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

Estratégias de Otimização para Alinhamento Múltiplo Paralelo Exato de Sequências Biológicas

Vinícius Manoel Texeira Ribeiro

Dissertação apresentada como requisito para
conclusão do Mestrado em Informática

Orientadora
Prof.a Dr.a Alba Cristina Magalhães Alves de Melo

Brasília
2026

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

Estratégias de Otimização para Alinhamento Múltiplo Paralelo Exato de Sequências Biológicas

Vinícius Manoel Texeira Ribeiro

Dissertação apresentada como requisito para
conclusão do Mestrado em Informática

Prof.a Dr.a Alba Cristina Magalhães Alves de Melo (Orientadora)
CIC/UnB

Prof. Dr. Márcio Bastos Castro
INE/UFSC

Prof. Dr. Ricardo Pezzuol Jacobi
CIC/UnB

Prof. Dr. Li Weigang
CIC/UnB

Prof.a Dr.a Cláudia Nalon
Coordenadora do Programa de Pós-graduação em Informática

Brasília, 5 de março de 2026

Dedicatória

À minha família e amigos, pela compreensão nas ausências e pelas palavras de incentivo que tornaram minha caminhada mais leve.

Agradecimentos

Sou grato, primeiramente, a Deus, pela vida, pela força e sabedoria concedidas ao longo desta caminhada.

Sou profundamente grato à minha orientadora, Professora Alba, pelo tempo que dedicou, pelas discussões sobre estratégias, pelo planejamento e pelas constantes revisões e reuniões para garantir que todo o trabalho estivesse pronto. Sem sua experiência e orientação excepcional, eu jamais teria chegado até aqui.

Agradeço ao Professor Daniel Sundfeld, por conceder a oportunidade de trabalhar com sua ferramenta. Além disso, sou profundamente grato pelo tempo que você dedicou às reuniões de estratégia e aos testes realizados na AWS; sem sua ampla experiência e conhecimento, eu nunca teria alcançado resultados tão significativos.

Agradeço ao projeto POC N2P da RNP/AWS por disponibilizar os créditos que foram utilizados nas execuções na nuvem. O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001

Por fim, agradeço a todos que, direta ou indiretamente, contribuíram para a concretização deste sonho.

A todos vocês, muito obrigado!

Resumo

O Alinhamento Múltiplo de Sequências (MSA) é um método utilizado na Bioinformática para a análise de relações evolutivas e funcionais entre sequências biológicas. No entanto, encontrar o alinhamento matematicamente ótimo é um problema NP-Completo, tornando os algoritmos exatos inviáveis para a maioria dos conjuntos de dados práticos. A ferramenta PA-Star, uma implementação paralela do algoritmo de busca A^*, representa um avanço ao garantir a otimalidade da solução. Contudo, entre outros fatores, sua eficiência em reduzir o espaço de busca é limitada pela função heurística $h_{2,all}$ (baseada em pares de sequências), resultando em uma poda pouco agressiva. Esta Dissertação de Mestrado propõe otimizações para o PA-Star, focando na substituição da heurística $h_{2,all}$ por uma função mais robusta, a $h_{3,all}$ (baseada em trios de sequências), e na paralelização de sua etapa de cálculo. A hipótese central é que a $h_{3,all}$, apesar do maior custo computacional, se comparado ao $h_{2,all}$, fornecerá um limite inferior mais preciso, resultando em uma poda mais eficaz do espaço de busca e, consequentemente, em uma redução do tempo de execução total. Os resultados obtidos em duas plataformas multithreaded (ambiente local e nuvem AWS) mostram que, apesar do tempo de cálculo da heurística $h_{3,all}$ ser bem maior do que o da $h_{2,all}$, o ganho médio do tempo total da execução da ferramenta PA-Star com $h_{3,all}$ é de cerca de 34,20% no ambiente da nuvem AWS com o conjunto `arp.fasta` de proteínas do benchmark BAliBASE. Além disso, mostramos que a solução apresenta boa aceleração (speedup) para até 64 cores.

Palavras-chave: Alinhamento Múltiplo de Sequências, Algoritmo A*, Computação Paralela, Bioinformática, Ferramenta PA-Star

Abstract

Multiple Sequence Alignment (MSA) is a method used in Bioinformatics for analyzing evolutionary and functional relationships between biological sequences. However, finding the mathematically optimal alignment is an NP-Complete problem, making exact algorithms infeasible for most practical datasets. The PA-Star tool, a parallel implementation of the A* search algorithm, represents an advance by guaranteeing solution optimality. However, among other factors, its efficiency in reducing the search space is limited by the heuristic function $h_{2,all}$ (based on sequence pairs), resulting in less aggressive pruning. This Master's Dissertation proposes optimizations for PA-Star, focusing on replacing the $h_{2,all}$ heuristic with a more robust function, $h_{3,all}$ (based on sequence trios), and on parallelizing its computation stage. The central hypothesis is that $h_{3,all}$, despite the higher computational cost compared to $h_{2,all}$, will provide a more precise lower bound, resulting in more effective pruning of the search space and, consequently, a reduction in total execution time. The results obtained on two multithreaded platforms (local environment and AWS cloud) show that, although the computation time of the $h_{3,all}$ heuristic is much higher than that of $h_{2,all}$, the average total execution time gain of the PA-Star tool with $h_{3,all}$ is about 34.20% in the AWS cloud environment with the `arp.fasta` protein dataset from the BAliBASE benchmark. In addition, we show that the solution presents good acceleration (speedup) for up to 64 cores.

Keywords: Multiple Sequence Alignment, A* Algorithm, Parallel Computing, Bioinformatics, PA-Star Tool

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

API Application Programming Interface.

AWS Amazon Web Services.

BAliBASE Benchmark Alignment dataBASE.

BLOSUM BLOcks SUbstitution Matrix.

DAPA Disk-Assisted PA-Star.

DNA Ácido Desoxirribonucleico.

FFT Fast Fourier Transform.

HMM Modelos Ocultos de Markov.

MAFFT Multiple Alignment using Fast Fourier Transform.

MPI Message Passing Interface.

MSA Multiple Sequence Alignment.

MUSCLE MUltiple Sequence Comparison by Log-Expectation.

MVC Model-View-Controller.

NP Nondeterministic Polynomial time.

PAM Point Accepted Mutation.

PSA Pairwise Sequence Alignment.

REST Representational State Transfer.

RNA Ácido Ribonucleico.

SIMD Single Instruction, Multiple Data.

SP Sum-of-Pairs.

T-Coffee Tree-based Consistency Objective For alignmEnt Evaluation.

WSP Weighted Sum-of-Pairs.

Capítulo 1

Introdução

A análise de sequências biológicas, como DNA, RNA e proteínas, representa uma das áreas centrais da Bioinformática, sendo fundamental para estudos em genômica comparativa, inferência de funções gênicas e investigação de relações evolutivas entre organismos [1, 2]. Nesse contexto, o Alinhamento Múltiplo de Sequências (MSA) surge como uma ferramenta computacional indispensável, permitindo a comparação simultânea de três ou mais sequências para identificar padrões evolutivos, relações funcionais e estruturas moleculares conservadas [3, 4].

Contudo, a busca pelo alinhamento matematicamente ótimo é um problema de elevada complexidade computacional, classificado como NP-Completo [4, 5, 6]. Tal complexidade implica que o tempo de execução dos algoritmos exatos conhecidos cresce exponencialmente com o número de sequências, tornando sua aplicação inviável para a maioria dos conjuntos de sequências. Diante desse desafio, a comunidade científica tem recorrido predominantemente a algoritmos heurísticos, que, embora não garantam a otimalidade da solução, oferecem um equilíbrio entre a eficiência computacional e a qualidade do alinhamento final [7].

Apesar da predominância de abordagens heurísticas, os algoritmos exatos desempenham um papel teórico importante, servindo como "padrão-ouro" para a avaliação de novas metodologias e definindo o limite superior de desempenho para uma dada função de pontuação [8]. Dentro deste paradigma, o algoritmo de busca A^* se apresenta como uma alternativa promissora à programação dinâmica tradicional, reformulando o MSA como um problema de busca pelo caminho de menor custo em um grafo [4].

A ferramenta PA-Star é uma implementação paralela e de alto desempenho baseada no algoritmo A^*, projetada especificamente para o MSA [9, 10]. Esta ferramenta garante a obtenção da solução ótima, atingindo resultados competitivos para conjuntos de dados de referência, como os *benchmarks* do BAliBASE [11], sendo capaz de resolver instâncias de até 10 sequências com tamanho máximo de 200 resíduos. O foco da presente Disserta-

ção de Mestrado reside no avanço de soluções exatas para problemas de Bioinformática, explorando os limites e os potenciais de métodos que garantem a otimalidade.

1.1 Motivação

A eficiência prática do PA-Star, assim como de qualquer algoritmo baseado em A^*, é fortemente influenciada pela qualidade da função heurística utilizada para guiar a busca. Atualmente o PA-Star utiliza a heurística clássica $h_{2,\text{all}}$, que consiste na soma dos escores dos alinhamentos ótimos de todos os pares de sequências [10]. Embora essa abordagem garanta a admissibilidade, que é uma condição necessária para a otimalidade, ela frequentemente resulta em uma poda pouco agressiva do espaço de busca. Isso ocorre porque o limite inferior do custo fornecido pela $h_{2,\text{all}}$ pode ser impreciso em relação ao custo real do alinhamento múltiplo, forçando o algoritmo a explorar um número excessivo de estados [12].

Adicionalmente, identifica-se uma limitação na arquitetura atual do PA-Star referente à etapa de pré-processamento. O cálculo da heurística $h_{2,\text{all}}$, que demanda o cálculo de todos os alinhamentos par a par, é executado sequencialmente. À medida que o número de sequências ou seu comprimento aumenta, essa etapa passa a representar um gargalo computacional significativo, limitando o uso da ferramenta.

1.2 Contribuições

O objetivo desta Dissertação de Mestrado é propor e avaliar otimizações para a ferramenta PA-Star, com foco na melhoria da função heurística e na eficiência de sua etapa de cálculo. As principais contribuições desta pesquisa são:

- **Implementação da Heurística baseada em Trios ($h_{3,\text{all}}$):** Propõe-se a substituição da heurística de pares pela função $h_{3,\text{all}}$, baseada no alinhamento exato de trios de sequências [12]. A hipótese central é que, apesar de seu maior custo computacional, a $h_{3,\text{all}}$ fornecerá um limite inferior mais preciso, resultando em uma poda mais eficaz do espaço de busca e, consequentemente, na redução do tempo total de execução da ferramenta PA-Star.
- **Paralelização do Cálculo Heurístico:** Será projetada uma estratégia paralela para a etapa de cálculo da heurística. Dado que a computação dos trios é independente e computacionalmente intensiva, a paralelização visa mitigar o custo extra introduzido pela nova heurística, permitindo que o seu processamento seja realizado em tempo hábil.

Como resultado, almeja-se alcançar um desempenho computacional superior, permitindo que a ferramenta PA-Star resolva instâncias mais complexas do problema de MSA de forma ótima.

1.3 Estrutura do Documento

O restante deste documento está organizado da seguinte forma:

- **Capítulo 2:** Detalha os fundamentos teóricos do Alinhamento Múltiplo de Sequências, abordando as principais estratégias de resolução exatas e heurísticas, com ênfase no funcionamento do algoritmo A^*.
- **Capítulo 3:** Apresenta uma revisão dos trabalhos relacionados, contextualizando a proposta deste trabalho frente ao estado da arte em alinhamento de sequências e computação paralela.
- **Capítulo 4:** Descreve a metodologia empregada para o projeto e implementação das otimizações propostas, incluindo a definição formal da heurística $h_{3,\text{all}}$ e a estratégia de paralelização adotada.
- **Capítulo 5:** Apresenta e discute os resultados experimentais obtidos, comparando o desempenho da nova abordagem com a versão original da ferramenta.
- **Capítulo 6:** Apresenta as considerações finais, as contribuições da pesquisa e apontando direções para trabalhos futuros.

Capítulo 2

Alinhamento Múltiplo de Sequências

Este capítulo trata do Alinhamento Múltiplo de Sequências biológicas do inglês *Multiple Sequence Alignment*. Para isso são discutidas as estratégias de avaliação de alinhamentos, bem como abordagens exatas e heurísticas para a resolução do problema, considerando seus aspectos de complexidade computacional e aplicabilidade prática. O capítulo também apresenta avanços recentes em algoritmos otimizados para MSA, destacando a relevância dessas técnicas para o tratamento eficiente de grandes volumes de dados biológicos.

2.1 Introdução

A análise de sequências biológicas (DNA, RNA e proteínas) constitui uma das áreas centrais da Bioinformática e da Biologia Molecular [1]. A capacidade de comparar sequências permite a realização de estudos em genômica comparativa, a inferência de funções para novos genes ou proteínas e a investigação de relações evolutivas entre diferentes organismos [1]. Essa comparação, contudo, depende da distinção precisa entre os conceitos de similaridade e homologia.

No contexto da Bioinformática, postula-se que a similaridade é uma medida quantificável, que representa o grau de correspondência entre os resíduos de duas ou mais sequências após um alinhamento [4]. Por outro lado, a homologia é uma hipótese sobre a ancestralidade evolutiva. Duas sequências são consideradas homólogas se compartilharem um ancestral comum, sendo esta uma proposição de natureza binária, isto é, ou há homologia, ou não há [4]. A similaridade entre sequências é frequentemente utilizada como evidência para se inferir a homologia, mas os termos não são sinônimos. No presente trabalho, será tratada a similaridade.

A tarefa computacional mais fundamental para a comparação de sequências é o alinhamento par a par do inglês *Pairwise Sequence Alignment* (PSA) [4]. O objetivo do PSA é determinar a melhor correspondência entre duas sequências, o que é alcançado

por meio da inserção de lacunas (*gaps*) para maximizar uma pontuação de similaridade. Essa pontuação reflete a verossimilhança de que as sequências tenham derivado de um ancestral comum [4].

O método da programação dinâmica é a solução clássica para o problema de PSA que assegura a obtenção do alinhamento par a par de maior pontuação, ou seja, o alinhamento ótimo [4, 13]. O algoritmo de Needleman-Wunsch, por exemplo, constrói uma matriz bidimensional na qual cada célula (i, j) armazena a pontuação máxima para o alinhamento dos prefixos de tamanho i e j das duas sequências [13]. Ao preencher a matriz de forma recorrente, o algoritmo identifica o caminho de maior pontuação, que corresponde ao alinhamento global ótimo.

O MSA surgiu como uma necessidade natural nas primeiras aplicações computacionais voltadas à Biologia Molecular e à Bioinformática, particularmente quando se tornou evidente a importância de comparar simultaneamente três ou mais sequências biológicas de DNA, RNA ou proteínas, para fins de análise estrutural, funcional e evolutiva [2]. Inicialmente, os esforços na área concentravam-se no alinhamento de pares de sequências, mas logo se verificou que, em diversos contextos, o alinhamento múltiplo oferecia uma representação mais abrangente das relações filogenéticas e das regiões conservadas entre sequências homólogas. Desde então, o MSA consolidou-se como um dos procedimentos metodológicos centrais na Bioinformática, permitindo a identificação de padrões evolutivos, domínios funcionais e estruturas moleculares comuns entre organismos ou genes relacionados [14].

A relevância biológica do MSA advém de sua capacidade de detectar regiões de similaridade e variabilidade em conjuntos de sequências correlacionadas, o que oferece suporte fundamental para problemas mais complexos, como dedução de funções moleculares e estruturas tridimensionais. O alinhamento múltiplo permite que as sequências sejam organizadas, de modo a maximizar sua semelhança estrutural e funcional, servindo como base para análises filogenéticas, caracterização de famílias gênicas e identificação de motivos conservados [3].

2.2 Definição de Alinhamento Múltiplo de Sequências (MSA)

Formalmente, o MSA de um conjunto $\mathcal{S} = \{S_1, S_2, \ldots, S_N\}$, com $N > 2$ sequências, consiste na inserção de lacunas (*gaps*) em posições apropriadas, de modo a igualar os comprimentos das sequências e formar uma matriz de N linhas por L colunas. Cada coluna dessa matriz contém um caractere ou uma lacuna proveniente de cada sequência, e o objetivo é alinhar os caracteres de maneira a maximizar a similaridade entre as sequências,

usualmente medida por um escore de substituição. As sequências são compostas por símbolos de um alfabeto Σ, sendo $\Sigma = \{A, T, G, C\}$ no caso de sequências de DNA, ou $\Sigma = \{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y\}$ para os 20 aminoácidos padrão, para sequências de proteínas. Por exemplo, considerando as sequências S_1 = DQLF, S_2 = DNVQ e S_3 = QGL, os caracteres são denotados por $S_i[j] = x$, ou seja, x representa o j-ésimo caractere da sequência S_i. A Figura 2.1 ilustra esse conceito por meio do alinhamento de três sequências curtas de aminoácidos.

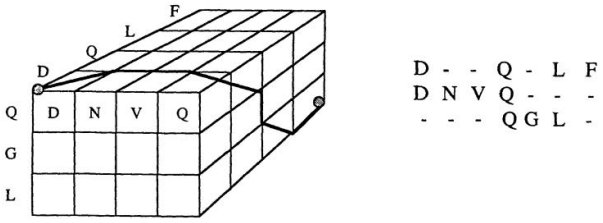

Figura 2.1: Alinhamento de Três Sequências [5]

Para calcular o alinhamento múltiplo, temos que determinar qual função vai ser utilizada para o cálculo do escore, que deve ser maximizado ou minimizado [15]. A função mais utilizada para tal fim é a SP (do inglês *Sum-of-Pairs*), que será explicada na Seção 2.3.

Do ponto de vista computacional, o problema de encontrar o alinhamento múltiplo que otimiza a função SP é NP-completo [4]. A dificuldade computacional é tão intrínseca que foi demonstrado que o problema permanece NP-difícil mesmo sob condições mais restritas, como quando a função de escore de pares satisfaz as propriedades de uma métrica [6]. Isso implica que o tempo de execução de algoritmos exatos conhecidos cresce exponencialmente com o número de sequências, tornando inviável a busca pela solução ótima para a maioria dos conjuntos de dados. Lipman et al. [5] propuseram um método para reduzir o espaço de busca, mantendo a garantia da solução ótima. Apesar de recuperar o alinhamento mais rápido no caso médio, no pior caso a complexidade de tempo é exponencial.

Os algoritmos de MSA dividem-se em duas categorias principais: **Algoritmos exatos**, que exploram todo ou grande parte do espaço de busca em busca da solução ótima; e **Algoritmos heurísticos**, que priorizam a eficiência computacional em detrimento da otimalidade. Os algoritmos exatos, como os baseados em programação dinâmica e no método de Carrillo-Lipman [5], apresentam complexidade exponencial, tornando-se inviáveis para conjuntos grandes de sequências [4]. Alternativas exatas como o algoritmo A^* também foram exploradas, com promissora aplicabilidade desde que acompanhadas de heurísticas admissíveis. Por outro lado, os algoritmos heurísticos, como os métodos progressivos e iterativos, oferecem soluções aproximadas com tempo de execução viável para dados em larga escala [7].

Adicionalmente, a classificação do MSA pode considerar critérios como o escopo do alinhamento (global ou local), e os objetivos específicos da análise (como predição estrutural, comparação de sequências ou inferência filogenética). Essas categorias refletem a versatilidade do MSA como técnica analítica fundamental, mas também destacam os desafios computacionais persistentes associados à sua aplicação em contextos complexos e de alto volume de dados [16].

2.3 Sum-of-Pairs (SP)

O método *Sum-of-Pairs* (SP), ou Soma de Pares, é uma das funções objetivo mais utilizadas para a avaliação da qualidade de MSA [15]. O seu objetivo é produzir uma medida quantitativa da similaridade em um alinhamento, levando em consideração todas as comparações possíveis entre os pares de sequências [4]. Devido à sua simplicidade de cálculo e interpretação, o escore SP foi amplamente adotado em diversos estudos sobre alinhamento múltiplo [3].

Conforme mencionado na Seção 2.1, um alinhamento múltiplo de N sequências, $\mathcal{S} = \{S_1, S_2, \ldots, S_N\}$, é representado por uma matriz com N linhas e L colunas, na qual lacunas são inseridas para que todas as sequências tenham o mesmo comprimento L [4]. O método SP avalia a qualidade desse alinhamento somando os escores de todos os alinhamentos par a par induzidos pela matriz. Formalmente, o escore SP para uma única coluna c do alinhamento, denotado por $SP(m_c)$, é definido pela Equação 2.1 [4]:

$$\mathrm{SP}(m_c) = \sum_{1 \le i < j \le N} \mathrm{score}(S_i[c], S_j[c]) \tag{2.1}$$

em que $\mathrm{score}(S_i[c], S_j[c])$ representa o escore da comparação entre os caracteres das sequências N e L na coluna c. A soma é realizada sobre todos os pares distintos de sequências. Os escores de caracteres podem ser definidos por valores para acertos (*matches*), erros (*mismatches*) e lacunas (*gaps*), ou por meio de matrizes de substituição, como PAM e BLOSUM, que modelam a probabilidade de substituições de aminoácidos ao longo do tempo evolutivo [4].

O escore total do alinhamento, SP_{total}, é obtido pela soma dos escores de todas as colunas, conforme apresentado na Equação 2.2 [4]:

$$\mathrm{SP}_{\mathrm{total}} = \sum_{c=1}^{L} \mathrm{SP}(m_c) \tag{2.2}$$

A Figura 2.2 ilustra o *sum-of-pairs*, onde, para cada par de sequências, o alinhamento de caracteres iguais recebe a pontuação zero enquanto o alinhamento de caracteres diferen-

tes ou o alinhamento com lacuna (-) recebe a pontuação 1. Como o objetivo é minimizar o escore total, escores próximos de zero representam alta similaridade das sequências.

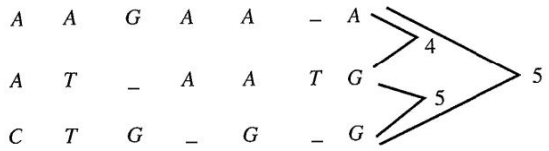

Figura 2.2: Escore de um alinhamento múltiplo com a função Sum of Pairs (SP) = 14 [4].

