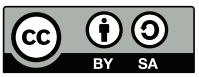
BIZ0433 - INFERÊNCIA FILOGENÉTICA: FILOSOFIA, MÉTODO E APLICAÇÕES.

Tutoriais
Prática e Teoria em Cladística Computacional

Responsável: Prof. Fernando P. de L. Marques (fernando@ib.usp.br)
Departamento de Zoologia
Instituto de Biociências
Universidade de São Paulo
Rua do Matão, tv. 14, no. 101 - Cidade Universitária
05508-090 - São Paulo - Brasil

São Paulo
Versão de 28 de junho de 2023



Sumário

Lista de Tabelas

Lista de Figuras

Tutorial 1

Introdução ao Linux

BIZ0433 - INFERÊNCIA FILOGENÉTICA: FILOSOFIA, MÉTODO E APLICAÇÕES.

Conteúdo

Objetivo

Este tutorial foi, em grande parte, copiado de uma apostila que encontrei nos servidores do IME/USP cujo objetivo era introduzir os sistema Lunix para principiantes. Como não constava nenhuma informação sobre os autores do documento, não pude dar devido crédito ao que foi inserido aqui.

O Objetivo deste tutorial é fazer com que o aluno entenda os princípios básicos do sistema operacional Linux, principalmente no que concerne ao uso de terminais (*i.e.*, modo texto) e comandos de linha na execução de tarefas básicas tais como, movimentação dentro do sistema operacional, criação de diretórios e arquivos, entre outros. Esse tutorial é fundamental para aqueles que estão matriculados nesse curso, uma vez que toda disciplina será baseada neste sistema operacional. No entanto, o aluno não deve ter a expectativa de que todos os comandos apresentados aqui serão absorvidos no primeiro contato. A proficiência nesses comandos vem com o tempo, e é um aprendizado que requer constante aprimoramento, pois sempre se aprende algo de novo. Portanto, este tutorial deverá ser sempre usado como referência para as demais aulas.

1.1 O que é Linux?

O termo Linux é usado em vários contextos com significados diferentes. A rigor, Linux é um kernel. No entanto, em alguns contextos, Linux significa sistema operacional (não qualquer sistema operacional, mas um que use o kernel Linux).

Sistema Operacional: é um *software* que serve de interface entre o computador e o usuário, gerenciando recursos (como memória, processamento, entre outros).

Kernel: é o núcleo ou cerne do sistema operacional (é a parte deste que fica mais "próxima" do hardware).

Você pode agora estar se perguntando se deve chamar apenas o kernel de Linux. Como dito anteriormente, a rigor, Linux é o kernel. Contudo, a expressão "sistema operacional Linux" tornou-se muito difundida. Outra pergunta pode ter surgido neste ponto: qual o nome do sistema operacional então? Mais uma controvérsia aqui. Quando algum usuário instala "o Linux", ele está instalando o kernel e mais uma série de outros *softwares* (*i.e.*, aplicativos ou programas). Grande parte desses aplicativos pertence a um projeto chamado GNU. Logo, o sistema operacional formado pelo kernel mais utilitários e aplicativos, como defendem alguns, deveria ser chamado de GNU/Linux.

1.1.1 Um breve histórico - Como surgiram o GNU e o Linux:

No ano de 1984, Richard Stallman iniciou o Projeto GNU, que tinha por objetivo criar um sistema operacional que fosse totalmente livre. Esse sistema operacional deveria ser compatível com outro sistema operacional - o UNIX (daí o nome GNU - GNU is Not Unix). No ano seguinte, Stallman fundou a FSF (*Free Software Foundation*), com o propósito de eliminar restrições de uso, cópia e distribuição de *software*.
Por volta de 1991, o sistema GNU estava quase pronto, exceto pelo kernel. Stallman estava trabalhando no desenvolvimento de um kernel chamado Hurd. Ao mesmo tempo, o finlandês Linus Torvalds havia criado um kernel compatível com as aplicações do projeto GNU. A esse kernel foi dado o nome de Linux. Atualmente, Linux tornou-se um termo genérico para se referir a sistemas operacionais "Unix-like" baseados no kernel Linux. Tornou-se, também, o melhor exemplo de Software Livre e de código aberto.

1.1.2 Software Livre e Licença GPL:

Na Seção anterior, foi dito que Stallman pretendia criar um sistema operacional livre e que o GNU/Linux era um exemplo de Software Livre. A definição de Software Livre, dada pela FSF é:

i. Executar o *software* com qualquer propósito (liberdade nº 0).

ii. Estudar o funcionamento do *software* e adaptá-lo às suas necessidades (liberdade nº 1).

iii. Redistribuir (inclusive vender) cópias do *software* (liberdade nº 2).

iv. Melhorar o programa e tornar as modificações públicas para que a comunidade inteira se beneficie da melhoria (liberdade nº 3).

Ao contrário do que as pessoas pensam, Software Livre (do inglês *Free Software*) não é sinônimo de gratuito. O que ocorre é uma confusão envolvendo a palavra "*free*" em inglês, que significa tanto gratuito como livre. Mas o sentido que Stallman queria dar era de "livre". De qualquer forma, a maioria dos *softwares* livres é distribuída de forma gratuita. Grande parte dos projetos de *software* livre (incluindo o GNU/Linux) é distribuída sob a GPL (*General Public License* - Licença Pública Geral), que é a licença idealizada por Stallman e que se baseia nas quatro liberdades citadas anteriormente. Com a garantia destas liberdades, a GPL permite que os programas sejam distribuídos e reaproveitados, mantendo, porém, os direitos do autor de forma a não permitir que essa informação seja usada de uma maneira que limite as liberdades originais.

1.1.3 Distribuições:

Distribuições Linux (também chamadas Distribuições GNU/Linux ou simplesmente distros) consistem em "pacotes" de *software* baseados no kernel Linux que incluem determinados tipos de *software* para satisfazer as necessidades de um grupo específico de usuários, dando assim origem a versões domésticas, empresariais e para servidores.
Exemplos de Distribuições Linux: Ubuntu, Debian, Slackware, Fedora, Red Hat, Arch, Gentoo, Mandriva, openSUSE etc.
Qual é a melhor distribuição é uma questão de necessidade e gosto. Apresentamos a seguir uma breve descrição de algumas distros, para que você possa ter uma ideia de suas principais características.

i. Debian:
A distro Debian (ou Debian GNU/Linux) é desenvolvida pelo Projeto Debian, um grupo de voluntários mantido por doações através da organização sem fins lucrativos Software in the Public Interest (SPI).
Debian baseia-se fortemente no projeto GNU e tem como principais características um alto compromisso com estabilidade e segurança bem como uma grande facilidade no que concerne à instalação de programas, através de um gerenciador de pacotes completo (dpkg) e sua interface (apt), utilizados amplamente em outras distribuições.
A última versão estável desta distro é 7.8 de 10 de janeiro de 2015.

ii. Red Hat Entreprise Linux:
Red Hat Enterprise Linux é uma distro criada pela empresa norte-americana Red Hat. O foco desta distribuição é o mercado corporativo, incluindo versões para servidores e para desktops.
O Red Hat Enterprise Linux não possui um ciclo de lançamentos fixo: a versão atual é a 7 de

2015.

iii. Slackware:

Simplicidade e estabilidade são duas características marcantes na distribuição do Slackware. Muito comum em servidores, procura ser uma distribuição "leve", praticamente sem enfeites e rápida, muito apreciada por usuários mais experientes.
Encontra-se atualmente na versão Slackware 14.1.

iv. Ubuntu:

Ubuntu é uma distro GNU/Linux baseada na distro Debian e é patrocinada pela Canonical. A proposta do Ubuntu é oferecer um sistema operacional que qualquer pessoa possa utilizar sem dificuldades, independentemente de nacionalidade, nível de conhecimento ou limitações físicas (a palavra Ubuntu é de origem africana e significa "humanidade para os outros"). Essa distro oferece um ambiente atualizado e estável, focado na usabilidade e na facilidade de sua instalação.
A cada seis meses, uma nova versão da distro é lançada, a versão atual com suporte longo prazo é Ubuntu 14.04.2 LTS (Trusty Tahr). Os números 14 e 4 são, respectivamente, o ano e o mês do lançamento da versão e a sigla LTS significa "*long time support*" que é geralmente de 5 anos.

1.1.4 Aplicativos, diretórios e arquivos:

i. Aplicativos:

Basicamente, para qualquer programa que você utilizava no Windows, existe uma alternativa no GNU/Linux. Por outro lado, vários dos aplicativos inicialmente concebidos sob "GNU/Linux" também estão disponíveis na versão para Windows (*e.g.*, VLC). Caso você queira saber quais programas são equivalentes entre Linux e Windows, consulte esta página.

ii. Visão geral da organização dos arquivos no Linux:

Grosso modo, pode-se dizer que, no Linux, tudo é arquivo. Se há algo que não seja um arquivo, então este algo é um processo. No GNU/Linux (como no UNIX), não há diferença entre arquivo e diretório, uma vez que um diretório é apenas um arquivo contendo nomes de outros arquivos. Imagens, músicas, textos, programas, serviços e assim por diante são todos arquivos. Dispositivos de entrada e saída, e geralmente, todos os dispositivos, são considerados como arquivos. Todos estes arquivos estão organizados de acordo com uma hierarquia, isto é, há critérios que prevêm os principais diretórios e seu conteúdo (veja Figura 1.1). Estes critérios são definidos por um padrão, o FHS (*Filesystem Hierarchy Standard*).

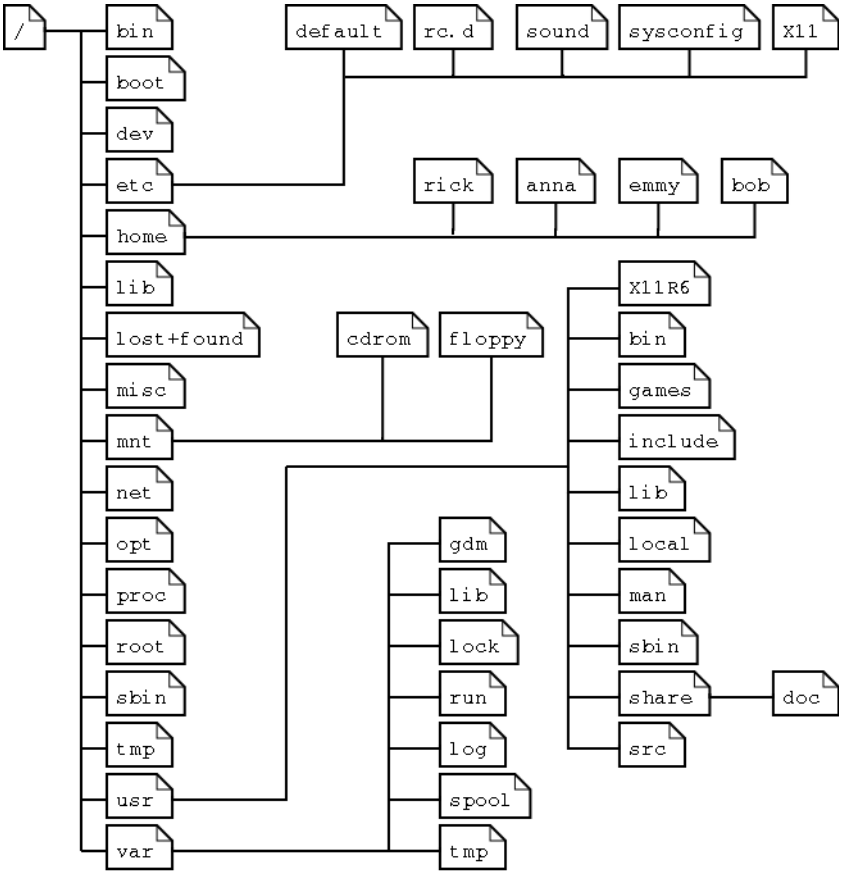


Figura 1.1: Estrutura hierárquica do *file system* do Linux.

No topo da hierarquia de arquivos fica o chamado diretório raiz (ou, mais apropriadamente, diretório *root*), pois a estrutura de diretórios é chamada também de "Árvore de Diretórios". Vejamos alguns exemplos citando os principais diretórios do sistema:

/: Chamado diretório root, é o diretório principal do sistema. Dentro dele estão todos os diretórios do sistema.

/bin: Contém comandos e programas essenciais para todos os usuários (alguns desses comandos serão tratados adiante).

/boot: Contém arquivos necessários para a inicialização do sistema.

/dev: Contém referências para todos os dispositivos, os quais são representados como arquivos com propriedades especiais.

/etc: Contém arquivos de configuração.

/home: Contém os diretórios dos usuários.

/lib: Contém bibliotecas (que são subprogramas ou códigos auxiliares utilizados por programas) essenciais para o funcionamento do Linux, e também os módulos do kernel.

/media: Contém subdiretórios que são usados como pontos de montagem para mídias removíveis, cdroms, discos externos e pen drives, entre outros.

/root: Este é o diretório "home" do super usuário (usuário root). Não confundir com o diretório root, o /. O diretório /root contém os arquivos do usuário root. O diretório / é o topo

da hierarquia de arquivos. **Usuário root** é o administrador do sistema, possui acesso a todos os comandos e arquivos.

/tmp: Contém arquivos temporários.

/usr: Contém programas, bibliotecas, entre outros arquivos.

/usr/bin: Contém os binários de programas não-essenciais (os essenciais ficam no /bin).

/usr/src: Contém os códigos-fonte de alguns programas instalados no sistema.

/var: Contém arquivos "variáveis", como logs, base de dados.

/var/log: Como o próprio nome diz, possui arquivos de log. **Arquivo de log:** é um arquivo que armazena registros de eventos relevantes de um programa ou do sistema.

/var/run: Contém informações sobre a execução do sistema desde a sua última inicialização.

iii. Caminho absoluto vs. Caminho relativo:

Caminho de um diretório (ou de um arquivo) é composto pelos diretórios que devemos percorrer até chegar a ele. Vamos diferenciar caminho absoluto de caminho relativo por meio de um exemplo.

Consideremos, por exemplo, o diretório xinit/. Consideremos que este diretório encontra-se dentro de um outro diretório, o diretório X11/. Este X11/, por sua vez, está dentro do diretório etc/, que, finalmente, está sob o diretório root, o /.

O raciocínio inverso seria: temos o / e dentro o etc/ (/etc/), e dentro o X11/ (/etc/X11) que contém o xinit/ (/etc/X11/xinit). Logo, "/etc/X11/xinit/" é o **caminho absoluto** ("*absolut path*") para o diretório xinit, ou seja, são os diretórios que devemos percorrer, começando pelo /, até o diretório xinit/.

Consideremos agora os mesmos diretórios do caso anterior (/etc/X11/xinit/). Suponhamos agora que estamos no diretório etc/. Para dizer qual é o caminho do diretório xinit/, bastaria dizer apenas X11/xinit - este é o **caminho relativo** ("*relative path*") do diretório (em relação ao diretório /etc – é seu **diretório de trabalho**). Se estivéssemos no diretório X11/, o **caminho relativo** seria simplesmente xinit/.

Em suma, caminho absoluto é aquele que utiliza toda a estrutura de diretórios, ao passo que o relativo toma um diretório como referência e define o caminho a partir daí.

iv. Permissões de acesso:

O Linux foi desenvolvido para ser um sistema multi-usuário. Isto significa que vários usuários podem ter configurações personalizadas, independentes das configurações dos demais usuários, bem como diferentes usuários podem executar tarefas ao mesmo tempo numa mesma máquina. Assim sendo, cada usuário pode querer negar ou permitir o acesso a determinado arquivo ou diretório. Por isso, existem as chamadas permissões de acesso do Linux: para impedir o acesso indevido de outros usuários ou mesmo de programas mal intencionados a arquivos e diretórios.

Mostraremos algumas destas permissões nesta seção. Mais adiante, mostraremos como manipulá-las.

v. Donos, grupos e outros:

No Linux, para cada arquivo são definidas permissões para três tipos de usuários: o **dono** do arquivo, um **grupo** de usuários e os demais usuários (**outros**, que não são nem o dono, nem pertencem ao grupo).

Dono: é o usuário que criou o mesmo, seu dono. Somente o dono e o usuário root podem mudar as permissões para um arquivo ou diretório.

Grupo: é um conjunto de usuários. Grupos foram criados para permitir que vários usuários tivessem acesso a um mesmo arquivo.

Outros: são os usuários que não se encaixam nos tipos de usuários supracitados.

vi. Tipos de permissões:

Os três tipos básicos de permissão para arquivos e diretórios são:

r (read): permissão de leitura para arquivos. Caso seja um diretório, permite listar seu conteúdo (com o comando ls, por exemplo - que será visto adiante).

w (write): permissão de escrita para arquivos. Caso seja um diretório, permite a gravação de arquivos ou outros diretórios dentro dele. Para que um arquivo/diretório possa ser apagado, é necessário o acesso à escrita (gravação).

x (execute): permite executar um arquivo. Caso seja um diretório, permite que seja acessado através do comando cd (você verá este comando também adiante, equivale a ”entrar” no diretório).

Em suma, para cada arquivo do sistema, são definidas permissões para o dono do arquivo, para um grupo de usuários e para os demais usuários. Essas permissões são de leitura, escrita e execução (r, w ou x). Você entenderá melhor estes conceitos adiante, mas tente familiarizar-se com eles desde já.

1.2 Modo texto

Não é apenas pelo modo gráfico que o usuário consegue interagir com o sistema. É possível fazer isso pelo modo texto, digitando comandos e nomes de programas para conseguir uma “resposta” do sistema. Por isso, o modo texto é também chamado de linha de comando.

É importante para um usuário do GNU/Linux aprender a trabalhar no modo texto por vários motivos: otimiza várias tarefas, existem alguns programas que só podem ser executados no modo texto e também porque o modo gráfico consome mais recursos computacionais.

Há duas formas de acessar o modo texto do sistema. Você pode acessar um terminal “puro”, pressionando as teclas “Ctrl+Alt+F1” simultanemamente. Na verdade, você pode substituir a tecla F1 por F2, F3, etc até F6. Isso lhe permite iniciar seis seções simultâneas no modo texto. Para retornar ao modo gráfico, basta pressionar as teclas “Ctrl+Alt+F7”! Veja como o modo texto se parece na Figura 1.2).

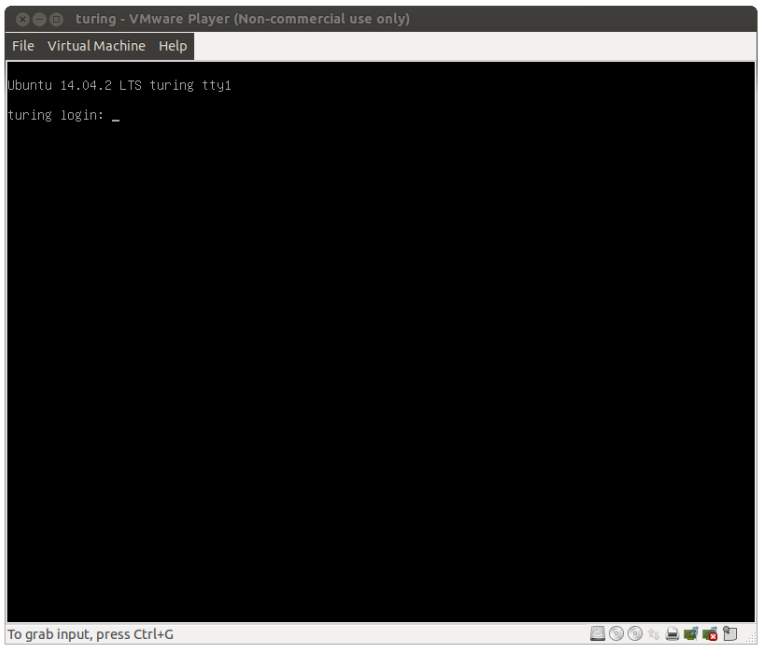


Figura 1.2: Login via terminal de texto (acesso com Ctrl+Alt+F1).

A segunda forma é usar um "emulador de terminal", isto é, dentro do modo gráfico, abre-se um programa que funciona com linha de comando. Na instalação disponibilizada no curso, o terminal está disponível na barra de ferramentas lateral (veja Figura 1.3). De forma geral, em Ubuntu 14.4.02 LTS, você pode acessar o terminal via Painel inicial (também na barra de ferramentas lateral) digitando o termo "terminal" (veja Figura 1.4).

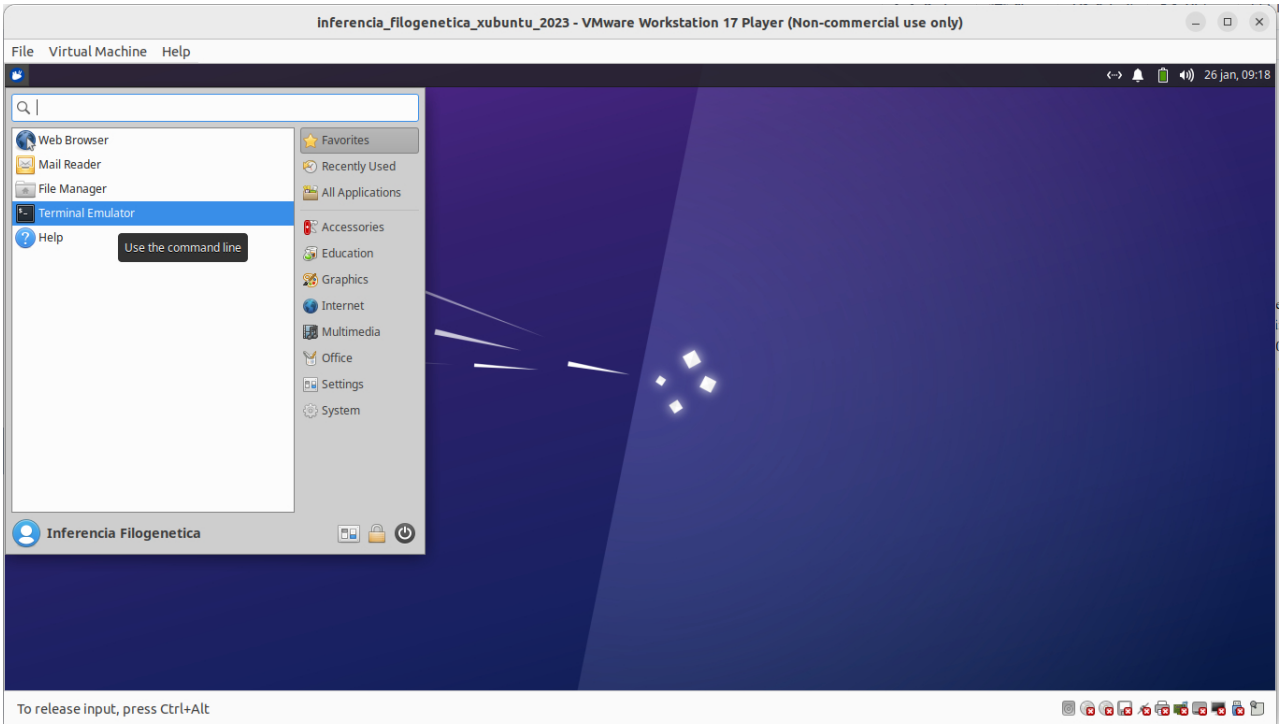


Figura 1.3: Acesso ao terminal via interface gráfica GNOME na barra de ferramentas lateral. [exemplo: imagem do Xubuntu, edição de 2023]

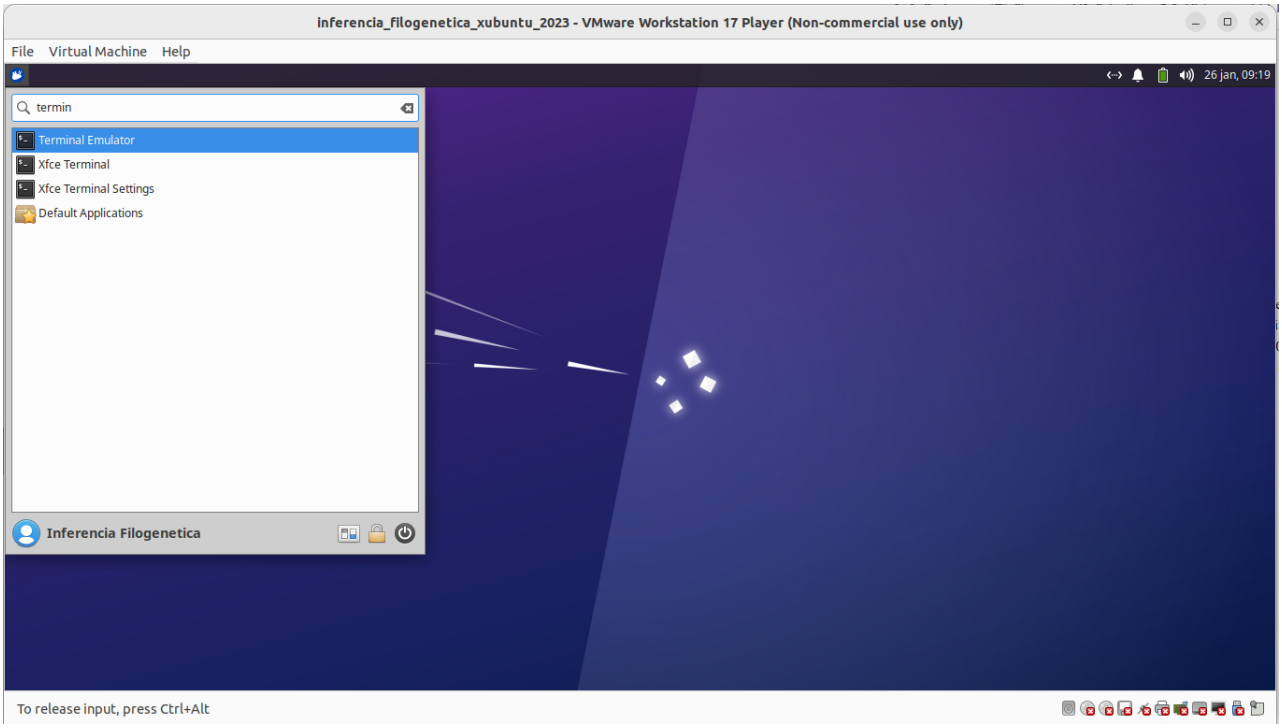


Figura 1.4: Acesso ao terminal via interface gráfica GNOME na barra de ferramentas lateral usando o Painel inicial. [exemplo: imagem do Xubuntu, edição de 2023

1.2.1 SHELL:

De qualquer uma das duas formas, o que você verá rodando (após logar-se ou acessar o Terminal) é um programa chamado shell, que é um interpretador de comandos.

1.2.2 BASH:

O BASH (Bourne Again Shell) é o shell desenvolvido para o projeto GNU, da *Free Software Foundation*, que se tornou padrão nas várias distribuições Linux (incluindo Ubuntu).

Figura 1.5: Estrutura de comandos *std_in* representa *standard input*, informações de entrada para o comando; *std_out* representa *standard output*, resultado do processamento feito pelo comando.

1.2.3 COMANDOS:

Nesta seção, examinaremos alguns comandos simples do BASH. É importante que você saiba que não é preciso decorar os comandos apresentados. Para aprendê-los de fato, você deve ir praticando com os exercícios propostos e conforme a sua necessidade.
Uma noção sobre comandos é importante. Comandos executam tarefas. Não existe comando que não

faça nada! Para executar tarefas, comandos necessitam de material para executar a tarefa para o qual ele foi concebido. Esse "material" é informação, dados. O comando pegará essa informação e retornará o resultado do processamento, ou seja, o resultado de seu trabalho. Veja a representação desses conceitos na Figura 1.5. Todo comando requer implícita ou explicitamente uma informação de entrada, *stardard input*, que será processada e exibida como resultado, *stardard input*, na tela ou redireciona a um arquivo (veja Seção **??** abaixo). Guarde esta estrutura, ela será fundamental para você entender o conceito de PIPES no final deste tutorial (Seção **??**)

1.2.3.1 Prompt:

O prompt do BASH tem a seguinte aparência:

```
alan@turing:~$
```

No caso de `alan@turing:~$` **alan** é o nome do usuário, **turing** é o nome da máquina, "~" é o diretório em que o usuário se encontra, neste caso, "~" representa o diretório **home** do usuário, e o "**$**" é o símbolo do tipo de usuário (nesse caso, um usuário normal). Se fosse o usuário root (administrador do sistema), o símbolo seria "**#**".

1.2.3.2 Sintaxe dos comandos:

É importante lembrar que a linha de comando é *case sensitive*, isto é, diferencia letras maiúsculas de minúsculas. Portanto, "echo" é diferente de "Echo", que são diferentes de "ECHO". Isso também vale para nomes de diretórios e arquivos. Os comandos são, em geral, em letras minúsculas. Muitos deles aceitam argumentos. Os argumentos que começam com um (ou dois) "-" são opções. Por exemplo:

```
$ comando -opção1 -opção2 -opção3 argumento
```

Quando os argumentos forem arquivos ou diretórios, tanto o caminho absoluto como o relativo poderão ser usados. Esta linha de comando assume a priori que se algum arquivo faz parte dos argumentos, ele está no seu diretório de trabalho.
Outro ponto importante é que você pode digitar os comandos e nomes de arquivos ou diretórios pela metade e depois pressionar "Tab". O shell "tentará completar" o que falta para você. Se houver mais de uma opção para completar o que foi digitado, as alternativas possíveis serão mostradas. Este é um recurso que facilita muito o uso da linha de comando. Vamos então aos comandos, você deverá abrir um terminal para que faça alguns dos exercícios.

1.2.3.3 pwd (print working directory):

Mostra o nome e o caminho do diretório atual, ou seja, diretório em que o usuário está, o diretório de trabalho.

```
$ pwd
   $ /home/alan
```

1.2.3.4 ls (list):

Lista os arquivos e subdiretórios de um ou mais diretórios.
Sintaxe básica:

```
$ ls [opções] [diretório1] [diretório2] ...
```

Exemplos:

i. O comando abaixo lista os diretórios e arquivos do /:

```
$ ls /
```

ii. O comando abaixo lista os diretórios e arquivos do /etc:

```
$ ls /etc
```

iii. Para listar o conteúdo do / e do /etc, de uma só vez, use:

```
$ ls / /etc
```

Exercício 1.1

Liste o conteúdo do diretório /tmp.
Para listar o conteúdo do diretório atual, basta digitar apenas "ls". Se o usuário estiver em seu diretório home e digitar ls, a saída será os arquivos e diretórios contidos no /home/alan.

Suponha ainda que o usuário encontra-se em seu diretório home. Existe, dentro do home do usuário, um diretório chamado "Documentos". Se quisermos listar o conteúdo deste, podemos usar o comando

```
$ ls /home/alan/Desktop
```

mas também podemos usar o caminho relativo (lembra-se?):

```
$ ls Desktop
```

Opções:

-a (ou all): Lista todos os arquivos e diretórios, incluindo os ocultos. No GNU/Linux, os arquivos e diretórios ocultos começam por ".". Quando usamos o comando ls como anteriormente (sem nenhuma opção), esses arquivos não são listados.
Exemplo:
O comando abaixo listará todos os arquivos e diretórios contidos no barra, incluindo os ocultos.

```
alan@turing:~$ ls -a
```

Exercício 1.2

Liste todo o conteúdo do seu diretório home, incluindo os itens ocultos. (Quando fizer isso, você notará que dois itens "estranhos" foram listados: o "." e o "..". Eles representam, respectivamente, o diretório atual e o diretório acima. Se você estiver em seu diretório home e usar o comando "ls ../", o conteúdo do /home será listado).

-R: Lista o conteúdo de um diretório e dos subdiretórios, recursivamente. Quando você utiliza o comando ls, os arquivos e diretórios contidos num determinado diretório são mostrados. Usando a opção -R, serão listados os arquivos contidos num determinado diretório, e para cada subdiretório também serão listados os arquivos e diretórios nele contidos. E para cada um desses diretórios, também será listado todo o seu conteúdo e assim sucessivamente. Se você usasse "ls -R /", o conteúdo de todos os diretórios seria mostrado (não execute esse comando, pois o *output* é longo, ele está aqui apenas para que você entenda o que faz esta opção).

-l: Usa o formato longo para listagem, o que significa que serão listados detalhes sobre cada arquivo e diretório mostrado. Vamos examinar que detalhes são estes. O comando

```
$ ls -l /
```

imprime:

Tutorial 2

Manipulação de texto

BIZ0433 - INFERÊNCIA FILOGENÉTICA: FILOSOFIA, MÉTODO E APLICAÇÕES.

Conteúdo

Objetivo

O objetivo deste tutorial é apresentar algumas ferramentas disponíveis em LINUX para visualizar e manipular arquivos textos. Aprender o básico sobre essas ferramentas é fundamental, pois todos os programas ou aplicativos que iremos utilizar durante o curso requerem dados de entrada e imprimem informações em arquivos textos. Este tutorial não explora de forma exaustiva todos os conceitos apresentados relacionados às ferramentas de manipulação de texto disponíveis. No entanto, introduz termos e conceitos, bem como comandos e aplicativos, que certamente lhes darão bases para executar muitas tarefas apresentadas ao longo do curso. Considerem, no entanto, que estas ferramentas poderão ser melhor estudadas fora da sala de aula à medida em que encontre necessidades especiais para executar determinadas operações. 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/
```

2.1 Arquivos textos

Uma das grandes vantagens dos sistemas Unix/Linux é que a grande maioria dos arquivos que gere e sustentam o sistema, bem como aqueles manipulados por aplicativos, são arquivos texto. Durante o curso, você verá que todos os arquivos de entrada (*i.e.*, *input files*) e saída (*i.e.*, *output files*), são arquivos aos quais você terá acesso direto via editores de texto ou que vocês serão capazes de imprimí-los diretamente no terminal para visualização. Adicionalmente, você notará que a manipulação (edição) de textos é muito comum. Isso porque arquivos de saída de determinados programas, ou parte deles, podem ser utilizados como arquivos de entrada para outros programas em análises subsequentes.

Há uma lista de editores de texto disponíveis para todos os sistemas, mas o que queremos usar são aqueles que não inserem caracteres ocultos em seu arquivo (*e.g.*, *Word* da MicroSoft Windows). Dentre eles podemos citar *Notepad* para sistemas Windows, *TextWrangler* para MAC, e `vim`, `elvis`, `emacs`, `nedit`, `nano`, `kedit`, `gedit`, entre muitos outros para sistemas LINUX/UNIX. Para Linux, se o sistema no qual você está operando disponibiliza interfaces gráficas e seu ambiente de *Desktop* é GNOME - como é o caso da imagem disponibilizada no curso - o `gedit` é um editor de texto muito versátil. Dentre aqueles que não requerem interface gráfica e, desta forma, são apropriados para uso em terminais, nós iremos adotar `nano`. Para aqueles que estiverem muito interessados em usar editores dessa natureza, considerem estudar o `emacs`, é inacreditável o que você pode fazer com ele! Ao explorar esses editores, considerem que o conforto e poder desses aplicativos são inversamente proporcionais [1]!

2.1.1 Editores de textos

2.1.1.1 GEdit

Para abrir o `gedit` basta procurar o aplicativo no Painel Inicial do Ubuntu (veja Figura 2.1) e digitar a palavra `gedit` ou verifique se ele já se encontra na barra lateral de aplicativos do seu sistema. Abra o `gedit` e com ele abra o arquivo `10_tax_all_1000_trees.tre` que está no diretório `tutorial_02`. Veja as opções do programa na barra de menu e explore um pouco o que ele pode fazer. Note que ele não difere muito do que você está acostumado a fazer com o seu editor de texto (*e.g.*, *Word* da MicroSoft Windows). Este editor é intuitivo e eu acredito que não seja necessário explicar como ele funciona exceto pelo fato de que você pode abrí-lo diretamente via terminal com o comando:

```
$ gedit &
```

Observação: O símbolo "`&`" apenas faz com que o terminal rode o `gedit` no *backgroud*. Se o "`&`" é omitido, seu terminal fica vinculado à execução do `gedit` até que você feche o programa.

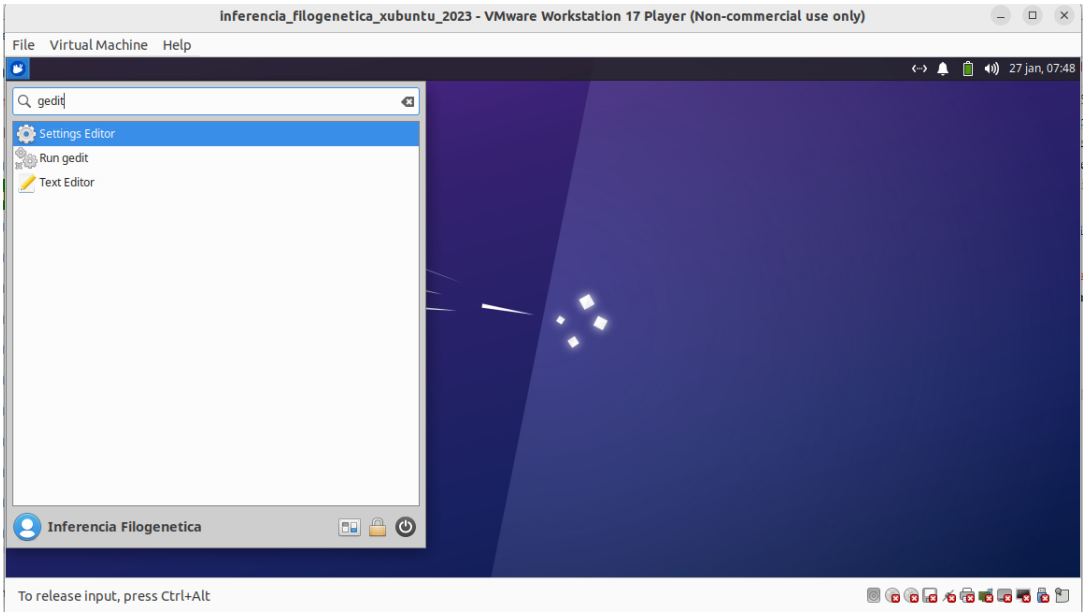


Figura 2.1: Abrindo `gedit` na imagem de Xubuntu via Painel inicial. Observe que a barra lateral de seu sistema lhe permite acesso direto ao gedit.

2.1.1.2 nano

O `nano` é um editor que deve ser executado a partir de um terminal, e se concentra em simplicidade. O `nano` é um clone do antigo editor de texto `pico`, o editor para o cliente de e-mail `pine`, que foi muito popular lá pelos anos 90, em UNIX e sistemas do tipo LINUX. O `nano` foi criado em 1999 com o nome de "`TIP`" (uma sigla, um acrônimo recursivo que significa "`TIP Isn't Pine`", por Chris Allagretta. Allagretta decidiu criar este clone do `pico` porque o programa não foi liberado sob a GPL. O nome foi mudado oficialmente em 10 de janeiro de 2000 para diminuir a confusão entre o novo editor e o comando "`tip`" (o comando "`tip`" é comum em Sun Solaris, uma distribuição de UNIX).
O `nano` usa combinações muito simples de teclas para trabalhar com arquivos. Um arquivo é aberto ou iniciado com o comando:

```
$ nano <arquivo>
```

Onde `<arquivo>` é o nome do arquivo que você deseja abrir.
Se você executa em um terminal o comando:

```
$ nano teste.txt
```

O `nano` irá abrir um arquivo vazio se `teste.txt` não existir no seu diretório de trabalho. No entanto, se você abrir o mesmo arquivo que abriu anteriormente utilizando `nano`, o arquivo `10_tax_all_1000_trees.tre` que está no diretório `tutorial_02`, você deverá observar o terminal ilustrado na Figura 2.2.

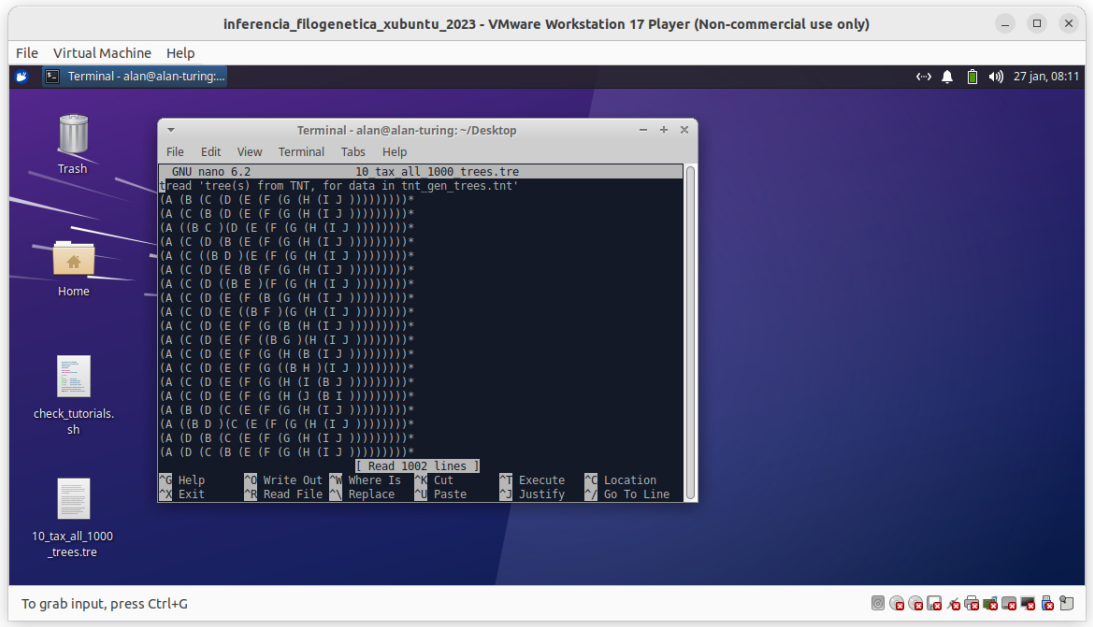


Figura 2.2: Abrindo `nano` na imagem do Xubuntu via terminal.

Quando o arquivo estiver aberto no `nano`, você verá uma pequena lista de sintaxe de comando na parte inferior da janela do terminal (marcado em vermelho na Figura 2.2). Todas as combinações de teclas para o `nano` começam com a tecla `Ctrl`, o `Ctrl` convencionalmente é representado pelo símbolo "^". Para executar um comando você deve manter a tecla `Ctrl` pressionada e clicar na segunda tecla para executar a ação. As combinações mais comuns para o `nano` são:

`Ctrl-x` - Sai do editor. Se você estiver no meio da edição de um arquivo, o processo de saída irá perguntar se você quer salvar seu trabalho.
`Ctrl-R` - Ler um arquivo em seu arquivo de trabalho atual. Isso permite que você adicione o texto de outro arquivo enquanto trabalha dentro de um novo arquivo.
`Ctrl-c` - Mostra a posição atual do cursor.
`Ctrl-k` - "recorta" o texto.
`Ctrl-U` - "cola" o texto.
`Ctrl S` - Salva o arquivo e continua trabalhando.
`Ctrl-T` - verifica a ortografia do seu texto.
`Ctrl-w` - faz uma busca no texto.
`Ctrl-a` - leva o cursor para o início da linha.
`Ctrl-e` - leva o cursor para o fim da linha.
`Ctrl-g` - mostra a ajuda do `nano`.

Existem muitos outros comandos disponíveis no `nano`. Para ver a lista de comandos, use o comando **`Ctrl-g`**.

Exercício 2.1

Você deverá criar um arquivo chamado `matrix.txt` utilizando `Nano` com o seguinte conteúdo:

```
p04 0000000
p02 1110000
p05 1110000
p06 1101000
p03 1101000
p01 1000100
p08 1000110
p10 1000111
p09 1000111
```

2.1.1.3 Outras formas de visualização de texto

Há outras formas de visualizar e/ou obter informações rápidas de arquivos texto. Abaixo iremos explorar algumas delas:

2.1.1.3.1 Comando `cat` : O comando `cat` foi introduzido a vocês no Tutorial 1. Na ocasião, aprenderam que este comando permite a impressão do conteúdo de arquivos texto no terminal, bem como seu uso para concatenar arquivos textos utilizando os caracteres de direcionamento (*i.e.*, “>” e “»”; veja o Tutorial 1). No entanto, o comando `cat` pode também ser utilizado para criar arquivos de forma bem mais rudimentar do que o `nano`. Um exemplo é dado no próximo exercício.

Exercício 2.2

Abra um terminal e digite:

```
$ cat > tnt_head.txt
```

Você deverá obter uma linha vazia no terminal. Digite:
“`xread`” (sem as aspas)
Pressione ENTER.
Agora digite:
“`7 10`” (novamente sem as aspas)
Pressione ENTER.
Finalmente pressione `Ctrl-D`

Examine o conteúdo de `tnt_head.txt`:

```
$ cat tnt_head.txt
```

Você deverá ter obtido um arquivo com o seguinte conteúdo:

```
xread
7 10
```

Muito bem! Você irá usar esse arquivo em breve, portanto deixe-o no diretório `tutorial_02`.

2.1.1.3.2 Comando `sort`: O `sort` é um programa interno de Linux que organiza as linhas de um arquivo texto ou da entrada padrão. A organização é feita por linhas e as linhas são divididas em campos que é a ordem em que as palavras aparecem na linha separadas por um delimitador (normalmente um espaço). Organizar aquivos pode facilitar muito a leitura, principalmente de tabelas e, em alguns casos ou seu arquivo é muito grande para ser organizado por programas como Excel ou você não terá acesso à interfaces gráficas. Portanto, o `sort` pode ser uma ferramenta muito poderosa para organizar informação.

Sintaxe:

```
$ sort -opção1 -opção2 -opção3 argumento
```

O `argumento` é, via de regra, o arquivo que contém as informações que você quer organizar. Há diversas opções para esse programa (veja `$ man sort`, para maiores detalhes); no entanto, iremos apresentar algumas poucas que podem ser úteis utilizando alguns exemplos.

Veja uma utilidade simples deste programa. O arquivo `sort_example_1.tre` contém uma série de topologias compiladas pelo POY [2, 3] – um programa desenvolvido para análises de homologia dinâmica, que veremos no Tutorial 9. Para saber o número de topologias contidas neste arquivo basta usar o comando `wc` com a opção `-l` (veja Tutorial 1). O programa `sort` tem o termo "`-u`" como uma de suas opções, nela são consideradas apenas as linhas que são únicas, ou seja, diferentes. Desta forma, a linha de comando:

```
$ sort -u sort_example_1.tre
```

retorna apenas as topologias únicas contidas neste arquivo.

Exercício 2.3

Quantas topologias únicas existem neste arquivo?.

Veremos outras utilidades do `sort`. Verifique o conteúdo do arquivo `random_taxon_table.tsv` no diretório `tutorial_02` utilizando o comando `cat`. A extenção "`.tsv`" (***T**ab **S**eparated **V**alues*) é utilizada para arquivos em que os valores ou conjunto de textos estão separados por `TAB`. Você consideraria que ele está organizado? Se você consultar o manual do `sort` utilizando o comando `man sort` constatará que há uma opção que permite verificar se seu arquivo possui ou não alguma ordem. A opção é a seguinte:

```
...
-c, -check, -check=diagnose-first
check for sorted input; do not sort
...
```

Se você executar:

```
$ sort -c random_taxon_table.tsv
```

Você deverá obter:

```
sort: random_taxon_table.tsv:3: desordenado: ...
```

A forma mais básica de organizar esse arquivo é executando `sort` sem nenhuma opção. Tente você mesmo e verifique como `sort` organizou seu arquivo. Note que o `sort` organizou sequencialmente as colunas de sua tabela a partir do ordenamento dos gêneros. Talvez isso fique mais óbvio se ordenarmos esses dados por estados. Para isso vejamos a opção **`-k`** de `sort`. Veja o que diz a documentação desse programa para esta opção (`man sort`):

```
...
-k, -key=POS1[,POS2]
start a key at POS1 (origin 1), end it at POS2 (default end of
line).  See POS syntax below
...
POS is F[.C][OPTS], where F is the field number and C the
character position in the field; both are origin 1.  If neither
-t nor -b is in effect, characters in a field are counted from
the beginning of the preceding whitespace.  OPTS is one or more
single-letter ordering options, which override global ordering
options for that key.  If no key is given, use the entire line
as the key.
```

```
...
```

As posições às quais a documentação se refere são os conjuntos de texto separados pelo delimitador, nesse caso TAB (tabulador). Por exemplo, note que a terceira coluna do arquivo `random_taxon_table.tsv` inclui os estados de origem do material listado. Vamos organizar a lista de exemplares deste documento por estado.

Se você executar:

```
$ sort -k 3 random_taxon_table.tsv
```

Você deverá obter:

```
Genus_3 sp_1 AC imaturo MZUSP 0.46 0.6 0.93 0.66 0.33
Genus_3 sp_1 AC imaturo MZUSP 0.87 0.82 0 0.1 0.8
Genus_4 sp_1 AC imaturo MZUSP 0.98 0.98 0.36 0.43 0.98
Genus_4 sp_0 AC indet.  BMNH 0.69 0.3 0.25 0.12 0.63
Genus_2 sp_1 AC indet.  MZUSP 0.16 0.54 0.89 0.08 0.04
Genus_1 sp_0 AC indet.  MZUSP 0.41 0.99 0.66 0.4 0.33
Genus_2 sp_2 AC indet.  MZUSP 0.52 0.96 0.38 0.76 0.05
Genus_2 sp_5 AC indet.  MZUSP 0.75 0.02 0.1 0.07 0.39
...
```

Observe o que acontece quando você delimitar o ordenamento à terceira coluna:

Se você executar:

```
$ sort -k 3,3 random_taxon_table.tsv
```

Você deverá obter:

```
Genus_1 sp_0 AC indet.  MZUSP 0.41 0.99 0.66 0.4 0.33
Genus_2 sp_1 AC indet.  MZUSP 0.16 0.54 0.89 0.08 0.04
Genus_2 sp_1 AC maturo BMNH 0.3 0.53 0.38 0.43 0.34
Genus_2 sp_2 AC indet.  MZUSP 0.52 0.96 0.38 0.76 0.05
Genus_2 sp_3 AC maturo AMNH 0.68 0.93 0.2 0.7 0.93
Genus_2 sp_4 AC maturo BMNH 0.52 0.06 1 0.31 0.84
Genus_2 sp_5 AC indet.  MZUSP 0.75 0.02 0.1 0.07 0.39
Genus_3 sp_1 AC imaturo MZUSP 0.46 0.6 0.93 0.66 0.33
...
```

Exercício 2.4

i. O que acontece com o ordenamento de duas colunas de seu arquivo quando você executa a linha de comando abaixo?

