Tutorial 10

Homologia Dinâmica: Buscas em POY, sensibilidade, comprimentos de ramos e alinhamentos implícitos

BIZ0433 - INFERÊNCIA FILOGENÉTICA: FILOSOFIA, MÉTODO E APLICAÇÕES.

Conteúdo

Objetivo

O objetivo deste tutorial é apresentar novos conceitos associados a análise de sequências nucleotídicas, principalmente utilizando otimização direta de dados moleculares. O tutorial está centrado no conceito de análise de sensibilidade para o qual será apresentado algumas ponderações epistemológicas – estas devem ser suplementadas pela literatura citada no tutorial –, bem como a implementação destas análises em POY. Antes disso, você irá conhecer como POY implementa novas tecnologias de busca em análises com tempo determinado, uma ferramenta muito útil deste programa. O tutorial também explora o uso de FigTree e Inkscape na preparação de figuras que contém diagramas de sensibilidade. Por fim, o tutorial mostra como o POY apresenta topologias com seus respectivos comprimentos de ramo e o alinhamento implícito relacionado com análises de homologia dinâmica. 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/
```

10.1 Novas tecnologias de busca em POY

No tutorial anterior utilizamos os algoritmos mais simples de busca existentes em POY (*i.e.*, RAS+SWAP). No entanto, o POY também permite que a maioria dos algoritmos implementados em TNT [1], considerados novas tecnologias de busca [2, 3], sejam implementados em buscas de homologia dinâmica. No entanto, devido às propriedades dos algoritmos de POY, algumas implementações requerem uma série de transformações dos dados que podem ser complexas para o usuário com pouca experiência. Considere por exemplo a implementação de *ratchet* [3]. Esse algoritmo atribui pesagem diferencial à uma proporção de seus caracteres, executa busca e refinamento na matriz, retorna ao esquema de pesagem inicial e avalia o custo da(s) topologia(s) em comparação àquela(s) inicialmente retida(s) na memória do programa. Em homologia dinâmica, é necessário transformar os caracteres dinâmicos em caracteres estáticos temporariamente para a implementação de *ratchet*. Isso é necessário porque a noção de caráter em otimização direta varia a cada ciclo de otimização durante as buscas. Portanto, a implementação em POY de algoritmos de dependem de homologia estática, como é o caso de *ratchet*, requer uma série de instruções no *script* que, como verão, são de certa forma desnecessárias na maioria das análises que você irá fazer.

Para facilitar o uso de POY, o desenvolvedores do programa criaram o comando "`search`" que implementa todas essas novas tecnologias de busca sem que o usuário tenha que se preocupar com as linhas de comando de cada uma das estratégias de refinamento. Por *default*, o comando "`search`" inclui a construção inicial de uma árvore de Wagner, em seguida, implementa *branch swapping* via TBR (veja Tutorial 4, Seção 4.7), *ratchet* [3] e *tree fusing* [2], sequencialmente. Esse ciclo de algoritmos é considerado pelo POY como repetições independentes, cujo número de iterações dependerá da complexidade de seus dados e dos argumentos implementados no comando "`search`".

Os argumentos do comando "`search`" permitem uma série de controles (veja item 3.3.22, página 119, do manual do programa). Por *default* o comando faz a busca por uma hora e usa 2 Gb de memória RAM em seu computador. Esse comando seria expresso literalmente da seguinte forma: `search(max_time:0:1:0,min_time:0:1:0,memory:gb:2)`[1]. Nesta linha de comando, o argumento "`max_time:0:1:0`" define o máximo de tempo estipulado para análise em dias:horas:minutos, o argumento "`min_time:0:1:0`" define o mínimo de tempo estipulado para análise da mesma forma, e "`memory:gb:2`" controla o máximo de memória alocada durante a execução. Este último restringe o número de topologias retidas na memória durante as buscas. O número de repetições independentes, neste caso, dependerá apenas da complexidade de seus dados, ou seja, o tempo necessário para completar uma iteração. Há uma série de outros argumentos para este comando e o leitor deve consultar o manual do programa caso esteja interessado em implementar alguns deles em suas análises. No exemplo abaixo iremos explorar apenas um deles (*i.e.*, `max_time:d:m:s`), pois é o mais utilizado.

Considere o *script* abaixo:

[1] No entanto, como estes são os argumentos originais do comando, bastaria escrever "`search()`".

