Algoritmo para Alinhamento de Sequências em Grafos de Sequências com Economia de Espaço

Thayná A. Gimenez[1], **Lucas B. Rocha**[1], **Said Sadique Adi**[1], **Eloi Araujo**[1]

[1] Faculdade de Computação
Universidade Federal de Mato Grosso do Sul (UFMS) – Campo Grande, MS

{thayna.gimenez, said.sadique, francisco.araujo}@ufms.br

lucas.lb.rocha@gmail.com

Resumo. *Sequências biológicas obtidas de múltiplos indivíduos podem ser representadas por grafos de sequências, nos quais um alinhamento corresponde a um passeio no grafo que representa uma sequência s. Trabalhos anteriores apresentam um algoritmo com tempo $O(V + mE)$ e espaço $O(\sqrt{m} \cdot V)$, em que m é o comprimento de s e V e E correspondem aos conjuntos de vértices e arcos do grafo. Entretanto, seu alto consumo de memória pode limitar aplicações práticas. Neste trabalho, propomos uma variação que aumenta o tempo por um fator $O(\log m)$ e reduz o espaço para $O(V + E + m)$.*
Palavras-chave*: Grafo de sequências, alinhamento de sequências*

1. Introdução

Na biologia computacional, grafos são amplamente utilizados para representar segmentos genômicos [Paten et al. 2017] e descrever a diversidade genética de espécies como em [Secomandi et al. 2025]. Entre as aplicações do alinhamento de sequências em grafos destacam-se a montagem de genomas [Uliano-Silva et al. 2023], a correção de erros [Jain et al. 2020] e a identificação de variantes [Dilthey et al. 2015].

Recentemente, diversas abordagens para o problema do alinhamento de sequências em grafos foram propostas, como em [Rautiainen and Marschall 2020, Kavya et al. 2019, Ivanov et al. 2020], considerando variantes como ciclos [Navarro 2000] e edições no grafo ou na sequência [Amir et al. 2000]. Em grafos de sequência simples, cada vértice é rotulado por um símbolo, e o problema consiste em encontrar um passeio p cuja concatenação dos rótulos forme uma sequência soletrada o mais semelhante possível a uma entrada s.

[Rautiainen and Marschall 2017] propuseram um algoritmo baseado em grafos multicamadas para calcular alinhamentos ótimos em tempo $O(V + mE)$, usando espaço $O(\sqrt{m}V)$. Posteriormente, [Jain et al. 2019] estenderam o método para permitir deleção de prefixos. Contudo, essas abordagens ainda podem demandar alto consumo de memória em aplicações genômicas de larga escala [Yang et al. 2017].

Neste trabalho, descrevemos e implementamos um novo algoritmo para alinhamento entre sequências e grafos de sequências simples inspirado em [Hirschberg 1975], combinando programação dinâmica e divisão e conquista. A proposta utiliza espaço $O(V + E + m)$, dependendo linearmente do tamanho da sequência apenas para armazenar o alinhamento ótimo. A Seção 2 apresenta conceitos e definições; as Seções 3 e 4 descrevem e analisam o algoritmo; e a Seção 5 traz comentários finais. Implementações e resultados experimentais preliminares estão disponíveis em `https://github.com/thayna-gimenez/SequencetoGraphAlignment`.

2. Preliminares

Um *alfabeto* é um conjunto finito de símbolos. Uma *sequência* s sobre um alfabeto Σ é uma justaposição $s[1]s[2]\dots s[m]$ de símbolos de Σ, onde $m = |s|$ é o *comprimento* de s. O reverso de s é $s^R = s[m]s[m-1]\dots s[1]$. A concatenação de sequências s e t é denotada por st.

Um *grafo simples de sequências sobre* Σ, ou simplesmente *grafo*, é um digrafo $G = (V, E, \sigma)$ em que V e E são os conjuntos de vértices e arcos, respectivamente, e $\sigma : V \to \Sigma$ atribui a cada vértice de V um rótulo em Σ. Um *passeio* p em G é uma sequência $v_0, v_1, \dots, v_{|p|}$ de vértices de V tal que $v_{i-1}v_i \in E$ para $0 < i \leq |p|$. O inteiro $|p|$ é o *comprimento* de p. Um *caminho* é um passeio sem repetição de vértices. A sequência $\sigma(p) = \sigma(v_0)\sigma(v_1)\dots\sigma(v_{|p|})$ é dita *soletrada* (ou *induzida*) por p.

