

Tutorial 11

Homologia Dinâmica: *Iterative pass*, dados lacunares e partições em POY

BIZ0433 - INFERÊNCIA FILOGENÉTICA: FILOSOFIA, MÉTODO E APLICAÇÕES.

Conteúdo

Objetivo	184
11.1 Iterative Pass	185
11.1.1 Implementação	186
11.1.2 Tempo de execução	187
11.1.3 Redução de MPTs	188
11.1.4 Outras propriedades de IP	189
11.2 Dados lacunares (<i>missing data</i>)	190
11.2.1 Dados lacunares em POY	190
11.2.2 Dados lacunares em subpartições de POY	192
11.3 Assignment 6	193
11.4 Referências	194

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 baixa 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**.

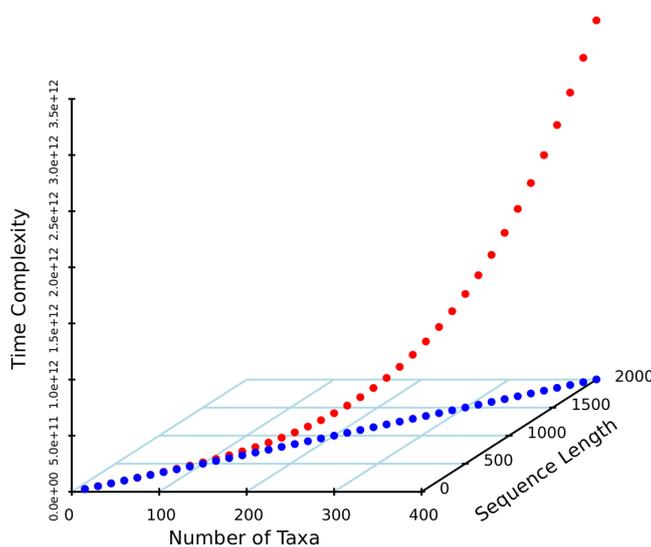


Figura 11.1: Complexidade computacional (medida em *time complexity* O) de **DO** ($O(km^2)$) – pontos azuis, onde k é o número de terminais e m o comprimento da sequência) em comparação com **IP** ($O(km^3)$) – pontos laranjas – em relação ao número de terminais e tamanho da sequência.

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 o 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).

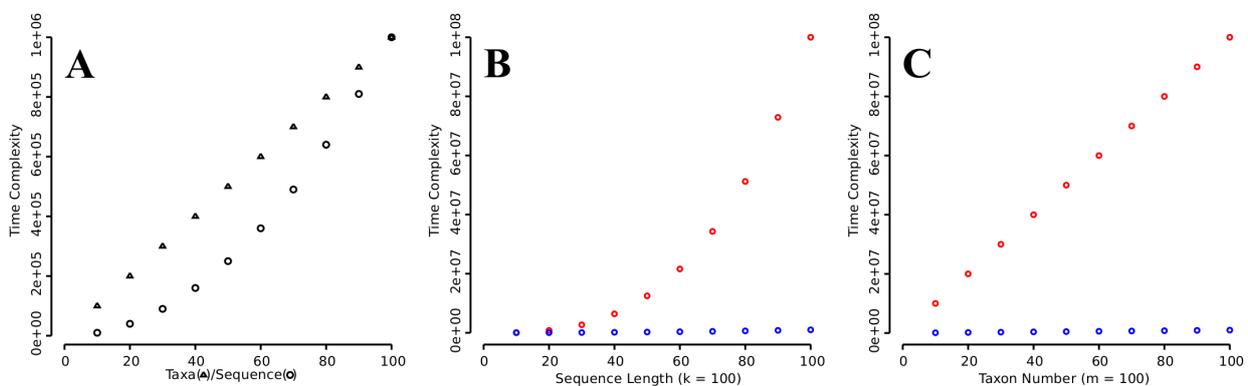


Figura 11.2: Número de terminais, comprimento da sequência e complexidade computacional de **DO** em comparação com **IP**. **A** – *Time complexity* O vs. números de terminais (para $k = 100$, triângulos) e comprimento de sequência (para $m = 100$, círculos). **B** – Comparação entre complexidade de **IP** (para $k = 100$, círculos vermelhos) e **DO** (círculos azuis) vs. comprimento da sequência. **C** – Comparação entre complexidade de **IP** (para $m = 100$, círculos vermelhos) e **DO** (círculos azuis) vs. números de terminais.

Exercício 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 contêm 6 terminais apenas e diferem quanto ao tamanho das sequências (*i.e.*, 30, 60 e 100, respectivamente). Neste exercício você deverá analisar esses dados utilizando **DO** e **IP** e o resultado de cada análise deverá ser colocado na Tabela 11.1, abaixo.

Listing 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.

Exercício 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 raiz 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].

Exercício 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 encontrada 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 inexistem 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:

Listing 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	<code>fasta2.fas</code>	<code>fasta1.fas</code>
<code>taxon1</code>	+	+
<code>taxon2</code>	-	+
<code>taxon3</code>	+	+
<code>taxon4</code>	+	+
<code>taxon5</code>	+	-

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(terminais, 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.

Exercício 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 sequência 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 *make_partitions.mp4* ilustra como isso deve ser feito. Este vídeo está no diretório associado a este tutorial.

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, incluindo 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 longo 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 Analysis. *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-Blackwell, 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 of 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 similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology* **48**: 443–453.
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.