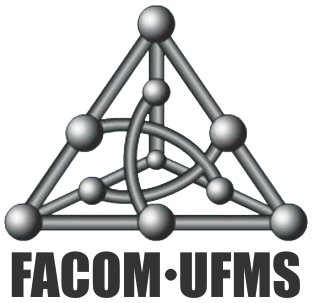
Parcimônia para Filogenia Viva

Tese de Doutorado

Rogério Güths

Orientação: Prof. Nalvo Franco de Almeida Jr.

Área de concentração: Bioinformática



Faculdade de Computação
Universidade Federal de Mato Grosso do Sul

Campo Grande, Maio de 2019

Agradecimentos

Meus mais profundos agradecimentos: ao orientador, por ter me aceitado como orientando após tanto tempo do meu mestrado, ter tido a paciência de começar uma caminhada do zero no tema Bioinformática, por ter repartido comigo uma parte de sua criação (Filogenia Viva) para que eu tivesse uma fatia do assunto para trabalhar, pela determinação em ler e acompanhar o meu trabalho embora diversas outras demandas estivessem aflorando, por ter me mostrado que uma pessoa pode ser altamente competente, ocupada, requisitada, mas, mesmo assim, ainda ter a sensibilidade de um ser humano e ser feliz; ao professor Marcelo Henriques de Carvalho, que em suas aulas de Análise de Algoritmos mostrou que um assunto difícil pode ser trabalhado com naturalidade e que, se um professor conseguir ter empatia com seus alunos o desenrolar da disciplina é muito melhor; ao pesquisador Guilherme Pimentel Telles pelas diversas contribuições, críticas e incentivo, em especial no direcionamento do trabalho; à pesquisadora Maria Emilia Machado Telles Walter pelas contribuições e incentivo, em particular na revisão criteriosa e valorosa do texto no momento da qualificação; aos colegas Graziela, Deiviston, Robson e Delair pela parceria, sugestões e incentivo ao trabalho; à minha esposa por ter dado suporte e incentivo entre altos e baixos durante estes mais de 4 anos de trabalho desenvolvido; à minha filha por me trazer pílulas de alegria diárias, tão importantes para seguir em frente; por fim à própria UFMS por ter me permitido o afastamento, onde por 4 anos pude me dedicar integralmente a este estudo.

Resumo

Este trabalho apresenta os principais conceitos e problemas relacionados ao estudo de Filogenia Viva baseada em Características, que é uma extensão do problema tradicional de filogenia baseada em características, porém admitindo a presença de objetos entre os nós internos da árvore. A principal abordagem para o tratamento do problema é baseada no critério da parcimônia, que procura minimizar eventos de reversão e evolução paralela, eventos evolutivos comuns e que dificultam a solução do problema. O tema da Parcimônia para Filogenia Viva é abordado em detalhes, com a caracterização do problema e a apresentação de possíveis soluções para dois subproblemas em separado: o Problema da Parcimônia Pequeno Viva, quando se tem como entrada, além de uma matriz de características, a topologia da árvore desejada, e o Problema da Parcimônia Grande Viva, quando a topologia da árvore não é dada. Para o Problema da Parcimônia Pequeno Viva são apresentadas duas soluções polinomiais, enquanto que para o Problema da Parcimônia Grande Viva apresentamos duas soluções baseadas em técnicas de *branch-and-bound*, além de duas heurísticas com diferentes abordagens. O Problema da Parcimônia Grande Viva é então subdividido em dois outros problemas, com base na informação sobre quantos ou quais são os nós internos vivos presentes na solução. No primeiro caso, sabemos quantos são os nós internos vivos, porém não sabemos quais são eles. No segundo caso, sabemos exatamente quais são eles. Provamos que estes dois problemas são NP-completos e propomos algoritmos e heurísticas para eles. Todos os algoritmos e heurísticas propostos foram implementados. Por fim, testes foram realizados para verificar o desempenho das heurísticas sugeridas. Os testes validaram a maioria das heurísticas propostas e fortaleceram a importância do estudo de Filogenia Viva como meio de representação do processo evolutivo.

Palavras-chave: Filogenia, Evolução, Parcimônia, Topologia, Heurísticas.

Abstract

This work presents the main concepts and problems related to the study of Live Phylogeny based on Characteristics, which is an extension of the traditional problem of Phylogeny based on characteristics, but admitting the presence of objects between internal nodes of the tree. The main approach to the problem is based on the parsimony criterion, which seeks to minimize reversal and parallel evolution events, common evolutionary events that increase the difficulty of solving the problem. The subject of Parsimony for Live Phylogeny is discussed in more detail, with problem definition and presentation of possible solutions for two separate subproblems: the Small Parsimony Problem, when the input, in addition to the characteristic matrix, has the topology of desired tree, and Large Parsimony Problem, when such topology is not given. For the Small Live Parsimony Problem two polynomial solutions are presented, whereas for the Large Live Parsimony Problem two solution based on branch-and-bound techniques, and two heuristics with different approaches are presented. The Large Live Parsimony Problem is then subdivided into two other problems, based on the information on how many or which are the live internal nodes present in the solution. In the first case we know how many live internal nodes there are in the solution, but we do not know what they are. In the second case we know exactly what they are. These two problems are proven to be NP-complete and we proposed algorithms and heuristics for them. All proposed algorithms and heuristics were implemented. Finally, tests were performed to verify the performance of suggested heuristics. The tests validated most of the proposed heuristics and strengthened the importance of the study of Live Phylogeny as a means of representing the evolutionary process.

Key words: Phylogeny, Evolution, Parsimony, Topology, Heuristics.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Tendo como base o processo evolucionário das espécies, busca-se explicar a história evolutiva das espécies existentes, além de estabelecer os relacionamentos entre elas, identificando possíveis ancestrais comuns. Este problema é chamado *problema da filogenia* [50].

Para representar a história evolucionária e as relações entre as espécies, são utilizadas árvores, chamadas *árvores filogenéticas* ou *filogenias*. Uma filogenia é uma árvore na qual as espécies existentes são representadas pelas folhas e os nós internos representam ancestrais hipotéticos. As arestas representam possíveis alterações evolutivas [50].

Embora a motivação principal para o estudo de filogenia esteja na Biologia, ela pode ser empregada em quaisquer outras áreas em que haja um processo evolutivo envolvido, por exemplo, documentos e registros de bancos de dados. No caso geral, os elementos para os quais se deseja representar a história evolutiva são chamados *objetos taxonômicos*, ou simplesmente *objetos*.

As filogenias são construídas com base nas comparações entre os objetos e, dependendo do tipo de dado de entrada, tem-se uma das seguintes abordagens: *filogenia baseada em distâncias* ou *filogenia baseada em características*.

Em filogenia baseada em distâncias tem-se como base valores numéricos que representam as distâncias evolucionárias entre os objetos. A entrada é uma matriz quadrada M em que cada elemento M_{ij} representa a distância evolucionária não-negativa do objeto i para o objeto j.

O tópico de interesse deste trabalho, no entanto, é o problema da filogenia baseada em características e, neste caso, são utilizadas características discretas como: forma do bico, número de dedos na pata, presença ou ausência de certas proteínas, etc. Cada característica pode ou não ter um número finito de valores, também chamados estados da característica. Essas características são agrupadas em uma matriz de entrada M, em que cada linha representa um objeto e cada coluna uma característica. Desta forma, M_{ij} representa o estado que a espécie i tem para a característica j. Esta matriz é chamada matriz de estados. A Tabela 1.1 apresenta um exemplo de matriz de estados. Apesar de existirem casos nos quais as características possuem correlação, será assumido nesta tese que as características evoluem independentemente umas das outras.

Tabela 1.1: Exemplo de matriz de estados.

		Características		
		Patas	Cor Olhos	Reprodução
Espécies	Cão	4	castanho	gestação
	Rã	4	castanho	ovo
	Ema	2	castanho	ovo
	Arara	2	azul	ovo
	Pato	2	vermelho	ovo

Em uma filogenia baseada em características, cada objeto existente está representado por uma folha na árvore e a tupla de estados da característica da espécie é chamada *rotulação* do nó. Aos nós internos, que representam ancestrais hipotéticos, também devem ser associadas tuplas de estados da característica, calculadas durante a construção da filogenia. A Figura 1.1 mostra o exemplo de uma filogenia construída para a Tabela 1.1 com uma possível rotulação.

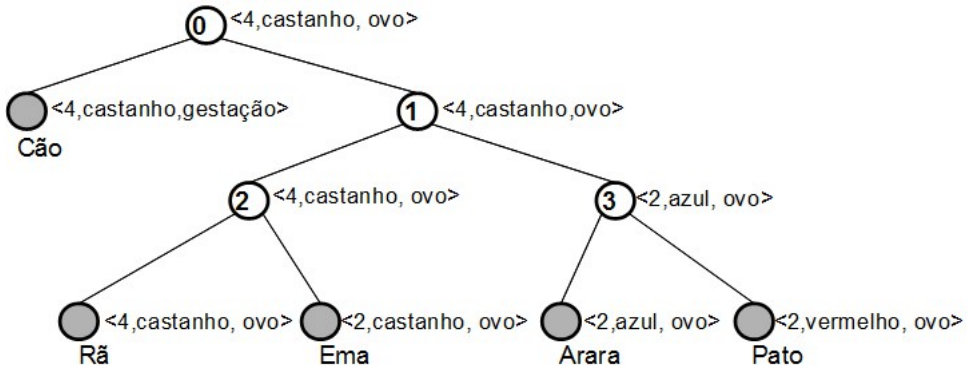


Figura 1.1: Exemplo de filogenia.

Conforme Setubal e Meidanis [50], a construção de uma filogenia a partir de uma matriz de estados depara-se com duas principais dificuldades, descritas a seguir:

- Convergência ou evolução paralela. Os métodos para reconstrução da árvore filogenética baseiam-se no fato de que objetos que compartilham o mesmo estado para uma dada característica são geneticamente mais relacionados que aqueles que não compartilham. Entretanto, existe a possibilidade que dois objetos compartilhem um estado mas não sejam evolutivamente próximos ou, pelo menos, não estejam próximos na árvore filogenética. Tal fenômeno é chamado de *convergência* ou *evolução paralela*.
- Reversão de estados. Tal dificuldade diz respeito à relação entre os estados de uma dada característica. Esta ocorre quando, para uma dada característica, a filogenia apresenta no percurso de um ancestral até uma espécie na árvore, mudanças de um estado x para outro estado y e posterior retorno do estado y para o estado x.

Conforme o relacionamento entre os estados, as características são classificadas como *ordenadas* ou *não-ordenadas*. Para as características não-ordenadas não há restrições quanto à mudança de estados, com qualquer estado podendo evoluir para qualquer outro. No caso das características ordenadas pode haver restrição: a ordem em que podem haver mudanças. Por exemplo, em uma característica de 3 estados podemos ter a seguinte ordem: $1 \leftrightarrow 2 \leftrightarrow 3$, significando que o estado 1 para uma dada característica só pode ser transformado para o estado 2 nesta mesma característica, e assim o estado 2 para o estado 3. Características ordenadas são chamadas *cladísticas* e características não-ordenadas são chamadas *qualitativas* [8].

De forma mais simples, o que se deseja é uma árvore filogenética sem eventos de convergência e reversão de estados. Para isto, a árvore obtida deve ter a seguinte propriedade: o conjunto de todos os nós (objetos) que possuem o mesmo estado para uma determinada característica deve formar uma subárvore da árvore construída. Esta filogenia é chamada de *filogenia perfeita* e, neste caso, diz-se que as características são *compatíveis*. A seguir a definição do problema da filogenia perfeita [50].

Definição 1.1. *Problema da Filogenia Perfeita (PP)*
Instância: Um conjunto S com n objetos, um conjunto C de m características, cada

característica possuindo no máximo k estados possíveis (n, m, k inteiros positivos). Questão: Existe uma filogenia perfeita para S?

Para o caso de características ordenadas e para o caso em que o número de estados possíveis é constante, em especial no caso de características binárias, existem algoritmos polinomiais que decidem PP e apresentam a solução. Entretanto, PP é NP-Completo para o caso de características não-ordenadas [3].

Quando é impossível a obtenção de uma filogenia perfeita, é feito um relaxamento nas restrições do PP. Uma das possíveis abordagens é a da *parcimônia*, que será vista em detalhes no Capítulo 2. Nessa abordagem, o objetivo é a obtenção de uma filogenia com o menor número possível de transições.

Até aqui supõe-se, como foi dito, que a árvore filogenética resultante deve ser tal que os objetos sejam representados pelas folhas, enquanto os nós internos da árvore representam ancestrais hipotéticos.

Telles e colegas [52] propuseram uma nova família de problemas de filogenia, em que admite-se a existência de objetos que podem ser ancestrais e, ao mesmo tempo, co-existirem. Dessa forma, admite-se que um objeto possa ser representado ou como uma folha ou como um nó interno na árvore, neste caso chamado de *nó interno vivo*. O problema geral foi chamado de *Filogenia Viva*. Naquele trabalho, um algoritmo polinomial foi apresentado para resolver o problema da filogenia perfeita no caso de características binárias, considerando a filogenia viva.

No nosso trabalho usamos o termo *filogenia tradicional* para referenciar o problema onde nós internos vivos não são permitidos, e *filogenia viva* caso contrário.

Esta nova caracterização do problema da filogenia demanda a revisão, adaptação e ampliação das análises e algoritmos formulados para o problema da filogenia tradicional. Conforme as informações a respeito dos objetos taxonômicos de entrada, é possível uma das seguintes situações:

- podem existir, ou não, nós internos vivos;
- existem $l > 0$ nós internos vivos, desconhecidos previamente;
- existem $l > 0$ nós internos vivos, conhecidos previamente.

A primeira situação será tratada no Capítulo 5. As demais serão abordadas separadamente no Capítulo 6.

Na filogenia tradicional, conforme Felsenstein [13], para n objetos têm-se $3 \times 5 \times 7 \times \cdots \times (2n-3)$, denotado por Felsenstein por $(2n-3)!!$, possíveis árvores filogenéticas. No caso de filogenia viva, podemos perceber um aumento substancial no número de possíveis árvores, visto que, além das árvores tradicionais (sem nós internos vivos), é necessário considerar todas as possíveis árvores com $l = 1, 2, \ldots, \lfloor \frac{n-1}{2} \rfloor$ nós internos vivos.

Até o momento não existe uma fórmula que sintetize o número total de possíveis árvores para filogenia viva, mas a Tabela 1.2 apresenta os valores até $n = 10$, nos dando uma ideia da magnitude do problema. O crescimento no número de árvores é expressivo e a última linha da Tabela 1.2 sugere que a relação entre quantidade de árvores para filogenia viva e quantidade de árvores para filogenia tradicional não seja linear.

Tabela 1.2: Quantidade de árvores para filogenia viva.

		Quantidade de objetos								
		2	3	4	5	6	7	8	9	10
	0	1	3	15	105	945	10.395	135.135	2.027.025	34.459.425
Nós	1		3	24	225	2.520	33.075	498.960	8.513.505	162.162.000
internos	2				60	1.350	26.460	529.200	11.226.600	255.405.150
vivos	3						3.150	141.120	4.762.800	149.688.000
	4								317.520	23.814.000
Total		1	6	39	390	4.815	73.080	1.304.415	26.847.450	625.528.575
Relação Tradicional Viva		1	2	2,6	3,71	5,10	7,03	9,65	13,24	18,15

Considerando o problema de filogenia viva, Telles e colegas [52] definem o conceito de Filogenia Perfeita Viva conforme a Definição 1.2. Observe que esta definição permite que T tenha nós internos rotulados por objetos fornecidos como entrada. Além da definição, Telles e colegas apresentaram um algoritmo polinomial que constrói uma filogenia perfeita viva a partir de uma matriz cujas colunas sejam comparáveis.

Definição 1.2. *Filogenia Perfeita Viva*
Seja M uma $n \times m$ matriz binária cujas linhas são rotuladas por $o_1, o_2 ... o_n$, e cujas colunas são rotuladas $c_1, c_2 ... c_m$ e são disjuntas ou comparáveis aos pares. Uma filogenia perfeita viva para M é uma árvore enraizada T tal que:

- *cada aresta em T é rotulada por um c_j distinto;*
- *cada objeto o_i rotula exatamente um nó em T;*
- *para cada nó rotulado por o_i, $M_{i,j} = 1$ se e somente se c_j está no caminho da raiz de T até o nó rotulado por o_i.*

Dada uma árvore enraizada T, um *nó interno vivo* v é um nó interno de T previamente rotulado por uma tupla de m estados da característica. O objeto de trabalho desta pesquisa consistiu no estudo do problema da filogenia viva baseada em características. Em particular, buscou-se o desenvolvimento de algoritmos e heurísticas para o problema da parcimônia em filogenia viva.

Na sequência deste texto serão apresentadas uma síntese da teoria e abordagens propostas para filogenia tradicional, começando com a definição do problema, complexidade e métodos de resolução exata, que vai estar no Capítulo 2; e heurísticas no Capítulo 3. Como veremos, essas heurísticas para filogenia tradicional são úteis para a formulação de heurísticas para filogenia viva. O leitor já familiarizado com estes conceitos e métodos, pode iniciar diretamente a leitura do Capítulo 4.

O Capítulo 4 trata exclusivamente de uma solução para o problema da parcimônia pequeno em sua versão para filogenia viva, ou seja, quando a topologia da árvore filogenética é dada como entrada e o que se deseja é rotular os seus nós. Essa versão para filogenia viva para o problema da parcimônia pequeno foi originalmente apresentada por nós em [24].

No Capítulo 5 o problema da parcimônica grande para filogenia viva é tratado, dando ênfase primeiramente para uma argumentação sobre sua complexidade, além de dois algoritmos baseados em *branch-and-bound* e três heuristicas para solucioná-lo. A definição do problema e uma das técnicas de *branch-and-bound* foram originalmente apresentadas em [25], enquanto que uma das heurísticas foi apresentada em [26].

O problema grande para filogenia viva então gera outros dois problemas: o caso em que existem $l > 0$ nós internos vivos, desconhecidos previamente, é tratado na Seção 6.1; já o caso em que existem $l > 0$ nós internos vivos conhecidos previamente é abordado na Seção 6.2.

No Capítulo 7 são apresentados os resultados de diversos testes comparando as técnicas de *branch-and-bound* e heurísticas propostas, incluindo aplicação para conjuntos de sequências de vírus. Por fim, no Capítulo 8, são apresentadas as conclusões deste trabalho.

Capítulo 2

Parcimônia para filogenia tradicional

No caso geral, PP é NP-Completo [3]. Além disso, nos casos práticos, é improvável que as matrizes de estados admitam filogenia perfeita, pois os dados de entrada podem conter erros e os requisitos de não ocorrer reversão de estados nem evolução paralela, às vezes, são violados [50]. Por isso, são necessárias outras abordagens para reconstruir a filogenia.

Duas abordagens são sugeridas: a primeira busca a minimização dos eventos de reversão de estados e evolução paralela, é conhecida como *critério da parcimônia*. A segunda é conhecida como *critério da compatibilidade*, que propõe evitar tais eventos excluindo as características que causam tais problemas. Neste caso, tenta-se encontrar um subconjunto máximo de características compatíveis entre si. Ambos os critérios resultam em problemas de otimização NP-difíceis para características ordenadas [8, 9] e não-ordenadas [9, 22].

Quando usamos o critério da parcimônia, doravante denominado simplesmente parcimônia, buscamos minimizar o número de transições de estado em uma filogenia, supondo que 0 (zero) é o estado ancestral. Se for permitida reversão, mas proibida evolução paralela, chama-se parcimônia Dollo [12]. Por outro lado, se for permitida a evolução paralela e proibida a reversão, chama-se parcimônia Camin-Sokal [4], a qual explica os dados assumindo que mudanças de estado $0 \rightarrow 1$ são permitidas, mas mudanças $1 \rightarrow 0$ não. A parcimônia Wagner [11] permite ambos os tipos de mudanças. Neste trabalho será utilizada a parcimônia Wagner.

Antes de definir formalmente os problemas relacionados à parcimônia, é necessário estabelecer uma função de custos, que quantifique o esforço evolucionário necessário para realizar uma mutação entre um nó pai e seu nó filho, quando da alteração de alguma característica.

Seguindo a abordagem de Jones e Pevzner [32], têm-se uma matriz de estados M que representa um conjunto S de n espécies com m características e Σ um alfabeto de k símbolos representando todos os possíveis estados da característica. Cada salto evolucionário tem um custo, que representa o esforço necessário para realizar esta mudança. Este custo é representado por δ, uma matriz $k \times k$, onde δ_{ab} representa o custo da mudança de um estado a para o estado b para uma dada característica.

Dada uma filogenia T para S, se dois nós v e w com as respectivas rotulações $(v_1, v_2, ..., v_m)$ e $(w_1, w_2, ..., w_m)$ estão conectados por uma aresta em T, então o custo da aresta (v, w), denotado por vw, é a soma de todos os custos de mudança dos estados da característica de v para w, considerando a matriz δ. Assim,

$$vw = \sum_{i=1}^{m} \delta_{v_i w_i}.$$

Somando todos estes custos vw de todas as arestas, obtêm-se um escore que representa o esforço total desta história evolucionária, chamado *escore parcimônia*, ou *custo*, e denotado por $S(T)$, isto é,

$$S(T) = \sum_{(v,w)\in T} vw = \sum_{(v,w)\in T} (\sum_{i=1}^{m} \delta_{v_i w_i}). \tag{2.1}$$

A Figura 2.1 ilustra uma árvore rotulada T. Se utilizarmos como custo de mudança a função $\delta_{ij} = 1$ quando $i \neq j$ e 0 quando $i = j$ teremos o custo da árvore $S(T) = \sum_{(v,w)\in T} vw = 1 + 2 + 1 + 1 = 5$.

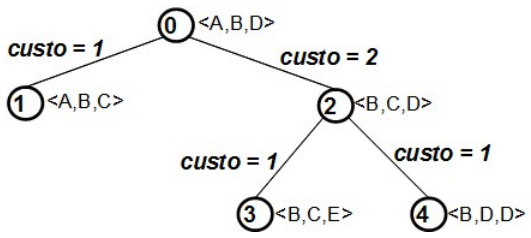


Figura 2.1: Árvore T rotulada e o cálculo do custo.

O problema de encontrar, dentre todas as possíveis topologias, a árvore T que admita uma rotulação tal que $S(T)$ seja mínimo é conhecido como o *Problema da Parcimônia Grande*.

Antes de descrever soluções para esse problema, será abordado na Seção 2.1 um problema mais fácil, conhecido como *Problema da Parcimônia Pequeno*. Esse problema tem solução interessante, sob o ponto de vista computacional, e além disso será útil na solução do Problema da Parcimônia Grande, discutido na Seção 2.2.

2.1 Problema da Parcimônia Pequeno

O problema da parcimônia pequeno consiste em, dada uma filogenia T, determinar uma rotulação que gere o menor escore parcimônia, ou seja, busca-se uma rotulação que minimize as mudanças requeridas para explicar as relações de uma árvore dada como entrada. Jones e Pevzner [32] definem o problema da seguinte maneira:

Definição 2.1. *Problema da Parcimônia Pequeno (SPP)*
Encontre a rotulação de todos os vértices internos de uma árvore evolucionária de escore parcimônia mínima.
Entrada: Uma árvore T com cada folha rotulada por elementos de uma sequência de m estados da característica.
Saída: Uma rotulação dos vértices internos de T minimizando o escore parcimônia $S(T)$.

Este problema pode ser resolvido em tempo polinomial [32]. O objetivo é encontrar uma rotulação (atribuição de valores para cada uma das características) para os nós internos da árvore, levando a um escore parcimônia mínimo. Observe que foi assumido que as características evoluem independentemente, desta forma os algoritmos apresentados realizam o cálculo do escore parcimônia e rotulação relativos a apenas uma característica. Para obter o escore parcimônia total basta somar os escores obtidos em cada característica. A rotulação total é a união das rotulações obtidas em cada característica.

A Figura 2.2 ilustra a resolução do problema SPP para uma árvore T com as folhas rotuladas por uma sequência de 3 estados da característica, onde utilizarmos como custo de mudança a função $\delta_{ij} = 1$ quando $i \neq j$ e 0 quando $i = j$.

	Patas	Cor Olhos	Reprodução
Cão	4	castanho	gestação
Rã	4	castanho	ovo
Ema	2	castanho	ovo
Arara	2	azul	ovo
Pato	2	vermelho	ovo

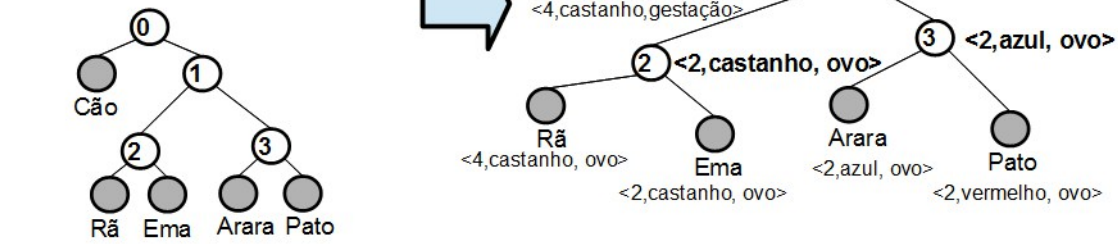


Figura 2.2: Exemplo de resolução do Problema da Parcimônia Pequeno.

A seguir apresentamos dois algoritmos que resolvem polinomialmente SPP, conhecidos como Algoritmo de Fitch e Algoritmo de Sankoff.

Algoritmo de Fitch

Se δ é definido de tal forma que $\delta_{ij} = 1$ quando $i \neq j$ e 0 quando $i = j$, temos a chamada *distância Hamming*. Para este caso, o algoritmo de Fitch [13] fornece uma maneira muito simples e rápida de calcular uma rotulação dos vértices internos de uma árvore T minimizando o escore parcimônia. Como as características evoluem independentemente umas das outras, o algoritmo apresentado considera apenas uma característica.

O algoritmo de Fitch (Algoritmo 2.1) usa uma estrutura auxiliar, chamada *possivel* : $V \rightarrow \Sigma^*$, que armazena, para cada nó v, um conjunto de possíveis valores para rotular v. O algoritmo é dividido em três partes.

A primeira parte define *possivel* para as folhas. Se v é uma folha rotulada por t, então define $possivel(v) = \{t\}$.

A segunda parte percorre a árvore em pós-ordem definindo *possivel* para os nós internos. Se v é um nó interno com filhos v_{left} and v_{right}, então define

Algoritmo 2.1 Fitch.

Entrada: (1) Uma árvore $T = (V, E)$ com $root \in V$ sendo a raiz de T, (2) um alfabeto de k letras Σ e (3) uma função $label : V \to \Sigma \cup \{\lambda\}$ rotulando todas as folhas por elementos de Σ.

Saída: (1) Uma função $label' : V \to \Sigma$ minimizando o escore parcimônia.

Parte I: Define *possivel* para as folhas;
for $\forall v \in V$ **do**
  if $label(v) \neq \lambda$ **then**
    $possivel(v) = \{label(v)\}$
  end if
end for
Parte II: Percorre T em pós-ordem definindo *possivel* para os demais nós;
for $\forall v \in V$ nó interno **do**
  if $label(v) = \lambda$ **then**
    if v tem dois filhos v_{left} , v_{right} **then**
      if $possivel(v_{left}) \cap possivel(v_{right}) = \emptyset$ **then**
        $possivel(v) = possivel(v_{left}) \cup possivel(v_{right})$
      else
        $possivel(v) = possivel(v_{left}) \cap possivel(v_{right})$
      end if
    end if
  end if
end for
Parte III: Percorre T em pré-ordem definindo $label'$;
$label'(root) = rand(possivel(root))$
for $\forall v \in V$ **do**
  if $label(v) \neq \lambda$ **then**
    $label'(v) = label(v)$;
  end if
  if $label(v) = \lambda$ com pai v_{father} **then**
    if $label'(v_{father}) \in possivel(v)$ **then**
      $label'(v) = label'(v_{father})$;
    else
      $label'(v) = rand(possivel(v))$;
    end if
  end if
end for

$$possivel(v) = \begin{cases} possivel(v_{left}) \cup possivel(v_{right}), \\ \qquad\qquad \text{se } possivel(v_{left}) \cap possivel(v_{right}) = \emptyset \\ possivel(v_{left}) \cap possivel(v_{right}), \\ \qquad\qquad \text{se } possivel(v_{left}) \cap possivel(v_{right}) \neq \emptyset. \end{cases}$$

Na terceira parte percorre a árvore em pré-ordem. Iniciando pela raiz r, rotula r por um elemento qualquer de $possivel(r)$. Então, para cada nó v com pai v_{father} é rotulado pela seguinte regra:

$$rotulo(v) = \begin{cases} rotulo(v_{father}), & \text{se } rotulo(v_{father}) \in possivel(v) \\ \text{qualquer elemento de } possivel(v), & \text{caso contrário.} \end{cases}$$

A Figura 2.3 ilustra as três partes da execução do algoritmo Fitch para uma característica com a rotulação sendo exibida no interior do nó.

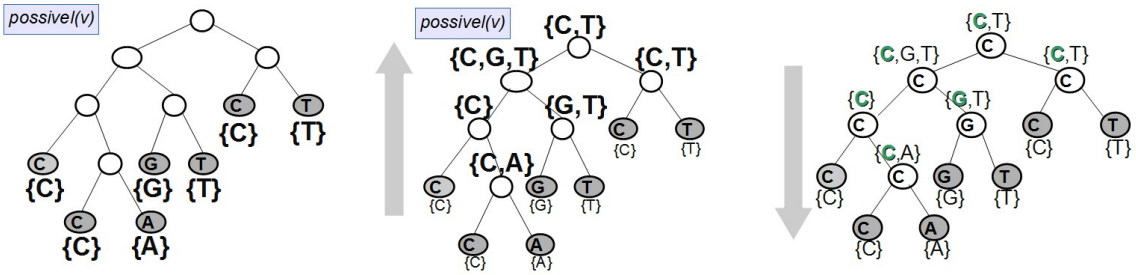


Figura 2.3: Execução do algoritmo Fitch.

Algoritmo Sankoff

Para o caso geral de δ_{ij}, ou seja, quando δ_{ij} são inteiros positivos quaisquer, o algoritmo de Sankoff [13] (Algoritmo 2.2) fornece uma maneira correta e genérica de calcular o escore parcimônia e uma rotulação para SPP. Este algoritmo é baseado na técnica de Programação Dinâmica. Da mesma forma como no algoritmo de Fitch, apresentamos o algoritmo considerando apenas uma característica.

O algoritmo calcula, para cada vértice v e cada estado da característica t, $s(v,t)$ o escore parcimônia mínimo da subárvore com raiz v assumindo que o vértice v seja rotulado por t. Iniciando nas folhas, em que a rotulação já está definida, o valor $s(v,t)$ é definido pela seguinte regra:

$$s(v,t) = \begin{cases} 0, & \text{se } v \text{ é rotulado por } t \\ \infty, & \text{caso contrário.} \end{cases}$$

Esta definição faz sentido pois a rotulação do vértice está definida e não vai mudar. No próximo passo o algoritmo calcula, para cada vértice interno v com filhos u e w e para cada estado da característica t, o escore parcimônia mínimo $s(v,t)$ considerando o custo da mudança de t para cada estado da característica e o escore parcimônia deste estado da característica no vértice filho:

$$s(v,t) = \min_i(s(u,i) + \delta_{it}) + \min_j(s(w,j) + \delta_{jt}). \tag{2.2}$$

O algoritmo faz uso de uma estrutura auxiliar $melhor : V \times \Sigma \times \{left, right\} \to \Sigma$ que armazena, para cada nó v, em cada estado da característica t e cada um dos filhos, à esquerda e à direita, o estado da característica do filho que fornece o menor valor para $\min_i$ e $\min_j$ na Fórmula 2.2, respectivamente.

Seja *root* a raiz de T. Na última etapa do algoritmo é selecionado um estado da característica t_{root} em *root* cujo escore parcimônia seja mínimo. Este escore parcimônia é o escore parcimônia daquela característica e *root* é rotulada por t_{root}. A partir daí o algoritmo percorre a árvore em pré-ordem, rotulando os nós da seguinte maneira: se o nó visitado é folha, nada precisa ser feito; se o nó visitado é um nó interno v rotulado por um estado da característica t_v, sejam v_{left} e v_{right} seus filhos, estes filhos são rotulados por $melhor(v, t_v, left)$ e $melhor(v, t_v, right)$, respectivamente.

O algoritmo assegura o escore parcimônia mínimo em cada vértice e cada estado da característica, de forma que, ao final, têm-se na raiz os escores parcimônia mínimos de cada estado da característica [13].

