UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

DEPARTAMENTO ACADÊMICO DE INFORMÁTICA

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

MATHEUS COSTACURTA

ALINHAMENTOS GLOBAIS DE DUAS SEQUÊNCIAS GENÉTICAS: RESULTADOS ÓTIMOS EM ESPAÇO LINEAR

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA

2019

MATHEUS COSTACURTA

ALINHAMENTOS GLOBAIS DE DUAS SEQUÊNCIAS GENÉTICAS: RESULTADOS ÓTIMOS EM ESPAÇO LINEAR

Trabalho de Conclusão de Curso apresentado como requisito parcial à aprovação na disciplina Trabalho de Conclusão de Curso 2, do curso de Bacharelado em Ciência da Computação, do Departamento Acadêmico de Informática, da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Leandro Miranda Zatesko (UTFPR-CT)
Coorientadora: Prof[a] Dr[a] Sheila Morais de Almeida (UTFPR-PG)

PONTA GROSSA

2019

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Ponta Grossa
Diretoria de Graduação e Educação Profissional
Departamento Acadêmico de Informática
Bacharelado em Ciência da Computação

TERMO DE APROVAÇÃO

Alinhamentos globais de duas sequências genéticas: resultados ótimos em espaço linear

por

Matheus Costacurta

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em vinte e cinco (25) de novembro de 2019 como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Prof. Dr. Leandro Miranda Zatesko (UTFPR-CT)
Orientador(a)

_____
Prof. Dr. Renato José da Silva Carmo (UFPR)
Membro

_____
Prof(a). MSc. Denise do Rocio Maciel
Membro

_____
Prof. Geraldo Ranthum
Responsável pelo Trabalho de Conclusão de Curso

_____
Prof(a). Mauren Louise Sguario
Coordenador(a) do curso

RESUMO

COSTACURTA, M. *Alinhamentos globais de duas sequências genéticas: resultados ótimos em espaço linear*. 2019. 51 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) — Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2019.

A Bioinformática é a aplicação de técnicas computacionais, matemáticas e estatísticas para analisar, interpretar e processar dados biológicos, principalmente dados relacionados a genética. Uma tarefa frequente nessa área é o alinhamento de sequências, que utiliza uma função de custo para apresentar o alinhamento ótimo entre sequências relacionadas. Esta tarefa é frequente por ser utilizada para comparação de material genético e construção da taxonomia de seres vivos. A complexidade de espaço dos algoritmos exatos conhecidos que apresentam mais que um alinhamento ótimo entre duas sequências genéticas é $O(mn)$, sendo m e n os tamanhos das duas sequências em pares de bases. Essa complexidade de espaço pode ser indesejada, visto que o tamanho das sequências pode chegar a milhões de bases. Por isso, muitos autores utilizam heurísticas, mais eficientes no uso do espaço, porém sem a garantia de uma solução ótima. O presente trabalho, contudo, tem como objetivo desenvolver um algoritmo exato com complexidade de espaço linear para encontrar mais que um alinhamento ótimo entre duas sequências genéticas.

Palavras-chave: Bioinformática. Alinhamento de sequências. Strings. Complexidade de algoritmos. Otimização.

ABSTRACT

COSTACURTA, M. *Global alignments of two genetic sequences: optimal results in linear space*. 2019. 51 f. Work of Conclusion Course (Graduation in Computer Science) — Federal University of Technology - Paraná. Ponta Grossa, 2019.

Bioinformatics is an application of computational, statistical and mathematical techniques for the analysis, interpretation and control of biological data, mainly related to genetics. A frequent tasks in this area is the sequence alignment, which uses a cost function to present an optimum alignment between related sequences. This task is frequent since it is used to compare genetic material and to construct the taxonomy of living beings. The space complexity of the known exact algorithms to present more than one optimum alignment between two genetic sequences is $O(mn)$, where m and n are the sizes of the two sequences given by the number of base pairs. This space complexity may be unwanted, since the size of the sequences can reach millions of bases. Hence, many authors use heuristics, more efficient in the use of space, but without the certainty of optimality. This work, however, aims to develop a linear space complexity exact algorithm to find more than one optimum alignment between two genetic sequences.

Keywords: Bioinformatics. Sequence Alignment. Strings. Complexity of Algorithms. Optimization.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

Desde que a estrutura de DNA foi descoberta em 1953, a biologia molecular teve grandes avanços. E com o fim do sequenciamento do genoma humano em 2003, uma enorme quantidade de dados vem sendo gerada. A necessidade de processar essas informações criou vários desafios interdisciplinares, em particular, nas áreas voltadas para Ciência da Computação e Matemática, o que fez surgir um novo campo de estudo. A Bioinformática tem como objetivo automatizar a obtenção, distribuição e análise de dados genéticos (BALDI; BRUNAK, 2001). Sua consolidação como potencial campo de estudo científico decorreu da melhoria do desempenho dos processadores.

Estudos de Bioinformática estão voltados para organizar e analisar grandes quantidades de dados, buscando tratar problemas de identificação e sequenciamento de genes, possibilitando que esses dados se tornem informações significativas do ponto de vista biológico. Esses dados são sequências de nucleotídeos ou de aminoácidos armazenadas em bases de dados privadas ou públicas.

Tais estudos são de interesse em várias áreas. Pode-se, por exemplo, compreender melhor a origem de doenças e identificar os genes que as causam, o que permite o desenvolvimento de novas drogas e um tratamento médico mais eficaz. Outra aplicação é na adaptação de organismos geneticamente modificados para a agricultura.

Segundo Setubal e Meidanis (1997), o alinhamento de sequências consiste em alinhar sequências relacionadas, otimizando uma função de custo[1]. O alinhamento genético de sequências é utilizado como base em outros procedimentos na área da Bioinformática e por isso se torna um problema computacional fundamental para a área. É com o alinhamento de sequências que se pode identificar o quão similares as cadeias são, sejam elas cadeias de DNA, RNA ou de proteínas.

O problema de comparar sequências genéticas pode ser visto como um caso particular do problema de comparar *strings* na Ciência da Computação. Então, métodos e algoritmos conhecidos para tratamento de *strings* podem ser reutilizados e adaptados para a Bioinformática.

Um alinhamento de sequências pode ser global ou local. No alinhamento global, deve-se obter o valor ótimo da função de custo considerando-se todo o comprimento das sequências analisadas. No alinhamento local, buscam-se ótimos locais da função de custo, que implicam em encontrar trechos de alta similaridade entre as sequências analisadas, não importando se existe ou não similaridade entre as sequências como um todo.

Considerando os métodos exatos para alinhamento global de duas sequências genéticas, o algoritmo de Needleman e Wunsch (1970) é baseado em programação dinâmica e capaz de computar todos os alinhamentos ótimos globais possíveis em tempo e espaço $O(mn)$, sendo m e n o tamanho das sequências. Observe-se que esta complexidade de tempo dificilmente pode

[1] A função de custo será apresentada em mais detalhes na Seção 2.3.

ser melhorada, pois o problema de alinhamento de sequências não pode ser resolvido em tempo $O(\max\{m,n\}^{2-\epsilon})$ para $\epsilon > 0$ algum, a menos que valha a Hipótese de Tempo Exponencial Forte, uma asserção da Complexidade Computacional que se conjectura não ser verdade (ABBOUD AMIR E WILLIAMS, 2014).

No problema em que queremos mais de um alinhamento global ótimo, o custo de tempo para gerar cada alinhamento ótimo adicional é $O(m+n)$. Portanto, para gerar q alinhamentos ótimos o algoritmo de Needleman e Wunsch (1970) gasta tempo $O(mn + q(m+n))$.

O algoritmo proposto por Hirschberg (1975), baseado em divisão e conquista e também em programação dinâmica possui a mesma complexidade de tempo do algoritmo de Needleman e Wunsch (1970) e complexidade de espaço $O(m+n)$, isto é, linear no tamanho da entrada. Entretanto, este algoritmo garante apenas um alinhamento ótimo entre duas sequências analisadas.

Observa-se então que, no cenário de apresentar vários alinhamentos ótimos globais, o espaço é um gargalo computacional deste problema. Neste trabalho apresentamos um algoritmo exato com complexidade de espaço linear e tempo polinomial em relação ao tamanho total da saída e da entrada, mas capaz de apresentar vários alinhamentos globais ótimos, caso existam.

1.1 OBJETIVOS

Segundo Myers e Miller (1988), quando se está pesquisando por arranjos "biologicamente significativos" é comum considerar vários alinhamentos. Desta forma, este trabalho apresenta uma adaptação de Hirschberg com complexidade de espaço linear que seja capaz de devolver mais que um alinhamento global ótimo entre duas sequências genéticas, caso exista.

1.1.1 Objetivos Específicos

- Adaptar o algoritmo de Hirschberg para produzir mais que um alinhamento global ótimo se existir, mantendo as complexidades de espaço linear (na entrada) e de tempo polinomial (no total da entrada e da saída);
- Apresentar uma análise da complexidade de tempo e espaço da adaptação.

1.2 ORGANIZAÇÃO DO TRABALHO

O restante deste documento é dividido da seguinte maneira: no Capítulo 2 são apresentados conceitos básicos da biologia molecular, uma introdução ao alinhamento de sequências genéticas e um conjunto de definições e algoritmos utilizados no campo da bioinformática que

são importantes para o desenvolvimento deste trabalho. No Capítulo 3 são mostrados os resultados obtidos neste projeto com base nas técnicas estudadas. Por fim, o Capítulo 4 conclui este trabalho com observações finais e sugestões para trabalhos futuros.

2 ARCABOUÇO TEÓRICO

Este capítulo apresenta definições e resultados da literatura que são importantes para o desenvolvimento deste trabalho. São englobados aqui conceitos básicos da biologia molecular, uma introdução ao alinhamento de sequências genéticas e uma amostra de definições e algoritmos utilizados no campo da Bioinformática.

2.1 CONCEITOS BÁSICOS DA BIOLOGIA MOLECULAR

2.1.1 Células

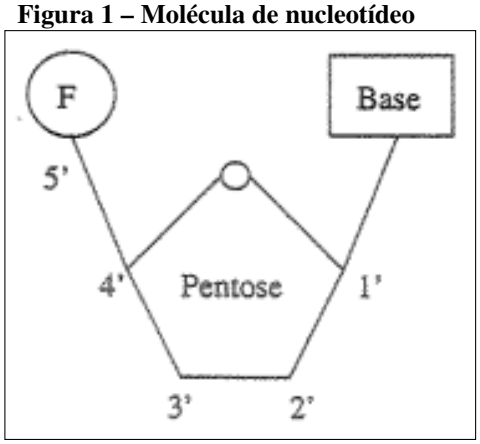
As células são formadoras de organismos. Nelas realizam-se processos metabólicos dos seres vivos para a fabricação (síntese) de moléculas simples, tais como açúcares e aminoácidos, ou macromoléculas, tais como ácidos nucleicos e proteínas (OKURA *et al.*, 2002). As células contêm informações genéticas e são capazes de transmitir essas informações no momento da divisão celular (OKURA *et al.*, 2002).

2.1.2 Ácidos nucleicos

As informações sobre a divisão e o desenvolvimento das células são armazenadas nas moléculas de ácidos nucleicos, que são compostas por várias unidades de nucleotídeos (OKURA *et al.*, 2002). Na natureza há dois tipos de ácidos nucleicos: DNA e RNA.

O nucleotídeo é um composto químico constituído por três partes: uma molécula de fosfato, uma pentose (açúcar) e uma base nitrogenada, sendo ela adenina (A), timina (T), uracila (U), citosina (C) ou guanina (G) (OKURA *et al.*, 2002). A Figura 1 apresenta uma molécula de nucleotídeo.

Figura 1 – Molécula de nucleotídeo

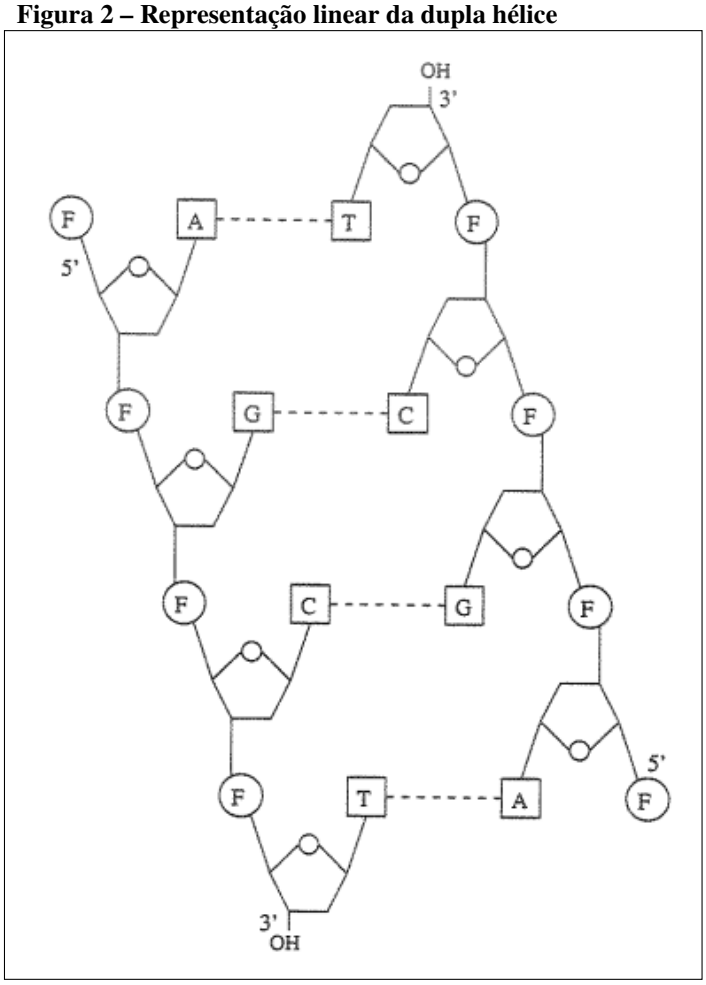


Fonte: Okura *et al.* (2002)

As moléculas de DNA (ácido desoxirribonucleico) e RNA (ácido ribonucleico) são formadas por um conjunto de nucleotídeos. As moléculas de RNA possuem fitas simples e são curtas comparadas às moléculas de DNA (OKURA *et al.*, 2002). As moléculas de DNA compõem-se de duas fitas, conhecidas como dupla hélice. As duas fitas são ligadas por pontes de hidrogênio. Na sua base nitrogenada, são encontradas as bases adenina (A), timina (T), citosina (C) e guanina (G), enquanto na molécula de RNA ocorre a substituição da base T pela base uracila (U).

