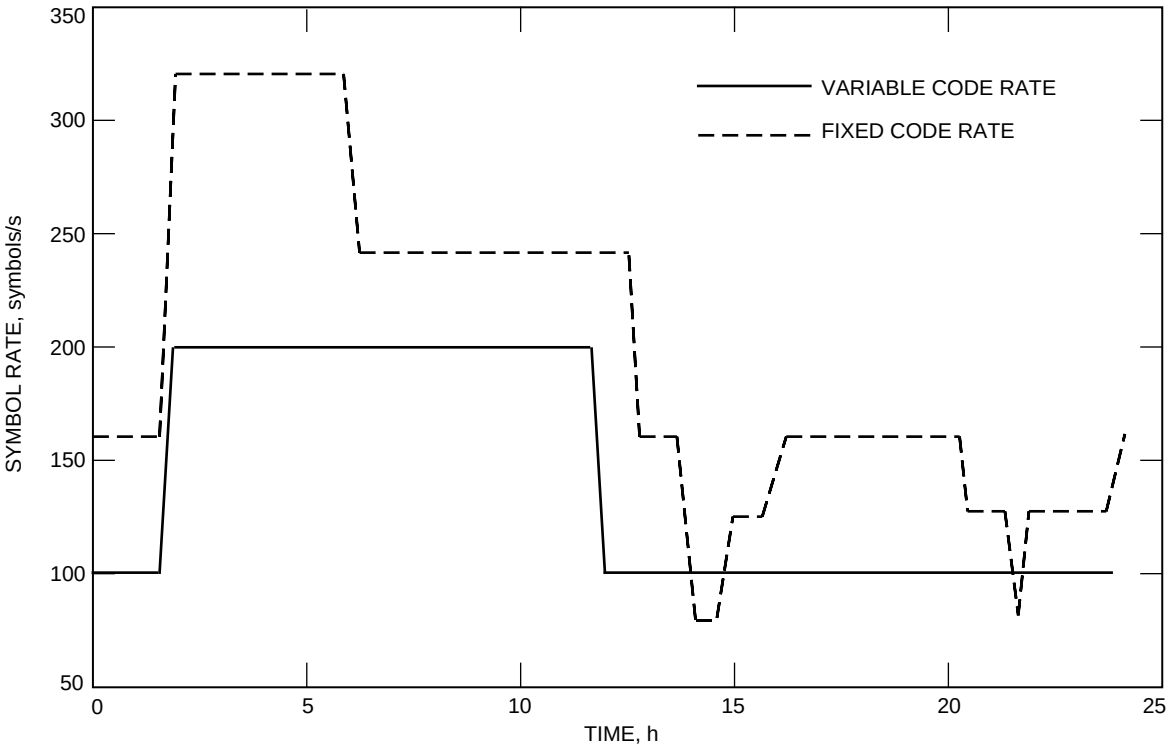

Fig. 3. Symbol rates on November 9, 1997, for Galileo using fixed and variable code rates.

