Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

MASA-Webaligner: Uma Plataforma Web para Execução e Visualização de Alinhamentos de Sequências de DNA

Bernardo C. Nascimento

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientadora
Prof.a Dr.a Alba C. M. A. de Melo

Brasília
2020

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

MASA-Webaligner: Uma Plataforma Web para Execução e Visualização de Alinhamentos de Sequências de DNA

Bernardo C. Nascimento

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof.a Dr.a Alba C. M. A. de Melo (Orientadora)
CIC/UnB

Prof.a Dr.a Maristela Terto de Holanda
CIC/UnB

Prof. Dr. Rodrigo Bonifácio de Almeida
CIC/UnB

Prof. Dr. Marcelo Grandi Mandelli
Coordenador do Bacharelado em Ciência da Computação

Brasília, 01 de setembro de 2020

Dedicatória

Dedico este trabalho à minha mãe, Marta Maria da Costa, que sempre me apoiou minhas decisões, me incentivou e lutou para que tivesse as melhores oportunidades. Sem ela, nada disso seria possível. Meu mais sincero obrigado.

Agradecimentos

Gostaria de agradecer à minha mãe pelo apoio nos últimos anos, por proporcionar todo suporte necessário para que eu pudesse tirar o máximo proveito da graduação.

Deixo também meu agradecimento à Prof. Dra. Alba C. M. A. de Melo por sua orientação, paciência e exigência durante todo o desenvolvimento deste trabalho. Seu profissionalismo e dedicação à educação e à computação são motivos de inspiração para mim.

À minha namorada, Clarissa e Palos Brito, por todo seu apoio. Ao longo dessa caminhada, foram diversos momentos de dificuldade e, felizmente, tive a sorte de ter essa mulher incrível ao meu lado para me dar forças e seguir em frente.

Agradeço também aos meus amigos pelos momentos de descontração e suporte, que me ajudaram a lidar com as dificuldades ao longo da graduação.

Por fim, a todos que participaram direta, ou indiretamente, da minha formação, o meu muito obrigado.

Resumo

Na Bioinformática, o alinhamento de sequências biológicas é uma operação muito importante, capaz de verificar a similaridade entre as sequências analisadas e gerar informações para determinar suas funções, estruturas e evolução. Existem várias ferramentas que podem ser utilizadas para realizar alinhamentos de sequências genéticas. Entre elas está a plataforma MASA, capaz de gerar alinhamentos globais, locais e semi-globais, em tempo razoável e complexidade linear de memória, além de fornecer um sistema para visualização dos resultados em forma gráfica e textual. Apesar do grande potencial da plataforma MASA, sua instalação e utilização são complexas para usuários leigos e, portanto, apresentam uma barreira para sua utilização. No presente trabalho de graduação foi projetado, implementado e avaliado o *MASA-Webaligner*, uma plataforma *web*, que unifica os *softwares* de alinhamento e visualização da plataforma MASA, abstraindo detalhes de instalação e utilização do sistema. A plataforma conta com sistemas de armazenamento das requisições, fila que permite que cada requisição de alinhamento possa utilizar todos os recursos computacionais disponíveis, possibilidade de aviso por *e-mail* após a conclusão do processamento de um alinhamento e visualização textual e gráfica dos resultados obtidos.

Palavras-chave: Bioinformática, Alinhamnto de Sequências, CUDAlign, MASA, Plataforma Web

Abstract

In Bioinformatics, biological sequence alignment is a very important operation, capable of checking similarities between the analyzed sequences and generating information to determine their functions, structures and evolution. There are several tools that can be used to perform genetic sequence alignments. Among them is the MASA plataform, capable of generating global, local and semi-global alignments, in reasonable time and linear memory complexity, while providing a graphical and textual visualization system for the results. Despite the great potential of the MASA plataform, its installation and use are complex to lay users and, therefore, present as a barrier for its use. In the present undergraduate work the MASA-Webaligner, a web plataform that unifies the alignment and visualization tools of the MASA plataform, abstracting the installation and utilization of the system, was designed, implemented and evaluated. The plataform has a storage system for the requests, a queue that allows that each request can use all the computational resources available, the possibility of e-mail warning after an alignment processing ending and graphic and textual visualization of the obtained results.

Keywords: Bioinformatics, Sequences Alignment, CUDAlign, MASA, Web Plataform

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

É cada vez mais comum o aparecimento de áreas de estudo interdisciplinares na ciência. A Bioinformática é um exemplo que envolve as disciplinas de Biologia, Computação e Estatística. O campo da Bioinformática utiliza a computação para análise, modelagem e simulação, utilizando dados biológicos, como sequências genéticas (DNA, RNA e proteínas), e auxiliando biólogos na melhor compreensão de organismos [7]. No entanto, a área vem lidando com problemas cada vez mais complexos, exigindo algoritmos cada vez mais eficientes.

Um exemplo de análise na Bioinformática é o alinhamento entre duas sequências biológicas, utilizando algoritmos específicos. Alinhamentos de sequências genéticas permitem a biólogos verificar a similaridade entre as sequências analisadas e determinar suas funções, estruturas e informações evolutivas, e, para isso, é importante obter o melhor alinhamento possível, chamado de alinhamento ótimo [7].

De forma a obter alinhamentos ótimos entre duas sequências biológicas, vários algoritmos já foram desenvolvidos [7]. No entanto, algoritmos que geram estes alinhamentos têm um alto custo computacional, pois possuem complexidade temporal quadrática. Devido à sua complexidade, o uso desses algoritmos para o alinhamento de sequências longas é geralmente inviabilizado. Os algoritmos de Needleman-Wunsh [8] e Smith-Waterman [9] são exemplos de algoritmos que geram alinhamentos ótimos entre pares de sequências biológicas.

Para realizar o alinhamento de sequências muito longas em tempo razoável, foram desenvolvidos algoritmos heurísticos como o FASTA [10] e BLAST [11]. Apesar destes algoritmos serem capazes de alinhar sequências longas, eles não garantem a geração de um alinhamento ótimo, o que é um ponto negativo.

Visando resolver o problema do alinhamento ótimo, em tempo razoável, Edans Sandes [1] desenvolveu o CUDAlign. A ideia é utilizar a arquitetura CUDA (Compute Unified Device Architecture) para obter paralelismo no processamento dos alinhamentos de

sequências genéticas, além de utilizar variantes dos algoritmos que retornam alinhamentos ótimos, de forma que sua complexidade de memória, antes quadrática, seja linear.

Posteriormente, o CUDAlign foi incorporado a um projeto mais amplo, o MASA [1], que propõe facilitar o desenvolvimento de soluções de alinhamento para outras plataformas, além da arquitetura CUDA. Como exemplo de solução, podemos citar o MASA-OpenMP (*Open Multi-Processing*), que utiliza múltiplos processadores para obter paralelismo no processamento de alinhamento de sequências.

Embora o MASA-CUDAlign apresente-se como uma poderosa solução para o alinhamento de sequências genéticas longas [12], sua instalação e utilização são feitas por meio do terminal Linux, o que não é ideal para pessoas leigas. Ainda, a visualização gráfica dos resultados necessita de outro programa (MASA-Viewer) e a visualização textual não possui nenhum tipo de ajuste ou realce dos dados.

O objetivo do presente trabalho de graduação é propor, implementar e avaliar o MASA-Webaligner, uma plataforma *web*, que unifique o MASA-CUDAlign, MASA-OpenMP e MASA-Viewer em um único sistema. A plataforma proposta visa abstrair os detalhes de instalação e utilização dos *softwares* citados, facilitando sua utilização pelos biólogos, além de garantir que múltiplos usuários utilizem a plataforma de forma concorrente, sem prejuízo de performance. A plataforma irá contar com um sistema de armazenamento das informações dos alinhamentos requisitados e um sistema de fila para que cada alinhamento processado utilize todos os recursos de *hardware* disponíveis. Além disso, os resultados obtidos serão utilizados para construção de um gráfico e saída textual ajustável e com realce das áreas de maior diferença entre as sequências.

O presente documento está organizado como se segue. No Capítulo 2 são descritos os conceitos básicos sobre comparações de sequências biológicas, alinhamento de pares e os principais algoritmos utilizados para obter alinhamentos ótimos. O Capítulo 3 descreve o CUDAlign, suas versões e otimizações, bem como a arquitetura MASA. No Capítulo 4 são apresentadas algumas das principais ferramentas disponíveis para alinhamento de sequências biológicas, além da descrição do *software* de visualização MASA-Viewer. No Capítulo 5 é apresentada a plataforma MASA-Webaligner, sua arquitetura e funcionamento. O Capítulo 6 apresenta os testes realizados e resultados obtidos, e uma avaliação da plataforma MASA-Webaligner. Por fim, no Capítulo 7, são descritas as conclusões deste trabalho e apresentadas sugestões para trabalhos futuros.

Capítulo 2

Comparação de Sequências Biológicas

Na Biologia, a comparação entre duas ou mais sequências biológicas (DNA, RNA ou proteínas) tem por objetivo determinar a similaridade entre elas e a sua função. Como resultado, são produzidos um *score* de similaridade e um alinhamento. Embora a análise possa ser feita de maneira experimental - e essa ser a forma mais confiável de fazê-la -, o mais comum é utilizar-se de métodos computacionais, pois estes são mais rápidos e fáceis de serem executados [13].

Apesar da evolução do *hardware* e dos métodos computacionais, a comparação de sequências biológicas ainda apresenta um grande desafio para computação, seja pelo tempo de execução exigido, seja pelo consumo excessivo de memória. Em [12], os autores mostram que, em alguns casos, a execução de algoritmos de alinhamento ótimo pode levar até 5 dias para ser concluída e, além disso, até a publicação do mesmo artigo, a literatura mostrava alinhamentos ótimos de até 60 MBP (*Millions of Base Pairs*), devido ao altíssimo tempo de execução.

O software CUDAlign implementa uma nova estratégia para realizar o alinhamento ótimo entre sequências biológicas muito longas, tendo executado comparações de cromossomos completos de até 249 MBP x 228 MBP, em tempo razoável, para utilização de pesquisadores da área [12]. O presente capítulo tem por objetivo descrever os conceitos básicos a respeito do alinhamento entre duas sequências biológicas, de forma a apresentar a base necessária para compreensão de como a ferramenta de visualização, aqui desenvolvida, funciona em combinação com o CUDAlign.

2.1 Conceitos Básicos

Quando se realiza a comparação entre duas sequências de nucleotídeos (DNA ou RNA) ou aminoácidos (proteínas), as sequências são consideradas como *strings* e utiliza-se técnicas de reconhecimento aproximado de padrões e *strings* [13].

Ao realizar a comparação, temos 3 casos básicos de combinações, quais sejam: *match*, *mismatch* e *gap* (este último pode ser dividido entre inserções e deleções). O caso do *match* ocorre quando dois caracteres das sequências x e y, em uma dada posição x_i e y_j, são iguais. O *mismatch* se dá quando dois caracteres, x_i e y_j, são diferentes. Os modelos mais simples de comparação utilizam apenas as combinações de *match* e *mismatch*, porém modelos mais complexos fazem uso do *gap*, ou seja, a inserção de uma lacuna em uma das sequências. O *gap* pode significar que um dado nucleotídeo (DNA, RNA) ou aminoácido (proteínas) foi inserido ou deletado ao longo do processo evolutivo [13].

Ao final da comparação entre os pares, a cada combinação encontrada - *match*, *mismatch* e *gap* - são atribuídos valores, é o chamado *score*. O *score* é de extrema importância ao realizar os cálculos de similaridade entre duas sequências. Existem diversos estudos sobre modelos de *score* para comparação entre sequências de proteínas tais como PAM200, PAM250, BLOSUM62, dentre outros. De maneira geral, os esquemas de *score* pontuam as combinações que geram *match* e penalizam *mismatch*.

A penalidade sobre os *gaps* também tem bastante variedade. Existem quatro formas de penalizar os *gaps* [14]: constante, *affine*, convexa e arbitrária. O constante é o mais simples, onde o primeiro elemento do *gap* tem peso W_g e os demais tem peso zero. O modelo *affine* consiste em penalizar o primeiro elemento do *gap* com o valor W_g e cada elemento subsequente com o valor W_s, onde $W_s < W_g$ [14], gerando a função de peso $W_{gap} = W_g + W_s * q$. No modelo convexo, a ideia é que o primeiro elemento contribui com um determinado peso e cada elemento adicional contribui menos do que o anterior, gerando, como exemplo, a função de peso $W_{gap} = W_g + log * q$. Por fim, tem-se o modelo arbitrário que é o mais geral. Nele, a função peso dos *gaps* é uma função arbitrária. Os modelos constante, *affine* e convexo são sub casos do modelo arbitrário.

A matriz de programação dinâmica (DP) é a forma como os principais algoritmos de comparação de sequências (Seção 2.2) calculam o alinhamento ótimo. Existem algumas formas para processar a matriz de DP, mas, no presente trabalho, apenas a técnica de *wavefront* será abordada. O *wavefront* consiste em processar as células da matriz de DP em ondas, uma anti-diagonal por vez. No caso da comparação de sequências biológicas, há um padrão para o cálculo de cada célula da matriz de DP, onde a célula (i, j) depende das células $(i-1, j)$, $(i, j-1)$ e $(i-1, j-1)$. Dessa forma, as células de uma anti-diagonal não possuem dependências entre si. A Figura 2.1 exemplifica o processamento da matriz de DP, utilizando o método do *wavefront*.

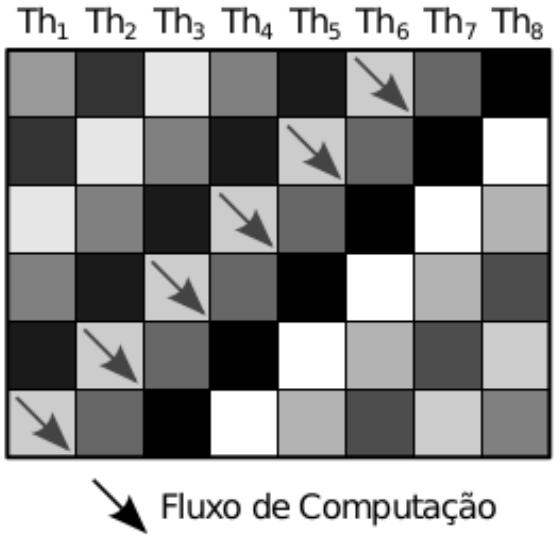


Figura 2.1: Exemplo de processamento pelo método *wavefront* [1].

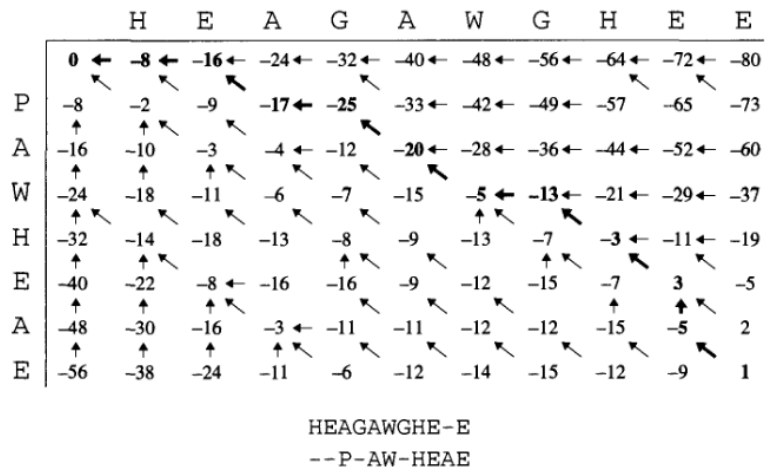
Com a noção de como os alinhamentos e os cálculos de similaridade são realizados, podemos definir o que são alinhamentos de pares, os tipos de alinhamentos existentes e os principais algoritmos para encontrar os alinhamentos ótimos entre duas sequências biológicas.

2.1.1 Alinhamento de Pares

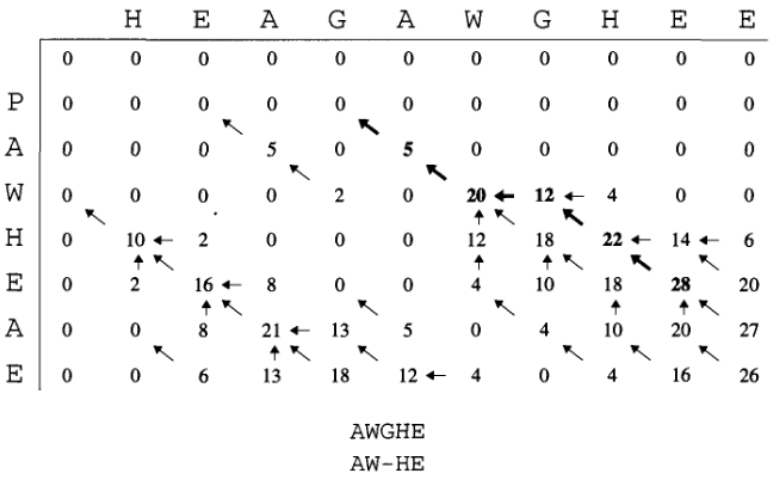
Quando pesquisam similaridades entre sequências genéticas, pesquisadores utilizam o procedimento de alinhamento de pares, que consiste em comparar duas sequências distintas para encontrar caracteres individuais ou padrões que apareçam na mesma ordem [7], que possam indicar relações de funcionalidades, de estruturas ou evolutivas entre elas [15]. De maneira geral, podem ser separadas em duas categorias, quais sejam: alinhamento global e alinhamento local.

O alinhamento global, como o próprio nome indica, considera a totalidade dos caracteres de sequências sendo analisados até o final destas. A ideia é encontrar a maior quantidade possível de nucleotídeos ou aminoácidos idênticos [7]. O alinhamento global é útil em casos onde deseja-se deduzir a história evolutiva examinando semelhanças e diferenças entre sequências biológicas na mesma família.

Em contraponto ao alinhamento global, o local começa e termina em regiões de grande similaridade entre as sequências biológicas. Este tipo de alinhamento possui um grande foco em áreas que são muito semelhantes ao invés de considerar a similaridade entre as sequências como um todo. O alinhamento local tem por objetivo encontrar padrões de



(a) Exemplo de alinhamento global [13]



(b) Exemplo de alinhamento local [13]

nucleotídeos, sequências de DNA, ou padrões de aminoácidos que foram conservados ao longo da evolução [7]. Esse tipo de alinhamento é especialmente importante no estudo das proteínas [14]. As Figuras 2.2a e 2.2b apresentam um exemplo de alinhamento global e local entre as sequências S_0 = [P,A,W,H,E,A,E] e S_1 = [H,E,A,G,A,W,G,H,E,E].

2.2 Algoritmos para Comparação de Sequências

Como dito na Seção 2.1, a forma utilizada para que todas as combinações de caracteres seja considerada é a Matriz de Pontos. Ao se comparar duas sequências, considera-se os alfabetos Σ = {A, T, G, C} para sequências de DNA, Σ = {A, T, G, U} para sequências de RNA ou Σ = {A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y} para proteínas e um caractere adicional para representação dos *gaps*.

Existem formas diferentes de alinhamento e *score* e, portanto, existem também distintos algoritmos que levam essas diferenças em consideração. Nesta seção serão apresentados quatro algoritmos que serviram de base para o desenvolvimento do CUDAlign e levam em consideração o alinhamento global (Needleman-Wunsch), alinhamento local (Smith-Waterman) e modelo de *score Affine Gap* (Gotoh e Myers-Miller).

2.2.1 Algoritmo de Needleman-Wunsch (NW)

Em 1970, Needleman e Wunsch propuseram um algoritmo para o alinhamento global de duas sequências biológicas [8]. Apesar de gerar o alinhamento ótimo, a versão inicial tinha complexidade cúbica ($O(n^3)$), posteriormente, foi alterada por Gotoh para uma versão que utiliza matrizes de programação dinâmica, com complexidade quadrática ($O(nm)$), onde m e n são os tamanhos das *strings* S_1 e S_2. [13]. A ideia do algoritmo é criar uma matriz de similaridade, A, com as *strings* S_1 e S_2.

Para realizar o cálculo do *score* em cada célula da matriz A, o algoritmo irá determinar o maior valor entre três possibilidades: (i) com base na célula $A_{i-1,j-1}$, acrescentar o valor obtido pelos caracteres $S_1[i]$ e $S_2[j]$ - que pode ser um *match* ou *mismatch*; (ii) com base na célula $A_{i-1,j}$, subtrair o valor de um *gap*; e (iii) com base na célula $A_{i,j-1}$, subtrair o valor de um *gap*. A função de *score* é, então, ser representada de acordo com a Equação 2.1.

$$A_{i,j} = max \begin{cases} A_{i-1,j-1} + s(i,j), \\ A_{i-1,j} - g, \\ A_{i,j-1} - g. \end{cases} \tag{2.1}$$

Como a função de *score* precisa utilizar o valor de uma célula previamente calculada, é necessário determinar valores iniciais para que o algoritmo possa ser executado. Dessa maneira, adiciona-se uma linha e coluna no inicio da matriz - tornando seu tamanho $(n+1)\times(m+1)$ -, onde cada célula adicionada terá o valor $-i*g$, para a linha, e $-j*g$ para a coluna. Considerando as *strings* $S_0 = \{A,T,G,C,A,T,T,G\}$ e $S_1 = \{T,G,C,A,A,T,G,G,G\}$ e os valores +1 para *match*, -1 para *mismatch* e -2 para *gaps*, o algoritmo NW produz a matriz A representa na Tabela 2.1 e o alinhamento 2.2.

Após o cálculo de cada célula $A_{i,j}$, é armazenada uma referência para a célula anterior que foi utilizada para produzir a atual. Feito o preenchimento de todas as células da matriz, inicia-se o processo de *traceback*. Este processo consiste em determinar o alinhamento utilizando a referência armazenada em cada célula da matriz. Começando pela célula $A_{n+1,m+1}$, retorna-se até a posição inicial da matriz, gerando o alinhamento de trás para frente, exemplificado na Tabela 2.2.

Tabela 2.1: Matriz de alinhamento global utilizando Needleman-Wunsch.

*	*	A	T	G	C	A	T	T	G
*	0	-2	-4	-6	-8	-10	-12	-14	-16
T	-2	-1	-1	-3	-5	-7	-9	-11	-13
G	-4	-3	-2	0	-2	-4	-6	-8	-10
C	-6	-5	-4	-2	1	-1	-3	-5	-7
A	-8	-5	-6	-4	-2	2	0	-2	-4
A	-10	-7	-6	-5	-3	0	-1	-1	-3
T	-12	-9	-6	-7	-5	-2	1	0	-2
G	-14	-11	-8	-5	-7	-4	-1	0	1
G	-16	-13	-10	-7	-6	-6	-3	-2	1
G	-18	-15	-12	-9	-8	-7	-5	-4	-1

Tabela 2.2: Alinhamento global resultante de Needleman-Wunsch, com *score* -1.

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

2.2.2 Algoritmo de Smith-Waterman (SW)

Diferente do algoritmo de Needleman-Wunsch, o algoritmo de Smith-Waterman [9], proposto em 1981, tem por objetivo encontrar o melhor alinhamento local ótimo [13]. Para retornar tal alinhamento, foram feitas alterações no algoritmo de Needleman-Wunsch. As modificações não alteraram a complexidade, que se mantém quadrática ($O(nm)$).

A principal alteração sobre o algoritmo de Needleman-Wunsch se dá na equação de cálculo das células. Como deseja-se encontrar apenas os trechos que são muito semelhantes, a equação deve considerar que, caso o valor se torne negativo, seja substituído por zero, reiniciando o alinhamento. Temos, então, a Equação 2.2 para o cálculo das células da matriz de programação dinâmica A de Smith-Waterman:

$$A_{i,j} = max \begin{cases} 0, \\ A_{i-1,j-1} + s(i,j), \\ A_{i-1,j} - g, \\ A_{i,j-1} - g. \end{cases} \tag{2.2}$$

Assim como em Needleman-Wunsch (Seção 2.2.1), uma referência para célula anterior utilizada é armazenada na atual, com exceção de quando o valor zero é obtido. A matriz de similaridade também tem sua primeira linha e coluna alteradas para os valores 0 - ao invés de utilizar $-i * g$ e $-j * g$. Após a construção da matriz de similaridade, tem-se o maior valor da matriz, que é o *score* do alinhamento local ótimo. Ao encontrá-lo, realiza-se o *traceback* até que se encontre o valor zero. O alinhamento local é, então,

retornado. A Tabela 2.3 apresenta um exemplo da matriz de programação dinâmica A de Smith-Waterman, utilizando as mesmas *strings* e *scores* da Seção 2.2.1 e a Tabela 2.4 apresenta o alinhamento local obtido.

Tabela 2.3: Matriz de alinhamento local utilizando Smith-Waterman.

*	*	A	T	G	C	A	T	T	G
*	0	0	0	0	0	0	0	0	0
T	0	0	1	0	0	0	1	1	0
G	0	0	0	2	0	0	0	0	2
C	0	0	0	0	3	1	0	0	0
A	0	1	0	0	1	4	2	0	0
A	0	1	0	0	0	2	3	1	0
T	0	0	2	0	0	0	3	4	2
G	0	0	0	3	1	0	1	2	5
G	0	0	0	1	2	0	0	0	3
G	0	0	0	1	0	1	0	0	1

Tabela 2.4: Alinhamento local resultante de Smith-Waterman com *score* 5.

T	G	C	A	T	T	G
T	G	C	A	A	T	G

2.2.3 Algoritmo de Gotoh

O algoritmo de Gotoh [16], proposto em 1982, diferente dos anteriores, possibilita a implementação do modelo *affine gap*. Como descrito na Seção 2.1, este modelo penaliza o primeiro *gap* com um valor W_g e *gaps* os subsequentes com um valor W_s, onde $W_s < W_g$. Temos então a Equação 2.3 para calcular o valor da penalização, que será aplicada no alinhamento ao encontrar um *gap*.

$$\lambda(k) = -W_g - (k-1) * W_s, \text{ onde k é o número de } gaps \text{ consecutivos} \tag{2.3}$$

A algoritmo de Gotoh possui a mesma complexidade de tempo ($O(nm)$) que os algoritmos NW e SW, porém gasta o triplo de tempo e espaço, pois torna-se necessário manter múltiplos valores para cada par de caracteres [13]. Tem-se agora três matrizes para calcular o *score* de cada célula: $A_{i,j}$, $E_{i,j}$ e $F_{i,j}$.

