Tutorial 5

TNT - Buscas Avançadas

BIZ0433 - INFERÊNCIA FILOGENÉTICA: FILOSOFIA, MÉTODO E APLICAÇÕES.

Conteúdo

Objetivo

Este tutorial visa apresentar as ferramentas mais avançadas de busca em TNT. Estas ferramentas são os algoritmos mais poderosos de TNT e garantem a esse programa muita eficiência em explorar o espaço de árvores, especialmente em bancos de dados complexos (*i.e.*, grande número de terminais e caracteres). O tutorial explica brevemente os métodos implementados em TNT. No entanto, o componente teórico contido neste tutorial é superficial e o estudante é encorajado a consultar a literatura primária citada neste documento. Este tutorial também contém exercícios sobre reconstrução de estados ancestrais em topologias, bem como os comandos associados à diagnose de grupos em TNT. Eles são importantes para que o estudante tenha conhecimento de como explorar evolução de caracteres, bem como identificar os caracteres que sustentam grupos em sua análise usando este aplicativo. Os arquivos associados a este tutorial estão disponíveis no GitHub. Você baixar todos os tutoriais com o seguinte comando:

```
svn checkout https://github.com/fplmarques/cladistica/trunk/tutorials/
```

5.1 Buscas com novas tecnologias

A maior vantagem do TNT [1] em comparação a muitos programas de inferência filogenética é a rapidez com a qual este aplicativo executa buscas. Parte da velocidade do TNT está no código de execução de tarefas, mas há um outro componente que é resultado da implementação de novos algoritmos de buscas [2, 3]. No tutorial anterior vimos como as buscas tradicionais são feitas por algoritmos que começam com a construção de árvores de Wagner e subsequente refinamento por *branch swapping* (*i.e.*, RAS+SPR e/ou TBR, veja Tutorial 4, seção 4.7). Estas estratégias de busca por trajetórias são considerados "*Hill-climbing algorithms*" uma vez que eles departem de uma topologia inicial em direção a outra melhor utilizando rearranjos internos (*i.e., intra-tree branch swapping*, veja Giribet [4] e Goloboff [2]). Como pôde ser visto no tutorial anterior, esses algoritmos possuem limitações, principalmente em espaços de árvores complexos, pois eles podem restringir suas buscas a áreas nas quais se encontram apenas ótimos locais.

5.1.1 Perturbações:

De acordo com Giribet [4], uma das estratégias mais inovadores de busca utilizando os algoritmos de rearranjos internos disponíveis naquele momento foi a técnica de *ratchet* proposta por Nixon [3]. Esta técnica de aceleração de buscas heurísticas está implementada em TNT. A técnica de *ratchet* usa ciclos de perturbações no espaço de topologias atribuindo pesos diferenciais à parte dos dados na expectativa de que a busca saia de ótimos locais (Figura 5.1). Nesse algoritmo, uma topologia é criada e refinada pelos algoritmos convencionais (*i.e.*, RAS+TBR), ao término do rearranjo uma porcentagem dos dados recebe pesagens diferenciais, um novo ciclo de rearranjos é iniciado, após sua conclusão os caracteres voltam à pesagem original e o custo da topologia é avaliado. Se há mais ciclos a serem executados, esses passos são repetidos, caso contrário o custo da topologia resultante é comparado com a topologia inicial (Figura 5.1).

5.1.2 Buscas setoriais:

Outro conjunto de algoritmos disponíveis em TNT para explorar espaços de árvores mais complexos são aqueles destinados à buscas setoriais (*sectorial search*, senso Goloboff [2]). Como o próprio nome indica, estes algoritmos – conhecidos como algoritmos da família "*divide and conquer*" – reduzem a dimensão do espaço de soluções restringindo o problema à subconjuntos de problemas menores. No caso de buscas filogenéticas, o algoritmo implementado em TNT seleciona um setor (*i.e.*, clado) da topologia que é reanalisado. Se uma configuração melhor é encontrada para esse clado, ele é substituído na topologia original (para maiores detalhes veja Goloboff [2] e Wheeler [5]). Os ciclos de iterações desse algoritmo está representado na Figura 5.2.

