Operations on nucleotide sequences to get some parameters of evolution

Ernesto Álvarez González*[1] and Ricardo Balam Narváez[2]

[1]Facultad de Ciencias Matemáticas, Universidad Complutense de Madrid, España
[2]Laboratorio de Biodiversidad, Escuela de Ciencias, Universidad Autónoma 'Benito Juárez' de Oaxaca, México

2020

Abstract

Phylogenetics deals with the task of reconstructing all the ancestral relations among a set of lineages. For any given tree explaining the evolution of those lineages, subject to kimura three parameter as their model of molecular evolution, 'Hadamard Conjugation' is a matrix equation that relates its tree weights (within a matrix Q) and its distribution of different substitution patterns (within a matrix P) on leafs. The construction of the matrix P is the main goal of this article. However, there are some challenges we have to face in the process. These challenges are overcome by implementing some work on the nucleotide sequences: delete sites where the multiple sequence alignment (obtained by other programs) produced errors; fill the gaps when they appear; compute the relative frequencies of all substitution patterns after selecting one of the lineages as a reference one. We conclude this work with a full developed example on three mexican, endemic spieces of plants for which we obtained the matrix P.

1 Introduction

Phylogenetic systematics comprise the principles and methods by which we reconstruct the evolutionary history of organisms [**?**]. Many techniques exist

*eralva01@ucm.es

for inferring phylogenetic relationships from molecular data. For any given tree that explains the relationship among a set of lineages, under Kimura three parameter model, 'Hadamard Conjugation' relates its weights to the distribution of substitution patterns on leafs. [**?**, **?**].

The following sections explain these concepts and include procedures for their obtantion.

2 Definitions

Let $a_1, a_2, \cdots a_n$ be sequences of characters A, C, G, T. We call an alignment the identification of characters belonging to each sequence per site.

table 1: Example of four sequences aligned with sixteen sites. Taken from [**?**]

site	1	2	3	4	5	6	7	8	9	10	11	12	$\cdots$
$\sigma_1 =$	C	C	A	T	C	A	A	A	C	G	T	G	
$\sigma_2 =$	A	C	A	G	C	A	A	T	G	T	T	A	
$\sigma_3 =$	C	C	A	T	T	G	A	A	G	A	T	G	
$\sigma_4 =$	A	C	A	G	T	A	G	T	G	T	T	A	

site	13	14	15	16
$\sigma_1 =$	T	G	A	C
$\sigma_2 =$	T	C	T	C
$\sigma_3 =$	C	G	T	T
$\sigma_4 =$	C	C	A	G

We can produce substitution patterns by indicating per site the substitutions from one row to the rest of rows:

table 2: Substitution patterns

site	1	2	3	4	5	6	7	8	9	10	11	12	$\cdots$
$\sigma_2 \to \sigma_1$	3	0	0	3	0	0	0	$T \to A = 2$	2	3	0	1	
$\sigma_2 \to \sigma_3$	3	0	0	3	1	1	0	$T \to A = 2$	0	2	0	1	
$\sigma_2 \to \sigma_4$	0	0	0	0	1	0	1	$T \to T = 0$	0	0	0	0	

continuación...

site	13	14	15	16
$\sigma_2 \to \sigma_1$	0	2	2	$C \to C = 0$
$\sigma_2 \to \sigma_3$	1	2	0	$C \to T = 1$
$\sigma_2 \to \sigma_4$	1	0	2	$C \to G = 2$

Table 2 uses the following notation: 0 stands for no change, 1 stands for transitions, 2 stands for type I transversions, 3 stands for type II transversions. See figure 1.

Hendy et al [**?**] use the following notation:

Select one reference lineage, say $i \in [n] = \{1, 2, 3, \cdots n\}$. Take $A, B \subset [n] \backslash \{i\}$. The order pair (A, B) is the substitution pattern such that

- $A \backslash B$: set of lineages obtained by transitions.
- $B \backslash A$: set of lineages obtained by type I transversions.
- $A \cap B$: set of lineages obtained by type II transversions.
- $[n] \backslash (A \cup B)$: set of lineages sharing the same character as row i.

Example **2.0.1** *From table 2, substitution patterns in sites eigth and sixteen are*

site 8 $(\emptyset, \{1, 3\})$*;*

site 16 $(\{3\}, \{4\})$.

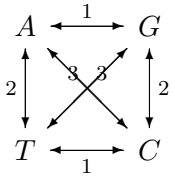


Figure 1: Substitutions.

3 Hadamard Conjugation

For any given tree that explains the relationship among a set of lineages, under Kimura three parameter model, 'Hadamard Conjugation' relates its weights to the distribution of substitution patterns on leafs in correspondence to the multiple alignment of the lineages.

The goal of this section is to show the elements involved in 'Hadamard Conjugation', putting special interest in their obtantion.

'Hadamard Conjugation' involves two espectral matrices: the 'Edge Length Spectrum', Q, and the 'Spectral Sequence Spectrum', P. The first one includes all the tree weights present in the model; the second one includes all the probabilities of different substitution patterns of the corresponding sample space.

Tree weights are important because they represent the expected number of substitutions along an edge of a given tree. In fact, there are three weights per edge ($q_e(1), q_e(2)$ and $q_e(3)$), each one of them counting the amount of substitutions (type 1, 2 or 3, respectively) along the edge over time.

3.1 Edge Lenght Spectrum

Before starting with the definition of the 'Edge Lenght Spectrum', it makes sense to introduce some notation.