As matrizes $E_{i,j}$ e $F_{i,j}$ são criadas para determinar se deve ser criado um novo *gap* ou estendido um já existente. Além disso, é necessário fazer uma nova alteração na função de cálculo da matriz A para considerar as novas matrizes. Temos então as Equações 2.4 2.5 2.6.

$$A_{i,j} = max \begin{cases} A_{i-1,j-1} + s(i,j) \\ A_{i-1,j} - E_{i,j} \\ A_{i,j-1} - F_{i,j} \end{cases} \tag{2.4}$$

$$E_{i,j} = max \begin{cases} E_{i,j-1} - W_s \\ A_{i,j-1} - W_g \end{cases} \tag{2.5}$$

$$F_{i,j} = max \begin{cases} F_{i-1,j} - W_s \\ A_{i-1,j} - W_g \end{cases} \tag{2.6}$$

De maneira semelhante aos algoritmos anteriores, após calcular toda a matriz de semelhança, o algoritmo executa o *traceback* para retornar o alinhamento ótimo.

2.2.4 Algoritmo de Myers-Miller (MM)

O algoritmo de Myers-Miller [2], foi proposto em 1988 e tem por objetivo obter o alinhamento ótimo com complexidade de espaço linear. É fácil perceber a motivação do algoritmo. Suponha a comparação entre duas sequências de DNA com 10^6 caracteres. Ao gerar a matriz de similaridade, temos um total (10^6+1)x(10^6+1) células, totalizando aproximadamente 4 Terabytes de memória, caso sejam utilizados 4 bytes por célula. Considerando, por exemplo, que o genoma humano contém aproximadamente 3*10^9 pares de bases [17], tornam-se inviáveis algoritmos de análise com complexidade espacial quadrática.

Assim, utilizando o algoritmo de Gotoh (Seção 2.2.3) e o algoritmo de comparação de *strings*, de complexidade espacial linear proposto por Hischberg, o algoritmo de Myers-Miller é capaz de executar o modelo *affine gap* para sequências biológicas maiores.

O algoritmo funciona da seguinte maneira. Inicialmente, suponha as *strings* S_0 = [T,G,C,A,A,T,G,G,G] e S_1 = [A,T,G,C,A,T,T,G] e a Equação 2.1 do algoritmo NW (Seção 2.2.1). Temos então a matriz A de similaridade. Seleciona-se a linha central da matriz A, tal que $\frac{m}{2}$. Após selecionar a linha central, incia-se o processamento da matriz A da seguinte forma: a linha 1 é obtida a partir da equação de recorrência e os valores da linha 0 são descartados. Com a linha 1, obtém-se a linha 2 e descarta-se a linha 1. O processo se repete até que se calcule a linha $\frac{m}{2}$. Feito isso, realiza-se o mesmo procedimento, com o reverso das *strings*, começando na linha m e indo até a linha $\frac{m}{2}$. Com os dois possíveis *scores* para a linha central, somam-se os resultados obtidos em cada posição da linha e, o maior resultado, é a coordenada pertencente ao alinhamento ótimo. As Figuras 2.3b e 2.3a, e a Tabela 2.5 exemplificam a execução do algoritmo.

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

(a) Execução da linha 0 até a linha $\frac{m}{2}$

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

(b) Execução da linha m até a linha $\frac{m}{2}$

Tabela 2.5: Cálculo da coordenada pertencente ao alinhamento global. A células em azul representam a coordenada obtida após a soma dos valores. Para o exemplo, a coordenada final é (4, 5).

-5	-6	-4	-1	2	0	-2	-4
-9	-6	-5	-2	-1	-3	-5	-5
-14	-12	-9	-3	1	-3	-7	-9

O procedimento descrito, irá dividir a matriz em sua linha central e obter as coordenadas do alinhamento ótimo de maneira recursiva, até que se chegue no caso trivial de apenas um ou zero caracteres, a Figura 2.4 exemplifica as recursões. É a estratégia de "Dividir para Conquistar". Como os cálculos de cada linha só levam em consideração a linha imediatamente anterior, a complexidade de espaço do algoritmo é linear ($O(n+m)$).

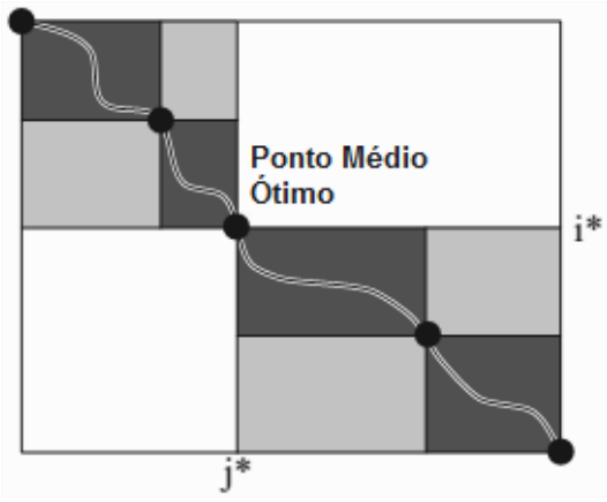


Figura 2.4: Exemplo da execução do algoritmo de Myers-Miller [2].

Capítulo 3

Ferramentas CUDAlign e MASA

O CUDAlign foi desenvolvido para encontrar o alinhamento ótimo de sequências longas em tempo razoável. O CUDAlign apresenta cinco versões desenvolvidas por Sandes em [18] e [1]. Cada versão apresenta otimizações em relação à anterior. Tais otimizações contemplam paralelismo do processamento, particionamento da matriz de DP, *Block Pruning*, múltiplas GPUs e *Incremental Speculative Traceback*. O objetivo do presente capítulo é descrever em detalhes as ferramentas CUDAlign e MASA, que serão utilizadas neste trabalho de graduação. Em particular, as estratégias desenvolvidas no CUDAlign contribuem para a compreensão de como é gerado o arquivo de saída, que será utilizado para visualização gráfica dos alinhamentos de sequências.

A Seção 3.1 apresenta a primeira versão do CUDAlign e o paralelismo interno e externo. A Seção 3.2 apresenta a segunda versão do CUDAlign e as otimizações de *matching* baseado em objetivo, execução ortogonal e divisão balanceada. A Seção 3.3 apresenta melhorias à segunda versão, com a otimização *block pruning*. A Seção 3.4 apresenta a terceira versão do CUDAlign e sua nova arquitetura Multi-GPU. A Seção 3.5 apresenta a última versão do CUDAlign e a otimização *Incremental Speculative Traceback*. Por fim, a Seção 3.6 apresenta a arquitetura MASA.

3.1 CUDAlign 1.0

O CUDAlign busca encontrar alinhamentos ótimos de sequências longas, em tempo razoável, utilizando o algoritmo de Smith-Waterman (Seção 2.2.2) com uso do modelo *affine gap* (Seção 2.1).

Para alcançar os objetivos, o autor em [18] propõe o uso de paralelismo externo e interno. No primeiro, as células da matriz de programação dinâmica (DP) são divididas em blocos de R linhas e C colunas, o que resulta em um *grid* G $= \frac{m}{R} \times \frac{n}{C}$, onde n e m são os tamanhos das strings S_0 e S_1.

Os tamanhos de R e C são definidos de acordo com o total B de blocos concorrentes e o número T de *threads* por bloco. Os valores de B e T são definidos de acordo com a GPU que será utilizada para o processamento, e estes serão usados nas chamadas de *kernel*. Dessa maneira, R e C são definidos da seguinte forma:

$$R = \alpha * T, \text{ onde } \alpha \text{ é o número de linhas que cada thread processa} \tag{3.1}$$

$$C = \frac{n}{B} \tag{3.2}$$

Para executar o processamento, o blocos são organizados em diagonais externas, D_k = $\{G_{ij} \mid i + j = k\}$, onde $k = \{0, 1, 2, ...\}$. O total de diagonais externas é dado por |D| = B + $\frac{m}{\alpha * T}$ - 1 e o total de blocos nas diagonais varia de 1 até B. Apenas uma diagonal é processada por vez e, ao final do processamento de cada uma, há sincronização, devido ao retorno de controle para a CPU. O processamento se repete até que todos os blocos da matriz de DP sejam processados. A Figura 3.1 mostra como é feito o processamento das diagonais externas.

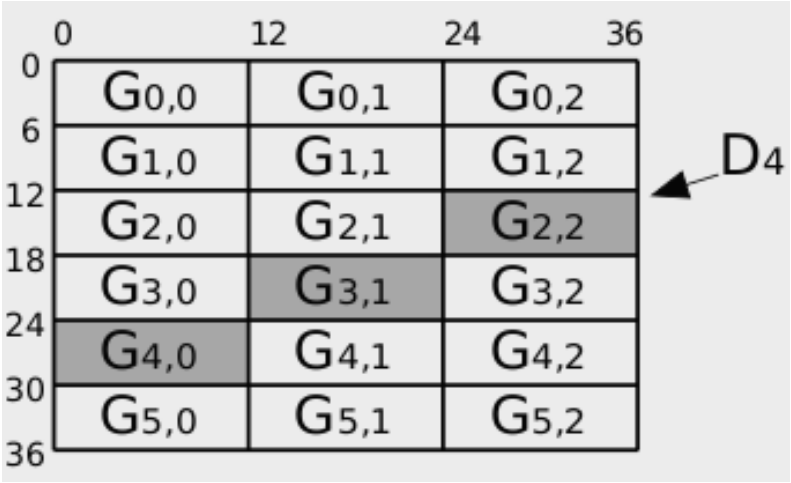


Figura 3.1: Exemplo de organização das D_k diagonais externas, formando o *grid* G. Em destaque a diagonal D_4 sendo processada [1].

No paralelismo interno, em cada bloco B, as T *threads* cooperam de forma a processar todas as $R \times C$ células deste. De maneira semelhante ao paralelismo externo, as células são processadas em diagonais internas, d_k = $\{(i, j) \lfloor \frac{i}{\alpha} \rfloor + j = k\}$, onde $k = \{0, 1, 2, ...\}$. Cada *thread* T_k processa $\alpha * k$ até $\alpha * k - 1$ linhas, indo da esquerda para a direita. As *threads* sempre processam a mesma diagonal d_k, dessa maneira, T_i processa a coluna j, ao mesmo tempo que T_{i-1} processa a coluna $j - 1$. Da mesma maneira que ocorre no paralelismo externo, ao final do processamento das diagonais internas, há uma sincronização das *threads* realizada pela diretiva ___syncthreads(), definida pela

arquitetura CUDA. A Figura 3.2 mostra como é feito o processamento das diagonais internas.

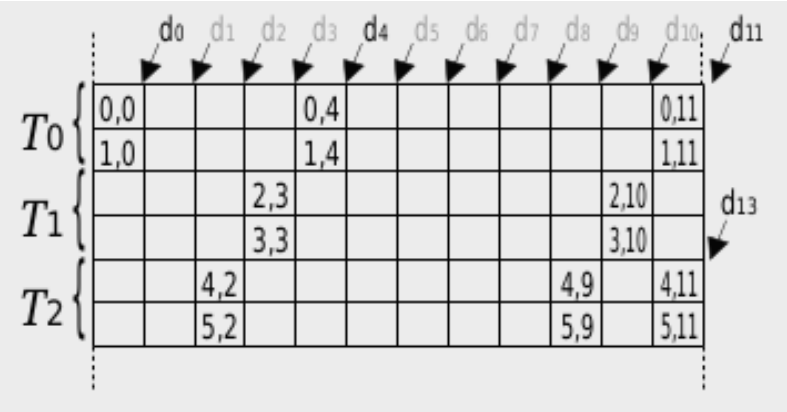


Figura 3.2: Exemplo de processamento de diagonais internas (bloco). São 3 *threads* processando 2 linhas cada. As diagonais internas d_0, d_4, d_{11} e d_{13} estão em destaque [1].

Para aumentar o paralelismo, algumas otimizações foram propostas no CUDAlign 1.0. Com o uso de blocos retangulares ($R \times C$), o paralelismo interno só é máximo entre as diagonais internas d_2 e d_{k-2}. Nas diagonais d_0, d_1, d_{k-1} e d_k, o paralelismo diminui com *threads* ociosas, como pode ser visto na Figura 3.2. Assim, foi implementado o mecanismo de delegação de células e paralelismo em paralelogramo.

Na delegação de células, as *threads* computam as células até a diagonal d_{k-2} (maior paralelismo) e finalizam o processamento, deixando o restante das células para serem processadas no próximo bloco. Dessa maneira, não tem-se mais blocos retangulares, e sim blocos paralelogramos, como mostrado na Figura 3.3.

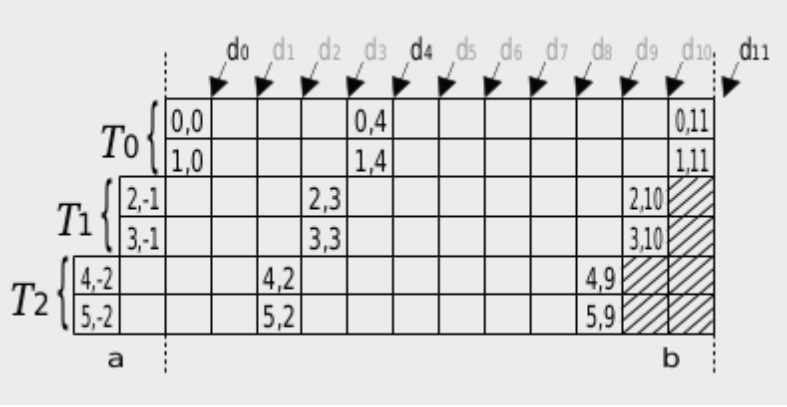


Figura 3.3: Exemplo de bloco paralelogramo. As células hachuradas mostram as células pendentes, que serão enviadas para processamento no próximo bloco. As células com j negativo indicam células que foram delegadas por um bloco anterior [1].

Apesar da delegação de células prover alto paralelismo, os blocos agora possuem dependência entre si, o que não pode acontecer já que o processamento feito pela GPU pode ocorrer em qualquer ordem. Para evitar isso, é necessário que o controle seja dado à CPU, forçando um ponto de sincronização global. Assim, criou-se duas fases de processamento (cada qual em seu *kernel*): curta e longa.

Na fase curta, o objetivo é terminar o processamento das células pendentes. A GPU processa as $T-1$ primeiras diagonais internas do bloco, a CPU assume e força a sincronização, eliminando a dependência.

Na fase longa, a GPU termina de processar as $\frac{n}{B}-(T-1)$ diagonais restantes. A fase é chamada de "longa", pois o número de colunas C de cada bloco tende a ser muito maior que o número de *threads*, resultando em um tempo maior de execução, quando comparada com a fase curta. As Figuras 3.4 e 3.5 mostram a dependência entre os blocos e as fases de processamento.

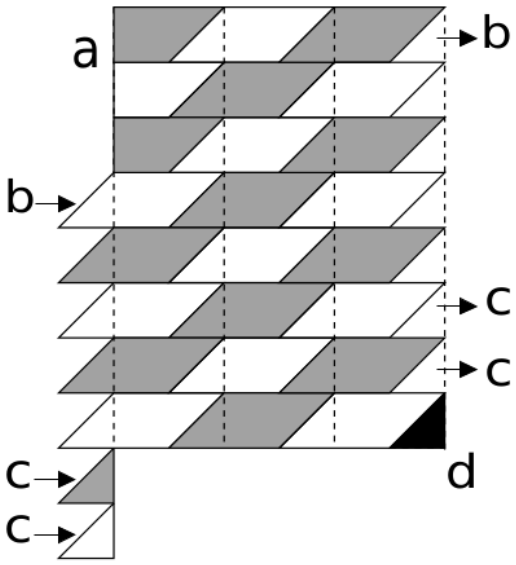


Figura 3.4: Dependência entre os blocos. Se o bloco 2 for processado antes do 3, os valores da área cinza serão indeterminados [1].

3.2 CUDAlign 2.0

O CUDAlign 2.0 recupera o alinhamento ótimo, executando a fase de *traceback* (Seção 2.2.1). Para tanto, adotou-se uma abordagem "dividir para conquistar", onde as coordenadas do alinhamento ótimo são obtidas iterativamente, até que se obtenha o alinhamento

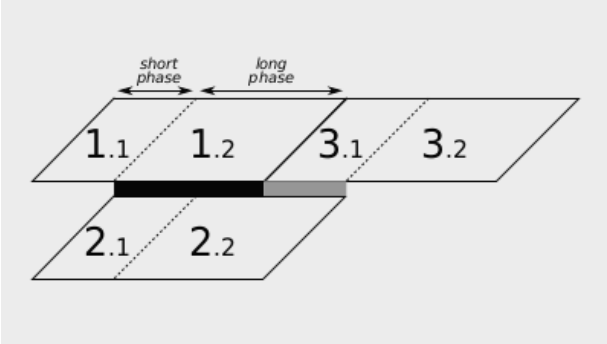


Figura 3.5: Resolução da dependência por meio das fases curta e longa. Ao processar a diagonal com os blocos 3 e 2, a fase curta irá processar 3.1 antes que a fase longa processe 2.2, eliminando a dependência [1].

completo. Dessa forma, foram desenvolvidas três estratégias - *matching* baseado em objetivo, execução ortogonal e divisão balanceada -, divididas em seis estágios.

3.2.1 Estágio 1 - Obtenção do *Score* Ótimo

O Estágio 1 executa o mesmo algoritmo básico do CUDAlign (Seção 3.1), com uma alteração: algumas linhas são salvas em disco para *matching* e *checkpoint*, permitindo que a execução seja retomada em caso de interrupção. Estas linhas são denominadas "linhas especiais" e são obtidas a partir do final do processamento de um bloco e, portanto, somente as linhas que tem índice múltiplo da altura de um bloco são candidatas.

As linhas especiais são salvas em disco, em uma área especial chamada *Special Rows Area* (SRA). O tamanho da SRA é constante, e definido em tempo de execução. Cada linha possui $8n$ bytes e a SRA pode armazenar um máximo de $\lceil \frac{8*m*n}{\alpha*T*|SRA|} \rceil$ linhas, e armazena pelo menos uma linha especial.

3.2.2 Estágio 2 - *Traceback* Parcial

No Estágio 2, é realizado o calculo da equação de recorrência do alinhamento de maneira modificada, levando em conta o alinhamento semi-global [1]. Nessa fase, serão encontradas as coordenadas do alinhamento ótimo que cruzam as linhas especiais, além da coordenada inicial do alinhamento ótimo.

Para realizar os cálculos, duas otimizações foram propostas: *matching* baseado em objetivo e execução ortogonal.

***Matching* baseado em objetivo**: o procedimento de *matching* realizado no Estágio 2 difere do procedimento proposto de Myers-Miller (Seção 2.2.4), pois já se conhece o *score* máximo (chamado de "*score* alvo"). Dessa forma, o algoritmo irá procurar pelo

score alvo e, assim que encontrá-lo, encerra o procedimento de *matching*, iniciando uma nova iteração buscando o próximo *score* alvo relativo à próxima linha especial. Ainda, a busca é realizada no sentindo inverso ao processamento do Estágio 1 e de maneira ortogonal.

Execução ortogonal: Para haver ganho de desempenho, o algoritmo não deve realizar a busca das coordenadas em sentindo horizontal - tal qual o processamento do estágio 1 faz. Para tornar a busca mais eficiente, as *threads* realizam a busca na direção vertical, no sentido ortogonal ao Estágio 1. Assim, a área da busca é diminuída, pois a computação para ao se encontrar o *score* alvo. A Figura 3.6 mostra a diferença entre a execução original de Myers-Miller e a execução ortogonal.

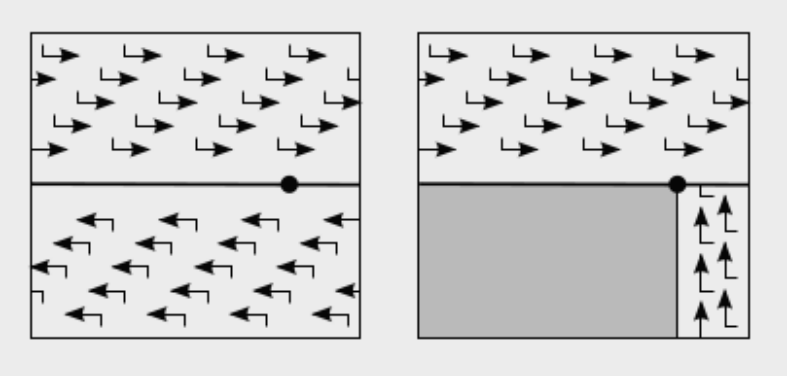


Figura 3.6: Exemplo da execução original de Myers-Miller e da execução ortogonal do CUDAlign. A área em cinza é ignorada durante a busca, diminuindo a área de busca da coordenada [1].

3.2.3 Estágio 3 - Divisão das Partições

O Estágio 3 irá aumentar o número de coordenadas conhecidas que cruzam o alinhamento ótimo. O Estágio 2 gera diversas partições bem definidas - as coordenadas de início e fim definem as partições - e sem qualquer dependência, possibilitando seu processamento em qualquer ordem pela GPU (ou múltiplas GPUs).

As partições são divididas em várias colunas, uma para cada coordenada pertencente ao alinhamento ótimo, como mostra a Figura 3.7

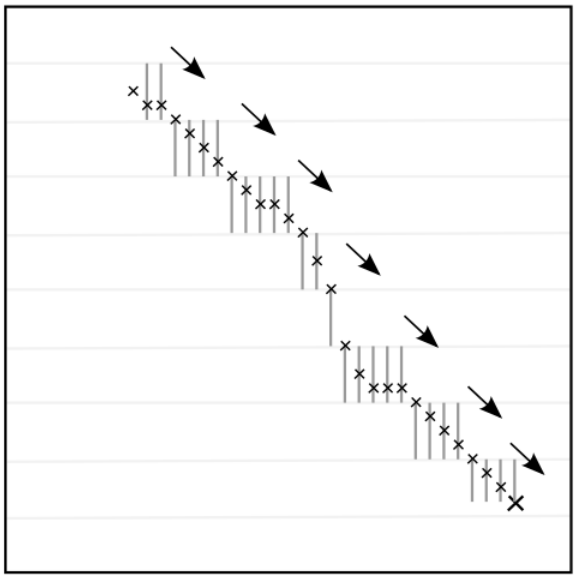


Figura 3.7: Exemplo da execução do estágio 3. [1].

3.2.4 Estágio 4 - Myers-Miller Otimizado

A partir do estágio 4, a execução não ocorre mais em GPU. Aqui já são conhecidos vários pontos do alinhamento e o controle é passado para a CPU. Com base nas coordenadas conhecidas, mais partições são criadas, e mais coordenadas são encontradas. A criação de novas partições para busca é realizada enquanto o tamanho da partição for maior que 16 bases - tamanho definido empiricamente.

A cada nova iteração, o número de coordenadas conhecidas aumenta em até 2x, e várias iterações podem ocorrer. Da mesma forma, como no Estágio 3, a ordem de execução aqui é irrelevante.

Como visto na Seção 2.2.4, o algoritmo divide as partições em problemas menores. No entanto, isso pode levar a um tamanho muito diferente entre partições. Uma otimização foi proposta para melhorar o desempenho do Estágio 4: a divisão balanceada.

Divisão balanceada: essa otimização altera o algoritmo para que as partições possam ser divididas nas linhas ou colunas centrais. Dessa maneira, a maior dimensão da partição será dividida, diminuindo o esforço de busca pelas coordenadas. A Figura 3.8 mostra a diferença entre o método original e a otimização proposta.

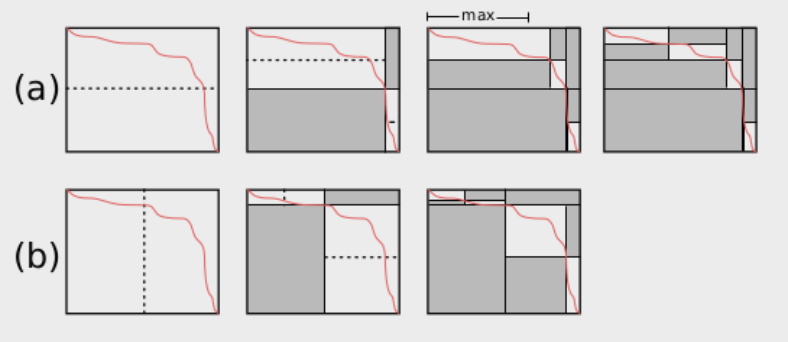


Figura 3.8: Divisão balanceada proposta no CUDAlign - a) mostra a divisão original proposta por Myers-Miller e b) mostra a divisão balanceada [1].

3.2.5 Estágio 5 - Obtenção do Alinhamento Completo

O Estágio 5 alinha as partições, obtidas no Estágio 4, em CPU utilizando o algoritmo de Needleman-Wunsch (Seção 2.2.1). Os alinhamentos de cada partição são concatenados, gerando o alinhamento das sequências.

Com o alinhamento encontrado, o algoritmo gera um arquivo de saída em formato binário - para diminuir o tamanho da saída de dados -, com as seguintes informações: coordenadas inicial e final do alinhamento, *score* ótimo, e duas listas GAP_1 e GAP_2 com tuplas de coordenadas de abertura e tamanho do *gap*, uma para cada *string*. O arquivo pode ser utilizado para reconstrução do alinhamento, mas são necessárias as sequências.

3.2.6 Estágio 6 - Visualização

O Estágio 6 é opcional. Aqui, utiliza-se a saída gerada no Estágio 5 para criar um arquivo texto ou utilizar alguma aplicação para gerar uma visualização gráfica do alinhamento. O foco deste trabalho será o desenvolvimento de tal ferramenta de visualização gráfica.

Tendo o arquivo binário - gerado pelo estágio 5 - e as sequências utilizadas para gerá-lo, é bastante simples reconstruir o alinhamento para utilização na ferramenta. Para realizar a reconstrução, basta fixar as coordenadas inicial (i_0, j_0) e final (i_1, j_1) e a união de todos os *gaps*, utilizando sempre o mais próximo a coordenada atual.

