Matheurísticas para o Problema da Filogenia Viva

Élani Marques Zanlucas

GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
DA FACOM-UFMS

Data da Defesa: 1/12/2023 □

Assinatura:_____

Matheurísticas para o Problema da Filogenia Viva

Élani Marques Zanlucas

Orientador: *Edna Ayako Hoshino*

Monografia apresentada como requisito parcial para aprovação na Componente Curricular Não-Disciplinar Trabalho de Conclusão de Curso do curso de Graduação em Ciência da Computação, da Universidade Federal de Mato Grosso do Sul.

UFMS - Campo Grande
novembro/2023

Resumo

Este trabalho possui o intuito de apresentar uma nova alternativa para resolver o Problema da Filogenia Viva (PFV), utilizando matheurísticas. Para isso, foi reformulado um modelo de programação linear inteira para o problema a fim de minimizar a função objetivo, cujas restrições serão apresentadas adiante. Por conseguinte, foi implementado o algoritmo *Destroy & Repair*, também conhecido como *Large Neighborhood Search* (LNS), a fim de melhorar os resultados do algoritmo *Live Neighbor Joining* (LNJ) de (Telles et al., 2018) utilizado neste trabalho como solução inicial. Ademais, expandimos o LNS em dois casos com o objetivo de avaliar o desempenho entre eles. Baseado nos testes realizados e na avaliação feita com o auxílio do algoritmo *Branch & Bound*, podemos indicar que houve uma melhoria na qualidade das soluções geradas em relação à solução inicial do LNJ.

Sumário

CAPÍTULO

1

Introdução

Neste capítulo, será explicado o que é filogenia, assim como apresentada a justificativa e o objetivo deste trabalho.

1.1 Introdução

Em 1950, o entomólogo alemão Willi Henning apresentou a Sistemática Filogenética ou Filogenia em seu livro *Phylogenetic Systematics* (Hennig, 1999), o qual descreve um método de análise que possibilita recuperar as relações de parentesco entre espécies. Conforme descrito por (Lopes and Vasconcelos, 2012), o resultado das análises forma a árvore filogenética, em que pode-se identificar qual a espécie ancestral e suas respectivas mutações. Com isso, a filogenia desenvolve um importante papel de explicar a diversidade biológica por meio da evolução das características morfológicas, comportamentais, ecológicas, fisiológicas, citogenéticas e moleculares dos grupos, as quais geram as relações de parentesco (Amorim et al., 2001). Por simplicidade, neste trabalho a árvore filogenética será denotada por T.

A filogenia foi introduzida para fins biológicos, porém o método de análise também pode ser utilizado para outras aplicações como inspeção evolutiva de documentos da web. A partir disso, este trabalho procura encontrar mais aplicações além das biológicas para a filogenia. Ademais, para melhor convenção, o termo espécies será substituído por objetos.

1.2 Trabalhos Relacionados

(Telles et al., 2018) apresenta a heurística *Live Neighbor Joining* (LNJ) para resolver o problema da filogenia viva aplicado em datasets de vírus, bactérias,

conjunto de imagens e textos.

O uso de um algoritmo genético foi proposto por (Almeida and Walter, 2018), o qual é uma versão extendida dos algoritmos Fitch e Sankoff para resolver o problema da filogenia viva.

(Papamichail et al., 2017) apresentam uma resolução do problema da filogenia viva com politomia utilizando dois algoritmos de Branch and Bound: *Mixed Tree Enumeration Algorithm* (MTEA) e *Cubic Tree Enumeration and Edge Contraction Algorithm* (CTEECA).

1.3 Motivação

Para melhor compreensão do trabalho proposto, uma heurística consiste em técnicas para otimizar o desempenho de métodos de solução de problemas, de acordo com seus critérios especificados. Assim temos que a matheurística é uma abordagem híbrida que combina métodos heurísticos e algoritmos matemáticos para resolver problemas de otimização complexos.

Este trabalho foi desenvolvido como pioneiro na área de matheurística para o problema abordado, visto que não foi encontrado na literatura atual algum trabalho resolvendo o problema da filogenia viva utilizando matheurísticas. Além disso, a existência de um algoritmo polinomial para a filogenia depende das restrições impostas sobre a estrutura da árvore e de sua respectiva modelagem, de modo que este problema possa ser classificado como NP-completo. Por conta disso, é preciso encontrar um método heurístico para resolvê-lo.

1.4 Objetivos

Como objetivo principal temos o intuito de melhorar os resultados do trabalho desenvolvido por (Telles et al., 2018), aplicando a matheurística nas soluções já construídas. Adicionalmente, propomos um novo modelo de programação linear inteira para o problema.

CAPÍTULO

2

Descrição Formal do Problema

Neste capítulo, descrevemos formalmente o problema estudado e apresentamos um modelo de programação linear inteira para o problema.

