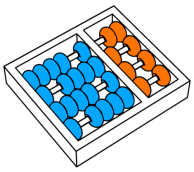
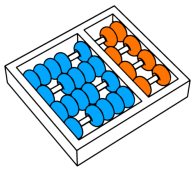


Sergio Jeferson Rafael Ordine

Alinhamento múltiplo de proteínas utilizando algoritmos genéticos

CAMPINAS
2015



Universidade Estadual de Campinas
Instituto de Computação

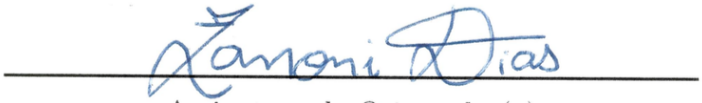
Sergio Jeferson Rafael Ordine

Alinhamento múltiplo de proteínas utilizando algoritmos genéticos

Orientador(a): **Prof. Dr. Zanoni Dias**

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Computação da Universidade Estadual de Campinas para obtenção do título de Mestre em Ciência da Computação.

Este exemplar corresponde à versão final da Dissertação defendida por Sergio Jeferson Rafael Ordine, sob orientação de Prof. Dr. Zanoni Dias.



Assinatura do Orientador(a)

CAMPINAS
2015

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Maria Fabiana Bezerra Muller - CRB 8/6162

Or2a

Ordine, Sergio Jeferson Rafael, 1974-
Alinhamento múltiplo de proteínas utilizando algoritmos genéticos / Sergio Jeferson Rafael Ordine. – Campinas, SP : [s.n.], 2015.

Orientador: Zanoni Dias.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Computação.

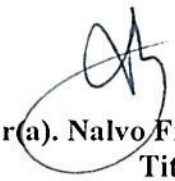
1. Alinhamento múltiplo de sequências. 2. Proteínas. 3. Algoritmos genéticos. 4. Biologia computacional. I. Dias, Zanoni,1975-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Protein multiple sequence alignment using genetic algorithms
Palavras-chave em inglês:
Multiple sequence alignment
Proteins
Genetic algorithms
Computational biology
Área de concentração: Ciência da Computação
Titulação: Mestre em Ciência da Computação
Banca examinadora:
Zanoni Dias [Orientador]
Nalvo Franco de Almeida Junior
Guilherme Pimentel Telles
Data de defesa: 03-07-2015
Programa de Pós-Graduação: Ciência da Computação

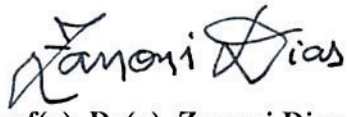
TERMO DE APROVAÇÃO

Defesa de Dissertação de Mestrado em Ciência da Computação, apresentada pelo(a) Mestrando(a) **Sergio Jeferson Rafael Ordine**, aprovado(a) em **03 de julho de 2015**, pela Banca examinadora composta pelos Professores(as) Doutores(as):



Prof(a). Dr(a). Nalvo Franco de Almeida Junior
Titular

Prof(a). Dr(a). Guilherme Pimentel Telles
Titular



Prof(a). Dr(a). Zanoni Dias
Presidente

Instituto de Computação
Universidade Estadual de Campinas

Alinhamento múltiplo de proteínas utilizando algoritmos genéticos

Sergio Jeferson Rafael Ordine

03 de julho de 2015

Banca Examinadora:

- Prof. Dr. Zanoni Dias (Supervisor/*Orientador*)
- Prof. Dr. Nalvo Franco de Almeida Junior
  Faculty of Computing - UFMS
- Prof. Guilherme Pimentel Telles
  Institute of Computing - UNICAMP
- Profa. Dra. Maria Emilia Machado Telles Walter
  Institute of Exact Sciences - UnB (Substitute/*Suplente*)
- Prof. Dr. Felipe Rodrigues da Silva
  Embrapa Informática Agropecuária (Substitute/*Suplente*)

Abstract

MSA, that stands for Multiple Sequence Alignment, is one of the most important tools on bioinformatics and computational biology fields. Its usage includes phylogenetic trees creation, motifs and conservated domains prediction, as well as protein secondary and tertiary structure and function prediction.

Genetic algorithms are heuristics for local search used on optimization problems, inspired on the natural selection concepts: a population of possible solution is created and iterativelly mate and selected in order to search for a suitable solution to the target problem.

This dissertation describes the study we proceed on using genetic algorithms to solve the multiple sequence alignment problem.

As results of this work, two tools were developed: ALGAe, a genetic algorithm execution environment for MSA solving, and Anubis, a multiple sequence alignment viewer that allows comparing a given MSA against a benchmark, in order to find discrepancies among them.

Resumo

Uma das ferramentas mais importantes no campo da bioinformática e biologia computacional é o alinhamento múltiplo de sequências (MSA, do inglês *Multiple Sequence Alignment*). Sua função abrange a criação de árvores filogenéticas, identificação de *motifs* e domínios conservados, assim como a predição de funções das proteínas e de suas estruturas secundárias e terciárias.

Algoritmos genéticos são heurísticas de busca local, utilizadas para problemas de otimização, inspirados nos conceitos da seleção natural: uma população de possíveis soluções é criada e iterativamente cruzada e selecionada buscando um resultado mais adequado para o problema sendo tratado.

Esta dissertação descreve o estudo realizado no uso de algoritmos genéticos para a resolução do problema de alinhamento múltiplo de proteínas.

Como resultado deste trabalho, duas ferramentas foram desenvolvidas: ALGAe, um ambiente para a execução de um algoritmo genético visando o alinhamento múltiplo de proteínas e o Anubis, um visualizador de alinhamentos que permite a comparação entre dois alinhamentos para o mesmo conjunto de proteínas, visando identificar as discrepâncias entre eles.

Aos meus pais Altamyr e Léa Ordine e à minha esposa Daniele Ordine.

Agradecimentos

Gostaria de agradecer a todos aqueles que, direta ou indiretamente, me auxiliaram de forma indelével na realização deste trabalho.

Agradeço a Deus por me dar a força necessária para persistir neste intento e acalentar o meu coração nos momentos mais difíceis.

Ao meu orientador, Zanoni Dias, pelo suporte, paciência e apoio, sempre respeitando minhas opiniões e criatividade.

A André Almeida pela parceria e importante troca de idéias. Também a Alex Grilo, pelo trabalho conjunto no início deste projeto.

Ao Instituto de Pesquisas Eldorado por auxiliar com que este trabalho fosse realizado concomitantemente às minhas obrigações profissionais. Em especial as seguintes pessoas: Marcelo Henrique de Souza e Danilo Angelo, pela compreensão nos momentos em que precisei me ausentar ou dividir a atenção entre as duas obrigações e a Alessandra Peguim Rosa, pelo suporte com os elementos visuais e diagramação de diversos trabalhos.

Por fim, agradeço à minha esposa, Daniele Ordine, sempre companheira, que me apoiou e auxiliou durante todo este período. Agradeço também aos meus pais, Altamyr Ordine e Léa Ordine, por todo o suporte dado na minha vida acadêmica. Aos três dedico este trabalho.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Alinhamento Múltiplo de Sequências (MSA, do inglês *Multiple Sequence Alignment*) é uma ferramenta muito importante nos campos da Bioinformática e Biologia Computacional. Suas aplicações abrangem o suporte a construção de árvores filogenéticas, a identificação de *motifs* e domínios conservados e a predição de funções de proteínas, assim como de suas estruturas secundárias e terciárias [61, 81]. Além disto é útil também na definição de *primers* para PCR (*Polimerase Chain Reaction*) e validação de dados [61].

O problema de alinhamento múltiplo de sequências pode ser descrito como: dado um conjunto de três ou mais sequências de caracteres e um conjunto de regras de comparação, organizá-las de forma que seus caracteres sejam alinhados em colunas (sendo que cada linha representa uma sequência) de forma a maximizar a similaridade entre elas, de acordo com o conjunto de regras dado.

Como tanto sequências de DNA como de proteínas podem ser representadas como sequências de caracteres (onde cada caracter representa um nucleotídeo, no caso do DNA, ou um aminoácido, no caso de uma proteína), pode-se definir métodos de comparação que permitam utilizar a solução do problema de MSA para estas sequências, buscando evidenciar uma origem evolucionária comum entre elas [30].

O problema de MSA é altamente complexo e multidisciplinar, envolvendo problemas biológicos, estatísticos e computacionais.

Do ponto de vista biológico, três grandes dificuldades técnicas se destacam: a escolha das sequências a serem alinhadas, a escolha de uma função de comparação adequada e a otimização desta função [81]. Em termos da função de comparação e sua otimização a grande dificuldade é fazer com que esta reflita com precisão as similaridades bioquímicas das sequências e sua origem evolucionária comum, garantindo a sua qualidade [61, 84].

Em termos computacionais, este problema é reconhecidamente MAX-SNP-Difícil [58, 125], o que torna o esforço computacional praticamente proibitivo para solucionar, de forma ótima, a maior parte das instâncias práticas deste problema.

Devido a impossibilidade computacional da resolução deste problema, contraposta à sua importância prática, diversas heurísticas foram propostas para a obtenção de bons resultados para o problema de MSA. Entre estes destacam-se duas categorias principais: algoritmos progressivos e iterativos.

Algoritmos progressivos [37] ordenam as sequências alvo de acordo com algum critério e, de forma incremental, montam a solução (alinhamento de todas as sequência alvo) através da inclusão de uma nova sequência, ou de um alinhamento de um sub-conjunto de sequências, à solução parcial construída até então.

Os algoritmos progressivos são os mais utilizados atualmente, sendo que uma das aplicações mais conhecidas baseada neste tipo de algortimo é o ClustalW [117]. Uma das grandes vantagens deste tipo de abordagem é o seu desempenho. Devido a sua natureza, porém, este tipo de algoritmo não consegue recuperar-se de suas decisões prévias, o que pode causar uma degeneração do resultado final (em especial no caso de uma má decisão ser tomada em um passo muito inicial do processo). Reside portanto aí sua principal desvantagem.

Algoritmos iterativos, por outro lado, são estruturados na forma de passos nos quais a solução é gradualmente refinada. Existem diversas soluções iterativas para o problema de MSA baseadas em abordagens distintas como: modelos ocultos de Markov, arrefecimento simulado (*simulated annealing*) [62], amostragem de Gibbs (*Gibbs sampling*) [66], algoritmos genéticos [42, 82], entre outros.

Os algoritmos iterativos demandam, em geral, mais tempo de processamento que algoritmos progressivos. Contudo, são capazes de efetuar buscas mais amplas no espaço de soluções sendo mais aptos a sobrepujar decisões ruim tomadas em etapas anteriores e encontrar resultados melhores, algo que não ocorre com os algoritmos progressivos. Isto faz com que estes algoritmos possam ser vistos como alternativas para aqueles ou, pelo menos, como possíveis refinadores para as soluções encontradas de forma progressiva, usando os benefícios de ambos os métodos.

Este trabalho visa estudar os algoritmos iterativos para MSA, com foco em algoritmos genéticos, buscando melhorias em relação às soluções existentes no momento.

O restante deste Capítulo descreve os principais conceitos associados à pesquisa realizada, sendo dividido nas seguintes seções: Seção 1.1, que descreve a base biológica sobre a qual este trabalho é construído, Seção 1.2 que descreve em maiores detalhes o problema de alinhamento de sequências e MSA, Seção 1.3 que detalha as soluções existentes para esse, Seção 1.4 que apresenta os algoritmos genéticos e sua aplicação para alinhamento múltiplo de sequências e Seção 1.5 que detalha algumas ferramentas auxiliares para verificação e análise de alinhamentos múltiplos.

1.1 Conceitos Biológicos

Esta seção descreve os conceitos biológicos associados a este trabalho. A Seção 1.1.1 descreve em linhas gerais as moléculas de DNA e RNA, a Seção 1.1.2 apresenta as moléculas de proteínas, a Seção 1.1.3 resume o chamado "Dogma Central da Biologia Molecular" que correlaciona DNA, RNA e proteínas e a Seção 1.1.4 apresenta a Teoria da Seleção Natural.

1.1.1 DNA e RNA

Todo ser vivo tem a capacidade de se reproduzir, sexuada ou assexuadamente, deixando uma prole que conserva parcialmente suas características. O mecanismo responsável por tal fenômeno baseia-se, principalmente, em moléculas capazes de autorreplicação existentes no interior das células que os compõe, chamadas DNA (do inglês *Deoxyribonucleic acid* - Ácido desoxirribonucléico) e RNA (do inglês *Ribonucleic acid* - Ácido ribonucléico) [19]. A estrutura destas moléculas foi descoberta em 1953 por Watson e Crick [127]. Tais moléculas carregam a informação que permite a replicação de características de um ser vivo para seus descendentes.

Estas moléculas são compostas por cadeias de moléculas mais simples, chamadas nucleotídeos. Estas tem em suas estrutura uma pentose, uma base nitrogenada e um grupo fosfato. A pentose é um monossacarídeo (a mais simples forma de açúcar) composta por cinco átomos de carbono. A base nitrogenada é um composto orgânico que possui ao menos um átomo de nitrogênio em sua composição e possui as propriedades químicas de uma base. Uma representação da molécula de um nucleotídeo pode ser vista na Figura 1.1.

A estrutura de um nucleotídeo permite que ele se conecte a dois outros nucleotídeos: a posição 3' pode se conectar a um grupo fosfato (ligação fosfodiéster) que, por sua vez, conecta-se posição 5' de outra pentose, permitindo assim seu encadeamento. Tais conexões permitem a geração de longas cadeias destas moléculas, gerando moléculas muito mais complexas (polinucleotídeos) tais quais as moléculas de RNA e DNA.

As moléculas de RNA são compostas por nucleotídeos que possuem sempre a mesma pentose em sua estrutura: a ribose, que possui um átomo de oxigênio ligado a sua posição 2'. Já as moléculas de DNA são compostas por outra pentose, a 2'-desoxirribose, que não possui um oxigênio ligado a sua posição 2' e sim apenas um átomo de hidrogênio. Sendo assim, a pentose presente nos nucleotídeos é o que diferencia as moléculas de DNA e RNA.

Como para um mesmo tipo de molécula tanto a pentose como o grupo fosfato são idênticos, são as bases nitrogenadas que diferenciam os nucleotídeos que formam o DNA e RNA, permitindo a estas moléculas codificarem informações. No caso do DNA existem quatro tipos de nucleotídeos distintos: as purinas Adenina (representada pela letra A) e a Guanina (G), e as pirimidinas Timina (T) e Citosina (C). Já no RNA, a Uracila (U)

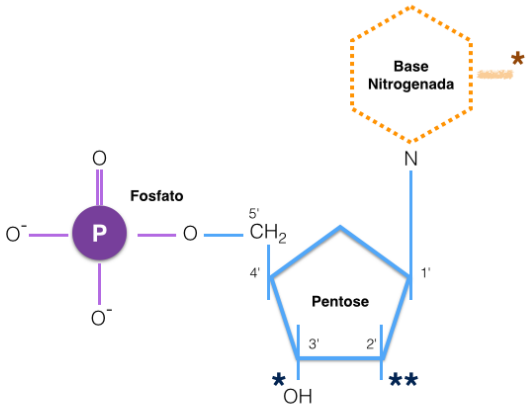


Figura 1.1: Representação de uma molécula de um nucleotídeo. O símbolo * indica onde esta molécula pode se conectar a outros nucleotídeos. O símbolo ** indica a posição 2' da pentose: no caso da ribose, esta posição está ligada a um átomo de oxigênio, já na desoxirribose existe a perda deste átomo (de onde se origina seu nome) deixando uma ligação a um átomo de hidrogênio.

assume o lugar da Timina.

Além das conexões fosfodiéster, uma base nitrogenada pode se conectar a outra base complementar através de pontes de hidrogênio. Estas conexões ocorrem respeitando afinidades quimicas e elétricas entre os nucleotídeos, de forma que uma molécula de Adenina consegue conectar-se apenas moléculas de Timina (ou Uracila) enquanto uma molécula de Citosina consegue conectar-se apenas a moléculas de Guanina, ou vice-versa.

Estas conexões na molécula de DNA fazem com que ela tenha uma estrutura de dupla hélice, com uma delas iniciada em um nucleotídeo com a extremidade 3' livre e outra com a extremidade 5', onde cada nucleotídeo em uma das hélices se conecta a um nucleotídeo complementar na outra. Isto permite que através de processos químicos no interior da célula esta dupla hélice seja dividida em duas e que uma sequência complementar seja conectada a cada uma das hélices originais, duplicando a molécula original e a informação que ela continha. A Figura 1.2 ilustra a estrutura da molécula de DNA.

As moléculas de RNA tem grandes similaridades com as moléculas de DNA, embora sejam mais frágeis e normalmente estruturadas como uma fita simples (embora também existam fitas duplas de RNA). Como já mencionado, ela é formada por nucleotídeos que diferem dos que formam o DNA pelas pentoses que as compõem (ribose) e pela presença da base nitrogenada Uracila (U) ocupando as funções da Timina (T) existente no DNA.

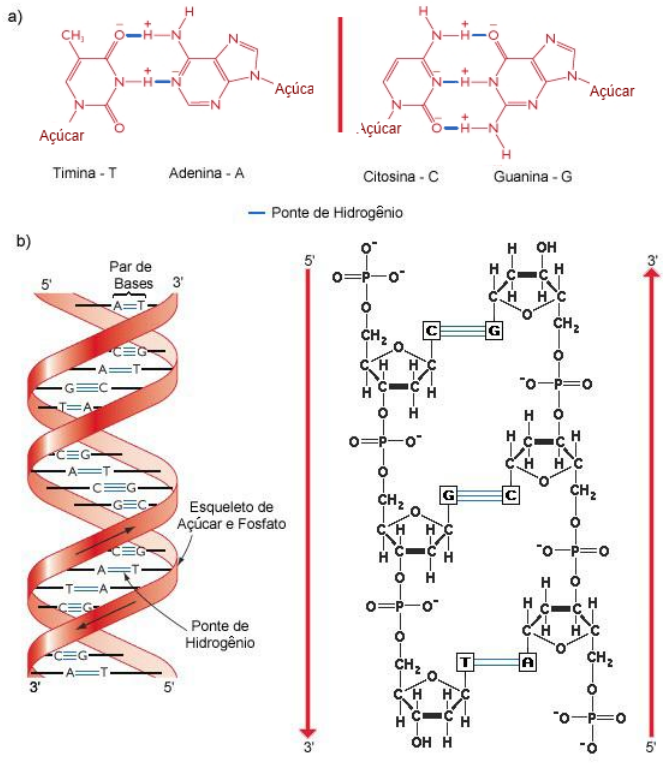


Figura 1.2: Representação da molécula de DNA: a) Bases que compõem o DNA e ligações de bases complementares por pontes de hidrogênio. b) Estrutura de dupla hélice do DNA com terminações 5' (fosfato livre) e 3' (hidroxil livre). **Fonte:** Souza [108] traduzido de Brown [19].

1.1.2 Proteínas

Proteínas [17] são os blocos fundamentais que formam todos os seres vivos, sendo responsáveis por diversas funções biológicas. Estas moléculas são estruturas complexas, formadas por unidades mais simples chamadas aminoácidos.

Os aminoácidos são moléculas orgânicas, formadas por um átomo de carbono central (C_α), conectado a um átomo de hidrogênio, a um grupo amina (NH_2), a um grupo carboxila ($COOH$) e a uma cadeia lateral. Esta cadeia é o que diferencia os diversos aminoácidos existentes. Devido a sua estrutura, as moléculas de aminoácidos, com exceção da mais simples, a glicina, podem ter dois isômeros (moléculas com mesma fórmula molecular, porém com estrutura molecular distinta), com os grupos carboxila, a cadeia lateral e o grupo amina organizados de forma dextrógira (sentido horário em relação ao carbono central, CO-R-N – onde CO denota o grupo carboxila, R a cadeia lateral e N o grupo amina) ou levógira (sentido anti-horário, N-R-CO). Na natureza, apenas apenas a fórmula dextrógira (horária) é encontrada [17]. A Tabela 1.1 apresenta os vinte

aminoácidos comumente encontrados nas proteínas.

Tabela 1.1: Os vinte aminoácidos comumente encontrados nas proteínas

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

Os aminoácidos se ligam através de uma reação onde ocorre a condensação do grupo carboxila de um deles com o hidrogênio de outro, liberando uma molécula de água (H_2O) e formando uma ligação entre os resíduos dos aminoácidos originais, chamada ligação peptídica. É ainda possível que as extremidades amino-terminal e carboxi-terminal deste novo composto sofram o mesmo processo, permitindo a criação de cadeias protéicas mais longas. Uma visão esquemática dos aminoácidos e ligações peptídicas pode ser vista na Figura 1.3.

Devido as características físico-químicas diversas dos aminoácidos, várias interações ocorrem entre os resíduos de uma molécula de proteína, gerando forças de repulsão ou atração entre eles. Além disto, a ligação peptídica gera uma conexão rígida, deixando como grau de liberdade a estes resíduos apenas rotacionar ao redor do carbono central (C_α). Tais interações e restrições fazem com que as moléculas de proteína tenham estruturas tridimensionais muito complexas e distintas, sendo estas estruturas que definem suas funções bio-químicas.

A estrutura de uma proteína é classificada em diversos níveis [17]:

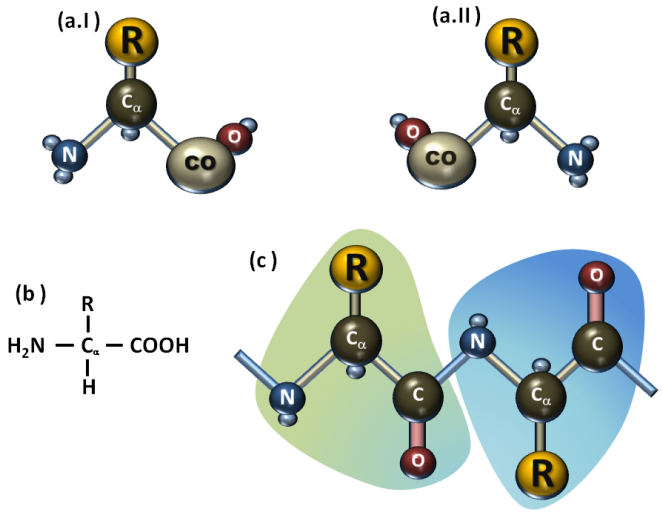


Figura 1.3: Representação de um aminoácido e de uma ligação peptídica: a) Representação de uma molécula de aminoácido em suas duas formas quirais: levógira (I) e destrógira (II). b) Fórmula química da molécula de aminoácido. c) Representação de ligações pepetídicas (linhas azuis) e dos resíduos de aminoácidos (regiões sombreadas) em uma molécula de proteína. Figuras baseadas no livro de Branden e Tooze [17].

- **Estrutura Primária** - É a sequência de resíduos de uma proteína vista de forma linear a partir de sua extremidade amino-terminal (N) até a extremidade carboxi-terminal (C). Como, de forma final, toda a conformação tridimensional da proteína depende desta sequência, todos os outros níveis de organização mais complexos derivam diretamente deste.

- **Estrutura Secundária** - É o primeiro nível de organização estrutural tridimensional de uma proteína, resultante da interação entre resíduos próximos. As principais estruturas que formam a estrutura secundária são as α-hélices, onde os resíduos se estruturam na forma de uma espiral e as β-folhas, onde estes se organizam como folhas paralelas ou anti-paralelas.

- **Estrutura Terciária** - Algumas sequências de estruturas secundárias são comumente encontradas em proteínas, sendo denominadas estruturas superssecundárias ou *motifs* [96]. Quando estes *motifs* se organizam de forma local, compacta e semi-independente temos estruturas globulares chamadas domínios. Estas estruturas são os elementos básicos que definem a estrutura terciária de uma proteína.

- **Estrutura Quaternária** - Algumas proteínas são formadas por apenas uma cadeia polipeptídica (monoméricas), porém várias são formadas por uma cadeia replicada

ou mesmo por diversas cadeias, deixando estas como sub-unidades da cadeia mais complexa. A identificação destas sub-unidades gera a estrutura quaternária de uma proteína.

Os níveis de estruturas de proteínas podem ser vistos na Figura 1.4.

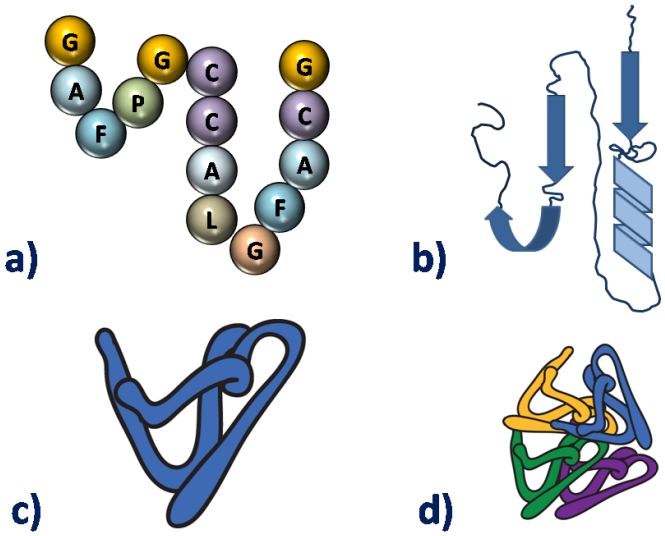


Figura 1.4: Estrutura de uma proteína: a) Estrutura Primária b) Estrutura Secundária c) Estrutura Terciária d) Estrutura Quaternária. Ilustração baseada em Branden e Tooze [17]

1.1.3 O Dogma Central da Biologia Molecular

As moléculas de DNA e RNA estão diretamente relacionadas a síntese de proteínas e possuem uma forte correlação. O processo que correlaciona estas moléculas foi inicialmente descrito por Francis Crick em 1958 [23] e posteriormente apresentado em um artigo publicado na revista *Nature* em 1970 [24] e é conhecido como "Dogma Central da Biologia Molecular".

O Dogma pode ser visto como as regras que definem os fluxos válidos de informação contidos em determinado polímero (seja este um DNA, um RNA ou uma proteína) para outro. Devido a vasta observação experimental e base teórica das possíveis transferências de informação, estas são classificadas em três categorias:

I (a) DNA $\rightarrow$ DNA

(b) DNA $\rightarrow$ RNA

(c) RNA → Proteína

(d) RNA → RNA

II (a) RNA → DNA

(b) DNA → Proteína

III (a) Proteína → Proteína

(b) Proteína → RNA

(c) Proteína → DNA

É opinião geral que os itens da classe III não existem [24], o que pode ser postulado como "uma vez que a informação chega a proteína ela não pode sair". Os itens da classe I são considerados situações comuns enquanto os da classe II possivelmente podem ocorrer, visto que não existe nenhum forte impedimento estrutural. A síntese destas regras pode ser vista de forma gráfica na Figura 1.5.

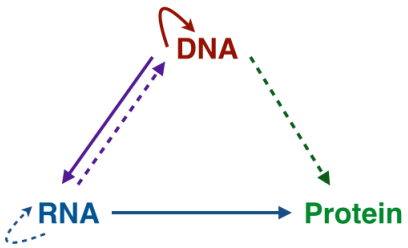


Figura 1.5: Dogma central da biologia molecular: o diagrama apresenta as possíveis transferências de informação entre os diversos polímeros. As setas cheias indicam processos gerais de transferência e as tracejadas processos especiais. Ilustração baseada em Crick [24].

O processo de troca de informação do DNA para o RNA é conhecido como transcrição e o deste para a proteína como tradução.

Através desses processos a informação genética contida no DNA é transportada através da célula por um tipo especial de RNA (o chamado RNA mensageiro - RNA_m) até os ribossomos, estruturas celulares capazes de, em conjunto com as moléculas de RNA transportador (RNA_t), sintetizar proteínas a partir da informação contida no RNA_m. Cada conjunto de três bases do RNA_m, chamado códon, é traduzido neste processo para um aminoácido, de forma que existe uma relação entre a informação contida no DNA com

a informação transportada pelo RNA_m e finalmente a proteína produzida com esta informação nos ribossomos. A relação entre os códons e os aminoácidos produzidos pode ser vista na Tabela 1.2.

Tabela 1.2: Tabela de tradução Códon X Aminoácido

	U		C		A		G		
U	UUU	Phe	UCU	Ser	UAU	Tyr	UGU	Cys	U
	UUC		UCC		UAC		UGC		C
	UUA	Leu	UCA		UAA	STOP	UGA	STOP	A
	UUG		UCG		UAG		UGG	Trp	G
C	CUU	Leu	CCU	Pro	CAU	His	CGU	Arg	U
	CUC		CCC		CAC		CGC		C
	CUA		CCA		CAA	Gln	CGA		A
	CUG		CCG		CAG		CGG		G
A	AUU	Ile	ACU	Thr	AAU	Asn	AGU	Ser	U
	AUC		ACC		AAC		AGC		C
	AUA		ACA		AAA	Lys	AGA	Arg	A
	AUG	Met/START	ACG		AAG		AGG		G
G	GUU	Val	GCU	Ala	GAU	Asp	GGU	Gly	U
	GUC		GCC		GAC		GGC		C
	GUA		GCA		GAA	Glu	GGA		A
	GUG		GCG		GAG		GGG		G

As proteínas estão associadas a muitas funções bioquímicas e podem ser consideradas os blocos fundamentais de todo ser vivo. Como postulado pelo Dogma, estas são diretamente sintetizadas a partir da informação contida no DNA, que também apresenta o seu processo de replicação. Desta forma, podemos ver que a informação contida no DNA é a responsável direta pela manutenção e expansão da vida, dando a dimensão de sua importância.

1.1.4 Seleção Natural

A Seleção Natural é uma teoria que propõe um modelo pelo qual as espécies se adaptam ao seu meio ambiente e evoluem devido as pressões impostas por este e pelas mudanças que nele ocorrem. Esta teoria foi proposta pelos naturalistas ingleses Charles Darwin e Alfred Russel Wallace em 1844, após ambos terem desenhado esta teoria de forma quase independente. O trabalho inicial neste campo foi o livro "A Origem das Espécies por meio da Seleção Natural" [25].

Podemos dizer que o que é chamado usualmente de "teoria Darwiniana da evolução" é formada, na verdade, por um conjunto de cinco teorias [38]:

1. **"*Evolution as such*"** - Linhagens de organismos se modificam, de forma natural, ao longo do tempo.

2. **Ancestral comum** - Seres vivos divergem de ancentrais comuns, de forma que é possível retroceder até uma espécie original. Desta forma, todas as espécies podem ser correlacionadas em uma grande árvore filogenética.

3. **Gradualismo** - Mesmo as diferenças mais radicais entre duas formas de vida surgiram de mudanças pequenas que gradualmente vão diferenciando as espécies.

4. **Mudança populacional** - A evolução ocorre em decorrência de mudanças de proporção de dada característica em uma certa população. Estas características são hereditárias, de forma que podem ser transmitidas à descendência de um indivíduo que a possui. Esta foi uma tese original quando de sua apresentação por contrastar com a idéia de mudanças abruptas ou associadas a indivíduos, como se acreditava então.

5. **Seleção natural** - Quando uma dada característica é favorável a um indivíduo em dado ambiente, esta aumenta a probabilidade deste indivíduo sobreviver e reproduzir-se, passando esta característica a sua prole e aumentando a proporção desta na população. De mesma forma, uma característica desfavorável dificulta a sobrevivência e reprodução do indivíduo, dificultando a passagem desta a uma próxima geração. Dado um período suficientemente grande de tempo, tais fatores podem levar a dominância ou desaparecimento de uma característica. Reside nesta tese o ponto revolucionário da teoria vislumbrada por Darwin e Wallace.

A teoria da seleção natural apresentou a forma como informações hereditárias moldam a evolução da vida. Trabalhos posteriores, conhecidos como Síntese Evolucionária [38], conseguiram unir esta teoria às descobertas dos processos bioquímicos associados à genética, de forma que foram a fundamantação teórica da biologia evolucionária moderna.

A seleção natural pode ser vista como um dos trabalhos científicos com maior impacto na história moderna. Em parte isto deve-se à maneira, ao mesmo tempo revolucionária e simples, como foi proposta. Porém, parte de seu impacto deve-se também às profundas implicações filosóficas e religiosas associadas a ela, sendo que até hoje é alvo de controvérsias dentro e fora do meio científico.

1.2 Alinhamento Múltiplo de Sequências - MSA

Alinhamento múltiplo de sequências é uma forma de correlacionar sequências de caracteres visando explicitar suas similaridades. Isto é de particular interesse para o campo da

genética, por permitir modelar os eventos evolutivos que causaram a divergência entre sequências moleculares, sejam de nucleotídeos ou aminoácidos, com origem comum.

Esta seção está subdividida na Seção 1.2.1 que apresenta o problema de alinhamento de pares de sequência e sua solução, na Seção 1.2.2 que apresenta o problema de alinhamento múltiplo de sequências e, por último, na Seção 1.2.3 que descreve alguns critérios para calcular a similaridade de resíduos e mensurar a qualidade de alinhamentos múltiplos.

1.2.1 Alinhamento de Pares de Sequências

Alinhamento de Pares de Sequências (*pairwise alignment*), como o nome indica, é o alinhamento de duas sequências de caracteres de forma a maximizar suas similaridades, de acordo com algum critério de comparação dado.

Podemos definir este problema como: dadas uma sequência α, com tamanho n, formada pelos caracteres $\alpha 1, \alpha_2, \ldots, \alpha_n$, e outra β de tamanho m, formada pelos caracteres $\beta_1, \beta_2, \ldots, \beta_m$, e uma função $S(a, b)$ de similaridade entre dois caracteres. Devemos encontrar duas sequências α' e β', de mesmo tamanho t e que diferem de α e β apenas pela inserção de caracteres especiais "-"(*gaps*), que maximizem o valor da Equação 1.1. Vale ressaltar que para uma mesma posição de α' e β' no máximo uma das sequências pode conter um *gap*. Isto permite que ocorra o alinhamento, em cada posição, de caracteres iguais (*match*), caracteres distintos (*mismatch*) ou entre um caracter de uma sequência e um *gap*.

$$score(\alpha', \beta') = \sum_{i=1}^{t} S(\alpha'_i, \beta'_i) \tag{1.1}$$

É possível utilizar este problema para modelar a similaridade entre sequências de DNA, RNA ou proteínas através do alinhamento de sua sequência de resíduos (nucleotídeos ou aminoácidos) em busca de sua ancestralidade comum [30].

Como sequências homólogas derivam ao longo do tempo, a maneira como ocorre o alinhamento de dados resíduos pode ajudar a entender como foi o processo evolucionário: um *match* indica a manutenção destes resíduos ao longo do processo, um *mismatch* indica que uma mutação causou a diferenciação destes na sequência. Podem ocorrer também alinhamentos de trechos de uma sequência com *gaps*, indicando que este trecho desapareceu da outra sequência ou foi inserido nesta ao longo do processo evolutivo. Tais trechos são conhecidos como *indels* (inserções ou remoções).

Para que esta modelagem seja precisa, a definição da função de similaridade deve levar a um alinhamento que reflita da melhor forma possível o cenário evolucionário, considerando os *matches*, *mismatches* e *indels* que ele apresenta.

Needleman e Wunsch [78] propuseram uma solução exata para este problema utilizando

programação dinâmica, com complexidade $\Theta(mn)$, onde m e n representam o tamanho das sequências alinhadas.

Este algoritmo utiliza uma pontuação para indicar a similaridade entre as duas sequências e maximizá-la. Esta pontuação é dada por uma função $\sigma(a, b)$ que quantifica a similaridade entre dois caracteres a e b. No caso do alinhamento de um caracter de uma das sequências com um *gap*, a pontuação é dada pelo valor *gap*. O pseudocódigo deste algoritmo pode ser visto no Algoritmo 1. Como este algoritmo considera o alinhamento da totalidade das sequências, seu resultado é conhecido como alinhamento global.

Algoritmo 1: Algoritmo de Needleman e Wunsch

Entrada: Sequências α e β com tamanhos m e n, respectivamente
$\sigma(a, b)$ - pontuação para alinhar os caracteres a e b
gap - pontuação para alinhamento de um caracter e um espaço
Saída: Matriz de pontuação para o alinhamento caracter a caracter de α e β

for *all* i *in* $[0..m]$ **do** $M[i, 0] \leftarrow gap * i$;
for *all* j *in* $[0..m]$ **do** $M[0, j] \leftarrow gap * i$;
for *all* i *in* $[1..n]$ **do**
  for *all* j *in* $[1..m]$ **do**

$$M[i, j] \leftarrow max \begin{cases} M[i-1, j-1] + \sigma(\alpha[j], \beta[i]) \\ M[i, j-1] + gap \\ M[i-1, j] + gap \end{cases}$$

  end
end
return M

Cada posição $M[i, j]$ da matriz retornada por este algoritmo guarda o valor máximo, de acordo com a função de similaridade, entre os prefixos $\alpha_1 \dots \alpha_i$ e $\beta_1 \dots \beta_j$. Baseado nesta informação é possível, à partir da última posição $M[n, m]$, reconstruir o alinhamento global ótimo.