Listing 10.1: conteúdo do arquivo script1.poy (Tutorial 10)

```
read("partition2.fas")
set(root:"Taxon1")
search(max_time:0:0:0.1)
select()
report(trees)
report(searchstats)
exit()
```

Neste *script* o arquivo `partition2.fas` é lido pelo POY [linha 1], o táxon `Taxon1` é utilizado como raiz [2] e a busca será feita por 1/10 minutos [3] após a qual as topologias únicas e mais curtas serão retidas [4] e impressas [5] juntamente com a estatística da busca [6].

Execute este *script* da seguinte maneira:

```
$ poy5.1.2a script1.poy >std.out 2>std.err &
```

Nesta linha de comando, o *prompt* do terminal será liberado logo após você executá-lo, pois o caráter "`&`" no final da linha faz com que o programa seja executado no *background*. Para saber se o programa terminou a execução pressione a tecla `ENTER` após alguns segundos. Em algum momento você deverá obter:

```
alan@turing: /Desktop/tutorials/tutorial_10$ poy5.1.2a script1.poy >std.out 2>std.err &
[1] 3992[2] alan@turing: /Desktop/tutorials/tutorial_10$

[1]+ Concluído poy5.1.2a script1.poy >std.out 2> std.err
```

Observe os redirecionamentos implementados na linha de comando (veja Tutorial 1)[3]. O primeiro deles (*i.e.*, "`>std.out`"), os registros de saída convencionais de POY – geralmente aqueles que direcionamos para arquivos, como por exemplo `report("trees.tre",trees)` –, ou os chamados *stdout*, são direcionados para o arquivo `std.out`. O segundo deles (*i.e.*, "`>std.err`"), os registros de execução ou erro convencionais de POY – geralmente aqueles impressos no terminal durante a execução –, ou os chamados *stderr*, são direcionados para o arquivo `std.err`. Vamos examinar o conteúdos destes dois arquivos.

Se você examinar o conteúdo do arquivo `std.out`, você deverá encontrar algo semelhante ao que está impresso abaixo:

```
(Taxon1,((Taxon5,(Taxon2,(Taxon4,Taxon9))),((Taxon3,Taxon8),(Taxon6,(Taxon10,Taxon7)))));
(Taxon1,((Taxon5,(Taxon2,(Taxon4,Taxon9))),(Taxon6,(Taxon8,(Taxon3,(Taxon10,Taxon7))))));
Search Stats:
# of Builds + TBR  16
# of Fuse         93
```

[2] número do processo, ou seja, o número que a máquina atribuiu para a linha de comando que você acaba de executar.

[3] mais informações sobre *standard I/O streams* em linux veja http://linux.about.com/library/cmd/blcmdl3_stderr.htm

# of Ratchets 11

Tree length Number of hits

191. 195

192. 7

193. 2

194. 5

195. 3

196. 2

197. 1

198. 1

199. 1

Neste arquivo de saída, as duas primeiras linhas contém as duas topologias encontradas na análise, impressas nesse arquivo pelo comando "`report(trees)`" e as demais linhas registram as estatísticas de busca, impressas nesse arquivo pelo comando "`report(searchstats)`" do *script* 10.1. Os registros de busca indicam que durante o tempo estipulado (1/10 minutos) POY realizou 16 réplicas independentes, 11 iterações de *ratchet* e 93 iterações de *tree fusing* sendo que em 195 ocasiões obteve uma topologia com 191 transformações. Lembre-se que eu poderia direcionar esses arquivos de saída para arquivos individuais implementando a seguinte linha de comando: `report("treefile.tre", trees,"searchstatistics.txt", searchstats).`

Examine o arquivo `>std.err`. Este registra toda a execução de POY que você acabou de implementar. Ele deverá conter registros semelhantes aos impressos abaixo, onde cada etapa da busca é marcada por "`<- ...`":

...

Information : The file partition2.fas contains sequences of 10 taxa, each
sequence holding 1 fragment. <– Leitura do arquivo de entrada

Status : RAS + TBR : 0

Status : RAS + TBR : 0 Searching on tree number 1

Status : Wagner : 2 of 10 – Wagner tree with cost 36. <– Construção da árvore de Wagner

...

Status : Wagner : 9 of 10 – Wagner tree with cost 172.

Status : Wagner Finished

Status : Building Wagner Tree Finished

Status : TBR : 191 191. <– Inicio do refinamento por TBR

Status : TBR Finished

Status : Automated Search : 0 Best tree: 191.; Time left: 6 s; Hits: 1 <– Melhor topologia depois de TBR

Status : Transforming : 1 of 1 – transformations applied <– Tranformação dos caracteres em homologia estática

Status : Transforming : 0 of 1 – characters transformed

Status : Transforming Finished

Status : Diagnosis : 1 of 1 – Recalculating trees

Status : Diagnosis Finished

Status : Implied Alignments : 1 of 19 – vertices calculated

Status : Implied Alignments : 2 of 19 – vertices calculated

...

