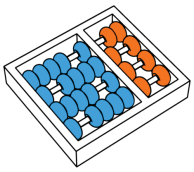
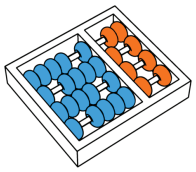


André Atanasio Maranhão Almeida

"Novas abordagens para o problema do alinhamento múltiplo de sequências"

CAMPINAS
2013



Universidade Estadual de Campinas
Instituto de Computação

André Atanasio Maranhão Almeida

"Novas abordagens para o problema do alinhamento múltiplo de sequências"

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

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Computação da Universidade Estadual de Campinas para obtenção do título de Doutor em Ciência da Computação.

Este exemplar corresponde à versão final da Tese defendida por André Atanasio Maranhão Almeida, sob orientação de Prof. Dr. Zanoni Dias.

Assinatura do Orientador(a)

CAMPINAS
2013

FICHA CATALOGRÁFICA ELABORADA POR
ANA REGINA MACHADO - CRB8/5467
BIBLIOTECA DO INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E
COMPUTAÇÃO CIENTÍFICA - UNICAMP

Almeida, André Atanasio Maranhão, 1981-
AL64n Novas abordagens para o problema do alinhamento múltiplo de sequências / André Atanasio Maranhão Almeida. – Campinas, SP : [s.n.], 2013.

Orientador: Zanoni Dias.
Tese (doutorado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Bioinformática. 2. Sequência de aminoácidos. 3. Alinhamento múltiplo de sequências. 4. Proteínas. I. Dias, Zanoni,1975-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Informações para Biblioteca Digital

Título em inglês: New approaches for the multiple sequence alignment problem
Palavras-chave em inglês:
Bioinformatics
Amino acid sequence
Multiple sequence alignments
Proteins
Área de concentração: Ciência da Computação
Titulação: Doutor em Ciência da Computação
Banca examinadora:
Zanoni Dias [Orientador]
Guilherme Pimentel Telles
Orlando Lee
Felipe Rodrigues da Silva
Maria Emília Machado Telles Walter
Data de defesa: 21-02-2013
Programa de Pós-Graduação: Ciência da Computação

TERMO DE APROVAÇÃO

Tese Defendida e Aprovada em 21 de Fevereiro de 2013, pela Banca examinadora composta pelos Professores Doutores:

Dr. Felipe Rodrigues da Silva
Informática Agropecuária / EMBRAPA

Profª. Drª. Maria Emília Machado Telles Walter
CIC / UnB

Prof. Dr. Guilherme Pimentel Telles
IC / UNICAMP

Prof. Dr. Orlando Lee
IC / UNICAMP

Prof. Dr. Zanoni Dias
IC / UNICAMP

Instituto de Computação
Universidade Estadual de Campinas

Novas abordagens para o problema do alinhamento múltiplo de sequências

André Atanasio Maranhão Almeida[1]

21 de Fevereiro de 2013

Banca Examinadora:

- Prof. Dr. Zanoni Dias (Orientador)
- Prof. Dr. Guilherme Pimentel Telles
  Instituto de Computação - UNICAMP
- Prof. Dr. Orlando Lee
  Instituto de Computação - UNICAMP
- Dr. Felipe Rodrigues da Silva
  Embrapa Informática Agropecuária
- Prafa. Dra. Maria Emília Machado Telles Walter
  Instituto de Ciências Exatas - UnB
- Prof. Dr. João Meidanis
  Instituto de Computação - UNICAMP (Suplente)
- Prof. Dr. Cid Carvalho de Souza
  Instituto de Computação - UNICAMP (Suplente)
- Prof. Dr. Nalvo Franco de Almeida Júnior
  Faculdade de Computação - UFMS (Suplente)

[1]Suporte financeiro: bolsa CAPES 2007–2008 e bolsa FAPESP (processo 2007/08020-0) 2008–2011.

Abstract

Sequence alignment is one the most important tasks of bioinformatics. It is a basic operation used for several procedures in that domain, such as sequence database searches, evolution effect visualization in an entire protein family, phylogenetic trees construction and preserved motifs identification.

Sequences can be aligned in pairs and generate a pairwise alignment. Three or more sequences can also be simultaneously aligned and generate a multiple sequence alignment (MSA). MSAs could be used for pattern recognition for protein family characterization and secondary and tertiary protein structure prediction. Let l be the sequence length. The pairwise alignment takes time $O(l^2)$ to build an exact alignment. However, multiple sequence alignment is a NP-Hard problem.

In this work, heuristic methods were developed for multiple protein sequence alignment. The main approaches and methods applied to the problem were studied and a series of aligners developed and evaluated.

In a first moment 342 multiple aligners using the progressive approach were built. That is a largely used approach for MSA construction and is composed by three steps. In the first one a distance matrix is computed. Then, a guide tree is built based on the matrix and finally the MSA is constructed through pairwise alignments. The order to the pairwise alignments is defined by the tree. The developed aligners combine distinct methods applied to each of steps.

Then, evaluations in the consistency based alignment context were performed. In that approach, a MSA is optimal when agree with the majority along all possible optimal pairwise alignments. MUMMALS is a known consistency based aligner. It was changed in this evaluation. The k-mer counting method was modified in two distinct ways. The k value and the compressed alphabet were ranged. In an other evaluation, the k-mer counting method and guide tree construction method were replaced.

In the last stage of the work, iterative algorithms were developed and evaluated. That methods are characterized by other aligners dependence. The other aligners generate an initial population and the iterative aligner performs a refinement procedure, which iteratively changes the alignments until the alignments quality are stabilized. Several

evaluations were performed. However, a genetic algorithm for MSA construction stood out along this stage. In that aligner were added other approaches for multiple sequence alignment construction, such as block based, consensus based and template based. The first one was applied to initial population generation, the second one was used for a crossover operator creation and the third one defined an fitness function.

Resumo

Alinhamento de sequências é, reconhecidamente, uma das tarefas de maior importância em bioinformática. Tal importância origina-se no fato de ser uma operação básica utilizada por diversos outros procedimentos na área, como buscas em bases de dados, visualização do efeito da evolução em uma família de proteínas, construção de árvores filogenéticas e identificação de *motifs* preservados.

Sequências podem ser alinhadas aos pares, problema para o qual já se conhece algoritmo exato com complexidade de tempo $O(l^2)$, para sequências de comprimento l. Pode-se também alinhar simultaneamente três ou mais sequências, o que é chamado de alinhamento múltiplo de sequências (MSA, do inglês *Multiple Sequence Alignment*). Este, que é empregado em tarefas como detecção de padrões para caracterizar famílias proteicas e predição de estruturas secundárias e terciárias de proteínas, é um problema NP-Difícil.

Neste trabalho foram desenvolvidos métodos heurísticos para alinhamento múltiplo de sequências de proteína. Estudou-se as principais abordagens e métodos existentes e foi realizada uma série de implementações e avaliações.

Em um primeiro momento foram construídos 342 alinhadores múltiplos utilizando a abordagem progressiva. Esta, que é uma abordagem largamente utilizada para construção de MSAs, consiste em três etapas. Na primeira delas é computada a matriz de distâncias. Em seguida, uma árvore guia é gerada com base na matriz e, finalmente, o MSA é construído através de alinhamentos de pares, cuja ordem é definida pela árvore. Os alinhadores desenvolvidos combinam diferentes métodos aplicados a cada uma das etapas.

Para a computação das matrizes de distâncias foram desenvolvidos dois métodos, que são capazes também de gerar alinhamentos de pares de sequências. Um deles constrói o alinhamento com base em alinhamentos locais e o outro utiliza uma função logarítmica para a penalização de *gaps*. Foram utilizados ainda outros métodos disponíveis numa ferramenta chamada PHYLIP. Para a geração das árvores guias, foram utilizados os métodos clássicos UPGMA e *Neighbor Joining*. Usou-se implementações disponíveis em uma ferramenta chamada R.

Já para a construção do alinhamento múltiplo, foram implementados os métodos seleção por bloco único e seleção do par mais próximo. Estes, que se destinam à seleção do

par de alinhamentos a agrupar no ciclo corrente, são comumente utilizados para tal tarefa. Já para o agrupamento de um par de alinhamentos, foram implementados 12 métodos inspirados em métodos comumente utilizados – alinhamento de consensos e alinhamento de perfis. Foram feitas todas as combinações possíveis entre esses métodos, resultando em 342 alinhadores. Eles foram avaliados quanto à qualidade dos alinhamentos que geram e avaliou-se também o desempenho dos métodos, utilizados em cada etapa.

Em seguida foram realizadas avaliações no contexto de alinhamento baseado em consistência. Nesta abordagem, considera-se MSA ótimo aquele que está de acordo com a maioria dos alinhamentos ótimos para os $n(n-1)/2$ alinhamentos de pares contidos no MSA. Alterações foram realizadas em um alinhador múltiplo conhecido, MUMMALS, que usa a abordagem. As modificações foram feitas no método de contagem k-mer, assim como, em um outro momento, substituiu-se a parte inicial do algoritmo. Foram alterados os métodos para computação da matriz de distâncias e para geração da árvore guia por outros que foram bem avaliados nos testes realizados para a abordagem progressiva. No total, foram implementadas e avaliadas 89 variações do algoritmo original do MUMMALS e, apesar do MUMMALS já produzir alinhamentos de alta qualidade, melhoras significativas foram alcançadas.

O trabalho foi concluído com a implementação e a avaliação de algoritmos iterativos. Estes caracterizam-se pela dependência de outros alinhadores para a produção de alinhamentos iniciais. Ao alinhador iterativo cabe a tarefa de refinar tais alinhamentos através de uma série de ciclos até que haja uma estabilização na qualidade dos alinhamentos. Foram implementados e avaliados dois alinhadores iterativos não estocásticos, assim como um algoritmo genético (GA) voltado para a geração de MSAs. Nesse algoritmo genético, implementado na forma de um ambiente parametrizável para execução de algoritmos genéticos para MSA, chamado ALGAe, foram realizadas diversas experiências que progressivamente elevaram a qualidade dos alinhamentos gerados.

No ALGAe foram incluídas outras abordagens para construção de alinhamentos múltiplos, tais como baseada em blocos, em consenso e em modelos. A primeira foi aplicada na geração de indivíduos para a população inicial. Foram implementados alinhadores baseados em blocos usando duas abordagens distintas e, para uma delas, foram implementadas cinco variações. A segunda foi aplicada na definição de um operador de cruzamento, que faz uso da ferramenta M-COFFEE para realizar alinhamentos baseados em consenso a partir de indivíduos da população corrente do GA, e a terceira foi utilizada para definir uma função de aptidão, que utiliza a ferramenta PSIPRED para predição das estruturas secundárias das sequências. O ALGAe permite a realização de uma grande variedade de novas avaliações.

Agradecimentos

Aos meus pais por todo o apoio e participação ativa ao longo de toda a minha formação. Assim como, aos meus irmãos e familiares, em especial tias Denilma, Vitória e Telma.

Ao meu orientador, Zanoni Dias, pela disposição, pelos incentivos, pela paciência e pela confiança em mim depositada ao longo de todo o doutorado.

A Maria Angélica Lopes de Souza, Alex Bredariol Grilo e Sergio Jeferson Rafael Ordine pela parceria na realização de trabalhos.

Aos amigos com os quais convivi em Campinas, Marcelo Sampaio e Elis, Rinaldo Vieira e Ingrid, Leonardo Rangel e Sânia, Renato Neves e Mayza, Ulisses Dias e Dani, Neumar Malheiros, Diego Solarte, Jorge Lima, Carlos Froldi, André Drummond, Walisson Pereira e Christian Baudet, pelo fato de tornarem esse período muito mais agradável, o que certamente repercutiu na condução do doutorado.

Às agências de fomento, Capes e FAPESP, pelo suporte financeiro.

Finalmente, dedico este trabalho a minha esposa, Taciana, que sempre ofereceu-me apoio e foi compreensiva nos diversos momentos de ausência ou stress.

Sumário

Lista de Tabelas

Lista de Figuras

Lista de Algoritmos

Capítulo 1

Introdução

Quando um genoma é sequenciado, um dos principais objetivos é entender o conjunto de genes do qual ele é composto. Em outras palavras, busca-se entender a estrutura, a função e a evolução dos genes. Tradicionalmente investiga-se a função de um gene através de análise de fenótipos mutantes. Entretanto, análise comparativa de sequências homólogas tem mostrado ser uma abordagem muito eficiente para estudar função de gene [65]. Estudar padrões de mutação através de análise de sequências homólogas é útil não apenas para o estudo de relacionamentos evolucionários, mas também para identificar características estruturais ou funcionais nas sequências de RNA, DNA ou proteína [65].

Alinhamento de sequências é, sem dúvida, a tarefa mais comum no contexto de bioinformática [180]. Diversos procedimentos necessitam de comparação de sequências, desde buscas em bancos de dados [12] até predição de estrutura secundária [207]. Sequências podem ser comparadas aos pares numa varredura por sequências homólogas em uma base de dados ou podem ser alinhadas simultaneamente, construindo o chamado MSA (do inglês, *Multiple Sequence Alignment*), para visualização do efeito da evolução em uma família inteira de proteínas.

MSAs tornam possível a indagação de uma grande variedade de questões biológicas importantes. Por exemplo, árvore filogenética é um dos instrumentos utilizado para elucidar relações evolucionárias que existem entre organismos. Atualmente, a construção de árvores filogenéticas altamente precisas fazem uso de dados moleculares e a geração de um MSA é uma etapa essencial de suas construções. O papel do alinhamento múltiplo é prover uma estimativa muito precisa das distâncias entre pares e, assim, possibilitar estimar a confiabilidade de cada ramo através de *bootstrapping* [72].

MSAs também possibilitam a identificação de *motifs* preservados pela evolução. *Motifs* desempenham um papel importante na estrutura e função de um grupo de proteínas relacionadas. Ao trabalhar com dados experimentais, esses *motifs* constituem um meio muito poderoso de caracterizar funções ainda não determinadas de sequências [180].

Predição de estrutura é outro importante uso de alinhamentos múltiplos. Predição de estrutura secundária e terciária objetivam a predição do papel que um dado resíduo desempenha na estrutura da proteína (oculto ou exposto, hélice ou fita, etc.). Predições de estruturas secundárias baseadas em uma única sequência possuem uma baixa precisão [81], enquanto que predições baseadas em MSAs são mais precisas [129, 206, 207]. Isso ocorre porque padrões de substituição observados em uma coluna refletem diretamente os tipos de características impostas naquela posição no curso da evolução. No contexto de determinação de estrutura terciária ou predição de contatos não locais, alinhamentos múltiplos podem também ajudar na identificação de mutações correlacionadas. Essa abordagem tem dado resultados limitados quando aplicada a proteínas [83], mas tem conseguido sucesso em análises de RNA [103].

O alinhamento de sequências homólogas consiste na tentativa de posicionar resíduos (nucleotídeos ou aminoácidos) em colunas que derivam de um resíduo de um ancestral comum [65]. Isso é obtido pela introdução de *gaps*, que representam *indels* (inserções ou remoções), nas sequências. Dessa forma, um alinhamento é um modelo hipotético de mutações (substituições, inserções ou remoções) que ocorreram durante a evolução da sequência. O melhor alinhamento será aquele mais parecido com um cenário evolucionário.

No contexto do alinhamento de duas sequências, Needleman e Wunsch em 1970 [174] descreveram um algoritmo que constrói um alinhamento global ótimo através de programação dinâmica. Para sequências de tamanhos m e n, tal algoritmo tem $O(mn)$ como complexidade de tempo e memória. Posteriormente, Smith e Waterman [226] descreveram uma variação desse algoritmo para alinhamento local, que tem a mesma complexidade. Por alinhamento global entende-se o alinhamento das sequências em suas extensões completas. No alinhamento local só são alinhadas as regiões de maior similaridade entre as sequências. Além desses algoritmos, há soluções heurísticas que, apesar de não garantirem a melhor solução, obtêm bons resultados em um tempo menor. Um exemplo é o FASTA [152], que limita a programação dinâmica a uma faixa, de largura parametrizada, na diagonal da matriz. Ele parte do pressuposto de que o alinhamento ótimo não foge muito da diagonal principal da matriz. Um outro exemplo é o BLAST [12].

Para alinhar mais que duas sequências, é possível adaptar o algoritmo de Needleman e Wunsch, porém essa solução ficará limitada a um pequeno número de sequências não muito longas [151]. Esse algoritmo requer $\Omega(2^n l^n)$ de tempo e $\Omega(l^n)$ de espaço, onde n é o número de sequências e l o comprimento de cada uma delas, para computar um alinhamento ótimo. Essa complexidade pode ainda ser mais elevada caso a penalidade para *gaps* não seja linear [65].

Alinhamento múltiplo de sequências é uma tarefa complexa que implica em três dificuldades técnicas distintas: escolha das sequências, escolha da função objetivo e otimização da função. Os métodos tratados nesse contexto, geralmente, fazem sentido apenas se

assumido que o conjunto de sequências é composto por sequências homólogas e, quando se trata de métodos de alinhamento global, há ainda o requisito de que as sequências sejam semelhantes em todo, ou quase todo, seu comprimento para que o alinhamento gerado tenha alguma relevância. Para os casos em que não se cumprem esses requisitos, deve se considerar o uso de métodos para computação de MSAs locais, tais como: Gibbs Sampler [147], Match-Box [57] e MACAW [215]. Para maiores informações sobre os problemas que podem surgir devido a uma má escolha das sequências a alinhar, consultar trabalho de Henikoff [105]. É comum o uso de uma das ferramentas BLAST (WU-BLAST, PSI-BLAST, Gapped BLAST, etc.) [14] para avaliar o nível de relacionamento entre as sequências do conjunto. Tais ferramentas utilizam modelos estatísticos, desenvolvidos por Altschul e Karlin [133], para estimar o grau de relacionamento entre as sequências. É importante, entretanto, salientar que esses modelos são uma mera aproximação da realidade biológica.

A função objetivo (OF, do inglês *Objective Function*) deve permitir a quantificação da qualidade de um alinhamento. Como consequência, permite comparações e a seleção do alinhamento que possua um maior significado biológico. Dada uma OF "perfeita", o alinhamento ótimo matemático deveria ser um alinhamento ótimo biológico [180]. Na prática, tal OF "perfeita" não existe e mesmo que existisse a tarefa de demonstrar que é "perfeita" já se trata de um grande problema. Na teoria, uma OF deveria incorporar tudo o que é conhecido acerca das sequências, incluindo sua estrutura, função e história evolucionária. Entretanto, esse tipo de informação nem sempre está disponível e, mesmo que esteja, é de difícil utilização [180]. Dessa forma, é comum computar similaridade entre sequências através da simples função de soma dos pares com pesos e função afim para penalidade de *gaps* [13]. Embora essa OF seja claramente incorreta do ponto de vista evolucionário [13], por assumir que toda sequência é ancestral de toda outra no conjunto, é popular em métodos conhecidos para construção de MSAs [91, 151, 245], devido a facilidade de implementação. Em 2000, Gonnet e colegas propuseram uma variação de soma dos pares que parece ser mais fidedigna aos eventos evolucionários [87].

Algumas soluções fazem uso de OFs que parecem ser menos sensíveis à penalidade para *gaps* graças a incorporação de informações locais. Nesse contexto pode-se citar o DiAlign [168] e o T-COFFEE [183]. O primeiro inclui avaliação baseada em segmento e o segundo usa uma OF baseada em consistência. Outros trabalhos fazem uso de HMMs (do inglês *Hidden Markov Models*) [26, 104], que descrevem o MSA em um contexto estatístico, usando uma abordagem Bayesiana. Embora, de um ponto de vista formal, esses métodos (HMM) parecerem soluções mais atrativas, suas performances para alinhamentos *ab initio* decepcionam [180]. Há, entretanto, relato de um trabalho [134] que exibiu uma solução que faz uso de HMM e foi capaz de produzir resultados melhores que o Clustal W [245], método para computação de MSA largamente utilizado [180]. Outros métodos

baseados em estatísticas que tentam associar um *P-value* ao alinhamento múltiplo foram descritos [111, 147]. Infelizmente essas medidas são restritas a MSAs sem *gaps*. Em um mundo ideal, uma OF perfeita deveria existir para todas as situações. Na prática, isso não ocorre [180] e, assim, o usuário terá sempre de tomar a decisão de qual o método mais apropriado para o conjunto de sequências do qual dispõe.

Mesmo assumindo que se tem um conjunto adequado de sequências e uma OF biologicamente perfeita, a computação de um alinhamento múltiplo matematicamente ótimo é ainda uma tarefa complexa. O custo computacional de um método exato para computação de MSAs é tão elevado que inviabiliza seu uso para conjuntos de dados reais. É um problema que, inclusive, já foi provado ser NP-Difícil [131, 263]. Dessa forma, os algoritmos para computação de MSAs utilizados na prática são heurísticas, que não garantem um alinhamento ótimo como resultado.

Neste trabalho buscou-se o desenvolvimento de soluções heurísticas eficientes para a computação de alinhamentos múltiplos de sequências de proteína. Nessa busca empregou-se diversas abordagens, que são introduzidas no Capítulo 2. Ainda neste capítulo, são apresentados conceitos pertinentes ao contexto de MSAs, assim como o método utilizado para avaliar a qualidade dos alinhadores desenvolvidos.

Em um primeiro momento do trabalho, foram construídos 342 alinhadores múltiplos utilizando a abordagem progressiva. Esta, que é uma abordagem amplamente utilizada para a construção de MSAs, consiste em três etapas. Na primeira delas é computada uma matriz de distâncias. Em seguida, uma árvore guia é gerada com base na matriz e, finalmente, o MSA é construído através de alinhamentos de pares, cuja ordem é definida pela árvore. Os alinhadores desenvolvidos combinam diferentes métodos aplicados a cada uma das etapas.

Para a computação das matrizes de distâncias foram desenvolvidos dois métodos, que são capazes também de gerar alinhamentos de pares de sequências. Em ambos houve a preocupação de realizar uma penalização adequada dos *gaps*. Um deles constrói o alinhamento com base em alinhamentos locais e o outro utiliza uma função logarítmica para a penalização de *gaps*. Foram utilizados ainda quatro métodos, para computação de matrizes de distâncias, disponíveis numa ferramenta chamada PHYLIP [73]. Para a geração das árvores guias foram utilizados os métodos clássicos UPGMA [228] e *Neighbor Joining* [210]. Usou-se implementações disponíveis em uma ferramenta, para computação estatística e produção de gráficos, chamada R [82] e no pacote APE (*Analysis of Phylogenetics and Evolution*) [192].

Já para a construção do alinhamento múltiplo, foram implementados os métodos seleção por bloco único e seleção do par mais próximo. Estes, que se destinam à seleção do par de alinhamentos a agrupar no ciclo corrente, são comumente utilizados para tal tarefa. Para o agrupamento de um par de alinhamentos, foram implementados 12 métodos

inspirados em métodos comumente utilizados – alinhamento de consensos e alinhamento de perfis.

Foram feitas todas as combinações possíveis entre os métodos, resultando em 342 alinhadores. Eles foram avaliados quanto à qualidade dos alinhamentos que geram e avaliou-se também o desempenho dos métodos, utilizados em cada etapa. Este trabalho é detalhado no Capítulo 3.

Em seguida, foram realizadas avaliações no contexto de alinhamento baseado em consistência. Nesta abordagem, considera-se MSA ótimo aquele que está de acordo com a maioria dos alinhamentos ótimos para os $n(n-1)/2$ alinhamentos de pares contidos no MSA. Alterações foram realizadas em um alinhador múltiplo conhecido, MUMMALS [196], que usa a abordagem. As modificações foram feitas no método de contagem k-mer, que é utilizado na computação da matriz de distâncias, assim como, em um outro momento, substituiu-se a parte inicial do algoritmo. Foram alterados os métodos para computação da matriz de distâncias e para geração da árvore guia por outros que foram bem avaliados nos testes realizados para a abordagem progressiva. No total, foram implementadas e avaliadas 89 variações do algoritmo do MUMMALS e, apesar do alinhador original já produzir alinhamentos de alta qualidade, melhoras significativas foram alcançadas. Este trabalho é detalhado no Capítulo 4.

O trabalho foi concluído com a implementação e a avaliação de algoritmos iterativos. Estes caracterizam-se pela dependência de outros alinhadores para a produção de alinhamentos iniciais. Ao alinhador iterativo cabe a tarefa de refinar tais alinhamentos através de uma série de ciclos até que haja uma estabilização na qualidade dos alinhamentos. Foram implementados e avaliados dois alinhadores iterativos não estocásticos, assim como um algoritmo genético (GA, do inglês *Genetic Algorithm*) voltado para a construção de MSAs. Nesse algoritmo genético, implementado na forma de um ambiente parametrizável para execução de algoritmos genéticos para construção de MSAs, chamado ALGAe, foram realizadas diversas experiências, que progressivamente elevaram a qualidade dos alinhamentos gerados.

No ALGAe foram incluídas outras abordagens para construção de alinhamentos múltiplos, tais como baseada em blocos, em consenso e em modelos. A primeira foi aplicada na geração de indivíduos para a população inicial. Foram implementados alinhadores baseados em blocos usando duas abordagens distintas e, para uma delas, foram implementadas cinco variações. A segunda foi aplicada na definição de um operador de cruzamento, que faz uso da ferramenta M-COFFEE [260] para realizar alinhamentos baseados em consenso a partir de indivíduos da população corrente do GA, e a terceira foi utilizada para definir uma função de aptidão, que utiliza a ferramenta PSIPRED [129] para predição das estruturas secundárias das sequências. O ALGAe permite a realização de uma grande variedade de novas avaliações. Este trabalho é detalhado no Capítulo 5.

As considerações finais são apresentadas no Capítulo 6. Há ainda três apêndices. No Apêndice A há um glossário para ajudar na leitura do texto e no Apêndice B são detalhados os alinhadores progressivos implementados. Já no Apêndice C é apresentada uma revisão bibliográfica com artigos de grande relevância no contexto de alinhamento múltiplo.

Capítulo 2

Conceitos básicos

Duas sequências são chamadas homólogas quando derivam de um ancestral comum. Geralmente homologia é inferida por similaridade de sequências, porém, similaridade não reflete necessariamente homologia. O acaso ou convergência evolucionária [63], por exemplo, podem levar a similaridade entre fragmentos curtos de sequências. É possível também haver similaridade em trechos longos de sequências, tal como repetições CA em moléculas de DNA ou domínios ricos em prolina nas proteínas [268]. Geralmente, similaridades entre tais sequências com "regiões de baixa complexidade" não refletem relacionamento evolucionário, mas – na ausência de tais composições tendenciosas – similaridade em uma região extensa frequentemente implica em homologia. Testes estatísticos podem ser usados para avaliar as chances de uma similaridade observada ocorrer ao acaso e, assim, aceitar ou não a homologia hipotética [11].

Buscar pelo melhor alinhamento consiste em identificar aquele que aparentemente apresenta um melhor cenário evolucionário. Assim, a probabilidade de ocorrência de diferentes eventos mutacionais durante a evolução deve ser levada em consideração ao computar um alinhamento múltiplo. Em alinhamentos, tipicamente três tipos de mutações são considerados: substituições, inserções ou remoções. Os dois últimos eventos são frequentemente indistinguíveis e são comumente referenciados como *indels*. A probabilidade de substituição de um aminoácido por outro depende da estrutura do código genético (por exemplo: o número de mutações necessárias para transformar um códon em outro) e também do efeito fenotípico da mutação. Substituições de um aminoácido por outro com propriedades bioquímicas similares geralmente não têm grande efeito na estrutura e consequentemente na função da proteína. Assim, durante a evolução, substituições conservativas são relativamente frequentes se comparadas a outras substituições. É importante notar que a probabilidade de substituição de um aminoácido por outro depende da distância evolucionária entre as sequências.

Vários métodos foram propostos para a construção de séries de matrizes que estimam a

probabilidade de todas as possíveis substituições de aminoácidos para diferentes distâncias evolucionárias [56, 85, 106, 217]. As séries de matrizes mais utilizadas são PAM [56, 217] e BLOSUM [106]. Geralmente os alinhadores permitem ao usuário selecionar a matriz de substituição a ser utilizada.

Probabilidades de substituição também variam ao longo das sequências de acordo com ambientes locais de aminoácidos na proteína dobrada (por exemplo, α-hélice ou folha-β). Matrizes de substituição de ambiente específico foram desenvolvidas [191], porém são raramente utilizadas nas ferramentas disponíveis atualmente. Em sequências de DNA, probabilidades de substituições variam de acordo com as bases. Transições, trocas de bases de uma mesma classe (purínicas ou pirimidínicas), são geralmente mais frequentes que transversões, trocas de bases de classes distintas. Assim, alinhadores múltiplos geralmente propõem um parâmetro de peso mais rígido para transversões do que para transições. Probabilidade de substituição de nucleotídeos também depende das bases vizinhas. Por exemplo, em vertebrados, C no dinucleotídeo CG é extremamente mutável [22, 112]. Os alinhadores correntes ainda não fazem uso de tal informação.

A probabilidade de ocorrência de um *indel* depende do tamanho dele. Assim, ao computar um alinhamento, frequentemente estima-se penalidades (p) associadas com *gaps* usando um modelo linear ou "afim" tal como $p = a + bL$, onde L é o tamanho do *gap*, a a penalidade para abertura e b a penalidade para extensão do *gap*. Entretanto, análises de alinhamentos têm mostrado que esse modelo subestima a probabilidade de *indels* longos [85, 86, 100]. Penalidades mais realísticas para *indels* podem ser estimadas com modelos tais como $p = a+b\times log(L)$ [65]. Devido à complexidade computacional, tais modelos não têm sido implementados nos alinhadores comumente utilizados. Felizmente outras abordagens para alinhar sequências com grandes *indels* foram propostas, como poderá ser visto na Seção 2.3.4, que apresenta os métodos baseados em blocos. Em proteínas a probabilidade de ocorrência de *indels* também depende do grau de divergência entre as sequências [85, 86] e da composição das moléculas. Por exemplo, *indels* são mais frequentes em laços externos que no núcleo da estrutura. É importante salientar que, na maioria dos programas, parâmetros padrões para penalidades de *gap* têm sido definidos para proteínas globulares típicas, que podem não ser os melhores para outras sequências [65].

Considere que uma sequência é um conjunto ordenado de letras de um alfabeto Σ. Um alinhamento de n sequências $S_1, S_2, \ldots, S_n$ pode ser definido como uma matriz $A = a(S_1, S_2, \ldots, S_n)$, onde cada entrada A_{ij} é uma letra de Σ ou um símbolo nulo (frequentemente representado por "-"). A linha i de A, depois dos *gaps* (símbolos nulos) removidos, é a sequência S_i. Em princípio, a pontuação de um alinhamento múltiplo deve refletir o grau de semelhança de suas sequências, de acordo com um modelo evolucionário dado. Há diversas formas de mensurar a pontuação de um alinhamento. No modelo

mais simples, a pontuação de um alinhamento de n sequências é definido como a soma das pontuações de suas colunas. Um defeito desse modelo é considerar cada coluna do alinhamento independentemente de seu contexto. Por exemplo, um *gap* de tamanho L é considerado o mesmo que L *indels* independentes.

Em modelos mais realísticos, um *gap* é considerado um único evento mutacional e associado com uma penalidade proporcional ao seu tamanho. Em tais modelos, pontuação para alinhamento de par é definida como a soma das pontuações de substituições e penalidades para *gaps*. Para alinhamentos múltiplos, uma solução bastante utilizada, conhecida como soma dos pares [47], consiste em calcular a pontuação do alinhamento múltiplo a partir da soma das pontuações das projeções de $n(n-1)/2$ alinhamentos de pares, desprezando-se os alinhamentos de dois *gaps*. Soma dos pares simples, entretanto, pode ser inapropriada quando alguns grupos de sequências são extremamente ou pouco representados em uma família. Esse defeito pode ser corrigido pela introdução de um sistema de pesos apropriado [10, 92], que associa um peso a cada sequência. Assim, pode-se dar um peso menor às sequências de grupos super-representados. Outra solução consiste em usar uma função de pontuação baseada em uma árvore evolucionária. As folhas da árvore são as sequências que se deseja alinhar, e os nós internos são suas sequências ancestrais hipotéticas. Apesar de parecer uma função melhor que a soma dos pares, o problema de alinhamento usando árvore evolucionária é também difícil [127].

Na Seção 2.1, conceitos biológicos importantes no contexto de MSA são apresentados. O algoritmo de Needleman e Wunsch [174] para alinhamento global de um par de sequências, que será citado diversas vezes ao longo do texto, é detalhado na Seção 2.2. Na Seção 2.3 abordagens empregadas para alinhamento múltiplo de sequências são apresentadas. Métodos utilizados para avaliação de alinhamentos, e consequentemente de alinhadores, são expostos na Seção 2.4. Já na Seção 2.5 são esclarecidos aspectos a respeito dos algoritmos implementados.

2.1 Conceitos biológicos

Um dos tipos de ácido nucléico presente nos organismos vivos é o ácido ribonucléico, também conhecido por RNA. O RNA é uma molécula composta por uma cadeia de moléculas mais simples, chamadas nucleotídeos [219]. Os nucleotídeos, numa molécula de RNA, podem ser adenina (A), guanina (G), citosina (C) ou uracila (U).

O outro tipo de ácido nucléico presente nos organismos vivos é o ácido desoxirribonucléico, também conhecido por DNA. As moléculas de DNA são semelhantes às moléculas de RNA, porém são observadas quatro diferenças principais. A primeira está na molécula de açúcar da qual é composta. No DNA, o açúcar presente é o $2'$-desoxirribose. Já no RNA é a ribose. A segunda está na substituição das uracilas por timinas.

A terceira está no fato de que as moléculas de DNA estão no espaço em forma de dupla fita. Tais fitas são complementares e, assim, possuindo uma delas, é possível calcular a outra. Uma base A sempre é pareada com uma base T ou, em outras palavras, A é o complemento de T e vice-versa ou, ainda, A e T são bases complementares. As bases C e G também são complementares. Seja $f_1 = TACGAC$ uma das fitas de uma molécula de DNA. Pode-se obter sua fita complementar f_2 através do complemento reverso de f_1: primeiro inverte-se a ordem de f_1 e obtem-se $f_1' = CAGCAT$, depois troca-se cada base por seu complemento, obtendo assim $\overline{f_1} = GTCGTA$. Comumente uma molécula de DNA é simplesmente definida por sua sequência de bases em uma das fitas na chamada direção canônica $5' \mapsto 3'$.

A última delas está no fato de que a única função do DNA é armazenar informação, diferentemente do RNA, que possui diversas funções. As diferenças apresentadas pelas moléculas de DNA, em relação às moléculas de RNA, as tornam mais estáveis e levam a uma menor variedade de estruturas tridimensionais.

O tamanho das moléculas de DNA é normalmente definido em número de pares de bases, que é denotado por *bp*. Na natureza, moléculas de DNA são muito compridas. Em células humanas, as moléculas de DNA possuem dezenas ou centenas de milhões de pares de bases. Na Tabela 2.1 são listados os cromossomos humanos juntamente com o tamanho das moléculas.

As proteínas encontradas nos organismos vivos são de diversos tipos e podem assumir diversas funções. Elas podem, por exemplo, atuar como estruturas em diversas partes do organismo ou catalizadoras de reações químicas (enzimas). Da mesma forma que os ácidos nucléicos, as proteínas são moléculas compostas por uma cadeia de moléculas mais simples, os aminoácidos. Na natureza, são comumente encontrados 20 diferentes aminoácidos, que são listados na Tabela 2.2.

O tamanho de uma proteína pode ser mensurado pelo número de aminoácidos. Trezentos aminoácidos é o tamanho médio apresentado pelas proteínas, embora seu tamanho possa variar bastante, podendo possuir 100 ou 5000 aminoácidos [219].

Quando uma proteína é estudada, o foco concentra-se na descoberta de sua função. Para se atingir tal objetivo, a determinação de sua forma pode ser uma etapa a ser executada. Partindo de sua sequência de aminoácidos, que é conhecida como estrutura primária, observa-se como essa cadeia dobra-se dando origem a uma estrutura tridimensional conhecida por estrutura secundária. Essas estruturas (secundárias), por sua vez, agrupam-se formando a estrutura terciária que, finalmente, agrupa-se a estruturas terciárias de diferentes proteínas e origina a estrutura quaternária. Na estrutura secundária há definição de estruturas tais como: hélices e folhas. Na estrutura terciária a posição tridimensional de cada átomo é observada. Na estrutura quaternária observa-se como as subunidades se ligam, caso a proteína seja composta de mais que uma.

Tabela 2.1: Lista de cromossomos humanos com o respectivo tamanho da molécula [2].

Cromossomo	**Tamanho (Mbp)**
1	247,0
2	243,0
3	200,0
4	191,0
5	181,0
6	171,0
7	159,0
8	146,0
9	140,0
10	135,0
11	134,0
12	132,0
13	114,0
14	106,0
15	100,0
16	89,0
17	79,0
18	76,0
19	64,0
20	62,0
21	46,9
22	50,0
X	155,0
Y	58,0

O conjunto de moléculas de ácido nucléico que definem todas as proteínas de um organismo é conhecido como genoma. Cada célula de um organismo – com exceção das células germinativas e, nos humanos, as hemácias e as células do sistema imune – possui uma cópia de todo o genoma, que será chamado simplesmente de DNA. No DNA há trechos que codificam as informações necessárias para construir uma dada proteína, tais trechos são conhecidos por genes.

Nas células existem mecanismos capazes de reconhecer onde começam e terminam os genes. Com o auxílio de uma enzima chamada transcriptase, é feita uma cópia do gene em uma molécula de RNA que recebe o nome de RNA mensageiro ou mRNA. Durante tal processo, chamado de transcrição, são substituídas as bases T por U.

Posteriormente, essa molécula de mRNA irá migrar até uma organela celular chamada ribossomo, que é basicamente composta de proteínas e um tipo de RNA chamado RNA

Tabela 2.2: Os vinte aminoácidos comumente encontrados nas proteínas [219].

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

ribossomal ou rRNA. Essa organela tem como função a síntese protéica. Usando como entrada uma molécula de mRNA e moléculas de um tipo de RNA chamado de RNA transportador ou tRNA, o ribossomo produz uma proteína.

Agrupando os nucleotídeos três a três, formando o que são chamados de códons, os ribossomos promoverão a ligação desses códons a moléculas de tRNA que, por sua vez, estão ligadas a um aminoácido, cada uma. A Tabela 2.3 apresenta o código genético padrão de conversão de códons em aminoácidos. Nem todos os organismos utilizam exatamente essa tabela padrão, mas o procedimento é similar para todos os organismos. Dessa forma, a cadeia de aminoácidos vai sendo formada. Ao final do processo, as moléculas de mRNA e os tRNAs são liberados para degradação e futura utilização. Tem-se também a proteína montada. Esse processo é chamado de tradução.

Observe que há, na Tabela 2.3, três ocorrências do chamado `STOP` códon. É através desse códon que os mecanismos celulares reconhecem o final de um gene. O gene sempre inicia com uma metionina (AUG). Na Figura 2.1 é apresentado o fluxo da informação genética na célula, que é conhecido como dogma central da biologia molecular.

Tabela 2.3: O código genético padrão de mapeamento de códons em aminoácidos [219].

Primeira posição	Segunda posição				Terceira posição
	G	A	C	U	
G	Gly	Glu	Ala	Val	G
	Gly	Glu	Ala	Val	A
	Gly	Asp	Ala	Val	C
	Gly	Asp	Ala	Val	U
A	Arg	Lys	Thr	Met	G
	Arg	Lys	Thr	Ile	A
	Ser	Asn	Thr	Ile	C
	Ser	Asn	Thr	Ile	U
C	Arg	Gln	Pro	Leu	G
	Arg	Gln	Pro	Leu	A
	Arg	His	Pro	Leu	C
	Arg	His	Pro	Leu	U
U	Trp	STOP	Ser	Leu	G
	STOP	STOP	Ser	Leu	A
	Cys	Tyr	Ser	Phe	C
	Cys	Tyr	Ser	Phe	U

2.2 Alinhamento de um par de sequências

O algoritmo de Needleman e Wunsch [174] é normalmente empregado para o alinhamento global de um par de sequências. Ele utiliza programação dinâmica e tem complexidade $O(mn)$ para tempo e espaço, onde m e n são os comprimentos das sequências. O algoritmo possui as seguintes etapas:

1. Preenchimento da matriz de pontuação, M; e
2. Recuperação do alinhamento a partir da matriz.

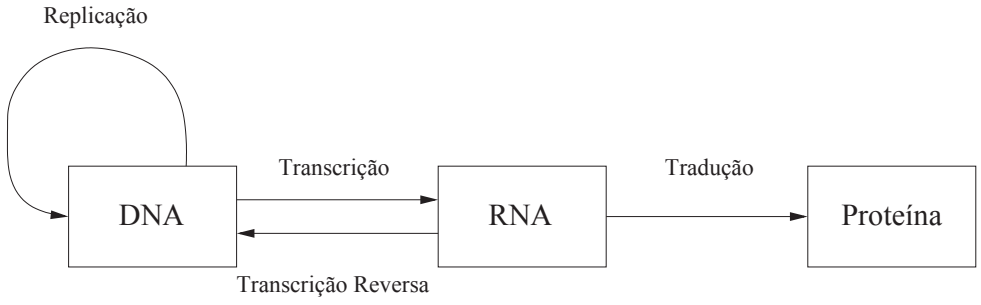


Figura 2.1: Fluxo da informação genética em uma célula [219].

Cria-se a matriz M com $m+1$ linhas por $n+1$ colunas. Inicializa-se os valores da primeira coluna e da primeira linha. Na primeira linha, para $0 \leq j \leq n$, atribui-se $j \times gap$ para a célula na coluna j. Na primeira coluna, para $0 \leq i \leq m$, atribui-se $i \times gap$ para a célula na linha i.

Seja $score(i, j)$ a pontuação do alinhamento do i-ésimo resíduo da primeira sequência com o j-ésimo resíduo da segunda. Quando dois resíduos similares são alinhados, diz-se que houve um *match*. Já quando os resíduos são distintos, tem-se um *mismatch*. Na implementação do algoritmo pode-se definir um valor para *match* e outro para *mismatch*, ou ainda usar uma matriz de distâncias de séries tais como PAM ou BLOSUM. O restante da matriz é definida, da esquerda para a direita e de cima para baixo, segundo a Equação 2.1.

$$M[i,j] = max \begin{cases} M[i-1,j] + gap \\ M[i-1,j-1] + score(i,j) \\ M[i,j-1] + gap \end{cases} \tag{2.1}$$

O valor da célula assume o máximo entre esses três valores. O primeiro deles significa o alinhamento do j-ésimo resíduo da segunda sequência com um *gap*. O segundo é o alinhamento dos dois resíduos e o terceiro é o alinhamento do i-ésimo resíduo da primeira sequência com um *gap*. Na Tabela 2.4 é apresentada a matriz M preenchida para as sequências `TCGAACTGCT` e `TGGACTCT`. Utilizou-se $match = 1$, $mismatch = -1$ e $gap = -2$.

Tabela 2.4: Matriz de pontuação M para alinhamento global preenchida para as sequências `TCGAACTGCT` e `TGGACTCT`. Utilizou-se $match = 1$, $mismatch = -1$ e $gap = -2$. Em negrito está o caminho da recuperação, que indica o alinhamento gerado pela matriz.

		T	C	G	A	A	C	T	G	C	T
	0	-2	-4	-6	-8	-10	-12	-14	-16	-18	-20
T	-2	**1**	**-1**	-3	-5	-7	-9	-11	-13	-15	-17
G	-4	-1	0	**0**	-2	-4	-6	-8	-10	-12	-14
G	-6	-3	-2	1	**-1**	-3	-5	-7	-7	-9	-11
A	-8	-5	-4	-1	2	**0**	-2	-4	-6	-8	-10
C	-10	-7	-4	-3	0	1	**1**	-1	-3	-5	-7
T	-12	-9	-6	-5	-2	-1	0	**2**	**0**	-2	-4
C	-14	-11	-8	-7	-4	-3	0	0	1	**1**	-1
T	-16	-13	-10	-9	-6	-5	-2	1	-1	0	**2**

A recuperação é feita traçando um caminho da célula do canto inferior direito até a célula do canto superior esquerdo. Iniciando da célula do canto inferior direito, busca a célula que deu origem ao seu valor. Caso o seu valor tenha sido gerado por $M[i-1,j]+gap$, segue para a célula acima. Caso o seu valor tenha sido gerado por $M[i,j-1]+gap$, segue

para a célula a esquerda. Se o seu valor foi gerado por $M[i-1, j-1] + score(i, j)$, segue para a célula na diagonal. A iteração segue em busca da célula que deu origem ao valor da célula corrente até chegar na célula do canto superior esquerdo.

O alinhamento é construído de trás para a frente, seguindo o caminho conforme descrito acima. Se navegar para a célula acima, significa que alinhou um resíduo da sequência à esquerda com um *gap*. Se navegar para a célula à esquerda, significa que alinhou um resíduo da sequência do topo com um *gap*. Já se navegar na diagonal, significa que alinhou um par de resíduos. O alinhamento produzido pela matriz é o seguinte:

$$\begin{array}{cccccccccc} T & C & G & A & A & C & T & G & C & T \\ T & - & G & G & A & C & T & - & C & T \end{array}$$

Com pequenas alterações na inicialização da matriz e na recuperação do alinhamento, é possível produzir alinhamentos locais. Neste caso, só se alinha o trecho de maior similaridade. Para isso, inicializa-se a primeira linha e a primeira coluna com 0's e inclui-se 0 na Equação 2.1, o que não permitirá valores negativos na matriz. A navegação na matriz, para construção do alinhamento local, inicia no maior valor presente na matriz e segue até encontrar 0. Esse maior valor é a pontuação do alinhamento local. Esse algoritmo foi descrito por Smith e Waterman [226].

Há ainda alinhamento semi-global. Neste caso, *gaps* no início ou no final do alinhamento não são penalizados. Isso irá privilegiar o alinhamento do prefixo de uma sequência com o sufixo da outra. Tal variação é empregada, por exemplo, em montagem de fragmentos. Pode-se alterar o algoritmo de Needleman e Wunsch e gerar um alinhamento semi-global. A primeira coluna e a primera linha são inicializadas com 0's. A navegação na matriz, para construção do alinhamento semi-global, inicia no maior valor presente na última linha ou última coluna e pára quando chegar na primeira linha ou na primeira coluna. Observe que o alinhamento deve conter as sequências originais por inteiro e assim deve-se alinhar o prefixo e/ou sufixo ausentes com *gaps*.

2.3 Abordagens empregadas para MSA

Na tentativa de computar alinhamentos múltiplos exatos, Carillo e Lipman [47] propuseram um algoritmo *branch and bound* para computar um alinhamento cuja pontuação soma dos pares é máxima. Este faz uso de um limiar superior para a pontuação e, assim, impõe limites para o espaço e tempo. Essa abordagem foi implementada no alinhador MSA [151]. Stoye e colegas [235] descreveram um novo algoritmo por divisão e conquista, chamado DCA, que estendeu as capacidades do MSA original. O algoritmo DCA fragmenta as sequências em segmentos que são suficientemente pequenos para utilizar o MSA. Depois, resta ao DCA a tarefa de montar o alinhamento a partir dos sub-alinhamentos

gerados pelo MSA. Testes no BAliBASE [180, 248], uma ferramenta para *benchmark* em MSA amplamente utilizada pela comunidade científica, indicaram que a estratégia DCA produz resultados ligeiramente melhores que o Clustal W [245], que provavelmente é o alinhador múltiplo mais conhecido. É importante salientar, entretanto, que limitações ainda existem e que o DCA não é capaz de realizar o alinhamento de qualquer entrada, devido a essas limitações. Posteriormente, uma implementação iterativa do DCA, chamada OMA [204], foi descrita. Com ela pretende-se tornar a estratégia DCA mais rápida e reduzir os requisitos de memória. Ainda assim, as limitações para o conjunto de sequências de entrada, apesar de amenizadas, persistem.

Computar MSAs já foi provado ser um problema NP-Difícil [131, 263]. Dessa forma, a construção de MSAs exatos fica bastante limitada no que se refere ao tamanho da entrada (número e comprimento das sequências). Sendo assim, algoritmos de aproximação e diversas heurísticas, usando as mais diversas abordagens, têm sido descritos para o problema de construção de MSAs.

O primeiro algoritmo de aproximação foi proposto por Gusfield [102]. Tem um fator de aproximação $2 - 2/n$ e complexidade $O(n^2k^2)$ para n sequências com comprimento k. Pevzner [200] melhorou o algoritmo, alcançando um fator de $2 - 3/n$ e complexidade $O(n^3k^3 + n^4)$. Posteriormente, Bafna, Lawler e Pevzner [19] apresentaram um algoritmo com fator $2 - l/n$ e complexidade $O(n^{l+1}(2^n + n\ m(l, k)))$ para qualquer $l < n$ e complexidade $m(l, k)$ para obter um alinhamento múltiplo ótimo de l sequências de comprimento k.

Nos algoritmos heurísticos destacam-se seis abordagens, que são listadas na Tabela 2.5. As soluções não pertencem necessariamente a uma única abordagem. Por exemplo, ProbCons utiliza as abordagens baseada em consistência e progressiva. Embora não haja garantias, observa-se que os resultados são muito úteis na prática e frequentemente estão muito próximos da solução exata [65].

Desde 1990 houve uma grande evolução da área com a introdução de diversos novos algoritmos e novos métodos de avaliação dos algoritmos. Nessa evolução destaca-se o crescente interesse em estratégias iterativas e uso de esquemas de pontuação baseados em consistência [180]. Outro ponto importante em MSA ao longo dessa evolução são os métodos baseados em HMM [26, 104]. Para maiores informações acerca de métodos HMM para construção de MSAs, consulte o livro de Durbin e colegas [64].

2.3.1 Algoritmos progressivos

Alinhamento progressivo é uma das maneiras mais simples e efetivas para a realização de alinhamentos múltiplos com um pequeno requisito de tempo e memória. Essa abordagem foi inicialmente descrita por Hogeweg e Hesper [116] e depois reinventada por Feng e

Tabela 2.5: Abordagens heurísticas para alinhamento múltiplo.

Abordagem	Descrição
Progressiva	Caracteriza-se por construir MSAs através de alinhamentos de pares. Maiores detalhes na Seção 2.3.1.
Iterativa	Necessita de métodos para gerar MSA(s) inicial(is). Cabe a ele a tarefa de refiná-lo(s). Maiores detalhes na Seção 2.3.2.
Baseada em consistência	Considera MSA ótimo aquele que está de acordo com a maioria dos alinhamentos ótimos de pares. Maiores detalhes na Seção 2.3.3.
Baseada em consenso	Recebe MSAs como entrada e realiza a combinação deles de forma que o alinhamento resultante seja consistente com os MSAs de entrada. Maiores detalhes na Seção 2.3.4.
Baseada em modelos	Utiliza alinhamentos estruturais ou de perfis, construídos a partir das sequências de entrada, para melhorar a qualidade de MSAs com sequências de entrada de menor grau de similaridade. Maiores detalhes na Seção 2.3.4.
Baseada em blocos	Utiliza blocos, alinhamentos locais, como âncoras para guiar a construção do MSA. Maiores detalhes na Seção 2.3.4.

Doolittle [74] e Taylor [239]. Diversos pacotes para construção de MSAs são baseados nessa abordagem, dentre eles: Pileup (componente do pacote GCG [58]), MultiAlign [53] e Clustal W [245]. Essa abordagem oferece um bom desempenho quando as sequências são homólogas e relativamente bem conservadas [74, 239]. O principal problema da abordagem progressiva é a sua natureza gulosa, onde erros cometidos em etapas iniciais do processo não serão mais corrigidos [65].

Estratégias de otimização iterativas têm sido propostas para resolver tal problema, tais como: RIW ou DNR [93]. Em testes, esses métodos apresentaram melhores desempenhos que o Clustal W, por exemplo. Porém, mesmo sendo mais rápidos que algoritmos ótimos, são ainda muito lentos para entradas grandes.

Alinhamento progressivo consiste em construir um alinhamento múltiplo a partir de alinhamentos de pares. Nessa abordagem é possível identificar três passos: computação dos alinhamentos de pares, construção da árvore guia a partir das distâncias de pares e construção do alinhamento múltiplo guiado pela árvore. As diferenças entre as ferramentas para MSA que fazem uso da abordagem progressiva estão nos métodos que utilizam em cada um dos três passos.

Clustal W, por exemplo, permite o uso de programação dinâmica ou métodos heurísticos para a computação dos alinhamentos de pares. Com programação dinâmica, tem-se resultados mais precisos, no entanto, com os métodos heurísticos é possível ganhar em velocidade de processamento. Para a construção da árvore guia há diversos métodos, dentre eles UPGMA [228] e *Neighbor Joining* [210]. É conhecido um problema no UPGMA, onde uma ordem de ramificação incorreta pode ser produzida quando as taxas de substituição variam em diferentes linhagens. Esse problema levou a utilização de *Neighbor Joining*, em vez de UPGMA, no Clustal W[1] [65]. No terceiro passo é preciso, inicialmente, determinar um método para a seleção do par de sequências (ou alinhamentos) a agrupar. O principal problema desse passo, entretanto, consiste no alinhamento de dois alinhamentos. O método mais simples é reduzir cada alinhamento a uma sequência consenso e usar um algoritmo de alinhamento de pares comum. Outro método, utilizado pela maioria dos programas, é representar alinhamento através de perfis, que reduzem colunas a distribuições das frequências de cada letra. Alinham-se dois perfis de forma similar a de duas sequências por programação dinâmica.

O esquema de pontuação é o componente de maior influência nos resultados dos algoritmos progressivos. Tais esquemas podem ser divididos em duas categorias: baseados em matriz e baseados em consistência. Os algoritmos baseados em matriz usam uma matriz de substituição para avaliar o custo de relacionar dois símbolos. São exemplos de algoritmos baseados em matriz: Clustal W [245], MUSCLE [68] e Kalign [146]. Os algoritmos baseados em consistência incorporam um maior conjunto de informações na avaliação. Inicialmente tal método foi utilizado no T-COFFEE [183], inspirado no DiAlign [168]. Depois surgiram outros algoritmos que utilizam a mesma idéia. O PCMA [199] reduz os requisitos computacionais exigidos pelo T-COFFEE. O ProbCons [60] adiciona o uso de consistência Bayesiana e HMM. MUMMALS [196] combina o esquema de pontuação do ProbCons com a estratégia do PCMA. Outro exemplo ainda de algoritmo baseado em consistência é o MAFFT [135], que utilizando transformada rápida de Fourier para gerar uma árvore guia e, assim, gera alinhamentos precisos em um tempo curto [136]. Estudos têm indicado que os métodos baseados em consistência são mais precisos que os métodos baseados em matriz, mas geralmente requerem um maior tempo de processamento.

MUSCLE [68, 69] é um pacote de alinhamento progressivo extremamente rápido e preciso. Seu primeiro passo é gerar um esboço de um alinhamento através de uma árvore guia imatura. As distâncias entre os pares de sequências de entrada são estimadas usando contagem k-mer para um alfabeto restrito [67] e, juntamente com um algoritmo de agrupamento, dão origem à árvore guia inicial. MUSCLE implementa a pontuação LE (do inglês,

[1]Nas versões anteriores, Clustal e Clustal V, era utilizado UPGMA. Além destas, ainda há Clustal X, que é uma versão gráfica para Clustal W com ferramentas para visualização de MSAs, e versões mais recentes, como DbClustal, que utiliza a abordagem baseada em modelos, e Clustal Ω [221], que produz alinhamento de boa qualidade com um elevado throughput.

Log Expectation) para alinhar perfis durante o alinhamento progressivo. Uma vez gerado o MSA inicial, passa-se ao refinamento do alinhamento pela geração de uma árvore guia mais precisa, baseando-se no alinhamento inicial. LE tem apresentado bons resultados em buscas por homologia [252]. Nas versões mais recentes do MUSCLE, foi adicionado um refinamento iterativo utilizado pelo ProbCons.

Dentre os algoritmos progressivos, destaca-se o Clustal W [245], que inclui uma variedade de heurísticas altamente especializadas destinadas a máxima exploração de informações acerca das sequências. Com isso, diferencia-se da maioria dos outros alinhadores progressivos por prover penalidade de *gap* local, escolha automática da matriz de substituição, ajuste automático de penalidade para *gap* e retardo do alinhamento de sequência com baixo grau de relacionamento.

A Tabela 2.6 apresenta uma breve descrição de alinhadores progressivos de destaque.

Tabela 2.6: Alinhadores progressivos de destaque.

Alinhador	**Descrição**	**Ano**
Clustal W [245]	Inclui uma variedade de heurísticas altamente especializadas destinadas a máxima exploração de informações acerca das sequências.	1994
MUSCLE [68]	Utiliza um algoritmo de contagem k-mer e um alfabeto comprimido [67] para computar a matriz de distâncias e um algoritmo de agrupamento para gerar a árvore guia. Implementa uma pontuação chamada LE (do inglês, *Log Expectation*) para alinhar perfis.	2004
MAFFT [135]	Utiliza transformada rápida de Fourier para gerar uma árvore guia e, assim, gerar alinhamentos precisos em um tempo curto.	2005
Clustal Ω [221]	Método preciso e veloz, é capaz de alinhar grandes entradas em um tempo curto quando comparado a outros alinhadores. Faz uso de uma vasta quantidade de informações pré-computadas, disponíveis em bases de dados públicas como Pfam [75].	2011

2.3.2 Algoritmos iterativos

Os alinhadores iterativos dependem de um algoritmo capaz de produzir alinhamentos iniciais. Cabe a eles a tarefa de refinar tais alinhamentos através de uma série de ciclos até que não seja mais possível incrementar a qualidade do alinhamento. Métodos iterativos podem ser estocásticos ou não estocásticos. Os métodos mais simples são não estocásticos. Suas

estratégias implicam em extrair sequências, uma por vez, de um alinhamento múltiplo e então realinhá-las [29, 93]. Métodos iterativos estocásticos incluem HMM [143], *simulated annealing* (SA) [139] e algoritmos genéticos (GA) [15, 45, 49, 88, 182, 269]. A principal vantagem desses é permitir uma boa separação conceitual entre processo de otimização e OF. É possível a existência de métodos que usam estratégias mistas, progressiva e iterativa ao mesmo tempo [108].

O primeiro algoritmo iterativo não estocástico data da origem de MSAs [29]. Sua estratégia consiste em realinhamentos para corrigir possíveis erros em estágios anteriores e para tanto faz uso de algoritmos padrões de alinhamento com programação dinâmica [174]. O procedimento é encerrado quando as iterações falham consistentemente na tentativa de melhorar o alinhamento. Esse é o procedimento utilizado pela maioria das estratégias descritas até o início da década de 1990. A principal variação nesses algoritmos encontra-se na forma como as sequências são divididas em dois grupos antes de serem realinhadas. Por exemplo, em AMPS [29] as sequências são escolhidas de acordo com sua ordem de entrada e realinhadas uma a uma. Já no algoritmo de Berger e Munson [36] a escolha é feita de maneira aleatória e as sequências são divididas em dois grupos. A introdução da aleatoriedade na seleção dos grupos torna o algoritmo mais robusto e incrementa sua precisão. Poucos desses métodos iterativos iniciais foram efetivamente avaliados por ferramentas de *benchmark*, o que torna difícil uma estimativa de sua verdadeira significância biológica.

O mais sofisticado algoritmo iterativo baseado em programação dinâmica foi descrito por Gotoh [93] e é conhecido por PRRP. Trata-se de uma estratégia iterativa de duplo aninhamento com aleatorização que otimiza uma pontuação de soma dos pares com peso e função afim para penalidade de *gaps*. Sua originalidade está na otimização simultânea de pesos e alinhamento. A iteração interna otimiza a soma dos pares com peso, enquanto a iteração externa otimiza os pesos que são calculados em uma árvore filogenética estimada do alinhamento corrente [10]. O algoritmo é encerrado quando os pesos convergem. PRRP foi o primeiro programa para alinhamento múltiplo a ser extensivamente avaliado por *benchmark*, usando JOY [164], um banco de dados de alinhamentos estruturais. Posteriormente os resultados foram confirmados no BAliBASE [183, 249]. PRRP apresenta resultados significativamente melhores que a maioria dos métodos progressivos e iterativos disponíveis até meados da década de 1990 [180].

SA [162] foi o primeiro método iterativo estocástico descrito para alinhamento múltiplo. Vários esquemas foram publicados [124, 139] e todos envolvem a mesma cadeia de processos: modifica aleatoriamente um alinhamento, calcula a pontuação, decide se mantém ou descarta a modificação de acordo com a função de aceitação que se torna mais exigente com o aumento no número de iterações. O procedimento é repetido até que seja satisfeito um critério de término. É importante observar que SA tem se mostrado pouco

eficiente, em termos de tempo, para a construção de alinhamentos *ab initio*, mas tem se mostrado uma boa ferramenta para refinamento de alinhamentos.

GAs [117] constituem uma interessante alternativa aos SAs. SAGA [182], assim como SA, é uma caixa preta de otimização na qual qualquer OF pode ser testada. Seu esquema é direto e segue a risca GA simples [84]. SAGA usa a OF definida como sua função de aptidão, suas mutações inserem ou deslocam *gaps* e cruzamentos combinam o conteúdo de dois alinhamentos. Ao todo são 20 operadores concorrendo para executar. Testes têm mostrado que SAGA é capaz de produzir resultados próximos ou similares ao MSA, usando a mesma OF. Os testes também mostraram que GAs são lentos para alinhamentos múltiplos, no uso do dia-a-dia. SAGA foi paralelizado por dois grupos independentes [15, 185] em busca de eficiência, assim como novos métodos foram implementados baseando-se em princípios descritos no SAGA [45, 49, 88]. Zhang e Wong [269] implementaram uma nova ferramenta que faz uso de GAs. Foi observado um alto grau de eficiência da ferramenta, mas deve-se notar que o método é dirigido pela presença de segmentos completamente conservados para guiar a montagem do alinhamento, algo nem sempre realístico.

O pacote Gibbs Sampler [147] faz uso de *Gibbs Sampling*, que é uma outra interessante estratégia iterativa estocástica. Ele implementa um método de alinhamento local que encontra *motifs* sem *gaps* ao longo de um conjunto de sequências não alinhadas. Da perspectiva de alinhamento múltiplo, sua característica mais interessante está em sua OF. O algoritmo objetiva construir um alinhamento com um bom *P-value* (por exemplo, baixa probabilidade de ter sido gerado por acaso). A cada iteração, adicionam-se ou removem-se segmentos de acordo com a probabilidade do modelo corrente poder gerá-los. Se aquela probabilidade é suficientemente alta, o modelo é atualizado com os novos segmentos e o algoritmo passa para a próxima iteração.

Dentre as estratégias iterativas estocásticas ainda destaca-se HMM, que tenta simultaneamente maximizar os dados e o modelo através de uma abordagem bayesiana [26, 104].

Dois outros métodos de alinhamento iterativo foram descritos: PRALINE [108] e IterAlign [41], que compartilham protocolos muito similares. Quanto a suas potenciais performances, nenhum deles foi adequadamente avaliado. Todavia deve ser dado ênfase a novos conceitos que eles incorporaram: uso de informação local no IterAlign, que objetiva um decremento na sensibilidade quanto a parametrização de penalidade de *gap*; e o conceito de consistência. A busca por consistência tem se tornado um dos pontos mais fortes nos desenvolvimentos em MSA. Tal conceito é, também, central nos métodos não-iterativos.

A Tabela 2.7 apresenta uma breve descrição de alinhadores iterativos de destaque.

Tabela 2.7: Alinhadores iterativos de destaque.

Alinhador	Descrição	Ano
PRRP [93]	Não estocástico, é baseado em programação dinâmica. Trata-se de uma estratégia de duplo aninhamento com aleatorização que otimiza uma pontuação de soma dos pares com peso e função afim para penalização de *gaps*. Otimiza simultaneamente pesos e alinhamento.	1996
SAGA [182]	Implementa um algoritmo genético que permite otimizar facilmente qualquer OF. Possui um total de 20 operadores, que modificam indivíduos (possíveis alinhamentos) ao longo das gerações.	1996

2.3.3 Métodos baseados em consistência

O primeiro método para construção de MSAs baseado em consistência foi descrito por Kececioglu na década de 1990 [138]. Dado um conjunto de sequências, o MSA ótimo é definido como aquele que está de acordo com a maioria de todos os possíveis alinhamentos ótimos para os $n(n-1)/2$ alinhamentos de pares contidos no MSA. Há, pelo menos, três boas razões que fazem OFs baseadas em consistência muito interessantes: não dependem de uma matriz de substituição específica, mas de qualquer método, ou coleção de métodos, capaz de alinhar duas sequências por vez; o esquema baseado em consistência é dependente de posição; e experiências mostraram que dado um conjunto de observações independentes, a maioria dos consistentes estão frequentemente próximos da verdade [65]. Embora a primeira OF baseada em consistência tenha sido descrita em 1993, passaram-se ainda alguns anos para o desenvolvimento de algoritmos heurísticos capazes de tratar sua otimização. Um GA (SAGA [182]) foi usado para mostrar os avanços biológicos de tal função, denominada COFFEE [184], que emula o problema do *trace* de peso máximo. No SAGA-COFFEE, a coleção de alinhamentos de pares com peso é nomeada uma biblioteca e o SAGA é utilizado para computar o alinhamento que tem o mais alto nível de consistência com a biblioteca. Na prática, a biblioteca pode conter mais que um alinhamento para cada par de sequências. A informação que ela contém pode ser redundante, conflitante e pode se originar de fontes que variam tanto quanto se desejar (análise de estrutura, comparação de sequências, busca em bases de dados, conhecimento experimental, etc.).

O SAGA-COFFEE obteve resultados interessantes, mas havia problema quanto ao desempenho. Isso levou ao desenvolvimento de um novo alinhador heurístico, chamado T-COFFEE [183], para otimizar a função COFFEE de maneira que fosse eficiente em termos de tempo. Neste, a biblioteca COFFEE é transformada na biblioteca estendida, uma matriz de substituição para cada posição específica, onde a pontuação associada a

cada par de resíduos depende da compatibilidade daquele par com o resto da biblioteca. O alinhador faz uso de um procedimento baseado na multiplicação de matrizes de pontos de Vingron e Argos [256] e da sobreposição de pesos de Morgenstern e colegas [169]. O alinhamento múltiplo é montado usando um algoritmo de alinhamento progressivo semelhante ao Clustal W. Uma característica importante do T-COFFEE é que a biblioteca primária é criada de uma combinação de alinhamentos globais (produzidos pelo Clustal W) e alinhamentos locais (produzidos pelo Lalign). Testes realizados através do BAliBASE mostraram que a combinação de informação global e local permite ao T-COFFEE superar o PRRP, Clustal W e DiAlign nas cinco categorias de alinhamentos de referência contidas nessa ferramenta [183]. T-COFFEE é de grande interesse, não apenas pela forma que permite combinar dados heterogêneos em alinhamentos, como pode ser observado em seu uso no 3D-COFFEE [189], mas também como um precursor para o método baseado em probabilidade do ProbCons [60].

ProbCons é um método de alta precisão para construção de MSAs, conforme avaliação realizada utilizando BAliBASE [259]. Ele obteve os melhores resultados, dentre os alinhadores testados, em todos os cinco conjuntos de referência providos pela ferramenta de avaliação. De uma forma geral, ProbCons funciona de forma semelhante ao T-COFFEE, mas usa probabilidade em vez da heurística para pesos de pares de resíduos. Para tanto usa HMM de pares de sequências gerados por uma função objetivo de máxima precisão esperada [121]. Faz uso de um algoritmo de alinhamento progressivo, seguido de um procedimento iterativo de refinamento.

A Tabela 2.8 apresenta uma breve descrição de alinhadores, de destaque, baseados em consistência.

Tabela 2.8: Alinhadores, de destaque, baseados em consistência.

Alinhador	**Descrição**	**Ano**
T-COFFEE [183]	Primeira solução eficiente na otimização de uma função objetivo baseada em consistência. Utiliza a abordagem progressiva para construir o alinhamento múltiplo.	2000
ProbCons [60]	Utiliza probabilidade, em vez de heurística como no T-COFFEE, para definir pesos de pares de resíduos. Implementa consistência bayesiana e faz uso de HMMs.	2005
MUMMALS [196]	Fortemente inspirado no ProbCons, implementa múltiplos estados de *match* que descrevem as estruturas locais.	2006

2.3.4 Métodos baseados em consenso, modelos e blocos

M-COFFEE [260] implementa um meta-método consenso baseado no T-COFFEE. Inicialmente, diversos métodos para construção de MSAs são utilizados para computar alinhamentos alternativos. Então utiliza-se o T-COFFEE para combiná-los em um MSA final que seja consistente com os alinhamentos de entrada. É importante salientar, entretanto, que o M-COFFEE enfrenta problemas à medida que as sequências apresentem menor similaridade [181].

Para tentar resolver os problemas decorrentes de sequências de entrada com baixa similaridade, uma alternativa apontada é incorporar informações relativas às sequências. Seguindo essa abordagem, tem-se os métodos de alinhamento baseados em modelos [17], que podem ser implementados com extensão estrutural ou extensão por homologia. Extensão estrutural foi inicialmente descrita por Taylor [238]. Tal método inicia pela busca de um modelo estrutural, no Protein Data Bank (PDB) [37], para cada sequência, usando BLAST [12]. Então alinham-se modelos usando um método para superposição de estruturas e mapeam-se as sequências originais em seus alinhamentos de modelo. Tais resultados compõem uma biblioteca primária que é enviada a um método baseado em consistência para calcular o resultado final.

3D-COFFEE [189] foi projetado para criar alinhamentos de sequências de proteínas que incorporam informação de estruturas tridimensionais, quando essas existem. Em um conjunto de sequências de entrada, é comum encontrar entradas PDB [37] para uma ou mais dessas sequências. O objetivo de utilizar as estruturas tridimensionais é facilitar o alinhamento de sequências com relacionamentos distantes, pois elementos estruturais são geralmente mais conservados que sequências primárias e assim permitem um bom alinhamento, mesmo na *twilight zone* (menos de 25% de identidade). Na prática, usar esse tipo de informação pode ser complicado, mas há trabalhos descrevendo seu uso [5]. Explorando a habilidade do T-COFFEE em incorporar informações heterogêneas, 3D-COFFEE implementa um método rápido e simples que adiciona dados estruturais no alinhamento e, assim, alcança um elevado nível de precisão. Quando é dada apenas uma estrutura, o 3D-COFFEE combina cada sequência com a estrutura, usando uma ferramenta externa (FUGUE [220]), e converte a saída para um alinhamento de duas sequências. Dessa forma, tem-se um conjunto de alinhamentos de pares que indicam como as sequências alinham com a estrutura e, indiretamente, como cada sequência se alinha às outras. Caso duas ou mais estruturas estejam disponíveis, é possível usar um pacote de superposição de estrutura completa (SAP [241]). Apesar de usar FUGUE e SAP por padrão, o 3D-COFFEE permite o uso de outras ferramentas do gênero.

A extensão por homologia foi originalmente aplicada no pacote DbClustal [251] e trabalha de forma semelhante. A diferença está no fato de usar um perfil (construído com PSI-BLAST [14], no caso do DbClustal) em vez da estrutura. Outros pacotes que fazem

alinhamento baseado em modelos são: Expresso [17], PROMALS [197], PRALINE [108], SPEM [273] e T-Lara [32]. A maioria dos métodos baseados em modelos são baseados em consistência. PRALINE e SPEM são exceções. PROMALS, que implementa uma extensão por homologia, apresenta um forte desempenho nas ferramentas de *benchmarks* [181]. Isso sugere um bom potencial para a abordagem de extensão por homologia.

Outra abordagem para computação de MSAs é a utilização de blocos conservados (vistos como âncoras) para guiar o alinhamento que, dessa forma, não irá depender tanto dos parâmetros para penalidades de *gaps* quando as sequências compartilharem blocos conservados separados por regiões não conservadas contendo longos *indels*. Blocos são alinhamentos de fragmentos de sequências (alinhamentos locais). A maioria dos métodos que fazem uso dessa abordagem consideram apenas blocos sem *gaps*. Os blocos permitidos podem ser exatos (composto por segmentos idênticos) ou não exatos e podem ser uniformes (encontrado em todas as sequências) ou não. O primeiro programa de alinhamento múltiplo de blocos [229] usou um algoritmo de ordenação para computar blocos exatos uniformes. Algoritmos mais rápidos, baseados em árvores de sufixo [159], ou estruturas de dados equivalentes, podem também ser utilizados para computar blocos exatos. ASSEMBLE [256] realiza uma análise em matrizes de pontos para todos os pares de sequência e então compara essas matrizes para encontrar blocos uniformes, não necessariamente exatos. Na prática, é comum existirem alguns blocos que não estão presentes em todas as sequências. Melhorias têm consistido no desenvolvimento de métodos que permitam blocos não necessariamente uniformes.

No contexto dos métodos baseados em blocos, a ferramenta DiAlign [168] é a de maior destaque. Para computar um "bom" conjunto consistente de diagonais (numa matriz de pontos), faz uso de um algoritmo heurístico, que adiciona diagonais pela ordem decrescente de pontuação em um conjunto consistente de diagonais. Aquelas diagonais que não sejam consistentes com o conjunto corrente são rejeitadas.

A Tabela 2.9 apresenta uma breve descrição de métodos, de destaque, que fazem uso de alinhamentos baseados em consenso, modelos e blocos.

2.4 Avaliação de alinhadores

Existem diversas ferramentas para *benchmark* em MSA na forma de bases de dados de alinhamentos pré-compilados aos quais os alinhamentos gerados pelos métodos testados são comparados. Essas comparações são frequentemente realizadas pelo cálculo da fração de colunas de resíduos alinhados que são idênticas em ambos os alinhamentos (avaliado e referência). Esse método é conhecido como TC [134]. Outra forma comum de comparar os alinhamentos é através da pontuação SP, que computa a fração de pares de resíduos alinhados da mesma forma em ambos os alinhamentos [249]. Tal abordagem para *ben-*

Tabela 2.9: Alinhadores, de destaque, baseados em consenso, modelos e blocos.