Considerando o valor numa posição $M[i, j]$ qualquer, verifica-se para as posições adjacentes para ver como este valor foi construído: caso $M[i, j] = M[i-1, j-1]+\sigma(\alpha_i, \beta_j)$, isto indica que as posições α_i e β_j estão alinhadas; caso $M[i, j] = M[i-1, j] + gap$ a posição α_i se alinha a um *gap* na sequência β e de mesma forma $M[i-1, j] = M[i, j-1] + gap$ a posição β_j se alinha a um *gap* na sequência α. A Figura 1.6 ilustra este processo.

A partir daí sabe-se qual é a posição utilizada no passo anterior para os alinhamentos da subsequência imediatamente menor, permitindo assim repetir esta análise e reconstruir, da posição $M[n, m]$ para a posição $M[0, 0]$, o alinhamento como um todo. A Figura 1.7 apresenta um exemplo da execução deste algoritmo.

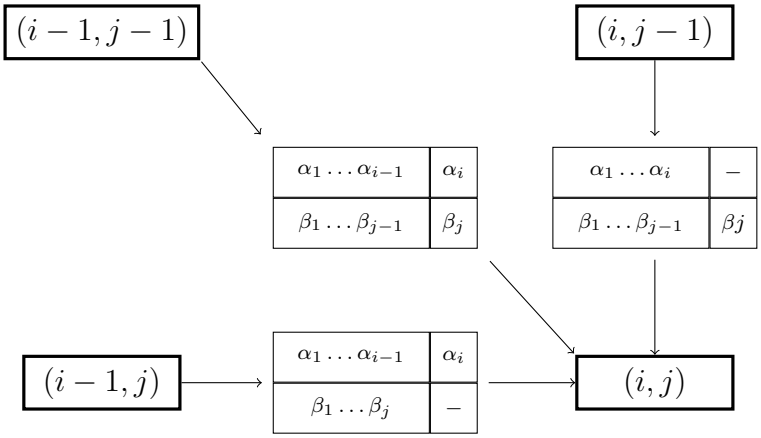


Figura 1.6: Programação dinâmica para Alinhamento de 2 sequências - A posição (i, j) possui o valor do melhor alinhamento entre os prefixos $\alpha_1 \ldots \alpha_i$ e $\beta_1 \ldots \beta_j$. Este valor pode ser obtido do *score* das 3 posições adjacentes, somado ao valor do alinhamento do último caracter de uma sequência a um *gap* ou ao alinhamento dos últimos caracteres de ambas as sequências (o que representa um *match* ou um *mismatch*).

Muitas variações deste algoritmo foram criadas, como o algoritmo semi-global, que não penaliza os gaps iniciais e finais. Tal variação é útil em casos onde as sequências divergiram devido a inserções ou remoções de resíduos em suas extremidades.

Além disto, é importante citar o trabalho de Smith e Waterman [106] que é utilizado para alinhamento local, ou seja, a busca da região mais bem conservada (com mais alta homologia) entre duas sequências.

1.2.2 Alinhamento Múltiplo de Sequências

Assim como o alinhamento de pares de sequência pode ser usado para modelar as mudanças sofridas no processo evolutivo a partir de uma sequência existente em um ancestral comum, o mesmo pode ser feito, de forma ainda mais completa se considerarmos múltiplas sequências. Ao explorar as similaridades entre múltiplas sequências com uma origem comum, a solução deste problema permite sua utilização na construção de árvores filogenéticas, indentificação de *motifs* e domínios conservados e a predição de funções de proteínas, assim como de suas estruturas secundárias e terciárias [61, 81].

Um algoritmo ótimo para extrapolar o alinhamento de pares de sequência para múltiplas sequências foi proposto em 1988 por Humberto Carrillo e David Lipman [20], sendo implementado em uma ferramenta desenvolvida por de Lipman, Altschul e Kececioglu em 1989 [69]. Este problema foi provado ser NP-Completo posteriormente [58, 125].

Como o custo computacional é proibitivo, diversas soluções aproximadas e heurísticas

	α	V	L	S	P	A	D	K
β	0	-1	-2	-3	-4	-5	-6	-7
V	-1	4	3	2	1	0	-1	-2
L	-2	3	8	7	6	5	4	3
S	-3	2	7	12	11	10	12	11
E	-4	1	6	11	11	10	9	8
G	-5	0	5	10	10	11	11	10
E	-6	-1	4	9	9	10	13	12
W	-7	-2	3	8	8	9	12	11

(a)

	α	V	L	S	P	A	D	K
β	○	←	←	←	←	←	←	←
V	↑	↖	←	←	←	←	←	←
L	↑	↑	↖	←	←	←	←	←
S	↑	↑	↑	↖	←	←	←	←
E	↑	↑	↑	↑	↖	↖	↖	←
G	↑	↑	↑	↑	↑	↖	↑	↖
E	↑	↑	↑	↑	↖	↑	↖	↖
W	↑	↑	↑	↑	↑	↑	↑	←

(b)

V	L	S	P	A	D	-	K
V	L	S	E	G	E	W	-
↖	↖	↖	↖	↖	↖	↑	←

(c)

Figura 1.7: Exemplo do uso do algoritmo de Needleman e Wunsch para alinhar um pequeno trecho de duas proteínas: α sendo o complexo haptoglobina-hemoglobina humano e a β-mioglobina do cachalote (*Physeter catodon*). Para a pontuação de *matches* e *mismatches* utilizou-se uma matriz de substituição BLOSUM62, que define a pontuação para todas as combinações de aminoácidos (maiores detalhes podem ser vistos na Seção 1.2.3). A penalização para um *gap*, no exemplo, é de -1. A figura (a) apresenta a matriz montada a partir do Algoritmo 1, a figura (b) indica, para cada célula da matriz, de qual outra célula ela se originou. Por fim, a figura (c) mostra o alinhamento reconstruído a partir desta matriz (e qual a próxima célula a ser utilizada). Vale lembrar que o alinhamento é construído da posição final para a inicial, ou seja, da direita para a esquerda.

foram criadas para buscar uma solução sub-ótima em um tempo aceitável. Algumas destas soluções são descritas na Seção 1.3.

1.2.3 Qualidade de um MSA

Um dos desafios associados ao alinhamento de sequências é como definir, de forma biologicamente significativa e acurada, a similaridade entre elas. Isto é feito, via de regra, pela definição de uma função objetivo, de forma explícita ou implícita, que quantifica esta similaridade. Esta função pode então ser utilizada em um algoritmo para encontrar um alinhamento ótimo, ou próximo ao ótimo, de acordo com o critério que ela define [27].

Uma das formas mais simples de se avaliar a similaridade em um alinhamento múltiplo é a chamada soma de pares. Podemos definir esta função de acordo com a Equação 1.2.

$$SP = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \sum_{k=i}^{m} \sigma(a_{ik}, a_{jk}) \tag{1.2}$$

Na Equação 1.2, a_{xy} representa o y-ésimo caracter da sequência x do alinhamento, que é composto por n sequências de tamanho m (contendo cada uma a sequência original e os *gaps* utilizados para gerar o alinhamento). A função $\sigma(e_1, e_2)$ fornece a similaridade entre dois caracteres $e1$ e $e2$. Esta função define os valores de similaridade para os possíveis *matches* e *mismatches* entre os caracteres, para o alinhamento de um caracter e um *gap* ou mesmo para o alinhamento *gap-gap*.

Os critérios de pontuação para *indels* (inserções e remoções) são, em geral definidos de forma empírica [124]. Alguns destes critérios consideram que tais eventos na natureza afetam trechos das sequências e não caracteres separados, penalizando um conjunto contíguo de *gaps* de forma menos rigorosa que para cada caracter, através do uso de pontuações afim ou semi-afins para estes [4]. Porém, existem críticas à efetividade desta abordagem [112] e algoritmos que tentam considerar a homologia entre *indels* das sequências alvo na função objetivo [16, 71].

Para homologias e mutações a pontuação é dada por valores para cada uma destas situações no caso de moléculas de DNA e RNA. Para proteínas, existem critérios mais refinados de pontuação, visto que existem diversos níveis de similaridade entre os possíveis aminoácidos devido as diferenças bio-químicas existentes entre eles. Esta pontuação é dada de forma geral por matrizes de substituição que valoram todas as variações possíveis entre os aminoácidos, sendo as principais as matrizes do tipo PAM [26] e BLOSUM [48].

As matrizes PAM (*Percentage (Point) of Acceptable Mutations*) foram criadas por Dayhoff *et al.* em 1978 [26]. As matrizes desta família tem o nome seguido de um número, o qual indica a taxa de troca esperada entre um par de aminoácidos dada uma "distância evolucionária". Sendo assim, a matriz PAM1 indica qual o valor esperado de troca entre dois aminoácidos quaisquer quando houver 1% de mutação nas sequências alvo, o que grosseiramente representa um período de 50 milhões de anos de evolução. Vale ressaltar que duas proteínas com distância k-PAM não necessariamente diferem em k% aminoácidos, devido a possibilidade de sobreposição de eventos de mutação em alguma posição.

As matrizes BLOSUM (*Blocks of Amino Acid Substitution Matrix*), de Henikoff e Henikoff, foram criadas em 1992 [48]. As pontuações destas matrizes são dadas em razão da taxa de troca entre aminoácidos estatisticamente encontradas em grupos de proteínas com um determinado grau de similaridade. Este grau é apresentado no nome da matriz, sendo um número que segue a palavra BLOSUM. Por exemplo, no caso da matriz BLO-

SUM62, a pontuação indica a taxa de troca esperada entre pares de aminoácidos no caso de proteínas que apresentam similaridade de 62% de sua cadeia.

No caso das matrizes PAM, matrizes com numeração menor (ou seja, para distâncias evolucionárias menores) são utilizadas para alinhamento de sequências mais próximas. No caso das matrizes BLOSUM ocorre o inverso: quanto mais alta a numeração, maior o grau de similaridade utilizado para que esta fosse calculada.

O maior problema sobre este tipo de função objetivo, para alinhamento de proteínas, está no fato de basear-se em matrizes de substituição genéricas, calculadas estatisticamente para que atendam uma gama ampla de alinhamentos. Tal fator aumenta sua flexibilidade, porém pode reduzir sua eficiência ao não evidenciar possíveis similaridades que podem ser naturais ao grupo de sequências sendo alinhado [84]. Como alternativas a estas funções foram propostos métodos baseados, por exemplo, em Modelos Ocultos de Markov (HMMs, do inglês *Hidden Markov Models*) e métodos baseados em consistência (*consitency-based schemes*).

Algoritmos que implementam HMM utilizam as sequências de entrada para gerar modelos estatísticos, que depois são utilizados como referência para o alinhamento destas. O grande revés deste método é a necessidade de um grande número de sequências para a geração de um modelo acurado.

Outra alternativa são os métodos baseados em consistência, ou em perfil, que parte do princípio que é melhor alinhar todas as sequências simultaneamente, de forma a utilizar toda a informação disponível para melhorar a qualidade do resultado. Este conceito foi inicialmente introduzido por Gotoh [44] para identificar pontos âncora que diminuissem o espaço de busca de um MSA. Uma reformulação deste conceito utilizando multiplicação de matrizes booleanas foi desenvolvido por Vingron e Argos [123].

Vários algoritmos distintos foram criados utilizando esta abordagem: a aplicação DIALIGN [75] utiliza alinhamentos locais sem gaps entre pares de sequência como base para definição de pesos e construção do alinhamento. A aplicação ProbCons [27] utiliza uma abordagem chamada consistência probabilística que combina o método de soma de pares com probabilidades obtidas no uso e treino de um HMM. São utilizados também métodos similares a soma de pares com pesos, ajustados através de algoritmos que reflitam as similaridades das sequências alvo, como o COFFEE [84] e T-COFFEE [83], entre muitas outras.

1.3 Soluções para MSA

Existem diversas heurísticas propostas para solucionar o problema de MSA. Estas heurísticas podem ser agrupadas de acordo com o tipo de abordagem, ou combinação de abordagens, que utilizam para resolver este problema. A Figura 1.8 apresenta duas taxo-

nomias de programas para MSA encontradas na literatura.

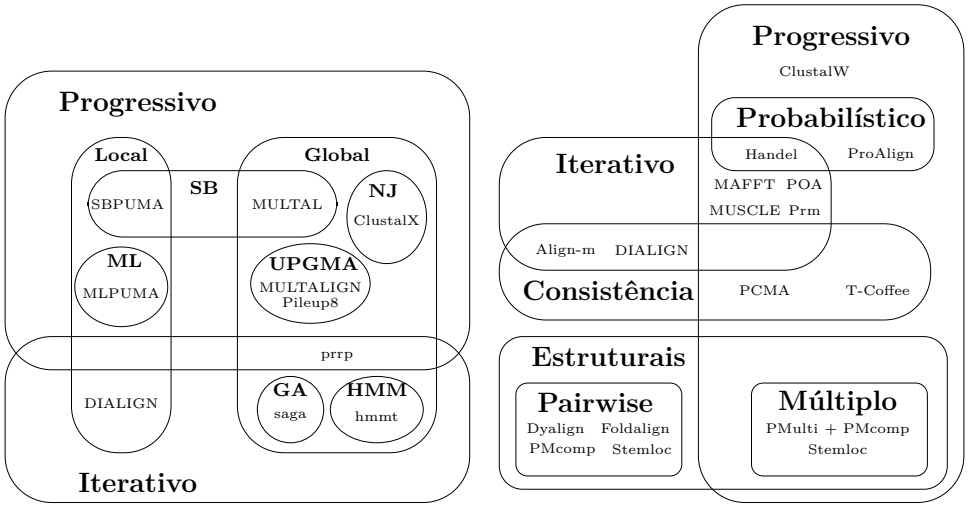


Figura 1.8: Duas taxonomias para MSA encontradas na literatura: a) Taxonomia definida por Thompson *et al.*, 1999 [120]. b) Taxonomia definida por Gardner *et al.*, 2005 [40]. As figuras são baseadas nos textos referenciados.

Esta seção descreve algumas das principais abordagens e soluções para MSA. A Seção 1.3.1 apresenta os algoritmos progressivos. A Seção 1.3.2 apresenta os algoritmos iterativos. A Seção 1.3.3 descreve outras abordagens utilizadas que não se enquadram completamente em nenhuma das duas anteriores. Várias soluções utilizam uma mescla de duas ou mais destas abordagens de forma a tentar contornar as suas limitações, a Seção 1.3.4 sintetiza algumas destas.

1.3.1 Algoritmos Progressivos

Algoritmos progressivos são heurísticas que tentam construir o alinhamento em passos, incluindo em cada um deles uma nova sequência ou sub-alinhamento previamente computado à solução. Esta abordagem foi descrita por Hogeweg e Harper em 1984 [50], sendo reformulada posteriormente por Feng e Doolittle [37].

A abordagem progressiva pode ser dividida, de modo geral, em algumas etapas principais: o alinhamento par a par das sequências alvo, a construção de uma árvore guia para orientar o processo e a integração das sequências ao alinhamento de acordo com esta árvore. A possibilidade de utilização de diversas estratégias em cada um desses passos permitiu o surgimento de diversos programas distintos baseados nesta abordagem.

Vários algoritmos progressivos utilizam o alinhamento global das sequências [78] para construir a árvore guia que será utilizada no processo. Podem ser citados nesta categoria

aplicativos como o MULTAL [114], MULTALIGN [12], PILEUP e ClustalX [116], que se diferenciam na maneira como esta é construída . A solução MULTAL utiliza as duas sequências mais próximas par a par para iniciar o processo e sequencialmente adiciona cada uma das sequências mais próximas ao alinhamento (*sequential branching*). Tanto o MULTALIGN quanto o PILEUP utilizam o algoritmo UPGMA [107] para construir a árvore, enquanto o CLUSTALX, uma versão do programa CLUSTALW [117] incluindo uma interface gráfica, utiliza o algoritmo conhecido como *Neighbour-Joining* [98].

O CLUSTALW [117] é uma das soluções mais utilizadas e conhecidas para o problema de MSA. Este programa permite o uso de programação dinâmica ou de uma heurística de aproximação rápida para efetuar os alinhamentos par a par de sequência, construindo então uma árvore guia através do algoritmo *Neighbour-Joining* (embora versões anteriores da aplicação utilizavam o UPGMA). Em seguida, a árvore gerada é utilizada para definir pesos para cada sequência, de forma a evitar que padrões existentes em sequências com alto grau de similaridade sejam super-valorizadas no processo. Estes pesos são, por fim, utilizados na construção do alinhamento final, através do consenso entre os sub-alinhamentos que vão sendo gerados para construir alinhamentos cada vez maiores. Este processo é sujeito a uma série de parametrizações para refinar o resultado como, por exemplo: uso de matrizes de pontuação distintas dependendo do estágio do algoritmo (considerando asssim as diferentes distâncias evolucionárias das sequências envolvidas) e ajustes das penalidades dadas a *gaps* (de acordo com a diferença de tamanho entre as sequências, suas similaridades, posição destes *gaps* entre outros fatores).

A solução PUMA [105] também utiliza uma abordagem progressiva, porém baseada em alinhamentos locais [106], buscando os *motifs* conservados nas sequências. Esta solução ordena então seus alinhamentos utilizando a técnica de *sequential branching* ou *maximum linkage*, motivo pelo qual é dividida em duas soluções distintas (SBPUMA e MLPUMA) na taxonomia apresentada na Figura 1.8a.

Métodos probabilísticos têm se popularizado como ferramenta para resolver problemas biológicos. A solução ProAlign [72] utiliza uma evolução do algoritmo proposto por Durbin *et al.* [29] para buscar similaridades entre sequências utilizando Modelos Ocultos de Markov (HMM) de forma a ordenar as sequências por sua similaridade para, em seguida, construir o alinhamento entre elas de forma progressiva.

Apesar de terem, em geral, uma boa performance, os métodos progressivos tem como principal problema na sua natureza gulosa. Uma decisão tomada em certo passo de sua execução não pode ser corrigida num momento subsequente (o que é conhecido como "*once a gap, always a gap*"). Sendo assim uma má decisão, em especial se tomada nas etapas inicias do processo, podem comprometer a qualidade do resultado final.

1.3.2 Algoritmos Iterativos

Uma outra classe muito importante de algoritmos para MSA são os algoritmos iterativos. Estes algoritmos refinam gradativamente a solução baseados em alguma heurística. Esta abordagem reduz consideravelmente o problema de propagação de erros que afeta os algoritmos progressivos. Por outro lado, estes algoritmos demandam em geral mais tempo e processamento que aqueles, sendo menos eficientes.

Uma solução sofisticada e com resultados muito bons é o PRRP, construído a partir de um algoritmo iterativo baseado em alinhamentos globais proposto por Gotoh [45]. Esta solução se baseia na otimização simultânea do valor do alinhamento e dos pesos utilizados. O algoritmo termina quando os pesos convergem.

Outro programa é o DIALIGN [75], que utiliza alinhamentos locais sem gaps entre pares de sequências para definição de pesos que serão utilizados para construir iterativamente o alinhamento final.

Vários programas que utilizam algoritmos iterativos são baseados em métodos probabilísticos, tais quais: arrefecimento simulado (*simulated annealing*) [62], amostragem de Gibbs (*Gibbs sampling*) [66] e modelos ocultos de Markov [31].

Alguns programas utilizam soluções inspiradas na natureza (bio-inspiradas) para resolver o problema de MSA. Estas soluções são de especial interesse devido a área de pesquisa deste trabalho.

Algoritmos bio-inspirados tentam reproduzir o comportamento de entidades biológicas para resolver problemas de otimização [131]. Por exemplo, o algoritmo AntAlign [77] utiliza um algoritmo de colônia de formigas para alinhamento múltiplo de sequências.

O algoritmo de colônia de formigas tenta simular a maneira como formigas estabelecem um caminho entre seu formigueiro e uma fonte de alimento. Para fazer isto, as formigas utilizam feromônios, marcadores químicos reconhecidos pelas outras formigas através do olfato para reforçarem bons caminhos. De forma análoga, os algoritmos baseados neste comportamento tentam marcar trechos de um caminho em um grafo para otimizá-lo.

O programa AntAlign reforça regiões bem conservadas entre sequências através de "trilhas de feromônios", informação que é utilizada para remover *gaps* mal posicionados nestes trechos.

Além do algoritmo de colônia de formigas, os algoritmos genéticos, outro tipo de algoritmo com inspiração biológica é utilizado para resolver o problema de MSA. Os algoritmos baseados nesta abordagem são detalhados na Seção 1.4 devido ao seu especial interesse neste projeto.

1.3.3 Outras Abordagens

Outra abordagem utilizada é o método baseado em consistência. Este método consiste na construção da solução através da extração de alinhamentos par a par a partir de uma biblioteca, garantindo que estes alinhamentos não sejam contraditórios ou inconsistentes entre si [40]. Dado um conjunto de n sequências, o MSA ótimo entre eles pode ser definido como o alinhamento que está de acordo com a maioria de todos os $n(n-1)/2$ alinhamentos ótimos de pares possíveis entre as sequências de entrada.

Os métodos baseados em consistência possuem algumas características que os tornam interessantes para a criação da função objetivo: primeiramente eles não se utilizam de uma matriz de substituição específica, podendo utilizar como entradas alinhamentos par a par fornecidos por quaisquer métodos ou coleções de métodos; os métodos baseados em consistência levam em conta a posição dos resíduos, considerando o contexto em que um resíduo está para procurar seu melhor alinhamento; por fim, experimentos mostram que para um conjunto de observações independentes, a maioria daquelas que são consistentes entre si, em geral, estão próximas da verdade [30].

O primeiro método de consistência foi descrito por Kececioglu em 1993 [60]. Alguns anos se passaram, porém, até que surgissem trabalhos sobre a sua otimização. Um trabalho importante desenvolvido sobre isto foi a criação da função objetivo COFFEE [84].

O COFFEE cria uma biblioteca que armazena informações sobre resíduos de todas as $\binom{n}{2}$ combinações par a par de sequências de entrada em uma base de dados, que é utilizada para mensurar os pesos das sequências e construir a pontuação do alinhamento. A função de pontuação COFFEE é dada pela Equação 1.3:

$$C = \frac{\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} W_{ij} \times S(A_{ij})\right]}{\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} W_{ij} \times L(A_{ij})\right]} \tag{1.3}$$

Onde $S(A_{ij})$ representa o número de resíduos compartilhados entre as sequências dentro do alinhamento e na biblioteca, ou seja, o nível de consistência entre ambos; e $L(A_{ij})$ o tamanho deste alinhamento. O valor W_{ij} é relativo ao peso do alinhamento par a par entre as sequências i e j.

Esta abordagem é muitas vezes utilizada em conjunto com outras abordagens, como a progressiva, no caso dos programas PCMA [93], T-COFFEE [83] e ProbCons [27], ou iterativa, como no caso do DIALIGN [75], citado na Seção 1.3.2. Os outros programas são descritos na Seção 1.3.4.

Existem também métodos que tentam incorporar ao processo de montagem do alinhamento informações adicionais sobre as estruturas tridimensionais das sequências, de forma

a contornar os problemas causados por graus muito baixos de homologia que podem afetar as abordagens descritas anteriormente. Isto é conhecido como abordagem estrutural.

Esta abordagem tende a gerar resultados melhores que outros métodos, principalmente devido a maneira como ocorre a evolução de sequências e estruturas. Estruturas tendem a evoluir mais lentamente que as sequências [68]. Desta forma, mesmo se duas sequências divergiram significativamente ao longo do processo evolutivo, sendo praticamente irreconhecíveis entre si, sua homologia muito possivelmente pode ser traçada com base na estutura tridimensional de suas moléculas [9].

Para o alinhamento de sequências de RNA o algoritmo de Sankoff [99], mescla o algoritmo de alinhamento local de Smith e Waterman [106] com algoritmos de predição de estruturas por pareamento de bases maximal (*maximal base-pairing*) [85] ou baseado em energia (*energy based*) [133]. Entre as implementações do algoritmo de Sankoff podem ser citados os programas Foldalign [43], PMComp [49] e Stemloc [53].

A ferramenta 3D-COFFEE [90] foi desenvolvida sobre a ferramenta T-COFFEE [83], porém incorporando à biblioteca desta os alinhamentos par a par baseados nas estruturas das sequências envolvidas. Estes alinhamentos podem ser entre estruturas, utilizando métodos de superposição estrutual como, por exemplo, Sap [115] e Lsqman [64] ou alinhamentos entre estruturas e sequências, o que é conhecido como *threading*, utilizando métodos como o Fugue [103]. Com estas informações o alinhamento é gerado de forma progressiva.

Uma versão posterior do 3D-COFFEE chamada Expresso [9] propõe tornar sua utilização mais simples mesmo para não especialistas. Enquanto sua versão anterior exige muita interação do usuário para definir como e quais estruturas devem ser consideradas no alinhamento, o Expresso busca *templates* estruturais adequados às sequências alvo no *Protein Data Bank* (PDB) [13] utilizando o BLAST [5]. Com esta informação a ferramenta utiliza a mesma abordagem do 3D-COFFEE: efetuar o alinhamento par a par entre estruturas ou estruturas e sequências e construir, por fim, o alinhamento progressivamente.

Outro programa baseado na abordagem estrutural é o PROMALS [91]. Esta ferramenta utiliza uma abordagem baseada em consistência probabilística (utilizando modelos ocultos de Markov - HMM) combinada com informações estruturais obtidas via predição da estutura secundária das sequências alvo e da busca de sequências homólogas em bases de dados.

Esta ferramenta inicialmente monta um pré-alinhamento, organizando as sequências de entrada através de uma árvore gerada utilizando o algoritmo UPGMA. Em seguida, as sequências são agrupadas de forma a considerar as sequências com altos graus de similaridade e uma sequência representante de cada grupo é escolhida. As ferramentas PSI-BLAST [6] e PSIPRED [57] são então utilizadas sobre estas sequências para a criação de perfis para estes agrupamentos, considerando as estruturas secundárias.

Baseado nos perfis, um HMM é construído, gerando a partir dele uma função objetivo. Os grupos são então alinhados de forma progressiva baseada nesta função. Por fim, os grupos previamente alinhados são "expandidos" novamente, de forma a gerar um alinhamento de todas as sequências alvo e os *gaps* são refinados para gerar o alinhamento final.

Este programa foi evoluido, gerando o PROMALS3D [92], que busca sequências homólogas, em relação a estrutura tridimensional, às sequencias alvo para refinar ainda mais o resultado final. São utilizadas as bases UNIREF90 [130] e SCOP40 [18] para tal, sendo a primeira em busca de um perfil e a segunda em busca de sub-sequências que representam domínios com estrutura bem conhecida.

As soluções Phylo [59] e Open-Phylo [65] usam "*crowd-sourcing*" para ajudar na solução de problema de MSA para trechos de DNA.

Estas soluções modelam o problema de MSA em um jogo onde os usuários devem alinhar blocos de cores iguais, representando resíduos idênticos ou pagar uma penalização por *gaps* e *mismatches*. Como a compreensão deste problema é simples e pode ser feita sem maiores conhecimentos dos problemas biológicos e computacionais inerentes, qualquer pessoa disposta a jogar este *puzzle* pode colaborar com a comunidade científica.

O jogo é apresentado em duas versões: o Phylo-Classic que atende os jogadores casuais e provê problemas pequenos e mais simples e o Phylo-Expert, destinado a um menor número de jogadores mais experientes, onde são desafiados com alinhamentos maiores e mais complexos.

1.3.4 Soluções com Abordagens Mistas

Muitas aplicações criadas para MSA utilizam-se de múltiplas abordagens de forma a minimizar as deficiências destas e obter um resultado com maior qualidade. Devido a isto, a categorização de um programa baseando-se apenas em uma abordagem pode ser difícil.

Podemos citar, por exemplo, o programa DIALIGN [75], que apesar de uma abordagem iterativa é considerado, de acordo com Gardner [40], também como um alinhador baseado em consistência, por utilizar alinhamentos locais sem gaps, par a par, para reforçar trechos com alta homologia entre as sequências alvo.

A solução PCMA [93] é um alinhador que utiliza um método parcialmente progressivo e parcialmente baseado em consistência. Inicialmente são alinhadas sequências com identidade acima de um dado *threshold* e agrupadas. Para melhorar o resultado de sequências pouco similares é então criada uma biblioteca que define uma função objetivo que é então utilizada para alinhar os grupos gerados na etapa anterior. Esta solução utiliza o algoritmo do ClustalW [117] para a primeira etapa e o T-Coffee para a segunda.

O T-Coffee [83] usa o método baseado em consistência, gerando uma biblioteca em duas etapas: inicialmente cria alinhamentos globais e locais entre as sequências alvo, utilizando respectivamente os algoritmos ClustalW e Lalign [55]. Esta biblioteca é estendida, criando uma matriz de substituição para cada posição específica, onde a pontuação associada a cada par de resíduos reflete sua compatibilidade em relação ao resto da biblioteca. O processo é concluído utilizando uma abordagem progressiva para alinhar as sequências alvo de acordo com as matrizes de substituição definidas anteriormente.

Outras aplicações, como o MUSCLE [33,34], utilizam prioritariamente uma abordagem progressiva, mas aplicam alguma heurística iterativa em algum ponto do processo de forma a refinar o resultado, evitando assim tanto a propagação de erros dos algoritmos progressivos quanto o tempo excessivo de se utilizar uma abordagem totalmente iterativa.

O MUSCLE utiliza um algoritmo que pode ser dividido em três etapas principais: na primeira, uma árvore guia é construída utilizando a distância K-mer, ou seja, a maior sequência contígua de caracteres em comum. Esta distância é calculada utilizando-se um alfabeto comprimido. Esta árvore é utilizada para gerar um alinhamento inicial de forma progressiva. O segundo passo gera uma nova árvore guia utilizando a distância de Kimura [63] e refina o alinhamento, também de forma progressiva. Por fim, são gerados perfis para sub-árvores, utilizando uma função LE (*log-expectation*) que serve para, iterativamente, refinar o alinhamento construído até então.

O programa ProbCons [27] adota uma estratégia com caracteristicas progressiva, iterativa e baseada em consistência.

Inicialmente o ProbCons atua baseado em consistência, porém ao invés de uma abordagem determinística define esta consistência de forma probabilística, através do treinamento de um modelo oculto de Markov (HMM). Este modelo é construído considerando-se, para cada par de sequências, a probabilidade de dois de seus resíduos estarem alinhados. Após isto, o algoritmo segue uma estratégia progressiva, montando uma árvore-guia e alinhando as sequência através de uma matriz de transformação probabilística, gerada a partir do HMM. Como etapa final existe o refinamento do alinhamento, feito de forma iterativa, onde o alinhamento gerado na etapa progressiva é particionado em dois grupos que são realinhados e recombinados.

1.4 Algoritmos Genéticos e sua aplicação em MSA

Esta seção descreve os algoritmos genéticos, foco deste trabalho, e suas aplicações na solução do problema de MSA. Essa se subdivide nas seguintes seções: Seção 1.4.1, que apresenta uma breve descrição dos algoritmos genéticos e sua implementação, Seção 1.4.2 que apresenta as primeiras soluções baseadas em algoritmos genéticos para MSA, Seção 1.4.3 que descreve as soluções que buscaram modelos simplificados para o problema ou

para a aplicação de algoritmos geneticos na sua solução e Seção 1.4.4, que apresenta soluções que mesclam algoritmos genéticos com outras abordagens ou que utilizam variações destes.

1.4.1 Algoritmos Genéticos

Algoritmos genéticos (GA - *genetic algorithms*) são algoritmos inspirados nos princípios da seleção natural, que é descrito na Seção 1.1.4.

Em linhas gerais [109] estes princípios postulam que indivíduos mais aptos na competição por recursos escassos ou limitados em seu ambiente são aqueles que sobrevivem. As características que garantem tal aptidão são controladas por unidades chamadas genes, que podem ser passadas a sua prole, aumentando a presença desta na população. De forma inversa, características negativas tendem a ter sua presença reduzida ou mesmo extinta da população.

Além disto, o processo de reprodução permite não apenas o crescimento na população de características vantajosas, mas também o aumento da diversidade, que pode por si só gerar novas características que podem alavancar a sobrevivência dos indivíduos que a possuem.

Os algoritmos genéticos são algoritmos de otimização construídos sobre uma metáfora da seleção natural. Nesses, uma população de potenciais soluções para um dado problema passa por um processo de "cruzamento", seleção e sobrevivência que, após um número de ciclos (gerações), evidencia a melhor solução ou gera uma solução melhor que as inicialmente consideradas.

O primeiro trabalho de simulação computacional do processo evolutivo foi proposto por Barricelli na década de 50 [11]. Porém, a ideia de algoritmos genéticos e sua popularização estão fortemente relacionadas ao trabalho de Holland, por volta de 20 anos depois [51].

No algoritmo proposto por Holland cada possível solução é codificada como uma cadeia de bits, imitando o código genético. A cada uma desta soluções é associado um valor de aptidão (*fitness value*) que reflete o quão boa esta solução é para resolver o problema proposto. Isto também permite compará-la a outros indivíduos (possíveis soluções) na população.

Estas soluções podem ser combinadas, duas as duas, gerando uma nova solução que apresenta características de ambas as suas soluções "pais", em um processo chamado *crossover*. Além disto, uma solução pode sofrer o processo chamado mutação, onde um ou mais dos bits que a compõe são trocados, possivelmente gerando um novo indivíduo que possui características até então inexistentes na população.

Após estes processos de diversificação, parte das soluções é selecionada, gerando uma nova população. Isto pode ser repetido de forma iterativa até que um critério de término

seja alcançado, quando a melhor solução (aquela com o maior valor de aptidão) é escolhida como solução final para o problema.

O algoritmo de Holland, conforme descrito, é conhecido como Algoritmo Genético Simples (SGA - *Simple Genetic Algorithm*) ou Canônico, sendo composto pelos seguintes elementos:

- Um mecanismo para representar as possíveis soluções como um *bitmap*;
- uma função de aptidão, que permite quantificar quão adequada é cada solução;
- um conjunto de soluções iniciais (população inicial) definida de acordo com o mecanismo de representação;
- operadores genéticos para realizar o *crossover* e mutação;
- um mecanismo de seleção para a população;
- parâmetros de controle como, por exemplo, o critério de parada.

Este processo, considerando tais elementos, é sumarizado no Algoritmo 2.

Algoritmo 2: Algoritmo Genético Simples de Holland [109]

Entrada: $Op(P)$ - operadores de *crossover* e mutação sobre a população P
$Eval(P)$ - função que calcula a aptidão $Fit(I)$ para todo indivíduo I da população P
$Sel(P)$ - função de seleção na população P
C - critério de término

$P \leftarrow InitializePopulation()$
$P \leftarrow Eval(P)$
while $\neg C$ **do**
  $P \leftarrow Sel(P)$
  $P \leftarrow P \cup Op(P)$
  $P \leftarrow Eval(P)$
end
return $\{P_{max} \in P \mid \forall P_i \in P, Fit(P_i) \leq Fit(P_{max})\}$

Os operadores e definição da função de *fitness* estão intimamente relacionados ao problema que se pretende resolver. Porém, em relação aos mecanismos de seleção (a função $Sel(P)$ no Algoritmo 2) existem alguns padrões amplamente adotados [41], independentemente do problema que a implementação pretende resolver.

Os métodos de reprodução proporcional (*proportionate reproduction*) consideram que a probabilidade de seleção de um indivíduo depende do seu valor de aptidão em relação ao total da população. Estes métodos incluem a seleção através do método de roleta (ou Monte Carlo), seleção randômica dos remanescentes (*stochastic remainder selection*) e seleção randômica universal.

Os métodos de seleção por ranqueamento (*ranking selection*) ordenam os indivíduos de acordo com seu valor de aptidão, geram cópias destes indivíduos de acordo com esta ordenação para então aplicar um mecanismo de reprodução proporcional.

Os métodos de seleção por torneio (*tournment selection*) escolhem indivíduos de forma randômica na população e dentre estes selecionam o mais apto. Apesar de usualmente os torneios serem entre pares de indivíduos, existem implementações com grupos maiores. Este processo é repetido quantas vezes seja necessário para que o número de indivíduos seja selecionado.

O algoritmo SGA utiliza um mecanismo de seleção que recria e reavalia a aptidão de seus indivíduos a cada geração. Uma variação deste mecanismo, chamado SSGA (*Steady State Genetic Algorithm*), opera sobre os indivíduos ao invés de sobre toda a população.

Em linhas gerais, um SSGA seleciona um ou dois indivíduos na população, cria uma progênie utilizando os operadores e seleciona, entre os indivíduos originalmente existentes e os recém-criados, os mais aptos. Em seguida, estes passam a ocupar as posições originalmente destinadas aos indivíduos pai. Por este motivo, são chamados também de algoritmos genéticos iterativos em contraponto aos algoritmos por geração (*generational genetic algorithm*) como o SGA. Algoritmos iterativos apresentam, segundo alguns estudos [122], melhor desempenho que os algoritmos por geração.

