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

Algoritmos genéticos para filogenia viva com matriz de características

Rafael Lins Fernandes

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Informática

Orientadora
Prof.ª Dr.ª Maria Emília M. T. Walter

Brasília
2018

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Mestrado em Informática

Coordenador: Prof. Dr. Bruno Luiggi Macchiavello Espinoza

Banca examinadora composta por:

Prof.ª Dr.ª Maria Emília M. T. Walter (Orientadora) — CIC/UnB
Prof. Dr. Guilherme Pimentel Telles — IC/Unicamp
Prof. Dr. Li Weigang — CIC/UnB

CIP — Catalogação Internacional na Publicação

Fernandes, Rafael Lins.

Algoritmos genéticos para filogenia viva com matriz de características / Rafael Lins Fernandes. Brasília : UnB, 2018.
191 p. : il. ; 29,5 cm.

Dissertação (Mestrado) — Universidade de Brasília, Brasília, 2018.

1. algoritmos genéticos, 2. filogenia, 3. filogenia viva, 4. filogenia grande, 5. filogenia pequena, 6. algoritmo de Sankoff, 7. algoritmo de Fitch, 8. Bioinformática

CDU 004.4

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

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

Algoritmos genéticos para filogenia viva com matriz de características

Rafael Lins Fernandes

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Informática

Prof.ª Dr.ª Maria Emília M. T. Walter (Orientadora)
CIC/UnB

Prof. Dr. Guilherme Pimentel Telles
IC/Unicamp

Prof. Dr. Li Weigang
CIC/UnB

Prof. Dr. Bruno Luiggi Macchiavello Espinoza
Coordenador do Mestrado em Informática

Brasília, 10 de abril de 2018

Agradecimentos

Agradeço a todos os professores do departamento de Computação da UnB, em especial à professora orientadora Maria Emília Walter por ter me acompanhado e ajudado desde o início do curso. Agradeço ao colega Artur Queiroz pelo apoio no estudo de caso 2 desse trabalho.

Agradeço a minha família, que esteve sempre apoiando meus estudos, e à Katarina por ter me acompanhado e incentivado ao longo do tempo de realização desse trabalho.

Resumo

O conceito de filogenia viva generaliza o de árvore filogenética, admitindo organismos vivos como ancestrais. Os problemas da filogenia pequena e da filogenia grande são naturalmente estendidos para filogenia viva. Neste trabalho, inicialmente, introduzimos modificações nos algoritmos de Fitch e de Sankoff para resolver o problema da filogenia viva pequena. Em seguida, propusemos um algoritmo genético que usa os algoritmos de Fitch (ou Sankoff) para resolver o problema da filogenia viva grande, com matriz de características como entrada. A partir do método proposto, desenvolvemos experimentos com dados de alinhamentos múltiplos de vírus H1N1 e H2N3, de diferentes países (Estados Unidos, Rússia, Coréia do Sul, Taiwan, Itália e China). As filogenias obtidas mostraram um agrupamento de vírus de regiões próximas. Quando essas filogenias vivas foram comparadas com uma filogenia gerada pelo Phylip, observamos agrupamentos muito semelhantes. Por fim, executamos o método usando um alinhamento múltiplo de diversas leituras da região *env* do vírus HIV de um mesmo paciente, por um certo período de tempo, variando o número de nós vivos. Comparamos essas filogenias vivas com uma filogenia gerada pelo PAUP, observando que os agrupamentos foram coerentes com períodos próximos de coleta.

Palavras-chave: algoritmos genéticos, filogenia, filogenia viva, filogenia grande, filogenia pequena, algoritmo de Sankoff, algoritmo de Fitch, Bioinformática

Abstract

Live phylogeny trees relate taxonomic units using their similarities over a set of characteristics and generalize phylogeny trees, since they allow the existence of living ancestors. Consequently, small phylogeny problem and large phylogeny problem are naturally extended to live phylogeny. We introduced modifications in Fitch's and Sankoff's algorithms to solve the small live phylogeny problem. Thus, we proposed a genetic algorithm witch uses Fitch (or Sankoff) to solve the large live phylogeny problem, taking a character matrix as an input. Using the proposed method, we developed experiments using multiple alignments of H1N1 and H2N3 viruses, from different countries (United States, Russia, South Korea, Taiwan, Italy and China). The output phylogenies showed clusters representing neighbour regions. These phylogenies showed similar clusters, when compared to a phylogeny generated with Philyp. Finally, we applied the algorithm to a multiple alignment obtained by reads of the *env* region of the HIV virus, of a same patient, during a period of time, with variations in the number of live nodes. We presented three different phylogenies with different numbers of live ancestors, having compared them to a tree generated by PAUP. We observed that the clusters generated by our phylogeny were compatible with close periods of data capture.

Keywords: genetic algorithms, phylogeny, large phylogeny, small phylogeny, live phylogeny, Sankoff's algorithm, Fitch's algorithm, Bioinformatics

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A construção de árvores filogenéticas é um problema bastante estudado em Biologia [3, 27, 36]. Tal construção é utilizada para tentar entender as relações evolutivas entre espécies, ou proteínas, através das eras e também de como as espécies atuais se relacionam através de seus ancestrais comuns. Isto pode ser realizado com a construção de árvores filogenéticas, nas quais os nós internos representam ancestrais hipotéticos e as folhas, espécies atuais, conforme mostra a Figura 1.1.

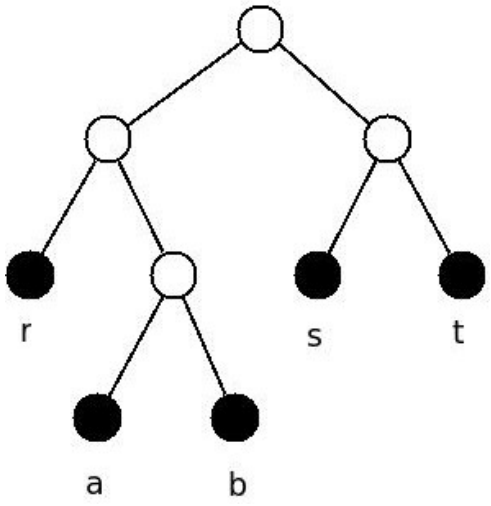


Figura 1.1: Árvore filogenética, com os objetos r, a, b, s e t nas folhas, representando espécies vivas atualmente.

Desde 1950, vários métodos, baseados na intuição e experiência de pesquisadores, foram utilizados para construir árvores filogenéticas entre várias espécies de organismos [36]. Gradualmente, os formalismos matemáticos foram sendo introduzidos e hoje são conhecidos diversos algoritmos para construção de árvores filogenéticas.

Na Biologia Evolutiva [25], podemos construir árvores filogenéticas para espécies, populações, gênero, ou outras tantas unidades taxonômicas. Os principais aspectos de árvores em que os cientistas estão interessados são: a topologia, isto é, como os nós internos conectam-se uns aos outros em relação às folhas; existência ou não de uma raiz, que pode determinar a relações entre ancestrais comuns dos objetos; e a distância existente entre um par de nós, que pode ser determinada quando as arestas possuem pesos. Esses

três aspectos citados podem ser usados como uma estimativa de distância evolutiva entre os organismos (mostrados nas folhas). No caso da Biologia Evolutiva, as folhas são consideradas espécies vivas, e os nós internos são considerados ancestrais hipotéticos [36].

Existem duas categorias principais de classificação dos dados de entrada de uma árvore filogenética [36]:

1. Características discretas, que podem possuir um número finito de estados. Alguns exemplos de estados seriam tipo de bico, número de dedos, presença ou não de alguma característica molecular. Os dados são inseridos em uma **matriz de características** de objetos x características;

2. Dados numéricos, chamados de distância entre objetos, e construídos em uma **matriz de distâncias** de objetos $\times$ objetos.

Embora a filogenia seja uma área de pesquisa antiga, com muitos problemas e métodos propostos [34], existem dois problemas importantes que são bastante estudados [16]:

- **Problema da filogenia grande**: dados um conjunto de objetos de entrada e uma matriz de características, encontre uma filogenia e uma rotulação dos nós internos de tal forma que os custos associados aos passos evolutivos seja minimizado;

- **Problema da filogenia pequena**: dados um conjunto de objetos de entrada, uma filogenia e uma matriz de características, encontre a rotulação de nós internos que minimize os custos associados aos passos evolutivos.

Telles et al [44] generalizam o problema da filogenia, admitindo a existência de ancestrais vivos como nós internos (Figura 1.2). Este problema é adequado tanto para o estudo de espécies que evoluem rapidamente, como vírus, quanto para a construção de filogenias para objetos não biológicos, como imagens, documentos e registros de banco de dados.

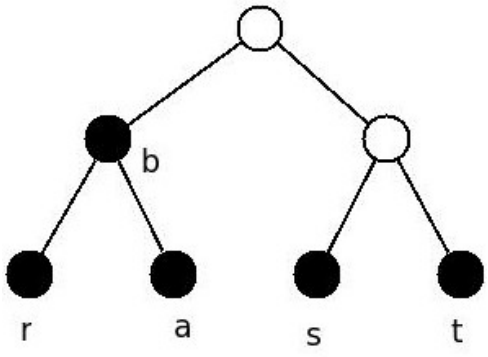


Figura 1.2: Árvore filogenética com ancestral vivo b e demais espécies vivas nas folhas (r, a, s e t).

Analogamente aos problemas da filogenia citados por Gupta et al. [16], temos o problema da filogenia viva grande e o problema da filogenia viva pequena:

- **Problema da filogenia viva grande**: dados um conjunto de objetos de entrada, que possam ser ancestrais vivos, e uma matriz de características, encontre uma filogenia e uma rotulação dos nós internos de tal forma que minimize os custos associados aos passos evolutivos necessários;

- **Problema da filogenia viva pequena**: dados um conjunto de objetos de entrada, que possam ser ancestrais vivos, uma filogenia e uma matriz de características, encontre a rotulação de nós internos que minimize os custos associados aos passos evolutivos necessários.

Por outro lado, algoritmos genéticos são uma heurística inspirada na evolução de organismos biológicos, e realiza operações similares às que ocorrem na natureza, como mutações, seleção natural ao longo de gerações e recombinações [9]. Inicialmente, uma população de soluções possíveis para o problema é gerada, e uma função de *fitness* avalia a qualidade de cada candidato. Então, o algoritmo entra em um laço, que corresponde a uma geração, realizando certas operações em sua população. Essas operações, chamadas de mutações e recombinações, alteram um indivíduo da população de acordo com a sua codificação, que modela características de cada indivíduo, podendo ser um número binário, inteiro, ou uma outra estrutura de dados, dependendo da aplicação [22, 23]. Ao final de cada geração, é realizada seleção natural[1] para compor a população da próxima geração [23].

Algoritmos genéticos são utilizados para modelar a evolução biológica em Inteligência Artificial [23]. Problemas como o da filogenia grande podem ser modelados por um método que permita que soluções sejam criadas e melhoradas, avaliando toda sua população em busca de uma solução que expresse da melhor forma possível as relações evolutivas entre os organismos estudados.

1.1 Justificativa

Como o problema da filogenia viva proposto por Telles et al [44] é novo e ainda pouco explorado, a proposição de novos métodos ou algoritmos usando metodologias diferentes, eficientes e disponíveis publicamente, pode contribuir para seu uso em diferentes aplicações. Em particular, algoritmos genéticos constituem-se em uma técnica eficaz para resolver problemas considerados difíceis e são utilizados para construções de filogenias por algoritmos como GAML [21] e GARLI [10].

1.2 Problema

Não existem softwares que permitam construir filogenias vivas, em particular, não existe método para resolver o problema da filogenia viva usando como entrada uma matriz de características.

1.3 Objetivos

O objetivo principal deste trabalho é construir uma filogenia viva, tendo como entrada uma matriz de características, usando algoritmos genéticos.

[1] A seleção natural de um algoritmo genético avalia seus indivíduos de acordo com uma função de *fitness* definida, de modo a selecionar apenas os melhores candidatos que irão compor uma solução do problema para a próxima geração. As estratégias utilizadas para essa seleção são variadas e serão discutidas posteriormente.

Os objetivos específicos são:

- propor e implementar algoritmos para o problema da filogenia viva pequena;
- propor e implementar algoritmos genéticos para o problema da filogenia viva grande usando os algoritmos para o problema da filogenia pequena;
- executar o algoritmo proposto e analisar e discutir os resultados.

1.4 Descrição dos capítulos

No Capítulo 2, iremos detalhar os principais problemas relacionados à filogenia e alguns métodos de construção de filogenia. Aqui, falaremos sobre os dois tipos de entradas diferentes, matriz de características e matriz de distâncias, e os problemas da filogenia grande e da filogenia pequena, bem como daqueles relacionados à filogenia viva. Em seguida, apresentaremos algoritmos genéticos, que serão usados para desenvolver o algoritmo para construção de filogenia viva.

No Capítulo 3, iremos apresentar uma revisão de literatura focando nos algoritmos de construção de filogenia já conhecidos e em alguns softwares comumente utilizados.

No Capítulo 4, iremos apresentar o algoritmo genético para resolver o problema de construção de uma filogenia, além de apresentarmos uma proposta para resolver o problema de construção de uma filogenia viva.

No Capítulo 5, iremos apresentar os resultados obtidos com a execução do software, além de descrever as entradas utilizadas e mostrar como foi feita a calibragem de parâmetros do software. Discutiremos ainda dois estudos de caso realizados com dados biológicos.

No Capítulo 6, concluiremos o trabalho, e vamos sugerir trabalhos futuros.

Capítulo 2

Filogenia viva e algoritmos genéticos

Neste capítulo, descrevemos conceitos básicos de filogenia e de algoritmos genéticos, necessários ao entendimento deste trabalho. Na Seção 2.1, apresentaremos a definição de filogenia, suas aplicações e métodos de construção de árvores filogenéticas baseados em dois tipos de entrada possíveis: matriz de características e matriz de distâncias; e algoritmos usados para construção de filogenias de acordo com os dois tipos de dados de entrada. Descreveremos também o problema da filogenia viva. Na Seção 2.2, apresentaremos inicialmente a motivação biológica que permitiu a criação de algoritmos genéticos, uma heurística utilizada para resolução de problemas computacionais que exigem busca exaustiva de um espaço de soluções. Em seguida, descrevemos algumas operações utilizadas em algoritmos genéticos.

2.1 Filogenia

O DNA (ácido desoxirribonucleico) armazena informações necessárias para produzir proteínas. Por meio da reprodução sexuada, o DNA é transmitido de uma geração para seus descendentes. Durante a replicação de genes, podem haver "erros", conhecidos por mutações. Através de várias gerações, com mutações ocorrendo entre gerações e ocasionando pequenas mudanças em seus descendentes, espécies diferentes podem surgir. Esse fato justifica a necessidade de estudar relações evolutivas entre espécies vivas atualmente.

Filogenia é o estudo das relações evolutivas entre diferentes espécies e a reconstrução da história evolucionária de um conjunto de indivíduos [36]. Geralmente, a filogenia é representada por uma árvore rotulada com os indivíduos nas folhas. Indivíduos próximos evolutivamente serão representadas com arestas indicando quão próximos estão, e serão conectados por um ancestral comum. O objetivo da análise filogenética é explicitar as relações entre seus indivíduos e o tamanho das arestas.

No exemplo da Figura 2.1, temos uma árvore filogenética indicando a relação entre diferentes espécies (nossos indivíduos, nesse caso) de lagartos. Nessa figura, as espécies em estudos estão nas folhas, representadas na dimensão horizontal, e os nós internos representam ancestrais hipotéticos em um tempo remoto, na dimensão vertical. Estes nós internos devem ser interpretados como um ancestral a partir do qual do qual duas espécies divergiram. O ponto mais acima de todos, a raiz da árvore, pode ser interpretado como o ancestral comum do qual todas as espécies derivaram ao longo do tempo da evolução.

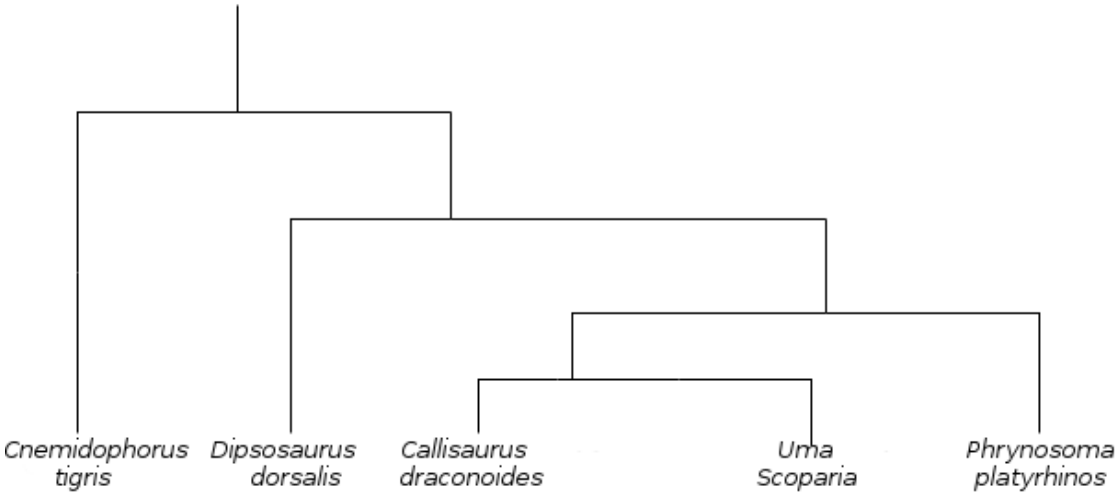


Figura 2.1: Filogenia de lagartos, adaptado de [40]

Além da compreensão da história evolucionária da vida, a filogenia tem diversos outros usos, por exemplo, ajudar na predição de estruturas de proteínas e RNA, ajudar a explicar a expressão gênica e relacionar estruturas para criação de novas drogas, entre outros [40].

Se possuirmos um sequenciamento genômico de diversos organismos e construirmos uma árvore filogenética, é bastante provável que organismos com sequências similares derivem de um mesmo ancestral comum ou se encontrem a poucos nós internos, uns dos outros, na árvore. A reconstrução dessas árvores é o problema central da filogenia. Seu maior propósito é predizer ou estimar qual seria a filogenia a partir de um conjunto de espécies vivas atualmente, denominadas de objetos.

Existem dois tipos de entradas que podem ser utilizadas para a reconstrução de uma árvore filogenética: baseadas em **características** ou baseadas em **distância**. Ambos os métodos e a maneira de construção das árvores filogenéticas serão abordados a seguir.

2.1.1 Matriz de características

Para a construção de uma matriz de características, devemos observar que:

- As características utilizadas devem ter algum significado no contexto da construção da árvore;
- Cada característica deve ser identificada de maneira independente das demais;
- Todos os estados observados de uma característica devem ter evoluído de algum estado original em um ancestral comum;
- Os nós internos são chamados de objetos ancestrais hipotéticos. Os algoritmos definem estados específicos para estes objetos, mas isso não significa que eles tenham algum significado biológico. O objetivo final destes ancestrais hipotéticos é agrupar os indivíduos, que serão associados às folhas.

Definimos a matriz de características como uma matriz M com n linhas (objetos) e m colunas (características). M_{ij} denota o estado do objeto i em relação à característica j. Podem haver até r estados distindos em cada característica. Esses estados são representados por números não negativos.

Um exemplo de uma matriz de características pode ser visto na Tabela 2.1. Neste exemplo, lidamos com características binárias, com valores possíveis de 0 ou 1, ou seja, possui ou não possui a característica. Então, nesta tabela, o objeto A, por exemplo, tem as características c_1 e c_2, e não tem c_3, c_4 e c_5.

Tabela 2.1: Exemplo de uma matriz de características, com 5 indivíduos (A, B, C, D, E) e 5 características $(c_1, c_2, c_3, c_4, c_5)$.

	Característica				
Objeto	c_1	c_2	c_3	c_4	c_5
A	1	1	0	0	0
B	0	0	1	0	1
C	1	1	0	0	1
D	0	1	1	1	0
E	1	1	0	0	1

É possível também a construção de uma matriz com r estados distintos. Existem exemplos em que temos que ordenar as características de modo que a árvore só apresente mudanças segundo uma ordem definida de estados. Estes estados podem ter uma ordem linear, por exemplo, 3-1-4-2, ou serem parcialmente ordenados a partir de uma árvore de derivação. Características ordenadas são chamadas na literatura de *características qualitativas*, e as não ordenadas são chamadas de *características cladísticas* [36].

Filogenia Perfeita

Inicialmente, vamos explorar a propriedade da compatibilidade. Dado um conjunto de características que representam um conjunto de espécies, tentaremos encontrar uma árvore filogenética que seja compatível com o máximo de características possível. Para iniciar a discussão do problema da compatibilidade, vamos assumir que a matriz seja binária, isto é, possua características 0 ou 1.

Dada uma característica c, podemos indicar seus valores nas folhas (0 ou 1), e indicar nos nós internos da árvore o estado desta característica. Quando apenas uma mudança para uma certa característica é encontrada nas arestas de uma árvore, é dito que esta árvore é compatível com esta característica. Caso contrário, se existe mais de uma mudança nas arestas da árvore, dizemos que a árvore é incompatível com a característica. Podemos expressar a propriedade da compatibilidade da forma descrita abaixo:

Propriedade de compatibilidade: Se uma característica c é compatível com uma árvore T, então uma aresta (u, v) pode ser identificada em T, de tal maneira que

- Todas as folhas na subárvore de raiz v tem o mesmo estado s para a característica c;
- As outras folhas possuem estado $(1-s)$ (estado complementar) para a característica c.

Baseado no conceito de compatibilidade, o problema da **filogenia perfeita** pode ser definido da seguinte forma [39]:

- Entrada: Dadas n espécies caracterizadas por m características. A entrada é representada na forma de uma matriz binária M com n linhas e m colunas, onde M_{ij} é o estado da característica j para a espécie i;
- M admite uma filogenia perfeita se e somente se existe uma árvore T com raiz e n espécies de tal modo que as m características são compatíveis.

Existem dois problemas baseados em compatibilidade, descritos como:

- **Problema da compatibilidade**: Dadas n espécies caracterizadas por m características binárias, representados por uma matriz binária M, o problema é verificar se este conjunto de espécies admite uma filogenia perfeita. Em outras palavras, precisamos responder se existe uma árvore T tal que todas as m características são compatíveis com a árvore.
- **Problema da filogenia**: Dada a mesma entrada citada no item anterior, o problema é encontrar o conjunto máximo de características que possam admitir uma filogenia perfeita e, caso M não admita uma filogenia perfeita, devemos encontrar um subconjunto máximo que a admita.

Princípio da parcimônia

O princípio da parcimônia é definido como uma regra científica que diz que, se existem duas respostas a um problema ou questão, e se, para que uma delas seja verdade, leis bem estabelecidas da lógica e da ciência precisam ser reescritas, ignoradas ou suspensas, então a resposta mais simples das duas é muito mais provável de ser a correta [39]. Em outras palavras, podemos dizer que a resposta mais simples que explica uma série de observações é preferível que a mais complexa.

Dito isso, a parcimônia é o método mais popular para construção de filogenias baseadas em matriz de característica. A ideia por trás da parcimônia é construir uma filogenia com o menor número possíveis de mutações.

Existem dois grupos de problemas na parcimônia:

- **Parcimônia pequena**: Encontrar a quantidade de parcimônia, dada uma árvore e uma determinada topologia;
- **Parcimônia grande**: Encontrar a árvore com a melhor parcimônia.

O problema da parcimônia pequena pode ser resolvido rapidamente, em tempo polinomial, pelos algoritmos de Fitch [14] e de Sankoff [35]. Ambos os algoritmos serão discutidos no capítulo 4. Sung et al [40] mostraram que o problema da parcimônia grande é um problema NP-difícil, e pode ser aproximado com um algoritmo com tempo polinomial, mas que não necessariamente converge para a melhor árvore (solução ótima).

Um algoritmo para o problema da parcimônia pode ser descrito com os quatro passos seguintes:

1. Construa todas as árvores possíveis;
2. Para cada árvore e para cada coluna da matriz de entrada, conte o número mínimo de mudanças necessárias (mutações);
3. Calcule o escore de cada árvore calculando número total de mudanças necessárias;
4. Escolha a árvore com o menor escore.

O passo 1 explora todo espaço de soluções e para implementá-lo precisaríamos resolver o problema da parcimônia grande. Os passos 2 e 3 representam uma implementação para resolver o problema da parcimônia pequena. Porém, o número total de possibilidades de entrada para o problema da filogenia pequena é muito alto. Com n folhas, a quantidade de árvores binárias e sem raiz que são possíveis chega a $\frac{(2n-5)!}{(n-3)!2^{n-3}}$ [12]. Para ilustrar o aumento dessa quantidade em função do número de folhas, podemos ver na Tabela 2.2 como esse número aumenta consideravelmente, com pequenos aumentos em nossa entrada.

Tabela 2.2: Números de árvores possíveis em função da quantidade de folhas

Objetos (folhas)	Quantidade de árvores
4	3
6	105
8	10.395
10	2.027.025
15	$7 \cdot 10^{12}$
20	$2 \cdot 10^{20}$

Necessitamos de algumas heurísticas que possam encontrar uma solução adequada dentre todas as possibilidades de árvores. Borenstein [6] indica alguns métodos possíveis, como os métodos *hill-climbing* [38], *Nearest-Neighbor Interchange* [11] e algoritmos genéticos [6]. Essas heurísticas retornam uma solução aceitável em um espaço de soluções, mas podem não convergir necessariamente para a solução ótima do problema.

Máxima Verosimilhança

O método da máxima verosimilhança é uma metodologia estatística que tenta estimar quais são os parâmetros de um determinado modelo. Assim, a partir de um conjunto de dados de entrada D, o método busca valores para os parâmetros de um modelo M de tal forma que a probabilidade dos dados amostrados, nossa entrada D, seja a maior possível.

Em filogenia, temos como entrada um conjunto de indivíduos, uma árvore filogenética e uma matriz de características, e podemos modelar M através de parâmetros que representem suas duas principais componentes:

- Uma árvore com raiz, que representa a evolução entre os indivíduos;
- Um parâmetro para cada aresta da árvore de entrada;
- Um modelo de substituição que determine os custos de transição de um estado para outro.

Os modelos de substituição utilizados são basicamente formados por uma matriz Q, que possui os custos de transição de um estado para outro, dentre os estados possíveis da matriz de entrada. Dentre os diversos modelos que podem ser utilizados, incluem-se os de Canveder-Felsenstein [8], Jukes-Cantor [18] e Hasegawa-Kishino-Yano (modelo HKY) [17]. Na maior parte dos modelos, são apresentados custos de transição entre as quatro bases de DNA possíveis, utilizando no cálculo a probabilidade de transição entre cada estado e a frequência de cada um.

Quanto à evolução que ocorre nos indivíduos ao longo do tempo, assume-se que:

- As características evoluem de forma idêntica e independente;
- A árvore possui a propriedade de Markov, ou seja, a evolução que ocorre em uma subárvore é independente das outras partes da árvore.

Portanto, possuímos um modelo, isto é, uma árvore com seus parâmetros e um modelo de transição de estados, e queremos calcular a probabilidade que os dados observados possam ocorrer a partir dessa modelagem. A partir de vários modelos possíveis, podemos calcular a função que represente essa probabilidade e seu resultado para cada um, buscando seu máximo valor. O valor dessa função também é conhecido por *likelihood* (verossimilhança). O valor máximo entre os *likelihood* analisados indica que o modelo é o melhor dentre todos os analisados, para a entrada em estudo.

Vamos mostrar o cálculo da verossimilhança através de um exemplo simples. Assuma que temos quatro objetos, cada um possuindo um sequenciamento genômico, e um alinhamento, sem *gaps*, entre as sequências. O alinhamento é representado através de uma matriz de características (entrada), mostrada na Tabela 2.3, e cada característica possui quatro estados possíveis (A, G, C e T). Temos como modelar a relação entre esses quatro indivíduos por uma filogenia, com seis nós e sem raiz, representada na Figura 2.2 (a). Os quatro indivíduos de entrada estão representados pelos nós (1), (2), (3) e (4), e os nós (5) e (6) representam nós internos, ancestrais hipotéticos. Precisamos analisar o *likelihood* de nossa entrada para essa modelagem.

Tabela 2.3: Matriz de entrada representando um alinhamento genético de quatro espécies distintas

	1					j				$\cdots$N
(1)	A	G	G	C	T	**C**	C	A	A	$\cdots$A
(2)	A	G	G	T	T	**C**	G	A	A	$\cdots$A
(3)	A	G	C	C	C	**A**	G	A	A	$\cdots$A
(4)	A	T	T	T	C	**G**	G	A	A	$\cdots$C

Necessitamos calcular o *likelihood* de cada uma das colunas separadamente, analisando a filogenia de entrada (nosso modelo). Tomemos como exemplo a coluna de entrada j. É conveniente estabelecer uma raiz para a árvore de nosso modelo, e escolhemos um nó interno arbitrariamente para isso. A árvore está representada na Figura 2.2 (b). Para calcular o *likelihood* da coluna j, rotulamos cada folha com sua característica correspondente na coluna. Os cenários possíveis para os nós internos 5 e 6 seriam de rotulação por cada um dos nucleotídeos A, G, C e T, perfazendo um total de $4 \times 4 = 16$ possibilidades.

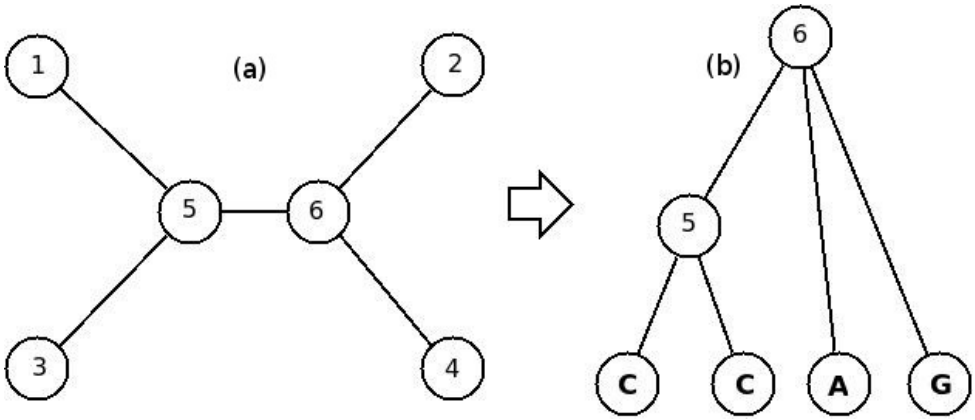


Figura 2.2: Árvore de entrada com quatro objetos e árvore com raiz correspondente

Essas 16 possibilidades representam todos os cenários possíveis dos quais os nucleotídeos presentes nas folhas possam ter evoluído.

O valor do *likelihood* total, representado por L, é o produto do *likelihood* para cada coluna separadamente.

$$L = L(1) \times L(2) \times \cdots \times L(N) = \prod_{j=1}^{N} L(j) \tag{2.1}$$

Como valores individuais de L são números muito pequenos, é conveniente somar o logaritmo de cada coluna separadamente, resultando no valor do logaritmo total, representado em lnL.

$$lnL = lnL(1) + lnL(2) + \cdots + lnL(N) = \sum_{j=1}^{N} lnL(j) \tag{2.2}$$

Após o somatório do valor do *likelihood* de cada coluna da matriz de características, temos o valor total, que avalia a filogenia em relação à matriz de entrada. Esse valor é utilizado como função de *fitness* dos algoritmos genéticos que serão discutidos no Capítulo 3.

2.1.2 Matriz de distâncias

Tendo como entrada uma matriz de características, estamos interessados em minimizar a quantidade de mutações. Intuitivamente, podemos dizer que espécies próximas evolutivamente deveriam estar em posições próximas em uma filogenia. Isso motivou a definição de distância entre duas espécies [39].

Para as definições abaixo, iremos considerar uma matriz de distâncias $M_{n\times n}$, de modo que M_{ij} represente a distância entre os objetos i e j.

Definição 2.1. Uma matriz de distâncias M é métrica se e somente se ela satisfaz as seguintes condições:

- **Simetria**: $M_{ij} = M_{ji}$;

- **Irreflexividade**: $M_{ii} = 0$;
- **Desigualdade triangular**: $M_{ij} + M_{jk} \geq M_{ik}$.

Exemplo 2.2. Na Tabela 2.4, podemos observar essas três propriedades.

