

UNIVERSIDADE REGIONAL DE BLUMENAU
CENTRO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

**RECONSTRUÇÃO FILOGENÉTICA EM
AMBIENTE DISTRIBUÍDO**

FELIPE FERNANDES ALBRECHT

BLUMENAU
2006

2006/II-14

FELIPE FERNANDES ALBRECHT

**RECONSTRUÇÃO FILOGENÉTICA EM
AMBIENTE DISTRIBUÍDO**

Trabalho de Conclusão de Curso submetido
à Universidade Regional de Blumenau para
a obtenção dos créditos na disciplina Traba-
lho de Conclusão de Curso II do curso de
Ciências da Computação - Bacharelado.

Prof. Jomi Fred Hübner – Orientador

**BLUMENAU
2006**

2006/II-14

RECONSTRUÇÃO FILOGENÉTICA EM AMBIENTE DISTRIBUÍDO

Por

FELIPE FERNANDES ALBRECHT

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente: Prof. Jomi Fred Hübner – Orientador, FURB

Membro: Prof. Paulo César Rodacki Gomes, FURB

Membro: Prof. Mauro Mattos, FURB

Dedico este trabalho a todos os professores, verdadeiros mestres e pesquisadores.
Aqueles que fazem do conhecimento mais do que um amontoado de informações,
mas um modo para se buscar um mundo melhor.

Freude trinken alle Wesen
An den Brüsten der Natur;
Alle Guten, alle Bösen
Folgen ihrer Rosenspur.
Küsse gab sie uns und Reben,
einen Freund, geprüft im Tod;
Wollust ward dem Wurm gegeben,
und der Cherub steht vor Gott.
- Friedrich Schiller

AGRADECIMENTOS

Agradeço especialmente a minha namorada Emanuelle, pela compreensão e carinho proporcionados durante a elaboração deste trabalho e nesses anos de participação de suma importância em minha vida.

Aos meus pais: minha mãe Rogéria, pelo amor aos livros e as pesquisas e ao meu pai Carlos, pela gosto com a matemática e ciências exatas. Aos professor Alberto M. R. Dávila da FioCruz, pelo apoio dado a elaboração deste trabalho e demais pesquisas na área da bioinformática. Ao amigo Henrique, por auxiliar em diversos assuntos na biologia. Aos professores Geraldo Moretto e André Paulo Nascimento do curso de Ciências Biológicas da FURB, pelo auxílio e motivação inicial. Ao Professor Jomi Fred Hübner, pelo auxílio a elaboração deste trabalho.

RESUMO

Este trabalho apresenta otimizações para um *workflow* de filogenias de proteínas homólogas distantes e um algoritmo paralelo para inferência de árvores filogenéticas utilizando o método de *Least Squares*. O algoritmo possui heurísticas para minimizar o seu tempo total de processamento e estas heurísticas podem ser utilizadas em outros algoritmos de inferência filogenética que utilizem o método *Least Squares*. Para validar o algoritmo, implementa-o utilizando o padrão *Message Passing Interface* (MPI) e executado-o. Assim, os seus tempos de execução são comparados com os de um software já existente. Os resultados, dependendo dos parâmetros utilizados, apresentaram redução de tempo do processamento, porém, as árvores inferidas não são ótimas. As otimizações no *workflow*, sendo uma delas, um agendador para múltiplas execuções num *cluster*, minimizaram o tempo total de execução.

Palavras Chave: Bioinformática. Algoritmos distribuídos. Filogenética.

ABSTRACT

This work presents optimizations for a distant homologous proteins phylogenies workflow and a parallel algorithm to inference of phylogenetic trees using Least Squares method. The algorithm has heuristics to minimize the total processing time and they can be utilized in others algorithms that uses this method. To validate the algorithm, it is implemented utilizing the MPI standard. The result, depending the utilized parameters, shows a processing time reduction , however, the inferred trees do not be the bests. The workflow optimizations, being one of them, a multiple execution scheduler on a MPI cluster, they had minimized the total execution time of the workflow.

Key-Words: Bioinformatics. Distributed Algorithms. Phylogenetics

LISTA DE ILUSTRAÇÕES

Figura 2.1 – Dogma central da biologia molecular	19
Quadro 2.1 – Lista dos aminoácidos	20
Quadro 2.2 – Estrutura primária da Hemoglobina	20
Figura 2.2 – Estrutura terciária da hemoglobina	21
Figura 2.3 – Duplicação gênica	22
Figura 2.4 – Genes homólogos	23
Figura 2.5 – Resultado da especiação nos genes	23
Figura 2.6 – Genes parálogos	24
Figura 2.7 – Genes ortólogos	24
Figura 2.8 – Crescimento da quantidade de dados de seqüência no <i>GenBank</i>	25
Figura 3.1 – Funcionamento do MPI_Send e do MPI_Recv	32
Figura 3.2 – Funcionamento do MPI_Isend e do MPI_Irecv	33
Figura 3.3 – Envio de dados a diversos nós utilizando ponto a ponto	33
Figura 3.4 – Envio de dados a diversos nós utilizando <i>broadcast</i>	34
Figura 3.5 – Criação de novo tipo na linguagem de programação <i>C</i>	34
Figura 3.6 – Descrição de uma estrutura para o <i>MPI</i>	35
Figura 3.7 – Criação de um vetor em <i>MPI</i>	35
Figura 3.8 – Populando um vetor de <i>pessoa_t</i>	36
Figura 4.1 – Diagrama de evolução de espécies por Charles Darwin	38
Figura 4.2 – Uma árvore representando filogenia entre cinco táxons	39
Figura 4.3 – Uma árvore sem raiz representando filogenia entre cinco táxons	40
Figura 4.4 – Uma árvore exibindo os comprimentos dos ramos	41
Quadro 4.1 – Alinhamento múltiplo para a inferência filogenética	41
Figura 4.5 – Árvore representando uma inferência filogenética através da máxima parcimônia	42
Figura 4.6 – Fluxo para a criação das matrizes de distância	44
Figura 4.7 – Árvore com o comprimento dos ramos e a matriz de distâncias utilizada	46

Quadro 4.2 – Cálculo do menor quadrado	47
Figura 5.1 – Processo de execução do <i>workflow</i> proposto por Theobald e Wuttke (2005)	50
Quadro 6.1 – Tempo de execução do <i>workflow</i> sem alterações	53
Figura 6.1 – Seqüências de execução do agendador de múltiplas execuções do <i>compass</i>	56
Figura 6.2 – Ganho com a utilização do agendador do <i>compass</i>	57
Figura 7.1 – Árvore ótima	60
Figura 7.2 – Poda de uma árvore	61
Figura 7.3 – Poda de uma árvore (continuação)	62
Figura 7.4 – Pesquisa pela melhor árvore	67
Quadro 7.1 – Cálculo dos trios mais próximos	69
Figura 7.5 – Diagrama de estados do processo agendador	79
Figura 7.6 – Diagrama de estados dos processos agendadores	79
Figura 7.7 – Diagrama de distribuição do algoritmo	80
Quadro 7.2 – Estrutura do tipo <i>tree_t</i>	82
Quadro 7.3 – Estrutura do tipo <i>taxon_node_t</i> , que representa um táxon na árvore .	82
Quadro 7.4 – Estrutura do tipo <i>internal_node_t</i> , que representa um nó interno na árvore	83
Figura 7.8 – Principais arquivos da implementação	84
Quadro 7.5 – Estrutura que representa as informações a serem enviadas de uma árvore	84
Quadro 7.6 – Tipo TREE_INFO_MPI que representa ao MPI o tipo <i>tree_info_t</i> . .	85
Quadro 7.7 – Matriz de distâncias do pacote <i>PHYlogeny Inference Package</i> (PHYLIP)	85
Quadro 7.8 – Exemplo de árvore num formato de texto	86
Figura 7.9 – Tempos das execuções	88

LISTA DE ALGORITMOS

7.1	Calcular o <i>Least Square</i> da árvore	60
7.2	Algoritmo proposto por Felsenstein (1997)	64
7.3	Criação e seleção dos trios mais próximos	68
7.4	Reorganizando os comprimentos dos ramos de uma árvore	71
7.5	Podando um nó	72
7.6	Agendador para o algoritmo paralelo	75
7.7	Trabalhador para o algoritmo paralelo	77
7.8	Função para geração de assinatura das árvores	78

SUMÁRIO

1	INTRODUÇÃO	14
1.1	OBJETIVOS DO TRABALHO	15
1.2	ESTRUTURA DO TRABALHO	16
2	GENÉTICA MOLECULAR E BIOLOGIA MOLECULAR	17
2.1	BIOLOGIA MOLECULAR	17
2.2	PROTEÍNAS	19
2.3	GENÉTICA MOLECULAR	21
2.4	BIOINFORMÁTICA	25
3	SISTEMAS E ALGORITMOS DISTRIBUÍDOS	27
3.1	ALGORITMOS DISTRIBUÍDOS	27
3.1.1	Modelo de rede de comunicação síncrona	28
3.2	PADRÃO MPI	30
3.2.1	Envio e recepção de mensagens	31
3.2.2	Definição de novos tipos de dados	34
4	FILOGENÉTICA	37
4.1	HISTÓRIA DA FILOGENÉTICA	37
4.1.1	Representações de Filogenias	39
4.2	TÉCNICAS DE FILOGENÉTICA MOLECULAR	40
4.2.1	Taxonomia cladística	40
4.2.2	Taxonomia numérica	43
5	FILOGENÉTICA DE HOMOLOGIAS DISTANTES DE PROTEÍNAS	48
5.1	INTRODUÇÃO A HOMOLOGIA DISTANTES DE PROTEÍNAS	48
5.1.1	<i>Workflow</i> para inferência de filogenias distantes	49
6	DESENVOLVIMENTO DAS OTIMIZAÇÕES DO <i>WORKFLOW</i>	53

6.1	ANÁLISE E OTIMIZAÇÃO DO <i>WORKFLOW</i>	53
6.1.1	Obtenção das seqüências similares	54
6.1.2	Pesquisa por seqüências similares	54
6.1.3	Comparação dos alinhamentos múltiplos	54
6.1.4	Otimização da Criação dos Alinhamentos Múltiplos	58
6.2	CONCLUSÕES SOBRE OTIMIZAÇÕES NO <i>WORKFLOW</i>	58
7	INFERÊNCIA FILOGENÉTICA DISTRIBUÍDA	59
7.1	ALGORITMO ALTERNATIVO PARA INFERÊNCIA FILOGENÉTICA UTILIZANDO <i>LEAST SQUARES</i>	59
7.1.1	Funcionamento do algoritmo alternativo para inferência filogenética utili- zando <i>Least Squares</i>	62
7.2	ALGORITMO PARALELO	63
7.2.1	Análise dos software <i>kitsch</i> e <i>fitch</i>	63
7.2.2	Heurísticas	66
7.2.3	Paralelização do método de inferência filogenética utilizando <i>Least Square</i> .	71
7.3	RESULTADOS	87
7.3.1	Dificuldades encontradas	88
8	CONCLUSÃO	90
8.1	EXTENSÕES	91
	REFERÊNCIAS BIBLIOGRÁFICAS	93

1 INTRODUÇÃO

Desde o início da história, a humanidade preocupa-se em compreender a vida e suas origens. Diversos foram os filósofos e cientistas que propuseram teorias e métodos para explicar a origem da vida. Com a publicação do livro “A Origem das Espécies” por Charles Darwin em 1859, o conceito da evolução foi apresentada pela primeira vez. A Teoria da Evolução diz que os organismos sofrem mutações entre diferentes gerações e as modificações vantajosas são perpetuadas, enquanto as desvantajosas são eliminadas pela seleção natural. Com os conceitos propostos por Charles Darwin, é possível analisar as mudanças que ocorreram nas espécies de seres vivos e propor uma linhagem evolutiva delas. Através destes conceitos, pode-se afirmar que os seres humanos e demais espécies de primatas possuem uma espécie ancestral em comum.

O estudo das relações evolucionárias entre espécies de seres vivos, tanto vivas quanto extintas e a inferência de possíveis árvores evolutivas é denominado de filogenética (MOUNT, 2004). Este estudo era feito primordialmente pela observação das características morfológicas, ou seja, através da aparência e funcionamento dos órgãos e sistemas dos seres vivos. Com o advento da genética molecular, onde são estudadas principalmente seqüências genéticas e protéicas, a filogenética passa a utilizar estas informações moleculares. O estudo da filogenética destes dados tem como principal objetivo inferir árvores evolutivas destas seqüências e das espécies que as possuem com o maior grau de confiabilidade possível. Esta nova abordagem de filogenética utilizando dados moleculares é chamado de filogenética molecular. Desta forma, a filogenética molecular descreve a origem e evolução de seqüências genéticas e protéicas e, segundo Mount (2004, p. 282), uma análise filogenética de uma família de ácidos nucléicos ou de proteínas relacionadas é a determinação de como os membros desta família derivaram-se durante a evolução.

Em diversas situações é utilizado o termo reconstrução filogenética, para denotar uma inferência filogenética. Isto porquê, através de informações, visíveis ou molulares, sobre os seres vivos e do estudo destes dados, pretende-se reconstruir, ou inferir, uma árvore mais próxima possível da verdadeira, formada pela evolução dos organismos.

Nas pesquisas de filogenias mais complexas, é comum a utilização de *workflows*, que são um conjunto de softwares, cada qual com sua função específica, que executam uma operação. Para a filogenética, utiliza-se um *workflows* composto por um software de pesquisa de seqüências em banco de dados, outro que recebe estas seqüências e faz um

alinhamento delas, ressaltando suas semelhanças e por fim um software que lê o resultado do alinhamento das seqüências e reconstrói uma árvore filogenética baseada nestes dados.

Um problema comum a todos os métodos de reconstrução de árvore filogenética é a alta necessidade computacional caso o número de seqüências seja alto. Para resolver esta questão, a principal solução seria distribuir o problema entre diversos processadores. Desta forma, duas soluções são possíveis: a utilização de supercomputadores ou a de *clusters*. A utilização de supercomputadores esbarra no alto custo desses equipamentos, inacessíveis para diversas instituições de pesquisa e uma solução que se destaca em ambientes distribuídos são os *clusters beowulf*.

Cluster é um termo largamente utilizado e significa uma interligação de computadores através de software e rede independentes num único sistema, ou seja, uma interligação de computadores independentes para resolverem um problema em comum. Os *clusters* podem ser utilizados para sistemas *High Availability* (HA) para garantir alta disponibilidade do sistema ou em *High Performace Computing* (HPC) para proporcionarem poder computacional maior do que um único computador proporcionaria (STERLING, 2002).

Os *clusters beowulf* são de desempenho escalável baseados em *hardware* facilmente encontrado no mercado, em sistemas de redes comuns e tendo como infraestrutura o software livre. Os *clusters beowulf* possuem alta adaptabilidade, podendo ser formados por dois nodos conectados via *ethernet* ou ser um complexo sistema de 1024 nodos conectados através de rede de alta velocidade (BOWULF..., 2004).

A comunicação entre os nodos de um *cluster beowulf* é feita através de bibliotecas de troca de mensagens. Atualmente o principal padrão é o MPI (MESSAGE..., 2006). Possui diversas implementações que são utilizadas como bibliotecas nos programas a serem implementados, fazendo abstração da comunicação entre os nodos. É importante ressaltar que os softwares executados em *clusters beowulf* devem ser preparados para isto, utilizando algoritmos para processamento distribuído e tendo na sua implementação, uma biblioteca para a comunicação entre os nodos.

1.1 OBJETIVOS DO TRABALHO

A filogenética é uma importante área de estudo, pois nela são feitos os estudos de inter-relações parentescas entre táxons¹ são ordenados de acordo com sua relação parentescas e através dela pode-se conhecer as relações evolucionárias entre os seres vivos. Porém, para efetuar estes estudos, são necessários diversos processos com alto custo com-

¹Táxon é uma unidade associada a um sistema de classificação. Táxons (ou taxa) podem estar em qualquer nível de um sistema de classificação podendo ser um reino, um gênero, uma espécie ou qualquer outra unidade de um sistema de classificação dos seres vivos.

putacional, que podem ser significativamente lentos se forem executados em um único computador.

Com a intenção de reduzir o tempo de reconstrução de árvores filogenéticas, surge a idéia de distribuir este processo. Através da utilização de padrões de comunicação abertos, softwares livres e com a reutilização de softwares já existentes, esta nova ferramenta auxiliará, na redução do tempo computacional, diversos estudos de filogenética.

O objetivo deste trabalho é disponibilizar uma ferramenta para a inferência de árvores filogenéticas em um ambiente distribuído.

Os objetivos específicos do trabalho são:

- a) propor um algoritmo para inferência de árvores filogenéticas em ambiente distribuído;
- b) implementar o algoritmo num software de reconstrução de árvores filogenéticas já existente;
- c) substituir o software PAUP* (SWOFFORD, 2004) no *workflow* proposto por Theobald e Wuttke (2005) pelo software desenvolvido neste trabalho.

Com o início deste trabalho, percebeu-se que o tempo de execução do *workflow* proposto por Theobald e Wuttke (2005) é longo não pela inferência filogenética, mas pelos processos executados anteriormente para obtenção dos dados necessários à inferência. Desta forma, acrescentou-se um novo objetivo no trabalho, que é a otimização deste *workflow* em busca de minimizar o seu tempo de execução.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está dividido em 7 capítulos. O segundo capítulo introduz os conceitos de biologia e genética molecular e bioinformática. O capítulo 3 apresenta as principais características dos sistemas e algoritmos distribuídos e o padrão MPI. No capítulo 4 é apresentada a filogenética, os seus primórdios, a filogenética molecular e as técnicas utilizadas. No quinto capítulo é apresentado a filogenia de homolias distantes e o *workflow* proposto por Theobald e Wuttke (2005) e no capítulo seguinte, são apresentadas as otimizações feitas no *workflow*. O sétimo capítulo apresenta o algoritmo proposto por Felsenstein (1997) e seu funcionamento. Em seguida é apresentado o processo de desenvolvimento do algoritmo paralelo, como a análise dos softwares *fitch* e *kitsch*, as heurísticas propostas e por fim o algoritmo paralelo e as principais dificuldades no seu desenvolvimento. O capítulo 8 apresenta as conclusões referentes a este trabalho e sugestões para futuras extensões.

2 GENÉTICA MOLECULAR E BIOLOGIA MOLECULAR

Neste capítulo é apresentada uma breve introdução sobre os conceitos de biologia molecular, proteínas e genética molecular, importantes para o leitor contextualizar o trabalho e poder compreender o problema de Inferência Filogenética sob a ótica biológica. Na primeira seção, tem-se a introdução sobre o genoma, os genes e a síntese de proteínas e na seguinte, apresenta as proteínas, sua importância e suas principais características.

2.1 BIOLOGIA MOLECULAR

A Biologia Molecular estuda as reações químicas e as moléculas presentes nos seres vivos. Nesta seção é introduzida a biologia molecular, a saber: as moléculas de Ácido Desoxirribonucléico (DNA), Ácido Ribonucléico (RNA) e proteínas. As moléculas de DNA e RNA são importantes, pois nelas são carregadas o material genético dos seres vivos e as proteínas, que são os blocos que constituem os seres vivos.

Estima-se que existam de dez milhões - talvez cem milhões - de espécies que atualmente habitam a Terra (ALBERTS et al., 2004, p. 2). Cada uma destas espécies possui características próprias e meios de reprodução para que a espécie e conseqüentemente suas características sejam perpetuadas. As características de cada espécie são hereditárias e estão armazenadas no seu genoma.

Ainda segundo Alberts et al. (2004, p. 2), o genoma é a informação genética total carregada por uma célula ou um organismo. No genoma estão todas as características genéticas, ou seja, o genoma de um organismo contém as informações de como o organismo é. Alberts et al. (2004, p. 199) dizem que um gene é, normalmente, definido como um segmento de DNA que contém as instruções para produzir uma determinada proteína (ou uma série de proteínas relacionadas) e complementa afirmando que existe uma relação entre a complexidade de um organismo e o número total de genes em seu genoma. Por exemplo, o número total de genes varia entre menos de 500 em bactérias simples a cerca de trinta mil nos humanos. Os genes dos seres vivos estão em moléculas chamadas cromossomos. Sobre os cromossomos, os mesmos autores dizem que cromossomo é a estrutura composta por uma molécula de DNA muito longa e proteínas associadas que contém parte (ou toda) da informação genética de um organismo. Os seres humanos possuem 23 pares de cromossomos em cada célula do organismo, com exceção das células sexuais, que possuem apenas uma cópia dos 23 cromossomos.

Os genes são segmentos de DNA, que é uma molécula composta por uma seqüência

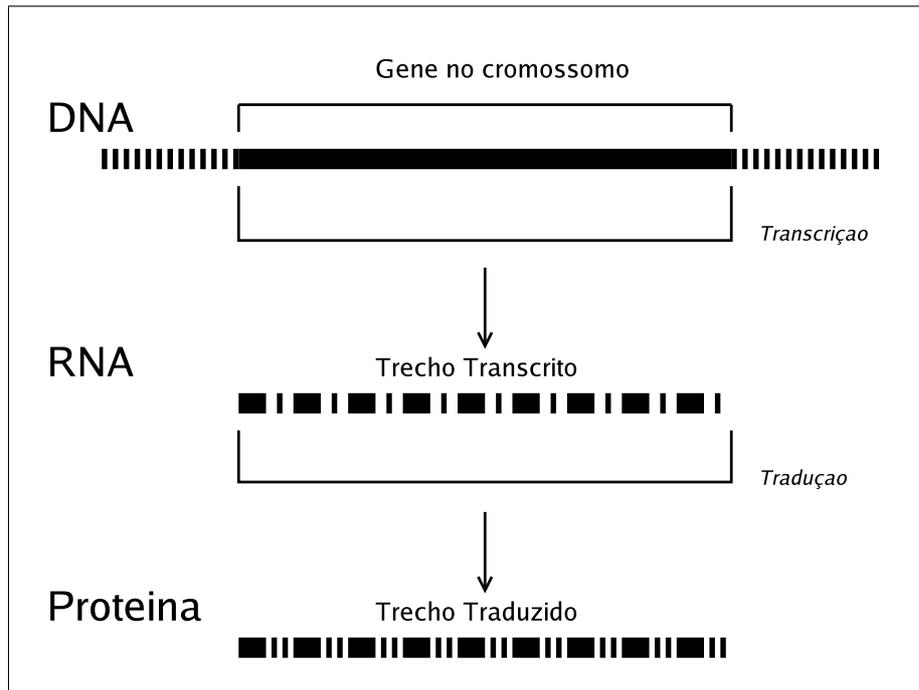
de moléculas idênticas, chamadas nucleotídeos. Cada um dos nucleotídeos que formam o DNA é constituído por três partes: um açúcar de cinco carbonos, a desoxirribose, um fosfato e uma base nitrogenada. São quatro as possíveis bases nitrogenadas, ou seja, o DNA tem quatro nucleotídeos diferentes, que são: a adenina, a guanina, a citosina e a timina. É importante saber que o DNA é formado por uma fita dupla complementar. Desta forma, cada uma das bases dos nucleotídeos do DNA possui uma base complementar a ela. A base complementar da adenina é a citosina e da guanina é a timina e vice-versa. As bases do DNA podem ser classificadas de acordo com suas propriedades químicas como purinas, a adenina e a guanina ou como pirimidinas, a citosina e a timina ou uracila, no caso do RNA.

Para representar as seqüências de DNA, utiliza-se a primeira letra de cada base: *A* para adenina, *C* para citosina, *G* para guanina e *T* para a timina. Um exemplo de seqüência de DNA é: *ACTCGGTAC* e sua seqüência complementar é: *CAGATTGCA*.

Strachan e Read (2002, p. 1) afirmam que em todas as células, a informação genética está armazenada nas moléculas de DNA. Ele complementa dizendo que regiões específicas das moléculas de DNA servem como moldes para a síntese de moléculas de RNA. O RNA é molecularmente muito similar ao DNA, as diferenças são que o RNA é uma fita simples e a base timina no DNA é substituída pela uracila, *U*, no RNA. As moléculas de RNA são utilizadas direta ou indiretamente para a expressão gênica. A expressão gênica é o processo de leitura dos genes armazenados nos cromossomos, da criação de seqüências de RNA baseadas nessas seqüências e, por fim, a síntese de proteínas utilizando como moldes estas seqüências de RNA criadas a partir da leitura do DNA.

A Figura 2.1 exhibe os principais passos da expressão gênica. Passos foram omitidos para proporcionar melhor compreensão. Caso deseja-se aprofundar no assunto, recomenda-se a leitura de Alberts et al. (2004) e Strachan e Read (2002).

Observa-se na Figura 2.1 os dois passos para a expressão gênica: o primeiro, chamado de transcrição e ocorre no núcleo da célula, é a leitura do gene na forma de seqüência DNA e a criação de uma seqüência de RNA complementar a seqüência de DNA e no segundo passo, chamado de tradução e ocorre nos ribossomos da célula, é a síntese de proteínas utilizando como molde a seqüência de RNA criada no primeiro passo. O processo de leitura do DNA até a síntese da proteína é conhecido como *dogma central da biologia molecular* e ele segue o fluxo DNA para RNA para proteínas (STRACHAN; READ, 2002, p. 9).



Fonte: baseado em Strachan e Read (2002, p. 11).

Figura 2.1 – Dogma central da biologia molecular

2.2 PROTEÍNAS

As proteínas são os blocos que constituem as células e executam praticamente todas as funções celulares. Elas constituem a maior parte da massa molecular seca das células (ALBERTS et al., 2004, p. 129).

Alberts et al. (2004, p. 129) citam como algumas funções das proteínas: o intercâmbio de moléculas em nível intra e extracelular, transmissora de mensagens entre células, catalizadoras, constituintes das organelas citoplasmáticas e conseqüentemente das células. Os anticorpos, toxinas e hormônios também são proteínas. Sendo que as moléculas de DNA e RNA e as proteínas é que fazem a leitura, interpretação do DNA e a síntese de novas proteínas.

As proteínas são formadas por aminoácidos. Existem 20 tipos diferentes de aminoácidos e cada um pode ser identificado pelo seu nome, sua abreviatura ou pela sua sigla. Cada aminoácido possui propriedades químicas diferentes, sendo as principais propriedades a sua polaridade e sua hidrofobicidade. A polaridade diz respeito a carga eletromagnética do aminoácido, podendo ser polar negativa, polar positiva, polar não carregada e apolar. A hidrofobicidade indica o grau de repulsão à água do aminoácido. Os aminoácidos apolares são hidrofóbicos, ou seja, eles evitam ao máximo o contato com a água. A importância de se conhecer a polaridade dos aminoácidos se dá, porque a estrutura tridimensional, a forma, de uma proteína depende desta característica dos

aminoácidos que a compõe, e o que define a função de uma proteína no organismo é a sua forma tridimensional. No Quadro 2.1 são apresentados os 20 aminoácidos existentes, suas abreviaturas, siglas e sua polaridade.