A Figura 2.4 ilustra as duas primeiras partes da execução do algoritmo Sankoff para uma característica, considerando a função δ_{ij} definida na Tabela 2.1. Na figura

Algoritmo 2.2 Sankoff.

Entrada: (1) Uma árvore $T = (V, E)$ com $root \in V$ sua raiz, (2) um alfabeto de k letras Σ , (3) uma função $label : V \to \Sigma \cup \{\lambda\}$ rotulando todas as folhas por elementos de Σ e (4) uma $k \times k$ matriz de custos δ_{ij} .

Saída: (1) Uma função $label' : V \to \Sigma$ minimizando o escore parcimônia.

Parte I: Define $s(v, t)$ para folhas;

for $\forall v \in V$ **do**
  if $label(v) \neq \lambda$ **then**
    for $\forall t \in \Sigma$ **do**
      $s(v, t) = \infty$;
    end for
    $s(v, label(v)) = 0$;
  end if
end for

Parte II: Percorre T em pós-ordem calculando $s(v, t)$ para demais nós;

for $\forall v \in V$ nó interno com filhos v_{left} , v_{right} **do**
  for $\forall t \in \Sigma$ **do**
    $s(v, t) = \min_i(s(v_{left}, i) + \delta_{it}) + \min_j(s(v_{right}, j) + \delta_{jt})$;
    $melhor(v, t, left) = i$;
    $melhor(v, t, right) = j$;
  end for
end for

Parte III: Percorre T em pré-ordem definindo $label'$;

$label'(root) = i | \min_i s(root, i)$
for $\forall v \in V$ **do**
  if v nó interno com filhos v_{left} , v_{right} **then**
    $label'(v_{left}) = melhor(v, label'(v), left)$;
    $label'(v_{right}) = melhor(v, label'(v), right)$;
  end if
end for

a rotulação é exibida no interior do nó. A Figura 2.5 ilustra as última parte da execução do algoritmo Sankoff. Observe que as setas menores indicam a informação contida no *melhor* de cada rotulação escolhida.

Tabela 2.1: Exemplo de função δ_{ij} para $\mathcal{S}$ ={A, C, G, T}.

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

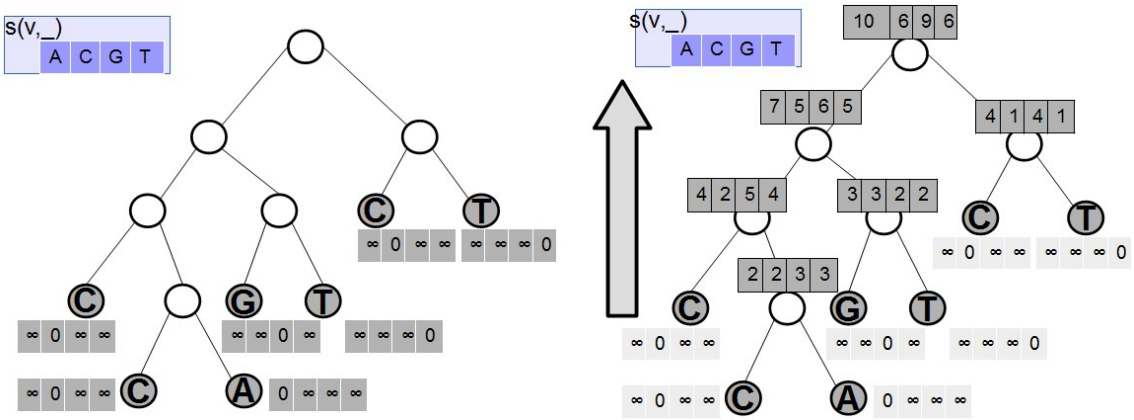


Figura 2.4: Execução do algoritmo Sankoff parte I e II.

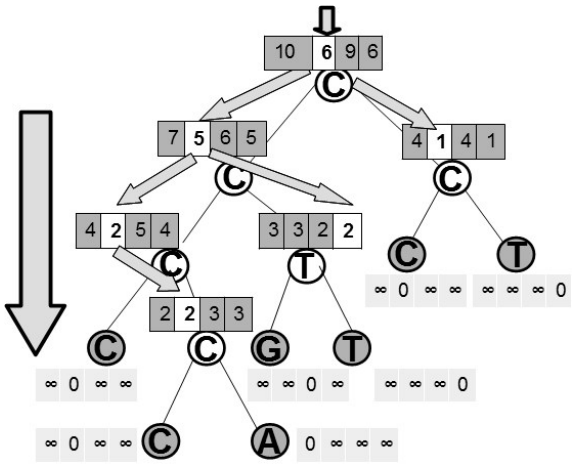


Figura 2.5: Execução do algoritmo Sankoff parte III.

Conforme Jones e Pevzner [32], ambos os algoritmos de Fitch e Sankoff resolvem SPP em tempo polinomial. Mais especificamente, o algoritmo de Fitch tem complexidade $O(k.n)$ e o de Sankoff complexidade $O(k^2.n)$, com $k = |\Sigma|$.

2.2 Problema da Parcimônia Grande

Em geral, a filogenia não é previamente conhecida e não basta apenas calcular o custo de uma dada filogenia, é necessário encontrar a topologia da árvore que minimiza este custo. No problema da parcimônia grande busca-se a melhor árvore no espaço de todas as árvores possíveis.

O problema da parcimônia grande é definido em Jones e Pevzner [32] como um problema de otimização e foi transformado no problema de decisão definido abaixo para utilização em provas de NP-completude. Considera-se a função δ_{ij} como a distância de Hamming e o escore parcimônia ($S(T)$) definido na Equação 2.1 (pg. 10).

Definição 2.2. *Problema da Parcimônia Grande (LPP)*
Instância: Uma matriz $M_{n\times m}$ e uma constante $B \in \mathbb{R}_+$.
Questão: Existe uma árvore T, completamente rotulada, com exatamente n folhas rotuladas pelas n linhas da matriz M, tal que $S(T) \leq B$?

Uma árvore T que possibilite rotulação com escore parcimônia mínimo é chamada *árvore mais parcimoniosa*. Convém observar que esta árvore pode não ser única, ou seja, podem existir várias árvores com o mesmo escore parcimônia mínimo.

LPP foi provado ser NP-completo por Graham e Foulds [16, 22], mesmo se o conjunto de estados da característica for binário.

Busca Exaustiva para Parcimônia Grande

Uma possível resolução para LPP é a utilização de estratégia de *backtracking* para montagem de todas as possíveis árvores para uma dada instância do problema e subsequente cálculo do escore parcimônia através dos algoritmos polinomiais que resolvem Parcimônia Pequeno. Desta forma pesquisa-se todo o espaço de busca, ou seja, todas as possíveis árvores e, ao final, retorna-se a de menor escore parcimônia.

Esta solução não é aceitável pois o conjunto de possíveis árvores a serem geradas cresce exponencialmente. Conforme Felsenstein [13], dado um grupo de n nós folha, existem $3 \times 5 \times 7 \times ... \times (2n-3)$ possíveis árvores bifurcadas enraizadas.

Árvores Âncora

De acordo com Ford e colegas [15], existe uma outra forma de busca pela árvore mais parcimoniosa, utilizando o conceito de *ilhas filogenéticas* e *Árvores Âncora* dentro do espaço de busca.

Maddison [39] propôs o conceito de ilhas filogenéticas, que são conjuntos de árvores com escore parcimônia menor que um dado valor L, cada uma podendo ser obtida a partir da outra através de alguma operação de rearranjo de árvore, por exemplo permutando ramos da árvore, sem visitar árvore de escore maior ou igual a L. Este conceito é útil para matrizes com dados incompletos que podem ter várias árvores ótimas, conforme apresentado por Sanderson e colegas [48]. Para um dado L, podem existir diversas ilhas.

Considerando C o conjunto de m características, seja $C_1, C_2, ..., C_l$ uma partição de C tal que $\nexists i C_i = \emptyset$, $\forall i, j \;\; C_i \cap C_j = \emptyset$, $C_1 \cup C_2 \cup ... \cup C_l = C$ e $\forall i C_i$ é formada por características compatíveis. A última condição é necessária para garantir que possamos construir uma filogenia perfeita para cada C_i. Esta partição de características é dita *compatível*. Ford e colegas [15] constroem uma ilha filogenética para cada C_i, com uma visão um pouco diferente da proposta de Maddison [39], pois utiliza fundamentalmente distância. Como cada C_i é formado por características compatíveis, é possível desconsiderar as demais características e construir uma filogenia perfeita T_{C_i} para as características de C_i, a qual é chamada de *árvore âncora*. As arestas destas árvores âncora são chamadas *arestas âncora*. Ford e colegas [15] constroem todas as árvores âncora; em seguida incluem novamente as demais características. Cada árvore induz uma ilha filogenética com relação à sua vizinhança.

Como as características são independentes, os autores mostram que uma árvore mais parcimoniosa deve existir a partir de um número fixo de passos a partir das diversas árvores âncora, e para isso usam um relaxamento da métrica Robinson-Foulds [46], explicada em seguida.

Métrica Robinson-Foulds

A métrica de Robinson-Foulds leva em consideração duas operações sobre árvores filogenéticas: contração e resolução. Dada uma árvore T, a operação de contração de dois nós adjacentes u e v, gera uma árvore T' na qual a aresta (u, v) e os nós u e v são substituídos por um nó $\{u, v\}$.

A operação inversa é a resolução a qual, dada uma árvore T e um nó $\{u, v\}$, gera uma árvore T' em que o nó $\{u, v\}$ é substituído pelo nós u e v ligados pela aresta (u, v) e as arestas que eram conectadas a $\{u, v\}$ são aleatoriamente conectadas aos nós u ou v.

Observe que uma sequência de operações de contração pode gerar nós que acumulam vários nós da árvore original.

Definição 2.3. *Distância Robinson-Foulds*
Dadas duas árvores T_1, T_2 com n folhas, a distância Robinson-Foulds, $d_{RF}(T_1, T_2)$, é o número mínimo de contrações e resoluções necessários para converter T_1 em T_2.

Robinson e Foulds [46] propõem um algoritmo linear para calcular a distância entre duas árvores quaisquer, como apresentado por Day [7]. Apesar disso, para alcançar os resultados desejados, pode ser utilizada uma versão mais simples. Antes de apresentar esta versão mais simples, é necessário definir o conceito de compatibilidade entre arestas e árvores.

Definição 2.4. *Compatibilidade entre arestas e árvores*
Dadas duas árvores T_1, T_2 com n folhas, uma aresta (u, v) em T_2 é compatível com T_1 se existe uma aresta (s, t) em T_1 tal que a partição dos nós folhas induzida por (u, v) em T_2 é igual à induzida por (s, t) em T_1.

Definição 2.5. *Distância Robinson-Foulds Relaxada*
Dadas duas árvores T_1, T_2 com n folhas, a distância Robinson-Foulds Relaxada, $dr_{RF}(T_1, T_2)$, é o número de arestas em T_2 que são incompatíveis com a árvore T_1.

Observe que dr_{RF} não é uma métrica, pois não é simétrica. Para árvores binárias $dr_{RF} = d_{RF}$.

Pode-se utilizar qualquer outra medida de distância, dentre as quais a quantidade de operações *NNI*(*Nearest Neighbor Interchange*), *SPR* (*Subtree Pruning and Regrafting*) ou *TBR* (*Tree Bisection and Reconnection*), que serão vistas na Seção 3.2. Entretanto, o problema de calcular a distância para duas árvores quaisquer utilizando uma destas três medidas geram problemas NP-completos [6].

Resumidamente, a abordagem baseada em árvores âncora consiste nos seguintes passos:

- particiona-se o conjunto de características em subconjuntos compatíveis;
- constrói-se e calcula-se o escore parcimônia de cada árvore âncora para cada subconjunto de características compatíveis;
- calcula-se uma configuração inicial para o espaço de busca restrito, utilizando uma árvore inicial; e
- percorre-se exaustivamente o espaço de busca, reduzindo este espaço cada vez que se encontrar uma árvore melhor, mediante o recálculo do diâmetro de busca.

Considere $S(T)_{Ci}$ o escore parcimônia da árvore T calculado somente sobre as características de C_i.

A base desta abordagem reside no fato de que, dados uma árvore qualquer T e um subconjunto de características compatíveis C_i com sua respectiva árvore âncora T_{C_i} têm-se: $S(T)_{Ci} \geq S(T_{C_i})_{Ci} + dr_{RF}(T_{C_i}, T)$. Ford e colegas [15] provam esta desigualdade, a qual é válida pois T_{C_i} é filogenia perfeita. Como as características são independentes, o escore parcimônia de T, considerando C, é maior ou igual à soma dos escores de todas as árvores âncora e da distância Robinson-Foulds Relaxada entre estas árvores e T, ou seja,

$$S(T)_C \geq \sum_{i=1}^{l} S(T_{C_i})_{Ci} + dr_{RF}(T_{C_i}, T),$$

em que l é o número de subconjuntos da partição.

Como as árvores âncora T_{C_i} são as que tem o melhor escore parcimônia, considerando somente as características de C_i, o resultado acima indica que uma solução ótima possui um número calculável de arestas que são compatíveis com as arestas das árvores âncora. Dito de outra maneira, uma árvore candidata não pode ter um número excessivo de arestas incompatíveis com as arestas das árvores âncora, pois se isto ocorrer $\sum_{i=1}^{l} dr_{RF}(T_{C_i}, T)$ será muito alto e a árvore candidata certamente deve ser descartada.

Ford e colegas [15] definem um limite inferior estático M para o escore parcimônia como a soma do número de mudanças de estados de característica considerando todas as características. Por exemplo, se se pretende resolver Parcimônia Grande para o

conjunto de espécies apresentado na matriz da Tabela 2.2, o número de mudanças da característica 1 é igual a 1 (pois usa apenas dois valores) e da característica 3 é igual a 2. Neste caso $M = 1 + 1 + 2 + 1 = 5$.

Tabela 2.2: Matriz de estados com estados da característica $\mathcal{S}$ ={A, C, G, T}.

		Características			
		1	2	3	4
Espécies	1	A	G	C	C
	2	A	G	G	G
	3	C	C	T	G

Ford e colegas [15] utilizam M para definir o diâmetro da busca $Diam = min_i(S(T_{C_i})) - M$. Na etapa de busca, propõe que sejam geradas árvores a partir das árvores âncora. Para tanto usa $Diam$ para calcular quantas arestas uma árvore candidata T precisa ter compatíveis com as árvores âncora. Seja $|arestas|$ a quantidade total de arestas de árvores âncora diferentes (incompatíveis entre si). T precisa ter, pelo menos, $|arestas| - Diam$ arestas compatíveis com alguma árvore âncora. Desta forma, Ford propõe que sejam construídas todas as combinações possíveis contendo $|arestas| - Diam$ arestas de árvores âncora. Estas combinações são árvores não binárias, então precisam ser testadas todas as árvores binárias obtidas a partir destas.

A prova do resultado apresentado por Ford e colegas [15] não exige minimalidade no tamanho da partição (quantidade de subconjuntos compatíveis), apesar do espaço de busca ser menor quanto menor for a partição.

É importante observar que, se o número de conjuntos de características compatíveis necessários para cobrir todo o conjunto de características for muito grande, pode ser que a delimitação do espaço de busca via árvores âncora retorne o espaço de busca inteiro, o que significaria não haver ganho na utilização desta abordagem.

A Tabela 2.3 mostra um exemplo para melhor entendermos essa abordagem, com 9 espécies para as quais busca-se a árvore mais parcimoniosa utilizando árvores âncora.

Lembrando os passos desta abordagem, o primeiro consiste em particionar o conjunto de características de forma a obter uma partição compatível. É fácil verificar que {{1,2}, {3,4}, {5,6}} é uma partição compatível.

Para cada subconjunto é construída uma árvore âncora conforme ilustrado na Figura 2.6. A árvore da Figura 2.6(a) é a filogenia perfeita construída a partir do

Tabela 2.3: Matriz de estados utilizada no exemplo de aplicação de árvores âncora com estados da característica $\mathcal{S}$ ={A, C, G, T}.

		Características					
		1	2	3	4	5	6
Espécies	1	A	G	C	C	C	C
	2	A	G	C	G	A	T
	3	C	C	C	G	A	T
	4	C	C	A	A	A	T
	5	C	T	A	A	A	G
	6	C	T	A	T	C	A
	7	A	G	C	G	C	C
	8	C	C	C	C	C	A
	9	C	T	A	A	A	T

conjunto de características {1,2}. Já a da Figura 2.6(b) é a filogenia perfeita construída a partir do conjunto de características {3,4}. Finalmente a da Figura 2.6(c) é a filogenia perfeita construída a partir do conjunto de características {5,6}.

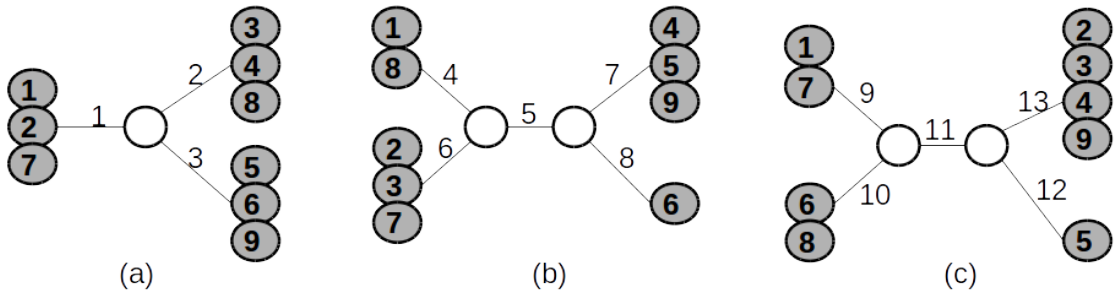


Figura 2.6: Árvores âncora para subconjuntos de características (a) {1, 2} (b) {3, 4} (c) {5, 6}.

Em seguida são listadas todas as arestas âncora que compõem as árvores âncora. Na Tabela 2.4 estão enumeradas todas as 13 partições induzidas pelas arestas âncora. Cada coluna da tabela apresenta as partições geradas na respectiva árvore da Figura 2.6. Por exemplo, a aresta 1, apresentada na Figura 2.6(a) separa as espécies 1, 2 e 3 das demais, o que é mostrado na primeira linha da primeira coluna da Tabela 2.4.

A próxima etapa é a delimitação do espaço de busca. Analisando cada uma das características, o limite inferior estático é dado por $M = 1+2+1+3+1+3 = 11$.

Considera-se a árvore inicial apresentada na Figura 2.7, que foi obtida por uma heurística. A Figura 2.8 mostra a rotulação obtida aplicando-se o algoritmo Fitch resultando no escore parcimônia 18.

Tabela 2.4: Partições induzidas pelas arestas âncora geradas por cada uma das árvores âncora da Figura 2.6.

Árvore (a)	Árvore (b)	Árvore (c)
1) 127 - 345689	4) 18 - 2345679	9) 17 - 2345689
2) 348 - 125679	5) 12378 - 4569	10) 68 - 1234579
3) 569 - 123478	6) 237 - 145689	11) 1678 - 23459
	7) 459 - 123678	12) 5 - 12346789
	8) 6 - 12345789	13) 2349 - 15678

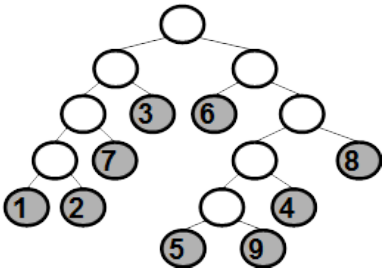


Figura 2.7: Árvore inicial.

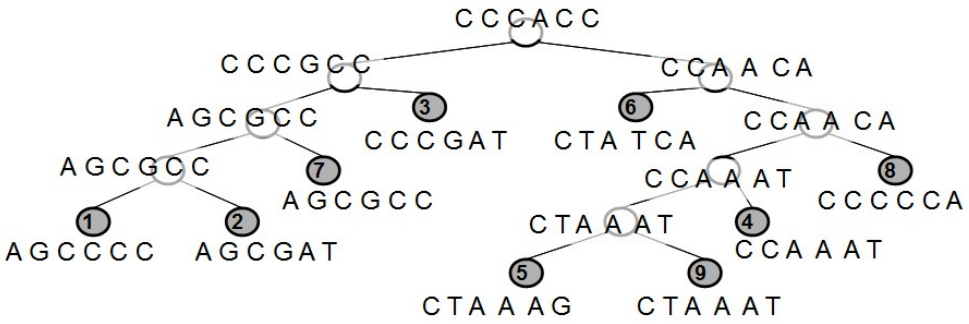


Figura 2.8: Árvore inicial rotulada.

Desta forma, o diâmetro desta busca será igual à diferença entre o melhor escore parcimônia até o momento e o limite inferior estático, ou seja, $Diam = 18 - 11 = 7$.

O número mínimo necessário de arestas âncora em uma possível solução ótima é dado pelo número total de arestas âncora menos o diâmetro da busca. No exemplo, temos 13 arestas âncora e diâmetro 7, portanto é necessário ter pelo menos 6 arestas âncora. Então são geradas todas as possíveis árvores, possivelmente não resolvidas completamente, com 6 arestas âncora.

Muitas combinações geradas são incompatíveis e descartadas, por exemplo não é possível combinar as arestas 5 e 10 pois a aresta 10 coloca as espécies 6 e 8 no mesmo ramo, isoladas das demais, e a aresta 5 coloca as espécies 6 e 4 no mesmo ramo, isoladas da 8. Da busca exaustiva de todas as combinações resultam apenas

6, apresentadas na Figura 2.9.

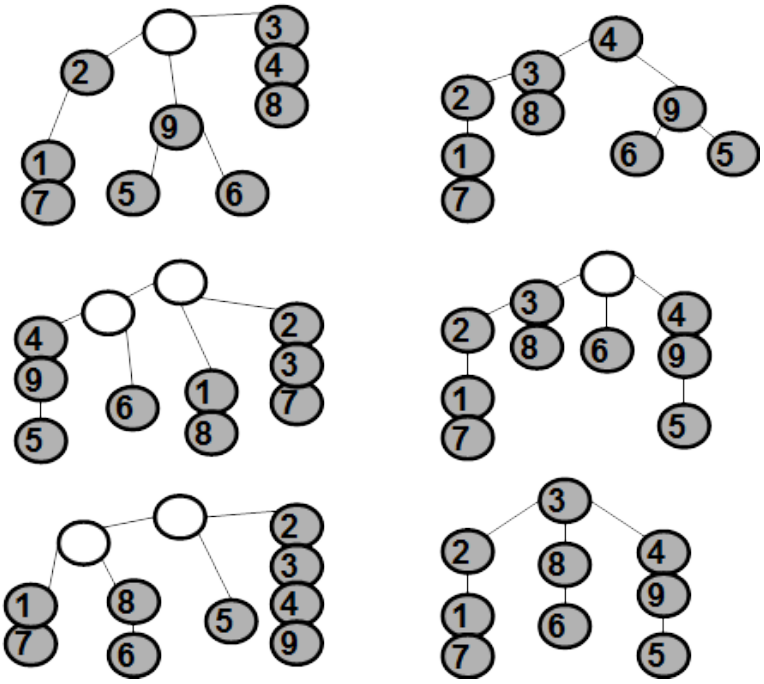


Figura 2.9: Possíveis árvores com 6 arestas âncora.

O passo seguinte é gerar todas as árvores binárias possíveis a partir das árvores âncora não resolvidas e calcular seu escore parcimônia. No exemplo, a partir das 6 possíveis árvores foram obtidas 126 árvores binárias.

Um dos aspectos interessantes desta abordagem é que, além de ser uma busca exaustiva no espaço de soluções, ela retorna várias árvores ótimas. No exemplo, foram obtidas 12 árvores ótimas de escore parcimônia igual a 15. Uma destas árvores é apresentada na Figura 2.10.

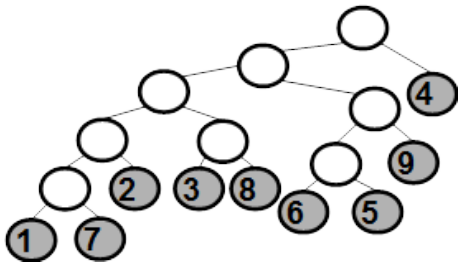


Figura 2.10: Árvore ótima obtida.

Branch-and-Bound para Parcimônia Grande

Uma outra alternativa é utilizar uma estratégia *branch-and-bound*, criada por Hendy e Penny [27] e apresentada por Felsenstein [13].

A estratégia está baseada na construção gradual da árvore através da inserção de uma espécie por vez, analisando todas as possíveis arestas em que esta pode ser ligada. Antes de explicitar a estratégia, convém realizar algumas observações sobre as árvores obtidas a partir da inserção de uma espécie por vez.

Em primeiro lugar, Hendy e Penny provam que, do ponto de vista do escore parcimônia, é possível considerar apenas árvores binárias quando tratar de filogenia. Outra propriedade importante, citada pelos autores, é a de que uma árvore não enraizada pode ser convertida em uma árvore enraizada, sem alterar o escore parcimônia, apenas pela inserção de uma raiz em qualquer aresta. A recíproca também é verdadeira. Apesar de matematicamente indiferente, Hendy ressalta o fato de que a inclusão de uma raiz pressupõe informações biológicas adicionais.

Eles também observam que, se uma árvore T tem escore parcimônia $S(T)$ então, qualquer que seja o nó v, a árvore T' obtida de T pela inserção de v em qualquer aresta é tal que $S(T') \geq S(T)$. Isto é útil caso, durante a contrução de uma possível solução, tenhamos uma árvore parcial com escore maior ou igual ao melhor escore obtido. Nesse caso a árvore parcial, e todas as possíveis obtidas a partir dela, podem ser descartadas. Finalmente, eles provam que o *branch-and-bound* gera todas as possíveis árvores não enraizadas, independentemente da ordem de inserção das espécies.

Inicia-se o *branch-and-bound* definindo uma ordem de inserção das espécies, a qual pode ser, por exemplo, a ordem de leitura.

Para exemplificar a estratégia, será utilizada a matriz M na Figura 2.11 com a distância de Hamming.

Inicia-se a resolução construindo rapidamente uma solução com um bom escore parcimônia, que será o menor escore atual. A Figura 2.12 mostra um exemplo de árvore inicial para a matriz M da Figura 2.11. Ao lado mostra uma possível rotulação mínima para esta árvore que tem escore parcimônia igual a 7.

Em cada iteração é utilizada uma rotina de controle, baseada em *backtracking*, para

	Características	
Espécies	1	2
0	A	A
1	T	T
2	C	G
3	A	C
4	G	A
5	C	T

Figura 2.11: Matriz M exemplo.

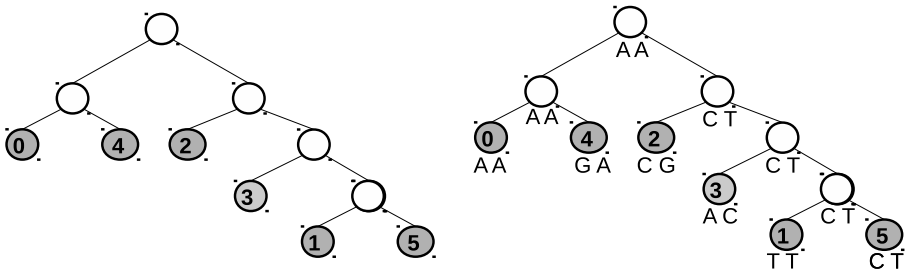


Figura 2.12: Árvore inicial e possível rotulação.

guiar a geração de uma nova solução. Esta rotina fornece uma ordem de inclusão dos nós nas arestas da árvore em construção. Apesar da rotina guiar a formação de uma árvore completa, isto não é realizado imediatamente. São realizados testes prévios, com a árvore em construção, para verificar se existe a possibilidade de que a árvore gerada ao final possua um escore parcimônia menor que o menor escore até agora encontrado. Caso a árvore parcial for passando nos testes, o processo de construção continua. Um exemplo de teste pode ser simplesmente calcular o escore parcimônia da árvore parcial, se este for maior ou igual ao menor escore, não é necessário testar as próximas árvores resultantes da inclusão dos nós, pois independentemente de quais forem será impossível diminuir o escore.

Caso a árvore parcial não passe no teste, aquela construção é abandonada e é acionada a rotina de controle de forma que gere uma nova possível solução desconsiderando todas as possíveis soluções derivadas daquela árvore parcial. Esta normalmente implica na alteração da última atribuição realizada antes do teste. Então prossegue a construção desta nova árvore, retornando-se ao estágio anterior, quando realizam-se os testes prévios.

Caso a construção continue até completar a árvore e for obtida árvore com menor escore, então este é atualizado e a árvore obtida é guardada. Feito isto o processo é

retomado solicitando-se uma possível próxima solução à rotina de controle.

A etapa anterior é realizada até esgotar todas as possíveis soluções. Ao final, obtem-se a árvore com menor escore parcimônia para a instância do problema fornecida. A Figura 2.13 ilustra uma árvore com escore parcimônia mínimo para a matriz M, a qual foi obtida pela aplicação deste *branch-and-bound*. Neste exemplo o escore é 6.

	Características	
Espécies	1	2
0	A	A
1	T	T
2	C	G
3	A	C
4	G	A
5	C	T

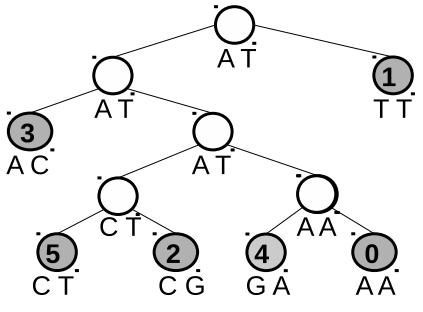


Figura 2.13: Matriz M e respectiva árvore com escore parcimônia mínimo igual a 6.

Para melhorar o desempenho do algoritmo, é utilizada a técnica *ainda-ausente*, descrita em Felsenstein [13]. Esta calcula quantos estados da característica estão presentes nas espécies que ainda faltam ser inseridas e que não estão presentes nas espécies já inseridas na árvore em construção. Estes estados ainda não apareceram, mas vão aparecer na árvore e aumentarão o escore parcimônia. Como a ordem de inclusão das espécies é fixa, este acréscimo pode ser previamente calculado e considerado no teste prévio, de forma a aumentar a quantidade de construções descartadas.

No exemplo da Figura 2.14, observe que, quando as espécies 4 e 5 faltam ser incluídas, a quantidade de estados que ainda não apareceram é 1 (o estado G para a característica 1). Então, caso já tenha sido encontrada uma árvore com escore 6, a análise da árvore apresentada na Figura 2.14 pode ser descartada, pois tem escore 5 e ainda faltam incluir as espécies 4 e 5 que acrescentarão, pelo menos, mais 1 no escore parcimônia igualando o melhor até o momento.

Algoritmos de Aproximação para Parcimônia Grande

Uma outra abordagem possível é a de algoritmos de aproximação, que fornecem uma solução em tempo polinomial e garantem que esta solução esteja distante, no

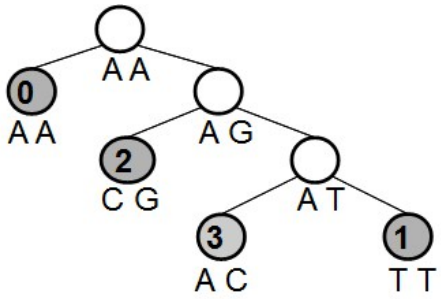


Figura 2.14: Árvore parcial T com escore 5 que será descartada pois ainda-ausente é igual a 1.

máximo, a uma razão de aproximação x da solução ótima.

Sung [51] apresenta um algoritmo de aproximação de razão 2 baseado no problema da árvore geradora mínima. Sung indica que o melhor algoritmo de aproximação obtido até aquele momento possui uma razão de 1,55 [45], a qual não é considerada uma boa aproximação para LPP na maioria dos casos pois as heurísticas obtém melhores resultados.

Neste capítulo foi apresentado o problema geral de filogenia, em particular, o caso da filogenia baseada em características. Os principais resultados e classificações deste problema foram vistos, assim como o conceito de parcimônia e os problemas relacionados com parcimônia (SPP e LPP). As soluções polinomiais clássicas para SPP foram descritas. No caso LPP foram vistas possíveis buscas exaustivas e um algoritmo *branch-and-bound* que o resolve. Entretanto, LPP é NP-completo o que, no caso geral, torna ineficiente o uso dos algoritmos apresentados. Isto motiva a pesquisa de outras alternativas que encontrem boas soluções, mas que não garantam, necessariamente, encontrar a solução ótima (a árvore mais parcimoniosa). Estas alternativas serão vistas no próximo capítulo.