Status : Implied Alignments : 16 of 19 – vertices calculated

Status : Implied Alignments : 17 of 19 – vertices calculated

Status : Implied Alignments Finished

Status : Static Approximation Finished <– Tranformação em homologia estática termina após ler alinhamento implícito

Status : Loading Characters : 0 of 10 – Storing the character specifications

Status : Loading Characters : 1 of 10 – Storing the character specifications

...

Status : Loading Characters : 9 of 10 – Storing the character specifications

Status : Loading Characters :

10 of 10 – Storing the character specifications

Status : Loading Characters Finished

Status : Transforming : 1 of 1 – transformations applied

Status : Transforming : 138 of 1 – characters transformed

Status : Transforming Finished

Status : Diagnosis : 0 Recalculating original tree

Status : Diagnosis Finished

Status : Diagnosis : 1 of 1 – Recalculating trees

Status : Diagnosis Finished

Status : Perturb Iteration : 1 of 4 <– Inicio do algoritimo de *ratchet*

Status : Ratcheting : 0 of 1 – trees in the current optimum

Status : Diagnosis : 0 Recalculating original tree

Status : Diagnosis Finished

Status : Ratcheting : 1 of 1 – trees in the current optimum

Status : TBR : 234 234.

Status : TBR Finished

Status : Diagnosis : 0 Recalculating original tree

Status : Diagnosis Finished

Status : Ratcheting Finished

Status : Perturb Iteration : 2 of 4 –

...

Status : Perturb Iteration : 4 of 4 –

Status : Ratcheting : 0 of 1 – trees in the current optimum

Status : Diagnosis : 0 Recalculating original tree

Status : Diagnosis Finished

Status : Ratcheting : 1 of 1 – trees in the current optimum

Status : TBR : 225 225.

Status : TBR Finished

Status : Diagnosis : 0 Recalculating original tree

Status : Diagnosis Finished

Status : Ratcheting Finished

Status : Perturb Iteration Finished

Status : Diagnosis : 0 Recalculating original tree

Status : Diagnosis Finished

Status : Ratcheting : 1 of 1 – trees in the current optimum

Status : TBR : 225 225.

Status : TBR Finished

Status : Diagnosis : 0 Recalculating original tree

Status : Diagnosis Finished

Status : Ratcheting Finished

Status : Perturb Iteration Finished

Status : Diagnosis : 0 Recalculating original tree

Status : Diagnosis Finished <– Termina a iteração após TBR final

Status : Automated Search : 0 Best tree: 191.; Time left: 6 s; Hits: 2

...

Status : Automated Search : 0 Best tree: 191.; Time left: 6 s; Hits: 1 <– Subsequentes iterações

...

Status : Automated Search : 0 Best tree: 191.; Time left: 6 s; Hits: 2

...

Status : Automated Search : 0 Best tree: 191.; Time left: 3 s; Hits: 5

...

Status : Automated Search : 0 Best tree: 191.; Time left: -0 s; Hits: 6

Status : Automated Search Finished

Status : RAS + TBR Finished

Status : Fusing Trees : 0 <– *tree fusing no final*

Status : Fusing Trees Finished

Status : Automated Search Finished

Information : The search evaluated 16 independent repetitions with ratchet and fusing for 93 generations. The shortest tree was found 6 times.

Estes arquivos de saída de POY são úteis para documentar a análise e permitir avaliar o comportamento de determinadas estratégias de busca (o que transcende os objetivos desse tutorial) e identificar problemas de execução. Desta forma, é interessante sempre redirecionar estas informações de saída em suas análises.

10.2 Análise de sensibilidade

10.2.1 Contextualização

Em termos gerais, análise de sensibilidade objetiva acessar a relação existente entre parâmetros analíticos e resultado [4]. Dentro do contexto de análises filogenéticas de dados moleculares, Wheeler [5] foi o primeiro a avaliar de forma sistemática a relação entre de parâmetros de alinhamento e resultados de inferência filogenética. Em sua concepção inicial, a análise sensibilidade foi considerada uma etapa necessária que precede a análise de congruência. Esta última tem como objetivo identificar o conjunto de parâmetros analíticos que maximiza congruência de caracteres entre partições de dados [veja 5–7].