Tabela 2.4: Um exemplo de matriz de distâncias $M_{5\times5}$, onde M_{ij} representa a distância entre espécies i e j

M	a	b	c	d	e
a	0	8	8	14	14
b	8	0	2	14	14
c	8	2	0	14	14
d	14	14	14	0	10
e	14	14	14	10	0

Nas próximas duas definições, admitimos que a matriz já satisfaça a condição de ser métrica.

Definição 2.3. Seja S um conjunto de espécies, seja M a matriz de distâncias para S. M é uma matriz métrica aditiva e T é uma árvore aditiva se:

- Cada aresta de T é positiva e cada folha é rotulada por uma espécie distinta de T;
- Para cada $i, j \in S, M_{ij}$ é igual à soma dos pesos das arestas no caminho de i até j.

Uma matriz aditiva métrica possui uma propriedade importante, a condição dos quatro pontos. O teorema é descrito por Buneman [7].

Teorema 2.4. *Uma matriz M é aditiva se, e somente se, para quaisquer quatro espécies, podemos rotulá-las como i, j, l, k tal que em S, temos*

$$M_{ik} + M_{jl} = M_{il} + M_{jk} \geq M_{ij} + M_{kl}$$

.

Uma ilustração do teorema pode ser visto para os quatro pontos i, j, k, l mostrados na Figura 2.3. Baseada nesta condição, podemos verificar se uma matriz é aditiva ou não. Além disso, caso M seja aditiva, existe uma árvore aditiva T que a represente. Segundo a Figura 2.3, temos que:

$$M_{ik} + M_{jl} = (M_{ix} + M_{xy} + M_{yk}) + (M_{jx} + M_{xy} + M_{yl}) = M_{ix} + M_{jx} + M_{yk} + M_{yl} + 2M_{xy}$$

$$M_{jk} + M_{il} = (M_{jx} + M_{xy} + M_{yk}) + (M_{ix} + M_{xy} + M_{yl}) = M_{ix} + M_{jx} + M_{yk} + M_{yl} + 2M_{xy}$$

$$M_{ij} + M_{kl} = M_{ix} + M_{xj} + M_{ky} + M_{yl}$$

A prova para a condição dos quatro pontos pode ser verificada notando que $M_{ik}+M_{jl} = M_{il} + M_{jk} \geq M_{ij} + M_{kl}$.

A Tabela 2.5 nos dá um exemplo de uma matriz aditiva métrica e a Figura 2.4 sua árvore correspondente. Temos uma filogenia sem raiz e que satisfaz as condições descritas na definição de árvore aditiva.

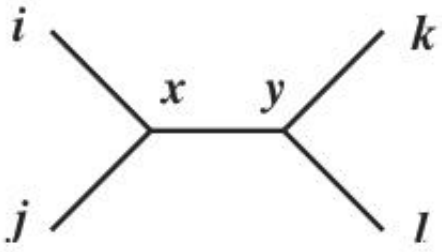


Figura 2.3: A condição de quatro pontos de Buneman [41]

Tabela 2.5: Um exemplo de matriz métrica aditiva [41]

M	a	b	c	d	e
a	0	11	10	9	15
b	11	0	3	12	18
c	10	3	0	11	17
d	9	12	11	0	8
e	15	18	17	8	0

Definição 2.5. Seja M uma matriz aditiva métrica. M é dita ultramétrica e a árvore T é dita ultramétrica se:

- A distância entre duas espécies i e j é igual à soma dos pesos das arestas no caminho entre i e j;
- Uma raiz da árvore T possa ser identificada de tal modo que a distância da raiz a cada uma das folhas seja a mesma, ou seja, tenha um valor fixo.

A matriz da Tabela 2.4 representa uma matriz ultramétrica, e uma árvore T ultramétrica com raiz pode ser representada com as propriedades descritas, como mostrado na Figura 2.5.

A seguir, apresentaremos alguns métodos de construção de filogenias baseadas em matriz de distâncias.

UPGMA

O UPGMA (*Unweighted Pair Group Method with Arithmetic Mean*) constrói uma filogenia baseada em uma matriz ultramétrica [41].

Considere uma árvore ultramétrica T. Um subconjunto de espécies de S que formem uma subárvore de T é chamada de *cluster*. Cada espécie isoladamente também forma um *cluster*. Defina a $altura(u)$ como sendo o caminho de um nó interno u a cada um de seus descendentes que sejam folhas. Como a árvore é ultramétrica, esta altura é igual, pois cada subárvore de uma árvore ultramétrica também é ultramétrica. Sejam i e j folhas descendentes de u. Para que a distância da raiz para ambas as folhas seja a mesma, a $altura(u) = M_{ij}/2$.

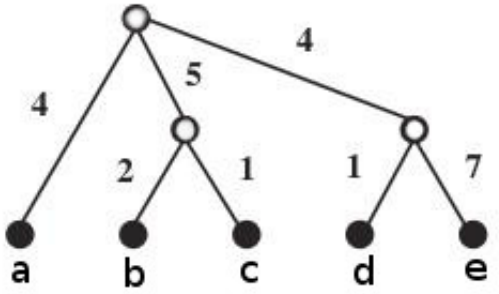


Figura 2.4: A árvore aditiva correspondente à matriz da Tabela 2.5 [41]

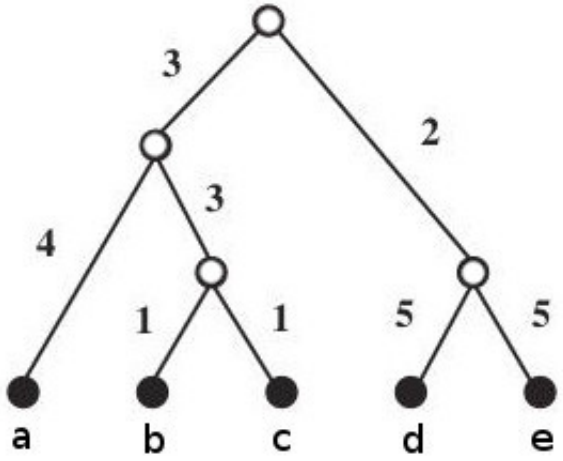


Figura 2.5: Uma árvore ultramétrica formada a partir da matriz da Tabela 2.4 [41]

Para quaisquer *clusters* C_1 e C_2, definimos que

$$dist(C_1, C_2) = \frac{\sum_{i \in C_1, j \in C_2} M_{ij}}{|C_1| \times |C_2|} \tag{2.3}$$

Assim, para um conjunto C de *clusters*, sejam C_i e C_j dois *clusters* em C, e seja C_k uma árvore formada unindo C_i e C_j com uma raiz. C_k é um *cluster* (subárvore) da árvore ultramétrica T. O processo de construção da árvore é ilustrado na Figura 2.6. Inicialmente, cada folha é um *cluster*. Gradativamente, partindo das distâncias mínimas entre *clusters*, o algoritmo UPGMA une dois *clusters* e em $n - 1$ interações, o algoritmo cria a árvore ultramétrica.

A complexidade final do algoritmo é de $O(n^2)$ e a complexidade de espaço, definida pela matriz usada, é de $O(n^2)$ [39].

WPGMA

O WPGMA (*Weighted Pair Group Method with Arithmetic Mean*) funciona de maneira similar ao UPGMA [41]. A diferença é no cálculo da distância entre os *clusters*, que faz com que cada espécie contribua igualmente ao resultado final. A Equação 2.4 mostra como calcular as distâncias do *cluster* recém criado a todos os outros *clusters*, onde C_i e

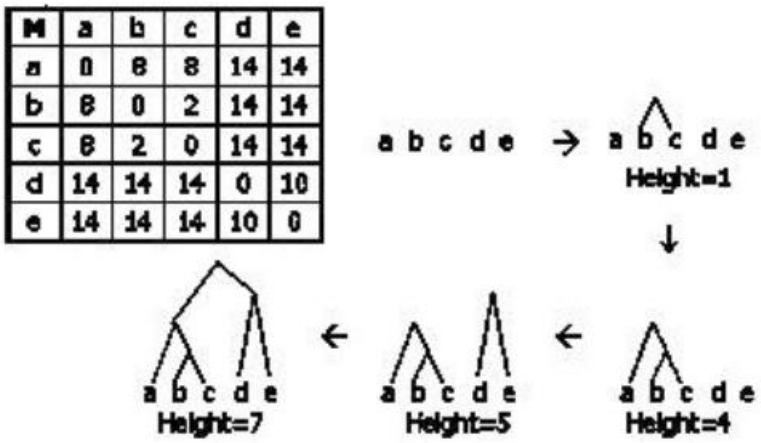


Figura 2.6: Construção de uma árvore ultramétrica [41]

C_j são *clusters* a serem unidos, C_k o *cluster* resultante dessa união, C_x *cluster* em C mas não em C_k, n_i e n_j são os tamanhos respectivos de C_i e C_j.

$$dist(C_{i\cup j}, C_x) = dist(C_k, C_x) = \frac{n_i * dist(C_i, C_x) + n_j * dist(C_j, C_x)}{n_i + n_j} \tag{2.4}$$

Reconstrução de árvore aditiva

A matriz ultramétrica permite estimar o tempo biológico que teria ocorrido ao longo da evolução das espécies. Porém, nem sempre as taxas de evolução são fixas, o que permitiria construir uma filogenia ultramétrica. Essas taxas variam de acordo com genes, espécies e famílias diferentes [45]. Vamos apresentar um método para construção de filogenias a partir de uma matriz métrica aditiva.

Supondo que M seja uma matriz métrica aditiva, podemos construir uma árvore aditiva com complexidade de tempo $O(n^2)$. Esta construção é realizada por inserções consecutivas, e o método foi proposto por Waterman et al [46].

Antes de mostrar o algoritmo, precisamos definir a regra de inserção dos 3-pontos, que nos indica como inserir uma terceira folha em uma determinada aresta. Suponha que necessitamos inserir um objeto B em uma árvore inicialmente formada pelos objetos A e C, com uma aresta que os ligam. A matriz de distâncias que representa este exemplo pode ser vista na Tabela 2.6.

Para isso, conectamos o novo nó B por uma aresta e^B. A distância entre A e C está representada na Tabela 2.6 e possui valor $M_{AC} = 7$. Pela regra dos 3-pontos, a nova aresta que liga B aos demais objetos irá cortar a aresta de distância M_{AC} em um ponto v. Quanto às distâncias das arestas, percebemos que:

$$M_{AB} + M_{BC} = d_{AB} + d_{BC} = d_{Av} + d_{vB} + d_{Bv} + d_{vC} = 2 \cdot d_{vB} + d_{AB} \tag{2.5}$$

Tabela 2.6: Matriz métrica aditiva para o exemplo de reconstrução de uma árvore aditiva [45]

M	A	B	C	D	E
A	0	3	7	10	7
B		0	8	11	8
C			0	9	6
D				0	5
E					0

Chegamos ao valor da nova aresta inserida, que seria

$$d_{vB} = \frac{1}{2}(d_{AB} + d_{BC} - d_{AC}) \tag{2.6}$$

e substituindo, chegamos às equações

$$d_{Av} = d_{AB} - d_{vB} \tag{2.7}$$

$$d_{Cv} = d_{CB} - d_{vB} \tag{2.8}$$

Assim, conectamos B pela aresta e^B de peso

$$\tfrac{1}{2}(d_{AB} + d_{BC} - d_{AC}) = \tfrac{3+8-7}{2} = 2$$

Posteriormente, tentamos conectar D por uma aresta e^D pelo caminho entre os nós B e C. O peso desta aresta seria e^D de

$$\tfrac{1}{2}(d_{BD} + d_{CD} - d_{BC}) = \tfrac{11+9-8}{2} = 6$$

Essa inserção faz com que D seja inserido na posição indicada na Figura 2.7, quebrando o caminho de B à C. A Figura 2.7 indica as inserções de B e D realizadas até aqui.

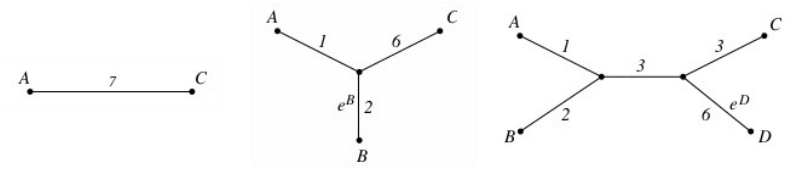


Figura 2.7: Inserção dos novos nós B e D em T' segundo a matriz de distâncias da Tabela 2.6

Finalmente, conectamos E por uma aresta e^E e tentamos inserí-lo no caminho entre B e C. O tamanho de e^E seria de

$$\tfrac{1}{2}(d_{BE} + d_{CE} - d_{BC}) = \tfrac{8+6-8}{2} = 3$$

o que fará com que a nova aresta seja inserida no nó v indicado pela Figura 2.8. Assim, recalculamos e repetimos o procedimento substituindo C por D. O novo peso da aresta e^E será de

$$\frac{1}{2}(d_{BE} + d_{DE} - d_{BD}) = \frac{8+5-11}{2} = 1$$

e o procedimento é finalizado, recalculando as demais arestas e resultando na filogenia mostrada na Figura 2.8.

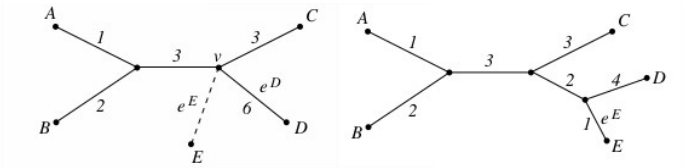


Figura 2.8: Inserção do nó E da matriz de distâncias da Tabela 2.6, após uma repetição do procedimento de inserção da aresta e^E

Neste momento, podemos mostrar o algoritmo para, a partir da matriz M e um conjunto de objetos O, construir uma árvore T. Inicialmente, o algoritmo escolhe dois objetos arbitrários i e j e os conectam por uma aresta de peso M_{ij}, resultando em uma árvore intermediária T'. Iterativamente, para cada objeto $k \in O$ ainda não presente em T', faça:

1. Escolha um par de folhas $i, j \in T'$;
2. Compute o peso da aresta a^k pela regra dos 3-pontos;
3. Se a inserção de a^k em T' implica que a aresta tenha que ser inserida dentro de outra aresta, então divida a aresta, insira um nó interno e conecte a^k àquele nó. Ccaso contrário, substitua j (ou i) por outra folha da subárvore que estaria conectada ao ponto de inserção e continue com o passo 2.

Neighbor joining

O método *neighbor joining* [33] consiste em gerar uma árvore a partir de uma matriz de entrada M não aditiva através da geração de *clusters*, iniciando pelos vizinhos mais próximos, unindo dois *clusters* em cada iteração. Inicialmente, o método considera como um *cluster* cada uma das espécies, depois une dois *clusters* a cada iteração, até que restem dois grandes *clusters* a serem unidos por uma raiz, resultando na árvore final.

A Figura 2.9 mostra os passos do algoritmo, a partir de uma matriz de distâncias M de tamanho $n \times n$ como entrada. O algoritmo é inicializado de maneira que cada folha forme um *cluster*. Depois, dois a dois, os *clusters* são unidos, combinando os mais próximos do grupo, as novas arestas são calculadas, o conjunto de *clusters* atualizado até que restem apenas os dois últimos, que serão unidos pela raiz formando a árvore final.

Inicialização:

1. Seja $Z = 1, 2, ..., n$ o conjunto inicial de *clusters*.
2. Para cada $i, j \in Z$, faça $dist(i, j) = M_{ij}$.

Passos iterativos: repita os passos abaixo $n - 1$ vezes:

1. Para cada *cluster* $A \in Z$, compute $u_A = \frac{1}{n-2} \sum_{D \in Z} dist(D, A)$

2. Encontre dois *clusters* $A, B \in Z$ tais que $dist(A, B) - u_A - u_B$ é mínimo.
3. Faça A e B como duas subárvores de um nó interno r. Nomeie este *cluster* resultante como C.
4. Compute os pesos das arestas:

$$d_{Ar} = \frac{1}{2}dist(A, B) + \frac{1}{2}(u_A - u_B) \text{ e } d_{Ar} = \frac{1}{2}dist(A, B) + \frac{1}{2}(u_B - u_A)$$

5. Atualize o conjunto de *clusters* $Z = Z \cup C - A, B$.
6. Atualize os pares de distâncias entre os *clusters* em Z: compute $dist(D, C)$ para cada $D \in Z - C$,

$$dist(D, C) = dist(C, D) = \frac{1}{2}(dist(A, D) + dist(B, D) - dist(A, B))$$

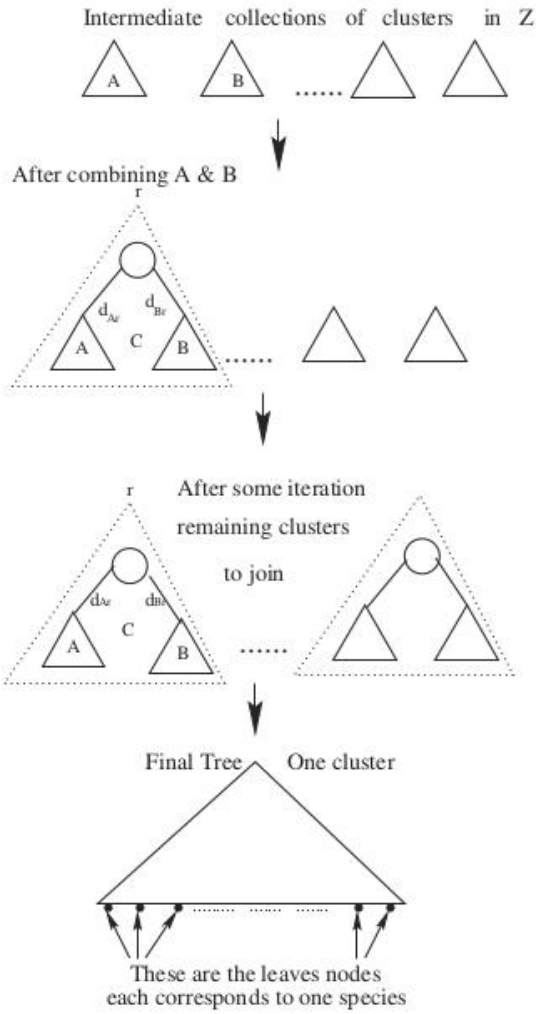


Figura 2.9: Método de construção de uma árvore filogenética usando *neighbor joining* [41]

2.1.3 Filogenia viva

Na filogenia convencional, o resultado da construção da história evolucionária de um conjunto de objetos é uma árvore cujas folhas são os objetos e os nós internos são ancestrais comuns hipotéticos. A filogenia viva, definida por Telles et al [44], trabalha com ancestrais comuns que podem estar na lista de objetos de entrada. Caso nossos objetos de estudo sejam espécies, os nós internos representam ancestrais vivos. Outras aplicações incluem a análise de população de vírus, organismos de rápida evolução, análise de documentos, imagens, etc.

Filogenia viva baseada em matriz de distâncias

Comprovaremos que, dada uma matriz qualquer aditiva M^n de n objetos, é sempre possível construir uma filogenia viva, representada por uma árvore T^n. Dizemos que

$$T^n \sim M^n \text{ (árvore } T \text{ é compatível com a matriz } M).$$

Seja M^k aditiva e T^k tal que $M^k \sim T^k$. Um novo objeto o_{k+1} distinto é acrescentado. Podemos adicionar esse objeto de tal forma que $T^{k+1} \sim M^{k+1}$.

Para provar, primeiramente assumimos a possibilidade de, a partir de dois nós apenas, o novo nó ser acrescentado. Então, de uma árvore T^2 com x e y, adiciona-se z gerando T^3. Nos três primeiros casos estudados, assume-se z sendo inserido na mesma aresta de x e y, ou entre os nós ou nas pontas. A Figura 2.10 ilustra as situações possíveis. Nela, é possível observar que algumas vezes o nó z se torna um ancestral vivo, e outras vezes os nós x ou y tornam-se ancestrais vivos. Nas figuras, diferenciamos ancestrais vivos de hipotéticos com a cor do nó que os representam: ancestrais vivos são pintados de preto, assim como as folhas, enquanto ancestrais hipotéticos são representados por um círculo com interior branco.

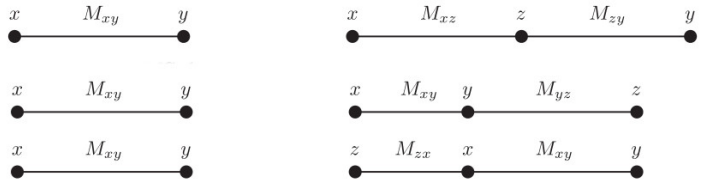


Figura 2.10: Casos de inserção de um novo nó z em uma árvore contendo inicialmente os nós x e y [44]

Além disso, é possível inserir z conectando-o a um novo nó interno c, que se torna um ancestral hipotético de x, y e z. O caso é similar ao já discutido na matriz aditiva e as distâncias podem ser calculadas pelas Equações 2.6, 2.7 e 2.8.

Para $k \geq 3$, a prova que um nó z pode ser adicionado a T^k, resultando em uma nova árvore $T^{k+1} \sim M^{k+1}$, pode ser descrita pelos passos indutivos a seguir. Analisamos quatro casos distintos de inserção para que possamos comprovar que $d_{zw} = M_{zw}$ para qualquer nó $w \neq x, y$.

1. Caso 1: temos que $M_{xz} + M_{zy} = M_{xy}$. Suponha que o nó w fica fora do caminho entre x e y, caminho que z é inserido. O novo nó z pode se tornar um ancestral vivo

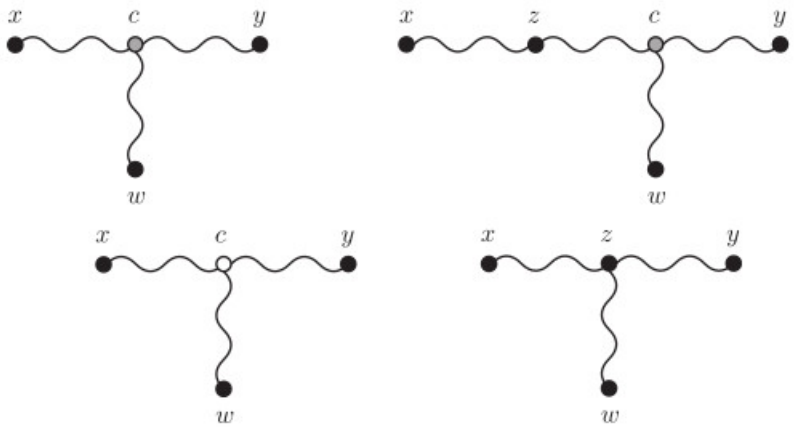


Figura 2.11: Caso 1: inserção de z como ancestral vivo [44]

no caminho de x, y ou substituir um nó interno c, anteriormente ancestral hipotético, conforme ilustrado na Figura 2.11.

2. Caso 2: temos que $M_{xz} = M_{xy} + M_{yz}$. Desta maneira, z passa a ser conectado por uma nova aresta a y, conforme Figura 2.12.

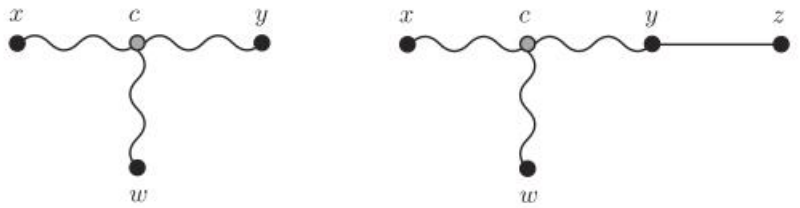


Figura 2.12: Caso 2: inserção de z em nova aresta [44]

3. Caso 3: temos que $M_{yz} = M_{xy} + M_{xz}$. Caso similar ao caso anterior.

4. Caso 4: se nenhum dos anteriores ocorrerem, z deve ser conectado a um novo nó. Este nó c ao qual z será conectado por uma nova aresta pode ser um novo ancestral hipotético ou um nó já existente, seja vivo ou não. Os casos possíveis de inserção estão ilustrados na Figura 2.13.

As condições dos quatro pontos para provar que a árvore gerada a partir de M a partir dos casos citados é aditiva é demonstrada em Telles et al [44]. O algoritmo possui complexidade de tempo $O(n^3)$.

Algoritmo para matriz de características da filogenia viva

Necessitamos construir uma árvore com raiz que represente a relação evolucionária dos objetos. A árvore é construída a partir de uma matriz $n \times m$ binária, representando os objetos e as características presentes em cada objeto. O algoritmo para filogenia viva, descrito no Algoritmo 2.1, é uma simples adaptação do algoritmo de Waterman et al [46] e possibilita a geração de ancestrais vivos. A complexidade de tempo desse algoritmo é polinomial, $O(n * m)$.

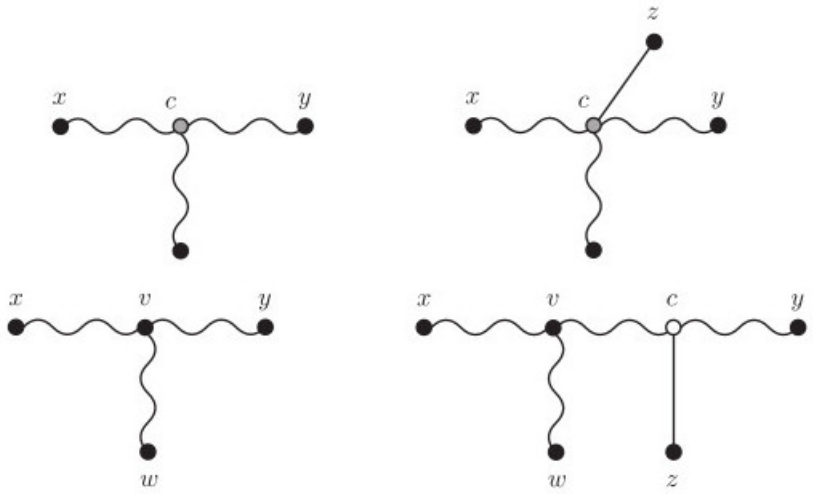


Figura 2.13: Caso 4: possibilidades de inserção de z a um nó c entre x e y [44]

Algorithm 2.1 Algoritmo Perfect-Live-Phylogeny(M) [44]

```
1:  Crie a raiz
2:  for i = 1 to n do
3:      cur = raiz
4:      for j = 1 to m do
5:          if M_ij = 1 then
6:              if não existem arestas (cur, u) rotuladas com c_j then
7:                  crie nó u e aresta (cur, u) rotulada c_j
8:              end if
9:              cur = u
10:         end if
11:     end for
12:     rotule cur com o_i
13: end for
14: return raiz
```

2.2 Algoritmos genéticos

O homem utiliza a ciência para gradualmente obter o conhecimento necessário para compreender fenômenos que ocorrem no mundo. Ganhamos a capacidade de predizer, até certo grau, o tempo, movimento dos astros, eclipses, variações econônimcas e outros fenômenos naturais e culturais. A introdução desta seção é baseada em Mitchell [25].

Desde o início da computação, com os cientistas Alan Turing e John von Neumann, entre outros, já havia um objetivo de criar algum tipo de inteligência artificial, uma espécie de vida artificial. Esses pioneiros se interessavam muito por Biologia, Psicologia, Eletrônica e outros sistemas naturais encontrados no mundo para tentar guiar a construção dessa inteligência computacional. As motivações biológicas foram aprimoradas com o passar dos anos, mas foi principalmente nos anos 80 que elas passaram a ser estudadas por uma comunidade de pesquisadores, passando por temas como redes neurais, aprendizado

de máquina e computação evolucionária.

A área da Computação Evolucionária, na qual podemos citar algoritmos genéticos, baseia-se em mecanismos encontrados na natureza. Possuem, em geral, três ideias básicas:

1. A criação de uma população de soluções;
2. A criação de uma função que avalie os indivíduos da população;
3. A criação de operadores de seleção, que gerem mutações e recombinações.

A evolução pode ser usada como uma inspiração aos problemas computacionais, porque seus mecanismos são adequados a diferentes situações, dentre as quais podemos citar a evolução. Muitos problemas necessitam uma busca exaustiva em um grande número de possibilidades de soluções. Além dessa inteligência estratégica, um certo grau de paralelismo computacional também é desejável para resolução de problemas desse tipo.

2.2.1 Conceitos biológicos

Na Biologia, devido às imensas possibilidades de sequências de DNA ou RNA distintas, também são grandes as possibilidades de um organismo expressar proteínas. Entretanto, é desejável que esse organismo se adapte bem ao seu meio, isto é, consiga sobreviver e reproduzir em seu ambiente. A evolução, através de mutações ou recombinações, é um mecanismo pelo qual são geradas soluções inovadoras para problemas complexos (sobrevivência no meio).

Os conceitos biológicos a seguir [9] são utilizados em algoritmos genéticos como uma analogia ao mundo biológico:

- Organismo é um ser que possui uma série de regras e é constituído de células. Em cada célula, existe um conjunto de cromossomos;
- Cromossomos são cadeias de DNA que servem de modelo para o organismo inteiro. Um cromossomo é constituído por genes;
- Genes são constituídos por DNA. Cada gene irá codificar algum tipo particular de proteína do organismo. Cada gene tem sua posição própria dentro do cromossomo;
- Genoma é o conjunto completo de cromossomos de um organismo;
- Fenótipo é um conjunto de características físicas que são expressas por um organismo, representando o resultado de seus genes. Pode ser, por exemplo, a cor dos cabelos, olhos ou tipo físico, como altura e estrutura muscular;
- Recombinação é um processo em que dois organismos se combinam e passam seus genes aos descendentes, que passam a ter metade dos genes de cada um dos pais;
- Mutação é um processo que modifica ligeiramente elementos de um DNA, ou seja, ocasiona pequenas alterações na sequência de bases (A, C, G ou T). Após uma recombinação, é possível que o novo descendente também sofra mutação;
- *Fitness* do organismo, também chamado de valor adaptativo ou aptidão, é uma medida de quão adaptado o organismo está ao meio, uma espécie de função que mede sua taxa de sucesso. Para organismos vivos, esta seria uma medida de sua capacidade de sobrevivência.

2.2.2 Operações de algoritmos genéticos

Na Figura 2.14, apresentamos um esquema geral de um algoritmo genético e do processo evolutivo de sua população. No esquema, a população (*population*) é inicializada (*initialization*), os pais (*parents*) da nova geração são selecionados, sofrem processos de recombinação e mutação e geram descendentes (*offspring*). A partir daí, os sobreviventes fazem parte de uma nova população em uma nova geração, e o processo é repetido até um determinado critério de terminação.

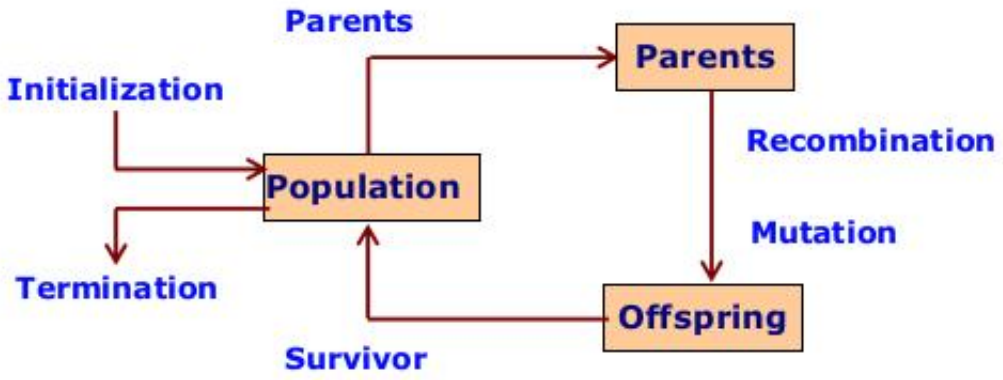


Figura 2.14: Esquema geral de processo evolutivo [9]

De acordo com as definições descritas no parágrafo anterior, temos um exemplo simples de pseudo-código no Algoritmo 2.2 .

Algorithm 2.2 Algoritmo Genético [9]

Inicialize a população com uma solução aleatória
Avalie cada indivíduo (função de *fitness*)
repeat
1. Selecione os pais;
2. Recombine pares de pais;
3. Realize mutações na nova geração;
4. Avalie novamente cada indivíduo para próxima geração

until condição de término alcançada

O fluxograma da Figura 2.15 representa uma implementação possível de um algoritmo genético. Nele, temos alguns passos iniciais, a geração de indivíduos e definição do *fitness* de cada um, e posteriormente o laço principal do algoritmo, representando cada geração. Dentro das gerações, indivíduos são selecionados para sofrerem recombinações e mutações e passam pela função de *fitness* para serem avaliados. Um critério de terminação avalia se o algoritmo chegou ao fim ou uma nova geração deve ser criada.

