Vetor de sufixos métrico compacto*

Frederico R. Rosa[1], **Felipe A. Louza**[1], **Humberto L. Razente**[1]

[1]Universidade Federal de Uberlândia (UFU)
Uberlândia – MG – Brasil

{frederico.rosa,louza,humberto.razente}@ufu.br

Abstract. *In this paper we present a compact data structure for the Metric Suffix Array (MSA), a permutation-based index for similarity queries. The proposed data structure uses variable-length integer encoding to store the values in the MSA using less memory, which are decoded during the queries. The compact structure is built directly from the data, where it is not necessary to compute the complete MSA beforehand to obtain its compact representation, which allows the indexing of larger volumes of data. In experiments with high-dimensional features extracted from 20 million images, our compact representation used* 33% *of the original method's total memory, increasing the query execution time by a factor of* 3.4.

Resumo. *Neste artigo apresentamos uma estrutura de dados compacta para o vetor de sufixos métrico (MSA), um índice baseado em permutações para consultas por similaridade. A estrutura proposta utiliza codificação de inteiros de tamanho variável para armazenar os valores do MSA, os quais são decodificados durante as consultas. A estrutura compacta é construída diretamente dos dados, sem a necessidade de computar o MSA para depois compactá-lo, o que permite a indexação de maiores volumes de dados. Em experimentos com dados de alta dimensionalidade extraídos de 20 milhões de imagens, nossa representação compacta utilizou* 33% *da memória total do método original, com aumento no tempo das consultas por um fator de* $3{,}4$.

1. Introdução

A recuperação de dados baseada em similaridade é uma tarefa importante com aplicações em diversas áreas [Lew et al. 2006]. Nas últimas décadas, diversos trabalhos propuseram estruturas de indexação para manipular dados com alta dimensionalidade visando responder consultas de modo exato, no entanto, essas estruturas possuem limitações quanto à escalabilidade. Para lidar com essa questão, métodos para consultas aproximadas baseadas em similaridade têm sido propostos, e entre eles, os métodos de indexação baseados em permutações, os quais calculam uma distância aproximada entre dois objetos o_i e o_j de um conjunto D a partir de suas listas ordenadas, L_{o_i} e L_{o_j}, que indicam quais os objetos de um conjunto de referências R estão mais próximos de o_i e o_j, respectivamente [Chávez et al. 2008].

O vetor de sufixos métrico (MSA) [Mohamed and Marchand-Maillet 2013] é um método de indexação baseado em permutações que armazena em um vetor de inteiros

*Os autores agradecem o financiamento do Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq (projeto 406418/2021-7).

valores referentes às listas L_{o_i} de cada objeto $o_i \in D$. Os valores no MSA são particionados em *buckets* a partir de cada objeto de referência em R. Em particular, os valores dentro de cada *bucket*, estão sempre em ordem crescente, o que permite que o uso de estratégias mais eficientes para o armazenamento do MSA, como a codificação de inteiros de tamanho variável [Navarro 2016].

Neste artigo apresentamos uma estrutura de dados compacta para o vetor de sufixos métrico baseado no armazenamento das diferenças entre os valores consecutivos em cada *bucket* utilizando codificação de inteiros de tamanho variável. Os valores do MSA são decodificados durante as consultas com um custo adicional constante, dado que o acesso ao MSA é feito sequencialmente durante essa etapa, não é necessario realizar decodificações em posições aleatórias. A estrutura compacta proposta é construída diretamente a partir dos dados, sem a necessidade de computar o MSA primeiro para depois compactá-lo, o que permite a indexação de maiores volumes de dados.

2. Definições e trabalhos relacionados

Seja um conjunto de referências $R = \{r_0, r_1, ...r_n\}$ e um domínio D de objetos, onde cada objeto $o_i \in D$ é representado por uma lista ordenada L_{o_i}, tal que esta lista para cada objeto contenha as referências ordenadas por proximidade com o respectivo objeto, segundo uma função de distância. Logo, $P\left(L_{o_i,r_j}\right)$ retorna a posição da referência r_j na lista ordenada L_{o_i} do objeto o_i. Portanto, se $P\left(L_{o_i,r_j}\right) = 4$, significa que r_j é a quarta referência mais próxima de o_i. Para uma dada consulta q, uma lista L_q é computada e a similaridade dos objetos em D é calculada pela comparação das listas ordenadas usando a métrica *Spearman Footrule Distance* [Fagin et al. 2003], que é dada pela Equação 1.

$$\mathsf{SFD}\left(o_i, q\right) = \sum_{r \in R} \left| P\left(L_{o_i}, r\right) - P\left(L_q, r\right) \right| \tag{1}$$

A utilização desta métrica visa o ganho em desempenho de tempo em detrimento de perda de precisão com relação ao método direto euclidiano, economizando com operações computacionalmente custosas como potências e raízes.