Let $a_1, a_2, \cdots a_n$ be n lineages. Select the i-th lineage. Every order pair of disjoint subsets of $[n] = \{1, 2, \cdots n\}$ recovering the set $[n]$ is a bipartition for the set of lineages. It is a custom to identify those partitions with the subset that excludes the i-th lineage. Whe use the symbol $[n]_i$ to mean $[n]\backslash i$.

On the base of a phylogenetic tree as a model of evolution, a cut on one of its edges also produces a bipartition. That edge is identified by the corresponding bipartition.

Example **3.1.1** *Let's select the third leaf on tree of figure 2. Its bipartitions are identified by the subsets excluding the third leaf:*

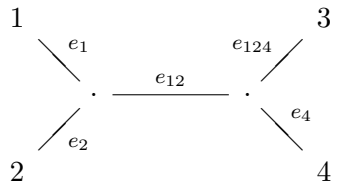


Figura 2

We reserve the symbol $e(T)$ for denoting the set of edges of the tree T

For any given phylogenetic tree T that explains all the ancestral relations of a set lineages, its 'Edge Lenght Spectrum', Q, is defined as follows:

$$q_{A,B} = \begin{cases} q_{e_A}(1) & si \quad e_A \in e(T) \quad y \quad si \quad B = \emptyset \\ q_{e_B}(2) & si \quad e_B \in e(T) \quad y \quad si \quad A = \emptyset \\ q_{e_A}(3) & si \quad e_A \in e(T) \quad y \quad si \quad A = B \\ -K_T & si \quad A = B = \emptyset \\ 0 & otro \quad caso, \end{cases}$$

where $q_{e_A}(j)$ (respectively $q_{e_B}(j)$) means the weight at the edge e_A (respectively e_B) corresponding to substitution of type j according to figure 1. We precise the term $K_T = \sum_{\Gamma \subset [n]_i}(q_\Gamma(1) + q_\Gamma(2) + q_\Gamma(3))$.

Rows and columns of matrix Q are indexed lexicografically according to the subsets of $[n]_i$.

The matrix Q is a square matrix of order $2^{n-1} \times 2^{n-1}$.

Example **3.1.2** *With respect to the third lineage of tree in figure 2, the sequence of subsets of* $\{1,2,4\}$*, ordered lexicografically, are:*

$$\begin{array}{rcl} 0 & \to & \emptyset \\ 1 & \to & \{1\} \\ 10 & \to & \{2\} \\ 11 & \to & \{1,2\} \\ 100 & \to & \{4\} \\ 101 & \to & \{1,4\} \\ 110 & \to & \{2,4\} \\ 111 & \to & \{1,2,4\} \end{array}$$

The third lineage is left out just to represent the bipartitions of the tree in figure 2.

With respect to this order for subsets of $\{1,2,4\}$*, the 'Edge Length Spectrum' Q for the tree in figure 2 is*

$$Q = \begin{array}{c} \\ \emptyset \\ \{1\} \\ \{2\} \\ \{1,2\} \\ \{4\} \\ \{1,4\} \\ \{2,4\} \\ \{1,2,4\} \end{array} \begin{array}{c} \begin{array}{cccccccc} \emptyset & \{1\} & \{2\} & \{1,2\} & \{4\} & \{1,4\} & \{2,4\} & \{1,2,4\} \end{array} \\ \begin{pmatrix} -K & q_1(\beta) & q_2(\beta) & q_{12}(\beta) & q_4(\beta) & 0 & 0 & q_{124}(\beta) \\ q_1(\alpha) & q_1(\gamma) & 0 & 0 & 0 & 0 & 0 & 0 \\ q_2(\alpha) & 0 & q_2(\gamma) & 0 & 0 & 0 & 0 & 0 \\ q_{12}(\alpha) & 0 & 0 & q_{12}(\gamma) & 0 & 0 & 0 & 0 \\ q_4(\alpha) & 0 & 0 & 0 & q_4(\gamma) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ q_{124}(\alpha) & 0 & 0 & 0 & 0 & 0 & 0 & q_{124}(\gamma) \end{pmatrix} \end{array}$$

Observe that the zeroes of main row, main column and main diagonal correspond to edges do not belong to the tree in figure 2.

3.2 Spectral Sequence Spectrum

For any set of n lineages, the 'Spectral Sequence Spectrum', P, is the square matrix of order $2^{n-1} \times 2^{n-1}$, whose entries are probabilities of different substitution patterns for the corresponding multiple alignment. Its rows and columns are indexed the same way as for the 'Edge Lenght Spectrum' Q.

Observe that for n lineages, after selecting one of them in order to consider substitutions recovering the rest of characters per site, there are at most $4^{n-1} = 2^{n-1} \times 2^{n-1}$ substitution patterns.

Next example shows how the 'Spectral Sequence Spectrum' P can be approximated on the base of relative sequences of different substitution patterns present on the corresponding multiple alignment.