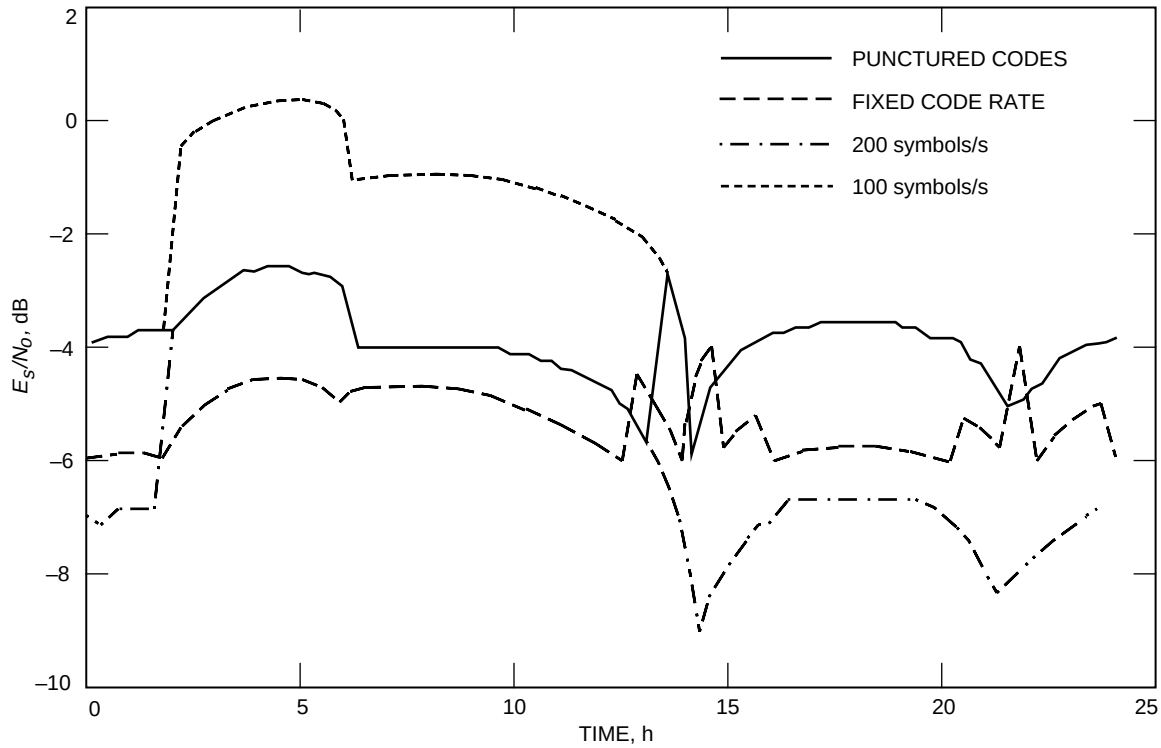

Fig. 4. Symbol SNR on November 9, 1997, for Galileo using fixed and variable code rates.

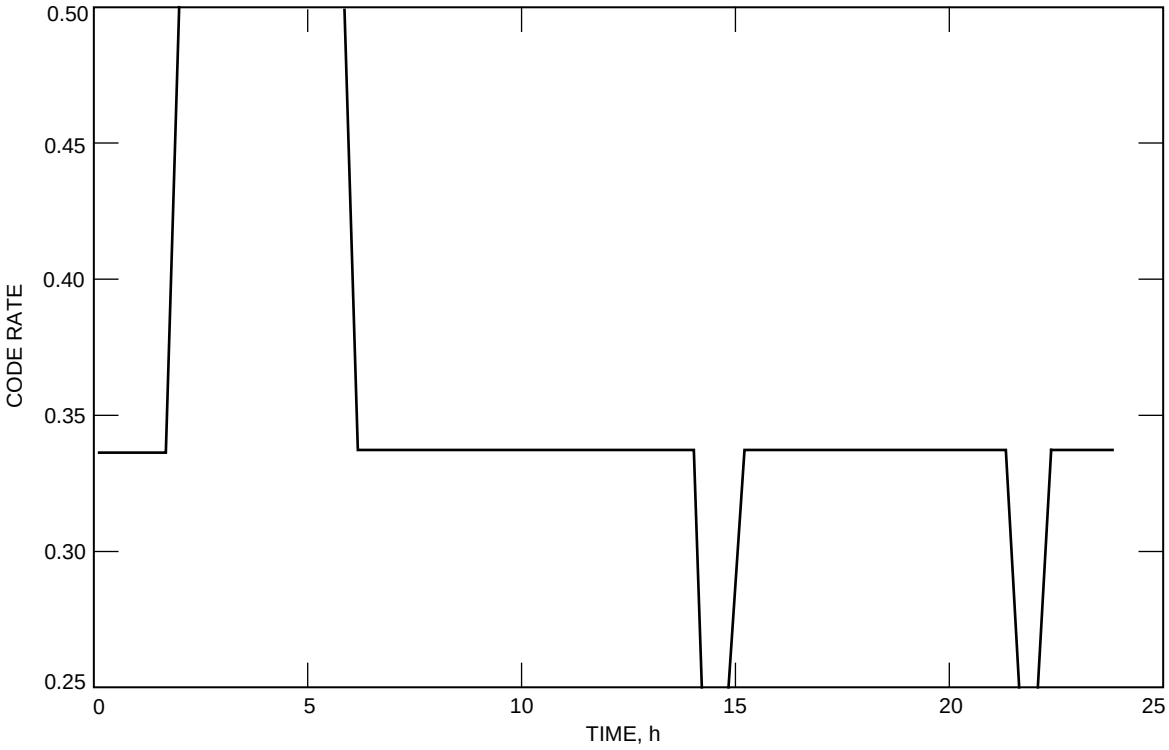

Fig. 5. Variable code rates on November 9, 1997, for Galileo.