2.1 Problema da Filogenia

Conforme descrito por (Lenzini and Marianelli, 1997), dado um conjunto S de objetos, um grafo não orientado $T = (V, E)$ é uma **árvore filogenética** se:

- $S \subseteq V$, em que S são as folhas de T e os nós em $V - S$ representam as espécies extintas;
- E é um conjunto $(i, j) : i, j \in V$ de arestas, em que se j for descendente direto de i, existe uma aresta $(i, j) \in E$ conectando os vértices.

Dada uma árvore filogenética T com pesos nas arestas, a **distância** D_{ij}^T **entre os vértices** i **e** j **em** T é o comprimento do único caminho em T que conecta i e j, o qual é calculado como a soma dos pesos das arestas ao longo do caminho.

Dados um conjunto S de n objetos e uma matriz de distância M, no qual $M[i, j]$ representa a semelhança ou a distância genética entre os objetos i e j, o **problema da filogenia perfeita** (FPP) visa encontrar uma árvore filogenética T com pesos nas arestas de modo que $D_{ij}^T = M[i, j]$, para todos os objetos i e j em S. Desta forma, espera-se que objetos semelhantes (conforme o critério estabelecido) possuam um grau de parentesco maior, e assim, fiquem mais próximos entre si na árvore.

Quando M não admite uma árvore T que resolva o FPP, considera-se um problema aproximado, descrito a seguir.

Dados um conjunto S de n objetos e uma matriz de distância M, no qual $M[i,j]$ representa a semelhança ou a distância genética entre os objetos i e j, o **problema da filogenia** (FP) visa encontrar uma árvore filogenética T com pesos nas arestas que minimiza a soma da diferença entre D_{ij}^{T} e $M[i,j]$, para todo par de objetos i e j em S.

	0	1	2	3	4	5	6	7	8	9
0	0.00	94.00	57.00	64.00	89.00	73.00	74.00	53.00	26.00	50.00
1	94.00	0.00	99.00	106.00	35.00	67.00	20.00	95.00	68.00	44.00
2	57.00	99.00	0.00	15.00	94.00	78.00	79.00	4.00	31.00	55.00
3	64.00	106.00	15.00	0.00	101.00	85.00	86.00	11.00	38.00	62.00
4	89.00	35.00	94.00	101.00	0.00	62.00	15.00	90.00	63.00	39.00
5	73.00	67.00	78.00	85.00	62.00	0.00	47.00	74.00	47.00	23.00
6	74.00	20.00	79.00	86.00	15.00	47.00	0.00	75.00	48.00	24.00
9	53.00	95.00	4.00	11.00	90.00	74.00	75.00	0.00	27.00	51.00
8	26.00	68.00	31.00	38.00	63.00	47.00	48.00	27.00	0.00	26.40
7	50.00	44.00	55.00	62.00	39.00	23.00	24.00	51.00	26.40	0.00

Tabela 2.1: Matriz de distância M

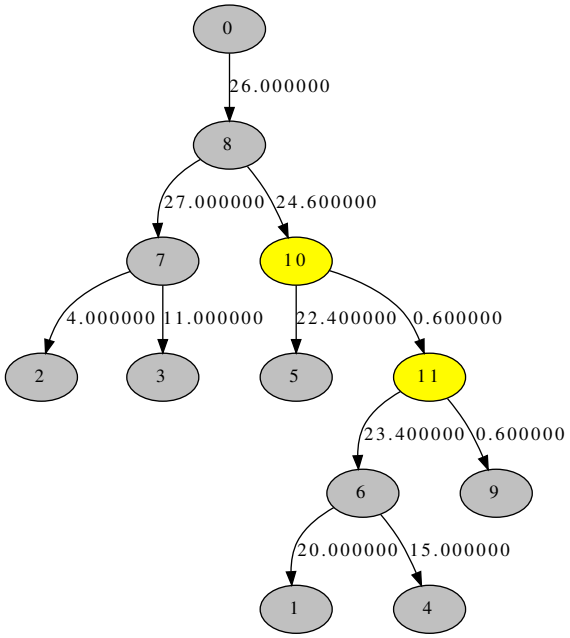


Figura 2.1: Árvore filogenética

A Tabela 2.1 apresenta uma matriz de distância de um dos dados utilizados como teste seguido de sua respectiva árvore filogenética da Figura 2.1, construída a partir do algoritmo implementado. Vale observar que a matriz de distância é simétrica e a árvore filogenética apresentada na Figura 2.1 aparece com a raiz no objeto 0 e orientada, devido à modelagem de programação linear inteira, que será descrita na Seção 2.3. A orientação não altera a definição do problema, mas neste caso, deve-se considerar o comprimento do caminho não orientado entre cada par de objetos na árvore.