Example **3.2.1** *Table 2 summarizes the substitution patterns present in the corresponding alignment of four lineages. These ones fit into the following square matrix P of order $2^{4-1} \times 2^{4-1} = 2^3 \times 2^3 = 8 \times 8$:*

$$P = \begin{array}{c} \\ \emptyset \\ \{1\} \\ \{3\} \\ \{1,3\} \\ \{4\} \\ \{1,4\} \\ \{3,4\} \\ \{1,3,4\} \end{array} \begin{array}{c} \begin{array}{cccccccc} \emptyset & \{1\} & \{3\} & \{1,3\} & \{4\} & \{1,4\} & \{3,4\} & \{1,3,4\} \end{array} \\ \begin{pmatrix} \frac{3}{16} & \frac{1}{16} & 0 & \frac{2}{16} & 0 & \frac{1}{16} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{16} & 0 & 0 & 0 & 0 \\ \frac{1}{16} & 0 & 0 & 0 & \frac{1}{16} & 0 & 0 & 0 \\ \frac{1}{16} & 0 & 0 & \frac{2}{16} & 0 & 0 & 0 & 0 \\ \frac{1}{16} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{2}{16} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Observe that lineage σ_3 takes σ_2's place in correspondence to the lexicographical order for the subsets of $\{1,2,3,4\}$:

$$\begin{array}{cccccccc} \emptyset & \{1\} & \{3\} & \{1,3\} & \{4\} & \{1,4\} & \{3,4\} & \{1,3,4\} \\ \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow \\ 0 & 1 & 10 & 11 & 100 & 101 & 110 & 111 \end{array}$$

For example, to locate the substitution pattern $(S_2, S_4) = (\{3\}, \{4\})$, we change $3 \to 2$ and $4 \to 3$: $(S_2, S_4) \to (2^{2-1} + 1, 2^{3-1} + 1) = (3, 5)$.

3.3 The main theorem

Theorem **3.3.1 (Hadamard Conjugation)** *Let Q be the 'Edge Length Spectrum' for any given phylogenetic tree with n leafs, under Kimura three para-*

mater as molecular evolution model. Let P be the corresponding 'Spectral Sequence Spectrum'. Then, it holds

$$H_n P H_n = \exp(H_n Q H_n), \tag{1}$$

where H_n is the $2^{n-1} \times 2^{n-1}$ Hadamard matrix obtained inductively as follows:

1.

$$H_1 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

2. $H_n = H_1 \otimes H_{n-1}$.

The term exp *stands for the exponential function acting termwise on $H_n Q H_n$.*

Symbol $\otimes$ in theorem 3.3.1 means Kronecker product.

4 Workflow

The obtention of the observed 'Spectral Sequence Spectrum' for any given set of lineages (in case when the alignment includes no gaps nor errors) is the result of some work on the corresponding sequences of nucleotides:

- classification of all subsitution patterns present in the alignment;
- location of all substitution patterns within teh Spectral Sequence Spectrum;
- counting the relative frequencies of all substitution patterns.

All this work can be done by following five procedures written by the main author to be run on the computer algebra system Maple: 'Patterns', 'SPL', 'SPLM', 'SPF' and 'SSS'. These procedures will be shown in sections 4.0.1-4.0.5, with a review on the ideas they are based on. The computation of the substitution patterns througout these procedures take as the reference lineage, from which substitutions recover the characters in the rest of rows (per site), the last one. This restriction is apparent: take the last lineage, say m, as the reference. Select another lineage, say $j \neq n$. Fix a site. Define $\chi(i)$ to be the character at row i in that site. The computation of all substitutions with respect to the j-th lineage is done by compounding the current substitutions by the substitution $\chi(n) \to \chi(j) = nj$ (see figure below).

$$
\begin{array}{rcl}
n1 & \to & nj \circ n1 = j1 \\
n2 & \to & nj \circ n2 = j2 \\
. & . & . \\
. & . & . \\
. & . & . \\
nj & \to & nj \circ nj = id \\
. & . & . \\
. & . & . \\
. & . & . \\
n \to n-1 & \to & nj \circ (n \to n-1) = j \to n-1 \\
n \to n = id & \to & nj \circ id = nj
\end{array}
$$

4.0.1 Patterns

Let $\sigma_1^n // \sigma_2^n // \sigma_3^n // \cdots \sigma_m^n$ be the alignment of m sequences with n nucleotides each one, and assume no errors and no gaps.

Procedure Patterns runs the alignment from left to right to classify different substitution patterns. These are saved within the matrix N_I.

In view of [BMS1], nucleotides and substitutions are represented by ordered pairs:

Nucleotides			**Sustitutions**		
Adenine	=	$(1,0)$	transitions	$\leftrightarrow$	$(0,1)$
Cytosine	=	$(0,1)$	type I transversions	$\leftrightarrow$	$(1,0)$
Guanine	=	$(1,1)$	type II transversions	$\leftrightarrow$	$(1,1)$
Thymine	=	$(0,0)$			

table 3

Termwise sum modulus 2 of pairs provide substitutions transforming the corresponding molecules. For example, $A = (1,0) \leftrightarrow G = (1,1)$ is a transition:
$(1,0) + (1,1) = (1 + 1 \bmod 2 = \mathbf{0},\ 0 + 1 \bmod 2 = \mathbf{1}) =$(0,1)$\leftrightarrow \alpha$.

```
> Patterns := proc(m, n, L)
global N_I:
local es_m, es_n, es_L, suma, entero, i, j, x, y:
entero := 0:
es_m := m:
es_n := n:
es_L := L:
N_I := randmatrix(es_m - 1, es_n):
```

```
for i from 1 to es_n by 1 do
for j from 0 to es_m - 2 by 1 do
if es_L[i + j*es_n] = "A" then
x := (1,0):
end if:
if es_L[i + j*es_n] = "C" then
x := (0,1):
end if:
if es_L[i + j*es_n] = "G" then
x := (1,1):
end if:
if es_L[i + j*es_n] = "T" then
x := (0,0):
end if:
if es_L[i + (es_m - 1)*es_n] = "A" then
y := (1,0):
end if:
if es_L[i + (es_m - 1)*es_n] = "C" then
y := (0,1):
end if:
if es_L[i + (es_m - 1)*es_n] = "G" then
y := (1,1):
end if:
if es_L[i + (es_m - 1)*es_n] = "T" then
y := (0,0):
end if:
suma := x + y:
if modp(suma[1],2) = 1 and modp(suma[2], 2) = 0 then
entero := 2:
end if:
if modp(suma[1],2) = 0 and modp(suma[2], 2) = 1 then
entero := 1:
end if:
if modp(suma[1],2) = 1 and modp(suma[2], 2) = 1 then
entero := 3:
end if:
if modp(suma[1],2) = 0 and modp(suma[2], 2) = 0 then
entero := 0:
end if:
N_I[j+1,i] := entero:
```

```
end do:
end do:
end proc:
```