Apesar de sua ampla aplicação, o método SP possui limitações. Uma das principais é a ponderação igualitária de todos os pares de sequências, o que pode levar à contagem múltipla de eventos evolutivos em ramos densos de uma árvore filogenética. Para mitigar essa distorção foi proposto o método *Weighted Sum-of-Pairs* (WSP) [17], que atribui pesos distintos aos pares de sequências com base em suas relações evolutivas, reduzindo a duplicação de escores.

Embora o método SP apresente limitações, ele permanece como uma ferramenta central na Bioinformática. A sua aplicação oferece a base teórica para o desenvolvimento e a avaliação de novas técnicas de alinhamento. Mesmo com as dificuldades associadas ao cálculo da solução ótima, o SP continua a ser a função objetivo padrão em muitos métodos de otimização de MSA.

2.4 Alinhamento Múltiplo Exato

Um algoritmo exato garante a obtenção do alinhamento que maximiza (ou minimiza, no caso de uma função de distância) a pontuação, dado um conjunto de sequências, uma função de pontuação e penalidades por lacunas. No entanto, é importante destacar que um alinhamento matematicamente ótimo não é necessariamente o "biologicamente verdadeiro". Ao contrário dos algoritmos heurísticos, que geram soluções aproximadas, os algoritmos exatos garantem o alinhamento com a melhor pontuação possível, calculada com base em um sistema de pontuação previamente definido, que normalmente corresponde à função SP [4], que é um modelo simplificado da evolução biológica.

Uma abordagem simples para resolver esse problema é a generalização do algoritmo de programação dinâmica. Ao alinhar N sequências de comprimento médio L, essa abordagem constrói uma matriz multidimensional, com tempo de execução na ordem de $\Theta(L^N)$, tornando-a impraticável para mais do que um pequeno número de sequências.

Uma melhoria significativa foi proposta por Carrillo e Lipman [5], que demonstraram que não é necessário explorar todo o espaço de busca para encontrar o alinhamento ótimo.

Eles definiram limites superior e inferior para a pontuação do alinhamento ótimo, descartando áreas do espaço de busca que não poderiam conter a solução ótima. Contudo, a complexidade de tempo do algoritmo de Carrillo–Lipman ainda é exponencial [5].

Outra alternativa para a busca exata é o algoritmo A^*, que modela o problema como a busca pelo caminho de menor custo em um grafo [4]. Nesse caso, uma função é aplicada para estimar o custo restante, permitindo uma exploração mais eficiente do espaço de busca. A garantia de otimalidade do A^* depende do uso de uma heurística admissível, que nunca superestima o custo real para atingir a solução.

Apesar de garantirem a otimalidade matemática, a alta demanda computacional desses métodos continua a restringir sua aplicabilidade a conjuntos de sequências de tamanho limitado.

2.4.1 Algoritmo *Naïve* (ingênuo)

O algoritmo *naïve* para o alinhamento múltiplo exato é a extensão direta da técnica de programação dinâmica do PSA para múltiplas sequências [4]. O objetivo principal deste algoritmo é determinar um alinhamento global para um conjunto de sequências, otimizado de acordo com um critério de escore pré-definido, tipicamente a soma dos pares (SP) [6].

Considerando um conjunto de N sequências do conjunto $\mathcal{S} = \{S_1, S_2, \ldots, S_N\}$ com comprimentos $L_1, L_2, \ldots, L_N$, o algoritmo opera em um espaço de busca N-dimensional. Nesse espaço, cada ponto $(i_1, i_2, \ldots, i_N)$ representa o alinhamento dos prefixos $S_1[1..i_1], S_2[1..i_2], \ldots, S_N[1..i_N]$ [1]. O ponto de origem $(0, 0, \ldots, 0)$ representa o alinhamento de prefixos vazios, e o escore no ponto final $(L_1, L_2, \ldots, L_N)$ representa o escore do alinhamento global ótimo [2].

Para ilustrar o funcionamento, considere o caso de três sequências ($N = 3$). O algoritmo utiliza uma matriz tridimensional D de dimensões $(L_1 + 1) \times (L_2 + 1) \times (L_3 + 1)$, na qual cada célula $D(i, j, k)$ armazena o escore ótimo do alinhamento dos prefixos $S_1[1..i], S_2[1..j]$ e $S_3[1..k]$. A computação dos escores é feita, iniciando com $D(0, 0, 0) = 0$. Para calcular o escore de uma célula interna $D(i, j, k)$, são consideradas as $2^3 - 1 = 7$ transições possíveis, que correspondem às operações de alinhamento envolvendo os últimos caracteres das sequências ou a inserção de lacunas [4]. Essas operações incluem:

- Alinhar $S_1[i], S_2[j]$ e $S_3[k]$;
- Alinhar $S_1[i]$ e $S_2[j]$, com lacuna em S_3;
- Alinhar $S_1[i]$ e $S_3[k]$, com lacuna em S_2;
- Alinhar $S_2[j]$ e $S_3[k]$, com lacuna em S_1;

- Alinhar apenas $S_1[i]$, com lacunas em S_2 e S_3;
- Alinhar apenas $S_2[j]$, com lacunas em S_1 e S_3;
- Alinhar apenas $S_3[k]$, com lacunas em S_1 e S_2;

Para cada uma dessas operações, é calculado um escore, e o valor de $D(i, j, k)$ é definido como o máximo entre os sete escores possíveis, representando a melhor pontuação para o alinhamento dos prefixos até as posições i, j e k. De forma geral, para N sequências, o cálculo de cada célula da matriz considera $2^N - 1$ transições possíveis. Este processo algorítmico é ilustrado no Algoritmo 2.1.

Algorithm 2.1 Exemplo de Implementação do Algoritmo *Naïve* [2]

```
1:  for i = 1 → L1 do
2:      for j = 1 → L2 do
3:          for k = 1 → L3 do
4:              c_ij = CostMatchMismatch(S1[i], S2[j])
5:              c_ik = CostMatchMismatch(S1[i], S3[k])
6:              c_jk = CostMatchMismatch(S2[k], S3[j])
7:              d1 = D(i − 1, j − 1, k − 1) + c_ij + c_ik + c_jk
8:              d2 = D(i − 1, j − 1, k) + c_ij + γ
9:              d3 = D(i − 1, j, k − 1) + c_ik + γ
10:             d4 = D(i, j − 1, k − 1) + c_jk + γ
11:             d5 = D(i − 1, j, k) + 2 × γ
12:             d6 = D(i, j − 1, k) + 2 × γ
13:             d7 = D(i, j, k − 1) + 2 × γ
14:             D(i, j, k) = min[d1, d2, d3, d4, d5, d6, d7]
15:         end for
16:     end for
17: end for
```

Inicialmente, o algoritmo percorre todas as combinações possíveis de posições i, j e k nas três sequências, com i variando de 1 a L_1, j de 1 a L_2 e k de 1 a L_3, onde L_1, L_2 e L_3 são os comprimentos das sequências. Para cada tripla de índices, calcula-se o custo de alinhamento entre os elementos das sequências utilizando a função CostMatchMismatch. Por exemplo, o custo de alinhar o elemento $S_1[i]$ com $S_2[j]$ é calculado na linha 4, com $c_{ij} = \text{CostMatchMismatch}(S_1[i], S_2[j])$, e de maneira semelhante para os outros pares de sequências, como c_{ik} para $S_1[i]$ com $S_3[k]$ e c_{jk} para $S_2[j]$ com $S_3[k]$. O custo é obtido com matrizes de substituição, no caso de proteínas, utilizando matrizes do tipo PAM ou BLOSUM [2]. Para sequências de DNA ou RNA, o custo pode ser obtido com as matrizes NUC ou com valores fixos para match e mismatch [2]. Com esses custos, o Algoritmo 2.1 então calcula os custos de diferentes transições, considerando a introdução de *gaps*,

representados pelo custo γ. As transições possíveis incluem alinhamentos entre todos os três elementos, como em

$$
\begin{aligned}
d_1 &= D(i-1, j-1, k-1) + c_{ij} + c_{ik} + c_{jk}, \\
d_2 &= D(i-1, j-1, k) + c_{ij} + \gamma, \\
d_5 &= D(i-1, j, k) + 2 \times \gamma,
\end{aligned}
$$

sendo que, na linha 7, d_1 representa a soma dos custos de alinhamento entre todos os três elementos, e d_2 e d_5 representam transições parciais com a introdução de *gaps*. A cada iteração, o algoritmo armazena o custo mínimo entre todas as transições possíveis na matriz $D(i, j, k)$, como na linha 14 $D(i, j, k) = \min[d_1, d_2, d_3, d_4, d_5, d_6, d_7]$, até que o custo total do alinhamento ótimo seja obtido. O resultado é o alinhamento múltiplo das três sequências, minimizando o custo acumulado de correspondências e *gaps* como visto na Figura 2.3.

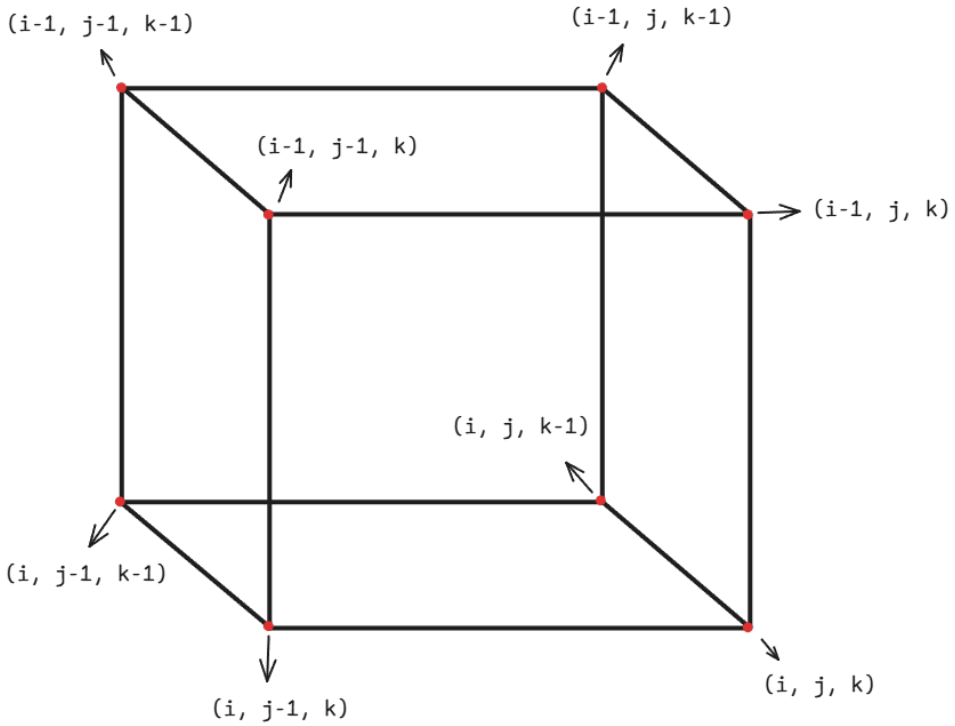

Figura 2.3: Representação do Algoritmo Naïve no plano 3D.

Embora o algoritmo *naïve* garanta a otimalidade da solução, sua principal limitação está na complexidade computacional. Para N sequências de comprimento médio L, a complexidade de tempo e espaço é $\mathcal{O}(L^N)$ [4]. Essa complexidade exponencial torna o algoritmo inviável para mais do que um pequeno número de sequências ou para sequências

longas, devido à alta demanda computacional. Por essa razão, algoritmos heurísticos ou métodos exatos mais eficientes, como o de Carrillo-Lipman [5] ou o algoritmo A^* [18], são preferíveis para problemas de maior escala.

2.5 Algoritmos Heurísticos

Diante da complexidade do problema MSA, algoritmos heurísticos surgem como as abordagens predominantes para encontrar soluções de alta qualidade em um tempo de execução viável. Ao contrário dos algoritmos exatos, esses métodos não garantem a otimalidade, mas exploram o espaço de busca de maneira eficiente para encontrar soluções aproximadas, buscando um equilíbrio entre a eficiência computacional e a qualidade do alinhamento final [7].

2.5.1 Método Progressivo

A abordagem heurística mais utilizada para MSA é o alinhamento progressivo [19]. Essa abordagem decompõe o problema do MSA em uma série de alinhamentos par a par, construindo o alinhamento final de maneira gradual. O processo ocorre em duas etapas principais [3]:

1. **Construção da Árvore Guia:** Inicialmente, todas as sequências são comparadas par a par para o cálculo de uma matriz de distâncias que reflete a similaridade entre elas. Com base nessa matriz, uma "árvore guia", que funciona como uma aproximação da relação filogenética, é construída utilizando algoritmos como o *neighbor-joining* [20]. A topologia da árvore define a ordem hierárquica do alinhamento, iniciando pelos pares mais similares;

2. **Alinhamento Progressivo Guiado:** A segunda etapa consiste no alinhamento propriamente dito. O processo percorre a árvore, alinhando as sequências ou grupos de sequências mais próximos, ou seja, o escore SP otimizado. Quando dois subalinhamentos são fundidos, eles são tratados como perfis, e um alinhamento de perfil com perfil (*profile-profile alignment*) é realizado para criar um novo perfil, associado ao nó ancestral, até que todas as sequências sejam incorporadas [3].

A principal vantagem desta abordagem é a sua velocidade computacional. No entanto, sua natureza gulosa para introduzir uma falha crítica, a propagação de erros. Erros cometidos nas fases iniciais do alinhamento, especialmente entre sequências distantemente relacionadas, são "fixados" e não podem ser corrigidos posteriormente. A política de "uma vez uma lacuna, sempre uma lacuna" (*once a gap, always a gap*) significa que as lacunas

introduzidas para alinhar um subconjunto são propagadas para o resto do processo, o que pode levar a desalinhamentos significativos no resultado [3].

`ClustalW`

O programa `ClustalW` [20], desenvolvido por Thompson, Higgins e Gibson, é um dos algoritmos mais representativos da abordagem progressiva. O processo no `ClustalW` inicia com o cálculo de uma matriz de distâncias, otimizado com uma heurística veloz baseada em *k-tuples*. Em seguida, uma árvore guia é construída com o método de *neighbor-joining* e o alinhamento ocorre de forma progressiva. Para aumentar a sensibilidade, o `ClustalW` introduziu melhorias que se tornaram referência, como a ponderação de sequências, o uso de penalidades de lacuna específicas por posição e a flexibilidade na escolha de matrizes de substituição [20].

`Clustal` Ω

O `Clustal` Ω [21] representa a nova geração da família `Clustal`, desenvolvido para superar a principal limitação de seus predecessores: a escalabilidade . Enquanto o `ClustalW` era limitado a algumas centenas de sequências, o `Clustal` Ω foi projetado para alinhar de forma rápida e precisa conjuntos de dados massivos, sendo capaz de processar centenas de milhares de sequências [21].

A melhoria de desempenho do `Clustal` Ω se deve a duas inovações principais em relação ao `ClustalW`. Primeiramente, a etapa de construção da árvore guia, que era um gargalo computacional, foi otimizada. Em vez de realizar todos os alinhamentos par a par, o `Clustal` Ω utiliza o algoritmo `mbed` [22] para agrupar sequências de forma muito mais rápida. Em segundo lugar, a fase de alinhamento de perfis foi aprimorada com a incorporação de técnicas de modelos ocultos de Markov (HMM), utilizando o motor do HHalign [23]. Essa abordagem baseada em HMM melhora a acurácia do alinhamento, especialmente, ao fundir subalinhamentos distantes [21].

`T-Coffee`

O T-Coffee (Tree-based Consistency Objective For alignmEnt Evaluation) [24] é um método de alinhamento múltiplo de sequências baseado em consistência, desenvolvido para aumentar a acurácia em comparação com abordagens progressivas tradicionais. Sua principal inovação reside na construção de uma biblioteca de PSA, que combina informações oriundas de alinhamentos globais, como os produzidos pelo `ClustalW` com informações obtidas a partir de alinhamentos locais.

Diferentemente de métodos progressivos convencionais, que dependem fortemente de uma única estratégia de alinhamento, o T-Coffee avalia a consistência com que cada par de resíduos é alinhado ao longo de toda a biblioteca. Pares de resíduos repetidamente alinhados de maneira semelhante em diferentes fontes recebem um peso maior, refletindo sua confiabilidade. Essa informação de consistência é então utilizada para guiar o alinhamento progressivo final.

Por exemplo, considere três sequências A, B e C. Se o resíduo A_i estiver alinhado com B_j em um alinhamento global, e B_j estiver alinhado com C_k em um alinhamento local, então o T-Coffee infere, por transitividade, que A_i deve estar alinhado com C_k. Se esse alinhamento indireto também for observado em outras combinações, o par A_i–C_k receberá um peso elevado, indicando alta consistência [24].

Essa estratégia baseada em múltiplas evidências torna o T-Coffee particularmente robusto para alinhar sequências distantemente relacionadas, cenário em que métodos puramente progressivos tendem a cometer erros significativos na ordenação e posicionamento de resíduos homólogos.

2.5.2 Método Iterativo

Para superar as limitações do método progressivo, foram desenvolvidas estratégias iterativas que refinam um alinhamento inicial (frequentemente obtido por métodos progressivos) por meio de iterações sucessivas [3]. Essas estratégias podem envolver o realinhamento de subconjuntos de sequências ou a reconstrução da árvore guia para repetir o processo de alinhamento, continuando até que um critério de convergência seja alcançado. Dentro das abordagens iterativas, existem diferentes estratégias, como o refinamento da árvore guia (utilizado em programas como `MAFFT` e `MUSCLE`), e a divisão e realinhamento do alinhamento existente [7]. Além disso, algoritmos estocásticos, como os Algoritmos Genéticos (GAs), podem ser empregados para explorar o espaço de soluções e melhorar a qualidade dos alinhamentos ao longo de gerações [14, 25].

A principal vantagem dos métodos iterativos é sua capacidade de melhorar a acurácia do alinhamento, corrigindo erros dos estágios iniciais e evitando a convergência para ótimos locais [3]. Embora possam ter um custo computacional maior devido às múltiplas iterações, ferramentas como `MAFFT` e `MUSCLE` frequentemente apresentam uma acurácia superior à do `ClustalW`, com maior velocidade computacional [7]. Em resumo, enquanto as técnicas progressivas geram um alinhamento inicial de forma rápida, as técnicas iterativas focam em refinar esse alinhamento para aumentar a precisão.

MAFFT

O programa MAFFT (Multiple Alignment using Fast Fourier Transform) [26] destaca-se por sua notável velocidade na execução de MSA. Desenvolvido por Katoh et al. [27], o MAFFT é amplamente utilizado para grandes conjuntos de dados, servindo como base para análises filogenéticas robustas. O seu diferencial reside na estratégia de refinamento iterativo combinada com o uso de *Fast Fourier Transform* (FFT) para identificar rapidamente segmentos similáres. Nessa abordagem, as sequências são convertidas em vetores baseados em suas propriedades físico-químicas, e a FFT identifica regiões de alta similaridade que servem como âncoras para o alinhamento. Esse método confere ao MAFFT uma velocidade significativamente maior do que a de abordagens tradicionais, sem grande perda de precisão [26]. O MAFFT disponibiliza diversas modalidades, como o FFT-NS-i (iterativo), que oferecem flexibilidade para diferentes tipos de dados.

MUSCLE

O programa MUSCLE (MUltiple Sequence Comparison by Log-Expectation) [28] é uma abordagem iterativa estruturada em três fases, projetada para refinar progressivamente a qualidade do alinhamento. A primeira fase, alinhamento progressivo preliminar, gera uma primeira versão do alinhamento de forma rápida. A segunda fase, melhoramento progressivo, utiliza as informações do primeiro alinhamento para construir uma árvore guia mais precisa e, com base nela, gerar um segundo alinhamento de maior qualidade.

A terceira, e principal fase, é a de refinamento iterativo. Nela, a árvore guia é particionada recursivamente, dividindo o alinhamento em dois subgrupos. Os perfis correspondentes a esses subgrupos são então realinhados entre si. A qualidade do novo alinhamento completo é avaliada com uma função de pontuação (tipicamente *Sum-of-Pairs*), e o processo é repetido, mantendo-se o alinhamento de maior pontuação a cada iteração, até que não haja mais melhorias. Essa fase final confere ao MUSCLE uma vantagem sobre os métodos puramente progressivos, ao permitir a correção de erros introduzidos nas etapas iniciais, resultando em maior acurácia [28].

2.5.3 Comparação entre os Algoritmos para MSA Heurístico

Apesar dos avanços recentes, os métodos heurísticos ainda operam em uma margem entre precisão, velocidade e consumo de recursos computacionais, apresentando modos de falha bem documentados [29, 30]. O ClustalW, por exemplo, é conhecido por sua rapidez, embora costume apresentar menor acurácia em comparação a abordagens mais modernas [3, 20].

Tabela 2.1: Comparação dos Programas Heurísticos Dominantes

Nome do Programa	**Estratégia Heurística Central**	**Ponto Forte Principal**	**Limitação Principal**	**Caso de Uso Típico**
ClustalW [20]	Alinhamento Progressivo	Rápido e computacionalmente leve; amplamente utilizado e compreendido.	Suscetível à propagação de erros (guloso); menor precisão com sequências divergentes.	Análises rápidas e preliminares de conjuntos de dados de tamanho moderado.
MUSCLE [28]	Alinhamento Progressivo + Refinamento Iterativo	Excelente equilíbrio entre velocidade e precisão; muito mais rápido que os métodos de consistência.	A precisão pode ser inferior à dos métodos de consistência em casos difíceis.	Alinhamento de conjuntos de dados grandes (centenas a milhares de sequências) onde a precisão é importante.
T-Coffee [24]	Baseado em Consistência	Alta precisão, especialmente para sequências distantemente relacionadas; combina múltiplas fontes de informação.	Lento e consome muita memória; impraticável para conjuntos de dados muito grandes.	Alinhamento de alta qualidade de pequenos conjuntos de sequências (<100) para análise filogenética detalhada.
MAFFT [26]	Progressivo, Iterativo e Baseado em Consistência (adaptativo)	Extremamente rápido e escalável (modos FFT-NS), mas também oferece modos de alta precisão (L-INS-i).	A precisão nos modos mais rápidos é inferior à dos modos de consistência.	Ferramenta versátil adequada tanto para alinhamentos rápidos de genomas inteiros como para alinhamentos precisos de famílias de proteínas.
Clustal Ω [21]	Alinhamento Progressivo com Árvores Guia aprimoradas	Altamente escalável para dezenas de milhares de sequências; boa precisão para o seu nível de velocidade.	A precisão é geralmente comparável, mas não superior, à dos melhores métodos de consistência em benchmarks menores.	Alinhamento de conjuntos de dados em larga escala, como os encontrados em estudos de metagenómica ou filogenómica.