```
$ sort -k 5,5 -k 3,3 random_taxon_table.tsv
```

ii. Você deverá criar um arquivo chamado **`matrix.tnt`** em que as duas primeira linhas contenham o texto que você inseriu no arquivo `tnt_head.txt`, criado anteriormente, seguido do texto **ordenado** por táxon da matrix contida no arquivo `matrix.txt` e que a última linha contenha um ponto e vírgula (*i.e.*, "`;`"). Esta última etapa pode ser realizada com o comando `echo ';'` direcionado de forma apendiciada para o arquivo final. Em nenhum momento você deverá usar qualquer editor de texto! O comando `cat`, `echo` e as formas de redicionamento são suficienetes para criar esse arquivo.

Se você conseguiu **parabéns!**, você acabou de criar um arquivo de entrada para o programa de análise filogenética chamado **TNT** [4].

Finalmente, para terminar a discussão sobre o `sort` vejamos o conteúdo do arquivo `random_taxon_table.csv`. Ele contém as mesmas informações que o arquivo `random_taxon_table.tsv`. A extensão "`.csv`" refere-se à documentos cujo delimitador é uma vírgula (***C**omma **S**eparated **V**alues*), muito comum em bases de dados e uma forma de exportar informações de planilhas eletrônicas (*i.e.*, Excel e/ou OppenOffice Calculator) para arquivos que possam ser processados como textos.

Se você tentar executar as mesmas linhas de comando anteriores para o ordenamento do conteúdo deste arquivo, principalmente no que concerne ao ordenamento por valores internos, você não irá conseguir. Isso porque o `sort` usa, por *default*, espaços como delimitadores. No entanto, em `sort` há como você especificar o delimitador com a opção **`-t`**, vejamos o que diz a documentação:

```
...
-t, -field-separator=SEP
use SEP instead of non-blank to blank transition
...
```

Confuso não!? Enfim, o que esta opção define é qual delimitador você quer especificar. Por exemplo, se seu delimitador é "`,`" sua opção dever ser `-t ','`.

Se você executar:

```
$ sort -t ',' -k 3,3 random_taxon_table.csv
```

Você obterá o mesmo ordenamento feito anteriormente pelos estados de procedência.

Exercício 2.5

i. Considere o conteúdo do arquivo `indexed_trees.txt` no diretório `tutorial_02` que possui a sequinte estrutura:

```
(A,(B,(C,(E,(D,F)))))[4.1.4.7];
...
(A,((D,E),(C,(B,F))))[4.3.3.5];
(A,((D,E),(B,(C,F))))[4.3.3.4];
(A,(D,((B,C),(E,F))))[4.3.2.5];
(A,((E,F),(D,(B,C))))[4.3.1.2];
(A,((E,F),(C,(B,D))))[4.2.1.2];
...
```

Esse arquivo possui um conjunto de topologias com um indexador de 4 dígitos entre colchetes (*i.e.*, `[4.1.4.7]`).
Neste exercício você deverá ordenar as topologias pelos seus indexadores.

2.1.1.3.3 ***Começo e final de arquivos:*** Geralmente, arquivos que registram uma análise filogenética são longos e a impressão de seu conteúdo em um terminal excede o limite de linhas que ele retém como registro. Veja por exemplo o arquivo `garli_screen.log`. Este arquivo contém o *log* de uma análise em Garli, um programa de análises filogenéticas que utiliza máxima verossimilhança como critério de otimalidade [5].

Se você executar:

```
$ cat garli_screen.log | wc -l
```

Você observará que este arquivo possui uma quantidade enorme de texto e que ao ser impressa no terminal você perdeu o acesso ao início do documento (veja isso executando o comando).
Há uma série de informações importantes nesse arquivo que devem ser apresentadas quando você descreve sua análise e documenta seus resultados. Por exemplo, o início deste arquivo registra informações quantitativas e qualitativos de seus dados, modelos utilizados, entre outras coisas; ao passo que no final do mesmo arquivo há informações importantes sobre o resultado final de sua análise. Embora os detalhes destas informações sejam irrelevantes no momento, vejamos como podemos acessar o início e o final deste arquivo utilizando os comandos `head` e `tail`.

Se você executar:

```
$ head -n 70 garli_screen.log
```

Você obterá as 70 linhas iniciais deste arquivo.

Por outro lado, se você executar:

```
$ tail -n 35 garli_screen.log
```

Você obterá as 35 linhas finais deste arquivo.

Exercício 2.6

Execute os comandos abaixo e para cada um deles descreva quais tarefas ele efetuam:

```
$ head -n 1 random_taxon_table.tsv >
random_taxon_table_sorted_2.txt
```

Descrição:

_____

_____

```
$ tail -n +2 random_taxon_table.tsv | sort -k 5,5 -k 3,3 -k 4,4 »
random_taxon_table_sorted_2.txt
```

Descrição:

_____

_____

```
$ head -n -1 matrix.tnt | tail -n +3
```

Descrição:

_____

_____

Finalmente, vamos juntar alguns conceitos. Neste componente no exercício, você deverá usar o comando `head/tail` e `sort` em conjunção com os conceitos de redirecionamento do Tutorial 1 – especificamente a seções **??** e **??**, para executá-lo. Você deverá organizar o conteúdo do arquivo `random_taxon_table.tsv` sequencialmente por museu depositado, estado de origem e maturidade. Este arquivo deverá ser salvo com o nome `random_taxon_table_sorted.txt` e o cabeçário (*i.e.*, `#Genero Especie Estado matur. museu Var_1 ..`) deverá estar na primeira linha.

2.1.1.3.4 Comando `less`: Algumas vezes é desejável verificar o conteúdo de um documento total ou parcialmente, principalmente quando você está examinando a estrutura do documento e o arquivo ocupa mais do que seu terminal é capaz de reter. O comando `less` permite que isso seja feito com um arquivo texto e/ou a saída de algum comando.

A síntax do comando é simples:

```
$ less <arquivo>
```

ou ainda:

```
$ comando argumento | less
```

Após evocar o comando `less`, a tecla ENTER faz com que a rolagem do documento ocorra linha por linha ao passo que a tecla PAGE DOWN apresenta a página seguinte. Para sair do `less` basta pressionar a tecla `q`. Há dois comandos que você deve saber para o comando `less`. Se você digitar "`/`" seguido por qualquer padrão de busca (seja uma palavra ou uma expressão regular [Seção 2.2, abaixo]), `less` executa a busca a partir da segunda linha exibida na tela. Se você digitar "`?`" seguido por qualquer padrão, `less` executará a busca no sentido reverso.

Exercício 2.7

Examine o arquivo `garli_screen.log` usando o comando `less` e busque pela palavra `best`. Em qual réplica desta análise encontra-se a melhor solução?

2.1.1.3.5 Comando `grep`: O comando `grep` (de *Global regular expression parser*) pode ser visto como uma forma simplificada de consulta a um arquivo texto, em que cada linha representa um registro. Ele pode ser usado para retirar um conjunto de *strings* (cadeias de caracteres) do resultado de um comando dado ou de um arquivo texto. O `grep` é um dos comandos mais poderosos dentro de seu sistema, basta consultar sua documentação para ver a quantidade de opções disponíveis. Abaixo vamos explorar algumas propriedades desse comando com exemplos relacionados com nossa prática.
Sintaxe:

```
$ grep -opção1 -opção2 ...  argumento
```

Na sua forma mais simples, `grep` pode ser utilizado para imprimir as linhas que contém uma determinada palavra, ou caracteres literais. Considere um arquivo de *log* de uma análise realizada em POY, um programa para analisar caracteres filogenéticos utilizando homologia dinâmica [2]. O arquivo `onychophora_poy_std.err` possui 102418 linhas e registrou 10 horas de análise! Estas análises envolveram iterações de vários algorítimos. O que iremos fazer com `grep` é tentar extrair as informações relacionadas à estas iterações. Os dados que nos interessam estão em linhas com o seguinte padrão:

```
Information :  The search evaluated ...
```

Vamos tentar alguns comandos e ver o resultado que eles produzem:

i. `$ grep Information onychophora_poy_std.err`
Execute o comando acima. Ele imprime a informação que você gostaria de obter?

Certamente não, apenas uma das linhas impressas no final está de acordo com o padrão que estamos buscando. Tente o comando acima substituindo "`Information`" por "`search`". Melhorou? Sim, mas note que o resultado inclui linhas desnecessárias que fogem do padrão desejado!

ii. `$ grep Information onychophora_poy_std.err | grep search`
Vejamos o que acontece com os comandos concatenados acima. Você tem dificuldade em entender o que eles estão fazendo? Simples, o resultado de "`grep Information onychophora_poy_std.err`" serviu de entrada para "`grep search`", ou seja, você filtrou duas vezes.

Veja o que acontece com esses comandos:

```
$ grep '^Information .* search' onychophora_poy_std.err

$ grep '^Information .*\d*\stimes' onychophora_poy_std.err
```

Ambos os comandos imprimem o mesmo resultado do item iiiii, mas se valem de expressões regulares (ou operadores lógicos), tais como "`.*`" e "`.*\d*\s`", que iremos explorar a seguir.

2.2 Expressões regulares (REGEX)

Uma expressão regular, em ciências da computação, define um padrão a ser usado para procurar ou substituir palavras ou grupos de palavras. É um meio preciso de se fazer buscas de determinadas porções de texto. Como todas as buscas, o problema é encontrar [1]. Quanto melhor você definir o que está à procura, melhor será o resultado de sua busca. O uso de Expressões Regulares (ER, Ereg ou RegEx para *Regular Expression*) é um método rápido e simples de manipulação e combinação avançada de *strings*. Você verá que essas expressões podem ser úteis no seu dia a dia, principalmente se suas atividades de pesquisa e/ou estudo incluem manipulação de arquivos texto e busca rápida de informação. Se você utilizar REGEX com habilidade, elas podem simplificar muitas tarefas de programação e processamento de texto, além de permitir outras que não seriam possíveis sem elas [6].

2.2.1 TERMINOLOGIA E ESTRUTURA:

Antes de entendermos como as REGEX são expressas e usadas, é necessário introduzir alguns termos. **Caracteres literais**, *literals*, é um conjunto de caracteres que usamos em uma busca, por exemplo, para encontrar *inu* em *Linux*, *inu* é uma sequência de caracteres (em computação definida como *string*) literais. **Meta-caráter**, *meta character*, é um ou mais caracteres especiais que possuem um significado único e não são usados como **caracteres literais** na busca, ou define um **meta-caráter**. Por exemplo, no item iiiii de 2.1.1.3.5 os termos "`.`" e "`*`" são exemplos

de **meta-caracteres**. **Sequência de escape**, *scape sequence*, é uma forma de indicar que nós queremos usar **meta-caracteres** como **caracteres literais**. Em REGEX uma sequência de escape envolve a inserção de "\" anterior a um **meta-caráter**. Por exemplo, para encontrar um "`.`" ou um "`*`" em um texto é necessário o uso das sequências de escape "`\.`" e "`\*`", respectivamente. A **sequência de caracteres alvo**, *target string*, define a sequência de caracteres que estamos buscando e o **padrão de busca**, *search pattern* ou *construct*, descreve a expressão que estamos usando para buscar a **sequência de caracteres alvo**.

Há três componentes importantes em REGEX: âncoras, conjunto de caracteres e modificadores. **Âncoras** são usadas para especificar a posição do padrão em relação a uma linha de texto e, por definição, são meta-caracteres. **Conjunto de caracteres** correspondem a um ou mais caracteres em uma única posição, e aqui podem ser expressos caracteres literais e/ou meta-caracteres. **Modificadores** especificam quantas vezes o conjunto de caracteres anterior é repetido e, também são considerados meta-caracteres. Um exemplo simples que demonstra todos os três componentes é a expressão regular "`^#*`". O acento circunflexo é uma âncora que indica o início da linha. O caráter "`#`" é um conjunto de caracteres simples que corresponde ao de um único caráter "`#`". O asterisco é um modificador de "`#`" – aceitando zero ou mais ocorrência deste caráter. Esta é uma expressão regular inútil, mas exemplifica os componentes frequentemente encontrados em REGEX.

Vamos entender as duas instâncias em que foram utilizadas REGEX no item iiiii acima e entender os componentes das REGEX utilizadas. Ao executarmos o comando `grep '^Information .* search' onychophora_poy_std.err` a REGEX está contida entre aspas, "`^`", "`.`" e "`*`" são **meta-caracteres** - dentre os quais "`^`" é uma âncora e "`*`" é um **modificador**, ao passo que "`Information`" e "`search`" são **caracteres literais**. No comando `grep '^Information .*\d*\stimes' onychophora_poy_std.err`, "`times`" é um novo **caráter literal** e "`\d`" "`\s`" são novos **meta-caracteres** expressos como **sequência de escape**.

2.2.2 ÂNCORAS:

Não é fácil de fazer uma busca de padrão de caracteres simples que corresponda ao de um único caráter "`*`". O asterisco é um meta-caráter modificador. Em uma REGEX, "`*`" especifica que o caráter, literal ou não, se repete de zero a N vezes pela linha. O fim da linha de caracteres não está incluído no bloco de texto que é pesquisado. O fim da linha é um separador. Expressões regulares examinam o texto entre os separadores. Se você quiser procurar por um padrão que está em uma extremidade ou em outra, você usa âncoras. O caráter "`^`" é a âncora inicial, e o caráter "`$`" é a âncora final. A expressão regular "`^A`" irá corresponder a todas as linhas que começam com um A maiúsculo. A expressão "`A $`" irá corresponder a todas as linhas que terminam com A maiúsculo. Se os caracteres de ancoragem não são usados no final do próprio padrão, então eles

não funcionaram como âncoras. Ou seja, o "^" é apenas uma âncora se for o primeiro caráter em uma expressão regular. O "$" é apenas uma âncora se for o último caráter. A expressão "$1" não tem uma âncora. O mesmo ocorre para a regex "1^". Se você precisa encontrar um "^" no início da linha, ou um "$" no final de uma linha, você deve escapar os caracteres especiais com uma barra invertida (e.x., "^" ou "$", veja Tabela 2.1). Finalmente, o uso de "^" e "$" como indicadores do início ou final de uma linha é uma convenção em várias ferramentas de Linux/Unix. O editor **vi**, por exemplo, usa esses dois caracteres como comandos para ir para o início ou fim de uma linha.

Para ilustrar o conceito de âncoras, vamos executar uma busca no arquivo `random_trees.tre` para identificar em quantas topologias os táxons `A`, `B`, `C` e `D` estão na raiz.

Exercício 2.8

Execute o seguinte comando:

```
$ egrep A random_trees.tre
```

Observe que este comando imprime as linhas (topologia) onde o táxon `A` é encontrado, independente de sua posição. O comando `egrep` é a versão REGEX do `grep`. Para obter as topologias nas quais `A` está na raiz devemos inserir a seguinte REGEX:

```
$ egrep '^\(A' random_trees.tre
```

Nesta REGEX, expressa entre aspas, o **padrão de busca** `'^\(A'` descreve a expressão que estamos usando para buscar uma **sequência de caracteres alvo** em que a topologia (linha) esteja escrita de forma que os caracteres `"(A"` estejam no início da linha.
Finalmente, execute o comando e responda:

```
$ egrep '^\(A|^\(B' random_trees.tre
```

Qual e a fução do pipe (*i.e.*, "`|`") nesta REGEX?

_____

_____

_____

Exercício 2.9

A opção "`-c`" do `egrep`, faz com que esse comando retorne o número de linhas que contém o padrão de busca que você usou. Isto posto, você deverá anotar abaixo o número de topologias que os táxons `A`, `B`, `C` e `D` estão na raiz.

`A:` ___ `B:` ___ `C:` ___ `D:` ___

Exemplos: A Tabela 2.1 ilustra alguns exemplos do padrão e busca com âncoras.

Tabela 2.1: Exemplos de âncoras.

Padrão	Busca
`^A`	"A" no começo da linha
`A$`	"A" no final da linha
`A^`	"A^" qualquer lugar da linha
`$A`	"$A" qualquer lugar da linha
`^^`	"^" no começo da linha
`$$`	"$" no final da linha

2.2.3 Modificadores e caracteres de escape:

As Tabelas 2.2, 2.3 e 2.4 contém padrões de RegEx relacionados com classes de caracteres pré-definidos, caracteres de escape e modificadores. Esta é uma pequena amostra de meta-caracteres disponíveis para expressar padrões de busca em RegEx. O uso de RegEx é pessoal, pois basta examinar alguns destes meta-caracteres para perceber que há diversas maneiras de expressar um mesmo padrão de busca. Para o propósito deste curso, é importante que vocês saibam o que é uma RegEx e como elas funcionam e podem ser usadas. Para atingir este objetivo, alguns poucos exemplos serão apresentados a seguir. Você verá que alguns desses meta-caracteres são usados frequentemente e que o domínio de alguns poucos elementos de RegEx pode ser uma ferramenta muito vantajosa NO SEU DIA A DIA. Como só se aprende na prática, vamos fazer algumas tentativas de utilizar RegEx para extrair algumas informações de alguns arquivos.

Exercício 2.10

i. Considere a seguinte topologia gerada por TNT:

```
(D (H ((E (A F ))(B (J (G (C I )))))))
```

Esta topologia está entre as 2924 árvores contidas no arquivo `random_trees.tre`. Suponha que você queira saber quais são e quantas são as topologias que contém um grupo monofilético formado por `E, A` e `F`. Qual seria seu padrão de busca e quantas topologias contém este clado?
Dica: Consulte a Tabela 2.3 para ver como expressar parênteses literais, a Tabela 2.2 para ver como você busca um padrão ou outro e examine todos os clados possíveis antes de conjugá-los em uma única RegEx. Adicionalmente considere que esses três táxons podem estar relacionados de outra forma (*e.g.*. `(F (A E ))`) o que requer que você considere uma coisa **ou** outra – veja Tabela 2.2.

ii. Considere o seguinte o comando: `egrep '.*T\w.*' random_taxon_table.tsv`
O padrão de busca dessa REGEX está recuperando qual sequência de caracteres alvo?

iii. Em uma única REGEX extraia do arquivo `random_taxon_table.tsv` todos os exemplares coletados nos estados do AM e RO.
Escreva sua REGEX abaixo:

```
$ egrep '                                    '
```

Tabela 2.2: Classes de caracteres pré-definidas utilizadas em REGEX.

Padrão	Busca
`.`	Qualquer caráter
`[r|s]`	Ou "r" ou "s", pode ser escrito "(r|s)"
`[a-z]`	Qualquer letra minúscula
`[A-Z]`	Qualquer letra maiúscula
`[a-zA-Z]`	Qualquer letra maiúscula ou minúscula
`[0-9]`	Qualquer número
`[0-9.-]`	Qualquer número, ponto ou sinal de subtração
`[^-0-9]`	Qualquer caráter exceto um número ou "-"
`[[:alpha:]]`	Qualquer letra (alfabética)
`[[:digit:]]`	Qualquer número (dígito)
`[[:alnum:]]`	Qualquer letra ou número (alfanumérico)
`[[:space:]]`	Qualquer caráter de espaço
`[[:upper:]]`	Qualquer letra maiúscula
`[[:lower:]]`	Qualquer letra minúscula
`[[:punct:]]`	Qualquer caráter de pontuação

Tabela 2.3: Caracteres de Escape usados em REGEX.

Padrão	Busca
`\t`	Caráter de tabulação (TAB)
`\n`	Linha nova
`\)`	Um parêntese literal
`\\`	Uma barra invertida literal

Tabela 2.3 – Continuação.

Padrão	Busca
`\-`	Um hífen literal
`\w`	Qualquer caráter alfanumérico incluindo "_"
`\W`	Qualquer caráter que não seja alfanumérico
`\s`	Qualquer espaço em branco
`\S`	Qualquer espaço que não seja em branco
`\d`	Qualquer caracter que seja um dígito
`\D`	Qualquer caracter que não seja um dígito

Tabela 2.4: Multiplicadores de caracteres em REGEX.

Padrão	Busca
`?`	Uma ocorrência ou nenhuma (Equivale a {0,1})
`*`	Nenhuma ocorrência ou qualquer número de ocorrência (Equivale a {0,})
`+`	Uma ou mais ocorrências (Equivale a {1,})
`r|s`	equivalente a [rs], "r" ou "s"
`^[[:alpha:]]{3}$`	Qualquer palavra de três letras
`^[[:digit:]]{3}$`	Qualquer número com três dígitos
`^a{4}$`	A expressão recupera o padrão "aaaa"
`^a{2,4}$`	A expressão recupera os padrões "aa", "aaa" e "aaaa"
`^a{2,}$`	A expressão recupera os padrões "aa", "aaa", "aaaa", ...

2.3 Sed:

2.3.1 INTRODUÇÃO E SINTAXE BÁSICA:

Sed, de ***stream editor***, é uma das ferramentas mais antigas e usadas em Linux. Ele é um editor não interativo e orientado por linhas. Isso significa que os comandos de edição são inseridos por comandos de linhas ou estão contidos em um arquivo. **Sed** manipula os arquivos sem modificar o original e caso você queira manter a cópia modificada basta direcionar o resultado do comando para um outro arquivo (veja Tutorial 1). **Sed** é fundamentalmente uma ferramenta para substituir textos e sua vantagem está no fato de que ele lê linha por linha, diferentemente editores de texto convencionais que carrega todo o arquivo inicialmente. Por esta razão, **Sed** é capaz lidar com arquivos muito grandes, o que pode ser impraticável utilizando os editores de texto que você tem mais familiaridade. Considere por exemplo o arquivo `sed_Exercício_1.txt` no diretório `tutorial_02`. Este arquivo contém 5000 dados para 20 gêneros que variam de 1 a 20 espécies. No total, este documento possui 180 linhas, 900.360 palavras e 5.403.645 de caracteres! Para cada linha você encontra dados para um gênero, uma espécie e 5000 medidas obedecendo

o seguinte formato: `Genus_01 species_0 0.869 0.610 0.904 ....` Suponha que você desejasse, substituir todos os pontos (“`.`”) por vírgulas (“`,`”). Como você faria isso? A edição convencional em um computador com 8 processadores e 24 G de RAM tomou mais de 9 minutos entre abrir, substituir e salvar a cópia modificada! Outro benefício do **Sed** que você pode criar arquivos contendo comandos de substituição para tarefas repetitivas, algumas das quais você se deparará durante este curso.

Sintaxe:

```
$ sed -opção 'COMANDO_DE_EDIÇÃO' arquivo_de_entrada
```

Por exemplo, execute o comando:

```
$ sed -n 'p' sed_example_1.txt
```

Você obterá:

```
>seq_1
acgcaggaatggcaga
>seq_2
acgcagcaagggacgttt
>seq_3
acgcagctaccgacgttt
>seq_4
acgttctacaccgacgttt
>seq_5
attcaataccgacgttt
```

A opção **`-n`** em conjunção com o modificador **`p`** faz com que **Sed** imprima o texto do arquivo `sed_example_1.txt` no terminal. As substituições em **Sed** são relativamente simples.

Por exemplo, execute o comando:

```
$ sed 's/seq/taxon/' sed_example_1.txt
```

Você obterá:

```
>taxon_1
acgcaggaatggcaga
>taxon_2
acgcagcaagggacgttt
>taxon_3
acgcagctaccgacgttt
>taxon_4
acgttctacaccgacgttt
```

```
>taxon_5
attcaataccgacgttt
```

A sintaxe de substituição é super simples:
`'s/`*`conjunto de caracteres alvo/conjunto de caracteres de substituição/`*`'`

Há alguns detalhes que merecem atenção. Veja por exemplo o comando abaixo.

Execute o comando:

```
$ sed 's/a/A/' sed_example_1.txt
```

Você obterá:

```
>seq_1
Acgcaggaatggcaga
>seq_2
Acgcagcaagggacgttt
>seq_3
Acgcagctaccgacgttt
>seq_4
Acgttctacaccgacgttt
>seq_5
Attcaataccgacgttt
```

Observe que somente a substituição do primeiro par de base da sequência de cada linha foi modificado. Isso porque, por *default*, **Sed** faz a primeira modificação ao encontrar o padrão de busca e parte para a outra linha. No entanto, da mesma forma que o modificador **`p`** gerencia o padrão de impressão de **Sed** o modificador **`g`**, de *global*, faz com que o **Sed** verifique múltiplas ocorrências do padrão de busca na mesma linha. Veja como esse modificador funciona executando o comando abaixo:

```
$ sed 's/a/A/g' sed_example_1.txt
```

Você obterá:

```
>seq_1
AcgcAggAAtggcAgA
>seq_2
AcgcAgcAAgggAcgttt
>seq_3
AcgcAgctAccgAcgttt
```

```
>seq_4
AcgttctAcAccgAcgttt
>seq_5
AttcAAtAccgAcgttt
```

Há uma forma de implementar substituições sequenciais em **Sed** que é fundamental conhecer. Considere o exemplo abaixo.

Se você executar:

```
$ sed -e 's/a/A/g' -e 's/c/C/g' sed_example_1.txt
```

Você obterá:

```
>seq_1
ACgCAggAAtggCAgA
>seq_2
ACgCAgCAAgggACgttt
>seq_3
ACgCAgCtACCgACgttt
>seq_4
ACgttCtACACCgACgttt
>seq_5
AttCAAtACCgACgttt
```

Neste caso em particular, você poderia criar REGEX sequenciais e transformar todas as sequências em letras minúsculas. No entanto, considerem o exemplo abaixo:

Se você executar:

```
$  sed 's/^\w*/\U&\E/' sed_example_1.txt
```
[1]

Você obterá:

```
>seq_1
ACGCAGGAATGGCAGA
>seq_2
ACGCAGCAAGGGACGTTT
>seq_3
ACGCAGCTACCGACGTTT
>seq_4
ACGTTCTACACCGACGTTT
>seq_5
ATTCAATACCGACGTTT
```

[1] Se você estiver utilizando Mac OS X esse comando não funciona. Há variações nas versões de **sed** entre alguns sistemas operacionais.

Uau! Vamos entender as REGEXS utilizadas acima. Meu padrão de busca é "`^\w*`", ou seja, toda linha que inicie ("`^`") com um caráter alfanumérico ("`\w`") que seja seguido por 0 a infinito dos mesmos caracteres ("`*`"), veja Tabelas 2.3 e 2.4. Por outro lado, meu padrão de busca é "`\U&\E`" definem dois meta-caracteres desconhecidos até o momento para você. O meta-caráter "`\U`" transforma todas as letras em maiúsculas, "`&`", concatena o próximo meta-caráter "`\E`" que garante que as modificações efetuadas por "`\U`" sejam mantidas.

Exercício 2.11

Quanto tempo você leva para substituir os pontos (*i.e.*,"`.`") por vírgulas (*i.e.*,"`,`") do arquivo sed_Exercício_1.txt?

2.3.2 ENDEREÇAMENTO:

Como dito anteriormente, **Sed** aplica comandos de edição linha por linha, de seu início (*i.e.*, "`^`") ao fim (*i.e.*,"`$`"), principalmente a opção de edição global (*i.e.*,"`g`") é evocada. O endereçamento em **Sed** limita a ação das edições.

Sintaxe:

```
$ sed -opção 'ENDEREÇAMENTO+COMANDO_DE_EDIÇÃO'
arquivo_de_entrada
```

Vejamos na prática como isso funciona:

O arquivo `sed_example_2.txt` contém 17 linhas das quais 15 delas possuem todas as topologias possíveis para 5 terminais geradas em **TNT** [4]. Portanto, o arquivo possui a seguinte estrutura:

```
tread 'tree(s) from TNT, for data in tnt_gen_trees_T5.tnt'
(A (B (C (D E ))))*
(A (C (B (D E ))))*
...
(A (E (B (C D ))))*
(A (E (C (B D ))))*
(A (E (D (B C ))));
proc-;
```

Neste arquivo, a primeira e a última linha são parte do *tree block* do **TNT** e segue a sintaxe do programa para o comando `tread` cuja função é ler topologias.

Se você executar:

```
$ sed -n 'p' sed_example_2.txt
```

Sed executa o comando *print* (*i.e.*,"`p`") de todas as linhas do arquivo.

No entanto, se você executar:

```
$ sed -n '1p' sed_example_2.txt
```

Sed executa o comando *print* (*i.e.*,"`p`") apenas na primeira linha do arquivo (*i.e.*,"`1`") e você terá:

```
tread 'tree(s) from TNT, for data in tnt_gen_trees_T5.tnt'
```

Os limites de endereçamento são feitos da seguinte maneira.

Se você executar:

```
$ sed -n '1,3p' sed_example_2.txt
```

Sed imprime as 3 primeiras linhas do arquivo.

Observe como você combina o endereçamento com edição.

Se você executar:

```
$ sed -n '2,16 s/\s//gp' sed_example_2.txt
```

Sed executa o comando *print* (*i.e.*,"`p`") e substitui globalmente (*i.e.*,"`s`" com "`g`") as linhas 2 a 16 (*i.e.*,"`2,16`") e você terá:

```
(A(B(C(DE))))*
(A(C(B(DE))))*
...
(A(E(B(CD))))*
(A(E(C(BD))))*
(A(E(D(BC))));
```

Note que um caráter de escape (e meta-caráter) de REGEX "`\s`", que representa espaço em branco, foi utilizado. Portanto, embora nos exemplos acima tenhamos utilizados caracteres literais em substituições, `sed` permite o uso de REGEX.

Exercício 2.12

i. Neste exercício, você deverá usar os conceitos acima para fazer algumas edições no arquivo `sed_example_2.txt`. Uma dica importante: o argumento "`'1d;$d'`" de `sed` imprime todas as linhas do arquivo exceto a primeira e a última linha.

Observe atentamente a estrutura desse arquivo e transforme-o em um arquivo que deverá ser salvo com o nome de `sed_example_2.tre` que contenha apenas as topologias no seguinte formato:

```
(A,(B,(C,(D,E))));
(A,(C,(B,(D,E))));
...
(A,(E,(B,(C,D))));
(A,(E,(C,(B,D))));
(A,(E,(D,(B,C))));
```

Se você conseguiu fazer esse exercício parabéns! Você acaba de transformar um arquivo de topologias de TNT em um arquivo que pode ser lido por outros programas, tais como PAUP [7] e Figtree [8]!

ii. Neste exercício, você deverá criar uma arquivo texto no qual cada linha contenha o padrão de substituição que você utilizou no exercício acima. Por exemplo:

```
1d;$d s/A/B/g
s/X/Y/g
...
```

Você deverá salvar esse arquivo com o nome `tnt2figtree.sed`.

Qual o resultado da execução do comando abaixo?

```
$ sed -f tnt2figtree.sed sed_example_2.txt
```

Esse arquivo que contém as regras de substituições para trasnformar topologias vindas de TNT para que possam ser lidas em outros programas é últil. Guarde-o, pois você deverá usá-lo no futuro.

2.4 Trabalho para entregar

No diretório deste tutorial há dois arquivos destinados a este exercício:

1. `opilio_tree.tre`: Contém uma topologia no formato parentético.

2. `rename_opilio.csv`: Contém uma tabela no formato CSV com os nomes dos terminais que se encontram na topologia. Esse arquivo poder ser aberto em qualquer editor de texto ou no `LibreOffice Calc`. No entanto, voce poderá modificá-lo simplesmente utilizando o terminal e aplicando os conceitos que foram explorados nesse tutorial.

Com esses dois arquivos você deverá gerar uma figura editada da topologia usando Figtree e Inkscape em formato PDF. Para cumprir este objetivo, siga os seguintes passos:

a. Gerar um arquivo com as regras de substituição (veja Excercício 2.3.2) que lhe permita usar o **sed** para fazer as substituições dos terminais no arquivo `opilio_tree.tre` (veja seção 2.3.1).
Uma dica importante: O Figtree requer que os terminais que possuam nomes compostos (*e.g.*, *Hypophyllonomus longipes*), e que portanto, possuam espaços em branco entre palavras, estejam contidos entre aspas (*e.g.*, "`Hypophyllonomus longipes`"). **Sua topologia deverá conter todo o conteúdo da segunda coluna do arquivo CSV**.

b. Execute o **sed** redirecionando o resultado para um arquivo com outro nome.

c. Assista ao vídeo `figtree_1.ogv` para uma explicação breve de como o programa funciona e como você deverá proceder para gerar uma figura em SVG com ele.

d. Assista ao vídeo `inkscape_1.ogv` para uma explicação breve de como o programa funciona e como você deverá proceder para editar a figura e gerar um arquivo em formato PDF com resolução de 150 dpi chamado "`seu_nome_topologia_figura.pdf`".

e. Submeter a figura no formulário de *upload* ao final desta aula.

2.5 Referências

1. Wünschiers, R. 2004. Computational Biology: Unix/Linux, data processing and programming. Berlin, Germany: Springer, 2004. 284.

2. Varón, A.; Vinh, L. S. & Wheeler, W. C. 2010. POY version 4: phylogenetic analysis using dynamic homologies. *Cladistics* **26**: 72–85.

3. Varon, A.; Lucaroni, N.; Hong, L. & Wheeler, W. C. 2011–2014. POY version 4: phylogenetic analysis using dynamic homologies, version 5.0. New York, NY: American Museum of Natural History, 2011–2014.

4. Goloboff, P.; Farris, J. S. & Nixon, K. 2008. TNT a free program for phylogenetic analysis. *Cladistics* **24**: 1–14.

5. Zwickl, D. J. Genetic algorithm approaches for the phylogenetic analysis of large biological sequence datasets under the maximum likelihood criterion. Tese de doutorado (The University of Texas, Austin, 2006).

6. Goyvaerts, J. & Levitahn, S. 2009. Expressões Regulares Cookbook. São Paulo: Novatec Editora Ltda, 2009. 156.

7. Swofford, D. 2003–2016. PAUP*. Phylogenetic Analysis Using Parsimony (*and Other Methods, Version 4.0a131). Sunderland, Massachusetts: Sinauer Associates, 2003–2016.

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

Tutorial 3

Espaço de topologias

BIZ0433 - INFERÊNCIA FILOGENÉTICA: FILOSOFIA, MÉTODO E APLICAÇÕES.

Conteúdo

Objetivo

Este tutorial apresenta os conceitos de grafos, enumeração e espaço de topologias. O objetivo do tutorial é apresentar os requisitos necessários para a compreensão da complexidade computacional associada às buscas de topologias de acordo com critérios de otimalidade. Durante o tutorial, você terá a oportunidade de executar uma série de exercícios cujos resultados gráficos lhe permitirão adquirir uma melhor compreensão dos conceitos em discussão. 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/
```

3.1 Requisitos de sistema

Este tutorial requer que seu computador possua os seguintes aplicativos:

i. TNT: Esse programa [1] deve estar instalado em seu computador e executável pelo comando `$ tnt`.

ii. R: O **R** é um ambiente livre de computação gráfica e estatística. Caso seu computador não possua o **R** instalado, você deverá fazê-lo pelo comando "`sudo apt-get install r-base-core`". Adicionalmente você deverá instalar algumas bibliotecas que são necessárias para a execução deste tutorial: `scatterplot3d` e `vegan`. Para instalar essas bibliotecas você deverá abrir o **R** com o comando "`sudo R`" e executar o seguinte comando no *prompt* do **R**: "`install.packages("scatterplot3d")`".
O **R** solicitará que você escolha um repositório do qual ele deverá baixar o pacote solicitado. Você poderá escolher qualquer um dos países listados. Feita a escolha, o pacote começará a ser instalado.
A próxima biblioteca que deverá ser instalada é "`vegan`". Use a mesma lógica acima.

iii. Java VM: Um dos aplicativos utilizados nesse tutorial requer a instalação de Java Virtual Machine. O repositório do Ubuntu possui várias versões de Java disponível. Na imagem utilizada pelo curso foi instalado o pacote "`oracle-java7-installer`".

iv. YBYRÁ: O YBYRÁ [2] é um aplicativo desenvolvido por Denis J. Machado (Depto. Zoologia – IB/USP). O programa tem uma série de aplicações, tais como cálculo de distâncias topológicas, análise de sensibilidade, diagnose de clados, entre outras. Neste tutorial iremos usá-lo para o cálculo de distâncias entre topologias dentro do espaço de enumeração. Ao longo do curso, iremos usar esse mesmo programa para explorar outras aplicabilidades.

3.2 Conteúdo do diretório

3.2.1 Arquivos

O diretório associado a este tutorial deverá ser baixado da página da disciplina. Nele você encontrará os seguintes itens:

```
-rw-rw-r-- 1 alan alan    1618079 Mar  2 13:02 03_tutorial.pdf
-rw-rw-r-- 1 alan alan       3359 Mar  2 13:02 06_enumeration.tre
-rw-rw-r-- 1 alan alan      35909 Mar  2 13:02 07_enumeration.tre
-rw-rw-r-- 1 alan alan     459269 Mar  2 13:02 08_enumeration.tre
-rw-rw-r-- 1 alan alan    6822899 Mar  2 13:02 09_enumeration.tre
-rw-rw-r-- 1 alan alan  115709579 Mar  2 13:02 10_enumeration.tre
-rw-rw-r-- 1 alan alan       2206 Mar  2 13:02 graph_data_dist.r
```

```
-rw-rw-r-- 1 alan alan     1514 Mar  2 13:02 graph_data.r
drwxrwxr-x 3 alan alan     4096 Mar  2 13:02 literatura
drwxrwxr-x 7 alan alan     4096 Mar  2 13:07 MSdist
-rw-rw-r-- 1 alan alan    10450 Mar  2 13:02 tree_space.pl
-rwxr--r-x 1 alan alan    51504 Mar  2 13:44 ybyra_sa.py
```

i. *_enumeration.tre (linhas 2 a 6): Estes arquivos contém a enumeração das topologias para N táxons, variando de 6 a 10.

ii. graph_data.r (linhas 7 e 8): São as rotinas em **R** para a produção dos gráficos gerados pelo *script* `tree_space.pl`.

iii. Literatura: Diretório que contém alguns artigos associados com o tema desse tutorial.

iv. MSdist: Diretório que contém o aplicativo MSdist (v. 0.5, documentação em [3]) que calcula distância entre topologias utilizando *Matching Cluster (MC) distance* (veja [4]).

v. tree_space.pl: O *script* tree_space.pl gera matrizes e topologias, faz buscas em TNT, compila resultados e alimenta as rotinas de **R** para gerar os gráficos. Nos exercícios a seguir iremos usar as rotinas implementadas neste *script* para explorar algumas propriedades relacionadas com o espaço de topologias.

3.3 Contextualização teórica

3.3.1 Topologias binárias e grafos

As discussões levantadas por Mindell [5] e Morrison [6] com relação ao modelo de representação gráfica geralmente adotado em estudos filogenéticos é muito interessante. Os argumentos expressos por Morrison [6], principalmente, sugere que devemos refletir se o modelo que estamos adotando é apropriado para o estudo que estamos interessados. De qualquer forma, atualmente, topologias binárias são prevalentes em estudos filogenéticos e embora isso possa mudar no futuro, devemos explorar algumas de suas propriedades – mesmo porque o curso fará uso exclusivo desses diagramas em inferência filogenética.

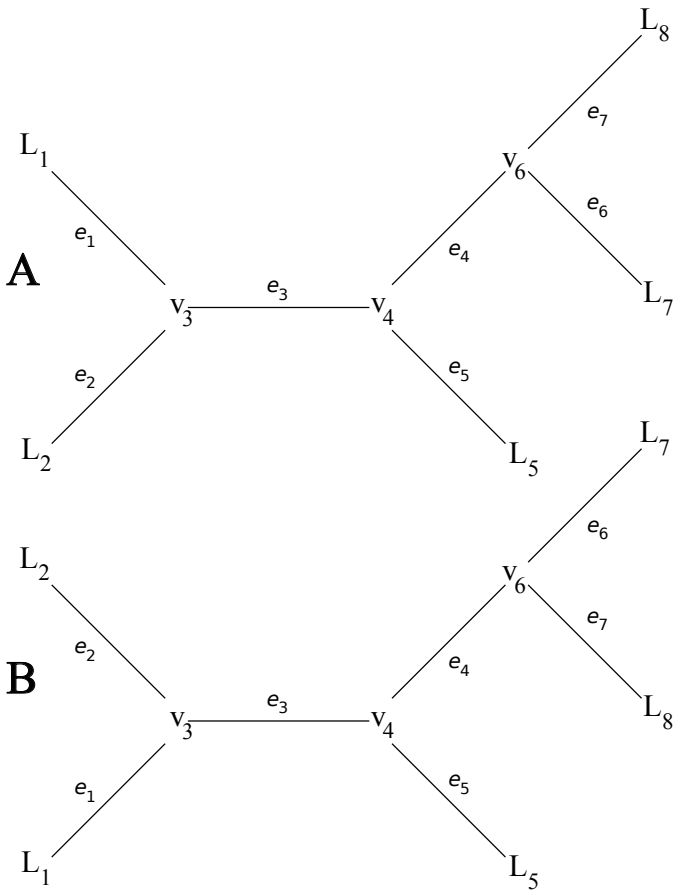


Figura 3.1: Dois grafos (**A** e **B**) com a mesma topologia expressa pelas relações entre vértices (*i.e.*, **V**), terminais (*i.e.*, **L**) e arestas (*i.e.*, **e**).

Grafos são objetos matemáticos que consistem de um par de conjuntos (*V/L*,*E*) de vértices (nós,*V/L*) e arestas (ramos, *E*; veja Figura 3.1). O grau de um vértice é o número de arestas conectados a ele. Desta forma, uma topologia binária $T = (V/L,E)$ é um grafo conectado sem ciclos em que os terminais (L) são vértices de grau 1 ao passo que os nós são vértices de grau 3.

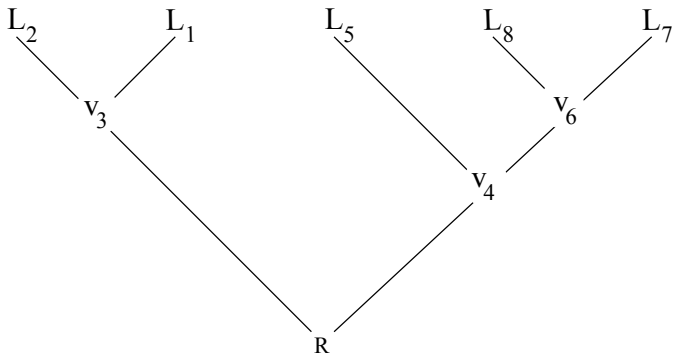


Figura 3.2: Grafo acíclico direcionado. Direcionamento é dado pela raiz (**R**), único vértice de graus 2.

O enraizamento de uma topologia binária (Figura 3.2) se da pela inserção de um vértice de grau 2 (*i.e.*, **R**). Consequentemente, diagramas enraizados possuem um vértice e uma aresta adicional. Diagramas binários não-enraizados possuem ainda algumas propriedades matemáticas que devem ser conhecidas. A primeira delas é que o número de vértices internos é igual a $|L| - 2$. A segunda

é que o número de arestas (*i.e.*, ramos) é igual a $2 * |L| - 3$. Finalmente, o número de topologias possíveis é uma função de $|L|$, ou seja, número de terminais. Portanto, o conjunto de topologias pode ser enumerado.

3.3.2 ENUMERAÇÃO

Em matemática e na ciência da computação, enumeração é a lista de todos os elementos de um conjunto. Em cladística, o termo é aplicado em buscas exatas no qual todas as topologias possíveis são avaliadas (ver Tutorial 4). A enumeração de topologias binárias não-enraizadas obedecem a seguinte fórmula:

$$\text{para } n \geq 3\text{: } \frac{(2n-4)!}{(n-2)!2^{n-2}}$$

O número de topologias enraizadas pode ser calculado multiplicando essa a fórmula acima pelo o número de ramos $(2n - 3)$ ou incrementado n por 1 (veja Tabela 3.1).

Tabela 3.1: Número de topologias binárias em função do número de terminais (*i.e.*, n).

n	**não-enraizada**	**enraizada**
3	1	3
4	3	15
5	15	105
6	105	945
7	945	10395
8	10395	135135
9	135135	2027025
10	2027025	34459425
11	34459425	654729075
12	654729075	13749310575
13	13749310575	316234143225
14	316234143225	7905853580625
15	7905853580625	213458046676875

Como pode ser observado, o número de topologias incrementa exponencialmente à medida que adicionamos terminais. Neste tutorial, o número de diagramas binários é um dos parâmetros que define o espaço de topologias (*i.e.*, *tree space*). Observe que até este momento, estas topologias são simplesmente objetos matemáticos, cujas características são relevantes para análises filogenéticas. A transição entre estes objetos e o conceito de teste de hipóteses requer análise de mérito relativo destes objetos. O mérito relativo depende de como esses modelos explicam nossas observações. Desta, a transição entre objetos meramente matemáticos ("*tree-shaped-objects*", senso [7]) e sua qualidade relativa para explicar (*i.e.*, distribuição de caracteres)

irá depender da implementação de critérios de otimalidade – o que será explicado no próximo tutorial.

3.4 Explorando o espaço de topologias

A seguir, vamos explorar alguns conceitos relacionados com o espaço de topologias. Iremos fazer uma série de execuções do *script* `tree_space.py` e é importante que ao executá-las, você tome nota dos resultados (uma opções é salvas as figuras geradas), pois você deverá compará-los à medida em que vai respondendo os exercícios. Este *script* possui três rotinas básicas que serão explicadas no decorrer deste tutorial (Figura 3.3).

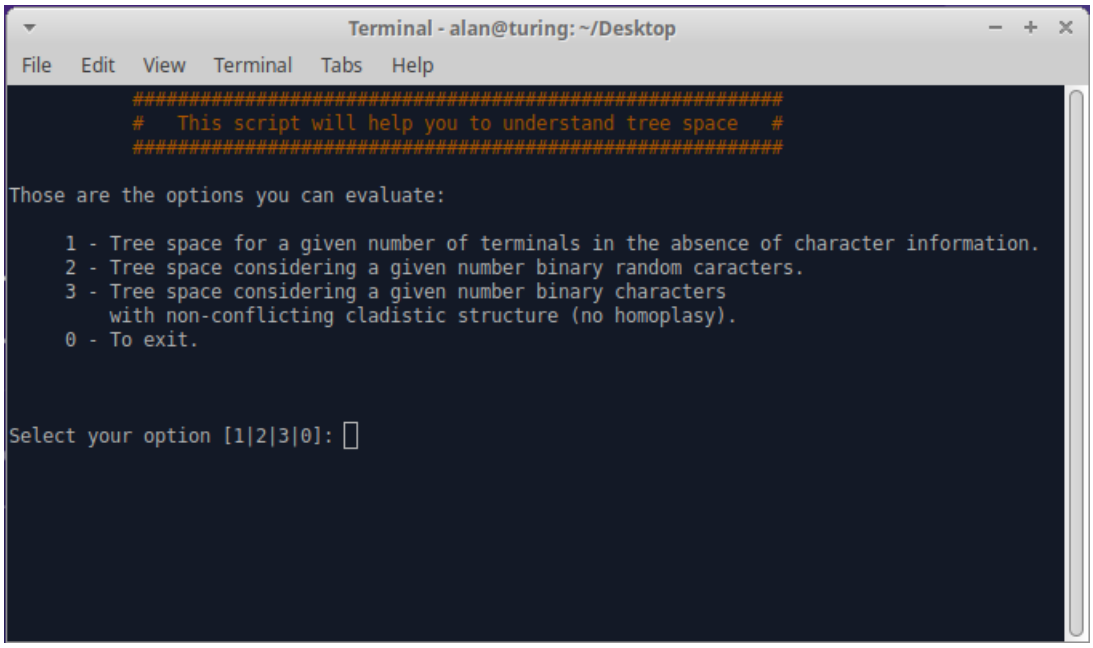


Figura 3.3: Opções de `tree_space.py`. Veja texto abaixo para a descrição de cada uma dessas opções.

3.4.1 Opção 1

Iremos iniciar avaliando como o número de terminais afeta o espaço de topologias na ausência de informação provinda de caracteres cladísticos. Na opção 1, este *script* avalia todas as topologias possíveis (*i.e.*, grafos binários) para determinado número de terminais $\geq 4 \leq 10$[1]. Se você selecionar a opção 1 e pedir para o *script* avaliar o espaço de topologias para 5 terminais você deverá obter dois gráficos representados da Figura 3.4.

[1] não exceda o limite superior desta rotina, pois o resultado demorará muito.

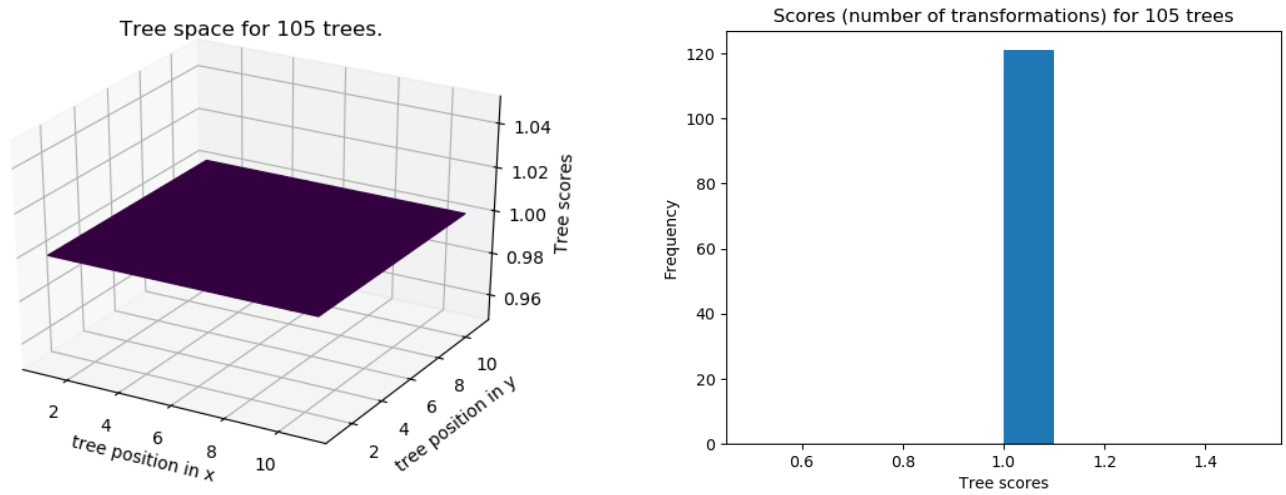


Figura 3.4: Gráficos associados à opção 1 considerando 6 terminais sem informação cladística. À direita você encontra o custo de todas as topologias e à direita a frequência de distribuição destes custos.

Nesta figura, à direita você encontra um gráfico no qual o eixos x e y representam as coordenadas de todas as topologias geradas e no eixo z seus respectivos custos (*Scores*)[2]. Ao lado direito desta figura, há um histograma mostrando a distribuição dos custos das topologias encontradas.

Exercicio 3.1

Considere estes gráficos como uma representação aproximada da complexidade do espaço de topologias. Faça algumas execuções deste *script* variando o número de terminais e responda:

i. Há um único parâmetro deste espaço que é afetado pelo número de terminais. Qual seria esse parâmetro e de que forma o espaço de topologias está variando?

Como você pode observar, os custos de todas as topologias são idênticos. No entanto, considere a Figura 3.5.

[2]O "custo" refere-se ao número de transformações de estado de caráter para cada topologia.

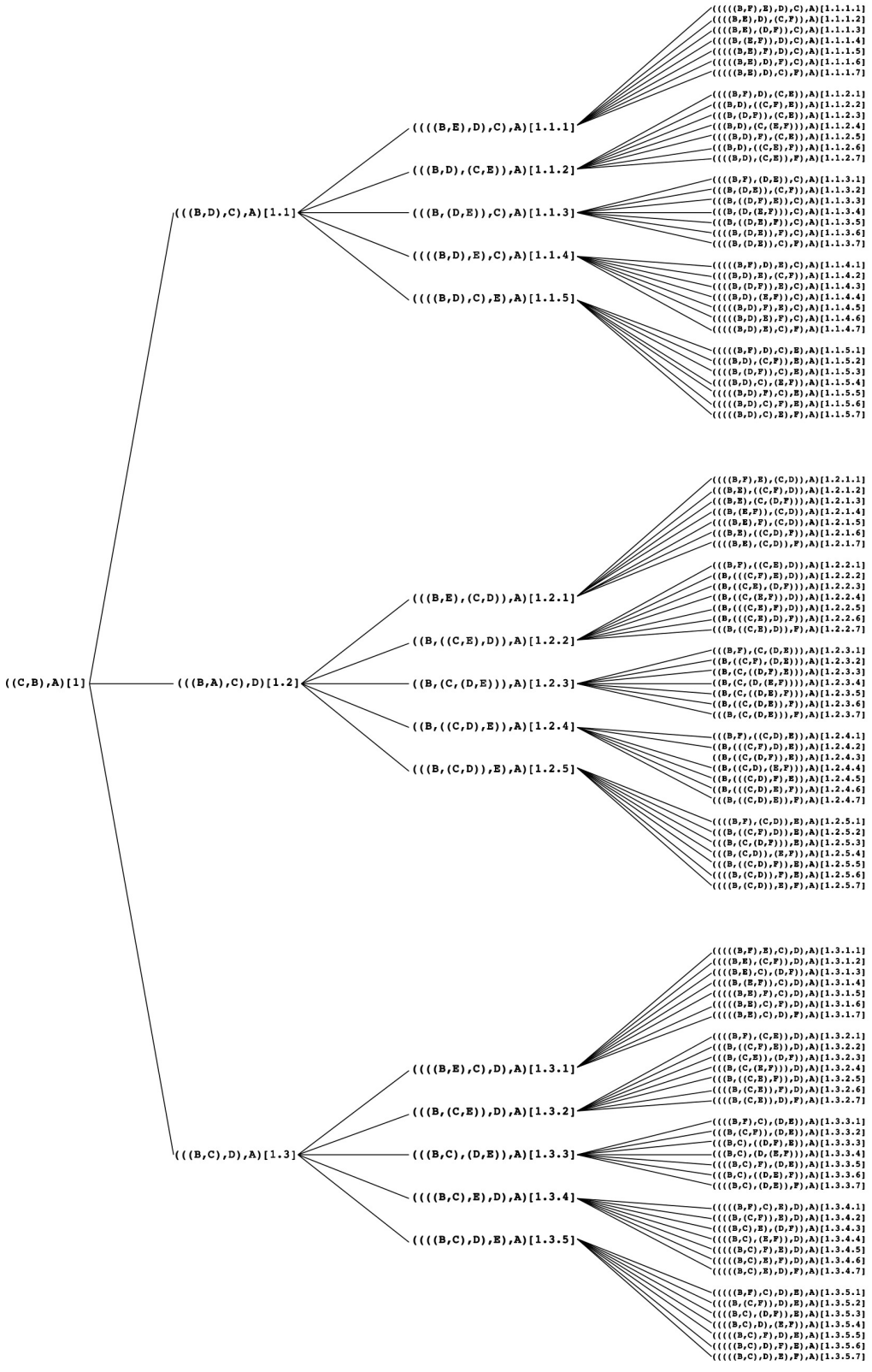


Figura 3.5: Enumeração das topologias para 6 terminais indexadas segundo o arquivo `06_enumeration.tre`. Observe a lógica de indexação desta topologia pelos dígitos entre colchetes (*i.e.*, `[1.3.5.1]`).

Observe que essas tolologias divergem à medida em que se afastam da topologia inicial (*i.e.*, `[1]`). É evidente que nosso primeiro exercício não detecta este componente do espaço de árvores. No entanto, isso seria possível avaliando a distância topológica de acordo com alguma outra métrica.

Robinson & Foulds [8] foram uns dos primeiros a propor uma métrica que avaliasse diferença entre árvores filogenéticas. Os detalhes do cálculo de distância são irrelevantes para os propósitos deste turorial, mas o leitor mais curioso deve consultar o artigo de Robinson & Foulds [8] e Bogdanowicz & Giaro [4] para maiores detalhes (no diretório `literature` anexo a este tutorial). Via de regra, métricas que computam distâncias entre topologicas consideram os passos necessários, de acordo com algum critério, para tranformar uma topologia em outra. Neste tutorial iremos avaliar distâncias entre topologias utilizando o YBYRÁ, que usa a métrica proposta por Robinson & Foulds [8].

O uso do UBYRÁ para este propósito é relativamente simple. Suponha, por exemplo, que eu queira avaliar a distância entre a topologia `[1.1.1.1]` e as demais topologias enumeradas a partir de `[1.1.1]`. Os passos a seguir seriam:

i. Extrair as topologias desta família (*i.e.*, `[1.1.1]` do arquivo `06_enumeration.tre`. A forma mais simples de fazer isso seria utilizar a seguinte linha de comando:
`$ egrep '\[1\.1\.1\..*' 06_enumeration.tre > 1.1.1.n.tre`

ii. Extrair a topologia `1.1.1.1.tre` de `1.1.1.n.tre` com a seguinte linha de comando:
`$ head -n 1 1.1.1.n.tre > 1.1.1.1.tre`

iii. Criar o arquivo de configuração necessário para executar o YBYRÁ. Para computar as distâncias entre a topologia `1.1.1.1.tre` e as demais topologias em `1.1.1.n.tre`, o arquivo de configuração (*e.g.*, `conf.txt`) deve ter o seguinte conteúdo:

```
>id = CalcDist_6a
<begin files
        1.1.1.1.tre;
        1.1.1.n.tre;
end files>
>n = 1 [1.1.1.1.tre]
>opt = 3
>compare = 1
>root=A
>verbose
```

Neste arquivo de configuração, a linha 1 define a ID (identidade) da análise. As linhas 2 a 5 definem os arquivos que contém as topologias a serem comparadas. A linha 6 define a topologia de referência. As linhas 7 e 8 configuram o tipo de comparação que o programa irá executar (veja documentação do programa para maiores detalhes). A linha 9 informa o táxon de enraizamento – necessário para computar as distâncias. Finalmente, a linha 10 configura o programa para informar ao usuário as etapas que estão sendo executadas à medida em que o programa prossegue.

iv. Executar o YBYRÁ da seguinte forma:

```
$ python ybyra_sa.py -f conf.txt
```

Após a execução do programa, você deverá obter, dentre várias linhas de saida, a seguinte informação:

```
The file MATRIX_calcDist_6a.txt was created.
```

Este arquivo contém os resultados que deseja e estão sumarizados na Tabela 3.2.

Tabela 3.2: Distâncias topológicas de acordo com Robinson & Foulds [8]. **SC**, Clados compartilhados (*i.e.*, comuns entre as duas topologias). **CC**, Clados combinados (*i.e.*, soma de todos os clados distintos encontrados em ambas topologias). **LD**, Distância local.

Tree No.	**Tree No.**	**SC**	**CC**	**LD =** $1-(\frac{SB}{CB})$
1	2	5	5	0
1	3	2	8	0.75
1	4	3	7	0.57
1	5	4	6	0.33
1	6	4	6	0.33
1	7	3	7	0.57
1	8	2	8	0.75

Exercicio 3.2

Com base no exemplo acima, no qual foram calculadas as distâncias entre topologias responda:

i. Como se comportam as distâncias entre a topologia `1.1.1.1.tre` e aquelas enumeradas para as demais famílias (*i.e.*, `1.1.2` e `1.1.3`)?

ii. O tamanho do espaço de topologias influencia os resultados que você obteve no item anterior? (**OBS.**: considere que há topologias enumeradas para 7, 8, 9 2 10 terminais.)

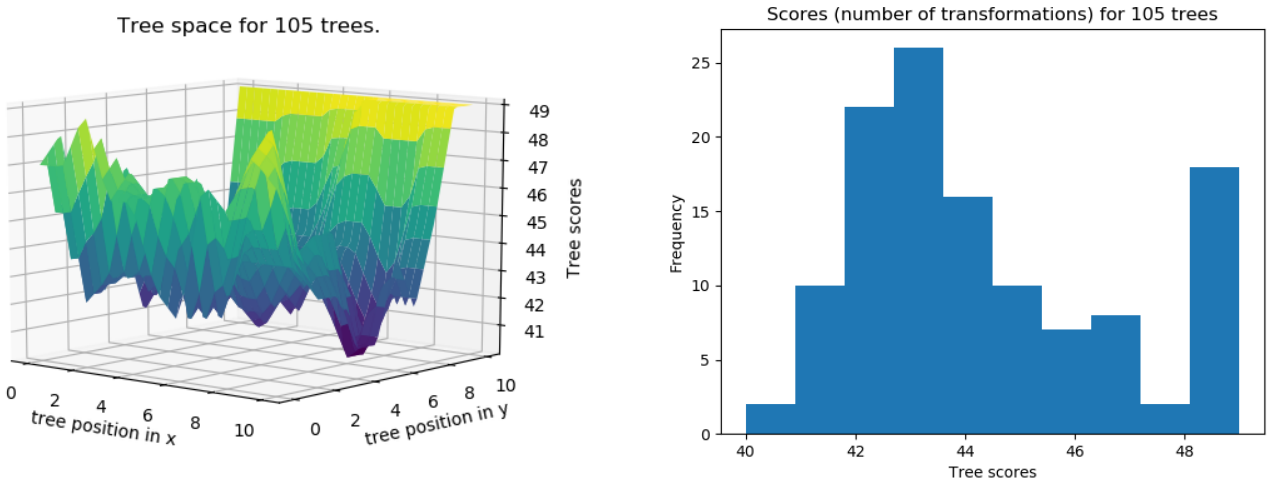


Figura 3.6: Gráficos associados à opção 2 considerando 6 terminais e 24 caracteres aleatórios. À direita você encontra o custo de todas as topologias e à direita a frequência de distribuição destes custos.

3.4.2 Opção 2

A opção 2 faz com que o *script* proceda da seguinte forma (Figura 3.6). Dado o número de terminais e caracteres, `tree_space.py` gera uma matriz aleatória com n caracteres binários cuja probabilidade de 0 ou 1 é igual a 0.50. Esta matriz de dados é submetida ao programa TNT para computar o custo de todas a topologias possíveis dado o número de terminais.

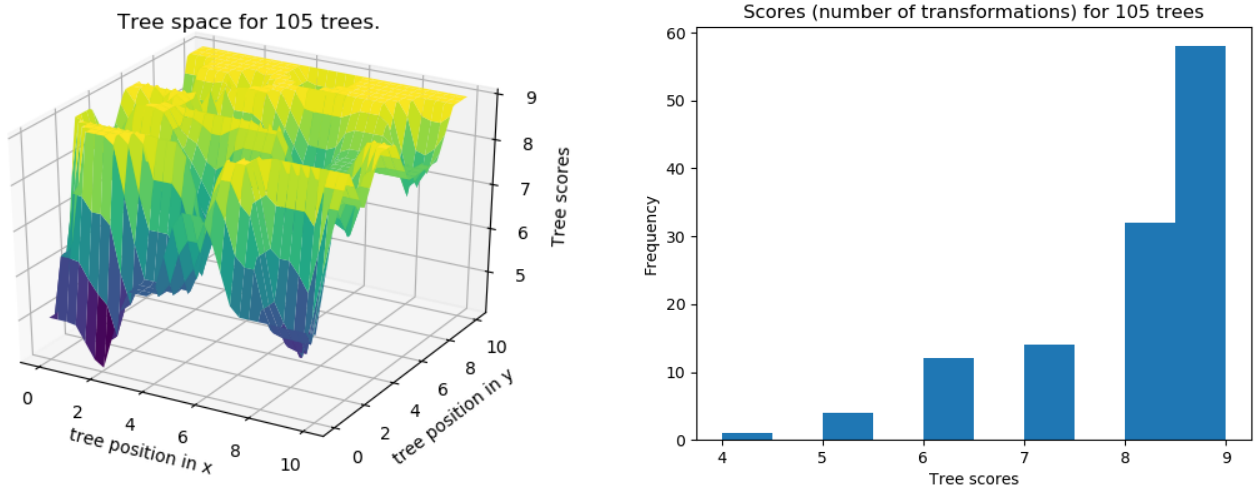


Figura 3.7: Gráficos associados à opção 2 considerando 6 terminais e uma matriz de representação (sem homoplasias). À direita você encontra o custo de todas as topologias e à direita a frequência de distribuição destes custos.