4.0.2 SPL

This procedure returns the location of one substitution pattern within the matrix S (to be computed in procedure SSS) as an ordered pair. These ordered pair have as first entry, the row; as the second entry, the column.

According to [MS], entry (i, j) of S corresponds to the substitution pattern (A, B), $A, B \subseteq N' = N - \{n\}$ (where $N = \{1, 2, \cdots n\}$ is the set of species), in the following manner:

- $k \in A - B \Rightarrow 2^{k-1}$ appears in the binary expansion of i, but not in the binary expansion of j; they correspond to transitions.
- $k \in B - A \Rightarrow 2^{k-1}$ appears in the binary expansion of j, but not in the binary expansion of i; they correspond to type I transversions.
- $k \in A \cap B \Rightarrow 2^{k-1}$ appears in both the binary expansion of i and the binary expansion of j; they correspond to type II transversions.
- $k \notin A \cup B$ correspond to the species showing the same nucleotide as the reference one: no substitution.

We know that the number of subsets of N' is 2^{n-1}. We order these subsets, other than $\emptyset$, by identifying each one with the binary expansion they produce using the last rules. As we assign to the empty set $\emptyset \subset N'$ the first place to include those species showing no difference with the last one (no substitution), it means for the sequence of nonempty subsets of N', a shifting to the right (or equivalently, addition to the corresponding binary expansions by 1): $\underbrace{1}_{1st}, \underbrace{10}_{2nd}, \underbrace{11}_{third}, \cdots 11 \cdots 1 \rightarrow \underbrace{0}_{1st}, \underbrace{1}_{2nd}, \underbrace{10}_{third}, \cdots 11 \cdots 10, 11 \cdots 11.$

Example **4.0.1** *From data in table 2, we obtain the following substitution patterns for the corresponding sequence alignment $\sigma_1 - \sigma_2 - \sigma_3 - \sigma_4$ with reference lineage σ_4 at sites 1, 10 and 16:*

site	*1*	*10*	*16*
$\sigma_4 \rightarrow \sigma_1$	$A \rightarrow C$ =3	3	$C \rightarrow C$ =2
$\sigma_4 \rightarrow \sigma_2$	$A \rightarrow A$ =0	0	$C \rightarrow T$ =2
$\sigma_4 \rightarrow \sigma_3$	$A \rightarrow C$ =3	2	$C \rightarrow G$ =3
	$\updownarrow$	$\updownarrow$	$\updownarrow$
	$(\{1,3\},\{1,3\})$	$(\{1\},\{1,3\})$	$(\{3\},\{1,2,3\})$

The following chart includes the location of the corresponding substitution patterns previously registered, according to the coloring code in this section:

Substitution Pattern	*ith row*	*jth column*
$(\{1,3\},\{1,3\})$	$2^{1-1}+2^{3-1}+1=6$	$2^{1-1}+2^{3-1}+1=6$
$(\{1\},\{1,3\})$	$2^{1-1}+1=2$	$2^{1-1}+2^{3-1}+1=6$
$(\{3\},\{1,2,3\})$	$2^{3-1}+1=5$	$2^{1-1}+2^{2-1}+2^{3-1}+1=8$

```
> SPL := proc(r, R)
global pareja:
local es_r, esR, i, j, k, q:
es_r := r:
esR := R:
i := 0:
j := 0:
k := 0:
for q from 1 to es_r by 1 do
if R[q,1] = 1 then
i := i + 2^(q-1):
end if:
if R[q,1] = 2 then
j := j + 2^(q-1):
end if:
if R[q,1] = 3 then
k := k + 2^(q-1):
end if:
end do:
i := i + k + 1:
j := j + k + 1:
pareja := (i,j):
end proc:
```

4.0.3 SPLM

This procedure returns the location of every sustitution pattern in N_I within the matrix S into the list *parejas*. It calls the procedure SPL for each sustitution pattern in N_I.

```
> SPLM := proc(N_I)
```

```
global parejas:
local es_m, es_nc, esN_I, column, nuevo, es_nuevo, pares, i, j:
es_m := Size(N_I)[1]:
es_nc := Size(N_I)[2]:
esN_I := N_I:
parejas := []:
es_nuevo := columnas(es_m,es_nc):
for i from 1 to es_nc by 1 do
for j from 1 to es_m do
nuevo[i][j,1] := esN_I[j,i]:
end do:
end do:
for j from 1 to es_nc by 1 do
pares[j] := SPL(es_m,nuevo[j]):
end do:
parejas := Array(1..1,1..es_nc,(i,j)->pares[j]):
end proc:
```