A base A é sempre pareada com a base T, enquanto a base C é pareada com a base G. Esses pareamentos são a medida de comprimento da molécula, denotado por bp (*base pair*), ou seja, é o número de pares de bases que a formam. Por exemplo, uma molécula de DNA que possui uma fita de 800 bases nitrogenadas tem comprimento de 800bp (OKURA *et al.*, 2002). A Figura 2 apresenta uma dupla hélice de uma molécula de DNA com tamanho 4bp.

Figura 2 – Representação linear da dupla hélice



Fonte: Okura *et al.* (2002)

Um cromossomo humano possui cerca de 10^8 pares de bases (SETUBAL; MEIDANIS, 1997). Conforme dados retirados do NCBI, na Tabela 1 estão listados os cromossomos humanos juntamente com o tamanho das moléculas (bp).

Tabela 1 – Lista de cromossomos humanos

Cromossomo	Tamanho (Mbp)
1	249,0
2	242,0
3	198,0
4	190,0
5	182,0
6	171,0
7	159,0
8	145,0
9	138,0
10	134,0
11	135,0
12	133,0
13	114,0
14	107,0
15	102,0
16	90,0
17	83,0
18	80,0
19	59,0
20	64,0
21	47,0
22	51,0
X	156,0
Y	57,0

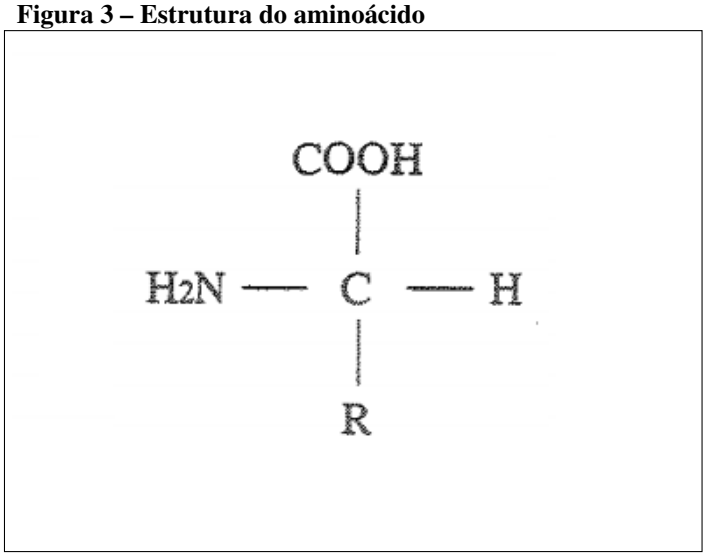
Fonte: Autoria própria

2.1.3 Proteínas

As proteínas são formadas por unidades de aminoácidos e possuem várias funções dentro de um organismo, como na estruturação das células e tecidos do corpo humano, no transporte de substâncias e na defesa de um organismo (OKURA *et al.*, 2002). As moléculas de proteínas não são apenas uma sequência linear de aminoácidos.

Os aminoácidos são moléculas orgânicas que possuem um carbono central (C), um hidrogênio (H), um grupo amino (H_2N), um grupo carboxil (COOH) e uma cadeia lateral (R) (SETUBAL; MEIDANIS, 1997). A Figura 3 apresenta a estrutura de um aminoácido.

Figura 3 – Estrutura do aminoácido



Fonte: Okura *et al.* (2002)

Na natureza são encontrados 20 diferentes aminoácidos (Tabela 2) e a cadeia lateral é capaz de distinguir cada um deles (SETUBAL; MEIDANIS, 1997). Essa quantidade propicia uma grande e complexa variedade de proteínas (OKURA *et al.*, 2002).

Tabela 2 – Os vinte aminoácidos

Código uma-letra	Código três-letras	Nome
A	Ala	Alanina
C	Cys	Cisteina
D	Asp	Ácido aspártico
E	Glu	Ácido glutâmico
F	Phe	Fenilalanina
G	Gly	Glicina
H	His	Histidina
I	Ile	Isoleucina
K	Lys	Lisina
L	Leu	Leucina
M	Met	Metionina
N	Asn	Asparagina
P	Pro	Prolina
Q	Gln	Glutamina
R	Arg	Arginina
S	Ser	Serina
T	Thr	Treonima
V	Val	Valina
W	Trp	Tripofano
Y	Tvr	Tirosina

Fonte: Autoria própria

O tamanho de uma proteína é o número de aminoácidos que a compõem. Trezentos aminoácidos é o tamanho médio apresentado pelas proteínas, embora seu tamanho possa variar entre 100 a 5000 aminoácidos (SETUBAL; MEIDANIS, 1997).

2.2 BASE DE DADOS

Devido ao número de dados gerados em laboratórios no mundo, a construção de bancos de dados tornou-se indispensável para a organização acessível e não redundante dos dados.

O Centro Nacional de Informação Biotecnológica, nos Estados Unidos (NCBI) é o banco de dados central sobre informações genéticas e possui todas as sequências de DNA, RNA e proteínas disponíveis publicamente. Existem aproximadamente 26 bilhões de pares de bases (pb) em sua base de dados (COHEN, 2004). O maior gene conhecido no servidor possui cerca de 20 milhões de pares de bases e a maior proteína cerca de 34.000 aminoácidos (COHEN, 2004).

O grande volume de dados concentrados no NCBI criou a necessidade de programas eficientes para serem utilizados na consulta das bases de dados. Em uma aplicação típica na busca de semelhanças, é comparada uma sequência com todas aquelas armazenadas no banco de dados. Isso significa uma grande quantidade de comparações de sequências (SETUBAL; MEIDANIS, 1997).

A complexidade dos métodos exatos que serão apresentados na Seção 2.5 a fim de computar alinhamentos ótimos entre pares de sequências os tornam inviáveis para grandes pesquisas na bases de dados. Para acelerar a pesquisa, métodos novos e mais rápidos foram desenvolvidos. Em geral, esses métodos são baseados em heurísticas (SETUBAL; MEIDANIS, 1997).

Um dos programas mais populares para pesquisas em banco de dados de sequências genéticas é o BLAST (do inglês, *Basic Local Alignment Search Tool*) (SETUBAL; MEIDANIS, 1997). Esta ferramenta trabalha com heurística e é capaz de comparar uma sequência de DNA com todas as sequências genômicas de domínio público (ALTSCHUL *et al.*, 1997).

2.3 DEFINIÇÕES DA BIOINFORMÁTICA

As definições apresentadas a seguir foram retiradas da fonte (SANDES, 2015) e serão utilizadas no restante deste documento.

Definição 2.1 (alfabeto)**.** *Um alfabeto Σ é um conjunto finito de símbolos, também chamados de caracteres. Em particular, $\Sigma = \{A, T, G, C\}$ é o alfabeto que usamos quando estamos tratando de sequências genéticas em segmentos de DNA.*

Definição 2.2 (sequência)**.** *Sequência é uma série de caracteres ou símbolos extraídos de um alfabeto Σ. O termo sequência é sinônimo de string. Uma sequência pode ter caracteres repetidos, por exemplo, $S = (A, T, C, \cdots, G, T, A, A)$.*

Definição 2.3 (elemento de sequência)**.** *O símbolo ocupando a posição i de uma sequência S é denotado por $S[i]$. No exemplo anterior, $S[1] = A$ e $S[2] = T$.*

Definição 2.4 (comprimento da sequência)**.** *O comprimento de uma sequência S é definido pelo número de símbolos que a compõem e representado por $|S|$.*

Definição 2.5 (concatenação de duas sequências)**.** *A concatenação de duas sequências S_1 e S_2 é denotada por S_1S_2 e é formada por todos os elementos de S_2 anexados após os elementos de S_1. Por exemplo: se $S_1 = (A, T, C)$ e $S_2 = (A, G, G)$, temos $S_1S_2 = (A, T, C, A, G, G)$. O comprimento de S_1S_2 é igual a $|S_1| + |S_2|$.*

Definição 2.6 (reverso)**.** *O reverso de uma sequência $S = (S[1], \ldots, S[|S|])$ é a sequência $(S[|S|], \ldots, S[1])$, denotado por $rev(S)$.*

Definição 2.7 (alinhamento)**.** *Um* alinhamento *de duas sequências S_1 e S_2 é um par de sequências $A = (S'_1, S'_2)$ de tal modo que:*

- *para $i \in \{1, 2\}$, a sequência S'_i pode ser obtida a partir da sequência S_i com a inserção de espaços (gaps);*
- *S'_1 e S'_2 possuem o mesmo comprimento, que definimos como o* comprimento do alinhamento *A, denotado por $|A|$.*

A Figura 4 apresenta um exemplo de alinhamento entre as sequências S_1 e S_2 dadas por $S_1 = (A, T, G, C, G, C)$ e $S_2 = (T, A, A, T, G, C)$.

Figura 4 – Alinhamento de S_1 e S_2

A =	-	A	T	G	C	G	C
	T	A	A	T	-	G	C

Fonte: Autoria própria

Definição 2.8 (*gap* linear)**.** *A ocorrência de gaps no alinhamento está associada a uma pontuação numérica negativa. O valor negativo para cada gap inserido no alinhamento é uma constante negativa $-G$. Ao introduzir espaços (gaps) nas sequências, um algoritmo de alinhamento pode combinar mais caracteres e consequentemente encontrar alinhamentos com maior pontuação.*

Definição 2.9 (matriz de substituição)**.** *Na implementação do algoritmo, define-se um valor para match e outro para mismatch ou também pode-se utilizar uma matriz de substituição para atribuir esses valores. A matriz de substituição sbt define a pontuação para cada par de elementos $a, b \in \Sigma$. Quanto maior a pontuação $sbt(a, b)$ mais próximo evolutivamente eles são. No caso dos nucleotídeos, utiliza-se uma pontuação positiva para os matches (pareamento de bases iguais) e uma pontuação negativa para as mismatches (pareamento de bases diferentes), conforme exemplificado nas Tabelas 3 e 4.*

Tabela 3 – Matriz de substituição para DNA com valor de *match* igual a 1

	A	C	G	T
A	1	-1	-1	-1
C	-1	1	-1	-1
G	-1	-1	1	-1
T	-1	-1	-1	1

Fonte: Autoria própria

Tabela 4 – Matriz de substituição para DNA com valor de *match* igual a 2

	A	C	G	T
A	2	-1	-1	-1
C	-1	2	-1	-1
G	-1	-1	2	-1
T	-1	-1	-1	2

Fonte: Autoria própria

No caso dos aminoácidos, pode-se utilizar as matrizes PAM (DAYHOFF; SCHWARTZ; ORCUTT, 1978) e BLOSUM (HENIKOFF; HENIKOFF, 1992) para distinguir entre as diferentes combinações de aminoácidos possíveis.

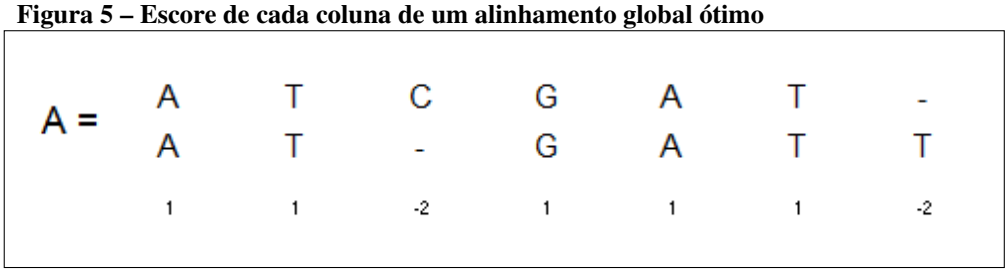
Vale ressaltar que o resultado de um alinhamento de sequências pode ser diferente conforme a matriz de substituição utilizada.

Definição 2.10 (escore de uma coluna)**.** *Sejam a_i e b_i dois caracteres pareados na posição i de um alinhamento de duas sequências. O escore e_i é definido por:*

$$e_i = \begin{cases} gap_i, & \textit{se} \quad a_i = - \textit{ ou } b_i = - \\ sbt(a_i, b_i), & \textit{caso contrário.} \end{cases} \tag{2.1}$$

A Figura 5 apresenta o escore de cada coluna de um alinhamento para as sequências (A, T, C, G, A, T) e (A, T, G, A, T, T) com a matriz de substituição da Tabela 3 e um valor de gap igual a -2.

Figura 5 – Escore de cada coluna de um alinhamento global ótimo



Fonte: Autoria própria

Em todo problema de otimização visamos encontrar soluções que maximizem uma *função objetivo*, ou *função de custo*. No problema de alinhamento de sequências, as soluções que buscamos são os alinhamentos, e a função de custo que queremos maximizar é o *escore do alinhamento*, conforme definimos a seguir.

Definição 2.11 (escore de um alinhamento)**.** *A soma dos escores de todas as colunas resulta no escore do alinhamento A, isto é, $score(A) = \sum_{i=0}^{n} e_i$, sendo n o comprimento do alinhamento.*

Considerando o valor de *match* igual a 1, de *mismatch* igual a -1, e um *gap* linear igual a -2, a Figura 5 apresenta o escore de um alinhamento A igual a $+1$. Segundo Setubal e Meidanis (1997), este sistema de pontuação é muitas vezes usado na prática.