Alinhador	Descrição	Ano
DiAlign [168]	É o alinhador baseado em blocos de maior destaque. Para computar um "bom" conjunto consistente de diagonais (numa matriz de pontos), faz uso de uma heurística, que adiciona diagonais pela ordem decrescente de pontuação em um conjunto consistente de diagonais.	1996
DbClustal [251]	Baseado em modelos, foi o primeiro a aplicar extensão por homologia. Utiliza um perfil, construído com PSI-BLAST, em vez de estruturas.	2000
3D-COFFEE [189]	Baseado em modelos, utiliza, quando disponíveis, informações de estruturas tridimensionais. Explora a habilidade do T-COFFEE em incorporar informações heterogêneas.	2004
M-COFFEE [260]	Implementa um método baseado em consenso, que combina alinhamentos em um alinhamento consistente com eles.	2006
PROMALS [197]	Baseado em modelos, é uma extensão ao método do MUMMALS. Faz buscas por sequências homólogas e utiliza predição de estruturas secundárias.	2007
PROMALS3D [198]	Baseado em modelos, explora o uso de informações estruturais para guiar alinhamentos construídos pelo PROMALS.	2008

chmark é atrativa devido a sua simplicidade, mas deve-se tomar cuidado para, na busca por melhorar o alinhador, não parametrizá-lo de forma a apresentar bons resultados nos conjuntos de referência das ferramentas para *benchmark* e degradar a performance em situações gerais. Tais ferramentas para *benchmark* têm sido efetivas na evolução dos algoritmos, levando a alinhamentos estruturalmente corretos [181], porém têm falhado pela ausência de conjuntos de referência com grande número de sequências, o que inviabiliza uma avaliação apropriada dos diversos métodos disponíveis quanto a suas capacidades diante de tal situação [181].

Dentre as soluções para *benchmark* de MSAs, BAliBASE [246] foi o primeiro construído com o propósito de *benchmarking* em larga escala. Seus conjuntos de dados são subdivididos em diversos conjuntos de referência manualmente refinados, sendo que cada um deles destina-se a avaliação de um problema específico em MSA (alinhamentos globais/locais de diferentes tamanhos e identidade de sequências, longos *gaps* internos, etc.). O refinamento manual e a subdivisão em conjuntos de referência são suas principais van-

tagens e grandes diferenciais, que o tornam uma das principais ferramentas para avaliação de alinhadores múltiplos [259].

HOMSTRAD [165] compreende um conjunto de mais de 1.000 alinhamentos, cada um deles baseado em uma família de proteínas em particular. É exclusivamente baseado em sequências com estruturas tridimensionais e arquivos PDB conhecidos. Em cada entrada, um alinhamento estrutural das proteínas é automaticamente gerado. Pode-se utilizar esses alinhamentos para *benchmark*. Possui uma menor cobertura, se comparado ao BAliBASE.

PREFAB [69] é também automaticamente gerado. Duas proteínas com estruturas conhecidas são estruturalmente alinhadas e suas sequências são usadas para consultar um banco de dados, onde resultados com altas pontuações são selecionados. Os parâmetros de consulta e seus resultados são combinados e então alinhados, usando o *software* que está sendo testado. A precisão é calculada apenas no par original, pela comparação com seu alinhamento/superposição estrutural.

O banco de dados SABmark [261] contém duas grandes bases de dados de alinhamentos de até 25 sequências de entrada cada. Há 425 grupos representando famílias de proteínas com identidade entre 25% e 50%. Há ainda 209 grupos para avaliação de *twilight zone*. Cada um desses grupos representa uma dobra de proteína definida pelo SCOP [171] e as sequências compartilham menos de 25% de identidade.

Já a ferramenta IRMbase [236] constrói seu conjunto de referência implantando *motifs* gerados pelo ROSE [234] em sequências aleatórias. Provê diversos grupos de alinhamento que variam no tamanho e no número de implantes.

Diferentemente de todos os métodos de avaliação citados, APDB [190] não recai na comparação de alinhamentos de referência pré-existentes. Em vez disso, a qualidade é mensurada com base na superposição de estruturas PDB conhecidas das sequências de entrada do alinhamento múltiplo de teste. Sua grande vantagem está na independência quanto aos alinhamentos de referência, o que evita qualquer chance de definir métodos, para construção de MSAs, otimizados para conjuntos específicos de referência. A dependência da existência das entradas PDB é a sua grande desvantagem.

Resultados de estudos sugerem que os melhores métodos têm se tornado indistinguíveis, exceto em situações de baixa similaridade (menos que 25% de identidade). Infelizmente, sequências de baixa similaridade são fracamente indicadas para a geração de alinhamentos de referência, pois suas sobreposições frequentemente aceitam diversos alinhamentos alternativos com estruturas equivalentes [144]. Visando eliminar tal dificuldade na avaliação de um método, é possível comparar diretamente o alinhamento avaliado com alguma superposição 3D idealizada, que tem sido adotada por diversos autores [17, 190, 196].

Um problema, a ser observado, refere-se à suposição de que alinhamentos estruturalmente corretos são os melhores MSAs para modelar qualquer tipo de sinal biológico (evolução, homologia ou função) [181]. Um relatório acerca de construção de perfil [99]

mostrou que alinhamentos estruturalmente corretos não resultavam necessariamente em melhores perfis. É necessário sistematicamente questionar e quantificar o relacionamento entre a precisão de MSAs e a relevância biológica de qualquer modelo oriundo deles.

Diversas análises comparativas de programas para alinhamento múltiplo têm sido publicadas [60, 68, 135, 153, 196, 197, 198, 205, 221, 249, 260]. Essas comparações são baseadas na habilidade de detectar padrões de *motifs* em diversas famílias de proteínas ou baseadas em alinhamentos de referência derivados de estruturas tridimensionais de proteínas. Análises comparativas podem também ser baseadas no efeito dos programas de alinhamento múltiplo na filogenia. Quando as sequências são similares (mais que 50% de identidade para proteínas e mais que 70% para DNAs) e essa similaridade está presente ao longo da sequência inteira, todos os métodos de alinhamento global produzem resultados bons. Além disso, em tais casos, qualquer conjunto razoável de parâmetros geram alinhamentos similares. Porém, quando pelo menos duas sequências em uma dada família compartilham uma identidade menor, ou as regiões similares são interrompidas por *gaps* longos de tamanhos diferentes, o resultado do alinhamento pode variar consideravelmente de acordo com os programas e parâmetros utilizados [65].

Métodos progressivos enfrentam grandes dificuldades com *gaps* longos devido ao esquema linear para peso de *gap*, que tende a aplicar uma penalização excessiva a *indels* longos. Métodos globais baseados em blocos, como o DiAlign [168, 169] ou IterAlign [41], são menos sensíveis a esses *gaps* longos e são particularmente apropriados para tais situações [65]. Em testes realizados, utilizando BAliBASE [249], observou-se que, em geral, o Clustal W executa melhor quando a árvore filogenética é relativamente densa. Para ele não importa o quão longa são as sequências, mas sim o quão similares elas são. Métodos globais são apropriados apenas se todos os blocos conservados são consistentes. Se alguns domínios estão duplicados ou ordenados diferentemente ao longo das sequências é necessário usar um método de alinhamento local para alinhar todos os domínios relacionados.

Um grande obstáculo na construção de MSAs é o processamento de repetições, que causam confusão em todos os métodos globais existentes para MSA. No caso de existirem repetições nas sequências de entrada, a única solução é o pré-processamento das sequências com extração desses e apenas realizar o alinhamento de regiões similares [180]. Essa extração pode ser realizada por ferramentas de alinhamento múltiplo local tal como Gibbs Sampler [147], Mocca [179] ou Repro [110]. Infelizmente nenhuma delas está bem integrada com um procedimento de alinhamento múltiplo global.

2.4.1 Avaliação dos alinhadores desenvolvidos

A principal ferramenta de avaliação utilizada ao longo deste trabalho foi BAliBASE [246]. Na Tabela 2.10 os conjuntos de referência do BAliBASE 3.0, versão que foi utilizada neste trabalho, são apresentados.

Tabela 2.10: Conjuntos de referência do BAliBASE 3.0.

ID	Nome	Descrição
RV1X	Referência 1	Sequências equidistantes. Subdivide-se em dois conjuntos com diferentes níveis de conservação: RV11, com menos de 20% de identidade entre as sequências, e RV12, com identidade entre 20 e 40%.
RV20	Referência 2	Família de proteína mais sequências "órfãs" (sequências altamente divergentes do conjunto).
RV30	Referência 3	Subgrupos de sequências altamente relacionadas, mas com menos de 25% de identidade entre sequências de grupos distintos.
RV40	Referência 4	Sequências com extensões longas nas extremidades.
RV50	Referência 5	Sequências com inserções internas longas.

Para cada teste oferecido pelo BAliBASE, há diversos arquivos. É fornecido um arquivo FASTA com as sequências de entrada, um MSF com o alinhamento de referência e um XML, que além do alinhamento de referência possui anotações associadas. Além disso há entradas com sequências completas e entradas apenas com as regiões de alta similaridade. Com exceção apenas do conjunto de referência 4, todos possuem tanto entradas de sequências completas como entradas de regiões de alta similaridade. Tal exceção deve-se ao tipo de teste projetado para essas entradas. No total são oferecidas 386 entradas de teste, sendo 218 de sequências completas e 168 de regiões de alta similaridade. Na Tabela 2.11 são apresentadas informações mais detalhadas sobre a composição desses conjuntos.

Na Figura 2.2 há um exemplo de entrada em formato FASTA. Essa é a entrada `BB11001` do BAliBASE. O prefixo `BB` indica que é composta de sequências completas. Se fosse composta apenas de regiões de alta similaridade, o prefixo seria `BBS`. O `11`, em seguida, indica que é do conjunto de referência RV11 e `001` é o identificador da entrada no conjunto. A cada entrada, está associado um número. Nessa entrada há quatro sequências, `1aab_`, `1j46_A`, `1k99_A` e `2lef_A`. Observe que, para cada sequência, há um cabeçalho de uma linha que inicia sempre com o caracter >. Nesse cabeçalho, além de um nome ou identificador para a sequência, outras informações podem estar presentes. Um alinhamento para a entrada `BB11001`, em formato MSF e gerado pelo Clustal W, é apresentado na Figura 2.3.

Tabela 2.11: Composição dos conjuntos de referência do BAliBASE. Para cada conjunto de referência a tabela apresenta uma série de valores. O parâmetro A_{num} indica o número total de alinhamentos no dado conjunto de referência e AC_{num} e AH_{num} o número de alinhamentos com sequências completas e apenas regiões de alta similaridade, respectivamente. O número médio de sequências por alinhamento é dado por S_{num}/A_{num}. Já S_{min}, S_{max}, S_{med} e S_{σ} apresentam comprimento mínimo e máximo, além de comprimento médio e desvio padrão no comprimento das sequências, dentre todas as sequências do conjunto de referência. Ainda é possível visualizar esses últimos valores restrito às entradas com sequências completas (SC_{xxx}) e apenas regiões de alta similaridade (SH_{xxx}).

Parâmetro	**RV11**	**RV12**	**RV20**	**RV30**	**RV40**	**RV50**
A_{num}	76	88	82	60	49	31
S_{num}/A_{num}	6,87	9,00	45,59	63,17	27,63	28,55
S_{min}	47	49	36	50	55	151
S_{max}	1192	2766	1520	1293	7923	1902
S_{med}	275,20	345,38	314,96	320,86	465,51	422,12
S_{σ}	40,10	53,01	50,75	56,87	281,27	99,75
AC_{num}	38	44	41	30	49	16
SC_{min}	51	49	52	63	55	172
SC_{max}	1192	2766	1520	1293	7923	1902
SC_{med}	309,47	387,44	384,65	380,66	465,51	515,06
SC_{σ}	62,86	93,25	93,30	98,25	281,27	139,02
AH_{num}	38	44	41	30	0	15
SH_{min}	47	49	36	50	-	151
SH_{max}	544	1006	602	621	-	957
SH_{med}	240,93	303,33	245,26	261,07	-	327,28
SH_{σ}	17,33	12,77	8,21	15,49	-	57,86

No BAliBASE há um utilitário, chamado `bali_score`, que calcula as pontuações de um dado alinhamento em formato MSF, comparando com o seu alinhamento de referência em formato MSF ou XML. Se é dado como entrada o alinhamento de referência em formato MSF é calculada a pontuação levando em consideração toda a extensão das sequências. Porém, se o XML é utilizado, o cálculo é feito utilizando-se apenas as regiões denominadas *core blocks*, delimitadas pelas anotações lá contidas. Tais regiões correspondem àquelas que podem ser confiavelmente alinhadas e descartam regiões de menor similaridade, onde permite-se o alinhamento de diversas maneiras. São calculadas duas pontuações, SP e TC. Tais pontuações variam entre 0 e 100. Ao calcular a pontuação do alinhamento do exemplo anterior usando o XML de referência, `bali_score` retorna 100 para as pontuações SP e TC. Ao usar o MSF de referência, `bali_score` retorna 96,10 para SP e 92,00 para TC.

```
>1aab_
GKGDPKKPRGKMSSYAFFVQTSREEHKKKHPDASVNFSEFSKKCSERWKT
MSAKEKGKFEDMAKADKARYEREMKTYIPPKGE
>1j46_A
MQDRVKRPMNAFIVWSRDQRRKMALENPRMRNSEISKQLGYQWKMLTEAE
KWPFFQEAQKLQAMHREKYPNYKYRPRRKAKMLPK
>1k99_A
MKKLKKHPDFPKKPLTPYFRFFMEKRAKYAKLHPEMSNLDLTKILSKKYK
ELPEKKKMKYIQDFQREKQEFERNLARFREDHPDLIQNAKK
>2lef_A
MHIKKPLNAFMLYMKEMRANVVAESTLKESAAINQILGRRWHALSREEQA
KYYELARKERQLHMQLYPGWSARDNYGKKKKRKREK
```

Figura 2.2: Exemplo de entrada para um alinhador múltiplo em formato FASTA.

Para avaliar um alinhador, executa-se o mesmo para cada um dos arquivos FASTA dos conjuntos de referência desejados. Depois de construídos os alinhamentos para todas as entradas, verifica-se a validade dos mesmos, ou seja, se excluídos os *gaps*, tem-se as sequências de entrada novamente. Posteriormente os resultados são pontuados pelo bali_score. Nos testes realizados, foram utilizadas apenas as entradas de sequências completas e a pontuação ficou restrita às regiões *core blocks*. Tal decisão deveu-se a uma tentativa de tornar a avaliação o mais realista possível e só avaliar regiões confiáveis do alinhamento de referência.

A Tabela 2.12 apresenta, resumidamente, os resultados de alinhadores múltiplos conhecidos quando avaliados através do BAliBASE. As tabelas 2.13 e 2.14 apresentam os resultados detalhados por conjunto de referência para as pontuações SP e TC, respectivamente. Nesses testes foram utilizadas todas as 218 entradas de sequências completas e a pontuação ficou restrita às regiões *core blocks*.

2.5 Implementações

Devido ao grande número de abordagens exploradas e a grande variedade de métodos que foram aplicados na implementação dos alinhadores múltiplos, decidiu-se por usar uma linguagem de programação que oferecesse recursos suficientes para acelerar o processo de desenvolvimento, assim como provesse um elevado nível de reutilização de código. A linguagem deveria também contar com uma API (do inglês, *Application Programming Interface*) extensa disponível publicamente.

A linguagem selecionada foi Java. É uma linguagem orientada a objetos, o que facilita a reutilização de código, possui uma API extensa e um grande número de bibliotecas

```
PileUp


   MSF:   96  Type: P    Check:  8018   ..

 Name: 1j46_A oo  Len:   96  Check:  2463  Weight:  1.000
 Name: 2lef_A oo  Len:   96  Check:  1984  Weight:  1.000
 Name: 1k99_A oo  Len:   96  Check:  8746  Weight:  1.000
 Name: 1aab_ oo  Len:   96  Check:  4825  Weight:  1.000

//



1j46_A      ------MQDR VKRPMNAFIV WSRDQRRKMA LENPRMRN-- SEISKQLGYQ
2lef_A      --------MH IKKPLNAFML YMKEMRANVV AESTLKES-- AAINQILGRR
1k99_A      MKKLKKHPDF PKKPLTPYFR FFMEKRAKYA KLHPEMSN-- LDLTKILSKK
1aab_       ---GKGDPKK PRGKMSSYAF FVQTSREEHK KKHPDASVNF SEFSKKCSER


1j46_A      WKMLTEAEKW PFFQEAQKLQ AMHREKYPNY KYRP---RRK AKMLPK
2lef_A      WHALSREEQA KYYELARKER QLHMQLYPGW SARDNYGKKK KRKREK
1k99_A      YKELPEKKKM KYIQDFQREK QEFERNLARF REDH---PDL IQNAKK
1aab_       WKTMSAKEKG KFEDMAKADK ARYEREMKTY IPPK---GE- ------
```

Figura 2.3: Exemplo de saída de um alinhador múltiplo em formato MSF.

com as mais diversas finalidades, disponíveis publicamente na Internet. Foram ainda pontos positivos para a seleção da linguagem, as IDEs (do inglês, *Integrated Development Environment*) de alta qualidade com suporte a desenvolvimento em Java distribuídas livremente e de código aberto como Eclipse [77] e NetBeans [51], além de afinidade com a linguagem devido a experiências anteriores.

Escolhida a linguagem, o próximo passo foi a busca por bibliotecas que dessem suporte a métodos que foram utilizados nas implementações. Um exemplo de biblioteca que foi utilizada nas implementações é BioJava [118], que é um *framework* Java de código aberto para processamento de dados biológicos. Um outro exemplo é a biblioteca APE (do inglês, *Analysis of Phylogenetics and Evolution*) [192], que é distribuída na forma de um pacote R e oferece uma série de funcionalidades relativas a estudos em filogenia e evolução. APE foi utilizada para a geração de árvores guias empregadas em alinhadores

Tabela 2.12: Desempenho de alinhadores múltiplos conhecidos quando avaliados através dos 218 alinhamentos de referência, compostos por sequências completas, no BAliBASE 3.0. Na primeira coluna tem-se o nome do alinhador e na segunda o tempo, em segundos, requerido para computar todos os alinhamentos. Nas duas últimas colunas tem-se as médias para as pontuações SP e TC, respectivamente.

Alinhador	**Tempo (s)**	**SP**	**TC**
ProbCons 1.12	21.978,50	86,38	55,66
T-COFFEE 8.14	14.631,89	86,11	55,46
MUMMALS 1.01	50.594,17	85,54	53,83
MUSCLE 3.7	1.636,36	82,19	47,59
DiAlign 2.2	7.286,96	77,49	41,52
Clustal W 2.0.10	2.040,10	75,36	37,38

Tabela 2.13: Desempenho, de alinhadores múltiplos conhecidos, por conjunto de referência utilizando pontuação SP.

Alinhador	**RV11**	**RV12**	**RV20**	**RV30**	**RV40**	**RV50**
ProbCons 1.12	66,97	94,11	91,67	84,60	90,24	89,17
T-COFFEE 8.14	66,32	94,10	91,94	83,79	89,76	89,32
MUMMALS 1.01	66,97	94,30	91,04	84,79	87,18	87,90
MUSCLE 3.7	57,90	91,67	89,17	80,60	87,26	83,39
DiAlign 2.2	50,73	86,66	86,91	74,05	83,17	80,69
Clustal W 2.0.10	50,06	86,44	85,16	72,50	78,93	74,24

múltiplos implementados. O R [82] é um programa, de distribuição livre, para computação estatística e produção de gráficos.

Tabela 2.14: Desempenho, de alinhadores múltiplos conhecidos, por conjunto de referência utilizando pontuação TC.

Alinhador	**RV11**	**RV12**	**RV20**	**RV30**	**RV40**	**RV50**
ProbCons 1.12	41,68	85,52	40,49	54,30	52,86	56,69
T-COFFEE 8.14	41,92	85,27	38,90	49,50	55,62	58,75
MUMMALS 1.01	41,63	83,98	42,78	49,33	48,55	52,87
MUSCLE 3.7	33,03	80,45	35,24	38,80	45,96	44,94
DiAlign 2.2	26,50	69,55	29,22	31,23	44,27	42,50
Clustal W 2.0.10	22,74	71,30	21,98	27,23	39,55	30,75

Capítulo 3

Alinhamento progressivo

Alinhamento progressivo [74] é uma das maneiras mais simples e efetivas para a realização de alinhamentos múltiplos com pequenos requisitos de tempo e memória. Essa abordagem oferece um bom desempenho quando as sequências são homólogas e relativamente bem conservadas [74, 239] e seu principal problema é a sua natureza gulosa, onde qualquer erro cometido em etapas iniciais do processo não são mais corrigidos [65].

Alinhamento progressivo consiste em construir um alinhamento múltiplo a partir de alinhamentos de pares conforme o seguinte procedimento:

1. computação da matriz de distâncias;
2. geração da árvore guia a partir das distâncias de pares; e
3. construção do alinhamento múltiplo, guiado pela árvore construída no passo anterior.

As duas primeiras etapas visam a criação de um mecanismo que determine a proximidade entre as sequências. Já a última etapa, iterativamente, utiliza esse mecanismo para determinar o próximo par a agrupar. A cada passo da iteração, da etapa 3, um par de sequências (ou alinhamentos produzidos em passos anteriores da iteração) é agrupado. Ao final de $n-1$ iterações, onde n indica o número de sequências da entrada, tem-se o alinhamento múltiplo completo. Para maiores informações sobre a abordagem progressiva, consulte a Seção 2.3.1.

Neste capítulo, um extenso trabalho realizado no contexto de alinhamento progressivo é detalhado. Os métodos, que foram empregados em cada uma das três etapas, são descritos nas seções 3.1, 3.2 e 3.3. Na Seção 3.4 são apresentados os 342 alinhadores progressivos implementados. No final, expõe-se como foram realizados os testes na Seção 3.5 e os resultados são detalhados na Seção 3.6.

3.1 Cálculo da matriz de distâncias

Nesta seção, métodos utilizados para computar matriz de distâncias são detalhados. Inicialmente são apresentados nas seções 3.1.1, 3.1.2, 3.1.3 e 3.1.4 os métodos PAM, JTT, PMB e modelo das categorias, cujas implementações são fornecidas pelo pacote PHYLIP [73]. Esses métodos restringem-se ao cálculo na matriz. Posteriormente, dois métodos desenvolvidos ao longo deste trabalho são apresentados. O primeiro, detalhado na Seção 3.1.5, utiliza alinhamentos locais para computar o alinhamento global entre um par de sequências. Já o segundo, apresentado na Seção 3.1.6, utiliza uma função logarítmica para penalizar *gaps*. Esses dois últimos métodos têm a capacidade de gerar alinhamentos de pares, assim como computar distâncias que derivam desses alinhamentos.

3.1.1 Modelo PAM

PAM [56] é um conjunto de matrizes de substituição construído empiricamente com base em um modelo de Markov de evolução de proteínas. Esse modelo indica a probabilidade de uma substituição a cada 100 resíduos nas sequências de aminoácidos, ou seja, duas sequências têm distância 1 PAM se elas diferem entre si por 1 substituição a cada 100 aminoácidos.

Para determinar a matriz de probabilidades, $P(t)$, em um tempo evolucionário $t > 0$ é utilizada uma matriz de taxas instantâneas, frequentemente denotada por $Q = (q_{ij})_{i,j=1,\ldots,20}$, que é relacionada a $P(t)$ pela Equação 3.1. O valor de q_{ij} é dado pela Equação 3.2, onde n_{ij} é o número observado de trocas entre os aminoácidos i e j, e m_i é a taxa de mutação para o aminoácido i, que é dada pela Equação 3.3.

$$P(t) = exp(tQ) = I + tQ + \frac{(tQ)^2}{2!} + \frac{(tQ)^3}{3!} + \ldots \tag{3.1}$$

$$q_{ij} = \frac{m_i n_{ij}}{\sum_{k \neq i} n_{ik}} \tag{3.2}$$

$$m_i = \frac{\sum_{j \neq i} n_{ij}}{\sum_k n_{ik}} \tag{3.3}$$

A seguir, um exemplo de matriz de distâncias gerada pelo método PAM.

	1j46_A	2lef_A	1k99_A	1aab_
1j46_A	0,000000	1,733140	2,496830	3,469881
2lef_A	1,733140	0,000000	2,666700	3,266899
1k99_A	2,496830	2,666700	0,000000	2,224336
1aab_	3,469881	3,266899	2,224336	0,000000

Nos alinhadores implementados foi utilizada a implementação do modelo PAM provida pelo pacote PHYLIP [73], que usa a versão DCMut [142] do modelo PAM de Dayhoff [56] para os cálculos. A matriz do exemplo anterior foi gerada pelo PHYLIP, recebendo `BB11001` como entrada.

3.1.2 Modelo Jones-Taylor-Thornton (JTT)

O modelo Jones-Taylor-Thornton [130] é similar ao modelo PAM, exceto que é baseado em uma recontagem do número de trocas observadas. Usa uma amostragem muito maior e a distância, assim como no modelo PAM, é mensurada em unidades da fração esperada de aminoácidos trocados. Sua amostragem mais extensa o torna mais preciso.

A matriz de mutação de dados (MDM, do inglês *Mutation Data Matrix*) é construída em três etapas:

- Agrupa as sequências em famílias homólogas;
- Calcula as mutações observadas entre sequências altamente similares;
- Relata as frequências de mutações que ocorreram ao acaso.

O modelo usa um método de aproximação para inferir relações filogenéticas entre conjuntos de sequências.

Nos alinhadores implementados foi utilizada a implementação do modelo JTT provida pelo pacote PHYLIP [73].

3.1.3 Modelo PMB

O modelo PMB (*Probability Matrix from Blocks*) [253] é baseado na abordagem original de agrupamento de Henikoff e Henikoff [106] para gerar matrizes BLOSUM de uma versão atualizada do banco de dados Blocks. A partir das matrizes BLOSUM derivam-se matrizes de mutabilidade e, usando a propriedade aditiva das distâncias evolucionárias, estima-se o relacionamento entre a frequência de substituição atual e a frequência de substituição observada média. Assim, as matrizes de mutabilidade podem ser expressas como uma função da distância evolucionária atual, definindo um modelo evolucionário que é consistente com as matrizes de pontuação BLOSUM e que é aplicável sobre um intervalo completo de divergência evolucionária.

A série de matrizes BLOSUM possuem maior precisão que a série PAM para conjuntos de sequências com maior divergência, porém BLOSUM define apenas matrizes de pontuação, não definindo um modelo probabilístico de substituição como feito pelo modelo PAM.

Nos alinhadores implementados foi utilizada a implementação do modelo PMB provida pelo pacote PHYLIP [73].

3.1.4 Modelo das categorias (PCM)

O modelo das categorias baseia-se no modelo de dois parâmetros de Kimura [140] para sequências de nucleotídeos. Os aminoácidos são agrupados em uma série de categorias. Trocas entre aminoácidos de uma mesma categoria são sempre permitidas, mas quando pertencem a categorias distintas irão depender de um parâmetro. As trocas entre resíduos de categorias distintas são permitidas quando o valor deste é 1 e vai se tornando cada vez mais difícil a medida que o valor se aproxima de 0. Tal parâmetro tem seu valor regulado pelo tempo (distância evolucionária). Foi desenvolvido por Felsenstein e implementado no pacote PHYLIP [73].

3.1.5 Distância local recursiva (LD)

Fundamenta-se em pontuar buscando os trechos mais similares. Utiliza o algoritmo de alinhamento local de Smith e Waterman [226] recursivamente para calcular a distância entre um par de sequências.

A recursão inicia realizando um alinhamento local. Em seguida, extrai-se o trecho não alinhado à esquerda e à direita do alinhamento local. Para cada um desses trechos é feita uma chamada recursiva. A distância é dada pela soma da pontuação do alinhamento local com os retornos das chamadas recursivas à direita e à esquerda. A condição de parada da recursão é quando pelo menos uma das sequências possuir comprimento 0. Nesse caso, a função retorna a penalidade proporcional ao tamanho da entrada. Sua implementação é apresentada em Algoritmo 1.

A implementação para o algoritmo de alinhamento local (`localAlignment`) utilizada foi a da biblioteca BioJava [118], que implementa Smith e Waterman [226]. Utilizou-se BLOSUM62 como matriz de substituição e penalidades 12 e 3 para abertura e extensão de *gap*.

Testes preliminares com o alinhador local resultaram em alinhamentos com muitos *gaps* e *mismatches*. Uma hipótese para tal situação é de que a falta de um limite mínimo para o comprimento do alinhamento local poderia estar particionando as regiões conservadas das sequências.

Partindo desse pressuposto, decidiu-se avaliar o desempenho do algoritmo quando há um limite mínimo para comprimento do alinhamento local. Então surgiu o problema de decidir qual o valor adequado para esse limiar. Para resolver tal situação, testes foram realizados com diferentes limiares e utilizou-se a porcentagem de *core blocks* corretamente alinhados como critério de avaliação dos resultados.

Algoritmo 1: Algoritmo recursivo (`localDistance`) para cálculo da distância local.

```
   Input: sequence1, sequence2
   Output: distance
 1 if sequence1.length > 0 and sequence2.length > 0 then
 2 |   aln ← localAlignment(sequence1, sequence2)
 3 |   lSeq1 ← sequence1[1..(aln.start1 - 1)]
 4 |   lSeq2 ← sequence2[1..(aln.start2 - 1)]
 5 |   lDistance ← localDistance(lSeq1, lSeq2)
 6 |   rSeq1 ← sequence1[(aln.end1 + 1)..sequence1.length]
 7 |   rSeq2 ← sequence2[(aln.end2 + 1)..sequence2.length]
 8 |   rDistance ← localDistance(rSeq1, rSeq2)
 9 |   return aln.score + lDistance + rDistance
10 else
11 |   if sequence1.length > 0 then
12 |   |   return 12 + 3 × sequence1.length
13 |   else
14 |   |   if sequence2.length > 0 then
15 |   |   |   return 12 + 3 × sequence2.length
16 |   |   else
17 |   |   |   return 0
```

Como entrada para os testes foram utilizados todos os pares de sequências de cada um dos dez maiores arquivos de cada conjunto de referência do BAliBASE, o que totaliza 103.759 pares. O número de sequências de cada entrada do BAliBASE utilizada é apresentado na Tabela 3.1. Por exemplo, o arquivo 5 do conjunto RV11 do BAliBASE possui 14 sequências e 91 pares. Já o arquivo 4 do conjunto RV12 possui 15 sequências e 105 pares. Observe que combinando 14 sequências duas a duas, tem-se:

$$C_2^{14} = \binom{14}{2} = \frac{14!}{2!(14-2)!} = \frac{14 \times 13}{2} = 91$$

Cada um dos pares de sequências foram alinhados com o alinhador local recursivo, utilizando diferentes valores para o limiar (0, 50, 100, 150 e 200). Dessa forma, para cada entrada utilizada do BAliBASE construiu-se 5 alinhamentos locais - um para cada limiar - e um alinhamento global.

Para cada alinhamento produzido, computou-se a porcentagem de *core blocks* corretamente alinhados - segundo o alinhamento de referência do BAliBASE. Então calculou-se a média dessas porcentagens para todos os pares de cada entrada e para cada conjunto. Calculou-se a média não só para o conjunto completo de pares de sequências. Calculou-se

Tabela 3.1: Número de sequências de cada entrada usada nos alinhamentos para escolha do limite para alinhador local. A coluna **E** indica a entrada do BAliBASE e a coluna **S** indica o número de sequências da entrada.

RV11		**RV12**		**RV20**		**RV30**		**RV40**		**RV50**	
E	**S**	**E**	**S**	**E**	**S**	**E**	**S**	**E**	**S**	**E**	**S**
5	14	4	15	3	74	1	116	2	55	1	34
7	9	8	13	14	65	3	142	4	67	3	43
18	14	11	12	25	81	5	113	11	39	6	60
19	10	15	12	31	60	13	86	16	42	7	23
20	9	26	18	32	61	18	78	17	42	8	26
28	10	27	13	34	59	21	140	37	46	9	28
30	14	29	12	36	91	23	77	41	55	11	36
31	11	35	27	37	65	26	81	44	47	12	49
33	11	37	13	39	91	28	89	47	54	14	30
38	8	43	34	40	87	29	98	49	62	15	32

também as médias de pares de sequências formados por combinação entre sequências curtas e curtas, curtas e longas e longas e longas. Uma sequência é considerada curta quando tem comprimento menor que a mediana dos comprimentos das sequências do mesmo conjunto e longa caso contrário. Na Tabela 3.2 o resultado médio para todos os conjuntos do BAliBASE é apresentado. Como pode-se observar, ao comparar os resultados, o valor 100 é o mais adequado para o limiar.

3.1.6 Função logarítmica para penalização de *gaps* (LOGD)

Quando há *gaps* longos, funções como a aditiva e a afim não pontuam adequadamente o alinhamento. Isso pode levar a resultados ruins. Uma alternativa é utilizar uma função logarítmica, que foi inicialmente abordada por Waterman e colegas em 1976 [266]. Um exemplo de tal tipo de função é apresentado na Equação 3.4, onde $w(k)$ é a penalidade dada à sequência de k *gaps*, *gop* é uma constante de penalidade para abertura de um bloco de *gaps* consecutivos e c é uma constante de penalidade associada ao comprimento do bloco de *gaps*.

$$w(k) = gop + c \times log(k) \tag{3.4}$$

O peso para abertura do *gap* rotineiramente será maior que o de c. Na Tabela 3.3 é apresentado o comportamento da função. Como exemplo, utilizou-se $gop = -8$ e $c = -5$. A medida que o *gap* cresce, a penalidade aumenta, porém o incremento vai decaindo com

Tabela 3.2: Média da porcentagem de *core blocks* bem alinhados nos alinhamentos de pares de sequências das 10 maiores entradas de cada um dos conjuntos, RV11 a RV50. A coluna mm corresponde a média do alinhamento de pares de sequências curtas. A coluna mM refere-se a pares com uma sequência curta e uma longa. Já a coluna MM corresponde a média do alinhamento de pares de sequências longas. Finalmente, $Total$ são pares de todas as sequências contra todas as outras.

Alinhamento	mm	mM	MM	$Total$
Local, limiar=0	65,22	56,88	57,65	58,60
Local, limiar=50	68,88	61,88	62,20	63,09
Local, limiar=100	71,19	65,09	65,72	66,29
Local, limiar=150	70,52	63,54	64,78	65,05
Local, limiar=200	69,54	62,87	64,42	64,23
Global	66,79	57,83	59,02	59,16

o tempo. Isso, em teoria, deve permitir que a penalidade do *gap*, mesmo que longo, seja tratada como algo viável pelo alinhador [163].

Tabela 3.3: Comportamento da função logarítmica para pontuação de *gaps*.

k	$w(k)$
1	-8.00
2	-13.00
3	-15.92
4	-18.00
5	-19.61
6	-20.92
7	-22.04
8	-23.00
9	-23.85
10	-24.61
100	-41.22
1000	-57.83

Miller e Myers [163] ressaltam que a análise de *indels* consecutivos deve ser tratada como uma operação atômica e não como a inserção ou remoção individual de resíduos. Eles ainda mostram como utilizar uma função côncava ($\Delta w_k \geq \Delta w_{k+1}$, onde $\Delta w_k = w_{k+1} - w_k$), como a logarítmica, para representar um bloco de *gaps*. Seja n o tamanho das sequências. Um algoritmo simples para alinhar pares de sequências com penalidade logarítmica para *gaps*, como o apresentado no Algoritmo 2, tem complexidade $O(n^3)$ para

tempo e $O(n^2)$ para espaço. Esse custo é muito elevado e pode tornar-se impraticável a medida que as sequências tornam-se mais longas. Miller e Myers apresentaram uma solução melhor, que tem complexidade $O(n^2 \log n)$ para tempo e $O(n)$ para espaço [163]. Foi necessário fazer uso desse último algoritmo nos alinhadores devido à performance.

Algoritmo 2: Algoritmo cúbico para alinhamento com função logarítmica para penalidade de *gaps*

```
   Input: sequence1, sequence2, gop, c
   Output: matrix
 1 length1 ← sequence1.length
 2 length2 ← sequence2.length
 3 matrix[length1 + 1,length2 + 1]
 4 matrix[0,0] ← 0
 5 for i ← 1 to length1 do
 6 |  matrix[i,0] ← gop + c × log i
 7 for j ← 1 to length2 do
 8 |  matrix[0,j] ← gop + c × log j
 9 for i ← 1 to length1 do
10 |  for j ← 1 to length2 do
11 |  |  M ← matrix[i - 1,j - 1] + scoreAlign(sequence1_i, sequence2_j)
12 |  |  for k ← 1 to i do
13 |  |  |  if M < matrix[k - 1,j] + gop + c × log(i − k) then
14 |  |  |  |  M ← matrix[k - 1,j] + gop + c × log(i − k)
15 |  |  for k ← 1 to j do
16 |  |  |  if M < matrix[i,k - 1] + gop + c × log(j − k) then
17 |  |  |  |  M ← matrix[i,k - 1] + gop + c × log(j − k)
18 |  |  matrix[i,j] ← M
19 return matrix
```

Na Figura 3.1 são apresentados três alinhamentos. O primeiro usa um algoritmo com função logarítmica para penalidade de *gaps* com 6, -1, -8 e -2 para *match*, *mismatch*, *gop* e *c*, respectivamente. O segundo usa uma função linear com $match = 4$, $mismatch = -1$ e $gap = -5$. O terceiro usa uma função afim com $match = 4$, $mismatch = -1$, $gop = -17$ e $gep = -1$.

Observe como o alinhamento linear (o segundo) tende a espalhar mais os *gaps*. Isso, analisando do ponto de vista biológico, é mais difícil de acontecer entre sequências relacionadas. Já os resultados dos alinhamentos com funções afim (o terceiro) e logarítmica (o primeiro) tiveram resultados bem distintos em relação a função linear, mas similares

```
GCGATGACGATGACGATGACGATGACGATGACGATGACGATGACGATATATTATAGACGATGAGAGGGGATTACGATGACAG
GCGATGACTGGGATGATGACGATGACTA-GA-------GAAGACGATATAT----GACGATGGGT--------------CTC

GCGATGACGATGACGATGACGATGACGATGACGATGACGATGACGATATATTATAGACGATGAGAGGGGATTACGATGACAG
GCGATGAC--TGG-GATGA---TGACGATGACTA-GA-GAAGACGATATAT----GACGATG-G-G----T--C--T--C--

GCGATGACGATGACGATGACGATGACGATGACGATGACGATGACGATATATTATAGACGATGAGAGGGGATTACGATGACAG
GCGATGACTGGGATGATGACGATGAC--------TAGAGAAGACGATATAT----GACGATG--------------GGTCTC
```

Figura 3.1: Três exemplos de alinhamentos. O primeiro usa função logarítmica para penalização de *gaps*, o segundo função linear e o terceiro função afim.

entre si. Note que para diferenciar os resultados entre os alinhadores com função afim e logarítmica seria necessário um exemplo com sequências mais longas e com tamanhos bem diferentes.

Uma vez que o alinhamento esteja computado, é possível calcular a pontuação com função logarítmica para penalidade de *gaps* utilizando o Algoritmo 3.

Utilizando como entrada o alinhamento do exemplo anterior para o Algoritmo 3 com $match = 6$, $mismatch = -1$, $gop = -8$ e $c = -2$ tem-se como resultado a seguinte pontuação.

$$\begin{aligned} score &= 6 \times 46 - 1 \times 10 - 8 \times 4 - 2 \times (\log 1 + \log 7 + \log 4 + \log 14) \\ &= 276 - 10 - 32 - 2 \times (0 + 2.8073 + 2 + 3.8073) \\ &= 234 - 2 \times (8.6146) \\ &= 216.7708 \end{aligned}$$

Uma vez computadas as pontuações dos alinhamentos entre os pares de sequências, pode-se converter essas pontuações em distâncias. A pontuação é normalizada para um valor entre 0 e 1 e, em seguida, subtrai-se esse valor de 1 para se obter a distância.

3.2 Construção da árvore guia

Nesta seção, métodos utilizados para a geração de uma árvore guia, a partir de uma matriz de distâncias, são apresentados. Tal árvore guiará a seleção dos pares de sequências (ou alinhamentos) a agrupar a cada passo da iteração onde o alinhamento é progressivamente construído. Na Seção 3.2.1 é apresentado o método UPGMA e na Seção 3.2.2 é apresentado o método *Neighbor Joining* (NJ). Foram utilizadas implementações em R [82] para ambos os métodos. UPGMA já está implementado nas bibliotecas padrões do R e NJ está implementado em uma biblioteca chamada APE [192].

Algoritmo 3: Cálculo da pontuação do alinhamento de duas sequências com penalidade logarítmica para *gaps*

```
Input: alignedSeq1, alignedSeq2, match, mismatch, gop, c
Output: score
1  score ← 0
2  gap_size ← 0
3  gap_openned ← false
4  for k ← 1 to alignedSeq1.length do
5      if alignedSeq1_k ≠ "-" and alignedSeq2_k ≠ "-" then
6          if gap_openned then
7              score ← score + c × log(gap_size)
8              gap_openned ← false
9          if alignedSeq1_k = alignedSeq2_k then
10             score ← score + match
11         else
12             score ← score + mismatch
13     else
14         if gap_openned then
15             gap_size ← gap_size + 1
16         else
17             score ← score + gop
18             gap_openned ← true
19             gap_size ← 0
20 if gap_openned then
21     score ← score + c × log(gap_size)
22 return score
```

3.2.1 *Unweighted Pair Group Method with Arithmetic Mean* (UPGMA)

Este, que também é conhecido por UPGMA, é o método mais simples dessa categoria [175]. Inicialmente descrito por Sokal e Michener em 1958 [230], foi redefinido por Sneath e Sokal em 1973 [228]. É exatamente essa segunda versão que é utilizada atualmente. Originalmente foi utilizado para a construção de fenogramas. Entretanto pode ser utilizado para a construção de filogenias moleculares desde que a taxa de substituição de genes seja aproximadamente constante. Especialmente quando dados de frequência genética são usados para reconstrução filogenética, esse modelo mostra-se superior nos resultados, quando comparado com outros métodos de distância [176, 237]. UPGMA fre-

quentemente recai em erros quando a taxa de substituição de genes não é constante ou quando o número de genes (ou nucleotídeos) usados é pequeno.

Seja a matriz esquematizada na Figura 3.2 a matriz de distâncias dada como entrada para o algoritmo. A tabela contém, inicialmente, as distâncias de todos os pares de sequências de entrada (rotuladas por 1, 2, 3, 4 e 5 nesse exemplo), sendo as distâncias representadas por $d_{i,j}$. O algoritmo começa pela inicialização da árvore (na verdade um grafo até a última etapa do algoritmo, quando finalmente é composta a árvore) definindo um número de nós equivalente ao número de sequências de entrada. Cada nó é associado a uma das sequências de entrada e na árvore resultante esses nós iniciais corresponderão às folhas. Nessa etapa não há arestas.

Sequência	**1**	**2**	**3**	**4**
2	$d_{1,2}$			
3	$d_{1,3}$	$d_{2,3}$		
4	$d_{1,4}$	$d_{2,4}$	$d_{3,4}$	
5	$d_{1,5}$	$d_{2,5}$	$d_{3,5}$	$d_{4,5}$

Figura 3.2: Representação esquemática de uma matriz de distâncias. Os números 1, 2, 3, 4 e 5 rotulam as sequências de entrada. As distâncias entre pares de sequências são representadas por $d_{i,j}$.

Então determina-se o par de sequências com menor distância, no exemplo suponha que $d_{1,2}$. O algoritmo então agrupa as sequências 1 e 2 criando um novo nó, u, ao qual são associadas duas arestas: uma para 1 e outra para 2. A ambas as arestas é associado o peso $p = d_{1,2}/2$. A tabela então é refeita desprezando 1 e 2 e adicionando u. A distância de qualquer outro nó, k, para u é então dada por $d_{u,k} = (d_{1,k} + d_{2,k})/2$. Na Figura 3.3 é apresentada a matriz de distâncias atualizada.

Sequência	**u**	**3**	**4**
3	$d_{u,3}$		
4	$d_{u,4}$	$d_{3,4}$	
5	$d_{u,5}$	$d_{3,5}$	$d_{4,5}$

Figura 3.3: Representação esquemática da matriz de distâncias depois da geração do nó u pela combinação dos nós 1 e 2.

Suponha que, agora, $d_{u,3}$ seja a menor distância. Assim, os nós u e 3 são combinados em um novo agrupamento, v. É criado o nó v, ao qual são associadas duas arestas: uma para u e outra para 3. Pesos são atribuídos às novas arestas de forma que a soma dos pesos das arestas no caminho entre cada folha descendente de v (1, 2 e 3, no caso) e v seja $d_{u,3}/2$, onde $d_{u,3} = (d_{1,3} + d_{2,3})/2$. A distância de qualquer outro nó, k, para v é

dada por $d_{v,k} = (d_{1,k} + d_{2,k} + d_{3,k})/3$. Na Figura 3.4 é apresentada a matriz de distâncias atualizada.

Sequência	**v**	**4**
4	$d_{v,4}$	
5	$d_{v,5}$	$d_{4,5}$

Figura 3.4: Representação esquemática da matriz de distâncias depois da geração do nó v pela combinação dos nós u e 3.

Agora suponha que $d_{4,5}$ seja a menor das distâncias. Dessa forma, combina-se os nós 4 e 5 em um novo agrupamento, w. Às arestas associadas a esse novo nó é atribuído o peso $p = d_{4,5}/2$. A distância de qualquer outro nó, k, para w é então dada por $d_{w,k} = (d_{4,k} + d_{5,k})/2$. Finalmente, resta agora agrupar w e v, que criará mais um nó com duas arestas associadas (a w e a v). Às novas arestas são atribuídos pesos de forma que a soma dos pesos das arestas entre cada folha e o novo nó seja $d_{v,w}/2$, onde $d_{v,w} = (d_{1,4} + d_{1,5} + d_{2,4} + d_{2,5} + d_{3,4} + d_{3,5})/6$.

A distância entre dois agrupamentos (A e B) é dada pela seguinte fórmula.

$$d_{A,B} = \frac{\sum_{i \in A, j \in B} d_{i,j}}{|A| \times |B|}$$

Na Figura 3.5 é apresentada a árvore guia gerada. Já na Figura 3.6 é apresentada a árvore guia gerada, através do método UPGMA, para a entrada `BB12021` do BAliBASE.

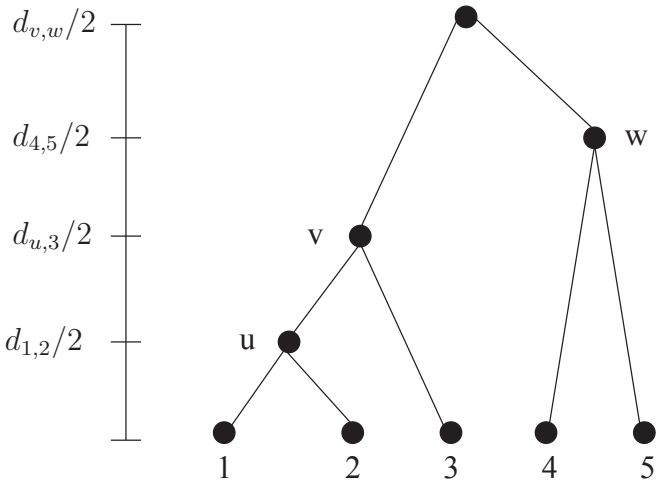


Figura 3.5: Árvore guia gerada pela execução do algoritmo UPGMA na matriz de distâncias esquematizada na Figura 3.2.

Nos alinhadores foi utilizada a implementação de UPGMA provida pelo R [82] através da função `hclust`.

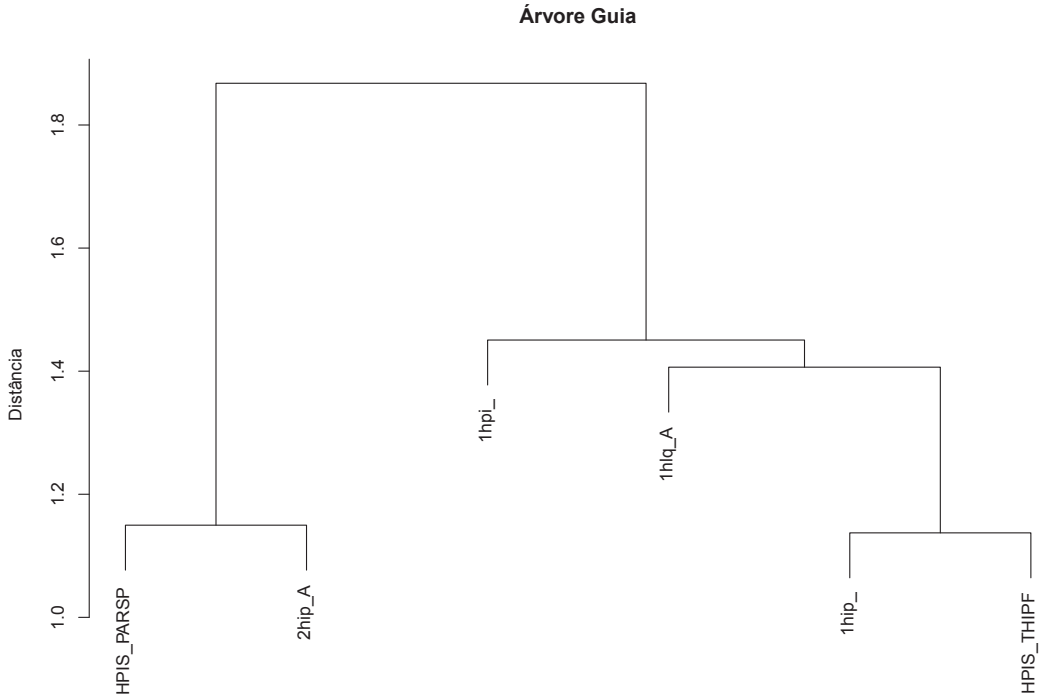


Figura 3.6: Árvore gerada por UPGMA para BB12021.

3.2.2 *Neighbor Joining* (NJ)

Desenvolvido por Saitou e Nei [210], é um método eficiente para construção de árvores que é baseado no princípio de evolução mínima [175].

Um importante conceito no método NJ é vizinhança. Dois nós são vizinhos quando são conectados por um único nó em uma árvore sem raiz. O algoritmo inicia por definir uma árvore estrela, onde é criado um nó central e todos os nós relativos às sequências de entrada são ligados a ele. Nessa árvore, cada nó relativo a uma sequência de entrada é vizinho a todo outro e, em outras palavras, significa que não há agrupamento de sequências.

Em seguinda, calcula-se a soma dos tamanhos estimados dos ramos da árvore inicial, S_0, que é dado pela Equação 3.5. Em tal equação, X refere-se ao nó central, $L_{i,X}$ denota o tamanho estimado do ramo entre o nó i e X, $T = \sum_{i<j} d_{i,j}$ e m denota o total de nós.

$$S_0 = \sum_{i=1}^{m} L_{i,X} = \frac{1}{m-1} \sum_{i<j} d_{i,j} = \frac{T}{(m-1)} \tag{3.5}$$

Passa-se então a tentar otimizar essa árvore. Isto é feito através de ciclos, onde em cada um deles é buscado o par de nós que se agrupados reduzirá o valor da soma dos tamanhos estimados dos ramos da árvore. Por exemplo, na Figura 3.7 apresenta-se uma

árvore inicial para o método NJ e na Figura 3.8 expõe-se como são agrupados os nós 1 e 2 dessa árvore pela criação de um novo nó, A.

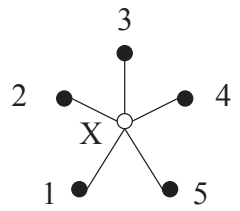


Figura 3.7: Árvore inicial do método NJ.

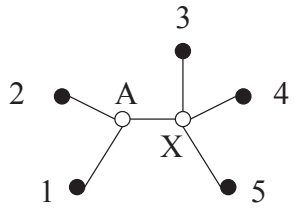


Figura 3.8: Árvore em fase de otimização no método NJ.

A soma dos tamanhos estimados dos ramos da árvore apresentada na Figura 3.8 é dada por $L_{1,A} + L_{2,A} + L_{A,X} + \sum_{i=3}^{m} L_{i,X}$, onde $L_{1,A} + L_{2,A} = d_{1,2}$,

$$L_{A,X} = \frac{1}{2(m-2)} \left[\sum_{i=3}^{m} (d_{1,i} + d_{2,i}) - (m-2)(L_{1,A} + L_{2,A}) - 2 \sum_{i=3}^{m} d_{i,X} \right]$$

e

$$\sum_{i=3}^{m} L_{i,X} = \frac{1}{m-3} \sum_{3 \leq i < j} d_{i,j}.$$

Sendo assim, tem-se:

$$\begin{aligned} S_{1,2} &= L_{1,A} + L_{2,A} + L_{A,X} + \textstyle\sum_{i=3}^{m} L_{i,X} \\ &= d_{1,2} + \tfrac{1}{2(m-2)} \textstyle\sum_{i=3}^{m} (d_{1,i} + d_{2,i}) - \tfrac{1}{2} d_{1,2} - \tfrac{1}{(m-2)} \textstyle\sum_{i=3}^{m} d_{i,X} + \tfrac{1}{(m-3)} \textstyle\sum_{3 \leq i < j} d_{i,j} \\ &= \tfrac{1}{2} d_{1,2} + \tfrac{1}{2(m-2)} \textstyle\sum_{i=3}^{m} (d_{1,i} + d_{2,i}) - \tfrac{1}{(m-2)} \tfrac{1}{(m-3)} \textstyle\sum_{3 \leq i < j} d_{i,j} + \tfrac{1}{(m-3)} \textstyle\sum_{3 \leq i < j} d_{i,j} \\ &= \tfrac{1}{2} d_{1,2} + \tfrac{1}{2(m-2)} \textstyle\sum_{i=3}^{m} (d_{1,i} + d_{2,i}) + \tfrac{1}{(m-2)} \textstyle\sum_{3 \leq i < j} d_{i,j} \end{aligned} \tag{3.6}$$

Note que se pode definir $S_{x,y}$ trocando-se 1 e 2 por x e y na Equação 3.6. A cada ciclo são calculados os valores $S_{x,y}$ para $1 \leq x < y \leq m$, onde m denota o total de nós correntes. É selecionado o menor $S_{x,y}$ e a árvore é modificada pelo agrupamento dos nós

x e y através da criação de um novo nó, A. Então são calculados os tamanhos dos ramos criados de acordo com as Equações 3.7 e 3.8, onde $R_k = \sum_{u=1}^{m} d_{k,u}$.

$$b_{A,x} = \frac{1}{2(m-2)} \left[(m-2)d_{x,y} + R_x - R_y\right] \tag{3.7}$$

$$b_{A,y} = \frac{1}{2(m-2)} \left[(m-2)d_{x,y} - R_x + R_y\right] \tag{3.8}$$

O próximo passo é computar a distância entre o novo nó, A, e o restante dos nós. Seja $1 \leq k \leq m$ com $k \neq x$ e $k \neq y$, a distância é dada por:

$$d_{A,k} = \frac{d_{x,k} + d_{y,k} - d_{x,y}}{2}$$

Computando essas distâncias, é criada uma nova matriz de distâncias com dimensão $(m-1) \times (m-1)$, onde x e y são descartados e A é adicionado. Então é criada uma nova matriz com as somas dos tamanhos estimados dos ramos e o ciclo se repete. O procedimento é repetido enquanto $m \geq 3$.

Nos alinhadores foi utilizada a implementação de NJ provida pelo R [82] através da função `nj` da biblioteca APE [192].

3.3 Geração do alinhamento múltiplo

Uma vez construída a árvore, inicia-se a construção do alinhamento múltiplo, propriamente dito, guiado pela árvore. Esta etapa é composta por um laço, onde a cada iteração é selecionado um par de sequências (ou alinhamentos) e, em seguida, é realizado o agrupamento desse par. Na Seção 3.3.1 são apresentados métodos para indicação do par a alinhar no passo corrente. Já os métodos para agrupamento do par de alinhamentos são apresentados na Seção 3.3.2. Na Seção 3.3.3 é apresentado um mecanismo de pesos, que pode ser empregado para melhorar a qualidade de alinhamentos compostos por um grande número de sequências altamente relacionadas e uma (ou poucas) sequências com um nível de similaridade inferior.

3.3.1 Seleção de par

Nos alinhadores implementados foram utilizados dois métodos de seleção, bloco único e par mais próximo, que são descritos a seguir. Estes são comumente empregados pelos alinhadores progressivos e foram implementados conforme descritos na literatura.

Seleção por bloco único (BU)

Não permite a criação de diversos alinhamentos durante o processo de alinhamento progressivo. Escolhe a sequência central, ou seja, aquela que possui a menor soma das distâncias em relação às outras sequências. Em seguida é selecionada a sequência de menor distância para ela. No passo seguinte é selecionada a sequência com menor distância para as sequências do bloco corrente. Para determinar essa distância, para cada sequência que não pertença ao bloco, soma-se sua distância para cada sequência do bloco. Nos passos seguintes é realizado o mesmo procedimento. O alinhamento progressivo é encerrado quando todas as sequência estejam no bloco [249].

Seleção do par mais próximo (NP)

Recebe como entrada a árvore construída no passo anterior e a cada passo seleciona o par de folhas vizinhas mais próximas. Por vizinhos entenda-se dois nós conectados por um outro nó. Ao final da iteração as folhas selecionadas são descartadas da árvore, transformando o nó pai em folha, e a esse nó pai é associado o alinhamento gerado pelo agrupamento das sequências (ou alinhamentos) associadas aos nós descartados. O alinhamento progressivo é encerrado quando não houver dois ou mais nós válidos na árvore [249]. É possível calcular as distâncias de pares, entre as folhas, em tempo quadrático, utilizando buscas em profundidade ou largura.

3.3.2 Agrupamento de alinhamentos

Para agrupar duas sequências é utilizado o algoritmo de Needleman e Wunsch [174]. Porém, quando se tem dois alinhamentos ou uma sequência e um alinhamento para agrupar, outros métodos tornam-se necessários. A seguir, são descritos métodos para agrupamento de alinhamentos utilizados nos alinhadores desenvolvidos. Eles foram implementados com base em dois métodos comumente empregados para tal tarefa, alinhamento de consensos e alinhamento de perfis.

Alinhamento de consensos (AC)

É o método mais simples de agrupamento. Consiste em extrair a sequência consenso de cada alinhamento e alinhá-las. Em seguida, sempre que dois resíduos dos consensos estiverem alinhados põe-se as colunas correspondentes dos alinhamentos de entrada numa mesma coluna do alinhamento de saída. E sempre que houver *gap* no alinhamento pareia-se a coluna correspondente ao resíduo com uma coluna de *gaps* correspondente ao outro alinhamento de entrada [65]. Para o alinhamento das sequências consenso, um al-

goritmo global ótimo parametrizado com BLOSUM62 e penalidades 12 e 3 para abertura e extensão de *gap*, respectivamente, foi utilizado.

Para extrair a sequência consenso de um alinhamento é adotado o seguinte procedimento. A cada coluna do alinhamento testa-se o resíduo que maximiza uma pontuação que é calculada pela combinação do resíduo testado com cada resíduo presente na coluna. A sequência consenso possuirá comprimento igual ao número de colunas do alinhamento e a cada coluna será necessário calcular a pontuação para cada possível resíduo.

A seguir é apresentada uma entrada (BB12021 do BAliBASE) para o alinhador 049, conforme especificado na Tabela B.1 do Apêndice B. Ele faz uso de alinhamento de consensos. Para tal entrada o alinhador agrupa inicialmente 1hip com HPIS_THIPF. Em seguida adiciona 1hlq_A a esse grupo e depois 1hpi. No próximo passo HPIS_PARSP e 2hip_A são agrupados e finalmente funde-se os dois alinhamentos em um só.

```
>1hpi
MERLSEDDPAAQALEYRHDASSVQHPAYEEGQTCLNCLLYTDASAQDWGPCSVFPGKLVSANGWCTAWVAR
>1hip
SAPANAVAADNATAIALKYNQDATKSERVAAARPGLPPEEQHCADCQFMQADAAGATDEWKGCQLFPGKLINVNGWCASWTLKAG
>2hip_A
EPRAEDGHAHDYVNEAADASGHPRYQEGQLCENCAFWGEAVQDGWGRCTHPDFDEVLVKAEGWCSVYAPAS
>1hlq_A
AAPLVAETDANAKSLGYVADTTKADKTKYPKHTKDQSCSTCALYQGKTAPQGACPLFAGKEVVAKGWCSAWAKKA
>HPIS_THIPF
EDLPHVDAATNPIAQSLHYIEDANASERNPVTKTELPGSEQFCHNCSFIQADSGAWRPCTLYPGYTVSEDGWCLSWAHKTA
>HPIS_PARSP
QDLPPLDPSAEQAQALNYVKDTAEAADHPAHQEGEQCDNCMFFQADSQGCQLFPQNSVEPAGWCQSWTAQN
```

Na última iteração do alinhador progressivo os alinhamentos a agrupar são:

```
SAPANAVAADNATAIALKYNQDATKSERVAAARPGLPPEEQHCADCQFMQADAAGATDEWKGCQLFPGKLINVNGWCASWTLKAG
EDLPHVDAATNPIAQSLHYIEDANASERNPVTKTELPGSEQFCHNCSFIQADSGA----WRPCTLYPGYTVSEDGWCLSWAHKTA
AAPLVAETDAN--AKSLGYVADTTKADK---TKYPKHTKDQSCSTCALYQ----GKTAPQGACPLFAGKEVVAKGWCSAWA-KKA
MERLSED---DPAAQALEYRHDAS------SVQHPAYEEGQTCLNC-LLYTDASAQ--DWGPCSVFPGKLVSANGWCTAWVAR--
```

e

```
QDLPPLDPSAEQAQALNYVKDTAEAADHPAHQEGEQCDNCMFF-QADSQGCQL-----FPQNSVEPAGWCQSWTAQN
------EPRAEDGHAHDYVNEAADASGHPRYQEGQLCENCAFWGEAVQDGWGRCTHPDFDEVLVKAEGWCSVYAPAS
```

Os consensos são computados e, então, alinhados da seguinte forma:

```
SAPLNADAATNPTAQALHYIQDATKSERNPATKHPLPPEEQHCANCSFLQADAGGQTDDWGPC--PLFPGKLVSANGWCTAWAHKTA
QDLPPLDPRAE-DGHAHNYVNDTA-----DAADHPRHQEGQQCDNCMFW---GQADQDGWGRCTHPDFPQNLVEPEGWCQSWTPQN-
```

Para determinar a primeira letra do consenso do primeiro alinhamento, calculou-se as pontuações para cada um dos possíveis resíduos. Por exemplo a pontuação para o resíduo A, seria:

$$\begin{aligned} score &= score(A,S) + score(A,E) + score(A,A) + score(A,M) \\ &= 1 - 1 + 4 - 1 \\ &= 3 \end{aligned}$$

O resíduo que obteve a maior pontuação para a primeira coluna foi S.

$$\begin{aligned} score &= score(S,S) + score(S,E) + score(S,A) + score(S,M) \\ &= 4 + 0 + 1 - 1 \\ &= 4 \end{aligned}$$

O alinhamento resultante foi o seguinte:

```
SAPANAVAADNATAIALKYNQDATKSERVAAARPGLPPEEQHCADCQFMQADAAGATDEWKGC--QLFPGKLINVNGWCASWTLKAG
EDLPHVDAATNPIAQSLHYIEDANASERNPVTKTELPGSEQFCHNCSFIQADSGA----WRPC--TLYPGYTVSEDGWCLSWAHKTA
AAPLVAETDAN--AKSLGYVADTTKADK---TKYPKHTKDQSCSTCALYQ----GKTAPQGAC--PLFAGKEVVAKGWCSAWA-KKA
MERLSED---DPAAQALEYRHDAS------SVQHPAYEEGQTCLNC-LLYTDASAQ--DWGPC--SVFPGKLVSANGWCTAWVAR--
QDLPPLDPSAE-QAQALNYVKDTA-----EAADHPAHQEGEQCDNCMFF----QADSQGCQL-----FPQNSVEPAGWCQSWTAQN-
------EPRAE-DGHAHDYVNEAA-----DASGHPRYQEGQLCENCAFW---GEAVQDGWGRCTHPDFDEVLVKAEGWCSVYAPAS-
```

Variações para alinhamento de consensos (ACb, LC, ACLog e ACLogb)

Foram implementadas quatros variações para o alinhamento de consensos.

A primeira troca o algoritmo global por um semi-global. Nesse tipo de alinhamento, os *gaps* nas extremidades não são penalizados. Dessa forma, o algoritmo pode apresentar resultados melhores em algumas situações onde as sequências apresentam comprimentos discrepantes. Esse foi parametrizado da mesma forma que a versão global.

O segundo algoritmo, em vez de utilizar um algoritmo global, usa um local recursivo. O alinhamento das sequências consenso é a junção dos diversos resultados locais. O foco desse algoritmo é a busca pelas regiões mais conservadas das sequências.

Na terceira e quarta variação para o alinhamento de consensos, foi utilizada a pontuação logarítmica para penalização de *gaps*. Em uma delas utilizou-se um algoritmo global de alinhamento e na outra um algoritmo semi-global.

Alinhamento de perfis (AP)

É uma adaptação do algoritmo de Needleman e Wunsch [174]. A entrada passa a ser dois alinhamentos múltiplos e o restante do algoritmo é adaptado para considerar esse novo tipo de entrada. Ao invés de alinhar dois resíduos alinham-se duas colunas e calcula-se a pontuação de um par de colunas pela soma das pontuações de todos os pares formados por um resíduo da coluna corrente do primeiro alinhamento de entrada com um resíduo da coluna corrente do segundo. Alinhamento de pares de *gaps* recebem 0 como pontuação. A pontuação do alinhamento de uma coluna de um dos alinhamentos de entrada com um *gap*

é calculada da mesma forma, só que substituindo toda a coluna do segundo alinhamento por *gaps* [65]. Como parâmetros utilizou-se BLOSUM62 e penalidade para *gap* 8.

No exemplo anterior, usado para explicar alinhamento de consensos, se fosse aplicado alinhamento de perfis para realizar o último agrupamento, ter-se-ia o seguinte alinhamento resultante:

```
SAPANAVAADNATAIALKYNQDATKSERVAAARPGLPPEEQHCADCQFM-QADAAGATDEWKGCQ-L-FPGKLINVNGWCASWTLKAG
EDLPHVDAATNPIAQSLHYIEDANASERNPVTKTELPGSEQFCHNCSFI-QADSGA----WRPCT-L-YPGYTVSEDGWCLSWAHKTA
AAPLVAETDAN--AKSLGYVADTTKADK---TKYPKHTKDQSCSTCALY-Q----GKTAPQGACP-L-FAGKEVVAKGWCSAWA-KKA
MERLSED---DPAAQALEYRHDAS------SVQHPAYEEGQTCLNC-LL-YTDASAQ--DWGPCS-V-FPGKLVSANGWCTAWVAR--
QDLPPLDPSAEQ-AQALNYVKDTA--E---AADHPAHQEGEQCDNCMFF-QADSQG----CQL-----FPQNSVEPAGWCQSWTAQN-
------EPRAED-GHAHDYVNEAA--D---ASGHPRYQEGQLCENCAFWGEAVQDG----WGRCTHPDFDEVLVKAEGWCSVYAPAS-
```

Nesse caso, para definir, por exemplo, o valor da posição $(10, 5)$ da matriz de programação dinâmica precisar-se-ia, além dos valores das células vizinhas, dos seguintes cálculos:

- Alinhamento da décima coluna do primeiro alinhamento (D, T, -, D) com a quinta coluna do segundo (-, P)

$$\begin{aligned} score &= 2 * score(D, P) + score(T, P) + score(-, -) - 4 * gap \\ &= 2 * (-1) - 1 + 0 - 32 \\ &= -35 \end{aligned}$$

- Alinhamento da décima coluna do primeiro alinhamento (D, T, -, D) com uma coluna de *gaps* (-, -)

$$\begin{aligned} score &= 2 * score(-, -) - 6 * gap \\ &= -48 \end{aligned}$$

- Alinhamento da quinta coluna do segundo alinhamento (-, P) com uma coluna de *gaps* (-, -, -, -)

$$\begin{aligned} score &= 4 * score(-, -) - 4 * gap \\ &= -32 \end{aligned}$$

A posição da matriz é, então, preenchida com -32, uma vez que esse é o maior valor entre os três computados.

Variações para alinhamento de perfis (APb, APA, APAb, APLog e APLogb)

Para o alinhamento de perfis foram feitas algumas variações também.

A primeira delas é o uso de função afim para penalização de *gaps*. Utilizou-se a matriz BLOSUM62 e penalidades 12 e 3 para abertura e extensão de *gap*.

Para os dois métodos de alinhamento de perfis já descritos, implementou-se versões substituindo-se o alinhamento global por um semi-global. O algoritmo semi-global com função linear de penalização para *gaps* usa matriz BLOSUM62 e penalidade 5. Já a versão com função afim usa BLOSUM62 e penalidades 12 e 3.

Outra variação realizada ao par de métodos originais (global e semi-global) foi a implementação com a função logarítmica para penalização de *gaps*.

Ajuste automático de parâmetros (APAp)

Tendo em vista a alta sensibilidade do alinhador de perfis em relação aos parâmetros de entrada, realizou-se um estudo e a implementação de um mecanismo para ajuste automático dos parâmetros para o alinhador de perfis com pontuação afim para penalidade de *gaps*. O objetivo é que o alinhador ajuste-se às sequências recebidas como entrada. Tomando como base a similaridade entre as sequências de entrada, determina-se a matriz de substituição a utilizar e, então, os valores para *gop* e *gep*, previamente determinados, são incrementados ou decrementados.

O Algoritmo 4 mostra como são computados os parâmetros a partir das sequências de entrada. Observe que a escolha da matriz de substituição é feita com base na similaridade das sequências. Quanto mais similares são as sequências, maior é o índice da matriz BLOSUM utilizada. Os valores finais para *gop* e *gep* são determinados com base em valores iniciais e em incrementos ou decrementos determinados pela média dos elementos não diagonais da matriz de substituição, selecionada em uma etapa anterior, assim como pelo tamanho das sequências de entrada. A função `avgNonDiagonal` retorna a média dos valores de uma matriz de substituição, descartando-se a diagonal. A função `getMax` retorna o comprimento da maior sequência de um alinhamento.

Seja -17 o *gop* inicial, -1 o *gep* inicial e as seguintes sequências de entrada:

```
GKGDPKKPRGKMSSYAFFVQTSREEHKKKHPDASVNFSEFSKKCSERWKTMSAKEKGKFEDMAKADKARYEREMKTYIPPKGE
MKKLKKHPDFPKKPLTPYFRFFMEKRAKYAKLHPEMSNLDLTKILSKKYKELPEKKKMKYIQDFQREKQEFERNLARFREDHPDLIQNAKK
MHIKKPLNAFMLYMKEMRANVVAESTLKESAAINQILGRRWHALSREEQAKYYELARKERQLHMQLYPGWSARDNYGKKKKRKREK
MQDRVKRPMNAFIVWSRDQRRKMALENPRMRNSEISKQLGYQWKMLTEAEKWPFFQEAQKLQAMHREKYPNYKYRPRRKAKMLPK
```

Ao aplicar o Algoritmo 4 para as duas primeiras sequências, a matriz de substituição escolhida foi a BLOSUM45. O valor do *gop* passou a ser $-7,63$ e o *gep* se manteve em -1. O alinhamento obtido utilizando o alinhamento de perfis com pontuação afim para penalidade para *gaps* foi o seguinte:

```
GK-----GD-PKKPRGKMSSYAFFVQTSREEHKKKHPDASVNFSEFSKKCSERWKTMSAKEKGKF-EDMAKADKARYEREMKTY-------IPPKGE
MKKLKKHPDFPKKP---LTPYFRFFMEKRAKYAKLHPEMS-NL-DLTKILSKKYKELPEKKKMKYIQDFQR-EKQEFERNLARFREDHPDLIQNAKK
```

Algoritmo 4: Computação dos parâmetros para alinhador de perfis com função afim para penalidade de *gaps*

Input: alignment1,alignment2,sequences,similarityMatrix,initialGop,initialGep
Output: gop, gep, substitutionMatrix

```
1  count ← 0
2  identity ← 0
3  numberOfSequences ← sequences.length
4  for i ← 1 to numberOfSequences do
5      for j ← 1 to numberOfSequences do
6          identity ← identity + similarityMatrix[i,j]
7          count ← count +1
8  identity ← identity/count
9  if identity > 0.8 then
10     substitutionMatrix ← BLOSUM80
11 else
12     if identity > 0.6 then
13         substitutionMatrix ← BLOSUM62
14     else
15         if identity > 0.3 then
16             substitutionMatrix ← BLOSUM45
17         else
18             substitutionMatrix ← BLOSUM30
19 scale ← identity
20 m ← getMax(alignment1)
21 n ← getMax(alignment2)
22 matAvgScore ← avgNonDiagonal(substitutionMatrix)
23 logmin ← log(min(m, n))
24 if matAvgScore ≤ 0 then
25     gop ← initialGop + logmin
26 else
27     gop ← scale × matAvgScore × (initialGop + logmin)
28 logNM ← |log(m/n)|
29 gep ← initialGep × (1 + logNM)
30 return (gop, gep, substitutionMatrix)
```

Para as duas últimas sequências, o algoritmo retornou BLOSUM45 com $gop = -7,65$ e $gep = -1,0$ e foi obtido o seguinte alinhamento:

```
MH--IKKPLNAFMLYMKEMRANVVAESTLKESAAINQILGRRWHALSREEQAKYYELARKERQLHMQLYPGWSARDNYGKKKKRKREK
MQDRVKRPMNAFIVWSRDQRRKMALENPRMRNSEISKQLGYQWKMLTEAEKWPFFQEAQKLQAMHREKYPNYKYRP---RRKAKMLPK
```

Por fim, para agrupar os dois alinhamentos, o algoritmo retornou BLOSUM62 com $gop = -11,56$ e $gep = -1,0$. O alinhamento final é apresentado a seguir.

```
GK-----GD-PKKPRGKMSSYAFFVQTSREEHKKKHPDASVNFSEFSKKCSERWKTMSAKEKGKF-EDMAKADKARYEREMKTY----------IPPKGE
MKKLKKHPDFPKKP---LTPYFRFFMEKRAKYAKLHPEMS-NL-DLTKILSKKYKELPEKKKMKYIQDFQR-EKQEFERNLARFREDH---PDLIQNAKK
------MH--IKKP---LNAFMLYMKEMRANVVAESTLKE--SAAINQILGRRWHALSREEQAKYYELARK-ERQLHMQLYPGWSARDNYGKKKKRKREK
------MQDRVKRP---MNAFIVWSRDQRRKMALENPRMR--NSEISKQLGYQWKMLTEAEKWPFFQEAQK-LQAMHREKYPNYKYRP---RRKAKMLPK
```

O Algoritmo 4 deu origem a um novo método de agrupamento, que basicamente combina este com o alinhamento de perfis global com função afim para penalidade de *gaps*.

Em resumo, para agrupamento, foram implementados e utilizados os seguintes métodos.

