Meta-Heurísticas para Geração Automática de Sistemas Corretores de Erros Baseados em Codificação Convolucional*

Lucas F. Muniz[1], **Carla N. Lintzmayer**[1], **Denis G. Fantinato**[1]

[1]Centro de Matemática, Computação e Cognição
Universidade Federal do ABC (UFABC) – Santo André – SP – Brazil

l.muniz@aluno.ufabc.edu.br,{carla.negri,denis.fantinato}@ufabc.edu.br

Abstract. *Due to the wide use of digital systems, bit error control is an important task. Convolutional codes are error correcting codes widely used because of their efficiency. However, the large number of parameters involved in their generation results in a problem with high complexity. In this work, we used the metaheuristics GA and BRKGA to search for efficient coding systems, which showed relevant results in our simulations.*

Resumo. *Devido ao grande uso de sistemas digitais, o controle de erros sobre os bits é uma tarefa essencial. Códigos convolucionais são códigos corretores de erros amplamente utilizados em várias aplicações devido à sua eficácia. No entanto, o grande número de parâmetros envolvendo sua geração pode gerar um problema de elevada complexidade. Neste trabalho, usamos as meta-heurísticas GA e BRKGA para a busca por sistemas de codificação eficientes, que apresentaram relevantes resultados nos cenários de simulação.*

1. Introdução

O avanço tecnológico dos últimos anos expandiu o uso de tecnologias digitais em diversos setores, o que tornou importante a preocupação com o controle de erros. Uma das formas de controle de erros é a codificação da informação. Em um de seus trabalhos mais notórios, Shannon estabelece um limite teórico para a maior eficiência possível de um codificador, mas não define qual método de codificação é capaz de realizá-lo [Shannon 1948]. Apesar de ainda não existir uma codificação que alcance esse limite, diversos métodos eficientes já foram propostos. Dentre eles, está a codificação convolucional [Huffman and Pless 2010], que introduz redundância ao combinar elementos da própria sequência. Entretanto, o desempenho dessa codificação depende de parâmetros que elevam consideravelmente sua complexidade. Por isso, há necessidade de busca por um codificador/decodificador de maior eficiência, isto é, que seja capaz de possuir uma estrutura simples e, ao mesmo tempo, de reduzir o número de erros ao máximo.

Dessa forma, neste trabalho, duas meta-heurísticas foram utilizadas para a busca de esquemas de codificação convolucionais com baixa taxa de erro e complexidade estrutural, o Algoritmo Genético (GA) e o *Biased Random-Key Genetic Algorithm* (BRKGA). O desempenho dos codificadores encontrados será avaliado e comparado em cenários com canal binário simétrico. Destacamos que não há na literatura trabalhos que usem meta-heurísticas para buscar por codificadores usando a decodificação por lógica majoritária.

*Esse trabalho foi financiado pela Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), processo nº 2019/16997-0.

2. Códigos convolucionais

Um sistema de transmissão de dados consiste em três partes. Um *codificador*, que recebe uma *mensagem* $\mathbf{u}$ e a transforma em uma *palavra de código* $\mathbf{v}$. Um *canal*, que recebe $\mathbf{v}$ para transmissão, mas pode alterar alguns bits devido a ruídos e, por isso, devolve uma sequência $\mathbf{r}$. E um decodificador, que recebe $\mathbf{r}$ e a transforma em $\hat{\mathbf{u}}$. Para que $\hat{\mathbf{u}}$ seja o mais próximo de $\mathbf{u}$ possível, bons esquemas de codificação/decodificação são necessários.