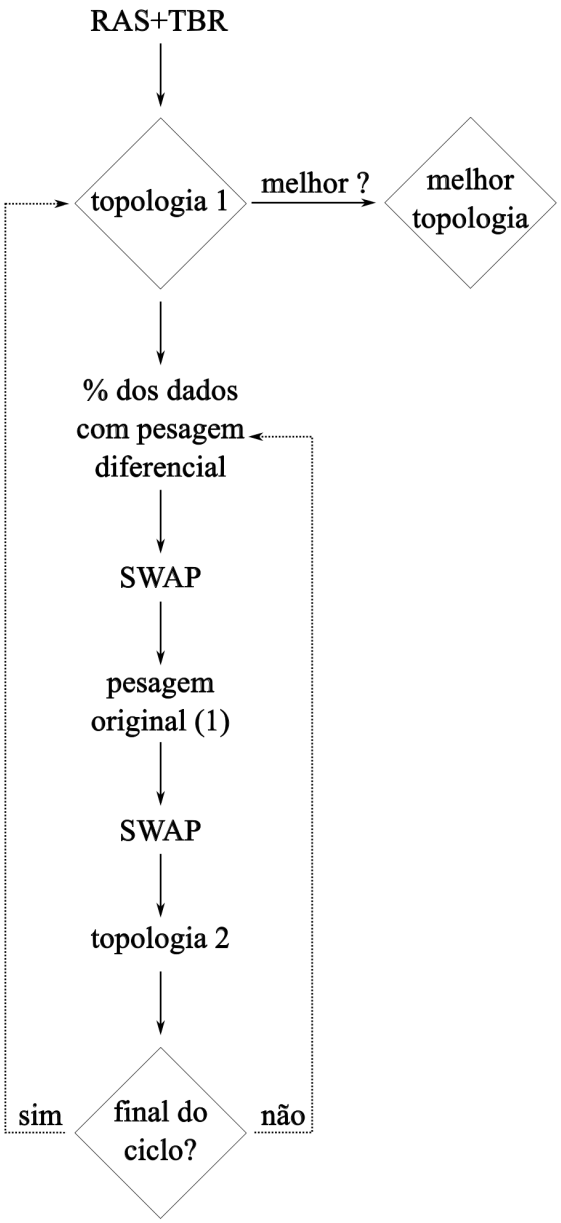

Figura 5.1: Fluxo de iterações de *ratchet*. **RAS**, *radom addition sequence*; **SWAP**, *branch swapping* via SPR ou TBR. Pesagem diferencial geralmente entre 5 e 10% dos caracteres. Número de ciclos (k) variável.