O uso de análise de sensibilidade em inferência filogenética é controversa. Há duas justificativas predominantes para seu uso em inferência filogenética. A primeira delas é que ela permite acessar níveis de suporte de determinados clados [*e.g.*, 6, 8]. A outra é que estas análises são úteis e necessárias para entender a relação existente entre premissas analíticas e resultados – principalmente diante da observação de que parâmetros de alinhamento são componentes arbitrários e inevitáveis em análises filogenéticas de dados moleculares [veja 9].

Grant & Kluge [10], ao contrário, argumentam que o uso de análise de sensibilidade em inferência filogenética não possui base epistemológica e portanto não pode ser justificado [veja resposta em 9]. Segundo esses autores, a análise de sensibilidade não deve ser considerada um método científico, pois não permite teste de hipóteses, nem um método heurístico, pois não permite identificar hipóteses ambiguamente ou pouco corroboradas pelos dados disponíveis. Independentemente destas disputas filosóficas, vamos aprender como essas análises são feitas e a interpretação dos resultados requer reflexão sobre os pontos levantados por esses autores [9, 10]. Minha recomendação é que consultem a literatura que discute o tema caso venham a implementar análises de sensibilidade em seus estudos.

10.2.2 IMPLEMENTAÇÃO

Há várias formas de implementar análises de sensibilidade em POY[4] e sua implementação depende, em última instância, de quais parâmetros analíticos você quer submeter à análise de sensibilidade. Considere por exemplo a base de dados utilizada no Tutorial 9, Seção 9.2.3. As partições apresentadas na ocasião possuem dados que não estavam sujeitos a alinhamento (*i.e.*, `partition1.fas`), dados sujeitos a alinhamento (*i.e.*, `partition2.fas`) e uma matriz de dados fenotípicos (*i.e.*, `partition3.tnt`). Nossa primeira análise destas partições considerou custos idênticos para todas as transformações. Desta forma, a primeira possibilidade seria avaliar como os resultados dependem desta premissa inicial simplesmente atribuindo custos diferenciais para INDELs e substituições (veja Seção 9.2.4 daquele tutorial). No entanto, há varias outras possibilidades. Na realidade, infinitas! Você poderia considerar diferentes pesos para cada uma das partições, já que elas diferem de tamanho. Você poderia excluir a terceira posição da partição que inclui regiões codificantes; e por que não as primeiras e segundas também. Você poderia dar pesos diferenciais para dados genotípicos *vs.* fenotípicos. Enfim, o número de parâmetros passíveis de exame são inúmeros, mesmo para essas 3 partições, e não há forma objetiva de identificar quais parâmetros serão avaliados. Esse é parte dos argumentos de Grant & Kluge [10] contra análise de sensibilidade.

Tradicionalmente, no entanto, as análises de sensibilidade centram em avaliar os custos de INDELs (algumas vezes diferenciando entre *gaps* de abertura e extensão), transversões e transições [5, 6, 8, 11–13]. Em POY isso é feito implementando matrizes de custo (*i.e.*, matrizes de Sankoff) em diferentes buscas como foi feito na seção 9.2.4 do Tutorial 9.

O conjunto de funções de custo avaliado durante a análise de sensibilidade define seu espaço de parâmetros (*i.e., parameter space*). Considere por exemplo os arquivos contendo matrizes de Sankoff disponíveis no diretório `tutorial_10`. São 9 arquivos nos quais as razões de custo entre transversões e transições podem ser de 1:1, 2:1 e 1:2; ao passo em que as razões de custo entre INDELs e substituições podem ser 1:1, 2:1 e 4:1[5], respectivamente. Os nomes desses arquivos referem-se às razões de custo expressas em seu conteúdo; por exemplo, o arquivo `m421` expressa a razão de custo 4:2:1 para INDELs:transversões:transições, respectivamente. Neste caso, transições recebem custo igual a 1, transversões igual a 2 e *gaps* igual a 8.

Suponha que você queira avaliar a sensibilidade dos seus resultados em relação às razões de custo entre INDELs e substituições e considere que as matrizes de custo nos arquivos `m111`, `m211` e `m411` definem meu espaço de parâmetros. O primeiro passo seria fazer três análises filogenéticas considerando uma função de custo distinta para cada uma delas.

Considere o *script* abaixo:

[4] essas análises podem ser executadas dentro do contexto de homologia estática. Por exemplo, nada impede que os protocolos que exploramos aqui sejam implementados em TNT.

[5] observe o conteúdo destes arquivos para ver como os custos absolutos das transformações expressam as razões de custo que dão nome aos arquivos.

Listing 10.2: Exemplo de *script* para implementar análises de sensibilidade (veja `tutorial_10/script2.poy`).