- **Alinhamento de consensos**
  - Global (AC) e semi-global (ACb)
  - Local recursivo (LC)
  - Global e semi-global com função logarítmica para penalização de *gaps* (ACLog e ACLogb)
- **Alinhamento de perfis**
  - Global (AP) e semi-global (APb)
  - Global e semi-global com função afim para penalização de *gaps* (APA e APAb)
  - Global e semi-global com função logarítmica para penalização de *gaps* (APLog e APLogb)
  - Com ajuste automático de parâmetros (APAp)

3.3.3 Esquema de pesos (PM)

No alinhamento de perfis é possível ainda adicionar um esquema de pesos com o objetivo de melhorar a qualidade dos alinhamentos quando a entrada possui sequências não equidistantes. As sequências que mais divergem recebem pesos maiores e, assim, a informação diferenciada que possuem terá um peso maior durante a construção do alinhamento múltiplo. No caso de não utilizar um esquema de pesos, as sequências que divergem do grupo tendem a ter seu alinhamento menosprezado [245].

Quando habilitado o uso de tal recurso no alinhador, atribui-se como peso de cada sequência a média das distâncias dela para todas as outras. O peso das sequências é dado pela sua distância normalizada às demais sequências do conjunto (no intervalo de 0 a 1). Sua aplicação é feita sempre que se pontua um *gap* ou o alinhamento de dois resíduos. Nesses casos a pontuação é multiplicada pelo produto dos pesos das duas sequências envolvidas.

3.4 Os alinhadores

Alinhadores, combinando os diversos métodos descritos nas seções 3.1, 3.2 e 3.3, foram implementados. Na Tabela 3.4 é apresentada a lista dos métodos que deram origem a 342 alinhadores progressivos. A lista completa dos alinhadores, com identificadores e detalhamento dos métodos que utiliza, está disponível na Tabela B.1 do Apêndice B.

Tabela 3.4: Métodos empregados na implementação dos alinhadores progressivos. Observe que foram agrupados de acordo com sua categoria (função). Consulte o glossário no Apêndice A para esclarecimentos a respeito dos métodos.

Distância	**Árvore**	**Seleção**	**Agrupamento**	**Pesos**
PAM	UPGMA	BU	AC	PM
PMB	NJ	NP	ACb	PP
PCM			LC	
JTT			ACLog	
LD			ACLogb	
LOGD			AP	
			APb	
			APA	
			APAb	
			APLog	
			APLogb	
			APAp	
			Total de alinhadores:	342

Foram realizadas todas as combinações possíveis entre os métodos. Observe que seleção por bloco único (BU), por exemplo, não utiliza árvore guia e sendo assim não é preciso fazer a sua combinação com UPGMA e *Neighbor Joining* (NJ). Note também que o esquema de pesos (PM) só é empregado quando se utiliza alinhamento de perfis. Sendo assim, ao utilizar seleção do par mais próximo (NP) sem fazer uso do esquema de pesos, tem-se $6 \times 2 \times 12 = 144$ alinhadores. São seis métodos para computação da matriz de

distâncias, dois métodos para geração da árvore guia e doze para realizar agrupamento de alinhamentos. Ao utilizar o esquema de pesos (PM), tem-se mais $6 \times 2 \times 7 = 84$ alinhadores. Utilizando seleção por bloco único sem esquema de pesos, tem-se mais $6 \times 12 = 72$ alinhadores. Já ao utilizar o esquema de pesos são mais $6 \times 7 = 42$ alinhadores.

3.4.1 Definição de parâmetros para alinhamento de perfis

Um estudo para avaliar os parâmetros para alinhadores de perfis foi realizado. Na ausência de um consenso sobre o assunto, decidiu-se realizar um estudo empírico a fim de determinar os melhores parâmetros a serem utilizados por cada alinhador de perfis. Arbitrariamente foram selecionados os alinhadores 51, 53, 51b e 53b. O primeiro utiliza alinhamento global de perfis (AP) e o segundo alinhamento global de perfis com função afim para penalidade de *gaps* (APA). Os dois últimos implementam as versões semi-globais dos dois primeiros, APb e APAb. Todos eles usam JTT, UPGMA e seleção do par mais próximo. Para cada método de agrupamento, foram realizados testes com um subconjunto das entradas providas pelo BAliBASE. Tal subconjunto foi composto por 5 entradas de sequências completas de cada um dos 6 conjuntos de referência do BAliBASE, totalizando assim 30 entradas para cada método de agrupamento (alinhador). Como critério para seleção das entradas, foi escolhido o número de sequências e depois os comprimento delas. Tal critério foi utilizado devido às limitações quanto a tempo de processamento. Selecionou-se entradas com menor número de sequências e para desempate utilizou-se o comprimento delas.

Para cada método de agrupamento com função aditiva para penalidade de *gaps*, AP e APb, variou-se a penalidade de -1 a -20 e matrizes BLOSUM 45, 50, 55, 60, 62, 65, 70, 75 e 80. Na Figura 3.9 um gráfico com as médias obtidas para AP (alinhador 51) é apresentado. Já na Figura 3.10 os resultados por conjunto do BAliBASE são expostos. A pontuação utilizada, para avaliar a qualidade dos alinhamentos gerados, foi a SP restrita a *core blocks*. As figuras 3.11 e 3.12 apresentam os resultados equivalentes para APb (alinhador 51b).

Para os alinhadores com função afim para penalidade de *gaps* não foram construídos gráficos por possuírem uma quarta dimensão (segundo parâmetro para penalidade de *gaps*). Isso dificultaria a visualização. Para estes testes, variou-se o gop (abertura de *gap*) de -1 a -20 e gep (extensão de *gap*) de -1 a -10. Com os experimentos concluídos, os seguintes parâmetros foram definidos. Alinhamento de perfis deve usar BLOSUM 62 e penalidade de *gap* -5. Alinhamento de perfis semi-global funciona bem com BLOSUM 45 e penalidade para *gap* -2. Alinhamento de perfis com função afim para penalidade de *gaps* deve usar BLOSUM 55, gop -17 e gep -1. Finalmente, a versão semi-global funciona bem com BLOSUM 45, gop -10 e gep -1.

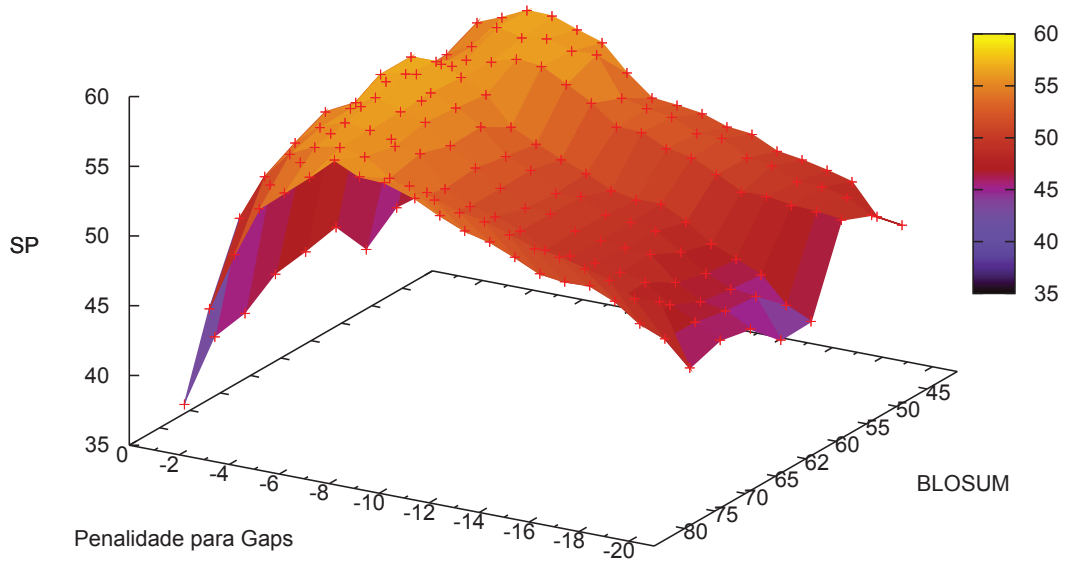
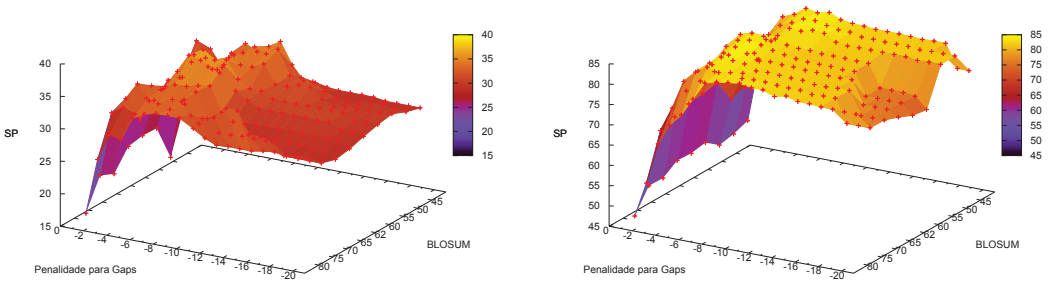
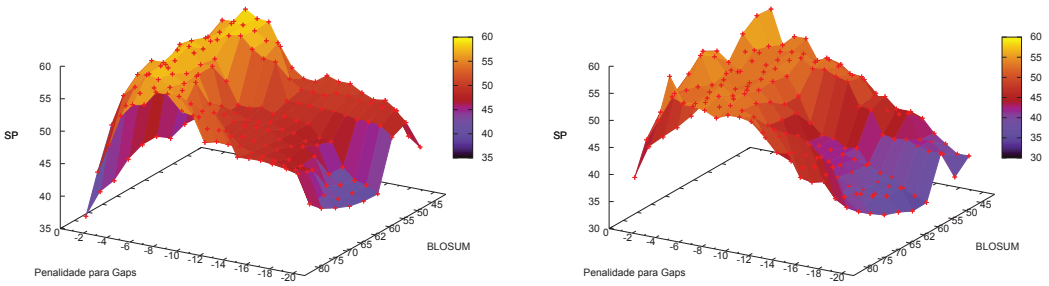


Figura 3.9: Variação na pontuação do alinhador global de perfis (AP) quando variou-se a penalização para *gap* de -1 a -20 e BLOSUM de 45 a 80. Variou-se BLOSUM com incrementos de 5 em 5 e foi também avaliada BLOSUM 62, que é uma matriz comumente utilizada.



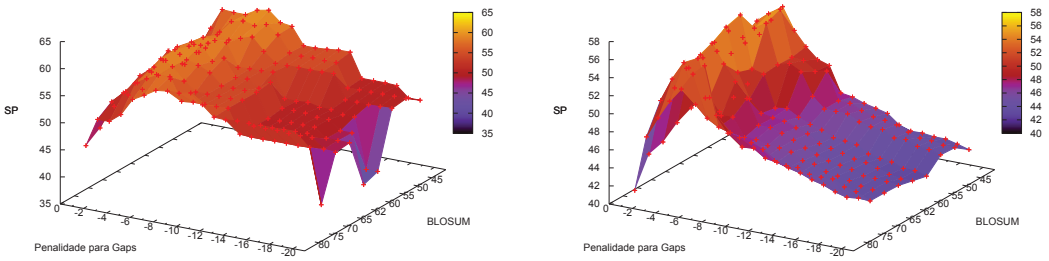
(a) RV11

(b) RV12



(c) RV20

(d) RV30



(e) RV40

(f) RV50

Figura 3.10: Variação na pontuação do alinhador de perfis com alinhamento global (AP). A penalização para *gap* variou de -1 a -20 e BLOSUM de 45 a 80. Variou-se BLOSUM com incrementos de 5 em 5 e foi também avaliada BLOSUM 62, que é uma matriz comumente utilizada. Em a) tem-se o gráfico para o conjunto RV11, em b) para RV12, em c) para RV20, em d) para RV30, em e) para RV40 e em f) para RV50.

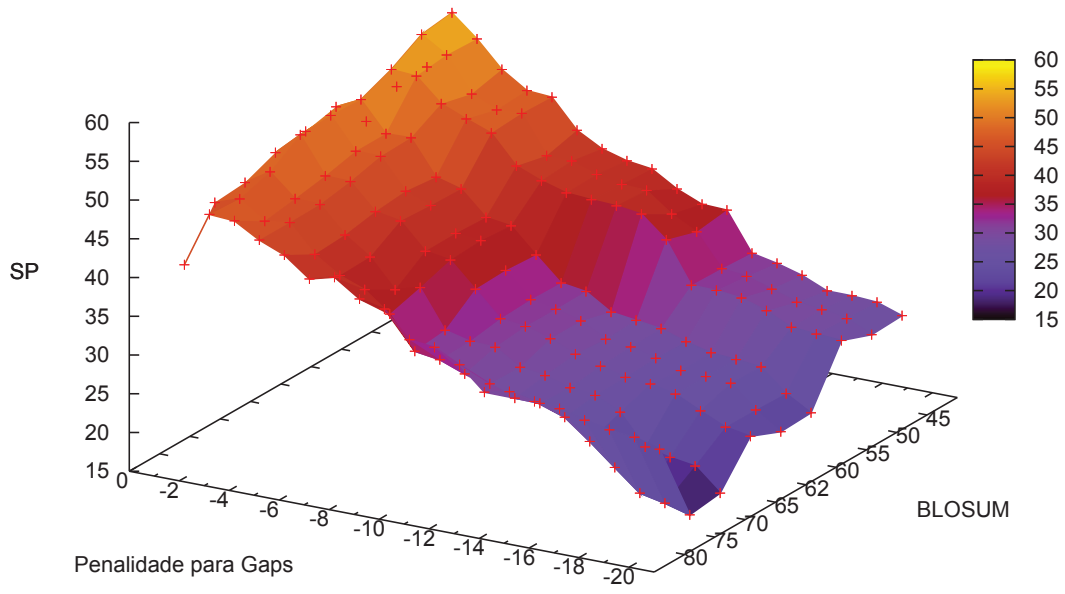
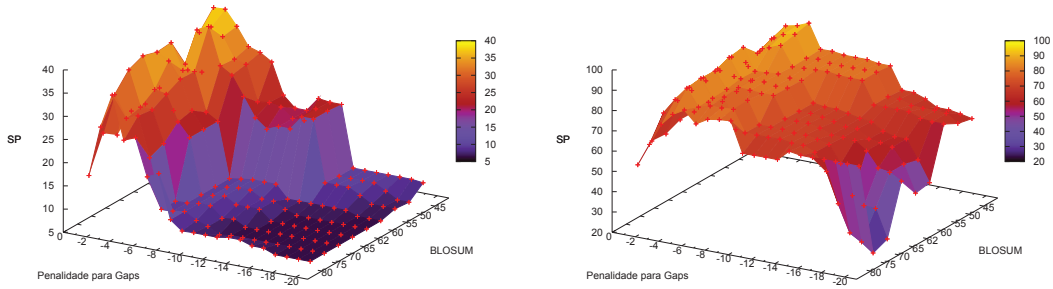
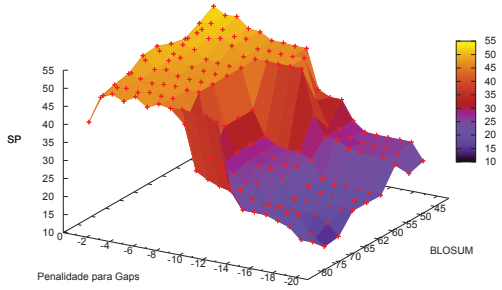


Figura 3.11: Variação na pontuação do alinhador semi-global de perfis (APb) quando variou-se a penalização para *gap* de -1 a -20 e BLOSUM de 45 a 80. Variou-se BLOSUM com incrementos de 5 em 5 e foi também avaliada BLOSUM 62, que é uma matriz comumente utilizada.

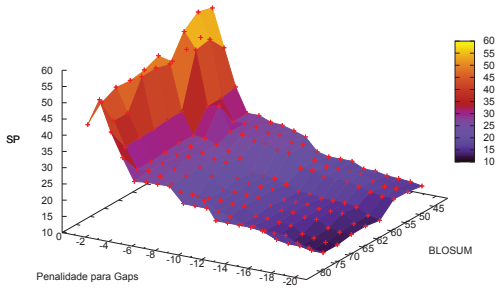


(a) RV11

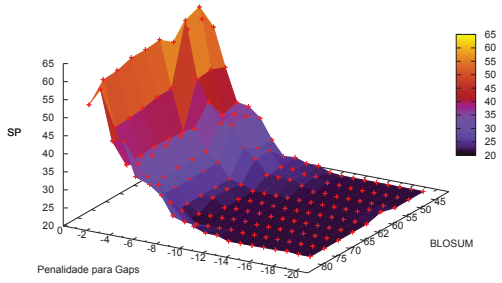
(b) RV12



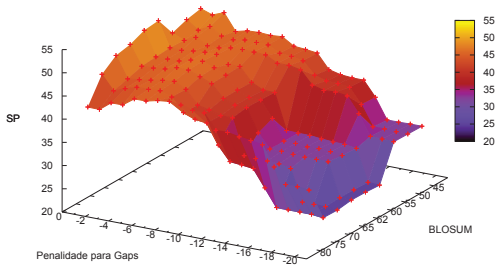
(c) RV20



(d) RV30



(e) RV40



(f) RV50

Figura 3.12: Variação na pontuação do alinhador de perfis com alinhamento semi-global (APb). A penalização para *gap* variou de -1 a -20 e BLOSUM de 45 a 80. Variou-se BLOSUM com incrementos de 5 em 5 e foi também avaliada BLOSUM 62, que é uma matriz comumente utilizada. Em a) tem-se o gráfico para o conjunto RV11, em b) para RV12, em c) para RV20, em d) para RV30, em e) para RV40 e em f) para RV50.

3.5 Ambiente para os testes

Nos testes realizados, usou-se alinhamentos de sequências completas do BAliBASE com pontuação apenas de regiões *core block*. No geral, salvo alguma observação, utilizou-se a pontuação SP para realizar as análises. Essa decisão deve-se a maior flexibilidade e sensibilidade perante à TC.

Os alinhadores progressivos, listados na Tabela B.1 do Apêndice B, foram testados recebendo como entrada dois subconjuntos de referência extraídos do BAliBASE, que foram chamados de RVS1 e RVS2. O conjunto RVS1 possui uma entrada de cada um dos seis conjuntos de referência do BAliBASE. A entrada escolhida é aquela de menor número de sequências. O conjunto RVS2 é constituído de forma semelhante, sendo composto por entradas com o segundo menor número de sequências. A Tabela 3.5 mostra as características das sequências do conjunto RVS1 e a Tabela 3.6 mostra as características das sequências do conjunto RVS2.

Tabela 3.5: Características das sequências do conjunto RVS1. A coluna Entrada indica o nome da entrada. A coluna Sequências indica o número de sequências da entrada. As colunas seguintes indicam os tamanhos mínimo, máximo, médio e a mediana do tamanho das sequências de entrada.

Entrada	Sequências	Mínimo	Máximo	Média	Mediana
BB11001	4	83,00	91,00	86,25	85,50
BB12020	4	118,00	129,00	125,50	127,50
BB20020	16	247,00	527,00	288,19	271,50
BB30017	15	231,00	370,00	313,00	331,00
BB40032	6	259,00	1179,00	846,00	960,50
BB50004	9	386,00	505,00	429,56	415,00

A escolha de um conjunto restrito de sequências do BAliBASE e de entradas com poucas e menores sequências ocorreu devido ao tempo de processamento necessário para executar todos os 342 alinhadores implementados. A Tabela 3.7 mostra o tempo médio, em segundos, de execução dos alinhadores para os conjuntos RVS1 e RVS2 separados por cada uns dos métodos de agrupamento utilizados. Todos os testes foram realizados em computadores `Core 2 Duo` de 2.33GHz e 3GB de memória. Pode-se observar que os alinhadores que utilizam alinhamento de perfis com função de penalidade logarítmica para *gaps* levam maior tempo para execução.

Do conjunto total de alinhadores com 342 alinhadores, há 147 que utilizam função logarítmica para penalidade de *gaps* como método para agrupamento. Estes foram testados apenas com o conjunto RVS1, devido ao grande consumo de tempo. Os 195 alinhadores

Tabela 3.6: Características das sequências do conjunto RVS2. A coluna Entrada indica o nome da entrada. A coluna Sequências indica o número de sequências da entrada. As colunas seguintes indicam os tamanhos mínimo, máximo, médio e a mediana do tamanho das sequências de entrada.

Entrada	**Sequências**	**Mínimo**	**Máximo**	**Média**	**Mediana**
BB11025	4	64,00	103,00	82,75	82,00
BB12021	6	71,00	85,00	75,67	73,00
BB20001	16	74,00	697,00	295,31	212,50
BB30006	18	97,00	923,00	303,39	108,50
BB40010	9	67,00	214,00	109,00	70,00
BB50002	13	172,00	819,00	416,69	400,00

restantes foram avaliados com ambos os conjuntos de entrada. Denomina-se o conjunto completo de alinhadores de Grupo 1 e o conjunto composto pelos 195 alinhadores mais velozes de Grupo 2. Observe que o Grupo 1 utilizou como entrada apenas RVS1 e o Grupo 2 usou como entrada RVS1 e RVS2. Note também que o Grupo 2 está contido no Grupo 1. A Tabela 3.8 apresenta os valores dos parâmetros utilizados em cada método de agrupamento.

Do conjunto de alinhadores progressivos que envolvem algoritmos locais, seja para determinar distância ou para agrupar alinhamentos, 34 haviam passado por testes preliminares contra os alinhadores locais desenvolvidos anteriormente, onde era permitido alinhamento com qualquer tamanho para cada etapa da recursão. Comparou-se os resultados desses alinhadores, utilizando as pontuações SP e TC, com os alinhadores equivalentes que exigem o limite mínimo de 100 resíduos para o tamanho do alinhamento local. Nota-se que, apesar de em alguns casos as pontuações dos alinhadores com restrição de tamanho terem sido inferiores às daqueles que não têm tal restrição, a nova implementação obteve uma melhora média de $7,35\%$ para a pontuação SP e $2,18\%$ para a pontuação TC. A Tabela 3.9 apresenta as pontuações SP e TC para os dez piores alinhadores locais antes e depois da inclusão do limite mínimo para o tamanho do alinhamento.

Tabela 3.7: Tempo médio, em segundos, de execução dos alinhadores para cada entrada dos conjuntos RVS1 e RVS2, agrupados pelos métodos de agrupamento utilizados.

Método	**Tempo RVS1**	**Tempo RVS2**
AC	187,74	401,58
ACb	220,08	39,11
ACLog	192,33	-
ACLogb	191,45	-
LC	92,18	356,35
AP	232,77	591,70
APb	143,80	152,20
APA	246,65	327,21
APAb	131,74	58,68
APLog	4013,65	-
APLogb	3681,00	-
APAp	129,70	994,80

Tabela 3.8: Parâmetros utilizados nos testes dos alinhadores. A primeira coluna indica o método de agrupamento, a última a matriz de substituição e as demais os valores para penalidade para *gaps*. A coluna *c* indica a constante para extensão na penalidade logarítmica. O símbolo "-" indica que o método não usa o parâmetro daquela coluna.

Agrupamento	*gap*	*gop*	*gep*	*c*	**Matriz**
AC	-	-12	-3	-	BLOSUM62
ACb	-	-12	-3	-	BLOSUM62
ACLog	-	-8	-	-5	BLOSUM62
ACLogb	-	-8	-	-5	BLOSUM62
LC	-	12	-3	-	BLOSUM62
AP	-5	-	-	-	BLOSUM62
APb	-2	-	-	-	BLOSUM45
APA	-	-17	-1	-	BLOSUM55
APAb	-	-10	-1	-	BLOSUM45
APLog	-	-9	-	-4	BLOSUM75
APLogb	-	-8	-	-5	BLOSUM62
APAp	-	-17	-1	-	BLOSUM55

Tabela 3.9: Pontuações dos dez piores alinhadores que usam agrupamento local, antes e depois da inclusão do limite mínimo de 100 resíduos para o tamanho do alinhamento.

Alinhador	**Antes**		**Depois**	
	SP	**TC**	**SP**	**TC**
160	35,56	18,92	47,24	25,42
159	37,20	22,75	47,66	25,42
154	37,83	22,67	52,85	21,50
147	38,12	22,67	45,39	22,75
148	39,57	21,75	53,06	21,50
153	43,37	23,75	45,50	22,75
162	46,10	23,42	62,71	39,50
150	47,09	24,08	69,83	44,08
156	50,06	25,00	69,83	44,08
221	51,46	28,58	50,15	24,42

3.6 Resultados

Na Tabela 3.10 apresenta-se as pontuações SP mínima, máxima, média e mediana agrupada por cada método utilizado para alinhadores do Grupo 1. Já na Tabela 3.11 apresenta-se valores equivalentes para alinhadores do Grupo 2.

Tabela 3.10: Pontuação SP mínima, máxima, média e mediana do conjunto RVS1 para cada um dos métodos utilizados nos alinhadores progressivos, do Grupo 1.

Categoria	Método	Mínimo	Máximo	Média	Mediana
Distância	JTT	52,40	82,87	70,44	71,53
	PAM	49,87	82,13	69,79	70,87
	PCM	52,50	83,40	70,47	70,73
	PMB	52,33	82,63	70,42	71,67
	LD	47,27	76,88	61,26	61,58
	LOGD	39,07	73,22	54,93	52,73
Árvore	NJ	48,37	82,72	68,14	70,53
	UPGMA	39,07	83,40	65,33	67,00
Seleção de Pares	BU	47,02	82,08	65,99	65,92
	NP	39,07	83,40	66,71	67,95
Agrupamento	AC	43,88	75,70	67,52	70,89
	ACb	43,88	58,30	53,27	53,72
	ACLog	58,88	74,50	69,04	72,19
	ACLogb	57,17	72,37	66,71	66,59
	LC	57,18	68,18	62,19	61,92
	AP	39,75	71,65	64,04	68,75
	APb	39,75	57,83	53,17	54,50
	APA	53,65	83,40	77,48	80,81
	APAb	53,65	82,87	73,62	76,44
	APAp	47,43	82,08	72,13	79,19
	APLog	39,07	68,17	62,79	65,78
	APLogb	39,07	75,77	68,81	72,27
Esquema de pesos	PM	45,43	82,63	68,22	69,47
	PP	39,07	83,40	65,70	66,90

Nas tabelas 3.10 e 3.11 pode-se observar que os métodos do PHYLIP para cálculo de matriz de distâncias (JTT, PAM, PCM e PMB) apresentaram resultados superiores e muito próximos. Dentre eles, o valor que destaca-se um pouco mais é a mediana para o PAM na Tabela 3.11.

O método *Neighbor Joining* mostrou-se um pouco superior ao UPGMA. Observe que

Tabela 3.11: Pontuação SP mínima, máxima, média e mediana dos conjuntos RVS1 e RVS2 para cada um dos métodos utilizados nos alinhadores progressivos, do Grupo 2.

Categoria	Método	Mínimo	Máximo	Média	Mediana
Distância	JTT	42,22	67,39	55,72	55,00
	PAM	42,26	66,84	55,82	59,01
	PCM	41,82	67,11	55,72	56,38
	PMB	42,08	66,16	55,50	55,10
	LD	31,26	62,21	44,50	44,41
Árvore	NJ	32,77	65,29	52,66	52,92
	UPGMA	31,26	67,39	53,26	52,85
Seleção de Pares	BU	36,14	67,11	52,00	51,66
	NP	31,26	67,39	52,96	52,92
Agrupamento	AC	48,18	56,73	53,76	54,29
	ACb	41,59	48,81	45,33	45,56
	LC	44,61	52,65	49,37	49,01
	AP	40,62	53,97	50,40	51,76
	APb	31,26	44,78	41,33	42,53
	APA	54,23	65,29	62,95	63,67
	APAb	42,88	67,39	60,70	63,34
	APAp	32,87	64,83	52,04	60,88
Esquema de pesos	PM	31,49	67,07	53,93	53,71
	PP	31,26	67,39	51,93	51,67

a mediana é maior em ambas as situações, sendo mais contundente nos resultados apresentados na Tabela 3.10.

Quanto ao método de seleção de pares, é observada uma pequena superioridade do método de seleção do par mais próximo, principalmente quando observa-se a primeira tabela.

Em ambas as tabelas, pode-se notar a superioridade de alinhamento de perfis com função afim para penalidade de *gaps* dentre os métodos de agrupamento. A versão que traz o método para ajuste automático de parâmetros (APAp), apesar de apresentar bons resultados, é superada por versões mais simples do alinhamento de perfis, tais como: APA e APAb. O esquema de pesos mostrou-se eficiente.

Na Tabela 3.10 pode-se observar que, dentre os alinhadores de agrupamento por consenso, os que fazem uso de função logarítmica para penalidade de *gaps* apresentam melhor desempenho.

Os alinhadores que alcançaram as vinte melhores pontuações médias para SP e TC, ao utilizar o Grupo 1 de alinhadores nos testes, são listados na Tabela 3.12. O melhor

alinhador foi o 125, que obteve pontuação média de 83.40 para SP e 64.67 para TC. Ele utiliza PCM para calcular a matriz de distâncias, UPGMA para construção de árvore guia, par mais próximo para seleção de pares e alinhamento de perfis global com função afim para penalidade de *gaps*. Observa-se na tabela que os vinte melhores alinhadores utilizam alinhamento de perfis global ou semi-global com função afim para penalidade de *gaps*, assim como um dos métodos para cálculo de matriz de distância implementado pelo PHYLIP (JTT, PAM, PCM ou PMB). Observa-se também a maior presença de par mais próximo para seleção de pares. Seleção por bloco único aparece apenas em 4 dos 20 alinhadores. Quanto aos métodos para construção de árvore guia, *Neighbor Joining* destacou-se. Interessante observar que o esquema de pesos (PM) aparece em apenas 6 dos melhores alinhadores. Isso é um tanto quanto contraditório se observarmos as tabelas 3.10 e 3.11.

Os alinhadores que alcançaram as vinte melhores pontuações médias para SP e TC, ao utilizar o Grupo 2 de alinhadores nos testes, são listados na Tabela 3.13. Resultados equivalentes foram obtidos para cálculo da matriz de distâncias e para agrupamento de alinhamentos. Seleção de par mais próximo continuou a ser predominante, mas com margem bem menor em relação a bloco único. UPGMA agora esteve mais presente que *Neighbor Joining*. Dessa vez, entretanto, o esquema de pesos apresentou um desempenho superior.

As tabelas 3.14 e 3.15 apresentam os vinte piores alinhadores utilizando os grupos 1 e 2, respectivamente. Agora, observa-se que os métodos do PHYLIP para cálculo de distâncias praticamente desaparecem, assim como os alinhadores de perfis com função afim para penalidade de *gaps*. Destaque para a grande presença de UPGMA e par mais próximos nos piores alinhadores.

Os resultados presentes nas tabelas 3.12, 3.13, 3.14 e 3.15 permitem observar que um método em particular não torna o alinhador bom, mas sim um bom conjunto de métodos e parâmetros de configuração. Mesmo assim, é possível dizer que utilizar um dos métodos do PHYLIP (JTT, PAM, PCM ou PMB) para cálculo de matriz de distâncias juntamente com alinhamento de perfis com função afim para penalidade de *gaps* e tendo o esquema de pesos habilitado é um bom começo para um alinhador progressivo. Independentemente do conjunto de métodos, bastante cuidado deve ser tomado com a escolha dos parâmetros.

Tabela 3.12: Pontuação SP média na segunda coluna e TC na terceira coluna, dos vinte melhores alinhadores, do Grupo 1, testados com o conjunto RVS1. CD indica os métodos para o cálculo da matriz de distância, AG indica os métodos para construção da árvore guia, SL indica os métodos para seleção de pares, MA indica os métodos de agrupamento e PS indica o esquema de pesos das sequências.

Alinhador	**SP**	**TC**	**CD**	**AG**	**SL**	**MA**	**PS**
125	83,40	64,67	PCM	UPGMA	NP	APA	PP
053b	82,87	64,00	JTT	UPGMA	NP	APAb	PP
137	82,72	64,50	PCM	NJ	NP	APA	PP
053	82,67	64,83	JTT	UPGMA	NP	APA	PP
077b	82,63	63,00	PMB	UPGMA	NP	APAb	PP
077	82,43	64,17	PMB	UPGMA	NP	APA	PP
114	82,13	63,50	PAM	NJ	NP	APA	PM
113	82,08	64,17	PAM	NJ	NP	APA	PP
059p	82,08	64,50	JTT	–	BU	APA	PP
083p	82,05	65,50	PMB	–	BU	APA	PP
065	82,03	64,17	JTT	NJ	NP	APA	PP
138	82,03	62,83	PCM	NJ	NP	APA	PM
101b	82,03	62,67	PAM	UPGMA	NP	APAb	PP
089	81,75	63,50	PMB	NJ	NP	APA	PP
066	81,70	63,50	JTT	NJ	NP	APA	PM
060p	81,65	61,67	JTT	–	BU	APA	PM
138p	81,63	65,17	PCM	NJ	NP	APA	PM
137p	81,60	63,00	PCM	NJ	NP	APA	PP
059	81,47	64,33	JTT	–	BU	APA	PP
090	81,42	64,17	PMB	NJ	NP	APA	PM

Tabela 3.13: Pontuação SP média na segunda coluna e TC na terceira coluna dos vinte melhores alinhadores, do Grupo 2, testados com o conjunto RVS1 e RVS2. CD indica os métodos para o cálculo da matriz de distância, AG indica os métodos para construção da árvore guia, SL indica os métodos para seleção de pares, MA indica os métodos de agrupamento e PS indica o esquema de pesos das sequências.

Alinhador	**SP**	**TC**	**CD**	**AG**	**SL**	**MA**	**PS**
053b	67,39	43,42	JTT	UPGMA	NP	APAb	PP
131b	67,11	42,33	PCM	-	BU	APAb	PP
132b	67,07	41,42	PCM	-	BU	APAb	PM
101b	66,84	42,58	PAM	UPGMA	NP	APAb	PP
077b	66,16	42,00	PMB	UPGMA	NP	APAb	PP
083b	65,70	40,08	PMB	-	BU	APAb	PP
060b	65,47	41,25	JTT	-	BU	APAb	PM
066	65,29	42,92	JTT	NJ	NP	APA	PM
138	65,24	42,25	PCM	NJ	NP	APA	PM
090	65,22	42,58	PMB	NJ	NP	APA	PM
084b	65,21	39,50	PMB	-	BU	APAb	PM
125b	65,15	41,08	PCM	UPGMA	NP	APAb	PP
059b	65,14	40,33	JTT	-	BU	APAb	PP
114	65,11	42,67	PAM	NJ	NP	APA	PM
126b	65,10	40,83	PCM	UPGMA	NP	APAb	PM
060p	64,83	43,33	JTT	-	BU	APA	PM
126	64,68	42,25	PCM	UPGMA	NP	APA	PM
125	64,60	40,75	PCM	UPGMA	NP	APA	PP
108b	64,53	38,83	PAM	-	BU	APAb	PM
053	64,33	40,83	JTT	UPGMA	NP	APA	PP

Tabela 3.14: Pontuação SP média na segunda coluna e TC na terceira coluna dos vinte piores alinhadores, do Grupo 1, testados com o conjunto RVS1. CD indica os métodos para o cálculo da matriz de distância, AG indica os métodos para construção da árvore guia, SL indica os métodos para seleção de pares, MA indica os métodos de agrupamento e PS indica o esquema de pesos das sequências.

Alinhador	**SP**	**TC**	**CD**	**AG**	**SL**	**MA**	**PS**
160b	49,48	24,00	LD	NJ	NP	APb	PM
159b	49,22	24,17	LD	NJ	NP	APb	PP
331	48,68	21,33	LOGD	–	BU	AP	PP
331b	48,68	21,33	LOGD	–	BU	APb	PP
339	48,37	23,67	LOGD	NJ	NP	AP	PP
339b	48,37	23,67	LOGD	NJ	NP	APb	PP
148b	47,80	22,33	LD	UPGMA	NP	APb	PM
325p	47,47	19,67	LOGD	UPGMA	NP	APA	PP
326p	47,43	19,17	LOGD	UPGMA	NP	APA	PM
147b	47,27	21,83	LD	UPGMA	NP	APb	PP
329	47,02	6,83	LOGD	–	BU	AC	PP
329b	47,02	6,83	LOGD	–	BU	ACb	PP
324	45,43	19,00	LOGD	UPGMA	NP	AP	PM
324b	45,43	19,00	LOGD	UPGMA	NP	APb	PM
321	43,88	7,17	LOGD	UPGMA	NP	AC	PP
321b	43,88	7,17	LOGD	UPGMA	NP	ACb	PP
323	39,75	13,83	LOGD	UPGMA	NP	AP	PP
323b	39,75	13,83	LOGD	UPGMA	NP	APb	PP
327	39,07	26,83	LOGD	UPGMA	NP	APLog	PP
327b	39,07	26,83	LOGD	UPGMA	NP	APLogb	PP

Tabela 3.15: Pontuação SP média na segunda coluna e TC na terceira coluna dos vinte piores alinhadores, do Grupo 2, testados com o conjunto RVS1 e RVS2. CD indica os métodos para o cálculo da matriz de distância, AG indica os métodos para construção da árvore guia, SL indica os métodos para seleção de pares, MA indica os métodos de agrupamento e PS indica o esquema de pesos das sequências

Alinhador	**SP**	**TC**	**CD**	**AG**	**SL**	**MA**	**PS**
082b	42,25	17,58	PMB	–	BU	APb	PM
051b	42,22	16,50	JTT	UPGMA	NP	APb	PP
081b	42,22	17,75	PMB	–	BU	APb	PP
076b	42,08	16,17	PMB	UPGMA	NP	APb	PM
157b	41,99	14,92	LD	NJ	NP	ACb	PP
099b	41,89	16,67	PAM	UPGMA	NP	APb	PP
123b	41,82	16,42	PCM	UPGMA	NP	APb	PP
159	41,73	20,17	LD	NJ	NP	AP	PP
151b	41,59	12,42	LD	–	BU	ACb	PP
160	41,48	20,17	LD	NJ	NP	AP	PM
153	40,68	18,08	LD	–	BU	AP	PP
147	40,62	18,08	LD	UPGMA	NP	AP	PP
153b	39,14	15,64	LD	–	BU	APb	PP
162p	38,13	18,67	LD	NJ	NP	APA	PM
161p	37,98	19,50	LD	NJ	NP	APA	PP
154b	37,32	14,83	LD	–	BU	APb	PM
160b	35,65	14,83	LD	NJ	NP	APb	PM
159b	32,77	14,58	LD	NJ	NP	APb	PP
148b	31,49	13,08	LD	UPGMA	NP	APb	PM
147b	31,26	12,83	LD	UPGMA	NP	APb	PP

Capítulo 4

Alinhamento múltiplo baseado em consistência

O primeiro método, para construção de MSAs, baseado em consistência foi descrito por Kececioglu na década de 1990 [138]. Dado um conjunto de sequências, o MSA ótimo é definido como aquele que está de acordo com a maioria de todos os possíveis alinhamentos ótimo de pares. Há, pelo menos, três boas razões que fazem OFs baseadas em consistência interessantes, são elas: não dependem de uma matriz de substituição específica, mas de qualquer método, ou coleção de métodos, capaz de alinhar duas sequências por vez; o esquema baseado em consistência é dependente de posição; e experiências mostraram que dado um conjunto de observações independentes, a maioria dos consistentes estão frequentemente próximos da verdade [65].

Notredame e colegas deram continuidade ao trabalho de Kececioglu. Eles apresentaram algoritmos heurísticos capazes de otimizar a função definida por Kececioglu. Inicialmente avaliaram o desempenho da função no SAGA [182], um algoritmo genético para alinhamento múltiplo de sequências previamente desenvolvido pelo grupo. Uma vez observada a boa qualidade dos resultados, passaram a desenvolver um método capaz de otimizar a função de forma mais eficiente, em termos de tempo. Implementaram um novo alinhador, que chamaram de T-COFFEE [183]. Este, por sua vez, inspirou o desenvolvimento de um novo alinhador, chamado ProbCons [60], por um outro grupo.

De uma forma geral, ProbCons funciona de forma semelhante ao T-COFFEE, mas usa probabilidade em vez da heurística para pesos de pares de resíduos. Para tanto usa HMM de pares de sequências gerados por uma função objetivo de máxima precisão esperada [121]. Faz uso de um algoritmo de alinhamento progressivo, seguido de um procedimento iterativo de refinamento.

MUMMALS [196] é uma outra ferramenta para construção de MSAs que usa uma medida de consistência probabilística juntamente com alinhamento de pares de sequências

através de HMM. Pei e Grishin [196], seus desenvolvedores, observam que alinhamentos produzidos pelo MUMMALS superam em qualidade os alinhamentos gerados por ferramentas tais como Clustal W [245], MUSCLE [69] e ProbCons [60]. Os modelos ocultos de Markov empregados no MUMMALS implementam múltiplos estados de *match*, que descrevem as estruturas locais sem realizar uma predição de estrutura explicitamente. Para estimar os parâmetros a serem utilizados pelo HMM, foi empregado um método de aprendizado supervisionado, usando um grande conjunto de alinhamentos estruturais, construídos pelo DaliLite [119], a partir de pares de domínio divergentes presentes no SCOP [171]. Para maiores informações sobre a abordagem baseada em consistência, consulte a Seção 2.3.3.

No restante deste capítulo, são detalhadas modificações realizadas no MUMMALS com o objetivo de melhorar a qualidade dos alinhamentos gerados pela ferramenta. Na Seção 4.1 o algoritmo do MUMMALS é descrito. As 89 variações no algoritmo original do MUMMALS, agrupadas em três experimentos distintos, são detalhadas na Seção 4.2. Na Seção 4.3 expõem-se os resultados e finalmente as conclusões são apresentadas na Seção 4.4.

4.1 O algoritmo do MUMMALS

O HMM padrão para alinhamento de pares de sequências tem três estados emitindo resíduos: um único estado de *match* 'M' emitindo um par de resíduos, um estado 'X' emitindo resíduos na primeira sequência e um estado 'Y' emitindo resíduos na segunda sequência [64]. Esse modelo é chamado $HMM\_1\_1\_0$. Nos modelos do MUMMALS, novos estados são introduzidos baseados em alinhamentos estruturais. Pares de resíduos alinhados em um *core block* são modelados por um estado de *match* 'M'. Um *core block* é uma região da sequência onde há um alinhamento estrutural. No caso do par de resíduos estar alinhado em uma outra região, é modelado pelo estado de *match* 'U'. Esse modelo é chamado $HMM\_1\_1\_1$. A Figura 4.1 compara o modelo HMM padrão para alinhamento de par de sequências com aqueles propostos no MUMMALS [196]. Observe na subfigura (a) a identificação dos *core blocks* e como seus estados variam de um modelo para outro. Subfiguras (b) e (c) representam respectivamente a estrutura de $HMM\_1\_1\_0$ e $HMM\_1\_1\_1$, com seus estados e transições.

Há ainda três outros modelos mais complexos no MUMMALS. $HMM\_1\_3\_1$ é um deles, onde estados de *match* são criados de acordo com os tipos de estruturas secundárias. Nesse modelo o estado de *match* 'M' é removido e três outros estados são adicionados: 'H' (*helix*), 'S' (*strand*) e 'C' (*coil*). Por exemplo, se um par de resíduos ocorre em uma região de uma hélice, o estado de *match* associado será 'H', como detalhado na Figura 4.1. De forma similar, no modelo $HMM\_3\_1\_1$, múltiplos estados de *match* são introduzidos

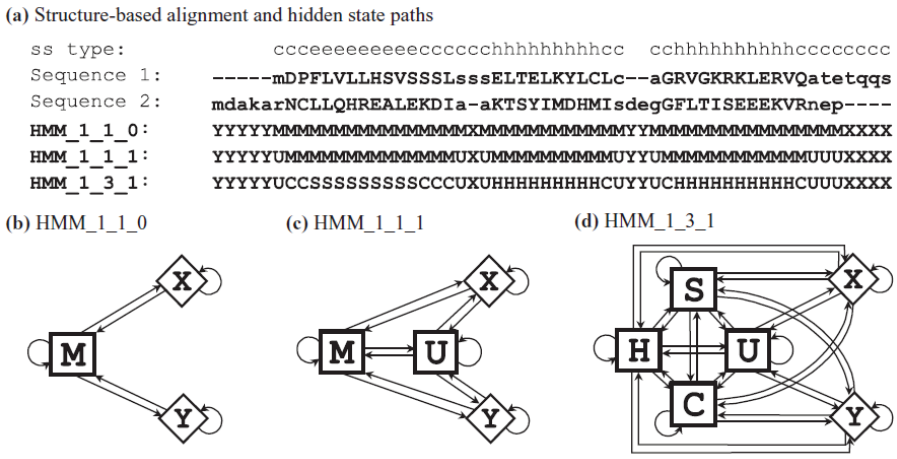


Figura 4.1: (a) Uma ilustração do alinhamento de sequências baseado em estrutura e esquema de diversos HMMs empregados pelo MUMMALS. Em *Sequence 1* e *Sequence 2*, letras maiúsculas e minúsculas representam, respectivamente, *core blocks* alinhados e regiões não alinhadas. Os tipos de estrutura secundária, identificados por *ss type*, referem-se à primeira sequência. (b) Apresenta um esquema com os estados e transições entre eles para o modelo $HMM\_1\_1\_0$. (c) Apresenta um esquema referente ao modelo $HMM\_1\_1\_1$. (d) Por fim, apresenta o esquema do modelo $HMM\_1\_3\_1$. Essa ilustração foi extraída do trabalho de Pei e Grishin [196].

baseado nas categorias de acessibilidade de solvente. O modelo mais complexo, chamado $HMM\_3\_3\_1$, combina os dois últimos modelos. Um método de aprendizagem supervisionada aplicado a um conjunto de alinhamentos estruturais é usado para estimar os parâmetros (probabilidades de transição e emissão) para o modelo.

Alinhamento progressivo usando uma função de pontuação baseada em consistência probabilística, similar àquela empregada pelo ProbCons [60], é usado para construir o MSA. Em primeiro lugar uma matriz de distâncias é computada baseada no método de contagem de k-mer [69]. Em seguida uma árvore é construída usando o método UPGMA [228]. O próximo passo é a computação da medida de consistência probabilística e finalmente as sequências são progressivamente alinhadas, na ordem definida pela árvore, usando a função de pontuação baseada em consistência. Para balancear adequadamente a velocidade e a precisão do alinhamento, é aplicada uma estratégia de alinhamento em dois estágios similar àquela empregada no PCMA [199]. No primeiro estágio, sequências com alto grau de similaridade são progressivamente alinhadas rapidamente sem a pontuação baseada em consistência. A função de pontuação nesse estágio é uma soma dos pares com peso usando BLOSUM62 como matriz de substituição. Durante o segundo estágio do alinhamento, as sequências (ou grupos pré-alinhados) são submetidos à medida de consistência probabilística, que consome mais tempo.

Tabela 4.1: Os alfabetos comprimidos avaliados neste estudo. Nas implementações realizadas, foram utilizadas letras do alfabeto como códigos para as classes. A primeira classe foi denominada A, a segunda B, a terceira C e assim sucessivamente.

Alfabeto	**Classes**
Dayhoff(6)	AGPST,C,DENQ,FWY,HKR,ILMV
SE-B(6)	AST,CP,DEHKNQR,FWY,G,ILMV
SE-B(8)	AST,C,DHN,EKQR,FWY,G,ILMV,P
Li-A(10)	AC,DE,FWY,G,HN,IV,KQR,LM,P,ST
Li-B(10)	AST,C,DEQ,FWY,G,HN,IV,KR,LM,P
Murphy(10)	A,C,DENQ,FWY,G,H,ILMV,KR,P,ST
SE-B(10)	AST,C,DN,EQ,FY,G,HW,ILMV,KR,P
SE-V(10)	AST,C,DEN,FY,G,H,ILMV,KQR,P,W
Solis-D(10)	AM,C,DNS,EKQR,F,GP,HT,IV,LY,W
Solis-G(10)	AEFIKLMQRVW,C,D,G,H,N,P,S,T,Y
SE-B(14)	A,C,D,EQ,FY,G,H,IV,KR,LM,N,P,ST,W

Os diversos modelos ocultos de Markov foram avaliados [196] e apresentaram um bom desempenho quando comparados a ProbCons [60], MUSCLE [69], MAFFT [135] e Clustal W [245]. Entre os modelos do MUMMALS, $HMM\_1\_3\_1$ e $HMM\_3\_3\_1$ destacaram-se. Dentre os conjuntos de testes com identidade abaixo de 20%, MUMMALS superou todos os outros alinhadores. No restante dos casos, MAFFT frequentemente apresentou o melhor desempenho. Dentre os dois melhores modelos do MUMMALS, $HMM\_1\_3\_1$ é a melhor opção. Constrói alinhamentos quase tão precisos quanto $HMM\_3\_3\_1$, porém executando cerca de três vezes mais rápido. $HMM\_1\_3\_1$ foi usado como referência para as avaliações realizadas neste trabalho.

O método de contagem k-mer, conforme aplicado ao MUMMALS, converte as sequências de entrada de acordo com um alfabeto comprimido. Dayhoff(6) foi usado como alfabeto com $k = 6$. Esse método (contagem k-mer) converte as sequências de entrada, compostas de um alfabeto de 20 resíduos (uma sequência de aminoácidos), em sequências compostas de um alfabeto de seis classes – no caso de Dayhoff(6). Classes são definidas por grupos de resíduos com propriedades similares. Em seguida uma estrutura é construída, para cada sequência, com a contagem das ocorrências das *substrings*. As *substrings* são de comprimento k, no caso aqui 6. A distância entre um par de sequências é calculada baseada nas diferenças em suas respectivas estruturas.

Um exemplo de uma sequência de aminoácidos convertida de acordo com Dayhoff(6) é apresentado a seguir.

```
Original:   MDPFLVLLHSVSSSLSSSELTELKYLCLCAGRVGKRKLERVQATE
Convertida: FCADFFFFEAFAAAFAAACFACFEDFBFBAAEFAEEEFCEFCAAC
```

Nesse exemplo a primeira *substring* com $k = 6$ é FCADFF e a segunda é CADFFF. Os alfabetos usados neste estudo são apresentados na Tabela 4.1.

4.2 Alterações realizadas

Três avaliações distintas foram executadas. Cada uma delas avaliou uma determinada alteração no algoritmo original do MUMMALS.

Durante a fase de planejamento do teste algumas questões surgiram, tais como: "Poderia uma alteração no valor de k levar a uma variação significativa na pontuação do MSA?" ou "Poderia uma alteração no valor de k afetar o tempo de execução do algoritmo?". Inicialmente foi avaliada uma versão com o valor do k variando entre 3 e 14, o que totalizou 11 variações no algoritmo original do MUMMALS. O limite inferior foi definido em 3, porque *substrings* abaixo desse valor não são significantes, e o limite superior foi definido em 14, devido à relação entre tempo de execução e qualidade do resultado. O objetivo desse teste foi visualizar o efeito da alteração do comprimento das *substrings*. Como será visto na Seção 4.3, as respostas para as questões iniciais foram positivas.

No segundo experimento, alfabetos comprimidos alternativos foram avaliados. Os alfabetos utilizados nesta avaliação, cujas classes são apresentadas na Tabela 4.1, foram SE-B(6), SE-B(8), Li-A(10), Li-B(10), Murphy(10), SE-B(10), SE-V(10), Solis-D(10), Solis-G(10) e SE-B(14). Nesse experimento escolheu-se variar k de 6 a 10 porque esses foram os valores que apresentaram a melhor relação entre tempo de execução e qualidade do resultado no experimento anterior. Nessa etapa foram avaliadas 50 variações no MUMMALS. Foram 10 alfabetos e, para cada um deles, 5 valores para k. Para maiores informações sobre os alfabetos comprimidos, consulte o estudo de Robert C. Edgar [67].

No último experimento, a parte inicial do alinhamento progressivo foi completamente alterada. Os métodos para computação de matriz de distâncias e construção de árvore guia foram redefinidos de acordo com um estudo prévio, detalhado no Capítulo 3. Neste, métodos aplicados em cada etapa do alinhamento progressivo foram avaliados e, dentre estes, algoritmos para computação de matriz de distâncias e algoritmos para construção de árvore guia foram comparados. Nesse experimento, dois novos alinhadores foram implementados, ambos usando o método PAM [56], disponibilizado pelo pacote PHYLIP [73], para computação de matriz de distâncias. O primeiro dos alinhadores usa *Neighbor Joining* (NJ) [210] para a construção da árvore guia. Já o segundo usa UPGMA [228] em vez de NJ. Em ambos os alinhadores foram avaliadas possíveis normalizações para os valores das distâncias. Foram testados 13 intervalos distintos tendo sempre 1,0 como limite superior, mas variando o limite inferior de 0,0 a 0,9 com incremento de 0,1 entre eles. Também foram avaliados os limites inferiores 0,65, 0,75 e 0,85, pois os melhores resultados foram obtidos com o limite inferior 0,6 ou maior. No total foram 28 variações no MUMMALS

para esse experimento.

O método de contagem k-mer com um alfabeto comprimido para computação de distância, foi usado no algoritmo do MUMMALS objetivando uma redução no tempo requerido para a computação da matriz de distâncias [67]. Entretanto, teoricamente esse método não possui a mesma precisão de métodos clássicos. A terceira avaliação objetivou verificar a possibilidade de ganho de precisão, quando são usados métodos clássicos para computação de distâncias. Verificou-se também o custo adicional em termos de tempo de execução.

4.3 Resultados

Durante a avaliação, todas as 218 entradas de teste, compostas por sequências completas, providas pelo BAliBASE 3.0 [246] foram usadas. Os métodos SP – também conhecido como Q-score – e TC foram empregados para a pontuação dos alinhamentos e foram considerados apenas os *core blocks*.

Na Tabela 4.2 são apresentados os resultados para o primeiro teste, onde variou-se o valor do k e o alfabeto comprimido padrão do MUMMALS, Dayhoff(6), foi mantido. Usando o MUMMALS original, foram necessárias aproximadamente 14 horas ($50.594s$) para construir os 218 alinhamentos em um computador com processador `Dual Core` de $2{,}5GHz$ e $3GB$ de memória. Atingiu-se uma pontuação SP média de 85,54 e uma pontuação TC média de 53,83. Na Tabela 4.2, o valor Δt é a variação no tempo de execução quando comparado ao MUMMALS original. Por exemplo, quando $k = 8$ o tempo requerido foi $81.017s$, 60,13% maior. Os valores ΔSP e ΔTC são as variações nas pontuações e a coluna ΔA exibe a média entre ΔSP e ΔTC. Uma maneira alternativa de observar a oscilação no desempenho é concentrar na variação do erro. Quando obtém-se uma pontuação SP 85,00, tem-se uma solução com uma taxa de erros de 15%. Assim se um novo algoritmo obtém uma pontuação SP 90,00, tem-se uma solução com 33,33% menos erros quando comparada à anterior. As colunas ΔSPe e ΔTCe apresentam a variação de erros, usando pontuação SP e TC, respectivamente, e a coluna ΔAe exibe a média entre ΔSPe e ΔTCe. Observe que a pontuação SP mais alta ocorre quando $k = 13$ ou $k = 14$. Com $k = 8$ foi obtida a melhor pontuação TC. Note que o melhor ΔA ocorre quando $k = 8$ e o melhor ΔAe aparece quando $k = 13$. Um ganho de desempenho usando a pontuação SP tem uma maior significância sob uma perspectiva de variação de erro pois o MUMMALS original tem uma pontuação SP elevada.

Os resultados da avaliação dos alfabetos alternativos são apresentados na Tabela 4.3. Observe que os melhores valores para SP e TC foram obtidos com SE-B(10) e $k = 7$. Requeriu $87.692s$ para alcançar uma pontuação SP de 86,70 e uma pontuação TC de 56,52.

Tabela 4.2: Resultados com alfabeto Dayhoff(6), quando o parâmetro k variou entre 3 e 14.

Alfabeto	**k**	Δt	ΔSP	ΔTC	ΔA	ΔSPe	ΔTCe	ΔAe
Dayhoff(6)	3	-93,46	-30,05	-56,94	-43,50	177,83	66,40	122,11
Dayhoff(6)	4	-92,53	-25,32	-46,17	-35,74	149,85	53,83	101,84
Dayhoff(6)	5	-79,93	-9,31	-19,07	-14,19	55,07	22,24	38,65
Dayhoff(6)	6	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Dayhoff(6)	7	38,73	1,08	2,82	1,95	-6,37	-3,29	-4,83
Dayhoff(6)	8	60,13	0,85	**3,47**	**2,16**	-5,05	**-4,05**	-4,55
Dayhoff(6)	9	75,60	0,94	3,00	1,97	-5,57	-3,50	-4,54
Dayhoff(6)	10	89,80	1,06	3,22	2,14	-6,30	-3,76	-5,03
Dayhoff(6)	11	98,92	1,23	2,52	1,87	-7,27	-2,94	-5,10
Dayhoff(6)	12	110,48	1,08	2,75	1,91	-6,39	-3,20	-4,80
Dayhoff(6)	13	120,64	**1,36**	1,86	1,61	**-8,08**	-2,17	**-5,12**
Dayhoff(6)	14	125,92	**1,36**	1,41	1,39	-8,07	-1,65	-4,86
	Melhores		1,36	3,47	2,16	-8,08	-4,05	-5,12

Finalmente, os resultados para a versão com os métodos para computação da matriz de distâncias e construção de árvore guia alterados são apresentados na Tabela 4.4 e na Tabela 4.5. Usou-se uma variedade de faixas de normalização para os valores das distâncias. Observe que, quando utiliza-se PAM+NJ e valores normalizados entre $0,7$ e $1,0$, os melhores resultados foram alcançados para essa avaliação. A versão PAM+UPGMA mostrou um desempenho inferior e provou ser menos sensível à faixa de normalização, pois diversas versões obtiveram um desempenho similar.

4.4 Conclusões

Melhorias significativas foram obtidas com alterações no algoritmo do MUMMALS. Os três experimentos realizados levaram a resultados mais precisos.

No primeiro teste, cujos resultados foram apresentados na Tabela 4.2, obteve-se 8,08% de redução de erros ao utilizar a pontuação SP e 4,05% menos erros ao utilizar a pontuação TC.

O segundo teste, cujos resultados foram exibidos na Tabela 4.3, extraiu os melhores resultados. O alinhador com alfabeto comprido SE-B(10) e $k = 7$ obteve pontuação SP 86,70 e pontuação TC 56,52. Em outras palavras, ele reduziu erros em 7,98% numa avaliação pela pontuação SP e obteve 5,81% menos erros numa perspectiva de pontuação TC.

O último teste, cujos resultados foram expostos na Tabela 4.4 e na Tabela 4.5, obteve

Tabela 4.3: Resultados para os testes com os alfabetos comprimidos alternativos.

Alfabeto	**k**	Δt	ΔSP	ΔTC	ΔA	ΔSPe	ΔTCe	ΔAe
SE-B(6)	6	-3,33	0,46	1,91	1,19	-2,73	-2,23	-2,48
SE-B(6)	7	30,89	0,65	3,08	1,86	-3,83	-3,59	-3,71
SE-B(6)	8	52,18	0,86	3,42	2,14	-5,08	-3,99	-4,53
SE-B(6)	9	68,08	1,02	4,27	2,65	-6,06	-4,98	-5,52
SE-B(6)	10	82,99	0,92	4,38	2,65	-5,46	-5,11	-5,29
SE-B(8)	6	33,47	0,89	4,76	2,83	-5,26	-5,55	-5,41
SE-B(8)	7	59,15	0,77	4,55	2,66	-4,53	-5,30	-4,92
SE-B(8)	8	76,88	1,19	4,57	2,88	-7,03	-5,33	-6,18
SE-B(8)	9	90,12	1,23	4,28	2,76	-7,28	-4,99	-6,14
SE-B(8)	10	102,52	1,30	3,04	2,17	-7,67	-3,55	-5,61
Li-A(10)	6	73,93	0,89	3,88	2,39	-5,28	-4,53	-4,90
Li-A(10)	7	94,16	1,21	3,55	2,38	-7,13	-4,14	-5,64
Li-A(10)	8	112,56	1,27	4,23	2,75	-7,52	-4,93	-6,23
Li-A(10)	9	124,15	1,01	2,24	1,62	-5,97	-2,61	-4,29
Li-A(10)	10	135,10	1,21	1,55	1,38	-7,15	-1,81	-4,48
Li-B(10)	6	63,69	0,66	3,14	1,90	-3,90	-3,67	-3,78
Li-B(10)	7	84,54	0,64	2,97	1,80	-3,80	-3,46	-3,63
Li-B(10)	8	99,84	1,03	3,39	2,21	-6,07	-3,95	-5,01
Li-B(10)	9	113,52	0,94	2,69	1,82	-5,58	-3,14	-4,36
Li-B(10)	10	123,42	0,91	0,48	0,70	-5,41	-0,56	-2,99
Murphy(10)	6	51,24	0,57	3,93	2,25	-3,35	-4,59	-3,97
Murphy(10)	7	74,06	0,67	4,09	2,38	-3,99	-4,77	-4,38
Murphy(10)	8	90,06	1,23	4,79	3,01	-7,28	-5,59	-6,43
Murphy(10)	9	103,29	1,13	3,19	2,16	-6,71	-3,72	-5,21
Murphy(10)	10	118,59	1,02	2,89	1,96	-6,05	-3,37	-4,71
SE-B(10)	6	49,35	0,94	3,76	2,35	-5,57	-4,39	-4,98
SE-B(10)	7	73,33	**1,35**	**4,98**	**3,16**	**-7,98**	**-5,81**	**-6,89**
SE-B(10)	8	89,51	1,12	4,28	2,70	-6,61	-4,99	-5,80
SE-B(10)	9	102,09	1,34	2,70	2,02	-7,94	-3,14	-5,54
SE-B(10)	10	116,11	1,12	2,37	1,74	-6,64	-2,76	-4,70
SE-V(10)	6	37,92	0,50	3,68	2,09	-2,97	-4,29	-3,63
SE-V(10)	7	61,58	1,00	4,25	2,63	-5,91	-4,96	-5,43
SE-V(10)	8	77,77	1,15	3,74	2,44	-6,80	-4,36	-5,58
SE-V(10)	9	92,03	1,12	2,81	1,97	-6,66	-3,28	-4,97
SE-V(10)	10	103,56	1,20	3,17	2,19	-7,12	-3,70	-5,41
Solis-D(10)	6	83,67	0,84	4,17	2,50	-4,97	-4,86	-4,92
Solis-D(10)	7	104,56	0,96	3,32	2,14	-5,65	-3,87	-4,76
Solis-D(10)	8	122,56	1,30	2,22	1,76	-7,70	-2,59	-5,14
Solis-D(10)	9	137,40	1,22	1,18	1,20	-7,21	-1,37	-4,29
Solis-D(10)	10	144,09	1,20	2,20	1,70	-7,12	-2,56	-4,84
Solis-G(10)	6	-57,23	-7,49	-17,83	-12,66	44,35	20,79	32,57
Solis-G(10)	7	14,62	-1,28	-3,07	-2,17	7,59	3,58	5,58
Solis-G(10)	8	59,45	0,27	1,76	1,01	-1,57	-2,06	-1,81
Solis-G(10)	9	89,64	0,22	2,08	1,15	-1,27	-2,42	-1,85
Solis-G(10)	10	108,24	0,28	1,29	0,78	-1,65	-1,50	-1,58
SE-B(14)	6	84,42	1,22	4,68	2,95	-7,19	-5,46	-6,33
SE-B(14)	7	103,52	1,01	3,41	2,21	-5,98	-3,98	-4,98
SE-B(14)	8	121,80	1,15	3,09	2,12	-6,83	-3,60	-5,21
SE-B(14)	9	136,18	0,92	1,17	1,04	-5,43	-1,36	-3,39
SE-B(14)	10	146,87	0,84	-0,11	0,36	-4,96	0,13	-2,42
		Melhores	1,35	4,98	3,16	-7,98	-5,81	-6,89

Tabela 4.4: Resultados ao alterar os métodos para computação de matriz de distâncias e para construção de árvore guia para PAM e NJ, respectivamente.

Normalização	Δt	ΔSP	ΔTC	ΔA	ΔSPe	ΔTCe	ΔAe
nenhuma	18,94	-4,70	-15,81	-10,25	27,80	18,44	23,12
0,00 - 1,00	-78,67	-12,09	-22,35	-17,22	71,55	26,06	48,81
0,10 - 1,00	-75,88	-10,46	-18,45	-14,46	61,91	21,52	41,71
0,20 - 1,00	-69,49	-8,66	-14,09	-11,38	51,27	16,43	33,85
0,30 - 1,00	-46,86	-4,72	-8,11	-6,42	27,94	9,46	18,70
0,40 - 1,00	91,52	0,36	-0,11	0,13	-2,12	0,13	-1,00
0,50 - 1,00	168,60	0,97	3,74	2,35	-5,73	-4,36	-5,04
0,60 - 1,00	168,74	1,03	3,88	2,45	-6,07	-4,52	-5,30
0,65 - 1,00	168,69	**1,05**	3,97	2,51	**-6,19**	-4.63	-5.41
0,70 - 1,00	168,59	1,04	**4,16**	**2,60**	-6.13	**-4,85**	**-5,49**
0,75 - 1,00	168,64	1,01	4,10	2,56	-6,00	-4,78	-5,39
0,80 - 1,00	168,59	1,04	4,09	2,56	-6,14	-4,77	-5,45
0,85 - 1,00	168,70	0,99	3,91	2,45	-5,88	-4,56	-5,22
0,90 - 1,00	168,27	1,04	4,12	2,58	-6,14	-4,81	-5,47
Melhores		1,05	4,16	2,60	-6,19	-4,85	-5,49

Tabela 4.5: Resultados ao alterar os métodos para computação de matriz de distâncias e para construção de árvore guia para PAM e UPGMA, respectivamente.

Normalização	Δt	ΔSP	ΔTC	ΔA	ΔSPe	ΔTCe	ΔAe
nenhuma	91,17	-1,27	-4,01	-2,64	7,53	4,68	6,10
0,00 - 1,00	-70,12	-6,13	-9,87	-8,00	36,26	11,51	23,89
0,10 - 1,00	-43,43	-4,13	-4,83	-4,48	24,46	5,63	15,05
0,20 - 1,00	168,52	0,85	**3,27**	2,06	-5,05	**-3,81**	-4,43
0,30 - 1,00	168,47	0,99	**3,27**	2,13	-5,88	**-3,81**	-4,84
0,40 - 1,00	144,76	**1,00**	**3,27**	**2,14**	**-5,94**	**-3,81**	**-4,88**
0,50 - 1,00	144,80	**1,00**	**3,27**	**2,14**	**-5,94**	**-3,81**	**-4,88**
0,60 - 1,00	145,04	**1,00**	**3,27**	**2,14**	**-5,94**	**-3,81**	**-4,88**
0,65 - 1,00	144,77	**1,00**	**3,27**	**2,14**	**-5,94**	**-3,81**	**-4,88**
0,70 - 1,00	144,90	**1,00**	**3,27**	**2,14**	**-5,94**	**-3,81**	**-4,88**
0,75 - 1,00	144,79	**1,00**	**3,27**	**2,14**	**-5,94**	**-3,81**	**-4,88**
0,80 - 1,00	144,82	**1,00**	**3,27**	**2,14**	**-5,94**	**-3,81**	**-4,88**
0,85 - 1,00	168,45	**1,00**	**3,27**	**2,14**	**-5,94**	**-3,81**	**-4,88**
0,90 - 1,00	144,85	**1,00**	3,26	2,13	**-5,94**	-3,80	-4,87
Melhores		1,00	3,27	2,14	-5,94	-3,81	-4,88

Tabela 4.6: Tempo de execução total, para as 218 entradas de sequências completas do BAliBASE, e pontuações SP e TC médias para as melhores variações do MUMMALS, por experimento realizado.

Alinhador	**Tempo (s)**	**SP**	**TC**
MUMMALS Original	50.594	85,54	53,83
MUMMALS $k = 8$	81.017	86,27	55,70
MUMMALS SE-B(10) $k = 7$	87.692	86,70	56,52
MUMMALS PAM+NJ 0,7-1,0	135.890	86,43	56,07

6,19% de redução de erros ao avaliar pela pontuação SP e 4,85% menos erros ao avaliar pela pontuação TC. Na Tabela 4.6 são litadas as melhores variações no MUMMALS para cada um dos três experimentos.

Assim como a qualidade dos alinhamentos gerados, o tempo de execução é extremamente afetado de acordo com o valor de k, alfabeto comprimido, método para computação de matriz de distâncias, procedimento para construção de árvore guia e faixa de normalização. O tempo de execução variou de $3.311s$ a $135.967s$. O MUMMALS original consome $50.594s$ para computar os resultados com uma pontuação SP média de 85,54 e uma pontuação TC média de 53,83. O melhor alinhador, desenvolvido na avaliação apresentada neste capítulo, requer $87.692s$ para realizar os mesmos 218 alinhamentos ou, em outras palavras, ele é 73,33% mais lento que o MUMMALS original. Entretanto, ele reduz erros em 7,98% na avaliação pela pontuação SP e obteve 5,81% menos erros numa perspectiva de pontuação TC.

Os resultados permitem concluir que um método de contagem k-mer pode ser preciso o suficiente para computar uma boa matriz de distâncias no contexto do alinhamento progressivo. É importante notar que o uso de alinhamento de pares através de um modelo oculto de Markov complexo, como aquele empregado pelo MUMMALS, contribuiu efetivamente para alcançar esses resultados. Seu HMM sofisticado e a medida de consistência probabilística já mostraram alta precisão para alinhamento de sequências com baixa similaridade.

Capítulo 5

Alinhamento múltiplo usando a abordagem iterativa

Alinhadores iterativos caracterizam-se pela dependência de algoritmos capazes de produzir alinhamentos iniciais. Uma vez construídos tais alinhamentos, cabe a eles a tarefa de refiná-los através de ciclos até que não seja mais possível incrementar a qualidade dos alinhamentos ou é realizado um número fixo de ciclos.

Os métodos iterativos mais simples são os não estocásticos. Esses basicamente realizam realinhamentos. Remove uma ou mais sequências do alinhamento e, em seguida, realinha elas ao restante das sequências. Há ainda os métodos iterativos estocásticos, que incluem HMM [143], *simmulated annealing* [139] e algoritmos genéticos [45, 49, 182, 269]. Para maiores informações sobre a abordagem iterativa, consulte a Seção 2.3.2. Na Seção 2.3.4, há informações sobre as abordagens baseadas em consenso, modelos e blocos, que foram utilizadas nas implementações dos alinhadores iterativos.

Neste capítulo, as implementações e testes realizados no contexto de alinhamento múltiplo usando a abordagem iterativa são detalhados. Na Seção 5.1 são apresentados os trabalhos relativos a métodos não estocásticos. Já os métodos estocásticos são apresentados na Seção 5.2.

5.1 Alinhadores iterativos não estocásticos

No contexto de alinhamento iterativo não estocástico foram implementados módulos de refinamento de alinhamento múltiplo. No primeiro deles, denominado R11, a cada iteração, divide-se aleatoriamente as sequências que compõem o alinhamento em dois grupos. Em seguida o alinhamento é dividido em dois de acordo com os grupos. Em cada um dos alinhamentos, removem-se as colunas compostas apenas por *gaps* e, finalmente, os alinhamentos são reagrupados, usando alinhamento de perfis semi-global com função afim para

penalidade de *gaps*. No agrupamento usa-se os parâmetros definidos pelos testes com alinhamento progressivo apresentados no Capítulo 3, ou seja, BLOSUM45 com $gop = -10$ e $gep = -1$. Ao final do ciclo é selecionado o alinhamento, seja o original ou o refinado, com a maior pontuação. Quando são feitas 5 tentativas consecutivas de melhorar a pontuação sem sucesso, a iteração é interrompida.

Uma vez definido o primeiro módulo de refinamento, passou-se, então, a avaliar dois tipos de pontuação para quantificar a qualidade dos alinhamentos gerados. Foi avaliada soma dos pares com função afim para penalidade de *gaps* e soma dos pares com função linear para penalidade de *gaps*. A primeira pontuação foi denominada $S1$ e a segunda $S2$. Para avaliar os parâmetros a utilizar, uma bateria de testes foi realizada. Para a função afim, utilizou-se como entrada o alinhamento gerado pelo alinhador 53b para a entrada BB12020.tfa do BAliBASE. Variou-se BLOSUM de 45 a 80 (inclusive e com intervalo de cinco entre uma matriz e outra), assim como *gop* e *gep* de -1 a -20. Para cada conjunto de parâmetros de entrada para o módulo de refino, três execuções foram realizadas e tomada a média destas para se chegar a conclusões. Os resultados apontaram que BLOSUM80 com $gop = -11$ e $gep = -1$ é um bom conjunto a utilizar para $S1$. De forma análoga chegou-se a conclusão de que BLOSUM80 com $gap = -1$ é um bom conjunto de parâmetros para $S2$.

Os conjuntos RVS1, RVS2 e RVS3 foram utilizados para avaliar as pontuações $S1$ e $S2$. Este terceiro conjunto foi construído de forma semelhante aos dois primeiros, porém nele foram adicionadas 5 entradas do conjunto RV11 e 5 do conjunto RV12. A Tabela 5.1 apresenta a composição de cada um desses conjuntos.

Tabela 5.1: Subconjuntos de referência extraídos do BAliBASE.

Conjunto de Teste	**RVS1**	**RVS2**	**RVS3**
Conjuntos do BAliBASE	BB11001	BB11025	BB11013
	BB12020	BB12021	BB11021
	BB20020	BB20001	BB11022
	BB30017	BB30006	BB11029
	BB40032	BB40010	BB11035
	BB50004	BB50002	BB12003
			BB12006
			BB12009
			BB12025
			BB12040

Além de testar $S1$ e $S2$ isoladamente, as funções $S1eS2$ e $S1ouS2$ também foram avaliadas. Na Tabela 5.2 são apresentados os resultados. Pode-se observar que a função $S2$ obteve melhores resultados, com maiores médias e desvios menores em ambas as pon-

tuações calculadas pelo BAliBASE. Dessa forma, foi a função utilizada para pontuar os alinhamentos.

Tabela 5.2: Avaliação do sistema de pontuação a utilizar para avaliar os alinhamentos ao longo do refino.

Pontuação	**SP**		**TC**	
	Média	**Desvio**	**Média**	**Desvio**
$S1$	66,80	34,70	49,30	38,80
$S2$	71,60	31,90	54,70	37,10
$S1 ou S2$	69,10	32,70	51,80	38,40
$S1 e S2$	67,90	33,20	49,20	40,10

Uma variação de R11, denominada R12, foi implementada. Nesta, ao final de cada ciclo, o alinhamento corrente sempre é atualizado, independentemente de pontuar mais ou menos. O alinhamento de maior pontuação ao longo da execução do módulo, entretanto, é armazenado e dado como saída ao final das iterações.