Capítulo 3

Heurísticas para parcimônia na filogenia tradicional

Como dito anteriormente, LPP é NP-completo. Apesar da utilização de *branch-and-bound* para este problema resultar em resposta mais rápida, esta não elimina o caráter exponencial da busca, apenas diminui o valor do expoente ou divide o tempo total necessário. Hendy e Penny [27] afirmam que, apesar dos dados de sequências biológicas estarem longe do pior caso, é possível construir exemplo que exija percorrer todos os ramos da busca, com o descarte da solução acontecendo apenas no estágio final, requerendo $O(m.n^n)$ passos, sendo n o número de objetos e m o número de características. Desta forma, para o caso geral, a utilização de *branch-and-bound* para resolução do LPP não é aceitável.

Na prática, uma alternativa bastante utilizada é o desenvolvimento e utilização de heurísticas que acelerem o processo de busca de uma boa solução, mesmo sem garantir que a solução obtida seja ótima. O estudo de heurísticas tradicionais é importante para nosso trabalho, pois pode servir de inspiração para preparação de heurísticas que ajudem a encontrar boas soluções do problema da filogenia viva estudado nesta tese.

Heurísticas podem ser aplicadas em diversas situações associadas ao problema. Tomando por base Felsenstein [13], as heurísticas foram classificadas em: Heurísticas de Construção (Seção 3.1), Heurísticas de Rearranjo (Seção 3.2), Métodos de Dividir para Conquistar (Seção 3.3) e Reponderamento de Características (Seção 3.4). Os termos Heurísticas de Construção e Métodos de Dividir para Conquistar não são

explicitamente definidos por Felsenstein [13], mas as heurísticas que pertencem a estas classes são apresentadas de forma agrupada. Dividir para Conquistar é uma classe sugerida por Giribet [17] e o termo Heurísticas de Construção é proposto por Andreatta e Ribeiro [1].

3.1 Heurísticas de Construção

A hipótese aqui é que pode-se obter bons resultados iniciando com árvores bem construídas. As heurísticas a seguir buscam construir filogenias com um bom escore parcimônia.

Adição Sequencial

Felsenstein [13] define esta estratégia, em que o processo é semelhante ao *branch-and-bound*, no sentido de que, em cada etapa, adiciona-se uma espécie x na árvore T em construção, de forma gulosa. O escore parcimônia de todas as possíveis inclusões de x em T é calculado e a árvore parcial com menor escore é mantida para o próximo passo. A ordem de inclusão das espécies é aleatória, sendo que ordens diferentes podem resultar em árvores diferentes.

A Figura 3.1 ilustra a construção de uma árvore utilizando Adição Sequencial para a matriz exemplo M (Figura 2.11 pg 26), considerando como ordem de inclusão a ordem de leitura (0, 1, 2, 3, 4, 5).

Também é possível combinar esta estratégia com heurísticas de rearranjo (Seção 3.2), obtendo um método de busca de melhor desempenho. Neste caso, a inserção de espécies é interrompida, passando para uma etapa onde são testados rearranjos desta árvore intermediária em busca de um melhor escore. Após encontrar uma árvore melhor ou esgotar um limite de tentativas, retorna-se à etapa de inserção de espécies.

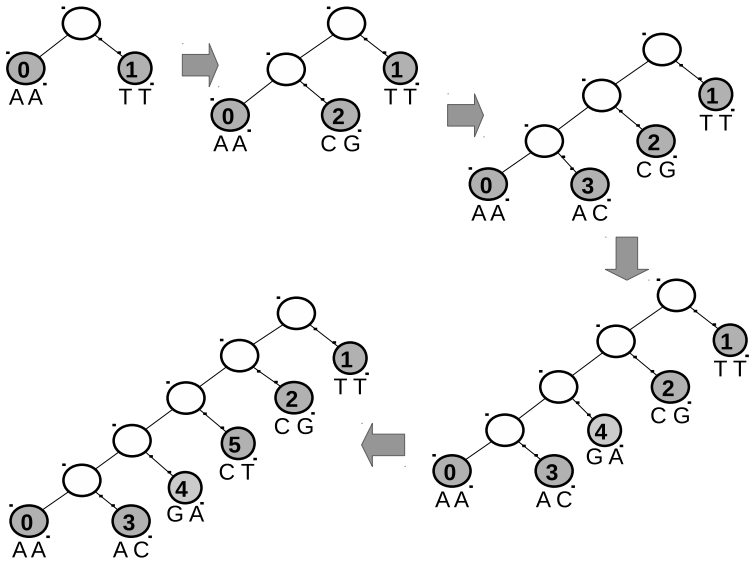


Figura 3.1: Árvore obtida por Adição Sequencial.

Decomposição Estrela

Felsenstein [13] define esta estratégia. Inicia com todas as espécies presentes, em uma árvore multifurcada com a raiz ligada a todas as espécies. Em cada passo, dois ramos são agrupadas em um novo nó que é ligado à raiz. Existem muitas maneiras de decompor uma estrela. Observe que duas vias diferentes podem produzir a mesma árvore.

A Figura 3.2 ilustra a construção de uma árvore utilizando Decomposição Estrela para a matriz exemplo M (Figura 2.11 pg 26). Neste exemplo os nós são agrupados com base na distância de Hamming entre eles. Em havendo empate, o par unido é escolhido aleatoriamente.

A seguinte estratégia pode ser usada para decompor uma estrela.

Neighbor-joining - NJ

Proposto por Saitou e Nei [47] NJ (Neighbor-Joining) é um algoritmo que trabalha agrupando os objetos vizinhos com menor distância. O princípio do método NJ pode ser estendido para filogenia baseada em características usando a definição de distância dada pela distância de Hamming total, que é o número de posições com valores diferentes nos dois objetos.

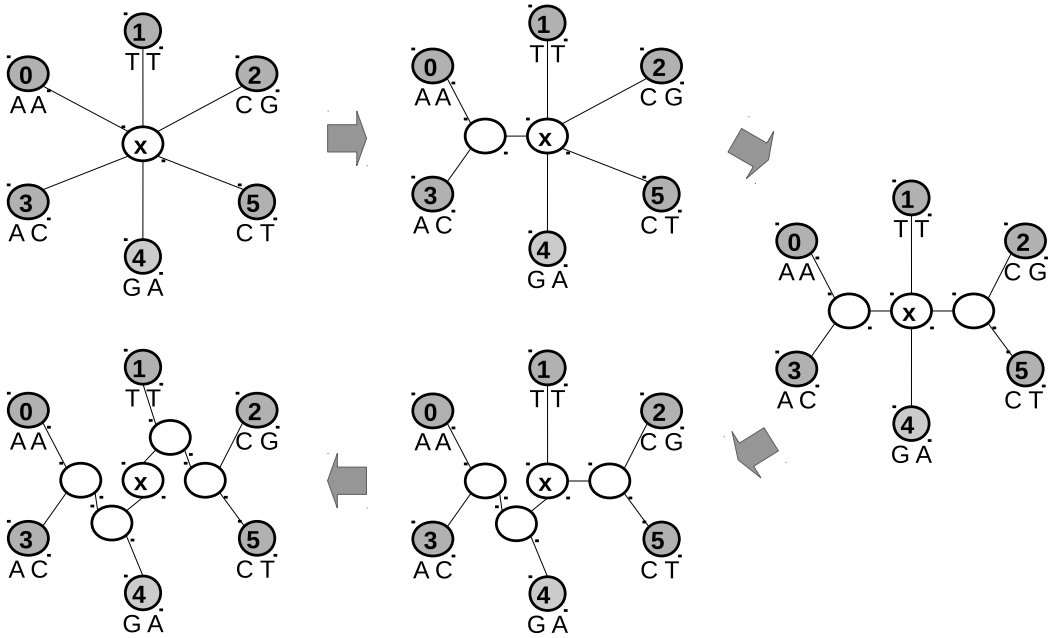
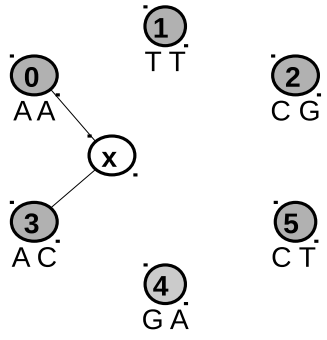


Figura 3.2: Árvore obtida por Decomposição Estrela.

Para exemplificar a construção de uma árvore utilizando NJ será utilizada a matriz M (Figura 2.11 pg 26). A Figura 3.3a ilustra a tabela de distâncias Hamming entre as espécies. Os pares de nós 0 e 3; e 0 e 4 são os mais próximos. De forma aleatória foi escolhido o par de nós 0 e 3 para serem agrupados em um nó x apresentado na Figura 3.3b.

Distância	Espécies					
Espécies	0	1	2	3	4	5
0	0	2	2	1	1	2
1	2	0	2	2	2	1
2	2	2	0	2	2	1
3	1	2	2	0	2	2
4	1	2	2	2	0	2
5	2	1	1	2	2	0

(a)



(b)

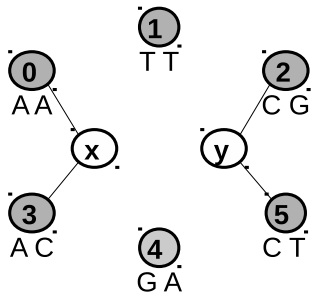
Figura 3.3: Distâncias Hamming entre espécies e nó x agrupando 0 e 3.

Em seguida, calcula-se a distância de cada nó restante w ao nó x, como sendo a média entre as distâncias deste nó w aos nós 0 e 3, que são retirados da matriz. A tabela de distâncias Hamming recalculada é apresentada na Figura 3.4a. Os pares de nós 1 e 5;e 2 e 5 são os mais próximos. De forma aleatória foi escolhido o par de

nós 2 e 5 para serem agrupados no nó y apresentado na Figura 3.4b. A Figura 3.5 apresenta a sequência dos nós criados até o final do algoritmo.

Distância	Espécies				
Espécies	x	1	2	4	5
x=03	0	2	2	1,5	2
1	2	0	2	2	1
2	2	2	0	2	1
4	1,5	2	2	0	2
5	2	1	1	2	0

(a)



(b)

Figura 3.4: Distâncias Hamming entre espécies e nó y agrupando 2 e 5.

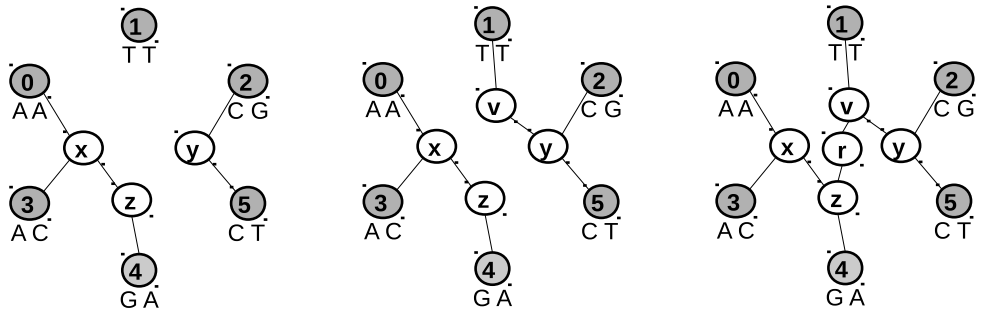


Figura 3.5: Conclusão do NJ.

Richer e colegas [44] propõem outras pequenas variações para NJ. A primeira utilizando como distância o escore parcimônia, que deve ser recalculado a cada etapa.

A segunda realizando um misto de NJ com Adição Sequencial, separando metade das espécies para realização de NJ e a outra metade para Adição Sequencial. Constrói a árvore usando NJ para a metade das espécies e depois vai inserindo as demais espécies com Adição Sequencial.

Colônia de Formigas

Lopes e Perretto [36] propuseram a utilização da heurística Colônia de Formigas para criação da árvore filogenética. Esta heurística inicia com um grafo completo contendo todas as espécies e um conjunto de agentes chamados *formigas*, que irão percorrer o grafo. A cada aresta é atribuída uma distância, que é a distância de Hamming entre os nós.

Em cada etapa, uma quantidade fixa de tempo ou iterações é realizada, em que as formigas percorrem uma distância fixa depositando um valor numérico (feromônio) nas arestas percorridas (Figura 3.6a e Figura 3.6d). A escolha de cada formiga sobre qual aresta percorrer é aleatória. O feromônio também é diminuído (evaporação) com o passar do tempo ou iterações.

Ao final da etapa, os vértices da aresta com o maior valor de feromônio são selecionados para agrupamento. Supondo que, após uma quantidade fixa de iterações a quantidade de feromônio restante em cada aresta do grafo da Figura 3.6(a) seja ilustrado na Figura 3.6b. A aresta em negrito possui o valor mais alto, indicando que foi mais utilizada e cujos vértices devam ser agrupados. O valor de feromônio desta aresta é zerado e é criado um nó intermediário entre os dois vértices. Este nó intermediário é ligado a todos os demais e o valor de feromônio destas arestas é calculado levando-se em conta o valor de feromônio entre estes vértices e os que foram agrupados (Figura 3.6c). Em seguida os nós agrupados são desconectados dos demais nós e é retomada a caminhada das formigas (Figura 3.6d).

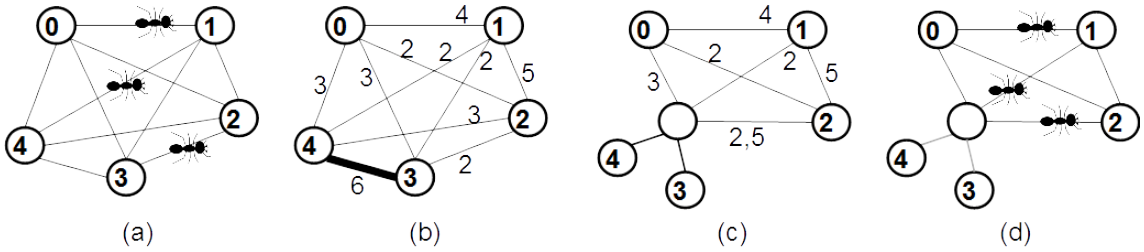


Figura 3.6: Heurística Colônia de Formigas: (a) formigas percorrem o grafo depositando feromônio; (b) feromônio restante em cada aresta após etapa; (c) aresta (4,3) é eliminada e criado nó intermediário; (d) formigas percorrem grafo sem visitar arestas adjacentes aos nós 3 e 4.

3.2 Heurísticas de Rearranjo

Frequentemente chamados de métodos de busca em vizinhança, estes métodos levam em conta uma árvore já construída em relação à qual deseja-se melhorar o escore através de pequenas alterações.

Em cada um destes métodos, é possível guardar mais de uma árvore com melhor escore, melhorando a busca.

Intercâmbio de vizinho mais próximo - NNI

Felsenstein [13] apresenta esta estratégia, chamada *NNI* (*Nearest Neighbor Interchange*). A heurística tem como entrada uma árvore. Em cada etapa é selecionada uma aresta interior, que possua duas arestas adjacentes em cada uma de suas extremidades. Estas cinco arestas são apagadas, resultando em quatro subárvores desconexas. Então são reconectadas nas duas maneiras possíveis diferentes da árvore original e calculados seus escores. A Figura 3.7 exemplifica a aplicação de NNI sobre a aresta (X, Y).

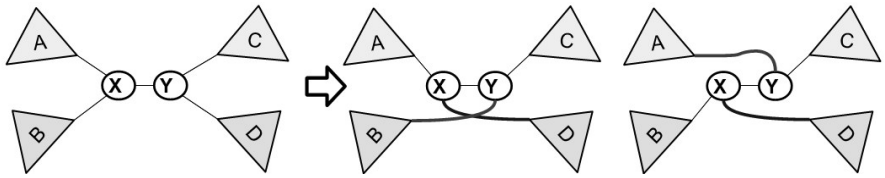


Figura 3.7: NNI aplicado na aresta (X, Y) gerando árvores: com a subárvore A adjacente à subárvore D e com a subárvore A adjacente à subárvore C.

A operação é repetida para todas as arestas. Ao final desta etapa, continua-se com a árvore obtida de menor escore. Uma alternativa gulosa pode continuar o processo com a árvore obtida a partir da primeira melhoria. Observe que, para uma árvore binária enraizada com n folhas, existem $n - 2$ arestas internas e $2(n - 2)$ possíveis vizinhos a considerar. Esta operação pode ser repetida um número limitado de vezes ou enquanto proporcionar melhoria no escore.

Poda e enxerto de subárvore - SPR

Esta estratégia *SPR* (*Subtree Pruning and Regrafting*) [13] consiste no corte de uma aresta qualquer da árvore. Este corte gera uma subárvore e a eliminação do nó interno ligado à aresta eliminada. Em seguida a subárvore é reconectada na árvore em alguma outra aresta, em que é inserido um nó interno para realizar esta conexão. A Figura 3.8 mostra um exemplo de possível árvore gerada por SPR.

Felsenstein [13] mostra que o número de árvores possíveis de serem obtidas a partir de uma árvore de n folhas é na ordem de n^2. Assim, SPR possui muito mais possibilidades de alterações do que NNI, resultando numa busca muito mais ampla e com maior chance de encontrar melhor resultado.

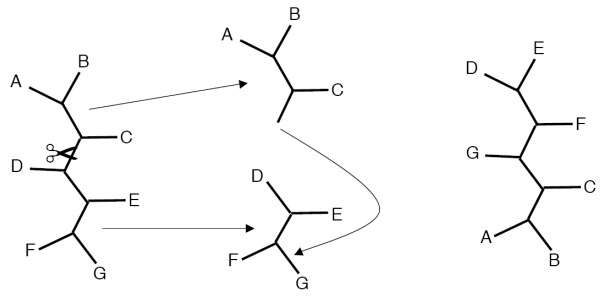


Figura 3.8: SPR gerando nova árvore.

Bisseção e reconexão de árvore - TBR

A estratégia *TBR* (*Tree Bisection and Reconnection*) [13] é ainda mais elaborada que SPR. Da mesma forma é eliminada uma aresta. Entretanto, desta vez é permitida a reconexão de qualquer aresta da subárvore em qualquer aresta da árvore. A Figura 3.9 apresenta a aplicação de TBR sobre a mesma aresta da mesma árvore inicial apresentada na Figura 3.8. Observe que o TBR apresenta mais possibilidades de árvores geradas do que o SPR, entretanto não existe uma fórmula geral para esta quantidade pois depende da forma da árvore e do corte.

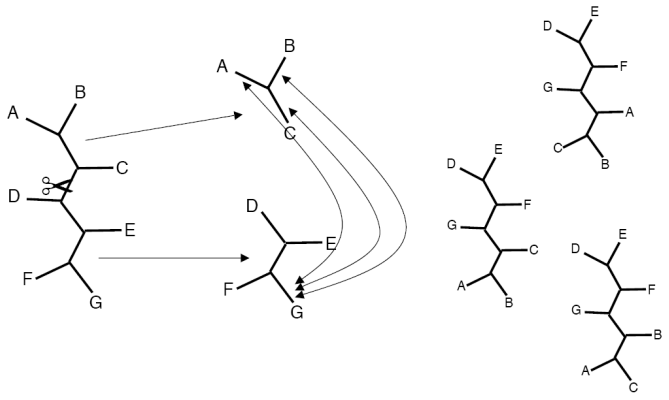


Figura 3.9: TBR gerando 3 novas árvores.

Algoritmos Genéticos

Algoritmo Genético é uma técnica criada por Holland [30] amplamente utilizada na resolução de problemas de otimização combinatória, que se caracteriza pela manipulação de uma população de soluções que são misturadas e alteradas aleatoriamente gerando outras possíveis soluções. Transformar um problema de otimização em um Algoritmo Genético envolve a definição dos seguintes mapeamentos: genótipo, que

é a codificação de uma possível solução; função de *fitness*, que mede a qualidade de uma possível solução; operadores de crossover e mutação, que misturam e alteram as soluções gerando outras; e a escolha de parâmetros diversos, entre eles a taxa de mutação, tamanho da população e taxa de mortalidade da população.

No contexto de filogenias, o genótipo de um algoritmo genético descreve uma árvore e a função de *fitness* reflete a otimalidade desta árvore, na maioria das vezes o escore parcimônia. Conforme Felsenstein [13],

> *Matsuda parece ter sido o primeiro a usar um algoritmo genético em filogenias. Ele otimizou os comprimentos de ramos em cada árvore e usou um operador de recombinação que permutava subárvores particularmente boas entre as árvores.*

Matsuda [40] utilizou a abordagem *Maximum Likelihood* e definiu *crossover* a partir de duas árvores. Em cada uma delas calcula as matrizes de distância entre as folhas. Para cada par de folhas, calcula a diferença relativa entre as duas distâncias e identifica as folhas com maior diferença, ou seja, o par em que houve maior discrepância entre as distâncias geradas a partir das duas árvores. Toma por referência a árvore em que a distância do par foi menor, preservando a menor subárvore que contenha o par. Então escolhe aleatoriamente outra folha v e elimina todas as demais folhas na árvore referência. Na outra árvore, elimina as folhas que restaram na árvore referência, com exceção de v. Então une as duas árvores resultantes sobrepondo v.

O operador de mutação é feito através da troca de ramos.

Lewis [35] também usou *Maximum Likelihood* e uma operação de *crossover* semelhante a SPR. As árvores são recombinadas pela escolha de uma subárvore em uma árvore T_1 (por exemplo a árvore da Figura 3.10a, em que a parte tracejada corresponde à subárvore escolhida), excluindo as espécies desta subárvore em uma outra árvore T_2 (Figura 3.10b) e inserindo a subárvore nesta (Figura 3.10c). A Figura 3.10 mostra um exemplo da operação de *crossover*.

Katoh e colegas [33] usaram mutações como rearranjos TBR e *crossover* formado por permutações de subárvores que continham o mesmo conjunto de espécies.

Hill e colegas [28] utilizaram mutação conforme Matsuda [40] e *crossover* conforme Lewis [35] e realizaram diversos testes alterando parâmetros de algoritmos genéticos,

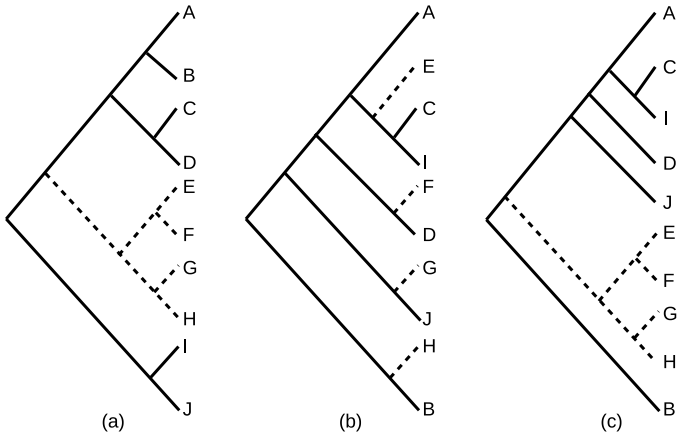


Figura 3.10: *Crossover* entre as árvores em (a) e (b), gerando a árvore em (c).

como tamanho da população e taxa de mutação e crossover, verificando sua influência no tempo consumido e qualidade dos resultados.

Fusão de árvores (*tree-fusing*)

Fusão de árvores foi proposta por Goloboff [20] e melhorada pelo próprio Goloboff [21] em uma versão posterior com a inclusão do comando *hybrid*. De acordo com o próprio autor, a ideia básica é trocar sub-grupos entre duas árvores diferentes. Este método, em sua versão inicial, requer duas árvores que contenham uma subárvore com a mesma lista de espécies. Então outras duas árvores serão criadas pela permuta de subárvores.

Na versão com o comando *hybrid* é permitido que as subárvores não sejam idênticas. Ele toma as duas árvores de entrada e identifica uma partição induzida por aresta em cada árvore. Estas partições devem ser tão similares quanto possível.

As espécies não compartilhadas pelas subárvores são removidas, as duas metades das árvores são trocadas e, então, as árvores são completadas pela reinclusão das espécies faltantes pela criação de uma subárvore mediante uma sequência de adição aleatória e reconexão desta subárvore usando TBR. A Figura 3.11 mostra o início da operação *hybrid* para duas árvores com partição induzida pela aresta adjacente à raiz das árvores, com árvores geradas pela retirada das espécies não compartilhadas (nós cinza) e montagem de subárvore com estas espécies. A Figura 3.12 mostra a troca entre as metades e reinclusão das espécies via TBR.

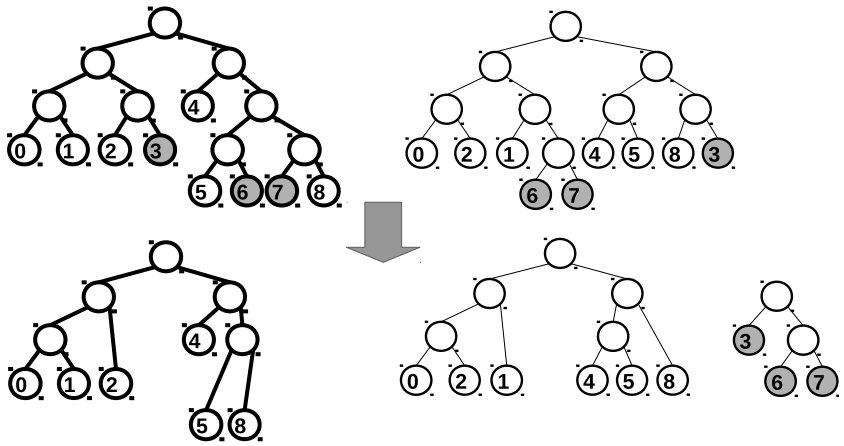


Figura 3.11: Remoção de espécies não-compartilhadas.

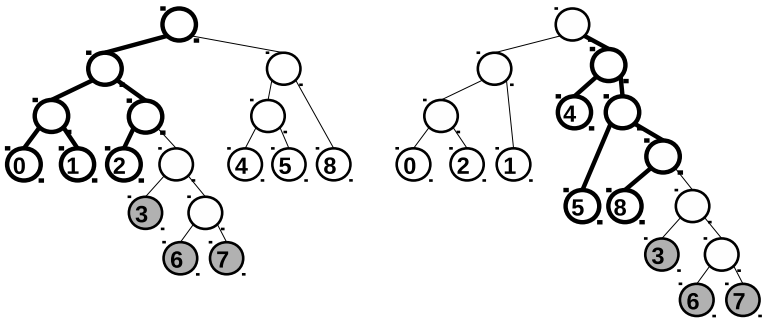


Figura 3.12: Permuta de subárvores e reinclusão de espécies não-compartilhadas.

Giribet [17] classificou fusão de árvores como um algoritmo genético e afirma que, até o momento de sua publicação, era a implementação mais comum e eficiente de algoritmos genéticos para LPP.

Simulated annealing

Conforme Felsenstein [13], *simulated annealing* aceita uma nova solução se ela é melhor, e algumas vezes sendo ela pior. A definição do quão pior esta solução pode ser é dada pelo esquema de resfriamento adotado, que inicia com uma permissividade inicial e vai diminuindo com o passar das iterações até chegar a zero. Isto resulta na perambulação entre as soluções que estão tendendo às melhores.

Goloboff [20] implementa *simulated annealing* em seu método *tree-drifting*. Este método realiza TBR para obter nova solução, que será avaliada usando uma medida *Diferença de Adequação Relativa*, que enfatiza pequenas diferenças no escore parcimônia entre a nova solução e a anterior. O método alterna etapas em que so-

mente soluções que melhorem o escore são aceitas e etapas em que soluções que não melhorem o escore, mas que tenham a Diferença de Adequação Relativa pequena, também sejam aceitas.

Richer e colegas [44] implementam *simulated annealing* iniciando por uma solução obtida por Adição Sequencial e gerando novas soluções através da aplicação de TBR e SPR aleatoriamente. A cada 25 iterações, realizam uma operação de descida inspirada na descrita por Lu e colegas [37]. Esta operação consiste na busca sistemática pela melhor solução em toda a vizinhança da solução atual. O número de iterações realizadas até disparar a operação de descida é um parâmetro que pode ser ajustado. O valor 25 foi o de melhor desempenho para os *benchmarks* utilizados por Richer e colegas. O resfriamento é feito através da atribuição de uma temperatura inicial e decréscimo de 1% a cada etapa. Entretanto, se não for encontrada melhoria durante 50 iterações é realizado um reaquecimento de 40%. Este reaquecimento é aplicado um número limitado de vezes, por exemplo 3. O algoritmo para se, após um determinado número de resfriamentos w, não é encontrada solução melhor que a atual. Richer e colegas usaram $w = 40$.

3.3 Métodos de Dividir para Conquistar

Giribet [17] cita três métodos na família de Dividir para Conquistar: Busca Setorial, técnica de quartetos e Métodos de Cobertura de Disco. Os Métodos de Cobertura de Disco generalizam a técnica de quartetos [13].

Estes métodos aplicam outras heurísticas (geralmente heurísticas de rearranjo) em frações dos dados de entrada ou da solução em construção para obter uma nova solução.

Busca Setorial

Busca setorial, também chamada janela de árvore, foi explorada por Goloboff [20] e Sankoff e colegas [49]. Ela toma um nó interno da árvore e um conjunto de outros nós conectados a ele, não necessariamente a subárvore deste nó interno. Estes nós definem uma região chamada setor ou uma janela. O método rearranja a árvore localmente neste setor.

Este método generaliza o método NNI e pode escapar de ótimos locais.

Este método pode ser usado em uma abordagem dividir para conquistar [19], dividindo a árvore em setores e criando conjuntos de dados reduzidos baseados neles. Cada setor é analisado separadamente e uma nova árvore do setor é reintroduzida na árvore original.

Métodos de cobertura de disco - DCM

Warnow [2, 31, 54] é um dos autores dos métodos *DCM* (*Disk-Covering Methods*) e afirma que um DCM é projetado para melhorar o desempenho de um dado método base, por exemplo uma outra heurística.

DCM é uma meta-técnica para análise filogenética possuindo diversas versões. Basicamente executa os seguintes passos: primeiro decompõe o conjunto de objetos em conjuntos sobrepostos; constrói árvores para cada subconjunto usando o método base; funde as árvores em conjunto, usando um método (por exemplo, *Strict Consensus Merger* [31] ou *MRP* [41]). Se a árvore gerada não for binária, é transformada em binária otimizando o escore parcimônia.

Como primeiro passo, Warnow sugere construir um grafo triangulado cujo conjunto de vértices corresponde ao conjunto de espécies. Este grafo é chamado grafo limiar e é contruído tomando por base um limiar q e a matriz de distâncias entre as espécies $[D_{ij}]$. Cada espécie é representada por um nó e dois nós v_i e v_j são ligados por aresta se e somente se $D_{ij} \leq q$. Se $[D_{ij}]$ for aditiva então o grafo gerado é triangulado, caso contrário são incluídas arestas, minimizando o custo das arestas incluídas, para tornar o grafo triangulado[1].

Grafos triangulados permitem enumeração dos cliques maximais em tempo polinomial. Cada clique maximal define um subconjunto de espécies. O conjunto de cliques gera vários subconjuntos sobrepostos. Em seguida gera uma árvore para cada subconjunto, utilizando o método base. Então realiza a mescla das árvores utilizando o método *Strict Consensus Merger* [31].

[1] G é um grafo triangulado (ou cordal) se todo circuito de G de comprimento maior que 3 tem uma corda.

Dado um caminho P em um grafo, uma corda em P é uma aresta cujos extremos são dois vértices não consecutivos de P.

Observe que o valor q é um parâmetro importante. Se for muito pequeno gerará grafo com cliques muito pequenas e não conseguirá gerar a árvore com todas as espécies. Por outro lado, se for muito grande gerará grafo completo e o método se reduzirá à aplicação do método base na obtenção da árvore. Desta forma, sugere-se que valores q sejam tomados dentre os valores de $[D_{ij}]$.

Existem diversas versões de DCM's, entre elas Rec-I-DCM3 (DCM3 recursivo e iterativo). Seu procedimento é baseado na aplicação recursiva e iterativa de decomposições DCM3, selecionando subconjuntos de táxons terminais baseados em uma dada topologia de árvore (a *árvore guia*). Cada um destes subconjuntos de táxons terminais é usado para criar conjuntos de dados reduzidos (subproblemas); se a primeira subdivisão do conjunto de dados produz subproblemas muito grandes, o método é recursivamente aplicado até que eles sejam menores que um tamanho máximo (especificado pelo usuário).

Goloboff e Pol [19] afirmam que Rec-I-DCM3 é uma das mais bem sucedidas versões de DCM's. Warnow [53] sugere ainda a utilização de heurísticas diferentes em cada iteração do método, na busca de uma árvore local ótima, a qual é utilizada para gerar os conjuntos sobrepostos.

