HAL
archives-ouvertes.fr

A MILP approach for designing robust variable-length codes based on exact free distance computation

Hassan Hijazi, Amadou Diallo, Michel Kieffer, Leo Liberti, Claudio Weidmann

▶ **To cite this version:**

Hassan Hijazi, Amadou Diallo, Michel Kieffer, Leo Liberti, Claudio Weidmann. A MILP approach for designing robust variable-length codes based on exact free distance computation. DCC 2012, Apr 2012, Snowbird, United States. pp.257 - 266, 10.1109/DCC.2012.33 . hal-00727540

HAL Id: hal-00727540
https://hal-supelec.archives-ouvertes.fr/hal-00727540

Submitted on 3 Sep 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A MILP approach for designing robust variable-length codes based on exact free distance computation

Hassan Hijazi[a,1], Amadou Diallo[b], Michel Kieffer[b,c,*],
Leo Liberti[a], Claudio Weidmann[d]

[a] *LIX - École Polytechnique - Laboratoire d'Informatique, 91128 Palaiseau Cedex, France*
[b] *L2S - CNRS - SUPELEC - Univ Paris-Sud, 91192 Gif-sur-Yvette, France*
[c] *LTCI - CNRS - Telecom ParisTech, 75014 Paris, France*
[d] *ETIS - CNRS UMR 8051 - ENSEA - Univ Cergy-Pontoise, 95014 Cergy-Pontoise, France*

Abstract

This paper addresses the design of joint source-channel variable-length codes with maximal free distance for given codeword lengths. While previous design methods are mainly based on bounds on the free distance of the code, the proposed algorithm exploits an exact characterization of the free distance. The code optimization is cast in the framework of mixed-integer linear programming and allows to tackle practical alphabet sizes in reasonable computing time.

Keywords: joint source-channel variable-length codes, error correcting codes, mixed-integer linear programming

1. Introduction

When designing joint source-channel Variable-Length Codes (VLCs), *i.e.*, source codes robust to transmission errors, one wants to maximize the error correction capability for a given redundancy level. Redundancy is measured by the difference between average codeword length and source entropy [4], while the error correction capability is determined by the distance spectrum, mainly the free distance, see *e.g.* [2].

Early work on robust VLC design focused on minimizing the average codeword length of *reversible* VLCs, without considering error correction performance, see [13, 15, 9, 16, 17], and [11]. Techniques for evaluating the distance spectrum as well as bounds on the free distance of VLCs were introduced in [2]. These bounds have then been extensively used in [3, 8, 10, 11, 12, 14] to develop robust VLC construction heuristics. Recently, Abedini *et al.* [1] proposed an efficient SAT-based approach for designing robust VLCs with minimal redundancy properties, also based on the bounds

*Corresponding author
Email address: Michel.Kieffer@lss.supelec.fr (Michel Kieffer)

[1] Financial support by grants: Digiteo Emergence "PASO", Digiteo Chair 2009-14D "RMNCCO", Digiteo Emergence 2009-65D "ARM" is gratefully acknowledged. Michel Kieffer is partly supported by Institut Universitaire de France.

in [2]. Methods using these bounds impose a sufficient but not necessary condition to achieve a target free distance, thus they may disregard certain optimal solutions.

The aim of this paper is to introduce a mixed-integer linear programming approach to design a VLC maximizing the free distance for a given set of codeword lengths. It will be shown that prefix, suffix, as well as distance constraints may be formulated as linear inequalities involving integer-valued as well as real-valued variables. Contrary to previous approaches, the exact free distance characterization is used in the algorithm, leveraging recent results on low-complexity free distance evaluation for VLCs proposed in [5].

Section 2 introduces some notations. Efficient free distance evaluation for VLCs is recalled in Section 3. The robust VLC construction problem is then cast in the framework of mixed-integer linear programing in Section 4. The main algorithm is proposed in Section 5, before reporting numerical results in Section 6.

2. Notations

Consider a discrete memoryless source X with alphabet $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$ and corresponding probability vector $\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_n)$, where $\pi_i = \Pr(X = a_i)$. A binary VLC C for X maps $\mathcal{A}$ to a set of codewords $\mathcal{C} = \{\mathbf{c}^1, \mathbf{c}^2, \ldots, \mathbf{c}^n\}$, where $\mathbf{c}^i$ is a codeword of ℓ_i bits associated to a_i. The redundancy of code C is $\rho(C) = \bar{\ell}(C) - H(X)$, where $\bar{\ell}(C) = \sum_{i=1}^{n} \pi_i \ell_i$ denotes the average codeword length and $H(X) = -\sum_{i=1}^{n} \pi_i \log_2 \pi_i$ the source entropy [4]. The error correction performance of C is mainly determined by its free distance, defined as follows. Consider the set $\mathcal{C}^\infty = \mathcal{C} \cup \mathcal{C}^2 \cup \mathcal{C}^3 \cup \ldots$ of all finite and semi-infinite sequences of codewords. The free distance $d_{\mathrm{f}}(\mathcal{C})$ is the minimum Hamming distance $d_{\mathrm{H}}(\mathbf{s}_1, \mathbf{s}_2)$ between any two distinct sequences $\mathbf{s}_1, \mathbf{s}_2$ in $\mathcal{C}^\infty$ of equal length in bits (denoted $\ell(\mathbf{s}_1) = \ell(\mathbf{s}_2)$), that is

$$d_{\mathrm{f}}(C) = \min_{\substack{\mathbf{s}_1, \mathbf{s}_2 \in \mathcal{C}^\infty : \\ \ell(\mathbf{s}_1) = \ell(\mathbf{s}_2) \ \wedge \ \mathbf{s}_1 \neq \mathbf{s}_2}} d_{\mathrm{H}}(\mathbf{s}_1, \mathbf{s}_2). \tag{1}$$

For given redundancy ρ, one is interested in designing a code C that maximizes $d_{\mathrm{f}}(C)$. The design parameters are the length vector (the vector of codeword lengths) $\boldsymbol{\ell} = (\ell_1, \ldots, \ell_n)$ and the bit assignment of each codeword of $\mathcal{C}$.