Em resposta a essas limitações, o `MUSCLE` foi desenvolvido para alcançar um equilíbrio superior. Ele é consideravelmente mais rápido do que os métodos baseados em consistência, mantendo, ao mesmo tempo, uma acurácia comparável em diversos contextos [28]. Em contraste, métodos como o `T-Coffee` estão entre os mais precisos, mas esse ganho em qualidade vem acompanhado de um custo computacional elevado, o que limita sua aplicabilidade a conjuntos de dados muito extensos [24].

Ferramentas mais recentes, como o `Clustal Ω` e o `MAFFT`, procuram contornar essas limitações por meio de estratégias adaptativas. Esses programas são capazes de selecionar automaticamente o algoritmo mais apropriado, levando em consideração o tamanho e a complexidade do conjunto de dados. Dessa forma, alternam entre modos de alinhamento progressivo mais rápidos e refinamentos iterativos ou por consistência, que, embora mais lentos, tendem a ser mais precisos [31, 26]. A Tabela 2.1 apresenta uma comparação

resumida entre os métodos discutidos nesta seção.

Em geral, esses métodos integram etapas que resolvem, subproblemas bem definidos. A etapa inicial em ferramentas como o `ClustalW` ou o `MUSCLE` consiste na realização de todos os PSA possíveis, utilizando algoritmos de programação dinâmica para resolver esses casos de forma ótima [3, 4]. Assim, mesmo métodos heurísticos partem de soluções exatas em partes das sequências, e a aproximação se dá na forma como essas soluções são combinadas. O processo de construção do alinhamento múltiplo é guiado por uma árvore filogenética, sendo ela própria uma aproximação, seguindo uma estratégia gulosa, que prioriza decisões locais sem considerar globalmente o espaço de soluções.

Esse cenário revela como algoritmos heurísticos dependem, de forma estruturada, de soluções exatas para enfrentar problemas de maior escala, aproveitando seus componentes fundamentais de maneira eficiente.

O aprimoramento contínuo desses métodos exige uma avaliação objetiva da acurácia. Para isso, pesquisadores recorrem a "padrões-ouro" com os quais possam comparar os alinhamentos gerados. Idealmente, esse padrão seria o alinhamento matematicamente ótimo, produzido por algoritmos exatos [8].

No entanto, há uma limitação conceitual importante. Os métodos de alinhamento maximizam funções de escore, como o SP, que são apenas aproximações da verdadeira homologia biológica. A escolha de matrizes de substituição e, especialmente, dos parâmetros de penalidade por abertura e extensão de lacunas pode ter mais impacto no resultado do que a escolha do próprio algoritmo [32]. Como não existe um conjunto universal de parâmetros corretos, a minimização de uma função matemática não garante, por si só, um alinhamento biologicamente ótimo [33].

2.6 Algoritmo de Busca A*

Ao invés de procurar uma solução "aceitável", os algoritmos exatos são projetados para encontrar a solução que otimiza uma função objetivo predefinida, retornando a melhor solução [18]. Apesar das limitações práticas impostas pela complexidade computacional, tais algoritmos têm papel teórico essencial: servem como padrão-ouro para avaliar heurísticas [34]. Dada uma instância específica (um conjunto de sequências e um esquema de pontuação), a solução exata define o limite superior de acurácia, pois nenhuma heurística pode, por definição, produzir um alinhamento com melhor pontuação. Assim, a distância entre a solução heurística e a ótima é uma medida objetiva de sua eficácia.

Para contornar a barreira da programação dinâmica no problema de MSA, o algoritmo de busca A^* se apresenta como uma alternativa exata viável. Esse algoritmo reformula o

MSA como a busca do caminho de menor custo em um grafo, guiado por uma função de avaliação [18], conforme apresentado na Equação 2.3:

$$f(n) = g(n) + h(n) \tag{2.3}$$

na qual $g(n)$ representa o custo real do caminho até o nó n, e $h(n)$ é uma estimativa heurística do custo restante até o objetivo.

A garantia de otimalidade do A^* depende da admissibilidade de $h(n)$, isto é, da condição de que $h(n)$ nunca superestime o custo real restante. Sob essa condição, o A^* encontra sempre o alinhamento ótimo [35]. No entanto, a eficiência da busca depende fortemente da qualidade da heurística: quanto mais precisa, maior a capacidade de poda do espaço de busca, reduzindo o tempo de execução.

A aplicabilidade prática do algoritmo A^* em problemas de bioinformática foi empregada por Lermen e Reinert [18], que mostraram como estratégias de limites inferiores (*lower bounds*) e superiores podem restringir o espaço de busca sem comprometer a otimalidade [18]. Além disso, Ikeda e Imai [36] discutiram o uso de heurísticas mais potentes em casos n-dimensionais anos antes, reforçando que a transição para projeções de ordem superior é um caminho natural para otimizar a exploração de grafos de estados em MSA [36].

2.6.1 Algoritmo PA-Star

O *PA-Star* é uma extensão paralela do A^* que modela o MSA como um problema de busca em grafo N-dimensional [10]. Como se trata de uma variante do A^*, o PA-Star utiliza a Equação 2.3.

A heurística comumente adotada para a estimativa da função g é a $h_{2,\text{all}}$, que calcula a soma dos alinhamentos ótimos entre todos os pares de sequências, conforme mostrado na Equação 2.4:

$$h_{2,\text{all}} = \sum_{1 \leq i < j \leq N} \text{MSA}(S_i, S_j) \tag{2.4}$$

Essa heurística é admissível [12, 37] e desempenha papel fundamental na redução do espaço de busca. O algoritmo mantém duas estruturas principais: a `OpenList`, que contém os nós ainda a serem expandidos, e a `ClosedList`, com os nós já processados. A expansão dos nós segue a prioridade imposta pelo menor valor de $f(n)$, assegurando a otimalidade da solução [9, 10].

Para aumentar o desempenho, a etapa de busca do PA-Star foi paralelizada, utilizando uma divisão do espaço de busca com base em uma função de *hash* sensível à localidade,

que distribui os nós entre *threads* para minimizar comunicação e maximizar o uso de cache [9, 10]. Adicionalmente, a `OpenList` foi estruturada como uma fila multi-índice, e a `ClosedList`, como um mapa eficiente. Assim, foram utilizados *templates* em `C++` para especializar os nós, reduzindo o uso de memória.

Mesmo com as otimizações, o consumo de memória continuava sendo um desafio. Isso motivou o desenvolvimento do *Disk-Assisted PA-Star* (DAPA), que armazena partes das listas em disco, permitindo o processamento de instâncias maiores [10]. Apesar de ter atingido o objetivo de comparar conjuntos de sequências maiores, o DAPA apresentou problemas no desempenho, devido aos acessos a disco no caso de instâncias complexas.

Com o surgimento de processadores *multicore* assimétricos, foi proposto o *PA-Star2*, uma versão projetada para arquiteturas heterogêneas, compostas de processadores rápidos e processadores lentos. Ele introduz uma função de *hash* de dois níveis, capaz de balancear a carga de trabalho entre núcleos com diferentes capacidades de processamento [38].

2.6.2 Impacto da Heurística no Desempenho

A discussão sobre a eficiência do A^* leva à questão central da escolha heurística. Heurísticas simples e relativamente rápidas, como a $h_{2,\text{all}}$, fornecem limites inferiores admissíveis, mas frequentemente imprecisos, forçando o A^* a explorar muitos nós.

Por outro lado, heurísticas melhores, como $h_{3,\text{all}}$ [12, 37], baseada em trios de sequências, podem proporcionar limites muito mais estritos, conforme na apresentado Equação 2.5:

$$h_{3,all} = \sum_{1 \leq i < j < k \leq N} \text{MSA}(S_i, S_j, S_k) \tag{2.5}$$

Apesar disso, o custo computacional de avaliação de $h_{3,\text{all}}$ é significativamente maior, e pode anular os ganhos obtidos pela poda mais agressiva do espaço de busca [12]. Portanto, a escolha da heurística constitui um problema de otimização por si só: o tempo economizado na busca deve superar o tempo gasto no cálculo da heurística. Isso mostra que, mesmo em algoritmos exatos, a introdução de aproximações inteligentes é inevitável. O desempenho prático do A^* é fortemente influenciada pela qualidade da heurística utilizada.

Capítulo 3

Trabalhos Relacionados

O problema de MSA, dada a sua complexidade computacional NP-Completo, motivou o desenvolvimento de uma vasta e diversificada gama de algoritmos ao longo das últimas décadas. Assim sendo, para contextualizar a contribuição deste trabalho, este capítulo realiza uma análise do estado da arte, apresentando, para cada abordagem, suas estratégias, vantagens e limitações [7].

3.1 Categorização dos Algoritmos de MSA

A estrutura aqui apresentada baseia-se na pesquisa conduzida por Zhang *et al.* [7], que dividiram as estratégias em alinhamento progressivo, métodos iterativos, divisão e conquista, baseadas em meta-heurísticas e modelos estatísticos, e baseadas em aprendizado de Máquina, como visto na Figura 3.1. A seguir, as categorias são explicadas de maneira sucinta:

- **Métodos Progressivos e Métodos Iterativos**: Definidos nas Seções 2.5.1 e 2.5.2, respectivamente, são os métodos mais utilizados em MSA;
- **Métodos de Dividir para Conquistar:** Esta estratégia consiste em decompor um problema complexo de *Multiple Sequence Alignment* em múltiplos subproblemas menores, que são resolvidos de forma independente e depois combinados. Essa abordagem apresenta vantagens em termos de desempenho e é particularmente adequada para a combinação com técnicas de processamento paralelo para melhorar a eficiência [7]. Exemplos de ferramentas que utilizam esta técnica são MAGUS [39] e FMAlign [40];
- **Abordagens Baseadas em Meta-heurísticas e Modelos Estatísticos:** Esta categoria abrange um conjunto de algoritmos iterativos estocásticos que buscam

soluções viáveis para problemas NP-Completo. Entre eles, destacam-se as Meta-heurísticas, que incluem o Recozimento Simulado (*Simulated Annealing*), Algoritmos Genéticos (como o SAGA [14]), e a Inteligência de Enxame, e os Modelos Ocultos de Markov (HMM), que utilizam um modelo de análise estatística para descrever o processo de alinhamento. Ferramentas como ProbCons [41], MSAProbs e Clustal Omega [31] são exemplos que empregam HMM;

- **Abordagens Emergentes com Aprendizado de Máquina:** O uso de aprendizado de máquina para *Multiple Sequence Alignment* continua em um estágio inicial de desenvolvimento [7]. Abordagens baseadas em Aprendizado por reforço, como *Q-learning* e A3C, têm sido exploradas. No entanto, o campo enfrenta desafios significativos, como a falta de amostras de alinhamento ótimas para treinamento, a ausência de uma função de perda adequada e a dificuldade em construir modelos generalizáveis para sequências de comprimentos e quantidades variáveis;

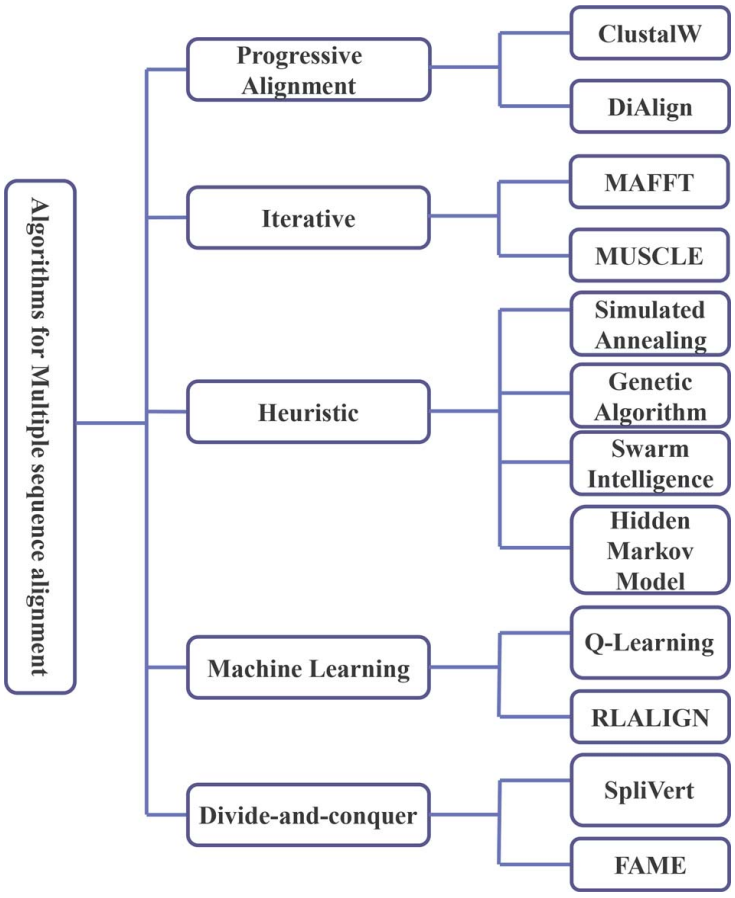

Figura 3.1: Classificação dos Algoritmos de MSA [7]

Tabela 3.1: Análise Comparativa dos Algoritmos de Alinhamento Múltiplo de Sequências

Categoria	Estratégia Central	Vantagens	Desvantagens	Exemplos
Métodos Progressivos	Construção de uma árvore-guia seguida por um alinhamento gradual das sequências de acordo com a árvore.	Simples e eficiente; baixo consumo de memória.	Problema "uma vez uma lacuna, sempre uma lacuna"; propagação de erros das fases iniciais.	ClustalW, T-Coffee, Kalign.
Métodos Iterativos	Otimização contínua de um alinhamento inicial por meio de iterações sucessivas até a convergência.	Melhora a precisão do resultado; boa robustez.	Pode convergir para ótimos locais.	MAFFT, MUSCLE, SATé.
Métodos de Dividir e Conquistar	Decomposição do problema principal em múltiplos subproblemas menores e independentes.	Bom desempenho geral; adequado para processamento paralelo.	A qualidade da fusão dos subproblemas pode ser um desafio.	MAGUS, FMAlign, SpliVert.
Meta-heurísticas e Modelos Estatísticos	Uso de algoritmos estocásticos (GA, SA) ou modelos estatísticos (HMM) para explorar o espaço de soluções.	Flexibilidade na busca; capacidade de escapar de ótimos locais.	Convergência pode ser lenta; parametrização complexa.	SAGA (GA), ProbCons (HMM), Clustal Omega (HMM).
Aprendizado de Máquina	Uso de técnicas como aprendizado por reforço para gerar alinhamentos, ainda em estágio inicial.	Potencial para aprender modelos complexos da evolução diretamente dos dados.	Falta de amostras ótimas para treinamento; dificuldade em construir modelos generalizáveis.	Abordagens baseadas em Q-learning e A3C.

3.2 Estratégias Paralelas para MSA

A crescente demanda por análise de grandes volumes de dados biológicos impulsionou a adaptação de algoritmos de MSA para arquiteturas de processamento paralelo. O objetivo central dessas abordagens é reduzir o custo computacional do problema, permitindo que análises complexas sejam executadas em tempo viável. As soluções exploram inúmeras técnicas, que vão desde a formulação do problema sob novas perspectivas até a otimização de baixo nível de ferramentas já usadas.

A abordagem apresentada por Rubio-Largo et al. [42], reformula o MSA como um problema de otimização multiobjetivo. O escopo da pesquisa foi o alinhamento de grandes conjuntos de sequências, partindo da premissa de que a qualidade de um alinhamento não deve ser medida por um único escore de similaridade. Assim, o objetivo do trabalho foi desenvolver uma ferramenta que otimizasse simultaneamente dois critérios distintos: a acurácia, quantificada pelo escore *Weighted Sum-of-Pairs* (WSP), e o número de colunas totalmente conservadas, que frequentemente indicam resíduos essenciais para a função molecular. Para tal, os autores desenvolveram a H4MSA, uma meta-heurística memética híbrida e paralela que combina um algoritmo de otimização em enxames com buscas

locais. Os resultados mostraram que a paralelização da busca permitiu à ferramenta alcançar boa escalabilidade, operando cerca de 25 vezes mais rápido (com um *speedup* de 24,98 e eficiência média de 78,12%) em um ambiente com 32 núcleos. Em termos de precisão, como se trata de um algoritmo heurístico, a ferramenta apresentou uma melhoria média na conservação dos alinhamentos de aproximadamente 14,98% no maior conjunto de dados avaliado (1.056 sequências).

Em contraste com a criação de novas meta-heurísticas, outra vertente de pesquisa possui seu foco na otimização de algoritmos heurísticos já estabelecidos, estendendo sua aplicabilidade a conjuntos de dados maiores. Nesse contexto, o trabalho de Nakamura et al. [26] buscou superar as limitações de escalabilidade do MAFFT. O objetivo específico era tornar os modos de alta precisão do programa, como o L-INS-i, computacionalmente viáveis para dezenas de milhares de sequências. A técnica empregada foi uma paralelização híbrida, utilizando MPI para distribuição de tarefas entre nós de um cluster e Pthreads para exploração de múltiplos núcleos em cada nó. O método focou em acelerar a etapa de cálculo de distância *all-to-all*, o principal gargalo computacional do algoritmo. Como resultado, a implementação paralela do MAFFT demonstrou uma aceleração significativa em que o tempo computacional para a etapa de alinhamento *all-to-all* decresce quase linearmente de acordo com o aumento de núcleos de processamento. Aliado ao ganho de velocidade, o consumo de memória foi reduzido de 26,0 GB para 5,72 GB em comparação a versão convencional para alinhar 4.000 sequências, o que viabilizou o processamento de 50.000 a até cerca de 90.000 sequências simultaneamente.

Seguindo uma abordagem algorítmica distinta, baseada na estratégia de dividir e conquistar, o trabalho de Smirnov [39] buscou um avanço em escalabilidade para a ferramenta MAGUS. O escopo do trabalho foi o alinhamento de até um milhão de sequências, mantendo a alta acurácia característica do método. Para alcançar esse objetivo, a técnica central foi o aprimoramento da abordagem de dividir para conquistar com a introdução de recursão e paralelismo de nós. O método decompõe o problema em subproblemas menores de forma recursiva, alinhando-os de forma independente e em paralelo antes de mesclar os resultados. O Recursive MAGUS apresentou ser capaz de alinhar conjuntos de dados de grande escala em tempo computacional razoável, concluindo que a combinação de recursão com paralelismo de nós é uma solução poderosa para os desafios impostos por *datasets* de grande volume.

Enquanto o MAGUS tem seu foco no alinhamento de inúmeros sequências, um desafio distinto surge quando lidamos com sequências individuais extremamente longas, como genomas completos. Para enfrentar esse problema, Zhang et al. [40] propuseram o FMAlign2, um método projetado para ser rápido e eficiente em uso de memória, mesmo ao lidar com dados na ordem de milhões de pares de bases, um cenário geralmente inviável

para ferramentas tradicionais de MSA. A abordagem central do FMAlign2 é a estratégia de divisão vertical, que segmenta as sequências com base em cadeias parciais, pequenas regiões homólogas que servem como pontos de ancoragem. A partir dessas âncoras, o alinhamento global é dividido em diversos subproblemas independentes, possibilitando a execução de múltiplos alinhamentos parciais em paralelo com ferramentas externas, como MAFFT e HAlign. Com essa estratégia, o FMAlign2 superou as limitações de métodos anteriores ao escalar para genomas completos com eficiência. Seus resultados reforçam que estratégias de segmentação baseadas em propriedades intrínsecas das sequências são importantes para viabilizar o alinhamento múltiplo em larga escala, especialmente no contexto de genômica de alto volume.

Além das estratégias de paralelização em nível de processo ou tarefa, otimizações que exploram a arquitetura do processador também se mostram eficazes. O trabalho de Lassmann [43] na reescrita do Kalign exemplifica essa abordagem. O objetivo foi melhorar a velocidade da ferramenta para dezenas de milhares de sequências sem perdas significativas de acurácia. O método implementado no Kalign 3 foi duplo: em nível algorítmico, a construção da árvore-guia foi substituída por uma abordagem mais rápida de *sequence embedding* com *bi-secting K-means*; em nível de programação, o cálculo de distâncias foi otimizado com instruções SIMD. Os resultados mostraram um ganho de desempenho de uma ordem de magnitude, concluindo que a modernização de componentes algorítmicos, aliada a otimizações de baixo nível, é importante para o desenvolvimento de ferramentas de bioinformática de alto desempenho.

A necessidade de respostas rápidas em cenários de saúde pública motivou o desenvolvimento de ferramentas altamente especializadas. O ViralMSA, proposto por Moshiri [44], foi criado para o alinhamento em tempo real de dezenas de milhares de genomas virais para o monitoramento de pandemias. A sua alto desempenho advém de uma mudança de paradigma: em vez de um alinhamento múltiplo tradicional, o ViralMSA utiliza uma abordagem de alinhamento guiado por referência, delegando a tarefa computacionalmente intensiva a mapeadores de leitura (*read mappers*) paralelos e otimizados, como o Minimap2 [45]. O resultado foi uma ferramenta ordens de magnitude mais rápida para este escopo, concluindo que, para conjuntos de sequências proximamente relacionadas, o alinhamento guiado por referência contorna os gargalos tradicionais e viabiliza a análise genômica em tempo real para vigilância epidemiológica.