3.4 Reponderamento de Características

Da mesma forma que os métodos de dividir para conquistar (Seção 3.3) estes métodos aplicam outras heurísticas (geralmente heurísticas de rearranjo) para obter uma nova solução.

Estes métodos iniciam com uma árvore e alteram, temporariamente, a importância de algumas características, na maioria das vezes duplicando a ocorrência delas na matriz de estados. Então aplicam heurísticas obtendo árvores diferentes e, finalmente, retornam à matriz de estados original e verificam se a árvore obtida tem menor escore parcimônia.

Ratchet

Ratchet foi proposto por Nixon [42] e seleciona um conjunto de 5% a 25% das

características para enfatizar através do aumento dos pesos das características. Este aumento pode ser feito pela duplicação de cada uma destas características de forma que apareçam duas ou mais vezes no conjunto de dados.

Em seguida utiliza uma heurística de rearranjo, como NNI ou TBR, no conjunto de dados modificados, guardando uma ou mais árvores. No próximo passo, retorna todos os pesos aos valores originais e executa novamente o método de rearranjo, guardando uma ou mais árvores. Este processo é repetido muitas vezes.

Goloboff [21] classifica Ratchet, tree-drifting (Seção 3.2) e Rec-I-DCM3 (Seção 3.3) como esquemas de perturbação cíclica, pois alternam etapas de busca local com etapas de perturbação na solução sendo construída.

Bootstrap

Bootstrap [13] e *Jackknife* são métodos estatísticos para estimar, empiricamente, a variabilidade de uma estimativa. Eles são diferentes mas pertencem à mesma familia de métodos. Jackknife realiza uma observação por vez de uma amostra, e calcula a estimativa. A variabilidade da estimativa é inferida pelas variações que ela causa, por extrapolação.

Bootstrap considera características como amostras independentes. A hipótese geral de filogenia, de que características evoluem independentemente, satisfaz a independência requerida pelo Bootstrap. Considere m o número de características. Bootstrap gera m novas matrizes de estados, com o mesmo número de características e objetos que a original. A geração duplica aleatoriamente algumas características e elimina outras.

Então usa um método de filogenia para construir árvores para todas as matrizes de estados.

Após isto, têm-se um conjunto de árvores. Pode ser atribuída uma probabilidade P para o ramo se uma fração P das árvores tem este ramo presente. Mas isto se torna complexo se m é grande e houver muitos ramos a observar. Então é usado um método de árvore consenso, por exemplo *Majority-rule consensus* [13]. Este método conta o número de vezes que cada partição de espécies é encontrada no conjunto de árvores. Então outra árvore é construída tomando o maior número para construir uma aresta

entre as partições. O processo continua em cada subconjunto até completar a árvore.

Jackknife

Jackknife difere de Bootstrap apenas na geração de novas amostras. Diferente de Bootstrap, Jackknife gera $m/2$ novas amostras, cada uma delas é gerada pela retirada de uma característica selecionada aleatoriamente.

De acordo com Felsenstein [13] não fica em que condições Jackknife apresenta quaisquer vantagens substanciais sobre Bootstrap.

Reponderamento não aleatório de características

Quicke e colegas [43] propõem que o reponderamento das características não seja aleatório. Mas que as características sejam reponderadas proporcionamente à sua qualidade de ajuste nas árvores previamente encontradas.

Um possível indicador da qualidade do ajuste é *CI* (*character consistency index*). CI [29] de uma característica C em uma árvore T é definido como a razão entre a quantidade de variações em C e o escore parcimônia calculado apenas sobre a característica C na árvore T. Por exemplo, se para uma dada característica C, existem 4 valores diferentes na matriz de estados e o escore parcimônia na árvore T for 8 então $CI = 0,5$. Se o escore parcimônia fosse 5 então $CI = 0,8$. Quanto maior este valor, mais próxima a árvore T está de uma Filogenia Perfeita com relação à característica C.

Reponderamento não aleatório, em cada iteração, realiza as seguintes tarefas: avalia as características que mais contribuíram para a melhor árvore até o momento (por exemplo usando a característica com maior CI); repondera as características de acordo com esta avaliação; aplica novamente a heurística de rearranjo para obter nova árvore, que é ótima local considerando o reponderamento; então retira este reponderamento; aplica uma heurística de rearranjo (por exemplo TBR) para obter árvore ótima local.

3.5 Comentários

Neste capítulo apresentamos as principais heurísticas para LPP. A Tabela 3.1 sintetiza as heurísticas vistas e suas principais características.

As heurísticas apresentadas aqui para o problema da parcimônia grande na filogenia tradicional podem servir de base à proposta de uma heurística para solução dos problemas tratados neste trabalho.

A adaptação das heurísticas ao problema da filogenia viva demanda análise criteriosa, buscando obter o máximo de melhoria no desempenho por conta das novas possibilidades ou informações acrescentadas com a existência de nós internos vivos. Devido a isso, serão priorizadas algumas heurísticas.

A maior parte das heurísticas, em particular as de rearranjo, demandam uma árvore com todas as espécies para ser manipulada na busca por outra árvore com melhor escore parcimônia. Por causa da necessidade de construção de uma solução inicial e do comportamento guloso, foi selecionada a Adição Sequencial. Como as heurísticas de rearranjo implementam os conceitos de busca local e servem de base para diversas outras estratégias, também foram selecionadas duas heurísticas: NNI e TBR. Conforme Felsenstein [13] NNI possui uma vizinhança muito menor do que a vizinhança de TBR. Portanto, NNI realiza busca local enquanto que TBR pode ser utilizada para fugir de ótimos locais.

Outras heurísticas que não foram usadas nesta tese podem indicar caminhos interessantes de investigação de soluções, em trabalhos futuros sobre filogenia viva.

Tabela 3.1: Quadro com as heurísticas para parcimônia em filogenia tradicional.

	Heurística	Características
Construção	Adição Sequencial	Constrói árvore incluindo uma espécie por vez na aresta que gera árvore temporária de menor escore.
	Decomposição Estrela	Inicia com árvore multifurcada contendo todas as espécies e vai agrupando ramos de 2 em 2 até gerar árvore binária.
	Neighbor-joining (NJ)	Inicia com todas espécies representadas por nós desconectados; segue conectando os dois mais próximos.
	Colônia de Formigas	Inicia com grafo completo ponderado e quantidade fixa de agentes (formigas) que percorrem o grafo de forma aleatória, distribuindo feromônio.
Rearranjo	Intercâmbio de vizinho mais próximo (NNI)	Toma uma aresta e faz a permuta dos vértices das arestas adjacentes.
	Poda e enxerto de subárvore (SPR)	Toma uma aresta, desconecta da árvore e reconecta em outra aresta da árvore.
	Bisseção e reconexão de árvore (TBR)	Toma aresta e a elimina formando uma subárvore; então permite a reconexão de uma aresta qualquer desta subárvore em outra da árvore.
	Algoritmos Genéticos	Mantém um conjunto de árvores, sobre as quais aplica operações de recombinação e mutação, gerando novas árvores que, conforme o escore, são incluídas no conjunto ou descartadas.
	Fusão de árvores	Toma duas árvores e identifica uma partição induzida por aresta em cada árvore; em seguida retira as espécies não compartilhadas entre as partições; então faz permuta de ramos entre as árvores e a reinclusão das espécies não compartilhadas.
	Simulated Annealing	Utiliza outra heurística de rearranjo para gerar árvores, alternando etapas onde somente soluções melhores são aceitas e etapas onde soluções que não melhorem o escore também são.
Dividir para conquistar	Busca Setorial	Recorta uma janela na árvore, rearranja esta janela e reconecta as extremidades desta janela.
	Métodos de cobertura de disco (DCM)	Decompõe o conjunto de espécies em subconjuntos sobrepostos; constrói árvores para cada subconjunto; funde as árvores.
Reponderamento de características	Ratchet	Repondera 5% a 25% das características e utiliza heurística de rearranjo para gerar nova árvore.
	Bootstrap/Jackknife	Geram várias novas matrizes de estado, duplicando ou eliminando características; para cada matriz gera uma árvore; usa método de árvore de consenso para gerar nova árvore.
	Reponderamento não aleatório	Avalia as características que mais contribuíram para a melhor árvore; repondera as características, considerando esta informação; aplica heurística de rearranjo retira o reponderamento; aplica novamente a heurística.

Capítulo 4

O problema da parcimônia pequeno para filogenia viva

Na Seção 2.1, tratamos do problema da parcimônia pequeno para filogenia tradicional. Dois algoritmos, Fitch e Sankoff, foram descritos, seguindo o roteiro proposto em [32]. Neste capítulo, tratamos do problema da parcimônia pequeno para filogenia viva, apresentando as respectivas adaptações para os mesmos algoritmos.

Adaptando a Definição 2.1, têm-se a definição abaixo, em que $S(T)$ é dado pela Equação 2.1 (pg. 10).

Definição 4.1. *Problema da Parcimônia Pequeno Viva (SLPP)*
Entrada: Uma árvore T em que as $n - l$ folhas e os $l \geq 0$ nós internos vivos são rotulados por uma cadeia de m estados da característica.
Saída: Uma rotulação dos vértices internos (não vivos) de T preservando a rotulação dos nós internos vivos e minimizando o escore parcimônia $S(T)$.

Para este problema, foram construídas adaptações dos algoritmos Fitch e Sankoff descritos na Seção 2.1 para SLPP. Essas adaptações foram apresentadas em Güths e colegas [24] e são descritas nas Seções 4.1 e 4.2.

4.1 Algoritmo Fitch Live

Em sua descrição original, não se pode utilizar o algoritmo Fitch no caso de filogenia viva pois Fitch não permite nós internos vivos. Desta forma, precisamos adaptar o algoritmo para nós internos vivos. Observe que o nó interno vivo é similar a uma folha pois já possui rotulação e ela é fixa. Seja v um nó interno vivo, como sua rotulação é fixa, definimos $possivel(v)$ da mesma forma como nas folhas.

O Algoritmo 4.1 apresenta o algoritmo Fitch Live, que usa a função $rand : \Sigma^* \to \Sigma$ que seleciona aleatoriamente um estado da característica em um dado conjunto.

As alterações foram feitas nas Partes I e II do algoritmo. Na Parte I a definição do conjunto $possivel(v)$ como unitário contendo apenas a rotulação é feita não apenas para as folhas, mas também para os nós internos vivos. Isto é coerente com o conceito de nó interno vivo, pois a rotulação deste nó já está definida e não pode mudar. Consequentemente, na Parte II, o cálculo do conjunto $possivel(v)$ só acontece para os nós internos comuns. A Parte III permanece igual, pois trata apenas de escolher um rótulo dentre os possíveis. Como o conjunto de possíveis no caso dos nós internos vivos é unitário contendo apenas a rotulação já definida, esta escolha respeita a definição de nó interno vivo.

4.2 Algoritmo Sankoff Live

Da mesma forma que o algoritmo Fitch, o algoritmo Sankoff também não permite nós internos vivos. Então, propõe-se uma mudança no algoritmo para manipular nós internos vivos. O Algoritmo 4.2 apresenta o algoritmo Sankoff Live.

O algoritmo inicia como antes, definindo escores parcimônia nas folhas, 0 para o estado da característica rotulado e ∞ para os demais. Nesta etapa, ainda que temporariamente, faz a mesma atribuição para os nós internos vivos.

No passo seguinte, calcula os escores parcimônia para cada nó interno e cada possível estado da característica. Nos nós internos não vivos, o cálculo permanece o mesmo. A mudança ocorre nos nós internos vivos. É preciso considerar que o nó, assim como as folhas, já possui rotulação, ou seja, o estado da característica já está definido e é impossível rotular o mesmo com outro estado. Suponha, por exemplo, que v

Algoritmo 4.1 Fitch Live.

Entrada: (1) Uma árvore $T = (V, E)$ com $root \in V$ sendo a raiz de T, (2) um alfabeto de k letras Σ e (3) uma função $label : V \to \Sigma \cup \{\lambda\}$ rotulando todas as folhas e nós internos vivos por elementos de Σ.

Saída: (1) Uma função $label' : V \to \Sigma$ minimizando o escore parcimônia.

Parte I: Define *possivel* para folhas e nós internos vivos;
for $\forall v \in V$ **do**
  if $label(v) \neq \lambda$ **then**
    $possivel(v) = \{label(v)\}$
  end if
end for
Parte II: Percorre T em pós-ordem definindo *possivel* para os demais nós;
for $\forall v \in V$ nó interno **do**
  if $label(v) = \lambda$ **then**
    if v tem dois filhos v_{left} , v_{right} **then**
      if $possivel(v_{left}) \cap possivel(v_{right}) = \emptyset$ **then**
        $possivel(v) = possivel(v_{left}) \cup possivel(v_{right})$
      else
        $possivel(v) = possivel(v_{left}) \cap possivel(v_{right})$
      end if
    end if
  end if
end for
Parte III: Percorre T em pré-ordem definindo $label'$;
$label'(root) = rand(possivel(root))$
for $\forall v \in V$ **do**
  if $label(v) \neq \lambda$ **then**
    $label'(v) = label(v)$;
  end if
  if $label(v) = \lambda$ com pai v_{father} **then**
    if $label'(v_{father}) \in possivel(v)$ **then**
      $label'(v) = label'(v_{father})$;
    else
      $label'(v) = rand(possivel(v))$;
    end if
  end if
end for

é nó interno vivo com estado da característica t. Então, em v não faz sentido considerar a possibilidade de rotular com estados diferentes de t. Desta forma, o escore parcimônia em v, para estados diferentes de t deve ser mantido como ∞. Entretanto, a definição do escore parcimônia para t deve ser diferente da atribuição feita no caso das folhas, que foi 0. Caso fosse atribuído 0, seriam ignorados os custos envolvidos nas trocas da subárvore cuja raiz é v. Mas o algoritmo Sankoff , dado o nó v e estado da característica t, calcula o menor custo parcimônia possível caso escolhido t para rotular v. Como a rotulação do nó interno vivo v é t, então usa-se este valor pois ele manifesta o escore parcimônia da subárvore enraizada por v.

Ao final, o algoritmo rotula os nós internos conforme o menor valor obtido na raiz. Observe que a rotulação definida pelas linhas 27 e 28 irá preservar a rotulação dos nós folhas e internos vivos, pois se v é nó interno vivo ou folha rotulado por t e for filho à esquerda, por exemplo, de v_{father}, então $\forall(r \neq t).s(v,r) = \infty$, consequentemente $melhor(v_{father}, r, left) = t$ qualquer que seja r estado da característica que rotule v_{father}.

4.3 Comentários

Neste capítulo, apresentamos soluções polinomiais para SLPP, baseadas em simples adaptações dos algoritmos originais de Fitch e Sankoff, apresentados na Seção 2.1, e que foram propostos para SPP segundo a abordagem sugerida em [32]. Essas adaptações alteram apenas a parte I do Algoritmo 4.1, incluindo a definição do conjunto *possivel* para os nós internos vivos; e as partes I e II do Algoritmo 4.2, definindo e atualizando a função $s(v,t)$ para v nó interno vivo. Desta forma, o algoritmo Fitch Live tem a mesma complexidade $O(k.n)$ do algoritmo Fitch e o algoritmo Sankoff Live tem a mesma complexidade $O(k^2.n)$ do algoritmo Sankoff. Os algoritmos Fitch Live e Sankoff Live foram originalmente apresentadas por nós em [24].

A partir de agora, todo o texto segue no tratamento do problema da parcimônia grande para filogenia viva.

Algoritmo 4.2 Sankoff Live.

Entrada: (1) Uma árvore $T = (V, E)$ com $root \in V$ sua raiz, (2) um alfabeto de k letras Σ , (3) uma função $label : V \to \Sigma \cup \{\lambda\}$ rotulando todas as folhas e nós internos vivos por elementos de Σ e (4) uma $k \times k$ matriz de custos δ_{ij} .

Saída: (1) Uma função $label' : V \to \Sigma$ minimizando o escore parcimônia.

Parte I: Define $s(v, t)$ para folhas e nós internos vivos;

for $\forall v \in V$ **do**
  if $label(v) \neq \lambda$ **then**
    for $\forall t \in \Sigma$ **do**
      $s(v, t) = \infty$;
    end for
    $s(v, label(v)) = 0$;
  end if
end for

Parte II: Percorre T em pós-ordem calculando $s(v, t)$ para demais nós;

for $\forall v \in V$ nó interno com filhos v_{left} , v_{right} **do**
  if $label(v) \neq \lambda$ **then**
    $t = label(v)$;
    $s(v, t) = \min_i(s(v_{left}, i) + \delta_{it}) + \min_j(s(v_{right}, j) + \delta_{jt})$;
    $melhor(v, t, left) = i$;
    $melhor(v, t, right) = j$;
  end if
  if $label(v) = \lambda$ **then**
    for $\forall t \in \Sigma$ **do**
      $s(v, t) = \min_i(s(v_{left}, i) + \delta_{it}) + \min_j(s(v_{right}, j) + \delta_{jt})$;
      $melhor(v, t, left) = i$;
      $melhor(v, t, right) = j$;
    end for
  end if
end for

Parte III: Percorre T em pré-ordem definindo $label'$;

$label'(root) = i | \min_i s(root, i)$
for $\forall v \in V$ **do**
  if v nó interno com filhos v_{left} , v_{right} **then**
    $label'(v_{left}) = melhor(v, label'(v), left)$;
    $label'(v_{right}) = melhor(v, label'(v), right)$;
  end if
end for

Capítulo 5

O problema da parcimônia grande para filogenia viva

Neste capítulo, definimos formalmente o Problema da Parcimônia Grande Viva e propomos algoritmos *branch-and-bound* de diferentes complexidades, além de algumas heurísticas. A partir do problema LPP, definido na Seção 2.2, tem-se a adaptação abaixo, para a definição do que chamamos de LLPP, em que $S(T)$ é dado pela Equação 2.1 (pg. 10).

Definição 5.1. *Problema da Parcimônia Grande Viva (LLPP)*
Instância: Uma matriz $M_{n\times m}$ e uma constante $B \in \mathbb{R}_+$.
Questão: Existe uma árvore T, completamente rotulada, com $l \geq 0$ nós internos vivos e $n - l$ folhas, sendo as $n - l$ folhas e os l nós internos vivos rotulados pelas n linhas da matriz M, tal que $S(T) \leq B$?

Antes de apresentarmos os algoritmos baseados em *branch-and-bound* e heurísticas para o problema, vamos aqui fazer considerações acerca de sua complexidade, com base no trabalho feito por Graham e Foulds [22], que provam que uma versão do Problema de Steiner em Grafos, vista a seguir, é NP-completo. Vamos então mostrar que essa versão é equivalente ao nosso problema LLPP.

Antes de apresentar os problemas e detalhes da prova, Graham e Foulds [22, pg. 134] fazem uma observação bastante importante e interessante, levando-se em conta Filogenia Viva:

O problema é construir uma filogenia em que cada unidade taxonômica operacional (OTU) é representada por um único ponto. Todos os outros pontos representam unidades taxonômicas hipotéticas (HTUs), cada uma com uma nova sequência distinta introduzida a fim de minimizar o número total de substituições de nucleotídeos necessárias para conectar o conjunto dado de taxons. Note que não é necessário que as OTUs sejam associadas apenas às pontas dos ramos ou pontos finais da filogenia, nem que a filogenia seja bifurcada no sentido de que todos os pontos interiores são de grau três. Isso representa uma partida do problema mais restritivo estudado por Fitch.

Definição 5.2. *Grafo Ponderado*
Um grafo ponderado é um par ordenado (G, w), no qual $G = (V, E)$ é um grafo e w uma função $w : E \to \mathbb{R}$.

Definição 5.3. *Problema de Steiner em Grafos (SPG - Steiner Problem in Graphs) Seja (G, w) um grafo ponderado conexo, com $G = (V, E)$, e $X \subseteq V$ não-vazio. Encontre $F \subseteq E$ tal que (X, F) é uma árvore chamada de Árvore de Steiner Minimal (SMT) e $\Sigma_{(i,j)\in F} w(i, j)$ é mínimo.*

O problema de otimização SPG pode ser transformado no seguinte problema de decisão SPG'.

Definição 5.4. *Problema de Steiner em Grafos - versão de decisão - (SPG')*
Instância: um grafo ponderado conexo (G, w) com $G = (V, E)$, um conjunto não vazio $X \subseteq V$ e uma constante $B \in \mathbb{R}_+$.
Questão: Existe $F \subseteq E$ tal que:
(1) (X, F) é uma árvore de Steiner Minimal; e
(2) $\sum_{(i,j)\in F} w(i, j) \leq B$ *?*

Considere $G = (\Sigma^m, S*)$, com Σ sendo o alfabeto (conjunto de possíveis estados da característica), $S* = \{(x, y) : x, y \in \Sigma^m \wedge d_H(x, y) = 1\}$, $d_H(x, y)$ é a distância de Hamming e $\{s_1, s_2, ..., s_n\}$ o conjunto de pontos associados às n espécies sobre as quais se pretende construir a filogenia. Graham e Foulds [22] restringem $\Sigma = \{0, 1\}$, ou seja, consideram apenas características binárias. G desta forma construído é também chamado o 1-esqueleto do m-cubo[1], ou seja, G é formado por tuplas de

[1] O 1-esqueleto do m-cubo é o grafo formado por todos os vértices e arestas do m-cubo. Por exemplo, o 1-esqueleto do 2-cubo é um circuito com 4 vértices e 1-esqueleto do 3-cubo é o grafo que também é conhecido como 3-cubo.

tamanho m contendo valores 0 ou 1 e dois vértices são adjacentes se e somente se sua distância de Hamming é 1. Em outras palavras, dois vértices são adjacentes se e somente se diferem apenas em um valor da tupla.

Graham e Foulds [22] afirmam que o problema de construir a árvore mais parcimoniosa é um caso especial de SPG (respectivamente SPG') sobre $G = (\Sigma^m, S*)$ no qual X, o conjunto de pontos a serem conectados, é $\{s_1, s_2, ..., s_n\}$.

Graham e Foulds [22] denotam Σ^m por Q_m e definem o Problema de Steiner para Q_m o qual é provado ser NP-completo.

Definição 5.5. *Problema de Steiner para Q_m - SPQ (Steiner Problem for Q_m) é o SPG' em que G é o 1-esqueleto do m-cubo.*

Observe que, na caracterização de SPQ, em nenhum momento exige-se que os pontos a serem cobertos estejam nas folhas da árvore.

Observe também que ser 1-esqueleto implica em que todas as arestas (p_i, p_j) de uma árvore que é uma possível solução terão $w(i, j) = 1$, ou seja, distância de Hamming entre nós adjacentes igual a 1.

Mapeamento de LLPP ao SPQ

Provaremos que LLPP e SPQ são equivalentes. O mapeamento de instâncias é imediato: uma instância de LLPP é formada por uma matriz $M_{n\times m}$ e $B \in \mathbb{R}_+$, já uma instância de SPQ é formada pelo grafo $G = (\Sigma^m, S*)$, $X = \{s_1, s_2, ..., s_n\}$ e $B \in \mathbb{R}_+$. Utilizaremos o mesmo valor B e, para esta prova, consideraremos a matriz $M_{n\times m}$ uma matriz binária, ou seja, $\Sigma = \{0, 1\}$. Em ambos os problemas estamos utilizando distância de Hamming. Finalmente, mapeamos cada elemento de $X = \{s_1, s_2, ..., s_n\}$ em uma linha de $M_{n\times m}$. É importante ressaltar que em SPQ uma possível resposta é uma árvore que é um subgrafo de $G = (\Sigma^m, S*)$ enquanto que em LLPP uma possível resposta é uma árvore em que cada nó é rotulado por um valor em Σ^m.

Teorema 5.6. *Equivalência LLPP - SPQ*
Os problemas LLPP e SPQ são equivalentes.

Demonstração. Será mostrado que uma resposta *sim* para SPQ permite construção de uma resposta *sim* para LLPP e vice-versa.

Uma resposta *sim* para SPQ é uma árvore T' que cobre X e T' é subgrafo de $G = (\Sigma^m, S*)$ o 1-esqueleto do m-cubo.

Como T' é subgrafo de G, $S*$ é distância de Hamming e G é 1-esqueleto do m-cubo, a distância entre nós adjacentes em T' é sempre 1. Desta forma, o caminho entre quaisquer dois nós p e q pertencentes à T' tem comprimento maior ou igual a $d_H(p, q)$.

T'' é construída a partir de T' eliminando-se todos os nós internos v tal que $v \notin X$ e $grau(v) = 2$. A eliminação consiste na retirada do nó v e substituição das arestas adjacentes por uma única aresta ligando os nós adjacentes a v. A Figura 5.1 apresenta um exemplo da retirada consecutiva de dois nós no caminho entre p e q, considerando $m = 4$.

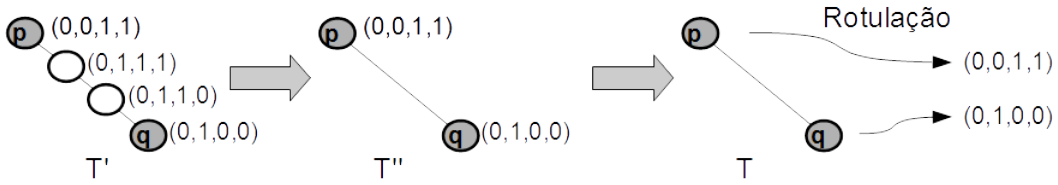


Figura 5.1: Transformação de caminho em T' para aresta em T'' e posterior aresta em T.

Para quaisquer p e q adjacentes em T'', o caminho entre p e q em T' tem comprimento maior ou igual a $d_H(p, q)$ que é a distância entre p e q em T.

Consequentemente, $S(T'') = \sum_{(p,q)\in T''} d_H(p, q) \leq \sum_{(p_i,p_j)\in T'} d_H(p_i, p_j) \leq B$.

Como $\sum_{(p_i,p_j)\in T'} d_H(p_i, p_j) \leq B$ e cada aresta, já observamos no começo da prova, tem distância igual a 1, então esta retirada de nós é efetuada, no máximo, B vezes, ou seja, é uma transformação polinomial.

Agora geramos a árvore T a partir de T'', em que cada nó $p'' = (p_1, ..., p_m)$ em T'' gerará um nó p em T rotulado por $(p_1, ..., p_m)$ (segunda transformação na Figura 5.1). Caso o nó $p'' \in X$ então a rotulação de p é uma das linhas de $M_{n\times m}$ e p é folha ou nó interno vivo.

Entretanto, a árvore T pode conter nó interno vivo q com $grau(q) = 2$; ou nó interno q, vivo ou não, com $grau(q) > 3$. Será realizada mais uma sequência de

transformações em T que não alteram seu custo, de forma a eliminar estas situações gerando uma árvore bifurcada. As transformações serão de dois tipos.

O primeiro tipo de transformação elimina os nós internos vivos com grau 2. Seja q um nó interno vivo qualquer com $grau(q) = 2$ adjacente a p e r. É criado um novo nó interno x que é ligado a p e r com a mesma rotulação de q. O nó q é transformado em folha e ligado ao nó interno x. Desta forma, o caminho entre p e r mantém o mesmo custo pois q foi substituido por x com a mesma rotulação. Além disto, a nova aresta que liga q a x tem custo zero, ou seja, esta transformação não altera o custo da árvore. Podemos então transformar todos os nós internos vivos com grau 2 em folhas mantendo o custo da árvore. A Figura 5.2 mostra esta transformação.

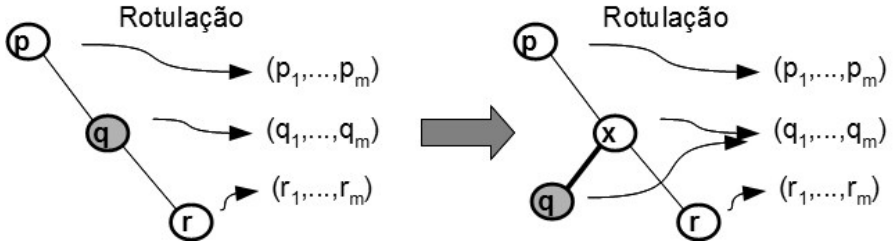


Figura 5.2: Transformação de q nó interno vivo com $grau(q) = 2$ em folha, preservando a rotulação e o custo de T.

O segundo tipo de transformação diminui o grau dos nós internos, vivos ou não, que tiverem grau maior que 3. Seja q um nó interno vivo qualquer com $grau(q) = c + 2 > 3$ adjacente a p, r, $x_1 \ldots x_c$. É criado um novo nó interno z que é ligado a q e $x_1 \ldots x_c$ com a mesma rotulação de q. O nó q é ligado ao nó interno z e desligado dos nós $x_1 \ldots x_c$. Desta forma o caminho entre p e x_i $(0 < i \leq c)$ mantém o mesmo custo pois o nó interno que foi acrescido ao caminho tem a mesma rotulação de q. Então esta transformação não altera o custo da árvore. Podemos então transformar todos os nós internos q com $grau(q) = c+2 > 3$ em um nó interno de grau 3 e outro nó interno z com $grau(z) = c+1$ mantendo o custo da árvore. A Figura 5.3 mostra esta transformação.

Observe que o número máximo de nós internos que necessitem de transformações do primeiro tipo é n. Da mesma forma, o número máximo de aplicações da transformação de segundo tipo também é n.

Aplicando recursivamente ambas as transformações, a um custo polinomial, é possivel construir $T*$ uma árvore com o mesmo custo que T contendo apenas folhas e nós internos de grau 3 e que cobre X.

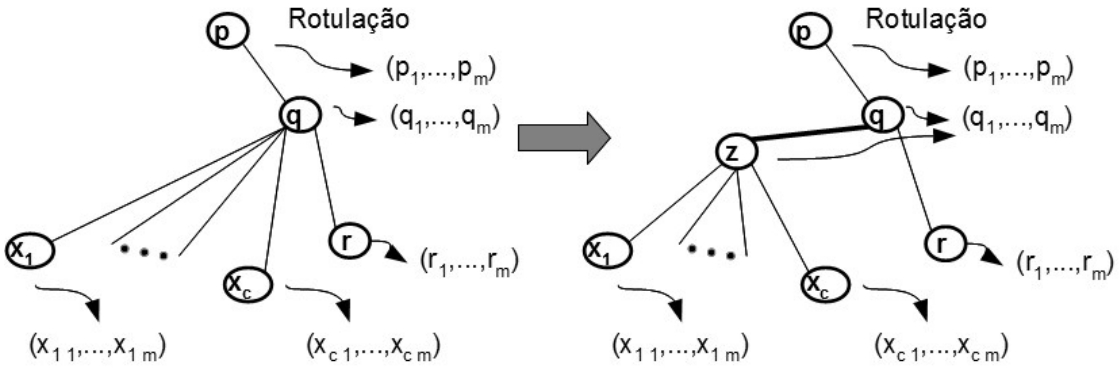


Figura 5.3: Transformação de q nó interno, vivo ou não, com $grau(q) = c + 2 > 3$ em nó q com $grau(q) = 3$ e nó z com $grau(z) = c + 1$, preservando a rotulação e o custo de T.

Como $T*$ tem nós rotulando todas as linhas de $M_{n\times m}$ e $S(T*) = S(T) \leq B$, temos que $T*$ é uma resposta *sim* para LLPP.

Por outro lado, uma resposta *sim* para LLPP é uma árvore T com nós rotulando todas as linhas de $M_{n\times m}$, cujos demais nós estão rotulados em Σ^m e $S(T) = \sum_{(p,q)\in T} d_H(p,q) \leq B$.

Caso existam p e q adjacentes em T com a mesma rotulação $(p_1, ..., p_m)$, ou seja, $d_H(p,q) = 0$ estes serão transformados em um único nó $(p_1, ..., p_m)$ na árvore T'.

Dados p e q adjacentes em T com $j = d_H(p,q) > 0$ rotulados por $(p_1, ..., p_m)$ e $(q_1, ..., q_m)$ é possível construir uma sequência de nós p_i, $0 \leq i \leq j$, adjacentes em T', tal que $p_0 = (p_1, ..., p_m)$, $p_j = (q_1, ..., q_m)$ e $d_H(p_i, p_{i+1}) = 1$. Esta sequência é construída iniciando por $p_0 = (p_1, ..., p_m)$ e alterando, cumulativamente, em cada p_i uma característica cujo estado é diferente em p e q. Um exemplo pode ser visto na Figura 5.4. Como $d_H(p,q) = j$ ao final de j alterações será obtido q. Esta sequência é tal que: $d_H(p,q) = \sum_{0\leq i\leq j-1} d_H(p_i, p_{i+1})$.