Aminoácido	Sigla	Símbolo	Carga elétrica
Ácido aspártico	Asp	D	negativa
Ácido glutâmico	Glu	G	negativa
Arginina	Asg	R	positiva
Lisina	Lys	K	positiva
Histidina	His	H	positiva
Asparagina	Asn	N	polar não-carregada
Glutamina	Gln	Q	polar não-carregada
Serina	Ser	S	polar não-carregada
Treonina	Thr	T	polar não-carregada
Tirosina	Tyr	Y	polar não-carregada
Alanina	Ala	A	apolar
Glicina	Gly	G	apolar
Valina	Val	V	apolar
Leucina	Leu	L	apolar
Isoleucina	Ile	I	apolar
Prolina	Pro	P	apolar
Fenilalanina	Phe	F	apolar
Metionina	Met	M	apolar
Triptofano	Trp	W	apolar
Cisteína	Cys	C	apolar

Quadro 2.1 – Lista dos aminoácidos

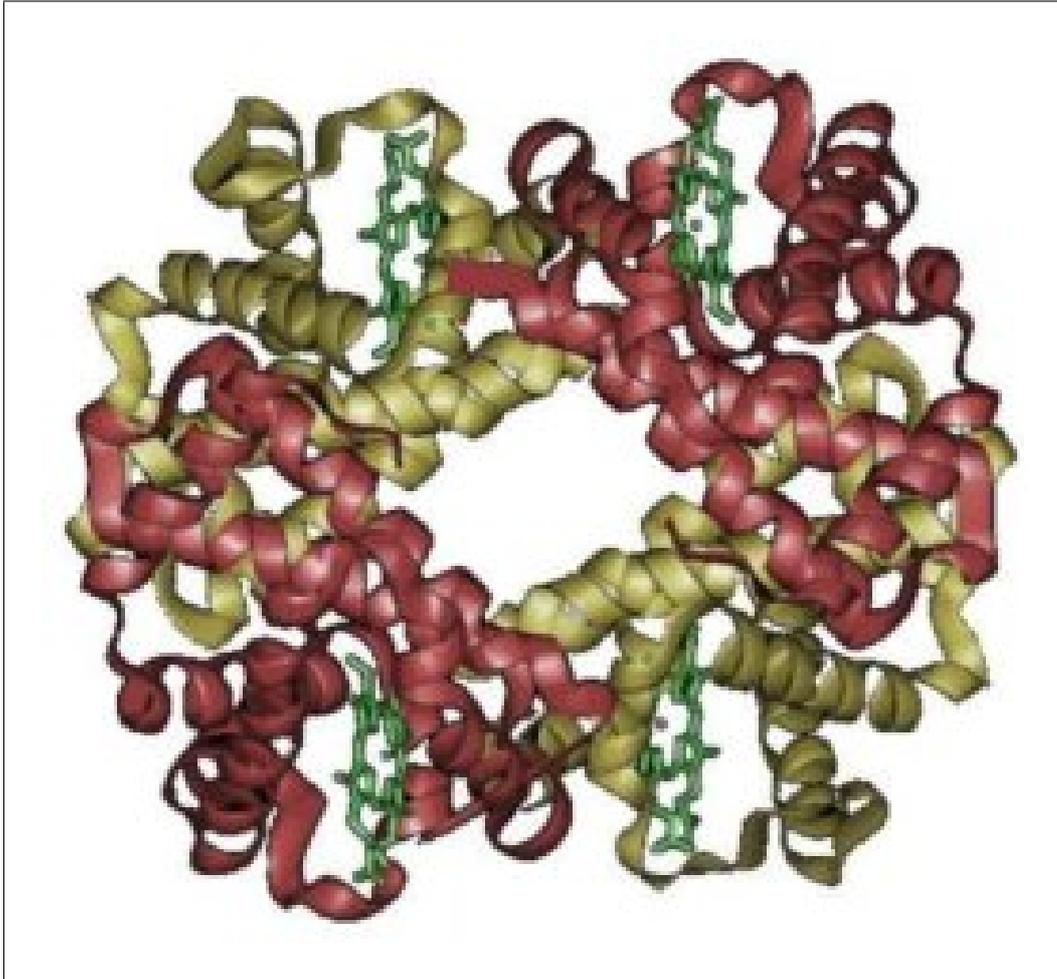
As proteínas são representadas pela sua estrutura primária, ou seja, a seqüência de aminoácidos que as compõem. Um exemplo de proteína é a hemoglobina, que tem sua estrutura primária exibida no Quadro 2.2 e sua estrutura terciária é apresentada na Figura 2.2.

Por sua vez, cada proteína tem uma estrutura tridimensional, mas proteínas com estruturas primárias diferentes podem ter estruturas tridimensionais similares. Isto ocorre porque mesmo que os aminoácidos que compõem as proteínas sejam diferentes, eles possuem características similares. Desta forma, o que define a função de uma proteína, não é a sua estrutura primária, mas sim sua estrutura tridimensional.

```
001 mvlspadktn vkaawgkvgg hageygaeal ermflsfptt ktyfphfdls hgmaqvkghg
061 kkvadaltna vahvddmpna lsalsdlhah klrvdvvnfk llshcllvtl aahlpaeftp
121 avhasldkfl asvstvltsk yr
```

Fonte: Genbank (2006).

Quadro 2.2 – Estrutura primária da Hemoglobina



Fonte: Park et al. (2006).

Figura 2.2 – Estrutura terciária da hemoglobina

2.3 GENÉTICA MOLECULAR

A genética molecular trata de como as informações hereditárias são transmitidas dos seres vivos aos seus descendentes. Como dito na Seção 2.1, as características próprias de cada espécie são hereditárias e estão armazenadas no seu genoma. A importância do genoma está no fato de que, ao mesmo tempo, ele assegura que as informações genéticas serão transmitidas de geração em geração, como também fornece um meio para que ocorram mutações e com isto surgimento de novas características.

O surgimento de novas características nas espécies é causado pelas mutações. Existem diversos tipos de mutações, algumas apenas removem, modificam ou adicionam um nucleotídeo, outras podem remover ou duplicar parte inteira de uma seqüência e outras modificar parte da seqüência de posição ou invertê-la. Os resultados das mutações podem ser os mais diversos, desde mutações que não afetam o aminoácido que forma a proteína até a duplicação ou remoção do gene que é responsável por determinada proteína.

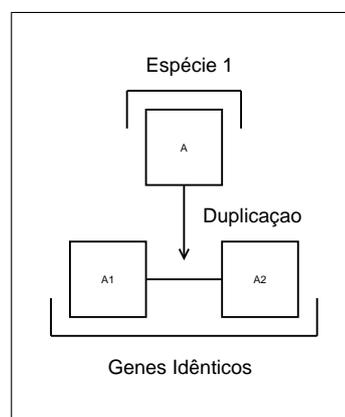
Estas mudanças que ocorrem no genoma dos organismos são as responsáveis pela

sua evolução. Desta forma, remete-se aos conceitos de evolução das espécies publicados pela primeira vez em 1859 no livro “A Origem das Espécies” de Charles Darwin. A Teoria da Evolução, proposta por Charles Darwin, diz que os organismos sofrem mutações entre diferentes gerações e as modificações vantajosas são perpetuadas, enquanto desvantajosas são eliminadas pela seleção natural.

Alberts et al. (2004) afirmam que todas as células executam as operações básicas: síntese de proteínas, hereditariedade, produção de energia, da mesma forma. Somando-se isto com a análise dos genes que executam tais operações, pode-se afirmar que todos os seres vivos originam-se de um único ancestral comum. A informação para estabelecer quais são os parentes mais próximos de cada espécie e sustentar quem poderia ser último ancestral comum de cada grupo de espécies está contida no seu genoma.

Todos os seres vivos e conseqüentemente suas células, desenvolveram-se a partir de um único ancestral. Desta forma, as espécies compartilham semelhanças no genoma. Essas semelhanças traduzem-se em genes e proteínas homólogos (BERNARDES, 2004, pg. 16).

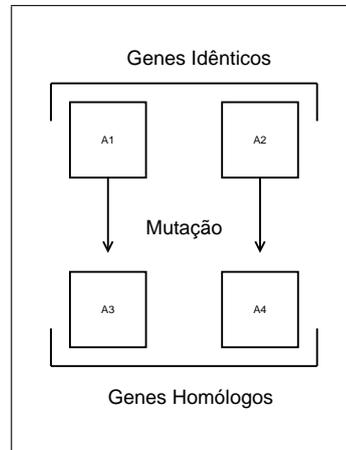
Conforme detalhado na Seção 2.1, que aborda a biologia molecular, os genes contêm as informações para a síntese de proteínas. Durante a existência dos seres vivos, os genes sofrem processos evolutivos, como a duplicação gênica e as mutações. A duplicação gênica é o processo em que a seqüência de um gene é copiada uma ou mais vezes no genoma. A Figura 2.3 exhibe o efeito da duplicação gênica num pseudo-gene A. Como resultado da duplicação, teremos cópias do mesmo gene que seguirão caminhos evolutivos independentes. Nos caminhos evolutivos, os genes sofrem mutações e ao longo do tempo, o conteúdo dos genes se distanciarão do ancestral. Estes genes que têm um ancestral comum, são chamados de genes homólogos.



Fonte: Bernardes (2004).

Figura 2.3 – Duplicação gênica

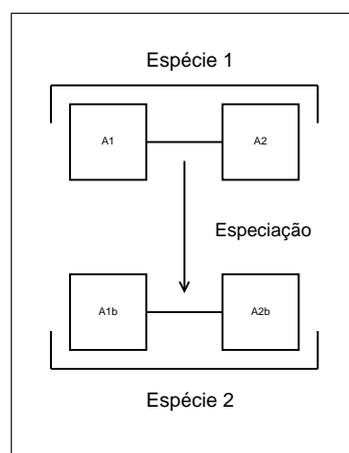
A Figura 2.4 exibe o efeito da mutação gênica nos genes resultantes da duplicação gênica da Figura 2.3. Quando uma espécie sofre o processo de especiação e dá origem a uma nova espécie, os genes homólogos da mesma espécie são chamados de parálogos.



Fonte: Bernardes (2004).

Figura 2.4 – Genes homólogos

A Figura 2.5 mostra o resultado da especiação, em que os genes *A1* e *A2* estarão presentes tanto na espécie 1 como na espécie 2. Desta forma, os genes homólogos na mesma espécie são chamados de parálogos.

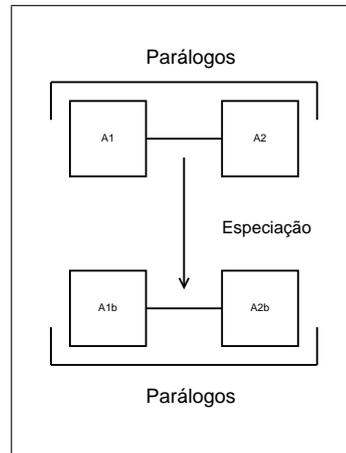


Fonte: Bernardes (2004).

Figura 2.5 – Resultado da especiação nos genes

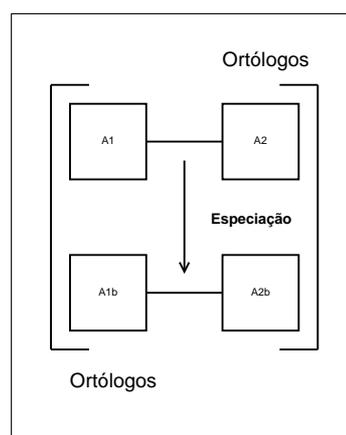
A Figura 2.6 demonstra a relação entre os genes *A1* e *A2* com os genes resultados da especiação, *A1b* e *A2b*. Os genes homólogos em espécies diferentes são chamados de ortólogos. A Figura 2.7 exibe a relação ortológica entre os genes *A1* e *A1b* e os genes *A2* e *A2b*.

De uma forma mais direta, genes homólogos são os genes que possuem um ancestral comum, genes parálogos são genes homólogos na mesma espécie e genes ortólogos são genes



Fonte: Bernardes (2004).

Figura 2.6 – Genes parálogos



Fonte: Bernardes (2004).

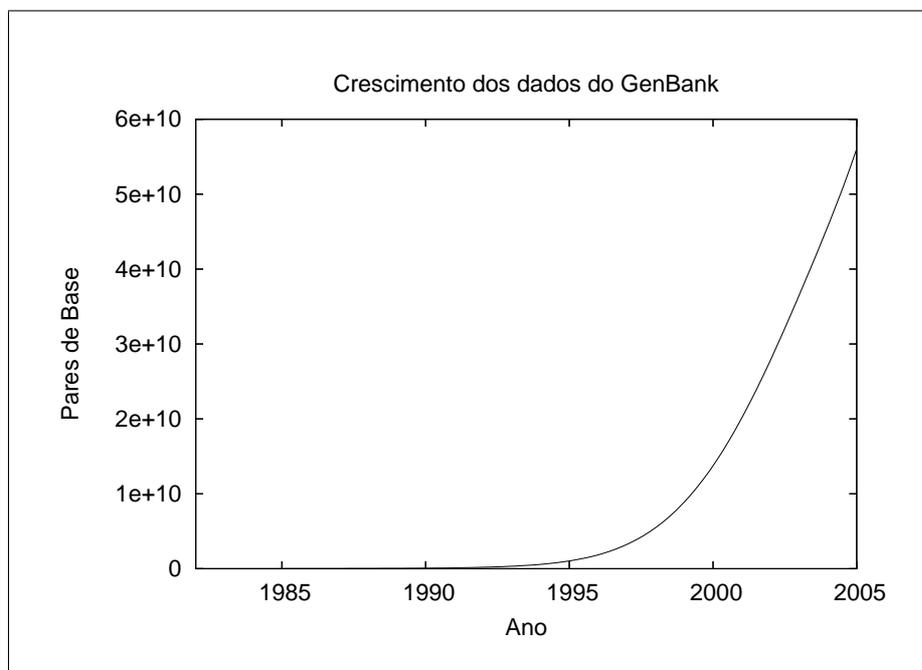
Figura 2.7 – Genes ortólogos

homólogos em espécies diferentes. As seqüências homólogas são classificadas em famílias, onde cada família tem a uma ou mais funções definidas e as seqüências que fazem parte delas possuem um alto grau de similaridade.

Outra questão importante a respeito de homologia: duas seqüências podem ser homólogas ou não, não podendo afirmar quão homólogas duas seqüências são. Quando trata-se de similaridade entre seqüências, pode-se definir um grau similaridade entre as seqüências, podendo variar numericamente de nada similar a muito similar.

2.4 BIOINFORMÁTICA

A comparação e análise de seqüências genéticas é parte de um campo científico chamado bioinformática. Este campo científico é baseado numa interação entre a estatística, biologia molecular e a ciência da computação (ALBRECHT, 2005, p. 139). Com os projetos de seqüenciamento de diversos genomas, entre eles o humano, a quantidade de informações nos bancos de dados de seqüência cresce exponencialmente. Como exemplo do crescimento da quantidade de dados, pode-se verificar na Figura 2.8 o crescimento das seqüências de DNA disponíveis no *GenBank* (WHEELER et al., 2004). O *GenBank* é um banco de dados para seqüências nucleicas e proteínas e informações a respeito delas. Todas as seqüências e informações armazenadas neste banco são disponibilizadas livremente.



Fonte: Genbank (2006).

Figura 2.8 – Crescimento da quantidade de dados de seqüência no *GenBank*

Sobre o acúmulo de dados genéticos, os métodos tradicionais de laboratório não são capazes de acompanhar a taxa de crescimento de novas informações e como conseqüência,

biólogos moleculares passaram a utilizar métodos estatísticos e computacionais capazes de analisar estas grandes quantidades de dados de forma mais automatizada (ALBRECHT, 2005, p. 139). Isto ocorre porque o crescimento da quantidade de novas seqüências disponíveis extrapola a capacidade de análise *in vitro* destas novas seqüências.

Entre as principais tarefas da bioinformática, pode-se citar: o armazenamento de seqüências, tanto genéticas, quanto protéicas, a pesquisa ou comparação destas seqüências, o alinhamento múltiplo de seqüências, predição de genes e a inferência filogenética.

O armazenamento de seqüências genéticas, tem como objetivo proporcionar meios de pesquisas às seqüências já disponíveis e informações sobre elas. Por exemplo, a seqüência apresentada no Quadro 2.2, pode ser acessado pelo banco de seqüências Genbank (2006) através do identificador *gi* : 4504347. Outra vantagem do armazenamento de seqüências, é a possibilidade de buscas de seqüências similares.

A busca por seqüências similares é feita através da comparação das seqüências para verificar quantitativamente a similaridade entre duas seqüências. Existem diversas formas de comparar seqüências genéticas, como a utilização de programação dinâmica, o uso de heurísticas como o BLAST(ALTSCHUL et al., 1995) e a utilização de modelos probabilísticos, como os modelos ocultos de markov (ALBRECHT, 2005, p. 145).

Um dos objetivos da busca por seqüências similares é poder agrupar seqüências homólogas e classificá-la numa família. Esta tarefa da bioinformática é importante porque os testes em laboratório para classificar uma proteína numa família são caros e demorados (BERNARDES, 2004).

Agrupando os genes de uma família, pode-se compará-los utilizando o alinhamento múltiplo. Mount (2004, pg. 164) diz que através de alinhamentos simultâneos de seqüências de genes é possível analisar os padrões de mudanças nas seqüências. Outra utilidade do alinhamento múltiplo é alinhar diversas seqüências para que seja possível fazer uma inferência filogenética delas.

3 SISTEMAS E ALGORITMOS DISTRIBUÍDOS

Sistemas distribuídos são sistemas compostos por mais de um processador, onde a computação do processo é distribuído entre eles. Um modelo de sistemas distribuídos são os *clusters*. Neles, as tarefas de processamento de dados são divididas entre diversos nós, sendo cada nó um computador independente.

Um modelo de *cluster* que está sendo intensamente utilizado é o *beowulf*. A computação paralela num *clusters beowulf* é realizada dividindo-se o problema computacional em partes, fazendo uso de múltiplos processos e atribuindo a cada um dos processos uma parte do problema. Outro modelo existente, o *Symmetric Multiprocessing* (SMP) é baseado na utilização de diversos processadores que estão conectados a uma memória em comum e boa parte dos multiprocessadores atuais.

Uma vantagem dos *clusters beowulf* sobre os modelos SMP, é o custo. Enquanto sistemas que são baseados em SMP necessitam de hardware especial e com custo mais elevado, os *clusters beowulf* podem ser construídos através de hardware comumente encontrada no mercado. Pode-se construir *clusters beowulf* utilizando computadores *desktop* interconectados através de uma conexão *ethernet*.

Para comunicação entre os processos do *cluster beowulf* é utilizada a técnica de passagem de mensagens. Cada processo possui sua memória e os dados que são necessários compartilhar com os demais processos, são enviados através de mensagens. Para efetuar o trabalho de troca de mensagens, um grupo de fornecedores de computadores paralelos especificou o padrão MPI.

3.1 ALGORITMOS DISTRIBUÍDOS

Segundo Lynch (1997, p. 2), algoritmos distribuídos são algoritmos desenvolvidos para serem executados em vários processadores interconectados. Lynch (1997, p. 2) complementa que partes de um algoritmo distribuído executam concorrentemente e independentemente, cada qual apenas com uma quantidade limitada de informações.

Sobre as principais características de algoritmos distribuídos, Lynch (1997, p. 3) cita a *Intercomunicação entre Processos* (IPC), o modelo de tempo, o modelo de controle de falhas e os principais problemas nas quais se encontrará e se pretende resolver.

Sobre o IPC, pode-se ter comunicação entre os processos através de memória compartilhada, quando estes fizerem parte de um modelo SMP ou pode-se utilizar troca de

mensagens ponto a ponto ou *broadcast*, ambas através de redes de comunicação locais ou a de longa distância, ou a comunicação entre os processos também pode ser feita através de *Chamada de Procedimento Remoto* (RPC).

No modelo de tempo, vários modelos podem ser utilizados. Num extremo estão os algoritmos síncronos, onde todas as operações são sincronizadas em todos os processos. No outro extremo, estão os algoritmos assíncronos, onde os passos de execução de cada processo são executados independente dos outros processos e sem nenhum modo de sincronismo. No meio termo, há os algoritmos parcialmente síncronos, onde os processos possuem informações parciais sobre os eventos de sincronização. Deste modo, os processos são executados livremente, porém em alguns momentos há pausas para troca de informações e sincronização.

Nos modelos de falha, há alguns algoritmos que possuem tolerância caso ocorra alguma falha na execução e meios para continuar a execução ou evitar que todo o grupo de processos tenha que ser finalizado. As falhas podem ser: problemas na comunicação entre os processos, falha no próprio algoritmo ou implementação deste ou no hardware ou software que este está sendo executado. Por fim, alguns algoritmos são preparados para tolerar certo limite de falhas de execução e assim continuar o processamento.

Os demais problemas, são problemas como o de detecção de *deadlocks*, problemas de concorrência, de alocação de recursos, visão global atual do grupo, consenso entre os diferentes processos e implementação e distribuição de tipos variados de objetos. Estes problemas estão presentes em diversos algoritmos distribuídos e estes algoritmos estão presentes em diversos lugares, por exemplo nos banco de dados, no controle de nomes da internet, no controle de saques e depósitos dos bancos, num grupo de processos que pesquisam homologais entre seqüências genéticas e na redes de distribuição de dados *per-to-per*.

3.1.1 Modelo de rede de comunicação síncrona

Nesta subseção, é dada uma introdução sobre um modelo de comunicação que é largamente utilizado entre os projetos de algoritmos distribuídos. Este modelo tem a vantagem de ser de fácil compreensão, especificação e para análise dos algoritmos que o otimizam.

Segundo Lynch (1997, p. 17), sistema de rede de comunicação síncrona consiste numa coleção de elementos computacionais localizados nos nós de um grafo dirigido $G = (V, E)$. Sendo V os nós na rede e E a comunicação entre eles. Associado a cada nó i incluso em V , tem-se um processo, que consiste formalmente nos seguintes componentes:

- a) $estados_i$, um (não necessariamente finito) conjunto de estados;
- b) $inicio_i$, o conjunto não vazio de $estados_i$, conhecidos como o início dos estados ou estado inicial;
- c) $msgs_i$, uma função geradora de mensagem, mapeada como $estados_i \times vizinhos_out_i$;
- d) $trans_i$, uma função de transição de estados mapeando $estados_i$ e vetores (indexada por $vizinhos_in_i$).

Cada processo possui um conjunto de estados, sendo possível separar os estados iniciais de cada processo. O conjunto de estados não precisa ser necessariamente finito e esta generalidade é importante, já que permite um modelo de sistema que incluem estrutura de dados sem limite, como contadores. A função geradora de mensagens especifica o próximo estado e qual vizinho que receberá a mensagem que é enviada partindo do estado dado. E a função de transição de estados, mapeia para o estado atual e cada mensagem recebida de determinado vizinho, qual será o próximo estado do processo (LYNCH, 1997, p. 18).

Resumindo, no modelo de redes de comunicação síncrona, o estado de cada processo depende do seu estado atual e das mensagens recebidas de seus vizinhos. Apenas haverá mudanças do estado, quando receber uma mensagem. As mensagens enviadas também dependem do estado atual e do vizinho que a receberá.

As falhas podem ser tanto na execução de um processo, quanto de comunicação, falha de canal. Um processo pode enviar a outros processos mensagens alertando uma falha ocorrida no seu processamento ou enviar a próxima mensagem normalmente, ignorando o erro ocorrido e continuando o processo.

A execução no modelo de redes de comunicação síncrona define n canais de comunicação, podendo ser definidos como uma seqüência infinita $C_0, M_1, N_1, C_1, M_2, N_2, C_2, \dots$, aonde cada C_r é um estado e cada M_r e N_r é uma mensagem, sendo C_r representa o estado após r rodadas e M_r o remetente e o N_r o destinatário da mensagem.

Nos algoritmos não distribuídos, a análise de eficiência dos algoritmos se dá através do tempo de execução e do consumo de memória. Segundo Cormen et al. (2001, p. 23-25) pode-se analisar um algoritmo através do pior caso, melhor caso e caso médio em termos de tempo de execução e de consumo de memória através das entradas dadas ao algoritmos. Para os algoritmos distribuídos, Lynch (1997, p. 21) cita a análise pela complexidade de tempo e complexidade de comunicação. Onde a complexidade de tempo é o número de rodadas necessárias para que seja a computação seja concluída e a complexidade de comunicação é a quantidade de mensagens não vazias que são necessárias para a conclusão

da computação do algoritmo. Lynch (1997, p. 22) complementa dizendo que a medida de tempo é a medida mais importante na prática, não apenas para algoritmos distribuídos síncronos, mas para todos os algoritmos distribuídos.

A complexidade de comunicação é importante para evitar casos de congestionamento, mas o impacto da carga de comunicação de um algoritmo numa complexidade de tempo não é simplesmente uma função de um algoritmo distribuído individual. Pois o tempo de execução de um algoritmo distribuído depende de outros fatores, tais como o tipo de comunicação entre os processos participantes e a carga deste meio de comunicação no momento de execução, por exemplo, outros algoritmos sendo executados no momento, pois numa rede de comunicação típica, diversos algoritmos distribuídos são executados simultaneamente. Lynch (1997, p. 22) finaliza dizendo que devido a dificuldade de quantificar o impacto que a mensagens de qualquer algoritmo tem na desempenho de tempo na execução de outros algoritmos, deve-se analisar e tentar minimizar a quantidade de mensagens geradas pelos algoritmos individualmente.

3.2 PADRÃO MPI

O MPI não é uma implementação específica, mas um padrão para ser seguido na implementação de bibliotecas de troca de mensagens para computação paralela. Para o desenvolvedor é transparente qual implementação do MPI será utilizada, pois ele desenvolverá utilizando as interfaces especificadas por ele. Desta forma, pode-se abstrair completamente qual implementação do MPI será utilizada no momento de execução. Entre as principais implementações, podem ser citados os seguintes softwares: *MPICH* (MPICH2..., 2006), o *LAM* (BURNS; DAOUD; VAIGL, 1994) e o *OpenMPI* (GABRIEL et al., 2004). Estes softwares fornecem implementação do MPI às linguagens *C*, *C++* e *FORTRAN*.

O padrão MPI define diversas operações, sendo as principais: informações sobre o número de processos participantes do *cluster* e identificação do nó em tempo de execução, envio e recepção de mensagens, definição de tipos e entrada e saída paralela, acesso a memória remota e gerência de sub-grupos de processamento. Mesmo o MPI possibilitando diversas operações, Sterling (2002, p. 209) diz que conhecendo apenas as operações básicas citadas, é possível implementar softwares distribuídos utilizando o padrão MPI.

A seguir é apresentada uma introdução, que utiliza informações de Sloan (2004), sobre as principais funcionalidades do MPI e as que são mais relevantes à compreensão do trabalho.

3.2.1 Envio e recepção de mensagens

A tarefa fundamental de sistemas distribuídos baseados em passagem de mensagens é o envio e recepção de mensagens. Para tal operação, o padrão MPI especifica diversas operações, sendo *MPI_Send* e *MPI_Recv* as operações fundamentais. Ambas operações possuem parâmetros para informar qual dado enviar, a quantidade, o tipo dele, o destinatário ou remetente, um identificador para a mensagem e o grupo de processos. O *MPI_Recv* possui um parâmetro a mais que contém informações sobre o estado dos dados recebidos.

O *MPI_Send* e o *MPI_Recv* são bloqueantes, ou seja, o próximo comando será executado somente quando a mensagem for totalmente enviada ou recebida. Eles funcionam de forma síncrona, sendo que o processo só continua após o envio e recepção da mensagem. A Figura 3.1 exibe o tempo de espera na utilização do *MPI_Send* e *MPI_Recv*. Observa-se duas linhas de processamento, sendo que o processo *A* envia para o processo *B* uma mensagem. Antes do processo *A* enviar a mensagem, o processo *B* já esperava por ela e estava bloqueando todo o seu fluxo de processamento. O processo *B* também bloqueia o seu fluxo de processamento enquanto aguarda a mensagem ser totalmente enviada. Após a mensagem ser recebida pelo processo *B*, ele executa algumas operações utilizando os dados recebidos nela e envia o retorno ao processo *A*. Porém, o processo *A* estava aguardando um retorno antes do envio, e novamente nota-se interrupção do processamento.

As interrupções no processamento ocasionam desperdício do processador, pois neste tempo ele está voltado ao aguardo ou envio da mensagem. Para resolver este problema, o MPI especifica funções para o envio e recepção de mensagens de forma não bloqueante. Para o *MPI_Send* e *MPI_Recv*, as funções correlatas não bloqueantes são o *MPI_Isend* e o *MPI_Irecv*. O *MPI_Isend* e *MPI_Irecv* trabalham de forma assíncrona, não sendo necessário total envio ou recepção para que o processo prossiga. Os parâmetros destas duas funções são os mesmos das suas correlatas bloqueantes, com adição de um novo parâmetro de estado com informações a respeito da mensagem, por exemplo se ela já foi enviada ou recebida.

O funcionamento das funções não bloqueantes é demonstrado na Figura 3.2. Percebe-se que o processo é bloqueado somente na hora de executar a função, não sendo bloqueado durante o envio e recepção das mensagens. Quando a mensagem é completamente enviada ou recebida, o parâmetro de estado utilizado para executar a função modifica um dos seus atributos informando assim que a função foi totalmente executada.