```
read(prealigned:("partition1.fas", tcm:("m111")))
read("partition2.fas","partition3.tnt")
transform(names:("partition2.fas"),(tcm:("m111")))
set(root:"Taxon1")
search(max_time:0:00:1)
select()
report("trees_m111.tre",trees:(total),
"scores_m111.sts",treestats,searchstats)
exit()
```

Neste *script*, a matriz de custo `m111` é implementada nos dados moleculares – embora `partition1.fas` não esteja sujeito à alinhamento – e os resultados são direcionados para arquivos correspondentes às matrizes de custo[6]. Seguindo a lógica deste *script*, a análise de sensibilidade gerará três arquivos de árvores (*i.e.*, `trees_m111.tre`, `trees_m211.tre` e `trees_m411.tre` – veja diretório `tutorial_10`). Subsequentemente, você poderia inspecionar visualmente as topologias resultantes para identificar os clados que são dependentes dos parâmetros de alinhamento. No entanto, essa tarefa pode consumir um bom tempo e há ferramentas disponíveis que podem ser úteis nessa tarefa. É o que veremos a seguir.

10.2.3 Avaliação

Em dados reais, raramente você terá tempo e paciência para inspecionar visualmente os resultados de uma análise de sensibilidade. Há duas ferramentas disponíveis para este propósito. A primeira delas é o aplicativo Cladescan [14; disponível para baixar aqui]. Alternativamente, Machado [15] desenvolveu o YBYRÁ, um aplicativo mais versátil e rápido quando comparado ao Cladescan. Finalmente, algumas ferramentas de visualização de árvores [*e.g.*, FigTree; 16] são muito úteis para esse propósito (veja Tutorial 2). O resultado de uma análise de sensibilidade é ilustrado na Figura 10.1. Essa figura foi gerada a partir da análise simultânea e individuais dos dados nos arquivos `partition1.fas`, `partition2.fas` e `partition3.tnt`. Ela mostra a frequência de cada um dos clados da análise simultânea na topologias recuperadas pelas análises individuais.

[6] caso tenha dificuldade de compreender os demais comandos deste *script* revise o Tutorial 9 e/ou consulte a documentação do programa.

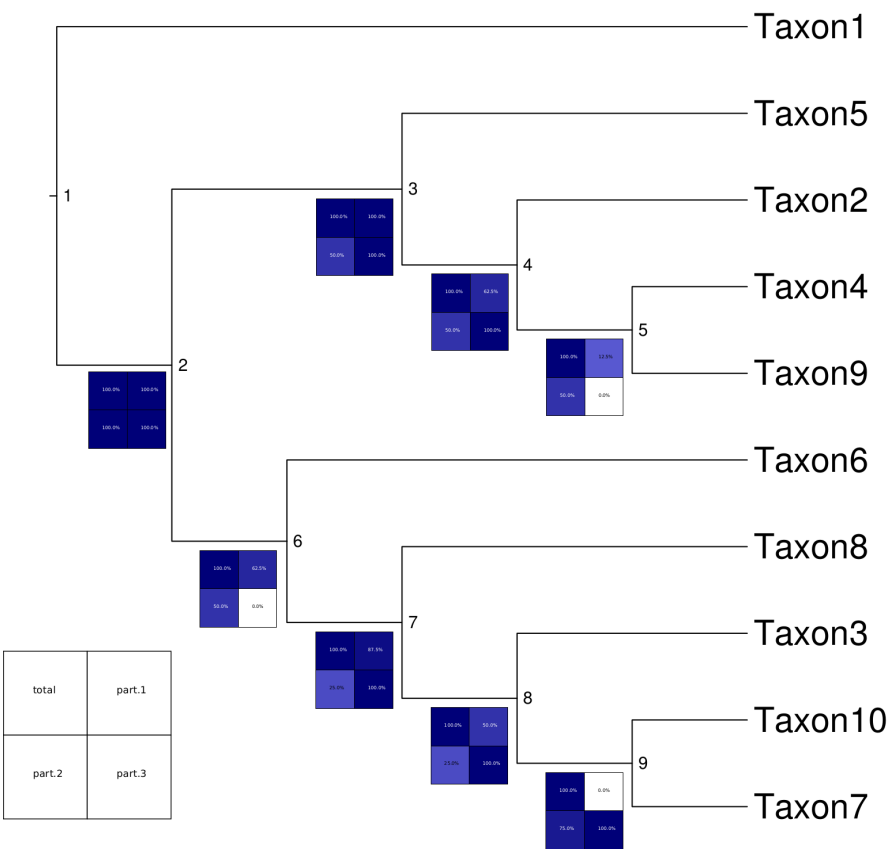

Figura 10.1: Diagrama de sensibilidade (*Navajo's rug*) indicando como cada clado da análise simultânea se comporta nas análises individuais de cada partição. Quadrados em azul indicam a presença do clado de acordo com o mapa do *plot* de sensibilidade ao lado do cladograma.