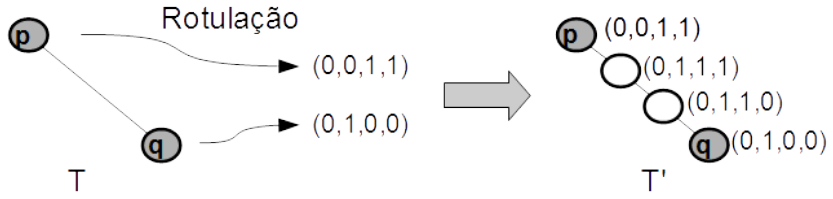


Figura 5.4: Transformação de aresta em T para caminho em T'

Repetindo esta construção para cada p e q adjacentes em T, obtem-se uma árvore T'

que cobre X em G, o 1-esqueleto de Q_m. Observe que o número de nós criados é, no máximo, B, ou seja, a transformação de T em T' é polinomial. T' construída desta forma é tal que $S(T') = \sum_{(p_i,p_j)\in T'} d_H(p_i,p_j) = \sum_{(p,q)\in T} d_H(p,q) \leq B$. Desta forma, T' é resposta *sim* para SPQ.

□

Apresentamos agora uma solução exata utilizando uma estratégia *branch-and-bound*.

Posteriormente LLPP será tratado através de heurísticas, que são estratégias que buscam obter solução com escore aceitável, sem garantia de encontrar a solução ótima, mas que não têm um crescimento exponencial na quantidade de operações.

5.1 *Branch-and-Bound*

Propõe-se a utilização de *branch-and-bound* como forma de obter solução exata para instâncias de tamanho maior do que as obtidas por solução *backtracking*.

A ideia inicial é aplicar a mesma estratégia de inclusão de espécies do caso tradicional, com a permissão de que as espécies possam ser inseridas em nós internos, transformando-os em nós internos vivos. A Figura 5.5 ilustra duas possíveis inclusões do nó 4, sendo na primeira (Figura 5.5a) como nó folha e na segunda (Figura 5.5b) como nó interno vivo.

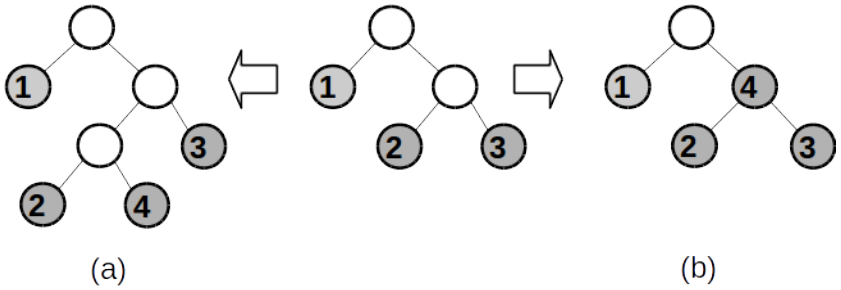


Figura 5.5: Duas possíveis inserções de um nó 4.

Uma das premissas da estratégia *branch-and-bound* é que todas as possíveis árvores possam ser, de alguma forma, obtidas. A simples ampliação do *branch-and-bound* original com a permissão da inclusão de espécies em nós internos, infelizmente, não consegue alcançar todas as árvores. A Figura 5.6 mostra uma árvore com um nó

interno vivo impossível de ser obtida caso a ordem de inclusão de espécies seja: 0 1 2 3 4. Então é preciso refinar esta ideia.

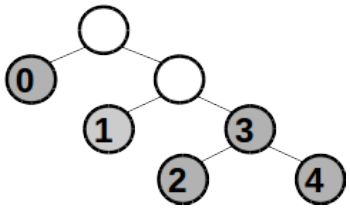


Figura 5.6: Árvore impossível de obter via estratégia básica estendida.

A solução proposta é gerar todas as possíveis ordens de inclusão das espécies e executar a estratégia anterior para cada uma delas. Esta estratégia será chamada de *branch-and-bound estendido* sendo detalhada no Algoritmo 5.1.

Em linhas gerais, o algoritmo funciona com três laços aninhados:

- o mais externo gera todas as possíveis ordens de inclusão das espécies, armazenadas em um vetor de n posições $OrdemEspecie$;
- o laço seguinte gera a ordem de construção das árvores, controlando em que aresta ou nó interno cada espécie será incluída. Esta ordem será armazenada em um vetor de n posições $OrdemConstrucao$;
- o mais interno executa a sequência até completar a árvore ou abandonar esta construção:

  (1) percorre a árvore parcial numerando as arestas e nós internos;

  (2) insere espécie $OrdemEspecie[i]$ na aresta $OrdemConstrucao[i]$ ou no nó interno (tornando-o vivo) indicado por $OrdemConstrucao[i] - 2(i-1)$;

  (3) testa se a árvore parcial pode gerar solução ótima, caso a resposta seja negativa interrompe o laço.

Observe que para numerar as arestas e nós internos é utilizada pós-ordem. A cada nova espécie incluída é necessário percorrer e numerar novamente as arestas e nós internos, o que é feito pela subrotina $PercoreArvoreNumerando$ (linha 7).

O algoritmo sempre inicia com uma árvore binária contendo as espécies indicadas por $OrdemEspecie[0]$ e $OrdemEspecie[1]$. Como não há variação neste início, define-se $OrdemConstrucao[0] = OrdemConstrucao[1] = 0$.

Algoritmo 5.1 *Branch-and-Bound Estendido.*

Entrada: (1) um conjunto de n espécies $\{E_1, ..., E_n\}$, (2) um conjunto de m características $\{C_1, ..., C_m\}$.

Saída: Uma árvore T minimizando o escore parcimônia.

1: $melhor \leftarrow MaxNum; OrdemEspecie \leftarrow E_1, ..., E_n$
2: **while** $OrdemEspecie[0] < n$ {Não testou todas Ordens de Espécies} **do**
3: $OrdemConstrucao \leftarrow ZeraOrdemConstrucao[n]$
4: **while** $OrdemConstrucao[1] = 0$ {Não testou todas Ordens de Construções} **do**
5: $i \leftarrow 0; Continua \leftarrow true$
6: **while** $Continua = true$ e $i < n$ {Árvore incompleta e promissora} **do**
7: PercoreArvoreNumerando(T)
8: **if** $OrdemConstrucao[i] < 2(i-1)$ {Vai ligar em Aresta} **then**
9: $T \leftarrow EspetaNaAresta(T, OrdemEspecie[i], OrdemConstrucao[i])$
10: **else**
11: $T \leftarrow PromoLive(T, OrdemConstrucao[i] - 2(i-1), OrdemEspecie[i])$
12: **end if**
13: $Continua \leftarrow TestaParcimonia(T, melhor)$ {T é promissora?}
14: $i \leftarrow i + 1$
15: **end while**
16: **if** $i = n$ e $Parcimonia(T) < melhor$ {Saiu pois completou T} **then**
17: $melhor \leftarrow Parcimonia(T)$
18: $melhorArvore \leftarrow T$
19: **end if**
20: $OrdemConstrucao \leftarrow GeraProxOrdemConstrucao(OrdemConstrucao, i)$
21: **end while**
22: $OrdemEspecie \leftarrow GeraProximaOrdemEspecie(OrdemEspecie)$
23: **end while**

O *branch* é garantido na execução da função *GeraProxOrdemConstrucao*, que recebe i como parâmetro, indicando em que posição a construção da árvore parou. Esta função executa um *backtracking* que carrega em *OrdemConstrucao* a próxima árvore a ser construída no nível em que ela está, dispensando a análise das árvores geradas a partir da atual. Caso o nível atual tenha esgotado as possibilidades, a função carrega a próxima árvore no nível acima. Repete isto até gerar $OrdemConstrucao[1] = 1$ o que indica condição de término.

Vamos apresentar um exemplo para ilustrar o funcionamento dos vetores *OrdemConstrucao* e *OrdemEspecie*. Considere a ordem de inclusão das espécies 0 1 2 3 4. A Figura 5.7 apresenta as árvores resultantes da aplicação das ordens de construção 0 0 1 0 5 (Figura 5.7a) e 0 0 1 2 3 (Figura 5.7b). As arestas estão rotuladas conforme a numeração atribuída pelo algoritmo e as setas indicam qual espécie será inserida em qual aresta.

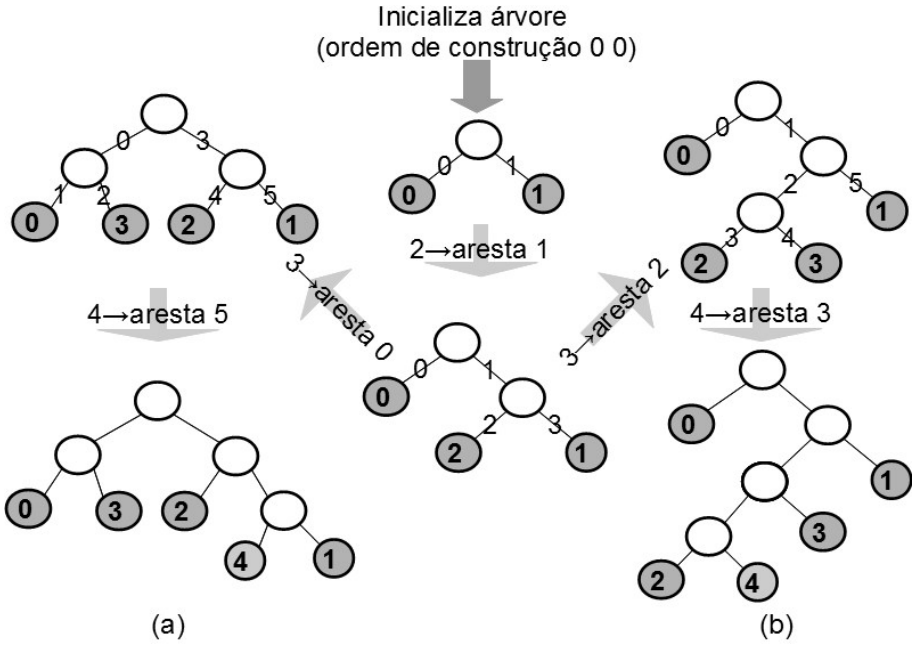


Figura 5.7: Árvore geradas pelas ordens de construção 0 0 1 0 5 e 0 0 1 2 3.

De forma a contemplar a inclusão de espécies como nós internos vivos é permitido que, em uma posição $i \geq 2$, a ordem de construção tenha valores maiores do que a quantidade de arestas na árvore parcial, ou seja, $OrdemConstrucao[i] \geq 2(i-1)$. Neste caso, vai falhar no teste do *if* da linha 8 e executar a linha 11, colocando a espécie indicada por $OrdemEspecie[i]$ no nó interno indicado pelo valor $OrdemConstrucao[i] - 2(i-1)$. Entretanto, faz-se necessário controlar a quantidade

de nós internos vivos da árvore parcial para evitar tentar ligar nós em arestas ou nós internos inexistentes. Isto é feito na função *GeraProxOrdemConstrucao*.

Agora veremos dois exemplos em que serão gerados nós internos vivos. Considerando a ordem de inclusão das espécies 0 1 2 3 4, a Figura 5.8 apresenta as árvores resultantes da aplicação das ordens de construção 0 0 1 5 2 e 0 0 1 2 6. Na segunda, a última espécie é inserida no nó raiz (numerado como 0).

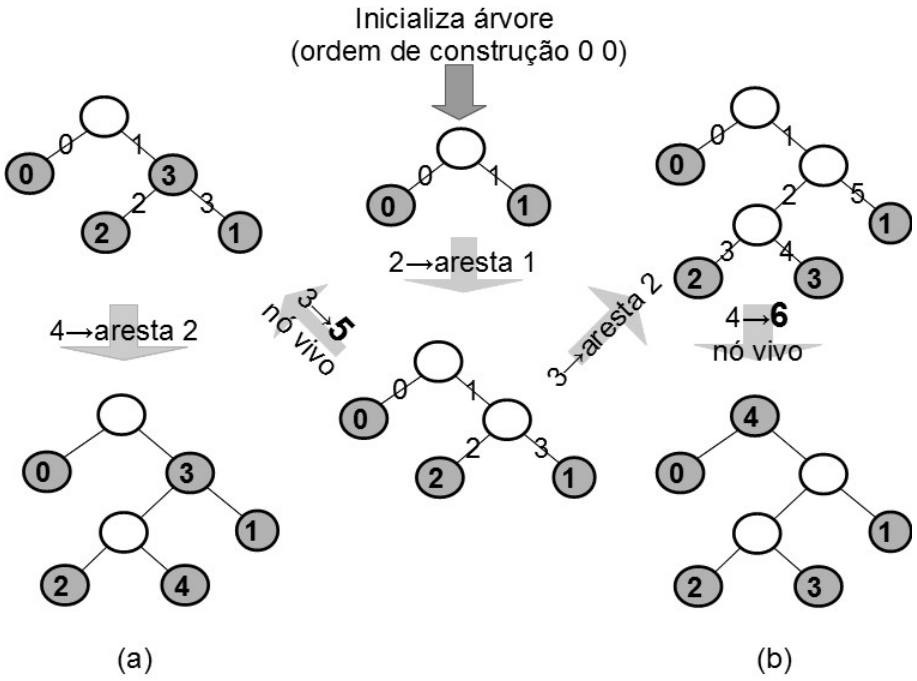


Figura 5.8: Árvore geradas pelas ordens de construção 0 0 1 5 2 e 0 0 1 2 6.

Outro exemplo da construção de árvores é mostrado na Figura 5.9. Considerando uma ordem de inclusão das espécies 0 1 2 3 e uma ordem de construção parcialmente executada 0 0 1, as próximas árvores parciais a serem construídas e testadas pela inclusão de folhas (opções 0, 1, 2 e 3) são vistas na Figura 5.9(a) e as com inclusão de nós internos vivos (opções 4 e 5) na Figura 5.9(b).

Um exemplo de aplicação do *branch-and-bound estendido* é apresentado na Figura 5.10. Para a matriz M o algoritmo retorna a árvore apresentada na figura.

Observe que uma árvore mais parcimoniosa obtida pelo *branch-and-bound estendido* pode ser exatamente a mesma, como neste caso, da obtida pelo *branch-and-bound* tradicional apresentada na Seção 2.2 ou pelo menos ter o mesmo escore. Se desejar obter a árvore mais parcimoniosa com $i > 0$ nós internos vivos, basta alterar a

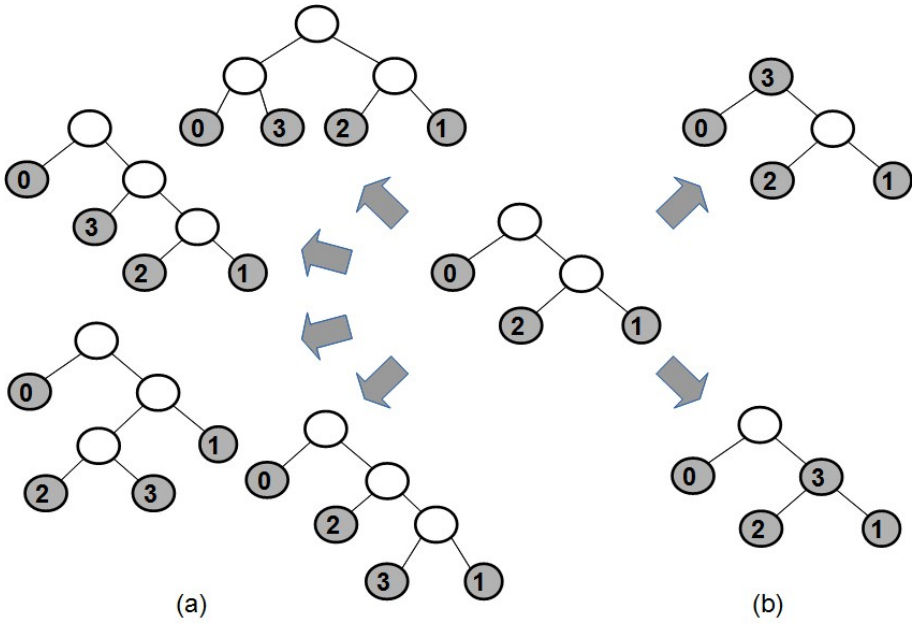


Figura 5.9: Árvores parciais obtidas a partir da ordem de construção parcialmente executada 0 0 1 gerando folhas (a) ou nós internos vivos (b).

	Características	
Espécies	1	2
0	A	A
1	T	T
2	C	G
3	A	C
4	G	A
5	C	T

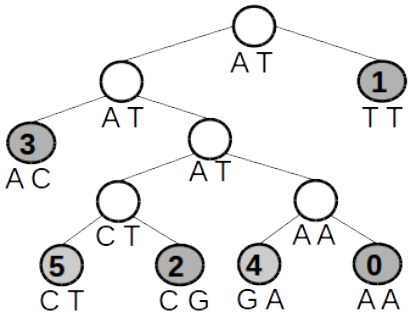


Figura 5.10: Matriz M e árvore obtida pelo *branch-and-bound estendido* com escore parcimônia igual a 6.

função *GeraProxOrdemConstrucao* para que gere apenas construções que tenham i nós internos vivos.

A Figura 5.11 mostra a árvore com escore parcimônia mínimo para a matriz M obtida pela aplicação deste *branch-and-bound* com $i > 0$ nós internos vivos.

Para provar que todas as árvores com $l \geq 0$ nós internos vivos podem ser geradas no *branch-and-bound estendido*, considere T uma árvore qualquer com l nós internos vivos. Como o *branch-and-bound estendido* inclui toda a geração de árvores do *branch-and-bound* tradicional, se $l = 0$ é garantido que T será gerada. Se $l > 0$ basta considerar uma ordem de espécies em que as l espécies que aparecem como

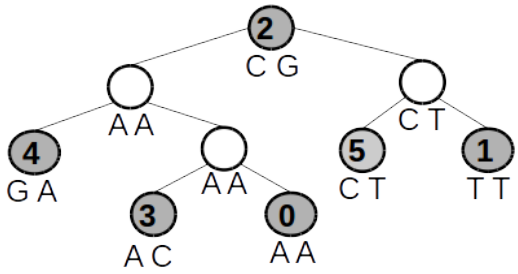


Figura 5.11: Árvore com $i > 0$ nós internos vivos e escore parcimônia mínimo igual a 6.

internos vivos em T sejam as últimas na ordem de inclusão. Todas as ordens de construção são analisadas, em particular todas as que geram uma árvore parcial T' igual a T sem considerar os nós internos vivos. Dentre estas, será analisada a que promove as l últimas espécies em nós internos vivos exatamente da maneira que estão dispostos em T.

5.2 *Branch-and-Bound* otimizado

O *branch-and-bound estendido*, apresentado por nós em [25], resolve LLPP alcançando todas as árvores no espaço de busca, porém apresenta uma complexidade de tempo elevada. Nesta seção será apresentado um outro *branch-and-bound* que não necessita gerar todas as possíveis ordens de inserção das espécies, diminuindo consideravelmente a complexidade de tempo. Este é chamado de *branch-and-bound otimizado*.

Para utilizar apenas uma ordem de inserção das espécies e, mesmo assim, alcançar todas as possíveis árvores contendo nós internos vivos, será necessário flexibilizar a construção das árvores, permitindo temporariamente a existência de nós internos vivos com grau 2. Este tipo de nó é chamado de *nó interno vivo perneta*, que é um nó interno vivo que possui apenas um filho. Estes nós serão utilizados de forma provisória na construção de árvores, exigindo que, antes de concluir a inserção de todas as espécies, o outro filho seja gerado transformando o nó interno vivo perneta em um nó interno vivo.

Inicialmente serão definidas as operações utilizadas na construção das árvores. Em seguida mostraremos que estas operações permitem a construção de todas as

possíveis filogenias vivas. Finalmente, mostramos que a construção de árvores utilizada neste *branch-and-bound* não repete árvores.

Considerando a definição de LLPP, dado um conjunto de espécies $S = \{E_1, E_2, ..., E_n\}$ a resposta é um árvore binária enraizada que minimize o custo. Desta forma, para construção das árvores, inicia-se com uma árvore contendo apenas o nó raiz, seguido-se operações de inserção de cada uma das espécies.

Por um abuso de notação, dadas uma espécie E_i e uma filogenia viva T, E_i em T será referido como sendo o nó em T que está associado à espécie E_i. Para simplificar a notação, proporcionando maior clareza na escrita, dados dois vértices adjacentes X e Y iremos denotar por XY a aresta que os liga.

As operações de inserção recebem uma filogenia viva T, contendo as espécies $E_1, E_2, ..., E_{i-1}$ $(i > 0)$, uma espécie E_i a ser incluída e algum argumento que indique a localização em T onde será processada a inclusão e retornam uma filogenia viva T' contendo as espécies $E_1, E_2, ..., E_i$.

As duas primeiras operações de inserção (tipo 0 e 1) são as utilizadas no *branch-and-bound* apresentado em Felsenstein [13] o qual resolve filogenia tradicional. A operação tipo 2 foi acrescentada às duas anteriores em [25] para permitir a construção de filogenia viva. As últimas três lidam com nós interno vivo perneta.

As operações são as seguintes:

- tipo 0, denotada por $O0_{E_i}(T)$: liga E_i como filho da raiz conforme Figura 5.12;

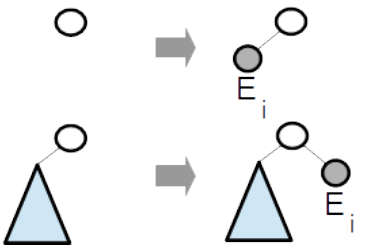


Figura 5.12: Operações tipo 0.

- tipo 1, denotada por $O1_{E_i,XY}(T)$: liga E_i na aresta XY dada, criando um novo nó interno Z, eliminando a aresta XY e inserindo as arestas XZ, ZY e ZE_i conforme Figura 5.13;

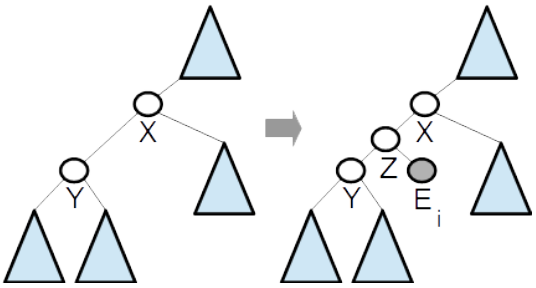


Figura 5.13: Operações tipo 1.

- tipo 2, denotada por $O2_{E_i,X}(T)$: dado um nó interno X insere E_i no lugar de X transformando este nó em nó interno vivo conforme Figura 5.14.

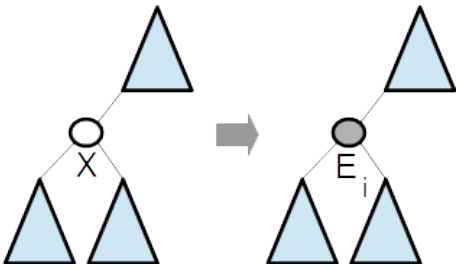


Figura 5.14: Operações tipo 2.

- tipo 3, denotada por $O3_{E_i,E_j}(T)$: dado um nó interno vivo perneta E_j liga E_i como o outro filho de E_j, criando a aresta E_jE_i transformando E_j em nó interno vivo conforme Figura 5.15;

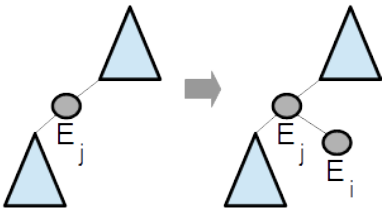


Figura 5.15: Operações tipo 3.

- tipo 4, denotada por $O4_{E_i,XY}(T)$: dada uma aresta XY liga E_i na aresta como nó interno vivo perneta, criando as arestas XE_i e E_iY conforme Figura 5.16;
- tipo 5, denotada por $O5_{E_i,E_j}(T)$: dada uma folha E_j liga E_i como filho desta folha, criando a aresta E_jE_i e transformando E_j em nó interno vivo perneta conforme Figura 5.17;

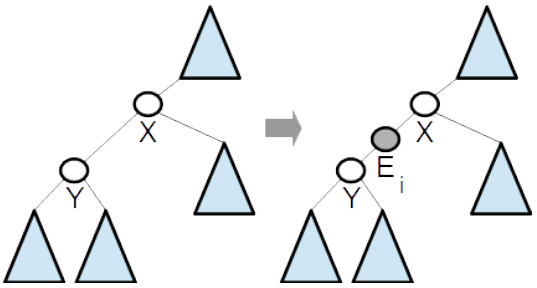


Figura 5.16: Operações tipo 4.

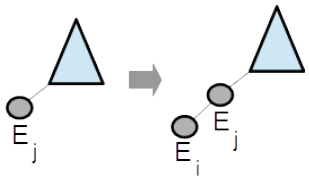


Figura 5.17: Operações tipo 5.

Todas estas operações devem respeitar a imposição de que a árvore produzida T' possua todos os nós com grau no máximo 3 e que operações tipo 0 sejam aplicadas, no máximo, duas vezes.

É necessário provar que o conjunto de operações proposto, respeitando as restrições acima, permite gerar todas as possíveis filogenias vivas. Isto é feito no Teorema 5.7.

Teorema 5.7. *Cobertura BB Otimizado*
Dado um conjunto de espécies $S = \{E_1, E_2, ..., E_n\}$, é possível obter qualquer filogenia viva sobre S a partir de uma árvore inicial T_I contendo apenas o nó raiz através da aplicação de operações de tipo 0, 1, 2, 3, 4 e 5 sobre S observando a ordem de inserção $E_1, E_2, ..., E_n$.

Demonstração. A prova será por indução sobre $|S|$.

Base: caso $|S| = 2$ apenas uma árvore é possível (Figura 5.18), cuja construção é obtida por $O0_{E_2}(O0_{E_1}(T_I))$.

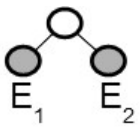


Figura 5.18: Possível árvore para $|S| = 2$.

Apesar de não ser necessário, pode ser visto que caso $|S| = 3$ existem 6 possíveis árvores, com as seguintes construções:

$O0_{E_3}(O1_{E_2,RaizE_1}(O0_{E_1}(T_I)))$ (Figura 5.19a)
$O1_{E_3,RaizE_1}(O0_{E_2}(O0_{E_1}(T_I)))$ (Figura 5.19b)
$O1_{E_3,RaizE_2}(O0_{E_2}(O0_{E_1}(T_I)))$ (Figura 5.19c)
$O2_{E_3,Raiz}(O0_{E_2}(O0_{E_1}(T_I)))$ (Figura 5.20a)
$O0_{E_3}(O2_{E_2,Raiz}(O0_{E_1}(T_I)))$ (Figura 5.20b)
$O0_{E_3}(O0_{E_2}(O2_{E_1,Raiz}(T_I)))$ (Figura 5.20c)

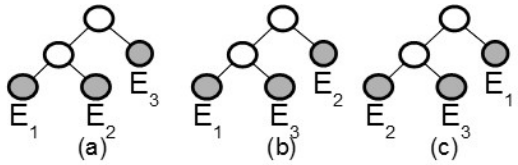


Figura 5.19: Possíveis árvores sem nós internos vivos para $|S| = 3$.

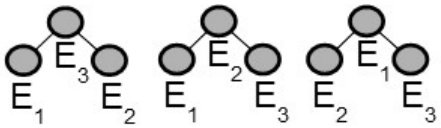


Figura 5.20: Possíveis árvores com nós internos vivos para $|S| = 3$.

Passo da Indução: Como hipótese de indução supõe-se, dado $S' = \{E_1, E_2, ..., E_n\}$, que é possível, usando as operações, gerar todas as filogenias vivas obedecendo a ordem de inserção $E_1, E_2, ..., E_n$.

É necessário provar, usando a hipótese de indução, que é possível gerar todas as filogenias vivas para $S = \{E_1, E_2, ..., E_n, E_{n+1}\}$ usando as operações e obedecendo a ordem de inserção $E_1, E_2, ..., E_n, E_{n+1}$.

Seja T uma filogenia viva qualquer sobre S, será provado que é possível obter T usando as operações e obedecendo a ordem de inserção $E_1, E_2, ..., E_n, E_{n+1}$.

Considerando a posição de E_{n+1} em T tem-se dois casos: E_{n+1} é nó interno vivo ou E_{n+1} não é nó interno vivo.

Caso E_{n+1} for interno vivo, considere T' a árvore obtida a partir de T transformando o nó interno vivo E_{n+1} em um nó interno X. Pela hipótese de indução, T' pode ser obtida a partir de S' usando as operações e obedecendo a ordem de inserção

$E_1, E_2, ..., E_n$. T' e T possuem a mesma topologia e diferem unicamente no nó interno vivo E_{n+1}.

Então $T = O2_{E_{n+1},X}(T')$.

Caso E_{n+1} não for interno vivo então será folha. A Figura 5.21 mostra as possibilidades para E_{n+1}.

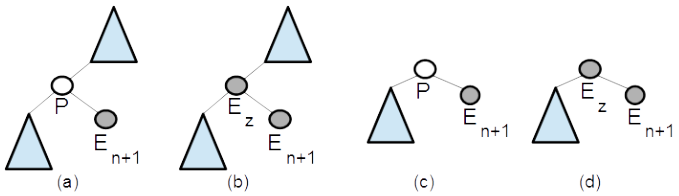


Figura 5.21: Caso em que E_{n+1} é folha.

Considerando P o pai de E_{n+1} agora analisa-se cada um dos 4 casos:

Caso a: P é nó interno e não é raiz (Figura 5.21a).

Sejam Pai_P o pai de P e $Filho_P$ o outro filho de P. Considere T' a árvore obtida a partir de T retirando os nós P e E_{n+1} e ligando diretamente Pai_P e $Filho_P$. Pela hipótese de indução T' pode ser obtida a partir de S' usando as operações e obedecendo a ordem de inserção $E_1, E_2, ..., E_n$.

Então $T = O1_{E_{n+1},Pai_P Filho_P}(T')$.

Caso b: P é nó interno vivo e não é raiz (Figura 5.21b), seja E_z $(1 \leq z \leq n)$ a espécie associada a P.

Sejam Pai_P o pai de P e $Filho_P$ o outro filho de P. Considere T' a árvore obtida a partir de T transformando o nó P em nó interno e o nó E_{n+1} em nó E_z.

Como a espécie E_{n+1} foi retirada, é possível aplicar a hipótese de indução, obtendo T' a partir de S' usando as operações e obedecendo a ordem de inserção $E_1, E_2, ..., E_n$.

Dependendo de como foi a inserção de E_z em T' serão feitas as alterações para obter T. Serão consideradas cada uma das operações para possível inserção de E_z, sempre lembrando que o pai de E_z em T' é nó interno:

- tipo 0: como o pai de E_z não é raiz ao final da construção de T', então deve

ser um outro nó interno n_k que foi gerado na inclusão E_k $(z < k \leq n)$ em uma operação $O1_{E_k,XE_z}(T'')$ em que T'' contém $E_1, E_2, ..., E_{k-1}$ e X é pai de n_k. Neste caso, T^* é gerada usando as mesmas operações que geram T' apenas substituindo $O1_{E_k,XE_z}(T'')$ por $O5_{E_k,E_z}(T'')$, tornando E_z nó interno vivo perneta em T^*. A Figura 5.22 ressalta a diferença que esta alteração produz. Observe que as operações subsequentes podem transformar o ramo criado pela inserção de E_k em uma subárvore.

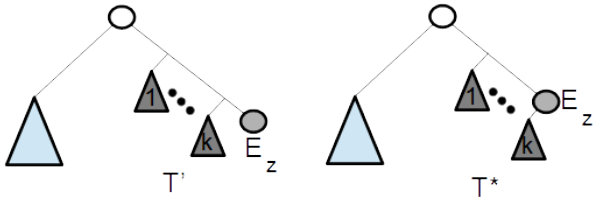


Figura 5.22: Diferença entre as árvores T' e T^*.

Desta forma, T^* é uma árvore obtida a partir de S' usando as operações e obedecendo a ordem de inserção $E_1, E_2, ..., E_n$. Observe que T^* possui um nó interno vivo perneta E_z. Então $T = O3_{E_{n+1},E_z}(T^*)$.

- tipo 1: neste caso, E_z foi inserida em uma operação $O1_{E_z,XY}(T'')$ em que T'' contém $E_1, E_2, ..., E_{z-1}$. Esta operação cria um nó interno n_w em T'.

  Caso, ao final da construção de T', n_w ainda seja pai de E_z basta substituir $O1_{E_z,XY}(T'')$ por $O4_{E_z,XY}(T'')$ gerando, ao final, uma árvore T^* com um nó interno vivo perneta E_z. A Figura 5.23 ressalta a diferença que esta alteração produz.

