2
O problema da filogenia

Um dos problemas centrais em biologia comparativa (sistemática) é o esclarecimento das relações de ancestralidade entre espécies, grupos de espécies, populações de espécies distintas, populações de uma mesma espécie ou genes homólogos em populações de espécies distintas (mas próximas do ponto de vista evolutivo) [6, 71]. Estas entidades podem ser classificadas sob a designação comum de taxon. A expressão da ancestralidade é feita através de uma árvore com raiz, onde as folhas representam os taxons sob análise e os nós internos representam ancestrais hipotéticos. Diz-se que esta árvore é uma árvore filogenética, uma árvore evolucionária ou uma filogenia. Os taxons sob análise são ditos taxons operacionais, enquanto os taxons eventualmente associados aos nós internos da árvore filogenética são denominados *taxons hipotéticos*. A associação de um taxon hipotético a cada nó interno de uma filogenia é uma *reconstrução*.

Cada taxon possui um conjunto de atributos denominados características. Uma característica pode representar um atributo morfológico (por exemplo, o tipo de mandíbula de espécies de peixes) ou uma posição de aminoácidos ou nucleotídeos numa proteína (por exemplo, proteínas como as hemoglobinas e os fibrinospeptídeos). A suposição fundamental sobre o conjunto de características consideradas na análise é que sejam independentes entre si. Isto simplifica a análise, pois não há necessidade de modelar o problema considerando a correlação entre as características.

Uma característica possui um conjunto discreto e finito de estados que podem ser assumidos por cada taxon. Este conjunto de estados é o domínio ou conjunto subjacente da característica.

A avaliação da qualidade de uma árvore filogenética depende do critério de otimização empregado para inferi-la. O critério pode ser baseado num modelo estocástico, na compatibilidade apresentada pelos dados, na comparação de distâncias num espaço métrico ou no *princípio de parcimônia*, sendo os dois últimos os mais comumente utilizados. No primeiro critério, as características são formuladas em um espaço métrico.

Um espaço métrico é um par (C, d), onde C é o conjunto subjacente e $d{:}C \times C \rightarrow \mathbb{R}$ é uma função de distância que tem as seguintes propriedades:

- $d(x, y) \geq 0, \forall x, y \in C$;
- $d(x, y) = 0 \Leftrightarrow x = y, \forall x, y \in C$;
- $d(x, y) = d(y, x), \forall x, y \in C$;
- $d(x, y) \leq d(x, z) + d(z, y), \forall x, y, z \in C$.

O grafo de transição de estados de uma característica que representa um espaço métrico é não direcionado e sua matriz de adjacência é simétrica.

Este critério é bastante utilizado em trabalhos sobre genoma [5, 29, 75]. Por exemplo [75], seja T uma árvore na qual os nós são representados por genomas. Seja G_i e G_j dois nós de T. Seja P_{ij} o caminho entre G_i e G_j em T, e seja k_e o número real de eventos ao longo da aresta $e \in P_{ij}$. A *distância evolucionária* entre G_i e G_j é $k_{ij} = \Sigma_{e \in P_{ij}} k_e$.

Pelo critério de parcimônia [71], a melhor árvore filogenética (mais parcimoniosa) é aquela que pode ser explicada pelo menor número de *passos evolutivos* [15, 43]. É frequentemente afirmado que o critério de parcimônia pode ser legitimado como o melhor critério de otimização na obtenção de árvores filogenéticas, desde que a probabilidade de ocorrerem mudanças evolucionárias seja pequena [56, 69]. O critério de parcimônia é utilizado ao longo deste trabalho.

A Figura 2.1 apresenta um conjunto de taxons e uma filogenia para este (ambos adaptados de [46]). Os dados são binários e estão representados por uma matriz, onde as linhas são indexadas por taxons e as colunas pelas características. As características presentes são: (a) pares de nadadeiras, (b) mandíbulas, (c) grandes ossos dérmicos, (d) barbatanas em fila, (e) pulmões, (f) línguas ásperas. O valor atribuído ao par (i, j) é 1 se o taxon i possui a característica j e 0 se a característica estiver ausente. A filogenia mostrada assume que os taxons descendem de um ancestral comum (raiz da árvore) onde todas as características estão ausentes. São sinalizados na figura os pontos onde cada característica muda de estado.

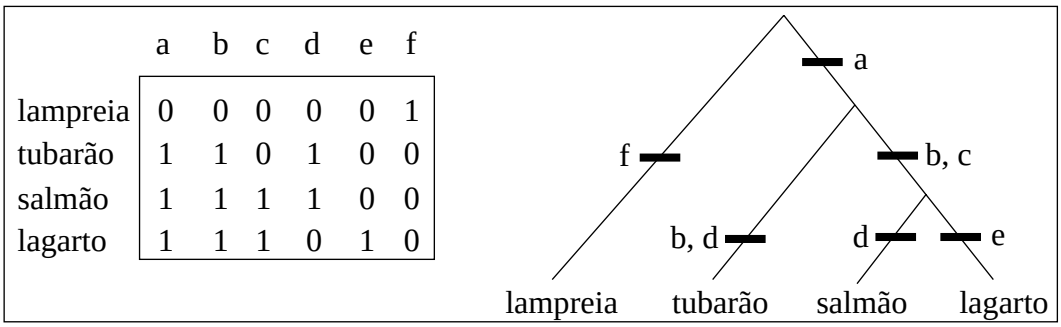

Figura 2.1: Exemplo de um conjunto de taxons e uma filogenia associada.

2.1 Aplicações

A inferência de árvores filogenéticas é importante para a fundamentação de taxonomias, que por sua vez tem uma estreita relação com a paleontologia [78]. Isso pode ocorrer de duas maneiras:

1. Espécimes fósseis podem apresentar estados para características morfológicas que não estão presentes nas espécies contemporâneas, podendo indicar também uma ordenação provável para os conjuntos de estados das características.

2. A introdução de uma espécie já extinta na análise após a obtenção de uma árvore evolucionária é uma das formas de testar sua plausibilidade.

Em biogeografia, árvores evolutivas auxiliam na identificação de uma possível correlação entre mudanças ambientais e mudanças morfológicas sobre uma mesma espécie (co-evolução ambiente-espécie) [77]. Em biologia molecular, dois genes são *homólogos* se possuem um ancestral comum. Os taxons operacionais podem ser genes homólogos de uma espécie ou de espécies distintas. A comparação entre os genes permite inferir tamanhos de populações de uma espécie ao longo do tempo [6]. Esta inferência é conseguida pela comparação das variações moleculares inter/extra-espécies de genes homólogos. Utilizando-se um modelo probabilístico para as variações de nucleotídeos nas seqüências e sabendo-se quando a função genética passou a ser realizada por organismos vivos, é possível estimar estatisticamente o tamanho populacional mínimo para gerar um subconjunto dos genes que foi agrupado monofileticamente numa árvore evolucionária proposta.

Há algum tempo atrás, utilizou-se a análise filogenética como evidência numa acusação de tentativa de homicídio [74]. O acusado, médico de um

grupo de infectados pelo vírus HIV, teria injetado sangue contaminando numa mulher. Comparou-se a população virótica da mulher com as populações viróticas dos pacientes do acusado e de uma amostragem das populações do vírus na América. O resultado apontou que o vírus que infectou a vítima era filogeneticamente muito mais próximo da população virótica dos pacientes do acusado do que do restante das populações utilizadas na análise.

2.2 Revisão bibliográfica/trabalhos anteriores

O primeiro trabalho de comparação de programas de inferência de filogenias sob o critério de parcimônia parece ser o de Luckow e Pimentel [48], publicado em 1985. As estratégias testadas por Luckow e Pimentel fazem parte dos sistemas WAGNER78 [18], WAGPROG [70], PHYSYS [52] e PHYLIP [19].

Os programas analisados no artigo de Luckow e Pimentel eram utilizados em computadores de grande porte, como o CDC Cyder 170 730/760 em que foram realizados os testes. A disseminação dos microcomputadores durante a década de 80 motivou os criadores destes programas a escreverem novas versões de seus sistemas para este tipo de máquinas. Platnick [57, 58] realizou o primeiro trabalho abrangente de comparação de sistemas de inferência de filogenias para microcomputadores.

Estes trabalhos se encontram em periódicos ou livros da área de biologia. Neles, os autores demonstraram preocupação em detalhar a contribuição proporcinada para a área de biologia, mas não demonstraram a mesma preocupação em detalhar os algoritmos implementados. Isto torna praticamente impossível a reconstrução de seus algoritmos por outros pesquisadores.

Outro detalhe importante sobre estes trabalhos é que não deixam claro como eram obtido os tempos relatados. Em comunicação pessoal com Goloboff [38], foi questionado qual seria o custo da melhor solução encontrada para a instância GOLO (instância construída por ele). Foi questionado também qual seria o tempo necessário para sua obtenção. A resposta foi a seguinte: “Hoje eu consigo uma solução com custo 497 em poucos segundos usando um computador Pentium III”. Essa resposta deixou a entender que Goloboff, em uma execução de seu programa, conseguiu tal solução em poucos segundos. Entretanto, não comenta quantas outras execuções foram feitas antes dessa, ou seja, não foi mencionado quanto tempo foi gasto antes de obter a melhor solução.