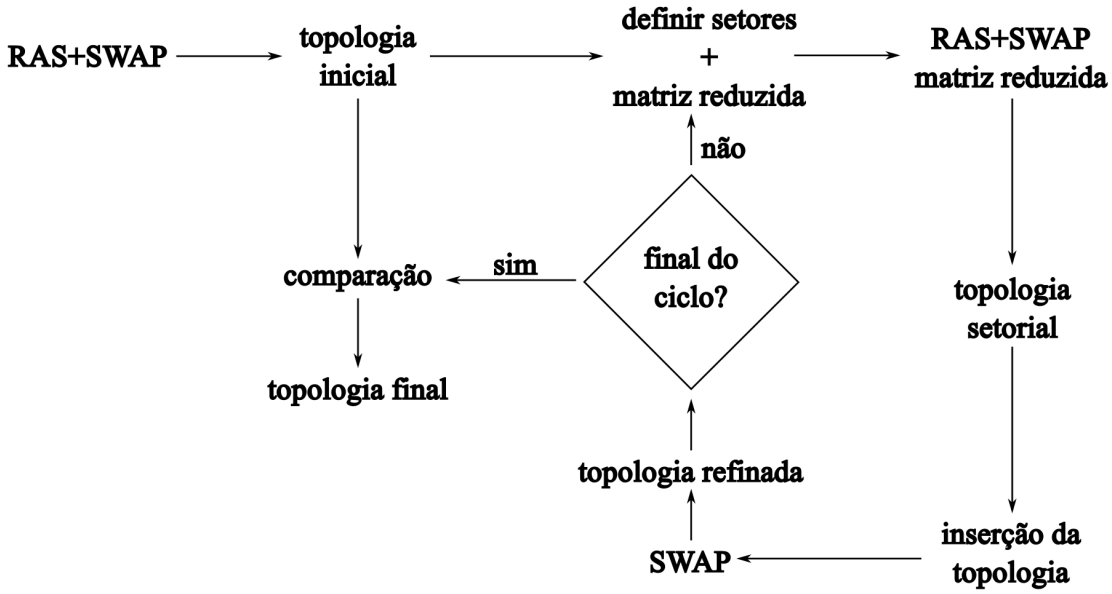

Figura 5.2: Fluxo de iterações de buscas setoriais de acordo com Goloboff [2].

5.1.3 Annealing & algoritmos genéticos:

Dentro das novas tecnologias de busca de TNT há dois outros algoritmos que merecem consideração. O primeiro deles é o *tree-drift*. Parte de métodos conhecidos como *simulated annealing*, o *tree-drift* é caracterizado por aceitar certa proporção de topologias subótimas durante o processo de rearranjo interno (*i.e., swap*) alternado com a configuração original do *swap*. Posteriormente, como em *ratched*, topologias subótimas são descartadas e uma nova iteração do ciclo se inicia. Tais ciclos, que se repetem por um número determinado de vezes, são quase tão eficientes como o *ratchet* na busca de topologias ótimas.

O segundo deles é o *tree-fusing*. O *tree-fusing* pertence a uma classe de algoritmos conhecidos como *genetic algorithms*, pois ao contrário dos algoritmos ilustrados acima – que baseiam-se em uma única topologia –, algoritmos genéticos promovem rearranjos de clados entre topologias. Implementado em TNT, esse método compara diferentes topologias e promove a troca de clados compatíveis (*i.e.*, mesma composição entre as topologias). Esse método leva a melhores resultados quando há um número grande de topologias disponíveis para a troca. Para maiores detalhes sobre o método consulte Goloboff [2] e Giribet [4].

Exercicio 5.1

No Tutorial 4 fizemos uma série de buscas no arquivo `zilla.tnt` utilizando os algoritmos tradicionalmente implementados em programas filogenéticos (*i.e.*, *random sequence addition*, RAS ou árvore de Wagner, seguidas de refinamento por algoritmos de *swap*, tais como SPR e TBR) para buscas heurísticas. Vocês devem ter observado que muito provavelmente ninguém obteve 100 topologias ao custo de 16218 passos. Neste tutorial iremos explorar a eficiência dos algoritmos descritos acima e verificar se de fato ele

possibilitam um melhor resultado com a análise da matriz em `zilla.tnt`.

i. Configuração de *default* do comando XMULT.
As buscas com novas tecnologias são implementadas no TNT pelo comando `xmult`, ou `xmu`.

a. Abra o arquivo `zilla.tnt` no TNT.

b. Execute o comando `xmu;` no TNT.

c. Qual o custo da topologia que você encontrou e quantas topologias você recuperou?

---

d. Qual foi a redução de custo desta topologia em comparação com sua melhor análise de `zilla.tnt` no Tutorial 4, Exercício 4.7?

---

e. Utilizando o comando `xmu:;` e o arquivo `xmu.txt` – que contém o *log* do comando `help xmu` –, escreva abaixo quais foram os parâmetros de execução do comando `xmu;` executado acima?

---

ii. Modificando as Configurações de *default* do comando XMULT.
Nesse exercício iremos explorar como os diferentes algoritmos implementados no comando `xmu` modificam sua performance. Você deverá realizar seis análises, uma utilizando os algoritmos tradicionais de busca em TNT e as demais implementando sequencialmente os algoritmos mais agressivos sob o comando `xmu`. Os resultados dessas análises deverão ser anotados na Tabela 5.1. O arquivo `xmu.txt` deverá ser consultado caso tenha dificuldades de visualizar todas as opções do comando `xmu` na tela do terminal onde o TNT será executado. O arquivo que contém a matriz de `zilla.tnt` foi modificado para que o TNT utilize 512 MB de RAM e possa guardar 10000 topologias. Considere verificar os parâmetros de `xmu` à cada etapa do exercício para certificar-se de que os parâmetros que deseja estão de fato implementados.