A primeira implementação deste tipo de algoritmo é o GENITOR, criado por Whitley [128].

1.4.2 Soluções para MSA com Algoritmos Genéticos

Algoritmos genéticos, por serem heurísticas para a solução de problemas de otimização, podem ser aplicados ao problema de MSA.

Uma das propostas pioneiras no uso de algoritmos genéticos para a solução de MSA foi feita por Tajima em 1993 [113].

A solução de Tajima monta a população inicial através da inserção randômica de *gaps* nas sequências originais, de forma que todas fiquem com o mesmo tamanho. Parte da população é então escolhida para reprodução através do método da roleta. Estes indivíduos podem ser sujeitos a um operador de *crossover* que seleciona uma coluna em comum em ambos os alinhamentos pais, divide ambos os alinhamentos usando esta coluna como pivô e os mescla baseado nos sub-alinhamentos gerados neste processo. Existe

também um operador de mutação que insere randomicamente novos *gaps* no alinhamento. Este algoritmo foi desenhado de forma a ser executado de forma paralela em diversas células de processamento, possuindo uma etapa de troca onde indivíduos hospedados em uma célula podem ser trocados com indivíduos de outra a cada geração.

Uma outra solução foi proposta por Isokawa e colegas em 1996 [56], cujos cromossomos, ou seja a representação das possíveis soluções (no caso o alinhamento), são armazenados como uma matriz binária com 0's representado elementos das sequências (no exato número de resíduos existentes em cada sequência) e 1's representando um *gap*. A população inicial é gerada randomicamente através da criação destas matrizes (respeitando o número de resíduos). Este algoritmo propõe um operador de *crossover* chamado "*window frame*" que seleciona randomicamente janelas com um certo número de resíduos (0's) em dois alinhamentos pais e faz a troca destas janelas entre eles. Um operador de mutação chamado "*island-shift*" também é aplicável, onde pequenos trechos iniciados por um *gap* e terminado por um resíduo, ou vice-versa, são selecionados ($0\ldots1$ ou $1\ldots0$) e estas extremidades são invertidas.

É do mesmo ano do trabalho de Isokawa e colegas a publicação de uma das soluções utilizando GA mais conhecidas, o programa SAGA, proposto por Notredame e Higgins [82]. Este programa representa os cromossomos através dos próprios alinhamentos.

O SAGA possui um grande número de operadores que podem ser aplicados:

- *Single Point Crossover* - Este operador seleciona uma coluna em um dos alinhamentos, encontra os mesmos resíduos no outro alinhamento e efetua um corte em ambos usando esta referência. Como no segundo alinhamento este corte não é necessariamente vertical, como no primeiro, as lacunas criadas são preenchidas com *gaps*. Após isto o sub-alinhamento à esquerda do corte de cada um dos alinhamentos pais é combinado ao sub-alinhamento à direita do outro, gerando dois novos alinhamentos.

- *Uniform Crossover* - Como o *crossover* sobre um único ponto pode, por vezes, ser muito disruptivo, foi criado um segundo operador. Neste, as colunas idênticas em ambos os alinhamentos pai são marcados e os trechos entre elas são permutados.

- *Gap Insertion* - Este operador divide as sequências do alinhamento em dois blocos, inserindo um bloco de *gaps* de mesmo tamanho em cada um deles. Os *gaps* são colocados na mesma coluna entre sequências de um mesmo bloco. Para a seleção das sequências de cada bloco é criada uma árvore filogenética estimada que é seccionada em algum nó. Os blocos consistem das sequências (folhas) de cada um dos dois ramos da árvore gerados nesta divisão.

- *Block Shuffling* - Este conjunto de operadores seleciona um bloco de *gaps* ou resíduos e o move para a direita ou esquerda. Esta ação pode ser efetuada sobre todo o bloco de *gaps*, ou dividindo-o verticalmente (por colunas) ou horizontalmente (por sequências), movendo cada partição em uma direção diferente. Considerando que estes operadores podem ser estocásticos ou determinísticos (usando uma estratégia de *semi hill climbing*) existe um conjunto de 16 variações deste operador implementadas.

- *Block Searching* - Esta mutação seleciona randomicamente um pequeno bloco (resíduos sem *gaps*) em uma sequência do alinhamento, busca o trecho que melhor se alinha com ele nas outras sequências e tenta realinhá-las.

- *Local Optimal* - Este operador seleciona um alinhamento local e tenta rearranjar seus *gaps* em busca de uma solução ótima ou sub-ótima. Isto é feito de forma exaustiva ou mediante a aplicação de um algoritmo genético focado no alinhamento local (LAGA).

Considerando que se pode escolher uma estratégia estocástica ou de *semi hill climbing* para os operadores de inserção de gaps e de *crossover* uniforme, chega-se a um número de 22 operadores possíveis. Para decidir quais operadores serão utilizados é adotada uma estratégia de *dynamic scheduling*, onde os operadores são escolhidos de forma semi-randômica, onde a probabilidade de ser utilizado é proporcional a qualidade dos indivíduos gerados com ela nas gerações anteriores. Inicialmente todos os operadores possuem a mesma probabilidade de serem utilizados.

O SAGA foi utilizado como plataforma para testes da função objetivo COFFEE [84], uma função objetivo baseada em consistência e descrita na Seção 1.3.3. Isto evidencia uma outra utilidade de algoritmos genéticos para MSA: servir como uma plataforma para teste ou refinamento de estratégias, funções objetivo ou algoritmos de MSA.

Um pesquisa posterior, realizada do Thomsen e Boomsma [121] testou a qualidade de alinhamentos gerados pela aplicação SAGA com diversos parâmetros. Algumas observações interessantes surgiram nesta pesquisa: primeiramente, o mecanismo de *dynamic scheduling* não apresentou diferença significativa quando comparada a seleção puramente estocástica dos operadores; além disto, o uso de operadores de *crossover* não afetou de forma significativa os resultados. Isto é um indicador que o SAGA foi desenhado com uma complexidade maior que a necessária para obter os seus resultados.

1.4.3 Em Busca de Simplicidade

Soluções posteriores ao SAGA propuseram abordagens simplificadas do uso de GA, utilizando menos operadores, para a resolução do problema de MSA.

Zhang e Wong [132] desenvolveram uma solução que opera sobre blocos de colunas com alinhamento exato. Devido a isto este algoritmo é limitado ao uso apenas em conjuntos de sequências com alto grau de similaridade. Porém como os próprios autores apontam ele serve para demonstrar que para alguns cenários mesmo um algoritmo genético razoavelmente simples pode obter resultados tão bons quanto outros alinhadores.

Horng e colegas [54] usam GA para fazer um pré-alinhamento (até que os número de gerações limite seja alcançada ou que não haja melhoria no resultado) para em seguida executar um algoritmo progressivo para refinar os trechos com alinhamento não exato deixados pelo algoritmo de GA. Este algoritmo representa um indivíduo como uma matriz contendo as posições dos *gaps*.

Esse algoritmo tem um operador para *crossover*, que seleciona dois cromossomos pais, divide-os em blocos consistentes e os recombina de acordo com a sua qualidade (baseado em soma de pares). Em termos de operadores de mutação, quatro estão presentes: a concatenação de *gaps*, movimentação destes para colunas adjacentes, movimentação dos *gaps* para uma coluna apenas com *gaps* (de forma a removê-la do alinhamento e simplificá-lo) e por último um operador que move os *gaps* de um conjunto de sequências dentro de um raio ao redor de sua posição.

Tal programa obteve resultados melhores que o de Zhang e Wong, porém ainda abaixo do CLUSTALW, utilizado como referência no trabalho, em especial para um número grande de sequências ou para sequências com baixa homologia.

Gondro e Kinghorn propuseram o MSA-GA [42] em 2007. Este programa representa os cromossomos diretamente como alinhamentos, da mesma forma que o SAGA, porém com um número muito menor de operadores.

O MSA-GA possui dois operadores de *crossover*, recombinando os cromossomos pai através de cortes verticais (em relação a colunas) ou horizontais (em relação a sequências) e quatro operadores de mutação: criar, remover, estender ou reduzir um bloco de *gaps*.

Este algoritmo conseguiu resultados superiores ao CLUSTALW para vários conjuntos do BAliBASE 2 [10]. Porém, um fator que deve ser considerado é o grande número de indivíduos e gerações utilizado nos testes (1000 indivíduos e 20000 gerações, respectivamente).

Botta e Negro [15] apresentaram a solução PWMAligner (2010). Neste programa, os indivíduos são representados como uma matriz com pesos posicionais (PWM - *positional weight matrix*) contendo a probabilidade de um dado resíduo estar presente em dada posição de um perfil de MSA para as sequências de entrada.

Dado um alinhamento múltiplo, o cálculo da PWM é direto: dado um alinhamento A de comprimento n, composto de k sequências e formado por um alfabeto α de tamanho m (incluindo o caracter para *gap*), gera-se uma PWM de tamanho $n \times m$ onde uma dada posição (i, j) representa a probabilidade do caracter α_j aparecer na coluna i do

alinhamento. Este valor é dado pela Equação 1.4.

$$PWM_{ij} = \frac{\sum_{y=1}^{k} \begin{cases} 1 & \text{se } A_{iy} = \alpha_j \\ 0 & \text{caso contrário} \end{cases}}{k} \tag{1.4}$$

De forma inversa é possível, dado o conjunto de sequências e a PWM, obter o alinhamento múltiplo. Para tanto é utilizada uma variação do algoritmo de programação dinâmica de Needleman e Wunsch [78] onde cada sequência é alinhada par a par não com outra sequência, mas com as posições no alinhamento. Sendo assim, uma sequência a de tamanho m, ao ser alinhada a uma PWM de comprimento n, gera com este algoritmo uma matriz de tamanho $n \times m$ onde cada posição (i, j) representa o melhor valor do prefixo $\hat{a} = a_{1...j}$ posicionado até a coluna i do alinhamento.

A montagem desta matriz ocorre de forma similar a do algoritmo de Needleman e Wunsch [79]: para calcular a posição (i, j) consideram-se a posição $(i-1, j)$, que indica o prefixo $\hat{a} = a_{1...j}$ alinhado de forma ótima até a coluna $i-1$ com a posição i neste caso alinhada com um *gap*, e a posição $(i-1, j-1)$, que indica o prefixo $\hat{a} = a_{1...j-1}$ alinhado de forma ótima até a coluna $i-1$ e o caracter a_j posicionado na coluna j do alinhamento. A posição $(i, j-1)$ não é considerada pois seria equivalente a ignorar um caracter da sequência, não sendo um estado válido entre a sequência de entrada e a PWM.

O alinhamento da sequência é obtido percorrendo o caminho que gerou o maior valor (armazenado na posição (n, m)) e verificando de qual das duas posições utilizadas para seu cáclulo o valor veio. Repetindo este processo quantas vezes necessário, alcançar-se-á a posição $(0, 0)$, momento no qual a sequência alinhada ao PWM foi obtida de forma inversa. Repetindo este processo para todas as sequências se obtem o alinhamento múltiplo.

O PWMAligner foi testado com duas funções objetivo: soma de pares com pesos e soma de pares em relação a uma sequência *template*, que pode ser uma sequência entre as sequências de entrada, a sequência consenso ou outra sequência calculada externamente. Segundo o trabalho dos autores, este segundo método obteve melhores resultados.

Utilizando o BAliBASE versão 2, os resultados obtidos pela solução proposta foram muito bons, superando as ferramentas ClustalX e o SAGA, entre outras, em quase todos os grupos de referência. Tais testes foram feitos com população de 200 indivíduos e 500 gerações.

Nizam e colegas [80] apresentaram em 2011 uma solução baseada em SOGA (*Self-Organizing Genetic Algorithm*). Esta família de algoritmos tenta resolver o problema com pouca ou nenhuma informação do usuário no que tange vários parâmetros dos algoritmos genéticos como número de gerações, tamanho da população, modos de seleção e taxas de mutação ou *crossover*.

Este algoritmo utiliza técnicas como testar a aptidão obtida ao se utilizar inicialmente

populações grandes ou pequenas. Se a diferença não for significativa, utiliza-se uma população pequena. Quando a solução converge, a população é dobrada até um limite máximo.

Um processo similar também é realizado com a taxa dos operadores. Os operadores de mutação e *crossover* também são construídos ao longo processo de forma a evitar problemas considerados pelos autores como costumeiros em algoritmos de GA, como o número excessivo de gaps.

1.4.4 Abordagens mistas e variações

Algumas soluções encontradas utilizam abodagens que combinam GA com outros métodos, ou utilizam variações de GA. Esta seção descreve algumas delas.

Meshoul, Abdessalem e colegas [1, 73] propuseram em 2005 o algoritmo QUEAMSA (*Quantum Evolutionary Algorithm for Multiple Sequence Alignment*) para alinhamento múltiplo de sequências utilizando algoritmos evolucionários quânticos como alternativa.

Este algoritmo utiliza a representação binária clássica para as posições do alinhamento, onde um 0 indica um *gap* enquanto um 1 representa um resíduo. Porém, cada posição não é representada por um bit e sim por um qubit, uma representação quântica de um bit que possui uma superposição dos dois estados possíveis, tendo uma probabilidade de estar tanto em um quanto em outro.

O estado de um qubit é dado pela Equação 1.5. Esta equação denota a função de onda no espaço de Hilbert, onde α e β são número complexos que especificam a amplitude de probabilidades dos estados $|0\rangle$ e $|1\rangle$. Quando se realiza uma observação sobre o qubit, obtêm-se um valor único para ele, que pode ser 0, com uma probabilidade de $|\alpha|^2$ ou 1, com probabilidade $|\beta|^2$.

$$|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{1.5}$$

No caso do algoritmo QUEAMSA, a representação quântica de um qubit é dada por dois números reais a_i e b_i onde $|a_i|^2 + |b_i|^2 = 1$. Desta forma cada posição do alinhamento pode ser representado pela tupla (a_i, b_i). A matriz quântica gerada pode ser vista como a representação de todos os potenciais alinhamentos superpostos, sendo que em conjunto com um algoritmo evolucionário faz o papel de um cromossomo que pode representar toda a população.

Para se obter um alinhamento válido, é realizada uma operação de medição, quando as probabilidades superpostas são transformas em uma matriz de bits determinística, considerando as probabilidades $|a_i|^2$, $|b_i|^2$ e o número de resíduos das sequências.

Uma outra operação importante é a interferência quântica, que visa reforçar a probabilidade de um bom alinhamento encontrado em uma medição ser obtido novamente.

Isto é feito rotacionando as probabilidades de cada qubit na direção do bit desejado (0 para gap, 1 para um resíduo naquela posição) em $\pm\delta\Theta$, um ângulo experimentalmente calculado de forma a evitar uma convergência prematura, porém direcionando a busca da solução próxima ao melhor resultado encontrado até o momento.

Além destas operações, o algoritmo utiliza uma série de operadores de mutação similares aos propostos por outras soluções baseadas em algoritmos genéticos, como deslocamento ou troca de qubits.

Lee e colegas [67] propuseram em 2008 o algoritmo GA-ACO que combina algoritmos genéticos com otimização de colônia de formigas.

O algoritmo de colônia de formigas é inspirado no processo utilizado por colônias de formigas para encontrar fontes de comida, utilizando marcações com feromônios em posições chave. Este locais tendem a ser mais visitados por outras formigas. No caso dos algoritmos inspirados neste processo, as "formigas virtuais" partem de um ponto inicial e se movem em direção a vizinhos válidos. Durante este processo, a informação coletada por estas formigas é armazenada na forma de uma trilha de feromônios que é iterativamente alterada durante o processo. Ocorre também um processo chamado evaporação de feromônios, que decrementa com o tempo a sua quantidade de forma a impedir que ocorra uma convergência em torno de um máximo local.

O algoritmo genético utilizado no GA-ACO é muito similar a outras implementações, contendo, em adição, alguns operadores heurísticos para realizar um processo de *hill climbing*.

O algoritmo de otimização de colônia de formigas é aplicado após cada geração como uma busca local para melhorar a busca da solução. É realizada a identificação de blocos desalinhados no melhor alinhamento da geração atual através de uma janela deslizante e o algoritmo é aplicado sobre estas regiões de forma a melhorar seu *score*. Duas sequências com blocos desalinhados são escolhidas como alvo para a aplicação do algoritmo.

1.5 Ferramentas Auxiliares

Esta seção descreve as ferramentas associadas a MSA de interesse para o trabalho apresentado. Na Seção 1.5.1 são apresentadas ferramentas de *benchmark* para avaliação de ferramentas de alinhamento múltiplo, com especial atenção no BAliBASE e na Seção 1.5.2 os visualizadores de alinhamentos, em especial aqueles para comparação de dois alinhamentos distintos construídos a partir do mesmo conjunto de sequências como entrada.

1.5.1 Ferramentas de Benchmark

A quantidade de ferramentas para MSA, assim como a quantidade de cenários onde uma ou outra destas pode ser efetiva, criou a necessidade de elaboração de algum mecanismo sistemático e quantitativo de comparação entre elas.

Observando-se os trabalhos iniciais na área de alinhaments múltiplos de sequências vê-se a utilização de um conjunto arbitrário de alinhamentos, muitas vezes selecionado a partir de bancos de estruturas que, em geral, são organizados em famílias homólogas.

Qualquer nova ferramenta naquela época, caso fosse comparada a ferramentas prévias, teria que recorrer aos conjuntos de testes apresentados naqueles trabalhos. Tal metodologia podia levar a conclusões tendenciosas ao poder desconsiderar cenários onde uma das ferramentas (ou ambas) tinham deficiências ou méritos. Como apontado por Thompson [119], as bases de dados não forneciam informação classificada e estruturada para uma avaliação sistemática dos programas de alinhamento.

Devido a este problema surgiram conjuntos de testes e ferramentas para comparar a qualidade dos alinhamentos gerados por programas de MSA, funcionando como *benchmarks*. Entre as ferramentas que medem alinhamentos de cadeias proteicas, podem ser citadas o BAliBASE [10, 118, 119], HOMSTRAD [74] e PREFAB [34]. Para alinhamento de cadeias de RNA um exemplo é o BRAliBASE [40].

O BAliBASE, atualmente em sua terceira versão, foi o primeiro *benchmark* para alinhamento de cadeias de proteínas e é um dos mais utilizados para este fim, sendo a escolha de vários *surveys* de comparação [36, 47, 101], além de em muitos trabalhos sobre as próprias ferramentas de MSA.

As versões anteriores desta ferramenta eram baseadas em um conjunto de alinhamentos criados manualmente e bem conhecidos. Isto limitava o conjunto no tamanho e número de sequências. Para contornar estes problemas, a versão 3 foi construída sobre alinhamentos assistidos por algoritmos computacionais que foram posteriormente refinados manualmente.

O BAliBASE 3 possui 5 grupos de referência, que representam diferentes cenários aos quais um programa de MSA pode ser exposto. Uma descrição sumarizada destes grupos e sua composição pode ser vista na Tabela 2.3.

Os alinhamentos do BAliBASE são disponibilizados em dois formatos: truncados apenas com as regiões homólogas ou contendo as sequências integrais. Além disto estes alinhamentos são anotados de forma a reforçar as regiões consideradas muito confiáveis para fins de avaliação, que são chamados *core blocks*.

O BAliBASE provê, além do alinhamento "ideal", para os cenários disponibilizados, duas métricas para análise quantitativa dos alinhamentos gerados por uma ferramenta de MSA em relação a estes [14]:

Tabela 1.3: Grupos de Referência do BAliBASE 3

Referência	Descrição	Número de Alinhamentos	Número de Sequências
RV11	Sequências equidistantes com menos de 20% de identidade entre si e sem grandes inserções ($>$ 35 resíduos).	38	265
RV12	Sequências equidistantes que compartilham entre 20 e 40% de identidade entre si, sem grandes inserções.	45	411
RV20	Família de sequências que possuem mais de 40% de identidade, porém com uma sequência "orfã" com menos de 20% de identidade com qualquer outra sequência.	41	1896
RV30	Alinhamento de sub-famílias, onde as sequências de uma mesma sub-família compartilham mais de 40% de identidade entre si, mas menos de 20% de identidade com qualquer sequência de outra sub-família.	30	1882
RV40	Sequências que possuem mais de 20% de identidade entre si, porém com grandes extensões nas terminações N/C.	48	1317
RV50	Sequências que possuem mais de 20% de identidade entre si, com grandes inserções.	16	483
	Total	217	6255

- **SP *score* (*Sum of Pairs*)** - Indica o percentual de pares de resíduos alinhados corretamente no alinhamento de teste quando comparado ao mesmo par de resíduos no alinhamento referência em relação ao número total de pares de resíduos existentes no alinhamento referência. Considerando um alinhamento com N sequências e M colunas, onde a i-ésima coluna do alinhamento pode ser descrita pelos resíduos $A_{i1}, A_{i2} \ldots, A_{iN}$, podemos definir p_{ijk} como sendo o indicador de acerto em relação ao alinhamento referência. Desta forma, $p_{ijk} = 1$ se A_{ij} e A_{ik} estão alinhados no alinhamento referência e $p_{ijk} = 0$ caso contrário. Com isto o *score* S_i da i-ésima

coluna do alinhamento de teste pode ser descrita pela Equação 1.6.

$$S_i = \sum_{j=1}^{N} \sum_{k=1, j \neq k}^{N} p_{ijk} \tag{1.6}$$

Baseado nesta medida, o SP pode ser definido de acordo com a Equação 1.7, onde M_r denota o tamanho do alinhamento de referência e S_{ri} o *score* de sua i-ésima coluna.

$$SP = \frac{\sum_{i=1}^{M} S_i}{\sum_{i=1}^{M_r} S_{ri}} \tag{1.7}$$

- **TC *score* (*Total Columns*)** - Indica o percentual de colunas do alinhamento de teste perfeitamente alinhados em relaçao ao alinhamento referência. Sendo C_i o *score* da i-ésima coluna do alinhamento de teste, temos $C_i = 1$ se todos os resíduos da coluna estão alinhados no alinhamento referência e $C_i = 0$, caso contrário. Baseado nisto podemos definir TC de acordo com a Equação 1.8.

$$TC = \frac{\sum_{i=1}^{M} C_i}{M} \tag{1.8}$$

Apesar de ser um dos mais utilizados *benchmarks* para alinhamento de sequência de proteínas, o BAliBASE não é isento de críticas. Błażewicz e colegas [14] apontam que o uso destas métricas não é calculado corretamente pelo ferramental do BAliBASE quando regiões que não são *core blocks* são avaliadas.

Edgar [35] aponta que muitas proteínas contidas no repositório não possuem estrutura conhecida, muitos conjuntos não são globalmente alinháveis por possuirem proteínas com domínios distintos em ordem distinta, ou por haver algumas regiões não homólogas alinhadas entre si. Além disto, são apontados problemas de alinhamentos estruturais e definição incorreta de *core blocks*, por não representarem, de forma inequívoca segundo estes autores, uma estrutura secundária conservada entre as sequências alinhadas.

1.5.2 Ferramentas para Comparação Visual de MSAs

Uma das necessidades que sentimos neste projeto foi a de uma ferramenta visual para comparação de MSAs alternativos, de forma que fosse possível fazer uma análise fina de similaridades e diferenças entre dois alinhamentos gerados para o mesmo conjunto de sequências, em geral sendo um deles um alinhamento de referência.

Existem diversas ferramentas para visualização de alinhamentos múltiplos, porém a grande maioria tem como foco a análise de informações de diferentes naturezas sobre

um único alinhamento, sendo destinadas a uma análise mais profunda, porém intrínseca, deste. Nesta categoria de aplicações pode-se destacar o ClustalX [116] e a Jalview [21,126]. Além destas ferramentas é importante mencionar formas diferentes de visualização como a Fingerprint [70] e a ProfileGrids [97].

A ferramenta Jalview é disponibilizada tanto em versão *web* quanto *desktop* e possui uma série de funcionalidades associadas a visualização, análise e anotação de alinhamentos múltiplos de sequências. Entre as funcionalidades existem as opções de edição e marcação de trechos do alinhamento, possibilidade de cópia destes trechos, ocultar sequências ou colunas, criação de linhas de anotação assim como a visualização de informações extras sobre o alinhamento ou sequências como, por exemplo, dados estruturais.

A ferramenta Fingerprint apresenta o alinhamento como um conjunto de barras, cada um representando uma coluna, onde cores distintas podem representar valores diversos de dada informação. Entre as opções de visualização desta ferramenta estão os nucleotídeos, a sua heterogeneidade, variabilidade de resíduos e posicionamento de *gaps*.

O programa ProfileGrids utiliza uma representação matricial e por cores. As colunas da matriz representam uma sequência base, enquanto as linhas representam todos os possíveis resíduos. Desta forma, cada célula representa, considerando o montante total, a porcentagem de resíduos de dado tipo (representado pela linha) que se alinha com o resíduo da sequência base em dada coluna. Tal informação é apresentada por um sistema de cores pelo programa. Além disto, trechos onde ocorrem percentuais muito grandes de identidade são marcados, pois indicam potenciais *motifs*.

Em termos de ferramentas para comparação de alternativas de alinhamentos, visando uma análise comparativa, a variedade encontrada não é tão expressiva quanto a de programas para análise e visualização de um único alinhamento.

O AltAVist [76] é uma ferramenta simples que permite a entrada de dois alinhamentos pré-calculados ou de um conjunto de sequências, caso no qual o programa as alinha utilizando tanto o DIALIGN quanto o ClustalW. Em ambos os casos o resultado é apresentado visualmente através da sobreposição dos dois alinhamento, marcados com um sistema de cores por grupos. Grupos podem ser definidos como conjuntos maximais de resíduos que estão alinhados de forma consistente em ambos os alinhamentos.

O programa SinicView (*Sequence-aligning INnovative and Iteractive Comparison VIEWer*) [104] é voltado para a visualização e comparação de alinhamentos de sequências de nucleotídeos, permitindo a justaposição de alinhamentos gerados por diversas ferramentas.

O SinicView é formado por três elementos de interface principais: um visualizador global, uma visão detalhada e uma visualização de informações extras.

Na visão global é possível ver o percentual de identidade em relação a uma sequência base pertencente aos alinhamentos, calculando a soma de pares em relação às outras. Considerando os alinhamentos providos como entrada, esta ferramenta apresenta, utili-

zando um esquema de cores, qual deles obteve melhor performance para cada região do alinhamento. Além disto, um gráfico de linha apresenta o percentual de identidade em relação a sequência base. É possível escolher qual trecho da sequência se deseja visualizar.

A visão detalhada permite ver o percentual de identidade de cada um dos alinhamentos fornecidos independentemente. Um gráfico de linha com o percentual de identidade em relação a sequência base é apresentado no caso do trecho sendo visualizado ser muito longo (maior que 100 resíduos), caso o trecho seja mais curto, o alinhamento é apresentado na forma de colunas. Neste caso, aquelas que são idênticas entre todas as sequências do alinhamento são marcadas e apresentadas com destaque.

Além destas duas visões, a ferramenta ainda apresenta dados extras para cada um dos alinhamentos fornecidos, como *gaps* e anotações. Ferramentas extras, como gráficos de barras e circulares, fornecem informação adicional sobre o grau de identidade por coluna, de forma percentual.

O SuiteMSA [8] é um conjunto de ferramentas desenvolvido para comparação e análise de dois alinhamentos. Este conjunto é formado por um visualizador de MSA, um comparador visual entre os dois alinhamentos, um visualizador global (*pixel plot*) além de ferramentas para geração, visualização e edição de árvores filogenéticas.

O visualizador de alinhamentos apresenta não só o alinhamento como também as suas estruturas secundárias (de acordo com o PSIPRED [57]) e o grau de conservação em dada coluna.

O comparador visual de alinhamentos (*MSA Comparator*) permite a escolha de um alinhamento referência e apresenta informações relativas entre os dois alinhamentos, para um trecho selecionado, utilizando este balizador. Colunas consistentemente alinhadas em ambos são apresentadas em cor azul enquanto as inconsitentes em cor vermelha. Informações adicionais, como o soma de pares por coluna, também podem ser visualizadas.

O visualizador global (*Pixel Plot*) permite a visão geral do alinhamento, enquanto o *MSA Comparator* é voltado para a comparação de trechos deste. Cada resíduos é representado por um pixel e não por uma letra, porém mantendo o esquema de cores que permite visualizar *gaps*, regiões inconsistentes ou consistentes de todo alinhamento.

Para a geração da árvore filogenética, o SuiteMSA utiliza o iSG versão 2.1 [110, 111] (indel-Seq-Gen), que simula para uma família de proteínas os eventos que moldaram sua história evolutiva, como mutações, *indels* ou mesmo rearranjos. Após a execução desta ferramenta, o SuiteMSA permite a visualização da árvore filogenética gerada com o *Phylogeny Viewer*, indicando com cores diferentes os eventos de inserção e remoção de resíduos.

O SuiteMSA pode apresentar tanto alinhamentos quanto árvores filogenéticas fornecidas em formatos bem conhecidos, como o FASTA para alinhamentos de sequências e o Newick para filogenias.

Capítulo 2

ALGAe

Como parte deste trabalho foi implementado um ambiente para a execução de um algoritmo genético focado na geração de MSAs. Este ambiente foi desenvolvido de forma a ser parametrizável para que diversas combinações de estratégias de geração de população inicial, operadores de mutação e *crossover*, métodos de seleção e funções objetivos pudessem ser testados em conjunto, buscando uma combinação que gerasse bons resultados.

O ambiente desenvolvido foi chamado ALGAe (do inglês *ALigning by Genetic Algorithm environment*), desenvolvido utilizando a linguagem Java.

A escolha desta linguagem foi motivada por alguns fatores: por ser uma linguagem orientada a objetos, é possível desenvolver o ambiente com uma clara modularização e divisão de responsabilidades. Isto permite, além de uma manutenção mais simples e efetiva, a implementação independente de diversos algoritmos para os parâmetros que desejávamos testar, em especial com a utilização de padrões de projeto [39] como, por exemplo: *singletons*, *strategies* e *factory methods*.

Além disto, recursos da linguagem como interfaces e reflexão facilitam o acoplamento dos algoritmos para os diversos parâmetros considerados para serem testados e medidos. O trabalho de combiná-los também é simplificado, ao permitir que isto seja feita através de um arquivo de configuração, em tempo de execução, sem a necessidade de recompilar o código.

Para este trabalho, utilizou-se um algoritmo genético básico, onde diversos parâmetros podem ser configurados, conforme descritos no Algoritmo 3. Mais informações sobre algoritmos genéticos podem ser vistos na Seção 1.4.

Neste algoritmo um conjunto de sequências a serem alinhadas S é utilizado como entrada. Além disto o tamanho da população a ser utilizada a cada geração e o número de gerações podem ser configurados.

Para cada um dos outros passos do algoritmo, podem ser utilizadas estratégias diferentes de execução. Sendo assim, a estratégia de criação da população inicial, o método de

Algoritmo 3: Algoritmo Genético utilizado no ALGAe

Entrada: S (conjunto de sequências), $MaxPopulationSize$, $MaxGeneration$
Saída: Melhor alinhamento após $MaxGeneration$ gerações
$population \leftarrow initialPopulation(S, MaxPopulationSize)$
for $generation \leftarrow 1$ **to** $MaxGeneration$ **do**
  $breedingPopulation \leftarrow selectForBreeding(population)$
  $population \leftarrow population \cup breedingPopulation$
  $population \leftarrow select(population, MaxPopulationSize)$
end
return $bestAlignment(population)$

seleção para reprodução, os operadores de reprodução (*crossover* e mutação), o método de seleção da população contendo os novos indivíduos e a função objetivo que permeia todos estes passos podem ser escolhidos através de um arquivo de configuração. Para que isto se sustente, cada um destes elementos deve apenas implementar uma interface condizente com sua função, o que permite mantê-los desacoplados do corpo principal do algoritmo.

As próximas seções descrevem em maiores detalhes os elementos do ALGAe: A Seção 2.1 descreve as estratégias de geração de população inicial, a Seção 2.2 os operadores de mutação, a Seção 2.3 os operadores de *crossover*, a Seção 2.4 os mecanismos de seleção, a Seção 2.5 apresenta as funções de *fitness*, a Seção 2.6 os mecanismos de configuração e *log* e a Seção 2.7 os primeiros resultados obtidos com a ferramenta.

2.1 População inicial

A criação da população inicial é definida no ALGAe por classes que implementam a interface *InitialPopulation*. Esta interface possui apenas um método: *generate*(), que retorna uma lista de alinhamentos que formam a população inicial.

Foram implementados, em um primeiro momento, 4 algoritmos para a geração de população inicial descritos nas próximas seções.

2.1.1 BasicStarAlignmentPopulation

Este algoritmo cria todos os indivíduos da população inicial utilizando uma variação do algoritmo de alinhamento estrela para MSA [46, 102].

O algoritmo utiliza um *pool* com todas as sequências a serem alinhadas, de onde seleciona inicialmente um par aleatoriamente. De forma arbitrária, a primeira sequência é escolhida como âncora para o resto do processo de alinhamento. As duas sequências

são alinhadas par a par (utilizando o algoritmo de alinhamento global ou semi-global, selecionado também de forma randômica). Após isto as sequências restantes vão sendo sorteadas e removidas do *pool*, alinhadas a sequência âncora e incluídas no alinhamento múltiplo gerado até este ponto.

O alinhamento é feito respeitando o princípio *"once a gap, always a gap"*, ou seja, um gap gerado em um alinhamento par a par prévio é replicado na nova sequência, assim como um gap que teve que ser incluído na sequência âncora durante o alinhamento par a par é replicado para as outras sequências no alinhamento múltiplo parcial. Um exemplo ilustrativo deste algoritmo pode ser visto na Figura 2.1.

Este processo é repetido até que sejam gerados indivíduos suficientes para criar a população inicial.

2.1.2 RandomTreeAlignmentPopulation

Este algoritmo é similar ao algoritmo *BasicStarAlignmentPopulation* apresentado da Seção 2.1.1, por utilizar uma abordagem de seleção aleatória de sequências e alinhamento estrela. A diferença reside no fato de que a sequência âncora não é definida *a priori* e sim a cada passo.

O algoritmo começa de forma similar ao outro, ao selecionar um par de sequências e alinhá-las. A partir dai, uma nova sequência é selecionada do *pool*, porém, uma sequência aleatória é selecionada entre as já alinhadas para servir como âncora. Este processo se repete até que o *pool* esteja vazio e todas as sequências alinhadas.

Como neste processo existe a possibilidade de escolhas ruins para âncoras, isto pode causar o surgimento de indivíduos que representam alinhamentos ruins. De forma a evitar que isto contamine o processo, o algoritmo gera uma população inicial com o dobro do tamanho requisitado, deixando para a primeira etapa de seleção eliminar os indivíduos em excesso.

Este método permite a geração de um número maior de indivíduos do que o possível com *BasicStarAlignmentPopulation*, pois ao termos uma âncora variável o posicionamento dos gaps pode ser significantemente alterado.

2.1.3 SuperPopulation

Este método gera uma superpopulação inicial ao unir a população gerada pelos algoritmos *BasicStarAlignmentPopulation* e *RandomTreeAlignmentPopulation*, descritas nas Seções 2.1.1 e 2.1.2. Assim como neste segundo algoritmo, a *SuperPopulation* gera uma população maior do que a requisitada, fazendo que um grande número de indivíduos sejam "extintos" na primeira seleção.

