Short Huffman Codes Producing 1s Half of the Time

Fabian Altenbach, Georg Böcherer, and Rudolf Mathar
Institute for Theoretical Information Technology
RWTH Aachen University, 52056 Aachen, Germany
Email: {altenbach,boecherer,mathar}@ti.rwth-aachen.de

***Abstract*—The design of the channel part of a digital communication system (e.g., error correction, modulation) is heavily based on the assumption that the data to be transmitted forms a fair bit stream. However, simple source encoders such as short Huffman codes generate bit streams that poorly match this assumption. As a result, the channel input distribution does not match the original design criteria. In this work, a simple method called half Huffman coding (HALFHC) is developed. HALFHC transforms a Huffman code into a source code whose output is more similar to a fair bit stream. This is achieved by permuting the codewords such that the frequency of 1s at the output is close to 0.5. The permutations are such that the optimality in terms of achieved compression ratio is preserved. HALFHC is applied in a practical example, and the resulting overall system performs better than when conventional Huffman coding is used.**

I. Introduction

The key part of a digital communication system is the binary interface between source coding and channel coding [1, p. 1–3]. The ultimate objective of source coding is to transform the data into a sequence of 0s and 1s that is indistinguishable from a fair bit stream, i.e., a sequence of independent and identically distributed (iid) equiprobable bits. Accordingly, most work on channel coding starts with the assumption that the data to be transmitted comes as a fair bit stream.

In [2], we have shown that the capacity achieving input probability mass function (pmf) of a channel can be approximated arbitrarily well by parsing a fair bit stream by a matcher code and by doing so, capacity achieving modulation can be established [3]. However, this result is heavily based on the assumption that the binary interface is a fair bit stream. Asymptotically in terms of source encoder complexity, the binary interface can indeed be turned into a fair bit stream [4]. In practice, however, we are often restricted to simple encoders, and the resulting binary output differs significantly from a fair bit stream. As a simple measure to evaluate how close the generated bit stream is to a fair bit stream, the frequency of generated 1s can be considered. For a fair bit stream, this frequency should be 0.5. One possibility to achieve this goal is to pass the source encoder output through a scrambler [5]. The drawback of this approach is that the corresponding descrambler has to be known and implemented at the receiver [6].

This work has been supported by the UMIC Research Center, RWTH Aachen University.

In this work we propose an algorithm called *half Huffman coding* (HALFHC) that constructs prefix-free source codes. HALFHC achieves the optimal compression ratio (i.e., the same as Huffman coding (HC)), and the frequency of 1s of HALFHC is closer to 0.5 than the frequency of 1s of HC. As in the case of conventional HC, no additional descrambler at the receiver is required. We apply HALFHC to the shaping problem in [7]. We show that the resulting channel input pmf is closer to the one predicted by the fair bit stream assumption than the channel input pmf resulting from conventional HC. A complete implementation of HALFHC in Matlab can be found at our website [8].

The remainder of this paper is organized as follows. In Section II, we state the main problem. The idea of our solution is formulated in Section III. We then formulate in Section IV our idea as a mathematical optimization problem, propose methods for solving it, and formulate our new algorithm HALFHC. Finally, in Section V we apply HALFHC to a practical example.

II. Problem Statement

A. Motivating example

In [7], we consider a channel with three input symbols `r,l,m` with the associated costs

$$\boldsymbol{w} = (0.18,\ 0.18,\ 0.31)^T, \tag{1}$$

where $(\cdot)^T$ is the transpose. The channel input pmf $\boldsymbol{p}$ is subject to the cost constraint

$$\boldsymbol{w}^T\boldsymbol{p} \le S = 0.2063. \tag{2}$$

The optimal channel input probability mass function (pmf) was calculated as

$$\boldsymbol{p}^* = (0.3988,\ 0.3988,\ 0.2023)^T. \tag{3}$$

Note that $\boldsymbol{p}^*$ fulfills the cost constraint with equality. The class of pmfs that can be generated at the channel input by parsing a fair bit stream by a matcher code are the *dyadic pmfs* [2]. A pmf is dyadic if each entry p_i can be written in the form

$$p_i = 2^{-\ell}, \quad \ell \in \mathbb{N} \tag{4}$$

i.e., each entry is a positive integer power of $1/2$. The optimization problem in [2] of finding an optimal matcher