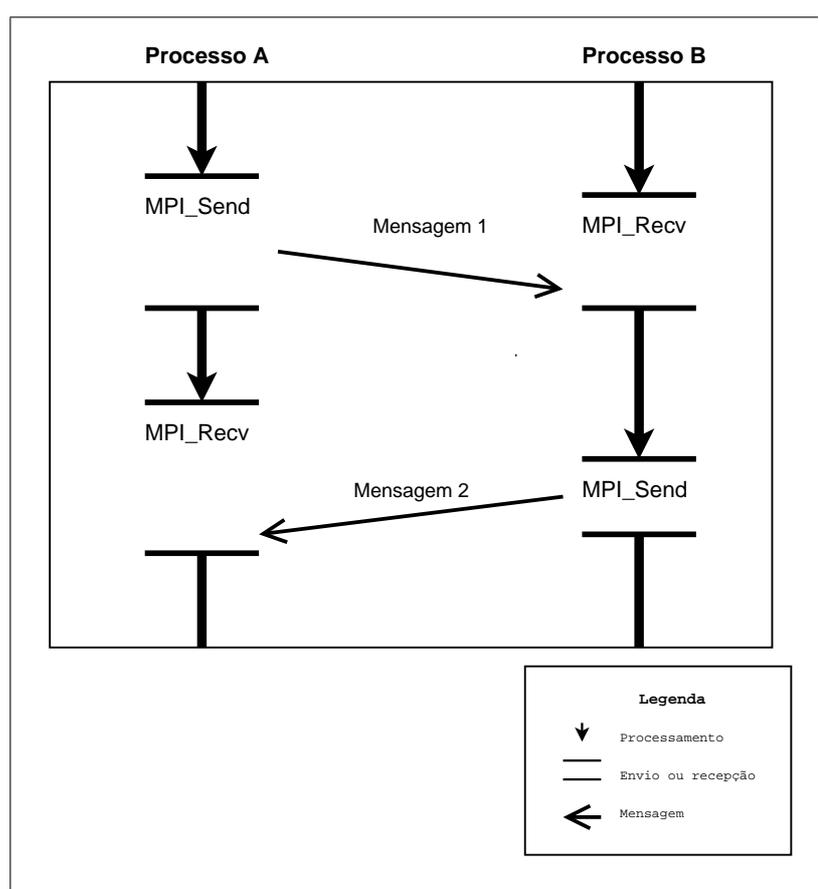


Figura 3.1 – Funcionamento do MPI_Send e do MPI_Recv

As funções apresentadas são para a comunicação ponto a ponto entre dois nós. Por exemplo, caso deseje-se mandar um conjunto de dados do processo *A* para os processos *B* e *C*, com o *MPI_Send* ou *MPI_Isend* e o *MPI_Recv* e o *MPI_Irecv*, serão necessárias duas mensagens. A Figura 3.3 exibe como é o envio destes dados utilizando comunicação ponto a ponto.

Para resolver este tipo de problema e quando é necessário sincronizar determinado dado em todos os nós, o MPI especifica uma função que envia os dados para todos os nós, como um *broadcast*. Esta função, *MPI_Bcast*, recebe como parâmetros a posição da informação que será enviada ou recebida, a quantidade, o tipo dele, o identificador do processo que enviará os dados e o grupo. A utilidade do *MPI_Bcast* é encapsular em uma função o envio de mensagem a todos os nós e neles o processo de recepção desta mensagem. A Figura 3.4 exibe como é o envio de dados a diversos nós utilizando o *MPI_Bcast*.

É importante salientar que a função *MPI_Bcast* encapsula o envio de mensagens a diversos destinatários, porém, ela necessariamente não funciona conforme o *broadcast* do protocolo *TCP-IP*, enviando apenas uma mensagem e todos os nós a receberão. O seu funcionamento depende da implementação do MPI utilizado, qual o tipo de comunicação e interconexão dos processos, entre outros.

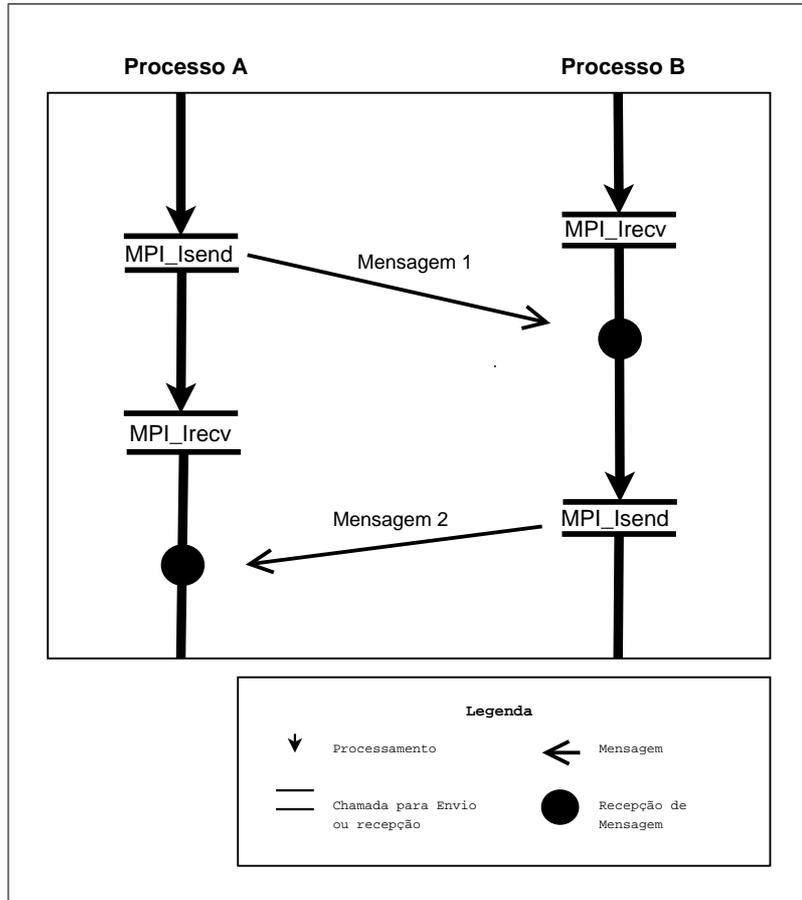


Figura 3.2 – Funcionamento do MPI_Isend e do MPI_Irecv

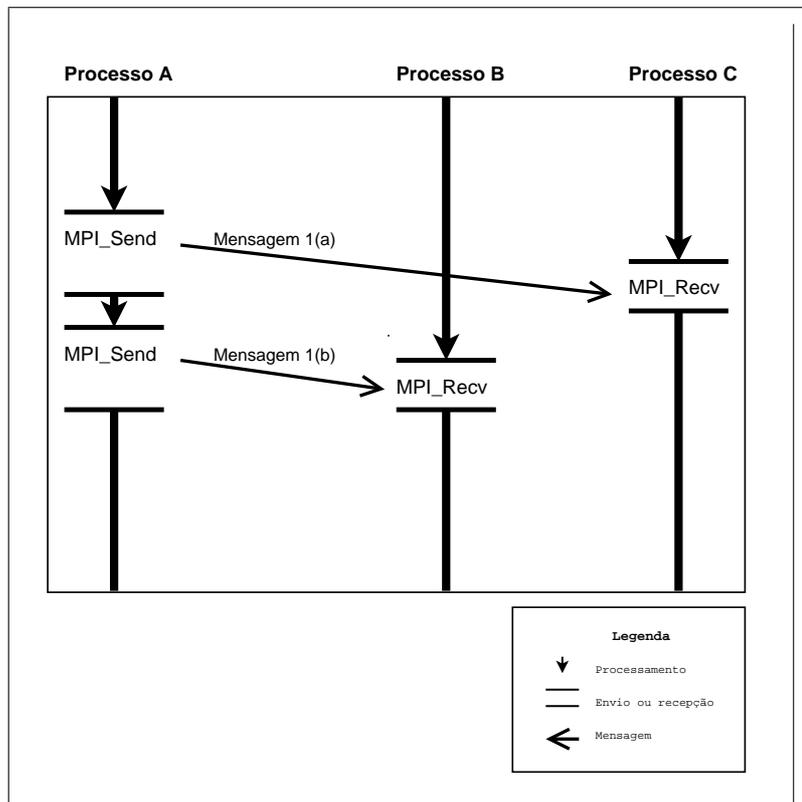


Figura 3.3 – Envio de dados a diversos nós utilizando ponto a ponto

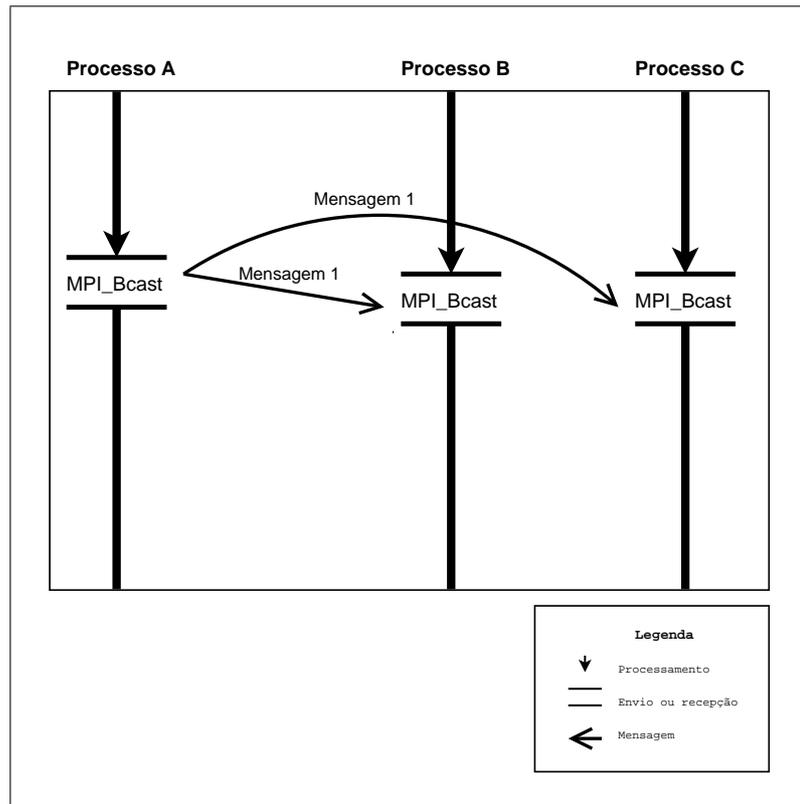


Figura 3.4 – Envio de dados a diversos nós utilizando *broadcast*

```

1  #define STRING_LEN 256
2
3  typedef struct __pessoa pessoa_t;
4  struct __pessoa {
5      char nome[STRING_LEN];
6      unsigned int idade;
7      float salario;
8  };

```

Figura 3.5 – Criação de novo tipo na linguagem de programação *C*

3.2.2 Definição de novos tipos de dados

Nesta seção é apresentado a definição de novos tipos de dados para o MPI. Na linguagem de programação *C* um novo tipo de dado utilizando *typedef* e definir uma estrutura utilizando *struct*. A Figura 3.5 exhibe a criação do tipo *pessoa.t*.

O tipo *pessoa.t* poderá ser utilizado pela linguagem de programação *C*, porém a linguagem não tem como descrevê-la às funções do MPI. Isto ocorre porque a linguagem *C* não possui meios de ler meta-estruturas e as utilizar, como é possível na linguagem de programação *Java* através do uso das classes do pacote *java.lang.reflect*.

Para que o MPI tome conhecimento dos novos tipos e como eles são compostos e possa trabalhar com estes tipos, torna-se necessário descrever estes tipos a ele. Para

```

1  MPI_Datatype pessoa_mpi;
2  MPI_Datatype types[3] = {MPI_CHAR, MPI_UNSIGNED, MPI_REAL};
3  int blocklengths[3] = {1, STRING_LEN, 1};
4  MPI_Aint displacements[3] = {0, sizeof(int),
                               sizeof(int) + sizeof(char) * STRING_LEN};
5
6  MPI_Type_create_struct(3, blocklengths, displacements,
                        types, &pessoa_mpi);
7  MPI_Type_commit(&pessoa_mpi);

```

Figura 3.6 – Descrição de uma estrutura para o *MPI*

```

1  MPI_Datatype pessoa_vector_mpi;
2
3  MPI_Type_contiguous(10, pessoa_mpi, &pessoa_vector_mpi);
4  MPI_Type_commit(&pessoa_vector_mpi);

```

Figura 3.7 – Criação de um vetor em *MPI*

descrição de um tipo estrutura, o *MPI* define a função *MPI_Type_create_struct* que tem como objetivo definir uma nova estrutura que poderá ser utilizada. Os parâmetros desta função são: o número de atributos que a estrutura terá, a quantidade de elementos em cada um dos atributos, o início de cada atributo, os tipos dos atributos e a variável que conterà esta nova estrutura. A Figura 3.6 exibe a descrição do tipo *pessoa_t* para o *MPI*. A linha 1 define a variável *pessoa_mpi* que é um meta-tipo para o *MPI*. A segunda linha informa os 3 tipos que formam a estrutura e a linha abaixo informa a quantidade deles. A quarta linha define a posição de cada elemento na memória e a linha 6 cria o tipo *pessoa_t*. A sétima linha transmite a todos os nós o novo tipo e habilita-o a ser utilizado.

Caso deseje-se criar um vetor de determinado tipo, o *MPI* define a função *MPI_Type_contiguous*. A Figura 3.7 exibe a criação de um vetor *pessoa_mpi* de 10 posições. A primeira linha define uma variável que conterà o meta-tipo, a seguir define-se que o tipo *pessoa_vector_mpi* será um vetor de 10 posições e por fim ele transmite o novo tipo e habilita-o a ser utilizado.

Suponha-se que um nó que criará o vetor com 10 elementos *pessoa_t* definidos na Figura 3.5 e deseja enviá-lo a todos os demais nós. O primeiro passo é definir o tipo *pessoa_t* para o *MPI* como exibido na Figura 3.6 e após um vetor deste tipo, como demonstrado na Figura 3.7. Então deve-se criar um vetor de 10 posições e popula-lo, como na Figura 3.8. Da primeira a quinta linha, são declaradas as variáveis que serão utilizadas. A sétima linha, obtém o identificador do processo. O *MPI_COMM_WORLD* informação que é em relação todos os processos desta execução do *cluster*. Ele é normalmente utilizado quando não são feitos agrupamentos de processos em sub-grupos. Os dados são

```
1  #define PROCESS_ROOT 0
2
3  int process_id;
4  pessoa_t pessoas[10];
5  int i;
6
7  MPI_Comm_rank(MPI_COMM_WORLD, &process_id);
8
9  if (process_id == PROCESS_ROOT) {
10     for (i = 0; i < 10; i++) {
11         pessoas[i].nome    = ler_nome();
12         pessoas[i].idade   = ler_idade();
13         pessoas[i].salario = ler_salario();
14     }
15 }
16
17 MPI_Bcast(pessoas, 1, pessoa_vector_mpi, PROCESS_ROOT, MPI_COMM_WORLD);
```

Figura 3.8 – Populando um vetor de *pessoa_t*

populados da nona a décima quinta linha, porém este código somente é executado caso seja o *PROCESS_ROOT* ou seja, o processo 0. Na linha 17 o *PROCESS_ROOT* envia a todos os demais nós o vetor *pessoas*.

4 FILOGENÉTICA

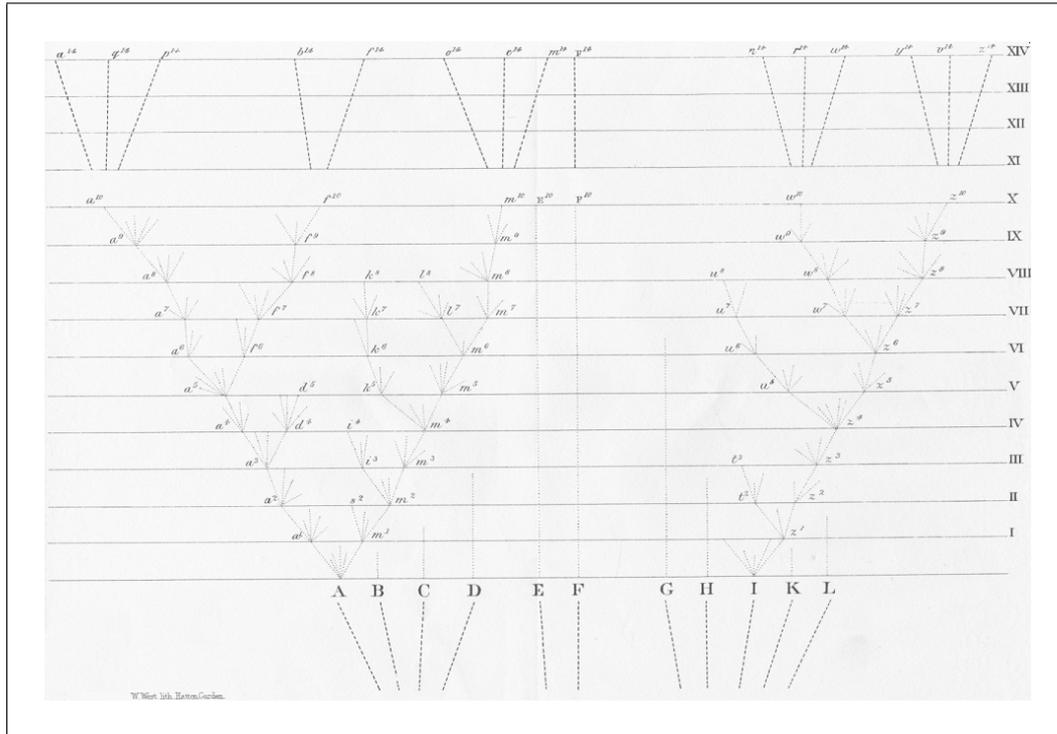
Neste capítulo, apresenta-se a filogenética e seus principais métodos. Posteriormente é apresentado mais profundamente a análise filogenética cladística e a análise filogenética utilizando matrizes de distâncias.

4.1 HISTÓRIA DA FILOGENÉTICA

Desde o início da história, a humanidade tenta compreender a origem da vida. Sarkar (2006, p. 3) cita o filósofo Aristóteles como o primeiro a registrar a vida no planeta Terra de forma categorizada. Sarkar (2006, p. 3) menciona citando Carl Linnaeus e Jean-Baptiste LaMarck como sendo cientistas que também organizaram os organismos. Linnaeus propôs a forma de dois nomes para referir-se a determinada espécie, como nos casos dos humanos: *Homo sapiens*. Porém, Aristóteles, Linnaeus e LaMarck apenas organizaram os organismos em grupos, enquanto que com a publicação da Origem das Espécies (DARWIN, 1859) a organização dos seres vivos passou a ter uma base evolucionária.

A Teoria da evolução de Darwin diz que as características observáveis dos seres vivos são selecionadas através de um processo de evolução. Neste processo, os seres melhores adaptados, ou seja, que possuem as melhores características àquele ambiente, perpetuam descendentes, enquanto os mal adaptados são eliminados. As características dos seres vivos hoje, são herdadas dos seu ancestrais e em algum ponto da história elas se mostraram vantajosas. A Origem das Espécies de Darwin (1859) representa espécies como ramificações no processo de evolução. Cada ramo, representando um espécie, surgiu de outro ramo, que por sua vez era outra espécie. A Figura 4.1, apresenta um diagrama desenhado por Charles Darwin que exhibe as ramificações entre as espécies e conseqüentemente surgimento de novas espécies.

Através de análises morfológicas, é possível traçar linhas evolutivas entre espécies. Sarkar (2006, p. 3) cita como exemplo os dinossauros e as aves, onde analisando o tamanho do fêmur e sua orientação é possível verificar que as aves originaram-se dos dinossauros. Outro exemplo, é entre o homem e as espécies de macacos, que compartilham muitas semelhanças. Através de análise de fósseis, é possível verificar possíveis ancestrais comuns. Sarkar (2006, p. 3), diz: o estudo e desenvolvimento destes métodos no contexto da história evolucionária, representando relações hierárquicas, como Darwin demonstrou na Figura 4.1, ficou conhecido como filogenética. O trabalho de Darwin tenta definir como os organismos (como espécies) surgem em relação a cada outro e suas relações evoluti-



Fonte: Wyhe (2006).

Figura 4.1 – Diagrama de evolução de espécies por Charles Darwin

vas. Sarkar (2006, p. 3) acrescenta, dizendo que assim como as técnicas filogenéticas descrevem a evolução de uma maneira sistemática, a filogenética é também referida como sistemáticas.

Com o surgimento da era genômica, o estímulo do Projeto do Genoma Humano (COLLINS et al., 1998) e o seqüenciamento do genoma de outras espécies, com o acúmulo de informações sobre o DNA, produziram um crescimento exponencial na quantidade de dados em repositórios como o *GenBank* (WHEELER et al., 2004). Técnicas filogenéticas foram adaptadas para tratar com informações genéticas, pois as primeiras técnicas apenas trabalhavam com informações morfológicas e não com informações moleculares.

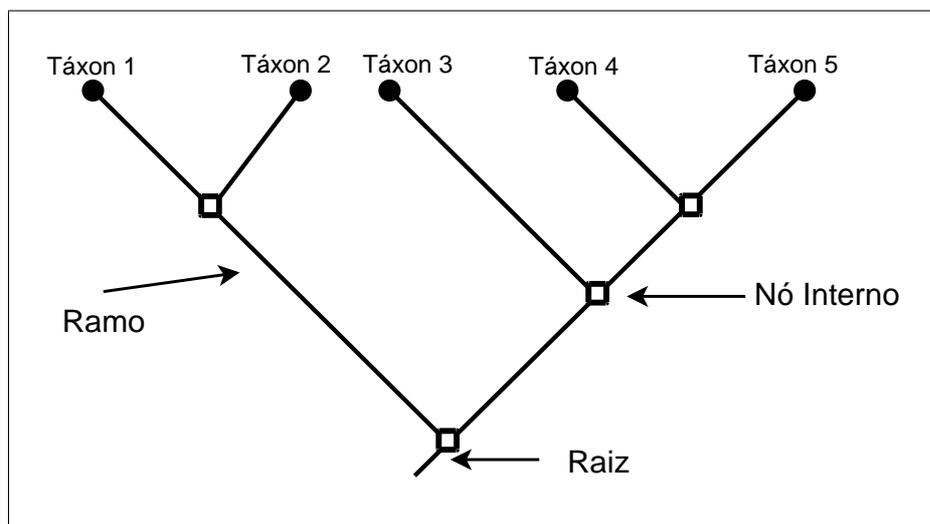
Uma das maiores contribuições na análise de seqüências genéticas e protéicas é a possibilidade de inferir relações entre as seqüências analisadas. A inferência é possível, porque seqüências homólogas possuem similaridades, e seqüências com maior similaridade normalmente estão mais próximas evolutivamente do que seqüências com alta divergência. O objetivo da análise filogenética de seqüências de ácidos nucléicos e de proteínas é observar a relação entre vários grupos de seqüências que podem ser alinhados.

As relações filogenéticas entre os genes ou proteínas pode ajudar a prever as funções destes, pois genes ou proteínas homólogas possuem características similares. Utilizando a filogenética, pode-se agrupar seqüências em grupos e desta forma inferir a função

destes.

4.1.1 Representações de Filogenias

As relações filogenéticas são normalmente apresentadas na forma de árvores. A Figura 4.2 apresenta uma árvore exibindo as relações filogenéticas de cinco táxons¹ e algumas características presentes em todas as árvores filogenéticas. Nas folhas da árvores, estão os táxons que são analisados quanto a sua homologia. Os táxons são ligados através dos ramos e entre os ramos há os nós. Os nós representam o surgimento de um novo táxon a partir de um já existente. Por exemplo, se for uma árvore filogenética exibindo espécies, os nós representariam o evento de especialização. A árvore também apresenta uma raiz, que representa o táxon hipotético ascendente de todos os taxóns exibidos.



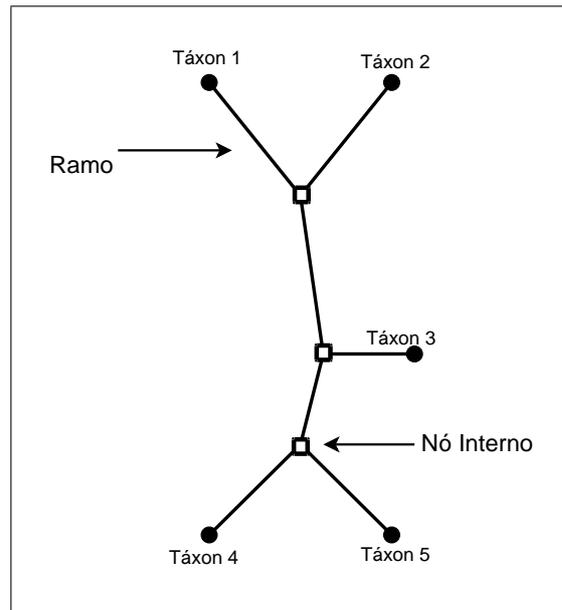
Fonte: Felsenstein (2003).

Figura 4.2 – Uma árvore representando filogenia entre cinco táxons

Há um outro tipo de árvore filogenética, a árvore sem raiz, que é utilizado para representar as relações entre os táxons quando não se conhece ou não é necessário representar o táxon originário. As árvores filogenéticas sem raiz apresentam todas as características das árvores com raiz, com exceção da raiz originária. A Figura 4.3 apresenta uma árvore filogenética sem raiz.

As árvores filogenéticas também podem apresentar distâncias entre os táxon ou comprimento dos ramos. Este comprimento não possui uma medida definida, por exemplo os anos que separam o surgimento do táxon, é uma forma escalar de representar a distância evolutiva que separam os táxon e nós a partir dos dados utilizados para a construção

¹Táxon é uma unidade associada a um sistema de classificação. Táxons (ou taxa) podem estar em qualquer nível de um sistema de classificação podendo ser um reino, um gênero, uma espécie ou qualquer outra unidade de um sistema de classificação dos seres vivos.



Fonte: Felsenstein (2003).

Figura 4.3 – Uma árvore sem raiz representando filogenia entre cinco táxons

da árvore. Estas distâncias representam o tempo evolutivo entre os táxons e os seus surgimentos. A Figura 4.4 apresenta uma árvore sem raiz com o tamanho dos ramos.

Como exemplo, na Figura 4.4 os táxons 1 e 2 possuem a distância de 1, pois esta é a soma dos dois ramos que separam estes dois táxons. Já os táxons 1 e 4 tem a distância 3,35, já que esta é a soma dos 4 ramos que os separam. Desta forma, pode-se concluir que os táxons 1 e 2 são mais próximos evolutivamente que os táxons 1 e 4.

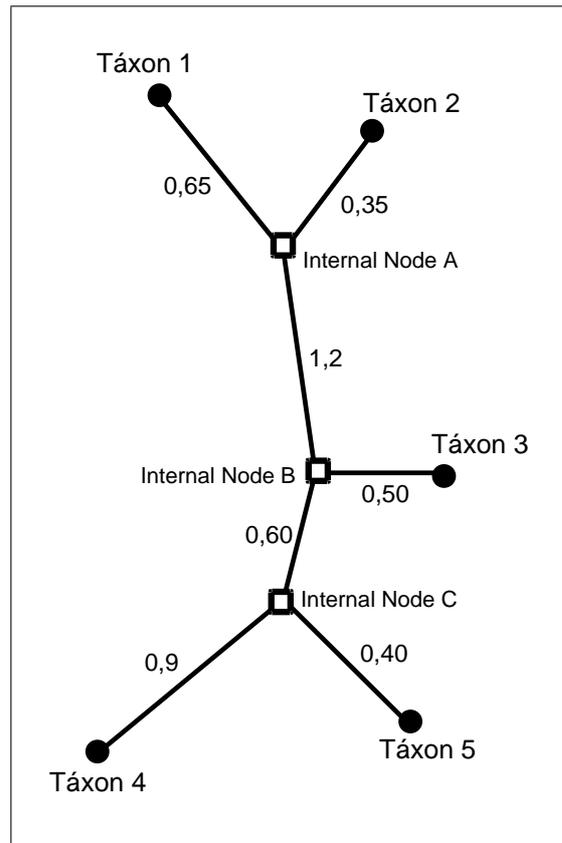
Para a criação das árvores filogenética, deve-se verificar a topologia da árvore, ou seja, as posições dos táxons em relação a outros táxons e nós e as distâncias entre estes elementos, que representam os comprimentos dos ramos que os separam. Para a criação das árvores, também denominada, a inferência filogenética dos táxons, há diversas técnicas, e várias delas são apresentadas na próxima seção.

4.2 TÉCNICAS DE FILOGENÉTICA MOLECULAR

As técnicas de filogenética que utilizam dados moleculares, denominadas filogenética molecular ou sistemática molecular, se dividem em dois grupos: taxonomia cladística e taxonomia numérica. A seguir apresenta-se a utilização e as principais características destas técnicas.

4.2.1 Taxonomia cladística

A filogenética molecular que utiliza taxonomia cladística ocorre através das análises das características de forma unitária. O primeiro passo consiste no um alinhamento



Fonte: Felsenstein (2003).

Figura 4.4 – Uma árvore exibindo os comprimentos dos ramos

Sequencia_1	ACTG--CCTAC
Sequencia_2	A-TGAACCTAC
Sequencia_3	A-GGAACGTAC
Sequencia_4	ACTG-ACCTA-
Sequencia_5	ACTG-A-CTTA

Quadro 4.1 – Alinhamento múltiplo para a inferência filogenética

múltiplo entre as seqüências em que se desejam inferir a filogenia. O Quadro 4.1 apresenta o alinhamento múltiplo de 5 pequenos trechos de DNA.