3. Free distance evaluation

A VLC may be regarded as a nonlinear code generated by a trivial Finite-State Encoder (FSE). This section briefly recalls how FSEs provide an efficient way to evaluate the free distance of VLCs, as detailed in [5]. An FSE is a directed graph $\Gamma(\mathcal{S}, \mathcal{T})$, where $\mathcal{S} = \{s_0, s_1, \ldots, s_{k-1}\}$ is the set of vertices (states) and $\mathcal{T} = \{t_0, t_1, \ldots, t_{m-1}\}$ is the set of directed edges (transitions). For $t \in \mathcal{T}$, $\sigma(t) \in \mathcal{S}$ and $\tau(t) \in \mathcal{S}$ denote the source and target states. A label $(a_i/\mathbf{c}^i)$ is attached to each edge t_i, corresponding to the pair (input symbol/output codeword). In the simple case of a VLC, $\mathcal{S}$ contains a single state s_0 from which (to which) all transitions diverge (converge), see Figure 1(a). Any sequence of codewords in $\mathcal{C}^\infty$ may be represented by a path in Γ.

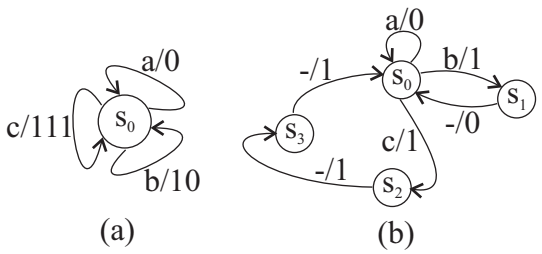

Figure 1: (a) Symbol-clock FSE and (b) bit-clock FSE corresponding to $\mathcal{A} = \{a, b, c\}$ and $\mathcal{C} = \{0, 10, 111\}$

A bit-clock FSE $\Gamma_b(\mathcal{S}_b, \mathcal{T}_b)$ is obtained by replacing each edge t_i in Γ by a sequence of transitions $t_{i,1}, t_{i,2}, \ldots, t_{i,\ell_i}$ with corresponding labels $(a_i/c_1^i), (-/c_2^i), \ldots, (-/c_{\ell_i}^i)$ and intermediate states $s_{i,1}, s_{i,2}, \ldots, s_{i,\ell_i-1}$. Each transition in $\mathcal{T}_b$ has thus a single output bit, see Figure 1(b).

Free distance computation is simplified using the Pairwise Distance Graph (PDG) $\Gamma_{\mathrm{P}}(\mathcal{S}_{\mathrm{P}}, \mathcal{T}_{\mathrm{P}})$ introduced in [5], which is derived from the product graph $\Gamma_b^2(\mathcal{S}_b^2, \mathcal{T}_b^2)$ of the bit-clock FSE. Consider the set $\mathcal{S}_{\mathrm{div}}$ of states $(s_i, s_i) \in \mathcal{S}_b^2$ from which distinct transitions are diverging, as well as the set $\mathcal{S}_{\mathrm{conv}}$ of states to which distinct transitions are converging. One obtains $\mathcal{S}_{\mathrm{P}}$ by merging all states in $\mathcal{S}_{\mathrm{div}}$ into a single state s_{div} and those in $\mathcal{S}_{\mathrm{conv}}$ into a single state s_{conv}, as well as merging symmetric states (s_i, s_j) and (s_j, s_i), $i < j$, into a single state (s_i, s_j). In the simple case of VLCs, one has $s_{\mathrm{div}} = (s_0, s_0)$ and $s_{\mathrm{conv}} = (s_0, s_0)$. Each transition $t_{ij} \in \mathcal{T}_{\mathrm{P}}$ is labeled by the Hamming distance $d_{\mathrm{H}}(\mathrm{Out}(t_i), \mathrm{Out}(t_j))$ between the output bits of $t_i \in \mathcal{T}_b$ and $t_j \in \mathcal{T}_b$.

A path $\mathbf{p}$ in Γ_{P} is an ordered sequence of transitions $\mathbf{p} = (\theta_1, \theta_2, \ldots, \theta_k)$ such that $\tau(\theta_i) = \sigma(\theta_{i+1})$, $1 \leqslant i < k$, and represents a pair of equal-length paths in Γ_b. The set of all paths from s_{div} to s_{conv} is denoted by $\mathcal{P}$. The weight $w_{\mathrm{H}}(\mathbf{p})$ of a path $\mathbf{p}$ in Γ_{P} is the sum of its edge labels. Thus $w_{\mathrm{H}}(\mathbf{p})$ equals the Hamming distance between two sequences of the same length in bits generated by the original encoder. The free distance of a code C is then the smallest weight of a path in $\mathcal{P}$, which may be computed efficiently using Dijkstra's algorithm, see [5].