Enquanto as abordagens anteriores se concentram em otimizar a velocidade de métodos heurísticos, uma frente de pesquisa distinta dedica-se ao desafio de tornar os algoritmos exatos, que garantem a otimalidade da solução, computacionalmente viáveis. Nessa área, o trabalho de Sundfeld et al. [10] foi desenvolvido, focando na paralelização do algoritmo de busca A^*. O escopo do trabalho foi atacar os dois principais gargalos dos métodos

exatos: o tempo de execução e, de forma crítica, o consumo excessivo de memória. O objetivo era, portanto, reduzir esses custos para ampliar o alcance dos métodos exatos as instâncias mais complexas. A técnica central foi o desenvolvimento da ferramenta PA-Star, que modela o MSA como a busca pelo caminho de menor custo em um grafo [10]. O paralelismo foi introduzido por meio de uma função de hash sensível à localidade, que distribui os nós do grafo entre múltiplos threads para minimizar a comunicação e maximizar o uso de cache [9]. Adicionalmente, o método foi estendido no Disk-Assisted PA-Star (DAPA), uma contribuição significativa que gerencia o alto consumo de memória ao utilizar o armazenamento em disco para excedentes das listas de busca, permitindo processar instâncias que de outra forma falhariam [10]. Os resultados apresentaram que essa combinação de paralelismo e gerenciamento inteligente de memória permitiu resolver instâncias maiores do que era possível anteriormente com métodos exatos, concluindo que esta é uma estratégia viável para expandir a aplicabilidade da busca pela solução ótima.

A Tabela 3.2 sintetiza as contribuições e as estratégias centrais dos trabalhos discutidos.

3.3 Análise Comparativa de Ferramentas MSA

A avaliação objetiva do desempenho dos algoritmos é importante para que os usuários escolham as ferramentas mais adequadas e para que os desenvolvedores identifiquem pontos de aprimoramento. Estudos de *benchmark* são importantes para mapear o estado da arte e entender as relações entre acurácia, velocidade e uso de recursos computacionais.

Nesse contexto, o trabalho de Reddy and Fields [11] oferece uma análise de desempenho comparativa e sistemática de ferramentas de MSA amplamente utilizadas, como Clustal Omega, MAFFT e MUSCLE. O objetivo do estudo foi fornecer uma avaliação quantitativa e imparcial para guiar pesquisadores na escolha do algoritmo mais adequado às suas necessidades. A metodologia envolveu a execução das ferramentas sobre os conjuntos de dados de referência do *benchmark* BAliBASE [46], baseados em estruturas tridimensionais de proteínas e considerados "padrão-ouro". A qualidade dos alinhamentos resultantes foi aferida por métricas padrão, como o *SP Score* e o *TC Score*, que medem a recuperação de pares de resíduos e de colunas inteiras, respectivamente. Os resultados da análise quantificaram as diferenças de desempenho, evidenciando os compromissos a cada algoritmo. A principal conclusão do estudo é a de que não existe uma única ferramenta universalmente superior; a escolha ótima depende de um balanço entre acurácia, velocidade e uso de recursos computacionais, e outros fatores avaliados. Trabalhos de *benchmark* como este são indispensáveis para avaliar a tomada de decisão na comunidade científica.

Tabela 3.2: Quadro Comparativo de Estratégias Paralelas para o MSA

Artigo	**Tipo**	**Objetivo Principal**	**Classificação**	**Técnica Paralela Utilizada**
Rubio-Largo et al. [42]	Heurístico	Otimizar simultaneamente múltiplos critérios (acurácia e conservação) para grandes conjuntos de sequências.	Meta-heurística Memética Multiobjetivo.	Memória compartilhada com OpenMP.
Nakamura et al. [26]	Heurístico	Tornar os modos de alta precisão do MAFFT escaláveis para conjuntos de dados massivos (>50.000 sequências).	Método Iterativo.	Modelo híbrido com MPI (memória distribuída) e Pthreads (memória compartilhada).
Smirnov [39]	Heurístico	Aumentar a escalabilidade do MAGUS para alinhar até 1 milhão de sequências, mantendo alta acurácia.	Divisão e Conquista.	Paralelismo de Tarefas.
Zhang et al. [40]	Heurístico	Alinhar sequências ultralongas (milhões de pares de bases) de forma rápida e eficiente.	Divisão e Conquista.	Alinhamento paralelo de segmentos independentes. (Paralelismo de Tarefas)
Lassmann [43]	Heurístico	Melhorar a velocidade do Kalign para lidar com dezenas de milhares de sequências sem perda de acurácia.	Método Progressivo.	Paralelismo de dados ao nível de instrução (SIMD).
Moshiri [44]	Heurístico	Viabilizar o alinhamento em tempo real de dezenas de milhares de genomas virais durante epidemias.	Método Iterativo Determinístico.	Delegação de processamento a mapeadores de leitura (*read mappers*) paralelos.
Sundfeld et al. [10]	Exato	Reduzir o tempo de execução e o consumo de memória do A* em MSA de larga escala.	Método Exato com Heurística Admissível	Paralelismo de dados com Hash Sensível à Localidade (Paralelismo de Threads).
Esta Dissertação	**Exato**	**Aumentar a eficiência computacional (velocidade) de um método exato (PA-Star), sem sacrificar a otimalidade.**	Método Exato com Heurística Admissível	**Paralelização do cálculo da função heurística (Paralelismo de Threads).**

3.4 Síntese e Lacunas Identificadas

A revisão da literatura revela um campo de pesquisa ainda ativo, com avanços expressivos principalmente em métodos heurísticos. Em contraste, a pesquisa em métodos exatos é mais árdua, uma vez que enfrenta diretamente a explosão combinatória do espaço de busca. Ainda assim, observa-se um progresso contínuo nessa vertente, impulsionado principalmente pela aplicação de algoritmos baseados em A^*, que permitem garantir otimalidade desde que apoiados por heurísticas admissíveis, capazes de fornecer limites inferiores mais precisos [47].

Nesse contexto, a qualidade da heurística desempenha um papel importante no desempenho dos métodos exatos, pois determina diretamente a capacidade de poda do espaço de busca. Embora os métodos heurísticos apresentem elevada eficiência prática, eles não fornecem garantias de otimalidade, e a diferença entre a solução obtida e a solução ótima constitui uma métrica objetiva para avaliar sua eficácia [34]. Por outro lado, métodos exatos baseados em A^* oferecem alinhamentos ótimos, porém seu custo computacional é fortemente influenciado pela heurística empregada.

A análise dos trabalhos existentes revela uma predominância quase absoluta do uso da heurística $h_{2,\text{all}}$ em abordagens exatas baseadas em A^*, como observado em algoritmos como PA-Star, DAPA e PA-Star2. Tal escolha é justificada pelo baixo custo computacional dessa heurística e pela sua integração relativamente simples ao processo de busca. Contudo, ao considerar apenas informações derivadas de alinhamentos par-a-par, $h_{2,\text{all}}$ ignora interações de ordem superior entre as sequências, o que pode resultar em limites inferiores pouco estritos e, consequentemente, em um aumento significativo do espaço de busca explorado [12].

Apesar do reconhecimento teórico de que heurísticas baseadas em trios, como a $h_{3,\text{all}}$, são potencialmente mais informativas e capazes de fornecer limites inferiores mais precisos [47], observa-se uma lacuna importante na literatura: a ausência de estudos sistemáticos que avaliem seu impacto prático em *benchmarks* padronizados, como o BAliBASE [46]. Em particular, não há uma análise abrangente que investigue o equilíbrio entre o aumento do custo computacional associado ao cálculo de $h_{3,\text{all}}$ e os possíveis ganhos em termos de redução do espaço de busca, tempo de execução e consumo de memória.

A Tabela 3.3 sintetiza os principais trabalhos revisados que empregam estratégias baseadas em A^* para o MSA, destacando suas contribuições e limitações. Nota-se que os avanços mais recentes concentram-se predominantemente em aspectos de engenharia computacional, como paralelização, uso de armazenamento secundário e adaptação a arquiteturas modernas, mantendo inalterada a heurística. Essa evidência reforça a relevância de investigar alternativas heurísticas mais informativas como uma via complementar para melhorar a eficiência de métodos exatos.

Tabela 3.3: Comparação de trabalhos relacionados ao uso do A^* em MSA

Trabalho	**Contribuição Principal**	**Heurística Utilizada**	**Limitação Identificada**
PA-Star	Paralelização do A^* com uso de função hash sensível à localidade	$h_{2,\text{all}}$	Alto consumo de memória, mesmo em paralelo
DAPA	Extensão do PA-Star com suporte a armazenamento em disco	$h_{2,\text{all}}$	Desempenho limitado por operações de I/O
PA-Star2	Adaptação a arquiteturas heterogêneas multicore	$h_{2,\text{all}}$	Complexidade na gestão de balanceamento de carga
Este Trabalho	Implementar uma nova heurística mais informativa baseada em trios e paralelizar o seu cálculo	$h_{3,\text{all}}$	Custo computacional reduzido porém consumo de memória ainda alto

Dessa forma, a lacuna relacionada à avaliação sistemática da heurística $h_{3,\text{all}}$ conecta-se diretamente à motivação e aos objetivos desta Dissertação, que se propõe a realizar uma análise abrangente dessa heurística no contexto do PA-Star, comparando-a com a tradicional $h_{2,\text{all}}$ a fim de avaliar seu impacto real na eficiência computacional de métodos exatos para o problema de MSA.

Capítulo 4

Projeto das Otimizações para a Ferramenta PA-Star

Este capítulo descreve as decisões tomadas no projeto das otimizações propostas para a ferramenta *PA-Star*, visando reduzir o tempo de execução e mantendo a obtenção do resultado ótimo. A presente Dissertação de Mestrado possui duas contribuições principais:

1. A paralelização da etapa de cálculo da heurística admissível em ambientes de memória compartilhada para reduzir o tempo de execução. Adotamos uma estratégia *multithreaded* que inicialmente determina quantas e quais comparações serão feitas. Com isso, utilizamos uma estratégia particionada para dividir o trabalho entre as *threads*, sem *overhead* de comunicação.

2. A incorporação da função heurística $h_{3,\text{all}}$ baseada em trios de sequências na ferramenta PA-Star, visando limitar mais o espaço de busca do que a função heurística original $h_{2,\text{all}}$. Apesar do cálculo da heurística $h_{3,\text{all}}$ possuir um tempo de execução maior do que o da heurística $h_{2,\text{all}}$, ela tem a capacidade de reduzir o espaço de busca significativamente, acelerando a fase de execução da Busca A-Star. Com isso, o tempo total de execução tende a ser reduzido.

4.1 Visão Geral da Abordagem Proposta

A ferramenta PA-Star é executada segundo um fluxo de processamento composto por três fases, conforme ilustrado na Figura 4.1. Nesta figura, a fase em cinza foi a fase alterada pela nossa abordagem.

Na Fase 1, o PA-Star recebe como entrada o conjunto de sequências a ser alinhado. De posse desse conjunto, executa-se a função heurística, que vai preparar os dados para a Fase 2. No PA-Star, a função heurística executada é a $h_{2,\text{all}}$. Na presente Dissertação,

adicionamos a heurística $h_{3,\text{all}}$ (Seção 4.2). Além disso, o cálculo foi paralelizado tanto para a $h_{2,\text{all}}$ como para a $h_{3,\text{all}}$.

As Fases 2 e 3 são as fases originais da ferramenta *PA-Star*. Na Fase 2, o algoritmo de busca A^* é executado empregando os limites inferiores calculados na fase anterior para a poda do espaço de busca. A Fase 3 realiza o procedimento de *traceback*, responsável pela reconstrução do alinhamento ótimo a partir dos nós explorados durante a busca. Por fim, os resultados obtidos são consolidados em relatórios que permitem a análise quantitativa de métricas de desempenho, incluindo tempo de execução, consumo de memória, número de nós expandidos e escore final do alinhamento.

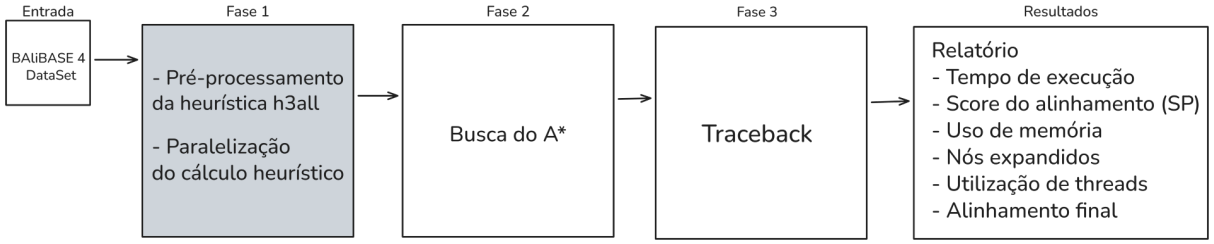

Figura 4.1: Fluxo do PA-Star e alterações feitas na Fase 1

4.2 Heurística Baseada em Trios (h3,all)

A motivação central dessa proposta é reduzir o espaço de busca, se comparada às heurísticas tradicionais h_1 e h_2 [12], fornecendo um limite inferior mais rigoroso para o algoritmo A^*, conforme fundamentado teoricamente por Colbourn e Kumar [47].

Na versão original da ferramenta PA-Star, a estimativa do custo é fornecida pela heurística $h_{2,all}$. Essa heurística é definida como a soma dos escores dos alinhamentos ótimos entre todos os pares de sequências do conjunto (Equação 2.4). Apesar do $h_{2,all}$ atender à propriedade de admissibilidade, condição necessária para assegurar a otimalidade da solução global, sua formulação tende a gerar limites inferiores conservadores. Como resultado, o algoritmo é frequentemente forçado a explorar um grande número de nós no espaço de busca [10].

Para reduzir ainda mais o espaço de busca, a presente Dissertação incorporou no PA-Star a heurística $h_{3,all}$, que calcula o limite inferior a partir do somatório dos alinhamentos ótimos de todos os trios de sequências. A hipótese principal, que é fortemente apoiada pela literatura [47], é que $h_{3,all}$ oferece uma estimativa de custo consideravelmente mais restritiva (*tighter lower bound*) se comparada a heurística $h_{2,all}$, o que leva a uma poda mais agressiva e eficaz do espaço de busca. Além disso, foi mostrado que a $h_{3,all}$ é uma heurística admissível e, portanto, a obtenção do resultado ótimo é garantida [12].

4.2.1 Definição Formal e Admissibilidade

A heurística $h_{3,all}$ baseia-se no cálculo exato de alinhamentos para todos os subconjuntos de três sequências derivados do conjunto de entrada (Equação 2.5). Enquanto a heurística baseada em pares considera apenas pares de sequências (Equação 2.4), a abordagem de Colbourn e Kumar [47] mostra que o alinhamento exato de três sequências é capaz de capturar dependências locais ignoradas nas projeções par-a-par, resultando em uma estimativa de custo significativamente mais próxima do custo real do alinhamento múltiplo.

Para assegurar a otimalidade da solução obtida pelo algoritmo A^*, a função heurística deve satisfazer a propriedade de admissibilidade. A validade matemática da heurística $h_{3,all}$ é garantida pelo tratamento adequado da redundância combinatória intrínseco à soma dos trios [47]. Dado um conjunto de N sequências, cada par específico participa de exatamente $(N-2)$ trios distintos. Assim, o valor heurístico associado a um nó n é definido como o somatório dos escores ótimos de todos os trios, normalizado por esse fator de redundância a Equação 2.5 passa a ser conforme a Equação 4.1 [12, 47]:

$$h_{3,all}(n) = \frac{\sum_{1 \leq i < j < k \leq N} \mathrm{MSA}(S_i, S_j, S_k)}{N-2} \tag{4.1}$$

onde $\mathrm{MSA}(S_i, S_j, S_k)$ representa o escore do alinhamento ótimo entre as sequências S_i, S_j e S_k. Essa normalização assegura que o custo de cada par de resíduos não seja contabilizado múltiplas vezes, preservando o limite inferior admissível ($h(n) \leq h^*(n)$) necessário para a busca exata.

4.2.2 Estruturas de Dados e Otimização de Memória

A implementação da heurística $h_{3,all}$ impõe desafios de memória significativamente superiores à implementação original da heurística $h_{2,all}$, uma vez que o armazenamento das matrizes de programação dinâmica cresce de forma cúbica em relação ao comprimento das sequências, isto é, $O(L^3)$ [12].

A primeira decisão tomada foi o uso da técnica de linearização de memória. Em vez de alocar matrizes tridimensionais ou estruturas baseadas em múltiplos níveis de ponteiros, os dados associados aos alinhamentos de trios são armazenados em vetores unidimensionais contíguos, conforme o proposto por Sundfeld [10] para os alinhamentos de pares. O mapeamento das coordenadas tridimensionais (i, j, k) para uma posição no vetor linear é realizado por meio da função de indexação apresentada na Equação 4.2, onde L_2 e L_3 representam os comprimentos das três sequências que compõem o trio.

$$\mathrm{Index}(i, j, k) = i \times (L_2 + 1) \times (L_3 + 1) + j \times (L_3 + 1) + k \tag{4.2}$$

Essa estratégia melhora a localidade espacial dos dados, reduzindo potencialmente falhas de *cache* (*cache misses*) durante as consultas massivas realizadas pelo algoritmo A^* na Fase 2 do processamento [9, 10].

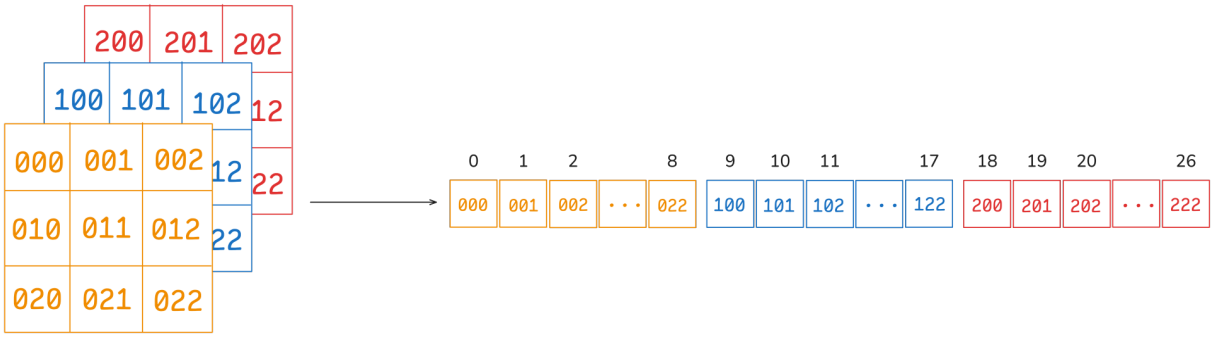

Figura 4.2: Linearização da Matriz 3D para 1D 0-based

4.2.3 Algoritmo de Programação Dinâmica Tridimensional

O cálculo dos custos para cada trio segue a generalização tridimensional do algoritmo de Needleman–Wunsch (Algoritmo 2.1) [13], amplamente descrita na literatura de alinhamento exato. Para cada célula da matriz cúbica, o custo ótimo é obtido a partir da minimização entre sete transições possíveis (Figura 4.3), correspondentes a:

- **Inserções de lacunas (*gaps*)**: movimentos que representam a inserção de lacunas em uma ou duas sequências do trio, associadas ao custo γ;
- **Alinhamento de resíduos**: a transição vinda da diagonal $(i-1, j-1, k-1)$, que acumula os custos de substituição de todos os pares induzidos (S_1, S_2), (S_1, S_3) e (S_2, S_3).

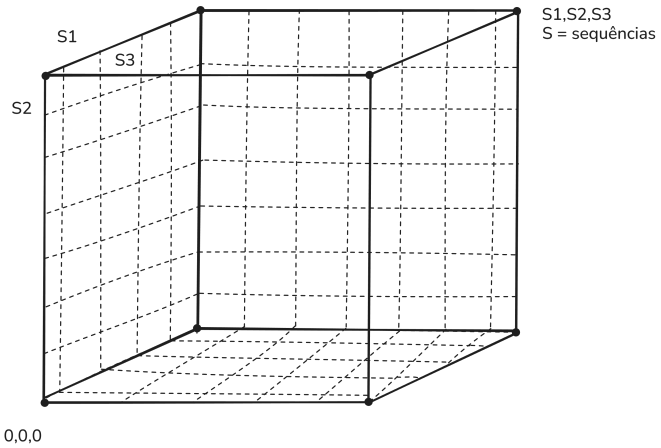

Figura 4.3: Generalização Tridimensional do Algoritmo

Essa etapa, computacionalmente intensiva, corresponde à Fase 1 do fluxo proposto e é executada de forma paralela para minimizar o custo de pré-processamento antes do início da busca heurística.

Tabela 4.1: Comparação teórica entre a heurística $h_{2,all}$ e $h_{3,all}$ [47].

Característica	$h_{2,all}$ **(Original)**	$h_{3,all}$ **(Proposta)**
Base de cálculo	Pares de sequências	Trios de sequências
Complexidade de espaço	$O(N^2 \cdot L^2)$	$O(N^3 \cdot L^3)$
Tempo (pré-processamento)	$O(N^2 \cdot L^2)$	$O(N^3 \cdot L^3)$
Tempo (consulta na busca)	$O(N^2)$	$O(N^3)$
Informatividade (poda)	Moderada	Alta
Admissibilidade	Garantida	Garantida

A Tabela 4.1 sintetiza as principais diferenças entre a heurística $h_{2,all}$, tradicionalmente empregada no estado da arte, e a abordagem proposta, que utiliza a heurística $h_{3,all}$. Embora a heurística $h_{3,all}$ imponha um crescimento cúbico nos requisitos de memória e tempo de pré-processamento, o custo é reduzido pela estratégia de paralelização apresentada na Seção 4.3, assim oferecendo um ganho significativo de desempenho. Esse *trade-off* é justificado pela redução significativa no número de nós expandidos durante a fase de busca A^*, conforme demonstrado teoricamente por McNaughton et al. [12].

4.3 Estratégia da Paralelização

Visto que o custo de processamento cresce de maneira combinatória, pois o número de subalinhamentos requeridos passa de $\binom{N}{2}$ para $\binom{N}{3}$, a viabilidade prática da aplicação da heurística $h_{3,all}$ depende diretamente da redução do custo extra introduzido [47].

Ao contrário da abordagem sequencial do PA-Star original na fase de pré-processamento, a estratégia proposta neste estudo aproveita a característica *embarrassingly parallel* do problema, em que o cálculo de cada trio (S_i, S_j, S_k) é uma tarefa independente. Dessa forma, foi utilizado o modelo de paralelismo por tarefas (*task parallelism*) para distribuir o cálculo dos trios entre os núcleos de processamento disponíveis, buscando compensar o aumento do tempo de pré-processamento causado pela heurística $h_{3,all}$ e visando uma redução no tempo total de busca.

4.3.1 Visão Geral da Estratégia de Paralelização

Esta seção apresenta uma visão geral da solução proposta para a paralelização do cálculo da heurística baseada em trios utilizada no algoritmo PA-Star. A Figura 4.4 mostra a

estrutura da solução, destacando as etapas principais e mostrando como o paralelismo é utilizado.