No exemplo abaixo iremos explorar o uso de YBYRÁ e FigTree para avaliar o resultado da análise de sensibilidade que foi executada na Seção 10.2.2.

10.2.3.1 YBYRÁ

Toda documentação do YBYRÁ está diposponível do GitLab do desenvolvedor. Para o propósito deste tutorial iremos apenas explorar como o YBYRÁ [7] pode ser utilizado para executar uma análise de sensibilidade.

O programa recebe instruções de um arquivo de configuração. No caso do exemplo que iremos apresentar aqui, essas instruções estão no arquivo `sensibilidade.conf`. Abra esse arquivo no `gedit`[8] para que você siga as linhas que foram modificadas para que o YBYRÁ execute a análise de sensibilidade. As modificações feitas neste arquivo foram as seguintes:

i. Na linha 75 foi definida a `id` da análise (*i.e.*, `sa_aula`). A definição deste parâmetro no arquivo de configuração regula os nomes dos arquivos de *output* (veja abaixo).

ii. Nas linhas 94–98 foram definidas as topologias que serão examinadas pelo programa em seus respectivos arquivos (*i.e.*, `trees_m111.tre`, `trees_m211.tre` e

[7] Antes de iniciar este componente do tutorial execute o *script* install_dependencies.sh. Isso garantirá que você possui os programas necessários para executar os exercícios

[8] Verifique se a numeração de linhas está habilidada no `gedit`. Se não estiver, vá em **Preferences** e clique em **Display line numbers**.

`trees_m411.tre`). Em frente de cada arquivo, entre colchetes estão os *labels* que serão utilizados nos diagramas de sensibilidade.

iii. Na linha 123 foi definida a topologia de referência, no caso aquela contida no primeiro arquivo da lista apresentada nas linhas 94–98. Entre colchetes está o nome do arquivo – que é opicional. Por *defaul*, YBYRÁ irá sempre utilizar a primeira topologia do primeiro arquivo da lista.

iv. Na linha 174 está definida o tipo de análise que você quer fazer – nesse caso iremos comparar uma única topologia com as demais (1). Verifique as opções disponíveis no programa.

v. Na linha 188 está definido que o YBYRÁ irá comparar clados – opção 1 –, e não vértices de um diagrama não enraizados (*i.e., splits*).

vi. Na linha 211 foi definido que você quer que o YBYRÁ imprima os diagramas de sensibilidade no formato SVG (*Scalable Vector Graphics*) o que pode ser útil para gerar figuras como a apresentada acima (Figura 10.1). A cor desses diagramas é especificada na linha 237.

vii. Uma vez configurado, basta executar o YBYRÁ com a seguinte linha de comando:
`$ ybyra_sa.py -f sensibilidade.conf`[9]

A execução do YBYRÁ gera uma série de arquivos de saída e o você deve consular o tutorial apresentado pelo desenvolvedor do programa, bem como a documentação do YBYRÁ, caso tenha interesse de saber o conteúdo específico de cada um deles. Para o propósito deste tutorial iremos descrever apenas alguns.

Após a execução diretório `tutorial_10/` deverá conter os resultados da execução do YBYRÁ usando o arquivo de configuração `sensibilidade.conf`. Dentre estes arquivos, `LABELED_sa_aula.tre` contém a topologia de referência (*i.e.*, `trees_m111.tre`) com os nós anotados em referência aos arquivos `RUG_000*_sa_aula.svg`. Estes últimos são arquivos vetoriais que mapeiam a frequência de cada clado presente na topologia de referência que foi encontrado nas demais topologias de acordo com o mapeamento apresentado no arquivo `RUG_map_sa_aula.svg`.

10.2.3.2 FigTree & Inkscape

O resultado da análise de sensibilidade executada acima está ilustrado na Figura 10.2. Esta figura foi gerada utilizando dois aplicativos disponíveis na imagem do curso FigTree e Inkscape. Ambos programas podem ser executados em todas as plataformas (*i.e.*, OS X, Linux e Windows) e são *freeware*. Há dois vídeos em `tutorial_10/videos/` que dão uma expliação geral de como usar o FigTree e o Inkscape nesse tutorial. O vídeo `figtree_1.mp4` explica como o FigTree foi usado para gerar uma figura no formato SVG que representa a topologia que YBYRÁ utilizou para referenciar os diagramas de sensibilidade. O vídeo `inkscape_1.mp4` explica como o Inkscape

[9] esta linha de comando assume que você possui o programa instalado no seu sistema. A execução local deste programa pode ser feita da seguinte forma: "`$ python ybyra_sa.py -f arquivo.conf`". No diretório deste tutorial há uma versão simplificada deste arquivo de configuração chamado `sensibilidade_simple.conf`.