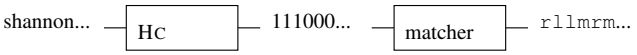

Fig. 1. We first compress the text to a binary sequence by HC and then match the binary sequence to the design criteria by a matcher code. At the output of the matcher, a sequence of the symbols `r,l,m` is generated.

code can be stated as

$$\begin{aligned} \text{minimize} \quad & \mathrm{D}(\boldsymbol{d}\|\boldsymbol{p}^*) \\ \text{subject to} \quad & \boldsymbol{w}^T\boldsymbol{d} \leq S \\ & \boldsymbol{d} \text{ is a dyadic pmf,} \end{aligned} \tag{5}$$

where $\mathrm{D}(\cdot\|\cdot)$ is the Kullback-Leibler (KL) distance, which is defined as

$$\mathrm{D}(\boldsymbol{p}\|\boldsymbol{q}) = \sum_i p_i \log \frac{p_i}{q_i}, \tag{6}$$

with $\log$ denoting the natural logarithm. In [7], the algorithm *cost constrained geometric Huffman coding* (CCGHC) is developed, and it is shown that for a fair bit stream at the binary interface, the resulting matcher code optimally solves Problem (5). The solution $\boldsymbol{d}$ achieves

$$\mathrm{D}(\boldsymbol{d}\|\boldsymbol{p}^*) = 0.0048392 \tag{7}$$

$$\boldsymbol{w}^T\boldsymbol{d} = 0.20607 \leq S. \tag{8}$$

However, the bit stream at the binary interface is generated by compressing the English text [9] by a simple HC, and this bit stream is then parsed by the matcher code, see Fig. 1. By HC we refer to Huffman coding as implemented by `huffmandict.m` in the Matlab Communications System Toolbox. The effective pmf, the effective KL-distance, and the effective cost at the output of the matcher are respectively given by

$$\boldsymbol{d}_{\mathrm{HC}} = (0.40663,\ 0.35761,\ 0.23576)^T \tag{9}$$

$$\mathrm{D}(\boldsymbol{d}_{\mathrm{HC}}\|\boldsymbol{p}^*) = 0.0050066 \tag{10}$$

$$\boldsymbol{w}^T\boldsymbol{d}_{\mathrm{HC}} = 0.21065 > S \tag{11}$$

i.e., what is actually observed significantly differs from what we would expect under the fair bit stream assumption.

B. Approach

While observations (9)–(11) may occur since the effectively observed values are a realization of a random process, which can always diverge from the expected values, the reason can also be that the assumption of a fair bit stream is false. To evaluate the quality of the binary interface, we calculate the effective relative number of 1s. For a fair bit stream, this number should be close to $q = 0.5$. However, the effective frequency that we observe after applying HC to the English text [9] is

$$q_{\mathrm{HC}} = 0.45821. \tag{12}$$

A maximum likelihood estimator (MLE) of q yields for the bit stream at the output of HC the 95% confidence interval $(0.4464,\ 0.47006)$. Since the corresponding value of $q = 0.5$ is not contained in this interval, the MLE rejects the hypothesis of having a fair bit stream after applying HC. Our approach is therefore to look for optimal prefix-free source codes that yield a q close to 0.5.