A solução fundamenta-se na observação de que o cálculo da heurística $h_{3,\text{all}}$ pode ser decomposto em um conjunto de subproblemas completamente independentes, cada um correspondente ao alinhamento múltiplo exato de um trio distinto de sequências. Como visto na Seção 4.2, o número de subalinhamentos requeridos passa de $\binom{N}{2}$ para $\binom{N}{3}$, o que implica um custo computacional elevado à medida que N cresce. Entretanto, como cada alinhamento de trio é independente dos demais, essa decomposição é particularmente adequada para execução paralela.

A etapa de pré-processamento possui dois estágios. O primeiro estágio consiste na criação de um *pool* de tarefas, no qual cada tarefa representa a computação do alinhamento exato de um trio específico de sequências. Formalmente, cada tarefa é definida como

$$\text{mAlign}(S_i, S_j, S_k), \quad \text{com } i < j < k,$$

em que S_i, S_j e S_k pertencem ao conjunto original de sequências. Esse *pool* é formado por uma lista de tarefas independentes que, em conjunto, abrangem todos os trios necessários para o cálculo da heurística.

No segundo estágio, as tarefas são distribuídas entre um conjunto fixo de *threads*. Cada *thread* recebe um subconjunto de tarefas, adotando-se uma estratégia de particionamento estático. Cada *thread* é responsável por acionar um *worker*, que executa efetivamente os alinhamentos dos trios. O *worker* realiza o cálculo por meio de programação dinâmica, produzindo o custo ótimo associado a cada trio processado. Os resultados obtidos são armazenados em uma estrutura de dados compartilhada, indexada conforme a enumeração das tarefas. Nesse estágio, devido ao particionamento, não há overhead de sincronização.

Por fim, os custos individuais calculados pelos *workers* são agregados em uma estrutura global chamada de `mAligns`, que também acessada de maneira particionada, contém os resultados referentes a todos os trios avaliados.

A estrutura `mAligns` é utilizada então na segunda etapa da Fase 1 do PA-Star chamada de Cálculo Heurístico. Nessa etapa, o caminho ótimo será obtido, e o valor final da heurística $h_{3,\text{all}}$ será calculado. Esse valor será usado pelo algoritmo PA-Star para guiar a exploração do espaço de busca, na Fase 2.

Essa solução paralela apresenta vantagens significativas, pois maximiza o uso dos recursos computacionais disponíveis, reduzindo o tempo total de cálculo da heurística, fator importante para a manter o algoritmo viável em conjuntos grandes de sequências.

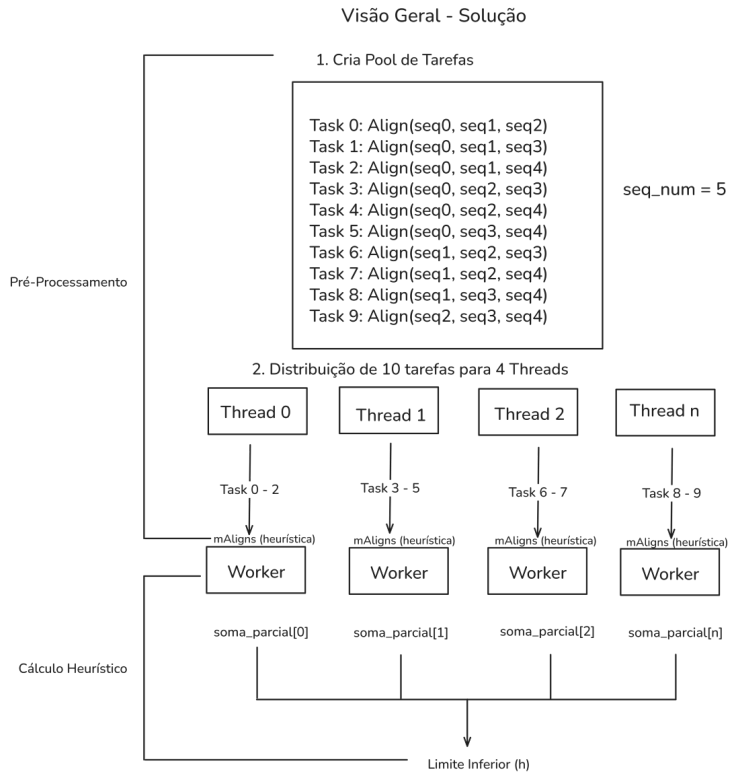

Figura 4.4: Visão Geral da Solução de Paralelização

4.3.2 Decomposição e Distribuição de Carga

Uma parte importante desse modelo de paralelização é a distribuição de carga de trabalho entre as *threads* que, nesse trabalho, é feita de maneira estática e particionada. Nesse modelo, o número total de tarefas, que corresponde aos alinhamentos de trios, é dado pela fórmula combinatória $\binom{N}{3}$, permitindo uma divisão uniforme do espaço de trabalho entre as *threads* disponíveis. O particionamento estático é vantajoso em sistemas paralelos, pois elimina a comunicação entre as threads nessa etapa e permitindo um aumento da eficiência computacional [48, 49, 50, 51].

O particionamento estático foi escolhido devido ao perfil uniforme de cada tarefa. Em cada coluna a soma de trios envolve operações de programação dinâmica cujo custo depende do comprimento das sequências, e esse comprimento é o mesmo para todas as *threads*.

Dessa forma optou-se por usar o particionamento estático, que reduz o *overhead* de escalonamento e evita a necessidade de mecanismos adicionais de balanceamento dinâmico de carga.

Caso houvesse grande variação no custo das tarefas, estratégias de escalonamento dinâmico poderiam ser consideradas. Entretanto, para o perfil de carga observado nesta Dissertação de Mestrado, a abordagem estática mostrou-se adequada e eficiente.

De acordo com a estratégia de particionamento estático, utilizada nesta Dissertação, o total de tarefas T_{total} é calculado conforme a Equação 4.3 [52]:

$$T_{\text{total}} = \binom{N}{3} = \frac{N(N-1)(N-2)}{6} \tag{4.3}$$

Embora o número de tarefas seja determinado pelo número de sequências (N), o custo computacional de cada tarefa depende do comprimento das sequências envolvidas. O alinhamento múltiplo de três sequências por programação dinâmica possui complexidade $O(L^3)$, onde L representa o comprimento médio das sequências. Dessa forma, o custo total da fase de pré-processamento é dado pela Equação 4.4 [52]:

$$O\left(\binom{N}{3} \cdot L^3\right) \tag{4.4}$$

Inicialmente, define-se o número de tarefas atribuídas a cada *thread*. Considerando o conjunto total de tarefas, o número máximo de tarefas por *thread*, expresso por T, é dado pela Equação 4.5 [52]:

$$T = \left\lceil \frac{T_{\text{total}}}{N_{\text{threads}}} \right\rceil \tag{4.5}$$

A partir desse valor, as tarefas são distribuídas entre as *threads* de forma contígua. Para a *thread* i, os índices de início (`start_idx`) e fim (`end_idx`) do bloco de tarefas são definidos nas Equações 4.6 e 4.7 [52]:

$$\text{start\_idx}_i = (i-1) \times T + 1 \tag{4.6}$$

$$\text{end\_idx}_i = \begin{cases} i \times T, & \text{se } i < N_{\text{threads}} \\ T_{\text{total}}, & \text{se } i = N_{\text{threads}} \end{cases} \tag{4.7}$$

Por exemplo, considerando $N = 6$ e $N_{\text{threads}} = 4$, o número total de alinhamentos de trios será 20, usando como base a Equação 4.3: $\binom{6}{3} = 20$. Nesse caso, as 20 tarefas são distribuídas entre as quatro *threads*. Utilizando a estratégia de particionamento estático descrita anteriormente, cada *thread* recebe $\lceil 20/4 \rceil = 5$ tarefas. Assim, cada *thread* processa 5 alinhamentos, garantindo uma divisão equilibrada do trabalho.

Esse particionamento assegura que cada *thread* processe suas tarefas de forma independente, sem necessidade de comunicação entre elas. Dessa forma, elimina-se a sobrecarga de sincronização explícita, característica comum em outros modelos de paralelismo dinâmico [53, 54, 55, 56].

Além disso, a abordagem estática favorece a previsibilidade de desempenho. Como a carga de trabalho é dividida de forma igual e sem comunicação entre threads, o ganho de desempenho (*speedup*) tende a se aproximar do comportamento ideal, com o tempo de execução diminuindo proporcionalmente ao número de *threads*, sempre que o número de tarefas for grande o suficiente [57, 58]. Essa característica torna o particionamento estático uma boa escolha para problemas onde o número total de tarefas é conhecido.

Nas Seções 4.3 e 4.4, as Fases de Pré-Processamento e de Cálculo Heurístico são detalhadas.

4.4 Etapa de pré-processamento

A etapa de pré-processamento tem como objetivo a construção da estrutura `mAligns`, que armazena os custos ótimos dos alinhamentos múltiplos exatos de todos os trios possíveis de sequências. As atividades executadas pelas *threads* nessa fase são ilustradas na Figura 4.5.

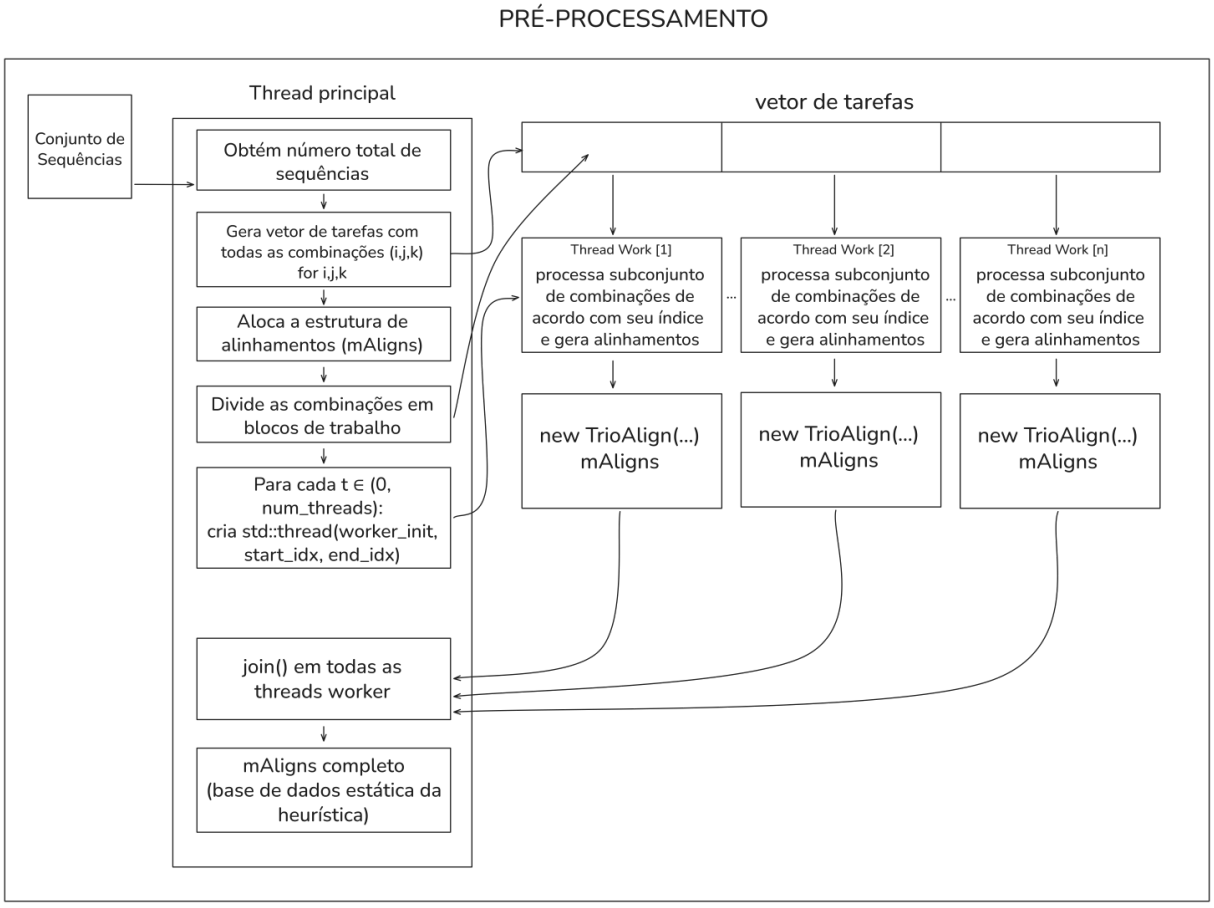

Figura 4.5: Etapa de Pré-Processamento

A execução é coordenada pela *thread* principal, que inicialmente obtém o número de sequências de entrada e, a partir desse valor, gera o vetor de tarefas contendo todas as combinações possíveis de índices (i, j, k), com $i < j < k$. Cada elemento desse vetor representa uma tarefa independente, correspondente ao cálculo do alinhamento exato de um trio específico de sequências.

Em seguida, a *thread* principal aloca a estrutura global `mAligns`, que funcionará como uma estrutura de dados estática da heurística. Essa estrutura é dimensionada de modo a armazenar, de forma indexada, o custo ótimo associado a cada trio de sequências.

As tarefas geradas são então particionadas em blocos de trabalho, definidos por intervalos de índices no vetor de tarefas. Para cada bloco, a *thread* principal cria uma *thread*, associando-a a um intervalo específico por meio dos parâmetros de início e fim, utilizando as Equações 4.6 e 4.7. Cada *thread* processa exclusivamente o seu subconjunto de tarefas, instanciando objetos responsáveis pelo cálculo do alinhamento e armazenando diretamente os resultados nas posições correspondentes do `mAligns`. Como não há sobreposição de escrita entre diferentes *threads*, não são necessários mecanismos de sincronização.

Ao término do processamento, a *thread* principal realiza a sincronização final por meio da operação de `join` em todas as *threads* . Após essa etapa, a estrutura `mAligns` encontra-se completamente preenchida e passa a ser considerada imutável, estando pronta para ser utilizada na etapa de cálculo heurístico.

4.5 Etapa de cálculo heurístico

4.5.1 Fluxo de Execução

A etapa de cálculo heurístico ocorre durante a execução do algoritmo A* e utiliza exclusivamente a base `mAligns` previamente construída. As operações executadas nessa etapa são apresentadas na Figura 4.6.

Nessa etapa, a *thread* responsável pelo algoritmo A* avalia inicialmente a soma dos tamanhos das sequências, que determina o tamanho da estrutura `mAligns`. Caso a soma seja suficientemente pequena, o cálculo da heurística é realizado de forma sequencial, por meio de um percurso linear sobre os elementos de `mAligns`. Essa decisão evita a sobrecarga associada à criação e sincronização de *threads* em situações nas quais o paralelismo não traria ganhos significativos de desempenho.

Por outro lado, quando a soma dos tamanhos das sequências ultrapassa um limiar pré-definido, o cálculo heurístico é paralelizado. Nesse caso, os alinhamentos armazenados em `mAligns` são particionados em blocos de trabalho distribuídos entre múltiplas *threads*.

Cada *thread* processa seu subconjunto de alinhamentos, calculando uma soma parcial das contribuições heurísticas correspondentes usando a Equação 4.1.

Após a conclusão das *threads*, a *thread* principal realiza a sincronização por meio da operação de `join` e agrega as somas parciais em um valor global. Esse valor agregado é então normalizado conforme a definição da heurística visto na Seção 4.2.1, resultando no valor final admissível $h(n)$, utilizado pelo PA-Star para guiar a exploração do espaço de busca.

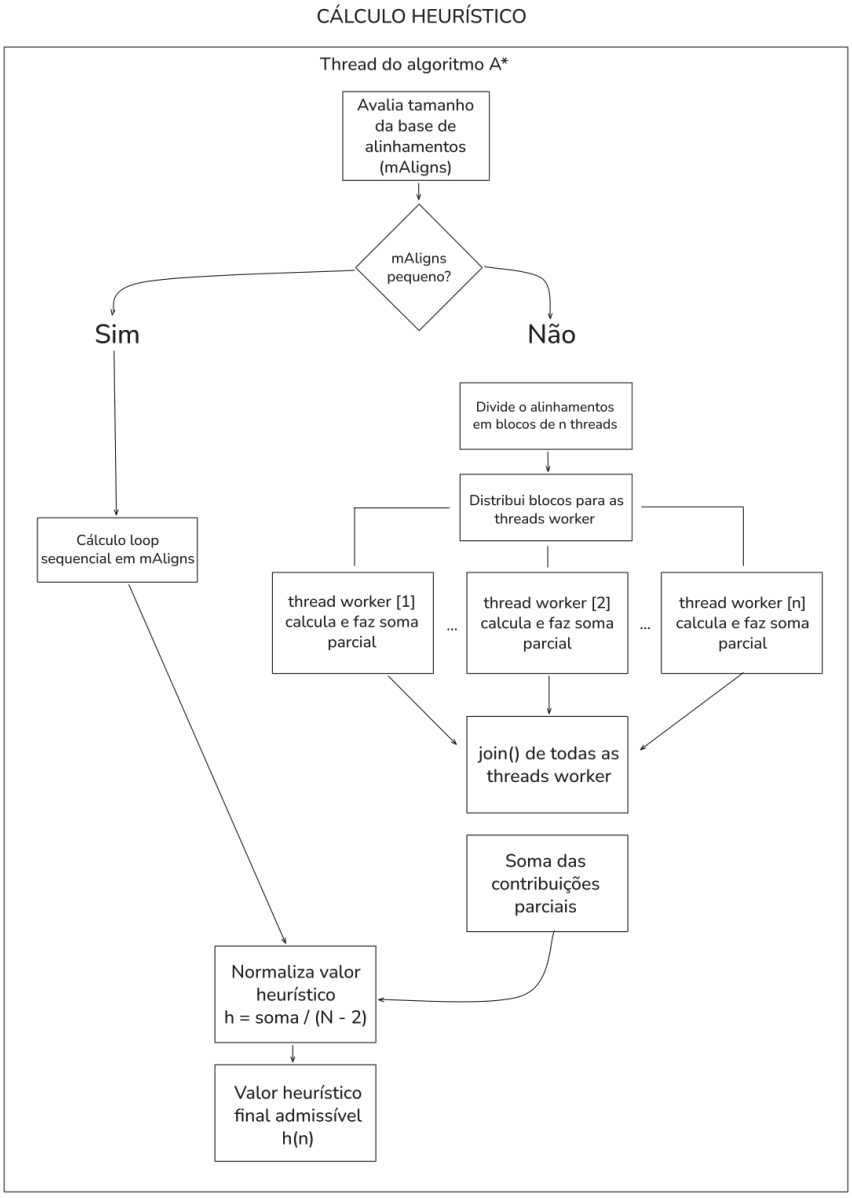

Figura 4.6: Etapa do Cálculo Heurístico

A relação entre os componentes pode ser definida como:

- **mAligns**: Vetor contendo todos os alinhamentos ótimos entre os trios de sequências e o custo mínimo de alinhamento múltiplo para cada trio. É calculado usando programação dinâmica 3D para todas as combinações (i, j, k) (Algoritmo 2.1).
- $h(n)$: A função heurística derivada do *mAligns*, calculada usando a Equação 4.1.
- **PA-Star**: Utiliza o valor de $h(n)$ como estimativa de custo admissível para guiar a busca.

De maneira geral, a solução proposta explora paralelismo tanto na fase de pré-processamento quanto na fase de cálculo heurístico, preservando a independência entre os alinhamentos de trios e garantindo a correção e a admissibilidade da heurística.

Antes de iniciar o processamento, o sistema realiza uma avaliação do número total de alinhamentos a serem calculados e, com base nesse *profiling*, escolhe o modo de execução mais adequado, conforme descrito a seguir:

- **Execução sequencial:** Quando o número de alinhamentos é inferior a um *threshold*, o cálculo é realizado exclusivamente pela *thread* principal, onde o *threshold* é definido de maneira empírica. Essa estratégia evita o custo de inicialização e sincronização do ambiente paralelo;
- **Execução paralela:** Para cargas de trabalho que excedem o limiar estabelecido, o *pool* de *threads* é ativado, permitindo a exploração do paralelismo disponível na arquitetura.

O fluxo de execução paralela segue o modelo *Fork–Join*, no qual as *threads* são disparadas simultaneamente para processar seus respectivos blocos de alinhamentos. Ao término da computação local, o método `join()` é utilizado para sincronizar a conclusão de todas as tarefas.

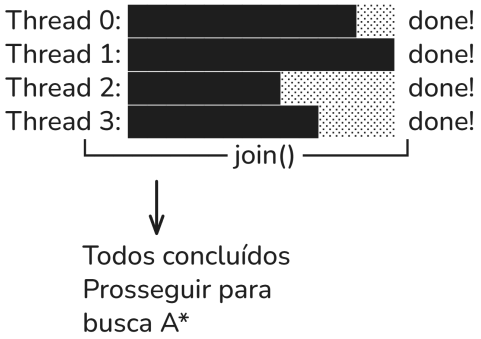

Figura 4.7: Modelo fork-join para a execução das threads

Para agregar o valor final da heurística, foi adotado um padrão de Redução Paralela (*Parallel Reduction*). Cada *thread* mantém uma variável local de soma parcial

(`partial_sum`) durante a iteração sobre seu subconjunto de trios. Após a barreira de sincronização, essas somas parciais são combinadas para compor o valor total da heurística.

Essa abordagem, determinada por um *profiling* da carga de trabalho, elimina a necessidade de operações atômicas a cada incremento, reduzindo a contenção entre *threads* e preservando a eficiência do uso de cache do processador [48, 49, 50].

4.5.2 Comparação entre o PA-Star Original e a Estratégia de Paralelização Proposta

Nesta seção, apresenta-se um comparativo entre a abordagem sequencial original, utilizada nas versões anteriores da ferramenta PA-Star para o cálculo de alinhamentos par-a-par, e a nova implementação paralela baseada em tarefas.

O Algoritmo 4.5.1 ilustra a abordagem sequencial [10]. Observa-se o uso de laços aninhados nos quais a instanciação e o cálculo de cada alinhamento são executados de maneira sequencial pela *thread* principal. As linhas 1 e 2 estabelecem a iteração sobre todas as combinações de pares de sequências. Na linha 3, cria-se a estrutura de identificação do par, enquanto a linha 4 realiza a operação computacionalmente intensiva de alinhamento e alocação de memória. Por fim, a linha 5 insere o resultado no vetor `mAligns`.