2.2 Problema da Filogenia Viva

O **problema da filogenia viva** (PFV) é a especificação do problema da filogenia, o qual admite a existência de ancestrais vivos (Telles et al., 2013), isto é, admite-se que objetos taxonômicos ocorram na árvore como nós internos. Conforme introduzido acima, a instância I do problema inclui a matriz de distância M e o conjunto S.

A seguinte notação e conceitos são adotados no restante do texto:

- V denota o conjunto dos vértices que necessariamente devem ocorrer na árvore e representam os objetos em S;
- H é o conjunto dos vértices hipotéticos que podem ser usados na árvore;
- V' é o conjunto dos vértices necessários e dos hipotéticos;
- $\delta^-(i)$ arestas que entram no vértice i;
- $\delta^+(i)$ arestas que saem do vértice i;
- fluxo é dado por um caminho a partir da raiz até o vértice determinado, no qual este fluxo é consumido, isto é, o fluxo não continua para os próximos vértices;
- último ancestral comum é o último vértice no qual passa um fluxo comum entre i e j.

Consideramos um objeto arbitrário para representar a raiz r da árvore filogenética. Vale observar que isso não altera o objetivo do PFV e a solução do problema.

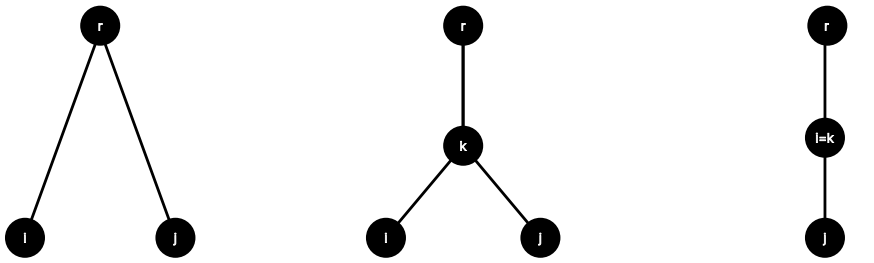
2.3 Modelagem para o Problema da Filogenia Viva

Dada uma instância I do problema da filogenia viva, as seguintes variáveis de decisão são usadas para descrever uma solução (T, c), no qual T denota a árvore filogenética e c a função de distância dos objetos em T:

- c_{ij} (D_{ij}^T) representa a distância do caminho entre i e j em T;
- d_{ij} é a diferença em valor absoluto entre M_{ij} e c_{ij};
- y_a^k é o fluxo no arco a para o vértice k;
- $\hat{y}_a^{ij}$ é o fluxo no arco a comum para i e j;
- g_k^{ij} indica que k é o último ancestral comum entre i e j;

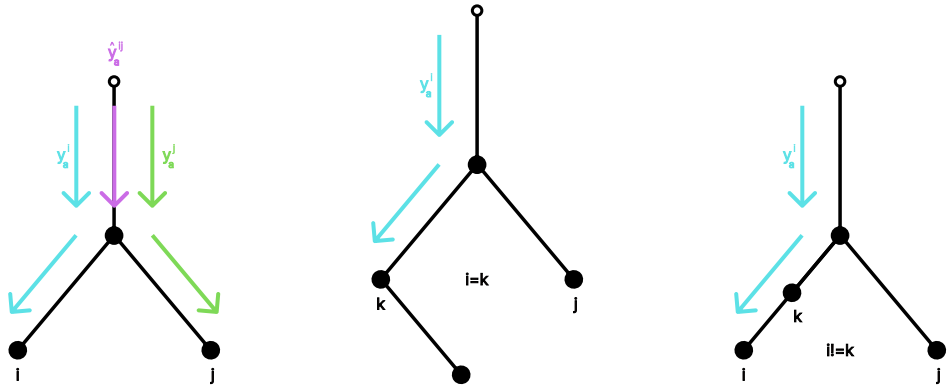
- z_i indica se i é um vértice interno em T;
- x_a é igual a 1 se, e somente se, a aresta a for pertencente à árvore.

Para melhor compreensão visual do último ancestral comum e de controle de fluxo, foram ilustrados nas Figuras (2.2) e (2.3) todos os possíveis casos de cada conceito com suas respectivas explicações.



(a) raiz r (b) $k \neq r$ (c) $k = i$

Figura 2.2: Possíveis casos do último ancestral comum para um par de objetos i e j. A Figura 2.2(a), ilustra o caso em que a raiz é o último ancestral comum. Em 2.2(b), o último ancestral comum é um vértice k qualquer, diferente da raiz. Em 2.2(c), o último ancestral comum de i e j é o próprio i, já que $k = i$, ou seja, i é o "pai"de j. Note que o inverso também pode ocorrer, ou seja, j ser o último ancestral comum de i e j.