TABLE I
HC FOR THE ENGLISH TEXT [9]

	i	p_i	$\boldsymbol{c}_i$	$l(i)$	j
_	1	0.1699	000	3	1
e	2	0.0964	110	3	1
t	3	0.0777	0010	4	2
a	4	0.0717	0100	4	2
i	5	0.0663	0101	4	2
o	6	0.0645	0111	4	2
n	7	0.0614	1000	4	2
r	8	0.0530	1010	4	2
s	9	0.0500	1110	4	2
h	10	0.0373	00110	5	3
c	11	0.0325	01101	5	3
l	12	0.0325	01100	5	3
m	13	0.0277	10110	5	3
u	14	0.0277	10011	5	3
d	15	0.0235	11110	5	3
f	16	0.0199	11111	5	3
g	17	0.0181	001110	6	4
y	18	0.0145	100100	6	4
p	19	0.0139	100101	6	4
b	20	0.0133	101110	6	4
w	21	0.0114	101111	6	4
v	22	0.0054	00111100	8	5
k	23	0.0042	00111101	8	5
x	24	0.0024	001111100	9	6
q	25	0.0018	001111110	9	6
z	26	0.0018	001111101	9	6
j	27	0.0012	001111111	9	6

III. MAIN IDEA

As stated in [10, Sec. 5.8], optimal prefix-free source codes are not unique and for a given pmf, HC constructs *one* optimal code. All other optimal codes can be obtained by applying appropriate permutations of the code generated by HC. In this section we will show how the frequency of 1s can be influenced by specific permutations.

A. Frequency of 1s

Denote the number of symbols of the considered source by n. Without loss of generality, we assume that the pmf $\boldsymbol{p}$ describing the source is ordered with decreasing probabilities, i.e,

$$p_1 \geq p_2 \geq \cdots \geq p_n. \tag{13}$$

Denote by $\mathcal{C}$ the ordered set of codewords generated by HC, i.e.,

$$\mathcal{C} = (\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n). \tag{14}$$

Denote by $\mathbf{1}^T\boldsymbol{c}_i$ the number of 1s in the ith codeword. We write the length of codeword $\boldsymbol{c}_i$ as $l(i)$. Denote by m the number of distinct codeword lengths and order and index the set of distinct codeword lengths $\{\ell_j\}_{j=1}^m$. For the English text [9] we have $i, p_i, \boldsymbol{c}_i, l(i)$ and j displayed in Table I. Denote

the pmf of the codeword lengths by $\boldsymbol{r}$, i.e.,

$$r_j = \sum_{i:l(i)=\ell_j} p_i, \qquad j = 1, \ldots, m. \tag{15}$$

Denote by $p_{i|j}$ the probability that symbol i is generated given that the codeword length is ℓ_j, i.e., if $p_{i|j} > 0$, then

$$p_{i|j} = \frac{p_i}{r_j}. \tag{16}$$

Denote by N_j the expected number of 1s conditioned on codeword length ℓ_j, i.e.,

$$N_j = \sum_{i:l(i)=\ell_j} p_{i|j} \mathbf{1}^T \boldsymbol{c}_i. \tag{17}$$

Thus the expected frequency q of 1s given by the expected number of 1s N per average length L can be written as

$$q = \frac{N}{L} \tag{18}$$

$$= \frac{\sum_{j=1}^{m} r_j N_j}{\sum_{j=1}^{m} \sum_{i:l(i)=\ell_j} p_{i|j} r_j \ell_j}. \tag{19}$$

B. Permutations of $\mathcal{C}$

Any permutation of codewords of the same length again gives an optimal code. However, while the achieved compression rate is invariant under these permutation, the mean number of 1s is not. This is because in general, there are codewords of the same length that occur with different probabilities. A naive approach consists of searching through all permutations, and choose the one with the mean number of 1s being closest to $0,5$. However, the number of such permutations becomes very fast prohibitively large. For instance, for the code displayed in Table I, this number is approximately

$$2!7!7!5!2!4! \approx 3 \cdot 10^{11}. \tag{20}$$

Consider now the set of codewords $\mathcal{C}_j$ of codewords of length ℓ_j. We consider two special permutations of $\mathcal{C}_j$. First, the one that *maximizes* the expected number of 1s and second, the one that *minimizes* the expected number of 1s. The first is obtained by ordering the codewords in $\mathcal{C}_j$ with decreasing number of 1s, and the second one is obtained by ordering the codewords in $\mathcal{C}_j$ with increasing number of 1s. Denote the corresponding permutations by π_j^+ and π_j^-, respectively. For example, for the code in Table I, the corresponding permutations for codeword length $\ell_1 = 3$ are