Suponha o alinhamento mostrado nas Figuras 2.1 e 2.2. O Estágio 5 iria gerar o binário contendo os seguintes dados: (0, 0), (9,8), GAP_1 = [(5,5,1), (9,8,1)] e GAP_2 = [(0,1,1)]. Sabendo que S_0 = [T,G,C,A,A,T,G,G,G] e S_1 = [A,T,G,C,A,T,T,G], tem-se que o alinhamento ocorre nas coordenadas: (0,0), (0,1) *gap* mais próximo, (1,2), (2,3), (3,4), (4,5), (5,5) *gap* mais próximo, (6,6), (7,7), (8, 8), (9, 8) *gap* mais próximo e coordenada final. A Figura 3.9 mostra a reconstrução do alinhamento global das sequências.

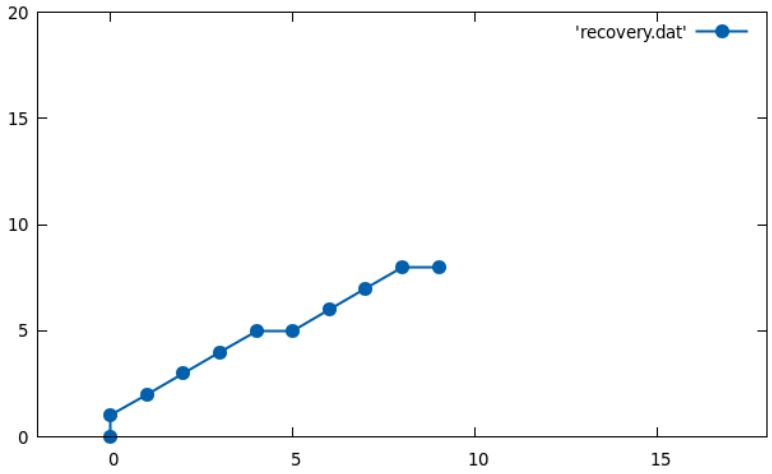


Figura 3.9: Recuperação do alinhamento por meio do software GNUPlot. Os pontos em X representam os caracteres da *string* S_0 e os pontos em Y os caracteres na *string* S_1.

3.3 CUDAlign 2.1

Na versão 2.1, o autor em [1] propõem a otimização de *block pruning* para acelerar o processamento da matriz de DP para alinhamento local ótimo. A otimização é baseada na ideia de que dependendo da semelhança entre as sequências, o algoritmo pode descartar células com *score* muito baixo e que, portanto, não têm chances de fazerem parte do alinhamento local ótimo.

3.3.1 Definições

Para uma célula (i, j), tem-se que Δ_i é a distância da linha i até a última linha da matriz, tal que $\Delta_i = m - i$. De maneira semelhante, tem-se que Δ_j é a distância da coluna j até a última coluna da matriz, tal que $\Delta_j = n - j$.

Temos também que $\Delta_i - cell$ é uma célula que está mais próxima da última linha do que da última coluna, tal que $\Delta_i < \Delta_j$. Ainda, $\Delta_j - cell$ é a célula que está mais próxima da última coluna do que da última linha, tal que $\Delta_j < \Delta_i$

Por fim, se a célula (i, j) possui *score* $H(i, j)$, temos que o *score* máximo obtido no alinhamento local é $H_{max}(i, j) = H(i, j) + min(\Delta_i, \Delta_j) * ma$, o que ilustra uma situação onde a partir de (i, j), tem-se o alinhamento de *match* perfeito entre as bases até chegar na última linha ou coluna da matriz (o que estiver mais perto). Assim, pode-se definir como *prunable* uma célula cujo o *score* máximo $H_{max}(i, j)$ é menor que o *score* máximo encontrado até o momento.

3.3.2 *Block Pruning*

Para reduzir o tempo de processamento, temos que definir se uma célula é *prunable* antes mesmo de saber seu *score*. Dessa maneira, utiliza-se a seguinte ideia: se (i-1, j-1), (i-1, j) e (i, j-1) são *prunable*, então (i, j) também é, dispensando o processamento de (i, j). A otimização *block pruning* estende essa ideia para blocos inteiros, descartando-os. O processamento nos blocos considera a célula com maior *score* no canto superior esquerdo - maior Δ_i e Δ_j possíveis -, tornando-o mais rápido do que a aplicação a células. A formalização do método não será abordada neste trabalho, porém pode ser lida em [19].

3.4 CUDAlign 3.0

Apesar das otimizações propostas até a versão 2.1, Sandes mostra que o alinhamento em sequências de 33 MBP, possui tempo de execução foi superior a 8 horas [1]. Assim, uma nova arquitetura com múltiplas GPUs foi projetada para a versão 3.0, para acelerar o Estágio 1, que é o mais longo.

3.4.1 Arquitetura Multi-GPU

Nesta versão, o CUDAlign utiliza múltiplas GPUs simultaneamente. Cada GPU é responsável por um subconjunto de colunas da matriz de DP (intervalo). Se as GPUs forem idênticas, os intervalos irão possuir o mesmo tamanho, porém se forem diferentes, os intervalos serão proporcionais à capacidade de processamento de cada GPU.

Para cada GPU, é associado um processo, e cada processo possui três *threads*, sendo duas *threads* de comunicação e uma de gerenciamento. A última é responsável por invocar o *kernel* em GPU e por interagir com as *threads* de comunicação. As *threads* de comunicação irão enviar e receber as células entre os processos, por meio de *sockets*. A Figura 3.10 mostra como é organizada a arquitetura da versão 3.0.

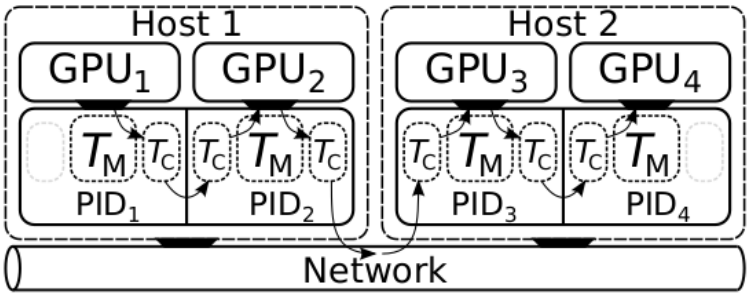


Figura 3.10: Arquitetura Multi-GPU do CUDAlign 3.0 [1].

3.4.2 *Buffers* de Comunicação

Como há comunicação entre processos, foram associados *buffers* de comunicação a cada *thread*, que podem receber ou enviar dados. O proposito destes é esconder a latência provocada pela comunicação entre as GPUs, tanto pelo *overhead*, quanto pela qualidade da conexão de rede.

Os *buffers* também são indicadores da qualidade do balanceamento da divisão de intervalos entre as GPUs. Na Figura 3.11 o *buffer* de saída O_1 possui poucos dados esperando para serem enviados para o *buffer* de entrada I_2, que possui muitos dados esperando para serem consumidos. É um sinal de que a GPU_1 está produzindo dados mais rapidamente do que a GPU_2 pode processar. Porém, se ambos os *buffers* de entrada e saída estão praticamente vazios, como é o caso dos *buffers* O_3 e I_4, significa que a divisão está balanceada.

Dessa forma, quando a divisão é balanceada, o uso dos *buffers* é praticamente constante e a velocidade de processamento é máxima. Quando a divisão está desbalanceada, os *buffers* podem ficar cheios, bloqueando as *threads* e reduzindo a velocidade de processamento.

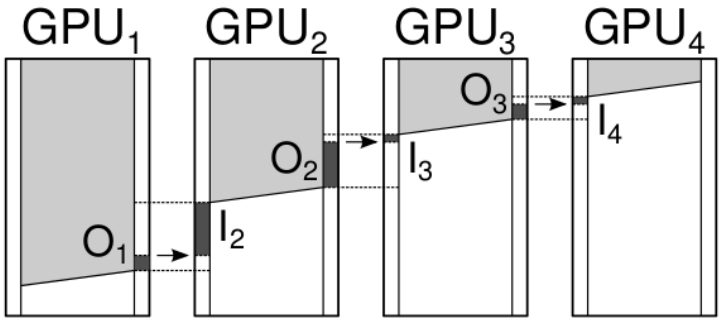


Figura 3.11: *Buffers* de comunicação entre GPUs [1].

3.5 CUDAlign 4.0

Com a utilização da arquitetura Multi-GPU (seção 3.4), o desafio é utilizá-la para recuperação do alinhamento - Estágios 2 a 5 - de forma paralela, já que a execução do estágio 2 é sequencial. Foram propostas uma alteração no Estágio 1, além da otimização: *Incremental Speculative Traceback* (IST).

3.5.1 Estágio 1 Modificado

A versão 3.0 do CUDAlign apresentava uma limitação no tamanho das sequências devido à memória de textura disponível nas GPUs. A limitação impedia o alinhamento entre sequências maiores que 134 MBP (2^{27}). Para permitir o alinhamento de sequências maiores, a versão 4.0 propõem a subdivisão da matriz de DP em quadrantes que caibam na memória de textura da GPU. Cada subdivisão é calculada individualmente, uma após a outra, de tal maneira que a primeira linha/coluna de uma subpartição é obtida da última da subpartição anterior.

Além disso, as linhas especiais agora serão salvas em um diretório exclusivo para cada GPU, ou em memória RAM para evitar o *overhead* das operações de I/O em disco, que geram gargalo na execução.

3.5.2 *Pipelined Traceback* (PT)

Com a utilização das modificações no Estágio 1 (3.5.1), a utilização de várias GPUS cria um *pipeline* para os Estágios 2 a 4.

Após a conclusão do Estágio 1, a última GPU inicia o Estágio 2, enquanto as demais ficam ociosas, pois dependem do *crosspoint* (ponto médio pelo qual o alinhamento ótimo passa), calculado pela GPU à direita, criando um caminho crítico de execução. Tendo o *crosspoint*, a GPU pode calcular os Estágios 3 e 4 em *pipeline*.

Há duas fases em que parte das GPUs fica ociosa: no início do preenchimento do *wavefront* (Seção 2.1) e após a conclusão do Estágio 1, em que a GPU_{i-1} aguarda pelo *crosspoint* obtido pela GPU_i no estágio 2. A Figura 3.12 exemplifica as fases ociosas das GPUs.

3.5.3 *Incremental Speculative Traceback* (IST)

Após realizar a análise de várias execuções do CUDAlign, notou-se que os *crosspoints* encontrados em colunas intermediárias tendem a coincidir com as posições onde ocorrem os *scores* máximos dessas colunas. Sabendo disso, o IST gera uma possível coordenada pertencente ao alinhamento ótimo e, após a execução do Estágio 1, utiliza essa possível coordenada para realizar o processamento da partição (Estágio 2). Quando a GPU_i for acionada para o Estágio 2, esta verifica o *crosspoint* recebido pela GPU_{i+1}. Se os valores especulado e real coincidirem, a GPU_i apenas repassa o *crosspoint* para a próxima GPU_{i-1}. Caso contrário, a GPU_i recalcula a partição novamente, com o valor correto.

Também observou-se que regiões com pouca similaridade - muitos *gaps* ou *mismatches* - são difíceis de prever. Assim, ao invés de utilizar apenas os *crosspoints* das colunas intermediárias, as GPUs vizinhas trocam informações. Dessa maneira, as GPUs enviam os

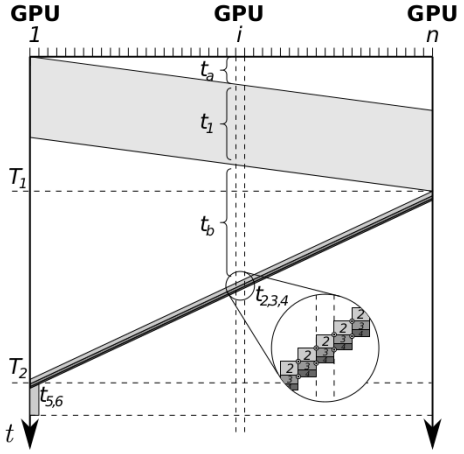


Figura 3.12: As áreas em branco indicadas por t_a e t_b definem os tempos que a i-ésima GPU fica ociosa. As áreas cinzas indicam os momentos em que a i-ésima GPU está processando [1].

crosspoints especulados, e calculam um novo *crosspoint* com base na especulação recebida - menos no caso em que o *crosspoint* recebido já tenha sido calculado pea GPU_{i+1}. Esse processo incremental aumenta a chance de se obter uma especulação correta. Os resultados das especulações - *scores*, linhas especiais, dentre outros - são armazenados em um área chamada de *cache*. Ainda, há um valor máximo de especulações, *max_spec_tries*, para limitar o uso de memória. A Figura 3.13 mostra como funciona a otimização IST.

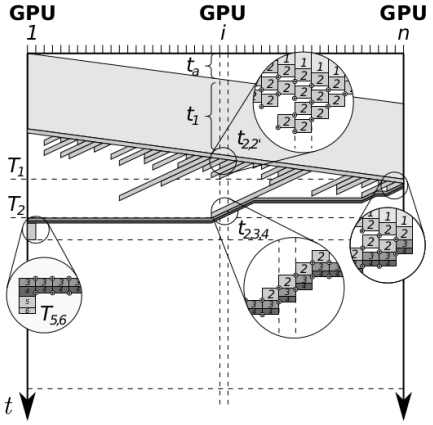


Figura 3.13: t_a é o tempo que cada GPU fica ociosa durante o preenchimento do *wavefront* e t_1 mostra o tempo que cada GPU gasta no estágio 1. As áreas em cinza, tais como exemplificadas por $t_{2.2'}$, exemplificam o processamento do IST, onde a GPU estaria ociosa. Por fim, as linhas diagonais em cinza escuro indicam o tempo gasto para recalcular uma partição com valor especulado incorretamente [1].

3.6 *Multi-plataform Architecture for Sequence Aligners* (MASA)

O CUDAlign, como o próprio nome indica, se baseia na arquitetura CUDA da NVidia. No entanto, durante seu desenvolvimento, viu-se que era possível portar algumas de suas otimizações para outras plataformas de de hardware (tais como CPUs e GPUs de outros fabricantes), e de software (tais como OpenCL, OpenMP e OmpSs). Assim, Sandes propôs o desacoplamento entre o código independente e específico, resultando na arquitetura MASA [1].

O MASA é um *framework*, baseado no CUDAlign, para facilitar o desenvolvimento de soluções de alinhamento em outras plataformas. Por ser uma arquitetura modular, possibilita a criação de novas implementações multiplataforma. Cada customização do MASA é definida como uma "extensão", e deve produzir um binário executável para a plataforma que for escolhida.

Dentre as otimizações propostas em todas as versões, o *Block Pruning* (Seção 3.3.2) é a mais relevante e, portanto, foi implementada para ser portável para qualquer plataforma. No entanto, ainda só está disponível para gerar alinhamentos locais e em dispositivos únicos (1 GPU ou 1 CPU).

3.6.1 Arquitetura MASA

A arquitetura MASA foi projetada com cinco módulos: Gerência de Dados, Comunicação, Estatística, Gerência de Estágios e *Aligner*. A Figura 3.14 apresenta a organização da arquitetura MASA.

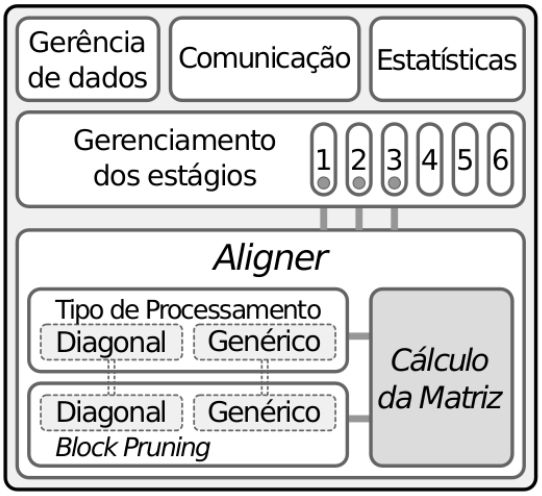


Figura 3.14: Organização da arquitetura Masa [1].

Os quatro primeiros módulos foram projetados para serem portados para qualquer plataforma. Já o módulo *Aligner*, que é o responsável por calcular a matriz de DP, é onde está o código específico de cada plataforma. Ele possui três submódulos: (a) Tipo de processamento; (b)*Block Pruning*, que são customizáveis para cada plataforma; (c) Cálculo da matriz, que é passível de otimizações específicas à plataforma escolhida.

Por fim, é importante destacar as funções de cada módulo. O módulo *Aligner*, como dito anteriormente, é responsável por calcular o alinhamento. Ele executa a equação de recorrência de SW (Seção 2.2.2) ou NW (Seção 2.1). Ainda, permite a escolha do tipo de processamento - *wavefront* ou genérico [1] -, e *block pruning* (Seção 3.3.2).

A Gerência de Dados é responsável pela entrada e saída de dados do sistema. É o módulo responsável por ler as *strings* das sequências que serão alinhadas, processar os parâmetros passados pela linha de comando, armazenar as linhas especiais (Seção 3.2.1) , gerar os arquivos de saída (binário ou texto) (Seções 3.2.5 e 3.2.6), dentro outros.

O módulo de Estatísticas irá gerar as informações a respeito do tempo de execução em cada estágio, uso de memória e disco, porcentagem de blocos descartados no *block pruning*, dentre outras.

O módulo de Gerência de Estágios coordena a execução dos estágios 1 a 3, utilizando o *Aligner*, e executa os estágios 4 a 6 em CPU, como descrito na Seção 3.2.

Por fim, o módulo de Comunicação é responsável pela troca de informações entre os diversos nós do processamento, permitindo o uso de múltiplas GPUs ou dispositivos para processamento.

Capítulo 4

Ferramentas Para Execução de Comparações e Visualização de Alinhamentos

Na bioinformática, o campo de alinhamento de sequências genéticas - sejam nucleotídeos ou proteínas - tem muita importância. Atestar este fato é simples, pois ao pesquisar por ferramentas de alinhamento na Internet, encontra-se uma grande variedade de ferramentas, públicas e privadas, com propósito de realizar alinhamentos de sequências genéticas.

Este capítulo tem por objetivo apresentar algumas das ferramentas disponíveis, mostrando suas vantagens e desvantagens, para que construa-se uma base do que se deve ou não fazer no que diz respeito a interface, documentação e apresentação de resultados aos usuários.

4.1 EMBOSS

A ferramenta EMBOSS é desenvolvida pelo EMBL-EBI (European Molecular Biology Laboratory - European Bioinformatics Institute). O EMBOSS possui um conjunto de ferramentas, de uso aberto, de bioinformática. No alinhamento de pares, a plataforma possui ferramentas de alinhamento global - Needle, que utiliza o algoritmo de Needleman-Wunsch (Seção 2.2.1), e Strecher, que utiliza uma versão modificada do algoritmo de Needleman-Wunsch, para o alinhamento de sequências maiores -, local - Water, que utiliza o algoritmo de Smith-Waterman (Seção 2.2.2), Matcher e LALIGN - que fazem o alinhamento heurístico -, e, por fim, genômico - GeneWise [15].

4.1.1 EMBOSS Needle e EMBOSS Water

Como dito anteriormente, as ferramentas EMBOSS Needle e Emboss Water realizam o alinhamento global e local, respectivamente, de sequências de DNA ou proteínas. Ambas podem ser utilizadas diretamente no site [20][21], e apresentam uma interface bastante simples. A Figura 4.1 mostra a interface de usuário para utilização na plataforma online.

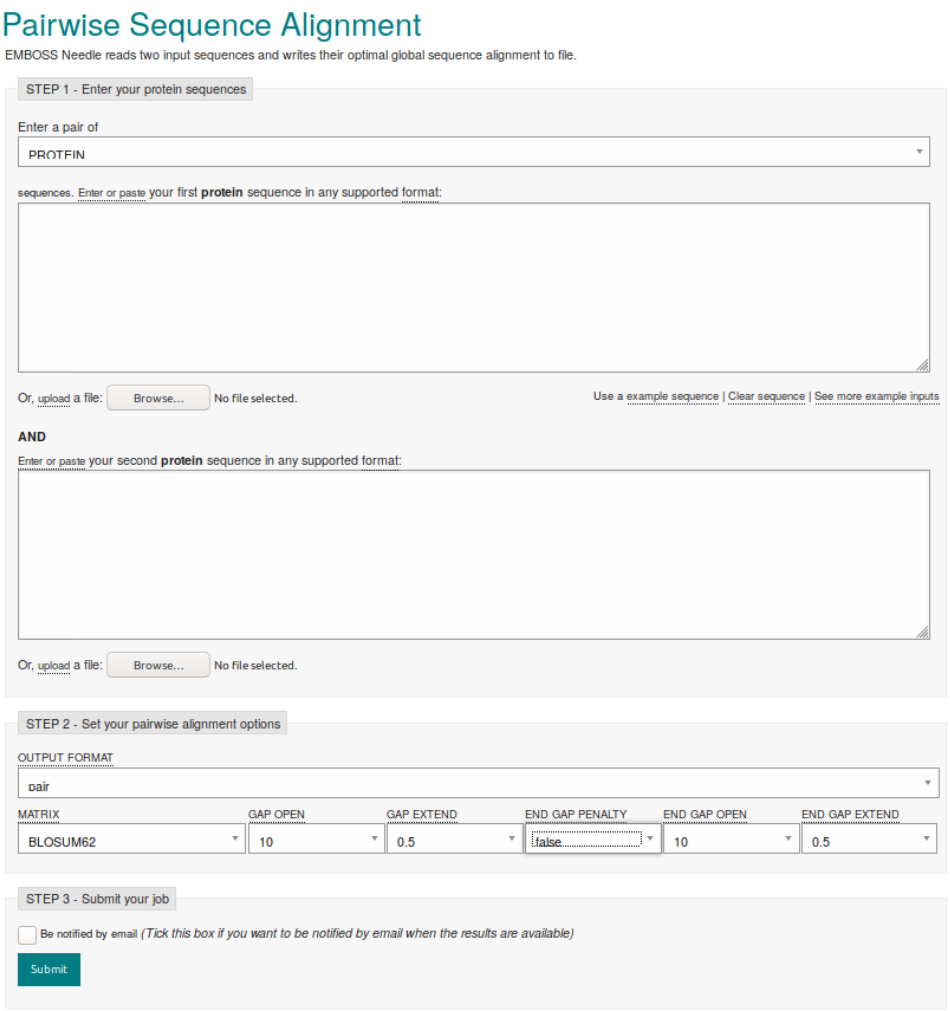


Figura 4.1: Interface de usuário (UI) da ferramenta EMBOSS Needle e Water, disponível na plataforma EMBL-EBI.

Apesar da facilidade do uso, o EMBOSS apresenta um problema com relação a customização dos valores do modelo de *score* (Seção 2.1). Os valores de *match* e *mismatch* são definidos pela matriz de substituição escolhida (Seção 2.1), dentre as opções disponíveis. Os valores para *gaps* também podem ser escolhidos, de maneira igualmente limitada. O usuário é apresentado com algumas opções de valores para o *scoring* de *gaps*, mas não

pode escolher valores arbitrários. As Figuras 4.2 e 4.3 mostram a customização de valores para o *scoring*.

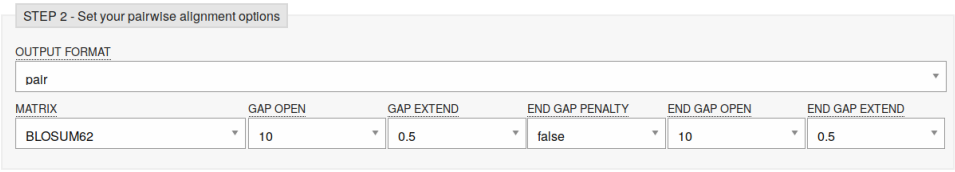


Figura 4.2: Customização do modelo de *scoring* da ferramenta EMBOSS Needle.

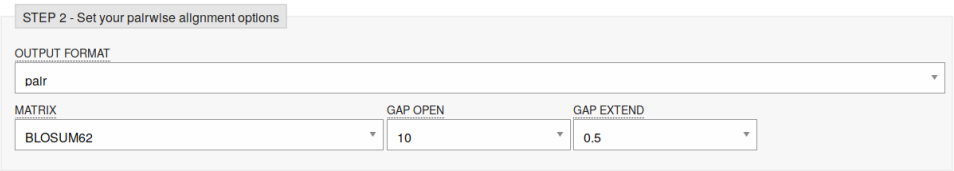


Figura 4.3: Customização do modelo de *scoring* da ferramenta EMBOSS Water.

Em sua documentação, a ferramenta não apresenta nenhum limite no tamanho das sequências sendo analisadas, e diz apenas que "The maximum size for any program depends only on how much memory your system has." [22]. Apesar de não possuir um limite explícito, ao executar a ferramenta em *hardware* próprio, com 8 GB de memória RAM instalados, para o alinhamento de arquivos de 10 KB [5][6], 150 KB [23][24], 1 MB [25][26] e 5 MB [3][4], o limite máximo foi atigindo com dois arquivos de 10 KB. Por utilizar os algoritmos de Needleman-Wunsch e Smith-Waterman, a complexidade espacial é $O(n*m)$, onde n e m são os tamanhos das sequências, o que torna a ferramenta inviável para comparação de sequências médias ou longas em computadores pessoais. As Figuras 4.4 e 4.5 mostram a ferramenta falhando na tentativa de alinhamentos de 150KB.

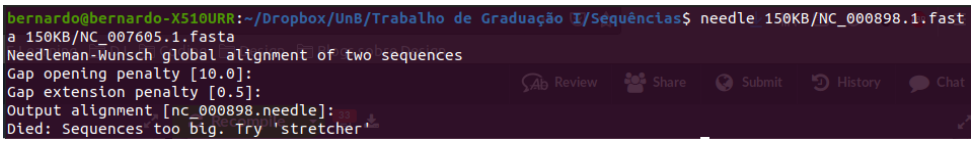


Figura 4.4: EMBOSS Needle falhando ao tentar alinhar duas sequênciasde 150KB.