3.4.3 Opção 3

Esta rotina difere da anterior em alguns pontos importantes (Figura 3.7). Ao definir o número de terminais, `tree_space.py` gera uma topologia aleatória e uma matriz de representação. Por

esta razão, não há como definir o número de caracteres, pois estes são dependentes dos números de nós da topologia binária para os quais a matriz de representação dará suporte. Posteriormente, TNT avalia o custo desta matriz de representação em relação ao universo de enumeração de todas as topologias binárias.

Exercício 3.3

Você deverá fazer uma série de execuções das opções 2 e 3. Compare os resultados salvando as figuras se considerar necessário. Com base nos seus resultados, responda:

i. Como se comporta o espaço de topologias em ambas as rotinas?

ii. O comportamento dos valores de custo mínimo direfem entre as duas rotinas?

iii. Qual é a observação mais relevante que você observa com relação aos histogramas produzidos por essas duas rotinas?

iv. Como você obteve algum resultado da opção 2 que resultou em uma única topologia com o menor custo? Você acha isso relevânte para inferência filogenética?

Exercicio 3.4

Abaixo, defina o que você entende por espaço de topologia e quais são o elementos que afetam sua conformação:

3.5 Referências

1. Goloboff, P.; Farris, J. & Nixon, K. 2008. TNT, a free program for phylogenetic analysis. *Cladistics* **24**(5): 774–786.
2. Machado, D. J. 2015. YBYRÁ, a fast and resourceful tool for sensitivity analysis. *BMC Bioinformatics Journal* **16**: 204.
3. Bogdanowicz, D. MSdist 0.5 – an application for computing the Matching Split distance between phylogenetic trees. `http://www.kaims.pl/~dambo/mcdist/`. Version 0.5. 2010.
4. Bogdanowicz, D. & Giaro, K. 2013. On a matching distance between rooted phylogenetic trees. *International Journal of Applied Mathematics and Computer Science* **23**: 669–684.

5. Mindell, D. P. 2013. The tree of life: metaphor, model, and heuristic device. *Systematic Biology* **62**(3): 479–489.

6. Morrison, D. A. 2014. Is the tree of life the best metaphor, model, or heuristic for phylogenetics? *Systematic Biology* **63**(4): 628–638.

7. Wheeler, W. C. 2012. Systematics: a course of lectures. Malaysia: Wiley-Backwell, 2012. 426.

8. Robinson, D. F. & Foulds, L. R. 1981. Comparison of phylogenetic trees. *Mathematical Biociences* **53**: 131–147.

Tutorial 4

Introdução ao TNT

BIZ0433 - INFERÊNCIA FILOGENÉTICA: FILOSOFIA, MÉTODO E APLICAÇÕES.

Conteúdo

Objetivo

O objetivo deste tutorial é introduzir conceitos básicos associados ao uso do TNT. Este aplicativo é um dos mais versáteis e rápidos disponíveis para a busca de árvores filogenéticas utilizando parcimônia como critério de otimalidade. Neste tutorial iremos explorar a sintaxe dos arquivos de entrada deste programa para matrizes e topologias, verificar como TNT salva as topologias encontradas e entender como o TNT executa buscas exaustivas e heurísticas. 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/
```

4.1 Considerações gerais

O TNT [1] é um programa para inferência filogenética que utiliza parcimônia como critério de otimalidade na buscas de topologias com menor custo. O programa permite a análise de dados morfológicos discretos e contínuos, bem como de dados genotípicos concatenados ou em partições distintas. Há duas distribuições básicas de TNT. Uma para o sistema operacional Windows, que possui interface gráfica, e outra para Linux/Mac OS X, que deve ser executada por comandos de linha. Neste tutorial, iremos utilizar a versão sem interface gráfica, pois ela é mais versátil, principalmente quando você deseja fazer uma série de análises cuja implementação pode ser feita por meio de *scripts*. Esse tutorial assume que você possui o TNT instalado em seu computador. Caso seja necessário instalá-lo, baixe a versão compatível com seu sistema operacional na página http://www.lillo.org.ar/phylogeny/tnt/.

4.2 Estrutura dos arquivos de entrada

A estrutura mais simples de um arquivo de entrada para TNT obedece a seguinte formatação:

```
xread
'qualquer comentário'
No._de_caracteres No._de_taxóns
taxon_1 dados
taxon_2 dados
...
taxon_n dados
;
```

Para dados morfológicos discretos essa matriz seria:

```
xread
'dados morfologicos'
5 4
taxon_1 00010
taxon_2 10000
taxon_3 11001
taxon_4 11100
;
```

Para dados moleculares usaríamos:

```
nstates dna;
xread
'formato para dados moleculares'
8 6
taxon_1 ACGTACGT
```

```
taxon_2 AAGTACGT
taxon_3 AAATACGT
taxon_4 AAAAACGT
taxon_5 AAAAAAGT
taxon_6 AAAAAAAT
;
```

Note que, para dados moleculares, a instrução dada ao TNT é expressa na primeira linha do arquivo de entrada – onde se lê "`nstates dna;`".

Você pode ainda combinar dados genotípicos e fenotípicos utilizando uma estrutura como esta:

```
nstates 32
xread
14 5
& [dna]
A ACCCCTGT
B ACCCCTGT
C NCCCCTGT
D ACCCCTGA
E ACCCCTGA
& [prot]
A KKKQ
B KKKQ
C KKKQ
D KKKI
E KKKI
& [num]
A 00
B 11
C 12
D 12
E 13
;
```

Neste último caso, o termo "`nstates 32`" define o número máximo de estados de caráter que o TNT deverá considerar, e os termos "`& [dna]`", "`& [prot]` e "`& [num]`" definem os conjuntos de dados para nucleotídeos, proteínas e numéricos, respectivamente.

Seguindo essas estruturas, você poderá usar qualquer editor de texto simples (i.e., word pad, gedit e textedit, entre outros) para escrever qualquer matriz a ser usada pelo TNT. Independentemente do editor escolhido, é importante que o arquivo seja salvo em formato texto, caso contrário o

editor irá inserir caracteres ocultos que impedirão o TNT de ler o documento. Há editores de matrizes, tais como Mesquite – que funciona em todas as plataformas e NEXUS – exclusivamente para WINDOWS. Para o propósito de entender como o TNT funciona não será necessário utilizar esses aplicativos, mas é bom que vocês saibam que eles existem, pois os mesmos podem ser úteis para desenvolver projetos mais complexos para os quais você deseja anotar dados adicionais relacionados às definições de caracteres e estados de caráter.

Exercicio 4.1

Em um teminal, você devera examinar o conteúdo arquivo "`matriz_1.tnt`" do diretório "`tutorial_4`" utilizando comandos de linha ou editores de sua escolha. Correlacione o conteúdo deste arquivo com as estruturas apresentadas acima. A primeira linha deste arquivo ("`xread`") informa ao TNT que ele deve ler uma matriz de dados. Na segunda linha, há um comentário (*i.e.*, `'essa eh sua primeira matrix'`) que será impresso no terminal quando o TNT ler a matrix. A terceira linha ("`1 6`) informa ao programa que essa matriz possui uma coluna de caracteres e seis linhas de terminais. Posteriormente, você encontra a matriz de dados propriamente dita, e finalmente um "`;`" que indica ao TNT que a matriz terminou.

i. Você deverá criar uma arquivo chamado "`exemplo_1.tnt`" que contenha os dados abaixo e no formato que possa ser lido pelo TNT.

ii. Ao lado, você deverá desenhar a topologia mais parcimoniosa para esta matriz.

```
taxon_1 00000
taxon_2 10000
taxon_3 11000
taxon_4 11100
taxon_5 11110
taxon_6 11101
```

Antes de seguirmos à diante, veremos se o TNT está lendo o arquivo "`exemplo_1.tnt`" – ou seja, verificaremos se o arquivo não possuí nenhum erro se sintaxe. A primeira coisa de devemos fazer é executar o TNT. No *prompt* do seu terminal execute:

```
$  tnt
```

Você deferá obter o *prompt* de iniciação do TNT ilustrado na Figura 4.1:

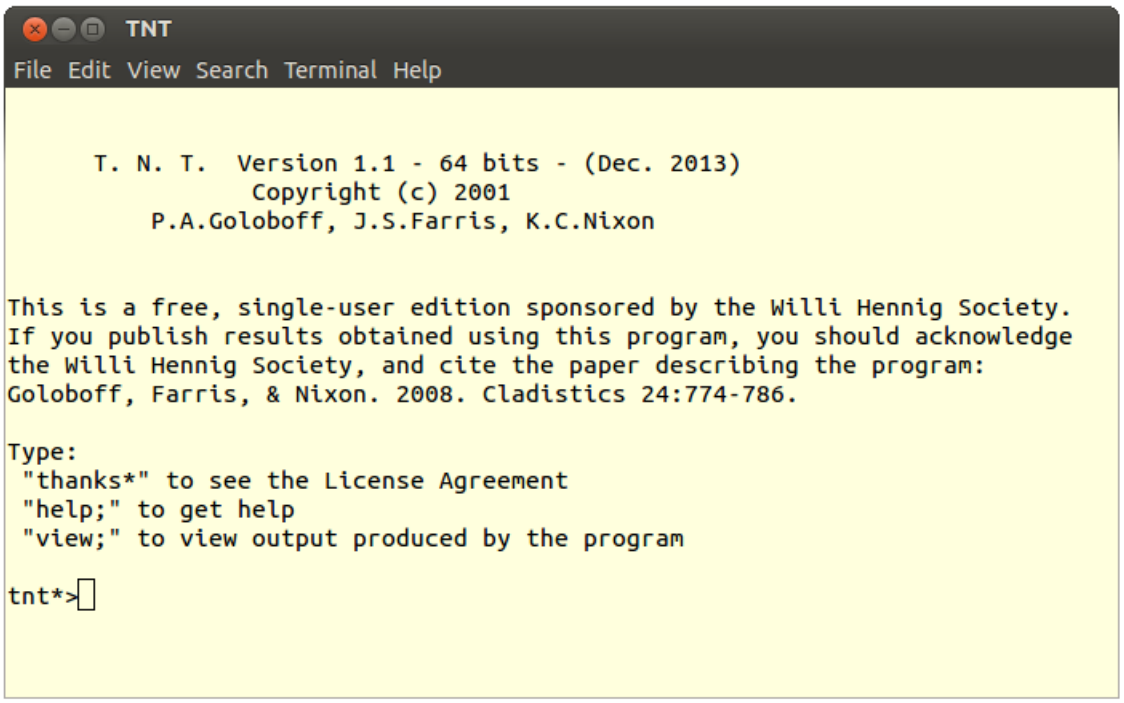


Figura 4.1: *Prompt* de iniciação do TNT.

O comando de leitura de arquivos no TNT é "`proc`", de *procedure*. Todos os comandos em TNT devem ser seguidos de "`;`". Se você digitar "`help proc;`" você obterá:

```
tnt*>help proc;

PROCEDURE
Redirect input
XXX;    take commands from file XXX
/;      close input file (include at end of file)
...
```

portanto, para que o TNT leia o arquivo "`exemplo_1.tnt`", basta você executar o seguinte comando:

```
tnt*> proc exemplo_1.tnt;
```

o TNT deverá retornar:

```
Reading from exemplo_1.tnt
Matrix (5x6, 16 states).  Memory required for data:   0.02
Mbytes
```

Caso você não tenha obtido o resultado acima, você deverá tentar identificar o erro e tentar fazer com que o TNT leia seu arquivo de entrada novamente.

4.3 Topologias em TNT

Obter topologias em programas de inferência filogenética é relativamente fácil; basta saber a sintaxe do(s) arquivo(s) de entrada e alguns comandos de execução. No entanto, entender como esses aplicativos tratam seus dados, quais são os tipos de análises disponíveis e como proceder para ter certeza de que você fez o que queria e explorou seus dados de maneira adequada requer um pouco mais do que simplesmente executar alguns comandos. Isso vale para comandos de linha e/ou análises em que você seleciona opções nos menus do programa.

Veja como é fácil obter uma topologia a partir do momento em que o TNT leu seu arquivo de entrada. Após ler o arquivo "`exemplo_1.tnt`", se você executar o comando:

```
tnt*> ie;
```

Você deverá obter:

```
Implicit enumeration, 1 trees found, score 5.
```

A mensagem de saída de TNT descreve o algorítmo utlizado na busca (*i.e.*, "Implicit enumeration"), número de topologia(s) encontrada(s) (*i.e.*, "1 trees found") e o custo – número de transformações – desta(s) topologia(s) (*i.e.*, "score 5"). Para visualizar a topologia encontrada pelo TNT basta executar o seguinte comando:

```
tnt*> tplot;
```

Observe que o TNT utiliza o primeiro terminal de sua matriz como raiz caso nenhum outro seja especificado. Verifique o resultado impresso pelo TNT no terminal e responda:

Exercicio 4.2

A topologia apresentada é a mesma que você obteve no exercício 4.1? No que elas diferem?

Vamos repetir a busca novamente, mas antes disso iremos executar o comando "`collapse [;`". Após executá-lo, faça uma nova busca usando "`ie;`" e imprima a topologia no terminal. Essa nova topologia deve ser a mesma que você obteve manualmente. Por que isso acontece? Por *default* o TNT irá sempre lhe fornecer **topologias binárias**, *i.e.*, totalmente resolvidas ou

dicotômicas, mesmo quando não existem caracteres sustentando um determinado nó. Portanto, **muito cuidado**! Muitas pessoas ignoram este componente da configuração do TNT e registram resultados irreais. Uma forma de evitar isso é sempre inserir o termo "`collapse [;`" no próprio arquivo de TNT, como no exemplo abaixo:

```
xread
5 6
taxon_1 00000
taxon_2 10000
taxon_3 11000
taxon_4 11100
taxon_5 11110
taxon_6 11101
;
collapse [;
proc/;
```

4.4 Obtendo ajuda no TNT

O TNT possui uma ferramenta de ajuda interna que pode ser evocada pelo comando "`help`". Esta função de ajuda é de certa forma críptica, principalmente se você não está familiarizado com os comandos de TNT. Vamos ver como o comando funciona.

No *prompt* do TNT digite:

```
tnt*> help;
```

Você deverá obter o resultado ilustrado na Figura 4.2 abaixo.

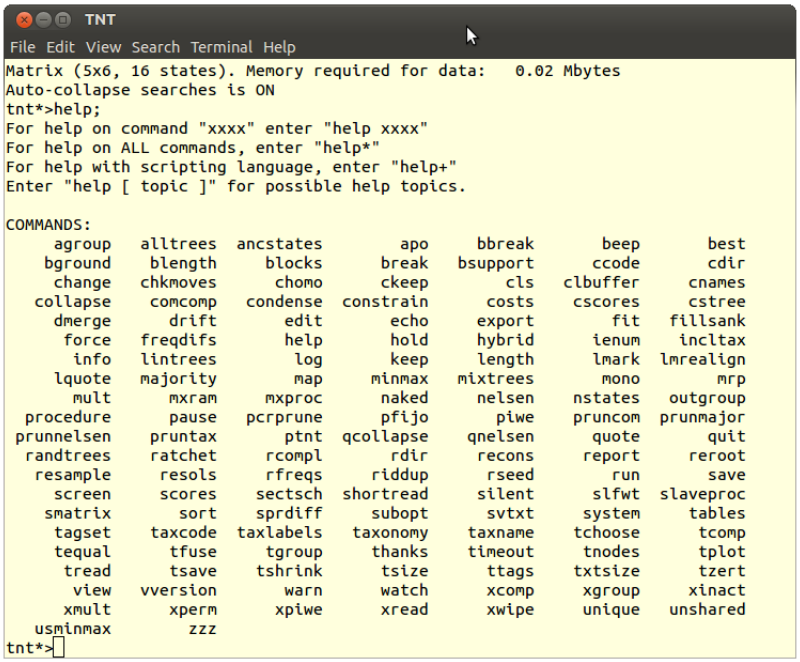


Figura 4.2: Lista de comandos disponíveis na documentação interna de TNT.

Os detalhes desses comandos podem ser vistos executando "`help nome_do_comando`", como por exemplo "`help collapse`", "`help proc`" e "`help zzz`". O primeiro comando regula as regras de colapso de ramos (que discutiremos em momento oportuno), o segundo controla o direcionamento da entrada de comandos em TNT e o terceiro termina a seção de TNT. Execute esse último comando!

4.5 Leitura de topologias em TNT

Da mesma forma que o TNT lê matrizes de dados, o programa também permite ler e salvar tolopogias. Os arquivos de topologia também obecedem uma lógica, muito similar àquela utilizada nos arquivos que contém dados. Veja o exemplo abaixo:

```
tread
'topologias para o exemplo_1.tnt'
(taxon_1 (taxon_3 (taxon_2 (taxon_4 (taxon_5 taxon_6)))))*
(taxon_1(taxon_5(taxon_3(taxon_2(taxon_4 taxon_6)))))*
(taxon_1 (taxon_2 (taxon_3 (taxon_6 (taxon_4 taxon_5)))));
```

A primeira linha deste arquivo deve conter o termo "`tread`", de *tree read*, indicando ao TNT que ele deverá ler uma ou mais topologias. Na próxima linha você pode ou não inserir um comentário (*e.g.*, "`'topologias para o exemplo_1.tnt'`"). A seguir você deve inserir a(s) topologia(s) em notação parentética obedecendo as seguinte regras:

a. Topologias que não sejam a última são seguidas de "`*`".

b. A topologia final é seguida de "`;`".

c. Terminais confinados a um conjunto de parênteses (*i.e.*, "`(taxon_4 taxon_6)`") devem estar separados por um espaço.

Exercicio 4.3

Neste exercício você deverá ler uma matriz e um conjunto de topologias e verificar o custo destas topologias.

a. Inicie o TNT;

b. Leia a matriz a "`exemplo_1.tnt`" que você ditou no exercicio anterior;

c. Leia o arquivo "`exemplo_1.tre`" da mesma forma que você leu a matriz de dados; Observe que o TNT leu três topologias as quais foram atribuídas números de 0 a 2. Essa é outra peculiaridade de TNT, toda contagem se inicia em **0 (ZERO)** neste programa, **lembrem-se sempre desta particularidade do TNT!**

d. Consulte o *help* para o comando "`scores`" e execute-o.

Ao final desse exercício você deverá ter obtido:

```
    0  1  2
0   6  7  5
```

A primeira linha e a primeira coluna desta tabela impressa pelo TNT servem de referência para identificar a qual topologia o custo se refere. Por exemplo, a segunda linha e segunda coluna, onde se lê `6`, refere-se a topologia `00`; a segunda linha e terceira coluna, onde se lê `7`, refere-se a topologia `01`; finalmente, a segunda linha e quarta coluna, onde se lê `5`, refere-se a topologia `02`.

Exercicio 4.4

Use o comando "*help*" do TNT para o comando de linha "`tchoose`". Com base no conhecimento sobre o TNT que você obteve até este momento, você deverá manter apenas a topologia mais curta na memória do TNT e posteriormente imprimí-la na tela de seu computador. Quais foram os comandos que você executou para cumprir estas tarefas?

_____

_____

Exercicio 4.5

Neste exercício, você deverá editar manualmente o arquivo "`exemplo_1.tre`". Você deverá criar uma topologia qualquer, diferente das demais, na última linha do arquivo – ou seja, topologia "`3`". Posteriormente, você deverá salvar o arquivo editado sob o nome de "`exercicio_1b.tre`". Abaixo, ilustre todas as topologias neste arquivo, bem como seus respectivos custos e indique os comandos utilizados para completar o exercício.

4.6 Como salvar topologias em TNT

O TNT salva topologias em 3 formatos. No primeiro deles, *default* de TNT, as topologias são salvas em formato binário. Arquivos binários são aqueles que só podem ser interpretados por programas e/ou processadores. Desta forma, embora este formato seja apropriado para gerar arquivos pequenos com muito conteúdo – topologias – ele só será lido em TNT. Os demais formatos usados pelo TNT para salvar topologias geram arquivos textos. Estes tem a vantagem de poderem ser lidos por outros programas – algumas vezes com pequenas modificações – e/ou examinados em editores de texto. Isso é muito vantajoso, embora o custo evidente é que os arquivos são maiores que aqueles gerados no formato binário. A seguir iremos ver as diferenças entre esses dois formatos.

Exercicio 4.6

Neste exercício, iremos explorar duas formas de salvar topologias em TNT e verificar no que eles diferem.

i. Abra o arquivo `exemplo_2.tnt` e execute uma busca por *implicit enumeration*. Você deverá obter 2 topologias com o custo de 6 passos.

ii. Execute os seguintes comandos: "`tsave* exercicio_2a.tre;`", "`save;`", e "`tsave/;`"

iii. Execute os seguintes comandos: "`taxname =;`", "`tsave* exercicio_2b.tre;`", "`save;`", e "`tsave/;`"

iv. Consulte o "*help* do TNT e anote a função de cada comando utilizado nas linhas abaixo:

Comandos:

`taxname =`: _____

`tsave*`: _____

`save`: _____

`tsave/`: _____

v. Verifique os conteúdos dos arquivos `exercicio_2a.tre` e `exercicio_2b.tre` e responda: Qual é a diferença entre eles?

_____

_____

4.7 Busca de topologias em TNT

Algoritmos de busca podem ser exatos ou heurísticos. Algorítmos exatos, sejam eles por enumeração implícita ou explícita [2, 3], avaliam todo o espaço de solução – no nosso caso o espaço de topologias –, e garantem que a solução ótima para a sua matriz (*i.e.*, menor custo) foi encontrada. Algoritmos heurísticos, não lhe fornecem esta garantia, pois examinam uma amostra do seu espaço de topologias. Desta forma, poderíamos pensar de imediato que sempre deveríamos escolher algoritmos exatos. No entanto, como vocês verão, há limitações quanto ao uso desses algoritmos em inferência filogenética.

TNT possui um algoritmo de busca exata por enumeração implícita. Enumeração implícita significa que "implicitamente" o algoritmo examinou todas as topologias que potenciamente poderiam ter um custo menor do que uma aproximação inicial – como discutimos na aula teórica. Vejamos algumas propriedades destes algoritmos.

Exercício 4.7

Considere os arquivos da Tabela 4.1. Para cada um deles, execute uma busca com o algoritmo "`ie`", registre o tempo de execução da análise na Tabela 4.1 e responda:

a. Quais são as diferenças entre estas matrizes de dados?

b. O que acontece com o tempo de execução da análise em relação às diferenças observadas nestes arquivos?

c. O que impede o uso deste algorítmo em análises filogenéticas?

Tabela 4.1: Templo de execução em *implicit enumeration*.

Matriz de dados	Tempo
`10x500.tnt`	
`10x1000.tnt`	
`10x1500.tnt`	
`15x500.tnt`	

Deve ter ficado claro que o número de terminais limita o uso de buscas exatas. A busca por

topologias ótimas não é tarefa trivial, principalmente pelo fato de que o número de topologias possíveis cresce exponencialmente à medida em que adicionamos terminais – como vimos anteriormente (veja Tutorial 3). Mesmo algoritmos que utilizam enumeração implícita, no qual grande parte do universo de topologias possíveis é descartada, o tempo computacional pode ser muito alto. Desta forma, métodos heurísticos são frequentemente utilizados para esse propósito.

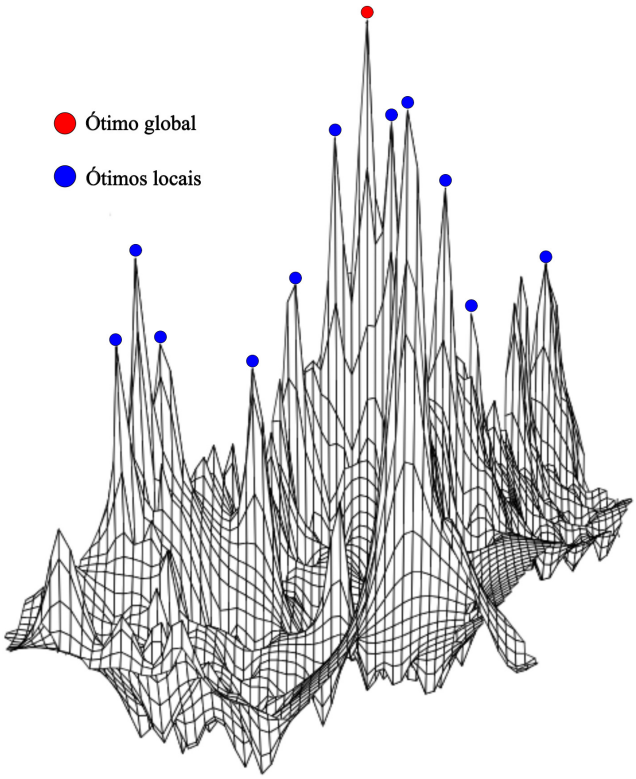


Figura 4.3: Ótimos locais e globais representados em uma superfície de relevo.

O desafio de métodos heurísticos é explorar o espaço de topologias o suficiente para evitar ficar restrito a ótimos locais (Figura 4.3). A analogia geralmente utilizada para explicar o conceito de ótimos locais e global é ilustrada no relevo representado na Figura 4.3. Este relevo indica um pico mais alto que os demais (em vermelho) que representaria a solução ótima – *i.e.*, a(s) topologia(s) mais curta(s). O relevo inclui ainda uma série de outros picos menores que representam uma ou mais topologias sub-ótimas. O objetivo destes algoritmos é evitar ficar trancados em ótimos locais – o que depende do quão eficientemente ele explora o espaço de topologias. Vale ressaltar que a complexidade deste relevo depende de vários fatores, tais como o número de soluções possíveis – que determina e espaço e o incremento de distâncias topológicas – e estruturação dos dados (veja Tutorial 3).

Dentro do que é conhecido como busca por trajetória (*trajectory search*), a grade maioria dos algoritmos heurísticos inicia sua busca construindo uma árvore de Wagner [4], em uma etapa conhecida como *random addition sequence* (**RAS**). Após a construção inicial, a árvore de Wagner é submetida a algoritmos de refinamento por um procedimento conhecido como *branch swapping*. Neste contexto, a árvore de Wagner determina um ponto inicial neste espaço e os procedimentos

de refinamento asseguram a melhor solução local. Dentre estes algoritmos de *branch swapping* mais comumente implementados, podemos citar *nearest-neighbor interchange* [**NNI**, 5, 6], *Tree-Bisection and Regrafting* [TBR, 7] ou "*Branch-Breaking*" [8]. Estes algoritmos estão virtualmente implementados em todos os programas de inferência filogenética, incluindo TNT. Detalhes sobre esses algoritmos podem se encontrados em [9–13].

Exercicio 4.8

Neste exercício vamos explorar os algoritmos de buscas por trajetória existentes no TNT. Considere o arquivo `zilla.tnt`. Este arquivo contém 500 terminais, o que torna impraticável o uso de algoritmos exatos. Veremos o comportamento de TNT em relação aos parâmetros de análises de buscas heurísticas simples (*i.e., traditional search*).

i. Reinicie o TNT para nos certificarmos de que estamos usando os parâmetros de *default* do programa.

ii. Leia o arquivo `zilla.tnt` em TNT.

iii. O algoritmo de busca tradicional do TNT inclui **N** adições aleatórias (**RAS**) que controem árvores de Wagner seguidas de refinamento por SPR e/ou TBR. O comando para sua execução é "`mu`" e para saber quais são os parâmetros de *default* deste comando você deve executar o comando "`mu:;`". Execute este último comando e anote os parâmetros abaixo:

```
Settings for multiple random addition sequences:
 * __ random sequences
 * saving up to __ trees per replication
 * swapping trees with __
 * keeping only the best trees found
```

iv. Execute o comando "`mu`" e verifique o resultado impresso no terminal. Você deverá ter obtido um resultado muito semelhante àquele ilustrado na Figura 4.4.

```
PISH (Phylogenetic Inference SHell)

Reading from zilla.tnt
'matrix from Pee-Wee'
Matrix (759x500, 16 states). Memory required for data:   2.39 Mbytes
tnt*>mu;
Repl. Algor.     Tree        Score       Best Score   Time       Rearrangs.
10    TBR       99 of 100    ------      16228        0:00:16    2,173,372,399
Completed 10 random addition sequences.
Total rearrangements examined: 2,173,372,399.
Best score hit 1 times out of 10 (some replications overflowed).
Best score (TBR): 16228.  10 trees retained.
tnt*>
```

Figura 4.4: Resultado de busca em TNT utilizando os parâmetros de *default* do comando `mu`.

O resultado informa que foram feitas 10 réplicas (**RAS**) e que as topologias mais curtas possuíam 16228 passos, que esse custo foi obtido em apenas uma

réplica (*i.e.*, "`Best score hit 1 times`") e que foram retidas 10 topologias na memória (*i.e.*, "`10 trees retained`"). Observe que o TNT informa quantas topologias foram examinadas, o tempo de execução, e que durante a busca, algumas réplicas excederam o número de topologias que poderia ser mantida (*i.e.*, "`some replications overflowed`").

Você consideraria que esses parâmetros configuram uma boa estratégia de busca heurística para esses dados? Saiba que para esta matriz há pelo menos 46 topologias com o custo de 16218 passos! Você terá muita sorte se conseguir atingir esse resultado utilizando esses algoritmos, principalmente com os parâmetros iniciais.

v. Há vários parâmetros que controlam a qualidade de buscas heurísticas, vejamos como controlá-los:

replic: Essa opção é a mais óbvia de todas, pois ela controla o número de adições aleatórias do comando "`mu`" (*ex.* `mu: rep 100`).

hold: Possui duas funções. Como **comando** de TNT ele controla o número máximo de topologias que o TNT irá manter na memória. Seu valor de *default* é 100. Portanto, ao chegar a esse valor o TNT interrompe a busca! Como **opção** do comando "`mu`", "`hold`" controla o número máximo de topologias que será mantida durante cada réplica.

É necessário manter um balanço entre o comando "`hold`" e as opções de "`mu`". Por exemplo, se você modificar o número de réplicas em "`mu`" utilizando o comando 'mu: rep 100, ao executar a busca o TNT irá interrompê-la na décima réplica e retornará a seguinte mensagem: "`NOTE: Search terminated after replication 10 (tree buffer full)`". Isso ocorre porque havia espaço para 100 topologias na memória do TNT e a busca por "`mu`" estava configurada para manter apenas 10 topologias por réplicas. Portanto, 10 x 10 = 100! Fim. Desta forma, o número de topologias atribuídas ao **comando** "`hold`" deve ser maior ou igual ao número atribuído à **opção** "`hold`" de "`mu`" vezes o número de réplicas. Entender os conceitos associados a esse balanço é muito importante.

mxram: Por *default*, o TNT aloca 16 MB de memória para si. No entanto, esse número pode ser insuficiente para algumas análises uma vez que ele influencia diretamente o número de topologias que o TNT poderá manter na memória. Por exemplo, se você digitar o comando hold no *prompt* do TNT saberá quantas topologias ele irá manter durante sua busca. Tente modificar esse número para 1000 (*i.e.*, `hold 1000;`). Você não deve ter tido nenhum problema, mas se tentar modificar para 10000 verá que o TNT não poderá fazê-lo. No entanto, se você alocar mais memória ao TNT, *i.e.*, "`mxram 512;`" isso será possível. **Um detalhe, quando "mxram" é modificado, o TNT requer que você leia seus dados novamente!**

Algoritmos de refinamento: Por *default*, o TNT utiliza TBR após a construção de árvores de Wagner ao invés de SPR. O TBR é um algoritmo mais agressivo de refinamento (*branch swapping*), porém sua complexidade possui uma função cúbica ao passo que SPR possui uma complexidade quadrática. Portanto, a complexidade desses algoritmos tem influência direta no tempo de execução. Considere que SPR é um subconjunto das topologias examinadas por TBR.

vi. Antes de passar para o próximo exercício, manipule esses comandos e opções para que você tenha uma ideia melhor de como o TNT pode ser controlado.

Exercicio 4.9

Neste último exercício, você deverá conceber um experimento de 15 a 20 execuções de TNT usando a matriz em `zilla.tnt` variando o número de adições aleatórias e o número de topologias mantidas em cada uma das réplicas. Seu objetivo é conceber um desenho experimental no qual você possa avaliar como esses dois parâmetros afetam a eficiências de sua análise. A eficiência deve ser medida pelo tempo necessário para obter o menor custo e o número de topologias recuperadas. Para cada execução você deverá registrar os dados na Tabela 4.2 abaixo. Executado o experimento, responda:

a. Você consegue enxergar algum padrão que lhe indique alguma estratégia eficiente de busca? Qual seria?

b. Você conseguiu obter topologias com custo igual o inferior a 16218? Caso não tenha conseguido, qual seria sua estratégia para chegar a esse custo?

Há três *scripts* no diretório deste tutorial sob a extensão `*.run` que podem automatizar a execução desse exercício. Em comum, esses *scripts* executam o TNT em uma série de vezes e registra os resultados em um arquivo texto. A explicação de um deles será suficiente para que você entenda a lógica dos demais. Considere o conteúdo do arquivo `mu_hold.run`:

```
mxram 512;
macro =;
proc zilla.tnt
hold 10000;
log mu_hold.txt;
loop 10+1 20
```

```
    mu: rep 5 hold #1;
    mu;
stop
log /;
macro -;
zzz
```

Neste *script* – ou macro de TNT, a primeira linha (1) aloca 512M de memória para o programa – o que é desejável em muitos casos, pois isso lhe permite acumular mais topologias durante a execução do programa. A segunda linha habilita a linguagem do macro de TNT. Macros funcionam como uma linguagem interna que lhe permite fazer algumas operações que seus comandos gerais não permitem, como por exemplo *loops* – veja abaixo. Na terceira linha, o TNT é instruído a ler a matriz `zilla.tnt`. A linha 4 define o número máximo de topologias que o TNT poderá manter na memória. Até aqui, não há novidades, o mais interessante começa agora. Na linha 5, o *script* abre um arquivo de log. O log do TNT permite que tudo que é impresso no console do programa durante sua execução seja também direcionado para um arquivo texto. Isso pode ser desejável se você precisa coletar informações de execuções longas ou mesmo documentar o que fez. Na linha 10, esse arquivo de log é fechado. Portanto, tudo que ocorrer entre as linhas 6 e 9 será automaticamente gravado no arquivo `mu_hold.txt`. A linha 6 implementa um comando de macro denominado `loop`. Como o próprio nome sugere, esse comando abre um ciclo de execuções. Este ciclo começa no 10 e termina no 20 em intervalos de 1, portando "$10 + 1$ 20". Na linha 7, a variável "#1" receberá o valor da contagem de cada `loop` para definir os parâmetros de `mu`. Desta forma, no primeiro `loop` a linha 7 será "`mu: rep 5 hold 10`", no segundo "`mu: rep 5 hold 11`", no terceiro "`mu: rep 5 hold 12`", etc – até "`... hold 20`". Para cada modificação da linha 7, o TNT executa a busca heurística pelo comando "`mu`". Ao final (*e.i.*, `hold 20`) o `loop` é finalizado (lina 9), o log é fechado (linha 10), a linguagem macro é desabilitada (linha 11) e o TNT é fechado. Se você examinar os outros dois arquivos com a extensão `*.run` verá que eles obedecem a mesma lógica. Você pode mudar os valores do loop como quiser ou até mesmo implementar outros comandos de TNT dentro e fora do `loop`. Bom exercício!

Tabela 4.2: Número de sequências aleatórias (**RAS**), topologias mantidas em cada réplica, 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	# Rearrangements	Time	# Hits Best	Best Score

Tabela 4.2 – Continuação.

RAS	Trees/RAS	# Rearrangements	Time	# Hits Best	Best Score

4.8 Referências

1. Goloboff, P.; Farris, J. S. & Nixon, K. 2008. TNT a free program for phylogenetic analysis. *Cladistics* **24**: 1–14.
2. Land, A. H. & Doig, A. G. 1960. An automatic method of solving discrete programming problems. *Econometrica* **28**: 497–520.
3. Hendy, M. D. & Penny, D. 1982. Branch and bound algorithms to determine minimal evolutionary trees. *Mathematical Biosciences* **60**: 133–142.
4. Wagner, W. H. 1961. Problems in the classification of ferns. *Recent Advances in Botany* **1**: 841–844.
5. Camin, J. H. & Sokal, R. R. 1965. A method for deducing branching sequences in phylogeny. *Evolution* **19**: 311–326.
6. Robinson, D. F. 1971. Comparison of labelled trees with valency three. *Journal of Combinatorial Theory* **11**: 105–109.
7. Swofford, D. 1990. PAUP: Phylogenetic Analysis Using Parsimony. Champaign, IL: Illinois Natural History Survey, 1990.
8. Farris, J. S. 1970. A method for computing Wagner trees. *Systematic Zoology* **19**: 83–92.

9. Swofford, D. L.; Olsen, G. J.; Waddell, P. J. & Hillis, D. M. em *Molecular Systematics, 2nd. Edition* eds. Hillis, D. M.; Moritz, C & Mable, B. K., 655. Sunderland: Sinauer Associated, 1996.

10. Page, R. D. M. & Holmes, E. C. 1998. Molecular Evolution: A phylogenetic approach. Boston: Blackwell Science, 1998.

11. Schuh, R. T. 2000. Biological Systematics. Principles and Applications. Ithaca: Cornell University Press, 2000.

12. Felsenstein, J. 2004. infering Phylogenies. Sunderland, Massachussets: Sinauer Associated, 2004. 664.

13. Giribet, G. 2007. Efficient tree searches with Available Algorithms. *Evolutionary Bioinformatics* **3**: 341–356.

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.

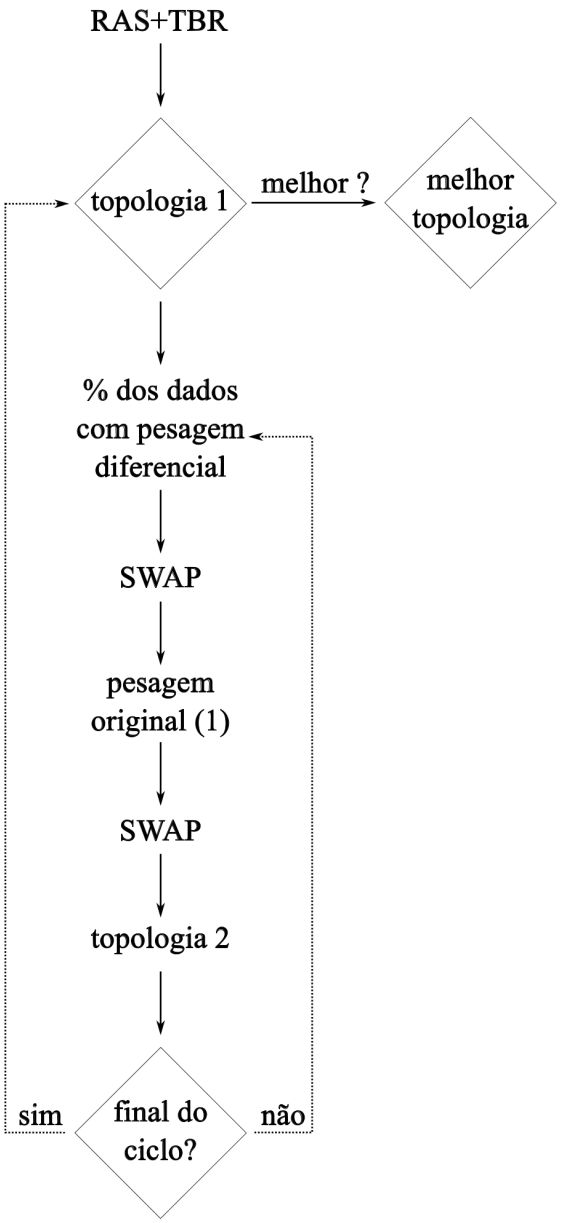


Figura 5.1: Fluxo de iterações de *ratchet*. **RAS**, *radom addition sequence*; **SWAP**, *branch swapping* via SPR ou TBR. Pesagem diferencial geralmente entre 5 e 10% dos caracteres. Número de ciclos (k) variável.

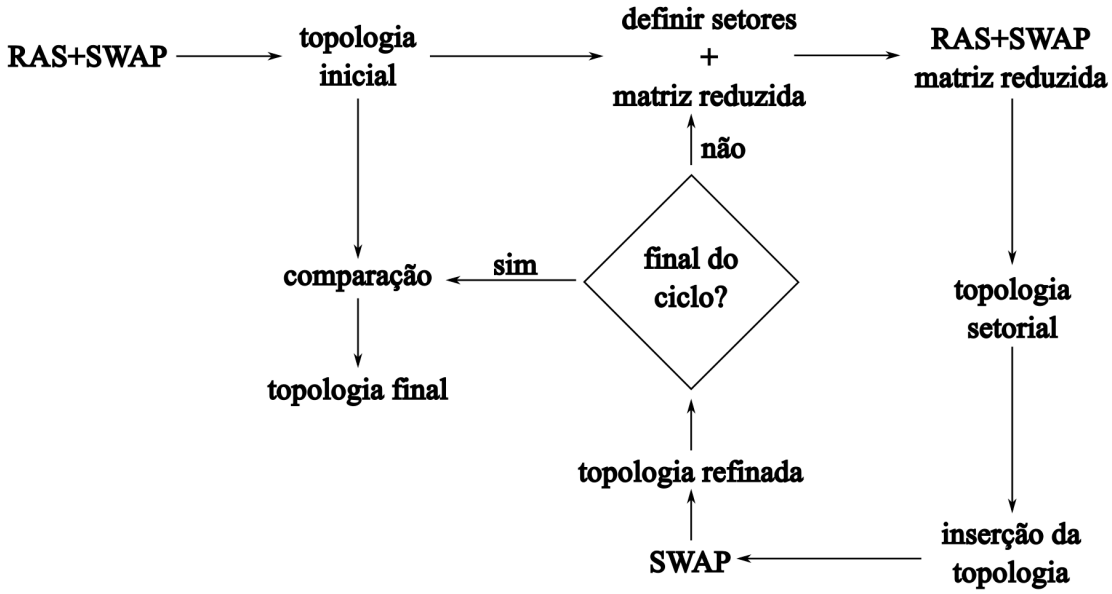


Figura 5.2: Fluxo de iterações de buscas setoriais de acordo com Goloboff [2].

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 addition sequence*, 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.

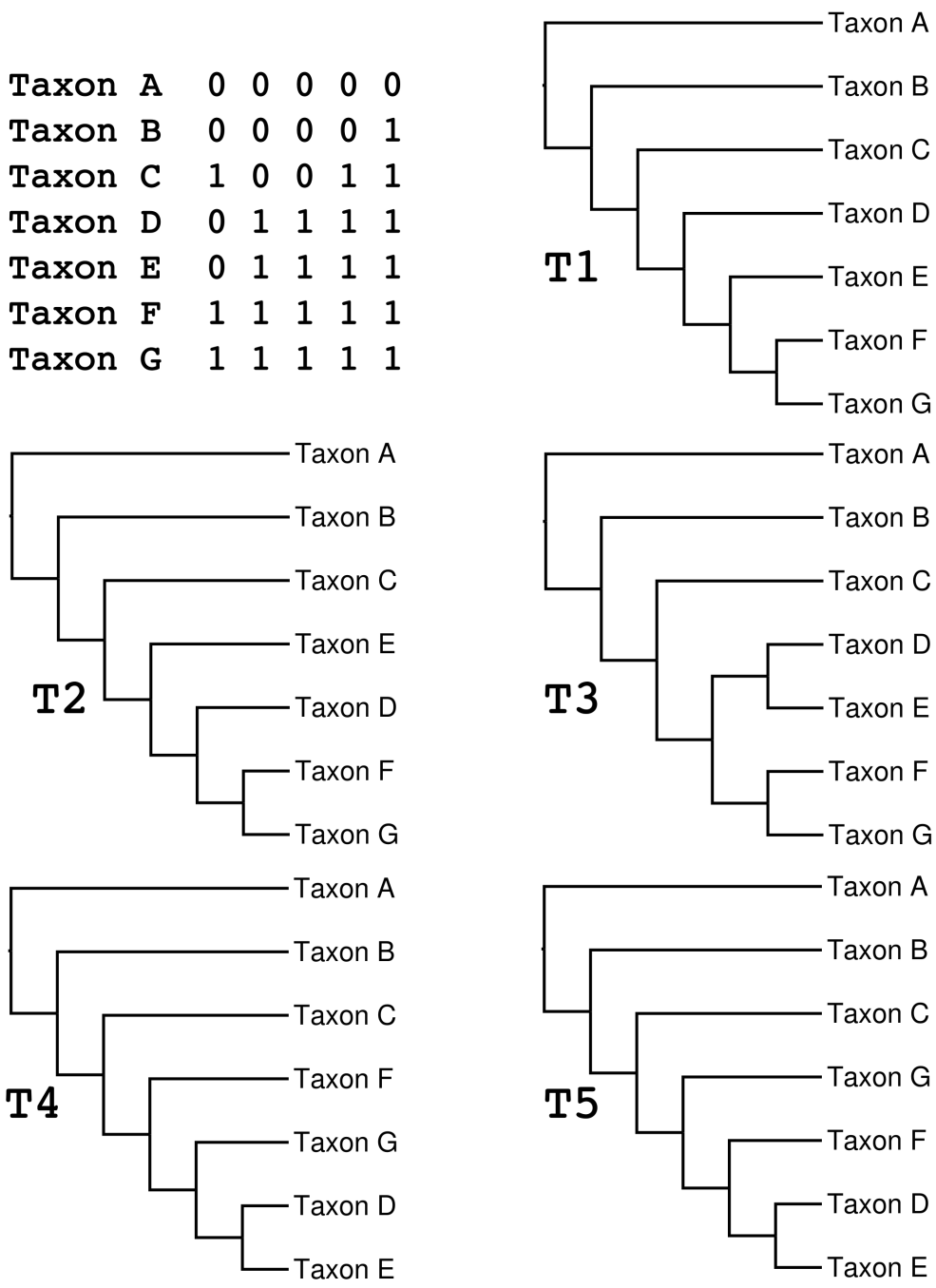


Figura 5.3: Matriz da Tabela 1 de Coddington & Scharff [6] e 5 topologias obtidas em TNT por enumeração implícita e opção de colapso de ramos `collapse 0`.

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.

Tutorial 6

TNT - tipos de caracteres e topologias de consenso

BIZ0433 - INFERÊNCIA FILOGENÉTICA: FILOSOFIA, MÉTODO E APLICAÇÕES.

Conteúdo

Objetivo

O primeiro objetivo deste tutorial é apresentar como caracteres são, ou podem ser tratados em análises filogenéticas. Neste componente do tutorial iremos explorar os impactos do ordenamento de caracteres em inferência filogenética e aplicação de custos arbitrários para séries de transformação. O segundo objetivo deste tutorial é apresentar algumas técnicas de consenso e como computá-las operacionalmente em TNT. Por fim, é apresentado um protocolo para averiguar estabilidade de topologias de consenso em espaços de árvores complexos. 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/
```

6.1 Tipos de caracteres

Dentro do contexto de homologia estática de caracteres (*senso* Wheeler [1]), no qual caracteres e seus respectivos estados de caráter (*i.e.*, séries de transformação) são postulados a priori, existem 3 classes de caracteres: aditivos (ordenados), não-aditivos (não-ordenados) e caracteres de Sankoff (matrizes de custo). Caracteres aditivos [2] são aqueles em que o custo de transformação é determinado pela diferença entre o índice de cada estado no qual cada índice sucessivo representa um incremento de proposições de homologias mais restritivo [3]. Considere por exemplo a Figura 6.1A. Assumindo que a raiz desta transformação é o estado "0", a aquisição do estado "2" assume que a transformação passou pelo estado "1", ou seja, "0" → "1" → "2". Por outro lado, séries de transformações não-aditivas [4], o custo de cada transformação é constante entre qualquer par de estados de caráter (veja Figura 6.1B). Neste caso, a aquisição do estado "2" assume apenas uma transformação independente do estado plesiomórfico do qual este derivou (*i.e.*, "0", "1", ou "3"; Figura 6.1B).

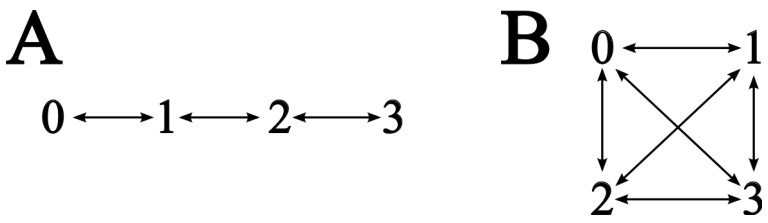


Figura 6.1: Tipos de caracteres: aditivo e não-aditivo. **A**, caráter aditivo; **B**, caráter não-aditivo.

Não há consenso na literatura sobre ordenar ou não séries de transformações de caracteres multi-estados (seja [5–7]). Nixon & Carpenter [7:8] argumenta:

> *Additive characters are just more explicit, compound hypotheses of homology. The fact that nonadditive codings tend to produce shorter trees does not a priori make them better. Throwing away characters, or lying, can also produce shorter trees. Nonadditive codings are better only when they are better justified (or more defensible) in terms of homology assessment. In fact, a nonadditive multistate character implies ambiguity in our understanding of the homology among the states, because it is not generally possible that all allowed transformations are simultaneously true.*

Os argumentos apresentados por Nixon & Carpenter [7] sugerem que é necessário explicitar os critérios que levam o investigador a considerar determinada série de transformação de forma aditiva. Isso porque seguindo o raciocínio desses autores, o ordenamento revela maior conhecimento sobre as noções de homologia putativa entre os estados. No entanto, Hauser & Presch [5] verificaram que a grande maioria dos estudos que consideravam séries de transformações ordenadas não apresentava nenhum critério para determinar a ordem dos estados de caracteres. Tendências morfológicas, sequências ontogenéticas (raramente disponíveis), similaridade entre estados, entre outros, podem ser utilizadas para justificar ordem de estados de caráter [mas veja 5] –, nenhuma delas sem assumir premissas cuja necessidade fica a cargo

do pesquisador. Considere que o ordenamento de estados de caráter representa uma hipótese específica sobre a relação evolutiva entre os estados de caráter.

Vejamos como o TNT permite implementar estes conceitos. A definição de tipos de caracteres em TNT é feita pelo comando "`ccode`":

```
tnt*>help ccode
CCODE
!  re-sets ccode to the one defined in the data file
Other than that, sets character codes.  Specifiers are:
+   make following character(s) additive
-     "      "         "        non-additive
[     "      "         "        active
]     "      "         "        inactive
(     "      "         "        Sankoff
)     "      "         "        non-Sankoff
/N   apply weight N to following character(s)
=N   apply N additional steps to following character(s)
```

Vejamos como podemos implementar caracteres aditivos e não aditivos em TNT. Considere a seguinte matrix:

```
xread
7 6
taxon_A 0000000
taxon_B 1000000
taxon_C 2000011
taxon_D 3001112
taxon_E 4111212
taxon_F 5111322
;
```

Considere os seguintes comandos e seus respectivos efeitos:

i. "`ccode + 0`": Faz com que TNT considere o caráter `0` additivo.

ii. "`cc + 0 4.6`": Faz com que TNT considere os caracteres `0, 4, 5` e `6` aditivos[1].

iii. "`cc -.`": Faz com que TNT considere todos caracteres não-aditivos.

iv. "`cc`": Faz com que TNT exiba a codificação de cada caráter, por exemplo:

```
Ccode
```

[1] Você pode abreviar esse comando utilizando simplesmente "`cc`"

```
+[/1  0     -[/1  1     -[/1  2     -[/1  3     +[/1  4
+[/1  5     +[/1  6     ;
```

Neste exemplo acima, os caracteres `0, 4, 5` e `6` serão considerados aditivos (*i.e.*, identificados pelo símbolo "+"). Observe também que a notação "`[/1`", comum a todos os caracteres, indica que todos os caracteres estão sendo considerados e possuem peso `1`.

Exercicio 6.1

Neste exercício, iremos utilizar a matriz contida no arquivo `exemplo_1a.tnt` para implementar os conceitos descritos acima.

i. Faça uma análise cladística da matrix em TNT do arquivo `exemplo_1a.tnt` e responda:

a. A topologia depende do ordenamento de algum caráter? Justifique.

ii. Faça uma análise cladística da matriz em TNT do arquivo `exemplo_1b.tnt` e responda:

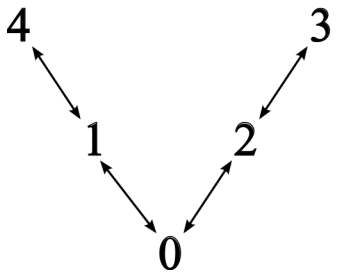
a. Qual caráter quando considerado aditivo reduz o número de topologias igualmente parcimoniosa?

b. Existe algum caráter cujo ordenamento (*i.e.*, "`0`" → "`1`" → "`2`" ...) pode ser defendido utilizando o critério de parcimônia? Justifique.

c. Supondo que você tenha uma justificativa para ordenar o caráter que reduz o número de topologias, implemente a ordenação deste caráter e salve no arquivo

`exemplo_1b_recons.txt` a reconstrução do caráter 2[2] e responda: Quantas reconstruções o TNT postula para esse caráter?

O ordenamento de caracteres não precisa ser necessariamente linear (*i.e.*, "`0`" → "`1`" → "`2`" → "`3`" → "`4`") como nos exemplos acima. Considere, por exemplo, a Figura 6.2. Nela, as relações entre os estados de caráter `0-4` é apresentada de forma ramificada. Assumir essa estrutura hierárquica entre os estados de caráter não é diferente do ordenamento linear desta série de transformação. No entanto, esta relação entre os estados requer uma implementação distinta em TNT. A matriz à direita na Figura 6.2 expressa os custos associados entre cada uma das transformações possíveis. Por exemplo, a transformação "`1`" → "`3`" tem o custo de 3 transformações (*i.e.*, "`1`" → "`0`" → "`2`" → "`3`").



	[0]	[1]	[2]	[3]	[4]
[0]	0	1	1	2	2
[1]	1	0	2	3	1
[2]	1	2	0	1	3
[3]	2	3	1	0	4
[4]	2	1	3	4	0

Figura 6.2: Tipos de caracteres: matriz de transformção.

Esta matriz define os custos de transformação do que chamamos caracteres de Sankoff [8]. Matrizes de Sankoff são implementadas para caracteres para os quais assume-se custos arbitrários de transformação, via de regra associados à uma ou mais premissas sobre a evolução destes caracteres. Embora Sankoff [8] tivesse desenvolvido este procedimento de otimização para estudar evolução macromolecular, matrizes de Sankoff podem ser aplicadas a qualquer série de transformação – mesmo binária.

Em TNT, a implementação de matrizes de Sankoff requer os seguintes passos:

1. Definição da matriz de Sankoff:
   A definição de matrizes de Sabkoff é feita pelo comando `smatrix` do TNT obedecendo a seguinte sintaxe:
   `smatrix =S (xxxx) ...costs... ;`
   onde "`S`" é um número entre 0-31 e indexa internamente a matriz no TNT; "`xxxx`" é o nome atribuído à matriz (opcional) e "`costs`" são os custos de cada transformação. Os custos são definidos considerando a direção da transformação, como por exemplo `0>4 2` que atribui custo 2 para a transformação "`0`" → "`4`", ou estabelecendo custos simétricos, como por exemplo `0/2 1` que atribui custo 1 para as transformações "`0`" → "`2`" ou "`2`" → "`0`".

[2] numeração de acordo com TNT. Veja Tutorial 5 item 5.2 para detalhes de como verificar reconstruções em TNT.

2. Aplicação da matriz de Sankoff:
   Uma vez definida a matriz de Sankoff, ela deve ser aplicada ao caráter desejado, ou caracteres utilizando o comando "`smatrix +S N`" ou "`smatrix +xxxx N`", onde "`N`" é o caráter para o qual se quer aplicar a matriz de Sankoff.
3. Implementação matriz de Sankoff em `ccode`:
   Finalmente, é necessário habilitar o caráter de Sankoff no TNT utilizando o comando "`ccode (N`".

Exercicio 6.2

Neste exercício, você deverá criar uma matriz de Sankoff e implementar esta função de custo na análise filogenética usando TNT. **No entanto, antes de executar esse exercício examine as últimas linhas do arquivo `exemplo_2.tnt`.**

i. Construa uma matriz te Sankoff para a série de transformação ordenada ilustrada na Figura 6.3.

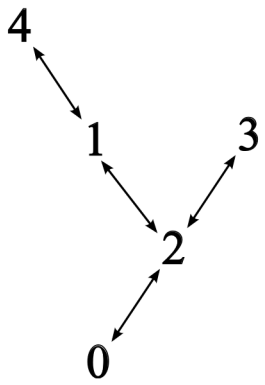


Figura 6.3: Série de transformação ordenada não-linear.