$$\pi_1^+ : 1 \mapsto 2, \quad 2 \mapsto 1 \tag{21}$$

$$\pi_1^- : 1 \mapsto 1, \quad 2 \mapsto 2. \tag{22}$$

The maximum expected number N_j^+ of 1s in a codeword, conditioned on the codeword length ℓ_j is given by

$$N_j^+ = \sum_{i:l(i)=\ell_j} p_{i|j} \mathbf{1}^T \boldsymbol{c}_{\pi_j^+(i)} \tag{23}$$

and the minimum expected number of 1s N_j^- is given by

$$N_j^- = \sum_{i:l(i)=\ell_j} p_{i|j} \mathbf{1}^T \boldsymbol{c}_{\pi_j^-(i)}. \tag{24}$$

For example, in case of codeword length 3 (corresponding to ℓ_1), we can see from Table I that

$$N_1^+ = \frac{p_1}{p_1+p_2} \cdot 2 + \frac{p_2}{p_1+p_2} \cdot 0 \approx 1.276$$
$$N_1^- = \frac{p_1}{p_1+p_2} \cdot 0 + \frac{p_2}{p_1+p_2} \cdot 2 \approx 0.724$$

The idea is now to choose for each codeword length ℓ_j between N_j^+ and N_j^-, with the objective to get the overall expected number of 1s divided by the expected codeword length as close to $q = 0.5$ as possible. This means, we would like to solve

$$q = \frac{\sum_{j=1}^{m} r_j N_j^{z_j}}{L} \overset{!}{\approx} 0.5 \tag{25}$$

where each z_j takes values in $\{+, -\}$. The corresponding optimization problem is discussed in the next section.

IV. Half Huffman Coding

We will express goal (25) in terms of a mathematical optimization problem. For each index j corresponding to a specific codeword length ℓ_j, we can either take the maximizing permutation π_j^+ or the minimizing permutation π_j^-. We introduce a binary vector $\boldsymbol{x} \in \{0,1\}^m$, which serves as a selection variable between both permutations. By using (25), the expected frequency of 1s can be expressed in terms of x_j as

$$q = \frac{\sum_{j=1}^{m} r_j N_j^{z_j}}{L} \tag{26}$$

$$= \frac{1}{L} \sum_{j=1}^{m} r_j \left[x_j (N_j^- - N_j^+) + N_j^+ \right]. \tag{27}$$

For $x_j = 0$, we choose N_j^+, and for $x_j = 1$, we choose N_j^-. We measure the quality of our selection $\boldsymbol{x}$ by the absolute deviation between q and 0.5. Hence, in vector notation, the objective we want to minimize has the form

$$|q - 0.5| = \left| \frac{1}{L} \left[\boldsymbol{r} \circ (\boldsymbol{n}_- - \boldsymbol{n}_+) \right]^T \boldsymbol{x} + \frac{1}{L} \boldsymbol{n}_+^T \boldsymbol{r} - 0.5 \right|, \tag{28}$$

where $\circ$ denotes the elementwise Hadamard vector product, and $\boldsymbol{n}_-$ and $\boldsymbol{n}_+$ are defined as

$$\boldsymbol{n}_- = (N_1^-, \ldots, N_m^-)^T \tag{29}$$

$$\boldsymbol{n}_+ = (N_1^+, \ldots, N_m^+)^T. \tag{30}$$

For notational convenience, we further substitute

$$\boldsymbol{a} = \frac{1}{L} \left[\boldsymbol{r} \circ (\boldsymbol{n}_- - \boldsymbol{n}_+) \right] \tag{31}$$

$$b = \frac{1}{L} \boldsymbol{n}_+^T \boldsymbol{r} - 0.5. \tag{32}$$

We can now state the optimization problem

$$\begin{array}{ll} \underset{\boldsymbol{x}}{\text{minimize}} & |\boldsymbol{a}^T\boldsymbol{x}+b| \\ \text{subject to} & x_j \in \{0,1\}, \quad j=1,\ldots,m. \end{array} \tag{33}$$