The structure of the PDG for a VLC may be obtained from the length vector $\boldsymbol{\ell}$, which yields a symbolic codebook

$$\mathcal{C} = \left\{ c_1^1 c_2^1 \ldots c_{\ell_1}^1, c_1^2 c_2^2 \ldots c_{\ell_2}^2, \ldots, c_1^n c_2^n \ldots c_{\ell_n}^n \right\}. \tag{2}$$

When no bit of $\mathcal{C}$ is specified, instead of having a PDG with edges labeled by 0 or 1, one obtains symbolic labels such as $c_{j_1}^{i_1} \oplus c_{j_2}^{i_2}$ (where $\oplus$ is the exclusive-or operation), corresponding to the Hamming distance between two bits of codewords of $\mathcal{C}$; see Figure 2 for the case $\mathcal{C} = \{c_1^1, c_1^2 c_2^2, c_1^3 c_2^3 c_3^3\}$. This is an important property, since it enables one to obtain a symbolic expression for the Hamming weight of any path from s_{div} to s_{conv} as a function of bits of $\mathcal{C}$, without specifying any of those bits.

4. Mixed-Integer Linear Programming

A Mixed-Integer Linear Program (MILP) is an optimization problem described by a linear objective function and linear inequality constraints. A MILP can be written

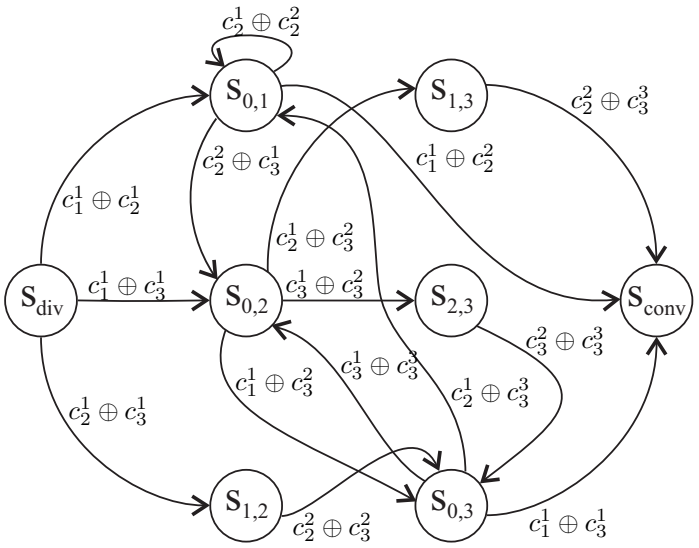

Figure 2: Pairwise distance graph derived from the B-FSE of $\mathcal{C} = \left\{c_1^1, c_2^1 c_2^2, c_3^1 c_3^2 c_3^3\right\}$

in canonical form as follows:

$$\begin{array}{ll} \max_{\mathbf{x}} & \boldsymbol{\alpha}^T \mathbf{x} \\ \text{s.t.} & \mathbf{A}\mathbf{x} \leqslant \mathbf{b}, \\ & \mathbf{x} \geqslant 0, \\ & \mathbf{x} \in \mathbb{R}^n, \mathbf{x}^I \in \mathbb{Z}^{|I|} \end{array} \qquad (M)$$

where $I \subseteq \{1, 2, \ldots, n\}$ is the set of indices corresponding to integer variables, $\boldsymbol{\alpha} \in \mathbb{Q}^n$ represents the cost vector, $\mathbf{A} \in \mathbb{Q}^{m \times n}$ is the coefficient matrix, and $\mathbf{b} \in \mathbb{Q}^n$ denotes the right hand side vector. The subvector $\mathbf{x}^I$ contains all integer entries of $\mathbf{x}$. Very efficient techniques based on continuous linear optimization theory are available to solve large-scale MILPs, see *e.g.* [18].

Given a length vector $\boldsymbol{\ell}$ corresponding to a code C, the aim in the remainder of this section is to cast the bit assignment problem into a MILP maximizing $d_{\text{f}}(\mathcal{C})$.

4.1. Objective function

The vector $\mathbf{x}^I$ in (M) contains binary decision variables, which are the bit components of the codewords in $\mathcal{C}$, as well as one integer variable d corresponding to the free distance to maximize. Thus $\mathbf{x}^I = \left(d, c_1^1, \ldots, c_{\ell_1}^1, \ldots, c_1^n, \ldots, c_{\ell_n}^n\right)^T$. Without loss of generality, one may choose $\mathbf{x}^I$ to form the first entries of $\mathbf{x}$. The cost vector is then $\boldsymbol{\alpha} = (1, 0, \ldots, 0)^T$ and the objective function $\boldsymbol{\alpha}^T \mathbf{x}$ leads to the maximization of d.