a. Faça uma busca convencional em TNT utilizando 100 réplicas para adições aleatórias (**RAS**) e mantendo 10 topologias durante cada réplica para a matriz de `zilla.tnt`. Os resultados dessa análise deverão ser inseridos na Tabela 5.1.

b. Utilizando 20 réplicas e a manutenção de 10 topologias por réplicas – que deverá ser implementado em `xmu` (*e.g.*, `xmu: rep 20 hold 10`) – execute uma análise utilizando a configuração para o comando `xmu` que implemente **apenas buscas setoriais**. Observe que por *default*, o TNT executa o comando `xmu` com os algoritmos de *sectorial searches* e *tree-fusing*. Desta forma você deverá desabilitar a execução do *tree-fusing*. O arquivo `xmu.txt` – que contém o *log* do comando `help xmu` – deverá

ser consultado caso tenha dificuldades de visualizar todas as opções do comando `xmu` na tela. Os resultados dessa análise deverão ser inseridos na Tabela 5.1.

c. O TNT utiliza concomitantemente duas estratégias de buscas setoriais implementadas nos algoritmos CSS (*constraint-based sector selections*) e RSS (*random sector selections*). Nesta execução de TNT nós queremos avaliar a performance do *ratchet*. Desta forma você deverá desabilitar as buscas setoriais, implementar 10 iterações de *ratchet*, executar a análise e registrar os resultados na Tabela 5.1.

d. Excecute `xmu` implementando 10 ciclos de *drift* e registre os resultados na Tabela 5.1. Certifique-se de que somente o *drift* está habilitado.

e. Excecute `xmu` implementando *fuse* para ao conjunto de árvores e registre os resultados na Tabela 5.1. Certifique-se de que somente o *fuse* está habilitado.

f. Finalmente, você deverá implementar todos os algoritmos que executou acima e suas respectivas configurações em uma única análise e registrar os resultados na Tabela 5.1.

Tabela 5.1: Número de sequências aleatórias (**RAS**), topologias mantidas em cada réplica, algoritmo implementado, número de topologias examinadas, tempo de execução, número de *hits* no menor custo e custo encontrado para buscas heurísticas utilizando a matriz `zilla.tnt`.

RAS	Trees/RAS	Algoritmo	# Rearrangement	Time	Best Score
100	10	TBR			
20	10	SECT			
20	10	RAT 10			
20	10	DRIFT 10			
20	10	FUSE 10			
20	10	ALL			

iii. Com base em seus resultados da Tabela 5.1 responda:

a. Como você explica que mesmo visitando um maior número de rearranjos, o algoritmo de RAS+TBR não encontrou uma ou mais topologias com custo igual ou inferior a 16218 passos?

_____

_____

_____

b. Você consideraria que um destes algoritmos é mais eficiente em relação aos outros? Justifique.

_____

_____

_____

c. Quantas topologias com 16218 passos você encontrou durante as análises acima?

d. A matriz de dados `zilla.tnt` resulta em mais de 90 MPTs (*i.e., Most Parsimonious Trees*). Modifique os parâmetros de `xmu` e a analize a matriz `zilla.tnt` de modo que você obtenha o máximo de topologias possível com o custo de 16218. Abaixo indique quantas topologias você encontrou e quais os parâmetros de análise que você utilizou.

5.2 Reconstruções e regras de colapso de ramos

Nesta seção iremos explorar as reconstruções de estados ancestrais em TNT bem como as regras disponíveis nesse programa para colapsar ramos. O exemplo utilizado aqui é o mesmo utizado por Coddington & Scharff [6] em um artigo clássico que discute o problemas com comprimentos de ramo iguais a zero – vale a pena dar uma olhada nesse artigo, especialmente se você utiliza outros programas de inferência filogenética como por exemplo o PAUP* [7].

Exercicio 5.2

Antes de entender como as regras de colapso funcionam e como o TNT reconstroi estados ancestrais nos nós de uma topologia, façamos o seguinte exercício:
A Figura 5.3 contém uma matriz de dados e 5 topologias. Sua primeira tarefa será otimizar cada caráter da matriz nas 5 topologias. Caso tenha dificuldade de fazer as otimizações, consulte este vídeo.