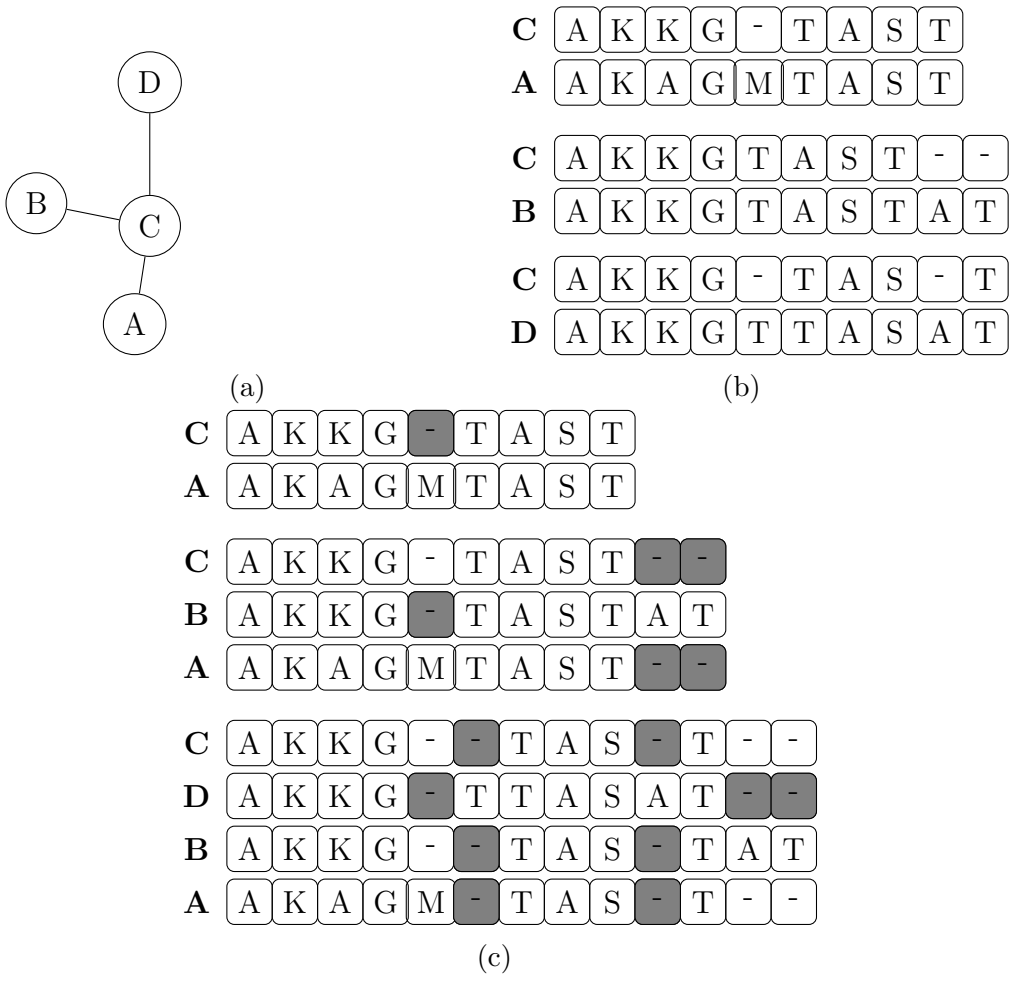


Figura 2.1: Exemplo hipotético de um alinhamento estrela. A Figura (a) apresenta de forma visual as distâncias entre as sequências a serem alinhadas, tendo como âncora a sequência C. Os alinhamentos par a par destas sequências pode ser visto na Figura (b). O alinhamento estrela, considerando a sequência C como âncora, é montado, passo a passo, na Figura (c). Os *gaps* sombreados indicam onde estão sendo inseridos de forma a manter a premissa de *"once a gap, always a gap"*.

2.1.4 PopulationCR

Este método adiciona à população gerada pelo algoritmo *SuperPopulation* descrito na Seção 2.1.3 um indivíduo extra, criado a partir de um algoritmo progressivo baseado em blocos que chamamos de CRAligner (*Conserved Region Aligner*).

Para o desenvolvimento deste algoritmo foram testadas algumas formas de se utilizar o alinhamento de blocos em pares de sequências e depois construir o alinhamento múltiplo de forma progressiva.

A primeira abordagem (A1) consiste em buscar, para cada par de sequências, o melhor alinhamento local através do algoritmo de Smith e Waterman [106]. Após este passo, o processo é repetido de forma recursiva para os trechos que precedem e sucedem o trecho alinhado, até que o tamanho do alinhamento local máximo encontrado seja inferior a um tamanho de corte k dado (no caso utilizamos $k = 3$). Por fim resíduos restantes entre alinhamentos locais são alinhados utilizando o algoritmo de alinhamento global de Needleman e Wunsch [78] e aqueles que antecedem o primeiro alinhamento local ou sucedem o último são alinhados utilizando uma variação semi-global deste algoritmo, que não penaliza gaps no início das sequências para o primeiro trecho e que não penaliza gaps no fim destas para o último trecho.

A segunda abordagem (A2) utiliza uma janela deslizante de tamanho fixo k para encontrar blocos idênticos e contínuos (sem *gaps*) entre cada par de sequências.

Inicialmente, para cada par de sequências, são procurados blocos contínuos comuns em ambas as sequências. O Algoritmo 4 apresenta a maneira como os blocos são procurados e armazenados, onde $block(i, j, k)$ representa um bloco contínuo iniciado na posição i da primeira sequência e na posição j da segunda, com tamanho k.

Algoritmo 4: Busca de blocos comuns

Entrada: Sequências α e β com tamanhos m e n, respectivamente e k, o tamanho da janela
Saída: *BlockList* - lista de blocos comuns

```
BlockList ← ∅
for i ← 1 to m − k + 1 do
    for j ← 1 to n − k + 1 do
        if α[i ... i + (k − 1)] = β[j ... j + (k − 1)] then
            BlockList ← BlockList ∪ block(i, j, k)
        end
    end
end
return BlockList
```

Com a definição dos blocos, o algoritmo monta um grafo direcionado de blocos compatíveis. São montados dois nós, um representando o início da sequência e outro representando o final da mesma. Todos os blocos também são representados como nós neste grafo. São então geradas arestas partindo do nó inicial para todos os nós de "bloco" e arestas partindo de cada um destes até o nó final.

A seguir, os blocos compatíveis entre si são também conectados por arestas. Como compatível podemos definir blocos que podem coexistir no alinhamento das sequências, ou seja, blocos que podem estar alinhados neste sem gerar inconsistência. A Equação 2.1

propõe uma maneira formal para definir a consistência $\mathscr{C}$ entre os blocos β^1 e β^2. Nesta equação β_i^m representa a posição onde o bloco se inicia na primeira sequência do bloco m e β_j^m representa o inicio deste na segunda sequência.

$$\mathscr{C}(\beta^1,\beta^2) = \begin{cases} ((\beta_i^1 + k < \beta_i^2) \wedge (\beta_j^1 + k < \beta_j^2)) \\ \\ \vee \\ \\ ((\beta_i^1 < \beta_i^2) \wedge (\beta_i^1 + k \geq \beta_i^2)) \wedge \\ ((\beta_j^1 < \beta_j^2) \wedge (\beta_j^1 + k \geq \beta_j^2)) \wedge \\ (\beta_i^2 - \beta_i^1 = \beta_j^2 - \beta_j^1) \end{cases} \tag{2.1}$$

A Figura 2.2 ilustra este conceito de forma visual, apresentando casos de consistência e inconsistência entre blocos e como o grafo de consistência é construído.

O algoritmo busca, de posse do grafo com blocos consistentes, o maior caminho entre o nó inicial e o nó final. Como o grafo é orientado e acíclico o problema pode ser resolvido encontrando uma ordenação topológica do grafo.

Caso existam nós no caminho máximo que representem blocos com sobreposição, eles são mesclados. Os trechos intermediários são alinhados através de alinhamento global.

Por fim, baseado nas consistências par a par das sequências, uma árvore guia é montada e o alinhamento múltiplo construído com base nesta.

Uma variação desta abordagem (A2a) também foi elaborada consistindo em aplicar o mesmo algoritmo, porém substituindo os resíduos das sequências por um alfabeto comprimido [32]. Os alfabetos comprimidos agrupam resíduos similares sob uma mesma representação, de forma a ressaltar os pontos comuns entre resíduos similares. Foi utilizado neste algoritmo o alfabeto comprimido Dayhoff6, que agrupa os aminoácidos em seis classes definidas por Dayhoff e colegas [26].

Os três algoritmos foram testados com todos os cenários do BAliBASE 3 de forma a decidir qual a melhor opção a ser utilizada na geração da população inicial. A Tabela 2.1 apresenta os resultados obtidos e o tamanho da janela deslizante k para os casos aplicáveis.

Tabela 2.1: Resultados obtidos com os alinhadores de blocos. Os valores são associados à métrica SP (%) do BAliBASE e quanto mais próximos de 100% melhor o resultado.

Alinhador	RV11	RV12	RV20	RV30	RV40	RV50	Média
A1	30,2	67,5	73,1	58,4	58,7	58,0	57,7
A2 ($k = 5$)	20,3	60,7	66,5	49,4	49,4	44,6	48,5
A2a ($k = 12$)	26,9	67,6	73,3	50,8	50,1	51,1	53,3

Como pode ser verificado, o algoritmo A1 (alinhamento local recursivo) superou a

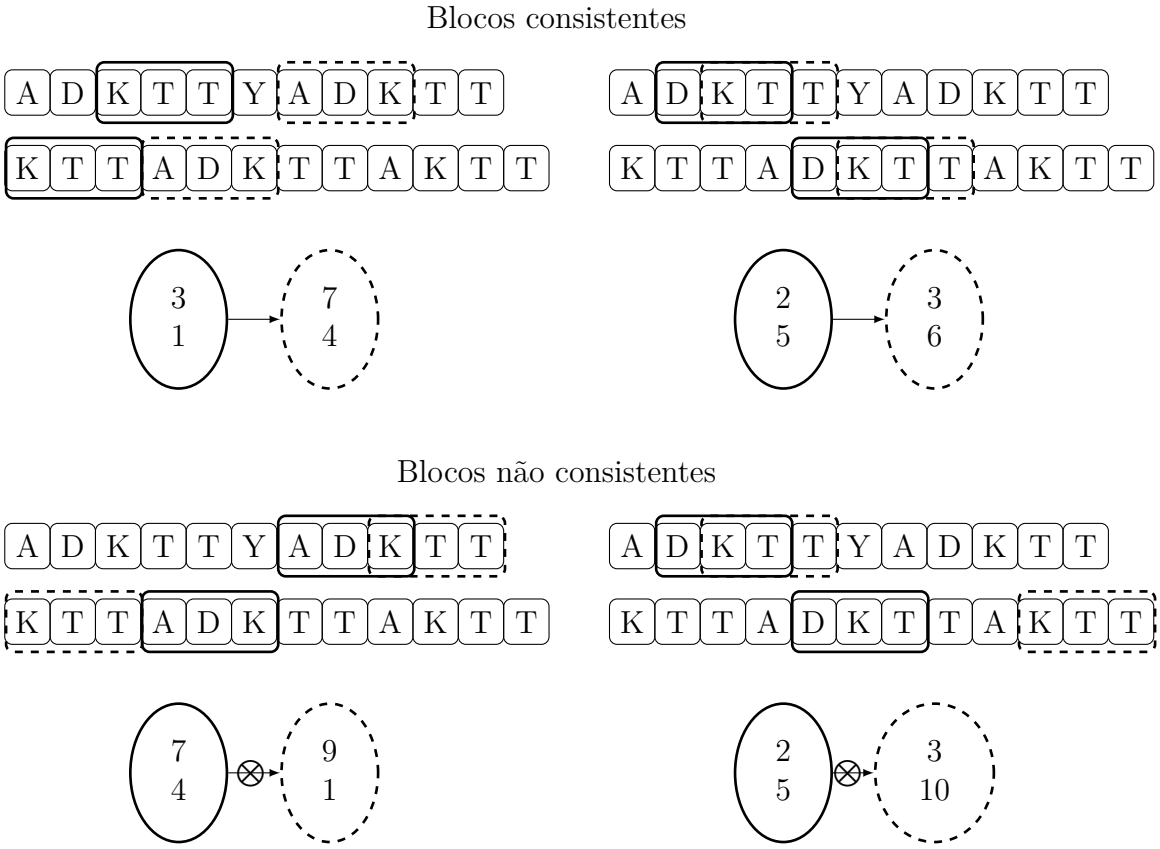


Figura 2.2: Consistência entre blocos - Os retângulos indicam a posição dos blocos iguais em cada uma das duas sequências e cada nó representa um par de blocos iguais (onde os números indicam onde ele inicia em cada sequência). Os dois casos superiores representam blocos consistentes e os dois inferiores blocos inconsistentes entre si (que não podem ser conectados no grafo de consistência). Figura baseada em Almeida [3].

média total e também, em quase todos os casos, a média de cada um dos subconjuntos de teste. Desta forma, este algoritmo foi escolhido como base para o *CRAligner*.

2.2 Operadores de mutação

As operações de mutação são criadas por classes que implementam a interface *Mutation Operator* que possui o método *execute*(*parent*) que recebe um cromossomo pai *parent* como parâmetro e retorna um novo cromossomo derivado deste através da operação de mutação.

Inicialmente foram desenhados três operadores para mutação descritos nas próximas seções.

2.2.1 SimpleMutation

Este operador escolhe de forma randômica uma sequência e posição no alinhamento pai e o percorre até encontrar um bloco de *gaps*. Feito isto o próximo resíduo após este bloco é trocado de posição com o último *gap* do bloco. Caso não seja encontrado um bloco a partir da posição escolhida, este processo é repetido por até 4 outras vezes. Um exemplo desta operação pode ser vista na Figura 2.3.

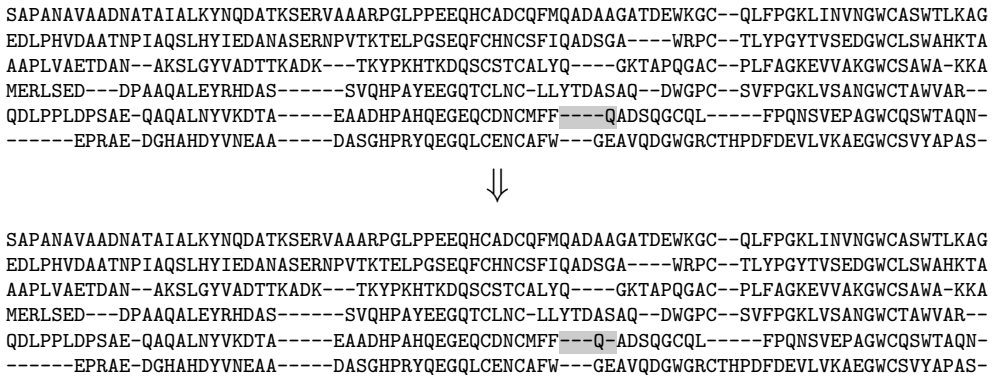


Figura 2.3: Exemplo do operador *SimpleMutation*.

Esta operação de troca é repetida entre 2 e $(2 + |sequences|/2)$ vezes. Uma verificação é realizada ao término destas operações de forma a remover quaisquer colunas contendo apenas *gaps* que possa, porventura, ter se originado.

2.2.2 ShiftGapBlockMutation

Este operador seleciona aleatoriamente um entre os blocos contínuos de *gaps* do alinhamento e uma posição de corte neste. Em seguida, o bloco é dividido em duas seções de acordo com a posição de corte. O trecho final do bloco de *gaps* é então deslocado um número de posições (também escolhido de forma randômica) para a direita. Os resíduos que ocupavam estas posições são deslocados para a esquerda, trocando de posição com o bloco de *gaps* deslocado. Caso uma coluna contendo apenas *gaps* seja criada neste processo ela é eliminada como último passo do algoritmo. Um exemplo da aplicação deste operador pode ser visto na Figura 2.4.

2.2.3 ChangeGapBlockMutation

O operador *ChangeGapBlockMutation* seleciona de forma arbitrária um bloco contínuo de *gaps* e o desloca uma posição para a esquerda ou para a direita. O resíduo que ocupava

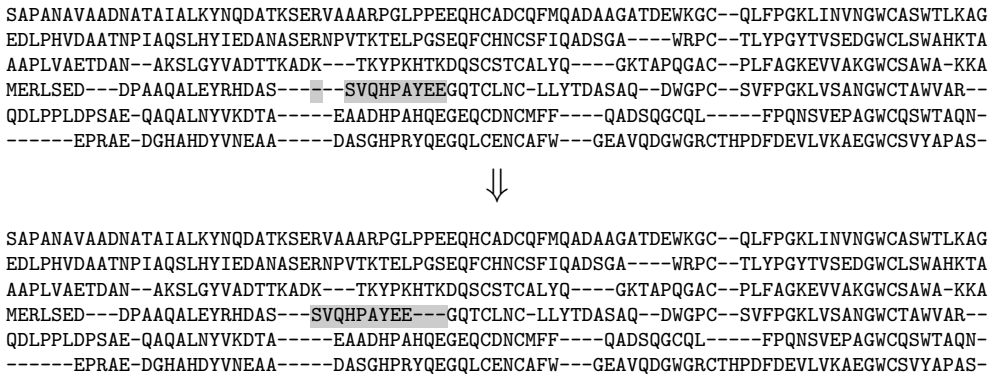


Figura 2.4: Exemplo do operador $ShiftGapBlockMutation$.

este posição é deslocado no sentido contrário, mesclando-se ao próximo bloco contíguo de resíduos. Assim como nos outros operadores, existe a possibilidade de geração de colunas apenas com *gaps* que devem ser removidas.

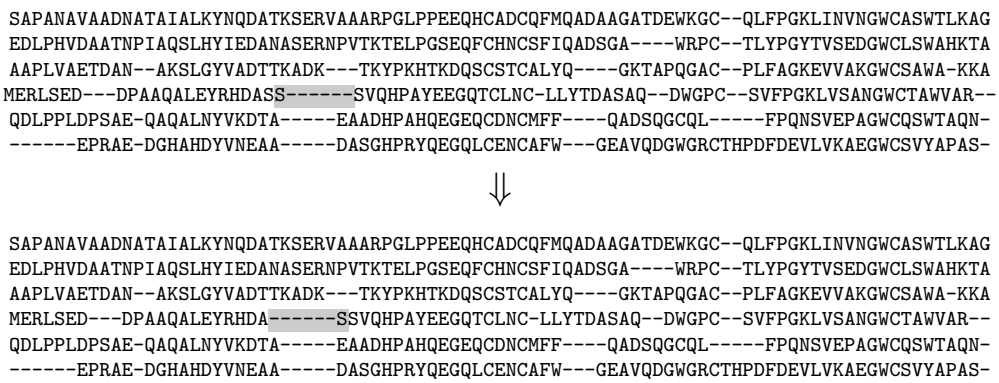


Figura 2.5: Exemplo do operador $ChangeGapBlockMutation$.

2.3 Operadores de *crossover*

As operações de *crossover* do ALGAe são definidas por classes que implementam a interface $CrossOverOperator$ que requisita apenas a implementação do método $execute(p1, p2)$ que recebe dois cromossomos pais ($p1$ e $p2$) e retorna um conjunto de cromossomos filhos que fundem características dos dois pais de acordo com a lógica que o operador implementa.

Inicialmente três operadores de *crossover* foram definidos.

2.3.1 SinglePointCrossOver

Este operador seleciona uma coluna no alinhamento *parent*1 que será utilizada como ponto de corte, dividindo-o em dois sub-alinhamentos $parent1_1$ e $parent1_2$. O algoritmo busca, então, os mesmos resíduos no alinhamento *parent*2 que é seccionado em dois sub-alinhamentos $parent2_1$ e $parent2_2$. Como o corte feito no alinhamento *parent*2 não é necessariamente vertical como no alinhamento *parent*1 os dois sub-alinhamentos gerados tem suas lacunas preenchidas com *gaps*.

Os cromossomos resultantes são gerados pelas combinações $parent1_1 \cup parent2_2$ e $parent2_1 \cup parent1_2$. Colunas compostas apenas por *gaps*, caso criadas, são removidas. Um exemplo deste operador pode ser visto na Figura 2.6.

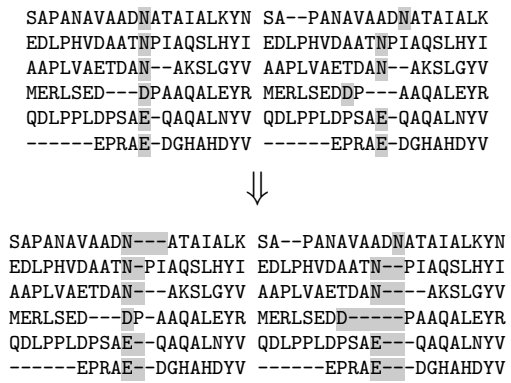


Figura 2.6: Exemplo do operador *SinglePointCrossOver*.

2.3.2 BestPartialAlignmentCrossOver

Este operador divide os alinhamentos pais em dois sub-alinhamentos cada. Para isto ele seleciona, de forma alternada, a sequência com maior similaridade às demais. Desta forma a primeira sequência mais similar se mantém em um sub-alinhamento com a terceira, a quinta e assim por diante. De mesma forma, a segunda sequência mais similar se mantém em outro sub-alinhamento com a quarta, a sexta e demais sequências de ordem par.

São definidas então as sequências consenso para cada um dos quatro sub-alinhamentos (dois gerados a partir de cada alinhamento pai) e estas são alinhadas de forma cruzada utilizando-se o algoritmo de alinhamento global de Needleman e Wunsch. Alinhamentos múltiplos são então criados como resultado, substituindo-se as sequência consenso alinhadas pelo conjunto de sequências que o definiu. Este processo pode ser visto na Figura 2.7.

2.3.3 SequenceSimilarityCrossOver

Neste operador uma árvore filogenética das sequências, criada sobre a pontuação dos seus alinhamentos par a par, é gerada utilizando o algoritmo UPGMA [107]. Cada folha desta árvore representa uma das sequências. Como as sequências são sempre as mesmas, este processo só precisa ser executado uma vez e a árvore gerada é usada em todas as operações subsequentes.

Durante a execução do operador, uma aresta é escolhida de forma aleatória, dividindo a árvore em duas sub-árvores $tree_1$ e $tree_2$. Todas as sequências representadas como folhas em $tree_1$ são separadas dos alinhamentos pais, gerando os sub-alinhamentos $parent1_1$ e $parent1_2$. De mesma forma, os nós representados em $tree_2$ são separados nos sub-alinhamentos $parent2_1$ e $parent2_2$.

Os consensos de cada sub-alinhamento são gerados e alinhados par a par de forma cruzada ($parent1_1$ se alinha a $parent2_2$ e $parent1_2$ se alinha a $parent2_1$). Como resultado, dois alinhamentos múltiplos são expandidos a partir destes alinhamentos de consenso. Como nos outros operadores, colunas contendo apenas *gaps* que porventura tenham se originado são removidas. Este processo é ilustrado na Figura 2.8.

2.4 Métodos de seleção

Os métodos de seleção implementam a interface *Selection*, cujo único método é *select*(*population*, *popSize*) que recebe uma lista de indivíduos *population* que forma uma população intermediária e retorna a lista de indivíduos representando a população selecionada com *popSize* indivíduos no máximo.

Foram implementados dois métodos de seleção para o ALGAe, descrito nas próximas subseções.

2.4.1 RouletteSelection

Este método de seleção funciona como uma roleta, onde cada indivíduo equivale a um "arco" nesta roleta proporcional a sua aptidão (*fitness*). A cada seleção a roleta "gira" e um elemento é selecionado. Desta forma, os indivíduos mais aptos tem uma probabilidade maior de serem selecionados, porém ao mesmo tempo garante aos menos aptos uma chance, mesmo que menor, disto ocorrer. Esta lógica está descrita no Algoritmo 5.

2.4.2 MostFittedSelection

Este método seleciona os indivíduos mais aptos dentre a população até preencher *popSize*. Ao contrário do algoritmo *RouletteSelection* descrito na Seção 2.4.1 este algoritmo não

Algoritmo 5: RouletteSelection

Entrada: População inicial *population*
Saída: p' - população selecionada de tamanho *popSize*

$worstScore \leftarrow \infty$
$p' \leftarrow \emptyset$
for *all c in p* **do**
  if $c.score < worstScore$ **then**
    $worstScore \leftarrow c.score$
  end
end
$totalScore \leftarrow 0$
for *all c in p* **do**
  $totalScore \leftarrow c.score - worstValue$
end
$startProbability \leftarrow 0$
for *all c in p* **do**
  $endProbability \leftarrow startProbability + (c.score - worstValue)/totalScore$
  $c.accProbability \leftarrow [startProbability..endProbability)$
  $startProbability \leftarrow endProbability$
end
for *all i in* $[1..popSize]$ **do**
  $selected \leftarrow random(0..1)$
  $selectedChromossome \leftarrow c \in p : selected \in c.accProbability$
  $p' \leftarrow p' \cup selectedChromossome$
end
return p'

dá chance alguma de sobrevivência aos indivíduos menos aptos.

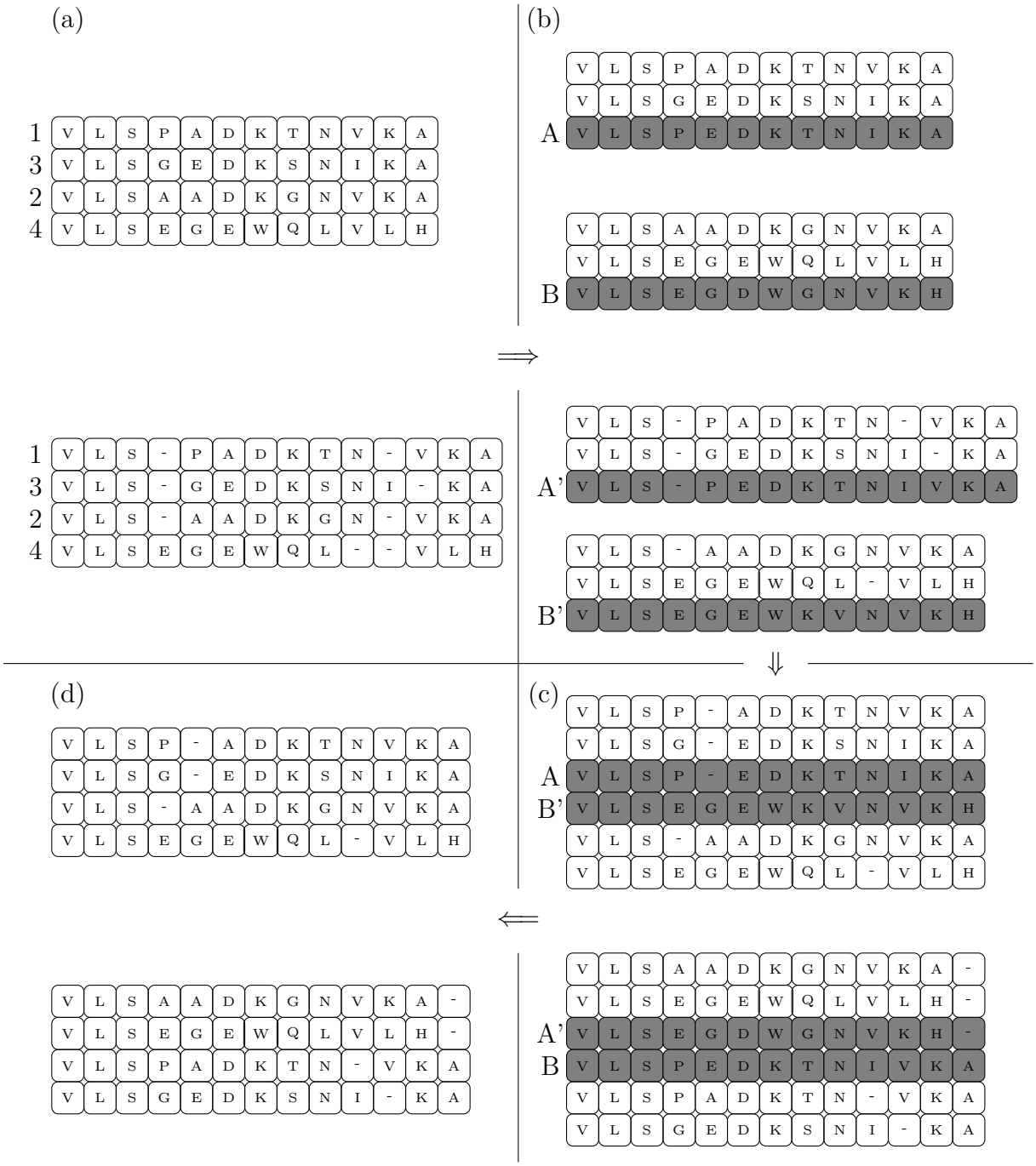


Figura 2.7: Exemplo do operador *BestPartialAlignmentCrossOver* e do seu processo. As sequências dos alinhamentos pais são ordenadas de acordo com a sua similaridade com as demais sequências. As sequências originais (a) são separadas em dois sub-alinhamentos de forma alternada e os consensos (b) gerados. Por fim, uma sequência consenso vindo de um alinhamento pai e seu complemento vindo do outro são alinhados (c) e servem de guia para o alinhamento final, gerando dois novos alinhamentos filhos (d).

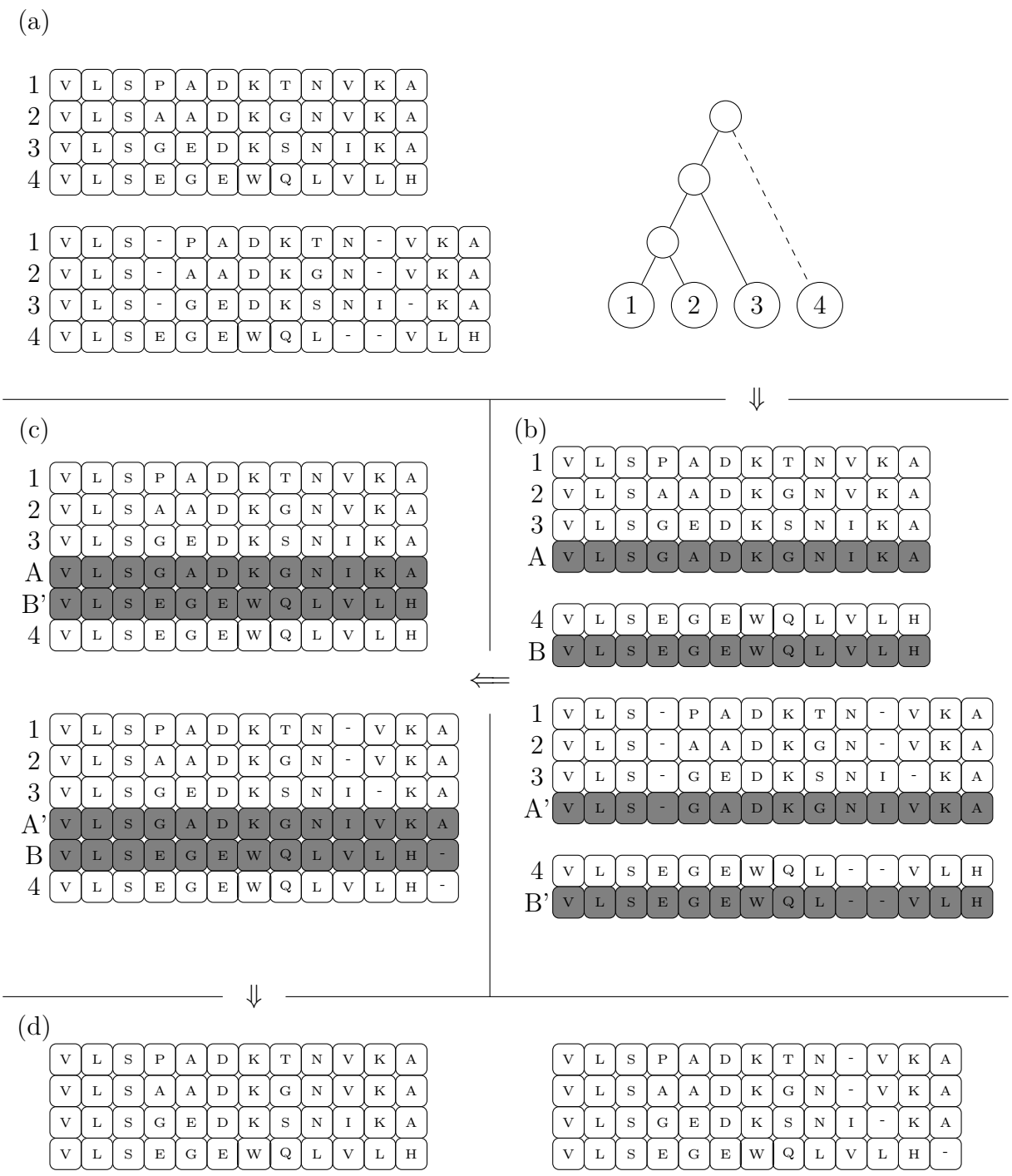


Figura 2.8: Exemplo do operador *SequenceSimilarityCrossOver* e do seu processo. As sequências envolvidas no alinhamento são utilizadas para a criação de uma árvore filogenética (a), de acordo com as suas similaridades par a par, onde cada sequência ocupa uma folha desta árvore. Uma aresta é selecionada (aquela tracejada no exemplo) e os alinhamentos pai são divididos em sub-alinhamentos de acordo com as sub-árvores geradas. Em seguida as sequências consenso (b) são geradas e alinhadas de forma alternada (c), servindo como guias para criação de dois novos alinhamentos (d).

2.5 Funções objetivo

As funções objetivo (*fitness function*) são implementadas seguindo a interface *Score* que possui o método $calculateScore(c)$, que recebe como parâmetro um cromossomo c e retona um valor de ponto flutuante que representa o valor de *fitness* deste cromossomo segundo a função objetivo definida.

Foram inicialmente implementados duas funções objetivo.

2.5.1 SumOfPairsMatrixBasedScore

Este método de soma de pares considera, baseado em uma matriz de substituição, o valor da soma par a par de todos os resíduos, por coluna. Podemos definir isto conforme a Equação 2.2, onde n representa o número de sequências do alinhamento, m o seu tamanho, $\sigma(c_1, c_2)$ a distância entre os resíduos (ou resíduo e *gap*) c_1 e c_2 e seq_n^i o i-ésimo elemento da sequência n deste alinhamento.

$$fitness = \sum_{i=1}^{m} \sum_{j=1}^{n-1} \sum_{k=j+1}^{n} \sigma(seq_j^i, seq_k^i) \tag{2.2}$$

2.5.2 GapAffinitySumOfPairsMatrixBasedScore

Este método possui similaridades com o método de soma de pares descrito na Seção 2.5.1. Porém, assume-se que *indels* (inserções ou remoções ao longo do processo evolutivo) ocorrem em conjuntos contíguos de posições. Por isso, a pontuação de conjuntos de *gaps* contíguos não é penalizada posição por posição e sim através de uma penalização mais alta para abertura de uma sequência de *gaps* e uma penalização menor para cada *gap* consecutivo. Isto pode ser visto na Equação 2.3, onde *gop* indica o custo de abertura de um *gap*, *gep* o custo para expandi-lo e n o tamanho do bloco de *gaps*.

$$score_{gap} = gop + n \times gep \tag{2.3}$$

Considerando isto, a função afim para *gaps* se relaciona a função de soma de pares (apresentada na Equação 2.2) de acordo com a Equação 2.4.

$$\sigma(c_1, c_2) = \begin{cases} gop + gep & \text{se } c_1 \text{ ou } c_2 \text{ inicia um bloco de } gaps, \\ gep & \text{se } c_1 \text{ ou } c_2 \text{ estende um bloco de } gaps. \end{cases} \tag{2.4}$$

2.6 Configuração e *Log*

Como forma de tornar a ferramenta mais flexível e apta a realizar diversos testes com configurações diversas, o ALGAe foi desenvolvido com um mecanismo de configuração que permite, via um arquivo de texto com formato $Chave : Valor$, definir diversos de seus parâmetros.

A classe $ConfigurableGeneticAlgorithim$ é responsável por processar estes registros de configuração e aplicar aqueles que são válidos. Esta classe também é reponsável por instanciar as classes requisitadas (para estratégias de população inicial, operadores, função de *fitness* etc.) através do mecanismo de reflexão presente na linguagem Java. Este mecanismo permite a instanciação de objetos e execução de métodos baseados em seus nomes, fornecidos como *strings* de texto.

As informações pertinentes apenas à execução do algoritmo genético, como por exemplo quais operadores serão utilizados, são armazenadas no escopo da própria classe *ConfigurableGeneticAlgorithim*. Outras são armazenadas em um *singleton* (padrão de projeto onde uma classe mantém uma única instância disponível para todo o contexto de execução, de forma a servir como ponto de compartilhamento de informação) da classe *Environment*, de forma a serem acessíveis por qualquer elemento do ALGAe que o necessite. Exemplos destas informações são: os custos de penalidade de *gaps* (tanto o custo *gop* de abertura quanto o custo *gep* de extensão) e a matriz de substituição utilizada.

Para que possa ser feita uma análise fina sobre como o algoritmo genético está sendo executado e a população evoluindo, o ALGAe possui também um mecanismo de *log*.

Este mecanismo gera um arquivo em formato XML para as execuções do ALGAe contendo as informações de configuração utilizadas (de forma que a análise dos dados possa ser baseada nos parâmetros utilizados) e os dados de cada execução do algoritmo. Como a ferramenta permite múltiplas execuções de um mesmo cenário (como os algoritmos genéticos não são determinísticos isto permite uma análise estatística mais apurada), as execuções são divididas em *test sets*, que representam um cenário (conjunto de sequências que devem ser alinhados) e, dentro destes, em *test runs*, que representam a execução de um alinhamento para este cenário.

Para cada *test run*, o ALGAe gera um sumário da execução de cada geração, separando por tipo de operação que gerou os indivíduos, quantos deles existem na população e os valores máximos e mínimos de *fitness*.