Algorithm 4.5.1 PA-Star (Original)

Require: Conjunto de sequências S, Número de sequências N

1: $mAligns \leftarrow \emptyset$
2: **for** $i \leftarrow 0$ **to** $N - 2$ **do**
3: **for** $j \leftarrow i + 1$ **to** $N - 1$ **do**
4: $p \leftarrow$ `Pair`(i, j)
5: $a \leftarrow$ `new PairAlign`$(p, S[i], S[j])$
6: $mAligns$.`push_back`(a)
7: **end for**
8: **end for**

Em contraste, o Algoritmo 4.5.2 apresenta a estratégia paralela proposta. Nessa abordagem, a lógica de processamento é desacoplada da definição das tarefas. Inicialmente, o vetor de tarefas é inicializado como vazio na linha 1. O bloco compreendido pelas linhas 2 a 6 é responsável por preencher o vetor com todas as combinações de alinhamentos necessárias (*tasks*), sem, contudo, executar o processamento naquele momento.

Posteriormente, as linhas 7 a 9 realizam a distribuição de carga e o disparo (*fork*) das *threads*. A linha 8 delega a cada *thread* a execução da função `Worker`, passando um intervalo específico de índices (*start_idx*, *end_idx*), conforme a estratégia estática

descrita na Seção 4.3.2. As linhas 10 a 12 implementam a barreira de sincronização (*join*), assegurando que o fluxo principal aguarde a conclusão de todas as tarefas.

O procedimento `Worker`, detalhado entre as linhas 13 e 18, define a rotina executada concorrentemente. A linha 14 estabelece o laço de repetição restrito ao intervalo de alinhamentos atribuído àquela *thread* específica. Na linha 15, os dados do trio de identificadores (i, j, k, p) são recuperados do vetor global de tarefas. A linha 16 executa o alinhamento dos trios, que corresponde à operação de maior custo computacional. Por fim, a linha 17 armazena o resultado no vetor de alinhamentos (`mAligns`).

Algorithm 4.5.2 Abordagem Paralela com Pool de Tarefas (Proposta)

Require: Conjunto de sequências S, Número de sequências N, Threads $N_{threads}$

1: $tasks \leftarrow \emptyset$
▷ Geração das tarefas para Trios (i, j, k)
2: **for** $i \leftarrow 0$ **to** $N - 3$ **do**
3: **for** $j \leftarrow i + 1$ **to** $N - 2$ **do**
4: **for** $k \leftarrow j + 1$ **to** $N - 1$ **do**
5: $tasks$.`push_back`(i, j, k)
6: **end for**
7: **end for**
8: **end for**
▷ Distribuição e execução das threads
9: **for** $t \leftarrow 0$ **to** $N_{threads} - 1$ **do**
10: $start\_idx, end\_idx \leftarrow$ `CalculateBounds`$(t, |tasks|)$
11: $threads$.`emplace_back`(WORKER, $start\_idx, end\_idx$)
12: **end for**
13: **for all** $thread \in threads$ **do**
14: $thread$.`join`()
15: **end for**
▷ Procedimento executado por cada *thread*
16: **function** WORKER($start\_idx, end\_idx$)
17: **for** $t \leftarrow start\_idx$ **to** end_idx **do**
18: $(i, j, k) \leftarrow tasks[t]$
19: $a \leftarrow$ `new Align`$(S[i], S[j], S[k])$
20: $mAligns[t] \leftarrow a$ ▷ Escrita direta na posição pré-alocada
21: **end for**
22: **end function**

Essa reestruturação permite a exploração simultânea de múltiplos núcleos do processador. Além disso, a separação entre a definição de tarefas (linhas 1–6) e a lógica de execução (linhas 13–18) favorece a manutenção do código.

4.6 Detalhes de Implementação

A implementação proposta nesta Dissertação de Mestrado foi construída diretamente sobre a base da ferramenta PA-Star original. Em vez de introduzir novas interfaces de paralelização, como OpenMP, optou-se por manter o modelo já presente no código, que utiliza os mecanismos de concorrência da biblioteca padrão do `C++` atual, particularmente `std::thread`.

Essa decisão foi motivada pelo fato de que o código original já possuía uma estrutura modular bem definida projetada para o `std::thread` e seguiu-se esse padrão no cálculo da heurística.

A seguir serão explicadas as etapas da implementação da solução.

4.6.1 Geração das tarefas de alinhamento

A fase inicial da heurística consiste em gerar todos os alinhamentos possíveis entre trios de sequências. Para um conjunto de n sequências, são criadas $\binom{n}{3}$ combinações distintas. Cada combinação representa uma tarefa independente de alinhamento.

O código a seguir ilustra a geração dessas tarefas:

Listing 4.1: Geração das combinações de trios de sequências na classe TrioAlign.

```
for (int i = 0; i < seq_num - 2; i++)
{
    for (int j = i + 1; j < seq_num - 1; j++)
    {
        for (int k = j + 1; k < seq_num; k++)
        {
            AlignTask task;
            task.i = i;
            task.j = j;
            task.k = k;
            task.t = Trio(i, j, k);
            tasks.push_back(task);
        }
    }
}
```

Cada tarefa corresponde ao cálculo de um alinhamento entre três sequências, realizado pela classe `TrioAlign`. Como essas tarefas são independentes entre si, elas podem ser executadas em paralelo sem necessidade de sincronização complexa.

4.6.2 Paralelização com `std::thread`

A paralelização é feita por meio da criação explícita de *threads* utilizando `std::thread`. O número de *threads* é determinado automaticamente com base na capacidade do hardware por meio da função `std::thread::hardware_concurrency()`. No caso improvável de não haver detecção, o número de *threads* é definido como 4.

Listing 4.2: Detecção automática do número de threads disponíveis.

```
m_num_threads = std::thread::hardware_concurrency();
if (m_num_threads == 0)
    m_num_threads = 4; // fallback
```

Cada *thread* executa uma função de trabalho responsável por processar um subconjunto das tarefas de alinhamento.

4.6.3 Particionamento estático das tarefas

Conforme discutido da Seção 4.3.2, a distribuição das tarefas entre as *threads* é realizada por meio de particionamento estático, conforme ilutrado na Listagem 4.3.

Listing 4.3: Distribuição estática das tarefas entre as threads.

```
size_t tasks_per_thread =
    (total_tasks + m_num_threads - 1) / m_num_threads;
for (int t = 0; t < m_num_threads; ++t)
{
    size_t start_idx = t * tasks_per_thread;
    size_t end_idx = std::min(start_idx + tasks_per_thread,
                              total_tasks);
    if (start_idx < total_tasks)
    {
        threads.emplace_back(worker, start_idx, end_idx);
    }
}
```

Cada *thread* executa a função `worker`, responsável por calcular os alinhamentos correspondentes ao intervalo atribuído. A implementação da função *worker* está ilustrado na Listagem 4.4.

Listing 4.4: Função worker responsável pelo cálculo dos alinhamentos de trios.

```
auto worker = [&](size_t start_idx, size_t end_idx)
{
    for (size_t idx = start_idx; idx < end_idx; ++idx)
    {
        const AlignTask &task = tasks[idx];
```

```
6           TrioAlign *a = new TrioAlign(
7               task.t,
8               seq->get_seq(task.i),
9               seq->get_seq(task.j),
10              seq->get_seq(task.k)
11          );
12          mAligns[idx] = a;
13      }
14  };
```

Esse modelo elimina a necessidade de sincronização entre as *threads*, pois cada uma escreve em posições distintas do vetor de resultados.

4.6.4 Paralelização do cálculo da heurística

Além da fase de geração dos alinhamentos, o cálculo da heurística durante a execução do algoritmo A* foi paralelizado quando o número de alinhamentos é elevado.

Para conjuntos pequenos de alinhamentos, a execução sequencial (Listagem 4.5) é mantida, pois o custo de criação de *threads* poderia superar os benefícios da paralelização.

Listing 4.5: Execução sequencial para pequenos conjuntos de alinhamentos.

```
1  if (num_aligns < 50)
2  {
3      int h = 0;
4      for (auto it = mAligns.begin(); it != mAligns.end(); ++it)
5      {
6          int x = std::get<0>((*it)->getTrio());
7          int y = std::get<1>((*it)->getTrio());
8          int z = std::get<2>((*it)->getTrio());
9          h += (*it)->getScore(c[x], c[y], c[z]);
10     }
11     return h / N-2;
12 }
```

Quando o número de alinhamentos é maior, o cálculo é distribuído entre múltiplas *threads* (Listagem 4.6). Cada *thread* calcula o seu subconjunto de colunas de maneira independente (linhas 7 – 15) em seguida as diversas somas parciais são agregas no resultado final (linha 16).

Listing 4.6: Cálculo paralelo da heurística com redução das somas parciais.

```
1  std::vector<int> partial_sums(m_num_threads, 0);
2  auto worker = [&](int thread_id,
3                    size_t start_idx,
4                    size_t end_idx)
```

```
{
    int local_sum = 0;
    for (size_t idx = start_idx; idx < end_idx; ++idx)
    {
        TrioAlign *align = mAligns[idx];
        int x = std::get<0>(align->getTrio());
        int y = std::get<1>(align->getTrio());
        int z = std::get<2>(align->getTrio());
        local_sum += align->getScore(
            c[x], c[y], c[z]);
    }
    partial_sums[thread_id] = local_sum;
};
```

4.7 Disponibilidade do Código

O código-fonte do algoritmo PA-Star com a heurística $h_{3,all}$ está disponível publicamente no repositório no GitHub[1], sob licença *MIT*. O repositório reúne a implementação utilizada nos experimentos desta Dissertação de Mestrado, bem como instruções para compilação e execução, permitindo que outros pesquisadores reproduzam os resultados apresentados.

O projeto foi desenvolvido em C++ e organizado de forma modular, facilitando a compreensão, manutenção e eventual extensão do código. Ao longo do desenvolvimento, diferentes versões foram disponibilizadas, contemplando tanto a implementação original quanto as soluções relacionadas à paralelização e à incorporação da heurística $h_{3,all}$.

A estrutura do repositório e o sistema de compilação foram planejados com foco em reprodutibilidade e clareza, oferecendo suporte à execução dos experimentos descritos neste trabalho e possibilitando futuras comparações ou aprimoramentos por parte da comunidade acadêmica.

[1] https://github.com/vncsmnl/astar_msa

Capítulo 5

Resultados Experimentais

Este capítulo apresenta a avaliação experimental das otimizações propostas para a ferramenta PA-Star. O foco da análise reside na comparação entre a eficácia da nova função heurística $h_{3,all}$ e a abordagem tradicional $h_{2,all}$, bem como no impacto da paralelização do cálculo heurístico no desempenho global do sistema.

5.1 Sequências Utilizadas

Para a realização dos testes, foram selecionados conjuntos de dados provenientes do *benchmark* BAliBASE 4[46], especificamente da referência 1 (Ref1), que reúne alinhamentos de sequências globalmente homólogas. O BAliBASE é amplamente reconhecido na literatura por fornecer alinhamentos de alta qualidade, construídos com base em informações estruturais tridimensionais de proteínas, sendo frequentemente utilizado como padrão-ouro para a avaliação de algoritmos de MSA [8].

Os conjuntos escolhidos pertencem à categoria de sequências longas (*long*), caracterizadas por comprimentos elevados, variando de aproximadamente 340 aminoácidos até mais de 800 aminoácidos. Além disso, os conjuntos diferem quanto ao nível de identidade entre as sequências, abrangendo cenários de alta, média e baixa similaridade. Em particular, foram selecionados conjuntos com alta identidade (*high_id*), identidade intermediária (*medium_id*) e baixa identidade (*low_id*), permitindo avaliar o comportamento do algoritmo sob diferentes graus de dificuldade de alinhamento.

A Tabela 5.1 apresenta os quatro conjuntos de dados utilizados: `1gpb.fasta`, `2myr.fasta`, `1sesA.fasta` e `arp.fasta`. Essa variedade de características permite uma análise do desempenho ao longo do tempo em relação a instâncias de diversas complexidades [4, 12].

Tabela 5.1: Conjuntos de dados do BAliBASE 4 utilizados.

Conjunto	Número de Sequências	Comprimento Mínimo	Comprimento Máximo	Identidade
1gpb.fasta	5	796	828	Alta (*high_id*)
2myr.fasta	4	340	474	Baixa (*low_id*)
1sesA.fasta	5	417	442	Média (*medium_id*)
arp.fasta	5	380	418	Média (*medium_id*)

Adicionalmente, foi selecionada uma instância específica da família proteica PF03426, obtido da base PFAM[1], para a análise qualitativa da exploração do espaço de busca. Este conjunto, cujas características estão mostradas na Tabela 5.2, foi empregado para a geração de visualizações tridimensionais, permitindo a verificação empírica da capacidade de poda da heurística baseada em trios [12, 47].

Tabela 5.2: Conjunto dos dados utilizados para a visualização do espaço de busca.

Conjunto	Número de Sequências	Comprimento Mínimo	Comprimento Máximo
PF03426.fasta	3	159	163

A escolha destas sequências visa garantir que a avaliação do ganho de desempenho da heurística $h_{3,all}$ sobre a $h_{2,all}$ seja robusta, considerando tanto o tempo de pré-processamento quanto a redução efetiva do número de nós expandidos durante a fase de busca exata [12].

5.2 Ambiente de Teste

A avaliação experimental foi conduzida em dois ambientes distintos, visando observar o comportamento das otimizações tanto em uma infraestrutura local baseada na arquitetura x86, quanto em um ambiente de computação em nuvem com arquitetura ARM.

Ambiente Local

O ambiente local foi configurado com uma estação de trabalho equipada com um processador Intel Xeon E5-2680 v4 (microarquitetura Broadwell), operando a uma frequência de 3,30 GHz. Este processador possui 14 núcleos físicos e 28 núcleos lógicos (*threads*). O sistema conta com 32 GB de memória RAM DDR4. O sistema operacional utilizado foi o Fedora 43, e a ferramenta foi compilada com o GCC 11.3 e *flags* `-O3`. Devido às restrições de memória RAM da máquina ao crescimento cúbico da heurística $h_{3,all}$ [47], os testes neste ambiente foram restritos ao conjunto `2myr`, com cada alinhamento executado 10 vezes utilizando as 28 *threads* disponíveis para garantir estabilidade estatística. No ambiente local, foi observada uma variação mínima do tempo de execução, com desvio padrão baixo.

[1] https://www.ebi.ac.uk/interpro/entry/pfam/PF03426/

Ambiente em Nuvem (AWS)

Para testar a carga de trabalho e avaliar o desempenho na arquitetura em ARM, utilizou-se uma instância `c8g.16xlarge` na Amazon Web Services (AWS). Esta instância é equipada com o processador AWS Graviton4, dispondo de 64 vCPUs e 128 GB de memória RAM. O ambiente de software foi composto pelo Amazon Linux 2023, com o compilador GCC 11.3 e *flags* `-O3`.

Neste ambiente, dado o custo da instância `c8g.16xlarge` (USD 2.55/hora em novembro 2025), realizou-se uma execução para cada configuração de *threads* (8, 16, 32 e 64) para cada um dos conjuntos de dados (1gpb, 2myr, 1sesA e arp). Esta abordagem permitiu observar o *speedup* e o comportamento do consumo de memória em uma arquitetura ARM recente.

5.3 Ganho do h3,all sobre o h2,all

Nesta seção será mostrado o impacto da substituição da heurística de pares ($h_{2,all}$) pela heurística de trios ($h_{3,all}$) paralelizada, levando em conta o *trade-off* entre o custo de pré-processamento e a redução do esforço de busca na ferramenta PA-Star.

5.3.1 Execuções no Ambiente Local

Os testes realizados localmente com o conjunto `2myr` destacam o ganho de desempenho entre as versões do algoritmo. Cada versão foi executada 10 vezes, e a Figura 5.1 apresenta os resultados dessas execuções. Para facilitar a análise, as médias foram separadas pelas fases do algoritmo, conforme ilustrado na Figura 4.1.

Como mostra a Figura 5.2, a Fase 1 (pré-processamento e cálculo da heurística) da versão $h_{3,all}$ apresenta aumento significativo no tempo de execução, passando de 8,3 milissegundos para cerca de 2,84 segundos. Entretanto, esse *overhead* é compensado na Fase 2 (Busca A^*), resultando em melhor desempenho geral.

Fica claro que na Fase 2 (Busca A^*), o algoritmo usando a heurística $h_{3,all}$ apresenta um tempo de execução bem inferior, superando o aumento do tempo de execução da Fase 1. A redução média de 18,85% no tempo da Busca A^*, aliada a um speedup de 1,23x, mostra que o limite inferior (*lower bound*) obtido através da heurística $h_{3,all}$ reduz significativamente o espaço de busca, se comparada com a $h_{2,all}$. O uso do $h_{3,all}$ resultou em uma poda muito mais rigorosa do espaço, resultando não apenas em maior rapidez, mas em também um uso de memória mais eficaz, como mostrado pela economia média de 1,09 GB de RAM. Dessa forma, o custo de aproximadamente 2,8 segundos da Fase 1 torna-se pequeno em comparação com o ganho de mais de 220 segundos na execução

Total (2myr.fasta)							
PA-Star (h2,all) | Tempo (s)	RAM (GB)	PA-Star (h3,all) | Tempo (s)	RAM (GB)	Ganho RAM	Ganho de Tempo (s)	Speedup	%
1168,343	17,84	960,725	16,5	1,4	207,618	1,22	17,77%
1176,136	18,22	963,71	17,1	1,1	212,426	1,22	18,06%
1169,496	18,41	958,412	17,6	0,9	211,084	1,22	18,05%
1173,035	17,75	942,469	16,3	1,4	230,566	1,24	19,66%
1171,663	18,39	942,339	17,3	1,1	229,324	1,24	19,57%
1168,951	17,68	944,878	16,9	0,8	224,073	1,24	19,17%
1175,32	17,95	958,494	17,8	0,1	216,826	1,23	18,45%
1172,498	18,68	950,757	16,2	2,5	221,741	1,23	18,91%
1174,135	17,53	955,304	16,9	0,6	218,831	1,23	18,64%
1170,014	18,56	962,204	17,4	1,2	207,81	1,22	17,76%

Figura 5.1: Resultado de 10 execuções do h2,all e h3,all em ambiente local com o conjunto 2myr.

Fase 1						
Versão	Média Tempo (s)	Ganho Tempo (s)	RAM (GB)	Ganho RAM (GB)	Speedup Médio	%
PA-Star (h2_all)	0,0083		0,361			
PA-Star (h3_all)	2,8395	-2,8312	0,893	-0,532	0,003	-34111%

Fase 2						
Versão	Média Tempo (s)	Ganho Tempo (s)	RAM (GB)	Ganho RAM (GB)	Speedup Médio	%
PA-Star (h2_all)	1171,9498		17,74			
PA-Star (h3_all)	951,0887	220,8611	16,09	1,65	1,232	18,85%

Fase 3						
Versão	Média Tempo (s)	Ganho Tempo (s)	RAM (GB)	Ganho RAM (GB)	Speedup Médio	%
PA-Star (h2_all)	0,001		0,00			
PA-Star (h3_all)	0,001	0	0,00	0	1,000	0,00%

Total						
Versão	Média Tempo (s)	Ganho Tempo (s)	RAM (GB)	Ganho RAM (GB)	Speedup Médio	%
PA-Star (h2_all)	1171,9591		18,101			
PA-Star (h3_all)	953,9292	218,0299	16,983	1,118	1,229	18,60%

Figura 5.2: Resultado dividido por fases do conjunto 2myr avaliado localmente.

total, mostrando que, nesse caso, a heurística $h_{3,all}$ possui um desempenho total muito melhor do que a $h_{2,all}$.

A Fase 3, que consiste no *traceback* para a reconstrução do alinhamento final, apresenta um tempo de execução desprezível (0,001 s) em ambas as versões, não possuindo relevância no desempenho global. Enquanto a Fase 2 lida com a complexidade polinomial da busca no espaço, o *traceback* possui complexidade linear, limitando-se a percorrer os ponteiros dos nós já validados para recuperar a solução ótima. Dessa forma, essa etapa é independente da heurística utilizada e reflete apenas o custo marginal de acesso à memória, tornando-se estatisticamente insignificante frente ao esforço computacional das fases anteriores.

A redução de 18,60% no tempo total e a diminuição da ocupação de memória evidenciam a eficácia da heurística baseada em trios para a comparação do conjunto `2myr.fasta` no nosso ambiente local. O limite inferior mais estrito fornecido por $h_{3,all}$ permitiu uma poda mais agressiva do espaço de busca, reduzindo o número de nós expandidos durante

a execução do algoritmo A^*, quando aplicado ao MSA [12, 10].

5.3.2 Execuções na AWS

Os testes efetuados na arquitetura ARM Graviton4 mostram resultados positivos de forma consistente em todos os conjuntos de dados. A Figura 5.3 apresenta uma visualização comparativa do ganho da heurística $h_{3,all}$ sobre a $h_{2,all}$. Nessa figura, o ganho é calculado em termos absolutos (Gain) e percentuais, se comparado à $h_{2,all}$.

1gpb

	h2_all			h3_all				
Th	Tempo (s)	RAM (Gb)		Tempo (s)	RAM (Gb)	Ganho (s)	Ganho (GB)	%
8	2355,52	88,50		2435,35	99,06	-79,83	-10,56	-3,39%
16	1218,16	102,93		1185,01	99,43	33,15	3,51	2,72%
32	865,17	122,44		601,54	101,04	263,63	21,41	30,47%
64	346,25	90,80		331,20	102,99	15,05	-12,19	4,35%

2myr

	h2_all			h3_all				
Th	Tempo (s)	RAM (Gb)		Tempo (s)	RAM (Gb)	Ganho(s)	Ganho (GB)	%
8	238,48	17,08		231,42	15,45	7,06	1,63	2,96%
16	117,83	17,11		110,44	15,49	7,39	1,61	6,27%
32	63,70	17,33		55,66	15,55	8,03	1,78	12,61%
64	44,38	17,39		38,10	16,00	6,28	1,39	14,15%

1sesA

	h2_all			h3_all				
Th	Tempo (s)	RAM (Gb)		Tempo (s)	RAM (Gb)	Ganho(s)	Ganho (GB)	%
8	574,94	23,92		521,34	21,97	53,60	1,95	9,32%
16	281,53	24,06		243,14	21,99	38,39	2,07	13,64%
32	149,74	26,46		124,98	22,38	24,76	4,08	16,53%
64	90,56	26,05		72,57	22,82	17,99	3,23	19,86%

arp

	h2_all			h3_all				
Th	Tempo (s)	RAM (Gb)		Tempo (s)	RAM (Gb)	Ganho(s)	Ganho (GB)	%
8	943,63	37,00		653,62	24,76	290,01	12,23	30,73%
16	463,25	37,35		310,27	24,99	152,98	12,36	33,02%
32	241,14	37,74		153,84	25,08	87,30	12,65	36,20%
64	138,38	38,35		87,38	25,61	51,00	12,74	36,85%

Figura 5.3: Resultados obtidos nos diferentes conjuntos de dados avaliados na AWS.