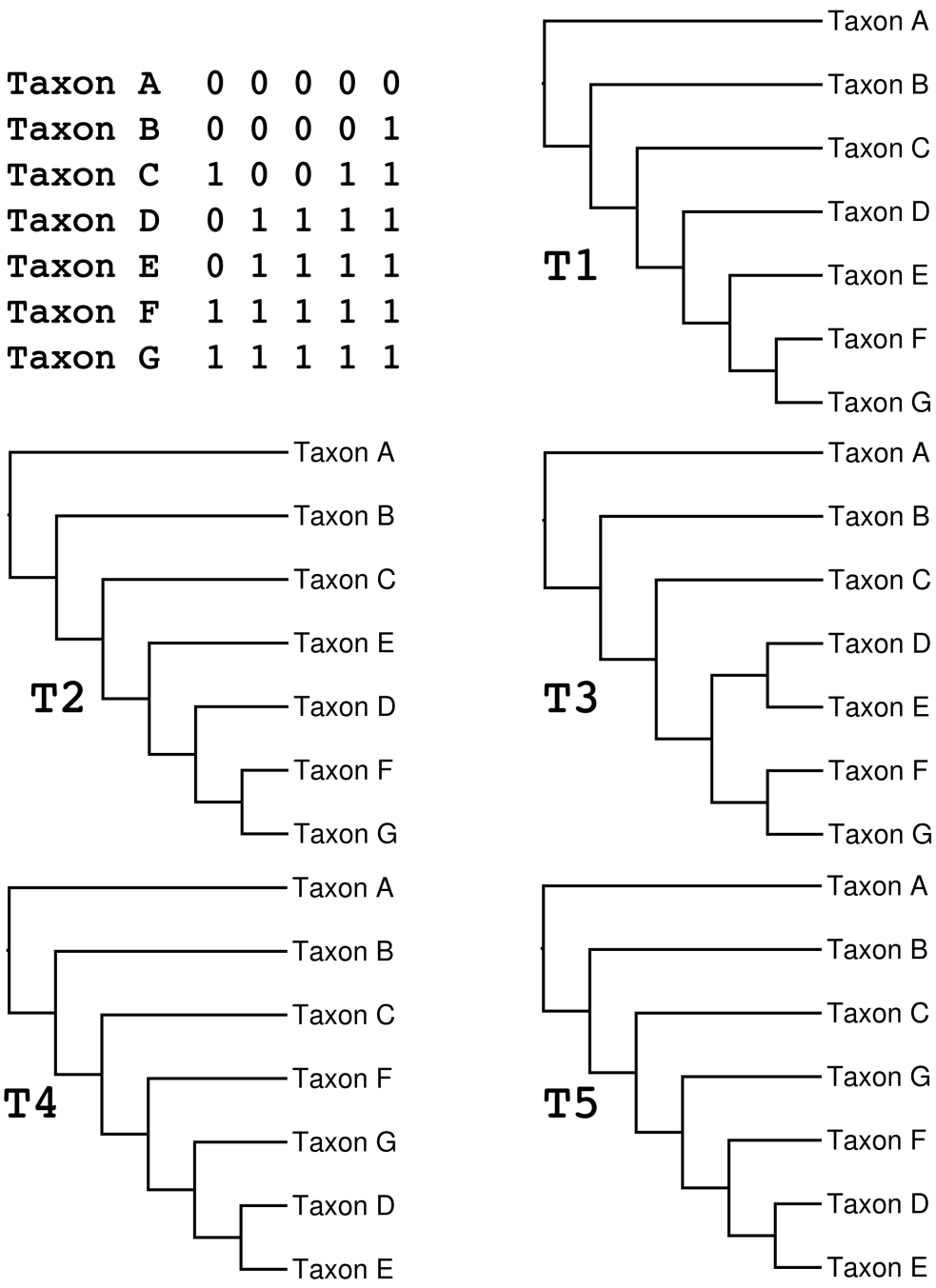

Figura 5.3: Matriz da Tabela 1 de Coddington & Scharff [6] e 5 topologias obtidas em TNT por enumeração implícita e opção de colapso de ramos `collapse 0`.