Em 1998, Andreatta e Ribeiro [3, 4] propuseram uma arquitetura de projeto (sob o paradigma de programação orientada a objetos) para o desenvolvimento de heurísticas de busca local e metaheurísticas para problemas de otimização combinatória. Nestes trabalhos foram realizadas implementações da arquitetura proposta para o problema da filogenia sob o critério de parcimônia. O sistema implementado tem opções de busca local do tipo melhoria iterativa, GRASP, VNS e busca tabu. Foram comparados de forma não tendenciosa (reutilizando amplamente código implementado) diferentes algoritmos aproximados para o problema da filogenia.

Os algoritmos mais promissores foram avaliados com base em 8 problemas reais de inferência de filogenias que são apresentados na Tabela 2.1. Na tabela são apresentados para cada instância o seu nome, o número n de taxons, o número m de características e, na coluna *melhor*, os melhores resultados conhecidos (antes dos algoritmos propostos nesta tese). Sete destes problemas foram obtidos através de contato com os editores da revista *Cladistics* (http://www.cladistics.org/journal.html). Estes problemas têm sido utilizados na comparação de programas para inferência de filogenias [48, 57, 58]. Além destas instâncias, foi utilizado também a instância GOLO fornecida por Goloboff [37].

Instância	n	m	*melhor*
ANGI	49	59	216
GRIS	47	93	172
TENU	56	179	682
ETHE	58	86	372
ROPA	75	82	326
GOLO	77	97	497
SCHU	113	146	760
CARP	117	110	548

Tabela 2.1: Melhores resultados para as oito instâncias da literatura.

O trabalho de Andreatta e Ribeiro [4] foi o primeiro a detalhar de forma clara os algoritmos utilizados. Também foi o primerio a alcançar os melhores resultados para as oito instâncias testes - é citado neste trabalho que os melhores resultados até então não foram obtidos por um mesmo algoritmo.

O fato de Andreatta e Ribeiro terem proposto uma arquitetura de projeto para o desenvolvimento de heurísticas de busca local e metaheurísticas para problemas de otimização combinatória torna esta ferramenta bastante genérica. O preço a pagar por isso é produzir algoritmos não muito otimiza-

dos. Quando uma aplicação é específica para um problema, características deste problema são aproveitadas para tornar o algoritmo mais rápido.

Um dos objetivos desta tese é o de melhorar os resultados produzidos por Andreatta e Ribeiro [3, 4], aproveitando alguns métodos que foram amplamente testados nestes trabalhos. Por exemplo, Andreatta e Ribeiro citam que dentre todos os algoritmos de construção testados, o melhor é o `GStep_wR` (*greedy step with randomness*). Essa informação foi aproveitada neste trabalho. No entanto, as implementações dos métodos de construção e também das estruturas de vizinhanças podiam ser implementadas mais eficientemente. Em [36, 37, 38], Goloboff propõe estratégias de otimização que sugerem uma mudança na estrutura de dados para o armazenamento das características de cada taxon, o que em média reduz o tempo de execução dos algoritmos de construção e busca local pela metade. Também é proposta uma nova maneira de avaliar a inserção de uma subárvore em outra em tempo $O(m)$, o que até então era feito em tempo $O(mn)$, onde n é o número de taxons sob análise e m o número de características associadas a cada taxon. Na Seção 2.4, estas otimizações propostas por Goloboff e outras que são propostas desta tese são descritas com maiores detalhes. Experimentos computacionais também mostram a economia de tempo atribuída a estas otimizações.

Neste trabalho também é proposta uma nova estrutura de vizinhança para o problema da filogenia (Capítulo 3) e também novos algoritmos baseados na metaheurística GRASP (Capítulo 3) e em algoritmos genéticos (Capítulo 4).

2.3 Avaliação de uma filogenia

Como dito anteriormente, o critério de avaliação da qualidade de uma árvore filogenética utilizado neste trabalho é baseado no princípio da parcimônia [71]. Este critério assume que a melhor filogenia é aquela que possui o número mínimo de passos evolutivos [15, 43]. Em outras palavras, dada uma filogenia s, a tarefa de avaliá-la consiste em definir o conjunto de características (taxons hipotéticos) de cada nó interno da árvore de tal forma que o número de mudanças de estado seja o menor possível.

A Figura 2.2 mostra uma filogenia que possui um conjunto de taxons hipotéticos baseados na evolução descrita na Figura 2.1. Nesta árvore, em cada aresta é sinalizada a quantidade de passos evolutivos ocorridos, totalizando um total de oito em toda a filogenia. Já na Figura 2.3 houve

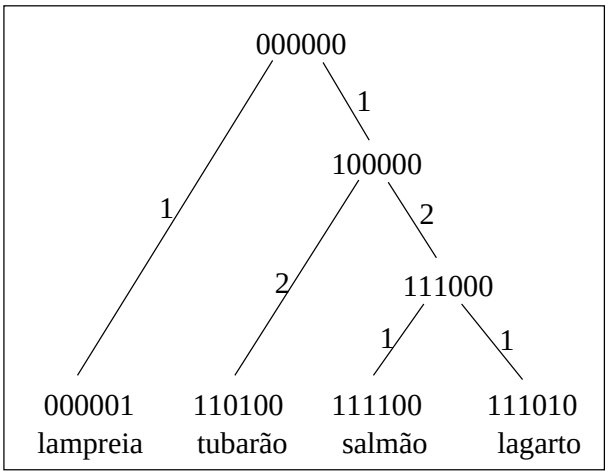

Figura 2.2: Exemplo de representação de uma filogenia.

uma melhor definição dos taxons hipotéticos dos nós internos, obtendo-se um número de passos evolutivos igual a sete, que representa o menor valor de parcimônia para tal filogenia.

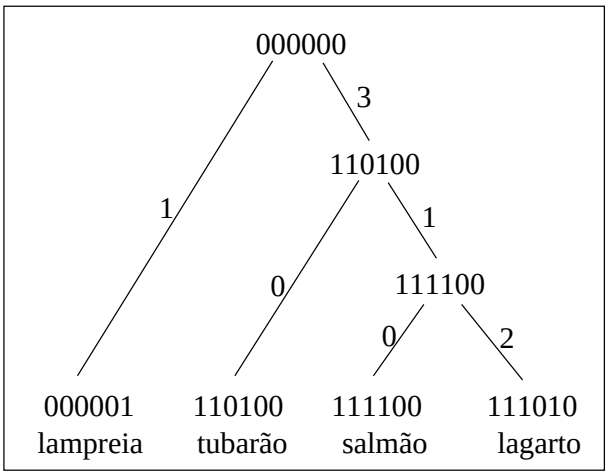

Figura 2.3: Exemplo de uma filogenia bem avaliada.

Existem na literatura científica alguns trabalhos sobre como definir o melhor conjunto de taxons hipotéticos [17, 24, 25]. Nestes artigos foram propostos algoritmos polinomiais com complexidade $O(mn)$ para realizar a computação do menor valor de parcimônia de uma filogenia, onde n é o número de taxons operacionais e m o número de características binárias. Eles assumem que cada característica binária pode assumir três estados: 0, 1 e *?*. Este último significa uma indeterminação, que é tratada nos algoritmos como se a característica pudesse assumir tanto o valor 0 quanto o valor 1, ou seja, o estado *?* é equivalente a {0,1}.

```
Procedimento CalcularCC(s)
Entrada
    s - a filogenia de entrada;
Saída
    Custo - o valor de parcimônia;
    CC - o vetor CC de cada nó de s;
Início
01.   Inicialize a pilha Q ← (raiz de s,0);
02.   Custo ← 0;
03.   Enquanto Q ≠ ∅ faça
04.   Início
05.     Desempilhar (N, t) de Q;
06.     Enquanto t ≠ 1 faça
07.     Início
08.       Empilhar (N, 1) em Q;
09.       Se N não é folha então
10.       Início
11.         Sejam FD e FE os nós filhos à direita e à esquerda de N, respectivamente;
12.         Empilhar (FD, 0) em Q;
13.         Empilhar (FE, 0) em Q;
14.       Fim-se
15.       Desempilhar (N, t) de Q;
16.     Fim-enquanto
17.     Se N é folha então
18.     Início
19.       Para i=1, ..., m faça
20.         CC[i] ← o estado da característica i do taxon relativo à folha N;
21.     Senão
22.       Seja CCFD o vetor CC do nó filho à direita de N;
23.       Seja CCFE o vetor CC do nó filho à esquerda de N;
24.       Para i=1, ..., m faça
25.         Se CCFD[i] = CCFE[i] então
26.           CC[i] ← CCFD[i];
27.         Senão
28.           Se CCFD[i] ≠ ? e CCFE[i] ≠ ? então
29.           Início
30.             CC[i] ← ?;
31.             Custo ← Custo+ 1;
32.           Senão
33.             Se CCFD[i] = ? então
34.               CC[i] = CCFE[i];
35.             Senão
36.               CC[i] = CCFD[i];
37.           Fim-se
38.     Fim-se
39.   Fim-enquanto
40.   Retorne Custo e o vetor CC de cada nó;
Fim-CalcularCC
```

Figura 2.4: Algoritmo para avaliação de uma filogenia s e para determinação do vetor CC de cada nó de s.