Para sequências com arquivos maiores que 10 KB, o programa sugere a utilização das aplicações *stretcher*, *matcher* ou *supermatcher*, para alinhamentos global - utilizando Myers-Miller (Seção 2.2.4), local - utilizando Waterman-Eggert - e local não-ótimo - utilizando uma mistura de *word-match* e Smith-Waterman -, respectivamente. As aplicações

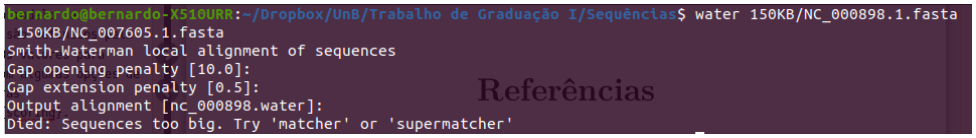


Figura 4.5: EMBOSS Water falhando ao tentar alinhar duas sequências de 150KB.

alternativas - para alinhamentos maiores - tem limite máximo de 1 MB na plataforma online e, apesar de ser possível executar as alternativas em computadores pessoais, o tempo para conclusão do alinhamento de sequências longas é inviável. Ao executar o alinhamento de dois arquivos de 1 MB, o *stretcher* precisou de aproximadamente 90 minutos para concluir o alinhamento.

O EMBOSS não gera saída gráfica para alinhamentos de pares, apenas a versão textual, que apresenta o alinhamento das sequências caractere a caractere, mostrando os *matches*, *mismatches* e *gaps*. Ainda, a saída também apresenta dados estatísticos, embora limitados, sobre o alinhamento, quais sejam: tamanho do alinhamento, identidade, similaridade, *gaps* e *score*. A Figura 4.6 mostra uma saída gerada após a execução do EMBOSS Needle.

É importante ressaltar ainda, que a versão online possui APIs REST (*Representational State Transfer*) e SOAP (*Simple Object Access Protocal*), e, para sequências longas, o usuário pode disponibilizar um endereço de *e-mail* para que seja avisado pela plataforma da conclusão do alinhamento.

4.2 MUMmer

A ferramenta de alinhamento MUMmer foi desenvolvida para realizar, de maneira rápida, o alinhamento heurístico de sequências longas de DNA. Para realizar tais alinhamentos, o MUMmer utiliza *suffix trees*, que são estruturas de dados que permitem construção e pesquisa em tempo linear ($O(n)$). Para atingir maiores velocidades em alinhamentos longos, o MUMmer não gera alinhamentos ótimos. Assim como o EMBOSS (Seção 4.1), o MUMmer é um software aberto e está disponível para *download* e contribuições na plataforma Github [27].

Em sua documentação, o MUMmer não apresenta limitações explícitas para o tamanho das sequências, informando apenas que é necessário ter memória RAM e espaço em disco suficientes, mas que, de maneira geral, 512 MB de memória RAM e 1 GB de espaço em disco são suficientes para a maioria das comparações de tamanho médio. Ao realizar os testes de execução com os mesmo arquivos (10 KB [5][6], 150 KB [23][24], 1 MB [25][26] e 5 MB [3][4]), o MUMmer executou todos sem problemas - o maior tempo de execução

```
########################################
# Program: needle
# Rundate: Mon 18 Nov 2019 20:18:40
# Commandline: needle
#    [-asequence] 10KB/AF133821.1.fasta
#    [-bsequence] 10KB/AY352275.1.fasta
# Align_format: srspair
# Report_file: af133821.needle
########################################

#=======================================
#
# Aligned_sequences: 2
# 1: AF133821.1
# 2: AY352275.1
# Matrix: EDNAFULL
# Gap_penalty: 10.0
# Extend_penalty: 0.5
#
# Length: 10771
# Identity:    8567/10771 (79.5%)
# Similarity:  8567/10771 (79.5%)
# Gaps:        1227/10771 (11.4%)
# Score: 37138.5
#
#
#=======================================

AF133821.1         1 AAANTACCTTNTTAGAACTGCCTTTGCTGTACCCTGTAAGTTTTGGGTTG     50

AY352275.1         0 --------------------------------------------------      0

AF133821.1        51 TTGCATTTTCATGTTCATTTATCTCTACATAGTTTTTAAAACTTCCTTTT    100

AY352275.1         0 --------------------------------------------------      0

AF133821.1       101 TGATTTCTTCTTGACCCATTGGTTGTTCAGGATCATGTTGTTTAGTTTCT    150

AY352275.1         0 --------------------------------------------------      0

AF133821.1       151 ACATACTTGTGAATTGTCCAAAATGCCTTCTGTTATTTATTTCTAGTTTT    200

AY352275.1         0 --------------------------------------------------      0

AF133821.1       201 ATACCATAGTGTCCAGAAAAGTTACATGATATGATTTCTGTCTTCTTAAT    250

AY352275.1         0 --------------------------------------------------      0

AF133821.1       251 GTGTTAAGATTCATTTTGTGGCTTAACATATGATCTATCCTGGAA-----    295
                                                                  |||||
AY352275.1         1 ----------------------------------------TGGAAGGGCT     10
```

Figura 4.6: Saída gerada pelo EMBOSS Needle ao alinha duas sequências de 10 KB. Apenas o início da saída é mostrado, para fins de exemplo.

foi para os arquivos de 5 MB, que demorou 5 segundos. A Figuras 4.7 mostra a execução do alinhamento dos arquivos de 5 MB.

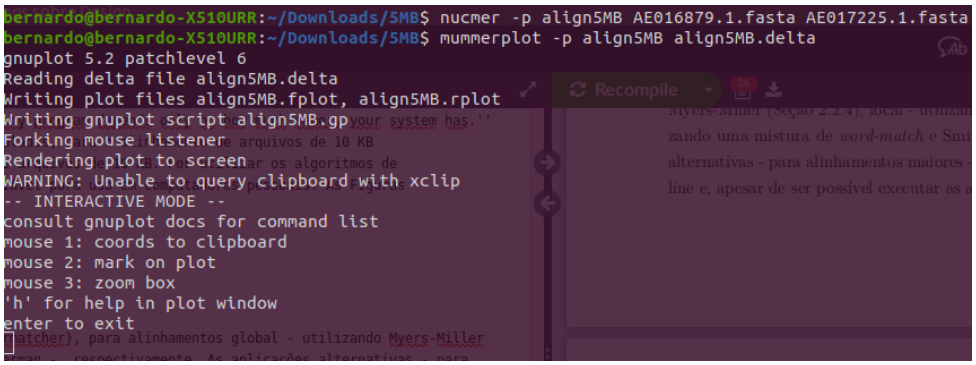
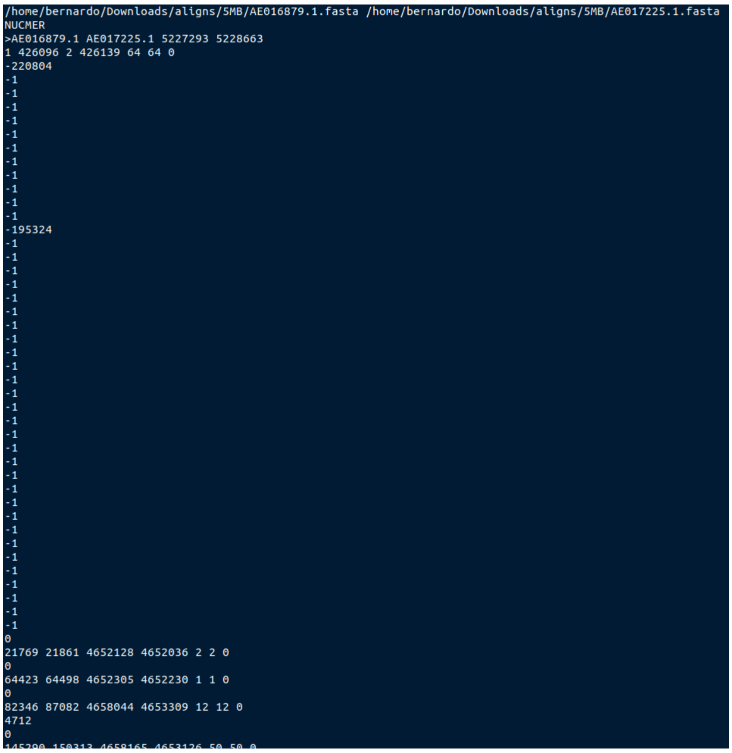


Figura 4.7: Execução do alinhamento dos arquivos de 5 MB [3][4] pelo MUMmer, e geração do arquivo de plotagem pelo *mummerplot*.

O arquivo saída gerado pela execução do MUMmer pode gerar um arquivo para plotagem no GNUPlot [28], por um dos *scripts* disponíveis na ferramenta, nomeadamente o *mummerplot*. É possível interagir com o gráfico gerado, podendo ter acesso as coordenadas, *zoom-in* e *zoom-out*, mas não é possível ter acesso as sequências no gráfico. As Figuras 4.8 e 4.9 mostram a saída textual e a plotagem do alinhamento gerado em 4.7.



```
/home/bernardo/Downloads/aligns/5MB/AE016879.1.fasta /home/bernardo/Downloads/aligns/5MB/AE017225.1.fasta
NUCMER
>AE016879.1 AE017225.1 5227293 5228663
1 426096 2 426139 64 64 0
-220804
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-195324
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
0
21769 21861 4652128 4652036 2 2 0
0
64423 64498 4652305 4652230 1 1 0
0
82346 87082 4658044 4653309 12 12 0
4712
0
145290 150313 4658165 4653126 50 50 0
```

Figura 4.8: Saída textual do alinhamento gerado em 4.7 pelo MUMmer.

Como não há uma plataforma online para utilização do MUMmer, seu uso é restrito à sistemas UNIX. Ainda, para sua utilização, o usuário deve se certificar que as sequências sendo alinhadas estão corretas, pois o MUMmer não irá realizar verificação de formato das sequências. Caso os formatos das sequências estejam incorretos, a execução irá retornar erros. Por fim, é importante ressaltar que a plataforma possui uma arquitetura modular, mas as versões dos módulos podem não ser compatíveis entre si, ocasionando problemas na execução.

4.3 BLAST

O BLAST (*Basic Local Alignment Search Tool*) é desenvolvido e mantido pelo *National Center for Biotechnology Information* nos Estados Unidos. É composto por um conjunto de ferramentas de bioinformática e está disponível para uso online em [11]. Assim como

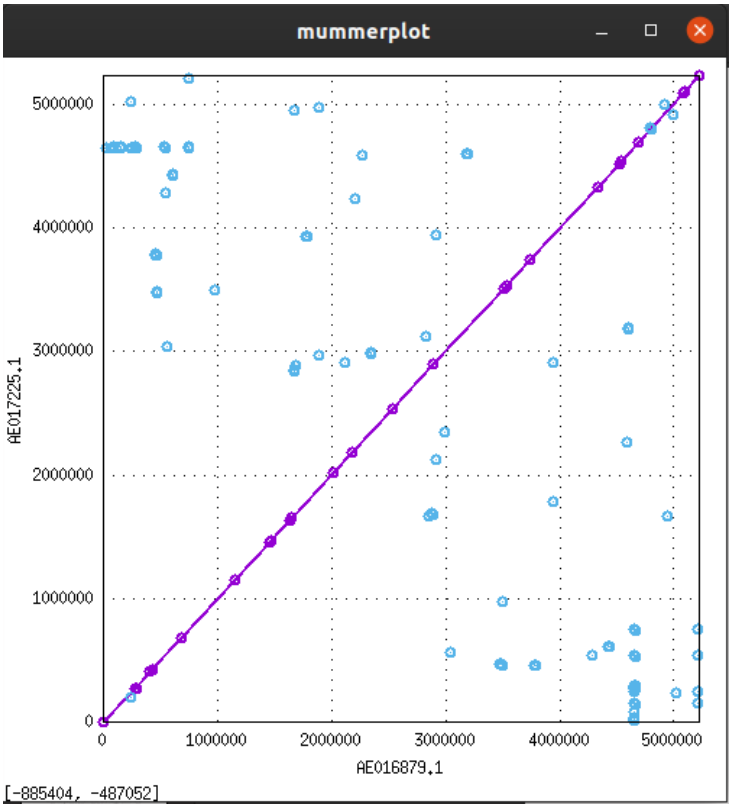


Figura 4.9: Plotagem do resultado do MUMmer, utlizando o *software* GNUPlot, do alinhamento gerado em 4.7.

o EMBOSS (Seção 4.1), a plataforma online possui uma API REST para utilização em outras aplicações. Ainda, é possível fazer o *download* da ferramenta para uso em computadores pessoais ou servidores, e está disponível para sistemas UNIX, macOS e Windows.

A plataforma online é bastante simples de ser utilizada, porém não possui opção de aviso de conclusão do alinhamento via e-mail como, por exemplo, a ferramenta EMBOSS (Seção 4.1). A Figura 4.10 mostra a interface de usuário do BLASTn. De maneira similar ao EMBOSS, é possível alterar o modelo de *scoring*, mas apenas para valores pré-determinados pela plataforma, como mostra a Figura 4.11.

Apesar da plataforma online possuir opções para alinhamentos locais e globais de DNA (BLASTn e Global Align, respectivamente), a versão para *download* possui apenas a opção de alinhamento local.

O BLASTn gera alinhamentos heurísticos, enquanto o Global Align gera alinhamentos ótimos. O BLASTn utiliza um algoritmo guloso, que garante o alinhamento ótimo somente para sequências de DNA que diferem em poucos caracteres, seguindo as condições $ind = mis - mat/2$, onde $ind < 0$, $mis < 0$ e $mat > 0$ são os *scores* de inserção/deleção, *mismatch* e *match* respectivamente [29]. A ferramenta não possui limites no tamanho das sequências sendo alinhadas, e conseguiu realizar o alinhamento de arquivos de 5 MB [3][4]

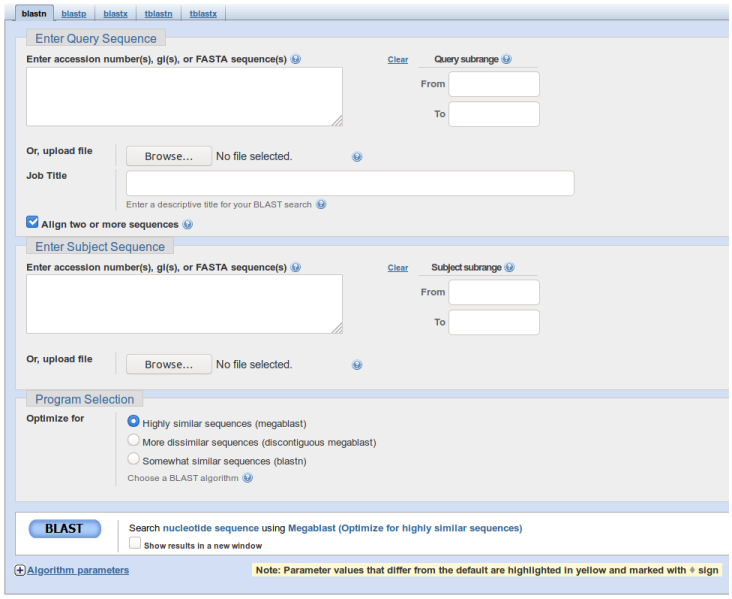


Figura 4.10: Interface de usuário (UI) da ferramenta BLASTn online.

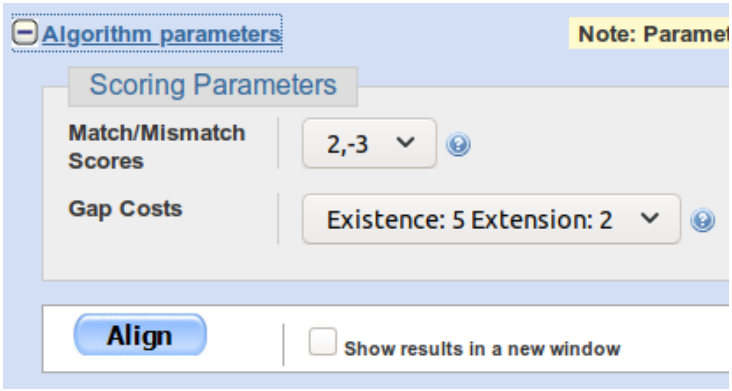


Figura 4.11: Campos para definição do modelo de *scoring* para alinhamentos globais utilizando a ferramenta Global Align do BLAST.

em poucos segundos. Já o Global Align utiliza o algoritmo de Needleman-Wunsch (Seção 2.2.1) para realizar o alinhamento. A plataforma limita o tamanho dos inputs para até 150 mil caracteres por sequência. Ambos geram saídas textuais e gráficas, como mostram as Figuras 4.12 e 4.13, e alguns dados estatísticos sobre o alinhamento, ilustrados na Figura 4.14.

4.4 FASTA

Em 1988 o FASTA foi apresentado como uma nova opção para alinhamentos heurísticos de sequências biológicas [10] (nucleotídeos e proteínas). O algoritmo se tornou popular

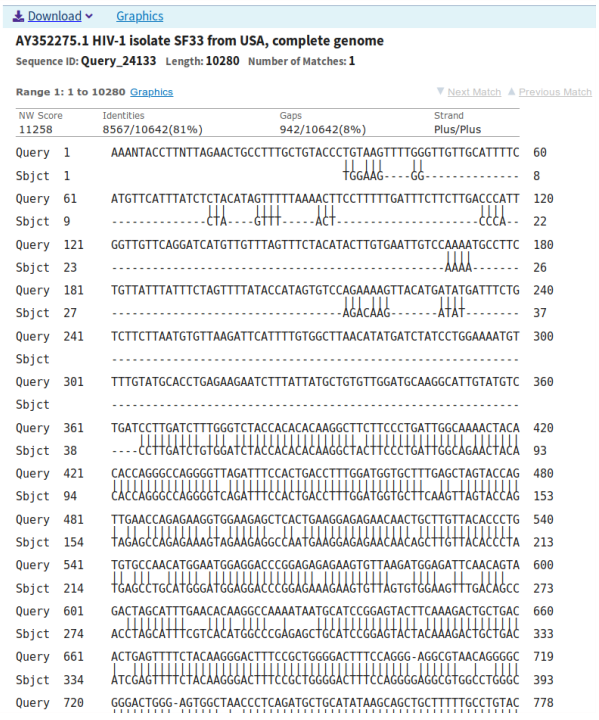


Figura 4.12: Saída textual para o alinhamento global das sequências dos arquivos de 10 KB [5][6] utilizando o BLAST.

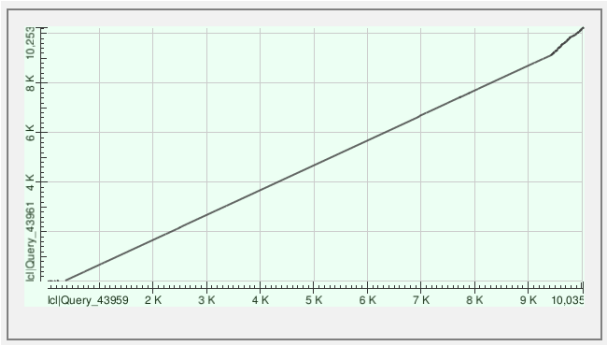


Figura 4.13: Saída gráfica para o alinhamento global das sequências dos arquivos de 10 KB [5][6].

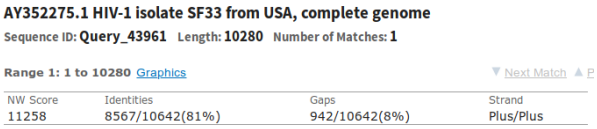


Figura 4.14: Estatísticas do alinhamento realizado em 4.13.

de tal forma, que seu formato de entrada foi adotado como padrão e é utilizado até hoje, o formato FASTA.

O algoritmo utiliza uma estratégia heurística, o que possibilita uma execução mais rápida do que algoritmos que geram alinhamentos ótimos, como NW (2.2.1), SW (Seção 2.2.2), dentre outros.

Assim como o EMBOSS (Seção 4.1), MUMmer (Seção 4.2) e BLAST (Seção 4.3), o FASTA possui uma versão para utilização online [30] e, também, é possível fazer o *download* da ferramenta para utilização em computadores ou servidores próprios. Ainda, o *software* é *open source* e hospedado em [31].

Apesar de disponibilidade na Internet facilitar a utilização, a interface de usuário do FASTA está bastante defasada e torna a experiência pouco intuitiva para novos usuários. A Figura 4.15 mostra a interface de usuário da aplicação web do FASTA.

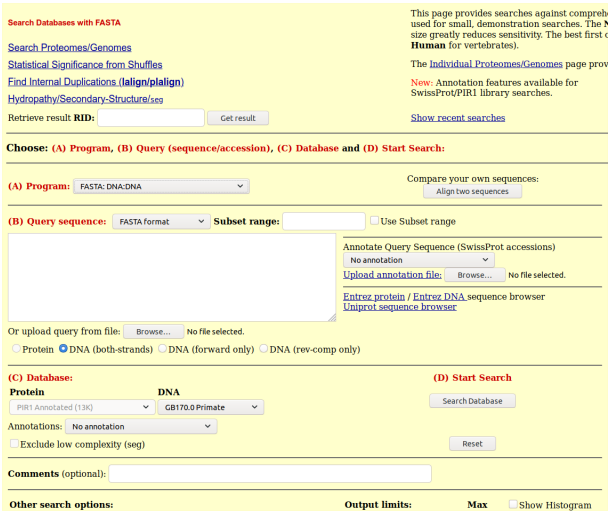


Figura 4.15: Interface de usuário da aplicação web do FASTA.

A aplicação disponibiliza uma variedade de algoritmos para alinhamentos de sequências, como: FASTA (para proteína:proteína e DNA:DNA), SSEARCH (para proteínas) que utiliza SW (Seção 2.2.2), GGSEARCH (para proteínas) que utiliza NW (Seção 2.2.1), entre outras variações do FAST. A Figura 4.16 mostra todas as opções de algoritmos disponíveis na aplicação web do FASTA.

Por utilizar uma estratégia heurística, a aplicação FASTA DNA:DNA é capaz de alinhar arquivos de 5 MB [3][4] em poucos minutos - embora seja mais lento que o MUMmer (Seção 4.2), é possível executá-lo, e em tempo razoável, diferente do EMBOSS (Seção 4.1) e BLAST (Seção 4.3). Apesar disso, a saída gerada é apenas textual, como pode ser visto nas Figuras 4.17a e 4.17b.

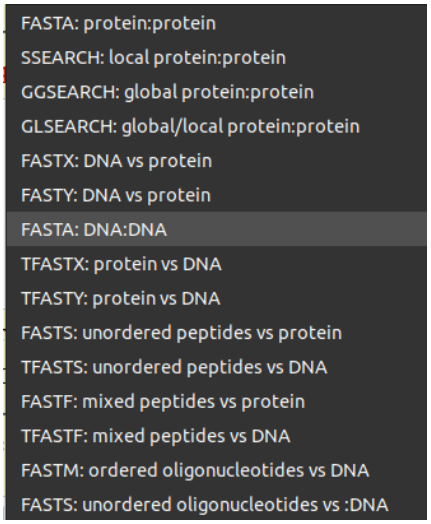
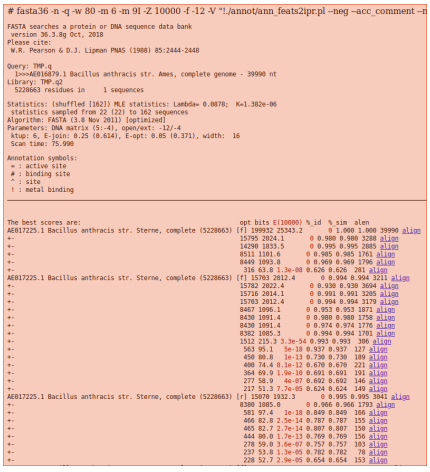
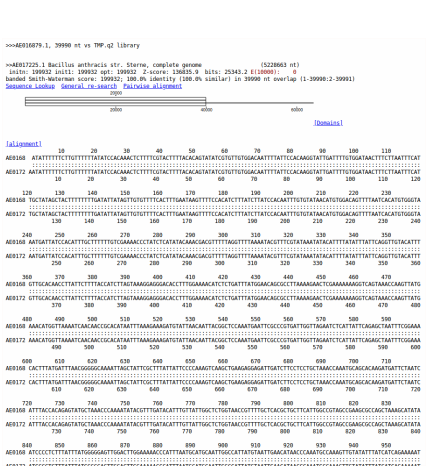


Figura 4.16: Opções de algoritmos disponíveis para execução no site do FASTA.



(a) Estatísticas e melhores *scores* do alinhamento dos arquivos de 5 MB no FASTA [3][4]



(b) Saída textual do alinhamento dos arquivos de 5 MB no FASTA [3][3]

4.5 MASA-Viewer

O MASA-Viewer é um software para visualização de alinhamentos ótimos gerados por quaisquer extensões da plataforma MASA (Seção 3.6). Desenvolvido em 2009 por Edans Sandes, o MASA-Viewer foi escrito em Java e, portanto, pode ser executado em qualquer sistema que seja compatível com a *Java Virtual Machine* (JVM).

O programa tem por objetivo unificar a visualização gráfica e textual de alinhamentos em um único software. Por ser uma aplicação Java, dispensa qualquer procedimento de instalação - além da instalação da JVM -, facilitando o uso por pesquisadores.

Ao final do Estágio 5 (Seção 3.2.5), a plataforma MASA gera um arquivo binário, que é utilizado pelo MASA-Viewer para construir os resultados. Também são necessários

os arquivos FASTA das sequências alinhadas. Com os três arquivos, o MASA-Viewer constrói o gráfico e a saída textual do alinhamento processado pela plataforma MASA. A Figura 4.18 mostra a visualização dos resultados gerados pela plataforma MASA no MASA-Viewer.