2.7 Primeiros Resultados

Os testes iniciais do ALGAe foram feitos utilizando como referência o GAADT, um alinhador que utiliza algoritmos genéticos baseado em tipos abstratos de dados e tema da

dissertação de mestrado de Santos [100]. Utilizamos este alinhador como teste inicial por ter sido desenvolvido em um escopo similar ao do ALGAe.

O teste foi realizado utilizando os operadores inicialmente desenvolvidos para o ALGAe, tendo como alvo os cenários descritos no trabalho do GAADT. Os resultados obtidos são mostrados na Tabela 3.2. A métrica utilizada foi a SP do benchmark BAliBASE versão 2 como medida de *fitness* dos resultados. Mais informações sobre esta métrica e sobre o BAliBASE podem ser obtidas na Seção 1.5.1.

Tabela 2.2: Resultados da métrica SP (porcentagem) dos alinhadores ALGAe e GAADT para a família de proteínas 1aab e grupos correlatos (em porcentagem). Quanto mais próximo de 100% melhor o resultado.

	1aab	1fjlA	1hpi	1csy	1tgxA	média
`GAADT`	88,1	81,4	70,8	70,3	69,2	76,0
`ALGAe` (média)	88,4	93,8	88,3	76,2	68,5	83,0
`ALGAe` (máximo)	89,6	100,0	96,2	80,7	77,2	88,7

Para obter um resultado mais acurado, foram realizadas 20 baterias de teste para cada cenário, sendo os resultados apresentados a média e o máximo das execução. Como parâmetros foi utilizada uma população de 250 indivídos e 2000 gerações para o alinhador ALGAe. Não foi possível obter informações sobre estes parâmetros para o GAADT.

Como pode ser visto na Tabela 3.2, em 4 dos 5 dos cenários testados, o ALGAe obteve melhores resultados. Além disto o algoritmo, mesmo em seu estágio inicial de desenvolvimento, apresentou um resultado 9,2% melhor que o do GAADT sobre os resultados médios e 16,7% considerando apenas os melhores resultados.

O mesmo conjunto de operadores foi testado com o primeiro conjunto de referência do BAliBASE versão 3, que se divide em dois sub-conjuntos: RV11 e RV12. Para este teste foram utilizados como função objetivo tanto a soma de pares simples (SP) como a soma de pares por afinidade de *gaps* (GASP) sobre diversas matrizes de substituição. Os resultados apresentados na Tabela 2.3 são as médias de execução de 20 baterias de teste, com população de 100 indivíduos e 650 gerações.

Caso os valores obtidos sejam comparados a ferramentas populares, podemos ver que os resultados são inferiores. Isto pode ser visto na Tabela 2.4, baseada no trabalho de Hang [47]. Não foi possível estender este teste para o GAADT visto que seu código não estava disponível e a documentação não cobria os grupos alvo de teste.

Desta forma, a comparação com a ferramenta GAADT indica que a ferramenta ALGAe pode ser promissora, porém a comparação com ferramentas mais populares também indica que existe um grande trabalho de otimização e melhoria necessário nesta.

O Capítulo 3 descreve os experimentos realizadas neste trabalho com objetivo de melhorar os resultados obtidos.

Tabela 2.3: Resultados do ALGAe para os grupos RV11 e RV12 do BAliBASE3 (métrica SP em porcentagem). Os melhores resultados obtidos para cada grupo são mostrados em negrito.

Reference	BLOSUM62	BLOSUM80	PAM100	PAM250
RV11 (SP)	33,6	34,9	28,9	28,9
RV11 (GASP)	35,6	**38,1**	30,5	32,7
RV12 (SP)	73,9	75,7	74,2	75,2
RV12 (GASP)	73,2	75,9	72,8	**76,1**

Tabela 2.4: Resultados do alinhador ALGAe, utilizando os melhores resultados obtidos, comparado a ferramentas populares para MSA (baseado nos resultados de Hang [47]).

Conjunto BAliBASE	T-COFFEE	Muscle	ClustalW	ALGAe
`RV11` (SP)	47,1	47,9	43,4	38,1
`RV12` (SP)	85,5	84,7	81,7	76,1

Capítulo 3

Otimizações do ALGAe

Como os resultados apresentados no Capítulo 2 indicam, o ALGAe em sua implementação inicial superou a ferramenta GAADT [100], desenvolvida em um escopo similar. Comparado aos resultados disponíveis deste, o ALGAe superou o GAADT em 9,2% considerando o cenário médio de execução, e em 16,7%, considerando apenas a melhor execução do ALGAe.

Apesar disto, os resultados se mostraram aquém daqueles de ferramentas populares como pode ser visto na Tabela 2.4, baseada no trabalho de Hang [47].

Esta comparação serviu como indicador de que a ferramenta ALGAe deveria ser revista, melhorada e otimizada para que seus resultados se aproximassem ou conseguissem em alguns casos superar soluções já existentes.

Muitos dos testes realizados utilizaram principalmente os conjuntos de Referência 1 do BAliBASE, por serem conjuntos com sequências razoavelmente curtas. Isto permite que os testes sejam realizados com baterias grandes (de forma a minimizar desvios devido a natureza estocástica dos algoritmos genéticos) em um tempo moderado. Com isto um número maior de testes e tentativas de melhoria são possíveis em menos tempo.

Este capítulo descreve as tentativas realizadas visando melhorar os resultados obtidos inicialmente com o ALGAe. Tanto os resultados positivos quanto os negativos, visando o seu registro histórico, estão descritos aqui. Uma análise dos resultados obtidos e ações tomadas também é apresentada.

3.1 Análise inicial

Como uma primeira análise foi verificado como o resultado converge com o uso do ALGAe de forma a medir a eficiência dos operadores e função objetivo utilizadas.

Para este teste foram executados testes com a métrica definida na Equação 3.1 para

a execução do ALGAe.

$$m(gen) = \frac{SP_{max}(gen)}{SP_{max}(n)} \tag{3.1}$$

Nesta equação $m(gen)$ indica o valor da métrica definida para a execução da geração *gen* do algoritmo genético, $SP_{max}(gen)$ o valor máximo de SP obtido nesta geração, n o número de gerações que são executadas e $SP_{max}(n)$ o valor máximo de SP encontrado nesta geração (sendo o valor da solução retornada pela execução). Desta forma quanto mais próximo de 1.0 o valor se aproxima, mais isto indica que o resultado desta geração se aproxima do melhor resultado. Um valor superior a 1.0 indica que foi encontrado um resultado melhor (em termos de SP) do que o resultado retornado ao final da execução, mas que se perdeu ao longo do processo de seleção.

Ao utilizar uma métrica que relativiza o resultado ao valor de saída do algoritmo diversos cenários distintos podem ser considerados conjuntamente para o teste. Desta forma todo o conjunto RV12 do BAliBASE versão 3.0 pode ser testado e a média das execuções utilizada.

A escolha deste conjunto se deveu ao fato de se ter um resultado razoavelmente bom e ser um conjuntos de sequências não muito longas, permitindo rápidas execuções (de forma a reduzir o fator estocástico no resultado), servindo bem como uma primeira ferramenta de análise.

Este conjunto foi executado com diversas combinações de valores para abertura e extensão de *gaps* e de penalizações para alinhamentos entre *gaps* em uma mesma coluna. A matriz de substituição utilizada foi a BLOSUM80.

Como pode ser visto na Figura 3.1, o resultado converge rapidamente (em aproximadamente 100 gerações para todos os casos) para o resultado máximo. Isto indica algumas possibilidades a serem exploradas:

- A função objetivo não reflete, a partir de certo ponto, efetivamente o melhor resultado (segundo o BAliBASE) e, com isto, a melhor solução estabiliza em um valor aquém do que poderia se obter com uma função objetivo mais próxima ao que é considerado o melhor resultado.

- Os operadores utilizados não são eficientes em melhorar o resultado após um primeiro refinamento, degenerando a população.

Outro ponto é que os resultados foram distintos dependendo dos valores de penalização em relação a *gaps*, sendo um ponto de estudo interessante verificar como estes valores afetam o resultado final.

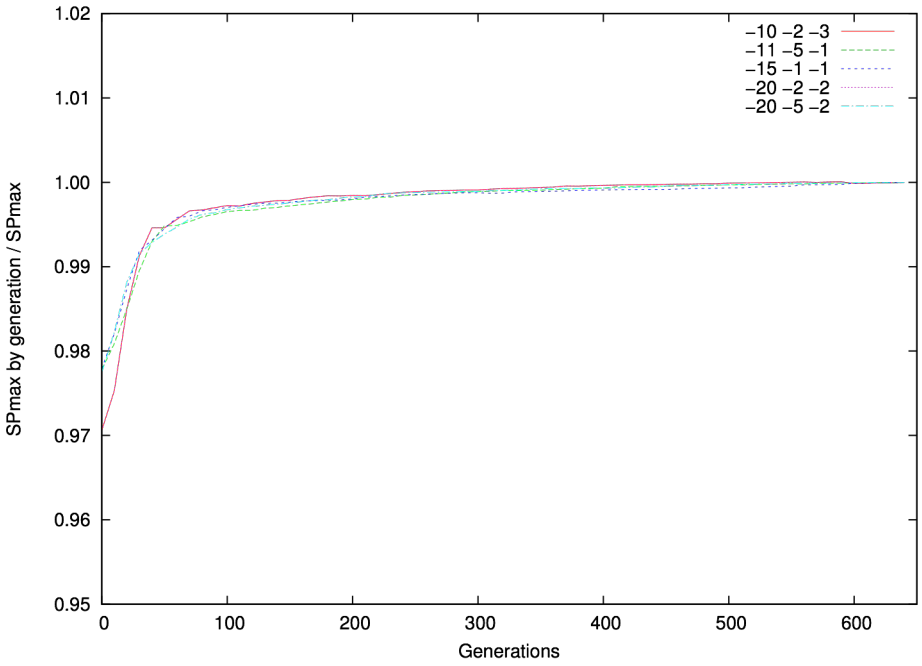


Figura 3.1: Evolução do resultado SP do ALGAe por geração. Medida feita sobre o conjunto RV12 do BAliBASE 3.0 com matriz de substituição BLOSUM80. Os números na legenda indicam valores para abertura de um *gap*, extensão de um *gap* já existente e penalização para alinhamento *gap* x *gap*, nesta ordem.

3.2 Análise das penalizações para *gaps*

As análises iniciais descritas na Seção 3.1 indicam que os resultados são afetados pelos valores de penalização para *gaps* (abertura, extensão e alinhamento *gap* x *gap*). Devido a isto, um estudo para verificar como estes parâmetros afetavam o resultado final foi realizado.

As matrizes de substituição PAM [26] e BLOSUM [48] são bons recursos para auxiliar na pontuação de similaridades em sequências de proteínas, porém provêem um modelo apenas para *matches* e *mismatches*, deixando uma lacuna no que tange a pontuação de *indels*, representados por *gaps* no alinhamento.

Por não conter uma base teórica tão forte e, consequentemente um modelo bem estabelecido como as matrizes de substituição, a penalização de *gaps* é, em geral, atribuída de forma empírica.

3.2.1 Trabalhos relacionados

Um dos primeiros trabalhos a realizar uma análise empírica da penalização de *gaps* foi realizado por Vingron e Waterman [124]. Neste trabalho é feita uma análise de como valores distintos de abertura e extensão de *gaps* afetam o alinhamento local de duas sequências de imunoglobulinas associadas às cadeias variáveis do domínio do anticorpo Fab, cujas similaridades estruturais haviam sido apresentadas por Amzel e Poljak [7].

Reese e Pearson [95] posteriormente publicaram um estudo sobre valores de penalização para *gaps* para maximizar a efetividade de busca por trechos homólogos em bancos de sequências. Tal trabalho focou-se na família de matrizes PAM e propôs uma relação sistemática entre distância PAM e penalidades para abertura e extensão de *gaps*. Baseado nos resultados experimentais este trabalho também sugeriu que não existe alteração significativa nos valores de extensão de *gaps* em relação à distância PAM. Além disto, este trabalho extrapola, baseado nos resultados para a família PAM, os valores de penalização para algumas matrizes BLOSUM.

Wrabl e Grishin [129] desenvolveram um algoritmo para otimizar *gaps* de forma local na base FSSP [52]. Além disto neste trabalho foi proposta uma alteração nos valores de penalidade de abertura de *gaps* no ClustalW [117] associado ao resíduo imediatamente anterior a esta. Resíduos com características hidrofóbicas (Cisteína, Fenilalanina, Isoleucina, Leucina, Valina, Triptofano e Tirosina) tendem a estar associados com valores de penalização maiores enquanto resíduos com cadeias laterais menores (Ácido aspártico, Glicina, Asparagina, Prolina e Serina) tendem a ter penalizações também menores. Mais informações sobre a estrutura dos aminoácidos pode ser visto na Seção 1.1.2.

Agrawal e colegas [2] realizaram uma análise sobre a família BLOSUM, utilizando significância estatística par a par (*pairwise statistical significance*), focando principalmente no valor de abertura de *gaps*, sendo que o valor de extensão foi mantido constante ($k = 2$).

3.2.2 Metodologia e resultados

Este estudo analisou como as penalidades de *gaps* afetam os resultados dos alinhamentos gerados pelo ALGAe. Para tanto foi utilizado um conjunto bem conhecido de alinhamentos com resultados para comparação também bem definidos.

A escolha do conjunto de testes foi o sub-conjunto V2 do conjunto de referência 1 do BAliBASE 3.0 (Conjunto RV12). Este conjunto representa alinhamentos de sequências equidistantes que compartilham entre 20 e 40% de identidade. Mais informações sobre o *benchmark* BAliBASE e os conjuntos de teste que o formam podem ser encontrados na Seção 1.5.1.

A escolha de tal conjunto de testes deveu-se ao fato de tratar de sequências equidistantes, o que faz com que a variação de penalização dos *gaps* entre cada par de sequências

seja mais uniforme do que nos outros conjuntos. A escolha do sub-conjunto V2 sobre o V1 (RV11) se deveu ao fato da similaridade ser maior no primeiro, gerando resultados melhores (mais próximos do resultado de referência) e, com isto, fazendo com que o erro no resultado fosse proporcionalmente menor, gerando dados com maior precisão para a análise. O fato dos alinhamentos e sequências deste conjunto serem menores é, em menor escala, um fator de seleção também pois permite a execução de um número maior de baterias de testes em um tempo menor.

De forma a isolar os parâmetros de interesse, todos os outros parâmetros do ALGAe foram mantidos ao longo de todas as execuções feitas. Isto inclui todos os operadores e suas probabilidades e os métodos de criação e seleção de população. Mais informações sobre estes parâmetros podem ser vistos no Capítulo 2.

A função de aptidão (*fitness function*) utilizada foi a soma de pares com função afim para a penalização de *gaps*. Esta função como apresentada na Seção 2.5.2. Considerando um alinhamento com *gaps* de n sequências com tamanho m, a função pode ser definida como na Equação 3.2.

$$fitness = \sum_{i=1}^{m} \sum_{j=1}^{n-1} \sum_{k=j+1}^{n} \sigma(seq_j^i, seq_k^i) \tag{3.2}$$

Onde seq_x^y indica o y-ésimo elemento da sequência x no alinhamento e $\sigma(e_1, e_2)$ a pontuação do alinhamento dos elementos e_1 e e_2. Como o teste foi feito sobre um conjunto de sequências de proteínas, este valor é baseado em uma matriz de substituição para *matches* e *mismatches*. Caso o alinhamento envolva um *gap*, uma função afim como a definida na Equação 3.3 é utilizada.

$$score_{gap} = q + (k)r \tag{3.3}$$

Nesta equação q indica o custo de abertura de um *gap*, r o valor de estender este em um caracter e k o tamanho total do bloco de *gaps*. Isto faz com que a penalização de um conjunto de *gaps* seja penalizado como se fosse um evento único, de forma a considerar a natureza da ocorrência de *indels*. No caso de alinhamento *gap-gap* um valor constante g é aplicado, de forma similar ao que ocorre com *matches* e *mismatches*. Podemos correlacionar isto com a Equação 3.2 de acordo com a Equação 3.4 .

$$\sigma(c_1, c_2) = \begin{cases} q + r & \text{se } c_1 \text{ ou } c_2 \text{ inicia um bloco de } gaps, \\ r & \text{se } c_1 \text{ ou } c_2 \text{ estende um bloco de } gaps, \\ g & \text{se } c_1 \text{ e } c_2 \text{ são gaps.} \end{cases} \tag{3.4}$$

Com esta configuração o conjunto RV12 foi testado para as métricas SP e TC do BAliBASE versão 3.0 para diversas combinações de valores de q, r. O teste também

foi realizado com as matrizes de substituição BLOSUM62 e BLOSUM80 e com valores distintos para penalização para g.

Para as penalizações g em uma mesma coluna um teste inicial foi feito, utilizando a matriz BLOSUM80 e algumas combinações de q e r. Observando os resultados, apresentados na Tabela 3.1, é possível notar a tendência do resultado decair conforme esta penalidade aumenta. Devido a isto o valor de penalizações *gap-gap* foi definido como $g = 0$ em todos os testes posteriores.

Os testes utilizaram também duas importantes matrizes da família BLOSUM: a BLOSUM62 e BLOSUM80, com uma série de valores para q e r.

Para o caso da matriz BLOSUM80 os valores utilizados foram $q = [-4, -20]$ e $r = [-5, 0]$ enquanto para BLOSUM62 os valores utilizados foram $q = [-2, -15]$ e $r = [-5, 0]$. Os testes com estes parâmetros foram realizados para todos os cenários do conjunto RV12 do BAliBASE versão 3.0, sendo que para cada um deles 20 execuções independentes foram realizadas e a média destas calculada para as métricas de interesse. Os resultados podem ser vistos na Figura 3.2.

A análise dos resultados dos valores de abertura e extensão (q e r) de *gaps* apresentados na Figura 3.2 permite verificar que, ao manter um dos valores constantes, o valor máximo tende a ficar próximo de um mesmo valor para o outro parâmetro, servindo como indicador de independência entre eles. Também é possível notar que a métrica SP é mais sensível a mudanças nas penalidades de *gaps* que a métrica TC.

Considerando o desvio padrão e o comportamento independente de q e r, pode-se identificar, considerando apenas valores inteiros, que o melhor valor para estes parâmetros, de acordo com as observações experimentais são: $q = 5$ e $r = 1$ para BLOSUM62 e $q = 9$ e $r = 2$ para BLOSUM80.

Comparando os valores obtidos neste experimento com os valores encontrados nos trabalhos de Reese e Pearson [95] e Agrawal e colegas [2] é possível notar discrepâncias. A Tabela 3.2 apresenta os valores propostos nestes trabalhos comparados aos valores experimentais encontrados para o ALGAe e a diferença percentual da métrica SP considerando o uso do valor empírico encontrado neste trabalho e o valor que seria obtido caso os valores propostos nestes trabalhos fossem utilizados para a execução.

Apesar da discrepância encontrada em relação a outros trabalhos, alguns pontos devem ser considerados: o conjunto de teste é menor do que aquele utilizado nos outros trabalhos, com características mais homogêneas e os valores apresentados nestes são associados a busca em bancos de sequências e não especificamente para MSA. Porém, tal teste serve como indicador, exatamente por isto, que os valores de penalização para *gaps* é fortemente dependente das sequências alvo. Isto também serve como indicador de que um valor geral para tal penalidade, visando resultados de qualidade, pode não ser possível. Tal argumento já havia sido proposto por Barton e Sternberg [12].

Tabela 3.1: Resultados dos testes de penalização $gap \times gap$ para o conjunto RV12 do BAliBASE versão 3.0. Foram utilizados valores $q \in \{-10, -11, -12, -13, -15 \text{ e } -20\}$ e $r \in \{-1, -2 \text{ e } -5\}$. O valor para $gap \times gap$ estava no intervalo $g \in \{0, -3\}$. A matriz de substituição utilizada foi a BLOSUM80. O melhor resultado é apresentado em negrito.

	-1	**-2**	**-5**
-10	80,56	**80,58**	77,35
-11	80,28	80,33	77,40
-12	80,28	80,25	77,04
-13	79,96	80,14	76,08
-15	79,28	79,13	74,30
-20	73,42	74,65	69,39

(a) $gap \times gap = 0$ (**SP**)

	-1	**-2**	**-5**
-10	**54,94**	54,90	51,07
-11	54,53	54,56	51,68
-12	54,72	54,72	51,27
-13	54,43	54,67	50,07
-15	53,73	53,73	48,15
-20	45,86	47,93	40,90

(b) $gap \times gap = 0$ (**TC**)

	-1	**-2**	**-5**
-10	74,56	74,86	74,80
-11	75,18	74,08	75,18
-12	**76,67**	75,00	75,11
-13	74,95	76,18	76,09
-15	75,62	75,73	75,65
-20	72,41	72,58	72,64

(c) $gap \times gap = -1$ (**SP**)

	-1	**-2**	**-5**
-10	50,18	50,36	50,23
-11	51,27	51,58	51,42
-12	**51,73**	50,93	51,06
-13	50,87	50,78	50,82
-15	50,87	50,94	50,70
-20	43,96	44,24	44,49

(d) $gap \times gap = -1$ (**TC**)

	-1	**-2**	**-5**
-10	73,71	73,73	74,23
-11	74,41	74,24	74,21
-12	73,84	74,15	73,88
-13	73,78	73,76	73,65
-15	**75,04**	74,69	74,70
-20	71,05	71,33	71,31

(e) $gap \times gap = -2$ (**SP**)

	-1	**-2**	**-5**
-10	48,57	48,65	49,45
-11	50,11	49,90	50,06
-12	49,86	**50,54**	50,05
-13	49,00	49,25	48,90
-15	49,10	48,90	48,92
-20	42,94	42,78	42,57

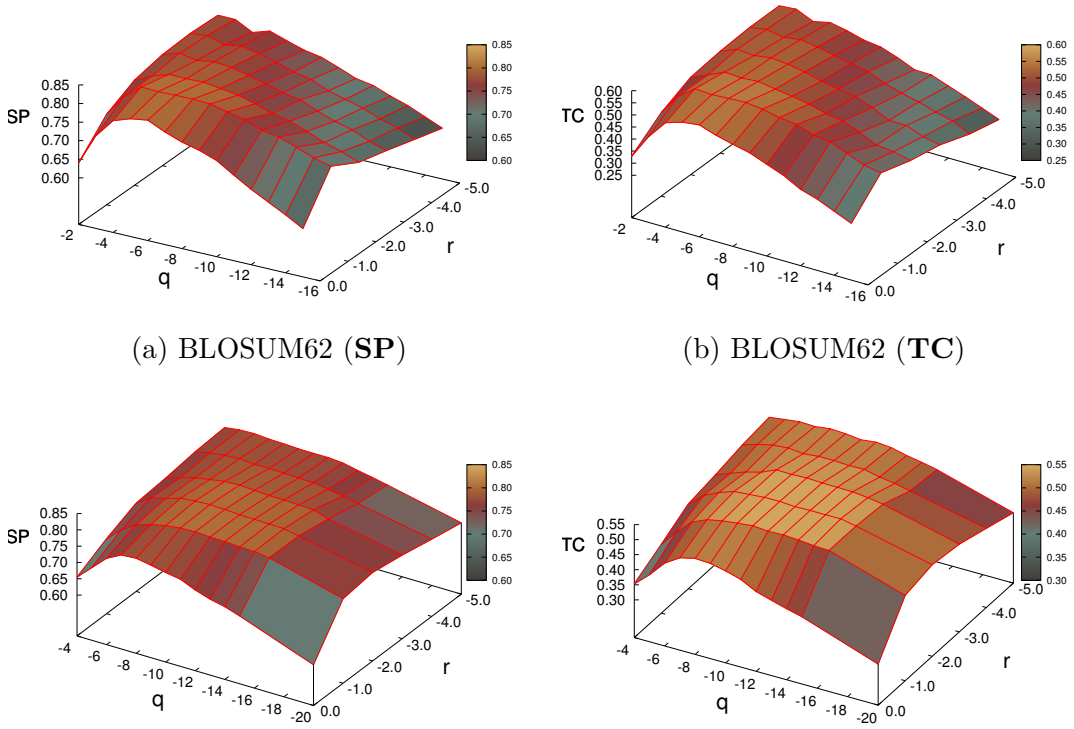
(f) $gap \times gap = -2$ (**TC**)

	-1	**-2**	**-5**
-10	74,30	72,95	74,64
-11	73,22	73,21	72,37
-12	73,20	**75,08**	73,30
-13	73,67	73,36	73,49
-15	74,91	74,87	74,57
-20	71,10	71,17	70,51

(g) $gap \times gap = -3$ (**SP**)

	-1	**-2**	**-5**
-10	48,30	47,89	48,38
-11	48,82	48,40	49,13
-12	48,32	**49,64**	48,82
-13	48,10	47,74	48,04
-15	48,03	48,42	48,16
-20	42,29	42,44	42,12

(h) $gap \times gap = -3$ (**TC**)



(a) BLOSUM62 (**SP**)

(b) BLOSUM62 (**TC**)

(c) BLOSUM80 (**SP**)

(d) BLOSUM80 (**TC**)

Figura 3.2: Testes de penalização para abertura (q) e extensão (r) de *gaps*, com valor $g = 0$. Os testes foram feitos para as métricas SP e TC do BAliBASE 3.0 para o conjunto RV12 utilizando as matrizes de substituição BLOSUM62 (figuras (a) e (b)) e BLOSUM80 (figuras (c) e (d)).

Tabela 3.2: Resultados dos testes de valores para penalização de abertura(q) e extensão(r) de *gaps*. São indicadas as combinações de (q,r) onde os resultados experimentais apresentaram melhores valores que aqueles propostos em outros trabalhos. As colunas mais a direita apresentam a melhoria obtida para SP, em pontos percentuais, ao se utilizar estes valores ao invés dos propostos na literatura.

Matriz	Melhor resultado (q,r)	1 - Agrawal e colegas (q,r)	2 - Reese e Pearson (q,r)	Melhoria para 1	Melhoria para 2
BLOSUM62	(5,1)	(11,2)	(9,5)	4.0 %	14.0 %
BLOSUM80	(9,2)	(13,2)	(13,5)	0.7 %	6.0 %

3.3 Análise de melhor matriz par-a-par

Os resultados obtidos pelo experimento apresentado na Seção 3.2 indicaram que um valor de penalização para *gaps* e mesmo uma matriz genérica para resolver o problema de MSA

pode resultar em uma solução de baixa qualidade ou, ao menos, um resultado que poderia ser melhorado com uma escolha mais apropriada destes parâmetros.

Devido a isto, um novo experimento foi realizado de forma a buscar uma seleção dinâmica da matriz (e, em caso de sucesso, dos parâmetros de penalidade de *gaps*) para resolver um dado MSA. De forma ainda mais específica, foi testado o uso de uma matriz para cada par de sequências do alinhamento desejado.

3.3.1 Metodologia e resultados

Para realizar este experimento, os casos de teste do conjunto RV12 do BAliBASE versão 3.0 foram divididos em cenários de alinhamento par a par de sequências, cobrindo todas as combinações possíveis de pares de sequência em cada um dos casos de teste originais.

Utilizou-se para alinhar estes cenários uma variação do algoritmo de Needleman e Wunsch [78], descrito na Seção 1.2.1, que utiliza uma função afim para penalização de *gaps*. Esta função de aptidão é descrita em maiores detalhes na Seção 3.2.2.

Cada um dos cenários foi executado com matrizes de substituição da família BLOSUM variando de BLOSUM30 a BLOSUM100, com intervalos de 10 entre elas, além das matrizes BLOSUM45 e BLOSUM62. Para cada uma dessas matrizes, foram testados valores de $q = [-1, -20]$ e $r = [-1, -6]$.

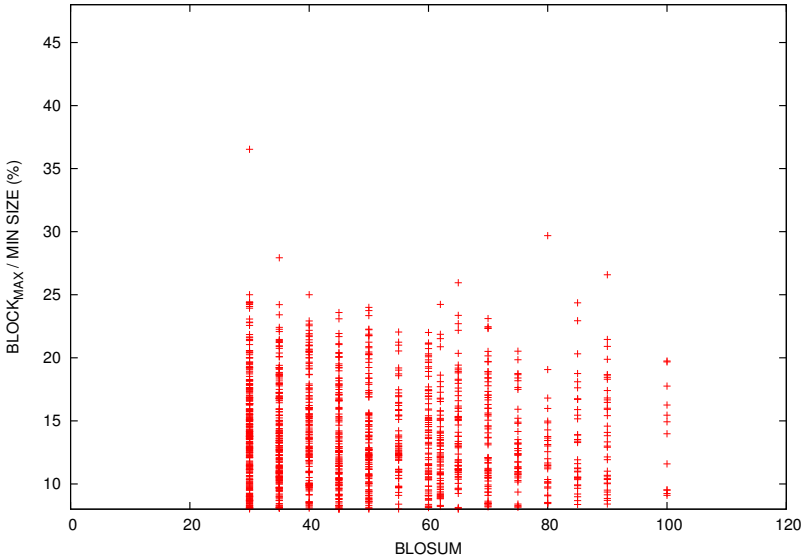
Cada um dos alinhamentos gerados foi comparado ao alinhamento no *benchmark*, de forma similar a métrica SP do BAliBASE, e a matriz que gerou o melhor resultado para cada cenário registrada.

Para que esta informação pudesse ser utilizada efetivamente, era necessário buscar alguma medida que pudesse ser computada previamente a execução do alinhamento ou sua comparação ao *benchmark*. Foi escolhido para isto o tamanho do maior bloco de resíduos idêntico compartilhado entre as duas sequências.

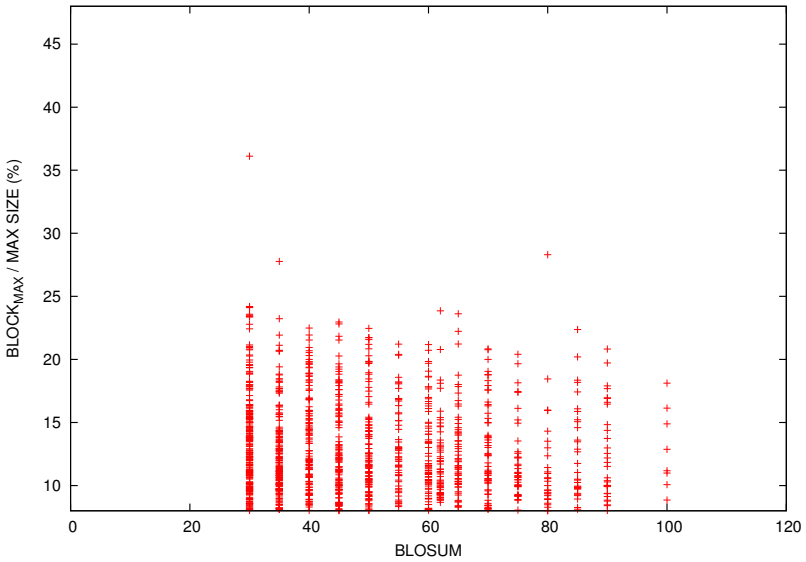
Para buscar uma correlação desta métrica com a matriz utilizada, é necessário que todas as medidas sejam tratadas de forma relativa. Por isso, a razão entre o tamanho destes blocos em relação ao tamanho de cada uma das sequências foi utilizado. Os gráficos de espalhamento apresentados na Figura 3.3 foram criados para auxiliar na visualização de uma possível correlação existente entre estas medidas.

Pode-se notar que não existe uma correlação geral entre esta métrica e a matriz utilizada. Talvez uma análise mais detalhada, tentando identificar *clusters* e correlações entre eles, poderia encontrar alguma correlação (mesmo que fraca).

Além disto, considerando a maneira como as matrizes BLOSUM são construídas e o fato das sequências do conjunto RV12 compartilharem entre 20 e 40% de similaridade, a dispersão de melhor resultado encontrada está, pelo menos em parte, em desacordo com o esperado, visto que o melhor resultado para muitos deles estão associados com matrizes



(a)



(b)

Figura 3.3: Dispersão da razão entre o maior bloco conservado $BLOCK_{MAX}$ e o tamanho das sequências em relação a matriz BLOSUM com melhor resultado de alinhamento par a par comparado ao *benchmark* do BAliBASE. A figura (a) indica a dispersão em relação a menor das duas sequências e a figura (b) em relação a maior.

para graus de similaridade muito maiores.

3.4 Função de aptidão baseada em estrutura

Outro teste realizado foi a tentativa de utilizar dados estruturais das sequências a serem alinhadas. O uso de estruturas secundárias para MSA é uma abordagem utilizada em algumas ferramentas, como o PROMALS [91] e o PROMALS3D [92], descritos em maiores detalhes na Seção 1.3.3.

Para obter informações sobre as estruturas secundárias dos casos de teste utilizou-se a ferramenta PSIPRED [57], por ser uma ferramenta largamente utilizada e conhecida, e por ter sido utilizada com propósito similar como parte da solução PROMALS.

As sequências dos casos de teste utilizados foram separadas em arquivos independentes e processadas pelo PSIPRED. As saídas deste processo foram armazenadas para uso do ALGAe como uma base de estruturas secundárias. Optou-se por este pré-processamento independente da execução do alinhamento pois o teste desejado era em relação a melhoria de qualidade dos resultados. Desta forma, a contabilização do tempo de processamento do PSIPRED não é relevante neste estágio, o que seria caso desejássemos contabilidar o desempenho deste processo como um todo.

O PSIPRED retorna como resultado um arquivo indicando, para cada resíduo da sequência de entrada, a qual estrutura secundária ele provavelmente pertence, sendo as estruturas possíveis: α-hélices, β-folhas e *coils*. As *coils* (ou *random coils*) não são estruturas secundárias no sentido estrito, sendo mais uma classe de conformações que indicam que dado resíduo não faz parte de uma estrutura secundária bem definida. Além desta informação também é provido um fator que indica a confiabilidade, segundo o algoritmo, da predição de estrutura estar correta (variando entre 0 e 9).

3.4.1 Metodologia e resultados

Uma função de aptidão, baseada na soma de pares com afinidade de *gaps* descrita na Seção 2.5.2 foi criada levando em conta as predições de estrutura secundária das sequências envolvidas no alinhamento. Esta função pode ser definida como na Equação 3.5.

$$fitness = \sum_{i=1}^{m} \sum_{j=1}^{n-1} \sum_{k=j+1}^{n} \sigma(seq_j^i, seq_k^i) + \sigma'(seq_j^i, seq_k^i) \tag{3.5}$$

Nesta equação, σ' indica a função de ajuste feita sobre o alinhamento de dois resíduos, considerando as estruturas secundárias às quais pertencem e o grau de confiabilidade da predição desta estrutura estar correta e pode ser descrita como na Equação 3.6.

$$\sigma'(seq_j^i, seq_k^i) = \pi \times \mu(seq_j^i, seq_k^i) \times bias(seq_j^i, seq_k^i) \tag{3.6}$$

Onde π indica um fator de proporcionalidade entre a pontuação de alinhamento dos

resíduos e do alinhamento de estruturas, de forma a equilibrar a participação de cada um na pontuação final (o valor utilizado foi $\pi = 2$). A função μ é um fator devido a confiabilidade da predição das estruturas e é calculado sobre a média de confiabilidade dos resíduos, de forma percentual (onde uma confiabilidade 0 indica 0% e uma confiabilidade 9 indica 100%). Por último, o fator *bias* apresenta como esta pontuação deve ser aplicada no caso de *matches* ou *mismatches* de estruturas, de acordo com as seguintes regras:

- $bias = 0$ se os dois resíduos estiverem em um *coil*;
- $bias = -0,5$ se apenas um dos dois resíduos estiver em um *coil*;
- $bias = 1$ se os dois resíduos participam da mesma estrutura secundária, seja ela uma α-hélice ou uma β-folha;
- $bias = -1$ se os dois resíduos participam de estruturas secundárias distintas (uma α-hélice ou uma β-folha)

Para este teste foram escolhidos os conjuntos RV11, RV12 e RV13 do BAliBASE versão 2.0. A escolha desta versão e conjuntos, e não da versão 3.0 foi devido ao fato de existirem dados disponíveis sobre as execuções do SAGA [82], outra solução baseada em algoritmos genéticos para MSA. Com isto haveria um benchmark para comparação do ALGAe caso os resultados fossem promissores. Os conjuntos escolhidos possuem sequências equidistantes, com diversos graus de similaridade, sendo os menores para o RV11 e os maiores para o RV13.

Inicialmente alguns testes foram realizados sobre a função de média do grau de confiabilidade μ, utilizando médias aritmética, geométrica e harmônica, conforme as Equações 3.7, 3.8 e 3.9 respectivamente. Como as variações não foram significativas, apenas a média aritmética foi mantida.

$$\mu(x, y) = \frac{x + y}{2} \tag{3.7}$$

$$\mu(x, y) = \sqrt{xy} \tag{3.8}$$

$$\mu(x, y) = \frac{2(xy)}{x + y} \tag{3.9}$$

Foram realizadas 20 execuções independentes para cada um dos cenário testados, tanto utilizando a função de soma de pares com função afim para *gaps* quanto a versão ajustada para considerar estruturas secundárias. Os resultados podem ser vistos nas Tabelas 3.3, 3.4 e 3.5.