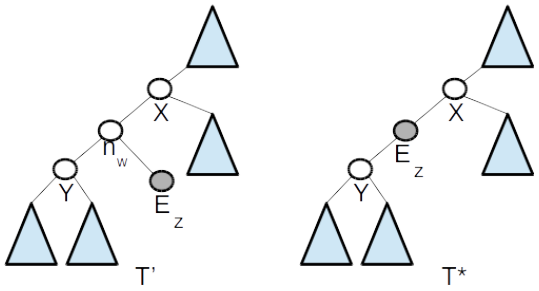


Figura 5.23: Diferença entre as árvores T' e T^*.

Por outro lado, se ao final da construção de T', o pai de E_z for um outro nó interno n_k que foi gerado na inclusão E_k $(z < k \leq n)$, então E_k foi inserida em uma operação $O1_{E_k,XE_z}(T''')$ na qual T''' contém $E_1, E_2, ..., E_{k-1}$ e X é pai

de n_k. Neste caso, T^* é gerada usando as mesmas operações que geram T', apenas substituindo $O1_{E_k,XE_z}(T''')$ por $O5_{E_k,E_z}(T''')$, tornando E_z nó interno vivo perneta em T^*. A Figura 5.24 ressalta a diferença que esta alteração produz. Observe que as operações subsequentes podem transformar o ramo criado pela inserção de E_k em uma subárvore.

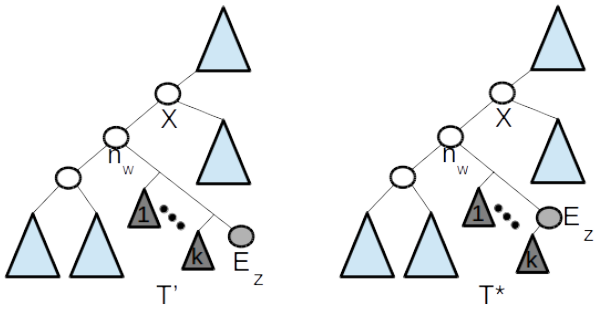


Figura 5.24: Diferença entre as árvores T' e T^*.

Em ambas as situações T^* é uma árvore obtida a partir de S' usando as operações e obedecendo a ordem de inserção $E_1, E_2, ..., E_n$. Observe que T^* possui um nó interno vivo perneta E_z. Então $T = O3_{E_{n+1},E_z}(T^*)$.

- tipo 2: esta opção é impossível pois E_z é folha.

- tipo 3: como o pai de E_z em T' é nó interno e a operação tipo 3 insere E_z como filho de nó interno vivo então deve haver outra operação subsequente tipo 1 que insira um nó interno como pai de E_z.

  Ou seja, $\exists k > z$ tal que a operação de inserção de E_k gera P nó interno pai de E_z e é do tipo $O1_{E_k,XE_z}(T'')$ em que T'' contém $E_1, E_2, ..., E_{k-1}$.

  Neste caso, T^* é gerada usando as mesmas operações que geram T' apenas substituindo $O1_{E_k,XE_z}(T'')$ por $O5_{E_k,E_z}(T'')$, tornando E_z nó interno vivo perneta em T^*. Também neste caso, a Figura 5.24 ressalta a diferença que esta alteração produz. T^* possui um nó interno vivo perneta E_z. Então $T = O3_{E_{n+1},E_z}(T^*)$.

- tipo 4: esta opção é impossível pois E_z é folha.

- tipo 5: da mesma forma que no caso tipo 3, $\exists k > z$ tal que a operação de inserção de E_k gera P, nó interno pai de E_z, e é do tipo $O1_{E_k,XE_z}(T'')$ em que T'' contém $E_1, E_2, ..., E_{k-1}$.

Novamente, substituindo esta operação por $O5_{E_k,E_z}(T'')$, obtém-se T^* que obedece a ordem de inserção $E_1, E_2, ..., E_n$ e tem um nó interno vivo perneta E_z. Então $T = O3_{E_{n+1},E_z}(T^*)$.

Caso c: P é raiz não viva (Figura 5.21c).

Seja T' a subárvore obtida do outro filho da raiz de T. T' é árvore filogenética sobre S' e, pela hipótese da indução, pode ser obtida a partir de S' usando as operações e obedecendo a ordem de inserção $E_1, E_2, ..., E_n$.

Sejam R_T a raiz de T' e $j \geq 2$, tal que E_j é inserida por $O0_{E_j}(T'')$, sendo esta a segunda operação de tipo 0 na construção de T' e T'' contém $E_1, E_2, ..., E_{j-1}$.

T^* é obtida da seguinte maneira: substituindo $O0_{E_j}(T'')$ por $O1_{E_j,R_TX}(T'')$ em que X é o outro filho da raiz R_T na árvore em construção T'', esta operação gera o nó interno n_j; alterando as operações de inserção de E_k, $k > j$, onde R_TY for argumento por n_jY, qualquer que seja Y nó de T'.

A Figura 5.25 apresenta a estratégia de construção de T^*.

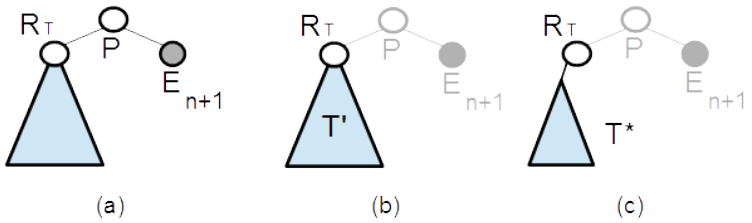


Figura 5.25: Transformação de T em árvore T' que permite aplicação da hipótese de indução, e posterior obtenção de T^*.

Desta forma T^* é árvore obtida usando as operações e obedecendo a ordem de inserção $E_1, E_2, ..., E_n$. Observe que T^* tem apenas um filho na raiz. Então $T = O0_{E_{n+1}}(T^*)$.

Caso d: P é raiz viva (Figura 5.21d), seja E_z $(1 \leq z \leq n)$ a espécie associada a P.

T' é obtida a partir de T substituindo os nós raiz e E_{n+1} por nó interno comum e E_z. A Figura 5.26 ilustra as árvores.

Por hipótese de indução, T' pode ser obtida a partir de $\{E_1, E_2, ..., E_n\}$ usando as operações e obedecendo a ordem de inserção $E_1, E_2, ..., E_n$.

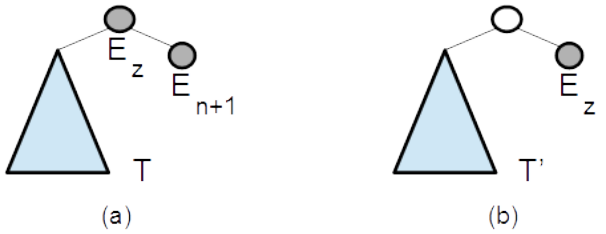


Figura 5.26: Transformação de T em árvore T' que permite aplicação da hipótese de indução.

T^* é construída alterando a operação de tipo 0, que insere E_z na construção de T', por operação de tipo 2 no nó raiz, promovendo ele a nó interno vivo. Observe que T^* tem apenas um filho na raiz. Então $T = O0_{E_{n+1}}(T^*)$.

Em todos os casos, é possível construir T sobre $S = \{E_1, E_2, ..., E_n, E_{n+1}\}$ usando as operações e obedecendo a ordem de inserção $E_1, E_2, ..., E_n, E_{n+1}$. Isto conclui o passo da indução e completa a prova.

□

Em seguida é apresentado o algoritmo *branch-and-bound* otimizado. O algoritmo constrói T obedecendo a ordem de inserção das espécies $\{E_1, E_2, ..., E_n\}$ e uma ordem de construção, que indica a operação e local de inserção de cada espécie. Funciona baseado em dois laços: o mais externo vai alternando e testando todas as ordens de construção; o mais interno constrói a árvore baseado na ordem de construção em análise.

A cada passo, no laço de construção da árvore: são recontados os nós e arestas; verificado o valor de $OrdConstrucao[i]$, que indica o tipo de operação e o local (aresta ou nó) onde será executada a inserção; é chamada a função $TestaParcimonia$ que calcula o escore parcimônia da árvore em construção e verifica se esta pode gerar árvore com escore menor que a melhor. Se a árvore for promissora então continua o processo de construção; caso contrário sai do laço. Se a construção da árvore terminou, também sai do laço.

Quando o algoritmo sai do laço de construção porque a construção da árvore terminou, testamos se um novo melhor escore parcimônia foi encontrado, caso afirmativo atualizamos o melhor escore e guardamos a melhor árvore.

Em seguida, é executada a função $GeraProxOrdemConstrucao$ que calcula a

próxima ordem de construção. Observe que esta função é responsável pelo controle da busca executada pelo *branch-and-bound*. Este controle é feito através do parâmetro i que indica em que posição deve ser tomado um caminho alternativo, caso a construção da árvore a partir da espécie E_i não seja mais promissora, ou simplesmente solicitando outra alternativa quando a construção tiver sido completada ($i = n$). Esta função também deve levar em conta o número de nós internos vivos e internos vivos pernetas na ordem de construção, de forma a não permitir que a árvore completa tenha nó interno vivo perneta e nem propor operações inválidas na construção.

Algoritmo 5.2 *Branch-and-Bound Otimizado.*

Entrada: (1) um conjunto de n espécies $\{E_1, E_2, ..., E_n\}$, (2) um conjunto de m características $\{C_1, C_2, ..., C_m\}$.

Saída: (1) Uma árvore T minimizando o escore parcimônia.

1: $melhor \leftarrow MaxNum; OrdEsp \leftarrow E_1, E_2, ..., E_n$
2: $T \leftarrow ArvorecomRaiz$
3: $OrdConstrucao \leftarrow -1 - 1\ 0 \ldots 0$
4: **while** $OrdConstrucao[0] < 2$ **do** {Não testou todas Construções}
5: $i \leftarrow 0; Continua \leftarrow true$
6: **while** $Continua = true$ e $i < n$ **do** {Árvore incompleta e promissora}
7: PercoreArvoreNumerando(T)
8: **if** $OrdConstrucao[i] = -1$ **then** {Vai ligar como filho de Raiz}
9: $T \leftarrow EspetaNaRaiz(T, OrdEsp[i])$
10: **else**
11: **if** $OrdConstrucao[i] < 2(i-1)$ **then** {Vai ligar em Aresta}
12: $T \leftarrow EspetaNaAresta(T, OrdEsp[i], OrdConstrucao[i])$
13: **else**
14: **if** $OrdConstrucao[i] < 3(i-1)$ **then** {Vai promover vivo}
15: $T \leftarrow PromoLive(T, OrdEsp[i], OrdConstrucao[i] - 2(i-1))$
16: **else**
17: **if** $OrdConstrucao[i] < 6(i-1)$ **then** {Vai gerar perneta}
18: $T \leftarrow GeraOneLeg(T, OrdEsp[i], OrdConstrucao[i] - 3(i-1))$
19: **else** {Vai completar perneta}
20: $T \leftarrow CompletaOneLeg(T, OrdEsp[i], OrdConstrucao[i] - 6(i-1))$
21: **end if**
22: **end if**
23: **end if**
24: **end if**
25: $Continua \leftarrow TestaParcimonia(T, melhor)$ {T é promissora?}
26: $i \leftarrow i + 1$
27: **end while**
28: **if** $i = n$ e $Parcimonia(T) < melhor$ **then** {Saiu pois completou T}
29: $melhor \leftarrow Parcimonia(T)$
30: $melhorArvore \leftarrow T$
31: **end if**
32: $OrdConstrucao \leftarrow GeraProxOrdemConstrucao(OrdConstrucao, i)$
33: **end while**

Tratando agora sobre complexidade, o *branch-and-bound* otimizado é melhor que o *branch-and-bound* estendido, pois dispensa a necessidade de testar várias ordens de inserção de espécies.

Conforme Hendy e Penny [27], a complexidade do *branch-and-bound* tradicional para n espécies e m características é $O(mn^n)$. No *branch-and-bound* estendido há a necessidade de gerar todas as permutações de espécies o que agrega o fator $O(n!)$, resultando em complexidade $O(n!mn^n)$.

O *branch-and-bound* otimizado substitui esta necessidade de testar todas as permutações por um aumento no número de possíveis árvores geradas em cada inclusão, permitindo nós internos vivos pernetas e nós internos vivos. Este aumento agrega fator $O(n)$ à complexidade do *branch-and-bound* tradicional, resultando numa complexidade total de $O(nmn^n)$.

Em [25], dois exemplos, um com 8 espécies e 10 características e outro com 9 espécies e 10 características, foram resolvidos pelo *branch-and-bound* estendido em aproximadamente 54 minutos e 20 horas, respectivamente, em um Intel Core i5-4590 CPU 3.30GHz com 4GB de memória. Utilizando o *branch-and-bound* otimizado, os mesmos exemplos foram resolvidos em 6 segundos e 1 minuto, respectivamente.

5.3 Heurísticas

Apesar do *branch-and-bound* otimizado melhorar o desempenho em relação ao *branch-and-bound* estendido, ainda apresenta complexidade exponencial, tornando impraticável seu uso para exemplos com mais de 12 espécies. Então é necessário o desenvolvimento de heurísticas. Propõe-se uma heurística de construção e duas de rearranjo, a partir das heurísticas descritas, respectivamente, nas Seções 3.1 e 3.2.

Heurística de Construção: Adição Sequencial Live.

Esta heurística foi apresentada em [26] juntamente com o *branch-and-bound* otimizado. Neste caso, vamos acrescentar à Adição Sequencial tradicional, a possibilidade de inserção da espécie em um nó interno, produzindo um nó interno vivo. A Figura 5.27 apresenta as possíveis inserções da espécie 3, sendo que as duas possibilidades à esquerda produziriam nó interno vivo.

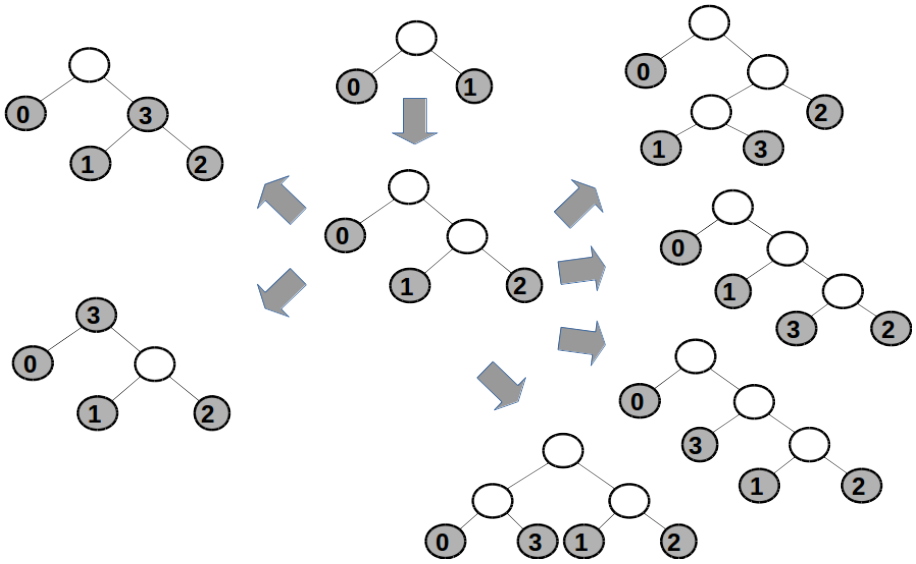


Figura 5.27: Possíveis inserções da espécie 3.

A motivação para a proposta desta heurística, adaptada da heurística Adição Sequencial tradicional, é a sua ampla utilização e baixa complexidade.

Heurística de Rearranjo 1: NNI Live.

Para NNI Live, acrescentamos ao NNI tradicional a possibilidade de, caso uma das subárvores restantes após a retirada das 5 arestas seja apenas um nó folha, transformar este nó folha em um nó interno vivo contraindo os nós internos da aresta selecionada. Esta operação é chamada *promoção*. A Figura 5.28 mostra a promoção do nó D a nó interno vivo.

Observe que a promoção não pode ser realizada se um dos nós da aresta selecionada for nó interno vivo, pois neste caso não é possível fazer a contração dos nós internos.

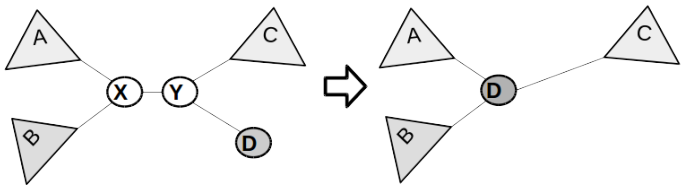


Figura 5.28: Aplicação da heurística com promoção do nó D à nó interno vivo.

Nesta heurística, assim como na seguinte, serão guardadas várias árvores com melhor escore. Quando esgotada a busca de melhoria por parte de uma destas árvores, a heurística continua a busca utilizando outra árvore dentre as guardadas, até esgotar-se o número de repetições ou todas as árvores guardadas terem sido pesquisadas.

A motivação para proposta desta heurística é que, no caso tradicional, a mesma é utilizada encontrar ótimos locais.

Heurística de Rearranjo 2: TBR Live.

Aqui vamos modificar a TBR tradicional, permitindo que a reconexão também possa ocorrer em nó folha. Dada T' a subárvore que será desconectada para TBR, a heurística escolhe Y como nó de ligação, na árvore restante (o que sobrou após a retirada de T'). Y será o ponto de reconexão entre as subárvores, tornando-se nó interno vivo.

Para exemplificar, considere a árvore na Figura 5.29, a aresta a ser retirada é a mais espessa. Na Figura 5.30 está o resultado da bisseção, supondo que o nó 3 seja o nó com menor somatório de distâncias Hamming para os nós folha de T'. A Figura 5.31 mostra o resultado da reconexão, tendo o nó 3 como nó interno vivo.

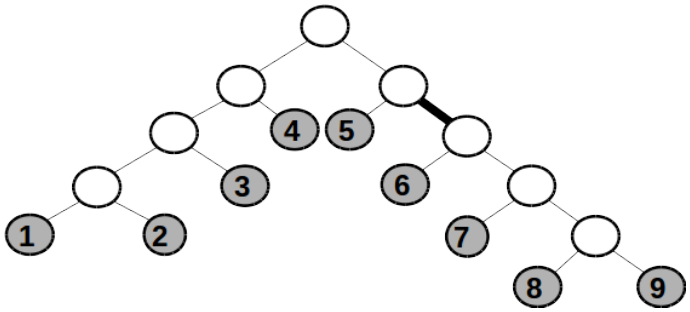


Figura 5.29: Árvore a ser aplicado TBR Live na aresta mais espessa.

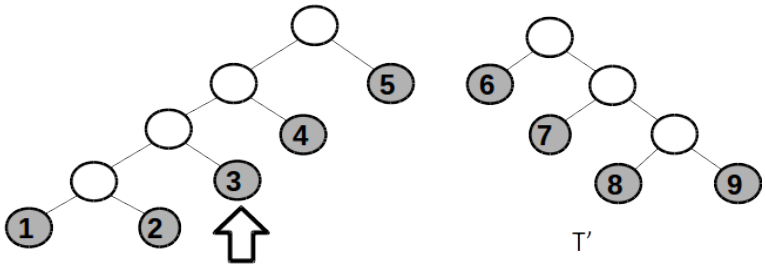


Figura 5.30: Resultado da bisseção sobre a árvore da Figura 5.29.

Propomos esta heurística para tentar escapar de ótimos locais. Ainda, são propostas duas variações como forma de otimizar seu desempenho.

A primeira é, após a bisseção, identificar a aresta "ponto fraco" da árvore (aresta que agrega maior valor ao escore parcimônia) e fazer a reconexão nesta aresta. A hipótese é que esta aresta ponto fraco, pelo fato de ser a que mais agrega ao

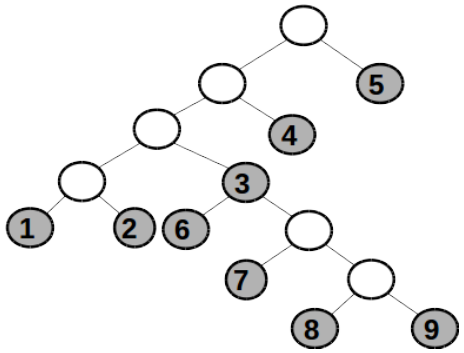


Figura 5.31: Reconexão da árvore T' da Figura 5.30 no nó 3.

escore, será o local onde a reconexão da subárvore fará o menor acréscimo no escore, consequentemente gerando a árvore de menor escore nesta iteração. Esta variação é chamada *aresta ponto fraco*.

A segunda variação consiste em, antes de realizar a bisseção, calcular em cada nó interno ou interno vivo uma taxa de ajuste. Esta taxa de ajuste considera a subárvore cuja raiz é o nó, e é dada pela razão entre o escore parcimônia da subárvore e a quantidade de espécies nela. A hipótese é de que subárvores com taxas mais baixas estejam melhor ajustadas e não devem ser alteradas. Desta forma, escolhe-se a aresta de corte de forma a isolar a subárvore com menor taxa. Esta variação é chamada *subárvore forte*.

5.4 Comentários

Neste capítulo, foi definido o problema LLPP, apresentado por nós em [25], além de considerações acerca de sua complexidade. Também foram criadas duas versões de algoritmos *branch-and-bound* que resolvem LLPP com complexidades diferentes, embora nenhum deles deixe de ter complexidade exponencial. O primeiro também foi apresentado em [25], enquanto que o segundo algoritmo, chamado *branch-and-bound* otimizado, foi publicado em [26].

Seguimos então com a busca por soluções heurísticas, sendo propostas uma heurística construtiva e duas de rearranjo. A heurística construtiva também foi publicada em [26]. Ao final deste trabalho, nos próximos capítulos, serão apresentados resultados de testes comparativos entre o *branch-and-bound* otimizado e as heurísticas propostas.

Capítulo 6

Parcimônia grande para filogenia viva com informação sobre os nós internos vivos

O problema geral (LLPP) permite a presença, ou não, de um número qualquer de nós internos vivos na árvore solução. LLPP é então dividido, neste capítulo, em dois problemas: o caso em que existem $l > 0$ nós internos vivos, desconhecidos previamente, chamado de LLPP-l, é tratado na Seção 6.1; já o caso em que existem $l > 0$ nós internos vivos conhecidos previamente, chamado de LLPP-l-def, é abordado na Seção 6.2.

6.1 Nós internos vivos quaisquer

Adaptando a Definição 5.1, temos a definição abaixo.

Definição 6.1. *Problema da Parcimônia Grande Viva l-indefinido (LLPP-l)*
Instância: Uma matriz $M_{n\times m}$, um inteiro $l > 0$ e uma constante $B \in \mathbb{R}_+$.
Questão: Existe uma árvore T, completamente rotulada, com l nós internos vivos e $n - l$ folhas, sendo as $n - l$ folhas e os l nós internos vivos rotulados pelas n linhas da matriz M, tal que $S(T) \leq B$?

Diferentemente de LLPP, para LLPP-l é possível obter uma fórmula que calcule a

quantidade de possíveis árvores. Inicialmente separa-se os l nós internos vivos. Para um conjunto de n objetos, a quantidade de possíveis subconjuntos de l objetos é dada pela quantidade de combinações simples de n elementos tomados l a l, dada pela Fórmula 6.1

$$C_l^n = \frac{n!}{l!(n-l)!} \tag{6.1}$$

Fixos os l nós internos vivos, calcula-se quantas possíveis árvores com $n-l$ folhas podem ser construídas. Conforme Felsenstein [13] existem $(2(n-l)-3)!!$ árvores com $n-l$ folhas, cada uma com $(n-l)-1$ nós internos.

Definidas a topologia e as folhas da árvore só resta calcular as possíveis distribuições dos l nós internos vivos. Neste caso, a ordem dos elementos importa e o cálculo será de arranjos simples considerando a quantidade de nós internos. Precisa-se calcular a quantidade de arranjos simples de $(n-l)-1$ elementos tomados l a l, dada pela Fórmula 6.2.

$$A_l^{(n-l)-1} = \frac{((n-l)-1)!}{((n-l)-1-l)!} = \frac{(n-l-1)!}{(n-2l-1)!} \tag{6.2}$$

Então o total de possíveis filogenias vivas com n objetos e l nós internos vivos é dado pela Fórmula 6.3.

$$T_n^l = \frac{n!}{l!(n-l)!}(2(n-l)-3)!!\frac{(n-l-1)!}{(n-2l-1)!} = \frac{(2(n-l)-3)!!n!}{l!(n-2l-1)!(n-l)} \tag{6.3}$$

NP-completude

Vamos mostrar que LLPP-l é NP-completo reduzindo o problema LPP a ele. Antes porém observam-se dois lemas:

Lema 6.2. *Sejam uma árvore T qualquer, uma folha v de T e T' obtida de T pela retirada de v e do nó interno adjacente a v. Temos $S(T') \leq S(T)$.*

Lema 6.3. *Sejam uma árvore T qualquer, v nó interno vivo em T e T' obtida de T pela transformação do nó interno vivo v em um nó interno. Temos $S(T') \leq S(T)$.*

A prova destes lemas é imediata a partir das definições de $S(T)$ (Equação 2.1 na pg. 10), do problema Parcimônia Pequeno Viva (SLPP Definição 4.1) e pela solução de SLPP (Algoritmo 4.2).

Teorema 6.4. *LLPP-l é NP-completo.*

Demonstração. Inicialmente observa-se que a solução polinomial obtida para SLPP garante que uma instância qualquer (M', l, B) de LLPP-l seja verificada em tempo polinomial, ou seja, LLPP-l está em NP.

Para completar a prova, um problema NP-completo precisa ser reduzido a LLPP-l. Faremos isso reduzindo LPP a LLPP-l.

Seja (M, B) uma instância de LPP, em que M é uma matriz $n \times m$ e $B \in \mathbb{R}_+$.

Seja o_x uma linha qualquer de M, obtém-se M' a partir de M duplicando todas as linhas e acrescentando a linha o_{x1} com estados iguais aos de o_x, ou seja, M' é obtida de M triplicando o_x e duplicando as demais linhas. É importante observar que esta redução é polinomial.

Seja $(M', 1, B)$ a instância de LLPP-l.

É necessário provar que a redução é válida. Prova-se que uma resposta *sim* a (M, B) em LPP implica em uma resposta *sim* a $(M', 1, B)$ em LLPP-l e que uma resposta *sim* a $(M', 1, B)$ em LLPP-l implica em uma resposta *sim* a (M, B) em LPP.

Se a resposta a (M, B) em LPP é *sim* então existe árvore T satisfazendo (M, B) em LPP. Obtem-se T' a partir de T duplicando todos os nós folha, ou seja, para cada v folha substitui-se v por u nó interno com filhos $v1$ e $v2$ com a mesma rotulação de v. Finalmente transforma-se o nó interno criado pela duplicação do nó relativo a o_x em nó interno vivo rotulado por o_x. A Figura 6.1 mostra como seria a aplicação desta transformação caso $n = 4$. A árvore T (Figura 6.1(a)) é transformada numa árvore T' (Figura 6.1(b)) com exatamente um nó interno vivo.

Desta forma obtêm-se T' tal que $S(T') = S(T) \leq B$, T' tem um nó interno vivo e todas as folhas e o nó interno vivo rotuladas pelas linhas de M', ou seja, T' confere resposta *sim* a $(M', 1, B)$ em LLPP-l.

Se a resposta a $(M', 1, B)$ em LLPP-l for *sim* então existe T' com $S(T') \leq B$ e exatamente um nó interno vivo.

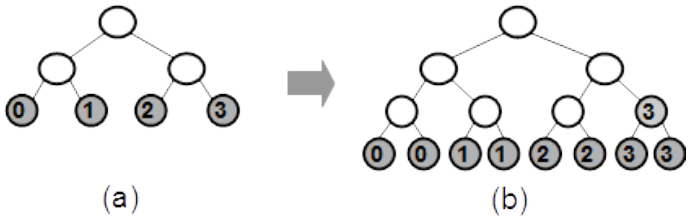


Figura 6.1: Árvore de LPP transformada em árvore de LLPP-l com $l = 1$.

Gera-se T'' a partir de T' transformando o nó interno vivo em um nó interno. Pelo Lema 6.3 temos que $S(T'') \leq S(T')$. Observe que, como M' duplicou todas as linhas de M, T'' tem pelo menos uma folha relacionada a cada linha de M.

Em seguida obtem-se T a partir de T'' retirando todas as folhas duplicadas, ou seja, se u e v são folhas rotuladas pela linha original e pela sua cópia, será eliminada uma destas folhas, bem como o nó interno pai dela. A Figura 6.2 mostra como seria a transformação de uma árvore T', resposta *sim* a $(M', 1, B)$ em LLPP-l, em outra árvore T, candidata a resposta *sim* a (M, B) em LPP.

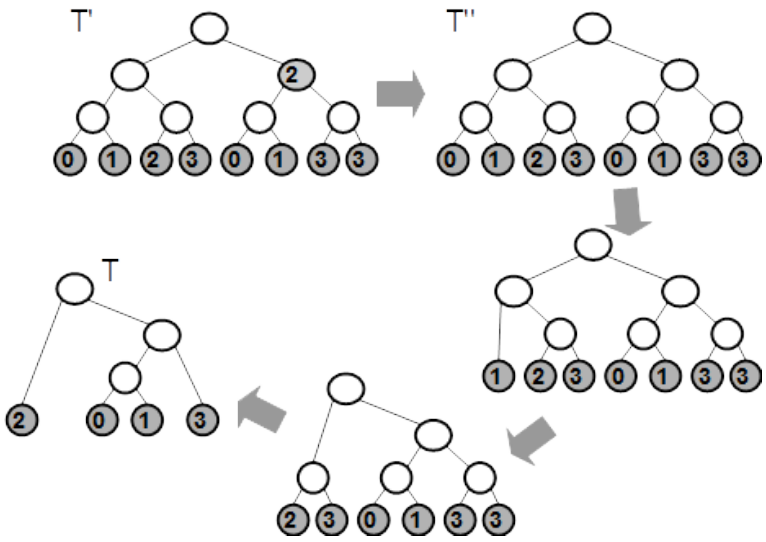


Figura 6.2: Árvore de LLPP-l com $l = 1$ transformada em árvore de LPP.

Como T'' tem pelo menos uma folha relacionada a cada linha de M e são tiradas apenas as folhas duplicadas, T também terá pelo menos uma folha relacionada a cada linha de M. Pelo Lema 6.2, temos $S(T) \leq S(T'')$. Como $S(T'') \leq S(T') \leq B$ temos $S(T) \leq B$, ou seja, T confere resposta *sim* a (M, B) em LPP.

Como LPP é reduzido a LLPP-l e LPP é NP-completo então LLPP-l é NP-completo.

□

Branch-and-Bound

No problema LLPP-l o espaço de busca consiste nas árvores com exatamente l nós internos vivos. O Algoritmo 5.2 (pg. 78) foi adaptado de forma a testar apenas árvores que têm ou podem ter l nós internos vivos. Esta adaptação ocorre da seguinte forma: antes de completar a árvore, conta o número de nós internos vivos e nós internos vivos pernetas e verifica se existem posições em aberto suficientes para obter l nós internos vivos. Caso não existam, aborta a construção. Além disto, caso a árvore em construção já possua l nós internos vivos, não vai permitir inserção de outras espécies como nós internos vivos.

Esta verificação foi implementada na função $GeraProxOrdemConstrucao$ para que gere apenas construções que tenham exatamente l nós internos vivos.

Como o Algoritmo 5.2 cobre todo o espaço de busca para árvores com um valor qualquer de nós internos vivos, cobre inclusive todo o espaço de busca de árvores com exatamente $l > 0$ nós internos vivos. Então, com o Algoritmo 5.2 restrito à análise das árvores com exatamente $l > 0$ nós internos vivos, é possível cobrir todo o espaço de busca para o problema LLPP-l.

Heurísticas

Como o problema LLPP-l é NP-completo busca-se a construção de heurísticas para resolver o problema. São propostas quatro heurísticas: duas de construção e duas de rearranjo.

Heurística de Construção 1: L-means Live.

Para o caso LLPP-l com $l \geq 2$ propõe-se o algoritmo de agrupamento k-means [38] para separar as espécies em subgrupos e sugerir os l nós internos vivos. Como neste caso utiliza-se $k = l$, esta heurística é chamada de *L-means Live*. Observe que algoritmos de agrupamento não fazem sentido para separar um conjunto em apenas um subconjunto, desta forma esta heurística serve apenas para $l \geq 2$.