A taxonomia cladística analisará cada coluna de aminoácidos e verificará quais colunas são divergentes e para cada conjunto de divergências, uma ramificação será criada. A taxonomia cladística se divide em duas classes: as estatísticas e não estatísticas. Segundo Durbin et al. (1998, p. 193), os métodos não estatísticos funcionam procurando uma árvore que pode explicar as seqüências observadas com o menor número de substituições. Ou seja, o método de máxima parcimônia cria diversas árvores e escolhe aquela que representa a relação das seqüências com o menor número de substituições necessárias para a construção da árvore. A técnica de considerar o menor número de substituições

ou o menor caminho possível, é chamado de modelo de evolução mínima.

A Figura 4.5 exibe uma possível árvore filogenética para o alinhamento múltiplo exibido no Quadro 4.1. A figura mostra que para uma inferência filogenética através de máxima parcimônia, o objetivo é criar uma árvore com o menor número de mutações possíveis. Desta forma, as seqüências mais similares estão agrupadas juntas, como neste caso, as seqüências 1, 2 e 3 derivam do mesmo ramo, enquanto as seqüências 4 e 5, que possuem um maior número de mutações em relação as demais, estão agrupadas em outro ramo. Também é exibido as prováveis seqüências dos nós internos, baseados no conceito de evolução mínima.

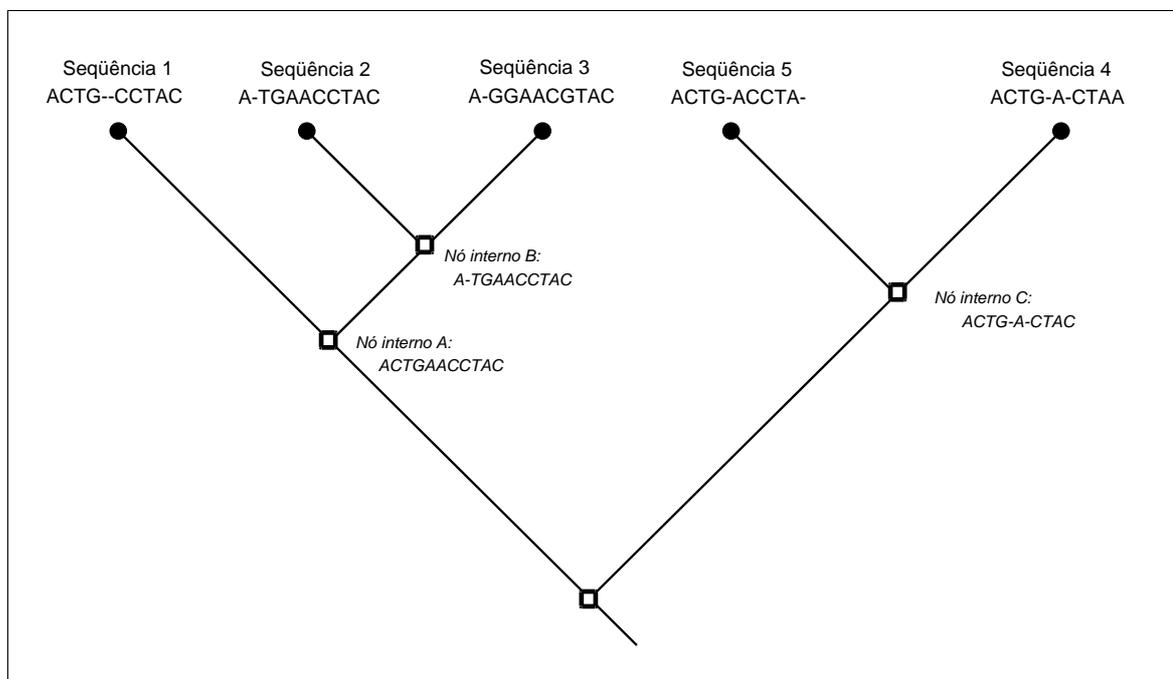


Figura 4.5 – Árvore representando uma inferência filogenética através da máxima parcimônia

É importante ressaltar que tratando-se de inferência filogenética, não se pode afirmar que os resultados são os corretos, mas sim, os mais prováveis, constituindo-se na construção de hipóteses com os dados obtidos e a verificação da ou das hipóteses mais prováveis.

As classes estatísticas de taxonomia cladística utilizam modelos de evolução e probabilidade deles estarem corretos. Enquanto a máxima parcimônia tem como objetivo minimizar o número de mutações necessárias, os métodos probabilísticos possuem modelos que verificam a possibilidade de determinada seqüência x ser ancestral de uma seqüência y na forma de $P(x|y, t)$. Durbin et al. (1998, p. 193) dizem que para poder fazer este tipo de verificação, são necessários modelos de evolução. Sabendo-se que na

evolução alguns resíduos² podem ser removidos, substituídos por outros ou inseridos em outras posições, são necessários modelos que verificam estes eventos e façam uma análise filogenética. O funcionamento destes métodos de uma forma geral é a criação de diversas árvores e a verificação de qual ou quais delas se enquadram melhor nos modelos propostos.

4.2.2 Taxonomia numérica

Enquanto a taxonomia cladística infere a filogenia através de análises pontuais das seqüências a serem analisadas, a taxonomia numérica utiliza distâncias entre os táxons a serem analisados. A taxonomia numérica utiliza uma matriz de distância entre os táxons.

Segundo Felsenstein (2003, p. 147), a idéia dos métodos que utilizam matrizes de distância é calcular a medida da distância entre cada par de espécies e armazenar numa matriz de distâncias. Após utilizar esta matriz para encontrar uma árvore que prediz o mais próximo possível, as distâncias entre as espécies da árvore e distâncias na matriz. Isto deixará de fora toda informação de alta ordem sobre as combinações dos caracteres que compõem as seqüências, reduzindo a matriz de dados a uma simples tabela de distâncias de pares. Felsenstein (2003, p. 147) complementa dizendo que estudos em simulações em computadores mostraram que a perda de informações sobre a filogenia é consideravelmente pequena quando utiliza-se matrizes de distâncias para representar as distâncias evolutivas entre as espécies.

4.2.2.1 Criação das matrizes de distância

Os métodos para a criação das matrizes de distância calculam o número de substituições entre as seqüências no alinhamento múltiplo. O método mais simples é o cálculo do número de substituições entre as seqüências alinhadas. Por exemplo, no Quadro 4.1, as Seqüências 1 e 2, possuem 3 substituições entre si, que são duas bases *A* removidas na seqüência 1 e uma base *C* removida na seqüência 2. Entre as seqüências 2 e 3, uma base *T* substituída por uma *G* e uma base *C* substituída por uma *G*. Percebe-se que o cálculo de distância das seqüências é fortemente influenciado pelo alinhamento múltiplo das seqüências. Caso o alinhamento múltiplo seja diferente, maximizando o número de encontros de determinadas seqüências e minimizando de outras, a matriz de distância resultando será diferente. A Figura 4.6 apresenta o fluxo para a criação da matriz de distância entre as seqüências de entrada. Primeiramente as seqüências são alinhadas e em seguida são calculadas as quantidades de diferenças entre elas.

²Resíduos são partes de seqüências, podendo ser um nucleotídeo ou aminoácido ou uma seqüência contígua destes.

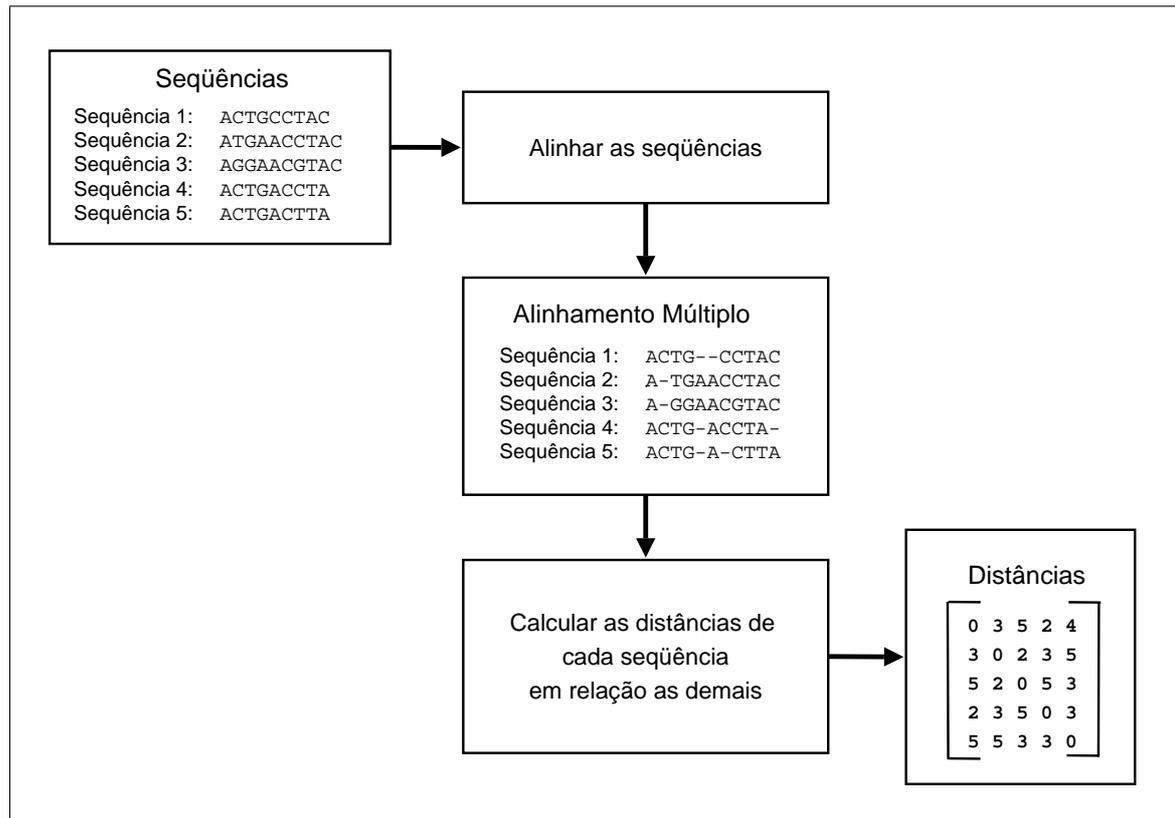


Figura 4.6 – Fluxo para a criação das matrizes de distância

A matriz de distância da Figura 4.6 representa as diferenças pontuais entre as seqüências, porém estas diferenças representam o tipo de mutação sofrida pela seqüência e a probabilidade desta mutação ocorrer. Existem modelos que buscam representar da melhor forma as substituições entre as seqüências. Felsenstein (2004) cita alguns modelos que o software *dnadist* presente no pacote PHYLIP, que é apresentado adiante, utiliza. O modelo de Jukes e Cantor estipula que todas as bases possuem a mesma probabilidade de substituição, porém a probabilidade aumenta em relação ao tempo. O modelo de Kimura-2 utiliza a mesma técnica de Jukes e Cantor, porém estabelece probabilidades diferentes para transições e transversões.

As transversões são mutações que modificam toda a estrutura química da base e por isto mais mudanças ocasionadas são mais severas e a probabilidade de ocorrer é menos comum do que das transições. Pelos motivos das transições e transversões ocorrerem em quantidades diferentes e também terem resultados diferentes, alguns modelos para os cálculos de distâncias e modelos para taxinomias cladísticas os tratam de maneira diferentes. Desta forma, deve-se utilizar o modelo que melhor represente a quantidade de transversões e transições existentes entre as seqüências e o grau de homologia entre as mesmas.

4.2.2.2 Métodos para inferência filogenética utilizando matrizes de distância

Os três métodos mais citados na literatura são o *Unweighted Pair-Group Method using Arithmetic averages* (UPGMA), método de grupo de pares sem peso usando médias aritméticas, o *Neighbor-joining* (NJ), juntando vizinhos, e o *Least Squares* (LS), menores quadrados. A seguir são apresentados estes três métodos, suas principais características e funcionamento, dando uma maior ênfase no método LS, pois uma versão modificada dele é utilizada como base deste trabalho.

O método UPGMA trabalha agrupando os táxons da matriz de distância em grupos, junto os mais próximos e formando grupos. Segundo Nei e Kumar (2000, p. 87), o UPGMA é o método mais simples em sua categoria. Os membros do grupo criado passam a representar um único táxon em relação aos outros grupos e táxons presentes. A árvore é construída a partir das criações dos grupos e junção deles.

Segundo Felsenstein (2003, p. 166), o NJ é outro algoritmo que trabalha pelo agrupamento. Primeiramente o algoritmo calcula a distância entre todos os pares possíveis de táxons e então ele escolhe o par que possui a menor distância. Então, para cada táxon restante, é escolhido aquele que possui a menor distância em relação aos táxons já adicionados na árvore. Este processo é repetido até que todos os táxons da matriz de distâncias sejam adicionados na árvore.

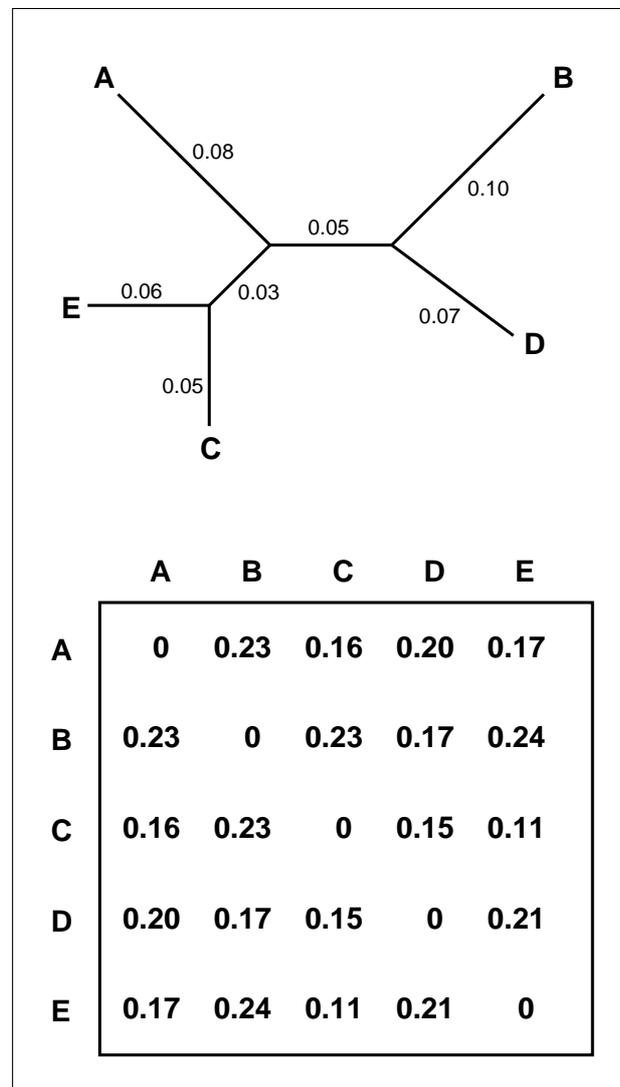
O método LS, ou, menor quadrado, é segundo Felsenstein (2003, p. 148) um dos métodos mais bem justificado estatisticamente. O mesmo autor diz que a idéia fundamental do método LS é que tem-se uma tabela (matriz) de distâncias (D_{ij}), e que qualquer árvore que contenha os comprimentos dos seus ramos, pode ser predito um conjunto de distâncias (que denota-se de d_{ij}). Ainda afirma que possui-se um meio para medir a discrepância entre as distâncias observadas e as distâncias esperadas. A medida usada no método de LS é exibida na Equação (4.1).

$$Q = \sum_{i=1}^n \sum_{j=1}^n w_{ij} (D_{ij} - d_{ij})^2 \quad (4.1)$$

O w_{ij} são pesos que diferem entre os métodos de LS. Felsenstein (2003, p. 148) apresenta alguns pesos, como o Cavalli-Sforza e Edwards (1967) que definem um modelo sem peso, onde o $w_{ij} = 1$ e Fitch e Margoliash (1967) que usa $w_{ij} = 1/D_{ij}^2$. Nei e Kumar (2000, p. 94) dizem que tanto utilizando pesos como o sem pesos normalmente dão a mesma topologia ou topologias muito similares.

Resumindo, a Equação (4.1) calcula as diferenças entre as distâncias observadas e as distâncias presentes na matriz. Sendo que quanto cada vez menor o valor de Q , melhor

a árvore representa a matriz de distâncias. A Figura 4.7 apresenta uma árvore com 5 táxons e a matriz de distância que foi utilizada para a construção da árvore.



Fonte: Felsenstein (2003, p. 149).

Figura 4.7 – Árvore com o comprimento dos ramos e a matriz de distâncias utilizada

O Quadro 4.2 apresenta como é feito o cálculo do LS. Primeiramente são calculadas as distâncias entre todos os táxons das árvores, somando o comprimento dos ramos que as separam. Após, é feita a diferença entre a distância da matriz e a distância observada elevada ao quadrado. E por fim, as diferenças são somadas, retornando assim o cálculo do menor quadrado da árvore. A árvore exibida na Figura 4.7 e seu cálculo apresentado no Quadro 4.2, possui como LS o valor de 0,250. Então, o objetivo dos algoritmos que utilizam o método LS é minimizar o valor da Equação (4.1) da árvore criada.

Para o cálculo dos comprimentos dos ramos, Felsenstein (2003, p. 150) e Nei e Kumar (2000, p. 96) apresentam um método onde os comprimentos dos ramos são considerados variáveis e as distâncias são resolvidas por um conjunto de equações lineares.

Calculo das diferenças:

$$(D_{AB} - d_{AB})^2 = (0,23 - (0,08 + 0,05 + 0,10))^2 = 0,0$$

$$(D_{AC} - d_{AC})^2 = (0,16 - (0,08 + 0,03 + 0,05))^2 = 0,0$$

$$(D_{AD} - d_{AD})^2 = (0,20 - (0,08 + 0,05 + 0,07))^2 = 0,0$$

$$(D_{AE} - d_{AE})^2 = (0,17 - (0,08 + 0,03 + 0,06))^2 = 0,0$$

$$(D_{BC} - d_{BC})^2 = (0,23 - (0,10 + 0,05 + 0,03 + 0,05))^2 = 0,0$$

$$(D_{BD} - d_{BD})^2 = (0,17 - (0,10 + 0,07))^2 = 0,0$$

$$(D_{BE} - d_{BE})^2 = (0,24 - (0,10 + 0,05 + 0,03 + 0,06))^2 = 0,0$$

$$(D_{CD} - d_{CD})^2 = (0,15 - (0,05 + 0,03 + 0,05 + 0,07))^2 = (0,5)^2 = 0,250$$

$$(D_{CE} - d_{CE})^2 = (0,11 - (0,05 + 0,06))^2 = 0,0$$

$$(D_{DE} - d_{DE})^2 = (0,21 - (0,07 + 0,05 + 0,03 + 0,06))^2 = 0,0$$

Soma das diferenças ao quadrado:

$$Q = 0,0 + 0,0 + 0,0 + 0,0 + 0,0 + 0,0 + 0,0 + 0,0 + 0,25 + 0,0 + 0,0$$

$$Q = 0,25$$

Quadro 4.2 – Cálculo do menor quadrado

Neste método há o problema de dependência de uma topologia para a árvore já existente. Felsenstein (1997) propõe outro algoritmo que utiliza o método LS, que trabalha de forma incremental, tendo um conjunto de táxons na árvore e adicionando novos táxons e é base do algoritmo proposto neste trabalho. Ele é melhor apresentado no Capítulo 7.

5 FILOGENÉTICA DE HOMOLOGIAS DISTANTES DE PROTEÍNAS

Este capítulo apresenta o trabalho de Theobald e Wuttke (2005) cuja finalidade é a inferência filogenética de proteínas através de modelos matemáticos dos domínios das proteínas.

5.1 INTRODUÇÃO A HOMOLOGIA DISTANTES DE PROTEÍNAS

As seqüências homólogas possuem similaridades, porém, proteínas homólogas podem possuir pouca similaridade na estrutura primária, apresentando maiores similaridades na suas estruturas terciárias e com isto poder-se inferir uma possível homologia, com maior consistência. Homólogos distantes são seqüências homólogas, mas com uma divergência acentuada, o que é refletido no baixo grau de similaridade das suas estruturas primárias.

Segundo Theobald e Wuttke (2005), muitas proteínas com estruturas primárias dissimilares dobram-se em estruturas similares. Isto quer dizer que o espaço das formas das proteínas é mapeado de forma redundante no espaço das seqüências das proteínas, já que diferentes seqüências podem ter formas muito similares. O motivo disto é discutido na Seção 2.2, onde diz que a importância das proteínas está em sua estrutura tridimensional e não na sua estrutura primária, a seqüência.

Theobald e Wuttke (2005, p. 722) dizem que o banco de dados de seqüências proteínas *wwPDB* (BERMAN; HENRICK; NAKUMA, 2002) possui mais de 20.000 seqüências de estruturas de domínios e apenas 700 a 800 dobras únicas de proteínas são conhecidas. As seqüências de estrutura de domínios são segmentos de seqüências de proteínas onde é conhecida a forma resultante dela. Desta forma, percebe-se que deve haver diversas seqüências divergentes que suas estruturas tridimensionais são similares. Theobald e Wuttke (2005, p. 722) afirmam que as proteínas que divergiram de um ancestral comum irão preservar elementos da sua herança em termos de função, estrutura e seqüência. Porque as estruturas terciárias são relativamente robustas a perturbações das seqüências, as dobras das proteínas são melhor conservadas e desenvolvem-se muito mais lentamente que as seqüências de amino ácidos.

Outra questão que deve ser analisada é a convergência evolutiva. Um exemplo morfológico de convergência evolutiva são as asas dos seres vivos. Tanto aves, quanto alguns mamíferos e insetos possuem asas, mas isto não significa necessariamente que

eles evoluíram de um ancestral comum portador destes membros. O motivo para que eles tenham membros similares e com o mesmo propósito foi a convergência evolutiva. O mesmo evento de convergência evolutiva ocorre com seqüências genéticas, protéicas e estruturas terciárias das proteínas. Como a quantidade de seqüências é praticamente infinita, imagina-se a quantidade de possíveis seqüências que possam ser formadas pelos 4 nucleotídeos ou os 20 aminoácidos sem limite de tamanho, a quantidade de possíveis dobras nas proteínas é finita e pequena. Considerando-se isto, a chance de ocorrer uma convergência evolutiva na forma tridimensional de proteínas é maior que em seqüências.

Através do conhecimento sobre as formas das proteínas, Theobald e Wuttke (2005) propõem a inferência filogenética utilizando estas informações para casos em que a similaridade das seqüências sejam baixa, menores que 25 – 40%. O trabalho anteriormente citado complementa ,ainda, que quando comparando dois domínios de proteínas, similaridade significativa entre as seqüências é uma evidência forte de um ancestral comum. Na falta de uma similaridade significativa entre as seqüências, função similar e estrutura da proteína são largamente consideradas como evidências de uma ancestralidade comum, baseado na improbabilidade de convergência de ambas estruturas e funções.

O mesmo trabalho apresenta um método baseado em modelos matemáticos dos modelos das proteínas a terem a filogenética inferida. Este modelo matemático tem como objetivo representar as principais características dos modelos das proteínas e permitir um meio de medição das similaridades entre os domínios das proteínas. Através das distâncias dos domínios das proteínas é construída uma matriz de distâncias, cujo a finalidade é ser entrada para um algoritmo de inferência filogenética que utilize taxonomia numérica. Para tal processo, tais autores disponibilizam um *workflow* para automatizar o a inferência filogenética de proteínas com homologies distantes.

5.1.1 *Workflow* para inferência de filogenias distantes

O *workflow* para inferência de filogenias distantes tem como objetivo apresentar uma árvore filogenética que represente as relações evolutivas das seqüências protéicas com baixa similaridades entre si. O *workflow* consiste de quatro etapas:

- a) pesquisar por seqüências similares de cada uma das seqüências de entrada;
- b) criar um alinhamento múltiplo para as seqüências similares de cada seqüência de entrada;
- c) comparar todos os alinhamentos múltiplos e calcular as distâncias entre eles;
- d) inferir a filogenia utilizando as distâncias calculadas.

Na Figura 5.1 é exibido cada etapa do workflow. Primeiramente tem-se um conjunto de seqüências genéticas com baixo similaridade, para cada seqüência, é pesquisado num banco de dados todas as seqüências que apresentem um grau de similaridade acima de um limite pré estabelecido. Após, é criado um alinhamento múltiplo para cada conjunto de seqüências similares encontradas. Os alinhamentos múltiplos são comparados um com os outros e verificado a similaridade entre eles e o resultado é posto numa matriz de distâncias. Com a matriz de distâncias geradas, é inferida a filogenia utilizando um método de taxonomia numérica.

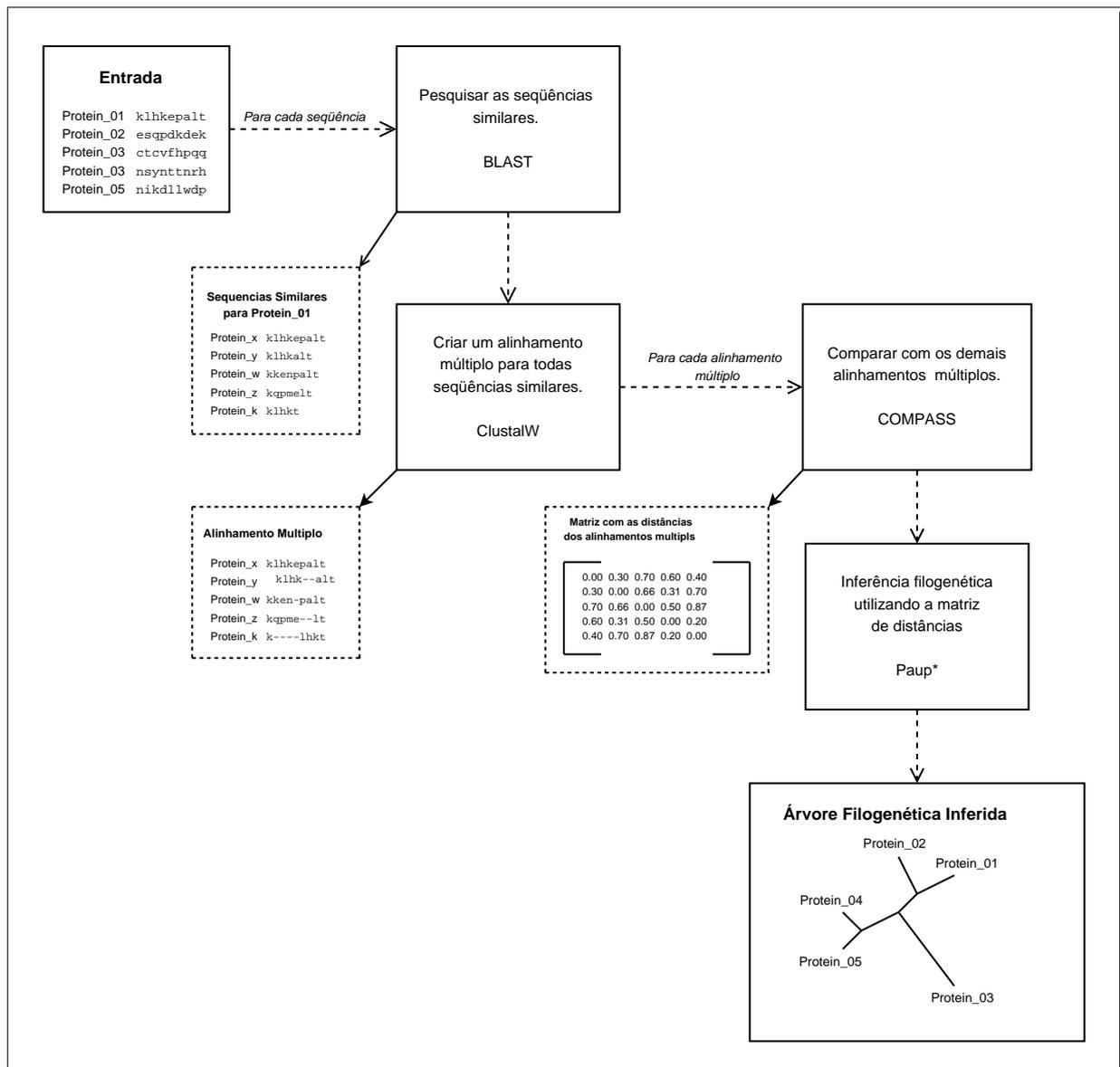


Figura 5.1 – Processo de execução do *workflow* proposto por Theobald e Wuttke (2005)

A pesquisa por seqüências similares é feita utilizando o algoritmo *Basic Local Alignment Search Tool* (BLAST) (ALTSCHUL et al., 1995) e uma implementação homônima. Este algoritmo é usado em pesquisas de bioinformática para encontrar seqüências simila-

res em banco de dados. Segundo Mount (2004, p. 248), entre as vantagens do BLAST, podem ser destacadas a velocidade para buscas de seqüências em grandes bancos de dados. Este apresenta um valor representando a probabilidade da seqüência similar encontrada no banco de dados ter ocorrido por acaso. Este valor é denominado *E-value*, que é calculado levando-se em consideração basicamente o comprimento da seqüência de entrada, comprimento da seqüência encontrada e uma constante que depende do tipo de seqüência, protéicas ou nucléicas.