4.2. Constraints derived from bounds on the free distance

A first set of inequalities may be derived from the following bounds on free distance in [2, 3]:

$$d_{\text{b}}(\mathcal{C}) \geqslant d_{\text{f}}(\mathcal{C}) \geqslant \min(d_{\text{b}}(\mathcal{C}), d_{\text{div}}(\mathcal{C}) + d_{\text{conv}}(\mathcal{C})) \qquad (3)$$

where

$$d_{\text{b}}(\mathcal{C}) = \min\left\{d_{\text{H}}(c_1^i c_2^i \ldots c_{\ell_i}^i, c_1^j c_2^j \ldots c_{\ell_j}^j) \,:\, (\mathbf{c}^i, \mathbf{c}^j) \in \mathcal{C}^2,\, i \neq j, \text{ and } \ell_i = \ell_j\right\} \qquad (4)$$

is the *overall minimum block distance* (between codewords of the same length),

$$d_{\text{div}}(\mathcal{C}) = \min\left\{ d_{\text{H}}(c_1^i c_2^i \dots c_{\ell_j}^i, c_1^j c_2^j \dots c_{\ell_j}^j) \,:\, (\mathbf{c}^i, \mathbf{c}^j) \in \mathcal{C}^2,\, \ell_i > \ell_j \right\} \tag{5}$$

is the *minimum diverging distance* (between prefixes), and

$$d_{\text{conv}}(\mathcal{C}) = \min\left\{ d_{\text{H}}(c_{\ell_i-\ell_j+1}^i \dots c_{\ell_i}^i, c_1^j \dots c_{\ell_j}^j) \,:\, (\mathbf{c}^i, \mathbf{c}^j) \in \mathcal{C}^2,\, \ell_i > \ell_j \right\} \tag{6}$$

is the *minimum converging distance* (between suffixes).

From (3) and (4), one may deduce the following set of constraints which have to be satisfied by an optimal code:

$$\textstyle\sum_{k=1}^{\ell_i} c_k^i \oplus c_k^j \geqslant d,\ \forall (\mathbf{c}_i, \mathbf{c}_j) \in \mathcal{C}^2 \text{ such that } \ell_i = \ell_j. \tag{7}$$

As in [1], one may search for codes maximizing the free distance with bit assignments such that $d_{\text{div}}(\mathcal{C}) \geqslant \lceil d/2 \rceil$ and $d_{\text{conv}}(\mathcal{C}) \geqslant \lfloor d/2 \rfloor$, where $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ denote rounding towards $-\infty$ and ∞, respectively. From (5) and (6) one then gets the set of constraints

$$\textstyle\sum_{k=1}^{\ell_j} c_k^i \oplus c_k^j \geqslant \lceil d/2 \rceil,\ \forall (\mathbf{c}_i, \mathbf{c}_j) \in \mathcal{C}^2 \text{ such that } \ell_i > \ell_j, \tag{8}$$

and

$$\textstyle\sum_{k=1}^{\ell_j} c_{(\ell_i-k+1)}^i \oplus c_{(\ell_j-k+1)}^j \geqslant \lfloor d/2 \rfloor,\ \forall (\mathbf{c}_i, \mathbf{c}_j) \in \mathcal{C}^2 \text{ such that } \ell_i > \ell_j. \tag{9}$$

However, constraints (8) and (9) may discard many optimal codes, especially ones having a minimum codeword length $\ell_{\min}$ such that $\ell_{\min} < \lfloor d_{\text{f}}(\mathcal{C})/2 \rfloor$. For example, $\mathcal{C} = \{01, 10000010\}$ has $d_{\text{f}}(\mathcal{C}) = 6$, but would not satisfy (8) nor (9) with $d = 6$, and thus can not be obtained as a solution of algorithms imposing such constraints.

4.3. Constraints for prefix-free or suffix-free codebooks

Prefix-free codebooks are obtained by imposing that the diverging distance be at least one, *i.e.*, that

$$\textstyle\sum_{k=1}^{\ell_j} c_k^i \oplus c_k^j \geqslant 1,\ \forall (\mathbf{c}_i, \mathbf{c}_j) \in \mathcal{C}^2 \text{ such that } \ell_i > \ell_j. \tag{10}$$

Similar inequalities can be derived for suffix-free codebooks:

$$\textstyle\sum_{k=1}^{\ell_j} c_{(\ell_i-k+1)}^i \oplus c_{(\ell_j-k+1)}^j \geqslant 1,\ \forall (\mathbf{c}_i, \mathbf{c}_j) \in \mathcal{C}^2 \text{ such that } \ell_i > \ell_j. \tag{11}$$

Fix-free codebooks are then obtained by simultaneously imposing (10) and (11).

4.4. Exact free distance constraints

Consider the PDG Γ_{P} associated to some codebook $\mathcal{C}$ and let $\mathcal{P}_e$ denote the set of elementary (cycle-free) paths going from s_{div} to s_{conv}. Based on the definition of free distance one has

$$d_{\text{f}}(\mathcal{C}) \geqslant d \ \text{ if and only if } \ \forall \mathbf{p} \in \mathcal{P}_e,\ w_{\text{H}}(\mathbf{p}) \geqslant d. \tag{12}$$

Assume that a path $\mathbf{p} \in \mathcal{P}_e$ may be written as $\mathbf{p} = (\theta_1, \theta_2, \ldots, \theta_k)$, then each θ_κ is associated to a unique pair of codeword bits $(c_{j_\kappa}^{i_\kappa}, c_{j'_\kappa}^{i'_\kappa})$ such that $w_{\mathrm{H}}(\theta_\kappa) = c_{j_\kappa}^{i_\kappa} \oplus c_{j'_\kappa}^{i'_\kappa}$. Thus, the following constraint may be obtained from (12) for a given path $\mathbf{p} = (\theta_1, \theta_2, \ldots, \theta_k)$ in $\mathcal{P}_e$

$$\sum_{\kappa=1}^{k} c_{j_\kappa}^{i_\kappa} \oplus c_{j'_\kappa}^{i'_\kappa} \geqslant d. \tag{13}$$

Let us emphasize that the number of cycle-free paths in Γ_{P} grows exponentially with the size of $\mathcal{C}$. Therefore, the number of constraints (13) associated to all cycle-free paths in $\mathcal{P}_e$ may become intractable. This issue is addressed in Section 5.