O algoritmo usado neste trabalho para avaliar uma filogenia [25] baseia-se no princípio de definir o conjunto de características de um nó com base nos conjuntos de características de seus dois sucessores imediatos (filhos). Por exemplo, a definição do estado da característica i de um nó depende dos estados associados a esta mesma característica nos seus sucessores imediatos. Se o estado 0 aparecer em maior número nos filhos então esta característica no nó pai é definida como 0; se for o estado 1, então será 1; caso haja empate, a característica do pai será indefinida, representada por *?* (par de estados {0,1}).

A Figura 2.4 apresenta o algoritmo implementado, que recebe como entrada uma filogenia s e retorna como saída o valor $f(s)$ de parcimônia de s, além de definir as características de cada nó interno da árvore. Essas características ficam armazenadas no vetor de catacterísticas comuns (vetor CC). A pilha Q inicializada com o nó raiz de s na linha 1 armazena tuplas do tipo (N, t), onde N é um nó da árvore s e t assume o valor 1 se os filhos do nó N já foram analisados, 0 caso contrário. O valor de parcimônia de s será acumulado na variável $Custo$ que é inicializada com 0 na linha 2. O laço nas linhas 3-39 garante que todos os nós da filogenia s serão analisados. Na linha 5 um nó N é desempilhado para possível análise. O laço nas linhas 6-16 faz com que se desça na árvore até chegar em uma folha para explorar os nós daí para cima (*bottom-up*). Na linha 8 o nó N é empilhado novamente com t=1. Nas linhas 12 e 13 os filhos de N são empilhados. Isso garante que N só será desempilhado novamente quando seus filhos já tiverem sido analisados. Nas linhas 17-20 é verificado se o nó N, que está sendo analisado, é uma folha. Neste caso, o vetor CC de N receberá as características do taxon operacional relativo à folha N. Caso contrário, o vetor CC de N é calculado com base no vetor CC de seus sucessores imediatos, como descrito nas linhas 21-38. Nas linhas 22 e 23 são definidas as variáveis $CCFD$ e $CCFE$ que armazenam, respectivamente, o vetor CC dos nós filhos à direita e à esquerda de N. No laço nas linhas 24-37 são determinadas todas as características que compõem o vetor CC de N. Na linha 25 verifica-se se a característica i assume o mesmo estado em $CCFD$ e em $CCFE$. Neste caso, a característica i do vetor CC de N assumirá também este estado. Na linha 28 é verificado se a característica i não assume o estado *?* nem em $CCFD$ nem em $CCFE$. Neste caso, a característica i do vetor CC de N assumirá o estado *?* e um passo evolutivo é detectado, fazendo com que a variável $Custo$ seja incrementada em uma unidade. Na linha 33 é verificado se a característica i de $CCFD$ é igual a *?*. Neste caso, a característica i do vetor CC de N assumirá o mesmo estado

que a característica i de $CCFE$. Caso contrário, assumirá o mesmo estado que a característica i de $CCFD$. Na linha 40 são retornados o número de passos evolutivos encontrados na filogenia s e o vetor CC de cada nó de s.

Todos os n nós da filogenia s são desempilhados duas vezes. Na segunda vez que um determinado nó é desempilhado, ele é analisado e todas as suas m características são definidas. Logo, o algoritmo descrito na Figura 2.4 tem complexidade $O(mn)$, onde n é o número de taxons sob análise e m o número de características.

2.4 Otimizações na implementação

Com a finalidade de tornar os algoritmos desenvolvidos mais rápidos, foram realizadas algumas otimizações no código. Algumas foram sugeridas por Goloboff [36, 37, 38] e outras são propostas deste trabalho. Estas otimizações podem ser divididas em dois grupos:

- O primeiro engloba as que envolvem uma diminuição da complexidade algorítmica.
- No segundo utilizou-se uma melhor estrutura de dados para o armazenamento do conjunto de características de cada taxon, o que permitiu uma economia de tempo de aproximadamente 50%.

Nas subseções a seguir, estas otimizações serão explicadas com maiores detalhes.

2.4.1 Otimização de complexidade

A inserção e a remoção de uma subárvore em uma árvore filogenética são operações muito utilizadas por qualquer heurística que trate o problema da filogenia. Tanto estratégias de construção de soluções (métodos construtivos) quanto de refinamento (busca local) realizam diversas vezes essas operações. Uma redução na complexidade de alguma delas acarreta uma grande economia de tempo.

Estratégias para se avaliar o impacto de uma inserção e de uma remoção em tempo $O(m)$ são descritas a seguir.

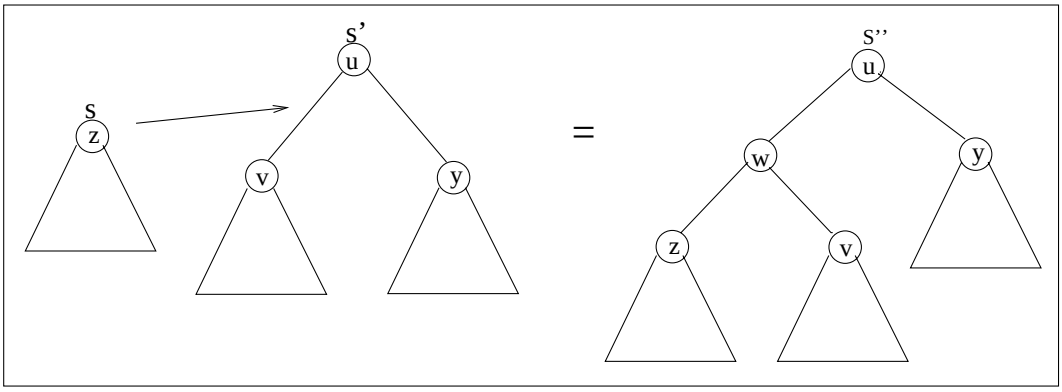

Figura 2.5: Exemplo de inserção de uma subárvore em uma filogenia.

Inserção em tempo $O(m)$

Um exemplo de inserção é descrito na Figura 2.5, onde a subárvore s com raiz z está sendo inserida na filogenia s' entre os nós u e v, resultando na filogenia s''. O custo incremental desta inserção é dado por $f(s'') - (f(s') + f(s))$, onde $f(.)$ representa o valor de parcimônia da filogenia.

Até então, para se verificar o impacto causado por uma operação de inserção era necessário avaliar a filogenia resultante para se calcular a diferença no valor de parcimônia entre a filogenia resultante e a filogenia original (essa diferença é denominada custo de inserção). Essa operação tem complexidade $O(mn)$, onde n é o número de taxons sob análise e m o número de características.

Goloboff em [36, 37] sugere uma nova estratégia para calcular em tempo $O(m)$ o custo de uma inserção, sem precisar calcular o valor de parcimônia da filogenia resultante. Esta estratégia se baseia no fato que as duas filogenias submetidas ao processo de inserção (s e s' no exemplo da Figura 2.5) já estão avaliadas, ou seja, são duas filogenias cujos valores de parcimônia já foram calculados. Esta estratégia propõe modificar o processo de avaliação de uma filogenia sem alterar a sua complexidade, mas de modo a permitir que se reduza a complexidade de uma operação de inserção. No momento em que uma filogenia s for avaliada, uma dupla travessia será realizada nesta árvore. A primeira travessia é feita de baixo para cima (*bottom-up*) para se calcular o seu custo e os vetores CC de cada nó, ou seja, é a análise descrita no algoritmo `CalcularCC` da Figura 2.4. A segunda é feita de cima para baixo (*top-down*) para se obter um outro conjunto de características para cada nó, que será armazenado no vetor de características especiais, CE. Estas características, assim como no vetor CC, podem assumir os estados 0, 1 e *?*. Ao contrário do vetor CC, elas não

representam o taxon associado a cada nó, mas servem apenas para otimizar a implementação, permitindo que as operações de inserção e remoção sejam feitas mais rapidamente (com complexidade $O(m)$, onde m é o número de características). Esse vetor é definido em um processo semelhante ao cálculo do vetor CC. Três informações serão utilizadas para a determinação do vetor CE de um nó N: os vetores CC dos seus dois sucessores imediatos (informações que também são utilizadas no cálculo do vetor CC) e o vetor CE do nó pai de N (que já estará disponível pois esta análise é realizada de cima para baixo). Se para uma característica i o estado 0 aparecer nestes três vetores em um número total de vezes maior que o número total de vezes que aparece o estado 1, a característica i do vetor CE de N assumirá o estado 0; se for o estado 1 a aparecer em um maior número total de vezes, a característica i do vetor CE de N assumirá o estado 1; em caso de empate, a característica i do vetor CE de N assumirá o estado *?*. Assim, o vetor CE de um nó N possui os estados predominantes para cada característica quando analisadas as informações que estão abaixo deste nó (vetores CC dos sucessores imediatos de N) na árvore e as informações que estão acima (vetor CE do nó pai de N). Com isso, quando um passo evolutivo ocorre, este fica descrito em uma aresta através dos vetores CE de suas extremidades.