Tabela 3.3: Resultados dos testes da função de aptidão baseada em estrutura para o conjunto RV11 do BAliBASE versão 2.0. São apresentados os resultados da execução utilizando esta função, a função padrão (soma de pares com função afim para penalização dos *gaps* e o comparativo, em pontos percentuais, entre as duas ($\frac{Score_{SecondaryStructure}}{Score_{Default}} \times 100$). Valores acima de 100 indicam que a função baseada em estruturas secundárias superou a originalmente utilizada.

	Estrutura		**Padrão**		**Comparativo**	
Entrada	**SP**	**TC**	**SP**	**TC**	**% SP**	**% TC**
1aab	76,10	65,80	82,30	73,70	92,47	89,28
1aboA	60,50	26,20	61,50	32,80	98,37	79,88
1aho	89,80	82,80	88,70	80,50	101,24	102,86
1csp	90,90	83,20	93,90	88,80	96,81	93,69
1csy	72,70	61,90	74,80	63,40	97,19	97,63
1dox	89,90	82,10	92,30	85,60	97,40	95,91
1fjlA	59,40	00,40	94,40	85,10	62,92	0,47
1fkj	89,80	77,70	92,30	84,70	97,29	91,74
1fmb	83,60	75,00	83,30	73,00	100,36	102,74
1hfh	81,10	65,00	83,90	71,70	96,66	90,66
1hpi	68,00	50,40	69,00	49,50	98,55	101,82
1idy	25,00	0,40	26,30	6,3	95,06	6,35
1krn	88,10	82,90	88,70	86,90	99,32	95,40
1pfc	78,20	63,10	76,60	60,30	102,90	104,64
1plc	85,40	68,60	86,80	71,60	98,39	95,81
1r69	22,40	1,10	35,30	12,20	63,46	9,02
1tgxA	65,50	44,90	72,90	55,90	89,85	80,32
1tvxA	16,40	1,00	24,10	11,50	68,05	8,70
1ubi	25,60	15,50	37,80	25,70	67,72	60,31
1wit	64,20	38,50	66,40	46,20	96,69	83,33
2fxb	97,00	92,00	97,00	92,00	100,00	100,00
2mhr	91,00	83,80	96,40	93,00	94,40	90,11
2trx	45,40	24,00	58,90	35,10	77,08	68,38
3cyr	76,00	63,00	70,50	53,10	107,80	118,64
451c	53,50	28,40	50,90	31,90	105,11	89,03
9rnt	95,80	90,20	97,20	92,60	98,56	97,41
Média	68,90	52,61	73,16	60,12	94,17	87,51

Os resultados obtidos com estes parâmetros e função objetivo baseada em estruturas secundárias foram, na grande maioria dos casos, inferiores aos obtidos com a função de soma de pares com função afim para *gaps*.

Apesar disto, esta abordagem é considerada promissora e precisaria ser estudada para

Tabela 3.4: Resultados dos testes da função de aptidão baseada em estrutura para o conjunto RV12 do BAliBASE versão 2.0. São apresentados os resultados da execução utilizando esta função, a função padrão (soma de pares com função afim para penalização dos *gaps* e o comparativo, em pontos percentuais, entre as duas ($\frac{Score_{SecondaryStructure}}{Score_{Default}} \times 100$). Valores acima de 100 indicam que a função baseada em estruturas secundárias superou a originalmente utilizada.

	Estrutura		**Padrão**		**Comparativo**	
Entrada	**SP**	**TC**	**SP**	**TC**	**% SP**	**% TC**
1ad2	80,00	71,20	82,70	74,70	96,74	95,31
1amk	94,50	89,00	96,30	92,40	98,13	96,32
1ar5A	87,60	78,20	87,20	78,10	100,46	100,13
1aym3	81,80	73,30	83,50	74,30	97,96	98,65
1bbt3	26,10	0,00	29,80	0,80	85,58	0,00
1ezm	93,80	86,70	94,60	88,10	99,15	98,41
1gdoA	68,80	57,30	74,00	64,70	92,97	88,56
1havA	9,70	0,20	20,70	9,00	46,86	2,22
1ldg	83,40	70,60	89,20	80,10	93,50	88,14
1led	82,80	74,50	84,90	77,60	97,53	96,01
1mrj	84,60	76,50	88,80	82,40	95,27	92,84
1pgtA	84,60	75,00	84,00	71,40	100,71	105,04
1pii	71,90	59,80	75,00	62,20	95,87	96,14
1ppn	92,60	87,70	93,90	90,40	98,62	97,01
1pysA	87,70	82,90	88,80	83,90	98,76	98,81
1sbp	29,00	10,60	30,70	13,00	94,46	81,54
1thm	88,00	77,30	88,70	78,80	99,21	98,10
1tis	88,90	83,90	91,50	87,10	97,16	96,33
1ton	66,60	51,10	71,30	55,30	93,41	92,41
1uky	29,10	7,70	49,60	20,30	58,67	37,93
1zin	84,80	75,20	89,40	80,40	94,85	93,53
2cba	62,00	40,00	61,90	37,80	100,16	105,82
2hsdA	41,50	15,10	47,60	23,70	87,18	63,71
2pia	42,70	27,50	46,70	31,10	91,43	88,42
3grs	30,60	10,10	33,60	10,40	91,07	97,12
5ptp	90,50	81,10	91,70	81,40	98,69	99,63
kinase	47,50	23,00	52,20	32,60	91,00	70,55
Média	67,82	55,02	71,42	58,59	94,96	93,90

um conjunto maior de valores para os parâmetros associados as informações estruturais. Eventualmente, com mais testes esta função pode ser otimizada e gerar resultados melhores.

Tabela 3.5: Resultados dos testes da função de aptidão baseada em estrutura para o conjunto RV13 do BAliBASE versão 2.0. São apresentados os resultados da execução utilizando esta função, a função padrão (soma de pares com função afim para penalização dos *gaps* e o comparativo , em pontos percentuais, entre as duas ($\frac{Score_{SecondaryStructure}}{Score_{Default}} \times 100$). Valores acima de 100 indicam que a função baseada em estruturas secundárias superou a originalmente utilizada.

	Estrutura		**Padrão**		**Comparativo**	
Entrada	**SP**	**TC**	**SP**	**TC**	**% SP**	**% TC**
1ac5	67,20	53,70	68,60	55,50	97,96	96,76
1ad3	93,70	90,00	94,10	90,70	99,57	99,23
1adj	87,20	78,90	92,00	86,50	94,78	91,21
1ajsA	13,90	2,10	21,00	6,90	66,19	30,43
1cpt	57,50	33,60	60,90	40,60	94,42	82,76
1dlc	76,30	60,90	78,50	64,20	97,20	94,86
1eft	73,30	56,90	79,90	67,00	91,74	84,93
1fieA	86,40	77,00	88,70	80,50	97,41	95,65
1gowA	59,10	41,10	69,40	56,60	85,16	72,61
1gpb	93,90	86,10	94,20	86,40	99,68	99,65
1gtr	87,60	77,00	89,60	80,90	97,77	95,18
1lcf	90,20	78,70	91,60	82,40	98,47	95,51
1lvl	34,00	3,20	35,00	4,90	97,14	65,31
1pamA	27,10	10,10	25,20	4,50	107,54	224,44
1ped	52,30	39,70	54,60	41,10	95,79	96,59
1pkm	82,60	73,30	85,20	77,70	96,95	94,34
1rthA	88,40	75,80	90,00	79,40	98,22	95,47
1sesA	81,20	64,80	86,60	74,80	93,76	86,63
1taq	82,50	73,80	82,90	74,50	99,52	99,06
2ack	61,70	42,90	61,50	41,90	100,33	102,39
2myr	12,50	0,10	13,30	0,00	93,98	∞
3pmg	92,80	88,80	93,60	89,60	99,15	99,11
4enl	33,80	20,40	33,90	20,10	99,71	101,49
actin	90,20	80,30	91,10	82,50	99,01	97,33
arp	75,90	63,30	77,10	65,80	98,44	96,20
gal4	22,6	4,20	24,90	7,60	90,76	55,26
glg	78,30	65,10	80,90	69,50	96,79	93,67
Média	66,75	53,40	69,05	56,74	96,67	94,11

3.5 Avaliação dos Resultados

Como pode ser visto, os resultados foram satisfatórios para o experimento com valores de abertura e extensão de blocos de *gaps* descrito na Seção 3.2. Porém, existem indicadores que valores únicos para qualquer cenário genérico podem não ser ideais.

Além disto, os experimentos com análise de melhor matriz par a par, descritos na Seção 3.3, e com o uso de função de aptidão baseada em estruturas secundárias, descrita na Seção 3.4, foram inconclusivos.

Devido a isto, nota-se que um ferramental para uma análise mais refinada e sistemática dos resultados se faz necessário. O Capítulo 4 propõe uma ferramenta para a comparação visual de um MSA contra um *benchmark* como possível solução para este problema.

Capítulo 4

Anubis

Os testes realizados com o ALGAe e apresentados no Capítulo 3 conseguiram identificar alguns pontos de melhoria. Porém, nota-se que um método sistemático para analisar os resultados é uma ferramenta essencial para que a avaliação e refinamento das tentativas de melhoria seja mais efetiva.

Devido a tal necessidade, foi desenvolvida uma ferramenta para visualizar de forma comparativa os alinhamentos múltiplos gerados em relação àqueles fornecidos como *benchmark*. Esta ferramenta permite uma análise que pode fornecer indicações dos pontos que estão influenciando negativamente ou positivamente no alinhamento que podem servir como fonte de correção de erros ou inspiração para novos experimentos e tentativas de melhoria.

Um dos primeiros passos para este desenvolvimento foi a pesquisa de outras aplicações com funcionalidades similares, apresentadas na Seção 1.5.2.

A ferramenta encontrada mais próxima dos objetivos almejados foi a SuiteMSA [8]. Porém o código desta é fortemente acoplado, dificultando a sua alteração ou inclusão de novas funcionalidades. Como necessitávamos funcionalidades específicas não oferecidas pela ferramenta esta não atendeu as necessidades do nosso projeto. Desta forma vislumbramos um ponto de melhoria importante que poderia ser um diferencial de uma nova ferramenta de visualização.

Baseado nisto, a ferramenta Anubis foi projetada e implementada no escopo deste projeto, de forma a servir como ferramenta auxiliar de análise para os resultados do ALGAe e como um aplicativo que poderia ser utilizado e expandido pela comunidade para satisfazer necessidades de outros projetos que trabalhem ou utilizem soluções de MSA.

Este capítulo se divide na Seção 4.1, que fornece uma visão geral sobre a ferramenta Anubis, na Seção 4.2, que apresenta os detalhes arquiteturais da solução proposta e na Seção 4.3, onde são apresentadas as funcionalidades implementadas no escopo deste pro-

jeto.

4.1 Visão Geral

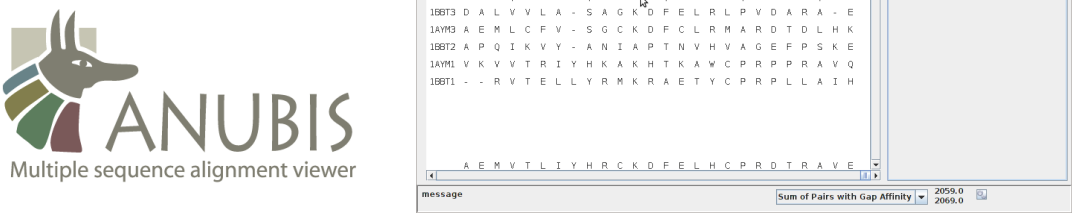
O Anubis é uma ferramenta para visualização e comparação de dois alinhamentos, tendo sido desenvolvido como ferramenta auxiliar ao ALGAe para analisar seus resultados e compará-los aos benchamarks, em especial do BAliBASE.

Esta ferramenta foi desenhada de forma a ser extensível e flexível, permitindo que funcionalidades pudessem ser incluídas ou removidas de acordo com as necessidades do usuário. Maiores detalhes sobre o funcionamento da ferramenta podem ser vistos na Seção 4.2.

A interface gráfica do Anubis pode ser vista na Figura 4.1. Esta interface tem como elementos centrais as visualizações dos alinhamentos de referência e o alinhamento alvo para avaliação. Tais visualizações permitem o uso de múltiplas camadas sobrepostas de informação, agrupadas pelo seu posicionamento nesta área (parte superior, esquerda, direita, inferior e central, que é dedicada ao alinhamento propriamente dito).

Estas informações podem ser ocultadas e novamente visualizadas de forma independente, sendo que para algumas existe também a possibilidade de controlar seu grau de transparência. Para este controle uma área da interface é dedicada a isto.

Por fim, existe também uma barra de status onde mensagens importantes ou controles mais utilizados podem ser adicionados. Tais áreas são apresentadas na Figura 4.2.



(a) (b)

Figura 4.1: Visão da interface principal do Anubis. A Figura (a) apresenta a *splash screen* da aplicação e o logotipo criado para a Aplicação. A Figura (b) mostra a interface principal da aplicação.

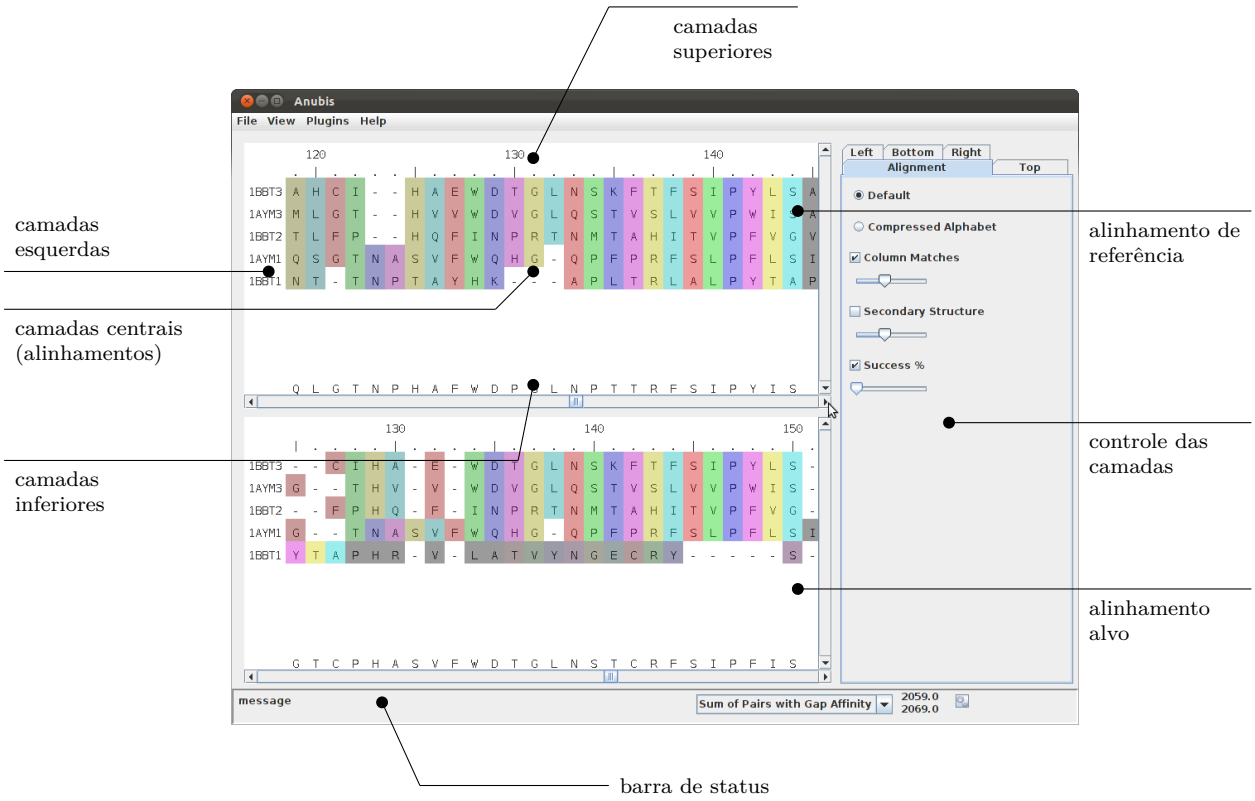


Figura 4.2: Visão dos principais elementos de interface presentes no Anubis.

4.2 Arquitetura e *design*

Esta seção descreve a arquitetura e *design* do sistema Anubis, assim como o racional para as decisões associadas. A Seção 4.2.1 apresenta os requisitos arquiteturais que nortearam as definições e decisões tomadas e a Seção 4.2.2 descreve a arquitetura de forma geral. As seções seguintes detalham mecanismos importantes que compõem esta arquitetura.

4.2.1 Requisitos Arquiteturais

Uma das principais necessidades deste projeto era a possibilidade de comparar visualmente, com o máximo de recursos auxiliares, os alinhamentos gerados pela ferramenta ALGAe contra aqueles fornecidos como benchmark, em especial os do BAliBASE [10,118, 119]. Portanto, a capacidade de processar os arquivos fornecidos por este era um requisito essencial. Da mesma maneira, a possibilidade de visualizar os dois alinhamentos de forma contrastante, com o máximo de informações disponíveis, era outra funcionalidade de grande valia.

Um dos principais pontos negativos encontrados nas ferramentas de visualização disponíveis era a dificuldade em expandi-las. Desta forma a característica diferencial que foi exercitada no aplicativo Anubis foi a sua flexibilidade e capacidade de expansão e customização.

Outro ponto ao qual foi dado grande importância no desenvolvimento foi a de visualizar de forma seletiva dados sobre os alinhamentos, de forma a facilitar análises e comparações. Sendo assim a possibilidade de mostrar ou esconder determinados dados foi priorizada.

4.2.2 Arquitetura de Alto-nível

Como um dos principais pontos desejados para o Anubis era a sua flexibilidade para a inclusão de novas funcionalidades. Para que isto fosse feito de forma estruturada foi trabalhada também a baixa coesão da solução, de forma que novas funcionalidades pudessem ser adicionadas ao *core* do sistema afetando-o de forma mínima. Isto permitiria que novas funcionalidades fossem testadas, adicionadas e removidas sem causar efeitos colaterais em outras funcionalidades.

Considerando isto, decidiu-se implementar a arquitetura do Anubis baseada em *plug-ins* que poderiam ser carregados pela aplicação em sua inicialização de forma dinâmica. Com isto novas funcionalidades não precisariam ser compiladas conjutamente com o *core* da aplicação. Além disto, os usuários da ferramenta poderiam escolher quais destes *plug-ins* lhes conviriam para seu uso específico, melhorando o uso de memória e reduzindo o ruído causado por excesso de informação.

Um *plug-in* do Anubis disponibiliza uma ou mais funcionalidades que, para serem acessados pelo resto do sistema, devem respeitar um conjunto de interfaces bem definido que categorizam e identificam estas funcionalidades. Mais detalhes sobre a implementação deste mecanismo pode ser visto na Seção 4.2.3.

Devido ao baixo acoplamento existente entre os *plug-ins* e os elementos principais do Anubis, um mecanismo de comunicação entre estes e o *core* do sistema ou mesmo entre dois *plug-ins* se faz necessário. Para resolver este problema, um mecanismo de troca de mensagens foi desenvolvido como parte do motor principal do Anubis. Tal mecanismo é descrito na Seção 4.2.4.

Para manter os dados e recursos compartilhados pela aplicação, foi desenvolvido um mecanismo para manipulação dos dados e recursos utilizados pela aplicação. Este mecanismo é descrito na Seção 4.2.5.

Por fim, outro mecanismo importante do Anubis é o de visualização de alinhamentos. Este mecanismo cuida de apresentar os dados relevantes ao usuário sem comprometer o desempenho da aplicação, o que poderia servir como um fator negativo na experiência de uso. Além disto este mecanismo trata da seleção de informações a serem visualizadas e

sua comparação. Os detalhes deste mecanismo se encontram na Seção 4.2.6.

Uma representação esquemática de como estes mecanismos se comunicam pode ser vista na Figura 4.3.

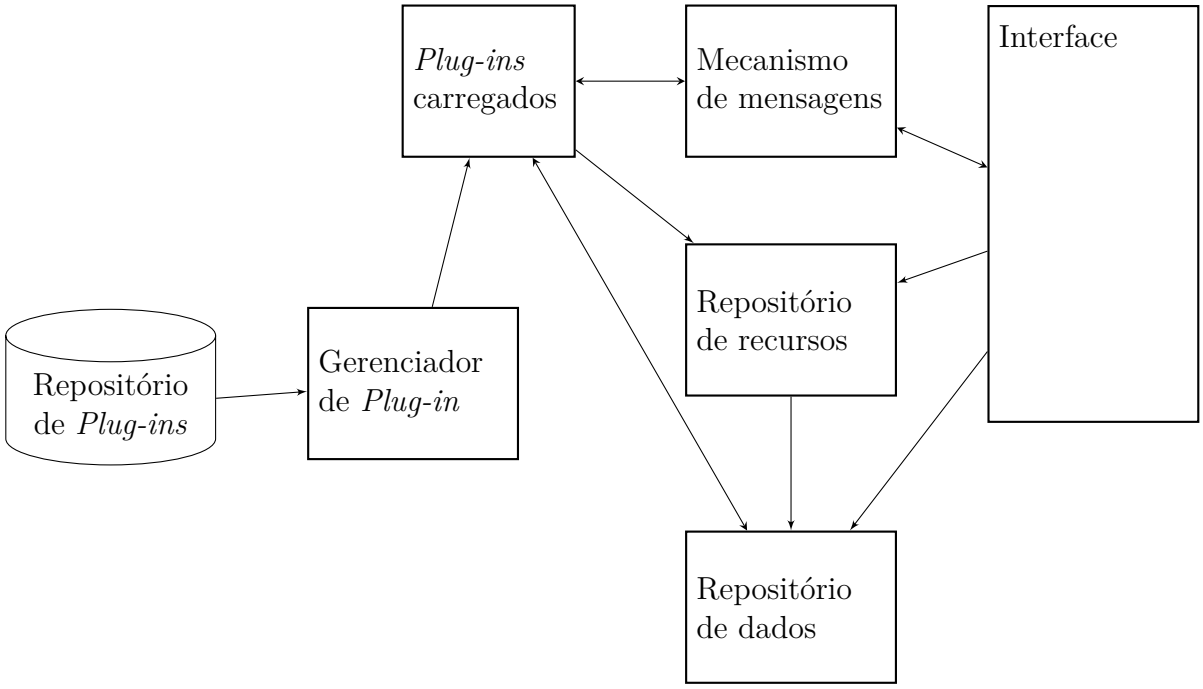


Figura 4.3: Visão geral da arquitetura do sistema Anubis, dos seus principais módulos e de como se comunicam.

Para manter os diversos módulos do sistema desacoplados foram utilizadas interfaces bem definidas para acesso a eles e um mecanismo de busca dos serviços ou recursos desejados através de um identificador único. Para esta identificação foram utilizados UUIDs (*universally unique identifiers*), números de 16 octetos (128 bits), pois são praticamente únicos. Tal fato dá uma garantia alta o suficiente de que dois itens distintos terão uma chance infinitesimal de possuirem o mesmo identificador, se gerados de forma independente.

Além disto, para permitir que os *plug-ins* sejam implementados e compilados de forma independente das funcionalidade principais, estas foram divididas em dois componentes principais: os mecanismos *core* e uma biblioteca contendo as interfaces e classes utilitárias necessárias para a comunicação entre os *plug-ins* e os mecanismos principais. Esta biblioteca é o único conjunto que precisa ser integrado a ambos, permitindo que o baixo acoplamento seja mantido. A visão destes componentes é ilustrado na Figura 4.4.

A linguagem escolhida para o desenvolvimento desta ferramenta foi Java pois, sendo uma linguagem orientada a objetos, permite que o baixo acoplamento desejado entre os

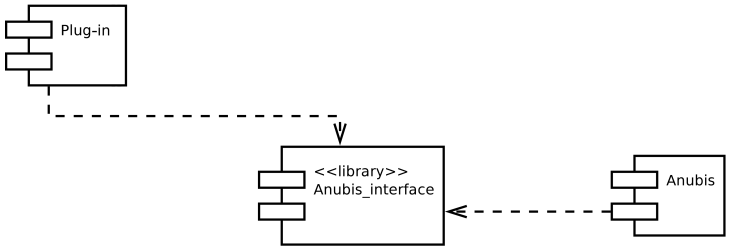


Figura 4.4: Diagrama de componentes representando as dependências entre eles e como foi mantido o baixo acoplamento entre os módulos *core* do Anubis e os componentes desenhados como *plug-ins*. Os elementos que permitem a comunicação entre ambos foram fatorados e mantidos na biblioteca *Anubis interface*.

módulos do sistema fosse facilmente alcançado. Além disso Java possui uma série de recursos que permitem a carga e execução dinâmica de código, algo que torna viável muitos dos requisitos aqui apresentados.

4.2.3 Mecanismo de *Plug-ins*

Um dos principais focos no desenvolvimento do Anubis foi em sua flexibilidade. Portanto um dos primeiros itens desenhados foi um mecanismo que permitisse a fácil extensão das funcionalidades desta aplicação. Um outro ponto que mereceu atenção neste item foi a possibilidade de que estas funcionalidades pudessem ser adicionadas ao sistema sem a necessidade de serem compiladas conjuntamente ao *core* do sistema, o que facilitaria a personalização e reduziria os efeitos colaterais que a adição de uma nova funcionalidade pudesse causar no restante do sistema.

Pensando nestes itens foi decidida a implementação de um mecanismo de *plug-ins* que pudessem ser adicionadas dinamicamente ao sistema de forma a manter o baixo acoplamento desejado para minimizar o impacto de uma funcionalidade sobre o resto do sistema.

O mecanismo desenhado permite que os plug-ins sejam implementados como pacotes binários (armazenados em arquivos JAR, visto que a linguagem e plataforma utilizada foi Java) que são depositados em uma pasta que serve como repositório de *plug-ins*. Ao iniciar, o Anubis consulta este diretório e dinamicamente carrega tais pacotes, agregando as funcionalidades ali definidas às suas.

Dois conceitos são importantes no funcionamento deste mecanismo:

- ***Plug-in*** - É um pacote que contém um conjunto de funcionalidades, preferencialmente coesas, que podem ser adicionadas ao conjunto de funcionalidades básicas do Anubis. Deve implementar a interface $IPlugin$.

- **Funcionalidade (*Feature*)** - É uma unidade que adiciona uma ação ou algum processamento sobre os alinhamentos. Deve implementar a interface $IFeature$.

O mecanismo inicialmente carrega os *plug-ins* de acordo com o seguinte processo:

1. Uma classe chamada $PluginManager$, principal responsável pelo processo, busca dentro do diretório configurado como repositório os pacotes (.jar) que contenham uma classe que implementa a interface $IPlugin$. Isto é feito com auxílio da classe $PluginFinder$.

2. A classe $PluginFinder$ integra todas as classes dos pacotes encontrados ao binário da aplicação, utilizando o $URLClassLoader$, uma classe provida pela plataforma Java para dinamicamente carregar classes dada a localização do arquivo que a contém.

3. Entre as classes carregadas, aquelas que descrevem os *plug-ins* (aquelas que implementavam a interface $IPlugin$) são devolvidas ao $PluginManager$ para que as funcionalidades contidas sejam integradas ao Anubis, de forma a serem utilizadas pelo resto do sistema.

Neste ponto, o código binário dos *plug-ins* estão integrados ao do Anubis, porém não são facilmente acessados por este, assim como todo o conjunto de funcionalides que estes provêem. Para que isto ocorra, o mecanismo executa um segundo processamento visando "publicar" as funcionalidades para uso no sistema, o gerenciamento de funcionalidades e sua ativação.

Um ponto importante é que determinada funcionalidade pode depender de outras, algumas vezes disponibilizadas até mesmo em um *plug-in* distinto. Para controlar esta rede de dependências, as funcionalidades foram implementadas podendo definir as interfaces que disponibilizam.

Uma interface define unicamente as funções providas ao restante do sistema assim como a semântica destas funções. Tal identificação permite que uma funcionalidade possa ser intercambiada por outra análoga, ou mesmo por uma versão atualizada dela mesma sem impacto nas funções que dependem dela, mantendo-as desacopladas.

Quando uma funcionalidade é desenvolvida, ela pode definir de quais interfaces ela depende. Baseado neste conceito, o gerenciamento e ativação das funcionalidades ocorre segundo o processo definido a seguir:

1. Considerando todos os *plug-ins* carregados, o motor do Anubis requisita a eles a lista de funcionalidades que os compõem. Esta lista é passada à classe $FeatureManager$, que gerencia esta parte do processo

2. A classe $FeatureManager$, baseada nas interfaces disponibilizadas pelas funcionalidades e na lista de dependências de outras interfaces também definidas nestas, cria um grafo orientado que representa toda a rede de dependências para as funcionalidades disponíveis. Para obter a interface provida por uma funcionalidade é utilizado o método $getInterfaceID()$ e para a obtenção das interfaces requeridas, o método $getRequiredInterfaces()$. Ambas são providas na interface $IFeature$.

3. Em seguida, uma ordenação topológica é realizada sobre este grafo. Se um ciclo é encontrado, existe uma dependência cíclica entre as funcionalidades. Isto inviabiliza o processo, notificando o usuário e abortando a execução. Caso contrário, o processo continua.

4. Em caso de sucesso, a classe $FeatureManager$ habilita todas as funcionalidades, respeitando a ordenação topológica realizada. Um método definido na interface $IFeature$ chamado *enable*() é disparado neste processo, permitindo que cada funcionalidade execute seu *preset* no evento de sua ativação.

As principais classes envolvidas no processo podem ser vistas no diagrama apresentado na Figura 4.5. O diagrama de sequência apresentado na Figura 4.6 ilustra como tais classes interagem nos processo de carga de *plug-ins* e no posterior gerenciamento e ativação de funcionalidades.

4.2.4 Mecanismo de Trocas de Mensagem

O mecanismo de *plug-ins* descrito na Seção 4.2.3 permite que o sistema seja flexível e personalizável, mantendo um baixo acoplamento entre funcionalidades externas e os mecanismos principais do Anubis.

Porém, devido ao mesmo baixo acoplamento um novo problema surge: como permitir que os componentes se comuniquem ou notifiquem outros? Para resolver este problema foi desenvolvido também um mecanismo de comunicação baseado em mensagens.

O mecanismo desenvolvido para o Anubis se baseia em um padrão de projeto (*design pattern*) conhecido como *Observer/Observable* [39], onde um objeto se registra para receber informações de um objeto de interesse e este último, no caso de um evento relevante, notifica os objetos interessados.

No caso específico do Anubis, um componente se registrar para receber um tipo de evento, o que é dado pela classe que o define. Toda classe que representa um evento deve implementar a interface $IEvent$.

Quando alguém deseja notificar algo ao sistema ele o faz através do componente $EventManager$, que enfileira o evento a ser notificado e avisa cada um dos interessados

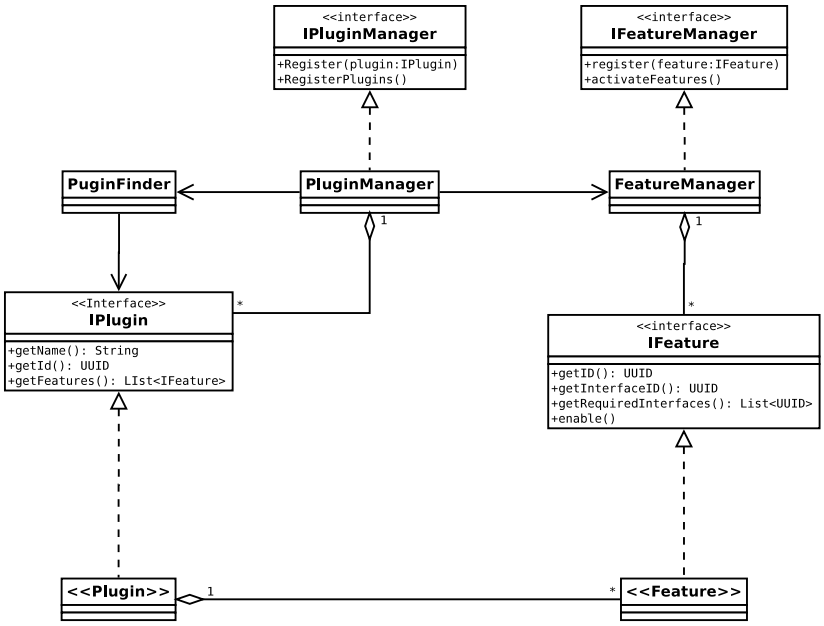


Figura 4.5: Diagrama de classes apresentando as principais classes e interfaces que compõe o mecanismo de gerenciamento de *plug-ins* do Anubis.

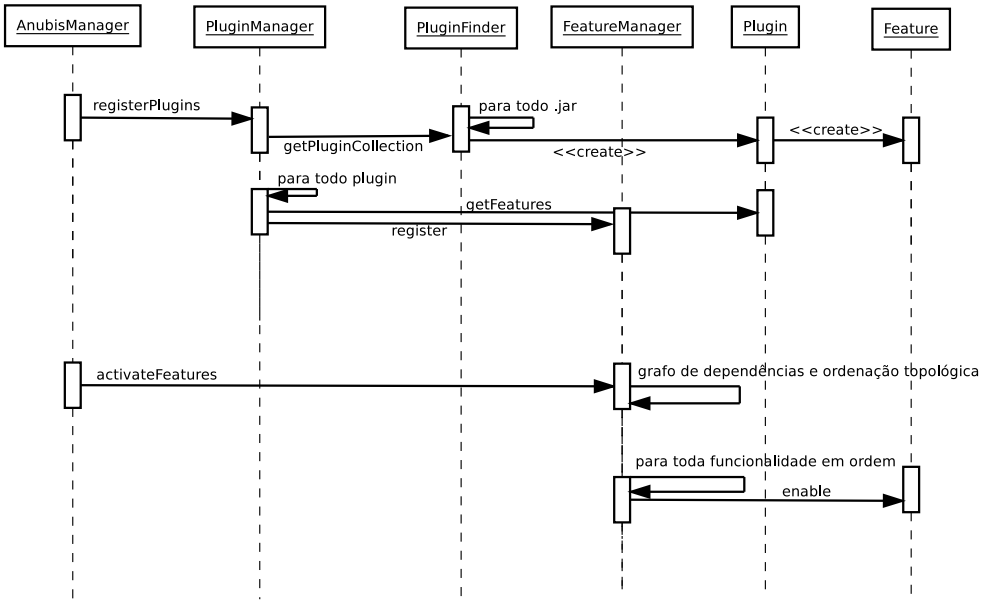


Figura 4.6: Diagrama de sequência apresentando como os principais elementos do mecanismo de *plug-ins* interagem no processo de carregamento de *plug-ins* e posterior ativação das funcionalidades neles contidas.

a ele de forma assíncrona. Para que um objeto possa se registrar para receber uma mensagem ele deve implementar a interface $IEventHandler$, que garante ao $EventManager$ que este responderá ao método $eventRaised$, onde ele pode tratar um evento que se registrou para receber.

Desta forma, componentes que não se conhecem diretamente podem se comunicar através de um evento mutuamente conhecido, garantindo que, mesmo com baixo acoplamento, seja possível que operem conjuntamente.

A Figura 4.7 apresenta as classes principais envolvidas neste mecanismo e a Figura 4.8 ilustra, através de um diagrama de sequência, o processo descrito anteriormente para o registro e notificação de um evento.

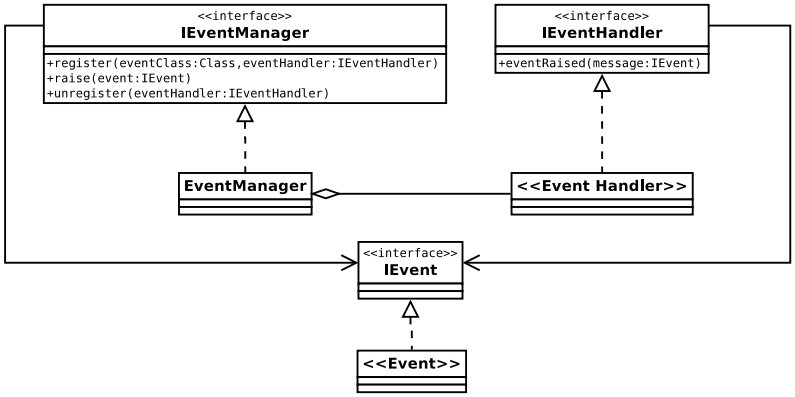


Figura 4.7: Diagrama de classes apresentando as principais classes e interfaces que compõem o mecanismo de troca de mensagens do Anubis.