Introducing the epigraph variable $t \in \mathbb{R}_+$ [11], we can directly see that the problem (33) is equivalent to

$$\begin{array}{ll} \underset{\boldsymbol{x},t}{\text{minimize}} & t \\ \text{subject to} & -t \le \boldsymbol{a}^T\boldsymbol{x}+b \le t \\ & x_j \in \{0,1\}, \quad j=1,\ldots,m. \end{array} \tag{34}$$

This problem is a mixed integer linear program (MILP) in its canonical form [12], and is generally hard to solve. Since we are focusing on short-length Huffman codes, the number of different codeword lengths m will not be too large. As discussed below, we can use standard methods for solving this problem globally.

A. Optimization methods

1) Naive exhaustive search: In order to find the global solution to problem (33), we can simply try all possible vectors $\boldsymbol{x} \in \{0,1\}^m$. In our example from Table I, $m = 6$, that is, we have to choose between $2^6 = 64$ possible vectors $\boldsymbol{x}$. Thus, in our example we have overcome the huge number of possible permutations (20), but in general, we still might be constrained by the combinatorial nature of problem (34), since the complexity of exhaustive search grows exponentially in the number of distinct codeword lengths m. This problem can be overcome by considering smarter search algorithms, as discussed next.

2) Combinatorial feasibility method via bisection: We can also exploit some structure in the MILP formulation (34) by using a bisection method [11, p. 146]. Suppose we set the epigraph variable to a fixed value t. We can now try to find a feasible solution to the remaining combinatorial feasibility problem

$$\begin{array}{ll} \text{find} & \boldsymbol{x} \\ \text{subject to} & -t \le \boldsymbol{a}^T\boldsymbol{x}+b \le t \\ & x_j \in \{0,1\}, \quad j=1,\ldots,m. \end{array} \tag{35}$$

There are two possible cases that can occur:

1) If we find a feasible solution, the particular choice of t is greater or equal than the smallest possible value. Hence the value of t can be further decreased.
2) When there is no feasible solution, the choice of t was too small and we have to increase it.

After checking both cases we can solve the feasibility problem with an updated version of t, and repeat until convergence. This approach is summarized in Algorithm 1.

3) Specific branch and bound method: Formulation (33) falls into the class of problems discussed in [13, Sec. 2]. Thus, if none of the methods proposed in Subsection IV-A1 and Subsection IV-A2 can solve the problem in acceptable time, a specific branch and bound solver for problem (33) can easily be implemented and still finds the optimal solution in hopefully reasonable time.

Algorithm 1.

set l, u, tolerance $\epsilon > 0$
repeat
1. $t := (l+u)/2$
2. Solve the combinatorial feasibility problem (35)
3. **if** (35) is feasible
   decrease $u := t$;
   else
   increase $l := t$;

until $u - l \le \epsilon$
return $\boldsymbol{x}$

B. Half Huffman coding

We can now state HALFHC, see Algorithm 2 for a summary. In detail, we are given a pmf with entries sorted in descending order. First, we calculate the conventional Huffman code $\mathcal{C} = \text{HC}(\boldsymbol{p})$. Then, for each codeword length ℓ_j, we determine the maximum and minimum permutations π_j^+ and π_j^-, respectively. We use these permutations to calculate the vectors $\boldsymbol{n}_+, \boldsymbol{n}_-$ of maximum and minimum expected numbers of 1s. For the resulting vectors, we solve Problem (33) by any method from IV.A in order to find an optimal selection vector $\boldsymbol{x}$. The selection vector now determines which permutation has to be applied for each codeword length ℓ_j, i.e.,

$$\pi_j = \begin{cases} \pi_j^+, & \text{if } x_j = 0 \\ \pi_j^-, & \text{if } x_j = 1 \end{cases}, \qquad j = 1,\ldots,m. \tag{36}$$

Finally, the resulting permutation $\boldsymbol{\pi} = (\pi_1,\ldots,\pi_m)$ is applied to get the final code, i.e., $\text{HALFHC}(\boldsymbol{p}) = \pi(\mathcal{C})$. A complete implementation of HALFHC in Matlab can be found at our website [8].