(a) fluxo para i e j (b) $i = k$ (c) $i \neq k$

Figura 2.3: Grafos de controle de fluxo. Em 2.3(a), y_a^{ij} gera fluxo para y_a^i e y_a^j. Assim, quando esses fluxos chegam respectivamente em i e j, são consumidos, isto é, estes fluxos chegam nestes vértices, mas não saem. A Figura 2.3(b), apresenta o caso em que o fluxo y_a^i é consumido pelo vértice k, já que $i = k$. Em 2.3(c), o fluxo y_a^i chega em k e sai sem ser consumido, pois $i \neq k$.

Baseado nas variáveis introduzidas acima e no modelo descrito por (Hoshino et al., 2018), o novo modelo de programação linear inteira para o problema da filogenia viva pode ser definido da seguinte forma:

$$Min \quad \sum_{i \in V} \sum_{j \in V} d_{ij}$$

$$s.a.$$

$$\sum_{a\in\delta^-(i)} y_a^k - \sum_{a\in\delta^+(i)} y_a^k = \begin{cases} 1, & \text{se } i = k, i \neq r \quad (2.1) \\ 0, & \text{se } i \neq k \quad (2.2) \end{cases}$$

$$y_a^k \leq x_a, \quad \forall a \in V' \times V', \quad \forall k \in V \quad (2.3)$$

$$\sum_{a\in\delta^-(i)} x_a = \begin{cases} 0, & \text{se } i = r \quad (2.5) \\ z_i, & \text{se } i \in H \quad (2.6) \end{cases}$$

$$\sum_{a\in\delta^+(i)} x_a = \begin{cases} 1 + 2z_i, & \text{se } i = r \quad (2.7) \\ 2z_i, & \text{se } i \in V \cup H \quad (2.8) \end{cases}$$

$$d_{ij} \leq M_{ij} - c_{ij}, \quad \forall i, \forall j \in V \quad (2.9)$$
$$d_{ij} \leq c_{ij} - M_{ij}, \quad \forall i, \forall j \in V \quad (2.10)$$

$$c_{ij} = c_{ri} + c_{rj} - 2 \sum_{k\in V'} c_{rk}.g_k^{ij}, \quad \forall i, \forall j \in V \quad (2.18)$$

$$\sum_{k\in V'} g_k^{ij} = 1, \quad \forall i, \forall j \in V \quad (2.19)$$

$$\left.\begin{aligned} \hat{y}_a^{ij} &\leq y_a^i \quad (2.20) \\ \hat{y}_a^{ij} &\leq y_a^j \quad (2.21) \\ \hat{y}_a^{ij} &\geq y_a^i + y_a^j - 1 \quad (2.22) \end{aligned}\right\}, \forall i, \forall j \in V, \quad \forall a \in V' \times V'$$

$$g_k^{ij} = \sum_{e\in\delta^-(k)} \hat{y}_e^{ij} - \sum_{e\in\delta^+(k)} \hat{y}_e^{ij}, \quad \forall i, \forall j \in V, \quad \forall k \in V', \quad k \neq r \quad (2.23)$$

$$x_a, z_i, y_a^k, g_k^{ij}, \hat{y}_a^{ij} \in \{0,1\} \quad (2.24)$$

$$c_{ij}, d_{ij} \in \mathbb{R}^+ \quad (2.25)$$

O sistema composto por (2.1) e (2.2) representa restrições de fluxo. A expressão (2.3) garante a existência do fluxo para o vértice (caso $y_a^k = 1$). Os sistemas formados pelas equações (2.5) a (2.8) verificam se i é um vértice interno, a partir das arestas que saem e que entram, particularizando caso seja raiz. As inequações (2.9) e (2.10) garantem a distância absoluta, impedindo de ser negativa. A restrição (2.18) informa o resultado do custo c_{ij} para as três possibilidades de ancestral comum discutidas. O somatório (2.19) estabelece que k é o último ancestral comum entre os vértices i, j. O sistema constituído

pelas expressões (2.20), (2.21) e (2.22) refere-se ao controle de fluxo comum (entre i e j) para a árvore orientada. A restrição (2.23) apresenta a definição do último ancestral comum. As restrições (2.24) a (2.25) representam o domínio das variáveis.

CAPÍTULO

3

Metodologia

Neste capítulo, será introduzida a técnica de programação linear, a matheurística utilizada para resolver o problema mostrado no Capítulo 2, com seu respectivo pseudocódigo, assim como o algoritmo para avaliação da melhoria.

3.1 PL e PLI

A **Programação Linear** (PL) é um método de otimização matemática utilizado para encontrar uma solução ótima, buscando maximizar ou minimizar uma função objetivo sujeita a restrições lineares.