Codificação

Para fazer uso de um algoritmo genético, é necessário definir a codificação que será utilizada para cada indivíduo, geralmente definida de acordo com cada problema. As

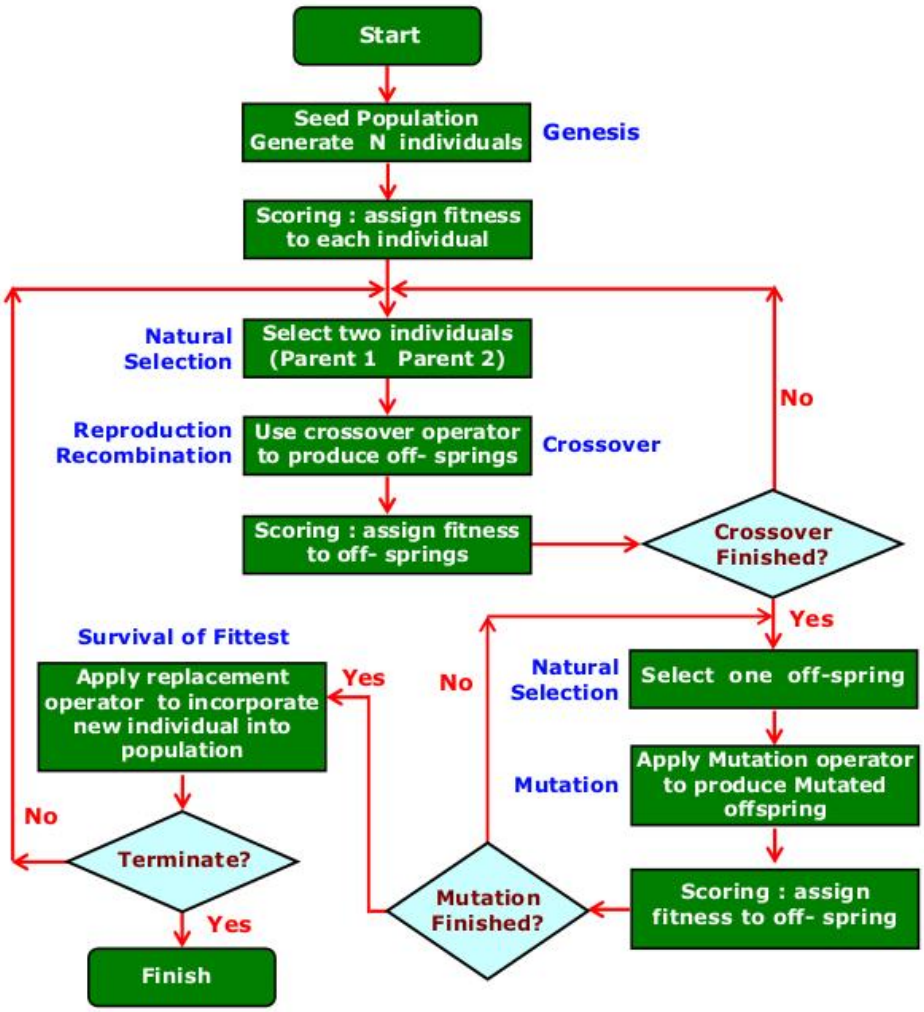


Figura 2.15: Fluxograma de um algoritmo genético [9]

codificações podem ser binárias, de diversos valores, de permutação ou de árvores, por exemplo.

Cada cromossomo (indivíduo) pode possuir diversos genes. Para ilustrar a codificação, tomemos como exemplo a codificação binária. Um cromossomo poderia ser, neste exemplo, um vetor de genes na forma binária. Cada cromossomo abaixo contém alguns genes de 4 bits cada:

- Cromossomo 1: 1101 1001 0011 0110
- Cromossomo 2: 1101 1110 0001 1110

Se definirmos que o valor representado pelos 4 bits pode ser lido como um valor numérico, teríamos valores de 0 a 15 representados em cada um dos genes. Neste caso, as operações de mutação poderiam alterar um bit de algum gene, e operações de recombinação poderiam combinar genes de diferentes indivíduos para formar um descendente.

Quanto a representação de grafos, existem alguns tipos de codificações disponíveis. Seja um grafo G de n nós e m arestas possíveis. Raidl et al [29] citam algumas codificações possíveis:

- *Pruefer numbers*: representados por vetores de números inteiros, de tamanho $n-2$. Cada vetor representa uma árvore distinta, e toda árvore pode ser construída a partir dos números de Pruefer, definidos em Abuali et. al [4];
- *Vetor de características*: representados por uma *string* binária em que cada posição corresponde às arestas possíveis do grafo G, e o vetor indica quais dessas arestas fazem parte dele;
- *Network random keys*: representados por vetores de tamanho m, sendo m o total de arestas possíveis. Cada valor do vetor representa o peso da aresta. O vetor é ordenado por pesos;
- *ELink-and-Node Biasing*: representados por um vetor que indica pesos associados aos nós e, opcionalmente, às arestas. Utiliza o algoritmo de Prim [26] para recuperar o grafo.

Cada solução de codificação deve ser estudada de acordo com o problema estudado. Para problemas que necessitem representar árvores em topologia estrela, por exemplo, a codificação de Pruefer possui uma localidade aceitável [31], ou seja, alterações pequenas no grafo codificado provocam pequenas alterações no grafo final. Já em outras topologias, essa codificação pode não ser adequada, uma vez que pequenas alterações provocariam grandes alterações topológicas [32]. A codificação possui grande influência para que um algoritmo genético encontre um ótimo local, portanto, deve ser escolhida apropriadamente.

Na utilização de algoritmos genéticos para filogenia, são mais utilizadas árvores binárias, que são um tipo particular de grafo. Algoritmos como GAML [21] e GARLI [10], por exemplo, utilizam o formato newick [43] para codificar as árvores filogenéticas. Alterações são realizadas na topologia diretamente através de determinadas operações, descritas na Seção 3.2, em vez de alterações realizadas diretamente na codificação da árvore.

Reprodução e seleção

Usualmente, a primeira ação de um algoritmo genético é aplicar a reprodução e seleção nos indivíduos, isto é, selecionar os pais dos quais virão os novos descendentes. Devemos selecionar os indivíduos que melhor se adaptam à solução, ou seja, algo semelhante com a sobrevivência dos melhores que ocorre no mundo biológico.

A função de *fitness* do algoritmo é a responsável por definir a qualidade de cada um dos indivíduos. É possível selecionar múltiplas cópias de determinado indivíduo para a próxima geração, ou realizar operações que os selecionem. Os métodos utilizam uma função de *fitness* $f(x)$, onde F_i é o *fitness* do objeto i, p_i a probabilidade do objeto i ser selecionado, n o total de indivíduos em uma população. Alguns desses métodos são listados abaixo:

- **Seleção de roleta**: a probabilidade do i^{esimo} indivíduo ser selecionado depende de seu *fitness* em relação ao somatório total. Quanto maior o valor de *fitness*, mais adequado julgamos ser o indivíduo e mais chances ele teria de ser selecionado. É definida como

$$p_i = \frac{F_i}{\sum_{j=1}^{n} F_j} \tag{2.9}$$

- **Seleção de Boltzmann**: a seleção introduz o conceito de temperatura e a introduz como parâmetro no cálculo da distribuição de probabilidade de Boltzmann. A altas temperaturas, as probabilidades se tornam mais uniformes do que em temperaturas mais baixas;
- **Seleção por *rank***: a seleção é uma probabilidade baseada na posição do indivíduo em um *rank*. Em vez de ser em função do valor de *fitness*, é em função da posição do indivíduo em relação aos demais;
- **Seleção de campeonato**: similar à seleção por *rank*, mas difere na maneira de implementação, pois a seleção por *rank* necessita ordenar os indivíduos pelo *fitness* antes de serem selecionados, consumindo recurso e tempo computacional. Na seleção por campeonato, dois indivíduos são selecionados aleatoriamente, e caso um valor aleatório r seja maior que um parâmetro k, por exemplo, r entre 0 e 1 e $k = 0.75$, então apenas o indivíduo com melhor *fitness* é selecionado. O método e análise foram propostos em Goldberg e Deb [15];
- **Seleção *steady-state***: a maioria dos algoritmos genéticos modificam a nova geração em grande número de indivíduos. Nesse método, uma boa parte da população anterior é mantida na nova geração, o que pode ser utilizado para resolver problemas em que o conjunto de soluções é importante como resposta, e em que lembrar de um conjunto de soluções anteriores pode ser útil [25].

Uma variação da seleção por *rank* foi utilizada por Meyer e Packard [24]. A diferença é que foi utilizada uma estratégia elitista, em que E indivíduos foram selecionados para os descendentes, todos copiados do primeiro do *rank*, e outros $N - E$ indivíduos são selecionados para compor a nova geração. Esse método pode ser útil em casos onde a função de *fitness* provoca algum tipo de ruído, geralmente ocasionado por alguma aleatoriedade provocada pela função [25].

Crossover

Operações de *crossover* são utilizadas para combinar dois pais e produzir um novo cromossomo para a geração descendente. O objetivo do *crossover* é que um novo indivíduo, composto por dois indivíduos da geração anterior, pode conter características (genes) positivas de cada um dos pais, resultando em um descendente mais forte.

As operações podem ser dos tipos: um-ponto, dois-pontos, uniforme, aritmética ou heurística. Os tipos de operações que serão adotadas dependem muito da maneira em que os cromossomos são codificados. As operações de um-ponto escolhem um ponto do cromossomo, quebram o pai neste ponto e realizam um intercâmbio de seus genes com outro pai, também cortado neste mesmo ponto. Podem ser facilmente visualizadas com codificações binárias. A operação de dois-pontos é similar, difere apenas por quebrar o indivíduo em dois locais distintos e realizar a troca. Ambas as operações podem ser visualizadas na Figura 2.16.

A operação uniforme determina, com certa probabilidade, de qual pai virá cada um dos genes. Ela permite que os descendentes contenham uma mistura maior de segmentos dos pais em vez de apenas um ou dois segmentos de cada um.

Operações aritméticas trabalham com codificações em ponto flutuante, por exemplo. Os descendentes são gerados de acordo com um parâmetro a que indica um peso de 0 a

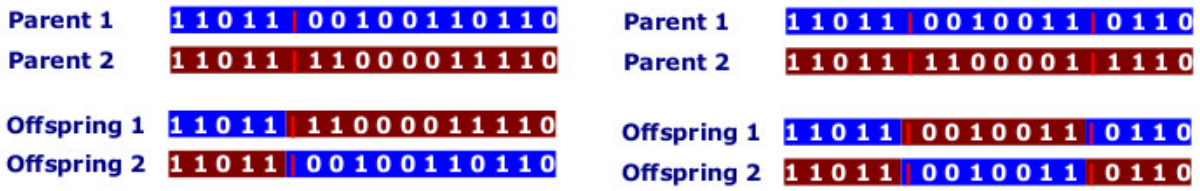


Figura 2.16: Exemplos de *crossover* um-ponto e dois-pontos [9]

1, e que é multiplicado em cada um dos genes dos pais para gerar a nova geração. Por exemplo, podemos obter dois descendentes na forma:

1. $a * Pai1 + (1 - a) * Pai2$
2. $a * Pai2 + (1 - a) * Pai1$

Também é possível a definição de *crossover*s variados, como veremos no Capítulo 3, nas recombinações de cromossomos codificados como árvores. Essas operações variadas, geradas de acordo com cada problema a ser resolvido, consistem nas operações heurísticas.

Mutação

A mutação é realizada após o descendente ter sido selecionado, tendo ou não passado por operações anteriores, como o *crossover*.

A mutação altera um ou mais genes do estado inicial de um cromossomo. Isso pode resultar em genes totalmente novos surgindo na população de indivíduos. A mutação é um processo importante e pode evitar que a busca do algoritmo genético fique presa em um ótimo local, sem conseguir novas possibilidades de melhores soluções [9].

As operações de mutação podem ser de diversos tipos, e também devem ser definidas de acordo com a codificação utilizada:

- ***Flip bit***: A mutação simplesmente altera o valor de bits, de 0 para 1 ou 1 para 0. Usada em codificações binárias;
- **Fronteira**: a mutação escolhes valores limites para um determinado gene. Usada para codificações binárias ou em ponto flutuante;
- **Não-uniforme**: A probabilidade de operações de mutações descrescem com o passar das gerações, impedindo que o algoritmo pare no início, sem encontrar novas soluções, e melhore ao final da execução, através de ajustes finos na população para encontrar melhores indivíduos;
- **Uniforme**: o usuário especifica fronteiras de valores máximos e mínimos que um gene poderá receber em uma mutação. Usado para codificações binárias ou em ponto flutuante;
- **Gaussiano**: a operação de mutação seleciona um valor através de uma função de probabilidade de distribuição gaussiana. Usado para codificações binárias ou em ponto flutuante.

As operações de mutação para representações de filogenia são diferenciadas em função do problema em estudo e codificação da árvore. Alguns tipos de mutações são definidos por cada software, e serão explorados no Capítulo 3.

Capítulo 3

Revisão de literatura

Neste capítulo, faremos uma revisão de literatura de algoritmos capazes de resolver o problema da filogenia pequena, na Seção 3.1, e de softwares que utilizam algoritmos genéticos para resolver o problema da filogenia grande, na Seção 3.2.

3.1 Filogenia pequena

No problema da filogenia pequena, estamos interessados em computar uma medida de parcimônia de uma árvore dada como entrada. Dada uma filogenia, representada por uma topologia representada em uma árvore T de raiz r e folhas rotuladas por um conjunto S, queremos encontrar uma medida de parcimônia de T em relação a entrada S. Para isso, precisamos responder a duas perguntas:

1. Qual o número mínimo de mutações que seriam necessárias para que esta topologia represente a realidade?
2. Qual o rótulo mais adequado a cada um dos nós internos?

A resposta do item 1 pode ser calculada analisando cada uma das características separadamente, assumindo que essas características sejam mutuamente independentes. Podemos responder a essas perguntas utilizando o algoritmo de Fitch [14] ou o algoritmo de Sankoff [35], sendo que ambos executam em tempo polinomial.

Para ilustrar os algoritmos, vamos assumir como entrada a matriz representada na Tabela 3.1, e a árvore topológica de entrada a representada na Figura 3.1.

Tabela 3.1: Matriz de entrada com alinhamento entre seis espécies de primatas [6]

Humano	C	A	C	T
Chimpanzé	T	A	C	T
Bonobo	A	G	C	C
Gorila	A	G	C	A
Gibão	G	A	C	T
Lemur	T	A	G	T

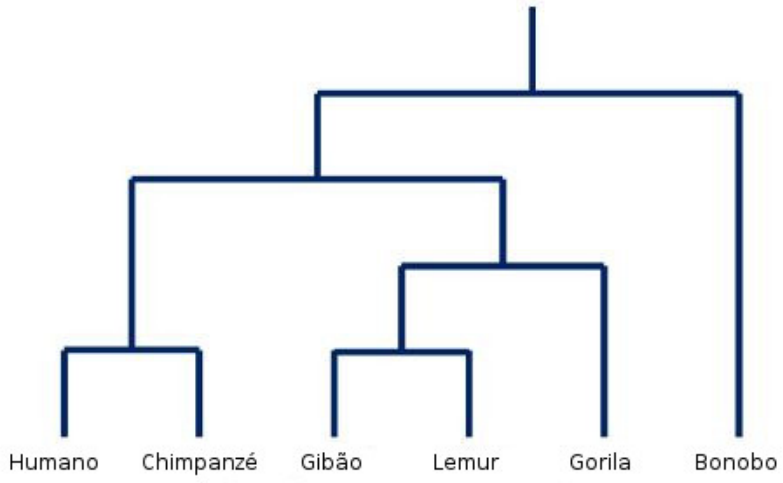


Figura 3.1: Filogenia para primatas [6]

3.1.1 Algoritmo de Fitch

O algoritmo de Fitch possui duas fases, mostradas nos Algoritmos 3.1 e 3.2, que devem ser executadas independentemente para cada característica. Na primeira fase do algoritmo, o processo é das folhas para a raiz, chamado de fase *bottom-up*. Na segunda fase, o processo é da raiz para as folhas, ou *top-down*.

Algorithm 3.1 Algoritmo de Fitch: *Bottom-up*

1: Inicialize cada folha s_i definindo $R_i = \{s_i\}$
2: Percorra a árvore das folhas até a raiz (pós-ordem)
3: Determine R_i de cada nó interno i com filhos j, k:

$$R_i = \begin{Bmatrix} Se R_j \cap R_k \neq \emptyset \rightarrow R_j \cap R_k \\ c.c. \rightarrow R_j \cup R_k \end{Bmatrix}$$

4: O escore de Fitch é o valor total de operações de união

Algorithm 3.2 Algoritmo de Fitch: *top-down*

1: Escolha um estado arbitrário de (R_{root}) para ser o estado da raiz, s_{root}.
2: Percorra a árvore da raiz às folhas (pré-ordem)
3: Determine o estado s_i do nó interno i com pai j.

$$s_i = \begin{Bmatrix} \text{Se } s_i \in R_i \rightarrow s_i \\ c.c. \rightarrow \text{estado aleatório} \in R_i \end{Bmatrix}$$

O escore total de Fitch será a soma das operações de união em cada uma das execuções do algoritmo, ou seja, em cada uma das colunas da matriz de entrada. Essa soma total nos dá a quantidade de mutações necessárias para que esta árvore represente aquela entrada. É desejável, pelo princípio da parcimônia, que esta quantidade seja mínima.

Vamos exemplificar a execução do algoritmo da fase *bottom-up* através da Figura 3.2. Nela, executamos o primeiro passo do algoritmo utilizando os valores da primeira coluna

da matriz. O algoritmo inicia nas folhas e, ao subir a um nó interno acima, vai executando operações de união ou interseção dos valores de R_i, que representam os valores possíveis de estado para aquele nó. As operações de união estão marcadas com um círculo. Este passo somará 4 no escore total do algoritmo, pois o algoritmo executou 4 operações de união.

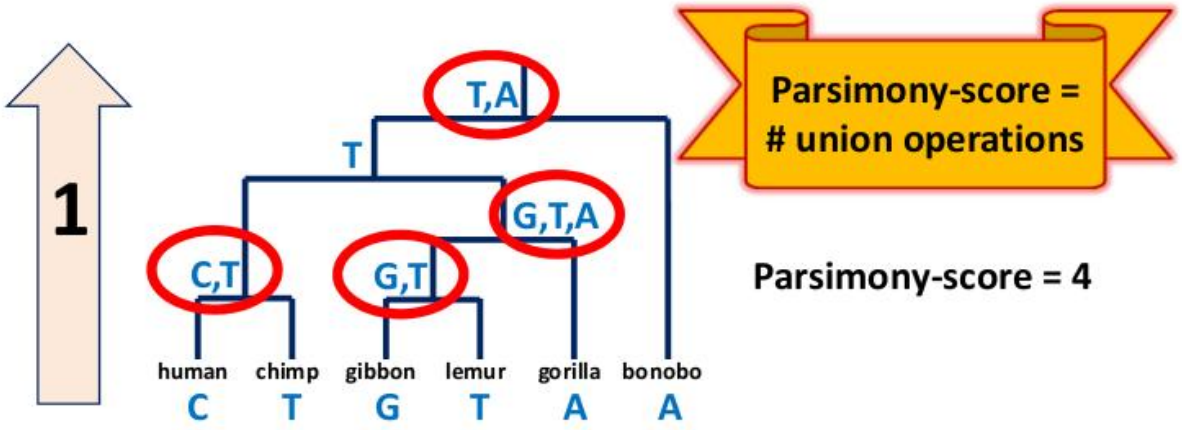


Figura 3.2: Execução do algoritmo de Fitch na árvore dada como entrada [6]

O algoritmo segue com o processo de descida na fase *top-down*, em que se define qual a melhor característica para cada um dos nós internos, aquela em que faz a árvore ter o menor número possível de mutações. Este processo final do algoritmo rotula a árvore mas não altera o cálculo do escore do passo anterior. Após o processo de descida, o algoritmo seleciona os valores de estados para cada nó interno. O processo é ilustrado na Figura 3.3.

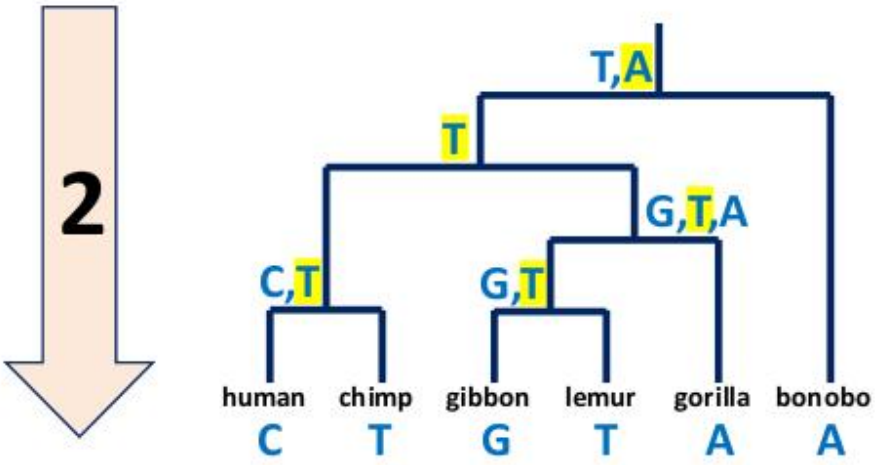


Figura 3.3: Execução da fase final de Fitch na árvore de entrada, em que o estado associado aos nós internos estão destacados [6]

3.1.2 Algoritmo de Sankoff

O custo de transição de uma característica para outra no algoritmo de Fitch é restrito a uma unidade. Sankoff [35] desenvolveu uma generalização para custos independentes e simétricos entre as características possíveis. Similarmente ao Fitch, o algoritmo executa fases de subida e descida, e calcula separadamente cada uma das características.

As fases do algoritmo são descritas da seguinte forma:

1. **Fase *bottom-up*:** aqui, a árvore é percorrida das folhas à raiz. Defina $C(u)$ como sendo o custo da melhor solução para a mutação mínima de uma subárvore T_u de raiz u. Seja $C(u, a)$ o custo de rotular o nó u com o estado a. Assim, $C(u) = min_a C(u, a)$. O valor de $cost(a, b)$ do custo de transição do estado a para b é dado pela matriz de custo de entrada.

   (a) **Folhas:** O estado para uma folha l é fixado como entrada. Os custos são definidos como $C(l, a) = 0$ para a sendo o estado de l, e $C(l, a) = \infty$ caso contrário;

   (b) **Nós internos:** A forma recorrente de computar $C(u, a)$ para um nó interno é dada por

   $$C(u, a) = \sum_{v \text{ filho de } u} min_b(cost(a, b) + C(v, b))$$

2. **Fase *top-down***: O algoritmo percorre da raiz às folhas escolhendo o estado ótimo de cada nó.

   (a) A **raiz** r recebe um estado a tal que $C(r) = C(r, a)$;

   (b) Para os **demais nós**, seja um nó v filho de um nó u, com estado já definido por a, rotulamos v com o estado b tal que tenha sido mínimo na fase *bottom-up*, de modo que

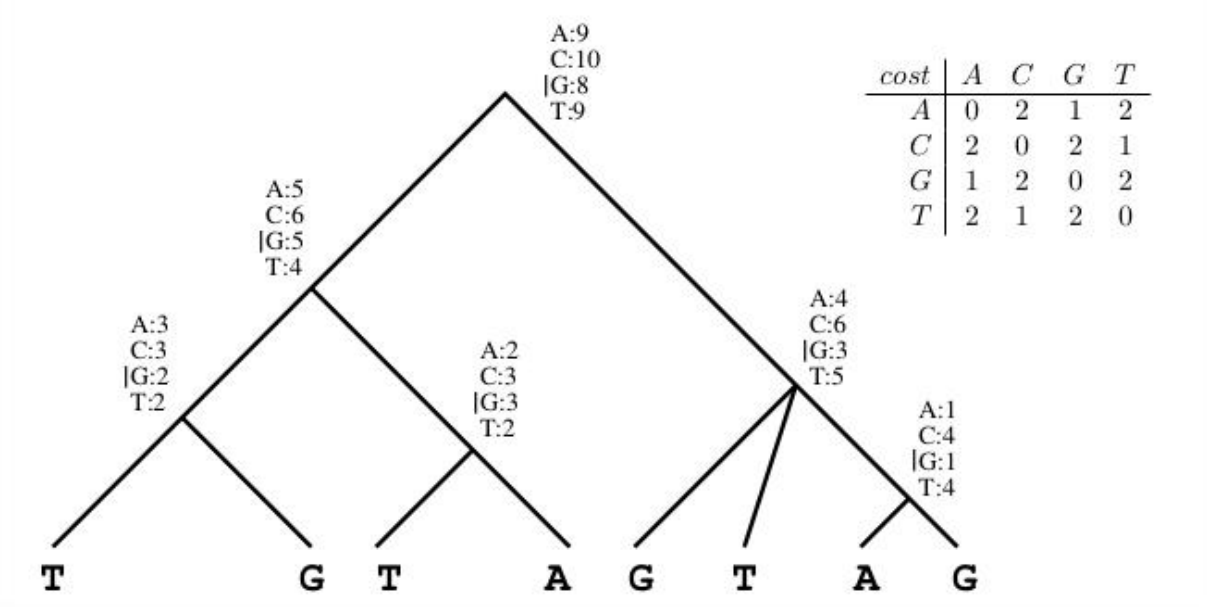
   $$cost(a, b) + C(v, b) = min_{b'}[cost(a, b') + C(v, b')]$$

Uma ilustração do algoritmo de Sankoff pode ser visto na Figura 3.4. Nesse exemplo, a matriz simétrica de custo de transição dos estados está representada. Na fase de *bottom-up*, o custo de rotular um nó é calculado para cada um dos estados possíveis. O cálculo segue de acordo com a fórmula apresentada, somando o custo de transição a partir de um estado b, em que $C(v, b)$ é mínimo, para cada um dos filhos.

3.2 Filogenia grande

Como o número de soluções possíveis para construir árvores que representem uma matriz de entrada aumenta consideravelmente com a entrada, algumas heurísticas são necessárias para explorar algumas soluções possíveis.

Para resolver o problema da filogenia grande, alguns softwares foram desenvolvidos utilizando algoritmos genéticos. Serão detalhados nessa seção os softwares GAML [21] e GARLI [10], que foram usados como base para a nossa proposta de algoritmo genético para a filogenia viva.



cost	A	C	G	T
A	0	2	1	2
C	2	0	2	1
G	1	2	0	2
T	2	1	2	0

Figura 3.4: Exemplo de execução do algoritmo de Sankoff utilizando a matriz de custo simétrica *cost* e os estados definidos nas folhas [45]

3.2.1 GAML

GAML significa *Genetic Algorithm for Maximum-Likelihood Phylogeny Inference* e é um algoritmo genético que utiliza *Maximum Likelihood* como função de *fitness* [21]. Utiliza como entrada dados um alinhamento.

O GAML define seus cromossomos com três valores. Na primeira coluna, o valor de seu escore, calculado com o logaritmo natural do valor obtido no processo de cálculo da Máxima Verossimilhança (lnL); na segunda, um parâmetro k, usado na modelagem HKY[1]; na terceira, a árvore no formato newick, estabelecido por Swofford e Begle [43]. Um exemplo de cromossomos pode ser visto na Figura 3.5.

No início do programa, cada indivíduo recebe uma topologia aleatória com tamanhos de arestas iguais. Cada aresta é modificada ligeiramente, ou seja, de acordo com um número aleatório dado por uma distribuição gaussiana de tal forma que não provoque alterações significativas. O parâmetro k é iniciado em 4.0. O valor de *fitness* de cada indivíduo do programa é o lnL indicado no cromossomo. A cada geração, ocorrem os sequintes eventos:

1. Calcula-se o lnL (escore) sem otimização de *branch*;
2. Indivíduos da população são classificados baseado na função de *fitness*;
3. Calcula-se a chance do indivíduo i deixar descendentes;

[1]Hasegawa, Kishino e Yano [17] unificaram dois modelos existentes de evolução de DNA: Kimura [19] e Felsenstein [13]. O modelo de Felsenstein é baseado em 4 parâmetros, relacionados às frequências de cada um dos nucleotídeos, e o modelo HKY utiliza um 5º parâmetro em sua modelagem, usado para distinguir a transição, quando há mutação entre bases purinas (A↔G) ou entre bases pirimidinas (T↔C), da transversão, quando uma base purina (A ou G) se torna pirimidina (C ou T).

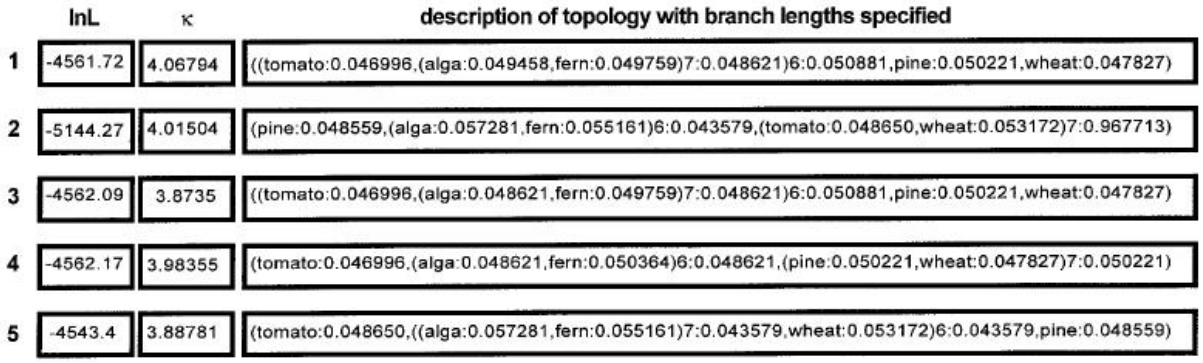


Figura 3.5: Representação de indivíduos de uma população, no algoritmo GAML [21]

4. O indivíduo com maior escore pode deixar e descendentes, e o restante deixará n-e. Esta é chamada de estratégia elitista do algoritmo;
5. O primeiro indivíduo é protegido de mutações para preservar o melhor da geração anterior, e os demais são passíveis de sofrer mutações, sendo chamados de indivíduos mutáveis;
6. Realiza mutações nos indivíduos mutáveis.

No item 3, a chance de um indivíduo deixar descendentes é calculada com a probabilidade $p(n-i+1)$, onde n é o total de indivíduos, i é a posição do indivíduo na lista ordenada pelo escore, p é o parâmetro tal que a soma das probabilidades seja um, ou seja, $\frac{2}{n(n+1)}$. Essa seleção é, portanto, realizada segundo a seleção por *rank*, citada na Seção 2.2.2.

Os eventos de mutação no passo 6 são:

- Todos os indivíduos sofrem mutação em uma quantidade aleatória de arestas, e o tamanho de cada uma é alterado por uma função baseada em uma distribuição γ, gaussiana, com parâmetro α;
- Mutações topológicas ocorrem com probabilidade m_1. A mutação consiste em transportar um ramo aleatório para outro ponto aleatoriamente da árvore;
- O parâmetro k é mutado com probabilidade m_2. A mutação também multiplica o parâmetro por um fator em uma distribuição γ;
- Recombinações ocorrem com probabilidade m_3. Utilizam o indivíduo mutante e um indivíduo da população dos pais. Isto permite a possibilidade de porções de dois indivíduos bons serem mantidas juntas. Esta mutação é ilustrada na Figura 3.6.

3.2.2 GARLI

GARLI significa *Genetic Algorithm for Rapid Likelihood Inference* e é um algoritmo desenvolvido com intenção de aumentar a velocidade e inferência do *Maximum Likelihood* [10]. O GARLI é derivado do GAML, possuindo assim alguns passos bastante similares. A função de *fitness* também utiliza os princípios do GAML.