4.5. Transforming exclusive-or operations

All previously-introduced constraints feature exclusive-or operations ($\oplus$). The corresponding inequalities do not fit in the MILP framework. This issue is addressed in Proposition 1, where it is shown that each exclusive-or operation can be translated into a set of linear inequalities by introducing real-valued variables.

Proposition 1. *Consider two binary variables x and y, and a real-valued variable z. Then*

$$z = x \oplus y \text{ if and only if } \begin{cases} z \geqslant x - y, \\ z \geqslant y - x, \\ z \leqslant x + y, \\ z \leqslant 2 - (x + y), \\ 0 \leqslant z \leqslant 1. \end{cases} \tag{14}$$

Proof. If $z = x \oplus y$, one may easily verify that all inequalities in (14) are satisfied. In the opposite direction, one only needs to check all possible combinations of x and y. If $(x, y) = (0, 0)$ then the third constraint in (14) induces $z \leqslant 0$. If $(x, y) = (1, 0)$ then the first constraint induces $z \geqslant 1$. If $(x, y) = (0, 1)$ then the second constraint induces $z \geqslant 1$. Finally, if $(x, y) = (1, 1)$ then the last constraint induces $z \leqslant 0$. Intersecting each of these constraints with the bounds $0 \leqslant z \leqslant 1$ completes the proof. □

Note that the artificial variables such as z in (14) required to transform exclusive-or operations into inequality constraints constitute the real-valued entries of the vector $\mathbf{x}$ introduced in (M).

5. Codebook generation algorithm

For a given length vector $\boldsymbol{\ell}$, using the results of Section 4, the search for a codebook optimizing the free distance may be cast in the framework of MILP. A first MILP may be obtained by imposing block, diverging, and converging distance constraints as suggested in [1]. The resulting model is

$$\begin{array}{ll} \max & d \\ \text{s.t.} & (7), (8), \text{ and } (9). \end{array} \tag{M_B}$$

All exclusive-or are transformed into linear inequalities as proposed in (14).

Table 1: Codebook generation algorithm

$\mathrm{CGA}\big((\ell_1,\ldots,\ell_n),d,(\hat{\mathbf{c}}^1,\ldots,\hat{\mathbf{c}}^m)\big)$	
1	Build Γ_{P} from $(\ell_1,\ldots,\ell_n)$;
2	Store in $\mathcal{P}_e^{(0)}$ the μ shortest paths in Γ_{P};
3	$k=0$; $\bar{d}_{\mathrm{f}}=\bar{d}$;
4	Define MILP $(M_P^{(k)})$ using $\bar{d}_{\mathrm{f}}$ and the constraints $c_j^i=\hat{c}_j^i$, $i=1,\ldots,m$, $j=1,\ldots,l_i$;
5	Solve $(M_P^{(k)})$ to get $\mathcal{C}^{(k)}$ and $d_{\mathrm{f}}^{(k)}$;
6	$\bar{d}_{\mathrm{f}}=d_{\mathrm{f}}^{(k)}$;
7	If $\mathcal{C}^{(k)}=\emptyset$, return $(\emptyset,0)$; End If;
8	Update $\Gamma_{\mathrm{P}}^{(k)}$ from $\mathcal{C}^{(k)}$; $\left(\widehat{\mathbf{p}}^{(k)},d^{(k)}\right)=\mathrm{Dijkstra}\left(\Gamma_{\mathrm{P}}^{(k)}\right)$;
9	If $d^{(k)}=\bar{d}_{\mathrm{f}}$, return $\left(\mathcal{C}^{(k)},d_{\mathrm{f}}^{(k)}\right)$; End If;
10	Add to $\mathcal{P}_e^{(k)}$ the path corresponding to $\widehat{\mathbf{p}}^{(k)}$ in Γ_{P} and ν additional shortest paths in Γ_{P} to get $\mathcal{P}_e^{(k+1)}$;
11	$k=k+1$; Go to 4.

A second MILP may be obtained imposing the prefix constraints (10) as well as distance constraints (13) on paths of the PDG, leading to the following model

$$\begin{array}{ll}\max & d\\ \text{s.t.} & (10),\\ & (12)\text{ for all }\mathbf{p}\in\mathcal{P}_e,\\ & d\leqslant\bar{d},\end{array}\qquad(M_P)$$

where $\bar{d}$ is a given upper bound on the free distance (this last constraint helps to reduce the size of the search space; without prior knowledge on $\bar{d}$, one may take $\bar{d}=\infty$). Nevertheless, the number of paths in $\mathcal{P}_e$ may be too large to be manageable.