Uma série de testes, para ambos os refinadores, foi realizada utilizando os cinco melhores (053, 053b, 077b, 125 e 137) e os cinco piores (321b, 323, 323b, 327, 327b) alinhadores progressivos, que podem ser observados nas tabelas 3.12 e 3.14. Avaliou-se as pontuações originais e após cada procedimento de refinamento para os conjuntos de referência RVS1, RVS2 e RVS3. Na Tabela 5.3 são apresentadas as pontuações obtidas pelos alinhadores progressivos originais. Nas tabelas 5.4 e 5.5 são apresentadas as pontuações para as versões que incluem os módulos de refinamento R11 e R12, respectivamente.

Na Tabela 5.6 é apresentado um resumo dos resultados obtidos pelos refinadores R11 e R12. Nas duas últimas colunas pode-se observar o quanto os refinadores melhoraram (ou pioraram) os resultados em relação aos alinhadores progressivos originais. Por exemplo, para o alinhador 053, o refinador R11 alcançou uma melhoria de $1,87\%$. Já para o alinhador 137, o refinador R12 piorou o resultado em $0,38\%$ quando comparado a pontuação original. Em média, R11 melhorou os resultados em $5,67\%$ e R12 em $5,25\%$.

Tabela 5.3: Resultados dos testes para os alinhadores progressivos originais. Na primeira coluna são listados os alinhadores. Nas três seguintes são detalhadas suas respectivas pontuações SP médias para RVS1, RVS2 e RVS3. Já na última coluna são apresentadas as médias das pontuações para os três conjuntos de referência.

Alinhador	**RVS1**	**RVS2**	**RVS3**	**Média**
053	91,15	53,35	64,27	68,62
053b	84,12	44,83	55,23	60,27
077b	84,27	47,52	55,67	61,25
125	91,93	52,58	64,39	68,68
137	91,05	50,98	65,18	68,36
321b	47,47	42,63	46,00	45,48
323	54,27	24,60	41,42	40,34
323b	47,57	27,55	42,52	39,81
327	68,35	44,17	57,27	56,72
327b	52,62	35,25	47,24	45,44
Média	71,28	42,35	53,92	55,50

Tabela 5.4: Resultados dos testes para os alinhadores progressivos com o uso do módulo de refinamento R11. Na primeira coluna são listados os alinhadores. Nas três seguintes são detalhadas suas respectivas pontuações SP médias para RVS1, RVS2 e RVS3. Já na última coluna são apresentadas as médias das pontuações para os três conjuntos de referência.

Alinhador	**RVS1**	**RVS2**	**RVS3**	**Média**
053	90,32	53,92	67,25	69,90
053b	84,73	44,38	63,49	64,07
077b	84,95	47,52	60,99	63,85
125	90,60	50,78	65,99	68,55
137	90,82	50,98	64,70	68,08
321b	70,53	47,92	60,94	60,00
323	54,27	24,23	40,33	39,74
323b	49,82	26,83	42,52	40,23
327	69,28	43,77	59,48	57,87
327b	60,07	34,75	54,38	50,58
Média	74,54	42,51	58,01	58,29

Tabela 5.5: Resultados dos testes para os alinhadores progressivos com o uso do módulo de refino R12. Na primeira coluna são listados os alinhadores. Nas três seguintes são detalhadas suas respectivas pontuações SP médias para RVS1, RVS2 e RVS3. Já na última coluna são apresentadas as médias das pontuações para os três conjuntos de referência.

Alinhador	**RVS1**	**RVS2**	**RVS3**	**Média**
053	89,17	53,03	66,15	68,85
053b	83,35	44,53	62,67	63,36
077b	82,57	47,17	65,28	65,05
125	90,88	51,02	67,20	69,25
137	90,18	50,98	65,13	68,10
321b	68,30	43,52	60,17	57,85
323	54,27	24,78	43,88	41,50
323b	47,57	26,72	42,52	39,59
327	69,05	43,42	60,39	58,12
327b	59,83	33,83	52,11	49,23
Média	73,52	41,90	58,55	58,09

Tabela 5.6: Resultados obtidos pelos refinadores iterativos R11 e R12. Na primeira coluna há a indicação do alinhador progressivo utilizado no teste e na segunda a pontuação SP média original para RVS1, RVS2 e RVS3. Nas colunas seguintes são listados os valores obtidos depois de aplicados os refinadores R11 e R12, respectivamente. Nas duas últimas colunas compara-se os novos valores em relação ao original.

Alinhador	**PO**	**PR11**	**PR12**	**Efeito R11**	**Efeito R12**
053	68,62	69,90	68,85	1,87%	0,34%
053b	60,27	64,07	63,36	6,30%	5,13%
077b	61,25	63,85	65,05	4,24%	6,20%
125	68,68	68,55	69,25	-0,19%	0,83%
137	68,36	68,08	68,10	-0,41%	-0,38%
321b	45,48	60,00	57,85	31,93%	27,20%
323	40,34	39,74	41,50	-1,49%	2,88%
323b	39,81	40,23	39,59	1,06%	-0,55%
327	56,72	57,87	58,12	2,03%	2,47%
327b	45,44	50,58	49,23	11,31%	8,34%
			Média	5,67%	5,25%

5.2 Alinhadores iterativos estocásticos

No contexto de alinhamento iterativo estocástico, foram feitas implementações e testes com algoritmos genéticos [117], uma categoria de algoritmo evolutivo. GAs são inspirados na Teoria da Seleção Natural de Darwin [54, 80]. Nestes, uma população de indivíduos (possíveis soluções) evolui por gerações. A cada geração, indivíduos são gerados por eventos de mutação ou cruzamento e, então, se seleciona a população para a próxima geração de acordo com o grau de adaptação dos indivíduos. Esse grau de adaptação é dado por uma função, que na prática é a função objetivo que se deve otimizar. A iteração é encerrada quando há uma estabilização no grau de adaptação da população ao longo de gerações ou pode ser definido um número fixo de gerações. Nesta implementação adotou-se um número fixo de gerações. O procedimento básico adotado é apresentado no Algoritmo 5.

Algoritmo 5: Algoritmo Genético utilizado na implementação do alinhador iterativo.

```
Input: Seqs, MaxPopulationSize, Generations
Output: The best alignment
1 population ← createInitialPopulation(Seqs, MaxPopulationSize)
2 for n ← 1 to Generations do
3 |   breedingPopulation ← selectForBreeding(population)
4 |   population ← population ∪ offspring(breedingPopulation)
5 |   population ← select(population, MaxPopulationSize)
6 return bestAlignment(population)
```

Observe que há três parâmetros para esse algoritmo. O primeiro (`Seqs`) é a lista de sequências a alinhar. O segundo deles (`MaxPopulationSize`) determina o tamanho máximo da população de uma dada geração. Já o terceiro (`Generations`) determina o número total de gerações.

O algoritmo inicia pela construção de uma população inicial. Em seguida, há um laço, onde a cada ciclo indivíduos são selecionados, dentre aqueles da população corrente, para a geração de novos indivíduos através de operadores de cruzamento e mutação. Esses novos indivíduos, gerados pela execução dos operadores, são adicionados à população corrente. O ciclo é concluído com a seleção dos indivíduos que sobreviverão para a próxima geração. Esse ciclo se repete de acordo com o parâmetro `Generations`, que determina o número de ciclos. Ao final da iteração é retornado o indivíduo mais adaptado.

Nas seções que seguem, os procedimentos utilizados em cada passo do algoritmo genético são detalhados. A geração da população inicial é descrita na Seção 5.2.1. Nas seções 5.2.2 e 5.2.3 são apresentadas, respectivamente, a forma como os indivíduos da po-

pulação corrente são selecionados para a reprodução (aplicação dos operadores de mutação e cruzamento) e os operadores utilizados. O procedimento para a seleção dos indivíduos que sobreviverão para a próxima geração é detalhado na Seção 5.2.4.

Na Seção 5.2.5 são apresentados os resultados de testes iniciais com esse alinhador, onde é definida a função de aptidão que será utilizada e avaliada a evolução da população ao longo das gerações. A partir dos resultados obtidos, foram definidos novos testes. No primeiro deles, detalhado na Seção 5.2.6, são avaliados os melhores parâmetros para penalização de *gaps*. Em uma outra etapa, apresentada na Seção 5.2.7, tenta-se melhorar a qualidade dos alinhamentos através de uma função de aptidão que considera alinhamentos de estruturas secundárias. Avaliou-se ainda métodos alternativos para geração da população inicial, que é apresentado na Seção 5.2.8, e um operador baseado em metamétodo consenso, que é apresentado na Seção 5.2.9.

5.2.1 Geração da população inicial

A primeira abordagem adotada para a criação da população inicial foi a utilização de alinhamentos de pares de sequências, considerando tanto alinhamentos globais quanto semi-globais. Tais alinhamentos são gerados, respectivamente, pelo algoritmo de Needleman e Wunsch [174] e sua variação para alinhamento semi-global [219]. A geração de cada indivíduo inicia pela seleção arbitrária de um par de sequências e um tipo de alinhamento. Em seguida, a cada ciclo, um par de sequências, onde uma dessas necessariamente já pertence ao alinhamento e a outra não, é arbitrariamente selecionado. É sorteado o tipo de alinhamento e então a sequência é inserida ao alinhamento utilizando como âncora seu par, que já pertence ao alinhamento. Respeita-se o princípio "uma vez *gap*, sempre *gap*". Esse processo segue até que todas as sequências sejam adicionadas ao alinhamento, quando o indivíduo é incluído à população. São gerados x indivíduos utilizando esse método, onde x é o tamanho da população definido pelo parâmetro `MaxPopulationSize`.

A segunda abordagem é semelhante à primeira. A diferença é que a âncora dessa vez é definida apenas uma vez. Todos os alinhamentos de pares são realizados utilizando a mesma sequência como âncora. Nessa abordagem utilizou-se apenas alinhamentos globais. São gerados por esse método $2 \times |sequences|$ indivíduos, onde *sequences* é o conjunto de sequências de entrada.

5.2.2 Seleção dos indivíduos para reprodução

A cada geração, ciclo do algoritmo evolutivo, indivíduos são selecionados para reprodução. Nesta implementação foi utilizado o método da roleta. Este, que é um procedimento clássico, objetiva dar oportunidade para todos os indivíduos. Aqueles mais adaptados, porém, têm maior probabilidade de serem selecionados.

No método da roleta cria-se uma roleta desigualmente dividida. Cada indivíduo recebe uma área proporcional a sua aptidão em relação a aptidão geral da população. Isso torna um indivíduo mais adaptado propenso a ser selecionado, porém não exclui a possibilidade de um indivíduo menos adaptado também o ser.

Metade da população é selecionada para reprodução pelo método da roleta, tal como descrito anteriormente. Desses (pré-selecionados), cada indivíduo tem 20% de chance de ser selecionado para aplicação de operadores de mutação e 20% para cruzamento. Observe que um mesmo indivíduo pode ser selecionado para mutação e para cruzamento. Observe também que indivíduos selecionados para reprodução podem não ser selecionados para mutação ou cruzamento. Para cada indivíduo selecionado para mutação é sorteado um operador de mutação, que é aplicado a ele. No caso dos x indivíduos selecionados para cruzamento, são realizados $x/2$ cruzamentos. Para cada par é sorteado um operador de cruzamento, que é aplicado a ele.

5.2.3 Operadores

Um operador de mutação é caracterizado por realizar modificação em um indivíduo, gerando um novo indivíduo. Essa modificação, no caso de MSA, pode ser a inserção ou a remoção de *gaps*, assim como o deslocamento destes. Três foram os operadores de mutação implementados e utilizados. Estes são listados e descritos a seguir.

- ***Simple Mutation***: Este operador seleciona um valor para $swapCount$ entre 2 e $(2 + |sequences|/2)$. Em seguida realiza uma iteração, $swapCount$ vezes. Para cada passo, seleciona arbitrariamente uma sequência e em seguida uma posição nesta. Busca o primeiro bloco de *gaps* e inverte o primeiro resíduo depois do bloco com o último *gap* do bloco. Caso não encontre um bloco de *gaps* na posição ou depois dela, faz até quatro novas tentativas de realizar a modificação. A seguir, um exemplo de modificação realizada por esse operador.

```
SAPANAVAADNATAIALKYNQDATKSERVAAARPGLPPEEQHCADCQFMQADAAGATDEWKGC--QLFPGKLINVNGWCASWTLKAG
EDLPHVDAATNPIAQSLHYIEDANASERNPVTKTELPGSEQFCHNCSFIQADSGA----WRPC--TLYPGYTVSEDGWCLSWAHKTA
AAPLVAETDAN--AKSLGYVADTTKADK---TKYPKHTKDQSCSTCALYQ----GKTAPQGAC--PLFAGKEVVAKGWCSAWA-KKA
MERLSED---DPAAQALEYRHDAS------SVQHPAYEEGQTCLNC-LLYTDASAQ--DWGPC--SVFPGKLVSANGWCTAWVAR--
QDLPPLDPSAE-QAQALNYVKDTA-----EAADHPAHQEGEQCDNCMFF----QADSQGCQL-----FPQNSVEPAGWCQSWTAQN-
------EPRAE-DGHAHDYVNEAA-----DASGHPRYQEGQLCENCAFW---GEAVQDGWGRCTHPDFDEVLVKAEGWCSVYAPAS-

                                          **
SAPANAVAADNATAIALKYNQDATKSERVAAARPGLPPEEQHCADCQFMQADAAGATDEWKGC--QLFPGKLINVNGWCASWTLKAG
EDLPHVDAATNPIAQSLHYIEDANASERNPVTKTELPGSEQFCHNCSFIQADSGA----WRPC--TLYPGYTVSEDGWCLSWAHKTA
AAPLVAETDAN--AKSLGYVADTTKADK---TKYPKHTKDQSCSTCALYQ----GKTAPQGAC--PLFAGKEVVAKGWCSAWA-KKA
MERLSED---DPAAQALEYRHDAS------SVQHPAYEEGQTCLNC-LLYTDASAQ--DWGPC--SVFPGKLVSANGWCTAWVAR--
QDLPPLDPSAE-QAQALNYVKDTA-----EAADHPAHQEGEQCDNCMFF--Q-QADSQGCQL-----FPQNSVEPAGWCQSWTAQN-***
------EPRAE-DGHAHDYVNEAA-----DASGHPRYQEGQLCENCAFW---GEAVQDGWGRCTHPDFDEVLVKAEGWCSVYAPAS-
```

- ***Shift Gap Block Mutation***: Este operador seleciona arbitrariamente um bloco de *gaps* e desloca parte dele para a direita. O tamanho do deslocamento é selecionado aleatoriamente. A seguir, um exemplo de modificação realizada por esse operador.

```
SAPANAVAADNATAIALKYNQDATKSERVAAARPGLPPEEQHCADCQFMQADAAGATDEWKGC--QLFPGKLINVNGWCASWTLKAG
EDLPHVDAATNPIAQSLHYIEDANASERNPVTKTELPGSEQFCHNCSFIQADSGA----WRPC--TLYPGYTVSEDGWCLSWAHKTA
AAPLVAETDAN--AKSLGYVADTTKADK---TKYPKHTKDQSCSTCALYQ----GKTAPQGAC--PLFAGKEVVAKGWCSAWA-KKA
MERLSED---DPAAQALEYRHDAS------SVQHPAYEEGQTCLNC-LLYTDASAQ--DWGPC--SVFPGKLVSANGWCTAWVAR--
QDLPPLDPSAE-QAQALNYVKDTA-----EAADHPAHQEGEQCDNCMFF----QADSQGCQL-----FPQNSVEPAGWCQSWTAQN-
------EPRAE-DGHAHDYVNEAA-----DASGHPRYQEGQLCENCAFW---GEAVQDGWGRCTHPDFDEVLVKAEGWCSVYAPAS-

SAPANAVAADNATAIALKYNQDATKSERVAAARPGLPPEEQHCADCQFMQADAAGATDEWKGC--QLFPGKLINVNGWCASWTLKAG
EDLPHVDAATNPIAQSLHYIEDANASERNPVTKTELPGSEQFCHNCSFIQADSGA----WRPC--TLYPGYTVSEDGWCLSWAHKTA
AAPLVAETDAN--AKSLGYVADTTKADK---TKYPKHTKDQSCSTCALYQ----GKTAPQGAC--PLFAGKEVVAKGWCSAWA-KKA
MERLSED---DPAAQALEYRHDAS------SVQHPAYEEGQTCLNC-LLYTDASAQ--DWGPC--SVFPGKLVSANGWCTAWVAR--
QDLPPLDPSAE-QAQALNYVKDTA-----EAADHPAHQEGEQCDNCMFF----QADSQGCQL-----FPQNSVEPAGWCQSWTAQN-
------EPRAE-DGHAHDYVNEAA--DASGHPRYQEGQLC---ENCAFW---GEAVQDGWGRCTHPDFDEVLVKAEGWCSVYAPAS-***
```

- ***Change Gap Block Mutation***: Este operador seleciona arbitrariamente um bloco contínuo de *gaps* e o desloca uma posição para a esquerda ou para a direita. A seguir, um exemplo de modificação realizada por esse operador.

```
SAPANAVAADNATAIALKYNQDATKSERVAAARPGLPPEEQHCADCQFMQADAAGATDEWKGC--QLFPGKLINVNGWCASWTLKAG
EDLPHVDAATNPIAQSLHYIEDANASERNPVTKTELPGSEQFCHNCSFIQADSGA----WRPC--TLYPGYTVSEDGWCLSWAHKTA
AAPLVAETDAN--AKSLGYVADTTKADK---TKYPKHTKDQSCSTCALYQ----GKTAPQGAC--PLFAGKEVVAKGWCSAWA-KKA
MERLSED---DPAAQALEYRHDAS------SVQHPAYEEGQTCLNC-LLYTDASAQ--DWGPC--SVFPGKLVSANGWCTAWVAR--
QDLPPLDPSAE-QAQALNYVKDTA-----EAADHPAHQEGEQCDNCMFF----QADSQGCQL-----FPQNSVEPAGWCQSWTAQN-
------EPRAE-DGHAHDYVNEAA-----DASGHPRYQEGQLCENCAFW---GEAVQDGWGRCTHPDFDEVLVKAEGWCSVYAPAS-

SAPANAVAADNATAIALKYNQDATKSERVAAARPGLPPEEQHCADCQFMQADAAGATDEWKGC--QLFPGKLINVNGWCASWTLKAG
EDLPHVDAATNPIAQSLHYIEDANASERNPVTKTELPGSEQFCHNCSFIQADSG----AWRPC--TLYPGYTVSEDGWCLSWAHKTA***
AAPLVAETDAN--AKSLGYVADTTKADK---TKYPKHTKDQSCSTCALYQ----GKTAPQGAC--PLFAGKEVVAKGWCSAWA-KKA
MERLSED---DPAAQALEYRHDAS------SVQHPAYEEGQTCLNC-LLYTDASAQ--DWGPC--SVFPGKLVSANGWCTAWVAR--
QDLPPLDPSAE-QAQALNYVKDTA-----EAADHPAHQEGEQCDNCMFF----QADSQGCQL-----FPQNSVEPAGWCQSWTAQN-
------EPRAE-DGHAHDYVNEAA-----DASGHPRYQEGQLCENCAFW---GEAVQDGWGRCTHPDFDEVLVKAEGWCSVYAPAS-
```

Um operador de cruzamento é caracterizado pela combinação de dois ou mais indivíduos, gerando novo(s) indivíduo(s). Dois foram os operadores de cruzamento implementados e utilizados. Estes são listados e descritos a seguir.

- ***Single Point Crossover***: Este operador recebe como entrada dois indivíduos. Ele executa um corte vertical em um dos alinhamentos pai e um corte compatível com esse no outro. A cada sub-alinhamento são eventualmente adicionados *gaps* nas extremidades para manter a consistência do alinhamento. Em seguida são gerados dois novos alinhamentos. O primeiro pela concatenação do primeiro sub-alinhamento do indivíduo 1 com o segundo sub-alinhamento do indivíduo 2. O segundo alinhamento é gerado pela concatenação dos sub-alinhamentos restantes. A seguir, um exemplo com dois alinhamentos e um corte vertical compatível entre eles.

```
SAPANAVAADNATAIALKYNQDATKSERVAAARPGLPPEEQHCADCQFMQADAAGATDEWKGC--     QLFPGKLINVNGWCASWTLKAG
EDLPHVDAATNPIAQSLHYIEDANASERNPVTKTELPGSEQFCHNCSFIQADSGA----WRPC--     TLYPGYTVSEDGWCLSWAHKTA
AAPLVAETDAN--AKSLGYVADTTKADK---TKYPKHTKDQSCSTCALYQ----GKTAPQGAC--     PLFAGKEVVAKGWCSAWA-KKA
MERLSED---DPAAQALEYRHDAS------SVQHPAYEEGQTCLNC-LLYTDASAQ--DWGPC--     SVFPGKLVSANGWCTAWVAR--
QDLPPLDPSAE-QAQALNYVKDTA-----EAADHPAHQEGEQCDNCMFF----QADSQGCQL---     --FPQNSVEPAGWCQSWTAQN-
------EPRAE-DGHAHDYVNEAA-----DASGHPRYQEGQLCENCAFW---GEAVQDGWGRCTH     PDFDEVLVKAEGWCSVYAPAS-

SAPANAVAADNATAIALKYNQDATKSERVAAARPGLPPEEQHCADCQFM-QADAAGATDEWKGC       Q-L-FPGKLINVNGWCASWTLKAG
EDLPHVDAATNPIAQSLHYIEDANASERNPVTKTELPGSEQFCHNCSFI-QADSGA----WRPC       T-L-YPGYTVSEDGWCLSWAHKTA
AAPLVAETDAN--AKSLGYVADTTKADK---TKYPKHTKDQSCSTCALY-Q----GKTAPQGAC       P-L-FAGKEVVAKGWCSAWA-KKA
MERLSED---DPAAQALEYRHDAS------SVQHPAYEEGQTCLNC-LL-YTDASAQ--DWGPC       S-V-FPGKLVSANGWCTAWVAR--
QDLPPLDPSAEQ-AQALNYVKDTA--E---AADHPAHQEGEQCDNCMFF-QADSQG----CQL-       ----FPQNSVEPAGWCQSWTAQN-
------EPRAED-GHAHDYVNEAA--D---ASGHPRYQEGQLCENCAFWGEAVQDG----WGRCTH       PDFDEVLVKAEGWCSVYAPAS-
```

- ***Sequence Similarity Crossover***: Este operador escolhe um nó de forma arbitrária em uma árvore filogenética, construída a partir das distâncias de pares entre as sequências de entrada utilizando UPGMA. Em seguida, seleciona arbitrariamente um nó e divide os alinhamentos pais baseado nas sequências que se encontram abaixo (na sub-árvore cuja raiz é o nó selecionado) e acima deste (restante). As sequências são novamente combinadas em um único alinhamento pelo alinhamento dos consensos de cada sub-alinhamento.

5.2.4 Seleção dos indivíduos para a próxima geração

A última etapa do ciclo consiste no corte. Esta tem por objetivo promover a longevidade dos indivíduos mais adaptados (mais fortes), assim como manter o tamanho da população constante ao longo das gerações por uma questão de performance. Observe que, quanto maior for a população, maior será o tempo demandado para a execução de cada ciclo. O corte numa população consiste em eliminar os indivíduos menos adaptados até que se atinja um determinado tamanho para a população, em outras palavras, seleciona os x indivíduos mais adaptados. Na implementação feita, atribuiu-se a x o valor do parâmetro `MaxPopulationSize`.

5.2.5 Testes iniciais

Uma aplicação, chamada ALGAe (do inglês, *ALigning by Genetic Algorithm environment*), foi desenvolvida. Consiste de um ambiente parametrizável para a execução de algoritmos genéticos voltados para MSA. A aplicação foi implementada em Java e permite que, através da implementação de determinadas interfaces e do uso de reflexão, seja configurada uma série de parâmetros para o GA. Ela permite a definição de parâmetros como algoritmo para seleção dos indivíduos para reprodução e geração da população inicial, função objetivo a utilizar para determinar o grau de aptidão de um indivíduo, operadores

a utilizar e suas probabilidades de execução, assim como tamanho da população e número de gerações.

Ainda nessa seção são apresentados testes que levaram a decisão sobre a função de aptidão a utilizar, assim como foi feita uma avaliação da evolução da população ao longo das gerações.

Definição da função de aptidão

A função de aptidão é responsável por determinar o grau de adaptação de um indivíduo. Ela é utilizada para a seleção dos indivíduos para reprodução, assim como para determinar os indivíduos que irão sobreviver para a próxima geração. A seguir são apresentadas duas funções de aptidão comumente utilizadas para MSA.

- **Soma dos Pares (SP)**: esta função considera, baseada em uma função de pontuação, a soma de cada par de resíduos em uma mesma coluna. Para um alinhamento de m sequências com tamanho n e c_{ik} representando a k-ésima coluna da i-ésima sequência no alinhamento, essa função pode ser definida como segue:

$$SP = \sum_{k=1}^{n} \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} score(c_{ik}, c_{jk}). \tag{5.1}$$

- **Soma dos Pares com Função Afim para penalidade de *gaps* (SPFA)**: esta função é similar a primeira, mas penaliza menos uma sequência contínua de *gaps* devido a seu significado biológico. A pontuação para o alinhamento de um par de resíduos é feita como na função apresentada na Equação 5.1. Já a penalização de *gaps* deixa de ser individualizada, *gap* a *gap*, para uma onde um bloco contínuo de x *gaps* recebe uma pontuação menor que $x \times gap$ e, dessa forma, prioriza o agrupamento deles. Na Equação 5.2 é apresentada a função utilizada para penalização de um bloco contínuo de *gaps*. Nesta, *gop* é a penalização atribuída a abertura do bloco e *gep* a penalização para sua extensão de acordo com o comprimento k do bloco. O valor de *gep* deve ser menor que o valor de *gop*. Para s_i representando a sequência i do alinhamento, a função para SPFA pode ser definida como apresentada na Equação 5.3.

$$g = gop + k \times gep \tag{5.2}$$

$$SPFA = \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} score(s_i, s_j) \tag{5.3}$$

Utilizando os métodos descritos nas seções 5.2.1, 5.2.2, 5.2.3 e 5.2.4, testes foram realizados para avaliar a função de aptidão mais adequada para utilizar no alinhador. Utilizou-se os conjuntos de referência RV11 e RV12. Para cada entrada nos conjuntos, foram realizadas 20 execuções com populações de 100 indivíduos, que evoluíram por 650 gerações. Os resultados são apresentados na Tabela 5.7.

Tabela 5.7: Resultados dos testes para a função de aptidão. O ALGAe foi avaliado usando SP e SPFA. Nos testes foram utilizados os conjuntos de referência RV11 e RV12. Os valores correspondem a média da pontuação SP.

Conjunto	**Função**	**BLOSUM62**	**BLOSUM80**	**PAM100**	**PAM250**
RV11	SP	33,60	34,90	28,90	28,90
RV11	SPFA	35,60	38,10	30,50	32,70
RV12	SP	73,90	75,70	74,20	75,20
RV12	SPFA	73,20	75,90	72,80	76,10

Os resultados desse teste levaram a definição de soma dos pares com função afim para penalidade de *gaps* como a função de aptidão padrão.

Evolução da aptidão ao longo das gerações

Uma vez definida a função de aptidão, passou-se a avaliar a evolução da população ao longo das gerações. O objetivo foi identificar a efetividade dos operadores e verificar se o valor definido para número de gerações estava adequado.

Utilizou-se, neste teste, o conjunto RV12 completo e BLOSUM80. Variou-se a penalização para *gaps*. Foram cinco as penalizações para *gap* aplicadas (-10,-2,-3), (-11,-5,-1), (-15,-1,-1), (-20,-2,-2) e (-20,-5,-2), onde o primeiro valor refere-se a `gop`, o segundo refere-se a `gep` e o terceiro $gap \times gap$. Na Figura 5.1 é apresentado um gráfico com os resultados. No eixo X tem-se a geração (número da iteração no GA) e o no Y a pontuação SP alcançada. Observe que aqui a pontuação SP refere-se ao percentual atingido na geração em relação ao SP máximo ao longo de todas as gerações e, assim, no final todos chegam a 100%.

É possível notar que rapidamente a população estabiliza. Com cerca de 100 gerações a população atinge pontuação muito próxima àquela que atingirá ao final, com 650 gerações. Nota-se também que o grau de aptidão não evolui muito. A população inicial já possui indivíduos com cerca de 97% da aptidão do indivíduo de maior aptidão no final do GA.

Esses resultados levaram a novos testes, onde buscou-se parâmetros adequados para penalização de *gaps*, variações na geração da população inicial, uma nova função de aptidão que leva em consideração alinhamento de estruturas e inserção de um novo operador baseado em consensos.

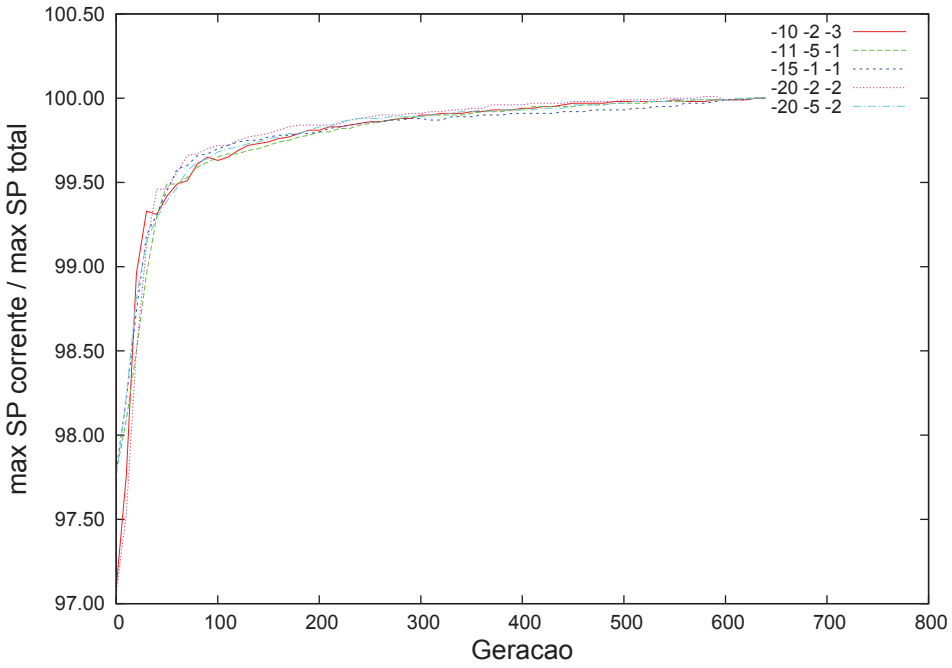


Figura 5.1: Evolução da aptidão ao longo das gerações.

5.2.6 Definição de penalização para *gaps*

O trabalho prosseguiu com uma busca por valores adequados para penalização de *gaps*. Note que para pontuar o alinhamento entre dois resíduos tem-se séries de matrizes de substituição, tais como PAM [56] e BLOSUM [106]. Essas matrizes são construídas fundamentando-se nas semelhanças e discrepâncias entre os aminoácidos, com base nas probabilidades que cada um tem de substituir os outros em sequências relacionadas. Dessa forma, pontuações para *matches* e *mismatches* são definidas sob uma sólida fundamentação biológica e estatística. Penalidade para *gaps*, por outro lado, não é suportada por teoria forte. Assim, é geralmente definida com base em métodos empíricos.

Vingron e Waterman [257] realizaram uma das primeiras análises empíricas para penalidade de *gaps*. Neste trabalho é apresentada uma comparação experimental de duas sequências de imunoglobinas e indica que as menores penalidades para *gaps*, que ainda produzam bons alinhamentos locais, são as combinações mais eficazes.

Em um outro trabalho, Reese e Pearson [203] analisaram as penalidades para *gaps* que maximizam a eficácia de buscas por sequências homólogas em um banco de dados de sequências. Este trabalho focou nas matrizes PAM e propôs um relacionamento sistemático entre distância PAM e as penalidades para abertura e extensão de *gaps*. Eles também afirmam, baseados nos resultados experimentais, que as penalidades para extensão não mudam consideravelmente de acordo com as distâncias PAM. Baseados nos

resultados experimentais para matrizes PAM, os autores extrapolaram as penalidades a utilizar para algumas matrizes BLOSUM.

Já Agrawal e colegas [4], usando significância estatística para pares, realizaram um estudo similar para matrizes BLOSUM focando em penalidades para abertura de *gaps*. A penalidade para extensão de *gap* foi mantida constante com valor -2.

Neste trabalho também foi realizado um estudo acerca de valores para penalização de *gaps*. Agora, porém, o foco esteve na observação de como a penalidade pode afetar a qualidade de um alinhamento construído por um algoritmo genético.

Inicialmente buscou-se observar qual a penalização adequada para a penalização do alinhamento de um par de *gaps* em uma mesma coluna. Avaliou-se penalizações entre 0 e -3, variando penalização para abertura de *gaps* entre -10 e -20[1] e a penalização para extensão de *gaps* entre -1 e -5. Nessa avaliação utilizou-se o conjunto de entradas RV12 completo e pontuou-se através de SP e TC. Os resultados obtidos são apresentados na Tabela 5.8.

As maiores pontuações (SP e TC) obtidas para cada penalização $gap \times gap$ foram destacadas em negrito. Ao observar esses valores é possível notar uma clara tendência de queda na qualidade do alinhamento a medida que a penalização é incrementada. Dessa forma, passou-se a utilizar penalização $gap \times gap = 0$.

Em seguida, passou-se a avaliar penalizações para abertura (`gop`) e extensão (`gep`) de *gaps*. Todos os parâmetros do ALGAe, com exceção apenas da matriz de substituição e os valores para `gop` e `gep`, foram mantidos inalterados ao longo de toda a avaliação. Dessa forma, evita-se a influência destes nos resultados.

Neste teste foram avaliadas diferentes combinações para os valores de `gop` e `gep`, usando BLOSUM62 e BLOSUM80 como matrizes de substituição. Para BLOSUM62 avaliou-se $gop = [-2, -15]$ e $gep = [0, -5]$. Já para BLOSUM80 avaliou-se $gop = [-4, -20]$ e $gep = [0, -5]$. *Matches gap-gap* não foram penalizados. Para comparar os resultados as médias das pontuações SP e TC no conjunto RV12 inteiro foram utilizadas. Para cada entrada do conjunto foram realizadas 20 execuções.

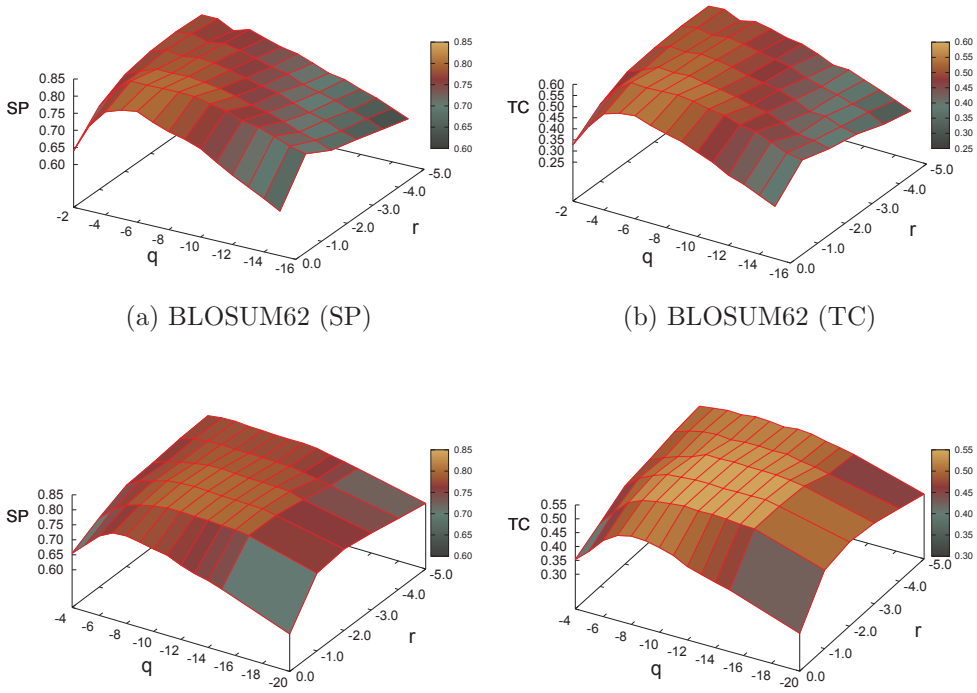
Na Figura 5.2 são apresentados os resultados obtidos pelo ALGAe nessa avaliação.

Analisando os resultados da Figura 5.2 é possível notar que, se um dos parâmetros (`gop` ou `gep`) é mantido constante, os valores máximos para SP e TC tendem a estar associados com os mesmos valores para a variável livre, o que pode indicar a independência entre `gop` e `gep`. É importante também destacar que os resultados para pontuação SP indicam que ela é mais sensível a alterações na penalidade para *gaps*.

Considerando o desvio padrão e o comportamento independente das variáveis `gop` e `gep`, a melhor penalidade encontrada para *gap* nesse experimento foi $gop = -5$ e $gep = -1$ para BLOSUM62 e $gop = -9$ e $gep = -2$ para BLOSUM80.

[1] Devido ao alto custo computacional, o intervalo de inteiros não foi completamente testado.

Para finalizar esse experimento, comparou-se os melhores resultados encontrados com os valores que puderam ser obtidos usando o ALGAe com as penalidades para *gap* sugeridas por Reese e Pearson [203] e Agrawal e colegas [4]. Os resultados estão sumarizados na Tabela 5.9.



(a) BLOSUM62 (SP) (b) BLOSUM62 (TC)

(c) BLOSUM80 (SP) (d) BLOSUM80 (TC)

Figura 5.2: Resultados apresentados pelo ALGAe quando variou-se a matriz de substituição e os valores para penalização de abertura (*gop*) e extensão (*gep*) de *gaps*.

Tabela 5.8: Resultados de testes para penalização $gap \times gap$. Nesse teste foram utizadas penalizações $gap \times gap$ entre 0 e -3, penalizações para abertura de *gaps* -10, -11, -12, -13, -15 e -20, e penalizações para extensão de *gaps* -1, -2 e -5. Foram avaliadas todas as entradas do conjunto RV12 e são apresentadas as pontuações SP e TC.

Penalização $gap \times gap = 0$

	SP				**TC**		
	-1	**-2**	**-5**		**-1**	**-2**	**-5**
-10	80,56	**80,58**	77,35	**-10**	**54,94**	54,90	51,07
-11	80,28	80,33	77,40	**-11**	54,53	54,56	51,68
-12	80,28	80,25	77,04	**-12**	54,72	54,72	51,27
-13	79,96	80,14	76,08	**-13**	54,43	54,67	50,07
-15	79,28	79,13	74,30	**-15**	53,73	53,73	48,15
-20	73,42	74,65	69,39	**-20**	45,86	47,93	40,90

Penalização $gap \times gap = -1$

	SP				**TC**		
	-1	**-2**	**-5**		**-1**	**-2**	**-5**
-10	74,56	74,86	74,80	**-10**	50,18	50,36	50,23
-11	75,18	74,08	75,18	**-11**	51,27	51,58	51,42
-12	**76,67**	75,00	75,11	**-12**	**51,73**	50,93	51,06
-13	74,95	76,18	76,09	**-13**	50,87	50,78	50,82
-15	75,62	75,73	75,65	**-15**	50,87	50,94	50,70
-20	72,41	72,58	72,64	**-20**	43,96	44,24	44,49

Penalização $gap \times gap = -2$

	SP				**TC**		
	-1	**-2**	**-5**		**-1**	**-2**	**-5**
-10	73,71	73,73	74,23	**-10**	48,57	48,65	49,45
-11	74,41	74,24	74,21	**-11**	50,11	49,90	50,06
-12	73,84	74,15	73,88	**-12**	49,86	**50,54**	50,05
-13	73,78	73,76	73,65	**-13**	49,00	49,25	48,90
-15	**75,04**	74,69	74,70	**-15**	49,10	48,90	48,92
-20	71,05	71,33	71,31	**-20**	42,94	42,78	42,57

Penalização $gap \times gap = -3$

	SP				**TC**		
	-1	**-2**	**-5**		**-1**	**-2**	**-5**
-10	74,30	72,95	74,64	**-10**	48,30	47,89	48,38
-11	73,22	73,21	72,37	**-11**	48,82	48,40	49,13
-12	73,20	**75,08**	73,30	**-12**	48,32	**49,64**	48,82
-13	73,67	73,36	73,49	**-13**	48,10	47,74	48,04
-15	74,91	74,87	74,57	**-15**	48,03	48,42	48,16
-20	71,10	71,17	70,51	**-20**	42,29	42,44	42,12

Tabela 5.9: Comparação das penalidades para *gaps*. Compara-se os resultados obtidos com os melhores valores encontrados para penalidade contra os valores sugeridos por Reese e Pearson [203] e por Agrawal e colegas [4].

Matriz	Melhores resultados	1- Agrawal et al.	2 - Reese e Pearson	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

Os valores mostram uma variação considerável entre os valores encontrados. Os resultados obtidos usando os novos valores para penalização de *gaps* superaram os resultados para o conjunto RV12, especialmente na pontuação SP, quando comparado com valores sugeridos na literatura. É importante notar, entretanto, que esses valores anteriores foram definidos usando um conjunto maior e mais amplo e são focados em buscas em bancos de dados, não em MSA. Tais resultados podem indicar que penalidade para *gaps* são fortemente dependentes do conjunto de sequências alvo e, assim, não se deveria utilizar um valor geral para penalidade, tal como já afirmado por Barton e Sternberg [28].

5.2.7 Função de aptidão baseada em alinhamentos estruturais

Numa tentativa de aprimorar os resultados do alinhador foi definida uma nova função de aptidão. Esta parte do valor de soma dos pares com função afim para penalidade de *gaps* e soma um fator de correção para cada combinação de par de resíduos. Nos casos onde há um alinhamento de um resíduo com um *gap* ou onde dois *gaps* são alinhados não foi realizada tal soma. Foram definidas três possibilidades de combinação, onde para cada uma é realizada uma soma distinta. A seguir são listadas as possibilidades e o valor a somar.

1. Estrutura do par é a mesma: $adjust \times score(r1, r2)$
2. Estrutura do par é distinta e uma delas é *coil*: $-adjustCoil \times score(r1, r2)$
3. Estrutura do par é distinta e nenhuma delas é *coil*: $-adjust \times score(r1, r2)$

O valor de $score(r1, r2)$ é a pontuação atribuída ao par de resíduos $r1$ e $r2$, considerando a matriz de substituição definida. Os valores utilizados para $adjust$ e $adjustCoil$ foram 3 e 2, respectivamente.

Utilizou-se a ferramenta PSIPRED [129] para predição da estrutura secundária das sequências. Tal ferramenta recebe como entrada um sequência e como saída indica as estruturas que encontrou na sequência e aponta para cada resíduo a estrutura que está

contribuindo para construir. As estruturas preditas são hélice, folha e *coil*. As duas primeiras são estruturas mais bem definidas e, sendo assim, priorizou-se o alinhamento de resíduos nessas estruturas.

Testes foram realizados para essa função utilizando os conjuntos RV11, RV12 e RV13 do BAliBASE 2.0, cujos resultados são apresentados respectivamente nas tabelas 5.10, 5.11 e 5.12. O conjunto RV13 é composto por sequências equidistantes, assim como RV11 e RV12, a característica que os diferencia é a distância entre as sequências. O conjunto RV11 é o de maiores distâncias e o RV13 os de menores.

Os resultados foram inferiores ao que o GA estava alcançando com a função anterior, sendo assim, foi descartada. Esta é uma função que, entretanto, tem potencial de apresentar bons resultados. Eventualmente, com mais testes, seu desempenho pode ser melhorado.

Tabela 5.10: Resultados obtidos pela função objetivo que faz uso de alinhamentos de estruturas para entradas do conjunto RV11 do BAliBASE 2.0. Nesta tabela são apresentadas as pontuações obtidas pela nova função e pela função padrão, assim como um comparativo entre elas.

Entrada	**Estrutura**		**Padrão**		**Comparativo**	
	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	6,30	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	102,09	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

Tabela 5.11: Resultados obtidos pela função objetivo que faz uso de alinhamentos de estruturas para entradas do conjunto RV12 do BAliBASE 2.0. Nesta tabela são apresentadas as pontuações obtidas pela nova função e pela função padrão, assim como um comparativo entre elas.

Entrada	**Estrutura**		**Padrão**		**Comparativo**	
	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	87 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

Tabela 5.12: Resultados obtidos pela função objetivo que faz uso de alinhamentos de estruturas para entradas do conjunto RV13 do BAliBASE 2.0. Nesta tabela são apresentadas as pontuações obtidas pela nova função e pela função padrão, assim como um comparativo entre elas.

Entrada	**Estrutura**		**Padrão**		**Comparativo**	
	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	22,60	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

5.2.8 Métodos alternativos para geração da população inicial

Ao final dos testes descritos na Seção 5.2.6, tinha-se um alinhador múltiplo capaz de obter 80,81 para pontuação SP e 56,80 para pontuação TC. Essas pontuações referem-se a média para todas as entradas do conjunto RV12. Nesta seção são descritas e avaliadas alterações no procedimento de geração da população inicial, denominado `Algoritmo 1` e detalhado na Seção 5.2.1.

Como um primeiro teste, adicionou-se uma terceira abordagem para geração de indivíduos. Gera-se mais um indivíduo utilizando o método ABOT, que é descrito a seguir. Este faz uso de janela deslizante, grafo orientado e ordenação topológica para construir alinhamentos de pares. Para produzir o alinhamento múltiplo utiliza uma variação do alinhamento estrela. Denominou-se essa variação, em relação ao método original para geração da população inicial, de `Algoritmo 2`.

Um outro teste consistiu em adicionar ao `Algoritmo 2` outros novos métodos para a geração de indivíduos. Os métodos AB1, AB2, AB3, AB4 e AB5, que são baseados em blocos, assim como ABOT, foram utilizados. Juntamente a estes, aplicou-se um refinador. Foram adicionados mais 20 indivíduos. Cada alinhador baseado em blocos gerou quatro alinhamentos, o original e mais três através de refinamentos. O refinador possui um parâmetro X que indica a porcentagem mínima de concordância de resíduos para que uma coluna seja mantida. Caso contrário, ela é refeita. Utilizou-se $X = 40$, $X = 50$ e $X = 60$. Chamou-se esse novo procedimento para geração da população inicial de `Algoritmo 3`. Esses novos métodos e o refinador são descritos a seguir.

São também apresentados a seguir os resultados obtidos com a utilização do `Algoritmo 2` e do `Algoritmo 3` para geração da população inicial.

Primeira abordagem (ABOT)

Para cada par de sequências, busca-se, utilizando uma janela deslizante de tamanho fixo k, blocos contínuos (sem *gaps*) e idênticos. Cria-se um grafo onde cada nó representa um bloco β encontrado. Denota-se por β_i e β_j as posições iniciais do bloco e por $\beta_i + k$ e $\beta_j + k$ as posições finais. Arestas (orientadas) conectam blocos consistentes. Por blocos consistentes entende-se dois blocos β^1 e β^2 de forma que:

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

A primeira condição é satisfeita quando um bloco antecede o outro sem qualquer sobreposição. A segunda condição trata do caso de sobreposição, garantindo que esta seja coerente nas duas sequências, ou seja, $\beta_i^2 - \beta_i^1 = \beta_j^2 - \beta_j^1 \geq 1$. Na Figura 5.3 não apresentados dois exemplos de blocos consistentes, subfiguras (a) e (b), e dois exemplos de blocos não consistentes, (c) e (d). Em cada uma deles há um bloco delimitado por um retângulo com linha contínua e um outro com linha tracejada. Por exemplo, em (a) há um bloco composto pelos resíduos KTT que inicia no terceiro resíduo da primeira sequência e no primeiro da segunda. Há outro bloco composto pelos resíduos ADK que inicia no sétimo resíduo da primeira sequência e no quarto resíduo da segunda.

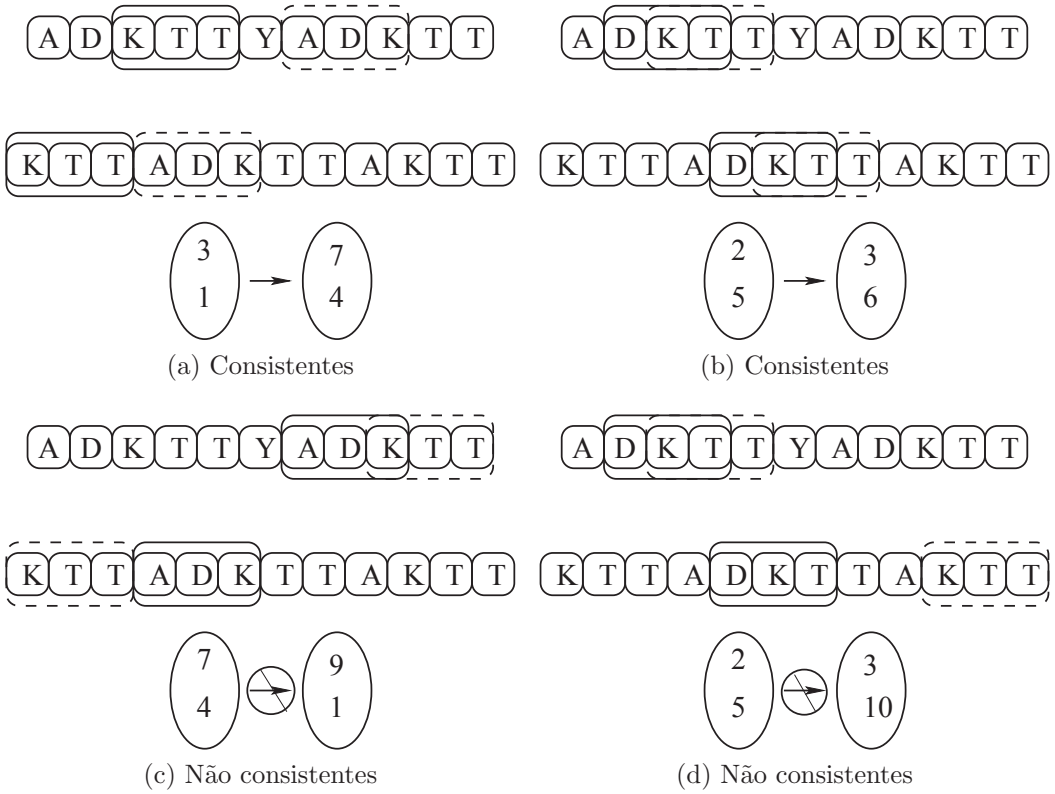
Após esse passo são criados dois nós especiais representando o início e o fim das sequências. São também criadas arestas conectando o nó que representa o início a todos os outros nós, assim como são criadas arestas conectando todos os nós ao nó representando o final das sequências. Procura-se então o caminho máximo nesse grafo. Tal procedimento pode ser feito em tempo polinomial, utilizando-se ordenação topológica, pelo fato do grafo ser orientado e acíclico. Esse caminho representa o maior conjunto consistente de blocos conservados para o par de sequências. Para alinhar as regiões entre os blocos utiliza-se o algoritmo de alinhamento global de Needleman e Wunsch[174].

Um algoritmo progressivo constrói o alinhamento múltiplo. Este utiliza uma árvore gerada por UPGMA [228] a partir de distâncias inversamente proporcionais ao número de blocos conservados entre os pares de sequência. Em seguida, utiliza-se uma variação do alinhamento estrela [219]. Para unir uma sub-árvore A com uma sub-árvore B é utilizado o par de sequências de menor distância. Observe que o par deve ser composto por uma sequência de A e outra de B. Este algoritmo usa os alinhamentos de pares já construídos para o cálculo das distâncias e para a efetiva composição do alinhamento múltiplo em sua etapa final.

Para esse método, denominado ABOT, usou-se tamanho $k = 5$ para a janela deslizante.

Segunda abordagem (ABx)

Inicia procurando todas as *substrings*, de um tamanho pré-definido k, que ocorrem nas sequências de entrada. Toma-se a *substring* que ocorre no maior número de sequências



(a) Consistentes (b) Consistentes

(c) Não consistentes (d) Não consistentes

Figura 5.3: Exemplos de blocos consistentes e de blocos não consistentes para as sequências ADKTTYADKTT e KTTADKTTAKTT.

como referência para a construção do bloco, que é gerado a partir da primeira ocorrência da *substring* em cada sequência. Faz-se chamadas recursivas para alinhar o restante das sequências, tanto do lado esquerdo como do lado direito.

Note que ao longo da execução desse algoritmo recursivo, sequências ou regiões de sequências são "ignoradas" por não possuírem a *substring*. Essas sequências ou regiões são inseridas no alinhamento através de alinhamentos de perfis, que as adiciona em ordem arbitrária.

Utiliza-se para esse algoritmo, denominado AB1 e detalhado em Algoritmo 6, $k = 6$. Na implementação deste, utiliza-se o alfabeto comprimido Dayhoff(6) [67]. Sendo assim, as sequências são analisadas como uma sequência de classes e não como uma sequência de resíduos de aminoácidos.

Nas linhas 2 e 3 do código apresentado em Algoritmo 6 é implementada a condição de parada. Isso ocorre quando resta menos de duas sequências a alinhar. Na linha 5 é computada a *substring* que ocorre em um maior número de sequências. Na linha 7 é gerada uma lista com a primeira ocorrência da *substring* em cada uma das sequências.

Algoritmo 6: Algoritmo AB1.

```
   Input: sequences
   Output: msa
 1 //Condição de parada
 2 if not canContinueAligment(sequences) then
 3     return computeFinalAlignment(sequences)
 4 // Computa a substring
 5 substring ← computeMaxOccurSubstring(sequences)
 6 //Computa o bloco
 7 occurrences ← getFirstSubstringOccur(sequences)
 8 if length(occurrences) > 1 then
 9     block ← getBlock(sequences,occurrences)
10     //Separa as sequências que pertencem e as que não pertencem ao bloco
11     alignedSeqs ← getAlignedSeqs(block)
12     otherSeqs ← getOtherSeqs(sequences,alignedSeqs)
13     //Realiza as chamadas recursivas
14     leftSeqs ← extractLeftSide(sequences, block)
15     leftAlign ← AB1(leftSeqs)
16     rightSeqs ← extractRightSide(sequences, block)
17     rightAlign ← AB1(rightSeqs)
18     //Combina os resultados da recursão com o bloco
19     partialAlignment ← leftAlign + block + rightAlign
20 else
21     partialAlignment ← null
22     otherSeqs ← sequences
23 //Adiciona as sequências "ignoradas" ao alinhamento
24 return addSeqsByProfileAlignment(partialAlignment,otherSeqs)
```

Caso a *substring* ocorra em mais que uma sequência, é executado o bloco entre as linhas 9 e 19. Nesse caso, inicia-se computando o bloco na linha 9 e gera-se uma lista com as sequências que não foram adicionadas ao bloco nas linhas 11 e 12. Em seguida realiza-se a recursão à esquerda nas linhas 14 e 15 e a recursão à direita nas linhas 16 e 17. Esse bloco do código é concluído com a combinação dos alinhamentos obtidos pelas recursões com o bloco computado. No caso em que não for possível gerar um bloco com pelo menos duas sequências, simplesmente define alinhamento parcial como nulo e indica que nenhuma das sequências foi alinhada ainda. O algoritmo termina pela adição das sequências "ignoradas" ao alinhamento parcialmente gerado e então retorna-se o alinhamento produzido.

Numa variação de AB1, denominada AB2 e detalhada em Algoritmo 7, tenta-se agrupar as sequências ou regiões de sequência "ignoradas" utilizando-se o algoritmo recursivo.

Podem ser criados tantos grupos quanto necessário. Os subalinhamentos oriundos dos grupos formados são agrupados por alinhamentos de perfis e as sequências ou regiões que ainda não estejam alinhadas ao final desse processo são adicionadas em ordem arbitrária através de alinhamento de perfis.

Observe que o código foi alterado apenas no bloco do código onde são realizadas as recursões. Foram inseridas as linhas 20 a 24 no final do bloco. Nelas faz-se uma chamada recursiva com as sequências que foram inicialmente "ignoradas". Dessa forma, recursivamente, gera quantos grupos forem possíveis e finaliza inserindo as sequências que ainda assim foram "ignoradas". É nesse trecho também que é feita a combinação do grupo e o alinhamento gerado com as sequências inicialmente "ignoradas".

Numa variação de AB2, denominada AB3, em vez de construir os blocos pela escolha arbitrária da primeira ocorrência da *substring* nas sequências, realiza-se uma busca pela melhor combinação de ocorrências. Para isso é realizado um passo de expansão. Para cada combinação das ocorrências, expande-se lateralmente, tanto quanto possível, o bloco. Tanto à direita quanto à esquerda, avança-se adicionando ao bloco a classe de maior ocorrência naquela hipotética nova coluna do alinhamento. O procedimento é interrompido quando não há ao menos duas ocorrências de uma mesma classe. Observe que o bloco pode ser formado por subsequências de comprimentos distintos. A pontuação para a combinação das ocorrências é dada pela soma dos comprimentos das subsequências no bloco formado ao final da expansão.

Essa variação é detalhada em Algoritmo 8. Note que a linha 7 do código foi alterada para trocar a primeira ocorrência da *substring* pela melhor. Foi também inserida a linha 10, onde realiza-se a expansão lateral seguida de um ajuste pela inserção de *gaps* para eventualmente completar o bloco.

Em AB4, que é uma variação de AB3, adiciona-se um passo de expansão vertical. Este adicionará sequências que seriam "ignoradas" na recursão desde que possua uma *substring* com apenas uma diferença em relação a *substring* que ocorre em um maior número de sequências. O passo de expansão lateral é realizado em seguida.

Essa variação é detalhada em Algoritmo 9. Note que agora foi inserida, na linha 10, a expansão vertical antes da expansão lateral.

A variação AB5, inspirada em AB4, é detalhada em Algoritmo 10. Adiciona-se um segundo passo de expansão lateral. Este permitirá uma classe distinta do consenso a cada k classes na sequência. Esse passo, entretanto, não é capaz de expandir a largura do bloco.

As alterações realizadas em relação ao algoritmo anterior estão nas linhas 11 e 12. Na linha 11 é feita a expansão lateral e em seguida a nova expansão lateral, que é mais flexível. Na próxima linha é então feito o ajuste que eventualmente insere *gaps* para completar o bloco.

Refinador

Tal método baseia-se em refazer colunas mal alinhadas. Para isso, busca-se por colunas com menos de $X\%$ de concordância de classe de Dayhoff(6) no alinhamento. Colunas contínuas mal alinhadas determinam blocos que são refeitos.

Para refazer o bloco é utilizado um alinhador progressivo. Este usa modelo das categorias do PHYLIP para computar a matriz de distâncias, UPGMA para gerar a árvore guia e par mais próximo com alinhamento de perfis e função afim para penalização de *gaps* para produzir o alinhamento múltiplo.

Na Tabela 5.13 são apresentados os resultados obtidos pelos alinhadores de blocos da segunda abordagem. São apresentados também os resultados ao aplicar o refinador descrito nesta seção com $X = 40$, $X = 50$ e $X = 60$. Os valores correspondem a média da pontuação SP para todas as entradas do conjunto RV12 do BAliBASE 3.0.

Pode-se observar que o uso do refinador eleva a qualidade dos alinhamentos, assim como, à medida que ele se torna mais restritivo, observa-se também um incremento na qualidade. Observa-se uma exceção a esse comportamento para AB1, que inclusive obteve os melhores resultados. Isso pode ter ocorrido por efetivamente realizar um menor percentual de alinhamento por blocos e assim faz um maior uso do alinhamento progressivo, um algoritmo mais robusto.

Resultados

Na Tabela 5.14 são apresentados os resultados dos testes com os novos algoritmos para geração da população inicial. Seria esperado que, pela inclusão de novos indivíduos e incremento na diversidade da população, a pontuação fosse elevada a medida que novas abordagens fossem adicionadas à geração da população inicial. Os resultados não expuseram isso, assim como não foi observada uma relação direta entre as pontuações SP e TC. Observa-se a melhor pontuação SP com o `Algoritmo 2` e a melhor pontuação TC com o `Algoritmo 1`. A variação na pontuação, entretanto, é pouco significativa.

Algoritmo 7: Algoritmo AB2.

```
   Input: sequences
   Output: msa
 1 //Condição de parada
 2 if not canContinueAligment(sequences) then
 3     return computeFinalAlignment(sequences)
 4 // Computa a substring
 5 substring ← computeMaxOccurSubstring(sequences)
 6 //Computa o bloco
 7 occurrences ← getFirstSubstringOccur(sequences)
 8 if length(occurrences) > 1 then
 9     block ← getBlock(sequences,occurrences)
10     //Separa as sequências que pertencem e as que não pertencem ao bloco
11     alignedSeqs ← getAlignedSeqs(block)
12     otherSeqs ← getOtherSeqs(sequences,alignedSeqs)
13     //Realiza as chamadas recursivas
14     leftSeqs ← extractLeftSide(sequences, block)
15     leftAlign ← AB2(leftSeqs)
16     rightSeqs ← extractRightSide(sequences, block)
17     rightAlign ← AB2(rightSeqs)
18     //Combina os resultados da recursão com o bloco
19     partialAlignment ← leftAlign + block + rightAlign
20     //Tenta formar outro grupo e o respectivo alinhamento
21     groupAlign ← AB2(otherSeqs)
22     alignedSeqs ← getAlignedSeqs(groupAlign)
23     otherSeqs ← getOtherSeqs(otherSeqs,alignedSeqs)
24     partialAlignment ← combineByProfileAlignment(partialAlignment,groupAlign)
25 else
26     partialAlignment ← null
27     otherSeqs ← sequences
28 //Adiciona as sequências "ignoradas" ao alinhamento
29 return addSeqsByProfileAlignment(partialAlignment,otherSeqs)
```

Algoritmo 8: Algoritmo AB3.

```
   Input: sequences
   Output: msa
 1 //Condição de parada
 2 if not canContinueAligment(sequences) then
 3     return computeFinalAlignment(sequences)
 4 // Computa a substring
 5 substring ← computeMaxOccurSubstring(sequences)
 6 //Computa o bloco
 7 occurrences ← getBestSubstringOccur(sequences)
 8 if length(occurrences) > 1 then
 9     block ← getBlock(sequences,occurrences)
10     block ← adjustAlignment(sideExpansion(block,sequences))
11     //Separa as sequências que pertencem e as que não pertencem ao bloco
12     alignedSeqs ← getAlignedSeqs(block)
13     otherSeqs ← getOtherSeqs(sequences,alignedSeqs)
14     //Realiza as chamadas recursivas
15     leftSeqs ← extractLeftSide(sequences, block)
16     leftAlign ← AB3(leftSeqs)
17     rightSeqs ← extractRightSide(sequences, block)
18     rightAlign ← AB3(rightSeqs)
19     //Combina os resultados da recursão com o bloco
20     partialAlignment ← leftAlign + block + rightAlign
21     //Tenta formar outro grupo e o respectivo alinhamento
22     groupAlign ← AB3(otherSeqs)
23     alignedSeqs ← getAlignedSeqs(groupAlign)
24     otherSeqs ← getOtherSeqs(otherSeqs,alignedSeqs)
25     partialAlignment ← combineByProfileAlignment(partialAlignment,groupAlign)
26 else
27     partialAlignment ← null
28     otherSeqs ← sequences
29 //Adiciona as sequências "ignoradas" ao alinhamento
30 return addSeqsByProfileAlignment(partialAlignment,otherSeqs)
```

Algoritmo 9: Algoritmo AB4.

```
Input: sequences
Output: msa
//Condição de parada
if not canContinueAligment(sequences) then
    return computeFinalAlignment(sequences)
// Computa a substring
substring ← computeMaxOccurSubstring(sequences)
//Computa o bloco
occurrences ← getBestSubstringOccur(sequences)
if length(occurrences) > 1 then
    block ← getBlock(sequences,occurrences)
    block ← verticalExpansion(block,sequences)
    block ← adjustAlignment(sideExpansion(block,sequences))
    //Separa as sequências que pertencem e as que não pertencem ao bloco
    alignedSeqs ← getAlignedSeqs(block)
    otherSeqs ← getOtherSeqs(sequences,alignedSeqs)
    //Realiza as chamadas recursivas
    leftSeqs ← extractLeftSide(sequences, block)
    leftAlign ← AB4(leftSeqs)
    rightSeqs ← extractRightSide(sequences, block)
    rightAlign ← AB4(rightSeqs)
    //Combina os resultados da recursão com o bloco
    partialAlignment ← leftAlign + block + rightAlign
    //Tenta formar outro grupo e o respectivo alinhamento
    groupAlign ← AB4(otherSeqs)
    alignedSeqs ← getAlignedSeqs(groupAlign)
    otherSeqs ← getOtherSeqs(otherSeqs,alignedSeqs)
    partialAlignment ← combineByProfileAlignment(partialAlignment,groupAlign)
else
    partialAlignment ← null
    otherSeqs ← sequences
//Adiciona as sequências "ignoradas" ao alinhamento
return addSeqsByProfileAlignment(partialAlignment,otherSeqs)
```

Algoritmo 10: Algoritmo AB5.

```
   Input: sequences
   Output: msa
 1 //Condição de parada
 2 if not canContinueAligment(sequences) then
 3     return computeFinalAlignment(sequences)
 4 // Computa a substring
 5 substring ← computeMaxOccurSubstring(sequences)
 6 //Computa o bloco
 7 occurrences ← getBestSubstringOccur(sequences)
 8 if length(occurrences) > 1 then
 9     block ← getBlock(sequences,occurrences)
10     block ← verticalExpansion(block,sequences)
11     block ← flexibleSideExpansion(sideExpansion(block,sequences))
12     block ← adjustAlignment(block)
13     //Separa as sequências que pertencem e as que não pertencem ao bloco
14     alignedSeqs ← getAlignedSeqs(block)
15     otherSeqs ← getOtherSeqs(sequences,alignedSeqs)
16     //Realiza as chamadas recursivas
17     leftSeqs ← extractLeftSide(sequences, block)
18     leftAlign ← AB5(leftSeqs)
19     rightSeqs ← extractRightSide(sequences, block)
20     rightAlign ← AB5(rightSeqs)
21     //Combina os resultados da recursão com o bloco
22     partialAlignment ← leftAlign + block + rightAlign
23     //Tenta formar outro grupo e o respectivo alinhamento
24     groupAlign ← AB5(otherSeqs)
25     alignedSeqs ← getAlignedSeqs(groupAlign)
26     otherSeqs ← getOtherSeqs(otherSeqs,alignedSeqs)
27     partialAlignment ← combineByProfileAlignment(partialAlignment,groupAlign)
28 else
29     partialAlignment ← null
30     otherSeqs ← sequences
31 //Adiciona as sequências "ignoradas" ao alinhamento
32 return addSeqsByProfileAlignment(partialAlignment,otherSeqs)
```

Tabela 5.13: Resultados obtidos pelos alinhadores de blocos da segunda abordagem e do refinador usando $X = 40$, $X = 50$ e $X = 60$.

Alinhador	**Sem refinamento**	$X = 40$	$X = 50$	$X = 60$
AB1	50,82	50,73	50,64	50,75
AB2	42,28	42,35	42,68	43,92
AB3	39,70	39,73	40,03	41,63
AB4	36,39	37,08	37,70	39,89
AB5	36,65	37,31	37,94	40,16

Tabela 5.14: Resultados obtidos pelo ALGAe quando variou-se o procedimento para geração da população inicial. Os valores correspondem a média para o conjunto RV12.

População Inicial	**SP**	**TC**
Algoritmo 1	80,81	**56,80**
Algoritmo 2	**81,36**	55,20
Algoritmo 3	80,32	55,75

5.2.9 Operador baseado em consenso

Em seguida, passou-se a testar um operador baseado em consenso. O objetivo foi avaliar a possibilidade de elevar a qualidade dos alinhamentos produzidos pelo alinhador iterativo estocástico através de um operador de cruzamento mais robusto. Para a implementação do operador, foi utilizada a ferramenta M-COFFEE [260]. Esta implementa um meta-método consenso baseado no T-COFFEE [183], que produz como saída um alinhamento consistente com os alinhamentos recebidos como entrada.

Inicialmente esse novo operador foi avaliado concorrendo pelo uso juntamente com os outros operadores de cruzamento já existentes. Os resultados desse teste são apresentados na Tabela 5.15 e não exibiram variação significativa em relação aos resultados apresentados na Tabela 5.14.

Tabela 5.15: Resultados obtidos pelo ALGAe quando variou-se o procedimento para geração da população inicial e foi incluído o operador baseado em consenso concorrendo pelo uso juntamente com os outros operadores de cruzamento.

População Inicial	**SP**	**TC**
Algoritmo 1	**81,95**	55,93
Algoritmo 2	80,77	**56,02**
Algoritmo 3	79,90	55,93

Foi ainda realizado um segundo teste, onde desta vez o operador baseado em consenso não concorria pelo uso. Após a aplicação dos operadores de cruzamento e mutação originais, listados na Seção 5.2.3, foi executado o operador baseado em consenso utilizando como entrada sequências arbitrariamente escolhidas da população corrente.

Nesse teste variou-se o método para geração da população inicial, o número de vezes que o novo operador foi executado em cada geração e o número de sequências dadas como entrada para o operador. Na Tabela 5.16 são apresentados os resultados obtidos.

Dessa vez pode-se observar uma considerável elevação na qualidade dos alinhamentos quando compara-se com os resultados apresentados nas tabelas 5.14 e 5.15. Antes o maior valor alcançado para pontuação SP era 81,95 e 56,80 para TC. Agora tem-se 87,42 para SP e 70,84 para TC, cujos resultados foram alcançados com o `Algoritmo 3` para geração da população inicial e duas execuções do operador usando cinco sequências como entrada. Foi uma variação de 5,47 para SP e de 14,04 para TC numa escala para pontuação que vai de 0 a 100. Mesmo os piores resultados apresentados na Tabela 5.16 superam significativamente os melhores resultados apresentados nas tabelas 5.14 e 5.15.

Na tabela também é possível observar que não há uma relação direta entre o número de execuções do operador e a elevação na qualidade do alinhamento. Nota-se, entretanto,

uma forte relação entre o número de sequências dadas como entrada para operador e a elevação na qualidade do alinhamento.

Tabela 5.16: Resultados do ALGAe utilizando o operador baseado em consenso para o conjunto RV12. Os valores referem-se a média para todas as entradas do conjunto de referência. Os resultados estão agrupados pelo número de sequências utilizadas em cada execução do operador. Testou-se execuções com duas, três, quatro e cinco sequências. Testou-se também, para cada uma dessas opções, três algoritmos para geração de população inicial e variou-se o número de execuções do operador a cada geração. A maior e a menor pontuação SP e TC para cada bloco de resultados estão destacadas.

2 Sequências								
PI	**SP**				**TC**			
	1x	**2x**	**3x**	**4x**	**1x**	**2x**	**3x**	**4x**
Algoritmo 1	**85,79**	84,96	85,77	85,47	**67,02**	66,05	66,64	66,25
Algoritmo 2	84,84	85,65	84,48	85,70	63,64	66,64	**62,82**	65,14
Algoritmo 3	85,61	**83,89**	85,20	85,12	66,52	64,23	63,86	64,34
3 Sequências								
PI	**SP**				**TC**			
	1x	**2x**	**3x**	**4x**	**1x**	**2x**	**3x**	**4x**
Algoritmo 1	**86,77**	85,83	86,22	85,87	**69,50**	66,52	67,59	66,61
Algoritmo 2	85,93	85,98	85,69	85,89	67,02	66,20	**66,11**	67,41
Algoritmo 3	86,15	86,40	**85,40**	85,66	68,32	68,66	66,39	67,09
4 Sequências								
PI	**SP**				**TC**			
	1x	**2x**	**3x**	**4x**	**1x**	**2x**	**3x**	**4x**
Algoritmo 1	87,04	87,12	87,07	86,72	68,86	69,61	69,34	69,34
Algoritmo 2	86,55	**86,08**	86,61	**87,41**	68,77	**66,52**	68,41	69,64
Algoritmo 3	87,10	86,98	87,02	86,52	69,93	**70,11**	69,30	68,39
5 Sequências								
PI	**SP**				**TC**			
	1x	**2x**	**3x**	**4x**	**1x**	**2x**	**3x**	**4x**
Algoritmo 1	87,01	**86,01**	87,08	86,99	69,05	**67,95**	69,50	69,91
Algoritmo 2	86,66	86,50	86,86	86,73	68,41	68,68	69,16	68,73
Algoritmo 3	87,30	**87,42**	86,86	86,87	70,09	**70,84**	69,52	68,55

Capítulo 6

Considerações finais

Durante o trabalho desenvolvido nesta tese, abordou-se o problema do alinhamento múltiplo de sequências de proteína. Visando o desenvolvimento de soluções eficientes para a construção de MSAs, o problema foi abordado de três formas distintas.

A primeira abordagem trabalhada, tratada no Capítulo 3, foi alinhamento progressivo. Além de desenvolver e avaliar 342 alinhadores múltiplos, pela combinação de diferentes métodos aplicados a cada uma das três etapas que compõem a abordagem, avaliou-se esses métodos empregados na implementação dos alinhadores e foram identificados os melhores e piores por categoria de método. Foram implementados dois métodos para computação de matriz de distâncias, além de usar mais quatro disponíveis na ferramenta PHYLIP. Utilizou-se dois métodos clássicos para geração de árvores guias, disponíveis na ferramenta R. Dois métodos clássicos para seleção de par foram implementados, baseando-se em suas descrições. Foram implementados doze métodos para agrupamento de alinhamentos, inspirando-se nos métodos clássicos, alinhamento de consensos e alinhamento de perfis. Implementou-se também o esquema de pesos, que pode ser empregado em alinhamentos de perfis.

Um trabalho, intitulado "Progressive Multiple Protein Sequence Alignment" (A. Almeida, M. Souza e Z. Dias) [7], foi publicado na forma de um resumo estendido e apresentado na forma de pôster no *6th International Symposium on Bioinformatics Research and Applications* (ISBRA 2010), que ocorreu em Storrs, Connecticut, EUA, em 2010.

A segunda abordagem utilizada para o desenvolvimento de soluções, detalhada no Capítulo 4, foi alinhamento baseado em consistência. Neste estágio do trabalho, em vez de implementar novos alinhadores, decidiu-se tentar incrementar o desempenho de um alinhador conhecido e que já era capaz de produzir alinhamentos de ótima qualidade. Foram implementadas 89 variações do algoritmo original do MUMMALS e melhorias significativas foram alcançadas.