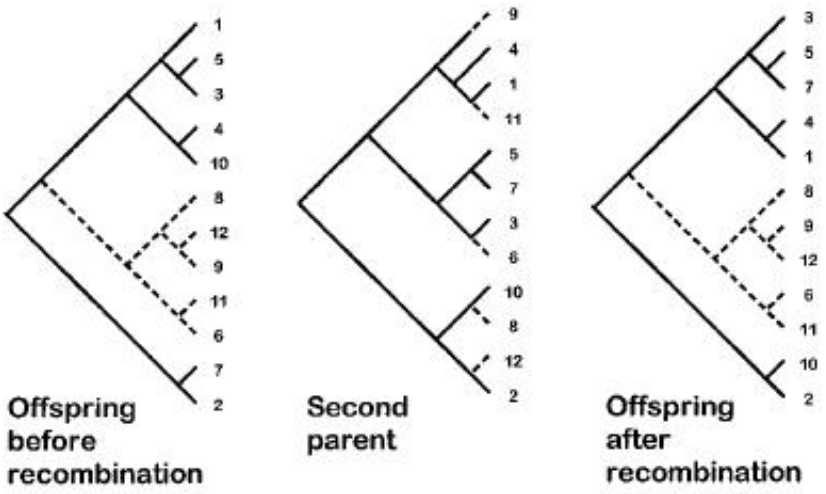


Figura 3.6: Mutação de Recombinação do GAML [21]

O algoritmo apresenta os passos descritos na Figura 3.7. As condições iniciais podem ser aleatórias ou fornecidas pelo usuário. Nos ciclos de cada uma das gerações, ocorrem os passos:

1. A função de *fitness* é usada para calcular o valor de *fitness* de cada indivíduo, similar ao GAML, sendo o valor de lnL;
2. O indivíduo com maior escore pode deixar k descendentes. Esta é a estratégia elitista;
3. Os remanescentes $N - k$ pais são escolhidos para serem descendentes, e até então, são cópias exatas. Esses indivíduos são selecionados com determinada probabilidade, definida no algoritmo;
4. Com exceção de 1 indivíduo, de melhor escore, todos os demais sofrem alguma operação de mutação;
5. Os escores são recalculados e o algoritmo para de acordo com o critério de parada.

Vamos explicar mais detalhadamente alguns dos passos acima. No passo 3, a probabilidade do indivíduo ser selecionado é dado pela fórmula

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j} \tag{3.1}$$

Cada indivíduo possui um valor, definido na função f, que é relativo à posição do indivíduo na tabela de escores e na distância desse escore em relação ao melhor escore da geração. A função f é definida da seguinte forma:

1. Defina o valor do maior escore;
2. Calcule o valor relativo da posição i ao maior escore. Ex: $lnL_r = lnL_i - lnL_{max}$;

3. Calcule $f = exp(s \cdot lnL_r)$ para cada indivíduo. O parâmetro s é chamado de parâmetro de intensidade, pois sua variação torna mais ou menos provável que indivíduos com escores maiores sejam selecionados;

4. A probabilidade de ser selecionado é proporcional a função f, conforme equação de p_i na Equação 3.1.

Todos os indivíduos que podem sofrer mutações recebem algum tipo de mutação disponível no algoritmo, tais como:

- Mutação topológica: inspira-se nos algoritmos *Robinson-Foulds distance* [30], que realizam a mutação considerando a distância de reconexão, que representa a distância em que o ramo escolhido irá ser inserido em relação à sua posição anterior, medida em quantidade de nós de distância. GARLI permite um controle na distribuição de rearranjos topológicos;
- *Model parameter mutation*: realiza mutação em algum dos 12 parâmetros da aplicação, relacionados ao funcionamento do algoritmo, que não iremos detalhar aqui;
- *Branch-length parameter mutation*: realiza mutação nos tamanhos das arestas, multiplicando por uma variável de distribuição γ e média 1.0. Um número de arestas a serem afetadas é sorteada e as arestas são escolhidas aleatoriamente.

Por fim, ao final de cada geração, o algoritmo verifica seus critérios de parada. O software encerra sua execução quando:

- Não houveram mudanças topológicas significativas (alta variação no escore) por um período específico de gerações; e
- O aumento total no escore por um período de *update intervals*, definido com parâmetro, deve ser menor que determinado valor; e
- O valor do parâmetro GA que controla o grau da otimização de *branchs* deve ter alcançado seu valor mínimo.

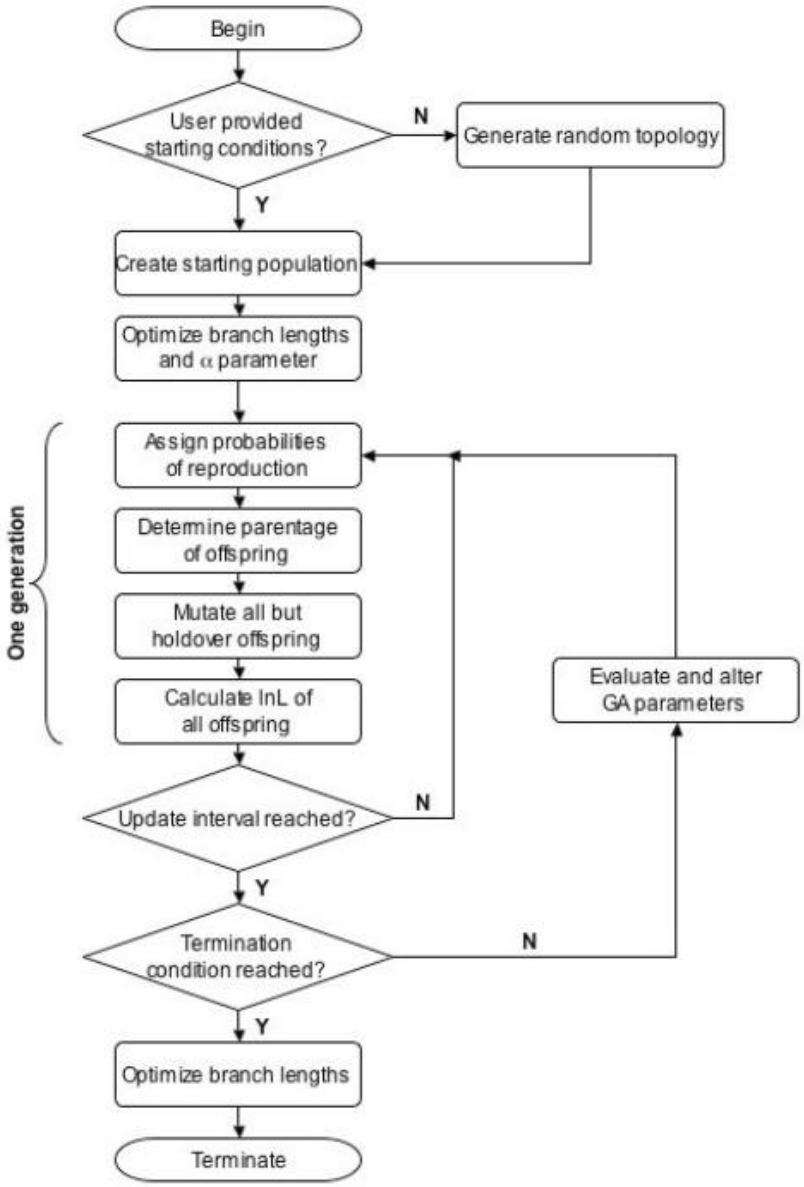


Figura 3.7: Diagrama de fluxo do software GARLI [10]

Capítulo 4

Algoritmo genético para filogenia viva

Neste capítulo, descrevemos o algoritmo genético para construção de filogenia viva, tendo como entrada uma matriz de características. Na Seção 4.1, apresentamos o método e os módulos do software. Na Seção 4.2, apresentamos as estruturas de dados utilizadas na implementação. Na Seção 4.3, detalhamos o algoritmo de cada módulo e calculamos as complexidades de tempo de cada módulo.

4.1 Descrição dos algoritmos

O fluxograma do algoritmo proposto pode ser visto na Figura 4.1, e é baseado no fluxograma apresentado na Figura 2.15.

O software foi implementado na linguagem C e compilado com o gcc versão 7.3.1 no linux, sistema operacional Fedora 27. O programa possui alguns módulos independentes para executar determinadas funções, e outras são executadas pelo algoritmo principal. Cada uma das etapas é detalhada a seguir.

- **Entrada**: o programa recebe como entrada uma matriz de características. Os demais itens são opcionais e servem para inicializar os parâmetros do algoritmo;
  `Uso: ./bin/ga-fitch [-p pop] [-g ger] [-e E] [-m M1 M2 M3] [-r R1] [-f esc] [-i int] [-v fv] [-a amin amax] matriz`
  onde
  - -p indica o tamanho da população (*default* 200),
  - -g indica o número de gerações (*default* 1000),
  - -e indica o tamanho da estratégia elitista (*default* 10% de P),
  - -m indica as probabilidades M1, M2 e M3 de ocorrerem as mutações 1, 2 e 3 (*default* x x x),
  - -r indica a probabilidade R1 de ocorrer a recombinação 1 (*default* 60),
  - -f indica o escore que interrompe o algoritmo caso a população o tenha alcançado (*default* 0),
  - -s indica a quantidade de gerações máximas que o algoritmo executa caso o melhor escore não seja alterado,

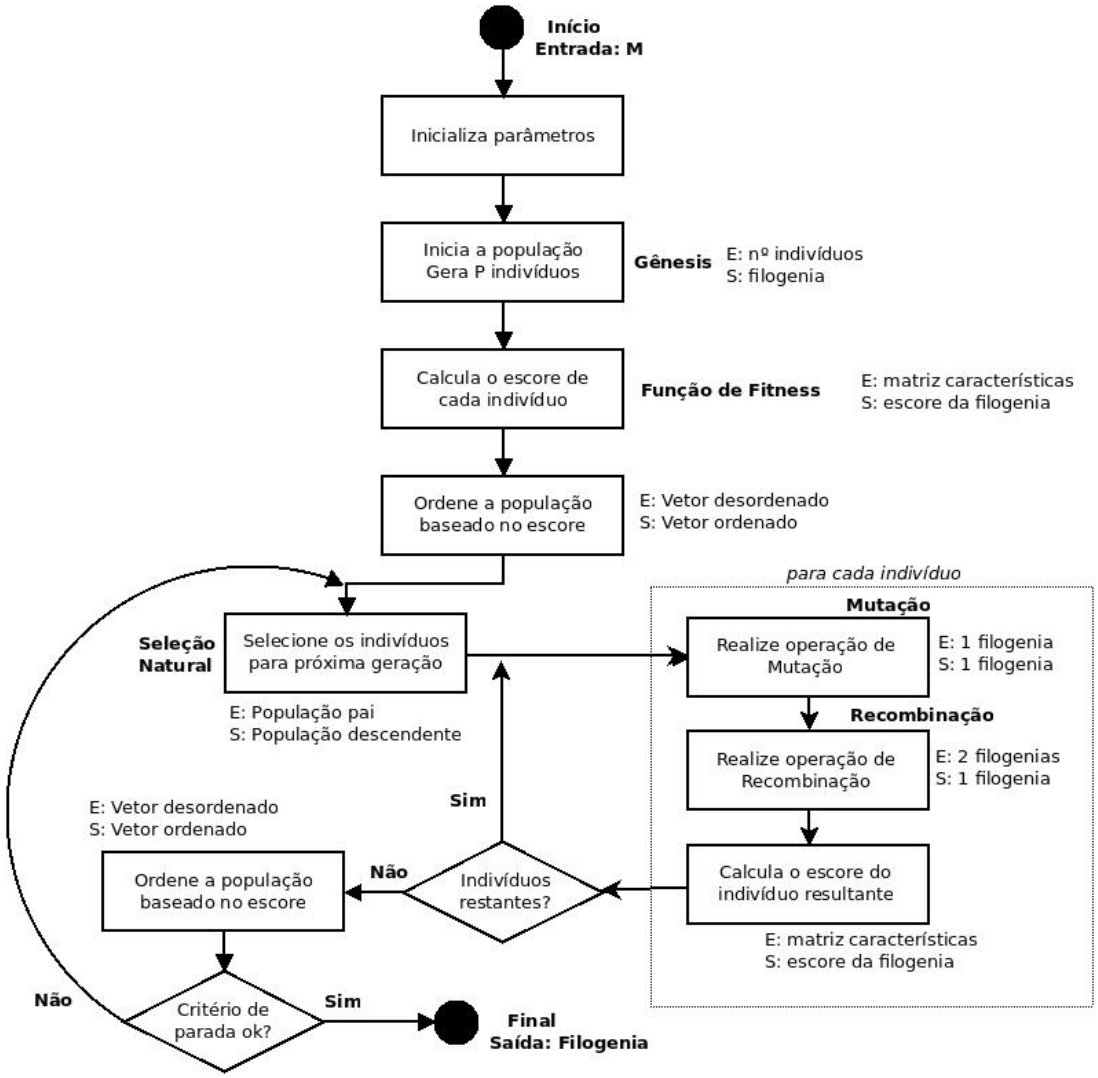


Figura 4.1: Fluxograma do algoritmo genético proposto para o problema da filogenia viva grande, tendo como entrada uma matriz de características

- -i indica um número de intensidade para a seleção natural baseada em rank (*default* 100),
- -v indica a porcentagem de tentativa de geração de nós vivos na população inicial (*default* 0),
- -a indica os valores mínimo e máximo da quantidade de nós vivos que é desejada na saída do programa,
- –verbose [0-2] indica a quantidade de impressão na saída no programa (*default* 0),
- –exclude-equal exclui todas as colunas da matriz de características que possuam a mesma característica para todos os indivíduos, pois tais colunas não influenciam nos cálculos de escore das funções de *fitness*,
- –sankoff informa que o algoritmo de sankoff deverá ser utilizado para calcular o escore da função de *fitness*, e é obrigatório que exista uma matriz de

custos no diretório data/custos, com todos os rótulos presentes na matriz de características.

- **Gênesis**: Nesse passo, o algoritmo inicializa a população, de tamanho *pop*. A geração é feita de forma aleatória, gerando filogenias binárias com quantidade de folhas iguais aos objetos de entrada da matriz M. O parâmetro -v indica a quantidade de nós vivos que podem surgir na geração inicial. O valor padrão (0) indica que não existirão nós vivos, e o valor 100 tenta inserir ao máximo nós vivos durante a criação. Esta etapa é executada por um módulo independente do programa principal;

- **Função de *Fitness***: Realiza o cálculo do escore de cada indivíduo segundo a função de *fitness*. A função de *fitness* é executada por um módulo independente, que realiza o cálculo do escore segundo o algoritmo de Fitch [14];

- **Seleção Natural**: O algoritmo utiliza uma seleção baseada em *rank*. Após ordenar a população de acordo com o escore, indivíduos mais bem posicionados possuem maior probabilidade de serem escolhidos. O primeiro colocado é preservado de mutações e uma quantidade E, informada pelo parâmetro de entrada, é copiada para a geração seguinte, sendo esta estratégia chamada de estratégia elitista. Caso o parâmetro -a seja usado, esta quantidade é copiada da melhor filogenia que esteja dentro da faixa informada. O restante dos indivíduos são escolhidos a partir de uma função densidade de probabilidades baseada em sua posição no *rank*. O parâmetro fornecido com -i ajusta essa probabilidade, onde 1 representa a mesma chance de todos serem escolhidos, 100 representa uma distribuição linear que aumenta uniformemente com peso 1 a cada posição no *rank*, e valores maiores passam a aumentar as chances de indivíduos mais bem posicionados serem escolhidos;

- **Mutação**: Define qual operação de mutação o indivíduo irá sofrer, de acordo com as probabilidades M1, M2 e M3;

- **Recombinação**: Realiza a operação de recombinação, de acordo com a probabilidade R1;

- **Final**: o critério de parada do algoritmo é o término das G gerações ou, caso o parâmetro -f tenha sido fornecido, o programa é interrompido caso escore *esc* seja alcançado. O algoritmo retorna o indivíduo de melhor escore.

Através das probabilidades $M1$, $M2$, $M3$ e $R1$, definimos a chance de ocorrer cada uma das operações de mutação. Será sorteado M1 ou M2 ou M3, e após a mutação, será sorteado R1. Cada operação é realizada por um módulo independente, e são descritas da seguinte forma:

- **Mutação 1**: realiza uma troca de duas subárvores escolhidas aleatoriamente. Seleciona um nó interno, retira a subárvore correspondente e a reinsere em outro ponto da árvore, aleatoriamente. As subárvores não podem estar contidas uma na outra, nem conter a raiz. Em caso de presença de ancestrais vivos em alguma subárvore, eles são naturalmente migrados. A Figura 4.2 mostra dois exemplos da operação de mutação, trocando a posição de uma subárvore em uma filogenia convencional e em uma filogenia viva. A representação em string da filogenia é uma adaptação do

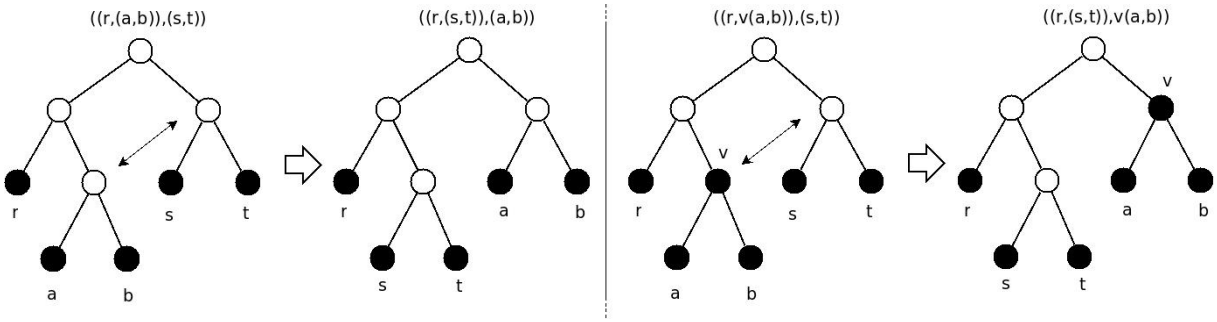


Figura 4.2: Exemplos de mutação 1, para uma árvore sem ancestrais vivos e para uma árvore com um ancestral vivo

formato newick [43], em que o ancestral vivo é representado imediatamente antes dos parênteses.

- **Mutação 2**: realiza uma troca de rótulos de forma aleatória. Seleciona dois nós que possuam rótulos, folhas ou nós vivos, e os troca de posição. Esta operação mantém a topologia da árvore, apenas o rótulo é trocado entre os nós, e as subárvores correspondentes são mantidas. Em caso de nós vivos, a troca ocorre alterando apenas os rótulos dos nós, como ocorre na troca entre nós nas folhas. A Figura 4.3 mostra dois exemplos, entre nós folhas e entre um nó vivo e um nó folha.

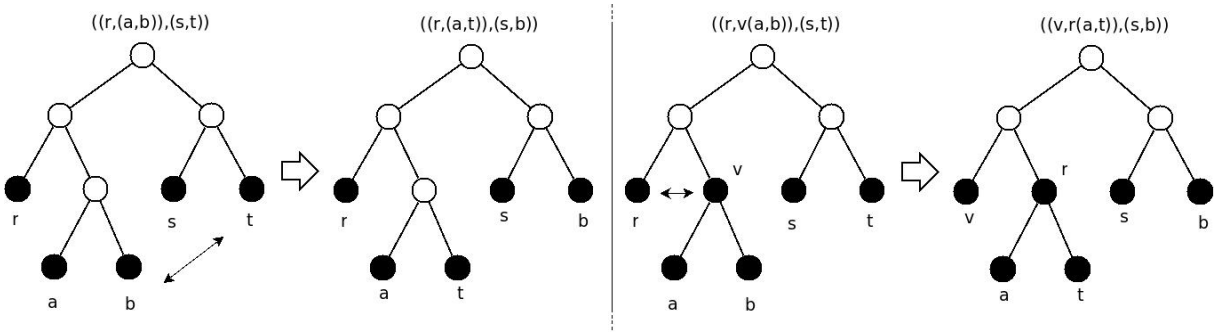


Figura 4.3: Exemplos de mutação 2, para uma árvore sem ancestrais vivos e para uma árvore com um ancestral vivo

- **Mutação 3**: realiza uma transformação de nós folhas para nós vivos ou vice-versa. Recebe um valor *prob* de 0 a 100 que define a probabilidade de ocorrer uma alteração que aumente o número de nós vivos, ou mova os existentes em direção à raiz, ou que desça nós vivos em direção às folhas, eventualmente criando uma nova folha. O valor padrão dessa probabilidade é fixado em 50% e é modificado dinamicamente de acordo com a entrada -a. O valor *prob* representa as chances de ocorrer uma das operações de subida (operação 1 ou 2) ou uma das operações de descida (operações 3 ou 4). A partir de então, um nó que seja compatível com uma das operações é sorteado. Dependendo da topologia da árvore, é possível que essa mutação não realize nenhuma operação. Isso pode ocorrer quando, por exemplo, o nó sorteado possui ancestrais vivos, impossibilitando a subida em direção à raiz, e descendentes

vivos não-folhas, impossibilitando a descida até as folhas. Caso não seja desejado trabalhar com filogenias vivas, é necessário iniciar a mutação com 0 (0% de chances de ocorrer). A Figura 4.4 mostra as 4 operações que são possíveis de ocorrerem.

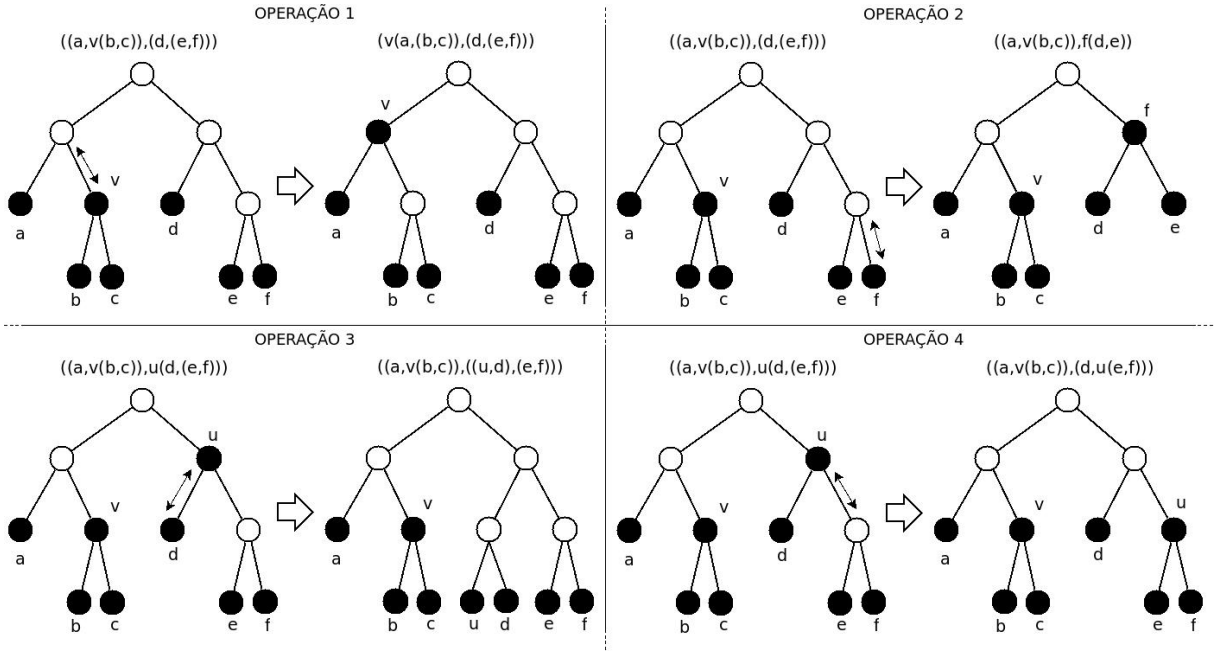


Figura 4.4: Exemplos de mutação 3, para uma árvore sem ancestrais vivos e para uma árvore com um ancestral vivo

- **Recombinação 1**: realiza uma operação de recombinação, similar à operação do software GAML [21], e a Figura 3.6 ilustra a operação. Utiliza um indivíduo pai e um da geração descendente. Seleciona aleatoriamente um indivíduo da geração pai que esteja entre os 50% melhores da população, seleciona uma subárvore do pai, retira as folhas correspondentes na árvore descendente, e insere a subárvore do pai na árvore descendente.

4.2 Estrutura de dados

Existem três estruturas de dados principais, descritas em seguida.

- Um tipo chamado de **notype**, para representar um nó da árvore. Uma árvore passa, então, a ser representada por um vetor dessa estrutura. Cada nó possui informações de rótulo, pai, filho, irmão;

```
typedef struct {
int info;
int right;
int left;
int father;
int son;
```

```
int bro;
} notype;
```

- Um tipo chamado de **filoinfo**, para guardar informações sobre a árvore, como quantidade de nós, folhas e a *string newick* que a representa;
```
typedef struct {
int qtdn;
int qtdf;
char newick[NUMNODES];
} filoinfo;
```

- Um tipo chamado **cromossomo**, usado pelo algoritmo genético, com as estruturas necessárias para a árvore e um valor de escore.
```
typedef struct {
filoinfo info;
int score;
notype nos[NUMNODES];
} cromossomo;
```

O algoritmo armazena uma matriz de características $M_{n \times m}$, com n objetos e m características. A cada geração, o algoritmo realiza operações em um vetor de tamanho p, população, contendo uma estrutura *notype* para cada nó da árvore. Para n indivíduos, representamos a árvore binária com tamanho proporcional a n, de até $n \times 2 - 1$ nós. Utilizamos uma matriz de custos para o cálculo de Sankoff de tamanho $n \times |\Sigma|$, onde $|\Sigma|$ representa o alfabeto de rótulos possíveis. A complexidade de espaço total é de $n \times m + n \times |\Sigma| + p \times n$.

4.3 Algoritmo de cada módulo

A implementação foi dividida nas partes a seguir.

- Programa principal: algoritmo genético, seleção por *rank*, sorteio de mutações indicadas na Figura 4.1.
- Módulo para geração de árvore
- Módulo para cálculo do escore Fitch
- Módulo para cálculo do escore Sankoff
- Módulo para rotulação da árvore
- Módulo para Mutação 1
- Módulo para Mutação 2
- Módulo para Mutação 3
- Módulo para Recombinação 1

Descreveremos os algoritmos em função de suas variáveis de entrada. Deve ser recebida uma matriz de características $M_{n \times m}$, com m colunas e n linhas. As n linhas representam os n indivíduos de uma filogenia, e as m colunas os dados da matriz de características. Definimos um indivíduo $TREE$ como uma estrutura formada pelas duas estruturas de dados *filoinfo* e *notype*, definidas na Seção 4.2, que contêm as informações necessárias para representar uma filogenia. As demais variáveis são parâmetros do algoritmo, já detalhados na Seção 4.1: P, G, E, M_1, M_2, M_3, R_1, ESC, I e FV.

4.3.1 Módulo Geração de árvores

O módulo de geração de árvores é responsável pela gênesis do algoritmo genético, a geração inicial de indivíduos. Para gerar árvores com n nós, um vetor de tamanho n é gerado e embaralhado. Cada posição do vetor é considerada uma subárvore. O módulo de geração de árvores une sempre as subárvores das posições n e $n-1$, até que sobrem apenas os dois indivíduos das posições 1 e 2, que são unidos e geram a árvore final.

O parâmetro FV indica a probabilidade de inserir um nó vivo. Caso, na união das subárvores n e $n-1$, apenas uma subárvore contenha apenas um nó, é possível inserir esse nó como um ancestral vivo da outra subárvore. Caso FV seja 0, nenhum nó vivo será gerado, e caso seja 100, sempre será gerado um nó vivo caso as duas subárvores que serão unidas atendam a essa condição.

O algoritmo do módulo pode ser vista no Algoritmo 4.1.

Algorithm 4.1 Módulo Geração de árvores

Entrada: O número de indivíduos n e o parâmetro FV
Saída: Um indivíduo $TREE$

1: Gere um vetor de tamanho n com um indivíduo em cada possição, atribuído de forma aleatória
2: **while** $n > 1$ **do**
3: Una as subárvores das posições n e $n-1$. Sorteie um número aleatório r de 0 a 99.
4: **if** $FV > r$ AND n ou $n-1$ contém apenas um nó **then**
5: Una n a $n-1$ de modo a gerar um nó vivo em um dos nós internos da subárvore
6: **else**
7: Una n a $n-1$ com a adição do nó r, nova raiz da subárvore resultante
8: **end if**
9: Sorteie uma posição j, de 1 a $n-2$ para inserir a árvore resultante. Insira a subárvore da posição j na posição $n-1$.
10: Faça $n := n-1$
11: **end while**
12: **return** a árvore restante do vetor na forma de um indivíduo $TREE$

Para geração de árvores, consideramos cada um dos n indivíduos de entrada como uma subárvore e unimos cada uma delas até que reste apenas a árvore final. A linha 1 possui um total de n passos, e a linha 2 possui um laço que é executado $n-1$ vezes. Os passos internos do laço são constantes. A complexidade de tempo do módulo é de $O(n)$.

4.3.2 Módulo Mutação 1

A mutação 1 realiza uma troca da posição de duas subárvores, sorteadas aleatoriamente. O algoritmo do módulo pode ser visto no Algoritmo 4.2.

Algorithm 4.2 Módulo Mutação 1

Entrada: Um indivíduo $TREE$
Saída: O indivíduo $TREE$ modificado

1: Sorteie dois nós n_1 e n_2 que não estejam na raíz e nem sejam da mesma subárvore
2: Troque os nós e a subárvore correspondente de posição
3: **return** a árvore de entrada modificada

As operações do módulo da Mutação 1 modificam a topologia recebida. A operação de sorteio realizado na linha 1 possui um número de tentativas máximas fixado. As operações de troca de nós são realizadas em tempo constante, portanto, o módulo possui complexidade de tempo de $O(1)$.

4.3.3 Módulo Mutação 2

A mutação 2 realiza uma troca de dois nós, escolhidos aleatoriamente. O algoritmo do módulo pode ser visto no Algoritmo 4.3.

Algorithm 4.3 Módulo Mutação 2

Entrada: Um indivíduo $TREE$
Saída: O indivíduo $TREE$ modificado

1: Sorteie dois nós n_1 e n_2 que possuam um rótulo atribuído a eles
2: Troque o rótulo dos dois nós
3: **return** a árvore de entrada modificada

As operações do módulo da Mutação 2 trocam os rótulos presentes em dois nós. As operações são realizadas em tempo constante, portanto, o módulo possui complexidade de tempo de $O(1)$.

4.3.4 Módulo Mutação 3

A mutação 3 foi desenhada exclusivamente para operar com nós vivos. O algoritmo do módulo pode ser visto no Algoritmo 4.4.

As operações do módulo da Mutação 3 trocam a posição de um nó folha para vivo e vice-versa. As operações de 1 a 4 são realizadas em tempo constante. A etapa de seleção do nó adequado possui um número máximo n de tentativas, portanto, o módulo possui complexidade de tempo de $O(n)$.

4.3.5 Módulo Recombinação 1

A recombinação foi inspirada no algoritmo GAML [21]. O algoritmo do módulo pode ser visto no Algoritmo 4.5.

Algorithm 4.4 Módulo Mutação 3

Entrada: Um indivíduo $TREE$ e uma probabilidade $prob$ de subir ou descer
Saída: O indivíduo $TREE$ modificado

1: Sorteie se a operação será de movimento em direção à raiz ou em direção às folhas, de acordo com $prob$
2: Sorteie um nó n_1 que possua um rótulo atribuído a ele e que seja compatível com a operação de movimento em direção à raiz (op_1 ou op_2) ou movimento em direção às folhas (op_3 ou op_4)
3: Defina uma das quatro operações para o nó sorteado
4: **if** operação = op_1 **then**
5: Transfira o nó vivo sorteado ao nó pai
6: **else if** operação = op_2 **then**
7: Transfira o nó folha sorteado ao nó interno acima, removendo os nós desnecessários e reorganizando a árvore
8: **else if** operação = op_3 **then**
9: Transfira o nó vivo sorteado ao filho da esquerda, gerando uma nova folha caso necessário
10: **else**
11: op_4: Transfira o nó vivo sorteado ao filho da direita, gerando uma nova folha caso necessário
12: **end if**
13: **return** a árvore de entrada modificada

A recombinação recebe duas filogenias, uma da geração de descendentes e uma da geração pai, e as une em um só indivíduo. A linha 1 seleciona uma subárvore, em uma operação que é dependente do tamanho da árvore, dependente de n. A linha 2 percorre os nós da topologia para retirar folhas. O total de nós percorridos é de $2 \cdot n - 1$. A linha 3 refaz a estrutura da árvore, operação que tem complexidade de tempo $O(n)$. Como cada passo tem, no máximo, complexidade de tempo de $O(n)$, esse módulo possui complexidade total de $O(n)$.