Com os valores mencionados anteriormente a melhor pontuação ocorre quando duas sequências idênticas de comprimento n estão alinhadas, portanto a pontuação resultante é n. Quando se alinha uma sequência com outra vazia, a pontuação é de $-2n$, a menor pontuação possível.

Definição 2.12 (escore ótimo e alinhamento ótimo)**.** *Visto que dadas duas sequências pode-se ter diversos alinhamentos distintos, o alinhamento que tiver o maior escore entre todos é considerado o alinhamento ótimo e esse é o escore ótimo. Denotaremos um alinhamento ótimo de A^{opt} e o escore ótimo de $score^{opt}$. O escore ótimo é único, mas pode haver vários alinhamentos globais ótimos, cujo escore é igual ao escore ótimo.*

A Figura 5 apresenta um alinhamento ótimo entre duas sequências, com $score^{opt}$ = +1.

Definição 2.13 (soma de vetores)**.** *Dados dois vetores $V_1 = [x_1, \ldots, x_n]$ e $V_2 = [y_1, \ldots, y_n]$ de n inteiros, a soma destes vetores é o vetor $sum(V_1, V_2) = [x_1 + y_1, \ldots, x_n + y_n]$.*

2.4 ALINHAMENTO DE SEQUÊNCIAS

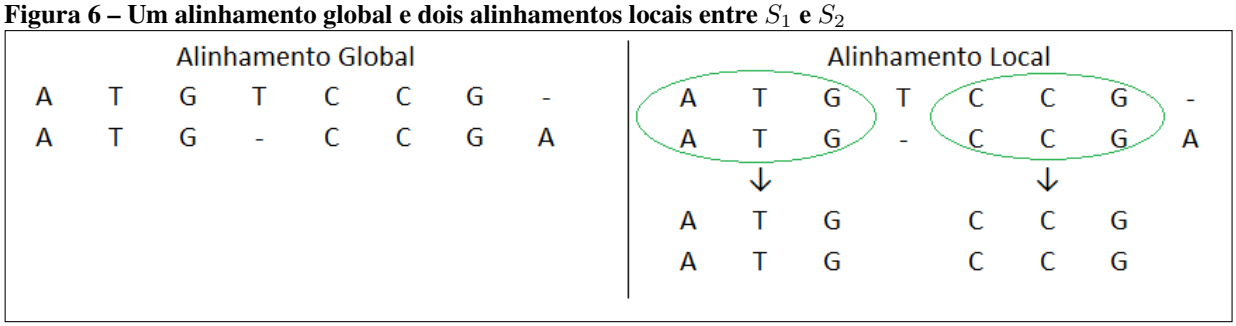
O alinhamento de sequências é utilizado em outros procedimentos na área da bioinformática, como na construção de árvores filogenéticas, visualização na evolução de uma família de proteínas e na busca em base de dados. Por isso se torna uma das tarefas mais importantes da área. Com o alinhamento de sequências pode-se identificar o quão similares as cadeias são, sejam elas de DNA, RNA ou de proteínas.

As sequências podem ser alinhadas em pares e este processo é denominado de alinhamento simples de sequências, para o qual ja se conhecem algoritmos exatos com complexidade de tempo e espaço $O(mn)$, onde m e n são o comprimento das sequências. Também pode-se alinhar três ou mais sequências simultaneamente, o que é chamado de alinhamento múltiplo, um problema NP-difícil (WANG; JIANG, 1994) para o qual é comum o uso de métodos heurísticos.

Além da classificação dos alinhamentos pelo número de sequências dadas como entrada, também pode-se classificá-los de acordo com o objetivo: alinhar as sequências inteiras ou alinhar os trechos mais similares das sequências. Um alinhamento é considerado global caso ele se estenda por todo o comprimento das sequências envolvidas. O alinhamento local consiste no alinhamento de regiões das sequências, ou seja, encontra as *substrings* mais similares nas sequências.

Dadas duas sequências $S_1 = (A, T, G, T, C, C, G)$ e $S_2 = (A, T, G, C, C, G, A)$, a Figura 6 apresenta um alinhamento global e dois alinhamentos locais dentre um conjunto de muitos possíveis.

Figura 6 – Um alinhamento global e dois alinhamentos locais entre S_1 e S_2



Fonte: Autoria própria

Existem vários métodos para alinhar sequências e o uso deles varia de acordo com a necessidade da aplicação, todos possuem suas vantagens e desvantagens.

Como discutimos no Capítulo 1, o problema de encontrar vários alinhamentos ótimos de sequências provavelmente não possui algoritmos exatos subquadráticos, o que torna os algoritmos existentes inviáveis para algumas sequências das bases de dados do NCBI. Desta forma, métodos que trabalham de forma heurística são bastante utilizados, devido à sua agilidade nas respostas, podendo ser 40 vezes mais rápidos que algoritmos exatos (LAU, 2000). No entanto, podem causar perdas de resultados relevantes e induzir o usuário ao erro, devido à sua busca por proximidade.

Na procura de um método que retorna o resultado ótimo, deparamo-nos com os algoritmos exatos, eficientes, porém inviáveis para sequências grandes.

2.5 ALGORITMOS EXATOS

Uma abordagem para calcular o alinhamento entre duas sequências poderia ser gerar e testar os escores de todos os alinhamentos possíveis e depois escolher o melhor. No entanto, o número de alinhamentos entre duas sequências é exponencial no tamanho das sequências (SETUBAL; MEIDANIS, 1997), e tal abordagem resultaria em um algoritmo ineficiente.

Uma forma de resolver o problema de alinhamento de sequências é com a utilização de programação dinâmica, onde se divide o problema (sequência completa) em vários subproblemas (segmentos de sequência curta) mais simples, armazenando as soluções de cada subproblema para uso posterior a fim de determinar uma solução ótima, ou seja, o melhor alinhamento possível (SETUBAL; MEIDANIS, 1997). A ideia é evitar que algum subproblema seja resolvido mais de uma vez.

No que segue, até o final deste documento, consideramos o problema de encontrar ali-

nhamentos globais ótimos entre duas sequências A e B, com comprimentos respectivamente m e n.

2.5.1 Algoritmo de Needleman–Wunsch (NW)

Needleman e Wunsch (1970) propuseram um algoritmo baseado em programação dinâmica com complexidade de tempo e espaço $O(mn)$ utilizado para o alinhamento global de duas sequências com a capacidade de garantir alinhamentos ótimos em pares de sequências. O custo de tempo para gerar um alinhamento ótimo é $O(m + n)$, portanto, para gerar q alinhamentos ótimos o algoritmo gasta $O(mn + q(m + n))$ tempo. É importante lembrar que os alinhamentos globais realizam o pareamento de todos os caracteres entre as sequências analisadas.

O algoritmo possui as seguintes etapas: inicialização, preenchimento da matriz de pontuação e alinhamento (*traceback*).

Sendo duas sequências A e B de tamanho m e n respectivamente, cria-se uma matriz M de tamanho $m + 1$ por $n + 1$.

Primeiro, inicializa o elemento $M_{0,0} = 0$. Em seguida, a primeira coluna e a primeira linha da matriz são inicializados da seguinte forma, $M_{i,0} = -i \times G$ e $M_{0,j} = -j \times G$, sendo G a penalidade do *gap*.

Conforme podemos ver na Figura 7, onde é computado o alinhamento ótimo entre as sequências $A = (A, T, C, G, T, A)$ e $B = (A, T, G, T, G, A)$, a sbt demonstrada na Tabela 3 (p. 17) e um valor de *Gap* igual a -2.

Figura 7 – Matriz de inicialização

		-	A	T	C	G	T	A
	-	0	-2	-4	-6	-8	-10	-12
A		-2						
T		-4						
G		-6						
T		-8						
G		-10						
A		-12						

Fonte: Autoria própria

Utilizando a técnica de programação dinâmica, é preenchida toda a matriz M, do canto superior esquerdo ao canto inferior direito, gravando o escore do alinhamento de A até a posição

i e B até a posição j, e a célula vizinha ($M_{i,j-1}$, $M_{i-1,j}$ ou $M_{i-1,j-1}$) que originou seu escore, de acordo com a Equação 2.2, a seguir.

$$M_{i,j} = \max \begin{cases} M_{i-1,j-1} + sbt(a_i, b_i) \\ M_{i,j-1} - G \\ M_{i-1,j} - G \end{cases} \tag{2.2}$$

Como exemplo, a matriz M preenchida para as sequências (A, T, C, G, T, A) e (A, T, G, T, G, A) é apresentada na Figura 8. Observa-se que na posição (m, n) da matriz encontra-se o $score^{opt}$ entre A e B.

Por fim é feito o processo de *traceback* que consiste em iniciar na posição (m, n) da matriz, percorrendo a matriz do canto inferior direito ao canto superior esquerdo. Se na célula $M_{i,j}$ prosseguirmos para a diagonal ($M_{i-1,j-1}$), alinha o caractere de uma sequência com o da outra, sendo esse alinhamento um *match* ou um *mismatch*. Se prosseguirmos para cima ($M_{i-1,j}$), alinhamos B com um *gap*, ou seja, insere *gap* na sequência A. Se prosseguirmos para a esquerda ($M_{i,j}$), alinhamos A com um *gap*. Essas escolhas são determinadas quando o máximo da Equação 2.2 foi tomado.

Em alguns casos, o processo de *traceback* pode ter mais de um caminho a ser percorrido, o que permite ter mais de um alinhamento global ótimo. Na Figura 8 é possível visualizar a matriz de programação dinâmica gerada pelo alinhamento. As células em destaque indicam um percurso de *traceback*.

Figura 8 – Matriz de programação dinâmica preenchida

	-	A	T	C	G	T	A
-	0	-2	-4	-6	-8	-10	-12
A	-2	**1**	-1	-3	-5	-7	-9
T	-4	-1	**2**	**0**	-2	-4	-6
G	-6	-3	0	1	**1**	-1	-3
T	-8	-5	-2	-1	0	**2**	0
G	-10	-7	-4	-3	0	**0**	1
A	-12	-9	-6	-5	-2	-1	**1**

Fonte: Autoria própria

Vale ressaltar que apenas para obter o $score^{opt}$ não se torna necessário armazenar a matriz inteira. Pode-se perceber que a derivação da linha i da matriz de programação dinâmica utiliza apenas informações da linha acima $(i - 1)$, portanto, uma matriz de tamanho $2(n + 1)$ pode ser utilizada para o cálculo do $score^{opt}$.

2.5.1.1 Análise de complexidade

A etapa de inicialização representada na Figura 7 possui dois laços que consomem tempo $O(m)$ e $O(n)$, respectivamente.

Outros dois laços aninhados são necessários para realizar a etapa de preenchimento da matriz, representada na Figura 8, em que realizamos uma operação $O(1)$ — a tomada do máximo da Equação 2.2 — em cada célula da matriz. Assim, o algoritmo consome tempo $O(mn)$ nesta etapa e este é o termo dominante na complexidade do tempo.

A construção do alinhamento (processo de *traceback*) é feita em $O(tam)$, onde tam é o comprimento do alinhamento retornado — que é $O(m + n)$ no pior caso, quando as duas sequências são alinhadas com *gaps*.

O espaço usado é também proporcional ao tamanho da matriz. Assim, a complexidade de espaço do algoritmo é $O(mn)$.

O algoritmo possui uma complexidade $O(mn)$ de tempo e espaço. Como se está trabalhando com grandes quantidades de dados, o alcance de uma solução pode demorar dias, semanas ou anos para ser encontrada. Além do tempo, o espaço é muitas vezes o fator limitante no cálculo dos alinhamentos ótimos de sequências. Se a entrada para um alinhamento global for duas sequências de DNA com 10^8 pares de bases cada uma[1], a matriz necessária para a execução do Algoritmo de Needleman e Wunsch (1970) necessitará de pelo menos 4×10^{16} bytes, ou seja, 40 petabytes de memória RAM.

2.5.2 Algoritmo de Hirschberg (original)

O algoritmo proposto por Hirschberg (1975) usa o cálculo baseado em programação dinâmica e o paradigma de dividir e conquistar para encontrar a subsequência comum mais longa, denotada por S. Posteriormente, foi estendido para diferentes adaptações de algoritmos de comparações de sequências. O algoritmo possui complexidade de tempo $O(mn)$ e complexidade de espaço $O(m + n)$.

Dadas as sequências $A_{1...m}$ e $B_{1...n}$, recursivamente o Algoritmo 2, apresentado a seguir, reduz o tamanho da instância do problema até que o problema se torne trivial. No caso trivial, se $|B| = 0$, então não há nenhum caractere em B para alinhar, fazemos $S \leftarrow \emptyset$. Se $|A| = 1$, então há apenas um caractere de A para alinhar, portanto, percorre-se toda a sequência de B restante a fim de encontrar um caractere comum para atribuir a S. Se não existir tal caractere em B, fazemos $S \leftarrow \emptyset$.

A redução do tamanho do problema ocorre dividindo a sequência A ao meio ($i =$

[1] De acordo com Setubal e Meidanis (1997), é o tamanho típico de uma sequência de DNA, conforme discutido na Seção 2.1.