Apresenta-se na Figura 2.6 o algoritmo para cálculo dos vetores CE de cada nó, que recebe como parâmetro de entrada a filogenia s que está sendo avaliada e retorna o vetor CE de cada nó desta filogenia. A pilha Q inicializada com a raiz de s na linha 1 armazena os nós da filogenia que serão analisados. O laço nas linhas 2-21 garante que todos os nós da filogenia serão analisados. Na linha 4 um nó N é desempilhado para análise. Na linha 5 é atribuído ao vetor $CEPAI$ o vetor CE do nó pai de N, caso N não seja a raiz de s. Caso contrário, é atribuído a $CEPAI$ o vetor CC do nó N. Na linha 6 é atribuído ao vetor $CCFD$ o vetor CC do nó filho à direita de N, caso N não seja uma folha. Caso contrário, é atribuído a $CCFD$ o vetor CC de N. Na linha 7 é atribuído ao vetor $CCFE$ o vetor CC do nó filho à esquerda de N, caso N não seja uma folha. Caso contrário, é atribuído a $CCFE$ o vetor CC de N. O laço nas linhas 8-15 determina o valor de todas as características do vetor CE do nó N. Nas linhas 9 e 10 verifica-se se o vetor CE do nó pai pode ser propagado para o vetor CE de N. Na linha 12 é verificado se em ambos os filhos a característica i do vetor CC possui o mesmo estado. Neste caso, na linha 13 é atribuído este estado à característica i do vetor CE de N. Caso contrário, a característica i do vetor CE de N recebe, na linha 15, o estado *?*. Nas linhas 18 e 19 os filhos de N são empilhados em Q para garantir que estes também serão analisados. Na

linha 22 é retornado o vetor CE de cada nó.

```
Procedimento CalcularCE(s)
Entrada
   s - a filogenia de entrada;
Saída
   CE - vetor CE de cada nó.
Início
01.   Inicialize a pilha Q ← (raiz de s);
02.   Enquanto Q ≠ ∅ faça
03.   Início
04.      Desempilhar N de Q;
05.      Seja CEPAI o vetor CE do nó pai de N. Se N for a raiz de s
         CEPAI será uma cópia do vetor CC de N;
06.      Seja CCFD o vetor CC do nó filho à direita de N. Se N for uma folha
         CCFD será uma cópia do vetor CC de N;
07.      Seja CCFE o vetor CC do nó filho à esquerda de N. Se N for uma folha
         CCFE será uma cópia do vetor CC de N;
08.      Para i=1, ..., m faça
09.         Se CC[i] = ? ou CEPAI[i] = CC[i] então
10.            CE[i] ← CEPAI[i];
11.         Senão
12.            Se CCFD[i] = CCFE[i] então
13.               CE[i] ← CCFD[i];
14.            Senão
15.               CE[i] ← ?;
16.      Se N não é folha então
17.      Início
18.         Empilhar o filho à direita de N em Q;
19.         Empilhar o filho à esquerda de N em Q;
20.      Fim-se
21.   Fim-enquanto
22.   Retorne o vetor CE de cada nó;
Fim-CalcularCE
```

Figura 2.6: Algoritmo para determinação do vetor CE de cada nó.

Cada um dos n nós de s é desempilhado para análise uma única vez. Durante a análise de um determinado nó, as m características do vetor CE deste nó são definidas. Logo, o algoritmo descrito na Figura 2.6 tem complexidade $O(mn)$, onde n é o número de taxons sob análise e m o número de características. Sendo o processo de avaliação de uma filogenia composto pela execução do algoritmo `CalcularCC` descrito na Figura 2.4 seguido da execução do algoritmo `CalcularCE` descrito na Figura 2.6 e sendo a complexidade de ambos algoritmos $O(mn)$, então a complexidade do processo de avaliação continua sendo $O(mn)$.

Uma vez que um passo evolutivo fica descrito em uma aresta (u, v) através do vetores CE dos nós u e v, para se testar a inserção de uma subárvore s em uma filogenia s' entre os nós u e v basta comparar o vetor CE do nó raiz de s com os dos nós u e v de s'. Verifica-se quais os passos

Procedimento `CalcValorInsercao`(CE_Z, CE_U, CE_V)
Entrada
CE_Z - Vetor CE da raiz (nó z) da subárvore cuja inserção será avaliada;
CE_U - Vetor CE do nó u da aresta (u, v) onde ocorrerá a inserção;
CE_V - Vetor CE do nó v da aresta (u, v) onde ocorrerá a inserção;
Saída
$Custo$ - Custo da inserção;
Início
01. $Custo \leftarrow 0$;
02. **Para** i=1, ..., m **faça**
03. **Se** $CE\_U[i] = CE\_V[i]$ **e** $CE\_U[i] \neq CE\_Z[i]$ **e** $CE\_Z[i] \neq ?$ **e** $CE\_U[i] \neq ?$ **e** $CE\_V[i] \neq ?$ **então**
04. $Custo \leftarrow Custo + 1$;
05. **Retorne** $Custo$;
Fim-`CalcValorInsercao`

Figura 2.7: Algoritmo para cálculo do custo de inserção.

evolutivos que não ocorrem em s' e que passarão a ocorrer na filogenia resultante s'' se a inserção for realizada.

A Figura 2.7 apresenta um algoritmo $O(m)$ (complexidade verificada no laço nas linhas 2-4) para o cálculo do custo de uma inserção. Na linha 1 é atribuído o valor 0 à variável $Custo$, que armazena o custo total da inserção. O laço nas linhas 2-4 garante que todas as características serão analisadas. A condição na linha 3 só será verdadeira se a característica i dos vetores CE dos nós u e v for 0 e a característica i do vetor CE do nó z for 1, ou vice-versa. Neste caso, não havia um passo evolutivo descrito na aresta (u, v) para a característica i e com a inserção da subárvore enraizada no nó z um passo evolutivo será gerado. Logo, na linha 4 o valor da variável $Custo$ é incrementado em uma unidade.

Remoção em tempo $O(m)$

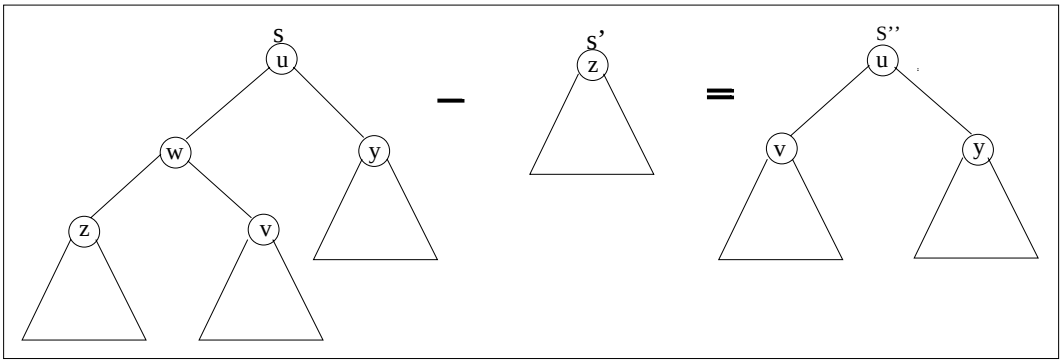

Figura 2.8: Exemplo de remoção de uma subárvore de uma filogenia.

```
Procedimento CalcularCC_2(s)
Entrada
    s - a filogenia de entrada;
Saída
    Custo - o valor de parcimônia;
    CC e CCAux - vetores CC e CCAux de cada nó;
Início
01.   Inicialize a pilha Q ← (raiz de s,0);
02.   Custo ← 0;
03.   Enquanto Q ≠ ∅ faça
04.     Desempilhar (N, t) de Q;
05.     Enquanto t ≠ 1 faça
06.       Empilhar (N, 1) em Q;
07.       Se N não é folha então
08.         Sejam FD e FE os nós filhos à direita e à esquerda de N, respectivamente;
09.         Empilhar (FD, 0) em Q;
10.         Empilhar (FE, 0) em Q;
11.       Fim-se
12.       Desempilhar (N, t) de Q;
13.     Fim-enquanto
14.     Se N é folha então
15.       Para i=1, ..., m faça
16.         CC[i] ← o estado da característica i do taxon relativo a folha N;
17.         Se CC[i] = ? então
18.           CCAux[i] ← ?;
19.         Senão
20.           CCAux[i] ← 0;
21.     Senão
22.       Seja CCFD o vetor CC do nó filho à direita de N;
23.       Seja CCFE o vetor CC do nó filho à esquerda de N;
24.       Seja CCAuxFD o vetor CCAux do nó filho à direita de N;
25.       Seja CCAuxFE o vetor CCAux do nó filho à esquerda de N;
26.       Para i=1, ..., m faça
27.         Se (CCAuxFD[i] = ?) e (CCAuxFE[i] = ?) então
28.           CCAux[i] ← ?;
29.         Senão
30.           CCAux[i] ← 0;
31.         Se CCFD[i] = CCFE[i] então
32.           CC[i] ← CCFD[i];
33.         Senão
34.           Se CCFD[i] ≠ ? e CCFE[i] ≠ ? então
35.             CC[i] ← ?;
36.             Custo ← Custo+ 1;
37.           Senão
38.             Se CCFD[i] = ? então
39.               CC[i] = CCFE[i];
40.             Senão
41.               CC[i] = CCFD[i];
42.       Fim-Para
43.     Fim-se
44.   Fim-enquanto
45.   Retorne Custo e os vetores CC e CCAux de cada nó;
Fim-CalcularCC_2
```