Um *alinhamento* A *de uma sequência* s *em um passeio* p é utilizado para representar um conjunto de operações de edição em s, e é obtido inserindo um símbolo '$-$' $\notin \Sigma$, chamado *espaço*, em algumas posições de s e de $\sigma(p)$, de modo que as sequências s' e $\sigma'(p)$ resultantes tenham o mesmo comprimento e não exista posição j tal que $s'[j] =$ '$-$' $= \sigma'(p)[j]$. O *tamanho* de A é $|s'| = |\sigma'(p)|$, isto é, o comprimento de s ou de $\sigma(p)$ após a inserção dos símbolos iguais a '$-$'. O par $(s'[j], \sigma'(p)[j])$ é a *coluna* j de A. Uma coluna j tal que $s'[j] =$ '$-$' representa a *inserção* de $\sigma'(p)[j]$ em s; $\sigma'(p)[j] =$ '$-$' representa a *remoção* de $s'[j]$; e $s'[j] \neq \sigma'(p)[j]$ representa a *substituição* de $s'[j]$ por $\sigma'(p)[j]$. O número total de inserções, remoções e substituições é o *custo de edição* de A. A *distância de edição* entre uma sequência s e um grafo G, com vértices inicial e/ou final possivelmente especificados, é o menor custo de edição de um alinhamento entre s e um passeio de G que respeite tais restrições, quando aplicáveis. Um *alinhamento ótimo* de s em G é um alinhamento cujo custo de edição é igual à distância de edição entre s e G.

Problema de Alinhamento de Sequências em Grafos de Sequências, ou **Problema de Mapeamento de Sequências em Grafos de Sequências** – `GSMP`: dados uma sequência s e um grafo G, encontrar um alinhamento ótimo de s em G.

[Rautiainen and Marschall 2017] e, posteriormente, [Jain et al. 2019] descreveram uma estrutura para a solução do problema `GSMP`, denominada grafo de alinhamento. Trata-se de um grafo de sequências composto por $m = |s|$ camadas, cada uma contendo uma cópia de G. O grafo é construído de modo que cada alinhamento válido de s em G corresponda a um caminho que parte de um vértice especial `fon` e termina em outro vértice especial `dst`. Formalmente, para uma sequência s e um grafo de sequências $G(V, E, \sigma)$, o *grafo de alinhamento* é um digrafo $G_a(V_a, E_a, \omega_a)$, onde

$$V_a = (\{1, \dots, m\} \times (V \cup \{\delta\})) \cup \{\texttt{fon}, \texttt{dst}\},$$

e $\omega_a : E_a \to \mathbb{R}_{\geq 0}$ é uma função de pesos em $\{0, 1\}$ definida pela operação de edição representada pela aresta (inserção, remoção ou substituição). Os arcos de E_a são exatamente aqueles para os quais ω_a está definida. Uma função $c(j, v)$ devolve o custo de um alinhamento ótimo entre o prefixo $s[1 : j]$ e um passeio de G que termina em v, isto é, o custo de um caminho ótimo em G_a de `fon` até (j, v).

O grafo G_a^R é obtido a partir de G_a pela reversão de seus arcos. Um caminho de custo mínimo de `fon` até `dst` em G_a corresponde a um alinhamento de s em G de mesmo custo. Além disso, o custo de um alinhamento de s em G_a é igual ao de um alinhamento de $s^{\mathbf{R}}$ em G_a^R.

3. Algoritmo

Nesta seção, descrevemos resumidamente o algoritmo. O pseudocódigo, bem como a implementação e detalhes adicionais, encontram-se disponíveis no GitHub. O algoritmo de [Jain et al. 2019] busca um caminho mínimo no grafo multicamadas correspondente a um passeio ótimo em G entre os vértices `sr` e `tg`, usando os vértices especiais `fon` e `dst` para permitir a deleção de prefixos (Figura 1).