4.3.6 Módulo para escore: resolvendo o problema da filogenia pequena por Fitch

O problema da filogenia pequena pode ser resolvido aplicando o algoritmo de Fitch [14] e retornando seu escore como um *fitness* a ser avaliado pelo algoritmo genético. Quanto menor o valor desse escore, melhor o *fitness* do indivíduo, pois mais adequada será a filogenia para representar as espécies de entrada, visto que o número de mutações necessárias é o menor possível.

O valor do escore é calculado pela fase *bottom-up*. A descrição pode ser vista no Algoritmo 4.6.

A modificação realizada no algoritmo original é representada na linha 8. Caso um nó interno i seja vivo, o algoritmo pode incrementar o escore em 1 ou 2, dependendo da intersecção do estado de i, já definido, com cada um dos seus filhos. Evolutivamente, isso significa que, caso o estado de i não esteja presente em nenhum dos seus filhos,

Algorithm 4.5 Módulo Recombinação 1

Entrada: Dois indivíduos $TREE_o$ e $TREE_{pai}$
Saída: Recombinação da entrada $TREE$

1: Selecione uma subárvore da filogenia $TREE_{pai}$ e a denomina de *subtreep*
2: Percorra $TREE_o$ e retire os nós que contenham rótulos de *subtreep*
3: Refaça a árvore $TREE_o$ com os nós que restaram
4: Verifique locais possíveis de inserção de *subtreep* em $TREE_o$
5: Insira *subtreep* em $TREE_o$, sorteando o local de inserção caso haja várias possibilidades
6: **return** a árvore $TREE$ resultante da recombinação

seriam necessárias duas mutações para que essa filogenia represente as relações entre seus indivíduos. A fase *top-down* do algoritmo não é alterada em relação ao algoritmo original, que foi mostrado no Algoritmo 3.2.

O algoritmo realiza suas operações um total de m vezes, representado pelo laço externo iniciado na linha 1. A linha 2 precisa percorrer toda a árvore para buscar os rótulos e definir R_i, consumindo um tempo $O(n)$. Dentro do laço, é necessário percorrer a árvore das folhas à raiz, indicado na linha 4, e realizar as operações indicadas, todas em tempo constante. Assim, a complexidade de tempo total do módulo é de $O(m * n)$.

4.3.7 Módulo para escore: resolvendo o problema da filogenia pequena por Sankoff

O problema da filogenia pequena também pode ser resolvido aplicando o algoritmo de Sankoff [35] e retornando seu escore como um *fitness* a ser avaliado pelo algoritmo genético. Analogamente ao algoritmo de Fitch, quanto menor o valor desse escore, melhor o *fitness* do indivíduo. Além da matriz de entrada, é necessário informar uma matriz de custos que possua todos os rótulos presentes na matriz de características, em que cada rótulo representa uma característica. A quantidade de rótulos é representada pela letra l no algoritmo.

O valor do escore é calculado pela fase *bottom-up*. A descrição pode ser vista no Algoritmo 4.7. O algoritmo define uma matriz de custos s para cada nó, onde $s(i, a)$ é o custo de rotular o nó i com o rótulo a, presente em um dos l rótulos da matriz de entrada, s_i é o estado definido para o nó i, e o valor de $cost(a, b)$ é dado pela matriz de entrada de custos.

A modificação realizada no algoritmo original é representada na linha 11. Caso um nó interno i seja vivo, o algoritmo só calcula o custo referente ao rótulo definido em i, mantendo os demais valores em ∞.

O algoritmo realiza suas operações um total de m vezes, representado pelo laço externo iniciado na linha 1. A linha 2 precisa percorrer toda a árvore para buscar os rótulos e definir R_i, consumindo um tempo $O(n)$. Dentro do laço, é necessário percorrer a árvore das folhas à raiz, indicado na linha 7, e realizar as operações indicadas, todas em tempo constante, visto que a quantidade de rótulos é fixa e fornecida pela matriz de custos de entrada. Assim, a complexidade de tempo total do módulo é de $O(m * n)$.

Algorithm 4.6 Algoritmo de Fitch live: *Bottom-up*

Entrada: Matriz $M_{n \times m}$ e uma filogenia $TREE$
Saída: Inteiro *score* representando o escore

1: **for all** coluna j da matriz de entrada **do**
2: Inicialize cada nó rotulado com s_i definindo $R_i = \{s_i\}$.
3: $score := 0$
4: **for all** nó i das folhas até a raiz (pós-ordem) **do**
5: **if** nó i é um nó não vivo (ancestral hipotético) **then**
6: Determine R_i de cada nó interno *não vivo* i com filhos j, k da seguinte forma:

$$R_i = \begin{Bmatrix} \text{Se } R_j \cap R_k \neq \emptyset \rightarrow R_j \cap R_k \\ c.c. \rightarrow R_j \cup R_k, score := score + 1 \end{Bmatrix}$$

7: **else**
8: Determine R_i de cada nó interno *vivo* i de estado s_i com filhos j, k da seguinte forma:

$$R_i := \{s_i\}$$

9: Se $R_i \cap R_j = \emptyset \rightarrow score := score + 1$
10: Se $R_i \cap R_k = \emptyset \rightarrow score := score + 1$
11: **end if**
12: **end for**
13: **end for**
14: **return** *score*

4.3.8 Módulo para rotulação dos nós internos da árvore

A rotulação dos nós interiores da árvore correspondentes aos ancestrais hipotéticos é realizada pela fase de descida do algoritmo de Sankoff. Nesta fase, o algoritmo irá rotular apenas os nós internos que não sejam vivos, seguindo o mesmo algoritmo original, já descrito na Seção 3.1. Esse módulo de rotulação é executado apenas após a escolha da árvore final, uma única vez, antes da saída. O módulo não é executado ao longo da execução do algoritmo genético, pois não possui contribuição no escore, que é calculado na fase *bottom-up*, e usado para avaliar os indivíduos ao longo das gerações. O Algoritmo 4.8 descreve o módulo de rotulação dos nós internos da filogenia.

Note que o algoritmo de rotulação não sofre alteração em relação a topologias com nós vivos. Na fase *top-down*, os nós vivos mantém a mesma rotulação fixada na fase *bottom-up*. A rotulação realizada pelo algoritmo de Fitch também não é alterada em relação ao algoritmo original, que foi mostrado no Algoritmo 3.2.

O algoritmo de rotulação possui a fase de subida, na linha 1, com complexidade de tempo de $O(m * n)$. Como o laço da linha 4 percorre os nós em $O(n)$, a complexidade total de tempo do módulo se mantém em $O(m * n)$.

4.3.9 Programa principal

O programa principal é descrito no Algoritmo 4.9.

Algorithm 4.7 Algoritmo de Sankoff live: *Bottom-up*

Entrada: Matriz $M_{n \times m}$, matriz de custos $C_{l \times (l+1)}$ e uma filogenia $TREE$
Saída: Inteiro *score* representando o escore

1: **for all** coluna j da matriz de entrada **do**
2: **for all** nós i com rótulos definidos $s_i = a$ **do**
3: $s(i, l) = \infty$ para cada $l \neq s_i$
4: $s(i, a) = 0$
5: **end for**
6: $score := 0$
7: **for all** nó i das folhas até a raiz (pós-ordem) **do**
8: **if** nó i é um nó não vivo (ancestral hipotético) **then**
9: Para cada rótulo l_k, $1 \leq k \leq |\Sigma|$

$$s(i, l) = \sum_{v \text{ filho de } i} (cost(l, b_v) + s(v, b_v)$$

$$\text{tal que } b_v = min\{s(v, r_j) + cost(l, r_j)\},\ 1 \leq j \leq |\Sigma|$$

10: **else**
11: Realize a operação para l tal que $s_i = l$

$$s(i, l) = \sum_{v \text{ filho de } i} (cost(l, b_v) + s(v, b_v)$$

$$\text{tal que } b_v = min\{s(v, r_j) + cost(l, r_j)\},\ 1 \leq j \leq |\Sigma|$$

12: **end if**
13: **end for**
14: Incremente *score* com o menor valor presente em $s(raiz, l)$
15: **end for**
16: **return** *score*

As linhas 1 e 2 iniciam vetores de tamanho P. Na linha 3, temos um laço de tamanho P e internamente a geração de uma árvore executando em tempo $O(n)$ e cálculo de escore executando em tempo $O(m*n)$, totalizando uma complexidade de $O(P*m*n)$. A linha 8 possui o laço principal do programa, de tamanho G. Dentro dele, temos a seleção natural, que percorre toda a população atribuindo pesos a cada indivíduo que será escolhido, e ao selecionar a nova geração, percorre o vetor de pesos para realizar a operação. A complexidade de tempo da linha 9 é de $O(P^2)$. Na linha 10, para cada indivíduo da população, é selecionada uma operação de mutação e/ou recombinação, sendo que, no pior caso, temos uma complexidade de $O(n)$, o que nos dá um total de $O(P * n)$ para o laço. Na linha 25, temos um laço que percorre cada indivíduo calculando seu escore, o que nos dá uma complexidade de tempo de $O(P * m * n)$. As linhas 7 e 29 possuem uma complexidade de tempo de pior caso de $O(P^2)$. Temos uma complexidade total de tempo dependentes dos valores de P, G, n e m. Podemos dizer que a complexidade é de $O(G * P^2 + G * P * m * n)$ ou, em casos em que $P > n$, de $O(G * P^2 * m)$.

Algorithm 4.8 Algoritmo de Sankoff live: *Top-down*

Entrada: Matriz $M_{n \times m}$, matriz de custos $C_{l \times (l+1)}$ e uma filogenia $TREE$
Saída: Inteiro *score* representando o escore

1: Realiza a fase *bottom-up*, descrita pelo Algoritmo 4.7
2: Percorra a árvore da raiz até as folhas (pré-ordem)
3: Defina $s_{root} = l \mid min_l s(root, l)$
4: **for all** Nós n **do**
5: **if** n é um nó interno com filhos n_1 e n_2 **then**
6: $s_{n1} = l \mid min_l s(n_1, l) + cost(l, s_n)$
7: $s_{n2} = l \mid min_l s(n_2, l) + cost(l, s_n)$
8: **end if**
9: **end for**

Algorithm 4.9 Algoritmo genético para construção de filogenia viva GA-LF

Entrada: Matriz $M_{m \times n}$ e os parâmetros P, G, E, M_1, M_2, M_3, R_1, ESC, I, FV
Saída: Indivíduo $TREE$ de melhor escore.

1: Inicialize *popt* := cromossomo[P] para população atual
2: Inicialize $offspring$:= cromossomo[P] para população descendente
3: **for all** indivíduo i em *popt* **do**
4: Gere a árvore chamando geraArvore(FV,$MATRIZ$)
5: Calcule o escore com fitch ou sankoff(i,$MATRIZ$)
6: **end for**
7: Ordene *popt* utilizando *quicksort*
8: **for** $g := 1$ **to** G **do**
9: Realize a seleção natural, copiando uma quantidade E do primeiro colocado (ou melhor colocado da faixa de nós vivos) de *popt* para $offspring$ e selecionando o restante de acordo com I
10: **for all** indivíduo i em $offspring$ **do**
11: Sorteie se e qual mutação ocorrerá, de acordo com M_1, M_2, M_3
12: **if** M1 **then**
13: mutacao1(i)
14: **else if** M2 **then**
15: mutacao2(i)
16: **else if** M3 **then**
17: mutacao3(i)
18: **end if**
19: Sorteie se a recombinação R1 ocorrerá, de acordo com R_1
20: **if** R1 **then**
21: Sorteie *pai* da geração em *popt* para realizar a recombinação com i
22: recombinacao1(i,*pai*)
23: **end if**
24: **end for**
25: **for all** indivíduo i em $offspring$ **do**
26: Calcule o escore com fitch ou sankoff(i,$M_{m \times n}$)
27: Copie i para a geração *popt*
28: **end for**
29: Ordene *popt* utilizando *quicksort*
30: **end for**
31: **return** a melhor árvore i (menor escore)

Capítulo 5

Resultados

Neste capítulo, na Seção 5.1, descrevemos como foram escolhidos os parâmetros para o algoritmo. Na Seção 5.2, discutimos os estudos de caso realizados. O primeiro foi realizado utilizando como matriz de entrada um alinhamento dos vírus H1N1 e H3N2 de diversos países, sequenciados no mesmo ano (2016). O segundo foi realizado utilizando dados de uma região do vírus HIV de um mesmo paciente, durante diversas visitas ao longo de 12 anos. A intenção é observar como a filogenia é representada com e sem ancestrais vivos, e como os diversos indivíduos de entrada são agrupados em uma filogenia viva.

5.1 Parâmetros

Para testar o comportamento do algoritmo de acordo com cada parâmetro de entrada, foram realizados vários experimentos. Observamos que em todos os experimentos usamos filogenias binárias. Nesta seção, discutiremos os resultados de acordo com três entradas:

- $M_{34\times306}$: uma matriz formada por 34 organismos, representados em cada linha, e 306 características, representadas em cada coluna. Os dados foram extraídos de um alinhamento múltiplo de sequência (AMS). Essa entrada foi utilizada para os testes iniciais, e a matriz está disponível no Apêndice A;
- $M_{75\times759}$: a matriz foi gerada a partir de 75 indivíduos (linhas) e possui 759 características (colunas). Corresponde a um AMS referente à RNA polimerase PB2, obtidos em 2016, grande parte nos EUA, mas também em países como Rússia, Coréia do Sul, Taiwan e China. Todas as sequências correspondem ao vírus humano H1N1 ou H3N2. Os dados foram extraídos do *NCBI Influenza Virus Sequence Database* [2];
- $M_{52\times765}$: a matriz foi gerada a partir de 52 indivíduos (linhas) e possui 765 características (colunas). Corresponde a um AMS referente ao setor C2-V5 do gene *env* do vírus HIV. A entrada corresponde a amostras de um mesmo paciente, em visitas variadas ao longo de 12 anos. A origem dos dados são citados por Shankarappa [37] e estão disponíveis no alinhamento 1 do site *HIV Databases* [1];
- $M_{136\times680}$: a matriz foi gerada a partir de 136 indivíduos (linhas) e 680 características. Corresponde ao mesmo AMS da matriz do item anterior, referentes ao vírus HIV. Os dados para essa matriz foram extraídos e estão disponíveis no *GenBank*

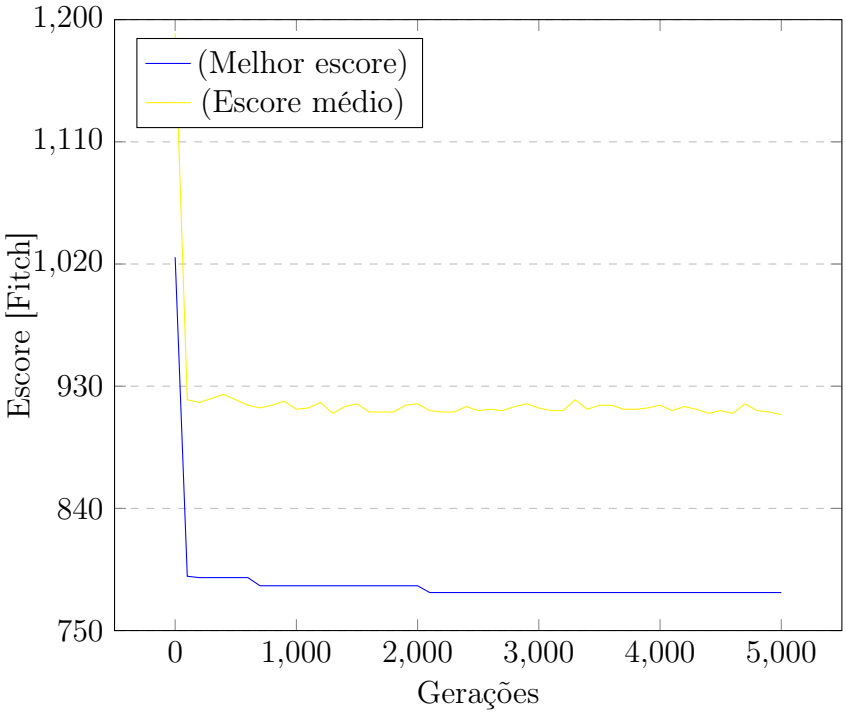


Figura 5.1: Melhor escore e escore médio da população ao longo das gerações para M1

database, com os números de acesso AF137629 a AF138163, AF138166 a AF138263 e AF138305 a AF138703.

Os testes foram realizados em um computador com processador Intel core-i5, 4GB de memória RAM, HD SSD de 240 GB e sistema operacional Fedora.

5.1.1 Avaliação inicial de parâmetros para filogenia

Para compreender o papel da mutação e da recombinação, foram realizados experimentos que analisam a evolução do escore mínimo e da média de escore da população, ao longo das gerações. Os experimentos foram realizados com filogenia tradicional, apenas para fixar valores iniciais para diversos parâmetros. Os gráficos das Figuras 5.1 e 5.2 mostram essas análises para uma execução, com apenas operações de Mutação 1 e com apenas operações de Recombinação 1, respectivamente. Foi utilizada como entrada a matriz $M_{34\times306}$.

É possível perceber que a Mutação 1 decresce mais rapidamente ao melhor escore, enquanto a Recombinação 1 permite diminuir mais rapidamente a média de escore da população.

Um outro tipo de teste realizado foi o que utiliza o parâmetro -f. Neste teste, foi incluído um critério de parada para que o software interrompa a partir de determinado escore atingido. Assim, podemos traçar um gráfico que representa no eixo x o escore a ser atingido e no eixo y a quantidade de gerações necessárias para atingir cada valor de escore. Para cada entrada x, foram realizadas dez execuções, e o valor de y representa a média dos resultados. Foi calculado ainda o desvio padrão para análise da variação dos resultados. A Figura 5.3 ilustra as gerações necessárias para atingir cada escore, para

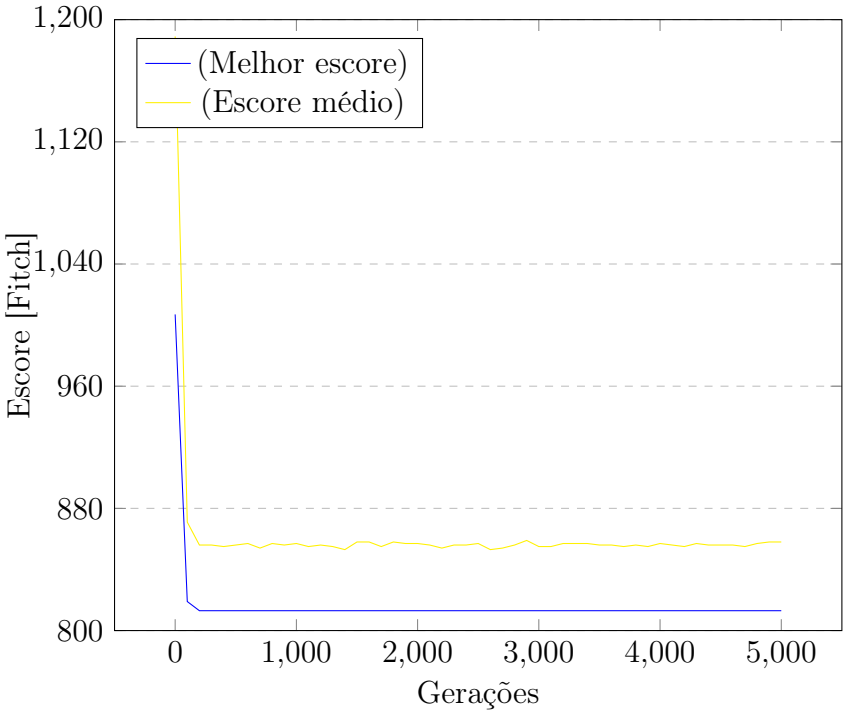


Figura 5.2: Melhor escore e escore médio da população ao longo das gerações para R1

uma determinada entrada, e para as mutações indicadas: $M1$, $M2$ ou $M1 + M2$. A recombinação não foi utilizada nesse teste. Foi mantida uma população de tamanho 200 e um máximo de 10.000 gerações.

É possível notar o papel da Mutação 1 para diminuir o escore mais rapidamente. A Mutação 2 não realiza uma mudança na topologia da árvore, o que faz com que a população não consiga atingir escores mais baixos, em muitos casos.

Foram realizados testes variando o parâmetro -i *int*, que modifica a seleção por *rank*. Com o uso da elite na proporção de 10% da população, não foram encontrados resultados significativos, e o parâmetro foi mantido com seu valor padrão, equivalente a -i 100, nos demais testes. Quanto aos valores de população e elite, foram encontrados inicialmente os valores de 200 para P e 20 para E, que foram reexaminados ao final dos testes. Para execuções com -f, que indica um critério de parada ao atingir determinado escore, foi utilizado o valor 5.000 para G, que indica o número máximo de gerações que irão ocorrer, caso não seja encontrado um escore menor que o informado.

5.1.2 Avaliação de parâmetros com filogenia viva

A possibilidade de existirem filogenias com nós vivos na população é realizada pela utilização dos parâmetros -v VF, que indica nós vivos na geração inicial, e da Mutação 3, capaz de alterar a topologia e gerar nós internos vivos.

Com o algoritmo genético funcionando para filogenias vivas, foram realizados alguns testes com cada uma das mutações, sendo realizado a comparação de acordo com o parâmetro FV, que indica a quantidade de nós vivos na geração inicial da população. Foi utilizada como entrada a matriz $M_{75\times759}$.

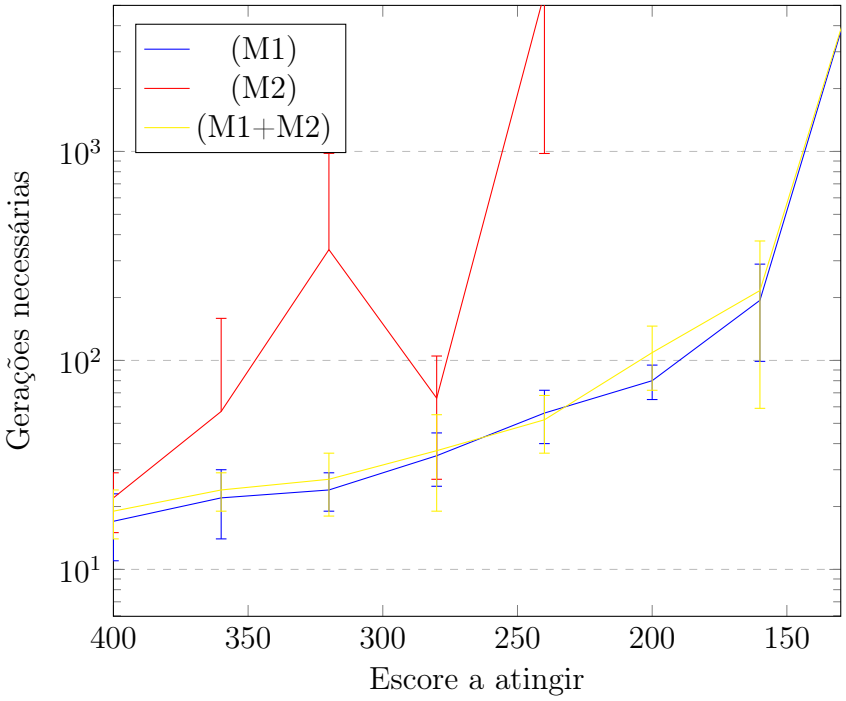


Figura 5.3: Quantidade de gerações necessárias para cada combinação de M1, M2 e M1+M2

Com a utilização em 100% da Mutação 1, comparamos as gerações necessárias para atingir cada escore de acordo com a utilização do parâmetro -v FV. O resultado pode ser visto na Figura 5.4.

Da mesma maneira, comparamos o desempenho da Mutação 2 com diferentes valores de FV. Nesse caso, com poucos nós vivos, o desempenho já fica bastante comprometido, uma vez que muitas gerações são necessárias para atingir escores ainda distantes dos mais baixos obtidos, como pode ser visto na Figura 5.5.

A utilização de nós vivos em conjunto com as Mutações 1 e 2 não é suficiente para fazer variar a população e chegar aos valores mais baixos de escores. A utilização da Recombinação 1 também não é suficiente, pois remove os nós vivos da população. É necessária a utilização da Mutação 3, caso se deseje utilizar ancestrais vivos, para que a variedade de topologias na população seja mantida ao longo das gerações. Esses resultados são importantes pois refletem o comportamento do algoritmo genético e cada uma de suas mutações com a filogenia viva, de modo que os parâmetros possam ser ajustados de maneira mais adequada.

Assim, nota-se a necessidade de utilizar todas as mutações e a recombinação, para variar a resposta final do algoritmo em relação à diferentes topologias e à quantidades de nós vivos. A Mutação 3 mantém a variedade de topologias com nós vivos ao longo das gerações, e sem ela, a população mantém uma quantidade fixa de nós internos vivos em todas as árvores.

A partir de testes realizados com várias combinações de entradas possíveis das mutações, passaram a ser analisados os valores de retorno fixando o uso do parâmetro -f 130, que interrompe o algoritmo apenas quando um escore atinge valor menor que 130. Como o menor escore observado para essa entrada, $M_{75\times759}$, foi de 128, o valor -f 130 representou

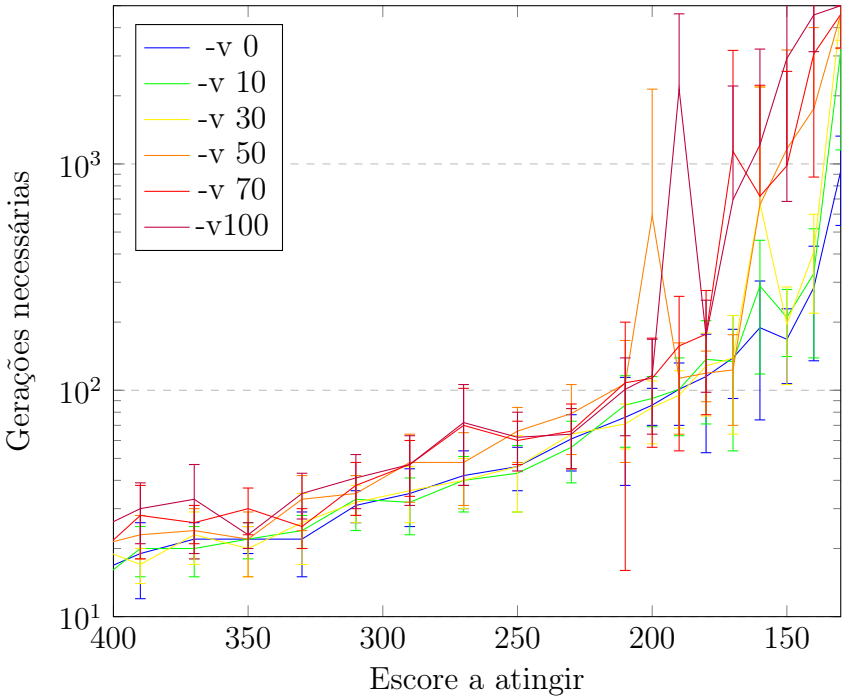


Figura 5.4: Utilização de M1 e diferentes valores de FV

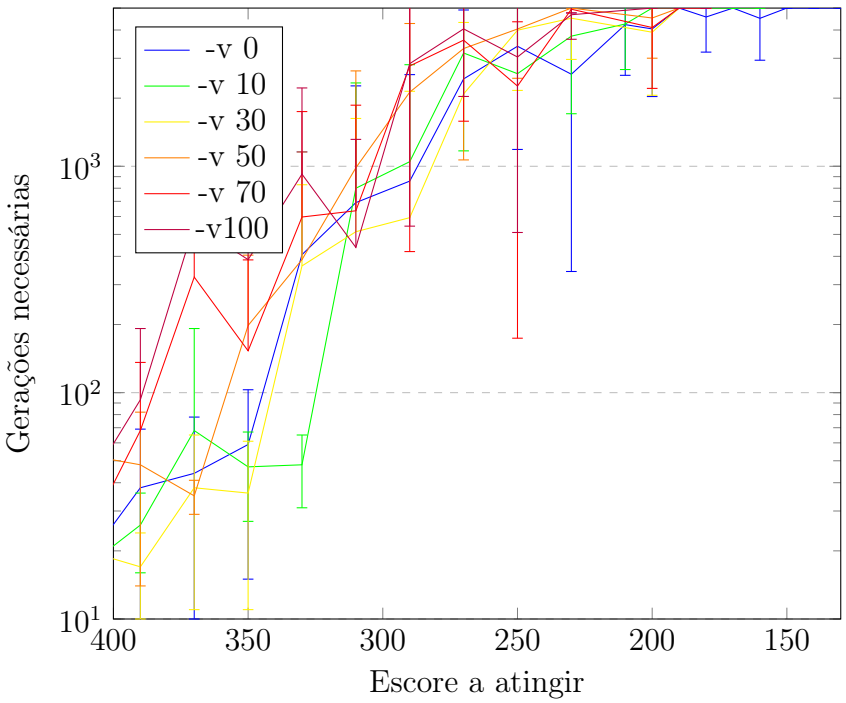


Figura 5.5: Utilização de M2 e diferentes valores de FV

um bom critério de parada.

Para buscar valores mais adequados às três mutações, foram realizadas todas as combinações de 0, 25, 50 e 75 entre cada uma delas, com os outros parâmetros fixados em: -p 200 -g 5000 -e 20 -f 130 -v 50 -r 70. A Tabela 5.1 fornece os melhores resultados dessas combinações. A coluna *Gerações necessárias* mostra a quantidade de gerações executadas para que algum escore menor que 130 fosse alcançado, além do desvio padrão desse resultado, com base em 10 execuções de cada entrada. Os tempos totais de execução das 10 repetições estão na coluna *Tempo das execuções*.

Tabela 5.1: Melhores resultados para diferentes combinações de mutações

M1	M2	M3	Gerações necessárias
75	25	0	1945 ± 1321
70	15	15	2368 ± 1486
75	0	25	2524 ± 1660
50	25	25	2566 ± 1323
50	25	25	2920 ± 1446

Os resultados indicam que a combinação de M1 em um valor mais alto que os demais, e o restante dividido entre M2 e M3, seria a melhor combinação. Para manter sempre a utilização de todas as mutações, garantindo uma variedade maior na população, foram escolhidos os valores de mutação M1=70, M2=15 e M3=15 para os testes seguintes.

A Figura 5.6 representa os valores de gerações necessárias para cada valor da Recombinação 1, de acordo com as mutações fixadas e critério de parada definido por -f 130. Cada linha do gráfico utiliza entradas de FV fixadas em valores distintos.

Ao final de todos os experimentos, representamos na Tabela 5.2 os melhores resultados de FV e da combinação de Mutações para o critério de parada -f 130, considerando -p 200 (população), -e 20 (elite), -g 5000 (gerações máximas). Foram consideradas combinações de entrada que tivessem todas as mutações fixadas em algum valor maior que zero. Podemos notar pelos resultados que a utilização ideal da Recombinação 1 é por volta de 40 ou 50. Para os demais testes, fixaremos R1 como 40. Os piores resultados foram com R1 em valores altos. As duas últimas linhas da tabela representam os piores resultados encontrados, com mais de 4.600 gerações necessárias e R1 no valor de 100.

Por fim, realizamos novos testes para avaliar os parâmetros de elite E e de população P. Fixamos as mutações em M1=70, M2=15, M3=15, R1=40 e a quantidade de nós vivos inicial FV=50.

Para a variação da elite E, observamos uma diminuição na quantidade de gerações necessárias à medida que o valor de E cresce. O resultado é demonstrado na Figura 5.7. Os testes foram realizados para uma população de 100 e para uma população de 200, e os valores de E estão medidos em porcentagem com relação ao total de indivíduos na população.