	[0]	[1]	[2]	[3]	[4]
[0]	-				
[1]		-			
[2]			-		
[3]				-	
[4]					-

ii. Com base na sua matriz de Sankoff, defina a `smatrix` abaixo:

```
smatrix =0 (minha_matriz) ...
```

iii. Execute uma busca no TNT com o arquivo `exemplo_2.tnt` e responda:
A implementação da matriz de Sankoff teve algum impacto na reconstrução do caráter 2? Justifique.

Matrizes de Sankoff são muito utilizadas em análise de dados moleculares – em concordância com sua concepção inicial [8]. A Figura 6.4 representa duas classes de transformações (*i.e.*, transições e transversões) de caracteres genotípicos – sequências nucleotídicas – comumente utilizadas em análises filogenéticas de dados moleculares. A premissa associada ao custo diferencial entre essas duas classes de transformações reside na expectativa de que transversões (*i.e.*, purinas $\longleftrightarrow$ pirimidinas) tem impacto bioquímico mais acentuado nas moléculas e, consequentemente, estão sob maiores restrições de transformação do que transições (*i.e.*, purinas $\longleftrightarrow$ purinas ou pirimidinas $\longleftrightarrow$ pirimidinas). Sequências nucleotídicas são sempre consideradas caracteres não-aditivos e via de regra são submetidos à matrizes de Sankoff.

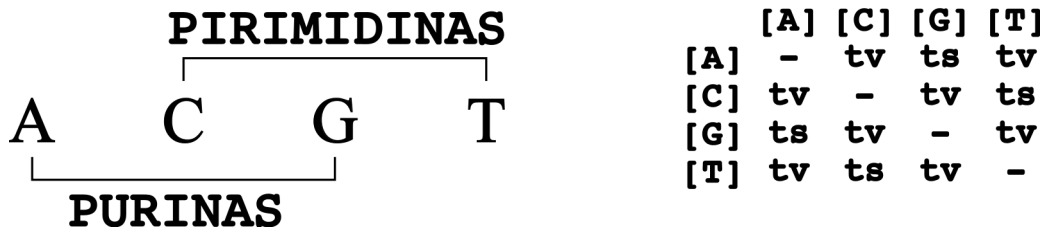


Figura 6.4: Classes de transformação de caracteres genotípicos: transições e transversões

Exercício 6.3

Considere os dados na matriz do arquivo `exemplo_3.tnt` e execute as seguintes tarefas:

i. Faça uma análise filogenética destes dados e responda:

a. Qual é a topologia recuperada e qual é o seu custo (*i.e.*, número de transformações)?

ii. Abaixo defina a matriz de Sankoff no qual você atribuirá peso 2 para transversões e peso 1 para transições:

```
smatrix =0 C/A _ G/A _ G/C _ T/A _ T/C _ T/G _;
```

iii. Modifique o arquivo `exemplo_3.tnt` de modo que todos os caracteres sejam analisados de acordo com sua matriz de Sankoff, reanalise esses dados e responda:

a. A implementação desta nova função de custo modificou seus resultados? Explique.

6.2 Árvores de consenso

Métodos de consenso apresentam um sumário de informações contidas em um conjunto de topologias. Fundamentalmente, topologias de consenso é o resultado da aplicação de uma série de regras e/ou algoritmos sob um conjunto de topologias que resulta em uma única árvore com o mesmo conjunto de terminais [9]. De acordo com Bryant [9], há uma série de controvérsias relacionadas à topologias de consenso; no entanto estas estão mais relacionadas com as interpretações sobre as topologias de consenso do que com os métodos usados para gerá-las [veja, 10].

Há algumas generalidade sobre topologias de consenso que devem ser consideradas. A primeira delas é que topologias de consenso são sumários de informação e o número de transformações nestas topologias é via de regra subótima [11, 12] – portanto, **otimizações em topologias de consenso não devem ser consideradas**. Outra propriedade comum à topologias de consenso, especialmente relacionadas com o sumário de informações filogenéticas provenientes de fontes de dados distintas, é que em alguns casos a topologia de consenso difere do resultado que seria obtido considerando a análise simultânea destas bases de dados (seja Barrett *et al.* [10]). Portanto, considere que a maioria das técnicas de consenso foram concebidas como ferramentas de representação e não para **inferência filogenética** propriamente dita (mas veja Holder *et al.* [13]).

Neste componente do tutorial iremos explorar três tipos de consenso, aqueles mais frequentemente utilizados em análises filogenéticas. Essa exposição breve sobre o tópico tem mais caráter operacional do que fomentar as discussões sobre o uso de métodos de consenso em inferência filogenética.

6.2.1 Consenso estrito

O consenso estrito [14], como o nome sugere, é o mais simples – e o mais frequente na literatura. Esta técnica de consenso produz uma topologia que é o sumário de todos os componentes (*i.e.*, clados) que estão presentes em **todas** as topologias fundamentais.

6.2.2 Consenso semi-estrito

O consenso semi-estrito [15], também conhecido como *compatible components*, é menos restritivo do que o consenso estrito. Neste método de consenso a topologia final contém todos os componentes presentes nas topologias fundamentais **com a inclusão daqueles que não são contraditórios entre si**. Colocado de outra forma, enquanto que no cálculo do consenso estrito um determinado componente só será representado se estiver necessariamente em todas as topologias fundamentais, no consenso semi-estrito ele deve estar presente em pelo menos uma delas e não ser contradito por nenhum outro componente das demais topologias que fazem parte do conjunto.

6.2.3 Consenso de Maioria

O consenso de maioria (*i.e.*, *majority-rule*; Margush & McMorris [16]) baseia-se na frequência dos clados presentes nas topologias fundamentais. Margush & McMorris [16] define este tipo de consenso como sendo topologias de consenso M_l no qual o parâmetro l define a porcentagem mínima da frequência dos componentes que deverão estar presentes na topologia de consenso. Observe que se l é 100% você obtém o consenso estrito das topologias fundamentais. Este tipo de consenso é utilizado em análises de suporte e de inferência filogenética que utiliza como critério de otimização probabilidades posteriores [*i.e.*, análises bayesianas; veja 13].

Exercício 6.4

Neste exercício iremos explorar as propriedades destes métodos de consenso utilizando as topologias existentes no arquivo `exemplo_4.tnt`.

i. Verifique os clados em cada uma das três topologias.

ii. Calcule o consenso estrito utilizando o comando "`ne`".

iii. Represente a tolologia de consenso no espaço abaixo e compare novamente com as topologias fundamentais.

iv. Calcule o consenso semi-estrito utilizando o comando "`comcomp`".

v. Represente a topologia de consenso no espaço abaixo e compare novamente com as topologias fundamentais.

vi. Como é a resolução desta topologia de consenso em relação à topologias fundamentais?

_____

_____

vii. Calcule o consenso de maioria utilizando o comando "`majority = 50`".

viii. Represente a topologia no espaço abaixo e compare novamente com as topologias fundamentais.

ix. Qual observação você faria ao comparar as topologias fundamentais com a de consenso de maioria?

_____

_____

6.2.4 Estabilidade do consenso

Independente da técnica de consenso que você deseja utilizar para calcular topologias de consenso, é importante que você tenha uma boa amostra de topologias do seu espaço de árvore – principalmente quando este apresenta relativa complexidade. Neste último componente sobre o assunto irei sugerir um protocolo para verificar se, potencialmente, você obteve estabilidade em sua topologia de consenso. Considere que nem sempre a inspeção visual é uma forma imediata de observar se houve ou não mudança entre uma topologia de consenso e outra à medida em que você compila topologias com o mesmo custo. Considere por exemplo os arquivos `consenso_*.tre`. Suponha que esses arquivos foram obtidos com buscas incrementalmente mais agressivas e que à cada uma delas um número maior de topologias foi amostrado. Se você listar de forma longa esses arquivos (*i.e.*, com o comando `ls -l`), você obterá:

```
-rw-rw-r- 1 alan alan 84 Apr  7 20:29 consenso_1.tre
```

```
-rw-rw-r- 1 alan alan 82 Apr  7 20:33 consenso_2.tre
-rw-rw-r- 1 alan alan 80 Apr  7 20:33 consenso_3.tre
-rw-rw-r- 1 alan alan 80 Apr  7 20:33 consenso_4.tre
```

A primeira dica refere-se ao tamanho desses arquivos. Observe que o arquivo consenso_1.tre possui 84 bites, o consenso_2.tre 82 e os demais 80. O número decremental de bites sugere que os arquivos possuem um número menor de caracteres, ou seja, parênteses que são usados para definir grupos! Inspecione o conteúdo destes arquivos.

No caso da sugestão acima, você deve considerar que dois arquivos podem possuir o mesmo número de bites e, no entanto, suas topologias podem ser distintas. Um outra forma de verificar, ou comparar essas topologias de forma mais segura, é utilizar o comando "`diff`" – um comando interno de LINUX/UNIX. Se você executar em um terminal:

```
$ diff -q -s consenso_1.tre consenso_2.tre
```

você deverá obter:

```
Files consenso_1.tre and consenso_2.tre differ
```

Por outro lado, se você executar em um terminal:

```
$ diff -q -s consenso_3.tre consenso_4.tre
```

você deverá obter:

```
Files consenso_3.tre and consenso_4.tre are identical
```

Exercicio 6.5

Neste exercício você irá explorar estabilidade de consensos. Você deverá fazer algumas análises em TNT, compilar os dados na Tabela 6.1 e identificar a partir de que momento destas análises você obteve a estabilidade de sua topologia de consenso.

Considere os seguintes comandos de TNT e suas respectivas ações:

```
log log_run.txt;
xmu:  hold 10 rep 50 ratchet 5 drift 5 fuse 10;
xmu;
ne*;
tchoose/;
tsave* busca_1.tre;
save;
tsave/;
log/;
```

Esta sequência de comandos abre um arquivo de *log* chamado `log_run.txt`, executa uma busca usando novas tecnologias em TNT para um determinado arquivo de entrada, calcula o consenso estrito e faz com que a topologia de consenso seja inserida no *buffer* de memória do TNT, seleciona a última topologia e descarta as demais (no caso a topologia de consenso é mantida), abre um arquivo para salvar topologias chamado `busca_1.tre`, salva a topologia e fecha os arquivos de árvore e de *log*.

Neste exercício você deverá estabilizar o consenso as topologias encontradas na matriz `zilla.tnt` em 10 buscas. Os números de topologias encontradas em cada uma dessas análises deverá ser sempre superior à encontrada na análise anterior. Você deverá preencher a tabela abaixo (Tabela 6.1) e comparar os arquivos que julgar necessário com o comando `diff -q -s [arquivo1] [arquivo2]`.

Tabela 6.1: Buscas heurísticas e estabilidade de consenso em TNT

Busca	Parâmetros de Busca	# Topologias retidas	Tamanho do Arquivo
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			

Observações de comparação:

6.3 Referências

1. Wheeler, W. C. 2001. Homology and the optimization os DNA sequence data. *Cladistics* **17**: S3–S11.
2. Farris, S. 1970. Methods for computing Wagner trees. *Systematic Zoology* **19**: 83–92.
3. Wheeler, W. C. 2012. Systematics: a course of lectures. Malaysia: Wiley-Backwell, 2012. 426.
4. Fitch, W. M. 1971. Toward defining the course of evolution: Minimum change for a specific tree topology. *Systematic Zoology* **10**: 406—416.
5. Hauser, D. L. & Presch, W. 1991. The effect of ordered characters on phylogenetic reconstrucion. *Cladistics* **7**: 243–265.
6. Slowinski, J. B. 1993. "Unordered" versus "ordered" characters. *Systematic Biology* **42**: 155–165.
7. Nixon, K. C. & Carpenter, J. M. 2011. On homology. *Cladistics* **27**: 1–10.
8. Sankoff, D. 1975. Minimal mutation trees of sequence. *SIAM Journal on Applied Mathematics* **28**: 35–42.
9. Bryant, D. em *DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Volume 61* eds. Janowitz, M. F.; Lapointe, F. J.; McMorris, F. R.; Mirkin, B & Roberts, F. S. Providence, Rhode Island: American Mathematical Society, 2003.
10. Barrett, M.; Donoghue, M. J. & Sober, E. 1991. Against consensus. *Systematic Zoology* **40**: 486–493.
11. Miyamoto, M. M. 1985. Consensus cladograms and general classifications. *Cladistics* **1**: 186–186.
12. Carpenter, J. M. 1988. Choosing among equally parsimonious cladograms. *Cladistics* **4**: 291–296.
13. Holder, M. T.; Sukumaran, J. & Lewis, P. O. 2008. A justification for reporting the Majority-rule consensus tree in Bayesian Phylogenetics. *Systematic Biology* **57**: 814–821.
14. Rohlf, F. J. 1982. Consensus indices for comparing classifications. *Mathematical Biociences* **59**: 131–144.
15. Bremer, K. 1990. Combinable component consensus. *Cladistics* **6**: 369–372.
16. Margush, T & McMorris, F. R. 1981. Consensus n-trees. *Bulletin of Mathematical Biology* **2**: 239–244.

Tutorial 7

Dados Moleculares - Introdução

BIZ0433 - INFERÊNCIA FILOGENÉTICA: FILOSOFIA, MÉTODO E APLICAÇÕES.

Conteúdo

Objetivo

O objetivo deste tutorial é apresentar ao aluno algumas ferramentas de manipulação de dados moleculares desde a obtenção de dados no GenBank/NCBI à configurações de matrizes de dados passíveis de serem analisadas em programas de análise filogenética. Ao concluir esse tutorial, o aluno será capaz de reanalisar dados já publicados utilizando TNT. 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/
```

7.1 GenBank

7.1.1 Formatos de arquivos

Antes de entrarmos na parte analítica de dados moleculares é necessário entender alguns formatos de arquivos que contenham dados moleculares. Há vários deles, mas em caráter introdutório iremos apresentar apenas dois. Se você está familiarizado com artigos que usam dados moleculares para inferência filogenética, já deve ter notado que grande parte deles deposita sequências nucleotídicas ou pepitídicas em um repositório chamado GenBank. De fato, todos os periódicos obrigam autores a depositarem esses dados em repositórios como o **GenBank** e informar aos leitores os números de tombo. Observe a tabela no APPENDIX I de Dias *et al.* [1], publicação anexada a este tutorial. As colunas sob o termo "*GenBank accession number*" contém os números de tombo das sequências nucleotídicas ou pepitídicas depositadas pelos autores para cada uma das regiões genômicas consideradas nesse estudo (*i.e.*, cyt b, 16S, Rag-1 e 12S). Estes números de tombo permitem que qualquer pessoa verifique e use esses dados.

O acesso ao banco de sequências nucleotídicas do GenBank é feito pelo endereço: http://www.ncbi.nlm.nih.gov/nucleotide. Se você possui um determinado número de tombo, por exemplo FJ685663 – que de acordo com a tabela apresentada por Dias *et al.* [1] refere-se ao fragmento de Citocromo B de *Cycloramphus acangatan*–, você poderá obter os dados deste número de tombo simplesmente usando-o na busca disponível nessa página.

Por *default*, quando você executa uma busca no GenBank com um único número de tombo, o GenBank retorna o resultado em seu formato mais detalhado. Neste caso em particular isso seria:

```
Cycloramphus acangatan voucher AF1605 cytochrome b (cytb) gene, partial cds; mitochondrial
GenBank: FJ685663.1
FASTA Graphics PopSet
Go to:
LOCUS       FJ685663                611 bp    DNA     linear   VRT 15-APR-2009
DEFINITION  Cycloramphus acangatan voucher AF1605 cytochrome b (cytb) gene,
            partial cds; mitochondrial.
ACCESSION   FJ685663
VERSION     FJ685663.1  GI:226510745
KEYWORDS    .
SOURCE      mitochondrion Cycloramphus acangatan
  ORGANISM  Cycloramphus acangatan
            Eukaryota; Metazoa; Chordata; Craniata; Vertebrata; Euteleostomi;
            Amphibia; Batrachia; Anura; Neobatrachia; Hyloidea; Cycloramphidae;
            Cycloramphus.
REFERENCE   1  (bases 1 to 611)
  AUTHORS   Amaro,R.C., Pavan,D. and Rodrigues,M.T.
  TITLE     On the generic identity of Odontophrynus moratoi Jim & Caramaschi,
            1980 (Anura, Cycloramphidae)
  JOURNAL   Zootaxa 2071, 61-68 (2009)
REFERENCE   2  (bases 1 to 611)
```

```
  AUTHORS   Amaro,R.C., Pavan,D. and Rodrigues,M.T.
  TITLE     Direct Submission
  JOURNAL   Submitted (30-JAN-2009) Departamento de Zoologia, Instituto de
            Biociencias, Universidade de Sao Paulo, Rua do Matao, ravessa 14,
            101, Sao Paulo 05508-090, Brazil
FEATURES             Location/Qualifiers
     source          1..611
                     /organism="Cycloramphus acangatan"
                     /organelle="mitochondrion"
                     /mol_type="genomic DNA"
                     /specimen_voucher="AF1605"
                     /db_xref="taxon:635146"
     gene            <1..>611
                     /gene="cytb"
     CDS             <1..>611
                     /gene="cytb"
                     /codon_start=3
                     /transl_table=2
                     /product="cytochrome b"
                     /protein_id="ACO59916.1"
                     /db_xref="GI:226510746"
                     /translation="LIMQIAPGLFLAMHYTADTSLAFSSIAHICRDVNNGWLLRSLHA
                     NGASFFFICIYLHIGRGLYYGSYLFKETWNIGVILLFMTMATAFVGYVLPWGQMSFWG
                     ATVITNLLSAAPYIGTDLVQWIWGGFSVDNATLTRFFTFHFILPFIVTGLILLHLIFL
                     HETGSSNPTGLNPNPDKVPFHTYFSYKDILGFAIMLSLLASLS"
ORIGIN
        1 gcttaattat acaaattgca ccaggactat tcttagccat acattacacc gcggatacct
       61 cattagcatt ttcatctatt gcccatatct gccgagatgt aaacaacggg tgacttcttc
      121 gaagcctcca cgcaaatggt gcctcattct tctttatctg tatctacctt catattggcc
      181 gaggattata ctacggctca tacttattta aagaaacatg aaacattgga gtgattctcc
      241 tatttataac catagccaca gcctttgtcg gatacgtact accatgagga caaatatcat
      301 tctgggggcc cacagtcatc accaacttat tatctgcagc cccctatatt ggcacagact
      361 tagtgcaatg aatctgaggc ggattttcag tagataacgc caccctcaca cgcttcttca
      421 catttcactt tatcctccca ttcattgtta caggattaat cctcctacac ctaatctttc
      481 ttcatgaaac aggatcttca aaccccacag gcctaaaccc taacccagat aaagtcccat
      541 tccacaccta cttctcctat aaagacatcc taggatttgc catcatactc tcccttcttg
      601 cctcactatc a
```

Observe o resultado da busca acima no qual o GenBank lhe fornece uma série de informações sobre essa sequência de nucleotídeos, tais como: autores, região genômica, posicionamento taxonômico do organismo sequenciado, tradução para aminoácidos (para regiões codificantes) e a sequência nucleotídica propriamente dita. Desta forma, este é o formato que contém o maior número de informações sobre a sequência nucleotídica ou peptídica associada ao número de tombo. Minha sugestão é que sempre que possível, obtenha as sequências no Genbank neste formato, pois isso evita ter que revisitar este banco de dados para obter informações adicionais no futuro caso você esteja usando dados disponíveis nesse repositório.

Há um outro formato que é muito utilizado na manipulação e análise de sequências nucleotídicas ou pepitídicas: o formato FASTA. Na sua forma mais simples, ele contém terminais precedidos

de ”>” e em outra linha a sequência nucleotídica ou pepitídica. Por exemplo:

```
>sequencia_1
ACGTACGTACGT
>sequencia_2
AAGTAAGTAAGT
>sequencia_3
AAATAASTAAAT
```

Esse formato é bem simples, útil e frequentemente utilizado para transformar sequências nucleotídicas ou pepitídicas em matrizes de dados. No entanto, este formato permite inserir comentários e outras informações no cabeçalho da sequência. Vejamos por exemplo como o GenBank retornaria a mesma sequência acima (*i.e.*, FJ685663) no formato FASTA:

```
Cycloramphus acangatan voucher AF1605 cytochrome b (cytb) gene, partial cds; mitochondrial
GenBank: FJ685663.1
GenBank Graphics PopSet
>gi|226510745|gb|FJ685663.1| Cycloramphus acangatan voucher AF1605 cytochrome b (cytb) ...
GCTTAATTATACAAATTGCACCAGGACTATTCTTAGCCATACATTACACCGCGGATACCTCATTAGCATT
TTCATCTATTGCCCATATCTGCCGAGATGTAAACAACGGGTGACTTCTTCGAAGCCTCCACGCAAATGGT
GCCTCATTCTTCTTTATCTGTATCTACCTTCATATTGGCCGAGGATTATACTACGGCTCATACTTATTTA
AAGAAACATGAAACATTGGAGTGATTCTCCTATTTATAACCATAGCCACAGCCTTTGTCGGATACGTACT
ACCATGAGGACAAATATCATTCTGGGGGGCCACAGTCATCACCAACTTATTATCTGCAGCCCCCTATATT
GGCACAGACTTAGTGCAATGAATCTGAGGCGGATTTTCAGTAGATAACGCCACCCTCACACGCTTCTTCA
CATTTCACTTTATCCTCCCATTCATTGTTACAGGATTAATCCTCCTACACCTAATCTTTCTTCATGAAAC
AGGATCTTCAAACCCCACAGGCCTAAACCCTAACCCAGATAAAGTCCCATTCCACACCTACTTCTCCTAT
AAAGACATCCTAGGATTTGCCATCATACTCTCCCTTCTTGCCTCACTATCA
```

7.1.2 Obtendo sequências do GenBank

Todos os arquivos de sequências são arquivos texto. Portanto, nada impede que você copie e cole os dados diretamente da página no Genbank no formato que desejar em um editor de texto. Isso porém pode se tornar inviável e extremamente tedioso – dependendo do número de sequências que você precisa retirar do repositório. O GenBank possui uma ferramenta para retirar sequências nucleotídicas de forma bem eficiente. Essa ferramenta é chamada *Batch Entrez* e pode ser acessada pelo endereço `http://www.ncbi.nlm.nih.gov/sites/batchentrez`. A única coisa que você precisa para acessar essa ferramenta é possuir um arquivo texto contendo todos os números de tombo que deseja.

Vejamos o exemplo abaixo. Suponha que você possua um arquivo com o seguinte conteúdo[1]:

```
FJ685663
FJ685664
FJ685665
...
KF214177
```

[1] Estes são alguns números de tombo para as sequências de Citocromo B de Dias *et al.* [1]

```
KF214178
FJ685662
```

Basta acessar a página do *Batch Entrez*/Genbank e fazer o *upload* do seu arquivo selecionando a opção *Choose File* (veja Figura 7.1) e em seguida pressionar *Retrieve*.

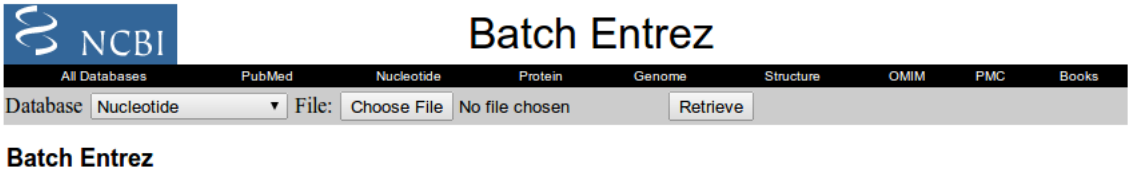


Figura 7.1: Página do *Batch Entrez* do GenBank.

Você deverá obter:

```
Received lines:  40
Rejected lines:  0
Removed duplicates:  0
Passed to Entrez:  40
Retrieve records for 40 UID(s)
```

Neste exemplo em particular, meu arquivo possuía 40 números de tombo e todos eles foram recuperados sem erro.
O próximo passo é obter os registros das sequências pressionando "`Retrieve records for 40 UID(s)`". Você deverá obter o resultado ilustrado na Figura 7.2.

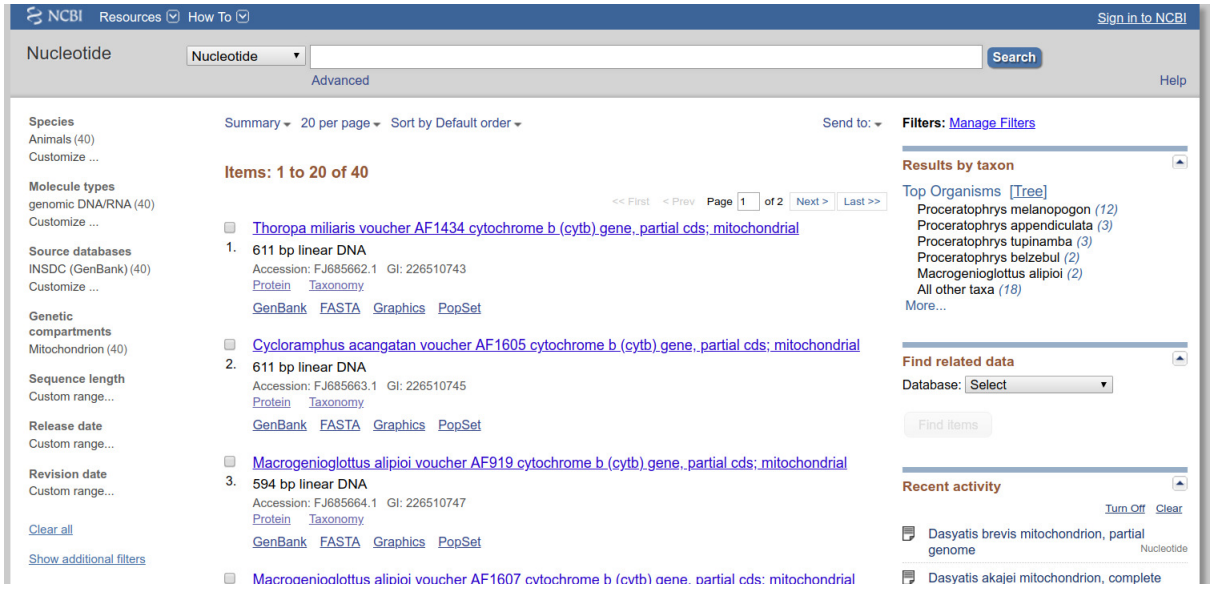


Figura 7.2: Página do *Batch Entrez* do GenBank exibindo resultado de uma busca.

Por *default* o GenBank irá apresentar as sequências solicitadas de forma sumarizada (veja Figura 7.2). Isto lhe permite verificar se os números de acesso de fato referem-se às sequências que estava interessado e/ou selecionar parte delas para exame mais detalhado e/ou para baixá-las. Para baixar

essas sequências, você precisa definir o formado que deseja utilizar. A definição do formado deve ser selecionada dentre as opções disponíveis nas opções de *Summary* (veja Figura 7.3).

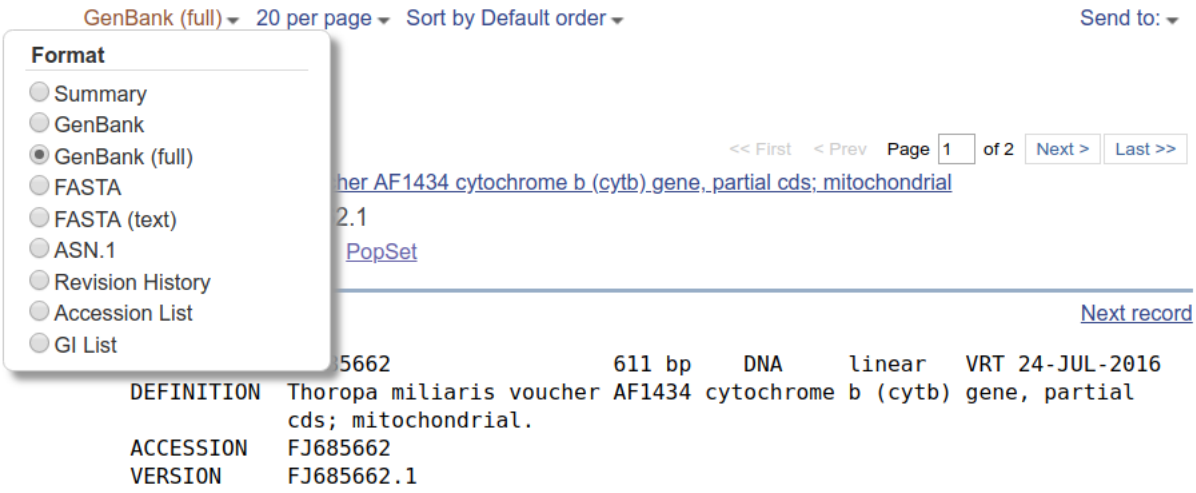


Figura 7.3: Página do *Batch Entrez* do GenBank exibindo as opções para exibição dos resultados de busca.

Ao selecionar, por exemplo, o formato *Genbank (full)*, você deverá obter a exibição das sequências no formato referido.

No entanto, para baixar as sequências em um determinado formato, não é necessário exibí-las da mesma forma. O *download* de sequências é executado pela seleção das opções em *Send to* (veja Figura 7.3). Ao pressionar *Send to*, você deverá obter uma janela com as opções exibidas na Figura 7.4.

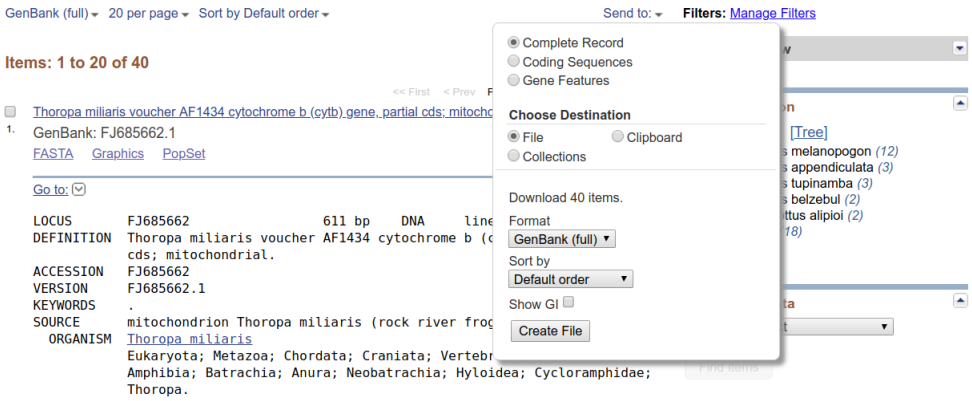


Figura 7.4: Página do *Batch Entrez* do GenBank exibindo as opções para baixar os resultados de busca.

Uma vez selecionadas as opções que deseja nesta janela, basta pressionar *Create File*. O arquivo será salvo com o nome de "`sequence.gb`" no diretório de *default* para *download* de seu sistema. Minha recomendação é que você modifique o nome deste arquivo de forma que seu nome indique seu conteúdo logo após baixar as sequências – ao fazê-lo, mantenha a extensão "`.gb`" para indicar o formato do arquivo.

Exercício 7.1

O arquivo "`dias_et_al_2013_accession_numbers.csv`" contém todos os números de tombo para as sequências utilizadas por Dias *et al.* [1] em um arquivo texto no formato CSV (*i.e.*, *Comma Separated Values*). Arquivos neste formato podem ser abertos em programas que manipulam planilhas, tais como EXCEL da Microsoft e OpenOffice Calculator. Este arquivo em particular possui 5 colunas, cada uma referente a uma região genômica utilizada por Dias *et al.* [1]. Para cada uma destas colunas você deverá

i. Gerar um arquivo texto que contenha apenas os números de tombo referentes a respectiva região genômica. Para cada um dos arquivos, utilize a extensão "`.acc`" – de *accession* (*e.g*, `12s.acc`). Esta não é uma extensão particular, somente uma sugestão para que você possa mater seus arquivos organizados.

Dica: Inspecione o arquico em CSV, voce poderia usar a seguinte linha de comando para extrais os números de acesso do cyt-b, que estão na segunda coluna, com a seguinte linha de comando: `tail -n +2 dias_et_al_2013_accession_numbers.csv | cut -d',' -f5 | sed 's/"//g' > 12s.acc`

ii. Utilizar a feramenta do *Batch Entrez* para recuperar os registros referentes a cada um dos arquivos.

Atenção: Ao fazer o download do GenBank/NCBI o repositório criará um documento chamado `sequence.gb` que será salvo no diretório de Download de seu sistema.

iii. Salvar as sequências de cada região genômica em seus respectivos arquivos no formato *GenBank (full)*. Para cada um dos arquivos, utilize a extensão "`.gb`" – de *GenBank* (*e.g*, `12s.gb`).

7.2 Manipulação de arquivos de sequência

Nosso objetivo a partir deste momento é extrair as informações dos arquivos gerados pelo GenBank para transformar essas sequências em um arquivo de dados que possa ser analisado por programas de inferência filogenética. No entanto, programas distintos são capazes de ler arquivos em formatos diferentes. No que se segue, manipularemos os arquivos contendo as sequências do Genbank com o objetivo de gerar uma matriz de dados que possa ser analisada em TNT [2]. Os passos apresentados a seguir poderão ser modificados à medida que você se familiarize com as inúmeras ferramentas disponíveis para manipulação de arquivos de sequência. Seja qual for a ferramenta que você venha a escolher no futuro, há dois componentes principais que eu considero importante na preparação de arquivos de sequências para análises filogenéticas. O primeiro deles é o nome dos terminais. O segundo é verificar se as sequências estão alinhadas e no mesmo sentido.

7.2.1 Compondo arquivos FASTA

Os arquivos no formato *GenBank (full)* possuem um grande número de informações. No entanto, muitas delas não podem ser inseridas em um arquivo que se destine a análise filogenética. Para este último propósito, os nomes dos terminais devem ser curtos e bem definidos e as sequência devem estar alinhadas[2]. O grande número de informações contido nos arquivos no formato *GenBank*

[2] Alinhamentos serão discutidos com detalhes nos próximos tutoriais.

(full), por outro lado, permite que nós possamos extrair o que queremos.

A ferramenta que iremos utilizar nesta primeira etapa de manipulação de arquivos de sequência é o *script* chamado *phyloconvert*. Esse *script* foi concebido primariamente para gerar arquivos no formato FASTA para serem analisados diretamente ou serem subsequentemente transformados em arquivos passíveis de serem analisados em TNT e/ou PAUP [2, 3].

O *script* chamado *phyloconvert* faz parte dos aplicativos disponíveis na imagem utilizada no curso, mas caso você não esteja utilizando a imagem, ele está disponível no diretório "`tutorial_7`". Este *script* requer que BioPython esteja instalado no seu sistema.

A execução do *script* chamado *phyloconvert* é feita do terminal:

```
$ phyloconvert
```
[3]

Ao executá-lo você obterá as seguintes opções:

```
PhyloConvert 0.0.4 - May 2022 by Fernando Marques

#########################################################################
#          THIS PROGRAM CONVERTS FILES FOR PHYLOGENETIC ANALYSES        #
#                                                                       #
# YOUR OPTIONS ARE:                                                     #
#                                                                       #
#   1. For GenBank (*.gb) format to FASTA/POY using accession numbers.  #
#   2. For GenBank (*.gb) format to FASTA/POY using terminal names.     #
#   3. For GenBank (*.gb) format to FASTA/POY using numbers and names.  #
#   4. For clean FASTA files to XREAD format (for TNT).                 #
#   5. For clean FASTA files to NEXUS format (for PAUP).                #
#   6. Generate accession numbers list from GenBank (*.gb) file.        #
#   7. Generate taxon name list from GenBank (*.gb) file.               #
#   8. Generate a file containing translation rules for SED.            #
#   x. Exit program.                                                    #
#                                                                       #
#########################################################################

Select the option desired:
```

As três primeiras opções manipulam arquivos no formato *GenBank (full)* transformando-os em arquivos FASTA. A diferença entre estas opções reside na informação que será inserida nos terminais. Por exemplo:

```
Select the option desired:  1
```

[3] Se você quer executar o script que está no diretório você deverá digitar "`./phyloconvert`".

Resulta em:

```
>FJ685662
GCCTAATTACACAAATTATTACAGGACTTTTTTT...
>FJ685663
GCTTAATTATACAAATTGCACCAGGACTATTCTT...
>FJ685664
GTCACAGGACTCTTCCTTGCAATACACTATACTG...
...
```

```
Select the option desired:  2
```

Resulta em:

```
>Thoropa_miliaris
GCCTAATTACACAAATTATTACAGGACTTTTTTT...
>Cycloramphus_acangatan
GCTTAATTATACAAATTGCACCAGGACTATTCTT...
>Macrogenioglottus_alipioi
GTCACAGGACTCTTCCTTGCAATACACTATACTG...
...
```

```
Select the option desired:  3
```

Resulta em:

```
>FJ685662_Thoropa_miliaris
GCCTAATTACACAAATTATTACAGGACTTTTTTT...
>FJ685663_Cycloramphus_acangatan
GCTTAATTATACAAATTGCACCAGGACTATTCTT...
>FJ685664_Macrogenioglottus_alipioi
GTCACAGGACTCTTCCTTGCAATACACTATACTG...
...
```

A escolha destas opções depende de uma série de fatores. Por exemplo, se sua intenção é concatenar diferentes bases de dados, então você deve escolher a opção que considera apenas o nome dos terminais. Neste caso, assume-se que todas as partições (*i.e.*, distintas regiões genômicas ou diferentes fonte de dados) terão o mesmo nome para cada terminal. No entanto há um problema a considerar. Pode existir no seu banco de dados terminais com o mesmo nome. Neste caso, você terá problemas quando for analisar os dados ou mesmo quando for concatenar as partições – isso é evidente nesse conjunto de dados. A inserção do número de tombo não permite a concatenação da base de dados, pois cada número é único para determinado táxon e determinada região genômica. No entanto há como contornar esse problema, como veremos adiante. Seja qual for sua opção, pense bem em sua estratégia de análise, pois você poderá encontrar problemas à medida em que executa inúmeras tarefas em seu procedimento analítico.

Exercício 7.2

Neste exercício você deverá gerar um arquivo no formato FASTA para cada um dos arquivos que você obteve no Exercício 7.1. Estes arquivos deverão ser criados com a opção **1** de *phyloconvert*, ou seja, os terminais deverão conter o número de tombo. A razão pela qual não iremos incluir os nomes é porque isso geraria terminais com o mesmo nome e isso seria um problema quando você for concatenar os dados. Por outro lado, a adoção dos números de tombo faria com que cada terminal, em cada partição, fosse considerado como um terminal distinto – quando, na realidade, você dispõe de 5 regiões genômicas distintas para a grande maioria dos terminais. Portanto esse exercício deverá ser completado por dois procedimentos distintos. No primeiro deles, você irá usar o *phyloconvert* para fazer com que os nomes das sequências recebam o número de tombo do GenBank (opção 1). Na segunda etapa, você deverá substituir os números de tombo pelos nomes dos táxons na primeira coluna do arquivo `dias_et_al_2013_accession_numbers.csv`. Isso será feito com o auxílio do programa `sed` – que foi apresentado no Tutorial 2. No diretório `tutorial_7` há 5 arquivos de substituição (*_sub.sed). Após usar o *phyloconvert* você deverá executar o `sed` utilizando a seguinte linha de comando, por exemplo:

```
$ sed -f 12s_sub.sed 12s.fas > 12s_renamed.fas
```

A opção `-f` diz ao `sed` que as instruções de substituição estão no arquivo `12s_sub.fas` que serão aplicadas ao arquivo `12s.fas` e o resultado será redirecionado (>) para o arquivo `12s_renamed.fas`.

7.2.2 Verificação e edição FASTA em AliView

Uma vez editado os nomes dos terminais é necessário verificar as sequências nucleotídicas e, via de regra, editar o arquivo. Para isso usaremos o aplicativo chamado AliView [4]. AliView é uma aplicativo muito útil e intuitivo de usar. Há outras ferramentas disponíveis para esse propósito, tais como SeaView e Geneious – entre outros, caso ele não atenda suas necessidades. Para os propósitos de nosso curso, ele será suficiente.

O AliView é iniciado em um terminal. Abra um terminal e execute o comando `aliview`. Após a execução, você deverá obter a janela do programa tal qual na Figura 7.5.

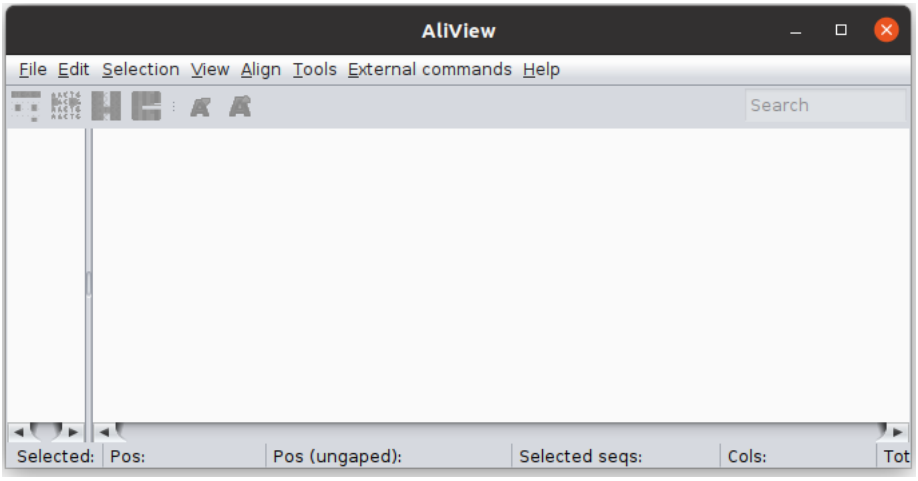


Figura 7.5: Janela inicial do AliView.

Para demonstrar alguns atributos de AliView, eu irei apresentar uma série comandos no programa utilizando o arquivo `exemplo_aliwiew_cyt-b.fas` que encontra-se disponível no diretório `tutorial_07`. Esse arquivo foi gerado a partir das sequências de Citocromo B de Dias *et al.* [1] após eu ter criado um arquivo FASTA (usando a opção 1) utilizando o *script* `phyloconvert`, como vocês fizeram no Exercício 7.1.

No AliView, vá em **File > Open** File e abra o arquivo `exemplo_aliwiew_cyt-b.fas` que está no diretório `tutorial_07`. Uma vez aberto, corra a barra de rolagem horizontal até o final das sequências nucleotídicas como ilustrado na Figura 7.6.

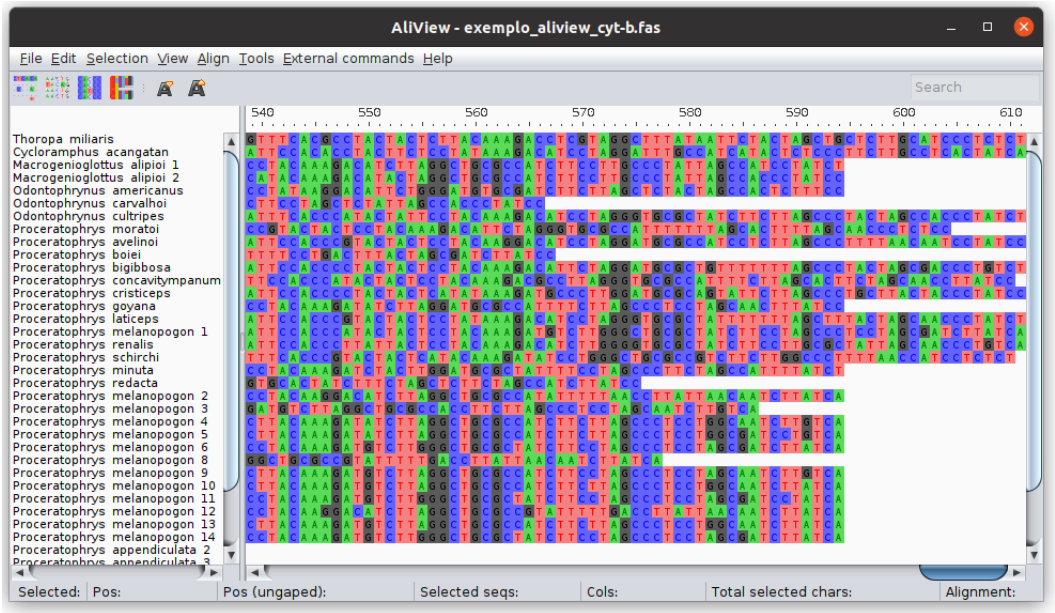


Figura 7.6: Arquivo `exemplo_aliwiew_cyt-b.fas` aberto no AliView.

Você deverá observar que no arquivo `exemplo_aliview_cyt-b.fas` as sequências nucleotídicas diferem de tamanho (Figura 7.6). Estas diferenças se dão por sequenciamento diferencial das amostras. Adicionalmente, o Citocromo B é um gene codificante, onde

raramente ocorrem inserções e deleções e quando ocorrem tendem a atuar sobre códon – portanto três nucleotídios. Para que possamos proceder com análises filogenéticas desses dados é necessário alinhar essas sequências. Alinhamento é o processo pelo qual sequências de tamanhos diferentes são igualadas quanto ao tamanho com a inserção de *gaps* ("-"). Nós iremos discutir detalhadamente algoritmos de alinhamento e a relação entre alinhamento e hipóteses filogenéticas no próximo tutoria. No momento, por se tratar de uma região codificadora, a expectativa é que os *gaps* sejam inseridos no começo e no final de cada sequência.

O alinhamento de sequências em AliView requer inicialmente que todas as sequências sejam selecionadas. Para fazer isso clique em **Selection > Select all**. Uma vez selecionadas, clique em **Align > Realign everything** e clique em OK para iniciar o alinhamento. Na configuração padrão de AliView, o programa de alinhamento é o Muscle – que será discutido no próximo tutorial. Quando o processo de alinhamento terminar – indicado pela mensagem "`Done`", feche a janela `Align with Muscle`. O resultado final deverá ser semelhante ao que é apresentado na Figura 7.7.

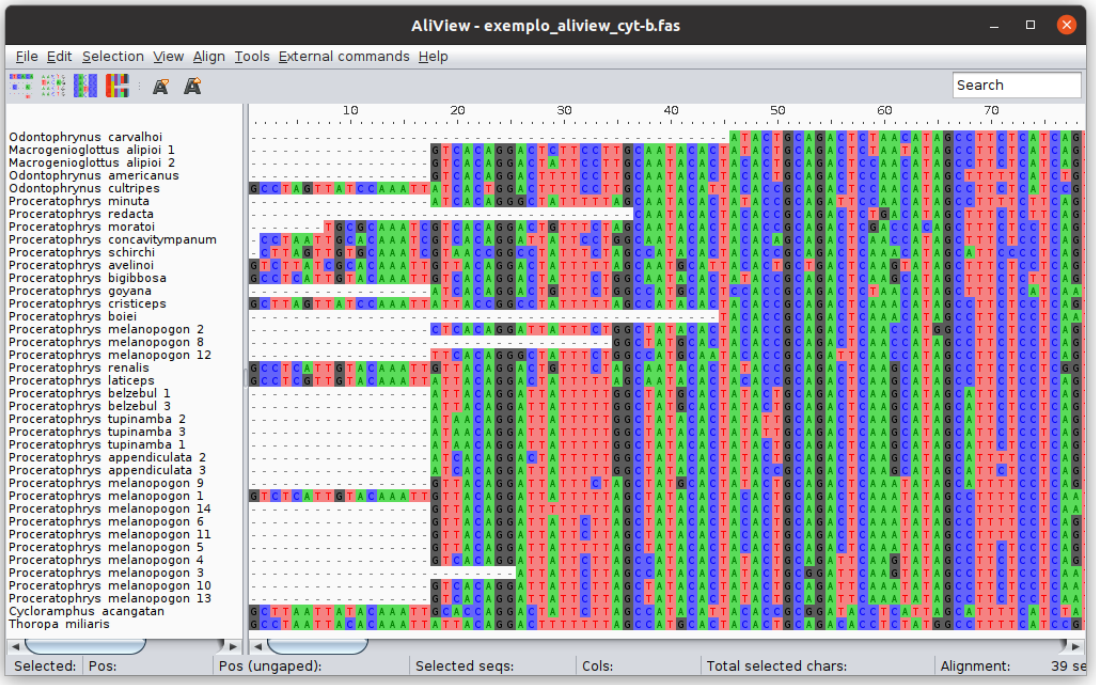


Figura 7.7: Alinhamento via Muscle em AliView das sequências em `exemplo_aliview_cyt-b.fas`.

Observe que após esse procedimento todas as sequências possuem o mesmo comprimento (612 pares de base [bp]). O Muscle inseriu a maioria dos *gaps* na porção inicial das sequências, chamados *leading gaps*, nenhum *gap* no meio das sequências, e algumas sequências com *gaps* no final, estes chamados de *trailling gaps*. Os *gaps* iniciais e finais resultam do processo de sequenciamento diferencial para cada terminal. Se as sequências são mantidas como estão, estes *gaps* podem ser considerados como um quinto estado de caráter (*i.e.*, A, C, G, T e -) quando não deveriam. Desta forma, é necessário editar as regiões iniciais destas sequências. Isso pode ser feito de duas formas. Uma delas é removendo as regiões iniciais das sequências, a outra forma é

substituindo os *gaps* por "Ns". Considere que a substituição de *leading* e *trailling gaps* por "N" insere ambiguidade nos dados, pois "N" é considerado como "?" durante análises filogenéticas. Por outro lado, a remoção destas regiões pode resultar na perda de dados que pode ser importante para resolver a topologia. Por exemplo, há 7 sequências em que há $\tilde{3}00$ pares de base faltantes no final. Isso levou a inserção de muitos *gaps*. Se você optar pela exclusão desta região, você perde a metade dos seus dados aproximadamente. A decisão entre remover ou substituir é arbitrária.

Neste exemplo iremos fazer os dois procedimentos. O primeiro deles é remover os 17 pares de base da região inicial do alinhamento. Para remover esta região no AliView basta você selecionar esta região com o mouse e pressionar `Ctrl+Del`. Você deverá obter o que é ilustrado na Figura7.8.

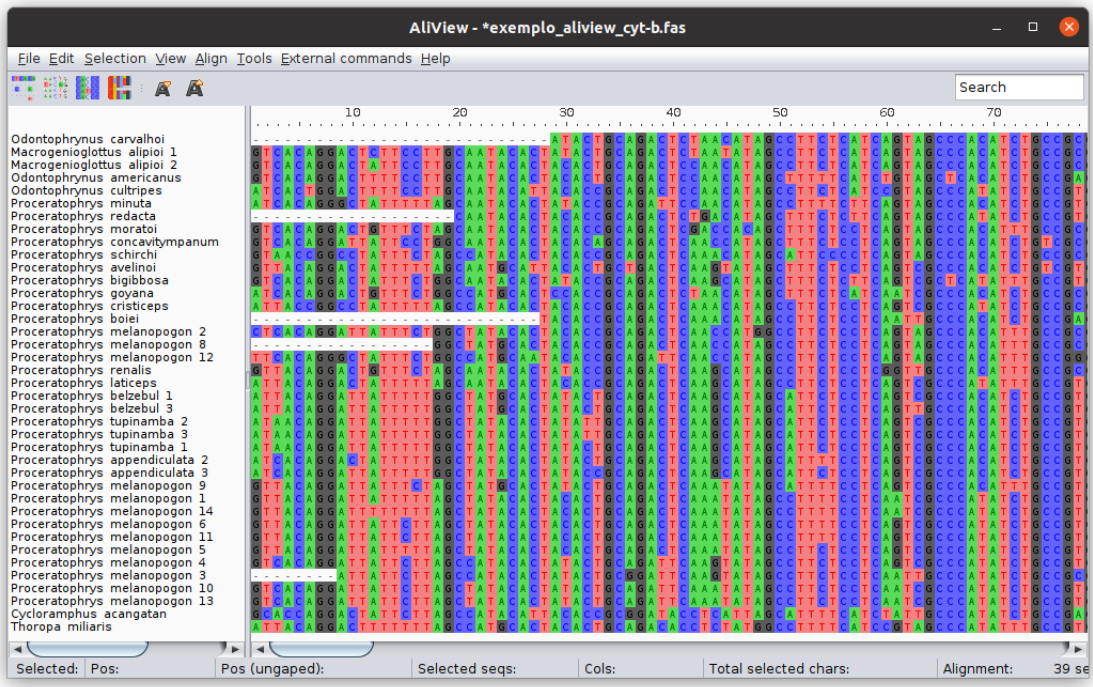


Figura 7.8: Arquivo `exemplo_aliview_cyt-b.fas` com a remoção dos primeiros 17 pares de base.

O segundo passo será transformar os *gaps* iniciais e finais em "Ns". Isso pode ser feito de duas formas. A primeira delas é manualmente, mas isso pode ser indesejável quando o número de sequências é relativamente grande, a outra é usando um *script* que faz isso por você. Vamos usar as duas estratégias. Para fazer isso manualmente na região inicial, basta selecionar a região de *gaps* de uma determinada sequência e pressionar a tecla "n". Você deverá obter o que está ilustrado na Figura 7.9. A segunda forma é utilizar o *script* `subedgesgaps.py` para executar a mesma tarefa, mas agora na região final das sequências nucleotídicas. Para isso, eu vou salvar este alinhamento parcialmente editado em **File > Save as Fasta** sob o nome de `exemplo_aliview_cyt-b_aln_trim.fas` e vou fechar o AliView.

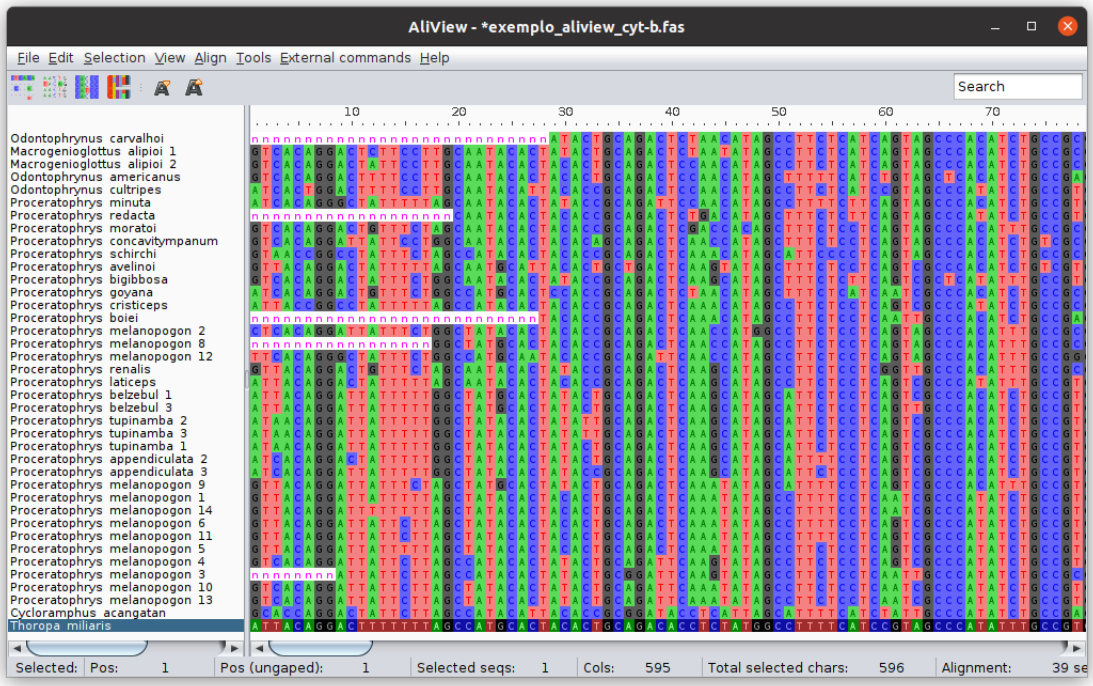


Figura 7.9: Inserção manual de "Ns" na região inicial do alinhamento.

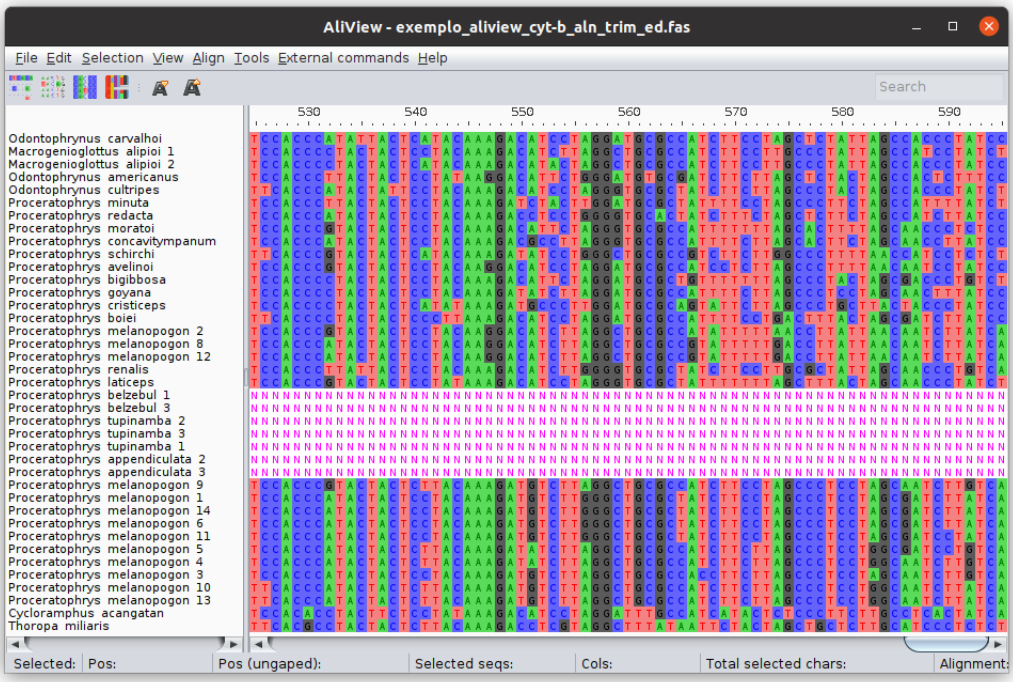


Figura 7.10: Inserção de "Ns" na região final do alinhamento utilizando o *script* `subedgesgaps.py`.

Agora vamos editar a parte final deste alinhamento utilizando o *script* `subedgesgaps.py`. Essa operação é relativamente simples. Em um terminal execute o seguinte comando:

```
$ ./subedgesgaps.pl -i exemplo_aliview_cyt-b_aln_trim.fas >
exemplo_aliview_cyt-b_aln_trim_ed.fas
```

Se abrirmos o arquivo `exemplo_aliview_cyt-b_aln_trim_ed.fas` no AliView e observarmos o final do alinhamento iremos ver que todos aqueles *gaps* terminais foram preenchidos com "`Ns`" como ilustrado na Figura 7.10. Essa estratégia poderia ter sido adotada logo no início, após a deleção dos primeiros 17 pares de base. A operação anterior foi simplesmente para demonstrar que o AliView permite a edição das sequências manualmente.

Recomendo que em cada etapa de edição parcial você salve o arquivo editado sob outro nome. Geralmente eu uso a notação "`_aln`" para arquivos editados, "`_trim`" para arquivos em que eu removi as regiões iniciais eou finais (de *trimmed*) e "`_ed`" quando eu implemento edições. Isso possibilita que você volte a alguma etapa anterior caso detecte algum erro. Seu objetivo é chegar a uma edição final no qual o arquivo está no formato compatível com o programa de inferência filogenética que você deseja utilizar, incluindo como as regiões de inserção e deleção devam ser tratadas. Uma última ponderação sobre o exemplo acima; em alguns casos, o alinhamento apresentará *gaps* internos. Para genes ribossomais, estes devem ser tratados como quinto estado de caráter. Em regiões codificantes, a existência desses *gaps* não é usual e deve ser acompanhada de *triplets* – três *gaps* via de regra.

Exercício 7.3

Para que você se familiarize com o AliView, você deverá editar e salvar as edições de todos os arquivos que você gerou no Exercício 7.1.

7.3 Configuração de matrizes de dados moleculares

Transformar um arquivo FASTA em um arquivo passível de análises filogenéticas é relativamente simples – embora em alguns casos isso seja desnecessário como veremos ao longo desses tutoriais. Como vocês estão familiarizados com o formato de arquivos de entrada para TNT (veja Seção 4.3 do Tutorial 4), iremos explorar como trasformar um arquivo FASTA em formato `xread`.

Considere as seguintes sequências em formato FASTA:

```
>taxon_1
GCCTAATTACACAAATTATT
>taxon_2
GCTTAATTATACAAATTGCA
>taxon_3
GTCACAGGACTCTTCCTTGC
>taxon_4
GTCACAGGACTCATACACTA
```

Neste exemplo temos 4 terminais com sequências de 20 pares de base. A edição deste arquivo para que seja passível de análise filogenética em TNT é simples e pode ser feita manualmente. Veja como seria:

```
nstates dna;
xread
20 4
taxon_1     GCCTAATTACACAAATTATT
taxon_2     GCTTAATTATACAAATTGCA
taxon_3     GTCACAGGACTCTTCCTTGC
taxon_4     GTCACAGGACTCATACACTA;
proc/;
```

ou ainda:

```
xread
20 4
taxon_1     21130033010100033033
taxon_2     21330033030100033210
taxon_3     23101022013133113321
taxon_4     23101022013103010130;
proc/;
```

No primeiro caso, as sequências estão inseridas como vieram do arquivo FASTA e isso requer que a primeira linha do arquivo contenha "`nstates dna;`". No segundo caso, o arquivo é idêntico aos que vocês já viram anteriormente. Neste caso, as bases `A`, `C`, `G` e `T` foram substituídas por `0`, `1`, `2` e `3`, respectivamente. Para arquivos simples como esse, a edição manual é possível, mas pode se tornar inviável à medida em que o número de terminais e/ou de pares de base aumentam. Neste caso, o melhor a fazer é usar *scripts* ou aplicativos concebidos para esse propósito.

Como vocês viram na seção 7.2.1, o *script phyloconvert* possui duas opções que possibilitam a configuração de arquivos para TNT e PAUP a partir de arquivos FASTA (opções 4 e 5, veja acima). Acredito que não terá dificuldades em usar *phyloconvert* para transformar arquivos no formato FASTA em arquivos para TNT e PAUP. Portanto, vamos explorar outra ferramenta.

7.3.1 SequenceMatrix

SequenceMatrix [5] é um aplicativo relativamente simples, mas muito útil para transformar arquivos FASTA em outros formatos. O aplicativo também permite concatenar bancos de dados distintos de forma muito amigável. SequenceMatrix tem uma série de funções que não serão exploradas em detalhe neste tutorial, consulte a página do aplicativo e o trabalho de Vaidya *et al.* [5] para maiores informações. Vejamos como ele funciona considerando as três partições ilustradas na Figura 7.11.

Partição 1
```
>taxon_1
AAAACCCC
>taxon_2
AAA-CCCC
>taxon_3
AAAA-CCC
>taxon_4
AAAACCC-
```

Partição 2
```
>taxon_1
GGGGTTTT
>taxon_2
GGG-TTTT
>taxon_3
GGGG-TTT
>taxon_4
GGGGTTT-
```

Partição 3
```
xread
8 4
taxon_1
00000010
taxon_2
11000001
taxon_3
11110000
taxon_4
11111100
taxon_5
11111110;
proc/;
```

Figura 7.11: Partições de dados utilizadas no exemplo de SequenceMatrix. As duas primeiras partições estão em formato FASTA. A terceira partição é uma arquivo para dados morfológicos no formato `xread` de TNT. Observe que a terceira partição contém um terminal a mais que as demais.

A execução do `SequenceMatrix` é feita do terminal da sequinte forma:

```
$ sequencematrix
```

Ao executá-lo você deverá obter a sequinte janela:

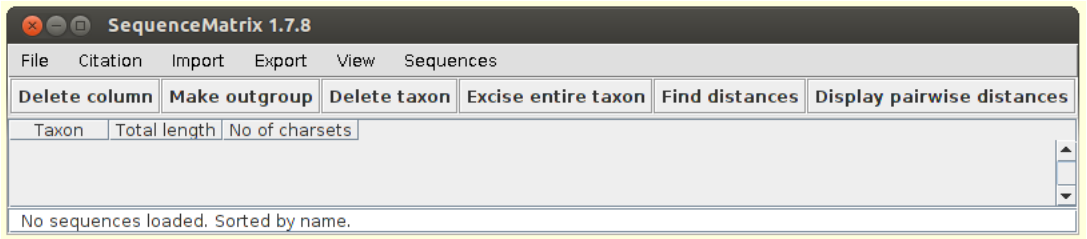


Figura 7.12: Janela de abertura de SequenceMatrix.

Para importar partições de dados basta selecionar `Import/Add sequence` no menu principal. No exemplo abaixo (Figura 7.13), eu adicionei a primeira partição que estava em um arquivo chamado `sequencematrix_1.fas`.

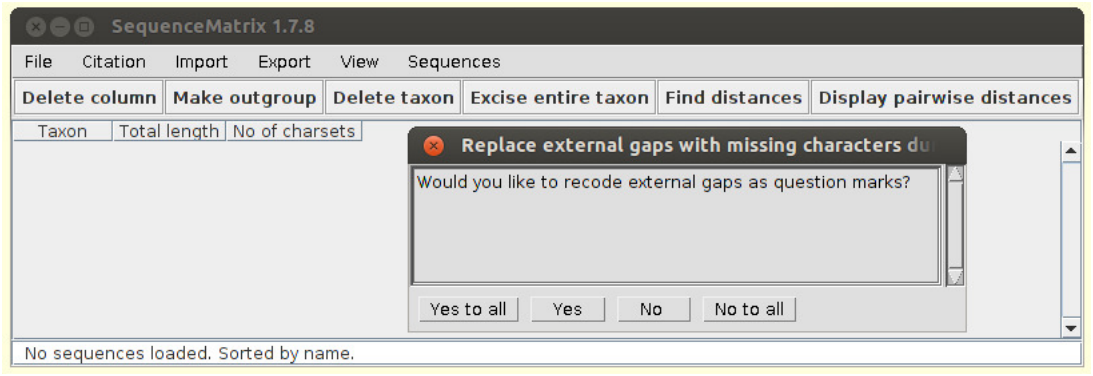


Figura 7.13: Janela de importação de SequenceMatrix.

SequenceMatrix irá lhe perguntar se você deseja substituir os *gaps* por `?`. Você deverá sempre selecionar *No to all*, a não ser que você realmente queira tratar eventos de INDELs como *missing data*. Em breve iremos discutir esse aspecto da análise filogenética da dados moleculares.

Em alguns casos, o programa poderá lhe perguntar se você deseja usar o nome do táxon ou da sequência (Figura 7.14). Você deverá optar pelo nome da sequência, caso contrário, o programa gerará – neste caso em particular – terminais com o mesmo nome. Isso lhe traria proplemas mais adiante. Minha sugestão é que você configure os nomes dos terminais (ou sequências) da forma mais simples e informativa possível nesse estágio de sua análise – antes de concatená-los. Lembre-se que você poderá substituir esses nomes no final, quando for gerar figuras e ou avaliar resultados. Quanto mais simples os nomes são, mais fácil se torna a tarefa de gerenciar arquivos e evitar erros, principalmente se você está lidando com várias bases de dados para os mesmos terminais

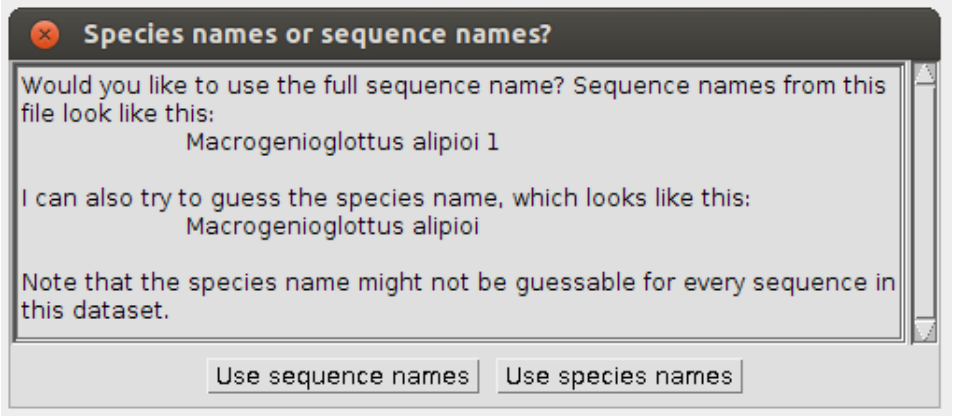


Figura 7.14: Janela de de seleção de nomes em SequenceMatrix.

Ao importar as sequências, você deverá obter resultado semelhante ao ilustrado na Figura 7.15. SequenceMatrix informa alguns detalhes sobre os dados importados, tais como número de terminais, tamanho das sequências e presença de INDELs.

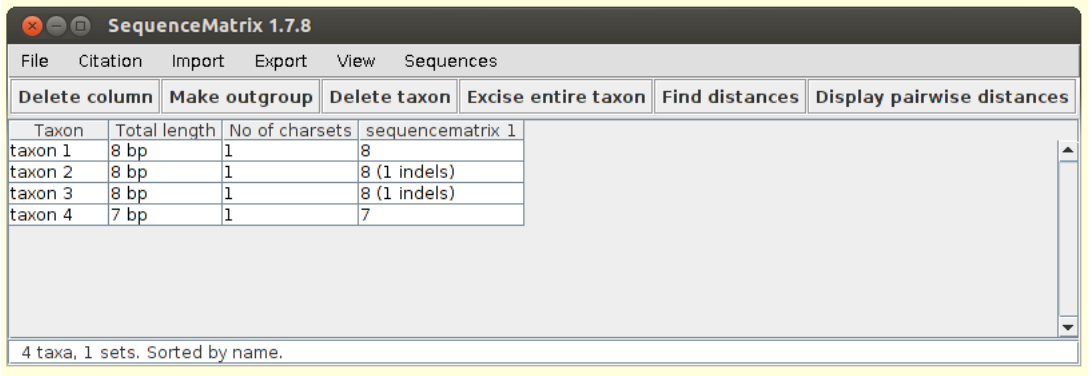


Figura 7.15: Sequências importadas em SequenceMatrix.

Para importar todas as partições da Figura 7.11 basta repetir a tarefa. SequenceMatrix importa matrizes de TNT da mesma forma que importa sequências em formato FASTA. A importação de todas as partições é ilustrada na figura seguinte (Figura 7.16):

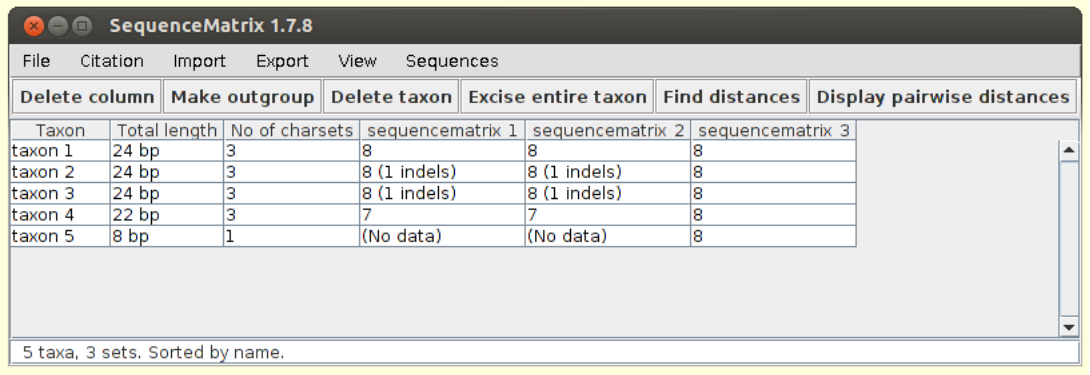


Figura 7.16: Partições importadas em SequenceMatrix.

Para exportar esses dados, basta selecionar uma das opções disponíveis em `Export` do menu principal. No exemplo abaixo eu selecionei a opção `Export/Export sequences as TNT`. O arquivo de exportação terá a seguinte configuração:

```
nstates dna;
xread
'Exported by SequenceMatrix 1.7.8 on Tue Apr 22 09:39:44 BRT 2014.'
24 5
taxon_1 AAAACCCCGGGGTTTT00000001
taxon_2 AAA-CCCCGGG-TTTT11000000
taxon_3 AAAA-CCCGGGG-TTT11110000
taxon_4 AAAACCC-GGGGTTT-11111100
taxon_5 ????????????????11111110;

xgroup
=0 (sequencematrix_1) 0 1 2 3 4 5 6 7
=1 (sequencematrix_2) 8 9 10 11 12 13 14 15
=2 (sequencematrix_3) 16 17 18 19 20 21 22 23
;
```

```
agroup
=0 (CHARSETS_ATLEAST_3) 0 1 2 3
=1 (CHARSETS_ATLEAST_2) 0 1 2 3
=2 (TAXA_HAVING_sequencematrix_1) 0 1 2 3
=3 (TAXA_HAVING_sequencematrix_2) 0 1 2 3
=4 (TAXA_HAVING_sequencematrix_3) 0 1 2 3 4
;
```

No exemplo acima, a matrix de dados contempla todas as partições e insere pontos de interrogação para os caracteres das duas primeira partições nas quais o `taxon_5` não está representado. Há dois blocos adicionais neste arquivo. O primeiro deles é o `xgroup` que define grupos de caracteres. O segundo é o `agroup` que define grupos de táxons em TNT. Esses blocos são utilizados em TNT para manipular grupos como um todo – para maiores detalhes veja documentação de TNT.

Exercício 7.4

Neste exercício você deverá transformar os arquivos que gerou no Exercício 7.3 em arquivos no formato de TNT. As tarefas específicas a serem realizadas são as sequintes:

i. Gerar arquivos no formato TNT para cada partição de Dias *et al.* [1].

ii. Gerar um arquivo no formato TNT no qual todas as partições estão representadas.

iii. Analisar estes arquivos em TNT e fazer um resumo abaixo no qual você compara os resultados que você obteve com a topologia apresentada na Figura 11 (pg. 294) de Dias *et al.* [1].

7.4 Referências

1. Dias, P. H. S.; Amaro, R. C.; de Carvalho-e Silva, A. M.P. T. & Rodrigues, M. T. 2013. Two new species of *Proceratophrys* Miranda-Ribeiro, 1920 (Anura; Odontophrynidae) from the Atlantic forest, with taxonomic remarks on the genus. *Zootaxa* **3682**: 277–304.
2. Goloboff, P.; Farris, J. S. & Nixon, K. 2008. TNT a free program for phylogenetic analysis. *Cladistics* **24**: 1–14.
3. Swofford, D. 2003–2016. PAUP*. Phylogenetic Analysis Using Parsimony (*and Other Methods, Version 4.0a131). Sunderland, Massachusetts: Sinauer Associates, 2003–2016.
4. Larsson, A. 2014. AliView: a fast and lightweight alignment viewer and editor for large data sets. *Bioinformatics* **30**(22): 3276–3278.
5. Vaidya, G.; Lohman, D. J. & Meier, R. 2010. SequenceMatrix: concatenation software for the fast assembly of multi-gene datasets with character set and codon information. *Cladistics* **27**: 171–180.

Tutorial 8

Dados Moleculares - Alinhamento

BIZ0433 - INFERÊNCIA FILOGENÉTICA: FILOSOFIA, MÉTODO E APLICAÇÕES.

Conteúdo

Objetivo

O objetivo deste tutorial é apresentar os conceitos associados ao alinhamento múltiplo de sequências nucleotídicas e subsequente análise filogenética. Neste tutorial iremos avaliar os resultados de três programas tradicionalmente utilizados para produzir alinhamentos (Clustalw, Clustal–Ω, Muscle e MAFFT) e alguns dos componentes analíticos comuns a estes programas. Este tutorial requer a execução de uma série de exercícios e a avaliação dos resultados obtidos frente a leitura do artigo de Phillips *et al.* (2000). Ao completar este tutorial, o estudante terá uma visão geral dos algoritmos e estratégias de alinhamento múltiplo disponíveis para análises filogenéticas dentro do conceito de homologia estática, bem como dos problemas associados a esta prática em inferência filogenética. 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/
```

8.1 Contextualização

A maioria das análises filogenéticas convencionais baseadas em dados moleculares requer um esquema de correspondência entre os pares de base que antecede à buscas de topologias – ou seja, a análise filogenética propriamente dita [1]. Isso decorre da observação de que, em muitos casos, regiões homólogas do genoma diferem em tamanho em consequência da composição diferencial de pares de base. O processo de conversão entre sequências de tamanhos distintos em sequências de mesmo tamanho é chamado de *alinhamento* ou ainda *alinhamento múltiplo*. Considere por exemplo a seguinte observação:

```
taxon1      TTGCA
taxon2      TGGGCCA
taxon3      TGCAA
taxon4      TGGCCA
```

Um possível esquema de correspondência (*i.e.*, alinhamento) entre os pares de bases das sequências nucleotídica para estes quatro terminais seria:

```
Alinhamento 1
taxon1      TTG--CA
taxon2      TGGGCCA
taxon3      TG--CAA
taxon4      TGG-CTA
```

ou poderia ser:

```
Alinhamento 2
taxon1      T-TG-CA
taxon2      TGGGCCA
taxon3      TG--CAA
taxon4      T-GGCTA
```

Observe que estes dois alinhamentos propõem diferentes séries de transformações para os caracteres 2 e 3, portanto assumem hipóteses de identidades históricas [senso 2] distintas. Em consequência disso, estes dois alinhamentos resultam em hipóteses filogenéticas distintas (Figura 8.1). A influência dos alinhamentos em hipóteses filogenéticas é conhecido há tempos. Morrison & Ellis [3] examinaram esta relação e concluíram que o resultado de análises filogenéticas são mais influenciadas por alinhamentos do que critérios de otimalidade.

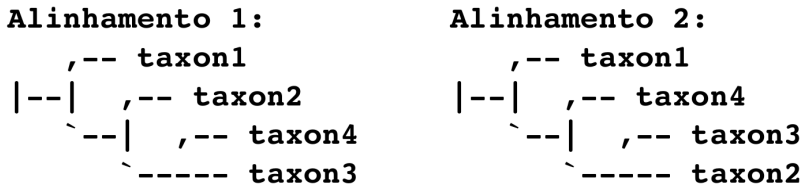


Figura 8.1: Hipóteses filogenéticas mais parcimoniosas resultantes dos alinhamentos 1 e 2, acima.

Um dos conceitos mais importantes relacionados aos procedimentos de alinhamento é compreendê-lo como um processo de inferência similar às estimativas de hipóteses filogenéticas. Pares de bases (*i.e.*, `A`, `C`, `G` e `T`) são observações, mesmo que indiretamente apresentadas pelas plataformas de sequenciamento. Os *gaps*, por outro lado, são inseridos durante o processo de alinhamento para traçar as correspondências necessárias – dentro do contexto de homologia estática [4] – para proceder com a análise filogenética. Igualmente importante é reconhecer que a inserção de *gaps* postulam eventos de inserção e deleções (*i.e.*, **INDELs**), eventos históricos que consequentemente possuem informação filogenética. Isso é relevante, pois requer que *gaps* sejam tratados como um quinto estado de caráter em análises filogenéticas de sequências nucleotídicas.

Neste tutorial iremos explorar os principais conceitos associados ao procedimento de alinhamento de sequências nucleotídicas. O tutorial seguirá um formato distinto do que vocês estão acostumados até então. Durante o tutorial vocês farão uma série de exercícios cujo objetivo é apresentar a vocês algumas das ferramentas disponíveis para alinhamentos múltiplos, entender os componentes metodológicos associados aos alinhamentos e verificar como as premissas de alinhamento impactam os resultados de análises filogenéticas. No final deste tutorial você encontrará uma série de perguntas que deverão ser respondidas para a próxima aula em referências ao artigo de Phillips *et al.* [5].

8.2 Alinhamento, aplicativo e topologias

8.2.1 Alinhamento manual no AliView

Não é incomum encontrarmos publicações nas quais o alinhamento foi efetuado integralmente ou parcialmente à mão. Os chamados alinhamentos "*by eye*" são menos frequentes atualmente, isso é fato, mas seus defensores argumentavam que nosso cérebro era capaz de tomar decisões mais prudentes que os algoritmos disponíveis para essa tarefa. Há uma série de problemas com essa prática que iremos discutir na próxima aula [veja 5].

Exercício 8.1

Você deverá fazer o alinhamento manual das sequências no arquivo `seqdata1.fas` utilizando AliView. Em AliView, para inserir um gap em determinada região, basta posicionar o cursor na região desejada e pressionar a tecla de espaço. Note que o Aliview tende a inserir gaps no final da sequência que deverão ser removidos ao final de seu alinhamento manual. É importante que você tente fazer o melhor alinhamento que considerar possível – embora devemos reconhecer que a noção de "melhor" é extremamente subjetiva –, mas gaste alguns bons minutos ponderando as decisões que o levaram a inserir um *gap* em determinadas regiões. Após finalizar este alinhamento você deverá salvá-lo sob o nome de "`seqdata1_man.fas`".

8.2.2 CLUSTALW

Um dos aplicativos mais utilizados para alinhamento múltiplo é o Clustalw [6]. A documentação do programa pode ser encontrada em `http://www.clustal.org/clustal2/`. Não é o objetivo deste tutorial fazer qualquer explanação sobre os algoritmos e estratégias utilizadas pelo Clustalw, mas em Phillips *et al.* [5] você encontrará parte dessas informações. O Clustalw está disponível no sistema de vocês. Caso esteja usando uma outra instalação ou sistema operacional, certifique-se que o programa está instalado.

Exercício 8.2

Neste exercício você deverá fazer um alinhamento das sequências do arquivo `seqdata1.fas` utilizando Clustalw. Para fazê-lo, você deverá executar o Clustalw em um terminal utilizando a seguinte linha de comando:

```
$ clustalw -align -output=fasta -outorder=input -infile=seqdata1.fas
-outfile=seqdata1_cltw.fas
```

Esta linha de comando, a opção "`-align`" faz com que Clustalw execute o alinhamento múltiplo para as sequências existentes em "`-infile=seqdata1.fas`" obedecendo a ordem dos terminais ("`-outorder=input`") quando imprimir os resultados no arquivo de saída "`-outfile=seqdata1_clt.fas`" no formato "`-output=fasta`".

8.2.3 CLUSTAL–Ω

Recentemente, Sievers *et al.* [7] desenvolveram uma nova versão de Clustal. Chamada de Clusta–Omega (Ω), esta versão foi considerada mais rápida e "acurada" que Clustalw. Dada à sua recente publicação, está versão não é tão popular como a anterior, mas pode ser que isso mude ao longo dos próximos anos.

Exercício 8.3

Neste exercício você deverá fazer um alinhamento das sequências do arquivo `seqdata1.fas` utilizando Clustal-Ω. Para fazê-lo, você deverá executar o Clustal-Ω em um terminal utilizando a seguinte linha de comando:

```
$  clustalo --full --iter=16 -i seqdata1.fas -o seqdata1_clto.fas -v
```

A descrição destas opções pode ser verificada digitando "`clustalo --help`" em um terminal ou consultando a doscumentação do programa (Doumentação).

8.2.4 MUSCLE

Muscle [8] é outro aplicativo utilizado para alinhamentos múltiplos. Em termos gerais, esse

programa é bem mais rápido que Clustalw, embora menos utilizado na literatura. A forma de alinhamento de Muscle difere razoavelmente de Clustalw. Uma breve explanação sobre estas diferenças é apresentada no item 4.1 da documentação de Muscle e não será discutida aqui.

Exercício 8.4

Neste exercício você irá alinhar as mesmas sequências que alinhou com o Clustalw e Clustal–Ω utilizando Muscle. Você irá executar o Muscle utilizando seus parâmetros padrão com a seguinte linha de comando:

```
$ muscle -in seqdata1.fas -out seqdata1_mus.fas
```

Na linha de comando acima, o arquivo de entrada (`-in seqdata1.fas`) será alinhado no Muscle e o resultado impresso no arquivo de saída (`-out seqdata1_mus.fas`). No entanto, no arquivo `-out seqdata1_mus.fas` a ordem dos terminais está alterada (veja utilizando o comando "`less`"). Há uma opção em Muscle para manter a ordem dos terminais idêntica ao arquivo de entrada, porém esta opção (`-stable`) está com problemas e foi desabilitada nas versões mais recentes do programa. No entanto, o autor disponibilizou um *script* que reordena o alinhamento segundo o arquivo de entrada. Para fazer o mesmo com seus resultados, execute o seguinte comando de linha:

```
$ python ./stable.py seqdata1.fas seqdata1_mus.fas > seqdata1_mus_ord.fas
```

8.2.5 MAFFT

MAFFT [9, 10] é o último programa que iremos utilizar. Este é um dos programas mais recentes desenvolvidos e possui uma série de algoritmos muito eficientes para gerar alinhamentos. Detalhes sobre tais algoritmos podem ser encontrados nas referências citadas anteriormente ou na página do programa referente aos algoritmos de alinhamento.

Exercício 8.5

Neste exercício, você deverá fazer um novo alinhamento das sequências no arquivo "`seqdata1.fas`". Para isso, você deverá executar o MAFFT com a seguinte linha de comando:

```
$ mafft --maxiterate 1000 --preservecase seqdata1.fas > seqdata1_mafft.fas
```

Na linha de comando acima, o termo "`--maxiterate 1000`" configura o número máximo de iterações que o MAFFT irá usar durante o processo de refinamento do alinhamento das sequências no arquivo de entrada "`seqdata1.fas`" cujo resultado será impresso no arquivo de saída "`seqdata1_mafft.fas`" preservando as letras maiúsculas dos pares de base de acordo com a

opção "`--preservecase`"[1].

8.2.6 AVALIAÇÃO DOS RESULTADOS

Ao completarem os exercícios acima, seu diretório de trabalho deverá conter os seguintes aquivos: `seqdata1_man.fas`, `seqdata1_cltw.fas`, `seqdata1_clto.fas`, `seqdata1_mus_ord.fas` e `seqdata1_mafft.fas`. Verifique se todos estão presentes e que tenham conteúdo, caso contrário será necessário repetir alguns dos exercícios acima.

Exercício 8.6

Sua primeira avaliação será puramente visual. Abra todos os arquivos simultaneamente em AliView e responda:

i. Todos os alinhamentos são idênticos?

ii. Se você tivesse que optar por um desses alinhamentos, qual seria? Justifique sua resposta.

iii. Abaixo você deverá fazer uma análise filogenética em TNT de todos os alinhamentos gerados, completar a Tabela 8.1 e anotar o relacionamento filogenético encontrado em cada uma das análises nos espaços específicos abaixo. Para transformar esses arquivos no formato FASTA em formato compatível com arquivos de entrada no TNT (`xread`) você poderá usar o SequenceMatrix (veja Tutorial 7) ou você poderá usar o *script* `fasta2tnt.py` com a seguinte linha de comando (exemplo):

```
$ ./fasta2tnt.py seqdata1_mafft.fas > seqdata1_mafft.tnt
```

Tabela 8.1: Análise cladística dos alinhamentos gerados por diferentes programas.

Alinhamento	Número de MPTs	Custo das MPTs
manual		
Clustalw		
Clustal-Ω		
Muscle		
MAFFT		

[1] na configuração padrão do MAFFT ele imprime os pares de base em letras minúsculas. Embora para efeitos analíticos isso não importe, letras maiúsculas são desejáveis quando se deseja inspecionar os alinhamentos em programas como AliView.

Topologias encontradas nas análises filogenéticas dos alinhamentos da Tabela 8.1:

8.3 Parâmetros de alinhamento

Todos os alinhamentos produzidos pelos programas acima baseiam-se em alinhamentos par a par computados pelo algoritmo de Needleman & Wunsch [11]. Este algoritmo está documentado no artigo de Phillips *et al.* [5] e depende de parâmetros numéricos que são utilizados no cômputo das distâncias de edição entre duas sequências alinhadas. A distância de edição entre duas sequências é definida pelo número de operações necessárias para tornar uma sequência idêntica a outra. Por exemplo, considere o seguinte alinhamento:

```
taxon1      TATG--A
              ** **
taxon2      TGGGCCA
```

Observe que são necessárias 4 operações para tornar uma das sequências igual a outra (denotadas com "$\star$"). Por exemplo, se considerarmos a sequência do `taxon1` e substituirmos **A** por **G** na segunda posição, **T** por **G** na terceira posição e **-** por **C** na quinta e sexta posições, as sequências de ambos os terminais tornam-se idênticas. Neste caso, se o custo de cada evento de edição é atribuído a 1, o custo deste alinhamento seria 4.

Parâmetros de alinhamento podem considerar custos distintos para eventos de edição diferentes – na expectativa de acomodar algumas premissas associadas ao que sabemos sobre evolução molecular. No exemplo acima, a edição necessária na segunda posição envolve um evento de transição (*i.e.*, substituições entre purinas ou pirimidinas), ao passo que na terceira posição o evento de edição envolve uma transversão (*i.e.*, substituições de purina por uma pirimidina, ou vice versa). Em casos como esse, você poderia atribuir custos diferenciais para esses dois tipos de operações de edição. Você poderia ainda levar em consideração que o custo de um *gap* é diferente de dois *gaps* consecutivos. A justificativa para estes custos diferenciais associados à abertura e extensão de *gaps* está no argumento de que INDELs (*i.e.*, inserções e deleções) podem ocorrer em bloco [mas veja 5, 12].

O cálculo do custo de um alinhamento é necessário por que a função objetiva que cada programa está buscando otimizar depende dele. Todos os programas exemplificados acima tentam maximizar a semelhança entre as sequências alinhadas e, portanto, sua função objetiva minimiza a distância de edição entre elas. Considere o seguinte alinhamento:

```
taxon1      TATG----A
              ** ****
taxon2      T--GGGCCA
```

Neste caso, o custo do alinhamento seria 6, maior que o anterior. Mas isso só é verdade se assumirmos que qualquer operação de edição possui o mesmo custo, no caso 1. No entanto, considere custos diferenciais para abertura e extensão de *gap*, no qual o primeiro *gap* recebe valor igual a 1 e os demais 0.5. Neste caso ambos alinhamentos teriam custo igual a 3,5. Adicionalmente, consideremos custos diferenciais para transições e transversões, por exemplo

1 e 2, respectivamente. Neste caso, o primeiro exemplo teria custo igual a 4,5 ao passo que o segundo permaneceria com custo igual a 3,5 – pois nenhuma operação de edição neste último alinhamento requer substituição entre pares de base. O que fizemos acima foi variar os chamados **parâmetros de custo** de alinhamentos, e como se observa, eles influenciam o resultado da função objetiva e portanto pode selecionar alinhamentos distintos.

8.3.1 Manipulação de parâmetros em MAFFT

Para verificar as relações entre parâmetros de alinhamento e inferência filogenética iremos manipular dois parâmetros disponíveis em MAFFT para 4 alinhamentos distintos e posteriormente fazer uma análise cladística destes alinhamentos.

Exercicio 8.7

O parâmetro "`--op`" define o que é conhecido como *gap open penalty* em programas de alinhamento e o parâmetro "`--ep`" define o custo de extensão de *gaps*. Em MAFFT, esses parâmetros recebem os valores de 1.53 e 0.123, respectivamente. Obedecendo a sintaxe das linhas de comando abaixo, você deverá produzir 4 alinhamentos distintos, sendo que para um deles será considerado os valores padrão para estes parâmetros. Para os demais serão atribuídos custos diferentes, porém idênticos para "`--op`" e "`--ep`". Os comandos de linha para a execução destes alinhamentos são as seguintes:

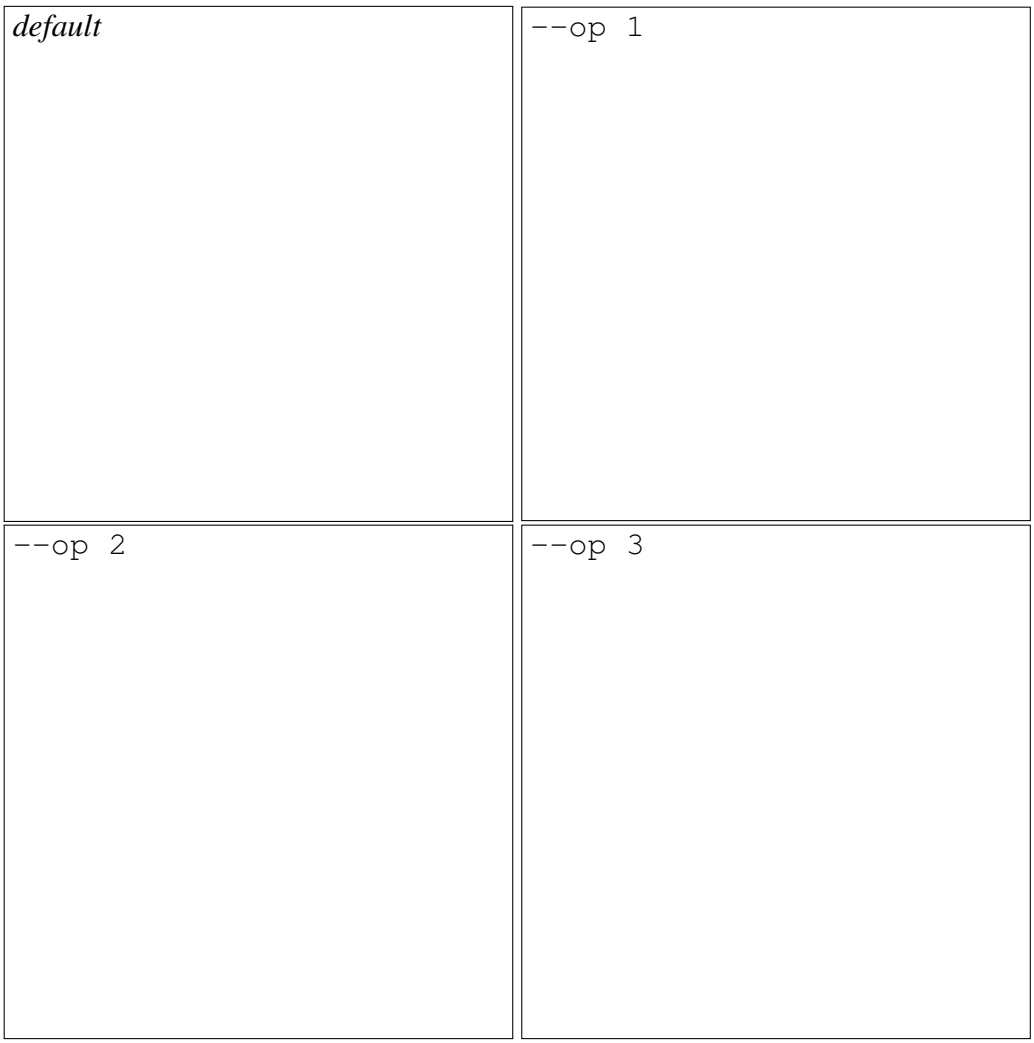
```
$ mafft --maxiterate 1000 --preservecase seqdata2.fas > seqdata2.d.fas
$ mafft --op 1 --ep 1 --maxiterate 1000 --preservecase seqdata2.fas > seqdata2.1.fas
$ mafft --op 2 --ep 2 --maxiterate 1000 --preservecase seqdata2.fas > seqdata2.2.fas
$ mafft --op 3 --ep 3 --maxiterate 1000 --preservecase seqdata2.fas > seqdata2.3.fas
```

Após gerar estes alinhamentos, você deverá fazer uma análise filogenética destes dados alinhados e anotar os resultados na Tabela 8.2 e no campo disponível para ilustrar as topologias obtidas, abaixo

Tabela 8.2: Efeito de parâmetros de alinhamento em inferência filogenética para as sequências do arquivo `seqdata2.fas`.

Alinhamento	Número de MPTs	Custo das MPTs
default		
`--op 1`		
`--op 2`		
`--op 3`		

Topologias encontradas nas análises filogenéticas sob diferentes parâmetros de alinhamento da Tabela 8.2:



8.4 Alinhamento progressivo e árvores-guias

O algoritmo de Needleman-Wunsch [11], utilizado pela maioria dos aplicativos de alinhamento múltiplo, foi concebido para alinhar pares de sequências. Desta forma, ele é bidimensional. Por esta razão, para que os chamados alinhamentos múltiplos sejam executados pelo alinhamento par a par das sequências envolvidas é necessário definir quais seriam os pares alinhados para que ao final tivéssemos um alinhamento completo de todas as sequências de interesse. Todos

os programas selecionam pares, sejam eles de táxons terminais (OTUs) ou de vértices internos (HTUs) de acordo com o que denominamos de árvore-guia (*i.e., guide trees*). Estas árvores-guias são geradas, via de regra, na fase inicial do alinhamento [consulte 5] e determinam a ordem progressiva dos pares que serão submetidos ao algoritmo de Needleman-Wunsch. O resultado final é o chamado "alinhamento múltiplo".

8.4.1 Examinando a influência de diferentes árvores-guias em alinhamentos múltiplos

Neste componente do tutorial iremos examinar como árvores-guias influenciam os alinhamentos. Iremos considerar quatro árvores-guias para as sequências do arquivo `seqdata2.fas`[2] (Figura 8.2).

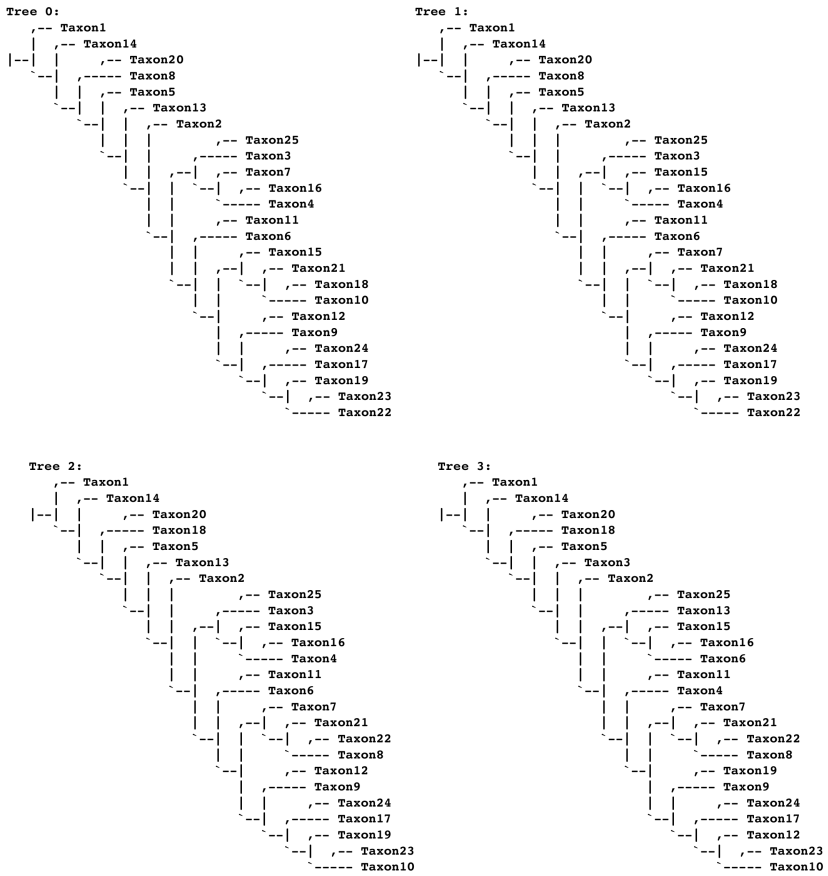


Figura 8.2: Quatro árvores-guias utilizadas no Exercício 8.8 para verificar o efeito dessas topologias nos alinhamentos múltiplos. Estas topologias estão nos arquivos `new_guide_*.tre` e foram transformados para o formato compatível com MAFFT nos arquivos `new_guide_tree*.mafft`.

[2] os comandos de linha para obtenção e controle da árvores-guias em MAFFT estão disponíveis em http://mafft.cbrc.jp/alignment/software/treein.html e http://mafft.cbrc.jp/alignment/software/treeout.html.

Exercicio 8.8

Você deverá executar as linhas de comando abaixo, cada uma delas irá alinhar as sequências do arquivo `seqdata2.fas` considerando cada uma das árvores-guias disponíveis nos arquivos `new_guide_tree*.mafft`:

```
$ mafft --maxiterate 1000 --preservecase seqdata2.fas > seqdata2.d.fas
$ mafft --op 1 --ep 1 --treein new_guide_tree0.mafft --maxiterate 1000 --preservecase
seqdata2.fas > seqdata2.1.0.fas
$ mafft --op 1 --ep 1 --treein new_guide_tree1.mafft --maxiterate 1000 --preservecase
seqdata2.fas > seqdata2.1.1.fas
$ mafft --op 1 --ep 1 --treein new_guide_tree2.mafft --maxiterate 1000 --preservecase
seqdata2.fas > seqdata2.1.2.fas
$ mafft --op 1 --ep 1 --treein new_guide_tree3.mafft --maxiterate 1000 --preservecase
seqdata2.fas > seqdata2.1.3.fas
```

Após gerar estes alinhamentos, você deverá fazer uma análise filogenética destes dados alinhados e anotar os resultados na Tabela 8.3 e no campo disponível para ilustrar as topologias obtidas, abaixo.

Tabela 8.3: Efeito de árvore-guia em inferência filogenética para as sequências do arquivo `seqdata2.fas`.

Árvore-guia	Número de MPTs	Custo das MPTs
`new_guide_0.tre`		
`new_guide_1.tre`		
`new_guide_2.tre`		
`new_guide_3.tre`		

Topologias encontradas nas análises filogenéticas utilizando diferentes árvores-guias:

new_guide_0.tre	new_guide_1.tre
new_guide_2.tre	new_guide_3.tre

8.5 Consistência interna da análise

As análise executadas até o momento sofrem de pelo menos duas inconsistências analíticas severas. A primeira delas está relacionada ao fato de que os algoritmos utilizados para criar as árvores-guias possuem uma função objetiva completamente diferente daquela que você utilizou nas análises filogenéticas. Todas as árvores-guias geradas pelos programas de alinhamento múltiplo que utilizamos até o momento são geradas por algoritmos fenéticos, para os quais a função objetiva minimiza a distância fenética entre as sequências. Desta forma, árvores-

guias em Clustalw, Clustal-Ω, Muscle e MAFFT são fenogramas. No entanto, ao analisarmos os alinhamentos em TNT, nós abandonamos esse critério de otimalidade (distância fenética) e adotamos outro (distância patrística de série de transformações por parcimônia)! Não seria prudente epistemologicamente manter a consistência da função objetiva, e consequentemente critério de otimalidade, ao longo de toda a análise? A segunda inconsistência analítica refere-se ao fato de que os parâmetros dos parâmetros de custos de alinhamento, escolhidos de arbitrariamente – diga-se de passagem –, foram ignorados durante a etapa de inferência filogenética. Desta forma, você utilizou uma série de premissas para gerar os dados que foram desconsideradas no memento em que você utilizou estes mesmos dados para inferir relações de parentesco. O MAFFT, por exemplo, utiliza uma razão de 2:1 para transversões e transições que não foi considerada quando você fez uma análise cladística daquele alinhamento. O mesmo pode ser dito para todas as análises do Exercício 8.7. Em todos esses casos, as premissas analíticas não foram consistentes ao longo de todas as etapas de sua análise.

8.5.1 Consistência de premissas de alinhamento

Quando se adota programas como Clustalw, Clustal-Ω, Muscle e MAFFT não é possível manter consistência dos critérios de otimização – assumindo que sua análise filogenética não utilize métodos fenéticos. Se seu critério de otimalidade em inferência filogenética é parcimônia, é necessário utilizar programas como, por exemplo, MALIGN [13] para manter a consistência interna da análise. Seja como for, o que iremos fazer a seguir é minimizar estas inconsistências analíticas implementando as premissas de custo de alinhamento em análises filogenéticas.

Para manter as premissas de alinhamento durante a análise filogenética é preciso implementar as mesmas funções de custo atribuídas para os abertura e extensão de *gaps* e também custos diferenciais de transformação durante o alinhamento na análise filogenética. MAFFT atribui penalidades distintas para transversões e transições na razão de 2:1, respectivamente. Nas análises do Exercício 8.7, nós atribuímos custos diferentes dos parâmetros padrão do programa para os *gaps*. A implementação dessas premissas em TNT é feita por matrizes de Sankoff. Consulte o a seção 6.1 do Tutorial 6 para relembrar como caracteres de Sankoff são implementados em TNT.

Exercício 8.9

Sua tarefa é implementar as premissas de alinhamento na reanálise das sequências dos arquivos `seqdata2.2.fas` e `seqdata2.3.fas` produzidas no Exercício 8.7 deste tutorial. Os resultados dessas análises devem ser compilados na Tabela 8.4 e nos campos abaixo.

Tabela 8.4: Implementação dos parâmetros de alinhamento em análises filogenéticas.

Dados	Número de MPTs	Custo das MPTs
`seqdata2.2.fas`		
`seqdata2.3.fas`		

Topologias encontradas nas análises filogenéticas para as quais você implementou as premissas de alinhamento durante a análise filogenética:

`seqdata2.2.fas`	`seqdata2.3.fas`

8.6 Questões

1. Você tem alguma objeção ao uso de alinhamentos manuais? Comente.

2. Os resultados dos exercícios da seção 8.2 lhe permitem identificar algum critério objetivo que possa ser usado para a seleção de algum dos alinhamentos que você produziu? Justifique.

3. Parâmetros de alinhamento influenciam reconstruções filogenéticas? Comente de acordo com os resultados que você obteve nos exercícios acima.

4. Você considera que esses parâmetros de alinhamento são arbitrários? Justifique.

5. Árvores-guias influenciam o resultado final do alinhamento? Comente de acordo com os resultados que você obteve nos exercícios acima..

6. Dentre as árvores-guias exploradas no Exercício 8.8, alguma(s) dela(s) gerou(am) um alinhamento que você consideraria melhor do que aqueles obtidos até o momento para as sequências do arquivo `seqdata2.fas`? Justifique.

7. A topologia da árvores-guias nos exercícios que você fez diferem daquelas obtidas pelas análises filogenéticas das sequências alinhadas? Por que você acha que isso ocorre?

8. A implementação dos premissas de custo de alinhamento durante a análise filogenética modificou os resultados que você havia obtido até então para esses dados? Comente de acordo com os resultados que você obteve nos exercícios acima.

8.7 Leitura recomendada para discutir seus resultados

Phillips, A. *et al.* 2000. Multiple sequence alignments in phylogenetic Anlalysis. *Molecular Phylogenetics and Evolution* **16**(3): 317–330

8.8 Referências

1. Wheeler, W. C. 2012. Systematics: a course of lectures. Malaysia: Wiley-Backwell, 2012. 426.
2. Grant, T. & Kluge, A. G. 2004. Transformation series as an idiographic character concept. *Cladistics* **20**: 23–31.
3. Morrison, D. A. & Ellis, J. T. 1997. Effects of nucleotide sequence alignment on phylogenetic estimation: a case study of 18S rDNAs of Aplicomplexa. *Molecular Biology and Evolution* **14**(4): 428–441.
4. Wheeler, W. C. 2001. Homology and the optimization os DNA sequence data. *Cladistics* **17**: S3–S11.

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

6. Larkin, M. A. *et al.* 2008. ClustalW and ClustalX, version 2. *Bioinformatics* **23**(21): 2947–2948.

7. Sievers, F. *et al.* 2011. Fast, scalable generation of high-quality protein multiple sequence alignment using Clustal Omega. *Molecular Systems Biology* **7**: 539.

8. Edgar, R. C. 2004. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research* **32**(5): 1792–1797.

9. Katoh, K.; Misawa, K.; Kuma, K. & Miyata, T. 2004. MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform. *Nucleic Acids Research* **30**(14): 3059–3066.

10. Katoh, K. & Standley, D. M. 2013. MAFFT Multiple Sequence Alignment Software Version 7: Improvements in performance and usability. *Molecular Biology and Evolution* **30**(4): 771–780.

11. Needleman, S. B. & Wunsch, C. D. 1970. A general method applicable to the search of similatiries in the amino acid sequence of two proteins. *Journal of Molecular Biology* **48**: 443–153.

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

13. Wheeler, W. C. & Gladstein, D. L. 1994. MALIGN, version 1.93. New York, NY: American Museum of Natural History, 1994.

Tutorial 9

Homologia Dinâmica

BIZ0433 - INFERÊNCIA FILOGENÉTICA: FILOSOFIA, MÉTODO E APLICAÇÕES.

Conteúdo

Objetivo

O objetivo deste tutorial é introduzir o conceito de otimização direta e homologia dinâmica. O tutorial apresenta uma introdução ao uso de POY, programa no qual tais conceitos estão implementados operacionalmente. A introdução ao uso de POY explora os conceitos básicos relacionados à sintaxe de comandos do programa, o uso de linguagem de *scripts* a e implementação de funções de custo durante o processo de otimização direta. 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/
```

9.1 Contextualização

No tutorial anterior (Tutorial 8) observamos a interdependência de parâmetros de alinhamento e árvores-guias no alinhamento e consequentemente em inferência filogenética. Saber o que deve ser considerado um "bom" alinhamento é impossível. Há uma série de artigos comparando métodos de alinhamento [veja 1:145, e referências citadas]. Estas comparações levam em consideração as distâncias entre alinhamentos "reais" (*i.e.*, dados simulados) daqueles inferidos por programas de alinhamento múltiplos [*i.e.*, 2] ou performance de acordo com uma função objetiva qualquer (*e.g.*, SP[1] ou custo da topologia) [*e.g.*, 3].

Wheeler [1] argumenta que alinhamentos "reais" provavelmente nunca serão encontrados na natureza e que, portanto, comparações com dados simulados são inapropriadas. Contrariamente, Ogden & Rosemberg [2:190] consideram inapropriado comparar custos de topologias geradas por diferentes alinhamentos. Isso porque alinhamentos distintos postulam diferentes proposições de homologia e assim devem ser considerados matrizes de dados distintas.

Em inferência filogenética, o foco central de qualquer procedimento analítico é identificar a(s) melhor(es) topologia(as) de acordo com o critério de otimalidade selecionado pelo investigador. Métodos que geram soluções melhores para este problema devem ser favorecidos [3]. Como a escolha de topologias é feita pelo ordenamento destas hipóteses de acordo com alguma função objetiva (*e.g.*, distância patrística em parcimônia), o melhor que um método pode fazer é otimizar custo de topologias – o chamado *Tree Alignment Problem* (**TAP**) – [4]. Como definido por Wheeler [1:133], **TAP** foi originalmente descrito como um problema no qual, dado uma topologia, objetiva-se identificar o alinhamento que minimiza o seu custo. Neste processo, o resultado final é o alinhamento implícito [senso 5], resultado dos estados de caráter atribuídos aos nós (vértices) da topologia. Devemos considerar que há uma complexidade combinatória relacionada às possíveis atribuições de estados de caráter no nós da topologia. De fato, pode existir um número exponencial de soluções dado uma topologia. Veja o exemplo na Figura 9.1.

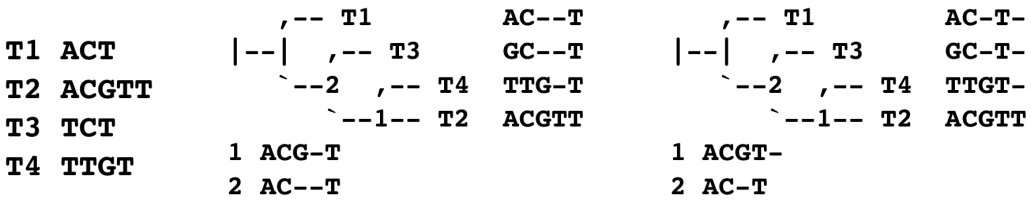


Figura 9.1: Exemplo de *Tree Alignment*. Considere as sequências à esquerda. Há uma única topologia para estas sequências com o custo de 5 transformações. No entanto há dois alinhamentos implícitos relacionados a esta topologia. Para cada um deles, a atribuição dos estados nos nós é diferente.

Análises filogenéticas de dados moleculares apresentam alguns desafios além daqueles inerentes da complexidade associada ao exame ou exploração do espaço de topologias que é uma função determinada pelo número de terminais – como vimos no Tutorial 3. Dados moleculares podem variar não somente em números de elementos, isto é, estados de caracteres, mas também em

[1] *sum-of-pairs* de um alinhamento qualquer é a soma dos custos dos alinhamentos par-a-par existentes neste alinhamento (maiores detalhes em http://cs124.cs.ucdavis.edu/lectures/multextra.pdf

tamanho (veja Tutorial 8). Nestes casos, tradicionalmente é feito um alinhamento a partir do qual se procede com as análises de inferência filogenética. Este alinhamento estabelece as correlações entre estados de caráter em cada coluna, considerados como proposições de homologia entre os pares de base de cada sequência dentro do contexto de homologia estática. Como visto anteriormente (seção 8.4.1 do Tutorial 8), esquemas putativos de homologia (alinhamentos) variam entre diferentes topologias utilizadas durante o alinhamento. Também foi observado que estes cenários putativos de alinhamento geram topologias com custos distintos. O conceito de homologia dinâmica [senso 6] integra alinhamento e busca de topologias em um único procedimento analítico conhecido como otimização direta [*direct optimization*; 7]. Ao integrar esses processos analíticos, conjugamos dois níveis de complexidades, um associado ao número de topologia e o outro associado ao número de alinhamentos possíveis.

Neste tutorial iremos explorar alguns conceitos práticos e teóricos associados à otimização direta de sequências nucleotídicas implementadas em POY [8, 9]. De nenhuma maneira este tutorial irá explorar toda a funcionalidade de POY e o interessado deve explorar os tutoriais disponíveis na documentação do programa (veja materiais associados a esse tutorial e literatura referenciada no final deste tutorial).

9.2 Introdução ao POY

9.2.1 Etapas de execução em POY

O POY é um programa extremamente versátil. Parte desta versatilidade está na possibilidade de analisar simultaneamente diferentes fontes de dados (*i.e.*, sequências nucleotídicas, matrizes morfológicas, entre outras), mas também na forma de interpretação e execução de *scripts*. Este último componente pode ser a parte mais complexa do uso de POY e o aluno interessado em possuir melhor domínio da linguagem de *script* de POY deve consultar a Seção 3.1 da documentação do programa (`tutorial_9/literature/poy_commands.pdf`). A execução mais simples de POY requer alguns componentes primários. O primeiro deles é a leitura dos arquivos de dados. O segundo, é a implementação da função de custo, ou seja, os parâmetros de alinhamento. Essa etapa é opcional caso o usuário queira utilizar os valores de *default* do programa – custo 1 para todas as transformações. Uma vez cumprida estas etapas, o usuário inicia a busca, seleciona a topologia mais curta e reporta o resultado. Vamos ver como isso é feito na prática.

Em um terminal, execute:

```
$ poy5.1.2a
```

Você deverá obter:

```
alan@turing: $ poy5.1.2a
Information : Welcome to POY 5.1.1 (2141b0ef4b2f+)
Compiled on Thu Jul 23 10:03:53 EDT 2020 with parallel off, interface flat,
likelihood on, and concorde off.
Copyright (C) 2011, 2012, 2013, 2014 Andres Varon, Nicholas Lucaroni, Lin
Hong, Ward Wheeler, and the American Museum of Natural History.
POY 5.1.1 comes with ABSOLUTELY NO WARRANTY; This is free software, and you
```

```
are welcome to redistribute it under the GNU General Public License Version
2, June 1991.