O *workflow* através do BLAST pesquisa por seqüências similares no banco de dados. Através do software *curl* (CURL..., 2006)¹ obtem-se do banco de dados *GenBank* as seqüências em que o *E-value* estão acima de um limite pré-estabelecido. Segundo Mount (2004, p. 498): o *E-value* ou *expectation value* é um valor estatístico provido por programas que fazem análise de genoma através de pesquisas por similaridade de seqüências. A pontuação do *E-value* de um alinhamento entre uma seqüência de pesquisa e uma seqüência do banco de seqüências é um número que informa sobre a qualidade deste alinhamento e a chance dele ter ocorrido de forma aleatória. Quanto maior o valor *E-value*, maior a chance do alinhamento ter ocorrido ao acaso. Ou seja, o *E-value* é utilizado para informar sobre a qualidade do alinhamento feito entre a seqüência de pesquisa e cada uma das seqüências similares encontradas e no caso deste *workflow* é utilizado como pontuação para selecionar as seqüências similares.

Para cada conjunto de seqüências similares encontradas e que o seus valores *E-value* estejam abaixo do limite pré-estabelecido, cria-se um alinhamento múltiplo. O software utilizado para a operação de alinhamento múltiplo é o *ClustalW* (CHENNA et al., 2003). O *ClustalW* recebe como entrada um conjunto de seqüências e retorna estas seqüências alinhadas. Cada conjunto de seqüências alinhadas representa o domínio de uma proteína que foi dada como entrada do *workflow*, ou seja, no passo seguinte, a comparação dos alinhamentos múltiplos, é feita a comparação das domínios das proteínas.

A comparação dos alinhamentos múltiplos que representam os domínios das proteínas é feita utilizando o software *COMPASS*. Segundo Sadreyev e Grishin (2003, p. 326), o *COMPASS* é uma ferramenta que tem como objetivo a comparação de dois alinhamentos múltiplos de seqüências protéicas retornando o grau de similaridade entre os alinhamentos múltiplos e os segmentos mais similares dos alinhamentos. O *COMPASS* é utilizado por Theobald e Wuttke (2005) no *workflow* porque ele apresenta o *E-value* da comparação dos alinhamentos múltiplos das proteínas, que pode ser utilizado como medida para inferir a distância evolucionária entre os domínios das proteínas represen-

¹O software *curl* é uma ferramenta para linha de comando para transferência de arquivos utilizando endereços na internet.

tados nos dois alinhamentos múltiplos comparados. Sadreyev e Grishin (2003, p. 327) afirmam que o *COMPASS* trabalha montando dois modelos ocultos de *markov*, um para cada alinhamento múltiplo, e por fim compara os modelos construídos.

Com os resultados das comparações dos alinhamentos múltiplos, tem-se uma matriz de distâncias entre os domínios das proteínas. O *workflow* utiliza o software *Phylogenetic Analysis Using Parsimony (*and other methods)* (PAUP*) Swofford (2004). O PAUP* é um software para inferência filogenética utilizado em diversas pesquisas. O PAUP* pode ser utilizando tanto para taxonomia numérica como cladística. No *workflow*, como tem-se uma matriz de distâncias, utiliza-se taxonomia numérica, utilizando o método NJ para a inferência da árvore filogenética. A execução do PAUP* retorna uma possível árvore filogenética representando a filogenia entre as seqüências dadas como entrada do *workflow*.

O *workflow* gera um total de n árvores utilizando o PAUP*, sendo n a quantidade de seqüências de entrada. Para a geração de cada árvore, uma seqüência é removida da matriz de distâncias. As árvores geradas são utilizadas como entrada do software *consensus* do pacote PHYLIP² (FELSENSTEIN, 2005), o software *consensus* processa as árvores geradas e verifica a probabilidade de cada um dos ramos das árvores ocorrer, gerando uma árvore que exhibe um consenso destas probabilidades. Esta técnica de gerar árvores com dados diferentes e verificar a mais provável ou criar um consenso entre elas é chamada de *bootstrap*. A árvore retornada pelo software *consensus* é o resultado da execução do *workflow*.

²O PHYLIP é um pacote de diversos softwares, com objetivo de inferências filogenéticas, exibição de árvores filogenéticas e avaliação da credibilidade das árvores inferidas.

6 DESENVOLVIMENTO DAS OTIMIZAÇÕES DO *WORKFLOW*

A utilização do *workflow* apresentado na Seção 5.1.1 apresenta um problema em sua utilização: o tempo necessário para a execução completa do mesmo. O tempo de execução total do *workflow* com o conjunto de 91 seqüências disponibilizadas pelo autor do *workflow*, num computador Xeon 3Ghz com 4 gigabytes de memória *ram* é de aproximadamente 210 minutos. Com o intuito de minimizar este tempo, foi proposto o desenvolvimento de um algoritmo para inferência filogenética de forma distribuída.

Com o início deste trabalho, criou-se um *profiler* para o *workflow* para verificar o tempo gasto em cada parte da execução do *workflow*. O *profile* foi criado colocando-se marcações em pontos estratégicos do *workflow* e armazenando o momento em que aquele trecho foi executado. Estes dados armazenados são lidos e processados e por fim é exibido o tempo total gasto em cada parte do *workflow*. Na Quadro 6.1 é apresentado o tempo total de cada uma das fases do *workflow* em ordem de execução e o tempo total de execução do *workflow*.

Trecho	Tempo em segundos	Descrição
BLAST	4675	Pesquisa por seqüências similares
Curl	2202	Obtenção das seqüências similares
ClusalW	4811	Alinhamento das seqüências similares
Compass	918	Comparação dos alinhamentos múltiplos
PAUP*	0	Inferência da árvore filogenética
Consense	3	Consenso das árvores inferidas
Total	12614	Tempo total de execução

Quadro 6.1 – Tempo de execução do *workflow* sem alterações

Com a Quadro 6.1 iniciou-se a primeira fase do trabalho, que consiste na análise e otimização do *workflow*, visando a diminuição do tempo total de sua execução.

6.1 ANÁLISE E OTIMIZAÇÃO DO *WORKFLOW*

Analisando a Quadro 6.1 pode-se tirar algumas conclusões acerca o tempo de execução do *workflow*. A principal e mais relevante é que o tempo para inferência da árvore filogenética utilizando o software PAUP* é irrelevante se comparado com os demais passos do *workflow*. A partir desta constatação, procurou-se verificar os principais pontos a serem otimizados.

6.1.1 Obtenção das seqüências similares

A primeira otimização é sobre a utilização do software *curl*. Ele obtém os identificadores das seqüências que irão compor o alinhamento múltiplo e faz o *download* destas seqüências do banco de dados *GenBank*. A questão é que o *workflow* já possui estes dados localmente, pois o BLAST fez a pesquisa das seqüências similares utilizando-as. Ao invés de obter estas seqüências via *download*, elas devem ser obtidas via banco de dados local.

Os bancos de dados do BLAST são um conjunto de seqüências genéticas formatadas de maneira que as operações de pesquisa do BLAST sejam efetuadas de melhor forma. Para obter determinada seqüência neste banco de dados, existe o software *fastacmd* que acompanha o software BLAST. O *fastacmd* recebe como parâmetro o banco de seqüências e o identificador da seqüência que se deseja obter. Então, foi substituído o uso do software *curl* pelo software *fastacmd*. Substituído o software, foi feita uma nova execução no *workflow* e o tempo total de execução foi de 175 minutos, ou seja, com esta modificação, houve um ganho de 35 minutos.

6.1.2 Pesquisa por seqüências similares

A pesquisa por seqüências similares efetuada pelo BLAST, junto com o alinhamento das seqüências similares, são os trechos mais demorados do *workflow*. Para minimizar o tempo de pesquisa de seqüências, buscou-se um software que realizasse esta pesquisa de forma distribuída. O software escolhido foi o *mpiBLAST* (DARLING; CAREY; FENG, 2003). O *mpiBLAST* é uma modificação do software BLAST original para execução em ambientes distribuídos utilizando o MPI. O *mpiBLAST* fraciona o banco de seqüências e no processo de pesquisa por seqüências similares, atribui as frações do banco a cada um dos nós envolvidos na pesquisa. Uma vez que a pesquisa é distribuída pelos nós, seu tempo de execução é reduzido. Com esta modificação e executando o *workflow*, num *cluster* com 5 nós, seu tempo para as pesquisas de seqüências similares baixou de aproximados 77 minutos para 25 minutos.

6.1.3 Comparação dos alinhamentos múltiplos

Para a utilização do *mpiBLAST* é necessário um *cluster* MPI. Visando utilizar esta infraestrutura nas demais partes do *workflow* foram pesquisadas alternativas de softwares distribuídos que utilizem o padrão MPI. Para as comparações dos alinhamentos múltiplos criados, foi implementado um software agendador para execuções múltiplas do software num ambiente distribuído.

Utilizando o *workflow* com os dados disponibilizados pelos seus autores, há a comparação de 91 alinhamentos múltiplos entre si utilizando o *compass*, resultando num total de 4190 comparações. Representando isto em tempos, num computador Intel Xeon 3.0, totaliza 22 minutos. Visando diminuir este tempo total duas alternativas são possíveis: modificação do *compass*, para ser executado em modo distribuído ou criação de um agendador para possibilitar diversas execuções simultâneas no mesmo ambiente. Os problemas com a primeira opção envolvem a necessidade de conhecer o algoritmo e o código fonte do *compass* para efetuar qualquer mudança. Tal caminho não resolveria o cerne da questão, pois as execuções do *compass* não são lentas, o problema ocorre quando são necessárias diversas execuções. A solução encontrada foi o desenvolvimento de um agendador para múltiplas execuções do *compass* utilizando como infraestrutura um *cluster* MPI.

A técnica utilizada para permitir a execução de múltiplas instâncias simultâneas do *compass* foi a delegação através do agendador de um par de alinhamentos múltiplos a ser computado a cada processo do *cluster*. O agendador delega a cada processo um par de alinhamentos múltiplos. Após ele verifica em cada processo, se a comparação foi finalizada e obtém e armazena o resultado. Enquanto houver pares de alinhamentos múltiplos a serem comparados, este processo de delegação, esperar e obtenção dos resultados é repetida. Um diagrama que ilustra a seqüência de funcionamento do software é apresentado na figura 6.1.

6.1.3.1 Implementação do agendador

O agendador de execuções distribuídas do *compass*, foi implementado utilizando as linguagens *C* e *python* e o padrão MPI para a comunicação entre os processos. Todo o núcleo do agendador foi escrito na linguagem C, enquanto a parte dos processos trabalhadores, responsável pela execução e interpretação dos resultados do *compass* foi implementadas em *python*.

A opção por utilizar o *python*, uma linguagem de *script*, para a execução e leitura dos resultados do *compass* foi feita objetivando a possibilidade de modificar facilmente os parâmetros de execução do *compass*. É importante lembrar da existência de diferentes versões do *compass*, onde a exibição dos resultados da comparação diferem, tornando necessário a escrita de leitores dos resultados para as diferentes versões. Utilizando o *python*, a modificação da leitura e interpretação destes resultados torna-se bastante simples e adaptável.

Como tratamento dos parâmetros de execução e da leitura dos resultados são facilmente modificáveis, o agendador pode ser utilizado para agendar diversas execuções

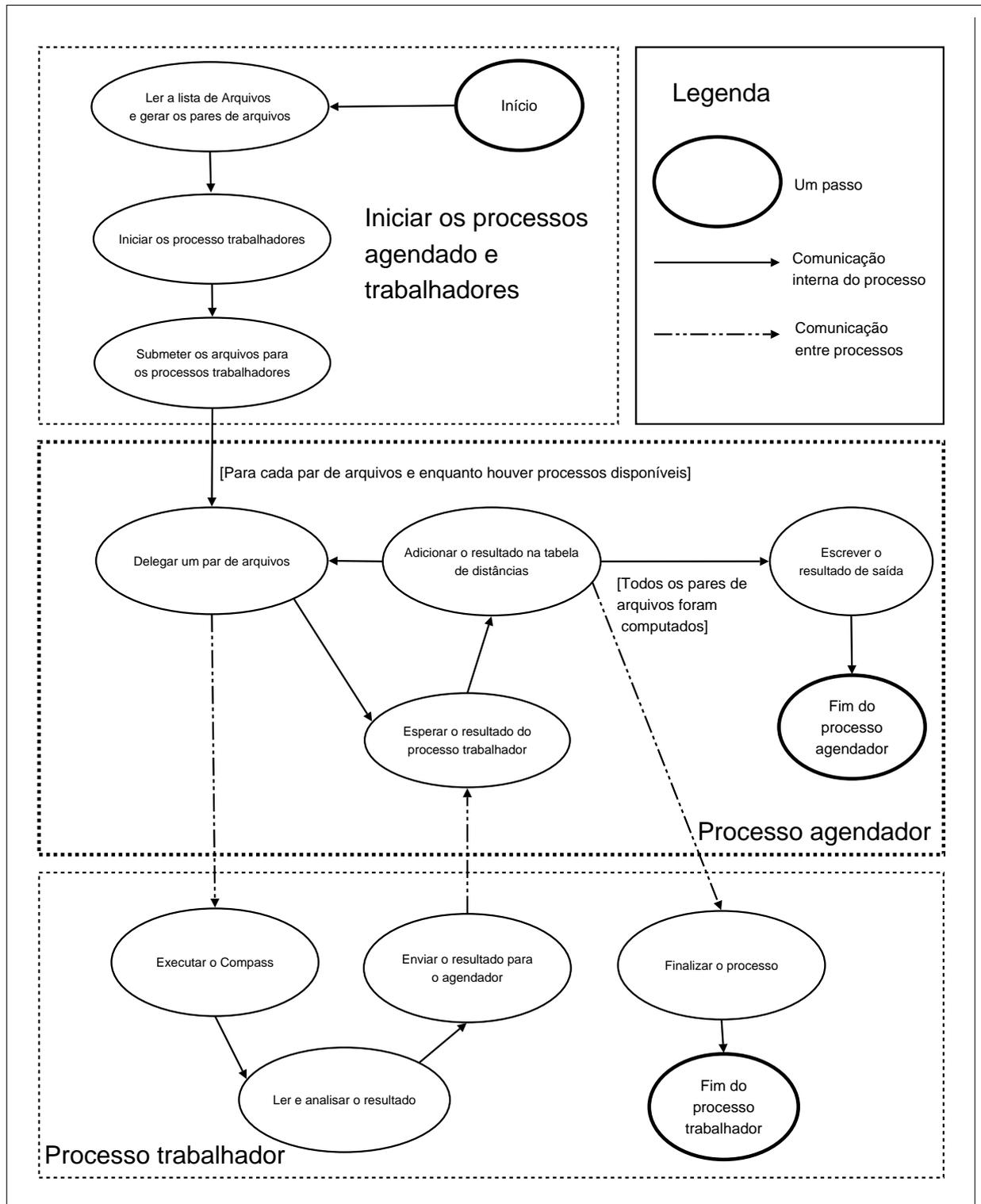


Figura 6.1 – Sequências de execução do agendador de múltiplas execuções do *compass*

num cluster MPI de outros softwares. Para isto, deve-se apenas modificar o *script exec_compass.py*, respeitando que a função chamada será a *get_value* e receberá como parâmetros duas *Strings*, que serão os nomes dos arquivos a serem comparados, e o retorno desta função será um *double*, contendo o resultado da comparação.

6.1.3.2 Resultados do agendador

O agendador foi testado num cluster formado por cinco computadores AMD Athlon XP 2600+ tendo como sistema operacional o Linux Fedora Core versão 5 e utilizado o *LAM* (BURNS; DAOUD; VAIGL, 1994) como implementação do padrão MPI. Foram feitas 11 rodadas de testes, a primeira com um processo executando o *compass* e um o agendador, outra com dois processos duas instâncias do *compass* e um o agendador e uma terceira rodada com três instâncias do *compass* e assim sucessivamente.

Como pode-se observar no gráfico na figura 6.2, a utilização do agendador proporciona um ganho proporcional de tempo até todos os processadores do cluster estejam executando uma instância do *compass*, algo que ocorre no valor 6. Após atingir o valor de 6 processos simultâneos (5 para o *compass* e um para o agendador), ainda ocorre uma pequena melhora com o aumento da quantidade de processos executados simultaneamente.

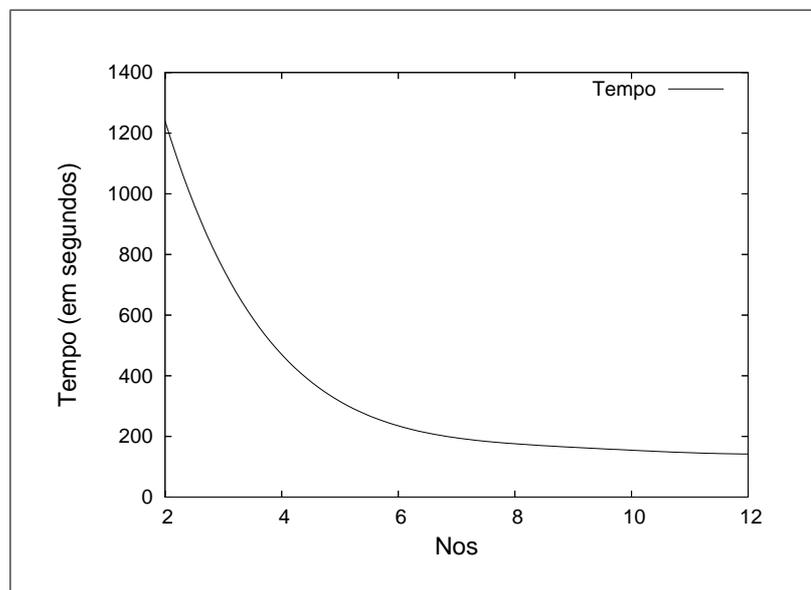


Figura 6.2 – Ganho com a utilização do agendador do *compass*

O agendador é uma ferramenta muito importante para a comparação de diversos alinhamentos múltiplos. Como visto na seção 6.1.3.2, ele consegue reduzir o tempo necessário para a comparação, proporcionalmente ao aumento da capacidade computacional do cluster. Outra característica, é a flexibilidade para efetuar modificações nos parâmetros de execução do *compass* e no modo em que é feito a interpretação dos resultados do soft-

ware executado.

O agendador está sendo utilizado nas pesquisas de genômica comparativa no DBBM/Instituto Oswaldo Cruz/FIOCRUZ e está se demonstrando uma ferramenta de grande valia. Ele é um software livre e sua última versão pode ser obtida em <http://www.sourceforge.net/projects/distphylo>.

6.1.4 Otimização da Criação dos Alinhamentos Múltiplos

Nas otimizações realizadas, ficou faltando uma otimização da criação dos alinhamentos múltiplos. Para este caso, podem ser feitas algumas abordagens: substituição por um software que possua um algoritmo mais otimizado, resultando em execuções mais velozes, implementação de um agendador para execuções múltiplas, da mesma forma que o agendador para o *compass* e uma implementação do software *ClustalW* distribuída. Para uma implementação do software *ClustalW* há o software *ClustalW-MPI* (LI, 2003).

Li (2003) diz que o *ClustalW-MPI* é uma implementação paralela e distribuída do software *ClustalW*, que paraleliza os processos internos do *ClustalW* proporcionando um ganho de tempo de 14 vezes num *cluster* com 16 processadores. Tal software não foi testado e utilizado no *workflow* porque não houve tempo para execução de testes mais precisos.

6.2 CONCLUSÕES SOBRE OTIMIZAÇÕES NO *WORKFLOW*

As otimizações feitas no *workflow* tiveram um impacto positivo no tempo de execução do *workflow*, minimizando o tempo total de execução de 210 para 110 minutos nos testes realizados, o que permite que o *workflow* seja executado com mais seqüências e que sejam feitas análises mais profundas com os dados através de variações dos parâmetros de pesquisa de seqüências e comparações de alinhamentos múltiplos.

7 INFERÊNCIA FILOGENÉTICA DISTRIBUÍDA

Este capítulo apresenta o algoritmo para inferência filogenética em ambientes distribuídos. Inicialmente é apresentado o algoritmo proposto por Felsenstein (1997) e em seguida o processo de desenvolvimento do algoritmo paralelo, quando então o algoritmo paralelo é formalmente apresentado e por fim tem-se os resultados e conclusões.

7.1 ALGORITMO ALTERNATIVO PARA INFERÊNCIA FILOGENÉTICA UTILIZANDO *LEAST SQUARES*

Conforme descrito na Seção 4.2.2.2, o método utilizado como base para o algoritmo distribuído de inferência filogenética é o LS. Segundo Felsenstein (1997), os métodos de LS são particularmente interessantes porque eles usam uma função objetiva singular para resolver o comprimento dos ramos e para escolha entre as topologias das árvores e podem ser utilizados para estimativas estatísticas sobre as melhores árvores. Felsenstein (1997) complementa dizendo que algumas (não todas) simulações em computadores têm mostrado que os métodos LS atuam de forma melhor que os outros métodos baseados em matrizes de distâncias.

O objetivo dos métodos de LS é inferir uma árvore filogenética que correspondam o melhor possível à matriz de distâncias que é utilizada como entrada. Para o cálculo do LS, que é o valor utilizado para verificar a qualidade da árvore, é utilizado o Algoritmo 7.1 que implementa a Equação (4.1). O Algoritmo 7.1 entre as linhas 3 e 10 itera entre todos os táxons da árvore. Em cada iteração, ele obtém um táxon da árvore e na linha 5 obtém a distância na matriz de distância em relação aos outros táxons e na linha 6 obtém esta distância na árvore. Na linha 53 é subtraído a distância dos táxons na da matriz de distâncias da distância destes táxons na árvore. Na linha seguinte a diferença desta subtração é elevada ao quadrado e adicionada na soma do quadrado das diferenças. O processo de obter um táxon e compará-lo a todos os demais táxons é efetuado para todos os táxons e as diferenças são somadas. Por fim, na linha 58 o valor representando as diferenças entre a matriz de distâncias e a árvore é retornada.

Um exemplo de uma árvore ótima e sua matriz de distâncias são exibidas na Figura 7.1. Percebe-se que o valor de Q utilizando a Equação (4.1) ou o Algoritmo 7.1 é zero. Ressalta-se que nem sempre é possível inferir uma árvore que o seu valor de Q será zero. Isto porque as distâncias entre os táxons dadas como entrada dependem de fatores, como o cálculo utilizado ou propriamente as seqüências utilizadas, que interferem

```

1 Algoritmo: Calcule-Least-Square
   input : tree que pretende-se calcular o least square
   output: O valor de least square da árvore

2 sum  $\leftarrow$  0;
3 foreach taxonNode  $\in$  tree do
4   foreach otheTaxonNode  $\neq$  taxonNode  $\in$  tree do
5     matrixDistance  $\leftarrow$  Get-Matrix-Distance(taxonNode, otheTaxonNode);
6     treeDistance  $\leftarrow$  Get-Tree-Distances(taxonNode, otheTaxonNode);
7     difference  $\leftarrow$  matrixDistance - treeDistance ;
8     sum  $\leftarrow$  sum + difference*difference;
9   end
10 end
11 return sum

```

Algoritmo 7.1: Calcular o *Least Square* da árvore

nas distâncias, fazendo com que elas apresentem diferenças nos valores. Outra questão inerente ao método de LS é a não possibilidade de saber se o valor de Q será zero ou qual será este valor. Para apoiar esta afirmação, pode-se fazer uma analogia do problema do caixeiro viajante: onde tem-se conhecimento do menor caminho apenas após calculá-lo e para saber qual é o menor caminho, é necessário executar o algoritmo, tornando inevitável a execução do algoritmo para conhecer a tamanho do menor caminho ou no caso do LS, para conhecer o valor de Q .

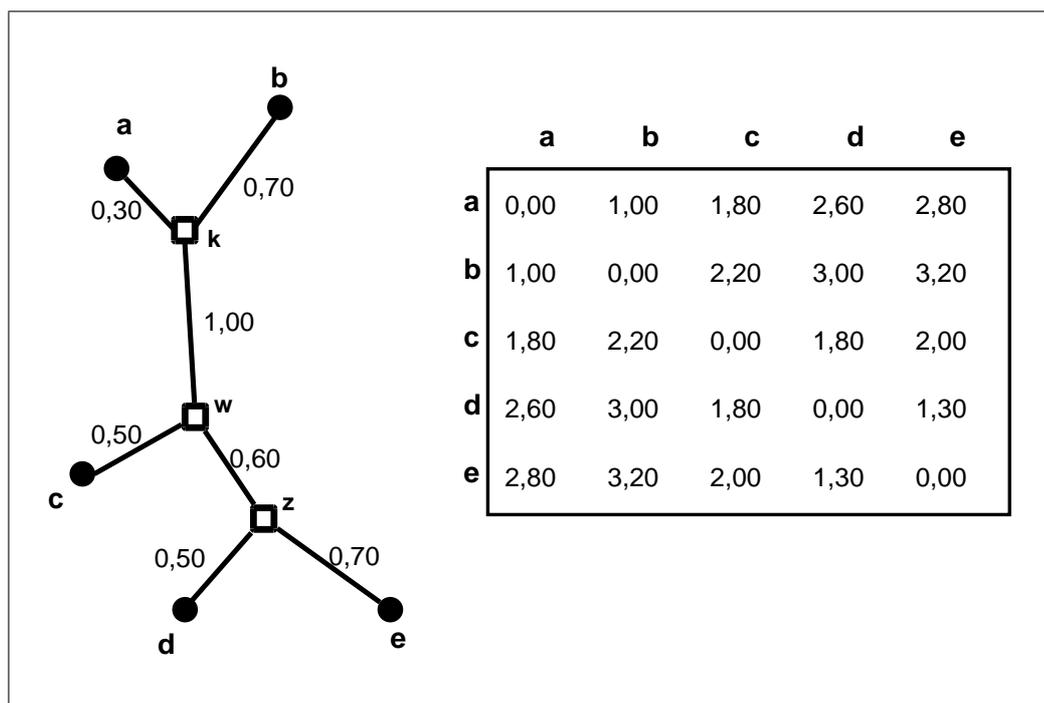


Figura 7.1 – Árvore ótima

Uma das dificuldades com os métodos LS é o cálculo do comprimento dos ramos. Eles podem ser calculados, dado uma topologia da árvore, através de um conjunto de equações lineares. Porém, segundo Felsenstein (1997, p. 101), o cálculo destas equações lineares pode ser complicados e os métodos computacionais para a resolução de tais equações podem ser computacionalmente dispendiosos.

O método alternativo de LS depende de transformações que temporariamente reduzem a dimensionalidade do problema. Ele encontra os comprimentos da árvore “podando-a”, diminuindo o número de nós e ramos, facilitando assim o cálculo do comprimento dos ramos. Um exemplo do funcionamento é exibido na Figura 7.2. Sempre transformando duas folhas em uma, até restar somente três folhas na árvore. Primeiramente são podados as folhas *A* e *B*, criando uma folha temporária *AB*. Após as folhas *AB* e *C* são podadas, restando as folhas *ABC*, *D* e *E*.