Os resultados relativos a essa abordagem foram publicados na forma de um artigo

completo, intitulado "Improvements to a multiple protein sequence alignment tool" (A. Almeida e Z. Dias) [6], e apresentado de forma oral na "International Conference on Bioinformatics Models, Methods and Algorithms", que ocorreu em Vilamoura, Algarve, Portugal, em 2012.

Ao final do trabalho, abordou-se o problema através de alinhamentos iterativos. Essa terceira parte do trabalho foi apresentada no capítulo 5, onde foram descritas as implementações de dois alinhadores iterativos não estocásticos e um alinhador iterativo estocástico que faz uso de algoritmos genéticos. Nesse último alinhador foram feitas diversas variações, que foram avaliadas passo-a-passo. Nele também fez-se uso de outras abordagens para construção de MSAs. Usou-se alinhamentos baseados em blocos para gerar indivíduos para a população inicial, alinhamento baseado em consenso na implementação de um operador de cruzamento e alinhamento baseado em modelos na definição de uma função de aptidão. Foram implementados seis alinhadores de bloco, utilizando duas abordagens distintas. Assim como utilizou-se M-COFFEE para gerar alinhamentos baseados em consenso e PSIPRED para predição de estruturas secundárias. Foi desenvolvido um ambiente parametrizável para execução de algoritmos genéticos para construção de MSAs, chamado ALGAe, que pode ser utilizado para realizar avaliações de novos algoritmos genéticos.

Dois trabalhos, "ALGAe: A test-bench environment for a Genetic Algorithm-based Multiple Sequence Aligner" (S. Ordine, A. Grilo, A. Almeida e Z. Dias) [187] e "An Empirical Study for Gap Penalty Score Using a Multiple Sequence Alignment Genetic Algorithm" (S. Ordine, A. Almeida e Z. Dias) [186], foram publicados como resumos estendidos e apresentados na forma de pôster no "Brazilian Symposium on Bioinformatics", sendo o primeiro em Brasília, DF (BSB'2011) e o segundo em Campo Grande, MS (BSB'2012).

A Tabela 6.1 apresenta o desempenho do melhor alinhador progressivo, "53b", e do melhor alinhador baseado em consistência, "SE-B 10 K=7", diante de alinhadores conhecidos. Ambos foram desenvolvidos ao longo deste trabalho. O alinhador "SE-B 10 K=7" é uma variação de MUMMALS 1.01, usa o alfabeto comprimido SE-B(10) e $k = 7$ no método de contagem k-mer. Ele superou todos os alinhadores avaliados. O alinhador "53b" usa JTT para computar a matriz de distâncias, UPGMA para gerar a árvore guia, seleção do par mais próximo e alinhamento de perfis semi-global com função afim para penalização de *gaps*. Não faz uso do esquema de pesos.

Os resultados por conjunto de referência do BAliBASE são apresentados nas tabelas 6.2 e 6.3. A primeira refere-se à pontuação SP e a segunda à pontuação TC.

Tabela 6.1: Desempenho dos alinhadores "SE-B 10 K=7" e "53b", que foram desenvolvidos ao longo deste trabalho, diante de alinhadores conhecidos.

Alinhador	**Tempo (s)**	**SP**	**TC**
SE-B 10 K=7	87.692,75	86,70	56.52
ProbCons 1.12	21.978,50	86,38	55,66
T-COFFEE 8.14	14.631,89	86,11	55,46
MUMMALS 1.01	50.594,17	85,54	53,83
MUSCLE 3.7	1.636,36	82,19	47,59
DiAlign 2.2	7.286,96	77,49	41,52
Clustal W 2.0.10	2.040,10	75,36	37,38
53b	58.810,98	66,45	26,71

Tabela 6.2: Desempenho dos alinhadores "SE-B 10 K=7" e "53b", que foram desenvolvidos ao longo deste trabalho, diante de alinhadores conhecidos, por conjunto de referência e utilizando pontuação SP.

Alinhador	**RV11**	**RV12**	**RV20**	**RV30**	**RV40**	**RV50**
SE-B 10 K=7	69,02	94,89	91,20	84,29	89,82	89,56
ProbCons 1.12	66,97	94,11	91,67	84,60	90,24	89,17
T-COFFEE 8.14	66,32	94,10	91,94	83,79	89,76	89,32
MUMMALS 1.01	66,97	94,30	91,04	84,79	87,18	87,90
MUSCLE 3.7	57,90	91,67	89,17	80,60	87,26	83,39
DiAlign 2.2	50,73	86,66	86,91	74,05	83,17	80,69
Clustal W 2.0.10	50,06	86,44	85,16	72,50	78,93	74,24
53b	41,99	82,36	78,85	63,90	64,34	60,21

6.1 Outros trabalhos desenvolvidos

No período de desenvolvimento desta tese, também foi desenvolvido um trabalho na área de montagem de fragmentos de sequências. Participaram também deste trabalho Maria Angélica L. de Souza (mestranda na ocasião), Felipe R. da Silva (pós-doutorando na ocasião) e Prof. Zanoni Dias.

O trabalho consistiu em desenvolver uma metodologia para avaliação da qualidade dos resultados de diversos montadores, quando na entrada são dados fragmentos oriundos de sequenciamentos Sanger (clássico) e fragmentos curtos oriundos de sequenciamento 454, um dos métodos mais novos que apresentam um alto *throughput* mas, por outro lado, tem tamanhos de *reads* menores. Se a alta taxa de geração de leituras atrai pela consequente redução no tempo e nos custos, os *reads* menores criam novos desafios para

Tabela 6.3: Desempenho dos alinhadores "SE-B 10 K=7" e "53b", que foram desenvolvidos ao longo deste trabalho, diante de alinhadores conhecidos, por conjunto de referência e utilizando pontuação TC.

Alinhador	**RV11**	**RV12**	**RV20**	**RV30**	**RV40**	**RV50**
SE-B 10 K=7	43,34	86,82	40,83	52,80	54,15	58,87
ProbCons 1.12	41,68	85,52	40,49	54,30	52,86	56,69
T-COFFEE 8.14	41,92	85,27	38,90	49,50	55,62	58,75
MUMMALS 1.01	41,63	83,98	42,78	49,33	48,55	52,87
MUSCLE 3.7	33,03	80,45	35,24	38,80	45,96	44,94
DiAlign 2.2	26,50	69,55	29,22	31,23	44,27	42,50
Clustal W 2.0.10	22,74	71,30	21,98	27,23	39,55	30,75
53b	18,39	61,57	9,15	15,70	24,31	23,56

os montadores. Os montadores avaliados foram Cap3, Celera, Minimus, Mira e Velvet. Nesse estudo foram utilizados ESTs de soja, sendo que os *reads* curtos foram obtidos no NCBI [2] (accession number SRR001333) e os longos no *website* do The Genome Institute (TGI) [1], da Universidade de Washington.

Um trabalho, intitulado "A method to evaluate assemblers that make use of short and long transcript reads simultaneously" (F. Silva, M. Souza, A. Almeida e Z. Dias) [222], foi publicado na forma de resumo e apresentado na forma de pôster na "5th International Conference of the Brazilian Association for Bioinformatics and Computational Biology", que ocorreu em Angra dos Reis, RJ, em 2009.

Um outro trabalho, intitulado "Best of Both Words: Using Short and Long Reads to Construct UniGenes" (M. Souza, A. Almeida, A. Andrade, F. Silva e Z. Dias) [231], foi publicado na forma de resumo e apresentado na forma de pôster na "*Plant & Animal Genomes XVII Conference*" (PAG'2009), realizada em San Diego, Califórnia, EUA, em 2009.

6.2 Trabalhos futuros

Com o ambiente para a implementação de algoritmos genéticos (ALGAe), é possível facilmente realizar uma grande variedade de novas avaliações. A introdução de novos operadores baseados em alinhamentos estruturais pode elevar o desempenho do GA, assim como a inserção de diversos métodos, usando as mais variadas abordagens, para a geração da população inicial. O aumento da diversidade dos indivíduos pode propiciar a geração de alinhamentos melhores.

Alinhamento baseado em modelos e alinhamento baseado em consistência parecem ser

o caminho para a evolução das ferramentas para construção de MSAs. São abordagens promissoras para se trabalhar em futuros alinhadores.

Uma outra linha de trabalho, que pode ajudar na geração de alinhadores melhores, é a implementação de ferramentas para visualização de alinhamentos múltiplos com o objetivo de identificar onde estão os trechos mal alinhados. Existem ferramentas para visualização de MSAs, que geralmente recebem como entrada apenas um alinhamento e destacam as colunas com maior nível de compatibilidade entre os resíduos. Esta nova ferramenta estaria focada na comparação de um alinhamento com um alinhamento de referência. Tal ferramenta pode ser útil na detecção dos problemas em alinhamento gerados e, eventualmente, ajudar no desenvolvimento de soluções para lidar com eles.

Ainda uma outra linha de trabalho, que pode resultar em trabalhos de maior impacto, seria trabalhar na busca de melhores alinhamentos múltiplos para problemas específicos em bioinformática. Para o sucesso de um trabalho destes, será imprescindível a aproximação com grupos de biológos ou profissionais que tratem do problema alvo.

Referências Bibliográficas

[1] The Genome Institute, January 2013. http://genome.wustl.edu/.

[2] National Center for Biotechnology Information, January 2013. http://www.ncbi.nlm.nih.gov/.

[3] E. Aart and P. Laarhoven. *Simulated Annealing: a review of theory and applications*. Kluwer Academic Publishers, Amsterdam, 1987.

[4] 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*, volume 4983 of *Lecture Notes in Computer Science*, pages 50–61. Springer Berlin / Heidelberg, 2008.

[5] B. Al-Lazikani, F. Sheinerman, and B. Honig. Combining multiple structure and sequence alignments to improve sequence detection and alignment: application to the SH2 domains of Janus kinases. *Proc. Natl. Acad. Sci. USA*, 98:14796–14801, 2001.

[6] A. Almeida and Z. Dias. Improvements to a multiple protein sequence alignment tool. In *International Conference on Bioinformatics Models, Methods and Algorithms*, pages 226–233, Vilamoura, Portugal, 2012.

[7] A. Almeida, M. Souza, and Z. Dias. Progressive multiple protein sequence alignment. In *6th International Symposium on Bioinformatics Research and Applications - Short Abstracts*, pages 102–105, Storrs, CT, USA, 2010.

[8] E. Althaus, A. Caprara, H. Lenhof, and K. Reinert. Multiple sequence alignment with arbitrary gap costs: computing an optimal solution using polyhedral combinatorics. *Bioinformatics*, 18(90002):S4–S16, 2002.

[9] S. Altschul, M. Boguski, W. Gish, and J. Wootton. Issues in searching molecular sequence databases. *Nat Genet*, 6(2):119–129, 1994.

[10] S. Altschul, R. Carroll, and D. Lipman. Weights for data related by a tree. *J Mol Biol*, 207:647–653, 1989.

[11] S. Altschul and W. Gish. Local alignment statistics. *Methods Enzymol.*, 266:460–480, 1996.

[12] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman. Basic local alignment search tool. *J Mol Biol*, 215(3):403–410, 1990.

[13] S. Altschul and D. Lipman. Trees, stars and multiple biological sequence alignment. *SIAM J. Appl. Math*, 49:197–209, 1989.

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

[15] L. Anabarasu. Multiple sequence alignment using parallel genetic algorithms. In *The Second Asia-Pacific Conference on Simulated Evolution (SEAL-98)*, Canberra, Australia, 1998.

[16] P. Argos. A sensitive procedure to compare amino acid sequences. *J. Mol. Biol.*, 193:385–396, 1987.

[17] 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 Res*, 34:W604–608, 2006.

[18] T. Attwood, M. Croning, D.Flower, and et al. PRINTS-S: the database formerly known as PRINTS. *Nucleic Acids Res.*, 28(1):225–227, 2000.

[19] V. Bafna, E. Lawler, and P. Pevzner. Approximation algorithms for multiple sequence alignment. In *CPM*, pages 43–53, 1994.

[20] T. Bailey and C. Elkan. Fitting a mixture model by expectation maximization to discover motifs in biopolymers. *Proc. Int. Conf. Intell. Syst. Mol. Biol.*, 2:28–36, 1994.

[21] T. Bailey and C. Elkan. The value of prior knowledge in discovering motifs with MEME. In *ISMB*, volume 3, pages 21–29, 1995.

[22] W. Bains. Local sequence dependence of rate of base replacement in mammals. *Mutat. Res.*, 267:43–54, 1992.

[23] A. Bairoch and R. Apweiler. The Swiss-Prot protein sequence data bank and its supplement TrEMBL in 1999. *Nucleic Acids Research*, 27(1):49–54, 1999.

[24] A. Bairoch, B. Boeckmann, S. Ferro, and E. Gasteiger. Swiss-Prot: Juggling between evolution and stability. *Brief. Bioinform.*, 5:39–55, 2004.

[25] A. Bairoch, P. Bucher, and K. Hofmann. The PROSITE database - its status in 1997. *Nucleic Acids Res.*, 25:217–221, 1997.

[26] P. Baldi, Y. Chauvin, T. Hunkapiller, and M. McClure. Hidden Markov models of biological primary sequence information. *Proc. Nat. Acad. Sci.*, 91:1059–1063, 1994.

[27] W. Barker, J. Garavelli, P. McGarvey, C. Marzec, B. Orcutt, G. Srinivasarao, L. Yeh, R. Ledley, H. Mewes, Pfeiffer, A. Tsugita, and C. Wu. The PIR - international protein sequence database. *Nucleic Acids Research*, 27:39–43, 1999.

[28] G. Barton and M. Sternberg. Evaluation and improvements in the automatic alignment of protein sequences. *Protein Engineering*, 1(2):89–94, 1987.

[29] G. Barton and M. Sternberg. A strategy for the rapid multiple alignment of protein sequences: confidence levels from tertiary structure comparisons. *J. Mol. Biol.*, 198:327–337, 1987.

[30] D. Bashford, C. Chothia, and A. Lesk. Determinants of a protein fold: Unique features of the globin amino acid sequences. *J. Mol. Biol.*, 196:199–216, 1987.

[31] A. Bateman, E. Birney, R. Durbin, S. Eddy, K. Howe, and E. Sonnhammer. The Pfam protein families database. *Nucleic Acids Res.*, 28(1):263–266, 2000.

[32] M. Bauer, G. Klau, and K. Reinert. Multiple structural RNA alignment with lagrangian relaxation. *Lecture Notes in Computer Science*, 3692:303–314, 2005.

[33] L. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *Ann. Math. Statist.*, 41(1):164–171, 1970.

[34] A. Baxevanis and D. Landsman. Histone sequence database: new histone fold family members. *Nucleic Acids Research*, 26(1):372–375, 1998.

[35] D. Benson, I. Karsch-Mizrachi, D. Lipman, J. Ostell, and D. Wheeler. GenBank. *Nucleic Acids Res.*, 36:25–30, 2008.

[36] M. Berger and P. Munson. A novel randomized iterative strategy for aligning multiple protein sequences. *Comput. Applic. Biosci.*, 7:479–484, 1991.

[37] H. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. Bhat, H. Weissig, I. Shindyalov, and P. Bourne. The Protein Data Bank. *Nucleic Acids Res*, 28(1):235–242, 2000.

[38] G. Blackshields, I. Wallace, M. Larkin, and D. Higgins. Analysis and comparison of benchmarks for multiple sequence alignment. *In Silico Biol*, 6(4):321–339, 2006.

[39] N. Bray and L. Pachter. MAVID: constrained ancestral alignment of multiple sequences. *Genome Res.*, 14:693–699, 2004.

[40] P. Briffeuil, G. Baudoux, C. Lambert, X. de Bolle, C. Vinals, E. Feytmans, and E. Depiereux. Comparative analysis of seven multiple protein sequence alignment servers: clues to enhance reliability of predictions. *Bioinformatics*, 14(4):357–366, 1998.

[41] L. Brocchieri and S. Karlin. A symmetric-iterated multiple alignment of protein sequences. *J. Mol. Biol.*, 276(1):249–264, 1998.

[42] M. Brudno, M. Chapman, B. Gottgens, S. Batzoglou, and B. Morgenstern. Fast and sensitive multiple alignment of large genomic sequences. *BMC Bioinformatics*, 4:66, 2004.

[43] M. Brudno, C. Do, G. Cooper, M. Kim, E. Davydov, , E. Green, A. Sidow, and S. Batzoglou. LAGAN and multi-LAGAN: efficient tools for large-scale multiple alignment of genomic DNA. *Genome Res.*, 13:721–731, 2003.

[44] K. Bucka-Lassen, O. Caprani, and J. Hein. Combining many multiple alignments in one improved alignment. *Bioinformatics*, 15:122–130, 1999.

[45] L. Cai, D. Juedes, and E. Liakhovitch. Evolutionary computation techniques for multiple sequence alignment. In *Congress on Evolutionary Computation*, 2000.

[46] L. Cardon and G. Stormos. Expectation maximization algorithm for identifying protein-binding sites with variable lengths from unaligned DNA fragment. *Journal of Molecular Biology*, 223:159–170, 1992.

[47] H. Carrillo and D. Lipman. The multiple sequence alignment problem in biology. *SAIM J. Appl. Math.*, 48(5):1073–1082, 1988.

[48] S. Chan, A. Wong, and D. Chiu. A survey of multiple sequence comparison methods. *Bull. Math. Biol.*, 54:563–598, 1992.

[49] K. Chellapilla and G. Fogel. Multiple sequence alignment using evolutionary programming. In *Congress on Evolucionary Computation*, 1999.

[50] R. Chenna, H. Sugawara, T. Koike, R. Lopez, T. Gibson, D. Higgins, and J. Thompson. Multiple sequence alignment with the Clustal series of programs. *Nucleic Acids Res.*, 31:3497–3500, 2003.

[51] NetBeans Community. Netbeans, 4 2009. http://www.netbeans.org/.

[52] The UniProt Consortium. The Universal Protein Resource (UniProt). *Nucleic Acids Res.*, 36:D190–D195, 2008.

[53] F. Corpet. Multiple sequence alignment with hierarchical clustering. *Nucleic Acids Res.*, 16:10881–10890, 1988.

[54] 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.

[55] L. Davis. Adapting operator probabilities in genetic algorithms. In *ICGA*, pages 61–69, 1989.

[56] M. Dayhoff, R. Schwartz, and B. Orcutt. A model for evolutionary change in proteins. *Atlas of Protein Sequence and Structure*, 5(3):345–352, 1978.

[57] E. Depiereux, G. Baudoux, P. Briffeuil, I. Reginster, X. de Bolle, C. Vinals, and E. Feytmans. Match-Box server: a multiple sequence alignment tool placing emphasis on reliability. *Comput. Appl. Biosci.*, 13(3):249–256, 1997.

[58] J. Devereux, P. Haeberli, and O. Smithies. GCG package. *Nucleic Acids Res.*, 12:387–395, 1984.

[59] S. Dietmann, J. Park, C. Notredame, A. Heger, M. Lappe, and L. Holm. A fully automatic evolucionary classification of protein fold: DALI domain dictionary version 3. *Nucleic Acids Res.*, 29(1):55–57, 2001.

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

[61] C. Do, M. Mahabhashyam, M. Brudno, and S. Batzoglou. ProbCons Web Site, February 2009. http://probcons.stanford.edu.

[62] R. Doolittle. Similar amino acid sequences: chance and common ancestry? *Science*, 214:149–159, 1981.

[63] R. Doolittle. Convergent evolution: the need to be explicit. *Trends Biochem Sci.*, 19:15–18, 1994.

[64] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis*. Cambridge University Press, Cambridge, UK, 1998.

[65] 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.

[66] S. Eddy. Multiple alignment using hidden Markov models. In *Third International Conference on Intelligent Systems for Molecular Biology (ISMB)*, Cambridge, England, 1995.

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

[68] R. Edgar. MUSCLE: a multiple sequence alignment method with reduced time and space complexity. *Bioinformatics*, 5:113, 2004.

[69] R. Edgar. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Res.*, 32:1792–1797, 2004.

[70] I. Eidhammer, I. Jonassen, and W. Taylor. Structure comparison and structure patterns. *J. Comput. Biol.*, 7(5):685–716, 2000.

[71] T. Etzold and P. Argos. SRS - an indexing and retrieval tool for flat file data libraries. *Comput. Appl. Biosci.*, 9:49–57, 1993.

[72] J. Felsenstein. Confidence limits on phylogenies: an approach using the bootstrap. *Evolution*, 39:783–791, 1985.

[73] J. Felsenstein. PHYLIP home page, April 2011. http://evolution.genetics.washington.edu/phylip.html.

[74] D. Feng and R. Doolittle. Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *J. Mol. Biol.*, 25:351–360, 1987.

[75] R. Finn, J. Mistry, J. Tate, P. Coggill, A. Heger, J. Pollington, O. Gavin, P. Gunasekaran, G. Ceric, K. Forslund, L. Holm, E. Sonnhammer, S. Eddy, and A. Bateman. The Pfam protein families database. *Nucleic Acids Res*, 38:211–222, 2009.

[76] W. Fitch and K. Yasunobu. Phylogenies from amino acid sequences aligned with gaps: the problem of gap weighting. *J. Mol. Evol.*, 5:1–24, 1974.

[77] Eclipse Foundation. Eclipse, 4 2013. http://www.eclipse.org/.

[78] M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *J. Am. Stat. Assoc.*, 32:675–701, 1937.

[79] M. Frith, U. Hansen, J. Spouge, and Z. Weng. Finding functional sequence elements by multiple local alignment. *Nucleic Acids Res.*, 32:189–200, 2004.

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

[81] J. Garnier, J. Gibrat, and B. Robson. GOR method for predicting protein secondary structure from amino acid sequence. *Meth. Enzymol*, 266:540–553, 1996.

[82] R. Gentleman and R. Ihaka. The R project for statistical computing, 4 2009. http://www.r-project.org/.

[83] U. Goebel, C. Sander, R. Schneider, and A. Valencia. Correlated mutations and residue contacts in proteins. *Proteins: Structure Function and Genetics*, 18(4):309–317, 1994.

[84] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, New York, 1989.

[85] G. Gonnet, M. Cohen, and S. Benner. Exhaustive matching of the entire protein sequence database. *Science*, 256:1443–1445, 1992.

[86] G. Gonnet, M. Cohen, and S. Benner. Empirical and structural models for insertions and deletions in the divergent evolution of proteins. *Journal of Molecular Biology*, 229:1065–1082, 1993.

[87] G. Gonnet, C. Korostensky, and S. Benner. Evaluation measures of multiple sequence alignments. *J. Comput. Biol.*, 7:261–276, 2000.

[88] R. Gonzalez. Multiple protein sequence comparison by genetic algorithms. In *SPIE-98*, 1999.

[89] O. Gotoh. Consistency of optimal sequence alignments. *Bull Math Biol.*, 52(4):509–525, 1990.

[90] O. Gotoh. Optimal alignment between groups of sequences and its application to multiple sequence alignment. *Computer Applications in the Biosciences*, 9:361–370, 1993.

[91] O. Gotoh. Further improvement in methods of group-to-group sequence alignment with generalized profile operation. *CABIOS*, 10(4):379–387, 1994.

[92] O. Gotoh. A weighting system and algorithm for aligning many phylogenetically related sequences. *Computer Applications in the Biosciences*, 11(5):543–551, 1995.

[93] O. Gotoh. Significant improvement in accuracy of multiple protein sequence alignments by iterative refinement as assessed by reference to structural alignments. *J. Mol. Biol.*, 264:823–838, 1996.

[94] M. Gouy, C. Gautier, M. Attimonelli, C. Lanave, and G. di Paola. ACNUC - a portable retrieval system for nucleic acid sequence databases: logical and physical designs and usage. *Computer Applications in the Biosciences*, 1(3):167–172, 1985.

[95] J. Gracy and P. Argos. Automated protein sequence database classification. *Bioinformatics*, 14(2):164–173, 1998.

[96] C. Grasso and C. Lee. Combining partial order alignment and progressive multiple sequence alignment increases alignment speed and scalability to very large alignment problems. *Bioinformatics*, 20:1546–1556, 2004.

[97] M. Gribskov, R. Luethy, and D. Eisenberg. Profile analysis. *Meth. Enzymol.*, 183:146–159, 1990.

[98] M. Gribskov, A. McLachlan, and D. Eisenberg. Profile analysis: detection of distantly related proteins. *Proc. Natl. Acad. Sci.*, 84:4355–4358, 1987.

[99] S. Griffiths-Jones and A. Bateman. The use of structure information to increase alignment accuracy does not aid homologue detection with profile-HMMs. *Bioinformatics*, 18:1243–1249, 2002.

[100] X. Gu and W. Li. The size distribution of insertions and deletions in human and rodent pseudogenes suggests the logarithmic gap penalty for sequence alignment. *Journal of Molecular Evolution*, 40:464–473, 1995.

[101] S. Gupta, J. Kececioglu, and A. Schaffer. Improving the practical space and time efficiency of the shortest-paths approach to sum-of-pairs multiple sequence alignment. *J. Computational Biology*, 2:459–472, 1995.

[102] D. Gusfield. Efficient methods for multiple sequence alignment with guaranteed error bounds. *Bull Math Biol*, 55(1):141–154, 1993.

[103] R. Gutell, B. Weiser, C. Woese, and H. Noller. Comparative anatomy of 16S-like ribosomal RNA. *Prog. Nucleic Acid Res. Mol. Biol.*, 32:155–216, 1985.

[104] D. Haussler, A. Krogh, I. Mian, and K. Sjölander. Protein modeling using hidden Markov models: analysis of globins. In *Proceedings for the 26th Hawaii International Conference on Systems Sciences*, Wailea, 1993.

[105] S. Henikoff. Playing with blocks: some pitfalls of forcing multiple alignments. *The New Biologist*, 3(12):1148–1154, 1991.

[106] S. Henikoff and J. Henikoff. Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci.*, 89(22):10915–10919, 1992.

[107] S. Henikoff, J. Henikoff, W. Alford, and S. Pietrokovski. Automated construction and graphical presentation of protein blocks from unaligned sequences. *Gene-COMBIS, Gene*, 163:17–26, 1995.

[108] J. Heringa. Two strategies for sequence comparison: profile-preprocessed and secondary structure-induced multiple alignment. *Computers and Chemistry*, 23:341–364, 1999.

[109] J. Heringa. Local weighting schemes for protein multiple sequence alignment. *Comput. Chem.*, 5:459–477, 2002.

[110] J. Heringa and P. Argos. A method to recognize distant repeats in protein sequences. *Proteins*, 17(4):391–411, 1993.

[111] G. Hertz and G. Stormo. Identifying DNA and protein patterns with statistically significant alignments of multiple sequences. *Bioinformatics*, 15:563–577, 1999.

[112] S. Hess, J. Blake, and R. Blake. Wide variations in neighbor-dependent substitution rates. *J. Mol. Biol*, 236:1022–1033, 1994.

[113] D. Higgins, A. Bleasby, and R. Fuchs. Clustal V: improved software for multiple sequence alignment. *Computer Applications in the Biosciences*, 8(2):189–191, 1992.

[114] D. Higgins and P. Sharp. Clustal: a package for performing multiple sequence alignment on a microcomputer. *Gene*, 73(1):237–244, 1988.

[115] M. Hirosawa, M. Hoshida, M. Ishikawa, and T. Toya. MASCOT: multiple alignment system for protein sequences based on three-way dynamic programming. *Computer Applications in the Biosciences*, 9(2):161–167, 1993.

[116] P. Hogeweg and B. Hesper. The alignment of sets of sequences and the construction of phyletic trees: An integrated method. *J. Mol. Evol.*, 20:175–186, 1984.

[117] J. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Michigan, USA, 1975.

[118] R. Holland, T. Down, M. Pocock, A. Prlic, D. Huen, K. James, S. Foisy, A. Dräger, A. Yates, M. Heuer, and M. Schreiber. Biojava: an open-source framework for bioinformatics. *Bioinformatics*, 24(18):2096–2097, 2008.

[119] L. Holm and C. Sander. Mapping the protein universe. *Science*, 273:595–603, 1996.

[120] I. Holmes and W. Bruno. Evolutionary HMMs: a bayesian approach to multiple alignment. *Bioinformatics*, 9:803–820, 2001.

[121] I. Holmes and R. Durbin. Dynamic programming alignment accuracy. *J. Comput. Biol.*, 5:493–504, 1998.

[122] I. Holmes and G. Rubin. An expectation maximization algorithm for training hidden substitution models. *J. Mol. Biol.*, 5:753–764, 2002.

[123] X. Huang and W. Miller. A time-efficient linear-space local similarity algorithm. *Adv. Appl. Math.*, 12:337–357, 1991.

[124] M. Ishikawa, T. Toya, M. Hoshida, K. Nitta, A. Ogiwara, and M. Kanehisa. Multiple sequence alignment by parallel simulated annealing. *Computer Applications in the Biosciences*, 9(3):267–273, 1993.

[125] A. Jennings, C. Edge, and M. Sternberg. An approach to improving multiple alignments of protein sequences using predicted secondary structure. *Protein Eng.*, 14(4):227–231, 2001.

[126] J. Jiang and H. Jacob. EbEST: an automatic tool using expressed sequence tags to delineate gene structure. *Genome Research*, 8(3):268–275, 1998.

[127] T. Jiang, E. Lawler, and L. Wang. Aligning sequences via an evolutionary tree: complexity and approximation. In *ACM Sumpos. Theory Comput.*, volume 26, pages 760–769, 1994.

[128] I. Jonassen. *Bioinformatics: sequence, structure and databanks*, chapter Methods for discovering conserved patterns in protein sequences and structures. Oxford University Press, Oxford, UK, 2000.

[129] D. Jones. Protein secondary structure prediction based on position-specific scoring matrices. *J. Mol. Biol.*, 292(2):195–202, 1999.

[130] D. Jones, W. Taylor, and J. Thornton. The rapid generation of mutation data matrices from protein sequences. *CABIOS*, 8(3):275–282, 1992.

[131] W. Just. Computational complexity of multiple sequence alignment with SP-score. *J Comput Biol*, 8(6):615–623, 2001.

[132] W. Kabsch and C. Sander. Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers*, 12:2577–2637, 1983.

[133] S. Karlin, P. Bucher, V. Brendel, and S. Altschul. Statistical methods and insight for protein and DNA sequences. *Biophys. Chem.*, 20:175–203, 1991.

[134] K. Karplus and B. Hu. Evaluation of protein multiple alignments by SAM-T99 using the BAliBASE multiple alignment test set. *Bioinformatics*, 8:713–720, 2001.

[135] K. Katoh, K. Kuma, H. Toh, and T. Miyata. MAFFT version 5: improvement in accuracy of multiple sequence alignment. *Nucleic Acids Res*, 33(2):511–518, 2005.

[136] K. Katoh, K. Misasa, K. Kuma, and T. Miyata. MAFFT: a novel method for rapid multiple sequence alignment based on fast fourier transform. *Nucleic Acids Res.*, 30:3059–3066, 2002.

[137] J. Kececioglu. *Exact and approximation algorithms for DNA sequence reconstruction*. PhD thesis, University of Arizona, 1991.

[138] J. Kececioglu. The maximum weight trace problem in multiple sequence alignment. *Lecture Notes In Computer Science*, 684:106–119, 1993.

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

[140] M. Kimura. *The Neutral Theory of Molecular Evolution*. Cambridge University Press, 1983.

[141] K. Koretke, R. Russell, R. Copley, and A. Lupas. Fold recognition using sequence and secondary structure information. *Proteins*, 37:141–148, 1999.

[142] C. Kosiol and N. Goldman. Different versions of the Dayhoff rate matrix. *Molecular Biology and Evolution*, 22(2):193–199, 2005.

[143] A. Krogh, M. Brown, I. Mian, K. Sjlander, and D. Haussler. Hidden Markov models in computational biology. Applications to protein modeling. *J. Mol. Biol.*, 235(5):1501–1531, 1994.

[144] P. Lackner, W. Koppensteiner, M. Sippl, and F. Domingues. ProSup: a refined tool for protein structure alignment. *Protein Eng.*, 13:745–752, 2000.

[145] T. Lassmann and E. Sonnhammer. Quality assessment of multiple alignment programs. *FEBS Lett*, 529:126–130, 2002.

[146] T. Lassmann and E. Sonnhammer. Kalign - an accurate and fast multiple sequence alignment algorithm. *BMC bioinformatics*, 6:298, 2005.

[147] C. Lawrence, S. Altschul, M. Boguski, J. Liu, A. Neuwald, and J. Wootton. Detecting subtle sequence signals: A Gibbs Sampling strategy for multiple alignment. *Science*, 262:208–214, 1993.

[148] C. Lawrence and A. Reilly. An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences. *Proteins*, 7(1):41–51, 1990.

[149] C. Lee, C. Grasso, and M. Sharlow. Multiple sequence alignment using partial order graphs. *Bioinformatics*, 18:452–464, 2002.

[150] H. Lenhof, B. Morgenstern, and K. Reinert. An exact solution for the sequence-to-sequence multiple sequence alignment problem. *Bioinformatics*, 15(3):203–210, 1999.

[151] D. Lipman, S. Altschul, and J. Kececioglu. A tool for multiple sequence alignment. *Proc. Natl. Acad. Sci.*, 86:4412–4415, 1989.

[152] D. Lipman and W. Pearson. Rapid and sensitive protein similarity searches. *Science*, 227(4693):1435–1441, 1985.

[153] Y. Liu, B. Schmidt, and D. Maskell. MSAProbs: multiple sequence alignment based on pair hidden markov models and partition function posterior probabilities. *Bioinformatics*, 26(16):1958–1964, 2010.

[154] H. Lopes and G. Moritz. A graph-based genetic algorithm for the multiple sequence alignment problem. *Lecture Notes in Artificial Intelligence*, 4029:420–429, 2006.

[155] A. Lukashin, J. Engelbrecht, and S. Brunak. Multiple alignment using simulated annealing: branch point definition in human mRNA splicing. *Nucleic Acids Res.*, 20(10):2511–2516, 1992.

[156] R. Lüthy, I. Xenarios, and P. Bucher. Improving the sensitivity of the sequence profile method. *Protein Sci.*, 3:139–146, 1994.

[157] B. Marsden and R. Abagyan. SAD - a normalized structural alignment database: improving sequence-structure alignments. *Bioinformatics*, 15:2333–2344, 2004.

[158] M. McClure, T. Vasi, and W. Fitch. Comparative analysis of multiple protein-sequence alignment methods. *Mol. Biol. Evol.*, 11:571–592, 1994.

[159] E. McCreight. A space-economical suffix tree construction algorithm. *JACM*, 23(2):262–272, 1976.

[160] J. Meidanis. *Algorithms for problems in computacional genetics*. PhD thesis, University of Winsconsin-Madison, 1992.

[161] J. Meidanis and J. Setubal. Multiple alignment of biological sequences with gap flexibility. In R. Baeza-Yates and U. Manber, editors, *Proceedings of Second South American Workshop on String Processing*, pages 138–153, Valparaíso, Chile, 1995.

[162] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *J. Chem. Phys.*, 21:1087–1092, 1953.

[163] W. Miller and E. Myers. Sequence comparison with concave weighting functions. *Bull. Math. Biol.*, 50:97–120, 1988.

[164] K. Mizuguchi, C. Deane, T. Blundell, M. Johnson, and J. Overington. Joy: protein sequence-structure representation and analysis. *Bioinformatics*, 14:617–623, 1998.

[165] K. Mizuguchi, C. Deane, T. Blundell, and J. Overington. HOMSTRAD: a database of protein structure alignments for homologous families. *Protein Sci.*, 7:2469–2471, 1998.

[166] B. Morgenstern. DiAlign 2: improvement of the segment-to-segment approach to multiple sequence alignment. *Bioinformatics*, 15:211–218, 1999.

[167] B. Morgenstern. DiAlign: multiple DNA and protein sequence alignment at BiBiServ. *Nucleic Acids Res.*, 32:33–36, 2004.

[168] B. Morgenstern, A. Dress, and T. Wener. Multiple DNA and protein sequence alignment based on segment-to-segment comparison. *Proc. Natl. Acad. Sci.*, 93:12098–12103, October 1996.

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

[170] M. Murata, J. Richardson, and J. Sussman. Simultaneous comparison of three protein sequences. *Proc. Natl. Acad. Sci.*, 82:3073–3077, 1985.

[171] A. Murzin, S. Brenner, T. Hubbard, and C. Chothia. SCOP: a structural classification of proteins database for the investigation of sequences and structures. *J. Mol. Biol.*, 247:536–540, 1995.

[172] E. Myers and W. Miller. Optimal alignments in linear-space. *Comput. Appl. Biosci.*, 4:11–17, 1988.

[173] E. Myers and W. Miller. Chaining multiple-alignment fragments in sub-quadratic time. In *SODA*, pages 38–47, 1995.

[174] S. Needleman and C. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.*, 48:443–453, 1970.

[175] M. Nei and S. Kumar. *Molecular Evolution and Phylogenetics*. Oxford University Press, New York, 2000.

[176] M. Nei, T. Maruyama, and C. Wu. Models of evolution of reproductive isolation. *Genetics*, 103:557–579, 1983.

[177] A. Neuwald, J. Liu, D. Lipman, and C. Lawrence. Extracting protein alignment models from the sequence database. *Nucleic Acid Research*, 25:1665–1677, 1997.

[178] H. Nguyen, I. Yoshihara, K. Yamamori, and M. Yasunaga. Aligning multiple protein sequences by parallel hybrid genetic algorithm. *Genome Inform Ser*, 13:123–132, 2002.

[179] C. Notredame. Mocca: semi-automatic method for domain hunting. *Bioinformatics*, 17(4):373–374, 2001.

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

[181] C. Notredame. Recent evolutions of multiple sequence alignment algorithms. *PLoS Computational Biology*, 3(8):1405–1408, 2007.

[182] C. Notredame and D. Higgins. SAGA: sequence alignment by genetic algorithm. *Nucl. Acid. Res.*, 24(8):1515–1524, 1996.

[183] C. Notredame, D. Higgins, and J. Heringa. T-COFFEE: A novel method for fast and accurate multiple sequence alignment. *J Mol Biol*, 302(1):205–217, 2000.

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

[185] C. Notredame, E. O'Brien, and D. Higgins. RAGA: RNA sequence alignment by genetic algorithm. *Nucleic Acids Res.*, 25(22):4570–4580, 1997.

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

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

[188] C. Orengo and W. Taylor. A rapid method of protein structure alignment. *J. Theor. Biol.*, 147:517–551, 1990.

[189] O. O'Sullivan, K. Suhre, C. Abergel, D. Higgins, and C. Notredame. 3D-COFFEE: combining protein sequences and structures within multiple sequence alignments. *J. Mol. Biol.*, 340:385–395, 2004.

[190] O. O'Sullivan, M. Zehnder, D. Higgins, P. Bucher, A. Grosdidier, and et al. APDB: a novel measure for benchmarking sequence alignment methods without reference alignments. *Bioinformatics*, 19:1215–1221, 2003.

[191] J. Overington, D. Donnelly, M. Johnson, A. Sali, and T. Blundell. Environment-specific amino acid substitution tables: Tertiary templates and prediction of protein folds. *Protein Science*, 2:216–226, 1992.

[192] E. Paradis, J. Claude, and K. Strimmer. APE: analyses of phylogenetics and evolution in R language. *Bioinformatics*, 20:289–290, 2004.

[193] S. Pascarella and P. Argos. Analysis of insertions/deletions in protein structures. *J. Mol. Biol.*, 224:461–471, 1992.

[194] L. Patthy. Exon shuffling and other ways of module exchange. *Matrix Biol*, 15:301–310, 1996.

[195] W. Pearson and D. Lipman. Improved tools for biological sequence comparison. *Proc Natl Acad Sci USA*, 85(8):2444–2448, 1988.

[196] J. Pei and N. Grishin. MUMMALS: multiple sequence alignment improved by using hidden Markov models with local structural information. *Nucleic Acids Res*, 34:4364–4374, 2006.

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

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

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

[200] P. Pevzner. Multiple alignment, communication cost, and graph matching. *SIAM J. Appl. Math.*, 52(6):1763–1779, 1992.

[201] F. Plewniak, J. Thompson, and O. Poch. Ballast: BLAST post-processing based on locally conserved segments. *Bioinformatics*, 16(92000):750–759, 2000.

[202] G. Raghava, S. Searle, P. Audley, J. Barber, and G. Barton. OXBench: a benchmark for evaluation of protein multiple sequence alignment accuracy. *BMC Bioinformatics*, 4(47), 2003.

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

[204] K. Reinert, J. Stoye, and T. Will. An iterative method for faster sum-of-pair multiple sequence alignment. *Bioinformatics*, 16(9):808–814, 2000.

[205] U. Roshan and D. Livesay. Probalign: multiple sequence alignment using partition function posterior probabilities. *Bioinformatics*, 22(22):2715–2721, 2006.

[206] B. Rost. Review: protein secondary structure prediction continues to rise. *J. Struct. Biol.*, 134:204–218, 2001.

[207] B. Rost, C. Sander, and R. Schneider. PHD - An automatic server for protein secundary structure prediction. *CABIOS*, 10:53–60, 1994.

[208] R. Rughey and A. Krogh. Hidden Markov models for sequence analysis: extension and analysis of the basic method. *Computer Application in Biological Science*, 12:95–107, 1996.

[209] N. Saitou. Maximum likelihood methods. *Meth. Enzymol.*, 183:584–598, 1990.

[210] N. Saitou and M. Nei. The Neighbor Joining method: A new method for reconstructing phylogenetic trees. *Mol. Biol. Evol.*, 4:406–425, 1987.

[211] C. Sander and R. Schneider. Database of homology derived protein structures and the structural meaning of sequence alignment. *Proteins*, 9:56–58, 1991.

[212] D. Sankoff and J. Kruskal, editors. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*. Addison-Wesley, Reading, MA, 1983.

[213] D. Santos. Alinhamento múltiplo de proteínas via algoritmo genético baseado em tipos abstratos de dados. Master's thesis, Instituto de Computação - Universidade Federal de Alagoas, 2008.

[214] A. Schaffer, L. Aravind, T. Madden, S. Shavirin, J. Spouge, Y. Wolf, E. Koonin, and S. Altschul. Improving the accuracy of PSI-BLAST protein database searches with composition-based statistics and other refinements. *Nucleic Acids Res.*, 14:2994–3005, 2001.

[215] G. Schuler, S. Altschul, and D. Lipman. A workbench for multiple alignment construction and analysis. *Proteins*, 9(3):180–190, 1991.

[216] G. Schuler, J. Epstein, H. Ohkawa, and J. Kans. Entrez: molecular biology database and retrieval system. *Methods Enzymol*, 266:141–162, 1996.

[217] R. Schwartz and M. Dayhoff. Matrices for detecting distant relationships. *Atlas of Protein Sequence and Structure*, 5:353–358, 1978. suppl. 3.

[218] P. Sellars. On the theory and computation of evolutionary distances. *SIAM J. Appl. Math.*, 26:787–793, 1974.

[219] J. Setubal and J. Meidanis. *Introduction to Computational Molecular Biology*. PWS, 1997.

[220] J. Shi, T. Blundell, and K. Mizuguchi. FUGUE: sequence-structure homology recognition using environment-specific substitution tables and structure-dependent gap penalties. *J. Mol. Biol.*, 310:243–257, 2001.

[221] F. Sievers, A. Wilm, D. Dineen, T. Gibson, K. Karplus, W. Li, R. Lopez, H. McWilliam, M. Hemmert, J. Söding, J. Thompson, and D. Higgins. Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega. *Molecular Systems Biology*, 7, 2011.

[222] F. Silva, M. Souza, A. Almeida, and Z. Dias. A method to evaluate assemblers that make use of short and long transcript reads simultaneously. In

X-Meeting Eletronic Abstracts Book 2009, Angra dos Reis, RJ, 2009. AB3C. http://lgmb.fmrp.usp.br/xmeeting2009/abstractbook/.

[223] V. Simossis. and J. Heringa. PRALINE: a multiple sequence alignment toolbox that integrates homology-extended and secondary structure information. *Nucleic Acids Res*, 33:289–294, 2005.

[224] S. Sinha, M. Blanchette, and M. Tompa. PhyME: a probabilistic algorithm for finding motifs in sets of orthologous sequences. *BMC Bioinformatics*, 5:170, 2004.

[225] R. Smith and T. Smith. Pattern-Induced Multi-sequence Alignment (PIMA) algorithm employing secondary structure-dependent gap penalties for comparative protein modelling. *Protein Engng*, 5:35–41, 1992.

[226] T. Smith and M. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[227] T. Smith, M. Waterman, and W. Fitch. Comparative biosequence metrics. *J. Mol. Evol.*, 18:38–46, 1981.

[228] P. Sneath and R. Sokal. *Numerical Taxonomy*. Freeman, San Francisco, 1973.

[229] E. Sobel and H. Martinez. A multiple sequence alignment program. *Nucleic Acids Research*, 14(1):363–374, 1986.

[230] R. Sokal and C. Michener. A statistical method for evaluating systematic relationships. *The University of Kansas Scientific Bulletin*, 38:1409–1438, 1958.

[231] M. Souza, A. Almeida, A. Andrade, F. Silva, and Z. Dias. Best of both words: Using short and long reads to construct UniGenes. In *Plant & Animal Genomes XVII Conference*, San Diego, CA, EUA, 2009.

[232] G. Srinivasarao, L. Yeh, C. Marzec, B. Orcutt, and W. Barker. Database of protein sequence alignments: PIR-ALN. *Bioinformatics*, 15:382–390, 1999.

[233] G. Stoesser, M. Tuli, R. Lopez, and P. Sterk. The EMBL nucleotide sequence database. *Nucleic Acids Research*, 27:18–24, 1999.

[234] J. Stoye, D. Evers, and F. Meyer. ROSE: generating sequence families. *Bioinformatics*, 14:157–163, 1998.

[235] J. Stoye, V. Moulton, and A. Dress. DCA: an efficient implementation of the divide-and-conquer approach to simultaneous multiple sequence alignment. *Comput. Appl. Biosci.*, 13(6):625–626, 1997.

[236] A. Subramanian, J. Weyer-Menkhoff, M. Kaufmann, and B. Morgenstern. DiAlign-T: an improved algorithm for segment-based multiple sequence alignment. *BMC Bioinformatics*, 6:66, 2005.

[237] N. Takezaki and M. Nei. Genetic distances and reconstruction of phylogenetic trees from microsatellite dna. *Genetics*, 144:389–399, 1996.

[238] W. Taylor. Identification of protein sequence homology by consensus template alignment. *J Mol Biol*, 188(2):233–258, 1986.

[239] W. Taylor. A flexible method to align a large number of sequences. *J. Mol. Evol.*, 28:161–169, 1988.

[240] W. Taylor. Dynamic sequence databank searching with templates and multiple alignments. *J. Mol. Biol.*, 280:375–406, 1998.

[241] W. Taylor. Protein structure comparison using SAP. *Methods Mol. Biol.*, 143:19–32, 2000.

[242] W. Taylor, G. Saelensminde, and I. Eidhammer. Multiple protein sequence alignment using double-dynamic programming. *Comput. Chem.*, 24(1):3–12, 2000.

[243] J. Thompson. Introducing variable gap penalties to sequence alignment in linear space. *Computer Applications in the Biosciences*, 11(2):181–186, 1995.

[244] J. Thompson, T. Gibson, F. Plewniak, F. Jeanmougin, and D. Higgins. The Clustal X window interface: flexible strategies for multiple sequence alignment aided by quality analysis tools. *Nucleic Acids Res.*, 24:4876–4882, 1997.

[245] J. Thompson, D. Higgins, and T. Gibson. Clustal W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, positions-specific gap penalties and weight matrix choice. *Nucl. Acid. Res.*, 22:4673–4680, 1994.

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

[247] J. Thompson, P. Koehl, R. Ripp, and O. Poch. BAliBASE 3.0 Web Site, February 2009. http://www-bio3d-igbmc.u-strasbg.fr/balibase/.

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

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

[250] J. Thompson, F. Plewniak, R. Ripp, J. Thierry, and O. Poch. Towards a reliable objective function for multiple sequence alignments. *J. Mol. Biol.*, 314(4):937–951, 2001.

[251] J. Thompson, F. Plewniak, J. Thierry, and O. Poch. DbClustal: rapid and reliable global multiple alignments of protein sequences detected by database searches. *Nucleic Acids Res*, 28(15):2919–2926, 2000.

[252] N. van Ohsen and R. Zimmer. Improving profile-profile alignments via log average scoring. In *WABI*, pages 11–26, 2001.

[253] S. Veerassamy, A. Smith, and E. Tillier. A transition probability model for amino acid substitutions from blocks. *J. Comput. Biol.*, 10(6):997–1010, 2003.

[254] R. Vieira. *Um Algoritmo Genético Baseado em Tipos Abstratos de Dados e sua Especificação em Z.* PhD thesis, Universidade Federal de Pernambuco, 2003.

[255] M. Vingron and P. Argos. A fast and sensitive multiple sequence alignment algorithm. *Comput. Appl. Biosci.*, 5:115–121, 1989.

[256] M. Vingron and P. Argos. Motif recognition and alignment for many sequences by comparison of dot-matrices. *J. Mol. Biol.*, 218:33–43, 1991.

[257] M. Vingron and M. Waterman. Sequence alignment and penalty choice: review of concepts, case studies and implications. *J. Mol. Biol.*, 235:1–12, 1994.

[258] A. Viterbi. Error bounds for computacional codes and an symptotically optimal decoding algorithm. *IEEE Trans. Inf. Theory*, IT-13:260–269, 1967.

[259] I. Wallace, G. Blackshields, and D. Higgins. Multiple sequence alignments. *Curr. Opin. Struct. Biol.*, 15:261–266, 2005.

[260] I. Wallace, O. O'Sullivan, D. Higgins, and C. Notredame. M-COFFEE: combining multiple sequence alignment methods with T-COFFEE. *Nucleic Acids Res*, 34:1692–1699, 2006.

[261] I. Walle, I. Lasters, and L. Wyns. SABmark - a benchmark for sequence alignment that covers the entire known fold space. *Bioinformatics*, 21(7):1267–1268, 2005.

[262] I. Van Walle, I. Lasters, and L. Wyns. Align-m: a new algorithm for multiple alignment of highly divergent sequences. *Bioinformatics*, 20:1428–1435, 2004.

[263] L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *J. Comput. Biol.*, 1(4):337–348, 1994.

[264] Y. Wang and K. Li. An adaptive and iterative algorithm for refining multiple sequence alignment. *Comput. Biol. Chem.*, 2:141–148, 2004.

[265] M. Waterman and R. Jones. Consensus methods for DNA and protein sequence alignment. *Meth. Enzym.*, 183:221–236, 1990.

[266] M. Waterman, T. Smith, and W. Bayer. Some biological sequence metrics. *Adv. Math.*, 20:267–287, 1976.

[267] F. Wilcoxon. Probability tables for individual comparisons by ranking methods. *Biometrics*, 3:119–122, 1947.

[268] J. Wootton and S. Federhen. Statistics of local complexity in amino acid sequences and sequence databases. *Computers in Chemistry*, 17:149–163, 1993.

[269] C. Zhang and A. Wong. A genetic algorithm for multiple molecular sequence alignment. *Comput. Appl. Biosci.*, 13(6):565–581, 1997.

[270] Y. Zhang and M. Waterman. An Eulerian path approach to local multiple alignment for DNA sequences. *Proc. Natl. Acad. Sci. USA*, 102:1285–1290, 2005.

[271] Z. Zhang, B. He, and W. Miller. Local multiple alignment via subgraph enumeration. *Discrete Appl. Math.*, 71:337–365, 1996.

[272] Z. Zhang, B. Raghavachari, R. Hardison, and W. Miller. Chaining multiple-aligned blocks. *Journal of Computational Biology*, 1:217–226, 1994.

[273] H. Zhou and Y. Zhou. SPEM: improving multiple sequence alignment with sequence profiles and predicted secondary structures. *Bioinformatics*, 21(18):3615–3621, 2005.

Apêndice A

Glossário

3D-COFFEE: é um alinhador múltiplo baseado em modelos. Para maiores informações, consulte a Seção 2.3.4.

AB1: é um alinhador baseado em blocos desenvolvido, ao longo deste trabalho, para gerar indivíduos para a população inicial do ALGAe. Para maiores informações, consulte a Seção 5.2.8.

AB2: é um alinhador baseado em blocos desenvolvido, ao longo deste trabalho, para gerar indivíduos para a população inicial do ALGAe. Para maiores informações, consulte a Seção 5.2.8.

AB3: é um alinhador baseado em blocos desenvolvido, ao longo deste trabalho, para gerar indivíduos para a população inicial do ALGAe. Para maiores informações, consulte a Seção 5.2.8.

AB4: é um alinhador baseado em blocos desenvolvido, ao longo deste trabalho, para gerar indivíduos para a população inicial do ALGAe. Para maiores informações, consulte a Seção 5.2.8.

AB5: é um alinhador baseado em blocos desenvolvido, ao longo deste trabalho, para gerar indivíduos para a população inicial do ALGAe. Para maiores informações, consulte a Seção 5.2.8.

*Ab initio***:** é um termo latino que significa desde o princípio.

ABOT: é um alinhador baseado em blocos desenvolvido, ao longo deste trabalho, para gerar indivíduos para a população inicial do ALGAe. Para maiores informações, consulte a Seção 5.2.8.

AC: refere-se a alinhamento de consensos global, que é um método para agrupamento de alinhamentos desenvolvido ao longo deste trabalho. Para maiores informações, consulte a Seção 3.3.2.

ACb: refere-se a alinhamento de consensos semi-global, que é um método para agrupamento de alinhamentos desenvolvido ao longo deste trabalho. Para maiores informações, consulte a Seção 3.3.2.

ACLog: refere-se a alinhamento de consensos global com função logarítmica para penalização de *gaps*, que é um método para agrupamento de alinhamentos desenvolvido ao longo deste trabalho. Para maiores informações, consulte a Seção 3.3.2.

ACLogb: refere-se a alinhamento de consensos semi-global com função logarítmica para penalização de *gaps*, que é um método para agrupamento de alinhamentos desenvolvido ao longo deste trabalho. Para maiores informações, consulte a Seção 3.3.2.

Agrupamento de alinhamentos: consiste em combinar dois alinhamentos de forma que, se o primeiro alinhamento tem n sequências e o segundo m, o alinhamento resultante terá $n + m$ sequências. Tal procedimento é empregado em alinhamentos progressivos. Para maiores informações, consulte a Seção 3.3.2.

ALGAe: é um ambiente parametrizável para execução de algoritmos genéticos voltados para alinhamento múltiplo. Foi desenvolvido ao longo deste trabalho. Para maiores informações, consulte a Seção 5.2.

Algoritmo 1: é um método para geração de população inicial, desenvolvido ao longo deste trabalho e utilizado pelo ALGAe. Para maiores informações, consulte a Seção 5.2.8.

Algoritmo 2: é um método para geração de população inicial, desenvolvido ao longo deste trabalho e utilizado pelo ALGAe. Para maiores informações, consulte a Seção 5.2.8.

Algoritmo 3: é um método para geração de população inicial, desenvolvido ao longo deste trabalho e utilizado pelo ALGAe. Para maiores informações, consulte a Seção 5.2.8.

Algoritmo de Needleman e Wunsch: algoritmo exato para alinhamento global de par de sequências, que utiliza programação dinâmica. Sejam m e n os comprimentos das sequências. O algoritmo tem complexidade $O(mn)$ para tempo e espaço. Para maiores informações, consulte a Seção 2.2.

Algoritmo de Smith e Waterman: algoritmo exato para alinhamento local de par de sequências. É uma adaptação no algoritmo de Needleman e Wunsch. Para maiores informações, consulte a Seção 2.2.

Alinhamento de consensos: é um método comumente empregado para agrupamento de alinhamentos na abordagem progressiva. Para maiores informações, consulte a Seção 3.3.2.

Alinhamento de perfis: é um método comumente empregado para agrupamento de alinhamentos na abordagem progressiva. Para maiores informações, consulte a Seção 3.3.2.

Alinhamento estrela: consiste em construir um alinhamento múltiplo usando uma sequência como âncora para adicionar as outras ao alinhamento.

Alinhamento global: consiste em alinhar as sequências por completo.

Alinhamento local: consiste em alinhar apenas a região de maior similaridade entre as sequências.

Alinhamento semi-global: consiste em alinhar as sequências por completo sem penalizar os *gaps* nos extremos das sequências. É utilizado, por exemplo, em montagem de fragmentos.

AMPS: é um alinhador múltiplo iterativo. Consulte a Seção 2.3.2 para maiores informações.

AP: refere-se a alinhamento de perfis global, que é um método para agrupamento de alinhamentos desenvolvido ao longo deste trabalho. Para maiores informações, consulte a Seção 3.3.2.

APA: refere-se a alinhamento de perfis global com função afim para penalização de *gaps*, que é um método para agrupamento de alinhamentos desenvolvido ao longo deste trabalho. Para maiores informações, consulte a Seção 3.3.2.

APAb: refere-se a alinhamento de perfis semi-global com função afim para penalização de *gaps*, que é um método para agrupamento de alinhamentos desenvolvido ao longo deste trabalho. Para maiores informações, consulte a Seção 3.3.2.

APAp: refere-se a alinhamento de perfis com ajuste automático de parâmetros, que é um método para agrupamento de alinhamentos desenvolvido ao longo deste trabalho. Para maiores informações, consulte a Seção 3.3.2.

APb: refere-se a alinhamento de perfis semi-global, que é um método para agrupamento de alinhamentos desenvolvido ao longo deste trabalho. Para maiores informações, consulte a Seção 3.3.2.

APDB: é um ferramenta para avaliação de alinhamentos múltiplos. Para maiores informações, consulte a Seção 2.4.

APE: acrônimo de *Analysis of Phylogenetics and Evolution*, é uma biblioteca da ferramenta `R`, que oferece uma série de funcionalidades para estudos em filogenia e evolução. NJ é uma das funções que implementa e foi utilizada nos alinhadores progressivos desenvolvidos ao longo deste trabalho. Para maiores informações, consulte o artigo que descreve a biblioteca [192].

APLog: refere-se a alinhamento de perfis global com função logarítmica para penalização de *gaps*, que é um método para agrupamento de alinhamentos desenvolvido ao longo deste trabalho. Para maiores informações, consulte a Seção 3.3.2.

APLogb: refere-se a alinhamento de perfis semi-global com função logarítmica para penalização de *gaps*, que é um método para agrupamento de alinhamentos desenvolvido ao longo deste trabalho. Para maiores informações, consulte a Seção 3.3.2.

Árvore guia: é uma árvore utilizada para guiar os alinhamentos de pares na abordagem progressiva. Para maiores informações, consulte a Seção 3.2.

ASSEMBLE: é um alinhador múltiplo baseado em blocos. Para maiores informações, consulte a Seção 2.3.4.

BAliBASE: é uma ferramenta para avaliação de alinhamentos múltiplos. Para maiores informações, consulte a Seção 2.4.

BLAST: acrônimo de *Basic Local Alignment Search Tool*, é um alinhador heurístico local para par de sequências. Para maiores informações, consulte o artigo que o descreve [12].

BLOSUM: é uma série de matrizes de substituição de aminoácidos. Para maiores informações, consulte o artigo que a descreve [106].

BU: refere-se a seleção por bloco único.

*Change Gap Block Mutation***:** é um operador de mutação, utilizado no ALGAe e desenvolvido ao longo deste trabalho. Para maiores informações, consulte a Seção 5.2.3.

Clustal W: é um alinhador múltiplo progressivo. Para maiores informações, consulte a Seção 2.3.1.

COFFEE: é uma função objetivo baseada em consistência. Para maiores informações, consulte a Seção 2.3.3.

Core block: é um região que pode ser confiavelmente alinhada em um alinhamento múltiplo.

Dayhoff(6): é um alfabeto comprimido. Para consultar as classes que define, ver a Tabela 4.1. Para maiores informações sobre alfabetos comprimidos, consulte o artigo de Robert C. Edgar [67].

DbClustal: é um alinhador múltiplo baseado em modelos. Para maiores informações, consulte a Seção 2.3.4.

DCA: é um alinhador múltiplo. Para maiores informações, consulte a Seção 2.3.

Distância com função logarítmica para penalização de *gaps*: é um método para computação de matriz de distâncias desenvolvido ao longo deste trabalho. Para maiores informações, consulte a Seção 3.1.6.

Distância local recursiva: é um método para computação de matriz de distâncias desenvolvido ao longo deste trabalho. Consulte a Seção 3.1.5 para maiores informações.

Esquema de pesos: é um recurso que pode ser utilizado em alinhamentos de perfis para melhorar a qualidade dos resultados quando a entrada possui sequências não equidistantes. Para maiores informações, consulte a Seção 3.3.3.

Expresso: é um alinhador múltiplo baseado em modelos. Para maiores informações, consulte a Seção 2.3.4.

Extensão estrutural: é uma das formas de implementar alinhamento baseado em modelos. Para maiores informações, consulte a Seção 2.3.4.

Extensão por homologia: é uma das formas de implementar alinhamento baseado em modelos. Para maiores informações, consulte a Seção 2.3.4.

FASTA: (1) é um alinhador heurístico para par de sequências. Para maiores informações, consulte o artigo que o descreve [152]. (2) É um formato de arquivo que pode ser utilizado como entrada para alinhadores múltiplos. Pode conter diversas sequências de nucleotídeos ou aminoácidos e para cada uma delas é possível associar um cabeçalho com identificador para a sequência e, eventualmente, outras informações. Para maiores informações, consulte a Seção 2.4.1.

FUGUE: ferramenta para alinhamento entre uma sequência e uma estrutura. Para maiores informações, consulte o artigo que o descreve [220].

GA: é um acrônimo de *Genetic Algorithm*.

Gapped BLAST: refere-se a versão 2.0 do BLAST. Como o nome sugere, passou a permitir *gaps* no alinhamento local. Para maiores informações, consulte o artigo que o descreve [14].

Gibbs Sampler: é um alinhador múltiplo local. Para maiores informações, consulte a Seção 2.3.2.

HMM: é um acrônimo de *Hidden Markov Model*.

HMMER: é um alinhador múltiplo que faz uso de HMMs. Para maiores informações, consulte o artigo que o descreve [143].

HOMSTRAD: é uma ferramenta para avaliação de alinhamentos múltiplos. Para maiores informações, consulte a Seção 2.4.

Indel: indica um evento de mutação do tipo inserção ou deleção.

IRMbase: é uma ferramenta para avaliação de alinhamentos múltiplos. Para maiores informações, consulte a Seção 2.4.

IterAlign: é um alinhador múltiplo iterativo. Para maiores informações, consulte a Seção 2.3.2.

JOY: é um banco de dados de alinhamentos estruturais que pode ser utilizado para a avaliação de alinhamentos múltiplos. Para maiores informações, consulte o artigo que o descreve [164].

JTT: é um método para computação de matriz de distâncias implementado na ferramenta PHYLIP. Para maiores informações, consulte a Seção 3.1.2.

Kalign: é um alinhador múltiplo progressivo. Para maiores informações, consulte a Seção 2.3.1.

LC: refere-se a alinhamento de consensos recursivamente construído através de alinhamentos locais, que é um método para agrupamento de alinhamentos desenvolvido ao longo deste trabalho. Para maiores informações, consulte a Seção 3.3.2.

LD: refere-se a distância local recursiva.

Li-A(10): é um alfabeto comprimido. Para consultar as classes que define, ver a Tabela 4.1. Para maiores informações sobre alfabetos comprimidos, consulte o artigo de Robert C. Edgar [67].

Li-B(10): é um alfabeto comprimido. Para consultar as classes que define, ver a Tabela 4.1. Para maiores informações sobre alfabetos comprimidos, consulte o artigo de Robert C. Edgar [67].

LOGD: Refere-se a distância com função logarítmica para penalização de *gaps*.

MACAW: é um alinhador múltiplo local. Para maiores informações, consulte o artigo que o descreve [215].

MAFFT: é um alinhador múltiplo progressivo. Para maiores informações, consulte a Seção 2.3.1.

Match: em um alinhamento, um *match* ocorre quando os resíduos de um par, em uma dada coluna, coincidem.

Match-Box: é um alinhador múltiplo local. Para maiores informações, consulte o artigo que o descreve [57].

Matriz de distâncias: é uma matriz que indica as distâncias entre cada par de sequências. Para maiores informações, consulte a Seção 3.1.

M-COFFEE: é um alinhador múltiplo baseado em consenso. Para maiores informações, consulte a Seção 2.3.4.

Método da roleta: é um método utilizado para seleção de indivíduos em um GA. Objetiva dar chances a todos os indivíduos, mas os indivíduos mais adaptados têm uma maior probabilidade de serem selecionados.

Mismatch: em um alinhamento, um *mismatch* ocorre quando os resíduos de um par, em uma dada coluna, não coincidem.

Mocca: é um alinhador múltiplo local. Para maiores informações, consulte o artigo que o descreve [179].

Modelo das categorias: é um método para computação de matriz de distâncias implementado na ferramenta PHYLIP. Para maiores informações, consulte a Seção 3.1.4.

Montagem de fragmentos: sequências de DNA comumente possuem milhões de pares de bases. As tecnologias atuais de sequenciamento não lêem mais que 1.000 pares

de base em uma dada molécula. Montagem de fragmentos consiste em, dado um conjunto de fragmentos, reconstruir a sequência original com base em sobreposições de fragmentos.

MSA: (1) é uma acrônimo de *Multiple Sequence Alignment*. (2) É um alinhador múltiplo exato. Para maiores informações, consulte a Seção 2.3.

MSF: é um formato de arquivo para armazenar um alinhamento múltiplo. Para maiores informações, consulte a Seção 2.4.1.

MultiAlign: é um alinhador múltiplo progressivo. Para maiores informações, consulte a Seção 2.3.1.

MUMMALS: é um alinhador múltiplo baseado em consistência que faz uso da abordagem progressiva também. Para maiores informações, consulte o Capítulo 4.

Murphy(10): é um alfabeto comprimido. Para consultar as classes que define, ver a Tabela 4.1. Para maiores informações sobre alfabetos comprimidos, consulte o artigo de Robert C. Edgar [67].

MUSCLE: é um alinhador múltiplo progressivo. Para maiores informações, consulte a Seção 2.3.1.

NCBI: é parte da Biblioteca Nacional de Medicina dos EUA. Mantém uma série de bases de dados relevantes para biotecnologia e biomedicina, tais como: GenBank e PubMed. O primeiro para sequências de DNA e segundo para referências bibliográficas na área biomédica.

*Neighbor Joining***:** é um método para construção de árvores filogenéticas. Neste trabalho, as árvores geradas foram utilizadas como árvores guias na construção de alinhamentos através da abordagem progressiva. Para maiores informações, consulte a Seção 3.2.2.

NJ: é um acrônimo de *Neighbor Joining*.

NP: refere-se a seleção do par mais próximo.

OF: é um acrônimo de *Objective Function*.

OMA: é um alinhador múltiplo. Para maiores informações, consulte a Seção 2.3.

PAM: é um método para computação de matriz de distâncias implementado na ferramenta PHYLIP. Para maiores informações, consulte a Seção 3.1.1.

PCM: refere-se a modelo das categoria.

PCMA: é um alinhador múltiplo baseado em consistência que faz uso da abordagem progressiva também. Para maiores informações, consulte o artigo que o descreve [199].

PDB: é um acrônimo de *Protein Data Bank*.

PHYLIP: é um pacote de programas para inferir filogenias. Para maior informações, consulte seu *website* [73].

Pileup: é um alinhador múltiplo progressivo. Para maiores informações, consulte a Seção 2.3.1.

PM: indica a presença do esquema de pesos em um alinhamento de perfis. Para maior informações, consulte a Seção 3.3.3.

PMB: é um método para computação de matriz de distâncias implementado na ferramenta PHYLIP. Para maiores informações, consulte a Seção 3.1.3.

PP: refere-se a peso padrão. É o mesmo que desativar o esquema de pesos no alinhamento de perfis. Para maiores informações, consulte a Seção 3.3.3.

PRALINE: é um alinhador múltiplo baseado em modelos. Para maiores detalhes, consulte a Seção 2.3.4.

PREFAB: é uma ferramenta para avaliação de alinhamentos múltiplos. Para maiores informações, consulte a Seção 2.4.

ProbCons: é um alinhador múltiplo baseado em consistência que faz uso da abordagem progressiva também. Para maiores informações, consulte a Seção 2.3.3.

PROMALS: é um alinhador múltiplo baseado em modelos. Para maiores informações, consulte a Seção 2.3.4.

PROMALS3D: é um alinhador múltiplo baseado em modelos. Para maiores informações, consulte o artigo que o descreve [198].

Protein Data Bank: é um repositório para dados estruturais 3D de grandes moléculas biológicas, tais como proteínas e ácidos nucleicos. Para maiores informações, consulte o artigo que o descreve [37].

PRRP: é um alinhador múltiplo iterativo. Consulte a Seção 2.3.2 para maiores informações.

PSI-BLAST: é uma ferramenta para busca por sequências homólogas. Para maiores informações, consulte o artigo que o descreve [14].

PSIPRED: é uma ferramenta para predição de estrutura secundária. Para maiores informações, consulte o artigo que o descreve [129].

Q-score: é o mesmo que SP.

R: é um programa, de distribuição livre, para computação estatística e produção de gráficos. Implementa UPGMA em suas bibliotecas padrões e NJ na biblioteca APE. Para maiores informações, consulte o seu *website* [82].

R11: é um dos alinhadores iterativos não estocásticos desenvolvidos ao longo deste trabalho. Para maiores informações, consulte a Seção 5.1.

R12: é um dos alinhadores iterativos não estocásticos desenvolvidos ao longo deste trabalho. Para maiores informações, consulte a Seção 5.1.

Repro: é um alinhador múltiplo local. Para maiores informações, consulte o artigo que o descreve [110].

RV11: é um dos subconjuntos do primeiro conjunto de referência do BAliBASE. As entradas possuem sequências com menos de 20% de identidade entre elas. Para maiores informações, consulte a Seção 2.4.1.

RV12: é um dos subconjuntos do primeiro conjunto de referência do BAliBASE. As entradas possuem sequências com identidade entre 20 e 40%. Para maiores informações, consulte a Seção 2.4.1.

RV13: é um dos subconjuntos do primeiro conjunto de referência do BAliBASE. As entradas possuem sequências com elevado grau de identidade entre elas. No BAliBASE 3.0 foi descartado porque os alinhadores facilmente alinhavam bem suas entradas e, assim, não era de grande utilidade. Para maiores informações, consulte a Seção 2.4.1.

RV1X: refere-se ao primeiro conjunto de referência do BAliBASE. Suas entradas são compostas de sequências equidistantes. Subdivide-se em RV11, RV12 e RV13. Para maiores informações, consulte a Seção 2.4.1.

RV20: refere-se ao segundo conjunto de referência do BAliBASE. Suas entradas são compostas por uma família de proteínas e sequências órfãs (sequências altamente divergentes do conjunto). Para maiores informações, consulte a Seção 2.4.1.

RV30: refere-se ao terceiro conjunto de referência do BAliBASE. Suas entradas são compostas por subgrupos de sequências altamente relacionadas, mas com menos de 25% de identidade entre sequências de grupos distintos. Para maiores informações, consulte a Seção 2.4.1.

RV40: refere-se ao quarto conjunto de referência do BAliBASE. Suas entradas são compostas por sequências com longas extensões nas extremidades. Há uma grande diferença de comprimento entre as sequências, o que exigirá a criação de *gaps* longos no alinhamento. Para maiores informações, consulte a Seção 2.4.1.

RV50: refere-se ao quinto conjunto de referência do BAliBASE. Suas entradas são compostas por sequências com inserções internas longas. Há uma grande diferença de comprimento entre as sequências, o que exigirá a criação de *gaps* longos no alinhamento. Para maiores informações, consulte a Seção 2.4.1.

RVS1: subconjunto de entradas do BAliBASE gerado a partir de sequências de todos os conjuntos de referência. Para maiores informações, consulte a Seção 3.5.

RVS2: subconjunto de entradas do BAliBASE gerado a partir de sequências de todos os conjuntos de referência. Para maiores informações, consulte a Seção 3.5.

RVS3: subconjunto de entradas do BAliBASE gerado a partir de sequências dos conjuntos RV11 e RV12. Para maiores informações, consulte a Seção 5.1.

S1: é uma função utilizada para quantificar a qualidade de alinhamentos gerados por um alinhador múltiplo iterativo não estocástico desenvolvido ao longo deste trabalho. Para maiores informações, consulte a Seção 5.1.

S2: é uma funções utilizada para quantificar a qualidade de alinhamentos gerados por um alinhador múltiplo iterativo não estocástico desenvolvido ao longo deste trabalho. Para maiores informações, consulte a Seção 5.1.

SA: é um acrônimo de *Simulated Annealing*.

SABmark: é uma ferramenta para avaliação de alinhamentos múltiplos. Para maiores informações, consulte a Seção 2.4.

SAGA: é um alinhador múltiplo iterativo. Consulte a Seção 2.3.2 para maiores informações.

SAGA-COFFEE: é um alinhador múltiplo baseado em consistência. Na verdade, uma adaptação no SAGA para otimizar a função COFFEE, que é baseada em consistência. Consulte a Seção 2.3.3 para maiores informações.

SAP: é uma ferramenta para superposição de estruturas. Para maiores informações, consulte o artigo que o descreve [241].

SE-B(6): é um alfabeto comprimido. Para consultar as classes que define, ver a Tabela 4.1. Para maiores informações sobre alfabetos comprimidos, consulte o artigo de Robert C. Edgar [67].

SE-B(8): é um alfabeto comprimido. Para consultar as classes que define, ver a Tabela 4.1. Para maiores informações sobre alfabetos comprimidos, consulte o artigo de Robert C. Edgar [67].

SE-B(10): é um alfabeto comprimido. Para consultar as classes que define, ver a Tabela 4.1. Para maiores informações sobre alfabetos comprimidos, consulte o artigo de Robert C. Edgar [67].

SE-B(14): é um alfabeto comprimido. Para consultar as classes que define, ver a Tabela 4.1. Para maiores informações sobre alfabetos comprimidos, consulte o artigo de Robert C. Edgar [67].

Seleção de par: é um dos procedimentos a executar na terceira etapa do alinhamento progressivo (construção do alinhamento). Consiste em determinar o próximo par a alinhar. Para maiores informações, consulte a Seção 3.3.1.

Seleção do par mais próximo: é um método para seleção de par em um alinhamento progressivo. Para maiores informações, consulte a Seção 3.3.1.

Seleção por bloco único: é um método para seleção de par em um alinhamento progressivo. Para maiores informações, consulte a Seção 3.3.1.