Figura 2.9: Algoritmo para avaliação de uma filogenia s e para determinação dos vetores CC e $CCAux$ de cada nó de s.

Quando uma subárvore s' é desconectada de uma filogenia s, o nó pai do nó raiz de s' é eliminado e a subárvore irmã de s' é conectada ao nó avô do nó raiz de s'. Em outras palavras, na Figura 2.8 a subárvore s' enraizada pelo nó z é desconectada, o nó w (nó pai) é eliminado de s e a subárvore irmã (enraizada pelo nó v) é conectada ao nó u (nó avô de z em s) gerando a subárvore s''. Assim, percebe-se que os nós mais importantes durante uma operação de remoção são a raiz z da subárvore s' a ser desconectada de s, o nó raiz v da subárvore irmã e o nó avô u de z em s. Comparando os vetores CE destes três nós é possível determinar a redução de passos evolutivos ocasionados pela remoção.

Existem duas ocasiões onde um passo evolutivo é reduzido em uma remoção. A primeira é semelhante à existente no processo de inserção: um passo evolutivo é reduzido quando no vetor CE do nó raiz z da subárvore s' a ser desconectada a característica i assume o estado 0 e nos vetores CE do nó irmão v e do nó avô u a característica i assume o estado 1, ou vice-versa. A outra ocasião onde um passo evolutivo é reduzido ocorre quando nos vetores CE dos nós u e v a característica i assume o estado *?* e no vetor CE do nó z a característica i assume um estado diferente de *?*. Além disso, na subárvore enraizada pelo nó u (nó avô de z) em s deve existir pelo menos um nó onde a característica i assuma um valor diferente de *?* no vetor CC.

Para saber, de maneira rápida e eficiente, se em uma subárvore enraizada por um nó N existe algum nó onde a característica i assume um estado diferente de *?* no vetor CC, foi definido um novo vetor, $CCAux$. Este assume o estado *?* para a característica i se na subárvore enraizada pelo nó N a característica i assume apenas o estado *?* no vetor CC e $CCAux[i] = 0$ caso contrário ($CCAux[i]$ só será consultado para saber se uma determinada característica i assume o estado *?* em toda subárvore enraizada por N).

O vetor $CCAux$ de cada nó será calculado juntamente com o vetor CC. Para isso, o algoritmo `CalcularCC` apresentado na Figura 2.4 foi modificado gerando o algoritmo descrito na Figura 2.9. O que muda em relação ao algoritmo da Figura 2.4 são os trechos nas linhas 17-20 e nas linhas 27-30. No primeiro destes trechos, quando o nó N que está sendo analisado é uma folha, se uma característica i assume no vetor CC o estado *?*, então esse mesmo estado é assumido para a característica i no vetor $CCAux$ (se a característica i assumir um estado diferente de *?* no vetor CC então $CCAux[i] = 0$). No segundo trecho, como N não é uma folha, deve-se verificar se nos vetores $CCAux$ de ambos os filhos a característica i assume o valor *?*. Neste caso, o valor da característica i no vetor $CCAux$ de N também será *?* (caso contrário, $CCAux[i] = 0$). Analogamente ao algoritmo

`CalcularCC` da Figura 2.4, este algoritmo tem complexidade $O(mn)$, onde n é o número de taxons sob análise e m o número de características.

```
Procedimento CalcValorRedução(CENo, CEIrmao, CEAvo, CCAuxAvo)
Entrada
    CENo - Vetor CE da raiz da subárvore cuja remoção será avaliada;
    CEIrmao - Vetor CE do nó irmão da raiz da subárvore cuja remoção será avaliada;
    CEAvo - Vetor CE do nó avô da raiz da subárvore cuja remoção será avaliada;
    CCAuxAvo - Vetor CCAux do nó avô da raiz da subárvore cuja remoção será avaliada;
Saída
    Redução - Redução de passos evolutivos ocasionados pela remoção;
Início
01.   Redução ← 0;
02.   Para i=1, ..., m faça
03.     Se CEAvo[i] = ? e CEIrmao[i] = ? então
04.       Se CCAuxAvo[i] ≠ ? e CENo[i] ≠ ? então
05.         Redução ← Redução + 1;
06.     Senão
07.       Se CEAvo[i] = CEIrmao[i] e CEAvo[i] ≠ CENo[i]
            e CENo[i] ≠ ? e CEAvo[i] ≠ ? e CEIrmao[i] ≠ ? então
08.         Redução ← Redução + 1;
09.   Retorne Redução;
Fim-CalcValorRedução
```

Figura 2.10: Algoritmo para cálculo da redução de passos evolutivos ocasionada por uma remoção.

O algoritmo proposto neste trabalho para cálculo da redução de passos evolutivos ocasionados pela remoção de uma subárvore em tempo $O(m)$ (complexidade verificada no laço nas linhas 2-8) é apresentado na Figura 2.10. Na linha 1 é atribuído o valor 0 à variável *Redução*, que armazena a redução total de passos evolutivos ocasionada pela remoção. O laço nas linhas 2-8 garante que todas as características serão analisadas. Se a característica i assumir o valor *?* no vetor CE do nó irmão (v) do nó raiz (z) da subárvore a ser desconectada e no vetor CE do seu nó avô (u), assumir um valor diferente de *?* no vetor CE do nó z e, além disso, na subárvore enraizada pelo nó u existir pelo menos um nó onde a característica i assume um estado diferente de *?* no vetor CC (isso é verficado com a condição $CCAux[i] \neq ?$), então um passo evolutivo é detectado e a variável *Redução* é incrementada em uma unidade na linha 5. A condição na linha 7 só será verdadeira se a característica i dos vetores CE dos nós u e v for 0 e a característica i do vetor CE do nó z for 1, ou vice-versa. Neste caso, havia um passo evolutivo para a característica i e com a remoção da subárvore enraizada no nó z esse passo evolutivo deixará de ocorrer. Logo, na linha 8 o valor da variável *Redução* é incrementada em uma unidade.

2.4.2 Otimização pela estrutura de dados

Outra otimização proposta por Goloboff [38] foi substituir a estrutura de dados que armazena as características. Esta estrutura deixa de ser um vetor numérico, onde cada posição representa uma característica, e passa a ser um vetor de valores numéricos de 32 *bits*, onde cada posição representa um grupo de 16 características. São necessários 2 *bits* para representar cada um dos três estados possíveis: 0, 1 e *?*. Utilizou-se a seguinte representação para cada um dos estados:

- o estado 0 é representado pela seqüência de *bits* 01;
- o estado 1 é representado pela seqüência de *bits* 10;
- o estado *?* é representado pela seqüência de *bits* 11; e
- não existe nenhum estado associado à seqüência de *bits* 00.

Nos algoritmos que serão descritos nesta subseção, os vetores CC, $CCAux$ e CE possuem as mesmas funções e armazenam a mesma quantidade m de características. O que difere em relação aos algoritmos descritos anteriormente é que cada posição g ($1 \leq g \leq m/16$) destes vetores armazena um número de 32 bits, onde um grupo de 16 características é definido (cada par de *bits* define uma característica).

As operações nesta nova estrutura são feitas por operadores binários, onde de uma só vez dezesseis características são comparadas. Os operadores utilizados são o operador *e* (&), o operador *ou* (|), o operador *não* (!), o operador *ou-exclusivo* ($\wedge$) e os operadores de *deslocamento* à direita ($>>$) e à esquerda ($<<$).