Um *código convolucional* é um codificador que insere bits de redundância para permitir a correção de erros. Ele separa $\mathbf{u}$ em blocos de k bits, que são codificados em blocos com n bits. Cada bloco codificado depende dos k bits do bloco original e de m blocos de mensagem anteriores, sendo m chamada *ordem de memória* do código. Assim, um código convolucional é caracterizado por (n,k,m) [Lin and Costello 2004]. A codificação de uma mensagem é definida por *sequências geradoras* $\mathbf{g}_\ell^{(j)} = (g_{\ell,0}^{(j)}, \ldots, g_{\ell,m}^{(j)})$, para $1 \leq j \leq n$ e $1 \leq \ell \leq k$, que relacionam por OU-exclusivo os registradores de memória da codificação, que irão gerar o bit codificado. Como a codificação de um bloco de mensagem depende dos m últimos blocos codificados, ela é caracterizada por uma unidade de tempo t, que denota o estado do sistema de codificação. Assim, cada bloco de mensagem e de palavra de código processados em um tempo t na codificação são denotados por u_t e v_t, respectivamente. Assim, a operação de convolução no tempo $t \geq 0$ será dada por $v_t^{(j)} = \sum_{i=0}^{m} \sum_{\ell=1}^{k} u_{t-i}^{(\ell)} g_{\ell,i}^{(j)}$, para $1 \leq j \leq n$, sendo a soma e multiplicação definidas pelas operações lógicas OU-exclusivo e E, respectivamente – efetuadas em módulo-2. Neste trabalho foram gerados códigos convolucionais sistemáticos, que preservam os k bits originais do bloco de mensagem e codificam os $n-k$ bits redundantes.

Para a decodificação, o algoritmo de *lógica majoritária* (ML) possui desempenho sub-ótimo em relação ao conhecido algoritmo de Viterbi, porém sua implementação é simples e sua complexidade é menor para códigos que utilizam grande quantidade de registradores e, por esse motivo, o utilizamos neste trabalho. Basicamente, o ML faz uso de uma *sequência de erro* $\mathbf{e}$, que é tal que $\mathbf{r} = \mathbf{v} + \mathbf{e}$. A partir de $\mathbf{r}$, definimos uma *sequência síndrome* $\mathbf{s}$, dada pela operação $\mathbf{s} = \mathbf{r}\mathbf{H}^T$, sendo $\mathbf{H}$ a *matriz de checagem de paridade* escrita em função das sequências geradoras do codificador [Lin and Costello 2004]. A matriz $\mathbf{H}$ é tal que, caso uma sequência $\mathbf{v}$ pertença à codificação, vale que $\mathbf{v}\mathbf{H}^T = 0$. Assim, $\mathbf{s} = (\mathbf{v} + \mathbf{e})\mathbf{H}^T = \mathbf{v}\mathbf{H}^T + \mathbf{e}\mathbf{H}^T = \mathbf{e}\mathbf{H}^T$, e temos que $\mathbf{s}$ depende apenas de $\mathbf{e}$, e não da palavra de código transmitida [Lin and Costello 2004]. Com base nisso, emprega-se o conceito de soma de checagem de paridade ortogonal. Cada bit ou soma de bits de síndrome representa uma soma dos bits de erro do canal e são chamados de *soma de checagem de paridade*. Se houver erros em $\mathbf{r}$, então alguns bits de síndrome terão valor 1. A partir de um conjunto de J somas ortogonais sobre o bit $e_i^{(\ell)}$ localizado na posição ℓ do bloco i, para $1 \leq \ell \leq k$, definimos $t_{\mathbf{ML}} = \lfloor J/2 \rfloor$ como a *regra de decodificação de lógica majoritária*. Se mais de $t_{\mathbf{ML}}$ somas tiverem valor 1, então o valor do bit de erro estimado será $\hat{e}_i^{(\ell)} = 1$. A recuperação do bit será efetuada pela operação $\hat{u}_i^{(\ell)} = r_i^{(\ell)} + \hat{e}_i^{(\ell)}$. A sequência decodificada $\hat{\mathbf{u}} = (\hat{u}_0, \ldots, \hat{u}_i)$, para $i \in \mathbb{N}$, em que $\hat{u}_i = (\hat{u}_i^{(1)}, \ldots, \hat{u}_i^{(\ell)})$, para $1 \leq \ell \leq k$. Se há $t_{\mathbf{ML}}$ ou menos erros nos bits checados pelas J somas de checagem, então $\hat{e}_i^{(\ell)}$ é estimado corretamente. Vale ressaltar que as somas dos bits de erro do canal (bits síndrome) são definidas pelas sequências geradoras, logo, codificação e decodificação dependem diretamente dessas sequências.