Information : Setting random seed value to 1431363032
Information :
Information :
Information :
Information : For help, type help().

          Enjoy!
Information :
            Trees:
              Storing 0 trees
            Cost Mode: Normal Direct Optimization
poy>
```

Nosso primeiro passo é fazer a leitura do arquivo se dados. Nesse exemplo iremos usar o arquivo `seqdata1.fas`. No *prompt* de POY execute:

```
poy> read("seqdata1.fas")
```

você deverá obter:

```
poy> read("seqdata1.fas")
Information : Reading file seqdata1.fas of type input sequences
Information :
The file seqdata1.fas contains sequences of 10 taxa, each sequence holding 1
fragment.
Status : Loading Trees Finished
Information :                Trees:
              Storing 0 trees
            Cost Mode: Normal Direct Optimization

poy>
```

Neste exemplo, POY informa que leu o arquivo `seqdata1.fas` e que o mesmo possui um único fragmento.

Nosso próximo passo é implementar a busca; pois iremos usar as funções de custo internas do programa. Iremos fazer isso utilizando os métodos convencionais para explorar o espaço de topologias como fizemos inicialmente em TNT (*i.e.*, **RAS+SWAP**), veja seção 4.7 do Tutorial 4). Para implementar este tipo de busca são necessários dois comandos. O primeiro é o comando `build()`, que constrói árvores de Wagner à medida em que calcula o custo utilizando o algoritmo de Needleman-Wunsch [10]. O segundo é o comando `swap()` que executa o refinamento por SPR e TBR e, para cada rearranjo, calcula o custo da topologia da mesma forma que o fez na construção da árvore de Wagner. Vejamos isso na prática. No prompt de POY, execute:

```
poy> build(100)
```

Observe que o comando `build()` requer um argumento que, neste caso, foi um valor numérico que é interpretado pelo POY como o número de RAS (*random addition sequence* que deverá ser executado. Após a execução deste comando, você deverá obter:

```
poy> build(100)
Status : Wagner : 2 of 10 – Wagner tree with cost 18.
Status : Wagner : 3 of 10 – Wagner tree with cost 34.
Status : Wagner : 4 of 10 – Wagner tree with cost 59.
Status : Wagner : 5 of 10 – Wagner tree with cost 75.
Status : Wagner : 6 of 10 – Wagner tree with cost 104.
Status : Wagner : 7 of 10 – Wagner tree with cost 114.
Status : Wagner : 8 of 10 – Wagner tree with cost 138.
Status : Wagner : 9 of 10 – Wagner tree with cost 152.
Status : Wagner Finished
Status : Building Wagner Tree Finished
Status : Running Pipeline : 1 of 100 – Estimated finish in 3 s
Status : Wagner : 2 of 10 – Wagner tree with cost 24.
Status : Wagner : 3 of 10 – Wagner tree with cost 39.
Status : Wagner : 4 of 10 – Wagner tree with cost 64.
Status : Wagner : 5 of 10 – Wagner tree with cost 88.
Status : Wagner : 6 of 10 – Wagner tree with cost 101.
Status : Wagner : 7 of 10 – Wagner tree with cost 112.
Status : Wagner : 8 of 10 – Wagner tree with cost 129.
Status : Wagner : 9 of 10 – Wagner tree with cost 155.
Status : Wagner Finished
Status : Building Wagner Tree Finished
Status : Running Pipeline : 2 of 100 – Estimated finish in 2 s
...
Status : Wagner Finished
Status : Wagner build : 100 of 100 – Estimated finish in 0 s
Status : Wagner : 2 of 10 – Wagner tree with cost 33.
Status : Wagner : 3 of 10 – Wagner tree with cost 56.
Status : Wagner : 4 of 10 – Wagner tree with cost 72.
Status : Wagner : 5 of 10 – Wagner tree with cost 93.
Status : Wagner : 6 of 10 – Wagner tree with cost 113.
Status : Wagner : 7 of 10 – Wagner tree with cost 129.
Status : Wagner : 8 of 10 – Wagner tree with cost 138.
Status : Wagner : 9 of 10 – Wagner tree with cost 148.
Status : Wagner Finished
Status : Wagner build : 101 of 100 – Estimated finish in 0 s
Status : Wagner build Finished
Information :
        Trees:
          Storing 100 trees with costs 153. to 165.
          Best cost in 9 trees
        Cost Mode: Normal Direct Optimization
poy>
```

Os resultados acima indicam que o POY contruiu 100 árvores de Wagner cujos custos variam de 153 a 165[2]. Dentro destas 9 delas possuem o menor custo.

Nossa próxima etapa será o refinamento destas topologias implementando rearranjos via SPR e TBR. O comando do POY para essas estratégias de refinamento é o `swap()`. Esse comando implementará os algoritmos de refinamento para todas as topologias contidas na memória. A forma mais simples de execução desse comando é a seguinte:

```
poy> swap()
```

cujo resultado seria:

```
poy> swap()
Status : Tree search : 1 of 100 --
Status : TBR : 174 Searching
Status : TBR Finished
...
Status : SPR : 157 157.
Status : SPR : 155 155.
Status : SPR : 154 154.
Status : SPR Finished
Status : Alternate : 0 Performing TBR swapping
Status : Single TBR Finished
Status : Alternate Finished
Status : Tree search : 100 of 100 – Estimated finish in 0 s
Status : Alternate : 0 Beginning search
Status : Alternate : 0 SPR search
Status : SPR : 153 153.
Status : SPR Finished
Status : Alternate : 0 Performing TBR swapping
Status : Single TBR Finished
Status : Alternate Finished
Status : Tree search Finished
Information :
        Trees:
          Storing 100 trees with costs 153. to 160.
          Best cost in 38 trees
        Cost Mode: Normal Direct Optimization
poy>
```

Após o refinamento, POY manteve 100 topologias que variaram de 153 a 160 em custo dentre as quais 38 possuem o menor custo.

Como dentro do conjunto de topologias mantidas na memória de POY há árvores subótimas, nosso próximo passo é selecionar aquelas que nos interessam. Para a seleção de topologias **únicas** e **ótimas**, utilizamos os parâmetros de *default* do comando `select()`. No terminal do POY execute:

[2] Estes números podem variar de execução a execução dado a aleatoriedade associada ao algoritmo.

```
poy> select()
```

você deverá obter:

```
poy> select()
Information :
        Trees:
         Storing 3 trees with costs 153. to 153.
         Best cost in 3 trees
        Cost Mode: Normal Direct Optimizationn
poy>
```

O resultado de POY indica que foram encontradas 3 topologias com o custo de 153. Observe que antes de selecionarmos as topologias **únicas** e **ótimas** do conjunto de topologias que compilamos após o refinamento – na etapa anterior –, haviam 38 árvores com o custo de 153. O resultado após implementarmos o comando `select()` significa que dentro deste conjunto de 38 topologias, apenas três delas eram diferentes uma das outras.

Finalmente, vamos verificar nossos resultados, ou seja, as topologias obtidas. POY imprime os resultados ou informações sobre os arquivos de entrada pelo comando `report()`. Há várias formas de obter as topologias encontradas por POY, vejamos algumas delas. No prompt de POY, execute:

```
poy> report(asciitrees)
```

Você deverá obter as topologias encontradas impressa no terminal no formato visual com caracteres ASCII. Se você executar:

```
poy> report(trees:(total))
```

Você deverá obter as topologias encontradas impressa no terminal em formato parentético (*i.e., newick format*) e seus respectivos custos entre colchetes.

Finalmente, o comando `report()` permite que você direcione os resultados para arquivos e que você faça isso utilizando um ou mais arguentos. Por exemplo, se você executar:

```
poy> report("newicktrees.tre",trees,"asciitrees.txt",asciitrees)
```

Você deverá obter dois arquivos em seu diretório de trabalho. O primeiro é o arquivo `newicktrees.tre` que conterá as topologias selecionadas em formato parentético. O segundo é o arquivo texto `asciitrees.txt` que conterá as topologias selecionadas em formato ASCII.

Finalmente, para sair de POY você deve digitar `exit()`.

9.2.2 *Scripts* DE EXECUÇÃO EM POY

A maneira mais efetiva de utilizar o POY é utilizando *scripts*. Os *scripts* de POY são arquivos textos contendo os comandos de execução seguindo a estrutura analítica que acabamos de fazer. Vamos repetir o que acabamos de fazer utilizando essa forma de execução.

Exercicio 9.1

Neste exercício você deverá executar o POY utilizando um *script*.

i. Crie um arquivo texto chamado `meu_script1.poy` com o seguinte conteúdo:

Arquivo texto 9.1: conteúdo do arquivo meu_script1.poy

```
read("seqdata1.fas")
set(root:"Taxon1")
build(100)
swap()
select()
report("seqdata1_do.tre", trees:(total))
exit()
```

Dentre estes comandos, o único que não foi utilizado anteriormente é o comando `set(root:"Taxon1")` – linha 2, que indica ao POY que todas as topologias deverão ser enraizadas pelo terminal `Taxon1`.

ii. Execute o *script* `meu_script1.poy` utilizandoo seguinte comando em seu terminal:

```
$ poy5.1.2a meu_script1.poy
```

Primeiro observe as últimas 31 linhas do registro de execução de POY. Você deve ter obtido:

... Status : Running Pipeline : 99 of 100 – Estimated finish in 0 s
Status : Wagner : 2 of 10 – Wagner tree with cost 30.
Status : Wagner : 3 of 10 – Wagner tree with cost 54.
Status : Wagner : 4 of 10 – Wagner tree with cost 66.
Status : Wagner : 5 of 10 – Wagner tree with cost 87.
Status : Wagner : 6 of 10 – Wagner tree with cost 113.
Status : Wagner : 7 of 10 – Wagner tree with cost 118.
Status : Wagner : 8 of 10 – Wagner tree with cost 139.
Status : Wagner : 9 of 10 – Wagner tree with cost 146.
Status : Wagner Finished
Status : Building Wagner Tree Finished
Status : Alternate : 0 Beginning search
Status : Alternate : 0 SPR search
Status : SPR : 158 158.
Status : SPR : 156 156.
Status : SPR Finished
Status : Alternate : 0 Performing TBR swapping
Status : Single TBR Finished
Status : Alternate Finished
Status : Running Pipeline : 100 of 100 – Estimated finish in 0 s
Status : Running Pipeline Finished
Status : Diagnosis : 0 Recalculating original tree

```
Status : Diagnosis Finished
Status : Diagnosis : 1 of 3 – Recalculating trees
Status : Diagnosis : 0 Recalculating original tree
Status : Diagnosis Finished
Status : Diagnosis : 2 of 3 – Recalculating trees
Status : Diagnosis : 0 Recalculating original tree
Status : Diagnosis Finished
Status : Diagnosis : 3 of 3 – Recalculating trees
Status : Diagnosis Finished
```

Quando POY executa um *script*, as instruções de execução funcionam de uma forma um pouco diferente em comparação com os comandos que você executou passo a passo anteriormente. No exemplo da seção 9.2.1, POY construiu todas as **RAS** para depois fazer o refinamento via SPR+TBR. Quando executado via *script*, POY faz o refinamento logo após a construção da árvore de Wagner e retém a(s) topologia(s) de menor custo. O resultado da análise pode ser verificado no arquivo `seqdata1_do.tre` que contém 3 topologias com o custo de 153:

```
(Taxon1,((Taxon4,(Taxon3,Taxon5)),((Taxon7,Taxon8),(Taxon9,(Taxon10,(Taxon2,Taxon6))))));
(Taxon1,((Taxon4,(Taxon3,Taxon5)),(Taxon10,((Taxon2,Taxon6),(Taxon9,(Taxon7,Taxon8))))));
(Taxon1,((Taxon4,(Taxon3,Taxon5)),((Taxon9,(Taxon2,Taxon6)),(Taxon10,(Taxon7,Taxon8)))));
```

Finalmente, vale ressaltar que a versão mais rescente de POY (atualmente 5.1.1) atribui custos idênticos para qualquer tipo de transformação (*i.e.*, substituições e INDELs) ao passo que nas versões anteriores a razão de custos entre INDELs:substituições é de 2:1.

iii. Os dados utilizados neste exercício são os mesmos que você utilizou no Tutorial 8 na seção 8.2. Compare os resultados de POY com aqueles obtidos por homologia estática registrados nas Tabelas 8.1 e 8.1 do Tutorial 8 e responda:

a. O custo obtido em POY é melhor ou pior do que os resultados que você havia obtido?

_____

_____

b. As topologias diferem?

_____

_____

9.2.3 Análises simultâneas em POY

Um dos componentes mais interessantes de POY é a possibilidade de analisar diferentes fontes de dados simultaneamente. Considere que você possui 3 partições com propriedades analíticas distintas. Suponha que você tenha uma partição que está sujeita a alinhamento, pois as sequências diferem de tamanho, uma outra partição cujos dados genotípicos provém de uma região codificadora que não contém INDELs, e ainda uma matriz de dados fenotípicos. Dentro do conceito de homologia estática, o procedimento analítico a ser adotado seria o alinhamento da primeira partição que posteriormente seria concatenada com as demais partições e este conjunto

de dados seria submetido à análise filogenética. No entanto, considere que o alinhamento da primeira partição depende de uma topologia para a qual as demais partições não possuem nenhuma influência. No exemplo que se segue iremos explorar como POY analisaria essas partições simultaneamente.

Considere os seguintes conjuntos de dados:

1. partition1.fas – dados não sujeitos a alinhamento
2. partition2.fas – dados sujeitos a alinhamento[3]
3. partition3.tnt – dados fenotípicos

Dentre as partições apresentadas acima, `partition1.fas` e `partition3.tnt` não devam ser submetidas á otimização direta ao passo que a `partition2.fas` deverá ser analisada por homologia dinâmica. A implementação de uma análise em POY que leve em consideração as propriedades destas partições requer a estrutura ilustrada no `script` abaixo (Script 9.2):

Arquivo texto 9.2: conteúdo do arquivo partitions_script.poy

```
read(prealigned:("partition1.fas"))
read("partition2.fas","partition3.tnt")
set(root:"Taxon1")
build(100)
swap()
select()
report(trees, treestats)
exit()
```

Este *script* está contido no arquivo `partitions_script.poy`. Na linha 1, POY lê os arquivos `partition1.fas` no qual o argumento `prealigned` do comando `read` informa ao POY que o arquivo `partition1.fas` contém sequências nucleotídicas pré-alinhadas. A linha 2, instrui o POY a ler os arquivos `partition2.fas` e `partition3.tnt` e considera que o primeiro será submetido à otimização direta ao passo que o segundo é uma matriz de dados de homologia estática – no caso, dados fenotípicos. Finalmente, o comando `report()` contém dois argumentos, `trees, treestats`, cujo primeiro retorna a(s) topologia(a) encontrada(s) e o segundo o custo da(s) mesma(s). Ao executá-lo você deverá obter:

```
...
Status : Wagner Finished
Status : Building Wagner Tree Finished
Status : Alternate : 0 Beginning search
Status : Alternate : 0 SPR search
Status : SPR : 376 376.
Status : SPR : 375 375.
```

[3] Estes dados encontram-se alinhados, porém POY irá desconsiderar os *gaps* a não ser que você especifique ao programa que esta partição deva ser considerada como pré-alinhada (veja abaixo)

```
Status : SPR Finished
Status : Alternate : 0 Performing TBR swapping
Status : Single TBR Finished
Status : Alternate Finished
Status : Running Pipeline : 100 of 100 – Estimated finish in 0 s
Status : Running Pipeline Finished
Information : Vectorized:8 of 8 characters.
Status : Diagnosis : 0 Recalculating original tree
Status : Diagnosis Finished
Status : Diagnosis : 1 of 1 – Recalculating trees
Status : Diagnosis Finished
(Taxon1,((Taxon5,(Taxon2,(Taxon4,Taxon9))),(Taxon6,(Taxon8,(Taxon3,(Taxon10,Taxon7))))))
;


Trees Found:


Tree length    Number of hits
370.           1
```

A análise destas partições resulta em uma única topologia com o custo de 370.

Exercicio 9.2

Neste exercício você deverá fazer uma análise de cada partição destes dados e comparar os resultados das análises individuais com a análise simultânea. Para cada análise, registre as topologias no espaço abaixo e responda:

Topologias encontradas nas análises filogenéticas das partições do Exercício 9.2:

Análise simultânea	Partição 1
Partição 2	Partição 3

i. As topologias são idênticas?

ii. Para cada clado presente na análise simultânea, identique se eles são corroborados pelas partições. Você considera que alguma partição tem maior influência na estrutura cladistica

(topologia) recuperada na análise simultânea?

_____

_____

iii. Considerando o número de caracteres de cada partição, existe alguma relação entre número de caracteres das partições e a influência na estutura da topologia da análise simultânea?

_____

_____

9.2.4 Implementação de matrizes de custo em POY

A implementação de parâmetros de alinhamento (= função de custo) em POY é feita pelo comando "`transform()`" e o argumento "`tcm:`". Este argumento define a matriz de custo de transformação para conjuntos de caracteres. O *default* de POY é `tcm:(1,1)`, no qual o custo para substituições e INDELs, respectivamente é 1. Em versões anteriores de POY, o custo de INDELs era 2, portanto, o argumento seria expresso como `tcm:(1,2)`. Para o exemplo anterior, há duas formas de implementar matrizes de custo de transformação. Na primeira delas, o Script 9.2 seria modificado da seguinte maneira:

Arquivo texto 9.3: Modificação do arquivo partitions_script.poy para implementar matriz de custo de transformação.

```
read(prealigned:("partition1.fas"))
read("partition2.fas","partition3.tnt")
transform(tcm:(1,2))
set(root:"Taxon1")
build(100)
swap()
select()
report(trees, treestats)
exit()
```

Neste *script* a linha 3 implementa os custos de transformação para todos os caracteres no qual o valor atribuído para substituições e INDELs é 1 e 2, respectivamente. No entanto, observe que somente uma das partições possuí INDELs e talvez seja desejável especificar para qual destas partições POY deverá considerar a matriz de custo em questão. Se modificarmos o Script 9.3 da seguinte maneira:

Arquivo texto 9.4: Modificação do arquivo partitions_script.poy para implementar matriz de custo de transformação para a partição `partition2.fas`.

```
read(prealigned:("partition1.fas"))
read("partition2.fas","partition3.tnt")
```

```
transform(names:("partition2.fas"),(tcm:(1,2)))
set(root:"Taxon1")
build(100)
swap()
select()
report(trees, treestats)
exit()
```

Neste caso, o argumento "`names:`" identifica a partição `partition2.fas` para qual a matriz de custo de transformação "`tcm:(1,2)`" será implementada.

Outra forma de implementar matrizes de custo em POY é utilizando arquivos que contém os custos de transformação. Estas matrizes são matrizes 5x5, no caso de sequências nucleotídicas, nas quais são especificadas os custos de transformação para cada estado (*i.e.*, `INDELs`, `A`, `C`, `G` e `T`). Por exemplo, verifique o conteúdo do arquivo 2111.tcm:

```
0 1 1 1 2
1 0 1 1 2
1 1 0 1 2
1 1 1 0 2
2 2 2 2 0
```

Esta matriz especifica as seguintes transformações:

```
    [A][C][G][T][-]
[A]  0   1   1   1   2
[C]  1   0   1   1   2
[G]  1   1   0   1   2
[T]  1   1   1   0   2
[-]  2   2   2   2   0
```

Essa é uma forma mais versátil de implementar matrizes de custo, pois possibilita que o usuário atribua custos diferenciais para determinados tipos de transformação (*e.g.*, transições e transversões). A linha de comando que implementa matrizes de custo expressas em arquivos com esse conteúdo é ilustrado na modificação do Script 9.5 feita abaixo:

Arquivo texto 9.5: Modificação do arquivo partitions_script.poy para implementar matriz de custo de transformação para a partição `partition2.fas`.

```
read(prealigned:("partition1.fas"))
read("partition2.fas","partition3.tnt")
transform(names:("partition2.fas"),(tcm:("211.tcm")))
set(root:"Taxon1")
```

```
build(100)
swap()
select()
report(trees, treestats)
exit()
```

Exercicio 9.3

Neste exercício você deverá executar 2 análises em POY para os dados contidos nos arquivos `partition1.fas`, `partition2.fas` e `partition3.tnt` em que o custo para INDELs seja 2 e 4. As topologias resultantes destas análises deverão ilustradas no espaço abaixo e você deverá comentar se os resultados diferem da análise anterios na qual os custos para INDELs e substituições eram idênticos.

`tcm:(1,2)`	`tcm:(1,4)`

9.3 Referências

1. Wheeler, W. C. 2012. Systematics: a course of lectures. Malaysia: Wiley-Backwell, 2012. 426.
2. Ogden, T. H. & Rosemberg, M. S. 2007. Alignment and topological accuracy of the direct optimization approach via POY and traditional phylogenetics via Clustalw + PAUP*. *Systematic Biology* **56**: 182–193.
3. 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.

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

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

6. Wheeler, W. C. 2001. Homology and the optimization os DNA sequence data. *Cladistics* **17**: S3–S11.

7. Wheeler, W. C. 1996. Optimization alignment: the end of multiple sequence alignment in phylogenetics? *Cladistics* **12**: 1–9.

8. Varón, A.; Vinh, L. S. & Wheeler, W. C. 2010. POY version 4: phylogenetic analysis using dynamic homologies. *Cladistics* **26**: 72–85.

9. Varon, A.; Lucaroni, N.; Hong, L. & Wheeler, W. C. 2011–2014. POY version 4: phylogenetic analysis using dynamic homologies, version 5.0. New York, NY: American Museum of Natural History, 2011–2014.

10. Needleman, S. B. & Wunsch, C. D. 1970. A general method applicable to the search of similatiries in the amino acid sequence of two proteins. *Journal of Molecular Biology* **48**: 443–153.

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()`".

Arquivo texto 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.

Arquivo texto 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.

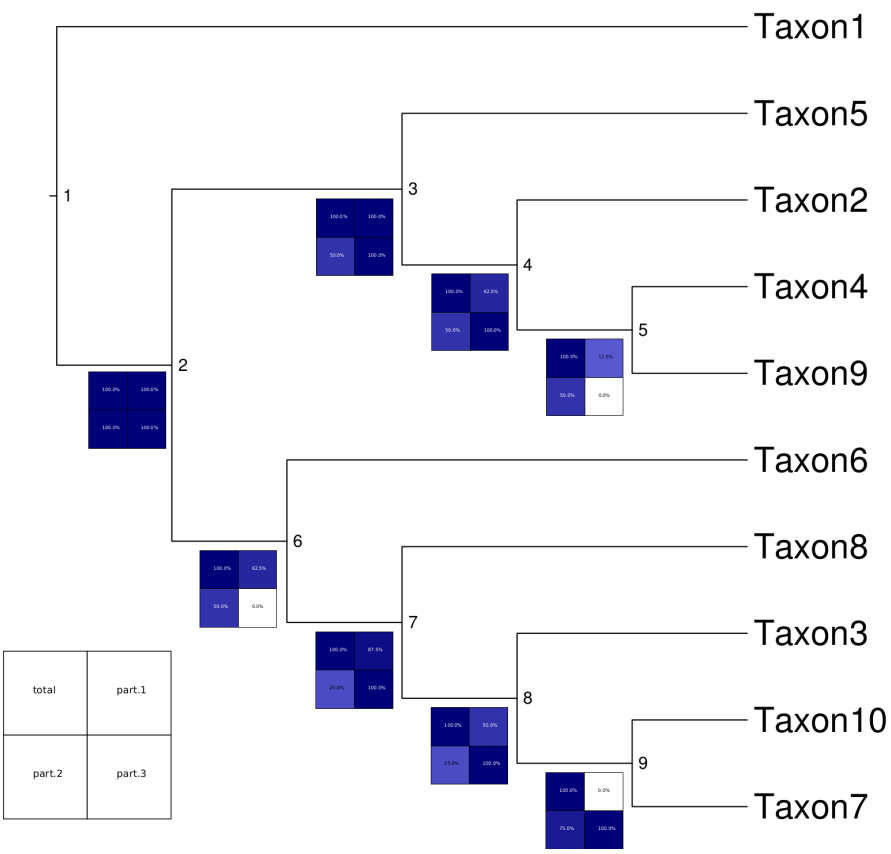


Figura 10.1: Diagrama de sensibilidade (*Navajo's rug*) indicando como cada clado da análise simultânea se comporta nas análises individuais de cada partição. Quadrados em azul indicam a presença do clado de acordo com o mapa do *plot* de sensibilidade ao lado do cladograma.

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.

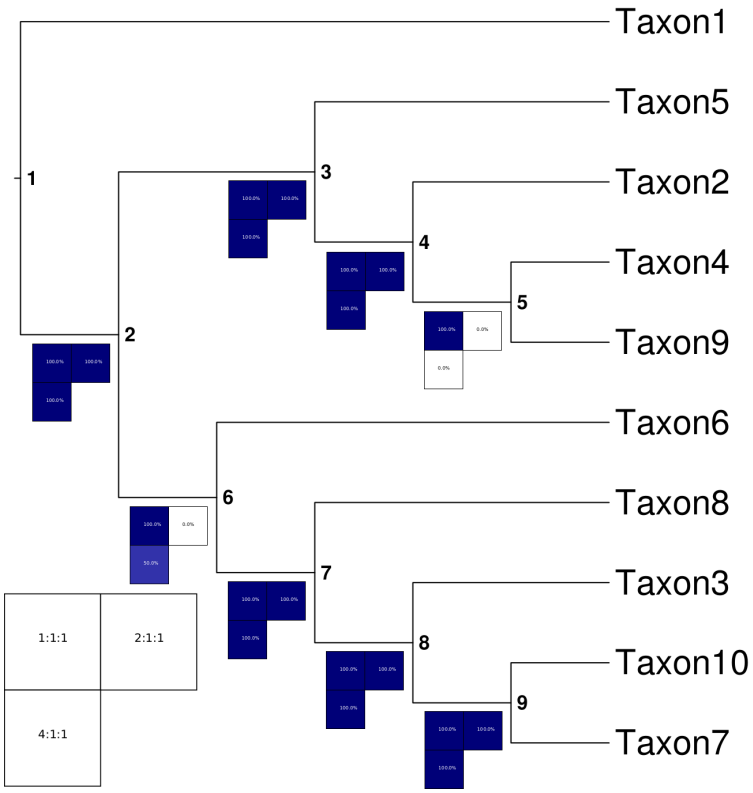


Figura 10.2: Diagrama de sensibilidade (*Navajo's rug*) para parâmetros de alinhamento indicando como cada clado da análise se comporta com relação aos custos diferenciais atribuídos para *gaps*. Quadrados em azul indicam a presença do clado de acordo com o mapa do *plot* de sensibilidade ao lado do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

Tutorial 11

Homologia Dinâmica: *Iterative pass*, dados lacunares e partições em POY

BIZ0433 - INFERÊNCIA FILOGENÉTICA: FILOSOFIA, MÉTODO E APLICAÇÕES.

Conteúdo

Objetivo

O objetivo deste tutorial é introduzir o conceito de *Iterative pass* em análises de homologia dinâmica e suas aplicações nos processos de refinamento de análises filogenéticas. Este tutorial também aborda, de forma breve, os efeitos de dados lacunares (*missing data*) em análises filogenéticas e aponta para os cuidados que devem ser tomados ao analisar sequências nucleotídicas em POY para as regiões de *gap* iniciais e finais. Finalmente, o tutorial introduz estratégias de particionamentos que visam (*i*) minimizar efeitos indesejados de dados lacunares e (*ii*) aumentar a eficiência computacional de POY durante análises filogenéticas. 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/
```

11.1 Iterative Pass

Desde que o conceito de *alinhamento* (ou *alinhamento múltiplo*) foi apresentado (Tutorial 8), ficou evidente não só sua importância em inferência filogenética, bem como o quão dependente alinhamentos são de árvores-guia e os parâmetros numéricos de definem as funções de custo que norteiam a função objetiva do algoritmo (veja Phillips *et al.* [1] e Giribet *et al.* [2]). Embora definir o que é um "bom alinhamento" seja uma tarefa subjetiva e que alinhamentos "reais" provavelmente nunca serão encontrados na natureza, como argumenta Wheeler [3], devemos considerar que o foco central em inferência filogenética é identificar a(s) melhor(es) topologia(as) de acordo com o critério de otimalidade que adotarmos. Consequentemente, os métodos que geram soluções melhores para este problema devem ser favorecidos [4].

A escolha da hipótese filogenética em inferência filogenética baseada em dados moleculares é feita pelo ordenamento do custo (distância patrística) de cada topologia em relação aos dados. No caso de sequências nucleotídicas sujeita a alinhamento, o cálculo deste custo define o que chamamos de *Tree Alignment Problem* (**TAP**) – [5]. Para resolver este problema, Wheeler [6] considerou o *Tree Alignment Problem* como sendo um problema de otimização de caracteres em árvores filogenéticas dentro de um procedimento analítico chamado *optimization alignment* ou *direct optimization* (**OA/DO**). Este procedimento analítico é também conhecido como homologia dinâmica (senso Wheeler [7]), em contraste à forma tradicional de análises filogenéticas baseadas em dados moleculares na qual o alinhamento é um procedimento isolado que precede a busca por topologias ótimas, cujo resultado permite a obtenção de um alinhamento implícito (senso Wheeler [8]).

Seja qual for o procedimento adotado para alinhamentos múltiplos, todos os programas examinados até o momento baseiam-se no algoritmo de Needleman-Wunsch [1, 9]. Este algoritmo alinha pares de sequências independentemente do procedimento ser feito dentro do contexto de homologia estática para uma árvore-guia ou utilizando uma determinada topologia em homologia dinâmica por otimização direta. Em ambos os casos, estas são estratégias heurísticas que indiretamente (utilizando uma árvore-guia, no caso de homologias estáticas) ou diretamente (computando o custo de uma determinada topologia, no caso de homologia dinâmica) estimam as sequências para os nós de uma árvore filogenética. Esta estimativa nada mais é do que a atribuição dos estados de caráter aos ancestrais hipotéticos (**HTU**, de *Hypothetical Taxonomic Units*), ou seja aos vértices da topologia.

Estimar sequências ancestrais [*protosequences* senso 10] que minimizam o custo de um alinhamento em uma topologia é atualmente um procedimento heurístico. Como vimos nos tutoriais anteriores, a otimização direta tende a encontrar soluções melhores em comparação com os procedimentos analíticos utilizando esquemas de homologia estática [veja Tutoriais 9 e 10, 2]. O caráter heurístico desse procedimento reside no fato de que o cálculo exato de um alinhamento múltiplo de n sequências implicaria o mesmo número de dimensões de cálculo do algoritmo de Needleman-Wunsch. Teoricamente, a solução exata para esse problema multi-dimensional foi proposta por Sankoff *et al.* [11] e já havia sido implementada para o alinhamento de três sequências por Sankoff & Cedergren [10] em uma versão tridimensional do algoritmo de

Needleman & Wunsch [9].

Wheeler [12] concebeu um procedimento chamado *iterative pass* (**IP**) que aplica otimização direta em três sequências, sejam elas hipotéticas ou não, simultaneamente. Este procedimento que também considera iterações analíticas que visam estabilizar o assinalamento de estados ancestrais aos vértices da topologia [*i.e., iterative improvement*; veja 12] tende a gerar hipóteses filogenéticas ainda mais curtas que **DO**. No entanto o custo computacional deste algoritmo faz com que seu uso seja proibitivo durante a etapa de busca por topologias mais curtas (Figura 11.1). Desta forma, historicamente seu uso está restrito a etapas de refinamento do alinhamento implícito no final da análise filogenética [*e.g.*, 13–17]. No entanto, Machado & Marques [18] revelaram outras propriedades do **IP** que transcendem apenas a redução em custos de topologias. Esses autores demonstram que esse algoritmo permite reduzir o número de topologias igualmente parcimoniosas e selecionar topologias que não seriam selecionadas com **DO**.

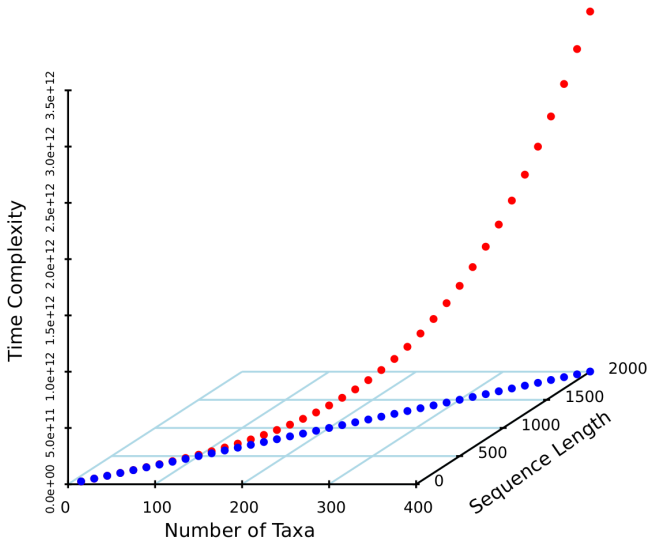


Figura 11.1: Complexidade computacional (medida em *time complexity O*) de **DO** ($O(km^2)$) – pontos azuis, onde k é o número de terminais e m o comprimento da sequência) em comparação com **IP** ($O(km^3)$) – pontos laranjas – em relação ao número de terminais e tamanho da sequência.

11.1.1 Implementação

A implementação de **IP** em POY é feita pelo argumento `iterative` do comando `set` do programa (veja seção 3.3.24 da documentação de POY) – portanto, seria `set(iterative)`. Há duas formas de *iterative pass* em POY. Por *default*, POY computa a forma exata da otimização tridimensional, cujo comando literal seria: `set(iterative:exact)`. No comando `set(iterative:approximate)`, as iterações feitas durante a análise se aproximam do alinhamento tridimensional porém usando alinhamentos par a par. Este argumento não tem impacto drástico no tempo computacional e requer muito menos memória RAM durante a execução. No entanto, sua performance comparada com o cálculo exato de **IP** não está documentada na literatura. Seja como for, **IP** deve ser aplicado na fase final de refinamento de sua análise e os dados que tempos sobre as propriedade desse algoritmo estão baseadas na

implementação exata.

11.1.2 Tempo de execução

Como pode ser visto na Figura 11.1A–C to tempo computacional de *iterative pass* incrementa consideravelmente quando comparado à otimização direta. A Figura 11.2 ilustra com maiores detalhes esta relação. Observe que o incremento no número de terminais, quando o comprimento da sequência é fixo (100), ou o incremento no comprimento da sequência, quando o número de terminais é fixo (100), afetam de maneira distinta o tempo computacional (Figura 11.2A). O mesmo efeito é visto quando comparamos estas duas variáveis com os algoritmos de **DO** e **IP** (Figuras 11.2B e C).

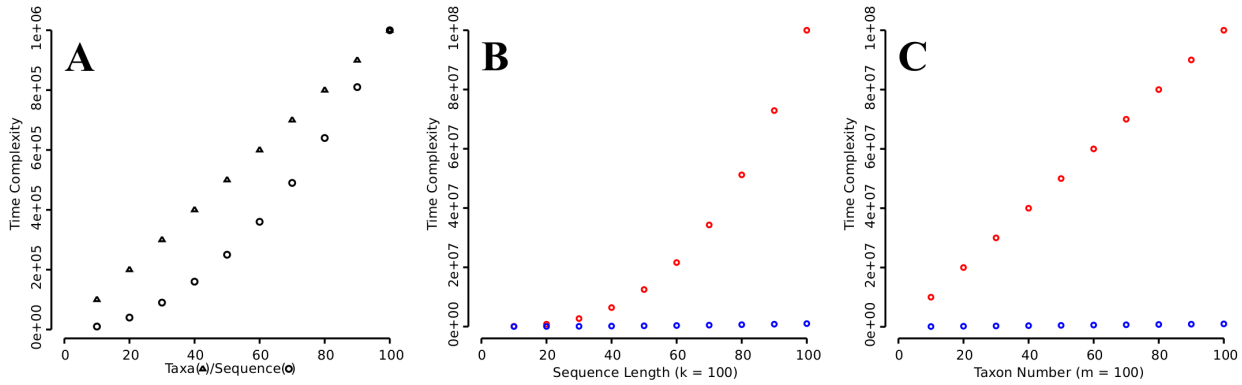


Figura 11.2: Número de terminais, comprimento da sequência e complexidade computacional de **DO** em comparação com **IP**. **A** – *Time complexity* O vs. números de terminais (para $k = 100$, triângulos) e comprimento de sequência (para $m = 100$, circulos). **B** – Comparação entre complexidade de **IP** (para $k = 100$, circulos vermelhos) e **DO** (circulos azuis) vs. comprimento da sequência. **C** – Comparação entre complexidade de **IP** (para $m = 100$, circulos vermelhos) e **DO** (circulos azuis) vs. números de terminais.

Exercicio 11.1

Considere o Script 11.1, abaixo, no qual a linha 3 configura o POY para fazer otimização por *iterative pass* durante a busca utilizando 10 **RAS+SWAP** das sequências contidas em `partition2.fas` e reporte o tempo de execução, o custo e o número de topologias encontradas. Os arquivos `partition2_6x30.fas`, `partition2_6x60.fas` e `partition2_6x100.fas` são modificações do arquivo `partition2.fas`. Todos contém 6 terminais apenas e diferem quanto ao tamanho das sequências (*i.e.*, 30, 60 e 100, respectivamente). Neste exercício você deverá analizar esses dados utilizando **DO** e **IP** e o resultado de cada análise deverá ser colocado na Tabela 11.1, abaixo.

Arquivo texto 11.1: Exemplo de *script* para implementar *iterative pass*.

```
read("partition2.fas")
set(root:"Taxon1")
set(iterative:exact)
Build(10)
```

```
swap()
select()
report(timer:"Time to run IP",treestats)
exit()
```

Tabela 11.1: Efeito do comprimento das sequências durante a otimização por *iterative pass*.

Dataset	Tempo (secs)	Custo das MPTs[1]	Número de MPTs
`partition2_6x30.fas`			
`partition2_6x60.fas`			
`partition2_6x1000.fas`			

i. Quais observações você pode fazer sobre essas análises?

11.1.3 Redução de MPTs

Uma das propriedades pouco conhecidas de **IP** é que nem todas as árvores consideradas ótimas em **DO** possuem o mesmo custo quando submetidas ao refinamento. Essa propriedade foi documentada pela primeira vez por Machado & Marques [18] e sua utilidade é evidente: em análises por otimização direta que resultam em múltiplas topologias, o algoritmo de *iterative pass* pode reduzir consideravelmente o número de topologias finais [veja também 16, 17]. O exercício a seguir demonstra essa propriedade.

Exercicio 11.2

Neste exercício você deverá executar dois *scripts*. No primeiro, você deverá fazer uma análise utilizando **DO** e salvar as topologias encontradas. O segundo *script* deverá fazer a rediagnose destas topologias via *iterative pass*. As linhas de instrução destes *scripts* devem obedecer as estruturas ilustradas abaixo.

Para o primeiro *script*,

1. Leia o arquivo "`partition2.fas`",
2. atribua a raíz ao táxon "`Taxon1`",
3. faça uma busca com o comando "`search`" por 10 minutos,
4. selecione as topologias únicas e melhores,
5. reporte no arquivo "`collected_trees.tre`" as topologias encontradas e em "`do.sts`" os custos e o número de topologias encontradas,

6. saia de POY.

Para o segundo *script*,

1. Leia os arquivos "`partition2.fas`" e "`collected_trees.tre`",
2. atribua a raiz ao taxon "`Taxon1`",
3. implemente o algoritmo exato de IP,
4. implemente o algoritmo de *tree fusing* (*default*) para as topologias contidas no arquivo "`collected_trees.tre`",
5. selecione as topologias únicas,
6. reporte no arquivo "`ip.sts`" os custos e o número de topologias encontradas,
7. saia de POY.

Finalmente, compare os conteúdos dos arquivos `do.sts` e `ip.sts` e faça um sumário dos seus resultados no espaço abaixo:

_____

_____

_____

11.1.4 Outras propriedades de **IP**

Há outras propriedades de **IP** documentadas por Machado & Marques [18] que devem ser ressaltadas. A primeira delas é que esse algoritmo quando conjugado com algoritmos de rearranjos se torna mais efetivo, principalmente *tree–fusing* [19]. Em alguns casos **IP**+*tree–fusing* é capaz de encontrar topologias que não estavam presentes no conjunto de topologias submetidas ao refinamento. No entanto, a propriedade mais interessante encontrada por esses autores está relacionada com a observação de que, para dados simulados no qual um conjunto de 10 topologias foram coletadas na fase de **DO** resultado de 10 iterações do comando `search`, **IP** considerou como topologia mais curta aquelas que foram consideradas subótimas por **DO** em 75% das simulações! Isso se deve em parte às aproximações e "*short cuts*" implementadas nos algoritmos de POY para reduzir o tempo computacional em **DO**. As implicações analíticas são óbvias. Sua análise será mais efetiva se você coletar um número razoável de topologias candidatas e submetê-las ao refinamento por **IP**+*tree–fusing*. Pinto-da-Rocha *et al.* [17], por exemplo, reduziram o custo das topologias em até 3.3% considerando topologias compiladas durante a análise de sensibilidade – estratégia de busca conhecida como *sensitivity search* [veja 20].

Exercicio 11.3

Neste exercício iremos explorar em conjunto a maioria dos conceitos apresentados neste e no tutorial anterior sobre análises por homologia dinâmica em POY. Este exercício lhe fornecerá as bases necessária para fazer uma boa análise de dados reais. O exercício é relativamente simples, mas requer atenção na confecção dos *scripts* de execução para que

você tenha bom controle do tempo de análise, bem como na coleção de informações que lhe permitirão responder às perguntas abaixo.

i. Qual é a diferença de custo entre a topologia encontrada por **DO** atribuindo pesos iguais para cada transformação e aquela encontrada por uma análise usando *sensitivity search* e diagnose por **IP+***tree–fusing*?

ii. A topologia final após *sensitivity search* e **IP+***tree–fusing* é a mesma encontracas por **DO** sob os mesmos parâmetros de alinhamento?

Para responder a essas perguntas você deverá fazer uma análise em POY utilizando os dados disponíveis nos arquivos `partition1.fas`, `partition2.fas` e `partition3.tnt`. Os resultados dessa sua análise deverão ser sumarizados no espaço abaixo:

11.2 Dados lacunares (*missing data*)

Aqueles que já tiveram experiência com compilação e análise de dados reais em inferência filogenética devem ter se deparado com lacunas em suas matrizes de dados – sejam elas genotípicas ou fenotípicas. Portanto, dados lacunares (*missing data*) são muito comuns em matrizes reais. Para dados moleculares, em particular, principalmente quando o estudo envolve mais de uma região genômica (*e.i.*, locus) e um número razoável de terminais, invariavelmente chega-se ao final da fase de compilação de dados com a dúvida se um determinado terminal deverá ou não ser incluído na análise em decorrência da observação de que para este há regiões – ou caracteres – que não foram obtidos. Há uma série de estudos que buscaram avaliar os efeitos de dados lacunares em inferência filogenética [veja 21–25; e referências citadas]. No entanto, estes estudos são inconclusivos – muito provavelmente resultado do caráter histórico e individual dos dados utilizados em inferência filogenética. A única generalização que se pode fazer é de que excesso de dados lacunares – seja lá como "excesso" é definido – tende a gerar múltiplas topologias igualmente parcimoniosas, pouco resolvidas e, para dados simulados, estimativas pouco acuradas de relacionamento filogenético [22]. No entanto, a quantidade de dados lacunares de um terminal, por si só, não deve guiar decisões sobre sua exclusão ou não em análises filogenéticas [22]. Isso porque, em alguns casos, a inserção destes terminais na análise pode melhorar drasticamente seu resultado final [23].

11.2.1 Dados lacunares em POY

Embora alguns estudos empíricos e com dados simulados tenham avaliado o efeito de dados lacunares em inferência filogenética, estes se restringem a análises de homologia estática. Não há nenhuma avaliação sobre os efeitos de dados lacunares em homologia dinâmica que eu tenha

conhecimento. No entanto, diante da situação na qual é necessário decidir se determinado terminal deve ou não ser incluído na análise devido a ausência de dados, aconselha-se a avaliar se os dados lacunares afetam ou não sua análise final. Isso pode ser feito avaliando o resultado da inferência filogenética incluindo ou não determinado(s) terminal(is) e/ou fragmento(s).

Ao contrário na maioria dos programas de inferência filogenética, POY não descarta uma base de dados após ler outra. Se um "novo" terminal não está presente no primeiro banco de dados, mas está presente no segundo, POY considerará que para este terminal inexiste dados para o primeiro fragmento, mas sem descartá-lo da análise. Por exemplo, considere dois bancos de dados:

`fasta_1.fas`

```
taxon1   AAAA
taxon2   ACAA
taxon3   AAGA
taxon4   AAAT
```

e

`fasta_2.fas`

```
taxon1   CCCC
taxon3   CACC
taxon4   CCGC
taxon5   CCCT
```

POY consideraria:

```
taxon1   AAAACCCC
taxon2   ACAA????
taxon3   AAGACACC
taxon4   AAATCCGC
taxon5   ????CCCT
```

Essa propriedade de POY tem um lado bom e um lado ruim. O lado bom é que o usuário não precisa se preocupar em manter bancos de dados distintos com os mesmos terminais. O lado ruim é que caso haja algum erro tipográfico em um terminal – em um determinado banco de dados–, POY o considerará como outro terminal. Portanto, uma dica: usem nomes simples para os terminais e lembre-se que você poderá fazer substituições no final utilizando ferramentas como o `sed` ou até mesmo comandos internos de POY.

Há dois comandos em POY que estão estreitamente relacionados com a manipulação de dados lacunares. O primeiro deles é o o argumento `cross_references` que pode ser implementado no comando `report()`. Seu uso permite gerar uma tabela indicando a distribuição de dados para os terminais. Considere o *script* abaixo:

Arquivo texto 11.2: Exemplo de *script* para implementar `cross_references`.

```
read("fasta1.fas","fasta2.fas")
report("refs.txt",cross_references)
exit()
```

O arquivo `refs.txt` teria o seguinte conteúdo:

```
File References:
Terminal    fasta2.fas        fasta1.fas
taxon1          +                 +
taxon2          -                 +
taxon3          +                 +
taxon4          +                 +
taxon5          +                 -
```

onde + e − indicam presença e ausência, respectivamente, de determinado fragmento para um terminal.

O segundo comando é `select()`. Este comando permite selecionar subconjuntos de terminais, caracteres e topologias (veja seção 3.3.23 da documentação de POY). Por exemplo, o comando `select(terminals,files:("file.txt"))` ou simplesmente `select(files:("file.txt"))` seleciona o conteúdo do arquivo `file.txt` (*i.e.*, terminais, caracteres ou topologias) para a análise. Esse comando, portanto, permitiria-lhe incluir e excluir terminais e caracteres para verificar o efeito de dados lacunares em sua análise.

Exercicio 11.4

Neste exercício você deverá fazer uma análise filogenética dos dados contido em `asteroids1.fas`, `asteroids2.fas` e `asteroids3.fas` que permita-lhe responder às perguntas abaixo.

i. Há dados lacunares nessas bases de dados? Quais?

ii. A inclusão/exclusão de terminais para os quais há dados lacunares influencia as relações filogenéticas entre os terminais? Justifique.

11.2.2 Dados lacunares em subpartições de POY

Há outras duas formas de dados lacunares que devem ser tratados adequadamente em POY. A primeira delas refere-se aos *gaps* iniciais e finais inseridos na sequências em decorrência do sequenciamento diferencial da região de interesse. Estes *gaps* não representam eventos

de transformação putativas e devem ser removidos ou substituídos por `Ns`. Também existe a possibilidade de que haja regiões em suas sequências onde exista falta de dados resultado do processo de sequenciamento. Nestes casos, é possível – e de certa forma desejável – criar subpartições em suas sequências. O objetivo destas partições é primariamente evitar artefatos analíticos. No entanto, há um outro motivo para a criação de subpartições e esta está relacionada com o desempenho de POY. Considere que quanto maior é o comprimento da sequência, maior é a dificuldade de encontrar soluções ótimas pelos algoritmos disponíveis. Afinal, o algoritmo de Needleman-Wunsch é míope. Giribet [26] notou que a criação de subpartições nos arquivos de dados pode reduzir drasticamente o tempo computacional das análises em POY. Subpartições podem ser criadas em AliView inserindo uma coluna de com um caráter alfabético inexistente em seus dados entre as regiões que você deseja particionar e posteriormente substituir esse caráter por "`#s`" em um editor de texto ou usando o comando `sed`. Onde definir essas partições é arbitrário e o POY pode fazer isso automaticamente (veja o comando `auto_sequence_partition` na documentação de POY. Geralmente, essas partições são inseridas em regiões de alinhamento inequívoco que separam regiões sujeitas a alinhamento ou para as quais há dados lacunares.
O vídeo $\texttt{make}_{p}artitions.mp4ilustracomoissodeveserfeito.Estevdeoestnodiretrioassociadoaestetuto$

11.3 Assignment 6

Neste trabalho você deverá apresentar uma reanálise de algum estudo filogenético já publicado de seu interesse que tenha sido feito com base em dados moleculares passíveis de análise por homologia dinâmica – *i.e.*, sujeitos a alinhamento. Ao selecionar o estudo de interesse, considere os recursos computacionais que possui a sua disposição, pois você já deve ter uma noção do custo computacional envolvido em análises por homologia dinâmica.

a. **Forma de apresentação:** Os resultados deste exercício deverão ser apresentados em um único documento em **formato PDF** contendo duas páginas:

1. A primeira página deve apresentar um texto contendo três itens: *i.* Introdução; *ii.* métodos, e *iii.* resultados e discussão.

i. Introdução: Neste item você deverá apresentar o trabalho que vocês escolheu abordar.

ii. Métodos: Aqui você deverá fazer uma breve descrição dos métodos que você abordou, includino os comandos que definem a estratégia de busca utilizada.

iii. Resultados e Discussão: Finalmente, neste item você deverá fazer uma breve discussão sobre os resultados que você obteve em relação àqueles apresentados pelos autores.

2. Na segunda página você deverá apresentar uma figura e **sua legenda** que sumarizem

seus resultados. Esta figura será avaliada considerando a implementação de todos os recursos gráficos que foram apresentados ao londo do curso.

b. **Nome do arquivo:** Os nomes do arquivo deverão obedecer o seguinte formato: `assignment_06.pdf`.

c. **Data de entrega:** 06/07/2023 até as 14:00 hrs (antes da Aula 12).

d. **Forma de entrega:** Fazer o *upload* em https://forms.gle/nUXWG1xVjinvPVnx8.

11.4 Referências

1. Phillips, A.; Janes, D. & Wheeler, W. C. 2000. Multiple sequence alignments in phylogenetic Anlalysis. *Molecular Phylogenetics and Evolution* **16**(3): 317–330.
2. Giribet, G.; Wheeler, W. C.; & Muona, J. em *Molecular Systematics and Evolution: Theory and Practice* eds. DeSalle, R.; Giribet, G. & Wheeler, W. C., 107–114. Basel: Birkhäuser Verlag, 2002.
3. Wheeler, W. C. 2012. Systematics: a course of lectures. Malaysia: Wiley-Backwell, 2012. 426.
4. 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.
5. Sankoff, D. 1975. Minimal mutation trees of sequence. *SIAM Journal on Applied Mathematics* **28**: 35–42.
6. Wheeler, W. C. 1996. Optimization alignment: the end of multiple sequence alignment in phylogenetics? *Cladistics* **12**: 1–9.
7. Wheeler, W. C. 2001. Homology and the optimization os DNA sequence data. *Cladistics* **17**: S3–S11.
8. Wheeler, W. C. 2003. Implied alignment: a synapomorphy-based multiple-sequence alignment method and its use in cladogram search. *Cladistics* **19**: 261–268.
9. Needleman, S. B. & Wunsch, C. D. 1970. A general method applicable to the search of similatiries in the amino acid sequence of two proteins. *Journal of Molecular Biology* **48**: 443–153.
10. Sankoff, D. & Cedergren, R. J. 1973. A test for nucleotide-sequence homology. *Journal of Molecular Biology* **77**: 159–164.
11. Sankoff, D.; Cedergren, R. J. & Lapalme, G. 1976. Frequency of insertion–deletion, transversion, and transition in the evolutions of 5S ribosomal RNA. *Journal of Molecular Evolution* **7**: 133–139.
12. Wheeler, W. C. 2003. Iterative pass optimization of sequence data. *Cladistics* **19**: 254–260.

13. Grant, T. *et al.* 2006. Phylogenetic systematics of dart-poison frogs and their relatives (Amphibia:Athesphatanura:Dendrobatidae). *Bulletin of the American Museum of Natural History* (299): 1–262.

14. Dikow, T. 2009. A phylogenetic hypothesis for Asilidae based on a total evidence analysis of morphological and DNA sequence data (Insecta: Diptera:Brachycera:Asiloidea). *Organisms, Diversity & Evolution* **9**: 165–188.

15. Jungfer, K.-H. *et al.* 2012. Systematics of spiny-backed treefrogs (Hylidae:Osteocephalus): an Amazonian puzzle. *Zoologica Scripta* **42**(4): 351–380.

16. Caira, J. N.; Marques, F. P. L.; Jensen, K; Kuchta, R & Ivanov, V. 2013. Phylogenetic analysis and reconfiguration of genera in the cestode order Diphyllidea. *International Journal for Parasitology* **43**: 621–639.

17. Pinto-da-Rocha, R; Bragagnolo, C; Marques, F. P. L. & Antunes jr., M. 2014. *Cladistics* **30**: 519–539.

18. Machado, D. J. & Marques, F. P. L. em *Proceedings of the XXXII Meeting of the Willi Hennig Society* 74. Rostock, Germany 2013.

19. Goloboff, P. 1999. Analyzing large data sets in reasonable times: solutions for composite optima. *Cladistics* **15**: 415–428.

20. Wheeler, W. C. *et al.* 2006. Dynamic homology and phylogenetic systematics: a unified approach using POY. New York, Germany: Americam Museum of Natural History, 2006. 284 pp.

21. Weins, J. J. 1998. Does adding characters with missing data increase or decrease phylogenetic accuracy? *Systematic Biology* **47**(4): 625–640.

22. Weins, J. J. 2003. Missing data, incomplete taxa, and phylogenetic accuracy. *Systematic Biology* **52**(4): 528–538.

23. Weins, J. J. 2006. Missing data and the disign of phylogenetic analyses. *Journal of Biomedical Informatics* **39**: 34–42.

24. Weins, J. J. 2011. Missing data in phylogenetic analysis: Reconciling results from simulations and empirical data. *Systematic Biology* **60**(5): 719–731.

25. Hall, T. A. 2013. Missing data and influential sites: Choice of sites for phylogenetic analysis can be as important as taxon sampling and model choice. *Genome Biology and Evolution* **5**(4): 681–687.

26. Giribet, G. 2001. Exploring the behavior of POY, a program for direct optimization of molecular data. *Cladistics* **17**: 60–70.

Tutorial 12

Introdução à verossimilhança

BIZ0433 - INFERÊNCIA FILOGENÉTICA: FILOSOFIA, MÉTODO E APLICAÇÕES.