Uma operação com os operadores binários &, | e $\wedge$ possui a sintaxe $A \leftarrow B\ op\ C$, onde A, B e C são variáveis numéricas de 32 *bits* e $op \in \{\&, |, \wedge\}$. Se $op = \&$, então um *bit* de A recebe o valor 1 se este mesmo *bit* assumir o valor 1 tanto em B quanto em C. Caso contrário, esse *bit* em A recebe o valor 0. Se $op = |$, então um *bit* de A recebe o valor 1 se este mesmo *bit* assumir o valor 1 em B ou em C. Caso contrário, esse *bit* em A recebe o valor 0. Se $op = \wedge$, então um *bit* de A recebe o valor 1 se este mesmo *bit* assumir o valor 1 ou em B ou em C. Caso contrário, esse *bit* em A recebe o valor 0.

Uma operação com os operadores binários $>>$ e $<<$ possui a sintaxe $A \leftarrow B\ op\ num$, onde A e B são variáveis numéricas de 32 *bits*, $op \in \{<<, >>\}$ e num é um número que indica de quantas casas será o deslocamento.

Se a expressão for $A \leftarrow B << 1$, por exemplo, a variável A recebe a variável B com seus *bits* deslocados uma casa à esquerda.

Uma operação com o operador binário ! possui a sintaxe $A \leftarrow !B$, onde A e B são variáveis numéricas de 32 *bits*. Neste caso, A recebe B com os *bits* invertidos (onde é 0 em B é 1 em A, e vice-versa).

Antes de apresentar os algoritmos com a nova estrutura de dados, é importante destacar a etapa de pré-processamento existente que define três variáveis bastante utilizadas nestes algoritmos:

- $Bits0$ - variável numérica de 32 *bits* com todos os *bits* de ordem par valendo 1, e os restantes iguais a 0:

  $Bits0 = 01010101010101010101010101010101$

- $Bits1$ - variável numérica de 32 *bits* com todos os *bits* de ordem ímpar valendo 1, e os restantes iguais a 0:

  $Bits1 = 10101010101010101010101010101010$

- $VetorCusto$ - vetor de 2^{32} posições, onde cada posição representa um número de 32 *bits*. Como já mencionado, as características estarão armazenadas em um vetor de valores numéricos de 32 *bits*. Todas as operações serão realizadas sobre números de 32 *bits*. Assume-se nos próximos algoritmos deste capítulo que os passos evolutivos a serem "contados"ficarão assinalados em uma variável auxiliar de 32 *bits* com a seqüência de *bits* 00, pois esta representação não é usada para representar estado algum. Por exemplo, em uma inserção, os passo evolutivos a serem contados são aqueles que não ocorrem e passarão a ocorrer após a inserção. Neste caso, esses passos evolutivos devem ficar assinalados em uma variável auxiliar. Depois, conta-se quantas características representadas pela seqüência de *bits* 00 existem nesta variável auxiliar. É nesta hora que o $VetorCusto$ é necessário, pois ele guarda para cada valor numérico de 32 *bits* o número de seqüências de *bits* 00 existentes.

Alguns dos algoritmos descritos no decorrer deste capítulo utilizam o procedimento `CalcularCusto`. Este recebe como parâmetro de entrada um número de 32 *bits* A e retorna como saída o número de seqüências de *bits* 00 existentes em A. Isso é feito em tempo $O(1)$, pois este procedimento faz uso do vetor $VetorCusto$ definido na etapa de pré-processamento.

Combinando as variáveis pré-processadas, os operadores binários e o procedimento `CalcularCusto` é possível reimplementar os algoritmos descritos nas seções anteriores deste capítulo utilizando a nova estrutura de dados para o armazenamento das características. A seguir serão apresentados alguns exemplos de como combinar estes elementos para extrair informações.

Exemplo 1. Sabe-se que o estado 1 é representado pela seqüência de *bits* 10, o estado 0 pela seqüência 01 e o estado *?* pela seqüência 11. Deseja-se extrair do grupo $CC[g]$ de 16 características aquelas onde os *bits* de ordem ímpar têm valor 1, ou seja, as características que possuem o estado 1 ou *?*. É comum nos algoritmos que serão descritos a seguir destacar-se, nas operações intermediárias, as características que se deseja extrair com a seqüência de *bits* 11, pois isso facilita uma próxima operação binária posterior.

1. $A \leftarrow CC[g]$ & $Bits1$
2. $A \leftarrow A \mid (A >> 1)$

Na linha 1 os *bits* de ordem ímpar com valor 1 em $CC[g]$ permanecem com o valor 1 em A. Todos os outros *bits* recebem o valor 0. Na linha 2, o deslocamento de uma casa à direita faz com que os *bits* de ordem par de A relativos aos *bits* que possuem o valor 1 passem a ter também o valor 1. Com essa operação a variável A passa a ter a seqüência de *bits* 11 nos pares de *bits* que representam as características que se deseja extrair e 00 nos restantes.

Exemplo 2. Destacar as características de $CC[g]$ que assumem o estado *?*:

1. $A \leftarrow CC[g]$ & $Bits1$
2. $B \leftarrow CC[g]$ & $Bits0$
3. $C \leftarrow A$ & $(B << 1)$
4. $D \leftarrow C \mid (C >> 1)$

Na linha 1 recebem valor 1 os *bits* de ordem ímpar de A relativos às características com estado 1 ou *?* em $CC[g]$. Na linha 2 recebem valor 1 os *bits* de ordem par de B relativos às características com estado 0 ou *?* em $CC[g]$. Na linha 3 os *bits* de ordem ímpar de C recebem o valor 1 quando este mesmo *bit* recebe o valor 1 em A e o *bit* par relativo a este recebe valor 1 em B (ou seja, as características com estado *?*). Na linha 4 são destacados

com a seqüência de *bits* 11 os pares de *bits* relativos às características com estado *?*.

Pode-se reescrever este exemplo como descrito a seguir.

1. $A \leftarrow (CC[g]\ \&\ Bits1)\ \&\ ((CC[g]\ \&\ Bits0) << 1)$
2. $A \leftarrow A \mid (A >> 1)$

Esta segunda maneira é mais frequentemente utilizada nos algoritmos que serão descritos no decorrer deste capítulo.

Exemplo 3. Destacar as características de $CC[g]$ que não assumem o estado *?*:

1. $A \leftarrow (CC[g]\ \&\ Bits1) \wedge ((CC[g]\ \&\ Bits0) << 1)$
2. $A \leftarrow A \mid (A >> 1)$

Na linha 1, um *bit* de ordem ímpar de A recebe o valor 1 ou quando este *bit* possui valor 1 em $CC[g]$ ou quando o *bit* par relativo em $CC[g]$ possui o valor 1 (ou-exclusivo). Na linha 2, os pares de *bits* relativos às características desejadas são destacados com a seqüência de *bits* 11.

Exemplo 4. Algumas variáveis auxiliares podem assumir, após algumas operações, alguns pares de *bits* com o valor 00 (o que não ocorre no vetor CC pois suas características assumem apenas os estados 0, 1 ou *?*). Deseja-se extrair da variável auxiliar B os pares de bits que não assumem o valor 00. Para isso, têm-se os seguintes comandos:

1. $A \leftarrow (B\ \&\ Bits1) \mid ((B\ \&\ Bits0) << 1)$
2. $A \leftarrow A \mid (A >> 1)$

Na linha 1, um *bit* de ordem ímpar de A recebe o valor 1 quando este *bit* possui valor 1 em B ou o *bit* par relativo em B possui o valor 1. Na linha 2, os pares de *bits* desejados são destacados com a seqüência de *bits* 11.

Exemplo 5. Sejam $CCFD[g]$ e $CCFE[g]$ os g-ésimos grupos de 16 características do filho direito e esquerdo de um nó N, respectivamente. Deseja-se saber quantas características valem 0 no filho à direita de N e 1 no filho à esquerda de N, ou vice-versa. Em outras palavras, deseja-se saber quantos passos evolutivos ocorrem entre o nó N e seus filhos.

Sabe-se que os passos evolutivos devem ficar assinalados em uma variável auxiliar de 32 *bits* através da seqüência de *bits* 00. Sendo assim,

os comandos abaixo são suficientes.

1. $A \leftarrow CCFD[g]$ & $CCFE[g]$
2. $Custo \leftarrow$ `CalcularCusto`(A)

Na linha 1 a variável A terá um par de *bits* representado pela seqüência de *bits* 00 somente quando a característica relativa a este mesmo par assumir o estado 0 em $CCFD[g]$ e o estado 1 em $CCFE[g]$, ou vice-versa. Na linha 2 a quantidade de passos evolutivos é calculada pelo procedimento `CalcularCusto` e armazenada na variável $Custo$.

A seguir serão apresentados os novos algoritmos para avaliação de uma filogenia, os novos algoritmos para se verificar o impacto de inserções e remoções em uma filogenia e os experimentos realizados para verificar a eficiência desta otimização.