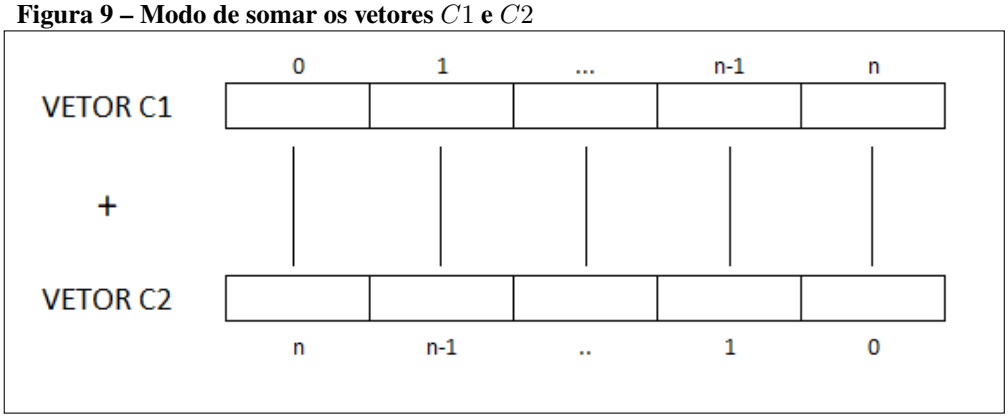
$\lfloor |A|/2 \rfloor$), de forma que a concatenação das subsequências comuns se torne a própria subsequência comum mais longa entre A e B.

Define-se por k a coluna ideal para particionar B. O vetor $C1$ armazena os escores de um alinhamento entre $A_{1...i}$ e $B_{1...n}$. O vetor $C2$ contém os escores de um alinhamento entre $rev(A[i+1..m])$ e $rev(B)$. A soma dos vetores $C1$ e $C2$ ocorre da seguinte forma: $C1(j) + C2(n-j)$ quando $0 \leq j \leq n$ e será representada por: $sum(C1, rev(C2))$, como se pode verificar na Figura 9. O índice k que procuramos corresponderá à posição do maior valor em $sum(C1, rev(C2))$.

Durante o cálculo de $sum(C1, rev(C2))$ pode-se obter resultados máximos iguais para diferentes valores de j, o que abre possibilidades para obter mais de uma subsequência comum mais longa. Caso tenha mais de um resultado com o valor máximo na soma dos vetores, k recebe o menor índice (j) do vetor $C1$ dentre as que possuem resultados máximos iguais, o que garante a subsequência comum mais à esquerda.

Observe que $C1(j)$ terá o valor de comprimento máximo de qualquer subsequência comum de $A_{1...i}$ e $B_{1...j}$. O valor de k define que a subsequência comum mais longa entre A e B passa pela posição (i, j).

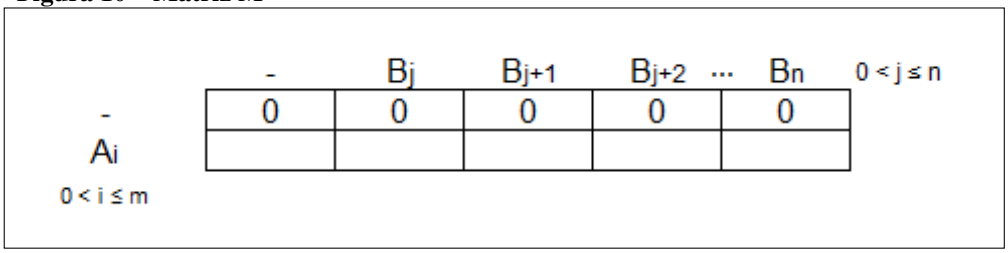
Figura 9 – Modo de somar os vetores $C1$ e $C2$



Fonte: Autoria própria

Os escores armazenados nos vetores $C1$ e $C2$ são resultantes de um cálculo realizado na matriz M, onde M possui tamanho aproximado de $2n$, exemplificado na Figura 10.

Figura 10 – Matriz M



Fonte: Autoria própria

Pode-se perceber que a derivação da linha i de uma matriz de programação dinâmica requer apenas informações da linha $i-1$. Portanto, não há necessidade de utilizar uma matriz de

tamanho mn como visto na Figura 8 para o cálculo de alinhamento, bastando substituir $M(0, j)$ para todo $j \in \{0, \ldots, n\}$ por $M(1, j)$ para todo $j \in \{0, \ldots, n\}$ e incrementar o valor de i a cada iteração.

Por fim, quando $i > m$, basta armazenar a última linha da matriz M em um vetor.

O cálculo da matriz M é realizado de acordo com o Algoritmo 1.

Algoritmo 1:

Entrada: m, n, A, B, CC

1 **início**
2 Inicialização: $M(1, j) \leftarrow 0$ $[j = 0...n]$
3 **para** $i \leftarrow 1$ **até** *m* **faça**
4 $M(0, j) \leftarrow M(1, j)$ $[j = 0...n]$
5 **para** $j \leftarrow 1$ **até** *n* **faça**
6 **se** $A(i) = B(j)$ **então**
7 $M(1, j) \leftarrow M(0, j - 1) + 1$
8 **fim**
9 **senão**
10 $M(1, j) \leftarrow \max[M(1, j - 1), M(0, j)]$
11 **fim**
12 **fim**
13 **fim**
14 $CC(j) \leftarrow M(1, j)$ $[j = 0...n]$
15 **fim**
16 **retorna** CC

Pode-se verificar que para realizar os cálculos que definem CC, o Algoritmo 1 não utiliza uma matriz de substituição. Como o objetivo é encontrar apenas a subsequência comum mais longa, o grau evolutivo de cada par de caracteres se limita a 1, quando os caracteres são iguais, ou 0, quando são diferentes.

Na linha 14 do Algoritmo 1, são inseridos no vetor CC a última linha da matriz M e na linha 16 retornado CC para o vetor $C1$ ou $C2$ expresso no Algoritmo 2.

O pseudocódigo do Algoritmo 2 é representado a seguir.

Algoritmo 2:

Entrada: m, n, A, B, S

início
  // Resolver problema trivial:
  se $n = 0$ **então**
    $S \leftarrow e$ (e é a string vazia)
  fim
  senão se $m = 1$ **então**
    se $A(1) = B(j)$ *tal que* $j < n$ **então**
      $S \leftarrow A(1)$
    fim
    senão
      $S \leftarrow e$
    fim
  fim
  // Caso contrário, reduzir tamanho do problema:
  senão
    $i \leftarrow \lfloor m/2 \rfloor$
    // Utilizar matriz M:
    $Algoritmo1(i, n, A_{1..i}, B_{1..n}, C1)$
    $Algoritmo1(m - i, n, rev(A_{i+1..n}), rev(B_{1...n}), C2)$
    // Encontrar o valor de k:
    $valMax \leftarrow \max\{C1(j) + C2(n - j)\}$ $(0 \leq j \leq n)$
    $k \leftarrow \min j$ tal que $C1(j) + C2(n - j) = valMax$
    // Resolver problemas mais simples:
    $Algoritmo2(i, k, A_{1..i}, B_{1..k}, S_1)$
    $Algoritmo2(m - i, n - k, A_{i+1..m}, B_{k+1..n}, S_2)$
    // Saída:
    $S \leftarrow S_1 S_2$
  fim
fim
retorna S

Obtidos os valores de i e k, o problema é subdividido de forma recursiva até encontrar um caso trivial.

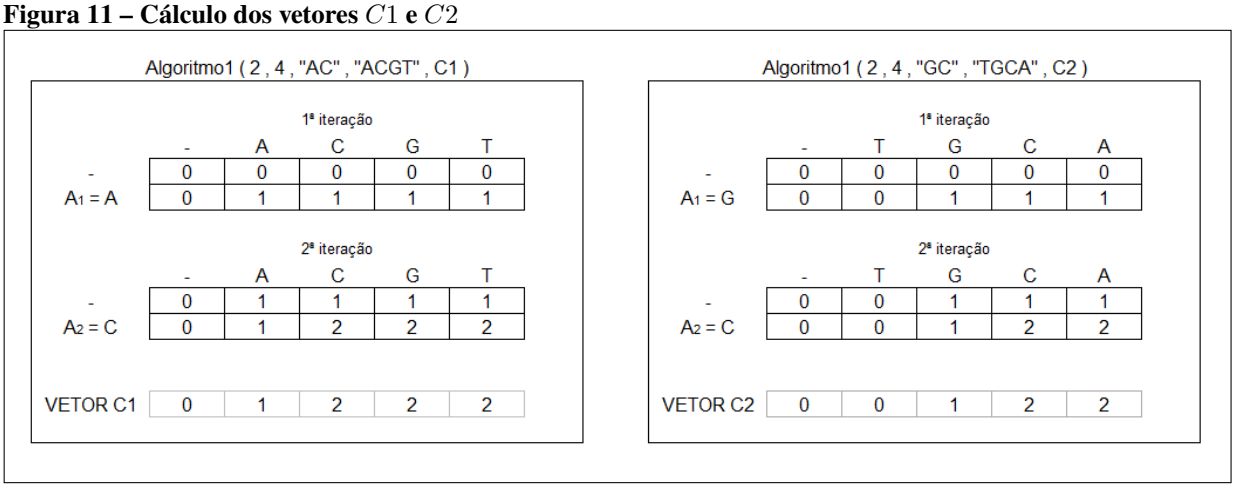
2.5.2.1 Exemplo

Dadas as sequências $A = (A, C, C, G)$ e $B = (A, C, G, T)$, um exemplo do Algoritmo é demonstrado abaixo.

Os valores de m, n e i na primeira recursão do algoritmo 2 respectivamente são $4, 4$ e 2. Duas chamadas para o Algoritmo 1 acontecem com os seguintes parâmetros de entrada, $Algoritmo1(2, 4, A_{1,2}, B_{1..4}, C1)$ e $Algoritmo1(2, 4, rev(A_{3...4}), rev(B_{1...4}), C2)$.

As chamadas para o Algoritmo 1 calculam e retornam os vetores $C1$ e $C2$, explícitos na Figura 11.

Figura 11 – Cálculo dos vetores $C1$ e $C2$



Fonte: Autoria própria

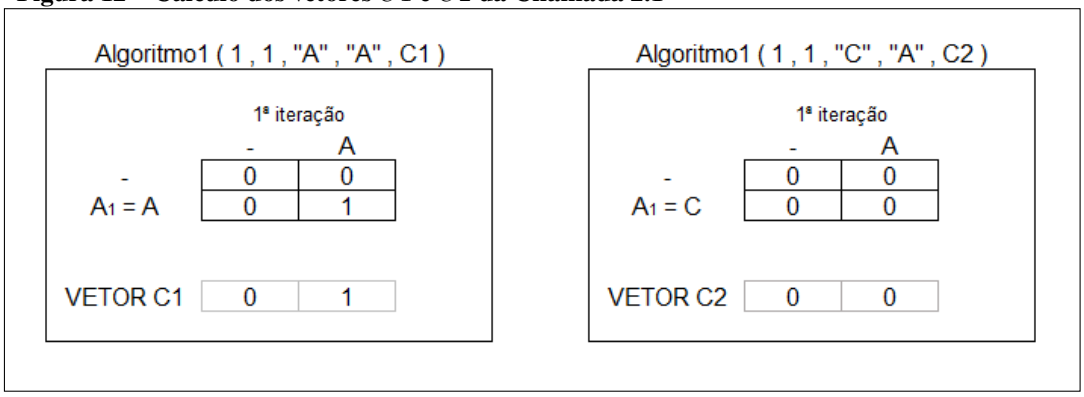
Os vetores $C1$ e $C2$ são somados, posição a posição, conforme apresenta a Figura 9. Para o exemplo, os valores resultantes desta soma, serão 2, 3, 3, 2, e 2, nesta ordem. O máximo dentre tais valores é o valor de $valMax$, ou seja, no exemplo $valMax = 3$. E assim pode-se obter o valor de k, que neste caso é 1. Lembre-se que k é o menor valor de j em que a célula tem valor $valMax$. O valor de k determina o ponto de corte para as próximas chamadas da recursão e, além disso, traz a informação de que a subsequência comum mais longa entre A e B pode ser obtida ao se determinar a subsequência comum mais longa entre $A_{1..2}$ e $B_{1..1}$ e concatená-la com a subsequência comum mais longa entre $A_{3..4}$ e $B_{2..4}$. As chamadas recursivas para esses dois casos serão denotadas, respectivamente, de Chamada 2.1 e Chamada 2.2 para melhor entendimento.

Chamada 2.1

As sequências recebidas por parâmetros são $A = (A, C)$ e $B = (A)$ e os valores de m, n e i são respectivamente $2, 1$ e 1. Os seguintes valores de m, n e i não são considerados para um problema trivial, portanto ocorrem duas chamadas para o Algoritmo 1.

Pode-se verificar na Figura 12 a execução do Algoritmo 1, que retorna os vetores $C1$ e $C2$ para posteriormente calcular o valor de k, que neste caso é 1.

Figura 12 – Cálculo dos vetores $C1$ e $C2$ da Chamada 2.1



Fonte: Autoria própria

Após definir o valor de k , o problema é subdividido com as seguintes chamadas, $Algoritmo2(1, 1, A_1, B_1, S1)$ e $Algoritmo2(1, 0, A_2, , S2)$.

Nota-se que na primeira chamada para o Algoritmo 2 o valor de m é igual a um, tornando-se um caso trivial, portanto atribui o valor A para a string S, retornando-a logo em seguida.

Na segunda chamada para o Algoritmo 2 o problema também se torna um caso base, consequência do valor $n = 0$, retornando um S vazio.

As duas substrings S são concatenadas conforme explícito na linha 27 do Algoritmo 2.

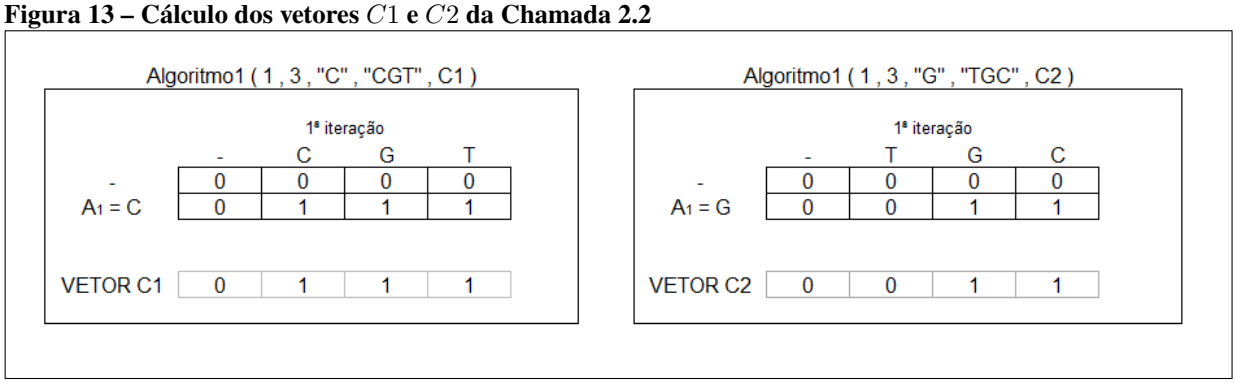
Após retornar da Chamada2.1 com $S1 = (A)$, é executada a Chamada2.2.