4.0.4 SPF

This procedure counts the number of different sustitution patterns in N_I and adds these numbers into the matrix M. It calls SPLM to have in hand the corresponding substitution pattern locations of those elements in N_I.

```
> SPF := proc(N_I)
global parejas,M, TheElements:
local es_m, es_nc, rango, ElementFrecuencies,  i, j, cardinalidad,
repetidos, columnSET, NewColumnSET:
cardinalidad := 0:
es_nc := Size(N_I)[2]:
es_m := Size(N_I)[1]:
rango := 2^(es_m):
SPLM(N_I):
M := randmatrix(rango,rango):
for i from 1 to rango by 1 do
for j from 1 to rango by 1 do
M[i,j] := 0:
end do:
end do:
repetidos := {}:
```

```
columnSET := {}:
for i from 1 to es_nc by 1 do
columnSET := columnSET union {i}:
end do:
NewColumnSET : columnSET:
for i in columnSET do
ElementFrecuencies := 1:
for j from 1 while j <> i to es_nc by 1 do
if parejas[1,i] = parejas[1,j] then
ElementFrecuencies := ElementFrecuencies + 1:
end if:
end do:
M[parejas[1,i][1],parejas[1,i][2]] := ElementFrecuencies:
end do:
end proc:
```

4.0.5 SSS

This procedure divides each entry in M by n to get the relative frequencies of all substitution patterns in N_I. This information is kept in matrix $S = \frac{1}{n}M$.

```
> SSS := proc(M,n)
global S:
local es_n, rango, i, j:
es_n := n:
rango := Size(M)[1]:
S := M:
for i from 1 to rango by 1 do
for j from 1 to rango by 1 do
S[i,j] := S[i,j]/es_n:
end do:
end do:
end proc:
```

4.1 Technical difficulties

The assumptions for the sequences to compute the 'Spectral Sequence Spectrum' are:

- all the sequences are the same length;

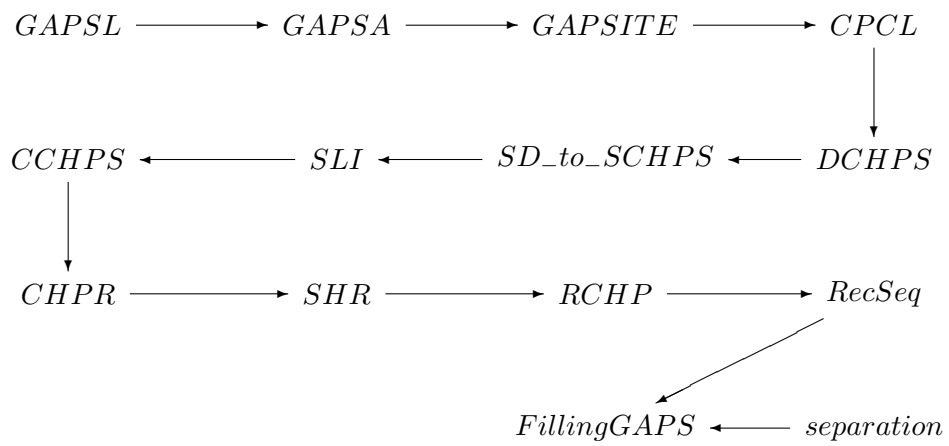
- they do not have any gaps.

Different lineages correspond to distinct sequences of nucleotides. Length is one of the characteristics they do not share, in general.

Gaps appear when some lineages are aligned: in order for some commun sites to fit, some gaps could appear.

The procedures of this section were built to overcome these limits. The criteria used to fill the gaps are not of biological nature, but of statistical importance: minimization of the variance.

These procedures relate according to the following dependence diagram:



4.1.1 GAPSL

Gaps are empty characters on a sequence. GAPSL locates those gaps and store the information within a list, $gaps_L$.

```
> GAPSL := proc(n,L)
global gaps_L:
local es_n, es_L1, i:
es_n := n:
es_L1 := L:
gaps_L := []:
for i from 1 to es_n by 1 do
if es_L1[i] = " " then
gaps_L := [op(gaps_L),i]:
```

```
end if:
end do:
end proc:
```

4.1.2 GAPSA

'GAPSA' calls 'GAPSL' for each n-sequence $\sigma_1, \sigma_2, \cdots \sigma_m$.

```
> GAPSA := proc(m, n, cadena2)
global Los_gaps, gaps_L:
local es_m, es_n, es_cadena, i:
Los_gaps := []:
es_m := m:
es_n := n:
es_cadena := cadena2:
for i from 1 to es_m by 1 do
GAPSL(es_n,es_cadena[i]):
Los_gaps := [op(Los_gaps),gaps_L]:
end do:
end proc:
```

4.1.3 GAPSITE

This procedure stores into the set 'siteset' the sites where the alignment shows at least one gap.

```
> GAPSITE := proc(m, n, Los_gaps)
global siteset:
local i, j, es_m, es_n, los_gaps:
es_m := m:
es_n := n:
los_gaps := Los_gaps:
siteset := {}:
for i from 1 to es_m by 1 do
for j from 1 to numelems(los_gaps[i]) by 1 do
siteset := siteset union {los_gaps[i][j]}:
end do:
end do:
end proc:
```