Seja $T[1,n] = T[1]T[2]\dots T[n]$ uma cadeia de caracteres de tamanho n, sobre um alfabeto Σ, tal que $T[i,j] = T[i]T[i+1]\dots T[j]$ é uma subcadeia de T, $T[1,i]$ é um prefixo e $T[j,n]$ é um sufixo de T, para $1 \leq i,j \leq n$. O vetor de sufixos [Manber and Myers 1993] para a cadeia T é um vetor de inteiros (SA) que armazena uma permutação de $\{1,\dots,n\}$ tal que $\mathsf{SA}[i] = j$ se o sufixo $T[j,n]$ é o i-ésimo menor sufixo de T.

2.1. Vetor de sufixos métrico

O vetor de sufixos métrico (MSA) [Mohamed and Marchand-Maillet 2013] armazena em um vetor de inteiros as listas L_{o_i} de cada objeto $o_i \in D$ a partir da seguinte ordenação. Inicialmente, cada $L_{o_i} = [i_1, i_2, \dots, i_{N_{\mathsf{refs}}}]$ é representada como uma cadeia de caracteres $S_{o_i} = i_1 i_2 \dots i_{N_{\mathsf{refs}}}$, tal que o alfabeto $\Sigma = \{1, 2, \dots, N_{\mathsf{refs}}\}$. Cada cadeia S_{o_i} é concatenada em $S = S_{o_1} S_{o_2} \dots S_{o_{N_{\mathsf{objs}}}}$. Em seguida, todos os sufixos da cadeia S são ordenados apenas a partir do seu primeiro caractere. A lista da posição inicial desses sufixos (parcialmente) ordenados é armazenada em um vetor de inteiros, chamado de MSA, de tamanho

$N = N_{\mathsf{refs}} \times N_{\mathsf{objs}}$. Como resultado, o MSA pode ser particionado em *buckets*, um para cada objeto de referência em R.

A consulta por similaridade aos k-vizinhos mais próximos no MSA é realizada primeiro obtendo a lista ordenada L_q do objeto de consulta q com relação às referências. Em seguida, cada *bucket* do MSA é percorrido sequencialmente uma única vez e os valores de $P(L_{o_i}, r)$ são recuperados a partir da posição do sufixo no MSA. A Equação 1 é aplicada a cada posição lida do MSA, e uma soma acumulada é armazenada em um vetor de distâncias que representa $\mathsf{SFD}(o_i, q)$. No final, esse vetor de distâncias é ordenado e apenas os k-objetos mais próximos são recuperados.

3. Vetor de sufixos métrico compacto

Seja MSA_{gap} um vetor de inteiros, de tamanho $N = N_{\mathsf{refs}} \times N_{\mathsf{objs}}$, que armazena os valores iniciais de cada *bucket* no MSA e a diferença entre valores consecutivos no vetor MSA, definido da seguinte forma.

Definição 1

$$\mathsf{MSA}_{gap}[i] = \begin{cases} \mathsf{MSA}[i] & \textit{se } i \textit{ é a primeira posição de um bucket} \\ \mathsf{MSA}[i] - \mathsf{MSA}[i-1] & \textit{caso contrário} \end{cases}$$

No MSA todos os valores dentro de um *bucket* estão em ordem crescente, dessa forma, todos os valores em MSA_{gap} serão maiores do que zero. Além disso, no MSA o maior valor de inteiro é igual à N (tamanho da cadeia S). No MSA_{gap}, por sua vez, o maior valor de inteiro é $N_{\mathsf{refs}} \times 2 - 1$, uma vez que essa é a maior diferença que pode ocorrer entre $\mathsf{MSA}[i]$ e $\mathsf{MSA}[i-1]$ e a maior posição de um início de *bucket* é igual à N_{refs} (referente ao último índice da lista L_{o_1}).