Algorithm 2. (HALFHC)

$p_1 \ge \cdots \ge p_n$
1. $\mathcal{C} = \text{HC}(\boldsymbol{p})$
2. find $\boldsymbol{n}_+, \boldsymbol{n}_-$ via (23), (24)
3. $\boldsymbol{x}$ = solution of (33) via any method from IV.A
4 . $\boldsymbol{\pi} = (\pi_1,\ldots,\pi_m)$ according to (36)
return $\boldsymbol{\pi}(\mathcal{C})$

V. Numerical Results

We apply HALFHC to the English text [9]. We execute HALFHC twice. First, we use in step 3 exhaustive search IV-A1. Second, we use the combinatorial feasibility method IV-A2. Both methods find the same selection vector $\boldsymbol{x}$, which is given by

$$\boldsymbol{x} = (1,\ 0,\ 0,\ 0,\ 0,\ 0)^T. \tag{37}$$

The generated code is displayed in Table II. As can be seen, for $\ell_1 = 3$, the codewords are sorted with decreasing number of 1s, while for remaining codeword lengths, the codewords with increasing number of 1s. Notice the differences to the

TABLE II
HALFHC FOR THE ENGLISH TEXT [9]

	i	p_i	$\boldsymbol{c}_i$	j	ℓ_j
_	1	0.1699	000	3	1
e	2	0.0964	110	3	1
t	3	0.0777	1110	4	2
a	4	0.0717	0111	4	2
i	5	0.0663	1010	4	2
o	6	0.0645	0101	4	2
n	7	0.0614	1000	4	2
r	8	0.0530	0100	4	2
s	9	0.0500	0010	4	2
h	10	0.0373	11111	5	3
c	11	0.0325	10011	5	3
l	12	0.0325	11110	5	3
m	13	0.0277	01101	5	3
u	14	0.0277	10110	5	3
d	15	0.0235	01100	5	3
f	16	0.0199	00110	5	3
g	17	0.0181	101111	6	4
y	18	0.0145	101110	6	4
p	19	0.0139	100101	6	4
b	20	0.0133	001110	6	4
w	21	0.0114	100100	6	4
v	22	0.0054	00111101	8	5
k	23	0.0042	00111100	8	5
x	24	0.0024	001111111	9	6
q	25	0.0018	001111110	9	6
z	26	0.0018	001111101	9	6
j	27	0.0012	001111100	9	6

code obtained by conventional HC as displayed in Table I. The resulting effective frequency of 1s of HALFHC is

$$q_{\text{HALFHC}} = 0.49985. \tag{38}$$

This is much closer to 0.5 than the value 0.45821 that resulted from conventional HC. Thus, HALFHC achieved the first objective given in (25), namely to get the frequency of 1s closer to 0.5.

Let's consider now if this has the desired effect on the effective distributions that are generated by a matcher code. For the English text [9], the resulting effective pmf $\boldsymbol{d}_{\text{eff}}$ is

$$\boldsymbol{d}_{\text{HALFHC}} = (0.38627,\ 0.41107,\ 0.20266)^T, \tag{39}$$

The resulting KL-distance and average cost are respectively given by

$$\mathrm{D}(\boldsymbol{d}_{\text{HALFHC}} \| \boldsymbol{p}^*) = 0.00048629 \tag{40}$$

$$\boldsymbol{w}^T \boldsymbol{d} = 0.20635. \tag{41}$$

Compared to HC, the KL-distance is reduced. Thus, by using HALFHC instead of conventional HC, the effective output of a matcher code is closer to the output expected under the fair bit stream assumption. The effective cost of HC exceeds the cost constraint by 2.11%, HALFHC exceeds the cost constraint $S = 0.2063$ by only 0.02%. Although both HC and HALFHC formally violate the cost constraint, the value achieved by HALFHC was adopted as a practical solution by the collaborating architects in [7], who originally formulated the cost constraint S. We can conclude that our approach of minimizing $|q - 0.5|$ leads to the desired result.