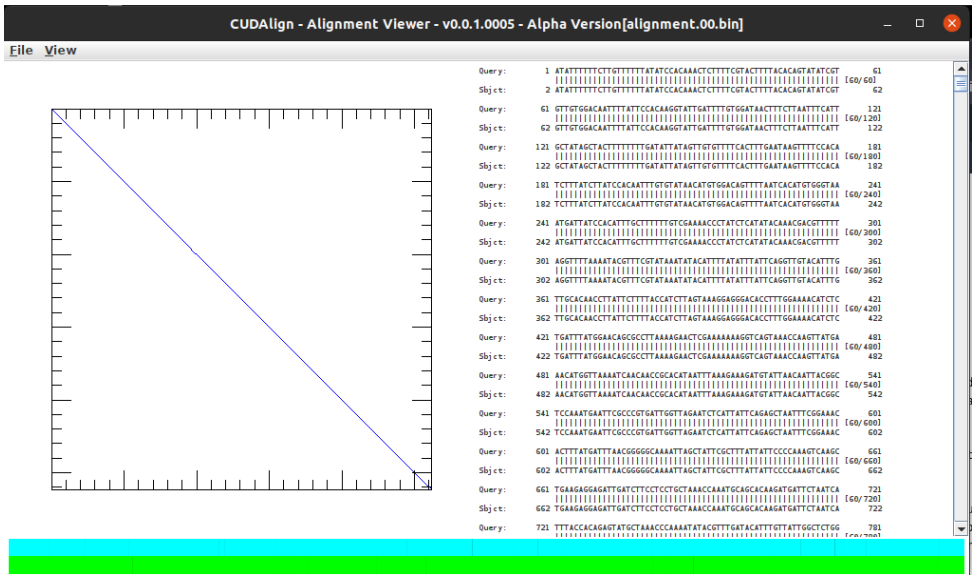


Figura 4.18: Construção do gráfico e saída textual de um alinhamento de sequências pelo MASA-Viewer.

Sobre a facilidade de uso, o MASA-Viewer apresenta uma interface gráfica bastante simples. Ao executar o programa, o usuário deve apenas clicar em *File > Open alignment*, selecionar o arquivo binário e selecionar, ou solicitar que o programa baixe, as sequências FASTA utilizadas no alinhamento, conforme a Figura 4.19 mostra.

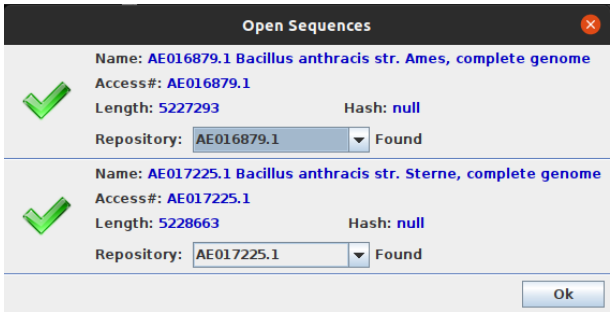


Figura 4.19: Processo de seleção das sequências FASTA para construção dos resultados pelo MASA-Viewer.

O MASA-Viewer consegue construir os resultados de grandes alinhamentos. Para referência, a Figura 4.18 executou a visualização do alinhamento de duas sequências de 5MB [3][4], consumindo aproximadamente 500MB de memória RAM.

Capítulo 5

Projeto do MASA-Webaligner

Ao utilizar o MASA-CUDAlign ou alguma das variações da plataforma, como MASA-OpenMP, a visualização é um estágio opcional (Seção 3.2.6), porém muito útil para o estudo das sequências sendo analisadas. Além da plataforma MASA de alinhamentos, foi desenvolvida, em 2009 por Edans Sandes, uma ferramenta de visualização simples chamada MASA-Viewer (Seção 4.5).

O MASA-Viewer recebe a saída binária gerada no Estágio 5 (Seção 3.2.5) e as sequências utilizadas no alinhamento, e constrói o alinhamento, gerando um gráfico e uma saída textual, além de dados sobre o alinhamento, como: *score* total, número de *matches*, *mismatches*, abertura de *gaps* e extensões de *gaps*.

O presente trabalho visa integrar o MASA-CUDAlign e MASA-OpenMP com o MASA-Viewer, através de um servidor WEB, tornando simples sua utilização por pesquisadores, abstraindo quaisquer instalações e configurações de ambiente. Sendo assim, o presente capítulo apresenta o projeto do MASA-Webaligner. Na Seção 5.1 é apresentada uma visão geral do projeto, discorrendo sobre as funcionalidades e mostrando-se os principais módulos. A seguir, na Seção 5.2, será abordado o *back-end* da aplicação, sua construção, linguagem e biblioteca utilizadas. Na Seção 5.3 será explicado o *front-end* da aplicação e o *framework* utilizado para facilitar a construção da interface de usuário da plataforma WEB.

5.1 Visão Geral

O MASA-Webaligner é uma ferramenta desenvolvida para integrar as funcionalidades do MASA-CUDAlign e MASA-OpenMP com o MASA-Viewer. É uma plataforma web desenvolvida com Express.JS [32] e React.JS [33] - *back-end* e *front-end* respectivamente - que permite ao usuário enviar um par de sequências de DNA, alinhá-las com MASA-CUDAlign/MASA-OpenMP e analisar os resultados com auxílio de gráfico e saída textual

interativos. A Figura 5.1 mostra a visão geral da plataforma. O MASA-Webaligner recebe do usuário informações sobre as sequências a serem comparadas. Caso sejam fornecidos os identificadores das sequências, o MASA-Webaligner se conecta ao site NCBI [34] e recupera as sequências em formato FASTA. As sequências são entregues ao módulo de escalonamento de execução, que vai decidir qual comparação será executada no momento. A comparação a ser executada é passada ao módulo de execução, que irá invocar o comando de execução do MASA. Ao final da execução, o MASA escreve o resultado em seus arquivos de controle e sinaliza o final da execução. O MASA-Webaligner percebe que a execução se encerrou e ativa o módulo de envio de email ao usuário ou executa o módulo de visualização do alinhamento. Os resultados são enfim entregues ao usuário, via visualização gráfica ou via e-mail. A visão geral do MASA-Webaligner está ilustrada na Figura 5.1.

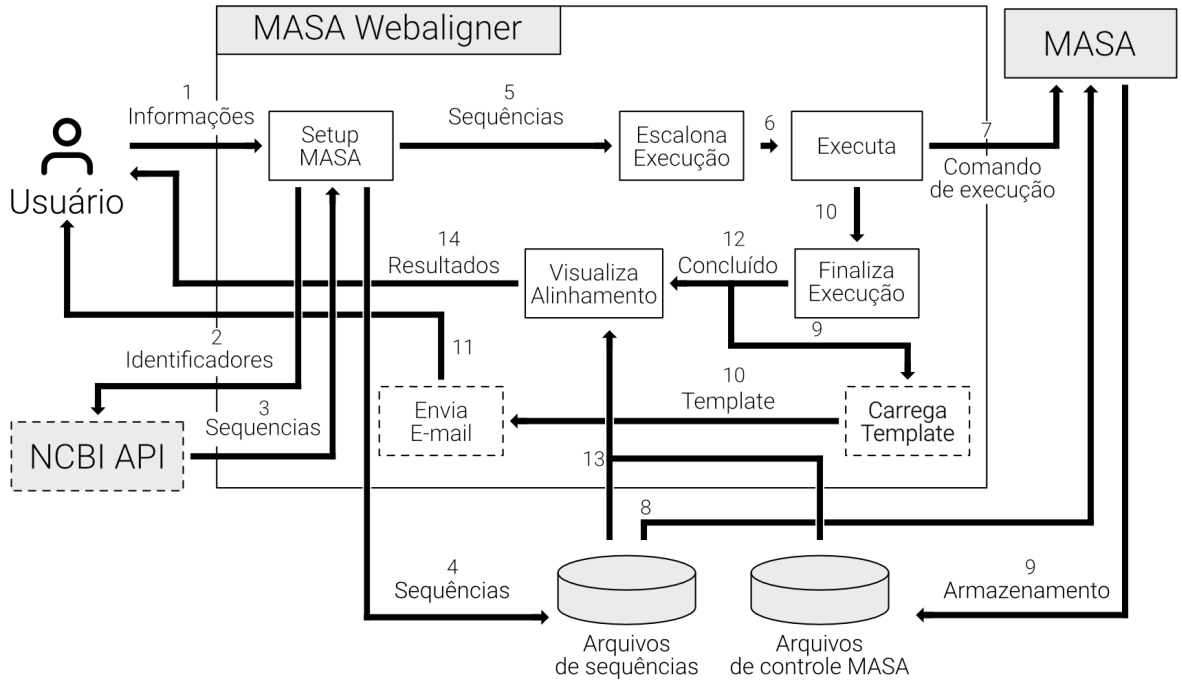


Figura 5.1: Visão geral da plataforma MASA-Webaligner.

Para persistência dos dados fornecidos pelo usuário e para implementação do sistema de escalonamento, o MASA-Webaligner utiliza os bancos de dados PostgreSQL e Redis, respectivamente. Ainda, os bancos não possuem qualquer integração entre si.

Para utilização do banco PostgreSQL, foi utilizada a biblioteca TypeORM, responsável por realizar o mapeamento objeto-relação. A biblioteca possui uma API para facilitar a comunicação com o banco de dados, além de possibilitar que o desenvolvedor também

escreva suas próprias *queries*, caso seja necessário. Apesar da camada de abstração fornecida pela biblioteca, o desenvolvedor tem total controle sobre a utilização do banco de dados.

Já no caso do banco Redis, o banco é utilizado pela biblioteca Bull e indiretamente pelo MASA-Webaligner. Assim sendo, o desenvolvedor, a princípio, não tem controle total sobre o banco, tendo seu acesso mediado por meio da API disponibilizada pelo Bull. Caso seja necessária a utilização do banco Redis para outro propósito além da fila, é necessária a utilização de uma nova biblioteca que permita, ao desenvolvedor, controle sobre o banco.

A linguagem da plataforma MASA-Webaligner era, inicialmente, o JavaScript. A linguagem, embora muito popular para desenvolvimento WEB, possui certas limitações para projetos de maior porte. Essas limitações devem-se ao fato de que o JavaScript é uma linguagem com tipagem dinâmica e, quando combinada com o interpretador Node.JS [35] - utilizado no desenvolvimento deste projeto -, não tem acesso às funcionalidades mais recentes da linguagem sem configurações adicionais. Por isso, foi decidido utilizar o TypeScript [36], que também é utilizável com o Express.JS e React.JS.

O TypeScript é uma linguagem *open-source* construída em JavaScript, que adiciona tipagem e outras funcionalidades ao JavaScript básico [36], o que motivou sua escolha. Com a utilização do TypeScript, torna-se simples aplicar conceitos que facilitam o desenvolvimento, manutenção e evolução do software, sem perder a versatilidade do JavaScript, pois ao realizar a *build* do projeto, o código será transpilado para JavaScript, podendo ser executado em qualquer dispositivo que rode JavaScript [36].

5.1.1 Funcionalidades

O MASA-Webaligner, em sua versão atual, possui onze funcionalidades necessárias para o alinhamento e visualização dos resultados. O usuário pode realizar a requisição de alinhamentos executados pelo MASA-CUDAlign ou MASA-OpenMP, pode optar pela execução apenas do Estágio 1 (Seção 3.2.1) ou dos Estágios 1 a 5 (Seções 3.2.2 e 3.2.5), escolher entre alinhamento local ou global, optar pela remoção dos caracteres "N" das sequências, habilitar ou desabilitar a otimização *Block Pruning* (Seção 3.3.2), substituir os caracteres da sequência, ou sequências, pelos seus complementos e alinhar o inverso da sequência, ou sequências, fornecidas.

Após a conclusão do processo de alinhamento, o usuário tem acesso aos resultados gerados. Dependendo dos estágios executados, os resultados apresentam as estatísticas do alinhamento, o melhor *score* e sua posição, ou o gráfico do alinhamento e a saída textual construídas pelo MASA-Viewer.

O processamento dos alinhamentos requisitados pode demorar de segundos a horas para ser concluído (dependendo do tamanho das sequências sendo analisadas). Para evitar concorrência no uso de recursos - o que aumentaria ainda mais o tempo de processamento - foi utilizada uma estratégia de enfileiramento de requisições de alinhamentos do tipo *First In First Out* (FIFO), implementada com auxílio da biblioteca Bull [37], que será abordada em maior detalhe na Seção 5.2.

Devido à quantidade de tempo que um determinado alinhamento pode levar para ser concluído, pode não ser viável fazer com que o usuário espere com o navegador aberto, enquanto o servidor processa a requisição. Dessa forma, o usuário pode, no momento da requisição, informar seu nome e e-mail para que, quando o processamento do alinhamento fique pronto, ele receba uma notificação com o endereço da página para visualização dos resultados.

5.1.2 Arquitetura de Software

O MASA-Webaligner foi construído utilizando o paradigma de Orientação a Objetos (OO). O uso de conceitos e princípios de OO foram alguns dos motivadores pela escolha do TypeScript como linguagem para o desenvolvimento da plataforma.

Dentre os princípios que foram amplamente utilizados no desenvolvimento estão o Princípio da Substituição de Liskov e o Princípio da Inversão de Dependência [38].

O Princípio de Substituição de Liskov diz que se, para cada objeto $o1$ do tipo S, houver um objeto $o2$ do tipo T, de modo que, para todo programa P definido em termos de T, o comportamento de P não seja modificado quando $o2$ for substituído por $o1$, então S é um subtipo de T [38]. Por exemplo, admita um sistema de cobrança de um software que pode ter licença do tipo pessoal ou empresarial. Para gerar o boleto, o sistema interage com a classe Licença e não com a as classes Pessoal ou Empresarial [38].

O Princípio de Inversão de Dependência determina que os sistemas mais flexíveis são aqueles em que as dependências de código-fonte se referem apenas a abstrações e não a itens concretos [38]. Para que isso seja alcançado, devem-se seguir algumas práticas: (a) Não fazer referência a classes concretas voláteis; (b) Não derivar de classes concretas voláteis; (c) Não sobrescrever funções concretas; e (d) Nunca mencionar o nome de algo concreto e volátil.

Para que essas práticas sejam possíveis, devemos utilizar o padrão de *Factories* [38]. Suponha uma aplicação que executa algum serviço específico. Esse serviço depende de uma dada implementação, que deve ser disponibilizada ao serviço. Para obter a instância sem realizar a referência à implementação, cria-se uma interface a qual irá implementar um método que cria a instância necessária e a retorna para o serviço, que solicita a execução

do código. O Princípio de Inversão de Dependência foi implementado com auxílio da biblioteca Tsyringe [39], que será abordada na Seção 5.2.

O MASA-Webaligner foi construído utilizando quatro partes fundamentais: o *front-end*, que lida diretamente com o usuário, provendo a interface para requisição de alinhamentos e para construção e visualização dos resultados dos alinhamentos requisitados; o *back-end* que recebe e processa as requisições de alinhamentos, além de retornar os dados necessários para a construção dos resultados no *front-end*; os *middlewares* [40], que estão contidos no *back-end* e interceptam as requisições de alinhamentos, verificando se os dados fornecidos são válidos e permitindo, caso os dados sejam válidos, o processamento das requisições pelo *back-end*, e carregam possíveis arquivos de sequências sendo enviados para alinhamento; e os serviços externos, que são responsáveis pelo eventual *download* de sequências FASTA e envio de e-mails. A Figura 5.2 mostra a organização e a comunicação entre as partes do MASA-Webaligner.

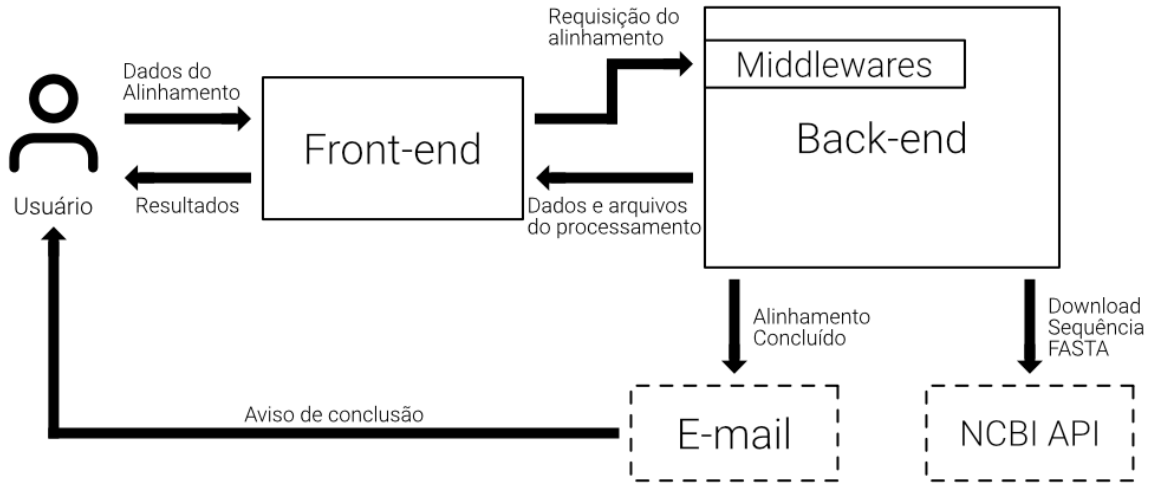


Figura 5.2: Módulos da plataforma MASA-Webaligner.

5.2 *Back-end* da aplicação

Existem diversos *frameworks* para o desenvolvimento do *back-end* de aplicações, que utilizam o Node.JS. O Express.JS é um *framework* para aplicativo da web do Node.JS mínimo e flexível que fornece um conjunto de recursos para aplicativos web e móvel [32].

O Express.JS é um *framework unopinionated*, ou seja, não há uma forma ou estrutura pré-definida sobre como realizar o desenvolvimento de aplicações. Cabe ao desenvolvedor pensar todos os detalhes do sistema. Em projetos de grande porte, *unopinionated*

frameworks podem ser um problema para o desenvolvimento, porém no caso do MASA-Webaligner, que não possui uma grande base de código, o Express.JS é perfeitamente aceitável.

Para processar requisições ao *back-end*, o Express.JS utiliza funções que recebem como argumento o endereço de roteamento e uma função com o seguintes argumentos: um objeto contendo os dados da requisição (Request), um objeto de resposta (*Response*) e uma função (*NextFunction*) que pode ser chamada para ser executada na sequência. Ao acessar o endereço de roteamento, o Express.JS executa a função passada como argumento.

Para conectar todos os módulos do servidor, utilizou-se a biblioteca Tsyringe [39], que é *open-source* e mantida pela Microsoft. A biblioteca permite a aplicação dos princípios de Substituição de Liskov e Inversão de Dependência (Seção 5.1.2).

Toda regra de negócio do sistema foi quebrada em serviços, que podem depender de diferentes classes de diferentes módulos. Ao invés de resolver a dependência por meio de referência à implementação concreta, o Tsyringe permite resolver as dependências por meio de injeções. Para isso, definimos interfaces para a utilização das classes dos módulos, que as implementam de acordo, e realizamos o registro das implementações no contêiner do Tsyringe.

Cada serviço, ao ser instanciado, solicita uma injeção de dependência - por quaisquer módulos necessários - ao contêiner do Tsyringe, que realiza a injeção em tempo de execução. A Figura 5.3 mostra como é definido o construtor do serviço *GetFileName* da plataforma MASA-Webaligner.

```
23  @injectable()
24  export default class GetFileNameService {
25    constructor(
26      @inject('StorageProvider')
27      private storageProvider: IStorageProvider,
28
29      @inject('SequenceFilesProvider')
30      private sequeceFiles: ISequenceFilesProvider,
31    ) {}
```

Figura 5.3: Construtor do serviço GetFileNameService responsável por obter, verificar e salvar as sequências FASTA fornecidas. O serviço está sendo injetado com as dependências dos módulos *Storage* e *NCBI*.

O *back-end* foi dividido em dez módulos, que são: *Middlewares, Alignments, Files, Storage, National Center for Biotechnology Information (NCBI), Object-Relational Mapping (ORM), Queue, Aligner, Mail* e *Mail Template*. A Figura 5.4 apresenta a organização do *back-end* do MASA-Webaligner.

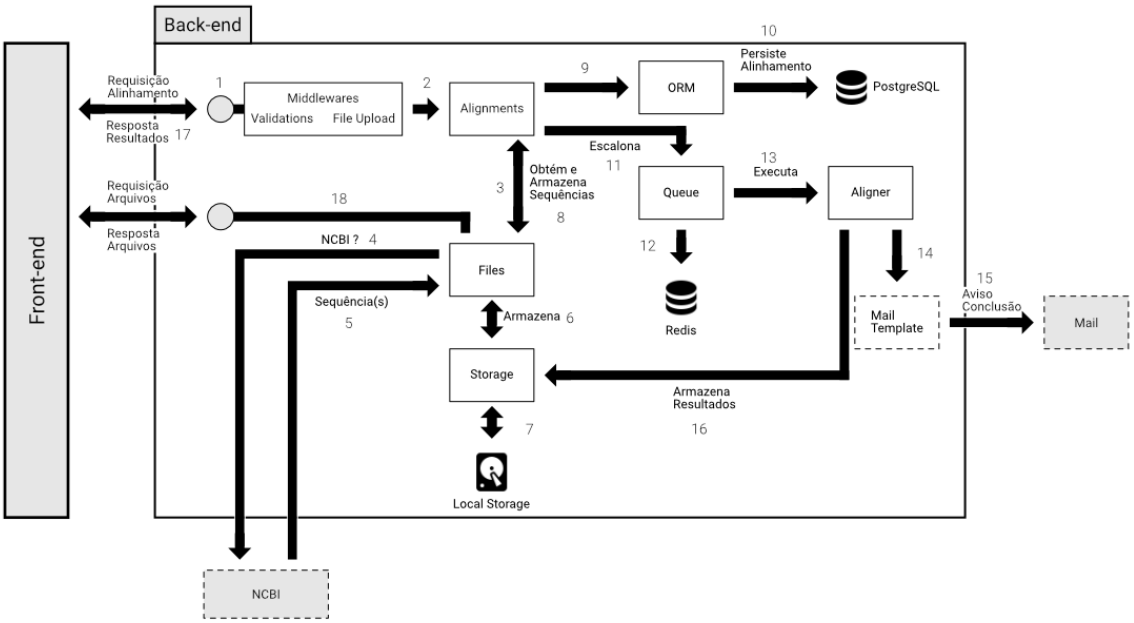


Figura 5.4: Visão do *back-end* da aplicação MASA-Webaligner.

5.2.1 *Middlewares*

No Express.JS, *middlewares* são funções que interceptam a requisição HTTP e realizam algum processamento ou validação, antes que as regras de negócio sejam executadas e gerem uma resposta. Os *middlewares* possuem os mesmos parâmetros - *Request, Response, NextFunction* - das funções de processamento das requisições. No MASA-Webaligner, foram utilizados dois *middlewares*: *Validations* e *File Upload*.

Validations

O *middleware* de Validations é construído utilizando a biblioteca Celebrate [41]. Ela permite a validação de rota, e garante que todos os *inputs* estão corretos antes de qualquer processamento do servidor [41]. Dessa forma, quando uma requisição de alinhamento chega do *front-end*, o Celebrate verifica se todos os campos estão corretos e, caso haja alguma problema, retorna um erro ao *front-end*, evitando qualquer processamento desnecessário pelo *back-end*.

File Upload

O *File Upload* permite que arquivos sejam carregados no servidor por meio da biblioteca Multer. Quando o usuário faz a requisição de um alinhamento e opta por carregar um ar-

quivo, o Multer [42] se encarrega do armazenamento temporário para que, posteriormente, o módulo *Storage* mova o arquivo para o local definitivo.

5.2.2 *Alignments*

Ao fazer uma requisição de alinhamento ou do resultado de um alinhamento previamente executado, o módulo *Alignments* é utilizado. Quando um alinhamento é requisitado, após o pré-processamento pelos *Middlewares* (Seção 5.2.1), o módulo *Alignments* aciona o módulo *Files* para processar as sequências fornecidas e obter os nomes dos arquivos FASTA salvos no armazenamento definitivo. Ao receber os nomes dos arquivos, o módulo *Alignments* solicita, ao módulo *Object-Relational Mapping* (ORM), a criação dos dados do alinhamento e das sequências no banco de dados, seleciona a extensão MASA que irá executar o alinhamento, e solicita o escalonamento da execução pelo módulo *Queue*. A requisição de alinhamentos conclui retornando o *id* do alinhamento criado no banco de dados ao *front-end*.

Quando a requisição solicitada é pelo resultado do alinhamento, o pré-processamento não é executado por nenhum dos *Middlewares*. Há apenas a solicitação pelos dados contidos no banco de dados, por meio do módulo *ORM*, e das estatísticas contidas nos arquivos de controles gerados pelo MASA-CUDAlign ou MASA-OpenMP, pelo módulo *Storage*. Após obter as informações, o resultado é retornado ao *front-end*.

5.2.3 *Files*

Assim como o módulo *Alignments* (Seção 5.2.2), o módulo *Files* também possui rotas para comunicação com o *front-end*, além de ser responsável pela obtenção, verificação e solicitação de armazenamento das sequências enviadas pelo usuário.

Ao receber a solicitação de processamento de sequências do módulo *Alignments*, inicia-se o processamento verificando qual tipo de entradas de dados foi escolhido pelo usuário. Existem três possibilidades de entrada: identificador do banco de dados do *National Center for Biotechnology Information* (NCBI API), carregamento de arquivo e inserção manual.

No caso da utilização da NCBI API, o módulo *Files* solicita a utilização do módulo *NCBI* para *download* da sequência fornecida. Não há validação dos dados, pois a utilização do banco de dados da NCBI garante que o arquivo está em um formato FASTA válido. O arquivo baixado é salvo diretamente no armazenamento definitivo.

Caso o usuário escolha carregar um arquivo, o *middleware* de *File Upload* (Seção 5.2.1) carrega o arquivo para o servidor, que é acessado pelo módulo *Files*. É executada, então,

uma validação, que verifica se o arquivo contém uma sequência FASTA válida, e o arquivo é salvo no armazenamento definitivo.

Se o usuário optar por inserir manualmente a sequência, o processo é semelhante com o carregamento de arquivo. Nesse caso, porém, não há arquivo salvo no espaço de armazenamento temporário. A sequência inserida fica contida na memória do servidor, e é validada e salva diretamente no armazenamento definitivo.