Chamada 2.2

Na Chamada2.2, as sequências recebidas por parâmetros são $A = (C, G)$ e $B = (C, G, T)$ e os valores de m, n e i são respecitavamente $2, 3$ e 1.

Duas chamadas para o Algoritmo 1 acontecem e são calculados os vetores $C1$ e $C2$ conforme a Figura 13.

Figura 13 – Cálculo dos vetores $C1$ e $C2$ da Chamada 2.2



Fonte: Autoria própria

Obtêm-se o valor de $k = 1$ e logo, o problema é subdividido com as seguintes chamadas, $Algoritmo2(1, 1, A_3, B_2, S1)$ e $Algoritmo2(1, 2, A_4, B_{3,4}, S2)$.

Nota-se que na primeira chamada para o Algoritmo 2 o valor de m é igual a 1, tornando-se um caso trivial, portanto, atribui o valor C para a string S, retornando-a logo em seguida.

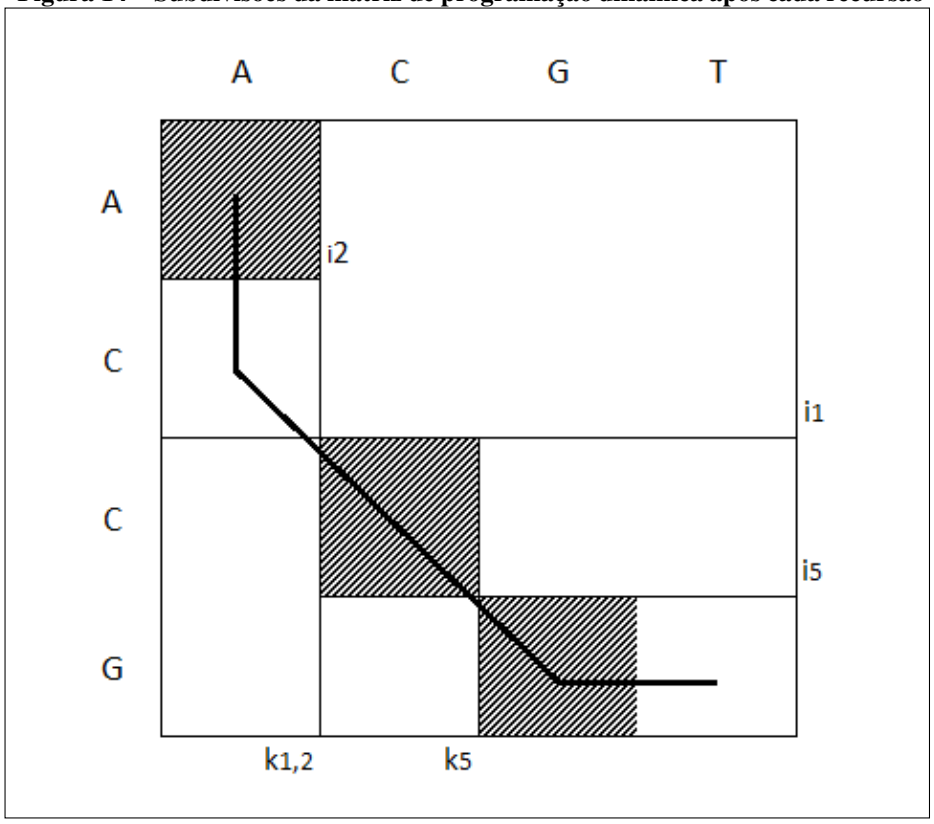
Na segunda chamada para o Algoritmo 2 o problema também se torna um caso base, consequência da mesma circunstância, retornando $S = G$.

As duas substrings S são concatenadas e após o encerramento da Chamada2.2 uma subsequência em $S1 = (A)$ e uma em $S2 = (CG)$ são obtidas.

É possível verificar que na linha 27 do Algoritmo 2 as saídas são concatenadas e a subsequência comum mais longa entre A e B, no caso (ACG), é retornada.

A Figura 14 apresenta uma visão geral das subdivisões da matriz de programação dinâmica após cada recursão, pode-se verificar onde ocorreu cada divisão das sequências A e B pelas letras i e k respectivamente e o número da recursão representado pelos índices de i e k.

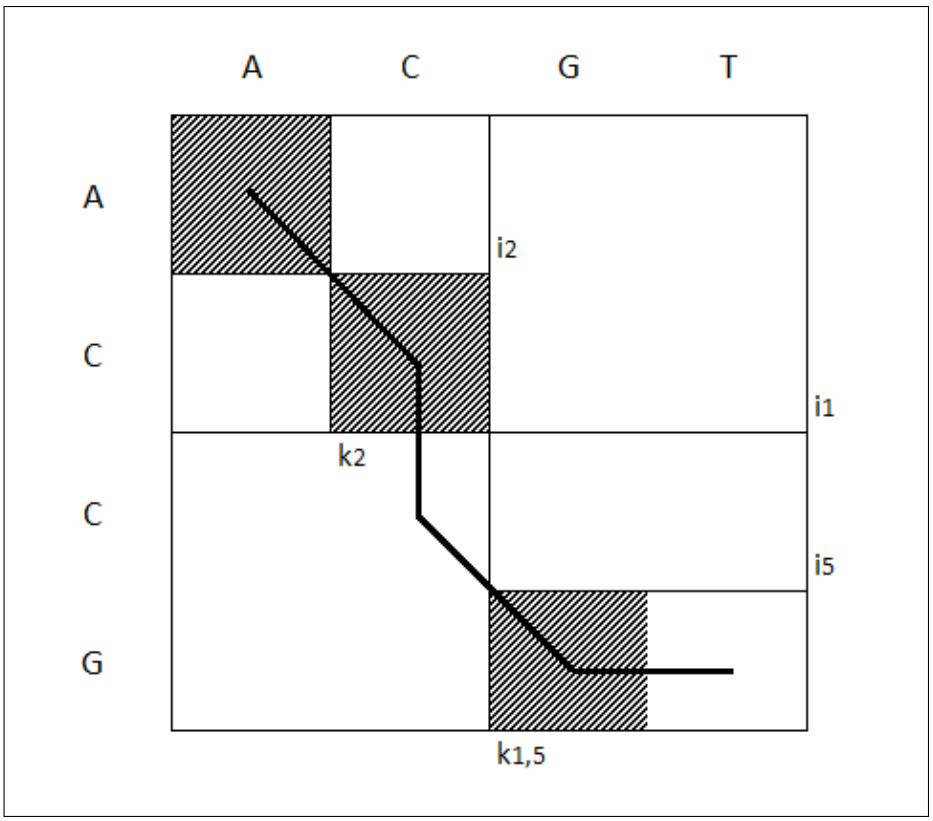
Figura 14 – Subdivisões da matriz de programação dinâmica após cada recursão



Fonte: Autoria própria

Uma alteração na linha 22 do Algoritmo 2 para: $k \leftarrow \max\{j$ tal que $C1(j) + C2(n - j) = valMax\}$ devolveria a mesma subsequência comum mais longa e resultaria nas seguintes subdivisões da matriz de programação:

Figura 15 – Subdivisões da matriz de programação dinâmica após alteração na linha 22



Fonte: Autoria própria

Nota-se que devido a $sum(C1, rev(C2))$ apresentarem mais de uma coluna com o valor máximo ($j = 1$ e $j = 2$) na primeira escolha de k, pode-se ter mais de um caminho para chegar na subsequência comum mais longa (A, C, G). Em alguns casos, uma subsequência diferente porém com o mesmo tamanho pode ser gerada.

2.5.2.2 Análise de complexidade

1. Análise do Algoritmo 1

   A instrução **se** na linha 6 é executada exatamente mn vezes. Matrizes de entrada e saída requerem $m + n + (n + 1)$ posições. Assim o Algoritmo 1 tem complexidade de tempo $O(mn)$ e espaço $O(m + n)$.

2. Análise do Algoritmo 2

   Para sequências de tamanhos 1 e n, procura-se uma única correspondência, limitado em tempo por $O(n)$.

   Para sequências de tamanhos m e n, permite-se que operações em vetores que são lineares (veja a análise de espaço do Item 1) em m e n são limitadas pelo tempo por $O(m + n)$.

   Isso deixa duas chamadas para o Algoritmo 1 e duas chamadas para o Algoritmo 2.

O tempo de execução de todas as chamadas do Algoritmo 1 é limitado por $O(mn)$ pela análise de tempo do Item 1. O tempo de execução de todas as chamadas do Algoritmo 2 é limitado por $O(mn)$, sendo $O(mk)$ para chamadas do primeiro tipo (linha 24) e $O(m(n-k))$ para chamadas do segundo tipo (linha 25).

Desta forma, teremos um tempo de $O(mn)$ para chamadas do Algoritmo 2 e $O(mn) + O(n) + O(m)$ para chamadas do Algoritmo 1.

Ou seja, o Algoritmo proposto por Hirschberg (1975) tem complexidade de tempo $O(mn)$.

Assumimos que os vetores estão em armazenamento comum e as *substrings* podem ser transferidas como argumentos, dando localizações iniciais e finais. Então, durante a execução, as chamadas para o Algoritmo 1 usam armazenamento temporário que é linear em m e n (veja a análise de espaço do Item 1). Portanto, o Algoritmo requer espaço de memória proporcional a m e n, ou seja, espaço $O(m + n)$.

2.5.3 Algoritmo de Hirschberg (alinhamento de sequências)

O algoritmo original de Hirschberg (1975) é capaz de retornar a subsequência comum mais longa. Uma adaptação para gerar e retornar um alinhamento global ótimo é explicada a seguir (MYERS; MILLER, 1988).

O algoritmo original de Hirschberg (1975) não possui uma função de penalidade para os *gaps* e a relação evolutiva entre os caracteres das sequências se limita a, quando dois caracteres são diferentes o valor permanece o mesmo e quando iguais soma-se 1.

Nesta adaptação a forma de pontuar a matriz M é alterada, como agora a busca é por um alinhamento global ótimo, o grau evolutivo é de suma importância, portanto, são usadas a Equação 2.2 e consequentemente uma matriz de substituição para pontuar as células da matriz, conforme se pode verificar na Linha 9 do Algoritmo 3.

No Algoritmo 1, a matriz M é iniciada com valores iguais a 0 e a célula $(1, 0)$ é inalterada e permanece nela o valor 0 até o fim do algoritmo. As linhas 4 e 7 do Algoritmo 3 o deixam

capaz de inicializar e pontuar a matriz conforme um valor de *gap* G predeterminado.

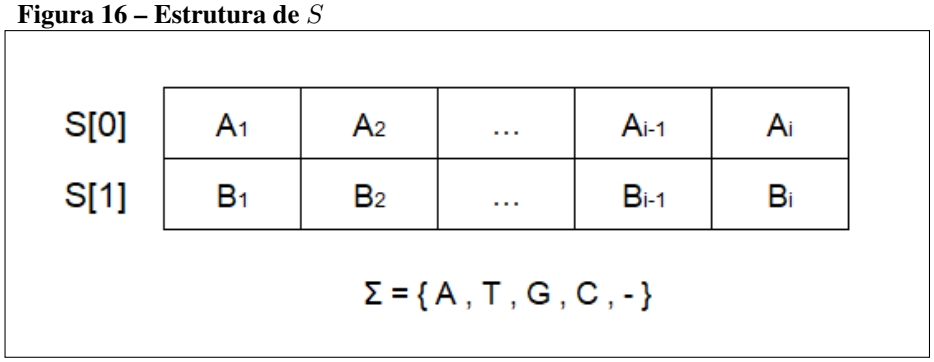
Algoritmo 3:

Entrada: m, n, A, B, CC

início
  Inicialização:
  $M(1,0) \leftarrow 0$
  $M(1,j) \leftarrow M(1,j-1) - G\ [j = 1...n]$ (G é o valor do *gap*)
  para $i \leftarrow 1$ **até** m **faça**
    $M(0,j) \leftarrow M(1,j)\ [j = 0...n]$
    $M(1,0) \leftarrow M(0,0) - G$
    para $j \leftarrow 1$ **até** n **faça**
      $M(1,j) \leftarrow$
      $\max[M(1,j-1) - G, M(0,j) - G, M(0,j-1) + sbt(A(i), B(j))]$
    fim
  fim
  $CC(j) \leftarrow M(1,j)\ [j = 0...n]$
fim
retorna CC

A mudança no retorno do Algoritmo, que deixou de ser uma subsequência comum e agora é um alinhamento ótimo implicou na mudança de S. Como visto na Figura 4, um alinhamento contém duas *strings* pareadas, portanto S passou a representar duas *strings*, onde $S[0]$ representa uma *string* da sequência pareada de A e $S[1]$ uma *string* da sequência pareada de B, onde i é o índice do último caractere pareado.

Figura 16 – Estrutura de S



Fonte: Autoria própria

Para um problema trivial, surgiu a necessidade de alterar a atribuição de valores para S, consequência da sua nova estrutura representada na Figura 16. Outro fator importante é que agora suas *strings* contém *gaps* e *mismatches*. Para sequências de DNA, o alfabeto considerado é $\Sigma = \{A, T, G, C, -\}$.