*Sequence Similarity Crossover***:** é um operador de cruzamento, utilizado no ALGAe e desenvolvido ao longo deste trabalho. Consulte a Seção 5.2.3 para maiores informações.

Sequenciamento: consiste na determinação da ordem sequencial das partes constituintes (bases ou aminoácidos) de uma molécula de DNA, RNA ou proteína.

SE-V(10): é um alfabeto comprimido. Para consultar as classes que define, ver a Tabela 4.1. Para maiores informações sobre alfabetos comprimidos, consulte o artigo de Robert C. Edgar [67].

*Shift Gap Block Mutation***:** é um operador de mutação, utilizado no ALGAe e desenvolvido ao longo deste trabalho. Para maiores informações, consulte a Seção 5.2.3.

Simple Mutation: é um operador de mutação, utilizado no ALGAe e desenvolvido ao longo deste trabalho. Para maiores informações, consulte a Seção 5.2.3.

Single Point Crossover: é um operador de cruzamento, utilizado no ALGAe e desenvolvido ao longo deste trabalho. Para maiores informações, consulte a Seção 5.2.3.

Solis-D(10): é um alfabeto comprimido. Para consultar as classes que define, ver a Tabela 4.1. Para maiores informações sobre alfabetos comprimidos, consulte o artigo de Robert C. Edgar [67].

Solis-G(10): é um alfabeto comprimido. Para consultar as classes que define, ver a Tabela 4.1. Para maiores informações sobre alfabetos comprimidos, consulte o artigo de Robert C. Edgar [67].

Soma dos pares: função comumente otimizada na computação de um alinhamento múltiplo. Consiste em calcular a pontuação do MSA a partir da soma das pontuações das projeções de $n(n-1)/2$ alinhamentos de pares.

SP: é um dos sistemas de pontuação utilizado pelo `BAliBASE` para avaliar a qualidade de um alinhamento. Indica a porcentagem de pares de resíduos corretamente alinhados. Para maiores informações, consulte a Seção 2.4.

SPEM: é um alinhador múltiplo baseado em modelos. Para maiores informações, consulte a Seção 2.3.4.

TC: É um dos sistemas de pontuação utilizado pelo `BAliBASE` para avaliar a qualidade de um alinhamento. Indica a porcentagem de colunas corretamente alinhadas. Para maiores informações, consulte a Seção 2.4.

T-COFFEE: é um alinhador múltiplo baseado em consistência. Para maiores informações, consulte a Seção 2.3.3.

T-Lara: é um alinhador múltiplo baseado em modelos. Para maiores informações, consulte a Seção 2.3.4.

UPGMA: é um método para geração de árvore filogenética. Neste trabalho, as árvores geradas foram utilizadas como árvore guia na construção de um alinhamento através da abordagem progressiva. Para maiores informações, consulte a Seção 3.2.1.

Apêndice B

Alinhadores progressivos desenvolvidos

Na Tabela B.1 são listados todos os 342 alinhadores progressivos implementados. Utilize o glossário, disponível no Apêndice A, para esclarecimentos a respeito dos métodos.

Tabela B.1: Descrição dos alinhadores progressivos implementados. Legenda do título: CD - Método utilizado para computação das distâncias, AG - Método utilizado para geração de árvore guia, SL - Método utilizado para seleção do par a agrupar, MA - Método utilizado para agrupamento de alinhamentos, PS - Método utilizado para atribuição dos pesos das sequências.

Alinhador	**CD**	**AG**	**SL**	**MA**	**PS**
049	JTT	UPGMA	NP	AC	PP
051	JTT	UPGMA	NP	AP	PP
052	JTT	UPGMA	NP	AP	PM
053	JTT	UPGMA	NP	APA	PP
054	JTT	UPGMA	NP	APA	PM
055	JTT	–	BU	AC	PP
057	JTT	–	BU	AP	PP
058	JTT	–	BU	AP	PM
059	JTT	–	BU	APA	PP
060	JTT	–	BU	APA	PM
061	JTT	NJ	NP	AC	PP
063	JTT	NJ	NP	AP	PP
064	JTT	NJ	NP	AP	PM

Continua na próxima página

Tabela B.1 – continuação da página anterior

Alinhador	CD	AG	SL	MA	PS
065	JTT	NJ	NP	APA	PP
066	JTT	NJ	NP	APA	PM
073	PMB	UPGMA	NP	AC	PP
075	PMB	UPGMA	NP	AP	PP
076	PMB	UPGMA	NP	AP	PM
077	PMB	UPGMA	NP	APA	PP
078	PMB	UPGMA	NP	APA	PM
079	PMB	–	BU	AC	PP
081	PMB	–	BU	AP	PP
082	PMB	–	BU	AP	PM
083	PMB	–	BU	APA	PP
084	PMB	–	BU	APA	PM
085	PMB	NJ	NP	AC	PP
087	PMB	NJ	NP	AP	PP
088	PMB	NJ	NP	AP	PM
089	PMB	NJ	NP	APA	PP
090	PMB	NJ	NP	APA	PM
097	PAM	UPGMA	NP	AC	PP
099	PAM	UPGMA	NP	AP	PP
100	PAM	UPGMA	NP	AP	PM
101	PAM	UPGMA	NP	APA	PP
102	PAM	UPGMA	NP	APA	PM
103	PAM	–	BU	AC	PP
105	PAM	–	BU	AP	PP
106	PAM	–	BU	AP	PM
107	PAM	–	BU	APA	PP
108	PAM	–	BU	APA	PM
109	PAM	NJ	NP	AC	PP
111	PAM	NJ	NP	AP	PP
112	PAM	NJ	NP	AP	PM
113	PAM	NJ	NP	APA	PP
114	PAM	NJ	NP	APA	PM
121	PCM	UPGMA	NP	AC	PP
123	PCM	UPGMA	NP	AP	PP
124	PCM	UPGMA	NP	AP	PM

Continua na próxima página

Tabela B.1 – continuação da página anterior

Alinhador	CD	AG	SL	MA	PS
125	PCM	UPGMA	NP	APA	PP
126	PCM	UPGMA	NP	APA	PM
127	PCM	–	BU	AC	PP
129	PCM	–	BU	AP	PP
130	PCM	–	BU	AP	PM
131	PCM	–	BU	APA	PP
132	PCM	–	BU	APA	PM
133	PCM	NJ	NP	AC	PP
135	PCM	NJ	NP	AP	PP
136	PCM	NJ	NP	AP	PM
137	PCM	NJ	NP	APA	PP
138	PCM	NJ	NP	APA	PM
145	LD	UPGMA	NP	AC	PP
147	LD	UPGMA	NP	AP	PP
148	LD	UPGMA	NP	AP	PM
149	LD	UPGMA	NP	APA	PP
150	LD	UPGMA	NP	APA	PM
151	LD	–	BU	AC	PP
153	LD	–	BU	AP	PP
154	LD	–	BU	AP	PM
155	LD	–	BU	APA	PP
156	LD	–	BU	APA	PM
157	LD	NJ	NP	AC	PP
159	LD	NJ	NP	AP	PP
160	LD	NJ	NP	AP	PM
161	LD	NJ	NP	APA	PP
162	LD	NJ	NP	APA	PM
185	JTT	UPGMA	NP	LC	PP
187	JTT	–	BU	LC	PP
189	JTT	NJ	NP	LC	PP
193	PMB	UPGMA	NP	LC	PP
195	PMB	–	BU	LC	PP
197	PMB	NJ	NP	LC	PP
201	PAM	UPGMA	NP	LC	PP
203	PAM	–	BU	LC	PP

Continua na próxima página

Tabela B.1 – continuação da página anterior

Alinhador	CD	AG	SL	MA	PS
205	PAM	NJ	NP	LC	PP
209	PCM	UPGMA	NP	LC	PP
211	PCM	–	BU	LC	PP
213	PCM	NJ	NP	LC	PP
217	LD	UPGMA	NP	LC	PP
219	LD	–	BU	LC	PP
221	LD	NJ	NP	LC	PP
049b	JTT	UPGMA	NP	ACb	PP
051b	JTT	UPGMA	NP	APb	PP
052b	JTT	UPGMA	NP	APb	PM
053b	JTT	UPGMA	NP	APAb	PP
054b	JTT	UPGMA	NP	APAb	PM
055b	JTT	–	BU	ACb	PP
057b	JTT	–	BU	APb	PP
058b	JTT	–	BU	APb	PM
059b	JTT	–	BU	APAb	PP
060b	JTT	–	BU	APAb	PM
061b	JTT	NJ	NP	ACb	PP
063b	JTT	NJ	NP	APb	PP
064b	JTT	NJ	NP	APb	PM
065b	JTT	NJ	NP	APAb	PP
066b	JTT	NJ	NP	APAb	PM
073b	PMB	UPGMA	NP	ACb	PP
075b	PMB	UPGMA	NP	APb	PP
076b	PMB	UPGMA	NP	APb	PM
077b	PMB	UPGMA	NP	APAb	PP
078b	PMB	UPGMA	NP	APAb	PM
079b	PMB	–	BU	ACb	PP
081b	PMB	–	BU	APb	PP
082b	PMB	–	BU	APb	PM
083b	PMB	–	BU	APAb	PP
084b	PMB	–	BU	APAb	PM
085b	PMB	NJ	NP	ACb	PP
087b	PMB	NJ	NP	APb	PP
088b	PMB	NJ	NP	APb	PM

Continua na próxima página

Tabela B.1 – continuação da página anterior

Alinhador	CD	AG	SL	MA	PS
089b	PMB	NJ	NP	APAb	PP
090b	PMB	NJ	NP	APAb	PM
097b	PAM	UPGMA	NP	ACb	PP
099b	PAM	UPGMA	NP	APb	PP
100b	PAM	UPGMA	NP	APb	PM
101b	PAM	UPGMA	NP	APAb	PP
102b	PAM	UPGMA	NP	APAb	PM
103b	PAM	–	BU	ACb	PP
105b	PAM	–	BU	APb	PP
106b	PAM	–	BU	APb	PM
107b	PAM	–	BU	APAb	PP
108b	PAM	–	BU	APAb	PM
109b	PAM	NJ	NP	ACb	PP
111b	PAM	NJ	NP	APb	PP
112b	PAM	NJ	NP	APb	PM
113b	PAM	NJ	NP	APAb	PP
114b	PAM	NJ	NP	APAb	PM
121b	PCM	UPGMA	NP	ACb	PP
123b	PCM	UPGMA	NP	APb	PP
124b	PCM	UPGMA	NP	APb	PM
125b	PCM	UPGMA	NP	APAb	PP
126b	PCM	UPGMA	NP	APAb	PM
127b	PCM	–	BU	ACb	PP
129b	PCM	–	BU	APb	PP
130b	PCM	–	BU	APb	PM
131b	PCM	–	BU	APAb	PP
132b	PCM	–	BU	APAb	PM
133b	PCM	NJ	NP	ACb	PP
135b	PCM	NJ	NP	APb	PP
136b	PCM	NJ	NP	APb	PM
137b	PCM	NJ	NP	APAb	PP
138b	PCM	NJ	NP	APAb	PM
145b	LD	UPGMA	NP	ACb	PP
147b	LD	UPGMA	NP	APb	PP
148b	LD	UPGMA	NP	APb	PM

Continua na próxima página

Tabela B.1 – continuação da página anterior

Alinhador	CD	AG	SL	MA	PS
149b	LD	UPGMA	NP	APAb	PP
150b	LD	UPGMA	NP	APAb	PM
151b	LD	–	BU	ACb	PP
153b	LD	–	BU	APb	PP
154b	LD	–	BU	APb	PM
155b	LD	–	BU	APAb	PP
156b	LD	–	BU	APAb	PM
157b	LD	NJ	NP	ACb	PP
159b	LD	NJ	NP	APb	PP
160b	LD	NJ	NP	APb	PM
161b	LD	NJ	NP	APAb	PP
162b	LD	NJ	NP	APAb	PM
281	JTT	UPGMA	NP	APLog	PP
282	JTT	UPGMA	NP	APLog	PM
283	JTT	–	BU	APLog	PP
284	JTT	–	BU	APLog	PM
285	JTT	NJ	NP	APLog	PP
286	JTT	NJ	NP	APLog	PM
289	PMB	UPGMA	NP	APLog	PP
290	PMB	UPGMA	NP	APLog	PM
291	PMB	–	BU	APLog	PP
292	PMB	–	BU	APLog	PM
293	PMB	NJ	NP	APLog	PP
294	PMB	NJ	NP	APLog	PM
297	PAM	UPGMA	NP	APLog	PP
298	PAM	UPGMA	NP	APLog	PM
299	PAM	–	BU	APLog	PP
300	PAM	–	BU	APLog	PM
301	PAM	NJ	NP	APLog	PP
302	PAM	NJ	NP	APLog	PM
305	PCM	UPGMA	NP	APLog	PP
306	PCM	UPGMA	NP	APLog	PM
307	PCM	–	BU	APLog	PP
308	PCM	–	BU	APLog	PM
309	PCM	NJ	NP	APLog	PP

Continua na próxima página

Tabela B.1 – continuação da página anterior

Alinhador	CD	AG	SL	MA	PS
310	PCM	NJ	NP	APLog	PM
313	LD	UPGMA	NP	APLog	PP
314	LD	UPGMA	NP	APLog	PM
315	LD	–	BU	APLog	PP
316	LD	–	BU	APLog	PM
317	LD	NJ	NP	APLog	PP
318	LD	NJ	NP	APLog	PM
321	LOGD	UPGMA	NP	AC	PP
322	LOGD	UPGMA	NP	LC	PP
323	LOGD	UPGMA	NP	AP	PP
324	LOGD	UPGMA	NP	AP	PM
325	LOGD	UPGMA	NP	APA	PP
326	LOGD	UPGMA	NP	APA	PM
327	LOGD	UPGMA	NP	APLog	PP
328	LOGD	UPGMA	NP	APLog	PM
329	LOGD	–	BU	AC	PP
330	LOGD	–	BU	LC	PP
331	LOGD	–	BU	AP	PP
332	LOGD	–	BU	AP	PM
333	LOGD	–	BU	APA	PP
334	LOGD	–	BU	APA	PM
335	LOGD	–	BU	APLog	PP
336	LOGD	–	BU	APLog	PM
337	LOGD	NJ	NP	AC	PP
338	LOGD	NJ	NP	LC	PP
339	LOGD	NJ	NP	AP	PP
340	LOGD	NJ	NP	AP	PM
341	LOGD	NJ	NP	APA	PP
342	LOGD	NJ	NP	APA	PM
343	LOGD	NJ	NP	APLog	PP
344	LOGD	NJ	NP	APLog	PM
353	JTT	UPGMA	NP	ACLog	PP
354	JTT	–	BU	ACLog	PP
355	JTT	NJ	NP	ACLog	PP
357	PMB	UPGMA	NP	ACLog	PP

Continua na próxima página

Tabela B.1 – continuação da página anterior

Alinhador	CD	AG	SL	MA	PS
358	PMB	–	BU	ACLog	PP
359	PMB	NJ	NP	ACLog	PP
361	PAM	UPGMA	NP	ACLog	PP
362	PAM	–	BU	ACLog	PP
363	PAM	NJ	NP	ACLog	PP
365	PCM	UPGMA	NP	ACLog	PP
366	PCM	–	BU	ACLog	PP
367	PCM	NJ	NP	ACLog	PP
369	LD	UPGMA	NP	ACLog	PP
370	LD	–	BU	ACLog	PP
371	LD	NJ	NP	ACLog	PP
373	LOGD	UPGMA	NP	ACLog	PP
374	LOGD	–	BU	ACLog	PP
375	LOGD	NJ	NP	ACLog	PP
281b	JTT	UPGMA	NP	APLogb	PP
282b	JTT	UPGMA	NP	APLogb	PM
283b	JTT	–	BU	APLogb	PP
284b	JTT	–	BU	APLogb	PM
285b	JTT	NJ	NP	APLogb	PP
286b	JTT	NJ	NP	APLogb	PM
289b	PMB	UPGMA	NP	APLogb	PP
290b	PMB	UPGMA	NP	APLogb	PM
291b	PMB	–	BU	APLogb	PP
292b	PMB	–	BU	APLogb	PM
293b	PMB	NJ	NP	APLogb	PP
294b	PMB	NJ	NP	APLogb	PM
297b	PAM	UPGMA	NP	APLogb	PP
298b	PAM	UPGMA	NP	APLogb	PM
299b	PAM	–	BU	APLogb	PP
300b	PAM	–	BU	APLogb	PM
301b	PAM	NJ	NP	APLogb	PP
302b	PAM	NJ	NP	APLogb	PM
305b	PCM	UPGMA	NP	APLogb	PP
306b	PCM	UPGMA	NP	APLogb	PM
307b	PCM	–	BU	APLogb	PP

Continua na próxima página

Tabela B.1 – continuação da página anterior

Alinhador	CD	AG	SL	MA	PS
308b	PCM	–	BU	APLogb	PM
309b	PCM	NJ	NP	APLogb	PP
310b	PCM	NJ	NP	APLogb	PM
313b	LD	UPGMA	NP	APLogb	PP
314b	LD	UPGMA	NP	APLogb	PM
315b	LD	–	BU	APLogb	PP
316b	LD	–	BU	APLogb	PM
317b	LD	NJ	NP	APLogb	PP
318b	LD	NJ	NP	APLogb	PM
321b	LOGD	UPGMA	NP	ACb	PP
323b	LOGD	UPGMA	NP	APb	PP
324b	LOGD	UPGMA	NP	APb	PM
325b	LOGD	UPGMA	NP	APAb	PP
326b	LOGD	UPGMA	NP	APAb	PM
327b	LOGD	UPGMA	NP	APLogb	PP
328b	LOGD	UPGMA	NP	APLogb	PM
329b	LOGD	–	BU	ACb	PP
331b	LOGD	–	BU	APb	PP
332b	LOGD	–	BU	APb	PM
333b	LOGD	–	BU	APAb	PP
334b	LOGD	–	BU	APAb	PM
335b	LOGD	–	BU	APLogb	PP
336b	LOGD	–	BU	APLogb	PM
337b	LOGD	NJ	NP	ACb	PP
339b	LOGD	NJ	NP	APb	PP
340b	LOGD	NJ	NP	APb	PM
341b	LOGD	NJ	NP	APAb	PP
342b	LOGD	NJ	NP	APAb	PM
343b	LOGD	NJ	NP	APLogb	PP
344b	LOGD	NJ	NP	APLogb	PM
353b	JTT	UPGMA	NP	ACLogb	PP
354b	JTT	–	BU	ACLogb	PP
355b	JTT	NJ	NP	ACLogb	PP
357b	PMB	UPGMA	NP	ACLogb	PP
358b	PMB	–	BU	ACLogb	PP

Continua na próxima página

Tabela B.1 – continuação da página anterior

Alinhador	CD	AG	SL	MA	PS
359b	PMB	NJ	NP	ACLogb	PP
361b	PAM	UPGMA	NP	ACLogb	PP
362b	PAM	–	BU	ACLogb	PP
363b	PAM	NJ	NP	ACLogb	PP
365b	PCM	UPGMA	NP	ACLogb	PP
366b	PCM	–	BU	ACLogb	PP
367b	PCM	NJ	NP	ACLogb	PP
369b	LD	UPGMA	NP	ACLogb	PP
370b	LD	–	BU	ACLogb	PP
371b	LD	NJ	NP	ACLogb	PP
373b	LOGD	UPGMA	NP	ACLogb	PP
374b	LOGD	–	BU	ACLogb	PP
375b	LOGD	NJ	NP	ACLogb	PP
053p	JTT	UPGMA	NP	APAp	PP
054p	JTT	UPGMA	NP	APAp	PM
059p	JTT	–	BU	APAp	PP
060p	JTT	–	BU	APAp	PM
065p	JTT	NJ	NP	APAp	PP
066p	JTT	NJ	NP	APAp	PM
077p	PMB	UPGMA	NP	APAp	PP
078p	PMB	UPGMA	NP	APAp	PM
083p	PMB	–	BU	APAp	PP
084p	PMB	–	BU	APAp	PM
089p	PMB	NJ	NP	APAp	PP
090p	PMB	NJ	NP	APAp	PM
101p	PAM	UPGMA	NP	APAp	PP
102p	PAM	UPGMA	NP	APAp	PM
107p	PAM	–	BU	APAp	PP
108p	PAM	–	BU	APAp	PM
113p	PAM	NJ	NP	APAp	PP
114p	PAM	NJ	NP	APAp	PM
125p	PCM	UPGMA	NP	APAp	PP
126p	PCM	UPGMA	NP	APAp	PM
131p	PCM	–	BU	APAp	PP
132p	PCM	–	BU	APAp	PM

Continua na próxima página

Tabela B.1 – continuação da página anterior

Alinhador	CD	AG	SL	MA	PS
137p	PCM	NJ	NP	APAp	PP
138p	PCM	NJ	NP	APAp	PM
149p	LD	UPGMA	NP	APAp	PP
150p	LD	UPGMA	NP	APAp	PM
155p	LD	–	BU	APAp	PP
156p	LD	–	BU	APAp	PM
161p	LD	NJ	NP	APAp	PP
162p	LD	NJ	NP	APAp	PM
325p	LOGD	UPGMA	NP	APAp	PP
326p	LOGD	UPGMA	NP	APAp	PM
333p	LOGD	–	BU	APAp	PP
334p	LOGD	–	BU	APAp	PM
341p	LOGD	NJ	NP	APAp	PP
342p	LOGD	NJ	NP	APAp	PM

Apêndice C

Revisão Bibliográfica

Neste apêndice são apresentados resumos de trabalhos relevantes em alinhamento múltiplo de sequências.

Nas seções C.1, C.2, C.3 e C.4, *surveys* são resumidos. Na Seção C.5 é apresentado um resumo do artigo que descreve a versão 3.0 do BAliBASE. O resumo de um trabalho com comparações de alinhadores, utilizando o BAliBASE, é apresentado na Seção C.6 e na Seção C.7 é apresentado um resumo do trabalho de Hogeweg e Hesper, onde se definiu a abordagem progressiva.

Em seguida, é apresentada uma série de resumos de trabalhos que descrevem alinhadores múltiplos conhecidos. Na Seção C.8 tem-se a descrição do progressivo Clustal W, do iterativo estocástico SAGA na Seção C.9 e do não estocástico PRRP na Seção C.10. Na Seção C.11 descreve-se o algoritmo do alinhador baseado em consistência ProbCons e do baseado em consenso M-COFFEE na Seção C.12. Por fim, há ainda decrição para o algoritmo do baseado em modelos DbClustal na Seção C.13 e na Seção C.14 do DiAlign, que é baseado em blocos.

Ainda estudou-se, e resumiu-se as respectivas descrições, três algoritmos de alinhadores múltiplos menos conhecidos. O primeiro foi definido por Meidanis e Setubal na Seção C.15, o segundo por Daniele Santos na Seção C.16 e o terceiro por Lopes e Moritz na Seção C.17.

C.1 Multiple alignments for structural, functional, or phylogenetic analyses of homologous sequences

Este trabalho [65] expõe uma extensa revisão acerca de alinhamento múltiplo de sequências. Apresenta desde conceitos básicos até ferramentas de suporte a MSA. Claro que passando pelos métodos existentes (até 2000, época da publicação).

Entender a estrutura, função e evolução dos genes é um dos principais objetivos de projetos de sequenciamento de genoma. Classicamente, a função de um gene é investigada experimentalmente através da análise de fenótipos mutantes. Porém, análise comparativa de sequências homólogas mostrou ser uma abordagem muito eficiente para estudar função de gene. Estudar padrões de mutação através de análise de sequências homólogas é útil não apenas para estudar relacionamentos evolucionários entre sequências, mas também para identificar restrições estruturais ou funcionais nas sequências de RNA, DNA ou proteína.

O alinhamento de sequências homólogas consiste de tentar posicionar resíduos (nucleotídeos ou aminoácidos) em colunas, que derivam de um resíduo de um ancestral comum. Isto é obtido pela introdução de *gaps*, que representam inserções ou remoções, nas sequências. Desta forma, um alinhamento é um modelo hipotético de mutações (substituições, inserções ou remoções) que ocorreram durante a evolução da sequência. O melhor alinhamento será aquele mais parecido com um cenário evolucionário.

Devido a complexidade computacional deste problema, algoritmos de alinhamento utilizáveis na prática não podem garantir a melhor solução. E, mesmo com um algoritmo ideal, encontrar o melhor alinhamento poderá não ser garantido porque o conhecimento corrente de probabilidade de ocorrência de diferentes tipos de mutações é ainda limitado. Entretanto, quanto menos divergentes forem as sequências homólogas, mais confiáveis tornam-se as heurísticas e algoritmos aproximados. Na prática, tais alinhadores são comumente utilizados em biologia molecular ou evolucionária. A seguir, são listados exemplos típicos de uso de MSA.

- Busca por similaridades fracas, mas significantes, em bases de dados de sequências;
- Demonstração de homologia entre sequências;
- Filogenia molecular;
- Identificação de genes relacionados;
- Identificação de sítios funcionalmente importantes;
- Predição de estrutura;
- Predição de função; ou
- Projeto de *primers* para PCR.

O procedimento geral para a computação de um alinhamento múltiplo de sequências homólogas consiste de três passos:

1. Busca por homólogas em bases de dados de sequências

2. Computação dos alinhamentos

3. Verificação e edição dos alinhamentos

Neste texto o foco estará nos dois últimos passos. Inicialmente serão definidos alguns conceitos gerais acerca da metodologia de alinhamento múltiplo. Então descreve-se e compara-se os diferentes métodos que têm sido desenvolvidos para alinhamento de sequências. Uma lista de URLs para sítios na *web* de alinhadores está disponível no seguinte endereço: `http://pbil.univ-lyon1.fr/alignment.html`.

Alguns problemas relacionados a alinhamento múltiplo, tal como montagem de *contigs*, não serão tratados neste texto. Apenas serão descritos métodos para alinhamento de sequências homólogas. Sendo assim, também não será abordado o problema da busca por *motifs* comuns em um conjunto de sequências não relacionadas neste texto. Para maiores detalhes sobre este assunto em particular, consulte o texto de Jonassen [128].

C.1.1 Conceitos básicos

Homologia

Duas sequências são chamadas homólogas quando derivam de um ancestral comum. Geralmente, homologia é inferida por similaridade de sequências. É importante salientar, entretanto, que similaridade não reflete, necessariamente, homologia. Similaridade entre fragmentos curtos de sequência podem resultar de convergência evolucionária [63] ou podem resultar simplesmente do acaso.

Há ainda o fato de muitas sequências conterem fragmentos relativamente longos de composições de nucleotídeos ou aminoácidos muito tendenciosas, por exemplo: repetições CA em DNA ou domínios ricos em prolina em proteínas [268]. Geralmente, similaridades entre tais sequências com "regiões de baixa complexidade" não refletem relacionamento evolucionário.

Na ausência de tais composições tendenciosas, similaridade em uma região extensa frequentemente implica em homologia. Testes estatísticos podem ser usados para avaliar as chances de uma similaridade observada ocorrer ao acaso e, assim, aceitar ou rejeitar a hipotética homologia [11]. Tais testes são geralmente fornecidos por programas de busca por similaridade.

Alinhamentos múltiplos podem ser úteis para ajudar a demonstrar homologia. Uma pequena similaridade, que poderia ser considerada não significante em um alinhamento de par de sequências, pode se mostrar altamente significante se os mesmos resíduos são conservados em outras sequências com relação distante. Deve-se, entretanto, ser enfatizado que, se as sequências divergiram muito, a homologia pode não ser reconhecida apenas com similaridade de sequências.

Alinhamento global e local

Em muitos casos, homologia está restrita a regiões limitadas das sequências. Muitas proteínas consistem de uma combinação de "módulos" discretos que foram deslocados durante a evolução. É claro que muitos genes codificadores de proteínas resultam da recombinação entre diferentes fragmentos de outros genes. Esta evolução modular tem um papel de maior importância na evolução de proteína e tem sido particularmente facilitado em eucariotos graças a presença de *introns* nos genes [194].

Cópias múltiplas de um dado módulo podem ser repetidas em uma sequência e um conjunto de módulos pode ocorrer em diferentes posições relativas em genes diferentes. Em tais casos, não é possível alinhar sequências em seu tamanho completo (alinhamento global) e é assim necessário executar alinhamentos apenas em regiões homólogas (alinhamento local).

Matrizes de substituição e pesos para *gaps*

Como indicado anteriormente, buscar pelo melhor alinhamento consiste de buscar por aquele que aparente um melhor cenário evolucionário. Assim, a probabilidade de ocorrência de diferentes eventos mutacionais durante a evolução deve ser levada em consideração ao computar um alinhamento múltiplo. Em alinhamentos, tipicamente três tipos de mutações são consideradas: substituições, inserções ou remoções. Os dois últimos eventos são frequentemente indistinguíveis e são comumente referenciados como *indels*.

A probabilidade de substituição de um aminoácido por outro depende da estrutura do código genético (por exemplo: o número de mutações necessárias para transformar um códon em outro) e também do efeito fenotípico da mutação. Substituições de um aminoácido por outro com propriedades bioquímicas similares geralmente não têm grande efeito na estrutura e consequentemente na função da proteína. Assim, durante a evolução, substituições conservativas são relativamente frequentes se comparadas a outras substituições.

É importante notar que a probabilidade de substituição de um aminoácido por outro depende da distância evolucionária entre as sequências. Vários métodos foram propostos para a construção de séries de matrizes que estimam a probabilidade de todas as possíveis substituições para diferentes distâncias evolucionárias [56, 85, 106, 217]. As mais utilizadas são as matrizes PAM [56, 217] e BLOSUM [106].

Nas matrizes PAM os índices das matrizes crescem de acordo com as distâncias evolucionárias. Por exemplo, a PAM80 é adequada para pequenas distâncias evolucionárias e a PAM250 para maiores distâncias. Já as matrizes BLOSUM adotam uma convenção oposta, assim BLOSUM80 é adequada para pequenas distâncias evolucionárias e BLOSUM45 para maiores distâncias. Geralmente os alinhadores permitem ao usuário

selecionar a matriz de substituição a utilizar. No Clustal W [245] as matrizes de substituição são automaticamente selecionadas e variam nos diferentes estágios do alinhamento de acordo com o grau de divergência entre as sequências.

Probabilidades de substituição também variam ao longo das sequências de acordo com ambientes locais de aminoácidos na proteína dobrada (por exemplo, hélice-α ou folha-β). Matrizes de substituição de ambiente específico foram desenvolvidas [191]. É importante salientar, entretanto, que estas matrizes são raramente utilizadas para alinhamento múltiplo nas ferramentas disponíveis atualmente.

Ao nível de DNA, probabilidades de susbtituições variam de acordo com as bases. Transições são geralmente mais frequentes que transversões. Transições são substituições entre bases púricas ou entre bases pirimídicas e transversões substituições entre uma base púrica e uma pirimídica. São bases púricas A e G e são bases pirimídicas C e T (U para RNA). Assim, alinhadores múltiplos geralmente propõem um parâmetro de peso mais rígido para transversões que transições.

Probabilidades de substituições de nucleotídeos também dependem das bases vizinhas, por exemplo: em vertebrados, C no dinucleotídeo CG é extremamente mutável [22, 112]. Os alinhadores correntes ainda não fazem uso de tal informação.

A probabilidade de ocorrência de um *indel* depende do tamanho. Assim, ao computar um alinhamento, frequentemente estima-se penalidades (p) associadas com *gaps* usando um modelo linear ou "afim" tal como:

$$p = a + bL$$

onde L é o tamanho do *gap*, a a penalidade para abertura de *gap* e b a penalidade para extensão de *gap*. Entretanto, análises de alinhamentos de sequências homólogas têm mostrado que este modelo, para sequências de proteínas e nucleotídeos, subestima a probabilidade de *indels* longos [86, 85, 100]. Penalidades mais realísticas para *indels* podem ser estimadas com modelos tais como o seguinte:

$$p = a + b \times log(L).$$

Devido à complexidade computacional, tais modelos não têm sido implementados nos alinhadores comumente utilizados. Felizmente, outras abordagens para alinhar sequências com grandes *indels* foram propostas, como poderá ser visto na Seção C.1.3.

Em proteínas a probabilidade de ocorrência de *indels* também depende do grau de divergência entre as sequências [86, 85]. Tais probabilidades variam de acordo com a natureza das sequências: proteína, RNA estrutural, DNA não codificante, etc. Em proteínas, *indels* são mais frequentes em laços externos que no núcleo da estrutura. Desta forma, é possível utilizar o conhecimento da estrutura da proteína para valorar pesos de *indels*.

No Clustal W, por exemplo, implementa-se penalidade para *indel* específica para resíduo e esta é localmente reduzida para encorajar novos *gaps* em potenciais regiões de laço. Em casos onde informações acerca da estrutura secundária estão disponíveis, máscaras para penalidades de *indels* também podem ser usadas para guiar o alinhamento.

É importante salientar que, na maioria dos programas, parâmetros padrões para penalidades de *gap* têm sido definidos para proteínas globulares típicas, que podem não ser os melhores para outras sequências.

C.1.2 Busca por sequências homólogas

O primeiro passo na análise de uma família de sequências homólogas consiste da busca por todos os membros disponíveis daquela família. Sequências publicadas são armazenadas em bancos de dados públicos, tais como GenBank [35] e EMBL [233] para sequências de nucleotídeo e Swiss-Prot/TrEMBL [23] e PIR [27] para sequências de proteína. Sistemas de recuperação de informação tais como Entrez [216], SRS [71] e ACNUC [94] foram desenvolvidos para a consulta destas bases de dados e extração de sequências de acordo com as anotações associadas (tais como palavras chaves, taxonomia ou autores).

Infelizmente, não é possível utilizar as anotações para identificar no banco de dados todas as sequências homólogas pertencentes a uma dada família. Atualmente, a forma mais eficiente de identificar estes homólogos consiste em comparar um membro da família com todas as outras sequências no banco de dados, utilizando-se para isso um programa de busca por similaridade tal como FASTA [195] ou BLAST [12, 14]. Para uma discussão abrangente acerca de busca por similaridade de sequências, consulte a revisão de Altschul e colegas [9].

C.1.3 Métodos para alinhamento múltiplo

O problema do alinhamento múltiplo de sequências é algoritmicamente difícil, ou seja, métodos que garantam o melhor alinhamento requerem tanto tempo e espaço de memória que não é possível utilizá-los na prática. Torna-se impraticável utilizar algoritmos exatos com entradas de 10 a 15 sequências de tamanho 100.

Desta forma, algoritmos alternativos têm sido desenvolvidos usando heurísticas para ganhar em velocidade e reduzir os requisitos de memória. Embora não hajam garantias, observa-se que os resultados são muito úteis na prática e frequentemente seus alinhamentos estão muito próximos da solução exata. Neste trabalho divide-se os algoritmos para MSA em quatro categorias:

1. Algoritmos que garantem o alinhamento ótimo para um dado esquema de pontuação. Estes possuem fortes limitações quanto ao número e tamanho das sequências;

2. Algoritmos heurísticos baseados na abordagem de alinhamento progressivo de pares;
3. Algoritmos heurísticos que constroem alinhamentos globais baseados em alinhamentos locais;
4. Algoritmos heurísticos que constroem alinhamentos múltiplos locais.

Esta lista não é exaustiva. Há outros métodos para MSA tais como os baseados em modelos ocultos de Markov [208] ou algoritmos genéticos [182]. Para uma revisão dos algoritmos de alinhamento múltiplo, consulte o trabalho de Chan e colegas [48].

Métodos ótimos para alinhamentos múltiplos globais

Deve-se chamar a atenção que o termo ótimo aqui refere-se a um ótimo matemático. Se um alinhamento matematicamente ótimo corresponde ou não a um alinhamento biologicamente correto irá depender da escolha dos parâmetros e da forma que se calcula a pontuação do alinhamento múltiplo.

Em princípio, a pontuação de um alinhamento múltiplo deve refletir o grau de semelhança de suas sequências, de acordo com um modelo evolucionário dado. Há diversas formas de mensurar a pontuação (ou custo) de um alinhamento múltiplo.

Considere que uma sequência é um conjunto ordenado de letras de um alfabeto Σ. Um alinhamento de n sequências $S_1, S_2, \ldots, S_n$ pode ser definido como uma matriz $A = a(S_1, S_2, \ldots, S_n)$, onde cada entrada A_{ij} é uma uma letra de Σ ou um símbolo nulo (frequentemente representado por "-"). A linha i de A é a sequência S_i, depois dos *gaps* removidos.

No modelo mais simples, a pontuação de um alinhamento de n sequências é definido como a soma das pontuações de suas colunas. Entretanto, este modelo é grosseiro, uma vez que cada coluna do alinhamento é considerada independentemente de seu contexto. Por exemplo, um *gap* de tamanho L é considerado o mesmo que L *indels* independentes.

Em modelos mais realísticos, um *gap* é considerado um único evento mutacional e associado com uma penalidade proporcional ao seu tamanho. Em tais modelos, pontuação para alinhamento de par é definida como a soma das pontuações de substituições e penalidades para *gaps*. Porém, a definição da pontuação de um alinhamento múltiplo é mais complexa.

Uma solução possível, conhecida como soma dos pares (SP) [47], consiste em calcular a pontuação do alinhamento múltiplo a partir das pontuações dos alinhamentos de pares. Um alinhamento múltiplo $a(S_1, S_2, \ldots, S_n)$ contém $n(n-1)/2$ alinhamentos de pares $a(S_i, S_j)$, onde $1 \leq i < j \leq n$ e onde desprezam-se todas as colunas onde há duas letras nulas. A pontuação SP de um alinhamento múltiplo é definida como a soma de todas estas pontuações de projeções [47].

Pontuação SP simples, entretanto, pode ser inapropriada quando alguns grupos de sequências são extremamente ou pouco representados em uma família. Este defeito pode ser corrigido pela introdução de um sistema de pesos apropriado [10, 92], que associa um peso a cada sequência. Assim, pode-se dar um peso menor a sequências de grupos super-representados. Outra solução consistem em usar uma função de pontuação baseada em uma árvore evolucionária. As folhas da árvore são as sequências que se deseja alinhar, e os nós internos são suas hipotéticas sequências ancestrais.

Um alinhamento ótimo é um que possua pontuação máxima. Um algoritmo eficiente, baseado em programação dinâmica, para computar alinhamentos ótimos para pares de sequências foi proposto por Needleman e Wunsch [174]. Esta abordagem pode ser facilmente generalizada para mais que duas sequências, porém o custo para alinhar, n sequências com comprimento l cada, é muito elevado. Ele requer $O(2^n l^n)$ em tempo e $O(l^n)$ em espaço. Esta complexidade pode ainda ser mais elevada caso a penalidade para *gap* não seja linear.

Carillo e Lipman [47] propuseram um algoritmo *branch and bound* para computar um alinhamento de pontuação SP máxima. Faz uso de um limiar superior para a pontuação e, assim, impõe limites para o espaço e tempo. Esta abordagem foi implementada pelo programa MSA [151].

Conforme comentado anteriormente, uma função de pontuação alternativa poderia fazer uso de uma árvore evolucionária. Em princípio, seria uma função melhor que a SP, entretanto o problema de alinhamento usando uma árvore evolucionária é também difícil, mesmo quando a árvore é dada [127].

Alinhamento global progressivo

Alinhamento progressivo é o método mais comumente utilizado para alinhamento múltiplo de sequências biológicas. Esta abordagem heurística é muito rápida, requer pouca memória e oferece um bom desempenho quando as sequências são homólogas e relativamente bem conservadas [74, 239].

Alinhamento progressivo consiste em construir um alinhamento múltiplo a partir de alinhamentos de pares. Nesta abordagem é possível identificar três passos: computação dos alinhamentos de pares, construção da árvore guia a partir das distâncias obtidas no passo anterior e construção do alinhamento múltiplo guiado pela árvore.

As diferenças entre as ferramentas para MSA que fazem uso da abordagem progressiva estão nos métodos que utilizam em cada um dos três passos.

Clustal W [245], por exemplo, permite o uso de programação dinâmica ou métodos heurísticos para a computação do alinhamento de pares. Com programação dinâmica tem-se resultados mais precisos, mas com os métodos heurísticos é possível ganhar em velocidade de processamento.

Para a construção da árvore guia há diversos métodos. Os mais conhecidos são UPGMA [228] e *Neighbor Joining* [210]. É conhecido um problema no UPGMA, onde é dada uma ordem de ramificação incorreta quando as taxas de substituição variam em diferentes linhagens. Este problema levou a utilização de *Neighbor Joining*, ao invés de UPGMA, nas versões mais recentes do Clustal (a partir do Clustal W).

O principal problema no terceiro passo consiste no alinhamento de dois alinhamentos. O método mais simples é reduzir cada alinhamento a uma sequência consenso e usar um algoritmo de alinhamento de pares comum. Na maioria dos programas, representa-se alinhamento com perfis, onde colunas são reduzidas a distribuições das frequências de cada letra. Alinham-se dois perfis de forma similar a de duas sequências por programação dinâmica. O alinhamento de dois perfis de tamanho l tem complexidade $O(a^2l^2)$, onde a é o tamanho do alfabeto. Clustal [114] usa alinhamento de consenso, mas Clustal W usa alinhamento de perfis com penalidades de *gap* para posição específica.

O principal problema da abordagem progressiva oriunda em sua natureza gulosa, onde qualquer erro cometido em etapas iniciais do processo não são mais corrigidos.

Estratégias de otimização iterativas têm sido propostas para resolver tal problema, tais como: RIW ou DNR [93]. Em testes, estes métodos apresentaram melhores desempenhos que o Clustal W, por exemplo. Porém, mesmo sendo mais rápido que algoritmos ótimos, estes são ainda muito lentos para entradas grandes.

Outra limitação para a abordagem progressiva é o requisito de computação das distâncias de pares entre todas as sequências para construir a árvore guia. Pode-se em alguns casos ter de alinhar sequências homólogas e alguns fragmentos não sobrepostos (por exemplo, sequências parciais de proteína). Quando as sequências não se sobrepõem, estas obviamente são não relacionadas e assim a árvore guia gerada pode ser completamente incorreta. O alinhamento produzido, neste caso, pode ser imprevisível.

Alinhamento global baseado em blocos

As sequências a comparar podem compartilhar blocos conservados, separados por regiões não conservadas contendo longos *indels*. Em tais casos, o resultado do alinhamento global ótimo ou progressivo irá depender fortemente da escolha dos parâmetros para penalidade de *gap*.

Uma alternativa consiste em buscar por blocos conservados para usar como âncoras no alinhamento das sequências. Blocos são alinhamentos de fragmentos de sequências (alinhamentos locais). A maioria dos métodos que fazem uso desta abordagem consideram apenas blocos sem *gaps*. Os blocos permitidos podem ser exatos (composto por segmentos idênticos) ou não exatos e podem ser uniformes (encontrado em todas as sequências) ou não.

O primeiro programa de alinhamento múltiplo de blocos [229] usou um algoritmo de

ordenação para computar blocos exatos uniformes. Algoritmos mais rápidos baseados em árvores de sufixo [159], ou estruturas de dados equivalentes, podem também ser utilizados para computar blocos exatos. Porém regiões homólogas raramente são perfeitamente conservadas.

ASSEMBLE [256] realiza uma análise em uma matriz de pontos em todos os pares de sequências e então compara estas matrizes de pontos para encontrar blocos uniformes, não necessariamente exatos. Na prática, é comum alguns blocos que não estão presentes em todas as sequências. Melhorias têm consistido no desenvolvimento de métodos que permitam blocos que não sejam necessariamente uniformes.

Um conjunto de blocos uniformes é consistente quando cada par de blocos está em ordem, ou seja, eles não se cruzam. O problema de selecionar um conjunto de blocos consistente e ótimo pode ser reduzido a um algoritmo clássico para caminho ótimo em um grafo [229], que requer tempo $O(M^2)$ para M blocos. Algoritmos mais rápidos (subquadráticos) para computação de conjuntos de blocos uniformes e consistentes ótimos foram propostos [173, 272]. Porém, quando os blocos são não uniformes o problema é intratável [271]. A consistência de blocos não uniformes não pode ser reduzida para uma relação binária entre eles. Um conjunto de três blocos não uniformes, tal que todos seus três pares de blocos são consistentes, não é necessariamente consistente.

Para computar um "bom" conjunto consistente de diagonais (numa matriz de pontos), DiAlign usa um algoritmo heurístico, que adiciona diagonais pela ordem decrescente de pontuação em um conjunto consistente de diagonais. Diagonais que não sejam consistentes com o conjunto corrente são rejeitadas.

Alinhamento múltiplos locais baseados em *motifs*

As sequências a comparar podem compartilhar regiões similares, mas não serem globalmente relacionadas. Estes módulos homólogos podem ocorrer em diferentes posições relativas e podem ser duplicados em diferentes sequências. Em tais casos não é possível computar um alinhamento global, mas pode-se procurar por "bons" alinhamentos locais.

Neste contexto, há diversas ferramentas disponíveis. Dentre estas, estão as ferramentas PRALIGN [265] e MACAW [215]. Destaca-se que o primeiro possui um requisito elevado para espaço e o segundo requisito elevado para tempo de processamento.

Os programas para alinhamento local mais recentes são baseados em métodos estatísticos, que usam heurísticas computacionalmente eficientes para resolver problemas de otimização. Por exemplo, Gibbs Sampler [147] usa *Gibbs sampling* iterativo para encontrar blocos. O tempo de computação desta abordagem cresce linearmente com o número de sequências de entrada. A ferramenta está disponível nos pacotes MACAW e Block Maker [107]. A ferramenta MEME [21] usa um algoritmo de maximização de expectativa (EM) [46, 148] para localizar padrões repetidos.

Comparação dos diferentes métodos

Quando as sequências são similares (mais que 50% de identidade de pares de proteínas e mais de 70% para DNA) e são homólogas em seu tamanho completo, todos os métodos de alinhamento global produzem resultados bons. Além disso, em tais casos, qualquer conjunto razoável de parâmetros geram alinhamentos similares. Porém, quando pelo menos duas sequências em uma dada família compartilham uma identidade menor, ou as regiões homólogas são interrompidas por *gaps* longos de tamanhos diferentes, o resultado do alinhamento pode variar consideravelmente de acordo com os programas e parâmetros utilizados.

Diversas análises comparativas de programas para alinhamento múltiplo têm sido publicadas [40, 93, 158, 169]. Estas comparações são baseadas na habilidade de detectar padrões de *motifs* em diversas famílias de proteínas ou baseadas em alinhamentos de referência derivados de estruturas tridimensionais de proteínas. Análises comparativas podem também ser baseadas no efeito dos programas de alinhamento múltiplo na filogenia.

Dependendo do tipo de entrada, há métodos mais adequados para se utilizar. Para alinhar duas sequências, deve-se usar um método ótimo de alinhamento de pares tal como Lalign ou SIM [123].

Para mais que duas sequências, geralmente tem-se de utilizar uma abordagem heurística. Como um primeiro passo, o usuário deve tentar computar o alinhamento múltiplo com um alinhador progressivo, pois são rápidos e demandam uma capacidade menor de memória. Dentre as ferramentas que utilizam esta abordagem, recomenda-se o Clustal W, que tem a capacidade de selecionar automaticamente a matriz de substituição durante o alinhamento e ajustar o peso de *gaps* ao longo do alinhamento, de acordo com a composição das sequências. Caso este primeiro alinhamento mostre que todas as sequências estão relacionadas a cada outra em seu comprimento completo, dificilmente qualquer outro método gerará um resultado melhor.

Porém, se há algumas sequências altamente divergentes, *gaps* longos ou regiões pouco conservadas, recomenda-se comparar o resultado de diferentes métodos e/ou conjuntos de parâmetros. Métodos progressivos enfrentam grandes dificuldades com longos *gaps* devido ao esquema linear para peso de *gap*, que tende a aplicar uma penalização excessiva a *indels* longos. Métodos globais baseados em blocos, como o DiAlign [168, 169] ou IterAlign [41], são menos sensíveis a esses *gaps* longos e são particularmente apropriados para tais situações.

Um outro defeito dos métodos progressivos, mas que também ocorre com os métodos de alinhamento globais ótimos, é que sempre produzem alinhamento mesmo quando as sequências não são relacionadas. Já o DiAlign e IterAlign não tentam gerar um alinhamento global se as sequências são apenas localmente relacionadas. Outra característica

Jalview	http://www.jalview.org/
CINEMA	http://utopia.cs.manchester.ac.uk/cinema
SeaView	http://pbil.univ-lyon1.fr/software/seaview.html
MPSA	http://mpsa.ibcp.fr/
Se-Al	http://tree.bio.ed.ac.uk/software/seal/

Tabela C.1: Editores de MSAs.

interessante destes alinhadores é a indicação de significância do alinhamento. No DiAlign, por exemplo, regiões que não são consideradas alinhadas (tal como em regiões não conservadas entre dois blocos alinhados) são impressas com letras minúsculas, enquanto que os resíduos alinhados estão em letras maiúsculas.

Métodos globais (ótimos, progressivos ou baseados em blocos) são apropriados apenas se todos os blocos conservados são consistentes. Se alguns domínios estão duplicados ou ordenados diferentemente ao longo das sequências é necessário usar um método de alinhamento local para alinhar todos os domínios relacionados.

C.1.4 Visualizando e editando alinhamentos múltiplos

Resultados de alinhadores múltiplo são geralmente salvos em arquivos texto. Não há um formato padrão para MSA. Entretanto, o formato MSF é provido pelos mais populares programas de alinhamento e é reconhecido por muitos programas que requerem alinhamentos como entrada, tal como filogenia molecular ou buscas de perfis. O formato MASE apresenta a vantagem de permitir a inclusão de anotações do alinhamento como um todo ou específicas a cada sequência.

Interfaces gráficas têm sido desenvolvidas para manipular e editar alinhamentos múltiplos. Geralmente permitem os usuários colorir ou sombrear resíduos de acordo com vários critérios, tais como propriedades físico-químicas, grau de conservação no alinhamento, hidrofobicidade ou estrutura secundária. O uso das cores ajuda muito na interpretação do MSA, dando uma visão muito mais abrangente da informação contida em um alinhamento múltiplo. Além disso, estas interfaces propõem diversos recursos interessantes, tais como verificação ou refino manual de alinhamentos, adição de anotações e extração de subalinhamentos.

Na Tabela C.1 são listadas ferramentas para visualização e edição de MSAs. É importante salientar que a ferramenta Clustal X adiciona uma interface ao Clustal W, que permite uma visualização gráfica do alinhamento. Entretanto, esta não permite a edição dos alinhamentos.

Se o objetivo é apenas visualização e geração de figuras para implessão, há também uma série de ferramentas com este propósito, que são listadas na Tabela C.2.

BoxShade	`http://sourceforge.net/projects/boxshade/`
WebLogo	`http://weblogo.berkeley.edu/`
MView	`http://bio-mview.sourceforge.net/index.html`
AMAS	`http://www.compbio.dundee.ac.uk/www-amas/`

Tabela C.2: Ferramentas para visualização e geração de figuras para impressão de MSAs.

C.1.5 Conclusões

A grosso modo, o melhor alinhamento consiste naquele que seja o mais parecido com o cenário evolucionário (substituições, inserções e remoções). Diversos algoritmos de alinhamento foram desenvolvidos, mas nenhum deles é ideal, ou seja, bom em qualquer situação.

Por causa de requisitos de tempo e memória, algoritmos que garantem o melhor alinhamento para um dado modelo evolucionário podem, na prática, ser utilizados apenas para um número muito restrito de pequenas sequências. Assim, diversos algoritmos heurísticos têm sido propostos. Estes reduzem os requisitos de tempo e memória, mas não garantem alinhamentos ótimos.

C.2 Recent progresses in multiple sequence alignment: a survey

Este trabalho [180] apresenta uma breve descrição das técnicas existentes (até 2002) e expõe os pontos fortes e fracos dos pacotes mais utilizados para alinhamento múltiplo. Como citado pelo próprio autor, o texto pode ser visto como um complemento ao texto de Duret e Abdeddaim [65]. Há um outro trabalho de Cédric Notredame [181] onde ele apresenta o que foi visto de novidade no contexto de MSA entre 2002 e 2007.

Alinhamento de sequências é sem dúvida a tarefa mais comum em bioinformática. Diversos procedimentos necessitam de comparação de sequências, desde buscas em bancos de dados [12] até predição de estrutura secundária [207]. Sequências podem ser comparadas aos pares numa varredura por sequências homólogas em uma base de dados ou elas podem ser alinhadas simultaneamente para visualização do efeito da evolução em uma família de proteínas inteira. Neste estudo o foco estará na comparação global simultânea de mais que duas sequências. Será dada uma maior ênfase nas técnicas mais recentes.

Alinhamentos múltiplos constituem um meio extremamente poderoso de revelar restrições impostas por estrutura e função na evolução de uma família de proteínas. Eles tornam possível a indagação de uma grande variedade de questões biológicas importantes, tais como as comentadas nos parágrafos a seguir.

Árvore filogenética é um dos instrumentos utilizado para elucidar relações evolucionárias existem entre organismos. Atualmente, a construção de árvores filogenéticas altamente precisas fazem uso de dados moleculares. Para a produção de tais árvores, geralmente tem-se os seguintes passos:

- coleta de um conjunto de sequências ortólogas;
- alinhamento múltiplo destas sequências;
- mensuração das distâncias filogenéticas de pares no alinhamento múltiplo e computação da matriz de distâncias; e
- computação da árvore pela aplicação de um algoritmo de agrupamento [210, 228] utilizando a matriz de distâncias como entrada.

Alternativamente, os dois últimos passos podem ser substituidos por uma computação de máxima semelhança [209]. Em qualquer dos casos, o papel do MSA é prover uma estimativa muito precisa de distância de pares e possibilitar estimar a confiabilidade de cada ramo através de *bootstrapping* [72].

MSAs possibilitam a identificação de *motifs* preservados pela evolução. *Motifs* desempenham um papel importante na estrutura e função de um grupo de proteínas relacionadas. Ao trabalhar com dados experimentais, estes *motifs* constituem um meio muito poderoso de caracterizar sequências de função desconhecida. Bancos de dados como PROSITE [25] e PRINTS [18] fazem uso deste princípio. No caso de um *motif* ser muito sutil para ser definido com modelos padrões, pode-se fazer uso de outros tipos de descritores tais como perfil [97] ou HMM (do inglês, *hidden Markov model*) [104]. Estes permitem uma sumarização exaustiva (coluna por coluna) das propriedades de uma família ou domínio de proteínas. Perfis e HMMs permitem a identificação de membros muito distantes de uma família de proteínas. Suas sensibilidade e especificidade são muito maiores que as providas por uma única sequência ou um modelo. Na prática, pode-se derivar um perfil de alinhamentos múltiplos usando ferramentas tais como: PFTOOLS [156], coleções pré-definidas como Pfam [31], ou computar os perfis com PSI-BLAST [14]. É importante salientar que a sensibilidade e especificidade dos perfis dependerão da qualidade biológica dos alinhamentos múltiplos dos quais oriundam.

Predição de estrutura é outro importante uso de alinhamentos múltiplos. Predição de estrutura secundária e terciária objetivam a predição do papel que um dado resíduo desempenha na estrutura da proteína (oculto ou exposto, hélice ou fita, etc.). Predições de estruturas secundárias baseadas em uma única sequência possuem uma precisão baixa (cerca de 60%) [81], enquanto que predições baseadas em MSAs são muito mais precisas (cerca de 75%) [129, 207, 206]. Isso ocorre porque padrões de substituição observados em

uma coluna refletem diretamente o tipo de restrições impostas naquela posição no curso da evolução.

No contexto de determinação de estrutura terciária ou predição de contatos não locais, alinhamentos múltiplos podem também ajudar na identificação de mutações correlacionadas. Esta abordagem tem dado resultados limitados quando aplicada a proteínas [83], mas tem tido sucesso em análises de RNA [103].

Como pode ser notado, estas importantes aplicações explicam o porque se dedica tanta atenção ao problema de MSA. Infelizmente, as ferramentas disponíveis são heurísticas que muitas das vezes provêem soluções boas, mas não necessariamente ótimas, para o problema.

Alinhamento múltiplo de sequências (MSA) é uma tarefa complexa, que implica em três dificuldades técnicas distintas: escolha das sequências, escolha da função objetivo e otimização da função.

Os métodos tratados neste texto só fazem sentido se assumido que o conjunto de sequências é composto por sequências homólogas. E quando se trata de métodos de alinhamento global, com exceção do DiAlign [168], há ainda o requisito de que as sequências sejam semelhantes em todo o seu comprimento, ou pelo menos quase todo, para que o alinhamento de resultado tenha alguma relevância. Para os casos em que não se cumpre este requisito, deve se considerar o uso de métodos para MSA locais, tais como: Gibbs Sampler [147], Match-Box [57] e MACAW [215].

Para maiores informações sobre os problemas que podem surgir devido a uma má escolha das sequências a alinhar, consultar trabalho de Henikoff [105]. É comum o uso de uma das ferramentas BLAST (WU-BLAST, PSI-BLAST, Gapped BLAST, etc.) [14] para avaliar o nível de relacionamento entre as sequências do conjunto. Tais ferramentas utilizam modelos estatísticos, desenvolvidos por Altschul e Karlin [133], para estimar o grau de relacionamento entre as sequências. Claro que estes modelos são uma mera aproximação da realidade biológica.

A função objetivo (OF, do inglês *objective function*) deve permitir uma quantificação da qualidade de um alinhamento, ou seja, dado um alinhamento de entrada, gerar um valor, indicativo da qualidade, de saída e assim permitir a comparação de alinhamentos. Como consequência permita a seleção daquele alinhamento que tenha um maior significado biológico. Dada uma OF "perfeita", o alinhamento ótimo matemático deveria ser um alinhamento ótimo biológico.

Na prática tal OF "perfeita" não existe e mesmo que existisse, a tarefa de demonstrar que é "perfeita" já se trata de um grande problema. Na teoria, uma OF deveria incorporar tudo o que é conhecido acerca das sequências, incluindo sua estrutura, função e história evolucionária. Entretando, este tipo de informação raramente está disponível e, mesmo que esteja, é de difícil utilização. Desta forma, é comum utilizar similaridade de sequência

através da simples função de soma dos pares com pesos e função afim para penalidade de *gaps* [13].

Neste modelo, cada sequência recebe um peso proporcional ao quanto de informação independente ela contém [10] e a pontuação do alinhamento múltiplo é equivalente à soma das pontuações de todas as substituições de pares com peso. As pontuações para substituições, por sua vez, são calculadas usando um modelo evolucionário predefinido conhecido como matrizes de substituição [56, 85, 106, 217], no qual uma pontuação é associada a toda possível substituição ou conservação de acordo com a realidade biológica observada. Inserções ou remoções são pontuadas através de funções afim para penalidade de *gap*, que diferenciam abertura e extensão de *gap*. Normalmente há um custo elevado para a abertura e um custo inferior para a extensão do *gap*, que aumenta lentamente de acordo com o incremento no comprimento.

Um problema que surge então é a determinação destes custos (abertura e extensão). Estes valores são determinados empiricamente e podem variar de um conjunto de sequências para outro [257]. Embora esta OF seja claramente incorreta do ponto de vista evolucionário [13], por assumir que toda sequência é ancestral de toda outra no conjunto, é popular nos métodos MSA mais utilizados [91, 151, 245], devido a facilidade de implementação. Em 2000, Gonnet e colegas propuseram uma variação de soma dos pares que parece ser mais fidedigna aos eventos evolucionários [87].

Trabalhos recentes têm feito uso de OFs que parecem menos sensíveis à penalidade de *gap* graças a incorporação de informações locais. Neste contexto pode-se citar o DiAlign [168] e o T-COFFEE [183]. O primeiro inclui avaliação baseada em segmento, já o segundo usa uma OF baseada em consistência. Outros trabalhos fazem uso de modelos ocultos de Markov (HMMs, do inglês *hidden Markov models*) [26, 104]. HMMs descrevem o alinhamento múltiplo em um contexto estatístico usando uma abordagem Bayesiana. Embora de um ponto do vista formal estes métodos parecerem soluções mais atrativas, suas performances para alinhamentos *ab initio* decepcionam. É importante, entretando, salientar que um trabalho recente [134] exibiu uma solução que faz uso de HMM e foi capaz de produzir melhores resultados que o Clustal W. Outros métodos baseados em estatística que tentam associar um *P-value* ao alinhamento múltiplo foram descritos [111, 147]. Infelizmente estas medidas são restritas a MSAs sem *gaps*.

Em um mundo ideal, uma OF perfeita deveria existir para todas as situações. Na prática, isso não ocorre e, assim, o usuário terá sempre de tomar a decisão de qual o método mais apropriado para o conjunto de sequências que dispõe.

Assumindo que se tem um conjunto adequado de sequências e uma OF biologicamente perfeita, a computação de um alinhamento múltiplo matematicamente ótimo é ainda uma tarefa complexa. O custo computacional de um método exato de MSA é tão elevado que inviabiliza seu uso para conjuntos de dados reais [131, 263]. Desta forma os algoritmos

utilizados, na prática, para MSA são heurísticas que não garantem um alinhamento ótimo como resultado.

Considerando as propriedades mais óbvias dos algoritmos, pode-se classificá-los em três categorias: exato, progressivo e iterativo. Os algoritmos exatos são heurísticas de alto grau de qualidade que entregam frequentemente alinhamentos muito próximos ao ótimo [151, 235], em alguns casos, com limitantes bem definidos. Estes possuem fortes restrições quanto ao número de sequências e OFs.

Alinhadores progressivos são, sem dúvida, os mais utilizados [53, 114, 183]. Estes têm como principal característica a dependência de uma montagem progressiva do alinhamento múltiplo [74, 116, 239] onde, duas a duas, sequências (ou alinhamentos) são alinhadas através de programação dinâmica [174]. Dentre as vantagens desta abordagem estão velocidade, simplicidade e sensibilidade razoável.

Já os algoritmos iterativos dependem de um algoritmo capaz de produzir alinhamentos iniciais. Cabe a eles a tarefa de refinar tais alinhamentos através de uma série de ciclos (iterações) até que não seja mais possível incrementar a qualidade do alinhamento. Métodos iterativos podem ser determinísticos ou estocásticos, dependendo da estratégia utilizada. Os métodos mais simples são determinísticos. Sua estratégia implica em extrair sequências, uma a uma, de um alinhamento múltiplo e então realinhar [29, 93]. É possível ainda a existência de métodos que usam estratégias mistas progressiva e iterativa ao mesmo tempo [108]. Métodos iterativos estocásticos incluem HMM [143], *simulated annealing* [139] e algoritmos genéticos [15, 45, 49, 88, 182, 269]. A principal vantagem destes é permitir uma boa separação conceitual entre processo de otimização e OF.

C.2.1 Revisão

Nos últimos anos, MSA tem sofrido drásticas evoluções com a introdução de diversos novos algoritmos e novos métodos de avaliação destes algoritmos. A Tabela C.3 lista alguns dos métodos para MSA. Dentre estes, dois novos pontos têm emergido:

- o crescente interesse em estratégias de otimização iterativa; e
- o uso de esquemas de pontuação baseados em consistência.

Nesta seção, revisa-se alguns destes novos algoritmos com suas principais características e potenciais falhas. Outro ponto importante em MSA nos trabalhos recentes são métodos baseados em HMM [26, 104]. Para maiores informações acerca de métodos HMM para MSA, consulte o livro de Durbin e colegas [64].

Nome	Algoritmo	Ref.
MSA	Exato	[151]
DCA	Exato (requer MSA)	[235]
OMA	DCA iterativo	[204]
Clustal W	Progressivo	[245]
MultiAlign	Progressivo	[53]
DiAlign	Baseado em consistência	[168]
ComAlign	Baseado em consistência	[44]
T-COFFEE	Baseado em consistência/progressivo	[183]
PRALINE	Iterativo/progressivo	[108]
IterAlign	Iterativo	[41]
PRRP	Iterativo/estocástico	[93]
SAM	Iterativo/estocástico/HMM	[66]
HMMER	Iterativo/estocástico/HMM	[143]
SAGA	Iterativo/estocástico/GA	[182]
GA	Iterativo/estocástico/GA	[269]

Tabela C.3: Alguns métodos para MSA.

Algoritmos progressivos

Alinhamento progressivo é uma das maneiras mais simples e efetivas para a realização de alinhamentos múltiplos com pouco requisito de tempo e memória. Esta abordagem foi inicialmente descrita por Hogeweg e Hesper [116] e depois reinventada por Feng e Doolittle [74] e Taylor [239]. Os pacotes mais utilizados para MSA são baseados nesta abordagem, incluindo: Pileup (componente do pacote GCG [58]), MultiAlign [53] e Clustal W [245]. Estes tornaram-se os métodos padrões para alinhamento múltiplo.

Clustal W implementa um algoritmo determinístico não-iterativo que tenta otimizar o peso de uma soma dos pares com função afim para penalidade de *gap*. É uma estratégia de alinhamento progressivo direto onde as sequências são adicionadas uma a uma de acordo com a ordem indicada por um dendrograma pré-computado. A adição da sequência é realizada através de um algoritmo de alinhamento de pares [174]. Seu principal defeito é que uma vez que uma sequência foi alinhada, nunca mais o alinhamento será modificado, mesmo em caso de conflito com uma sequência a ser adicionada ao alinhamento. Clustal W inclui uma variedade de heurísticas altamente especializadas destinadas a máxima exploração de informações acerca da sequência. Com isso, o Clustal W diferencia-se da maioria dos outros alinhadores progressivos por prover:

- penalidade de *gap* local;
- escolha automática da matriz de substituição;

- ajuste automático de penalidade para *gap*; e
- retardo do alinhamento de sequência com baixo grau de relacionamento.

Em testes de *benchmark*, realizados com o BAliBASE [249], observou-se que, em geral, o Clustal W executa melhor quando a árvore filogenética é relativamente densa, sem qualquer esboço óbvio. Para ele não importa o quão longa são as sequências, mas sim o quão similares elas são. Longas inserções ou deleções causam problema, devido a uma limitação intrinseca do esquema de penalidade de *gap* usado.

O último avanço realizado em algoritmos progressivos é o T-COFFEE, uma nova estratégia onde utiliza-se uma OF baseada em consistência. Tal função permite minimizar potenciais erros, especialmente em estágios iniciais da montagem do alinhamento. Maiores detalhes na seção sobre algoritmos baseados em consistência.

Algoritmos exatos

Como mencionado, alinhamento progressivo é apenas uma solução aproximada. Para obter uma solução precisa, é necessário alinhar todas as sequências simultaneamente. Este tipo de procedimento pode ser especialmente útil quando estiver tratando de um conjunto de sequências extremamente divergente, onde todos os alinhamentos de pares parecem estar incorretos. Infelizmente, para realizar tal tarefa necessita-se generalizar o algoritmo de Needleman e Wunsch [174] para um espaço multidimensional e por razões práticas (tempo e memória) isto torna o número de sequências aceitável no conjunto de entrada extremamente restrito.

O programa MSA faz exatamente isso, mas tenta minimizar as restrições por identificar e desprezar porções do hiperespaço que não contribuem para a solução. MSA implementa o algoritmo de Carillo e Lipman [47]. É importante salientar que MSA é apenas uma implementação heurística do algoritmo de Carillo e Lipman e não garante o ótimo matemático. Este pacote não é muito usado devido a seu alto requisito de tempo e memória, além das limitações no número de sequências.

Stoye e colegas [235] descreveram um novo algoritmo por divisão e conquista, DCA, que estendeu as capacidades do MSA original. O algoritmo DCA fragmenta as sequências em segmentos que são pequenos o suficiente para utilizar o MSA. Posteriormente, resta ao DCA montar o alinhamento a partir dos sub-alinhamentos. Testes no BAliBASE indicaram que a estratégia DCA produz resultados ligeiramente melhores que o Clustal W. É importante salientar, entretanto, que limitações ainda existem e que o DCA não foi capaz de realizar o alinhamento de quatro conjuntos de testes do BAliBASE, exatamente por estas limitações.

Recentemente, uma implementação iterativa do DCA, OMA [204], foi descrita. Através do OMA pretende-se tornar a estratágia DCA mais rápida e reduzir os requisitos de memória.

Algoritmos iterativos

Estes são baseados na idéia de que a solução para um dado problema pode ser computada por modificar uma solução sub-ótima já existente. Cada passo de "modificação" é uma iteração e modificações podem ser realizadas utilizando programação dinâmica ou vários protocolos aleatórios. Faz-se, nestes, uma distinção entre estocásticos e determinísticos. Por estocástico aqui referimo-nos a métodos estocásticos iterativos tradicionais tais como SA (do inglês, *simulated annealing*) [162] ou GA (do inglês, genetic algorithm) [117].

SA foi o primeiro método iterativo estocástico descrito para alinhamento múltiplo. Vários esquemas foram publicados [124, 139] e todos envolvem a mesma cadeia de processos: modifica aleatoriamente um alinhamento, calcula a pontuação, decide se mantém ou descarta a modificação de acordo com função de aceitação que torna-se mais exigente com o aumento no número de iterações. O procedimento é repetido até que seja satisfeito um critério de término. É importante observar que SA tem se mostrado pouco eficiente, em termos de tempo, para a construção de alinhamentos *ab initio*, mas tem se mostrado um bom *improver* de alinhamentos.

GAs constituem uma interessante alternativa aos SAs como mostrado em SAGA [182]. Assim como SA, SAGA é uma caixa preta de otimização na qual qualquer OF pode ser testada. Seu esquema é direto e segue a risca o GA simples [84]. SAGA usa a OF definida como sua função de aptidão, suas mutações inserem ou deslocam *gaps* e cruzamentos combinam o conteúdo de dois alinhamentos. Ao todo possui 20 operadores, que competem pelo uso. SAGA não garante otimalidade, mas testes têm mostrado que é capaz de produzir resultados próximos ou até melhores que o MSA (usando a mesma OF). Os testes também mostraram que GAs são lentos para alinhamentos múltiplos, no uso do dia-a-dia. É importante salientar que os operadores implementados independem da OF utilizada e, assim, é fácil realizar a substituição da OF no SAGA.

Posteriormente, Zhang e Wong [269] implementaram uma nova ferramente que faz uso de GAs. Eles reportaram um alto grau de eficiência da ferramenta, mas deve-se observar que o método é dirigido pela presença de segmentos completamente conservados para guiar a montagem do alinhamento. A suposição da sempre existência de tais segmentos não é realística. Este método parece ser apropriado para alinhamento de sequências muito longas e com alto grau de similaridade.

SAGA foi paralelizado por dois grupos independentemente [15, 185] em busca de eficiência. Novos métodos para MSA foram implementados baseando-se em princípios descritos no SAGA [45, 49, 88].

O pacote Gibbs Sampler [147] faz uso de *Gibbs Sampling*, que é uma outra interessante estratégia iterativa estocástica. Ele implementa um método de alinhamento local que encontra *motifs* sem *gaps* ao longo de um conjunto de sequências não alinhadas. Da perspectiva de alinhamento múltiplo, sua característica mais interessante está em sua OF. O algoritmo objetiva construir um alinhamento com um bom *P-value* (por exemplo, baixa probabilidade de ter sido gerado por acaso). A cada iteração, adicionam-se ou removem-se segmentos de acordo com a probabilidade do modelo corrente poder gerar eles. Se aquela probabilidade é suficientemente alta, o modelo é atualizado com os novos segmentos e o algoritmo passa a próxima iteração.

Esta idéia Bayesiana de simultaneamente maximizar os dados e o modelo é também central a HMMs [26, 104] e, desta forma, HMMs também podem podem ser treinadas por maximização de expectativa [66, 143]. Assim como GAs, HMMs apresentaram resultados decepcionantes para a montagem de alinhamentos *ab initio*. Hoje, HMMs tais como aqueles encontrados no Pfam [31] não são mais gerados a partir de sequências não alinhadas. Atualmente, os métodos estão mais inclinados a transformar um alinhamento pré-computado em um HMM e então submetê-lo a um refinamento usando HMMER [143] ou SAM [66].

O primeiro algoritmo iterativo não estocástico data da origem de MSAs [29]. Sua estratégia consiste em re-alinhamentos para corrigir possíveis erros em estágios anteriores, para tanto faz uso de algoritmos padrões de alinhamento com programação dinâmica [174]. O procedimento é encerrado quando as iterações falham consistentemente na tentativa de melhorar o alinhamento. Este é o procedimento utilizado pela maioria das estratégias descritas até o início da década de 1990.

A principal variação nestes algoritmos encontra-se na forma como as sequências são divididas em dois grupos antes de serem re-alinhadas. Por exemplo, em AMPS [29] as sequências são escolhidas de acordo com sua ordem de entrada e re-alinhadas uma a uma. Já no algoritmo de Berger e Munsen [36] a escolha é feita de maneira aleatória e as sequências são divididas em dois grupos que podem conter mais que uma sequência. A introdução da aleatoriedade na seleção dos grupos torna o algoritmo mais robusto e incrementa sua precisão. Poucos destes métodos iterativos iniciais têm sido efetivamente avaliados por ferramentas de *benchmark*, o que torna difícil uma estimativa de suas verdadeiras significâncias biológicas.

O mais sofisticado algoritmo iterativo baseado em programação dinâmica foi descrito recentemente por Gotoh [93] e é conhecido por PRRP. Trata-se de uma estratégia iterativa de duplo aninhamento com aleatorização que otimiza uma pontuação de soma dos pares com peso e função afim para penalidade de *gap*. Sua originalidade está na otimização simultânea de pesos e alinhamento. A iteração interna otimiza a soma dos pares com peso enquanto a iteração externa otimiza os pesos que são calculados em uma árvore filo-

genética estimada do alinhamento corrente [10]. O algoritmo é encerrado quando os pesos convergem. PRRP foi o primeiro programa para alinhamento múltiplo a ser extensivamente avaliado por *benchmark*, usando JOY [164], um banco de dados de alinhamentos estruturais. Posteriormente os resultados foram confirmados no BAliBASE [183, 249]. PRRP apresenta resultados significativamente melhores que a maioria dos métodos progressivos tradicionais, assim como a maioria dos métodos iterativos recentes, como pode-se observar na Tabela C.4.

Método	**Ref1**	**Ref2**	**Ref3**	**Ref4**	**Ref5**	**Total**
DiAlign	71.0	25.2	35.1	74.7	80.4	57.3
Clustal W	78.5	32.2	42.5	65.7	74.3	58.7
PRRP	78.6	32.5	50.2	51.1	82.7	59.0
T-COFFEE	80.7	37.3	52.9	83.2	88.7	68.7

Tabela C.4: Avaliação do DiAlign, Clustal W, PRRP e T-COFFEE nos 5 conjuntos de referência do BAliBASE. A pontuação aqui utilizada indica a porcentagem de colunas confiáveis, do alinhamento de referência, que foram corretamente alinhadas. As colunas Ref1 a Ref5 da tabela correspondem a uma média das pontuações obtidas pelo alinhador nos conjuntos de teste da respectiva categoria do BAliBASE. A coluna total apresenta uma média do alinhador nestas 5 categorias.