Em todos os três casos, o processo finaliza devolvendo o nome do arquivo salvo no armazenamento definitivo para o módulo *Alignments*.

As rotas de comunicação com o *front-end*, contidas no módulo *Files*, permitem a requisição dos arquivos binário, gerado após o processamento do MASA-CUDAlign ou MASA-OpenMP (Seção 3.2.5), e das sequências FASTA fornecidas pelo usuário. Em ambos os casos, o módulo verifica se o alinhamento solicitado está registrado no banco e envia os arquivos para processamento no *front-end*. Caso os arquivos não sejam encontrados, uma exceção é lançada e o servidor retorna um código de erro.

5.2.4 *Storage*

O módulo *Storage* é responsável por todas as operações de armazenamento, carregamento e remoção dos arquivos utilizados pela plataforma MASA-Webaligner.

A interface do módulo define métodos para mover arquivos FASTA do espaço temporário de armazenamento para o espaço definitivo, salvar sequências FASTA em memória no armazenamento definitivo, apagar arquivos FASTA, apagar arquivos de controle MASA, carregar arquivos FASTA, carregar arquivos binários da plataforma MASA e carregar os arquivos de estatísticas do MASA. A implementação atual do módulo realiza todas as operações no disco rígido do próprio servidor.

5.2.5 *National Center for Biotechnology Information* (NCBI)

No caso de solicitar um alinhamento no qual uma ou ambas as sequências devem ser carregadas pela API do *National Center for Biotechnology Information*, o módulo é acionado.

O módulo NCBI possui apenas um método, que é responsável por acessar a API pública do *National Center for Biotechnology Information*, informar o *id* fornecido pelo usuário e realizar o *download* da sequência FASTA.

Um *streaming* de dados ocorre ao solicitar o *download* e os dados são escritos, de maneira assíncrona, no servidor. O servidor, então, bloqueia a execução do método até que o *download* seja concluído, ou lance uma exceção. No primeiro caso, retorna-se o nome do arquivo gerado ao módulo *Files* e no último caso, o arquivo parcial é deletado do servidor e uma nova exceção é gerada para retornar um erro ao *front-end*.

5.2.6 *Object-Relational Mapping* (ORM)

Ao realizar uma requisição - seja para gerar um alinhamento e solicitar resultados ou arquivos - o MASA-Webaligner, por meio da biblioteca TypeORM, acessa o banco de dados PostgreSQL [43] para criar entradas de alinhamentos e sequências, ou verificar se um dado existe no banco.

O MASA-Webaligner utiliza o modelo de repositórios para intermediar o acesso aos dados no banco utilizado. O TypeScript implementa o paradigma de Orientação a Objetos (OO) (Seção 5.1.2), porém o banco de dados utiliza o modelo relacional. Para realizar a comunicação entre a aplicação e o banco é necessário utilizar alguma forma de mapeamento objeto-relação. Para alcançar esse objetivo, foi utilizada a biblioteca TypeORM [44]. O TypeORM é um *Object-Relational Mapper* que executa no Node.JS, Browser e outras plataformas, e pode ser usado com TypeScript e JavaScript (ES5, ES6, ES7 e ES8) [44].

O módulo *Alignments* define entidades - *Alignments* e *Sequences* -, que definem os campos que serão mapeados de um objeto para uma tabela no banco de dados relacional. Cada entidade possui seu próprio repositório, que implementa métodos de busca e criação de dados no banco. No caso da entidade *Alignments*, existe também um método para atualizar a situação do alinhamento. O método é utilizado após a conclusão do alinhamento pelo MASA-CUDAlign ou MASA-OpenMP.

Por último, fazem parte do módulo ORM os arquivos de credenciais para conexão com banco e os arquivos de *migrations*. As *migrations* funcionam como um versionamento do banco de dados. Ao compartilhar as *migrations*, cada membro da equipe de desenvolvedores pode executar um comando e o banco de dados local - usado para desenvolvimento - será atualizado, ou revertido, de acordo com a implementação das *migrations*.

5.2.7 *Queue*

Como visto na Seção 5.1.2, os alinhamentos requisitados são escalonados para execução pelo módulo *Queue*. A fila de escalonamento é construída utilizando o Bull, uma biblioteca Node que implementa uma fila utilizando o Redis [37].

O Redis é um banco NoSQL que utiliza um sistema de chave-valor. Cada alinhamento escalonado tem uma chave e a ela estão associados todos os dados necessários para executar o MASA-CUDAlign ou MASA-OpenMP.

Ao iniciar o processo do servidor, gera-se uma instância da fila, por meio do Bull, e, simultaneamente, inicia-se um segundo processo para dar início ao processamento dos *jobs* contidos na fila. Ainda, são gerados *listeners* para os eventos de conclusão e erro de processamento dos *jobs* na fila.

Ao concluir uma execução, o *listener* aciona o módulo *ORM* para atualizar a situação do alinhamento processado e, caso o usuário tenha fornecido os dados de contato, um e-mail é enviado para avisá-lo da conclusão do alinhamento pelo MASA.

Por fim, o Bull implementa uma página, disponível por um ponto de acesso do *back-end*, que permite a visualização da fila, *jobs* concluídos e abortados, além de informações sobre o consumo de recursos do sistema. A Figura 5.5 mostra a interface disponibilizada pelo Bull para visualização dos dados das filas.

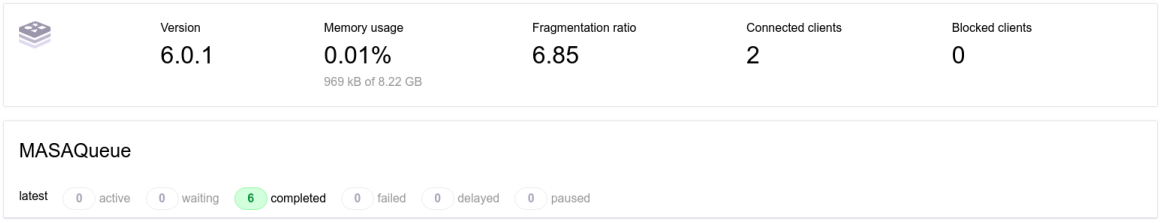


Figura 5.5: Interface de usuário da biblioteca Bull para visualização das filas do MASA-Webaligner.

5.2.8 *Aligner*

O módulo *Aligner* implementa um único método, responsável por criar um processo filho, que irá executar o MASA-CUDAlign e MASA-OpenMP.

A execução da plataforma MASA é feita por Estágios (Seção 3.2). Como há uma ordem na execução - cada Estágio, com exceção do primeiro, depende da conclusão do anterior - as execuções são feitas de maneira síncrona, ou seja, ao criar um processo filho para execução do MASA, a execução do módulo fica bloqueada, aguardando o retorno do processo filho.

O método não possui nenhum retorno, pois após a conclusão da execução, o *listener* do módulo *Queue* realiza as ações necessárias, como visto na Seção 5.2.7.

5.2.9 *Mail* e *Mail Template*

Os últimos módulos, *Mail* e *Mail Template*, serão explicados simultaneamente devido à sua íntima ligação.

Ao requisitar um alinhamento, o usuário pode optar por informar seu nome e e-mail, já que o alinhamento requisitado pode demorar algum tempo até ser concluído. Quando essas informações são fornecidas, após a conclusão do alinhamento, pelo módulo *Aligner*,

o *listener* do módulo *Queue* executa ações para finalizar o *job*. O método *listener*, caso o nome e e-mail tenham sido fornecidos, aciona o módulo *Mail*.

O módulo *Mail* implementa apenas um método, que envia os e-mails de aviso de conclusão de alinhamento para os usuários.

Para enviar um e-mail, é necessário carregar um *template* para o corpo da mensagem. Para realizar essa tarefa, o módulo *Mail* aciona o módulo *Mail Template*. O módulo é implementado com ajuda da biblioteca *Handlebars* [45], que provê o poder necessário para permitir a construção de *templates* semânticos efetivamente.

Os *templates* possuem código HTML, combinado com variáveis - identificadas por meio de chaves duplas -, que permitem adicionar informações em tempo de execução. Os *templates* são carregados para compor o corpo dos e-mails, que, então, são enviados pelo módulo *Mail*. A Figura 5.6 mostra o *template* carregado para envio de e-mails de aviso de conclusão do processamento de alinhamentos.

```
<style>
  .message-content {
    font-family: Arial, Helvetica, sans-serif;
    max-width: 600px;
    font-size: 18px;
    line-height: 21px;
  }
</style>

<div class="message-content">
  <p>Hello, {{full_name}}</p>
  <p>
    We are writing this e-mail to let you know that your request for a
    sequence alignment is ready!
    <br/>
  </p>
  <p>
    To access the results, just click
    <a href="{{address}}">here</a>
    or access the address: {{address}}.
    <br />
  </p>
  <p>
    Regards,
    <br />
  </p>
  <p>MASA Team</p>
</div>
```

Figura 5.6: *Template* escrito com sintaxe compatível com a biblioteca Handlebars.

5.3 *Front-end* da aplicação

Como dito na Seção 5.1, o MASA-Webaligner teve seu *front-end* construído com o *React.JS*. Trata-se de uma biblioteca para desenvolvimento WEB, *open-source*, escrita em JavaScript e mantida pelo Facebook.

O React.JS é baseado na utilização de *States* e Componentes para manipulação dos dados sendo apresentados na interface de usuário. Componentes quebram os vários elementos que compõem a interface de usuário de forma que cada Componente contenha seu próprio *State* interno e contribua para a construção de interfaces complexas [46].

A ideia por trás dos *States* do React.JS é que cada Componente será renderizado novamente ao identificar uma mudança de Estado. Pode-se, então, ter várias variáveis e objetos que determinam o *State* de um dado Componente. Para alterar o *State*, não podemos simplesmente realizar uma atribuição a uma variável ou objeto. Devemos utilizar uma função própria do React.JS, a função *setState*.

A função *setState* define um novo *State* ao Componente do React.JS. É importante ressaltar que a função tem comportamento assíncrono, pois o React.JS pode agrupar várias chamadas do *setState* por questões de desempenho [47].

Na linguagem HTML, as *tags* possuem atributos, que são utilizados pelo CSS e JavaScript, para identificá-las, estilizá-las ou gerar comportamentos dinâmicos na interface de usuário. No React.JS não exitem atributos, mas existem os chamados Props. Esses são como parâmetros de funções e o React.JS possui apenas uma regra para os Props: todos os Componentes React têm que agir como funções puras em relação aos seus Props [46], ou seja, os Componentes tratam seus Props como parâmetros somente leitura.

O *hook* é outro conceito muito importante no React.JS. *Hooks* permitem reutilizar lógica com estado sem mudar sua hierarquia de componentes [48]. Efetivamente, *hooks* permitem a criação de funcionalidades que podem ser disponibilizadas por toda a aplicação. O React.JS já possui alguns *hooks* nativos como, por exemplo, a função *setState*.

O *front-end* da plataforma possui uma organização bastante simples, composta por três páginas - *Home*, MASA-Aligner e Resultados -, a Figura 5.7 mostra a organização do *front-end*. A *Home* não possui impacto real na aplicação, pois apenas provê algumas informações básicas ao usuário, além de direcioná-lo para a página de requisição de alinhamentos (MASA-Aligner) e para o repositório do projeto no Github [49].

5.3.1 MASA-Aligner

A página de requisição de alinhamentos é composta por um simples formulário, que contém todas as configurações para um alinhamento, como descrito na Seção 5.1.1.

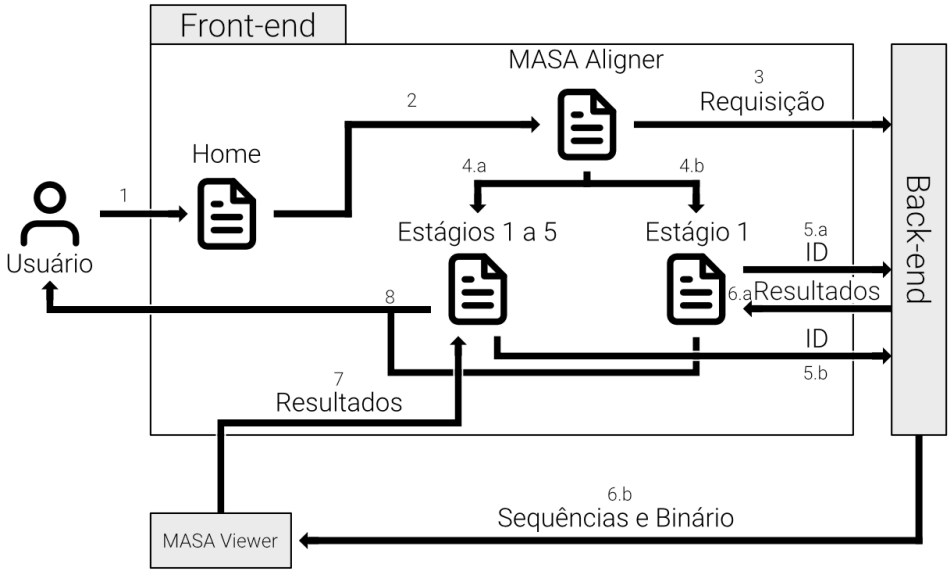


Figura 5.7: Visão do *front-end* da aplicação MASA-Webaligner.

Ao carregar a página, todos os campos do formulário têm seu *State* inicial definido e, a cada mudança gerada pelo *input* do usuário, seu *State* é atualizado. Os campos de *input* foram divididos em cinco Componentes, que são repetidos ao longo de todo o formulário: Button para renderização de botões; RadioInput para renderização de caixas de seleção; TextAreaInput para renderização de caixas de textos longos; TextInput para renderização de caixas de textos curtos; e UploadInput para renderização de campos de carregamentos de arquivos. A Figura 5.8 mostra os Componentes utilizados na página de requisição de alinhamentos.

Ao preencher o formulário com os dados, obrigatórios e opcionais, o usuário deve clicar no botão "*Align Sequences*". O *front-end* irá, então, realizar uma validação dos dados inseridos no formulário, por meio da biblioteca Yup [50]. A Yup determina os campos que são optativos e obrigatórios, e verifica se os últimos foram preenchidos. No caso do campo optativo "e-mail", ainda é verificado se esse é válido.

No caso de erros de validação no *front-end* ou erros de processamento no *back-end*, o *front-end* aciona o *hook*, desenvolvido para sinalizar erros, avisos e informações gerais, *addToast*.

Toasts são elementos gráficos renderizados com uma animação semelhante a uma torrada ficando pronta em uma torradeira. O *addToast* renderiza um *Toast* contendo algumas informações sobre o erro que foi gerado, para que o usuário tenha condições de ajustar os

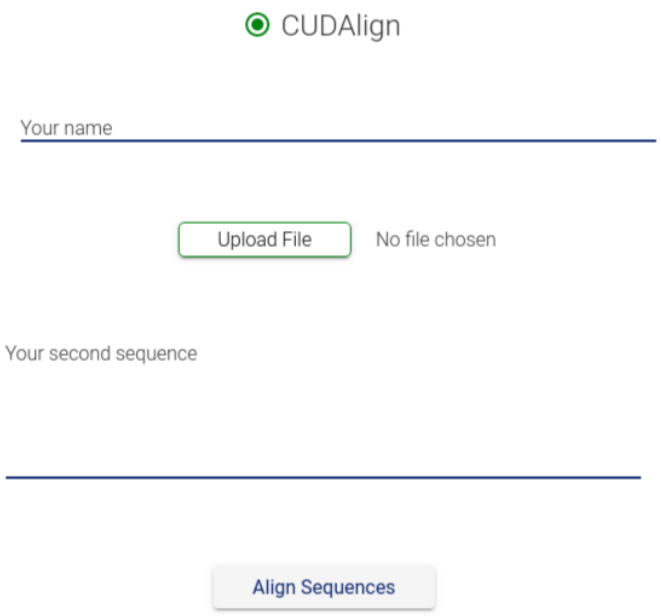


Figura 5.8: Componentes da plataforma MASA-Webaligner, em ordem: RadioInput, TextInput, UploadInput, TextAreaInput e Button.

inputs do formulário e tentar uma nova requisição, ou em casos de persistência dos erros, possa entrar em contato com os desenvolvedores. A Figura 5.9 mostra um *Toast* de erro renderizado pelo *hook*.

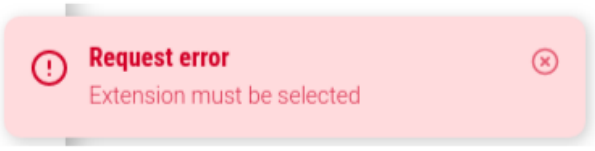


Figura 5.9: Resultado da execução do *hook addToast*, gerando um *Toast* de erro na interface do usuário.

Após a conclusão das validações, caso nenhum erro tenha ocorrido, o *front-end* faz

a requisição do alinhamento ao *back-end* e, caso o *back-end* não retorne erros, o *id* do alinhamento requisitado é recebido e o usuário é direcionado para a página Resultados.

5.3.2 Resultados

A página Resultados possui dois caminhos de renderização, pois a requisição pode ser para o processamento apenas do Estágio 1 (Seção 3.2.1) ou dos Estágios 1 a 5 (Seções 3.2.2 e 3.2.5). No entanto, em ambos os casos, há um passo inicial comum. A página contém um total de 11 variáveis de *State* que controlam quais e como serão exibidas as informações resultantes da requisição de alinhamento, listadas abaixo:

- *tries*: Variável que controla o número de requisições feitas ao *back-end* enquanto o resultado do processamento do alinhamento não está pronto;
- *isToggled*: A variável apresenta funcionalidade apenas para alinhamentos que executam os Estágios 1 a 5. Quando seu *State* é *true*, torna visível a caixa de conteúdo lateral com dados estatísticos do alinhamento, quando "false" faz com que a caixa fique escondida;
- *alignmentData*: Armazena todos os dados processados pelo MASA-Viewer. Seu *State* contém os dados que serão utilizados para construção do gráfico e da saída textual para alinhamentos que executam todos os Estágios;
- *alignmentInfo*: Armazena os dados obtidos do *back-end*. Para alinhamentos que executam apenas o Estágio 1, seu *State* armazena os dados base do alinhamento, das sequências e das estatísticas. Para os demais alinhamentos, além dessas informações, armazena também os bytes do arquivo ".bin", que contém os resultados do alinhamento, e o conteúdo dos arquivos ".fasta";
- *alignmentText*: Após o processamento do MASA-Viewer, armazena a saída textual dos alinhamentos;
- *renderGraph*: Após inserir os *gaps* nas sequências, se o tamanho final for igual ou maior 1MB, determina - com *true* ou *false* - se o gráfico deve ser renderizado;
- *graph*: Com os resultados lidos pelo MASA-Viewer, a variável armazena os valores das coordenadas (x, y), que serão utilizadas para montar o gráfico do alinhamento;
- *resetValues*: Para que a saída textual possa retornar ao seu estado inicial após ajustes pelo usuário, os valores de início e fim da sequência são armazenados nesta variável;

- *min* e *max*: Armazenam os valores digitados pelo usuário para ajuste na saída textual. Representam o limite inferior e superior respectivamente;
- *render*: Determina qual o formato da página que será renderizada. Seu *State* indica a tela de "loading" (0), resultados Estágio 1 (1) e resultados Estágios 1 a 5 (2);
- *errors*: Determina se os resultados ou uma tela de erro devem ser renderizadas.

Início da renderização

Quando o usuário acessa a página Resultados, o endereço, na barra do navegador, contém o *id* do alinhamento requisitado. Esse *id* é utilizado pelo *front-end* para realizar uma requisição ao *back-end* pelos dados do alinhamento.

Para que a requisição seja feita de forma automática, ao carregar a página, utiliza-se um *hook* nativo do React.JS, o *useEffect*. O *hook* aceita uma função e um *array* de dependências, de forma que a função é executada assim que o componente inicia o processo de renderização e toda vez que qualquer item do *array* sofra uma mudança de *State*. A resposta do *back-end* pode conter, além dos dados básicos do alinhamento, dois erros. O primeiro erro contém um código customizado, 452, que determina que o processamento do alinhamento ainda não foi concluído, e o segundo erro, 404, que indica que o *id* fornecido não corresponde a nenhum alinhamento cadastrado no sistema.

No caso do erro 452, o *useEffect* atualiza o *State* da variável que controla o número de tentativas de obter as informações do alinhamento pelo *back-end*, *tries*. A variável *tries* sofre um incremento (*setState*) a cada erro 452 e, como faz parte do *array* de dependências do *useEffect*, implica em uma nova execução do *hook*. O valor da variável *tries* determina o tempo até a próxima execução. Os intervalos aumentam a uma velocidade de 5 segundos, ou seja, a segunda tentativa ocorre após 5 segundos, a terceira após 10 segundos e assim sucessivamente. Esse processo irá se repetir até que o servidor retorne o alinhamento ou o usuário feche a página.

O erro 404 implica em fim do ciclo de requisição. O usuário será apresentado a uma mensagem de erro, como mostrado na Figura 5.10, e deverá tentar outro *id* ou pedir uma nova requisição de alinhamento.

Renderização Estágio 1

Para requisições que executam apenas o Estágio 1 (Seção 3.2.1), após o retorno do *back-end*, o processo de renderização atualiza o *State* "*render*" para o valor "1" e conclui a renderização, disponibilizando 2 *cards*, que contêm as estatísticas globais do alinhamento e os resultados do Estágio 1: melhor *score* e posição do melhor *score* na matriz de similaridade. A Figura 5.11 mostra o resultado final da renderização do Estágio 1.

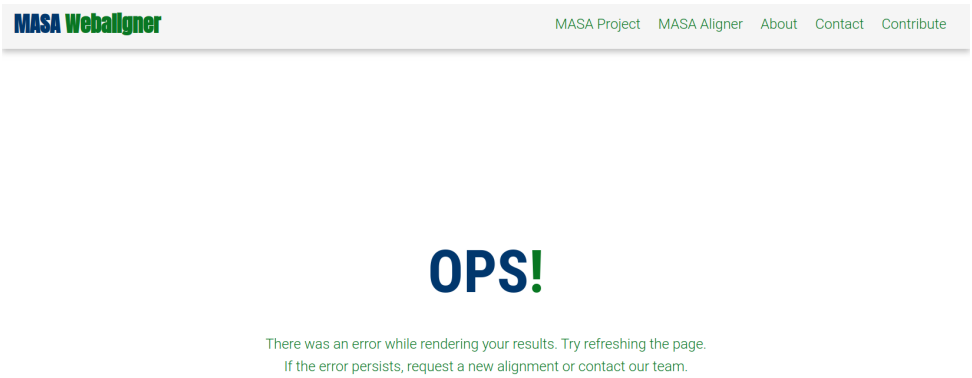


Figura 5.10: Mensagem de erro apresentada ao usuário, caso o *back-end* não responda a requisição como esperado.

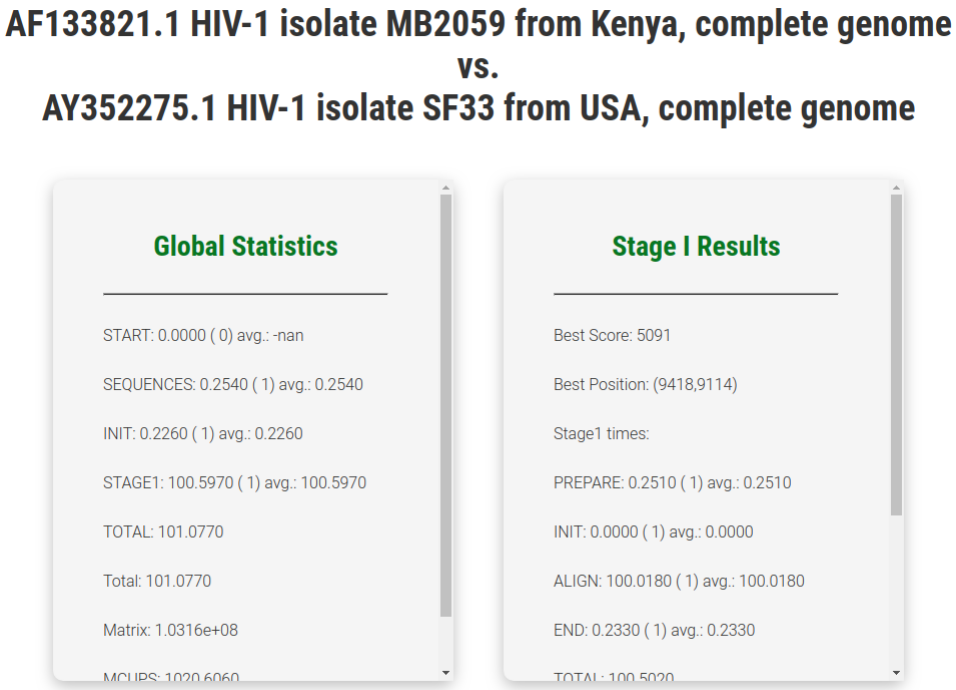


Figura 5.11: Resultado final do processo de renderização dos resultados de requisição de alinhamentos com execução somente do Estágio 1.

Renderização Estágios 1 a 5

Diferente das requisições de alinhamentos que executam apenas o Estaǵio 1, após a resposta inicial do *back-end*, o *front-end* realiza mais duas requisições, pelo arquivo ".bin"

e pelos arquivos ".fasta" respectivamente. Após o recebimento dos arquivos, a variável "*alignmentInfo*" tem seu *State* atualizado.

Com a atualização do *State*, tem-se início a execução do *hook useEffect*, que tem a variável "*alignmentInfo*" na sua lista de dependências. Com a execução deste *hook*, o MASA-Viewer processa o arquivo ".bin" e obtém todos os dados resultantes do alinhamento, atualizando, ao final, o *State* da variável "*alignmentData*", dando continuidade ao ciclo de renderização e ativando a execução de mais um *hook useEffect*.