The main idea of the Codebook Generation Algorithm (CGA) in Table 1 is to consider first the μ shortest paths (in terms of number of visited states) in the product graph Γ_{P} associated to the length vector $(\ell_1,\ldots,\ell_n)$ (Step 2). The μ paths are obtained via a breadth-first exploration of Γ_{P} from s_{div} to s_{conv}. In the corresponding MILP $(M_P^{(k)})$, a subset of binary variables are fixed based on the entries of $(\hat{\mathbf{c}}^1,\ldots,\hat{\mathbf{c}}^m)$. This allows to design codebooks with already determined codewords. $(M_P^{(k)})$ is then solved and the upper bound $\bar{d}_{\mathrm{f}}$ on the free distance is updated (Steps 5 and 6). If no codebook is obtained (for example when the length vector does not satisfy Kraft's inequality [4]), an empty codebook is returned (Step 7). The codebook $\mathcal{C}^{(k)}$ obtained at iteration k is used to build the PDG $\Gamma_{\mathrm{P}}^{(k)}$ on which Dijkstra's algorithm is applied to get a shortest weight path $\hat{\mathbf{p}}^{(k)}$ and the free distance $d^{(k)}$ of $\mathcal{C}^{(k)}$ (Step 8). If $d^{(k)}=\bar{d}_{\mathrm{f}}$, then $d^{(k)}$ is the optimal free distance and $\mathcal{C}^{(k)}$ is a solution (Step 9). Otherwise, $\mathcal{P}_e^{(k)}$ has to be supplemented with the path corresponding to $\hat{\mathbf{p}}^{(k)}$ in Γ_{P} and with a set of ν additional shortest paths in Γ_{P} to get $\mathcal{P}_e^{(k+1)}$ (Step 10). This algorithm is finite since the number of cycle-free paths in a graph is finite.

In order to reduce the size of MILPs solved at each step, an iterative version of CGA is proposed in Table 2. This algorithm exploits the fact that the largest free distance $d_{\mathrm{f}}(\boldsymbol{\ell}^n)$ that may be obtained for a VLC with length vector $\boldsymbol{\ell}^n=(\ell_1,\ldots,\ell_n)$

is an upper bound on the largest free distance $d_{\mathrm{f}}\left(\boldsymbol{\ell}^{n+1}\right)$ that may be obtained for a length vector $\boldsymbol{\ell}^{n+1} = (\ell_1, \ldots, \ell_n, \ell_{n+1})$.

ICGA starts with the optimization of a codebook containing only two codewords (Step 1). At iteration k, the first m codewords of the codebook obtained at iteration $k-1$ are reused to get an optimized codebook of k codewords (Step 4). If the free distance $d_{\mathrm{f}}^{(k)}$ obtained at iteration k, using the $m = k-1$ codewords from iteration $k-1$, is equal to its upper bound $d_{\mathrm{f}}^{(k-1)}$, the obtained codebook is again optimal. If $d_{\mathrm{f}}^{(k)} < d_{\mathrm{f}}^{(k-1)}$, the number of reused codewords is decreased and CGA is restarted until $d_{\mathrm{f}}^{(k)} = d_{\mathrm{f}}^{(k-1)}$ or $m = 0$. In the latter case, no previously obtained codeword is used, indicating that the additional length ℓ_k reduces the free distance of the best VLC.

Table 2: Iterative codebook generation algorithm

ICGA$(\ell_1, \ldots, \ell_n)$	
1	$((\mathbf{c}^1, \mathbf{c}^2), d_{\mathrm{f}}) = \mathrm{CGA}\left((\ell_1, \ell_2), \emptyset\right)$;
2	For $k = 3$ to n
3	$m = k - 1$;
4	$\left((\mathbf{c}^1, \ldots, \mathbf{c}^k), d_{\mathrm{f}}^{(k)}\right) = \mathrm{CGA}\left((\ell_1, \ldots, \ell_k), d_{\mathrm{f}}^{(k-1)}, (\mathbf{c}^1, \ldots, \mathbf{c}^m)\right)$;
5	If $d_{\mathrm{f}}^{(k)} < d_{\mathrm{f}}^{(k-1)}$ and $m > 0$
6	$m = m - 1$; Go to 4.
7	End If;
8	End For;
9	Return $\left((\mathbf{c}^1, \ldots, \mathbf{c}^n), d_{\mathrm{f}}^{(n)}\right)$;

6. Numerical experiments

Experimental results were obtained for alphabets of different sizes. All MILPs were solved using the Branch & Cut algorithm implemented in Cplex 12.2 [7] and ran on an Intel Xeon at 2.66 Ghz.

Table 3 provides CPU computing times (in seconds) for various codebook generation algorithms. ICGA with $\mu = 10$ and $\nu = 500$ (column T_A) is compared to a MILP with model (M_B) (Column T_{M_B}) and to a Branch & Cut method introduced in [6] (Column T_D). The optimal free distance obtained solving (M_B) is in Column d_{M_B}, that of ICGA is in Column d_A. If a solver is unable to prove optimality in less than two hours, lower and upper bounds are reported. For each instance, the number of nodes (resp. edges) of the corresponding PDG is given in Column $|\mathcal{S}_P|$ (resp. $|\mathcal{T}_P|$). The total number of paths generated by ICGA is reported in column $|\mathcal{P}|$.

For all length vectors of Table 3, ICGA was able to generate a codebook with $d_{\mathrm{f}} = 7$. The values of the free distance reported in column d_{M_B} illustrate the sub-optimality of the model imposing converging and diverging distance constraints (M_B). Moreover, computing time performance shows the efficiency of the iterative algorithm which converges in few minutes compared to the solution of (M_B) and the approach provided in [6], both becoming intractable even for small alphabets.