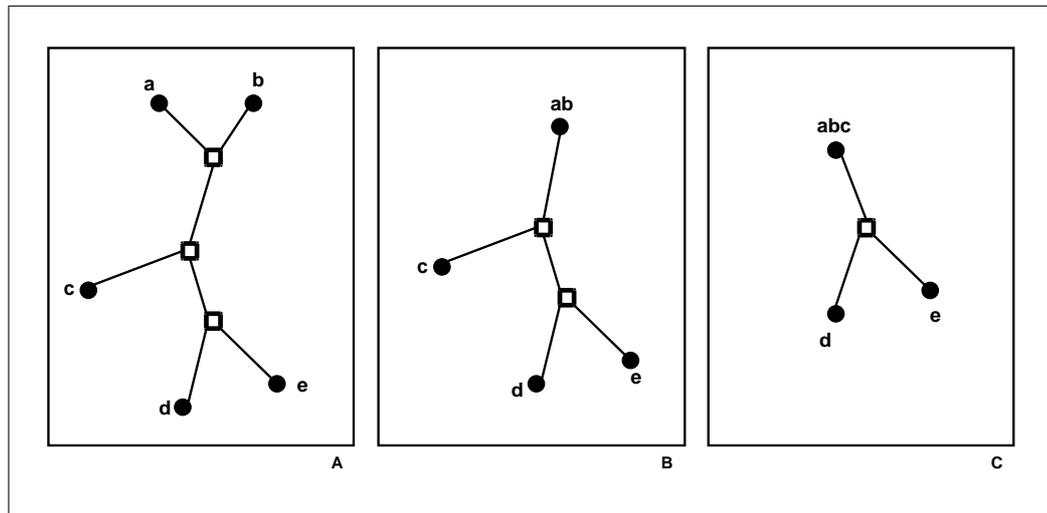


Figura 7.2 – Poda de uma árvore

A distância de um táxon em relação a um par de táxons é exibida na Equação (7.1). As distâncias D_{il} e D_{jl} são as distâncias das folhas em relação a um nó interno, preferencialmente o nó interno a qual estão ligadas e são lidos da matriz de distâncias. Os valores de w são os pesos utilizados para as distâncias. Segundo Felsenstein (1997) pode-se utilizar o valor de 1, onde assume-se homogeneidade na variância dos valores de D ou utilizar $w_{ij} = 1/D_{ij}^2$, onde assume-se que as distâncias são distribuídas proporcionalmente ao seu tamanho. Para este trabalho, será utilizado o valor de 1 para w . E os valores v_i e v_j são os comprimentos dos ramos que os ligam a seus pais. Na Figura 7.3 são exibidos na árvore as principais variáveis da equação.

$$D_{kl} = \frac{w_{il}(D_{il} - v_i) + w_{jl}(D_{jl} - v_j)}{w_{il} + w_{jl}} \quad (7.1)$$

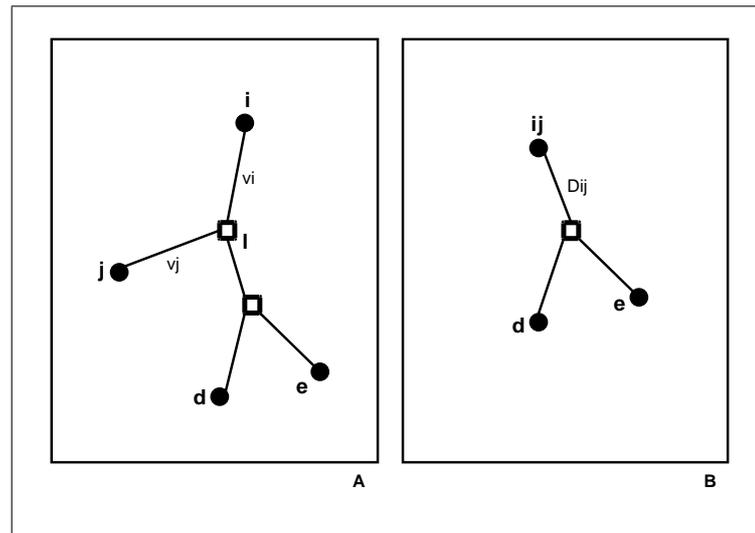


Figura 7.3 – Poda de uma árvore (continuação)

7.1.1 Funcionamento do algoritmo alternativo para inferência filogenética utilizando *Least Squares*

Nesta subseção é explicado o funcionamento do algoritmo proposto por Felsenstein (1997). A formalização do algoritmo é apresentada no Algoritmo 7.2 e entre cada parte da explicação é dita a linha da formalização do algoritmo do passo explicado.

Primeiramente na linha 2 são escolhidos três táxons da matriz de distâncias. Na linha 3 é obtido as distâncias entre estes táxons e na linha seguinte são calculados os comprimentos dos ramos de cada táxon na árvore utilizando a Equação (7.2).

Após o cálculo das distâncias, é criado um nó interno da árvore na linha 5 e entre as linhas 6 e 11 os três táxons com seus respectivos comprimentos dos ramos são adicionados ao nó interno previamente criado. Conforme a linha 12, enquanto a quantidade de táxons na árvore for menor que a quantidade de táxons na matriz de distâncias o processo a seguir é repetido.

Adiciona-se os nós em todas as posições possíveis e verificar em qual posição se obtém o menor valor para LS. Deve-se escolher a posição que obtiver o menor valor de LS e a manter nesta posição o táxon adicionado para a próxima iteração. Após cada iteração, deve-se atravessar a árvore otimizando os comprimentos dos ramos, primeiramente podando a árvore utilizando a Equação (7.1) e após restar três elementos, utilizar a Equação (7.2) para recalculer o comprimento dos ramos.

Na linha 14 se obtém um táxon ainda não adicionado na árvore e na linha seguinte, verifica-se cada possível posição para adicionar este táxon. Na linha 17 adiciona-se o novo táxon como vizinho de um táxon já presente na árvore e em seguida, calcula-se o LS

da árvore. Caso este seja o menor LS, mantém-se esta árvore. Após verificar todas as possíveis posições para adicionar o novo nó, na linha 23 são feitas as otimizações, que fica a cargo da implementação. Estas otimizações podem ser, por exemplo, o recálculo dos comprimentos dos ramos ou a troca dos táxons e ramos de posições. O objetivo destas otimizações é encontrar uma nova configuração para a árvore que contenha um menor LS. O processo de adição de um novo táxon, verificação da melhor posição e execução das otimizações, é repetido até que todos os táxons sejam adicionados. Por fim, a árvore inferida com o melhor LS é retornada.

$$\begin{aligned}v_a &= (D_{ab} + D_{ac} - D_{bc})/2 \\v_b &= (D_{ab} + D_{bc} - D_{ac})/2 \\v_c &= (D_{ac} + D_{bc} - D_{ab})/2\end{aligned}\tag{7.2}$$

Este algoritmo possui uma execução mais lenta que os demais métodos para inferência filogenética usando matrizes de distâncias, pois ele verifica todas as árvores possíveis para cada táxon e verifica diversas formas de otimizar a árvore. Outra questão sobre ele, ele é um algoritmo guloso, pois sempre utiliza unicamente a melhor opção para continuar a sua execução.

7.2 ALGORITMO PARALELO

Em busca de minimizar o tempo em que o algoritmo alternativo para LS é executado, procurou-se paralelizar o algoritmo e utilizar algumas heurísticas.

Primeiramente o algoritmo foi analisado e reportado alguns pontos cruciais sobre o seu funcionamento. Este processo tem a intenção de verificar possíveis pontos de paralelização do algoritmo e possíveis melhorias na sua execução. Além do algoritmo descrito em Felsenstein (1997), foram estudadas as implementações dos softwares *kitsch* e *fitch*, ambos do pacote PHYLIP.

Após estas análises são propostas algumas heurísticas, tendo como objetivo reduzir o tempo de computação da inferência filogenética utilizando LS. E por fim é apresentada a técnica utilizada para paralelizar este método e permitir execução de forma distribuída.

7.2.1 Análise dos software *kitsch* e *fitch*

Os softwares *kitsch* e *fitch* foram analisados porque são duas implementações do método de LS disponível no pacote PHYLIP. Eles podem fornecer sugestões de como

```

1 Algoritmo: An-Alternating-Least-Squarers
   input : distanceMatrix matriz de distâncias de táxons.
   output: A tree inferida com o o menor Least Square.

   // Obter três táxons da matriz de distâncias
2 taxona, taxonb, taxonc ← Get-Taxons(distanceMatrix);
3 dab, dac, dbc ← Get-Distances(Taxona, Taxonb, Taxonc);
4 va, vb, vc ← Calcule-Branch-Lengths(dab, Dac, Dbc);
5 internalNode ← New-Internal-Node(tree);
6 taxonNodea ← Add-Taxon(internalNode, taxona);
7 Set-Branch-Lenght(taxonNodea, va);
8 taxonNodeb ← Add-Taxon(internalNode, taxonb);
9 Set-Branch-Lenght(taxonNodeb, vb);
10 taxonNodec ← Add-Taxon(internalNode, taxonc);
11 Set-Branch-Lenght(taxonNodec, vc);
12 while Get-Size(tree) < Get-Size(distanceMatrix) do
13   bestTrees ← Null;
14   taxonToAdd ← Get-Taxon(distanceMatrix);
15   newTree ← Clone-Tree(tree);
16   foreach taxonNode ∈ newTree do
17     Add-Taxon-as-Neighbor(newTree, taxonNode, taxonToAdd);
18     Ls ← Calcule-Least-Square(newTree);
19     if Ls < Get-Least-Square(bestTrees) then
20       | bestTrees ← newTree ;
21     end
22   end
23   tree ← Optimize-Tree(bestTrees);
24 end
25 return tree ;

```

Algoritmo 7.2: Algoritmo proposto por Felsenstein (1997)

implementar um algoritmo utilizando o método LS. Um dos objetivos deste trabalho é a implementação de um algoritmo no pacote PHYLIP para execução em ambiente distribuído, com isto, torna-se fundamental o estudo das implementações contidas nele.

A análise foi dividida em duas partes: inicialmente estudou-se o código fonte destes softwares e após realizou-se a execuções deles utilizando um *profiler*, para medição dos tempos das execuções das suas funções.

O softwares *fitch* e *kitsch* são compostos por um arquivo fonte, *fitch.c* e *kitsch.c*, que contém a implementação do seu algoritmo e outros arquivos contendo funções auxiliares, como gerenciamento de memória, leitura e escrita de informações em arquivos e leitura das entradas do usuário. A análise do fonte focou a implementação do algoritmo com objetivo de compreender o seu funcionamento.

As definições das estruturas das árvores e dos nós são as mesmas para ambos os softwares, sendo definidas no arquivo *phylip.h*. Basicamente cada nós possui a sua distância da matriz de distância em relação a todos os outros nós e também a distância na árvore de todos os nós. A árvore possui um ponteiro para o nó interno inicial onde iniciou a sua construção. A distâncias entre os nós estão em vetores de distância, tornando o acesso a estas informações mais rápidas, porém piorando a compreensão da implementação.

O funcionamento de ambas implementações segue o mesmo padrão descrito na Seção 7.1.1, que é a construção de uma árvore com três táxons, utilizando a equação Equação (7.2) e adicionando táxons a esta árvore. Após cada inserção, a árvore é percorrida buscando otimizar as distâncias entre os nós. Ou seja, após cada adição do táxon numa posição, as distâncias são recalculadas e verificado se aquela posição e distâncias são realmente as melhores. No *kitsch* o cálculo das distâncias entre os nós é repetida após cada inserção e procura-se aproximar na árvore os táxons que possuem as menores distâncias, enquanto no *fitch* é feito basicamente a otimização das distâncias entre os nós folhas e os nós internos a que estão conectados.

Sobre o fonte, conclui-se que ele peca na organização, havendo diversas variáveis globais, funções e variáveis com nomes pouco descritivos e poucos comentários a respeito do funcionamento da implementação. A modificação do fonte para a implementação de um algoritmo distribuído é prejudicada, pois muito tempo seria gasto com uma compreensão mais profunda do código fonte ali presente.

Para verificar os pontos críticos da execução, foi utilizado o software *gnu prof* (FOUNDATION, 1997). Este software faz marcações dos tempos de execução do software e das funções que o compõe e após exibe um relatório com as funções mais custosas em termos de tempo do software.

Através da execução do *gnu prof* (FOUNDATION, 1997) verificou-se que as funções mais executadas do software *kitsch* são para recalcular as distâncias entre os nós das árvores e para a pesquisa de táxons cuja a distância são próximas. As funções mais executadas do software *fitch* são para atualização do comprimento dos ramos que ligam os nós e para o cálculo das distâncias dos ramos que conectam os nós folhas aos nós internos.

Através destas análises pode-se concluir que o principal trabalho dos softwares *fitch* e *kitsch* é a otimização dos comprimentos dos ramos. Então para especificar um algoritmo distribuído, deve-se considerar que o maior custo são as otimizações das árvores geradas e também deve-se considerar possíveis heurísticas para otimizar o tempo gasto com a criação de novas árvores e otimização dos seus valores.

7.2.2 Heurísticas

Loesch e Hein (1999, p. 196) afirmam que a eficiência significa resolver um problema, encontrando uma solução ótima, sem se preocupar com o tempo que isto leva. Já eficácia significa resolver um problema, isto é, encontrar uma solução, não necessariamente a melhor, dentro de um tempo razoável. Loesch e Hein (1999, p. 196) complementam que existe sempre o dilema da eficiência contra a eficácia. A partir destas afirmações, tem-se uma questão sobre o algoritmo para inferência filogenética: quanto é possível melhorar a eficácia do algoritmo sem sacrificar demasiadamente a eficiência do algoritmo?

A técnica para inferência filogenética proposta por Felsenstein (1997) pode ser visualizada como um problema de busca. Os árvore de busca, são outras árvores, resultantes da inferência filogenética até aquele ponto da busca. Na figura Figura 7.4 é exibido o processo de busca pela melhor árvore com valores de LS hipotéticos. A primeira árvore possui três táxons e apenas uma topologia é possível para ela. Para a inserção de um quarto táxon, três topologias são possíveis. Para o quinto táxon, são cinco as topologias possíveis. O crescimento do número de topologias possíveis para uma árvore sem raiz é dado por $(2n - 5)$, sendo n o número de táxons na árvore. Por exemplo, para uma árvore sem raiz com 20 táxons, são possíveis $2, 20 \times 10^{20}$ topologias diferentes.

A busca executada na Figura 7.4 é uma busca gulosa, sempre escolhendo a árvore que contenha o melhor LS. O problema com tal abordagem é que pode-se escolher um caminho na busca das árvores, porém este caminho não levará ao melhor resultado. Felsenstein (2003, p. 37) faz uma analogia das buscas gulosas com o problema da subida da montanha, onde nem sempre o trecho que possuiu maior elevação de altura leva a ponto mais alto da montanha.

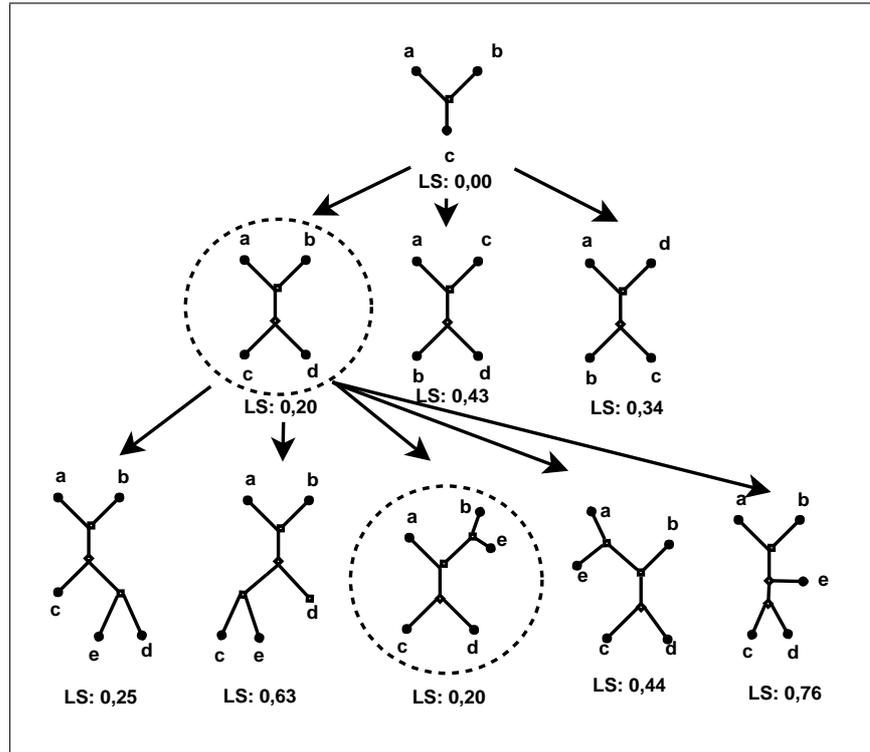


Figura 7.4 – Pesquisa pela melhor árvore

Sobre o crescimento do número de possíveis árvores, Stamatakis (2004, p. 14) diz que devido a esta explosão combinatória pode-se suspeitar que a inferência de árvores filogenéticas é *NP-Difícil*. Stamatakis (2004, p. 14) complementa, dizendo que esta dificuldade está para modelos menos elaborados e que encontrar uma árvore com filogenias perfeitas é realmente um problema *NP-Difícil*, porém com a utilização de funções rápidas para pontuações das árvores, permite encontrar árvores muito próximas às perfeitas em um tempo mais reduzido. A questão entre a qualidade das funções de pontuação e a sua velocidade de execução retorna a questão sobre o conflito de eficiência e eficácia.

As melhorias nos procedimentos para o cálculo da pontuação da árvore e para os cálculos dos comprimentos dos ramos, são baseadas na utilização de heurísticas. Loesch e Hein (1999, p. 196) dizem que a heurística é uma técnica que melhora a eficiência de um processo de busca, possivelmente sacrificando a obtenção da solução ótima. Loesch e Hein (1999, p. 197) complementam dizendo que é possível construir heurísticas de propósito específico, que exploram conhecimento específico do domínio da aplicação para resolver determinados problemas.

A primeira heurística proposta é selecionar os trios de táxons da matriz de distância que possuem as menores distâncias entre si. Por exemplo, na matriz exibida no Quadro 7.1, calcula-se as distâncias entre todos os possíveis trios. Após é escolhido os que possuem menores distâncias para iniciar a construção das árvores utilizando-os como

sementes. Segundo o exemplo apresentado no Quadro 7.1, os três trios mais próximos são o (a, b, c) , o (a, c, d) e o (b, c, e) . Desta forma, ao invés de iniciar a construção da árvore utiliza um trio qualquer, utiliza-se trios nas quais os táxons estão mais próximos.

O Algoritmo 7.3 apresenta um algoritmo para a escolha dos trios mais próximos. Na segunda linha deste algoritmo, é obtida a quantidade total de táxons na matriz de distâncias. Esta quantidade é utilizada entre as linhas 3 a 8, para obter-se todas as combinações possíveis de três táxons sem haver repetição de táxon no mesmo trio. Na linha 10 é calculada a distância total entre os táxons do trio e adiciona-se este trio a uma lista de trios. A lista de trios é ordenada de forma crescente na linha 15 e nas linhas 16 a 18 são adicionados, até que atinja o quantidade de trios requisitados, os trios na lista que será retorna.

```

1 Algoritmo:Create-Triples
   input : distanceMatrix matriz de distâncias de táxons e requestedTriples:
           quantidade de trios desejados como retorno.
   output: Uma lista contendo requestedTriples trios.

   // Quantidade de táxons na matriz de distancias
2 totalTaxons ← Get-Total-Taxons(distanceMatrix);

   // Iniciar criando todos os trios possíveis
3 for i ← 0 to totalTaxons do
4   | taxoni ← Get-Taxon(distanceMatrix, i);
5   | for j ← i + 1 to totalTaxons do
6   | | taxonj ← Get-Taxon(distanceMatrix, j);
7   | | for k ← j + 1 to totalTaxons do
8   | | | taxonk ← Get-Taxon(distanceMatrix, k);
9   | | | triple ← Create-Triple(Taxoni, Taxonj, Taxonk);
10  | | | tripleTotalDistance ← Calcule-Triple-Distances(triple);
11  | | | Add-Triple(listTriple, triple, tripleTotalDistance);
12  | | end
13  | end
14 end

   // Ordena os trios utilizando suas distâncias totais de maneira
   // crescente
15 listTriple ← Sort-List-Triples-by-Distances(listTriple);
16 for i ← 0 to requestedTriples do
17 | Add-Triple(listResult, triple);
18 end
19 return listResult ;

```

Algoritmo 7.3: Criação e seleção dos trios mais próximos

Através desta heurística, a construção da árvore iniciará através de um grupo de

Matriz de distâncias

$$\begin{array}{c}
 a \\
 b \\
 c \\
 d \\
 e
 \end{array}
 \begin{pmatrix}
 0,00 & 0,21 & 0,65 & 1,21 & 0,90 \\
 0,21 & 0,00 & 0,45 & 1,11 & 0,85 \\
 0,65 & 0,45 & 0,00 & 0,77 & 0,83 \\
 1,21 & 1,11 & 0,77 & 0,00 & 0,95 \\
 0,90 & 0,85 & 0,83 & 0,95 & 0,00
 \end{pmatrix}$$

Calculo das distâncias dos trios:

$$\begin{aligned}
 Dist(a, b, c) &= D_{ab} + D_{ac} + D_{bc} = 0,21 + 0,65 + 0,45 = 1,31 \\
 Dist(a, b, d) &= D_{ab} + D_{ad} + D_{bd} = 0,21 + 1,21 + 1,11 = 2,53 \\
 Dist(a, b, e) &= D_{ab} + D_{ae} + D_{be} = 0,21 + 0,90 + 0,85 = 1,96 \\
 Dist(a, c, d) &= D_{ac} + D_{ad} + D_{cd} = 0,65 + 1,21 + 0,77 = 2,63 \\
 Dist(a, c, e) &= D_{ac} + D_{ae} + D_{ce} = 0,65 + 1,90 + 0,83 = 3,38 \\
 Dist(a, d, e) &= D_{ad} + D_{ae} + D_{de} = 1,21 + 0,90 + 0,95 = 3,06 \\
 Dist(b, c, d) &= D_{bc} + D_{bd} + D_{cd} = 0,45 + 1,11 + 0,77 = 2,33 \\
 Dist(b, c, e) &= D_{bc} + D_{be} + D_{ce} = 0,45 + 0,85 + 0,83 = 2,13 \\
 Dist(b, d, e) &= D_{bd} + D_{be} + D_{de} = 1,11 + 0,85 + 0,95 = 2,91 \\
 Dist(c, d, e) &= D_{cd} + D_{ce} + D_{de} = 0,77 + 0,83 + 0,95 = 2,55
 \end{aligned}$$

Quadro 7.1 – Cálculo dos trios mais próximos

táxons que ficarão próximos na árvore resultante e a adição de algum táxon entre eles gerará uma árvore com o LS mais alto do que se o novo táxon adicionado não for vizinho delas. A construção da árvore partindo de um conjunto de táxons que se pressupõe que serão vizinhos na árvore resultante, fará com que o tempo gasto buscando otimizações, como troca dos lugares dos táxons, seja reduzido.

Conforme afirmado na Seção 7.2.1 os softwares *kitsch* e *fitch* dedicam muito do seu tempo de execução na otimização dos comprimentos dos ramos. O motivo disto, Felsenstein (1997) justifica explicando que o método recalcula diversas vezes as distâncias dos nós, até que as distâncias não sofram mais modificações e estes cálculos são feitos para todas as adições de táxon em posições diferentes.

Outra mudança proposta, é não recalcular as distâncias dos ramos de todas as árvores, porém após cada inserção, calcular o LS e eliminar as árvores que contenham este valor pior que a média de todas as árvores. Para verificar a qualidade do LS de determinada árvore em relação ao grupo, é calculado a dispersão deste valor em relação ao grupo. Sobre a dispersão de dados numéricos, Spiegel (1993, p. 104) afirma que o grau ao qual os dados numéricos tendem a dispersar-se em torno de um valor médio, chama-se variação ou dispersão de dados. Há várias medidas de dispersão ou de variação, sendo as

mais comuns a amplitude total, o desvio médio, a semi-interquartilica, a amplitude entre os centis 10-90 e o desvio padrão.

Ao invés de utilizar a média como linha de corte para as árvores, propõe-se utilizar o desvio padrão em conjunto com a média, cujo objetivo, é ter um parâmetro que considera a variação das distâncias das árvores para poder eliminar aquelas cujo os valores estão aquém de um limite tolerado e que se pressupõe que não possam conter no fim da execução do algoritmo a árvore com o menor LS. Desta forma, calcula-se a média entre o LS de todas as árvores e o desvio padrão. O desvio padrão é calculado conforme a Equação (7.3). Sendo σ o desvio padrão dos valores, \bar{x} a média dos valores e x_i o valor de cada um dos valores. Com o desvio padrão calculado, soma-se a média das distâncias e as árvores que possuem o LS acima deste valor são eliminadas.

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})^2} \quad (7.3)$$

O algoritmo proposto por Felsenstein (1997) procura otimizar as distâncias dos ramos de forma exaustivas, até que a árvore não sofra mais melhorias no seu LS. Considerando que tenha-se um conjunto de árvores e não somente uma para que as distâncias dos ramos sejam otimizadas, é muito dispendioso tentar otimizar ao máximo todas estas árvores. Desta forma, é executado uma otimização das distâncias dos ramos em cada árvore. Esta otimização é descrita no Algoritmo 7.4 e faz uso do Algoritmo 7.5.

O Algoritmo 7.4 itera em todos os nós internos da árvore, na linha 3 obtém os três nós conectados a ele e nas linhas 4 a 6 poda os nós para obter apenas três distâncias. Nas linhas 7 a 9, estas três distâncias são utilizadas para calcular os comprimentos dos ramos utilizando a Equação (7.2), o comprimento dos ramos dos três nós obtidos. E nas linhas 10 a 12 estes novos comprimentos são aplicados aos seus nós.

Após cada execução do algoritmo Algoritmo 7.4 é calculado o LS da árvore utilizando o Algoritmo 7.1. Neste caso, se a melhora das distâncias for baixa ou não houver, a árvore não é mais otimizada. Depois de cada otimização e cálculo do LS, é calculada a média e o desvio padrão e verifica-se se alguma árvore deve ser eliminada conforme descrito.

O Algoritmo 7.5 tem como objetivo podar um nó interno em relação a um nó que contenha um táxon obtendo uma distância entre estes dois elementos que possa ser utilizada na Equação (7.2). O Algoritmo 7.5 funciona da seguinte forma: na linha 2 se obtém todos os nós que não sejam conectados ao nó interno e que sejam diferentes do nó táxon-pai passado como parâmetro e ao outro nó utilizado para evitar recursividade

```

1 Algoritmo:Rearrange-Distances
   input : tree que pretende-se recalculer os comprimentos dos ramos
   output: tree com os comprimentos dos ramos recalculados

2 foreach internalNode  $\in$  tree do
3   nodea, nodeb, nodec  $\leftarrow$  Get-Children(internalNode);
4   distanceab  $\leftarrow$  Prunne-Node(Nodea, nodeb, internalNode);
5   distanceac  $\leftarrow$  Prunne-Node(Nodea, nodec, internalNode);
6   distancebc  $\leftarrow$  Prunne-Node(Nodeb, nodec, internalNode);

   // Calcular o comprimento dos ramos utilizando a Equação (7.2)
7   branchLengtha  $\leftarrow$  (distanceab + distanceac - distancebc) / 2;
8   branchLengthb  $\leftarrow$  (distanceab + distancebc - distanceac) / 2;
9   branchLengthc  $\leftarrow$  (distanceac + distancebc - distanceab) / 2;

   // Utilizar os comprimentos calculados
10  Set-Branch-Length(nodea, branchLengtha);
11  Set-Branch-Length(nodeb, branchLengthb);
12  Set-Branch-Length(nodec, branchLengthc);
13 end
14 return tree

```

Algoritmo 7.4: Reorganizando os comprimentos dos ramos de uma árvore

infinita. Sendo que os pesos dos nós internos na árvore binária deste algoritmo sempre será três e um dos nós será eliminado, pois será o nó táxon ou o nó táxon-pai, a iteração da linha 2 será repetida duas vezes. Na linhas 3 a 5, é verificado se o nó obtido é um nó interno, caso seja, este mesmo algoritmo é chamado novamente, passando como parâmetro o nó obtido para ser podado, o nó táxon, para calcular as distâncias e este nó interno, para evitar recursividade.

Como retorno, tem-se o nó obtido podado, ou seja, com uma distância relativa ao táxon node. Nas linhas 7 e 8 as distâncias relativas dos nós ao nó táxon são armazenadas numa matriz de duas posições e os comprimentos dos ramos dos nós obtidos também são armazenados. O cálculo da distância relativa ao nó táxon é feito na linha 10, onde as distâncias relativas do nó táxon e comprimentos dos ramos são obtidos da matriz onde foram armazenados e a Equação (4.1) é utilizada. Na linha 11 o resultado da cálculo é armazenado para futuras consultas e na linha seguinte este valor, representado a distância entre o nó táxon e o nó interno é retornada.

7.2.3 Paralelização do método de inferência filogenética utilizando *Least Square*

O objetivo de paralelizar o método para inferência filogenética utilizando *Least Square* é proporcionar a execução de forma distribuída, reduzindo o tempo de sua

```

1 Algoritmo:Prunne-Node
   input : internalNode contendo os nós a serem podados, taxonNode para referência
           para cálculo das distâncias e parentNode nó que chamou a função
   output: Um nó internalNode com a distância relativas dos nós podados

   // cada internalNode terá no máximo 3 filhos,
   // sendo que um nó é o taxonNode ou parentNode, pos será 0 e 1
2 foreach node  $\neq$  (taxonNode or parentNode)  $\in$  Get-Children(internalNode) do
3   | pos  $\leftarrow$  Pos(node);
   | // Chamada recursiva para poder este internal node filho de
   |   internalNode
4   | if node is InternalNode then
5   |   | node  $\leftarrow$  Prunne-Node(node, taxonNode, internalNode)
6   | end
7   | matrixDistances [pos]  $\leftarrow$  Get-Matrix-Distance(node,taxonNode);
8   | branchLengths [pos]  $\leftarrow$  Get-Branch-Length(node);
9 end

   // Calcular a distância utilizando a Equação (4.1), sendo  $w$  igual a
   1
10 distance  $\leftarrow$  ((matrixDistances [0] - branchLengths [0]) + ((matrixDistances [1] -
   branchLengths [1])) / 2;
11 Set-Matrix-Distance(internalNode, taxonNode, distance);
12 return internalNode

```

Algoritmo 7.5: Podando um nó

execução.