4.1.4 CPCL

CPCL classify all character patterns within two matrices, '*gap_chps*' and '*ngap_chps*'. The matrix *gap_chps* includes as m-columns all character patterns with gaps; the matrix *ngap_chps* includes as m-columns all character patterns with no gaps.

```
> CPCL := proc(m,n,L,siteset)
global gap_chps, ngap_chps, site_diference_set:
local es_m, es_n, es_L3, i, j,the_sites,es_siteset, contador:
es_m := m:
es_n := n:
es_L3 := L:
es_siteset := siteset:
the_sites := {}:
for i from 1 to es_n by 1 do
the_sites := the_sites union {i}:
end do:
site_diference_set := the_sites minus es_siteset:
gap_chps := randmatrix(es_m,numelems(es_siteset)):
contador := 0:
for i in es_siteset do
contador := contador + 1:
for j from 1 to es_m by 1 do
gap_chps[j,contador] := es_L3[(j-1)*es_n + i]:
end do:
end do:
ngap_chps := randmatrix(es_m,numelems(site_diference_set)):
contador := 0:
for i in site_diference_set do
contador := contador + 1:
for j from 1 to es_m by 1 do
ngap_chps[j,contador] := es_L3[(j-1)*es_n + i]:
end do:
end do:
end proc:
```

4.1.5 DCHPS

DCHPS computes the distance from two selected character patterns, one from *gap_chps* and the other from *ngap_chps*, by comparing the corres-

ponding row characters.

```
> DCHPS_v2 := proc(m,n,i,j,site_diference_set,gap_chps,ngap_chps)
global The_dist:
local es_m, es_gchps, es_ngchps, k, contador, es_g, es_ng, es_n,
es_site_diference_set, es_i, es_j:
es_m := m:
es_n := n:
es_site_diference_set := site_diference_set:
es_ng := numelems(site_diference_set):
es_g := es_n - es_ng:
es_gchps := gap_chps:
es_ngchps := ngap_chps:
contador := 0:
es_i := i:
es_j := j:
if es_i <= es_g then
if es_j <= es_ng then
for k from 1 to es_m by 1 do
if es_gchps[k,es_i] <> " " and es_gchps[k,es_i] <> es_ngchps[k,es_j]
then
contador := contador + 1:
end if:
end do:
end if:
end if:
The_dist := contador:
end proc:
```

4.1.6 SD_to_SCHP

SD_to_SCHP computes the distances from one selected character pattern in gap_chps to all character patterns in $ngap_chps$. It designs the minimum of these distances as the distance from the corresponding character (with gaps) to $ngap_chps$ (as a set of character patterns with no gaps).

```
> SD_to_SCHP := proc(m,n,s,site_diference_set,gap_chps,ngap_chps)
global the_distList:
local es_m, es_n, es_s, es_site_diference_set, es_gapchps,
es_ngapchps, distancia, i, es_ng:
```

```
es_m := m:
es_n := n:
es_s := s:
es_site_diference_set := site_diference_set:
es_ng := numelems(es_site_diference_set):
es_gapchps := gap_chps:
es_ngapchps := ngap_chps:
the_distList := []:
if es_s <= es_n - es_ng then
for i from 1 to es_ng by 1 do
distancia := DCHPS_v2(es_m,es_n,es_s,i,es_site_diference_set,
es_gapchps,es_ngapchps):
the_distList := [op(the_distList),distancia]:
end do:
end if:
end proc:
```

4.1.7 SLI

SLI produces the list "*lista_menores*" including the nearest character patterns from *ngap_chps* to the selected character pattern in *gap_chps*.

```
> SLI := proc(m,n,s,site_diference_set,gap_chps,ngap_chps)
global lista_menores, the_s, the_distList:
local PL, es_m, es_n, es_site_diference_set, es_ngapchps, i,
cardinality, el_entero1, es_siteset, es_gapchps:
es_m := m:
es_n := n:
the_s := s:
es_site_diference_set := site_diference_set:
es_gapchps := gap_chps:
es_ngapchps := ngap_chps:
SD_to_SCHP(es_m,es_n,the_s,es_site_diference_set,es_gapchps,
es_ngapchps):
cardinality := numelems(the_distList):
el_entero1 := 1:
lista_menores := []:
if cardinality > 1 then
for i from 2 to cardinality by 1 do
if Menor_entero(the_distList[el_entero1],the_distList[i]) =
```

```
the_distList[i] then
lista_menores := [op(lista_menores),i]:
el_entero1 := i:
end if:
end do:
end if:
for i from 1 to cardinality by 1 do
if the_distList[el_entero1] = the_distList[i] then
lista_menores := [op(lista_menores),i]:
end if:
end do:
end proc:
```

4.1.8 CCHPS

For any given pair of character patterns in *ngap_chps*, CCHPS returns 1 in case they are equal, otherwise it returns 0.

```
> CCHPS := proc(m,s1,s2,ngap_chps)
global el_contador:
local es_m, es_s1, es_s2,es_ngapchps, cuenta, k:
es_m := m:
es_s1 := s1:
es_s2 := s2:
es_ngapchps := ngap_chps:
el_contador := 0:
cuenta := 0:
for k from 1 to es_m by 1 do
if es_ngapchps[k,s1] = es_ngapchps[k,s2] then
cuenta := cuenta + 1:
end if:
end do:
if cuenta = es_m then
el_contador := 1:
end if:
end proc:
```

4.1.9 CHPR

Within the list 'listaRepeticiones', CHPR includes the absolute frequencies of nearest character patterns in $ngap_chps$ to the selected character pattern

of $gap_c hps$.

```
> CHPR := proc(m, lista_menores, ngap_chps)
global listaRepeticiones, el_contador:
local es_m, es_listaMenores, es_ngapchps, k, j, la_cuenta,
correccion, la_lista :
es_m := m:
es_listaMenores := lista_menores:
es_ngapchps := ngap_chps:
listaRepeticiones := []:
la_cuenta := 0:
correccion := 0:
if numelems(es_listaMenores) > 1 then
for k in es_listaMenores do
correccion := correccion + 1:
la_lista := subsop(correccion = NULL, es_listaMenores):
for j in la_lista do
CCHPS(es_m, k, j, es_ngapchps):
la_cuenta := la_cuenta + el_contador:
end do:
listaRepeticiones := [op(listaRepeticiones),la_cuenta]:
la_cuenta := 0:
end do:
else
listaRepeticiones := [op(listaRepeticiones),0]:
end if:
end proc:
```