Após ter concluida a otimização destes caracteres nestas topologias você deverá verificar se você optimizou corretamente. Para isso, há dois arquivos disponíveis no diretório deste tutorial: `exercicio_5.2_matrix.tnt` e `exercicio_5.2_trees.tre`, que se referem à matriz e topologias apresentadas na Figura 5.3. O comando de TNT que permite a reconstrução de caracteres em determinada topologia é o comando "`recons`". Inicie o TNT e digite "`help recons`". Você deverá obter:

```
tnt*>help recons
RECONS
N/L; most parsimonious reconstructions for character(s) L,
```

```
tree(s) N
```

Portanto, o comando "`recons 0/1`" apresentaria a(s) recontrução(ões) do caráter 2 na topologia 1 da Figura 5.3. Se você executar "`recons`" todas as reconstruções possíveis para todos os caracteres e topologias serão apresentadas. Finalmente, se você executar "`recons /0`", todas as recontruções do caráter 1 serão apresentadas em todas as topologias.

A apresentação das reconstruções em TNT obedece o formato ilustrado na Figura 5.4. A numeração nos nós referem-se ao estado de caráter estimado para o caráter de acordo com a reconstrução. Arestas cujos vértices possuem estados distintos indicam transformação de estados de caráter. Observe que em alguns casos (veja Figura 5.4) há mais do que uma reconstrução para um determinado caráter dado uma topologia. Neste caso em particular, a primeira reconstrução (`reconstruction 0`) foi executada pelo algoritmo de DELTRAN (de *delayed transformation*) ao passo que a segunda (`reconstruction 1`) pelo algoritmo de ACCTRAN (de *accelerated transformation*) [8, 9]. Quando há mais do que uma reconstrução possível e igualmente parcimoniosas, a otimização do caráter é ambígua [8] e baseado no critério de otimização não há justificativa para selecionar uma ou outra. O TNT é o único programa que apresenta todas as reconstruções possíveis para determinado caráter dado uma topologia utilizando ACCTRAN e DELTRAN ou a combinação dos dois algoritmos. Alguns autores, por exemplo de Pinna [10], argumentam que ACCTRAN seria defensável filosoficamente, argumentos que são rejeitados por Agnarsson & Miller [11]. Para uma revisão sobre o assunto consulte Agnarsson & Miller [11].

Observe se para as otimizações que você fez à mão havia uma outra alternativa e anote nas topologuias da Figura 5.3.

```
Tree 2, char. 0, reconstruction 0
    ,--0 taxon_A
    |   ,--0 taxon_B
|--0|   |   ,--1 taxon_C
    `--0|   |       ,--1 taxon_G
        `--0|   ,--1---1 taxon_F
            `--0|   ,--0 taxon_E
                `--0---0 taxon_D

Tree 2, char. 0, reconstruction 1
    ,--0 taxon_A
    |   ,--0 taxon_B
|--0|   |   ,--1 taxon_C
    `--0|   |       ,--1 taxon_G
        `--1|   ,--1---1 taxon_F
            `--1|   ,--0 taxon_E
                `--0---0 taxon_D
```

Figura 5.4: Otimização do caráter 1 na Topologia 3 da Figura 5.3 seguindo Coddington & Scharff [6].

Exercicio 5.3

Agora que você já conhece um pouco da teoria, vamos examinar duas regras de colapso de ramos existentes em TNT. Os comandos "`collapse`" e "`help collapse`" lhe fornecem a opção de *default* de TNT para este comando e as opções disponíveis no

programa, respectivamente. Iremos explorar duas destas opções, uma vez que são as mais comumente utilizadas na literatura e respondem por possíveis diferenças entre os resultados apresentados por TNT e PAUP*.

i. Utilizando a configuração original de TNT, faça uma busca por enumeração implícita, imprima as tolopogias no terminal e, com referência àqueles ilustradas na Figura 5.3 responda:

a. Quais tolopogias foram recuperadas? Considere que as topologias apresentadas na Figura 5.3 são binárias ao passo que as apresentadas após a análise tiveram seus ramos colapsados, verifique quais topologias binárias são compatíveis com as da Figura 5.3.