foi utilizado para compor a Figura 10.2. Se você já está familiarizado com esses aplicativos siga o tutorial a diante. No entanto, se você não os conhece é necessário que você assista esses vídeos antes de executar o exercício a seguir.

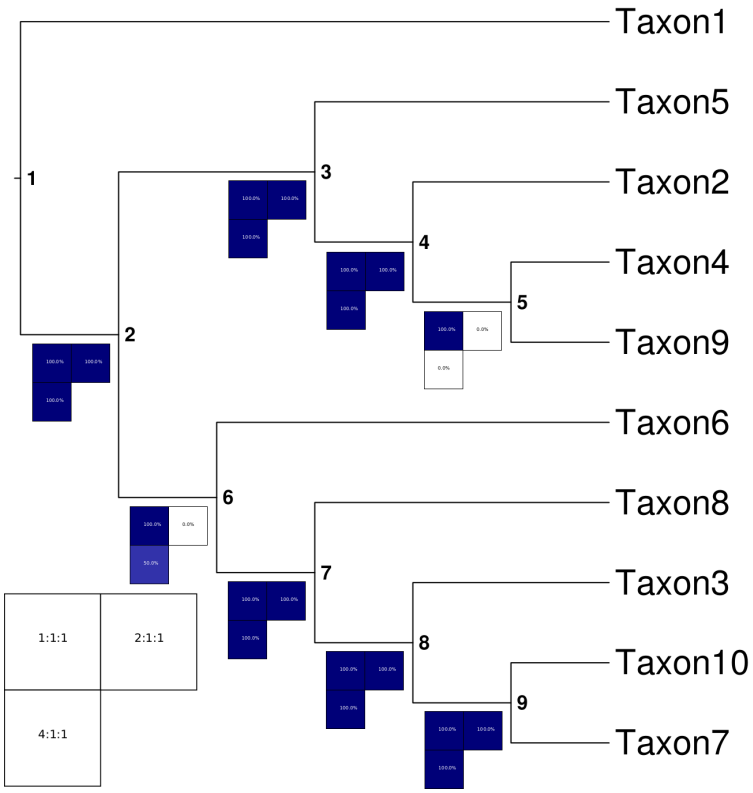

Figura 10.2: Diagrama de sensibilidade (*Navajo's rug*) para parâmetros de alinhamento indicando como cada clado da análise se comporta com relação aos custos diferenciais atribuídos para *gaps*. Quadrados em azul indicam a presença do clado de acordo com o mapa do *plot* de sensibilidade ao lado do cladograma.

Exercicio 10.1

Neste exercício você deverá fazer uma análise de sensibilidade considerando os dados em `partition1.fas`, `partition2.fas` e `partition3.tnt` aplicando os conceitos e componentes técnicos apresentados acima. Sua análise de sensibilidade deverá contemplar todas as matrizes de custos disponíveis no diretório associado a esse tutorial (*i.e.*, `m111`, `m112`, `m121`, etc.).

10.3 Comprimento de ramos & Alinamentos implícitos

10.3.1 Comprimento de ramos

A versão atual de POY permite imprimir topologias com seus respectivos comprimentos de ramo. Este componente do programa era há muito esperado por seus usuários, pois comprimentos de ramos indicam o número aproximado transformações em determinado ramo e é um bom indicativo de divergência entre linhagens. Em POY, as topologias com comprimentos de ramos são requisitadas pelo comando `report(trees:(branches))`. Por *default*, o comprimento de ramos é calculado por um único assinalamento de

estado de caráter para cada determinado ancestral hipotético (*i.e.*, HTU). No entanto, é possível solicitar ao POY que obtenha e reporte o número mínimo e máximo dos comprimentos de ramos com os comandos `report(trees:(branches:min))` e `report(trees:(branches:max))`, respectivamente.

Exercicio 10.2

Neste exercício você deverá fazer uma análise dos dados em `partition2.fas` em que todas as formas de cálculo de comprimento de ramos é registrada em arquivos de saída individuais. Posteriormente, avalie seus resultados e responda: Todos os ramos apresentam o mesmo comprimento? Justifique.