Enquanto isso, **Programação Linear Inteira** (PLI) é uma especialização da PL, na qual as variáveis de decisão são restritas a valores inteiros ou binários, tornando o problema mais complexo.

Em suma, ambas as técnicas são modelagens matemáticas que possuem uma função objetivo com restrições para tomada de decisões eficientes.

3.2 Large Neighborhood Search

Inicialmente aplicado ao Problema de Roteamento de Veículos, o *Large Neighborhood Search* (LNS) proposto por (Shaw, 1998) apresenta uma busca baseada em árvore com restrição de propagação, a qual possibilita alterar partes da solução que estejam "ruins". Trata-se de um processo contínuo de *destroy* (remoção) e *repair* (reconstrução) de uma solução inicial a fim de aprimorar os resultados posteriores. A partir disso, é possível aproximar-se de uma solução ótima.

O algoritmo de *destroy* tem como objetivo remover uma parte considerada ruim da solução, com base nos critérios de remoção. Para este trabalho, foram

elaboradas duas formas de *destroy*: remoção dos vértices e remoção das arestas, as quais serão detalhadas posteriormente. O algoritmo de *repair* tem o intuito de consertar a solução temporária do *destroy*, a fim de deixá-la viável, conforme os parâmetros determinados. Um método de otimização é utilizado para reconstruir a árvore. Assim como o *destroy*, há várias maneiras de implementar o *repair*. Para este trabalho, será utilizado o mesmo modelo de PLI apresentado na seção 2.3.

3.3 Pseudocódigo

Conforme descrito por (Pisinger and Ropke, 2019), seja x a solução atual, x^b a melhor solução encontrada durante a busca e x^t uma solução temporária. Para as funções, considere $destroy(.)$ como o método destroy, $repair(.)$ como o método repair, $accept(.,.)$ como um método que aceita apenas soluções melhoradas, $c(.)$ como o custo da solução. A partir das definições acima, tem-se o Algoritmo 1 como pseudocódigo:

Algorithm 1 Large Neighborhood Search

Input: x ▷ solução viável
$x^b = x$
repeat
  $x^t = repair(destroy(x))$
  if $accept(x^t, x)$ **then**
    $x = x^t$
  end if
  if $c(x^t) < c(x^b)$ **then**
    $x^b = x^t$
  end if
until critério de parada
return x^b

Baseado no Algoritmo 1, a solução viável inicial sofre a função de destruição seguido da reparação para encontrarmos uma solução temporária, a qual caso seja uma solução melhorada, avaliada pelo método de aceitação, torna-se a solução atual para o laço em questão.

Ademais, se o custo da solução temporária for menor que o custo da melhor solução, esta torna-se a solução atual. O laço é finalizado por meio de um critério de parada, podendo ser um número máximo de iterações, atingir um ótimo local, dentre outras. O critério de parada utilizado neste trabalho será abordado posteriormente.

3.4 Branch and Bound

O algoritmo *Branch and Bound* (B&B) é uma técnica de otimização utilizada para resolver problemas de busca. Seu princípio é dividir o espaço de busca em subproblemas menores, explorando cada subproblema e, ao mesmo tempo, usar limites superiores e inferiores para determinar se é viável continuar a exploração, fazendo com que reduza o número de cálculos desnecessários em problemas complexos e torne a busca por soluções mais eficiente. Este processo pode ser dividido em 5 passos:

1. Divisão: O problema inicial é dividido em subproblemas menores, criando um ramo para cada subproblema, como uma árvore.

2. Avaliação: Os limites superiores e inferiores são calculados para cada nó da árvore. O limite inferior é uma estimativa do melhor valor possível para o subproblema, enquanto o limite superior é uma estimativa do valor máximo que pode ser alcançado.

3. Podagem: Os ramos com limites superiores menores do que o melhor resultado já encontrado são descartados.

4. Seleção: O próximo nó a ser explorado é escolhido com base em algum critério, como o limite inferior mais alto.

5. Repetição: Os passos 2 a 4 são retomados até que todos os ramos tenham sido explorados ou até que um resultado satisfatório seja encontrado.

CAPÍTULO

4

Trabalho Desenvolvido

Neste capítulo, serão apontadas as melhorias idealizadas para o LNS, incluindo os pseudocódigos produzidos.

4.1 LNS

Inicialmente, foram utilizadas as soluções da heurística de (Telles et al., 2018) como solução inicial do LNS e discutidas duas possibilidades de destruição de uma árvore T. Para fins didáticos, dado um grafo G, denotaremos por $V(G)$ o conjunto de vértices de G e por $E(G)$ o conjunto de arestas de G. O grau de um vértice v em G é denotado por $g_T(v)$, o custo do caminho (u,v) é p_{uv} e a matriz de distância é M_{uv}.