POR DENIS JACOB MACHADO & FERNANDO MARQUES

Conteúdo

Objetivo

Este tutorial introduz o conceito de verossimilhança como método de inferência e como ele pode ser aplicado como critério de otimalidade em inferência filogenética. Três tipos de verossimilhança são apresentados como critério de otimalidade. Finalmente, o conceito e a prática da seleção de modelos de substituição é apresentado. 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/
```

12.1 Probabilidades

Há várias definições de probabilidades e uma breve consulta a literatura especializada revelará que não há consenso absoluto sobre esse conceito [1]. Independentemente do conceito que você adotar sobre probabilidade, há uma série de axiomas em Teoria de Probabilidades que você deve conhecer – ou relembrar para que nós possamos entender os conceitos de verossimilhança. Neste tutorial iremos adotar a definição "frequencista" de probabilidades. Dentro deste contexto, dado n eventos dentre os quais um conjunto de resultados mutuamente exclusivos (E_i) é observado, à medida em que $n \rightarrow \infty$, dizemos que a probabilidade de E_i (isto é, $P_{(E_i)}$) pode ser expressa por E_i/n. Há alguns axiomas da teoria de probabilidades, ou consequência desses, que você deve ter em mente:

$$0 \leq P_{(E_i)} \leq 1; \tag{12.1}$$

$$P_{(E_i)} = 1 - P_{(\tilde{E}_i)}; \tag{12.2}$$

no qual $P_{(\tilde{E}_i)}$ é a probabilidade de $P_{(E_i)}$ não ocorrer,

$$P_{(E_i ou E_j)} = P_{(E_i \cup E_j)} = P_{(E_i)} + P_{(E_j)}; \tag{12.3}$$

assumindo que E_i e E_j sejam eventos disjuntos exclusivos, e

$$P_{(E_i e E_j)} = P_{(E_i \cap E_j)} = P_{(E_i) * P(E_j)}; \tag{12.4}$$

no qual E_i e E_j são eventos eventos complementares independentes.

Por fim, probabilidades condicionais são expressas da seguinte forma:

$$P_{(E_i|E_j)} = \frac{P_{(E_i \cap E_j)}}{P_{(E_i)}}; \tag{12.5}$$

Neste caso, $P_{(E_i|E_j)}$ denota a $P_{(E_i)}$ ocorrer dado que $P_{(E_j)}$ já ocorreu.

12.2 Princípios de estimativas de verossimilhança máxima

Considere que você pegou uma moeda e jogou cara ou coroa 20 vezes e obteve o seguinte resultado [o mesmo exemplo é dado utilizando dictribuição binomial no Apêndice A de 2]:

Ca Co Co Ca Ca Ca Co Ca Co Co Co Ca Ca Co Ca Co Co Ca Ca Ca

ou seja 11 caras e 9 coroas.

Qual seria a probabilidade desta observação ($P_{(obs)}$)?

Ela seria definida da seguinte forma:

$$P_{(obs)} = P_{(Ca)} * P_{(Co)} * P_{(Co)} * P_{(Ca)} * P_{(Ca)} * P_{(Ca)} * P_{(Co)} * P_{(Ca)} * P_{(Co)} * P_{(Co)} * P_{(Co)} * P_{(Ca)} * P_{(Ca)} * P_{(Co)} * P_{(Ca)} * P_{(Co)} * P_{(Co)} * P_{(Ca)} * P_{(Ca)} * P_{(Ca)}$$

ou seja (veja equação 12.4),

$$P_{(obs)} = P_{(Ca)}^{11} * P_{(Co)}^{9} \tag{12.6}$$

Segundo a fórmula acima, e assumindo que a moeda não apresenta nenhum vício, você teria:

$$P_{(obs)} = 0.5^{11} * 0.5^{9} = 0.00000105415$$

No entanto você assumiu que $P_{(Ca)} = P_{(Co)} = 0.5$. Suponha que você desconheça o valor de $P_{(Ca)}$ ou $P_{(Co)}$. Seria possível estimar esses valores? Estimativas de Verossimilhança Máxima podem ser aplicadas nesse contexto.

Verossimilhança Máxima (L) é definida como:

$$L_{(\theta|obs)} \propto P_{(obs|\theta)} \tag{12.7}$$

onde, a Verossimilhança Máxima (L) de um parâmetro θ dado a observação é proporcional a probabilidade da observação dado um determinado parâmetro. Portanto, Verossimilhança Máxima lhe possibilita estimar o valor de θ que maximize a probabiliade de você observar os dados que observou.

Para aplicar Verossimilhança Máxima em nosso exemplo com caras e coroas, devemos considerar que:

$$P_{(Ca)} + P_{(Co)} = 1$$

portanto,

$$P_{(Co)} = 1 - P_{(Ca)}. \tag{12.8}$$

Se considerarmos que a equação (12.8) e a equação (12.6) teríamos:

$$P_{(obs)} = P_{(Ca)}^{11} * (1 - P_{(Ca)})^{9} \tag{12.9}$$

Considere agora que o parâmetro θ que você gostaria de estimar é o valor de $P_{(Ca)}$ que maximizasse L. O valor de $P_{(Ca)}$ que maximizaria a probabilidade de observar seus dados seria definico como:

$$L_{(P_{(Ca)}|obs)} \propto P_{(obs|P_{(Ca)})} \tag{12.10}$$

Se você tem familiaridade com R, você pode usar o *script* abaixo, disponível no diretório deste tutorial[1], para gerar os gráficos apresentados a seguir manipulando os valores de *head* e *tail*, para caras e coroas, respectivamente.

```
head <- 11    # mude aqui o numero de caras
tail <- 9    # mude aqui o numero de coroas
p <- seq(0,1,by=0.001)
for (i in p){L <- c((p^head)*(1-p)^tail)}
maxL <- max(L)
indexL <- match(maxL,L)
best_p <- p[indexL]
plot(p, L, xlab="P(Ca)", ylab="L", col="blue", frame=F, pch=16,cex=0.5)
text(0.8, maxL, paste("L =", maxL, "\n P(Ca) =", best_p))
```

A Figura 12.1 refere-se à nossa observação inicial e exibe os valores de L associados com as possíveis probabilidades de caras (*i.e.*, $P_{(Ca)}$) – que varia de 0 a 1 em intervalos de 0.001).

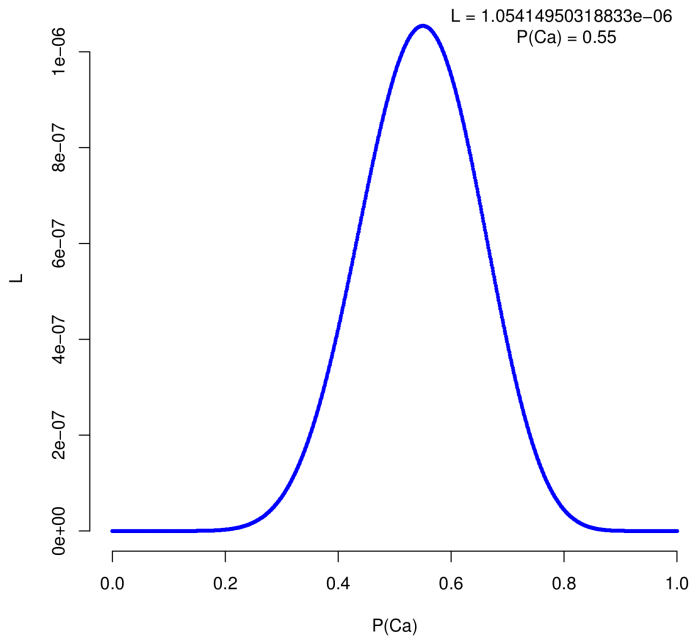


Figura 12.1: Valores de Verossimilhança ($L_{(P_{(Ca)}|obs)}$) em função da probabilidade de obter caras ($P_{(Ca)}$) para 20 eventos dos quais 11 resultaram em caras e 9 em coroas.

Observe que à medida em que $P_{(Ca)}$ se aproxima de 0.5, o valor de L vai aumentando até chegar ao seu máximo. A Verossimilhança Máxima é obtida para $P_{(Ca)} = 0.55$. O que isso significa? De acordo com o conceito de Verossimilhança Máxima, a melhor estimativa de $P_{(Ca)}$ que explica sua

[1] Você pode executar o seguinte comando "`Rscript likelihood_coin.r`" e o script irá gerar o arquivo "`Rplots.pdf`" no qual existe um gráfico exemplificando a estimativa de verossimilhança máxima.

observação (*i.e.*, 11 caras e 9 coroas) é 0.55. Desta forma, o modelo probabilístico que melhor explica sua observação é $P_{(Ca)} = 0.55$ e $P_{(Co)} = 0.45$, o que significa que a moeda aparentemente não apresenta nenhum vício, pois essa diferença é muito pequena.

Exercício 13.1

O que aconteceria se a observação fosse outra, como por exemplo, 3 caras e 17 coroas? A Figura 12.2 exibe os valores de L associados com as possíveis probabilidades de caras para esta observação. Note como esse gráfico difere do primeiro. Você considera que o modelo que melhor explica essa observação sugere que a moeda é viciada? Justifique.

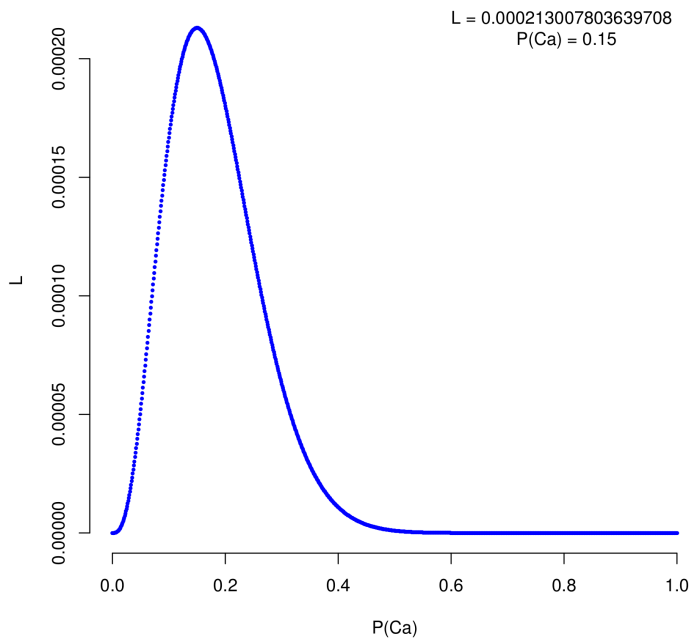


Figura 12.2: Valores de Verossimilhança ($L_{(P_{(Ca)}|obs)}$) em função da probabilidade de obter caras ($P_{(Ca)}$) para 20 eventos dos quais 3 resultaram em caras e 17 em coroas.

Exercício 13.2

O gráfico abaixo representa os valores de Verossimilhança (L) dada a probabilidade de obter caras ($P_{(Ca)}$) em dois ensaios com 20 eventos de cara ou coroa. No primeiro ensaio, uma moeda resultou em 10 caras e 10 coroas (linhas pontilhadas). No segundo ensaio, uma outra moeda resultou em 3 caras e 17 coroas (linhas contínuas). Com base nestas informações, explique como Verossimilhança Máxima é usada para estimar parâmetros.

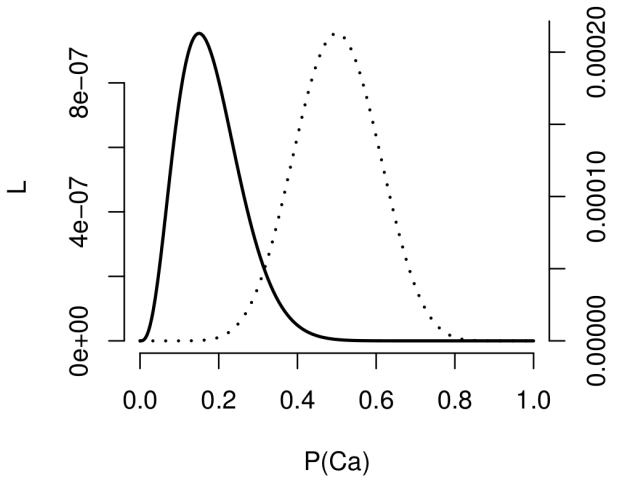


Figura 12.3: Valores de Verossimilhança ($L_{(P_{(Ca)}|obs)}$) em função da probabilidade de obter caras ($P_{(Ca)}$).

12.3 Modelos de substituição

A motivação pela qual Felsenstein [3] propôs o uso de estimativas de verossimilhança como critério de otimalidade em inferência filogenética veio da observação sob dados simulados de que topologias que possuíam comprimentos de ramos desproporcionais não eram recuperadas pelo critério da parcimônia. Desta forma, segundo o autor, a grande virtude do método proposto residia em considerar o comprimento de ramos durante a busca da melhor topologia.

Para considerar o comprimento de ramos é necessário a adoção de um modelo de substituição. De forma geral, o modelo estatístico é a formalização matemática da relação entre variáveis que correspondem a observações potenciais que inclui a descrição das incertezas sobre estas observações devido à variabilidade natural, erros ou informação incompleta. Neste caso em particular, esses modelos descrevem as probabilidades de transformação de caracteres dado sua frequência (π) e o comprimento de ramo (v). O comprimento deste ramo representa o número esperado de substituições por sítio e é definido por $v = 3\alpha * t$; no qual 3α é a taxa total de substituição e t uma unidade de tempo qualquer. Uma vez que taxa e unidade de tempo estão interligados, arbitrariamente convencionou-se que a taxa receberia o valor de **uma substituição** que temos a expectativa que ocorra em **uma unidade de tempo** para cada sítio. Ao fazermos isso, "tempo" (o comprimento de um ramo) é medido em unidades de distância evolutiva (ou ainda, número de substituições esperada por sítio). Essa equivalência é feita assumindo que $\alpha = \frac{1}{3}$, pois teríamos $v = 3(\frac{1}{3})t$, ou seja, $v = t$.

Um dos modelos mais simples de substituição é conhecido como JC69 [4]. Este modelo assume que todas as bases são igualmente frequentes (0.25) e que a taxa de substituição (α) é idêntica

para todas as possíveis substituições cujos eventos obedecem a distribuição de probabilidades de Poisson. Esta distribuição descreve o número de ocorrências de um determinado evento aleatório (estocástico) em um determinado espaço de tempo, assumindo uma taxa média de ocorrência (λ).[2] Desta forma, de acordo com esse modelo,

$$P_{ij}(t) = \frac{1}{4}(1 - e^{-4v/3}) = \frac{1}{4} - \frac{1}{4}e^{-4v/3} \tag{12.11}$$

e

$$P_{ii}(t) = e^{-4v/3} + \frac{1}{4}(1 - e^{-4v/3}) = \frac{1}{4} + \frac{3}{4}e^{-4v/3} \tag{12.12}$$

onde $P_{ii}(t)$ é a probabilidade de não-mudança do estado de caráter durante o tempo t.

Considere por exemplo, a transformação da sequência ancestral `ACGTACGTACGT` para a sequência descendente `ACGTACGTAAAA` dado que v é igual a 0.1. A probabilidade $P_{(ACGTACGTACGT \rightarrow ACGTACGTAAAA|\theta=v=0.1)}$ seria:

$$P_{(ACGTACGTACGT \rightarrow ACGTACGTAAAA|v=0.1)} = \left[\left(\frac{1}{4} - \frac{1}{4}e^{-4(0.1/3)}\right)^3\right] * \left[\left(\frac{1}{4} + \frac{3}{4}e^{-4(0.1/3)}\right)^9\right] \tag{12.13}$$

ou seja,

$$P_{(ACGTACGTACGT \rightarrow ACGTACGTAAAA|v=0.1)} = 0.00001254686 \tag{12.14}$$

No entanto, talvez $v = 0.1$ não seja o melhor valor deste parâmetro para explicar a transformação entre estas duas sequências. Usando o conceito de máxima verossimilhança poderíamos estimar o valore de v que maximizaria $P_{(ACGTACGTACGT \rightarrow ACGTACGTAAAA|\theta=v)}$. O script `likelihood_v.r`, disponível no diretório deste tutorial, computa os valores de $L_{(v|ACGTACGTACGT \rightarrow ACGTACGTAAAA)}$ – como ilustrado na Figura 12.4.

Exercício 13.3

Você considera que valor de v utilizado anteriormente é uma boa estimativa para esse parâmetro? Justifique

[2] Esta distribuição é derivada da distribuição binomial (veja https://www.khanacademy.org/math/probability/random-variables-topic/poisson-process/v/poisson-process-1).

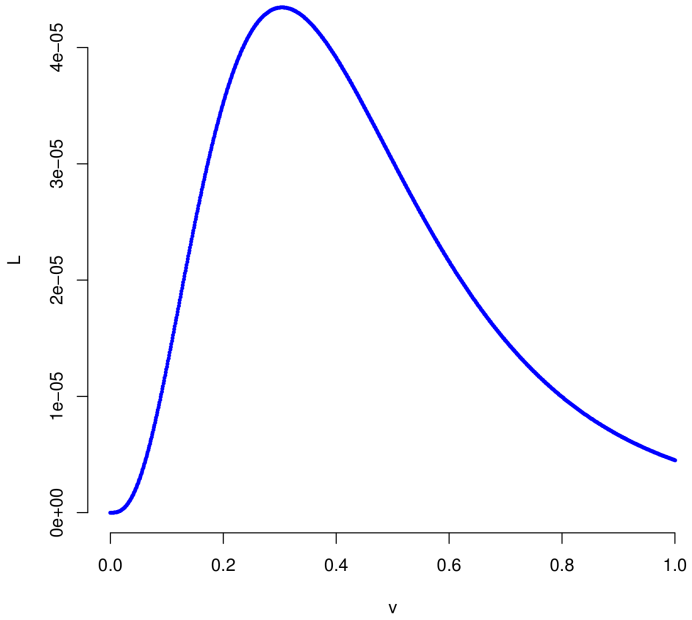


Figura 12.4: Valores de Verossimilhança ($L_{(v|obs)}$) em função do comprimento de ramo v.

12.4 Verossimilhança como critério de otimalidade

Assim como em parcimônia (MP), a busca de árvores sob o critério de máxima verossimilhança ("maximum likelihood", ML) assinala estados ancestrais (dentro de um contexto absoluto ou de médias) tal que a verossimilhança da árvore (hipótese) é maximizada. Para o conjunto de dados D, o modelo de substituição Θ e a topologia T:

$$L_{(T,\Theta|D)} \propto P_{(D|T,\Theta)} \tag{12.15}$$

Nesta expressão, uma determinada T é selecionada de modo que $P_{(D|T,M)}$ é maximizada.

12.4.1 Parâmetros inconvenientes e medidas de máxima verossimilhança

Os parâmetros inconvenientes são todos os demais parâmetros, fora T e D, necessários para calcular $P_{(D|T)}$. Os três parâmetros inconvenientes mais importantes são: (*i*) os parâmetros libres do modelo de substituição, (*ii*) tempo e taxa de transformação nos ramos e (*iii*) distribuição das taxas entre os caracteres. Estes parâmetros são coletivamente chamados θ. Assumindo uma distribuição para os parâmetros inconvenientes $\Phi(\theta \mid T)$, pode-se integrar θ (dentro do espaço de parâmetros Θ) para determinar $P_{(D|T)}$:

$$P_{(D|T)} = \int_{(\theta \in \Theta)} P_{(D|T,\theta)} d\Phi(\theta|T) \tag{12.16}$$

O T que maximize $P_{(D|T)}$ desta maneira é chamado máxima verossimilhança integrada

("maximum integrated likelihood", MIL) [5]. A MIL é igual à máxima probabilidade posterior (do inglês, "maximum a posteriori", MAP) quando a distribuição assumidas a priori ("*priors*") for uniforme ("flat").

Como θ é composto de muitos parâmetros de distribuição desconhecida, uma abordagem para lidar com estes parâmetros é selecionar θ tal que $P_{(D|T,\theta)}$ seja maximizada. Isto equivale à máxima verossimilhança relativa (do inglês "maximum relative likelihood", MRL). A MRL é a metodologia mais usada em análises empíricas. Seu cálculo independe de $P_{(T)}$ e $\Phi_{(\theta|T)}$. Os tipos de MRL estão listados abaixo:

- "Maximum average likelihood" [MAL, 6]: soma sobre todos os possíveis estados dos vértices. Esta forma de verossimilhança é a mais comumente empregada em análises empíricas.
- "Most parsimonious likelihood" (MPL, também conhecida como "ancestral maximum likelihood"): valores e parâmetros específicos são atribuídos para cada vértice. Apesar de se parecer em alguns aspectos com uma análise de parcimônia, MPL e parcimônia não convergem pois todas as taxas aplicadas sobre os ramos serão as mesmas para todos os caracteres.
- "Evolutionary path likelihood" [EPL, 7]: toda sequência de estados de caracter intermediários entre os vértices é especificada de tal modo que a verossimilhança de toda árvore é maximizada. A árvore que maximiza EPM é a árvore mais parcimoniosa.

12.4.2 Calculando $P_{(D|T,\theta)}$

Para um único caráter (x) em uma árvore, a verossimilhança do vértice i (Li) com vértices descendentes j e k será a soma da probabilidade entre xi e cada um de seus descendentes (dado um comprimento de ramo v) multiplicadas por suas respectivas verossimilhanças e somadas para todos os estados. A verossimilhança dos caracteres é multiplicada sobre todo o conjunto de dados para determinar a verossimilhança.

$$L_i(x) = \sum_{i}^{estados} \left[\left(\sum_{x_j} p_{x_i,x_j}(v_j) L_j(x_j) \right) \times \left(\sum_{x_k} p_{x_i,x_k}(L_k)(x_k) \right) \right] \tag{12.17}$$

Usando dados reais, os comprimentos dos ramos são quase sempre desconhecidos e precisam ser estimados. Isto pode ser feito de diversas maneiras mas normalmente depende da probabilidade marginal (mantendo todos os demais parâmetros constantes) de um dado ramo assumindo uma variedade de valores (parâmetro v) e escolhendo um valor ótimo [para mais detalhes, veja 8, Capítulo 11].

O cálculo da MAL de uma árvore é um procedimento heurístico devido ao grande número de parâmetros que devem ser estimados. Assim como em parcimônia, a árvore é obtida assinalando

estados medianos recursivamente. Este procedimento é inicializado assinalando um valor igual a 1 para todos os ramos terminais. A verossimilhança é determinada dos ramos terminais para a raiz, multiplicando-se pelas probabilidades *a priori* dos próprios estados.

$$L_T(x) = \prod_{i=1}^{estados} \pi_i \prod_{\forall u,v \in E} L_{u,v} \tag{12.18}$$

Dado que estes valores são geralmente muito baixos, é geralmente mais conveniente expressá-los em forma logarítmica.

12.4.3 EXEMPLO SIMPLES

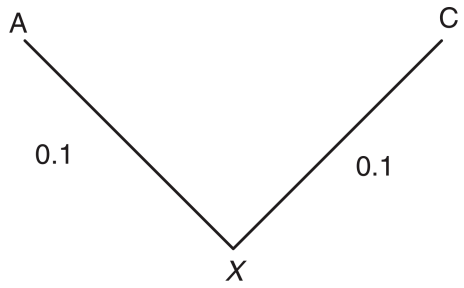
Considere um único nucleotídeo caracterizado sob o model JC69. As probabilidades de ramo serão ficadas em $v = \mu t = 0.1$.

$$f(n) = \begin{cases} \frac{1}{4} + \frac{3}{4}e^{-\mu t} & \text{i} = \text{j} \\ \frac{1}{4} - \frac{1}{4}e^{-\mu t} & \text{i} \neq \text{j} \end{cases} \tag{12.19}$$

Deste modo, as probabilidades de ramo são:

$$f(n) = \begin{cases} 0.929 & \text{i} = \text{j} \\ 0.0238 & \text{i} \neq \text{j} \end{cases} \tag{12.20}$$

Veja na **Figura 12.5** como seria o cálculo da sub-árvore à seguir com ramos terminais contendo os caracteres A e C e parâmetro de ramo 0.1. A verossimilhança média da árvore é $1{,}76 \times 10^6$ ou, em $-log$ (base e), 13,25.



$$
\begin{aligned}
L(x = \text{A}) &= [0.929 \cdot 1.0] \times [0.0238 \cdot 1.0] = 0.0221 \\
L(x = \text{C}) &= [0.0238 \cdot 1.0] \times [0.929 \cdot 1.0] = 0.0221 \\
L(x = \text{G}) &= [0.0238 \cdot 1.0] \times [0.0238 \cdot 1.0] = 0.000566 \\
L(x = \text{T}) &= [0.0238 \cdot 1.0] \times [0.0238 \cdot 1.0] = 0.000566 \\
\text{Total } L(x) &= 0.0453
\end{aligned}
$$

Figura 12.5: Sub-árvore anotada com cálculo de verossimilhança (de Wheeler, 2012: Fig. 11.8).

Exercício 13.4

Qual seria a verossimilhança da sub-árvore da figura acima caso "gaps"(inserções/ deleções, ou "InDels") fossem tratados como um quinto estado de caráter e o parâmetro dos ramos fosse 0.2?

12.4.4 Seleção de modelos e dados lacunares

A adoção do critério de verossimilhança em inferência filogenética requer a escolha de um modelo probabilístico de substituição. Quanto maior é o número de parâmetros livres em um modelo – aqueles são estimados durante a análise–, maior será o ajuste do modelo aos seus dados, no entanto você pode usar parâmetros desnecessários, cuja a inclusão não justifica o ganho no valor de verossimilhança. Desta forma, o procedimento de seleção de modelos, visa adotar o modelo com o menor número de parâmetros livres que de acordo com critérios que visam penalizar a sobre-parametrização de modelos.

A escolha destes modelos deve preceder a análise filogenética e deve ser feita de forma objetiva [veja conceitos e referências em 9]. Neste tutorial nos iremos adotar como critério de escolha de modelos o critério de informação de Akaike corrigido (*Akaike Information Criteria* – AIC_c). Esta métrica não corrigida é computada da seguinte forma:

$$AIC = -2l + 2k \tag{12.21}$$

no qual l é o log da verossimilhança e k é o número de parâmetros livres do modelo – aqueles que são maximizados na função de verossimilhança. Vale ressaltar que o k inclui os parâmetros livres do modelo de substituição [veja Tabela 1 de 9], mais a topologia e seus comprimentos de ramos (*i.e.*, $(t * 2) - 3$, onde t é o número de terminais). Sua correção é necessária quando o número amostral – neste caso número de caracteres n – é pequeno quando comparado com o número de parâmetros livres (*i.e.*, estimados; digamos $\frac{n}{k} < 40$) e é dada pela fórmula:

$$AIC_c = AIC + \frac{(2k(k-1))}{(n-k-1)} \tag{12.22}$$

Em principio, dados lacunares ("missing data") não são um problema para análises de verossimilhança. Estados de ramos terminais podem ser definidos com probabilidade de transformação igual a 1.0 para cada estado observado ou implícito. Porém, diferentes implementações podem diferir na forma de tratar estes dados. É evidente que a implementação de dados lacunares irá afetar as análises. Este problema torna-se ainda mais pernóstico quando INDELS (*i.e.*, inserções e deleções) são tratados como dados lacunares. Este tratamento é problemático, porém matematicamente necessário para manter o cálculo da verossimilhança factível dentro dos limites atuais de tempo e recursos computacionais.

Nos exercícios abaixo você deverá selecionar modelos para dois bancos de dados, incluindo três alinhamentos distintos. O objetivo é verificar como alinhamentos modificam os critérios de escolha de modelos e refletir sobre o uso de AIC_c em análises por verossimilhança máxima.

Exercício 13.5

Neste exercício você deverá fazer três linhamentos distintos para as sequências no arquivo `partition2.fas` (*i.e.*, `partition2aln1.fas`, `partition2aln2.fas` e `partition2aln3.fas`) utilizando `MAFFT` e os seguintes parâmetros de alinhamento: abertura de gaps (`--op`) 3.06, 1.53 e 0.123 no qual o valor dos gaps de extenção (`--ep`) será fixado em 0.123 (veja Seção 8.2.5 do Tutorial 8 caso tenha dúvidas).

i. Qual valor de "gap opening" resultou em um alinhamento com mais gaps? Porquê?

Existem algumas ferramentas para a seleção de modelos em análises filogenéticas, sendo jModelTest 2 [10, 11] uma das mais tradicionais e a leitura do manual deste programa pode ser

muito instrutiva. No entanto, o programa IQ-TREE [12] – um aplicativo relativamente recente – tornou o jModelTest 2 obsoleto na minha opinião. O IQ-TREE possibilita a avaliação de um maior número de modelos e é muito mais rápido e robusto que o jModelTest 2.

O IQ-TREE utiliza o algoritmo do ModelFinder [13] para a seleção de modelos e maiores detalhes sobre a seleção de modelos em IQ-TREE podem ser encontrados nos tutoriais do programa – na seção "Choosing the right substitution model" e/ou na documentação do programa – na seção "Substitution models".

Exercício 13.6

Neste exercício você deverá computar os parâmetros de seleção de modelos para o arquivo `partition1.fas` – que não requer alinhamento – e os três linhamentos distintos para as sequências no arquivo `partition2.fas` (*i.e.*, `partition2aln1.fas`, `partition2aln2.fas` e `partition2aln3.fas`) que você executou no exercício anterior. Adicionalmente, você fará o mesmo para os bancos de dados concatenados.

O IQ-TREE deverá estar instalado em seu computador, caso não esteja, verifique a versão apropriada para seus sistema na página de download do IQ-TREE. Uma vez instalado, a seleção de modelos no IQ-TREE pode ser obtida pela seguinte linha de comando:

```
$ iqtree2 -s partition1.fas -m TEST
```

Com esse comando, o IQ-TREE irá avaliar 88 modelos de substituição, equivalentes ao que seria examinado pelo jModelTest 2[3]. Após a execução do comando acima, os dados necessários para completar a Tabela 12.1 estarão no log da execução (*e.g.*, arquivo `partition1.fas.log`).

Tabela 12.1: Seleção de modelos pelo critério de AIC_c.

Dataset	-lnL	k	AIC_c	Model
`partition1.fas`				
`partition2aln1.fas`				
`partition2aln2.fas`				
`partition2anl3.fas`				
`partition1+partition2aln1.nex`[4]				
`partition1+partition2aln2.nex`				
`partition1+partition2anl3.nex`				

[3] A opção "`-m MFT`" explora modelos adicionais implementados em ModelFinder, consulte a documentação de IQ-0TREE para maiores detalhes.

[4] Você deverá utilizar o `sequencematrix` para concatenar os dados (veja seção 7.3.1 do Tutorial 7 e exportar os dados no formato `NEXUS` para `GARLI`.

Exercício 13.7

Com base nos exercícios acima responda:

i. Modelos diferentes podem ser selecionados para diferentes alinhamentos dos mesmos dados?

ii. Considerando os três alinhamentos para o arquivo `partition2.fas`, você teria algum critério para escolher qual deles deveria ser submetido à análise filogenética?

iii. Qual modelo de substituição você selecionaria para a análise do arquivo `partition1+partition2aln2.nex`? Justifique[5].

iv. Os dados que você analisou são os mesmos da seção 9.2.3 do Tutorial 9. Naquela ocasião, havia um outro conjunto de dados morfológicos (`partition3.tnt`) que se referia aos mesmos táxons. O que seria necessário para incorporá-lo em uma análises simultânea juntamente com os demais dados que você estimou o melhor parâmetro de substituição em uma análise sob o critério de verossimilhança máxima?

12.5 Referências

1. Thacker, N. A. Tutorial: Defining probability for Science. Acesso em: 2 jul. 2014. 2014. http://tree.bio.ed.ac.uk/.
2. Anderson, D. R. 2008. Model based inferences in the life sciences: a primer on evidence. Fort Collins, CO: Springer, 2008.
3. Felsenstein, J. 1978. Cases in which parsimony or compatibility methods will be positively misleading. *Systematic Zoology* **27**(4): 783–791.
4. Jukes, T. H. & Cantor, C. R. em *Mammalian protein metabolism.* ed. Munro, H. N. New York: Academic Press, 1969.

[5] Considere o modelo sugerido para cada uma das partições individualmente.

5. Steel, M & Penny, D. 2000. Parsimony, likelihood, and the role of models in molecular phylogenetics. *Molecular biology and evolution* **17**(6): 839–50.

6. Barry, D & Hartigan, J. 1987. Statistical Analysis of Hominoid Molecular Evolution. *Statistical Science* **2**(2): 191–207.

7. Farris, J. 1973. A Probability Model for Inferring Evolutionary Trees. *Systematic Biology* **22**(3): 250–256.

8. Wheeler, W. 2012. Systematics: A Course of Lectures. First. Oxford: Wiley-Blackwell, 2012, 426. ISBN: 9780470671702.

9. Darriba, D. & Posada, D. jModelTest 2.0 v0.1.1. http://code.google.com/p/jmodeltest2/. 2015.

10. Darriba, D.; Taboada, G. L.; Doallo, R & Posada, D. 2003. jModelTest 2: more models, new heuristics and parallel computing. *Nature Methods* **9**: 772.

11. Guidon, S & Gascuel, O. 2012. A simple, fast and accurate method to estimate large phylogenies by maximum-likelihood. *Systematic Biology* **52**: 696–704.

12. Nguyen, L. T.; Schmidt, H. A.; von Haeseler, A & Minh, B. Q. 2015. IQ-TREE: A fast and effective stochastic algorithm for estimating maximum likelihood. *Molecular Biology and Evolution* **32**: 268–274.

13. Kalyaanamoorthy, S; Minh, B. Q.; Wong, T. K. F.; von Haeseler, A & Jermiin, L. S. 2017. ModelFinder: fast model selection for accurate phylogenetic estimates. *Nature Methods* **14**: 587–589.

Tutorial 13

Inferência por Verossimilhança Máxima: homologia estática e dinâmica

BIZ0433 - INFERÊNCIA FILOGENÉTICA: FILOSOFIA, MÉTODO E APLICAÇÕES.

Conteúdo

Objetivo

Este tutorial apresenta implementações de busca sub o critério de verossimilhança máxima (**ML**) dentro do contexto de homologia estática e dinâmica. O tutorial começa com a apresentação do programa GARLI e como ele pode ser utilizado para análises filogenéticas de homologia estática considerando um único modelo de substituição ou distintos modelos para diferentes partições de dados. Em seguida o tutorial apresenta o uso de POY em análises de **ML** dentro do contexto de homologia estática e dinâmica. Para POY, o tutorial explora a implementação de *Maximum Avarege Likelihood* (MAL) e *Most Parsimonious Likelihood* (MPL). Adicionamente, este tutorial ilustra como POY pode ser utilizado para seleção de modelos de verossimilhança, bem como os modelos implementados atualmente no programa. Este tutorial deve ser considerado como um elemento introdutório para os conceitos e implementações dos métodos abordados. Maiores detalhes sobre o uso deste critério em análises filogenéticas, a forma de implementação de estratégias heurísticas de buscas mais agressivas, e como aplicar o métodos em diversos caracteres genotípicos e fenotípicos podem ser encontrados na documentação de POY e na literatura referenciada no mesmo. 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/
```

13.1 Verrossimilhança: homologia estática em GARLI

No tutorial anterior exploramos os conceitos relacionados com estimativas de verossimilhança máxima de parâmetros, cálculo de verossimilhança para reconstruções e topologias e seleção de modelos. Desta forma, ficou evidente que o processo lógico de inferência filogenética é o mesmo dentro deste critério de otimalidade, ou seja, topologias são avaliadas quanto ao seu mérito por medidas de verossimilhança e a tolologia que maximiza a probabilidade de obter os dados é selecionada – da mesma forma que fazemos com o critério de parcimônia.

Neste componente do tutorial iremos explorar os aspectos práticos de inferência sob o critério de verossimilhança máxima em GARLI [1]. Há uma série de outros programas disponíveis para o mesmo propósito (*e.g.*, IQ-TREE, RAxML, PhyML, PAUP*, entre outros), mas considero o GARLI um dos mais versáteis disponível atualmente[1].

O versatilidade de GARLI está na forma como ele é executado. O programa requer um arquivo de configuração – usando a mesma lógica que você já usou quando executou o YBIRÁ – no qual você define os parâmetros de análise, bem como o arquivo de dados e sufixos de saída em um arquivo de configuração que é lido e executado pelo programa. O exemplo abaixo mostra o conteúdo do arquivo de configuração padrão de GARLI:

```
[general]
datafname = arquivo_de_dados.nex
ofprefix = minha_analise
streefname = stepwise
attachmentspertaxon = 10
constraintfile = none
searchreps = 100
outgroup = 1

outputeachbettertopology = 0
outputcurrentbesttopology = 0

enforcetermconditions = 1
genthreshfortopoterm = 5000
scorethreshforterm = 0.001
significanttopochange = 0.01

writecheckpoints = 0
restart = 0

randseed = -1
availablememory = 2000
logevery = 10
saveevery = 100
refinestart = 1
outputphyliptree = 0
outputmostlyuselessfiles = 0
collapsebranches = 1

#linkmodels means to use a single set of model parameters for all subsets.
linkmodels = 1
```

[1] Em comparação ao IQ-TREE, GARLI parece explorar mais adequadamente o espaço de topologias, o que pode ser desejável em alguns casos. No entanto, IQ-TREE permite explorar modelos mais complexos de substituição e particionamento de dados e apresentar tempo computacional menor. Caso tenham interesse por esse critério de otimização, sugiro a consulta da documentação do IQ-TREE.

```
# subsetspecificrates means to infer overall rate multipliers for each data subset.
# This is equivalent to *prset ratepr=variable* in MrBayes.
subsetspecificrates = 1
# set for single model, different subset rates (like site-specific rates model in PAUP*)

[model0]
#JC
#number of free parameters: 0
datatype = nucleotide
ratematrix = (a a a a a a)
statefrequencies = equal
ratehetmodel = none
numratecats = 1
invariantsites = none

[master]
bootstrapreps = 0

nindivs = 4
holdover = 1
selectionintensity = 0.5
holdoverpenalty = 0
stopgen = 5000000
stoptime = 5000000

startoptprec = 0.5
minoptprec = 0.01
numberofprecreductions = 2
treerejectionthreshold = 20.0
topoweight = 0.01
modweight = 0.002
brlenweight = 0.002
randnniweight = 0.1
randsprweight = 0.3
limsprweight =  0.6
intervallength = 100
intervalstostore = 5

limsprrange = 6
meanbrlenmuts = 5
gammashapebrlen = 1000
gammashapemodel = 1000
uniqueswapbias = 0.1
distanceswapbias = 1.0

resampleproportion = 1.0
inferinternalstateprobs = 0
```

Este arquivo de configuração possui os parâmetros básicos de uma análise que considera um único modelo de substituição para sequências nucleotídicas. Os detalhes sobre cada uma dessas linhas não serão apresentados neste tutorial e o leitor deve consultar a documentação do programa para maiores detalhes. Para os propósitos deste tutorial, iremos comentar apenas algumas linhas, justamente aquelas que são frequentemente modificadas à cada análise:

i. Todos os arquivos de configuração do GARLI são divididos em três seções, `[general]`, `[model#]` e `[master]`.

ii. O comando "`datafname`", linha 2, especifica o arquivo de entrada. O GARLI aceita

arquivos de dados no formato FASTA ou NEXUS (formato de PAUP*). No entanto, você verá que para algumas análises é necessário o uso do segundo formato. Neste caso, o uso do `sequencematrix` é recomendável, uma vez que esse programa tem uma opção para exportar seus dados o formato simplificado de NEXUS compatível com o GARLI (veja Tutorial 12).

iii. O comando "`ofprefix`", linha 3, define o prefixo dos arquivos que o GARLI escreverá no diretório de execução. Use um nome curto que lhe informe a análise que está fazendo.

iv. O comando "`searchreps`", linha 7, define o número de réplicas que você deverá fazer em sua análise.

v. Linhas 30 a 34 especificam como os modelos serão serão considerados pelo GARLI. Neste arquivo de configuração, GARLI irá considerar um único modelo para a partição com uma única taxa para o modelo.

vi. Linhas 36 a 44, especifica o modelo de substituição a ser utlizado; nesse caso o modelo JC69 [2]. Os modelos disponíveis em GARLI estão no arquivo `garli_models.txt` no diretório deste tutorial. O modelo deve ser selecionado anteriormente, veja seção 12.4.4 do Tutorial 12.

13.1.1 GARLI: SINGLE MODEL

Iremos iniciar nossas análises em GARLI considerando um único modelo para os dados, independentemente se eles contém uma ou mais partições. Uma vez editado o arquivo de configuração do GARLI, para iniciar a análise execute:

```
alan@turing:~$ garli garli.conf
```

Por *default*, o GARLI espera encontrar no diretório de execução um arquivo chamado "`garli.conf`". Se esse arguivo existe, basta executar:

```
alan@turing:~$ garli
```

No entanto, exceto em execuções paralelas, ou seja, utilizando computadores de auto desempenho ou multi-processados, é recomendável atribuir nomes distintos para os arquivos de configuração. No diretório deste tutorial por exemplo, há o arquivo `garli_single.conf`. A execução deste arquivo seria:

```
alan@turing:~$ garli garli_single.conf
```

Exercicio 13.1

Execute a linha de comando acima.

A execução desta análise gera os seguintes arquivos de saida:

```
-rw-rw-r- 1 alan alan     4357 Jun  5 15:36 part1.best.all.tre
-rw-rw-r- 1 alan alan      766 Jun  5 15:36 part1.best.tre
-rw-rw-r- 1 alan alan   325906 Jun  5 15:36 part1.log00.log
-rw-rw-r- 1 alan alan   108822 Jun  5 15:36 part1.screen.log
```

O conteúdo destes arquivos são os seguintes:

1. `part1.best.all.tre`: Contém as topologias ótimas encontradas em cada uma das réplicas executadas.
2. `part1.best.tre`: Contém as melhores topologias encontradas na réplica que resultou na melhor topologia.
3. `part1.log00.log`: Contém o *log* da execução.
4. `part1.screen.log`: Contém o `sdt.err` do GARLI, ou seja, as informações de saida que o programa escreve durante a análise.

Este último arquivo possui algumas informações que devemos chamar sua atenção. Se você executar a linha de comando:

```
alan@turing:~$ head -n 50 part1.screen.log
```

Você deverá obter:

```
################################################
PARTITIONING OF DATA AND MODELS
GARLI data subset 1
CHARACTERS block #1 ("Untitled DATA Block 1")
Data read as Nucleotide data,
modeled as Nucleotide data
Summary of data:
       10 sequences.
       62 constant characters.
       48 parsimony-informative characters.
       10 uninformative variable characters.
       120 total characters.
       61 unique patterns in compressed data matrix.
Pattern processing required < 1 second
################################################
```

Os dados acima sumariam as informações de sua matriz de caracteres. Observe que dos 120 caracteres presentes em sua matriz, apenas 61 deles foram condiderados durante a análise. Se você executar a seguinte linha de comando:

```
alan@turing:~$ tail -n 50 part1.screen.log
```

Você deverá obter:

```
Completed 10 replicate search(es) (of 10).

NOTE: Unless the following output indicates that search replicates found the
same topology, you should assume that they found different topologies.
Results:
Replicate 1 : -685.0223
Replicate 2 : -689.6789
Replicate 3 : -689.6789        (same topology as 2)
Replicate 4 : -686.7873
Replicate 5 : -684.7616        (same topology as 4)
Replicate 6 : -684.7616        (same topology as 4)
Replicate 7 : -689.6789        (same topology as 2)
Replicate 8 : -684.7039 (best)
Replicate 9 : -685.0222        (same topology as 1)
Replicate 10 : -689.6789       (same topology as 2)

Parameter estimates across search replicates:
```

	r(AC)	r(AG)	r(AT)	r(CG)	r(CT)	r(GT)	pi(A)	pi(C)	pi(G)	pi(T)	alpha	pinv
rep 1:	1	46.32	1	1	46.32	1	0.066	0.039	0.330	0.565	0.175	0.355
rep 2:	1	8.028	1	1	8.028	1	0.071	0.040	0.323	0.566	0.159	0.000
rep 3:	1	8.032	1	1	8.032	1	0.071	0.040	0.323	0.566	0.159	0.000
rep 4:	1	12.68	1	1	12.68	1	0.068	0.040	0.328	0.565	0.292	0.363
rep 5:	1	30.39	1	1	30.39	1	0.066	0.039	0.330	0.566	0.076	0.043
rep 6:	1	30.41	1	1	30.41	1	0.066	0.039	0.330	0.566	0.076	0.043
rep 7:	1	8.036	1	1	8.036	1	0.071	0.040	0.323	0.566	0.159	0.000
rep 8:	1	35.09	1	1	35.09	1	0.066	0.039	0.330	0.566	0.073	0.040
rep 9:	1	46.43	1	1	46.43	1	0.066	0.039	0.330	0.565	0.175	0.355
rep 10:	1	8.015	1	1	8.015	1	0.071	0.040	0.323	0.566	0.159	0.000

```
Treelengths:

TL

rep 1:  482.826

rep 2:  8.665

rep 3:  8.673

rep 4:  32.616

rep 5:  544.587

rep 6:  544.992

rep 7:  8.669

rep 8:  707.949

rep 9:  483.584

rep10:  8.662

Saving final trees from all search reps to part1.best.all.tre

Saving final tree from best search rep (#8) to part1.best.tre
```

Estas linhas sumarizam seus resultados após 10 réplicas, dentre as quais a réplica 8[2] resultou na melhor solução com lnL -684.7039. Este arquivo também informa as estimativas dos parâmetros em cada uma das réplicas, tais como as taxas das diversas categorias de subistituição (*i.e.*, `r(AC)`, `r(AG)`, `r(AT)` e etc), as frequências das bases (*i.e.*, `pi(A)`, `pi(C)`, `pi(G)` e `pi(T)`), o parâmetro α da distribuição gama (Γ) e a porcentagem dos sítios invariáveis (I). Essa tabela é resultado dos parâmetros livres do modelo $HKY + I + \Gamma$ adotado na análise (veja arquivo de configuração).

Exercicio 13.2

Neste exercício você deverá fazer uma análise sob o critério de verossimilhança máxima em GARLI para os arquivos `partition1+partition2aln1.nex`, `partition1+partition2aln2.nex` e `partition1+partition2aln2.nex` de acordo com os modelos que você selecionou para seus respectivos dados concatenados no Tutorial 12. Para executar esse exercício você deverá:

1. Editar três os arquivos de configuração de GARLI, um para cada arquivo de dados.
2. Executar o GARLI para cada um dos arquivos de configuração.
3. Sumarizar seus resultados na Tabela 13.1, no qual n é o número de caracteres em seu alinhamento, k é o número de parâmetros livres[3] e o AIC_c deve ser computado de acordo com as equações 12.21 e 12.22 do Tutorial 12[4].

Tabela 13.1: Análise de modelo único em GARLI.

Dataset	lnL	n	k	AIC_c	Model
`partition1+partition2aln1.nex`					
`partition1+partition2aln2.nex`					
`partition1+partition2anl3.nex`					

13.1.2 GARLI: PARTITION MODEL

O GARLI permite que diferentes modelos sejam aplicados à distintas partições de dados. Embora isso seja mais uma das características que conferem versatilidade ao programa, considere que nem sempre – dentro dos critérios de seleção de modelos em estimativas de verossimilhança máxima – a adoção de diferentes modelos individuais confere melhores índices à sua análise. Este tutorial

[2] Seus resultados podem não ser idênticos a este, pois isso depende de aleatorização, mas devem obedecer a mesma lógica.
[3] Lembre-se que os parâmetros estimados pelo modelo compreende, a topologia, os comprimentos de cada um dos ramos desta topologia – cujo número é dado pela função $(2 * t) - 3$, no qual t é o número de terminais), e os parâmetros livres do modelo de substituição.
[4] O script `calculate_aicc.py` pode ser utilizado para obter essa métrica, basta executar *script*.

irá explorar a implementação dessas análises de forma superficial. Maiores detalhes sobre esse tipo de análise podem ser obtidos em GARLI partition model documentation[5].

Neste tipo de análise, o GARLI requer que os dados estejam no formato NEXUS. Isso é necessário, pois estes arquivos deverão conter um bloco de instrução definindo as partições. Considere por exemplo o arquivo `partition1+partition2aln1.nex`. Ele é a junção dos arquivos `partition1.fas` + `partition2aln1.fas` do Tutorial 12. O primeiro arquivo possui 120 caracteres, ao passo que o alinhamentdo do segundo resultou em 99 caracteres. Portando, no arquivo concatenado `partition1+partition2aln1.nex`, os caracteres 1 a 120 referem-se à primeira partição, ao passo que os caracteres 121 a 219 referem-se à segunda partição. Sendo este o caso, a primeira etapa de preparação para a análise requer a edição do arquivo `partition1+partition2aln1_garli.nex` para que contenha as seguintes linhas:

Arquivo texto 13.1: partition1+partition2aln1_garli.nexlabel

```
BEGIN SETS;
        CHARSET p1 = 1-120;
        CHARSET p2 = 121-219;
        charpartition partitions = s1:p1, s2:p2;
END;
```

As linhas 1 e 4 são usadas para abrir e fechar o bloco que define as partições, respectivamente. São definidos dois conjuntos de caracteres, `p1` e `p2`, o primeiro de 1 a 120 e o segundo de 121 a 219, linhas 2 e 3. Na linha 4, definimos os `submodels` de GARLI, `s1` e `s2`, para as partições `p1` e `p2`, respectivamente. Note que GARLI usa sequencialmente `s1`, `s2`, `s3` e etc para definir os `submodels`. No entanto, no arquivo de configuração, a sequência de modelos inicia-se no 0; portanto, `[model0]`, `[model1]`, `[model2]` e etc. Este arquivo foi editado e renomeado como `partition1+partition2aln1_garli.nex`

Vejamos como ficaria o arquivo de configuração para este arquivo. Considere que, ao selecionar o modelo para os arquivos `partition1.fas` e `partition2aln1.fas`, o jModelTest selecionou o modelo de acordo com o AIC_c. Os modelos selecionados foram $HKY + I + \Gamma$ e $HKY + \Gamma$, respectivamente. Desta forma, esses modelos foram implementados no arquivo `garli_p1+2aln1.conf` – verifique o conteúdo deste arquivo. Observe que além de especificar dois modelos, as linhas 34 a 38 deste arquivo foram modificadas para que o programa considere modelos distintos para cada uma das partições.

Os resultados desta análise estão no diretório do tutorial. A topologia seleciona, com lnL -1475.1010, foi recuperada na réplica #4 e está escrita no arquivo `results_p1+p2aln1.best.tre`. Considere que o particionamento dos modelos confere a essa análise um novo atributo: seu modelo inclui dois modelos de substituição distintos. Para avaliar esse modelo, é necessário calcular o AIC_c considerando esses dois modelos de substituição. Recorde que o AIC_c é calculado pela seguinte equação:

[5] Veja https://molevol.mbl.edu/index.php/Garli_using_partitioned_models

$$AIC_c = (-2l + 2k) + \frac{(2k(k-1))}{(n-k-1)}$$

Nesta equação, k é o termo que talvez seja menos intuitivo. Como já foi definido, o valor de k é a soma de todos os parâmetros estimados pelo modelo. Ele inclui, a topologia (1), os comprimentos dos ramos ($(2 * t) - 3$, onde t é o número de terminais; portando, $(2 * 10) - 3 = 17$), e o número de parâmetros livres dos modelos de substituição ($6 + 5 = 11$; veja comentários nos modelos implementados no arquivo `garli_p1+2aln1.conf`). Portanto, o valor de k para esta análise seria $1 + 17 + 11 = 29$.

Calculado o valor de k, o cálculo do AIC_c é feito da seguinte maneira:

$$AIC_c(p1 + 2aln1) = (-2 * -1475.1010 + 2 * 29) + \frac{(2*29(29-1))}{(219-29-1)}$$

$$AIC_c(p1 + 2aln1) = (2950.202 + 58) + \frac{1624}{192}$$

$$AIC_c(p1 + 2aln1) = 3008 + 8.4583 = 3016.6603$$

Exercicio 13.3

Neste exercício você deverá fazer uma análise sob o critério de verossimilhança máxima em GARLI para os arquivos `partition1+partition2aln2.nex` e `partition1+partition2aln3.nex` assumindo modelos distintos para cada partições de acordo com os modelos que você selecionou os conjuntos de dados individuais no Tutorial 12. Para execurar esse exercício você deverá:

1. Editar os arquivos de dados para implementar os blocos de partições.
2. Editar dois os arquivos de configuração de GARLI, um para cada arquivo de dados.
3. Executar o GARLI para cada um dos arquivos de configuração.
4. Sumarizar seus resultados na Tabela 13.2.
5. Responder as perguntas abaixo:

Tabela 13.2: Análise de modelo particionado em GARLI.

Dataset	lnL	n	k	AIC_c	Model
`partition1+partition2aln1.nex`	-1475.1010	219	29	3016.6603	$HKY + I + \Gamma / HKY + \Gamma$
`partition1+partition2aln2.nex`					
`partition1+partition2anl3.nex`					

1. De acordo com o critério de AIC_c, qual análise você selecionaria? Justifique.

2. Seu resultado é dependente do critério de otimalidade? Justifique.

13.2 Verrossimilhança: homologia dinâmica em POY

Para dados sujeitos a alinhamento[6], a implementação de análises de homologia dinâmica também é possível. O programa POY permite a implementação dessas análises. A análise dos dados moleculares sob verossimilhança em POY, considerando dados pré-alinhados (homologia estática) ou não (homologia dinâmica), pode ser significativamente mais demorada do que análises sob o critério de otimalidade de parcimônia. Portanto, é necessário adotar algumas estratégias que visem reduzir o tempo computacional sob verossimilhança. Uma das estratégias mais efetivas – embora com algumas deficiências – é iniciar a análise com parâmetros de busca menos complexos, adotando até mesmo parcimônia como critério de otimalidade inicial, e coletar topologias que seriam refinadas sob o critério de verossimilhança no qual se adota procedimentos mais complexos de cômputo até que se atinja o nível de agressividade de busca heurística desejado. Veja abaixo alguns componentes – ou procedimentos – que podem ser manipulados durante sua análise dentro do contexto exposto acima:

i. **Topologias candidatas sob o critério de parcimônia:**

Esta etapa da análise começa com a construção e busca inicial (de complexidade arbitrária) sob parcimônia antes de adotar o critério de verossimilhança na escolha de modelos e diagnose de topologias. Este procedimento economiza tempo, evitando RAS e refinamento (*i.e.*, TBR, SPR, *tree-fusing*, entre outros) sob o critério de verossimilhança – o que demanda enorme quantidade de cálculo. Um problema potencial desta estratégia heurística é que uma ou mais topologia considerada ótima sob o critério de parcimônia pode estar longe no espaço de árvores de uma topologia considerada ótima sob o critério de verossimilhança. Por esse motivo, é recomendável que sejam adotadas estratégias de refinamento (via algoritmos de rearranjo – *e.g.*, TBR – ou até mesmo algoritmos genéticos – *e.g., tree-fusing*) após a adoção do critério de verossimilhança.

ii. **Estimativa aproximada (grosseira) de parâmetros:**

A granularidade da estimativa dos parâmetros do modelo pode ser manipulada durante a análise. Por exemplo, durante os rearranjos sob o critério de verossimilhança, todos os

[6] Texto baseado na documentação de POY [3]

comprimentos de ramos (independentemente da distância entre a quebra e a reconexão) são re-otimizados, o mesmo ocorrendo para os parâmetros do modelo a cada rearranjo. O tempo de execução pode ser minimizado executando a otimização do modelo somente se o custo de um rearranjo está dentro de um número limite em comparação ao melhor custo encontrado até o momento, ou pela otimização do comprimento de ramos em determinadas condições – distância da quebra e da reconexão em cada rearranjo. Esta estratégia é adotada, por exemplo, pelo programa RAxML - Randomized Axelerated Maximum Likelihood [4] que inicia as buscas heurísticas adotando o modelo GTRCAT [7] e durante o processo de refinamento adota o modelo GTRGAMMA que otimiza mais agressivamente os parâmetros do modelo.

iii. **Granularidade na precisão de cálculos (*Floating point granularity*):**

A precisão das casas decimais pode ser diminuida durante os cálculos para limitar o tempo gasto em otimizar os valores dos parâmetros durante os rearranjos ou até mesmo durante a transformação dos caracteres em verossimilhança. A escolha da granularidade atua – limitando – na precisão dos cálculos e no número de iterações realizadas durante o processo de estimativa de parâmetros. Um problema potencial desta estratégia heurística é que a redução da precisão (*coarse granularity*) pode afetar negativamente as análises para as quais várias topologias e/ou comprimentos ramos são muito similares, ou até igualmente ótimos existam. Adicionalmente, os índices de verossimilhança – ou *likelihood scores* – sob baixa precisão não são comparáveis com aqueles obtidos em outros programas que adotam o mesmo critério de otimalidade. Neste caso, a otimização completa deve ser realizada na topologia final.

iv. **Cronograma de Otimização:**

Essa estratégia manipula a etapa na qual é aplicado maior rigor na otimização dos parâmetros do modelo. Por exemplo, sob o rearranjo tradicional sob o critério de verossimilhança, todos os comprimentos de ramo (independentemente da distância entre quebra e reconexão) e os parâmetros do modelo são recalculados para cada topologia. Pode-se diminuir o tempo computacional especificando quando tal rigor deve ser aplicado; otimizando os parâmetros do modelo somente se o custo de um rearranjo está dentro de um limite pré-especificado de acordo com a topologia ótima obtida até então, ou otimizando os comprimentos dos ramos dentro de uma distância específica entre quebra e reconexão para cada rearranjo. O problema potencial com esta estratégia heurística é definir a priori quando este rigor deve ser aplicado e é bem provável que estas propriedades são altamente dependente dos dados em mãos.

v. **Estratégias de rearranjo:**

O número de topologias visitadas durante a etapa de refinamento por algoritmos de rearranjo (que variam entre NNI e TBR) pode ser restringido nos estágios iniciais de busca. Esta

[7] Esse é um algoritmo de aproximação, veja http://sco.h-its.org/exelixis/resource/download/NewManual.pdf

abordagem minimiza o número de cálculos durante os rearranjos, enquanto que o aumento da granularidade restringe o tempo gasto em um determinado cálculo. Uma vez que topologias foram selecionadas e supostamente estejam perto da solução ótima, pode-se proceder com buscas e estimativa de parâmetros mais agressivas. O problema potencial desta estratégia é similar ao uso de topologias iniciais estimadas pelo critério de parcimônia e o usuário pode ficar restrito a um ótimo local limitado pelo espaço explorado durante o refinamento por algoritmos de rearranjo.