Para um problema trivial, se $|B| = 0$, B não possui nenhum caractere para alinhar com m caractere(s) de A, portanto $S[0]$ recebe os caracteres de A e $S[1]$ recebe $|A|$ *gaps*.

Se $|A| = 1$, então há apenas um caractere de A para alinhar com n caracteres de B, portanto, percorre-se toda sequência de B restante a fim de encontrar um caractere comum para atribuir a $S[0]$ e $S[1]$. Enquanto não acha e após achar um caractere comum, $S[0]$ é preenchido com *gaps* e $S[1]$ com caracteres de B. Se não existir um caractere comum entre $A(1)$ e $B_{1...n-1}$, considera-se um *mismatch* entre $A(1)$ e $B(n)$. Ou seja, uma coluna do alinhamento entre $A(1)$ e $B_{1...n}$ será um *match* ou *mismatch* e o restante *gap(s)* em $S[0]$ e os caracteres de B em $S[1]$.

O Algoritmo 4 demonstra o pseudocódigo para casos triviais do problema do alinhamento global de duas sequências.

Algoritmo 4: Problema Trivial

```
Entrada: m, n, A, B, S
1  início
2      se n = 0 então
3          para i ← 1 até m faça
4              S[0] ← A(i)
5              S[1] ← −
6          fim
7      fim
8      senão se m = 1 então
9          para j ← 1 até n faça
10             se A(1) = B(j) então
11                 // Match:
12                 S[0] ← A(1)
13                 S[1] ← B(j)
14                 para p ← j até n faça
15                     S[0] ← −
16                     S[1] ← B(p)
17                 fim
18                 retorna S
19             fim
20             senão
21                 se j = n então
22                     // Mismatch:
23                     S[0] ← A(1)
24                     S[1] ← B(j)
25                 fim
26                 senão
27                     Gap:
28                     S[0] ← −
29                     S[1] ← B(j)
30                 fim
31             fim
32         fim
33     fim
34 fim
35 retorna S
```

A etapa de divisão do problema não é alterada e pode ser visualizada no Algoritmo 5.

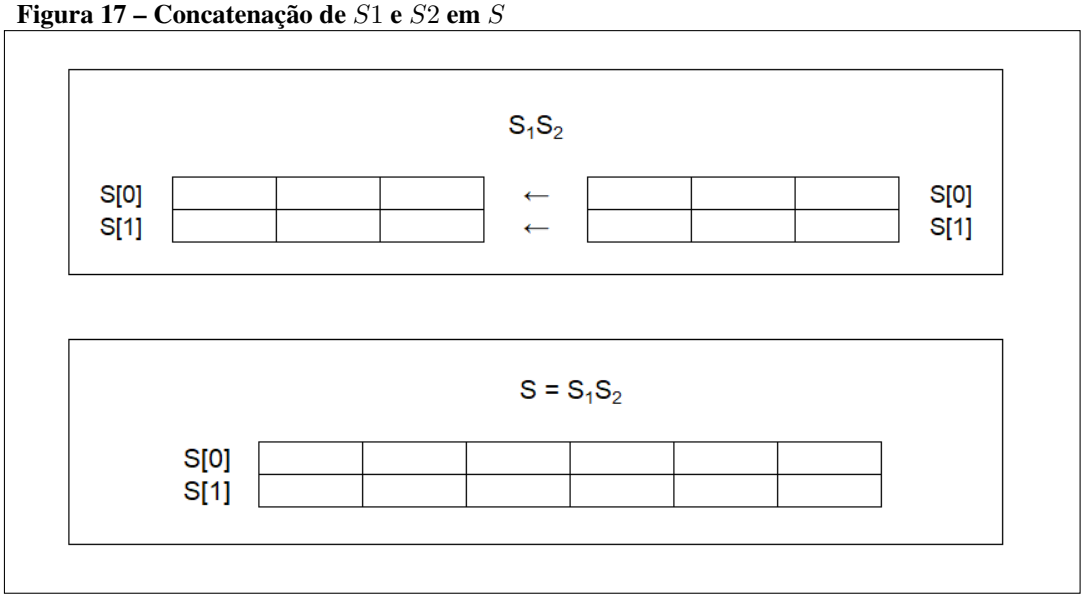
Algoritmo 5:

Entrada: m, n, A, B, S

início
  // Resolver problema trivial:
  se $n = 0$ *ou* $m = 1$ **então**
    $Algoritmo4(m, n, A, B, S)$
  fim
  // Caso contrário, reduzir o tamanho do problema:
  senão
    $i \leftarrow \lfloor m/2 \rfloor$
    // Utilizar a matriz M:
    $Algoritmo3(i, n, A_{1..i}, B_{1..n}, C1)$
    $Algoritmo3(m - i, n, rev(A_{i+1..n}), rev(B_{1...n}), C2)$
    // Encontrar o valor de k:
    $valMax \leftarrow \max\{C1(j) + C2(n - j)\}\ (0 \leq j \leq n)$
    $k \leftarrow \min j$ tal que $C1(j) + C2(n - j) = valMax$
    // Resolver problemas mais simples:
    $Algoritmo5(i, k, A_{1..i}, B_{1..k}, S_1)$
    $Algoritmo5(m - i, n - k, A_{i+1..m}, B_{k+1..n}, S_2)$
    // Saída:
    $S \leftarrow S_1 S_2$
  fim
fim
retorna S

A concatenação das *strings* retornadas (S_1 e S_2) ocorre de forma com que $S[0]$ da variavel S_1 concatena com $S[0]$ de S_2 e igualmente para $S[1]$, exemplificado na Figura 17.

Figura 17 – Concatenação de $S1$ e $S2$ em S



Fonte: Autoria própria

O algoritmo original de Hirschberg (1975) e esta adaptação não possuem o processo de *traceback*, ou seja, não possuem as informações (pontuações) anteriores da matriz M e dos vetores $C1$ e $C2$ guardadas para que seja possível um retorno e a escolha de um alinhamento diferente. Gerado o alinhamento, o último cálculo da matriz M (que ocorre na segunda recursão do Algoritmo 5) e os vetores $C1$ e $C2$ são as únicas informações guardadas, informações essas que não são suficientes para executar um processo de *traceback*. E é importante notar que métodos que usam informações de *traceback* necessitam de mais espaço para encontrar o(s) alinhamento(s) ótimo(s). Desta forma, esta adaptação é capaz de gerar um alinhamento global ótimo em espaço linear.

2.5.4 Análises de complexidade de tempo e espaço

1. Análise do Algoritmo 3

   Os laços das linhas 5 e 8 fazem com que a instrução da linha 9 seja executada mn vezes. A matriz M e o vetor de saída requerem espaço $O(n)$. Assim o Algoritmo 3 tem complexidade de tempo $O(mn)$ e espaço $O(n)$.

2. Análise do Algoritmo 4

   Para sequências de tamanho $n = 0$ o laço da linha 3 do algoritmo é executado m vezes.

   Para sequências com $m = 1$, a soma do número de execuções das linhas 9 e 14 é n. Sendo assim, para $m = 1$ a complexidade de tempo é limitada por n.

   Desta forma, o Algoritmo 4 possui uma complexidade de tempo $O(m + n)$ (assumindo os dois casos).

3. Análise do Algoritmo 5

   Para sequências de tamanhos $n = 0$ ou $m = 1$, a complexidade é limitada pelo tempo em $O(m + n)$ (análise de tempo do Item 2).

   Para sequências de tamanhos m e n, duas chamadas para o Algoritmo 3 são realizadas. As chamadas são limitadas por $O(mn)$ pela análise de tempo do Item 1.

   Encontrar o valor de k é limitado pelo tempo em $O(n)$

   Resolver problemas mais simples requer duas chamadas para o Algoritmo 5. Essas chamadas serão limitadas pelo tempo por $O(mk)$ para a primeira chamada (linha 16) e $O(m(n-k))$ para a segunda chamada, totalizando $O(mn)$. Desta forma, teremos um tempo de $O(mn) + O(mn) + O(m + n) + O(n)$.

   Assumimos que os vetores e matrizes estão em armazenamento comum e as *substrings* podem ser transferidas como argumentos, dando localizações iniciais e finais. Então, durante a execução, as chamadas para o Algoritmo 3 usam armazenamento temporário que é linear em m e n (veja a análise de espaço do Item 1). Portanto, o Algoritmo requer espaço de memória proporcional a m e n.

   Ou seja, esta adaptação é capaz de gerar um alinhamento ótimo em espaço $O(m + n)$ e tempo $O(mn)$.

3 UM NOVO MÉTODO DE ALINHAMENTO EM ESPAÇO LINEAR

Segundo Myers e Miller (1988), quando se está pesquisando por arranjos "biologicamente significativos" é comum considerar vários alinhamentos em suas pesquisas. Como o Algoritmo de Hirschberg (1975) gera apenas um alinhamento ótimo, apresentamos a seguir uma adaptação no algoritmo capaz de gerar mais de um alinhamento ótimo (caso haja), mantendo a complexidade de espaço linear na entrada e polinomial no tamanho total da entrada e da saída. Vamos nos referir a esta adaptação como "Algoritmo de Hirschberg estendido". Vale ressaltar que as *strings* iniciais não podem ser vazias.

3.1 IDENTIFICAÇÃO DO PROBLEMA

3.1.1 Caso base ($m = 1$)

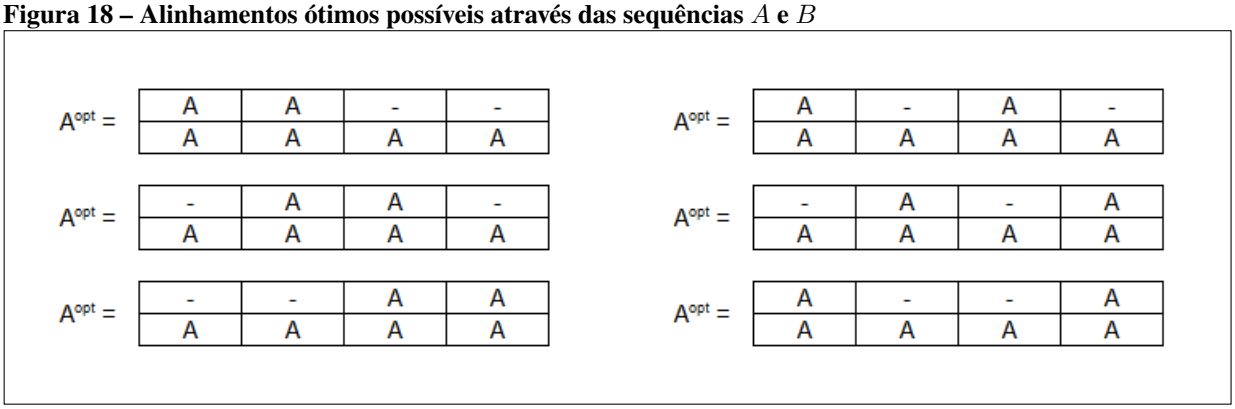
Um *mismatch* no caso base ocorre quando $A(1) \neq B(j)$ para todo $j \in \{1, \ldots, n\}$ e é capaz de gerar $|n|$ alinhamentos ótimos, pois o pareamento de $A(1)$ e $B(j)$ vai gerar o mesmo escore para todo valor de j. A variável x armazena $|n|$.

Um *match* no caso base ocorre quando $A(1) = B(j)$ para todo $j \in \{1, \ldots, n\}$. Pode-se ter mais de um *match* no caso base e isso ocorre quando $A(1) = B(j)$ para mais de um valor de j. A variável x armazena quantos *matches* são possíveis.

Portanto, quando $m = 1$ pode-se ter x alinhamentos ótimos.

Dadas as sequências $A = (A, A)$ e $B = (A, A, A, A)$, têm-se 6 alinhamentos ótimos possíveis (Figura 18).

Figura 18 – Alinhamentos ótimos possíveis através das sequências A e B

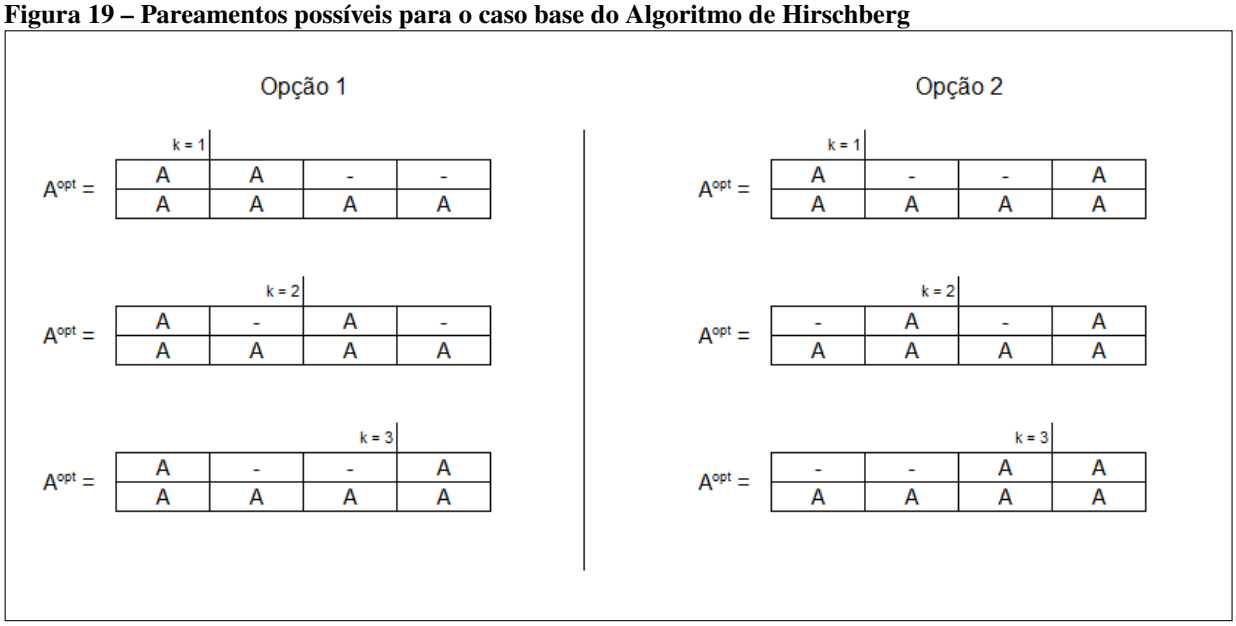


Fonte: Autoria própria

Iniciando o Algoritmo de Hirschberg, três possiveis valores para k (1, 2 e 3) são obtidos após reduzir o tamanho do problema ($i \leftarrow \lfloor m/2 \rfloor$). Pode-se observar na Opção 1 da Figura 19

que, independente do valor de k, o primeiro pareamento dos caracteres sempre será um *match*, pois a resolução do problema trivial ($m = 1$) se limita a parear com *match* o primeiro caractere igual entre A e B. Como consequência, não ocorre a combinação de todas as possibilidades de pareamentos disponíveis.