Portanto, podemos utilizar um método de codificação de inteiros de tamanho variável para representar MSA_{gap} utilizando menos espaço. Por exemplo, utilizando a codificação Elias-delta [Elias 1975] o tamanho máximo será $|\delta\,(2 \times N_{\mathsf{refs}} - 1)| \times N$ bits, onde $|\delta\,(2 \times N_{\mathsf{refs}} - 1)|$ é o comprimento em bits do pior caso, isto é, o valor máximo da diferença e δ é dado pela Equação 2.

$$|\delta\,(x)| = |x| + 2\,\lfloor \log |x| \rfloor \tag{2}$$

A representação compacta do vetor MSA_{gap} pode ser construída diretamente a partir da base de dados, isto é, não é preciso computar primeiro o MSA, obter o MSA_{gap}, e depois compactá-la. Na Seção 4 mostramos o comportamento prático de duas codificações para inteiros, a Elias-delta e Simple9 [Anh and Moffat 2005].

A codificação Elias-delta produz sequências de tamanho variável para números inteiros. Esta codificação permite que inteiros menores que $16.777.215$ possam ser codificados em até 32 bits, o que é menor que $4.294.967.295$, codificados por inteiros de 4 bytes. Entretanto, no MSA_{gap} os valores serão, no máximo, $2 \times N_{\mathsf{refs}} - 1$, e ainda que o número de referências seja da ordem de centenas ou milhares, temos que a codificação, por ter tamanho variável, ficará distante de $16.777.215$, o que permite uma economia no uso da memória.

A codificação Simple9 codifica o máximo de números inteiros possíveis dentro de uma palavra de 32 bits. Para isso, essa codificação reserva os 4 bits mais significativos para indicar a quantidade de números que serão codificados dentro de uma palavra, existindo, portanto, um total de 9 possibilidades de codificação dos próximos $\lfloor 28/m \rfloor$ valores usando $m = 1, 2, 3, 4, 5, 7, 9, 14$ ou 28 bits por valor. Valores cuja representação utiliza mais do que 28 bits não podem ser codificados, mas, como mencionado, espera-se que os valores em MSA_{gap} sejam menores do que 2^{28}. Em geral, a codificação Simple9 é mais rápida para decodificar do que a Elias-delta.

Durante a etapa de consulta no vetor de sufixos métrico, os valores dentro de cada *bucket* são lidos de forma sequencial, a partir do primeiro elemento. Dessa forma, a decodificação dos valores inteiros em MSA_{gap} é feita em tempo linear ao número de bits utilizados na representação compacta.

4. Experimentos

A estrutura de dados compacta proposta, chamada de cMSA, foi implementada em duas versões. A primeira utilizando codificação Elias-delta a segunda utilizando Simple9. O desempenho dessas duas versões do cMSA foi comparado com o método original MSA. Todos os algoritmos foram implementados em `C++` e os códigos fontes podem ser acessados em `https://github.com/universal-cnpq-edc/msa`. Os experimentos foram executados em um computador com processador AMD Ryzen 9 5950X 16-Core Processor, 64 GB de RAM, SSD NVMe 1 TB, Arch Linux kernel 6.4.7-arch1-3 (64 bits).

Os conjuntos de dados utilizados foram ANN_SIFT1B[1] [Jégou et al. 2011], que contém registros com 128 valores numéricos, e o descritor *color structure* do conjunto CoPhIR[2] [Bolettieri et al. 2009], que contém de registros com 64 valores numéricos. Os testes foram realizados para diferentes tamanhos de conjuntos (1, 2, 4, 8 e 16 milhões de objetos) e 256 referências. A disposição utilizada para os gráficos visa demonstrar o caráter linear do tempo e espaço em relação ao número de objetos de entrada.

A Figura 1a apresenta os resultados obtidos para a fase de construção para o conjunto de dados ANN_SIFT1B. São apresentados os tempos de construção para os três métodos e diferentes números de objetos, sendo que método original MSA, como esperado, requer menor tempo de construção que os algorimos propostos. Os métodos cMSA Elias-delta e cMSA Simple9 têm desempenhos semelhantes e requerem em média $3{,}48$ e $3{,}35$ vezes o tempo do MSA, respectivamente.