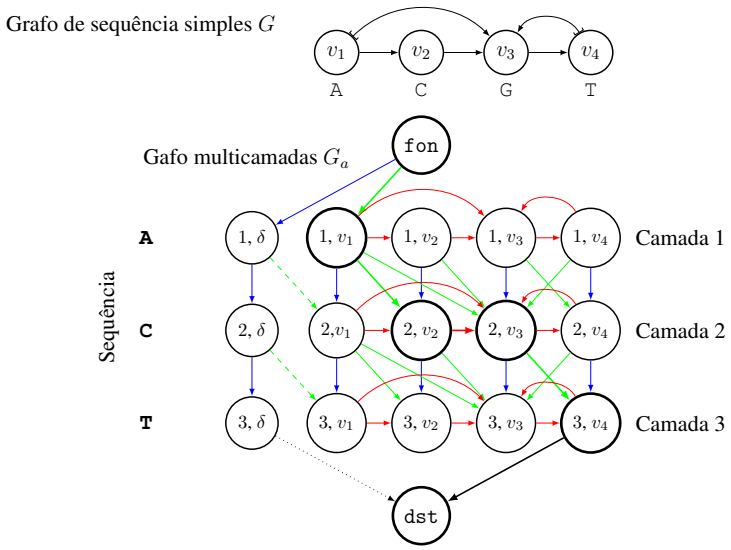

Figura 1. Instância do `GSMPst` com a sequência ACT e o grafo G acima, com $sr = v_1$ e $tg = v_4$. Os arcos na horizontal, vertical e diagonal representam os casos de inserção, deleção e substituição, respectivamente. Esse grafo multicamadas é uma adaptação do grafo usado em [Jain et al. 2019]

.

Uma versão com economia de espaço pode ser obtida mantendo apenas uma quantidade constante de camadas na memória. Entretanto, essa abordagem não preserva informação suficiente para reconstruir o alinhamento ótimo. Como em [Hirschberg 1975], utilizamos divisão e conquista sobre a sequência $s = x \cdot t \cdot y$, onde $x = s[1 : \lfloor m/2 \rfloor - 1]$, $t = s[\lfloor m/2 \rfloor]$ e $y = s^R[\lfloor m/2 \rfloor + 1 : m]$.

Com x, calculamos $M_1(u) = c(\lfloor m/2 \rfloor - 1, u)$ e $M_2(v) = c(\lfloor m/2 \rfloor, v)$, enquanto y é utilizada para calcular $M^R(w) = c(m - \lceil m/2 \rceil, w)$ no grafo reverso. Para cada tripla u, v, w tal que $uv, vw \in E$, calculamos o mínimo entre

$$\begin{aligned} M_1(u) + \sigma(v) + M^R(w), &\quad \text{(caso 1: substituição)} \\ M_1(v) + 1 + M^R(w), &\quad \text{(caso 2: deleção)} \\ M_2(v) + 1 + M^R(w). &\quad \text{(caso 3: inserção)} \end{aligned}$$

Dependendo do caso escolhido, o algoritmo realiza duas chamadas recursivas para reconstruir o alinhamento ótimo. Por exemplo, quando t é alinhado ao vértice v, fazemos uma chamada para x, iniciando em `sr` e terminando em u, e outra para y, iniciando em w e terminando em `tg`.

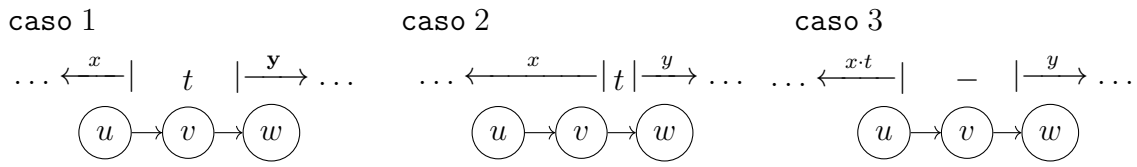

Figura 2. Caso 1 corresponde a substituição; caso 2 corresponde a deleção; caso 3 corresponde a inserção.

4. Análise de complexidade

Seja $J(m)$ o número de camadas processadas pelo algoritmo de Jain, onde $m = |s|$ denota o comprimento da sequência s, a qual será alinhada com o grafo G. O nosso algoritmo processa $T(m)$ camadas. Inicialmente, são processadas $m/2$ camadas para determinar M_1 e M_2, seguidas de mais $n/2$ camadas para determinar M^R. Em seguida, o símbolo $s_{\lfloor m/2 \rfloor}$ é alinhado a um vértice de G, e o problema é decomposto em duas chamadas recursivas independentes, cada uma responsável por processar no máximo $T(\lceil m/2 \rceil)$ camadas. Assim, temos assintoticamente a recorrência

$$T(m) = \frac{m}{2} + \frac{m}{2} + 2T(m/2) = 2T(m/2) + m.$$

Pela aplicação do Teorema Mestre, segue que $T(m) = \Theta(m \log m)$. Portanto, o overhead de tempo do nosso algoritmo em relação ao algoritmo de [Jain et al. 2019] é dado por $T(m)/J(m) = O(\log m)$, isto é, nosso algoritmo tem um fator $\log m$ mais lento que o algoritmo de [Jain et al. 2019]. Sendo assim, o tempo gasto é $O(m(V + E) \log m)$.