K-means é um método de *Clustering* que objetiva particionar n objetos em k grupos, considerando uma função de distância. Inicialmente, são definidos aleatoriamente k centroides dentre os objetos a serem particionados. Em cada iteração, os objetos são agrupados ao centroide mais próximo. Em seguida, é calculado o ponto médio do grupo, o qual é definido como novo centroide e refaz-se o passo anterior. O algoritmo termina quando não há mais mudança nos grupos.

A heurística executará os seguintes passos para $l \geq 2$:
1) Separa em subgrupos de objetos com maior proximidade (utilizando k-means para $k = l$).
2) Caso algum dos subgrupos tenha apenas um ou dois objetos, serão incluídos objetos mais próximos de outros subgrupos até completarem 3 objetos, sendo recalculado o centroide.
3) Retira do subgrupo a espécie mais próxima do centroide e constrói árvore do subgrupo usando Adição Sequencial tradicional (Seção 3.1).
4) Modifica as árvores obtidas na etapa anterior tornando a espécie mais próxima do centroide na raiz da respectiva subárvore.
5) Constrói árvore tendo os nós internos vivos (raízes das subárvores obtidas de 1-4) como folhas, usando Adição Sequencial tradicional (Seção 3.1).
6) Substitui, na árvore construída em 5, os nós folha pelas subárvores dos quais eles são raízes.

O passo 2 é necessário para que cada subgrupo tenha uma raiz, construída no passo 4, sendo nó interno vivo, o que não é possível caso o subgrupo contenha apenas 1 ou 2 objetos.

A Figura 6.3 ilustra os passos 1 a 4 para um lista de 10 espécies e $k = 2$, onde os nós mais próximos dos centroides foram os nós 6 e 2, tornados raízes das subárvores. Na Figura 6.4 apresentamos a finalização da heurística, com a obtenção de uma árvore com 2 nós internos vivos.

O k-means possui alguns problemas: é sensível a *outliers* e ruído; o desempenho e os resultados do algoritmo dependem da escolha dos centroides iniciais. Além disto, esta heurística não permite a existência de nó interno vivo sendo ancestral de outro nó interno vivo.

Para o caso $l \geq 2$ a utilização de um método de *Clustering* fornece uma boa sugestão de separação das espécies próximas, representadas pelo centroide.

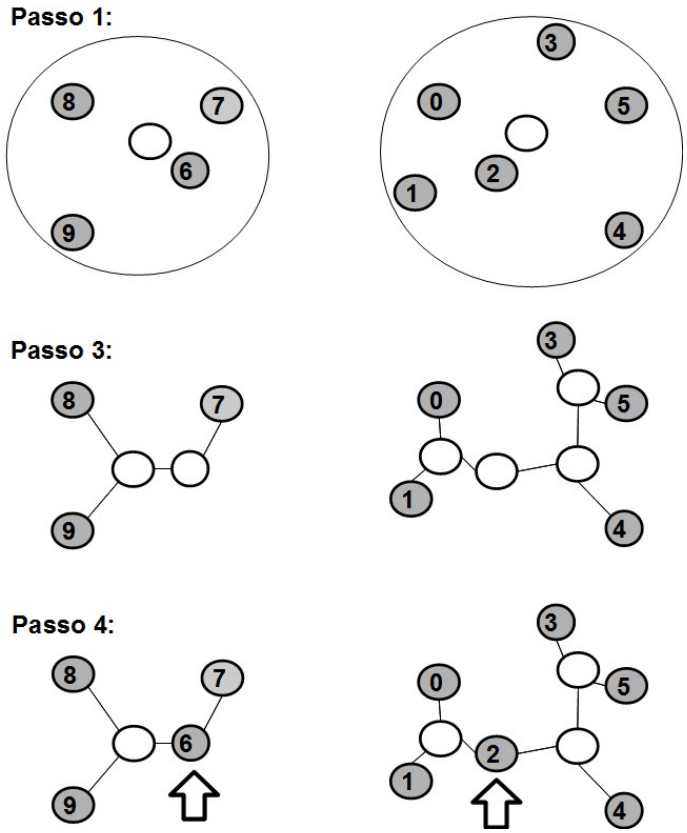


Figura 6.3: Aplicação da heurística para 10 espécies e $k = 2$ destacando as raízes das subárvores, em particular mostrando os passos 1, 3 e 4.

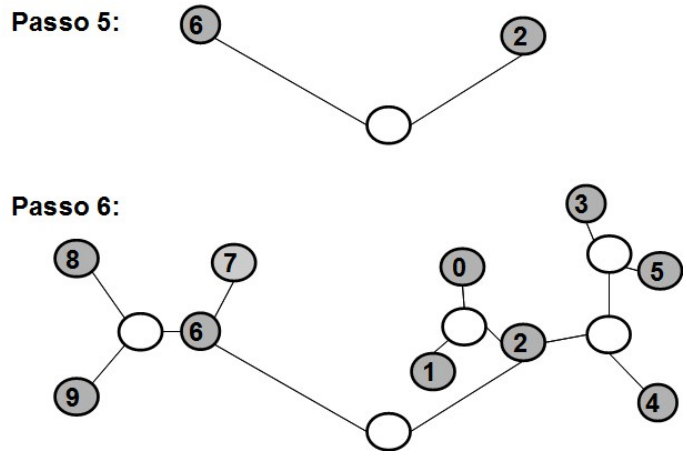


Figura 6.4: Finalização da aplicação da heurística com a obtenção de árvore com $l = 2$ nós internos vivos.

Heurística de Construção 2: Adição Sequencial Live para $l = 1$.

Como L-means Live não permite $l = 1$, é necessário uma heurística para LLPP-l com $l = 1$. Da mesma forma que na Adição Sequencial tradicional (Seção 3.1), será

definida uma ordem de inserção aleatória de espécies. A heurística irá pressupor que a última espécie será inserida na árvore como nó interno vivo. Observe que, como a ordem de inserção é aleatória, a escolha da espécie que será representada por nó interno vivo também será aleatória.

A heurística começa construindo uma árvore para as $n-1$ espécies que são folhas utilizando Adição Sequencial tradicional. Em seguida, a última espécie é inserida em um nó interno, transformando-o em nó interno vivo, na posição que gere a árvore com menor escore.

No caso tradicional, esta heurística produz bons resultados com complexidade de tempo razoável.

Heurística de Rearranjo 1: NNI Live para $l > 0$.

Observe que uma heurística de rearranjo para LLPP-l deve receber como entrada uma árvore com exatamente l nós internos vivos e testar outras árvores com exatamente l nós internos vivos de acordo com uma certa vizinhança.

Esta heurística é uma adaptação de NNI Live (Seção 5.3). Caso a aresta selecionada possua um nó interno vivo, será permitida apenas a operação de permuta de ramos (operação de NNI tradicional), pois esta operação não altera o número de nós internos vivos. Caso a aresta selecionada não possua nó interno vivo, também pode ser realizada a promoção, entretanto esta deve ser seguida de operação de transformação de um nó interno vivo em folha de forma a manter o mesmo número de nós internos vivos. Esta operação será chamada de *despromoção* e consiste na criação de um nó interno e de um nó folha. O nó interno criado é tornado pai do nó interno vivo e da nova folha. Finalmente, o nó interno vivo é transformado em nó interno hipotético e a espécie a ele associada é associada à nova folha.

A Figura 6.5 mostra a operação de despromoção aplicada ao nó interno vivo x na árvore da Figura 6.5(a), gerando a árvore onde x é folha (Figura 6.5(b)).

Na heurística, quando for necessária a operação de despromoção, esta será testada sobre todos os nós internos vivos, exceto o que foi recém promovido, e a árvore com menor escore será mantida.

Heurística de Rearranjo 2: TBR Live para $l > 0$.

A heurística *TBR Live para* $l > 0$ é uma adaptação da TBR Live (Seção 5.3) feita da

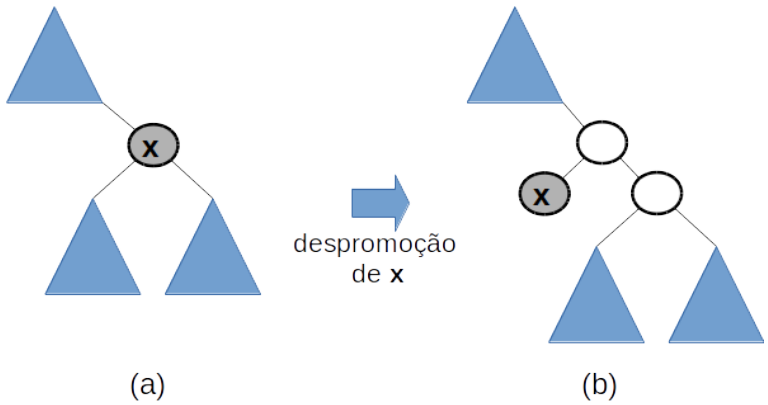


Figura 6.5: Aplicação da operação de despromoção sobre o nó interno vivo x em (a), gerando árvore onde x é folha (b).

mesma forma que a adaptação proposta na heurística anterior. Quando a reconexão ocorrer em uma folha (transformando-a em nó interno vivo) será seguida de operação de despromoção, visando garantir que a árvore a ser testada tenha exatamente l nós internos vivos.

Da mesma forma que em TBR Live são propostas duas otimizações na execução da heurística: arestas ponto fraco e subárvore forte.

6.2 Nós internos vivos definidos

O problema geral (LLPP) permite a presença, ou não, de nós internos vivos em um número qualquer na árvore solução. Caso seja conhecido que a solução proposta deva conter $l > 0$ nós internos vivos rotulados pelas linhas $o'_1, ..., o'_l$ da matriz M, têm-se um outro problema. Adaptando a Definição 5.1, têm-se a definição abaixo.

Definição 6.5. *Problema da Parcimônia Grande Viva l-definido (LLPP-l-def)*
Instância: Uma matriz $M_{n\times m}$, um inteiro $l > 0$, um subconjunto de linhas $\{o'_1, ..., o'_l\}$ da matriz M e uma constante $B \in \mathbb{R}_+$.
Questão: Existe uma árvore T, completamente rotulada, com l nós internos vivos e $n - l$ folhas, sendo os l nós internos vivos rotulados pelas linhas $o'_1, ..., o'_l$ da matriz M e as $n - l$ folhas rotuladas pelas demais linhas da matriz M, tal que $S(T) \leq B$?

Seguindo o raciocínio desenvolvido para LLPP-l, é possível calcular o número de possíveis árvores para LLPP-l-def, basta retirar o primeiro fator (relativo às va-

riações possíveis de l nós internos vivos) da Fórmula 6.3. O total de possíveis árvores filogenéticas com n objetos e l nós internos vivos definidos é dado pela Fórmula 6.4.

$$T_n^l = (2(n-l)-3)!!\frac{(n-l-1)!}{(n-2l-1)!} \tag{6.4}$$

NP-completude

Prova-se que LLPP-l-def é NP-completo reduzindo o problema LPP a ele.

Teorema 6.6. *LLPP-l-def é NP-completo.*

Demonstração. Inicialmente observe que a solução polinomial obtida para SLPP garante que uma instância qualquer $(M', l, \{o'_1, ..., o'_l\}, B)$ de LLPP-l-def seja verificada em tempo polinomial, ou seja, LLPP-l-def está em NP.

Para completar a prova, é necessário reduzir um problema NP-completo a LLPP-l-def. Faremos isso reduzindo LPP a LLPP-l-def.

Seja (M, B) uma instância de LPP, em que M é uma matriz $n \times m$ e $B \in \mathbb{R}_+$.

Seja o_x uma linha qualquer de M. Obtém-se M' acrescentando as linhas o_{x1} e o_{x2} com estados iguais aos de o_x, ou seja, M' é obtida de M triplicando o_x.

Seja $(M', 1, \{o_x\}, B)$ a instância de LLPP-l-def.

É necessário provar que a redução é válida. Prova-se que uma resposta *sim* a (M, B) em LPP implica em uma resposta *sim* a $(M', 1, \{o_x\}, B)$ em LLPP-l-def e que uma resposta *sim* a $(M', 1, \{o_x\}, B)$ em LLPP-l-def implica em uma resposta *sim* a (M, B) em LPP.

Se a resposta a (M, B) em LPP for *sim* então existe T árvore que satisfaz (M, B) em LPP e é construída T' inserindo o_{x1} e o_{x2} como folhas de o_x em T. A Figura 6.6 mostra como seria a aplicação desta transformação caso $n = 4$ e $o_x = 3$. A árvore T (Figura 6.6(a)) é transformada numa árvore T' (Figura 6.6(b)) com exatamente um nó interno vivo o_x.

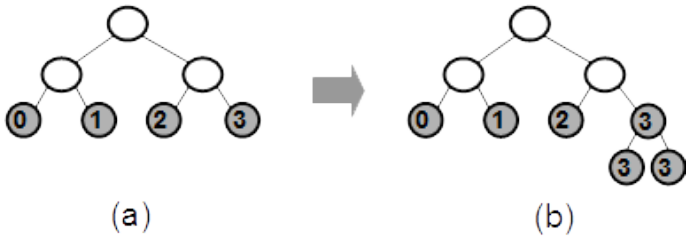


Figura 6.6: Árvore de LPP transformada em árvore de LLPP-l-def com $l = 1$ e $o_x = 3$.

Desta forma T' possui exatamente um nó interno vivo rotulado por o_x e é tal que $S(T') = S(T) \leq B$, ou seja, T' confere resposta *sim* a $(M', 1, \{o_x\}, B)$ em LLPP-l-def.

Se a resposta a $(M', 1, \{o_x\}, B)$ em LLPP-l-def for *sim* então existe T' com $S(T') \leq B$ e exatamente um nó interno vivo o_x. T é obtida a partir de T' transformando o_x em nó interno comum, o_{x1} em o_x e retirando o_{x2} e seu nó interno adjacente. A Figura 6.7 mostra um exemplo desta transformação caso $n = 4$ e $o_x = 3$.

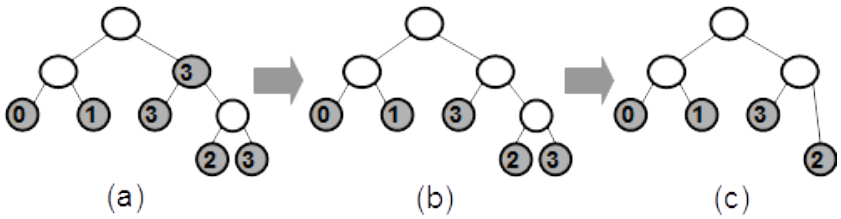


Figura 6.7: Árvore de LLPP-l-def com $l = 1$ e $o_x = 3$ transformada em árvore de LPP.

Pelos Lemas 6.2 e 6.3, $S(T) \leq S(T') \leq B$, ou seja, T confere resposta *sim* a (M, B) em LPP.

□

Branch-and-Bound

No problema LLPP-l-def o espaço de busca consiste nas árvores com exatamente l nós internos vivos rotulados pelas linhas $o'_1, ..., o'_l$ da matriz M. Novamente a solução é obtida adaptando o Algoritmo 5.2 fazendo uso desta restrição importante.

Pode-se pensar na construção da árvore de duas formas bastante diferentes. Na primeira, iniciando a montagem da árvore com os objetos que serão folhas e finalizando com a transformação dos nós internos em nós internos vivos. Esta abordagem é ilustrada na Figura 6.8 para $l = 3$ e A, B, C nós internos vivos. Na segunda possibilidade, atuando de forma inversa, montando a subárvore contendo apenas os nós rotulados pelas linhas $o'_1, ..., o'_l$, que serão nós internos vivos na árvore completa, e depois completando a árvore com as folhas. A Figura 6.9 mostra esta abordagem para $l = 3$ e A, B, C nós internos vivos.

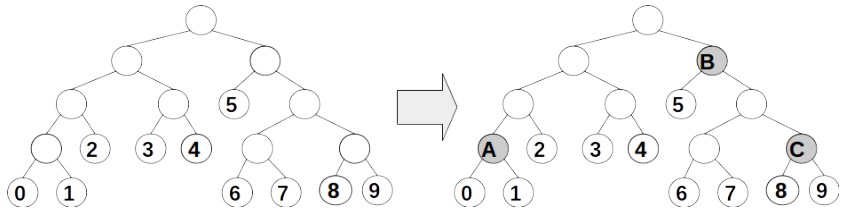


Figura 6.8: Construção iniciando com folhas e completando com nós internos vivos.

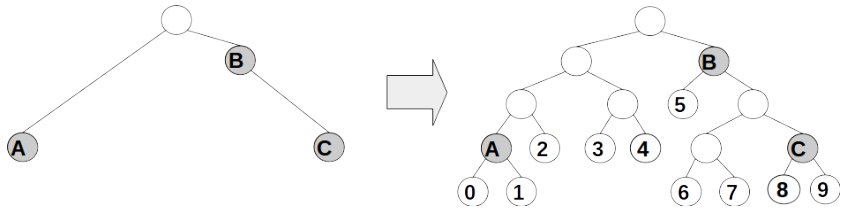


Figura 6.9: Construção iniciando com nós internos vivos e completando com folhas.

Ambas as estratégias podem ser implementadas a partir do Algoritmo 5.2, alterando a ordem de inserção das espécies e restringindo as operações.

Na primeira abordagem, testa-se todas as possibilidades, inserindo primeiro as espécies que serão folhas e não permitindo operações que gerem nós internos vivos ou nós internos vivos pernetas, em seguida inserindo as espécies que serão nós internos vivos utilizando sempre a operação *PromoLive*.

Na segunda abordagem, testa-se todas as possibilidades, inserindo primeiro as espécies que serão nós internos vivos. Observe que nesta etapa são permitidas, inclusive, operações que gerem nós internos pernetas. Em seguida, são inseridas as espécies que serão folhas, não permitindo a operação *PromoLive* e nem a *GeraOneLeg* quando realiza a inserção de nó interno vivo perneta em uma aresta. Este tipo de operação *GeraOneLeg* não é permitido pois geraria nós internos vivos dentre os últimos a serem inseridos, ou seja, dos que devem ser folhas. Mesmo o

outro tipo de *GeraOneLeg*, o que liga nó em uma folha transformando-a em nó interno perneta, só pode ser realizado sobre os nós que serão nós internos vivos.

Por questão de simplicidade, foi adotada a primeira abordagem. É fácil verificar que o *branch-and-bound* obtido por esta modificação no algoritmo cobre todo o espaço de busca. Suponha que T seja a melhor árvore com $n-l$ nós folhas e l nós internos vivos. O Algoritmo 5.2, retirando-se as operações que geram nós internos vivos ou nós internos vivos pernetas, comporta-se exatamente como o *branch-and-bound* criado por Hendy e Penny [27], o qual cobre todo o espaço de busca. Então, o *branch-and-bound* cobre todas as possíveis árvores contendo os $n-l$ nós folhas, dentre elas a árvore correspondente à T sem nós internos vivos. Em seguida, analisa-se todas as possíveis inserções dos l nós internos vivos, em particular a inserção que gera T. Então o *branch-and-bound* retornará T ou outra árvore com escore igual ao de T.

Heurísticas

Propomos aqui três ideias de heurísticas para LLPP-l-def: uma de construção e duas de rearranjo.

Como heurística de construção, especificamente Adição Sequencial Live para l definido, propõe-se uma heurística que irá funcionar de forma semelhante ao que foi feito no *branch-and-bound*. Altera-se a ordem de inserção das espécies de forma que as l espécies correspondentes aos nós internos vivos são inseridas ao final. Desta forma, a heurística começa construindo uma árvore para as $n-l$ espécies que são folhas utilizando Adição Sequencial tradicional (Seção 3.1). Em seguida, são inseridas as l espécies apenas testando sua colocação em nós internos, transformando-os em nós internos vivos, de forma a minimizar o escore parcimônia.

Nas heurísticas de rearranjo para LLPP-l-def, a entrada deve ser uma árvore com exatamente l nós internos vivos rotulados pelas linhas $o'_1, ..., o'_l$ da matriz M e a saída uma árvore com os mesmos nós internos vivos. Assim, a heurística pode alterar a posição dos nós internos vivos na árvore, mas não pode torná-los folha e nem transformar outra folha em nó interno vivo. Desta forma, qualquer das heurísticas de rearranjo tradicionais, vistas na Seção 3.2, podem ser aplicadas para LLPP-l-def, desde que não sejam permitidas operações que eliminem nó interno vivo. São propostas as heurísticas NNI e TBR tradicionais (vistas na Seção 3.2). No caso da

TBR, não será permitida, na fase bisseção, a eliminação de aresta com nó interno vivo adjacente.

Capítulo 7

Resultados

Para analisar o desempenho das heurísticas foram realizadas quatro baterias de testes.

Para as três primeiras baterias foram utilizados *benchmarks* apresentados no trabalho de Andreatta e Ribeiro [1]. São 8 instâncias de testes baseadas em casos reais. Cada instância, com seu número de espécies e características, é apresentada na Tabela 7.1. Estes *benchmarks* possuem características com estado iguais a 0, 1 ou ? (indefinido). A função δ_{ij} é a função distância de Hamming.

Tabela 7.1: *Benchmarks* utilizados.

Instância	Espécies	Características
GRIS	47	93
ANGI	49	59
TENU	56	179
ETHE	58	86
ROPA	75	82
GOLO	77	97
SCHU	113	146
CARP	117	110

Antes da realização das baterias, foram feitos testes para o ajuste de parâmetros na execução das heurísticas de rearranjo. Observando a convergência dos escores obtidos ao longo da execução destas heurísticas, foi definido em 10000 o limite máximo de repetições da operação principal da heurística. Além disto, o número de árvores guardadas durante a execução destas heurísticas foi ajustado em 20 e o

número de execuções nas heurísticas com alguma escolha aleatória (NNI, NNI Live, TBR, TBR Live e TBR Live variação Aresta Ponto Fraco) foi definido em 50 vezes.

Na primeira bateria de testes, foi abordado o problema LLPP. Nesta bateria buscou-se confrontar escores obtidos pelas heurísticas com os escores ótimos obtidos a partir do *branch-and-bound* para LLPP. Os testes realizados no *branch-and-bound* não apresentaram tempos aceitáveis a partir de exemplos com mais de 11 espécies. Desta forma, foram geradas instâncias baseadas nos *benchmarks* com no máximo 11 espécies. Isto tornou possível, para cada instância, executar o *branch-and-bound* e a heurística e então comparar os escores obtidos. Como forma de dar maior confiabilidade aos resultados obtidos nestas comparações, para cada *benchmark* foi gerado um conjunto de 100 instâncias com 11 espécies escolhidas aleatoriamente.

Na segunda bateria de testes, foram abordados os problemas LLPP-l e LLPP-l-def. Novamente foram comparados os escores obtidos pelas heurísticas e os escores ótimos obtidos a partir dos *branch-and-bound*, neste caso para LLPP-l e LLPP-l-def. Foi utilizado um subconjunto das instâncias da primeira bateria. Dado o número de espécies ser 11, foram feitos testes considerando o número de nós internos vivos variando de 1 a 3. Como o número de execuções do *branch-and-bound* em cada instância aumentou, o número de instâncias testadas, para cada *benchmark*, foi reduzido a 40.

Na terceira bateria, buscou-se comparar o desempenho das heurísticas em casos onde a quantidade de espécies fosse maior. Por conta disto, ficou inviável a aplicação do *branch-and-bound* e foram feitas comparações apenas entre as heurísticas. Para tanto foram utilizados os 8 *benchmarks* sem limitar a quantidade de espécies.

A quarta bateria de testes foi dedicada à análise de um caso real, parcialmente proposto na publicação [26]. Foram utilizados dados do vírus Zika, originalmente apresentados por Lanciotti e colegas [34], com um conjunto de 20 espécies. Nesta bateria buscou-se comparar a filogenia proposta por Lanciotti com as árvores sugeridas pelas heurísticas. Foram executadas heurísticas dos três problemas. Em virtude do tamanho da representação de cada espécie não foi executado *branch-and-bound*.

A Tabela 7.2 sintetiza as informações sobre os testes realizados.

Tabela 7.2: Quadro geral das baterias de testes.

	Baterias			
	1	2	3	4
Origem dos dados	Benchmarks	Benchmarks	Benchmarks	Zika
Quantidade de instâncias	100	40	8	1
Quantidade de espécies	11	11	47-117	20
Quantidade de características	59-179	59-179	59-179	2162
Problemas	LLPP	LLPP-l LLPP-l-def	LLPP LLPP-l LLPP-l-def	LLPP LLPP-l LLPP-l-def
Variações	-	l=1 l=2 l=3	l=1 l=2 l=3	l=1 l=2 l=3 l=4 l=5
Comparações	Branch&Bound Heurísticas - Construtivas - Rearranjo	Branch&Bound Heurísticas - Construtivas - Rearranjo	Heurísticas - Construtivas - Rearranjo	Heurísticas - Construtivas - Rearranjo

7.1 Testes usando *Benchmarks*

Primeira bateria de testes

A primeira bateria de testes teve por base o problema LLPP. Para cada instância foram executados: o *branch-and-bound* e as heurísticas de Adição Sequencial Live, NNI Live e TBR Live com suas variações Aresta Ponto Fraco e Subárvore Forte, definidas na Seção 5.3.

Uma primeira análise buscou identificar a qualidade da heurística construtiva Adição Sequencial Live na geração de árvores com escore igual ou próximo do escore ótimo obtido pelo *branch-and-bound*, assim como a quantidade de árvores geradas com nó interno vivo. Para ilustrar melhor o comportamento da heurística foram analisadas as árvores resultantes, se continham ou não nós internos vivos. Os resultados foram tabulados separadamente. A Tabela 7.3 resume os resultados obtidos. A coluna quantidade informa a quantidade de árvores geradas em cada caso e no total. Já a coluna "Obteve MPT" identifica quantas destas árvores têm escore igual à obtida

pelo *branch-and-bound*, ou seja, podem ser chamadas árvores mais parcimoniosas. A coluna "% de MPT" indica o percentual de árvores mais parcimoniosas em relação à quantidade de árvores do caso correspondente. Finalmente, a coluna "Erro relativo" apresenta o percentual de acréscimo médio nas demais situações, ou seja, qual o erro médio cometido nos casos em que a heurística não obteve uma árvore mais parcimoniosa.

Tabela 7.3: LLPP: Adição Sequencial Live comparada com *branch-and-bound*.

	Quantidade	Obteve MPT	% de MPT	Erro Relativo
Árvores com Vivos	109	35	32,11%	2,89%
Árvores sem Vivos	691	210	30,39%	2,37%
Total	800	245	30,63%	2,41%

Os testes mostraram que a heurística construtiva gera resultados bastante próximos aos ótimos, com uma boa quantidade de acertos. O erro relativo médio fica sempre abaixo dos 3%, com desvio padrão de 1,71%. A grande maioria das árvores geradas não possuem nós internos vivos.

Como a Adição Sequencial Live é uma heurística gulosa, uma vez inserido um nó interno vivo, não se analisa a possibilidade de despromoção deste nó. Tal funcionamento poderia sugerir que as árvores obtidas com nó interno vivo geram escores piores. Tal hipótese é descartada pela Tabela 7.3, uma vez que a distribuição de árvores mais parcimoniosas em todos os casos é muito próxima, com leve vantagem para o caso de árvores com nós internos vivos. Por outro lado, o erro relativo médio é inferior no caso sem nós internos vivos.

É intessante observar que mais da metade das árvores com nós internos vivos, um total de 58, ocorrem nas instâncias geradas a partir do *benchmark* GRIS, que contém a maior quantidade de estados ? (indefinido). Nesse *benchmark*, aproximadamente 43,3% do total de estados são ? (indefinido), sendo que nos demais casos não passa de 20%. Isto sugere que a existência destes estados favorece a obtenção de árvores com nós internos vivos e escore baixo.

Mudando agora o foco da análise para as heurísticas de rearranjo, essas iniciam com uma árvore qualquer e buscam obter árvore com melhor escore. Desta forma, o desempenho da heurística de rearranjo pode ficar distorcido por conta da heurística construtiva. Até que ponto a árvore inicial influencia no resultado obtido é o que

se busca analisar. Foram realizadas duas sequências de testes: uma iniciando com a árvore fornecida pela heurística Adição Sequencial Live e a outra iniciando com uma árvore do tipo caterpillar.

A Tabela 7.4 resume os resultados obtidos quando comparadas as heurísticas em relação aos escores ótimos obtidos pelo *branch-and-bound*, considerando duas árvores iniciais diferentes (Adição Sequencial Live e caterpillar). Novamente, a coluna "Obteve MPT" reporta quantas vezes, em um total de 800 instâncias, a heurística obteve árvore com escore igual ao obtido pelo *branch-and-bound*. A coluna "Erro relativo" apresenta o percentual de acréscimo médio nas demais situações, ou seja, qual o erro médio cometido nos casos em que a heurística não obteve uma árvore mais parcimoniosa.

Tabela 7.4: LLPP: Heurísticas de rearranjo comparadas com *branch-and-bound* dadas as árvores iniciais Adição Sequencial Live e caterpillar.

	Adição Sequencial Live			Caterpillar		
	Obteve MPT	% de MPT	Erro Relativo	Obteve MPT	% de MPT	Erro Relativo
NNI Live	451	56,38%	2,01%	264	33,00%	8,46%
TBR Live	259	32,38%	2,37%	0	0%	9,08%
TBR Live Aresta Ponto Fraco	260	32,50%	2,37%	0	0%	11,15%
TBR Live Subárvore Forte	443	55,38%	1,93%	67	8,38%	11,88%

Conforme a Tabela 7.3, a heurística Adição Sequencial Live obteve uma árvore mais parcimoniosa 245 vezes. Desta forma todas as heurísticas de rearranjo com árvore inicial fornecida por Adição Sequencial Live terão valores acima de 245 na coluna obteve MPT da Tabela 7.4.

Na Tabela 7.4 comparamos a execução de heurísticas de rearranjo para as mesmas instâncias, porém com árvores iniciais diferentes. Buscamos saber o quanto impacta na heurística de rearranjo iniciar com uma árvore qualquer, na qual não se buscou um melhor escore, que é o caso de uma árvore Caterpillar, ou iniciar com uma árvore com escore mais baixo, obtida por uma heurística construtiva, no caso Adição Sequencial Live. Os resultados mostram que as heurísticas têm desempenho fortemente afetado pela árvore inicial fornecida, tanto na quantidade de árvores mais parcimoniosas obtidas quanto no erro relativo das demais. Cabe ressaltar que TBR Live variação Subárvore Forte é a que mais é afetada.

Tendo como base o escore obtido na heurística construtiva é interessante observar quando e quanto as heurísticas de rearranjo conseguem melhorar este escore. Esta informação é apresentada na Tabela 7.5, na qual estão as comparações entre as heurísticas de rearranjo e a heurística construtiva. A coluna "melhorou adição sequencial" indica quantas vezes, em um total de 800 instâncias, a heurística conseguiu melhorar o resultado obtido na heurística Adição Sequencial Live. A coluna "melhoria relativa" indica o percentual médio de diminuição do escore obtido nestes casos.

Tabela 7.5: LLPP: Heurísticas de rearranjo comparadas com Adição Sequencial Live.

	Melhorou Adição Sequencial Live	Melhoria relativa
NNI Live	301	0,75%
TBR Live	32	0,07%
TBR Live Aresta Ponto Fraco	32	0,07%
TBR Live Subárvore Forte	326	0,78%

Os dados apontam NNI Live e TBR versão Subárvore Forte como as que mais vezes conseguem melhorar as árvores obtidas da Adição Sequencial Live, com uma leve vantagem de TBR Live versão Subárvore Forte nos escores obtidos.

Segunda bateria de testes

A segunda bateria de testes teve por base os problemas LLPP-l e LLPP-l-def. Como nestes problemas têm-se um valor definido de nós internos vivos, os testes foram feitos considerando o número de nós internos vivos variando de 1 a 3. Em LLPP-l-def a escolha das espécies que irão rotular os l nós internos vivos foi aleatória.

Foram utilizadas 40 instâncias em cada *benchmark* para cada l, totalizando 960 instâncias.

Para LLPP-l foram executados: o *branch-and-bound*; as heurísticas construtivas Adição Sequencial Live para $l = 1$ ou L-means Live quando $l \geq 2$; e as heurísticas de rearranjo NNI Live e TBR Live para $l > 0$ com suas variações Aresta Ponto Fraco e Subárvore Forte, definidas na Seção 6.1.

Para LLPP-l-def foram executados: o *branch-and-bound*; a heurística construtiva de Adição Sequencial Live para l definido; e as heurísticas de rearranjo NNI e TBR com suas variações Aresta Ponto Fraco e Subárvore Forte, definidas na Seção 6.2.

Cabe lembrar que, no caso LLPP-l-def, as heurísticas de rearranjo são NNI e TBR tradicionais pois a árvore construída já deve possuir os nós internos vivos definidos pelo problema.