b. Baseada nas reconstruções dos caracteres, você poderia explicar qual regra esta sendo adotada para colapsar os ramos?

ii. Modifique a regra de colapso adotando aquela que é *default* em PAUP*, execute novamente a análise e responda:

a. Quais tolopogias foram recuperadas com essa nova regra?

b. Baseada nas reconstruções dos caracteres, você poderia explicar qual regra esta sendo adotada para colapsar os ramos?

c. Qual regra você adotaria em seu estudo? Justifique.

5.3 Diagnoses

Neste seção iremos explorar o comando "`apo`" do TNT e sua relação com o comando "`recons`" discutido no tópico anterior.

O comando `apo` possui as seguintes opções:

```
tnt*>help apo;
APO
N     plot synapomorphies for tree(s) N
[N    plot synapomorphies common to tree(s) N
```

```
-     list instead of plotting on tree
[-N/L list synapomorphies common to tree(s) N, node(s) L
```

Considere a matrix no arquivo `vertebrados.tnt`. A análise cladística por enumeração implícita gera três topologias (`Tree 0 - Tree 2` da Figura 5.5).

```
Tree 0:
   ,-- peixe
   |  ,-- piramboia
|--|  |     ,-- salamandra
   `--|  ,----- sapo
      |  |  ,-- mamifero
      `--|  |-- tartaruga
         `--|     ,-- cobra
            |  ,----- lagarto
            `--|  ,-- aves
               `----- crocodilo
```

```
Tree 1:
   ,-- peixe
   |  ,-- piramboia
|--|  |     ,-- salamandra
   `--|  ,----- sapo
      |  |  ,-- mamifero
      `--|  |-- tartaruga
         |  |  ,-- cobra
         `--+----- lagarto
            |  ,-- aves
            `----- crocodilo
```

```
Tree 2:
   ,-- peixe
   |  ,-- piramboia
|--|  |     ,-- salamandra
   `--|  ,----- sapo
      |  |  ,-- tartaruga
      `--|  |  ,-- mamifero
         `--|  |  ,-- cobra
            `--+----- lagarto
               |  ,-- aves
               `----- crocodilo
```

```
Tree 0 [naked -]:
      ,--0 peixe
      |    ,--3 piramboia
|--10|    |          ,--4 salamandra
      `--13|     ,--11---1 sapo
           |     |     ,--8 mamifero
           `--12|     |--2 tartaruga
                `--14|          ,--9 cobra
                     |    ,--17---6 lagarto
                     `--16|    ,--7 aves
                          `--15---5 crocodilo
```

Figura 5.5: Topologias (`Tree 0 - Tree 2`) resultantes da enumeração implícita de `vertebrados.tnt`. A topologia "`Tree 0 [naked -]`" foi impressa com a opção do comando "`naked`" de TNT que apresenta a numeração dos nós na Topologia `0`.