Avaliação de uma filogenia com representação binária

A Figura 2.11 apresenta o novo algoritmo para determinação dos vetores CC e $CCAux$ de cada nó. O que muda em relação ao algoritmo `CalcularCC_2` descrito na Figura 2.9 são os trechos das linhas 19-26 e das linhas 32-42. O laço nas linhas 19-26 garante que todos os grupos de 16 características dos vetores CC e $CCAux$ serão definidos quando o nó N for uma folha. Na linha 21 o g-ésimo grupo de 16 características relativo à folha N é atribuído a $CC[g]$. No trecho nas linhas 23-25 o g-ésimo grupo de características do vetor $CCAux$ é definido. Nas linhas 23 e 24, analogamente ao Exemplo 3, são destacados na variável auxiliar C com a seqüência de *bits* 11 os pares de *bits* relativos às características que assumem um estado diferente de *?* em $CC[g]$. O comando da linha 25 faz com que as características que assumem o estado *?* em $CC[g]$ sejam representadas em $CCAux[g]$ pela seqüência de *bits* 11 e por 00 em caso contrário. O laço nas linhas 32-42 garante que todos os grupos de 16 características dos vetores CC e $CCAux$ serão definidos quando o nó N não for uma folha. Nas linhas 34 e 35, como mostrado no Exemplo 5, são computados, para o g-ésimo grupo de 16 características, os passos evolutivos existentes entre o nó N e seus filhos que são acumulados na variável $Custo$. No trecho nas linhas 37-39 o g-ésimo grupo de 16 características do vetor CC é definido. Nas linhas 37 e 38, analogamente ao Exemplo 4, são destacadas com a seqüência de *bits* 11 os pares de *bits* da variável auxiliar A representados por uma seqüência de *bits* diferente de 00. Na linha 39, $CC[g]$ assume os

```
Procedimento CalcularCCBin(s)
Entrada
    s - a filogenia de entrada;
Saída
    Custo - o valor de parcimônia;
    CC e CCAux - vetor CC e CCAux de cada nó;
Início
01.   Inicialize a pilha Q ← (raiz de s,0);
02.   Custo ← 0;
03.   Enquanto Q ≠ ∅ faça
04.   Início
05.     Desempilhar (N, t) de Q;
06.     Enquanto t ≠ 1 faça
07.     Início
08.       Empilhar (N, 1) em Q;
09.       Se N não é folha então
10.       Início
11.         Sejam FD e FE os nós filhos à direita e à esquerda de N, respectivamente;
12.         Empilhar (FD, 0) em Q;
13.         Empilhar (FE, 0) em Q;
14.       Fim-se
15.       Desempilhar (N, t) de Q;
16.     Fim-enquanto
17.     Se N é folha então
18.     Início
19.       Para g=1, ..., m/16 faça
20.       Início
21.         CC[g] ← g-ésimo grupo de 16 características do taxon relativo a folha N;
22.
23.         C ← ((CC[g] & Bits0) << 1) ∧ (CC[g] & Bits1);
24.         C ← C | (C >> 1);
25.         CCAux[g] ← C & CC[g];
26.       Fim-para
27.     Senão
28.       Seja CCFD o vetor CC do nó filho à direita de N;
29.       Seja CCFE o vetor CC do nó filho à esquerda de N;
30.       Seja CCAuxFD o vetor CCAux do nó filho à direita de N;
31.       Seja CCAuxFE o vetor CCAux do nó filho à esquerda de N;
32.       Para g=1, ..., m/16 faça
33.       Início
34.         A ← CCFD[g] & CCFE[g];
35.         Custo ← Custo + CalcularCusto(A);
36.
37.         C ← (A & Bits0) | ((A & Bits1) << 1) ;
38.         C ← C | (C >> 1);
39.         CC[g] ← A | (!C);
40.
41.         CCAux[g] ← CCAuxFE[g] & CCAuxFD[g];
42.       Fim-Para
43.     Fim-se
44.   Fim-enquanto
45.   Retorne Custo e os vetores CC e CCAux de cada nó;
Fim-CalcularCCBin
```

Figura 2.11: Algoritmo para avaliação de uma filogenia s e para determinação dos vetores CC e $CCAux$ de cada nó de s.

mesmos pares de bits de A, exceto quando representados pela seqüência de *bits* 00 que passam a ser representadas em $CC[g]$ pela seqüência de *bits* 11 (estado *?*). Na linha 41, o g-ésimo grupo de 16 características do vetor $CCAux$ é definido, fazendo com que uma característica de $CCAux[g]$ seja representada pelo estado *?* (seqüência de *bits* 11) se este estado for assumido por esta mesma característica nos elementos $CCAuxFD[g]$ e $CCAuxFE[g]$, e pela seqüência 00 em caso contrário.

Analogamente ao algoritmo `CalcularCC_2`, o algoritmo `CalcularCCBin` possui complexidade $O(mn)$, onde n é o número de taxons sob análise e m o número de características binárias.

Procedimento `CalcularCEBin`(s)
Entrada
s - a filogenia de entrada;
Saída
CE - o vetor CE de cada nó;
Início
01. Inicialize a pilha $Q \leftarrow$ (raiz de s);
02. **Enquanto** $Q \neq \emptyset$ **faça**
03. **Início**
04. Desempilhar N de Q;
05. Seja $CEPAI$ o vetor CE do nó pai de N. Se N for a raiz de s então N não tem pai e $CEPAI$ será uma cópia do vetor CC de N;
06. Seja $CCFD$ o vetor CC do nó filho à direita de N. Se N for uma folha então N não tem filhos e $CCFD$ será uma cópia do vetor CC de N;
07. Seja $CCFE$ o vetor CC do nó filho à esquerda de N. Se N for uma folha então N não tem filhos e $CCFE$ será uma cópia do vetor CC de N;
08. **Para** g=1, ..., $m/16$ **faça**
09. **Início**
10. $A \leftarrow CCFD[g]$ & $CCFD[g]$;
11. $C \leftarrow ((A \;\&\; Bits0) << 1) \wedge (A \;\&\; Bits1)$;
12. $C \leftarrow C \mid (C << 1)$;
13. $D \leftarrow CEPAI[g]$ & $!C$;
14. $E \leftarrow (A \mid (CCFD[g] \;\&\; CEPAI[g]) \mid (CCFE[g] \;\&\; CEPAI[g])) \;\&\; C)$;
15. $CE[i] \leftarrow D \mid E$;
16. **Fim-Para**
17. **Se** N não é folha **então**
18. **Início**
19. Empilhar o filho à direita de N em Q;
20. Empilhar o filho à esquerda de N em Q;
21. **Fim-se**
22. **Fim-enquanto**
23. **Retorne** o vetor CE de cada nó;
Fim-`CalcularCEBin`

Figura 2.12: Algoritmo para determinação do vetor CE de cada nó.

A Figura 2.12 apresenta o novo algoritmo para determinação do vetor de características CE de cada nó. O que muda em relação ao algoritmo `CalcularCE` descrito na Figura 2.6 é o trecho nas linhas 8-16. O laço nas linhas 8-16 garante que todos os grupos de 16 características do vetor CE

serão definidos. Na linha 10, a variável auxiliar A recebe o resultado de uma operação & entre $CCFD[g]$ e $CCFE[g]$ que atribui a seqüência de *bits* 00 aos pares de *bits* de A relativos às características que em um vetor possuem o estado 0 e no outro o estado 1. Nas linhas 11-12, analogamente ao Exemplo 3, são destacados na variável auxiliar C com a seqüência de *bits* 11 os pares de *bits* que em A são representados por uma seqüência de *bits* diferente de 00. Na linha 13 são definidas as características armazenadas em $CEPAI[g]$ que serão propagadas para $CE[g]$, uma vez que nos filhos essas características apresentam o mesmo número de estados assumindo o valor 1 e 0. Na linha 14 são definidas as características restantes baseadas nos vetores CC de cada filho. Na linha 15, $CE[g]$ é definido.

Analogamente ao algoritmo `CalcularCE`, o algoritmo `CalcularCEBin` possui complexidade $O(mn)$, onde n é o número de taxons sob análise e m o número de características binárias.

Inserção e remoção em uma filogenia com representação binária

Procedimento `CalcValorInsercaoBin`(CE_Z, CE_U, CE_V)
Entrada
CE_Z - Vetor CE da raiz (nó z) da subárvore cuja inserção será avaliada;
CE_U - Vetor CE do nó u da aresta (u, v) onde ocorrerá a inserção;
CE_V - Vetor CE do nó v da aresta (u, v) onde ocorrerá a inserção;
Saída
$Custo$ - Custo da inserção;
Início
01. $Custo \leftarrow 0$;
02. **Para** g=1, ..., $m/16$ **faça**
03. **Início**
04. $A \leftarrow CE\_Z[g]$ & $(CE\_U[g] \mid CE\_V[g])$;
05. $Custo \leftarrow Custo +$ `CalcularCusto`(A);
06. **Fim-Para**
07. **Retorne** $Custo$;
Fim-`CalcValorInsercaoBin`

Figura 2.13: Algoritmo para cálculo do custo de inserção.