Existem outras possibilidades de estratégias heurísticas, incluindo alternância entre os critérios de otimalidade em caracteres estáticos (transformações alternadas entre critérios de verossimilhança e parcimônia para caracteres estáticos), entre as variações de um mesmo critério de otimalidade (entre MPL e MAL), ou entre premissas para caracteres (entre quatro e cinco estados de caráter para um determinado modelo).

13.2.1 Formas de implementação

O POY permite o uso de dois critérios de verossimilhança em inferência filogenética. A primeira delas seria a *Maximum Average Likelihood* (MAL) na análise de dados quantitativos com alfabetos de qualquer tamanho (*i.e.*, sequências nucleotídicas, aminoácios, entre outros) para **sequências alinhadas**[8], considerando *gaps* como dados lacunares (*i.e., missing data*) ou como um quinto estado de caráter. A segunda seria a *Most Parsimonious Likelihood* (MPL) que pode ser aplicada aos mesmos tipos de dados, além de sequências não alinhadas (busca heurística MPL/DO). Adicionalmente, o POY avalia e permite a aplicação de uma série de modelos de substituição (14) além de considerar INDELs (*i.e.*, inserções e deleções) nesses modelos.

13.2.2 Seleção de modelos de verossimilhança em POY

Como discutido no Tutorial 12, este prodedimento define e seleciona o modelo estocástico de evolução (modelo de substituição) que otimiza a função de verossimilhança. POY, como na maioria dos programas que implementa esse critério de otimalidade, aplica um modelo homogêneo que estima as taxas de substituição utilizando uma única matriz de probabilidades ao longo de toda topologia. A adoção do critério de verosimilhança requer a adoção de um modelo de substituição de caracteres. Toda análise de verossimilhança requer justificativa para a adoção de modelos. Estas justificativas residem na aplicação de critérios (*e.g.*, LRT, AIC, AICc, BIC, entre outros) que visam avaliar a relação entre nível de parametrização do modelo e a otimização da função de verossimilhança [veja breve discussão e referências em 5; Tutorial 12].

Os modelos de substituição disponíveis em POY incluem JC69/Neyman, F81, K2P/K80, F84, HKY85, TN93, e GTR, todos sob os quais pode-se ainda implementar categorias de taxa heterogêneas de substituição com a distribuição Gama (Γ) (Figura 13.1). POY não considera a

[8] MAL pode ser aplicado em sequências não alinhadas, porém requer que os caracteres sejam transformados inicialmente em caracteres estáticos, veja abaixo. O cálculo de MAL sob homologia dinâmica seria computacionalmente impraticável (Wheeler, pers. comm.).

proporção de sítios ivariáveis (I) como um parâmetro do modelo, uma vez que ele tecnicamente estaria contemplado na distribuição Gama (Γ). A seleção de modelos pode feita de forma automatizada, cabendo ao usuário a escolha de um critério dentre os três disponíveis em POY: AIC, AICc and BIC [para a definição de BIC, veja 5].

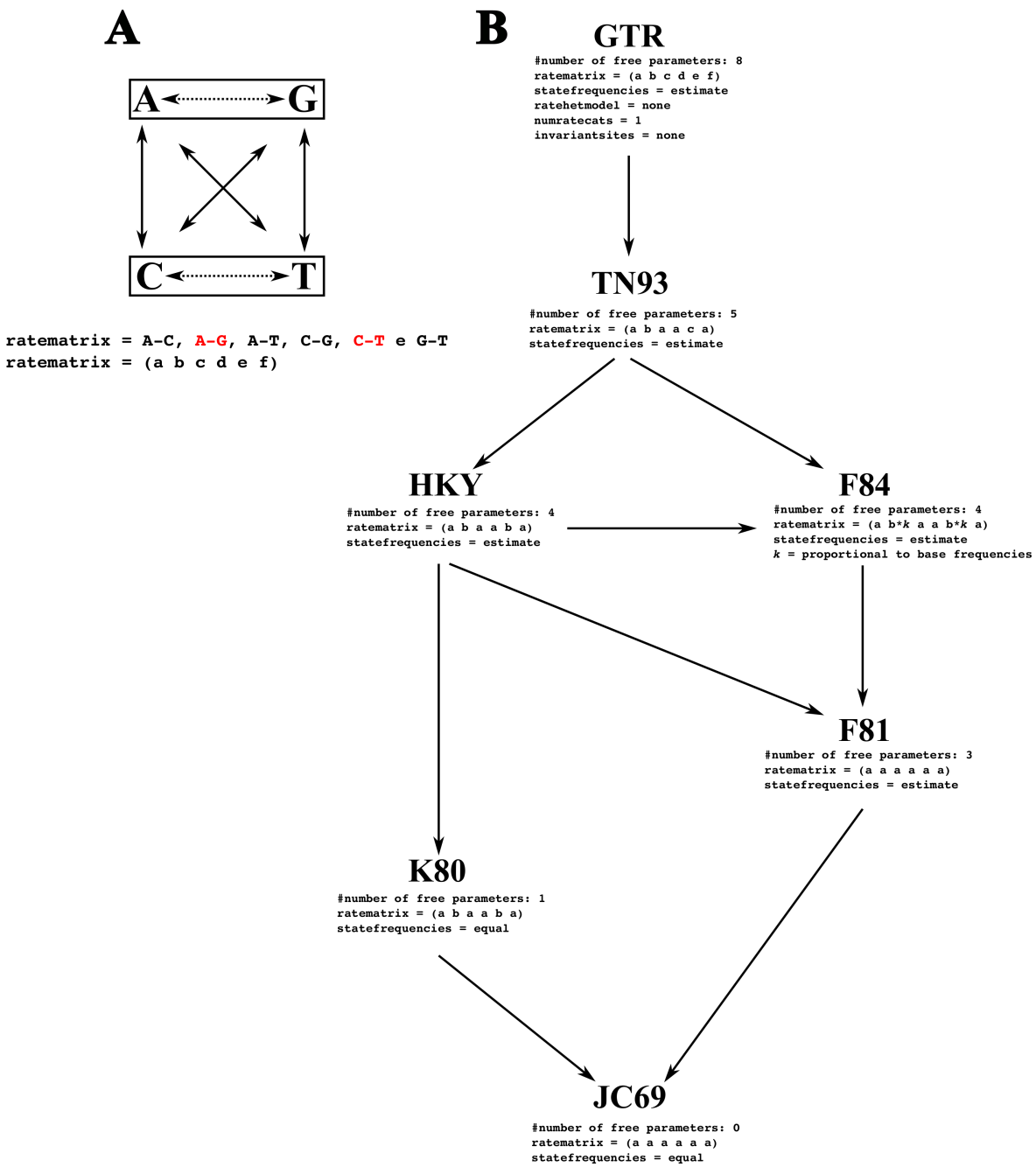


Figura 13.1: Modelos de substituição avaliados em POY. **A**. Seis substituições possíveis para nucleotídeos. Tranformações pontinhadas referem-se a transições, demais transformações são transversões. Estas seis transformações define os elementos das matrizes de substituições (`ratematrix` em Garli), cujas transformações em vermelho representam transições. A representação tradicional dessas transformações são expressas pelas letras *a–f*. **B**. Relação hierárquica dos modelos de substituição avaliados em POY. Para cada modelo é informado o número de parâmetros livres (*i.e.*, aqueles que são estimados durante a otimização da função de verossimilhança). Considere que para cada um desses modelos, POY considera taxas heterogêneas de substituição de acordo com 0 a 8 categorias da distribuição Gama (Γ).[9]

Arquivo texto 13.2: `model_test_part1.poy`.

```
1 (*MAL Analysis: Partitions and Model Selection*)
2 read(prealigned:("partition1.fas",tcm:(1,1)))
3 build(100)
4 swap()
5 select(best:1)
6 transform(likelihood:(aicc:"part1_AICc.txt",rates:gamma:(4), priors: estimate,mal))
7 swap()
8 report("part1_LK.tre",trees:(branches),"part1_LK_lkm.txt",lkmodel)
9 exit()
```

A forma de implementação da seleção de modelos em POY é relativamente simples. Considere, por exemplo o *script* 13.2. Na linha 1, o *script* lê os dados do arquivo `partition1.fas` e atribui peso 1 para todas as transformações. Posteriormente, linha 2 e 3, ele faz 100 RAS+TBR, cuja a melhor topologia é selecionada na linha 5. Observem que as buscas e seleção foram feitas aqui sob o critério de parcimônia. A linha 6 transforma o critério de otimalidade para verossimilhança, mais precisamente *Maxinum Average Likelihood* (veja abaixo), e inicia o teste de modelos adotando AIC_c como critério de seleção. O modelo de verossimilhança considera 4 categorias de taxas de substituições, distribuição Gamma (Γ), e os parâmetros livres dos modelos deverão ser estimados durante a análise. Posteriormente, o POY adota o melhor modelo sob esse critério de otimalidade e faz uma busca por `swap` cujos resultados, topologias e modelo, são impressos em seus respectivos arquivos de saida. Observe que esse *script* executa a seleção de modelo e faz buscas – embora muito heurística – em uma única análise.

A execução do `model_test_part1.poy` resulta em 3 arquivos. O arquivo `part1_AICc.txt` contém uma tabela comparativa dos modelos analisados cujo resultado é apresentado na Tabela 13.3. O arquivo `part1_LK_lkm.txt` detalha o modelo de substituição resultado da busca. Finalmente, o arquivo `part1_LK.tre` contém a topologia com seus respectivos comprimentos de ramos resultado da análise.

Arquivo texto 13.3: `model_test_part2a1mi.poy`.

```
1 (*ML Analysis: Partitions and Model Selection*)
2 read("partition2a1.fas")
3 transform(tcm:(1,1))
4 search(max_time:0:0:02)
5 select(best:1)
6 transform(static_approx)
7 transform(likelihood:(aicc:"part2a1mi_AICc.txt",rates:gamma:(4), priors:estimate, gap:missing, mal))
8 swap()
9 report("part2a1mi_LK.tre",trees:(branches),"part2a1mi_LK_lkm.txt",lkmodel)
10 exit()
```

No exemplo anterior, os dados iniciais encontravam-se pré-alinhados. O *script* 13.3 executa as mesmas análise do *script* anterior para dados não alinhados. Se compararmos os dois *scripts* notaremos que o segundo (*script* 13.3) lê e transforma os dados iniciais de forma diferente (linhas 2 e 3). Posteriormente, na linha 4, ele faz uma busca pelo comando `search` por dois minutos e seleciona uma árvore ótima, linha 5. Na linha 7, os caracteres que até então eram tratados como homologia dinânica são transformados em caracteres estáticos (`transform(static_approx)`) – essa transformação é necessária pois o POY só faz MAL

sob homologia estática. Em seguida, POY inicia a seleção de modelos como no *script* anterior, mas considerando gaps como dados lacunares (`gap:missing`) – da mesma forma que o GARLI. O POY permite três tratamentos diferentes para INDELs. Eles podem ser tratados como "*missing data*" e como tal não influenciam o cálculo de verossimilhança; podem ser tratados como caracteres (`gap:character`), neste caso inserções e deleções de A, C, G e T são tratadas individualmente como diferentes eventos e estimados independentemente; e finalmente, eles podem ser tratados de maneira conjugada (`gap:coupled`), e neste caso INDELs como um todo são tratados como um único parâmetro.

Tabela 13.3: Exemplo de arquivo de saída de POY para a seleção de modelos utilizando AIC_c mostrando os valores de verosimilhança para cada modelo avaliado. Colunas representam, modelo avaliado, *log* negativo da verossimilhança ($-L$), número de parâmetros estimados – inclui parâmetros livres do modelo + número de comprimentos de ramos (= $2t-3$, onde t é o número de terminais) + topologia – (K), número de caracteres (n), valores de AIC_c, diferença entre valores de AIC_c (Δ), pesos de Akaike (ω) e peso acumulado de Akaike (Cum(ω)). Neste exemplo, o modelo HKY+Γ possui o maior valor teórico de informação e seria o modelo selecionado.

Modelo	$-lnL$	K	n	AIC_c	$\Delta\ AIC_c$	ω	**Cum**(ω)
HKY+Γ	699.988278259	22	120	1454.40954621	0.	0.83608261736	0.83608261736
F84+Γ	701.73534071	22	120	1457.90367111	3.49412490153	0.145716795637	0.981799412997
TN93+Γ	702.530154326	23	120	1462.56030865	8.15076244343	0.0142014803771	0.996000893374
F81+Γ	707.445080336	21	120	1466.3187321	11.9091858917	0.00216871426416	0.998169607638
GTR+Γ	699.780583091	26	120	1466.65794038	12.2483941669	0.00183039236205	1.
F81	803.540920474	20	120	1655.56668943	201.157143224	1.74393601054e-44	1.
F84	812.286360077	21	120	1676.00129158	221.591745375	6.37108014702e-49	1.
K81+Γ	836.480712188	19	120	1718.56142438	264.151878168	3.65088083617e-58	1.
JC69+Γ	842.281419936	18	120	1727.3351171	272.925570891	4.54165865543e-60	1.
TN93	870.744628139	22	120	1795.92224597	341.512699759	5.80374993199e-75	1.
HKY	875.31326933	21	120	1802.05511009	347.64556388	2.70379754075e-76	1.
GTR	903.181048846	25	120	1870.19188493	415.782338718	4.32774342593e-91	1.
K81	944.292608696	18	120	1931.35749462	476.947948412	2.26109140474e-104	1.
JC69	955.127444482	17	120	1950.25488896	495.845342756	1.78156255514e-108	1.

Exercicio 13.4

Neste exercício você deverá fazer uma análise sob o critério de verossimilhança máxima em POY para o arquivo `partition2aln1.fas` que você gerou no Tutorial 12 implementando os três tratamentos de gaps possíveis no programa. Para executar esse exercício você deverá:

Tabela 13.4: Análise de modelo em POY assumindo diferentes modelos para INDELs. Valores de k podem ser obtidos na Figura 13.1.

Script	lnL	n	k	AIC_c	Model
`model_test_part2a1ch.poy`					
`model_test_part2a1co.poy`					
`model_test_part2a1mi.poy`					

1. Executar os *scripts* `model_test_part2a1ch.poy`, `model_test_part2a1co.poy` e `model_test_part2a1mi.poy`.
2. Sumarizar seus resultados na Tabela 13.4.
3. Resposder as pertuntas abaixo:

1. De acordo com o critério de AIC_c, o tratamento de gaps influencia os índices de verossimilhança? Justifique.

2. Os direrentes tratamentos de gaps geram topologias distintas? Justifique.

3. Qual desses resultados será comparável com a análise que você fez utilizando o GARLI? Como elas se comparam?

13.2.3 **MAL**: *Maximum Average Likelihood* EM POY

Este componente do tutorial apresenta a análise de caracteres estáticos pelo critério de verossimilhança média máxima (MAL) e explora algumas estratégias que podem ser usadas para otimizar tempo de execução em POY. Esta análise é similar em intensidade às buscas feitas pelo programa PhyML [6]. Análises totalmente executadas sob este critério, como por exemplo em PAUP* [7] no qual as buscas (RAS+SWAP) são calculadas por verossimilhança, podem ser computacionalmente muito intensas. Desta forma, o exemplo abaixo, *script* 13.4, ilustra elementos de busca sob outro critério, parcimônia, na etapa de construção das topologias e posterior refinamento sob verossimilhança.

Arquivo texto 13.4: mal_part1+2a1.poy.

```
1  read(prealigned:("partition1.fas",tcm:(1,1)))
2  read("partition2a1.fas")
3  transform(names:("partition2a1.fas"),(tcm:(1,1)))
4  set(root:"Taxon1")
5  search(max_time:00:00:01)
6  select()
7  transform(static_approx)
8  set(opt:coarse)
9  transform(likelihood:(tn93,rates:gamma:(4),priors:estimate,gap:coupled,mal))
10 swap(all:5,spr,optimize:(model:never,branch:never))
11 fuse(optimize:(model:never,branch:join_region))
12 select(best:1)
13 set(opt:exhaustive)
14 report("mal_p1+2a1.tre",trees:(branches),"mal_lkm_p1+2a1.txt",lkmodel)
15 exit()
```

No *script* 13.4, as três primeiras linhas tem a função de ler o arquivo pré-alinhado `partition1.fas` – pré-alinhado – e o arquivo `partition1.fas` – não-alinhado – atribuindo custos iguais as transformações. Posteriormente, o *script* implementa uma busca pelo comando `search` por um minuto e seleciona a(s) melhor(es) e única(s) topologia(s) desta análise de parcimônia (veja 13.2*iiii*). Após a seleção, os caracteres dinâmicos são transformados em caracteres estáticos (linha 7). Na linha 8 deste *script* é implementada a redução de precisão das casas decimais (*i.e., floating points*) durante as otimizações visando aumentar eficiência computacional (veja 13.2 *iiiiii*). Subsequentemente, linha 9, é implementado o critério de verossimilhança por MAL (`likelihood(...,mal)` sob o modelo de substituição `tn93`[10] considerando 4 categorias de taxas de substituição da distribuição Gama (`rates:gamma:(4)`) na qual a frequência das bases será estimada diretamente dos dados (`priors:estimate`) e os *gaps* serão considerados como quinto estado de caráter (`gap:coupled`) e tratados como um único parâmetro. Nas linhas 10 e 11 deste *script*, as estratégias de refinamento são implementadas sob a(s) topologia(s) coleta(s) durante a análise de parcimônia (veja 13.2*iiiv*). No entanto, observem que durante o *swap* tanto os parâmetros de modelo quando os comprimentos de ramos não serão otimizados (`optimize:(model:never,branch:never)`; veja 13.2*iiiiv*) e a estratégia de *swap* (`all:5,spr,`) é pouco agressiva (SPR – no qual rearranjo só ocorrerá dentro dos 5 ramos do ponto de quebra – ao invés de TBR que visitaria um maior número de topologias; veja 13.2*iiiv*). A etapa de *tree fusing* aumenta o rigor da otimização ao recalcular o comprimento de no máximo cinco ramos (`branch:join_region`, veja 13.2*iiiiv*). Após estas etapas de refinamento, uma única topologia ótima é selecionada, linha 12 (`select(best:1)`), e a precisão de *default* dos cálculos é re-estabelecida (`set(opt:exhaustive)`; veja 13.2*iiiiii*). Neste momento POY re-otimiza os parâmetro de verossimilhança. Finalmente, POY retorna a topologia encontrada com comprimentos de ramos (`"mal.tre",trees:(branches)`), o modelo de verossimilhança usado (`"mal_lkm.txt",lkmodel`) e termina a execução (`exit()`).

[10] esse modelo foi selecionado anteriormente em POY

Exercicio 13.5

Neste exercício você deverá fazer uma análise sob o critério de verossimilhança máxima em POY concatenando os dados do arquivos `partition1.fas` com os três alinhamentos feitos no Tutorial 12 para as sequências no arquivo `partition2.fas` utilizando o *script* 13.4 e as modificações do mesmo. Os resultados destas análises deverão ser compilados na Tabela 13.5 – na qual você encontra os modelos que deverão ser assumidos os quais foram selecionados anteriormente. Após a análise, responda as seguintes perguntas:

Tabela 13.5: Análise sob Maximum Average Likelihood considerando diferentes alinhamentos.

Data	**lnL**	n	k	AIC_c	**Model**
`partition1+partition2aln1.nex`					TN93
`partition1+partition2aln2.nex`					TN93
`partition1+partition2aln3.nex`					TN93

1. De acordo com o critério de AIC_c, Qual análise você selecionaria? Justifique.

Exercicio 13.6

Os resultados do Exercício 5 não podem ser comparados com os resultados que você obteve analisando os mesmo dados em GARLI. Neste exercício, você deverá modificar os arquivos do exercício anterior para que esses resultados possam ser comparados. Após a análise, compile seus dados na Tabela 13.6, responda a seguinte pergunta:

Tabela 13.6: Comparação das análises de Maximum Average Likelihood em POY e GARLI.

Data	**lnL**	n	k	AIC_c	**Model**
`partition1+partition2aln1.nex`					F84
`partition1+partition2aln2.nex`					F84
`partition1+partition2aln3.nex`					F84

1. Você poderia justificar a adoção de um desses programas em um estudo filogenético. Justifique.

_____

_____

_____

_____

_____

13.2.4 **MPL**: *Maximum Parsimonious Likelihood* EM POY

Como foi apresentado anteriormente, há diferentes sabores de verossimilhança. Este componente do tutorial apresenta a análise de caracteres dinâmicos pelo critério de verossimilhança máxima por parcimônia (MPL ou MPL/DO). Neste tipo de análise, alinhamento e busca de topologias são computados e examinados simultaneamente. Este processo é muito recente e pouco explorado. Tudo indica que dentro deste contexto, dado a complexidade do universo de soluções possíveis, análises heurísticas são insuficientes para obter resultados minimamente próximos da solução ótima. Desta forma, é possível que exceto em casos de banco de dados muito simples, análises dinâmicas sob MPL irão demandar recursos computacionais, incluindo paralelização de processos (*i.e., clusters*), além do que temos disponíveis para o dia a dia de nossas atividades.

Considere o *script* abaixo:

Arquivo texto 13.5: mpl_part1+2a1.poy.

```
1  read(prealigned:("partition1.fas",tcm:(1,1)))
2  read("partition2a1.fas")
3  transform(names:("partition2a1.fas"),(tcm:(1,1)))
4  set(root:"Taxon1")
5  search(max_time:00:00:01)
6  select()
7  set(opt:coarse)
8  transform(likelihood:(f81,rates:gamma:(4),priors:estimate,gap:coupled,mpl))
9  swap(all:5,spr,optimize:(model:never,branch:never))
10 fuse(optimize:(model:never,branch:join_region))
11 select(best:1)
12 set(opt:exhaustive)
13 report("mpl_p1+2a1.tre",trees:(branches),"mpl_lkm_p1+2a1.txt",lkmodel,"mpl_p1+2a1_ia.fas",fasta)
14 exit()
```

Este *script* apresenta a mesma estratégia do *script* 13.4 com algumas diferenças. Primeiro que não é necessária a transformação dos caracteres em homologia estática antes de submeter ao critério de verossimilhança. A transformação no entanto, define que o cálculo deve ser baseado em MPL. O comando `report` inclui como arquivo de saida o alinhamento implícito da análise.

Exercicio 13.7

Neste exercício você deverá executar o *script* `mpl_part1+2a1.poy`. Após a execução, examine os arquivos de saida e responda as seguintes perguntas:

i. As topologias encontradas por MPL e MAL são as mesmas para esses dados? Comente.

ii. hà como comparar estas duas formas de Verossimilhança?

13.3 Considerações finais sobre análises de verossimilhança e homologia dinâmica

Há duas coisas a considerar ao optar pela análise de dados reais sob critérios de verossimilhança. A primeira delas é que o uso de homologia dinâmica em análises de verossimilhança é relativamente recente e pouco sabemos sobre o comportamento dessas análises. A segunda delas é a extrema complexidade computacional destes algoritmos tornando impeditivo sua aplicação em computadores convencionais em dados reais. Mesmo nos casos apresentados aqui, há muito espaço para melhorar a agressividade das análises. A leitura do manual do programa aponta alguns caminhos. Desta forma, este tutorial explorou de forma muito breve as estratégias de busca sob o critério de verossimilhança e algumas das estratégias heurísticas que podem ser implementadas com o intuito explorar melhor o espaço de soluções possíveis de análises sob estes critérios. Aos interessados em aplicar o critério de verossimilhança em seus dados, eu recomendaria a leitura atenda da documentação de POY. Nele você encontrará alguns conceitos que foram omitidos neste tutorial – visando simplicidade – e outras estratégias e ferramentas associadas a este tipo de análise.

Outro componente que não foi abordado neste tutorial é a inclusão de dados morfológicos em análises filogenéticas sob critérios de verossimilhança. A documentação de POY possuí um tutorial para este fim e o aluno interessado em conhecer um pouco mais sobre este tema deve consultar esse material.

13.4 Referências

1. Zwickl, D. J. Genetic algorithm approaches for the phylogenetic analysis of large biological sequence datasets under the maximum likelihood criterion. Tese de doutorado (The University of Texas, Austin, 2006).
2. Jukes, T. H. & Cantor, C. R. em *Mammalian protein metabolism.* ed. Munro, H. N. New York: Academic Press, 1969.
3. Varon, A.; Lucaroni, N.; Hong, L. & Wheeler, W. C. 2011–2014. POY version 4: phylogenetic analysis using dynamic homologies, version 5.0. New York, NY: American Museum of Natural History, 2011–2014.

4. Stamatakis, A. 2014. RAxML Version 8: A tool for phylogenetic analysis and post-analysis of large phylogenies. *Bioinformatics* **doi: 10.1093/bioinformatics/btu033**:

5. Darriba, D. & Posada, D. jModelTest 2.0 v0.1.1. http://code.google.com/p/jmodeltest2/. 2015.

6. Guindon, S *et al.* 2010. New Algorithms and Methods to Estimate Maximum-Likelihood Phylogenies: Assessing the Performance of PhyML 3.0. *Systematic Biology* **56**(3): 307–321.

7. Swofford, D. 2003–2016. PAUP*. Phylogenetic Analysis Using Parsimony (*and Other Methods, Version 4.0a131). Sunderland, Massachusetts: Sinauer Associates, 2003–2016.

Tutorial 14

Suporte

BIZ0433 - INFERÊNCIA FILOGENÉTICA: FILOSOFIA, MÉTODO E APLICAÇÕES.

POR DENIS JACOB MACHADO & FERNANDO P. L. MARQUES

Conteúdo

Objetivo

O objetivo deste tutorial é apresentar conceitos teóricos e práticos do cálculo de índices de suporte em inferência filogenética. A parte conceitual é brevemente apresentada aqui e o leitor deve consultar Grant & Kluge [1], e referências ali citadas, para uma discussão mais aprofundada sobre os aspectos epistemológicos associados à cada métrica. O tutorial inicia-se com uma apresentação geral sobre o conceito de suporte. A seguir são apresentados 3 métricas, duas das quais são comumente utilizados em análises filogenéticas. O primeiro deles é o Bootstrap, o segundo os valores de Goodman–Bremer e, finalmente, a Verossimilhança diferencial. 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/
```

"It is sometimes observed that the measure of support does not have any 'meaning', by which is usually meant any 'probability interpretation'. It is indeed true that a statement of support, though derived from probabilities, does not make any assertion about the probability of a hypothesis being correct. And for a good reason: the method of support has been developed by people who explicitly deny that any such statement is generally meaningful in the context of a statistical hypothesis." [2:33]

14.1 Introdução

Como sugere Grant & Kluge [1], a avaliação de suporte de clados é um componente importante de análises filogenéticas, mesmo que pouca atenção tenha sido dada às bases conceituais dos métodos geralmente empregados para medi-lo. Adicionalmente, o próprio conceito de "suporte" difere entre cientistas, em geral, e filogeneticistas em particular.

Em estudos filogenéticos, há uma série de métricas utilizadas para inferir índices de suporte de clados. Dentro destes índices, há aqueles considerados métricas de suporte indireto, dentro dos quais encontramos Boostrap e Jackknife – entre outros, e métricas de suporte direto, tais como Goodman-Bremer e Verossimilhança diferencial – entre outros. Este tutorial irá apenas abordar a implementação de algumas destas métricas. Para uma revisão mais adequada sobre o assunto é recomendável a leitura de Siddall [3], Egan [4], Grant & Kluge [1] e Wheeler [5]; e referências citadas por esses autores.

Existem definições explícitas sobre suporte na literatura. No entanto, filogeneticistas parecem ignorá-las. Em seu livro clássico sobre Verossimilhança, Edwards [2] considera que índices de suporte devem aferir o mérito relativo de hipóteses rivais diante dos dados observados. Para este autor, a noção de suporte pode ser concebida por uma interpretação operacional perfeitamente simples da diferença entre as verossimilhanças entre duas hipóteses com relação aos dados. Neste contexto, essa diferença expressa, à longo prazo, a frequência na qual as hipóteses rivais explicam os dados observados.

Grant & Kluge [1] oferecem uma proposta de definição objetiva para suporte em inferência filogenética. Para esses autores, métricas de suporte só são adequadas quando diretamente relacionadas com o critério de otimalidade. Para estes autores, conhecimento adquirido (b), probabilidade (p), hipótese (h) e evidência (e) são definidos como:

1. O conhecimento adquirido (b) é toda soma de conhecimento que não esteja sujeito ao teste de hipóteses ou falseamento.
2. A probabilidade (p) diz respeito à "frequência relativa" de eventos.
3. A hipótese (h) é um cenário causal ou causativo que está sujeito a teste e falseamento.
4. Evidência (e) são os dados obtidos para testar a hipótese.

Segundo Popper [6], o suporte (S) é a medida da diferença entre (i) probabilidade da evidência dada a hipótese e o conhecimento adquirido e (ii) a probabilidade da evidência dado o conhecimento adquirido:

$$S = p(e|h,b) - p(e|b) \tag{14.1}$$

Desta forma, suporte pode ser visto como uma afirmação empírica sobre a força de hipóteses ($h_1, h_2, h_3...h_n$) em relação ao corpo de evidências e dado um certo conhecimento adquirido b. Sendo que otimalidade (O) é também uma afirmação empírica sobre a força de hipóteses concorrentes em relação ao corpo de evidências dado um certo conhecimento adquirido. Seguindo essa lógica, Grant & Kluge [1] concluem que ambos, S e O, devem ser diretamente proporcionais:

$$S(h|e,b) \propto O(h|e,b) \tag{14.2}$$

Observe que otimalidade e suporte não são sinônimos (veja Tabela 14.1). Desta forma, S é diretamente proporcional, mas não igual, a O. Deste modo, para que uma medida de suporte seja objetiva, ela deve quantificar suporte como uma função do poder explanatório. Segundo esta definição de suporte, o índice Goodman-Bremer é uma medida direta do suporte, mas não os valores de Bootstrap e Jackkinife.

Tabela 14.1: Suporte vs. Otimalidade

Suporte	**Otimalidade**
Refere-se à força *relativa* entre diferentes hipóteses	Refere-se à força *absoluta* de uma hipótese
É heurístico – facilita a descoberta	É científico
Geralmente está relacionado à clados	Diz respeito a topologias

14.2 Bootstrap

14.2.1 Considerações gerais

A proporção de Bootstrap é a métrica mais utilizada em estudos filogenéticos, apesar de ter sido criticado severamente quanto às violações de premissas do método [veja 3, 5; e referências citadas] e inconsistência epistemológica [5]. Este método estatístico foi concebido para melhorar estimativas de parâmetros de distribuições desconhecidas. O método se vale de obter n pseudo-réplicas da amostra sobre as quais é calculado o parâmetro de interesse.

Considere o seguinte exemplo executado em R:

```
> s <- c(320.44, 303.98, 264.10, 286.90, 274.59, 332.72, 346.37, 240.83
> mean(s);
[1] 296.2412
> r1 <- s sample(s,8,replace=T)
> r1
[1] 303.98 274.59 286.90 274.59 240.83 274.59 303.98 346.37
```

```
> r2 <- sample(s,8,replace=T)
> r2
[1] 320.44 286.90 320.44 264.10 286.90 286.90 240.83 303.98
> r2 <- sample(s,8,replace=T)
> r3 <- sample(s,8,replace=T)
> r4 <- sample(s,8,replace=T)
> r5 <- sample(s,8,replace=T)
> r6 <- sample(s,8,replace=T)
> r7 <- sample(s,8,replace=T)
> r8 <- sample(s,8,replace=T)
> r9 <- sample(s,8,replace=T)
> r10 <- sample(s,8,replace=T)
> R <- c(r1,r2,r3,r4,r5,r6,r7,r8,r9,r10)
> mean(R)
[1] 303.2571
```

Neste exemplo, a variável s recebe 8 medidas morfométricas para aqual a média estimada é 296.2412 (linha 2 e 3). O método de bootstrap, se vale do uso de pseudo-réplicas ($r1, r2, r3...r10$), cada qual com o mesmo tamanho amostral (8) e considerando reposição. Observe, linha 6, que na pseudo-réplica 1 ($r1$) a quinta medida de minha amostra, 274.59, foi considerado 3 vezes. Da mesma forma, na segunda pseudo-réplica, linha 9, foi a medida 286.90 que foi incluída na mesma proporção. Ao final de 10 pseudo-réplicas, ao computar os valores obtidos, a média estimada via bootstrap é 303.25. Teoricamente, esse valor é uma estimativa mais aproximada da média real da população do qual eu obtive minha amostra. É importante considerar, que o método assume (*i.*) que minha amostra foi obtida aleatoriamente, (*ii.*) que eles são independentes e (*iii.*) que eles são identicamente distribuídos – ou seja, foram obtidos da mesma distribuição. É interessante observar que nenhuma dessas premissas pode ser assumida para dados filogenéticos.

14.2.2 Bootstrap sob o critério de Parcimônia

Em sistemática filogenética, o bootstrap é aplicado da mesma maneira. As pseudo-réplicas são matrizes com o mesmo número de caracteres que a matriz original com reposição, ou seja, em uma pseudo-réplica, um ou mais caracteres podem estar representados mais que uma vez – consequentemente um ou mais caracteres podem estar presente. Para cada pseudo-réplica, faz-se uma busca e a(s) topologia(s) encontrada(s) é(são) retida(s) até que se complete o ciclo de pseudo-réplicas. Ao final, computa-se o consenso de maioria e a frequência de cada clado representa a proporção do clado em bootstrap. Desta forma, o método obedece o seguinte cronograma de execuções:

1. Determinar uma árvore ótima;
2. Re-amostrar os dados (colunas de caracteres) com reposição, gerando novas matrizes com o mesmo número de colunas que a matriz original;

3. Determinar uma árvore ótima para cada nova matriz criada usando o mesmo procedimento que o empregado em (1);
4. Repetir (2) e (3) k vezes, salvando todas as árvores assim encontradas;
5. Construir uma de consenso de maioria com todos os nós que tenham frequência ≥ 0.50

É interessante notar que para um conjunto de n, caracteres sendo que nenhum deles é homoplástico, o valor de bootstrap (V_{Bt}) para o vértice V cujo suporte reside em r caracteres é dado pela seguinte função:

$$V_{Bt} = 1 - (1 - \frac{r}{n})^n \tag{14.3}$$

Exercicio 14.1

No diretório deste tutorial há seis arquivos `bs_example_1*.tnt` (*=a–f) no formato xread. Adicionalmente, há um arquivo chamado `vbt.r` que pode ser editado para calcular o valor de V_{Bt} da equação acima (`$ Rscript vbt.r`). Verifique o conteúdo destes arquivo e note as diferenças que existem entre as 6 matrizes de dados. Observe também que o arquivo contém as instruções para o cálculo dos valores de boostrap em TNT. Você deverá executar esses arquivos em TNT (*e.g.*, `$ tnt proc bs_example_1a.tnt`) e responder às seguintes perguntas:

1. Quais são as principais diferenças que você observa nas matrizes apresentadas?

2. Os índices de Bootstrap que você obteve correspondem às expectativas teóricas da Equação 14.3?

3. Você acha que os números de caracteres necessários para prover suporte aos vértices é dependente no números de terminais?

4. Considere as duas matrizes que diferem quanto aos números de terminais mas possuem o menor números de caracteres sustentando cada clado. Você consideraria que seus dados não possuem suporte para seus resultados?

Quando proposto inicialmente por Felsenstein [7], os valores de deveriam fornecer intervalos de confiança para os vértices. De acordo com Felsenstein [7], vértices eram considerados como estatisticamente significantes quando os valores de Bootstrap eram ≥ 0.95. Esta ideia foi subsequentemente abandonada, no entanto há pesquisadores que ainda consideram esse valor como uma medida de suporte para clados. É interessante notar que a Equação 14.3 permite a seguinte derivação:

$$r = n - n * (e^{\frac{ln(1-V_{Bt})}{n}}) \tag{14.4}$$

Considere que o número mínimo de caracteres para resolver um diagrama binário é $t - 2$, onde t é o número de terminais. Desta forma, teríamos:

$$r = (t-2) - (t-2) * (e^{\frac{ln(1-V_{Bt})}{t-2}}) \tag{14.5}$$

Essa função estima o valor de r, ou seja, número de caracteres que sustentam um vértice, para uma matriz com t terminais e determinado valor de V_{Bt} – lembrando que assume-se que não há homoplasias nos dados.

Exercicio 14.2

No diretório deste tutorial há um *script* chamado `minimum_character.r` que produz um gráfico para os valores de r e t de acordo com o valor de V_{Bt}. Sua execução é feita com a seguinte linha de comando: `$ Rscript minimum_character.r`. Neste exercício você deverá fazer três execuções desse *script*. Após cada uma delas você deverá renomear o arquivo de saída `Rplots.pdf`, caso contrário a próxima execução apagará o resultado da anterior. Inicialmente, você deverá executar o *script*. Nas demais execuções, você deverá alterar is valores da variável `VBt`, linha 3, para .99 e .999999. Após as execuções, avalie os gráficos resultantes e responda:

i. Os resultados de sua análise modificam sua impressão relacionada ao exercício anterior sobre a relação entre a dependência entre número de terminais e o número de caracteres necessário para dar suporte ao nó de acordo com o os valores de bootstrap? Justifique.

ii. De acordo com suas observações e os argumentos levantados por Grant & Kluge [1], você coonsidera essa métrica uma boa medida de suporte?

14.2.3 Bootstrap sob o critério de Verossimilhança Máxima

O cálculo de Bootstrap sob este critério de otimalidade obedece a mesma lógica de como ele é calculado sob o critério de parcimônia. Inúmeras buscas sob este critério são executadas para as pseudo-réplicas cujas topologias obtidas são utilizadas ao final para calcular um consenso de maioria. A frequência de cada clado é denominada proporção de Bootstrap.

Como as análises de Verossimilhança foram efetuadas em GARLI [8], iremos utilizar o mesmo programa para calcular as proporções de Bootstrap. Considere o conteúdo do arquivo `garli_bootstrap.conf` (disponível no diretório deste tutorial). Este arquivo é uma modificação do arquivo de configuração do GARLI `garli_single.conf` utilizado no Tutorial 13. O comando `datafname` foi modificado para receber o arquivo `partition1+2aln3.nex`; o comando `ofprefix` foi modificado para indicar a análise que será feita; o comando `searchreps` foi configurado para uma única busca; e finalmente, o comando `bootstrapreps` foi configurado para 100 réplicas. Adicionalmente, o modelo especificado ($TrN + I + \Gamma$) foi selecionado anteriormente pelo jModelTest 2 (Tutorial 12).

A execução de GARLI utilizando o arquivo `garli_bootstrap.conf` gera três arquivos. Os arquivos `garli_bs.log00.log` e `garli_bs.screen.log` são os registros da execução. O arquivo `garli_bs.boot.tre` contém as topologias recuperadas para cada uma das pseudo-réplicas (100) e é este arquivo que é utilizado para calcular as proporções de Bootstrap.

A forma mais fácil de calcular estas proporções é utlizando SumTrees (*Phylogenetic Tree Summarization and Annotation*) que é uma ferramenta de DendroPy. No exemplo acima, após computada as topologias de Bootstrap, bastaria executar o seguinte comando:

```
sumtrees.py -d 0 -p -o garli_run.best_bootstrap.tre -t garli_run.best.tre garli_bs.boot.tre
```

Há várias formas de computar os valores de Bootstrap usando `sumtrees.py` e é recomendável verificar as opções disponíveis na documentação do programa[1]. Na linha de comando

[1] https://pythonhosted.org/DendroPy/programs/sumtrees.html

acima, a opção `-d 0` define o número de decimais, `-p` define que os resultados serão expressos em porcentagem, `-o` é a opção que define o arquivo de saída (*output*) – neste caso `garli_run.best_bootstrap.tre`, `-t` é a opção que define a topologia alvo, ou seja, aquela cujos clados você deseja verificar as frequência nas topologias recuperadas durante a análise de bootstrap – neste caso `garli_run.best.tre`[2], e finalmente, o arquivo no qual as topologias de Boostrap estão, `garli_bs.boot.tre`.

A etapa final deste processo é observar os valores de Bootstrap no programa FigTree[3]. Este programa já foi utilizado no Tutorial 2 e é utilizado para visualização e edição de topologias. Para verificar os valores de Bootstrap, basta seguir as seguintes etapas:

1. Abra o arquivo `garli_run.best_bootstrap.tre` em FigTree.
2. O programa exibirá uma janela solicitanto que você defina à que se referem os valores das topologias. Digite "bootstrap" e pressione OK.
3. Enraize a topologia no táxon "`Taxon1`".
4. No canto inferior esquerdo na janela de Figtree, selecione "Branch Lables".
5. Abra as opções de "Branch Lables" – pressionando o triângulo à esquerda desta opção – e em "Display", "selecione bootstrap".

Exercicio 14.3

Neste exercício, você deverá visualizar o arquivo `garli_bs.boot.tre` em FigTree e anotar os valores de Bootstrap na Figura 14.1.

14.3 Suporte de Goodman-Bremer

Ao contrário dos valores de Bootstrap, o suporte de Goodman-Bremer (GB) [9, 10] é uma medida empírica de suporte dado pela evidência a um determinado clado. Portanto, é considerado um valor de suporte direto. O suporte de Goodman-Bremer é a diferença de comprimento (*i.e.*, custo, número de passos) entre o cladograma mais parcimonioso (L) e o cladograma mais parcimonioso que não apresenta um determinado clado de interesse (L'):

$$GB = L' - L \tag{14.6}$$

Em outras palavras, o valor de Goodman-Bremer representa o número adicional de passos em uma topologia sub-ótima necessário para colapsar determinado nó. Uma outra maneira de pensar sobre essa métrica é que o seu valor indica a quantidade relativa de contra-evidências necessária para que você não tenha suporte nenhum sobre determinado grupo. Vale considerar que o GB é

[2]Esta topologia foi recuperada durante o Tutorial 12

[3] O Tutorial 2 possui um breve vídeo de como o programa funciona.

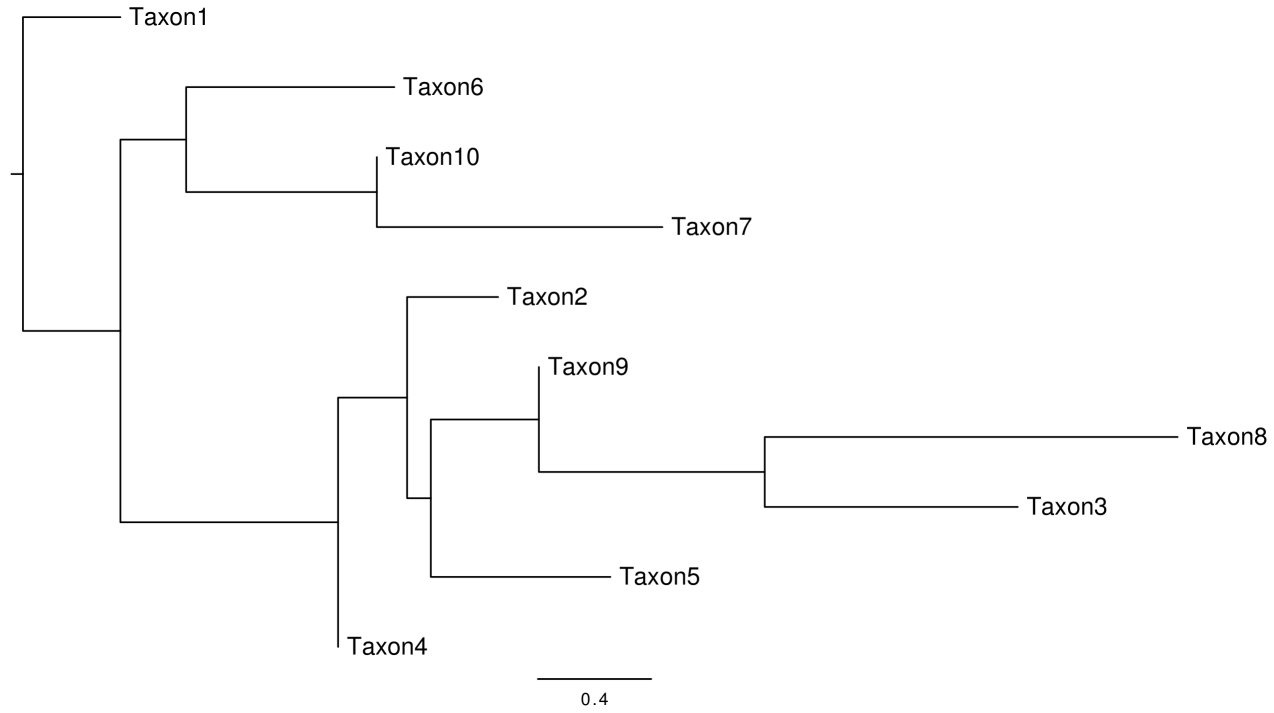


Figura 14.1: Topologia selecionada sob o critério de Verossimilhança máxima dos dados em `partition1+2aln3.nex` com lnL -1369.5791.

dependente do enraizamento da topologia e que grupos que estariam ausentes no consenso estrito, e portanto não são suportados pelas evidências, apresentam $GB = 0$.

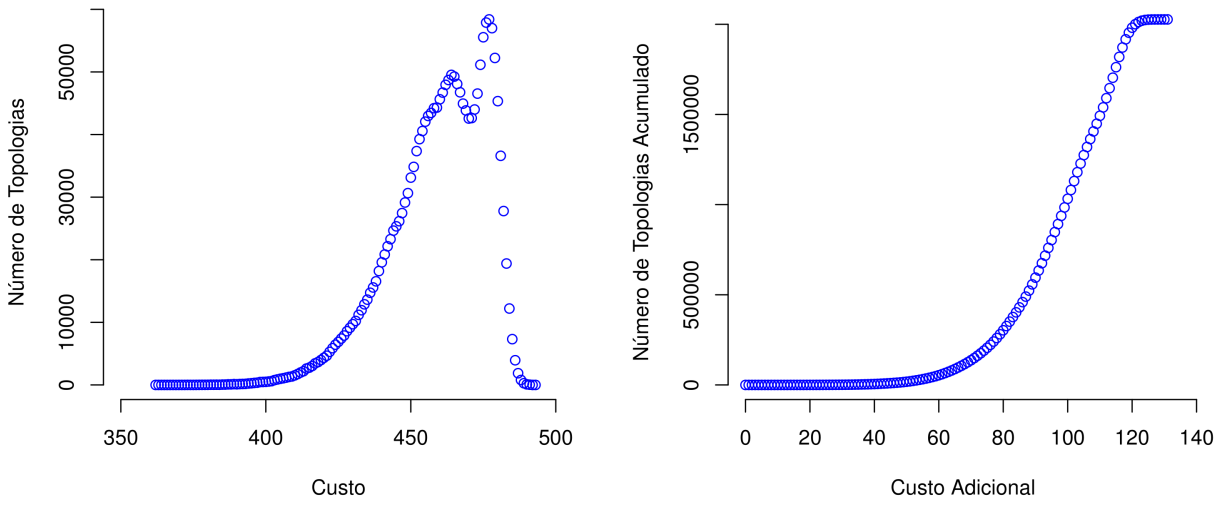


Figura 14.2: Distribuição de topologias de acordo com seu custo (esquerda) e acúmulo de topologias de acordo com número de passos adicionais com relação à topologia mais curta para os dados do arquivo `total_evidence.tnt` (direita).

Há algumas propriedades sobre esse índice que você deve considerar. Está claro que para computá-lo é necessário avaliar topologias sub-ótimas e isso pode ser um problema se você está diante de um espaço topológico complexo. Considere por exemplo o arquivo de dados

`total_evidence.tnt`. Neste arquivo estão concatenados os dados de `partition1.fas`, `partition2.fas` e `partition3.tnt`. A enumeração destes dados, que contém 10 terminais – portanto 2.027.025 topologias possíveis, resulta em custos que variam de 362 a 493 (Figura 14.2). Suponha que para determinado nó, o índice de GB seja 120, você teria que ter avaliado praticamente todo o espaço de topologias. Felizmente, para esses dados, o clado com maior índice de GB é 14. Desta forma, para esses dados o cômputo desses índices parece ser trivial – o que certamente não é o caso de muitos dados reais.

Há várias estratégias adotadas para calcular o índice de BG. Em sua forma mais primitiva, você poderia compilar todas as topologias com n passos de distância da topologia mais curta que você obteve para seus dados e posteriormente, calcular passo a passo esse índice filtrando as topologias $c + n$, no qual c é o custo mínimo e n é o número de passos adicionais, calculando o consenso, e verificando quais nós são colapsados. Fazer isso manualmente é laborioso e demanda tempo computacional e analítico.

A melhor forma de calcular o índice de GB é utilizando um macro disponível em TNT chamado `BREMER.RUN`. Este macro utiliza buscas com os algoritmos de novas tecnologias para compilar de tologias sub-ótimas, que posteriormente são utilizadas para obter os índices. Para executar esse macro é necessário que você possua uma topologia na memória. Caso seus dados resultem um múltiplas topologias, o ideal seria você calcular o consenso estrito desta topologia, mantê-lo na memória e descartar as topologias fundamentais (veja Seção 6.2 do Tutorial 6) e, finalmente, executar o macro do TNT.

O POY também permite o cálculo deste índice utlizando algumas de suas estratégias internas. Porém isso é feito assumindo homologia estática dos dados, uma vez que o uso de homologia dinâmica para esse propósito se mostrou computacionalmente inviável – além de desnecessário. Sendo assim, considero mais efetivo submeter o alinhamento implícito das análises em POY para o cálculo de BG. Considere o exemplo abaixo:

```
read("partition1.fas","partition2.fas","partition3.tnt")
set(root:"Taxon1")
search(max_time:0:0:5)
select()
set(iterative:exact)
swap(tbr)
fuse()
select(best:1)
transform (all, (static_approx))
report ("total_evidence.tnt", phastwinclad)
exit()
```

Neste *script* (denominado `poy_run.poy` e disponível no diretório deste tutorial), os dados são lidos e analisados utilizando os algoritmos de busca do comando `search` por 5 minutos, a melhor topologia e selecionada e rediagnosticada via *Interative Pass* com refinamento, uma única topologia mais curta é selecionada, todos os dados são transformados em homologia estática e impresso no arquivo `total_evidence.tnt` em formato TNT. Essa lógica analítica é uma boa estratégia para dados sujeitos a homologia dinâmica para os quais você deseja calcular índices que dependam de homologia estática – e consequentemente valores de Bootstrap (caso você ainda considere esse último útil para alguma coisa). Para obter os valores de BG para esse arquivo, então, basta: (*i*.) analisá-lo em TNT e obter uma topologia e (*ii*.) executar o macro com o comando `run BREMER.RUN`.

Exercicio 14.4

Neste exercício você deverá obter os índices de bootstrap e GB para o arquivo `total_evidence.tnt`. Use o espaço abaixo para desenhar a topologia e anotar os índices obtidos e, posteriormente, responda as seguintes perguntas:

i. Qual é o maior índice de GB que um determinado vértice (nó) pode possuir?

ii. Existe alguma correlação entre os índices de bootstrap e GB?

14.4 Verossimilhança diferencial (*Likelihood difference*)

A Verossimilhança diferencial é equivalente ao índice de GB dentro do contexto de análises sob o critério de Verossimilhança. Embora pouco empregado [veja referências em 1], esse índice possui todas as propriedades desejáveis de um índice de suporte [1, 2]. O índice é dado pela seguinte formulação:

$$P(e|h,b) - P(e|h',b) \tag{14.7}$$

no qual h é a hipótese selecionada e h' é a hipótese alternativa. Como GB, este índice está diretamente relacionado ao critério de otimalidade, e portando, deve ser considerado um índice de suporte direto. Dentro do contexto de análises filogenéticas, a diferença de Verossimilhança é aplicada em relação à topologia ótima e aquela considerada ótima na ausência do clado em questão Atualmente não há programas que possibilitam o cômputo imediato deste índice. Sua obtenção, portanto, dependerá de uma série de análises sequenciais que iremos demonstrar a seguir.

Suponha que seus dados estejam em `partition1+2aln3.nex` e que após selecionar o melhor modelo para sua análise, você configurou o arquivo `garli_single.conf`, ambos disponíveis no diretório deste tutorial. O resultado desta análise (arquivos `garli_run.*.*`) resultou em uma topologia com lnL -1369.5791. Figura 14.1 do Exercício 14.3 é resultado desta análise.

Para calcular a diferença de Verossimilhança para o clado (Taxon8+Taxon3) é necessário obter a topologia ótima na qual esse clado não existe. Isso é feito utilizando o que chamamos de buscas sob restrição reversa (*reverse/negative constraint*). A primeira etapa seria definir a topologia que contém esse clado. Para esse caso em particular, seria:

```
(Taxon1,Taxon2,Taxon4,Taxon5,Taxon6,Taxon7,Taxon9,Taxon10,(Taxon8,Taxon3))
```

Esta topologia deve estar contida em um arquivo texto com um sinal $-$ (negativo), que indica ao GARLI que ela deve ser utilizada como restrição reversa. Chamaremos esse arquivo de `contraint_8+3.tre`. O próximo passo é editar o arquivo de configuração de GARLI. Para isso, uma cópia do arquivo `garli_single.conf` foi feita e denominada `garli_constrain.conf` e os comandos `ofprefix` e `constraintfile` foram modificados para o cálculo (veja conteúdo desses arquivos).

O resultado desta análise gerou uma topologia com lnL -1369.8372, a qual é considerada a topologia ótima sob este critério na qual o clado (Taxon8+Taxon3) não existe. Para calcular a Verossimilhança diferencial para este clado, basta subtrair o lnL da hipótese sob restrição reversa daquela obtida na primeira análise onde este clado foi recuperado. Assim

$$LR_{T8+T3} = lnL_{comT8+T3} - lnL_{semT8+T3} \tag{14.8}$$

Portanto,

$$LR_{T8+T3} = 0.2581. \tag{14.9}$$

Exercicio 14.5

Neste exercício você deverá obter os índices das Razões de Verossimilhança para os cinco clados definidos abaixo, provenientes da Figura 14.1, utilizando as etapas descritas acima. Anote os valores obtidos na Figura 14.1 e responda:

Clados de interesse:

```
(Taxon1,Taxon2,Taxon4,Taxon5,Taxon6,Taxon7,Taxon10,(Taxon9,Taxon8,Taxon3))
(Taxon1,Taxon2,Taxon4,Taxon6,Taxon7,Taxon10,(Taxon5,Taxon9,Taxon8,Taxon3))
(Taxon1,Taxon4,Taxon6,Taxon7,Taxon10,(Taxon2,Taxon5,Taxon9,Taxon8,Taxon3))
(Taxon1,Taxon6,Taxon7,Taxon10,(Taxon4,Taxon2,Taxon5,Taxon9,Taxon8,Taxon3))
```

i. Existe alguma correlação observável entre os índices de Bootstrap obtidos no Exercício 14.3 e Verossimilhanca diferencial?

14.5 Referências

1. Grant, T & Kluge, A. 2008. Clade support measures and their adequacy. *Cladistics* **24**(6): 1051–1064.
2. Edwards, A. W. F. 1992. Likelihood. Baltimore, Maryland: The John Hopkins University Press, 1992. 275 pp.
3. Siddall, M. E. em *Methods and Tools in Biosciences and Medicine* eds. DeSalle, R.; Wheeler, W. C. & Giribet, G., 80–101. Basel: Birkhäuser Verlag, 2001.
4. Egan, M. G. 2006. Support versus Corroboration. *Journal od Biomedical Informatics* **39**: 72–85.
5. Wheeler, W. C. 2012. Systematics: a course of lectures. Malaysia: Wiley-Backwell, 2012. 426.
6. Popper, K. 1959. The Logic of Scientific Discovery. London: Routledge, 1959, 513. ISBN: 041507892X.
7. Felsenstein, J. 1985. Confidence Limits on Phylogenies: An Approach Using the Bootstrap Joseph. *Evolution* **39**(4): 783–791.
8. Zwickl, D. J. Genetic algorithm approaches for the phylogenetic analysis of large biological sequence datasets under the maximum likelihood criterion. Tese de doutorado (The University of Texas, Austin, 2006).
9. Goodman, M; Olson, C.; Beeher, J. & Czelusniak, J. 1982. New perspectives in the molecular biological analysis of mammalian phylogeny. *Acta Zoologica Fennica* **169**: 19–35.
10. Bremer, K. 1988. The Limits of Amino Acid Sequence Data in Angiosperm Phylogenetic Reconstruction. *Evolution* **42**(4): 795–803.