Durante a execução do algoritmo, mantemos apenas uma quantidade de camadas que é limitada por uma constante. Logo, o espaço necessário total também é limitado pelo espaço utilizado por uma camada mais uma estrutura para armazenar o próprio alinhamento. Ou seja, o espaço utilizado é $O(V + E + m)$.

5. Conclusões Finais e Discussão

A contribuição teórica do algoritmo proposto reside na complexidade de espaço $O(V + E + m)$, que contrasta com a abordagem de [Rautiainen and Marschall 2017] de complexidade $O(\sqrt{n} \cdot V)$, potencialmente proibitiva para sequências longas.

O algoritmo foi implementado e submetido a uma fase inicial de testes, na qual foi comparado a uma versão do algoritmo de [Jain et al. 2019] que retorna a distância de edição entre uma sequência s e um grafo G e o alinhamento ótimo de s em G. Os resultados evidenciaram uma diferença significativa no consumo de espaço entre as duas abordagens: para sequências menores, o algoritmo de [Jain et al. 2019] apresentou menor uso de memória; entretanto, para sequências maiores, nossa abordagem mostrou redução considerável nesse custo, ainda que com maior tempo de execução.

Os resultados obtidos até o momento são apresentados no GitHub. Mesmo sendo promissores, ainda são preliminares e indicam a necessidade de refinamentos, tanto na implementação do algoritmo quanto no monitoramento dos testes. Espera-se que, em cenários envolvendo sequências mais longas, a redução no custo de memória possa representar uma vantagem relevante em relação às abordagens existentes.

Referências

Amir, A., Lewenstein, M., and Lewenstein, N. (2000). Pattern matching in hypertext. *Journal of Algorithms*, 35(1):82–99.

Dilthey, A., Cox, C., Iqbal, Z., Nelson, M. R., and McVean, G. (2015). Improved genome inference in the mhc using a population reference graph. *Nature Genetics*, 47:682–688.

Hirschberg, D. S. (1975). A linear space algorithm for computing maximal common subsequences. *Communications of the ACM*, 18(6):341–343.

Ivanov, P., Bichsel, B., Mustafa, H., Kahles, A., Rätsch, G., and Vechev, M. (2020). Astarix: Fast and optimal sequence-to-graph alignment. In *Research in Computational Molecular Biology*, pages 104–119, Cham. Springer International Publishing.

Jain, C., Zhang, H., and Aluru, S. (2020). A comprehensive evaluation of long read error correction methods. *BMC Genomics*, 21:889.

Jain, C., Zhang, H., Gao, Y., and Aluru, S. (2019). On the complexity of sequence to graph alignment. In Cowen, L. J., editor, *Research in Computational Molecular Biology*, pages 85–100, Cham. Springer International Publishing.

Kavya, V. N. S., Tayal, K., Srinivasan, R., and Sivadasan, N. (2019). Sequence alignment on directed graphs. *Journal of Computational Biology*, 26(1):53–67.

Navarro, G. (2000). Improved approximate pattern matching on hypertext. *Theoretical Computer Science*, 237(1):455–463.

Paten, B., Novak, A. M., Eizenga, J. M., and Garrison, E. (2017). Genome graphs and the evolution of genome inference. *Genome Research*, 27(5):665–676.

Rautiainen, M. and Marschall, T. (2017). Aligning sequences to general graphs in $O(V + mE)$ time. *bioRxiv*, page 216127.

Rautiainen, M. and Marschall, T. (2020). Graphaligner: rapid and versatile sequence-to-graph alignment. *Genome Biology*, 21:253.

Secomandi, S., Guido Roberto Gallo, R. R., Fernandes, C. R., Jarvis, E. D., Gianfranceschi, A. B.-A. L., and Formenti, G. (2025). Pangenome graphs and their applications in biodiversity genomics. *Nature Genetics*, 57:13–26.

Uliano-Silva, M., Gabriel R. N. Ferreira, J., Krasheninnikova, K., of Life Consortium, D. T., Formenti, G., Abueg, L., Torrance, J., Myers, E. W., Durbin, R., Blaxter, M., and McCarthy, S. A. (2023). Mitohifi: a python pipeline for mitochondrial genome assembly from pacbio high fidelity reads. *BMC Bioinformatics*, 24:288.

Yang, A., Troup, M., and Ho, J. W. (2017). Scalability and validation of big data bioinformatics software. *Computational and Structural Biotechnology Journal*, 15:379–386.