A Figura 2.13 apresenta o novo algoritmo para o cálculo do custo de uma inserção. O que muda em relação ao algoritmo `CalcValorInsercao` descrito na Figura 2.7 é o trecho nas linhas 2-6. O laço nas linhas 2-6 garante que todos os grupos de 16 características serão analisados. Na linha 4 são detectados os passos evolutivos que não ocorrem e que passarão a ocorrer caso a inserção seja efetivada. A variável $Custo$ incrementada na linha 5 acumula o total de passos evolutivos ocasionados pela inserção.

Analogamente ao algoritmo `CalcValorInsercao`, o algoritmo `CalcValorInsercaoBin` possui complexidade $O(m)$, onde m é o número de características binárias associadas a cada taxon.

A Figura 2.14 apresenta o novo algoritmo para cálculo da redução de passos evolutivos ocasionada por uma remoção. Na linha 1, a variável *Redução* responsável por armazenar a redução total de passos evolutivos ocasionados pela remoção é inicializada. O laço nas linhas 2-27 garante que todos os grupos de 16 características serão analisados. No trecho nas linhas 4-11 são destacados na variável auxiliar C com a seqüência de *bits* 11 os pares de *bits* relativos às características "válidas"de $CEAvo[g]$. Chamam-se de válidas as características de $CEAvo[g]$, descartando-se aquelas que assumem o estado *?* tanto em $CEAvo[g]$ quanto em $CCAuxAvo[g]$. Nas linhas 4 e 5, analogamente ao Exemplo 2, são destacados na variável auxiliar B com a seqüência de *bits* 11 os pares de *bits* relativos às características que assumem o estado *?* em $CEAvo[g]$. Na linha 7, a variável auxiliar C recebe o resultado de uma operação & entre a variável B e $CCAuxAvo[g]$ que destaca em C com a seqüência de *bits* 11 os pares de *bits* que possuem valor 11 tanto em B quanto em $CCAuxAvo[g]$ (os pares de *bits* restantes são representados por 00). Nas linhas 8-9, analogamente ao Exemplo 2, são destacados em C com a seqüência de *bits* 11 os pares de bits relativos às características que assumem o estado *?* tanto em $CEAvo[g]$ quanto em $CCAuxAvo[g]$. Na linha 11 os pares de *bits* relativos às características válidas de $CEAvo[g]$ são destacados na variável auxiliar D. Na linha 13, a variável auxiliar E recebe o resultado de uma operação & entre $CEAvo[g]$ e $CEIrmao[g]$ que atribui a seqüência de *bits* 00 aos pares de *bits* de E relativos às características que em um vetor possuem o estado 0 e no outro o estado 1. Nas linhas 14-15, analogamente ao Exemplo 2, são destacados em E com a seqüência de *bits* 11 os pares de *bits* relativos às características com estado *?* tanto em $CEAvo[g]$ quanto em $CEIrmao[g]$. Nas linhas 17-19, analogamente ao Exemplo 2, são destacados em F com a seqüência de *bits* 11 os pares de *bits* relativos às características de $CENo[g]$ com estado *?*. Nas linhas 21-22, analogamente ao Exemplo 2, são destacados em G com a seqüência de *bits* 11 os pares de *bits* relativos às características de $CEIrmao[g]$ com estado *?*. Na linha 23, a variável G recebe os pares de *bits* de $CEIrmao[g]$, excetuando-se as características com estado *?*. Neste caso, essas características são representadas em G pela seqüência de *bits* 00. Nas linhas 25 e 26 a redução de passos evolutivos para o grupo de 16 características g é calculada de acordo com o número de passos evolutivos detectados pelo procedimento `CalcularCusto` descrito na Subseção 2.4.2. Na linha 28, a redução de passos evolutivos ocasionada pela

Procedimento `CalcValorReduçãoBin`($CENo$, $CEIrmao$, $CEAvo$, $CCAuxAvo$)
Entrada
$CENo$ - Vetor CE da raiz da subárvore cuja remoção será avaliada;
$CEIrmao$ - Vetor CE do nó irmão da raiz da subárvore cuja remoção será avaliada;
$CEAvo$ - Vetor CE do nó avô da raiz da subárvore cuja remoção será avaliada;
$CCAuxAvo$ - Vetor $CCAux$ do nó avô da subárvore cuja remoção será avaliada;
Saída
Redução - Redução de passos evolutivos ocasionados pela remoção;
Início
01. $Redução \leftarrow 0$;
02. **Para** g=1, ..., $m/16$ **faça**
03. **Início**
04. $B \leftarrow ((CEAvo[g]\ \&\ Bits0) << 1)\ \&\ (CEAvo[g]\ \&\ Bits1)$;
05. $B \leftarrow B\ |\ (B >> 1)$;
06.
07. $C \leftarrow B\ \&\ CCAuxAvo[g]$;
08. $C \leftarrow ((C\ \&\ Bits0) << 1)\ \&\ (C\ \&\ Bits1)$;
09. $C \leftarrow C\ |\ (C >> 1)$;
10.
11. $D \leftarrow CEAvo[g]\ \&\ !(C)$;
12.
13. $E \leftarrow CEAvo[i]\ \&\ CEIrmao[g]$;
14. $E \leftarrow ((E\ \&\ Bits0) << 1)\ \&\ (E\ \&\ Bits1)$;
15. $E \leftarrow E\ |\ (E >> 1)$;
16.
17. $F \leftarrow CENo[g]$;
18. $F \leftarrow ((F\ \&\ Bits0) << 1)\ \&\ (F\ \&\ Bits1)$;
19. $F \leftarrow F\ |\ (F >> 1)$;
20.
21. $G \leftarrow ((CEIrmao[g]\ \&\ Bits0) << 1)\ \&\ (CEIrmao[g]\ \&\ Bits1)$;
22. $G \leftarrow G\ |\ (G >> 1)$;
23. $G \leftarrow CEIrmao[g]\ \&\ !(G)$;
24.
25. $A \leftarrow (CENo[g]\ \&\ (G\ |\ D)\ \&\ (!E))\ |\ (E\ \&\ F)$;
26. $Redução \leftarrow Redução +$ `CalcularCusto`(A);
27. **Fim-Para**
28. **Retorne** *Redução*;
Fim-`CalcValorReduçãoBin`

Figura 2.14: Algoritmo para cálculo da redução de passos evolutivos gerada por uma remoção.

remoção é retornada.

Analogamente ao algoritmo `CalcValorRedução`, o algoritmo `CalcValorReduçãoBin` possui complexidade $O(m)$ (verificada no laço nas linhas 2-27), onde m é o número de características binárias associadas a cada taxon.

Experimentos realizados

Para avaliar o efeito da otimização pela estrutura de dados foi utilizado o algoritmo GRASP com vizinhança SPR (esta vizinhança será comentada em detalhes na Seção 3.4) proposto por Andreatta e Ribeiro [3, 4]. Como este algoritmo não estava disponível, o mesmo foi reimplementado de duas maneiras distintas. Na primeira, denominada `GRASP_1`, a implementação foi feita sem a otimização. Na segunda, denominada `GRASP_2`, a otimização da estrutura de dados foi incorporada. Em ambas as implementações a otimização de complexidade descrita na Subseção 2.4.1 foi incorporada. Os experimentos foram feitos para oito instâncias da literatura que estão descritas na Tabela 2.1. Foram realizadas cinco execuções de cada algoritmo para cada instância. Em cada execução foram realizadas 10 iterações.

Instância	`GRASP_1`	`GRASP_2`
ANGI	4,21	2,20
GRIS	4,87	2,57
TENU	15,88	8,15
ETHE	12,70	6,50
ROPA	21,19	10,79
GOLO	23,82	12,17
SCHU	158,86	80,38
CARP	122,33	61,90

Tabela 2.2: Tempo médio de execução dos algoritmos `GRASP_1` e `GRASP_2` em segundos.

Na Tabela 2.2 são apresentados os tempos médios obtidos por cada algoritmo para cada uma das oito instâncias. Os resultados apresentados nesta tabela mostram que a mudança de estrutura de dados para o armazenamento de características gerou uma economia de cerca de 50% no tempo computacional exigido.

2.5 Conclusão

A literatura científica cita a importância do problema da filogenia para a biologia comparativa (sistemática) [6, 71]. O estudo de algoritmos para resolver este problema vem sendo tema de vários trabalhos [3, 4, 18, 19, 37, 48, 52, 57, 58, 70].

O desenvolvimento de algoritmos robustos para o problema da filogenia, que sejam eficientes tanto em qualidade das soluções obtidas quanto no tempo computacional exigido para a obtenção destas, será de grande utilidade. Visando atingir este objetivo, este trabalho incorporou as otimizações sugeridas por Goloboff (Subseções 2.4.1 e 2.4.2) e também as que foram sugeridas nesta tese (Subseção 2.4.1) na implementação de seus algoritmos. O ganho obtido na diminuição da complexidade das operações de inserção e remoção de uma subárvore e o ganho médio de 50% (experimentos realizados na subseção anterior) no tempo computacional obtido pela modificação da estrutura de dados para o armazenamento de características tornarão os algoritmos desenvolvidos nesta tese bastante rápidos.