4.1.1 Fase de Destruição - Caso 1

O critério de destruição considerado neste caso consiste em remover um percentual dos objetos, cuja distância aos demais objetos em T é muito diferente da matriz de distância M. Se o objeto a ser removido é um vértice interno, substitui-se por um novo vértice hipotético. Essa ideia não altera a distância entre os objetos que ficaram em T. Baseado em (Lutz, 2015) para a construção dos pseudocódigos de destroy, tem-se:
Seja $x = (T, p)$ a solução atual, em que T é a árvore filogenética e p é o vetor de custos dos caminhos das arestas em T. A instância $I = (S, M)$ é composta pelo conjunto de objetos S e a matriz de distância M. Seja r uma taxa limite para a tolerância de erro, $iter$ o número de iterações, $maxiter$ o número máximo de iterações, R^- a lista de vértices removidos da árvore T, g_T o grau do vértice, C a lista de candidatos a serem removidos da árvore, u e v vértices de T, h um vér-

tice hipotético, x_t uma solução parcial temporária. Para a lista de candidatos C, temos duas formas de inicializá-la, as quais serão separadas em 2 critérios, denominados C_1 e C_2. A partir do caso 1 apresentado, tem-se o Algoritmo 2 como pseudocódigo:

Algorithm 2 Destroy - caso 1

Input: solução $x = (T, p)$**, instância** $I = (S, M)$**, taxa** r**, maxiter**
Output: lista de vértices R^-**, solução temporária** x_t

1: $R^- = \emptyset$
2: $C_1 = \{v \in S : \exists u \in S, |p_{uv} - M_{uv}| > r\}$
3: $C_2 = \{v \in S : \sum_{u \in S} |p_{uv} - M_{uv}| > r\}$
4: $C = selecionaC(C_1, C_2)$
5: $iter = 1$
6: $x_t = x$
7: **while** $|C| > 0$ **and** $iter \leq maxiter$ **do**
8: $v = random(C)$
9: **if** $g_T(v) > 1$, i.e., v é um vértice interno em T **then**
10: Substitua v por um novo vértice hipotético h em T
11: **else**
12: $V(T) = V(T) - \{v\}$
13: $E(T) = E(T) - \{(u, v)\}$
14: $g_T(u) = g_T(u) - 1$
15: **end if**
16: $R^- = R^- \cup \{v\}$
17: $C = C - \{v\}$
18: $iter = iter + 1$
19: atualiza x_t
20: **end while**
21: **return** R^-, x_t

Conforme o Algoritmo 2, um conjunto de candidatos é estabelecido a partir de uma taxa, a qual é comparada com a diferença absoluta entre o custo do caminho das arestas e a matriz de distância. A solução temporária é inicializada, onde na primeira iteração recebe a solução do algoritmo LNJ. Após isso, o algoritmo entra em um laço, onde um candidato do conjunto é escolhido baseado nos critérios definidos, verificado para determinar se é um vértice interno, substituído, caso necessário, e removido. A saída do algoritmo é a lista de vértices removidos e uma solução temporária x_t, a qual não possui os vértices em R^-.

4.1.2 Fase de Destruição - Caso 2

O critério de destruição do segundo caso consiste em remover um percentual das arestas de T, obtendo uma floresta F de modo que cada árvore em F seja uma "boa" filogenia (que a distância entre o custo do caminho entre as

arestas na árvore seja similar à matriz de distância M, de acordo com uma taxa de qualidade Δ).
Seja $x = (T, p)$ a solução atual, em que T é a árvore filogenética e p é o vetor de custos dos caminhos das arestas em T. A instância $I = (S, M)$ é composta pelo conjunto de objetos S e a matriz de distância M. Seja $iter$ o número de iterações, $maxiter$ o número máximo de iterações, Δ uma taxa de qualidade mínima para a árvore, F uma floresta (lista de árvores), C um conjunto de árvores candidatas à remoção, T', T_u e T_v árvores auxiliares, d a diferença absoluta entre a matriz de distância e os custos dos caminhos entre os objetos que ocorrem na árvore.

Dada uma árvore T, denotamos por $V(T)$ o conjunto dos vértices de T. Dado um par de vértices i e j de uma árvore T, denotamos por P_{ij} o custo do único caminho em T que liga i e j.

