

Tutorial 4

Introdução ao TNT

BIZ0433 - INFERÊNCIA FILOGENÉTICA: FILOSOFIA, MÉTODO E APLICAÇÕES.

Conteúdo

Objetivo	60
4.1 Considerações gerais	61
4.2 Estrutura dos arquivos de entrada	61
4.3 Topologias em TNT	65
4.4 Obtendo ajuda no TNT	66
4.5 Leitura de topologias em TNT	67
4.6 Como salvar topologias em TNT	69
4.7 Busca de topologias em TNT	70
4.8 Referências	76

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 em <http://lhe.ib.usp.br/cladistica>. Você pode baixá-los diretamente com o seguinte comando:

```
wget http://lhe.ib.usp.br/downloads/tutorial_04.zip
```

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 morfológicos'
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 AAAAAACGT
taxon_5 AAAAAAAGT
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.

Exercício 4.1

Em um terminal, você deverá 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 um 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 possui nenhum erro de sintaxe. A primeira coisa que devemos fazer é executar o TNT. No *prompt* do seu terminal execute:

```
$ tnt
```

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

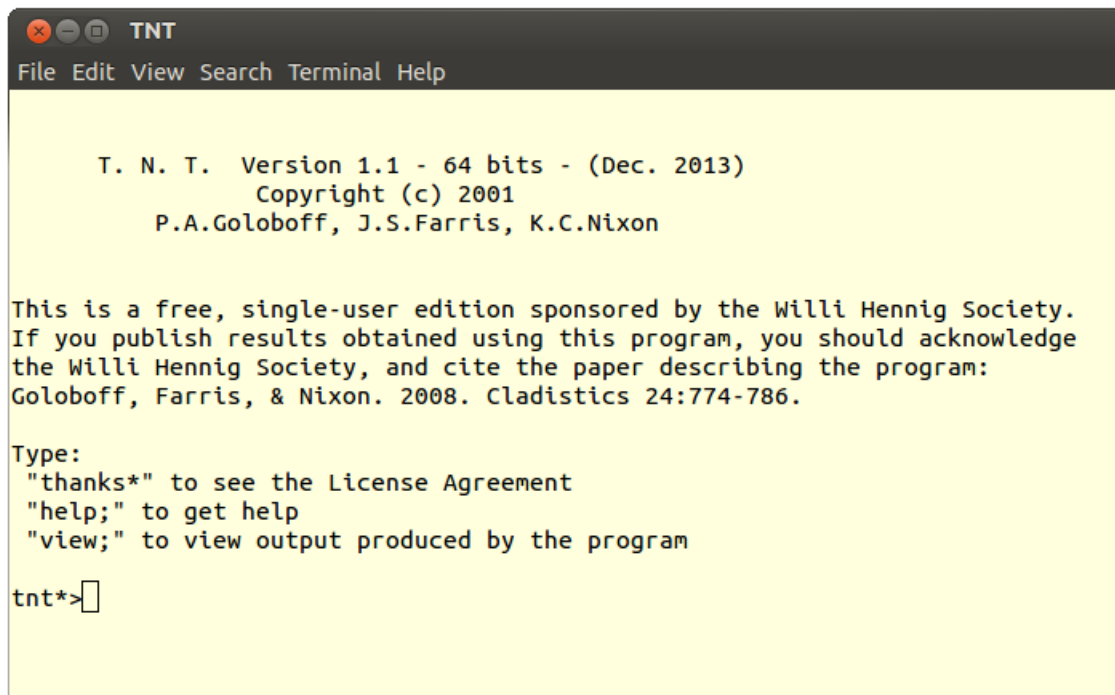


Figura 4.1: *Prompt de iniciação do TNT.*

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 algoritmo utilizado 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:

Exercício 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/;
```

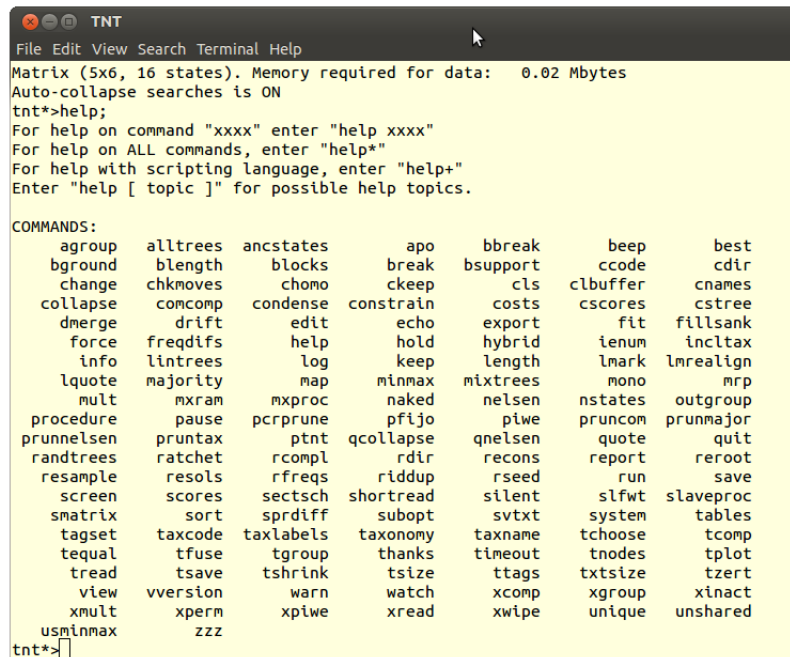
4.4 Obtendo ajuda no TNT

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.