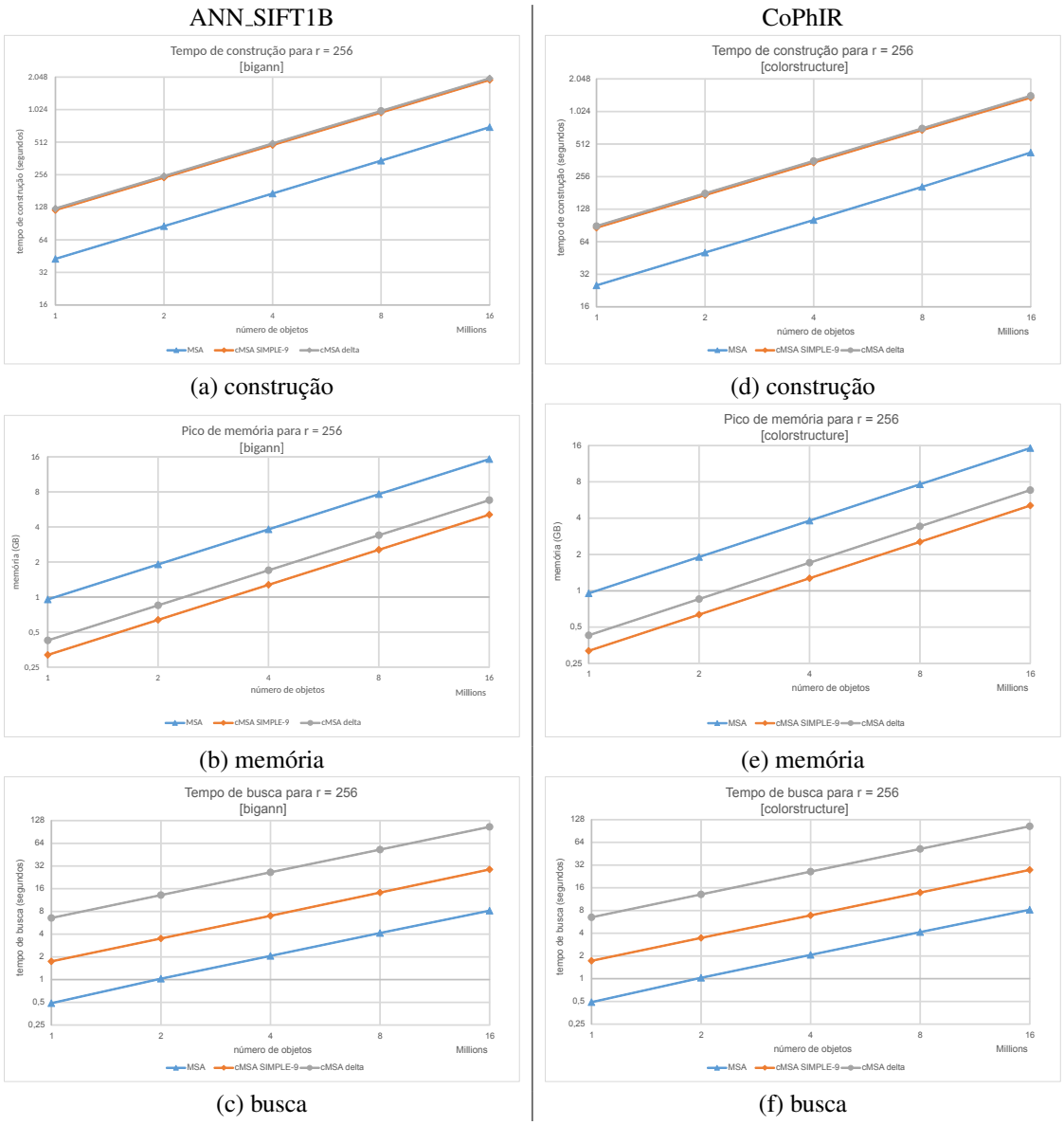
A Figura 1b apresenta os resultados com relação ao pico de memória durante a execução. Neste aspecto, os dois métodos propostos demonstram melhor desempenho do que o MSA original, sendo que, para o número de referências igual a 256, o cMSA Elias-delta requer 44% da memória requerida pelo MSA, enquanto cMSA Simple9 consegue atingir 33% da memória do MSA.

A Figura 1c mostra os tempos obtidos durante a fase de busca pelos $k = 5$ objetos mais próximos para o conjunto de dados ANN_SIFT1B. O método original do MSA utilizou menos tempo, sendo que o cMSA Elias-delta utilizou 12,8 vezes mais tempo do que o MSA, enquanto o cMSA Simple9 obteve desempenho superior, demandando $3{,}4$ vezes

[1] *Learning set* em http://corpus-texmex.irisa.fr/

[2] http://cophir.isti.cnr.it/

Figura 1. Experimentos: ANN_SIFT1B e CoPhIR.



o tempo do MSA.