Figura 19 – Pareamentos possíveis para o caso base do Algoritmo de Hirschberg



Fonte: Autoria própria

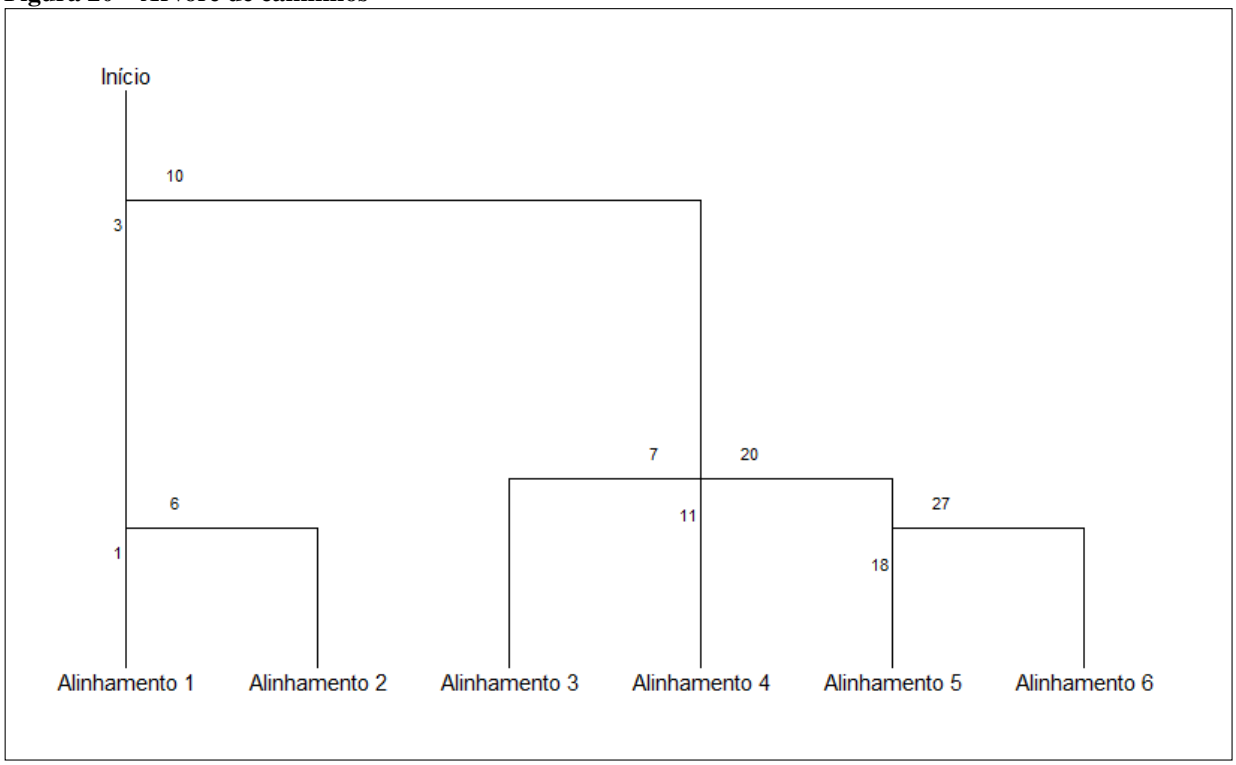
Quando se impõem a restrição de que o último par de caracteres iguais deve ser um *match*, apenas os alinhamentos apresentados na Opção 2 da Figura 19 são gerados. Para garantir que em todos os casos x alinhamentos ótimos sejam gerados, uma alteração no Algoritmo 4 é necessária.

3.1.2 Escolha de um valor para k

No algoritmo proposto por Hirschberg (1975) define-se por k a coluna ideal para particionar B, onde k é o índice da coluna que resulta na maior soma entre os vetores $C1$ e $C2$. Para mais de um valor máximo entre a soma dos vetores, mais de uma coluna para particionar B pode ser escolhida e qualquer k escolhido resultará em um alinhamento ótimo entre A e B.

Dadas duas sequências A e B, após executar o Algoritmo de Hirschberg obtém-se um alinhamento ótimo e consequentemente um caminho gerado por ele. Conforme citado anteriormente, escolhendo diferentes valores para k, diferentes caminhos e alinhamentos ótimos podem surgir. Na Figura 20 pode-se observar que em determinadas chamadas mais de um k pode ser escolhido. Por exemplo, a escolha dos menores valores possíveis para k resulta no Alinhamento 1, escolhendo os maiores valores possíveis para k obtém-se o Alinhamento 6. Vale ressaltar que até a primeira bifurcação, os Alinhamentos 1 e 6 eram o mesmo.

Figura 20 – Árvore de caminhos



Fonte: Autoria própria

O Algoritmo de Hirschberg é capaz de gerar apenas o Alinhamento 1. Para obter mais de um alinhamento ótimo, mais de um k deve ser considerado e, portanto, alterações na escolha de um caminho (valor de k) são necessárias.

3.2 RESOLUÇÃO DO PROBLEMA

A identificação, o armazenamento e a busca por novas escolhas se torna a base para percorrer novos caminhos e consequentemente gerar novos alinhamentos ótimos. Identificado mais de um possível valor para k ou para x, as informações de quantas escolhas possíveis há e de qual escolha foi tomada pelo algoritmo se tornam necessárias para posteriormente percorrer um novo caminho.

O conjunto Y representa uma lista ordenada de todos possíveis k's ou x's e é através dele que determinamos uma escolha para um novo caminho a ser percorrido.

Os valores mínimo, máximo e o tamanho para o conjunto são retornados através das funções $min\{Y\}$, $max\{Y\}$ e $tam\{Y\}$ respectivamente. A função $prox(c, Y)$ retorna o menor elemento no conjunto Y que seja maior que c.

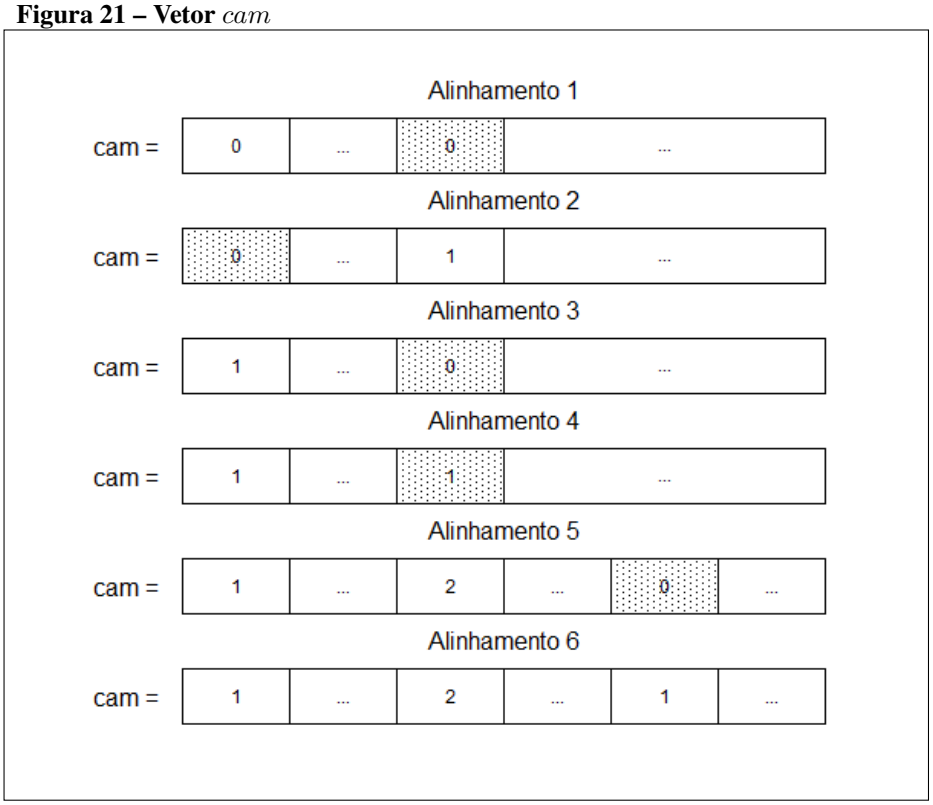
3.2.1 Armazenamento

Um caminho é formado por sucessivas escolhas. Define-se um caminho como um vetor de escolhas, denotado cam, o qual possui também uma variável de controle, capaz de identificar o último índice do vetor com mais de uma possibilidade de nova escolha.

O vetor cam armazena as escolhas do último caminho percorrido e a escolha marcada pela variável de controle é atualizada sempre que há a busca por um possível novo caminho. Após iniciar a busca por um novo caminho as escolhas posteriores do vetor cam são perdidas e substituídas pelo caminho corrente.

A Figura 21 demonstra o caminho final do vetor após gerar os alinhamentos representados na Figura 20.

Figura 21 – Vetor cam



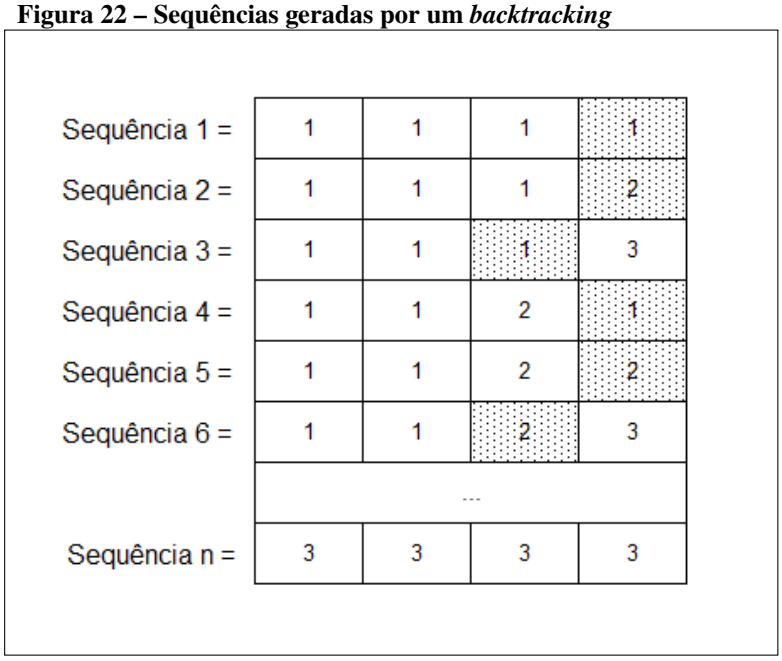
Fonte: Autoria própria

Pode-se observar que nas Figuras 20 e 21 são representadas as escolhas tomadas dentre mais de uma possibilidade de valor, mas vale ressaltar que o vetor armazenará todas as escolhas feitas. As células em destaque indicam a posição da variável de controle antes de iniciar um novo alinhamento.

3.2.2 Backtracking

Dado um vetor qualquer de tamanho 4 e possíveis valores de 1 a 3 para suas células, uma forma de representá-lo está disposto na Figura 22.

Figura 22 – Sequências geradas por um *backtracking*



Fonte: Autoria própria

Pode-se observar que as sequências são geradas de "trás para frente" no vetor e que a célula marcada é a célula alterada na próxima sequência gerada. Nesta adaptação o vetor cam e a variável de controle vão funcionar de forma similar.

Desta forma, percorre-se o vetor seguindo o mesmo caminho até encontrar a célula marcada pela variável de controle. Nesse momento a escolha se torna o próximo elemento do conjunto Y.

É importante observar que a variável de controle marca somente a última célula do vetor com possibilidades de novos caminhos, ou seja, uma célula com duas possibilidades de escolhas onde as duas já foram percorridas não será marcada pela variável de controle.

Nenhuma célula do vetor cam marcada com a variável de controle indica o fim de possíveis novos alinhamentos, como pode-se observar após terminar o Alinhamento 6 da Figura 21.

O Algoritmo de Hirschberg estendido é apresentado nos Algoritmos 6, 7, 8, 9 e 10. O Algoritmo 6 inicializa todas as variáveis e controla o laço para imprimir os alinhamentos ótimos gerados. O Algoritmo 7 estrutura as chamadas para os outros algoritmos. O Algoritmo 8 trata o problema trivial, chamando o Algoritmo 9 com os parâmetros corretos para realizar um *match* ou *mismatch*. O Algoritmo 10 determina um valor para k.