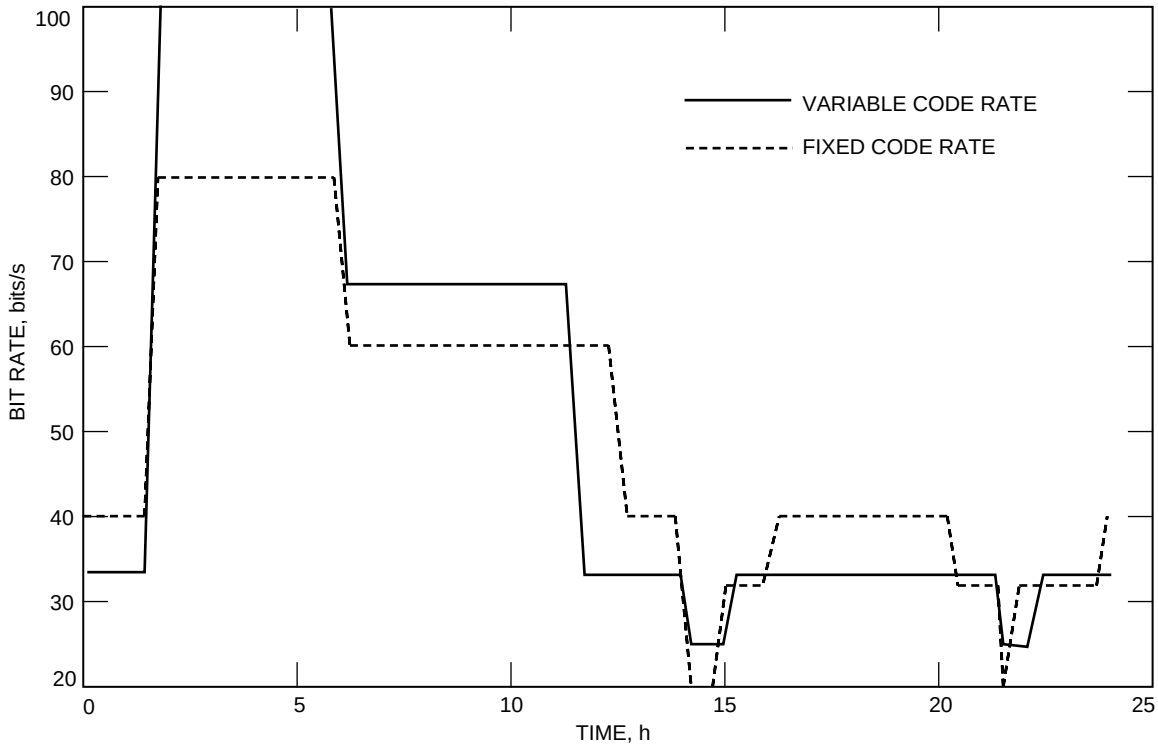

Fig. 6. Bit rates on November 9, 1997, for Galileo.

for a fixed available bandwidth, data rate is allowed to change for a larger data return. We applied this method to the Galileo SNR profile on November 9, 1997, as an example to demonstrate its effectiveness. We showed how this method reduces the number of symbol-rate changes from nine to two and gives a comparable data return in a day and a higher symbol SNR for most of the day.

Notice that, in this example, we arbitrarily picked 100 and 200 symbols/s as two symbol rates to be used. We are developing techniques to select the symbol rates that will maximize the data return. The problem is formulated below.

For a given SNR profile, $(P_t/N_o)(t)$, we wish to find the symbol rate $R_{sym}(t)$ and the code rate $R_c(t)$ such that the data return given by $\int_{t_1}^{t_2} R_{sym}(t)R_c(t)dt$ is maximal, subject to the following constraints:

(1) The number of changes of symbol rate R_{sym} is less than a desired number.

(2) The BER is below a designed value, $BER < BER_{design}$.

(3) The symbol SNR is above the minimum value for the tracking loops to maintain lock, $E_s/N_o > (E_s/N_o)_{min}$.

Acknowledgments

Special thanks to Fabrizio Pollara for his help and discussions and for providing software for weight spectra calculation. The authors are also very grateful to Sam Dolinar for his many suggestions, to David Bell for providing the Galileo profile software, and to Todd Chauvin for providing the Viterbi decoder simulation tools.

References

[1] J. Hagenauer, "Rate-Compatible Punctured Convolutional Codes (RCPC Codes) and Their Applications," *IEEE Transactions on Communications*, vol. 36, no. 4, pp. 389–400, April 1988.

[2] D. G. Daut, J. W. Modestino, and L. D. Wismer, "New Short Constraint Length Convolutional Code Constructions for Selected Rational Rates," *IEEE Transactions on Information Theory*, vol. IT-28, no. 5, pp. 794–800, September 1982.

[3] J. H. Yuen, *Deep Space Telecommunications Systems Engineering*, New York: Plenum Press, 1983.