Com o início da execução do *useEffect* dependente da variável "*alignmentData*", o *front-end* já possui todos os dados necessários para gerar as coordenadas. Com isso, chama-se uma função, "*buildGraphData*", que compara as duas sequências - com os *gaps* já inseridos. Para cada caractere válido (A, G, C, T e N), o valor da coordenada é incrementado em uma unidade, e para cada caractere inválido (-) o valor da coordenada se mantém igual. Caso o tamanho das sequências ultrapasse 1MB, o *State* de "*renderGraph*" é a atualizado para *false* e o gráfico não será renderizado. O valor inicial é definido pelos resultados obtidos do alinhamento e, portanto, varia de acordo com o tipo do alinhamento (global ou local).

Por fim, utiliza-se instâncias das classes TextChunk e TextChunkSum - pertencentes ao MASA-Viewer - para gerar a saída textual com base nos dados obtidos do alinhamento. Ao fim desse processo, os *States* das variáveis "*alignmentText*" e "*render*" são atualizados para conter a saída textual e o valor "2" respectivamente.

O último passo da renderização é iniciado com a execução do último *useEffect*, que tem as variáveis "render" e "alignmentText" em sua lista de dependências. Sua única ação é inserir os resultados textuais no HTML da página, encerrando o ciclo de renderização. As Figuras 5.12 e 5.13 mostram os resultados finais do processo de renderização, com e sem gráfico.

Como dito anteriormente, a página de resultados possui uma variável de *State* "isToggled", que controla a exibição das estatísticas dos alinhamentos que executam os Estágios 1 a 5. Ao clicar no botão "More Information", o *State* da variável é alterado para o inverso de seu valor atual, ou seja, caso seja "true" vira "false" e vice-versa. A Figura 5.14 mostra o efeito, na interface gráfica, dos *States* "true" e "false" da variável "isToggled".

5.4 Instalação

O MASA-Webaligner foi desenvolvido para rodar em ambiente Linux, assim como a plataforma MASA. Para instalação do sistema, é necessária a instalação do MASA-CUDAlign [51], MASA-OpenMP [52] e Node.js. Além disso, para os bancos de dados utilizados no sistema - PostgreSQL e Redis -, o usuário tem duas opções: realizar a instalação padrão

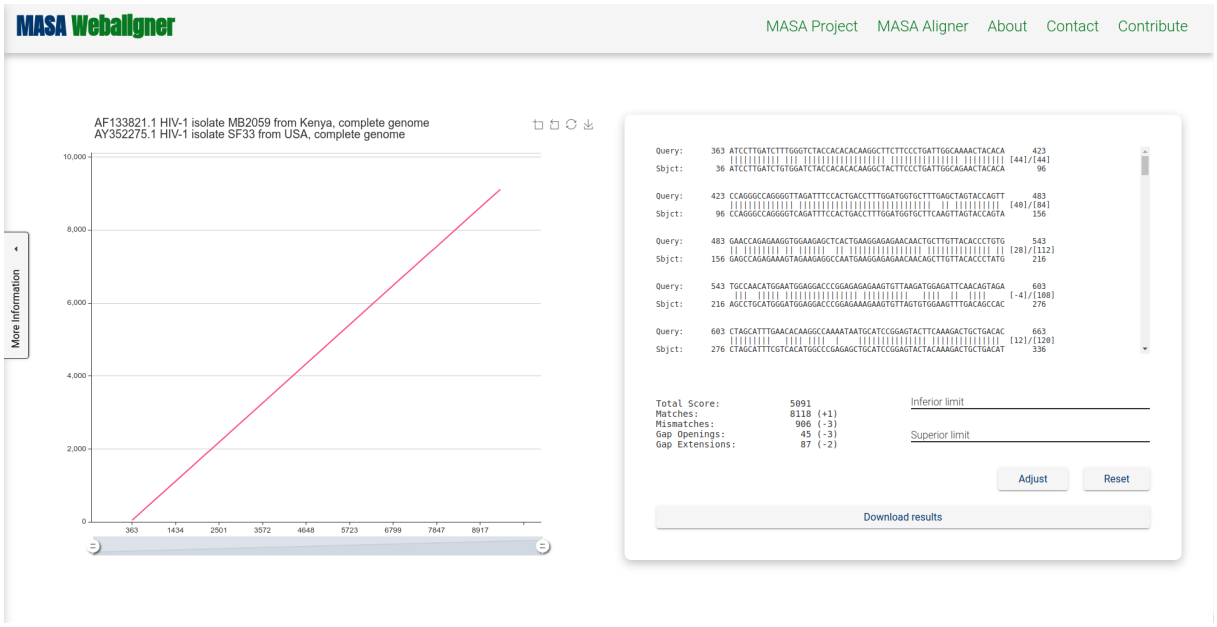


Figura 5.12: Resultado final do processo de renderização dos resultados de requisição de alinhamentos com execução dos Estágios 1 a 5 para alinhamentos menores que 1MB.

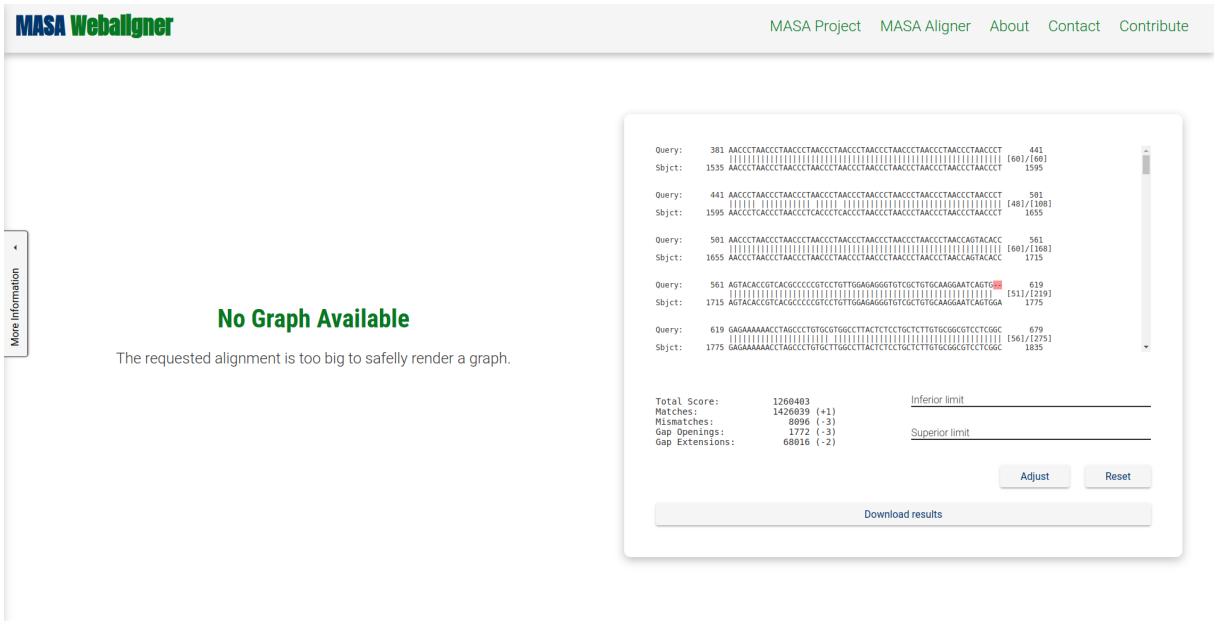


Figura 5.13: Resultado final do processo de renderização dos resultados de requisição de alinhamentos com execução dos Estágios 1 a 5 para alinhamentos maiores que 1MB.

do banco de dados ou utilizar um contêiner (Docker) para instalação isolada dos bancos de dados.

Para realizar a instalação do MASA-CUDAlign, o usuário deve se dirigir ao repositório do projeto no Github [51], realizar o *download* da última versão do *software*, abrir o

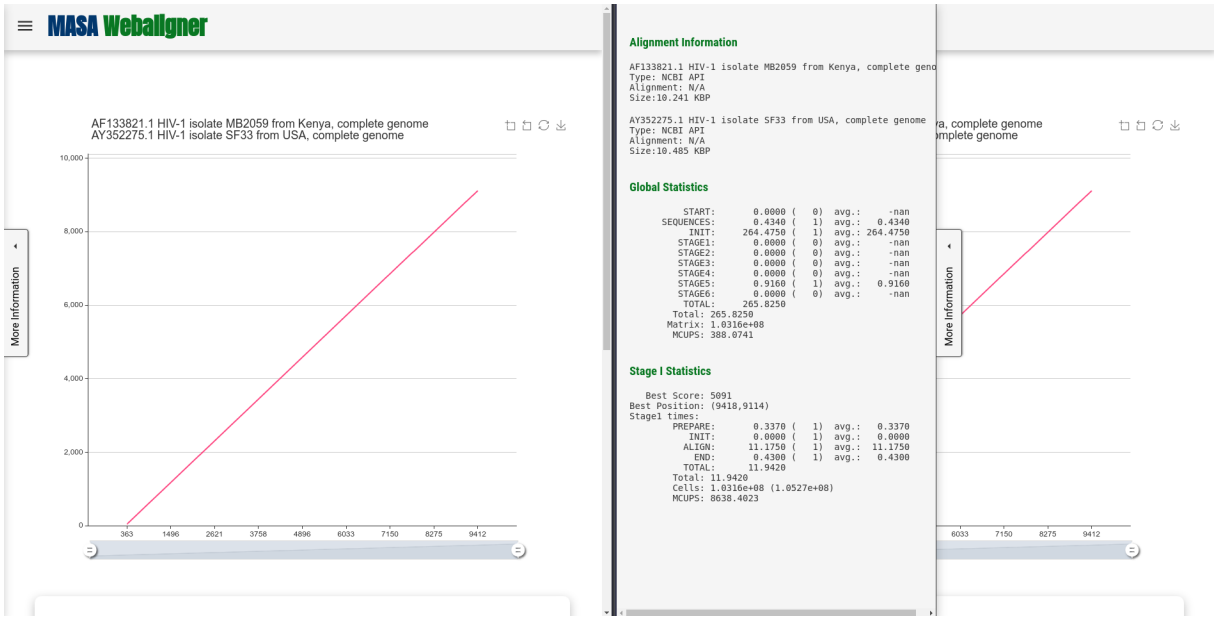


Figura 5.14: À esquerda, o elemento gráfico contendo as estatísticas geradas após o processamento do alinhamento está escondido, o que implica em um valor *false* para variável "isToggled". À direita, o elemento está visível, implicando um valor *true* para variável "isToggled".

terminal Linux na pasta em que o arquivo foi salvo e executar os seguintes comandos:

```
tar -xvzf masa-cudalign-3.9.1.1024.tar.gz
cd masa-cudalign-3.9.1.1024
./configure
make
sudo make install
```

O processo de instalação do MASA-OpenMP é bastante semelhante. O usuário deve acessar o repositório do projeto no Github [52], realizar o *download* da última versão do *software*, abrir o terminal Linux na pasta em que o arquivo foi salvo e executar os comandos:

```
tar -xvzf masa-openmp-1.0.1.1024.tar.gz
cd masa-openmp-1.0.1.1024
./configure
make
sudo make install
```

Para verificar que as instalações do MASA-CUDAlign e MASA-OpenMP tiveram êxito, basta executar os comandos "cudalign" e "masa-openmp". Se não houver problemas na instalação, o usuário deverá ver as mensagens, conforme a Figura 5.15.

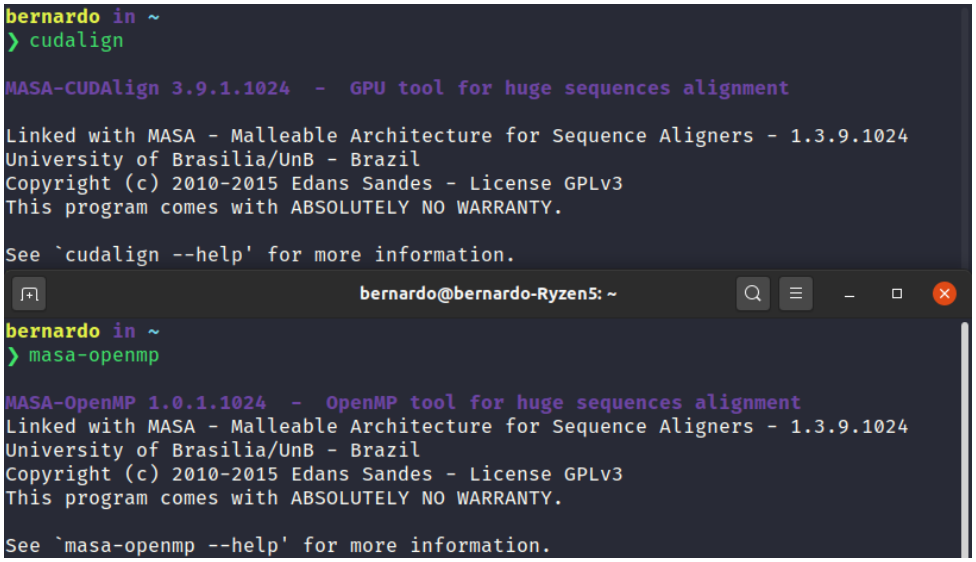


Figura 5.15: Execução e retorno dos comandos para verificação da instalação dos softwares de alinhamento MASA-CUDAlign e MASA-OpenMP.

As demais instalações necessárias podem variar de acordo com a distribuição Linux e, por simplicidade, serão abordados apenas os processos de instalação em distribuições baseadas no Debian.

Com a instalação dos *softwares* MASA concluída, o usuário deve seguir para o site do Node.js [35]. Existem algumas formas de realizar a instalação do Node.js, que podem variar entre distribuições Linux e até mesmo dentro da mesma distribuição. Por simplicidade, o método utilizado será por meio do gerenciador de pacotes, utilizado no desenvolvimento do MASA-Webaligner. Para instalar a versão *Long-Term Support*(LTS) do Node.js, o usuário deve executar os seguintes comandos:

```
curl -sL https://deb.nodesource.com/setup_12.x | sudo -E bash -
sudo apt-get install -y nodejs
```

Os comandos realizam a inserção do repositório Node.js ao sistema do usuário e a instalação do Node.js e NPM (*Node Package Manager*). Após a conclusão do processo de instalação, o usuário pode executar os comandos "node –version" e "npm –version" e, no êxito da instalação, deverá ver as versões dos *softwares* instalados.

Por facilitar o processo, a instalação dos bancos de dados será apresentada pelo método de contêiner, utilizando o Docker. Inicialmente, é necessário realizar a instalação do Docker. Para realizar a instalação, o usuário deve executar os seguintes comandos:

```
sudo apt-get update
sudo apt-get install \
```

```
     apt-transport-https \
     ca-certificates \
     curl \
     gnupg-agent \
     software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg
    | sudo apt-key add -
sudo add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
   $(lsb\_release -cs) \
   stable"
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io
sudo groupadd docker
sudo usermod -aG docker $USER
```

Os comandos listados irão realizar o *download* das dependências do Docker, *download* da chave pública, adicionar o repositório do Docker aos repositórios do sistema, realizar a instalação e adicionar privilégios de administrador ao Docker. Após a instalação, o usuário pode executar o comando "docker –version" para confirmar se o *software* foi instalado. Em caso de êxito, o usuário deverá ver a versão do *software* como resultado.

Após a instalação do Docker, o usuário deve realizar a instalação das imagens do PostgreSQL e Redis. Para isso, deve executar os seguintes comandos:

```
docker run --name some-postgres -e
    POSTGRES_PASSWORD=mysecretpassword -p port:port -d postgres
docker run --name some-redis -p port:port -d redis:alpine
```

Os parâmetros "some-postgres", "mysecretpassword", "port" e "some-redis" devem ser substituídos pelo nome do contêiner PostgreSQL, senha do banco de dados PostgreSQL, portas para direcionar o acesso do sistema operacional para o contêiner e nome do contêiner Redis respectivamente. Após a instalação dos bancos de dados, o usuário pode executar o comando "docker ps -a" para verificar se os contêineres estão executando.

Após a instalação dos bancos de dados, o usuário deve baixar o MASA-Webaligner [53]. Para realizar a instalação, o usuário deve executar o seguintes comandos:

```
git clone https://github.com/bernas1104/MASA-Webaligner
cd MASA-Webaligner
cd backend
```

```
npm i
cd ..
cd frontend
npm i
```

Os comandos listados irão baixar os arquivos e pastas do repositório do projeto MASA-Webaligner, instalar todas as dependências do *back-end* e instalar todas as dependências do *front-end*. Após a conclusão da instalação das dependências, o usuário deve acessar a pasta "backend" e editar o arquivo "ormconfig.json", adicionando o endereço do banco de dados, o usuário e login e nome do banco. Ainda, deve criar um arquivo ".env" e adicionar as seguintes linhas:

```
REDIS_HOST=endereço_banco_redis
REDIS_PORT=porta_banco_redis
```

Substituindo "endereço_banco_redis" e "porta_banco_redis" de acordo. O usuário pode executar o comando "npm run build" para realizar uma *build* do projeto e, com isso, a instalação do *back-end* da aplicação está concluída. O código de produção, gerado pela *build*, está localizado na pasta "dist" e, para dar início à execução do MASA-Webaligner, basta executar os comandos "node dist/shared/infra/http/server.js", para iniciar o servidor, e "node dist/shared/infra/http/queue.js", para iniciar o serviço de fila.

Para concluir a instalação do *front-end* da aplicação, o usuário deve acessar a pasta "frontend" e editar o arquivo ".env", substituindo o valor de "REACT_APP_API_URL" pelo endereço do *back-end*. A Figura 5.16 destaca a linha que deve ser alterada no arquivo ".env".

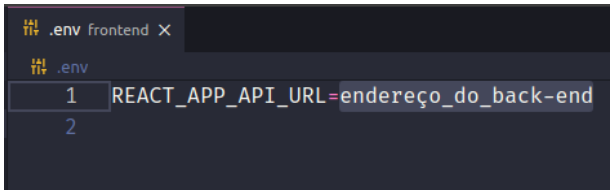


Figura 5.16: Edição do arquivo ".env". A linha em destaque deve ser alterada para conter o endereço do *back-end* da aplicação MASA-Webaligner.

Após a edição, o usuário deve voltar na pasta "frontend" e executar o comando "npm run build". Ao finalizar o processo de *build*, a instalação do MASA-Webaligner está concluída e o código de produção estará na pasta "build".

Para acessar as páginas *web* do *front-end*, o usuário necessita de um servidor *web*, como o NGINX [54] (para uso local) ou algum serviço dedicado, como Netlify [55]. No segundo

caso, o usuário deverá dar preferência por hospedar o *back-end* da aplicação em algum serviço dedicado, como a DigitalOcean [56]. A configuração dos ambientes de *back-end* dedicado e *web*, seja local ou dedicado, estão além do escopo deste projeto e, portanto, não serão abordadas.

Capítulo 6

Resultados

Após a implementação do MASA-Webaligner, foram realizados testes para verificar como o sistema se comporta quando houver múltiplas requisições simultâneas, e se a visualização de alinhamentos de sequências médias e grandes ($\geq$ 1MB) possui impacto no desempenho. Para as requisições simultâneas, os testes foram feitos considerando a execução dos Estágios 1 a 5 (Seções 3.2.2 a 3.2.5), pois a realização de todos os Estágios implica no "pior caso" tanto para o servidor, quanto para o cliente.

Na Seção 6.1 são descritos os ambientes de testes. A Seção 6.2 abordará os testes executados e seus resultados.

6.1 Ambiente de testes

Para execução dos testes da plataforma foram utilizados três computadores pessoais, que irão atuar como servidor e clientes respectivamente. Os computadores foram conectados a uma rede pessoal, onde o servidor possui uma conexão Ethernet de 1 Gbps e os clientes possuem conexão Wi-Fi com a rede. A velocidade da Internet disponível para a rede é de aproximadamente 240Mbps. A Tabela 6.1 apresenta a especificação dos computadores utilizados.

Além disso, foram utilizados três pares de sequências, que tipificam cargas pequenas, médias e grandes. Para as cargas pequenas foram utilizadas as sequências *Severe acute respiratory syndrome coronavirus 2 isolate SARS-CoV-2/human/BRA/SP02/2020*[57] (MT126808.1) e *Severe acute respiratory syndrome coronavirus 2 isolate Wuhan-Hu-1*[58] (NC_045512.2), que possuem tamanho aproximado de 30KB. Para cargas médias foram utilizadas as sequências *Chlamydia trachomatis A/HAR-13*[25] (CP000051.1) e *Chlamydia muridarum Nigg*[26] (AE002160.2), que possuem aproximadamente 1MB de tamanho. Por fim, para as cargas grandes foram utilizadas as sequências *Bacillus anthracis str. Ames*[3] (AE016879.1) e *Bacillus anthracis str. Sterne* [4] (AE017225.1), que pos-

Tabela 6.1: Especificação das plataformas de *hardware* e *software* utilizadas nos testes do MASA-Webaligner.

*	**Servidor**	**Cliente 1**	**Cliente 2**
Sistema Operacional	Ubuntu 20.04 LTS	Pop!_OS 20.04 LTS	Windows 10 Home Edition, 1909
CPU	AMD Ryzen 5 2600, 6-Cores 12-Threads @ 3.40 Ghz	Intel Core i5-8250u, 4-Cores 8-Threads @ 1.60 Ghz	Intel Core i3-7020u, 2-Core 4-Threads @ 2.40 Ghz
GPU	GTX 1050Ti, 768 CUDA Cores, 4 GB GDDR5 @ 1392 Mhz (Pascal)	Intel UHD Graphics 620 @ 300 Mhz	Intel HD Graphics 620 @ 300 Mhz
Memória	16 GB DDR4 @ 2400 Mhz	8 GB DDR4 @ 2400 Mhz	4 GB DDR4 @ 2400 Mhz
Disco	HDD 1 TB, 3.5', SATA	SSD 240 GB, 2.5', SATA	SSD 120 GB, 2.5', SATA

suem aproximadamente 5MB de tamanho. Todas as sequências foram obtidas no site do *National Center for Biotechnology Information*[34].

6.2 Testes

Os testes realizados foram divididos em duas categorias: requisições simultâneas e visualização. A primeira categoria visa avaliar a capacidade do servidor para lidar com múltiplas requisições sendo feitas ao mesmo tempo. A segunda categoria avalia a responsividade do computador no lado do cliente ao abrir os resultados do alinhamento. Em ambas categorias, foram requisitados os alinhamentos, ou seja, são executados os Estágios 1 a 5, já que esse tipo de alinhamento é o pior caso para o servidor e para o cliente.

Para a categoria requisições múltiplas, os cenários executados foram 1, 2, 5 e 10 requisições simultâneas. Para cada cenário, as requisições continham os arquivos das sequências de 30KB, 1MB e 5MB, descritas na Seção 6.1, e foram executados cinco vezes, anotando a média e o desvio padrão dos resultados. Ainda, testaram-se os casos onde as sequências eram recebidas pela API do *National Center for Biotechnology Information* e os casos onde as sequências eram fornecidas pelos usuários. As Tabelas 6.2, 6.3, 6.4, 6.5 mostram os tempos necessários para a resposta ao recebimento da requisição com arquivos obtidos pela API, os tempos necessários para a resposta ao recebimento da requisição com arquivos obtidos diretamente pela submissão do usuário, os tempos de processamento do alinhamento e os tempos de devolução dos resultados pelo servidor respectivamente. Para simular as múltiplas requisições, foram desenvolvidos dois pequenos *scripts* em JavaScript

que fazem as requisições por meio de laços. Os *scripts* estão disponíveis no Apêndice (A e B). É importante ressaltar que os tempos representam as médias obtidas após a última requisição ser concluída.

O tempo de resposta ao recebimento da requisição é o tempo necessário para que o servidor receba a requisição de um cliente, obtenha e armazene os arquivos da sequências, insira a requisição na fila de processamento e retorne o identificador do alinhamento para o cliente. O tempo de processamento é o tempo necessário para que todos os alinhamentos requisitados sejam concluídos e tenham seus resultados disponíveis. O tempo de devolução dos resultados é o tempo necessário para que o servidor receba a requisição de um cliente e devolva as informações e os arquivos binário e fasta para que o cliente possa construir os resultados.

Para os tempos de resposta e processamento há a possibilidade de obter os arquivos via API ou submissão do usuário. Os arquivos obtidos via API utilizam o módulo *National Center for Biotechnology Information* (Seção 5.2.5) para obter os arquivos das sequências que serão alinhadas. Já a submissão do usuário utiliza o *input* textual ou *upload* de arquivos de sequência pelo usuário.

Tabela 6.2: Tempo médio de resposta do servidor MASA-Webaligner para requisições simultâneas de alinhamentos. NC indica que não foi possível obter tempo devido à restrições da API.

Origem sequências	**API**			**Usuário**		
Nº de Requisições	**30KB**	**1MB**	**5MB**	**30KB**	**1MB**	**5MB**
1	1,57s	3,31s	7,49s	0,06s	0,32s	1,31s
2	1,63s	3,03s	6,94s	0,09s	0,57s	2,15s
5	1,55s	3,13s	7,35s	0,20s	0,17s	4,34s
10	NC	4,35s	8,45s	0,24s	2,28s	9,59s

Tabela 6.3: Desvio padrão do tempo de resposta para requisições simultâneas de alinhamentos. NC indica que não foi possível obter tempo devido à restrições da API.

Origem sequências	**API**			**Usuário**		
Nº de Requisições	**30KB**	**1MB**	**5MB**	**30KB**	**1MB**	**5MB**
1	0,08	0,75	2,70	0,02	0,05	0,03
2	0,16	0,03	1,11	0,04	0,12	0,11
5	0,07	0,12	0,61	0,01	0,05	0,12
10	NC	1,49	0,72	0,02	0,12	0,14

Além dos testes de requisição e retorno de alinhamentos, foram realizados testes para verificar como os clientes se comportam quando estão processando os resultados de alinhamentos pequenos (30KB), médios (1MB) e grandes (5MB). A Tabela 6.6 mostra a

Tabela 6.4: Tempo de processamento dos alinhamentos pelo servidor MASA-Webaligner após as requisições simultâneas. NC indica que não foi possível obter tempo devido à restrições da API e erros no processamento da fila.