3. Resultados

Dado que a estrutura dos códigos convolucionais se encontram no domínio binário, as meta-heurísticas populacionais GA e BRKGA foram escolhidas. Todos os algoritmos foram implementados em linguagem *Julia*[1]. As implementações do GA e do BRKGA seguiram as implementações clássicas [Martí et al. 2018, Boussaïd et al. 2013].

Para simular os códigos convolucionais, foi implementado um sistema de transmissão baseado nas três componentes básicas. O codificador utiliza um código dado pelo resultado das meta-heurísticas. O canal altera um bit com probabilidade p, de forma que a probabilidade de um bit em $\mathbf{e}$ ser 1 é p, e é $1-p$ caso contrário. O decodificador utiliza o método ML. Para avaliar a propriedade de correção de erros de um código C, foi criada uma medida de acúmulo de erro $E(C, \mathbf{u}) = \int_0^1 t_e(C, \mathbf{u}, p)dp$, em que $\mathbf{u}$ é uma sequência de bits, p é a probabilidade de erro inserido pelo canal, e $t_e(C, \mathbf{u}, p) = N_e(C, \mathbf{u}, p)/N$ é a taxa de erro medida, sendo $N_e(C, \mathbf{u}, p)$ o número de erros aferidos em $\mathbf{v}$, a codificação de $\mathbf{u}$ em C, após passar pelo canal, e N o total de bits transmitidos. Essa medida indica a capacidade de correção de erros de um código C, pois $E(C, \mathbf{u})$ representa a variação total dos erros introduzidos pelo canal e não recuperados pelo código. Dessa forma, valores baixos de $E(C, \mathbf{u})$ indicam que C possui boa capacidade de correção. A medida $E(C, \mathbf{u})$ foi calculada numericamente utilizando a regra dos trapézios.

Para as buscas, foram geradas sequências de bits $\mathbf{u}$ aleatórias com 1500 bits cada, e p foi variado em passos de 0.05 no intervalo $[0, 1]$. Os parâmetros dos códigos convolucionais (n,k,m) gerados pelas meta-heurísticas foram definidos nos intervalos $2 \leq n \leq 20$ e $1 \leq k, m \leq 20$, para garantir maior velocidade de execução e limitar o espaço de busca. Utilizando a medida $E(C, \mathbf{u})$, criamos cinco funções objetivos, que tinham o propósito de guiar as buscas por codificadores/decodificadores que possuíssem baixa complexidade estrutural (a estrutura de codificação possui complexidade espacial $O(nmk)$) e baixo acúmulo de erro $E(C, \mathbf{u})$. A função objetivo que obteve as melhores soluções foi $f(C) = R/(E(C, \mathbf{u}) + nkm)$, em que $R = k/n$ é a taxa de codificação.

A implementação do GA possui os parâmetros (n_G, t_p, p_m, p_p), em que $n_G \in [50, 200]$ é a quantidade de gerações, $t_p \in [50, 150]$ é o tamanho da população, $p_m \in [0.2, 0.4]$ é a porcentagem de indivíduos que sofrerão mutação na população, e $p_p \in [0.1, 0.3]$ é a porcentagem de indivíduos que serão selecionados para cruzamento. Os cromossomos são formados por $(n-k)k$ sequências de $m+1$ bits (código convolucional sistemático), que representam as sequências geradoras, e também possuem campos para os parâmetros (n,k,m). Os parâmetros que mostraram melhores resultados nas buscas foram (50, 150, 0.2, 0.1). O melhor código convolucional C_1 obtido pelo GA possui estrutura (2,1,1) e é definido pelas sequências geradoras $\mathbf{g}_1^{(1)} = (1, 0)$, $\mathbf{g}_1^{(2)} = (1, 1)$. A capacidade de correção desse código apresentou um acúmulo médio de erro de $E(C_1, \mathbf{u}) \approx 0.4$.

A implementação do BRKGA possui os parâmetros $(n_G, t_p, \rho_e, p_m, p_e)$, em que n_G e t_p são como no GA, $\rho_e \in [0.8, 0.9]$ é a probabilidade do indivíduo gerado herdar uma chave do pai que pertence à elite, $p_m \in [0.2, 0.4]$ é a porcentagem de mutantes que serão adicionados à população da próxima geração, e $p_p \in [0.1, 0.3]$ é a porcentagem de indivíduos que serão selecionados para a população elite. Os cromossomos do BRKGA são formados por $(n-k)k$ números reais, que representam sequências de $m+1$

[1]Endereço do site oficial da linguagem: `https://julialang.org`.