O parâmetro -a informa uma faixa de nós vivos para a saída final. Por exemplo, a entrada -a 5 8 indica que desejo como saída uma árvore que possua de 5 a 8 nós vivos. O algoritmo considera como árvore de saída a melhor que esteja na faixa de nós vivos informado, ou que tenha a maior quantidade de nós vivos próxima ao valor mínimo. Essa melhor árvore é também considerada como melhor indivíduo para ser usada como

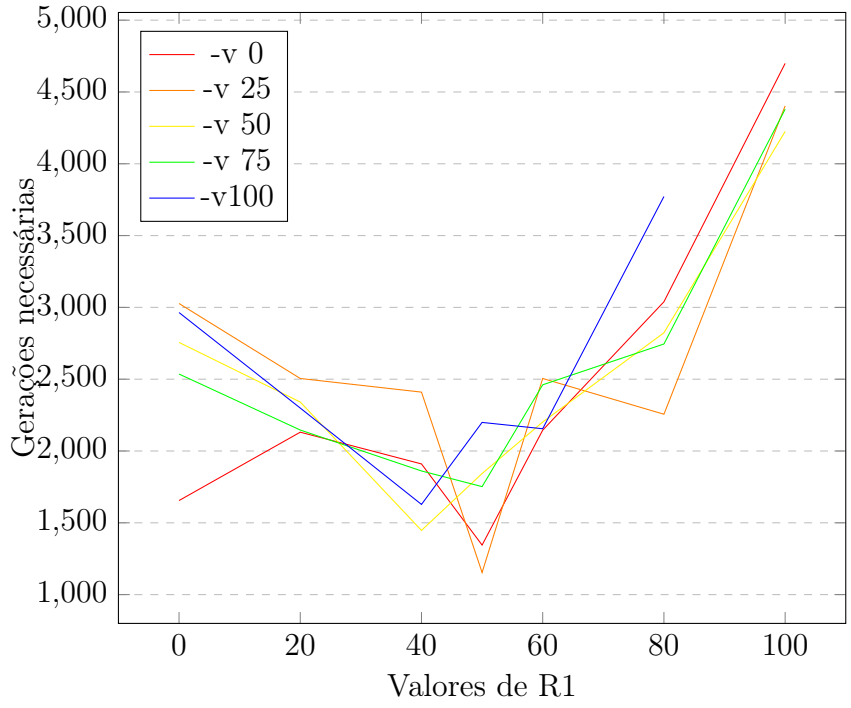


Figura 5.6: Gerações necessárias para cada valor de R1, dados M1=70, M2=15, M3=15 e FV conforme rótulo

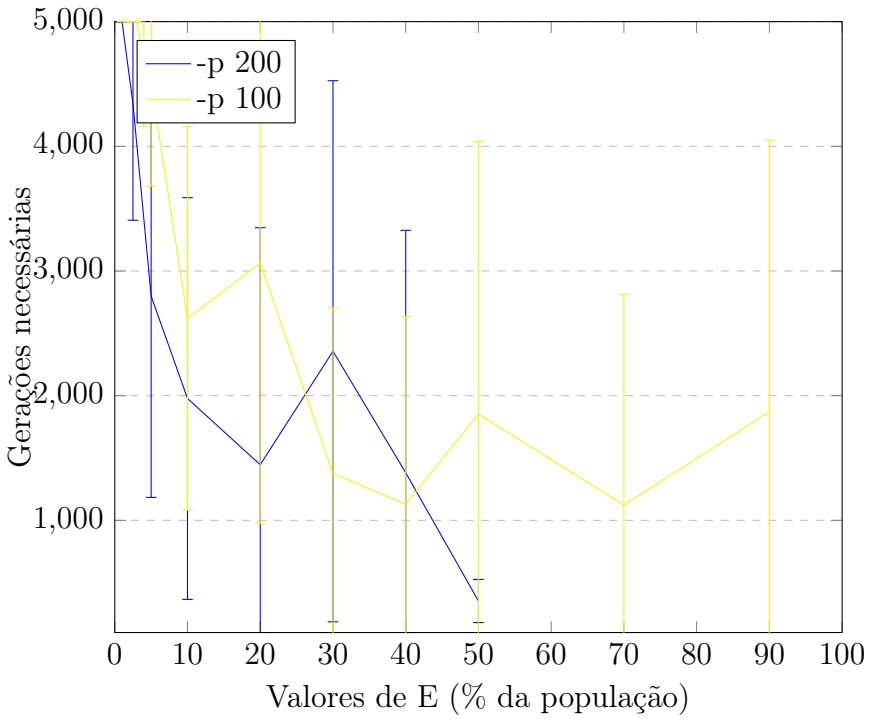


Figura 5.7: Gerações necessárias para cada valor de E, com M1=70, M2=15, M3=15, R1=40, FV=50 e P=200

Tabela 5.2: Melhores resultados para diferentes combinações de mutações e FV, para -p 200 -e 20 -f 130

FV	M1	M2	M3	R1	Gerações necessárias
0	70	15	15	50	1345 ± 851
50	70	15	15	40	1447 ± 625
25	70	15	15	60	1726 ± 799
100	70	15	15	40	1761 ± 1748
25	70	15	15	20	1840 ± 1751
50	70	15	15	50	1843 ± 1378
75	70	15	15	40	1861 ± 1502
0	70	15	15	40	1910 ± 1288
...	...	...	...	...	...
25	70	15	15	100	4635 ± 887
0	70	15	15	100	4699 ± 621

estratégia elitista, o que força o algoritmo a retornar sempre respostas mais próximas da desejada.

O comportamento do algoritmo foi projetado para que alcance essa quantidade de nós vivos, alterando dinamicamente o valor de R1, que tem um forte papel em diminuir a quantidade de nós vivos, e do parâmetro interno da mutação M3, que indica se nós se movem em direção à raíz ou às folhas. Foram realizados testes variando a faixa de nós vivos desejada. A Figura 5.8 mostra o valor final do escore alcançado, para uma média de 10 execuções do algoritmo, de acordo com cada faixa de nós vivos desejado de saída. No eixo x, o valor 2 representa a faixa de 1 a 3 nós vivos, 5 a faixa de 4 a 6 nós vivos, seguindo até o valor final de 16 a 18 nós vivos, o número máximo possível para a entrada $M_{34\times306}$. Foi utilizado o critério de parada fornecido com o parâmetro -s 500, ou seja, caso o algoritmo não melhore seu escore em 500 gerações, a execução é interrompida. Na média total, cada execução durou 1664 gerações.

Para a entrada $M_{75\times759}$, foi realizado o mesmo teste, com uma população de 100 indivíduos e parâmetro -s de 1000 gerações. Os resultados representam a média de 3 execuções e estão representados pela Figura 5.9. O número máximo de nós vivos para essa entrada é de 36 nós internos. Na média, foram executadas 4062 gerações.

Com a implementação do módulo do algoritmo de Sankoff, mais alguns testes foram realizados para verificar seu funcionamento. Foi utilizada a entrada $M_{52\times765}$ com os parâmetros fixos em: M1=60, M2=20, M3=20, R1=40, P=100, E=8, G=5000, S=500, V=50, e AMIN e AMAX variando. Foram realizadas 10 execuções para cada entrada de AMIN e AMAX, que variaram de 0-4 até 20-24, representando a quantidade de nós internos que se deseja na saída. O parâmetro S=500 corresponde ao critério de parada, que irá interromper a execução caso um escore menor não seja alcançado em 500 gerações, e G=5000 fixa o número máximo de gerações. O gráfico da Figura 5.10 representa a média do escore final, de acordo com o número de nós vivos que se deseja na entrada, representados no eixo x.

Conforme a quantidade de nós vivos desejada aumenta, o escore mínimo atingido passa a ser maior, o que mostra que, para essa entrada, filogenias com nós vivos possuem uma parcimônia pior. Para essas execuções, o algoritmo de Sankoff foi utilizado e foi arbritado

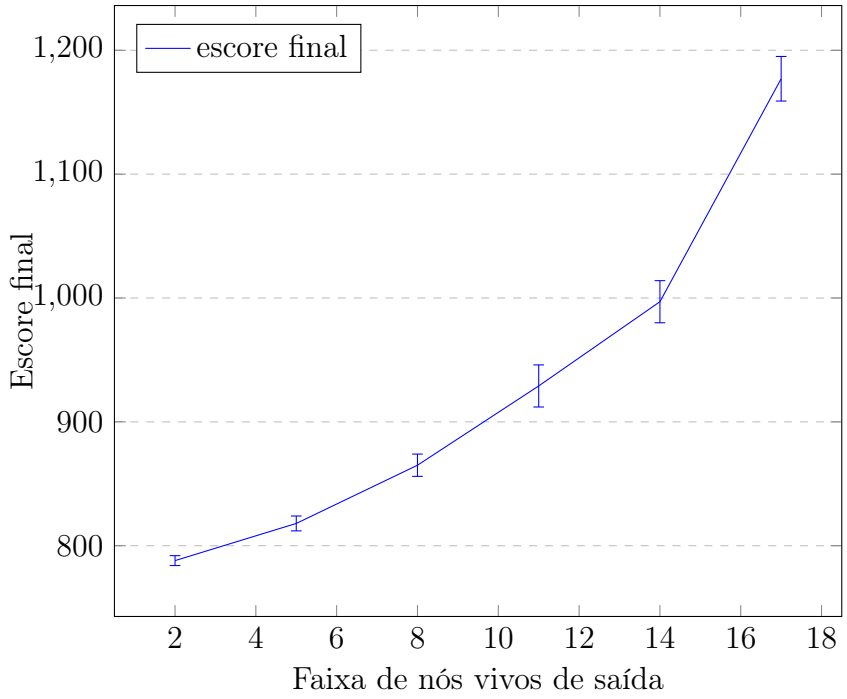


Figura 5.8: Escore final por faixa de nós vivos de saída, para entrada $M_{34\times306}$ com -m 70 15 15 -r 40 -p 200 -e 10 -s 500

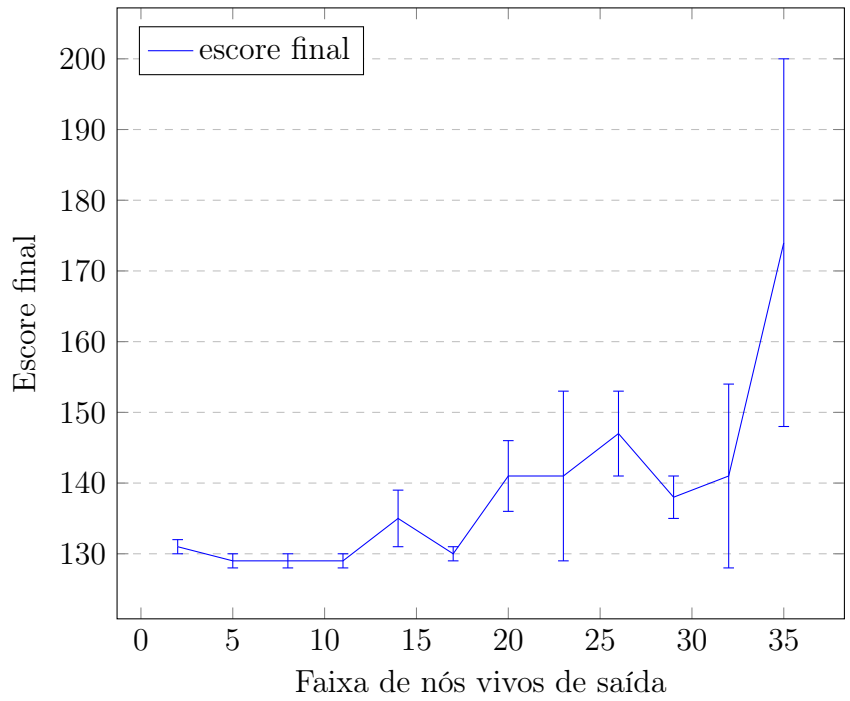


Figura 5.9: Escore final por faixa de nós vivos de saída, para entrada $M_{75\times759}$ com -m 70 15 15 -r 40 -p 100 -e 5 -s 1000

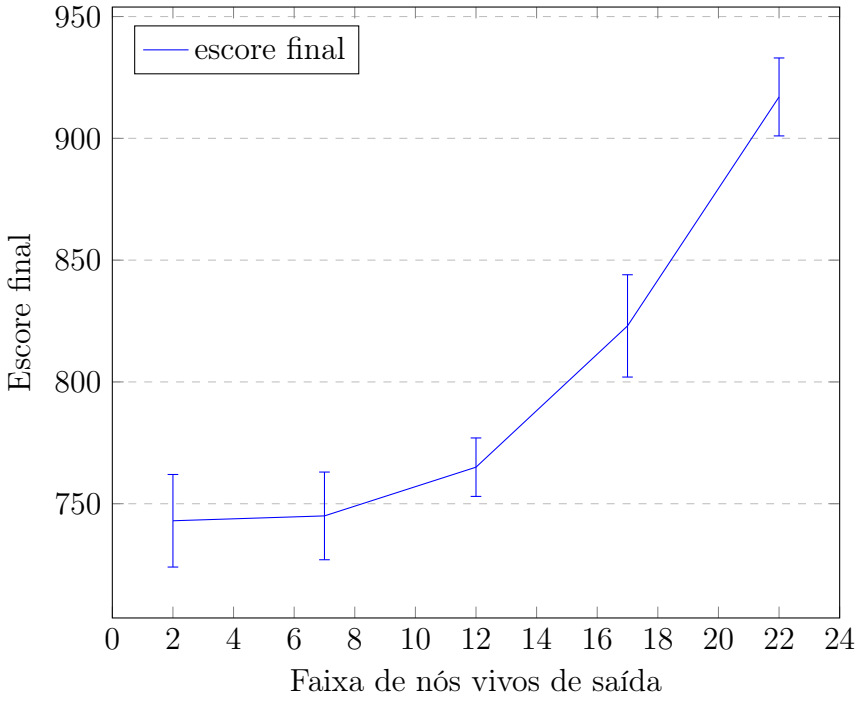


Figura 5.10: Escore final por faixa de nós vivos de saída, para entrada $M_{52\times765}$ com -m 60 20 20 -r 40 -p 100 -e 8 -s 500

uma matriz de custos C_v conforme mostrado na Tabela 5.3. Foi atribuído um custo menor para as transições (A↔C ou T↔G).

Tabela 5.3: Matriz C_v de custos de transição de estados de DNA variáveis, usado pelo algoritmo de Sankoff implementado neste trabalho.

	A	T	G	C
A	0	3	2	3
T	3	0	3	2
G	2	3	0	3
C	3	2	3	0

Também foi utilizada, para comparação dos resultados, uma matriz de custos em que qualquer transição de estado tem custo 1, exceto as transições da diagonal, que possuem custo 0. Chamaremos essa matriz de custos de C_1, e está mostrada na Tabela 5.4.

Como o algoritmo de Sankoff não possui tratamento de *gaps*, pois necessita de uma matriz de características como entrada, decidimos excluir as colunas que possuiam pelo menos uma característica no estado "-", que representa *gaps* nos alinhamentos. Similar ao parâmetro *–exclude-equal*, o parâmetro *–remove-gap* retira as colunas da matriz de entrada que apresentam pelo menos 1 *gap*, diminuindo o tempo de execução. Os tempos estão mostrados na Tabela 5.5. Essa retirada de colunas da matriz de características de entrada é realizada para calcular o escore ao longo das gerações do algoritmo genético. A matriz completa é utilizada para operação de rotulação dos nós internos.

Para efeito de comparação dos resultados e do tempo total de 10 execuções do algoritmo, a Tabela 5.5 nos fornece o escore de saída, com seu desvio padrão, a geração final,

Tabela 5.4: Matriz C_1 de custos de transição de estados de DNA, usado pelo algoritmo de Sankoff neste trabalho

	A	T	G	C
A	0	1	1	1
T	1	0	1	1
G	1	1	0	1
C	1	1	1	0

com seu desvio padrão, para os mesmos parâmetros das execuções anteriores, fixando AMIN em 0 e AMAX em 4, e variando alguns parâmetros opcionais:

1. Utilização do algoritmo de Sankoff como função de *fitness*, com matriz de custos C_v, e com parâmetros –exclude-equal e –remove-gap;
2. Utilização do algoritmo de Sankoff como função de *fitness*, com matriz de custos C_1, e com parâmetros –exclude-equal e –remove-gap;
3. Utilização do algoritmo de Fitch como função de *fitness*, com parâmetros –exclude-equal e –remove-gap;
4. Utilização do algoritmo de Sankoff como função de *fitness*, com matriz de custos C_1, e sem os parâmetros.

Tabela 5.5: Escores médios de saída e tempo gasto nas 10 execuções do algoritmo para cada tipo de algoritmo e matriz de custos usadas, descritas anteriormente

Descrição	Escore	Geração final	Tempo (10 ex.)
1. Sankoff C_v	743 ± 19	2358 ± 435	00:42:50
2. Sankoff C_1	331 ± 4	2319 ± 636	01:17:10
3. Fitch	332 ± 8	2305 ± 629	00:48:03
4. Sankoff C_1	569 ± 5	2294 ± 570	05:32:49

O escore final médio com o algoritmo de Sankoff e com a matriz de custos C_1 e com o algoritmo de Fitch foram muito próximos, o que é esperado, visto que o algoritmo de Fitch se comporta como o algoritmo de Sankoff caso os custos de transição de estados sejam todos fixados em 1 para características diferentes e em 0 para as mesmas características. Para a execução sem os parâmetros –exclude-equal e –remove-gap, como a matriz de entrada $M_{52\times765}$ possui 765 características, o cálculo do escore necessita mais operações para ser concluído. O uso de –exclude-equal e –remove-gap, para essa entrada, fez com que a quantidade de características passasse de 765 para 170, reduzindo drasticamente a quantidade de operações necessárias e, consequentemente, o tempo de execução do algoritmo.

5.2 Estudos de caso

O primeiro estudo de caso realizado foi com a entrada $M_{75\times759}$, também utilizada na realização dos testes de calibragem de parâmetros. O segundo estudo de caso foi realizado

com a entrada $M_{52\times765}$ criada a partir de um alinhamento múltiplo de diversas leituras da região *env* do vírus HIV, todos de um mesmo paciente, lidos ao longo de 12 anos.

5.2.1 Estudo de caso 1

Considerando a entrada $M_{75\times759}$, que contém AMS de vírus H1N1 e H3N2 de diferentes países, selecionamos duas saídas de escores mínimos (128) para serem analisadas. Nosso objetivo é avaliar como se agrupam os vírus em relação a sua região geográfica, tanto em filogenias tradicionais como em filogenias vivas. Foi utilizado o algoritmo de Fitch para os testes.

A primeira saída é representada pela árvore em formato newick:

(((19,1),26),((((23,(((((44,(43,(((60,37),((((28,11),46),(42,36)),((74,(65,10)),66))),(59,55)))),(58,68)),64),41),75)),40),(((57,(54,56)),72),
(8,31))),(((15,((((30,73),((47,((((7,6),((52,((3,(((69,61),((62,53),(67,70))),9)),((2,4),(12,63)))),22)),13),32)),(((35,27),16),(34,24)))),5),33)),
(20,71)),((50,(48,((38,18),14))),(49,((17,((25,39),29)),((45,21),51)))))))

As Figuras 5.11, 5.12 e 5.13 mostram partes dessa árvore final. O número inicial representado na figura corresponde ao número na árvore em formato newick, e é seguido do país, cidade (ou região) e tipo de vírus, H1N1 (HuNu) ou H3N2 (HtNd). A filogenia completa pode ser visualizada na Figura B.1 do Apêndice B.

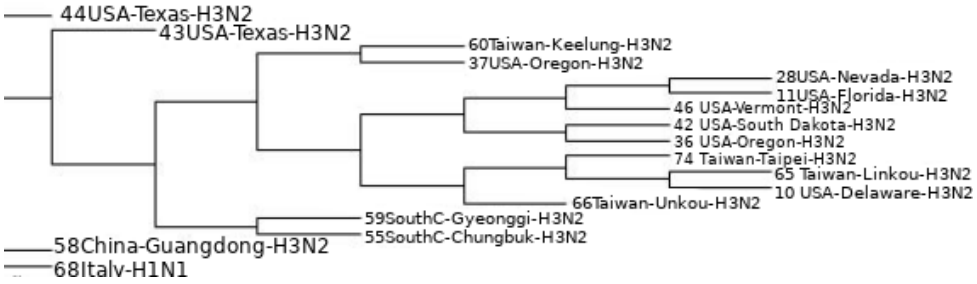


Figura 5.11: Trecho da árvore final mostrando vírus do tipo H3N2 agrupados na mesma região

Figura 5.12: Trecho da árvore final mostrando vírus da região da Rússia agrupados

A segunda saída foi selecionada de forma a ter o maior número de nós internos vivos dentre as várias opções de árvores. Pode ser representada pela árvore em formato newick:

(((36,(42,37(28(((55,59),10(74,(66,65))),11),(46,60)))),(((((((54,56),57),(31,8)),(21(2,5(((3(52,45),19((14,((20,((73,50),71)),(47,(48((24,13),

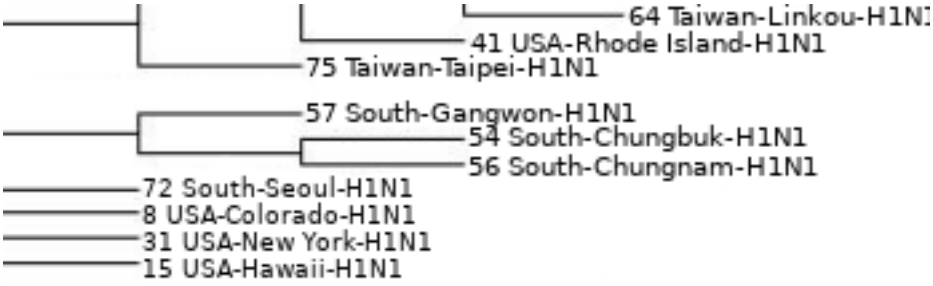


Figura 5.13: Trecho da árvore final mostrando vírus da região asiática agrupados

((63,7),((32((16,6),(39((12,30),((29,15),38(25,9))),(35,27))),(49,33)),17))),26(34,(1,18))))))),(51,22))),4),(69(62,53),((67,70),61)))),72)),
(23,40)),((64,41),75)),(68,58))),(43,44))

As Figuras 5.14, 5.15 e 5.16 mostram partes dessa árvore final, em que vários nós internos vivos podem ser vistos na solução. A filogenia completa pode ser visualizada na Figura B.2 do Apêndice B.

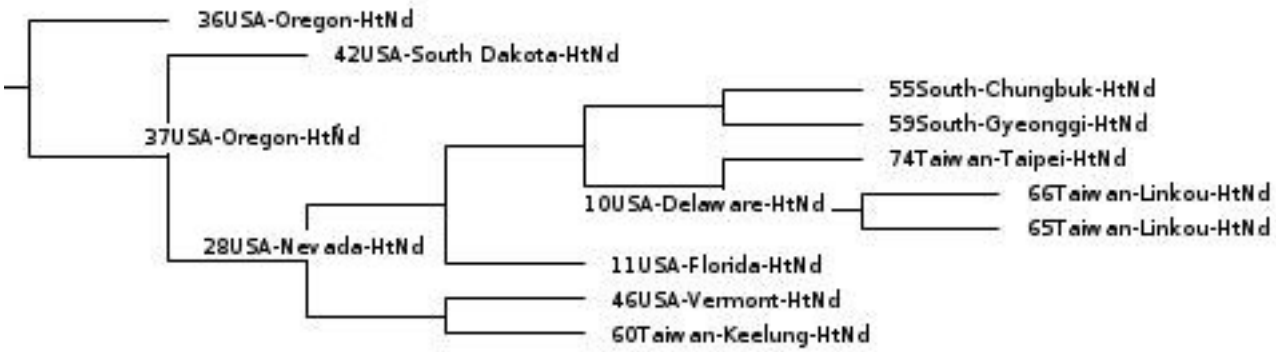


Figura 5.14: Trecho da árvore final mostrando vírus do tipo H3N2 agrupados na mesma região com nós vivos

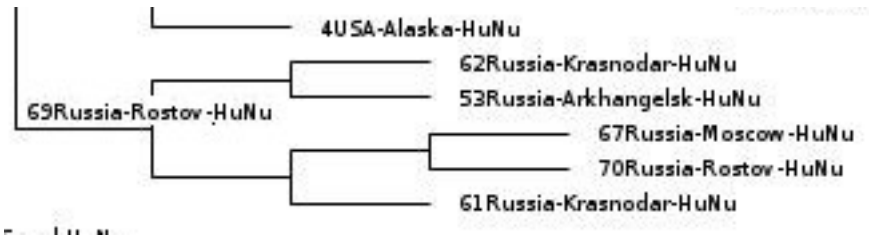


Figura 5.15: Trecho da árvore final mostrando vírus da região da Rússia agrupados, incluindo um nó vivo em um ponto mais interno

Utilizando a mesma matriz de características como entrada, utilizamos o software Phylip [5] para gerar uma filogenia. Utilizamos o módulo Proml (*Protein Maximum Likelihood program*). A Figura 5.17 mostra a filogenia, agrupando vírus da mesma região. A filogenia de saída completa também pode ser vista no Apêndice B.

Notamos que o Phylip gerou um agrupamento com os organismos coletados na Rússia, o que também pode ser visto nas filogenias de saída geradas pelo GA-LP. Alguns outros

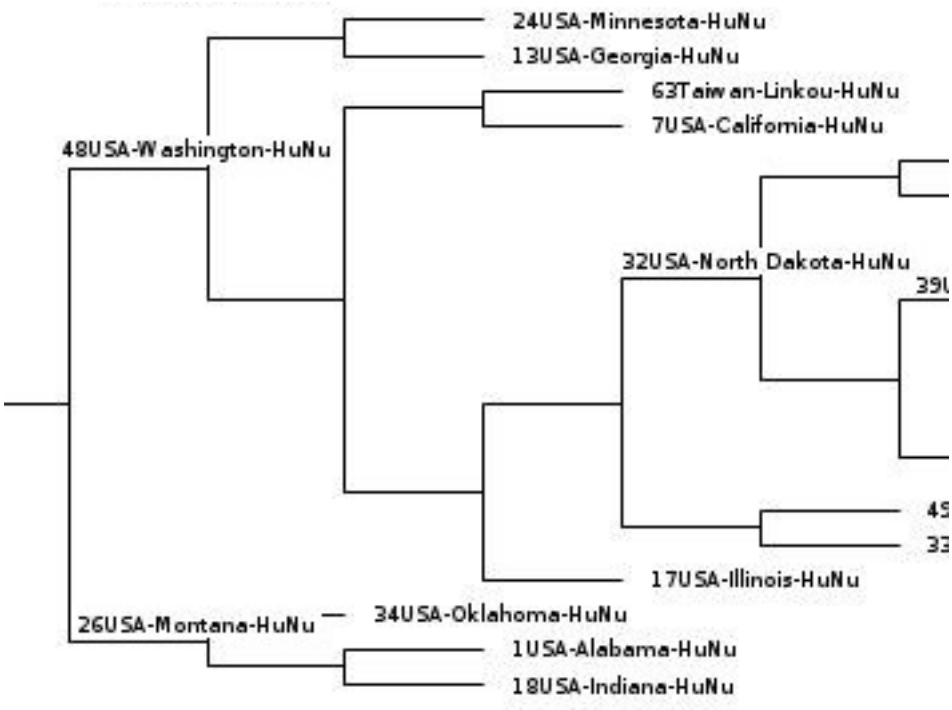


Figura 5.16: Vários indivíduos dos EUA agrupados em uma mesma região e com diversos nós internos vivos

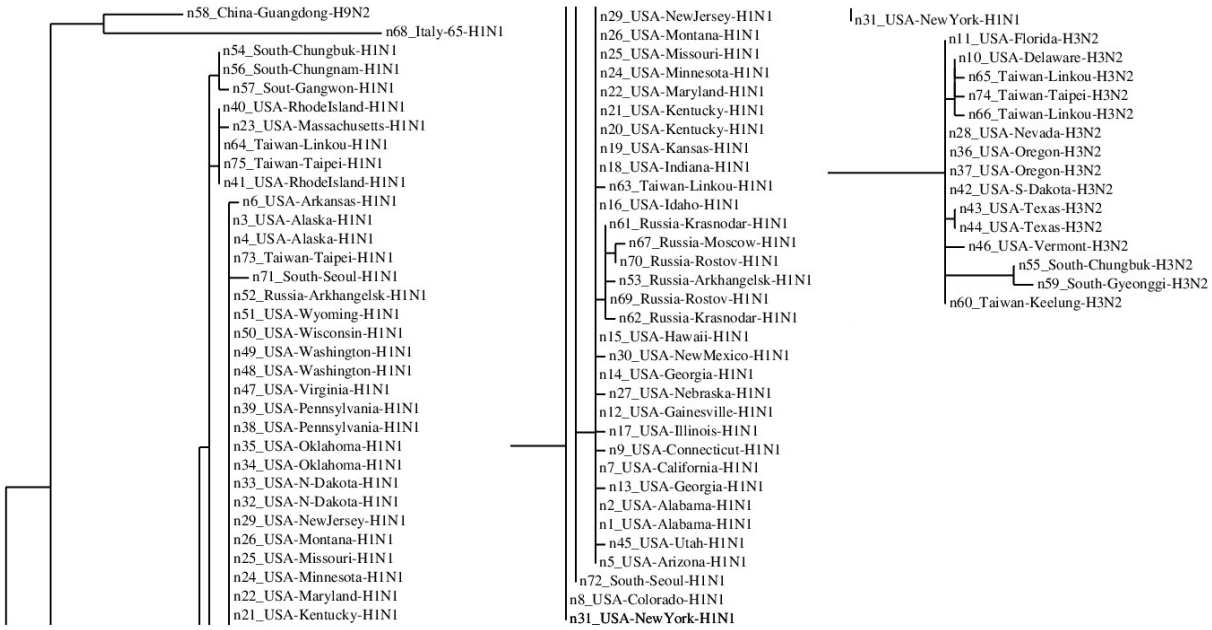


Figura 5.17: Filogenia de saída do software Phylip, agrupando vírus da região da Rússia.

agrupamentos são semelhantes, como os de South-Chungbuk e South-Chungnam, Taiwan, Alaska. A filogenia com nós vivos ainda contribui para indicar candidatos a ancestrais vivos, que possam ter dado origem aos demais, e para agrupar os indivíduos em um agrupamento com nós internos, fornecendo uma visualização das relações que ainda não

é gerada pelos softwares de construção de filogenias atuais.

5.2.2 Estudo de caso 2

O estudo de caso 2 foi realizado com os mesmos dados usados para gerar a matriz $M_{52\times765}$. O estudo analisava amostras de vírus HIV-1 de 11 pacientes diferentes, visitados anualmente, por um período de 6 a 12 anos de infecção. A região sequenciada foi a região C2-V5 do gene *env*, bastante estudada, por codificar um alvo importante de imuno repressores e possuir uma variação genética alta [20].

A entrada montada analisa amostras de um mesmo paciente, chamado de paciente 9. Os rótulos são representados por "p9cXXXXX", onde "XXXXX" é uma sequência de dígitos, dos quais os dois primeiros denotam o número da visita, sequenciada temporalmente, e os demais representam um determinado clone do total das amostras da visita. A entrada formou a matriz $M_{136\times680}$ com um total de 136 linhas (indivíduos de entrada) e 680 características, representando um alinhamento múltiplo de sequência. Foi utilizado o algoritmo de Sankoff, com a remoção de colunas iguais e de colunas que contenham gaps, e a matriz de custo de transição de estados C_v definida pela Tabela 5.3.

Diversas filogenias de saída foram obtidas, de acordo com a quantidade de nós vivos desejados. Inicialmente, executamos o algoritmo variando os nós vivos da saída de 6 em 6, e o escore final de cada execução, junto ao tempo de duração, pode ser visto na Tabela 5.6. Foram utilizados os parâmetros -p 100 -g 5000 -e 10 -m 60 20 20 -r 40 -v 50 -s 500 –exclude-equal –remove-gap e -a variável.

Tabela 5.6: Filogenias de saída obtidas com a entrada $M_{136\times680}$, junto ao escore de saída, nós vivos e tempo da execução

No.	Escore	Nós vivos	Tempo
1	1694	0	00:32:27
2	1702	6	00:29:50
3	1731	12	00:28:21
4	1771	18	00:26:24
5	1739	24	00:24:48
6	1817	30	00:23:03
7	1947	36	00:21:19
8	2034	42	00:19:59
9	2094	48	00:20:21
10	2112	54	00:17:37
11	2221	60	00:14:26
12	2561	66	00:10:15
13	2537	67	00:10:36

Selecionamos 3 árvores para desenhar e analisar, e executamos o algoritmo novamente com -g 20000 e -s 500, ou seja, com um critério de parada de 500 gerações sem alterar o escore, respeitando o máximo de 20000. Executamos o algoritmo para 0 nós vivos de saída, para 18 a 30, e para 54 a 60. Assim, temos a representação de uma saída sem nós vivos, uma com alguns (18) e uma com uma grande quantidade (54). O resultado das três execuções do algoritmo pode ser vista na Tabela 5.7.