Já as Figuras 1d, 1e, 1f, são referentes ao conjunto CoPhIR. A Figura 1d demonstra o comportamento semelhante a Figura1a, requerendo tempo menor devido à menor dimensionalidade do conjunto CoPhIR, o que exige menor número de computações na etapa de cálculos das distâncias euclidianas. Da mesma forma, as Figuras 1e, 1f mostram comportamento similar às curvas obtidas para o ANN_SIFT1B, o que era esperado, uma vez que após a construção das estruturas de dados, a quantidade de dimensões dos elementos de entrada não é mais relevante para o tempo de busca e espaço de memória do vetor de sufixos métrico.

Entre os aspectos que podem ser observados nos gráficos destaca-se o crescimento linear dos recursos computacionais com o aumento do número de objetos em todas as versões do vetor de sufixos métrico.

Por fim, deve-se notar que o MSA original está limitado a no máximo $2^{32} - 1$ elementos, uma vez que os índices são armazenados no vetor, e não as diferenças, como ocorre nos métodos cMSA Elias-delta e cMSA Simple9. Para conjuntos de dados maiores do que 2^{32}, é necessário aumentar do número de bits por elemento no vetor, o que, consequentemente, aumentará o pico de memória para este método. Por outro lado, o método proposto não apresenta esta limitação. De fato, testes do MSA com 16 milhões de objetos são limitados a 268 referências. Acima deste número de referências, o valor ultrapassa os 4 bytes do inteiro padrão. Por outro lado, para os métodos compactos, o único fator limitante é a memória disponível total para armazenar a estrutura de dados gerada durante o tempo de execução.

5. Conclusões

Neste artigo mostramos como utilizar estruturas de dados compactas para representar o vetor de sufixos métrico. Experimentos com dados reais mostraram que a estrutura compacta proposta reduz a 33% a memória utilizada, enquanto que o tempo de consulta aumenta no máximo 13 vezes para a codificação que utiliza Elias-delta e 3,4 vezes para a que utiliza Simple9. Como trabalhos futuros pretendemos investigar outras estratégias de codificação de inteiros e a realização das consultas em paralelo.

Referências

Anh, V. N. and Moffat, A. (2005). Inverted index compression using word-aligned binary codes. *Inf. Retr.*, 8(1):151–166. doi:10.1023/B:INRT.0000048490.99518.5c.

Bolettieri, P., Esuli, A., Falchi, F., Lucchese, C., Perego, Piccioli, and Rabitti (2009). CoPhIR: a test collection for content-based image retrieval. *CoRR*, abs/0905.4627.

Chávez, E., Figueroa, K., and Navarro, G. (2008). Effective proximity retrieval by ordering permutations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(9):1647–1658.

Elias, P. (1975). Universal codeword sets and representations of the integers. *IEEE Trans. Inf. Theory*, 21(2):194–203. doi:10.1109/TIT.1975.1055349.

Fagin, R., Kumar, R., and Sivakumar, D. (2003). Comparing top k lists. *SIAM J. Discret. Math.*, 17(1):134–160. doi:10.1137/S0895480102412856.

Jégou, H., Tavenard, R., Douze, M., and Amsaleg, L. (2011). Searching in one billion vectors: Re-rank with source coding. In *IEEE Int'l Conf. Acoustics, Speech, and Signal Processing (ICASSP)*, pages 861–864, Prague. doi:10.1109/ICASSP.2011.5946540.

Lew, M. S., Sebe, N., Djeraba, C., and Jain, R. C. (2006). Content-based multimedia information retrieval: State of the art and challenges. *ACM Trans. Multim. Comput. Commun. Appl.*, 2(1):1–19. doi:10.1145/1126004.1126005.

Manber, U. and Myers, E. W. (1993). Suffix arrays: A new method for on-line string searches. *SIAM J. Comput.*, 22(5):935–948. doi:10.1137/0222058.

Mohamed, H. and Marchand-Maillet, S. (2013). Metric suffix array for large-scale similarity search. In *ACM WSDM 2013 Workshop on Large Scale and Distributed Systems for Information Retrieval, Rome, IT*, pages 1–6.

Navarro, G. (2016). *Compact Data Structures - A Practical Approach*. Cambridge University Press.