A Tabela 7.6 resume os resultados obtidos quando comparadas as heurísticas em relação aos escores ótimos obtidos pelo *branch-and-bound* no problema LLPP-l. A coluna "obteve MPT" reporta quantas vezes, em um total de 960 instâncias, a heurística obteve árvore com escore igual à obtida pelo *branch-and-bound*. Novamente a coluna erro relativo apresenta o percentual de acréscimo médio nas demais situações.

Tabela 7.6: LLPP-l: Heurísticas comparadas com *branch-and-bound*.

	Obteve MPT	% de MPT	Erro Relativo
Adição Sequencial Live para $l = 1$ ou L-means Live	0	0%	27,21%
NNI Live	18	1,88%	9,60%
TBR Live	0	0%	14,10%
TBR Live Aresta Ponto Fraco	0	0%	17,32%
TBR Live Subárvore Forte	2	0,20%	16,37%

Observa-se que o erro relativo da heurística construtiva é grande. Quando separados os casos em que é aplicada Adição Sequencial Live para $l = 1$ daqueles em que é aplicada L-means Live ($l > 1$), fica evidente um desempenho inferior da segunda. A Tabela 7.7 detalha melhor os resultados da Tabela 7.6 separando os dados pela quantidade de nós internos vivos e, consequentemente, pela heurística construtiva utilizada.

O erro relativo das heurísticas construtivas é grande e, embora haja melhoria, o erro relativo das heurísticas de rearranjo também é grande. Isso corrobora com a análise realizada a partir dos dados da Tabela 7.4, o desempenho das heurísticas de rearranjo é fortemente afetado pelo das heurísticas construtivas.

Com um erro relativo grande por parte da heurística construtiva, é esperado que as heurísticas de rearranjo melhorem o escore na maioria dos casos, o que é mostrado

Tabela 7.7: LLPP-l: Heurísticas comparadas com *branch-and-bound* separado por quantidade de vivos.

	Vivos	Obteve MPT	Erro Relativo
Adição Sequencial Live para $l = 1$	1	0	19,84%
L-means Live	2	0	28,73%
	3	0	33,07%
NNI Live	1	17	3,91%
	2	1	9,29%
	3	0	15,29%
TBR Live	1	0	4,68%
	2	0	12,21%
	3	0	25,40%
TBR Live Aresta Ponto Fraco	1	0	6,49%
	2	0	18,18%
	3	0	27,28%
TBR Live Subárvore Forte	1	0	15,94%
	2	1	12,73%
	3	1	20,44%

na Tabela 7.8. Destaca-se o desempenho de NNI Live e TBR Live. A execução das heurísticas de rearranjo diminui consideravelmente o erro, entretanto não se aproxima do desempenho obtido pelas heurísticas para LLPP na primeira bateria de testes.

Tabela 7.8: LLPP-l: Heurísticas de rearranjo comparadas com Adição Sequencial Live para $l = 1$ e L-means Live.

	Melhorou Adição Sequencial Live para $l = 1$ ou L-means Live	Percentual melhoria
NNI Live	937	13,42%
TBR Live	841	10,02%
TBR Live Aresta Ponto Fraco	744	7,54%
TBR Live Subárvore Forte	794	8,19%

A Tabela 7.9 resume os resultados obtidos quando comparadas as heurísticas em relação aos escores ótimos obtidos pelo *branch-and-bound* no problema LLPP-l-def. A coluna "obteve MPT" reporta quantas vezes, em um total de 960 instâncias, a heurística obteve árvore com escore igual à obtida pelo *branch-and-bound*, ou seja, a heurística obteve uma árvore mais parcimoniosa. Novamente a coluna erro relativo apresenta o percentual de acréscimo médio nas demais situações.

Tabela 7.9: LLPP-l-def: Heurísticas comparadas com *branch-and-bound*.

	Obteve MPT	% de MPT	Erro Relativo
Adição Sequencial Live para l definido	16	1,67%	9,70%
NNI	39	4,06%	7,70%
TBR	16	1,67%	8,48%
TBR Aresta Ponto Fraco	19	1,98%	9,01%
TBR Subárvore Forte	76	7,92%	8,00%

O erro relativo, assim como a quantidade de acertos, da heurística Adição Sequencial para l definido são bem melhores que os obtidos pelas heurísticas construtivas de LLPP-l. Da mesma maneira, todos os demais indicadores das heurísticas de rearranjo de LLPP-l-def são melhores do que os obtidos em LLPP-l, com destaque para TBR variação Subárvore Forte. Em alguns casos, esta melhoria é consequência de uma melhor árvore obtida na heurística construtiva.

Neste caso não houve diferença significativa no desempenho das heurísticas conforme a quantidade de nós internos vivos.

A Tabela 7.10 mostra que, apesar da obtenção de melhores árvores pelas heurísticas de rearranjo ser bastante frequente, o percentual de melhoria é pequeno.

Tabela 7.10: LLPP-l-def: Heurísticas de rearranjo comparadas com Adição Sequencial Live para l definido.

	Melhorou Adição Sequencial Live para l definido	Percentual melhoria
NNI Live	592	1,90%
TBR Live	630	1,07%
TBR Live Aresta Ponto Fraco	281	0,63%
TBR Live Subárvore Forte	476	1,93%

Terceira bateria de testes

Na terceira bateria de testes foram feitas comparações apenas entre as heurísticas, a partir das árvores obtidas pelas heurísticas construtivas. Para tanto, foram utilizados os 8 *benchmarks* desta vez sem limitar a quantidade de espécies.

No problema LLPP, NNI Live sempre melhorou as árvores obtidas pela Adição Sequencial Live, ao passo que TBR Live melhorou 75%, TBR Live variação Aresta Ponto Fraco melhorou 50% e TBR Live variação Subárvore Forte apenas 25% das árvores obtidas.

No problema LLPP-l, todas as heurísticas melhoraram as árvores obtidas pelas heurísticas construtivas.

Em LLPP-l-def, as heurísticas NNI, TBR e TBR variação Subárvore Forte sempre conseguiram melhorar as árvores, mas TBR variação Aresta Ponto Fraco não conseguiu melhorar em 54,17% dos casos.

A melhoria das heurísticas de rearranjo em relação as heurísticas construtivas é mostrada na Tabela 7.11.

Tabela 7.11: Bateria 3: Percentual de melhoria das heurísticas de rearranjo em relação às soluções apresentadas pelas heurísticas construtivas.

	Vivos	NNI ou NNI Live	TBR ou TBR Live	TBR Aresta P. Fraco ou TBR Live Aresta P. Fraco	TBR Subárv. Forte ou TBR Live Subárv. Forte
LLPP	-	0,48%	0,01%	0,02%	0,06%
LLPP-l	1	3,07%	1,57%	0,92%	1,07%
	2	7,51%	3,90%	2,91%	3,31%
	3	9,34%	4,87%	4,58%	3,78%
LLPP-l-def	1	1,47%	0,26%	0,21%	1,07%
	2	1,56%	0,25%	0,28%	1,37%
	3	1,82%	0,25%	0,13%	1,24%

Observa-se que nos problemas LLPP e LLPP-l-def, apesar das heurísticas quase sempre melhorarem as árvores, o percentual de melhoria é baixo, principalmente no problema LLPP. Possivelmente por conta da árvore obtida pela heurística contrutiva já possuir um escore baixo.

Já no caso LLPP-l os percentuais de melhoria são altos, provavelmente por conta de uma solução ruim obtida pela heurística construtiva.

7.2 Testes usando Virus Zika

Vírus RNA consistem em um bom exemplo de espécies com alto grau de mutações [18]. Devido a isso, podem evoluir e co-existir ao mesmo tempo. Desta forma são bons candidatos à aplicação de heurísticas para LLPP, LLPP-l e LLPP-l-def.

Muitas ocorrências do vírus Zika têm sido reportadas recentemente na América e África. Uma grande importância tem sido dada a este vírus por conta de diversos problemas de saúde associados a ele. Por conta disso, nesta última bateria de testes, com utilização de dados reais, foram usados dados destes vírus. Parte do que é apresentado aqui foi proposto em [26].

Quarta bateria de testes

Foram utilizados dados do vírus Zika, apresentados por Lanciotti e colegas [34]. Nesta bateria foram executadas as heurísticas para LLPP, LLPP-l e LLPP-l-def.

Os dados utilizados foram de 20 sequências genéticas de vírus Zika obtidas do GenBank. Estas foram alinhadas usando MUSCLE [10], resultando em 10.109 colunas. Após o alinhamento, foi usado GBLOCKS [5] para remover colunas não informativas, resultando num alinhamento menor com 2.169 colunas. Cada coluna foi considerada uma característica para realização desta bateria de testes.

Além da análise dos escores obtidos, para algumas árvores foi verificado se as mesmas respeitam os três diferentes clados sugeridos por Lanciotti e colegas [34] para o vírus Zika: Ásia, Africa Ocidental e África Oriental.

Para LLPP-l-def, foi assumido que as espécies mais antigas são os ancestrais vivos. Ou seja, para $l = 1$ o ancestral foi Uganda47; para $l = 2$ foram Uganda47 e Malasia66; para $l = 3$ foram Uganda47, Malasia66 e Senegal68; para $l = 4$ foram Uganda47, Malasia66, Senegal68 e RCA68; para $l = 5$ foram Uganda47, Malasia66, Senegal68, RCA68 e Nigeria68.

No caso LLPP, nenhuma das heurísticas de rearranjo conseguiu melhorar o escore obtido pela heurística Adição Sequencial Live.

No caso LLPP-l, quase todas as árvores foram melhoradas, com exceção da obtida para $l = 1$ quando executada a heurística TBR Live variação Subárvore Forte.

No caso LLPP-l-def, na maioria das vezes não foi possível melhorar a árvore.

Os escores obtidos pelas heurísticas são apresentados na Tabela 7.12.

Tabela 7.12: Bateria 4: Escores obtidos para o conjunto de vírus Zika.

	Vivos	Ad.Seq.Live ou L-means Live	NNI ou NNI-Live	TBR ou TBR-Live	TBR Ar.P.Fraco ou TBR-Live Ar.P.Fraco	TBR Sub.Forte ou TBR-Live Sub.Forte
LLPP	-	3196	3196	3196	3196	3196
LLPP-l	1	3267	3209	3218,7	3225,4	3267
	2	4830	3431	3476,5	3407,9	3526
	3	4845	3342	3339,1	3543,4	3485
	4	4539	3675	3510,3	3614,4	4012
	5	3881	3580	3432,5	3578,8	3607
LLPP-l-def	1	3267	3267	3267	3267	3267
	2	3701	3701	3701	3701	3701
	3	3786	3783	3783	3783	3786
	4	4085	4085	4085	4085	4085
	5	4823	4692	4823	4823	4823

Observe que, como era esperado, os melhores escores obtidos para LLPP são melhores que os obtidos para LLPP-l, os quais são melhores que os para LLPP-l-def.

Em especial, para o caso LLPP-l-def, os valores aumentam bastante quando definidos, como nós internos vivos, as espécies Malasia66 (na passagem de $l = 1$ para $l = 2$) e Nigeria68 (na passagem de $l = 4$ para $l = 5$) sugerindo fortemente que estas espécies não devam ser ancestrais vivos na filogenia viva construída.

A seguir duas das melhores árvores resultantes da aplicação de heurísticas de rearranjo em LLPP-l. A da Figura 7.1 foi obtida pela aplicação da heurística TBR Live em LLPP-l com $l = 3$, tendo escore 3300. A da Figura 7.2 derivou da heurística TBR Live Aresta Ponto Fraco em LLPP-l com $l = 4$, obtendo escore 3410.

Ambas as árvores concordam com a formação de clados proposta por Lanciotti e colegas [34] e demonstram o potencial da filogenia viva em apresentar árvores com tamanho menor e com uma interpretação biológica mais adequada. Em especial, a

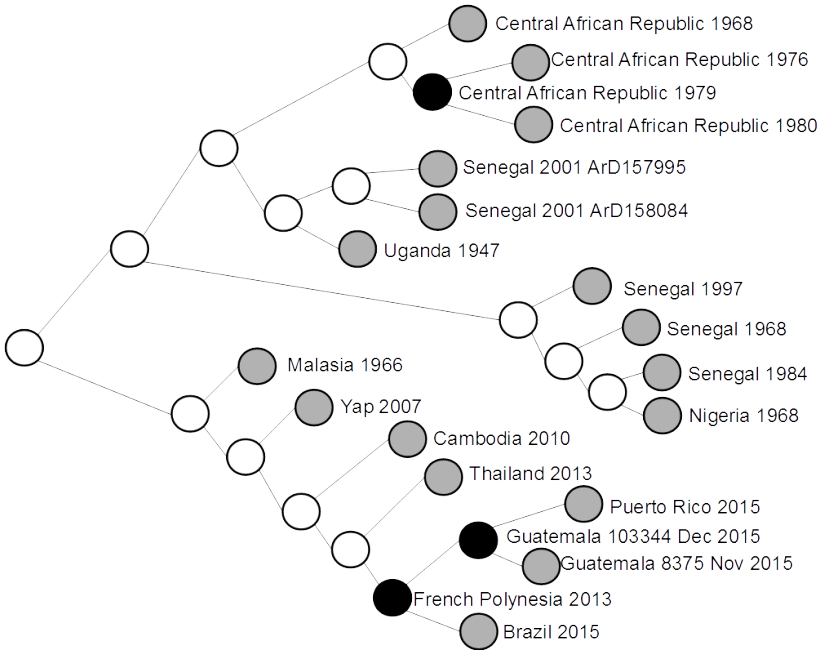


Figura 7.1: Árvore obtida por TBR Live para LLPP-l, $l = 3$.

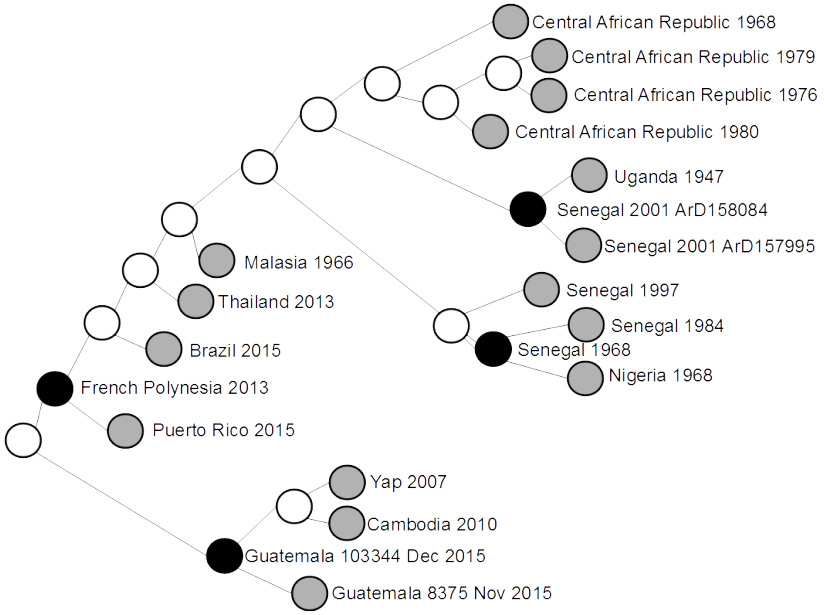


Figura 7.2: Árvore obtida por TBR Live Aresta Ponto Fraco para LLPP-l com $l = 4$.

árvore da Figura 7.1, com uma atribuição de ancestrais que é plausível com a ordem cronológica das amostras.

7.3 Comentários

Apesar das limitações com relação ao número de espécies, os testes realizados pelas baterias 1 e 2 indicam que as heurísticas construtivas tem construíram árvores com escore igual ou próximo do escore ótimo, obtido pelo *branch-and-bound*, para LLPP e LLPP-l-def, mas não para LLPP-l. O bom desempenho da heurística construtiva para LLPP-l-def é confirmado pela bateria 4, em que, para os casos $l = 1$, $l = 2$ e $l = 4$, não foi possível melhorar a solução através de heurísticas de rearranjo, sendo que a melhoria no caso $l = 3$ é muito pequena.

O mau desempenho das heurísticas construtivas para LLPP-l na bateria 2 provavelmente foi ocasionado pelas restrições impostas nas escolhas dos nós internos vivos. No caso da Adição Sequencial Live para $l = 1$ o nó interno vivo é escolhido aleatoriamente. Já no caso L-means Live, utilizada quando $l \geq 2$, embora a escolha dos nós internos vivos seja de forma a minimizar o custo de cada um dos l grupos, a heurística não constrói árvores nas quais um nó interno vivo seja ancestral de outro nó interno vivo, como na Figura 7.1.

Com relação às heurísticas de rearranjo, as baterias 2, 3 e 4, conseguiram melhorar o escore em LLPP-l. Para LLPP-l-def, as heurísticas de rearranjo também conseguiram melhorar os escores nas baterias 2 e 3, embora com um percentual de melhoria pequeno. Já na bateria 4, caso LLPP-l-def, a única a melhorar o resultado foi NNI para $l = 3$ e $l = 5$. NNI obtém melhores resultados na maioria das vezes, mas as diversas variações de TBR têm bons resultados em casos específicos. Já com respeito a LLPP é difícil concluir algo, pois na bateria 1 a melhoria no escore foi grande, porém, o mesmo não ocorreu nas baterias 3 e 4.

Capítulo 8

Conclusão

Nesta tese, o foco da pesquisa foi o problema da filogenia viva baseada em características. Em especial buscou-se elucidar o problema da parcimônia em filogenia viva.

Definimos e tratamos o Problema da Parcimônia Pequeno Viva (SLPP), que foi formulado originalmente em [24]. Sua solução é polinomial, consistindo em base para quaisquer heurísticas e métodos que necessitem calcular o escore parcimônia em filogenia viva, nesta tese.

Sendo filogenia viva uma extensão da teoria tradicional de filogenia e sendo o Problema da Parcimônia Grande (LPP) um problema NP-completo, é esperado que o Problema da Parcimônia Grande Viva (LLPP) também o seja. Considerações sobre sua complexidade foram feitas usando o Problema de Steiner em Grafos. Além de a própria definição do problema LLPP garantir que o número de árvores no universo de busca seja maior que em LPP, apresentamos dados mostrando que esta relação cresce de forma não linear. Esta diferença nas quantidades reflete-se na complexidade dos algoritmos de *branch-and-bound* propostos para LLPP. Na primeira abordagem, que resultou num algoritmo publicado em [25], a complexidade é $O(n!mn^n)$. Flexibilizando a montagem de árvores parciais durante o *branch-and-bound*, por meio da utilização de nós internos vivos pernetas, foi proposto outro algoritmo *branch-and-bound* com complexidade $O(nmn^n)$, publicado em [26].

A liberdade para inclusão de objetos atuais como nós internos da árvore, os nós internos vivos, permite a geração, a partir do problema inicial LLPP, de outros

dois possíveis problemas: LLPP-l e LLPP-l-def, quando há informação sobre a quantidade ou a definição de quais são os nós internos vivos, respectivamente.

Para o problema LLPP-l, sabe-se que a solução buscada deve possuir $l > 0$ nós internos vivos, desconhecidos previamente. Embora seja uma restrição importante no universo de busca, foi provado que LLPP-l é NP-completo. Foi proposta uma adaptação no *branch-and-bound* que resolve LLPP de forma a resolver LLPP-l.

Para o problema LLPP-l-def, que é uma situação mais restritiva, a solução buscada deve possuir um conjunto previamente conhecido de $l > 0$ nós internos vivos. Para este problema, um exemplo de motivação biológica bastante razoável é o caso em que temos espécies de vírus ainda em circulação, e para as quais temos pistas de serem ancestrais de outras também em circulação, como é o caso da espécie de vírus da Zika *French Polynesia 2013*, sabidamente ancestral da espécie *Brazil 2015*. Apesar desta ser uma restrição muito forte no universo de busca, o problema também é provado NP-completo. Apresentamos também uma adaptação do *branch-and-bound* que resolve LLPP de forma a resolver LLPP-l-def. Uma motivação prática como essa, vista para LLPP-l-def, obviamente não vale para LLPP-l. No entanto, o estudo e caracterização de LLPP-l certamente nos serve como instrumentação para um melhor entendimento do problema de filogenia viva como um todo.

Para cada um dos problemas, LLPP, LLPP-l e LLPP-l-def, foram propostos algoritmos baseados em *branch-and-bound*, que foram capazes de apresentar uma árvore com o menor escore, observado o universo de busca. Além disso, apresentamos heurísticas de construção e de rearranjo. A heurística construtiva para LLPP foi publicada [26], juntamente com o *branch-and-bound* otimizado. Neste mesmo trabalho foram feitos testes com uma modificação do *branch-and-bound* que gerava apenas filogenias vivas com pelo menos um nó interno vivo. Ainda, em um grande número de instâncias, a árvore retornada por este *branch-and-bound* modificado também era uma árvore mais parcimoniosa para LPP, ou seja, foi obtida uma árvore menor que a tradicional e com o mesmo escore. No momento da submissão desse artigo ([26]) não havíamos ainda proposto o problema LLPP-l e as árvores geradas por este *branch-and-bound* modificado provavelmente seriam as respostas para LLPP-l com $l = 1$, tendo em vista que a grande maioria apresentava apenas um nó interno vivo.

Embora não tenham sido adaptadas todas as heurísticas utilizadas para o problema LPP apresentadas nesta tese, é esperado que a maioria seja possível de adaptar e

utilizar para os problemas LLPP, LLPP-l e LLPP-l-def. Uma destas heurísticas, baseada em Algoritmos Genéticos, foi o tópico de trabalho envolvido em parceria, e publicado em [14]. Nesse trabalho, foram implementados três tipos de mutações: troca entre dois ramos da filogenia; troca da rotulação entre dois nós; movimentação de nós internos, por vezes gerando ou removendo uma folha. A operação de *crossover* foi a mesma apresentada nesta tese. O trabalho abordou vírus H1N1 e H1N3, de diferentes países, obtendo filogenias vivas interessantes.

Por fim, foram realizadas baterias de testes utilizando *benchmarks* apresentados por Andreatta e Ribeiro [1], baseados em casos reais. Como forma de aplicar a teoria em um caso real, foi feita uma bateria de testes com dados do vírus Zika, conforme apresentados por Lanciotti e colegas [34]. Os testes apontaram um bom desempenho das heurísticas construtivas sugeridas para LLPP e LLPP-l-def, mas não para LLPP-l. Já as heurísticas de rearranjo apresentaram bom desempenho para LLPP-l e LLPP-l-def, porém para LLPP os testes foram inconclusivos.

A revisão da teoria, com a inclusão dos problemas LLPP-l e LLPP-l-def, é tópico de um novo artigo em preparação [23].

Trabalhos Futuros

Alguns novos problemas propostos neste trabalho demandam uma série de novas abordagens, dentre as quais:

- adaptar, implementar e testar as outras heurísticas de LPP em cada um dos novos problemas;
- desenvolver meta-heurísticas compondo diversas heurísticas com seletor que altera entre elas;
- melhorar as heurísticas construtivas para LLPP-l, Adição Sequencial Live (para $l = 1$) e L-means Live (para $l > 1$);
- propor uma extensão à representação Newick que contemple nós internos vivos;
- propor heurística que receba como entrada uma filogenia tradicional e um percentual de aumento no escore tolerado e retorne filogenias vivas com escore dentro deste percentual.

Referências Bibliográficas

[1] A.A. Andreatta and C.C. Ribeiro. Heuristics for the phylogeny problem. *Journal of Heuristics*, 8(4):429–447, 2002.

[2] M. Bayzid, T. Hunt, and T. Warnow. Disk covering methods improve phylogenomic analyses. *BMC Genomics*, 15(6):1–11, 2014.

[3] H. Bodlaender, M. Fellows, and T. Warnow. Two strikes against perfect phylogeny. In *Proceedings of the 19th International Colloquium on Automata, Languages and Programming*, ICALP '92, pages 273–283, London, UK, UK, 1992. Springer-Verlag.

[4] J.H. Camin and R.R. Sokal. A method for deducing branching sequences in phylogeny. *Evolution*, 19(3):311–326, 1965.

[5] J. Cartresana. Selection of conserved blocks from multiple alignments for their use in phylogenetic analysis. *Molecular Biology and Evolution*, 17:540–552, 2000.

[6] B. Dasgupta, X. He, T. Jiang, M. Li, J. Tromp, and L. Zhang. On computing the nearest neighbor interchange distance. In *in: proc. Dimacs workshop on discrete problems with medical applications*, pages 125–143. Press, 1997.

[7] W.H.E. Day. Optimal algorithms for comparing trees with labeled leaves. *Journal of Classification*, 2(1):7–28, 1985.

[8] W.H.E. Day, D.S. Johnson, and D. Sankoff. The computational complexity of inferring rooted phylogenies by parsimony. *Mathematical Biosciences*, 1(81), 1986.

[9] W.H.E. Day and D. Sankoff. Computational complexity of inferring phylogenies by compatibility. *Systematic Biology*, 35(2):224–229, 1986.

[10] R.C. Edgar. MUSCLE: Multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, 32(5):1792–97, 2004.

[11] J.S. Farris. Methods for computing Wagner trees. *Systematic Zoology*, 19(1):83–92, 1970.

[12] J.S. Farris. Phylogenetic analysis under Dollo's Law. *Systematic Zoology*, 26(1):77–88, 1977.

[13] J. Felsenstein. *Inferring Phylogenies*, volume 2004. Sinauer Associates, 2004.

[14] R. L. Fernandes, R. Güths, G.P. Telles, N.F. Almeida, and M.E.M.T. Walter. A genetic algorithm for character state live phylogeny. In Ronnie Alves, editor, *Proc. 11th Brazilian Symposium on Bioinformatics, BSB 2018*, volume 11228 of *Lecture Notes in Bioinformatics*, pages 114–123, 2018. ISBN 978-3-030-01722-4.

[15] E. Ford, K.St. John, and W.C. Wheeler. Towards improving searches for optimal phylogenies. *Systematic Biology*, 64(1):56–65, 2015.

[16] L.R. Foulds and R.L. Graham. The Steiner problem in phylogeny is NP-complete. *Advances in Applied Mathematics*, 3(1):43–49, 1982.

[17] G. Giribet. Efficient tree searches with available algorithms. *Evolutionary Bioinformatics Online*, 1(3):341–356, 2007.

[18] T. Gojobori, E. Moriyama, and M. Kimura. Molecular clock of viral evolution, and the neutral theory. *P. Natl. Acad. Sci.*, 87(24):10015–10018, 1990.

[19] P. Goloboff and D. Pol. On divide-and-conquer strategies for parsimony analysis of large data sets: Rec-I-DCM3 versus TNT. *Systematic Biology*, 56(3):485–495, 2007.

[20] P.A. Goloboff. Analyzing large data sets in reasonable times: solutions for composite optima. *Cladistics*, 15(4):415–428, 1999.

[21] P.A. Goloboff. Computer Science and parsimony: a reappraisal, with discussion of methods for poorly structured datasets. *Cladistics*, 31(2):210–225, 2015.

[22] R.L. Graham and L.R. Foulds. Unlikelihood that minimal phylogenies for a realistic biological study can be constructed in reasonable computational time. *Mathematical Biosciences*, 60(2):133–142, 1982.

[23] R. Güths, G. P. Telles, M. E. M. T. Walter, and N. F. Almeida. The Large Live Parsimony Problem revised. In preparation.

[24] R. Güths, G.P. Telles, M.E.M.T. Walter, and N.F. Almeida. The weight small live parsimony problem. In *Proc. of The third International Society for Computational Biology Latin America*, 2014. ICSB-Latin America.

[25] R. Güths, G.P. Telles, M.E.M.T. Walter, and N.F. Almeida. A Branch and Bound for the Large Live Parsimony Problem. In *Proceedings of the 8th International Conference on Bioinformatics Models, Methods and Algorithms*, volume 3, pages 184–189, 2017. Biostec'2017.

[26] R. Güths, G.P. Telles, M.E.M.T. Walter, and N.F. Almeida. A heuristic for the Live Parsimony Problem. In Nathalia Peixoto, Margarida Silveira, Hesham H. Ali, Carlos Maciel, and Egon L. van den Broek, editors, *Biomedical Engineering Systems and Technologies*, pages 248–267, Cham, 2018. Springer International Publishing.

[27] M. Hendy and D. Penny. Branch and Bound algorithms to determine minimal evolutionary trees. *Mathematical Biosciences*, 59(2):277–290, 1982.

[28] T. Hill, A. Lundgren, R. Fredriksson, and H. Schioth. Genetic algorithm for large-scale maximum parsimony phylogenetic analysis of proteins. *Biochimica et Biophysica Acta (BBA) - General Subjects*, 1725(1):19–29, 2005.

[29] B. Holland, K. Huber, D. Penny, and V. Moulton. The MinMax Squeeze: Guaranteeing a minimal tree for population data. *Molecular Biology and Evolution*, 22(2):235–242, 2005.

[30] J.H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992.

[31] D. Huson, S. Nettles, and T. Warnow. Disk-covering, a fast-converging method for phylogenetic tree reconstruction. *Journal of Computational Biology*, 6(3/4):369–386, 1999.

[32] N.C. Jones and P.A. Pevzner. *An Introduction to Bioinformatics Algorithms*, volume 2004. MIT Press, 2004.

[33] K. Katoh, K. Kuma, and T. Miyata. Genetic algorithm-based maximum-likelihood analysis for molecular phylogeny. *Journal of Molecular Evolution*, 53(4-5):477–484, 2001.

[34] R.S. Lanciotti, A.J. Lambert, M. Holodniy, S. Saavedra, and L.C.C. Signor. Phylogeny of Zika virus in Western hemisphere, 2015. *Emerging Infectious Diseases*, 22(5):933–935, 2016.

[35] P.O. Lewis. A genetic algorithm for maximum-likelihood phylogeny inference using nucleotide sequence data. *Molecular Biology and Evolution*, 15(3):277–283, 1998.

[36] H.S. Lopes and M. Perretto. An Ant Colony system for large-scale phylogenetic tree reconstruction. *Journal of Intelligent & Fuzzy Systems*, 18(6):575–583, 2007.

[37] Z. Lu, J. Hao, and F. Glover. Neighborhood analysis: a case study on curriculum-based course timetabling. *Journal of Heuristics*, 17(2):97–118, 2010.

[38] D. MacKay. An example inference task: Clustering. *Information Theory, Inference, and Learning algorithms*, 22(3):284–292, 2003.

[39] D. Maddison. The discovery and importance of Multiple Islands of Most-Parsimonious trees. *Systematic Zoology*, 40(3):315–328, 1991.

[40] H. Matsuda. Protein phylogenetic inference using maximum likelihood with a genetic algorithm. In *Proc. of Pac Symp Biocomput.*, pages 512–523. World Scientific, 1996.

[41] N. Nguyen, S. Mirarab, and T. Warnow. MRL and SuperFine+ MRL: new supertree methods. *Algorithms for Molecular Biology*, 7(3):1–13, 2012.

[42] K.C. Nixon. The Parsimony Ratchet, a new method for rapid parsimony analysis. *Cladistics-the international journal of the Willi Hennig Society*, 15(4):407–414, 1999.

[43] D. Quicke, J. Taylor, and A. Purvis. Changing the landscape: a new strategy for estimating large phylogenies. *Systematic Biology*, 50(1):60–66, 2001.

[44] J. Richer, K. Vazquez-Ortiz, and D. Lesaint. Simulated annealing applied to the resolution of phylogenetic reconstruction with maximum parsimony. *Simulated Annealing*, pages 71–97, 2014.

[45] G. Robins and A. Zelikovsky. Tighter bounds for Graph Steiner Tree approximation. *SIAM J. Discret. Math.*, 19(1):122–134, May 2005.

[46] D.F. Robinson and L.R. Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53(1):131–147, 1981.

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

[48] M. Sanderson, M. McMahon, and M. Steel. Terraces in phylogenetic tree space. *Science*, 333(6041):448–450, 2011.

[49] D. Sankoff, Y. Abel, and J. Hein. A tree, a window, a hill: Generalization of nearest-neighbor interchange in phylogenetic optimization. *Journal of Classification*, 11(2):209–232, 1994.

[50] J.C. Setubal and J. Meidanis. *Introduction to Molecular Computational Biology.* PWS, 1997.

[51] W.K. Sung. *Algorithms in Bioinformatics: A Practical Introduction.* Chapman & Hall/CRC Mathematical and Computational Biology. CRC Press, 2009.

[52] 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. PMID: 23294270.

[53] T. Warnow. Large-scale phylogenetic reconstruction. In *Handbook of Computational Biology*, pages 1–21. Chapman & Hall, 2005.

[54] T. Warnow. *Models and Algorithms for Genome Evolution*, chapter Large-Scale Multiple Sequence Alignment and Phylogeny Estimation, pages 85–146. Springer, London, 2013.