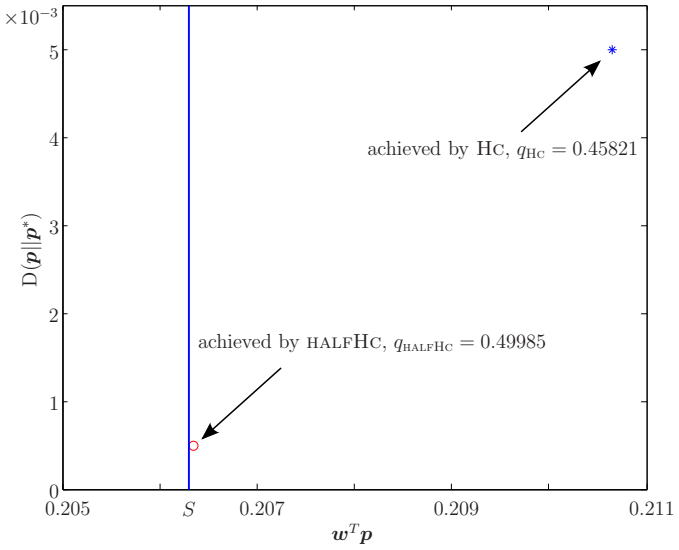

Fig. 2. Comparison between HC and HALFHC for the English text [9]. The horizontal axis corresponds to the average cost $\boldsymbol{w}^T\boldsymbol{p}$ while the vertical axis corresponds to the KL-distance $\mathrm{D}(\boldsymbol{p}\|\boldsymbol{p}^*)$. For HC, $\boldsymbol{p} = \boldsymbol{d}_{\text{HC}}$, see (9), and for HALFHC, $\boldsymbol{p} = \boldsymbol{d}_{\text{HALFHC}}$, see (39). The blue line marks the average cost constraint $S = 0.2063$ of the original design problem (5).

REFERENCES

[1] R. G. Gallager, *Principles of Digital Communication*. Cambridge University Press, 2008.

[2] G. Böcherer and R. Mathar, "Matching dyadic distributions to channels," in *Proc. Data Compression Conf.*, 2011.

[3] G. Böcherer, F. Altenbach, and R. Mathar, "Capacity achieving modulation for fixed constellations with average power constraint," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2011.

[4] T. S. Han, "Folklore in source coding: information-spectrum approach," *IEEE Trans. Inf. Theory*, vol. 51, no. 2, pp. 747–753, 2005.

[5] G. Ungerboeck, "Huffman shaping," in *Codes, Graphs, and Systems*, R. Blahut and R. Koetter, Eds. Springer, 2002, ch. 17, pp. 299–313.

[6] G. Ungerboeck and A. J. Carlson, "System and method for Huffman shaping in a data communication system," U.S. Patent US 7,460,595 B2, Dec. 2, 2008.

[7] G. Böcherer, F. Altenbach, M. Malsbender, and R. Mathar, "Writing on the facade of RWTH ICT Cubes: Cost constrained geometric Huffman coding," in *IEEE Int. Symp. Wireless Commun. Syst. (ISWCS)*, 2011. [Online]. Available: http://arxiv.org/abs/1106.5675

[8] "half Huffman coding," Jul. 2011. [Online]. Available: http://www.georg-boecherer.de/halfhc

[9] "Quotes from ingenious researchers." [Online]. Available: http://www.georg-boecherer.de/quotes.txt

[10] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 2nd ed. John Wiley & Sons, Inc., 2006.

[11] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[12] L. A. Wolsey and G. L. Nemhauser, *Integer and Combinatorial Optimization*, 1st ed. Wiley-Interscience, Nov. 1999.

[13] S. Boyd and J. Mattingley, "Branch and bound methods," Mar. 2007, notes for EE364b, Stanford University, Winter 2006-07.