4.2.5 Mecanismo de Manipulação de Dados e Recursos

Para manter os dados que são carregados ou processados para apresentação no Anubis foi desenhado um padrão para tratá-los de maneira uniforme.

Os dados são gerenciados pelos chamados repositórios de dados, que implementam a interface $IDataRepository$. Estes repositórios são responsáveis pelo armazenamento e gerenciamento dos dados como um todo (por exemplo, toda informação de um dado tipo associada a ambos os alinhamentos analisados).

A carga ou processamento associados aos repositórios de dados é, via de regra, iniciado com um evento externo, seja gerado pelo usuário (por exemplo, o carregamento de um arquivo que descreve um alinhamento) ou pelo próprio sistema (quando, por exemplo, um outro repositório de dados do qual este depende termina sua carga ou processamento).

Seguindo este evento, o repositório de dados processa e armazena os dados relevantes e

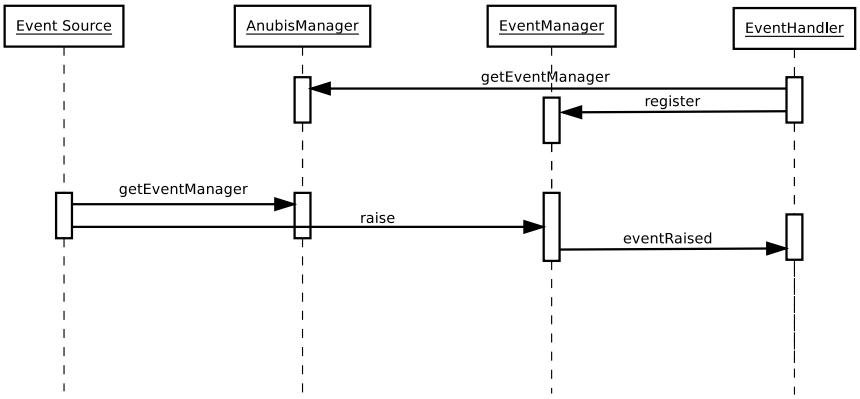


Figura 4.8: Diagrama de sequência apresentando como os principais elementos do mecanismo de mensageria interagem no processo registrar interesse em dado evento e, em seguida, de sua notificação.

notifica a qualquer interessado sua disponibilidade. Esta notificação utiliza o mecanismo de mensagens descrito na Seção 4.2.4.

Tal mecanismo, porém, é limitado no que tange tratar apenas porções desta informação de uma maneira genérica. A criação de um mecanismo mais específico permite tratar de forma mais simples funcionalidades como ocultar colunas ou linhas dos alinhamentos. Além disto, tal mecanismo permite um nível extra de controle para mesclar informações de múltiplos repositórios de dados sem adicionar complexidade nas classes que lidarão com a visualização destas.

O mecanismo desenvolvido para este fim foi chamado de conectores de dados (*DataConnectors*) e proporciona, além das funcionalidades apresentadas anteriormente, um mediador entre a camada de visualização e os repositórios de dados. Sendo assim sempre que novas informações são necessárias na visualização estas são requeridas ao conector de dados responsável que interfaceia com o repositório de dados (ou os repositórios, caso aplicável) para obtê-la e prepará-la.

Para manter o baixo acoplamento desejado, todos os repositórios e conectores de dados se registram em um repositório de recursos (a classe *ResourceManager*), através de identificadores únicos (UUIDs). Os interessados neste componentes requisitam a este repositório, através deste identificador único, o objeto nele registrado e, com isto, podem iniciar sua interação com ele. Tal mecanismo é expandido para aceitar o registro de qualquer recurso dinâmico utilizado no sistema, como por exemplo, elementos de interface com o usuário.

4.2.6 Mecanismo de Visualização

A visualização e comparação de dados é o objetivo principal para o desenvolvimento do Anubis. Logo, todas as outras funcionalidades servem a esta.

Os principais componentes deste mecanismo são os visualizadores de dados. Tais componentes, ao serem notificados de uma mudança que lhe é de interesse (como aquela geradas pela disponibilização de dados de um repositório de dados, como descrito na Seção 4.2.5), requisitam os dados relevantes a um conector de dados e as apresentam ao usuário. Outros tipos de mudanças que podem interessar aos visualizadores são aquelas que ocorrem mediante uma ação do usuário como através da alteração de uma barra de *scroll* ou mesmo um evento interno do sistema.

Os conectores de dados, além das funcionalidades descritas na Seção 4.2.5, servem também para melhorar o tempo de resposta e eficiência da interface ao permitir a carga apenas da fração relevante dos dados que devem ser apresentados. Desta forma servem também como um mecanismo para melhorar a experiência do usuário (UX, do inglês *user experience*).

A interação entre os componentes de manipulação de dados e de visualização pode ser vista na Figura 4.9.

Este mecanismo possui diversas classes, responsáveis por diversos papéis na visualização dos dados. A Figura 4.10 apresenta uma visão esquemática das principais classes envolvidas.

A classe $AnubisView$ é o elemento central deste mecanismo. É ela que agrega os outros componentes e serve como *Façade* para a comunicação com o resto do sistema.

Um dos principais elementos da qual a $AnubisView$ se compõe é o conjunto de objetos da classe $AlignmentViewer$ que apresentam as diversas formas de visualização dos alinhamentos.

Cada $AlignmentViewer$ possui o elemento gráfico $AlignmentViewerPanel$, responsável pela parte gráfica propriamente dita. Além disto, para separar a lógica de visualização da lógica de como desenhar a informação, esta parte foi delegada a uma interface distinta, $IAlignmentDrawingStrategy$. Esta implementação é inspirada no padrão de projeto *Strategy*, que permite que uma lógica de desenho distinta possa ser aplicada caso isto se torne relevante. No momento, porém, a única implementação disponível é a classe $AlignmentDrawer$.

O $AlignmentViewer$ permite a visualização de diversas camadas sobrepostas, para que múltiplas informações ou pontos de vista possam ser comparados. A lista destas camadas é controlada por uma implementação da interface $IAlignmentLayerSet$, que possui tanto camadas gráficas (que implementam a interface $ILayer$) ou que apresentam dados textuais ($ITextLayer$). Múltiplas camadas gráficas podem ser apresentadas ao mesmo tempo, mas apenas uma camada textual. Esta decisão foi tomada para evitar a

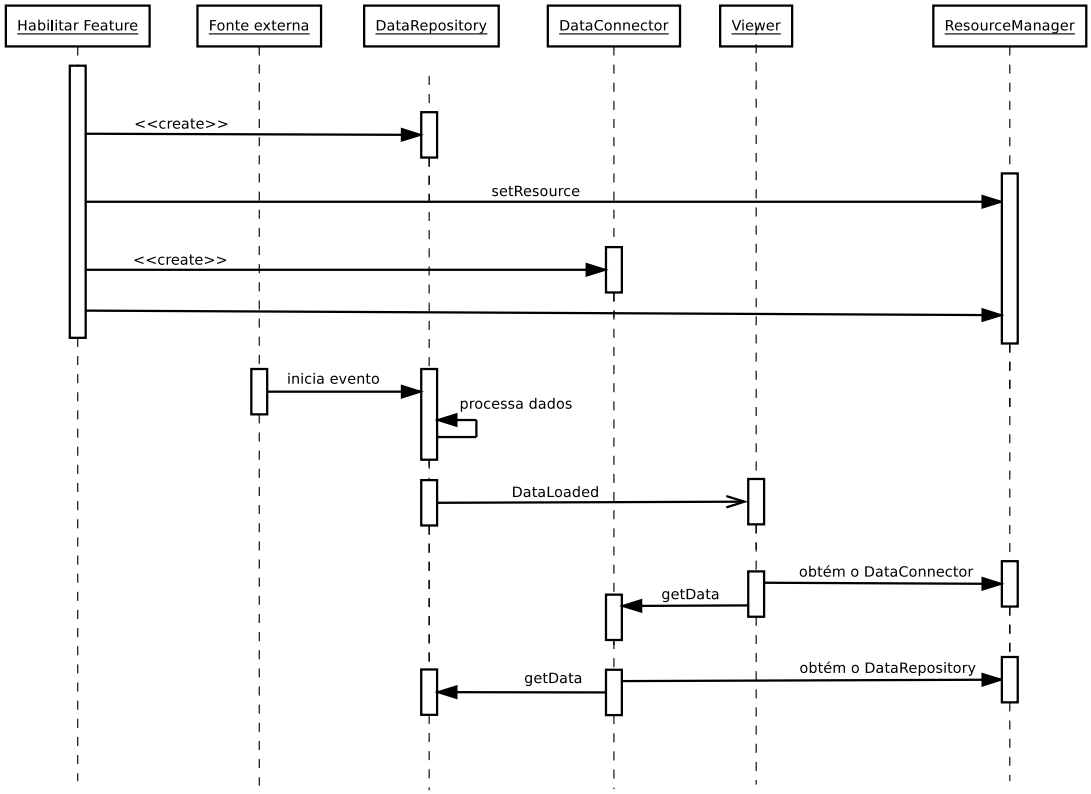


Figura 4.9: Diagrama de sequência apresentando como os principais elementos dos mecanismos de manipulação e visualização de dados interagem, de forma geral, no sistema Anubis.

confusão que a sobreposição de informações textuais causaria.

Para facilitar a visualização das camadas, um grau de transparência pode ser aplicado, de forma independente, para cada um deles. Além disto uma camada pode ser ocultada ou apresentada. As camadas selecionadas para apresentação ou ocultação são mantidas pela classe *LayerSelectionList*.

Outro elemento importante é o menu. Este pode ser composto por objetos da classe $MenuItemDefinition$ que possui as meta informações para a construção deste (texto do item de menu, prioridade para ordenação e lista de sub-menus) e pode possuir um objeto do tipo $MenuAction$, que seguindo o padrão de projeto *Command* permite executar alguma ação no sistema. Como esta ação pode depender de parâmetros dinâmicos esta classe implementa a interface $IParameterizedContext$ que permite defini-los dinamicamente mediante um conjunto chave-valor, como uma *hashtable*.

Da mesma forma que a interface $IParameterizedContext$ permite definir contextos para execuções específicas, a classe $Session$ permite manter parâmetros de escopo global

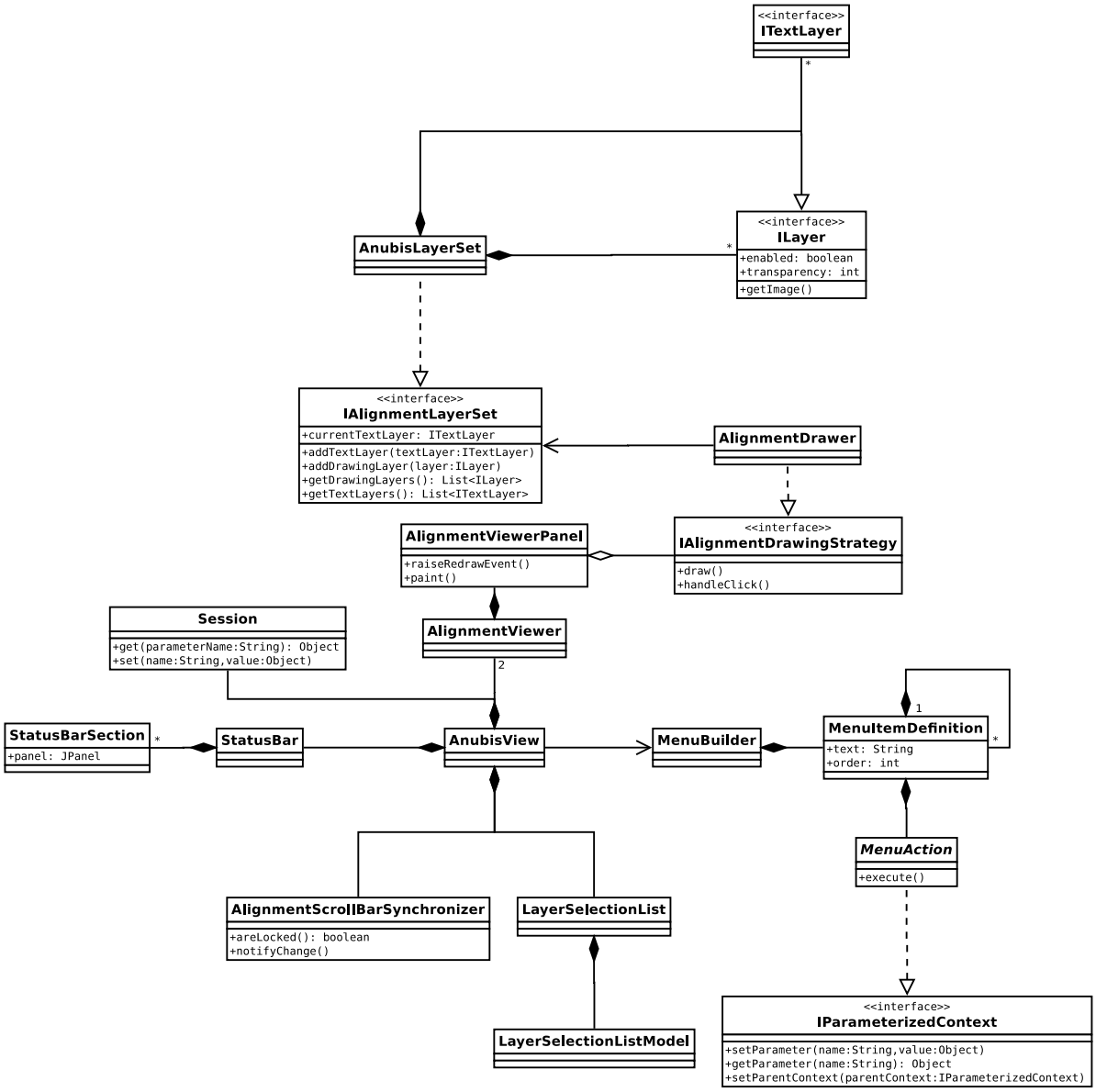


Figura 4.10: Diagrama de classes com os principais elementos do mecanismo de visualização de dados do Anubis.

também através de um conjunto chave-valor.

Por fim, um último componente que é importante no sistema é a barra de status, definida na classe *StatusBar*. Componentes do Anubis podem incluir seções nesta barra, utilizando a classe *StatusBarSection*. Cada seção pode apresentar informações de aviso ou mesmo permitir interação do usuário se necessário.

4.3 Funcionalidades Principais

Esta seção descreve as principais funcionalidades desenvolvidas para o Anubis, respeitando o modelo arquitetural apresentado na Seção 4.2.

4.3.1 Visualização de Alinhamentos

Como o principal objetivo do Anubis é a apresentação e comparação de alinhamentos múltiplos, a primeira funcionalidade desenvolvida após os mecanismos essenciais foi o visualizador destes.

O visualizador de MSA utiliza a estrutura matricial para representá-lo, sendo que cada linha representa uma sequência e as colunas um conjunto alinhado de resíduos (ou *gaps*). Este visualizador também possui a funcionalidade de apresentar dados em múltiplas camadas, permitindo a sobreposição de dados a serem comparados. Os alinhamentos, caso contenham muitas colunas ou sequências, são vistos parcialmente, sendo que a parte visualizada pode ser alterada através de *scroll bars* horizontais e verticais. Existe ainda a funcionalidade de "travar" ambos os alinhamentos de forma que suas visualizações sejam deslocadas em sincronia.

Este visualizador pode conter várias camadas textuais (representando os resíduos, por exemplo), mas permite a visualização de apenas uma por vez, por motivos de clareza da informação apresentada.

Também é permitida a existência de diversas camadas "gráficas" que, ao contrário das textuais, podem ser apresentadas simultaneamente. Para facilitar a visualização e comparação destas camadas, elas podem possuir graus diferentes de transparência controladas pelo usuário. Também é permitido que cada uma destas camadas seja ocultada ou visualizada de forma independente.

4.3.2 Alfabetos Comprimidos

Alfabetos comprimidos são partições criadas sobre as 20 letras que representam os amino ácidos. Estes alfabetos agrupam algumas destas letras, de forma a desconsiderar diferenças menores entre estes, formando grupos mais genéricos.

Como o uso dos alfabetos comprimidos é uma boa ferramenta para analisar homologias em sequências de proteínas, um *plug-in* para tratar esta informação é de grande valia para uma ferramenta como o Anubis. Desta forma, todos os alfabetos descritos no trabalho de Edgar sobre este assunto forma implementados [32].

Esta funcionalidade foi implementada como uma camada de texto (vide Seção 4.1 e Seção 4.2.6 para maiores informações). Desta forma, o usuário pode escolher entre visualizar os resíduos como definidos na sequência original ou sua representação de acordo

com um dos alfabetos comprimidos disponibilizados. O usuário tem ainda a possibilidade de trocar a qualquer momento a representação utilizada.

4.3.3 Correspondência de Colunas

Foram implementados algumas funcionalidades que permitem a comparação visual das correspondências entre os resíduos do alinhamento alvo e do *benchmark*.

O primeiro e mais simples deles: um valor é associado a cada coluna do alinhamento *benchmark* e este valor é atribuído a cada resíduo alinhado nesta coluna (os *gaps* não são considerados).

Isto cria uma função $col_b(\ll s, r \gg)$ onde s representa o índice de uma sequência e r o r-ésimo resíduo desta sequência (antes de ser alinhada, ou seja, sem considerar os *gaps*). O valor de col_b é o valor associado a cada coluna do alinhamento, ou seja, a coluna que o r-ésimo resíduo da sequência s ocupa no alinhamento.

Como o alinhamento alvo possui o mesmo conjunto de sequências e resíduos do *benchmark*, é possível associar a mesma tupla $\ll s, r \gg$ o valor de col_b. Porém, é importante notar que este mesmo resíduo ocupará uma posição col_t que possivelmente não será a mesma representada por col_b.

Para representar isto, associou-se uma cor a cada valor de col_b, que pode ser visualizado em ambos os alinhamentos. Como no *benchamrk* por definição toda coluna compartilha o mesmo valor, teremos blocos de mesma cor para todas as colunas deste alinhamento. Este mesmo comportamento não necessariamente se repete no alinhamento alvo, de forma que é possível ver onde este está "desalinhado" quando comparado ao *benchmark*. Isto é ilustrado na Figura 4.11.

Baseado nesta funcionalidade, uma outra foi implementada. Esta ferramenta considera a porcentagem de sucesso por resíduo ou por sequência.

A porcentagem de sucesso por resíduo considera a fração de resíduos que se alinham com ele tanto no *benchmark* quanto no alinhamento alvo sobre o número de resíduos alinhados com ele no *benchmark*. Isto pode ser visto nas equações 4.1, 4.2 e 4.3.

Nestas equações $res_a(c, s)$ representa o resíduo que está na c-ésima coluna da s-ésima sequência do alinhamento a, caso exista um resíduo nesta posição. Nestas equações o alinhamento b representa o *benchmark* e t o alinhamento alvo (*target*).

$$Match(col, s_1, s_2) = \begin{cases} 1 & \text{se } col = col_t(s_2, r_2) \\ 0 & \text{caso contrário} \end{cases} \tag{4.1}$$

Onde $r_1 = res_t(col, s_1)$, $c_b = col_b(s_1, r_1)$ e $r_2 = res_b(c_b, s_2)$.

$$NumRes(col) = \sum_{i=1}^{n} \begin{cases} 1 & \text{se } \exists res_b(col, i) \\ 0 & \text{caso contrário} \end{cases} \tag{4.2}$$

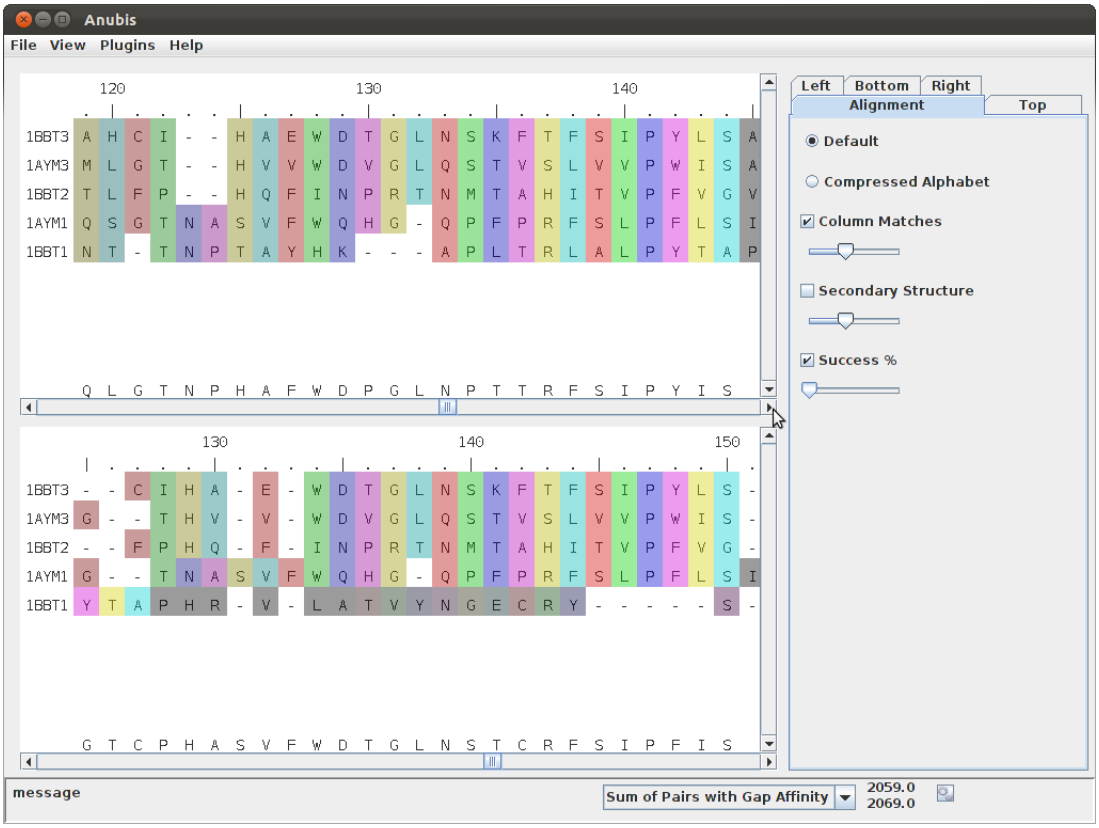


Figura 4.11: Interface do Anubis apresentando a correspondência de colunas entre dois alinhamentos. É possível notar que a grande maioria das sequências, com exceção da 1BBT1, está alinhada de forma idêntica no trecho que vai da coluna 128 a 144 no *benchmark* e no trecho que vai da coluna 134 a 150 no alinhamento alvo.

$$Success(col, row) = \frac{\sum_{i=1}^{n} Match(col, row, i)}{NumRes(col)} \tag{4.3}$$

Esta função foi estendida para considerar também o alinhamento de *gaps*. Para isto, quando um *gap* se alinha com um resíduo tanto no *benchmark* quanto no alvo ele é considerado. Neste caso $NumRes(col) = n$, onde n indica o número se sequências.

A métrica de sucesso baseada em sequências considera o número de resíduos que se alinham de forma correspondente em ambos os alinhamentos em relação àquela sequência. E, da mesma forma que feito com a porcentagem de sucesso por resíduos, esta métrica foi estendida para considerar os *gaps*.

4.3.4 Sequência Consenso

A sequência consenso, como o próprio nome diz, representa o consenso entre todas as sequências que formam um MSA. É a sequência formada pelos caracteres que maximizam a pontuação, coluna a coluna, em relação a todos os outros caracteres presentes naquela coluna.

A definição de consenso é dado pela sequência C que maximiza a Equação 4.4, onde $C[c]$ indica o caracter consenso na coluna c, A o alfabeto utilizado, $\alpha_i[c]$ o caracter presente na coluna c da i-ésima sequência, n o número de sequências do alinhamento e σ a função de pontuação utilizada.

$$C[c] = x \in A \mid \max Score[x] = \sum_{i=1}^{n} \sigma(x, \alpha_i[c]) \tag{4.4}$$

4.3.5 Árvores Filogenéticas

Alinhamentos múltiplos estão intrinsicamente relacionados a filogenia. As funções de pontuação representam as similaridades existentes devido a uma maior ou menor divergência entre as sequências, o que está, em grande parte das vezes, diretamente relacionado à distância evolutiva entre elas.

Devido a isto, árvores filogenéticas também são muitas vezes usadas como guias para diversos algoritmos de MSA.

Como consequência, visualizar e analisar a árvore filogenética é uma ferramenta muito importante para uma aplicação como o Anubis. A análise da árvore filogenética associada a um MSA pode auxiliar no entendimento da história evolutiva que as conecta e, no sentido contrário, a análise da árvore esperada permite também analisar possíveis pontos falhos no método utilizado para gerar o alinhamento.

Devido a estes fatos foi criado um *plug-in* que permitisse a visualização de árvores filogenéticas.

Foi utilizada para esta funcionalidade parte da aplicação Figtree [94], uma ferramenta de visualização de árvores. Esta aplicação foi componentizada e integrada ao Anubis através da estrutura de *plug-ins* de forma a manter a arquitetura definida (mais detalhes sobre ela podem ser vistos na Seção 4.2).

A escolha do Figtree deveu-se a uma série de fatores:

- Geração de árvore com boa qualidade gráfica e prontas a serem utilizadas em documentos;
- Geração de diversos tipos de árvore como: retilínea, polar e radial;

- As árvores podem ser geradas baseadas em diversos algoritmos, tais como UPGMA [107] e Neighbor Joining [98] ;. A geração destas árvores é feita pela biblioteca JEBL (*Java Evolutionary Biology Library*) [28].

- É uma ferramenta *open-source*, sobre licença GPL versão 2, o que permite o seu reaproveitamento;

- Foi desenvolvida utilizando as tecnologias Java e Swing da mesma forma que o Anubis, o que facilita a sua integração.

Um exemplo de como as árvores podem ser visualizadas é apresentado na Figura 4.12.

4.3.6 Estruturas Secundárias

As informações estruturais são importantes na melhoria da qualidade dos alinhamentos, sendo parte importante do algoritmo de ferramentas como o PROMALS [91] e o PROMALS3D [92]. O uso de estruturas secundárias também foi testado entre um dos métodos para melhoria do ALGAe, como apresentado na Seção 3.4. Além disto, MSAs são utilizados como ferramentas auxiliares para a predição de estruturas de proteínas ao compará-los a outras sequências cujas estruturas são bem conhecidas.

Uma funcionalidade para a visualização de estruturas secundárias de proteínas foi implementada para o Anubis, utilizando os dados da ferramenta PSIPRED [57].

Os arquivos gerados como saída desta ferramenta (*horiz-files*) são armazenados em um diretório configurável que serve como base de estruturas para o Anubis. Baseado no nome das sequências esta funcionalidade pode carregar as estruturas associadas e apresentá-las como uma camada no visualizador de sequências.

As α-hélices são representadas como cilindros verdes e as β-folhas como setas amarelas. Os *coils* não são representados por não estarem associados a uma estrutura secundária propriamente dita, desta forma sua apresentação geraria apenas poluição visual sobre os dados realmente importantes. Esta funcionalidade é ilustrada na Figura 4.13.

4.4 Análise do ALGAe

A ferramenta Anubis foi utilizada para analisar os resultados de algumas execuções do ALGAe. Nesta seção algumas destas análises são apresentadas.

As análises aqui apresentadas estão associadas aos testes com a função de aptidão baseada em estrutura, apresentados na Seção 3.4. O foco está principalmente nos cenários que tiveram os piores resultados.

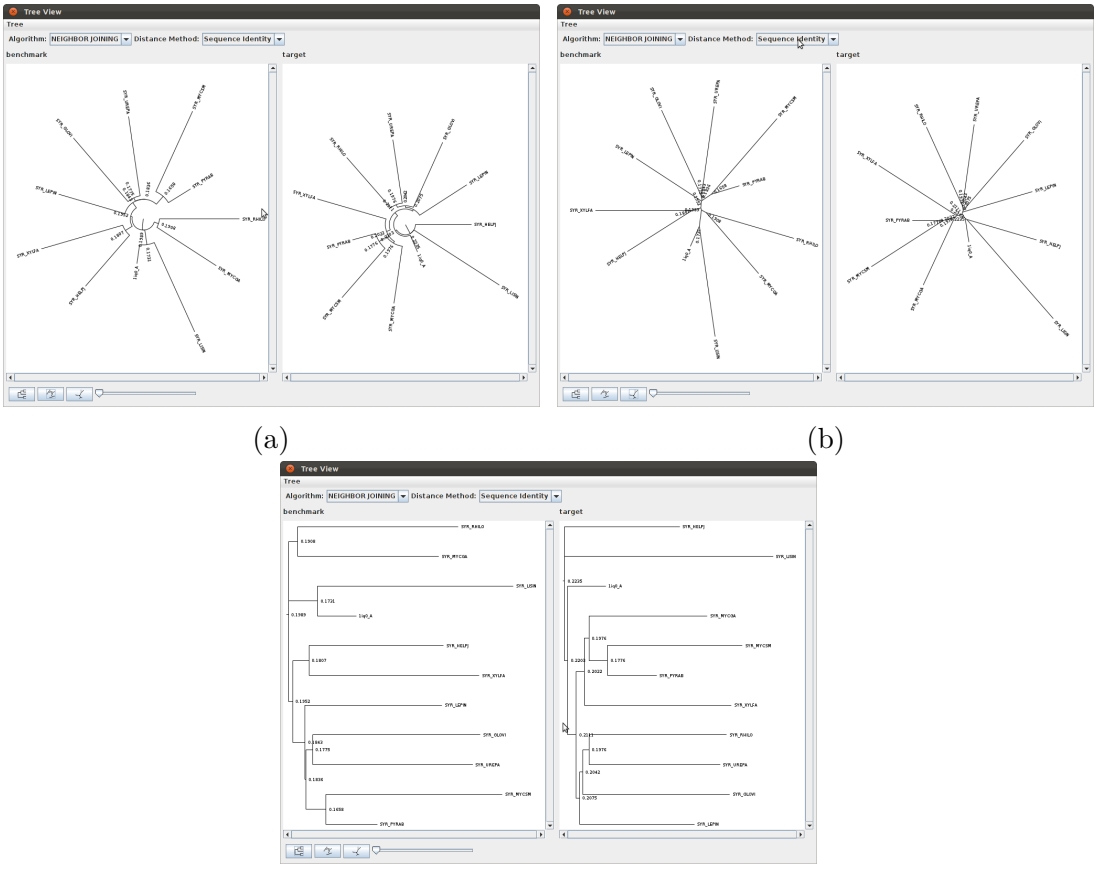


Figura 4.12: Exemplos de visualizadores de árvores implementados no Anubis. As três representações referem-se a comparação do cenário BB12044 do BAliBASE entre o *benchmark* (a esquerda) e um alinhamento gerado pelo ALGAe (a direita). A figura (a) apresenta a representação polar, a figura (b) a representação radial e a figura (c) a representação retilínea da árvore filogenética para estes alinhamentos, geradas através do algoritmo Neighbor Joining [98].

O primeiro cenário citado é o *GAL4*, pertencente ao conjunto RV13 do BAliBASE versão 2. Este cenário alinha sequências associadas a diversas proteínas do levedo (*Saccharomyces cerevisiae*).

A Figura 4.14 apresenta duas execuções do ALGAe para este cenário comparado, em termo de correspondência de colunas em um certo trecho, à referência do BAliBASE. Neste trecho a correspondência foi muito boa, com exceção da sequência *yb00_yeast*, associada à proteína TBS1 [22].

Podemos notar na Figura 4.15 como a árvore filogenética baseada no alinhamento de referência e a gerada pelo ALGAe refletem tal discrepância: existe uma similaridade

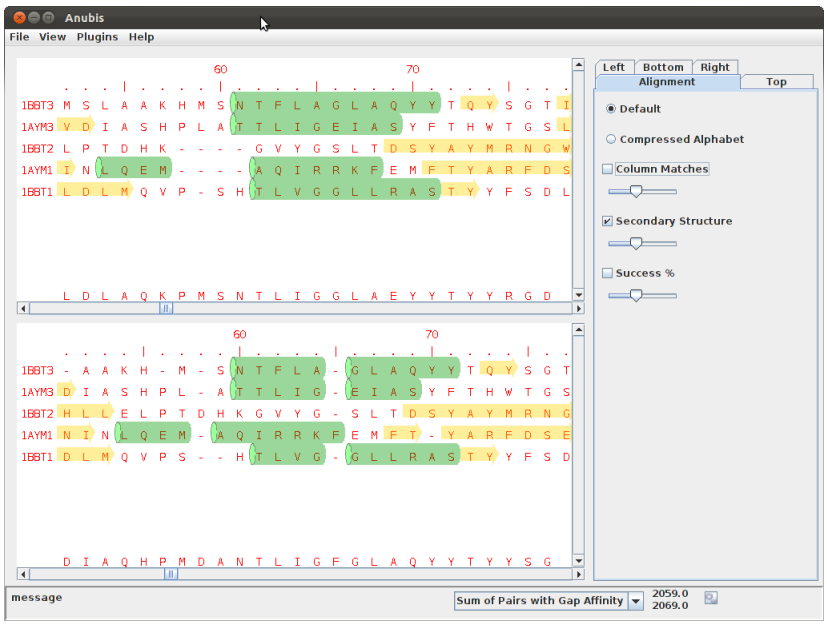
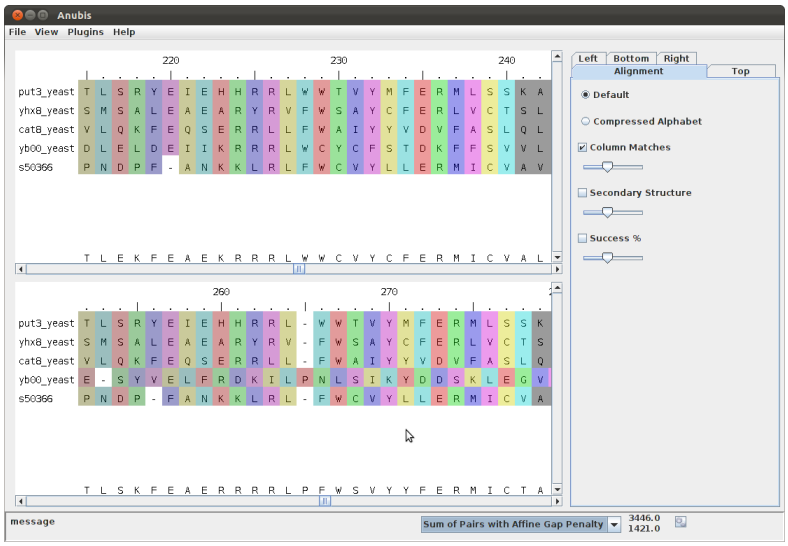


Figura 4.13: Exemplo do visualizador de estruturas secundárias do Anubis. É possível ver as α-hélices (em verde) e β-folhas (em amarelo).

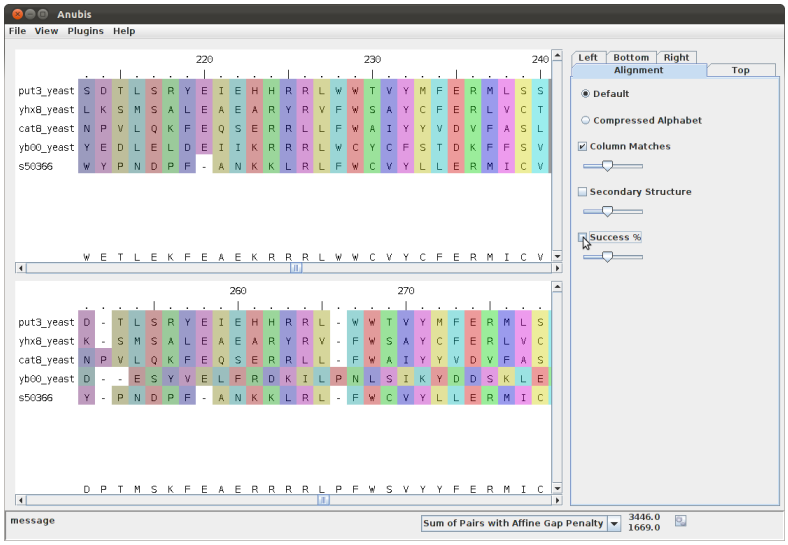
muito maior nas execuções do ALGAe entre o alinhamento da TBS1 com a sequência *cat8_yeast*, associada à proteína CAT8 [22], do que a esperada ao compararmos com a árvore filogenética gerada a partir do alinhamento referência. Esta similaridade pode ter guiado o processo de alinhamento de forma errônea, causando as discrepâncias verificadas.

Um outro ponto interessante pode ser verificado no cenário 1ubi, do conjunto RV11 do BAliBASE versão 2. Pode-se notar no cenário apresentado na Figura 4.16 que a pontuação do alinhamento gerado pelo ALGAe supera o do BAliBASE, indicando que neste caso a função objetivo não corresponde ao que se espera. Além disto, pode-se notar para o trecho apresentado que os alinhamentos estão próximos, com exceção de um conjunto de *gaps* contiguo. Isto indica que uma análise da função objetivo e de operadores que privilegiassem estas mudanças poderiam ser úteis.