A execução do PA-Star com heurística $h_{3,all}$ usando o conjunto `arp` apresentou o desempenho mais expressivo, atingindo ganhos acima de 30% para 8, 16, 32 e 64 *threads* e um *speedup* de até 1,58x com 64 *threads*. É notável que, além da melhoria temporal, a versão $h_{3,all}$ teve, em diversos cenários, uma utilização de memória RAM inferior ou comparável à $h_{2,all}$ na AWS. Isso sugere que a redução drástica no número de nós armazenados na *OpenList* e *ClosedList* compensa o espaço adicional exigido para armazenar as matrizes tridimensionais linearizadas da heurística [10, 47].

5.3.3 *Speedup*

O *speedup* do PA-Star foi avaliado na nuvem AWS usando as heurísticas $h_{2,all}$ e $h_{3,all}$. O *speedup* foi determinado utilizando os dados apresentados na Figura 5.3, tendo como referência o desempenho com 8 núcleos (T_{ref}), conforme a Equação 5.1:

$$S(n) = \frac{T_8}{T_n} \tag{5.1}$$

em que T_8 representa o tempo de execução com 8 núcleos e T_n o tempo com n núcleos.

Speedup do h2,all

Os resultados de *speedup* para a heurística baseada em pares ($h_{2,all}$) na execução paralela indicam um ganho de desempenho, porém com uma perda de eficiência ao se atingir 64 núcleos (Tabela 5.3).

Tabela 5.3: Speedup relativo a 8 núcleos para a heurística $h_{2,all}$.

Núcleos	**Linear**	**1gpb**	**2myr**	**1sesA**	**arp**
8	1	1	1	1	1
16	2	1,933	2,023	2,042	2,036
32	4	2,722	3,743	3,839	3,913
64	8	6,802	5,373	6,348	6,819

Observa-se que, para o conjunto `2myr.fasta`, o *speedup* com 64 núcleos foi de 5,37x, distanciando-se do limite linear de 8,0. Este comportamento sugere que, em instâncias menores, o custo de gerenciamento de *threads* e a comunicação entre núcleos começam a rivalizar com o tempo reduzido de busca da Fase 2. A Figura 5.4 ilustra este comportamento.

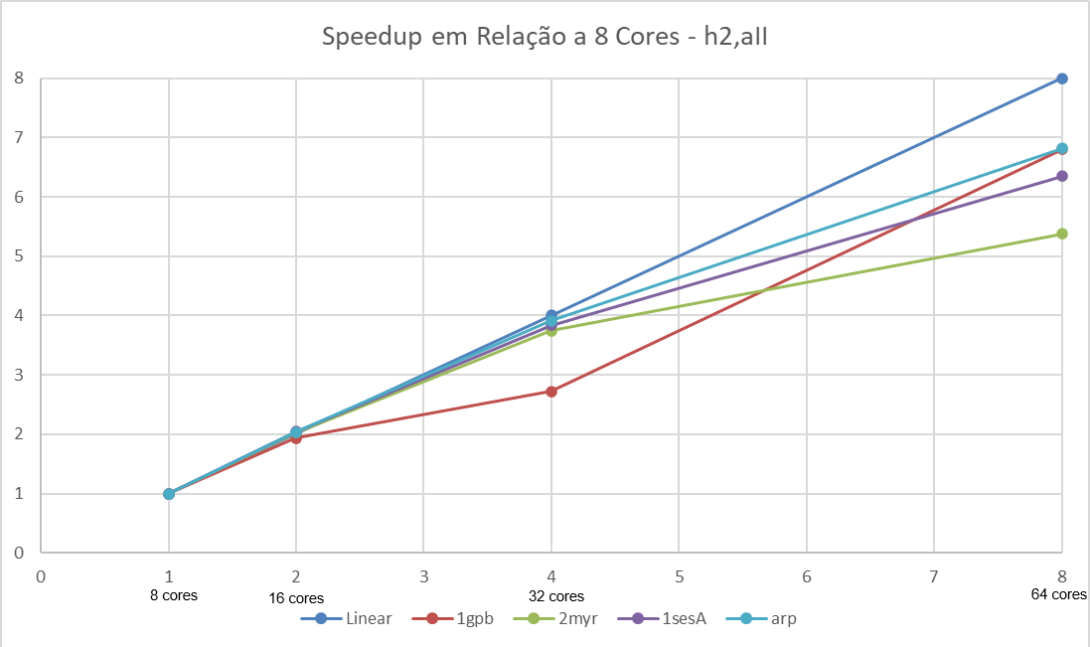

Figura 5.4: Speedup da heurística $h_{2,all}$

Speedup do h3,all

A heurística proposta baseada em trios ($h_{3,all}$) apresentou um comportamento de speedup superior, conforme detalhado na Tabela 5.4.

De forma notável, a $h_{3,all}$ apresentou um *speedup* levemente superlinear em 16 e 32 núcleos para a maioria dos conjuntos de dados, alcançando 4,24 para `arp`. Esse fenômeno está frequentemente ligado a uma utilização mais eficiente da memória, em que a divisão do problema possibilita que uma maior quantidade de dados permaneça nos níveis mais rápidos da hierarquia de memória. Ao alcançar 64 núcleos, o conjunto `arp` preservou um *speedup* de 7,48, mostram uma eficiência paralela de 93,5%. Isso representa uma eficiência

Tabela 5.4: Speedup relativo a 8 núcleos para a heurística $h_{3,all}$.

Núcleos	**Linear**	**1gpb**	**2myr**	**1sesA**	**arp**
8	1	1	1	1	1
16	2	2,055	2,095	2,144	2,106
32	4	4,048	4,157	4,171	4,248
64	8	7,353	6,074	7,183	7,480

significativamente superior aos 85,2% registrados pela versão $h_{2,all}$ na mesma instância. A Figura 5.5 ilustra esse ganho.

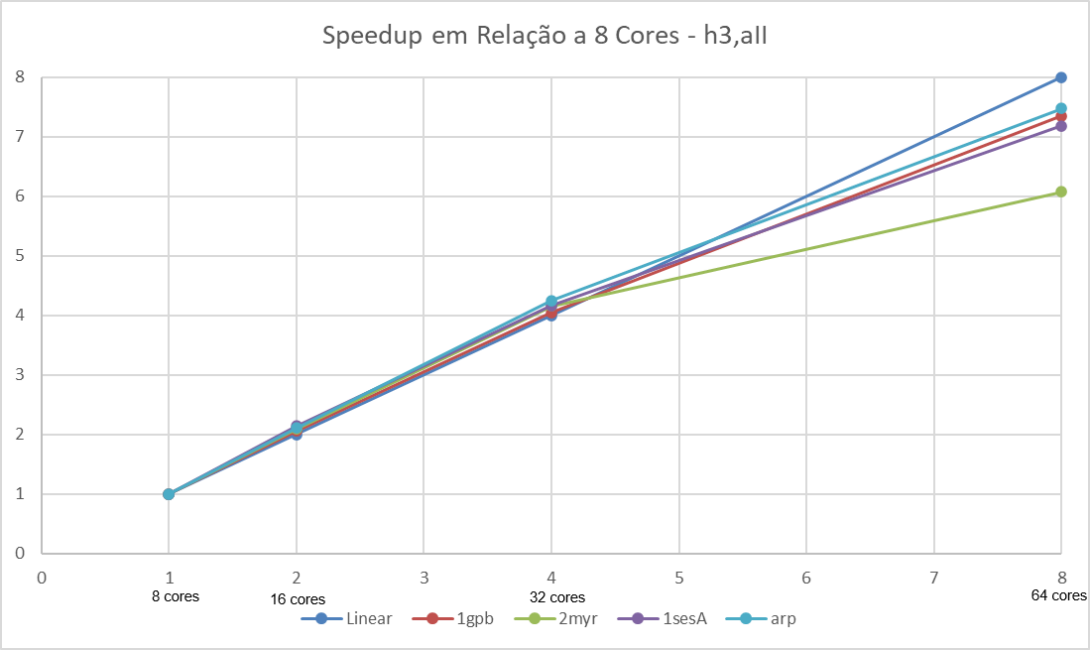

Figura 5.5: Speedup da heurística $h_{3,all}$

O maior *speedup* da $h_{3,all}$, se comparada à $h_{2,all}$ mostra as vantagens da estratégia de paralelização do cálculo de trios na Fase 1 e a eficácia da redução do espaço de busca na Fase 2, permitindo que o algoritmo de busca A^* explore o paralelismo de arquiteturas de CPUs modernas de forma eficiente.

5.3.4 Visualização do Espaço de Busca

Para melhor entender o comportamento da poda com $h_{3,all}$, foi feita uma visualização do espaço de busca em três dimensões, usando a família proteica PF03426 apresentada na Tabela 5.2. Essa análise possibilita a observação do comportamento do Pa-Star ao percorrer o grafo, mostrando a dispersão das iterações entre as heurísticas $h_{2,all}$ e $h_{3,all}$ [12] conforme as Figuras 5.6 e Figura 5.7.

A análise comparativa extraída das projeções (XY, XZ e YZ) indica que a heurística baseada em trios limita a busca a um volume significativamente menor. A Tabela 5.5 sintetiza os indicadores de exploração obtidos.

O código utilizado foi baseado em um repositório original[2] que foi posteriormente modificado para se adequar ao $h_{3,all}$[3].

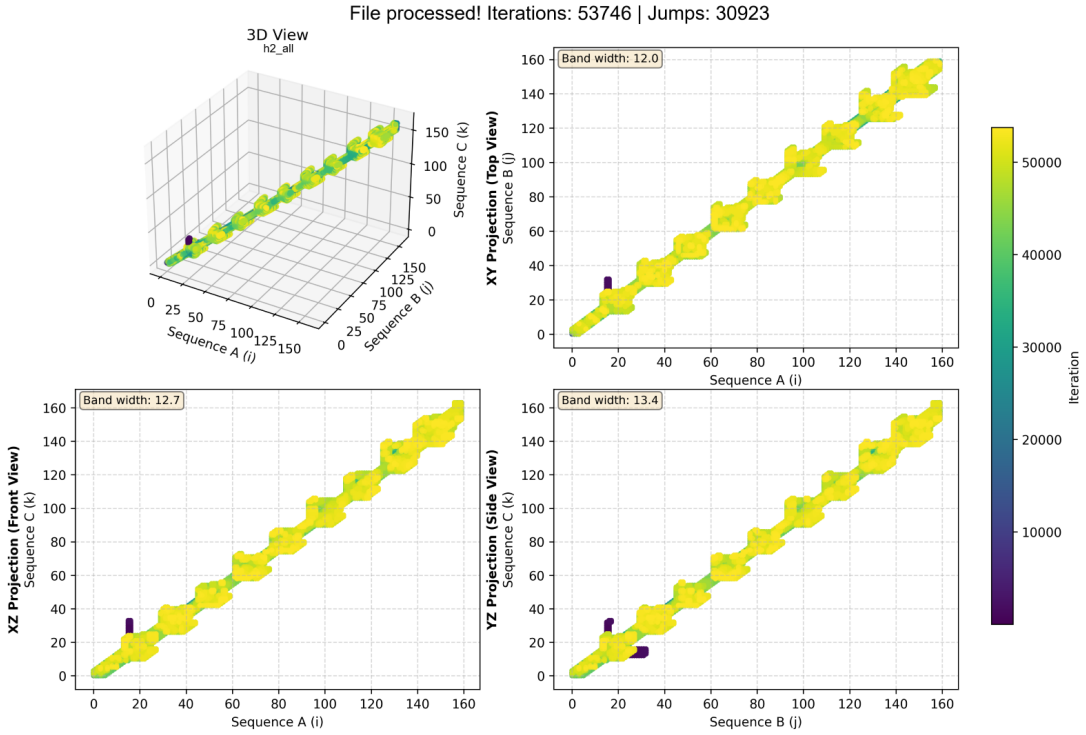

Figura 5.6: Projeções do Espaço de Busca usando h2,all

[2]Repositório original disponível em `https://github.com/edufrma/pa-star-rv`
[3]Fork disponível em `https://github.com/vncsmnl/pa-star-rv`

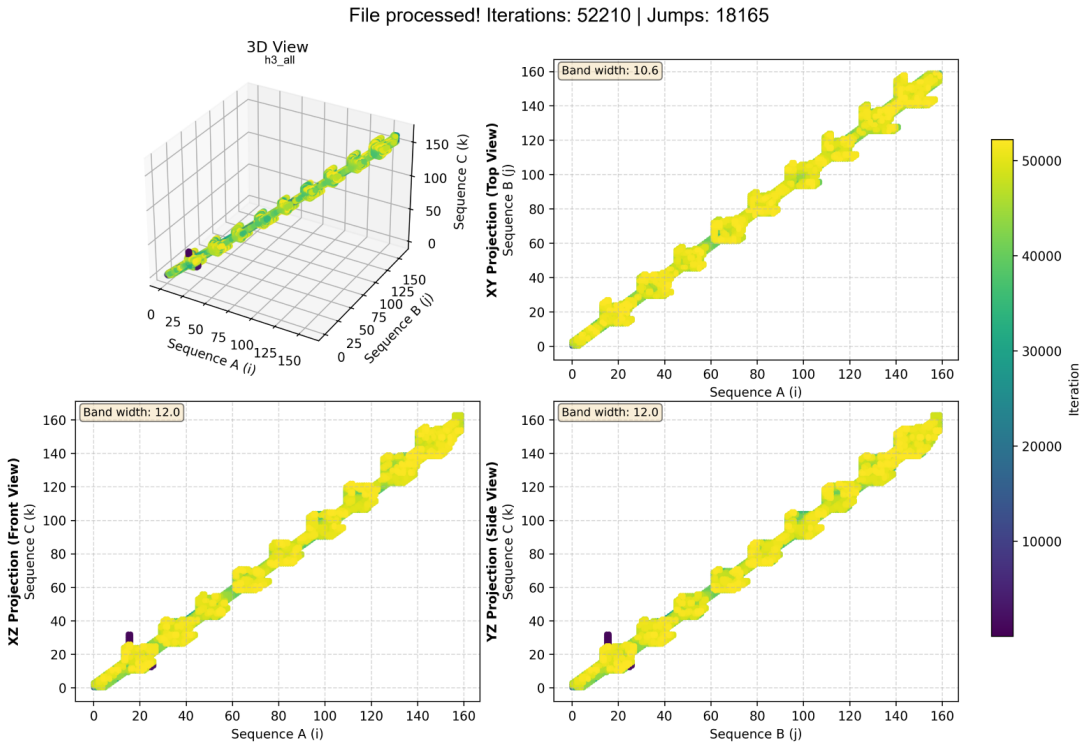

Figura 5.7: Projeções do Espaço de Busca usando h3,all

Tabela 5.5: Indicadores de exploração do espaço de busca para o conjunto PF03426.

Métrica	**Heurística** $h_{2,all}$	**Heurística** $h_{3,all}$	**Redução**
Iterações Totais	53.746	52.210	2,85%
Saltos (*Jumps*)	30.923	18.165	41,26%
Largura da Banda (Média)	12,70	11,53	9,21%

Os saltos (*jumps*) ocorrem cada vez que o algoritmo explora nós que não são vizinhos diretos dos nós atualmente abertos. Um valor alto para o *jump* indica que o algoritmo está explorando uma área maior do espaço de busca. Como pode ser visto, a heurística $h_{3,all}$ calcula um limite inferior mais preciso (*tighter lower bound*) [47], se comparada com a heurística $h_{2,all}$ (Figura. 5.6 e Figura. 5.7), apresenta uma diferença visível. Conceitualmente, inúmeros saltos pode indicar que a heurística está realizando várias reavaliações de estados ou que há muitos caminhos competitivos [10]. Por outro lado, poucos saltos indicam uma busca mais direta e focada no objetivo. Nesse sentido, a redução de 41,26% no número de saltos observada com a heurística $h_{3,all}$ indica uma orientação mais eficiente do algoritmo, caracterizando um comportamento de busca mais direcionado e com menor exploração de nós desnecessários, evidenciando o impacto positivo da heurística proposta.

As projeções ortogonais indicam que a largura da banda (*bandwidth*) da busca é reduzida na abordagem de trios (Figura. 5.7). A heurística de pares exibe uma banda de

12,70 aminoácidos nas projeções e, por outro lado, a abordagem baseada em trios diminui a banda para 11,53. Visualmente, isso resulta em um trajeto mais "fino" e focado na diagonal principal do cubo de busca, corroborando que a heurística baseada em trios capta dependências entre as sequências que as projeções par-a-par ignoram [47, 12].

Esta restrição do espaço de busca é o fator determinante para que a redução no tempo de execução total (Fase 2) compense o custo computacional mais elevado do pré-processamento paralelizado da heurística [10].

5.4 Ferramenta de Teste: MSA A-Star/PA-Star

Para avaliar as otimizações propostas neste trabalho, foi desenvolvida uma aplicação *web* chamada de *MSA App*, que fornece uma interface intuitiva para execução de alinhamentos múltiplos de sequências utilizando os algoritmos A-Star e PA-Star.

5.4.1 Visão Geral

A ferramenta consiste em uma aplicação *web* implementada em Python utilizando o *framework* Flask [59], oferecendo uma interface amigável para configuração, execução e análise dos experimentos. O sistema foi projetado para facilitar as execuções com diferentes configurações de parâmetros, conjuntos de dados de referência (*benchmarks*) e versões de algoritmos, permitindo uma avaliação do desempenho dos métodos propostos. A Figura 5.8 apresenta a tela inicial da ferramenta MSA_App.

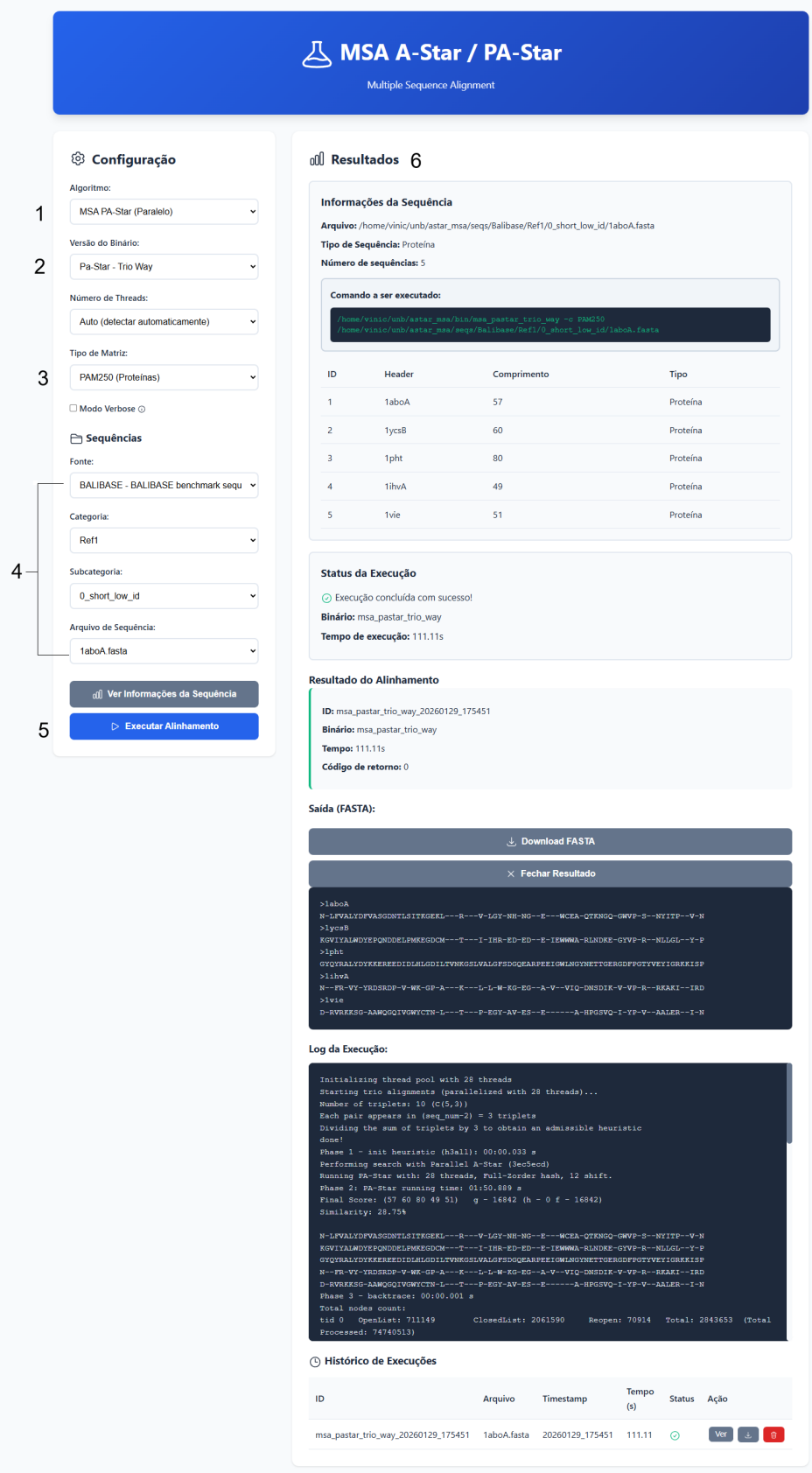

Figura 5.8: Interface do MSA APP

5.4.2 Arquitetura e Implementação

A aplicação segue o padrão arquitetural MVC (*Model-View-Controller*) adaptado para aplicações *web*, com separação clara entre camadas de apresentação, lógica de negócio e acesso a dados. A estrutura do projeto está organizada da seguinte forma:

- **Camada de Rotas** (`app/routes/`): Define os *endpoints* da API REST para gerenciamento de binários, sequências, execução de alinhamentos e visualização de resultados;
- **Camada de Serviços** (`app/services/`): Implementa a lógica de negócio, incluindo orquestração de execução dos algoritmos, gerenciamento de arquivos e processamento de resultados;
- **Camada de Utilidades** (`app/utils/`): Fornece funções auxiliares para *parsing* de arquivos FASTA, operações de sistema de arquivos e validações de segurança;
- **Interface Web** (`app/templates/` e `app/static/`): Implementa a camada de apresentação com interface HTML/CSS responsiva.