Dois outros métodos de alinhamento iterativo foram recentemente descritos: PRALINE [108] e IterAlign [41], que compartilham protocolos muito similares. Quanto a suas potenciais performances, nenhum deles foi adequadamente avaliado. Todavia deve ser dado ênfase a novos conceitos que eles incorporaram:

- uso de informação local no IterAlign, que objetiva um decremento na sensibilidade a parametrização de penalidade de *gap*; e
- o conceito de consistência.

A busca por consistência tem se tornado um dos pontos mais fortes no desenvolvimentos recentes em MSA. Tal conceito é, também, central nos métodos não-iterativos.

Algoritmos baseados em consistência

O primeiro método MSA baseado em consistência foi descrito por Kececioglu na década de 1990 [138]. Dado um conjunto de sequências, o MSA ótimo é definido como aquele que está de acordo com a maioria de todos os possíveis alinhamentos ótimo de pares. Computar tal alinhamento é um problema NP-Completo que pode ser resolvido apenas para um pequeno número de sequências relacionadas, usando um algoritmo tal como

MSA. Todavia, há pelo menos três boas razões que fazem OFs baseadas em consistência muito interessantes:

- não dependem de uma matriz de substituição específica, mas de qualquer método, ou coleção de métodos, capaz de alinhar duas sequências por vez;
- o esquema baseado em consistência é dependente de posição, dada a coleção de alinhamentos de pares. Isso significa que a pontuação associada com o alinhamento de dois resíduos dependem de seus índices ao invés de sua natureza individual; e
- experiências mostraram que dado um conjunto de observações independentes, a maioria dos consistentes estão frequentemente próximos da verdade.

Embora a primeira OF baseada em consistência tenha sido descrita em 1993 [138], passaram-se ainda alguns anos para o desenvolvimento de algoritmos heurísticos capazes de tratar sua otimização. Apenas recentemente um GA (SAGA [182]) foi usado para mostrar os avanços biológicos de tal função, denominada COFFEE [184], que emula o problema do *trace* de peso máximo. No SAGA-COFFEE, a coleção de alinhamentos de pares com peso é nomeada uma biblioteca e o SAGA é utilizado para computar o alinhamento que tem o mais alto nível de consistência com a biblioteca. Na prática, a biblioteca pode conter mais que um alinhamento para cada par de sequências. A informação que ela contém pode ser redundante, conflitante e pode se originar de fontes que variam tanto quanto se desejar (análise de estrutura, comparação de sequências, busca em bases de dados, conhecimento experimental, etc.).

Embora o SAGA-COFFEE tenha obtido resultados interessantes, o problema do desempenho ainda persistia. Isso levou ao desenvolvimento de um novo algoritmo heurístico para otimizar a função COFFEE de maneira que fosse eficiente em termos de tempo: T-COFFEE [183]. No T-COFFEE, a biblioteca COFFEE é transformada na biblioteca estendida, uma matriz de substituição para posição específica onde a pontuação associada a cada par de resíduos depende da compatibilidade daquele par com o resto da biblioteca. T-COFFEE faz uso de um procedimento reminiscente da multiplicação de matrizes de pontos de Vingron [256] e da sobreposição de pesos de Morgenstern [169]. O alinhamento múltiplo é montado usando um algoritmo de alinhamento progressivo semelhante ao Clustal W.

A principal diferença entre T-COFFEE e Clustal W está na biblioteca estendida, que assume o lugar da matriz de substituição. Outra característica importante do T-COFFEE é que a biblioteca primária é criada de uma combinação de alinhamentos globais (produzidos pelo Clustal W) e alinhamentos locais (produzidos com Lalign). Testes realizados através do BAliBASE mostraram que a combinação de informação global e local permite

ao T-COFFEE superar o PRRP, Clustal W e DiAlign nas cinco categorias de conjuntos de testes contidos neste banco de dados de referência [183].

Embora diferente nos detalhes, T-COFFEE e DiAlign [168], um outro algoritmo baseado em constência que tenta usar informação local para guiar um alinhamento múltiplo global, possuem alguma similaridade. No geral, DiAlign não é tão preciso quanto o Clustal W ou o PRRP, mas é superior nas categorias 4 e 5 do BAliBASE, que requer que inserções muito longas sejam alinhadas adequadamente [183, 249]. O algoritmo do DiAlign já sofreu modificações para melhorar a eficiência [150].

Um outro método que trabalha com consistência é o ComAlign [44], que é capaz de combinar diversos alinhamentos múltiplos em um único alinhamento, frequentemente mais preciso.

C.2.2 Conclusões

No início da década de 1990, quando os métodos para MSA começavam a se desenvolver, havia poucos dados disponíveis e o principal problema era usar todo o tipo de informação disponível. Esta situação tem se alterado dramaticamente. Hoje dispomos de uma grande quantidade de dados de uma grande variedade de tipos e facilmente acessíveis através de bancos de dados públicos disponíveis na *web*. Como consequência surgiu o desafio de ter de selecionar o que deve ser adicionado ou não como entrada no seu método. Há duas boas razões para não usar todas as sequências disponíveis:

- alinhar um grande número de sequências requer muito tempo de computação, além de ser difícil de analisar; e
- embora as soluções existente usem esquemas de peso que visam minimizar o efeito de sequências similares ou altamente correlacionadas, nenhuma delas é inteiramente satisfatória e sub-grupos super-representados acabam por dominar o alinhamento ou o perfil.

Uma trimagem cuidadosa feita pelo usuário é ainda a melhor forma disponível de minimizar tais efeitos.

Outra grande alteração que ocorreu nos últimos anos refere-se ao crescente número de estruturas 3D disponíveis. Embora a proporção de sequências de proteínas com uma estrutura 3D conhecida esteja ficando cada vez menor, a situação é muito diferente sob a perspectiva de famílias de proteínas onde, pelo menos, um membro tem a estrutura 3D conhecida, esta cresce regularmente. Isso significa que na maioria dos casos o alinhamento múltiplo pode se beneficiar da incorporação da informação da estrutura 3D objetivando reconhecer homologias remotas ou guiar a escolha de penalidades locais [125].

Poucas implementações disponíveis são capazes de combinar sequências e estruturas em um alinhamento múltiplo. Enquanto o Clustal W é capaz de usar informações de estrutura secundária do Swiss-Prot [24] para estimativa de penalidade de *gap*, falta uma ferramenta apropriada para alinhamento simutâneo de sequências e estruturas. Dois dos métodos introduzidos aqui são bons candidatos para tal tarefa. Os algoritmos baseados em consistência têm a vantagem de possuírem poucos requisitos na composição de suas bibliotecas. Por exemplo, o banco de dados de alinhamento múltiplo estrutural DALI [59] usa T-COFFEE para montar a coleção de alinhamentos de pares produzidos pelo algoritmo DALI no alinhamento múltiplo. O algoritmo de programação dinâmica dupla introduzido por Taylor [242] é outro bom candidato para este propósito. Já havia sido mostrado que tal solução era adequada para alinhamentos de estruturas [188]. Resultados recentes indicam que ela poderia ser usada no contexto de MSA e possivelmente no contexto da combinação de sequências e estruturas [70].

Um grande obstáculo na construção de MSAs é o processamento de repetições. Repetições (*in tandem* ou não) causam confusão a todos os métodos existentes para MSA. No caso de existir repetições nas sequências de entrada, a única solução é o pré-processamento das sequências com extração destes e apenas realizar o alinhamento de regiões homólogas. Esta extração pode ser realizada por ferramente de alinhamento múltiplo local tal como Gibbs Sampler [147], Mocca [179] ou Repro [110]. Infelizmente nenhum deles está bem integrado com um procedimento de alinhamento múltiplo global. Conta a favor do Gibbs Sampler e do Mocca o fato de proverem uma estimativa de relevância biológica de suas saídas.

Um quarto ponto de relevância no contexto de MSA refere-se a sua computação em si. A paralelização de algoritmos para MSA continua sendo uma tarefa difícil. As operações contidas na implementação de tais algoritmos requerem esquemas complexos de compartilhamento de memória que não são suportados por alguns *clusters*, tais como os `Linux`. Quando trabalha-se com grandes conjuntos de dados compostos por sequências longas, ainda são necessários super-computadores para a computação do alinhamento múltiplo.

Um último ponto importante é a estimativa de precisão local. Dentre todos os métodos apresentados neste texto, não há um em particular que seja o melhor. Todos podem ser superados em determinadas situações. Por esta razão, mostra-se ser mais importante ter a capacidade de calcular o nível exato de precisão de um alinhamento ao invés de melhorar as performances médias de cada método. Dentre os métodos tem-se conhecimento apenas de quatro que são capazes de realizar tal estimativa: Clustal X [244], PRALINE [108], T-COFFEE [183] e Match-Box [57]. Entretanto, é importante salientar que em nenhum dos casos estas estimativas foram apropriadamente avaliadas.

Um alinhamento múltiplo é um modelo muito restrito e uma maneira poderosa de visualizar inconsistência em conjuntos de dados, assim como permite a visualização de

relacionamentos que podem existir em pedaços de informação aparentemente independentes.

Métodos para MSA têm de evoluir. Eles precisam integrar informações heterogêneas tais como estruturas, resultados de buscas de bancos de dados, dados experimentais e qualquer informação que venha de dados de expressão e análise proteômica, incluindo informação regulatória. Integrar tais informações é uma tarefa complexa, pois quando trabalha-se com dados heterogêneos, conhecer quem está certo e quem está errado torna-se uma arte. Resolver este tipo de questão é difícil ao mesmo tempo que essencial. Métodos apropriados terão de fazer isso de uma forma transparente, permitindo ao usuário controlar todas as informações extras que serão utilizadas durante o alinhamento. Este método ideal deveria também permitir, ao usuário, adicionar ao modelo informações oriundas de seu próprio conhecimento.

De qualquer forma, estes métodos futuros continuarão a requerer muita memória e processamento. Comparado com buscas em bancos de dados, alinhamento mútiplo é difícil de otimizar. Poderá ser necessário o desenvolvimento de um *hardware* especial e consequentemente, reprojetar as implementações correntes.

Na área de genômica comparativa, a comparação simultânea de um grande número de objetos biológicos homólogos irá se tornar cada vez mais importante e continuará sendo central para análises biológicas.

C.3 Recent evolutions of MSA algorithms

Este trabalho [181] aborda os últimos trabalhos na área, onde aparecem métodos tais como meta-métodos e baseados em modelos. O mesmo autor havia publicado em 2002 um *survey* [180] com os trabalhos mais antigos.

Um número crescente de métodos para modelagem biológica têm uma forte dependência com um MSA preciso, mas a tarefa de construir um bom MSA não é trivial e nenhum dos métodos existentes consegue construir MSAs biologicamente perfeitos.

Calcular MSAs exatos é computacionalmente inviável, assim na prática há heurísticas que maximizam a similaridade. Frequentemente a relevância biológica dos resultados de um alinhador é avaliada pela comparação do alinhamento com coleções pré-estabelecidas de alinhamentos. Tais comparações são baseadas na estrutura e as coleções são chamadas "*gold standards*" [38]. Tais ferramentas têm dado efeito na evolução dos algoritmos, levando a alinhamentos estruturalmente corretos. Esta evolução também tem coincidido com uma harmonização nos algoritmos, onde agora a maioria utiliza a abordagem progressiva [116].

O esquema de pontuação é o componente de maior influência nos resultados dos algoritmos progressivos. Tais esquemas podem ser divididos em duas categorias: baseados em

matriz ou baseados em consistência. Os algoritmos baseados em matriz usam uma matriz de substituição para avaliar o custo de relacionar dois símbolos. São exemplo de algoritmos baseados em matriz o Clustal W [245], MUSCLE [68] e Kalign [146]. Os algoritmos baseados em consistência incorporam um maior conjunto de informações na avaliação. Inicialmente tal método foi utilizado no T-COFFEE [183], inspirado no DiAlign [168]. Depois surgiram outros algoritmos que utilizam a mesma idéia. O PCMA [199] reduz os requisitos computacionais exigidos pelo T-COFFEE. O ProbCons [60] adiciona o uso de consistência Bayesiana e modelos ocultos de Markov. MUMMALS [196] combina o esquema de pontuação do ProbCons com a estratégia do PCMA. Outro exemplo ainda de algoritmo baseado em consistência é o MAFFT [135]. Estudos têem indicado que os métodos baseados em consistência são mais precisos que os métodos baseados em matriz, mas eles tipicamente requerem um maior tempo de processamento.

Há também os métodos de consenso, tal como usado pelo M-COFFEE [260]. Este algortimo é um meta-método consenso baseado no T-COFFEE. Inicialmente são utilizados diversos métodos MSA para calcular alinhamentos alternativos. Então utiliza-se o T-COFFEE para calcular um MSA final, que seja consistente, utilizando-se os alinhamentos gerados inicialmente. M-COFFEE tem como vantagem uma execução mais rápida. Desconsiderando os algoritmos meta-métodos, o ProbCons é o mais rápido [260]. Quando é inserido o M-COFFEE neste grupo de algoritmos, M-COFFEE é mais rápido que o ProbCons. É importante salientar, entretanto, que o M-COFFEE enfrenta problemas a medida que as sequências se tornam menos homólogas. Sua principal vantagem é a habilidade de estimar a consistência local entre o alinhamento final e os MSAs combinados.

Para tentar resolver este problema com sequências pouco homólogas, uma alternativa apontada é incorporar informações relativas às sequências. Seguindo esta abordagem tem-se os métodos de alinhamento baseados em modelos [17], que podem ser implementados com extensão estrutural ou extensão por homologia. Extensão estrutural foi inicialmente descrita por Taylor [238]. Nele tenta-se identificar um modelo estrutural, no Protein Data Bank [37], para cada sequência usando BLAST [12]. Passa-se, então, a alinhar modelos usando um método para superposição de estruturas e mapear as sequências originais em seus alinhamentos de modelos. Tais resultados compõem uma biblioteca primária, que é enviada a um método baseado em consistência para calcular o resultado final. Há também a extensão por homologia, que foi originalmente aplicada no pacote DbClustal [251] e trabalha de forma semelhante. A diferença está no fato de usar um perfil (construído com PSI-BLAST [14]) ao invés da estrutura.

Outros pacotes que fazem alinhamento baseado em modelos são: Expresso [17], PROMALS [197], PRALINE [108], SPEM [273] e T-Lara [32]. A maioria dos métodos baseados em modelos são baseados em consistência. PRALINE e SPEM são exceções. Eles usam a abordagem progressiva. PROMALS implementa uma extensão por homologia. Os *bench-*

marks mais recentes mostram um forte desempenho do método (PROMALS) e sugerem um bom potencial para a abordagem de extensão por homologia.

Resultados de estudos recentes sugerem que os melhores métodos têm se tornado indistinguíveis, exceto em situações de homologia remota (menos que 25% de identidade). Infelizmente, homologias remotas são fracamente indicadas para a geração de alinhamentos de referência, pois suas sobreposições frequentemente aceitam diversos alinhamentos alternativos com estruturas equivalentes [144]. Visando eliminar tal dificuldade na avaliação de um método, é possível comparar diretamente o alinhamento avaliado com alguma superposição 3D idealizada, que tem sido adotada por diversos autores [17, 190, 196].

Os *benchmarks* correntes têm pecado pela ausência de conjuntos de refência com grande número de sequências, o que inviabiliza uma avaliação apropriada dos diversos métodos disponíveis quanto a suas capacidades diante de tal situação.

Um problema, levantado recentemente, refere-se a suposição de que alinhamentos estruturalmente corretos são os melhores MSAs para modelar qualquer tipo de sinal biológico (evolução, homologia ou função). Um relatório acerca de construção de perfil [99] mostrou que alinhamentos estruturalmente corretos não resultavam necessariamente em melhores perfis. É necessário sistematicamente questionar e quantificar o relacionamento entre a precisão de MSAs e a relevância biológica de qualquer modelo oriundo deles.

Alinhamento baseado em modelo é mais que uma extensão trivial de métodos baseados em consistência. Neste novo método, o propósito de um MSA é utilizar uma entrada expandida, com uma série de informações adicionais como estruturas das sequências a alinhar, como um ponto de partida para explorar e recuperar todas as informações relacionadas (às sequências) contidas em bancos de dados públicos. Desta forma, utilizando todo este conhecimento adiquirido acerca das sequências tanto com o objetivo de mapeamento quanto de guia para a computação do MSA. O uso de tais informações tornam métodos baseados em modelos uma verdadeira mudança de paradigma e um grande passo na direção de uma integração global de dados biológicos.

C.4 Multiple sequence alignments

Este trabalho [259] apresenta uma revisão, realizada por Wallace, Blackshields e Higgins, acerca das soluções para alinhamento múltiplo de sequências disponíveis até meados desta década.

Os principais métodos, ainda em uso, são baseados em alinhamento progressivo e datam da metade da década de 1980 em diante. Recentemente houveram melhorias dramáticas na metodologia no que se refere a velocidade e capacidade de trabalhar com grande número de sequências. Tem-se conquistado avanços muito significativos no que se refere a precisão dos resultados. Houve também avanços práticos na definição de métodos

para combinar informações de estruturas tridimensionais com sequências primárias, que é um dos principais fatores para a melhoria na qualidade dos alinhamentos.

Até meados da década de 1980, MSAs eram construídos manualmente, pois até então a única solução de que dispunham era uma extensão do algoritmo de programação dinâmica para alinhamento de pares [174]. Tal método era muito limitado devido a suas altas requisições de processamento e memória, o que o tornava proibitivo para casos reais.

T-COFFEE é de grande interesse não apenas por causa da forma como permite que dados heterogêneos sejam combinados em alinhamentos, como pode ser observado em seu uso no 3D-COFFEE [189], mas também como um precursor para o método baseado em probabilidade do ProbCons [60], que é o mais preciso hoje em dia. ProbCons funciona de forma semelhante ao T-COFFEE, mas usa probabilidades para as pontuações de resíduos.

3D-COFFEE [189] foi projetado para criar alinhamentos de sequências de proteína que incorporam informação de estruturas tridimensionais, quando estas existem. Em um conjunto de sequências de entrada, é comum encontrar entradas PDB [37] para uma ou mais destas sequências. O objetivo de utilizar as estruturas tridimensionais é facilitar o alinhamento de sequências com relacionamentos distantes, pois elementos estruturais são geralmente mais conservados que sequências primárias e assim permitem um bom alinhamento mesmo na *twilight zone* (menos de 25% de identidade). Na prática, usar este tipo de informação pode ser complicado, mas há trabalhos descrevendo seu uso [5].

3D-COFFEE é um método rápido, simples e preciso que explora a habilidade do T-COFFEE em incorporar informações heterogêneas para adicionar dados estruturais no alinhamento e, assim, melhorar sua precisão, mesmo quando não se dispõe de todas as estruturas. Quando é dada apenas uma estrutura, o 3D-COFFEE combina cada sequência com a estrutura usando uma ferramenta externa (FUGUE [220]) e converte a saída para um alinhamento de duas sequências. Desta forma, tem-se um conjunto de alinhamentos de pares que indicam como as sequências alinham com a estrutura e, indiretamente, como cada sequência se alinha a cada outra.

Caso duas ou mais estruturas estejam disponíveis, é possível usar um pacote de superposição de estrutura completa. O 3D-COFFEE usa por padrão o SAP [241]. Apesar de usar FUGUE e SAP por padrão, o 3D-COFFEE permite o uso de outras ferramentas do gênero.

C.4.1 Avaliação da qualidade dos alinhadores

Existem diversas ferramentas para *benchmark* em MSA na forma de bases de dados de alinhamentos pré-compilados aos quais os alinhamento gerados pelos métodos testados são comparados. Estas comparações são frequentemente realizadas pelo cálculo da fração de colunas de resíduos alinhados que são idênticas em ambos os alinhamentos (teste e

referência). Este método é conhecido como pontuação de coluna [134]. Existem diversos outros métodos. Tal abordagem para *benchmark* é atrativa devido a sua simplicidade, mas deve-se tomar cuidado para, na busca por melhorar o alinhador, não parametrizá-lo de forma a apresentar bons resultados nos conjuntos de referência das ferramentas para *benchmark* e degradar a performance em situações gerais.

Dentre as soluções para *benchmark*, BAliBASE [246] foi o primeiro construído com o propósito de *benchmarking* em larga escala e continua sendo regularmente utilizado hoje em dia. Seus conjuntos de dados são sub-divididos em diversos conjuntos de referência manualmente refinados, sendo que cada um deles destina-se a avaliação de um problema específico em MSA (alinhamentos globais/locais de diferentes tamanhos e identidade de sequências, longos *gaps* internos, etc.). Estas são suas principais vantagens e grandes diferenças, que o tornam a principal ferramenta utilizada pelo comunidade científica.

HOMSTRAD [165] compreende um conjunto de mais de 1000 alinhamentos, cada um deles baseado em uma família de proteínas em particular. É exclusivamente baseado em sequências com estruturas tridimensionais e arquivos PDB conhecidos. Em cada entrada, um alinhamento estrutural das proteínas é automaticamente gerado. Pode-se utilizar estes alinhamentos para *benchmark*. Possui uma menor cobertura, se comparado ao BAliBASE.

PREFAB [69] é também automaticamente gerado. Duas proteínas com estruturas conhecidas são estruturalmente alinhadas e suas sequências são usadas para consultar um banco de dados, onde resultados com altas pontuações são selecionados. Os parâmetros de consulta e seus resultados são combinados e então alinhados usando o *software* que está sendo testado. A precisão é calculada apenas no par original, pela comparação com seu alinhamento/superposição estrutural.

O banco de dados SABmark [261] contém dois grandes sub-bancos de dados de alinhamentos de até 25 sequências de entrada cada. Há 425 grupos representando famílias de proteína com identidade entre 25% e 50%. Há ainda 209 grupos para avaliação de *twilight zone*. Cada um destes grupos representa uma dobra de proteína definida pelo SCOP [171] e as sequências compartilham menos de 25% de identidade.

Já a ferramenta IRMbase [236] constrói seu conjunto de referência implantando *motifs* gerados pelo ROSE [234] em sequências aleatórias. Provê diversos grupos de alinhamento que variam no tamanho e no número de implantes.

Diferente de todos os métodos de avaliação citados, APDB [190] não recai na comparação de alinhamentos de referência pré-existentes. Ao invés disso, a qualidade é mensurada com base na superposição de estruturas PDB (das sequências de entrada) conhecidas. Sua grande vantagem está na independência de alinhamento de referência e, assim, evita qualquer chance de definir métodos MSA otimizados para conjuntos específicos de referência. Sua grande desvantagem encontra-se na dependência da existência das entradas PDB.

C.4.2 Novos pacotes para alinhamento

MAFFT [135] usa transformada rápida de Fourier para gerar uma árvore guia para alinhamento progressivo. Uma estratégia de iteração baseada em árvore é então utilizada para refinar o alinhamento pela otimização de uma função objetivo de soma dos pares com peso. Este protocolo resulta em alinhamentos muito rápidos e precisos, conforme avaliado pelo BAliBASE [136].

PSI-PRALINE [223] melhora a qualidade dos resultados do PRALINE [108] pela inclusão de sequências homólogas (àquelas do conjunto de sequências de entrada) obtidas pelo PSI-BLAST [14].

ProbCons [60] é atualmente o método MSA mais preciso, conforme avaliação realizada pelo BAliBASE. Ele obtém os melhores resultados em todos os cinco conjuntos de referência providos pela ferramenta de avaliação. De uma forma geral, ProbCons funciona de forma semelhante ao T-COFFEE, mas usa probabilidade ao invés da heurística para pesos de pares de resíduos. Para tanto usa HMM de pares de sequências gerados por uma função objetiva de máxima precisão esperada [121]. Faz uso de um algoritmo de alinhamento progressivo, seguido de um procedimento iterativo de refinamento.

MUSCLE (do inglês, *MUltiple Sequence Comparison by Log Expectation*) [68, 69] é um novo pacote de alinhamento progressivo que é extremamente rápido e preciso. Seu primeiro passo é gerar um alinhamento grosseiro através de uma árvore guia muito imatura. As distâncias entre os pares de sequências de entrada são estimadas usando contagem k-mer para um alfabeto restrito [67]. São estas distâncias que, através de um algoritmo de agrupamento, geram a árvore guia inicial. MUSCLE implementa a pontuação LE (do inglês, *log expectation*) para alinhar perfis durante o alinhamento progressivo. O próximo estágio no processo é o refinamento do alinhamento pela geração de uma árvore guia mais precisa, baseando-se no alinhamento inicial. LE tem apresentado bons resultados em buscas por homologia [252]. Nas versões mais recentes do MUSCLE foi adicionado um refinamento iterativo utilizado pelo ProbCons.

C.4.3 Conclusões

MSAs são largamente utilizados para diversos procedimentos e qualquer melhoria em seus métodos pode resultar em um impacto significativo na comunidade científica. Entretando, há um limite para a precisão que os algoritmos baseados em sequência podem chegar.

Assim, para o caso particular de alinhamento de proteínas, os métodos tendem e devem ser sensíveis e flexíveis de tal forma que permitam a incorporação de uma diversidade de tipos de informações oriundas de diversas fontes, tais como coleções de alinhamentos múltiplos existentes ou estruturas de proteína.

No caso de alinhamentos de sequências de RNA e/ou DNA, a situação é completa-

mente diferente. Não há ferramentas para *benchmark* apropriadamente estabelecidas para avaliações e comparações das soluções.

Neste contexto, a situação melhora bastante quando se trata de sequências de DNA longas, que recaem em problemas como alinhamento de genomas [39, 42, 43], busca por *motifs* [20, 79, 224] ou melhor alinhamento local [270]. Todas estas são áreas muito ativas de pesquisa.

C.5 BAliBASE 3.0: latest developments of the multiple sequence alignment benchmark

Este trabalho [246] apresenta a versão 3.0 do BAliBASE. Tal ferramenta é muito utilizada para avaliação de sistemas destinados a alinhamento múltiplo de sequências de proteína [136, 68, 264, 157, 178, 8, 145, 109, 122, 120, 134]. Além da qualidade de seus alinhamentos, o BAliBASE provê conjuntos de referência, o que permite avaliar o desempenho do alinhador em diversas situações e assim rapidamente detectar onde estão suas principais deficiências. Através de seu *web site* [247] é possível obter os conjuntos de referência e as ferramentas para a execução automática dos testes.

Diversos são os complicadores que um alinhador deve enfrentar na realização de sua tarefa de forma adequada, dentre eles pode-se citar: grandes proteínas multidomínio, em especial de organismos eucariotos; sequências heterogêneas, que podem induzir diversos erros de alinhamento; além de erros durante o processo de sequenciamento, que inevitavelmente levam a sequências incorretas. Fica cada vez mais evidente que soluções que utilizem um único método tendem a não ser capazes de lidar bem com todas estas situações, gerando assim resultado de qualidade inferior ao de soluções que empreguem estratégias cooperativas.

Diversas soluções têm surgido usando esta abordagem. Uma delas, que parece interessante, é combinar algoritmos globais e locais para produzir um único alinhamento [60, 149, 167, 183, 251]. Outro recurso que tem melhorado significativamente o desempenho do alinhadores é o uso de métodos para identificação rápida de regiões de alta similaridade nas sequências [68, 136]. Os *benchmarks* têm um papel de grande importância. Eles permitem uma avaliação precisa das soluções que se apresentam mais efetivas na solução do problema.

Neste contexto, BAliBASE não está sozinho. Recentemente diversos outros *benchmarks* para alinhamento múltiplo de sequências de proteína têm sido propostos. Dentre eles pode-se citar: OXBench [202], PREFAB [69] e SABmark [261]. Todos com grande conjunto de teste gerado por métodos automáticos de superposição de estrutura. Outra opção ainda é o HOMSTRAD [165], que apesar de não ter sido concebido com tal

finalidade, tem sido utilizado.

C.5.1 Visão geral

BAliBASE provê alinhamentos de referência de alta qualidade baseado na combinação de superposição de estruturas 3D com passo manual de validação e refinamento dos alinhamentos. Nesta nova versão são incluídos novos casos de teste mais desafiadores e que correspondem a problemas reais, encontrados ao alinhar grandes conjuntos de sequências complexas. Foi incluído também um novo protocolo semi-automático de atualização, que facilita a incorporação de novas famílias de proteína e conjuntos de referência, permitindo assim atualizações mais frequentes do *benchmark*. O processo é semi-automático devido ao passo manual de verificação e refinamento, essencial para a manutenção da qualidade na base de dados do *benchmark*.

No BAliBASE 3.0 há cobertura da maior parte dos tipos de dobra presentes nas proteínas. São 6.255 sequências em sua nova base, contra 1.444 da anterior. São 217 alinhamentos, contra 141. Há agora, em todos os casos de teste, sequências *full-length*, que tornam a tarefa do alinhador, seja global ou local, ainda mais difícil. A variabilidade das sequências nos conjuntos de teste também foi aumentada, devido a exclusão dos alinhamentos mais conservados.

Os alinhamentos são organizados em conjuntos de referência projetados para representar problemas reais em alinhamento múltiplo, são eles:

Referência 1: contém alinhamentos de sequências equidistantes. É subdividido em dois subconjuntos, de acordo com os níveis de variabilidade das sequências. Até a versão anterior eram três, mas o terceiro foi excluído porque não apresentava grande dificuldade para os alinhadores e assim não agregava informação útil;

Referência 2: contém famílias alinhadas com uma ou mais sequências "orfãs" altamente divergentes;

Referência 3: contém subfamílias divergentes;

Referência 4: contém sequências com grandes extensões de terminais N/C;

Referência 5: contém sequências com grandes inserções internas;

Referência 6: contém sequências com regiões de transmembranas;

Referência 7: contém sequências com repetições; e

Referência 8: contém sequências com domínios invertidos.

O protocolo de atualização inclui um definição automática de regiões "*core block*" em cada alinhamento. Tais regiões são aquelas que podem ser confiavelmente alinhadas, ou seja, desconsideram-se regiões ambiguas, que não podem ser estruturalmente sobrepostas.

C.5.2 Construção das referências

Em sua segunda versão, o BAliBASE possui um total de 141 alinhamentos múltiplos. Destes, há 82 alinhamentos de famílias de proteína na Referência 1 com um total de 532 sequências extraídas da base PDB [37]. Para a geração dos conjuntos de referência da versão 3.0 foram usadas estas sequências juntamente com famílias da classe multidomínio do SCOP (do inglês, *Structural Classification of Proteins*) [171].

Para cada família de proteína, realiza-se uma etapa de superposição de cada um dos seus membros com estruturas 3D conhecidas. Esta etapa, que resulta em um alinhamento de estrutura primária da família, é realizada pelo seguinte protocolo: (a) realiza-se uma busca no PDB, usando PSI-BLAST [14, 214], para cada membro da família. No caso de famílias que não estavam incluídas na versão anterior realiza-se a busca com a única sequência selecionada do SCOP. Filtram-se apenas as sequências PDB com $E < 10^{-3}$. (b) Visto que em estudo anterior [249] os alinhadores tinham facilidade em gerar resultados de boa qualidade quando as sequências possuíam mais de 40% identidade de resíduos, remove-se as sequências com identidade $> 40\%$. Neste passo utiliza-se o MAFFT [136] para determinar tais sequências. (c) Então constrói-se um alinhamento de referência com as sequências resultantes. Utiliza-se neste passo o programa SAP [241] para superposição 3D. Tal seleção foi feita com base em critérios de confiabilidade e funcionalidade. (d) O alinhamento de estrutura gerado é então manualmente verificado e refinado.

No passo anterior, constrói-se um alinhamento de estrutura primária para cada família. Em tais alinhamentos tem-se pequenos conjuntos de sequência PDB divergentes, que pode ser alinhado com precisão através de superposição de estruturas 3D. Visando a criação de conjuntos de teste maiores, foram incorporadas sequências de estrutura desconhecida oriundas do Uniprot [52]. Agora inclui-se a cada grupo as sequências Uniprot com alto grau de similaridade a pelo menos uma das sequências no alinhamento de sequências primárias, pois só assim são construídos alinhamentos precisos. O procedimento é o seguinte: (a) realiza-se uma busca BlastP [14] para cada uma das sequências no alinhamento de sequências primárias. Filtram-se as sequências com $E < 10^{-3}$. (b) Removem-se as sequências com mais de 80% de identidade de resíduos. (c) Por não dispôr de informações estruturais acerca das sequências Uniprot, filtram-se aquelas com mais de 40% de identidade com pelo menos umas das sequências PDB. (d) Uma vez alinhadas as sequências resultantes, realiza-se a verificação e refinamento manual. O resultado desta etapa constitui o alinhamento múltiplo primário.

A partir dos alinhamentos múltiplos primários, constrói-se automaticamente os conjuntos de referência de 1 a 5. Para o primeiro conjunto são selecionadas sequências equidistantes que não possuam grandes extensões internas (mais que 35 resíduos). Em V1 são alocadas as famílias com, pelo menos, quatro sequências onde quaisquer dois membros possuem menos de 20% de identidade de resíduos. Em V2 são alocadas as famílias com, pelo menos, quatro sequências onde quaisquer dois membros possuem identidade de resíduos entre 20 e 40%.

Para o segundo conjunto são selecionadas famílias cujas sequências possuem mais de 40% de identidade de resíduos e que, pelo menos, uma das sequências possua a estrutura 3D conhecida. Adicionam-se orfãs ($< 20\%$ de identidade) à cada família. Tais orfãs só podem ser sequências PDB, pois só assim haverá garantia de um alinhamento preciso.

Para o terceiro conjunto são selecionadas subfamílias tais que as sequências de uma mesma subfamília possuam $> 40\%$ de identidade, mas quaisquer duas sequências de subfamílias distintas possuam $< 20\%$ de identidade. Cada subfamília deve possuir, pelo menos, uma sequência PDB. Para as referências 1-3, o percentual de identidade é calculado sobre a região homóloga apenas. Nestes conjuntos não há sequências com grandes inserções internas.

Para as referências 4 e 5 são selecionadas sequências com $> 20\%$ de identidade com, pelo menos, uma outra sequência da família. No quarto conjunto são incluídas as sequências com grandes extensões de terminais N/C. No quinto conjunto adicionam-se as sequências com grandes inserções internas.

A seguir definem-se os "*core blocks*", regiões confiavelmente alinhadas, para cada alinhamento. Para sua definição é utilizado um método automático baseado na combinação de superposição de estrutura secundária e conservação de sequência. NorMD [250] é utilizado para mensuração da conservação das sequências.

Há ainda um último passo onde são adicionadas informações acerca de estrutura e funcionalidade das sequências. Para cada sequência PDB utiliza-se DSSP [132] para calcular elementos da estrutura secundária. Para as sequências Uniprot são extraídas entradas da FT (do inglês, *Feature Table*), além de realizar buscas por domínios da família de proteína no Pfam [250].

C.5.3 Conclusão

Comparações dos métodos no BAliBASE (ainda em sua versão 1.0) mostraram que a maioria das soluções alinham com sucesso sequências que possuam mais que 40% de identidade de resíduos, mas há uma grande perda de precisão quando esta identidade cai para menos de 20% (*twilight zone*) [249]. Tal estudo também mostrou que métodos globais, em geral, funcionam melhor quando as sequências possuem tamanho similar, mas

os métodos locais são superiores para o caso de identificação de *motifs* mais conservados em sequências com grandes extensões e inserções.

O uso de *benchmarks* para alinhamento múltiplo de sequências de proteínas é de grande importância na avaliação da efetividade das ferramentas e na comparação de seus desempenhos. Os diferenciais do BAliBASE neste contexto ficam por conta de sua alta qualidade, garantida pela verificação e refinamento manual dos alinhamentos por especialistas, e por sua organização dos conjuntos de teste, que permite avaliar o alinhador em diversas situações específicas.

C.6 A comprehensive comparison of MSA programs

Este trabalho [249] apresenta o primeiro estudo sistemático dos programas de alinhamento múltiplo mais comumente utilizados até então (1999). Para a realização de tal estudo utilizou-se, como casos de teste, alinhamentos providos pela ferramenta de *benchmark* BAliBASE 1.0 [248].

Tem se tornado uma prática comum o alinhamento múltiplo de grande número de sequências de proteína, onde neste estão inseridas sequências muito divergentes e proteínas multi-domínio frequentemente com grandes extensões de terminais N/C ou inserções internas. Outra prática que tem se tornado comum é o alinhamento de uma única sequência divergente (tipicamente de um eucarioto) com um grande grupo de sequências fortemente relacionadas (tipicamente de procariotos). O desenvolvimento de programas de alinhamento múltiplo confiáveis e precisos capazes de tratar estas situações é então de grande importância.

Há poucas comparações disponíveis acerca das performances relativas e confiabilidade das ferramentas para MSA. Doze alinhadores progressivos, dentre globais e locais, foram comparados usando alinhamentos de quatro domínios de proteínas diferentes como casos de teste [158]. Em geral métodos globais obtiveram resultados melhores que métodos locais nos testes, mas o desempenho de todos os programas foram afetados pelo número e o grau de identidade das sequências, assim como pelo número de inserções/ deleções no alinhamento.

Sete servidores de alinhamento múltiplo na web, cobrindo vários métodos globais e locais, também foram comparados para avaliar suas habilidades em identificar regiões confiáveis no alinhamento [40]. Entretanto não há estudos e comparações abrangentes dos diversos novos algoritmos. A falta de um conjunto padronizado de alinhamentos de referência tem impedido uma avaliação apropriada dos programas existentes, assim como ainda não foi possível mensurar com precisão os ganhos obtidos com os novos métodos iterativos.

Recentemente, um banco de dados de alinhamentos de referência para *benchmark*,

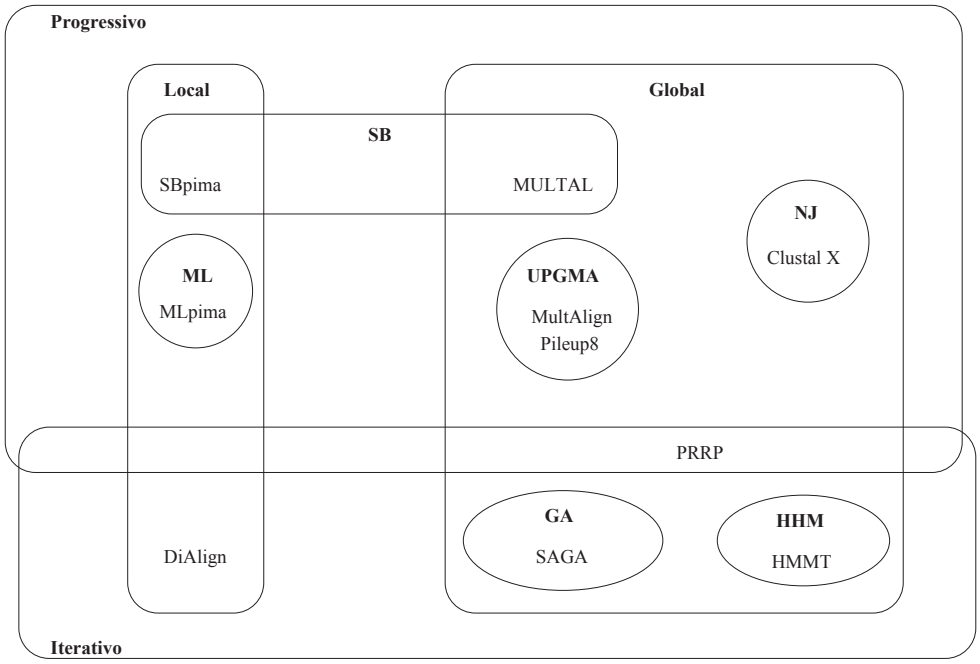


Figura C.1: Apresentação esquemática da relação entre os diferentes alinhadores e métodos. Os nomes dos métodos utilizados estão em negrito e seu escopo é delimitado por uma caixa, círculo ou elipse. Dentre eles tem-se: progressivo, iterativo, local, global, ML (do inglês, *Maximum Linkage*), SB (do inglês, *Sequential Branching*), UPGMA, NJ (do inglês, *Neighbor Joining*), GA (do inglês, *Genetic Algorithm*) e HMM (do inglês, *Hidden Markov Model*). A partir da figura é possível extrair que PRRP é um alinhador que utiliza as abordagem progressiva e iterativa para construir um MSA global, assim como, SBpima é um alinhador múltiplo local que constrói o alinhamento progressivamente por *sequential branching*.

chamado BAliBASE [248], foi desenvolvido especificamente para este propósito. Os 142 alinhamentos de teste validados de proteínas reais baseado em superposição tridimensional são organizados em conjuntos de referência, que representam alguns dos problemas mais comuns em MSA. Para cada alinhamento são definidos *core blocks*, que definem aquelas regiões que podem ser confiavelmente alinhadas.

Neste trabalho apresenta-se uma análise e comparação sistemática dos principais programas de alinhamento em uso atualmente (enumerados na Figura C.1), usando os alinhamentos de referência do BAliBASE como casos de teste.

C.6.1 Material e Método

Todos os programas foram instalados em um computador DEC Alpha 6100 executando OSF Unix e cada programa foi testado usando os parâmetros padrões (exceto para o

PRRP, onde utilizou-se a opção -b para indicar que as sequências de entrada não foram pré-alinhadas).

Alinhamentos de referência

O BAliBASE contém 142 alinhamentos de referência, divididos em cinco conjuntos de referência hierárquicos onde cada um possui pelo menos 12 alinhamentos representativos. Definem-se regiões, chamadas *core blocks*, onde pode-se alinhar confiavelmente. Tais regiões, que representam 58% dos resíduos nos alinhamentos, excluem trechos ambíguos ou não superpostos tridimensionalmente.

A primeira referência consiste de alinhamentos de pequeno número de sequências equidistantes e de tamanho similar. Não possuem grandes extensões ou inserções. Este teste avalia o efeito do tamanho da sequência e do percentual de identidade no desempenho do alinhador e provê a base para os outros testes.

A segunda referência contém alinhamentos de uma família (sequências fortemente relacionadas com mais que 25% de identidade) com a adição de até três sequências "órfãs" (membros distantes da família com menos que 20% de identidade, que compartilham uma dobra em comum). Foi projetado para avaliar a precisão do programa de acordo com dois critérios: a estabilidade do alinhamento da família quando órfãs são introduzidas no conjunto de sequências e a qualidade do alinhamento das sequências órfãs. O programa MULTAL foi removido deste teste pois frequentemente excluiu as sequências divergentes indicando tratarem-se de sequências não relacionadas ou não alinháveis.

A terceira referência demonstra a habilidade do programa de alinhar corretamente famílias aproximadamente equidistantemente divergentes em um único alinhamento. Os alinhamentos de referência consistem de até quatro famílias, com menos de 25% de identidade entre quaisquer duas sequências de famílias diferentes. MULTAL também não foi incluído nos testes com a terceira referência devido a um problema semelhante ao da segunda referência.

A quarta e quinta referências contêm sequências com grandes extensões de terminais N/C e inserções internas, respectivamente. Para avaliar a habilidade do programa de identificar a presença de inserções, os *core blocks* no BAliBASE definem apenas os mais conservados *motifs* delimitando extensões e inserções. Estes testes não foram projetados para julgar a qualidade de um alinhamento completo. Mais uma vez MULTAL teve de ser removido. Desta vez HMMT também foi excluído, pois muitos dos alinhamentos contêm apenas um pequeno número de sequências.

Pontuações para alinhamento

Para calcular o desempenho dos programas nestes estudo, estabeleceu-se duas pontuações distintas, que estimam a qualidade de uma alinhamento comparando os resultados aos alinhamentos de referência do BAliBASE. A pontuação SPS (do inglês, *sum-of-pairs score*) é calculada de forma que tenha seu valor acrescido de acordo com o número de sequências alinhadas corretamente. É usada para determinar a extensão dos alinhamentos entre pares de sequências em que se obteve sucesso. A pontuação CS (do inglês, *column score*) é uma pontuação binária que avalia a habilidade dos programas em alinhar todas as sequências corretamente.

Suponha que se dispõe de um alinhamento de teste com N sequências e M colunas. Seja $A_{i1}, A_{i2}, \ldots, A_{iN}$ a i-ésima coluna do alinhamento. Para cada par de resíduos A_{ij} e A_{ik} define-se p_{ijk} tal que $p_{ijk} = 1$ se A_{ij} e A_{ik} estão alinhados no alinhamento de referência e $p_{ijk} = 0$, caso contrário. A pontuação S_i para a i-ésima coluna é definida como:

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

A pontuação SPS é definida por:

$$SPS = \frac{\sum_{i=1}^{M} S_i}{\sum_{i=1}^{M_r} S_{ri}},$$

onde M_r é o número de colunas no alinhamento de referência e S_{ri} é a pontuação S_i para a i-ésima coluna do alinhamento de referência.

Define-se $C_i = 1$ se todos os resíduos na i-ésima coluna do alinhamento são os mesmos no alinhamento de referência e $C_i = 0$ caso contrário. A pontuação CS é dada por:

$$CS = \frac{\sum_{i=1}^{M} C_i}{M}$$

Para cada um dos testes de referência selecionou-se a função de pontuação mais adequada de acordo com a natureza do teste.

Métodos Estatísticos

Em cada referência, BAliBASE provê um número de alinhamentos representativos que são usados como uma amostra em análises estatísticas. Para cada alinhamento de referência calcula-se uma pontuação estimando a precisão do alinhamento produzido por todos os programas. Como é esperado que a distribuição das pontuações não seja normal ou simétrica, utiliza-se a mediana como uma medida de localização e o intervalo interquartil

como uma medida de dispersão. O primeiro e o terceiro quartis indicam a forma da distribuição.

Foi utilizado o teste de Friedman [78] para estimar se havia um padrão sistemático na forma como os programas eram classificados pela pontuação em todos os alinhamentos. Em outras palavras, verificar, por exemplo, se algum dos programas tende a executar significantemente melhor que outros nos alinhamentos de referância.

Os testes de Wilcoxon [267] foram utilizados para determinar se uma alteração nas condições de um alinhamento, tal como a adição de órfãos (referência 2) ou um incremento no número de famílias (referência 3), interferem de forma significativa na pontuação. HMMT foi removido deste teste por gerar soluções distintas para uma mesma entrada, o que impede uma comparação confiável.

C.6.2 Resultados

Referência 1: pequeno número de sequências equidistantes

Regiões ambíguas representam 42% dos resíduos no BAliBASE e somam, em média, 32, 22 e 11% das pontuações *full-length* calculadas nas categorias V1 ($< 25\%$ ID), V2 ($20 - 40\%$ ID) e V3 ($> 35\%$ ID), respectivamente.

Construiu-se um gráfico com os programas no eixo X, a precisão no eixo Y e com três curvas, respresentando V1, V2 e V3. A precisão no caso refere-se a mediana das pontuações para *core blocks*. A característica mais marcante do gráfico é uma nítida perda de precisão a medida que o grau de identidade entre as sequências diminui. A queda na precisão é mais marcante ainda na categoria V1, que refere-se a chamada *twilight zone*. Mesmo assim, é possível ainda realizar algum alinhamento abaixo da *twilight zone*.

O melhor alinhamento em V1 foi obtido pelo PRRP, com 72% do total de resíduos alinhados corretamente. Os programas que melhor pontuaram (PRRP, Clustal X e SAGA), alinharam corretamente, em média, 61% dos resíduos (ou 42% das colunas) nos *core blocks* e 47% do total de resíduos (ou 26% do total de colunas) em V1. Em contraste, para V2 92% dos resíduos (ou 87% das colunas) em *core blocks* e 82% dos resíduos (ou 72% das colunas) do total de resíduos foram alinhados corretamente.

As maiores diferenças nas pontuações dos programas são sempre observadas na categoria V1. De acordo com o teste de Friedman, PRRP classifica-se significantemente melhor que os outros programas. Clustal X e SAGA classificam-se em seguida. Observa-se também que, no geral, programas de alinhamento global executam melhor que os métodos locais.

Ao construir um gráfico onde as três curvas representam as sequências curtas (< 100 resíduos), médias ($200-300$) e longas (> 400) de V1, pode-se observar que para sequências longas os programas locais obtêm uma qualidade de alinhamento similar aos globais. A

única exceção ficou por conta do PRRP, que obtém resultados significantemente melhores que os outros programas de acordo com o teste de Friedman. No geral, *core blocks* são melhor alinhados em sequências médias e longas que em sequências curtas. A exceção fica por conta do Clustal X, que funciona melhor para sequência médias e curtas.

Já uma análise das pontuações *full-length* revela que: (a) há uma queda nas pontuações *full-length* comparadas às pontuações de *core blocks*; (b) a precisão dos resultados de programas globais cai com o aumento no tamanho das sequências; (c) já a pontuação de programas locais mantém a tendência de crescer a medida que o tamanho da sequência cresce.

Observou-se que os programas tendem a alinhar melhor *core blocks* em sequências longas que em sequências curtas. Tal fato surpreendeu. Isto pode ser provocado pelo tamanho da sequência em si ou por uma diferença na natureza dos *core blocks* em cada categoria de tamanho. Para investigar a causa, comparou-se os alinhamentos de tamanhos médios na categoria V1 com uma nova população de sequências curtas criadas artificialmente pela divisão de sequências de tamanho médio em duas com metade do tamanho. É importante salientar que algumas divisões foram deslocadas para não dividir *core blocks*. Utilizando-se o teste de Wilcoxon observou-se que, desta vez, os *core blocks* alinharam melhor em sequências curtas que nas sequências médias. Deduziu-se que a redução da precisão observada em sequências curtas na referência 1 não se deve simplesmente ao tamanho das sequências.

Referência 2: família e sequências órfãs

Para este teste utilizam-se famílias pequenas com 4 sequências e famílias maiores com um número de sequências variando de 14 a 22. Inicialmente as famílias foram alinhadas sem as sequências órfãs. Posteriormente foram construídos os alinhamentos com uma, duas e finalmente três órfãs (com identidade de 10 a 20% sempre que comparada com um membro da família). Para cada caso, calculou-se a pontuação SPS para o programa.

Surpreendentemente um teste de Wilcoxon indicou a ausência de uma redução significativa nas pontuações dos alinhamentos. Há, entretanto, um pequeno número de casos onde observou-se perda de qualidade de até $6,9\%$ para famílias grandes e perda de $23,6\%$ em famílias pequenas.

Avaliando as pontuações SPS para alinhamentos com uma única sequência órfã observou-se que programas de alinhamento global executam melhor que os locais. Entretanto, é importante salientar que agora Clustal X e SAGA obtêm melhores resultados que PRRP. Um teste de Wilcoxon realizado indica que há uma melhora significativa no alinhamento quando as famílias são maiores.

A habilidade de alinhar uma sequência órfã, para todos os programas, é também afetada pela presença de outras órfãs no conjunto de sequências. Entretanto, uma exata

correlação ainda não está clara. Percebe-se apenas que, dependendo do grau de relacionamento entre as órfãs e do relacionamento entre as órfãs e a família, o alinhamento por ser melhorado ou deteriorado.

Referência 3: diversas famílias em um único alinhamento

Para este teste foram utilizados pequenos números de famílias divergentes. A pontuação CS é usada neste teste, pois ela é uma melhor estimativa para alinhamento de famílias.

Utilizando um teste de Fredman, chegou-se a conclusão de que as estratégias iterativas do PRRP e do SAGA executaram melhor neste teste que os métodos de alinhamento progressivo tradicionais. O Clustal X, apesar de executar pior que PRRP e SAGA, executa melhor que os outros métodos progressivos. No geral, métodos globais apresentaram melhores resultados que os locais.

Foi ainda realizado um teste para verificar se o desempenho dos programas era melhor no alinhamento de famílias equidistantes ou no alinhamento de sequências equidistantes. Para realizar tal avaliação foi construído um novo conjunto de testes pela seleção de uma sequência de cada família nos alinhamentos da referência 3. Uma comparação nas pontuações obtidas mostrou que, no geral, famílias são melhor alinhadas que sequências individualmente. A exceção aqui ficou por conta do MLpima e SBpima.

Referência 4: extensões de terminais N/C

Todos os testes anteriores foram realizados com sequências de tamanhos similares. Agora são utilizadas sequências com grandes extensões de terminais N/C para investigar se os programas são capazes de alinhar os *core blocks* nesta situação. É importante salientar entretanto que ainda não há grandes inserções internas.

Observou-se que, utilizando um teste de Fridman e pontuação CS, três programas que implementam uma estratégia de alinhamento local (SBpima, DiAlign e MLpima) se sobressaem perante os demais. Pileup 8 é o único método baseado em alinhamento global que chega próximo aos locais neste teste. As estratégias iterativas do PRRP e do SAGA falham, devido devido a dificuldades inerentes aos métodos globais diante de tal situação.

Referência 5: inserções internas

Este teste também possui sequências de tamanhos distintos, mas desta vez as inserções são internas aos domínios homólogos. Utilizando-se a pontuação CS neste teste, observou-se que, assim como na referência 4, o DiAlign sobressaiu-se, entretando MLpima e SBpima já não obtiveram tanto sucesso desta vez e apresentaram os piores resultados, ficando atrás dos métodos globais.

C.6.3 Discussão

Um dos objetivos deste estudo foi estabelecer um sistema de avaliação (*benchmarking*) objetivo, que possa ser utilizado para comparar, avaliar e melhorar programas de alinhamento múltiplo.

Os alinhamentos do BAliBASE fornecem casos de teste reais contendo proteínas ou módulos cuja estrutura tridimensional já tenha sido determinada. Os alinhamentos são validados para assegurar o alinhamento correto de resíduos catalíticos e outros resíduos conservados. Apenas regiões que podem ser confiavelmente alinhadas, chamadas *core blocks*, estão anotadas na base e permitem uma avaliação mais precisa dos programas.

Os resultados deste trabalho deixam clara a necessidade de comparar alinhamentos utilizando tanto pontuações baseadas em sequências completas como aquelas restritas aos *core blocks*.

A *twilight zone*

Durante a realização deste estudo, observou-se que todos os programas analisados são capazes de alinhar, em média, 80% dos resíduos corretamente em um alinhamento de sequências com mais que 20% de identidade. Assim, comparações entre os programas neste nível de identidade de sequências tornam-se pouco significativos.

Observa-se, entretanto, uma importante perda na precisão quando são alinhadas sequências com identidade entre 10 e 20% (referência 1, V1). Neste caso, os melhores programas alinham corretamente, em média, apenas 47% dos resíduos. A chamada *twilight zone* [62] constitui uma barreira real para todos os programas analisados. Abaixo da *twilight zone*, os alinhamentos produzidos são frequentemente não confiáveis e com grande dispersão nas pontuações. Para sequências longas de mais de 400 resíduos em V1, os programas globais e locais podem não ser distintos no que se refere aos resultados.

Fica claro que os esforços para melhorar a qualidade dos alinhadores deve se concentrar no alinhamento de sequências abaixo de 20 ou 25% de identidade de resíduos. Todavia, é importante salientar resultados notáveis, com sequências abaixo da *twilight zone*, do PRRP, que é capaz de alinhar corretamente 27 a 72% dos resíduos.

Métodos iterativos e não iterativos

Uma avaliação das melhorias introduzidas pelos novos métodos iterativos é inconclusiva. Quatro programas destacaram-se como os mais bem sucedidos sob as condições distintas de alinhamento testadas, foram eles: PRRP, SAGA, Clustal X e DiAlign. Note que, com exceção apenas do Clustal X, todos utilizam uma estratégia iterativa para refinar o alinhamento.

Clustal X tem melhorado claramente os programas tradicionais de alinhamento progressivo, embora os parâmetros padrões possam não ser ótimos para sequências longas.

Os novos algoritmos iterativos frequentemente fornecem alinhamentos com melhores precisões, além de ter a capacidade de adiquirir conhecimento sobre as sequências a alinhar e melhorar o alinhamento, se houver informações suficientes no conjunto de dados.

É importante, entretanto, salientar que o processo iterativo pode algumas vezes ser instável na presença de ambiguidade no conjunto de sequências, tal como uma única sequência órfã. Tal situação pode levar a iteração a divergir do alinhamento correto.

Dentre os alinhadores locais, o DiAlign, que iterativamente usa um algoritmo de alinhamento de segmento local, é o mais bem sucedido.

Uma grande desvantagem das técnicas iterativas correntes é que requerem muito tempo de processamento. Como uma exemplo, para 89 sequências de histonas consistindo de 66 a 92 resíduos, o tempo de processamento requerido para o alinhamento é de $161s$ para o Clustal X, $13.649s$ para o DiAlign e $13.209s$ para o PRRP.

Métodos globais e locais

Em geral, desenvolveu-se duas classes básicas de programas de alinhamento. Programas de alinhamento global tentam alinhar sequências completas. Já os alinhadores locais buscam apenas pelos *motifs* mais conservados. Dependendo da natureza das sequências, uma classe sobrepõe-se sobre a outra.

Algoritmos globais produzem resultados mais precisos e confiáveis em situações que envolvam sequência equidistantes, famílias divergentes de sequências e o alinhamento de uma sequência órfã com uma família de sequências. Resultados semelhantes aos do trabalho de McClure e colegas [158].

Entretanto, na presença de grandes extensões de terminais N/C e inserções internas, DiAlign, que implementa uma algoritmo local de alinhamento de segmentos sem *gaps*, é a solução mais bem sucedida na tarefa de encontrar os *core blocks* altamente conservados. Os métodos globais, que tendem a gerar alinhamentos co-lineares no tamanho completo das sequências, são menos eficientes, produzindo frequentemente um alinhamento totalmente incorreto das sequências.

Uma estratégia de alinhamento melhorada

Os resultados destes testes sugerem possíveis caminhos para melhorias na precisão dos programas para famílias e sequências divergentes.

O alinhamento de sequências órfãs com uma família é mais bem sucedido se a família contém mais sequências. Entretanto, o efeito de alinhar muitas sequências órfãs simultaneamente é imprevisível, depende do grau de relacionamento entre as sequências. O

alinhamento de órfãs pode também ser melhorado se uma pequena sub-família puder ser criada para a órfã.

Parece ser uma boa estratégia incluir o número máximo possível de sequências para obter os melhores resultados. Mesmo que altamente relacionadas, sequências podem prover informações úteis adicionais.

Testes sugerem que alinhamento progressivos podem ser melhorados pela reconstrução da árvore guia durante o processo de alinhamento.

Neste estudo pôde-se observar que a escolha de um programa de alinhamento depende do conjunto de sequências a alinhar e que não há um único melhor programa. Fatores como a repartição das identidades das sequências, o tamanho e as extensões de terminais N/C afetam a precisão e a confiabilidade dos programas. Nenhum dos alinhadores incluídos neste estudo foram capazes de produzir alinhamento bons e confiáveis em todas estas situações.

Trabalhos futuros visando a melhoria dos resultados de alinhadores deve concentrar esforços em problemas como grandes inserções, extensões e fragmentos de sequência. O alinhamento de sequências de tamanho similar é relativamente bem sucedido, mesmo quando há uma baixa identidade entre elas. Outra importante área de interesse que está se tornando cada vez mais frequente é o alinhamento de famílias de sequências. Frequentemente, ao incrementar o número de sequências na entrada, a qualidade do alinhamento de sequências divergentes melhora significantemente.

Os resultados deste estudo podem ser utilizados para indicar os programas mais adequados para problemas particulares de alinhamento. Deve-se agora ser possível predeterminar a natureza do conjunto de sequências, em especial a repartição de identidade de sequências e a presença de sequências de tamanhos distintos. Um servidor de alinhamento deveria ser capaz de automaticamente selecionar o programa que muito provavelmente gerem os melhores resultados.

É importante salientar que neste estudo foram utilizados os parâmetros padrões dos programas e que o foco estava em avaliar a qualidade dos resultados. Questões importantes para a escolha de um alinhador, como facilidade de uso, portabilidade e requisitos de tempo e memória, foram desprezados.

C.7 The alignment of sets of sequences and the construction of phyletic trees: an integrated method

Este trabalho [116] foi a primeira descrição do método progressivo para alinhamento mútiplo de sequências. Nele é defendido que alinhamento de sequências e construção de árvores filogenéticas não podem ser tratados separadamente, uma vez que o conceito de

"bom alinhamento" perde sentido se não houver referência a uma árvore filogenética e a construção de árvores filogenéticas pressupõem alinhamento de sequências. Desta forma propõe-se um método integrado que gera ao mesmo tempo um alinhamento e uma árvore filogenética a partir de um conjunto de sequências. É um método iterativo que parte de uma árvore hipotética e iterativamente ajusta a árvore através de alinhamentos sucessivos de pares de sequências guiados pela árvore corrente.

Há algoritmos eficientes para alinhamento de pares de sequências [174, 218, 226, 266]. Estes utilizam alguma medida de similaridade como critério para otimização e adicionam *gaps* nas sequências durante o procedimento de busca pelo alinhamento de máxima pontuação. Desta forma, buscam maximar o número de bases idênticas pareadas. Tais algoritmos ainda utilizam uma penalização para inibir formações frequentes de *gaps*. Alguns autores desconsideram o tamanho de tais *gaps* apenas considerando o número de ocorrências, como Needleman e Wunsch [174]. Já outros consideram seu tamanho, como Sellars [218].

Um dos problemas nestes algoritmos para alinhamento de pares de sequência está no fato de produzir muitos possíveis alinhamentos com mesma pontuação. Por exemplo, podem haver diversos alinhamentos com distribuições de *gaps* distintas e mesma pontuação. O posicionamento dos *gaps* não é determinado unicamente e variações no pareamento de bases podem ocorrer em trechos onde há pequena similaridade. Uma grande preocupação aqui deve estar em selecionar, dentre aqueles com mesma pontuação, o alinhamento com maior relevância biológica. Há possibilidade de considerar a estrutura secundária das sequências nos alinhamentos, mas esta deve ser utilizada com cautela, uma vez que pode levar a uma super valorização da estrutura ao invés da ancestralidade. O mesmo método utilizado para alinhamento de par de sequências é, a priori, extensível para mais de duas sequências, mas neste caso os requisitos de tempo e espaço tornam-se impraticáveis a medida que aumenta-se o número de sequências.

A maioria dos algoritmos para construção de árvores filogenéticas tentam de alguma forma minimizar o custo mutacional ao longo dos ramos de uma árvore. Desta forma, uma das abordagens é fazer uma busca exaustiva, gerando todas as topologias possíveis e escolhendo a melhor. Não é difícil de perceber que este método torna-se rapidamente impraticável a medida que se aumenta o número de sequências e, assim, heurísticas tornam-se uma alternativa viável. Pode-se classificar tais métodos em: baseados em matriz de similaridade e baseados em caracter. Métodos baseados em matriz de similaridade são mais simples e são mais frequentemente utilizados na prática. A grande vantagem dos métodos baseados em matriz de similaridade está em precisar apenas de alinhamentos de pares de sequências, diferentemente dos baseados em caracter, que precisam do alinhamento global.

C.7.1 O método

A idéia é alinhar as sequências através de uma série de sucessivos alinhamentos de pares, que seguem os ramos da árvore inicial [76]. Então passa-se a otimizar a árvore e o alinhamento iterativamente usando a árvore corrente para construir o próximo alinhamento (árvore). Segue uma descrição mais detalhada do procedimento.

1. Alinha-se todos os pares de sequências e constrói-se uma matriz $(N \times N)$ com estes valores de similaridade.

2. Constrói-se uma árvore filogenética inicial a partir da matriz.

3. Sucessivamente alinha-se pares, seguindo os ramos da árvore. Inicia-se com as duas sequências que são mais similares e segue-se adicionando sempre a sequência mais próxima. Parte-se da premissa de que é mais confiável alinhar sequências similares. Dar-se prosseguimento utilizando o seguinte critério na construção dos nós internos: (a) se todas as m posições de uma dada coluna i possuem a mesma base x, x é assumida como a base da coluna i do alinhamento; (b) caso contrário, a decisão sobre a base a posicionar na coluna i do alinhamento é postergada. Durante o cálculo do alinhamento, nas posições ainda indefinidas utiliza-se a base que maximiza a pontuação do alinhamento.

4. O número de mutações ao longo dos ramos é calculado e assume-se estes comprimentos de ramos de árvore não alterando a topologia.

5. Calcula-se a frequência mutacional de cada posição do alinhamento da seguinte forma:

$$W_i = \frac{N_i}{M_i + 1},$$

   onde W_i é o peso da i-ésima posição, N_i é o número de nucleotídeos diferentes na posição e M_i é o número de mutações que ocorrem naquela posição. Essas frequências podem ser usadas como pesos de caracteres.

6. As sequências alinhadas são usadas na próxima execução do procedimento iterativo. Uma nova matriz de similaridade é então calculada.

7. O processo é repetido a partir do passo 2 até que haja convergência ou até que uma regra de parada é satisfeita.

C.7.2 Conclusão

Neste trabalho defende-se que a construção de árvores filogenéticas e alinhamento múltiplo de sequências são tarefas que não têm sentido se pensadas isoladamente. Sugere-se um método que ao mesmo tempo constrói árvores filogenéticas e alinhamentos múltiplos. No decorrer do tempo, tal método mostrou-se bastante eficiente e é hoje em dia o método mais empregado para alinhamento múltiplo, sendo utilizado em pacotes como o Clustal W [245].

Os dois principais problemas apontados neste método dizem respeito a convergência e parcimônia. O primeiro corresponde à dificuldade em se detectar ciclos nas árvores geradas, ou seja, dificuldade em descobrir quando os alinhamentos e árvores não irão mais ganhar em qualidade. O segundo surge pelo uso de um método não exato, que impossibilita qualquer garantia de que a árvore e o alinhamento obtidos são os melhores ou mais adequados.

C.8 Clustal W: improving the sensitivity of progressive multiple sequence alignment

Este trabalho [245] detalha o Clustal W, que é o *software* mais utilizado para alinhamento múltiplo de sequências. Esta é a terceira versão do *software*. Anteriormente foram implementadas as versões Clustal [114] e Clustal V [113].

Alinhamento múltiplo de sequências (MSA, do inglês *Multiple Sequence Alignment*) tornou-se uma ferramenta essencial em biologia molecular, como pode ser observado nas suas diversas aplicações: encontrar padrões para caracterizar famílias de proteínas, detectar homologia entre novas sequências e famílias existentes, dar suporte a predição de estruturas secundárias e terciárias de novas sequências, sugerir *primers* para PCR (do inglês, *Polymer Chain Reaction*) ou para análise evolucionária molecular.

Quando alinha-se apenas um par de sequências é comum utilizar o algoritmo de Needleman e Wunsch [174], que através de uma abordagem de programação dinâmica garante um alinhamento matematicamente ótimo, dada uma tabela de pesos (com pontuações para *matches* e *mismatches*) e penalização para *gaps* de diferentes tamanhos. Uma primeira abordagem para MSA é estender o algoritmo de Needleman e Wunsch, mas desta forma fica-se limitado a um pequeno número de sequências [151]. Isso ocorre porque a medida que o número e/ou o tamanho das sequências cresce, o problema torna-se impraticável devido aos elevados requisitos de memória ou tempo de computação. Assim, qualquer método capaz de tratar um grande número de sequências em um tempo razoável faz uso de heurísticas. A grande maioria dos algoritmos para MSA usa a abordagem progressiva de Feng e Doolittle [74], que será descrita a seguir. Tal abordagem apresenta bons

resultados quando as sequências são similares, mas quando todas as sequências são altamente divergentes (menos que 25-30% de identidade entre qualquer par de sequências), torna-se muito menos confiável.

A abordagem progressiva apresenta dois problemas principais: mínimo local e escolha dos parâmetros de alinhamento. O problema do mínimo local surge pela natureza gulosa da estratégia de alinhamento. Qualquer mal alinhamento realizado no início do processo não pode ser mais corrigido apenas com o uso da abordagem progressiva. A única forma de corrigir este problema é através do uso de um procedimento iterativo ou de amostragem estocástica [90, 147, 155]. Já o problema na escolha dos parâmetros surge do fato das escolhas das pontuações serem mais adequadas para sequências similares ou divergentes. Assim como surgem pelo fato das pontuações para *gaps* também serem sensíveis às sequências e às diferentes posições/estruturas na sequência. *Gaps* são mais frequentes em determinadas posições, que em outras.

Neste artigo são descritas melhorias para MSA progressivo, que aumenta a sensibilidade sem sacrifícios em termos de velocidade e eficiência. Tais melhorias focam na escolha dos parâmetros.

C.8.1 O método básico de alinhamento

O algoritmo progressivo básico consiste de três estágios:

1. Geração de uma matriz de distâncias a partir de alinhamentos de pares.

2. Geração de uma árvore guia a partir da matriz de distâncias.

3. Alinhamento progressivo das sequência de acordo com a árvore guia.

Para os alinhamentos do primeiro estágio, é possível optar pelo método aproximado descrito por Bashford e colegas [30] ou pelo algoritmo de Needleman e Wunsch [174]. O primeiro consome menos tempo e o segundo oferece resultados precisos. Uma vez alinhadas duas sequências, calcula-se sua pontuação dividindo o número de identidades pelo número de resíduos comparados, excluindo-se os *gaps*. O valor obtido é subtraído de 1 para gerar a distância, que é registrada na matriz.

Uma vez gerada a matriz de distâncias, construímos a árvore guia utilizando o método *Neighbor Joining* [210]. É importante salientar que versões anteriores do Clustal utilizavam o método UPGMA [228] para a construção da árvore guia. Tal método gera uma árvore onde as sequências são suas folhas e a cada aresta é associado um valor diretamente relacionado aos valores da matriz. A partir de tais valores das arestas são calculados pesos para cada uma das sequências. Os pesos são normalizados de forma que o maior deles

é redefinido para 1 e os outros são ajustados na mesma proporção do maior. A grosso modo, sequências com pesos maiores estão mais distantes das outras sequências.

Seguindo as ramificações da árvore guia e usando uma série de alinhamentos, alinha-se grupos cada vez maiores de sequências. Inicia-se por alinhar as folhas mais próximas (de menor peso, que estão em ramos diretamente ligados). Segue-se alinhando, sempre buscando os ramos ligados que possuam menor peso. A cada passo utiliza-se o algoritimo Needleman-Wunsch. *Gaps* não são removidos. Para calcular a pontuação de uma posição de uma sequência ou alinhamento e uma de outro, é calculada a média das pontuações de cada par formado por uma sequência de um grupo com uma sequência do outro. Por exemplo, se alinharmos 2 alinhamentos com 2 e 3 sequências, a pontuação de cada posição é a média de 6 comparações. As pontuações são multiplicadas pelos pesos das sequências. Comparações envolvendo *gaps* são pontuadas com 0. Os valores para *matches* e *mismatches* são sempre positivos.

C.8.2 Melhorias

A primeira parte das melhorias é a atribuição de pesos às sequências. Sequências altamente relacionadas recebem pesos menores, pois contém muita informação duplicada. Já sequências altamente divergentes recebem pesos maiores. A segunda parte das melhorias fica por conta dos *gaps*. São utilizadas duas penalizações para os *gaps*: abertura (GOP, do inglês *gap opening penalty*) e extensão (GEP, do inglês *gap extension penalty*). O GOP indica o custo de abrir um *gap*. Já o GEP indica o custo de cada item do *gap*, ou ainda, de estender um *gap*. Dados valores iniciais para o GOP e GEP, que podem ser definidos pelo usuário, o algoritmo tenta escolher valores apropriados para cada alinhamento de sequência.

Para a definição do GOP ele usa os seguintes parâmetros: tabela de *mismatches*, similaridade das sequências e tamanho das sequências. A tabela de *mismatches* é utilizada para variar as penalizações para *gaps* de acordo com a matriz de pesos, que já comprovou ser uma boa forma de aumentar a precisão do alinhamento [156, 227]. Aumenta-se o custo dos *gaps* em sequências relacionadas e diminui-se tal custo em sequências divergentes. Além disso o GOP é incrementado pelo logarítmo do tamanho da menor sequência.

Para a definição do GEP apenas utiliza-se a diferença entre os tamanhos da sequências como parâmetro. Se uma sequência é muito menor que a outra, aumenta-se o custo do GEP para inibir a criação de muitos *gaps* longos na sequência menor.

Antes de qualquer par de sequências (ou grupos de sequências alinhadas) ser alinhado, são recalculadas as penalizações para cada posição nos dois conjuntos de sequências. Se já há *gap* na posição, GOP e GEP são reduzidos e as outras regras não se aplicam. Se não há *gaps* na posição então o GOP é aumentado caso esteja a no máximo 8 resíduos de

distância de um *gap*. Esta regra inibe *gaps* muito próximo. Para qualquer posição em uma cadeia de resíduos hidrofílicos, que geralmente indicam laços na proteína, a penalização é reduzida. São geralmente considerados resíduos hidrofílicos: D, E, G, K, N, P, Q, R e S. Se a posição não pertence a uma cadeia de resíduos hidrofílicos a penalização é modificada de acordo com os resíduos [193]. Caso a posição possua uma variedade de resíduos, o fator a utilizar é dado pela média das contribuições de cada sequência.

As duas principais séries de matrizes de pesos são: PAM [56] e BLOSUM [106]. Sendo a segunda a mais comumente utilizada. Cada série provê matrizes que se aplicam às sequências de acordo com o grau de similaridade. Elas oferecem matrizes mais restritivas para o caso de sequências similares e mais flexíveis para sequências divergentes.

Para tornar possível o alinhamento de sequências compridas utiliza-se o algoritmo de Myers e Miller [172]. Tal algoritmo sacrifica um pouco o tempo de processamento, mas reduz sensivelmente o uso de memória. É importante salientar que este algoritmo inicialmente não permitia adotar pontuações distintas para abertura e extenção de *gap*. Thompson modificou tal algoritmo para permitir o uso do GOP e GEP [243].

C.8.3 Conclusão

É possível corrigir regiões mal alinhadas pela repetição do procedimento apenas no trecho problemático. Na repetição pode-se redefinir os parâmetros. Os autores mostram um caso onde conseguiram melhorar uma região mal alinhada com apenas mais uma execução e iguais parâmetros. Os autores citam dois outros trabalhos [225, 143] onde também alteram-se os parâmetros de acordo com as posições. Defende-se que a abordagem utilizada no Clustal W faz o mesmo só que utilizando apenas as sequências a alinhar. Desta forma, não cria dependências e torna-se mais flexível.

As modificações descritas neste trabalho aumentam a sensibilidade do MSA progressivo sem promover sacrifícios na velocidade e na eficiência. Mas os autores ainda citam áreas onde é possível melhorar. As matrizes de peso e as penalizações dos *gaps* podem se tornar mais precisas a medida que mais dados tornam-se disponível e que são criadas medidas para tipos específicos de sequência. Pode-se melhorar também através do uso de métodos de otimização iterativos ou estocásticos. Outro ponto de grande relevância para estudo é a função objetivo.

C.9 SAGA: MSA by genetic algorithm

Este trabalho [182] apresenta a primeira tentiva de aplicar algoritmos genéticos a alinhamento de sequências.