The best codebook for the 26-symbol English alphabet we were able to obtain by ICGA with $d_{\mathrm{f}} = 7$ is presented in Table 4. Its length vector was obtained by

trial-and-error, which leaves some space for further improvements. This codebook is not a solution of (M_B). Considering the probability vector given in [2], it has an average codeword length of 10.11. To the best of our knowledge, this is the best performance for an error-correcting VLC with $d_{\mathrm{f}} = 7$. In [10], an optimized heuristic returns a solution with an average length of 10.738 within 14 hours. Similarly, the best codebook we were able to design with $d_{\mathrm{f}} = 5$ has an average codeword length of 8.158, which is less than 8.4752 reported in [10].

Since the optimal solution of $(M_P^{(k)})$ constitutes an upper bound for the value of the best free distance, one can think of an enumeration technique for finding dominant length vectors for error-correcting VLCs as suggested in [1], based on these tight bounds. In our experiments, the solver provided a tight upper bound in an average computing time of 3 seconds on all instances.

7. Conclusion

This paper provides an efficient variable-length error-correcting code design technique optimizing the free distance. The design problem is cast in the framework of mixed-integer linear programming. It involves exact free distance evaluation in the design phase, and is thus able to provide an optimal codebook with respect to free distance for any given length vector.

With the proposed tools, one may determine dominant length vectors for the exact free distance criterion, instead of the lower bounds used in [1]. This allows to find the shortest average codeword length for a given source and a desired free distance. The results presented in Section 6 may thus be improved accordingly.

Table 3: Computing times (in s) and free distances for different codebook generation algorithms as a function of the size of the alphabet

n	ℓ	d_{M_B}	T_{M_B}	T_D	d_A	T_A	$\lvert\mathcal{S}_P\rvert$	$\lvert\mathcal{T}_P\rvert$	$\lvert\mathcal{P}\rvert$
4	(3,7,8,9)	6	**0.03**	23.6	7	4.3	278	351	511
5	(3,7,8,9,11)	5	**0.12**	1232	7	9	563	703	1012
6	(3,7,8,9,11,12)	5	**0.4**	$> 2h$	7	32	992	1225	1012
7	(3,7,8,9,11,12,13)	5	**0.4**	$> 2h$	7	33	1598	1953	511
8	(3,7,8,9,11,12,13,14)	4	97	$> 2h$	7	**35**	2417	2926	511
9	(3,7,8,9,11,12,13,14,15)	4	336	$> 2h$	7	**36**	3488	4186	1012
10	(2@7,8,9,2@10,4@11)	6	143	$> 2h$	7	**3**	3657	4465	511
11	(2@7,8,9,2@10,4@11,12)	(6,7)	$> 2h$	$> 2h$	7	**4**	4658	5671	511
12	(2@7,8,9,2@10,4@11,2@12)	6	182	$> 2h$	7	**7**	5780	7021	1012
13	(7,8,9,2@10,4@11,4@12)	6	385	$> 2h$	7	**9**	7023	8515	511
14	(7,8,9,2@10,4@11,4@12,13)	(6,7)	$> 2h$	$> 2h$	7	**15**	8387	10153	1513
15	(7,8,9,2@10,4@11,4@12,2@13)	(6,7)	$> 2h$	$> 2h$	7	**25**	10013	12090	2014
16	(7,8,9,2@10,4@11,4@12,3@13)	(6,7)	$> 2h$	$> 2h$	7	**51**	11783	14196	3517
17	(7,8,9,2@10,4@11,4@12,4@13)	(6,7)	$> 2h$	$> 2h$	7	**62**	13697	16471	1513
18	(7,8,9,2@10,4@11,4@12,5@13)	(6,7)	$> 2h$	$> 2h$	7	**80**	15755	18915	2014
19	(7,8,9,2@10,4@11,4@12,6@13)	(6,7)	$> 2h$	$> 2h$	7	**157**	18338	21945	5020
20	(7,8,9,2@10,4@11,4@12,6@13,14)	(6,7)	$> 2h$	$> 2h$	7	**263**	21117	25200	5521
21	(7,8,9,2@10,4@11,4@12,6@13,2@14)	(6,7)	$> 2h$	$> 2h$	7	**278**	24092	28680	1012
22	(7,8,9,2@10,4@11,4@12,6@13,3@14)	(6,7)	$> 2h$	$> 2h$	7	**352**	27263	32385	3517
23	(7,8,9,2@10,4@11,4@12,6@13,4@14)	(6,7)	$> 2h$	$> 2h$	7	**426**	30630	36315	3016
24	(7,8,9,2@10,4@11,4@12,6@13,5@14)	(6,7)	$> 2h$	$> 2h$	7	**567**	34193	40470	4519
25	(7,8,9,2@10,4@11,4@12,6@13,5@14,15)	(6,7)	$> 2h$	$> 2h$	7	**691**	37952	44850	3517
26	(7,8,9,2@10,4@11,4@12,6@13,5@14,2@15)	(6,7)	$> 2h$	$> 2h$	7	**811**	41907	49455	3016

Table 4: VLC for the English alphabet with $d_{\mathrm{f}} = 7$ and $\bar{\ell} = 10.11$.