Algoritmo 6: Laço

Entrada: m, n, A, B, S

início
- $cam[] \leftarrow 0$
- $controle \leftarrow 0$
- $controleAux \leftarrow 0$
- Faça
- $idxCam = 0$
- **se** $controle \leq controleAux$ **então**
  - $controle \leftarrow controleAux$
- **fim**
- $controleAux \leftarrow 0$
- $Algoritmo7(m, n, A, B, S)$
- Imprima S
- Enquanto($controle > 0$)

fim

Algoritmo 7: Geral

Entrada: m, n, A, B, S

início
  // Para o problema trivial:
  se $n = 0$ *ou* $m = 1$ **então**
    $Algoritmo8(m, n, A, B, S)$
  fim
  // Caso contrário, reduzir o tamanho do problema:
  senão
    $i \leftarrow \lfloor m/2 \rfloor$
    // Utilizar a matriz M:
    $Algoritmo3(i, n, A_{1..i}, B_{1..n}, C1)$
    $Algoritmo3(m - i, n, rev(A_{i+1..m}), rev(B_{1...n}), C2)$
    // Encontrar o valor de k:
    $Max \leftarrow \max\{C1(j) + C2(n - j) : 0 \leq j \leq n\}$
    $Y \leftarrow \{j \in \{0, \ldots, n\} : C1(j) + C2(n - j) = Max\}$
    $Algoritmo10(Y, k)$
    // Resolver problemas mais simples:
    $Algoritmo7(i, k, A_{1..i}, B_{1..k}, S_1)$
    $Algoritmo7(m - i, n - k, A_{i+1..m}, B_{k+1..n}, S_2)$
    // Saída:
    $S \leftarrow S_1 S_2$
  fim
fim
retorna S

Algoritmo 8: ProblemaTrivial

```
Entrada: m, n, A, B, S
início
    se n = 0 então
        para i ← 1 até m faça
            S[0] ← A(i)
            S[1] ← −
        fim
    fim
    senão se m = 1 então
        para j ← 1 até n faça
            se A(1) = B(j) então
                para p ← j até n faça
                    Y ← p tal que A(1) = B(p)
                fim
                // Match:
                Algoritmo9(Y, n, A, B, S)
                retorna S
            fim
            Y ← j
        fim
        // Mismatch:
        Algoritmo9(Y, n, A, B, S)
    fim
fim
retorna S
```

Algoritmo 9: Caso Base

Entrada: Y, n, A, B, S

início
  se $\max\{Y\} > cam[idxCam]$ *e* $tam\{Y\} > 1$ **então**
    se $controle = idxCam$ **então**
      para $j \leftarrow 1$ **até** n **faça**
        Alinhar A com *gaps* até que $j = prox(cam[idxCam], Y)$
        $S[0] \leftarrow A(1)$, $S[1] \leftarrow B(prox(cam[idxCam], Y))$
        Complete A com *gaps*
      fim
      $cam[idxCam] \leftarrow prox(cam[idxCam], Y)$
      $cam[u] \leftarrow -1 \forall u > idxCam$
      se $\max\{Y\} \leq cam[idxCam]$ **então**
        $controle \leftarrow controleAux$
      fim
    fim
    senão
      para $j \leftarrow 1$ **até** n **faça**
        Alinhar A com *gaps* até que $j = cam[idxCam]$
        $S[0] \leftarrow A(1)$, $S[1] \leftarrow B(cam[idxCam])$
        Complete A com *gaps*
      fim
      $controleAux \leftarrow idxCam$
    fim
    $idxCam++$
  fim
  senão
    se $cam[idxCam] = -1$ **então**
      $cam[idxCam] \leftarrow min\{Y\}$
    fim
    para $j \leftarrow 1$ **até** n **faça**
      Alinhar A com *gaps* até que $j = cam[idxCam]$
      $S[0] \leftarrow A(1)$, $S[1] \leftarrow B(cam[idxCam])$
      Complete A com *gaps*
    fim
    $idxCam++$
  fim
fim
retorna S

Algoritmo 10: Escolha de um valor para K

```
   Entrada: Y, k
1  início
2      se max{Y} > cam[idxCam] e tam{Y} > 1 então
3          se controle = idxCam então
4              cam[u] ← −1∀u > idxCam
5              k ← prox(cam[idxCam], Y)
6              cam[idxCam] ← k
7              se max{Y} ≤ cam[idxCam] então
8                  controle ← controleAux
9              fim
10         fim
11         senão
12             k ← cam[idxCam]
13             controleAux ← idxCam
14         fim
15         idxCam + +
16     fim
17     senão
18         se cam[idxCam] = −1 então
19             cam[idxCam] ← min{Y}
20         fim
21         k ← cam[idxCam]
22         idxCam + +
23     fim
24 fim
25 retorna k
```

3.3 ANÁLISES DE COMPLEXIDADE DE TEMPO E ESPAÇO

O vetor cam possui tamanho fixo de $m+n+\log m$. O conjunto Y e os vetores $C1$ e $C2$ tem complexidade de espaço $O(n)$. Assumimos que o conjunto, os vetores e as matrizes estão em armazenamento comum e as *substrings* de A e B podem ser transferidas como argumentos, dando localizações iniciais e finais.

1. Análise do Algoritmo 3

   Os laços das linhas 5 e 8 fazem com que a instrução da linha 9 seja executada mn vezes. A matriz M e o vetor de saída estão em armazenamento comum e por isso não vamos

computar seu armazenamento nesse algoritmo. Assim o Algoritmo 3 tem complexidade de tempo $O(mn)$ e espaço $O(1)$.

2. Análise do Algoritmo 9

   A complexidade de tempo $O(n)$ é dada pelas instruções executadas por apenas um dos laços (linhas 4, 17 e 31). Portanto, o algoritmo possui complexidade de tempo $O(n)$. O espaço utilizado pelo algoritmo é através do conjunto Y e do vetor cam, que estão em armazenamento comum. Portanto, o algoritmo possui complexidade de tempo $O(n)$ e espaço $O(1)$.

3. Análise do Algoritmo 8

   Para sequências de tamanho $n = 0$ o laço da linha 3 do algoritmo é executado m vezes.

   Para sequências de $m = 1$, a instrução da linha 12 ou da linha 18 é executada n vezes e por fim, o Algoritmo 9 é chamado. A chamada para o Algoritmo 9 é limitada em tempo por n (veja a análise de tempo do Item 2). Sendo assim, para $m = 1$ a complexidade de tempo é limitada por $O(n)$.

   O algoritmo utiliza o espaço de Y na linha 12 e $O(1)$ pela chamada do Algoritmo 9 (veja a análise de espaço do Item 2).

   Desta forma, o Algoritmo 8 possui uma complexidade de tempo $O(m + n)$ (assumindo os dois casos) e de espaço $O(1)$.

4. Análise do Algoritmo 10

   O Algoritmo 10 possui uma complexidade de tempo $O(m+n)$ (linha 4) e de espaço $O(1)$.

5. Análise do Algoritmo 7

   Para sequências de tamanhos $n = 0$ ou $m = 1$, a complexidade é limitada pelo tempo em $O(m + n)$ (análise de tempo do Item 3).

   Para sequências de tamanhos m e n, duas chamadas para o Algoritmo 3 são realizadas. As chamadas são limitadas por $O(mn)$ pela análise de tempo do Item 1.

   Encontrar o valor de k é limitado pelo tempo em $O(n)$ mais $O(m + n)$ (veja a análise de tempo do Item 4).

   Resolver problemas mais simples requer duas chamadas para o Algoritmo 7. Essas chamadas serão limitadas pelo tempo por $O(mk)$ para a primeira chamada (linha 17) e $O(m(n-k))$ para a segunda chamada, totalizando $O(mn)$.

   Desta forma, teremos um tempo de $O(mn) + O(mn) + O(m + n) + O(m + n) + O(n)$.

   Conforme citado acima, o conjunto, os vetores e as matrizes estão em armazenamento comum e as *substrings* podem ser transferidas como argumentos. Então, durante a execução, as chamadas para os algoritmos usam armazenamento temporário que é $O(m +$

$n + \log m) + O(n)$ (soma do espaço utilizado pelo conjunto, pelos vetores e matrizes). Portanto, o Algoritmo requer espaço de memória proporcional a m e n.

Ou seja, o Algoritmo 7 tem complexidade de tempo $O(mn)$ e de espaço $O(m + n)$.

6. Análise do Algoritmo 6

   O Algoritmo 6 possui complexidade de tempo $q(mn)$ conforme a análise de tempo do Item 5, sendo q uma constante que representa o número de alinhamentos ótimos gerados. Como o espaço é através de armazenamento comum, a complexidade de espaço é linear em m e n (análise de espaço do Item 5).

   Desta forma, o Algoritmo de Hirschberg estendido é capaz de gerar mais de um alinhamento ótimo, se existir, em espaço linear no tamanho da entrada e tempo polinomial no tamanho total da entrada e da saída.

4 CONCLUSÃO

A biologia molecular está avançando a passos largos com a evolução tecnológica e uma grande quantidade de dados foi e está sendo gerada. Esses dados tendem a ser cada vez maiores e, por isso, super computadores são necessários para armazenar as informações obtidas através desses dados. O grande problema da análise desses dados é a quantidade de informações contidas nos genomas, tornando essencial a aplicação de procedimentos lógicos para auxiliar na interpretação destes dados. O grande desafio é transformar o genoma bruto em informação útil para a ciência.

O presente estudo permitiu o conhecimento dos métodos ótimos utilizados atualmente e a demonstração de uma nova abordagem para o alinhamento de sequências, onde o domínio desta técnica pode conduzir a busca de resultados ótimos em espaço linear.

Revisamos o Algoritmo de Needleman e Wunsch (1970) que, baseado em programação dinâmica, é capaz de gerar alinhamentos ótimos com complexidade de tempo e espaço quadrática. O grande gargalo desse método está na memória utilizada, conforme pode-se observar na análise de complexidade 2.5.1.1. Com uma complexidade de espaço linear, revisamos o algoritmo proposto por Hirschberg (1975) e uma adaptação, a qual através de programação dinâmica e de divisão e conquista, é capaz de gerar um alinhamento ótimo em espaço linear e tempo polinomial.

O problema identificado foi o espaço e a memória das máquinas quando surge a necessidade de gerar mais de um alinhamento ótimo. Desta forma, o Algoritmo de Hirschberg Estendido, proposto por nós, torna-se um bom método para conduzir a busca por alinhamentos ótimos. Baseado em programação dinâmica, divisão e conquista, e usando um processo de *backtracking*, ele é capaz de gerar mais de um alinhamento ótimo, se existir, em espaço linear no tamanho da entrada e tempo polinomial no tamanho total da entrada e da saída.

Observe na Tabela 5 a comparação dos métodos dispostos no trabalho.

Tabela 5 – Comparação dos métodos

	Needleman e Wunsch (1970)		Hirschberg (1975)		Hirschberg estendido	
Quantidade	$1A^{opt}$	qA^{opt}	$1A^{opt}$	qA^{opt}	$1A^{opt}$	qA^{opt}
Espaço	$O(mn)$	$O(mn)$	$O(m+n)$	X	$O(m+n)$	$O(m+n)$
Tempo	$O(mn)$	$O((mn)+q(m+n))$	$O(mn)$	X	$O(mn)$	$O(q(mn))$

Fonte: Autoria própria

4.1 TRABALHOS FUTUROS

Os resultados obtidos foram promissores onde estudos mais aprofundados destas técnicas podem proporcionar novas adaptações. Encorajamos investigações futuras que visem evitar

a geração de alinhamentos ótimos iguais e que tenham uma complexidade de tempo menor, visto que a quantidade de alinhamentos gerados (q) multiplica mn no Algoritmo de Hirschberg estendido e $m + n$ no algoritmo proposto por Needleman e Wunsch (1970).

REFERÊNCIAS

ABBOUD AMIR E WILLIAMS, V. V. e. W. O. Consequências do alinhamento mais rápido das sequências. In: **Colóquio Internacional sobre Autômatos, Idiomas e Programação**. [S.l.: s.n.], 2014. p. 39–51.

ALTSCHUL, S. F. *et al.* Gapped blast and psi-blast: a new generation of protein database search programs. **Nucleic acids research**, Oxford University Press, v. 25, n. 17, p. 3389–3402, 1997.

BALDI, P.; BRUNAK, S. **Bioinformatics: the machine learning approach**. [S.l.]: MIT press, 2001.

COHEN, J. Bioinformatics—an introduction for computer scientists. **ACM Computing Surveys (CSUR)**, ACM, v. 36, n. 2, p. 122–158, 2004.

DAYHOFF, M.; SCHWARTZ, R.; ORCUTT, B. 22 a model of evolutionary change in proteins. In: **Atlas of protein sequence and structure**. [S.l.]: National Biomedical Research Foundation Silver Spring, 1978. v. 5, p. 345–352.

HENIKOFF, S.; HENIKOFF, J. G. Amino acid substitution matrices from protein blocks. **Proceedings of the National Academy of Sciences**, National Acad Sciences, v. 89, n. 22, p. 10915–10919, 1992.

HIRSCHBERG, D. S. A linear space algorithm for computing maximal common subsequences. **Communications of the ACM**, ACM, v. 18, n. 6, p. 341–343, 1975.

LAU, F. An integrated approach to fast, sensitive, and cost-effective smith-waterman multiple sequence alignment. **Bioinformatics Module**, 2000.

MYERS, E. W.; MILLER, W. Optimal alignments in linear space. **Bioinformatics**, Oxford University Press, v. 4, n. 1, p. 11–17, 1988.

NEEDLEMAN, S. B.; WUNSCH, C. D. A general method applicable to the search for similarities in the amino acid sequence of two proteins. **Journal of molecular biology**, Elsevier, v. 48, n. 3, p. 443–453, 1970.

OKURA, V. K. *et al.* Bioinformática de projetos genoma de bactérias. [sn], 2002.

SANDES, E. F. d. O. Algoritmos paralelos exatos e otimizações para alinhamento de sequências biológicas longas em plataformas de alto desempenho. 2015.

SETUBAL, J. C.; MEIDANIS, J. **Introduction to computational molecular biology**. [S.l.]: PWS Pub., 1997.

WANG, L.; JIANG, T. On the complexity of multiple sequence alignment. **Journal of computational biology**, v. 1, n. 4, p. 337–348, 1994.