As unidades, ou seja, a granularidade, em que os processos ficariam responsáveis seriam: partes das árvores, árvores e conjunto de árvores. Possíveis alternativas para a paralelizarão:

- a) cada processo é responsável por parte dos dados de cada árvore e os processo executarem os cálculos de distâncias de forma conjunta;
- b) cada processo é responsável por uma árvore e as otimizações e cálculo de cada árvore ser responsabilidade de cada processo;
- c) cada processo é responsável por um conjunto de árvores e os cálculos em cada árvore, otimizações e seleção das árvores ser responsabilidade de cada processo.

As três alternativas diferem principalmente na granularidade dos dados. A granularidade se refere ao tamanho dos dados que são divididos entre os processos. No item *a*, a granularidade é alta se comparado as outras duas alternativas, pois os dados são divididos muitas vezes e o processo fica responsável por um conjunto com grande quantidade de pequenos elementos. No outro extremo, há o elemento *c*, onde a granularidade é baixa, pois a quantidade de dados sob responsabilidade de cada processo é menor, porém o tamanho de cada um dos dados é maior.

Deve-se modelar o processo distribuído levando-se em conta alguns aspectos:

- a) qual modelo maximiza a utilização dos processadores e minimiza o tempo ocioso;
- b) qual modelo minimiza a quantidade de mensagens transferidas e com isto miniza o tempo de espera entre elas;
- c) qual modelo maximiza a utilização dos processadores e reduz o tempo de processamento.

Em um modelo onde cada processo ficaria responsável por uma parte de cada árvore, o número de mensagens necessárias para efetuar os cálculos das árvores, é muito alto, diminuindo o tempo de processamento para a solução do problema e transferindo-o para a troca de mensagens. A granularidade em nível de árvore para cada nó apresenta melhor opção, pois o cálculo das árvores pode ser efetuado pelos processadores sem necessidade de troca de mensagens e sendo necessário apenas a troca de mensagens para o cálculo da seleção das árvores que continuarão na busca. Porém, considerando que a quantidade de árvores criadas no processo é alta, a quantidade de mensagens necessárias pode afetar negativamente o desempenho da busca pela melhor árvore.

Este trabalho propõe um busca em largura pela melhor árvore e desta forma o volume de árvores geradas e pesquisadas é alto. Por este motivo, decide-se que cada processo é responsável por um conjunto de árvores. Esta responsabilidade abrange a

seleção das árvores e otimização das distâncias dos seus ramos.

7.2.3.1 Algoritmo distribuído para inferência filogenética utilizando *least squares*

Nesta subseção é apresentado o algoritmo distribuído e seu funcionamento.

Este algoritmo é dividido em duas partes: o processo agendador e os processos trabalhadores. O processo agendador tem a função de selecionar os trios mais próximos, submetê-los aos processos trabalhadores e receber a lista das árvores geradas e selecionar as que devem continuar. Os processos trabalhadores tem a função de adicionar os táxons na árvore, otimizar as distâncias internas e calcular o LS.

O Algoritmo 7.6 apresenta o algoritmo para o agendador deste método paralelo para inferência de árvore filogenética utilizando LS. Na linha 3 é gerado, utilizando o Algoritmo 7.3, o número de sementes passadas como parâmetro para o algoritmo. Estas sementes são divididas entre os processos trabalhadores nas linhas 4 a 10, onde o agendador envia estas sementes a cada um dos processos. Na linha 13 é iniciado as iterações. Em cada iteração, os processos trabalhadores adicionam um novo táxon em cada uma das suas árvores. Para isto, o agendador, na linha 14, envia uma mensagem informando a cada um dos processos trabalhadores para que eles adicionem os táxons.

Entre as linhas 16 e 25 o agendador verifica cada um dos processos trabalhadores para saber se ele já finalizou. Caso o processo trabalhador tenha finalizado, o agendador obtém uma lista contendo as árvores geradas. Quando todos os processos trabalhadores finalizarem, o agendador na linha 26 verifica comparando as assinaturas das árvores se há árvores duplicadas e remove as cópias das árvores duplicadas, deixando apenas uma. Na linha seguinte, o agendador verifica as árvores que possuem o LS acima do limite, que é média mais o desvio padrão. Nas linhas 28 a 32, o agendador itera entre todas as árvores duplicadas e também as que estão acima do limite e envia uma mensagem para o processo trabalhador gerador de cada uma das árvores solicitando a remoção da árvore.

Após todos os táxons serem adicionados e os processos trabalhadores retornarem suas árvores, o agendador na linha 35 verifica a árvore com o menor LS entre todas geradas e requisita para que o processo trabalhador que a gerou a envie para que o processo agendador retorne esta árvore.

O processo trabalhador do algoritmo paralelo é apresentado no Algoritmo 7.7. No início do algoritmo, na linha 2, o processo trabalhador recebe do processo agendador um conjunto de sementes e com estas sementes são geradas as primeiras árvores. Na linha 4 inicia-se as iterações para adicionar novos táxons nas árvores geradas. Entre as linhas 8 e 13, o processo trabalhador adiciona um novo táxon a cada uma de suas árvores. Na linha

```

1 Algoritmo:Parallel-Least-Square-Schedule
   input : distanceMatrix contendo as distâncias entre os táxons e totalSeeds
           contendo a quantidade de sementes que serão geradas.
   output: tree com o menor LS encontrado.

2 totalProcessesWorker ← Get-Total-Processes() - 1;
3 listTriple ← Create-Triples(distanceMatrix, totalSeeds);
4 triplePerWorker ← totalSeeds / totalProcessesWorker ;
   // Enviar os trios para os processos trabalhadores
5 for i ← 0 to totalProcessesWorker do
6   | for j ← 0 to triplePerWorker do
7   |   | triple ← Get-Element(j +(i × triplePerWorker)) ;
8   |   | Send-Triple(triple, i);
9   | end
10 end
   // Busca pela melhor árvore
11 iteration ← 0;
12 totalTaxons ← Get-Total-Taxons(distanceMatrix);
13 while iteration + β < totalTaxons do
   | // Envia a mensagem para adicionar um novo táxon
14   | Send-Bcast-Message(ADD_TAXON);
15   | createdTreesList ← []; responseList ← []; totalFinished ← 0;
16   | while totalFinished < totalProcessesWorker do
17   |   | for i ← 0 to totalProcessesWorker do
18   |   |   | if Get-Status(i) = FINISHED then
19   |   |   |   | createdTreesList ← Get-Info-Created-Trees(i);
20   |   |   |   | Add(responseList, createdTreesList);
21   |   |   |   | Set-Process-Status(i, CHECKED);
22   |   |   |   | totalFinished ← totalFinished + 1;
23   |   |   | end
24   |   | end
25   | end
   | duplicateList ← Get-Duplicate-Trees(responseList);
   | treeBelowList ← Get-Trees-Below(responseList, limit);
   | foreach tree ∈ (duplicateList and treeBelowList) do
29   |   | i ← Get-Tree-Generator(tree);
30   |   | Send-Remove-Tree-Message(i, tree);
31   |   | Remove-From-List(tree, responseList);
32   | end
33   | iteration = iteration + 1;
34 end
35 return Get-Best-Tree(responseList);

```

Algoritmo 7.6: Agendador para o algoritmo paralelo

9 se copia a árvore onde será adicionado o novo táxon e na linha 10 se adiciona o novo táxon na árvore. Na linha 10 é calculado o LS da nova árvore utilizando o Algoritmo 7.1 e em seguida a árvore é armazenada. Na linha 15 é calculada a média e o desvio padrão dos LS das árvores geradas para que na linha 16 seja calculado o limite superior. O limite superior é utilizado nas linhas 17 a 19, para que as árvores que tenham LS acima do limite superior sejam removidas.

Buscando otimizar os comprimentos dos ramos dos nós, entre as linhas 20 e 27 itera-se sobre todas as árvores e executa-se o Algoritmo 7.4 em cada uma das árvores. Caso o valor do LS da árvore melhore, a árvore antiga é substituída pela árvore otimizada na lista de árvores na linha 25. Após as otimizações, o processo trabalhador envia ao agendador uma lista contendo o identificador, assinatura e valor do LS de cada uma das suas árvores. O processo agendador executa suas filtrações, já descrito no Algoritmo 7.6 e envia a lista das árvores que devem ser removidas. Enquanto o processo agendador processa a lista de árvores recebidas, o processo trabalhador fica bloqueado na linha 29 aguardando que o processo agendador envie a lista das árvores a serem removidas. Então na linha 30, o processo trabalhador remove as árvores solicitadas pelo agendador e inicia uma nova iteração. Quando todos os táxons forem adicionados na árvore, na linha 32 o processo trabalhador aguarda a mensagem final do agendador, caso seja para enviar a melhor árvore, o processo trabalhador, na linha 32, envia para o processo agendador sua melhor árvore.

De acordo com o observado e anteriormente descrito, os processos trabalhadores não enviam as árvores geradas para o processo agendador porque os valores LS das árvores são calculados nos processos e o custo para enviar as árvores para o agendador e recalculá-las são altos. Desta forma, criou-se um algoritmo que gera uma assinatura para cada árvore, este algoritmo utiliza o comprimento dos ramos e as ligações dos nós. Ele recebe como parâmetro um nó interno, preferencialmente o que deu início à construção da árvore e retorna um valor que representa a assinatura da árvore, O algoritmo para a criação de uma assinatura para a árvore é apresentado no Algoritmo 7.8. Ao contrário das funções de assinatura encontradas na literatura, esta não retorna um inteiro, porém um real. O motivo é que ela baseia nos comprimentos dos ramos, que são valores reais, para melhor representar as árvores.

A função *Remove-Tree* utiliza dois parâmetros não exibidos no algoritmo. Um parâmetro especifica a quantidade mínima de árvores e outro a quantidade máxima de árvores. Quando esta função for executada, ele removerá árvores até que a quantidade mínima ou continuará a remover até que a quantidade máxima. Estes parâmetros possuem

```

1 Algoritmo:Parallel-Least-Square-Worker
   input : distanceMatrix contendo as distâncias entre os táxons.
   output: tree com o menor LS encontrado.

   // Receber os trios gerados no agendador e criar as árvores
2 treeList ← createTrees(ReceiveTriples ());
3 totalTaxons ← 3;
4 while totalTaxons < Get-Total-Taxons(distanceMatrix) do
5   | geratedTreeList ← [];
6   | foreach tree ∈ treeList do
7     | // Obtém um táxon da matriz de distâncias que não esta
8       | inserido na árvore
9       | taxon ← Get-Taxon(tree, distanceMatrix);
10      | foreach taxonNode ∈ tree do
11        | clonedTree ← Clone-Tree(tree);
12        | clonedTree ← Add-As-Neighbor-Taxon(taxon, taxonNode, clonedTree);
13        | leastSquare ← Calcule-Least-Square(clonedTree);
14        | Add-Gerated-Tree(geratedTreeList, clonedTree, leastSquare);
15      | end
16    | end
17    | // Calcula a média e o desvio padrão do Least Square das árvores
18    | average, standarDeviation ← Standard-Deviation(geratedTreeList);
19    | // Elimina as árvores cujo o Least Square está acima do limite
20    | estabelecido
21    | threshold ← average + standarDeviation);
22    | foreach (Get-Least-Square(tree) > threshold) ∈ geratedTreeList do
23      | Remove-Tree(tree, geratedTreeList);
24    | end
25    | foreach tree in geratedTreeList do
26      | leastSquare ← Calcule-Least-Square(tree);
27      | // Otimiza os comprimentos dos ramos da árvore
28      | Rearrange-Distances(tree);
29      | newLeastSquare ← Calcule-Least-Square(tree);
30      | if newLeastSquare < leastSquare then
31        | Update-List(tree, geratedTreeList);
32      | end
33    | end
34    | Send-To-Schedule(geratedTreeList);
35    | // Aguarda o agendador enviar a lista das árvores que devem ser
36    | removidas
37    | toRemoveList ← Get-ToRemove-List-From-Schedule ();
38    | treeList ← Remove-Trees(toRemoveList, geratedTreeList)
39  end
40 if Get-Message () = SEND_BEST_TREE then
41   | Send-Tree-To-Schedule(treeList);
42 end

```

Algoritmo 7.7: Trabalhador para o algoritmo paralelo

como objetivo dar ao usuário a opção de configurar a largura da sua busca pela melhor árvore.

```

1 Algoritmo:Tree-Hash
   input : internalNode da árvore.
   output: value representando a assinatura da árvore.

2 value  $\leftarrow$  0;
3 foreach node  $\in$  Get-Children(internalNode) do
4   | value  $\leftarrow$  value + (Id-Hash(node)  $\times$  Get-Parent-Distance(node));
5   | if node is InternalNode then
6   | | value  $\leftarrow$  value + Tree-Hash(node);
7   | end
8 end
9 return value ;

```

Algoritmo 7.8: Função para geração de assinatura das árvores

Seguindo as informações descritas na Seção 3.1, este algoritmo é parcialmente síncrono, pois cada processo é executado independentemente, há, porém, um ponto de sincronismo entre eles após cada iteração. Há dois grupos de processos neste algoritmo: que contém o processo agendador e dos processos trabalhadores. Cada um destes dois grupos possui um conjunto de estados diferentes, porém relacionados. Na Figura 7.5 é apresentado o diagrama de estados do processo agendador e na Figura 7.6 é apresentado o diagrama de estado dos processos trabalhadores.

Conforme a Seção 3.1.1, as transições de estados dos processos ocorre através do envio ou recepção de uma mensagem. Por exemplo, a mensagem: *AdicionTaxon₀*, *Agendador₀*, *Trabalhador₀*, parte do agendador e faz com que ele e os demais processos mudem de estado. O processo agendador passará para o estado aguardando e os processos trabalhadores para o estado processando. A mensagem: *TaxonAdicionado₁*, *Trabalhador₁*, *Agendador₁*, faz com que o trabalhador que enviou esta mensagem mude para o estado aguardando e o agendador inicie as verificações das árvores geradas. São estas duas as principais mensagens, que modificam os estados internos dos processos. Para verificação da finalização do processo, não é necessário o envio de mensagem, pois esta verificação é feita em cada um, apenas verificando se todos os táxons foram adicionados.

O diagrama de distribuição do algoritmo paralelo é exibido na Figura 7.7. Nele é exibido 4 processos, sendo 3 trabalhadores e 1 agendador que dependem do arquivo com a matriz de distâncias e o agendador, que depende do arquivo de árvore inferida, onde ele escreve o resultado da execução. O canal de comunicação entre os processos, no caso da implementação que é exibida a seguir, é o MPI, porém pode ser outra biblioteca de

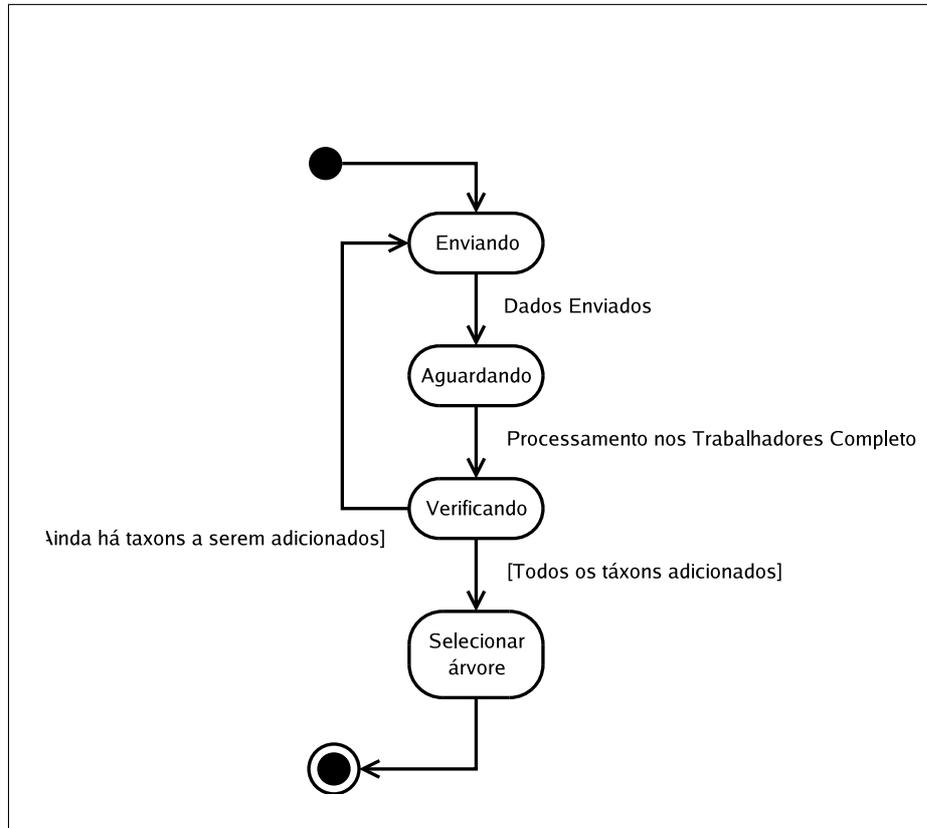


Figura 7.5 – Diagrama de estados do processo agendador

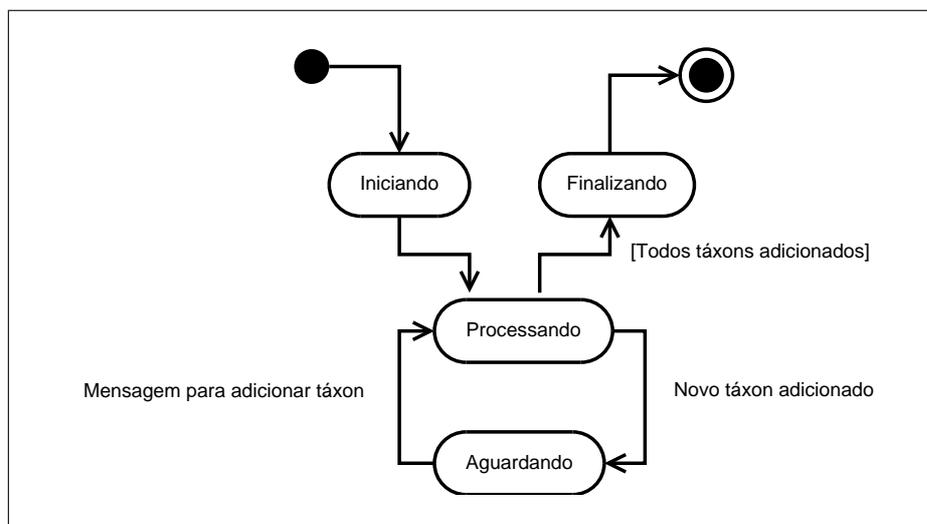


Figura 7.6 – Diagrama de estados dos processos agendadores

comunicação distribuída ou outros meios de comunicação entre processos, por exemplo, memória compartilhada.

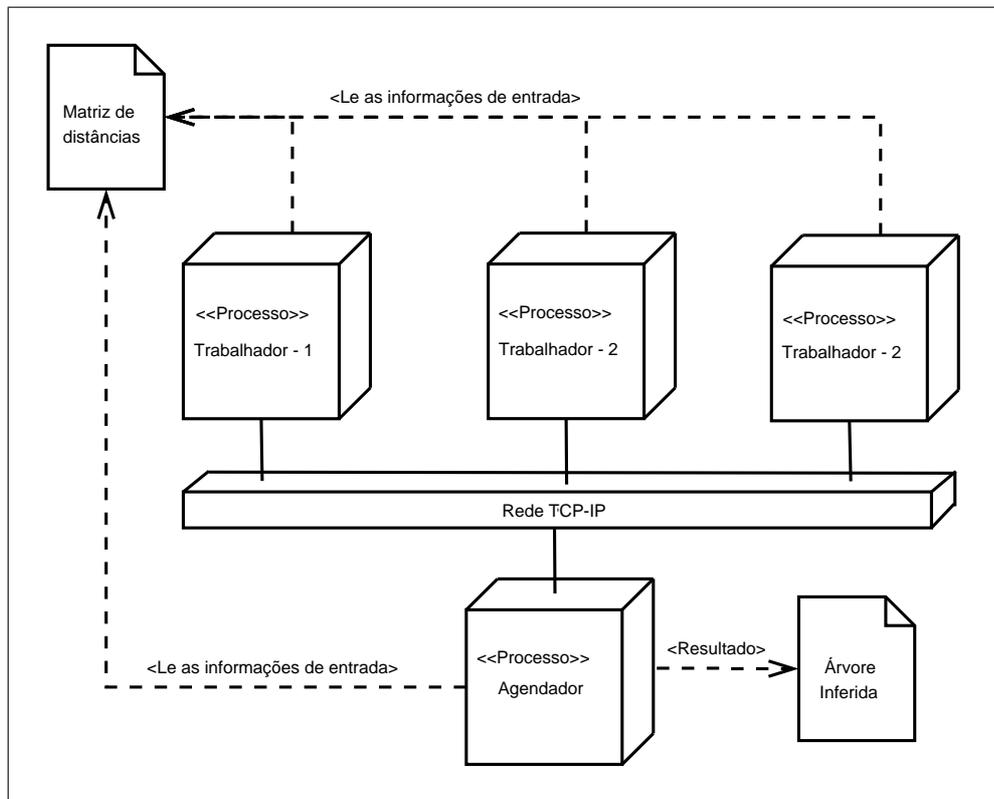


Figura 7.7 – Diagrama de distribuição do algoritmo

7.2.3.2 Implementação do algoritmo paralelo

A implementação do algoritmo paralelo para a execução num ambiente distribuído foi feita utilizando a linguagem de programação *C* e a biblioteca *LAM* (BURNS; DAUD; VAIGL, 1994) como implementação do padrão *MPI*. A escolha do padrão *MPI* é porque ele é utilizado em implementação distribuídas de softwares para bioinformática, como o *mpiBlast* (DARLING; CAREY; FENG, 2003), e é um padrão reconhecido e utilizado na indústria de informática para a construção de *clusters* computacionais. A linguagem *C* é utilizada porque a implementação do *MPI* utiliza, o *LAM* é implementado para esta linguagem. Há como utilizar o *LAM* em outras linguagens, porém por falta de tempo para efetuar testes e os riscos de não possuírem todas as funcionalidades desejadas e sendo *C* uma linguagem portátil para diversas plataformas, optou-se por ela. Como compilador e depurador, foi utilizado o *GNU Compiler Collection* (GCC) (FOUNDATION, 2006a) e o *GNU Project debugger* (GDB) (FOUNDATION, 2006b) respectivamente. Junto com a implementação, foram executadas rodadas de testes para verificar a existência de erros e corrigi-los.

A implementação focou a utilização do algoritmo e suas implementações, objetivando a funcionalidade delas e as legibilidade do código fonte. Por este motivo, possíveis otimizações do código fonte visando maior velocidade de execução não foram utilizadas, pois ocasionariam a perda de legibilidade e conseqüentemente compreensão do código fonte. Como exemplo, tem-se as implementações dos softwares *fitch* e *kitsch* utilizam vetores contendo as diversas distâncias utilizadas nos cálculos. Na implementação do algoritmo paralelo, utilizam-se tabelas *hash* para armazenar as distâncias utilizadas. Enquanto o acesso dos dados dos vetores é feito através de inteiros, resultantes de alguns cálculos, as tabelas *hash* permitem obter as distâncias utilizando os identificadores dos elementos.

O primeiro passo da implementação foi o uso do algoritmo sem distinção entre o processo agendador e os processos trabalhadores. Nesta primeira implementação, tem-se todas as fases do algoritmo: criação e seleção dos trios, adição dos nós nas árvores, verificação dos nós e otimização das distâncias. Esta primeira etapa teve como objetivo implementar os tipos e funções utilizadas no algoritmo e verificar suas funcionalidades. Para estruturas de dados como tabelas *hash* e listas, leitura de arquivos foi criada uma biblioteca chamada de *pih-lib*. Esta biblioteca é implementada na linguagem *C* e tem como objetivo abstrair o uso das estruturas de dados citadas e permitir a reusabilidade de código entre os softwares implementados, como o agendador de processos descrito na Seção 6.1.3 e a implementação do algoritmo aqui apresentado.

Para a leitura dos arquivos contendo as distâncias dos táxons, foram implementados funções que lêem os formatos de matrizes de distâncias dos softwares PAUP* e do software PHYLIP. O tipo *values_table_t* representa uma matriz de distâncias, este tipo possui dois principais tipos de operações sobre ele: a adição de um novo par de distâncias e a leitura de um par de distâncias, passando como parâmetro os nomes das seqüências. Uma possível otimização neste caso é ter apenas um ponto no software onde as *strings* são criadas e não criar *string* duplicadas. Desta forma, ao invés da comparação de *strings* caracter por caracter, que possui um custo $O(n)$, sendo n o comprimento da *string* mais curta, é possível compará-las utilizando suas referências, tendo como custo $O(1)$.

Para representar a árvore, o Quadro 7.2 apresenta a estrutura do tipo *tree_t*. Os itens que a compõem são: um identificador, o nó que criou esta árvore, a iteração atual da árvore e o contador de árvores criadas nesta iteração, estas informações utilizadas somente na implementação paralela. Em seguida, temos uma referência para a árvore que na qual foi originada. Logo após, têm-se duas tabelas *hash*, a primeira contém todos os nós internos e a seguinte contém os nós que representam os táxons, além de possuir uma lista

```

1 struct __tree {
2     char*         identifier;
3     int           node_name;
4     int           iteration;
5     int           tree_count;
6     tree_t        previous_tree;
7     hash_table_t  internal_nodes;
8     hash_table_t  taxon_nodes;
9     list_t        taxon_remains;
10    double         value;
11    double         hash;
12    unsigned int   last_identifier;
13    internal_node_t initial_node;
14 };

```

Quadro 7.2 – Estrutura do tipo *tree_t*

```

1 struct __taxon_node {
2     char          *identifier;
3     unsigned int  hash;
4     internal_node_t parent;
5     double        parent_distance;
6 };

```

Quadro 7.3 – Estrutura do tipo *taxon_node_t*, que representa um táxon na árvore

dos táxons que estão na matriz de distâncias não adicionados na árvore. A árvore possui um atributo que contém o valor do seu LS e outro atributo que contém o seu valor *hash*. Há inteiro cuja serventia é a criação dos identificadores dos nós internos e um ponteiro para o nó interno que deu início a construção da árvore.

Existem funções para operação nas árvores que criam uma nova árvore através de três táxons, para a clonagem de árvores e todos os seus dados, para a adição de um novo táxon e para o cálculo do seu LS.

Para representar os nós, são utilizados três tipos: um para representar um nó generalizado, outro para representar especificamente um nó interno e outro para um nó que contenha um táxon. O tipo *tree_node_t* apresenta um nó e contém dois atributos, o tipo do nó que ele é: um nó táxon ou um nó interno e uma referência para uma estrutura deste tipo que contém os demais dados. Para representar os nós que contém os táxons, é utilizado um tipo *taxon_node_t* que contém a estrutura apresentada no Quadro 7.3. Esta estrutura contém o identificador do táxon, uma assinatura *hash* deste identificador, um ponteiro para o nó interno que está conectado o comprimento do ramo em relação ao nó interno que se esta conectado.

```

1 struct __internal_node {
2     char          *identifier;
3     unsigned int   hash;
4     internal_node_t parent;
5     double         parent_distance;
7     hash_table_t   children;
9 };

```

Quadro 7.4 – Estrutura do tipo *internal_node_t*, que representa um nó interno na árvore

Para representar um nó interno, utiliza-se o tipo *internal_node_t* que contém a estrutura apresentada no Quadro 7.4. Esta estrutura contém o identificador do nó interno, uma assinatura *hash* deste identificador, um ponteiro para o nó interno que está conectado ou *NULL* caso seja o nó inicial, a distância para o nó interno que se está conectado uma uma tabela *hash* contendo os nós que estão conectados a ele.

Finalizada a etapa da implementação da versão não distribuída, deu-se início à implementação distribuída, onde utilizou-se todas as estruturas e funções da versão não distribuída. Para implementação do algoritmo distribuído, foram criados três arquivos: *dleast_squares*, com a iniciação do software, o *scheduler* contendo a implementação do agendador e *worker* com a implementação dos processos trabalhadores.