Symbol	Probability	ℓ	Codeword	Symbol	Probability	ℓ	Codeword
$a_1 = E$	$p_{a_1} = 0.1270$	7	1010100	$a_{14} = M$	$p_{a_{14}} = 0.0241$	12	001110111111
$a_2 = T$	$p_{a_2} = 0.0906$	7	0101011	$a_{15} = W$	$p_{a_{15}} = 0.0236$	13	0000000100000
$a_3 = A$	$p_{a_3} = 0.0817$	8	00011101	$a_{16} = F$	$p_{a_{16}} = 0.0223$	13	1100010001111
$a_4 = O$	$p_{a_4} = 0.0751$	9	111100010	$a_{17} = G$	$p_{a_{17}} = 0.0202$	13	1110111011000
$a_5 = I$	$p_{a_5} = 0.0697$	10	0010010011	$a_{18} = Y$	$p_{a_{18}} = 0.0197$	13	0111101000101
$a_6 = N$	$p_{a_6} = 0.0674$	10	1100001000	$a_{19} = P$	$p_{a_{19}} = 0.0193$	15	001100101000001
$a_7 = S$	$p_{a_7} = 0.0633$	11	00100011010	$a_{20} = B$	$p_{a_{20}} = 0.0149$	15	111001000100010
$a_8 = H$	$p_{a_8} = 0.0609$	11	11011111000	$a_{21} = V$	$p_{a_{21}} = 0.0098$	15	010000001001110
$a_9 = R$	$p_{a_9} = 0.0599$	11	10110100101	$a_{22} = K$	$p_{a_{22}} = 0.0077$	15	110010111010001
$a_{10} = D$	$p_{a_{10}} = 0.0425$	11	01001000111	$a_{23} = J$	$p_{a_{23}} = 0.0015$	15	101101110101100
$a_{11} = L$	$p_{a_{11}} = 0.0403$	12	010110000000	$a_{24} = X$	$p_{a_{24}} = 0.0014$	15	001011101011010
$a_{12} = C$	$p_{a_{12}} = 0.0278$	12	100001101000	$a_{25} = Q$	$p_{a_{25}} = 0.001$	15	111110101110110
$a_{13} = U$	$p_{a_{13}} = 0.0276$	12	011001010011	$a_{26} = Z$	$p_{a_{26}} = 0.0007$	15	011011110000111

References

[1] N. Abedini, S. P. Khatri, and S. A. Savari. A SAT-based scheme to determine optimal fix-free codes. In *Proc. Data Compression Conference (DCC)*, pages 169–178, 2010.

[2] V. Buttigieg. *Variable-Length Error Correcting Codes*. Ph.D. dissertation, University of Manchester, Manchester, U.K., 1995.

[3] V. Buttigieg and P.G. Farrell. Variable-length error-correcting codes. *IEE Proceedings on Communications*, 147(4):211–215, Aug. 2000.

[4] T. M. Cover and J. M. Thomas. *Elements of Information Theory*. Wiley, New-York, 1991.

[5] A. Diallo, C. Weidmann, and M. Kieffer. Efficient computation and optimization of the free distance of variable-length finite-state joint source-channel codes. *IEEE Trans. Commun.*, 59(4):1043–1052, April 2011.

[6] A. Diallo, C. Weidmann, and M. Kieffer. New free distance bounds and design techniques for joint source-channel variable-length codes. Submitted to IEEE Trans. Commun., October 2011.

[7] IBM. *ILOG CPLEX 12.2 User's Manual*. IBM, 2010.

[8] K. Lakovic and J. Villasenor. On design of error-correcting reversible variable length codes. *IEEE Commun. Lett.*, 6(8):337–339, 2002.

[9] K. Lakovic and J. Villasenor. An algorithm for construction of efficient fix-free codes. *IEEE Commun. Lett.*, 7(8):391–393, 2003.

[10] C. Lamy and J. Paccaut. Optimised constructions for variable-length error correcting codes. In *Proc. Information Theory Workshop*, pages 183–186, 2003.

[11] C.-W. Lin, J.-L. Wu, and Y.-J. Chuang. Two algorithms for constructing efficient Huffman-code based reversible variable length codes. *IEEE Trans. Commun.*, 56(1):81–89, 2008.

[12] R. Maunder and L. Hanzo. Genetic algorithm aided design of component codes for irregular variable length coding. *IEEE Trans Commun*, 57(5):1290 –1297, May 2009.

[13] Y. Takishima, M. Wada, and M. Murakimi. Reversible variable length codes. *IEEE Trans. on Commun*, 43(2/3/4):158–162, 1995.

[14] R. Thobaben and J. Kliewer. An efficient variable-length code construction for iterative source-channel decoding. *IEEE Trans. Commun.*, 57(7):2005–13, 2009.

[15] C.-W. Tsai and J.-L. Wu. On-constructing the Huffman-codes based reversible variable-length codes. *IEEE Trans. Commun.*, 49(9):1506–1509, 2001.

[16] H.-W. Tseng and C.-C. Chang. Construction of symmetrical reversible variable length codes using backtracking. *The Computer Journal*, 46(1):100–105, 2003.

[17] J. Wang, L.-L. Yang, and L Hanzo. Iterative construction of reversible variable-length codes and variable-length error-correcting codes. *IEEE Commun Letters*, 8(11):671– 673, 2004.

[18] L. A. Wolsey and G. L. Nemhauser. *Integer and Combinatorial Optimization*. Wiley-Interscience, 1999.