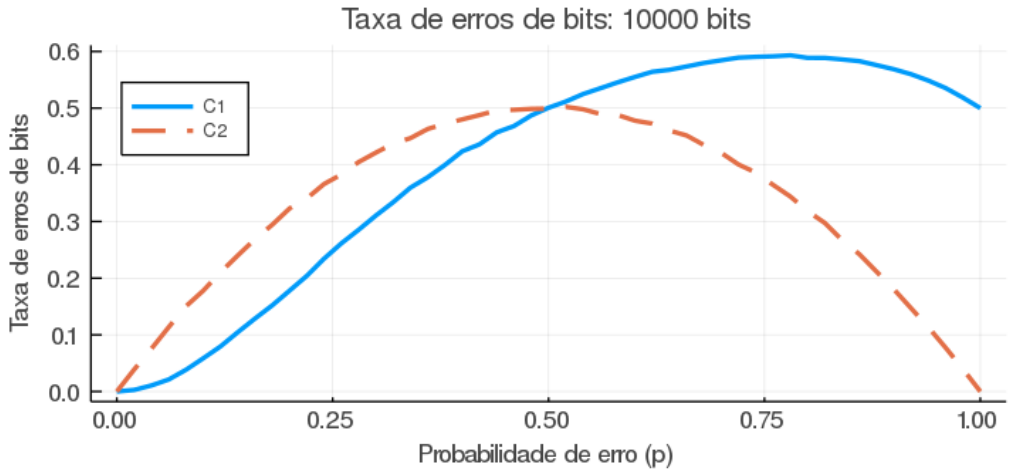

Figura 1. Taxa de erro de C_1 (linha contínua) e C_2 (tracejado) em função de p.

bits (sequências geradoras), e também possuem campos para os parâmetros (n,k,m). Os parâmetros que mostraram melhores resultados nas buscas foram (50, 50, 0.85, 0.2, 0.2). O melhor código convolucional C_2 obtido pelo BRKGA possui estrutura (3,2,1) e é definido pelas sequências geradoras $\mathbf{g}_1^{(1)} = (1,0)$, $\mathbf{g}_1^{(2)} = (0,0)$, $\mathbf{g}_1^{(3)} = (0,1)$, $\mathbf{g}_2^{(1)} = (0,0)$, $\mathbf{g}_2^{(2)} = (1,0)$, $\mathbf{g}_2^{(3)} = (0,1)$. A capacidade de correção desse código apresentou um acúmulo médio de erro de $E(C_2, \mathbf{u}) \approx 0.33$.

Para testar os códigos C_1 e C_2, recalculamos as taxas de erros médias com novas sequências de 10000 bits e p variando em passos de 0.02 no intervalo $[0, 1]$, conforme apresentado na Figura 1. Observe que para $p \in [0, 0.5)$, C_1 apresentou menor taxa de erro do que C_2 (maior capacidade de correção), o que se inverteu no intervalo $(0.5, 1]$.

4. Conclusão

Neste trabalho propusemos o uso de meta-heurísticas para gerar códigos convolucionais eficientes e, como mostrado na seção de resultados, tanto o GA quanto o BRKGA conseguiram encontrar soluções com baixa complexidade estrutural e baixa taxa de erro. Percebeu-se que as soluções encontradas dependeram diretamente da função objetivo. Logo, a busca por outras funções objetivo pode ser importante para encontrar códigos convolucionais ainda mais eficientes.

Referências

Boussaïd, I., Lepagnot, J., and Siarry, P. (2013). A survey on optimization metaheuristics. *Information Sciences*, 237:82 – 117.

Huffman, W. C. and Pless, V. (2010). *Fundamentals of error-correcting codes*. Cambridge university press.

Lin, S. and Costello, D. J. (2004). *Error control coding*. Pearson Education India.

Martí, R., Pardalos, P. M., and Resende, M. G. C., editors (2018). *Handbook of Heuristics*. Springer International Publishing.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423.