4.1.10 SHR

For every character pattern in gap_chps, SHR selects from $ngap\_chps$ one character pattern that is nearest.

```
> SHR := proc(listaRepeticiones)
global el_entero:
local PL, i, cardinality, lista_mayores:
PL := listaRepeticiones:
cardinality := numelems(listaRepeticiones):
el_entero := 1:
if cardinality > 1 then
```

```
for i from 2 to cardinality by 1 do
if Mayor_entero(PL[el_entero],PL[i]) = PL[i] then
el_entero := i:
end if:
end do:
end if:
el_entero := lista_menores[el_entero]:
end proc:
```

4.1.11 RCHP

RCHP fills the gaps of the selected character pattern in *gap_chps* by copying the characters from the selected character pattern from *ngap_chps* that is nearest.

```
> RCHP := proc(m, n, es_s,el_entero,gap_chps,ngap_chps)
global es_GAPchps:
local es_m, es_n, the_sth, the_integer, i, j,es_ngapchps:
es_GAPchps := gap_chps:
es_ngapchps := ngap_chps:
es_m := m:
es_n := n:
the_sth := es_s:
the_integer := el_entero:
for i from 1 to es_m by 1 do
if es_GAPchps[i,the_sth] = " " then
es_GAPchps[i,the_sth] := es_ngapchps[i,the_integer]:
end if:
end do:
end proc:
```

4.1.12 RecSeq

RecSeq calls RCHP as much as there are sites with gaps.

```
> RecSeq := proc(m, n, es_GAPchps, siteset, L, Los_gaps)
global TheArray:
local i, j, es_m, es_n, ES_gapCHPS,es_siteset, contador,
arr1, arr2, lOS_GAPS,esL:
es_m := m:
```

```
es_n := n:
ES_gapCHPS := es_GAPchps:
es_siteset := siteset:
esL := L:
arr1 := Array(1..es_m, 1..es_n,(i,j)->0):
arr2 := Array(1..es_m, 1..es_n,(i,j)->0):
contador := 0:
lOS_GAPS := Los_gaps:
for j in es_siteset do
contador := contador + 1:
for i from 1 to es_m by 1 do
if j in lOS_GAPS[i] then
arr1[i,j] := ES_gapCHPS[i,contador]:
end if:
end do:
end do:
for i from 1 to es_m by 1 do
for j from 1 to es_n by 1 do
if L[(i-1)*es_n + j] <> " " then
arr2[i,j] := L[(i-1)*es_n + j]:
end if:
end do:
end do:
TheArray := arr1 + arr2:
end proc:
```

4.1.13 separation

This is a very technical procedure, because the whole process implemented to fill the gaps on sequences, forced the programmer to plug all of them together into just one larger sequence. This current procedure, separates the whole sequence into the right number of smaller sequences, as in the beginning.

```
> separation := proc(m,n,L)
global cadena1, cadena2:
local es_m, es_n, i, j, es_L2, sitios1, sitios2:
es_L2 := L:
es_m := m:
es_n := n:
```

```
cadena1 := []:
cadena2 := []:
sitios1 := []:
sitios2 := "":
for i from 1 to es_m by 1 do
for j from 1 to es_n by 1 do
sitios1 := [op(sitios1),es_L2[(i-1)*es_n + j]]:
end do:
cadena1 := [op(cadena1),sitios1]:
for j from 1 to es_n by 1 do
sitios2 := Join([sitios2,convert(sitios1[j],string)]):
sitios2 := Delete(sitios2,numelems(sitios2)-1..
numelems(sitios2)-1):
end do:
cadena2 := [op(cadena2),sitios2]:
sitios1 := []:
sitios2 := "":
end do:
end proc:
```

4.1.14 FillingGAPS

FillingGAPS is the organizer of the previous procedures to act correctly on the sequences given at first.

```
> FillinGAPSv3 := proc(m,n,L)
global TheArray, Los_gaps, cadena2, siteset, site_diference_set,
gap_chps, ngap_chps, lista_menores, listaRepeticiones, el_entero,
es_GAPchps:
local s, es_m, es_n, es_L:
es_L := L:
es_m := m:
es_n := n:
separation(es_m, es_n, es_L):
GAPSA(es_m,es_n,cadena2):
GAPSITE(es_m, es_n, Los_gaps):
CPCL(es_m,es_n,es_L,siteset):
for s from 1 to numelems(siteset) do
SLI(es_m,es_n,s,site_diference_set,gap_chps, ngap_chps):
CHPR(es_m,lista_menores,ngap_chps):
```

```
SHR(listaRepeticiones):
RCHP(es_m,es_n,s,el_entero,gap_chps,ngap_chps):
end do:
RecSeq(m, n, es_GAPchps, siteset, L, Los_gaps):
end proc:
```

4.2 Split of a tree

Let T be a tree. This section provides two procedures to build the observed spectral sequence spectrum $arrS_-$ of a selected subtree of T. These procedures assumes that the Spectral Sequence Spectrum of T is known. The procedures are $SplittingTheTree$ and $splittingPattern$.