```

TNT
File Edit View Search Terminal Help
Matrix (5x6, 16 states). Memory required for data: 0.02 Mbytes
Auto-collapse searches is ON
tnt*>help;
For help on command "xxxx" enter "help xxxx"
For help on ALL commands, enter "help*"
For help with scripting language, enter "help+"
Enter "help [ topic ]" for possible help topics.

COMMANDS:
agroup      alltrees  ancstates  apo         bbreak      beep        best
bgroup      blength   blocks     break       bsupport    ccode       cdir
change      chkmoves  chomo      ckeep       cls          clbuffer    cnames
collapse     comcomp   condense   constrain   costs       cscores     cstree
dmerge      drift     edit       echo        export      fit         fillsank
force       freqdifs  help       hold        hybrid      ienum       incltax
info        lintrees  log        keep        length      lmark       lrealign
lquote      majority  map        minmax      mixtrees    mono        mrp
mult        mxram     mxproc     naked       nelsen      nstates     outgroup
procedure    pause     pcrprune   pfljo       piwe        pruncom     prunmajor
prunnelsen  pruntax   ptnt       qcollapse   qnelsen     quote       quit
randtrees   ratchet   rcompl     rdir        recons      report      reroot
resample     resols    rfreqs     riddup      rseed       run         save
screen      scores    sectsch    shortread   silent      slfwt       slaveproc
smatrix     sort      sprdiff    subopt      svtxt       system      tables
tagset      taxcode   taxlabels  taxonomy    taxname     tchoose     tcomp
tequal      tfuse     tgroup     thanks      timeout     tnodes     tplot
tread       tsave     tshrink    tsize       ttags       txtsize     tzert
view        vversion  warn       watch       xcomp       xgroup     xinact
xmult       xperm     xpiwe      xread       xwipe       unique      unshared
usminmax
tnt*>

```

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 topologias. Os arquivos de topologia também obedecem uma lógica, muito similar àquela utilizada nos arquivos que contêm 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.

Exercício 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 exercício 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.

Exercício 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?

Exercício 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.

Exercício 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. Algoritmos 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 potencialmente 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 algoritmo 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.

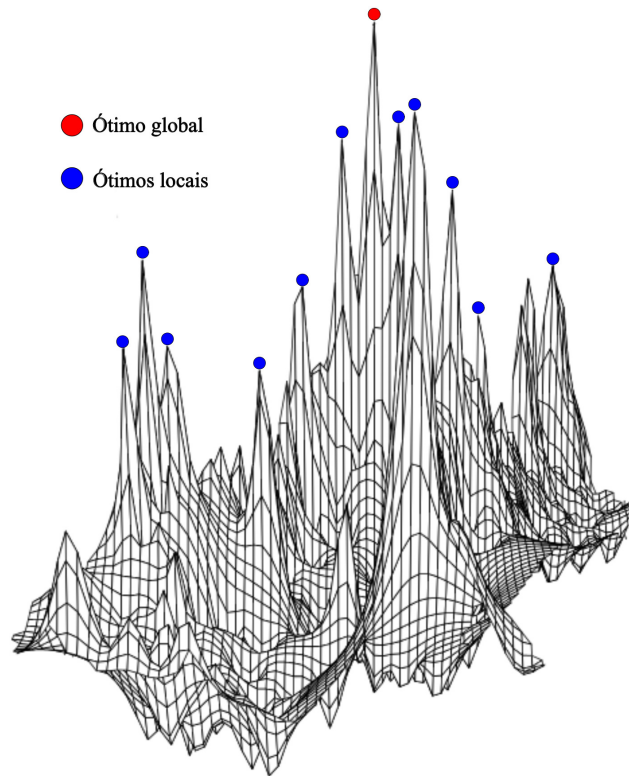


Figura 4.3: Ótimos locais e globais representados em uma superfície de relevo.

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 o 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 grande 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].

Exercício 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      Rearrang.
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, $10 \times 10 = 100$! 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.

Exercício 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 eficiência 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 (linha 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.

[illegible]

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.