Por fim, analisando os resultados do cenário *1R69* considerando suas estruturas secundárias vemos, na Figura 4.17a, que mesmo utilizando a função objetivo considerando estruturas o ALGAe falhou em alguns pontos em relação ao *benchmark*. Quando analisamos o mesmo cenário considerando o uso do alfabeto comprimido Dayhoff6, como visto na Figura 4.17b ficam mais evidentes os erros cometidos pelo algoritmo. Isto levanta a hipótese que uma combinação de informações estruturais e alfabetos comprimidos podem melhorar os resultados encontrados.



(a)



(b)

Figura 4.14: Correspondência de colunas para o cenário GAL4. No trecho apresentado as sequências se alinharam perfeitamente ao benchmark, com exceção da sequência *yb00_yeast*, associada à proteína TBS1 do levedo. São apresentados os resultados de duas execuções distintas do ALGAe, (a) e (b) , para alinhar este cenário. O alinhamento na parte superior da interface representam o *benchmark* e o alinhamento da parte inferior o alinhamento alvo.

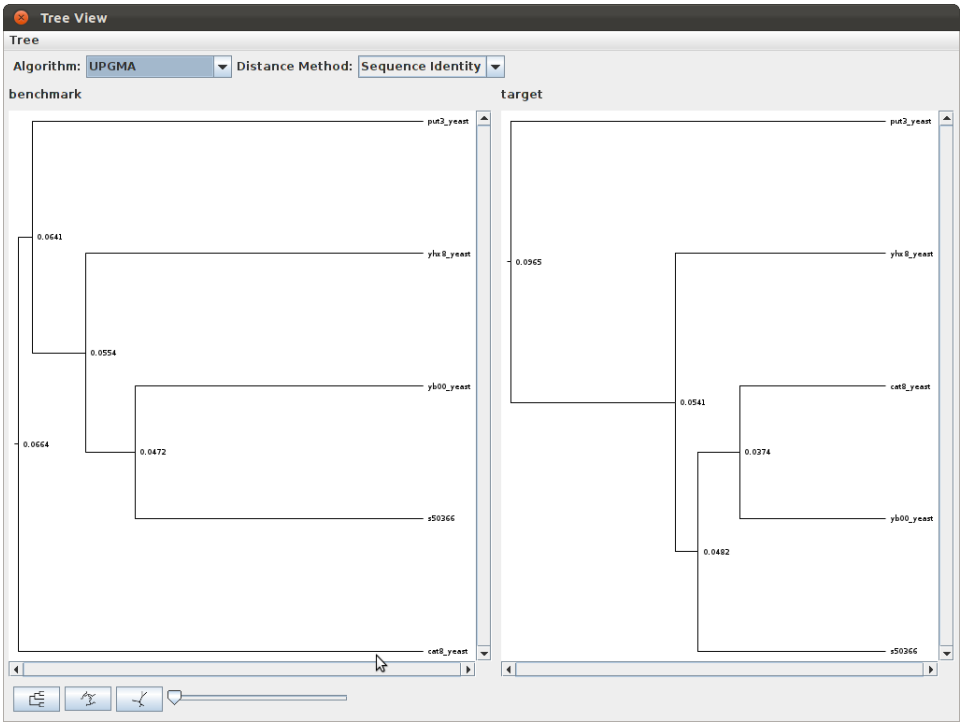


Figura 4.15: Interface do Anubis apresentando a árvore filogenética comparada de uma execução do ALGAe para o cenário GAL4 comparado ao alinhamento de referência do BAliBASE. Nota-se em especial a discrepância da sequência *cat8_yeast* em relação a esta.

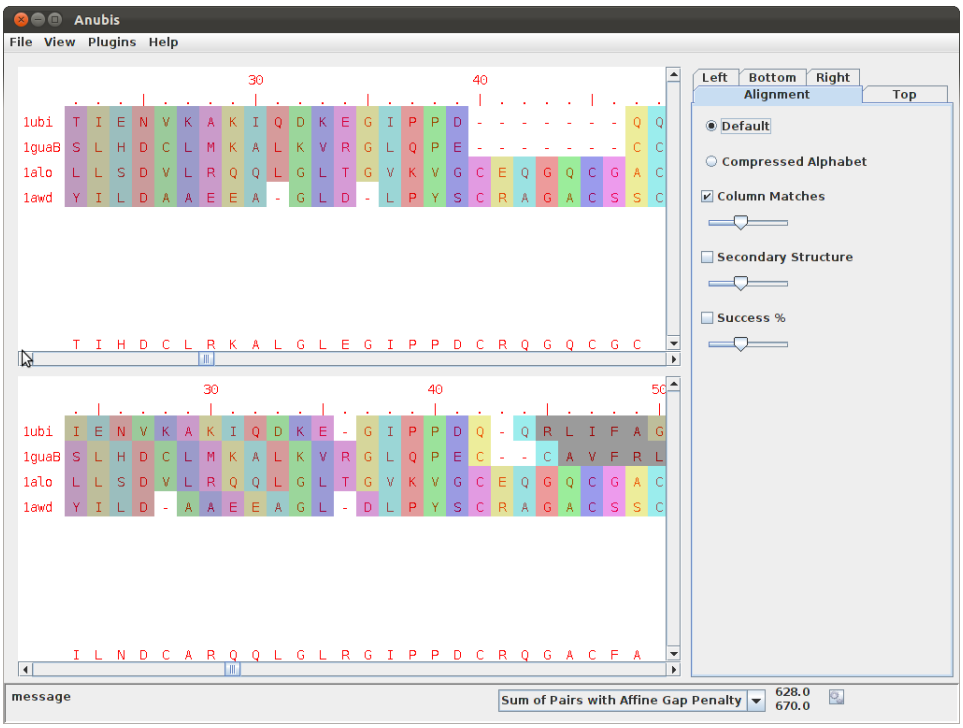
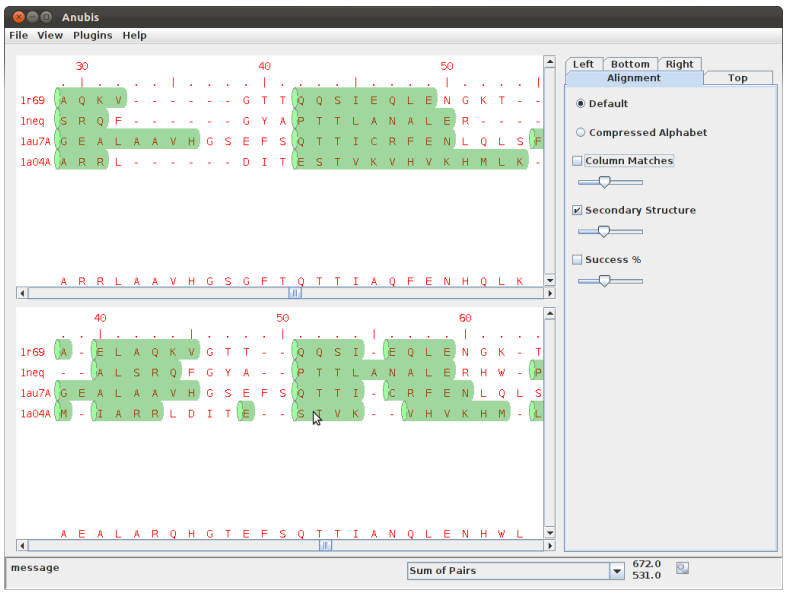
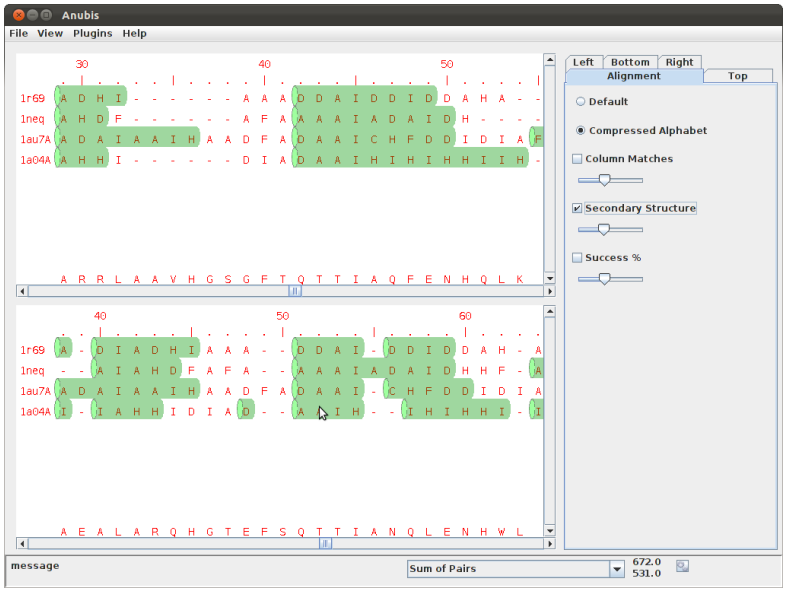


Figura 4.16: Correspondência de colunas em um trecho do cenário 1UBI. Vale notar como os dois cenários estão próximos neste trecho, com exceção do trecho de *gaps* existente no alinhamento de referência à partir da posição 40. Vale também notar como a pontuação para o alinhamento do ALGAe supera o obtido pelo alinhamento de referência, informação que pode ser vista na barra de mensagens na parte inferior da interface. O alinhamento na parte superior da interface representam o *benchmark* e o alinhamento da parte inferior o alinhamento alvo.



(a)



(b)

Figura 4.17: Comparação de estruturas secundárias no cenário 1R69. Podemos verificar na sequência 1a04A, referente a proteína NArL encontrada na *Escherichia coli*, que a α-hélice se alinha de forma um pouco deslocada quando comparada ao esperado no *benchmark*. Existem também alguns *gaps* que desalinham ainda mais a estrutura em relação a sequência 1neq, que se refere ao fator de transcrição PIT-1 encontrado no *Rattus norvegicus*. A figura (a) apresenta a comparação entre as sequências originais e a figura (b) os mesmos alinhamentos utilizando o alfabeto comprimido Dayhoff6. O alinhamento na parte superior da interface representam o *benchmark* e o alinhamento da parte inferior o alinhamento alvo.

Capítulo 5

Considerações Finais

O trabalho desenvolvido focou no problema de alinhamento múltiplo de sequências, voltado especialmente para sequências de proteínas. O objetivo era o desenvolvimento de soluções eficientes para a construção de MSAs utilizando algoritmos genéticos.

Os resultados obtidos foram razoáveis, porém não conseguiram aproximar-se dos resultados obtidos pelas melhores ferramentas disponíveis.

Porém, este trabalho gerou como colaboração a comunidade duas ferramentas que podem ser utilizadas em projetos correlatos no futuro. São elas: ALGAe, um ambiente configurável para a execução de um algoritmo genético para construção de MSAs e Anubis, um visualizador comparativo de MSAs com uma série de funcionalidades e uma estrutura que permite sua personalização e extensão.

As duas ferramentas se encontram disponíveis para uso e alterações, como *open-source*, em repositórios no *site* GitHub, sendo um para o ALGAe [86] e outro para o Anubis [87].

Além disto, dois trabalhos associados a esta dissertação, "*ALGAe: A test-bench environment for a Genetic Algorithm-based Multiple Sequence Aligner*" (S. Ordine, A. Grilo, A. Almeida e Z. Dias) [89] e "An Empirical Study for Gap Penalty Score Using a Multiple Sequence Alignment Genetic Algorithm" (S. Ordine, A. Almeida, Z. Dias) [88] foram publicados como resumos estendidos nos anais do "Brazilian Symposium on Bioinformatics", sendo o primeiro em Brasília, DF (BSB'2011) e o segundo em Campo Grande, MS (BSB'2012).

5.1 Trabalhos Futuros

Utilizando o ambiente ALGAe é possível facilmente que novas avalizações sobre os algoritmos genéticos sejam realizadas buscando melhorias no alinhador desenvolvido até o momento. Entre os testes que podem ser mais explorados, o uso de estruturas secundárias parece ser um dos mais promissores.

A ferramenta Anubis também pode ser melhorada para prover mais formas de visualização e comparação que possam ser úteis à comunidade. A estrutura flexível e personalizável desta ferramenta são fatores diferencias em relação a outras soluções similares encontradas.

Uma outra melhoria possível nesta ferramenta é permitir que ela não apenas sirva como um comparador de MSAs, mas também como um meio de mesclar dois alinhamentos. Com isto a ferramenta deixa de apenas comparar um bom alinhamento com um de qualidade inferior e passa também a permitir a comparação e união de dois alinhamentos razoáveis de forma a gerar um alinhamento melhor através da intervenção de um especialista.

Outro trabalho possível é na utilização de alinhamentos múltiplos para problemas específicos em bioinformática. Porém, para que este trabalho seja relevante necessita-se do apoio de grupos de biológos ou outros profissionais que tratem do problema alvo.

Existe ainda mais uma possibilidade vislumbrada: pode-se notar que as análises realizadas sobre as visualizações no Anubis necessitam de um suporte estatístico para verificar sua relevância em um conjunto maior de cenários e execuções. Sendo assim, uma linha de pesquisa futura pode ser a busca de ferramental estatístico que pode suportar a exploração de hipóteses ou mesmo auxiliar na busca de padrões de erro nos alinhamentos. O estudo de ferramentas estatísticas também pode ajudar na busca de correlações na análise dos alinhamentos par a par de sequências cuja busca iniciamos neste trabalho.

Referências Bibliográficas

[1] L. Abdesslem, M. Soham, and B. Mohamed. Multiple sequence alignment by quantum genetic algorithm. In *Proceedings of the 20th International Conference on Parallel and Distributed Processing*, IPDPS'2006, Washington, DC, USA, 2006. IEEE Computer Society.

[2] A. Agrawal, V. Brendel, and X. Huang. Pairwise statistical significance versus database statistical significance for local alignment of protein sequences. In *Bioinformatics Research and Applications*, pages 50–61. Springer, 2008.

[3] A. A. M. Almeida. *Novas abordagens para o problema do alinhamento múltiplo de sequências*. PhD thesis, University of Campinas, Brazil, 2013. In Portuguese.

[4] S. F. Altschul and B. Erickson. Optimal sequence alignment using affine gap costs. *Bulletin of Mathematical Biology*, 48(5):603–616, 1986.

[5] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[6] S. F. Altschul, T. L. Madden, A. A. Schäffer, J. Zhang, Z. Zhang, W.b Miller, and D. J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–3402, 1997.

[7] L. M. Amzel and R. J. Poljak. Three-dimensional structure of immunoglobulins. *Annual Review of Biochemistry*, 48(1):961–997, 1979.

[8] C. Anderson, C. Strope, and E. Moriyama. SuiteMSA: visual tools for multiple sequence alignment comparison and molecular sequence simulation. *BMC Bioinformatics*, 12(1):184, 2011.

[9] F. Armougom, S. Moretti, O. Poirot, S. Audic, P. Dumas, B. Schaeli, V. Keduas, and C. Notredame. Expresso: automatic incorporation of structural information in multiple sequence alignments using 3d-coffee. *Nucleic Acids Research*, 34(suppl 2):W604–W608, 2006.

[10] A. Bahr, J. D. Thompson, J. C. Thierry, and O. Poch. BAliBASE (Benchmark Alignment dataBASE): enhancements for repeats, transmembrane sequences and circular permutations. *Nucleic Acids Research*, 29(1):323–326, 2001.

[11] N. Barricelli. Esempi numerici di processi di evoluzione. *Methodos*, pages 45–68, 1954.

[12] G. J. Barton and M. J. E. Sternberg. A strategy for the rapid multiple alignment of protein sequences: Confidence levels from tertiary structure comparisons. *Journal of Molecular Biology*, 198(2):327 – 337, 1987.

[13] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne. The protein data bank. *Nucleic Acids Research*, 28(1):235–242, 2000.

[14] J. Błażewicz, P. Formanowicz, and P. Wojciechowski. Some remarks on evaluating the quality of the multiple sequence alignment based on the BAliBASE benchmark. *International Journal of Applied Mathematics and Computer Science*, 19(4):675–678, 2009.

[15] M. Botta and G. Negro. Multiple sequence alignment with genetic algorithms. In *Computational Intelligence Methods for Bioinformatics and Biostatistics*, volume 6160 of *Lecture Notes in Computer Science*, pages 206–214. Springer, 2010.

[16] R. K. Bradley, A. Roberts, M. Smoot, S. Juvekar, J. Do, C. Dewey, I. Holmes, and L. Pachter. Fast statistical alignment. *PLoS Computational Biology*, 5(5):e1000392, 2009.

[17] C. Branden and J. Tooze. *Introduction to protein structure*. Garland Publishing, Inc., 2nd edition, 1999.

[18] S. E. Brenner, C. Chothia, and T. J. Hubbard. Assessing sequence comparison methods with reliable structurally identified distant evolutionary relationships. *Proceedings of the National Academy of Sciences*, 95(11):6073–6078, 1998.

[19] T. A Brown. *Genomes*. Oxford: Wiley-Liss, 2nd edition, 2002.

[20] H. Carrillo and D. Lipman. The multiple sequence alignment problem in biology. *SIAM Journal on Applied Mathematics*, 48(5):1073–1082, 1988.

[21] M. Clamp, J. Cuff, S. M. Searle, and G. J. Barton. The Jalview java alignment editor. *Bioinformatics*, 20(3):426–427, 2004.

[22] UniProt Consortium et al. The universal protein resource (UniProt). *Nucleic Acids Research*, 36(suppl 1):D190–D195, 2008.

[23] F. H. C. Crick. On protein synthesis. In *Symposia of the Society for Experimental Biology*, volume 12, page 138, 1958.

[24] F. H. C. Crick. Central dogma of molecular biology. *Nature*, 227(5258):561–563, 1970.

[25] C. Darwin. *The Origin of Species By Means Of Natural Selecion, or the Preservation of Favoured Races in the Struggle for Life*. John Murray, 1st edition, 1859.

[26] M. O. Dayhoff and R. M. Schwartz. Chapter 22: A model of evolutionary change in proteins. In *Atlas of Protein Sequence and Structure*, 1978.

[27] C. B. Do, M. S. P. Mahabhashyam, M. Brudno, and S. Batzoglou. ProbCons: Probabilistic consistency-based multiple sequence alignment. *Genome Research*, 15:330–340, 2005.

[28] A. Drummond, A. Wilson, A. Rambaut, et al. Java evolutionary biology library. `http://sourceforge.net/projects/jebl/`. Acessado em 11/08/2015.

[29] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge University Press, 1998.

[30] L. Duret and S. Abdeddaim. *Bioinformatics: sequence, structure and databanks*, chapter Multiple alignment for structural functional or phylogenetic analyses of homologous sequences. Oxford University Press, Oxford, UK, 2000.

[31] S. R. Eddy et al. Multiple alignment using hidden markov models. In *Proceedings of the III International Conference on Intelligent Systems for Molecular Biology*, volume 3 of *ISMB'95*, pages 114–120, 1995.

[32] R. C. Edgar. Local homology recognition and distance measures in linear time using compressed amino acid alphabets. *Nucleic Acids Research*, 32(1):380–385, 2004.

[33] R. C. Edgar. MUSCLE: a multiple sequence alignment method with reduced time and space complexity. *BMC Bioinformatics*, 5(1):113, 2004.

[34] R. C. Edgar. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, 32(5):1792–1797, 2004.

[35] R. C. Edgar. Quality measures for protein alignment benchmarks. *Nucleic Acids Research*, page gkp1196, 2010.

[36] N. Essoussi, K. Boujenfa, and M. Limam. A comparison of MSA tools. *Bioinformation*, 2(10):452, 2008.

[37] D. F. Feng and R. F. Doolittle. Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *Journal of Molecular Evolution*, 25(4):351–360, 1987.

[38] D. J. Futuyma. *Evolution*. Sinauer Associates, Inc., 2nd edition, 2009.

[39] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[40] P. P. Gardner, A. Wilm, and S. Washietl. A benchmark of multiple sequence alignment programs upon structural RNAs. *Nucleic Acids Research*, 33(8):2433–2439, 2005.

[41] D. E. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. *Urbana*, 51:61801–2996, 1991.

[42] C. Gondro and B. P. Kinghorn. A simple genetic algorithm for multiple sequence alignment. *Genetics and Molecular Research*, 6(4):964–982, 2007.

[43] J. Gorodkin, L. J. Heyer, and G. D. Stormo. Finding the most significant common sequence and structure motifs in a set of rna sequences. *Nucleic Acids Research*, 25(18):3724–3732, 1997.

[44] O. Gotoh. Consistency of optimal sequence alignments. *Bulletin of Mathematical Biology*, 52(4):509–525, 1990.

[45] O. Gotoh. Significant improvement in accuracy of multiple protein sequence alignments by iterative refinement as assessed by reference to structural alignments. *Journal of Molecular Biology*, 264(4):823–838, 1996.

[46] D. Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology*. Cambridge University Press, 1997.

[47] N. T. Hang. *Comparison of multiple sequence alignment programs in practise*. PhD thesis, University of Århus, 2008.

[48] S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Science*, 89(22):10915–10919, 1992.

[49] I. L. Hofacker, S. Bernhart, and P. F. Stadler. Alignment of RNA base pairing probability matrices. *Bioinformatics*, 20(14):2222–2227, 2004.

[50] P. Hogeweg and B. Hesper. The alignment of sets of sequences and the construction of phyletic trees: An integrated method. *Journal of Molecular Evolution*, 20(2):175–186, 1984.

[51] J. Holland. *Adaptation in natural and artificial systems.* University of Michigan Press, 1975.

[52] L. Holm and C. Sander. The FSSP database of structurally aligned protein fold families. *Nucleic Acids Research*, 22(17):3600, 1994.

[53] I. Holmes. A probabilistic model for the evolution of RNA structure. *BMC Bioinformatics*, 5(1):166, 2004.

[54] J. Horng, L. Wu, C. Lin, and B. Yang. A genetic algorithm for multiple sequence alignment. *Soft Computing*, 9(6):407–420, 2005.

[55] X. Huang and W. Miller. A time-efficient, linear-space local similarity algorithm. *Advances in Applied Mathematics*, 12(3):337–357, 1991.

[56] M. Isokawa, M. Wayama, and T. Shimizu. Multiple sequence alignment using a genetic algorithm. *Genome Informatics*, 7:176–177, 1996.

[57] D. T. Jones. Protein secondary structure prediction based on position-specific scoring matrices. *Journal of Molecular Biology*, 292(2):195–202, 1999.

[58] W. Just. Computational complexity of multiple sequence alignment with SP-score. *Journal of Computational Biology*, 8:615–623, 2001.

[59] A. Kawrykow, G. Roumanis, A. Kam, D. Kwak, C. Leung, C. Wu, E. Zarour, Phylo players, L. Sarmenta, M. Blanchette, and J. Waldispühl. Phylo: a citizen science approach for improving multiple sequence alignment. *PLoS One*, 7(3):e31362, 2012.

[60] J. Kececioglu. The maximum weight trace problem in multiple sequence alignment. In *IV Annual Symposium on Combinatorial Pattern Matching*, CPM'93, pages 106–119. Springer, 1993.

[61] C. Kemena and C. Notredame. Upcoming challenges for multiple sequence alignment methods in the high-throughput era. *Bioinformatics*, 25(19):2455–2465, 2009.

[62] J. Kim, S. Pramanik, and M. J. Chung. Multiple sequence alignment using simulated annealing. *Computer Applications in the Biosciences*, 10(4):419–426, 1994.

[63] M. Kimura. *The neutral theory of molecular evolution*. Cambridge University Press, 1984.

[64] G. J. Kleywegt and T. A. Jones. Software for handling macromolecular envelopes. *Acta Crystallographica Section D: Biological Crystallography*, 55(4):941–944, 1999.

[65] D. Kwak, A. Kam, D. Becerra, Q. Zhou, A. Hops, E. Zarour, A. Kam, L. Sarmenta, M. Blanchette, and J. Waldispühl. Open-phylo: a customizable crowd-computing platform for multiple sequence alignment. *Genome Biology*, 14(10):R116, 2013.

[66] C. E. Lawrence, S. F. Altschul, M. S. Boguski, J. S. Liu, A. F. Neuwald, and J. C. Wootton. Detecting subtle sequence signals: a gibbs sampling strategy for multiple alignment. *Science*, 262(5131):208–214, 1993.

[67] Z. Lee, S. Su, C. Chuang, and K. Liu. Genetic algorithm with ant colony optimization (GA-ACO) for multiple sequence alignment. *Applied Soft Computing*, 8(1):55 – 78, 2008.

[68] A. M. Lesk and C. Chothia. How different amino acid sequences determine similar protein structures: the structure and evolutionary dynamics of the globins. *Journal of Molecular Biology*, 136(3):225–270, 1980.

[69] D. J. Lipman, S. F. Altschul, and J. D. Kececioglu. A tool for multiple sequence alignment. *Proceedings of the National Academy of Sciences*, 86(12):4412–4415, 1989.

[70] M. Lou and G. B. Golding. Fingerprint: Visual depiction of variation in multiple sequence alignments. *Molecular Ecology Notes*, 7(6):908–914, 2007.

[71] A. Löytynoja and N. Goldman. Phylogeny-aware gap placement prevents errors in sequence alignment and evolutionary analysis. *Science*, 320(5883):1632–1635, 2008.

[72] A. Löytynoja and M. C. Milinkovitch. Proalign, a probabilistic multiple alignment program. *Bioinformatics*, 19:1505–1513, 2003.

[73] S. Meshoul, A. Layeb, and M. Batouche. A quantum evolutionary algorithm for effective multiple sequence alignment. In *Progress in Artificial Intelligence*, volume 3808 of *Lecture Notes in Computer Science*, pages 260–271. Springer Berlin / Heidelberg, 2005.

[74] K. Mizuguchi, C. M. Deane, T. L. Blundell, and J. P. Overington. HOMSTRAD: a database of protein structure alignments for homologous families. *Protein Science*, 7(11):2469–2471, 1998.

[75] B. Morgenstern, K. Frech, A. Dress, and T. Werner. DIALIGN: finding local similarities by multiple sequence alignment. *Bioinformatics*, 14(3):290–294, 1998.

[76] B. Morgenstern, S. Goel, A. Sczyrba, and A. Dress. AltAVisT: comparing alternative multiple sequence alignments. *Bioinformatics*, 19(3):425–426, 2003.

[77] J. D. Moss and C. G. Johnson. An ant colony algorithm for multiple sequence alignment in bioinformatics. *Artificial Neural Networks and Genetic Algorithms*, pages 182–186, 2003.

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

[79] G. Negro. A stochastic evolutionary algorithm for multiple sequence alignment. Master's thesis, Università degli studi di Torino, Itália, 2010. In Italian.

[80] A. Nizam, B. Shanmugham, and K. Subburaya. Self-organizing genetic algorithm for multiple sequence alignment. *Global Journal of Computer Science and Technology*, 11(7), 2011.

[81] C. Notredame. Recent progress in multiple sequence alignment: a survey. *Pharmacogenomics*, 3(1):131–144, 2002.

[82] C. Notredame and D. G. Higgins. SAGA: sequence alignment by genetic algorithm. *Nucleic Acids Research*, 24(8):1515–1524, 1996.

[83] C. Notredame, D. G. Higgins, and J. Heringa. T-Coffee: A novel method for fast and accurate multiple sequence alignment. *Journal of Molecular Biology*, 302(1):205–217, 2000.

[84] C. Notredame, L. Holm, and D. G. Higgins. COFFEE: an objective function for multiple sequence alignments. *Bioinformatics*, 14(5):407–422, 1998.

[85] R. Nussinov, G. Pieczenik, J. R. Griggs, and D. J. Kleitman. Algorithms for loop matchings. *SIAM Journal on Applied Mathematics*, 35(1):68–82, 1978.

[86] S. Ordine. Algae. `https://github.com/sjordine/ALGAe.git`. Acessado em 04/01/2015.

[87] S. Ordine. Anubis. `https://github.com/sjordine/Anubis.git`. Acessado em 04/01/2015.

[88] S. Ordine, A. Almeida, and Z. Dias. An empirical study for gap penalty score using a multiple sequence alignment genetic algorithm. In *VII Brazilian Symposium on Bioinformatics 2012 Digital Proceedings*, BSB'2012, pages 108–113, 2012.

[89] S. Ordine, A. Grilo, A. Almeida, and Z. Dias. ALGAe: A test-bench environment for a genetic algorithm-based multiple sequence aligner. In *VI Brazilian Symposium on Bioinformatics 2011 Digital Proceedings*, BSB'2011, pages 57–60, 2011.

[90] O. O'Sullivan, K. Suhre, C. Abergel, D. G. Higgins, and C. Notredame. 3DCoffee: combining protein sequences and structures within multiple sequence alignments. *Journal of Molecular Biology*, 340(2):385–395, 2004.

[91] J. Pei and N. V. Grishin. PROMALS: towards accurate multiple sequence alignments of distantly related proteins. *Bioinformatics*, 23(7):802–808, 2007.

[92] J. Pei, B. Kim, and N. V. Grishin. PROMALS3D: a tool for multiple protein sequence and structure alignments. *Nucleic Acids Research*, 36(7):2295–2300, 2008.

[93] J. Pei, R. Sadreyev, and N. V. Grishin. PCMA: fast and accurate multiple sequence alignment based on profile consistency. *Bioinformatics*, 19(3):427–428, 2003.

[94] A. Rambaut. Figtree, a graphical viewer of phylogenetic trees. `http://tree.bio.ed.ac.uk/software/figtree/`. Acessado em 24/11/2014.

[95] J. T. Reese and W. R. Pearson. Empirical determination of effective gap penalties for sequence comparison. *Bioinformatics*, 18(11):1500–1507, 2002.

[96] J. S. Richardson. The anatomy and taxonomy of protein structure. *Advances in Protein Chemistry*, 34:167–339, 1981.

[97] A. Roca, A. Almada, and A. Abajian. Profilegrids as a new visual representation of large multiple sequence alignments: a case study of the reca protein family. *BMC Bioinformatics*, 9(1):554, 2008.

[98] N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4(4):406–425, 1987.

[99] D. Sankoff. Simultaneous solution of the rna folding, alignment and protosequence problems. *SIAM Journal on Applied Mathematics*, 45(5):810–825, 1985.

[100] D. Santos. Alinhamento múltiplo de proteínas via algoritmo genético baseado em tipos abstratos de dados. Master's thesis, Universidade Federal de Alagoas, 2008. In Portuguese.

[101] A. Sedaghatinia, R. B. Atan, K. Arifin, and M. Murad. Comparison and evaluation of multiple sequence alignment tools in bioinformatics. *International Journal of Computer Science and Network Security*, 9:51–56, 2009.

[102] J. C. Setubal and J. Meidanis. *Introduction to computational molecular biology.* PWS, 1997.

[103] J. Shi, T. L. Blundell, and K. Mizuguchi. FUGUE: sequence-structure homology recognition using environment-specific substitution tables and structure-dependent gap penalties. *Journal of Molecular Biology*, 310(1):243–257, 2001.

[104] A. C. C. Shih, D. T. Lee, L. Lin, C. Peng, S. Chen, Y. Wu, C. Wong, M. Chou, T. Shiao, and M. Hsieh. SinicView: a visualization environment for comparisons of multiple nucleotide sequence alignment tools. *BMC Bioinformatics*, 7(1):103, 2006.

[105] R. F. Smith and T. F. Smith. Pattern-induced multi-sequence alignment (PUMA) algorithm employing secondary structure-dependent gap penalties for use in comparative protein modelling. *Protein Engineering*, 5(1):35–41, 1992.

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

[107] P. H. A. Sneath and R. R. Sokal. *Numerical taxonomy. The principles and practice of numerical classification.* Freeman, 1973.

[108] M. A. L. Souza. Alinhamento múltiplo progressivo de sequências de proteínas. Master's thesis, University of Campinas, Brazil, 2010. In Portuguese.

[109] M. Srinivas and L. M. Patnaik. Genetic algorithms: A survey. *Computer*, 27(6):17–26, 1994.

[110] C. L. Strope, K. Abel, S. D. Scott, and E. N. Moriyama. Biological sequence simulation for testing complex evolutionary hypotheses: indel-Seq-Gen version 2.0. *Molecular Biology and Evolution*, 26(11):2581–2593, 2009.

[111] C. L. Strope, S. D. Scott, and E. N. Moriyama. indel-Seq-Gen: a new protein family simulator incorporating domains, motifs, and indels. *Molecular Biology and Evolution*, 24(3):640–649, 2007.

[112] C. L. Strope, S. D. Scott, and E. N. Moriyama. Gap profiling: Scoring indels in multiple sequence alignment. In *VI Biotechnology and Bioinformatics Symposium Proceedings*, BIOT'2009, pages 75–76. University of Nebraska-Lincoln, 2009.

[113] K. Tajima. Multiple sequence alignment using parallel genetic algorithms. In *IV Genome Informatics Workshop, Yokohama*, GIW'93, pages 183–187, 1993.

[114] W. R. Taylor. A flexible method to align large numbers of biological sequences. *Journal of Molecular Evolution*, 28(1-2):161–169, 1988.

[115] W. R. Taylor and C. A. Orengo. Protein structure alignment. *Journal of Molecular Biology*, 208(1):1–22, 1989.

[116] J. D. Thompson, T. J. Gibson, F. Plewniak, F. Jeanmougin, and D. G. Higgins. The CLUSTAL X windows interface: Flexible strategies for multiple sequence alignment aided by quality analysis tools. *Nucleic Acids Research*, 25(24):4876–4882, 1997.

[117] J. D. Thompson, D. G. Higgins, and T. 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.

[118] J. D. Thompson, P. Koehl, R. Ripp, and O. Poch. BAliBASE 3.0: latest developments of the multiple sequence alignment benchmark. *Proteins*, 61(1):127–136, 2005.

[119] J. D. Thompson, F. Plewniak, and O. Poch. BAliBASE: a benchmark alignment database for the evaluation of multiple alignment programs. *Bioinformatics*, 15(1):87–88, 1999.

[120] J. D. Thompson, F. Plewniak, and O. Poch. A comprehensive comparison of multiple sequence alignment programs. *Nucleic Acids Research*, 27(13):2682–2690, 1999.

[121] R. Thomsen and W. Boomsma. Multiple Sequence Alignment Using SAGA: Investigating the effects of operator scheduling, population seeding, and crossover operators. In *Applications of Evolutionary Computing*, volume 3005 of *Lecture Notes in Computer Science*, pages 113–122. Springer, 2004.

[122] F. Vavak and T. C. Fogarty. Comparison of steady state and generational genetic algorithms for use in nonstationary environments. In *Proceedings of IEEE International Conference on Evolutionary Computation*, CEC'96, pages 192–195. IEEE, 1996.

[123] M. Vingron and P. Argos. A fast and sensitive multiple sequence alignment algorithm. *Computer Applications in the Biosciences*, 5(2):115–121, 1989.

[124] M. Vingron and M. Waterman. Sequence alignment and penalty choice: Review of concepts, case studies and implications. *Journal of Molecular Biology*, 235(1):1–12, 1994.

[125] L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *Journal of Computational Biology*, 1(4):337–348, 1994.

[126] A. M Waterhouse, J. B. Procter, D. Martin, M. Clamp, and G. J. Barton. Jalview version 2 - a multiple sequence alignment editor and analysis workbench. *Bioinformatics*, 25(9):1189–1191, 2009.

[127] J. D. Watson and F. H. C. Crick. Molecular structure of nucleic acids: A structure for deoxyribose nucleic acid. *Nature*, 171:737–738, 1953.

[128] D. Whitley and J. Kauth. *GENITOR: A different genetic algorithm*. Colorado State University, Department of Computer Science, 1988.

[129] J. O. Wrabl and N. V. Grishin. Gaps in structurally similar proteins: towards improvement of multiple sequence alignment. *Proteins: Structure, Function, and Bioinformatics*, 54(1):71–87, 2004.

[130] C. H. Wu, R. Apweiler, A. Bairoch, D. A. Natale, W. C. Barker, B. Boeckmann, S. Ferro, E. Gasteiger, H. Huang, R. Lopez, et al. The universal protein resource (UniProt): an expanding universe of protein information. *Nucleic Acids Research*, 34(suppl 1):D187–D191, 2006.

[131] H. Zang, S. Zhang, and K. Hapeshi. A review of nature-inspired algorithms. *Journal of Bionic Engineering*, 7(Supplement 1):S232 – S237, 2010.

[132] C. Zhang and A. K. C. Wong. A genetic algorithm for multiple molecular sequence alignment. *Computer Applications in the Biosciences*, 13(6):565–581, 1997.

[133] M. Zuker and P. Stiegler. Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. *Nucleic Acids Research*, 9(1):133–148, 1981.