The main idea to track information from the Spectral Sequence Spectrum P of T is selecting one leaf of the given subtree as the reference lineage from which all substitution patterns were computed to get P.

Example **4.2.1** *This example works on the alignment in table 1. The reference lineage from which substitutions were computed to get the matrix P in example 3.2.1 is the second one, σ_2.*

We will use the following notation to mean a tree with its reference lineage: Put into a set all lineages, except the reference one. For the lineages in table two, we have $N = \{1, 3, 4\}$. Let $N' = \{1, 4\}$ correspond to a subtree.

The observed spectral sequence spectrum $arrS_-$ for the 3-alignment $\sigma_1 - \sigma_2 - \sigma_4$ can be obtained from that in table 1 by viewing the sequence σ_3 as a free variable (because σ_3 is not an element of T').

Example of a $\sigma_1 - \sigma_2 - \sigma_4$ substitution pattern as the sum of $\sigma_1 - \sigma_2 - \sigma_3 - \sigma_4$ ones

site	*1*	*4*	*10*
$\sigma_2 \to \sigma_1 = 3$	*3*	*3*	*3*
$\sigma_2 \to \sigma_3$	*3*	*3*	*2*
$\sigma_2 \to \sigma_4 = 0$	*0*	*0*	*0*

Observe that the frequency of the $\sigma_1 - \sigma_2 - \sigma_4$ substitution pattern- $(\{1\}, \{1\})$, that is 3, is the sum of the frequencies of the $\sigma_1 - \sigma_2 - \sigma_3 - \sigma_4$ substitution patterns $(\{1\}, \{1\})$, $(\{1\}, \{1, 3\})$, $(\{1, 3\}, \{1\})$ and $(\{1, 3\}, \{1, 3\})$, that are $0, 1, 0, 2$, respectively: $3 = 0 + 1 + 0 + 2$. See the entries in S colored yellow, where we just have to pay attention to the numerators.

The observation is that when we focus on a restricted alignment (subtree T'), one substitution pattern can be observed some times along the no

restricted one. This increases the observation of the restricted substitution pattern.

This way, by hand, we obtain the matrix $arrS\_$:

$$arrS\_ = \begin{array}{c} \\ S_0 = \emptyset \\ S_1 = \{1\} \\ S_2 = \{4\} \\ S_3 = \{1,4\} \end{array} \begin{array}{c} \begin{array}{cccc} \emptyset & \{1\} & \{4\} & \{1,4\} \end{array} \\ \begin{pmatrix} \frac{4}{16} & \frac{3}{16} & \frac{1}{16} & \frac{1}{16} \\ \frac{1}{16} & \frac{3}{16} & 0 & 0 \\ \frac{3}{16} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Observe that entries in $arrS\_$ sum to one the same way entries in S sum to one.

4.2.1 splittingPattern

Every substitution pattern for a subtree T', splittingPattern sum over all substitution patterns for the whole tree T that contain that for T'.

```
> splittingPattern := proc(N, N_, i, j, ReferenceTaxon)
global MM, TheR, A, B, arr2, C, DD :
local n, II, J, q, accumulation, accumulation1, accumulation2,
l, contador2, l2, HelpSet, n2, k, contador1, arr, m:
m := numelems(N) + 1:
n := numelems(N_) + 1:
DeParejaAPatron(i,j,n):
C := {}:
DD := {}:
contador1 := 0:
contador2 := 0:
for k in A do
C := C union {N_[k]}:
end do:
for k in B do
DD := DD union {N_[k]}:
end do:
HelpSet := choose(N minus N_):
n2 := numelems(N minus N_):
arr := Array(1..2^(n2), 1..2^(n2), (II,J)->0):
arr2 := Array(1..2^(n2), 1..2^(n2), (II,J)->0):
contador1 := 0:
contador2 := 0:
```

```
for k in HelpSet do
contador1 := contador1 + 1:
for l in HelpSet do
contador2 := contador2 + 1:
arr[contador1,contador2] := [C union k, DD union l]:
accumulation1 := 0:
for l2 in arr[contador1,contador2][1] do
if l2 > ReferenceTaxon then
accumulation1 := accumulation1 + 2^(l2 - 2):
else
accumulation1 := accumulation1 + 2^(l2 - 1):
end if:
end do:
accumulation2 := 0:
for l2 in arr[contador1,contador2][2] do
if l2 > ReferenceTaxon then
accumulation2 := accumulation2 + 2^(l2 - 2):
else
accumulation2 := accumulation2 + 2^(l2 - 1):
end if:
end do:
arr2[contador1,contador2] := [accumulation1 + 1, accumulation2 + 1]:
end do:
contador2 := 0:
end do:
end proc:
```

4.2.2 SplittingTheTree

SplittingTheTree calls splittingPattern for every substitution pattern in correspondence to the selected subtree.

```
> SplittingTheTree := proc(N, N_, S, ReferenceTaxon)
global arrS_, arr2:
local II, J, n, n2, i, j, k, l:
n := numelems(N_):
n2 := numelems(N minus N_):
arrS_ := Array(1..2^n, 1..2^n, (II,J)->0):
for i from 1 to 2^n by 1 do
for j from 1 to 2^n by 1 do
```

```
splittingPattern(N, N_, i, j, ReferenceTaxon):
for k from 1 to 2^(n2) by 1 do
for l from 1 to 2^(n2) by 1 do
arrS_[i,j] := arrS_[i,j] + S[arr2[k,l][1],arr2[k,l][2]]:
end do:
end do:
end do:
end do:
end proc:
```