Há duas formas de visualizar as sinapomorfias não ambíguas para essas tolologias. A primeira delas é impressa na topologia pelo comando, por exemplo, `apo 0` – que apresenta as apomorfias inequívocas para a topologia `Tree 0` (veja Figura 5.6). Outra forma de apresentar as sinapomorfias é listando-as com o comando, por exemplo, "`apo- 0`" – que lista as apomorfias inequívocas para a topologia `Tree 0` (veja Figura 5.6). Observe que neste último caso, é conveniente referenciar os nós da topologia utilizando o comando "`naked-`" antes de executar "`tplot`" (consulte o *help* para esses dois comandos noa TNT).

```
xread
13 10
peixe       000000000?0010
sapo        0110110000011
tartaruga   1120110000010
piramboia   000011100?0010
salamandra 0110110000011
crocodilo   1120112001100
lagarto     1120112110010
aves        1231112001100
mamifero    1[12]41111000010
cobra       1?20112110010
;
```

```
Synapomorphies for tree 0
   ,-- peixe
   |  ,-- piramboia
|--|  |        ,-- salamandra
   `--|   ,--12--- sapo
      |   |   ,--2,3,6 mamifero
      `--1|   |-- tartaruga
          `--0|         ,-- cobra
              |   ,--7,8--- lagarto
              `--6|         ,--1,2,3 aves
                  `--9,10,11--- crocodilo
```

```
Tree 0 :
    peixe :
      No autapomorphies
    sapo :
      No autapomorphies
    tartaruga :
      No autapomorphies
    piramboia :
      No autapomorphies
    salamandra :
      No autapomorphies
    crocodilo :
      No autapomorphies
    lagarto :
      No autapomorphies
    aves :
      Char. 1: 1 --> 2
      Char. 2: 2 --> 3
      Char. 3: 0 --> 1
    mamifero :
      Char. 2: 2 --> 4
      Char. 3: 0 --> 1
      Char. 6: 0 --> 1
    cobra :
      No autapomorphies
   Node 11 :
      Char. 12: 0 --> 1
   Node 12 :
      Char. 1: 0 --> 1
   Node 13 :
      No synapomorphies
   Node 14 :
      Char. 0: 0 --> 1
   Node 15 :
      Char. 9: 0 --> 1
      Char. 10: 0 --> 1
      Char. 11: 1 --> 0
   Node 16 :
      Char. 6: 0 --> 2
   Node 17 :
      Char. 7: 0 --> 1
      Char. 8: 0 --> 1
```

Figura 5.6: Matriz de dados `vertebrados.tnt` e apomorfias para `Tree 0` plotadas na topologia e listadas.

Exercicio 5.4

Os exercícios abaixo são referêntes à matriz de dados em `vertebrados.tnt`.

i. Na figura abaixo, indique as apomorfias comuns para as topologias `Tree 0 -2`.

```
   ,-- peixe
   |  ,-- piramboia
|--|  |     ,-- salamandra
   `--|  ,----- sapo
      |  |  ,-- mamifero
      `--|  |-- tartaruga
         |  |  ,-- cobra
         `--+----- lagarto
            |  ,-- aves
            `----- crocodilo
```

ii. Qual é a polarização dos caracteres que sustentam a hipótese de que Aves e Crocodilos são grupos irmãos nas topologias `Tree 0 -2`?

iii. Quais são as possiveis reconstruções do caráter 6 nas tolopogias `Tree 0 -2`? (inicio da contagem no zero)

_____

_____

_____

iv. Por que não há sinamoporfia(s) listadas para o nó 13 das tolopogias `Tree 0 -2` e mesmo assim ele é resolvido nestas hipóteses?

_____

_____

_____

5.4 Referências

1. Goloboff, P.; Farris, J. S. & Nixon, K. 2008. TNT a free program for phylogenetic analysis. *Cladistics* **24**: 1–14.
2. Goloboff, P. 1999. Analyzing large data sets in reasonable times: solutions for composite optima. *Cladistics* **15**: 415–428.
3. Nixon, K. C. 1999. The parsimony ratchet, a new method for rapid parsimony analysis. *Cladistics* **15**: 407–414.
4. Giribet, G. 2007. Efficient tree searches with Available Algorithms. *Evolutionary Bioinformatics* **3**: 341–356.
5. Wheeler, W. C. 2012. Systematics: a course of lectures. Malaysia: Wiley-Backwell, 2012. 426.
6. Coddington, J & Scharff, N. 1994. Problems with zero-length branches. *Cladistics* **10**: 415–423.
7. Swofford, D. 2003–2016. PAUP*. Phylogenetic Analysis Using Parsimony (*and Other Methods, Version 4.0a131). Sunderland, Massachusetts: Sinauer Associates, 2003–2016.
8. Farris, S. 1970. Methods for computing Wagner trees. *Systematic Zoology* **19**: 83–92.
9. Swofford, D. L. & Maddison, W. P. 1987. Reconstructing ancestral character states under Wagner parsimony. *Mathematical Bioscience* **87**: 199–229.
10. De Pinna, M. G. G. 1991. Concepts and tests of homology in the cladistic paradigm. *Cladistics* **7**: 367–394.
11. Agnarsson, I. & Miller, J. A. 2008. Is ACCTRAN better than DELTRAN? *Cladistics* **24**: 1–7.