Algorithm 3 Destroy - caso 2

Input: solução $x = (T, p)$**, instância** $I = (S, M)$**, maxiter,** Δ
Output: floresta F

1: $F = \emptyset$
2: $C = \{T\}$
3: **while** $|C| \neq 0$ **and** $iter \leq maxiter$ **do**
4: $T' = selecionaT(C)$
5: $(u, v) = selecionaE(T')$
6: Seja T_u e T_v as árvores obtidas ao remover a aresta (u, v) da árvore T'
7: $d = \sum_{i,j \in S \cap V(T_u)} |M_{ij} - P_{ij}|$
8: **if** $d \leq \Delta$ **then**
9: $F = F \cup \{T_u\}$
10: **else**
11: $C = C \cup \{T_u\}$
12: **end if**
13: $d = \sum_{i,j \in S \cap V(T_v)} |M_{ij} - P_{ij}|$
14: **if** $d \leq \Delta$ **then**
15: $F = F \cup \{T_v\}$
16: **else**
17: $C = C \cup \{T_v\}$
18: **end if**
19: $iter = iter + 1$
20: **end while**
21: $F = F \cup C$
22: **return** F

Conforme visto no Algoritmo 3, seleciona-se uma árvore dentre o conjunto de candidatas para a remoção e retira-se uma aresta selecionada da árvore T', resultando em duas árvores, sendo T_u e T_v. A partir disso, verifica-se se ambas as árvores são consideradas uma boa filogenia, isto é, caso o somatório da di-

ferença entre os custos dos caminhos entre os objetos que estão na árvore com a matriz de distância M seja $\leq \Delta$. Em caso afirmativo, a árvore é adicionada à floresta que contém boas filogenias. Caso contrário, a árvore é adicionada à lista de candidatas. O laço que envolve este algoritmo finaliza uma vez que não existam mais candidatas para a remoção ou que o $iter$ atingiu o número máximo de iterações. O retorno é uma floresta F que une as árvores com filogenia boa e as árvores com filogenia ruim restantes. O critério estabelecido para selecionar uma árvore em C é dado por uma ordenação decrescente de d com o intuito de remover inicialmente as piores filogenias. Ademais, o critério determinado para selecionar uma aresta em T' é a aleatoriedade.

4.1.3 Fase de Recuperação

Em relação ao *repair*, a ideia é reutilizar o mesmo modelo de PLI criado com as variáveis fixadas, conforme definido na Seção 2.3. Posteriormente, avalia-se uma melhoria utilizando o algoritmo B&B da seção 3.4.

CAPÍTULO

5

Resultados Computacionais

Neste capítulo, serão indicados as características das instâncias, o ambiente em que foi compilado o código e os resultados apurados a partir do programa desenvolvido.

5.1 Instâncias

Os testes foram realizados em um grupo de 23 instâncias contendo de 8 a 10 objetos.

5.2 Ambiente computacional

Todos os testes foram realizados em um máquina com processador Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz com 8GB de memória RAM e sistema operacional Linux. Os algoritmos foram implementados na linguagem C utilizando a biblioteca de otimização SCIP, versão 3.2.1, com o resolvedor Soplex.

5.3 Resultados

Os resultados são apresentados na Tabela 5.1, na qual a coluna `tempo` indica o total de tempo gasto, em segundos, `LB` o valor do limitante primal gerado e o total de nós explorados, respectivamente, por cada matheurística. Nos experimentos, limitamos o tempo total de processamento do LNS em 60 segundos. O símbolo $*$ é usado para denotar os casos em que o tempo limite é alcançado pela heurística e ela é finalizada. O percentual de destruição considerado foi de 30%. A coluna LNS 10s representa os resultados nos quais

limitamos o tempo total de processamento do LNS em 10 segundos, o qual foi utilizado para avaliar o impacto em reduzir o tempo de processamento do LNS.

instância	n	LNJ	LNS	LNS 10s	tempo LNS	tempo LNS 10s
d_6_2_1_10	8	58,8	47,2	47,2	22,7	19,9
d_6_2_1_1	8	23,9	5,8	5,8	44,9	24,7
d_6_2_1_2	8	43,2	13,3	13,3	*	36,7
d_6_2_1_3	8	42,6	15	15	15,4	15,4
d_6_2_1_4	8	54,2	20,8	20,8	*	53,9
d_6_2_1_5	8	56,4	26,4	26,4	51	30,7
d_6_2_1_6	8	53,9	33,7	33,7	*	42
d_6_2_1_7	8	74,5	40,1	40,1	*	54,6
d_6_2_1_8	8	66,3	41,4	41,4	46,1	26
d_6_2_1_9	8	61,8	44,4	44,4	44,2	23,9
d_6_4_1_10	10	147,8	147,8	147,8	*	*
d_6_4_1_1	10	19,2	19,2	19,2	*	*
d_6_4_1_2	10	15,9	15,9	15,9	*	*
d_6_4_1_3	10	84	84	84	*	*
d_6_4_1_4	10	63,3	63,3	63,3	*	*
d_6_4_1_5	10	127,2	127,2	127,2	*	*
d_6_4_1_6	10	121,8	121,8	121,8	*	*
d_6_4_1_7	10	130	130	130	*	*
d_6_4_1_8	10	129,9	129,9	129,9	*	*
d_6_4_1_9	10	147,9	147,9	147,9	*	*