Tabela 5.7: Filogenias de saída obtidas com a entrada $M_{136\times680}$, junto ao escore de saída, nós vivos e gerações necessárias para atingir o escore.

No.	Escore	Nós vivos	Gerações
1	1614	0	7821
2	1667	18	6660
3	2069	54	9226

As três filogenias completas são mostradas no Apêndice C. O escore obtido para 18 nós vivos não aumentou muito em relação a melhor saída obtida, com 0 nós vivos. Como o aumento de escore se estabiliza ou aumenta conforme aumentamos os nós vivos, é necessário forçar que o algoritmo nos retorne uma filogenia com muitos nós vivos, se desejarmos.

Foi utilizado o software PAUP [42] para gerar uma filogenia com a mesma entrada, $M_{136\times680}$, através do método *Neighbor Joining* [33]. Para selecionar o modelo da matriz de substituição do alinhamento, foi utilizado o *jModeltTest* [28], que indicou o modelo TIM+I+G (*transitional model plus proportion of invariate sites and gamma distribution*). A filogenia completa obtida é mostrada na Figura C.4 do Apêndice C.

Em geral, a filogenia de 18 nós vivos formou agrupamentos que indicam indivíduos lidos em uma mesma visita, ou de visitas próximas. Nem todos os indivíduos são resolvidos corretamente, o que também ocorre na filogenia gerada pelo PAUP. A Figura 5.18 mostra dois trechos da árvore de saída, um com todos os nós do agrupamento representando a visita 15, e outro com nós das visitas 11, 12, 13 e 15 em um mesmo agrupamento.

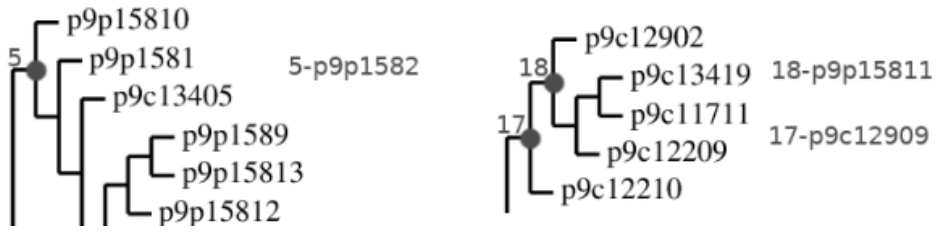


Figura 5.18: Trechos da filogenia de saída com 18 nós vivos, um agrupando indivíduos representando a visita 15, e outro com variadas visitas, de 11 a 15, em um mesmo trecho.

A Figura 5.19 mostra um trecho da árvore de saída com 54 nós internos. A quantidade de ancestrais vivos nessa árvore (54) é maior que a de ancestrais hipotéticos (13). Nesse trecho, é possível ver um agrupamento formado com indivíduos das visitas 00 e 01, em sua maioria.

Quando a filogenia possui muitos indivíduos para serem analisados, a visualização pode se tornar ruim devido a uma árvore muito extensa. Os nós vivos indicam ancestrais vivos, ou seja, candidatos a terem originado os demais indivíduos do agrupamento que se encontram nas folhas.

A Figura C.3 possui alguns agrupamentos que nos sugerem uma relação entre nós vivos e folhas que estaria dentro de uma possibilidade real de nós vivos gerando descendentes, e estando dentre os organismos estudados. Os organismos 16-p9c08605 e 54-p9c08619, referentes a amostras da visita 08, são mostrados como ancestrais vivos de organismos das visitas 08, 10 e 11. O organismo 7-p9c11707 gera organismos das visitas 11 e 12,

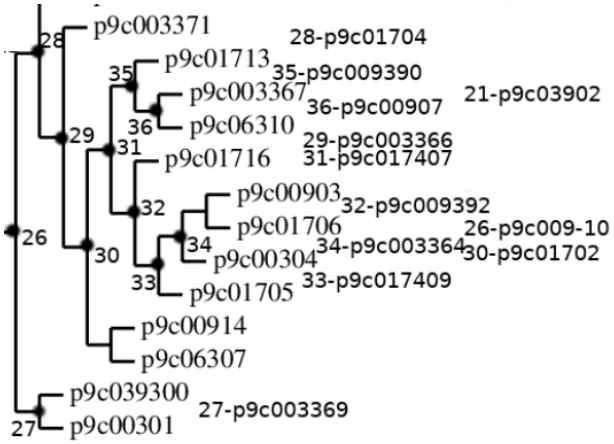


Figura 5.19: Trecho da filogenia de saída com 54 nós vivos, com um agrupamento de indivíduos, em sua maioria, das visitas 00, 01 e 03.

por exemplo. Outros exemplos podem ser notados na Figura C.3. Alguns exemplos não seguem estritamente essa linha temporal de amostra, como o indivíduo p9c08602 que é gerado pelo 42-p9c11716, mas estão em um agrupamento com visitas semelhantes. Esse indivíduo p9c08602 também não é resolvido junto a visitas semelhantes na árvore gerada pelo PAUP, que pode ser vista na Figura 5.20, e se encontra próximo a indivíduos das visitas 10 e 11, de forma semelhante à filogenia de sáida com 54 nós vivos do GA-LP. Em geral, essas observações mostram que nossas filogenias de saída estão coerentes com a gerada pelo software PAUP, agrupando indivíduos de maneira semelhante. A filogenia viva nos dá uma possibilidade de analisar candidatos a serem ancestrais vivos dentro dos dados de entrada, e nos fornece uma visualização diferente dos demais softwares de construção de filogenia.

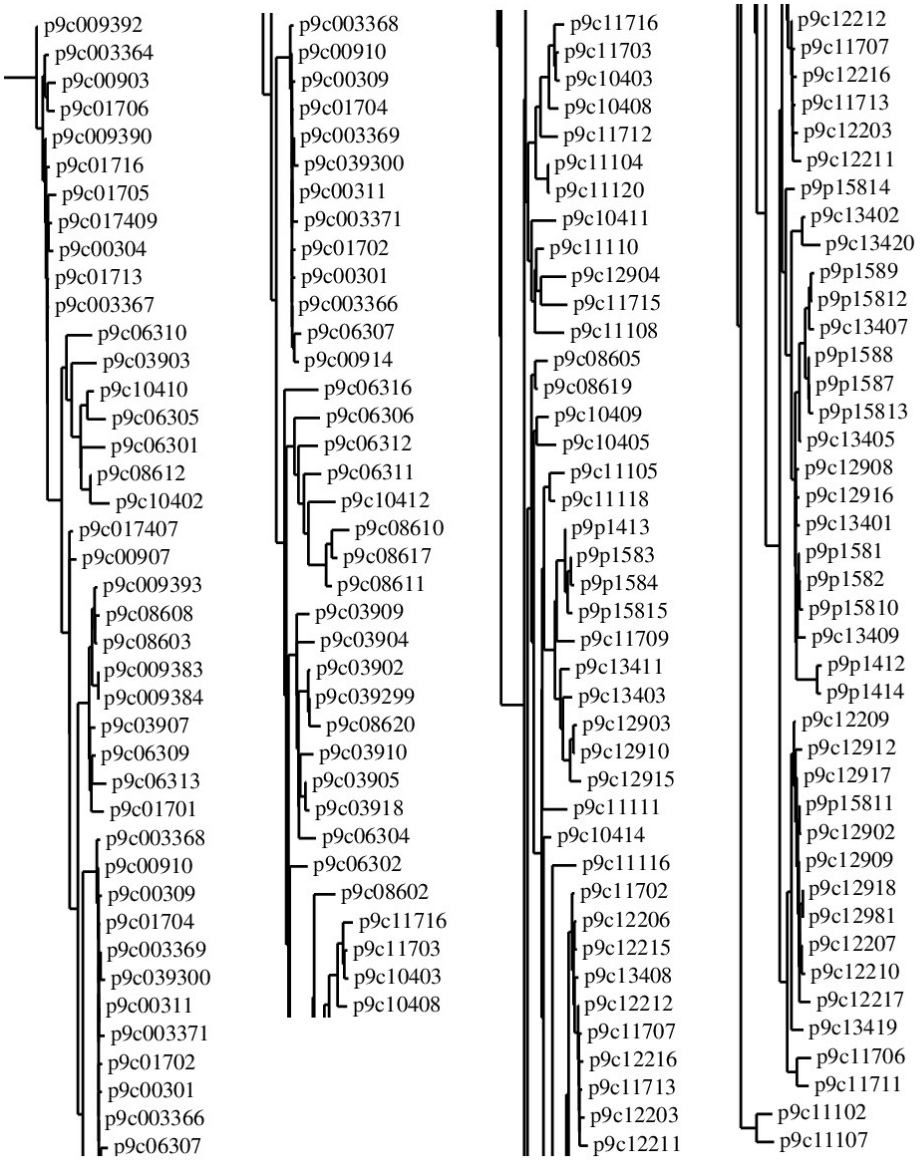


Figura 5.20: Trecho da filogenia de saída com 54 nós vivos, com um agrupamento de indivíduos, em sua maioria, das visitas 00, 01 e 03.

Capítulo 6

Conclusão

Nesta dissertação, apresentamos um método baseado em algoritmos genéticos para construir uma filogenia viva, tendo como entrada uma matriz de características. Propusemos e implementamos uma variação dos algoritmos de Fitch e Sankoff, de modo a resolver o problema da filogenia viva pequena. Propusemos uma nova operação, chamada de mutação 3, especificamente para tratar a geração de folhas ou nós internos vivos. As operações chamadas de mutação 1, mutação 2 e recombinação 1 foram adaptadas para a filogenia viva. Implementamos um algoritmo genético para resolver o problema da filogenia viva grande, usando os algoritmos adaptados (Fitch e Sankoff) para o problema da filogenia viva pequena. Analisamos o algoritmo para avaliar cada parâmetro e fixar seus valores de modo que contribuíssem para encontrar o menor valor de escore para uma determinada entrada, com o menor número de gerações possíveis.

Foram realizados dois estudos de caso. O primeiro tinha como entrada uma matriz de características composta por um alinhamento múltiplo de sequenciamento de vírus H1N1 e H3N2 de diferentes países (Estados Unidos, Rússia, Coréia do Sul, Taiwan, Itália e China). Executamos o algoritmo GA-LP com nós vivos, e as filogenias geradas formaram agrupamentos com vírus da mesma região. Analisamos a filogenia de saída, com e sem nós vivos, e geramos uma saída no software Phylip. As saídas com ancestrais vivos formaram agrupamentos com vírus de regiões próximas. A saída do software Phylip possui alguns agrupamentos semelhantes, com vírus da mesma região. O segundo estudo de caso foi realizado com um alinhamento múltiplo do sequenciamento da região *env* do vírus H1V. Foram coletadas amostras de vírus em um mesmo paciente, ao longo de várias visitas distribuídas por um período de 12 anos. Obtivemos como saída filogenias com diferentes números de nós vivos, e analisamos 3 delas, desenhando as árvores para observar os agrupamentos gerados. Neles, tanto os nós folhas como os nós vivos representavam indivíduos de um mesmo período, ou de um período próximo. Além disso, usamos o PAUP para gerar uma filogenia. A filogenia gerada pelo software PAUP mostrou agrupamentos coerentes com os períodos de coleta, assim como as filogenias geradas. Porém, as filogenias vivas nos fornecem uma possibilidade da existência de ancestrais vivos entre as amostras estudadas, além de agrupar os indivíduos de maneira diferente da convencional, o que pode indicar novas características de relações evolutivas entre os organismos estudados.

Como trabalhos futuros, sugerimos a realização de mais estudos de casos para validação do método proposto. Por exemplo, seria interessante construir mais árvores envolvendo indivíduos que estejam em uma linha temporal bem definida, como a realizada pelo estudo

de caso 2. Isso é interessante pois espera-se que, em uma filogenia de saída, ancestrais vivos mais antigos estejam representados como nós vivos, gerando descendentes mais recentes. Realizar experimentos com árvores genéricas permitiria investigar a capacidade de generalização do método GA-LP. Melhorar a formalização da representação de uma filogenia viva, como feito com o formato newick adaptado utilizado por esse trabalho, se faz necessária. Também é útil uma implementação que apresente graficamente a filogenia viva construída pelo método. Esse processo foi realizado manualmente ao longo desse trabalho, tendo em vista que não existem softwares para construção de filogenias vivas a partir de uma string newick ou de um arquivo .fasta. Outro projeto interessante seria submeter ao GA-LP entradas com dados de outras aplicações, como textos e documentos.

Apêndice A

Matriz de entrada $M_{34\times306}$

A matriz possui 34 linhas com 306 características em cada. Os gaps foram substituídos por "g".

```
01 MVTEFIFLGLSDSQELQTFLFMLFFVFYGGIVFGNLLIVITVVSDSHLHSPMYFLLANLSLIDLSLSSVTAPKMITDFFSQRKVISFKGCLVQIFLLHFFGGSEMVILIAMGFDRYIAICKPLHYTTIMCGNACVGIMAVTWGIGFLHSV
SQLAFAVHLLFCGPNEVDSFYCDLPRVIKLACTDTYRLDIMVIANSGVLTVCSFVLLIISYTIILMTIQHRPLDKSSKALSTLTAHITVVLLFFGPCVFIYAWPFPIKSLDKFLAVFYSVITPLLNPIIYTLRNKDMKTAIRQLRKWDAHSSVKFZ
02 MVTEFIFLGLSDSQELQTFLgMLFFVFYGGIVFGNLLIVRIVVSDSHLHSPMYFLLANLSLIDLSLCSVTAPKMITDFFSQRKVISFKGCLVQIFLLHFFGGSEMVILIAMGFDRYIAICKPLHYTTIMCGNACVGIMAVAWGIGFLHSV
SQLAFAVHLPFCGPNEVDSFYCDLPRVIKLACTDTYRLDIMVIANSGVLTVCSFVLLIISYTIILMTIQHRPLDKSSKALSTLTAHITVVLLFFGPCVFIYAWPFPIKSLDKFLAVFYSVITPLLNPIIYTLRNKDMKTAIRRLRKWDAHSSVKFZ
03 VVTEFIFLGLSDSQELQTFLFMLFFVFYGGIVFGNLLIVIIVVSDSHLHSPMYFLLANLSLIDLSLSSVTAPKMITDFFSQRKVISFKGCLVQIFLLHFFGGSELVILLAMGFDRYIAICKPLHYTTIMCGNACAGIMAVAWGIGFLHSV
SQLAFAVHLPFCGPNEVDSFYCDLPRVIKLACTDTYRLDIMVIANSGVLTVCSFVLLIISYTIILMTIRRCPSDKSSKALSTLTAHITVVLLFFGPCVFIYAWPFLIKSLDKFLAVFYSVITPLLNPIIYTLRNKDMKSAIRRLRKWDVHSSVRFZ
04 IVTEFIFLGLSDSQELQIFLFVLFLVFYGGIVFGNLLIVITVVSDSHLHSPMYLLLANLSVVDLSLSSVTAPKMITDFFSQRKAISFKGCLVQIFLLHFFGGSEMVILIAMGFDRYVAICRPLHYTTIMCGSACVGIMAVAWGIGFLHSV
SQLAFAVHLPFCGPNEVDSFYCDLPRVIKLACTDTYKLDIMVIVNSGVLTVCSFVLLIISYTIILMTIQHRPSDRSSKALSTLTAHITVVLLFFGPCVFISAWPFPIKSLDKFLAVFYSVITPLLNPIIYTLRNKDMKTAIRRLRKWDVHSggKSZ
05 IVTEFIFLGLSDSQELQIFLFVLFLVFYGGIVFGNLLIVITVVSDSHLHSPMYLLLANLSVIDLSLSSVTAPKMITDFFSQRKAISFKGCLVQIFLLHFFGGSEMVILIAMGFDRYVAICRPLHYTTIMCGSACVGIMAVAWGIGFLHSV
SQLAFAVHLPFCGPNEVDSFYCDLPRVIKLACTDTYKLDIMVIVNSGVLTVCSFVLLIISYTIILMTIQHRPSDRSSKALSTLTAHITVVLLFFGPCVFISAWPFPIKSLDKFLAVFYSVITPLLNPIIYTLRNKDMKTAIRRLRKWDVHSggKSZ
06 IVTEFIFLGLSDSQELQIFLFVLFFVFYGGIVFGNLLIVITVVSDSHLHSPMYFLLANLSVIDLSLSSVTAPKMITDFFSQRKAISFKGCLVQIFLLHFFGGSEMVILIAMGFDRYVAICRPLHYTTIMCGNACVGIMAVAWGIGFLHSV
SQLAFAIHLPFCGPNEVDSFHCDLPRVIKLACTDTYKLDIMVIANSGVFTVCSFVLLIISYTIILMTIQRRPSDRSSKALSTLTAHITVVLLFFGPCVFIYAWPFPIKSLDKFLAVFYSVITPLLNPIIYTLRNKDMKTAIRRLRKWDVHSSVKSZ
07 MVTEFIFLGLSDSRELQIFLFVLFFVFYGGIVFGNLLIVITVVSDSHLHSPMYFLLANLSLIDLSLSSVTAPRMITDFFSECKVISFKGCLVQIFLVHFFGGSELVILIAMAFDRYVAICKPLYYTTIMSGNTCVGIVAVAWGIGFLHSV
SQLAFAVHLPFCGPNEVDSFYCDLPRIIKLACTDTYRLDIMVIANSGVLTVCSFVLLIISYTIILMTIQRRPSDMSSKALSTLTAHITVVLLFFGPCIFIYAWPLPNKSLDKFLAVFCSVITPLLNPIIYTLRNKDMKTAIRRLRKWDVHSSIKSZ
08 MVTEFIFLGLSDSQELQIFLFVLFFVFYGGIVFGNLLIVITVVSDFHLHSPMYFLLANLSFIDLSLSSVTAPKMITDFFSECKVISFKGCLVQIFFLHFFGGSELVILIAMAFDRYVAICKPLHYTTIMCGNTCVGIVAVAWGIGFLHSV
SQLAFAVHLPFCGPNEVDSFYCDLPRVIELACTDTYRLDIMLIANSGVLTICSFVLLIISYTIILMTIQRCPSDSSSKALSTLTAHITVVLLFFGPCILIYAWPFPIKSLDKFLAVFSYVITPLLNPIIYTLRNKDMKTAIRRLRKWDVHPSVKSZ
09 MVSEFILLGLSNSWELQVFLFVLFCALYGGIVFGNLLIVITVASNSRLHSPMYFFLANLSLVDLSLSSVTAPKMITDFFSQHKVISFKGCLAQIFLLHFFGGSELVILIAMAFDRYIAICKPLHYTTIMSDKVCIGIVAAAWGIGFLHSV
SQLAFAVCLPFCGPNEVDSFYCDLPRVIKLACTDTYRLDIMVIANSGVLTVFSFILLIISYTIILMTIQYHPSGRSSKALSTLSAHITVVLLFFGPCIFIYAWPFPIKSLDKFLAVFYSVVTPFLNPIIYTLRNKDMKTAMKWLSQWNVNSSVNFZ
10 MVTEFIFLGLSDSQDLQIFLFVFFFVFYVGIVFGNLLIIIAVASDPHLHSPMYFLLANLSLIDLSLSSVTAPKMITDFFSKHKVISFKGCLAQIFLLHFFGGSELVILIAMAFDRYVAICKPLRYTTIMCGNVCVGIVFAAWGIGFLHSV
SQLAFAVNLPFCGPNEIDSFYCDLPRVIRLACTDTYRLGIMVIANSGVLTVCSFVLLIISYTIILKTIQHRPSDRSSKALSTLTAHVTVVLLFFGPCIFIYAWPFPIKSLDKFFAVFYSVVTPLLNPIIYTLRNKDMKTAMRKLKNRNVNSRVKSZ
11 VVTEFIFLGLSNSQELQIFLFLFFFVFYVGIVFGNLLIVITVASDSHLHSPMYFLLANLSLIDLSLSSVTAPKMIADFFRTRKVISFKGCLAQIFLLHLFGGSELVILIAMAFDRYTAICKPLRYTTIMCGNVCVGIVAAAWGIGFLHSV
SQLAFAVNLPFCGPNEVDSFYCDLPRVIKLACTDTYRLDIMVIANSGVLTVCSFVLLIISYTIILMTIQRRPSDRSSKALSTLTAHITVVLLFFGPCIFIYAWPFPVKSLDKFLAVFYSVVTPLLNPIIYTLRNKDMKTAMRRLGKWNSgggITSZ
12 VVTEFIFLGLSDSQELQIFLFMFFFVFYVGIVFGNLLIVITVVYDSHLHSPMYFLLANLSLIDLSLSSVTAPNMIIDFFRECKIISFKGCLAQIFLLHFFGGSELVILIAMAFDRYIAICKPLHYTAVMQANVCVGIVLTAWGIGFLHSV
SQLAFAVNLPFCGPNEIDSFYCDLPRVVKLACVDTYRLDIMVIANSGVLTVCSFILLIISYTIILMTIQRRPSDKSSKALSTLTAHITVVLLFFGPCIFIYAWPFPIKSLDKFLAVFYSVITPFLNPIIYTLRNKEMKTAMRRLRKWKLNSRVMFZ
13 VVTEFVLIGLSDSPELQIFLFVLFFVFYVGIVFGNLLIVITVTADSHLHSPMYFLLANLSFIDLSLSSVTAPKMITDFFSKRKVISVKGCFTQIFLLHFFGGSEMVTLIAMGFDRYVAICKPLRYSTIMRGNVCIGIVVAAWGVGFLHSV
SQLAFAVNLPFCGPNKVDSFYCDLPRVIKLACADTYRLDIMVIANSGILTLCSFVLLIISYTVILMTIQRRPSDRSYKALSTLTAHITVVLLFFGPCIFIYAWPFTTKSLDKFLAVFYSVVTPLLNPIIYTLRNTEMKTAMRRLRQWNZNFZIKPZ
14 VVTEFILMGLSNSRELQIFLFAFFFVFYVGIVFGNLLIVITVTNDSHLHSPMYFLLANLSFIDLCVSSVTAPKTIADFFYKRKVISVKGCFTQIFLLHFFGGSEMVTLVAMAFDRYVAICKPLSYTTVMRGNVCVSIVATAWAIGFLHSV
SQLAFAISLPFCGPNRVDSFYCDLPRVIKLACAETYRLDIMVIANSGVLSVCSFVLLIISYGIILMTIQRRPSDRSSKALSTLTAHITVVLLFFGPCIFIYAWPFPIKSLDKFLAVFYSVVTPLLNPIIYTLRNTEMKTAMRRLRQWNLSFWVKSZ
15 AVTEFIFLGLSNSQELQLFLFVFFFVFYVGIVFGNLLIVITVASDSHLHSPMYFLLANLSLTDLSLSSVTAPKMITDFFSKRKAISFKGCLAQIFLLHFFGGSELVILLAMAFDRYVAICKPLHYTTIMCSKVCLGIVAAAWGIGFLHSV
SQLAFAVNLPFCGPNKIDSFYCDLPRVIQLACTGTYRLDIMVIANSGVLTVCSFVFLIISYTIILMTIQRLPSDRSSKALSTLTAHITVVLLFFGPCIFIYAWPFPVKSLDKFLAVFYSVVTPLLNPIIYTLRNKDMKTAMRQLRKZNVNFRAKSZ
16 VITEFIFLGLSNSQZLQIFLFMFFFVFYVGIVFGNLLIVVTVASNSHLHSPMYFFLANLSVTDLSLSSVTAPKMLIDFFSKHKVISFRGCFTQIFLLHFFGGSELVILLAMAFDRCVAICKPLHYRTIMCNSICVGIMAAAWGTGLLHSV
SQLAFAVNLPFCGPNKVDSFYCDLPRVIQLACKDTYKLDIMIIANSGVLTVCSFVLLLVSYTIILMTIQRRPTDQSSKALSTLTAHITVVLLFFGPGIFIYACPFPIRSLDKFFAVFCSVVTPLLNPVIYTLRNKDMKTAMRQLRKVNVNFRVKAZ
17 VITEFIFLGLSNSQELQIFLFVFFVVSYVGIVFGNLLIVTTVASDSNLHSPMYFLLANLSVTDLSLSSVTAHKMLIDFFNKCKAISFKGCLAQIFLLHFFGGSELLSLLARVFDRYVATCgPLQCTTIMCSNICIGIVAAAWGIGFLHSV
CQLAFAVNLPFCGHNKVDSFCCDLLRVIQLTCTGNYRLggMVIANSGVLTVgSFILLIISYTIILMTIQHHPSNWSSKVLSTLTAHITVILLFFGPCIFIHAWPFPIKSLDKFLAVFYSVVTPLLNPMIYTLRNKDMKTAMSQLRKMIVNFRLKVZ
18 AVTEFIFLGLSNSQELQIFLFGFFFVFYAGIVLGNLLIILAVAVDSHLHSPMYFLLANLSLIDLSLSSVTAPKMITDFFRNCKVISFKGCLAQIFLLHFFGGSEMVILTAMAFDRYVAICKPLRYNTIMCSNVCVGIVAAAWGIGFLHSV
SQLAFAVNLPFCGPNEVDSFYCDLPRVVKLACTDTYRLDIMVIANSGVLTVCAFILLLISYTVILTTIQHRPSERSSKALSTLTAHISAVLLFFGPCVFIYAWPFPIKSSDKFLAVFYSVVTPFLNPLIYTLRNKDMKTAMRRLRKWNVDSRVNSZ
19 MVTEFVFLGLSDSLELQILLFGFFFVFYAGIVVGNVLIIVTVAFDSHLHSPMYLLLANLSLIDLCLSSVTAPKMMTDFFSISKVISFEGCFAQVFLLHLFGGSELVILIAMAFDRYAAICKPLHYTTIMCSKVCVGMVAAAWGTGFLHSV
SQLAFAVNLPFCGPNEIDSFYCDLPRVVKLACADTYRLDIMIIANSGVLTVCSFILLIISYTIILMTIQRHPSARSSKALSTLTAHISVVLLFFGPCIFIYAWPFPIKALDKFLAVFYSVVTPLLNPIIYTLRNKDMKFAMRRLRKWNVNSRVKSZ
20 MVTEFIFLGLSNSQELQTFLFVFFLVFYVGNVFGNLLIVLTVASDSHLHSPMYFLLANLSLIDLCLSSVSAPKMIADFFSKCKVISFKGCLAQIFLLHFFGGSELVILIAMAFDRYVAICKPLRYTTIMSSNVCVGIVAAAWGTGFLHSV
SQLAFAVNLPFCGPNEVDSFYCDLPRVIKLACTDTYRLDIMVIANSGVLTVCSFVLLIISYTVILVTIQHLPSQKSSKALSTLTAHITVVLLFFGPCIFTYAWPFPIKSLDKFLAVFYSVVTPLLNPIIYTLRNKDMKTAMKRLRKWNMNSRIKPZ
21 MVTEFIFLGLSNSQELQIFLFVFFLLFYVGIVFGNLLIVITVASDSHLHSPMYFLLANLSLIDLCLSSVTAPKMIADFFSKRKVISFKGCFAQIFLLHFFGGSELVILIAMAFDRYVAICKPLRYTTIMRGNVCVGIVAVAWGTGFLHSV
SQLAFAVNLPFCGPNEVDSFHCDLPRVIKLACTDTYRLDIMVIANSGVLTVCSFILLIISYTVILMTIQYLPSERSSKALSTLTAHITVVLLFFGPCIFIYAWPFPIKSLDKFLAVFYSVVTPLLNPIIYTLRNKDMKTAMRRLRKWNMNSRIKSZ
22 MVTEFIFLGLSNSQELQIFLFVFFLVFYVGIVFGNLLIVITVASDSHLQSPMYFLLANLSLIDLCLSSVTAPKMIADFFSKCKVISFKGCFAQIFLLHFFGGSELVILIAMAFDRYVAICKPLRYTTIMRGNVCVGIVAVAWGTGFLHSV
SQLAFAVNLPFCGPNEVDSFHCDLPRVIKLACTDTYRLDIMVIANSGVLTVCSFVLLIISYTVILMTIQYLPSERSSKAMSTLTAHITVVLLFFGPCIFIYAWPFPIKSLDKFLAVFYSVVTPLLNPIIYTLRNKDMKTAMRRLRKWNMNSRIKSZ
23 TVTEFIFLGLSNSQELQIFLFVFFFVFYVGIVFGNLLIVITVASDSHLHSPMYFLLANLSLIDLCLSSVTAPKMIADFFSKCKVISFKGCLAQIFLLHFFGGSELVILIAMAFDRYAAICKPLRYTTTMSGNVCAGFVAAAWGTGFLHSV
SQLAFVVNLPFCGPNEVDSFYCDLPRVIKLACTDTYRLDIMVIANSGVLTVCSFFLLIISNTVILVTIQHLPSETSSKALSTLTAHITVVLLFFRPCIFIYAWPFPIKSLDKFLAVFYSVVTPFLNPMIYTLRYKDMKTAMRRLRKWNINSRMKSZ
24 TVTEFIFLGLSNSQELQIFLFVFFFVFYVGIVFGNLLIVITVASDSHLHSPMYFLLANLSLIDLCLSSVTAPKTIADFFSKCKVISFKGCLAQIFLLHFFGGSELMILIAMAFDRYVAICKPLHYTTIMRGNVCVGFVAAAWGTGFLHSV
SQLAFAVNLPFCGPNEVDSFYCDLPRVVKLACTDTYRLDIMVIVNSGVLTVCSFFLLITSYTVILVTIQHLPSERSSKALSTLTAHITVVLLFFGPCIFIYAWPFPIKSLDKFLAVFYSVVTPFLNPMIYTLRNKDMKTAMRWLRKWNINSRIKSZ
25 TVTEFIFLGLSNSQELQIFLVVFFFVFYVAIVFGNLLIVITVASDSHLHSPMYFLLANLSLIDLCLSSVTAPKMIADFFSKCKAISFKGCLAQIFLLHFFGGSELVILIAMAFDRYVAICKPLRYTTIMRGNVCAGFVAAAWGTGFLHSV
SQLAFAVNLPFCGPNEVDSFYCDLPRVIKLACTDTYRLDIMVIANSGVLTVCSFFLLIISYTVILVTIQHLPSERSSKALSTLTAHITVVLLFFGPCIFIYAWPFPIKSLDKFLAVFYSVVTPFLNPMIYTLRNKDMKTAMRRLRKWNINSRIKSZ
26 TVTEFIFLGLSNSQELQIFLVVFFFVFYVGIVFGNLLIVITVASDSHLHSPMYFLLANLSLIDLCLSSVTAPKMIADFFSKCKVISFKGCLAQIFLLHFFGGSELVILIAMAFDRYVAICKPLRYTTVMRGNVCAGFVAAAWGTGFLHSV
SQLAFAVNLPFCGPNEVDSFYCDLPRVIKLACTDTYRLDIMVVANSGVLTVCSFFLLIISYTVILVTIQHLPSERSSKALSTLTAHITVVLLFFGPCIFIYAWPFPIKSLDKFLAVFYSVVTPFLNPMIYTLRNKDMKTAMRRLRKWNINSRIKSZ
27 MVTEFIFLRLSNSQELQIFLFVFFFVFYIGIVFGNLLIVITVACDSHLHSPMYFLLVNLSLIYLCLSSVTAPKIIAHFFSKCKVISFKCCLAQMFLFHFFGGSELVILIAMAFDRYVAICKPLHYSAIMCGNVCVGIVASAWGTGFLHTV
TQLAFAENLPFCGPNKVVSFYCDLPRVIRLTCTDSYRLDIMVVVNSGVLTVCgFILLIISYTIILVTIQHRPSDRSSKALSTLTAHITVVFLFVGPCIFIYAWPFPIKSLDKFFAMFYSVVTHLLNPIIYTLSNKDTKTAMKCLRNCNVNFRVNAZ
28 MVTEFIFLGLSNSQEFQIFLLVFFFIFYVGIVFGNLLIVITVASDSHLHSPMYFLLANLSLIDLCLSSVTAPKMIADFFSKRKVISFKGCLAQIFLVHFFGGSELVILIAMAFDRYVAICKPLRYSTVMCDHVCFGIVTAAWGTGFLHSV
SQLAFAVNLPFCGPNEVDSFYCDLPRVIKLACIDTYRLDVMVIANSGVLTVCSFVFLIISYAIILVTIQQRPSDKSSKALSTLTAHITVVLLFFGPCIFIYAWPFPIKSLDKSLAVFYSVVTPLLNPIIYTLRNKDMKTAMRRLSKWNVNSSVKFZ
29 MVTEFIFLRLSNSQELQIFPFVFFVFYVGIVCGNLPIVITVASDSYLHSPLYFLLANLSLIDLCLSSVTAPKMIADIFSKHKVISFKGCLAQLFILYFFGGSKLVILIAMAFDRYVTICKPPRYTTVMggggCVGLVAAAWGTGFLHSV
SQLAFAVNLPFCGPNEVDSFYCDLPRVIKLPCTDIYRLDIMVIANTGVLTVCgFIFLIISYAIILVTIQQHPSDKSSKALSTLTAHVTVVLLFFGPCIFIYAWPFPMKSLDEFLPVFYYVVTPVLNPIIYTLRNKDMKTALKWLRKWHVNSRVKSZ
30 MVTEFIFLGLSNSRELQIFLFVFFFVFYVGIVCGNLLIVITVASDSHLHSPMYFLLANLSLTDLCLSSVTAPKMIADFFSKRKVISFKGCLAQIFLLHFFGGSELVILIAMAFDRYIAICNPLRYTTVMCGNVCVGLVAAAWGTGFLHSV
SQLAFAVNLPFCGPNEVDSFYCDLPRVIKLACTDTYRLDIMVIANSGVLTVCSFVFLIISYVIILVTIRQRPSHQSSKALSTLTAHITVVLLFFGPCIFIYAWPFPIKSLDKSLAVFYSVVTPLLNPIIYTLRNKDMKTAMKRLRKWNVNSREKSZ
31 LVSEFVLLGLgSSWETKVFLTLIFSLISLGIIPGNLFIFggVILDYHLHFPMYFLLANLFFIDVDLFFITVPKMTTELSNPYKIISFQSCITQICFIHIMGGMEMVLLIAMAFDRYTAgggggHYLNIRNPTICIgFVTTNWVLGDLHgM
SQFVFIMNLCFCGPNKgKNFYYDFPRIIKLACICGAKFEFIVAASSNVMSMSTFFLLIgSYIFILVTVWECSSKDZSKAHGTGSAHITVVVLSFSPYIFLYVZPFPIRZZCLFLVgggFAITCDLNgAIYTLQNKDIKVIIKgggggggggggggg
32 LVSEFVLLGLgSSWETKVFLTLIFSLISLGIIPGNLFIFggVILDYHLHFPMYFLLANLSLIDVDLFFITVPKMTTELSNPYKIISFQSCMTQICFIHIMGGMEMVLLIAIAFDRYTAICKPLHYLNIRNPTICIgFVTTSWVLGDLHgM
SQFVFIINLCFCGPNKgKNFYYDFPRIIKLACIYGAggEFIVAASSNVMSMSTFFLLIgSYIFILVTVWKCSSKDZSKALGTGSAHITVVVLSFSPYIFLgARPFPIRZZCLFLVgggFAITCDLNgAIYTLQNKDIKVVIKgggggggggggggg
33 MVTEFIFLGLSNSRELQIFLFVFFFVFYVGIVFGNLLIVLTVSFDSHLHSPMYFLLANLSLIDLSLSSVTAPKMIADFFRKRKVISFKGCLTQIFLLHFFGGSELVILIAMAFDRYVAICKPLHYTTIMCGNVCIGIVSAAWGTGFLHSV
```

```
SQLAFAVNLPFCGPNEVDSFYCDLPRVIKLACTDTYRLDIMVIANSGVLTVCSFVLLIISYTIILAIILRRPSDRSSKALSTLTAHITVVLLFFGPCIFIYAWPFPIRSLDKFLAVFYSVVTPLLNPVIYTLRNKDMKTAMRRLKKZNADSRIKSZ
34 MVTEFIFLGLSDSQELQIFLFVFFLVFYVGIVFGNLFIVLTVTSDPHLHSPMYFLLANLSLIDVCVSTVTAPKMIADFFSKRKVISFKGCLTQIFLLHFFGGSELVILIAMAFDRYIAICKPLHYTTIMRGNVCVGLATASWGTGFFHSV
SQFVFAVNLPFCGPNEVDSFYCDLPRVIKLACTDTYKLDIMVIANSGILTLCSFVLLIISYSFILVTILHRPSDRSSKALSTLTAHITVVLLIFGPCIFIYAWPFPIKSLDKFLAVFYSVVTPLLNPIIYTLRNKDMKAAMRRLLKQNVNSRVKTZ
```

Apêndice B

Estudo de caso 1: Filogenias de saída

As filogenias de saída do estudo de caso 1 geradas pelo algoritmo podem ser vistas nas Figuras B.1 e B.2. A Figura B.3 mostra a saída completa obtida pelo Phylip [5].