Origem sequências	**API**			**Usuário**		
Nº de Requisições	**30KB**	**1MB**	**5MB**	**30KB**	**1MB**	**5MB**
1	≈ 5s	≈ 30s	NC	≈ 5s	≈ 30s	NC
2	≈ 10s	≈ 60s	NC	≈ 10s	≈ 60s	NC
5	≈ 25s	≈ 150s	NC	≈ 25s	≈ 150s	NC
10	NC	≈ 330s	NC	≈ 50s	≈ 330s	NC

Tabela 6.5: Tempo médio de resposta para os resultados dos alinhamentos, contendo: informações do banco de dados, arquivo binário e arquivos FASTA. NC indica que não foi possível possível obter tempo devido a erros no processamento da fila.

Nº de Requisições	**30KB**	**1MB**	**5MB**
1	0.04s	0.38s	NC
2	0.05s	0.60s	NC
5	0.10s	1.31s	NC
10	0.14s	2.67s	NC

quantidade de tempo e recursos utilizados para carregar o resultados no cliente. A aferição dos resultados foi feita com auxílio do *System Monitor* (Cliente 1) e do Gerenciador de Tarefas (Cliente 2).

Tabela 6.6: Utilização de recursos, para visualização dos resultados de um alinhamento, pelo cliente. NC indica que não foi possível obter o tempo devido a erros no processamento da fila.

Cliente	**1**			**2**		
Sequências	**30KB**	**1MB**	**5MB**	**30KB**	**1MB**	**5MB**
Tempo	≈ 1s	≈ 7s	NC	≈ 2s	≈ 13s	NC
CPU ao carregar	até 19%	até 23%	NC	até 23%	até 50%	NC
CPU ao interagir	até 24%	até 18%	NC	até 34%	até 30%	NC
Memória RAM	150MB	900MB	NC	200MB	1,4GB	NC

6.3 Análise dos resultados

Como pode ser visto na Tabela 6.2, o número de requisições tem pouca influência no tempo de resposta, com arquivos obtidos via API, para sequências de 30KB, porém, para os outros tamanhos, cresce proporcionalmente à medida que o tamanho da sequência aumenta. O mesmo comportamento pode ser observado para os tempos de resposta com arquivos submetidos pelo usuário.

Observa-se também que ao realizar requisições de alinhamentos que fazem uso da API do *National Center for Biotechnology Information* para obter as sequências FASTA, o tempo de resposta do servidor apresenta comportamento mais complexo. Por exemplo, o tempo de resposta, para requisições utilizando a API, com uma única requisição e utilizando arquivos de 1MB demorou, em média, 3,31 segundos, mas para cinco requisições simultâneas com os mesmo arquivos demorou, em média, 3,13 segundos. A Tabela 6.3 mostra que há uma variação maior no desvio padrão quando se utiliza a API, ou seja, há menos uniformidade nos tempos de resposta. Isso se deve à dependência da velocidade de conexão com a Internet pelo servidor e, ao fato de que, ao realizar o *download* da sequência, o processo do servidor fica travado, forçando que apenas uma requisição à API seja feita por vez, independente de quantas requisições estejam chegando. Sobre a velocidade de *download*, não há o que ser feito além de tentar garantir uma boa conexão com a Internet. No entanto, o segundo problema pode ser resolvido adicionando uma funcionalidade experimental do TypeScript: as *Worker Threads* [59]. A ideia seria tornar o *download* das sequências FASTA uma tarefa não blocante, reduzindo o tempo de resposta do servidor.

Nas Tabelas 6.2 e 6.4 verifica-se que apesar de ser possível obter uma resposta do servidor para requisição de alinhamentos grandes ($\geq$ 5MB), o servidor não é capaz de processar o alinhamento em tempo hábil. A biblioteca Bull, utilizada na implementação das filas (Seção 5.2.7), utiliza um mecanismo de trava (*lock*) com temporizador para execução de *jobs*. Os alinhamentos grandes estouram o tempo limite definido pelo Bull e, apesar do processamento do alinhamento ocorrer, o servidor retorna que o *job* falhou. Ainda, quando isso acontece, pode ocorrer, ou não, de a fila, com todos os outros *jobs*, concluídos ou não, ser apagada. É possível que um estudo em maior detalhe na biblioteca e a alteração de parâmetros na inicialização da fila, ou a substituição desta resolva essa questão.

Também nas Tabelas 6.2 e 6.4, observa-se que, quando utilizando a API, não foi possível realizar as requisições simultâneas e, consequentemente, o processamento dos alinhamentos para cenário de dez requisições de alinhamentos pequenos. A API do *National Center for Biotechnology Information* limita sua utilização a 10 requisições por segundo. No caso de 10 requisições simultâneas de alinhamentos pequenos, o limite da API foi estourado e não foi possível realizar os alinhamentos, já que não foi possível obter as sequências necessárias. No entanto, *National Center for Biotechnology Information* permite que o limite seja aumentado mediante solicitação e análise do pedido [60].

No lado dos clientes, é possível verificar que sistemas com *hardware* mais antigo, embora consigam visualizar os resultados de alinhamentos de 1MB, já apresentam limitações de recursos, principalmente de memória RAM. Na Tabela 6.6, o Cliente 2, ao visualizar o

alinhamento de 1MB, utilizou 1.4GB dos 4GB disponíveis. No entanto, o consumo total de memória RAM no cliente atingiu aproximadamente 90%, mostrando que alinhamentos maiores iriam forçar a máquina a um estado frequente de utilização da área *swap*, o que seria desastroso para performance do sistema. Uma forma de tentar remediar essa situação seria disponibilizar a versão Java do MASA-Viewer e dando a possibilidade do usuário poder analisar os resultados fora do ambiente do navegador, o que consome menos recursos. A Figura 6.1 mostra que a visualização de um alinhamento de 5MB consome menos memória do que a visualização dos alinhamentos de 1MB pelo navegador.

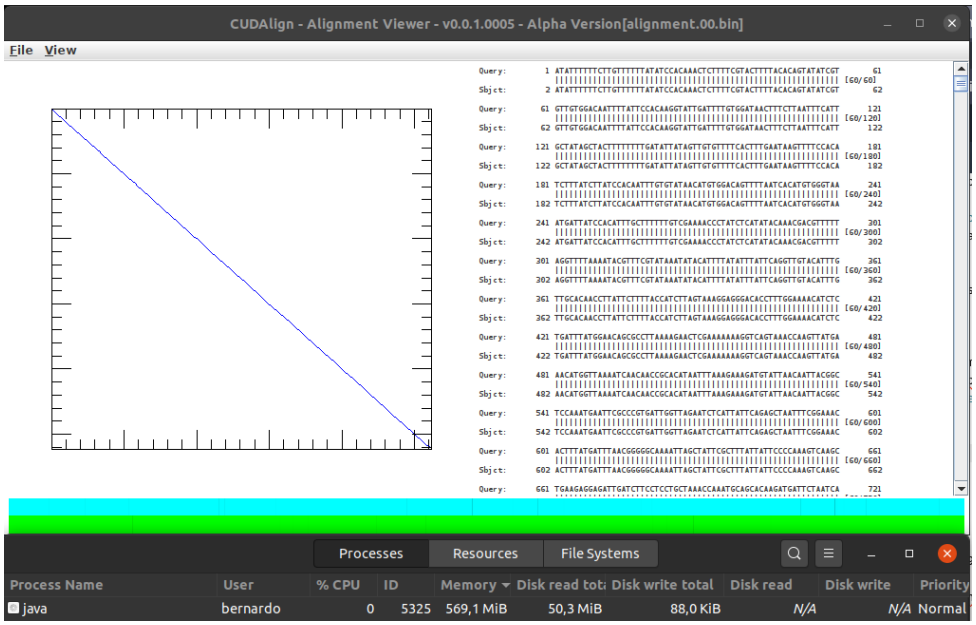


Figura 6.1: Visualização e consumo de memória RAM pelo MASA-Viewer ao abrir um alinhamento de duas sequências de 5MB [3][4].

Ainda sobre os clientes, verifica-se que o consumo de CPU pode alcançar níveis relativamente altos - até 50% no cliente 1, conforme a Tabela 6.6 -, porém isso ocorre apenas no momento em que o alinhamento é carregado. Para interações com o gráfico e a saída textual, o uso de CPU ficou em níveis bem mais baixos.

Por fim, é importante frisar que, embora seja possível testar e avaliar a experiência do usuário com o sistema, esse tipo de teste foge do escopo do presente trabalho. Ainda, é possível afirmar, sem necessidade de testes, que a experiência de usabilidade de uma plataforma com interface gráfica é muito superior à usabilidade dos *softwares* por meio de um terminal para usuários leigos.

Capítulo 7

Conclusão

Neste trabalho de graduação foi proposta, implementada e avaliada a plataforma MASA-Webaligner, que integra as ferramentas de alinhamento de sequências MASA-CUDAlign e MASA-OpenMP, e o sistema MASA-Viewer. O MASA-Webaligner foi implementado para a utilização na web, visando facilitar o uso do MASA-CUDAlign e MASA-OpenMP por biólogos e pesquisadores da área, abstraindo todo o processo de instalação e configuração dos sistemas.

Os resultados experimentais obtidos (Seção 5.3.2) mostram que o sistema suporta a utilização simultânea da plataforma por vários usuários, sem um prejuízo de tempo de resposta e processamento, para alinhamentos pequenos ($x <$ 500KB) e médios (500KB $\leq x \leq$ 1MB) em um servidor com uma única placa de vídeo, apresentando resultados melhores ao utilizar múltiplas placas de vídeo. Devido a integração com a API do *National Center for Biotechnology Information*, o sistema torna simples a utilização de sequências de DNA que pertencem ao banco de dados da instituição.

O sistema também facilita a utilização e a análise dos resultados obtidos. Com uma única instalação, múltiplos usuários têm acesso ao serviço de alinhamento, utilizando um sistema de filas para o processamento das requisições. Após o processamento, usuários podem ser avisados por *e-mail* sobre a conclusão de suas requisições, têm acesso à integração com o visualizador gráfico com função de zoom e possibilidade de *download* do gráfico obtido, e têm acesso à saída textual com possibilidade de *download* dos resultados obtidos. Ainda, recebem estatísticas sobre o processamento de alinhamento para comparações com outras ferramentas.

Foi observado, durante os experimentos, que o MASA-Webaligner interage bem com a requisição de alinhamentos que dependem da API do *National Center for Biotechnology Information*, porém possui algumas questões relacionadas a acessos concorrentes. Isso se deve à forma com que o JavaScript lida com as requisições no *back-end*. A linguagem não foi construída com foco em concorrência e, portanto, ao realizar as chamadas à API, o

servidor realiza uma ação blocante, o que o atrapalha a lidar com múltiplas requisições simultâneas.

No lado do cliente, os alinhamentos pequenos e médios não exigiram grandes recursos computacionais do sistema. A partir dos alinhamentos de tamanho médio ($x \geq$ 1MB), optou-se por omitir o gráfico dos resultados para que a utilização dos recursos do sistema mantenha-se baixa.

Tendo em vista os resultados obtidos, concluímos que o MASA-Webaligner atingiu os objetivos propostos. A plataforma realiza a integração do MASA-CUDAlign, MASA-OpenMP e MASA-Viewer, abstrai o processo de instalação, facilita o uso da plataforma MASA, proporciona acesso à total capacidade de processamento do sistema para todos os usuários, pode avisar os usuários sobre a conclusão do processamento, gera os resultados gráficos e textuais, disponibilizando-os para visualização e *download* sem que haja grande utilização de recursos computacionais no sistema dos usuários.

Como trabalhos futuros, sugerimos uma refatoração ou nova implementação do gerenciador de filas, para que seja possível utilizar o sistema sem a limitação no tamanho dos alinhamentos requisitados. Ainda, sugerimos também a introdução de mecanismo de concorrência para que o impacto de múltiplas requisições, que utilizam a API do *National Center for Biotchnology Information*, nos tempos de resposta do servidor seja diminuído.

Sugerimos também um estudo sobre a possibilidade de exibição dos gráficos, para quaisquer tamanhos de alinhamentos, sem que haja um impacto significativo, no desempenho. Caso essa alternativa se mostre inviável, sugerimos a implementação de uma opção de *download* do sistema MASA-Viewer, para que o usuário tenha a possibilidade de visualizar o gráfico do alinhamento em sua máquina pessoal e fora do ambiente do navegador.

Ainda, atualmente o MASA-Webaligner não é capaz de realizar uma limpeza dos alinhamentos requisitados, o que, com o passar do tempo, irá gerar uma grande quantidade de dados, os quais não serão mais utilizados. Sugerimos o desenvolvimento de uma funcionalidade, que periodicamente, apague todas as informações em disco e no banco de dados que sejam anteriores a um determinado número de dias.

A plataforma MASA também é capaz de realizar alinhamentos semi-globais, os quais não estão disponíveis para requisição na versão atual do MASA-Webaligner. Sugerimos a implementação da possibilidade de requisição dos alinhamentos semi-globais, de forma a utilizar de maneira completa a plataforma MASA, para trabalhos futuros.

A fila implementada pelo MASA-Webaligner cresce ou diminui de acordo com a utilização do sistema. No entanto, não há, atualmente, um sistema que persista dados sobre a utilização da fila. Sugerimos a implementação de uma funcionalidade que persista informações sobre a utilização da fila do sistema.

Por fim, o MASA-Webaligner está construído em forma de monólito, o que pode gerar um impacto negativo na performance do sistema, além de tornar sua manutenção mais custosa. Sugerimos que a arquitetura do sistema seja alterada para uma de micro serviços, de forma que haja um servidor responsável por receber as requisições de alinhamento e persistir os dados no banco, e um segundo servidor responsável por realizar os alinhamentos e persistir os arquivos de controle em disco.

Referências

[1] Oliveira Sandes, Edans Flávius de: *Algoritmos Paralelos Exatos e Otimizações para Alinhamento de Sequências Biológicas Longas em Plataformas de Alto Desempenho.* Tese de Doutoramento, Universidade de Brasília, 2015. x, xi, 1, 2, 5, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 24, 25, 26

[2] Myers, Eugene W. e Webb Miller: *Optimal alignments in linear space.* Computer Applications on Biosciences, 4(1):11–17, 1988. x, 10, 11

[3] *Bacillus anthracis str. ames, complete genome.* `https://www.ncbi.nlm.nih.gov/nuccore/AE016879.1?report=fasta`, acesso em 2019-11-19. xi, xiii, 29, 30, 31, 34, 36, 37, 38, 64, 69

[4] *Bacillus anthracis str. sterne, complete genome.* `https://www.ncbi.nlm.nih.gov/nuccore/AE017225.1?report=fasta`, acesso em 2019-11-19. xi, xiii, 29, 30, 31, 34, 36, 37, 38, 64, 69

[5] *Hiv-1 isolate mb2059 from kenya, complete genome.* `https://www.ncbi.nlm.nih.gov/nuccore/AF133821.1/?report=fasta`, acesso em 2019-11-19. xi, 29, 30, 35

[6] *Hiv-1 isolate sf33 from usa, complete genome.* `https://www.ncbi.nlm.nih.gov/nuccore/AY352275.1?report=fasta`, acesso em 2019-11-19. xi, 29, 30, 35

[7] Mount, David W.: *Bioinformatics - Sequence and Genome Analysis.* Cold Spring Harbor Laboratory Press, 2001. 1, 5, 6

[8] Needleman, S. B. e C. D. Wunsch: *A general method applicable to the search for similarities in the amino acid sequence of two proteins.* Journal of Molecular Biology, 48(3):443–453, 1970. 1, 7

[9] Smith, T. F. e M. S. Waterman: *Identification of common molecular subsequences.* Journal of Molecular Biology, 147(1):195–197, 1981. 1, 8

[10] Pearson, William R. e David J. Lipman: *Improved tools for biological sequence comparison.* 85:2444–2448, 1988. 1, 34

[11] *Blast: Basic local alignment search tool.* `https://blast.ncbi.nlm.nih.gov/Blast.cgi`, acesso em 2019-11-19. 1, 32

[12] Oliveira Sandes, Edans Flavius de, Guillermo Miranda, Xavier Martorell, Eduard Ayguade, George Teodoro e Alba Cristina Magalhaes Melo: *Cudalign 4.0: Incremental speculative traceback for exact chromosome-wide alignment in gpu clusters.* Parallel and Distributed Systems, IEEE Transactions on, 27(10), 2016. 2, 3

[13] Durbin, Richard, Anders Krogh Sean R. Eddy e Graeme Mitchison: *Biological Sequence Analysis - Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1999. 3, 4, 6, 7, 8, 9

[14] Gusfield, Dan: *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997. 4, 6

[15] *Pairwise sequence alignment tools*. `https://www.ebi.ac.uk/Tools/psa/`, acesso em 2019-10-03. 5, 27

[16] Gotoh, O.: *An improved algorithm for matching biological sequences*. Journal of Molecular Biology, 162(3):705–708, 1982. 9

[17] *Human genome project faq*. `https://www.genome.gov/human-genome-project/Completion-FAQ`, acesso em 2019-10-03. 10

[18] Sandes, Edans Flávius Oliveira: *Comparação paralela de sequências biológicas longas utilizando unidades de processamento gráfico (gpus)*. Tese de Mestrado, Universidade de Brasília, 2011. 12

[19] Oliveira Sandes, Edans Flavius de, George Luiz Medeiros Teodoro, Maria Emilia M. T. Walter, Xavier Martorell, Eduard Ayguadé e Alba Cristina Magalhaes Alves de Melo: *Formalization of block pruning: Reducing the number of cells computed in exact biological sequence comparison algorithms*. Computer Journal, 61(5):687–713, 2018. 21

[20] *Emboss needle*. `https://www.ebi.ac.uk/Tools/psa/emboss_needle/`, acesso em 2019-11-18. 28

[21] *Emboss water*. `https://www.ebi.ac.uk/Tools/psa/emboss_needle/`, acesso em 2019-11-18. 28

[22] *The european molecular biology open software suite*. `http://emboss.open-bio.org/`, acesso em 2019-11-18. 29

[23] *Human herpesvirus 6b, complete genome*. `https://www.ncbi.nlm.nih.gov/nuccore/NC_000898.1?report=fasta`, acesso em 2019-11-19. 29, 30

[24] *Human gammaherpesvirus 4, complete genome*. `https://www.ncbi.nlm.nih.gov/nuccore/NC_007605.1?report=fasta`, acesso em 2019-11-19. 29, 30

[25] *Chlamydia trachomatis a/har-13, complete genome*. `https://www.ncbi.nlm.nih.gov/nuccore/CP000051.1?report=fasta`, acesso em 2019-11-19. 29, 30, 64

[26] *Chlamydia muridarum nigg, complete genome*. `https://www.ncbi.nlm.nih.gov/nuccore/AE002160.2?report=fasta`, acesso em 2019-11-19. 29, 30, 64

[27] *Mummer*. `https://github.com/mummer4/mummer`, acesso em 2019-11-19. 30

[28] *Gnuplot homepage*. `http://www.gnuplot.info/`, acesso em 2019-11-19. 32

[29] Zhang, Zheng, Scott Schwartz, Lukas Wagner e Webb Miller: *A greedy algorithm for aligning dna sequences*. Journal of Computational Biology, 7(1/2):203–214, 2000. 33

[30] *Uva fasta server*. https://fasta.bioch.virginia.edu/, acesso em 2019-11-28. 36

[31] *Git repository for fasta36 sequence comparison software*. https://github.com/wrpearson/fasta36, acesso em 2019-11-28. 36

[32] *Express - framework de aplicativo da web node.js*. https://expressjs.com/pt-br/, acesso em 2020-06-03. 39, 43

[33] *React - uma biblioteca javascript para criar interfaces de usuário*. https://pt-br.reactjs.org/, acesso em 2020-07-13. 39

[34] *National center for biotechnology information*. https://www.ncbi.nlm.nih.gov/, acesso em 2020-06-18. 40, 65

[35] *Node.js*. https://nodejs.org/en/, acesso em 2020-07-13. 41, 60

[36] *Typescript: Typed javascript at any scale*. https://www.typescriptlang.org/, acesso em 2020-05-24. 41

[37] *Premium queue package for handling distributed jobs and messages in nodejs*. https://github.com/OptimalBits/bull, acesso em 2020-06-05. 42, 48

[38] Martin, Robert C.: *Arquitetura Limpa - O Guia do Artesão para Estrutura e Design de Software*. Alta Books, 2020. 42

[39] *Lightweight dependency injection container for javascript/typescript*. https://github.com/microsoft/tsyringe, acesso em 2020-06-09. 43, 44

[40] *Writing middleware for use in express apps*. http://expressjs.com/en/guide/writing-middleware.html, acesso em 2020-07-13. 43

[41] *A joi validation middleware for express*. https://github.com/arb/celebrate, acesso em 2020-06-03. 45

[42] *Multer is a node.js middleware for handling multipart/form-data*. https://github.com/expressjs/multer, acesso em 2020-07-13. 46

[43] *Postgresql: The world's most advanced open source database*. https://www.postgresql.org/, acesso em 2020-07-13. 48

[44] *Typeorm - amazing orm for typescript and javascript*. https://typeorm.io/, acesso em 2020-06-10. 48

[45] *Handlebars*. https://handlebarsjs.com/, acesso em 2020-06-10. 50

[46] *Components e props*. https://pt-br.reactjs.org/docs/components-and-props.html, acesso em 2020-05-29. 51

[47] *Estado e ciclo de vida*. https://pt-br.reactjs.org/docs/state-and-lifecycle.html, acesso em 2020-05-29. 51

[48] *Introdução aos hooks*. https://pt-br.reactjs.org/docs/hooks-intro.html, acesso em 2020-05-31. 51

[49] *Masa-core: Multi-platform architecture for sequence aligner*. https://github.com/edanssandes/MASA-Core, acesso em 2020-05-14. 51

[50] *Dead simple object schema validation*. https://github.com/jquense/yup, acesso em 2020-05-31. 52

[51] *Masa-cudalign*. https://github.com/edanssandes/MASA-CUDAlign, acesso em 2020-07-15. 57, 58

[52] *Masa-openmp*. https://github.com/edanssandes/MASA-OpenMP, acesso em 2020-07-15. 57, 59

[53] *Masa-webaligner*. https://github.com/bernas1104/MASA-Webaligner, acesso em 2020-07-15. 61

[54] *Nginx | high performance load balancer, web server, reverse proxy*. https://www.nginx.com/, acesso em 2020-07-15. 62

[55] *Netlify: All-in-one platform for automating modern web projects*. https://www.netlify.com/, acesso em 2020-07-15. 62

[56] *Digitalocean - the developer cloud*. https://www.digitalocean.com/, acesso em 2020-07-15. 63

[57] *Severe acute respiratory syndrome coronavirus 2 isolate sars-cov-2/human/bra/sp02/2020, complete genome*. https://www.ncbi.nlm.nih.gov/nuccore/MT126808.1?report=fasta, acesso em 2020-06-18. 64

[58] *Severe acute respiratory syndrome coronavirus 2 isolate wuhan-hu-1, complete genome*. https://www.ncbi.nlm.nih.gov/nuccore/NC_045512.2?report=fasta, acesso em 2020-06-18. 64

[59] *Node.js multithreading: What are worker threads, and why do they matter?* https://blog.logrocket.com/node-js-multithreading-what-are-worker-threads-and-why-do-they-matter-48ab102f8b1 acesso em 2020-06-25. 68

[60] https://www.ncbi.nlm.nih.gov/books/NBK25497/#:~:text=Only%20one%20API%20key%20is,associated%20with%20that%20NCBI%20account, acesso em 2020-06-25. 68

Apêndice A

Script para teste de requisição de alinhamentos

```
const fs = require('fs');
const path = require('path');
const axios = require('axios');

const api = axios.create({
  baseURL: 'http://192.168.0.250:3333',
});

const KB301 = fs.readFileSync(
  path.resolve(__dirname, '30KB', 'MT126808.1.fasta'),
  'utf-8',
);

const KB302 = fs.readFileSync(
  path.resolve(__dirname, '30KB', 'NC_045512.2.fasta'),
  'utf-8',
);

const MB11 = fs.readFileSync(
  path.resolve(__dirname, '1MB', 'AE002160.2.fasta'),
  'utf-8',
);

const MB12 = fs.readFileSync(
```

```
  path.resolve(__dirname, '1MB', 'CP000051.1.fasta'),
  'utf-8',
);

const MB51 = fs.readFileSync(
  path.resolve(__dirname, '5MB', 'AE016879.1.fasta'),
  'utf-8',
);

const MB52 = fs.readFileSync(
  path.resolve(__dirname, '5MB', 'AE017225.1.fasta'),
  'utf-8',
);

async function runTests(times) {
  const s0inputs = [KB301, MB11, MB51];
  const s1inputs = [KB302, MB12, MB52];

  for (let i = 0; i < times; i++) {
    const before = new Date().getTime();

    axios({
      method: 'post',
      url: 'http://192.168.0.250:3333/alignments',
      data: {
        extension: 1,
        type: 'global',
        only1: false,
        clearn: false,
        block_pruning: true,
        complement: 0,
        reverse: 0,
        s0origin: 3,
        s1origin: 3,
        s0input: s0inputs[1],
        s1input: s1inputs[1],
        full_name: 'John Doe',
```

```
        email: 'johndoe@example.com',
      },
      maxBodyLength: 20000000,
      maxContentLength: 20000000,
    }).then(() => {
      const after = new Date().getTime();
      console.log(
        after - before + ' milliseconds for request ' + (i + 1)
      );
    }).catch((err) => {
      console.log(err.response.data);
    });
  }
}

runTests(1);
```

Apêndice B

Script para teste de requisição dos resultados

```
const axios = require('axios');

const api = axios.create({
  baseURL: 'http://192.168.0.250:3333',
});

async function runTests(times) {
  const ids = [
        '4a618d36-a2f7-46f5-bb38-fd462c894b01',
        '2d19a486-fec5-430f-8476-ec1cca591474'
  ];

  for (let i = 0; i < times; i++) {
    const before = new Date().getTime();

    api.get(`alignments/${ids[0]}`).then(async () => {

      await api.get(`files/bin/${ids[0]}`);
      await api.get(`files/fasta/${ids[0]}`);

      const after = new Date().getTime();
      console.log(
        after - before + ' milliseconds for request ' + (i + 1)
      );
```

```
    }).catch((err) => {
      console.log(err.response.data);
    });
  }
}

runTests(10);
```