Tabela 5.1: Resumo comparativo dos testes computacionais

Analisando-se a tabela, nota-se que houve uma melhoria na qualidade das soluções d_{ij} geradas ao aplicar o LNS nas soluções geradas por LNJ. A redução de d_{ij} para as instâncias testadas ocorreu apenas para $n = 8$, gerando uma média de 49%. Comparando-se o impacto ao reduzir o tempo de processamento máximo do LNS, a cada iteração, em 10 segundos, nota-se que não houve redução na qualidade das soluções geradas e adicionalmente diminui, como esperado, o tempo total de processamento do LNS. Essa redução foi, em média, de 14 segundos.

CAPÍTULO

6

Contribuições e Conclusões

Neste capítulo, comparamos se os resultados obtidos foram os esperados conforme as hipóteses datadas.

6.1 Contribuições

Este trabalho trata-se de uma extensão do trabalho feito por (Telles et al., 2018), no qual aproveitamos os resultados das soluções apresentadas para utilizar de base na matheurística implementada, cujo objetivo era melhorar esta solução.

Para trabalhos futuros, temos a opção de implementar e avaliar o critério de destruição Caso 1 do *destroy*, descrito no Algoritmo 2, assim como testar em outras instâncias ambos os casos de *destroy*.

6.2 Conclusões

Em linhas gerais, ao reduzir o tempo de processamento, o LNS continuou gerando soluções com qualidade e melhorando o valor da solução ao diminuí-lo em comparação com o LNJ. Isso já nos indica que o Algoritmo 3 da matheurística se comportou de forma positiva para as instâncias utilizadas. Em contrapartida, conseguimos ver melhoria apenas para as menores instâncias, o que nos mostra que é preciso aperfeiçoar o caso 2 do *destroy* para obter melhores resultados.

Referências

Almeida, N. F. e Walter, M. E. M. (2018). A genetic algorithm for character state live phylogeny. In *Advances in Bioinformatics and Computational Biology: 11th Brazilian Symposium on Bioinformatics, BSB 2018, Niterói, Brazil, October 30–November 1, 2018, Proceedings*, volume 11228, pagina 114. Springer. Citado na página 2.

Amorim, D. d. S., Montagnini, D. L., Correa, R. J., Castilho, M. S. M., e Noll, F. B. (2001). *Diversidade biológica e evolução: uma nova concepção para o ensino de zoologia e botânica no 2º grau*. Holos. Citado na página 1.

Hennig, W. (1999). *Phylogenetic systematics*. University of Illinois Press. Citado na página 1.

Hoshino, E. A., Araujo, B. A., e Freitas, V. O. (2018). Algoritmos exatos para o problema da filogenia viva. *XIX Latin-Iberoamerican Conference on Operations Research*, pginas 276–283. Citado na página 6.

Lenzini, G. e Marianelli, S. (1997). Algorithms for phylogeny reconstruction in a new mathematical model. *Calcolo*, 34:1–24. Citado na página 3.

Lopes, W. R. e Vasconcelos, S. D. (2012). Representação e distorções conceituais do conteúdo filogenia em livros didátícos de biologia do ensino médio. *Ensaio Pesquisa em Educação em Ciências (Belo Horizonte)*, 14:149–165. Citado na página 1.

Lutz, R. (2015). Adaptive large neighborhood search. Citado na página 13.

Papamichail, D., Huang, A., Kennedy, E., Ott, J.-L., Miller, A., e Papamichail, G. (2017). Live phylogeny with polytomies: Finding the most compact parsimonious trees. *Computational Biology and Chemistry*, 69:171–177. Citado na página 2.

Pisinger, D. e Ropke, S. (2019). Large neighborhood search. *Handbook of metaheuristics*, pginas 99–127. Citado na página 10.

Shaw, P. (1998). Using constraint programming and local search methods

to solve vehicle routing problems. In *Principles and Practice of Constraint Programming—CP98: 4th International Conference, CP98 Pisa, Italy, October 26–30, 1998 Proceedings 4*, pginas 417–431. Springer. Citado na página 9.

Telles, G. P., Almeida, N. F., Minghim, R., e Walter, M. E. M. (2013). Live phylogeny. *Journal of Computational Biology*, 20(1):30–37. Citado na página 5.

Telles, G. P., Araujo, G. S., Walter, M. E., Brigido, M. M., e Almeida, N. F. (2018). Live neighbor-joining. *BMC bioinformatics*, 19(1):1–13. Citado nas páginas v, 1, 2, 13, e 19.