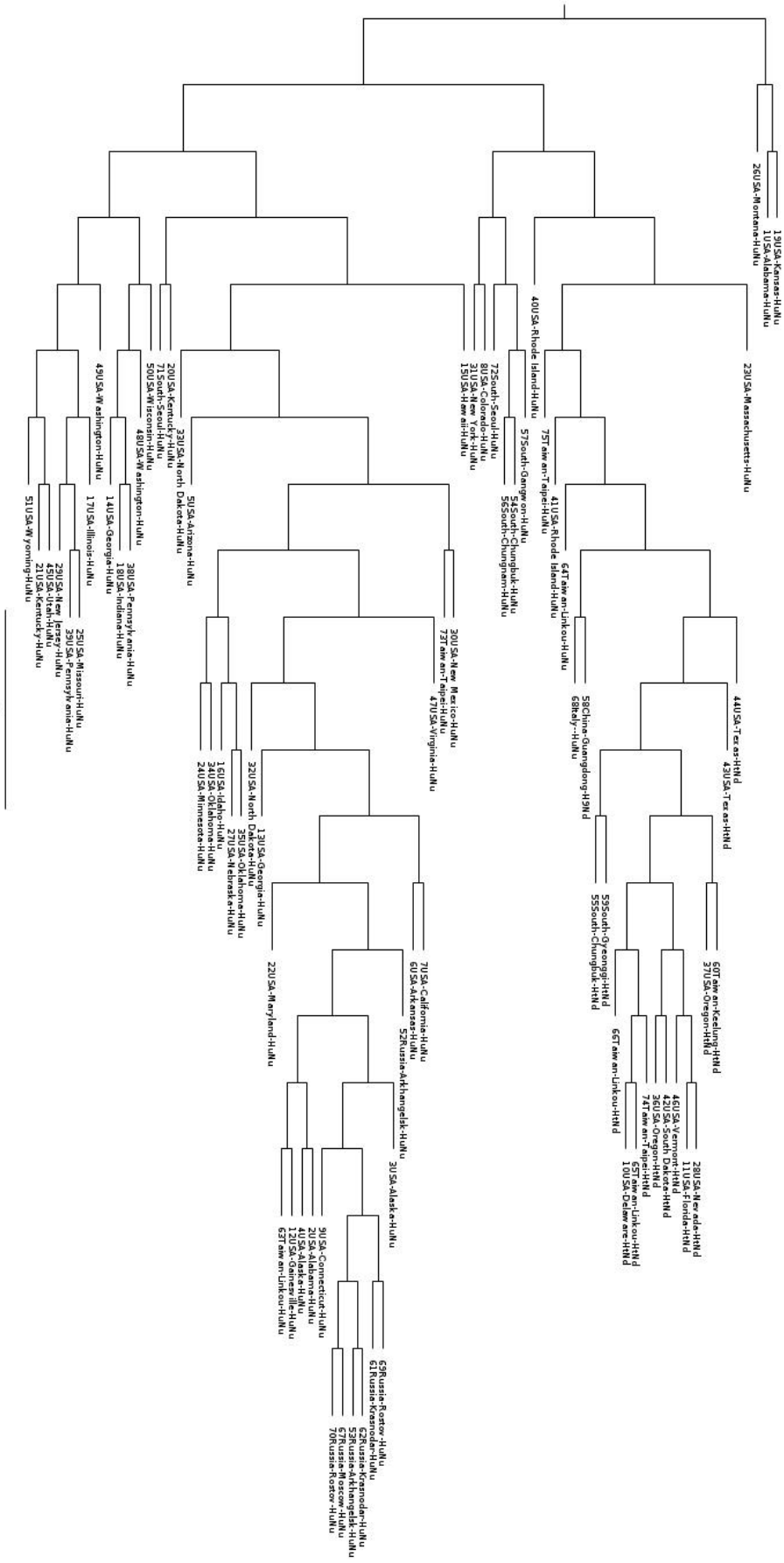


Figura B.1: Filogenia convencional com vírus dos tipos H3N2 (HtNd) e H1N1 (HuNu)

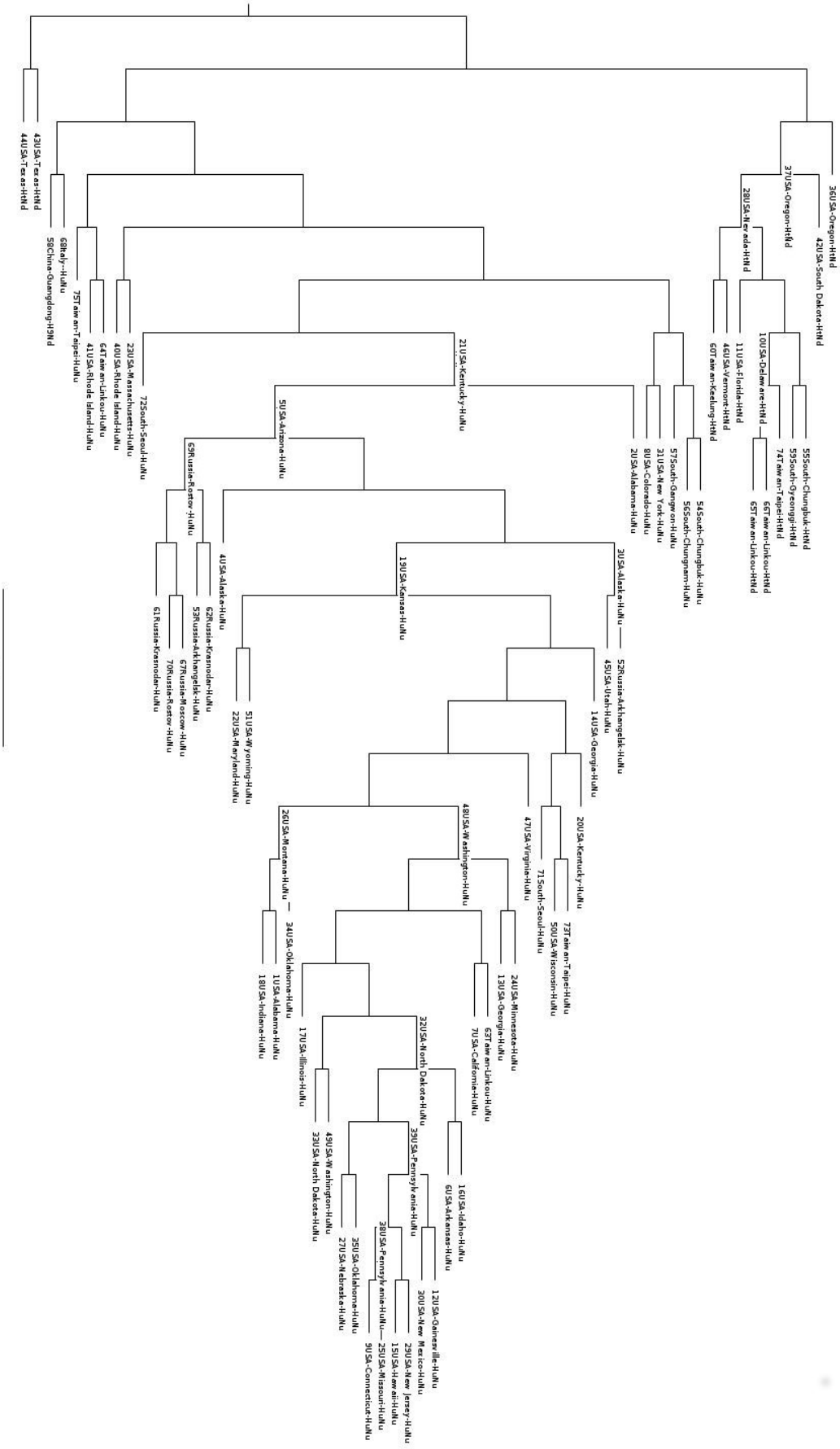


Figura B.2: Filogenia viva com vírus dos tipos H3N2 (HtNd) e H1N1 (HuNu)

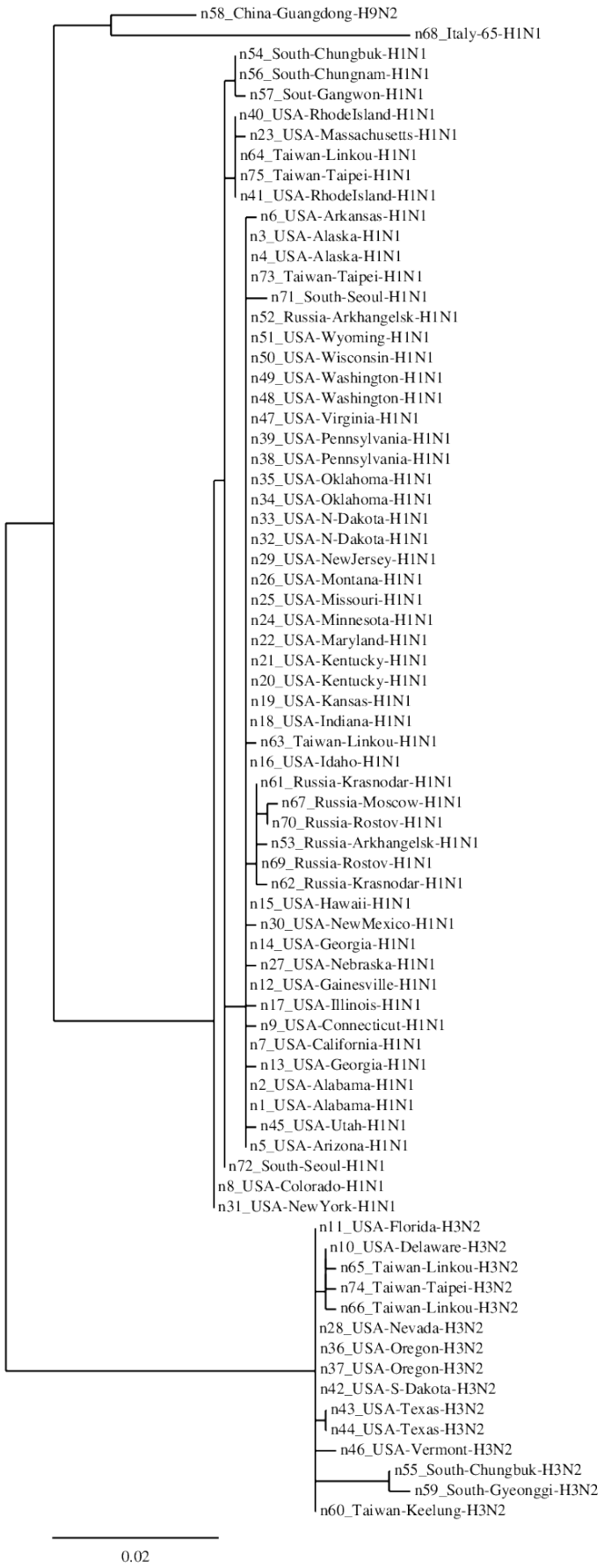


Figura B.3: Filogenia obtida com Phylip, com mesma entrada utilizada nas Figuras B.1 e B.2.

Apêndice C

Estudo de caso 2: Filogenias de saída

As filogenias de saída do estudo de caso 2 citadas nesse trabalho e geradas pelo algoritmo podem ser vistas nas Figuras C.1, C.2 e C.3. A filogenia mostrada na Figura C.4 foi obtida com o uso de *Neighbor Joining*, sendo utilizado o *jModelTest* [28] para indicar o melhor modelo, e o software PAUP [42] para executar o *Neighbor Joining*. Foi inserido um indivíduo chamado *outgroup* para ajudar na criação da raiz.

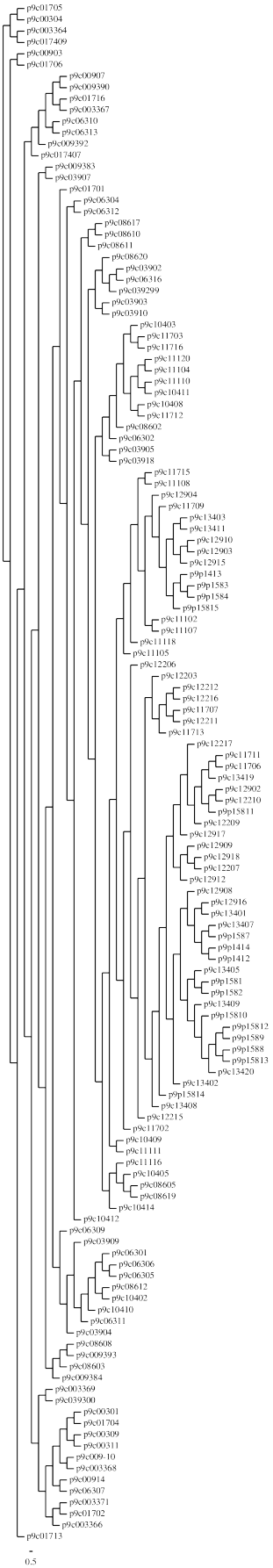


Figura C.1: Filogenia convencional obtida pelo algoritmo genético,

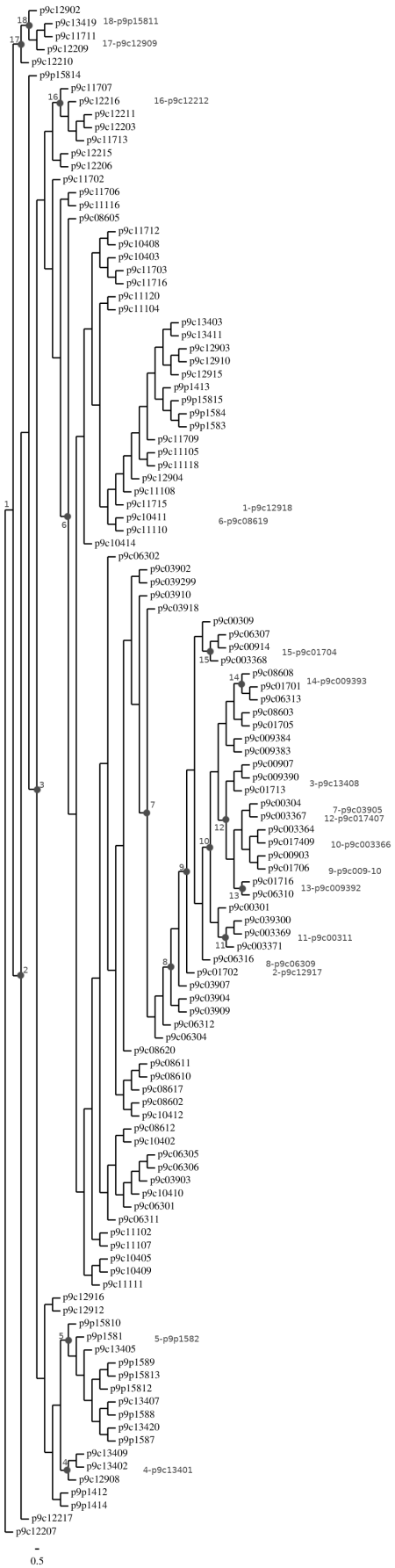


Figura C.2: Filogenia viva contendo 18 nós vivos

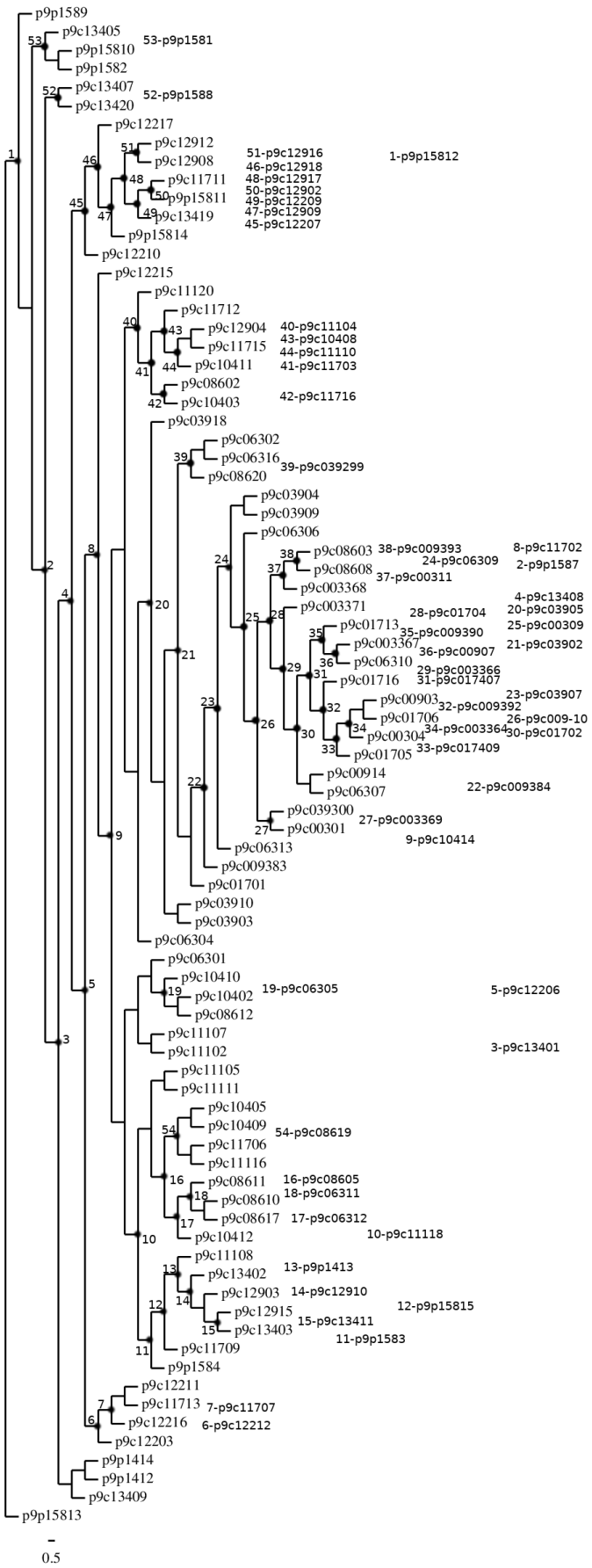


Figura C.3: Filogenia viva contendo 54 nós vivos

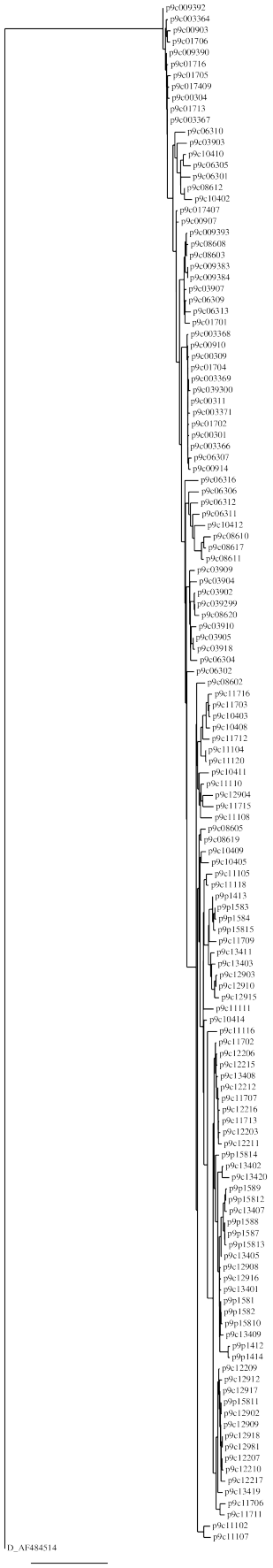


Figura C.4: Filogenia convencional obtida com a utilização de Neighbor Joining

Referências

[1] HIV sequence database - special interest alignment set 1. https://www.hiv.lanl.gov/content/sequence/HIV/SI_alignments/set1.html. Acesso em 20/04/2018. 52

[2] NCBI influenza virus sequence database. ftp://ftp.ncbi.nih.gov/genomes/INFLUENZA/. Acesso em 20/04/2018. 52

[3] Phylogeny - biology online. http://www.biology-online.org/dictionary/Phylogeny. Acesso em 12/01/2017. 1

[4] F. N. Abuali, R. L. Wainwright, and D. A. Schoenefeld. A new encoding scheme for spanning trees applied to the probabilistic minimum spanning tree problem. *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 470–477, 1995. 25

[5] B. R. Baum. Phylip: Phylogeny inference package (version 3.2). *The Quarterly Review of Biology*, 64:539–541, 1989. 64, 74

[6] E. Borenstein. Small parsimony and search algorithms. http://elbo.gs.washington.edu/ courses/GS_559_11_wi/slides/10A-Parsimony_Trees_2.pdf. Acesso em 13/11/2016. vi, ix, 9, 29, 30, 31

[7] P. Buneman. A note on the metric properties of trees. *Journal of Combinatorial Theory*, B:48–50, 1974. 12

[8] J. A. Cavender and J. Felsenstein. Invariants of phylogenies: Simple case with discrete states. *Journal of Classification*, 4:57–71, 1987. 10

[9] R. C. Chakraborty. Fundamentals of genetic algorithms. http://www.myreaders.info/ 09-Genetic_Algorithms.pdf. Acesso em 20/11/2016. vi, 3, 22, 23, 24, 27

[10] B. S. D. J. Zwickl. *Genetic Algorithm Approaches for the Phylogenetic Analysis of Large Biological Sequence Datasets Under the Maximum Likelihood Criterion*. PhD thesis, The University of Texas, May 2006. vi, 3, 25, 32, 34, 37

[11] B. DasGupta, X. He, T. Jiang, et al. On computing the nearest neighbor interchange distance. In *Proc. Dimacs Workshop on Discrete Problems with Medical Applications*, pages 125–143. Press, 1997. 9

[12] A. W. F. Edwards and L. L. Cavalli-Sforza. Reconstruction of evolutionary trees. pages 67–76. Systematics Association Publications, 1964. 9

[13] J. Felsenstein. Evolutionary trees from DNA sequences: a maximum likelihood approach. *Journal of Molecular Evolution*, 17(6):368–76, 1981. 33

[14] W. M. Fitch. Toward defining the course of evolution: minimum change for a specified tree topology. *Systematic Zoology*, 20:406–416, 1971. 8, 29, 40, 46

[15] D. E. Goldberg, K. Deb, and G. Rawlins. A comparative analysis of selection schemes used in genetic algorithms. Foundations of Genetic Algorithms, Morgan Kaufmann, 1991. 26

[16] A. Gupta, J. Manuch, L. Stacho, and C. Zhu. Small phylogeny problem: character evolution trees. In *Combinatorial Pattern Matching*, pages 230–243. 2004. 2

[17] M. Hasegawa, T. Kishino, and T. Yano. Dating of the human-ape splitting by a molecular clock of mitochondrial DNA. *Jornal of Molecular Evolution*, 22(2):696–704, 1985. 10, 33

[18] T. H. Jukes and C. Cantor. Evolution of protein molecules. In H. N. Munro, editor, *Mammalian Protein Metabolism*, pages 21–132. Academic Press, 1969. 10

[19] M. Kimura. A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. *Journal of Molecular Evolution*, 16(2):111–20, 1980. 33

[20] J. A. Levy. Pathogenesis of human immunodeficiency virus infection. *Microbiological Reviews*, 57:183–289, 1993. 66

[21] P. O. Lewis. A genetic algorithm for maximum-likelihood phylogeny inference using nucleotide sequence data. *Molecular Biology and Evolution*, 15(3):277–83, 1998. vi, 3, 25, 32, 33, 34, 35, 42, 45

[22] K. F. Man, K. S. Tang, and S. Kwong. *Genetic Algorithms: Concepts And Designs*. Springer, 2001. 3

[23] M. Melanie. *An Introduction to Genetic Algorithms*. The MIT Press, fifth edition, 1999. 3

[24] T. P. Meyer, N. H. Packard, M. Casdagli, and S. Eubank. Local forecasting of high-dimensional chaotic dynamics. In *Nonlinear Modeling and Forecasting*. Addison-Wesley. 26

[25] M. Mitchell. *Introduction to Genetic Algorithms*. The MIT Press, 1999. 1, 21, 26

[26] C. C. Palmer and A. Kershenbaum. Representing trees in genetic algorithms. *Proceedings of the First IEEE Conference on Evolutionary Computation*, pages 379–384, 1994. 25

[27] B. A. Pierce. *Genetics: A conceptual Approach*. W. H. Freeman, 3rd edition edition, 2007. 1

[28] D. Posada. jmodeltest: phylogenetic model averaging. *Molecular Biology and Evolution*, 25(7):1253–6, 2008. 67, 78

[29] G. R. Raidl and B. A. Julstrom. Edge sets: an effective evolutionary coding of spanning trees. *IEEE Transactions on Evolutionary Computing*, 7(3):225–239, 2003. 24

[30] D. F. Robinson and L. R. Foulds. Comparison of phylogenetic trees. *Biosci*, 53:131–147, 1981. 36

[31] F. Rothlauf and D. Goldberg. Tree network design with genetic algorithms - an investigation in the locality of the pruefernumber encoding. *Late Breaking Papers at the Genetic and Evolutionary Computation Conference*, pages 238–43, 1999. 25

[32] F. Rothlauf and D. E. Goldberg. Pruefer numbers and genetic algorithms: A lesson how the low locality of an encoding can harm the performace of gas. *Lecture Notes in Computer Science*, 1917:395–404, 2000. 25

[33] N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, pages 405–425, 1987. 17, 67

[34] M. Salemi, A. Vandamme, and P. Lemey. *The Phylogenetic Handbook: A pratical approach to phylogenetic analysis and hypothesis testing*. Cambridge University Press, second edition, 2009. 2

[35] D. Sankoff. Minimal mutation trees of sequences. *SIAM Journal of Applied Mathematics*, 28:35–42, 1975. 8, 29, 32, 47

[36] J. Setubal and J. Meidanis. *Introduction to Computational Molecular Biology*. PWS Publishing Company, 1997. 1, 2, 5, 7

[37] R. Shankarappa, J. B. Margolick, S. J. Gange, et al. Consistent viral evolutionary changes associated with the progression of human immunodeficiency virus type 1 infection. *Journal of Virology*, 73(12):10489–10502, 1999. 52

[38] S. Skiena. *The Algorithm Design Manual*. Springer Science+Business Media, 2nd edition, 2010. 9

[39] W. Sung. *Algorithms in Bioinformatics: A practical introduction*. CRC Press, 2010. 8, 11, 14

[40] W. Sung, H. S. Negi, L. Haiquan, X. Zhou, P. S. Wasan, and K. P. Wei. Lecture 7: Phylogenetic trees reconstruction. https://www.comp.nus.edu.sg/ksung/cs5238/2002Sem1/note/lecture7.pdf, 2002. Acesso em 13/11/2016. vi, 6, 8

[41] W. Sung, K. Ning, T. Shan, S. L. Xiang, and W. Shen. Lecture 8: Phylogenetic tree reconstruction: Distance based. http://www.comp.nus.edu.sg/bioinfo/phylogenetic%20tree%20reconstruction_2_8.pdf, 2003. Acesso em 28/11/2016. vi, ix, 13, 14, 15, 18

[42] D. L. Swofford. *Phylogenetic Analysis Using Parsimony (and Other Methods)*. Sinauer Associates, 2002. 67, 78

[43] D. L. Swofford and D. P. Begle. Paup: phylogenetic analysis using parsimony. http://paup.csit.fsu.edu/, 1993. Acesso em 28/11/2016. 25, 33, 41

[44] G. P. Telles, N. F. Almeida, R. Minghim, and M. E. M. T. Walter. Live phylogeny. *Journal of Computational Biology*, 20(1):30–37, 2013. vi, 2, 3, 19, 20, 21

[45] M. Vingron, J. Stoye, and H. Luz. Algorithms for phylogenetic reconstructions. http://lectures.molgen.mpg.de/ Algorithmische_Bioinformatik_WS0405/phylogeny_script.pdf, 2002. Acesso em 29/11/2016. vi, ix, 15, 16, 33

[46] M. S. Waterman, T. F. Smith, M. Singh, and W. A. Beyer. An optimal algorithm to reconstruct trees from additive distance. *Journal of Theoretical Biology*, 64:199–213, 1977. 15, 20