5.4.3 Funcionalidades Principais

A ferramenta oferece as seguintes funcionalidades para a condução dos experimentos:

1. **Suporte a Múltiplos Algoritmos**: Permite a execução e comparação entre as implementações A-Star e PA-Star, facilitando a análise de desempenho sequencial em relação ao paralelo;
2. **Gerenciamento de Benchmarks**: Integração nativa com conjuntos de dados de referência amplamente utilizados na literatura, incluindo:
    - BAliBASE [46] - conjunto de referência para validação de qualidade de alinhamentos;
    - Benchmark - sequências de teste padrão;
    - Matrizes de substituição NUC e PAM250 - para sequências nucleotídicas e proteicas;
3. **Configuração Flexível de Parâmetros**: Interface para ajuste de:
    - Tipo de matriz de custo/substituição (PAM250, etc.);
    - Número de *threads* para execução paralela;

- Nível de verbosidade dos *logs*;

4. **Rastreamento de Resultados**: Armazenamento e organização automática dos resultados de execução, incluindo:
   - Alinhamentos produzidos;
   - Métricas de desempenho (tempo de execução, uso de memória);
   - *Logs* detalhados de execução;
5. **API REST**: *Endpoints* documentados para integração com *scripts* de automação e análise em lote.

5.4.4 Fluxo de Utilização nos Experimentos

O processo típico de experimentação utilizando a ferramenta compreende as seguintes etapas (Figura 5.8):

1. **Seleção do Algoritmo**: Escolha entre A-Star ou PA-Star através da interface web;
2. **Escolha do Binário**: Seleção da versão compilada do algoritmo a ser testada;
3. **Configuração de Parâmetros**: Definição da matriz de substituição, número de threads (para PA-Star) e outras configurações;
4. **Seleção de Sequências**: Navegação e seleção de arquivos FASTA dos conjuntos de *benchmark* disponíveis;
5. **Execução**: Disparo do processo de alinhamento com *timeout* configurável;
6. **Análise de Resultados**: Visualização e exportação dos resultados através da interface ou API.

5.4.5 Vantagens para a Pesquisa

A utilização desta ferramenta como plataforma de testes proporcionou os seguintes benefícios ao desenvolvimento da pesquisa:

- **Reprodutibilidade**: Todas as execuções são registradas com seus parâmetros, permitindo replicação exata dos experimentos;
- **Execução automatizada**: A interface web elimina a necessidade de *scripts* manuais para cada experimento;

- **Organização**: Estrutura padronizada para armazenamento de datasets e resultados;
- **Comparabilidade**: Facilita a comparação direta entre diferentes implementações e configurações;
- **Flexibilidade**: Arquitetura permite adicionar novos algoritmos e *benchmarks* sem modificações estruturais.

5.4.6 Disponibilidade

O código-fonte da ferramenta está disponível publicamente sob licença MIT no repositório GitHub[4]. A documentação completa, incluindo instruções de instalação e uso da API, está incluída no repositório.

[4]Repositório da aplicação em Flask `https://github.com/vncsmnl/msa_app`

Capítulo 6

Conclusão

Esta Dissertação de Mestrao abordou o desafio da complexidade computacional do problema de MSA, classificado como NP-Completo, propondo e avaliando otimizações algorítmicas para a ferramenta PA-Star. A investigação concentrou-se na superação das limitações impostas pelas heurísticas baseadas em pares, frequentemente utilizadas no estado da arte, que, embora admissíveis, tendem a gerar limites inferiores pouco restritivos e resultam na exploração excessiva do espaço de busca.

6.1 Síntese do Trabalho Realizado

O objetivo principal deste trabalho foi aumentar a eficiência computacional de métodos exatos de alinhamento, sem sacrificar a garantia de otimalidade da solução. Para atingir tal propósito, foi implementada na ferramenta PA-Star a heurística $h_{3,all}$, baseada no alinhamento exato de trios de sequências, fundamentada teoricamente na capacidade de capturar dependências locais de ordem superior ignoradas pelas abordagens par-a-par [12, 47].

Reconhecendo o custo computacional cúbico associado ao pré-processamento desta nova heurística, desenvolveu-se uma arquitetura de paralelismo baseada em tarefas (*task parallelism*) para a etapa de cálculo dos trios. Esta estratégia permitiu a distribuição eficiente da carga de trabalho em ambientes de memória compartilhada, mitigando o impacto temporal da fase inicial e viabilizando a aplicação prática da heurística $h_{3,all}$ em instâncias de maior complexidade.

Adicionalmente, foi desenvolvida uma ferramenta de suporte à experimentação, denominada MSA App, que permitiu a execução sistemática, reprodutível e comparativa dos algoritmos, facilitando a análise de métricas de desempenho sobre conjuntos de dados de referência como o BAliBASE 4, uma extensão do conjunto de dados descrito originalmente por Thompson et al. [46].

6.2 Resultados Alcançados

A avaliação experimental evidenciou, para os conjuntos comparados, que o maior investimento computacional no pré-processamento da heurística $h_{3,all}$ foi compensado pela redução do espaço de busca durante a execução do algoritmo A^*. Os resultados obtidos nos mostraram:

- **Redução do Espaço de Busca:** A utilização da heurística baseada em trios resultou em uma poda mais agressiva do grafo de estados. Em testes visuais realizados com a família proteica PF03426, observou-se uma redução de 41,26% no número de saltos (*jumps*) e uma diminuição na largura da banda da busca, apontando um direcionamento mais eficiente do algoritmo em direção à solução ótima [10].
- **Desempenho Temporal:** Apesar do aumento no tempo da Fase 1 (pre-processamento e cálculo da heurística), a redução do tempo na Fase 2 (busca A^*) foi suficiente para diminuir o tempo total de execução. No conjunto de dados *2myr*, obteve-se uma redução média de 18,60% no tempo total em ambiente local, validando a eficácia da abordagem proposta.
- **Desempenho na nuvem:** Nos testes conduzidos em ambiente de nuvem (AWS Graviton4), a heurística $h_{3,all}$ apresentou um resultado superior à abordagem tradicional $h_{2,all}$. Para o conjunto *arp*, alcançou-se um *speedup* de 7,48 com 64 núcleos, relevando uma eficiência paralela de 93,5%, contra 85,2% da versão original.
- **Custo de Memória:** Observou-se que a redução no número de nós expandidos e armazenados nas listas *Open* e *Closed* compensou o uso de memória adicional necessário para armazenar as matrizes de trios linearizadas, resultando em um consumo global de memória RAM menor ou equivalente em diversos cenários de teste [10].

Em suma, as otimizações propostas elevaram o patamar de desempenho da ferramenta PA-Star, tornando viável a obtenção de alinhamentos exatos com menor tempo de execução e menor consumo de memória RAM.

6.3 Trabalhos Futuros

Apesar dos avanços obtidos, o problema do alinhamento múltiplo exato permanece um desafio aberto, com diversas oportunidades para investigação subsequente. Como desdobramentos naturais desta pesquisa, sugerem-se as seguintes linhas de atuação:

- **Implementação de Penalidades por Lacuna:** A versão atual utiliza um modelo de custo linear para lacunas (gaps). Trabalhos futuros podem incorporar o modelo de penalidade por lacuna (*Affine Gap Penalty*), que penaliza a abertura de uma lacuna de forma distinta de sua extensão. Este modelo é biologicamente mais realista, embora aumente a quantidade de cálculo das matrizes de programação dinâmica [2, 4].

- **Paralelismo em Múltiplos Nós (Memória Distribuída):** A atual paralelização limita-se a ambientes de memória compartilhada. A extensão da ferramenta para arquiteturas de memória distribuída, utilizando bibliotecas como MPI (*Message Passing Interface*), permitiria a execução em *clusters* de computadores, ampliando significativamente a capacidade de processamento e memória para lidar com conjuntos de sequências genômicas mais longas [10].

- **Investigação de novas Heurísticas:** Um possível avanço da ferramenta poderia ser a pesquisa e análise de heurísticas usando quartetos ($h_{4,all}$) ou quintetos ($h_{5,all}$). É preciso investigar se a redução do espaço de busca justifica a sua complexidade. Porém, ainda não temos na literatura uma fundamentação teórica para justificar a admissibilidade dessas funções e analisar a possibilidade de sua utilização prática em algoritmos exatos.

- **Desenvolvimento de um *Framework* Unificado:** Propõe-se a criação de uma plataforma integrada que reúna múltiplos algoritmos, tanto exatos quanto heurísticos (como MAFFT [27], MUSCLE [28] e Clustal Omega [31]). Tal *framework* facilitaria a seleção da estratégia mais adequada com base nas características dos conjuntos de entrada, promovendo uma interoperabilidade maior entre métodos exatos e heurísticos.

- **Aplicação de Inteligência Artificial e Modelos de Atenção:** Uma direção promissora reside na integração de técnicas avançadas de aprendizado de máquina. O objetivo seria desenvolver e validar um modelo de MSA baseado em arquiteturas de Aprendizado Profundo, em particular modelos fundamentados em mecanismos de atenção (*Transformers*). Essa abordagem visa maximizar a consistência estrutural e evolutiva dos alinhamentos, superando as limitações conhecidas das abordagens que se baseiam exclusivamente na otimização de funções de pontuação matemáticas tradicionais [7, 60].

Referências

[1] Mount, David W.: *Bioinformatics - sequence and genome analysis (2. ed.)*. Cambridge University Press, 2004. 1, 4, 9

[2] Durbin, Richard, Sean R. Eddy, Anders Krogh e Graeme J. Mitchison: *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998. 1, 5, 9, 10, 63

[3] Notredame, Cédric: *Recent evolutions of multiple sequence alignment algorithms*. PLoS Comput. Biol., 3, 2007. 1, 5, 7, 12, 13, 14, 15, 17

[4] Gusfield, Dan: *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997. 1, 4, 5, 6, 7, 8, 9, 11, 17, 47, 63

[5] Carrillo, Humberto e David Lipman: *The multiple sequence alignment problem in biology*. SIAM journal on applied mathematics, 48(5):1073–1082, 1988. 1, 6, 8, 9, 12

[6] Bonizzoni, Paola e Gianluca Della Vedova: *The complexity of multiple sequence alignment with sp-score that is a metric*. Theor. Comput. Sci., 259:63–79, 2001. 1, 6, 9

[7] Zhang, Yongqing, Qiang Zhang, Jiliu Zhou e Quan Zou: *A survey on the algorithm and development of multiple sequence alignment*. Briefings Bioinform., 23, 2022. 1, 6, 12, 14, 20, 21, 63

[8] Edgar, Robert C: *Quality measures for protein alignment benchmarks*. Nucleic acids research, 38(7):2145–2153, 2010. 1, 17, 47

[9] Sundfeld, Daniel, George Teodoro e Alba Cristina Magalhaes Alves de Melo: *Parallel a-star multiple sequence alignment with locality-sensitive hash functions*. Em *Ninth International Conference on Complex, Intelligent, and Software Intensive Systems, CISIS 2015, Santa Catarina, Brazil, July 8-10, 2015*, páginas 342–347, 2015. 1, 18, 19, 25, 32

[10] Sundfeld, Daniel, Caina Razzolini, George Teodoro, Azzedine Boukerche e Alba Cristina Magalhaes Alves de Melo: *Pa-star: A disk-assisted parallel a-star strategy with locality-sensitive hash for multiple sequence alignment*. J. Parallel Distributed Comput., 112:154–165, 2018. 1, 2, 18, 19, 24, 25, 26, 30, 31, 32, 41, 51, 55, 56, 62, 63

[11] Reddy, Bharath e Richard Fields: *Performance analysis of multiple sequence alignment tools*. Em *Proceedings of the 2024 ACM Southeast Conference, ACM SE 2024, Marietta, GA, USA, April 18-20, 2024*, páginas 167–174, 2024. 1, 25

[12] McNaughton, Matthew, Paul Lu, Jonathan Schaeffer e Duane Szafron: *Memory-efficient a* heuristics for multiple sequence alignment.* Em *AAAI/IAAI*, páginas 737–743, 2002. 2, 18, 19, 27, 30, 31, 33, 47, 48, 51, 53, 56, 61

[13] Needleman, Saul B e Christian 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. 5, 32

[14] Gondro, Cedric e Brian P Kinghorn: *A simple genetic algorithm for multiple sequence alignment.* Genetics and Molecular Research, 6(4):964–982, 2007. 5, 14, 21

[15] Gotoh, Osamu: *Multiple sequence alignment: algorithms and applications.* Advances in biophysics, 36:159–206, 1999. 6, 7

[16] Ranwez, V. e N.N. Chantret: *Strengths and limits of multiple sequence alignment and filtering methods.* Phylogenetics in the Genomic Era, 2020. 7

[17] Gotoh, Osamu: *A weighting system and aigorithm for aligning many phylogenetically related sequences.* Bioinformatics, 11(5):543–551, 1995. 8

[18] Lermen, Martin e Knut Reinert: *The practical use of the a* algorithm for exact multiple sequence alignment.* J. Comput. Biol., 7:655–671, 2000. 12, 17, 18

[19] Hogeweg, Paulien e Ben Hesper: *The alignment of sets of sequences and the construction of phyletic trees: an integrated method.* Journal of molecular evolution, 20:175–186, 1984. 12

[20] Thompson, Julie D., Desmond G. Higgins e Toby J. Gibson: *Clustal w: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice.* Nucleic Acids Research, 22(22):4673–4680, 1994, ISSN 1362-4962. 12, 13, 15, 16

[21] Sievers, Fabian, Andreas Wilm, David Dineen, Toby J Gibson, Kevin Karplus, Weizhong Li, Rodrigo Lopez, Hamish McWilliam, Michael Remmert, Johannes Söding *et al.*: *Fast, scalable generation of high-quality protein multiple sequence alignments using clustal omega.* Molecular systems biology, 7(1):539, 2011. 13, 16

[22] Blackshields, Gordon, Fabian Sievers, Weizhong Shi, Andreas Wilm e Desmond G. Higgins: *Sequence embedding for fast construction of guide trees for multiple sequence alignment.* Algorithms for Molecular Biology, 5:21, 2010. 13

[23] Söding, Johannes: *Protein homology detection by hmm-hmm comparison.* Bioinformatics, 21(7):951–960, 2005. 13

[24] Notredame, Cédric, Desmond G Higgins e Jaap Heringa: *T-coffee: A novel method for fast and accurate multiple sequence alignment.* Journal of molecular biology, 302(1):205–217, 2000. 13, 14, 16

[25] Ibrahim, M.K., U.K. Yusof, T.A.E. Eisa e M. Nasser: *Bioinspired algorithms for multiple sequence alignment: a systematic review and roadmap.* Applied Sciences, 14(6):2433, 2024. 14

[26] Nakamura, Tsukasa, Kazunori D. Yamada, Kentaro Tomii e Kazutaka Katoh: *Parallelization of MAFFT for large-scale multiple sequence alignments.* Bioinform., 34:2490–2492, 2018. 15, 16, 23, 26

[27] Katoh, Kazutaka, Kazuharu Misawa, Kei ichi Kuma e Takashi Miyata: *Mafft: a novel method for rapid multiple sequence alignment based on fast fourier transform.* Nucleic acids research, 30(14):3059–3066, 2002. 15, 63

[28] Edgar, Robert C: *Muscle: multiple sequence alignment with high accuracy and high throughput.* Nucleic acids research, 32(5):1792–1797, 2004. 15, 16, 63

[29] Pais, Fabrício S, Elene C de Renzende, Sérgio T de Souza, Priscila S de Oliveira, Patrícia Grynberg, Keith Mockaitis e Anderson Silva: *A comprehensive benchmark study of multiple sequence alignment methods for protein sequences.* PloS one, 9(5):e96278, 2014. 15

[30] Lassmann, Timo e Erik LL Sonnhammer: *A comprehensive benchmark study of multiple sequence alignment methods: current challenges and future perspectives.* PloS one, 6(3):e18093, 2011. 15

[31] Sievers, Fabian, Andreas Wilm, David Dineen, Toby J Gibson, Kevin Karplus, Weizhong Li, Rodrigo Lopez, Hamish McWilliam, Michael Remmert, Johannes Söding *et al.*: *Clustal omega, a new program for multiple sequence alignment.* Molecular systems biology, 7(1):539, 2011. 16, 21, 63

[32] Sievers, Fabian, Daniel G Dineen, Andreas Wilm e Desmond G Higgins: *Class of multiple sequence alignment algorithm affects genomic analysis.* Molecular biology and evolution, 30(3):642–653, 2013. 17

[33] Morrison, David A: *Multiple sequence alignment is not a solved problem.* arXiv preprint arXiv:1808.07717, 2018. 17

[34] Fakhravar, H: *Combining heuristics and exact algorithms: A review. arxiv 2022.* arXiv preprint arXiv:2202.02799, 2022. 17, 27

[35] Hart, Peter E, Nils J Nilsson e Bertram Raphael: *A formal basis for the heuristic determination of minimum cost paths.* IEEE transactions on Systems Science and Cybernetics, 4(2):100–107, 1968. 18

[36] Ikeda, Takahiro e Hiroshi Imai: *Improvement of the a(*) algorithm for multiple sequence alignment.* Genome Informatics, 5:90–99, 1994. `https://doi.org/10.11234/gi1990.5.90`. 18

[37] Yoshizumi, Takayuki, Teruhisa Miura e Toru Ishida: *A* with partial expansion for large branching factor problems.* Em *AAAI/IAAI*, páginas 923–929, 2000. 18, 19

[38] Sundfeld, Daniel, George Teodoro e Alba C. M. A. Melo: *Pa-star2: Fast optimal multiple sequence alignment for asymmetric multicore processors.* Em *33rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2025, Turin, Italy, March 12-14, 2025*, páginas 146–153, 2025. 19

[39] Smirnov, Vladimir: *Recursive MAGUS: scalable and accurate multiple sequence alignment*. PLoS Comput. Biol., 17, 2021. 20, 23, 26

[40] Zhang, Pinglu, Huan Liu, Yanming Wei, Yixiao Zhai, Qinzhong Tian e Quan Zou: *Fmalign2: a novel fast multiple nucleotide sequence alignment method for ultralong datasets*. Bioinform., 40, 2024. 20, 23, 26

[41] Do, Chuong B., Michael Brudno e Serafim Batzoglou: *PROBCONS: probabilistic consistency-based multiple alignment of amino acid sequences*. Em *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA*, páginas 703–708, 2004. 21

[42] Rubio-Largo, Álvaro, Mauro Castelli, Leonardo Vanneschi e Miguel A. Vega-Rodríguez: *A parallel multiobjective metaheuristic for multiple sequence alignment*. J. Comput. Biol., 25:1009–1022, 2018. 22, 26

[43] Lassmann, Timo: *Kalign 3: multiple sequence alignment of large datasets*. Bioinform., 36:1928–1929, 2020. 24, 26

[44] Moshiri, Niema: *Viralmsa: massively scalable reference-guided multiple sequence alignment of viral genomes*. Bioinform., 37:714–716, 2021. 24, 26

[45] Li, Heng: *Minimap2: pairwise alignment for nucleotide sequences*. Bioinformatics, 34(18):3094–3100, 2018. 24

[46] Thompson, J. D., P. Koehl, R. Ripp e O. Poch: *Balibase 3.0: Latest developments of the multiple sequence alignment benchmark*. Proteins: Structure, Function, and Bioinformatics, 61(1):127–136, 2005. 25, 27, 47, 58, 61

[47] Colbourn, Charles J e Sudhir Kumar: *Lower bounds on multiple sequence alignment using exact 3-way alignment*. BMC Bioinformatics, 8(1), abril 2007, ISSN 1471-2105. 26, 27, 30, 31, 33, 48, 51, 55, 56, 61

[48] Dean, Jeffrey e Sanjay Ghemawat: *Mapreduce: Simplified data processing on large clusters*. Communications of the ACM, 51(1):107–113, 2004. 35, 41

[49] Bull, Jonathan e Jim M. Mellor-Crummey: *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*. Prentice Hall, 2013. 35, 41

[50] Ghosh, Subrata e Ashutosh Gupta: *Load Balancing for Parallel Computing: A Solution to the Dynamic Load Balancing Problem*. Springer, 2011. 35, 41

[51] Wang, Liu e Xiaodong Li: *Dynamic Load Balancing for Parallel Systems*. Springer, 2014. 35

[52] Cormen, Thomas H, Charles E Leiserson, Ronald L Rivest e Clifford Stein: *Introduction to algorithms*. MIT press, 2022. 36

[53] Hennessy, John L. e David A. Patterson: *Computer Architecture: A Quantitative Approach*. Elsevier, 5th edição, 2011. 36

[54] Foster, Ian: *Designing and Building Parallel Programs*. Addison-Wesley, 1995. 36

[55] Wenzel, Jan e Martin Neumann: *Static Partitioning and Dynamic Scheduling of Parallel Workloads*. Springer, 2014. 36

[56] Batista, Luis e Paulo Silva: *Load Balancing in Distributed Computing Systems*. Springer, 2017. 36

[57] Almasi, George e Alan Gottlieb: *Highly Parallel Computing*. Benjamin/Cummings, 1994. 37

[58] Biswas, Anirban e Suman Chowdhury: *Parallel Processing and Applications*. Springer, 2018. 37

[59] Project, Pallets: *Flask: Web development, one drop at a time*, 2023. `https://flask.palletsprojects.com/`. 56

[60] Ibrahim, Mohammed K, Umi Kalsom Yusof, Taiseer Abdalla Elfadil Eisa e Maged Nasser: *Enhanced genetic method for optimizing multiple sequence alignment*. Mathematics, 11(22):4578, 2023. 63