Exercicio 10.3

Neste exercício você deverá fazer uma análise simultânea dos dados em `partition1.fas`, `partition2.fas` e `partition3.tnt` no qual o arquivo de saída é uma topologia contendo os comprimentos de ramos. Após a análise, você deverá gerar uma figura em FigTree.

10.3.2 Alinhamento implícito

A otimização de alinhamentos em determinada topologia, definido no Tutorial 9 como *Tree Alignment Problem* [**TAP**; 17], é o objetivo de POY. Como vocês observaram nos exercícios anteriores, é possível derivar um padrão filogenético a partir de sequências não-alinhadas por otimização direta na qual nenhuma referência ao alinhamento múltiplo é feito. Wheeler [18] definiu como **alinhamento implícito** o procedimento no qual POY produz a diagnose de uma topologia baseada em sequências não alinhadas cujo resultado foi definido como "*'lines of correspondence' [that] link ancestor–descendent states and, when displayed as linearly arrayed columns without hypothetical ancestors, are largely indistinguishable from standard multiple alignment.*" Embora o autor considere que em alguns caso a distinção seja difícil, alinhamentos implícitos são baseados em esquemas de sinapomorfias e em alguns casos, estes alinhamentos são visualmente estranhos, principalmente quando envolvem eventos de inserções e deleções [consulte 18].

O comando para que POY imprima o alinhamento implícito de uma ou mais topologias é `report(implied_alignment)` ou sua versão abreviada `report(ia)`. No entanto, esse comando insere algumas linhas no início do arquivo que não permitem seu uso imediato por outros

programas, como por exemplo BioEdit. Alternativamente, o comando `report(fasta)` produz a alinhamento implícito no formato FASTA, o que poderá ser utilizado imediatamente por outros programas que lêem esse formato de arquivo.

Exercicio 10.4

Você deverá gerar um alinhamento implícito resultante de uma análise filogenética por otimização direta em POY do arquivo `partition2.fas`. Após gerar esse alinhamento implícito, inspecione-o em BioEdit. Posteriormente, transforme esse alinhamento em uma matriz que possa ser analisada em TNT, faça uma busca e compile seus resultados no espaço abaixo.

10.4 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. Saltelli, A. em *Sensitivity Analysis* eds. Saltelli, A; Chan, K & Scott, E. M. Chichester: John Wiley & Sons, 2000.
5. Wheeler, W. C. 1995. Sequence alignment, parameter sensitivity, and the phylogenetic analysis of molecular data. *Sistematic Biology* **44**(3): 321–331.

6. Wheeler, W. C. & Hayashi, C. Y. 1998. The phylogeny of extant chelicerate orders. *Cladistics* **14**: 173–192.

7. Ramírez, M. J. 2006. Further problems with the incongruence length difference test: "hypercongruence" effect and multiple comparisons. *Cladistics* **22**: 289–295.

8. Giribet, G. 2003. Stability in phylogenetic formulations, and its relationship to nodal support. *Systematic Biology* **52**: 554–564.

9. Giribet, G. & Wheeler, W. C. 2007. The case for sensitivity: a response or Grant and Kluge. *Cladistics* **23**: 1–3.

10. Grant, T. & Kluge, A. G. 2003. Data exploration in phylogenetic inference: scientific, heuristic, or neither. *Cladistics* **19**: 379–418.

11. Giribet, G. & Wheeler, W. C. 1999. On gaps. *Molecular Phylogenetics and Evolution* **13**(1): 132–143.

12. Phillips, A.; Janes, D. & Wheeler, W. C. 2000. Multiple sequence alignments in phylogenetic Anlalysis. *Molecular Phylogenetics and Evolution* **16**(3): 317–330.

13. Wheeler, W. C. & Giribet, G. em *Sequence Alignment: Methods, models, concepts and strategies* ed. Rosemberg, M. S., 337. Berkeley, CA: University of California Press, 2009.

14. Sanders, J. G. 2010. Program note: Cladescan, a program for automated phylogenetic sensitivity analysis. *Cladistics* **26**: 114–116.

15. Machado, D. J. 2015. YBYRÁ, a fast and resourceful tool for sensitivity analysis. *BMC Bioinformatics Journal* **16**: 204.

16. Rambaut, A. Figtree: Tree Figure Drawing Tool. `http://tree.bio.ed.ac.uk/software/figtree/`. Version 1.3.1.

17. Sankoff, D. 1975. Minimal mutation trees of sequence. *SIAM Journal on Applied Mathematics* **28**: 35–42.

18. Wheeler, W. C. 2003. Implied alignment: a synapomorphy-based multiple-sequence alignment method and its use in cladogram search. *Cladistics* **19**: 261–268.