No contexto de alinhamento múltiplo de sequências, a abordagem progressiva de Feng

e Doolittle [74] ou variações [29, 239, 245] é a que mais se adota nas soluções existentes. Tal abordagem apresenta velocidade e simplicidade aliadas a uma boa sensibilidade, mas enfrenta problemas com mínimo local. Uma das principais alternativas à abordagem progressiva é o uso de funções objetivo, que quantificam a qualidade de um dado alinhamento. Os métodos que fazem uso de tais funções buscam o alinhamento de maior qualidade. Uma outra alternativa são modelos ocultos de Markov (HMM, do inglês *Hidden Markov Models*) [143]. Em ambos os casos enfrenta-se problemas com o crescimento no número de sequências a alinhar.

Visando solucionar este problema com o uso de funções objetivo, pode-se utilizar o pacote MSA ou métodos estocásticos. O pacote MSA [101, 151] tenta reduzir o espaço de solução para uma área relativamente pequena onde parece estar a solução. Ele encontra o alinhamento ótimo global ou um muito próximo dele começando com um conjunto de sequências sem qualquer alinhamento. Mesmo com esta redução no espaço de solução, o método continua com grandes limitações quanto ao número de sequências. Os métodos estocásticos podem ser implementados de diversas formas. Uma destas formas é o *simulated annealing* [3], que é bastante utilizado [115, 124, 139], mas parece apenas funcionar bem no refinamento de alinhamentos pré-construídos. Outra forma é o *Gibbs Sampling* [147], que mostrou bons resultados na busca por melhor bloco de alinhamento múltiplo local sem *gaps*. E outra forma ainda são os algoritmos genéticos (GA, do inglês *genetic algorithm*) [117]. Até então havia um único trabalho conhecido na área que fazia uso de GA, mas este utilizava um esquema híbrido de programação dinâmica/GA [124].

No restante desta seção é descrita a estratégia usada pelo método. Tal estratégia deu origem a um sistema chamado SAGA, cujos fontes é possível obter com os autores.

C.9.1 Método

O método basicamente usa uma medida para qualidade de alinhamento múltiplo, que é a chamada função objetivo (OF, do inglês *objective function*), e algoritmo genético para fazer a otimização. A OF utilizada é uma soma dos pares com peso e penalidades afim para *gaps*. A cada par de sequências é associado um peso que indica o grau de similaridade em relação às outras sequências e é uma tentativa de minimizar informação redundante. Observe que a OF pode ser variada de diversas formas: pode-se usar diferentes conjuntos de pesos, diferente conjuntos de custos para substituições (matrizes PAM [56] ou tabelas BLOSUM [106]) ou ainda diferentes esquemas para pontuação de *gaps*. Segue o custo total de um alinhamento múltiplo (A).

$$Custo(A) = \sum_{i=2}^{n} \sum_{j=1}^{i-1} W_{i,j} * Custo(A_i, A_j)$$

onde $Custo(A_i, A_j)$ indica a pontuação do alinhamento entre as sequências A_i e A_j e $W_{i,j}$ é o peso das sequências. Note que $Custo(A_i, A_j)$ inclui penalizações para abertura e extensão de *gaps*.

A estratégia utilizada deriva diretamente de algoritmo genético descrito por Goldberg [84]. Ou seja, há uma população inicial de soluções que evolui por meio de seleção natural. Em nosso caso, cada indivíduo na população é um alinhamento. Inicialmente, é gerada aleatoriamente uma geração zero (G_0) com 100 indivíduos (o tamanho da população é mantido constante). Para ir para a próxima geração, filhos são gerados de pais selecionados por uma espécie de seleção natural, com base no grau de afinidade dos pais mensurado pela OF. Uma vez selecionados os pais, para gerar o filho ainda é necessário selecionar um dos operadores. Para a nova geração é mantida uma porção de indivíduos da população corrente. Estes passos são iterativamente repetidos até que se chega a uma situação de estabilidade na população. No Algoritmo 11 é apresentado um pseudo-código com os passos.

Algoritmo 11: Versão simplificada do algoritmo utilizado pelo SAGA.

```
1  Cria G_0
2  while população não estabilizar do
3      Seleciona indivíduos para substituição
4      Avalia a prole esperada
5      while população G_{n+1} incompleta do
6          Seleciona os pais dentro da população corrente
7          Seleciona o operador
8          Gera um novo filho
9          Mantém ou descarta o novo filho em G_{n+1}
10     n = n + 1
```

Um operador é um pequeno programa que modifica um alinhamento. No SAGA foram definidos 22 operadores, que possuem probabilidades específicas de serem utilizados. Os operadores podem pertencer a uma de duas classes: cruzamento e mutação. Os operadores de cruzamento têm por característica combinar o conteúdo de dois indivíduos para gerar um terceiro. Já os operadores de mutação simplesmente gera um indivíduo a partir de modificações em um outro indivíduo.

Durante a execução do algoritmo é utilizado um escalonador dinâmico para a seleção do operador [55]. O escalonador atribui inicialmente probabilidades iguais de seleção para cada operador e altera tais probabilidades de acordo com a evolução da população. A probabilidade de uma dada operação ser utilizada é incrementada em função de sua eficiência nas últimas gerações. Quanto a escolha dos locais para a realização das mutações, foi cri-

ado um mecanismo que privilegia a seleção de pontos onde há maior concentração de *gaps*.

Foram feitos diversos testes com o SAGA comparando seu desempenho com o MSA e com o Clustal W. O SAGA obteve resultados tão bons ou, em alguns casos, melhores que os outros.

C.9.2 Conclusão

O uso de algoritmos genéticos para alinhamento múltiplo de sequências parece ser uma boa alternativa como heurística para o problema, mas certamente ainda há muito a melhorar. Um grande problema encontra-se em seu tempo de resposta, que possivelmente poderá ser resolvido com a utilização de um método híbrido usando abordagem progressiva / algoritmo genético. Desta forma, tentando unir a velocidade da abordagem progressiva com a precisão do algoritmo genético. Outro trabalho futuro seria avaliar a possibilidade de tornar o SAGA uma ferramenta para refinamento de alinhamentos. Para tanto poderia utilizar como entrada uma primeira geração composta por resultados do Clustal W.

Os autores creditam os bons resultados do SAGA ao grande número de operadores e ao escalonador. Eles também destacam como grande característica do *software* a flexibilidade para substituir a OF e poder mudar sensivelmente os resultados, podendo assim rapidamente testar novas OFs.

C.10 Further improvement in methods of group-to-group MSA with generalized profile operations

Este trabalho [91] foi a primeira tentativa de sistematicamente melhorar a precisão do método MSA pela uso de alinhamento estrutural.

É conhecido que é possível estender o algoritmo que faz uso de programação dinâmica para o caso de alinhamento entre dois grupos de sequências. Porém sua grande desvantagem encontra-se no fato do tempo computacional crescer na proporção que o produto do número de sequências nos dois grupos ($M \times N$) cresce. Neste trabalho é apresentado um método que possui precisão semelhante nos resultados, mas complexidade de tempo menos dependente no tamanho dos grupos. É importante salientar que tal desempenho só é alcançado quando há um número grande de sequências sendo alinhadas, por exemplo: testes realizados mostraram que ele executa ~ 10 vezes mais rápido quando $M \times N \approx 200$ e mais que 100 vezes mais rápido quando $M \times N > 2500$.

Alinhamento de grupos de sequências tem mostrado sua importância no contexto de MSA. Um bom exemplo disto é o fato de geralmente ser melhor alinhar grupos distantes

de sequências homólogas que fazer alinhamentos individuais [16]. Um outro bom exemplo é o fato de ser um dos passos na maioria dos métodos iterativos para MSA [48].

Anteriormente Gotoh havia apresentado algoritmos para alinhamento entre dois grupos de sequências [90], mas o mais preciso destes tinha sua complexidade de tempo proporcional a $M \times N$. Com o rápido crescimento no número de sequências de aminoácidos e nucleotídeos disponíveis, este algoritmo tem se tornado de uso impraticável. Já naquele trabalho Gotoh havia alertado que é possível reduzir os requisitos de processamento pelo uso de vetores de perfis [98], mas a complexidade de tempo continuava $O(M \times N)$, pois não havia forma de avaliar o custo de abertura de *gap* em tempo menor que $O(M \times N)$.

Neste trabalho, foi apresentado um novo algoritmo que permite uma avaliação rápida e precisa dos custos de *gap*. O ponto chave aqui é separar a avaliação dos *gaps* estáticos (pré-existentes) dos *gaps* dinâmicos, que são inseridos no alinhamento. Os *gaps* estáticos passam a ser avaliados apenas uma vez antes do procedimento principal de alinhamento. Desta forma são eliminadas operações redundantes e o tempo de computação do algoritmo deixa de ser diretamente dependente do número de sequências e passa a depender da distribuição de *gaps*. Baseado nos resultados deste trabalho o método corrente de alinhamento escolhe o algoritmo mais apropriado a utilizar, dentre os quatro desenvolvidos no trabalho anterior (alp, aln, rrp ou rrn) [90].

C.10.1 Algoritmo

Seja A um grupo de sequências de comprimento I e composto de M sequências pré-alinhadas. Seja X um conjunto de símbolos. Cada elemento de A, $a_{m,i} \in X$ ($1 \leq m \leq M$, $1 \leq i \leq I$), é um caracter representando um aminoácido, um nucleotídeo ou um *null*, que indica uma remoção. Desta forma X é composto por cinco (para nucleotídeos) ou 21 (para aminoácidos) elementos. Nesta seção quando fala-se em remoção refere-se a uma única posição e quando fala-se em *gap* refere-se a uma sequência de remoções consecutivas em uma sequência. Cada remoção estática é representada por $\triangle$. A variável booleana (0 ou 1) $q^A_{m,i} \equiv (a_{m,i} = \triangle)$ indica se $a_{m,i}$ representa uma remoção ou um resíduo. A variável $Q^A_{m,i}$ representa o estado de *gap* da posição, ou seja, o número de *nulls* consecutivos incluindo e imediatamente anterior ao elemento $a_{m,i}$ na linha m. Representa-se por $f^A_{x,i}$ o número de ocorrências do elemento $x \in X$ na coluna i de A. O vetor f^A_i, de dimensão 5 ou 21, é chamado vetor de frequência. O perfil da posição i é representado pelo vetor p^A_i de mesma dimensão que f^A_i:

$$p^A_{x,i} \equiv \sum_{y \in X} d(x,y) * f^A_{y,i},$$

onde $d(x,y)$ denota a dissimilaridade entre x e y.

Seja B um outro grupo de sequências com mesmo comprimento I e composto de N sequências pré-alinhadas. Calcula-se a pontuação SP (do inglês, *Sum of Pairs*) entre A e B da seguinte forma.

$$SP(A,B) = \sum_{i=1}^{I}\sum_{m=1}^{M}\sum_{n=1}^{N}(d(a_{m,i},b_{n,i}) + v * g_{m,n,i}), \tag{C.1}$$

onde v é o custo de abertura de *gap* e

$$g_{m,n,i} = (1-q_{m,i}^{A})q_{n,i}^{A}(Q_{m,i-1}^{A} \geq Q_{n,i-1}^{B}) + q_{m,i}^{A}(1-q_{n,i}^{B})(Q_{m,i-1}^{A} \leq Q_{n,i-1}^{B}).$$

Observe que, com o uso dos vetores de perfil e frequência, pode-se reescrever o somatório do primeiro termo na Equação C.1 de diversas formas

$$\sum_{m=1}^{M}\sum_{n=1}^{N} d(a_{m,i},b_{n,i}) = \sum_{n=1}^{N} p_{b_{n,i},i}^{A} = \sum_{m=1}^{M} p_{a_{m,i},i}^{B}, \tag{C.2}$$

assim como

$$\sum_{m=1}^{M}\sum_{n=1}^{N} d(a_{m,i},b_{n,i}) = \sum_{x\in X} p_{x,i}^{A} * f_{x,i}^{B} = \sum_{x\in X} f_{x,i}^{A} * p_{x,i}^{B}. \tag{C.3}$$

Dependendo dos valores de M e N, pode-se escolher a expressão que possa ser avaliada de forma mais econômica. De forma análoga há também formas eficientes de calcular o segundo termo da Equação C.1 através de estruturas que identificam valores semelhantes para $Q_{m,i}$.

No trabalho anterior, Gotoh [90] propôs quatro algoritmos, (A,B,C,D), para obter um alinhamento entre dois grupos de sequências. Os algoritmos mais simples, (A) e (B), não são capazes de avaliar precisamente o custo de abertura de *gaps*, ao contrário dos algoritmos (C) e (D). O algoritmo (C) segue o procedimento de programação dinâmica padrão [174]. Já o algoritmo (D) adota o paradigma da lista de candidatos [163], que permite otimizações rigorosas em casos que o algoritmo (C) não consegue.

No trabalho é apresentado como adaptar os algoritmos (C) e (D) para fazer uso dos perfis. O autor também mostra que é possível aplicar a estratégia iterativa, proposta por Berger e Munson [36], para realizar o alinhamento de grupos de sequências.

C.10.2 Conclusão

A maior contribuição do trabalho foi reduzir o tempo para computação de alinhamento de grupos de sequências sem perda de precisão. O novo método chega a executar ~ 100 vezes mais rápido que o método anterior. Outra contribuição foi uma redução significante nos

requisitos de memória. É sugerido que a inclusão de informações baseadas em estrutura seja algo a melhorar o desempenho de alinhamento múltiplo de grupos.

C.11 ProbCons: Probabilistic consistency-based multiple sequence alignment

Este trabalho [60] introduz o conceito de consistência probabilística, assim como apresenta o alinhador ProbCons. Tal ferramenta realiza alinhamento múltiplo de sequências de proteína através de um algoritmo progressivo baseado em consistência probabilística. De acordo com avaliações dos autores através de *benchmarks* tais como BAliBASE [248], SABmark [261] e PREFAB [69], o ProbCons alcança resultados significativamente mais precisos quando comparado com diversas ferramentas largamente empregadas para alinhamento múltiplo, ao mesmo tempo que mantém uma velocidade praticável de execução. Código fonte e executáveis estão disponíveis em domínio público [61].

Modelos ocultos de Markov (HMM, do inglês *hidden Markov models*) é um dos métodos que podem ser empregados na solução do problema do alinhamento múltiplo de sequências. Nesta abordagem parâmetros do modelo obtêm uma interpretação probabilística intuitiva e podem ser treinados em dados reais usando métodos probabilísticos padrões, supervisionados ou não. O algoritmo de Viterbi [258], por exemplo, computa a maior probabilidade de alinhamento de duas sequências de acordo com um alinhamento HMM. No HMM padrão para alinhamento (com três-estados), o algoritmo de Viterbi pode ser visto como uma instanciação do algoritmo de Needleman e Wunsch [174] no qual os parâmetros de alinhamento são determinados por uma transformação do esquema de pontuação do modelo HMM [64].

Para alinhamento múltiplo de sequências, a estratégia heurística de alinhamento progressivo [74] é, de longe, a mais popular nas ferramentas disponíveis. Neste esquema, o alinhamento final é construído a partir de diversos passos de alinhamento de pares. Como em qualquer abordagem hierárquica, erros cometidos em estágios iniciais do processo são propagados até o final, assim como podem levar a geração de novos erros devido a sinais incorretos de conservação. Para amenizar este tipo de problema são adicionados passos de pós-processamento tais como refinamento iterativo [93].

Esquemas baseados em consistência possuem uma visão alternativa da solução. Eles adotam a prática da "prevenção é o melhor remédio". Observe que para qualquer alinhamento múltiplo, os alinhamentos induzidos dos pares são necessariamente consistentes, ou seja, dado um alinhamento múltiplo contendo três sequências x, y e z, se a posição x_i alinha com a posição z_k e a posição z_k alinha com y_j nos alinhamentos projetados $x-z$ e $z-y$, então x_i deve alinhar com y_j no alinhamento projetado $x-y$. Técnicas baseadas em

consistência aplicam este princípio de forma invertida para guiar os alinhamentos de pares durante os passos de um alinhamento progressivo. Ao ajustar a pontuação do pareamento dos resíduos $x_i \sim y_j$ de acordo com o suporte de uma posição z_k que alinha a ambos x_i e y_j nas respectivas comparações $x - z$ e $z - y$, funções objetivo baseadas em consistência incorporam informação de múltiplas sequências na pontuação dos alinhamentos de pares.

Consistência já foi utilizada para identificar pontos âncora [89], que reduziam o espaço de busca de um alinhamento múltiplo. Consistência já foi reformulada para ser tratada por multiplicação de matrizes booleanas, que deu origem ao *software* MALI [255]. Uma formulação alternativa deu origem ao *software* DiAlign [168]. Recentemente foi introduzida uma nova função objetivo baseada em consistência COFFEE [184]. Baseado nesta função, foi implementada a ferramenta T-COFFEE [183], que apresenta resultados superiores a métodos como Clustal W [245], DiAlign e PRRP [93] de acordo com comparações realizadas no BAliBASE. Sendo que o Clustal W é o *software* mais utilizado para alinhamento múltiplo.

C.11.1 O algoritmo

ProbCons é um algoritmo de alinhamento progressivo baseado em HMM para pares de sequências. Ele utiliza o algoritmo de *máxima precisão esperada* ao invés do algoritmo de Viterbi, que é tipicamente utilizado, além de fazer uso de *transformações de consistência probabilística* para incorporar informações acerca de conservação de sequências durante alinhamento de pares. ProbCons utiliza o modelo HMM apresentado na Figura C.2 para especificar a probabilidade de distribuição sobre todos os alinhamentos entre um par de sequências. Probabilidades de emissão correspondem aos valores da matriz de substituição BLOSUM62 [106]. Probabilidades de transição, que correspondem a penalidades para *gaps*, são treinadas com *maximização de expectativa* não supervisionada (EM, do inglês *expectation maximization*).

Ao manter as probabilidades de emissão fixas (BLOSUM62), o modelo HMM do ProbCons é completamente especificado por apenas três parâmetros: probabilidade de iniciar o alinhamento com a inserção de um *gap* π_{insert}, probabilidade de abrir um *gap* δ e a probabilidade de estender um *gap* ϵ. Para treinar o ProbCons via EM, foram aplicadas 20 iterações do algoritmo Baum-Welch [33] nas sequências não alinhadas do BAliBASE. Os parâmetros foram inicializados com valores aleatórios. Os parâmetros resultantes ($\pi_{insert} = 0,19598$, $\delta = 0,019931$ e $\epsilon = 0,79433$) são usados como padrão pelo programa.

O modelo HMM do ProbCons é extremamente simples se comparado com outros como o de Durbin e colegas [64], que com uma abordagem de construção de perfis-HMM é mais rico, porém possui grande número de parâmetros e consequentemente torna a tarefa de treinamento mais complexa.

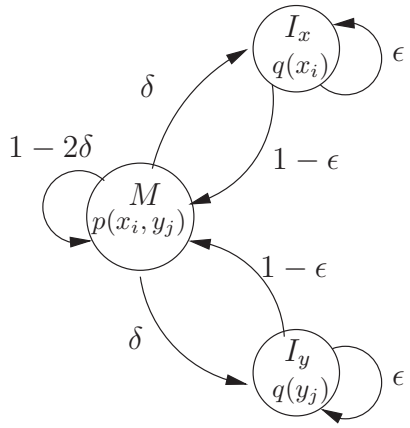


Figura C.2: Modelo oculto de Markov três-estados, utilizado pelo ProbCons, para o alinhamento entre duas sequências, x e y. O estado M emite duas letras, uma de cada sequência. Corresponde ao alinhamento delas. O estado I_x emite uma letra da sequência x e corresponde ao alinhamento da letra de x com um *gap*. De forma semelhante, I_y emite uma letra da sequência y. Os símbolos δ e ϵ denotam, respectivamente, probabilidades de abrir e estender um *gap*. A função p denota a probabilidade de emissão de M e tem seus valores calculados a partir da matriz de substituição. A função q denota a probabilidade de emissão para I_x e I_y.

Dadas m sequências $S = \{s^{(1)}, \ldots, s^{(m)}\}$, o algoritmo realiza o alinhamento da seguinte forma.

Passo 1: Cálculo das matrizes de probabilidade posterior

Para todo par de sequências $x, y \in S$ e todo $i \in \{1, \ldots, |x|\}$, $j \in \{1, \ldots, |y|\}$, calcule a matriz P_{xy}, onde $P_{xy}(i, j) = P(x_i \sim y_j \in a^*|x, y)$ é a probabilidade que letras x_i e y_j são pareadas em a^*, um alinhamento de x e y gerado pelo modelo.

Passo 2: Cálculo da precisão esperada

Defina a precisão esperada de um alinhamento a entre x e y como sendo o número esperado de pares de letras corretamente alinhados dividido pelo tamanho da sequência mais curta:

$$E_{a^*}(accuracy(a, a^*)|x, y) = \frac{1}{\min\{|x|, |y|\}} \sum_{x_i \sim y_j \in a} P(x_i \sim y_j \in a^*|x, y).$$

Para cada par de sequências $x, y \in S$, compute o alinhamento a que maximiza a precisão esperada através de programação dinâmica e defina o respectivo valor $E(x, y) = E_{a^*}(accuracy(a, a^*)|x, y)$.

Passo 3: Cálculo das transformações de consistência probabilística

Redefina as pontuações $P(x_i \sim y_j \in a^*|x,y)$ pela aplicação de transformações de consistência probabilística, que incorporam a similaridade de x e y a outras sequências de S no alinhamento $x-y$:

$$P'(x_i \sim y_j \in a^*|x,y) \leftarrow \frac{1}{|S|}\sum_{z\in S}\sum_{z_k} P(x_i \sim z_k \in a^*|x,z)P(z_k \sim y_j \in a^*|z,y).$$

Na forma de matriz, isso pode ser reescrito da seguinte forma:

$$P'_{xy} \leftarrow \frac{1}{|S|}\sum_{z\in S} P_{xz}P_{zy}.$$

Algo importante aqui é que a maioria dos valores nas matrizes P_{xz} e P_{zy} são próximos a zero. Assim a transformação pode ser computada eficientemente usando algoritmos para multiplicação de matrizes esparsas ao ignorar entradas menores que um limiar ω (por padrão $\omega = 0,01$). Este passo pode ser repetido tantas vezes quanto desejado. Com o uso de algoritmos para multiplicação de matrizes esparsas, este passo cai de uma complexidade $O(L^3)$ para $O(c^2L)$, onde L corresponde ao tamanho das sequências e c é o número médio de elementos diferentes de zero por linha (tipicamente $1 \leq c \leq 5$).

Foram feitos testes para validar a implementação do limiar, que gera matrizes esparsas e assim permite um melhor desempenho do sistema. Os resultados mostraram que as execuções com (algoritmo $O(c^2L)$) e sem uso (algoritmo $O(L^3)$) de limiar apresentam qualidade semelhantes. Tal qualidade é praticamente idêntida quando se realizam 2 passos de refinamento iterativo.

Passo 4: Geração da árvore guia

Contrua a árvore guia para S através de agrupamento hierárquico. Como uma medida de similaridade entre duas sequências x e y use $E(x,y)$, calculado no segundo passo. Defina a similaridade entre dois grupos por uma média ponderada das similaridades de pares entre sequências dos dois grupos.

A maioria dos alinhadores usam distância evolucionária estimada de alinhamentos de pares ou estatísticas de k-mer para construir uma árvore evolucionária aproximada via algoritmo "*Neighbor Joining*" [210] ou UPGMA [228]. No ProbCons não há uma tentativa de construção de uma árvore evolucionária correta, mas sim uma árvore com alta confiabilidade esperada de alinhamento através de um método guloso derivado do UPGMA.

Passo 5: Alinhamento progressivo

Alinhe os grupos de sequências iterativamente de acordo com a ordem especificada pela árvore guia. Alinhamentos são pontuados usando uma função de soma dos pares na qual resíduos alinhados são associados a pontuações transformadas $P'(x_i \sim y_j \in a^*|x, y)$ e penalidades para *gaps* são definidas como zero.

Passo 6: Refinamento iterativo

Particione o alinhamento múltiplo aleatoriamente em dois grupos de sequências e realinhe. Este passo também pode ser repetido tantas vezes quanto desejado.

C.11.2 Testes

A performance do ProbCons foi testada usando três diferentes pacotes de *benchmark* para alinhamento múltiplo. Foram eles BAliBASE 2.01 [246], PREFAB 3.0 [69] e SABmark 1.63 [261].

Os resultados do ProbCons foram comparados com os resultados de seis outros sistemas de alinhamento múltiplo: Clustal W 1.83 [245] (o método mais popular), DiAlign 2.2.1 [168] (alinhador local que usa homologia baseada em segmento), T-COFFEE 1.37 [183] (alinhador baseado em consistência que combina alinhamentos globais e locais), MAFFT 3.88 [136] (um conjunto de seis *scripts* com uma variedade de técnicas de refinamento iterativo), MUSCLE 3.3 [68] (alinhador recente que apresenta os melhores resultados no BAliBASE até então) e Align-m 1.0 [262] (utiliza um método baseado em consistência). Todos os sistemas foram avaliados de acordo com os respectivos valores padrões de parâmetros.

O modelo HMM do ProbCons foi treinado usando a base de dados do BAliBASE, sendo assim, o PREFAB e o SABmark foram utilizados, também, com o objetivo de prover uma validação externa. Os resultados mostraram que o ProbCons traz contribuições de grande relevância na qualidade dos alinhamentos. Mostra também que o ProbCons exige um tempo de execução elevado. O tempo requisitado é maior que a exigida pela maioria dos sistemas testados, entretando tal demanda não chega a tornar proibitiva sua utilização.

C.11.3 Conclusão

ProbCons utiliza um modelo simples de similaridade de sequências (HMM de três estados) e não faz tentativas de incorporar conhecimento biológico tal como pontuação de *gap* para posição específica, construção de árvores evolucionárias rigorosas e outras características usadas por alinhadores tais como Clustal W. ProbCons não usa informações além das

extraídas das matrizes de substituição BLOSUM. Realizando a substituíção desta por valores equivalentes para nucleotídeos é possível gerar alinhamentos múltiplos para DNA.

Como todo treinamento para o programa foi feito automaticamente em sequências não alinhadas usando EM sem intervenções humanas, é possível retreinar ProbCons para tipos específicos de sequência. E assim obter parâmetros mais apropriados para tarefas particulares de alinhamento.

Teste realizados com diferentes variações do ProbCons indicam que as características determinantes para sua precisão são o uso de máxima precisão esperada como função objetivo e a aplicação de transformações de consistência probabilística.

A abordagem de consistência probabilística pode ser interessante para busca comparativa de genes e predição de RNA ou estruturas de proteínas.

C.12 M-COFFEE: combining MSA methods with T-COFFEE

Este trabalho [260] apresenta um meta-método para construção de alinhamento múltiplo de sequências pela combinação da saída de diversos outros algoritmos em um único alinhamento. O M-COFFEE é capaz realiza combinação dos alinhamentos rapidamente, gastando menos tempo nesta tarefa que a execução de um algoritmo de alinhamento múltiplo. Sua grande vantagem em relação aos outros métodos está em sua precisão. M-COFFEE é parte do pacote do T-COFFEE, que foi escrito em `Perl` e `C` e executa em plataformas baseadas em `UNIX`. Ele é um *freeware* de código aberto sob a lincença `GNU` e disponível em `http://www.tcoffee.org`.

No decorrer da evolução dos algoritmos para MSA, um importante passo foi o desenvolvimento de métodos baseados em consistência, onde o propósito é gerar um alinhamento consistente com um conjunto de alinhamentos de pares. O uso de consistência foi primeiro descrito por Gotoh [89] e Kececioglu [138] e re-formalizado por Vingron e Argos [256] como um procedimento de multiplicação de matrizes de pontos. Posteriormente, consistência foi re-descoberta por Morgenstern e colegas [168] culminando no método DiAlign. Então, em 2000, Notredame e colegas [183] definiram o T-COFFEE, um método que combina as idéias do DiAlign com uma estratégia de alinhamento progressivo. Tal método resultou em uma significante melhoria na precisão dos resultados em MSA. Desde então, função objetivo baseada em consistência tem sido utilizada em vários dos novos pacotes, tais como: POA [149], MAFFT 5 [135], MUSCLE 6 [68], ProbCons [60] e PCMA [199].

A maneira que tem sido adotada para a avaliação dos diversos métodos para MSA é através de avaliações empíricas em alinhamentos de sequência baseados em estrutura. Há diversas coleções alternativas de *benchmark* disponíveis, dentre elas: BAliBASE [249],

PREFAB [69] e HOMSTRAD [165]. É razoável considerar que métodos com melhores performances médias são melhores. Entretanto é importante salientar que tal escolha não é garantia de sucesso, pois o método escolhido pode não ser o mais preciso em algum banco de dados específico. Dentre os principais problemas desta abordagem está o fato de gerar alinhamentos estruturalmente ao invés de evolucionariamente corretos e o fato de que a montagem de MSAs baseados em estrutura é uma tarefa difícil.

Em meta-métodos, uma etapa importante e, no caso de MSA, complicada é o procedimento de combinação das entrada. No M-COFFEE este problema é resolvido simple e elegantemente através de funções objetivas baseadas em consistência que geram consenso entre os elementos da entrada. Dada uma coleção de alinhamentos alternativos, tais funções definem um alinhamento ótimo como sendo aquele de maior nível de consistência com a coleção, em outras palavras, consenso. Esta abordagem foi inicialmente descrita por Bucka-Lassen e colegas [44] e é núcleo do algoritmo do T-COFFEE, onde é definido o conceito de bibliotecas. Ele não alinha sequências explicitamente, o que faz é compilar bibliotecas baseando-se em alinhamentos produzidos externamente. Durante o processo, as bibliotecas são combinada em um MSA final. Apesar de usar o Clustal W e Lalign, o T-COFFEE permite que qualquer pacote de alinhamento, múltiplo ou de par, seja utilizado. No M-COFFEE, o que é feito é utilizar exatamente esta possibilidade e explorar as alternativas. São analisados 15 algoritmos e é sugerido um subconjunto destes que apresentam resultados melhores que os apresentados até então.

C.12.1 O Método

No estudo, foram selecionados 15 programas para MSA largamente utilizados de um total de 8 diferentes laboratórios. O objetivo foi explorar uma grande variedade de algoritmos para alinhamento de sequências de proteína.

Clustal W 1.83 [50, 245]: é o programa mais utilizado para MSA. Utiliza uma abordagem progressiva.

T-COFFEE 2.03 [183]: usa função objetivo baseada em consistência [184] otimizada através de um alinhamento progressivo.

ProbCons 1.09 [60]: é também um método baseado em consistência, além de modelos ocultos de Markov. Até então era considerado o método mais preciso, conforme atestado pelo HOMSTRAD [165].

PCMA 2.0 [199]: utiliza função objetivo baseada em consistência para alinhar sequências de menor homologia e um algoritmo semelhante ao do Clustal W para sequências de maior homologia.

MUSCLE [68] :

3.52: usa um algoritmo de alinhamento progressivo.

6.0: usa uma função objetivo tal como o ProbCons para refinar o alinhamento do MUSCLE 3.52.

DiAlign2 2.2.1 [166]: através de melhorias no algoritmo do DiAlign [168] realiza alinhamentos múltiplos locais.

DiAlign-T 0.1.3 [236]: nova versão do DiAlign, que insere a função objetivo do DiAlign em um algoritmo de alinhamento progressivo.

MAFFT 5.531 [135] :

FFT-NS1: alinhamento progressivo que usa transformada rápida de Fourier para calcular a árvore guia.

FFT-NS2: Semelhante ao anterior, mas com a diferença de re-calcular a árvore guia depois do primeiro alinhamento e re-alinhar.

FFT-NSI: Semelhante ao FFT-NS1, mas inclui um passo iterativo de refinamento.

F-INSI: incorpora informação sobre alinhamento local de pares.

G-INSI: incorpora informação sobre alinhamento global de pares.

POA 2.0 [96]: usa grafos de ordem parcial para construir os MSAs. É possível optar pelo tipo de algoritmo, dentre eles:

Local: uma versão que realiza alinhamento local.

Global: uma versão que realiza alinhamento global.

A fim de exibir visualmente o nível de similaridade entre os vários métodos, assim como de calcular pesos para os métodos, foi calculada uma árvore. O primeiro passo foi computar uma matriz de distâncias onde cada entrada indica uma medida de diferença média entre dois métodos no conjunto de dados completo do HOMSTRAD. Então aplicou-se o algoritmo UPGMA na matriz.

O M-COFFEE foi implementado a partir do T-COFFEE. Bibliotecas são geradas dos alinhamentos criados pelos diferentes pacotes. Todas as bibliotecas são então dadas como entrada para o T-COFFEE. O peso padrão usado na biblioteca é o percentual de identidade das sequências pais. No M-COFFEE este esquema de peso muda um pouco. O peso original é multiplicado pelo peso do método (da biblioteca em questão). Foram

definidos quatro esquemas, sendo que dois deles utilizam a árvore de métodos descrita acima e os outros dois não.

A primeira tarefa na construção do M-COFFEE foi determinar como os 15 métodos utilizados deveriam ser considerados no consenso de alinhamento. A primeira tentativa foi usar um procedimento guloso a fim de determinar um subconjunto de métodos. Os métodos foram classificados de acordo com sua precisão no HOMSTRAD. Em seguida foram realizados testes, inicialmente utilizando apenas o melhor método (ProbCons), em seguida usando os dois melhores (ProbCons + MUSCLE 6.0), e assim sucessivamente. Observou-se que, a partir do teste com os três melhores métodos, os resultados são mais precisos, o que estabelece a eficiência da combinação. Observou-se também que com a inserção de métodos muito similares há uma degradação na precisão. Observe que desta forma haverá a repetição dos mesmos erros em diversos dos alinhamentos, o que resulta na inserção do erro no consenso. O maior pico na precisão ocorreu quando foi realizado um teste com os seis melhores algoritmos (ProbCons, MUSCLE 6.0, T-COFFEE, MUSCLE 3.52, F-INSI e PCMA).

Em uma etapa posterior, para evitar problemas com algoritmos similares, decidiu-se utilizar apenas um dos algoritmos (o melhor) de cada laboratório. Os oito algoritmos selecionados foram: POA-global, DiAlign-T, Clustal W, PCMA, F-INSI, T-COFFEE, MUSCLE 6.0 e ProbCons. Comparando o resultado do M-COFFEE com cada um dos 8 métodos isoladamente, o M-COFFEE sempre ganha. O M-COFFEE é mais preciso inclusive que o ProbCons, mesmo sem incluir os resultados deste. Esta versão do algoritmo que combina oito métodos foi então selecionada como padrão do M-COFFEE, mas note que o conjunto de alinhadores individuais pode ser facilmente substituido.

C.12.2 Conclusão

Os resultados dos testes mostraram que o M-COFFEE ganha em quase o dobro das vezes do melhor método individual no conjunto completo de dados do HOMSTRAD, PREFAB e BAliBASE. Sempre comparando o M-COFFEE com o melhor método individual no conjunto em questão. Em termos de tempo de CPU, o M-COFFEE é muito similar ao T-COFFEE padrão, sendo inclusive um pouco menor por não requerer a geração de uma biblioteca de pares. É importante salientar que não foi levado em consideração o tempo de computação dos alinhamentos pelos métodos individuais.

Apesar de mostrar ser significativamente mais preciso, o M-COFFEE requer grande tempo para execução se levado em consideração todo o tempo requerido, desde as computações dos alinhamentos por métodos individuais até a combinação dos alinhamentos num alinhamento consenso.

C.13 DbClustal: rapid and reliable global multiple alignments of protein sequences detected by database searches

Este trabalho [251] apresenta o DbClustal, pacote que produz alinhamentos globais de alta qualidade para sequências com boas pontuações em resultados de buscas com BLAST [12]. Seu procedimento é automático e o tempo de execução permite seu uso para análise de genomas em larga escala. O DbClustal combina as vantagens dos algoritmos de alinhamento global e local na tradicional abordagem progressiva.

Uma importante aplicação de alinhamentos múltiplos é o alinhamento de conjuntos de sequências detectadas por busca de homologia em bases de dados. Entretanto, com o crescimento exponencial das bases, os requisitos de tempo têm se tornado o principal fator limitante dos métodos automáticos para este tipo de tarefa. Em particular, a análise comparativa de genomas completos necessita de alto *throughput* no processamento automático das milhares de buscas por homologia.

Existem diversos pacotes para a construção de alinhamento múltiplo local das sequências com boas pontuações identificadas nas buscas em bases de dados [141, 177, 211, 240]. Entretanto só há alguns pacotes (semi-)automáticos para alinhamento múltiplo global de sequências detectadas por buscas em bases de dados, que requerem uma certa intervenção manual por um especialista. Gracy e Argos [95] usam um método progressivo com correção manual para classificação automática de sequências de proteína. Jiang e Jacob no EbEST [126] construíram alinhamentos de ESTs (do inglês *Expressed Sequence Tags*). Baxevanis e Landsman [34] construíram alinhamentos manuais de *motifs* de dobra de histonas e então usaram Clustal W [245] para alinhar o restante das sequências. Srinivasarao e colegas [232] também usaram Clustal W para gerar alinhamentos múltiplos no banco de dados PIR-ALN, seguido por correção manual.

Trabalhos recentes [41, 44] têm combinado alinhamentos de diversas fontes, incluindo tanto algoritmos locais como globais, com o objetivo de tratar padrões complexos induzidos por proteínas modulares altamente variáveis. Embora produzam alinhamentos precisos para entradas muito complexas, os requisitos computacionais são muito elevados.

C.13.1 O método

O DbClustal foi basicamente construído a partir de modificações no Clustal W 1.81 [245], que passou por um processo de modularização para permitir evoluções mais facilmente. Além disso, agora passa a receber na entrada informações produzidas pelo Ballast (pacote de pós-processamento do BLAST) [201], que gera âncoras entre pares de sequências. É importante salientar que, da forma que foi implementado, é possível substituir o Ballast

por outro pacote desde que gere uma saída no formato do Ballast. Pesos são associados às âncoras e definidos de forma a encorajar a manutenção de *motifs* conservados no alinhamento. Clustal W e DbClustal foram implemantados em `ANSI C` e têm seus fontes disponíveis em `ftp://ftp-igbmc.u-strasbg.fr/pub/DbClustal/`. Ballast também foi impletado em `ANSI C` e tem seus fontes disponíveis em `ftp://ftp-igbmc.u-strasbg.fr/pub/Ballast/`.

Ballast funciona da seguinte forma. Empilha pares de segmentos sem *gaps* extraídos de uma busca do BlastP para construir um perfil de conservação ao longo da sequência de consulta. São descartados do processamento *hits* com *e-value* maiores que $0,1$. Então usa-se este perfil para predizer segmentos de máximo local (LMS, do inglês *Local Maximum Segments*) na sequência de consulta. Segmentos de alinhamento (BLAST) que sobrepõem LMSs são chamados pares de segmentos de máximo local (LMSPs, do inglês *Local Maximum Segment Pairs*). A cada LMSP é atribuido uma pontuação dependendo de parâmetros do perfil e da similaridade entre as sequências do par. Para maiores detalhes, verificar o trabalho de Plewniak e colegas [201]. LMSPs assim definem âncoras entre a sequência de consulta e as sequências da base de dados com um peso igual à pontuação do LMSP dividida pela maior pontuação possível na busca corrente.

Clustal W utiliza o algoritmo global de programação dinâmica proposto por Needleman e Wunsch [174] para construir um alinhamento múltiplo. Inicialmente são alinhadas as duas sequências mais relacionadas e então progressivamente alinha-se grupos cada vez maiores de sequências até que todas estejam alinhadas. O algoritmo para duas sequências X e Y de tamanhos N e M, respectivamente, requer uma pontuação para alinhar qualquer dois resíduos X_i e Y_j, assim como penalidades para abertura e extensão de *gaps* no alinhamento. O algoritmo recursivo pode ser sumarizado como:

$$H_{i,j} = max \begin{cases} H_{i-1,j-1} + S_{i,j} \\ max_{1 \leq k < i}\{H_{i-k,j} - (g + hk)\} \\ max_{1 \leq l < j}\{H_{i,j-l} - (g + hl)\} \end{cases},$$

onde $S_{i,j}$ é a pontuação para o alinhamento dos resíduos X_i e Y_j, g e h são as penalidades para abertura e extensão de *gap*, respectivamente, e k e l indicam o tamanho do *gap* aberto para cada sequência. No Clustal W a pontuação $S_{i,j}$ é simplesmente igual a pontuação $C_{i,j}$ da matriz de comparação de resíduos. O alinhamento de dois grupos de sequências é uma simples extensão do algoritmo, onde a pontuação para alinhar dois resíduos é substituída pela pontuação para alinhar duas colunas nos respectivos grupos.

O sistema de pontuação no DbClustal foi modificado para incorporar informações sobre conservação local, oriundas do Ballast. Durante o alinhamento progressivo, uma matriz $M \times N$ posição-específica de âncoras é calculada para cada par de sequências (grupos) a serem alinhadas. Para a coluna i no primeiro grupo e coluna j no segundo grupo, a

pontuação $ANCHOR_{i,j}$ é dada por:

$$ANCHOR_{i,j} = max(0, max_{1 \leq k \leq L}(W_k)),$$

para todas as âncoras contendo qualquer par de resíduos nas colunas i, j, onde W_k é o peso definido pelo Ballast e L é o número de âncoras. A pontuação para alinhamento dos resíduos A_i e B_j passa a ser definido por

$$C_{i,j} + ANCHOR_{i,j},$$

onde $C_{i,j}$ é a pontuação para o alinhamento dos resíduos A_i e B_j segundo a matriz de comparação. Quanto às penalizações para *gap*, foi mantido tal como no Clustal W.

C.13.2 Conclusão

Com seu método para manipulação de âncoras, o DbClustal conseguiu avançar na solução de problemas tipicamente encontrados em pacotes para alinhamento múltiplo, tais como alinhamento de sequências divergentes e sequências com grandes inserções e extensões. DbClustal apresenta bons resultados mesmo com sequências de baixa homologia (*e-value* ≥ 0.1) e pelos testes também fica evidenciado o quão rápido ele é, permitindo, assim, seu emprego em cenários que exijam algoritmos automáticos e de alto *throughput* para alinhamento múltiplo, tal como em análise e anotação automática de genomas.

Neste trabalho também pode-se destacar a modularização do Clustal W, que permite a implementação ou modificação de funcionalidades de uma forma mais simples.

C.14 Multiple DNA and protein sequence alignment based on segment-to-segment comparison

Este trabalho [168] apresenta o primeiro método para alinhamento que não requer qualquer definição de penalidade para *gap*.

Em 1970, Needleman e Wunsch [174] publicaram um algoritmo para alinhamento de duas sequências. Tal algoritmo alinha sequências por maximizar uma pontuação que é atribuída de acordo com *matches* e adição de *gaps*. Desde então este algoritmo foi estendido e a grande maioria dos algoritmos hoje em dia são baseados nesta idéia. Tais algoritmos apresentam bons resultados se as sequências são altamente relacionadas, mas apresenta grande dificuldade em detectar curtas regiões similares em sequências longas de baixa similaridade. Há problema também no fato de seus resultados serem altamente sensíveis ao conjunto de parâmetros definidos pelo usuário, especialmente a penalidade para *gap*. Outro problema é que, embora facilmente extensível a mais que duas sequências,

a complexidade do algoritmos cresce exponencialmente com o número de sequências [170, 174].

Neste trabalho é proposto um novo conceito de alinhamento de sequências onde segmentos de tamanhos variados são alinhados sem uma definição explícita de *gap*.

C.14.1 Algoritmo básico para alinhamento de duas sequências

É requerido que os segmentos a alinhar possuam o mesmo tamanho e não é permitido qualquer *gap* nos segmentos. Os alinhamentos são também chamados de diagonais, uma vez que o par de sequências é uma diagonal na matriz de pontos. O método é chamado de DiAlign.

Um alinhamento é definido como uma relação consistente de equivalência no conjunto de todas as posições de todas as sequências envolvidas. Consistente significa que a ordem geral das posições em cada sequência é respeitada. Se apenas duas sequências são alinhadas, há consistência se para quaisquer duas diagonais D_1 e D_2 pertencentes ao alinhamento ou $D_1 \ll D_2$ ou $D_2 \ll D_1$, onde "$D_1 \ll D_2$" significa que as posições alinhadas em D_1 precedem aquelas alinhadas em D_2.

Define-se uma medida para avaliar a relevância de uma dada diagonal. Seja D uma diagonal, l o tamanho de D e m o número de *matches* em D. Denota-se por $P(l, m)$ a probabilidade de qualquer diagonal de tamanho l conter, pelo menos, m *matches*. Segue que:

$$P(l,m) = \sum_{i=m}^{l} \binom{l}{i} p^i (1-p)^{l-i}, \tag{C.4}$$

onde p é a probabilidade de um ponto da matriz de pontos representar um *match*. A título de simplificação, pode-se assumir uma distribuição uniforme ($p = 0.25$ para ácido nucléico e $p = 0.05$ para proteína).

Define-se também peso de uma diagonal D por:

$$w(D) = \begin{cases} E(l,m), & se\ E(l,m) > T \\ 0 & se\ E(l,m) \leq T, \end{cases} \tag{C.5}$$

onde T é um limiar definido pelo usuário e

$$E(l,m) = -\ln P(l,m). \tag{C.6}$$

Se as diagonais D_1, D_2, ..., D_c constituem um conjunto consistente de diagonais, a pontuação deste conjunto é definida como $\sum_{i=1}^{c} w(D_i)$. Um conjunto consistente de diagonais com pontuação máxima é chamado alinhamento máximo.

O método utiliza programação dinâmica e recursão para calcular um alinhamento máximo. Seja $X = (X_1, \ldots, X_{L_1})$ e $Y = (Y_1, \ldots, Y_{L_2})$ duas sequências. O algoritmo inicialmente determina, para todo par de posições (i, j) com $1 \leq i \leq L_1$ e $1 \leq j \leq L_2$, todos os inteiros $k \geq 0$, com $k \leq min(i-1, j-1)$, para os quais a diagonal $(X_{i-k}, Y_{j-k}; X_i, Y_j)$ de $(i-k, j-k)$ até (i, j) tem peso positivo. Depois, para cada um dos pares (i, j), define-se o valor $pontuacao(i, j)$, que é a pontuação de um alinhamento máximo dos prefixos $(X_1, \ldots, X_i)$ e $(Y_1, \ldots, Y_j)$. Sejam $D_1, D_2, \ldots, D_c$ as diagonais do alinhamento máximo dos prefixos $(X_1, \ldots, X_i)$ e $(Y_1, \ldots, Y_j)$ e $D_1 \ll D_2 \ll \ldots \ll D_c$. Define-se $prec(i, j) = D_c$.

Para cada diagonal $D = (X_{i-k}, Y_{j-k}; X_i, Y_j)$ com peso positivo, denota-se por $\sigma(D)$ a soma máxima dos pesos acumulados e incluindo D. Denota-se por $\pi(D)$ a diagonal que precede D. Note que as relações apresentadas nas Equações C.7 e C.8 são válidas.

$$\sigma(D) = pontuacao(i-k-1, j-k-1) + w(D) \tag{C.7}$$

$$\pi(D) = prec(i-k-1, j-k-1) \tag{C.8}$$

Assim, pode-se computar recursivamente o valor da pontuação por

$$pontuacao(i, j) = max\{pontuacao(i, j-1), pontuacao(i-1, j), \sigma(D_{i,j})\}$$

onde $D_{i,j}$ é qualquer diagonal que termina no ponto (i, j) e satisfaz

$$\sigma(D_{i,j}) = max\{\sigma(D) : \ D \ termina \ no \ ponto \ (i, j)\},$$

enquanto que $prec$ pode ser definido por

$$prec(i, j) = \begin{cases} prec(i, j-1) & se \ pontuacao(i, j) = pontuacao(i, j-1), \\ prec(i-1, j) & se \ pontuacao(i, j-1) < pontuacao(i, j) = pontuacao(i-1, j), \\ D_{i,j} & se \ pontuacao(i, j-1) < pontuacao(i, j) = \sigma(D_{i,j}) \\ & e \ pontuacao(i-1, j) < pontuacao(i, j) = \sigma(D_{i,j}). \end{cases}$$

Finalmente, define-se $D_1 = prec(L_1, L_2)$ e $D_{i+1} = \pi(D_i)$. Agora utilizando-se programação dinâmica, calcula-se todos estes valores, que então permite-nos encontrar o alinhamento máximo.

C.14.2 Alinhamento múltiplo

Uma possível abordagem para estender o algoritmo básico de duas sequências para n sequências, onde $n \geq 3$, seria utilizar diagonais N-dimensionais, mas tal abordagem

levaria a um grande crescimento na complexidade computacional do algoritmo. Decidiu-se então adotar uma abordagem onde constrói-se alinhamentos múltiplos a partir de todas as diagonais em duas dimensões de todos os $\frac{n}{2}(n-1)$ pares de sequência.

Uma vez calculadas as diagonais de todos os pares de sequências, construímos um conjunto com todas as diagonais. Denota-se tal conjunto por $\mathcal{D}$. Para reduzir o número total de diagonais em $\mathcal{D}$ e, assim, ganhar eficiência, decidiu-se por apenas adicionar a $\mathcal{D}$ aquelas diagonais que pertençam ao alinhamento máximo de um dos pares de sequências.

O passo seguinte é ordenar $\mathcal{D}$ de acordo com o peso das diagonais. Valendo-se de uma estratégia gulosa adiciona-se uma a uma as diagonais começando pela de maior peso. Antes de adicionar uma nova diagonal ao alinhamento múltiplo é necessário verificar se a consistência do alinhamento é mantida. Caso contrário ela é descartada.

C.14.3 Conclusão

O método é capaz de encontrar regiões limitadas de similaridade, ou seja, encontrar alinhamentos locais assim como definido por Smith e Waterman [226], mas não apenas o melhor alinhamento local. Ele é capaz de encontrar as diversas regiões similares cujo peso ultrapasse o limiar.

Apesar de construir alinhamentos múltiplos usando alinhamentos iterativos de pares assim como outros autores [13, 74, 90], no método DiAlign a ordem de execução dos alinhamentos dos pares não são cruciais para os resultados.

Uma grande vantagem do método é a pequena dependência nos parâmetros definidos pelos usuários. Tal vantagem origina-se em não tratar *gaps* explicitamente e assim não requer a definição de penalidades para *gaps*.

Em testes apresentados no trabalho fica claro o potencial do método. O método apresenta resultados próximos dos melhores métodos com a vantagem de ser independente quanto a definição de parâmetros.

C.15 Multiple alignment of biological sequences with gap flexibility

Este trabalho [161] apresenta uma abordagem heurística para obter alinhamento múltiplo de sequências biológicas. Faz uso de grafos direcionados acíclicos (DAGs, do inglês, *directed acyclic graph*) para representar alinhamentos. Tais grafos conseguem codificar diversos alinhamentos ótimos com uma estrutura compacta. A entrada para o algoritmo de alinhamento múltiplo, nesta abordagem, é um único DAG representando todas as sequências e cabe a ele gerar o alinhamento a partir delas, essencialmente associando colunas aos

caracteres e, consequentemente, posicionando os *gaps*. Os autores sugerem a aplicação desta abordagem para montagem de fragmentos de DNA.

Na maioria das vezes, alinhamentos múltiplos são construídos a partir de alinhamentos de pares. Um problema conhecido, porém comumente ignorado pelas abordagens para MSA é o fato de que não existe, necessariamente, um único alinhamento ótimo de pares. Podem existir diversos. Outro ponto importante é que alinhamentos próximos do ótimo também podem ser interessantes, uma vez que no processo de sequenciamento ou posterior manipulação das sequências podem existir erros, tais como remoção ou inserção de bases nucléicas (ou aminoácidos dependendo do tipo de sequência), ou ainda a substituição de uma destas bases (aminoácidos) por outra devido a um erro na leitura do fragmento.

Na abordagem apresentada neste trabalho é feita uma tentativa de manter um conjunto de bons alinhamentos entre duas sequências e deixar a decisão do alinhamento a utilizar (dentre os bons) para passos posteriores, onde se pode tomar uma decisão mais acertada. Os diversos alinhamentos entre duas sequências são armazenados em um DAG, chamado pelos autores de TLG (do inglês, *trace layout graph*), onde nós representam os caracteres das sequências e as arestas ligam caracteres consecutivos. A cada aresta é acrescido um rótulo que indica as sequências a que está relacionada. Como exemplo de sua estrutura e poder de síntese, tome como exemplo o TLG apresentado na Figura C.3. Ele representa os diversos alinhamentos entre as sequências `TTTAGC` e `TTTAAAC`, enumerados nas Tabelas C.5, C.6 e C.7. Tais sequências no TLG serão referenciadas como 1 e 2, respectivamente. Observe que os TLGs não fazem qualquer menção a *gaps* ou *mismatches*.

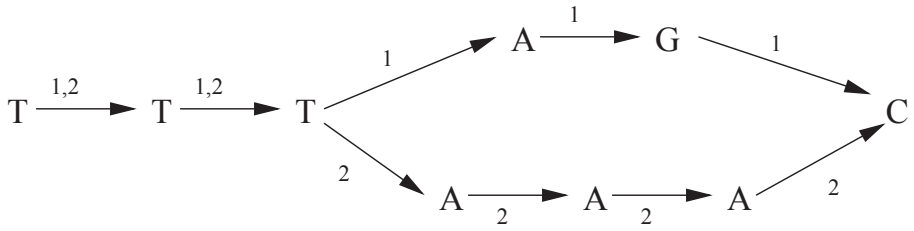


Figura C.3: TLG representando alinhamentos entre as sequências `TTTAGC` e `TTTAAAC`.

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

Tabela C.5: Primeira possibilidade de alinhamento entre as sequências `TTTAGC` e `TTTAAAC`.

C.15.1 TLGs: criação, junção e reparos

A abordagem utilizada por este trabalho deriva da noção de *trace* introduzida por Sankoff e Kruskal [212], posteriormente generalizada para múltiplas sequências por Kececioglu [137].

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

Tabela C.6: Segunda possibilidade de alinhamento entre as sequências `TTTAGC` e `TTTAAAC`.

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

Tabela C.7: Terceira possibilidade de alinhamento entre as sequências `TTTAGC` e `TTTAAAC`.

Utilizando-se um algoritmo de programação dinâmica padrão para alinhamento global e, tomando-se os alinhamentos ótimos *upmost* e *downmost*, constrói-se o TLG. Os *matches* que ocorreram em ambos os alinhamentos definem nós de junção. Os nós de junção são aqueles onde as sequências se fundem, acrescido dos nós onde as sequências começam e terminam.

Uma vez gerados os TLGs para todos os pares de sequências, precisa-se

uní-los em um único TLG, uma vez que a entrada para o algoritmo de alinhamento múltiplo é apenas um grande TLG. Para tanto, realizam-se junções no conjunto de TLGs de pares e então submete-se o resultado para uma etapa de pós-processamento, onde são feitos reparos e otimizações no grafo visando, dentre outras coisas, minimizar o grafo. Observe que nesta etapa de pós-processamento deve ser garantido que o grafo é um DAG, ou seja, ciclos devem ser eliminados.

A junção de duas TLGs é realizada através das partes comuns entre elas. Para isso é necessário navegar em ambas as estruturas, atualizando rótulos quando necessário e possivelmente estabelecendo novos nós de junção ou criando novos nós onde as sequências diferem. Para detalhes na implementação desta etapa, consultar a tese de doutorado de João Meidanis [160]. Como exemplo de junção, tome os TLGs das Figuras C.3 e C.4. A Figura C.5 apresenta a junção deles.

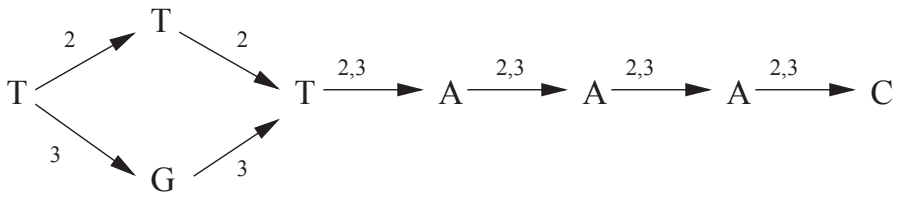


Figura C.4: TLG representando alinhamentos entre as sequências `TTTAAAC` e `TGTAAAC`.

O passo de pós-processamento (reparos) é responsável por eliminir possíveis ciclos e minimizar o grafo quando possível. Para entender melhor o porque é possível minimizar o grafo, observe que quando se tem três sequências A, B e C, por exemplo, e seus TLGs de pares AB, AC e BC, uma das possibilidades é juntar AB e BC formando uma TLG

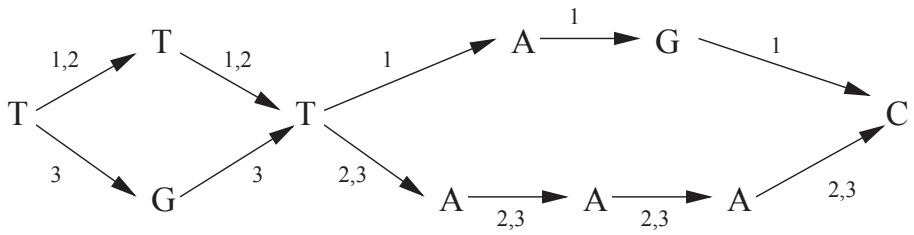


Figura C.5: TLG representando a junção dos TLGs apresentados nas Figuras C.3 e C.4.

com as três sequências. Note, entretanto, que na junção não havia qualquer informação sobre como A e C estão relacionados. É possível que ao juntar AB com BC tenha sido gerado um TLG onde ainda é possível colapsar nós entre A e C.

Este problema de reduzir o número total de nós tanto quanto possível, mantendo o significado original do TLG, é NP-Difícil [160]. Contudo, existem algoritmos eficientes (não exatos) com esta finalidade que resolvem o problema de forma satisfatória. Meidanis e Setubal apresentam um algoritmo para esta etapa que não garante um grafo mínimo, mas é eficiente e de implementação prática. Assumindo que a entrada seja um TLG, utiliza-se uma abordagem gulosa. O grafo é atravessado em ordem topológica e verificado se qualquer porção do caminho corrente alinha perfeitamente (de forma exata) com qualquer outro caminho percorrido anteriormente. É importante definir um limite inferior para o tamanho destes alinhamentos, caso contrário as alterações na estrutura podem não ser as desejadas. Os autores sugerem cinco caracteres como limite inferior.

Uma vez que se encontra um alinhamento perfeito de tamanho suficiente entre dois caminhos no grafo, é necessário verificar se estes são paralelos antes de realizar qualquer alteração na estrutura. Isso pode ser feito pela verificação das posições relativas dos pontos extremos dos caminhos. Sejam u,v e x,y os nós extremos dos caminhos. Se há um caminho que ligue v e x ou y e u, então os caminhos não são paralelos e, assim, não é possível reduzir a estrutura.

C.15.2 Algoritmo para alinhamento múltiplo

O algoritmo de alinhamento recebe como entrada um TLG, aquele resultante da etapa de pós-processamento. O algoritmo então introduzirá *gaps* no alinhamento quando caminhos ramificados no TLG não possuem o mesmo tamanho. Em outras palavras, o problema de alinhar TLG consiste em associar número de coluna para cada nó. Uma vez que se tenham tais números, basta preencher uma matriz inserindo, em cada linha, os caracteres da sequência correspondente nas colunas apropriadas. Deseja-se que haja espaço suficiente na matriz para os ramos mais longos, mas que não seja alocado mais espaço que o necessário.

Inicialmente calcula-se uma ordenação topológica para o TLG. Escolhe-se uma das

fontes e assoca-se o valor 0 a ela. A partir deste momento inicia-se uma sucessão de propagações *forward* e *backward* alternadamente até que não são encontrados novos nós extremos. Mantém-se duas variáveis auxiliares para cada nó, referentes a limites inferior (*lower bound*) e superior (*upper bound*). Atualizam-se os limites inferiores nos passos *forward* e os limites superiores nos passos *backward*. Tais variáveis são utilizadas para propagar valores.

Além desta variáveis ainda há uma fila para cada passo (`fwdqueue` e `backqueue`). É importante observar que estas filas seguem prioridades definidas pela ordenação topológica.

As principais unidades do algoritmo apresentado neste trabalho são exibidas nos Algoritmos 12, 13 e 14. Denota-se por $u \rightarrow v$ que o nó u alcança v considerando-se a orientação das arestas. As funções $GetMin$ e $GetMax$ retornam e removem o menor e o maior elemento da lista, respectivamente. A função $length(u, v)$ indica o número de arestas do menor caminho entre u e v.

Algoritmo 12: Rotina principal do algoritmo MSAGF.

Input: TLG
Inicializa todos os nós com $lb = -\infty$ e $ub = +\infty$
Calcula a ordenação topológica e armazena os índices dos nós
Seleciona um fonte, s, aleatoriamente
$lb(s) = ub(s) = 0$
$fwdqueue = \emptyset$
$backqueue = \emptyset$
$Insere(fwdqueue, s)$
$fwdflag = true$
repeat
    if $fwdflag$ **then**
        $ForwardPass(fwdqueue, backqueue)$
    else
        $BackwardPass(backqueue, fwdqueue)$
    $fwdflag = \mathbf{NOT}(fwdflag)$
until $Empty(fwdqueue)$ ***AND*** $Empty(backqueue)$

Uma vez concluído o processamento, utiliza-se o valor do limite inferior para definir a coluna do nó. A Tabela C.8 apresenta o resultado do processamento do algoritmo $MSAGF$ tendo como entrada o TLG da Figura C.5.

Seja n o número de nós no TLG e m o número de arestas. A complexidade de tempo do algoritmo apresentado neste trabalho é $O(m + n \log n)$. Quanto a espaço, a complexidade fica em $O(n)$.

Algoritmo 13: ForwardPass

```
Input: fwdqueue, backqueue
1  while NOT(Empty(fwdqueue)) do
2      node = GetMin(fwdqueue)
3      if node é um terminal then
4          if ub(node) = +∞ then
5              ub(node) = lb(node)
6              Insert(backqueue, node)
7      else
8          for todo v tal que node → v do
9              if lb(node) + length(node, v) > lb(v) then
10                 lb(v) = lb(node) + length(node, v)
11                 Insert(fwdqueue, v)
```

Algoritmo 14: BackwardPass

```
Input: backqueue, fwdqueue
1  while NOT(Empty(backqueue)) do
2      node = GetMax(backqueue)
3      if node é uma fonte then
4          if lb(node) = −∞ then
5              lb(node) = ub(node)
6              Insert(fwdqueue, node)
7      else
8          for todo u tal que u → node do
9              if ub(node) − length(u, node) < ub(u) then
10                 ub(u) = ub(node) − length(u, node)
11                 Insert(backqueue, u)
```

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

Tabela C.8: Resultado do alinhamento do TLG da Figura C.5.

C.15.3 Conclusão

As principais contribuições deste trabalho são a introdução de uma estrutura de dados (os TLGs) para entrada em algoritmo de alinhamento múltiplo e os algoritmos para a construção do alinhamento múltiplo baseado nelas. A flexibilidade de *gap* é outro ponto forte na solução.

O método parece ser interessante, mas foi muito pouco avaliado e otimizado. Não há qualquer referência de evolução deste a publicação deste trabalho em 1995.

C.16 Alinhamento Múltiplo de Proteínas via Algoritmo Genético Baseado em Tipos Abstratos de Dados

Este trabalho [213] apresenta uma solução para alinhamento múltiplo de proteínas, que faz uso de algoritmos genéticos. Para ser mais preciso, neste trabalho é apresentada uma instanciação de um algoritmo genético baseados em tipos abstratos de dados para MSA, de acordo com as definições apresentadas por Roberta Vieira em sua tese de doutorado [254].

Nesta abordagem, há a definição de conceitos como cromossomos, bases e genes. Um cromossomo representa um indivíduo do algoritmo genético e indica uma possível solução. Uma base é um par composto por um aminoácido (ou um *gap*) e um número natural. Bases são agrupadas em genes, que formam as características dos indivíduos. Em outras palavras, genes são as colunas de um alinhamento.

Além destes, tem-se ainda o conceito de bloco gênico e população. Um bloco gênico é composto por genes consecutivos e é utilizado na definição de algumas das operações. População é um conjunto de cromossomos.

Na montagem de um cromossomo cada sequência é alocada em uma linha e são inseridos *gaps*, representados pelo símbolo "-", para deixar todas as linhas com o mesmo comprimento. A *gaps* sempre há o natural 0 associado. A aminoácidos são associados números naturais maiores que 0, que indicam a posição do aminoácido na sequência original (sem *gaps*). Tal número é utilizado para não permitir que a ordem dos aminoácidos em uma sequência seja alterada.

Como grau de adaptação de um gene foi utilizado um método baseado na pontuação

de soma dos pares e, assim, a pontuação de um gene (coluna) é dada pela soma das pontuações de cada par de aminoácidos adicionado da soma das pontuações dos *gaps*. Por exemplo, uma forma simplista seria atribuir 1 para *match* entre aminoácidos, -1 para *mismatch* e -2 para cada *gap*. Outros exemplos de pontuação para pares de aminoácidos seria utilizar uma das tabelas de séries como PAM ou BLOSUM. Para *gaps* também é possível torná-la mais complexa levando em consideração sua posição ou a adjacência de outros *gaps*.

O grau de adaptação de um cromossomo é dado pela soma dos graus de seus genes. Adaptação média de uma população é dada pela média aritmética dos graus de adaptação de seus cromossomos.

Quanto às operações, são distribuídas em duas categorias: cruzamento e mutação. Cruzamento é uma operação onde dois cromossomos são combinados dando origem a um novo cromossomo. Mutações são operações que alteram um dado cromossomo na tentativa de fazê-lo melhorar quanto ao grau de adaptação.

Na categoria de cruzamento definiu-se apenas uma operação que recebeu o mesmo nome da categoria. Tal operação faz a combinação de forma que o cromossomo resultante seja ainda válido, ou seja, composto por sequências que caso removidos os *gaps* sejam iguais às sequências da entrada. O cromossomo resultante só sobreviverá para a próxima geração caso possua um grau de adaptação maior ou igual a adaptação média da população atual.

Na categoria de mutação, há as seguintes operações: inserção, supressão e mutação. Na inserção, uma coluna de *gaps* é inserida no cromossomo. Na supressão uma coluna de *gaps* é removida do cromossomo. É importante salientar que apesar de uma coluna de *gaps* não ter um significado biológico, definiu-se a operação de inserção com o objetivo de possibilitar uma maior variabilidade nos cromossomos. No resultado final do algoritmo deve-se remover todas as colunas de *gaps*. A mutação promove a mudança de posição de *gaps* em uma dada linha do cromossomo.

A população inicial é gerada aleatoriamente. Para cada cromossomo, adicionam-se q_{gaps} *gaps* na maior das sequências da entrada em posições aleatórias. O objetivo disto é permitir o deslocamento das bases desta maior sequência durante as aplicações do operador de mutação. Seja s_m a maior sequência da entrada. Definiu-se que $q_{gaps} = length(s_m)/25$. Depois disso adicionam-se *gaps*, também em posições aleatórias, no restante das sequências até que todas elas possuam o mesmo tamanho. Convencionou-se que o tamanho da população inicial é de 100 cromossomos.

O processo evolutivo começa com a definição do critério de preservação sobre a população atual. O critério utilizado foi um ponto de corte quanto ao grau de adaptação. Selecionam-se os cromossomos cujo grau seja maior ou igual ao grau de adaptação médio da população atual. Tal subconjunto será utilizado para os cruzamentos. Para as mutações

são selecionados todos aqueles cromossomos abaixo do ponto de corte. Os cromossomos que melhorarem o grau com a mutação, vão para a próxima geração. A população, que começa com tamanho 100, pode variar de tamanho de uma geração para outra, mas cresce no máximo até 200. Tal decisão deveu-se a restrições de *hardware*. A iteração é interrompida quando há uma convergência nos resultados.

Avaliou-se a solução com BAliBASE [248], que, além de permitir uma comparação simples de novas soluções com as existentes, provê diversos conjuntos de entrada divididos por tamanho e grau de similaridade entre as sequências.

Diversos testes foram realizados a fim de determinar o tamanho máximo da população. Decidiu-se por 200 pelo fato de ter apresentado uma boa relação entre tempo de execução e qualidade dos resultados. Outro fator importante nesta decisão foi a grande limitação de memória no *hardware* disponível para os testes.

Testes também foram realizados para comparar as diversas matrizes de substituição: BLOSUM62, Dayhoff, PAM40, PAM80, PAM120 e PAM250. Neste caso, o melhor resultado apresentado foi quando utilizou-se a matriz Dayhoff. Esta obteve uma pontuação no BAliBASE mais elevada para as sequências utilizadas nos testes (conjunto `1aab`).

É importante salientar que este resultado no máximo indica que entradas da mesma família (conjuntos de sequências semelhantes ao avaliado) devem ter resultados minimamente satisfaztórias. Para outras famílias, porém, deve-se verificar qual a melhor matriz a se utilizar.

Outros testes foram feitos para avaliar se deveria-se utilizar reprodução sexuada ou assexuada, ou seja, se os pais utilizados em cruzamentos deveriam ou não ser divididos em duas partições (machos e fêmeas).

Os testes apontaram que o melhor era fazer uma reprodução sexuada, dividindo os pais em dois conjuntos sem intersecção e permitir apenas o cruzamento entre cromossomos de conjuntos distintos. Isso porque reduz o número de combinações, o que afeta diretamente no tempo de execução, e por promover uma maior diversidade genética da população.

Uma vez especificados os parâmetros para o algoritmo genético, foram feitos os alinhamentos múltiplos de quatro outros conjuntos de entradas semelhantes extraídos da mesma família que `1aab`, foram eles: `1fj1A`, `1hpi`, `1csy` e `1tgxA`.

Calculadas as pontuações BAliBASE para estes alinhamentos, pôde-se comparar seu desempenho com uma série de soluções existentes: PRRP, Clustal X, SAGA, DiAlign, SBpima, MLpima, MultiAlign, Pileup, MULTAL e HMMT.

C.16.1 Conclusão

Os resultados indicam que a solução tem potencial, mas precisa ainda de uma série de evoluções e avaliações. Comparando diretamente as pontuações BAliBASE, com as outras

soluções, a solução vence apenas do HMMT e do SAGA (de forma pouco significativa, neste caso) para `1aab`, HMMT para `1fjlA`, HMMT e MULTAL (muito pouco significativo) para `1tgxA`.

Foram apenas 5 "vitórias" em 50 comparações. Fora que venceu apenas uma vez do SAGA e de forma pouco significativa. É importante salientar que o SAGA é a maior referência no que se refere a solução por algoritmos genéticos para MSA. Para a solução começar a ganhar credibilidade precisará apresentar resultados significativamente melhores que o SAGA.

São necessários mais testes da solução. Por exemplo, o BAliBASE é composto por 142 alinhamentos de referência divididos em diversas categorias. O objetivo de cada categoria é avaliar a solução em uma determinada situação. No caso dos testes realizados, até então, foram alinhados apenas 5 dos 142 conjuntos de que dispõe o BAliBASE e, além disso, todos os conjuntos de entrada pertenciam a mesma categoria. Para um teste mais preciso é necessário avaliar a solução nestas diversas soluções, pois é comum que uma dada solução seja muito boa em algumas categorias e apresente maus resultados para outras.

C.17 A Graph-Based GA for the MSA Problem

Este trabalho [154] apresenta uma solução para alinhamento múltiplo, que faz uso de algoritmos genéticos e descreve uma nova forma de representar um alinhamento através de grafos orientados multidimensionais. Desta forma consegue reduzir dramaticamente a complexidade de armazenamento.

A partir das n sequências de entrada, constrói-se uma matriz n-dimensional, na qual cada dimensão corresponde a uma sequência. Indivíduos no GA representam um caminho na matriz, que é assim um grafo orientado multimensional. Um caminho válido sempre inicia na célula correspondente ao primeiro aminoácido de todas as sequências e acaba na célula correspondente ao último aminoácido de todas as sequências.

Para representar o grafo usa-se um cromossomo de tamanho variável. Este cromossomo é composto por m genes, cujo número será no mínimo equivalente ao tamanho da maior das sequências e no máximo a soma dos tamanhos da sequências decrescido de um. Cada gene é composto de dígitos binários, que indicam a direção da próxima célula e que desta forma indica como os aminoácidos, associados as células de origem e destino, são alinhados. Por exemplo, para o alinhamento de duas sequências, um gene pode receber três valores: 01, 10, 11. Cada um deles associado com um deslocamento: célula a esquerda, célula abaixo ou célula na diagonal. O número de dígitos será diretamente proporcional ao número de sequências.

Para avaliar uma solução candidata, decodifica-se o cromossomo em um MSA. Em seguida calcula-se a pontuação SP para o alinhamento e normaliza-se o valor para poder

comparar com outros alinhamentos.

Três são os operadores genéticos nesta solução, são eles: cruzamento, mutação aleatória e mutação dinâmica. Todos eles foram projetados de forma a preservar a integridade e validade do caminho (solução).

A operação de cruzamento recebe como entrada dois cromossomos e seleciona um gene aleatoriamente. A partir deste momento, passa a combinar os cromossomos de entrada, copiando a parte inicial do primeiro e a parte final do segundo. Depois disso, muito provavelmente, ainda será necessário completar o caminho do cromossomo resultante. Para isso é utilizado o mesmo procedimento aleatório que gerou a população inicial.

A operação de mutação aleatória tem como objetivo aumentar a diversidade genética da população. Ela recebe como entrada um cromossomo e escolhe aleatoriamente dois genes e uma dimensão. A seguir inverte os *bits* correspondentes àquela dimensão e então realiza os ajustes necessários para manter a validade do caminho. Tal ajuste é necessário caso os *bits* não sejam complementares.

O operador de mutação dinâmica também tem o objetivo de aumentar a diversidade genética da população. Ele recebe como entrada um cromossomo e escolhe aleatoriamente um segmento do alinhamento (colunas consecutivas). O tamanho deste segmento é fixado por um parâmetro do algoritmo. A seguir é realizado um alinhamento dinâmico progressivo nas colunas correspondentes ao segmento selecionado.

C.17.1 Conclusão

A solução apresentada resulta em uma grande redução na complexidade de memória quando comparada com a solução exata obtida pela extensão direta do algoritmo de Needleman e Wunsch [174]. A solução proposta tem complexidade $O(N \times L)$ enquanto Needleman e Wunsch tem $O(L^N)$, para N sequências e L tamanho médio das sequências.

As avaliações realizadas ficaram restritas a uma comparação de desempenho com o Clustal utilizando o BAliBASE. Os resultados foram desfavoráveis, mas ainda mais crítico que isso é ausência de comparação com a solução MSA de referência usando GA, o SAGA.