A Figura 7.8 apresenta os principais arquivos da implementação e suas dependências. Todos os demais arquivos dependem de *pih_lib*, pois este implementa as estruturas de dados básicas e leitura de arquivos. No arquivos *tree* estão implementados os tipos da árvore e as funções que operam sobre as árvores, como criação, adição de nós, cálculo de LS e otimização das distâncias. As operações que utilizam diretamente as funções do MPI estão encapsuladas no arquivo *tree_mpi*. Desta forma, os arquivos, *schedule* e *worker*, que implementam os algoritmos distribuídos, executam as operações distribuídas sem utilizar diretamente o MPI, facilitando a compreensão e manutenção do código fonte.

As informações trocadas entre o processo agendador e os trabalhadores são os trios iniciais, as informações sobre as árvores e a melhor árvore encontrada. Para envio dos trios iniciais do processo agendador aos processos trabalhadores e das informações das árvores, foram definidos tipos para o MPI conforme a Figura 3.6. A estrutura que contém as informações de cada árvore é exibida no Quadro 7.5 e a criação do tipo para que o MPI transfira dados deste tipo é exibido na Quadro 7.6. O Quadro 7.5 apresenta na sua segunda linha o identificador da árvore, após um inteiro contendo o número do processo que contém esta árvore e em seguida a assinatura da árvore e o valor LS dela.

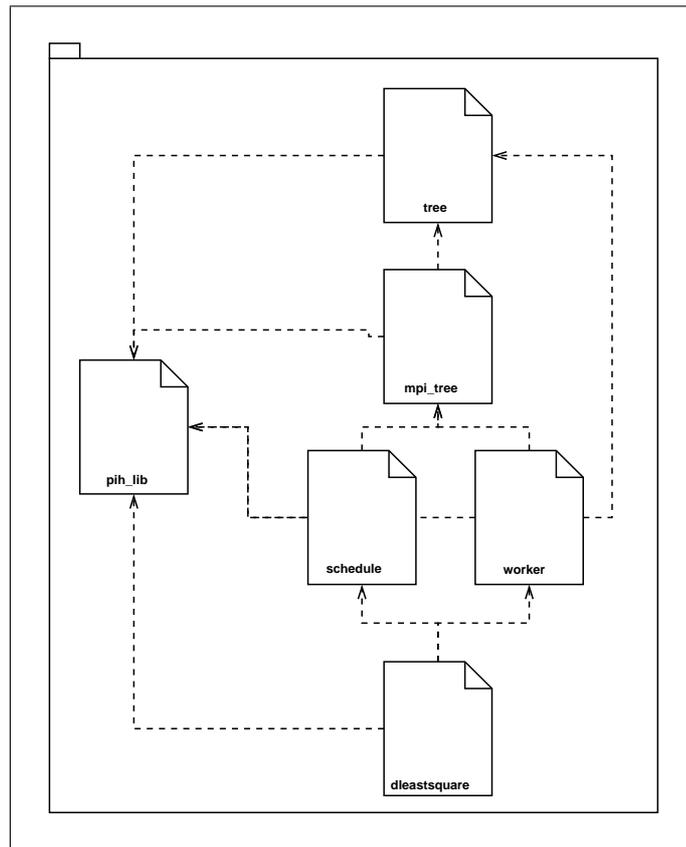


Figura 7.8 – Principais arquivos da implementação

```

1 struct __tree_info {
2     char        identifier[20];
3     unsigned int worker;
4     double      hash;
5     double      value;
6 };

```

Quadro 7.5 – Estrutura que representa as informações a serem enviadas de uma árvore

A descrição do tipo para o MPI utilizar na transferências de dados *tree_info* é apresentado no Quadro 7.6. Na primeira linha é criada a variável que conterà este novo tipo. Na linha número 2 é informado os tipos que esta estrutura contém e na linha 3 a quantidade de repetições de cada um destes tipos. Nota-se que a quantidade de caracteres é 20, pois o atributo *identifier* da estrutura *tree_info* é um vetor de 20 caracteres. Na quarta linha é informado a posição de cada elemento na estrutura calculando-se a quantidade de *bytes* ocupada por cada elemento anterior e na quinta linha o tipo utilizado pelo MPI, *TREE_INFO_MPI*, é criado informado-se os dados anteriormente descritos.

```

1 MPI_Datatype TREE_INFO_MPI;
2 MPI_Datatype types[4] = {MPI_CHAR,
                           MPI_UNSIGNED,
                           MPI_DOUBLE,
                           MPI_DOUBLE};

3 int blocklengths[4] = {20, 1, 1, 1};
4 MPI_Aint displacements[4] = 0,
        sizeof(char) * 20,
        (sizeof(char) * 20) + sizeof(unsigned int),
        (sizeof(char) * 20) + sizeof(unsigned int)
        + sizeof(double);

5 MPI_Type_create_struct(4, blocklengths, displacements, types, &TREE_INFO_MPI);

```

Quadro 7.6 – Tipo TREE_INFO_MPI que representa ao MPI o tipo *tree_info_t*

5					
taxon_1	0.00	1.00	1.80	2.60	2.80
taxon_2	1.00	0.00	2.20	3.00	3.20
taxon_3	1.80	2.20	0.00	1.80	2.00
taxon_4	2.60	3.00	1.80	0.00	1.30
taxon_5	2.80	3.20	2.00	1.30	0.00

Quadro 7.7 – Matriz de distâncias do pacote PHYLIP

7.2.3.3 Arquivos de entrada e saída

Um dos objetivos deste trabalho é implementar um algoritmo distribuído no pacote PHYLIP. Conforme descrito na Seção 7.2.1, o código fonte dos softwares deste pacote são complexos e pouco comentados. O tempo proporcionado para sua execução deste trabalho foi o suficiente apenas para compreender o funcionamento dos algoritmos ali implementados e não de toda as suas implementações. Desta forma, as implementações ocorreram totalmente separadas dos softwares do pacote PHYLIP.

Para permitir a interoperabilidade deste software implementado com o pacote PHYLIP foram adotados os formatos de arquivos, tanto para a matriz contendo os dados, como para a representação da árvore inferida. O formato de arquivo das matrizes de distâncias do pacote *phylip* é composto por um número inteiro na primeira contendo a quantidade de táxons matriz, seguido pelas linhas que contem a identificação do táxon e as sua distância entre os demais elementos. O Quadro 7.7 apresenta as distâncias dos táxon no formato do pacote PHYLIP utilizados na inferência da árvore exibida na Figura 4.4.

As árvores são armazenadas num formato de texto, onde cada nó interno inicia

```
(taxon_2:0.70000,((taxon_5:0.75000,taxon_4:0.55000):0.75000,
taxon_3:0.50000):1.00000,taxon_1:0.30000);
```

Quadro 7.8 – Exemplo de árvore num formato de texto

com um parênteses em seguida estão seus nós filhos separados por vírgula e o nó interno é finalizado com outro parênteses. Os nós filhos podem ser outros nós internos ou nós táxons, contendo o seu identificado , dois pontos e a distância relativa ao nó que está conectado. A representação em texto da árvore da Figura 4.4 é exibida no Quadro 7.8.

7.2.3.4 Execução da implementação do algoritmo paralelo

Para execução da implementação do algoritmo paralelo fruto deste trabalho, deve-se ter instalado no computador um ambiente de execução do padrão MPI e caso deseje-se executar num ambiente distribuído, os computadores devem ser configurados previamente para execução de softwares que utilizem o padrão MPI. Informações sobre obtenção, instalação e configuração do ambiente podem ser obtidas em MPICH2... (2006).

O nome dado ao software implementado é *dleastsquare*, *d* de distributed e *least-square* do método utilizado. O *dleastquares* tem os seguintes parâmetros para execução:

- a) *-f nome_do_arquivo*: informa o arquivo de entrada contendo a matriz de distâncias;
- b) *-o nome_do_arquivo*: informa o arquivo onde a árvores resultante será escrita;
- c) *-s número*: número de sementes geradas e enviadas aos processos trabalhadores;
- d) *-m número*: número máximo de árvores mantidas em cada processo trabalhador por iteração;
- e) *-n número*: número mínimo de árvores mantidas em cada processo trabalhador por iteração.

Para executá-lo, deve-se utilizar o *mpirun*, que é um software especificado no padrão MPI para execução dos softwares neste ambiente distribuído. Por exemplo, para executar o *dleastquares* no ambiente com 4 processos, sendo 1 o agendador e 3 trabalhadores, com o número máximo de 200 e mínimo de 50 árvores por processo trabalhador e gerando 21 sementes no início, deve executar: *mpirun -np 4 dleastsquare -m 200 -n 50 -s 21* .

A acréscimo do número de nós executando o *dleastsquare* não torna a sua execução mais rápida, porém aumenta a quantidade de árvores geradas e conseqüentemente o conjunto total de possíveis árvores para a busca da melhor. Caso deseje-se aumentar o número de nós e também ter um ganho no tempo, recomenda-se diminuir o número mínimo e o

número máximo de árvores mantidas por iteração. O porquê desta questão, pode ser observado no seguinte exemplo: com dois processos trabalhadores mantendo duzentas árvores por iteração, tem-se um espaço de quatrocentas árvores, caso sejam utilizados quatro processos trabalhadores, tem-se oitocentas árvores. Desta forma, pode-se diminuir o número de árvores mantidas para cinquenta, que o conjunto de árvores voltará a ser quatrocentos. Desta forma, diminuída a quantidade de árvores mantidas em cada iteração é conseqüentemente, minimizado o tempo de processamento.

7.3 RESULTADOS

Para testar o tempo de execução da implementação do algoritmo desenvolvido, chamada de *dleastquares*, foram geradas oito matrizes com distâncias hipotéticas, cada uma contendo de 10 a 80 táxons e suas respectivas distâncias. Foram executados o software *kitsch* e o *dleastquares* e medido os seus tempos de execução.

O software *kitsch* foi executado num computador *Intel Pentium 4* com 1 *gigabyte* de memória ram e o *dleastquares* num *cluster* contendo 5 destes computadores. O *dleastquares* foi executado utilizando como opções: 4 sementes iniciais por nó, o mínimo de 20 e o máximo de 40 táxon por iteração. Desta forma, a cada iteração, são mantidos no máximo 40 no conjunto de árvores de cada nó.

A Figura 7.9 apresenta os resultados destas execuções. Nela percebe-se que até aproximadamente uma matriz de 50 táxons, é vantajosa a utilização do software não distribuído e a partir das matrizes com 50 táxins, o *dleastquares* apresenta vantagem no tempo de execução.

O *dleastquares* possui pior tempo de execução com matrizes de distâncias menores porque há principalmente o custo de comunicação e sincronismo entre os processos, que o *kitsch* não possui. Nota-se que o tempo de execução do *kitsch* cresce em maior quantidade, isto porque o *dleastquares* controla a quantidade de árvores mantidas a cada iteração e com isto, a largura da pesquisa da melhor árvore e o tempo de execução. Sendo que são mantidas uma quantidade constante de árvores, o acréscimo do tempo se dá pelo maior número de possíveis posições para adicionar um novo táxon quando utiliza-se matrizes com número elevado de táxons.

O *dleastquares* possui um custo para o sincronismo entre os processos. Quando um processo finaliza sua iteração, ele iniciará a nova iteração após todos os outros processos terem enviados os dados da iteração atual ao agendador e este calcular e enviar as árvores que devem ser eliminadas. Desta forma, se tem-se um nó no *cluster* mais veloz que os demais, provavelmente sua capacidade será sub-utilizada. Um modo de resolver esta

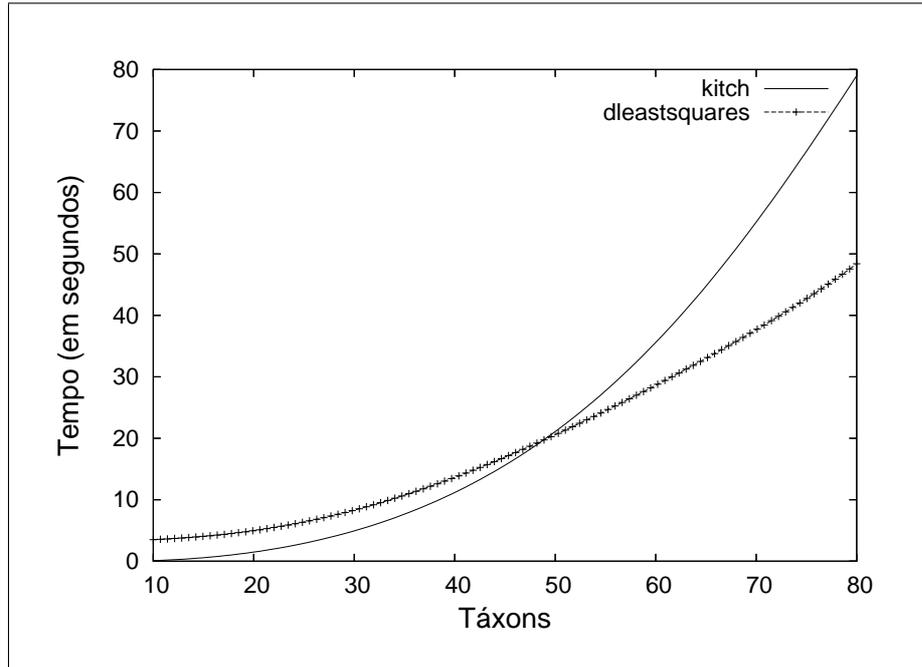


Figura 7.9 – Tempos das execuções

questão é a implementação de um balanceamento de carga, que é proposto como extensão deste trabalho.

É importante ressaltar que o *dleastsquares* não encontra a melhor árvore. A qualidade da otimização dos comprimentos dos ramos efetuada pelo *dleastsquares* é inferior as do *kitch*, porém as topologia das árvores retornadas são bem similares as topologias das árvores ótimas inferidas pelo *kitch*. Assim, a vantagem do *dleastsquares* é apresentar árvores num menor tempo, enquanto o *kitch* apresenta melhores árvores.

7.3.1 Dificuldades encontradas

O desenvolvimento do algoritmo original proposto por Felsenstein (1997) tomou grande parte do tempo da implementação. O artigo apenas descreve as fórmulas matemáticas utilizadas para se chegar aos valores e, superficialmente, o funcionamento do algoritmo. Desta forma, a compreensão deste algoritmo e suas etapas internas foram os primeiros desafios a ser superados. Uma das etapas internas, que se constitui no re-manejamento dos comprimento das distâncias dos ramos, ainda não esta completamente esclarecida, necessitando de maiores pesquisas.

Na implementação, houve dificuldade principalmente com a depuração do algoritmo paralelo e com a verificação da ocorrência de *deadlocks*. A depurando através do GDB é prejudicada porque os resultados dependem de vários processos executados ao mesmo tempo, tornando-a complexa. Para a maioria dos casos, a depuração ocorreu utilizando

saídas de texto informando o estado atual de cada um dos processos, exibindo assim, possíveis pontos onde estão os erros de programação.

8 CONCLUSÃO

Dentre os objetivos apresentados neste estudo, o primeiro constitui-se na proposta de um algoritmo para inferência de árvores filogenéticas executado num ambiente distribuído. A redução do tempo da inferência filogenética, a qual acreditava-se ser a etapa mais demorada do *workflow*, levaria a diminuição do tempo de execução deste. Com a verificação dos tempos das etapas do *workflow*, observou-se que esta, na realidade, representa uma parte insignificante do tempo total. Buscou-se então, a partir desta constatação, a otimização dos tempos das demais etapas.

A otimização do *workflow* e as execuções de várias de suas etapas em ambiente distribuído, resultando numa significativa melhora no tempo de execução e permitindo que este tempo seja mais reduzido aumentando o número de nós no *cluster*, constituiu-se na primeira etapa do trabalho. Estas otimizações estão sendo utilizadas nas pesquisas de genômica comparativa do DBBM/Instituto Oswaldo Cruz/FIOCRUZ. Uma das otimizações, o software agendador para múltiplas execuções do software *compass* em ambiente distribuído, é um software que pode ser estendido para usos em outras ocasiões que deseja-se executar diversas instâncias de um software num ambiente distribuído, delegando a execução de cada instância num nó diferente.

Através desse algoritmo, inédito na literatura, especificado e implementado neste trabalho, é possível inferir árvores filogenética especificando parâmetros como a quantidade de árvores analisadas a cada iteração e limites para eliminação de árvores. Mesmo que as árvores obtidas não possuam o LS igual ou inferior as inferidas por outros softwares, elas apresentam muita semelhança nos comprimentos dos ramos e na topologia.

Tal algoritmo permite especificar a largura da busca pela melhor árvores, para que o usuário possa escolher entre melhor qualidade ou menor tempo. Desta forma, dependendo dos parâmetros utilizados na execução, a implementação do algoritmo paralelo é mais veloz que os softwares *fitch* e *kitsch*. Caso o usuário disponha de um cluster, ele poderá utilizar esta capacidade computacional para pesquisar num conjunto maior de árvores ou diminuir o tempo de execução.

Na hipótese de um pesquisador não pretender utilizar a implementação proposta, ou o algoritmo totalmente, as heurísticas mostraram-se eficazes para serem utilizados em outros algoritmos. Exemplificando a questão: a seleção dos táxons mais próximos pode ser utilizada no início do algoritmo proposto por Felsenstein (1997).

Quanto à implementação do algoritmo num software de reconstrução filogenética do pacote PHYLIP, também um dos objetivos propostos neste trabalho, não houve concretização, devido o motivo abordado na Seção 7.2.1. Mesmo que o software não tenha sido implementado neste pacote, a implementação utiliza os seus formatos de arquivo, permitindo troca de informações entre o software implementado e os softwares do pacote PHYLIP.

Como terceiro objetivo se propôs a substituição do software PAUP* pelo software desenvolvido neste trabalho. Esta substituição não foi efetivada devido ao software desenvolvido não apresenta as melhores árvores se comparado aos softwares do pacote PHYLIP e ao PAUP*. Mesmo assim, caso deseje-se utilizar somente softwares livres, é possível informar ao agendador das execuções múltiplas do *compass* para que ele armazene a matriz de distâncias resultante das comparações no formato do pacote PHYLIP e utilizar seus softwares para a inferência da árvore filogenética.

Desta forma, indiretamente, este trabalho atingiu com sucesso os seus objetivos propostos, com adição à otimização do *workflow*, não prevista inicialmente e o software para execuções múltiplas em ambiente distribuído.

8.1 EXTENSÕES

Em relação ao *workflow*, sua principal extensão é a criação de uma interface gráfica para a utilização de diversos usuários, proporcionando a opção de armazenar as execuções e poder analisá-las em outra ocasião. Atualmente as operações de execução, configuração das seqüências de entradas e armazenamento das execuções para futuras análises são executadas via comandos digitados pelo usuário no terminal, criando dificuldades aos usuários menos familiarizados neste ambiente.

Para o algoritmo, algumas extensões são possíveis, como por exemplo a utilização de algoritmos genéticos para verificar os melhores parâmetros da quantidade de sementes utilizadas, os limites de corte das árvores e as quantidades mínimas e máximas. Estes parâmetros interferem diretamente no tempo e na qualidade da árvore final sendo complexo encontrar uma configuração para cada árvore. Desta forma, a utilização de algoritmos genéticos facilitaria o encontro das configurações ideais para cada matriz de distâncias utilizada como entrada.

Uma outra extensão, diz respeito à pesquisa e implementação de técnicas para otimizar o valor LS das árvores. Conforme apresentado, um dos problemas do algoritmo proposto é a qualidade inferior das árvores geradas se comparado aos softwares já existentes. A pesquisa e implementação de otimização deste valor deve proporcionar a melhora

do LS das árvores utilizando o ambiente distribuído e preferencialmente não sobrecarregando o software com as otimizações, para não influenciar negativamente no tempo de execução.

O algoritmo não possui um balanceamento de carga entre os processos, ocorrendo que enquanto alguns processos estão calculando os dados, outros já finalizaram a sua iteração e estão aguardando. O balanceamento de carga entre os processos é um tema complexo nos sistemas distribuídos e a utilização dele neste algoritmo é uma opção de pesquisa para extensão deste trabalho.

Um extensão proposta é analisar a viabilidade, e se necessário, fazer modificações, para implementar o algoritmo em sistemas *Grid*. Implementar o algoritmo sem a utilização do padrão MPI, mas utilizando *threads* para a utilização em computadores multiprocessados.

Por fim, tratando-se este algoritmo como uma busca, pode-se verificar sua usabilidade em outras áreas, como a inteligência artificial, como exemplo, o algoritmo *MinMax*.

REFERÊNCIAS BIBLIOGRÁFICAS

- ALBERTS, Bruce et al. **Biologia molecular da célula**. 2. ed. Porto Alegre: Artmed, 2004. 1463 p.
- ALBRECHT, Felipe Fernandes. Técnicas para comparação e visualização de similaridades entre seqüências genéticas. In: SEMINÁRIO DE COMPUTAÇÃO, 14., 2005, Blumenau. **Anais...** Blumenau: Universidade Regional de Blumenau, 2005. p. 139–149.
- ALTSCHUL, F. et al. A basic local alignment search tool. **Jornal of Molecular Biology**, [S.I.], v. 215, n. 3, p. 403–410, out. 1995.
- BEOWULF.ORG: the Beowulf cluster site. San Francisco: Beowulf.org, 2004. Disponível em: <www.beowulf.org>. Acesso em: 26 mar. 2006.
- BERMAN, H.M; HENRICK, K.; NAKUMA, H. Announcing the worldwide protein data bank. **Nature Structural Biology**, [S.I.], v. 10, n. 12, p. 980, dez. 2002.
- BERNARDES, Juliana Silva. **Detecção de homologias distantes utilizando HMMs e informações estruturais**. 2004. 116 f. Dissertação (Mestrando em ciências em engenharia de sistemas e computação) — COPPE/Universidade Federal do Rio de Janeiro, Rio de Janeiro.
- BURNS, Greg; DAOUD, Raja; VAIGL, James. Lam: an open cluster environment for MPI. In: SUPOERCOMPUTING SYMPOSIUM'94, Toronto. **Proceedings...** Toronto: University of Toronto, 1994. p. 379–386.
- CAVALLI-SFORZA, L.L; EDWARDS, A.W. Phylogenetic analysis: models and estimation procedures. **Am. J. Human Genetics**, [S.I.], v. 19, n. 3, p. 233–257, 1967.
- CHENNA, Ramu et al. Multiple sequence alignment with the clustal series of programs. **Nucleic Acids Res**, Illkirch, v. 31, n. 13, p. 3497–3500, mar. 2003.
- COLLINS, F. S. et al. New goals for the u.s. human genome project: 1998-2003. **Science**, [S.I.], v. 282, n. 5389, p. 682, oct. 1998.
- CORMEN, Thomas H. et al. **Introduction to Algorithms**. 2. ed. Massachusetts: MIT Press, 2001. 1180 p.
- CURL and libcurl. [s.l.]: [s.n.], 2006. Disponível em: <<http://curl.haxx.se/>>. Acesso em: 20 Out 2006.
- DARLING, A.; CAREY, L.; FENG, W. The desing, implementation, and evaluation of mpiblast. In: INTERNATIONAL CONFERENCE ON LINUX CLUSTERS: THE HPC REVOLUTION 2003 IN CONJUNCTION WITH THE CLUSTERWORLD CONFERENCE & EXPO, 4., 2003, San Jose, California, USA. **Proceddings...** San Jose, CA: LA-UR, 2003. p. 20–34.
- DARWIN, Charles. **The origin of species**. New York: Martin Claret, 1859. 629 p.

DURBIN, Richard et al. **Biological sequence analysis**: probabilistic models of proteins and nucleic acids. Cambridge, UK: Cambridge University Press, 1998. 356 p.

FELSENSTEIN, Joseph. An alternating least squarers approach to inferring phylogenies from pairwise distances. **Systematic Biology**, [S.I.], v. 46, n. 1, p. 101–111, mar. 1997.

_____. **Inferring phylogenies**. Massachusetts: Sinauer Associates, 2003. 580 p.

_____. **PHYLIP programs and documentation**. Washington, 2004. Disponível em: <<http://evolution.genetics.washington.edu/phylip/phylip.html>>. Acesso em: 26 ago. 2006.

_____. **PHYLIP (phylogeny inference package) version 3.6**. Washington, 2005. Disponível em: <<http://evolution.genetics.washington.edu/phylip/getme.html>>. Acesso em: 26 ago. 2006.

FITCH, W. M.; MARGOLIASH, E. Construction of phylogenetic trees. **Science**, [S.I.], v. 760, n. 157, p. 279, Jan. 1967.

FOUNDATION, Free Software. **GNU gprof**: the gnu profiler. Boston: [s.n.], 1997. Disponível em: <http://www.gnu.org/software/binutils/manual/gprof-2.9.1/html_mono/gprof.html>. Acesso em: 16 out 2006.

_____. **GCC, the GNU compiler collection**. Boston: [s.n.], 2006. Disponível em: <<http://gcc.gnu.org/>>. Acesso em: 31 out 2006.

_____. **GDB: the GNU project debugger**. 2006. Disponível em: <<http://www.gnu.org/software/gdb/>>. Acesso em: 31 out 2006.

GABRIEL, Edgar et al. Open MPI: goals, concept, and design of a next generation MPI implementation. In: EUROPEAN PVM/MPI USERS GROUP MEETING, 11., 2004, Budapest. **Proceedings...** Budapest, Hungary, 2004. p. 97–104.

GENBANK. Bethesda: [s.n.], 2006. Disponível em: <<http://www.ncbi.nlm.nih.gov/Genbank/>>. Acesso em: 28 jul 2006.

LI, Kuo-Bin. Clustalw-mpi: clustalw analysis using distributed and parallel computing. **Bioinformatics Application Note**, [S.I.], v. 19, n. 12, p. 1585–1586, mar. 2003.

LOESCH, Claudio; HEIN, Nelson. **Pesquisa operacional**: fundamentos e modelos. Blumenau: Editora da FURB, 1999. 269 p.

LYNCH, Nancy A. **Distributed algorithms**. San Francisco: Morgan Kaufmann, 1997. 872 p.

MESSAGE Passing Interface. Chicago: [s.n.], 2006. Disponível em: <www-unix.mcs.anl.gov/mpi/>. Acesso em: 26 mar. 2006.

MOUNT, David W. **Bioinformatics**: sequence and genome analysis. 2. ed. New York: Cold Spring Harbor Laboratory Press, 2004. 692 p.

MPICH2 home page. Chicago: [s.n.], 2006. Disponível em: <www-unix.mcs.anl.gov/mpi/mpich/>. Acesso em: 26 mar. 2006.

NEI, Masatoshi; KUMAR, Sudhir. **Molecular evolution and phylogenetics**. Oxford: Oxford University Press, 2000. 352 p.

PARK, S. Y. et al. A resolution crystal structures of human haemoglobin in the oxy, deoxy and carbonmonoxy forms. **J. Mol. Biol.**, Yokohama, Japan, p. 690–701, may 2006.

SADREYEV, R.; GRISHIN, N. Compass: a tool for comparison of multiple protein alignments with assessment of statistical significance. **Journal of Molecular Biology**, Dallas, v. 326, n. 1, p. 317–336, fev. 2003.

SARKAR, Indra Neil. Phylogenetics in the modern era. **Journal of Biomedical Informatics**, San Diego, v. 39, n. 1, p. 3–5, fev. 2006.

SLOAN, Joseph D. **High performance Linux clusters :with OSCAR, Rocks, openMosix, and MPI**. Beijing: O'Reilly, 2004. 350 p.

SPIEGEL, Murray R. **Estatística**. 3. ed. São Paulo: Makron Books, 1993.

STAMATAKIS, Alexandros. **Distributed and parallel algorithms and systems for inference of huge phylogenetic trees bases on the maximum likelihood method**. 2004. 132 p. Tese (Ph.D. on Computer Science) — Technischen Universität München, München.

STERLING, Thomas (Ed.). **Beowulf cluster computing with Linux**. Cambridge, Massachusetts: The Mit Press, 2002. 496 p.

STRACHAN, Tom; READ, Andrey P. **Genética molecular humana**. 2. ed. Porto Alegre: Artmed, 2002. 576 p.

SWOFFORD, D. L. **PAUP***: phylogenetic analysis using parsimony (*and other methods). 4. ed. Sunderland, Massachusetts: Sinauer Associates, 2004.

THEOBALD, Douglas L.; WUTTKE, Deborah S. Divergent evolution within protein superfolds inferred from profile-based phylogenetics. **Journal of Molecular Biology**, [S.I.], v. 354, n. 3, p. 722–737, dec. 2005.

WHEELER, D.L. et al. Database resources of the national center for biotechnology information: update. **Nucleic Acids Res**, [s.l.], v. 32, p. 35–40, 2004.

WYHE, John van. **The complete work of Charles Darwin online**. Cambridge: [s.n.], 2006. Disponível em: <darwin-online.org.uk/>. Acesso em: 16 dez 2006.